



中國計算機學會
CHINA COMPUTER FEDERATION

IOI2024中国国家集训队 论文集





中國計算機學會
CHINA COMPUTER FEDERATION

IOI2024中国国家集训队 论文集



2024 年国际信息学奥林匹克竞赛中国国家集训队论文集

目 录

关于积性函数求和问题的一些进展	周康阳 (1)
浅谈线性拟阵奇偶问题	郭羽冲 (11)
浅谈数据结构的合并与分裂	黄洛天 (24)
浅谈拉格朗日乘数及对偶在 OI 中的应用	施开成 (37)
范围修改查询问题相关算法及其应用	沈吉源 (54)
浅谈 OI 中动态规划问题的一些解题方法	曹 立 (69)
浅谈一些信息合并的处理方式	冯政玮 (85)
浅谈树拓扑序计数相关问题的一些方法	陈昕阳 (100)
论现代硬件上的常数优化	宋佳兴 (121)
浅谈一类容斥和反演方法的应用	董亦骐 (171)
使用生成函数刻画随机过程停时的一类技巧	叶 开 (182)
浅谈信息学竞赛中的线图问题	翟锦程 (196)
浅谈一类树上范围相关问题	朱屹帆 (206)
《世界沉睡童话》命题报告	叶李蹊 (217)
顶点带权的图匹配问题	柯绎思 (230)
再谈格路计数	吴 畅 (253)
连分数与欧几里得算法在 OI 中的应用	柯怿憬 (277)
浅谈复杂度及其在解决问题方面的应用	杨敏行 (299)
浅谈一种互质数对与最大公约数的维护算法	杜冠成 (315)
浅谈计算理论与 OI 中的难解问题	闫陈效 (321)
浅谈格算法在整系数多项式分解中的应用	谢梓涵 (336)
浅谈图的列表着色问题	钟自厚 (358)
由《蛭》出发举例分析观察法在信息学竞赛中的应用	李嘉佑 (367)
浅谈 q -analogue	李铭乐洋 (376)
浅谈最小比率环及其应用	魏佳泽 (400)

关于积性函数求和问题的一些进展

杭州学军中学教育集团文渊中学 周康阳

摘要

积性函数求和是一类重要的数论问题。关于该问题,在此前的研究旨在减少 $\text{soft-}\mathcal{O}(n^{2/3})$ 中的 $\log$ 因子数量¹。而我在思考后发现可以做到 $\mathcal{O}(n^{1/2} \text{poly log } n)$ 的时间复杂度²。本文将介绍这一筛法和其优化。

1 前置知识

1.1 定义与约定

定义 1.1. 数论函数是定义域为正整数集,陪域为复数域的函数。

对于一个数论函数 f , 我们约定 $S_f(n) = \sum_{i=1}^n f(i)$ 。

定义 1.2. 积性函数是指一个定义域为正整数 n 的数论函数 $f(n)$, 且满足如下性质: $f(1) = 1$, 且当 a 和 b 互质时, $f(ab) = f(a)f(b)$ 。

对于一个积性函数 f , 只要知道了对于所有 $p \in \mathbb{P}, k \in \mathbb{Z}^+$ 的 $f(p^k)$, 我们就可以知道整个 f 函数。

一些常见的积性函数在 $p \in \mathbb{P}, k \in \mathbb{Z}^+$ 的取值如下:

- ϵ 函数 (单位函数), 满足 $\epsilon(p^k) = 0$ 。
- μ 函数, 满足 $\mu(p^k) = -[k = 1]$ 。
- φ 函数, 满足 $\varphi(p^k) = (p-1)p^{k-1}$ 。
- I 函数, 满足 $I(p^k) = 1$ 。
- id_t 函数, 满足 $id_t(p^k) = p^{kt}$ 。

¹一个相关的研究是 <https://negiizhao.blog.uoj.ac/blog/7165>

²初稿于 2023-07-11 首发于 <https://www.cnblogs.com/zkyJuruo/p/17544928.html>

定义 1.3. 对于数论函数 f 和 g , 定义其 *Dirichlet* 卷积

$$(f * g)(x) = \sum_{d|x} f(d)g\left(\frac{x}{d}\right)$$

如果将 *Dirichlet* 卷积视为数论函数的乘法, 函数的直接加和为数论函数的加法。那么数论函数的加法和乘法是满足交换律、结合律和分配律的。

定理 1.1. 两个积性函数 f, g 的 *Dirichlet* 卷积 h 也是积性函数。

证明. 只要对于所有 $\gcd(x, y) = 1$ 证明 $h(xy) = h(x)h(y)$ 即可。

$$\begin{aligned} h(xy) &= \sum_{d|xy} f(d)g\left(\frac{xy}{d}\right) \\ &= \sum_{u|x, v|y} f(uv)g\left(\frac{xy}{uv}\right) \\ &= \sum_{u|x} f(u)g\left(\frac{x}{u}\right) \sum_{v|y} f(v)g\left(\frac{y}{v}\right) \\ &= h(x)h(y) \end{aligned}$$

□

1.2 Powerful Number

定义 1.4. 对于一个正整数 m , 如果对于其所有质因数 p , 均有 $p^2|m$, 则称其为 *Powerful Number*。

下文中, 我们用 PN 简称 *Powerful Number*。在公式中, 用 PN 代表所有 *Powerful Number* 构成的集合。

定理 1.2. 对于所有 PN , 我们都可以将其表示成 a^2b^3 的形式 (其中 a 和 b 都是正整数)。

证明. 设 $PN\ x = \prod_i p_i^{\alpha_i}$ 。我们只需要把每个 $p_i^{\alpha_i}$ 表示成 $a_i^2b_i^3$ 的形式, 然后取 $a = \prod a_i$ 和 $b = \prod b_i$ 即可。

如果 α_i 是奇数, 我们可以取 $a_i = p_i^{\frac{\alpha_i-3}{2}}, b_i = p_i$; 如果 α_i 是偶数, 我们则可以取 $a_i = p_i^{\frac{\alpha_i}{2}}, b_i = 1$ 。

□

定理 1.3. n 以内的 PN 个数为 $\mathcal{O}(\sqrt{n})$ 。

证明. 把所有 n 以内的 PN 表示成 a^2b^3 的形式。枚举 a , 我们就可以得到 PN 数不会超过 $\sum_{a=1}^{\sqrt{n}} \left(\frac{n}{a^2}\right)^{1/3}$ 。这个可以估计为

$$\begin{aligned}\int_1^{\sqrt{n}} \left(\frac{n}{x^2}\right)^{1/3} dx &= n^{1/3} \int_1^{\sqrt{n}} x^{-2/3} dx \\ &= n^{1/3} (3n^{1/6} - 3)\end{aligned}$$

即 $\mathcal{O}(\sqrt{n})$ 级别。

□

线性筛出 $\sqrt{n}$ 以内的所有质数，然后在质数上 DFS 就能找出所有不超过 n 的 PN。时间复杂度和 PN 的个数相同，即 $\mathcal{O}(\sqrt{n})$ 。

例题 1.1. (UOJ Long Round 1 校验码的 $n=1$ 部分) 给定 c 。积性函数 $q(x)$ 满足 $q(p^k) = p^{\lfloor \frac{k}{2} \rfloor}$ 。求 $\sum_{i=1}^m q(i)$ ，答案对 2^{32} 取模。 $n \leq 1.2 \times 10^{11}$ 。

设计积性函数 $f(x)$ 满足 $f(p) = q(p)$ ，并构造积性函数 g 满足 $g * f = q$ 。

对于素数 p ，有 $g(p)f(1) + g(1)f(p) = q(p)$ ，解得 $g(p) = 0$ 。因此 g 函数只在 PN 处有值！

$$\begin{aligned}\sum_{i=1}^m q(i) &= \sum_{i=1}^m \sum_{j|i} g(j)f(k) \\ &= \sum_{j \in \text{PN}, j \leq m} g(j) \sum_{k=1}^{\lfloor \frac{m}{j} \rfloor} f(k) \\ &= \sum_{j \in \text{PN}, j \leq m} g(j) \lfloor \frac{m}{j} \rfloor\end{aligned}$$

提前对于每个 p^k 预处理出 $g(p^k)$ （我们有 $h(p^k) = \sum_i f(p^i)g(p^{k-i})$ ，可以按 k 从小到大的顺序解出 $g(p^k)$ ），并在 DFS 找 PN 的同时维护 g 函数值。这样就在 $\mathcal{O}(\sqrt{m})$ 的复杂度内解决了这个问题。

1.3 杜教筛

对于一些特殊的积性函数（如 μ 和 φ ），我们可以通过构造 Dirichlet 卷积在 $\mathcal{O}(n^{\frac{2}{3}})$ 的时间复杂度计算其前缀和。这种算法在国内信息学竞赛界被称为“杜教筛”。

对于 μ 函数，我们有 $\mu * I = \epsilon$ 。

不妨考虑更一般的问题。如果有数论函数 A, B, C 满足 $A(1) = B(1) = C(1) = 1$ 且 $A * B = C$ ，并且我们能快求出 S_B 和 S_C 的点值，我们该如何求出 $S_A(n)$ 呢？

注意到

$$\begin{aligned}\sum_{i=1}^n C(i) &= \sum_{i=1}^n \sum_{j|i} A(j) B(k) \\ &= \sum_{k=1}^n B(k) \sum_{j=1}^{\lfloor \frac{n}{k} \rfloor} A(j) \\ &= S_A(n) + \sum_{k=2}^n B(k) S_A(\lfloor \frac{n}{k} \rfloor)\end{aligned}$$

因此 $S_A(n) = S_C(n) - \sum_{k=2}^n B(k) S_A(\lfloor \frac{n}{k} \rfloor)$ ，可以递归求 $S_A(n)$ 。

注意到 $\lfloor \frac{\lfloor \frac{n}{b} \rfloor}{a} \rfloor = \lfloor \frac{n}{ab} \rfloor$ ，在递归过程中，需要被计算的实际上只有 $S_A(\lfloor \frac{n}{k} \rfloor)$ (k 是正整数)。

定理 1.4. 对于正整数 n ，集合 $\{\lfloor \frac{n}{i} \rfloor | i \in \mathbb{Z}^+\}$ 的大小是 $\mathcal{O}(\sqrt{n})$ 的。

证明. 对于 $k \leq \sqrt{n}$ ，只有 $\sqrt{n}$ 个 $\lfloor \frac{n}{k} \rfloor$ ，显然只有 $\mathcal{O}(\sqrt{n})$ 种；

对于 $k > \sqrt{n}$ ， $\lfloor \frac{n}{k} \rfloor \leq \frac{n}{\sqrt{n}} = \sqrt{n}$ ，因此也只有 $\mathcal{O}(\sqrt{n})$ 种。 $\square$

因此，我们只需要计算 S_A 的 $\mathcal{O}(\sqrt{n})$ 个点值。

在计算单个 $S_A(m)$ 时，将连续的相同的 $\lfloor \frac{m}{k} \rfloor$ 一起计算（即整除分块），就能在 $\mathcal{O}(\sqrt{m})$ 的复杂度内递归到子问题了。

对于 $m \leq n^{\frac{2}{3}}$ 的 $S_A(m)$ ，我们通过提前预处理出 A 的前 $n^{\frac{2}{3}}$ 个点值来解决。对于 $m > n^{\frac{2}{3}}$ ，使用上面的递归式计算，这部分的时间复杂度 $\mathcal{O}(\sum_{i=1}^{n^{\frac{1}{3}}} \sqrt{\frac{n}{i}})$ 。这个可以估算为 $\mathcal{O}(\int_1^{n^{\frac{1}{3}}} \sqrt{\frac{n}{i}} di) = \mathcal{O}(n^{\frac{2}{3}})$ 。

总复杂度即为 $\mathcal{O}(n^{\frac{2}{3}})$ 加上预处理出 $\mathcal{O}(n^{\frac{2}{3}})$ 预处理出 A 的前 $n^{\frac{2}{3}}$ 个点值的复杂度（对于 μ 和 φ 来说都是可以 $\mathcal{O}(n^{\frac{2}{3}})$ 预处理的）。

定义 1.5. 定义数论函数 f 关于 n 的块筛为 $S_f(n)$ 在 $\{\lfloor \frac{n}{i} \rfloor | i \in \mathbb{Z}^+\}$ 的点值。

在上述过程当中，我们只用了 B 和 C 关于 n 的块筛，就求出了 A 关于 n 的块筛。

当然，只知道 A 和 B 关于 n 的块筛，我们也能求出 C 关于 n 的块筛。

2 块筛卷积

对于数论函数 A, B, C 满足 $A(1) = B(1) = C(1) = 1$ 且 $A * B = C$ ，上文的杜教筛根据 B 和 C 关于 n 的块筛解出了 A 关于 n 的块筛。

在这里，我们先考虑一个更加简单的问题：已知数论函数 A 和 B 关于 n 的块筛，要求求出满足 $C = A * B$ 关于 n 的块筛。我们将 C 称为 A 和 B 的块筛卷积。块筛上的运算仍然具有交换律，结合律和分配律。

这个问题实际上是可以 $\mathcal{O}(n^{\frac{1}{2}} \text{poly} \log n)$ 解决的。

下文中，我们默认块筛是关于 n 的块筛。

我们要解决的是 $C(z) = \sum_{xy=z} A(x)B(y)$ 在所有 $\lfloor \frac{n}{t} \rfloor$ 上的前缀和。

首先考虑 $x > \sqrt{n}$ 的情况 ($y > \sqrt{n}$ 是对称的)。如果 $xy \leq \lfloor \frac{n}{t} \rfloor$, 那么 $x \leq \frac{n}{ty}$ 。所以枚举 t 和 y , 有贡献的 x 是一段前缀。这部分的复杂度是 $\mathcal{O}(\sqrt{n} \log n)$ 。

这样我们就只需要解决 $x \leq \sqrt{n}$ 且 $y \leq \sqrt{n}$ 的情况了。

$xy \leq \frac{n}{t}$ 等价于 $\ln(x) + \ln(y) \leq \ln(\frac{n}{t})$ 。

我们做一个估计: 选取一个块长 $S(3 \leq S \leq n)$, 我们认为 $\ln(x) + \ln(y) \leq \ln(\frac{n}{t})$ 当且仅当 $\lceil S \ln(x) \rceil + \lceil S \ln(y) \rceil \leq S \ln(\frac{n}{t})$ 。

这个估计可以用多项式乘法完成。设计多项式 $F(z), G(z)$, 其中 $[z^k]F(z)$ 为满足 $\lceil S \ln(x) \rceil = k$ 的 $A(x)$ 的和, $[z^k]G(z)$ 为满足 $\lceil S \ln(y) \rceil = k$ 的 $B(y)$ 的和。计算多项式 $H(z) = F(z)G(z)$, 则 $[z^k]H(z)$ 就是 $\lceil S \ln(x) \rceil + \lceil S \ln(y) \rceil = k$ 的 $A(x)B(y)$ 和。使用一次长度为 $S \ln(n)$ 的卷积即可。

但是这样做显然是不对的。观察我们在什么情况下会估计错: 只有 $\ln(x) + \ln(y) \leq \ln(\frac{n}{t})$ 且 $\lceil S \ln(x) \rceil + \lceil S \ln(y) \rceil > S \ln(\frac{n}{t})$ 时才会估错。此时有 $S \ln(x) + S \ln(y) \in (S \ln(\frac{n}{t}) - 2, S \ln(\frac{n}{t}))$ 。

注意到这个时候 x, y, t 必须满足 $\ln(x) + \ln(y) + \ln(t) \in (\ln(n) - \frac{2}{S}, \ln(n))$, 即 $xyt \in (ne^{-\frac{2}{S}}, n]$ 。其中 $e^{-\frac{2}{S}}$ 是 $1 - \mathcal{O}(\frac{1}{S})$ 级别的。

因此, xyt 是在一个长度为 $\mathcal{O}(\frac{n}{S})$ 的区间内的!

不妨假设这个区间是 $[n - L, n]$ 。我们使用区间筛得到这个区间内所有数的质因数分解。更具体地说, 先筛出所有不超过 $\sqrt{n}$ 的小质数, 对于每个质数, 我们都可以求得区间 $[n - L, n]$ 中被它整除的数。这样我们就筛出了 $[n - L, n]$ 的所有数的 $\leq \sqrt{n}$ 的质因子。而 $> \sqrt{n}$ 的质因子最多只有一个, 所以只要看这个数除掉所有小质数后的值就行了。

得到质因数分解后, 我们就可以通过在质因子上 DFS 快速找出所有可能的 (x, y, t) 了, 对于一个 (x, y, t) 可以 $\mathcal{O}(1)$ 检查它是否被正确地估计了。对于一个 $v \in [n - L, n]$, 它贡献时间复杂度是 $d_3(v)$ (其中 $d_3(v) = \sum_{xyz=v} 1$)。

而根据解析数论中的结论 [1], $\sum_{i \leq n} d_3(i) = nP_3(\ln n) + \mathcal{O}(n^{43/96+\epsilon})$ 。所以 $\sum_{i=n-L}^n d_3(i) = LP_3(\ln n) + \mathcal{O}(n^{43/96+\epsilon})$, 其中 P_3 是二次多项式。算法的输出已经有 $\mathcal{O}(\sqrt{n})$, 比 $\mathcal{O}(n^{43/96+\epsilon})$ 大, 所以可以忽略这部分的复杂度。剩下部分的复杂度就是 $\mathcal{O}(L \log^2 n)$, 即 $\mathcal{O}(\frac{n}{S} \log^2 n)$ 。

这么做的总复杂度是 $\mathcal{O}(S \log^2 n + \frac{n}{S} \log^2 n)$, 取 $S = \sqrt{n}$ 即可得到 $\mathcal{O}(\sqrt{n} \log^2 n)$ 的时间复杂度。

3 积性函数求和

对于一个满足一些特殊性质 (比如满足 $f(p^k) = p^k(p^k - 1)$ 的积性函数) 的积性函数 f , 该如何求 f 的块筛呢?

首先, $f(p^k) (k \geq 2)$ 的值并不重要。如果另一个积性函数 g 满足 $f(p) = g(p)$, 且 g 的前缀和容易计算, 那么我们只需要设计满足 $h * g = f$ 的积性函数 h , 根据前文的结论, h

只在 PN 处有值且这些位置上的值可以 $\mathcal{O}(\sqrt{n})$ 算出。因此容易 $\mathcal{O}(\sqrt{n})$ 算出 h 的块筛, 然后通过 h 和 g 的块筛卷积算出 f 的块筛。

我们先设计数论函数 q 满足 q 只在质数处的值非零, 且 $q(p) = f(p)$ 。

定义一个数论函数 q ($q(1) = 0$) 的 $\exp(q)$ 为 $\sum_{i=0}^{\infty} \frac{q^i}{i!}$, 其中卷积是块筛卷积。观察 $d = \exp(q)$ 。对于一个数 $x = \prod_{i=1}^k p_i^{\alpha_i}$, $d(x)$ 只有在 $q^{\sum_{i=1}^k \alpha_i}$ 项上有值, 且值为 $\frac{\prod_{i=1}^k q(p_i)^{\alpha_i}}{(\sum_{i=1}^k \alpha_i)!} \binom{\sum_{i=1}^k \alpha_i}{\alpha_1, \alpha_2, \dots, \alpha_k} = \frac{\prod_{i=1}^k q(p_i)^{\alpha_i}}{\prod_{i=1}^k \alpha_i!}$ 。

因此, d 函数是一个积性函数, 且 $d(p^k) = \frac{f(p)^k}{k!}$, 满足 $d(p) = f(p)$ 。这样, 如果我们能求出 q 的块筛, 我们就可以通过 $\log n$ 次 (在 $i > \log n$ 时 q 函数的前 n 项没有值) 块筛卷积得到 d 函数的块筛, 从而得到 f 的块筛。

怎么得到 q 的块筛呢? 考虑一个类似 Min_25 筛的想法。将 q 函数拆成若干个 q_i 的带权和, 并通过这些 q_i 的块筛来算出 q 的块筛。比如 $q(p) = p(p-1)$, 我们就可以拆成 $q_1(p) = p^2$, $q_2(p) = p$, 然后就有 $q = q_1 - q_2$ 。

当 $q(p)$ 是关于 p 的常数次多项式时, 我们都可以按照这种方法拆函数。

如何求出 q_i 的块筛呢? 这里我们反过来。以求 $q_1(p) = p^2$ 为例。对于一个满足 $d(p^k) = q_1(p)^k$ 的积性函数 d , 它满足 $d(x) = x^2$, 块筛很好算; 而对于满足 $d(p^k) = \frac{q_1(p)^k}{k!}$ 的积性函数 d , 由于它和上个函数在 p 处的点值相同, 所以也可以用 PN 在一次块筛卷积的复杂度内相互转化, 所以它的块筛也是可以算出的。

而满足 $d(p^k) = \frac{q_1(p)^k}{k!}$ 的函数 d , 正是 $\exp(q_1)$ 。这启发我们用一个类似 $\ln$ 的思路来从 d 反推出 q_1 。

定义数论函数 d ($d(1) = 1$) 的 $\ln(d)$ 为 $\sum_{i=1}^{\infty} \frac{(-1)^{i-1}(d-1)^i}{i}$ 。

我们有 $q_1 = \ln(d)$, 证明如下。

证明. 我们只要分别说明 $\exp(\ln(d)) = d$ 和满足 $\exp(q) = d$ 的 q 是唯一的即可。

首先说明 $\exp(\ln(d)) = d$ 。

$$\exp(\ln(d)) = \sum_{j=0}^{\infty} \frac{(\sum_{i=1}^{\infty} \frac{(-1)^{i-1}(d-1)^i}{i})^j}{j!}$$

使用块筛的分配律和结合律将其展开, 我们将得到一个 $\sum_i a_i d^i$ 形式的式子。不妨用多项式 $A(z)$ 来刻画它, 其中 $[z^k]A(z) = a_k$ 。而 $\exp$ 和 $\ln$ 在块筛卷积中的定义和多项式相同 (即 $\exp(z) = \sum_{i=0}^{\infty} \frac{z^i}{i!}$, $\ln(1+z) = \sum_{i=1}^{\infty} \frac{(-1)^{i-1}z^i}{i}$), $A(z) = \exp(\ln(z)) = z$, 上面的 $a_i = [i=1]$, 因此 $\exp(\ln(d)) = d$ 。

而如果满足 $\exp(q) = d$, 那么 $\epsilon + q + \sum_{i \geq 2} \frac{q^i}{i!} = d$ 。我们可以从小到大依次确定 $q(i)$ 的值: 设函数 $p = \sum_{i \geq 2} \frac{q^i}{i!}$, 则 $p(i)$ 只和 $q(j)$ ($j < i$) 有关, 所以可以用 $q(i) = d(i) - [i=1] - p(i)$ 解出 $q(i)$ 。因此 q 函数是可以被唯一确定的。

□

上述的 $\ln$ 中也只需要枚举 $i \leq \log n$, 所以也只需要 $\log n$ 次块筛卷积。



因此，该算法的时间复杂度是 $\mathcal{O}(\sqrt{n} \log^3 n)$ 。

4 优化

虽然我们已经得到了一个 $\mathcal{O}(n^{1/2} \text{poly} \log n)$ 的算法，但是它太慢了，还不能在 OI 中得到实际应用。能不能优化一下呢？

下面提供一些简单的优化方向，可以将算法的时间复杂度优化至 $\mathcal{O}(\frac{\sqrt{n} \log^2 n}{\sqrt{\log \log n}})$ 。

4.1 优化 exp

我们计算 $\exp(f)$ (f 满足只在质数处有值) 的时候会做 $\log n$ 次块筛卷积，比较浪费，考虑如何优化。

首先，我们可以把 f 拆成下标 $\leq \sqrt{n}$ 的部分 f_0 和下标 $> \sqrt{n}$ 的部分 f_1 。

然后计算 $\exp(f) = \exp(f_0 + f_1) = \exp(f_0) \exp(f_1)$ 。

$\exp(f_1) = \varepsilon + f_1$ ，是容易计算的；而 $\exp(f_0)$ 相对难计算。

我们计算 $\exp(f_0)$ 时，对于最终结果中前 $\sqrt{n}$ 项，可以直接计算出其函数值；难点在于计算 $> \sqrt{n}$ 的部分。

考虑直接按照前文的取整加修正的方法来计算。选取整数 S ，并把 $f_{0,i}$ 加在 $v_{\lfloor S \ln i \rfloor}$ 上。然后，我们直接对 v 做多项式 $\exp$ 。

考虑这样做什么时候会计算错误：设 $m = \prod_{i=1}^k p_i^{\alpha_i}$ 。如果 m 的位置被估计错了，那么设 $t = \lfloor \frac{n}{m} \rfloor$ ，就有 $\sum_{i=1}^k \alpha_i \lceil S \ln(p_i) \rceil \in (S \ln(\frac{n}{t}), S \ln(\frac{n}{t}) + \sum_{i=1}^k \alpha_i)$ 。

那么就有 $\sum_{i=1}^k \alpha_i \ln(p_i) + \ln t - \ln n \in [-\frac{\sum_{i=1}^k \alpha_i}{S}, 0]$ ， $\ln(mt) \in [\ln n - \frac{\sum_{i=1}^k \alpha_i}{S}, \ln n]$ 。

这是一段长度为 $\mathcal{O}(\frac{n \sum_{i=1}^k \alpha_i}{S})$ 的区间，即不会超过 $\mathcal{O}(\frac{n \log n}{S})$ 。在这一段区间内的 mt 才会被估计错。

用区间筛来筛出这个区间内所有 $\leq \sqrt{n}$ 的质因子，筛出质因子后即可通过在质因子上 DFS 快速找出 (m, t) 了，在 DFS 的时候还可以同时找出 m 处的函数值。DFS 的复杂度和数对数量同阶。

数对数量不会找过 $\mathcal{O}(\sum_{m=1}^{\sqrt{n}} \frac{n \log n}{Sm})$ ，即 $\mathcal{O}(\frac{n \log^2 n}{S})$ 。前面多项式部分的复杂度是 $\mathcal{O}(S \log^2 n)$ 。

取 $S = \sqrt{n}$ 即可得到 $\mathcal{O}(\sqrt{n} \log^2 n)$ 的时间复杂度。

我们还可以取一个阈值 L ，将 $\leq L$ 的质因子特殊处理。在计算 $\exp(f_0)$ 的时候不考虑 $\leq L$ 的因子（即把它们在 f_0 对应的位置清零）。在算完之后，我们再将这些质因子一个一个加入。

加入质因子 p 的时候，假设之前求出的答案为 d ，那么新的 $d'(n) = \sum_{k=0}^{\log_p n} f(p^k) d(\lfloor \frac{n}{p^k} \rfloor)$ ，其需要的复杂度代价是 $\sqrt{n} \log_p n$ 。

因此加入的总代价就是 $\sum_{p \leq L, p \in \mathbb{P}} \frac{\sqrt{n} \log n}{\log p} = \mathcal{O}(\frac{L \sqrt{n} \log n}{\log L})$ 。

在去掉 $\leq L$ 的质因子之后, 上文的 m 就只有 $> L$ 的质因子了, 因此 m 的质因数分解 $\prod_{i=1}^k p_i^{\alpha_i}$ 满足 $\sum_{i=1}^k \alpha_i \leq \log_L n$, 因此会估算错的区间长度也减少到了 $\frac{n \log n}{S \log L}$ 。

这样做, 计算 $\exp(f_0)$ 的复杂度就变成了 $\frac{n \log n \log_L n}{S} + S \log^2 n$, S 取 $\frac{\sqrt{n}}{\sqrt{\log L}}$ 得时间复杂度 $\mathcal{O}(\frac{n \log^2 n}{\sqrt{\log L}})$ 。

取 $L = \log n$, 时间复杂度 $\mathcal{O}(\frac{\sqrt{n} \log^2 n}{\sqrt{\log \log n}})$ 。

时间复杂度 $\mathcal{O}(\frac{\sqrt{n} \log^2 n}{\sqrt{\log \log n}})$ 。

4.2 优化 $\ln$

我们计算 $\ln$ 时, 需要将一个满足 $f(p^k) = \frac{f(p)^k}{k!}$ 的积性函数转化成一个满足只在 $g(p)$ 处有值且 $g(p) = f(p)$ 的数论函数 g 。

可以直接根据上述性质得到 $g = \ln(f)$ 的前 $\sqrt{n}$ 项。

不妨只保留 g 的 $\sqrt{n}$ 项的数论函数是 g_0 , 并令 $g_1 = g - g_0$ 。那么 $\exp(g_0 + g_1) = \exp(g_0) \exp(g_1)$ 。

注意到两个 g_1 做卷积, 得到的最低项一定 $> n$, 块筛上的值均为 0, 可以不计算。

于是 $f = \exp(g_0) + \exp(g_0)g_1$ 。算出 $h = \exp(g_0)$ 后, 解 $f - h = hg_1$ 即可。

现在我们需要解决块筛除法问题。对于数论函数 A, B, C 满足 $B(1) = 1$, 且 $A * B = C$, 如何通过 B, C 的块筛得到 A 的块筛呢?

首先, A 的前 $\sqrt{n}$ 项容易 $\mathcal{O}(\sqrt{n} \log n)$ 解出。不妨也设 $A = A_0 + A_1$, 其中 A_0 是 A 只保留 $\leq \sqrt{n}$ 位置上的值得到的函数, A_1 是 A 只保留 $> \sqrt{n}$ 位置上的值得到的函数。

那么有 $C = A_0 B + A_1 B$ 。可以通过一次块筛卷积得到 $A_0 B$, 设 $D = C - A_0 B$, 我们只需要用 $A_1 B = D$ 解出 A_1 即可。

而 $D(\lfloor \frac{n}{k} \rfloor) = \sum_{i=1} B(i) S_{A_1}(\lfloor \frac{n}{ik} \rfloor)$, 因此就有 $S_{A_1}(\lfloor \frac{n}{k} \rfloor) = D(\lfloor \frac{n}{k} \rfloor) - \sum_{i=2} B(i) S_{A_1}(\lfloor \frac{n}{ik} \rfloor)$ 。当 $ik \leq \sqrt{n}$ 时 $S_{A_1}(\lfloor \frac{n}{ik} \rfloor)$ 才有值, 所以总共只用枚举 $\sum_{k=1}^{\sqrt{n}} \lfloor \frac{\sqrt{n}}{k} \rfloor = \mathcal{O}(\sqrt{n} \log n)$ 个 i , 时间复杂度 $\mathcal{O}(\sqrt{n} \log n)$ 。

至此, 我们用一次 (函数只在质数处有值的) 块筛 $\exp$ 、一次块筛卷积和 $\mathcal{O}(\sqrt{n} \log n)$ 的额外复杂度做到了块筛 $\ln$ 。

4.3 优化块筛卷积

现在复杂度瓶颈反而在块筛卷积上了。块筛卷积能不能做到更优呢? 这也是能的。

在做块筛卷积的估算的修正部分中, 对于每个 $m \in [n - L, n]$, 我们需要枚举所有满足 $xyt = m$ 的三元组 (x, y, t) 。固定 m , 这部分能不能优于 $d_3(m)$ 呢?

观察我们实际关心的值。

- xy 和 t 的值, 这会影响到被修正的位置;

- $\lceil S \ln(x) \rceil + \lceil S \ln(y) \rceil$ 的值, 这会影响到 xy 位置是否被更新。

注意到一个性质: $S \ln(xy) \leq \lceil S \ln(x) \rceil + \lceil S \ln(y) \rceil < S \ln(xy) + 2$ 。所以在确定了 xy 后, $\lceil S \ln(x) \rceil + \lceil S \ln(y) \rceil$ 只有两种可能。这样, 对于 x 和 y , 我们就只需要知道 $w_x = \lceil S \ln(x) \rceil \bmod 2$ 和 $w_y = \lceil S \ln(y) \rceil \bmod 2$ 了。

我们要做一个 x, y 贡献到 xy 的卷积。设 m 的质因数分解为 $m = \prod_{i=1}^k p_i^{\alpha_i}$ 。上文的卷积实际上是一个 k 维卷积, 第 i 维的值域是 $[0, \alpha_i]$ 。虽然卷积时还要受到 w 的影响, 但是只需要枚举 $w_x = 0/1$ 和 $w_y = 0/1$ 四种情况分别做普通的高维卷积, 或是用单位根反演 (利用 $c = 1/-1$ 时 $c^{w_x} c^{w_y} = c^{(w_x + w_y) \bmod 2}$) 做到两次卷积。

多元多项式的卷积问题可以参考李白天的 2021 年集训队论文 [2], 单次时间复杂度 $d(m)\omega(m) \log d(m)$ 。

考虑如何分析时间复杂度。

选取一个常数 l , $> n^{1/l}$ 的质因子只有 $\mathcal{O}(l)$ 个, 对 $\omega(m)$ 和 $\log(d(m))$ 的贡献均不超过 l , 因此可以忽略。

对于 $\log(d(m))$ 的项, 可以用 $\sum_{i=1}^k \sum_{e=1}^{\alpha_i} \frac{1}{e}$ 来估算。另外, 如果 $p^t | m$, 就有 $(t+1)d(\frac{m}{p^t}) \geq d(m)$, 因此可以将 $d(ipq^e)$ ($p, q \in \mathbb{P}$) 估计为 $\mathcal{O}(d(i)e)$

对于一个 m , $d(m) \log d(m) \omega(m) = \mathcal{O}(d(m)(\sum_{p|m, p \in \mathbb{P}} 1)(\sum_{q \in \mathbb{P}, e \geq 1, q^e | m} \frac{1}{e}))$ 。其中 $p = q$ 的情况在复杂度分析中可以忽略。我们可以进一步估计为 $\mathcal{O}(\sum_{p|m, p \in \mathbb{P}} \sum_{q \in \mathbb{P}, e \geq 1, q^e | m, p \neq q} \frac{e}{e} \times d(\frac{m}{pq^e}))$

所以我们可以这样分析:

$$\begin{aligned} & \sum_{m \in [n-L, n]} \sum_{p \leq n^{1/l}, p \in \mathbb{P}, p|m} \sum_{q \leq n^{1/l}, q \in \mathbb{P}, e \geq 1, p \neq q} d(\frac{m}{pq^e}) \\ & \leq \sum_{p \leq n^{1/l}, p \in \mathbb{P}} \sum_{q \leq n^{1/l}, q \in \mathbb{P}, e \geq 1} \sum_{i \in [\frac{n-L}{pq^e}, \frac{n}{pq^e}]} d(i) \end{aligned}$$

注意到 p, q, e 最多只有 $\mathcal{O}(\frac{n^{2/l}}{\ln n})$ 种可能, 所以我们继续使用解析数论中对因数前缀和的估计 [1], 可以将 $\sum_{i \in [L, R]} d(i)$ 估计为 $(R-L+1) \log R + \mathcal{O}(R^{131/416})$ 。只要常数 l 满足 $2/l + 131/416 < 1/2$, 我们就可以将 $\mathcal{O}(R^{131/416})$ 忽略。

所以我们可以进一步将复杂度估计成 $\sum_{p \leq n^{1/l}, p \in \mathbb{P}} \sum_{q \leq n^{1/l}, q \in \mathbb{P}, e \geq 1} \frac{L}{pq^e}$ 。

由于 $\sum_{p \leq S, p \in \mathbb{P}} \frac{1}{p}$ 是 $\mathcal{O}(\log \log S)$ 级别的, 而 $\sum_{p \leq S, p \in \mathbb{P}, e \geq 1} \frac{1}{p^e}$ 不超过两倍的前者, 所以这部分的复杂度就可以估计成 $L(\log \log n)^2$ 。

总复杂度是 $\frac{n \log^2 n}{S} + S(\log \log n)^2$ 。当 S 取 $\frac{\sqrt{n} \log n}{\log \log n}$ 时, 时间复杂度是 $\mathcal{O}(\sqrt{n} \log n \log \log n)$ 。

5 总结

在本文中, 我们得出了一种比较具有拓展性的筛法, 将块筛卷积、块筛除法、积性函数块筛前缀和和质数前缀统计问题做到了 $\mathcal{O}(\sqrt{n} \text{polylog } n)$ 的时间复杂度 (后两者需要满足



$f(p)$ 是关于 p 的常数次多项式或一些其他的特殊性质)。本文还探讨了这种筛法的优化, 并将数论函数的块筛卷积和块筛做到了 $\mathcal{O}(\sqrt{n} \log n \log \log n)$ 的时间复杂度, 将积性函数块筛前缀和和质数前缀统计问题做到了 $\mathcal{O}(\frac{\sqrt{n} \log^2 n}{\sqrt{\log \log n}})$ 。

虽然时间复杂度更加优秀, 但是由于常数较大、 $\log$ 较多, 这个做法在目前 OI 的数据范围中的表现比不过 Min_25 筛法。希望读者能够对积性函数求和作进一步的思考和研究, 期待有更加实用的做法出现。

6 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练彭思进、杨耀良的指导。

感谢学军中学徐先友等老师的关心与指导。

感谢家人对我的陪伴与支持。

感谢信友队的同学、学长给予我帮助与陪伴。

感谢章弥炫、张恒毅等同学为本文验稿并提出建议。

感谢其他给予我帮助的老师与同学。

参考文献

[1] Divisor summatory function. https://en.wikipedia.org/wiki/Divisor_summatory_function.

[2] 李白天. 信息学竞赛中的生成函数计算理论框架. 2021
年信息学奥林匹克中国国家队候选队论文集, 2021.



浅谈线性拟阵奇偶问题

华东师范大学第二附属中学 郭羽冲

摘要

线性拟阵奇偶问题 (Linear Matroid Parity Problem, lmpp) 定义了一类线性空间上的最优化问题。此模型建立了代数方法与组合最优化问题之间的联系。许多经典的组合问题 (例如: 一般图最大匹配、网络流) 均可归约到此问题。这种归约带来的代价是复杂度的上升, 但却得到了这些看似不同的问题内部的一些共性, 并用统一的模型解决它们。

1 记号与约定

e_i 表示第 i 个分量为 1, 其余分量均为 0 的单位向量。

对于一个矩阵 A 以及两个集合 S, T , 定义 $A_{S,T}$ 表示 A 中只保留 S 中的行以及 T 中的列所得到的子矩阵。

$\deg_x(F)$ 表示 (多元) 多项式 F 中 x 的最高次数。

2 线性拟阵奇偶问题

2.1 定义

给定 m 个二元组 (a_i, b_i) , 其中 a_i, b_i 均为 n 维向量。选出最多的二元组, 使得所有被选出的二元组中包含的向量线性无关。

2.2 稀疏型构造与紧凑型构造

令 v 为原问题的答案。

令 $x'_1 \dots x'_m$ 为未定元。构造矩阵 A, X :

$$A = (a_1, b_1, a_2, b_2, \dots, a_m, b_m)$$

$$X = \text{diag} \left(\begin{pmatrix} 0 & x'_1 \\ -x'_1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & x'_2 \\ -x'_2 & 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 & x'_m \\ -x'_m & 0 \end{pmatrix} \right)$$

将矩阵放在 $x'_1 \dots x'_m$ 的有理分式域下考虑, 则有 $2v + 2m = \text{rank} \begin{pmatrix} 0 & A \\ -A^T & X \end{pmatrix}$ 。此即稀疏型构造。

令 $x_1 \dots x_m$ 为未定元。构造矩阵 M :

$$M = \sum_{i=1}^m x_i (a_i b_i^T - b_i a_i^T)$$

将矩阵放在 $x_1 \dots x_m$ 的有理分式域下考虑则有 $2v = \text{rank}(M)$ 。此即紧凑型构造。

相比稀疏型构造, 紧凑型构造得到的矩阵规模较小, 在计算的时间复杂度上有优势。因此之后的算法均采用紧凑型构造进行处理。

稀疏与紧凑型构造的正确性在参考文献 [1] 中已有较为详细的证明, 这里不再详细展开。

2.3 算法

为了得到较好的时间复杂度, 我们需要对未定元 $x_1 \dots x_m$ 进行一些特殊处理。

任取一个大质数 p , $x_1 \dots x_m$ 均从 $[0, p) \cap \mathbb{Z}$ 中随机选取, 在 M 中将 $x_1 \dots x_m$ 的值代入后得到 M' 。 M' 为 $\mathbb{F}_p$ 上的矩阵, 可以用高斯消元 $O(n^3)$ 地计算 $\text{rank}(M')$ 。我们认为 $\text{rank}(M) = \text{rank}(M')$, 即可进一步得到答案。

2.4 正确率分析

上述算法是有一定错误概率的, 考虑对其进行分析。

可以发现一定有 $\text{rank}(M') \leq \text{rank}(M)$, 下证明两者有较大概率取等。

令 $r = \text{rank}(M)$, 则 M 一定有一个 r 阶可逆子矩阵 $M_{S,T}$ 。

$\det(M_{S,T})$ 为关于 $x_1 \dots x_m$ 的非零多元多项式, 而 $\det(M'_{S,T})$ 即为将 $x_1 \dots x_m$ 的值代入多项式得到的结果。因此考虑如下引理:

引理 (Schwartz-Zippel) 若 $f(x_1 \dots x_m)$ 为不超过 d 次的非零多元多项式, 且 $x_1 \dots x_m$ 均从集合 S 中随机选取, 则 $f(x_1 \dots x_m) = 0$ 的概率不超过 $\frac{d}{|S|}$ 。

证明 当 $m = 1$ 时多项式至多有 d 个根, 显然成立。

若在 $m - 1$ 处成立, 则单独提出 x_m :

$$f(x_1 \dots x_m) = \sum_{i=0}^d x_m^i f_i(x_1 \dots x_{m-1})$$

令 k 为 x_m 的最高次数。

取定 $x_1 \dots x_{m-1}$, 令 f' 为将 $x_1 \dots x_{m-1}$ 的值代入 f 后得到的关于 x_m 的一元多项式。

若 f' 为零多项式, 则意味着 $f_k(x_1 \dots x_{m-1}) = 0$, 而 f_k 次数不超过 $d - k$, 因此这种情况出现的概率不超过 $\frac{d-k}{|S|}$ 。

否则, 使得 $f'(x_m) = 0$ 的 x_m 的取值不超过 k 种, 因此这种情况出现的概率不超过 $\frac{k}{|S|}$ 。

综上, $f(x_1 \dots x_m) = 0$ 的概率不超过 $\frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}$ 。

由归纳, 结论 $\forall m \geq 1$ 成立。

于是 $\det(M'_{S,T}) = 0$ 的概率不超过 $\frac{r}{p}$ 。因此我们有至少 $1 - \frac{r}{p}$ 的概率能够得到正确的结果。

2.5 构造方案

依次考虑每个二元组, 若将当前二元组删除之后答案不会变小, 则将其删除, 否则保留。最终一定会剩下恰好 v 个二元组, 它们即构成了一组答案。

若使用暴力算法, 则我们需要在删除每个二元组之后在 $O(n^3)$ 的时间复杂度内求 $\text{rank}(M)$ 。则总时间复杂度为 $O(mn^3)$ 。

考虑到 $\text{rank}(a_i b_i^T - b_i a_i^T) \leq 2$, 我们可以使用经典的矩阵低秩扰动方法加速计算。

具体地, 我们有如下引理:

引理 (Sherman-Morrison-Woodbury, SMW) 对于可逆 n 阶矩阵 A, B , $n \times m$ 矩阵 U , $m \times n$ 矩阵 V , 有:

$$(A - UB^{-1}V)^{-1} = A^{-1} + A^{-1}U(B - VA^{-1}U)VA^{-1}$$

证明 我们只证:

$$(I - UV)^{-1} = I + U(I - VU)^{-1}V$$

容易变换为原命题形式。

考虑如下分块矩阵等式:

$$\begin{pmatrix} I & 0 \\ -V & I \end{pmatrix} \begin{pmatrix} I & U \\ 0 & I \end{pmatrix} \begin{pmatrix} I - UV & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ V & I \end{pmatrix} \begin{pmatrix} I & -U \\ 0 & I \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I - VU \end{pmatrix}$$

两边取逆得:

$$\begin{pmatrix} I & -U \\ 0 & I \end{pmatrix}^{-1} \begin{pmatrix} I & 0 \\ V & I \end{pmatrix}^{-1} \begin{pmatrix} (I - UV)^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & U \\ 0 & I \end{pmatrix}^{-1} \begin{pmatrix} I & 0 \\ -V & I \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 \\ 0 & (I - VU)^{-1} \end{pmatrix}$$

$$\begin{pmatrix} (I - UV)^{-1} & 0 \\ 0 & I \end{pmatrix} = \begin{pmatrix} I & 0 \\ V & I \end{pmatrix} \begin{pmatrix} I & -U \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & (I - VU)^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -V & I \end{pmatrix} \begin{pmatrix} I & U \\ 0 & I \end{pmatrix}$$

展开化简后即可得到原式。

我们同时证明了 $(A - UB^{-1}V)$ 可逆等价于 $(B - VA^{-1}U)$ 可逆。

但之前所构造的矩阵 M 未必可逆，无法直接利用上述引理。

由 M 的反对称性， M 一定有一个 r 阶可逆主子矩阵 $M_{S,S}$ 。

将所有向量在 S 上截取，即只保留 S 所给出的 r 个分量。对于截取后的问题中构造出一组方案，那么它一定为原问题的一组方案。

任取 r 个线性无关的行作为 S ，容易证明 $M_{S,S}$ 即为满足条件的主子矩阵。因此我们可以在 $O(n^3)$ 的时间复杂度内求出一组合法的 S ，将问题归约到 M 可逆的情况。

下假设 M 可逆。实时维护 $N = M^{-1}$ 。

令 $U = (a_i, -b_i)$, $V = (b_i, a_i)^T$ ，则 $a_i b_i^T - b_i a_i^T = UV$ 。

我们相当于要判断 $M - UV$ 是否可逆。由 SMW 引理可转化为判断 $I - VNU$ 是否可逆，可以在 $O(r^2)$ 的时间内计算出 $I - VNU$ 。而这是一个 2×2 的矩阵，可以 $O(1)$ 判断其可逆性。

若可逆，则将 N 更新为 $N' = (M - UV)^{-1}$ 。由 SMW 引理得：

$$N' = (M - UV)^{-1} = N + NU(I - VNU)^{-1}VN$$

可以在 $O(r^2)$ 的时间复杂度内计算出 N' 。总时间复杂度降为 $O(mr^2)$ 。

2.6 扩展

线性拟阵奇偶问题的定义要求每个组均有两个向量。考虑解决某些组中可能只有一个向量的情况。

设共有 m_1 个一元组暴力的想法是，将向量扩为 $n + m_1$ 维，对于第 i 个一元组，给它配上一个 $b_i = e_{n+i}$ 。正确性显然。但这会导致向量的维数增加至 $n + m_1$ ，在 m_1 较大时会增加时间复杂度。

考虑上述暴力最终得到的矩阵形式：

$$M = \begin{pmatrix} M_2 & A_1 \\ -A_1^T & 0 \end{pmatrix}$$

其中 M_2 为只考虑二元组时得到的 M ， A_1 为所有一元组中的向量依次排列组成的矩阵。

我们只关心 $\text{rank}(M)$ ，因此可以只保留 A_1 的一组列基而不影响答案，而基的大小 $\leq n$ 。此时我们将问题归约为了 $m_1 \leq n$ 的情况，此时再执行上述暴力算法不会增加时间复杂度。

2.7 小结

本节介绍了线性拟阵奇偶问题的求解与构造方案的通用随机化算法，同时提出了扩展形式下不增加时间复杂度的算法。

其时间复杂度组成如下：

- 求 $M = \sum_{i=1}^m x_i(a_i b_i^T - b_i a_i^T)$, $O(mn^2)$ 。
- 求 $\text{rank}(M)$, $O(n^3)$ 。

若构造方案，则需要增加：

- 求 M^{-1} , $O(n^3)$ 。
- 每次低秩扰动后判断是否可逆，更新 N , $O(mr^2)$ 。

实际情况中根据 a_i, b_i 的特殊性质可能可以进一步优化时间复杂度。

3 带权线性拟阵奇偶问题

3.1 简介

带权线性拟阵奇偶问题，顾名思义，是线性拟阵奇偶问题的带权版本。**参考文献 [4]** 中提出了本问题的 $O(\text{poly}(n, m))$ 算法。但此算法非常复杂，超出了笔者目前的能力范围。因此本节中我们将介绍一个相对简单但时间复杂度较高（非多项式）的算法。

3.2 定义

给定 m 个二元组 (a_i, b_i) ，其中 a_i, b_i 均为 n 维向量。同时第 i 个二元组有正整数权值 w_i 。选出权值和最大的若干二元组，使得所有被选出的二元组中包含的向量线性无关。

3.3 稀疏型构造与紧凑型构造

令 v 为原问题的答案。

令 w_i 的值域为 $[1, V] \cap \mathbb{Z}$ 。我们新引入一个未定元 z ，利用它的次数来刻画权值和。

类似于不带权情况下的稀疏型构造，令：

$$A = (a_1 z^{w_1}, b_1, a_2 z^{w_2}, b_2, \dots, a_m z^{w_m}, b_m)$$

$$X = \text{diag} \left(\begin{pmatrix} 0 & x'_1 \\ -x'_1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & x'_2 \\ -x'_2 & 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 & x'_m \\ -x'_m & 0 \end{pmatrix} \right)$$

$$Y = \text{diag} (y_1, y_2, \dots, y_n)$$

则我们有如下定理:

定理

$$2v = \deg_z \left(\det \begin{pmatrix} Y & A \\ -A^T & X \end{pmatrix} \right)$$

证明 Y 为对角矩阵, 因此将行列式展开得:

$$\det \begin{pmatrix} Y & A \\ -A^T & X \end{pmatrix} = \sum_S \det \begin{pmatrix} 0 & A \\ -A^T & X \end{pmatrix}_{S,S} \det(Y_{\bar{S}, \bar{S}})$$

其中 $\bar{S} = ([1, n+2m] \cap \mathbb{Z}) \setminus S$.

而反对称矩阵中有 $\det(A) = \text{pf}(A)^2$, 因此:

$$\deg_z \left(\det \begin{pmatrix} Y & A \\ -A^T & X \end{pmatrix} \right) = 2 \deg_z \left(\sum_S \text{pf} \begin{pmatrix} 0 & A \\ -A^T & X \end{pmatrix}_{S,S} \det(Y_{\bar{S}, \bar{S}}) \right)$$

由 Pffafian 的定义展开可得:

$$\text{pf} \begin{pmatrix} 0 & A \\ -A^T & X \end{pmatrix}_{S,S} = \sum_{T \subseteq S} \text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T} \text{pf} \begin{pmatrix} 0 & 0 \\ 0 & X \end{pmatrix}_{S \setminus T, S \setminus T}$$

因为 X, Y 的非零主子式均互不相同, 所以左式中 z 的最高次数一定是通过选择一组合适的 S, T 使得 $\det(Y_{\bar{S}, \bar{S}}) \neq 0, \text{pf} \begin{pmatrix} 0 & 0 \\ 0 & X \end{pmatrix}_{S \setminus T, S \setminus T} \neq 0$ 且 $\deg_z \left(\text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T} \right)$ 最大所贡献而来。

由 Y 右下角均为 0 可得 $n+1 \dots n+2m$ 均在 S 中。由 S 左上角均为 0 可得 $1 \dots n$ 均不在 T 中。

因此 $\text{pf} \begin{pmatrix} 0 & 0 \\ 0 & X \end{pmatrix}_{S \setminus T, S \setminus T}$ 非零当且仅当 $\forall i \in [1, m] \cap \mathbb{Z}$, 有 $n+2i, n+2i-1$ 同时在或同时不在 T 中。

而这与原问题的形式是高度吻合的。 $T \cap [n+1, n+2m]$ 刻画了每个二元组是否被选择 (在 T 中的元素对应的二元组均被选择, 否则不选择)。

假设已经固定了 $T \cap [n+1, n+2m]$ 。为了使得 $\text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T}$ 确实能够贡献出它所对应的 $\deg_z$, 我们还需要将 $[1, n] \cap \mathbb{Z}$ 的一个子集加入 T 中以确保其 Pffafian 非零。

若 $\text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T} \neq 0$, 则有:

$$\deg_z \left(\text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T} \right) = \sum_{n+2i \in T} w_i$$

通过简单的分析可得，能够找到这样的一个子集当且仅当所有被选择的向量线性无关，而这恰好是原题的要求。因此 $v = \max \left(\deg_z \left(\text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}_{T,T} \right) \right)$ (要求 S, T 满足之前列出的条件)。

定理得证。

考虑将其化为紧凑型构造。我们有如下引理：

引理 (Schur 补) 若 D 可逆，则如下等式成立：

$$\det \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \det(A - BD^{-1}C) \det(D)$$

证明

$$\begin{pmatrix} I & -BD^{-1} \\ 0 & I \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} I & 0 \\ -D^{-1}C & I \end{pmatrix} = \begin{pmatrix} A - BD^{-1}C & 0 \\ 0 & D \end{pmatrix}$$

由上述引理可得：

$$2v = \deg_z \left(\det \begin{pmatrix} Y & A \\ -A^T & X \end{pmatrix} \right) = \deg_z (\det(Y + AX^{-1}A^T))$$

令 $x_i = -\frac{1}{x'_i}$ ，展开可得：

$$Y + AX^{-1}A^T = Y + \sum_{i=1}^m -\frac{1}{x'_i} (a_i b_i^T - b_i a_i^T) z^{w_i} = Y + \sum_{i=1}^m x_i (a_i b_i^T - b_i a_i^T) z^{w_i}$$

令此矩阵为 M ，则我们得到了带权情况下的紧凑型构造，有 $2v = \deg_z(\det(M))$ 。

3.4 算法

只需要求 $\det(M)$ 即可。

与不带权的情况类似，其中的未定元 x_i, y_i 均可使用随机取值的方式处理。

M 中每个元素均为关于 z 的不超过 V 次多项式，因此 $\det(M)$ 为关于 z 的不超过 Vn 次多项式。

因为多项式的系数是在 $\mathbb{F}_p$ 下考虑的，所以我们可以任选 $Vn + 1$ 个不同的 z 的取值 $z_1 \dots z_{Vn+1}$ ，并求出代入每个 z_i 后 $\det(M)$ 的值，再使用拉格朗日插值法还原出 $\det(M)$ 对应的多项式。



3.5 构造方案

与之前的想法类似,我们维护代入每个 z_i 时的矩阵 $M_1 \dots M_{Vn+1}$ 以及它们的逆 $N_1 \dots N_{Vn+1}$ 。此时我们需要支持对矩阵低秩扰动,并维护其行列式。我们有如下引理:

引理 (Matrix Determinant Lemma) 若 A, B 可逆, 则如下等式成立:

$$\det(A - UB^{-1}V) = \frac{\det(A)}{\det(B)} \det(B - VA^{-1}U)$$

证明 与 SMW 引理的证明类似, 这里不再赘述。

假设 $M_1 \dots M_{Vn+1}$ 始终可逆, 则我们总是能用上述方法维护出行列式。

但实际上可能在某个时刻出现 M_i 不可逆的情况, 即 $\det(M_i) = 0$ 。

由于 $\det(M)$ 为关于 z 的不超过 Vn 次多项式, 若我们从 $[0, p) \cap \mathbb{Z}$ 中随机选取 $z_1 \dots z_{Vn+1}$, 则由 Schwartz-Zippel 引理, $\det(M_i) = 0$ 的情况期望会发生 $O\left(\frac{V^2 mn^2}{p}\right)$ 次。

于是我们考虑在发生 $\det(M_i) = 0$ 时更换 z_i 的取值, 不断地从 $[0, p)$ 中随机选取新的 z_i , 直到 z_i 与其它的 z_j 均不同且 $\det(M_i) \neq 0$ 即可。

若我们取足够大的 p , 则重构的时间复杂度可以忽略不计。

此时我们已实时维护出了 $\det(M_1) \dots \det(M_{Vn+1})$, 每次进行拉格朗日插值还原出 $\det(M)$ 后即可判断当前二元组是否在答案中。

3.6 小结

本节介绍了带权线性拟阵奇偶问题的求解与构造方案的通用随机化算法。其时间复杂度组成如下:

- 求矩阵中每个元素 $z_1 \dots z_{Vn+1}$ 处的值 (多点求值), $O(Vn^3 \log^2(Vn))$ 。
- 求每个 z_i 处的行列式, $O(Vn^4)$ 。
- 插值还原 $\det(M)$, $O(Vn \log^2(Vn))$ 。

若构造方案, 则需要增加:

- 求 $N_1 \dots N_{Vn+1}$, $O(Vn^4)$ 。
- 每次低秩扰动后插值还原 $\det(M)$, $O(Vmn \log^2(Vn))$ 。
- 更新 $\det(M_1) \dots \det(M_{Vn+1})$ 以及 $N_1 \dots N_{Vn+1}$, $O(Vmn^3)$ 。

4 线性拟阵奇偶问题的应用

4.1 简介

本节将会介绍若干将组合最优化问题归约到线性拟阵奇偶问题的案例。

4.2 一般图最大匹配问题

4.2.1 定义

给定一个无向图，要求选出最多的边满足任意两条选出的边均无公共点。

4.2.2 算法

解决此问题的经典方法为带花树。

若第 i 条边为 (u_i, v_i) ，则令 $a_i = e_{u_i}, b_i = e_{v_i}$ 。

若选出的边有公共点 u ，则 e_u 这个向量一定出现了至少两次，一定线性相关。否则每个 e_u 至多出现一次，一定线性无关。

因此上述构造将一般图最大匹配问题归约到线性拟阵奇偶问题。

有趣的是，观察我们得到的矩阵 M ，与一般图最大匹配的另一经典解法：Tutte 矩阵完全一致。

令边数为 n ，点数为 m 。则直接使用通用算法构造方案的时间复杂度为 $O(mn^2)$ 。

4.2.3 优化

考虑利用 a_i, b_i 的性质进一步优化，我们有如下引理：

引理 (Harvey) 若 A 可逆，且 A 与 B 只有集合 S 对应的主子矩阵不同。令 $D = B - A$ ，则有：

$$(B^{-1})_{T,T} = (A^{-1})_{T,T} - (A^{-1})_{T,S}(I + D_{S,S}(A^{-1})_{S,S})^{-1}D_{S,S}(A^{-1})_{S,T}$$

证明 令 S_i 为 S 中第 i 小的元素， $U_{i,j} = [S_j = i]$ ， $V = D_{S,S}U^T$ 。由 SMW 引理得：

$$((A + UV)^{-1})_{T,T} = (A^{-1})_{T,T} - (A^{-1}U(I - VA^{-1}U)^{-1}VA^{-1})_{T,T}$$

将 U, V 定义代入即可得到原式。

同时我们证明了 B 可逆当且仅当 $I + D_{S,S}(A^{-1})_{S,S}$ 可逆。

Harvey 引理启示我们，只需要知道局部的信息即可处理局部的修改。

而在一般图最大匹配问题中, (a_i, b_i) 只会影响 $M_{u_i, v_i}, M_{v_i, u_i}$ 这两个位置。于是考虑分治地
进行计算。

定义过程 $solve(S, T)$ 表示处理所有满足 $u_i \in S, v_i \in T$ 的二元组, 判断它们是否在答案
中。

令 $X = S \cup T$, 则我们只需要关注 $M_{X, X}$ 以及 $(M^{-1})_{X, X}$ 。

若 $|S| = |T| = 1$, 则直接暴力, 时间复杂度为 $O(1)$ 。

否则我们将 S 均分为 S_1, S_2 , 将 T 均分为 T_1, T_2 。依次枚举每一对 $(i, j) \in \{(1, 1), (1, 2), (2, 1), (2, 2)\}$,
并进行如下过程:

- 递归 $solve(S_i, T_j)$ 。
- 在上一步中我们求出了 S_i, T_j 对应的子问题中所有不在答案中的二元组。删除它们后
会将 M 变为 $M + D$, 显然 D 只可能在 X 对应的主子矩阵内有值。因此利用 Harvey
引理在 $O(|X|^3)$ 的时间复杂度内计算出 D 对 $(M^{-1})_{X, X}$ 的影响即可。

$solve(S, T)$ 的过程中递归每个子问题之后均正确地维护出了子问题中的二元组对于 $M_{X, X}$
以及 $(M^{-1})_{X, X}$ 的影响, 因此正确性显然。

为了方便分析时间复杂度, 不妨令 $S = T$ 或 $S \cap T = \emptyset$, 令两种情况的时间复杂度分别
为 $T_1(|S|), T_2(|S| + |T|)$, 则有:

$$T_2(n) = O(n^3) + 4T_2\left(\frac{n}{2}\right) = O(n^3)$$

$$T_1(n) = 2T_1\left(\frac{n}{2}\right) + 2T_2(n) + O(n^3) = O(n^3)$$

因此 $solve(\{1, 2, \dots, n\}, \{1, 2, \dots, n\})$ 的时间复杂度为 $T_1(n) = O(n^3)$ 。

4.3 最大 mex 生成树

本题出自 QOJ6402 MEXimum Spanning Tree。

4.3.1 定义

给定一个边权为非负整数的无向连通图, 求边权集合 mex 最大的生成树。其中集合 S
的 mex 表示最小的不在 S 中的非负整数。

4.3.2 算法

转化题意，相当于要求出最大的 k ，使得能够选出一个无环子图满足 $[0, k) \cap \mathbb{Z}$ 中的边均有恰好一条。

若固定 k ，将问题转化为判定，则相当于求图拟阵（所有生成森林）与颜色拟阵（所有颜色两两不同的子集）的交，两者均为线性拟阵。而线性拟阵交严格弱于线性拟阵奇偶。

具体地，令点数为 n ，则对于一条边 (u_i, v_i, w_i) （要求 $w_i \in [0, k) \cap \mathbb{Z}$ ），我们令 $a_i = e_{u_i} - e_{v_i}, b_i = e_{n+w_i}$ 。

上述构造实际上给出了将线性拟阵交归约到线性拟阵奇偶的一般方式。

其对应的 M 一定形如 $\begin{pmatrix} 0 & B \\ -B^T & 0 \end{pmatrix}$ 。因此 $\text{rank}(M) = 2\text{rank}(B)$ 。

在本题中，我们从小往大依次枚举 k ，直到 $\text{rank}(B) < k$ 则停止。从 $k-1$ 移动到 k 时会将 B 的第 k 列从全零变为 $\sum_{i=1}^m [w_i = k] a_i$ 。

类似于线性基，一边加入向量一边高斯消元即可。加入一个向量的时间复杂度的为 $O(n^2)$ ，总时间复杂度为 $O(n^3)$ 。

4.4 颜色重数限制生成树

4.4.1 定义

给定一个无向连通图，每条边有一个用正整数表示的颜色，求一个边数最多的生成森林使得 $\forall i$ ，其中颜色为 i 的边数不超过 lim_i 。

4.4.2 算法

将问题看作线性拟阵交。关键在于用线性拟阵刻画“颜色为 i 的边数不超过 lim_i ”。

给颜色 i 分配 lim_i 个维度，不同的边权之间相互独立。问题转化为在 $\mathbb{F}_p$ 下构造 m 个 $k = \text{lim}_i$ 维向量使得其中任意 k 个均线性无关。

只需要任取质数 $p > m$ ，并令第 i 个向量的第 j 维为 i^j 即可。由范德蒙德矩阵行列式易证其正确性。

4.5 其它例题

本题出自 UOJ785 新年的找对。

4.5.1 定义

给定一个无向图，点 u 的颜色为 w_u ，要求选出最多的两两无交点的路径，并且对于每条选出的路径 P ，令其起点终点分别为 u, v ，则满足以下条件：

- $w_u \neq 0, w_v \neq 0, w_u \neq w_v$ 。
- $\forall x \in P, x \neq u, x \neq v$ ，有 $w_x = 0$ 。

4.5.2 算法

转化题意，相当于要选出最多的边，使得只保留这些边时每个连通块中至多有两个点颜色非零，若某个连通块中有两个点颜色非零，则另外要求它们颜色不同。

稍加说明上述转化正确的原因：转化后的最优解中一定不会有颜色全零的连通块（除非所有点颜色全零），否则可以将它并入一个相邻的连通块中得到更优解。而总点数 - 选出的边数 = 连通块数 = 颜色非零点数 - 有两个颜色非零点的连通块数。总点数与颜色非零点数均为常数，因此我们只需要最大化选出的边数即可。

给颜色 i 分配一个二维向量 α_i ，满足这些向量之间两两线性无关。

对于一条边 (u_i, v_i) ，令 $a_i = e_{2u_i-1} - e_{2v_i-1}, b_i = e_{2u_i} - e_{2v_i}$ 。此时可以保证选出的边集组成森林。

对于一个点 u ，若 $w_u \neq 0$ ，则令 $a_u = \alpha_{w_u,1}e_{2u-1} + \alpha_{w_u,2}e_{2u}$ 。此时有两种方法可以解决问题：

- 将每个颜色非零的点 u 均看作可选可不选的一元组，并使用 2.7 节中提到的扩展形式解决。
- 强制所有颜色非零的点 u 均必选。令点对应的向量张成的子空间为 V_1 ，则我们将 $\mathbb{F}_p^n$ 分解为直和 $V_1 \oplus V_2$ ，并将所有边对应的 a_i, b_i 投影到 V_2 上即可消除强制选点带来的影响。

可以发现，最优解中所有点均必选，因此上述两种方法等价。同时容易证明上述构造可以保证每个连通块均满足要求。

观察本题 a_i, b_i 的形式，我们可以使用一般图最大匹配问题中提到的利用 Harvey 定理分治优化时间复杂度的方法。总时间复杂度为 $O(n^3)$ 。

5 总结

本文介绍了线性拟奇偶问题的通用算法及其扩展与应用，初步地探讨了代数方法在组合最优化问题中的应用。限于笔者能力，本文并没有涉及到诸如带权线性拟阵奇偶问题的多项式算法、非线性拟阵奇偶问题。



近年来学术界在这一方面有许多新的研究成果，同时也有许多其它的代数方法正在不断发展中。目前来说，这些方法还未被 OI 广泛接纳。

笔者希望通过本文能够引起读者探究代数方法的兴趣，并向 OI 中引入这一类全新的问题与方法。

6 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢金靖老师、高开老师、吴申广老师对我的指导。

感谢父母对我的培育和教诲。

感谢常瑞年、万成章学长，柯绎思、柯悻憬同学在本论文的写作中提供的帮助。

感谢所有帮助我的同学、老师。

参考文献

- [1] 常瑞年，浅谈线性代数在 OI 中的应用，IOI 2022 国家集训队论文集
- [2] 任清宇，Goodbye Renyin E 新年的找对 题解
- [3] Ho Yee Cheung, Lap Chi Lau, Kai Man Leung, Algebraic Algorithms for Linear Matroid Parity Problems
- [4] Satoru Iwata, Yusuke Kobayashi, A Weighted Linear Matroid Parity Algorithm



浅谈数据结构的合并与分裂

中国人民大学附属中学 黄洛天

摘要

本文讨论了数据结构的合并与分裂，介绍了线段树、平衡树的合并与分裂，并将其扩展到点分树、边分树、静态 Top Tree 上。提出了静态 Top Tree 的合并与分裂。此外，本文介绍了数据结构的合并与分裂在 OI 中的应用。

1 静态数据结构的合并与分裂

1.1 引入

我们熟悉的静态数据结构有很多种：线段树、点分树、边分树、静态 Top Tree。由于其静态的特点，当我们要合并或分裂就变得尤为方便——对应节点信息直接合并即可，分裂也只需要把某些节点拆成两个同样位置的节点。这点在线段树合并中也得以体现。

1.2 线段树合并

首先从不带懒标记的线段树合并出发，观察基本的算法并试图扩展。

线段树合并是指对于两个动态开点线段树合并为一个，并对同样位置的信息合并。一个暴力的想法是将同时存在于两个线段树的节点暴力合并，对于空节点和一个线段树子树的合并直接返回线段树的子树。

为了维护信息，对于暴力的合并的节点我们有两种选择：

1. 在合并完成子树后，从合并的后的两个儿子的信息合并上来。
2. 直接合并对应节点的信息。

这两种方法各有优劣，方法 1 需要线段树支持从儿子合并。对于大部分情况下线段树维护的信息。比如对于 01 序列，维护区间中 $[010, 01, 10, 0, 1]$ 子序列的出现次数。

但有时信息却不支持从儿子处合并，比如我们在合并线段树套平衡树，即每个线段树节点维护了一个平衡树。这时我们从两个儿子处合并平衡树后，需要复制一份作为当前节

点的信息，但不是任何种类的平衡树或操作都可以持久化。但对位合并平衡树则可以认为是同时在执行多个平衡树合并，可以沿用平衡树合并的时间复杂度分析。

两种方法各有优劣，对于上述两种方法可以处理的范围内，我们可以做到 $O(|T_1 \cap T_2|)$ 的时间复杂度合并了两棵树。经过 n 次修改和合并的总时间复杂度即为 $O(n \log n)$ 。

对于带懒标记的线段树合并，由于动态开点线段树在下放标记时会建立新的节点从而影响时间复杂度，因此我们需要用一些方法限制住新建点的数量。

对于我们正在维护的动态开点线段树，时刻保持每个节点要么没有儿子，要么有两个儿子。这样对于非叶子节点下放标记不会新建节点。于是问题变成如何处理叶子节点与线段树子树的合并。一个想法是由于叶子节点代表的区间往往有统一的性质，因此考虑提炼出这个统一的性质并作为标记打到另一个子树上。比如修改是区间加操作，那么叶子节点所代表的区间的值就全部一样。如果两个树的合并是对位加操作，那么就可以用把这个一样的值看成对于另一个树的子树加标记。

例题 1.1. 给一个二叉树，以 1 为根，有点权 a_i 。每个节点是叶子或有两个儿子。如果一个点是叶子，那么 $b_u = a_u$ ，否则 $b_u = |b_{lson} - b_{rson}| + a_u$ ，其中 $lson, rson$ 为 u 的左右儿子。单点修改 a_u ，每次询问 b_1 ，保证 $n, q \leq 10^5, 0 \leq a_u \leq 20$ 。

考虑离线下来，用线段树维护 u 节点在不同时刻的 b_u ，那么对于非叶子节点需要合并两颗线段树，对于所有节点需要做若干次区间加操作。

主要问题在于，合并一个叶子和一个动态开点线段树子树如何快速处理。此时我们需要进行的操作是将线段树的某所有的 $x \rightarrow |x - v|$ ，此处 v 指叶子节点上的区间加标记。显然与 v 取绝对值并不是一个可合并的标记，因此考虑利用势能处理与 v 绝对值的操作。

令 $\min_u, \max_u$ 表示线段树节点 u 所代表的时刻区间里， b 的最小值和最大值。当我们需要把 u 子树的与 v 取绝对值时可以有如下剪枝：

1. $v \leq \min_u$ 时，给子树打 $x \rightarrow x - v$ 的标记。
2. $v \geq \max_u$ 时，给子树打 $x \rightarrow v - x$ 的标记。

由于到 $x \rightarrow kx + b$ 的操作是可以合并的，因此可以用这两个剪枝。

考虑设计势能函数 $\Phi = \sum_T \sum_u \max_u - \min_u + 1$ ，如果不在这两个剪枝里，那么暴力递归下去会导致 $\max - \min$ 至少减少 1。区间加操作只会使得 $O(\log n)$ 个区间势能改变，显然最多改变 v 也就是不到 20。

那么总时间复杂度就是 $O(nv \log n)$ 。这个例子介绍了当合并叶子节点和线段树子树的时候另一种利用势能处理的方法。

1.3 空间复杂度的优化

线段树合并的空间复杂度直接实现是 $O(n \log n)$ 的，但如果能离线处理这些合并操作，可以优化到 $O(n)$ 。



考虑将合并的过程建树，对这个合并树做重链剖分，按照优先走重儿子方法跑线段树合并。然后对线段树合并写垃圾回收。

然后来证明这个做法的空间复杂度是 $O(n)$ 。

引理 1.1. 经过 m 次操作的动态开点线段树有 $O(m(\log n - \log m))$ 个节点。

考虑对线段树节点分类，依次考虑 $dep \leq \log m$ 的节点和 $dep > \log m$ 的节点。

对于前者即使全部建满也只会 $O(m)$ 个节点。

由于线段树修改是两条链的形式，因此最多有 $O(m)$ 条链。对于后者，每条链最多只能新建 $O(\log n - \log m)$ 个节点，因此总计 $O(m(\log n - \log m))$ 个节点。

引理 1.2. 对于任意一个结点 u 到根的链，会被划分为 $k \leq \log n$ 段重链前缀。假设这 k 段重链的顶端按照从上到下的顺序依次为 $z_1, z_2, \dots, z_k$ ，那么一定有 $size(z_i) \leq \frac{n}{2^{i-1}}$

由于每次往上跳重链，子树大小至少会 $\times 2$ 可得上述引理。

由上两个引理可得任意时刻线段树大小之和可以被限制为 $\sum_{i=1}^k O(\frac{n}{2^{i-1}}(\log n - \log \frac{n}{2^{i-1}})) = \sum_{i=1}^k O(\frac{n}{2^i}) = O(n)$ 。

因此对于后文可以离线的问题，空间复杂度均可以优化到 $O(n)$ 。

对于另一类特殊的动态开点线段树，他们每次修改都是单点修改。利用此性质，也可以将空间复杂度优化到 $O(n)$ 。

具体来说维护所有叶子节点在线段树结构上的虚树，并且只维护虚树节点。对于经过了 m 次操作的动态开点线段树，显然有 $O(m)$ 个虚树节点。

但是由于本方法实现起来较为麻烦，一般采用第一种方法。

1.4 更多静态数据结构的合并

1.4.1 点分树、边分树的合并

我们发现线段树合并具有很强的扩展性。

类似于动态开点线段树，我们可以定义动态开点的点分树、边分树。为了方便叙述，我们只讨论点分树，边分树也是类似的。

由于点分树每个节点有很多儿子，我们难以支持自上而下的检索，一些信息也难以维护。因此考虑将点分树的儿子建哈夫曼树，不难证明树高是 $O(\log n)$ 的。

对于一类点分树问题，经常需要维护重心到当前连通块其他点的信息。那么对于每个节点 u ，他会影响的节点就是点分树上的祖先。由此可见如果我们做单点修改操作只有 $O(\log n)$ 个节点在点分树上维护的信息会变化。

不同于线段树，点分树不支持从儿子上维护的信息合并到当前节点，因此点分树只能采用前文的第二种合并。

例题 1.2 (CTSC2018 暴力写挂). 给两棵带边权的树 T, T' , 边权可以为负数。求点 x, y 使得最大化 $\text{depth}(x) + \text{depth}(y) - (\text{depth}(\text{LCA}(x, y)) + \text{depth}'(\text{LCA}'(x, y)))$ 。

考虑对贡献进行一些转化, 使得第一棵树的 LCA 删去, 得到 $\frac{1}{2}(\text{depth}(x) + \text{depth}(y) + \text{dis}(x, y) - 2\text{depth}'(\text{LCA}'(x, y)))$ 。然后枚举 T' 中的 $\text{LCA}'(x, y)$, 也就是在 T' 上 dfs。假设我们枚举到 u , 每次将 u 的一个儿子 v 合并到 u 的子树内, 并计算贡献。

现在我们将问题转化为: 有若干集合初始每个集合是一个点, 要支持集合合并。合并 S_1, S_2 时需计算 $u \in S_1, v \in S_2, \max \text{dis}(u, v) + \text{depth}(u) + \text{depth}(v)$ 。考虑用边分树处理贡献, 对于每个集合开一个边分树, 每个节点存 $\text{info}_{u,0/1}$ 表示两部分到分治边的 $\text{dep} + \text{dis}$ 最大的权值。对于每个集合开一个动态开点边分树, 做边分树合并。由于空节点的信息是么元, 因此对于空节点和子树的合并直接返回子树。对于重合的两个点可以直接将 $\text{info}_{u,0/1}, \text{info}_{v,0/1}$ 合并为 $\max(\text{info}_{u,0/1}, \text{info}_{v,0/1})$ 。

总时间复杂度为 $O(n \log n)$, 空间复杂度 $O(n)$ 。

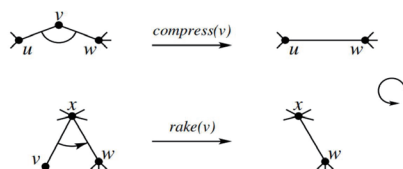
1.4.2 静态 Top Tree 的合并

首先我们将引入静态 Top Tree。

定义 1.1. 对于无向树 T , 定义 T 上一个簇为一个三元组 $C = (V, E, B)$, 其中 (V, E) 连通且是 T 的子图, 集合 $B \subseteq V, |B| \leq 2$, 且对于每个 $x \in V$, 若 $\exists y \notin V, (x, y) \in E(T)$, 则 $x \in B$ 。 B 中元素称为 C 中的界点。我们用 $V(C), E(C)$ 分别表示 C 的点集和边集, 用 $B(C)$ 表示 C 中的界点集合。

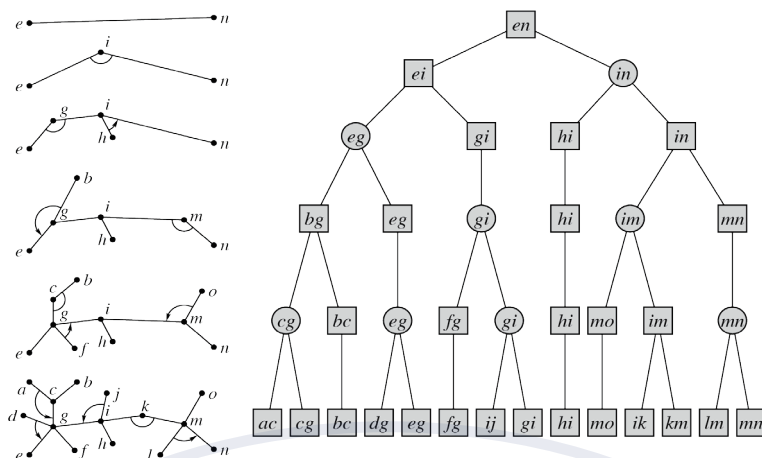
定义 1.2. compress 操作是指将两个簇合并为一个簇的操作。其中, 将两个界点集合交为 1 的簇合并成一个簇, 新的界点集合为两个界点集合的对称差。

定义 1.3. rake 操作是指将两个簇合并为一个簇的操作。其中, 将两个界点集合交为 1 的簇合并成一个簇, 新的界点集合为其中一个簇的界点集合。



对于一棵树来说, 我们可以用上面两种操作, 将整棵树进行收缩。具体来说, 可以将一个簇看成是缩成一条界点 (u, v) 之间的边, 初始每一条 (u, v) 的边都看成一个簇, 通过用上面两种操作不断合并簇, 最后可以把整棵树合并成一个簇。

收缩过程中, 簇的合并结构形成了一个树, 我们把这个树称为 Top Tree。



不难发现这样的 Top Tree 满足每个点最多有两个儿子，接下来，我们将给出一种建树方法使得 Top Tree 的深度为 $\Theta(\log n)$ 。

对原树轻重链剖分，dfs 地自底向上处理每一个重链，并认为每一个轻子树都已经合并成一个簇。对于重链上的每一个节点，将这个节点的父亲的轻子树的簇分治地 rake，得到的总簇再用一次 rake 操作合并到这个节点的父边。需要保证深度不会太大，考虑每次分成两半，rake 完两半之后再 rake 成一个簇。我们称这样操作形成的树为 rake tree。然后，我们得到了排成一条链的簇。接着我们就可以将这些簇 compress 为一个，同样需要分治操作来保证深度，得到的树称为 compress tree。

引理 1.3. 如果我们按簇的大小选择带权中点分治，那么这样建立出来的 Top Tree 深度为 $\Theta(\log n)$ 。

无论是 rake tree 还是 compress tree，每向上跳常数层，簇的大小至少会翻倍，因此深度为 $O(\log n)$ 。

接下来我们将讨论用上述算法建出的 Top Tree 的合并等问题。

例题 1.3. 定义一个操作叫选择一个区间 $[l, r]$ ，并把所有在区间里的字符从方括号变圆括号，从圆括号变方括号。

定义一个括号串的权值 $val(S)$ 为：如果这个括号串能通过操作变成合法，就是最小的操作次数；否则是 0。

给出 q 次修改查询，有以下两种可能。

1. 修改，给出一个区间 $[l, r]$ 把所有在区间里的字符从方括号变圆括号，从圆括号变方括号。
2. 查询，给出一个区间 $[l, r]$ ，求 $\sum_{[l', r'] \subseteq [l, r]} val(s[l', r'])$ 。

本题需要对题面中的权值进行初步的观察和转化，与文章关系不大所以在此省略。

本题数据结构需要支持：求一条自浅而深的竖直树链 $a_1, a_2, \dots, a_k$ 中 a_i 儿子中编号比 a_{i+1} 大的子树半群信息，和树链修改。

由于操作没有很好的性质，因此考虑建立出完整的静态 Top Tree。

由于本题儿子之间的顺序是关键，因此每个 compress 点需要连两个 rake 子树，分别表示重儿子左侧的轻儿子信息和重儿子右侧的轻儿子信息。

在 rake tree 或 compress tree 上定位时需要注意以下 case。

假设要定位到 u 到 v 之间的区间。

1. u 或 v 是另一方的祖先。
2. u 或 v 均不是另一方的祖先。
3. u, v 分别在重儿子左右的两个 rake tree 上。
4. u, v 其中一个是重儿子，即不在 rake tree 上。
5. u, v 在同一个 rake tree 上。

这五类描述了静态 Top Tree 定位过程中遇到的各种情况，对于每类如何计算贡献并不难，在本文不详细展开。时间复杂度 $O(n \log n)$ ，空间复杂度 $O(n)$ 。

类似于动态开点线段树，我们可以得到动态开点静态分治树。每次修改我们定位到会影响的 $O(\log n)$ 个节点，建立这些节点及其所有儿子节点。由于每个点要么是叶子，要么有所有儿子，所以只要不对叶子下放标记，则树的节点个数不会变化。

定位的过程不同于线段树自顶向下的检索，只需要记录当前簇对应的界点即可。因为我们无法承担每次寻找新界点的代价，这需要 $O(\log n)$ 的二分。

因此这里往往会先建立一个完整的静态 Top Tree，定位时同时维护在动态开点树和完整的树上的位置，通过完整的树知道需要定位的节点在哪里。具体来说，首先定位到修改两条链的末尾，也就是链修改的端点。然后不停在完整的静态 Top Tree 上跳父亲，并记录所属于哪个儿子。这里只需要一个空间为 $O(\log n)$ 的辅助数组，就可以记录下来路径。然后从根往下走两条路径，走的同时动态开点。这样就以 $O(\log n)$ 额外空间的代价用小常数解决了定位问题。

解决了定位之后我们根据需求会打链标记、子树标记或者做查询操作。

现在有了动态开点的静态 Top Tree，于是考虑怎么合并两个动态开点的静态 Top Tree。我们仍然自上而下的去合并 T_1, T_2 。

对于不带懒标记的情况，可以非常简单采用对应位置的节点暴力的递归的方法，在空节点处返回。此处合并信息存在和线段树一样的问题，需要在两种维护信息的方式中选择一种。假设能 $O(1)$ 合并信息，则得到一个时间复杂度 $O(n \log n)$ 的做法。



对于带懒标记的情况，我们要面对叶子节点和一个子树的合并。此时叶子节点上面带有两种标记——簇界点之间路径的标记和整个簇除了前者的标记。较为常见的情况是可以直接将两种标记打给另一个静态 Top Tree 的对应节点上。但如果不行需要针对簇界点之间路径的标记和整个簇除了前者的两套信息设计一个势能函数。由于两套信息之间存在关联，即 compress 树上维护的非界点之间路径的信息是 rake 树上两者之和。所以利用势能暴力递归时，需要在两种信息之间切换。

例题 1.4. 给你一个 n 个点的有根树，根为 1。每条边上有一个字符 $c = \{0, 1\}$ 。 S_u 表示从根到 u 的路径上每条边的字符依次写下来组成的字符串。保证每个节点向儿子的边上的字符互不相同。

对每个点 u ，有一个价值 val_u 和一个限制 a_u 。

对每个点 u 求出 $ans_u = \sum_{0 \leq i \leq |S_u|} \max_{i \geq a_u \wedge S_u = S_v[|S_v|-|S_u|+1, |S_v|] \wedge S_u[1, i] = S_v[1, i]} val_v$ 。

对输入 trie 树建立 ac 自动机，问题可以转化为对于 ac 自动机上每个节点 u ，维护一个带点权的 trie 树。需要支持查询链和，链取 max。合并两个 trie 树是对应节点点权取 max。

由于题面保证了树为二叉树而且没有子树修改查询，因此可以省略 rake tree 和非簇界点之间路径信息的维护。

考虑到区间取 max，区间和已经是 segment tree beats 的经典应用。所以在每个节点上维护最小值，最小值个数，次小值，区间和作为信息。同时维护全体最小值加多少，作为懒标记。在做链修改的时候先定位到节点上，然后仿照 beats，如果次小值大于等于 val ，则暴力递归，否则根据以后信息剪枝。容易证明这部分时间复杂度 $O(n \log n)$ 。

对于树合并的情况，仍然套用上文通用的静态 Top Tree 合并。唯一需要处理的是，对于叶子节点和静态 Top Tree 子树的合并，此时相当于对 compress 树上的点做与 val 取 max 的修改操作，其中 val 为叶子上的加法标记。这个取 max 的修改操作可以沿用 beats 的方法。

因此本题就在 $O(n \log n)$ 时间复杂度， $O(n)$ 空间复杂度下解决了。

1.5 线段树分裂

线段树分裂支持把线段树分裂成两个值域不交的部分。注意线段树分裂与线段树合并并不是逆操作的关系。

具体而言如果需要把序列 a_i 划分为 $b_i = \begin{cases} a_i & i \leq p \\ 0 & i > p \end{cases}$, $c_i = \begin{cases} 0 & i \leq p \\ a_i & i > p \end{cases}$ ，同时需要维护 b_i , c_i 的区间信息，考虑通过对维护 a 序列的线段树分裂实现。

注意到如果线段树的某个节点代表的区间 $[l, r]$ 满足 $r \leq p$ ，则这个区间的信息可以完全一样的复制到 b 序列对应的线段树节点上；同理若 $l > p$ ，则区间信息可以复制到 c 序列对应的线段树节点上。两者都不满足的节点显然只有 $O(\log n)$ 个，把他们复制成两份放在对应位置。

此做法只会增加 $O(\log n)$ 个节点，因此原来的势能分析仍然成立。

例题 1.5. 维护一个由 n 个非负整数构成的序列 $a_1, a_2, a_3, \dots, a_n$ ，支持三种操作：

1. 给定区间 $[l, r]$ ，将对应区间内的数异或上 x 。
2. 给定区间 $[l, r]$ ，将对应区间内的数从小到大排序。
3. 给定区间 $[l, r]$ ，求区间内的数的异或和。

由于区间排序操作难以维护，考虑用颜色段均摊的方法规避掉它。

我们时刻的将 a 写为集合 S_i 内的元素排序后依次 $\oplus x_i$ 再拼接起来的形式。

考虑用一个 01 trie 也就是值域线段树维护集合 S_i ，用一个线段树维护全局的 x_i ，以及每个集合异或 x_i 之后的异或和。

每遇到一个操作区间 $[l, r]$ ，将跨过 $l-1, l$ 的集合分裂开， $r, r+1$ 同理。这里需要支持 trie 的分裂。

区间异或操作只需要区间修改 x_i ，区间排序需要将这些 trie 树打上标记之后合并。区间查询在线段树上查询区间异或和。因此 01 trie 上维护子树大小，子树异或和即可。

由于需要分裂，无法建出来操作树。但本题全部为单点修改，于是可以采用压缩动态开点线段树，使空间复杂度变成 $O(n)$ 。时间复杂度为 $O(n(\log n + \log v))$ 。

1.6 更多静态数据结构的分裂

数据结构分裂之后信息从儿子合并而来是很简单的。但如果想直接用原来的信息分裂，则必须维护的信息支持分裂。因此点分树边分树分裂对信息的要求高并且应用范围窄，不作为本文讨论的重点。

考虑静态 Top Tree 的分裂出子树和子树补。

假设我们要分裂出 u 子树，可以在定位出 u 子树在静态 Top Tree 上对应的子树。即 u 节点所在 compress tree 中深度在 u 后面的节点，及其的 rake tree 子树，这一部分可以完整的分裂出来。对于 u 在静态 Top Tree 上的祖先节点由于子树内一部分在原树上的 u 子树，另一部分不在，因此需要复制到两棵树上。

每次只会新建 $O(\log n)$ 个节点。

例题 1.6. 给定一个树 T 。维护若干可重集合 S_i ，元素在 $1 \sim n$ 中。支持以下操作：

1. 将集合 S_p 中所有编号在 u 子树中的元素分裂到 S_q 。
2. 对于所有在 u 到 v 链上的节点 x ，把 x 往 S_p 里插入 k 次。
3. 合并两个集合。

4. 查询所有在 u 到 v 链上的节点 x , 在 S_p 中的出现次数之和。

考虑用静态 Top Tree 维护每个数字的出现次数, 修改等价于链加, 查询等价于链求和, 需要支持合并与分裂。

合并操作只需要对位加, 因此对于叶子和子树的合并是简单的, 直接打懒标记上去即可。分裂时需要先定位到 u 所在的 compress tree, 然后根据 u 的位置将当前节点拆分为两部分, 选择一侧递归。

势能函数设计为 $\Phi = \sum |T_i|$, 即静态 Top Tree 的节点个数, 可以分析得到时间复杂度为 $O(n \log n)$ 。

处理完了子树的分裂, 另一个想法是处理链的分裂。也就是说要将静态 Top Tree 中的一部分 compress tree 复制到 T' 的对应位置, 但是这些 compress tree 连接的 rake tree 并不应该复制到 T' 的对应位置, 而是应该保留在原来的位置。因此 T' 需要复制的节点并不是若干子树的形式, 暴力剥离掉 compress tree 连接的 rake tree 的时间代价是不可接受的。

但也不是遇到了链分裂的问题就会束手无策, 例如以下例题。

例题 1.7. 给定一个树 T 。维护若干可重集合 S_i , 元素在 $1 \sim n$ 中。支持以下操作:

1. 将集合 S_p 中所有编号在 u 到 v 的链中的元素分裂到 S_q 。
2. 对于所有在 u 到 v 链上的节点 x , 把 x 往 S_p 里插入 k 次。
3. 合并两个集合。
4. 查询所有在 u 到 v 链上的节点 x , 在 S_p 中的出现次数之和。

由于静态 Top Tree 本身的结构导致了此问题, 因此需要扩展出一个新的结构。由于本题不需要维护子树信息, 因此对于 compress node 的 rake tree 子树内的信息并不关键。也就是说不一定要让 rake tree 作为当前 compress node 的儿子。

考虑维护双层静态 Top Tree, 将原本的每个点划分为 u 和 u' , 分别表示第一层和第二层。第一层内部的结构同静态 Top Tree 一样。第二层内部只建立 compress tree。对于每个 compress tree 的根节点 rt 有额外的一条边指向 rt' 。

其中只有第二层维护链信息, 第一层仅作为定位和重量平衡的用途。

显然上述结构每个节点儿子个数 $O(1)$ 且树高为 $O(\log n)$ 。

对于动态开点的双层静态 Top Tree, 通过以下方式定位。首先建立一个满的静态 Top Tree, 并在上定位出 u 到 v 链对应的节点。然后找到这些位置在第二层静态 Top Tree 中对应的节点, 并动态开点定位到的点以及祖先, 这里包括第一层的点。

有了动态开点的双层静态 Top Tree, 就可以合并和支持分裂出一个链了。前者是类似于普通静态 Top Tree 的。后者考虑我们要分裂的链在第二层的若干 compress tree 的区间上, 而第二层的 compress tree 并不包含多余的 rake tree, 因此这些子树可以被直接复制为

T' 的对应节点。这些 compress tree 上共用的节点，包括第一层静态 Top Tree 上公用的节点总数为 $O(\log n)$ ，可以新建出来。对于其余节点则保留了除了链以外的信息。

势能函数设计为 $\Phi = \sum |T_i|$ ，即静态 Top Tree 的节点个数，可以分析得到时间复杂度为 $O(n \log n)$ 。

2 动态数据结构的合并与分裂

2.1 引入

动态数据结构的应用非常广泛，由于其动态的特点使得它能应对各种操作。但也因为结构的不确定性，导致了合并时的困难。启发式合并是最常见的应对手段，但是由于多了一个 $\log$ ，且不支持分裂并不能令人满意。因此下文来介绍一类平衡树上的问题。

2.2 平衡树的合并

例题 2.1. 要求维护若干个递增整数序列，要求支持：

1. 将一个序列中所有元素加等于常数。
2. 将一个序列中所有元素取相反数，并反转整个序列。
3. 将两个序列归并为一个。
4. 查询一个序列的区间和。

平衡树有很多种，合并的方法也不同，本文只介绍 treap 的合并。

首先我们证明一些关于 treap 的引理。

引理 2.1. n 个节点的 Treap 的期望深度为 $O(\log n)$ 。

假设所有元素按照 key 排序后的随机值依次为 $b_1, b_2, \dots, b_n$ （不妨假设所有元素各不相同），则对于 x 和 y ，分类讨论：

1. 若 $x \leq y$ ，则 x 是 y 的祖先等价于 $b_x < \min_{x < j \leq y} b_j$
2. 若 $x > y$ ，则 x 是 y 的祖先等价于 $b_x < \min_{y \leq j < x} b_j$

这件事的概率也就是 x 权值最小的概率，即 $\frac{1}{|x-y|+1}$

x 的深度也可以定义为 x 的祖先数，而其期望祖先数即为 $\sum_{i=1}^n \frac{1}{|x-i|+1} \sim O(\log n)$

定义 2.1. Finger Search，即在一个数据结构中，令 $d(x, y)$ 为在 x 和 y 之间的元素个数（包括 x 和 y ），当已经确定 x 的位置后，可以在 $O(f(d(x, y)))$ 的时间内访问 y 的操作。

显然 $f(x) > \log x$ 时没有任何意义, 不如直接重新检索一遍, 因此后文默认 $f(x) = \log x$ 。

引理 2.2. 在 Treap 上, x 到 y 的路径长度为期望 $O(\log d(x, y))$ 。

注意到 x 到 y 的路径长度即是 x 祖先且不是 y 祖先的节点数 + 是 y 祖先且不是 x 祖先的节点数, 根据对称性可以仅考虑前者。

假设 Treap 所有元素从小到大依次为 $a_1, a_2, \dots, a_n$, 其中 $a_i = x$ 且 $a_j = y$, 显然 $d(x, y) = j - i + 1$

考虑一个节点 a_k , 求出其满足“是 x 祖先且不是 y 祖先”的概率, 将其累加即可。我们需要对 k 分类讨论:

1. $1 \leq k \leq i$, 这等价于 k 是 $[k, i]$ 的最小值且 $[k, i]$ 的最小值大于 $(i, j]$ 的最小值, 前者概率为 $\frac{1}{i-k+1}$, 后者概率为 $\frac{j-i}{(i-k+1)(j-k+1)}$, 相乘后即 $\frac{j-i}{(i-k+1)(j-k+1)}$

注意到这就是 $\frac{1}{i-k+1} - \frac{1}{j-k+1}$, 累加后即

$$\sum_{k=1}^i \frac{1}{k} - \sum_{k=j-i+1}^j \frac{1}{k} = \sum_{k=1}^i \frac{1}{k} - \left(\sum_{k=1}^j \frac{1}{k} - \sum_{k=1}^{j-i} \frac{1}{k} \right) \leq \sum_{k=1}^{j-i} \frac{1}{k} \sim O(\log d(i, j))$$

2. $i < k < j$, 类似地概率为 $\frac{j-k}{(k-i+1)(j-i+1)} = \frac{1}{k-i+1} - \frac{1}{j-i+1}$, 累加后即 $\sum_{k=1}^{j-i+1} \frac{1}{k} - 1 \sim O(\log d(i, j))$

由此可见 treap 上 finger search 的方法就是不停的从 x 往上跳直到走到 $\text{lca}(x, y)$, 然后再往下定位到 y 。

然后考虑合并两个 treap T_1, T_2 , 他们的大小分别为 n, m 。

不妨设 $m \leq n$, 令 $p_1 < p_2 < \dots < p_m$ 为 T_2 中的元素在 $n+m$ 个元素中的排名。

考虑按照顺序依次插入 a_i , 则时间复杂度为 $O(\sum_{i=1}^m \log(p_i - p_{i-1}))$, 这里 $a_0 = 0$ 。

由于 $\log$ 的凸性, 上式最大值为 $O(m \log \frac{n}{m})$ 。

考虑设计势能为 $\Phi = \sum |T_i| \log |T_i|$, 每次合并使得势能变化了

$$(n+m) \log(n+m) - n \log n - m \log m \geq m \log(n+m) - m \log m \geq m \log \frac{n}{m}$$

即我们在 $O(m \log \frac{n}{m})$ 的时间内, 使得势能至少增加了 $m \log \frac{n}{m}$, 势能有上限 $n \log n$, 因此总时间复杂度为 $O(n \log n)$ 。

2.3 平衡树的分裂

例题 2.2. 要求维护若干个递增整数序列, 要求支持:

1. 对于一个序列 a , 给定权值 k , 将 a 切成小于 k 和大于等于 k 的两个序列
2. 将一个序列中所有元素加等于常数。

3. 将一个序列中所有元素取相反数，并反转整个序列。
4. 将两个序列归并为一个。
5. 查询一个序列的区间和。

平衡树的分裂也是指根据值域划分成两个不交的部分，对于 treap 来说可以在 $O(\log n)$ 时间复杂度内解决。

对于合并操作，我们考虑暴力的归并两个序列。每次取出序列的开头也就是最小值不妨设为 $x \leq y$ 。

然后分裂出 T_1 里所有值 $\leq y$ 的元素，此时一定有 $x > y$ ，再分裂出 T_2 里值 $\leq x$ 的元素，循环往复直到完成归并。接下来考虑证明此做法的时间复杂度。

对于一棵树 T ，我们把点的权值从小到大依次写出 $a_1, \dots, a_n$ ，考虑 $\Phi = \sum_T \sum_{i=1}^{n+1} \log(a_i - a_{i-1})$ ，其中 $a_0 = 0, a_{n+1} = v$ ，分别表示值域的下界和上界。

我们仍然沿用上一章的做法。令 $k = \sum [c_i \neq c_{i+1}]$ ，其中 $c_i \in \{0, 1\}$ ，表示 c_i 属于 T_1 还是 T_2 。

由此我们可以证明合并两棵树的时间复杂度是 $O(k \log n)$ ，而势能也会因此减少 $O(k)$ 。

Φ 有上界 $O(n \log v)$ ，修改操作不影响势能函数，因此总时间复杂度为 $O(n \log n \log v)$ 。

总结

本文介绍了数据结构的合并与分裂，根据静态和动态分为两个部分。静态部分介绍了一类静态数据结构的合并与分裂思想，从线段树出发，将其扩展到点分树、边分树、静态 Top Tree 上。并提出了静态 Top Tree 上的合并与分裂算法。动态部分介绍了平衡树是如何处理动态带来的缺点，通过 finger search 的角度解决了很多平衡树的合并问题。在信息学竞赛中具有广泛的应用前景。

但数据结构理论中还有很多的问题悬而未决。希望本文能够启发读者继续深入研究相关理论，使其更加成熟。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢人大附中叶金毅、梁霄、谷多玉老师的关心和指导。

感谢家人、朋友对我的支持与鼓励。

感谢楼宇辰同学、胡赵阳同学与我讨论以及给予我的启发。

感谢楼宇辰同学、叶李蹊同学为本文审稿。



参考文献

- [1] 2023 年集训队论文, 程思元,《浅谈静态 Top Tree 在树和广义串并联图上的应用》
- [2] 赵雨扬,《Top tree 相关东西的理论、用法和实现》, <https://uoj.ac/blog/4912>
- [3] Wikipedia, Finger search, https://en.wikipedia.org/wiki/Finger_search
- [4] 胡赵阳, 平衡树与 finger search, <https://www.cnblogs.com/PYWBKTDA/p/14576937>
- [5] 周欣, 关于一个平衡树相关的问题, <https://www.luogu.com.cn/blog/user19567/guan-yu-yi-ge-ping-heng-shu-xiang-guan-di-wen-ti>





浅谈拉格朗日乘数及对偶在OI中的应用

宁波市镇海中学 施开成

摘要

在最优化问题中，拉格朗日乘数是求解受方程约束的函数最值时引入的参数变量，拉格朗日对偶问题是指与一个最优化问题所对应的另一个最优化问题。本文介绍了拉格朗日乘数法，并揭示了拉格朗日对偶，线性规划对偶和纳什均衡问题的内在联系。

1 引言

拉格朗日乘数是拉格朗日提出的一种针对带限制最优化问题的处理手段，它通过添加松弛变量的方式把限制并入最优化的式子中，提供了一种针对此类问题的有效处理手段，并延伸出了拉格朗日乘数法和拉格朗日对偶问题。

拉格朗日对偶是拉格朗日提出的一种关于最优化问题的对偶方法。拉格朗日对偶本身在信息学中并不常见，但它的一种特殊情况是线性规划对偶，从拉格朗日乘数出发能快速推导出线性规划问题的对偶形式。关于拉格朗日对偶问题有 minimax 定理，该定理在双人零和博弈的纳什均衡问题中发挥着重要作用。本文将围绕拉格朗日乘数与拉格朗日对偶展开，详细阐述以上几类问题背后的密切联系。

2 基本记号

令 $\frac{\partial f}{\partial x}$ 表示函数 f 关于变量 x 的偏导数。

令 $\nabla f(x_1, \dots, x_n)$ 表示函数 f 在 $x_1, \dots, x_n$ 处的梯度，即 f 的所有偏导数构成的向量。

令 $(x_1, x_2, \dots, x_n)$ 表示一个 n 维向量，其第 i 维的值为 x_i 。

令 $\mathbb{R}^n$ 表示 n 维欧几里得空间。

令 $\max f, \min f$ 分别表示函数 f 的最大值和最小值，当 f 不存在上界时认为 $\max f = +\infty$ ，当 f 不存在下界时认为 $\min f = -\infty$ 。

$\sum a + b$ 等价于 $(\sum a) + b$ ， $\max a + b$ 等价于 $\max(a + b)$ ， $\min$ 与 $\max$ 相同。

对于 $x, y \in \mathbb{R}^n$ ， $x \leq y$ 成立当且仅当 $\forall i \in [1, n] \cap \mathbb{Z}, x_i \leq y_i$ ， $x \geq y, x < y, x > y$ 同理。

对于一个 $n \times m$ 的矩阵 A ， A_i 表示该矩阵的第 i 行 ($1 \leq i \leq n$)，即一个 m 维行向量。

3 拉格朗日乘数

3.1 引入

考虑一类具有等式约束的最优化问题¹:

$$\begin{aligned} & \max f(\mathbf{x}) \\ & s.t. \begin{cases} g_1(\mathbf{x}) = 0 \\ g_2(\mathbf{x}) = 0 \\ \dots \\ g_m(\mathbf{x}) = 0 \end{cases} \end{aligned} \quad (1)$$

其中 $\mathbf{x} = (x_1, x_2, \dots, x_n)$ 为 n 元实向量², $f, g_1, \dots, g_m$ 均为 $\mathbb{R}^n \rightarrow \mathbb{R}$ 的连续可微函数³。

对于这类最优化问题, 我们可以通过在等式约束前加入系数, 将原问题变为如下形式:

$$\max_{\mathbf{x}} \min_{\lambda_i} f(\mathbf{x}) + \lambda_1 g_1(\mathbf{x}) + \dots + \lambda_m g_m(\mathbf{x}) \quad (2)$$

其中系数 $\lambda_1, \dots, \lambda_m \in \mathbb{R}$ 被称作拉格朗日乘数, 函数 $F(\mathbf{x}, \lambda_1, \dots, \lambda_m) = f(\mathbf{x}) + \lambda_1 g_1(\mathbf{x}) + \dots + \lambda_m g_m(\mathbf{x})$ 被称为拉格朗日函数。

定理 3.1. (1) 式与 (2) 式等价。

证明. 注意到 $g_i(\mathbf{x}) \neq 0$ 时, 只要令 λ_i 取 $+\infty$ 或 $-\infty$, 就会得到 $\min_{\lambda_i} f(\mathbf{x}) + \lambda_1 g_1(\mathbf{x}) + \dots + \lambda_m g_m(\mathbf{x}) = -\infty$, 因此在外层套了一层 $\max_{\mathbf{x}}$ 之后对于取到最大值的 $\mathbf{x}$ 必然有 $g_i(\mathbf{x}) = 0$ 。

3.2 拉格朗日乘数定理

定理 3.2 (拉格朗日乘数定理). 假设 $f, g_1, \dots, g_m : \mathbb{R}^n \rightarrow \mathbb{R}$ 均为连续可微函数。则对于 f 在 $\forall i \in [1, m] \cap \mathbb{N}, g_i(\mathbf{x}_\star) = 0$ 时的任意极值点 $\mathbf{x}_\star$, 都有 $\nabla f(\mathbf{x}_\star), \nabla g_1(\mathbf{x}_\star), \dots, \nabla g_m(\mathbf{x}_\star)$ 线性相关。

由于篇幅有限, 且该定理的证明需要用到隐函数定理等微积分内容, 故此处不作展示。完整证明可以参见 [1]。

推论 3.1. 对于 f 在 $g_i(x_1, \dots, x_n) = 0 (1 \leq i \leq m)$ 时的任意一组极值点 $\mathbf{x}_\star$, 当 $\nabla g_1(\mathbf{x}_\star), \dots, \nabla g_m(\mathbf{x}_\star)$ 线性无关时, 存在唯一一组系数 $\lambda_1, \dots, \lambda_m$ 使得 $\nabla f(\mathbf{x}_\star) = \lambda_1 \nabla g_1(\mathbf{x}_\star) + \dots + \lambda_m \nabla g_m(\mathbf{x}_\star)$ 。

¹s.t. 表示 subject to, 意为以……为条件。

²如无特殊说明, 后文将不区分 $(x_1, x_2, \dots, x_n)$ 与 $x_1, \dots, x_n$ 。

³连续可微函数是指处处可导且导数连续的函数。

这一推论是定理3.2的等价形式,其结论常常被写为拉格朗日函数 $F(x_1, \dots, x_n, \lambda_1, \dots, \lambda_m) = f(x_1, \dots, x_n) + \lambda_1 g_1(x_1, \dots, x_n) + \dots + \lambda_m g_m(x_1, \dots, x_n)$ 关于所有 x_i 和 λ_i 的偏导数均为0。

注意条件中的 $\nabla g_1(\mathbf{x}_*), \dots, \nabla g_m(\mathbf{x}_*)$ 线性无关不可忽略。如果忽略条件,考虑函数 $f(x, y) = x + y, g(x, y) = x^2 + y^2$, 求 $f(x, y)$ 在 $g(x, y) = 0$ 时的最大值。此时拉格朗日函数 $F(x, y, \lambda) = x + y + \lambda(x^2 + y^2)$ 不存在任何临界点⁴, 然而 $g(x, y) = 0$ 时 $f(x, y)$ 的最大值应为 $f(0, 0) = 0$ 。这是因为在 $x = 0, y = 0$ 时有 $\nabla g(x, y) = \mathbf{0}$, 不满足 ∇g 线性无关的条件。拉格朗日乘数定理的严谨形式在OI中并不普及, 建议读者在使用拉格朗日乘数定理时留意这一条件是否满足。

3.3 拉格朗日乘数法

对于问题⁵

$$\begin{aligned} \max f(\mathbf{x}) \\ \text{s.t. } g_1(\mathbf{x}) = 0, g_2(\mathbf{x}) = 0, \dots, g_m(\mathbf{x}) = 0 \quad (m < n) \end{aligned}$$

在答案存在的情况下, 我们可以使用拉格朗日乘数法求解这类问题, 具体过程如下:

定义拉格朗日函数 $F(\mathbf{x}, \lambda_1, \dots, \lambda_m) = f(\mathbf{x}) + \lambda_1 g_1(\mathbf{x}) + \dots + \lambda_m g_m(\mathbf{x})$ 。列出 $n + m$ 个方程 $\frac{\partial F}{\partial x_i}(\mathbf{x}) = 0$ 和 $\frac{\partial F}{\partial \lambda_i}(\mathbf{x}) = 0$, 其中 $\frac{\partial F}{\partial \lambda_i}(\mathbf{x}) = 0$ 等价于 $g_i(\mathbf{x}) = 0$ 。取方程的所有解的 $\mathbf{x}$ 和使得 $\nabla g_1(\mathbf{x}), \dots, \nabla g_m(\mathbf{x})$ 线性相关且满足 $g_i(\mathbf{x}) = 0$ 的所有 $\mathbf{x}$, 分别代入 $f(\mathbf{x})$ 计算并取最大值, 该值即为 $f(\mathbf{x})$ 在 $g_i(\mathbf{x}) = 0$ 约束下的最大值。

例 3.1. 求 $f(x, y) = -x^2 + 4x + y$ 在 $x^2 + 2y^2 = 30$ 条件下的最大值。

解 3.1. 考虑拉格朗日乘数法, 令函数 $g(x, y) = x^2 + 2y^2 - 30, F(x, y, \lambda) = f(x, y) + \lambda g(x, y) = -x^2 + 4x + y + \lambda(x^2 + 2y^2 - 30)$ 。

当 $\nabla g(x, y)$ 线性相关 (即 $\nabla g(x, y) = \mathbf{0}$) 时, 有 $\frac{\partial g}{\partial x} = 2x = 0, \frac{\partial g}{\partial y} = 4y = 0$, 解得 $x = 0, y = 0$, 此时 $g(x, y) = -30$, 不符合 $g(x, y) = 0$ 的限制。故我们可以认为 $\nabla g(x, y)$ 在 $f(x, y)$ 取到极值时线性无关, 从而使用拉格朗日乘数定理。

当 $\nabla F(x, y, \lambda) = \mathbf{0}$ 时, 有方程 $\frac{\partial F}{\partial x} = 2(\lambda - 1)x + 4 = 0, \frac{\partial F}{\partial y} = 4\lambda y + 1 = 0, \frac{\partial F}{\partial \lambda} = x^2 + 2y^2 - 30 = 0$ 。该方程组共有四组解, 代入 $f(x, y)$ 可知最优解 $x \approx 1.8715, y \approx 3.6399, \lambda \approx -0.0687, f(x, y) \approx 7.6234$ 。

由于 $g(x, y) = 0$ 时, x, y 有界, 故 $f(x, y)$ 的最大值存在。因此上述讨论包含全部情况, $f(x, y)$ 在 $g(x, y) = 0$ 时的全局最大值为 $x \approx 1.8715, y \approx 3.6399, f(x, y) \approx 7.6234$ 。

3.3.1 [NOI2012] 骑行川藏

题目描述 给定实数 s_i, k_i, v'_i, E , 最小化 $f(v_1, \dots, v_n) = \sum_{i=1}^n \frac{s_i}{v_i}$ 。其中 $v_1, \dots, v_n$ 均为正实数而

⁴一个点是临界点当且仅当该点处的所有偏导数均为0。

⁵当 $m \geq n$ 时, 有 $\nabla f(\mathbf{x}_*), \nabla g_1(\mathbf{x}_*), \dots, \nabla g_m(\mathbf{x}_*)$ 线性相关恒成立。

且需要满足 $\sum_{i=1}^n k_i(v_i - v'_i)^2 s_i \leq E$ 。

数据范围 $1 \leq n \leq 10000, 0 \leq E \leq 10^8, s_i \in [0, 10^5], k_i \in (0, 15], v'_i \in (-100, 100)$ 。

解法分析 稍作分析可以发现，当 $f(v_1, \dots, v_n)$ 在题目的条件下取到最小值时，必然有 $\sum_{i=1}^n k_i(v_i - v'_i)^2 s_i = E$ 和 $v_i \geq v'_i$ 。因此我们定义 $g(v_1, \dots, v_n) = \sum_{i=1}^n k_i(v_i - v'_i)^2 s_i - E$ ，并使用拉格朗日乘数法求解 $F(v_1, \dots, v_n, \lambda) = f(v_1, \dots, v_n) - \lambda g(v_1, \dots, v_n)$ 的临界点并特判 ∇g 线性相关的情况。虽然题目中有 $0 < v_i$ 的限制，但注意到拉格朗日乘数法会求出全部极值点，我们只考虑满足 $0 < v_i$ 的极值点即可。

列出方程 $\frac{\partial F}{\partial v_i} = -\frac{s_i}{v_i^2} + 2\lambda s_i k_i(v_i - v'_i) = 0, \frac{\partial F}{\partial \lambda} = \sum_{i=1}^n k_i(v_i - v'_i)^2 s_i - E = 0$ 。整理得

$$2\lambda s_i k_i(v_i - v'_i) = \frac{s_i}{v_i^2} \quad (3)$$

$$\sum_{i=1}^n k_i(v_i - v'_i)^2 s_i = E \quad (4)$$

对于一个确定的 λ ，可以从 (3) 式中解出所有的 v_i ，然后代入 (4) 式中检验。考虑 (3) 式等式两端的函数图像，当 $\lambda > 0$ 时，等式两端的函数在 $v_i > 0$ 时有且仅有一个交点，且必然有 $v_i \geq v'_i$ 成立；当 $\lambda \leq 0$ 时，等式两端的函数在 $v_i \geq v'_i$ 时不存在交点。故必然有 $\lambda > 0, v_i \geq v'_i$ ，并且由函数图像可知 λ 递增时 v_i 会递减，此时 (4) 式的等式左侧也会递减。

因此可以在正实数范围内二分 λ ，使用 (3) 式计算 v_i 并代入 (4) 式检验，如果 (4) 式等式左侧大于 E 则增加 λ 的值，否则减小 λ 的值。

∇g 线性相关当且仅当所有 $v_i = v'_i$ 成立，此时特判处理即可。

3.3.2 [CCPC2023 Harbin] Energy Distribution

题目描述 给定 $w_{ij} (1 \leq i < j \leq n)$ ，你需要构造一组 x_i ，要求 $0 \leq x_i \leq 1, \sum_{i=1}^n x_i = 1$ ，求此条件下 $\sum_{i=1}^n \sum_{j=i+1}^n x_i x_j w_{ij}$ 的最大值。

数据范围 $n \leq 10, 0 \leq w_{ij} \leq 1000, w_{ij}, x_i$ 均为实数。

解法分析 令函数 $f(x_1, \dots, x_n) = \sum_{i=1}^n \sum_{j=i+1}^n x_i x_j w_{ij}, g(x_1, \dots, x_n) = \sum_{i=1}^n x_i - 1$ ，则答案即为 $g(x_1, \dots, x_n) = 0$ 且 $0 \leq x_i$ 条件下 $f(x_1, \dots, x_n)$ 的最大值。

如果没有 $0 \leq x_i$ 的限制，显然可以直接使用拉格朗日乘数法求出最大值。存在 $0 \leq x_i$ 的不等式约束时，如果直接使用拉格朗日乘数法，并检查所有满足 $0 \leq x_i$ 的极值点，可能会遗漏最终答案。这是因为拉格朗日乘数法只会找到所有完整定义域中的极值点，此时如果某个点有 $x_i = 0$ ，它在完整定义域中可能不是极值点，但在实际定义域中是一个极值点。（令完整定义域表示没有 $0 \leq x_i$ 的限制的定义域，实际定义域表示存在 $0 \leq x_i$ 的限制的定义域。）

因此，我们可以枚举哪些 x_i 等于 0，并把这些 $x_i = 0$ 加入等式限制中进行拉格朗日乘数法，并使用高斯消元求解。这样能保证求出所有实际定义域中的极值点，取其中最优的

即可得到答案。当高斯消元不存在唯一解时，意味着有自由变量，可以通过调整使得某个 x_i 为 0 且函数值不变，因此可以忽略高斯消元无唯一解的情况。可以发现，只要 x_i 不是全为 0，都有 ∇g 线性无关，故拉格朗日乘数法不会遗漏任何答案。总复杂度 $O(2^n n^3)$ 。

4 拉格朗日对偶

4.1 定义

对于 $f: \mathbb{R}^n \rightarrow \mathbb{R}, g: \mathbb{R}^n \rightarrow \mathbb{R}^m, h: \mathbb{R}^n \rightarrow \mathbb{R}^p$ ，考虑最优化问题

$$\begin{aligned} \max f(x) \\ \text{s.t. } g(x) \geq 0 \\ h(x) = 0 \end{aligned} \quad (5)$$

定义其拉格朗日函数 $F: \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}$:

$$F(x, \lambda, \nu) = f(x) + \lambda^T g(x) + \nu^T h(x)$$

定义其拉格朗日对偶函数 $L: \mathbb{R}^m \times \mathbb{R}^p \rightarrow \mathbb{R}$:

$$L(\lambda, \nu) = \max_x F(x, \lambda, \nu)$$

如果 $F(x, \lambda, \nu)$ 关于 x 的上界不存在，则 $L(\lambda, \nu) = +\infty$ 。

定义其拉格朗日对偶问题为

$$\begin{aligned} \min L(\lambda, \nu) \\ \text{s.t. } \lambda \geq 0 \end{aligned} \quad (6)$$

对于在限制下求 $\min f(x)$ 的问题也可以用类似的方式⁶定义其对偶问题。

4.2 性质

4.2.1 弱对偶性

定理 4.1 (min-max 不等式). 对于 $F: D_x \times D_y \rightarrow \mathbb{R}$ ，有 $\max_{y \in D_y} \min_{x \in D_x} F(x, y) \leq \min_{x \in D_x} \max_{y \in D_y} F(x, y)$ 。

⁶ $\min f(x)$ 的对偶问题可以通过考虑 $\max -f(x)$ 的对偶问题得到。

证明.

$$\begin{aligned} \forall i, j : F(i, j) &\leq F(i, j) \\ \forall i, j : \min_x F(x, j) &\leq F(i, j) \\ \forall i : \max_y \min_x F(x, y) &\leq \max_y F(i, y) \\ \max_y \min_x F(x, y) &\leq \min_x \max_y F(x, y) \end{aligned}$$

定理 4.2 (弱对偶定理). 令 (5) 式最优解的值为 s^* , 其拉格朗日对偶问题最优解的值为 t^* , 则有 $s^* \leq t^*$ 恒成立。

证明. 参考定理 3.1, 我们可以添加拉格朗日乘数来把 $g(x) \geq 0$ 和 $h(x) = 0$ 的限制并入最优化函数中:

$$\begin{aligned} s^* &= \max_{g(x) \geq 0, h(x)=0} f(x) \\ &= \max_x \min_{\lambda \geq 0, \nu} f(x) + \lambda^T g(x) + \nu^T h(x) \\ &\leq \min_{\lambda \geq 0, \nu} \max_x f(x) + \lambda^T g(x) + \nu^T h(x) \\ &= \min_{\lambda \geq 0, \nu} \max_x F(x, \lambda, \nu) \\ &= \min_{\lambda \geq 0, \nu} L(\lambda, \nu) \\ &= t^* \end{aligned}$$

观察这一证明可以发现, 对偶问题正是通过交换 $\min$ 和 $\max$ 的顺序来定义的。

4.2.2 强对偶性

若在定理 4.2 的记号下有 $s^* = t^*$, 则我们称强对偶性成立。

强对偶性并非恒成立, 但在函数具有一些良好的性质时, 会有强对偶性成立, 例如线性规划问题就具有强对偶性。拉格朗日对偶及强对偶性在凸优化问题中有极为重要的应用, 并可以从中延伸出与拉格朗日乘数定理极为相似的 KKT 最优化条件 [6]。但由于这一部分内容与 OI 关系不大, 故此处略去, 具体可以参见 [2]。

不过存在 minimax 定理, 其形式与拉格朗日对偶相似, 且在 OI 中使用较为方便:

引理 4.1. 若 $D_x \subseteq \mathbb{R}^n, D_y \subseteq \mathbb{R}^m$ 为紧致⁷凸集⁸, 连续函数 $f : D_x \times D_y \rightarrow \mathbb{R}$ 关于 x 是凸函数⁹, 关于 y 是凹函数¹⁰, 则存在 $x_0 \in D_x, y_0 \in D_y$, 满足:

$$\min_{x \in D_x} F(x, y_0) = F(x_0, y_0) = \max_{y \in D_y} F(x_0, y)$$

⁷ $\mathbb{R}^n$ 的子集是紧致的当且仅当它是闭集合且有界。

⁸ $\mathbb{R}^n$ 的子集 S 是凸集当且仅当 $\forall p, q \in S, \forall \lambda \in [0, 1], \lambda p + (1 - \lambda)q \in S$ 。

⁹对于凸集 C , $f : C \rightarrow \mathbb{R}$ 是凸函数当且仅当 $\forall p, q \in C, \forall \lambda \in [0, 1], f(\lambda p + (1 - \lambda)q) \leq \lambda f(p) + (1 - \lambda)f(q)$ 。

¹⁰对于凸集 C , $f : C \rightarrow \mathbb{R}$ 是凹函数当且仅当 $\forall p, q \in C, \forall \lambda \in [0, 1], f(\lambda p + (1 - \lambda)q) \geq \lambda f(p) + (1 - \lambda)f(q)$ 。

由于该引理的证明过于困难，故此处不作展示，可参见 [4]。

定理 4.3 (minimax 定理). 若 $D_x \subseteq \mathbb{R}^n, D_y \subseteq \mathbb{R}^m$ 为紧致凸集，连续函数 $f: D_x \times D_y \rightarrow \mathbb{R}$ 关于 x 是凸函数，关于 y 是凹函数，则有：

$$\max_{y \in D_y} \min_{x \in D_x} F(x, y) = \min_{x \in D_x} \max_{y \in D_y} F(x, y)$$

证明. 由引理 4.1 可知，存在 $x_0 \in D_x, y_0 \in D_y$ 满足

$$\max_{y \in D_y} F(x_0, y) = F(x_0, y_0) = \min_{x \in D_x} F(x, y_0)$$

则有

$$\min_{x \in D_x} \max_{y \in D_y} F(x, y) \leq \max_{y \in D_y} F(x_0, y) = F(x_0, y_0) = \min_{x \in D_x} F(x, y_0) \leq \max_{y \in D_y} \min_{x \in D_x} F(x, y)$$

由弱对偶定理可知

$$\max_{y \in D_y} \min_{x \in D_x} F(x, y) \leq \min_{x \in D_x} \max_{y \in D_y} F(x, y)$$

因此有

$$\max_{y \in D_y} \min_{x \in D_x} F(x, y) = \min_{x \in D_x} \max_{y \in D_y} F(x, y)$$

4.2.3 凸性

定理 4.4. 拉格朗日对偶函数 $L(\lambda, \nu) = \max_x F(x, \lambda, \nu)$ 是凸函数。

证明. 对于 $\lambda_1, \lambda_2 \in \mathbb{R}^m, \nu_1, \nu_2 \in \mathbb{R}^p, a \in [0, 1]$ ，有

$$L(a\lambda_1 + (1-a)\lambda_2, a\nu_1 + (1-a)\nu_2) = \max_x F(x, a\lambda_1 + (1-a)\lambda_2, a\nu_1 + (1-a)\nu_2)$$

令其取到最大值时的 x 为 x_* ，则有

$$\begin{aligned} & L(a\lambda_1 + (1-a)\lambda_2, a\nu_1 + (1-a)\nu_2) \\ &= F(x_*, a\lambda_1 + (1-a)\lambda_2, a\nu_1 + (1-a)\nu_2) \\ &= f(x_*) + (a\lambda_1 + (1-a)\lambda_2)^T g(x_*) + (a\nu_1 + (1-a)\nu_2)^T h(x_*) \\ &= a(f(x_*) + \lambda_1^T g(x_*) + \nu_1^T h(x_*)) + (1-a)(f(x_*) + \lambda_2^T g(x_*) + \nu_2^T h(x_*)) \\ &= aF(x_*, \lambda_1, \nu_1) + (1-a)F(x_*, \lambda_2, \nu_2) \\ &\leq aL(\lambda_1, \nu_1) + (1-a)L(\lambda_2, \nu_2) \end{aligned}$$

4.3 例题

4.3.1 [POJ Monthly 2015.5] Min-Max

题目描述 令 $F(x_1, \dots, x_n) = \sum_{i=1}^n \mu_i x_i$ 。给定 p_i, q_i, C ，你需要构造一组 μ_i 满足 $0 \leq \mu_i \leq 1$, $\sum_{i=1}^n \mu_i = 1, F(p_1, \dots, p_n) = C$ ，求出在此限制下 $F(q_1, \dots, q_n)$ 的最大值和最小值。

数据范围 $1 \leq n \leq 50000, |p_i|, |q_i| \leq 1000$ 。

解法分析 以求最大值为例。对限制 $F(p_1, \dots, p_n) = C$ 加入拉格朗日乘数，原问题变为 $\max_{\mu_i} \min_{\lambda} F(q_1, \dots, q_n) + \lambda (F(p_1, \dots, p_n) - C)$ ，其中 $0 \leq \mu_i \leq 1, \sum_{i=1}^n \mu_i = 1$ 。展开 F 后，内层等于 $\sum_{i=1}^n (q_i + \lambda p_i) \mu_i - \lambda C$ 。其关于 μ_i 和 λ 均为线性函数，且 μ_i 的定义域为凸集，因此可以使用 minimax 定理，将答案改写为 $\min_{\lambda} \max_{\mu_i} \sum_{i=1}^n (q_i + \lambda p_i) \mu_i - \lambda C$ 。对于一个确定的 λ ，有 $\max_{\mu_i} \sum_{i=1}^n (q_i + \lambda p_i) \mu_i$ 等于 $\max_i q_i + \lambda p_i$ ，故答案等于 $\min_{\lambda} \max_i q_i + \lambda(p_i - C)$ ，建立凸壳并求解即可。

5 线性规划对偶

5.1 标准型

对于线性规划标准型

$$\begin{aligned} \max_x \quad & c^T x \\ \text{s.t.} \quad & x \geq 0 \\ & Ax \leq b \end{aligned} \quad (7)$$

可以将其视为 (5) 式只有不等式约束的一种特殊情况，定义其拉格朗日函数

$$F(x, y_1, y_2) = c^T x + y_1^T (b - Ax) + y_2^T x$$

其拉格朗日对偶函数

$$\begin{aligned} L(y_1, y_2) &= \max_x F(x, y_1, y_2) \\ &= \max_x c^T x + y_1^T (b - Ax) + y_2^T x \\ &= \max_x (c^T - y_1^T A + y_2^T) x + y_1^T b \end{aligned}$$

其拉格朗日对偶问题 $\min_{y_1, y_2 \geq 0} L(y_1, y_2)$ 等价于

$$\begin{aligned} \min_{y_1, y_2 \geq 0} \quad & y_1^T b \\ \text{s.t.} \quad & y_2^T = y_1^T A - c^T \end{aligned}$$

消去上式中的 y_2 得

$$\begin{aligned} \min_{y_1 \geq 0} & y_1^T b \\ \text{s.t.} & y_1^T A \geq c^T \end{aligned}$$

其与线性规划的拉格朗日对偶形式完全相同，因此线性规划对偶是一类特殊的拉格朗日对偶。

5.2 任意型

对于一个非标准型线性规划问题，如果我们先将其转为标准型再进行常规的线性规划对偶，不仅步骤繁杂，而且可能会在矩阵中增加大量冗余元素，阻碍对问题形式的观察和判断。对于这种情况，我们可以在原问题中加入拉格朗日乘数，并使用 minimax 定理直接进行对偶。这种对偶方法可以很大程度上保留原问题的简洁形式，使得对偶问题的组合意义更加清晰。

以下问题展示了如何对任意的线性规划问题进行对偶。

5.2.1 [AtCoder World Tour Finals 2022 Day1] Welcome to Tokyo!

题目描述 给定 m 对 l_i, r_i ，你需要对于每个 $k = 1, 2, \dots, n$ ，求出：如果在 $1, 2, \dots, n$ 中标记 k 个数，最多能有多少个 i 使得 $[l_i, r_i]$ 中存在被标记的数。

数据范围 $1 \leq n, m \leq 10^6, 1 \leq l_i, r_i \leq n$ 。

解法分析 对于一个固定的 k 考虑问题。令 a_i 表示是否标记 i ， b_i 表示 $[l_i, r_i]$ 中是否存在被标记的数，则可以转化为求以下线性规划问题的整数解：

$$\max \sum_{i=1}^m b_i \quad \text{s.t.} \quad a_i \leq 1, \sum_{i=1}^n a_i = k, b_i \leq 1, b_i \leq \sum_{j=l_i}^{r_i} a_j, 0 \leq a_i, b_i$$

稍作改写得到

$$\max_{0 \leq a_i, b_i} \sum_{i=1}^m b_i \quad \text{s.t.} \quad a_i - 1 \leq 0, \sum_{i=1}^n a_i - k = 0, b_i - 1 \leq 0, b_i - \sum_{j=l_i}^{r_i} a_j \leq 0$$

该线性规划问题的矩阵是全幺模矩阵¹¹，这意味着必然存在所有变量均为整数的最优解，因此可以去除整数性的限制。

对限制加入拉格朗日乘数可得

$$\max_{0 \leq a_i, b_i} \min_{p_i, q_i, s_i \leq 0, \lambda} \sum_{i=1}^m \left(b_i + q_i(b_i - 1) + s_i \left(b_i - \sum_{j=l_i}^{r_i} a_j \right) \right) + \sum_{i=1}^n p_i(a_i - 1) + \lambda \left(\sum_{i=1}^n a_i - k \right)$$

¹¹ 由于此部分与文章主题无关，故不作叙述，可以参见 [5][7]。

使用 minimax 定理交换 max 和 min 并重新整理得

$$\min_{p_i, q_i, s_i \leq 0, \lambda} \max_{0 \leq a_i, b_i} \sum_{i=1}^n \left(p_i + \lambda - \sum_{j|i \in [l_j, r_j]} s_j \right) a_i + \sum_{i=1}^m (1 + q_i + s_i) b_i - \sum_{i=1}^n p_i - \sum_{i=1}^m q_i - \lambda k$$

将 a_i, b_i 视为拉格朗日乘数后得到

$$\begin{aligned} \min_{p_i, q_i, s_i \leq 0, \lambda} & - \sum_{i=1}^n p_i - \sum_{i=1}^m q_i - \lambda k \\ \text{s.t.} & p_i + \lambda - \sum_{j|i \in [l_j, r_j]} s_j \leq 0 \\ & 1 + q_i + s_i \leq 0 \end{aligned}$$

令所有 p_i, q_i, s_i, λ 变为原先的相反数后得到

$$\begin{aligned} \min_{p_i, q_i, s_i \geq 0, \lambda} & \sum_{i=1}^n p_i + \sum_{i=1}^m q_i + \lambda k \\ \text{s.t.} & p_i + \lambda \geq \sum_{j|i \in [l_j, r_j]} s_j \\ & q_i + s_i \geq 1 \end{aligned}$$

注意到 $s_i > 1$ 没有任何意义，因此可以增加 $s_i \leq 1$ 的限制并令

$$p_i = \max \left(0, \sum_{j|i \in [l_j, r_j]} s_j - \lambda \right), q_i = 1 - s_i$$

得到

$$\begin{aligned} \min_{s_i, \lambda} & \sum_{i=1}^n \max \left(0, \sum_{j|i \in [l_j, r_j]} s_j - \lambda \right) - \sum_{i=1}^m s_i + \lambda k + m \\ \text{s.t.} & 0 \leq s_i \leq 1 \end{aligned}$$

当 $\lambda < 0$ 时，在其它变量不变的情况下将 λ 变为 0 不会使答案变得更劣，故可以假定 $\lambda \geq 0$ 。当存在一个 i 满足 $\max \left(0, \sum_{j|i \in [l_j, r_j]} s_j - \lambda \right) > 0$ 时，可以适当减小一些 s_j 的值使得 $\max \left(0, \sum_{j|i \in [l_j, r_j]} s_j - \lambda \right) = 0$ ，且该过程维持答案不变。因此必然存在最优解使得 $\forall i \in [1, n] \cap \mathbb{Z}, \sum_{j|i \in [l_j, r_j]} s_j - \lambda \leq 0$ 。

问题化简为

$$\begin{aligned} m + \min_{s_i, \lambda} & \lambda k - \sum_{i=1}^m s_i \\ \text{s.t.} & \lambda \geq 0, 0 \leq s_i \leq 1, \sum_{j|i \in [l_j, r_j]} s_j \leq \lambda \end{aligned}$$

由之前提到的整数性可知,我们只需考虑此问题的最优整数解。结合 $\lambda = \max_i \sum_{j| i \in [l_j, r_j]} s_j$, 则该问题等价于从 m 条线段选取 t 条进行覆盖, 设所有位置中被覆盖次数最多的那个被覆盖了 λ 次, 最小化 $m + k\lambda - t$ 。

本文的侧重点在于如何对偶而非如何求解对偶后的问题, 故后续做法略去不表, 具体可参考该题的官方题解 [8]。更多关于如何求解线性规划对偶问题的内容可以参见丁晓漫的 IOI2021 国家集训队论文 [9]。

6 纳什均衡

6.1 基本博弈概念

定义 6.1 (策略). 一个参与者¹²的策略表示其在博弈中的全部行为, 也被称为纯策略。一个参与者所有可能的策略构成的集合称为策略集合。

定义 6.2 (博弈结果). 所有参与者的策略构成了一个博弈的结果, 其包含了此次博弈的全部信息。所有可能的博弈结果构成的集合成为该博弈的结果集合。

定义 6.3 (收益). 对于一个博弈结果, 每个参与者有一个实数值与之对应, 称为收益。每个参与者都希望最大化自己的收益, 其从博弈结果对应至收益的函数称为收益函数。

定义 6.4 (静态博弈). 静态博弈是指参与者同时进行决策且只决策一次的博弈。即每个人在决策时都不知道其它人采取了什么行动。

与静态博弈相对的概念是动态博弈, 此时参与者的行动存在先后顺序。如无特殊说明, 接下来描述的所有博弈均为静态博弈。

定义 6.5 (完全信息博弈). 完全信息博弈是指每个参与者都对其他人足够了解的博弈。即每个人都知道其他人的策略空间和收益函数。

与完全信息博弈相对的概念是不完全信息博弈, 如果其为动态博弈则参与者会在过程中获取其他人的信息。如无特殊说明, 接下来描述的所有博弈均为完全信息博弈。

定义 6.6 (非合作博弈). 如果博弈的参与者之间不存在任何有约束力的协议, 并且每个人的决策仅基于自身利益最大化, 那么称该博弈是一个非合作博弈。

与非合作博弈相对的概念是合作博弈, 此时参与者之间会相互合作。如无特殊说明, 接下来描述的所有博弈均为非合作博弈。

¹²如无特殊说明, 后文将不区分参与者与人。

定义 6.7 (混合策略). 参与者在行动时可以给每个纯策略分配一个概率，并保证总概率为 1。这一行动方式称为混合策略。

本文只考虑参与人数为二，每个参与者策略集合有限¹³的完全信息静态非合作博弈，此时博弈可以简化为以下模型：

设两位参与者策略集合的大小分别为 n 和 m ，则双方的收益函数可以用两个 $n \times m$ 的实矩阵 A, B 表示， A_{ij} 和 B_{ij} 分别表示当第一个人选择策略 i ，第二个人选择策略 j 时，第一个人的收益和第二个人的收益。

两位玩家的混合策略分别可以用一个长度为 n 的实列向量 $\mathbf{a}$ 和一个长度为 m 的实列向量 $\mathbf{b}$ 表示，并要求 $0 \leq a_i, b_i \leq 1, \sum_{i=1}^n a_i = 1, \sum_{i=1}^m b_i = 1$ 。此时第一个人的期望收益等于 $\mathbf{a}^T A \mathbf{b}$ ，第二个人的期望收益等于 $\mathbf{a}^T B \mathbf{b}$ 。

定义 6.8 (纳什均衡). 如果博弈的每个参与者都选择了自己的混合策略，且没有玩家能在只调整自己混合策略的情况下增大自己的期望收益，则称当前的混合策略集合与相应的博弈结果构成了纳什均衡（或纳什均衡点）。

我们可以通过几个经典的博弈来更好地理解纳什均衡。

例 6.1 (囚徒困境). 一个监狱中有两名嫌疑犯，他们每人有两个选择：认罪并检举对方和保持沉默。如果两人都检举对方，则各判刑 8 年；如果一人检举对方，另一人保持沉默，则检举对方者立刻出狱，保持沉默者判刑 10 年；如果两人都保持沉默，则各判刑 2 年。两名囚犯无法交流，并且都希望最小化自身刑期。

如果我们用刑期的相反数表示收益，则根据问题描述，有

$$n = m = 2, A = \begin{bmatrix} -8 & 0 \\ -10 & -2 \end{bmatrix}, B = \begin{bmatrix} -8 & -10 \\ 0 & -2 \end{bmatrix}$$

对双方而言，选择检举对方在任何情况下都严格优于选择保持沉默，因此存在唯一的纳什均衡点，即双方都选择检举对方，此时有 $\mathbf{a} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ 。

例 6.2 (性别博弈). 有一对夫妻，丈夫希望看球赛，妻子希望逛街，但他们都想待在一起而不是分开各做各的。具体而言，两个人会分别选择看球赛或者逛街，如果双方选择不一致则均获得 0 的满意度，否则与自己希望相符的一人获得 3 的满意度，另一人获得 2 的满意度。

根据问题描述，有

$$n = m = 2, A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}, B = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

¹³如无特殊说明，接下来描述的所有博弈均为每个参与者策略集合有限的博弈。

该博弈有三个纳什均衡点，分别为 $\mathbf{a} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ ，此时两人收益分别为 3, 2； $\mathbf{a} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$ ，此时两人期望收益分别为 $\frac{6}{5}, \frac{6}{5}$ ； $\mathbf{a} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ ，此时两人收益分别为 2, 3。

注意到第二个纳什均衡点对双方来说最公平，但其期望收益却严格低于另外两个纳什均衡点。

6.2 双人零和博弈的纳什均衡

定义 6.9 (零和博弈). 如果对于任意博弈结果都有所有人的收益之和为 0，则称这一博弈是零和博弈。

对于双人零和博弈而言，如果第一个人获得了 x 的收益，则第二个人的收益必然为 $-x$ 。因此可以只用一个矩阵 A 来描述收益函数，对于双方的混合策略 $\mathbf{a}, \mathbf{b}$ ，有第一个人的期望收益等于 $\mathbf{a}^T A \mathbf{b}$ ，第二个人的期望收益等于 $-\mathbf{a}^T A \mathbf{b}$ 。

定理 6.1. 对于任意一个双人零和博弈，不可能存在两个纳什均衡点，使得参与者在两个纳什均衡点处的期望收益不同。

证明. 反证法。假设双方在第一个纳什均衡点的混合策略分别为 $\mathbf{a}_1, \mathbf{b}_1$ ，在第二个纳什均衡点的混合策略分别为 $\mathbf{a}_2, \mathbf{b}_2$ ，且假设第一位玩家在第一个纳什均衡点的期望收益更高，即 $\mathbf{a}_1^T A \mathbf{b}_1 > \mathbf{a}_2^T A \mathbf{b}_2$ 。

由于纳什均衡的定义是没有玩家可以通过改变自身策略来增加期望收益，因此有

$$-\mathbf{a}_1^T A \mathbf{b}_1 \geq -\mathbf{a}_1^T A \mathbf{b}_2, \quad \mathbf{a}_2^T A \mathbf{b}_2 \geq \mathbf{a}_1^T A \mathbf{b}_2$$

即 $\mathbf{a}_2^T A \mathbf{b}_2 \geq \mathbf{a}_1^T A \mathbf{b}_2 \geq \mathbf{a}_1^T A \mathbf{b}_1$ ，与 $\mathbf{a}_1^T A \mathbf{b}_1 > \mathbf{a}_2^T A \mathbf{b}_2$ 矛盾。故双人零和博弈不可能存在两个期望收益不同的纳什均衡点。

定理 6.2. 双人零和博弈必然存在纳什均衡点。

证明. 设该博弈中两人的策略数量分别为 n, m ，第一个人的收益矩阵为 A 。令 D_a 表示第一个人的所有混合策略构成的集合， D_b 表示第二个人的所有混合策略构成的集合，则有 D_a, D_b 均为凸集。定义函数 $F: D_a \times D_b \rightarrow \mathbb{R}$ 满足 $F(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T A \mathbf{b}$ 。

由引理 4.1 可知，必然存在 $\mathbf{a}_0 \in D_a, \mathbf{b}_0 \in D_b$ ，使得

$$\max_{\mathbf{a} \in D_a} F(\mathbf{a}, \mathbf{b}_0) = F(\mathbf{a}_0, \mathbf{b}_0) = \min_{\mathbf{b} \in D_b} F(\mathbf{a}_0, \mathbf{b})$$

则双方的混合策略 $\mathbf{a}_0, \mathbf{b}_0$ 构成了该博弈的一个纳什均衡。

由上述讨论可知，任意双人零和博弈均存在且仅存在一个期望收益不同的纳什均衡点。因此实际应用中常常把纳什均衡视作双人零和博弈的解。对于 OI 中的双人零和博弈问题，题目中的“双方均采用最优策略”即指代达到纳什均衡的混合策略。

推论 6.1. 对于一个双人零和博弈，令 D_a 表示第一个人的所有混合策略构成的集合， D_b 表示第二个人的所有混合策略构成的集合， A 表示第一个人的收益矩阵， S 表示纳什均衡情况下第一个人的期望收益。则有 $S = \min_{b \in D_b} \max_{a \in D_a} a^T A b = \max_{a \in D_a} \min_{b \in D_b} a^T A b$ 。

证明. 设纳什均衡时双方的混合策略分别为 a_0, b_0 ，由定义可知

$$\max_{a \in D_a} a^T A b_0 = S = \min_{b \in D_b} a_0^T A b$$

$$\min_{b \in D_b} \max_{a \in D_a} a^T A b \leq \max_{a \in D_a} a^T A b_0 = S = \min_{b \in D_b} a_0^T A b \leq \max_{a \in D_a} \min_{b \in D_b} a^T A b$$

根据 minimax 定理有 $\min_{b \in D_b} \max_{a \in D_a} a^T A b = \max_{a \in D_a} \min_{b \in D_b} a^T A b$ ，故结论成立。

此推论在 OI 中非常重要，原本以平衡点形式定义的纳什均衡，现在被转化为一个清晰的式子。 $\min_{b \in D_b} \max_{a \in D_a} a^T A b$ 可以理解为，第一个人先制定一个混合策略，然后第二个人在了解第一个人混合策略的情况下制定自己的混合策略，双方均采用最优策略。此时博弈形式从静态博弈变为了动态博弈，与 OI 中大部分的博弈相同，便于选手进行分析。

此外，该推论还提供了一种把双人零和博弈的纳什均衡问题转化为线性规划问题的方法。第一个人在纳什均衡点的期望收益可以用以下线性规划问题表示：

$$\begin{aligned} & \max t \\ & s.t. \quad x_i \geq 0, \quad \sum_{i=1}^n x_i = 1, \quad \forall i \in [1, n] \cap \mathbb{Z}, \quad \sum_{j=1}^n x_j A_{ji} \geq t \end{aligned}$$

6.3 例题

6.3.1 [THUPC 2023 初赛] 欺诈游戏

题目描述 两个人进行静态零和博弈，双方各有 $n+1$ 种策略，第一个人的收益函数可以用一个 $(n+1) \times (n+1)$ 的矩阵 A 表示，其满足

$$A_{ij} = \begin{cases} \frac{j-1}{2} & i < j \\ 0 & i = j \\ i-1 & i > j \end{cases}$$

求双方的最优混合策略，对 998244353 取模。

数据范围 $1 \leq n \leq 4 \times 10^5$ 。

解法分析 根据推论 6.1, 将此博弈视为双方依次给出混合策略。当第一个人给出了混合策略 $\mathbf{a}$ 后, 第二个人选择各个策略的期望收益可以用一个 $n+1$ 维行向量 $\mathbf{c}$ 表示, 满足 $\mathbf{c} = \mathbf{a}^T \mathbf{A} = \sum_{i=1}^{n+1} a_i A_i$ 。此时第二个人的最优策略即为选择 $\mathbf{c}$ 中的最小值。

因此, 第一个人的目的就是最大化 $\mathbf{c}$ 的最小值。

性质. 如果存在第一个人的混合策略 $\mathbf{a}$, 使得 $\mathbf{c}$ 中所有元素的值相同, 则此策略即为第一个人的最优策略。

证明. 反证法, 假设第一个人存在更优的混合策略 $\mathbf{a}'$, 则有其对应的 $\mathbf{c}' = \sum_{i=1}^{n+1} a'_i A_i$ 满足 $\mathbf{c}' > \mathbf{c}$, 即 $\sum_{i=1}^{n+1} (a'_i - a_i) A_i > \mathbf{0}$ 。令 $\mathbf{t} = \sum_{i=1}^{n+1} (a'_i - a_i) A_i > \mathbf{0}$ 。
找到最大的 i 满足 $a'_i - a_i \geq 0$, 此时有

$$\begin{aligned} t_i &= \sum_{j=1}^{i-1} \frac{i-1}{2} (a'_j - a_j) + \sum_{j=i+1}^{n+1} (j-1) (a'_j - a_j) \\ &\leq \frac{i-1}{2} \sum_{j=1}^{i-1} (a'_j - a_j) + \frac{i-1}{2} (a'_i - a_i) + i \sum_{j=i+1}^{n+1} (a'_j - a_j) \\ &= -\frac{i-1}{2} \sum_{j=i+1}^{n+1} (a'_j - a_j) + i \sum_{j=i+1}^{n+1} (a'_j - a_j) \\ &= \frac{i+1}{2} \sum_{j=i+1}^{n+1} (a'_j - a_j) \\ &\leq 0 \end{aligned}$$

这与 $\mathbf{t} > \mathbf{0}$ 的条件相矛盾。

对于第二个人, 也可以使用上述方法证明类似的结论。

因此考虑 $\mathbf{c}$ 中所有元素值相同的情况, 则可以列出方程

$$\begin{aligned} &\frac{1}{2} a_2 + \frac{2}{2} a_3 + \frac{3}{2} a_4 + \cdots \\ &= a_1 + \frac{2}{2} a_3 + \frac{3}{2} a_4 + \cdots \\ &= 2a_1 + 2a_2 + \frac{3}{2} a_4 + \cdots \\ &= \cdots \end{aligned}$$

结合 $\sum_{i=1}^{n+1} a_i = 1$, 取所有相邻方程作差并递推计算即可。容易发现对于所有 i 有 $a_i \geq 0$ 成立, 因此该 $\mathbf{a}$ 为第一个人的最优混合策略。第二个人的最优混合策略也可以通过类似的方法计算得到。

7 总结

本文围绕拉格朗日乘数,介绍了最优化问题中拉格朗日乘数的处理技巧和拉格朗日乘数法在OI中的应用。围绕拉格朗日对偶,介绍了快速计算线性规划对偶问题的方法, minimax 定理,以及双人零和纳什均衡的部分性质。

关于拉格朗日乘数和对偶,还有许多有趣的性质和结论。由于作者才疏学浅,所以介绍的内容较为简单。这部分内容在OI中的普及度不高,相关的题目也不是很多。作者希望本文能起到抛砖引玉的作用,让更多人知道拉格朗日乘数的处理手段和 minimax 定理,也期待更多相关题目和有意思的拓展出现。

8 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练杨耀良的指导。

感谢父母对我的培养和教育。

感谢学校的栽培,符水波老师,应平安老师,郁庭老师,林乃杰老师,董毅老师的教导和同学们的帮助。

感谢汪直方同学,田俊峰同学与我交流讨论,给我启发。

感谢汪直方同学,房俊哲同学,刘恒熙同学为本文审稿。

感谢其他给予我帮助的老师与同学。

参考文献

- [1] Lagrange Multipliers and The Implicit Function Theorem, https://math.dartmouth.edu/archive/m14f04/public_html/math_14_lagrange.pdf.
- [2] Stephen Boyd and Lieven Vandenberghe, Convex Optimization, Cambridge University Press.
- [3] Minimax theorem, Wikipedia, https://en.wikipedia.org/wiki/Minimax_theorem.
- [4] Younggeun Yoo, Kakutani's Fixed Point Theorem and The Minimax Theorem in Game Theory, <https://math.uchicago.edu/~may/REU2016/REUPapers/Yoo.pdf#:~:text=In%201928%2C%20John%20von%20Neumann%20proved%20the%20minimax,the%20minimax%20theorem%20using%20Brouwer%27s%20fixed%20point%20theorem>.
- [5] Unimodular matrix, Wikipedia, https://en.wikipedia.org/wiki/Unimodular_matrix.
- [6] Karush-Kuhn-Tucker conditions, Wikipedia, https://en.wikipedia.org/wiki/Karush%E2%80%93Kuhn%E2%80%93Tucker_conditions.



- [7] Jian Li, Selected Topics in Optimization, <https://people.iis.tsinghua.edu.cn/~jianli/courses/ATCS2014spring/note1.pdf>.
- [8] Editorial of D - Welcome to Tokyo!, AtCoder, <https://atcoder.jp/contests/wtf22-day1-open/editorial/7106>.
- [9] 丁晓漫, 再探线性规划对偶在信息学竞赛中的应用, 2021 年信息学奥林匹克中国国家集训队论文。





范围修改查询问题相关算法及其应用

杭州学军中学教育集团文渊中学 沈吉灏

摘要

本文总结了范围修改查询问题，介绍了一种操作次数最优的离线范围修改查询算法，以及一些其他解决相关问题的算法及应用。

1 前言

线段树等序列数据结构是一类简洁的算法，可以维护一类序列修改查询的问题，在OI中具有广泛的应用。

但是当我们把修改查询问题从一维问题推广到更高维时，序列数据结构（例如线段树）不再能适应复杂的结构，相对于序列而言增加了不少难度。对于这类问题，我们需要寻求更好且更通用的解决思路。

对此，笔者希望通过本文，介绍一种操作次数最优的范围修改查询算法，以及高维空间修改查询问题的一些其他解决方法。

2 引入

2.1 一个经典问题

从一个经典问题引入：

例题 2.1. 维护一个序列，序列上的每个元素为一个 2×2 矩阵，要求做以下操作：

修改操作：给定 l, r, v ，将区间 $[l, r]$ 中的矩阵乘上矩阵 v 。

查询操作：给定 l, r ，查询区间 $[l, r]$ 的矩阵之和。

上述问题是线段树的经典问题，可以使用带懒惰标记的线段树解决。对于线段树上的每个节点，需要维护区间矩阵和，以及对区间中的矩阵乘的一个懒标记。

现在我们考虑抽象数据结构问题：不考虑具体维护什么信息，考虑在维护的信息满足某些特定条件下的通用问题形式。

对于线段树能维护问题，抽象后的问题是：

维护一个序列，可以用某种数据表示序列上一个区间的信息，可以用某种数据表示序列区间的一次修改，需要满足的性质：

1. 序列上两个区间 $[l, mid], [mid + 1, r]$ 的信息可以 $O(1)$ 合并出 $[l, r]$ 的信息。
2. 对一个区间进行的相邻两次修改操作可以 $O(1)$ 合并为一次修改操作。
3. 如果知道了一个区间的信息，可以 $O(1)$ 计算出这个区间经过一次修改后的信息。

矩阵的乘法满足结合律 $(A \times B) \times C = A \times (B \times C)$ 与分配律 $A \times (B + C) = A \times B + A \times C$ ，但不满足交换律。

在上面的问题中，信息的合并是矩阵加法；而结合律、分配律分别使这个问题能满足 2 和 3 的性质。

2.2 抽象信息的定义

定义 2.1. 定义半群为：

给定一个集合 D ，以及 D 上的一个二元运算： $D * D \rightarrow D$ 。

运算满足结合律： $\forall a, b, c \in D, (a * b) * c = a * (b * c)$ 。

定义 2.2. 定义幺半群为：

给定一个集合 D ，以及 D 上的一个二元运算： $D * D \rightarrow D$ 。

运算满足结合律： $\forall a, b, c \in D, (a * b) * c = a * (b * c)$ 。

有幺元： $\forall a \in D, a * e = e * a = a$ 。

定义 2.3. 定义交换半群为：

给定一个集合 D ，以及 D 上的一个二元运算： $D * D \rightarrow D$ 。

运算满足结合律： $\forall a, b, c \in D, (a * b) * c = a * (b * c)$ 。

运算满足交换律： $\forall a, b \in D, a * b = b * a$ 。

前面的例子维护的信息可以看作半群信息。

可以发现维护的是双半群信息，一个表示区间的信息，一个表示修改的懒标记的信息。

不难发现，只要区间的信息与修改的信息均为半群信息，就可以用线段树维护。

3 离线范围修改查询

3.1 更广的范围

在问题中，修改和询问的范围不一定是序列的区间，而可能为树简单路径，二维平面矩形，高维正交范围，半平面范围，曲线范围等。

初始有 n 个点，一个范围代表了初始点集的一个子集。

可以用一个信息表示范围内点的信息和，用一个信息表示对范围的一个修改。这两个信息满足的条件是：

1. 两个范围的信息 S, T 可以 $O(1)$ 合并成一个信息 $S + T$ 。
2. 对一个范围的相邻的两次修改可以 $O(1)$ 合并成一次修改。
3. 如果知道了一个范围的信息，可以 $O(1)$ 计算出这个范围经过一次修改后的信息。

3.2 问题的形式

初始给出 n 个点的一个点集，每个点有一个初始权值。

每次修改给一个范围以及一个修改的值，表示把范围内的点的权值与修改权值做一个二元运算。

每次查询给一个范围，求范围内的点的权值求和后得到的值。其中点的权值和修改的权值满足双半群的性质。

形式化的问题如下：

给定交换半群 $(\mathcal{D}, +)$ ，半群 $(\mathcal{M}, *)$ ，二元运算符 $*$ ： $\mathcal{D} * \mathcal{M} \rightarrow \mathcal{D}$ 。

其中二元运算符满足：

结合律： $\forall a \in \mathcal{D}, b, c \in \mathcal{M}, (a * b) * c = a * (b * c)$ 。

分配律： $\forall a, b \in \mathcal{D}, c \in \mathcal{M}, (a + b) * c = a * c + b * c$ 。

给定一个初始集合 I_0 ，一个有限集合 $I \subseteq I_0$ ，询问集合 Q 。

I 中每个点有初始权值 $d_0(x) \in \mathcal{D}$ 。

第 t 个操作定义为：

1. 范围修改：给出范围 $q_t \in Q$ 和 $f_t \in \mathcal{M}$ ，如果 $x \in q_t$ 则 $d_t(x) = d_{t-1}(x) * f_t$ ，否则 $d_t(x) = d_{t-1}(x)$ 。
 2. 范围查询：给出范围 $q_t \in Q$ ，设 $d_t(x) = d_{t-1}(x)$ ，答案为 $\sum_{x \in q_t} d_t(x)$ 。
- 用 $n = |I|$ 表示初始权重的个数， m 表示修改查询问题的操作数。

3.3 最优离线范围修改查询算法

下面将介绍一种离线范围修改查询算法，由清华大学李欣隆与蔡承泽提出¹。这个算法可以被简单实现，并且常数较小。可以证明这个算法的操作次数复杂度是最优的。

定义 3.1. 定义等价关系：

假设有一个操作序列 $q_1, q_2, \dots, q_n$ ，定义点 x_1 与点 x_2 的等价关系：若 $R(x_1, x_2) = \forall_{i \in [1, n], x_1 \in q_i \leftrightarrow x_2 \in q_i}$ ，其中 q_i 为第 i 次操作的范围，则称 x_1 与 x_2 等价。

¹可见 [2], <https://qoj.ac/files/ISAAC.2021-paper.100.pdf>

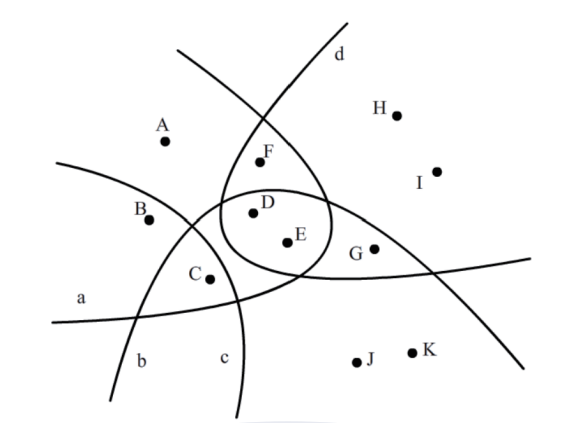


图 1: 被曲线分成的等价类

我们将等价的点归于一个等价类中,可以得到若干个等价类。

假设进行了 $q_l, q_{l+1}, \dots, q_r$ 这些操作, 设划分为的等价类集合为 $R_{l,r}$ 。

如果 $[l', r'] \in [l, r]$, 则 $R_{l,r}$ 比 $R_{l',r'}$ 将点集划为更细的部分, 称这个关系为 $R_{l,r} \subseteq R_{l',r'}$ 。

由于修改操作是给定的, 我们考虑设计一个在修改操作序列上分治的算法。

首先, 对于当前操作序列, 可以分出若干个等价类。如果操作越多, 等价类越细, 最细的划分是有 n 个等价类, 其中 n 为总点数。

我们希望建立一个树形数据结构, 使得满足:

- 叶子节点存储我们需要维护的单点的信息;
- 非叶子节点的建立是为了方便修改和查询而建立的辅助节点, 在非叶子节点上需要存储对应子树内的信息, 以及负责下传给子树内的标记;
- 在访问叶子节点前需访问其所有祖先非叶子节点。

不难发现, 在序列上建立线段树、在树上建立 Top Tree 等经典算法均满足这个结构。

在该问题中, 考虑维护如下的一个动态有根森林:

- 每个节点都对应一个操作序列的某个等价类;
- 每个叶子节点存储的信息为大小为 1 的等价类的信息, 即 n 个初始给定点各自的信息。

设当前操作序列的等价类集合为 R_1 , 那么当前有根森林的每个根节点都对应 R_1 中的一个等价类。

假设我们想将当前等价类集合从 R_1 变为 R_2 , 并满足 $R_1 \subseteq R_2$ (R_2 的等价类个数更少)。



若我们对当前有根森林做若干次合并操作，每次合并若干有根树，就可以将状态 R_1 移动到目标状态 R_2 。合并若干棵以 $x_1, x_2, \dots, x_k$ 为根的有根树的方法是新建一个点 y 作为 $x_1, x_2, \dots, x_k$ 的父亲节点。

同样，我们可以通过撤销状态 R_2 的若干次合并操作（即删除根节点），将其改变为 R_1 的状态。称这两种操作为有根森林之间的移动。

为了支持修改查询操作，考虑在有根森林的非叶子节点上维护额外的域。需要记录两个额外的值，一个存储修改的懒惰标记 $u(x)$ ，一个存储子树的和 $d(x)$ （即为等价类中元素的和）。

在有根森林加点时，我们会新建一个点 y 作为 $x_1, x_2, \dots, x_k$ 的父亲，此时设置 $u(y) = \epsilon_M, d(y) = \sum_{i=1}^k d(x_i)$ ，即可以上传信息。

在有根森林删点时，需要删除点 y 并分裂出 $x_1, x_2, \dots, x_k$ 这 k 棵有根树，此时需要下放点 y 的懒惰标记并删除节点 y 。

如果有一棵以 x 为根的树，其包含的叶子节点恰好是一次修改或查询中包含的点，则 $d(x)$ 即为本次询问的答案，修改则可以对 x 打上标记 $(d(x), u(x)) \rightarrow (d(x) * U, u(x) * U)$ 。

对于原问题，考虑分治。将当前操作序列分成左右两部分。对于两部分操作，由于操作数量更少，拥有的等价类个数也更少，这意味着分治后的两个区间都只需要处理更少的信息。

设有根森林为 n 个叶子节点，没有进行任何合并操作的状态为 R_0 。

设当前需要处理的操作序列为 $q_l, \dots, q_r$ 。

我们先把等价关系从 R_0 变为 $R_{l,r}$ ，假设此时对于所有点，都已经得到了做前 $l-1$ 次操作后的值 $d_{l-1}(x)$ 。然后执行 $[l, r]$ 上的分治算法，这样就得到了做前 r 次操作后的值 $d_r(x)$ ，接下来回退并进行下一个阶段。

当我们需要进行 $q_l, \dots, q_r$ 的分治时，设 $mid = \lfloor \frac{l+r}{2} \rfloor$ ，先将等价类集合从 $R_{l,r}$ 移动到 $q_l, \dots, q_{mid}$ 的状态 $R_{l,mid}$ ，这会在有根森林上加若干个点。处理完 $q_l, \dots, q_{mid}$ 后，将状态回退到 $R_{l,r}$ ，然后移动到 $R_{mid+1,r}$ ，对 $q_{mid+1}, \dots, q_r$ 执行分治算法，再将状态回退到 $R_{l,r}$ 。

当递归到 $l = r$ 时只有一个操作，也就只有 2 个等价类。修改就在需要操作的等价类对应的有根树的根上打上标记，查询则提取该根节点 d 的信息即可。

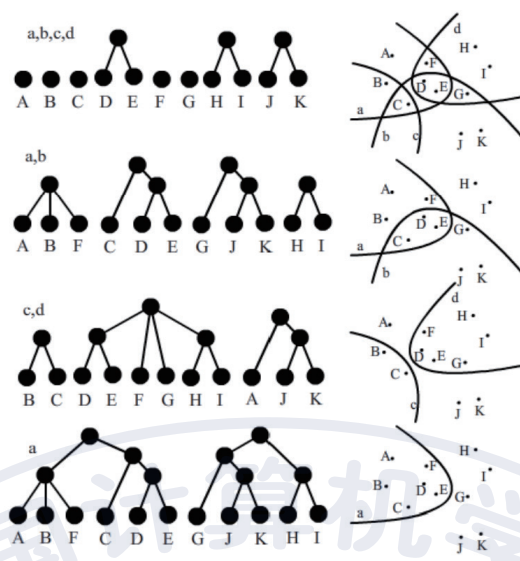


图 2: 等价类的变换

图 2 中, 演示了等价关系为 $R_{1,4}, R_{1,2}, R_{3,4}, R_{1,1}$ 时有根森林的结构。我们从 $R_{1,4}$ 移动到 $R_{1,2}$ 时, 将等价类 $\{A\}, \{B\}, \{F\}$ 合并为 $\{A, B, F\}$; $\{C\}, \{D, E\}$ 合并为 $\{C, D, E\}$; $\{G\}, \{J, K\}$ 合并为 $\{G, J, K\}$ 。从 $R_{1,2}$ 移动回 $R_{1,4}$ 时, 撤销上述合并。递归到 $R_{1,1}$ 时, 需要修改的树即为 a 树, 代表了等价类, 设 x 为 $\{A, B, C, D, E, F\}$ 即 a 树的根, 我们打上修改标记并查询 x 的值。

3.4 算法的复杂度

定义离散函数 F , $F(x)$ 表示 x 个操作最多将点集分成多少个等价类。

不难发现 $F(x)$ 是单调递增的, 且 x 个操作最多将集合 I 分成 $\min(F(x), |I|)$ 个等价类。

定义离散函数 G , $G(n)$ 满足 $F(G(n)) \leq n, F(G(n) + 1) > n$ 。 $G(n)$ 表示在给定范围下, 至少需要多少个操作能把 n 个点分成最细的等价关系, 即 n 个等价类。

假设有 n 个点与 m 次操作, 我们将 m 个操作分成每 $G(n)$ 个操作一组, 在每组中使用上述分治算法, 在代数结构上的运算次数为 $T(n, m) = O(n) + \frac{m}{G(n)} T_1(G(n)), T_1(n) = 2T_1(n/2) + O(F(n))$ 。

在序列上的问题中, 即区间修改查询操作, 有 $F(n) = O(n)$, 此时 $T(n, m) = O(n + m \log \min(n, m))$ 。

在树上路径修改查询操作中, 有 $F(n) = O(n)$, 此时 $T(n, m) = O(n + m \log \min(n, m))$ 。

对于修改查询范围为半平面, 多边形, 圆, 椭圆, 双曲线等情况, 有 $F(n) = O(n^2)$, 此时 $T(n, m) = O(n + m\sqrt{n})$ 。



对于修改查询范围为 d 维的空间范围时, 如 d 维半空间, 单纯形, 球, 正交范围, 多面体等, 其中 $d > 1$, 有 $F(n) = O(n^d)$, 此时 $T(n, m) = O(n + mn^{1-1/d})$ 。

特别的, 若修改查询范围为点集的任意子集, 有 $F(n) = O(2^n)$, 此时 $G(n) = O(\log n)$, 可得 $T(n, m) = O(n + m \frac{n}{\log n})$ 。这意味着修改查询为任意子集可以做到比暴力更优的复杂度。

对于我们的算法, 代数结构上的运算次数和 n, m, I_0, Q, F 有关, 与修改查询的具体范围和具体代数结构无关。

在参考文献[2]中, 证明了这种情况下在代数结构上的运算下界为 $\Omega(n + mn^{1-1/c})$, 即这个算法的操作次数是最优的。

需要注意的是, 该算法是在操作次数上最优的算法, 并不考虑平面或空间中点定位等计算几何部分的复杂度。

3.5 找区域

对于平面修改查询问题, 我们将操作分为一些大小为 $O(\sqrt{n})$ 的阶段, 每个阶段有 $O(n)$ 个区域, 使用平面图点定位的算法, 可以完成等价类的移动。这个算法可以在同复杂度完成圆, 双曲线, 椭圆等范围修改查询。

有一种通用的哈希方法, 如果要移动到 $R_{l,r}$, 就对每个范围随机一个 $[0, 2^w)$ 的整数, 并对于 $q_l, q_{l+1}, \dots, q_r$ 范围进行区间加, 这样属于不同等价类的大概率有不同的哈希值, 而相同等价类有相同的哈希值。对 $R_{l,r}$ 的每个等价类取一个代表元, 然后进行到 $R_{l,mid}$ 和 $R_{mid+1,r}$ 的移动。

问题转换为进行一些范围加之后输出每个点的值, 这里是一个新的但是更简单的抽象数据结构问题。

3.6 应用

例题 3.1 (CTS2021). 测测你的半平面修改查询²

给定交换半群 $(D, +)$, 半群 $(M, *)$, 二元运算符 $*$: $D * M \rightarrow D$, 其中 $*$ 满足结合律与分配律。

给定平面上的 n 个点 (x_i, y_i) , 每个点有初始权值 $d_i \in D$ 。

需要实现 m 个操作:

第 j 次操作给出 a_j, b_j, c_j 与半群信息 $o_j \in M$ 。

对于所有满足 $a_j x_i + b_j y_i < c_j$ 的 i , 需要先回答 $\sum_i d_i$, 后将所有 d_i 修改为 $o_j * d_i$ 。操作离线。

²<https://qoj.ac/problem/9020>, <https://qoj.ac/problem/4817>



在本题中，每次操作范围为半平面。在一个长度为 m 的操作序列下，平面会被划分为 $O(m^2)$ 个区域。

对操作序列每 B 个分一组进行分治，并应用上面的算法，设 $B = \sqrt{n}$ ，可以得到操作次数复杂度为 $O(n + m\sqrt{n})$ 。

接下来我们要处理的问题就是计算几何问题，可以使用平面图点定位的算法。

具体的，对于 $\sqrt{m}$ 个询问的直线，处理出每两条直线的交点，并将交点按照 x 坐标排序后扫描线。扫描线过程中，实时维护出所有直线在当前 x 坐标下，按照 y 坐标排序后的顺序，在遇到一个交点后就交换两条直线的顺序。同时对所有点按 x 坐标排序，在扫描到该点时二分定位出该点属于哪个区域。这样就得到了 $O(m)$ 个区域的初始信息。

对有根森林初始化后，需要处理从 $R_{l,r}$ 移动到 $R_{l,mid}$ 时有根森林的变化，即求出哪些区域会合并。

对于每条直线，将直线上的交点按 x 坐标增大排序，维护一个链表，每个交点属于两条直线的链表。

当删除该直线时，遍历链表上的每个交点，在该交点所属的另一个链表中删除该点，并合并两边的区域。

这样就完成了删直线并合并区域的过程。撤回删掉的直线时，也同样遍历该直线的链表，依次撤回操作即可。

这样做的时间复杂度为 $O(n + m\sqrt{n} \log n)$ ，瓶颈在对交点排序与扫描线二分。不过该题是交互题，只需要保证操作次数正确即可。

例题 3.2 (Ynoi2006). $rmpq^3$

给定含么半群 D ，给出一个平面，平面上每个点对应一个权值 $d(x, y) \in D$ 。

需要实现的操作：

1. 修改操作：给定直线 $x = x_0$ 或 $y = y_0$ 与权值 d_1, d_2 ，将直线一边的所有点 x, y 的 $d(x, y)$ 修改为 $d(x, y) * d_1$ ，另一边的 $d(x, y)$ 修改为 $d(x, y) * d_2$ 。

2. 查询操作：给定 x, y ，查询 $d(x, y)$ 的值。

操作强制在线。

在本题中，每次操作范围为二维正交范围。在一个操作序列下，相当于平面被若干条横竖直线划分成矩形网格，矩形网格中的每个格子为一个等价类。若操作序列长度为 m ，则等价类的个数为 $O(m^2)$ 。

对操作序列每 B 个分一组进行分治。首先分治处理出左一半操作与右一半操作得到的矩形网格与网格中每个格子的信息，合并两个网格时，只需要将水平、竖直的直线各自归并，就不难将儿子网格的每个格子对应到当前网格的格子中，也就完成了等价类的归并。

分治合并的过程等同于从最少的等价类开始，依次构造出每个区间的等价类，即从有根森林的根开始构造，这同时求出了等价类的对应关系。

³<https://www.luogu.com.cn/problem/P7881>



由于强制在线，可以每积累到 B 个散操作时就对这些操作分一块，进行分治处理。进行查询操作时，对前面的 $\frac{m}{B}$ 个块的网格中进行点定位来得到信息，并对最后小于 B 个操作暴力查询。

如果使用二进制分组的技巧，即每次在末尾加入大小为 1 的块，并在末尾两块大小相同，且不超过设定阈值 B 时，不断合并末尾两个块。这样同样做到了分治的效果，并且在散块查询时只需要在 $O(\log B)$ 个块中进行点定位，可以在操作复杂度不变的情况下减小常数。

分治处理一个长为 n 的操作序列的操作次数为 $T(n) = 2T(\frac{n}{2}) + O(n^2) = O(n^2)$ 。设修改与查询操作次数总和为 n ，则取 $B = \sqrt{n}$ 可得操作次数的复杂度为 $O(n\sqrt{n})$ 。

点定位需要在网格的 x 坐标与 y 坐标中进行二分，时间复杂度为 $O(n\sqrt{n} \log n)$ ，在本题中可以通过。可以使用分散层叠的技巧优化到 $O(n\sqrt{n})$ 。

例题 3.3 (Ynoi2007). $TB5x^4$

给定交换半群 $(D, +)$ ，半群 $(M, *)$ ，二元运算符 $*$: $D * M \rightarrow D$ ，其中 $*$ 满足结合律与分配律。

二维平面上初始有 n 个点 (i, p_i) ，其中 p 为一个排列。

需要实现 m 个操作：

每个操作给出 x_1, y_1, x_2, y_2 满足 $0 < x_1 < x_2 < n, 0 < y_1 < y_2 < n$ ，一个点的网格坐标定义为 $((x \geq x_1) + (x \geq x_2), (y \geq y_1) + (y \geq y_2))$ 。

1. 查询网格坐标为 (X, Y) 的点的权值之和，记录在 $ans[X][Y]$ 。
2. 对每个网格坐标为 (X, Y) 的点，进行修改 $o[X][Y]$ 。
3. 所有点的坐标同时发生改变，对于一个网格坐标为 (X, Y) 的点，它的坐标从 (x, y) 变为 $(x + dx[X], y + dy[Y])$ ；其中 $dx[0] = 0, dx[1] = n - x_2, dx[2] = x_1 - x_2$ 。 $dy[0] = 0, dy[1] = n - y_2, dy[2] = y_1 - y_2$ 。

操作离线。

在本题中，每次操作范围为二维正交范围。与上题不同的是，每次操作后，所有点的 x 坐标和 y 坐标会被复合一个排列。

仍然使用上一题的分治合并，从最少的等价类开始，依次构造出每个区间的等价类，即从分治的叶子节点、有根森林的根开始构造。

每个操作对于坐标的修改是一个由三个连续段构成的排列，而由 x 个连续段构成的排列复合一个由 y 个连续段构成的排列最多得到 $x + y$ 段，而且一个段就是一个等价类。

每次合并时，通过左右儿子区间的排列求出当前点的排列，同时可以求出当前排列的每一段（即每个等价类）对应左右儿子的哪个等价类。

建立等价类的对应和有根森林的结构后，再使用上文的分治算法即可完成本题。操作次数与时间复杂度均为 $O(n + m\sqrt{n})$ 。

⁴<https://www.luogu.com.cn/problem/P7723>

4 点集划分树

假设要设计一种点集划分树，使得每次修改查询都只递归到少数子树中，这样就可以支持范围修改查询，并且可以在线。

区间的最优点集划分树为线段树。操作的时间复杂度为 $O(n + m \log n)$ 。

树上简单路径的最优点集划分树为静态 LCT，Top Tree 等。操作的时间复杂度为 $O(n + m \log n)$ 。

二维平面矩形的最优划分树为 KDT。操作的时间复杂度为 $O(n + m\sqrt{n})$ 。

高维正交范围的最优划分树为 KDT。对于 $d(d > 1)$ 维空间，操作的时间复杂度为 $O(n + mn^{1-1/d})$ 。

二维半平面下，Optimal Partition Tree[1] 是最优的二维平面单纯型范围修改查询数据结构，即存在一种划分点集的方法，使得任何二维单纯型可以用 $O(\sqrt{n})$ 个子树所表示的点集的和表示。这个数据结构常数较大并且难以实现。

Optimal Partition Tree 可以拓展到高维单纯型范围上，同时保证理论最优。对于 d 维的问题，操作的时间复杂度为 $O(n + mn^{1-1/d})$ 。

注意在二维半平面下 KDT 作为点集划分树的复杂度错误，可以通过多种构造卡掉，例如将散点构造在一个圆上，做若干条近似圆的切线。

5 半平面范围查询问题

5.1 问题的形式

在该问题中，没有修改操作，只有查询操作，查询为半平面的一侧的交换半群信息。

若可以离线，则使用上述的修改查询算法，就可以达到运算次数的下界 $O(n + m\sqrt{n})$ 。

但在仅有查询，没有修改的情况下，有另一些复杂度可能更低、或可以强制在线的算法。下面将介绍一些针对该问题的算法。

5.2 旋转扫描线

将点集随意划分为大小为 B 的块。

定义半平面的标准型为将询问的半平面先一直向下平移，直到碰到第一个点，之后将询问的半平面一直顺时针方向旋转，直到碰到第一个点。可以证明这样的变换是不会改变这个半平面一侧的点集的。

在这样的变换下，在 B 个点的情况下，只存在 $O(B^2)$ 个不同的半平面的标准型。如果两个半平面变换到了相同的标准型，则这两个平面对应同一个点集，询问的答案相同，即只有 $O(B^2)$ 个需要预处理的答案。

旋转扫描线的过程为：假设当前有一条直线，斜率从 $-\infty$ 逐渐移向 $+\infty$ 。对于一个当前的角度 θ ，维护当前所有点 (x, y) 按照 $x \cos \theta + y \sin \theta$ 排序后的结果；也可看作对于当前斜率 k ，维护所有点按照截距 $y - kx$ 排序后的结果。

取任意一个点当原点，发出一条射线，将所有点垂直投射到射线上。在任意情况下，所有不同的半平面标准型即所有点引向射线的垂线。

对于 B 个点，可以求出它们两两所确定的最多 B^2 条直线。在旋转的过程中，当碰到与某两个点之间的直线斜率相同时，这两个点会交换顺序，此时会多出一个半平面标准型。共有 B^2 次交换，所以共有 B^2 个不同的半平面标准型。

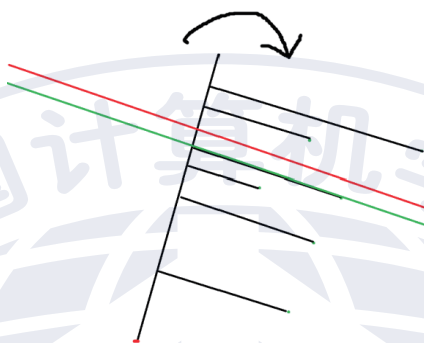


图 3：旋转扫描线

对于一次半平面范围询问，可以在扫描线扫到对应斜率时，在点的序列上二分求出前驱后继。然后相当于要在序列上支持前缀，后缀查询。动态维护当前旋转扫描线的序列，每次交换的元素都是相邻的，可以 $O(1)$ 进行修改。

取 $B = \sqrt{m}$ ，代数结构上的运算次数为 $O(n\sqrt{m})$ ，总时间复杂度 $O(n\sqrt{m} \log n)$ 。时间复杂度的瓶颈在于对斜率的排序以及点定位的二分。

对于分子分母值域为 V 的分数，可以用 $\text{poly}(V)$ 精度的浮点数表示。这样斜率都可以用一个浮点数表示，可以做对两两点对连线的一次极角基数排序，使得排序部分不带 $\log$ 因子，可降低时间复杂度到 $O(n\sqrt{m} \log n)$ 。

注意这里复杂度与修改查询问题的区别：范围修改查询问题的复杂度下界为 $O(n + m\sqrt{n})$ ，不会因为操作多而降低复杂度；而仅支持查询的复杂度可以做到 $O(n\sqrt{m})$ ，会因为操作多而降低复杂度。

若每块预先处理出 B^2 个结果，并用合理的方式二分出每个询问属于哪个半平面标准型，旋转扫描线可以做到在线。

5.3 点集划分树

可以使用上文所述 Optimal Partition Tree 做到达到下界的复杂度，为在线算法。结合旋转扫描线，可以得到一些理论更优秀的做法：



先使用最优点集划分树对点集进行分治，之后在较小的子树上使用旋转扫描线的算法来预处理出所有可能的情况。

设 $B = m^{\frac{2}{3}}n^{-\frac{1}{3}}$ ，对大小 $\leq B$ 的子树，跑旋转扫描线预处理出 $O(m^{\frac{4}{3}}n^{-\frac{2}{3}})$ 个不同的半平面的标准型。在点集划分树上递归查询，在 $\leq B$ 的子树上查询并停止递归。

代数结构上运算次数 $O(n^{\frac{2}{3}}m^{\frac{2}{3}})$ ，总时间复杂度 $O(n^{\frac{2}{3}}m^{\frac{2}{3}}\log^*n)$ ，但难以实现，non-practical。

5.4 半平面莫队

半平面莫队为离线算法，可以将莫队算法扩展到半平面询问上。

例题 5.1 (Ynoi2003). *Helesta*⁵

给定 n 个互不相同的点 (x_i, y_i) , $1 \leq i \leq n$, m 个集合 $S_j = \{(x_i, y_i) \mid A_j x_i + B_j y_i + C_j > 0\}$, $1 \leq j \leq m$ 。

需要找出一个 $1, 2, \dots, m$ 的排列 $p_1, \dots, p_m$ ，使得 $|S_{p_1}| + \sum_{i=2}^m |S_{p_i} \oplus S_{p_{i-1}}| \leq M$ 。

M 是给定的常数， $A \oplus B$ 表示 $(A \cup B) - (A \cap B)$ 。

问题转化为：维护点集，初始为空，支持加点、删点，使得每个半平面代表的点集都在过程中出现至少一次。即构造一个半平面莫队方案，使得加点、删点的次数不超过 M 次。

下面是一个基于随机的期望最优半平面莫队。

随机选取 B 个点，对于每个点，以这个点为中心进行旋转扫描线，这部分代价 $B \times 2n$ ，因为要维护上下两部分。 B 个旋转扫描线都从同一个斜率开始和结束，把它们连起来的代价不超过 n 。

对于每个询问半平面，选取一个代价上离它最近的旋转中心，从该旋转中心的集合移动到它，从而回答这个询问。由于随机撒点，每次的期望移动距离是 $\frac{n}{B}$ ，每个询问的期望代价是 $2\frac{n}{B}$ 。

总代价为 $n + 2nB + 2\frac{mn}{B}$ ，取 $B = \sqrt{m}$ 可以得到代价为 $4n\sqrt{m}$ 。

朴素实现需要对每个询问暴力找出一个最近的点，时间复杂度 $O(m\sqrt{m})$ 。

对于半平面莫队的转移，需要使用半平面数点的数据结构，在本题中并未体现。

5.5 数据随机的特殊情况

若题目保证所有点都在一个规定的矩形内随机生成，或询问通过一些方式随机生成，仍然只有查询半平面一边信息操作，此时我们有一些特殊算法可以解决。

⁵<https://www.luogu.com.cn/problem/P8529>



当数据中点集随机时, KD-Tree 可以近似看作最优点集划分树, 于是上述方法的最优点集划分树可以使用 KDT 来实现。

另一种方法是平面分块:

假设有 n 个点, 将平面划分成 $\sqrt{n} \times \sqrt{n}$ 的块, 因为点集均匀随机, 所以一个块中期望只有常数个点。这样块内询问的期望复杂度为 $O(1)$ 。

半平面可以看做是 $O(\sqrt{n})$ 行的前缀或后缀信息查询, $O(\sqrt{n})$ 次块内查询。可以预处理所有的前缀和后缀信息, 然后暴力查询散点。

代数结构上的运算次数 $O(n + m\sqrt{n})$, 总时间复杂度 $O(n + m\sqrt{n})$ 。

平面分块是运行效率最高的做法。

5.6 应用

例题 5.2 (UNR #4). 己酸集合⁶

给定平面上 n 个点 (x_i, y_i) 。

进行 m 次查询, 每次查询以 $(0, z_i)$ 为圆心, 半径为 R_i 的圆内的点数。

条件可以转化成 $x_i^2 + y_i^2 + z_i^2 - 2y_i z_i \leq R_i^2$, 即 $2z_i \times y_i + R_i^2 \geq x_i^2 + y_i^2$ 。

如果把 (x_i, y_i) 映射到 $(y_i, x_i^2 + y_i^2)$, 则问题转化为询问直线 $y = 2z_i x + R_i^2$ 下的整点个数, 可以用旋转扫描线解决。

例题 5.3 (CTS2022). 袜子⁷

给定 n 个点 (x_i, y_i, c_i) 。

共有 m 次查询操作, 每次查询给定 A, B, C , 问满足 $Ax_i + By_i + C < 0$, $Ax_j + By_j + C < 0$, $c_i = c_j$ 的二元组 (i, j) 的个数。

保证点集随机生成, 即 c_i, x_i, y_i 分别在某个预设的区间内随机选取。

本题为半平面上询问每种颜色点数平方和信息。

Solution 1. 旋转扫描线

将点按点权 c_i 排序, 并按照大小为 B 分块。对于每个询问, 查询每块内的信息 (半平面内第一种点权、最后一种点权的出现次数, 以及中间的点权的出现次数平方和)。发现这种信息是可以合并的半群信息, 将每块的答案信息依次合并得到答案。

使用旋转扫描线处理每块的询问。问题转化为维护点构成的序列, $O(B^2)$ 次交换相邻的位置, 并维护信息。可直接维护每个点当前在序列中的 rank 和当前的点序列。

时间复杂度 $O(\frac{n}{B}(B^2 + m) \log n)$, 取 $B = \sqrt{m}$ 得复杂度为 $O(n\sqrt{m} \log n)$ 。

Solution 2. 点集划分树

将点集和询问的范围反演, 计算点集对询问的贡献。

⁶<https://uoj.ac/problem/553>

⁷<https://www.luogu.com.cn/problem/P8261>



点 (a, b) 在半平面 $cx + dy < e$ 的一侧, 假设 e 非负, 条件可以转化为 $\frac{c}{e}x + \frac{d}{e}y < 1$, 即转化为点 $(\frac{c}{e}, \frac{d}{e})$ 在半平面 $ax + by < 1$ 的一侧。

这样每个询问被转化为一个点, 对所有询问的点建立 KDT。

对于每种颜色, 考虑该颜色对所有询问的贡献。在每个 KDT 的点上维护一个 cnt , 枚举颜色为 x 的点, 对于这个点对应的半平面, 在 KDT 上会访问若干个子树, 对这些子树的 cnt 打上加 1 的标记。

进行完这个颜色的所有操作后, 我们访问刚才访问到过的所有 KDT 结点, 对这些 KDT 结点进行子树答案加。

由于数据随机, KDT 的复杂度是正确的。如果使用最优点集划分树, 时间复杂度为 $O(n\sqrt{m})$ 。

例题 5.4 (Ynoi2000). hpr^8

给定 n 个互不相同的点 (x_i, y_i) 。

共 m 次询问, 每次询问给出 A, B, C , 问满足 $x_i < x_j, y_i < y_j, Ax_i + By_i + C > 0, Ax_j + By_j + C > 0$ 的二元组 (i, j) 的个数。

保证点集数据随机。

本题为半平面支配对计数。

对于询问的半平面的朝向进行分类讨论。首先特判掉 $A = 0$ 或 $B = 0$ 的半平面。

若 $A \times B > 0$, 则半平面在平面直角坐标系上朝向左下方或右上方, 两部分是对称的, 下面只讨论朝向左下方的情况。

设一个点的点权 f_i 为它左下方的点数, 即满足 $x_j < x_i, y_j < y_i$ 的个数。若半平面朝向左下方, 只需要统计半平面内点的 f_i 之和, 为半平面数点问题。

若 $A \times B < 0$, 则半平面在平面直角坐标系上朝向左上方或右下方, 两部分是对称的, 下面只讨论朝向右下方的情况。

设一个点的点权 f_i 为它右下方的点数, 即满足 $x_j > x_i, y_j < y_i$ 的个数。只需要统计半平面内所有的点对数量, 减去半平面内所有点的 f_i 即为答案, 同样为半平面数点问题。

本题数据随机, 因此可以使用 KDT 或平面分块的算法, 其中平面分块运行效率高于其他算法。

6 总结

本文首先介绍了最优离线范围修改查询算法的分治算法, 这种算法有较强的扩展性, 可以解决一类信息符合双半群性质的问题。

本文还介绍了点集划分树、旋转扫描线、平面分块等算法, 这些算法也可以解决一些半平面或高维空间中的问题, 且可以做到更低的复杂度或强制在线。

⁸<https://www.luogu.com.cn/problem/P9996>



由于目前计算机评测规模的原因，在传统题目下有时并不能区分较低复杂度算法和暴力算法的区别。高维空间修改查询问题在当前 OI 竞赛中的出现频率还非常低，其应用还有待进一步发掘。

希望本文能起到抛砖引玉的作用，对 OI 中相关的问题产生更多的启发。

7 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练彭思进、杨耀良的指导。

感谢家人对我的陪伴与支持。

感谢学军中学徐先友、王嘉宏、周邦老师的关心与指导。

感谢李欣隆学长、蔡承泽学长与我交流讨论、给我启发。

感谢学军中学的同学们为本文验稿。

感谢信友队的同学、学长给予我帮助与陪伴。

感谢其他给予我帮助的老师与同学。

参考文献

- [1] Timothy M. Chan. Optimal partition trees. Proceedings of the 26th ACM Symposium on Computational Geometry, pages 1–10, 2010.
- [2] Xinlong Li and Chengze Cai. Offline optimal range query and update algorithm. 2021.

浅谈OI中动态规划问题的一些解题方法

浙江省余姚中学 曹立

摘要

动态规划问题在OI中有着重要地位，且解题思路千变万化。本文列举了动态规划的设计原则，并讨论了在解题实践中具有普遍性、规律性的思考方式与处理手段，结合例题阐释了这些方式与手段的作用与应用场合。

1 动态规划的特征

在信息学中，动态规划（*Dynamic Programming*）是一类通过将原问题拆分成若干子问题，从而递归求解的方法^[1,2]。使用动态规划方法解决的组合问题一般具有以下性质：

- **最优子结构性质**（如果该问题是最优化问题）。即，原问题的最优解可以由子问题的最优解组合而成。这是动态规划方法正确性的必要条件。
- **重叠子问题性质**。即，可能出现的子问题之间存在重复，使得可以通过记录一个子问题的解，避免对其反复求解，从而降低时间复杂度。这使得利用动态规划方法求解比一般的递归与分治方法更优。
- **无后效性**。即，对原问题的拆分求解不能直接或间接地依赖其本身。这保证了最终算法的正常运行。

在信息学竞赛中，有一类问题，其定义了一类具有特定结构特征的组合对象，要求统计这类对象的某种信息（常见的有存在性、最值、和等）。动态规划是解决这类问题的有力手段之一。考虑寻找一种**逐步刻画该类对象**的方式，每一步只决策对象结构的**某个局部的取值**（此处，取值指一个部分所有满足条件的可能性，不是仅指单个变量的取值），那么原问题所规定的对象可以通过一系列步骤的组合得到，而其中一部分步骤组合得到的就是原对象的一个子部分，称为一个**阶段**。

基于该刻画，可以设计出一个具体的动态规划算法，一般包含如下几部分：

- **对子问题结构特征的描述**。即，对于每个阶段，根据该阶段与原问题之间的关系，列出有必要记录的信息。这一般被称为**状态**。

- **子问题的答案。**即，对于每个阶段及对应状态，考虑符合该状态的所有可能子部分取值，记录统计它们得到的信息。这一般被称为**值**。状态和值共同构成完整的**子问题**。
- **子问题之间的关系。**即，通过一个或多个步的决策，从一个阶段转到另一个阶段，这时产生的状态与值的变化。这一般被称为**转移**，或**状态转移方程**。
- **基本情况。**即，对于无法再划分的平凡的子问题，直接确定其值。这一般被称为**初值**。
- **目标情况。**即，原问题本身对应的子问题。这一般被称为**终值**。

将一个复杂模型拆分成若干较简单步骤的组合，这是动态规划在解题应用中的核心思想和重要特征，而这种思想又可以细分为两类：

1. 对于一个子问题，通过决策其对应阶段中的一部分，将问题拆分为若干无关的部分或情况，对应着若干规模较小的子问题。这是**递归**的思想。
2. 对于一个子问题，假定其已解决，通过决策其阶段之外的一部分，得到一个规模更大的子问题，并考虑对后者的值的贡献。这是**增量**的思想。

上述两者都可作为动态规划设计的思考方式，在解题过程中，要根据具体题目模型的不同，选择较易的一种。

2 动态规划的设计

下文将为叙述简洁，将动态规划简称为 **dp**。例题中未特殊说明的变量均为非负整数。受篇幅限制例题题解只列出子问题定义与转移思路，其余细节请读者自行分析。

在解题实践当中，设计 **dp** 的思路过程一般分为两类。

2.1 通过逐步刻画导出状态

首先分析所求对象的结构，构造一个逐步刻画该对象的顺序，根据该顺序，寻找一些刻画的中间状态作为阶段。抽象出这些阶段的代数特征，并确定状态和值的定义，最后导出转移。

该方法在对象结构较简单时应用较合适，例如链、有根树、集合，以及这些结构的一些简单组合等。在设计过程中，要遵循由粗到细，由整体到局部的规律，先确定大致的顺序，再细化处理阶段的选择以及局部多个量之间的决策先后关系，再调整状态与转移的细节。

顺序的选择是至关重要的，不同的选择会导致最终状态数量的差距。可以根据题目对所求的限制、对象结构的特征来推测顺序，也可以直接枚举所有可能的顺序，逐一试错。

例题 2.1.1 (Wooden Spoon¹) 有一棵满二叉树，叶子共有 2^n 个，叶子上写有数，形成一个 1 到 2^n 的排列。每个非叶子上的数为两个儿子上的数的较小值。如果写有数 i 的叶子到根路径上的数全不相同，则称 i 获奖。对于 $i = 1, \dots, 2^n$ ，求有几种情况下 i 获奖，答案对 998244353 取模。

$1 \leq n \leq 20$ 。

如果以自底向上的顺序考虑，那么刻画就是，对于当前有 2^k 个叶子的子树，与另一棵同样大小的，根上数大于当前根上数的子树共同构成一棵有 2^{k+1} 个叶子的树。如果考虑的是数的绝对大小，就必须同时枚举符合获奖的数与目前根上的数，时间复杂度过高；如果只考虑当前子树内数的相对大小，则会得到一个卷积形式的转移。尽管通过 NTT 优化可做到与下一个思路相同的时间复杂度，但细节复杂且常数大，这里略去。

如果以自顶向下的顺序考虑，那么刻画就是，当前点的数与一个儿子的数相同，这个儿子的子树内不可能获奖，直接利用数学方式计算；另一个儿子子树内会出现获奖。因此，每个阶段都只需额外记录当前点上的数即可。

设获奖者所在叶子到根路径上的数依次为 $x_0, \dots, x_n$ (单调递减)，可导出方案数公式：

$$2^n \cdot \prod_{i=1}^n \binom{2^n - x_i - 2^{i-1}}{2^{i-1} - 1} \cdot (2^{i-1})!$$

因此，令 $f_{i,j}$ 表示当前处于路径自顶向下第 $i+1$ 个点，且 $x_{n-i} = j$ ，除了下一个点的子树外其余部分的叶子上的数均确定的方案数，利用后缀和优化，即可做到 $O(1)$ 单次转移。由于状态数为 $O(n2^n)$ ，故时间复杂度也为 $O(n2^n)$ 。

如果对象具有复合结构，那么考虑顺序时也要根据各个组成部分之间的关系与相互作用，确定它们决策的相对顺序，找到状态数较少的阶段。

例题 2.1.2 (MEX counting²) 给定 n, k 与序列 $b_1, \dots, b_n$ ，求序列 $a_1, \dots, a_n$ 的数量，满足 $\forall 1 \leq i \leq n, a_i \in [0, n] \cap \mathbb{Z}, |\text{mex}(\{a_1, \dots, a_i\}) - b_i| \leq k$ 。答案对 998244353 取模。

$1 \leq n \leq 2000, 0 \leq k \leq 50, -k \leq b_i \leq n + k$ 。

由于题目条件中有对前缀 mex 的限制，故容易考虑从左到右逐一确定 a_i 。然而此时发现，由于状态中需记录目前的 mex，故为了使状态关于转移封闭，必须记录各个 $> \text{mex}$ 的已确定数字，从而导致指数级的状态数。

为避免该矛盾，考虑不决策前缀中 $> \text{mex}$ 的部分，当它们真正对 mex 产生影响时再决策。令 $f_{i,j,c}$ 表示已确定前 i 个数，目前 mex 为 j ($|j - b_i| \leq k$)，前面有 c 个数 $> j$ 的方案数。决策时，若 $a_{i+1} = j$ ，则会触发 mex 的增加，此时继续逐一决策先前未决策部分中 $= \text{mex} + 1, \text{mex} + 2, \dots$ 的数，若存在则会导致新 mex 值的连锁增加。

¹CF1784D

²CF1608F



该思路给出了一个 $O(n^3k)$ 的做法，由于连锁增加部分的转移是卷积形式，故至多优化到 $O(n^2k \log n)$ 。考虑将 c 的定义微调成“前面有 c 种不同的数 $> j$ ”，这样即可做到 $O(n^2k)$ 。

从直观的角度看，如果将 a_i 对应到平面直角坐标系中的点 (i, a_i) ，那么正解的各个阶段就是只确定以原点为左下顶点的某个矩形内的点，而该矩形上方的点只有纵坐标种类数被确定。反之，如果仅考虑按一个维度的顺序确定，就会失败。

在确定顺序后，阶段的选取也是多变的。如果选取过多或选取不合适，可能导致状态数过多；如果选取过少，可能出现转移过于复杂，无法优化的情况。

例题 2.1.3 (让热爱能量重新充满人间³) 给定一张 $n+m+1$ 个点， k 条边的无向图，边带正权，点编号为 $0, \dots, n+m$ 。给定 $L, V, c_0, c_1, w_1, \dots, w_n$ 。

要求走一条以点 0 为端点的回路，途中记录两个量：燃料与能量，初始分别为 L, V ，必须时刻非负。要求依次在点 $1, \dots, n$ 表演（可以经过但不表演），在点 i 花费 w_i 的能量。在点 0 可以花费 c_0 的时间同时将燃料和能量补到 L, V ，在 $n+1$ 到 $n+m$ 号点可以花费 c_1 的时间将燃料补到 L 。经过一条权为 t 的边要同时花费 t 的燃料和时间。求最少时间或报告无解。

$$1 \leq n \leq 200, 0 \leq m \leq 50, 1 \leq k \leq \binom{n+m+1}{2}, 1 \leq L \leq 10^6, 1 \leq V, c_0, c_1 \leq 10^9.$$

由于 L, V 均较大，应避免在状态中记录与燃料或能量相关的量。对于能量的限制，只需控制相邻两次在点 0 补能量之间经过的区间即可，于是问题可转化为，对于 $1, \dots, n$ 的所有子区间，求出从点 0 补满并出发依次完成这部分的表演再回到点 0 的最少耗时。

为避免记录剩余燃料，阶段必须选取为补充燃料的时刻，即， $f_{l,r,u}$ 表示，完成了在点 $l, \dots, r$ 的表演，且刚刚在点 u 补充燃料的最少耗时。如果直接转移，会达到 $O(n^3m^2)$ 的时间复杂度，无法通过。

记 $\text{dis}(u, v)$ 表示 u 与 v 间的最短路。 $f_{l,r,u}$ 能直接转移到 $f_{l,r',v} (r' > r)$ 的条件为： $\text{dis}(u, r+1) + \sum_{i=r+1}^{r'-1} \text{dis}(i, i+1) + \text{dis}(r', v) \leq L$ ，这可以通过数据结构优化，因为问题转化为，维护已得到的 f 值，支持将各自的值和目前的距离均加上某个数，求距离不超过某个数的所有 f 值之 $\min$ ，这可以用权值线段树或平衡树模拟。

至于同一对 l, r 内部的转移，可预处理所有点对之间互达的最短时间，逐一枚举转移。这部分是时间复杂度的瓶颈，为 $O(n^2m^2)$ 。

在确定各个阶段后，状态也并不总是显而易见的。要从阶段的本质出发设计描述量，避免设计冗余的量去区分不必区分的子问题情况。在状态过多或转移过复杂的情况下，要变换观察角度，反复微调决策方式与状态定义，得到较优的算法。

例题 2.1.4 (滚榜⁴) 给定 n, m 与序列 $a_1, \dots, a_n$ ，有未知的序列 $b_1, \dots, b_n$ 和 1 到 n 的排列 $p_1, \dots, p_n$ ，满足 $\sum_{i=1}^n b_i = m$ 且 $\forall 1 \leq i < n, b_{p_i} \leq b_{p_{i+1}}, a_{p_i} + b_{p_i} + [p_i < p_{i+1}] \leq a_{p_{i+1}} + b_{p_{i+1}}$ 。求共有几种不同的可能排列 $p_1, \dots, p_n$ 。

³THUSC2023 D1T2

⁴联合省选 2021 D2T2



$$1 \leq n \leq 13, 1 \leq m \leq 500, 0 \leq a_i \leq 10^4。$$

先按照题意考虑按排列的顺序逐一确定 b 值。除了记录目前已确定的下标与 b 之和，为了保证递增关系，还需记录上一个确定的下标与 b 值。状态数已达到 $O(nm^22^n)$ ，无法通过。

从顺序和阶段选取的角度已无优化空间，考虑通过进一步分析模型，减少不必记录的量。考虑排列中相邻的 p_i, p_{i+1} ，则递增关系可变形为 $b_{p_{i+1}} - b_{p_i} \geq \max(a_{p_i} - a_{p_{i+1}} + [p_i < p_{i+1}], 0)$ 。从中得到启发：可以直接决策排名相邻的 b 的差。尽管决策差和决策 b 值本身是等价的，但是这样转换观察角度的却能将状态中的“上一个 b 值”省去。

同时，利用 $\sum_{i=1}^n b_{p_i} = n \cdot b_{p_1} + \sum_{i=1}^{n-1} (n-i)(b_{p_{i+1}} - b_{p_i})$ 容易计算所有 b 之和。此外，如果最终 b 之和 $< m$ ，则可以通过增加 b_{p_n} 使和达到 m 且仍满足其余条件，因此决策时可取差值为其下界而不必枚举。最终状态数为 $O(nm2^n)$ ，时间复杂度为 $O(n^2m2^n)$ 。

例题 2.1.5 (树⁵) 对于 $n = 2, \dots, N$ ，求 n 点有根树有序对 (T_1, T_2) 的数量，满足 T_1 中根为 1，每个点的父亲编号均小于它自身； T_2 中根为 n ，每个点的父亲编号均大于它自身；且同一编号的点在其中一棵树中为叶子，另一棵树中为非叶子。答案对 M 取模。

$$2 \leq N \leq 500, 10 \leq M \leq 2^{30}。$$

很显然只能按编号顺序逐一决策，考虑如何描述一个阶段的情况。对于编号为一段前缀的点，分别考虑两棵树中它们的导出子图。在 T_1 中，它们形成一个含根连通块，并且点被分为三类：有儿子的点、要求有儿子的点但没有儿子、叶子。在 T_2 中，它们形成若干个连通块，均为原树的子树。为了决策，要记录 T_1 中前两类点各自的数量，以及 T_2 中连通块数量。如果新决策的点在 T_2 中非叶子，则需枚举其儿子数。因此状态数为 $O(n^4)$ ，单次转移为 $O(n)$ ，无法通过。

对于 T_1 状态过多的情况，可以利用容斥将“必须有儿子”条件转化为“可以有儿子”；也可以考虑改变描述的思路：只记录一个变量，表示已决策部分有几个点在接下来的决策中还会连儿子。在决策新的点的父亲时，额外决策是否是该父亲的最后一个儿子即可。

对于 T_2 转移复杂的情况，可以仿照例题 2.1.2 的思想，将连通块按父亲的异同关系分成等价类，只记录等价类的数量；也可以改变考虑的角度：既然已决策的点难描述，而未决策的点形成的与 T_1 中已决策的点结构相同，那么可以记录目前对未决策部分的“要求”，然后套用 T_1 部分的转移方法。这实质上是将决策每个点的儿子变为决策每个点的父亲，不过这里的父亲编号是未定的。

令 $f_{i,j,k}$ 表示考虑 1 到 i 号点， T_1 中 1 到 i 号点中有 j 个点有 $i+1$ 到 n 号点中的点作为儿子， T_2 中 $i+1$ 到 n 号点中有 k 个点有 1 到 i 号点中的点作为儿子，这样的方案数。时间复杂度为 $O(n^3)$ 。

⁵ZJOI2022 D1T1



2.2 根据问题依赖导出状态

首先着眼于一个固定的子问题（往往直接选择原问题），寻找其某个局部，并决策该部分取值。根据该局部与剩余部分之间的关系，分析出剩余部分对应的问题，如果能够拆分成多个部分或情况则拆分。递归地分析这些剩余问题，直到当前问题与其依赖的子问题形式相同为止，这是便可以同时确定状态与转移。

在对象结构复杂，条件难处理，或状态定义不显而易见时，该方法可以利用子问题间的转移关系推理出必要的阶段，而不必先确定出完整的逐步刻画顺序。当然，只有选择合适的局部决策，并正确刻画剩余问题的结构，才能使状态数足够少。接下来是三个由易到难的例子。

例题 2.2.1 (Bracket Insertion⁶) 给定 n, p 。对于一个初始为空的括号序列，考虑 n 次如下操作生成一个括号序列：等概率随机选择一个空隙，并向其中插入一对括号。这对括号有 p 的概率为 $()$ ，有 $1-p$ 的概率为 $)($ 。求最终得到合法括号序列的概率模 998244353。

$1 \leq n \leq 500$ 。

首先考虑求出所有生成合法括号序列的插入过程的权值之和，只需将其除以 $(2n-1)!!$ 即为答案。权值定义为插入过程中只考虑随机括号类型的概率。

由定义，最终括号序列合法的条件为：任意前缀中的左括号数都不少于右括号数。

考虑最终括号序列中的第一个括号（显然是左括号），以及与其同时被加入的另一个括号（并不一定是匹配的），将这对括号去除，得到两段子段。显然，任何一对同时被加入的括号不可能分处两段之中。同时，第一段内的括号必须在去除的这对括号之后插入，这两段的括号之间的先后关系也待决策，这都可以通过组合数计算。

于是原问题拆分为：两段都可以由题意中的方法生成，长度和为 $2n-2$ ；且对于第一段，任意前缀中的左括号数都不少于右括号数加 1；对于第二段，任意前缀中的左括号数都不少于右括号数。发现第一段的子问题的条件变化了，形式不同。进一步分析，发现子问题的形式为“长度为 $2i$ 的序列，任意前缀中的左括号数都不少于右括号数加 j 的答案”，记为 $f_{i,j}$ 。 $f_{0,j} = 1$ ，对于 $i \geq 1$ 有转移：

$$f_{i,j} = \sum_{k=1}^i \binom{i}{k} (p \cdot f_{k-1,j+1} + [j > 0](1-p) \cdot f_{k-1,j-1}) f_{i-k,j}$$

其中 j 只需考虑 0 到 $n-i$ 。时间复杂度为 $O(n^3)$ ，可以利用 NTT 优化到 $O(n^2 \log n)$ 。

如果考虑先选取第一对插入的括号作为局部，则序列会被分成三段，通过分析也可以达到同样的子问题与复杂度，但转移更复杂。

例题 2.2.2 (装饼干⁷) 给定 k, x ，有一个可重集合 S ，其中有 a_0 个 2^0 ，……， a_{k-1} 个 2^{k-1} 。

⁶CF1781F

⁷IOI2020 D2T1



要选出一部分数分成 x 组, 使得每组的和都相等, 称 S 可得到这个和。求 S 可得到的数的数量。

$$1 \leq k \leq 60, 1 \leq x \leq 10^{18}, 0 \leq a_i \leq 10^{18}, \sum_{i=0}^{k-1} a_i 2^i \leq 10^{18}.$$

下文提到 y 的数位时均指二进制表示下的数位, 从 0 开始。

对于某个 i , 如果在分配过程中 2^i 使用了超过 x 个, 那么存在一组内有超过一个 2^i , 可以将两个 2^i 等价认为成 2^{i+1} 。因此, 不妨假设所有 a_i 均 $\leq x+1$ 。

性质. 对于一个 y , 它可被得到的充要条件为: 设截取 y 的低 i 位得到的数为 y_i , 则 $\forall 0 \leq i < k, x \cdot y_i \leq \sum_{t=0}^i a_t 2^t$ 。

证明. 必要性显然。

充分性可以类似归纳证明: 设 y 的最高位为 i 。取 S 中尽量大的一些数, 使得和为 $x \cdot 2^i$ 。由于 $x \cdot y_i \leq \sum_{t=0}^i a_t 2^t$, 故这一定是可以做到的。显然可将取出的这些数平均分成 x 份。

若 $y = 2^i$, 则证毕; 否则情况归纳为 $y' = y - 2^i$, 剩余的 $2^0, \dots, 2^{k-1}$ 分别有 $a'_0, \dots, a'_{k-1}$ 个, 满足 $\forall 0 \leq j \leq i, \sum_{t=0}^j a'_t 2^t = \min(\sum_{t=0}^j a_t 2^t, \sum_{t=0}^i a_t 2^t - x \cdot 2^i)$ 。这仍然符合充分条件, 故 y 可被 S 得到。□

根据证明可以得到一个思路: 对于原问题, 枚举 y 的最高位设为 i , 使用与证明相同的方法凑出 $x \cdot 2^i$, 设剩余的数组成集合 S' , 其中各个数分别有 $a'_0, \dots, a'_{k-1}$ 个。待求可能的 $y' = y - 2^i$ 的数量, 就是求 S' 可得到的 $< 2^i$ 的数的数量。

设 S' 中 $\leq 2^i$ 的数最大为 2^j , 已知 a' 在 j 之前的部分均与 a 相等, 且 $j+1$ 到 i 的部分均为 0。若 $j = i$ 或 $a'_{i-1} \geq x$, 则答案即为 S 可得到的 $< 2^i$ 的数的数量; 否则, 若 $a'_j \geq x$, 则 y' 的最高位 $\leq j+1$, 且对于 y' 的最高位 $\leq j$ 的情况, 答案为 S 可得到的 $< 2^{j+1}$ 的数的数量; 否则, 即 $a'_j < x$, 由于 $\sum_{t=0}^j a_t 2^t \leq (x-1)2^j + (x+1)\sum_{t=0}^{j-1} 2^t = x2^{j+1} - x - 1 < x2^{j+1}$, 故 y' 的最高位 $\leq j$, 且对于 y' 的最高位 $< j$ 的情况, 答案为 S 可得到的 $< 2^j$ 的数的数量。

综上所述, 对于“求 S 可得到的最高位为 i 的数的数量”这样一个问题, 可分解为一个“求 S 可得到的最高位至多为 j 的数的数量”和一个特殊问题。这个特殊问题与 S' 有关, 但是形式仍与原问题类似, 故可以继续分解, 并且在 $O(k)$ 次分解后化为基本情况。因此, 令 f_i 表示 S 可得到的最高位至多为 i 的数的数量, 即可以 $O(k^2)$ 的时间复杂度求出最终答案。

由此可见, 在分析过程中, 尽量寻找只与初始给定信息有关, 而与局部决策无关或关系较小的子问题, 这样能使状态数较少。

例题 2.2.3 (最大权独立集问题⁸) 给定一棵二叉树, 点 i 上有权值 d_i 。要求以任意顺序依次删除所有的边, 删除一条边时, 其两 endpoint 权会交换, 且代价增加这两个权值之和。最小化总代价。

$$1 \leq n \leq 5000, 1 \leq d_i \leq 10^9.$$

⁸联合省选 2022 D2T3

下文只分析有 2 个儿子的点；叶子或只有 1 个儿子的点留给读者分析，较简单。对于一个点 u ，设其与父亲的边为 f_u ，与左右儿子的边分别为 l_u, r_u 。不妨设 l_u 总是先于 r_u 交换。

对于根，它与左儿子交换后权值变为某个 d_i ，然后将 d_i 交换给右儿子。左儿子的子问题形如“对于点 u ，父亲权值为输入权值，以某顺序删除子树内边以及 f_u ，最终父亲权值变为 f_i ，求最小总代价”，记为 $f_l(u, i)$ 。右儿子的子问题形如“对于点 u ，父亲权值为 d_i ，以某顺序删除子树内边以及 f_u ，求最小总代价”，记为 $f_r(u, i)$ 。

对于非根点 u ，设左右儿子分别为 v, w ，现分析 $f_l(u, i)$ 。 $f_l(u, i)$ 的分析是对称的，略去。考虑枚举 u 相连的三条边的三种删除顺序：

1. f_u, l_u, r_u 。 d_i 直接与 d_u 交换，子问题形如“对于点 u ，权值为 d_i ，以某顺序删除子树内边，求最小总代价”，记为 $g_l(u, i)$ 。
2. l_u, f_u, r_u 。需枚举 v 交换给 u 的权值 d_j ，因为它会与 d_i 交换产生代价。而由于代价是两数相加，故不必嵌套枚举 i, j 。预处理 $m = \min_j \{f_l(v, j) + j\}$ ，总代价为 $m + d_i + f_l(w, i)$ 。
3. l_u, r_u, f_u 。 u 子树内删完后， d_i 再与 u 的权值交换，子问题形如“对于点 u ，权值为输入权值，以某顺序删除子树内边，最终点 u 权值变为 f_i ，求最小总代价”，记为 $g_r(u, i)$ 。

发现又额外多出两类子问题。现分析 $g_l(u, i)$ 。 $g_l(u, i)$ 的分析是对称的，略去。

如果直接拆分子问题，需枚举 v 交换给 u 的权值 d_j ，且 v 的子问题会同时对初始与最终权值有限制，导致枚举量与状态数均为 $O(n^3)$ 。尝试再讨论 v 的三条边，设其左右儿子分别为 x, y 。

1. f_v, l_v, r_v 。这是简单的，总代价为 $g_l(v, i) + d_i + d_v + f_l(w, v)$ 。
2. l_v, f_v, r_v 。需枚举 x 交换给 v 的权值 d_j ，但仍然不必与 i 嵌套。预处理 d_j 途经 $x \rightarrow v \rightarrow u \rightarrow w$ ，只与 j 有关的部分代价的最优值 $m = \min_j \{f_l(x, j) + d_j + f_l(w, j)\}$ ，再加上只与 i 有关的 $d_i + f_l(y, i)$ 即为总代价。
3. l_v, r_v, f_v 。也只需预处理所有 v 交换给 u 的权值中最优解 $m = \min_j \{g_l(v, j) + d_j + f_l(w, j)\}$ 再加 d_i 即可。

现在所有状态关于转移封闭，时间复杂度 $O(n^2)$ 。

本题中，通过修改“局部”的选取（扩大单次讨论的内容），使得原先必须嵌套枚举的两个变量，利用加法对取 $\min$ 的分配律，拆成互相独立的部分，从而降次。

2.3 小结

首先，在解题实践中，不应拘泥于某种固定的思考方式，只有结合多种思考顺序、角度，才能迅速找到突破口。在很多题目的解题过程中，尤其是处理阶段选取细节和状态设计优化时，结合使用上述两种方法是最高效的。

在以上例题的思考过程中，可以寻找到一类性质的踪迹，这类性质往往是 dp 的标志与突破口，笔者称其为**无关性**。无关性体现在两个方面：



1. 对象的已决策和未决策部分的关系可以用较少的量抽象出来。即，对于所有必要的阶段，其对应的已决策内部的取值在很大程度上不影响整个对象的性质或不与未决策部分产生关系。
2. 多个子问题无关或它们之间的关系可以用较少的量抽象出来。即，这些子问题内部的取值不会或在很大程度上不会互相影响和限制，可以分离求解。

这个性质保证了，在子问题的刻画无需记录已决策部分的完整信息的前提下，符合最优子结构、不重不漏等正确性要求，从而出现重叠子问题，可以降低时空复杂度。若在解题时能敏感地捕捉到这一性质，就能抓住 dp 的契机，获取设计子问题结构的启发。

3 转化问题的技巧

有一些问题并不直接体现出 dp 问题的特征，也无法直接 dp 解决，但通过分析可以将其转化成 dp 能解决的问题。这些分析与转化手段往往随问题模型的特征与性质而变化，需要具体情况具体分析；但也存在一些手段可以较为通用地解决一类问题，接下来的内容介绍了其中较典型的部分手段。

3.1 枚举

通过枚举减少问题需考虑的条件数与信息量，这是非常基础且常见的技巧。

例题 3.1.1 (Infinite Game⁹) 给定字符串 s ，由 a、b 和 ? 组成，须将每个 ? 替换为 a 或 b。考虑基于 s^∞ 的一个过程：重复进行 a 与 b 的比赛，每次取出开头字符并删去，该字符对应一方得 1 分。当有一方得到 2 分时，该方获胜，当前轮比赛结束。令 $r(s)$ 表示，当轮数趋近于无穷时，a 获胜轮数与总轮数之比收敛到的值。求分别有几个 s 满足 $r(s) < 1/2$ 、 $r(s) = 1/2$ 、 $r(s) > 1/2$ 。答案对 998244353 取模。

$1 \leq |s| \leq 200$ 。

首先，只需考虑 a、b 获胜轮数之差与 0 的大小关系，而不必分别考虑两者。

在 s^∞ 进行的比赛必然出现循环节，但循环节可能与 s 本身的循环不同，这是由于一轮比赛可能横跨 s 的结尾与开头。由于比赛中途的比分只有 4 种，故只需考虑对于每种初始比分，完整地在 s 上运行一遍，最终得到的比分，及获胜轮数之差。这样就能用 $O(n^4)$ 的信息量描述一个 s ，为了区分 s 属于哪类情况，考虑初始比分与末尾比分的映射，可以理解成基环内向树，只需将环上的比分情况对应的获胜轮数之差加起来即可判断。

⁹CF1784E



然而状态数还是过多。考虑到只需求环上情况的和，因此先枚举在环上的（共 $2^4 - 1 = 15$ 种），这样状态中只需记录一个差值，以及 4 种比分配的映射（理论上有 $4^4 = 256$ 种，实际只有 67 种），最终强制只考虑符合枚举情况的那些映射状态即可。时间复杂度 $O(n^2)$ 。

3.2 二分

二分是在某个量具有某种单调性时，对枚举的优化。一般用于最优化类型的问题，将问题转化为可以用 dp 处理的判定类型的问题。与 dp 结合的二分常见的有两类。

3.2.1 二分答案

例题 3.2.1 (Assigning Fares¹⁰) 给定一棵 n 点树，给定树上 m 条链。要求给点构造一个正整数点权 $c_1, \dots, c_n$ ，使得每条链上的点权都严格单调递增或递减。最小化点权最大值。

$$2 \leq n \leq 5 \times 10^5, 1 \leq m \leq 5 \times 10^5.$$

如果直接 dp，则单个子树的信息有两项：其根的点权以及子树内的最大点权，这是无法接受的。先二分点权上限 k 。这时，一个点可以取的点权，取决于其各个儿子在必须小于它时可以取的最小值和必须大于它时可以取的最大值。因此考虑令 f_u 表示当 c_u 必须小于其父亲的点权时（如果不存在同时经过这两个点的链则无限制），可以取的最小值。显然另一种情况下可以取的最大值为 $k + 1 - f_u$ 。

对于点 u ，考虑所有同时与 u 在至少一条链内的儿子，它们与 c_u 的额外约束形如：

- 儿子 v ，必须 $c_v < c_u$ ；
- 两个儿子 v, w ，必须 $[c_v < c_u] \neq [c_w < c_u]$ 。

这些可以用带权并查集或种类并查集处理。最终对 c_u 的约束形式包括上、下界，以及某些 $[l, r] \cup [k + 1 - r, k + 1 - l]$ 。将它们取交即可求出 f_u 。

3.2.2 wqs 二分

wqs 二分优化的问题一般形如：有一个上凸函数 $f(x)$ ，给定 a ，希望求得 $f(a)$ ，但无法快速求。目前 OI 中多使用如下方法：通过二分斜率 k ，较快地求 $\max_x \{f(x) - kx\}$ 以及取到该值的最小或最大的 $x = x_0$ ，根据 x_0 与 a 的大小关系调整 k ，最终“强制”上界在 $x = a$ 处取到，算出答案。下凸求 min 的情况是对称的，接下来只讨论上凸的情况。

但在实际应用中，该方法往往会面临一个困难：由于要求出取到最值的 x_0 ，就需要根据 dp 求得的信息反过来还原方案，这无疑增加了思考与代码量。在嵌套 wqs 二分时，会出现正确性上的问题，且难以弥补。

¹⁰CF1322F

这里引入一个解决方法。

性质. 对于上凸的 $f(x)$, 令 $g(k) = \max_x \{f(x) - kx\} + ka$. $g(k)$ 是下凸的, 且最小值为 $f(a)$ 。

证明. 令 $h(k)$ 为使 $g(k)$ 里取到 $\max$ 的任一 x . $\forall k_1, k_2, \forall \lambda \in (0, 1)$, 令 $k_m = \lambda k_1 + (1 - \lambda)k_2$:

$$\begin{aligned} & \lambda g(k_1) + (1 - \lambda)g(k_2) \\ &= \lambda \max_x \{f(x) - k_1 x\} + (1 - \lambda) \max_x \{f(x) - k_2 x\} + k_m a \\ &\geq \lambda (f(h(k_m)) - k_1 h(k_m)) + (1 - \lambda) (f(h(k_m)) - k_2 h(k_m)) + k_m a \\ &= f(h(k_m)) - k_m h(k_m) + k_m a \\ &= g(k_m) \end{aligned}$$

显然 $\forall k, g(k) \geq f(a)$, 又由于 f 的凸性, 一定存在一个 k 使得 $g(k) = f(a)$, 故得证。□

于是, 问题转化为找 $g(k)$ 的极值点。这可以用三分或二分处理。关于该方法进一步的解释详见参考文献 [3]。

与 wqs 二分相关的许多问题, 其暴力解法或优化后的解法就是 dp。

例题 3.2.2 (Aliens¹¹) 给定一个 $m \times m$ 网格上的 n 个格子。给定 k , 要求选出至多 k 个正方形子网格, 使得正方形的主对角线与整个网格的主对角线重合, 且所有给定格子均被至少一个正方形覆盖。最小化所有正方形并集的格数。

$$1 \leq k \leq n \leq 10^5, 1 \leq m \leq 10^6。$$

首先, 可以将问题转化为: 有 n 个 $[1, m]$ 的子区间 $[l_1, r_1], \dots, [l_n, r_n]$ (称为给定区间), 要求选出至多 k 个区间 (称为选择区间), 能包含所有给定区间。所有端点均为整数。显然如果一个给定区间包含于另一个给定区间, 那么前者不必考虑, 故不妨设给定区间的左右端点同时递增。同理选择区间的左右端点也同时递增。

继续分析解的结构, 给定区间被分为至多 k 段, 每段被一个选择区间包含。令 $r_0 = 0$, 第 x 到第 y 个区间作为一段的代价为 $w(x, y) = (r_y - l_x + 1)^2 - [r_{x-1} \geq l_x](r_{x-1} - l_x + 1)^2$ 。减号后的是为了去除与上一段重合的部分。

令 $f_{c,i}$ 表示前 i 个给定区间, 分至多 c 段的最优解。特殊地, $\forall i \geq 1, f_{0,i} = \infty$ 。

$$\text{转移为 } f_{c,i} = \min_{j=1}^i \{f_{c-1,j-1} + w(j, i)\}。$$

性质. $\forall 0 \leq i \leq n, f_{c,i}$ 关于 c 下凸。

证明. 由于 $\forall x_1 \leq x_2 \leq y_1 \leq y_2, w(x_1, y_2) + w(x_2, y_1) - w(x_1, y_1) - w(x_2, y_2) = 2(r_{y_2} - r_{y_1})(l_{x_2} - l_{x_1}) \geq 0$, 故 w 满足四边形不等式。由相关性质, f 也满足四边形不等式 $\forall c_1 \leq c_2, i_1 \leq i_2, f_{c_1,i_1} + f_{c_2,i_2} \leq f_{c_1,i_2} + f_{c_2,i_1}$ 。

¹¹IOI2016 D2T2

现归纳证明原命题：

$\forall c \geq 0, f_{c,0} = 0$ ，符合命题。

对于某个 $i \geq 1$ ，假设 $\forall 0 \leq j < i, f_{c,j}$ 关于 c 下凸。显然 $2f_{1,i} \leq f_{0,i} + f_{2,i}$ 。对任意 $c \geq 2$ ，设 $f_{c-1,i}$ 的最优决策（若有多个取最小，下同）为 p ， $f_{c+1,i}$ 的最优决策为 q 。由相关性质， $p \leq q$ 。

$$\begin{aligned} & f_{c-1,i} + f_{c+1,i} - 2f_{c,i} \\ &= f_{c-2,p-1} + w(p,i) + f_{c,q-1} + w(q,i) - 2f_{c,i} \\ &\geq f_{c-2,p-1} - f_{c-1,p-1} + f_{c,q-1} - f_{c-1,q-1} \\ &\geq f_{c-1,p-1} - f_{c,p-1} + f_{c,q-1} - f_{c-1,q-1} \\ &\geq 0 \end{aligned}$$

故 $f_{c,i}$ 关于 c 下凸。 □

这给出了一个基于 dp 转移方程的易理解的证明。关于上文中提到的“相关性质”，以及不基于转移方程的证明，详见参考文献 [4, 5]。

因此可以先 wqs 二分，这时状态中就无需记录分段数。最终的 dp 可以用二分队列或斜率优化优化到 $O(n \log n)$ 或 $O(n)$ ，总时间复杂度 $O(n \log n \log m)$ 或 $O(n \log m)$ 。

3.3 容斥

利用容斥转化计数问题，这是一类常见且技巧性较强的方法。由于已有较多介绍复杂容斥的文章，这里就不展开详述了。

3.4 针对一类计数问题的转化手段

有这样一类计数问题：定义两类对象 X, Y ，以及一个从 (X, Y) 到 $\{0, 1\}$ 的映射 A ，求有几个合法的 $x \in X$ 满足 $\exists y \in Y, A(x, y) = 1$ 。 x 是被计数的对象，但不容易直接计算； y 可以理解成一种“解”或“证书”，往往能求出 $A(x, y) = 1$ 的 x 个数，但如果对于所有的 y 直接对这个个数求和则会导致重复计算。针对这类问题，常见的有以下几种转化思路。

3.4.1 转化判定方式

通过改变观察角度、发掘性质等手段。将 x 合法的条件转化为更简单的等价表述，使得可以直接 dp 求解。

例题 3.4.1 (Red and Blue Chips¹²) 给定 n 个筹码，一个一堆，颜色为红或蓝，第 n 个一定为蓝色。对于 $i = 1, \dots, n-1$ ，依次将第 i 个筹码所在的堆整体放到后面某个堆的顶部，要

¹² AGC064D

求后者底部的筹码必须为蓝色。求最终得到的一堆筹码自底向上看形成的颜色序列有几种。答案对 998244353 取模。

$$2 \leq n \leq 300。$$

考虑将题目条件转化成更简洁的静态刻画。考虑第 $i+1$ 到 n 个筹码形成的某个序列，现加入第 i 个筹码，并决策其放到哪里，考虑其对最终序列的影响。

这样的过程可以描述为：初始有一个蓝色筹码，每次可将一个颜色给定的筹码插入一个蓝色筹码之后。用数学语言描述，设目前每个蓝色筹码后紧邻的连续红色筹码数依次为 $x_1, \dots, x_k$ 。插入红色筹码就相当于选一个 x_i 加一，插入蓝色筹码就相当于在 x 序列开头以外的地方插入一个 0。

设蓝色筹码有 m 个，输入序列中倒数第 $i+1$ 个蓝色筹码后有 l_i 个红色筹码 ($l_m = n - m$)。最终序列 $x_1, \dots, x_m$ 可以被得到的充要条件为：

- $x_m \geq l_1$;
- $x_m + x_1$ 到 x_{m-1} 中最大值 $\geq l_2$;
- $x_m + x_1$ 到 x_{m-1} 中最大值和次大值之和 $\geq l_3$;
- 以此类推，直到 $\sum_{i=1}^m x_i = l_m$ 。

满足上述条件的序列个数即为答案。除去 x_m ，按值从大到小决策 $\{x_1, \dots, x_{m-1}\}$ 的多重集合。令 $f_{i,j,k}$ 表示 $x_m + x_1$ 到 x_{m-1} 中前 i 大的和为 j ，且这前 i 大的数均 $> k$ 的方案数。转移时枚举等于 k 的个数，乘组合数计算排列数。易分析得时间复杂度为 $O(n^3)$ 。

3.4.2 带权计数

既然对 y 进行 dp 相对容易，那么可以考虑设计一个“权值映射” $w(y)$ ，使得 $\forall x \in X$ ， $\sum_{y \in Y} A(x, y)w(y) = \bigvee_{y \in Y} A(x, y)$ ，这样只需求出 $\sum_{y \in Y} (w(y) \cdot \sum_{x \in X} A(x, y))$ 即可。

一种设计思路是，对于 $A(x, y) = 1$ 的所有 y ，只选择其中一个的 $w(y) = 1$ ，其余为 0。这可以理解为钦定一个“代表元”。

例题 3.4.2 (Range Argmax¹³) 给定 m 个 $[1, n]$ 的子区间 $[l_1, r_1], \dots, [l_m, r_m]$ 。对于一个 1 到 n 的排列 $p_1, \dots, p_n$ ，生成一个序列 $x_1, \dots, x_m$ ，其中 x_i 是 $p_{l_i}, p_{l_i+1}, \dots, p_{r_i}$ 中最大值的位置，记 $F(p) = x$ 。求可能的 $x_1, \dots, x_m$ 个数模 998244353。

$$2 \leq n \leq 300, 1 \leq m \leq \binom{n}{2}。$$

对于一组可能的 $x_1, \dots, x_m$ ，观察所有 $F(p) = x$ 的排列 $p_1, \dots, p_n$ ，将关注点集中到 n 所在的位置。

¹³AGC056B

显然必须存在位置 i 满足 $\forall [l_j, r_j] \ni i, x_j = i$, 称 i 为极大位置。并且注意到对于任意极大位置 i , n 一定可以处于 p_i , 这是因为如果有一组 $F(p) = x$ 的排列 $p_1, \dots, p_n$ 但 $p_i \neq n$, 找到 j 满足 $p_j = p_i + 1$, 可以交换 p_i 与 p_j 使 $F(p)$ 不变。不断这样交换最终就可以使 $p_i = n$ 。

因此不妨钦定最靠左的极大位置 i 使 $p_i = n$, 也就是说对于原问题, 枚举 i , 规定所有包含 i 的区间的 x 均为 i , 这样就可以把它们去掉, 剩余两侧两个无关的子问题。

设包含 i 的区间的最小左端点为 t (不存在则为 i), 对于左侧的子问题, 其极大位置必须 $\geq t$, 否则与“最靠左”矛盾。令 $f_{l,r,t}$ 表示只考虑 $[l, r]$ 的子区间, 最靠左的极大位置必须 $\geq t$ 的答案。时间复杂度 $O(n^3)$ 。

另一种设计思路是, 令某些 $w(y) < 0$, 这可以理解为容斥。例如 $Y = \{S \mid S \subseteq \{1, \dots, n\}\}$, 且对于合法的 x , 使 $A(x, y) = 1$ 的所有 y 恰好是某个集合的所有非空子集。这时就可以令 $w(y) = (-1)^{|y|-1}$ 。运用这类方法的例子包括机器人游戏¹⁴和希望¹⁵。

3.4.3 对判定过程 dp

考虑判定某个 x 是否合法的一个判定算法, 一般将 x 逐步输入给该算法, 算法会存储一些变量用于判定。将该算法抽象成一个 DFA¹⁶, 变量的可能中间状态即为自动机的状态, 根据输入作出的反应即为自动机的转移。这样, 只需在状态中记录 x 已确定了哪些, 以及目前所在自动机的状态即可直接对 x 进行 dp。这种设计思想在其他类型 dp 问题中也有出现, 例如 The Maximum Prefix¹⁷。

最简单的情况是只需额外记录一个布尔量表示目前已知 x 是否合法。若问题复杂, 其判定算法本身也是 dp, 这就是所谓的 dp 套 dp。也存在判定算法为贪心或其他过程的情况。

并非所有计数 dp 都存在状态数较少的判定算法。有些问题的 DFA 规模较大, 但可以通过一些手段缩小。这些手段包括: 利用程序忽略不可能达到和后续全部无法被接受的状态、合并状态等价类, 以及通过组合意义分析不必区分的状态并优化。关于 DFA 的最小化详见参考文献 [6]。

例题 3.4.3 (Phone Numbers¹⁸) 给定一个长度为 n 的仅包含 $1 \sim 9$ 的串 S , 求有几种满足相同限制的串 T , 使得存在一种划分方案, S 与 T 均按该方案划分后, 各个对应段数字集合分别相等。各段内的数字不能重复, 且在 $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ 中恰好形成一个长宽 ≤ 2 的子矩形。答案对 $10^9 + 7$ 取模。

$$1 \leq n \leq 10^5.$$

¹⁴NOI2021 D2T3

¹⁵十二省联考 2019 D2T3

¹⁶确定性有限状态自动机

¹⁷CF1810G

¹⁸USACO 2022 February Contest, Platinum, Problem 3

考虑如何判定。令 f_i 表示 1 到 i 的前缀是否存在一种满足条件的划分方案, f_i 根据 T 的情况从 $f_{i-1}, f_{i-2}, f_{i-4}$ 转移。逐位确定 T , 所需记录的包括 $f_{i-3}, f_{i-2}, f_{i-1}, f_i, T_{i-2}, T_{i-1}, T_i$, 至多有 $(2^4 - 1) \times 9^3 = 10935$ 种可能性, 状态数过多。

注意到仅当 $f_{i-3} = 1$, 且 $\{S_{i-2}, S_{i-1}, S_i, S_{i+1}\}$ 为四种合法集合之一, 且 $\{T_{i-2}, T_{i-1}, T_i\} \subset \{S_{i-2}, S_{i-1}, S_i, S_{i+1}\}$ 时, 才有必要记录 T_{i-2} , 这时 $\{T_{i-2}, T_{i-1}, T_i\}$ 只有 24 种可能; 否则仅当 $f_{i-2} = 1$, 且 $\{S_{i-1}, S_i, S_{i+1}, S_{i+2}\}$ 为四种合法集合之一, 且 $\{T_{i-1}, T_i\} \subset \{S_{i-1}, S_i, S_{i+1}, S_{i+2}\}$ 时, 才有必要记录 T_{i-1} , 这时 $\{T_{i-1}, T_i\}$ 只有 12 种可能; 否则仅当 $f_{i-1} = 1$ 时才有必要记录 T_i 且可以只记 4 种。于是状态数可缩小至百级, 可以通过。

随着状态数的优化, 实现细节会不断增加, 考虑直接用程序算出最小 DFA, 注意必须将 S_{i-2}, S_{i-1}, S_i 也记入状态, 外层 dp 转移时只枚举当前 S 对应的部分状态即可。实际情况为, 对于所有 (S_{i-2}, S_{i-1}, S_i) , 可能达到的本质不同的有效状态共有 507 种; 对于确定的 (S_{i-2}, S_{i-1}, S_i) , 状态至多 32 个, 平均 10.8 个; 对于确定的 $(S_{i-2}, S_{i-1}, S_i, S_{i+1})$, 转移至多 83 条, 平均 16.0 条。该解法的代码详见 <https://loj.ac/s/1977851>。

另外, 本题也存在常数优秀的容斥思路解法¹⁹。

例题 3.4.4 (黄焖鸡²⁰) 给定集合 $A_1, \dots, A_n \subseteq \{1, \dots, m\}$, 求有几个序列 $a_1 (\in A_1), \dots, a_n (\in A_n)$, 其任意非空子区间的最大权独立集²¹的权值都不等于全体和的一半。答案对 998244353 取模。

$1 \leq n \leq 200, 1 \leq m \leq 19$ 。

对于单个序列的判定, 令 f_0, f_1 分别表示目前结尾不选或选, 选的数字减不选的数字的最大值。若 $\max(f_0, f_1) = 0$ 则不合法, 而且显然其不可能 < 0 。每次结尾加入数 a , 转移到 $f'_0 = \max(f_0, f_1) - a, f'_1 = f_0 + a$ 。

对于原问题, 逐一决策 a_i , 并维护以当前数为右端点, 所有左端点对应的区间的 (f_0, f_1) 。显然无法通过, 且状态数过多以至于无法用机械化方法求最小 DFA。

对于一组 (f_0, f_1) , 如果无论接续的序列怎样, 它都不会不合法, 那就可以忽略它。设接续的序列奇数项减偶数项为 d , 那么如果 $\forall d, \max(f_0 + |d|, f_1 + d) > 0$ 则可忽略。当 $f_0 > 0$ 或 $f_0 + f_1 > 0$ 时该条件成立, 否则加入一个 $-f_0$ 就造成了不合法。

从而只需考虑 $f_0 \leq 0$ 且 $f_0 + f_1 \leq 0$ 的情况, 另易证有性质 $f_0 + f_1 \geq 0$ 和 $f_0 \geq -m$, 因此所有需要考虑的情况仅有 $(f_0, f_1) = (-1, +1), \dots, (-m, +m)$, 状态数为 2^m 。令 $g_{i,s}$ 表示已决策前 i 个数, 目前存在的需要考虑的 (f_0, f_1) 的 f_1 形成了集合 s 的答案。转移时, 新的 a 要满足 $a \notin s$, 新的集合 $s' = \{a - x \mid x \in s, x < a\} \cup \{a\}$ 。注意到 s 中所有 $> a$ 的数不影响转移, 故可以对它们任取的所有状态求和, 时间复杂度优化为 $O\left(n \sum_{a=1}^m 2^{a-1}\right) = O(n2^m)$ 。

¹⁹<https://loj.ac/s/1402037>

²⁰CTT2023 D3T3

²¹和最大的子序列, 不能同时选择相邻两个元素



4 总结

动态规划问题的解题过程可以大致分为观察性质、转化模型、设计 dp、优化 dp 四步。这四步互相联系、影响，形成一个有机整体。限于篇幅，本文只是列举了二、三两步的一部分方法，可谓是冰山一角，对于更全面的总结可参考文献 [7]。

限于笔者水平，本文仅给出了一个模糊、粗略的框架。解题没有绝对的万能药，但具体情况具体分析时也并非毫无章法。作者希望本文起到抛砖引玉的作用，能启发读者在解题实践中领悟，总结更多具有一定适用范围的方法，去寻找算法设计中的“思想珠玑”。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢王欣波老师、诸一行老师对我的指导。

感谢母亲对我的培育，感谢家人对我的关心与支持。

感谢常瑞年学长、朱睿哲同学、林睿同学与我讨论本文相关主题的内容，给我启发。感谢常瑞年学长、朱睿哲同学为本文提出的宝贵建议。

感谢训练道路上所有同学与家长给我的帮助与陪伴。

参考文献

- [1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 动态规划. In 算法导论第 3 版, pages 204–236. 机械工业出版社, 2009.
- [2] Dynamic programming. Wikipedia, https://en.wikipedia.org/wiki/Dynamic_programming.
- [3] Oleksandr Kulkov(adamant). Theoretical grounds of lambda optimization. Codeforces, <https://codeforces.com/blog/entry/98334>, 2021.
- [4] 四边形不等式优化. OI Wiki, <https://oi-wiki.org/dp/opt/quadrangle/>.
- [5] 郭羽冲. 浅谈函数的凸性在 OI 中的应用. 2023 年信息学奥林匹克中国国家队候选队论文, 2023.
- [6] 徐哲安. 浅谈有限状态自动机及其应用. 2021 年信息学奥林匹克中国国家队候选队论文, 2021.
- [7] 曹立. dp 题方法总汇. 洛谷, <https://www.luogu.com.cn/blog/yeah-potato/dp-ti-fang-fa-zong-hui>, 2023.

浅谈一些信息合并的处理方式

重庆市巴蜀中学 冯政玮

摘要

信息合并和信息学竞赛中是非常重要的部分，有许多种信息合并的处理方式，本文将从朴素信息合并，结构信息合并共两个角度简要探讨一下在算法竞赛中常见的一些信息合并具体问题中的处理方式与技巧。

本文尝试从朴实与实用的角度介绍一些笔者在算法竞赛经历中遇到的一些有特点的例子与处理方式，介绍经典结构上的一些不同的处理手段和分析方式，希望对读者有所启发。

1 前置知识

本文中将 $\log_2 n$ 简写为 $\log n$ ，表示的是以 2 为底数的对数函数。

信息一般指在具体问题中需要维护的东西，它可以是一个权值，一个节点，一条边，一个式子等。

而通常信息合并的过程指将信息类比为集合间的元素，将若干信息按照某些方式并在一起，维护一个信息结构，使得可以在结构中快速查找到想要的信息，不同的信息对应着不同的处理方式。

2 朴素信息合并

顾名思义朴素信息合并是用较为暴力的合并信息的方式。加以合适的分析后，可以说明其维护方式的正确时间复杂度。

具体而言，我们将介绍两种较为朴素的信息合并思路与维护方式，并简要分析其复杂度并给出一些例子。

2.1 启发式合并

启发式合并是一种信息合并方式，通常形如有若干个信息集合，每个集合中有一些信息元素，所有信息集合总大小（信息元素总个数）为 n ，当前需要执行若干次操作形如将两

个信息集合合并为一个信息集合，若我们可以在较小复杂度时间中往一个信息集合中插入元素，则我们可以快速的维护合并的过程。

具体地，对于两个大小分别是 $A, B (A < B)$ 的信息集合，我们将 A 集合的元素取出，一个一个的插入到 B 集合中，完成我们的信息合并。

复杂度分析：对于每个信息元素，当它被执行插入操作时，其所处的集合大小一定翻倍，所以对于每个信息元素只会被执行至多插入 $\log n$ 次，设一次插入复杂度为 $O(F(n))$ ，则总复杂度为 $O(F(n)n \log n)$ 。

例 1.¹ 给定一张 n 个点 m 条边有向图 $n, m \leq 10^5$ ，动态加入边，你需要返回当前图可以额外产生多少边，称一条边 (x, z) 是可以额外产生的，当且仅当这条边原来不存在，且存在 y ，使得存在边 $(x, y), (y, z), (z, y)$ ，注意，在额外产生边之后可能导致新的边可以额外产生。

解法

注意到对于由双向边联通的连通块，可以补全为完全图，所以我们维护所有连通块与剩下的单向边。

单向边可以用哈希表形如维护一个块到一个块的映射关系，需要同时维护一块指出的和指向一块的边。

对于加边时我们将块之间的边进行合并，根据启发式合并复杂度分析，有总体复杂度 $O(n \log n)$ 。□

启发式合并与整体二分结合

注意到启发式合并的复杂度为均摊 $O(n \log n)$ 次插入操作，单次合并的时间复杂度并没有确切保证，在一些非严格的过程中不能使用启发式合并进行处理。

有些题目看似破坏了启发式合并的复杂度分析，实则通过分析发现整体仍是严格的复杂度。

例 2. 给定一张 n 个点 m 条边的带权无向图， q 次询问，每次给定 x, y 询问 x 到 y 的路径的最小权值，一条路径的权值定义为路径上边权中的次大值，如果仅经过一条边，次大值定义为 0。可视为 n, m, q 同阶。

解法

特殊讨论最大值的一条边，剩下形如最小化次大值，即最小化经过边边集中边权最大的元素，尝试二分处理，考虑到多组询问，我们尝试整体二分进行处理。

考虑整体二分的过程，取 mid 为当前二分权值，此时将问题转化为如何快速验证通过所有边权在 $1 \sim mid$ 和一条额外边使得 x, y 相互可达。

¹<https://loj.ac/p/3275>

我们用并查集维护连通块，用启发式合并维护连通块与连通块是否存在额外边的映射关系，每次流程为先递归整体二分的右半部分，然后撤回然后再递归整体二分的左半部分。

似乎只要存在撤回就破坏了启发式合并的复杂度，对于上述过程，我们可以从另一个角度分析其时间复杂度：对于每个节点会被加入 $\log n$ 次，同理会被撤销 $\log n$ 次，撤销与加入等价，仅讨论加入。

注意到对于 i 点而言，每次加入时总是在加入完 $1 \sim i-1$ 后再加入 i 节点信息，可以视为顺序的执行了 $\log n$ 次从 $1 \sim n$ 的启发式合并，所以复杂度正确。 $\square$

类启发式合并思想：

在一些题目中在经过转换后可以描述为信息集合的合并，此时在合并中对于复杂度分析可以仿照启发式合并来处理。

例 3.² 给定一棵大小为 10^5 的二叉树有边权，非叶子节点有恰好两个儿子，你需要构造一个 dfs 序，最小化从相邻一个叶子到下一个叶子的距离和的最大值。

解法

二分答案后，树形 dp 检查是否存在叶子之间距离 $< mid$ 的 dfs 序。

设 dp 数组为 $f_{x,A,B}$ 表示 x 子树内进入 x 子树的距离和为 A ，离开子树的距离和为 B 是否可行，我们转而在 x 子树里，维护可行 (A,B) 点对。

我们尝试进行简化，对于任意 $(a,b),(c,d)$ 满足 $a < c, b < d$ ，我们可以去除掉 (c,d) 点对。

考虑子树间合并，注意到对于两个集合 S,T ，大小分别为 $|S|,|T|$ ($|S| < |T|$) 合并后集合大小变为 $2|S|$ （注意到简化后 $S+T, T+S$ 分别合并出的集合大小为 $|S|$ ）。

考虑类似启发式合并的复杂度分析，我们尝试对于每个信息元素分析均摊时间复杂度，注意形如当前合并 S,T 集合消耗 $|S|+|T|$ 的时间复杂度，得到大小为 $2|S|$ 的集合。

我们视为保留 S 和 T 中前 $|S|$ 个元素，对于 T 中剩下元素，我们视为消耗 1 的复杂度删去一个元素，对于 S 中的元素，我们视为消耗 2 的复杂度，并将集合大小翻倍。

至此，与启发式合并分析类似，可得当前整体合并的总时间复杂度为 $O(n \log n)$ 。

最终时间复杂度为 $O(n \log^2 n)$ 。 $\square$

2.2 树上启发式合并

通常情况下，树上启发式合并用在树上维护子树信息的情况，我们需要到某个点时维护出其子树信息并集，直观的想法是想要类似于启发式合并的方式加入删除信息并维护集合。

具体地，我们对于每个节点按照子树大小取最大的一个儿子成为重儿子，其余为轻儿子，将点和其重儿子之间的边称为重边，点和其轻儿子之间的边称为轻边。

²https://atcoder.jp/contests/agc007/tasks/agc007_e

我们每次保留重儿子信息，将轻儿子重新插入信息集合中，注意到对于一个节点被插入的次数即为到根轻边的条数，注意到经过一次轻边之后，子树大小至少翻倍，所以轻边数量不会超过 $O(\log n)$ 。

注意到树上启发式合并的另一种思路是对于每个子树同样动态独立维护信息，对于两个子树合并时，将小的一个子树插入到大的一个子树当中去，这样也是正确的，其处理方式和上述过程本质相同，但是上述处理方式相比其优点在于，可以全局的共用一个数组，在桶的记录等方面更有优势，如以下例子。

例 4.³ 给定 $n \leq 10^5$ 大小的树，对于每条边有小写字符，字符集为前 22 个小写字符，对于每个子树求最长的路径满足：这个路径上的字符经过重排后可以形如一个回文串。

解法

维护子树信息，直观的，对于每个子树，将子树路径分为过根和不过根讨论处理。

注意到当前限制形如，路径上至多存在一个字符出现奇数次，对于 (x, y) 这条路径合法我们可以拆分为 x 到根路径信息和 y 到根路径信息。

当前只有 2^{22} 种状态，我们对于子树内，存储每种状态深度最深的 dep 是多少，在合并时枚举出现奇数次的字符是什么拼接得到答案。

此时可以全局共用桶的信息，方便撤销与清除，总复杂度 $O(cn \log n)$ ，这里 c 是字符集大小。

□

启发式合并枚举公共祖先：

注意到启发式合并形式形如将一些信息暴力的插入一个整体的结构，于是我们可以视树上启发式合并的过程为枚举 lca 再将集合合并，此时可以解决一些需要用到 lca 信息的问题。

例 5.⁴ 给定 n 大小的树，给定 m 条链，问：有多少对链的交包含边数至少为 $k, n, m, k \leq 2 \times 10^5$ 。

解法

直观的讨论交的情况，若两条相交的链 lca 不同，我们枚举 lca 更深的一条链，注意到交的形式总是从 lca 起直上直下的长度为 k 以上的链，具体地可以用主席树维护 dfs 序查询。

若两条链的 lca 相同，此时我们尝试枚举 lca ，将所有 lca 等于这个点的链端点建立虚树，我们尝试在虚树上进行枚举处理，具体地讨论交的形式，交链的左端点形如两条链在左边一颗子树中的端点的 lca ，我们尝试启发式合并的过程枚举左侧 lca 的位置，注意到交链的形式形如从左侧 lca 开始长度至少为 k ，于是我们倍增找到对应点，用线段树合并维护并查询对应子树 dfs 序区间的情况。

³<https://codeforces.com/problemset/problem/741/D>

⁴<https://codeforces.com/problemset/problem/1336/F>



实现细节较为繁琐，存在一些边界情况需要特殊讨论。

□

类似的尝试用建立虚树再枚举 lca 的一些处理方式详见 [2]。

2.3 二进制分组

这里将以一道例题引入从朴素的合并到尝试使用二进制分组处理的想法。

例 6.⁵ 给定 $q \leq 3 \times 10^5$ 次操作，每次操作为加入一个字符串，删除一个存在的字符串，或者给定一个字符串，询问在加入的每个字符串在当前给定字符串中的出现次数总和是多少，保证输入总字符串长度不超过 3×10^5 ，操作强制在线。

解法

首先我们注意到，删除与加入独立，可以建立一个新集合，操作删除视为在新集合加入，询问答案为原集合出现次数-新集合出现次数，此时我们仅用考虑加入与询问。

我们注意到，对于询问一个定串与定集合的过程是经典的 AC 自动机可以完全解决的问题，但 AC 自动机形如静态结构，不能直接方便的增量在线加入结构，我们尝试用一些方式平衡建立自动机与查询的时间复杂度，具体 AC 自动机的处理方式不是本文重点，不在此讨论。

第一种平衡方式：朴素的讨论单串与整体的结构。

我们注意到对于单串与查询串进行匹配，我们可以使用 kmp 算法做到 $O(len)$ 的时间复杂度得到匹配数量，考虑从分块角度处理，令 B 块长，每 B 次加入之后或者当前剩余单串总长度超过 B 时将所有加入的字符串建立 AC 自动机，查询时讨论自动机与单串。

不妨设总串长为 n ，字符集大小为 c ，总复杂度可以做到 $O(qB + nB + n^2c/B)$ ，平衡后是根号级别。

第二种平衡方式：尝试二进制分组的考虑。

我们注意到串与串之间是独立的，我们可以分组建立 AC 自动机进行查询，每次加入一个串时我们将其放入一组，对于每组我们建立 AC 自动机，若当前集合中存在两组满足其包含的字符串个数相等，我们将两组合并，重新建立新的 AC 自动机。

可以发现任意时刻只会存在至多 $O(\log n)$ 个 AC 自动机，且对于任意串而言，类似于启发式合并复杂度分析，只会在所属集合大小翻倍时将其从新插入 AC 自动机，有总复杂度为 $O(cn \log n)$ ，这里 c 是字符集大小。

□

注意到第一种处理方式就是朴素的去做信息合并的过程，将当前问题转化为已知的问题，并暴力的划分，合并信息。

⁵<https://codeforces.com/problemset/problem/710/F>

对于当前问题的第二种处理方式，是巧妙的观察到询问信息独立，其本质是在建立形如线段树的结构，顺序加入叶子时进行向上更新的过程。

接下来我们介绍结构信息合并。

3 结构信息合并

在介绍了直接朴素的信息合并方式，我们尝试从维护结构信息的角度处理信息合并问题，在这个版块介绍几种常用信息合并结构与其复杂度分析过程。

3.1 左偏树

从简单的结构信息合并入手，我们先讨论可并堆的问题，这里介绍左偏树。

左偏树维护权值信息集合，支持快速找到集合中的最值，快速合并。

具体地，左偏树满足堆式结构，满足节点维护的权值信息和父亲节点有非严格偏序关系。

为保证合并时间复杂度我们选择对于每个节点维护 dist 。

其满足 $\text{dist}(0) = 0, \forall x, \text{dist}(lson(x)) \leq \text{dist}(rson(x)), \text{dist}(x) = \text{dist}(lson(x)) + 1$ 。

具体合并过程形如，比较节点权值，保留右子树，将左子树和另一棵树合并，验证是否交换左右儿子保证 dist 性质，更新 dist 。

复杂度分析：可以归纳说明，对于任意大小为 A 的以 x 为根的子树，有 $\text{dist}(x) \leq \log A + 1$ ，对于合并过程中，一次递归左子树会使其 $\text{dist} - 1$ 。

例 7.⁶ 给定一棵树大小为 n ，选 k 个点染黑， $n, k \leq 5 \times 10^5$ ，代价为边权求和，边权定义为两边子树黑色节点之差。

解法

子树 dp 维护 x, y 表示当前子树有选择 y 个点染黑，注意到对于一条边在上次过程中形如凸函数，两个子树合并的时候形如 $\min$ 卷积，即子树 dp 差分始终有序，尝试维护 dp 的差分数组并施加影响。

具体地，通过讨论，我们以 $k/2$ 为分界线，分别用大根堆和小根堆维护分界线，向上合并时需要支持单个堆的整体加减，支持堆与堆之间的合并，可以使用左偏树求解此问题。□

3.2 平衡树合并

平衡树一般维护有序信息集合，在集合上可以支持若干操作如区间求和，区间权值更新等，平衡树的合并可以视为两集合求并集后得到的一颗新的平衡树。

⁶Thupc2024_pre B

一些平衡树合并可以做到较优的复杂度，但是由于其考察较少，平衡树自身常数较大，所以在OI中较优复杂度的平衡树合并少有人提及与实现，通常对于平衡树的合并都是以启发式合并的方式进行处理。

本节将详细介绍复杂度较优的平衡树合并如何实现及具体细节并简要说明其复杂度。

3.2.1 finger search

finger search 指搜索的一种扩展，即维护上一次搜索得到的元素称为 finger，再基于 finger 元素做搜索操作。

具体地对于有序序列，定义 $d(x, y)$ 表示元素 x, y 集合中的排名之差，若上一次查找到的元素为 x ，下一次查找元素为 y ，其查找复杂度可以描述为一个关于 $d(x, y)$ 的函数如 $O(f(d(x, y)))$ ，我们称这个过程为 finger search。

finger search tree 定义为一种二叉搜索树，保留指向上一次操作的节点的指针，保留的节点称为 finger，同样的，我们定义 $d(x, y)$ 为中序遍历排名之差。

3.2.2 Splay

Splay 是一类二叉搜索树，在执行操作过程中使用旋转平衡其树高，保证其均摊时间复杂度。具体地定义与维护方式较为经典可见 [9]，不在此处介绍。

由于其每次操作后会将操作节点旋转到根，可以视根为 finger，于是每次操作可视在在上次的基础上做新的 finger search 操作。

在 [10, 11] 中有证明，其单次操作复杂度可以描述为 $O(\log d)$ 。

考虑 Splay 合并，我们将小的一颗平衡树按照中序遍历取出（即取出元素有序）顺序将其直接按照 Splay 的 insert 插入至第二棵平衡树中。

不妨设小树的大小为 m 大树的大小为 n ，设 m 个节点在 n 中的 rk 分别为 $D_1, D_2, \dots, D_m$ ，对于第 $i (i > 1)$ 次插入的复杂度为 $O(\log d_i)$ ， $d_i = D_i - D_{i-1}$ 。

此时第一次插入复杂度不超过 $O(\log n)$ ，剩下插入复杂度为 $\sum_{i=2}^m \log d_i$ 。

有 $\log A + \log B \leq 2 \log \frac{A+B}{2}$ ，根据类似均值的推导，我们有 $\sum_{i=2}^m \log d_i \leq m \log \frac{n}{m}$ 。

即合并时总插入复杂度不超过 $O(\log n + m \log \frac{n}{m})$ 。

对于一个元素，考虑其若干次合并的贡献，设其所属集合 siz 的变化量为一个序列 $p_1 = 1 < p_2 < \dots < p_k = n$ ，故当前节点消耗总复杂度为 $O(\sum_{i=1}^{k-1} \log \frac{p_{i+1}}{p_i}) = O(\log n)$

总合并复杂度为 $O(n \log n)$ 。

3.2.3 Treap

Treap 是一种通过随机权值优化的二叉搜索树，每个节点有一个键值和关联的随机优先级，每个节点的键值满足二叉搜索树的性质，每个节点的随机优先级满足堆的性质，顾名思义 $\text{Tree} + \text{Heap} = \text{Treap}$ 。

其保证复杂度的直接一种理解方式为视为当前是一个序列，然后每次选择随机优先级最高的作为根，然后划分为左右子树，注意到找随机优先值最高形如随机划分，则期望树高为 $O(\log n)$ 。

在 [12] 中有证明，对于任意两个树中元素 x, y 有其期望树上距离为 $O(\log d(x, y))$ 。

考虑在 Treap 上做 finger search 的插入合并两棵树的过程，不妨设这两棵树的所有键值互不相同，随机值互不相同。

考虑具体插入过程，取出小树的的中序遍历，依次尝试插入大树，我们维护上次插入元素作为 finger，我们尝试通过上次插入元素 x 在树上找到 $y(x < y)$ 。

对于二叉搜索树我们同时维护其父亲，考虑当前从 x 所在节点出发，我们想要向上跳跃找到 x, y 在树上的 lca 然后再向下递归到达 y 。

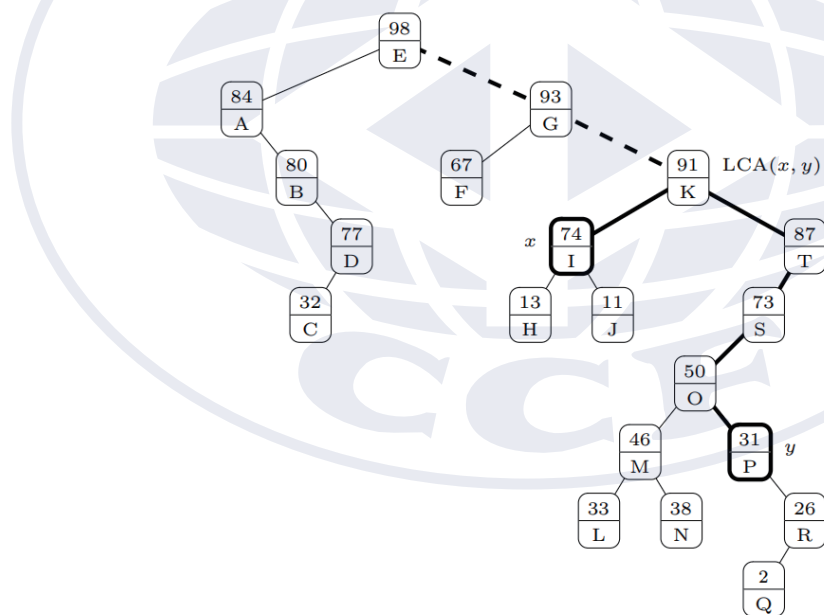


图 1: 在 Treap 上执行 finger search [13]

考虑如何执行当前过程找到 lca，注意到 lca 是满足 $x < \text{key}_{\text{lca}} < y$ 的深度最浅的点，我们初始维护 $l = x$ 做为 lca 的候选项，我们将从 x 开始执行若干次跳跃，不妨设当前跳跃到的节点键值为 v ，

- 若 $v < x$ ，则继续向上跳跃。

- 若 $x < v < y$ ，则将 l 改为当前节点，并继续向上跳跃。
- 若 $x < y < v$ ，则 l 不可能为当前节点往上，令 lca 为 l 并向下递归。

对于当前暴力跳跃的过程，看似时间复杂度正确，但是我们注意到图片虚线部分是我们一定需要遍历的部分，遍历这部分的时间复杂度与 x, y 在树上的距离无关，即复杂度并不能得到有效保证，我们尝试改变处理方式使得平衡最终消耗的向上的额外复杂度。

我们尝试在每次确定 l 之后，向上跳跃时同步从 l 向下进行寻找 y 的过程，每次改变 l 时，我们重新执行寻找 y 的过程，当前处理方式有总遍历节点数不超过 $2dis(x, y)$ 。（这里 dis 指树上距离）

此时根据上述分析有，当前有合并两颗大小为 $m, n (m \leq n)$ 的复杂度为 $O(m \log \frac{n}{m})$ 。

根据类似 Splay 合并的上述分析有，总合并复杂度为 $O(n \log n)$ 。

3.2.4 无旋 Treap

注意到上述 Treap 合并需要使用其旋转操作，即在某点插入元素后通过旋转保证整棵树的堆性质。

对于使用旋转操作的 Treap 而言，不能做到节点信息的可持久化且旋转操作实现较为繁琐。

这里简单介绍无旋 Treap 的基本操作与启发式合并的方式，注意当前一颗平衡树维护的内部键值形如一个集合（互不相同）。

- $\text{Split}(A, \text{val}) = \{C, D, x\}$ 将 A 按照 val 分裂成 $<\text{val}$ 与 $>\text{val}$ 的两个部分， $=\text{val}$ 权值的点 x （不存在则为空节点）。
- $\text{Merge}(A, B) = C$ 对于 A, B 集合满足 $\max A < \min B$ （这里比较 A, B 的键值）新集合为 A, B 集合的并。
- $\text{Union}(A, B) = C := A \cup B$ 求 A, B 集合的并集合。
- $\text{Intersection}(A, B) = C := A \cap B$ 求 A, B 集合的交集集合。
- $\text{Difference}(A, B) = C := A - B$ 求 A, B 集合的差集合。

对于前两个函数是为人熟知的无旋 Treap 的普通合并方式，这里不再详细介绍，其期望复杂度分别为 $O(\log |A|)$, $O(\log |A| + \log |B|)$ 。

对于启发式合并的过程我们可以视为 Treap 的 Union，这里详细介绍其处理方式：

对于合并 A, B Treap，不妨 A, B 根中 A 根随机权值优先级更高，取 A 根作为合并后的根，令 A 根的键值 k ，将 B 按照键值 k 分为 C, D, x 分别对应 $< k, > k, = k$ 的信息，舍弃 x 节点，将 C 递归与 A 左子树合并得到当前左子树， D 递归与 B 右子树合并得到当前右子树。

对于 Union, Intersection, Difference 合并两颗大小分别为 $n, m (m \leq n)$ 的平衡树其复杂度均为 $O(m \log \frac{n}{m})$ 。

Intersection 和 Difference 具体实现以及上述复杂度分析在原论文 [14] 有详细介绍，较为复杂不在此处说明。

根据类似 Splay 合并的上述分析有，总合并复杂度为 $O(n \log n)$ 。

这个过程是方便拓展到存在键值重复的元素，对于重复元素我们给它一个额外的标号顺序即可，合并过程是相同的。

笔者想要提醒读者的一点是当存在键值重复元素，一般的写法不是重标号离散化，而是重定义 Split 函数为分裂成 $\leq val$ 和 $> val$ 的两个部分。

注意，此时使用 Union 函数进行合并时可能导致树高退化，以下是一个极端情况的例子：

A, B 两树的所有节点的键值相同，考虑 $\text{Union}(A, B) = D$ 的过程：



图 2: 相同权值 Union 的例子

不妨设 A 树随机优先级高于 B 树，此时 Split 的过程就是将 B 树和 A 的右子树 C 进行合并，通过归纳，可以发现合并出来的 D 树其右链长度为 A, B 右链长度之和，此时树高并不满足高度期望为 $O(\log n)$ 。

3.3 线段树合并

线段树同样维护有序信息集合，相对平衡树其优势在于结构简洁明确，实现简单，可以支持大部分平衡树可处理的操作。

线段树的结构是为人熟知的，此处不再赘述。

线段树合并用于维护线段树间信息集合合并，由于相比平衡树其常数较小，时间复杂度易于分析，有较优拓展性，所以我们常用线段树合并。

复杂度分析：对于两颗合并的线段树，结构类似，两颗线段树合并的时候当一颗线段树节点为空时可以直接继承另一颗线段树的信息，每次合并后两个节点会变成一个节点，以节点个数作为势能，注意到设插入的操作次数为 $O(n)$ 次，总插入节点个数为 $O(n \log n)$ ，总线段树合并的过程复杂度也为 $O(n \log n)$ 。

例 8.⁷ 给定一个大小为 n 的有根树，给定 m 条限制关系 (u, v) ，满足任意 u, v 有 u 是 v 的祖先，询问对于每条边填 0/1 作为权值共 2^{n-1} 种方式，使得对于每对关系至少存在路径上一条边的权值为 1。 $n, m \leq 5 \times 10^5$

解法

当前尝试进行容斥，我们枚举每个条件是否满足，枚举当前条件不满足时带上 -1 的系数，具体地我们在树上做 dp，设 $f_{x,j}$ 表示当前处理完 x 子树内的所有钦定，当前要求 dep_x 到 j 的深度区间的所有边都钦定填 0 的权值。

考虑当前如何进行 dp 合并，对于两个子树的合并形如 $f'_{x,j} = (\sum_{i \geq j} f_{A,i})(\sum_{i \geq j} f_{B,i}) - f_{A,j}f_{B,j}$ 。

注意到额外可能的钦定为 m 组关系中 v 为当前 x 节点的关系，我们对于线段树需要维护当前子树和，需要维护乘法标记，合并时按照深度从大到小进行合并。

注意到当前不需要新建节点，按照上述节点的复杂度分析，总复杂度 $O(n \log n)$ 。

□

3.3.1 带新建节点标记的线段树合并

对于线段树合并而言，可能存在一些区间修改使得在标记下放时需要新建节点，此时对于节点的分析便需要更为精细的讨论，以下通过一个例子引入此类问题。

例 9.⁸ 给定 $n \leq 1500$ 大小的树，每个点给定一个权值，问：对于树上所有大小至少为 k 的连通块， k 大（从大到小）的权值和。

解法

将权值离散化，设权值从小到大为 $v_1, v_2, \dots, v_n$ ，尝试计算贡献，即枚举 v_i 计算有多少连通块满足包含至少 k 个权值 $\geq v_i$ 的节点。

此时将问题转化成为做若干次树形 dp，枚举 v_i 后设 $f_{u,t}$ 表示 u 子树中包含 u 节点且包含至少 t 个 $\geq v_i$ 节点的连通块数目，其转移形如子树间做卷积。

⁷NOI2020 命运

⁸九省联考 2018 秘密袭击

将其写为多项式的形式： $F_u(x) = \sum_i f_{u,i} x^i$ 注意到多项式次数不超过 $size$ 我们可以初始选择 $n+1$ 个点值，维护多项式在各个点值处的值。

更改枚举方式，我们先枚举点值 $s \in [0, n]$ 然后枚举 v_i 尝试统一计算出当前点值在各个 v_i 处多项式的值是多少，考虑当前转移：令 u 节点的权值排名为 A 则 $[1, A]$ 初值为 $s, (A, n]$ 初值为 1 转移形如子树内对位点值乘积，乘积后 $[1, n]$ 加 1。

注意到此时可以选择线段树区间加法操作，仅新建加法标记遍历到的节点，对位乘积可以描述为线段树合并的过程，若一边为一个节点，则可以对另一侧打区间加法/乘法标记，下放加法标记时需要新建线段树节点。

尝试分析线段树合并的时间复杂度，考虑当前线段树满足什么条件，有初始情况形如 $[1, A], (A, n], [1, n]$ 的修改，即对于任意线段树节点，要么同时有左右儿子，要么同时没有左右儿子，在所有标记下放的时候均不会造成节点的新建，故总复杂度仍 = 节点个数，即 $O(n \log n)$ 。

需要做 n 次线段树合并，最终用拉格朗日插值求出最终结果，故复杂度为 $O(n^2 \log n)$ 。

□

这个例子很好的告诉我们当线段树标记下放需要新建节点时复杂度仍尝试分析节点个数变化。

考虑对于不满足此初始条件时，标记下放会新建节点的线段树合并时间复杂度，相似的，我们假定在合并时，存在一边只有单个节点（没有左右儿子）时可以快速将影响应用到另一棵树上。

若直接考虑合并时分析其节点的新建情况，有例子形如，合并两条一致的黑色部分链，需要做标记下放而新建虚线节点，此时我们递归深度为 $\log n$ 但节点个数仅减少 1。

似乎此时导致总复杂度为 $O(n \log^2 n)$ 。

其实不然，注意到当前建立的叶子总是不存在左右儿子的，即会新建节点的线段树节点总是在原结构上只有一个儿子的节点，我们将其补齐，相似的，初始节点为 $O(n \log n)$ 个且满足例题性质。

有在此类线段树合并新建节点过程中我们仅会补齐上述节点，故总复杂度仍为 $O(n \log n)$ 。

3.3.2 线段树合并空间优化

考虑线段树合并的空间消耗，通常情况下我们空间复杂度的上界为 $O(n \log n)$ 。

注意到在值域为 $O(n)$ 且操作数量（单点插入）为 $O(n)$ 的一颗线段树中总消耗的空间复杂度为 $O(n)$ 。

这启发我们尝试改变线段树合并的顺序，并在删除节点时回收利用空间，得到紧的空间上界。

具体地，考虑当前问题形如一棵树，每个节点有若干个单点修改需要对于每个子树都求出子树内所有操作在线段树合并后的形式。

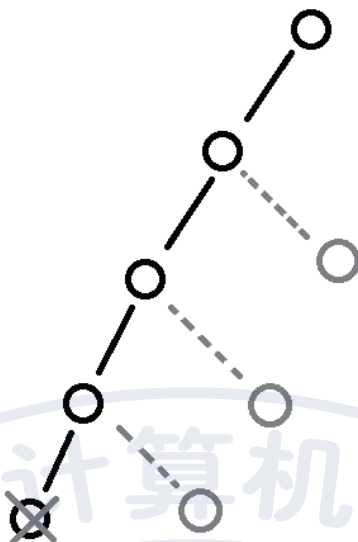


图 3: 合并的例子

若当前形如区间修改 $[l, r]$ ，我们考察其新建节点的情况，其新建的节点不会超过单点加入 l 并补全路径上右儿子与单点加入 r 并补全路径上左儿子，于是区间修改可以大致视为两次单点修改。

我们尝试按照每个点的操作次数做重链剖分（同树上启发式合并的处理过程），当前处理方式为先遍历重儿子，遍历完后递归轻儿子子树，与重儿子子树合并。

注意在空间回收意义下即消耗空间复杂度为任意时刻最多存在的节点数量。

可以发现任意时刻形如至多维护 $O(\log n)$ 棵线段树，且第 i 棵大小不会超过 $\frac{n}{2^i}$ ，我们尝试精细分析其占用的空间。

考虑线段树一共有 $\log n$ 层，对于加入 x 个节点的线段树，其第 i 层新建的节点数量为 $\min(x, 2^i)$ ，此时视为上界维护了 $\log n$ 棵线段树，第 $i \in [1, \log n]$ 棵有 2^i 个节点。

当前形如总节点数的上界为

$$\sum_{i=0}^{\log n} 2^{i+1} + (\log n - i)2^i \leq O(n) + \sum_{i=0}^{\log n} \sum_{j=i+1}^{\log n} 2^i = O(n) + \sum_{j=0}^{\log n} \sum_{i \leq j} 2^i \leq O(n) + \sum_{j=0}^{\log n} 2^{j+1} = O(n)$$

即总消耗空间复杂度为 $O(n)$ 。

4 总结

本文简单从朴素信息合并与结构信息合并的角度介绍了启发式合并，树上启发式合并，二进制分组，左偏树，平衡树合并（finger search 合并 Splay 与旋转 Treap，无旋 Treap 合并），



线段树合并（新建节点，空间复杂度），有较多的例子辅以读者理解，希望本文提及的信息合并处理方式与技巧对读者有所启发。

致谢

感谢中国计算机学会提供学习交流的平台。

感谢父母的养育之恩。

感谢巴蜀中学姜宗沛老师、黄新军老师的指导和关心。

感谢雅礼中学的肖子尧同学为本文写作提供帮助。

感谢为本文审稿的马未然、肖子尧、吴畅、郭雨豪、周子衡、颜桉、罗思远同学。

感谢各位读者抽出时间阅读本文。

参考文献

- [1] 2015 年集训队论文, 任之洲, 《浅谈启发式思想在信息学竞赛中的应用》
- [2] 2021 年集训队论文, 周镇东, 《浅谈一类树上路径相关问题》
- [3] OI Wiki, dsu on tree, <https://oi-wiki.org/graph/dsu-on-tree/>
- [4] xiaoziyao, 浅谈一类信息的暴力重构手法 <https://www.cnblogs.com/xiaoziyao/p/17413029.html>
- [5] 2013 年集训队论文, 许昊然 《浅谈数据结构题的几个非经典解法》
- [6] Wikipedia, leftist tree, <https://oi-wiki.org/ds/leftist-tree/>
- [7] Wikipedia, Finger search, https://en.wikipedia.org/wiki/Finger_search
- [8] Wikipedia, Finger search tree, https://en.wikipedia.org/wiki/Finger_search_tree
- [9] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. Journal of the ACM
- [10] R. Cole. On the dynamic finger conjecture for splay trees. part II: The proof. SIAM Journal of Computing, 30(1):44-85, 2000.
- [11] R. Cole, B. Mishra, J. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. part I: Splay sorting log n-block sequences. SIAM Journal of Computing, 30(1):1-43, 2000.
- [12] R. Seidel and C. R. Aragon. Randomized search trees. Algorithmica, 16(4/5):464-497, 1996.

- [13] Gerth Stølting Brodal. Finger Search Trees. University of Aarhus, 2005
- [14] Guy E. Blelloch and Margaret Reid-Miller. Fast set operations using treaps. In 10th Annual ACM Symposium on Parallel Algorithms and Architectures, Puerto Vallarta, Mexico, June -July 1998. ACM.
- [15] 2018 年集训队论文, 董炜隽, 《浅谈 Splay 与 Treap 的性质及其应用》
- [16] 约瑟夫用脑玩, finger-search+ 启发式合并, <https://www.luogu.com.cn/blog/ICANTAKIOI/finger-search-qi-fa-shi-ge-bing>
- [17] dpair, 记录一个线段树合并的 trick, <https://dpair.gitee.io/articles/sgtmrgt/>





浅谈树拓扑序计数相关问题的一些方法

杭州第二中学 陈昕阳

摘要

树拓扑序计数问题是一般图拓扑序计数问题的一个特殊情形。相较于一般图拓扑序计数问题目前不存在多项式复杂度的做法，由于树的结构较为特殊简单，树拓扑序计数问题存在优秀的性质，利用这些性质可以在线性时间内求解该问题。本文从树拓扑序计数问题出发，主要介绍解决树拓扑序计数问题及其拓展问题的几种常用方法，并通过一些例题展示了这些方法在信息学竞赛中的应用。

1 前言与约定

图拓扑序计数问题是图论中的经典问题：给定一张 n 个点 m 条边的有向图，请问其有多少不同的拓扑序？虽然找到有向图的任意一种拓扑序或报告无解是简单的，目前对拓扑序计数却不存在多项式复杂度的做法。

但如果对图这一结构进行弱化变为有向树（将所有有向边视为无向边后形成一颗树的有向图），并且保证存在一个点 r 使得图是以 r 为根的叶向树时（所有有向边方向是从深度较小点指向深度较大点的有向树），这个问题则存在非常优美的解法与结论，有时被称为树拓扑序计数问题。事实上这一问题在信息竞赛中经常作为转化后的题意出现，并且也已经有了许多方面的拓展。

本文将由浅入深地介绍解决这一类问题的各种方法，以及用这些方法该如何解决各种拓展问题。这些方法与问题一部分是已有的，一部分则是本文原创。

1.1 约定

一张图的拓扑序数量最多可能达到 $n!$ ，在 n 较大时会难以储存，所以我们约定只需计算答案取模大质数的结果。这同时意味着我们认为对较小的数（比如 $[1, n]$ 内的）求逆元总是可行。

事实上预处理 $1 \sim n$ 的逆元还需要做一次 $O(\log \text{mod})$ 的快速幂，下文也会忽略这部分的复杂度。

本文中提到的各种概念，如果没有特殊说明，一般与大众所了解的定义一致。

2 从经典问题引入

本节会介绍树拓扑序计数问题的经典做法，以及从叶向树到叶向森林（每个连通块都是叶向树的有向图）的拓展。

2.1 树拓扑序计数问题的经典解法

先明确一下问题：

例题一¹：给定一张 n 个点以 r 为根的叶向树，请问其有多少种不同的拓扑序？ $n \leq 10^6$ 。

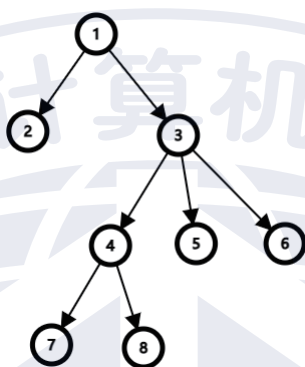


图 1: 一个叶向树的例子

该问题为人熟知的做法是使用动态规划，设 dp_u 表示仅考虑 u 子树内的点和边构成的子图的拓扑序数量，显然最终答案就是 dp_r 。自下至上 DP，假设对于点 u ，其所有孩子 $v_1, v_2 \dots v_k$ 的 DP 值都已经计算完成，考虑如何求出 dp_u 。首先因为 u 直接或间接地可以到达其子树内的所有点，所以其必须排在最前面，此后我们假设 $v_1, v_2 \dots v_k$ 每个子树内所有节点的排列顺序已经确定，那么根据树的结构， $v_1, v_2 \dots v_k$ 这些子树之间是没有先后限制的（之间没有边），问题等价于归并长度依次为 $siz_{v_1}, siz_{v_2} \dots siz_{v_k}$ （ siz_p 表示 p 的子树大小）的序列，确定 $v_1, v_2 \dots v_k$ 子树内所有节点排列顺序有 $dp_{v_1} \times dp_{v_2} \times \dots dp_{v_k}$ 种方案，而归并这些序列有 $\left(\sum_{i=1}^k siz_{v_i} \right)$ 种方案，相乘即可。

该做法还可以进一步化简，可以归纳证明 dp_u 事实上等于 $\frac{(siz_u)!}{\prod_{u \text{ is ancestor of } p} siz_p}$ ，即 dp_u 是 u 子树大小的阶乘除以其所有后代的子树大小之积。当 u 是叶子时这一点显然成立，而 u 并非叶子时，将多重组合数拆开，等于要乘上 $(\sum_{i=1}^k siz_{v_i})! = (siz_u - 1)! = \frac{(siz_u)!}{siz_u}$ ，除以所有 $(siz_{v_i})!$ ，

¹题目来源：经典问题

而根据归纳假设 $dp_{v_i} = \frac{(siz_{v_i})!}{\prod_{v_j \text{ is ancestor of } p} siz_p}$ ，整理后即可得到结论。

时间复杂度为 $O(n)$ 。

2.2 拓展到叶向森林

叶向森林即每个弱连通块都是一颗叶向树。

上文的经典做法其实基于两点：

- 如果一张图可以被划分为若干部分，不同的部分之间没有限制（连边），那么我们只需将各部分答案相乘，再乘上一个多重组合数即可得到整体的答案。
- 如果图中是存在一个点必须先于剩下的所有点，那么因为这个点必须占据拓扑序的第一个位置，可以将其直接删去。

那么叶向森林的情况根据第一点，只需分别求出每颗叶向树的答案，然后用多重组合数合并即可。

类似地，可以得到答案是 $\frac{n!}{\prod_{i=1}^n siz_i}$ ，求解的时间复杂度仍为 $O(n)$ 。

此处特别指出图为叶向森林情况的答案公式，是因为后文会试图把更复杂的问题规约到叶向森林的情况，并使用该公式求解。

3 经典问题的新解法

树拓扑序计数问题的答案公式非常简洁优美，笔者最近发现该公式存在组合意义下的解释，对比之前的数学归纳法，利用该组合解释还可以立刻解决一个较为复杂的拓展问题。

3.1 组合解释

考虑以下针对给定叶向树的过程：

- 每个点有黑或白的颜色，初始所有点都是白色，另有一个长度为 n 的辅助序列 a 。
- 进行 n 轮操作，每轮选一个白色点（之前选过的点也可以再选），找到其最浅的白色点祖先 p ，将其涂黑。假设这是第 i 轮，记录 $a_i = p$ 。

可以发现，无论怎么选择每一轮的白色点，最终得到的序列 a 都是原叶向树的一个拓扑序。但一个拓扑序可能对应多种 n 轮每轮选择一个白色点的方案。

不过可以观察到, 当 a 序列是一个拓扑序时, 会生成其的选择方案永远是 $\prod_{i=1}^n \text{siz}_i$ 种。这是因为, 如果拓扑序第 i 位是 u , 这代表在过程中第 i 轮被涂黑的点是 u , 而在这一轮 u 被涂黑的充要条件是, 选择的白色点 v 在 u 子树内, 所以涂黑 u 的一轮选择白色点的方案数是 siz_u , 而不同轮之间选择白色点的方案数是独立的, 由此得到结论。

而可以注意到, n 轮每轮选择一个白色点的方案数是 $n!$ 。因为该过程必定生成一个拓扑序, 且对于任意拓扑序, 都存在恰 $\prod_{i=1}^n \text{siz}_i$ 种生成其的方案。所以拓扑序的数量是 $\frac{n!}{\prod_{i=1}^n \text{siz}_i}$, 得证。

3.2 一个拓展问题

树拓扑序计数问题事实上是排列计数问题。排列计数问题的经典拓展方式是固定其中一项 $p_x = y$, 统计给剩下的项填上值并且符合限制的方案数。

那么如果对树拓扑序计数问题进行该拓展, 该如何解决呢?

例题二²: 给定一颗 n 个点, 以 1 为根的叶向树和长度为 n 的权值序列 b 。定义 $f(u)$ 为, 所有该图的拓扑序中 $b_{\text{id}x_u}$ 之和 (其中 $\text{id}x_u$ 表示在拓扑序上 u 出现的位置)。对 $1 \leq u \leq n$, 求 $f(u)$ 。 $n \leq 5000$ 。

事实上我们可以解决更一般的问题: 计算所有 $\text{ans}(x, y)$ 表示 x 出现于第 y 位的拓扑序数量。

考虑组合解释中描述的过程, 限制转化为 x 在第 y 轮被涂黑。可以发现要知道 x 点在某一轮被涂黑, 只需知道 x 的祖先中最浅的白点 p 是谁即可。每次从剩余白点中选一个也可分为两种情况, 第一种是选中了 p 子树内的点, 那么 p 被涂黑, 最浅的白点变为 p 到 x 简单路径上的第二个点 (或者如果 $p = x$ 那么我们知道这一轮 x 被涂黑); 或者选中的白点不在 p 子树内, 最浅的白点仍是 p 。

于是可以这样设计 DP: 设 $dp_{i,p}$ 为已经涂完 i 轮, x 祖先中最浅的白点是 p 的涂色方案数。边界是 $dp_{0,1} = 1$, 转移是枚举第 i 轮的涂色情况: 要么在涂完 $i-1$ 轮后最浅的白点就是 p , 第 i 轮选的点不在 p 子树内, 那么 $dp_{i,p}$ 加上 $(n+1-i-\text{siz}_p)dp_{i-1,p}$; 要么是涂完 $i-1$ 轮后最浅的白点是 fa_p , 第 i 轮选的点在 fa_p 子树内, 那么 $dp_{i,p}$ 加上 $\text{siz}_{fa_p}dp_{i-1,fa_p}$ 。

最后 $\text{ans}'(x, y) = dp_{y-1,x}(n-y)!$, 因为前 $y-1$ 轮涂色一定要涂黑 x 的所有祖先, 并且 x 仍为白色, 这样的方案数是 $dp_{y-1,x}$, 第 y 轮一定要涂 x , 而后 $[y+1, n]$ 这些轮则任意涂。

这里是 $\text{ans}'(x, y)$ 是因为我们现在统计的是 x 在第 y 轮被涂黑的不同选择方案数, 但每 $\prod_{i=1}^n \text{siz}_i$ 种选择方案对应一种拓扑序, 所以真正的 $\text{ans}(x, y) = \frac{\text{ans}'(x, y)}{\prod_{i=1}^n \text{siz}_i}$ 。

对于原问题则有 $f(u) = \sum_{i=1}^n \text{ans}(u, i)b_i$ 。

²题目来源: <https://qoj.ac/problem/7792>

时间复杂度 $O(n^2)$ ，对 i 一维使用滚动数组则能轻易做到线性空间。

3.3 拓展问题的进一步拓展

拓展问题中我们 $O(n^2)$ 算出了所有带单项钦定的答案。事实上使用类似的方法，可以 $O(n^{2k})$ 算出对于所有钦定 k 元组（这有 $\binom{n}{k} n^k$ 种）的答案，即所有从 n 个点中选 k 个点，分别钦定这 k 个点必须填 k 种对应值的答案。这里我们认为 k 是小常数。

但稍有不同的是，在单项钦定的情况中，一旦目标点 x 被涂黑，接下来的涂色过程就不需考虑了，可以直接乘上任意涂色的方案数 $(n-y)!$ 。在多项钦定的情况下，事实上要记录 $dp_{i,S,T}$ 其中 T 是一些二元组构成的集合， S 是 $\{1,2,3\dots n\}$ 的子集，表示已经涂了 i 轮，在这 i 轮中作出限制 $T = \{(a_j, b_j)\}$ ，点 a_j 必须在第 b_j 轮被涂黑， S 集合是所有尚未被涂黑的带有钦定的点祖先中首个白点构成的集合。

转移同样考虑第 i 轮涂黑了哪个点，可以发现被涂黑的点要么是 S 集合中某个点 p 的父亲，那么前驱状态就是向 S 集合加入 fa_p 并删去 fa_p 所有孩子；要么被涂黑的点在 T 中被记录；要么第 i 轮操作前 S 集合和第 i 轮操作后一模一样。

只保留 $|T| + |S| \leq k$ 的 DP 状态，并且对于 $|T| = k$ 的状态直接结算答案不再继续转移，即可做到 $O(n^{2k})$ 的复杂度。

4 容斥原理与枚举值域线

叶向森林的情况答案有非常简洁的公式，但使用条件稍显苛刻：必须满足每个点入度不超过 1，并且图无环。

使用容斥原理可以将不满足条件的图拆分为很多满足该条件的子问题，只需将子问题的答案带权求和（乘上对应容斥系数），即可得到原问题的答案。

枚举值域线则是一种拆贡献的技巧，如果要计算 $|u-v|$ ，其中 $u, v \in \mathbb{Z}$ ，只需计算 $\sum_{k \in \mathbb{Z}} [u \leq k][v > k] + [v \leq k][u > k]$ 。

本节将介绍如何使用容斥原理解决两个经典问题，并给出维护值域线技巧的一道例题。

4.1 有向树拓扑序计数

例题三³：给定一张 n 个点的有向图，保证将所有有向边视为无向边后图是一颗树，请问其有多少种不同的拓扑序？ $n \leq 5000$ 。

任意定根 r ，可以将限制边分为两类，根向的和叶向的。如果所有边均为叶向，那么显然这是叶向森林拓扑序计数。

³题目来源：<https://loj.ac/p/3802>



但事实上可能还有根向边，使用容斥原理枚举一个子集的根向边违反限制变为叶向，其余根向边忽略不管。这样得到了 $2^{\text{根向边}}$ 个子问题，每个子问题都是叶向森林拓扑序计数，将每个子问题的答案乘上容斥系数加和即可得到原问题答案。

具体地，我们设 $dp_{u,i}$ 表示确定了 u 子树内所有根向边是否违反限制， u 所在叶向树有 i 个点的 $\frac{(-1)^{|S|}}{\prod_{\substack{u \text{ is ancestor of } v \\ \text{is ancestor of } v}} \text{siz}_v}$ 之和。转移时先递归到 u 的各个孩子去计算 DP 值，然后每次合并上一个孩子 v_i 。如果 u 和 v_i 的边是叶向的，那么直接进行背包合并；如果 u 和 v_i 的边是根向的，那么有两种情况，第一种这条边变为无限制，那么合并后 u 所在叶向树点数不变；第二种这条边容斥变为叶向，那么系数要乘 -1 ，也是进行背包合并。

合并完各个孩子的信息后，对于 $dp_{u,i}$ ，将值乘上 $\frac{1}{i}$ 。

最后答案是 $n! \sum_{i=1}^n dp_{r,i}$ 。

时间复杂度根据树形背包合并的复杂度分析，是 $O(n^2)$ 的。

4.2 有向链的特殊情况

如果对有向树进一步特化，求解拓扑序数量的复杂度可以进一步降低。

例题四⁴：给定一张 n 个点的有向图，保证将所有有向边视为无向边后图形成一条链，请问其有多少种不同的拓扑序？ $n \leq 10^5$ 。

不妨认为链上点的编号从左到右分别是 $1 \sim n$ 。

如果图由许多条链组成，而每条链的所有边方向都一致，那么利用叶向森林的拓扑序计数公式可知，设这些链的长度分别为 $a_1, a_2, \dots, a_k$ 答案是 $\frac{n!}{\prod_{i=1}^k a_i!}$ 。

比如说我们希望所有边的方向都是从左到右，那么遇到从右到左的边就需要容斥，要么让这条边违反限制，要么不管这条边。设 dp_i 为已经确定了前 $i+1$ 个点之间边的容斥方向， i 作为一条链的右端点的所有方案的 $\frac{(-1)^{|S|}}{\prod_{j=1}^k a_j!}$ 之和，这里 $a_1 \sim a_k$ 是前 i 个点构成的链长度的集合。

可以发现 dp_i 不为 0，当且仅当要么 i 和 $i+1$ 之间的边初始是从右到左的，要么 $i = n$ 。转移是枚举 i 所在链的左端点 j ，将 dp_i 加上 $dp_{j-1} \frac{(-1)^{\text{sum}_i - \text{sum}_j}}{(i-j+1)!}$ ，其中 sum_i 表示 $1 \sim i$ 这些点之间边从右到左的数量。

可以发现这是卷积形式，进行半在线卷积即可 $O(n \log^2 n)$ 算出所有 $dp_1 \sim dp_n$ 。最后答案为 $n! dp_n$ 。

⁴题目来源：<https://loj.ac/p/575>



4.3 更复杂的答案形式

有一些树拓扑序计数的拓展问题不仅要求数出方案数，还针对拓扑序的具体内容赋予了对应权值，要求求出所有合法拓扑序的权值之和。

例题五⁵：给定一颗 n 个点，以 1 为根的叶向树。定义一个合法拓扑序 a 的权值 $f(a)$ 为 $\sum_{i=1}^{n-1} |a_i - a_{i+1}|$ ，对所有不同的 a ，求 $f(a)$ 之和。合法拓扑序 a 的定义是长度为 n 的排列满足 $\forall 2 \leq u \leq n, a_u > a_{fa_u}$ 。 $n \leq 500$ 。

考虑先固定 i ，试图算出所有合法拓扑序的 $|a_i - a_{i+1}|$ 之和。枚举值域线 k ，不失一般性变为求 $a_i \leq k$ 且 $a_{i+1} > k$ 的拓扑序数量。

将每个点 u 按照其填上的值 a_u 涂色，如果 $a_u \leq k$ 将其涂为白色，否则将其涂为黑色。那么限制是 i 点必须为白色， $i+1$ 点必须为黑色。

考虑如果一种涂色方案已经被确定，如何计算有多少拓扑序对应这种涂色方案。首先要保证对于所有边 $u \rightarrow v$ ，不存在 v 是白色而 u 是黑色，否则不可能有拓扑序对应这种涂色方案。接下来，只要白色点填的值在 $[1, k]$ 内，黑色点填的值在 $(k, n]$ 内，所有两侧点颜色不同的边的拓扑序限制一定是自然得到满足的。

接下来只需考虑两侧点颜色相同的边给予的拓扑序限制，那么首先每种颜色独立。只考虑某种确定的颜色的话，可以发现问题等价于叶向森林拓扑序计数，可以直接套用公式。我们记 siz'_u 为 u 子树内有多少点与 u 同色，那么答案就是 $k!(n-k)! \prod_{i=1}^n \frac{1}{siz'_i}$ ，当然我们需要保证白色点有 k 个，黑色点有 $n-k$ 个。不过换个角度不直接枚举 k 的话，可以看作是方案带有两种颜色出现次数阶乘之积的权值。

直接 DP，设 $dp_{u,k}$ 为确定了 u 子树内所有点的颜色， u 为白色，子树内共有 k 个黑色点，并且若 $i, i+1$ 在子树内的话， i 必须为白色， $i+1$ 必须为黑色的限制得到满足的所有涂色方案的 $\frac{1}{siz'_i}$ 之和。转移是先对所有孩子的信息背包合并（当然孩子是可以为黑色的，此时对其子树涂色只有一种方案，对应权值也容易预处理），再考虑自己的 $\frac{1}{siz'_u}$ 权值以及如果 $u = i$ 或 $u = i+1$ 的话这个点的黑白限制。最后算答案时方案数是 $\sum_{k=1}^n dp_{1,k} k!(n-k)!$ 。

一轮这样的 DP 是 $O(n^2)$ 的，要进行 $n-1$ 轮，故总复杂度为 $O(n^3)$ 。

5 优化时间复杂度的两个技巧

上文提到的主要是整体上如何解决树拓扑序计数相关问题的方法，关系的是是否能给出一个多项式复杂度的做法。但有时可能得到的做法时间复杂度过高，比如多一个 n ，此时可以用本节介绍的两个技巧降低时间复杂度。

⁵题目来源：原创

5.1 拉格朗日插值法

如果有一系列多项式 $f_1 \sim f_k$ ，现在希望计算 $\prod_{i=1}^k f_i$ 的各项系数，但直接乘复杂度太高。如果已知 $\prod_{i=1}^k f_i$ 的次数不超过 n ，可以算出 $f_1 \sim f_k$ 在 $x = 0 \sim n$ 时的点值，将点值相乘后得到 $\prod_{i=1}^k f_i$ 在 $x = 0 \sim n$ 时的点值，再使用拉格朗日插值法从点值插回多项式。

在此给出插值公式：已知 n 次多项式的 $n+1$ 个点值 (x_i, y_i) ，那么 $f(x) = \sum_{i=0}^n y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$ 。

本文中用到的插值法均为 $O(n^2)$ 的暴力插值，即先算出 $\prod_{i=0}^n (x - x_j)$ ，然后对所有 i 将总乘积 $O(n)$ 除以一次多项式 $(x - x_i)$ ，得到 $\prod_{j \neq i} (x - x_j)$ 各项系数后乘上 $\frac{y_i}{\prod_{j \neq i} (x_i - x_j)}$ 后加到答案多项式上。

5.2 允许不满足所有限制

先前我们研究的都是只要存在一条边 $u \rightarrow v$ ，那么 u 就必须出现于 v 之前的情况。如果我们改为求有多少排列 p 满足，删去原图的不超过 k 条边后其可以成为一个合法拓扑序呢？可以做出以下两个观察：

- 对于任意排列 p ，最少删边数量使其可以变为合法拓扑序，等于最少反转边数量（即选择最少的原图中的边将其反向），使其可以变成合法拓扑序。
- 对于任意排列 p ，恰好存在一个边集 $E' \in E$ ，使得反转 E' 中所有边方向后， p 变为一个合法拓扑序。

所以我们只需枚举原图所有大小不超过 k 的边子集 E' ，反转 E' 中所有边方向后求拓扑序数量并加和即可得到答案。

例题六⁶：给定一张 n 个点的有向图，保证将其所有边视为无向边后形成一颗树。对 $k \in [0, n]$ 计算有多少排列 p 满足，删去原图不超过 k 条边后排列 p 可以变为一个合法拓扑序。 $n \leq 500$ 。

考虑使用元 x 的指数记录反转了多少条边，即设出答案的 GF。

根据上面的分析，我们可以认为原图的每条边还有一条反向边，权值为 x ，而原边权值为 1。我们需要对每对重边选出一条，总共选出 $n-1$ 条边，定义这种选择方案的权值是所有选出边权值的乘积再乘上被选出所有边构成图的拓扑序数量，只需算出所有选择方案权值之和即可得到答案（最后还要做个前缀和）。

⁶题目来源：原创

任意定根 r ，在从一对重边中选出一条后，如果选出的边的方向并非叶向，就使用容斥原理将这条边拆开。类似有向树的拓扑序计数的方法进行 DP 即可，但此时 DP 数组的每一位都是多项式，计算多项式乘法需要消耗额外的时间，即使允许 NTT 时间复杂度仍为 $O(n^3 \log n)$ 。

注意到这些多项式次数都不超过 n ，于是使用拉格朗日插值法维护 $n+1$ 个点值最后插值出答案多项式即可做到 $O(n^3)$ 。

5.3 更复杂的值域线与插值法

事实上在枚举值域线时可能需要枚举不止一条，插值法也可以应用得更灵活。

例题七⁷：给定一颗 n 个点，以 1 为根的叶向树，和三个长度均为 n 的权值序列 b, c, d 。定义一个合法拓扑序 a 的权值 $f(a)$ 为 $\sum_{i=1}^n \sum_{j=1}^n c_i d_j b_{|a_i - a_j|}$ ，对所有不同的 a ，求 $f(a)$ 之和。合法拓扑序的定义是长度为 n 的排列满足 $\forall 2 \leq u \leq n, a_u > a_{fa_u}$ ，并且认为 $b_0 = 0$ 。 $n \leq 500$ 。

首先略微调整一下 f ，记 $F(a) = \sum_{i=1}^n \sum_{j=1}^n [a_i < a_j] c_i d_j b_{a_j - a_i}$ ，那么只要能计算 $F(a)$ 之和，交换 c, d 序列再求一次 $F(a)$ 之和即可得到原问题的答案。

这里的贡献比例题五的 $[a_i < a_j](a_j - a_i)$ 更加复杂，是 $[a_i < a_j]b_{a_j - a_i}$ ，枚举一层值域线难以处理，所以考虑枚举两层值域线 V_1, V_2 其中 $V_1 \leq V_2$ ，限制 $a_i \leq V_1, a_j > V_2$ 。还是将每个点 u 按照 a_u 涂色，如果 $a_u \leq V_1$ 将其涂为白色，如果 $V_1 < a_u \leq V_2$ 将其涂为灰色，如果 $a_u > V_2$ 将其涂为黑色。那么限制是 i 点必须为白色， j 点必须为黑色。

类似地，可以得到当一种合法涂色方案（即对于每条边，父亲的颜色总是不深于孩子的）被确定时，对应的拓扑序数量是 $(V_1!)(V_2 - V_1)!(n - V_2)! \prod_{i=1}^n \frac{1}{\text{siz}'_i}$ ， V_1, V_2 可以看作是对白色、灰色点的计数。

仍然 DP，设 f_u 为 u 为黑色， u 子树内所有点 v 的 $\frac{1}{\text{siz}'_v}$ 乘积； $g_{u,i}$ 为 u 为灰色，对 u 子树内所有点涂色，恰有 i 个灰点的所有合法涂色方案的 $\prod_{u \text{ is ancestor of } v} \frac{1}{\text{siz}'_v}$ 之和； $h_{u,i,j}$ 为 u 为白色，对 u 子树内所有点涂色，恰有 i 个白点， j 个灰点的所有合法涂色方案的 $\prod_{u \text{ is ancestor of } v} \frac{1}{\text{siz}'_v}$ 之和。这三个东西的转移都是先背包合并，再考虑自身的 $\frac{1}{\text{siz}'_u}$ 的影响。

不过我们还要从黑点中选出一个点作为 j ，权值乘上 d_j ；从白点中选出一个点作为 i ，权值乘上 c_i 。所以对 f, h 要额外加一维表示是否已经选出了对应的 j/i 。注意这里的 i, j 的定义不是上一段 DP 数组中的，而是被限制涂为白色的点和被限制涂为黑色的点。

转移要做二维背包合并，时间复杂度为 $O(n^4)$ ，不可接受。可以发现对于 h, j 这一维的值是多少不影响转移过程。所以我们可以先 $O(n^2)$ 算出 f, g 。然后对于 h ，将答案视作关于 j 的 n 次多项式，只维护 $j \in [0, n]$ 时多项式的点值，最后使用拉格朗日插值法插出多项

⁷题目来源：原创



式。这样 DP 部分和插值部分时间复杂度均为 $O(n^3)$ 。

这样就对所有 $V_1 \leq V_2$ 算出了对所有拓扑序 a , $\sum_{i=1}^n \sum_{j=1}^n c_i d_j [a_i \leq V_1][a_j > V_2]$ 之和。进行一次二维差分即可对所有 $V_1 \leq V_2$ 得到 $\sum_{i=1}^n \sum_{j=1}^n c_i d_j [a_i = V_1][a_j = V_2 + 1]$ 之和, 从而可以计算答案。并且可以发现我们事实上求出了比要求的答案更多的信息, 即我们对所有值的二元组 (i, j) 算出了, 对所有合法拓扑序 a , $a_p = i$ 的点的权值 c_p 乘上 $a_q = j$ 的点的权值 d_q 之积的和。

时间复杂度为 $O(n^3)$, 空间复杂度为 $O(n^2)$ 。

事实上使用 3.3 的做法取 $k = 2$ 并抛弃掉一些无用信息也可以做到 $O(n^3)$ 时间, 但实现起来可能更为复杂。

5.4 信息共用

在进行多次 DP 时, 一部分信息可能被重复使用, 那么事实上可以只计算这些信息一次从而减少消耗的时间。

事实上在 3.2 的解法中就用到了这种思想: 在 $dp_{i,p}$ 的定义中, 无论 x 是 p 子树内的哪个点, $dp_{i,p}$ 都是一致的, 不必重复计算。这是非常自然的, 因为无论 x 是哪个点, 要将其涂黑都必须将从根到 p 这条链上的所有点涂黑, 这一公共部分没有差别。

在此我们给出另一使用这一思想可以降低时间复杂度的例题:

例题八⁸: 给定一颗 n 个点, 以 1 为根的叶向树。对所有 $1 \leq u < v \leq n$ 求出 $f(u, v)$ 表示所有合法拓扑序 a 的 $|a_u - a_v|$ 之和。合法拓扑序的定义是所有是长度为 n 的排列满足 $\forall 2 \leq u \leq n, a_u > a_{fa_u}$ 。 $n \leq 500$ 。

还是像例题五一样枚举值域线, 那么要对所有 $p \neq q$ 算出钦定 p 必须是白色点, q 必须是黑色点的所有染色方案权值和。

直接枚举 q , $O(n^2)$ DP 算出所有 $g_{u,i}$ 表示对 u 子树内所有点染色, 构成合法染色方案, 一共有 i 个白色点, 并且 q 点如果在子树内则必须为黑色的所有染色方案权值和 (权值是子树内所有 $\frac{1}{siz'_u}$ 的乘积)。这样事实上压缩表示了 u 为黑色或白色的两种信息, $i > 0$ 时 u 为白色, $i = 0$ 时 u 为黑色。

接着可以考虑自上至下 DP, 设 $dp_{u,i}$ 表示对 u 子树外所有点染色, 构成合法染色方案, 如果 q 在子树外则必须为黑色, 已知 u 子树内有 i 个白色点的所有染色方案权值和 (这里权值是子树外所有 $\frac{1}{siz'_u}$ 的乘积再乘上两种颜色点数量阶乘之积), 这里只需考虑 $i > 0$ 的情况。

这样如果能 $O(n^2)$ 求出 dp , 钦定 p 白色, q 黑色的所有方案权值和就是 $\sum_{i=1}^{siz_p} dp_{p,i} g_{p,i}$ 。

考虑如何求出 dp , 首先 $\forall 1 \leq i \leq n, dp_{1,i} = i!(n-i)!$ 。对于 $u > 1$, $dp_{u,*}$ 应该从 $dp_{fa_u,*}$

⁸题目来源: 原创

先在第二维上执行 $\frac{dp_{fa_u,i}}{i} \rightarrow dp'_{fa_u,i-1}$, 这代表考虑 fa_u 的贡献并去掉其的影响。接着应该是将 $dp'_{fa_u,*}$ 在第二维上与 fa_u 除 u 以外的其他孩子 $v_1 \sim v_k$ 的 $g_{v_i,*}$ 逐个进行减法卷积, 因为从 fa_u 换到 u , u 的其他孩子也由子树内变为子树外, 需要考虑它们的贡献。

从 $dp \rightarrow dp'$ 是简单的, 对于剩下的部分, 相当于给出 k 个多项式 $f_1 \sim f_k$ 以及大多项式 F , $\deg(F) = \sum_{i=1}^k \deg(f_i)$, 要对所有 $1 \leq i \leq k$ 算出 $subconv(F, \prod_{j \neq i} f_j)$ 的 $0 \sim \deg(f_i)$ 次项系数, $subconv(P, Q)$ 表示对 P 和 Q 执行减法卷积。

考虑对 $1 \leq i \leq k$ 算出 $prd_i = \prod_{j=i}^k f_j$, 那么对于 $i = 1$ 结果就是 $subconv(F, prd_2)$ 。接下来要计算的所有信息都需要除去 f_1 , 所以可以直接令 $F \leftarrow subconv(F, f_1)$ 然后不管 f_1 这一项, 这时只需保留 F 的 $0 \sim \sum_{j=2}^k \deg(f_j)$ 项系数即可。接下来 $i = 2$ 的结果就是 $subconv(F, prd_3)$, 以此类推。

这样做的时间复杂度为 $O(\deg^2(F) - \sum_{i=1}^k \deg^2(f_i))$, 根据树形背包的时间复杂度分析整体是 $O(n^2)$ 的。

当 q 固定时计算一轮的复杂度为 $O(n^2)$, 故总时间复杂度为 $O(n^3)$ 。

例题八如果直接应用例题五的解法, 时间复杂度为 $O(n^4)$ 。为什么我们能降低一个 n 的复杂度? 用例题五的做法相当于要进行 n^2 轮 DP, 但每轮 DP 的不同点其实只在于有着不同的 p, q , 要限定 p 为白色点, q 为黑色点。所以每轮 DP 应当是“比较相似”的, 可以公用一些信息。于是我们考虑固定一个 q , 当 q 固定时不同点就只在于 p 了。但如果自下至上地 DP, 信息仍然有很大差异。转换思路考虑从上至下地 DP, 只要 p 在 u 子树内, 那么对 u 子树外所有点染色的 DP 信息就应是完全一样的, 可以共用这部分信息从而降低时间复杂度。

当然从上至下 DP 又会引出要除去其他信息可能时间复杂度不正确的问题。为此我们将一部分要除去的信息合并, 然后枚举各个孩子, 一边计算转移给这个孩子的信息, 一边除去这个孩子的信息为接下来的计算作准备。

事实上从转置原理的角度来看这样的方法是非常自然的, 因为从上至下 DP 可以看作是对经典的从下至上 DP 进行了转置, 这里我们描述的过程就是转置了从下至上 DP 时每次考虑合并一个孩子信息的过程。

6 利用优秀限制方向解决问题

即使有向图的结构比有向树更复杂, 如果限制方向具有优秀的性质, 使得可以以较为明确的顺序计算各个部分的拓扑序信息再对信息进行合并, 那么也可以在较为优秀的复杂度内解决问题。

本节将给出一道利用优秀限制方向解决问题的例题, 并将一道原本标算为 $O(n^3)$ 的题目通过转化到该例题将时间复杂度优化到 $O(n^2)$ 。

6.1 叶向边仙人掌拓扑序计数

边仙人掌的定义是无重边无自环且每条边至多属于一个简单环（不经过重复点的环）的连通图。而叶向边仙人掌类似叶向树，代表对边仙人掌的每条边定向，变为有向边仙人掌，但存在一个点 r 使得边的方向总是从 dis 较小的点指向 dis 较大的点的情况，请看例题：

例题九⁹：给定一个 n 个点 m 条边的边仙人掌以及根 r 。认为图是单位边权的，计算出图上每个点 u 到 r 的最短路 dis_u 。接下来将每条边 (u, v) 进行定向，由 dis 小的一方指向 dis 大的一方。保证不存在一条边连接的两个点 dis 相同。请问定向后的图有多少拓扑序？ $n \leq 5000, m \leq 2(n-1)$ 。

这个问题类似叶向树，考虑改进叶向树的做法，但如果我们仍然设 dp_u 表示 u 直接或间接可达的所有点以及这些点之间连边构成的子图的拓扑序数量的话，会遇到一个问题：设想一个大小为 4 的环，假设 $r = 1, 1 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 4, 3 \rightarrow 4$ 有边，并且 4 还指向了图的另外一些部分（如下图），此时 dp_3 的信息包含了 3, 4, 5, 6, 7 两个点， dp_2 的信息包含了 2, 4, 5, 6, 7 两个点，如果希望通过合并 dp_2, dp_3 来得到 dp_1 的话，点 4, 5, 6, 7 同时在两边出现，不能再用先前归并序列的角度来考虑问题了。

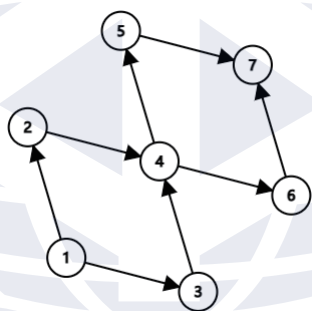


图 2: 一个上述想法不可行的例子

不过可以注意到，事实上这样做重复出现的点一定是一个环上唯一没有出度的点及其可达的点，所以对这个情况特殊处理应该还是比较简单的。

处理仙人掌的另一个问题是转移顺序并不明确，一种方法是对仙人掌建圆方树从而变成顺序更明确的树，不过这里我们会介绍另一种更直观的“收缩”方法。

我们注意到，对于一个点集 S ，如果对于任意 $u \in S$ ，其所有出边 $u \rightarrow v$ 都有 $v \in S$ ，且存在唯一的一个点 p 满足如果存在边 $q \rightarrow u$ ，要么 $q \in S$ ，要么 $q = p$ ，并且 p 直接或间接可达 S 集合所有点。那么可以删去 S 集合内所有点，记录这些点都“跟随”于 p 之后，在此后的计数过程中时，一旦考虑到 p ，就立即考虑如何排列“跟随”其的这些点，将方案数乘

⁹题目来源：校内模拟赛题

上点集 S 的拓扑序数量以及从此后拓扑序列选一些空位给这些点的方案数。我们称一次这样删去点集 S 的操作为针对点集 S 的一次“收缩”。

注意收缩掉点集 S 时， $u \in S$ 的点 u 可能也被一些点“跟随”，我们认为“跟随”其的点此后转而“跟随” p 。

事实上 2.1 介绍的树拓扑序计数问题的经典做法，就可以看作是在不断收缩叶子，直到最后只剩下根。

对于叶向边仙人掌，一定可以通过以下两种操作“收缩”到只剩根：

- 选一个点 u 以及其一条出边 $u \rightarrow v$ ，满足 v 只与 u 相连，收缩 v 。称这种操作是收缩叶子。
- 选一个点 u ， u 是某个包含其的简单环 C 中 dis 最小的点， $\forall v \in C \setminus \{u\}$ ，所有与 v 相连的边都在环 C 上，那么收缩环 C 除 u 以外的所有点。称这种操作是收缩环。

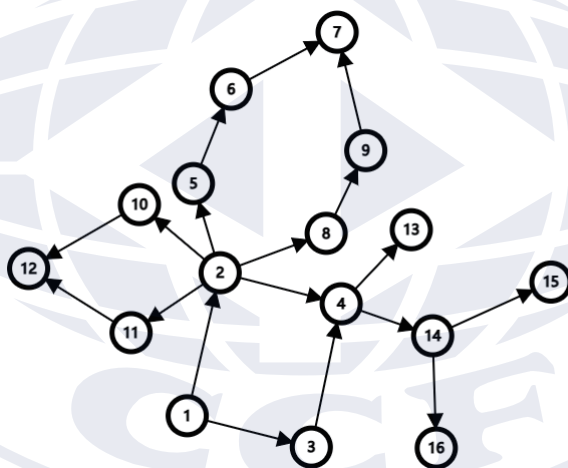


图 3: 一棵叶向边仙人掌

对于上图，一个可能的收缩过程是：选 $u = 14$ ，进行两次收缩叶子，分别收缩掉 15 和 16 两个点；选 $u = 4$ ，进行两次收缩叶子，分别收缩掉 13, 14 两个点；选 $u = 2$ ，进行收缩环，收缩掉 5, 6, 7, 8, 9 五个点；选 $u = 2$ 进行收缩环，收缩掉 10, 11, 12 三个点；选 $u = 1$ 进行收缩环，收缩掉 2, 3, 4 三个点。此时只剩下 1 这个根。

在收缩的过程中，我们维护 siz_u 表示 u 以及“跟随” u 的点的数量， ans_u 表示“跟随” u 的点以及 u 构成的点集的导出子图的拓扑序数量。初始显然所有 $siz_u = ans_u = 1$ 。

考察两种操作结束后对 siz_u 和 ans_u 的影响，可以发现 siz_u 总是会加上 S 集合内所有 siz 之和，主要需要考察的是对 ans 的影响。

- 对于收缩叶子操作,可以发现和叶向树等价,就相当于归并各个孩子对应子图的拓扑序列, ans_u 乘上 ans_v 以及 $\binom{siz_v + siz_u - 1}{siz_v}$ 。
- 对于收缩环操作,我们需要特殊处理。记环 C 的两端弧分别包含 $a_1, a_2 \dots a_k$ 以及 $b_1, b_2 \dots b_k$ 这些点, 满足 $a_1 = b_1 = u, a_k = b_k = v$, 设 $a_k = b_k = v$ 。考虑将拓扑序反过来, 一开始只考察 v 以及“跟随”于 v 的点怎么排, 维护指针 $i = j = k$ 。每次从 i, j 中选一个, 要么将 a_{i-1} 加在拓扑序最前面, 然后在拓扑序上插入所有跟随其的点, 要么选 b_{j-1} , 选完后对应指针减 1。注意只有 $i = j = 2$ 时才能加入 u , 并且加入 u 后立即停止。这里也需要一个 DP, 记 $dp_{i,j}$ 为 $a_i \sim a_k, b_j \sim b_k$ 这些点, 以及跟随这些点的点形成点集的导出子图的拓扑序数量, 每次要么加入 a_{i-1} 以及“跟随”其的点转移到 $dp_{i-1,j}$, 要么加入 b_{j-1} 转移到 $dp_{i,j-1}$ 。转移时也需要乘分配位置的组合数。进行一轮这样 DP 的复杂度是 $O(k^2)$ 的。

收缩到只剩根 r 时结束, 复杂度实际上是该边仙人掌所有简单环长的平方和, 而因为每条边至多属于一个简单环, 这不超过 m^2 。而边仙人掌事实上必定满足 $m \leq 2(n-1)$, 所以复杂度事实上是 $O(n^2)$ 的。

6.2 利用叶向边仙人掌的做法解决另一问题

例题十¹⁰: 给定一颗 n 个点以 1 为根的叶向树。定义一个排列 a 的权值 $f(a)$ 为 $\sum_{i=1}^{n-1} |a_i^{-1} - a_{i+1}^{-1}|$, 求原图所有合法拓扑序的权值之和。合法拓扑序的定义是所有是长度为 n 的排列 a 满足 $\forall 2 \leq u \leq n, a_u > a_{fa_u}$ 。 $n \leq 5000$ 。

这是例题五定义权值时对排列求逆的形式, 标算做法类似例题五的做法, 是枚举值域线, 最终时间复杂度为 $O(n^3)$, 这里我们给出一种截然不同的 $O(n^2)$ 做法。

权值是 $|a_i^{-1} - a_{i+1}^{-1}|$, 即对所有 $i \in [1, n-1]$, 计算值 i 所在点的编号到值 $i+1$ 所在点的编号之差, 然后取绝对值再求和。

如果我们能对所有点对 (u, v) , 算出 u 拓扑序上是 v 前驱的方案数 $res(u, v)$, 那么原问题就是求 $\sum_{u \neq v} res(u, v) |u - v|$ 。

我们根据 u, v 有无祖先后代关系分类讨论:

较为简单的情形是 u, v 有祖先后代关系, 那么当且仅当 u 是 v 父亲时 $res(u, v)$ 才不为 0。此时因为拓扑序上遇到 u 就必须立即选 v , 所以可以删去 v , 将所有 v 的孩子接到 u 上。考虑这样改造后的 $n-1$ 个点的树的合法拓扑序, 显然通过将 u 替换为 u, v 可以映射到原树的 u 必须是 v 前驱的合法拓扑序, 并且构成双射。改造后的树的拓扑序是易于求得的, 因为对 siz 序列的影响仅仅只有 u 到根的链上每个点 siz 减小 1, 以及删去了 siz_v 。

¹⁰ 题目来源: <https://qoj.ac/problem/3843>

更为复杂的情形是 u, v 没有祖先后代关系。我们连接边 $fa_v \rightarrow u$ ，并删去 v 点，将 v 的所有孩子接到 u 上。这样改造后的 $n-1$ 个点的图的合法拓扑序仍然与原树的 u 必须是 v 先驱的合法拓扑序构成双射，但问题在于因为多连了一条边，此时得到的不是树了。

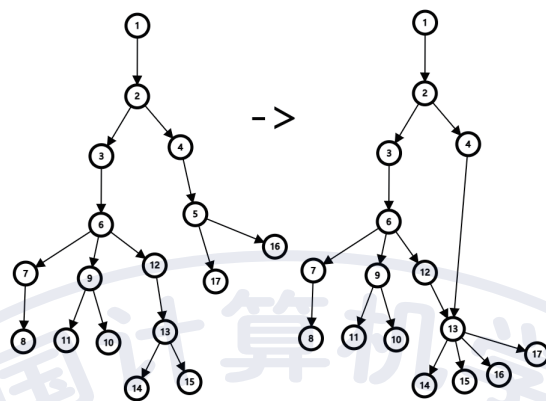


图 4: 当 $u = 13, v = 5$ 时对图进行的调整

不过可以注意到此时图的形态可以视为上文中提到的叶向边仙人掌，因为虽然此时环上边的方向不一定是按照到根的距离定向，但仍然保持是以环上距离根最近的点作为分界点，一段弧上的边为顺时针方向，一段弧上的边是逆时针方向，并且最近的点可以到达环上所有点。而且还具有更好的性质：一定只有一个环。

设 $w = \text{lca}(u, v)$ ，环距离根最近的点一定是 w ，从 w 向上的部分一定只有树边。所以如果能算出 w 下面部分的拓扑序数量，调整到整张图的拓扑序数量是简单的，只需不断进行归并标号即可，可知整张图的拓扑序数量一定是 w 下面部分的乘上只与 w 子树外部分相关的一个系数 prd_w 。

一个问题在于，在叶向边仙人掌一节中我们的做法是从环上距离根 dis 较大的部分向 dis 较小的部分 DP 的（也就是环上节点是按照拓扑序从大往小加入的），而现在如果我们想 $O(n^2)$ 求出所有 $res(u, v)$ ，显然需要设计一个从环上 dis 较小部分向 dis 较大部分 DP 的算法。

而之所以之前我们选择了按照拓扑序从大到小加入的顺序，是因为加入一个点时需要考虑“跟随”其的所有点，按照拓扑序从大到小的顺序便于我们将这些“跟随”点与已被考虑的点进行归并。

那么反转这个过程，按照拓扑序从小到大的顺序加入环上所有节点，就应该加入一个环上节点时从已经归并好的序列中拆分出所有这个点的“跟随”节点。即假设拓扑序有 s 个空位，初始 $s = \text{siz}_w - 1$ ，每次加入环上一个节点 u ，并且“跟随”其的有 k 个节点的话，首先 u 一定占据最前面的空位，然后要从剩下 $s - 1$ 个空位中选出 k 个分配给所有“跟随”其的节点，然后空位减少到 $s - 1 - k$ 个。

具体来说，我们这样进行 DP：设 $dp_{p,q}$ 表示两段环上可以加入的下一个点分别是 p, q

的方案数，当然只考虑 w 向下的部分。两种方案被认为是不同的当且仅当 p, q 直接或间接可达的点在拓扑序上占据位置构成的集合不同，或者存在 p, q 均不可达的点在拓扑序上占据的位置不同。

边界是 p 为 $w \rightarrow u$ 在原树路径上第二个点， q 为 $w \rightarrow v$ 在原树路径上第二个点，此时需要给 w 除 p, q 以外所有孩子分配空位。设 ord_u 为原树 u 子树拓扑序数量， w 除 p, q 有孩子 $v_1 \sim v_k$ ，那么 $dp_{p,q} = \prod_{i=1}^k ord_{v_i} \binom{siz_w - 2}{siz_{v_1}, siz_{v_2}, \dots, siz_{v_k}, siz_p + siz_q - 1}$ 。

对于一般的转移，如果 $fa_p \neq w$ ，那么 $dp_{p,q}$ 可能由 $dp_{fa_p,q}$ 通过在环上加入 fa_p 来得到，设 fa_p 除了 p 有孩子 $v_1 \sim v_k$ ，那么 $dp_{p,q}$ 加上 $dp_{fa_p,q} \prod_{i=1}^k ord_{v_i} \binom{siz_{fa_p} + siz_q - 2}{siz_{v_1}, siz_{v_2}, \dots, siz_{v_k}, siz_p + siz_q - 1}$ ，即给孩子 $v_1 \sim v_k$ 分配空位。如果 $fa_q \neq w$ ，也有对称的转移。

考虑最后如何计算答案，显然只需用 $dp_{u,u}$ 的值，再给 u, v 所有孩子分配空位即可（因为 v 的孩子被转接给 u 了），最后要乘上系数 prd_w 。

这样就顺利得到了环上节点按照拓扑序从小到大顺序加入的 DP 做法。虽然对于不同的 u, v ，环的形态有所不同。但当 p, q 确定时，无论怎么选 u, v ，只要 p 是 u 的祖先， q 是 v 的祖先， $dp_{p,q}$ 的值都一致，所以可以共用 DP 信息。唯一需要稍作修改的是这样 $dp_{u,u}$ 信息可能不存在，要用 $dp_{u,v}$ 。

进行一些预处理后转移都可以做到 $O(1)$ ，因而时间复杂度为 $O(n^2)$ 。

7 更为精巧地应用容斥原理

本节内容是第四节关于容斥原理部分的推进，通过更为精巧地应用容斥原理，我们可以对一般有向边仙人掌以及有向广义串并联图进行拓扑序计数。

7.1 一般有向边仙人掌

即只保证将所有有向边变为无向边后，图是一个边仙人掌。

例题十一¹¹：给定一张 n 个点 m 条边的有向图，保证将所有有向边视为无向边后，图是一个边仙人掌。请问原图有多少拓扑序？ $n \leq 500$ ， $m \leq 2(n-1)$ 。

叶向边仙人掌中，边的方向有很好的性质，便于我们进行“收缩”操作。一般有向边仙人掌则失去了这个性质，不过类比叶向树到一般有向树的拓展，我们应该仍能用容斥原理来解决该问题。

借助容斥原理，我们可以对每条边的方向设定一个预期，如果真实方向符合预期，那么就什么都不干。否则，可以把这条方向不符预期的边，拆成一条无限制边和符合预期的边（并带有一定容斥系数）。

¹¹ 题目来源：原创

任意定根 r , 我们可以把仙人掌上的边分为两类: 在环上的边, 称为环边; 不在任意环上的边, 称为树边。

还是求出 r 到每个点 u 的最短路 dis_u , 我们这样设定每条边的预期:

- 对于树边, 预期方向是两个点中 dis 较小的指向较大的。
- 对于环边, 我们依次考虑每个简单环 C , 显然这样每条环边会被考虑恰好一次。记该环上 dis 最小的点是 u , 从 u 开始顺时针考虑环上的每条边, 一开始我们设定预期方向是顺时针方向。每考虑一条边, 一定会拆出一条顺时针方向的边, 可能还有一条无限制边, 我们从这两条拆出的边 (可能只有一条) 中选择一条作为本组方案内这条边的限制类型。一旦我们选出了一条无限制边, 从环上的下一条边开始, 以预期方向为逆时针方向拆边, 直到所有边都被拆开。

拆完边后, 最后将所有拆边方案得到的图的拓扑序数乘上对应容斥系数加和即可。

对于环上的边, 我们实际上动态调整了我们的预期方向。这样做有什么好处呢? 注意到如果一组方案中存在一个简单环, 环上的所有边均为顺时针方向, 那么显然对应图拓扑序数为 0。否则这组方案中所有环都存在一条分界边 E , E 本身是无限制边, 在 E 前面的边均为顺时针限制边, 在 E 后面的边要么是无限边, 要么是逆时针限制边。

可以发现, 这样一来, 只考虑环上边的话, 环上所有点的入度都不超过 1, 且 u 入度必定为 0。并且根据对树边的容斥, 一个点至多被一条树边指向。再根据仙人掌图的结构, 一个点 p 如果可能被树边指向 (即从 r 到 p 的所有最短路最后一条边都是树边), 那么包含 p 的所有环, p 都是 dis 最小的点, 从而不可能被环边指向。

综合以上几点, 可以发现这样容斥后每个点入度不超过 1, 并且不存在环, 于是我们将问题转化为了叶向森林拓扑序计数。

考虑如何实现该容斥, 我们仍然可以借助叶向边仙人掌的“收缩”顺序来进行转移, 为此我们稍微修改一下“收缩”的定义:

对于一个点集 S , 如果存在一个唯一的点 p , 使得对于任意 $u \in S$, 所有与其相关的边 $u \rightarrow v$ 或 $v \rightarrow u$, 满足要么 $v \in S$, 要么 $v = p$ 。那么可以应用容斥原理, 将所有与任意 $u \in S$ 相关的边 $u \rightarrow v$ 或 $v \rightarrow u$ 最终容斥为 p 是一个根节点的叶向森林。删去 S 集合内所有点, 记录这些点都“跟随”于 p 之后。我们称一次这样删去点集 S 的操作为针对点集 S 的一次“收缩”。

仍需动态维护 siz_u , 修改 $ans_{u,i}$ 的定义为所有容斥方案, 使得“跟随”于 u 的点集 S_u 构成的叶向森林中, u 所在树有 i 个点的 $\frac{\text{该方案容斥系数}}{\prod_{v \in S_u} siz'_v}$ 之和, siz'_v 代表该容斥方案中 v 的子树大小。

仍然考察收缩叶子操作与收缩环操作对 ans 的影响:

- 对于收缩叶子操作，首先此时 v 在容斥方案中的子树大小一定已经确定，于是给 $ans_{v,i}$ 先乘上 $\frac{1}{i}$ 。然后考察 u, v 之间连边的方向，如果本就是 $u \rightarrow v$ 那直接背包合并即可；否则要拆成容斥系数为 -1 的 $u \rightarrow v$ 和一条无限制。这一步的复杂度是 $O(siz_u siz_v)$ 。
- 对于收缩环操作，在这一轮操作中所有 $\forall v \in C \setminus \{u\}$ ， siz'_v 被确定，要乘上这些 $\frac{1}{siz'_v}$ 。记这个简单环上的点按顺序是 $a_1, a_2, \dots, a_k, a_{k+1}$ ，其中 $a_1 = a_{k+1} = u$ 。枚举分界边 E 是 (a_i, a_{i+1}) 间的连边，显然只有这条边方向是 $a_{i+1} \rightarrow a_i$ 才可行。枚举 j 从 $i+1$ 到 k ，维护 f_s 表示所有容斥方案（只考虑 a_{i+1} 到 a_j 间所有边如何容斥）， a_j 子树大小为 s 的权值和，每次先合并上 a_j 的其他孩子，即与 $ans_{a_j,*}$ 背包合并，然后将 f_s 乘上 $\frac{1}{s}$ 。接着考虑 a_j, a_{j+1} 间的连边在容斥方案中选择哪种方向，对 f 进行对应修改。同理再枚举 j 从 i 到 1 ，维护 g_s 表示所有容斥方案（只考虑所有 a_j 到 a_i 间所有边如何容斥）， a_j 子树大小为 s 的权值和，转移也是类似的，但注意边必须选顺时针方向，并且最后 $\frac{1}{siz_u}$ 不用乘。最后对 f, g 背包合并得到 u 真实的子树大小对应的 DP 值，对所有 i 加和得到新的 ans_u 。

最后收缩到只剩点 r 时停止，最终答案是所有 $\frac{ans_{r,i}}{i}$ 之和乘上 $n!$ 。

分析时间复杂度，定义一个局面的势能是所有点 siz_u^2 之和，也就是对每个点，“跟随”其的点数 +1 的平方和。

初始局面势能为 n ，终止局面势能为 n^2 。收缩叶子操作消耗 $O(siz_u siz_v)$ 时间且使势能上升 $2siz_u siz_v$ ，收缩环操作消耗 $O(|S|((\sum_{p \in S} siz_p)^2 - \sum_{p \in S} siz_p^2))$ 时间，且使势能上升 $(\sum_{p \in S} siz_p)^2 - \sum_{p \in S} siz_p^2$ ，因此时间复杂度为 $O(n^3)$ 。

7.2 有向广义串并联图

广义串并联图是对于任意四个节点都不存在 6 条两两没有公共边的路径连接这 4 个节点中的每一对节点的无向连通图，不过在本文中我们只需要知道这样的图一定可以通过删一度点操作、缩二度点操作、叠合重边操作转化为只剩一个点的图即可。这一点的证明及更多关于广义串并联图的论文可见参考文献 [1]。

具体描述一下三种操作，分别是：

- 删一度点操作：如果存在点 u 度数恰为 1，删去它以及与它相邻的边。
- 缩二度点操作：如果存在点 u 度数恰为 2，设它的两个邻居是 v, w ，删去边 $(u, v), (u, w)$ ，加入边 (v, w) ，然后删去点 u 。
- 叠合重边操作：如果存在两条边 (u, v) ，将这对重边叠成一条边。

可以发现前文提到的树、仙人掌都是广义串并联图，但存在广义串并联图既不是树也不是仙人掌，所以这个问题显然更复杂。

接下来我们研究一下如果有向图在将所有边视为无向边后是广义串并联图，该如何高效地拓扑序计数。

例题十二¹²：给定一张 n 个点 m 条边的有向图，保证将所有有向边视为无向边后，图是广义串并联图。请问原图有多少拓扑序？ $n \leq 100, m \leq 2(n-1)$ 。

方便起见，我们认为视为无向边后图是无重边无自环的。因为方向相同的重边只需保留一条，方向相反的重边和自环会导致答案直接为 0。

这样一来，叠合重边操作一定立即发生于缩二度点操作之后，只有在 v, w 之间本来就有边时需要进行一次叠合重边操作。

我们还是希望应用容斥原理将问题转化到叶向森林拓扑序计数，也即需要通过容斥保证每个点的入度不超过 1，并且要确保不存在环。考虑如何设定容斥的预期方向，一个简单的想法是当要删去点 u 时，确定其所有邻边的预期方向为指向 u 。这样每个点只有在被删去时才可能增加入度，所以入度总是 ≤ 2 ，但我们希望的是入度 ≤ 1 。

显然只有缩二度点时才会出现入度 = 2 的情况，此时是容斥后两个邻居 v, w 均存在 $v \rightarrow u, w \rightarrow u$ 的边。但我们注意到如果此时 v, w 之间存在边，比方说存在 $v \rightarrow w$ 的边，那么就可以删去 $v \rightarrow u$ 的边，因为只要有 $v \rightarrow w, w \rightarrow u$ 的边， $v \rightarrow u$ 的限制是自然成立的，这样一来 u 的入度就变为 1 了。

但本来 v, w 之间可能不存在边，为了解决这一情况，我们可以创造两个子问题，这两个子问题中分别有 $v \rightarrow w$ 的边和 $w \rightarrow v$ 的边，只要分别解决两个子问题然后将答案加起来就得到了 v, w 之间没有边的初始情况的答案了。

考虑如何将上述分析写成 DP 的形式：首先任何时刻有 m 条边的图事实上都存在 2^m 个子问题，即 m 条边每条的定向选择哪一种导出的 2^m 种情况。对每条边维护 $dp_{id,0/1,i,j}$ 表示若将第 id 条边 = (u, v) 定向为正/反向，在容斥拆边过程中，使得最终叶向森林 u 子树大小恰增加 i ， v 子树大小恰增加 j 的所有方案的权值和。注意这里的 i, j 都是这条边带来的影响，对应的点事实上是在先前的三种操作过程中被收缩到这条边上的点。再维护 $val_{u,i}$ 表示不考虑所有边带来的子树大小增量，只考虑删一度点操作对 u 子树大小影响的话，容斥拆边过程中 u 子树大小为 i 的所有方案的权值和。

当然也需要维护 dp_{id} 的 $i + j$ 最大值 sz_{id} ，可以发现这个值就是 (u, v) 这条边上收缩了多少点；以及 val_u 的 i 最大值 Sz_u 。

初始所有 $val_{u,1} = 1$ ，代表每个点自身， $Sz_u = 1$ ；所有边都是 $dp_{i,0,0,0} = 1$ ，即边的初始方向有 1 的方案， $sz_i = 0$ 。我们永远能删去任何一个度数 ≤ 2 的点：

- 删孤立点：只需将 $\sum_{i=1}^{Sz_u} \frac{val_{u,i}}{i}$ 直接乘入答案即可。

¹²题目来源：原创

- 删一度点操作：设 u 唯一的邻居是 v , (u, v) 的编号是 id 。首先应该在删去点 u 时结算 $\frac{1}{siz_u}$ 的权值, u 的子树大小 siz_u 应当是 (v, u) 这条边的影响加上 val_u 本身的价值, 这里需要将 $dp_{id,0/1}$ 在 i 这一维上与 val_u 做背包合并。然后考虑删去 u 可能使 v 的子树大小增加对 val_v 产生影响, 对于边方向是 $v \rightarrow u$ 的子问题无需容斥, 是 v 的子树大小增加 u 的子树大小, 而对于边方向是 $u \rightarrow v$ 的子问题需要拆成一个无限制 (那对 v 的子树大小就没有影响了) 和一个 $v \rightarrow u$ 。不过即使是拆成无限制, 边对于 v 的子树大小增量此时也是要叠加到 val_v 上去的。这里先是要 $O(siz_{id}^2 S_{zu})$ 计算边和 val 对 u 子树大小增量的总和, 然后要 $O(siz_{id}(siz_{id} + S_{zu})S_{zv})$ 将边对 v 的贡献, v 本来的贡献, 以及可能的 u 在叶向树上是 v 儿子的贡献三者合并。最后 S_{zv} 增加 $S_{zu} + siz_{id}$ 。
- 缩二度点操作：设 u 的两个邻居是 v, w , (u, v) 的编号是 id_1 , (u, w) 的编号是 id_2 , (v, w) 的编号为 id_3 。 (v, w) 边此时可能不存在, 需要额外新建。如果要新建, $dp_{id_3,0,0,0} = dp_{id_3,1,0,0} = 1$, 且 $siz_{id_3} = 0$ 。类似地也需要在此时结算 $\frac{1}{siz_u}$ 的权值, u 的子树大小此时要考虑 $(u, v), (u, w)$ 两条边以及 val_u 本身。考虑先将 $dp_{id_1,0/1}$ 在 i 这一维上与 val_u 背包合并并将 val_u 的信息合并到 dp_{id_1} 上, 这消耗 $O(S_{zu} siz_{id_1}^2)$ 时间, 注意 0/1 两种方向都要合并, 此时我们可以认为 siz_{id_1} 增加了 S_{zu} 。这样就只需考虑 $(u, v), (u, w)$ 两条边, u 的子树大小直接是两条边对 u 子树大小的增量之和。还需要考虑删去 id_1, id_2 两条边后它们的信息需要传递给 id_3 , 先固定 id_3 的方向, 在这种方向前提下枚举 id_1, id_2 的 $2^2 = 4$ 种方向, 先结算 $\frac{1}{siz_u}$ 的贡献, 然后根据各自的方向进行容斥, 得到容斥后这两条边总共给 v 子树大小增加 i' , w 子树大小增加 j' 的所有容斥方案的权值和, 这一步消耗 $O(siz_{id_1}^2 siz_{id_2}^2)$ 时间。最后将这部分信息与 id_3 原本的信息背包合并, 这一步消耗 $O((siz_{id_1} + siz_{id_2})^2 siz_{id_3}^2)$ 时间。最终 siz_{id_3} 增加 $siz_{id_1} + siz_{id_2}$ 。

分析时间复杂度, 还是设势能为 $\sum_{u=1}^n S_{zu}^2 + \sum_{i=1}^m siz_i^2$, 初始势能为 0, 最终势能不超过 n^2 。可以发现合并过程中消耗的时间要么是 $O(ne)$ 的, 要么是 $O(e^2)$ 的, 其中 e 是一轮合并上升的势能, 所以时间复杂度是 $O(n^4)$ 的。

如果允许使用 NTT, 可以将缩二度点时的复杂度降为 $O(siz_{id_1} siz_{id_2} n \log n)$ 和 $O((siz_{id_1} + siz_{id_2}) siz_{id_3} n \log n)$, 从而做到 $O(n^3 \log n)$ 的时间复杂度。

8 结语

本文由浅至深地介绍了解决树拓扑序计数相关问题的各种方法, 并通过十二道例题展现了这些方法的具体应用。这些方法与例题一部分是对前人的结果进行相对系统性的总结, 一部分是笔者做出的全新的探索。但本文给出的做法与思路未必是完美的, 可能仍有改进的空间; 各种扩展例题之间可能也可以相互组合, 创造出全新的题目。这些可能性还要等待后人来探索!



9 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练彭思进、杨耀良的指导。

感谢父母对我的培养和教育。

感谢学校的栽培，李建老师的教导和同学们的帮助。

感谢章绍嘉同学、孙恒喆同学与我进行交流讨论，并为本文审稿。

感谢吴泽岳同学为本文的编写提供技术支持。

感谢其它所有本文所参考过的资料的提供者。

感谢各位百忙之中抽出宝贵的时间来阅读本文。

10 参考文献

- [1] 吴作同. 《《公园》命题报告》，IOI2019 中国国家队候选队员论文集。



论现代硬件上的常数优化

成都外国语学校 宋佳兴

摘要

本文深入探讨了单线程算法的优化思想和方法。鉴于上一篇关于同一主题的集训队论文已经是十年前的了，本文的内容与当前的硬件环境更为贴近。本文涵盖了 x86 架构与 Coffee Lake 微架构简介、通用优化方法、计算密集型算法、数据结构等多个方面，从汇编代码、硬件特性等底层概念出发，对常数优化进行了定性和定量的研究。

前言

随着计算机硬件的发展，渐近复杂性已经不再占据绝对的地位，相同的时间复杂度可能会产生很大的速度差异。此外，硬件的并行能力在不断增强，高速缓存和主存之间的速度差距也在扩大，因此底层优化的提升空间越来越大。很多时候，底层优化可以弥补算法上的先天不足，从而实际提升竞赛成绩。在竞赛之外，我们可以通过这种方式更深入地理解计算机，提高代码质量。

在竞赛中 SIMD 的使用是不允许的，所以本文并未涉及到 SIMD。然而，这些优化思想和方法仍然可以应用于 SIMD 的环境中。

我的电脑与 OI 赛场的评测机 i7-8700k 具有相同的微架构，因此在相同的主频下，它们的速度接近。程序的运行环境如下：

CPU 微架构	Coffee Lake
时钟频率	4.4GHZ 不变
主存	32GB 2933MHz
操作系统	Linux
编译器	GCC 9.5
编译选项	-O2

1 x86 架构与 Coffee Lake 微架构

指令集架构 (Instruction Set Architecture) 定义了机器语言的含义，它是硬件执行机器代码的基础。其中，x86-64 是最广泛使用的指令集架构，它的存在使得各种不同的硬件设备能够理解和执行相同的机器代码。

另一方面，微架构 (Microarchitecture) 定义了硬件执行机器指令的具体细节，它决定了硬件的性能。每一种微架构都有其独特的设计和优化，这些因素共同决定了处理器的性能。

Coffee Lake 是 Intel i7-8700k 处理器的微架构，我们选择它作为研究对象。

1.1 汇编语言

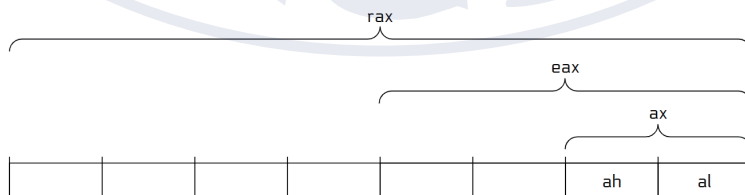
学习汇编的目的并不是为了手动编写它，因为在 O2 优化下，手写汇编的提升空间相当有限。我们的真正目标是能够理解编译器生成的汇编代码，知道程序运行的原始操作是什么，以及优化的可能性在哪里。一旦熟悉了汇编，我们就可以预想出 C++ 代码在编译后的样子。

汇编语言 (Assembly Language) 和机器代码几乎是一一对应的，可以被视为程序运行的原始操作。现在，让我们进入正题。

1.1.1 寄存器

寄存器 (Register) 是计算机中的一种重要存储单元，它们用于存储和操作数据。在汇编语言中，寄存器可以被视为具有名称的变量。通用寄存器有 16 个：rax, rbx, rcx, rdx, rsp, rbp, rsi, rdi, r8, r9, r10, r11, r12, r13, r14, r15。

通用寄存器都是 64 位无符号整数，为了支持 32, 16, 8 位的运算，每个寄存器的低 32, 16, 8 位和 8~15 位都可以当成一个完整寄存器用，如图所示。



mov 指令用于将一个值（可以是常量、寄存器中的值或内存中的值）复制到寄存器中。这是汇编语言中最常用的指令之一：

```
mov rax, 0x3f3f3f3f
mov rbx, 114514
mov al, 114
mov rcx, rax # let rcx = rax
mov edx, -1 # let edx = 0xffffffff
```

大多数算术运算的指令都很简单，如下所示。

汇编	C++
add rax, rbx	rax += rbx
sub ebx, ecx	ebx -= ecx
imul rsi, rdi	rsi *= rdi
neg rax	rax = -rax
not rax	rax = ~rax
and rax, rbx	rax &= rbx
or rax, rbx	rax = rbx
xor rax, rbx	rax ^= rbx
shl rax, 10	rax <<= 10
shr rax, 10	rax = (unsigned long long)rax >> 10
sar rax, 10	rax = (long long)rax >> 10

1.1.2 地址解引用

解引用是一种操作，它允许我们访问存储在特定地址上的值。在 C++ 中，数组访问就是一种解引用的例子。

最简单的解引用 `rbx = *(int*)rax` 所对应的汇编代码为：

```
mov rbx, DWORD PTR [rax]
```

解引用地址的一般的用法为：

```
SIZE PTR [base + index * scale + displacement]
```

base 和 index 都是 64 位寄存器，scale 和 displacement 都是常量，其中 scale 只能为 1,2,4,8 之一，每个部分都可以省略。这种格式的设计是为了提供灵活性，减少计算地址的开销。

SIZE PTR 描述解引用的数据大小，有以下选择：

BYTE PTR	8 位
WORD PTR	16 位
DWORD PTR	32 位
QWORD PTR	64 位

解引用不是只能用在 mov 指令中，请看下面的例子：有一个 int 数组首地址为 rax，把下标为 rbx 的元素加上 ecx，汇编如下：

```
add DWORD PTR [rax + rbx * 4], ecx
```


1.1.3 跳转

在汇编语言中，我们使用跳转指令来实现条件分支和循环。这些指令允许我们根据特定的条件改变程序的执行流程。来看一下这段 C++ 求和代码编译后的结果：

```
int sum(int* begin, int* end) {  
    int res = 0;  
    while (begin != end) res += *begin++;  
    return res;  
}
```

汇编中函数有一个规定：每个参数依次存放在 rdi, rsi, rdx, rcx, ... 中，返回值存进 rax，使用 ret 指令返回¹。以下为 sum 函数的汇编：

```
sum:  
    xor     eax, eax  
    cmp     rdi, rsi  
    je      end  
loop:  
    add     eax, DWORD PTR [rdi]  
    add     rdi, 4  
    cmp     rsi, rdi  
    jne     loop  
    ret  
end:  
    ret
```

在这段汇编中，je end 是条件跳转指令，它允许我们在两个值相等时跳转到程序的另一部分。cmp 指令用于比较两个值，并将结果存储在 FLAGS 寄存器中，然后，我们可以使用条件跳转指令（如 je）来根据比较的结果改变程序的执行流程。

1.1.4 获得编译结果

工具 compiler explorer² 可以人性化地显示编译器生成的汇编。不了解的汇编指令可以在参考资料 [6] 中查询。

1.2 指令流水线

指令流水线 (Instruction Pipelining) 是一种技术，它使 CPU 能够同时执行多条指令。这是通过将每条指令的执行过程分解为几个阶段来实现的，每个阶段都在同时处理多条指令。

执行任何机器代码都需要经过以下阶段：

1. 从内存中获取一大块机器代码。

¹函数调用约定 <https://stackoverflow.com/questions/18024672>

²<https://godbolt.org>

2. 在大块代码中分离出每条指令，并将其转换为微指令 (Micro-operation)。
3. 微指令进行寄存器重命名 (Register Renaming / Allocation)。
4. 微指令进入调度器 (Scheduler, Reservation Station), 输入操作数已经可用的微指令会被优先发送到执行端口。
5. 微指令进入执行单元进行执行。
6. 结果被临时存储，后续的微指令可以直接获取所需的数据。
7. 所有微指令按照最初的顺序，将结果写回寄存器。

前三个阶段被称为 CPU 的前端，后四个阶段被称为 CPU 的后端。CPU 前端的速度相对固定，而后端通常有提升空间，只有当指令以合适的顺序到达时，CPU 后端才能忙碌起来。

影响流水线速度的限制因素主要有三类：

- 结构风险 (Structural Hazards): 同一类型的指令过多，相应的执行单元不足。
- 数据风险 (Data Hazards): 当前指令需要等待上一条指令的结果才能开始执行。
- 控制流风险 (Control Hazards): 遇到跳转指令时，CPU 不能暂停等待结果，否则会导致流水线停顿。因此 CPU 会使用分支预测器 (Branch Predictor) 预测接下来的走向，并提前将指令放入流水线。如果分支预测未命中，就需要清空流水线。

控制流风险是最严重的，我们用一个例子来说明：

```
int foobar(bool a[100000]) {  
    int cnt0 = 0, cnt1 = 0;  
    for (int i = 0; i < 100000; i++)  
        if (a[i] == 0) cnt0++;  
        else cnt1++;  
    return cnt0 ^ cnt1;  
}
```

- 如果 $a_i = i \bmod 2$ ，那么分支预测器可以发现 if 语句是 01 交替的，此时不会发生控制流风险，foobar 函数会运行 $34 \mu\text{s}$ 。
- 如果 a_i 从 $\{0, 1\}$ 中随机选取，那么分支预测器就无能为力了，此时会发生控制流风险，foobar 函数会运行 $310 \mu\text{s}$ 。

数据风险通常发生在指令形成依赖链时，例如，一条指令的结果被用作下一条指令的输入：

```
int v0 = 1, v1 = 2, v2 = 3, v3 = 4;  
for (int i = 0; i < 1000000000; i++)  
    v1 += v0, v2 += v1, v3 += v2, v0 += v3;
```

这里第 3 行的四条指令就必须严格按顺序执行，而且 $v0 += v3$ 执行完了下一轮的 $v1 += v0$ 才能开始，因此 CPU 根本无法并行。如果把第 3 行改为：

```
v0 += v0, v1 += v1, v2 += v2, v3 += v3;
```

那么数据风险就几乎没有了，速度是原来的 2.9 倍，此时问题变成了结构风险，可以通过循环展开解决，详见 2.1。

1.3 Coffee Lake 微架构

x86 指令集极其庞大，导致制作工艺复杂。相比之下，RISC（精简指令集计算机）采用了一个更小但功能齐全的指令集，这大大简化了处理器的设计，从而提高了性能。然而，x86 指令集不能舍弃对旧软件的兼容性。为了在保证兼容性的同时提高性能，处理器内部采用了一个精简的微指令集。CPU 的前端部分负责将内存中的机器代码翻译成微指令 (Micro-operation, 简称 μop)，然后提供给后端执行。这样，处理器制造商就可以通过优化 CPU 的前后端来提高性能。

微架构定义了 x86 指令如何被翻译成微指令（以下简称 μop ），这些 μop 如何使用执行端口，以及执行所需的时间。OI 赛场的评测机 i7-8700k 采用了 Coffee Lake 微架构，因此，本文的所有性能分析都将基于 Coffee Lake 进行。

1.3.1 延迟与吞吐量

首先，我们需要理解 x86 指令的速度并不是一个可以用单一数字表示的概念。在流水线环境中，速度与并行程度有关。然而，我们可以通过考虑两个极端情况来理解一条指令的速度：

- 延迟 (Latency)：这是一条指令从开始执行到得到结果所需的时间。在完全串行的情况下，它可以被理解为每条指令的执行时间。
- 吞吐量倒数 (Reciprocal Throughput)：在同一类型的指令完全并行执行的情况下，它是每条指令的平均执行时间。

这里的时间是以时钟周期 (clock cycle) 数量来衡量的，时钟周期是 CPU 活动的一个周期。通常所说的主频就是每秒能进行多少个时钟周期。在后文中，我们将使用平均每次操作的时钟周期数 cycles 来描述一种操作的常数因子。例如，平均每次操作 0.5 cycle 意味着这种操作每秒大约能执行 9×10^9 次。测量时钟周期的方法在 6 中。

在 Coffee Lake 微架构上，一些指令的延迟和吞吐量倒数信息如下：

指令	操作数	延迟	吞吐量倒数	备注
add, sub	r, r/i	1	0.25	r 表示 8~64 bit 的任何寄存器, i 表示常量 (immediate value)
and, or, xor	r, r/i	1	0.25	加法和减法与位运算相比没有差距, 都是每个周期可以并行执行 4 条指令
shl	r, i	1	0.5	

1.3.2 微指令与执行端口

每条 x86 指令等价于一条或多条微指令 (μop), 每条 μop 会进入一个特定的执行端口 (Execution Port)。执行端口和执行单元的区别在于, 执行端口内部包含了一个小型流水线, 这个流水线又包含了若干个执行单元。因此, 即使上一个 μop 还在执行, 每个周期都可以有一个新的 μop 进入同一个执行端口。

在 Coffee Lake 微架构上, 共有 8 个执行端口, 分别命名为 p0, p1, ..., p7。其中, p0~p5 都包含算术逻辑单元 (ALU), 都可以完成 add 和 sub 指令。但是, 它们的其他功能却略有不同。p23 是加载端口, 用于将内存中的值加载到寄存器中; p4 是存储端口, 用于将值存储到内存中; 而 p7 则是一个功能较弱的辅助端口。

在 x86 指令到 μop 的翻译过程中, 还存在一种称为融合微指令 (fused- μop) 的概念。fused- μop 和 μop 的区别在于, 某些 fused- μop 实际上是 μop 的二元组。x86 指令首先会被翻译成 fused- μop , 在确定了执行端口后, 融合的 μop 二元组才会被分开, 分别进入不同的执行端口。fused- μop 的存在使得 CPU 的前端可以更快地向后端提供 μop 。

一些指令的 fused- μop 、 μop 和端口需求如下:

指令	操作数	吞吐量倒数	fused- $\mu\text{op}/\mu\text{op}$	端口需求
add, sub	r, r/i	0.25	1/1	p0~p5 (任选一个)
add	r, [address]	0.5	1/2	p23 + p0~p5
mov	[address], r	1	1/2	p23 + p4

可以发现吞吐量倒数可以由端口需求算出来, 比如端口需求为 p23 + p0~p5, 那么一个周期最多能执行两条指令, 不然 p23 就不够用了。因此通过端口需求可以定量地分析指令。

CPU 前端每个周期最多提供 4 条 fused- μop , 因此 fused- μop 的数量除以 4 是一个必须的时间。CPU 前端想要跑满 4 fused- $\mu\text{ops}/\text{cycle}$ (4 fused- μops 每周期) 很简单, fused- μops 是有缓存单元的, 名叫 DSB (Decode Stream Buffer), 可以记忆最近执行过的 fused- μops , 只要它能工作, 就不需要从内存中读取 x86 指令并翻译成 fused- μop 了。一个不长也不短的循环 (8~1000 fused- μops) 迭代很多次大约就是 4 fused- $\mu\text{ops}/\text{cycle}$, 里面不能有一些极其复杂的指令 (比如除法), 也不能发生跳转 (可以包含条件跳转但不能真的跳转)。

最后, x86 指令信息可以在参考资料 [13] 和 [14] 中查询。查询和流水线模拟工具的详细信息, 请参见6。

1.3.3 寄存器重命名

在 Coffee Lake 微架构中, 有 180 个物理通用寄存器, 调度器 (Scheduler, Reservation Station) 的容量为 97, 调度器会把输入操作数已经可用的 fused- μ ops 发送到执行端口。然而, 在汇编中只有 16 个通用寄存器, 这似乎使得大量的物理寄存器和调度器容量被浪费。

实际上, 处理器会通过寄存器重命名 (Register Renaming / Allocation) 来充分利用物理寄存器。每个汇编中的逻辑寄存器都对应一个物理寄存器, 这种对应关系可以动态变化。请看下面的例子:

```
mov eax, DWORD PTR [rdi]
imul eax, 2023
mov DWORD PTR [rdi], eax
mov eax, DWORD PTR [rsi]
imul eax, 2023
mov DWORD PTR [rsi], eax
```

这段代码在做两件无关的事情, 但是因为它们共用 `eax`, 似乎就只能顺序执行。然而, 实际上, `eax` 所指代的物理寄存器发生了变化, 这两件事情使用了不同的物理寄存器, 因此可以并行。

寄存器重命名的工作原理非常简单: 每当逻辑寄存器被写入或修改时, 它就会指代一个新的物理寄存器, 这有点像可持久化数据结构。因此, `mov eax, DWORD PTR [rsi]` 的执行并不会依赖于前面的任何指令。

除了上面的例子, 寄存器重命名还有其他的好处:

- ```
add DWORD PTR [rdi], eax
add eax, 1
```

在这里, `add eax, 1` 不必等待上一条指令执行完毕, 因为 `add` 只会把结果写入一个新的物理寄存器, 不会修改原来的物理寄存器。

- 一个每次迭代都无关的循环可以具有并行多轮迭代的能力, 因为使用同一个寄存器不会产生依赖, 而且调度器容量很大。一个特别的例子是线段树中一次操作还没有进行完就可以同时开始下一次操作。
- 可以利用寄存器重命名把 `mov rax, rbx` 消除掉, 只需要让 `rax` 指代 `rbx` 所指代的物理寄存器。但是, 这仍然会占用 CPU 前端的带宽。



| 指令  | 操作数            | 延迟 | 吞吐量倒数 | fused- $\mu\text{op}/\mu\text{op}$ | 端口需求  | 备注         |
|-----|----------------|----|-------|------------------------------------|-------|------------|
| mov | r32/64, r32/64 | 0  | 0.25  | 1/0                                |       | 不指代同一物理寄存器 |
| mov | r32/64, r32/64 | 1  | 0.25  | 1/1                                | p0156 | 已经是同一物理寄存器 |

## 1.4 高速缓存与内存

程序使用的临时存储包含若干层，每一层有不同的大小和速度。越往上，访问速度越快，但大小也越小。我的电脑上各层的大小如下（和 OI 赛场的评测机是一样的）：

- L1 缓存 (L1 cache) 由指令缓存 (L1i) 和数据缓存 (L1d) 组成，每核的 L1i 和 L1d 都是 32KB。
- L2 缓存 (L2 cache)，每核 256 KB，不分数据和指令。
- L3 缓存 (L3 cache)，6 核共享 12 MB。
- 主存 (RAM) 32 GB。

当数据被访问时，它所在的缓存行会被移入 L1 缓存，在长时间不访问后依次退回到 L2, L3 和主存。这种设计使得访问越频繁的数据平均开销越小，因为越频繁访问的数据更可能在高速缓存中。

### 1.4.1 缓存行

缓存与主存间数据传输的基本单位不是字节，而是缓存行 (Cache Line)。在 x86 架构下，缓存行是首地址为 64 倍数的 64 字节段，缓存里存的都是若干个缓存行。这意味着，如果请求的数据不在 L1 缓存时，就需要将整个缓存行移入 L1 缓存。

在瓶颈为内存的程序中，缓存行未命中的数量是重要的信息，如何测量它请参见6。

### 1.4.2 延迟和吞吐量

和 x86 指令的延迟和吞吐量倒数一样，延迟是一个请求从开始到完成的时间，吞吐量倒数是并行情况下的一个请求的平均时间。

在我的电脑上测量结果如下（仅供参考）：



| 时钟周期 | 延迟  | 吞吐量倒数（读） | 吞吐量倒数（写） |
|------|-----|----------|----------|
| L1   | 5   | 0.5      | 1        |
| L2   | 12  | 1.7      | 3.6      |
| L3   | 41  | 3.95     | 6.55     |
| 主存   | 240 | 12.4     | 20.8     |

可以发现，写入的吞吐量几乎总是读入的两倍。在 L1 上，这是执行端口的缘故。除此之外，都是缓存的写入策略导致的：如果写入请求未命中缓存，那么先将数据移动到缓存，然后仅修改缓存。这会导致各级缓存和主存中的数据不一致，因此缓存中每个缓存行都带有一个 dirty bit 记录它是否修改过，当一个 dirty bit 为 1 的缓存行被替换出去的时候，需要将它写回比它大一级的缓存或者主存中去，写入请求比读取请求会多做一件事：把 dirty bit 设置为 1，这在将来会触发一次写入。

写入策略带来的好处是：无论先读再写还是先写再读，都只有第一次未命中。

### 1.4.3 替换策略

请求在缓存未命中时，需要选择一个原来的缓存行，用新的缓存行替换它，这里怎么选择决定了缓存的性能。

最流行的选择是最近最少使用策略 (Least recently used, 简称 LRU)，它相当于可用缓存减半时的最优策略，有如下定理<sup>3</sup>：

**定理.** 设  $OPT_M$  表示可用缓存为  $M$  个缓存行时，理论最优解会有多少次缓存行未命中， $LRU_M$  表示 LRU 会有多少次缓存行未命中，那么：

$$LRU_M \leq 2OPT_{M/2} \quad (1)$$

但事实上，硬件并没有真正的实现 LRU，而是分组关联 + PLRU，以 L1 缓存为例：

- L1 数据缓存有 32 KB，因此有 512 个缓存行。
- 这 512 个缓存行被分成 64 组，每组包含 8 个缓存行。
- 每组的 8 个缓存行使用 Tree-PLRU，一种近似的 LRU，见参考资料 [9]

如何确定一个地址属于哪个组呢？理想的方式是用哈希函数把地址映射到缓存组，但是计算哈希太慢了，硬件中无法接受。64 位地址被分成 3 个部分，tag, index 和 offset，见下图，index 决定了这个缓存应该放在哪一个组中，tag 则用于在同一个组中区分地址。

<sup>3</sup>来源 <https://en.algorithmica.org/hpc/external-memory/policies/>

| tag     | index  | offset |
|---------|--------|--------|
| [12,63] | [6,11] | [0,5]  |

没有哈希函数就容易受到不良访问模式的影响：当地址以  $2^k$  为步长递增时，只有一小部分的组可用，比如访问的地址为  $0, 256, 512, \dots$  那么它们只会映射到  $\frac{1}{4}$  的组。

在 OI 中会遇到的情况是：二维数组第二维是  $2^k$ ，比如状压 DP，固定第二维下标不变，第一维下标变化时，地址就是以  $2^k$  为步长递增，导致缓存不能有效工作。解决的办法是把  $2^k$  这一维开得大一点，比如多开 256 Byte，这样可以避免地址冲突，使得缓存可以更加有效地工作。

#### 1.4.4 分页

分页是实现虚拟内存的技术，从 CPU 所使用的虚拟地址转化到物理地址，需要将高 52 位替换。需要知道的是：如果请求命中了 L1 缓存，那么不需要知道物理地址，否则，需要将  $\lfloor address/4096 \rfloor$  进行一个映射，转译后备缓冲器 (TLB) 是加速这个映射的缓存系统。如果 TLB 未命中，那么就需要从主存中查找对应的物理地址，速度非常有限。

一种受影响的情况是：有一些分散的缓存行，它们的总量适合 L2 缓存，但是因为  $\lfloor address/4096 \rfloor$  互不相同，所以 TLB 无法存下它们所有。

#### 1.4.5 内存优化原则

- 空间局部性：把一起使用的数据在内存中安排在一起，使得它们位于一块或者两块缓存行中。因此，数据结构节点信息的首选是结构体，而不是多个数组。
- 时间局部性：让数据在较小时间段内重复访问。典型的例子是滚动数组和矩阵分块乘法。
- 内存级并行：如果提前知道所有的内存地址，内存请求就可以并行。不能内存级并行的例子是链表。
- 读取快于写入：例子是 DP 的填表法比刷表法快。
- 避免有害现象：地址以  $2^k$  为步长递增导致缓存大部分不可用，以及访问太分散导致 TLB 未命中。

## 2 通用优化方法

### 2.1 循环展开技术

循环展开 (Loop Unrolling) 是一种提升程序性能的简单而有效的方法。

#### 2.1.1 int 类型求和

考虑以下的 C++ 代码：

```
for (int i = 0; i < 2000; i++) sum += a[i];
```

编译后的汇编代码如下：

```
loop:
 add eax, DWORD PTR [rdi]
 add rdi, 4
 cmp rdi, rdx
 jne loop
```

这个循环每次迭代至少需要 1 个 cycle，主要有三个瓶颈：

- 前两条 add 指令都依赖于上一轮的同一条语句。
- cmp 和 jne 两条语句只生成了一条  $\mu\text{op}$ （因为这个组合太常见了，所以它被设计出来提高性能），但是这个  $\mu\text{op}$  只能进入 p6 端口，因此每周期只能执行一条。
- 因为每个周期都要跳转，所以 CPU 前端也只能达到 3 fused- $\mu\text{ops}/\text{cycle}$  的速度。

虽然这三个瓶颈都使得每次迭代至少需要 1 cycle，但实际情况是 CPU 无法实现最优的调度，所以每次迭代实际上需要 1.4 cycles。

理想的循环长度为 8 到 1000 个 fused- $\mu\text{ops}$ ，这样 CPU 前端可以达到 4 fused- $\mu\text{ops}/\text{cycle}$  的速度，同时也减少了循环变量和跳转的开销。我们可以通过循环展开来实现这个目标：

```
for (int i = 0; i < 2000; i += 8)
 sum += a[i], sum += a[i + 1], ..., sum += a[i + 7];
```

展开后重新编译，得到的汇编代码如下：

```
loop:
 add eax, DWORD PTR [rdx]
 add rdx, 32
 add eax, DWORD PTR [rdx-28]
 add eax, DWORD PTR [rdx-24]
 ...
 add eax, DWORD PTR [rdx-4]
 cmp rcx, rdx
 jne loop
```

这时，只剩下一个瓶颈：add eax, [ ... ] 指令构成了依赖链。实际的平均时间达到了 1 cycle，刚好卡在这个瓶颈上。

最后一步是消除这个依赖链：

```
int s0 = 0, s1 = 0, s2 = 0, s3 = 0;
for (int i = 0; i < 2000; i += 8) {
 s0 += a[i + 0], s1 += a[i + 1], s2 += a[i + 2], s3 += a[i + 3];
 s0 += a[i + 4], s1 += a[i + 5], s2 += a[i + 6], s3 += a[i + 7];
}
int sum = s0 + s1 + s2 + s3;
```

现在，依赖链被打破了，唯一的瓶颈在 p23 端口，实际平均时间达到了 0.5 cycles。虽然只使用两个求和变量也能达到这个速度，但依赖链刚好也成了一个 0.5 cycles 的瓶颈，同时存在两个瓶颈会造成性能损失。

### 2.1.2 对 int 类型整体加上一个数

考虑以下的 C++ 代码：

```
for (int i = 0; i < 2000; i++) a[i] += x;
```

编译后的汇编代码如下：

```
loop:
 add DWORD PTR [rdi], esi
 add rdi, 4
 cmp rdi, rax
 jne loop
```

每次迭代的平均时间为 1.34 cycles，因为它有多个 1 cycle 的瓶颈，无法平衡这些瓶颈就会 > 1 cycle。具体瓶颈如下：

- 超短循环本身的缺点：循环变量加法的依赖、跳转、CPU 前端。
- add DWORD PTR [rdi], esi 每周期只能执行一条，因为它有一个  $\mu\text{op}$  只能进入 p4 端口。

| 指令  | 操作数                | 吞吐量倒数 | fused- $\mu\text{op}/\mu\text{op}$ | 端口需求                    |
|-----|--------------------|-------|------------------------------------|-------------------------|
| add | [address], r32/i32 | 1     | 2/4                                | p23 + p0156 + p237 + p4 |

循环展开 4 步就足够了，因为 fused- $\mu\text{ops}$  达到了 8。然后就只受限于 add [address], r32 的吞吐量，从而达到 1 cycle 的平均时间。

```
for (int i = 0; i < 2000; i += 4)
 a[i] += x, a[i + 1] += x, ..., a[i + 3] += x;
```

### 2.1.3 计算 int 类型的前缀和

考虑以下的 C++ 代码：

```
for (int i = 0; i < 2000 - 1; i++) a[i + 1] += a[i];
```

这里有一个冷知识，前缀和写  $a[i + 1] += a[i]$  比  $a[i] += a[i - 1]$  更好。O2 对后者的优化是不到位的。这个问题在 GCC 12 才得以修复，NOI-Linux 中的 GCC 9 是未修复的版本。

对于 fused- $\mu$ ops 不足 8 的超短循环，循环展开肯定有效果，但实际上没有，原来是编译器没有生成预期的汇编代码。

换了一种写法后，就达到了 1 cycle 的平均时间。

```
int sum = 0;
for (int i = 0; i < 2000; i += 4) {
 sum += a[i + 0], a[i + 0] = sum;
 ...
 sum += a[i + 3], a[i + 3] = sum;
}
```

解释一下为什么速度能达到 1 cycle， $sum += a[...]$ ， $a[...] = sum$  会生成如下汇编：

| 指令                         | fused- $\mu$ op/ $\mu$ op | 端口需求        |
|----------------------------|---------------------------|-------------|
| add eax, DWORD PTR [ ... ] | 1/2                       | p23 + p0156 |
| mov DWORD PTR [ ... ], eax | 1/2                       | p237 + p4   |

这两条语句可以分成三个部分：加载值 (p2)，加法 (p0156)，存储值 (p237 + p4)，只有加法的阶段构成了依赖链，但是执行加法只需要一个周期，而另外两个阶段都是完全并行的，执行端口刚好够用。

### 2.1.4 自动展开功能

手动循环展开可能会比较麻烦，因为复制粘贴后还需要修改下标。这里，我将介绍两种循环展开的简写方法。

需要注意的是，循环展开有一定的副作用，它会增大机器代码的长度，从而影响 CPU 前端获取机器代码的速度。因此，我们不建议使用编译选项进行全局展开，而是只对程序中的瓶颈循环进行展开。

### 2.1.5 使用 pragma 命令

在想要展开的 for 循环前面加上一句 `#pragma GCC unroll n`，其中 `n` 是一个数字，表示展开循环的步长。示例如下：

```
#pragma GCC unroll 4
for (int i = 0; i < 2000; i++) a[i] += x;
```

这段代码相当于：

```
for (int i = 0; i < 2000; i += 4)
 a[i] += x, ..., a[i + 3] += x;
```

特别地，当循环次数为常数时，可以彻底消除循环：

```
#pragma GCC unroll 10
for (int i = 0; i < 10; i++) a[i] += x;
```

### 2.1.6 使用 C++ 模板函数

由于竞赛通常会禁用 `pragma` 命令，所以我们还有第二种方法，即定义一个模板函数来辅助完全展开：

```
template <int R, int L = 0, int M = (L + R) / 2, class F>
__attribute__((always_inline)) inline void UL(int i, F lambda) {
 if (R - L == 1) lambda(i + L);
 else UL<M, L>(i, lambda), UL<R, M>(i, lambda);
}
```

使用方法如下：

```
for (int i = 0; i < 2000; i += 4)
 UL<4>(i, [&](int i) { a[i] += x; });
```

不必担心匿名函数的调用开销，因为编译器足够聪明。

## 2.2 循环融合

将需要访问同一数组的循环融合在一起是一种简单而有效的优化方法，其目标是减少内存读写次数。

例如，如果我们需要对数组 `a` 进行两次前缀和计算，常规的写法可能如下：

```
int sum0 = 0;
for (int i = 0; i < 2000; i++)
 sum0 += a[i], a[i] = sum0;
int sum1 = 0;
for (int i = 0; i < 2000; i++)
 sum1 += a[i], a[i] = sum1;
```

根据上一节的内容，如果我们进行循环展开，`sum0 += a[i], a[i] = sum0` 的平均时间为 1 cycle，瓶颈在依赖性和内存写入操作。

然而，如果我们将这两个循环融合在一起，写法如下：



```
int sum0 = 0, sum1 = 0;
for (int i = 0; i < 2000; i++)
 sum0 += a[i], sum1 += sum0, a[i] = sum1;
```

这样就减少了一半的内存读写次数。如果再加上循环展开，平均每次 `sum0 += a[i]`, `sum1 += sum0`, `a[i] = sum1` 就只需要 1.06 cycles，比融合前快了近一倍。

```
int sum0 = 0, sum1 = 0;
#pragma GCC unroll 32
for (int i = 0; i < 2000; i++)
 sum0 += a[i], sum1 += sum0, a[i] = sum1;
```

这里我们要展开 32 步，因为 CPU 在实现最优调度时可能会遇到困难，通过增大步数可以使平均时间更接近 1 cycle。

在 OI 中，循环融合可以应用于动态规划和数据结构。例如，当我们有以下 DP 转移方程时：

$$dp_{i,j} = \min\{dp_{i-1,j-1} + w_1, dp_{i-1,j-2} + w_2, dp_{i-1,j-3} + w_3\} \quad (2)$$

一个较快的写法是单独处理  $dp_{i,0}$ ,  $dp_{i,1}$ ,  $dp_{i,2}$ ，然后使用一个循环处理后面的。在这种情况下，循环展开可以带来好处，因为每个  $dp_{i-1,j}$  都会被连续使用 3 次，没有必要多次读取内存，循环展开可以帮助编译器更好地做这一优化。

## 2.3 去除分支

分支是指跳转指令。分支预测机制相当智能，当分支模式的循环节不超过 200 时，预测成功率几乎可以达到 100%。此外，嵌套循环和 `switch` 语句也能被成功预测。在预测成功的情况下，如果跳转条件为真，需要 1~2 cycles，而如果为假，平均需要 0.5 cycles。

对于总是执行或者不执行的 `if` 语句，我们可以大胆地使用它们。由于跳转比不跳转需要更多的时间，我们可以使用 `__builtin_expect` 函数来告诉编译器 `if` 语句的执行倾向，以便进行优化：

```
if (__builtin_expect(x < y, 0)) x = y; // 几乎总是不执行
if (__builtin_expect(x > y, 1)) x = y; // 几乎总是执行
```

分支预测器发现循环节的能力是可以被利用的，比如非递归线段树每次操作的循环次数都是固定的，就可以精准预测。

然而，分支预测未命中的情况也很常见，大部分算法中的 `if` 语句都是不可预测的，此时需要花费 15~20 cycles。因此，如果 `if` 语句后面的语句很少，执行的 cycles 不多，那么我们应该考虑去除这些分支。

### 2.3.1 cmov 指令

请看下面的代码：

```
for (int i = 0; i < 100; i++) a[i] = max(a[i], b[i]);
```

这段代码在编译后并没有生成跳转指令，而是使用了 `cmovge` 指令，含义是只有在大于等于的情况下才会进行赋值。对应的汇编代码如下：

```
mov edx, [address of ai]
cmp [address of bi], edx
cmovge edx, [address of bi]
mov [address of ai], edx
```

如果我们换一种写法：

```
void checkMax(int& a, int b) { if (a < b) a = b; }
for (int i = 0; i < 100; i++) checkMax(a[i], b[i]);
```

那么编译器就会生成跳转指令。因此，如果我们想让编译器生成 `cmov` 指令，就需要注意代码的编写方式。以下是几种编译器通常会生成 `cmov` 指令的写法：

- 定义两个变量 `a, b`，计算好值后接一个和 `cmov` 功能相同的 `if` 语句：

```
int a = ..., b = ...;
if (a < b) a = b;
```

- 定义两个变量 `a, b`，计算好值后使用三目运算符：

```
int a = ..., b = ...;
int c = a > b ? a : b;
```

- 使用 `min/max` 函数。

`cmov` 系列指令有很多，大多数的性能相同，但有两个例外：

| 指令                         | 操作数                         | 延迟 | 吞吐量倒数 | 端口需求  |
|----------------------------|-----------------------------|----|-------|-------|
| 大部分 <code>cmov</code>      | <code>r32/64, r32/64</code> | 1  | 0.5   | p06   |
| <code>cmova, cmovbe</code> | <code>r32/64, r32/64</code> | 1  | 1     | 2*p06 |

在 C++ 中，无符号的比较可能会生成较慢的 `cmova, cmovbe`，也可能生成较快的 `cmovae, cmovb`，这取决于写法，可以通过检查汇编代码来确认。从 GCC 12 开始，就不会再生成较慢的指令了。

### 2.3.2 位运算

- `&&` 和 `||` 会生成分支，如果这个分支是不可预测的，可以替换为位运算 `&, |`。
- 将一个数和 0/-1 进行 `and` 运算可以实现筛选，例如：

```
int mask = ... ? -1 : 0;
a[0] = max(a[0], b[0] & mask); // mask = -1 时才会 checkMax
a[1] = max(a[1], b[1] & mask);
```

- 使用 `__builtin_ctz` 遍历 `mask` 中的所有 1，可以用来优化状压 DP。

```
while (mask) {
 int i = __builtin_ctz(mask);
 ...
 mask &= mask - 1;
}
```

- 另一个位运算替代分支的技巧，见参考资料 [7]。

### 2.3.3 比较运算符

仅仅使用比较运算符而没有 `if` 是不会生成分支的，比如 `a = x < y` 会生成 `set` 系列指令。`set` 指令信息和 `cmov` 完全相同，也是 `seta` 和 `setbe` 较慢。

特别地，无符号的 `a + (c < d)` 和 `a - (c < d)` 还可以生成较快的 `adc` 和 `sbb` 指令。

## 2.4 使用 long long 替代 pair

由于 64 位的指令通常和 32 位一样快，因此使用 `long long` 替代 `pair<int, int>` 来执行某些操作会更快：

- 加载或存储一个 `long long` 比加载或存储两个 `int` 快。
- `long long` 的比较等价于 `pair<int, uint32_t>` 的比较，而 `uint64_t` 则等价于 `pair<uint32_t, uint32_t>`，这样可以去除分支。另外，`__int128` 的比较也是无分支的。

这在数据结构维护最小值的位置（如 `dijkstra`）和 DP 记录方案时非常有用。

- `long long` 的加法可以模拟两次 `int` 的加法，前提是没有负数且没有溢出。例如：

```
c[i][0] = c[i * 2][0] + c[i * 2 + 1][0];
c[i][1] = c[i * 2][1] + c[i * 2 + 1][1];
```

可以替换为

```
((long long)c[i]) = *((long long*)c[i * 2]) + *((long long*)c[i * 2 + 1]);
```

这还可以用于优化无负数无溢出的求和场景，从而获得两倍的速度。

在强制以 `long long` 解引用时，地址最好是 8 的倍数，否则数据可能跨越两个缓存行，这样会使速度慢一倍。可以使用 `alignas` 说明符<sup>4</sup>定义对齐的数据。

## 2.5 提高内存访问效率

### 2.5.1 空间局部性

空间局部性是一种定性的优化方法，其目标是让访问的数据尽可能在空间上连续，从而减少缓存行未命中的次数。

为了实现空间局部性，有两个需要做出选择的地方：

- 内存布局：对于高维数组，哪一维应该在前面？一组相关的数据应该用多个数组存储，还是用一个结构体数组存储？
- 访问顺序：当有多个 `for` 循环时，需要确定它们的顺序，以使地址尽可能连续。

在必要时，使用 `unsigned char` 或 `unsigned short` 来压缩空间也是一种方法，但这并不总是有效，因为 8 和 16 位的运算可能会带来性能损失，见参考资料 [8]。

例如，ST 表的预处理适合以下的内存布局和访问顺序：

```
int t[20][1 << 20];

for (int k = 0; k < log_n; k++)
for (int i = 2 << k; i <= n; i++)
 t[k + 1][i] = min(t[k][i - (1 << k)], t[k][i]);
```

区间 DP 的常规的写法可能如下：

```
for (int len = 1; len < n; len++)
 for (int l = 1; l <= n - len; l++) {
 int r = l + len;
 for (int k = l; k < r; k++)
 dp[l][r] <- dp[l][k] + dp[k + 1][r] + ...
 }
```

交换 `for` 循环的顺序可以使访问更连续：

```
for (int l = n; l >= 1; l--)
 for (int k = l; k <= n; k++)
 for (int r = k + 1; r <= n; r++)
 dp[l][r] <- dp[l][k] + dp[k + 1][r] + ...
```

<sup>4</sup><https://zh.cppreference.com/w/cpp/language/alignas>

## 2.5.2 时间局部性

时间局部性是让适合缓存的数据在一段时间内被反复访问。

最常见的例子是滚动数组，而滚动数组有两种写法：开两倍，开一倍（需要正确的转移顺序）。通常，开一倍会更快。

另一种方法是分块卡进缓存。例如，下面一段代码对  $a$  数组进行若干次区间求和：

```
int sum(int* a, int l, int r); // 对 a[l..r] 求和（循环展开）

int a[1000000], L[1000], R[1000], ans[1000];
memset(a, -1, sizeof a);
for (int i = 0; i < 1000; i++) ans[i] = sum(a, L[i], R[i])
```

$a$  数组太大了，只适合 L3 缓存。如果把  $a$  数组分割为若干适合 L1 的块，对每块扫 1000 遍再处理下一块，就达到了时间局部性的目的：

```
for (int k = 0; k < 1000000; k += 4000) // 处理块 [k, k + 3999]
 for (int i = 0; i < 1000; i++)
 ans[i] += sum(a, max(L[i], k), min(R[i], k + 3999));
```

在 OI 中，bitset 题<sup>5</sup>经常需要这样的分块操作（建议的每块的大小为 256/512 bit），一方面节省了空间，另一方面让数组适合 L3 缓存。

正确地选择块大小才能有效地利用缓存，对于 L1, L2 缓存，数据总量在缓存大小的 50% ~ 80% 比较合适，而 L3 缓存是共享的，最好不要超过  $\frac{1}{3}$ 。

还有一个方法是复用寄存器，减少内存读写的次数，比如  $1000 \times 1000$  的矩阵乘法。

```
for (int i = 0; i < 1000; i += 4)
 for (int k = 0; k < 1000; k += 2)
 for (int j = 0; j < 1000; j++)
 #pragma GCC unroll 4
 for (int ii = 0; ii < 4; ii++) {
 c[i + ii][j] += a[i + ii][k + 0] * b[k + 0][j];
 c[i + ii][j] += a[i + ii][k + 1] * b[k + 1][j];
 }
```

前两个 for 枚举了  $a$  矩阵中一个  $4 \times 2$  的小块，这 8 个元素会持久地保留在寄存器中。 $b$  矩阵读取的次数只有  $\frac{1000^3}{4}$ ， $c$  也只有  $\frac{1000^3}{4}$ ，而且  $c$  满足了时间局部性，一段时间内频繁访问的数据量适合 L1 缓存。这样就极大地缓解了内存的压力，平均每次  $c += a * b$  只有 1.15 cycle，而乘法的吞吐量倒数是 1 cycle。

## 2.5.3 内存级并行

内存请求的并行处理通常比串行处理要快得多，因此优先选择可以并行的算法。遍历链表和平衡树是不能并行的，而树状数组和非递归线段树可以并行，内存可以并行的条件是所有的地址都可以直接通过简单算术运算得到，不需要跟随指针。

<sup>5</sup>例如「JSOI2019」精准预测 <https://loj.ac/p/3101>

### 2.5.4 树重新编号

如果需要在树上进行多次动态规划或查询，重新编号树可以提高访问速度。不同的编号方式有不同的效果：

- 按 BFS 序重新编号后，记  $fa_u$  表示  $u$  的父亲，那么  $fa_u$  是一个单调不减的数组，所以在树上进行动态规划时访问连续，使得它比 DFS 序更适合动态规划。

自下往上的例子：

```
for (int u = n; u >= 2; u--) dp[fa[u]] = max(dp[fa[u]], dp[u] + 1);
```

自上往下的例子：

```
for (int u = 2; u <= n; u++) {
 int A = dp2[fa[u]], B = dp[fa[u]]; // 去除分支
 dp3[u] = max(dp3[fa[u]], dp[u] + 1 == B ? A : B) + 1;
}
```

如果需要从父亲枚举所有儿子，可以预处理  $R_u$  表示  $u$  的最后一个儿子的编号。

```
for (int i = 1; i <= n; i++) R[fa[i]] = i;
for (int i = 1; i < n; i++) R[i + 1] = max(R[i + 1], R[i]);
for (int u = n; u >= 1; u--) {
 sz[u] = 1;
 for (int v = R[u]; v != R[u - 1]; v--)
 sz[u] += sz[v];
}
```

如果树是通过 prufer 序随机生成的，枚举儿子的内层循环的分支预测未命中率会比较高，所以枚举儿子的写法效率并不高。

- 按 DFS 序重新编号可以支持快速遍历子树。
- 按重链剖分 DFS 序重新编号后，任意一条链可以拆成  $O(\log n)$  个区间，把这些区间的左右端点分别求出来，分别用 for 循环遍历，这样  $O(n^2)$  暴力可以处理更大的数据范围，可能得到更高的分数。

## 2.6 高效使用 STL

使用 STL 时，有时会遇到 TLE 的问题。然而，STL 中有许多函数是为了提高性能而设计的。在 [cppreference<sup>6</sup>](https://zh.cppreference.com/) 上，你可以了解到许多有用的函数。本节将总结这些函数的使用技巧，而不会详细介绍每个函数。对于不熟悉的函数，请参阅 [cppreference](https://zh.cppreference.com/) 进行查询。

<sup>6</sup><https://zh.cppreference.com/>



### 2.6.1 vector

- `reserve(n)`: 为 `vector` 预分配至少 `n` 的容量。当大小还不到 `n` 时, `push_back` 就不会重新分配内存。
- `emplace_back(...)`: 经过测试, 其速度不会慢于 `push_back`, 但是通常不会比 `push_back` 快, 只有像 `vector` 套 `vector` 这种复杂类型上 `emplace_back` 才有优势。
- 非成员函数 `move`: 如果你想将 `vector A` 赋值为 `B`, 并且 `B` 不再需要时, 可以使用 `A = move(B)`。这对其他 STL 容器也适用。

### 2.6.2 set

- `insert(x)`: 这个函数有返回值, 可以利用起来。
- `erase(it)`: 这个函数返回 `next(it)`, 并且复杂度为  $O(1)$ 。因此, 将 `s.erase(it++)` 改写为 `it = s.erase(it)` 会更好。当元素值域很小时, 可以开一个迭代器数组记录每个值的迭代器, 这样就可以通过迭代器删除元素, 复杂度为  $O(1)$ 。
- `insert(it, x)`: 当我们想插入 `x`, 并且知道插入的位置恰好在迭代器 `it` 之前时, 就可以使用这个函数。复杂度为  $O(1)$ , 返回值为插入元素的迭代器。如果 `x` 不应该插入在 `it` 之前, 复杂度退化到  $O(\log n)$ 。它的 `emplace` 版本为 `emplace_hint`。

两种典型的用法为:

- 从一些元素初始化 `set` 时, 可以先排序, 再连续调用 `s.insert(s.end(), x)`。
- 同时插入 `x` 和 `x-1`。

```
auto it = s.insert(x).first;
s.insert(it, x - 1);
```

- `it++`, `it--`: 常数很小的  $O(\log n)$ , 应当积极使用。

上述方法对 `map` 也适用。

### 2.6.3 priority\_queue

- `priority_queue` 的构造函数是  $O(n)$  的, 即线性构造堆算法。
- 两个 `priority_queue` 可以实现可删除堆, 避免使用 `multiset`。

## 2.6.4 unordered\_map

`__gnu_pbds::gp_hash_table` 的速度远快于 `unordered_map`，它唯一的不足是哈希函数不合理，在非随机数据下没有复杂度保证。因此，可以使用自定义哈希函数<sup>7</sup>来弥补这一点。

# 3 计算密集型算法

## 3.1 计数题通用优化

当模数为常量时，编译器会自动使用取模算法进行优化，因此这种情况下的程序会比需要输入模数的程序运行得更快。无论模数是否固定，减少取模次数都是优化的首选方法。

### 3.1.1 首选优化

以下是一些简单但普遍需要的优化方法：

- 当内层循环有公因式时，应将其提取出来。
- 只要在 `uint64_t` 的范围内，就不需要频繁地取模，例如 `a * b + c * d` 只需要在最后取模。
- 在模意义下进行加减运算时，不需要取模，只需要用 `if` 进行判断。值得一提的是，加法有一种替代写法。这种写法通常会稍微快一点，但有较低的概率会被编译成分支而不是 `cmov`，这时就会慢很多倍。

```
void inc(int& a, int b) {
 if (int(unsigned(a += b) - P) >= 0) a -= P;
}
```

### 3.1.2 使用无符号整数

使用 `uint64_t` 取模通常比使用 `long long` 快，因为处理负数取模相对复杂。以下是有符号取模和无符号取模的性能信息对比：

<sup>7</sup>推荐 <https://github.com/kth-competitive-programming/kactl/blob/main/content/data-structures/HashMap.h>

| 类型        | 延迟 | 吞吐量倒数 | fused-uops/uops | 端口需求                          |
|-----------|----|-------|-----------------|-------------------------------|
| 有符号（常量模数） | 10 | 2.5   | 10/8            | $2*p1 + p5 + 2*p06 + 3*p0156$ |
| 无符号（常量模数） | 9  | 2     | 7/6             | $2*p1 + p5 + p06 + 2*p0156$   |
| 有符号（变量模数） | 可变 | 24.38 | 57/56           | 很多                            |
| 无符号（变量模数） | 可变 | 21.04 | 36/33           | 很多                            |

有符号取模（常量模数）的吞吐量倒数受到前端限制，因为 fused-uops 太多。

无符号整数还有另一个优点：unsigned  $\rightarrow$  uint64\_t 的转换通常是免费的，而 int  $\rightarrow$  long long 的转换则不是，原因如下：

- int  $\rightarrow$  long long 的规则是非负数高 32 位补 0，负数补 1。这需要 movsxd 指令（性能信息和 add 一样）。
- unsigned  $\rightarrow$  uint64\_t 的规则是高 32 位补 0，由于每次修改 32 位寄存器时都会自动把高 32 位清 0，所以通常不需要用额外的指令清 0。

但是，uint64\_t  $\rightarrow$  unsigned  $\rightarrow$  uint64\_t 的转换并不是免费的，因为这需要把高 32 位清 0。为了避免这种转换，临时变量应始终使用 uint64\_t。

在下文中，我们约定 u32 = uint32\_t, u64 = uint64\_t, u128 = \_\_uint128\_t。

### 3.1.3 使用较大数据类型进行暂存

请看下面的代码：

```
const int P = 998244353;
u64 foobar(u32 a[2000], u32 b[2000]) {
 u64 res = 0;
 for (int i = 0; i < 2000; i++) res = (res + (u64)a[i] * b[i]) % P;
 return res % P;
}
```

一个简单的优化是使用 u128 来暂存结果。

```
u128 tmp = 0;
for (int i = 0; i < 2000; i++) tmp += (u64)a[i] * b[i];
return tmp % P;
```

u128 是用两个 64 位寄存器模拟出来的。加法需要 add 和 adc 指令，速度还是很快的。而取模则是调用库函数，虽然相对较慢，但是取一次模的代价还是值得的。

如果要计算 4 个数组的内积  $(\sum a_i b_i c_i d_i) \bmod P$ ，u128 就可能存不下，解决方案是每 128 次中断一次，对  $128P^4$  取一次模：

```
u128 tmp = 0;
for (int k = 0; k < 2000; k += 128) {
 for (int i = k; i < min(2000, k + 128); i++) {
 auto A = (u64)a[i] * b[i];
 auto B = (u64)c[i] * d[i];
 tmp += (u128)A * B;
 }
 const u128 _128P4 = (u128)4 * P * P * P * P;
 auto t = tmp - _128P4;
 if ((__int128)t >= 0) tmp = t;
}
return tmp % P;
```

这里的  $(u128)A * B$  看起来是非常慢的，其实不然。因为这个表达式会被编译成 `mul r64` 指令，该指令计算 `r64` 与 `rax` 的完整 128 位乘积，低 64 和高 64 分别放进 `rax`, `rdx`。

| 指令                   | 延迟 | 吞吐量倒数 | fused-uops/uops | 端口需求                 |
|----------------------|----|-------|-----------------|----------------------|
| <code>mul r64</code> | 4  | 1     | 2/2             | <code>p1 + p5</code> |

另一个例子是小型矩阵乘法，如果用 `u128` 的话就会进行  $n^2$  次 `u128` 取模，这样很慢，所以应该使用 `u64` 暂存结果，当最外层循环每迭代 8 次的时候就全体对  $8P^2$  取模。

### 3.1.4 打破取模链

取模操作的延迟很高，因此我们应该避免这样的取模链：

```
(u64)a * b % P * c % P * d % P * ...
```

我们可以将其替换为分治型合并：

```
// a * b * c * d
auto L = (u64)a * b % P;
auto R = (u64)c * d % P;
auto res = L * R % P;
```

对于超长的乘法链，我们可以使用循环展开来打破它。例如，计算阶乘可以这样写：

```
u64 tmp[8];
for (int i = 0; i < 8; i++) tmp[i] = 1;
for (u64 k = 1; k <= 1000000000; k += 8)
 #pragma GCC unroll 8
 for (int i = 0; i < 8; i++)
 tmp[i] = tmp[i] * (k + i) % P;
u64 ans = 1;
for (int i = 0; i < 8; i++) ans = ans * tmp[i] % P;
```

计算  $N!$  只需要  $3N$  cycles，瓶颈在端口 `p1`。在下一节 3.2 中，我们可以将其优化到  $N$  cycles。

### 3.1.5 求和与前缀和

在模意义下，求和与前缀和的操作都可以做得非常快，长度为  $N$  的数组求和用时  $0.26N$  cycles，前缀用时  $1.44N$  cycles。

求和可以使用 u64 来实现双倍的加载速度，从而突破速度瓶颈。而且，一次 u64 加法相当于两次 u32 加法，前提是结果不超过  $2^{32} - 1$ 。因此，我们可以每次加载 4 个 u64 并求和，然后再把高低 32 位分离开来。以下代码用时  $0.26N$  cycles。

```
u64 sum(u32 a[2000]) {
 assert((size_t)a % 8 == 0); // 确保定义时使用 alignas(8)
 u64 s0 = 0, s1 = 0;
 #pragma GCC unroll 32
 for (int i = 0; i < 2000; i += 8) {
 #define load(p) *(u64*)p
 u64 t = load(a[i]) + load(a[i + 2]) + load(a[i + 4]) + load(a[i + 6]);
 s0 += (u32)t, s1 += t >> 32;
 }
 return (s0 + s1) % P;
}
```

直接计算前缀和会很慢，因为模意义下的加法延迟较高而无法并行。为了实现并行，我们需要将数组等分成三个部分，通过提前对前两个部分求和，使得三个部分可以同时开始计算前缀和。下面的代码用时  $1.44N$  cycles（包含提前求和）。

```
void inc(u32& a, u32 b) {
 if (int((a += b) - P) >= 0) a -= P;
}

void psum(u32 a[2000]) {
 u32 s0 = 0, s1 = sum(a, 0, 666), s2 = sum(a, 666, 666*2);
 inc(s2, s1);
 #pragma GCC unroll 16
 for (int i = 0; i < 666; i++) {
 inc(s0, a[i]), a[i] = s0;
 inc(s1, a[i + 666]), a[i + 666] = s1;
 inc(s2, a[i + 666*2]), a[i + 666*2] = s2;
 }
 inc(s2, a[1998]), a[1998] = s2;
 inc(s2, a[1999]), a[1999] = s2;
}
```

## 3.2 内积优化

在模意义下，内积可以达到相当理想的速度（平均每次操作 1 cycle）。使用 SIMD，甚至可以达到  $1/6$  cycles。卷积和向量乘矩阵都是内积的形式，因此在许多情况下，可以替代 FFT 类算法骗分。

计算内积的最优实现方法是将每 16 个数分为一块，然后在块内使用 u64 暂存结果，最后再用 u128 求和。以下是相关的代码：

```
u64 iproduct(u32 a[2000], u64 b[2000]) {
 u128 res = 0;
 for (int i = 0; i < 2000; i += 16) {
 u64 s = 0;
 #pragma GCC unroll 16
 for (int j = 0; j < 16; j++) s += a[j + i] * b[j + i];
 res += s;
 }
 return res % P;
}
```

在 GCC 9 中, 长度为  $N$  的内积只需要  $1.02N$  cycles。特别注意,  $a$  和  $b$  这两个数组中至少有一个必须是  $u64$ , 否则实际性能会远低于理论性能。

从 GCC 11 开始, 编译的结果更加优秀, 至少有一个  $u64$  的情况下只需要  $1.01N$  cycles。即使都是  $u32$ , 通过对外层循环进行循环展开也可以达到  $1.04N$  cycles。在本节, 我们将会暂时使用 GCC 11 进行研究。

例题。(在线卷积)<sup>8</sup> 此题省略推导后, 有如下递推式:

$$A_i = \frac{1}{i} \left( 2 \sum_{j=1}^{i-1} j A_j A_{i-j} - (i-1) A_{i-1} \right) \quad (3)$$

$$n \leq 5 \times 10^4, 5.5s$$

令  $B_i = i A_i$ , 那么只需要对每个  $i$  计算  $\sum_{j=1}^{i-1} B_j A_{i-j}$ , 套用内积即可。需要决定一下用不用  $u64$ ,  $u64$  的内积速度更快, 但也会增加内存压力, 经过测试后, 不使用  $u64$  速度会更快。在本机用时 320ms。

```
a[1] = b[1] = 1;
for (int i = 2; i <= n; i++) {
 u128 sum = 0;
 #pragma GCC unroll 4
 for (int j = 0; j < i; j += 16) {
 u64 s = 0;
 #pragma GCC unroll 16
 for (int k = 0; k < 16; k++)
 s += (u64)b[j + k] * a[i - j - k];
 // a 有偏移量, 不会越界
 sum += s;
 }
 a[i] = (sum * 2 - b[i - 1]) * inv[i] % P;
 b[i] = (u64)a[i] * i % P;
}
```

例题。(多项式多点求值)<sup>9</sup> 给定  $n$  次多项式  $f(x)$ , 对  $i \in [1, m]$  计算  $f(x_i)$ 。

<sup>8</sup><https://codeforces.com/contest/1792/problem/F2>

<sup>9</sup><https://www.luogu.com.cn/problem/P5050>



$$n, m \leq 64000, 1.8s$$

$f(x) = \sum_{i=0}^n a_i x^i$  可以视作  $\{a_i\}$  和  $\{x^i\}$  的内积，可是后者无法直接算出，所以把系数分块：

$$f(x) = \sum_{k=0}^{\lfloor \frac{n}{256} \rfloor} x^{256k} \sum_{i=0}^{255} a_{256k+i} x^i \quad (4)$$

先把  $\{1, x, x^2, \dots, x^{255}\}$  算出来用 u64 存，然后把每个块与它做内积再合并。一个实现细节是 u128 的高效取模方法，详见代码。

在本机用时 1.14s。

```
// 多项式系数为 a[0..n]
// 求值点 b[0..m-1]
for (int k = 0; k < m; k++) {
 const int B = 256;
 u64 pows[B + 1] = {1};
 for (int i = 0; i < B; i++)
 pows[i + 1] = pows[i] * b[k] % P;
 u64 res = 0;
 for (int i = n & -B; i >= 0; i -= B) {
 u128 s0 = 0;
 #pragma GCC unroll 2
 for (int j0 = 0; j0 < B; j0 += 16) {
 u64 s1 = 0;
 #pragma GCC unroll 16
 for (int j1 = 0; j1 < 16; j1++)
 s1 += a[i + j0 + j1] * pows[j0 + j1];
 s0 += s1;
 }
 const u64 r64 = -(u64)P % P; // 2^64 % P
 u64 low = s0, high = s0 >> 64;
 if (low >= 8ULL * P * P) low -= 8ULL * P * P;
 res = (res * pows[B] + high * r64 + low) % P;
 }
 ans[k] = res;
}
```

这段代码还有两个优化方向：

- 加速预处理  $\{1, x, x^2, \dots, x^{255}\}$  的过程，因为这个过程是串行的，速度较为缓慢。
- 复用寄存器：扫描一遍系数的时候同时计算多个答案，这样可以减少加载指令。

**例题。**（快速阶乘算法）<sup>10</sup> 给定质数  $p$ ，计算  $n! \bmod p$ ，有  $T$  组数据。

$$n, p \leq 2^{32} - 1, T \leq 5, 2s$$

<sup>10</sup><https://www.luogu.com.cn/problem/P5282>

根据 Wilson 定理, 我们有

$$\begin{aligned}(p-1)! &\equiv -1 \pmod{p} \\ \prod_{i=1}^n i \prod_{i=1}^{p-1-n} (-i) &\equiv -1 \pmod{p} \\ n!(p-1-n)!(-1)^{p-1-n} &\equiv -1 \pmod{p}\end{aligned}\tag{5}$$

因此, 我们只需要选择  $n$  和  $p-1-n$  中较小的一个来计算阶乘, 这样就只有  $5 \times 10^9$  的计算量。

阶乘可以看作  $(x+1)(x+2)(x+3)\cdots(x+B)$  的多点求值, 套用一下就能过了。

值得一提的是, 还有一个可以加速 4.5 倍的改进: 对于每个  $i \leq [1, n]$ , 我们可以把它的 2, 3, 5, 7 的因子全部提出来, 这样每个数都变成了一个和 2, 3, 5, 7 互质的数。满足这个条件的取值只有  $N(1-\frac{1}{2})(1-\frac{1}{3})(1-\frac{1}{5})(1-\frac{1}{7})$  种。对于一个和 2, 3, 5, 7 互质的数  $x$ , 它的出现次数等于  $\lfloor \frac{n}{x} \rfloor$  以内只含有 2, 3, 5, 7 的数的数量。对于一个  $k$ , 出现次数恰好为  $k$  的数在一个区间中, 我们可以把区间的乘积算出来, 然后把乘积的  $k$  次方乘进答案里面。

### 3.3 整数除法

除法和取模指令一直以来都需要庞大的 CPU 资源, 而且会暂停 CPU 的前端。因此有各种各样的算法可以通过对除数预处理来大大加速之后的除法, 这些算法可以优化需要输入模数的程序、计算大量除法的数论算法。有的算法比编译器所用的方法还要高效, 在模数为常量的程序中也能发挥作用。

#### 3.3.1 Barrett reduction

首先  $\lfloor \frac{a}{m} \rfloor = \lfloor a \times \frac{1}{m} \rfloor$ , 把  $\frac{1}{m}$  近似为一个分数  $\frac{p}{2^k}$ , 为了让二者尽可能的接近,  $p$  应该等于  $\lfloor \frac{1}{m} \times 2^k \rfloor$  或者  $\lceil \frac{1}{m} \times 2^k \rceil$ , 选择哪一个之后再决定。

于是  $\lfloor \frac{a}{m} \rfloor \approx \lfloor \frac{a \times p}{2^k} \rfloor$ , 然后就可以愉快地用乘法和右移了, 那么问题来了, 它能算对吗?

**定理.** 设  $\frac{p}{2^k} = \frac{1}{m} + \epsilon$ ,  $\epsilon$  在不同范围内有不同的效果。

- 当  $|\epsilon| \leq \frac{1}{a}$  时,  $\lfloor \frac{a \times p}{2^k} \rfloor$  和  $\lfloor \frac{a}{m} \rfloor$  至多差 1, 而且  $\frac{p}{2^k} > \frac{1}{m}$  时答案只可能偏大,  $\frac{p}{2^k} < \frac{1}{m}$  时答案只可能偏小。
- 当  $0 \leq \epsilon < \frac{1}{am}$  时, 答案是准确无误的。

证明.

$$\begin{aligned}\lfloor \frac{a \times p}{2^k} \rfloor &= \lfloor a \times \left( \frac{1}{m} + \epsilon \right) \rfloor \\ &= \lfloor \frac{a}{m} + a\epsilon \rfloor\end{aligned}\tag{6}$$

只要  $|a\epsilon| \leq 1$ , 即  $|\epsilon| \leq \frac{1}{a}$  时, 答案就至多偏差 1。

什么时候答案是准确的呢,  $\frac{a}{m}$  加上非负的  $a\epsilon$  后要想改变整数部分,  $a\epsilon$  至少得为  $\frac{1}{m}$ , 只要让  $0 \leq a\epsilon < \frac{1}{m}$ , 整数部分就肯定不会改变, 即  $\lfloor \frac{a}{m} + a\epsilon \rfloor$  和  $\lfloor \frac{a}{m} \rfloor$  始终相等。□

那么定理第一部分怎么应用? 它可以用于取模, 假如  $\lfloor \frac{a}{m} \rfloor$  少算了 1, 那么  $a - \lfloor \frac{a}{m} \rfloor \times m$  仍然在  $[0, 2m)$  的范围内, 用一次 if 就可以搞定。在实践中, 我们通常会遇到  $10^9$  级别的模数  $m$ , 而  $a$  则是两个数的乘积, 即  $m^2$  级别, 我们令  $k = 64, p = \lfloor \frac{1}{m} \times 2^k \rfloor$ , 那么  $|\epsilon| < \frac{1}{2^{64}}$ , 对于  $[0, 2^{64})$  以内的  $a, m$  都是有效的。算法的关键步骤  $\lfloor \frac{a \times p}{2^{64}} \rfloor$  会将两个 u64 乘起来提取高 64 位, 可以通过 x86 的 `mul r64` 指令实现, 所以速度很快。而算法的最后一步“对  $[0, 2m)$  内的结果取模”可以选择去掉, 因为我们可以将所有数维持在  $[0, 2m)$  中, 加减法变为对  $2m$  取模。

接下来实现一个输入模数, 然后使用 Barrett reduction 取模的函数:

```
u64 p, m;
void set_mod(u32 mod) {
 assert(mod >= 2);
 m = mod, p = -m / m + 1; // 补码技巧计算 2^64 / m
}
u64 mul(u64 a, u64 b) {
 u64 c = a * b;
 c -= u64((u128)c * p >> 64) * m;
 if (c >= m) c -= m;
 return c;
}
```

定理第二部分就更有用了, 只不过条件有点苛刻。这也是编译器的默认取模算法, 我们了解一下编译器的实现。

```
u64 mod(u64 a) { return a % 998244353; }
```

根据定理, 我们需要让  $\epsilon < \frac{1}{(2^{64}-1) \times 998244353}$ , 令  $k = 93, p = \lfloor \frac{1}{m} \times 2^k \rfloor = 9920937979283557439$ ,  $\epsilon = \frac{p}{2^k} - \frac{1}{m}$  就足够小了, 编译器生成的汇编代码就这样做的!

```
rdi % 998244353 -> rax
movabs rax, -8525806094425994177 # =9920937979283557439
mul rdi
mov rax, rdx
shr rax, 29
imul rdx, rax, 998244353
mov rax, rdi
sub rax, rdx
```

在输入模数的情况下要用这个算法只能运行时决定  $k$ , 因为  $k$  过大会导致  $p$  就不能存进 u64, 过小又不能保证正确性。

定理第二部分的主要价值是计算除法, 在数论算法中会用到很多除法, 而且除数通常都不超过  $\sqrt{n}$ , 于是可以对  $[2, \sqrt{n}]$  预处理  $p$ , 来实现快速除法。通常选择  $k = 64$ , 这时的适用条件为  $am \leq 2^{64}$ 。示例代码如下:

```
u64 inv[32000];

void init(int n) {
 for (int i = 2; i <= n; i++) inv[i] = ~0ULL / i + 1; // ceil(2^64 / i)
}

u64 idiv(u64 a, u64 b) {
 // 确保 b != 1
 return (u128)a * inv[b] >> 64;
}
```

### 3.3.2 快速模乘算法

Barrett reduction 计算取模是间接的，意思是它是通过计算除法来得到余数的。还有直接计算余数的算法<sup>11</sup>，它比 Barrett reduction 更高效，可惜的是它不适合 OI 中  $10^9$  级别的模数，所以我提出了一个修改版本，让它  $10^9$  级别的模数上发挥作用。

首先

$$a \times z \bmod m = \{a \times \frac{z}{m}\} \times m \quad (7)$$

$$\{x\} = x - \lfloor x \rfloor$$

原理是  $a \times z \bmod m$  只和  $a \times \frac{z}{m}$  的小数部分有关。

和 Barrett reduction 一样，把  $\frac{z}{m}$  近似为分数  $\frac{p}{2^k}$ ，其中  $p = \lceil \frac{z}{m} \times 2^k \rceil$ ，那么

$$\begin{aligned} a \times z \bmod m &\approx \{a \times \frac{p}{2^k}\} \times m \\ &= \frac{ap \bmod 2^k}{2^k} \times m \\ &= \frac{ap \bmod 2^k \times m}{2^k} \end{aligned} \quad (8)$$

最终结果不一定是整数，因为之前  $p$  已经上取整了，这里就选择下取整。我们有一个与 Barrett reduction 类似的定理。

**定理.** 设  $\frac{p}{2^k} = \frac{1}{m} + \epsilon$ ，当  $\epsilon < \frac{1}{am}$  时，答案是准确无误的。

证明.

$$\begin{aligned} \lfloor \{a \times \frac{p}{2^k}\} \times m \rfloor &= \lfloor \{a \times (\frac{z}{m} + \epsilon)\} \times m \rfloor \\ &= \lfloor \{a \times \frac{z}{m} + a\epsilon\} \times m \rfloor \end{aligned} \quad (9)$$

如果  $a \times \frac{z}{m}$  加上  $a\epsilon$  后整数部分都改变了，那么小数部分也就大相径庭了，为了让整数部分不改变，需要保证  $a\epsilon < \frac{1}{m}$ ，然后继续推导：

<sup>11</sup><https://arxiv.org/pdf/1902.01961.pdf>

$$\begin{aligned}
 &= \lfloor \{a \times \frac{z}{m} + a\epsilon\} \times m \rfloor \\
 &= \lfloor \left( \{a \times \frac{z}{m}\} + a\epsilon \right) m \rfloor \\
 &= \lfloor \{a \times \frac{z}{m}\} \times m + am\epsilon \rfloor
 \end{aligned} \tag{10}$$

因为  $\{a \times \frac{z}{m}\} \times m$  是整数，只要后一个部分  $a_i m \epsilon < 1$  答案就是准确的，注意到  $a_i m \epsilon < 1$  其实和之前的必要条件  $a\epsilon < \frac{1}{m}$  等价。  $\square$

在实践中一般取  $k = 64$ ，因为  $k > 64$  的时候计算  $ap \bmod 2^k$  会遇到麻烦。根据定理， $k = 64$  适合  $2^{32} - 1$  以内的  $a, z, m$ 。

这个算法需要对  $z$  预处理出  $p$ ，由于  $z$  通常不是固定不变的，我们需要一个更快的预处理方法。本来  $p = \lfloor \frac{z}{m} \times 2^{64} \rfloor$ ，我们把它改成  $\lfloor \frac{z}{m} \times 2^{64} \rfloor + 1$  也无伤大雅，唯一的区别是  $\epsilon$  的范围从  $[0, \frac{1}{2^{64}})$  变成了  $[0, \frac{1}{2^{64}}]$ 。

$$\begin{aligned}
 p &= \lfloor \frac{z}{m} \times 2^{64} \rfloor + 1 \\
 \text{let } 2^{64} &= Am + B \\
 p &= \lfloor \frac{z}{m} \times (Am + B) \rfloor + 1 \\
 p &= \lfloor \frac{z}{m} \times (Am + B) \rfloor + 1 \\
 p &= \lfloor z \times \frac{B}{m} \rfloor + zA + 1
 \end{aligned} \tag{11}$$

注意到  $\lfloor z \times \frac{B}{m} \rfloor$  可以用 Barrett reduction 算，把  $\frac{B}{m}$  近似为一个分数  $\frac{q}{2^{64}}$ ，其中  $q = \lceil \frac{B}{m} \times 2^{64} \rceil$ ，即：

$$\lfloor z \times \frac{B}{m} \rfloor = \lfloor \frac{zq}{2^{64}} \rfloor \tag{12}$$

实现如下：

```

const int P = 998244353;

u64 trans(u64 x) {
 constexpr u64 A = -(u64)P / P + 1;
 constexpr u64 q = ((u128(-(u64)P % P) << 64) + P - 1) / P;
 return x * A + u64((u128)x * q >> 64) + 1;
}

u32 mul(u32 a, u64 z) {
 return a * z * (u128)P >> 64;
}

```

计算等比数列  $1, x, x^2, \dots$  时就可以把  $x$  转化成  $p$ ，从而更快地完成  $\cdot x \bmod P$  的操作。

```

void foobar(int n, u32 x, u32* pows) {
 u64 p = trans(x);
 pows[0] = 1;

```

```
for (int i = 0; i < n - 1; i++) pows[i + 1] = mul(pows[i], p);
}
```

trans 和 mul 的性能信息如下:

| 表达式       | 延迟 | 吞吐量倒数 | fused-uops/uops | 端口需求                      |
|-----------|----|-------|-----------------|---------------------------|
| mul(a, p) | 6  | 2     | 4/3             | 2*p1 + p5                 |
| trans(z)  | 7  | 3     | 7/6             | 3*p1 + p5                 |
| a * z % P | 12 | 3     | 7/6             | 3*p1 + p5 + p06 + 2*p0156 |

### 3.3.3 更快的小型内积

快速模乘算法还可以推广到小型内积  $(a_1b_1 + a_2b_2 + \dots + a_nb_n) \bmod m$ , 只需要令  $p_i = \lceil \frac{b_i}{m} \times 2^{64} \rceil$ , 那么:

$$\left( \sum a_i b_i \right) \bmod m = \left\lfloor \frac{(\sum a_i p_i) \bmod 2^{64} \times m}{2^{64}} \right\rfloor \quad (13)$$

它适用的条件是  $\sum a_i \leq \frac{2^{64}}{m}$ 。

这个方法可以广泛地优化计数类 DP, 因为 DP 的转移通常都是内积形式, 比如这种形式就很好用。

$$dp_{i,j} = \frac{i}{n} \times dp_{i-1,j-1} + \frac{n-i}{n} \times dp_{i-1,j} \quad (14)$$

### 3.3.4 整除判定

我们可以使用快速模乘算法来解决整除判定问题。例如, 如果我们想要判断一个数  $m$  是否能被另一个数  $a$  整除, 我们可以首先写出余数公式:

$$\begin{aligned} a \times z \bmod m &= \left\lfloor \left\{ a \times \frac{p}{2^{64}} \right\} \times m \right\rfloor \\ &= \left\lfloor \frac{ap \bmod 2^{64} \times m}{2^{64}} \right\rfloor \end{aligned} \quad (15)$$

在这里, 我们令  $z = 1$ , 那么  $p = \lceil \frac{2^{64}}{m} \rceil$ , 我们可以得到一个余数恰好等于 0 的充要条件:

$$ap \bmod 2^{64} < p \quad (16)$$

这是一个非常简洁的条件, 实际效率也非常高。它的适用条件是  $am \leq 2^{64}$ 。

此外, 还有另一个方法, 它能处理  $10^{18}$  级别的  $a, m$ , 但是要求  $m$  是奇数。我们先假设  $m$  能整除  $a$ , 那么存在整数  $k$  使得  $a = mk$ , 然后我们可以通过下面的方式计算出这个  $k$ :



$$\begin{aligned} a &\equiv mk \pmod{2^{64}} \\ k &\equiv am^{-1} \pmod{2^{64}} \end{aligned} \quad (17)$$

我们可以得出结论： $mk < 2^{64}$  是  $m$  整除  $a$  的充要条件。

证明. 必要性：如果  $m$  整除  $a$  那么  $mk = a < 2^{64}$ 。充分性： $mk < 2^{64}$  和  $mk \equiv a \pmod{2^{64}}$  可以推出  $mk = a$ 。□

因此，我们的最终判定条件为：

$$a \times (m^{-1} \bmod 2^{64}) \bmod 2^{64} < \lceil \frac{2^{64}}{m} \rceil \quad (18)$$

计算  $m$  模  $2^{64}$  的逆元最快的方法是使用牛顿迭代法，具体方法见参考资料 [10]。

### 3.3.5 Montgomery 乘法

Montgomery 乘法也是应用广泛的取模算法，要求模数必须是奇数，见参考资料 [11]。适合  $10^{18}$  级别的模数，因此可以加速 Pollard's rho。向量化后效率高也是它的亮点。

### 3.3.6 浮点数辅助取模技巧

使用浮点数也可以方便地实现取模。

假如有一个模数  $p \leq 10^{15}$ ，可以用下面的代码计算  $a * b \% p$ ：

```
long long mul(long long a, long long b, long long p) {
 auto c = a * b - (long long)(1.0 * a * b / p + 0.5) * p;
 if (c < 0) c += p;
 return c;
}
```

原理是这样的：

- 先算出  $\lfloor \frac{ab}{p} \rfloor$  的近似值，可以证明  $1.0 * a * b / p$  的误差小于  $\frac{p}{4 \times 10^{15}}$ 。
- 然后计算  $ab - \lfloor \frac{ab}{p} \rfloor \times p$ ，这里让 long long 溢出了，所以结果会对  $2^{64}$  取模，但是不会影响答案。
- 最后再用 if 解决  $\lfloor \frac{ab}{p} \rfloor$  算出来偏大的情况。

为了提升速度，预处理  $rp = 1.0 / p$ ，然后把  $1.0 * a * b / p$  替换成  $rp * a * b$ ，这样会稍微增大误差，不过  $p \leq 10^{15}$  时仍然能保证正确性。

```
long long mul(long long a, long long b, long long p, double rp) {
 auto c = a * b - (long long)(rp * a * b + 0.5) * p;
 if (c < 0) c += p;
 return c;
}
```

如果  $b$  也是定值的话，还可以进一步优化，预处理  $b\_p = 1.0 * b / p$ ，然后就把浮点数乘法减少到了 1 次。

有几个注意事项：

- 不要把 `long long` 换成 `u64`，因为无符号整数和浮点数的相互转化较慢。
- $10^{18}$  级别的模数需要切换成 `long double`。

即使是  $10^9$  级别的输入模数，这个算法也是比直接取模快不少的。性能信息对比如下：

| 表达式               | 吞吐量倒数 | fused-uops/uops |
|-------------------|-------|-----------------|
| $(u64)a * b \% p$ | 21    | 39/34           |
| 预处理 $1.0 / p$     | 5.05  | 19/16           |
| 预处理 $1.0 * b / p$ | 4     | 15/13           |

### 3.4 整除分块

整除分块的一个主要瓶颈在于除法运算，因此我提出了一个改进的方法。

通常，整除分块的目的是计算以下这种式子：

$$\sum_{ab \leq n} f(a)g(b) \quad (19)$$

一个更好的计算方法是分别计算  $a \leq \lfloor \sqrt{n} \rfloor$  和  $b \leq \lfloor \sqrt{n} \rfloor$  的贡献，然后再减去  $\max(a, b) \leq \lfloor \sqrt{n} \rfloor$  的贡献。用公式表达就是：

$$\sum_{ab \leq n} f(a)g(b) = \sum_{i=1}^{\lfloor \sqrt{n} \rfloor} \left( f(i)S_g(\lfloor \frac{n}{i} \rfloor) + g(i)S_f(\lfloor \frac{n}{i} \rfloor) \right) - S_f(\lfloor \sqrt{n} \rfloor)S_g(\lfloor \sqrt{n} \rfloor) \quad (20)$$

在这里， $S_f$  和  $S_g$  分别表示  $f$  和  $g$  的前缀和。这个公式只需要  $\lfloor \sqrt{n} \rfloor$  次除法，次数为传统整除分块的  $\frac{1}{4}$ 。当需要多次运行整除分块时，我们还可以使用 Barrett reduction 来进一步优化除法。

### 3.5 bitset

手写 bitset 可以轻松比 `std::bitset` 更快。即使某些字符串问题的数据范围较大<sup>12</sup>，手写 bitset 仍然有机会通过。

`std::bitset` 可以被轻松超越是因为它有以下不足：

- 不灵活，只能操作整体，不支持操作区间。
- 每完成一次运算都会把结果写入内存，比如 `a | (b & c)` 和 `a |= a << shift` 都会在内存中转一次。
- 部分函数本身速度较慢，比如 `count`, `any`, `_Find_next`。

现在我们来一一讨论每个问题该怎么用手写 bitset 优化。

首先我们应该用 `u64` 数组来存 bitset，AND, OR, XOR 运算都是简单的，需要提速的时候就循环展开。

不平凡的操作是位移。

### 3.6 位移

比起整体左移和右移，区间复制是更有用操作，我们实现一个类似于 `memcpy` 的 `bitcpy(d, di, s, si, len)` 函数，表示把  $d_{di..di+len-1}$  赋值为  $s_{si..si+len-1}$ ，实现流程如下：

1. 开始时每次复制 1 bit，直到  $di \bmod 64 = 0$  或者已经复制了  $len$  个 bit。
2. 从现在开始每次复制 64 bit，即  $d_{di..di+63} \leftarrow s_{si..si+63}$ ，直到剩余 bit 数不足 64。
3. 最后每次复制 1 bit，直到已经复制了  $len$  个 bit。

最复杂的是第二个阶段， $d_{di..di+63}$  刚好是一个 `u64`，而  $s_{si..si+63}$  可能跨越两个 `u64`，需要用位移把两个部分拼起来。注意特判  $si \bmod 64 = 0$  的情况，不然会发生位移量为 64 的位移，这是未定义行为。

代码如下，我们需要使用 `__restrict__` 关键字来提示编译器两个范围不相交，不然优化效果不好：

```
void bitcpy(u64* __restrict__ d, u64 di, u64* __restrict__ s, u64 si, u64 len) {
 while (len && di % 64) assign(d, di++, s, si++), len--; // assign 函数复制单个 bit
 auto dst = d + di / 64, src = s + si / 64;
 auto end = dst + len / 64;
 auto shift = si % 64;
 if (!shift) memcpy(dst, src, (end - dst) * 8);
```

<sup>12</sup>例如「NOI2023」字符串 <https://loj.ac/p/3983>

```
else while (dst != end) *dst++ = src[0] >> shift | src[1] << 64 - shift, src++;
di += len & -64, si += len & -64, len %= 64;
while (len--) assign(d, di++, s, si++);
}
```

### 3.7 运算融合

运算融合是指把表达式中的多次运算融合进一个循环里，这样中间结果就不需要写入内存再读取了。像  $a = (a \& b) | c$  这种没有位移的表达式很简单，有一个位移操作也还好，把表达式嵌入 `bitcpy` 即可。有多个位移时可以放弃把两个位移融合在一起，毕竟 `bitcpy` 的瓶颈不是内存读写。

### 3.8 popcount

如果能用 `pragma` 的话 `popcount` 根本不是问题。在代码中加入 `#pragma target("popcnt")` 后 `__builtin_popcountll` 就会编译成 x86 指令 `popcnt`，并行情况下每周可以处理 64 bits。

不能用 `pragma` 的话，`__builtin_popcountll` 的速度就比较慢了，这是因为它会调用二进制库中的函数 `__popcountdi2`，这个函数本身实现很好，但是调用开销会让它的性能减半，所以最好不要使用它，以下是一些替代方案：

- 对 0~65535 预处理 `popcount`，然后每次读取一个 `uint16_t` 再查表，速度大概 10.23 bits/cycle。

```
auto ptr = (uint16_t*)bits;
u64 res = 0;
for (int i = 0; i < N / 16; i++) res += lookUp[ptr[i]];
```

- 手写一个和 `__popcountdi2` 实现一样的函数，这个代码的原理见参考资料 [12]。速度大概 14.17 bits/cycle。

```
u64 popcnt(u64 n) {
 n -= n >> 1 & 0x5555555555555555;
 n = (n & 0x3333333333333333) + (n >> 2 & 0x3333333333333333);
 n = (n + (n >> 4)) & 0xF0F0F0F0F0F0F0F;
 return n * 0x101010101010101 >> 56;
}

u64 res = 0;
#pragma GCC unroll 8
for (int i = 0; i < N / 64; i++) res += popcnt(bits[i]);
```

- 继续优化上一个方法。我们的目的是求出很多 u64 的 popcount 之和，所以在完成前两步迭代后，进行若干次合并，把 u64 的数量减下去再完成后两步迭代。速度大概 19.00 bits/cycle。

```
u64 popcnt(u64 n) {
 n = (n & 0xF0F0F0F0F0F0F0F) + (n >> 4 & 0xF0F0F0F0F0F0F0F);
 return n * 0x101010101010101 >> 56;
}

u64 res = 0;
for (int i = 0; i < N / 64; i += 7) {
 u64 low = 0, high = 0;
 #pragma GCC unroll 7
 for (int j = 0; j < 7; j++) {
 u64 n = bits[i + j];
 n -= n >> 1 & 0x5555555555555555;
 low += n & 0x3333333333333333;
 high += n >> 2 & 0x3333333333333333;
 }
 res += popcnt(low) + popcnt(high);
}
```

这个合并的技巧还可以发挥到极致，见参考资料 [16]

### 3.9 判断 bitset 是否全 0

较优的实现是使用循环展开求出所有 u64 的 OR。

### 3.10 遍历所有的 1

较优的实现是枚举每个 u64，然后用如下方式遍历当前 u64 中的所有 1。

```
while (n) {
 int i = __builtin_ctzll(n);
 ...
 n &= n - 1;
}
```

### 3.11 快速傅里叶变换

快速傅里叶变换的常规写法每次蝴蝶变换的乘数都会切换，这个切换是可以消除的。有一种优秀写法只会切换  $O(n)$  次，它既不是 DIT 也不是 DIF，但和 DIT 和 DIF 都有一些相似之处，接下来我以 DIF 的方式来解释它：

记  $DFT(a, n, z)_k$  表示  $\sum_{i=0}^{n-1} (z^i a_i) \omega_n^{ik}$ ，相当于先给  $a$  点乘上等比数列  $\{1, z, z^2, \dots\}$  再做 DFT。考虑将多项式  $a$  的系数分为前半和后半：

$$\begin{aligned}
 DFT(a, n, z)_k &= \sum_{i=0}^{n-1} (z^i a_i) \omega_n^{ik} \\
 &= \sum_{i=0}^{n/2-1} \left( (z^i a_i^{(left)}) + (z^i a_i^{(right)}) z^{n/2} \omega_n^{n/2 \cdot k} \right) \omega_n^{ik} \\
 &= \sum_{i=0}^{n/2-1} \left( (z^i a_i^{(left)}) + (z^i a_i^{(right)}) z^{n/2} (-1)^k \right) \omega_n^{ik}
 \end{aligned} \tag{21}$$

通过讨论  $k$  的奇偶性，可以把  $(-1)^k$  去掉：

$$\begin{aligned}
 DFT(a, n, z)_{2k} &= \sum_{i=0}^{n/2-1} \left( (z^i a_i^{(left)}) + (z^i a_i^{(right)}) z^{n/2} \right) \omega_{n/2}^{ik} \\
 &= DFT(a^{(left)} + z^{n/2} a^{(right)}, n/2, z)_k \\
 DFT(a, n, z)_{2k+1} &= \sum_{i=0}^{n/2-1} \left( (z^i a_i^{(left)}) - (z^i a_i^{(right)}) z^{n/2} \right) \omega_{n/2}^{ik} \omega_n^{ik} \\
 &= DFT(a^{(left)} - z^{n/2} a^{(right)}, n/2, z \cdot \omega_n)_k
 \end{aligned} \tag{22}$$

由于偶数的答案和奇数的答案分别构成了子问题，实现上可以把偶数的答案放到左边，奇数的答案放到右边，这样会导致最终答案的顺序乱了，我们后面再讨论这个问题。

另一个问题是每次蝴蝶变换的乘数的  $z^{n/2}$  怎么求出来，有没有什么简洁的形式？结论是递归到区间  $[l, r]$  时，蝴蝶变换的乘数  $z^{(r-l)/2} = f(N/l)$ ，其中

$$f(2^{k_1} + 2^{k_2} + \cdots + 2^{k_n}) = \omega_{2^{k_1+1}} \omega_{2^{k_2+1}} \cdots \omega_{2^{k_n+1}} \quad (k_1 < k_2 < \cdots < k_n) \tag{23}$$

$f$  的预处理方法是倍增： $f_{i+2^k} = f_i \cdot \omega_{2^{k+1}}$

最后代码如下：

```

for (int i = n >> 1; i; i >>= 1)
 for (int j = 0; j < n / 2 / i; j++)
 for (int k = j * i * 2; k < j * i * 2 + i; k++) {
 u32 x = a[k], y = (u64)a[k + i] * w[j] % P;
 a[k] = add(x, y), a[k + i] = sub(x, y);
 }

```

这样 DFT 后点值顺序是乱的，但是我们可以不管他：直接把点值乘起来，要 IDFT 回去的话就把这个函数的每一步依次撤销。

其他有效的优化：

- 使用 3.3.2 中的快速模乘算法对  $w$  数组预处理。
- 循环展开， $i \leq 8$  的情况还要单独拎出来。

加上了所有优化的代码见参考资料 [16]。



## 4 数据结构

数据结构的性能瓶颈通常在于内存请求和分支预测未命中，使用指针的数据结构更加低效，因为内存请求无法并发。以下是一些参考优化方向：

- 使用结构体捆绑节点信息，不重要的信息放在外面。另外，成员类型要按大小从大到小排列，这样才能最小化结构体的大小。
- 尝试离散化、离线算法，选择简单的数据结构。
- 采用更好的实现方式，这也是本章的主要内容。

### 4.1 非递归线段树的优化

非递归线段树是一种效率高且易于编写的数据结构，其基本原理可以参考资料 [18]。本节将介绍一些可以优化的细节。首先，我们约定数组长度为  $M = 2^{20}$ ，数据类型为 `int`。

#### 4.1.1 修改

在进行修改操作时，如果可以避免使用 `pushUp`，那就尽量避免，因为 `pushUp` 会产生内存依赖链，并增加内存请求。

- 单点加区间和的最优写法。这种写法的所有内存请求都是独立的。

```
void add(int i, int v) {
 for (i += M; i; i >>= 1) c[i] += v;
}
```

- 单点增大区间 `Max` 的最优写法。实验证明，无论在何种数据下，这种写法都比 `c[i] = max(c[i], v)` 更快。

```
// let c[0] = inf
void checkMax(int i, int v) {
 for (i += M; v > c[i]; i >>= 1) c[i] = v;
}
```

- 单点修改维护 `Max` 的最优写法。虽然需要先读后写，但是读和写都是可以并行的。

```
void modify(int i, int v) {
 for (i += M; i; i >>= 1) c[i] = v, v = max(v, c[i ^ 1]);
}
```

这种方法本质上也是 `pushUp`，但是它利用了信息的交换律，从而减少了内存请求。这个方法还可以用在区间修改中，例如下面的区间加和区间 `Max`

```
int c[M << 1][2]; // c[l][0] is max, c[l][1] is tag;
void add(int l, int r, int v) {
 l += M - 1, r += M + 1;
 int L = c[l][0], R = c[r][0];
 while (l ^ r ^ 1) {
 int vl = ~l & 1 ? v : 0, vr = r & 1 ? v : 0; // 去除分支
 L = max(L, c[l ^ 1][0] += vl), c[l ^ 1][1] += vl;
 R = max(R, c[r ^ 1][0] += vr), c[r ^ 1][1] += vr;
 l >>= 1, c[l][0] = L += c[l][1];
 r >>= 1, c[r][0] = R += c[r][1];
 }
 L = max(L, R);
 while (l >>= 1) c[l][0] = L += c[l][1], L = max(L, c[l ^ 1][0]);
}
```

Dijkstra 算法的堆选择非递归线段树会更快。每个节点维护 `min` 和它的下标，尽量将值和下标压缩到一个 `u64` 中，这样就可以只维护 `u64` 的最小值。

#### 4.1.2 区间查询

如果没有维护任何标记，区间查询可以直接定位出所有需要的节点，而无需逐层跳跃。下面是单点修改和区间 `Max` 的示例：

```
int lg(int n) { return 31 ^ __builtin_clz(n); }
int query(int l, int r) {
 l += M - 1, r += M + 1;
 int o = l >> lg(l ^ r); // lca 的左儿子
 int L = -1e9, R = -1e9; // 解除两路的依赖关系
 while ((l >>= __builtin_ctz(~l)) != o) L = max(L, c[+l]);
 o++; // lca 的右儿子
 while ((r >>= __builtin_ctz(r)) != o) R = max(R, c[--r]);
 return max(L, R);
}
```

#### 4.1.3 懒标记下放

虽然没有标记永久化快，懒标记下放也是可行的。下面的代码可以避免某些节点被 `push-Down` 两次。

```
int lg(int n) { return 31 ^ __builtin_clz(n); }
int query(int l, int r) {
 l += M - 1, r += M + 1;
 for (int i = 20; i >= 1; i--) pushDown(l >> i);
 for (int i = lg(l ^ r); i >= 1; i--) pushDown(r >> i);
 ...
}
```

## 4.2 线段树上二分的优化

线段树上二分是一个常见的操作，有时它用来将  $O(\log^2 n)$  的复杂度降低到  $O(\log n)$ ，有时在 `pushUp` 的时候也需要它。

线段树的结构非常适合进行二分操作。通过本节的优化，线段树上二分将比树状数组的二分快 4 倍。

### 4.2.1 集合第 $k$ 小

**例题.** 维护一个可重集，支持加入/删除元素，查询第  $k$  小，值域  $10^6$ 。

树状数组可以简洁地查询第  $k$  小，原理见参考资料 [15]。

```
// M = 1 << 20
int kth(int k) {
 int p = 0;
 for (int i = M >> 1; i; i >>= 1)
 if (k > c[p + i]) p += i, k -= c[p];
 return p + 1;
}
```

为了压缩空间并提升内存访问效率，非递归线段树也可以采用树状数组的策略，只维护左儿子的信息。每个节点存储左半段的和，叶子节点都被删除，第  $k$  小的查询非常好写：

```
int c[M];
int kth(int k) {
 int i = 1;
 while (i < M) {
 if (k > c[i]) k -= c[i], i = i * 2 + 1;
 else i = i * 2;
 }
 return i - M;
}
```

由于存储方式的问题，`add` 和 `sum` 都需要重写，见参考资料 [16]。重写后的 `add` 的速度已经超过了树状数组的 `add`，但是 `sum` 还没有。

下面是速度测试环节，将  $10^6$  个均匀随机数插入可重集中，然后每次查询的  $k$  在  $[1, 10^6]$  均匀随机。

| 类型     | 单次查询周期数 |
|--------|---------|
| 树状数组   | 639     |
| 非递归线段树 | 356     |

为什么非递归线段树比树状数组快了整整一倍，这得谈到内存：

- 树形数据结构都有一个特点：离根越近的元素访问频率越高，最终形成的局面是前若干层在缓存中，后若干层不在缓存中。位于缓存的层数越多整体访问速度就越快。
- 线段树的前若干层空间是连续的，每 16 个元素才占用一个缓存行，而树状数组前若干层是分散的，每个元素都会占用一个缓存行。
- 树状数组前若干层的地址组成了公差为  $2^k$  的等差数列。根据缓存分组关联的特点 (1.4.3)，只有常数个元素进入了 L1 缓存。幸运的是，检索 L2, L3 缓存之前都会进行虚拟地址到物理地址的转化，这个等差数列会被破坏掉，所以 L2, L3 缓存至少还能利用起来（但是利用也不完整）。

综上，线段树有前若干层访问很快的优点，但是树状数组没有。

我们接下来继续优化线段树上二分操作<sup>13</sup>。目前的首要瓶颈是分支预测，这是因为随机数据导致预测的命中率只有 50%。我们首先将分支替换为 `cmov`。

```
int kth(int k) {
 int i = 1;
 while (i < M) {
 bool flag = k > c[i];
 if (flag) k -= c[i];
 i = i * 2 + flag;
 }
 return i - M;
}
```

再次测试后，结果令人大跌眼镜，每次查询需要 336 cycles，并没有显著的进步。通过 Profiler 测得的分治预测未命中率  $< 10^{-6}$ ，这说明问题出在了其他地方。

这里就要介绍一下分支预测的正面作用了。分支预测器预测了接下来的走向后，就会沿着这一路推测执行，还会进行预取 (Prefetch)，意思是提前将数据移入 L1 缓存。由于线段树的结构原因，当前节点的 2 个儿子、4 个孙子等都位于同一缓存行，无论预测是否命中，都获取了正确的缓存行。没有分支预测的话，处理器就完全不知道接下来要访问的数据，所以不会进行预取，然后每次都需要完整地等待读取结果。

预取也可以手动执行，使用 `__builtin_prefetch(address)` 函数可以预取 `address` 所在的缓存行，如果 `address` 非法只会忽略该请求，不会引发运行时错误。由于 `i` 的 16 个 4 级后代刚好适合一个缓存行，我们就预取所有四级后代，然后实际访问数据的时候延迟就很小了。

```
alignas(64) int c[M]; // 确保 i 的 4 级后代位于同一个缓存行
int kth(int k) {
 int i = 1;
 while (i < M) {
 __builtin_prefetch(&c[i * 16]);
 }
}
```

<sup>13</sup>灵感来源 <https://en.algorithmica.org/hpc/data-structures/binary-search/>

```
bool flag = k > c[i];
if (flag) k -= c[i];
i = i * 2 + flag;
}
return i - M;
}
```

现在速度得到了较大提升，每次循环只需要 203 cycles 了。

因为这是一个串行算法，我们在算术上做优化也能提升速度，利用一点奇技淫巧，下面的代码又加速到了 167 cycles。

```
int kth(uint32_t k) {
 k--;
 size_t i = 1;
 while (i < M) {
 __builtin_prefetch(&c[i * 16]);
 int t = k - c[i];
 if (t >= 0) k = t;
 i = i * 2 + (t >= 0);
 }
 return i - M;
}
```

你可能已经认为这段代码优化不动了，那么请看下面的实验：强制一遍 kth 得出结果后才能开始下一次 kth。

```
for (int i = 0; i < N; i++) last = kth(k[i] + (last == 114514));
```

这样会让 kth 变慢到 250 cycles，这是什么原因呢？如果我不强制 kth 顺序执行，即使当前 kth 还没有执行完，分支预测器就已经知道当前循环还有多少次结束，然后将提前下一次 kth 的指令放入流水线，但是调度器的容量也只有 97，这意味着只有当前 kth 执行到最后几十条指令时才能开始并行下一次 kth 调用。我们从中获得启发，将多个操作融合在一起就可以大力度地并行，例如设计 kth\_and\_kth, kth\_and\_add 这种的融合函数。

## 4.2.2 局部二分

对一个区间进行二分也是非常常见的操作。下面给出一个示例代码，该代码实现了“从 i 开始向后查找第一个大于 v 的位置”：

```
// 确保序列末尾是 inf
int find(size_t i, int v) {
 i += M - 1;
 while (i >= __builtin_ctz(~i), c[++i] <= v);
 while (i < M) {
 __builtin_prefetch(&c[i * 16]);
 i = i * 2 + (c[i * 2] <= v);
 }
 return i - M;
}
```

### 4.3 等位信息捆绑

“等位信息”是指每次修改位置和查询位置都相同的信息。最常见的例子是树状数组的区间加区间和，其原理可以参考资料 [17]。实现方法如下：

- 首先，我们需要维护两个数组  $A$  和  $B$ 。
- 当我们要对后缀  $[i, n]$  加上一个值  $v$  时，我们可以令  $A_i += v$  和  $B_i += v \cdot i$ 。
- 要求前缀  $[1, i]$  的和，我们可以计算  $(i + 1) \cdot S_A(i) - S_B(i)$ 。

在这里， $A$  和  $B$  就是等位信息。一个更优的实现方法是将  $A$  和  $B$  用一个结构体捆绑在一起，并且修改和查询都只使用一个循环。

#### 4.3.1 ST 表

ST 表的预处理最优写法见 2.5.1。本节主要讨论以下两个问题：

- 在数组中，哪一维放在前面会使查询更快。
- 如何最快地计算  $\lfloor \log_2(r - l + 1) \rfloor$ 。

对于第一个问题，答案是将较小的一维放在前面。原因是统计学效应：人为生成的或有意义的数据通常满足  $\log_2(r - l + 1)$  呈正态分布，频率较高的数据会留在缓存中。如果我们特意让  $\log_2(r - l + 1)$  在  $[0, \log_2 n]$  中均匀生成，那么哪一维放在前面的速度就会一样。

对于计算  $\lfloor \log_2(n) \rfloor$ ，最常见的方法是预处理一个 `log_2` 数组，或者使用 `std::lg(n)`。后者的效率很不错，但它并不是最快的。最快的方法是使用 `31 ^ __builtin_clz(n)`。这个原因需要在汇编层面解释：`bsr` 指令就是我们想要的功能，`std::lg(n)` 的实现是 `31 - __builtin_clz(n)`，`__builtin_clz(n)` 会被编译为 `bsr` 后再 `xor 31`，减法和异或不能抵消，就产生了冗余的操作。如果我们使用 `31 ^ __builtin_clz(n)`，那么两次异或就会被编译器优化掉。

最后，ST 表查询最小值位置可以用 `long long` 压缩一下，高 32 位存储值，低 32 位存储下标。如果值很大，我们可以分配更多的位来存储。

#### 4.3.2 无旋 Treap

见参考资料 [19]。



## 5 杂项

### 5.0.1 vector 替代品

vector 的主要使用场景是使用 vector 数组存储总量  $O(n)$  的元素，比如存图。这种使用方式的时间常数和空间常数并不优秀。

我们从存图出发，了解如何实现一个优秀的存图方式。最常用的两种存图方式为 vector 数组和邻接表，它们都不是完美的。

| 速度   | vector 数组 | 邻接表 |
|------|-----------|-----|
| 加边   | 慢         | 快   |
| 遍历出边 | 快         | 慢   |

完美的存图方式是存在的，它兼具加边速度和遍历出边的速度。首先，统计每个点的度数  $d_u$ ，然后对  $d$  数组做前缀和，最后把  $u$  的所有出边放进区间  $[d_{u-1}, d_u)$  中，这一步要按照计数排序的写法。示例如下：

```
// 无向图
int a[E], b[E], d[V + 1], G[E << 1];

void build(int n, int m) {
 for (int i = 0; i < m; i++)
 cin >> a[i] >> b[i], d[a[i]]++, d[b[i]]++;
 for (int i = 0; i < n; i++) d[i + 1] += d[i];
 for (int i = 0; i < m; i++) {
 G[--d[a[i]]] = b[i];
 G[--d[b[i]]] = a[i];
 }
}

void edges(int u) {
 for (int i = d[u]; i < d[u + 1]; i++)
 cout << G[i] << ' ';
}
```

这种写法可以轻易地推广到有向图和任意 vector 数组。

### 5.0.2 Binary GCD

Binary GCD 是一种小常数的 gcd 算法，其原理如下：

$$\gcd(a, b) = \begin{cases} 2 \gcd(a/2, b/2) & (a, b \text{ 均为偶数}) \\ \gcd(a/2, b) & (a \text{ 为偶数}, b \text{ 为奇数}) \\ \gcd(|a - b|, \min(a, b)) & (a, b \text{ 均为奇数}) \end{cases} \quad (24)$$

这个算法的关键在于实现：

- 首先，将  $a, b$  中的 2 除尽。
- 当  $a, b$  都是奇数时，会递归到  $|a - b|, \min(a, b)$ ，用 `__builtin_ctz` 将  $|a - b|$  中的 2 除尽后，又回到了  $a, b$  都是奇数的情况。这个过程会一直循环，直到  $a = b$ 。
- 在计算  $a - b$  的绝对值之前，先计算 `ctz`，因为  $a - b$  和  $b - a$  的 `ctz` 是相等的，所以在计算绝对值之前计算 `ctz` 可以与计算绝对值并行。

下面是一个示例<sup>14</sup>，它的速度足以通过「基于值域预处理的快速 GCD」<sup>15</sup>。

```
int gcd(int a, int b) {
 if (!a || !b) return a + b;
 int az = __builtin_ctz(a);
 int bz = __builtin_ctz(b);
 int z = min(az, bz);
 a >>= az, b >>= bz;
 while (a != b) {
 int diff = b - a;
 az = __builtin_ctz(diff);
 b = min(a, b), a = abs(diff) >> az;
 }
 return a << z;
}
```

### 5.0.3 常见问题解答

**Q.** `inline` 加上会变快吗？

内联函数的主要目的是消除函数调用的开销。编译器在这方面做得很好，它会自动内联应该内联的函数，所以通常我们不需要手动添加 `inline`。有时候，出于特殊目的，我们会使用内联，比如在 2.1.6 中，内联是必须的，因此我们还使用了 `__attribute__((always_inline))` 来强制内联。

**Q.** 手写读入中  $x = x * 10 + c - 48$  改成  $x = (x \ll 1) + (x \ll 3) + (c \wedge 48)$  更快吗？

$(x \ll 1) + (x \ll 3)$  是完全没有必要的，因为编译器的实现既没有乘法也没有位运算。

$c - 48$  和  $c \wedge 48$  哪个快是微架构相关的，这并不是因为减法和异或的速度差异，而是因为减法被编译成了 `lea eax, [ecx + eax * 2 - 48]`，这在 Coffee Lake 上特别

<sup>14</sup>来源于 <https://en.algorithmica.org/hpc/algorithms/gcd/>，做了少量修改

<sup>15</sup><https://www.luogu.com.cn/problem/P5435>

慢，而在较新的微架构上又特别快。另外，`mov eax, DWORD PTE [ecx + eax * 2 - 48]` 并不慢，这是因为加载端口里有专门计算 `lea` 的执行单元，它比 `ALU` 计算 `lea` 还要快一些。

**Q.** 数组大小开  $2^k$  会变慢吗？

可能会，但数组大小本身和速度没有直接关系，数组大小开  $2^k$  可能是损害性能的间接原因，直接原因见 1.4.3。

## 6 性能分析

在不借助工具的情况下，唯一能测量的性能信息就是时间。然而，性能分析工具可以提供更多的有用信息，来帮助寻找程序的优化方向。以下是一些我尝试过的工具：

### 6.1 Profiler

Profiler 可以统计各种事件，比如时钟周期、缓存未命中率、分支预测未命中率、各个函数的开销占比等等。

- `perf` 是一个命令行工具。获取方法：`apt-get install linux-tools-common linux-tools-generic linux-tools-`uname -r``。文档链接：<https://perf.wiki.kernel.org/index.php/Tutorial>
- `likwid perfctr` 也是一个命令行工具，它的功能比 `perf` 要丰富一些，比如测量代码局部的性能事件。获取链接：<https://github.com/RRZE-HPC/likwid>。文档链接：<https://github.com/RRZE-HPC/likwid/wiki/likwid-perfctr>。

### 6.2 缓存系统/分支预测模拟工具

`Cachegrind` 是一个缓存系统/分支预测模拟器。获取链接：<https://valgrind.org/downloads/>。文档链接：<https://valgrind.org/docs/manual/cg-manual.html>

### 6.3 机器代码分析/流水线模拟

`uiCA` 是一个流水线模拟器，可以计算吞吐量，还可以提供微架构中的细节。获取链接：<https://www.uops.info/uiCA.html>。文档链接：[\[4\]](#)，这篇文献也是一份关于微架构的可靠资料。

此外，`llvm-mca` 和 `OSACA` 也是流行的机器代码分析工具。



## 7 总结

在我们讨论了众多的优化原理和实例之后，相信大家已经理解到，程序优化是一个既复杂又有趣的领域。虽然程序优化本身没有固定的规律，但在大量的实践中，我们可以总结出一些有效的方法和技巧。

最后，程序优化是永无止境的，祝愿大家能够享受优化过程带来的乐趣和满足感。

## 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢父母对我的培养、关心与支持。

感谢祝敏磊老师和李翔老师对我的培养与教导。

感谢国家集训队杨耀良教练和彭思进教练的指导。

感谢李翔老师、何钊佑、崔隽章同学为本文审稿。

## 参考文献

- [1] Sergey Slotin, Algorithms for Modern Hardware, <https://en.algorithmica.org/hpc/>
- [2] Agner Fog, The microarchitecture of Intel, AMD, and VIA CPUs, <https://www.agner.org/optimize/microarchitecture.pdf>
- [3] Agner Fog, Optimizing software in C++, [https://www.agner.org/optimize/optimizing\\_cpp.pdf](https://www.agner.org/optimize/optimizing_cpp.pdf)
- [4] Andreas Abel and Jan Reineke, uiCA: Accurate Throughput Prediction of Basic Blocks on Recent Intel Microarchitectures, <https://arxiv.org/pdf/2107.14210.pdf>
- [5] 骆可强《论程序底层优化的一些方法与技巧》，IOI 2009 集训队论文集。
- [6] x86 and amd64 instruction reference, <https://www.felixcloutier.com/x86/>
- [7] 快速范围判断, <https://zhuanlan.zhihu.com/p/147039093>
- [8] 8 和 16 位运算的缺点, <https://stackoverflow.com/questions/41573502>
- [9] Wikipedia, Tree-Pseudo-LRU, <https://en.wikipedia.org/wiki/Pseudo-LRU#Tree-PLRU>

- [10] Fast-inverse-trick, [https://cp-algorithms.com/algebra/montgomery\\_multiplication.html#fast-inverse-trick](https://cp-algorithms.com/algebra/montgomery_multiplication.html#fast-inverse-trick)
- [11] Montgomery Multiplication, [https://cp-algorithms.com/algebra/montgomery\\_multiplication.html](https://cp-algorithms.com/algebra/montgomery_multiplication.html)
- [12] Algorithms behind Popcount, <https://nimrod.blog/posts/algorithms-behind-popcount/>
- [13] Instruction Table, <https://uops.info/table.html>, its introduction <https://dl.acm.org/doi/pdf/10.1145/3524059.3532396>
- [14] Instruction tables, [https://www.agner.org/optimize/instruction\\_tables.pdf](https://www.agner.org/optimize/instruction_tables.pdf)
- [15] 树状数组查询集合第  $k$  小, <https://oi-wiki.org/ds/fenwick/#%E5%8D%95%E7%82%B9%E4%BF%AE%E6%94%B9%E6%9F%A5%E8%AF%A2%E5%85%A8%E5%B1%80%E7%AC%AC-k-%E5%B0%8F>
- [16] 本文补充材料 <https://github.com/platelett/thesis>
- [17] 树状数组区间加区间和, <https://oi-wiki.org/ds/fenwick/#%E5%8C%BA%E9%97%B4%E5%8A%A0%E5%8C%BA%E9%97%B4%E5%92%8C>
- [18] 非递归线段树, <https://zhuanlan.zhihu.com/p/361935620>
- [19] skip2004, Treap, <https://www.cnblogs.com/skip2004/p/16574235.html>



# 浅谈一类容斥和反演方法的应用

中国人民大学附属中学 董亦骐

## 摘要

容斥原理是组合数学中一种重要的计数方法,用于解决包含排斥和重复的计数问题。它在OI的许多分支中都有广泛的应用。本文主要探讨这一模型在数论问题中的推广形式。

## 1 容斥

### 1.1 容斥基本概念

假定读者已经掌握基本的组合计数知识。

下文中,约定  $\binom{n}{m}$  表示组合数。

假设我们有一个大集合  $X, A, B, C$  是  $X$  的三个子集,要求  $X$  中不在  $A, B, C$  里任何一个的元素数量,一个熟知结论是这样的元素数量恰为  $|X| - |A| - |B| - |C| + |A \cap B| + |A \cap C| + |B \cap C| - |A \cap B \cap C|$ ,这也是容斥原理的经典形式。

这个式子很容易推广到  $n$  个集合的情形,如果有  $n$  个子集  $A_1, A_2, \dots, A_n, X$  中不在这些子集里的元素个数即为  $|X| - \sum_i |A_i| + \sum_{i < j} |A_i \cap A_j| - \dots + (-1)^n |A_1 \cap A_2 \cap A_3 \dots \cap A_n|$ 。

我们给出一个更一般的定义:我们现在有一个全集  $X$ , 和  $n$  个性质  $e_1, e_2, \dots, e_n$ , 接下来定义  $N_{\supseteq T} = \{x | x \text{ 满足 } T \text{ 中性质}\}$ ,  $N_{=T} = \{x | x \text{ 满足且只满足 } T \text{ 中性质}\}$ 。

那么我们可以给出一个容斥原理的更一般形式:如果有一个全集  $X$ , 和一个性质组成的集合  $E = \{e_1, e_2, \dots, e_n\}$ ,  $N_{=0} = \sum_{T \subseteq E} (-1)^{|T|} N_{\supseteq T} = \sum_{k=0}^n (-1)^k \sum_{|T|=k} N_{\supseteq T}$

如果  $N_{\supseteq T}$  只和  $|T|$  有关,那么我们称性质集合  $E$  是同质的,并可以改写上面的符号,对  $|T| = k$ , 定义  $N_{=k} = N_{=T}, N_{\geq k} = N_{\supseteq T}$ , 那么由上面地式子,我们可以得到  $N_{=0} = \sum_{k=0}^n \binom{n}{k} (-1)^k N_{\geq k}$  这里我们给出一个简单的例题:

**例题 1.1** (带上界的计数问题<sup>1</sup>). 给定  $n, m, s$ , 求有多少个长度为  $n$  的序列  $x$  满足  $\sum_{i=1}^n nx_i = m$  且  $0 \leq x_i < s$ , 答案对大质数取模。  $1 \leq n, m, s \leq 10^6$ 。

<sup>1</sup>来源: 经典问题



我们定义  $X$  是所有满足  $\sum_{i=1}^n nx_i = m$  的序列组成的集合, 根据熟知结论,  $|X| = \binom{n+m-1}{n-1}$ , 第  $i$  个性质是  $x_i \geq s$ , 那么可以发现由于元素之间是对称的, 性质集合是同质的, 具体求解  $N_{\geq k}$  时就是限制了  $k$  个元素必须大于等于  $s$ , 把它们减去  $s$  后可以发现我们要计数的就是满足  $\sum_{i=1}^n x'_i = n - ks$  的序列  $x'$  的个数, 因此我们可以得到  $N_{=0} = \sum_{j=0}^n n \binom{n}{j} \binom{n+k-j-s-1}{n-1}$ , 进而  $O(n)$  地解决了这个问题。接下来, 我们给出一个用容斥原理证明定理的例子。

对于给定正整数  $n$ , 我们定义它的一个分拆是满足如下条件的正整数序列  $a$ :

1.  $a_1 + a_2 + \dots + a_k = n$
2.  $a_1 \geq a_2 \geq a_3 \dots \geq a_k$

**定理 1.1.** 对于任意正整数  $n$ , 满足  $a_i$  均为奇数的分拆和满足  $a_i$  互不相同的分拆个数相等。

证明. 定义  $X$  为所有  $n$  的分拆组成的集合,  $f(n)$  为  $n$  的分拆总个数。先考虑  $a_i$  均为奇数的分拆, 定义第  $i$  个性质是  $2i$  在分拆中出现, 奇数分拆就是不满足这些性质的分拆, 那么考虑满足某个性质的分拆数实际上就是  $f(n - 2w(T))$ ,  $w(T)$  表示  $T$  中每个元素的和, 得到奇数分拆总个数  $f_o(n) = \sum_{T \subseteq E} (-1)^{|T|} f(n - 2w(T))$

再考虑所有  $a_i$  互不相同的分拆, 定义第  $i$  个性质是  $i$  出现了至少两次, 元素互不相同的分拆就是不满足这些性质的分拆, 可以发现满足某个性质的分拆数也是  $f(n - 2w(T))$ , 得到元素互不相同的分拆的总个数  $f_d(n) = \sum_{T \subseteq E} (-1)^{|T|} f(n - 2w(T))$ , 两者显然是相等的。□

另外, 一种更简单的证明是考虑将互不相同的分拆中的每个偶数  $x$  拆成  $2^k$  个  $\frac{x}{2^k}$ , 其中  $\frac{x}{2^k}$  是奇数, 容易发现这是一个双射, 因此两者的数量一定相同。

我们将容斥从  $N_{=0}$  扩展到对任意集合的  $N_{=A}$ , 这是容易的, 可以得到  $N_{=A} = \sum_{A \subseteq T} (1)^{|T|-|A|} N_{\subseteq T}$

另外, 如果我们希望求出恰好满足  $p$  个性质的集合个数, 遍历每个大小为  $p$  的子集可得  $N_{=p} = \sum_p (-1)^{k-p} \binom{k}{p} \sum_{|T|=k} N_{\supseteq T}$

如果性质集合  $E$  是同质的, 那么  $N_{=p} = \binom{n}{p} \sum_{k=p}^n (-1)^{k-p} \binom{n-p}{k-p} N_{\geq k}$

给出一个推广形式:

**定理 1.2.** 假设  $E$  是一个有限集合,  $K$  是一个域,  $f, g: 2^E \rightarrow K$  是两个函数, 那么对任意  $A, f(A) = \sum_{T \supseteq A} g(T)$  等价于对任意  $A, g(A) = \sum_{T \supseteq A} (-1)^{|T|-|A|} f(T)$

如果左边的式子成立, 那么  $\sum_{T \supseteq A} (-1)^{|T|-|A|} f(T) = \sum_{T \supseteq A} (-1)^{|T|-|A|} \sum_{U \supseteq T} g(U) = \sum_{U \supseteq A} (\sum_{U \supseteq T \supseteq A} (-1)^{|T|-|A|})$ , 由  $\sum_{k=0}^n (-1)^k \binom{n}{k} = [n=0]$  知右式即为  $g(A)$ . 将包含关系取反, 同理可得

**定理 1.3.** 假设  $E$  是一个有限集合,  $K$  是一个域,  $f, g: 2^E \rightarrow K$  是两个函数, 那么对任意  $A, f(A) = \sum_{T \supseteq A} g(T)$  等价于对任意  $A, g(A) = \sum_{T \supseteq A} (-1)^{|T|-|A|} f(T)$

这个定理的证明与上个定理几乎相同。

## 1.2 带权的容斥

我们对容斥做一个简单的扩展,即定义一个集合的权值函数  $w: X \rightarrow K$ ,  $E$  为一个性质的集合,那么  $W_{\supseteq \emptyset} = \sum T \subseteq E(-1)^{|T|} W_{\supseteq T}$ .

如果要求所有恰好满足  $p$  个性质的元素的权值和,可以发现  $W_{=p} = \sum_{k=p}^n (-1)^{k-p} \binom{k}{p} \sum_{|T|=k} W_{\supseteq T}$

为了说明带权容斥的作用,我们考虑如何对一个矩阵的积和式进行转化。积和式的定义为  $\sum_{p \in S_n} m_{1p(1)} m_{2p(2)} \dots m_{np(n)}$ , 即不带正负号的行列式。

考虑定义  $X$  为所有  $\{1, 2, \dots, n\}$  到自身的映射(不一定是排列), 一个映射的权值是  $m_{1p(1)} m_{2p(2)} \dots m_{np(n)}$ , 第  $i$  个性质是  $i$  不在映射的值域中, 那么我们要求的就是所有不满足这些性质的映射的权值和。

可以发现对于一个性质集合  $T, W_{\supseteq T} = \sum_p m_{1p(1)} m_{2p(2)} \dots m_{np(n)}$ , 其中  $p$  满足  $p(i) \notin T$ , 我们定义  $M|C$  为一个矩阵只保留列集合  $C$  之后的结果, 定义  $P(M|C)$  为这样的矩阵的每一行的乘积, 那么可以得到  $W_{\supseteq T} = P(M|\{1, 2, \dots, n\}/T)$ , 那么一个矩阵的积和式即为  $\sum_{k=1}^n (-1)^{n-k} \sum_{|T|=k} P(M|T)$ , 对于一般矩阵, 可以把积和式的复杂度降为  $O(2^n * n)$ , 状压 DP 等其它方法也能实现这一复杂度。

对于特殊矩阵, 这一算法可能进一步降低复杂度。

我们也可以用这一算法来给出一些组合恒等式。如对于全 1 矩阵, 它的积和式显然是  $n!$ , 我们可以得到  $n! = \sum_{k=1}^n (-1)^{n-k} \binom{n}{k} k^n$

## 1.3 在概率论中的应用

我们考虑从  $X$  中随机选取一个元素,  $E$  为一个性质集合, 一个随机变量  $Z: X \rightarrow 1, 2, \dots, n$ , 那么如果我们定义  $\sigma_k = \frac{1}{|X|} \sum_{|T|=k} N_{\supseteq T}$

那么我们可以得到  $\text{Prob}(Z = p) = \sum_{k=p}^n (-1)^{k-p} \binom{k}{p} \sigma_k$

我们知道  $E(Z) = \sum_{p=0}^n p \text{Prob}(Z = p) = \sum_{p=0}^n p \sum_{k=p}^n (-1)^{k-p} \binom{k}{p} \sigma_k = \sum_k k = 0^n \sigma_p = 1^n (-1)^{k-p} p \binom{k}{p} \sigma_k = \sum_{k=1}^n (-1)^{k-1} k \sigma_k \sum_{p=1}^k (-1)^{p-1} \binom{k-1}{p-1} = \sum_{k=1}^n (-1)^{k-1} k \sigma_k \sum_{p=0}^{k-1} (-1)^p \binom{k-1}{p}$

最右边的式子在  $k = 1$  时为 1,  $k > 1$  时为 0, 因此  $E(Z) = \sigma_1$ , 这也说明了期望的线性性。

同理我们也可以得到  $\text{Var}(Z) = E(Z^2) - E(Z)^2 = \sigma_1 + 2\sigma_2 - \sigma_1^2$ , 这说明方差和期望都是可以通过  $\sigma$  数列简单计算的。

我们考虑以下问题: 如果我们只知道一部分集合交的大小而非全部集合交的大小, 我们能否通过这些信息估计出全部集合的并的大小。具体来说, 假设有一个给定的  $K$ , 我们只知道所有不超过  $K$  个集合的并的大小, 我们考虑如何估计所有集合的并的大小。如果性质集合是同质的, 那么问题就等价于已知  $\sigma_1, \sigma_2, \dots, \sigma_K$ , 如何估计  $\sigma_n$ 。对于不同质的性质集合, 我们显然可以通过已知信息计算出  $\sigma_1, \dots, \sigma_k$ , 而存在一个同构的性质集合, 满足  $\sigma_1, \dots, \sigma_k$  与其相等, 因此同构的性质集合的信息量最少, 进而预测的误差最大。

事实上, 我们有如下结果: 在  $K$  不超过  $\sqrt{n}$  时, 我们没有好的办法预测所有集合的并, 我们的误差可能达到  $O(\frac{n}{K^2})$ 。当  $K$  超过  $\sqrt{n}$  时, 我们的预测值在真实值的  $1 \pm e^{O(\frac{K}{\sqrt{n}})}$  倍之间。

另外, 我们也有如下结果: 对于一个性质集合  $E$ , 定义  $Prob[E]$  为随机取出全集中一个元素, 使得它满足  $E$  中所有性质的概率,  $A_1, A_2, \dots, A_n$  和  $B_1, B_2, \dots, B_n$  为两个性质组成的集合, 如果存在某个  $\epsilon = \epsilon(n) > 0$  满足对任意  $\{1, 2, \dots, n\}$  的子集  $S$ , 有  $|Prob[\cap_{i \in S} A_i] - Prob[\cap_{i \in S} B_i]| < \epsilon$ , 我们定义  $B(n, \epsilon) = \sup |Prob[\cup_{i=1}^n A_i] - Prob[\cup_{i=1}^n B_i]|$ , 即满足上述条件的情况下至少满足  $A$  中和  $B$  中一条性质的概率之间的最大差值, 那么如果  $\log(1/\epsilon) = \Omega(\sqrt{n} \log(n))$ , 则  $B(n, \epsilon) = o(1)$ , 如果  $\log(1/\epsilon) = \Omega(\sqrt{n})$ , 则  $B(n, \epsilon) = 1 - o(1)$

上述问题的具体证明见参考资料 [1]。

## 2 莫比乌斯反演

### 2.1 莫比乌斯反演的定义

定义一个偏序集  $P$  是局部有限的, 如果每个区间  $[a, b] = \{x \in P \mid a \leq x \leq b\}$  都是有限集, 如在自然数集和整数集上定义的通常使用的大小关系, 和一个集合  $S$  的所有子集, 序定义为包含关系, 或者非 0 自然数集上用整除定义的大小关系都满足这个性质。

对于某个给定的域  $K$ , 在这个偏序集上定义  $\mathbb{A}(P) = \{f : P^2 \rightarrow K : x \not\leq y \Rightarrow f(x, y) = 0\}$ , 对于  $\mathbb{A}(P)$  上的函数使用通常的加法和数乘, 我们可以得到  $\mathbb{A}(P)$  是一个线性空间。

接下来我们定义两个函数的卷积:  $(f * g)(a, b) = \sum_{a \leq x \leq b} f(a, x)g(x, b)$ , 如果  $a \not\leq b$ , 那么  $(f * g)(a, b) = 0$

容易发现卷积具有结合律, 而且有左乘和右乘的单位元  $\delta(x, y) = \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases}$ 。

**定理 2.1.**  $f \in \mathbb{A}(P)$  有逆元当且仅当每个  $f(x, x)$  都不等于 0

如果  $f \in \mathbb{A}(P)$  有逆元, 那么  $f^{-1}(x, x)f(x, x) = \delta(x, x) = 1$ , 因此  $f(x, x) \neq 0$

如果  $f(x, x) \neq 0$ , 那么按照如下方法构造逆元  $f^{-1}$ :

$$f^{-1}(x, x) = \frac{1}{f(x, x)}$$

$$f^{-1}(x, y) = \frac{1}{f(y, y)} \left( -\sum_{x \leq z < y} f^{-1}(x, z)f(z, y) \right)$$

那么显然  $f^{-1}$  是  $f$  的左逆, 同理可以得到  $f$  存在右逆, 根据结合律可以得到左逆和右逆一定相等, 即  $f^{-1}$  是  $f$  的逆元。

接下来我们定义 zeta 函数  $\zeta \in \mathbb{A}(P)$ :

$$\zeta(x, y) = \begin{cases} 1 & x \leq y \\ 0 & x \not\leq y \end{cases}$$

接下来定义莫比乌斯函数为 zeta 函数的逆元, 即  $\mu = \zeta^{-1} \in \mathbb{A}(P)$

可以发现某个偏序集  $P$  上的莫比乌斯函数满足以下性质:

$$\mu(a, a) = 1 (a \in P)$$

$$\mu(a, b) = -\sum_{a \leq z < b} \mu(a, z) = -\sum_{a < z \leq b} \mu(z, b) (a < b)$$

根据以上性质, 我们可以定义较为一般的莫比乌斯反演:

**定理 2.2.** 假设  $P$  是一个局部有限的偏序集,  $f, g: P \rightarrow K$  是两个定义在  $P$  上的函数, 可以得到以下结论:

$$(1) f(a) = \sum_{x \leq a} g(x) (\forall a \in P) \text{ 等价于 } g(a) = \sum_{x \leq a} f(x) \mu(x, a)$$

$$(2) f(a) = \sum_{x \geq a} g(x) (\forall a \in P) \text{ 等价于 } g(a) = \sum_{x \geq a} f(x) \mu(a, x)$$

其中 (1) 假设了偏序集中存在最小元素  $0$ , (2) 假设了偏序集中存在最大元素  $1$ 。

定义  $f'(0, a) = f(a)$ ,  $x \neq 0$  时  $f'(x, y) = 0$ ,  $g'(0, a) = g(a)$ ,  $x \neq 0$  时  $g'(x, y) = 0$ , 那么 (1) 等价于  $f' = g' * \zeta$ , 也就等价于  $f' = g' * \mu$

(2) 与 (1) 是几乎相同的, 我们只需要通过  $f'(a, 1) = f(a)$  和  $g'(a, 1) = g(a)$  来定义  $f', g'$  即可。

## 2.2 差分形式的微积分

对于一个长度为  $n$  的链  $C_n = 0 < 1 < \dots < n$ , 我们可以得到  $\mu(i, j) = \begin{cases} 1 & i = j \\ -1 & i \neq j \\ 0 & j \geq i + 2 \end{cases}$

我们可以得到  $f(n) = \sum_{k=0}^{n-1} g(k)$  等价于  $g(n) = f(n+1) - f(n)$ , 这是非常显然的。

我们定义  $\Delta$  为前向的差分运算符号, 即  $\Delta f(x) = f(x+1) - f(x)$ ,  $\Sigma$  为前缀和符号, 可以得到  $\Sigma g = f$  等价于  $\Delta f = g$

根据这一性质, 我们容易得到  $\sum_{k=a}^b g(k) = f(b+1) - f(a)$

我们将这一性质应用到下降幂上, 可以得到  $\Delta x^n = nx^{n-1}$

同理可以得到对  $n \geq 0$ , 一定有  $\Sigma x^n = \frac{x^{n+1}}{n+1}$

根据这一性质, 我们事实上可以将下降幂延拓到负数, 即  $x^{-n} = \frac{1}{(x+1)(x+2)\dots(x+n)}$ , 那么上述规律也可以延拓到负数。

我们考虑  $\Sigma x^{-1}$ , 可以发现  $\Sigma x^{-1} = 1 + \frac{1}{2} + \dots + \frac{1}{x}$ , 我们称这个数列为  $H_x$ 。

这是与实际的微积分相当类似的, 这种形式的函数也可以分部积分。

由于  $\Delta(u(x)v(x)) = u(x+1)v(x+1) - u(x)v(x) = (\Delta u(x))v(x+1) + u(x)(\Delta v(x))$ , 我们可以得到  $\sigma u \Delta v = uv - \sigma(\Delta u)Tv$ , 其中  $Tv$  指的是将函数左移一位, 即  $Tv(x) = v(x+1)$

一个例子是考虑对给定的  $n, m$ , 求  $\sum_{k=1}^n \binom{k}{m} H_k$

$$\begin{aligned} \text{可以发现 } \sum_{k=1}^n \binom{k}{m} H_k &= \sum_{k=1}^{n+1} \binom{x}{m} H_x = \binom{x}{m+1} H_x \Big|_1^{n+1} - \sum_{k=1}^{n+1} \frac{1}{x+1} \binom{x}{m+1} = \binom{n+1}{m+1} H_{n+1} - \frac{1}{m+1} \sum_{k=1}^{n+1} \binom{x}{m} \\ &= \binom{n+1}{m+1} H_{n+1} - \frac{1}{m+1} \binom{x}{m+1} \Big|_0^{n+1} = \binom{n+1}{m+1} \left( H_{n+1} - \frac{1}{m+1} \right) \end{aligned}$$

## 2.3 莫比乌斯函数

我们考虑莫比乌斯函数的其它性质, 如果  $[a, b]$  和  $[c, d]$  是同构的, 那么显然有  $\mu(a, b) = \mu(c, d)$ , 可以根据  $\mu$  的定义归纳证明这一性质成立。

考虑如果在  $P_1$  和  $P_2$  上分别有一个偏序, 那么将  $P_1$  的元素和  $P_2$  的元素看作一个二元组定义二维偏序, 具体来说, 定义  $P = P_1 \times P_2, (a, b) \leq_P (c, d)$  当且仅当  $a \leq_{P_1} c$  并且  $b \leq_{P_2} d$ , 那么我们给出以下结论:

**定理 2.3.**  $\mu((a, b), (c, d)) = \mu_1(a, c)\mu_2(b, d)$

**证明.** 如果  $a = c$  且  $b = d$ , 显然成立。不妨设  $b < d$ , 考虑使用归纳法, 可以得到  

$$\mu((a, b), (c, d)) = -\sum_{(a,b) \leq (x,y) < (c,d)} \mu((a, b), (x, y)) = -\sum_{b \leq y < d} \mu((a, b), (c, y)) - \sum_{a \leq x < c} \sum_{b \leq y < d} \mu((a, b), (x, y)) =$$

$$\mu_1(a, c) \left( -\sum_{b \leq y < d} \mu_2(b, y) \right) - \sum_{a \leq x < c} \mu_1(a, x) \left( -\sum_{b \leq y \leq d} \mu_2(b, y) \right) = \mu_1(a, c)\mu_2(b, d)$$

这是因为  $b < d$  时  $\sum_{b \leq y \leq d} \mu(b, d) = 0$ , 所以上式中第二个求和一定为 0。

对于有限的偏序集  $P$ , 如果其中有最小元素和最大元素  $0_P$  和  $1_P$ , 那么我们称  $\mu(P) = \mu(0_P, 1_P)$

那么对于两个这样的偏序集  $P, Q$ , 我们可以得到  $\mu(P \times Q) = \mu(P)\mu(Q)$

因此对于偏序集  $C = C_{k_1} C_{k_2} \dots C_{k_t}$ , 我们可以得到  $\mu(C) = \begin{cases} (-1)^t & k_1 = k_2 = \dots = k_t = 0 \\ 0 & \text{存在 } k_i \geq 2 \end{cases}$

我们考虑非零自然数上由整除定义的偏序关系, 可以得到每个区间事实上就是若干个链的乘积, 因此我们可以得到通常的莫比乌斯函数:

$$\mu(l, m) = \begin{cases} 1 & l = m \\ (-1)^t & \text{如果 } \frac{m}{l} = p_1 p_2 \dots p_t \text{ 是互不相同的质因数} \\ 0 & \text{其它情形} \end{cases}$$

另外, 由于  $[l, m]$  总是和  $[1, \frac{m}{l}]$  同构, 因此  $\mu(l, m) = \mu(1, \frac{m}{l})$ , 可以简记为  $\mu(\frac{m}{l})$

## 2.4 计算一类特殊矩阵的行列式

假设  $P = \{a_1, a_2, \dots, a_n\}$  是一个偏序集, 对于任意两个元素  $a_i, a_j$ , 存在唯一的极大元素  $a_i \wedge a_j \leq a_i, a_j$ , 如在以整除为偏序的情形下极大元素就是  $\gcd(a_i, a_j)$ , 在以集合的包含关系为偏序的情况下极大元素就是  $a_i \cap a_j$

如果存在函数  $f, g$  使得  $f(a) = \sum_{x \leq a} g(x)$ , 那么如果有一个  $n \times n$  矩阵  $F$  满足  $F_{ij} = f(a_i \wedge a_j)$ , 我们给出以下结论:

**定理 2.4.**  $\det(F) = g(a_1)g(a_2)\dots g(a_n)$

为了证明这个结论, 我们考虑定义一个对角矩阵  $G$  满足  $G_{ii} = g(a_i)$ , 和矩阵  $Z$  满足  

$$Z_{ij} = \begin{cases} 1 & a_i \leq a_j \\ 0 & a_i \not\leq a_j \end{cases}$$

那么可以得到  $(Z^T G Z)_{ij} = \sum_{k=1}^n \sum_{l=1}^n n z_{ki} g_{kk} z_{kj} = \sum_{a_k \leq a_i, a_k \leq a_j} g(a_k) = \sum_{a_k \leq a_i \wedge a_j} g(a_k) = f(a_i \wedge a_j) = F_{ij}$



因此  $F = Z^T G Z$ , 而考虑如果按照拓扑序依次排列  $a_i, a_j$ , 那么可以得到  $Z$  经过若干次同时交换第  $i$  行和第  $j$  行, 第  $i$  列和第  $j$  列的操作后可以化为上三角矩阵, 因此  $\det(Z) = \det(Z^T) = 1$  因此我们可以得到  $\det(F) = \det(G) = g(a_1)g(a_2)\dots g(a_n)$ , 也就证明了上述结论。

**例题 2.1** (特殊矩阵行列式的例子). 给定  $n$ , 如果有  $A_{ij} = \gcd(i, j)$ , 求矩阵  $A$  的行列式.  
 $n \leq 10^6$

我们可以直接应用以上结论, 容易得到  $f(n) = n, g(n) = \varphi(n)$ , 即欧拉函数, 因此这一矩阵的行列式为  $\varphi(1)\varphi(2)\dots\varphi(n)$ , 这是容易计算的。

### 3 不动点定理

假设我们有一个有限集  $S$ , 如果我们把它划分为  $S^+$  和  $S^-$ , 即一个正的部分和一个负的部分, 并定义一个一一映射  $\phi$ , 满足这个映射的所有非不动点都满足  $\phi x$  和  $x$  符号不同, 我们关注的两个集合的元素数量差值, 那么容易发现  $|S^+| - |S^-| = |\text{Fix} S^+| - |\text{Fix} S^-|$

如果  $S^-$  中没有任何不动点, 那么原式可以化简为  $|S^+| - |S^-| = |\text{Fix} S|$

在一些问题中, 我们需要求解某个特定集合  $X$  的大小, 那么可以构造一个集合  $S = S^+ \cap S^-$ , 并且  $X \subseteq S^+$ , 然后构造一个映射  $\phi$  满足  $\text{Fix} S^+ = X, \text{Fix} S^- = \emptyset$ , 那么就可以得到  $|X| = |S^+| - |S^-|$

我们考虑这种方法的具体应用。

#### 3.1 卡特兰数

求卡特兰数  $C_n$ , 即从  $(0, 0)$  走到  $(n, n)$  且始终不越过直线  $y = x$  的方案数。

定义从  $(0, 1)$  走到  $(n, n+1)$  的集合  $S^+$ , 从  $(1, 0)$  走到  $(n+1, n)$  的路径集合为  $S^-$ ,  $S = S^+ \cap S^-$ , 那么我们构造这样的映射: 如果  $x$  是  $S$  中的路径, 并穿过了  $y = x$ , 那么将第一个交点之前的部分全部翻转, 否则不做任何操作。容易发现这个映射是一一映射, 且满足上面所说的条件,  $S^-$  中没有不动点,  $S^+$  中的不动点就是要求的答案, 因此可以得到卡特兰数  $C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{\binom{2n}{n}}{n+1}$   
这个例子也是相当经典的。

#### 3.2 范德蒙德矩阵行列式

求范德蒙德矩阵的行列式, 即对一个数列  $x, n * n$  矩阵  $A$  满足  $A_{ij} = x_j^{n-i}$ , 我们希望证明  $\det(A) = \prod_{1 \leq i < j \leq n} (x_i - x_j)$

根据行列式的定义, 我们可以得到  $\det(A) = \sum_{\sigma \in S(n)} (\text{sign} \sigma) x_{\sigma_1}^{n-1} x_{\sigma_2}^{n-2} \dots x_{\sigma_n}^0$

可以发现行列式可以转化为  $(-1)^m x_1^{a_1} x_2^{a_2} \dots x_n^{a_n}$ , 其中  $\sum_i = 1^n a_i = \frac{n(n-1)}{2}, m$  为  $\sum_{i < j} [a_i > a[j]]$



对于每个大小为  $n$  的竞赛图的每条边  $e: u \rightarrow v$ , 我们定义它的权重为  $w(e) = x_u$ , 他的正负号  $sign(e) = \begin{cases} 1 & u < v \\ -1 & u > v \end{cases}$

定义一个竞赛图的权值  $w(T) = \prod_{e \in T} w(e)$ , 它的正负号  $sign(T) = \prod_{e \in T} sign(e)$

容易发现一个竞赛图的权值就是每个点权值的出度次方的乘积, 对于所有无环的竞赛图, 容易证明这样的竞赛图在给定出度序列的情况下是唯一的, 且它的出度序列一定是  $0$  到  $n-1$  的排列.

并且可以发现, 无环的竞赛图恰好和行列式的每一行一一对应, 因此可以得到所有无环的竞赛图的权值之和恰好等于范德蒙德矩阵的行列式.

如果对所有竞赛图的权值求和, 可以发现每条边的贡献是独立的, 因此贡献和恰好就是  $\prod_{1 \leq i < j \leq n} (x_i - x_j)$ , 因此我们要证明的就是所有有环的竞赛图的权值之和为  $0$ .

考虑证明给定一个非  $0$  到  $n-1$  排列的出度序列, 我们证明所有满足这一出度序列的竞赛图的权重之和为  $0$ . 容易发现, 这样的竞赛图的权值的绝对值一定相同, 我们只需要证明符号为正的竞赛图和符号为负的竞赛图数量相等.

我们定义  $S^+$  为所有满足给定出度序列, 且符号为正的竞赛图.  $S^-$  为所有满足给定出度序列, 且符号为负的竞赛图.

容易发现, 既然出度序列不是排列, 那么一定存在两个点出度相等, 我们找出最小的  $i$  满足  $a_i$  出现多次, 再找出最小的  $j$  满足  $a_j = a_i$ , 假设  $i, j$  之间连边方向为  $i \rightarrow j$ .

考虑除了  $i$  和  $j$  以外的所有点, 按照它们和  $i, j$  的连边方向进行分类, 定义满足方向为  $i \rightarrow k \rightarrow j$  的所有  $k$  的集合为  $S_1$ , 满足方向为  $j \rightarrow k \rightarrow i$  的所有  $k$  的集合为  $S_2$ , 由于  $i, j$  出度相等, 且  $S_1, S_2$  以外的点不影响  $i, j$  的出度差, 容易发现  $|S_2| = |S_1| + 1$ , 我们考虑将  $(i, j)$  方向翻转, 并对于所有  $x \in S_1 \cup S_2$ , 将  $(x, i)$  和  $(x, j)$  方向翻转, 那么可以发现入度序列一定不变, 却我们进行了奇数次翻转操作, 因此竞赛图的符号取反.

这是一个满足上述条件的映射, 且不存在任何不动点. 因此, 我们证明了  $|S^+| - |S^-| = 0$ , 进而证明了  $\det(A) = \prod_{1 \leq i < j \leq n} (x_i - x_j)$

### 3.3 斜对称矩阵行列式

我们希望求出一个斜对称矩阵 ( $A^T = -A$ ) 的矩阵的行列式.

容易发现, 如果矩阵的行数为奇数, 那么  $\det(A) = \det(A^T) = \det(-A) = (-1)^n \det(A) = -\det(A)$ , 因此  $\det(A) = 0$ , 下文中我们只考虑  $A$  为  $n \times n$  矩阵, 且  $n$  为偶数的情形.

我们定义一个大小为  $\frac{n}{2}$  的匹配  $\mu$  满足  $\mu = i_1 j_1, i_2 j_2, \dots, i_{\frac{n}{2}} j_{\frac{n}{2}}$ , 其中  $i_k < j_k, i_k < i_{k+1}$

定义一个匹配的权值为  $(-1)^k A_{i_1 j_1} A_{i_2 j_2} \dots A_{i_{\frac{n}{2}} j_{\frac{n}{2}}}$ , 其中  $k$  表示匹配之间交的个数, 即有多少组  $x_1, x_2$  满足  $i_{x_1} < i_{x_2} < j_{x_1} < j_{x_2}$

定义所有匹配的权值之和为  $Pf(A)$ , 我们希望证明  $\det(A) = Pf(A)^2$

按定义显出  $A$  的行列式, 即  $\det A = \sum_{\sigma \in S(n)} (sign \sigma) a_{1\sigma(1)} a_{2\sigma(2)} \dots a_{n\sigma(n)}$

将一个排列看作有  $n$  条形如  $i \rightarrow \sigma(i)$  的有向图, 显然这个图有若干个环组成, 定义  $S$  为有至少一组奇环的排列,  $S^+$  为这些排列中符号为正的,  $S^-$  为这些排列中符号为负的.

对于所有这样的排列  $P$ , 我们把所有奇环按下标集合中的最小值从小到大排序, 并称  $\phi P$  为将第一个奇环翻转, 即将环中的每个  $P_i$  变为  $P_i^{-1}$ , 相当于进行了奇数次元素交换, 排列的符号取反.

这个映射是满足上述条件的, 并且没有任何不动点, 因此  $|S^+| = |S^-|$ , 因此我们只需要考虑只含偶环的排列.

可以发现, 如果我们把两个大小为  $\frac{n}{2}$  的匹配拼到一起形成一个图, 它恰好就是一个由偶环组成的图, 但是匹配中的边都是无向边, 因此我们还要考虑确定环的方向.

假设我们忽略排列和匹配的符号问题, 可以给出两个匹配和一个只由偶环组成的排列之间的一一映射.

如果我们有两个匹配  $\mu_1, \mu_2$ , 每次寻找最小的未访问点, 从  $\mu_1$  开始依次走两个匹配中的边, 显然会得到一个偶环, 递归直到得到一个排列.

如果我们有一个排列, 那么依然每次寻找最小的未访问点, 从这个点开始访问它所在的环, 每遍历到一条边, 就依次将它划分给  $\mu_1$  和  $\mu_2$ .

我们可以证明这样的映射是一一映射, 接下来只需要证明匹配的符号的乘积与再行列式中符号相同.

注意匹配是一定从下标小的连向下标大的, 而行列式中每个  $i > \sigma_i$  会额外贡献一个负号, 假设  $\sigma = \sigma_1 \dots \sigma_t$ , 每个  $\sigma_i$  是一个偶环, 容易发现  $\sigma$  的符号与  $(-1)^t$  相同, 因此我们要证明的就是  $\text{sign}(\mu_1)\text{sign}(\mu_2) = (-1)^{e(\sigma)+t}$ .

考虑交换排列的环的形式中  $i$  和  $i+1$  的位置, 可以发现除非  $i, i+1$  相邻, 所有符号不变, 如果  $i, i+1$  相邻, 那么  $e(\sigma)$  改变, 含有  $(i, i+1)$  这条边的匹配的正负号也改变, 因此  $\text{sign}(\mu_1)\text{sign}(\mu_2) = (-1)^{e(\sigma)+t}$  在交换前后的成立性是等价的.

因此, 若干次交换之后我们可以发现每个偶环的长度始终不变, 环的形式可以任意排列. 因此, 可以假设我们最后将每条环都交换成了  $l \rightarrow l+1 \rightarrow l+2 \dots \rightarrow r \rightarrow l$  的形式, 这样的每个环会对  $t$  和  $e(\sigma)$  各有 1 的贡献, 而对  $\mu_1$  和  $\mu_2$  的符号没有任何影响, 因此等号两边均为 1, 原命题一定成立.

因此, 这个矩阵的行列式就是所有匹配的二元组的权值乘积的和, 也就是  $Pf(A)^2$ , 我们证明了这个命题成立.

### 3.4 不动点定理

假设我们有两个有限集  $S, T$ , 他们都能被划分为正的部分和负的部分, 也就是  $S^+, S^-, T^+, T^-$ , 假设我们在  $S$  和  $T$  上各有一个符合上述条件的映射  $\phi$  和  $\psi$ , 并且还有一个  $S$  和  $T$  之间保证符号不变的一一映射  $f$ , 如果以上条件满足, 显然有  $|S^+| = |T^+|, |S^-| = |T^-|$ , 进而  $|S^+| - |S^-| = |T^+| - |T^-|$ , 因此  $|Fix_\phi S| = |Fix_\psi T|$ , 我们希望给出一个  $|Fix S^+|$  和  $|Fix T^+|$  之间的双射, 我们不能直接使用  $f$ , 因为虽然  $f$  保证符号不变, 但不一定保证不动点映射后仍然是不动点.



我们有如下结论:

**定理 3.1.** 对于每个  $x \in \text{Fix}_\phi S$ , 存在一个整数  $m(x)$  满足  $f(\phi f^{-1} \psi f)^{m(x)}(x) \in \text{Fix}_\psi T$ , 且映射  $\beta: x \rightarrow f(\phi f^{-1} \psi f)^{m(x)}(x)$  是一个不动点之间的双射。

考虑将能通过  $f$  相互映射的  $S, T$  中的元素连边, 并将通过  $\phi$  和  $\psi$  相互映射的不同点之间连边, 那么可以得到一个满足所有点的度数都不超过 2, 其中度数为 1 的恰好是  $\text{Fix}_\phi S$  和  $\text{Fix}_\psi T$  中的点, 因此可以发现原图形成了若干个环和链, 其中链的端点恰好是  $\text{Fix}_\phi S$  和  $\text{Fix}_\psi T$  中的点, 因此从链的一端一定能走到另一端, 且形成了一个双射。考虑一个应用这一定理的例子:

**例题 3.1.** 构造将  $n$  拆分成奇数的方案和将  $n$  拆分成不同的数的方案之间的双射。

定义  $S$  为所有拆分的二元组  $(\lambda, \mu)$ , 其中两组拆分的总和是  $n$ ,  $\mu$  必须满足所有数都是偶数且互不相同。定义  $S^+$  表示  $\mu$  中有偶数个数的二元组,  $S^-$  为  $\mu$  中有奇数个数的二元组。在这个例子中,  $S$  和  $T$  完全相同, 因此可以取  $f$  满足  $f(x) = x$ 。

构造  $\phi$  为如下映射: 取出  $\lambda$  和  $\mu$  中最小的偶数, 如果有多个取  $\mu$  中的, 并将这个数从  $\lambda$  移动到  $\mu$ , 或从  $\mu$  移动到  $\lambda$ 。如果没有任何偶数, 那么  $(\lambda, \mu)$  为不动点, 这个映射的不动点恰好是  $(\lambda, \emptyset)$ , 其中  $\lambda$  是  $n$  满足所有数都为奇数的拆分。

构造  $\psi$  为如下映射: 取出  $\lambda$  中最小的出现了两次的数, 比较它和  $\mu$  中最小的偶数, 如果有多个取  $\mu$  中的, 并将这个数从  $\lambda$  移动到  $\mu$ , 或将这种数的两次出现从  $\mu$  删除, 将这种数的两倍插入到  $\lambda$ 。如果  $\lambda$  中没有任何出现两次的数且  $\mu$  为空, 那么  $(\lambda, \mu)$  为不动点, 这个映射的不动点恰好是  $(\lambda, \emptyset)$ , 其中  $\lambda$  是  $n$  满足所有数都不同的拆分。

那么构造双射的算法就是交替使用  $\phi$  和  $\psi$ , 事实上可以发现这种算法本质上就是将所有偶数拆成  $2^k$  个奇数, 或者将所有奇数二进制拆分成不同的偶数。

另外, 本题中也存在不同的满足不动点相同的映射, 它们能得到类似的结果。

## 4 总结

我们可以看出, 这些方法对定理的证明以及 OI 中的问题的解决有所帮助, 且能应用于比较广泛的领域。

希望本文可以起到抛砖引玉的作用, 期待更多使用这类方法的问题在 OI 中出现。

## 5 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练彭思进的指导。

感谢梁霄教练、叶金毅教练对我的培养与教导。



感谢父母对我的陪伴与支持。

感谢与我交流讨论的同学。

## 参考文献

- [1] Nathan Linial and Noam Nisan. Approximate inclusion-exclusion. 11(1), 1990.
- [2] Wikipedia. Inclusion - exclusion principle. [https://en.wikipedia.org/wiki/Inclusion-exclusion\\_principle](https://en.wikipedia.org/wiki/Inclusion-exclusion_principle).
- [3] Wikipedia. Partition (number theory). [https://en.wikipedia.org/wiki/Partition\\_\(number\\_theory\)](https://en.wikipedia.org/wiki/Partition_(number_theory)).
- [4] 管晏如. 超立方体 (hypercube) 及其相关算法初探. 2023年信息学奥林匹克中国国家队候选队论文集, 2023.



# 使用生成函数刻画随机过程停时的一类技巧

杭州学军中学教育集团文渊中学 叶开

## 摘要

对随机过程的停时信息的刻画是算法竞赛中一类常见问题。经过归纳总结，本文给出了使用生成函数刻画与解决这类问题的一些常用工具与方法，包括 Laplace 变换、让随机过程不停时等。

## 1 基本定义

笔者将之后可能用到的基本定义同一罗列于此，便于之后阐述。

**定义 1.** (离散随机变量) 离散随机变量是在一个有限或者无限可列的样本空间  $\Omega$  中服从某一分布的随机变量。之后简称之为变量。

**定义 2.** (随机事件) 称一个关于变量  $x$  的随机事件是一个集合  $S \subseteq \Omega_x$ ，其中  $\Omega_x$  为变量  $x$  对应的样本空间。当满足  $x \in S$  时称该随机事件发生，否则称该随机事件未发生。简记随机事件  $S$  发生的概率为  $P(S)$ 。之后简称之为事件。

**定义 3.** (离散随机过程) 下文考虑的离散随机过程为变量族  $X = \{x_0, x_1, x_2, \dots\}$ ，满足  $\forall t \in \mathbb{N}$ ， $x_t$  服从由序列  $\langle x_0, x_1, x_2, \dots, x_{t-1} \rangle$  决定的一个分布。之后简称之为过程。

**定义 4.** (马尔可夫链) 过程  $X = \{x_0, x_1, x_2, \dots\}$  若满足  $\forall t \in \mathbb{N}^+$ ， $x_t$  服从由  $x_{t-1}$  决定的一个分布，这样的过程称为马尔可夫链。

**定义 5.** (过程的停时) 考察与过程的各个变量依次相关的事件  $S_0, S_1, S_2, \dots$ 。假如在时刻  $T \in \mathbb{N}$  事件  $S_T$  达成，那么过程终止 (称过程发生停时)， $T$  即称为该过程的停时。容易发现一个过程的停时是一个变量。在不引起歧义时，简称这些事件为停时事件。

**定义 6.** (普通生成函数) 对于实数列  $a_0, a_1, a_2, \dots$ ，称形式幂级数  $A(z) = \sum_j a_j z^j$  为其普通生成函数 (OGF)。

**定义 7.** (指数生成函数) 对于实数列  $a_0, a_1, a_2, \dots$ ，称形式幂级数  $\hat{A}(z) = \sum_j a_j z^j / j!$  为其指数生成函数 (EGF)。

**定义 8.** (概率生成函数) 对  $\mathbb{N}$  上变量  $x$ ，称序列  $\{P(x = j)\}$  的 OGF  $\sum_j P(x = j) z^j$  和 EGF  $\sum_j P(x = j) z^j / j!$  为变量  $x$  的概率生成函数 (PGF)。不加指明时指代 OGF 形式。



**定义 9.** (过程的 PGF) 对某过程各变量上的事件  $S_0, S_1, S_2, \dots$ , 称序列  $\{P(S_j)\}$  的 OGF  $\sum_j P(S_j)z^j$  和 EGF  $\sum_j P(S_j)z^j/j!$  为这个过程及这些事件的 PGF。不加指明时指代 OGF 形式。

## 2 基本技术

由于生成函数只是刻画问题的工具, 为了计算答案往往需要一些比较优秀的算法来对生成函数进行快速运算。

**技术 1.** (ODE) 对于一个生成函数, 假设其满足一个项数比较少的常微分方程 (ODE), 其前若干项系数可以在线性时间内求出。

**技术 2.** (多项式乘法) 在一些性质良好的系数环 (譬如  $\mathbb{F}_{998244353}$ ) 下, 多项式乘法可以在关于多项式长度的线性对数时间内解决。

**技术 3.** (分治卷积算法) 在允许使用线性对数复杂度的多项式乘法时, 给定若干个多项式  $F_1, F_2, \dots, F_n$ , 现要求出其乘积, 并假设  $S = n + \sum \deg F$ , 可以在  $O(S \log^2 S)$  的时间内解决该问题。进一步的, 形如分式求和、求若干多项式的前缀积之和等问题也可以采用类似的方法做到同样级别的复杂度。

**技术 4.** (自变量平移) 在允许使用线性对数复杂度的多项式乘法时, 一个多项式复合  $z + c$  可以在关于多项式次数的线性对数复杂度内完成。

等等。更进一步的关于多项式相关问题的快速计算的讨论可以参阅参考文献 2。

## 3 使用 PGF 刻画变量

现有一个  $\mathbb{N}$  上的变量, 希望刻画其一些信息, 譬如期望, 应当如何用其 PGF 刻画?

**定理 1.** 设变量  $x$  的 PGF 为  $F(z)$ , 则  $F(1) = 1$ 。

**定理 2.** 设变量  $x$  的 PGF 为  $F(z)$ , 则  $x$  的期望为  $F'(1)$ 。

证明

$$\mathbb{E}(x) = \sum_j jP(x=j) = \sum_j j[z^j]F(z) = \sum_j [z^j]F'(z) = F'(1)$$

■

进一步地, 还可以求出  $x$  的方差、 $k$  阶矩等。但这些与正文无关, 故略去。

## 4 使用 PGF 刻画过程

一般使用 PGF 刻画时, 都希望得到一个过程的一些信息, 譬如停时的期望。

**定理 3.** 设一个会发生停时的过程关于停时事件的 PGF 为  $F(z)$ , 则其同时也是停时  $T$  的 PGF。



证明  $[z^t]F$  是  $t$  时刻发生停时的概率，也即  $P(T = t)$ 。 ■

于是只用得到这个 PGF 的性质，即可快速统计关于停时所需要的信息。

为了得到这个 PGF 的性质，常常需要关心多个 PGF 之间的联系。使用一种通常称为符号化方法<sup>1</sup>的技术可以解决这一问题。

**定理 4.** 同时再设一个会发生停时的过程关于直到某时刻还未发生停时的事件的 PGF 为  $G(z)$ ，则有

$$F(z) + G(z) = 1 + zG(z)$$

证明 左式考虑了所有时刻的状态出现的状态概率：在该状态停时，或者还未停时。右式考虑了每个  $t > 0$  的时刻的状态只可能由一个还未停时的状态转移得到。显然两者相等。 ■

**定理 5.**  $\mathbb{E}(T) = G(1)$ 。

证明 对  $z$  求导得  $F'(z) + G'(z) = 1 + G(z) + zG'(z)$ 。取 1 代入就给出  $F'(1) = G(1)$ 。又有  $\mathbb{E}(T) = F'(1)$ ，故  $\mathbb{E}(T) = G(1)$ 。 ■

这也恰恰符合一个常见的组合意义：停时等于未停时的时刻数目，从而期望停时即为各时刻处未停时的概率和。

**例题 1.** (歌唱王国<sup>2</sup>) 给定一个长度为  $n$  的字符串  $s_{1 \sim n}$ ，字符集大小为  $m$ ，每次会独立均匀随机地报出一个字符，当已经报出的字符中存在连续一段等于给定字符串时过程停时，求期望停时对 10000 取模后的值。  $1 \leq n, m \leq 10^5$ ，询问组数不超过 50 组。

解法 按照刚刚的做法直接设  $F$  和  $G$ ，现在还需要得到另一个关系。有

$$\left(\frac{z}{m}\right)^n \times G = \sum_{1 \leq j \leq n, s_{1 \sim j} = s_{n-j+1 \sim n}} \left(\frac{z}{m}\right)^{n-j} \times F$$

理由是往当前还未停时的串末尾塞原串，只可能是从一个已在某个 Border 处停时的结果往末尾再继续塞。取 1 代入就给出

$$G(1) = \sum_{1 \leq j \leq n, s_{1 \sim j} = s_{n-j+1 \sim n}} m^j$$

即为答案。使用 kmp 算法或者字符串哈希算法即可做到  $O(n)$ 。 ♠

## 5 使用 EGF 刻画时间轴

**定理 6.** EGF 的乘法对应了序列的二项卷积。

因此，使用 EGF 的不定元的次数来刻画时间轴，可以很好的描述「分配时间标号」的关系。

<sup>1</sup>一份可以参考的学习资料：<https://www.luogu.com.cn/blog/zyxxs/zu-ge-jie-gou-fu-hao-hua-xue-xi-bi-ji>。

<sup>2</sup>2006 年 IOI 中国队选拔赛 (CTSC2006)，<https://www.luogu.com.cn/problem/P4548>。

然而如果使用的是 EGF 形式的 PGF，又该怎么统计信息呢？

定义 10. (Laplace 变换) 规定  $F(z)$  的 Laplace 变换结果为

$$\mathcal{L}F(z) = \int_0^{+\infty} F(xz)e^{-x}dx$$

其中  $\mathcal{L}$  为 Laplace 变换算子。

定理 7. 设一个序列  $a_0, a_1, a_2, \dots$  的 OGF 为  $A(z)$ ，EGF 为  $\hat{A}(z)$ ，则  $A(z) = \mathcal{L}\hat{A}(z)$ 。

证明 只用考虑

$$\forall k \in \mathbb{N}, \int_0^{+\infty} x^k e^{-x} dx = k!$$

这样 EGF 的每个  $z^k$  会带一个  $x^k$ ，也即变成了  $k!$ ，刚好消掉了 EGF 所带的  $1/k!$  因子，也就变成了 OGF。 ■

因此总是可以把 EGF 转化成 OGF 后再采用先前的方法来刻画。在实践中，通常所见到的 EGF 会形如若干  $z^a e^{bz}$  之和（其中  $a \in \mathbb{N}$ ， $a, b$  为常数），此时有结论

定理 8. Laplace 变换算子是线性算子。

定理 9.  $\mathcal{L}z^a e^{bz} = a!z^a/(1-bz)^{a+1}$ 。

从而可以直接逐个算出贡献。

实际刻画时，仅仅是 EGF 还不足以很好的刻画所有方案，这时可能就需要对常数步进行基本的枚举，最后再统计这常数步的贡献即可。

接下来将会看到，仅仅是上述内容就足以解决很大一部分相关的题目了。

例题 2. (Gachapon<sup>3</sup>) 现有一随机数生成器，每次生成  $1 \leq j \leq n$  的概率为  $A_j/\sum_{i=1}^n A_i$ ，使用其不断独立随机生成。当对任意  $1 \leq j \leq n$  都有  $j$  被生成了至少  $B_j$  次时过程停时。求期望停时。 $A_i, B_i \in \mathbb{N}^+$ ， $1 \leq n, \sum A_i, \sum B_i \leq 400$ 。

解法 直接做不好做，不妨枚举最后一步生成的数，最后把 1 的贡献加回去即可。

设  $S = \sum_i A_i$ ，那么立刻就有答案为

$$1 + \left( \mathcal{L} \left( \sum_{i=1}^n \frac{(A_i z/S)^{B_i-1}}{(B_i-1)!} \prod_{j \neq i} \left( \exp(A_j z/S) - \sum_{k < B_j} \frac{(A_j z/S)^k}{k!} \right) \right) \right)' \circ 1$$

设  $u = \exp(z/S)$ ，容易把内部的式子经过暴力乘法表示成关于  $u, z$  的多项式，复杂度不会超过  $O(v^3)$ （其中  $v = \max\{S, \sum B\}$ ）。

最后直接把每个  $z^a u^b$  对答案的贡献单独计算一遍即可。 ♠

事实上可以发现，这样不仅能求出停时，还能求出以某些事件发生停时的概率。

定理 10. 设一个会发生停时的过程在  $t$  时刻发生停时的事件为  $S_t$ ，考察  $T_t \subseteq S_t$ ，考虑  $T_0, T_1, T_2, \dots$  对应的 PGF  $F(z)$ ，那么  $F(1)$  即为因为这些事件而停时的概率。

<sup>3</sup>AtCoder Grand Contest 038, [https://atcoder.jp/contests/agc038/tasks/agc038\\_e](https://atcoder.jp/contests/agc038/tasks/agc038_e)。

**例题 3.** (猎人杀<sup>4</sup>) 现在有  $n$  个人, 每人有一个权值  $w_i$ 。每次会在剩下的人集合中, 以  $w_i$  比例的概率独立随机选择第  $i$  个人让其离开, 重复直到所有人都离开。求最后离开的人是第一个人的概率, 对 998244353 取模。 $w_i \in \mathbb{N}^+$ ,  $1 \leq \sum_{i=1}^n w_i \leq 10^5$ 。

**解法** 由于概率会变化, 看上去这并不好直接做。

但是注意到, 如果允许选择一个已经离开的人再次离开, 那么这个就变成了和上一题类似的问题, 只是  $B_i = 1$ 。

这样, 令  $S = \sum w_i$ , 就有答案为

$$\frac{w_1}{S} \left( \mathcal{L} \left( \prod_{j=2}^n (\exp(w_j z/S) - 1) \right) \right) \circ 1$$

设  $u = \exp(z/S)$ , 使用分治卷积算法在  $O(S \log^2 S)$  时间内计算出关于  $u$  的多项式后即可对每个  $u^a$  算出对答案的贡献, 进而求出答案。

使用  $f = \exp \ln f$  (其中  $f = \prod_{j \neq 1} (1 - u^{w_j})$ ) 的技巧可以进一步做到  $O(S \log S)$ , 此处略去。♠

**例题 4.** (通用测评号<sup>5</sup>) 现在有  $n$  个盒子, 每个盒子最多放  $a$  个球, 初始均为空。不断独立均匀随机选取一个不满的盒子放入一个球, 直到所有盒子内的球数均  $\geq b$  发生停时, 问最后期望有多少盒子被填满, 对 998244353 取模。 $1 \leq n \leq 250$ ,  $1 \leq b < a \leq 250$ 。

**解法** 先特判掉  $n = 1$  的情况: 此时答案显然为 0, 无须考虑。

注意到答案即为每个盒子被填满的概率和, 也即  $n$  乘上单个盒子被填满的概率, 因此只需求出单个盒子被填满的概率。

容易发现假如允许继续往已经被填满的盒子里填, 答案是不变的。枚举最后一步选择的是哪个盒子, 就得到概率为

$$\frac{n-1}{n} \left( \mathcal{L} \left( \left( \exp(z/n) - \sum_{j=0}^{a-1} \frac{(z/n)^j}{j!} \right) \left( \exp(z/n) - \sum_{j=0}^{b-1} \frac{(z/n)^j}{j!} \right)^{n-2} \frac{(z/n)^{b-1}}{(b-1)!} \right) \right) \quad (1)$$

设  $t = z/n$ ,  $u = \exp t$ , 那么发现核心就是要快速求出

$$\left( u - \sum_{j=0}^{a-1} \frac{t^j}{j!} \right) \left( u - \sum_{j=0}^{b-1} \frac{t^j}{j!} \right)^{n-2} \frac{t^{b-1}}{(b-1)!}$$

容易使用 ODE 在  $O(n^2 a)$  的复杂度下求出, 假设为  $\sum_{i,j} F_{i,j} t^i u^j$ , 那么容易得到概率即为

$$(n-1) \sum_{i,j} \frac{i! F_{i,j}}{(n-j)^{i+1}}$$

♠

<sup>4</sup>2018 年北京大学全国优秀中学生信息学冬季体验营 (PKUWC2018), <https://loj.ac/p/2541>。

<sup>5</sup>UOJ Round #19, <https://uoj.ac/problem/514>。

## 6 让随机过程不停时

很多时候一个停时的随机过程并不方便直接通过枚举或者简单的符号化方法刻画，这种时候可以考虑接下来一种间接计数的方法。值得一提的是，这也利用了符号化方法。

考虑将停时事件按某种方式划分成互不相交的  $c$  组，第  $i$  组称为  $S_{i,0}, S_{i,1}, \dots$  或者简记为  $\{S_{i,t}\}$ 。

考虑假设令过程  $X$  满足停时事件时不会直接停时，得到一个新的过程  $Y = \{y_t\}$ 。

使得对任意两组  $1 \leq i, j \leq c$ ，存在 PGF

$$G_{i,j}(z) = \sum_{k \geq t} P(S_{j,k}(y_k) | S_{i,t}(y_t)) z^{k-t} \forall t \in \mathbb{N} \wedge S_{i,t} \neq \emptyset$$

设  $X$  关于  $\{S_{i,t}\}$  的 PGF 为  $F_i(z)$ ， $Y$  关于  $\{S_{i,t}\}$  的 PGF 为  $H_i(z)$ 。

容易发现， $F_i$  本质上刻画了  $Y$  在  $t$  时刻发生第一次停时事件时，发生的是  $S_{i,t}$  的概率；而  $H_i$  则刻画了  $Y$  在  $t$  时刻发生停时事件时，发生的是  $S_{i,t}$  的概率。

于是对每个  $1 \leq i \leq c$ ，枚举  $Y$  第一次触发停时事件时的状态，就得到

$$H_i = \sum_{j=1}^c F_j G_{j,i}$$

于是设向量  $\mathbf{f} = (F_1, F_2, \dots, F_c)'$ ， $\mathbf{h} = (H_1, H_2, \dots, H_c)'$ ，并设矩阵  $\mathbf{M} = (G_{i,j})_{1 \leq i, j \leq c}$ ，也就得到

$$\mathbf{M}\mathbf{f} = \mathbf{h}$$

于是只用求出  $\mathbf{h}, \mathbf{M}, \mathbf{M}^{-1}$ （容易知道一定可逆）就可以求出  $\mathbf{f}$  了，问题转化为快速求出  $H_i, G_{i,j}$  和如何对这个矩阵快速求逆，这就需要就题分析了。

### 6.1 $c = 1$ 的情况

事实上通常遇上的题目都是  $c = 1$  的情况。

此时简记  $F = F_1$ ， $G = G_{1,1}$ ， $H = H_1$ ，就有  $F = H/G$ 。

现在如果要求解  $\mathbb{E}[T] = F'(1) = (H/G)'(1)$ ，应该怎么办呢？

事实上绝大多数遇上的情况下都有  $\hat{H} = \sum_j A_j \exp(jz/n)$  及  $\hat{G} = \sum_j B_j \exp(jz/n)$ ，接下来着重讨论这类情况。

根据定理 9，容易发现此时总有

$$H(z) = \sum_j \frac{nA_j}{n-jz}, G(z) = \sum_j \frac{nB_j}{n-jz}$$

于是

$$H'(z) = \sum_j \frac{jnA_j}{(n-jz)^2}, G'(z) = \sum_j \frac{jnB_j}{(n-jz)^2}$$

要求解的即为

$$\frac{H'G - G'H}{G^2}(1)$$

一般有  $B_n \neq 0$ , 将分子分母同乘以  $(1-z)^2$  得答案为  $\sum_{j \neq n} n(A_n B_j - B_n A_j)/B_n^2(n-j)$ 。  
特别地, 当  $A_n = B_n$  时即为  $\sum_{j \neq n} n(B_j - A_j)/B_n(n-j)$ 。

**例题 5.** (开关<sup>6</sup>) 一个  $n$  位二进制数, 初始各位全为 0, 每位有一个权值  $p_i$ 。每一步会独立地以  $p_i/\sum_{j=1}^n p_j$  的概率选择第  $i$  位并翻转。任意第  $i$  位均达到目标状态  $s_i$  后过程停时, 求期望停时对 998244353 取模后的值。  $1 \leq n \leq 100$ ,  $1 \leq \sum_{i=1}^n p_i \leq 5 \times 10^4$ ,  $p_i \in \mathbb{N}^+$ ,  $s_i \in \{0, 1\}$ 。

**解法** 设  $S = \sum_{j=1}^n p_j$ , 则

$$\hat{G}(z) = \prod_{j=1}^n \cosh(p_j z/S), \hat{H}(z) = \prod_{j=1}^n \begin{cases} \cosh(p_j z/S) & s_j = 0 \\ \sinh(p_j z/S) & s_j = 1 \end{cases}$$

这是容易的: 直接考虑每一位的选法序列即可。

乍一看很不好做, 但观察到, 如果设  $u = \exp(z/S)$ , 那么上述等式可以改写为

$$\hat{G} = \prod_{j=1}^n \frac{u^{p_j} + u^{-p_j}}{2}, \hat{H} = \prod_{j=1}^n \frac{u^{p_j} + (-1)^{s_j} u^{-p_j}}{2}$$

容易使用暴力乘法在  $O(nS)$  时间内把  $\hat{G}$  和  $\hat{H}$  表示成关于  $u$  的带负下标的形式幂级数, 且坐标范围在  $[-S, S]$  间, 假设分别为  $\sum_{-S \leq j \leq S} B_j u^j$  和  $\sum_{-S \leq j \leq S} A_j u^j$ 。那么就有答案为

$$2^n S \sum_{-S \leq j < S} \frac{B_j - A_j}{S - j}$$

(容易发现  $A_S = B_S = 1/2^n$ )

♠

**例题 6.** (Flip Cells<sup>7</sup>)  $n$  行  $m$  列的网格, 每个格子初始为黑或白, 第  $i$  行最初有  $b_i$  个黑格。不断独立均匀随机选择一个格子翻转颜色, 直到对所有  $1 \leq i \leq n$ , 第  $i$  行恰有  $a_i$  个黑色格子, 过程停时。求期望停时对 998244353 取模后的值。  $1 \leq n, m \leq 50$ ,  $1 \leq a_i, b_i \leq m$ 。

**解法** 显然有

$$\hat{H}(z) = \prod_{i=1}^n \sum_j \binom{b_i}{j} \binom{m-b_i}{a_i-j} \sinh^{a_i+b_i-2j}(z/nm) \cosh^{m-a_i-b_i+2j}(z/nm)$$

$$\hat{G}(z) = \prod_{i=1}^n \sum_j \binom{a_i}{j} \binom{m-a_i}{a_i-j} \sinh^{2a_i-2j}(z/nm) \cosh^{m-2a_i+2j}(z/nm)$$

考虑怎么求出。由于做法类似故以  $\hat{H}$  为例。

<sup>6</sup>2019 年 CCF NOI 浙江省省队选拔 (ZJOI2019), <https://uoj.ac/problem/469>。

<sup>7</sup>AtCoder Regular Contest 136, [https://atcoder.jp/contests/arc136/tasks/arc136\\_f](https://atcoder.jp/contests/arc136/tasks/arc136_f)。

设  $u = \exp(z/nm)$ , 就得到

$$\hat{H} = \left( \left( \prod_{i=1}^n \sum_j \binom{b_i}{j} \binom{m-b_i}{a_i-j} (u-1)^{a_i+b_i-2j} (u+1)^{m-a_i-b_i+2j} \right) \circ_u u^2 \right) / (2u)^{hw}$$

使用 ODE 在  $O(m^2)$  内预处理出所有  $(u-1)^a(u+1)^{m-a}$ , 即可在  $O(nm^2)$  时间内求出

$$\sum_{j=0}^{nm} A_j u^j = \prod_{i=1}^n \sum_j \binom{b_i}{j} \binom{m-b_i}{a_i-j} (u-1)^{a_i+b_i-2j} (u+1)^{m-a_i-b_i+2j}$$

于是  $\hat{H} = \sum_{j=0}^{nm} A_j u^{2j-nm}$ 。同样可以设  $\hat{G} = \sum_{j=0}^{nm} B_j u^{2j-nm}$ , 那么总有  $A_{nm} = B_{nm} = \prod_j \binom{m}{a_j}$ , 此时答案即为

$$\frac{nm}{2B_{nm}} \sum_{j=0}^{nm-1} \frac{B_j - A_j}{nm-j}$$

**例题 7.** (Finding Expected Value<sup>8</sup>) 现有  $n$  个不大于  $k$  的正整数, 不断独立均匀随机其中一个数独立均匀随机改成一个不大于  $k$  的正整数, 直到所有数相等时过程停时。现在给定初始序列的一部分, 另一部分在可能的序列内独立均匀随机, 求该过程的期望停时的期望, 对  $10^9 + 7$  取模。  $2 \leq n \leq 10^5$ ,  $2 \leq k \leq 10^9$ 。

**解法** 显然有  $\hat{G}(z) = (1 + (\exp(z/n) - 1)/k)^n + (k-1)((\exp(z/n) - 1)/k)^n$ 。

由于期望的线性性以及各部分计算贡献的线性性, 只用求出期望的  $\hat{H}$  即可计算答案。设  $j$  出现了  $a_j$  次, 且有  $c$  个数未知, 容易有

$$\hat{H}(z) = \sum_{j=1}^k (1 + (\exp(z/n) - 1)/k)^{a_j} ((\exp(z/n) - 1)/k)^{n-c-a_j} (1/k + (\exp(z/n) - 1)/k)^c$$

设  $u = \exp(z/n)$ ,  $r_i = \sum_{j=1}^k [a_j = i]$ , 那么就有

$$\hat{H} = \frac{u^c}{k^n} \sum_{i=1}^k r_i (u+k-1)^i (u-1)^{n-c-i}$$

使用 ODE 即可在线性时间内求出  $(u+k-1)^i(u-1)^{n-c-i}$  关于  $u$  展开后的结果, 而满足  $r_i \neq 0$  的  $i$  易知只有  $O(\sqrt{n})$  个, 于是即可在  $O(n\sqrt{n})$  的复杂度内表示出  $\hat{H} = \sum_j A_j u^j/k^n$ 。同样可以表示出  $\hat{G} = \sum_j B_j u^j/k^n$ , 这样由于  $A_n = B_n = k$ , 就有答案为

$$\frac{n}{k} \sum_{j=0}^{n-1} \frac{B_j - A_j}{n-j}$$

<sup>8</sup>COMP FEST 13 Finals, <https://codeforces.com/contest/1575/problem/F>。



不妨先停下来，回看例题 1 的情况。虽然形式不同了，但是这也可以使用这类方法解决吗？

**例题 8.** (歌唱王国) 同例题 1，但是这次要求使用该种方法解决本题。

**解法** 容易发现，使用这次的方法改写，就有

$$H(z) = \frac{z^n}{m^n(1-z)}, G(z) = \frac{z^n}{m^n(1-z)} + \sum_{1 \leq j \leq n \wedge s_1 \sim j = s_{n-j+1} \sim n} \left(\frac{z}{m}\right)^{n-j}$$

于是

$$F(z) = \frac{z^n}{z^n + (1-z) \sum_{1 \leq j \leq n \wedge s_1 \sim j = s_{n-j+1} \sim n} m^j z^{n-j}}$$

设  $f = \sum_{1 \leq j \leq n \wedge s_1 \sim j = s_{n-j+1} \sim n} m^j z^{n-j}$ ，就有

$$F'(z) = \frac{nz^{n-1}(z^n + (1-z)f(z)) - z^n(nz^{n-1} - f(z) + (1-z)f'(z))}{(z^n + (1-z)f)^2}$$

代入 1 后整理形式就发现

$$F'(1) = f(1) = \sum_{1 \leq j \leq n \wedge s_1 \sim j = s_{n-j+1} \sim n} m^j$$

与先前的答案吻合。

诚如《具体数学》中说的一样，一个强大的工具，不仅能解决相对繁杂的问题，一些相对简单的问题也同样可以迎刃而解。这也正是这一方法的优秀之处所在。

## 6.2 $c \geq 1$ 的情况

由于直接引入含 EGF 的情况可能略显突兀，不妨先看一道仅含 OGF 的版本：既然例题 8 这种单个终止串的情况是可做的，那多个终止串的情况也可以这么做吗？

**例题 9.** (硬币游戏<sup>9</sup>) 给定  $n$  个两两不同的长度为  $m$  的 01 串  $s_1, s_2, \dots, s_n$ ，现不断独立均匀随机生成 0 或 1，直到最后生成的  $m$  个字符恰好与  $s_1 \sim s_n$  中任意一个相同，过程停时。求以串  $s_i$  停时的概率，对 998244353 取模（保证答案存在）。 $1 \leq n, m \leq 300$ 。

**解法** 显然规定事件集  $\{S_{i,t}\}$  为以  $i$  结局，各个概率即为  $F_1(1), F_2(1), \dots, F_n(1)$ 。容易有

$$H_i(z) = \frac{z^m}{2^m(1-z)}, G_{i,j}(z) = \frac{z^m}{2^m(1-z)} + \sum_{1 \leq k \leq m \wedge s_{i,m-k+1} \sim m = s_{j,1} \sim k} \left(\frac{z}{2}\right)^{m-k}$$

由于有

$$\sum_i F_i(z) G_{i,j}(z) = H_j(z)$$

<sup>9</sup>2017 年 CCF NOI 山东省省队选拔 (SDOI2017)，<https://loj.ac/p/2004>，有改动。

因此

$$\sum_i F_i(z) \left( 1 - (1-z) \sum_{1 \leq k \leq m \wedge s_{i,m-k+1 \sim m} = s_{j,1 \sim k}} \left(\frac{z}{2}\right)^{-k} \right) = 1$$

于是设  $r_{i,j}(z) = \sum_{1 \leq k \leq m \wedge s_{i,m-k+1 \sim m} = s_{j,1 \sim k}} 2^k z^{-k}$ , 就有

$$\sum_i F_i(z) r_{i,j}(z) = \frac{1 - \sum_i F_i(z)}{z - 1}$$

从而对  $2 \leq j \leq n$  都有

$$\sum_i F_i(z) (r_{i,j}(z) - r_{i,1}(z)) = 0$$

取 1 代入即得

$$\sum_i F_i(1) (r_{i,j}(1) - r_{i,1}(1)) = 0$$

又有

$$\sum_i F_i(1) = 1$$

直接高斯消元即可解出。

使用字符串哈希或者 Z 函数即可求出  $r_{i,j}(1)$ , 时间复杂度  $O(n^2 m + n^3)$ 。

含 EGF 的情况毫无疑问更加复杂, 由于极为罕见, 仅给出一题以供参考。

**例题 10.** (多结局开关<sup>10</sup>) 一个初始为 0 的  $n$  位二进制数, 不断独立均匀随机其中一位翻转。给定  $c$  个两两不同的  $n$  位二进制数  $s_1 \sim s_c$ , 当存在  $1 \leq j \leq c$  满足当前的二进制数恰等于  $s_j$  后过程停时。求期望停时对 998244353 取模后的值。数据保证答案存在; 或者说数据在  $n, c$  确定时关于所有可能生成的数据等概率随机生成, 故不用担心模意义下除 0 时的问題, 以简化考虑。  $1 \leq n \leq 100, 1 \leq c \leq 10$ 。

解法 显然规定事件集  $\{S_{i,t}\}$  为以二进制数  $s_i$  终局。类似于例题 5, 容易写出如下形式

$$G_{i,j}(z) = \frac{1}{2^n} \sum_{k=0}^n \frac{B_{i,j,k}}{1 - (2k/n - 1)z}, H_i(z) = \frac{1}{2^n} \sum_{k=0}^n \frac{A_{i,k}}{1 - (2k/n - 1)z}$$

且总有  $A_{i,n} = B_{i,j,n} = 1$ 。

考虑怎么求  $M^{-1}$ 。设

$$T = 2^n \left( \prod_{k=0}^n (1 - (2k/n - 1)z) \right) M$$

显然  $T$  各项均为次数小于  $n$  的整式, 于是考察其逆矩阵就有

$$|T|T^{-1} = T^*$$

<sup>10</sup>笔者命制的练习题。

其中  $T^*$  为  $T$  的伴随矩阵；显然各项为  $O(nc)$  次的多项式。

使用矩阵多项式求逆即可在  $O(nc^4 \log(nc))$  内求出  $T^{-1}$  的前  $O(nc)$  项，现在难点来到如何求出  $|T|$  进而得到  $T^*$ 。

由于  $|T|$  是  $O(nc)$  次的，不妨考虑插值求行列式，最后再还原多项式，这样就可以在  $O(nc^3 \log^2(nc) + nc^4)$  内求解。还原的过程需要分治卷积算法来快速插值。

于是可以把  $T^{-1}$  表示成  $T^*/|T|$ ，从而

$$M^{-1} = \frac{2^n (\prod_{k=0}^n (1 - (2k/n - 1)z))}{|T|} T^*$$

然后就可以直接计算出  $f$  的分式表达，求导后消去公共的  $z - 1$  因子并代入 1 即可。使用 NTT 总复杂度不会超过  $O(nc^3 \log^2(nc) + nc^4 \log(nc))$ 。消去公共因子的步骤可以通过对  $z - 1$  的复合来实现。♠

事实上这道题目也给出了  $c$  较小时的一个比较通用的解法：对  $M$  通分，然后借助伴随矩阵仍为整式的特点来进行求逆。最后的问题就变成了直接操纵这些多项式，直接暴力操作即可。

只是这样的问题是会给复杂度带来大量  $c$  因子，以及多项式算法的常数。

从这道例题中可以间接看出， $c$  较大的情况其实是比较困难的，因而一般所解决的情况都是  $c = 1$  或者  $c = O(1)$  的：矩阵环本身并不优秀的运算性质限制了进一步的推动。

## 7 这一方法的缺陷

尽管使用 PGF 刻画过程非常强大，可以解决很多常见的停时问题，但当「每一步的概率」不再是一个恒定的值时，这一做法往往会显得很无力：这意味着需要进一步的描述才能完整表示出 PGF。

或者说，随着问题变得复杂，刻画出的 PGF 也会更为复杂，而与此同时一些比较偏「灵光一闪」的做法，譬如使用了鞅的停时定理<sup>11</sup>，反倒显得很简单。

同时，过程也往往可以被直接或间接表达为一个马尔可夫链，否则不能够很方便地描述「推进若干时刻」的概念。

**例题 11.** (Slime and Biscuits<sup>12</sup>)  $n$  个人，第  $i$  个人有  $a_i$  个球。设  $S = \sum a_i$ ，每次会在  $S$  个球中独立均匀随机选择一个并传给任意一个不是当前人的人，直到所有球都归一人所有后过程停时。求期望停时对 998244353 取模后的值。 $a_i \in \mathbb{N}$ ,  $1 \leq n \leq 10^5$ ,  $1 \leq S \leq 3 \times 10^5$ 。  
解法 依旧考虑让过程不终止。

设 GF  $g$  满足  $g' = (\exp z - g)/(n - 1)$ ,  $[z^0]g = 1$ 。

设 GF  $h$  满足  $h' = (\exp z - h)/(n - 1)$ ,  $[z^0]h = 0$ 。

<sup>11</sup> 见参考文献 3。

<sup>12</sup> Codeforces Round 641, <https://codeforces.com/contest/1349/problem/D>。

设  $S = \sum a$ ，显然有

$$\hat{H} = \sum_{i=1}^n g^{a_i}(z/S) h^{S-a_i}(z/S)$$

$$\hat{G} = g^S(z/S) + (n-1)h^S(z/S)$$

然后解微分方程即得

$$g = \frac{\exp z + n \exp(-z/(n-1))}{n}$$

$$h = \frac{\exp z - \exp(-z/(n-1))}{n}$$

于是就可以考虑计算出  $\hat{H}$  与  $\hat{G}$  的表达式，以  $\hat{H}$  为例，有

$$\hat{H} = \frac{\exp(-z/(n-1))}{n^S} \sum_{i=1}^n (\exp(nz/(n-1)S) + n-1)^{a_i} (\exp(nz/(n-1)S) - 1)^{S-a_i}$$

设  $u = \exp(nz/(n-1)S)$ ，容易发现只用求

$$\sum_{i=1}^n (u + n-1)^{a_i} (u-1)^{S-a_i}$$

考虑先复合  $u+1$ ，再复合  $u-1$ ，容易在  $O(n+S \log S)$  内算出其为（不妨设） $\sum_{i=0}^S A_i u^i$ ，也即

$$\hat{H}(z) = \frac{1}{n^S} \sum_{i=0}^S A_i \exp\left(\frac{ni-S}{(n-1)S} z\right)$$

同样设

$$\hat{G}(z) = \frac{1}{n^S} \sum_{i=0}^S B_i \exp\left(\frac{ni-S}{(n-1)S} z\right)$$

由于显然  $A_S = B_S = n$ ，立刻得到答案为

$$\frac{(n-1)S}{n^2} \sum_{i=0}^{S-1} \frac{B_i - A_i}{S-i}$$

总复杂度不会超过  $O(n+S \log S)$ ，复杂度瓶颈在于自变量平移。♠

而同样也是这个问题，使用鞅的停时定理则可以在  $O(n+S)$  的时间内直接解决<sup>13</sup>。

但是这也并不意味着这一方法在这个  $G$  和  $H$  下一定无法线性解决此题，譬如可以这么解决：

**例题 12.** (Slime and Biscuits) 同例题 11，但是要求使用同样的  $G$  和  $H$  在线性时间内解决此题。

<sup>13</sup><https://codeforces.com/blog/entry/77284>。

解法 由于本题中求出  $G$  的贡献不难于  $H$ ，我们现在只考虑  $A$  序列的贡献。核心就是要求出

$$\sum_{i=0}^{S-1} \frac{A_i}{S-i}$$

的值，而  $H$  的各部分贡献也是可以拆解的，所以问题就转化为要在  $O(a)$  时间内求出

$$\sum_{i=0}^{S-1} \frac{[z^i](z+n-1)^a(z-1)^{S-a}}{S-i}$$

稍加转化，即为

$$\sum_j (n-1)^j \binom{a}{j} [z^{S-a+j}](z-1)^{S-a} \ln \frac{1}{1-z}$$

设  $t = S - a$ ，也即在  $O(a)$  时间内对  $j = 0, 1, \dots, a$  求出  $[z^{t+j}](z-1)^t \ln(1/(1-z))$ 。

观察到若设  $f_t = (z-1)^t \ln(1/(1-z))$ ，就有

$$(z-1)f'_t = tf_t - (z-1)^t$$

这一形如 ODE 的形式，并且提取系数后会发现这是只需要上一项的递推，并对任意  $j > 0$  都可以直接求出  $[z^{j+t}]f_t$ 。

于是特殊考虑  $j = 0$  的情况，此时即设为  $w_t = [z^t]f_t$ ，则有

$$\begin{aligned} w_t &= \sum_{i=1}^t \frac{(-1)^i}{i} \binom{t}{i} \\ &= \left( \sum_{i=1}^{t-1} \frac{(-1)^i}{i} \binom{t-1}{i} \right) + \left( \sum_{i=1}^t \frac{(-1)^i}{i} \binom{t-1}{i-1} \right) \\ &= w_{t-1} - \frac{1}{t} \end{aligned}$$

特别地， $w_0 = 0$ 。于是  $w_t = -H_t$ ，其中  $H_n$  为调和数，显然可以预处理。

最后总复杂度即为  $O(n + S)$ 。

同样的技巧也可以施加在例题 7 上，于是例题 7 也可以线性。

这样虽然得到了结果，但缺陷也是无疑的：本做法极为繁杂，计算量远多于使用鞅的停时定理等方法。

此外，同样是使用该类方法，如果设的  $G$  和  $H$  不同，也可以轻松做到线性<sup>14</sup>。

## 8 总结

本文简单阐述了使用生成函数刻画随机过程停时的基本思路，从使用 Laplace 变换来将 EGF 转换成 OGF 来方便统计的技巧引入，指出了一类基于「让随机过程不停时」来间接统计贡献的技巧，能够较系统地刻画常见的随机过程停时问题的信息。

<sup>14</sup><https://codeforces.com/blog/entry/77284#comment-620879>。



同时也给出了这一类做法的缺陷：巧妙的观察往往可以大幅地简化计算，而直接采用这一方法往往需要极大的计算量才能做到同样的复杂度。

还留有的一点缺憾是，对于这一技巧，本文没能对  $c$  比较大的情况给出更加高效的通用解法。希望读者能够积极思考，尝试解决更复杂的情况！

## 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练杨耀良、张艺缤、彭思进的指导。

感谢学军中学徐先友等老师的关心与指导。

感谢家人、朋友对我的支持与鼓励。

感谢信友队的同学、学长给予我帮助与陪伴。

感谢周康阳、沈吉颢同学给我带来的启发。

感谢林卓铖同学为本文验稿并提出建议。

感谢其他给予我帮助的老师与同学。

## 参考文献

- [1] 杨懋龙，《浅谈生成函数在掷骰子问题上的应用》，2018 年信息学奥林匹克中国国家队候选队员论文集。
- [2] 李白天，《信息学竞赛中的生成函数计算理论框架》，2021 年信息学奥林匹克中国国家队候选队员论文集。
- [3] 彭博，《鞅与一类关于停时的概率与期望问题》，可于 <https://www.cnblogs.com/p-b-p-b/p/13297098.html> 下载其中一份版本。





# 浅谈信息学竞赛中的线图问题

绍兴市第一中学 翟锦程

## 摘要

图论是OI中的常考问题，本文简单介绍了图的线图并且展示了部分与之相关问题及解法。

## 1 引言

线图是一类特殊的图，它由原图生成，原图的每条边对应线图的一个点，两个点之间有边当且仅当它们对应的边有公共点。线图在图论中有着重要的地位，它与无爪图、弦图、完美图等图有着紧密的联系，也有着许多实际的应用。本文主要介绍了线图的定义、性质、生成、判定、还原和应用等方面的内容，旨在为读者提供一个线图的概览和参考。本文的主要内容包括：

第2节介绍了一些用到的记号和定义。

第3节介绍了一些线图的基础知识。

第4节通过几道例题，展示了线图知识在信息学竞赛中的一些应用。

## 2 定义与约定

如无特殊说明，本文中的图均指无重边、自环的无向图。

对于图  $G = (V, E)$ ，令  $n = |V|$  表示点集的大小， $m = |E|$  表示边集的大小。为了方便，有时会直接将点从  $1 \dots n$  编号，边从  $1 \dots m$  编号，用  $|G|$  表示  $G$  的点数。

**定义 2.1 (邻域).** 对于任意  $v \in V$ ，记  $v$  的邻域  $N_V(v) = \{u | (u, v) \in E\}$ ，即与  $v$  有边直接相连的点集。

**定义 2.2 (邻边).** 对于任意  $v \in V$ ，记  $v$  的邻边  $N_E(v) = \{(u, v) | (u, v) \in E\}$ ，即与  $v$  相连的边的集合。

**定义 2.3 (连通).** 对于任意  $u, v \in V$ ，称  $u, v$  连通当且仅当  $u, v$  之间存在至少一条路径。称  $G$  是连通的当且仅当  $\forall u, v \in V$ ， $u, v$  是连通的。

**定义 2.4** (导出子图). 对于任意  $A \subseteq V$ , 令  $A$  在  $G$  上的导出子图为  $G' = (A, E')$ , 其中  $E' = \{(u, v) | (u, v) \in E, u, v \in A\}$ , 即点集  $A$  和  $G$  中两端都在点集  $A$  中的边组成的子图。

**定义 2.5** (团). 对于任意集合  $A \subseteq V$ ,  $A$  是一个团当且仅当  $\forall u, v \in A, u \neq v$ , 有  $(u, v) \in E$ , 即点集  $A$  中任意两个不同的点都有边相连。如果不存在  $A' \supset A$  使得  $A'$  是一个团, 则称  $A$  为极大团。

**定义 2.6** (线图). 设  $G = (V, E)$ , 那么  $G$  的线图  $L(G) = (E, \tilde{E})$ , 且  $\forall e_1, e_2 \in E, e_1 \neq e_2, (e_1, e_2) \in \tilde{E} \iff e_1 \cap e_2 \neq \emptyset$ , 即线图上两点相连当且仅当原图上对应两边有公共点。

有一个线图的等价定义:

**定义 2.7** (线图). 设  $G = (V, E)$ , 那么  $G$  的线图  $L(G) = (E, \tilde{E})$  满足对于每个原图上的点  $x$ , 任意  $u, v \in N_E(x), u \neq v$  均在线图中有连边, 即每个点的邻边形成一个团。

## 3 基础知识

### 3.1 性质

根据线图的定义, 若原图  $G$  的性质/概念  $P$  只取决于边的邻接性, 那么  $P$  便可以对应到线图  $L(G)$  上变成性质/概念  $P'$ , 刻画点的邻接性。例如原图中一组匹配 (一组不相交的边), 即对应线图中一组独立集 (一组不相邻的点)。

因此我们可以得到如下性质:

**引理 3.1.** 若原图  $G$  连通, 则其线图  $L(G)$  连通。

证明. 对于任意  $u, v \in E$ , 设其在  $G$  上一条路径  $u - v$  可以对应到  $L(G)$  上一个点序列  $S$ 。  $S$  的相邻两点均有边相连, 因此在线图上亦构成一条路径。因此线图  $L(G)$  中任意两点均连通, 即线图  $L(G)$  连通。  $\square$

**引理 3.2.** 若原图  $G$  存在两条边  $u, v \in E$  不连通, 则其线图  $L(G)$  中  $u, v$  不连通。

证明. 假设  $u, v$  在线图  $L(G)$  中连通, 则一定存在一条路径  $u - v$ , 其经过的点序列形成在原图上形成一条路径, 与原图中两边不连通矛盾。  $\square$

**引理 3.3.** 设  $E' \subseteq E$ , 则生成子图  $G' = (V, E')$  的线图  $L(G')$  为  $E'$  在  $L(G)$  的导出子图。

证明. 由定义 2.5 可以发现, 线图上两点是否有边与原图上其它边无关, 因此一条边被删去不会影响其它边在线图上相连。  $\square$

**引理 3.4.** 原图  $G$  的线图  $L(G)$  中, 若一个点  $u$  是割点, 则其在原图上是连接两个度数不小于 2 的点的割边。

证明. 令  $E' = E - \{u\}$ , 套用**引理 3.3**. 由于  $u$  是原图  $G$  的割边, 因此原图被  $u$  割成两个连通块. 由于割开两点度数不小于 2, 两个连通块中均有边, 于是一定存在两条边不连通, 由**引理 3.2** 得证.  $\square$

**引理 3.5.** 若原图  $G$  中, 第  $i$  个点的度数为  $d_i$ , 则线图  $L(G)$  中边的数量为  $\frac{\sum_{i=1}^n d_i^2}{2} - m$ .

证明. 由**定义 2.6**, 每个原图的点  $x$  的邻边  $N_E(x)$  构成一个团, 这个团的边数为  $\frac{d_x^2 - d_x}{2}$ , 因此  $L(G)$  的总边数为

$$\sum_{i=1}^n \frac{d_i^2 - d_i}{2} = \sum_{i=1}^n \frac{d_i^2}{2} - \sum_{i=1}^n \frac{d_i}{2} = \frac{\sum_{i=1}^n d_i^2}{2} - m$$

 $\square$ 

**引理 3.6.**  $G$  中的一组匹配与  $L(G)$  中的一组独立集一一对应, 且其大小相等.  $G$  中最大匹配的大小等于  $L(G)$  中最大独立集的大小。

证明. 对于任意一组  $G$  的匹配, 由于它们之间不存在公共点, 因此在  $L(G)$  上不存在连边, 即构成一组独立集.  $\square$

**引理 3.7.**  $G$  边染色数等于  $L(G)$  的点染色数, 且染色方案可以对应。

证明. 一组颜色相同的边形成一组匹配, 由**引理 3.6** 对应一组独立集, 即一组颜色相同的点.  $\square$

**引理 3.8.** 若  $G$  是哈密顿图, 则  $L(G)$  也是哈密顿图。

证明. 考虑在  $G$  上构造方案。

维护回路, 初始时为  $G$  的任意哈密顿回路。

接下来每次考虑加入一条未加入回路的边  $(u, v)$ , 找到第一次  $u$  出现的位置  $(x, u) - (u, y)$ , 变为  $(x, u) - (u, v) - (u, y)$ , 由于存在公共点  $u$ , 因此回路在  $L(G)$  上依然合法。

当所有边均加入回路时, 就得到了  $L(G)$  的哈密顿回路.  $\square$

## 3.2 生成

给定原图  $G$ , 如何求出线图  $L(G)$ 。

不难发现, 只要满足原图  $G$  每个点的邻边在其线图  $L(G)$  上构成一个团即可。

因此我们枚举每个点的所有出边, 将这些出边在其线图上连成一个团即可。

### 3.3 判定

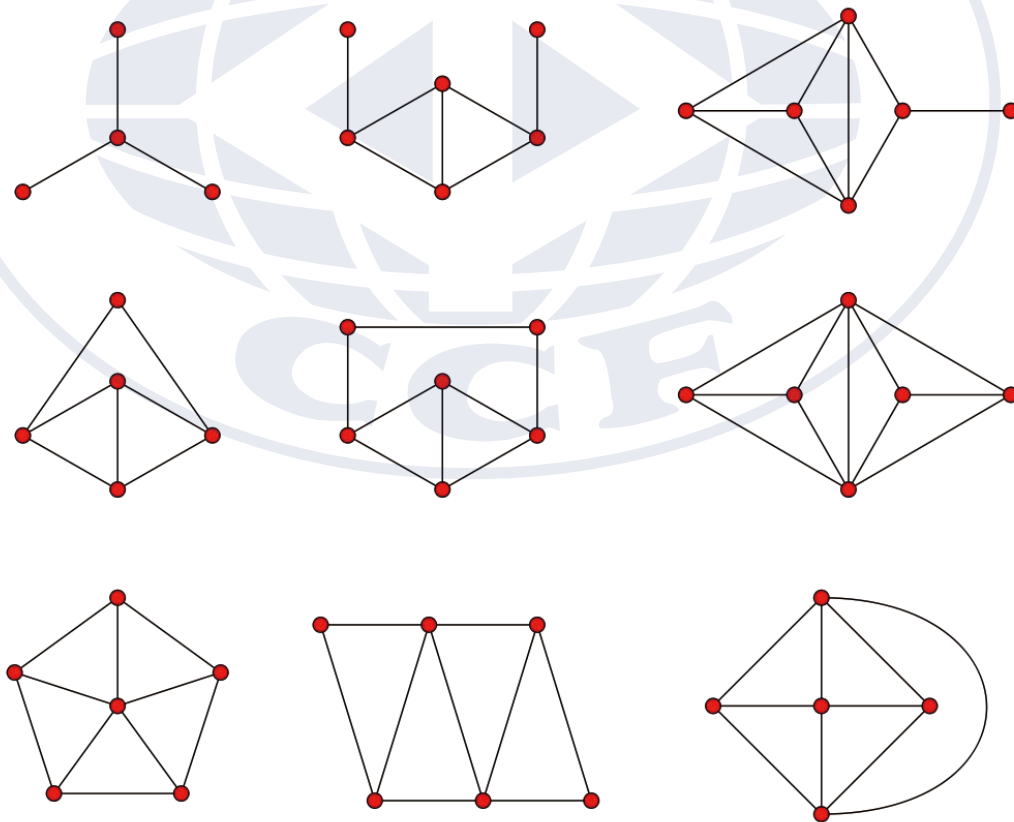
Lowell W. Beikene 于 1970 年证明了线图的四个等价定义 [2]。

首先需要给出奇三角形和偶三角形的定义。

**定义 3.1** (奇三角形). 若三个点  $a, b, c$  两两有连边, 则称有三角形  $abc$ 。若图  $G = (V, E)$  中一个三角形  $abc$ , 存在三角形外点  $e \in V - a, b, c$  满足其仅与三角形  $abc$  中其中一个点有连边或与三个点都有连边, 则称三角形  $abc$  是奇三角形。否则称三角形  $abc$  是偶三角形。

**定理 3.1.** 对于图  $G$  而言, 以下四个命题等价:

1.  $G$  是某个图  $H$  的线图。
2.  $G$  可以被分割为若干完全子图, 并且没有点被分割在超过两个子图中。
3.  $K_{1,3}$  不是图  $G$  的导出子图, 并且如果存在两个奇三角形  $abc$  和  $bcd$ , 则一定存在  $(a, d) \in E$ 。
4.  $G$  中没有导出子图与下图中九张图的任意一张同构 [1]:



根据定理 3.1, 我们可以枚举所有点数不超过 6 的导出子图, 然后判断是否与上图中某张图同构。因此我们得到了一个多项式复杂度判断一个图是否是线图的算法。



事实上，若我们能够将线图还原，那么判断一张图是否是线图只需要判断它是否能还原成原图即可。

### 3.4 还原

美国数学家哈斯勒·惠特尼（1907-1989）于 1932 年在他的论文中提出了惠特尼同构定理 [5]：

**定理 3.2** (惠特尼同构定理). 设有连通图  $G_1$  和  $G_2$  且它们均不是三角形  $K_3$  或爪形  $K_{1,3}$ 。如果  $L(G_1) \cong L(G_2)$ ，那么  $G_1 \cong G_2$ 。也就是说，除了极特殊的情形，图  $G$  的结构可以由线图  $L(G)$  的结构中唯一地恢复出来。

除此之外，我们有一个  $O(n+m)$  时空复杂度内将给定线图  $G$  还原原图  $L^{-1}(G)$  的算法。

我们假设给定线图  $G = (E, \tilde{E})$  是连通的，并且还原的原图  $L^{-1}(G) = (V, E)$  也是连通的。这样除了  $G = K_3$  的情况，还原的原图都是唯一的。

考虑转化问题。由于原图上一条边恰好连接两个点，根据线图的定义，相当于它在线图上恰好在两个团中。若我们能够把线图上每个点染两种颜色，使得颜色相同的点形成一个团，并且覆盖所有的边恰好一次，那么我们就还原了原图——将线图的每个颜色看作点，每个点的两个颜色相当于两个颜色在原图上对应的点连边。

不难发现不能有两个点存在相同的颜色分配，因为这样相当于在原图上出现了重边。

假如我们现在已经确定了一个点  $v \in V$  的邻边  $N_E(v)$  的子集是同一个颜色，即确定了线图  $G$  上一个颜色有哪些点。对于线图  $G$  上任意仅被染色一次的点  $e \in N_E(v)$ ，它的另一个颜色应该包含哪些元素可以直接确定是  $\{x | (e, x) \in N_E(e), x \in E\} - N_E(v)$ ，因为一个点只能有两种颜色，所以所有未被覆盖的边必须在此时被覆盖；而已经被覆盖过的边不能被多次覆盖，所以所有已被覆盖的边必须不再被覆盖。可以证明我们可以重复此过程直到线图  $G$  上所有点均被染色恰好两次，且所有边均被覆盖恰好一次。

证明. 考虑使用反证法。

假设存在点  $e \in E$  没有被染色恰好两次，则说明它被染色了一次或没有被染色。如果它仅被染色了一次，说明依然可以继续染色，与程序结束条件矛盾；如果它没有被染色过，说明存在  $\tilde{e} \in \tilde{E}$  未被覆盖。

若存在边  $\tilde{e} \in \tilde{E}$  没有被覆盖，由于我们染色的过程一直在选择极大团，说明这条边不能被任何一个颜色覆盖，也就是说至少需要三种颜色才能覆盖所有边。而这与每个点只能使用两种颜色矛盾。  $\square$

因此现在的问题是如何确定任意一个点  $v \in V$  的邻边。

考虑线图上一个点  $e \in E$ ，对  $|N_E(e)|$  分类讨论：

当  $|N_E(e)| = 0$  时，显然  $L^{-1}(G) = K_2$ 。

当  $|N_E(e)| = 1$  时, 显然  $e$  有一个独一无二的颜色。

当  $|N_E(e)| = 2$  时, 令  $N_E(e) = \{(e, e_1), (e, e_2)\}$ , 若  $(e_1, e_2) \notin \tilde{E}$ , 则  $e$  的颜色必定一个与  $e_1$  相同, 另一个与  $e_2$  相同。

若上面条件均不满足, 我们找到一条边  $(e_1, e_2) \in \tilde{E}$  满足  $(e, e_1), (e, e_2) \in \tilde{E}$ , 得到集合  $\{e, e_1, e_2\} \cup \{x | (e, x), (e_1, x), (e_2, x) \in \tilde{E}, x \in E\}$ , 将这个集合作为初始染色集合进行尝试; 若尝试不成功, 说明初始染色集合是  $\{e, e_1\} \cup \{x | (e, x), (e_1, x) \in \tilde{E}, (e_2, x) \notin \tilde{E}, x \in E\}$ 。

可以证明, 当  $|N_E(e)| \geq 3$  时, 取任意  $(e, e_1), (e, e_2), (e, e_3) \in N(e)$ , 一定存在  $(e_1, e_2)$ 、 $(e_2, e_3)$ 、 $(e_3, e_1)$  中至少一个, 因此我们只需要取任意三条边即可。

证明. 当  $|N_E(e)| = 0$  时, 由于一条边没有别的边与之相邻, 又由于图连通, 因此只能是单独的一条边。

当  $|N_E(e)| = 1$  时, 由于一个点要染两个颜色却只有一条边, 因此肯定有一个颜色是单独一个点。

当  $|N_E(e)| = 2$  且  $(e_1, e_2) \notin \tilde{E}$  时,  $e_1$  和  $e_2$  不存在颜色相同, 而  $e$  只有两个颜色, 因此  $e$  的颜色必定一个与  $e_1$  相同, 另一个与  $e_2$  相同。

若上面条件均不满足, 我们找到一条边  $(e_1, e_2) \in \tilde{E}$  满足  $(e, e_1), (e, e_2) \in \tilde{E}$ , 假设  $e, e_1, e_2$  有相同颜色, 此时所有与  $e, e_1, e_2$  均有边的点一定都与  $e, e_1, e_2$  的相同颜色相同。假设存在  $e' \in E$  满足颜色与  $e, e_1, e_2$  的相同颜色不相同, 那么  $e'$  需要在两种颜色中与三个点连边, 而由于  $e, e_1, e_2$  之间的边已经覆盖过, 不能重复覆盖, 因此不能再出现在相同颜色中, 所以不可能。另一种情况是  $(e, e_1), (e, e_2), (e_1, e_2)$  分别被三个不同的颜色覆盖, 这种情况下,  $e, e_1, e_2$  的两个颜色都已经分别确定, 因此每个点与其中任意两个点有连边就能被确定三个颜色中的一个。

根据定理 3.1, 图  $G$  不存在导出子图为  $K_{1,3}$ , 因此得证。  $\square$

至此, 我们完成了这个算法。

## 4 应用

**例题 4.1** (Inverse Line Graph[3]). 给定一张  $n$  个点  $m$  条边的简单无向图  $G$  (不保证连通), 你需要构造一张  $n'$  个点  $n$  条边的简单无向图  $H$ , 使得  $L(H) = G$ , 并且  $H$  的边编号与  $G$  的点编号需要一一对应。

$T$  组数据,  $1 \leq T \leq 3 \times 10^5, 1 \leq n \leq \sum n \leq 3 \times 10^5, 0 \leq m \leq \sum m \leq 3 \times 10^6$ 。

你需要最小化  $H$  的点数<sup>1</sup>。

由于本题不保证连通, 因此我们需要对每个连通块单独考虑。

<sup>1</sup>原题中并没有此要求, 仅要求点数不超过  $10^6$ , 笔者对此限制作了加强。



为了最小化  $H$  的点数，我们不应该在  $H$  中加入多余的孤立点。除此之外，我们在对  $G$  中的  $K_3$  进行还原时，应该选择  $K_3$  而不是  $K_{1,3}$ 。其它情况由于定理 3.2 答案是唯一的。

直接套用上文 3.4 节中的算法对每个连通块还原即可。

**例题 4.2** (Line Graph Sequence[6]). 给定一张  $n$  个点  $m$  条边的简单无向图  $G$  (不保证连通)，你需要找到  $L^0(G), L^1(G), L^2(G), \dots, L^{k-1}(G)$  中点数最少是多少，其中  $L^0(G) = G$ ,  $L^t(G) = L(L^{t-1}(G))$ ，其中  $t$  是正整数。

$$1 \leq n \leq 10^5, 0 \leq m \leq \min(\frac{n(n-1)}{2}, 10^5), 1 \leq k \leq 10^5$$

不难发现，如果给定图是一张性质不强的图，它的点数在不断迭代下会很快地增长。

更一般地，有以下引理：

**引理 4.1.** 对于任意图  $G = (V, E)$ ，定义函数  $f(x) = |L^x(G)|$ ，其中  $x \in \mathbf{N}$ ，则  $f(x)$  是一个凸函数。

证明. 要证明  $f(x)$  是一个凸函数，即要证明对于任意  $x \in \mathbf{N}$ ，有  $\delta f(x) \leq \delta f(x+1)$ ，也就是  $f(x+1) - f(x) \leq f(x+2) - f(x+1)$ 。

首先我们证明  $x=0$  时成立。令  $d_v$  表示  $G$  中点  $v \in V$  的度数，则：

$$\begin{aligned} f(0) &= \sum_{v \in V} 1 = n \\ f(1) &= \sum_{v \in V} \frac{d_v}{2} = m \\ f(2) &= \sum_{v \in V} \frac{d_v^2 - d_v}{2} \quad (\text{引理 3.5}) \end{aligned}$$

将  $x=0$  代入上式得：

$$\begin{aligned} f(1) - f(0) &\leq f(2) - f(1) \\ \sum_{v \in V} \frac{d_v}{2} - \sum_{v \in V} 1 &\leq \sum_{v \in V} \frac{d_v^2 - d_v}{2} - \sum_{v \in V} \frac{d_v}{2} \end{aligned}$$

若我们能对任意  $v \in V$  证明：

$$\frac{d_v}{2} - 1 \leq \frac{d_v^2 - d_v}{2} - \frac{d_v}{2}$$

就能证明  $x=0$  时不等式成立。

对上式进行整理：

$$d_v^2 - 3d_v - 2 \geq 0$$

解集为：

$$d_v \in (-\infty, 1] \cup [2, +\infty)$$

由于  $d_v \in \mathbf{N}$ ，不难发现其实无论何时不等式均成立，因此  $x = 0$  时不等式成立。

当  $x > 0$  时，我们可以令  $H = L^x(G)$ ，从而将情况转化为关于图  $H$ ， $x = 0$  的情况。□

因此，我们只需要找到最后一处函数值下降的位置即可。

考虑哪些图的点数会快速增长。

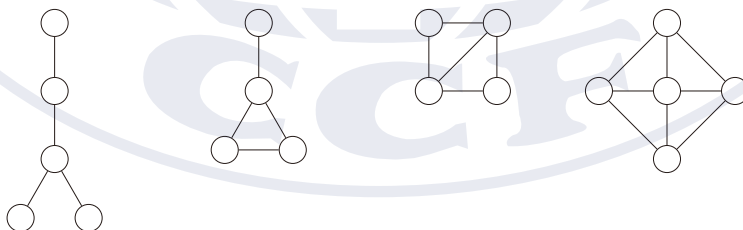
**引理 4.2.** 若图  $G$  存在一个点  $v$  满足其度数  $d_v \geq 4$ 。可以证明关于  $G$  的  $f(x)$  为指数级增长。

证明. 考虑一张图  $G = K_{1,4}$ ，根据引理 3.3，若关于它的  $f(x)$  为指数级增长，就可以证明所有含子图  $K_{1,4}$  的图都是这样。

模拟后发现  $L^2(G)$  的度数不少于 4 的点数为原来的 6 倍，我们把它们当作 6 个独立的  $K_{1,4}$ ， $L^4(G)$  至少有 36 个点度数不少于 4。每两次迭代后，度数不少于 4 的点数至少会变成原来的 6 倍。所以关于  $K_{1,4}$  的  $f(x)$  是指数级增长，从而证明了所有关于含有子图  $K_{1,4}$  的图的  $f(x)$  均为指数级增长。□

**引理 4.3.** 若图  $G$  存在一个点  $v$  满足其度数  $d_v \geq 3$  且不是  $K_{1,3}$ 。可以证明关于  $G$  的  $f(x)$  为指数级增长。

证明. 同上，由于引理 3.3，只需要考虑所有满足条件的图中，不存在其他满足条件的子图的图：



上图为从左到右依次迭代的结果。最左侧两张图即所有满足条件的图中，不存在其他满足条件的子图的图，它们满足边数为 4，并且没有别的边数为 4 的满足条件的图，而边数大于 4 的图一定会出现边数为 4 的子图。不难发现在迭代过程中，上图的最后一张图出现了度数为 4 的点，即出现了子图  $K_{1,4}$ ，由引理 4.2 得证。□

此时度数大于 2 的图只有  $K_{1,3}$  的点数不会指数级增长。而度数不超过 2 的连通块，只有可能是链和环。链的点数会每次减少 1 直到没有，而环的点数不会变化。

因此我们得到了此题的解法：若存在除了  $K_{1,3}$ 、链和环以外的点，则直接模拟找到答案；否则，我们求出每条链的长度，并根据这个求出  $L^{k-1}(G)$  的点数即可。

**例题 4.3** (线图 [4]). 给定一棵  $n$  个点的树  $G$ , 求  $L^k(G)$  的点数对 998244353 取模的结果。

$$1 \leq n \leq 5000, 0 \leq k \leq 9^2.$$

若我们将  $G$  中一个点  $x \in V$  表示成一个集合  $S_x = \{x\}$ , 一条边  $(u, v) \in E$  表示成集合  $S_u \cup S_v$ 。

则我们有引理:

**引理 4.4.**  $L^k(G)$  中的每个点  $v$  有  $|S_v| \leq k+1$ , 若两个点  $u, v$  之间有边, 则有  $|S_u - S_v| \leq 1$ 。

**证明.** 使用数学归纳法。  $k=0$  时根据定义成立。

当  $k>0$  时, 我们已经证明了  $k-1$  时是正确的。

先考虑  $L^{k-1}(G) = (V, E)$  中的边  $(u, v)$ , 有  $|S_u - S_v| \leq 1$ , 因此:

$$S_{(u,v)} = S_u \cup S_v \leq |S_v| + 1 \leq ((k-1)+1) + 1 = k+1$$

即说明  $L^k(G) = (E, \tilde{E})$  中的点  $(u, v)$  满足  $|S_{(u,v)}| \leq k+1$ 。

再考虑  $L^{k-1}(G) = (V, E)$  中的边  $(u, v), (w, v)$ , 有  $|S_u - S_v| \leq 1$ , 因此:

$$|S_{(u,v)} - S_{(w,v)}| = |(S_u \cup S_v) - (S_w \cup S_v)| \leq |(S_u \cup S_v) - S_v| = |S_u - S_v| \leq 1$$

即说明  $L^k(G) = (E, \tilde{E})$  中的边  $((u, v), (w, v))$  满足  $|S_{(u,v)} - S_{(w,v)}| \leq 1$ 。  $\square$

考虑一个点  $u$  的  $S_u$  的实际意义是什么: 在  $L^k(G)$  中,  $u$  的产生与且仅与原图  $G$  中  $S_u$  的点有关。

也就是说,  $L^k(G)$  中的每个点仅与原图中不超过  $k+1$  个点有关。

因此我们可以暴力求出原图  $G$  中所有不超过  $k+1$  个点的连通块, 然后求出它们对  $L^k(G)$  中答案的贡献。

而点数不超过 10 的无标号树是非常少的, 可以暴力对每种树  $T_i$  算出  $L^k(T_i)$  的点数。

要求出  $L^k(T_i)$  的点数, 我们可以求出  $P_i = L^{k-4}(T_i)$ , 然后利用公式计算  $L^4(P_i)$  的点数。

至此我们得到了此题的解法:

1. 对每种树  $T_i$  算出  $L^k(T_i)$  的点数  $d_i$ 。
2. 对每种树  $T_i$  算出它在  $G$  上出现的次数  $t_i$ 。
3. 答案即  $\sum d_i \cdot t_i$ 。

我们通过三个例题, 分别简单介绍了线图的还原、线图迭代点数的增长、线图迭代的点数的算法。

<sup>2</sup>原题中  $k$  以在每个测试点中给定的形式给出, 笔者假设整个题的限制如此。



## 5 总结

本文简单地介绍了线图的定义、性质、生成、判定、还原和应用等方面的内容。但是由于内容篇幅，本文并没有完整地介绍线图的所有内容。感兴趣的读者可以自行参阅相关论文了解。

本文提到了一些线图相关的应用，但在浩瀚的算法海洋中仍只是冰山一角。希望未来能有更多和线图问题相关的算法被发现和发明，也希望有更多有趣的线图问题例如对  $n$  个点  $m$  条边的线图计数、线图的完美匹配等等出现在信息学竞赛中。

## 6 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢绍兴一中的陈合力老师和董烨华老师的关心和指导。

感谢国家集训队教练杨耀良的指导和帮助。

感谢父母对我的关心、支持与无微不至的照顾。

感谢楼宇辰同学、侯之远同学给予我写作本文的启发。

感谢朱羿恺学长、孟煜皓学长、楼宇辰同学、张恒毅同学、侯之远同学为本文验稿并提出建议。

感谢其他对我有过帮助和启发的老师和同学。

## 参考文献

- [1] Line graph - Wikipedia. [https://en.wikipedia.org/wiki/Line\\_graph](https://en.wikipedia.org/wiki/Line_graph).
- [2] Lowell W. Beineke. Characterizations of derived graphs. Journal of Combinatorial Theory, 9(2):129–135, 1970.
- [3] flower、LeafSeek、Qingyu. Inverse Line Graph - QOJ.ac. <https://qoj.ac/problem/4818>, 2022.
- [4] quailty. Line Graph Sequence - QOJ.ac. <https://qoj.ac/problem/7784>, 2023.
- [5] Hassler Whitney. Congruent Graphs and the Connectivity of Graphs. American Journal of Mathematics, 54(1):150–168, 1932.
- [6] ZJOI2018. 线图 - QOJ.ac. <https://qoj.ac/problem/2304>, 2018.



# 浅谈一类树上范围相关问题

长沙市长郡中学 朱屹帆

## 摘要

本文结合树上问题经典算法的分析和改进,并通过引入若干例题,得到一类树上范围相关问题的部分模型的思考和总结。

## 1 前言

树上问题作为一类经典的问题,在信息学竞赛中经常出现,而树上范围相关问题是其中比较重要且具有广泛运用的一类问题。笔者对树链剖分,点分树,基础 Top Tree 进行研究,详细分析其与维护树上范围相关的一些性质,并结合例题进行总结。

本文中在通常情况下定义树的大小为  $O(n)$  级别。

## 2 树链剖分

重链剖分是树链剖分的一种类型,在处理树上范围相关问题中,重链剖分有着重要的作用。重链剖分相关的若干定义可以在 [1] 中找到,我们默认读者已经掌握重链剖分相关的若干定义,接下来分析重链剖分的性质。

### 2.1 重链剖分的性质

对树进行搜索,对于当前节点,优先访问重儿子对应子树,然后访问所有轻儿子对应子树,得到树的其中一个 dfs 序,定义节点在 dfs 序上的位置为该节点的标号。

**性质 2.1.** 从任意节点到根的路径中经过的轻边数量不超过  $O(\log n)$  条。

证明. 将轻边深度按照由深到浅依次考虑,每次经过一条轻边时,说明轻边对应父亲节点存在比当前子树更大的子树作为重儿子,因此父亲节点子树大小至少是轻儿子子树大小的两倍,经过轻边数量不超过  $O(\log n)$  条。□

**性质 2.2.** 树上所有节点的轻儿子子树大小之和至多为  $O(n \log n)$  级别。



证明. 由于任意节点到根的路径中经过的轻边数量不超过  $O(\log n)$  条, 因此每个节点对树上所有节点的轻儿子子树大小之和的贡献至多为  $O(\log n)$ , 因此树上所有节点轻儿子子树大小之和至多为  $O(n \log n)$  级别。□

**性质 2.3.** 对于任意一条重链, 重链底部的节点不存在子节点。

证明. 当一个节点存在子节点时, 其至少存在一个重儿子, 则当一个节点不存在重儿子时, 说明其不存在子节点, 因此重链底部的节点不存在子节点。□

**性质 2.4.** 对于任意一条重链, 将重链上的节点按照深度从小到大排序, 其在  $dfs$  序上的位置构成连续一段。

证明. 由于搜索时优先访问重儿子, 因此当访问到一条重链的顶端节点时, 一定会依次将这条重链上的节点按照深度从小到大加入到  $dfs$  序中。□

**性质 2.5.** 对于任意一个节点, 其子树所有节点在  $dfs$  序上的位置构成连续一段。

证明. 搜索到一个节点后, 会将该节点的子树全部搜索完毕之后再访问子树外的节点, 因此子树所有节点在  $dfs$  序中构成连续一段。□

**性质 2.6.** 对于树上任意一条路径, 其可以拆分成至多  $O(\log n)$  条重链区间, 更具体的来说是至多  $O(\log n)$  条重链前缀以及至多  $O(1)$  条重链区间。

证明. 考虑路径  $(u, v)$  的最近公共祖先  $lca$ , 其在  $lca$  所在的重链上与路径交的部分是一段连续区间, 因此构成  $O(1)$  条重链区间; 而对于其他在路径上的重链, 其与路径交的部分必然包含该重链的顶端节点, 因此构成  $O(\log n)$  条重链前缀。□

## 2.2 重链剖分的一种理解: 树上链分治

我们尝试从类似点分治的角度来理解重链剖分——树上链分治, 点分治可以将树上的一条路径唯一对应到点分治中经过一个分治重心的两条路径, 我们希望用类似的方式将树上的一条路径唯一对应到重链剖分上的一条重链区间上, 将一条路径的唯一对应到的重链区间设置为路径中深度最小的节点的所在重链与路径的交即可。

点分树可以处理一些涉及单点范围信息的维护, 因此我们不妨尝试使用树链剖分来处理一些涉及路径范围信息的维护。

**例题 2.1.** 给定一个  $n$  个节点的树, 树边权均为 1, 每个节点有点权, 初始点权均为 0, 有  $m$  次操作, 需要支持两种操作:

1. 给定路径端点  $u, v$ , 范围  $d$  和权值  $w$ , 将所有满足与路径  $(u, v)$  上的任意节点的最短距离  $\leq d$  的节点点权增加  $w$ 。

2. 给定单点  $u$ , 查询点  $u$  的权值。

数据范围:  $1 \leq n, m \leq 10^5$ 。



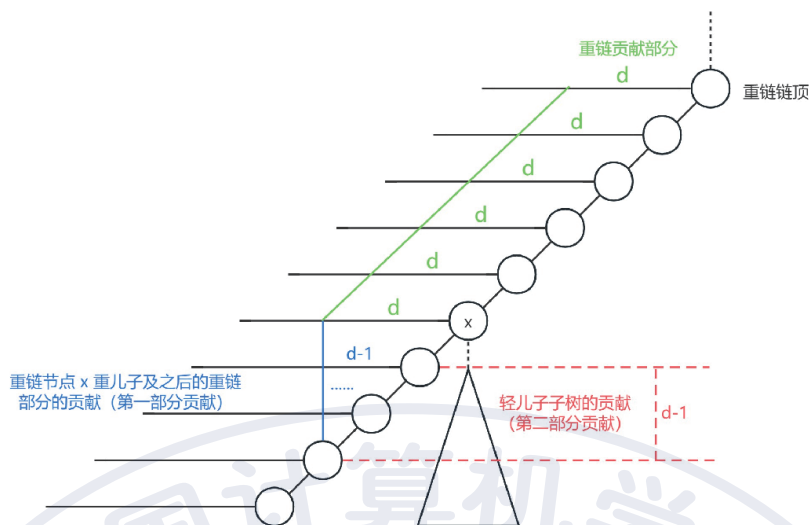


图 1: 重链部分贡献, 第一部分贡献, 第二部分贡献

## 解法

当给定路径端点满足  $u = v$  时, 可以使用点分树维护, 可以做到  $O(n \log^2 n)$ 。

在点分树中, 对于当前分治层上分治重心的标记, 是作用在以分治重心为根的子树上的; 因此考虑在树链剖分结构上进行类似的标记, 即节点  $x$  的标记作用在节点  $x$  的所有轻儿子的子树中, 修改时分析路径在重链剖分上的标记位置及其影响, 查询时对单点本身的标记和单点的祖先上所有轻边对应的父亲节点的标记进行查询。

先考虑直接维护重链部分贡献, 对原树进行重链剖分, 对于一次修改操作, 将路径  $(u, v)$  经过的所有重链区间取出, 并对每一个重链区间进行标记, 形如对于一个在重链区间内的点  $x$ , 其所有轻儿子的子树内与点  $x$  距离在范围  $d$  内的所有节点点权增加  $w$ , 注意受点  $x$  的标记影响的节点为点  $x$  及其轻儿子子树内的所有节点。

但注意到这样会有三个部分的贡献需要处理:

第一个部分是重链区间之后的部分的贡献, 形如与重链区间右端点距离在范围  $d$  内的右端点重子树内的所有节点点权增加  $w$ ;

第二个部分是通过轻边跳到一条新的重链时, 对重链区间上轻儿子子树的操作会影响跳到上来时的轻儿子子树, 原来的轻儿子的贡献需要去掉重复计算部分;

第三个部分是跳到路径深度最小的节点之后其上部分需要添加贡献。

如图所示, 考虑一条重链, 对于一个节点的标记为  $d$ , 可以发现:

重链贡献部分的标记形如重链区间上一段标记为  $d$ ;

第一个部分设重链区间右端点为  $r$ , 形如重链以  $r$  为端点的一段后缀上的点  $x$  的标记



为  $d - \text{dist}(r, x)$  ;

第二个部分注意到重复计算的部分恰好为一个重链的顶部节点的子树内的范围为  $d - 1$  内的节点, 在每条重链顶部维护标记。

第三个部分设路径深度最小的节点为  $lca$ , 形如重链以  $lca$  到该重链的轻边父节点  $y$  为端点的一段前缀上的点  $x$  的标记为  $d - \text{dist}(lca, y) - \text{dist}(x, y)$ 。对每条重链维护区间所有相同标记信息, 区间左端点向右标记递减标记信息, 区间右端点向左标记递减标记信息。

查询单点时, 对单点本身标记和单点祖先上所有轻边对应父亲节点的标记进行查询, 对每条重链顶部的标记进行查询即可。

使用线段树维护每条重链的标记, 时间复杂度  $O(n \log^3 n)$ , 使用全局平衡二叉树维护每条重链的标记, 时间复杂度  $O(n \log^2 n)$ 。

■

## 2.3 基于调整搜索顺序的重标号树链剖分

树链剖分的性质非常丰富且非常广泛, 在一些题目中我们需要维护一些复杂的信息, 树链剖分的其中一些性质非常适合维护这类信息, 但受到其结构的影响会导致一些信息不太方便很好的合并, 此时可以考虑通过改变其结构来保留其中一些有用的性质并更好的维护信息。

**例题 2.2** (数据结构<sup>1</sup>)。给定一棵  $n$  个节点构成的树, 这棵树以点 1 为根, 边的长度均为 1, 每个节点  $x$  有点权  $a_x$ , 初始所有节点的权值为 0。

有  $m$  次操作, 每次操作形如以下四种类型:

1. 给定  $x, y, k, v$ , 对满足节点  $i$  到树上唯一的最短路径  $(x, y)$  的最短距离  $\leq k$  的所有节点  $i$  的权值增加  $v$ , 即  $a_i \leftarrow a_i + v$ 。
2. 给定  $x, v$ , 将点  $x$  的子树内的所有节点  $i$  的权值增加  $v$ , 即  $a_i \leftarrow a_i + v$ 。
3. 给定  $x, y, k$ , 查询所有满足节点  $i$  到树上唯一的最短路径  $(x, y)$  的最短距离  $\leq k$  的所有节点  $i$  的权值的和。
4. 给定  $x$ , 查询点  $x$  的子树内的所有节点  $i$  的权值和。

由于答案会比较大, 输出对  $2^{64}$  取模即可。

数据范围:  $1 \leq n, m \leq 2 \times 10^5$ ,  $1 \leq x, y \leq n$ ,  $1 \leq v \leq 10^9$ ,  $0 \leq k \leq 3$ 。

解法

当修改和查询的路径均为单点时, 本题存在基于对操作序列进行根号重构, 并分析块内的修改对查询的贡献的做法, 可以做到时间复杂度  $O(k(n + m) \sqrt{m})$ 。

事实上这种类似根号重构的做法对于树上范围相关问题也比较有启发意义。

<sup>1</sup>来源: <https://qoj.ac/problem/7767>

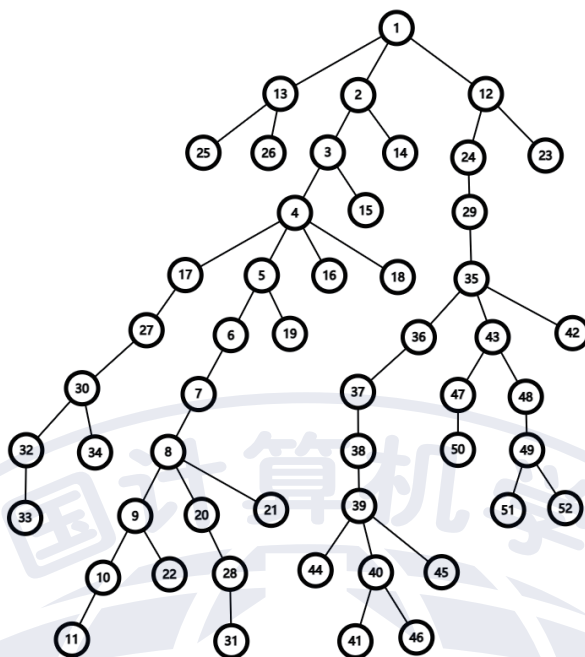


图 2: 进行上述标号方式后, 该树的一种合法的标号

注意到这种根号重构做法的核心思想在于计算操作对询问的贡献，我们不妨尝试换一种思路。

考虑到当  $k = 0$  时，我们可以依据重链剖分的性质，由于路径至多跳  $\log n$  条轻边，因此在  $\log n$  条重链上进行处理，而重链的标号具有连续性，因此可以支持快速对点进行修改和查询，这启发我们尝试在保持重链剖分考虑  $\log n$  条重链进行处理的性质的基础上，通过稍微调整重链的标号，使得重链上与重链上节点距离较小的节点标号大致连续，方便快速修改。

笔者在下文中将这种在保持树链剖分部分性质的基础上，通过调整标号顺序便于维护信息的方式称为“重标号树链剖分”，这里介绍本题的一种标号方式：

按照重链中深度最小的点的 dfs 顺序来处理每一条重链, 考虑对于一条重链, 先将这条重链上的所有点按照深度从小到大排序进行标号, 然后依次枚举距离  $d$  为 1 和 2 和 3, 每次将这条重链上的所有点按照深度从小到大处理每个点, 处理到点  $x$  时将点  $x$  的所有轻儿子子树中的与点  $x$  距离  $= d$  的所有点进行标号。注意到有可能处理一条重链时与重链顶部节点的距离  $\leq 3$  的节点已经被标号, 此时已经被标号的点不会再进行标号。

可以发现这样的结构保留了树链剖分的许多性质, 比如保证了至多在  $\log n$  条重链上进行处理的性质, 并且这样的标号方式使得树上路径的小范围信息具有了一些连续性。



对于每一个节点  $x$ ，考虑维护线段集合  $s_{0_{x,d}}$  表示与点  $x$  距离  $= d$  的在点  $x$  的子树内的所有节点编号区间（当  $d = 4$  时描述的是距离  $> 4$  的点），线段集合  $s_{1_{x,d}}$  表示与点  $x$  距离  $= d$  的在点  $x$  的轻儿子的子树内的所有节点编号区间（当  $d = 4$  时描述的是距离  $> 4$  的点），可以发现转移是简单的。

根据标号的连续性，可以发现  $s_{1_{x,d}} (0 \leq d \leq 4)$  均满足线段数量是  $O(1)$  的，具体来说均满足线段数量  $\leq 2$ ，因为一个节点的与其距离  $\leq 3$  的距离相同的轻儿子一定会在同一时刻被连续标号，但由于可能并不是处理当前节点所在重链时对其的标号（即与其距离  $< 3$  的祖先方向的节点与其并不在一条重链上，并且会将这一层的节点进行标号），因此这个节点的重儿子的编号可能会将连续编号划分成两段，因此线段数量  $\leq 2$ ；而当距离  $\geq 4$  时显然编号是连续的，因此线段数量  $\leq 1$ 。

同时也可以发现  $s_{0_{x,d}} (0 \leq d \leq 4)$  均满足线段数量是  $O(k)$  的，具体来说满足线段数量  $\leq 4$ ，因为一个节点的与其距离  $\leq 3$  的距离相同的所有节点中，对于相同的重儿子来说，每一层一定都是连续的，因此即使每一层之间都是独立的，线段数量仍然满足  $\leq 4$ 。

因此如果我们需要对于单点修改其  $\leq d$  的轻儿子子树的范围内的所有点，则线段总数为  $O(k)$ ，如果我们需要对于单点修改其  $\leq d$  的子树范围内的所有点，则线段总数为  $O(k^2)$ 。

考虑一次链邻域修改操作：考虑每次对同一条重链上的点  $x$  和点  $y$  之间的信息进行处理，首先对重链上的顶部的至多三个节点进行特殊处理，需要对其轻儿子子树信息进行处理；重链的尾部节点需要对其子树信息进行处理；中间的部分可以通过编号的连续性进行处理；重链贡献重复部分对其子树信息进行处理；总共需要处理  $O(\log n)$  条重链，因此该部分时间复杂度为  $O(k^2 \log^2 n)$ ；最后需要对最近公共祖先之上的部分进行处理；因此单次修改总时间复杂度为  $O(k^2 \log^2 n)$ 。

考虑一次链邻域查询操作：可以发现和修改操作是类似的，同样对重链顶部上至多三个节点进行特殊处理；重链尾部节点进行子树处理；中间部分通过编号连续性来处理；重链贡献重复部分进行子树处理；最后对最近公共祖先之上的部分进行处理；因此单次查询总时间复杂度为  $O(k^2 \log^2 n)$ 。

子树修改和子树查询是简单的，由于我们按照 dfs 顺序给每条重链标号，因此我们可以扫描节点  $x$  的子树信息进行处理，单次修改和查询的总时间复杂度为  $O(k^2 \log n)$ 。

该做法可以支持查询链上范围  $\leq k (0 \leq k \leq 3)$  的点权最值和子树的点权最值，做法和查询点权和基本相同。

时间复杂度  $O(mk^2 \log^2 n)$ ，常数较大，可以通过本题。

■

## 2.4 总结

通过树链剖分的树上链分治的角度，我们可以得到许多关于树上范围相关问题的思考和启发。



树链剖分具有非常丰富和广泛的性质，包含重链剖分、长链剖分等多种类型，还可以深入发掘其中更具有拓展性的思想。

### 3 点分树

点分树在处理树上范围相关问题中具有广泛应用。点分树相关内容和基本应用可以在[2]中找到，我们默认读者已经掌握点分树的相关内容和基本应用。

#### 3.1 点分树与树上范围传递性相关问题

**例题 3.1** (地雷<sup>2</sup>)。给定一棵  $n$  个节点的树，边  $i$  连接  $u_i$  和  $v_i$ ，边权为  $w_i$ 。点  $i$  有半径  $r_i$  和点权  $a_i$ ，定义  $\text{dist}(i, j)$  表示点  $i$  和点  $j$  在树上的最短路径长度。当点  $i$  被激活时，所有满足  $\text{dist}(i, j) \leq r_i$  的尚未被激活的点  $j$  将会被激活，反复进行该过程，直到没有节点被激活为止。

对于每一个节点  $i$ ，计算如果初始时激活了点  $i$ ，最终所有被激活的点的点权之和。

数据范围： $1 \leq n \leq 10^5, 1 \leq a_i \leq 10^9, 0 \leq r_i \leq 10^{18}, 1 \leq u_i, v_i \leq n, 1 \leq w_i \leq 10^{12}$ 。

解法

定义点  $i$  直接可达点  $j$  表示  $\text{dist}(i, j) \leq r_i$ ，定义点  $i$  可达点  $j$  表示点  $i$  直接可达点  $j$  或存在一个点  $k$  满足点  $i$  可达点  $k$  且点  $k$  可达点  $j$ 。

注意到点  $i$  直接激活的点在树上点  $i$  的半径为  $r_i$  的范围，建立点分树。定义  $rt$  为当前点分树中的一个子树的根。

设  $f_{rt,x}$  表示初始激活点  $x$  后，最终能给根  $rt$  提供的激活的范围余量，即设点  $y$  最终被激活，则  $f_{rt,x} = \max_y \{r_y - \text{dist}(y, rt)\}$ ；

设  $g_{rt,x}$  表示至少需要初始时  $rt$  提供的激活范围  $g_{rt,x}$  能够使得最终点  $x$  被激活，即存在点  $y$  满足  $\text{dist}(rt, y) \leq g_{rt,x}$  且点  $y$  可达点  $x$ 。

首先考虑给  $f$  和  $g$  添加点分树子树的限制，即在上述定义的基础上限制访问节点的范围为  $rt$  的子树，按深度由深到浅依次处理出  $f$  和  $g$ 。

对于一个根  $rt$ ，考虑逐渐增大其初始提供的激活范围  $r'$ ，由于其子树内的  $f$  和  $g$  更新完毕，因此当点  $x$  被激活时，设  $rt'$  为点分树上  $rt$  子树内且在点  $x$  祖先中的节点，则对于点  $y$  满足  $f_{rt',x} \geq g_{rt',y}$  时，点  $y$  将被激活，按照上述过程，可以处理出当前点分树子树的  $g$  的信息。

对于一个点  $x$ ，我们可以处理出其在  $rt$  对应儿子节点子树的范围内能够提供给  $rt$  的最大激活范围，即设  $rt'$  为点分树上  $rt$  子树内且在点  $x$  祖先中的节点，则对于点  $y$  满足  $f_{rt',x} \geq g_{rt',y}$  时，点  $y$  可以给  $f_{rt,x}$  提供  $r_y - \text{dist}(y, rt)$  的贡献。

<sup>2</sup>来源：<https://qoj.ac/problem/21728>



对  $rt$  的  $f$  和  $g$  排序后维护前缀信息即可转移。

接下来考虑去掉给  $f$  和  $g$  添加的点分树子树范围限制，此时注意到点分树上的根（全局重心）的信息是正确的，因此在点分树上换根处理，按深度由浅到深依次处理  $f$  和  $g$ 。

对于一个根  $rt$ ，其点分树上的祖先的  $f$  和  $g$  是正确信息，考虑如何更新其  $f$  和  $g$  的信息。

对于一个点  $x$ ，考虑其激活后首次激活到子树外的点，激活到一个子树外的点  $y$ ，且  $x$  和  $y$  路径上对应点分树祖先为  $rt'$ ，枚举  $rt'$ ，则当  $f_{rt',x} \geq g_{rt',y}$  时，点  $y$  可以给  $f_{rt,x}$  提供  $r_y - \text{dist}(y, rt)$  的贡献；同样的，对于一个点  $x$ ，考虑其最终被激活前最后一次由子树外的点激活到子树内的点，由一个子树外的点  $y$  激活，且  $x$  和  $y$  路径上对应点分树祖先为  $rt'$ ，枚举  $rt'$ ，则当  $f_{rt',y} \geq g_{rt',x}$  时，点  $y$  可以给  $g_{rt,x}$  提供  $\text{dist}(rt, y)$  的贡献。

按照上述方式进行换根处理，维护出所有  $rt$  的  $f$  和  $g$  的信息，查询时从点  $x$  向点分树上跳父亲节点，去掉点  $x$  所在儿子节点子树后查询经过该重心的点权和，时间复杂度  $O(n \log^2 n)$ 。

■

## 3.2 总结

点分树可以处理许多较为基础的树上范围相关问题。从上述例题中还可以发现，点分树在处理一类树上范围传递性相关问题中具有较强性质。

点分树由于其优秀的树形结构，在处理一类树上范围相关问题时，能够维护许多范围相关的信息，因此具有广泛应用。

## 4 Top Tree

Top Tree 功能非常强大，本文仅介绍较为基础的 Top Tree 与树上范围相关问题的一些应用。我们默认读者已经掌握 Top Tree 的相关定义，下文中簇的端点表示簇的界点。

### 4.1 静态 Top Tree 的一种构建

首先对原树进行重链剖分，并按照重链顶端深度从大到小依次处理每一条重链，当处理完一条重链时我们希望链的顶端有一个边集恰好为该链顶子树内所有边构成的集合，且点集中一端为链顶、另一端无限制的簇。

对于重链上的一个非重链底端节点，其有若干轻儿子，由于我们按照重链顶端深度从大到小依次处理的每一条重链，并且其所有轻儿子均为一条重链的链顶，因此我们可以先使用 compress 将轻儿子的信息与轻边合并，构成若干个其中一端为当前节点的簇，然后使





用 rake 操作将这些轻儿子的信息全部并到这个节点与其重儿子相连的重边上, 轻儿子顺序任意。

此时我们将会得到一条链, 每一条边构成一个当前节点与其所有轻儿子子树的信息并, 显然我们可以使用 compress 操作将其并到一端为重链顶部的节点上, 得到一个满足上述要求的簇, 此时这条重链处理完毕。

注意到上述构建过程中有两次将多个簇进行合并的过程: 第一部分为将若干轻儿子子树与对应轻边的簇的合并, 将其统一 rake 到对应重边上; 第二部分为将若重链上的边与其对应的轻儿子子树信息对应的簇的合并, 将重链上的这些重边 compress, 得到最终簇的两个端点为重链顶端和底端且最终簇的边集恰好为重链顶端节点子树的所有边构成的集合。

我们定义一个簇的权值为其对应边集大小, 采取这样的合并过程: 对一个元素为簇的序列进行分治, 类似建立广义线段树, 找到从左往右首个划分点满足其带权中位数位于左部分, 或者找到从右往左首个划分点满足其带权中位数位于右部分, 使得左右部分均不为空, 分治下去得到左右两部分的合并后的簇, 然后将左右的簇合并即可。

按照上述过程建立的表达式树即为 Top Tree, 构建方法类似树链剖分与全局平衡二叉树的结合。

**性质 4.1.** 按照该方式构建的 Top Tree 满足深度为  $O(\log n)$ 。

证明. 注意到 Top Tree 是一棵二叉树, 我们不妨对于任意节点, 证明其在 Top Tree 树上至多跳  $O(\log n)$  条边可以到达根。

任意一条边代表的簇在 Top Tree 一定是一个叶子节点, 这条边对应的儿子节点跳到根的过程中至多跨过  $O(\log n)$  条轻边; 此时将两个部分的广义线段树状物的合并过程放在一起分析, 我们可以证明的是, 在广义线段树上, 从一个非叶子节点开始跳到广义线段树的根上时, 连续跳两层时, 其所在的区域的权值至少会是原来的两倍。

如图所示, 将权值视作一段连续的长度, 则中位数为其对应区域内中间的点, 不妨假设我们找到从左往右首个划分点满足其带权中位数位于左部分, 即划分点为  $p$ , 并定义区域内的带权中位数位置为  $m$ , 此时区域  $[l, r]$  将被划分成  $[l, p]$  和  $[p+1, r]$  两个部分;

对于右部分, 此时当其从右部分子树跳上来时, 其所在区域的权值显然会至少是原来的两倍。

对于左部分, 不妨设此时该区域内的划分点为  $p'$ , 则容易发现  $p' \leq m$ , 否则由于  $p'$  在  $m$  的右侧, 在划分区域  $[l, r]$  时我们就会选择以  $p'$  作为划分点。

此时区域  $[l, p']$  的权值在连续跳两层时至少是原来的两倍, 而如果区域  $[p', p]$  权值严格大于区域  $[l, r]$  内的权值的一半时, 可以发现此时区域  $[p', p]$  是一个叶子节点。

因此由于一个点跳到根时其至多经过  $O(\log n)$  次轻儿子子树合并和  $O(\log n)$  次重链上的重边合并, 因此其至多作为  $O(\log n)$  次叶子节点, 否则其在当前层的广义线段树中每连续跳两层, 对应权值会至少是原来的两倍, 因此至多跳  $O(\log n)$  层。

因此我们证明了 Top Tree 在按照上述类似全局平衡二叉树的方式构建时其深度为  $O(\log n)$  级别。

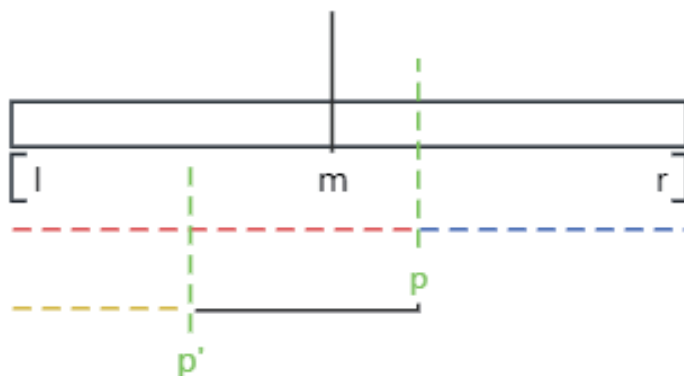


图 3: 带权分治合并过程中的划分结构

□

## 4.2 Top Tree 与树上范围相关的应用

**例题 4.1** (树上邻域数点<sup>3</sup>). 给定一棵  $n$  个节点的树, 你可以使用一些 compress 和 rake 操作进行簇运算, 可以进行不超过  $M$  次簇运算的预处理, 有  $m$  次询问, 每次询问给出节点  $u$  和范围  $d$ , 要求使用至多 1 次簇运算, 得到一个点集没有限制, 边集恰好为节点  $u$  的范围为  $d$  的所有边构成的集合的合法的簇。

数据范围:  $1 \leq n \leq 2 \times 10^5$ ,  $1 \leq m \leq 10^6$ , 预处理的簇运算次数限制  $M = 3 \times 10^7$ 。

解法

首先构建原树的 Top Tree, 考虑当节点  $u$  为原树的根时的做法, 我们在每个簇上对该簇的两个端点, 都分别维护出簇的边集中与端点距离为  $d$  的合法簇信息, 由于我们只维护簇内的信息, 因此需要维护的  $d$  的范围上界为该簇的端点对应的最长路径长度, 方便起见我们使用每个簇的大小作为  $d$  的上界。

设簇类型  $f_{x,0/1,d}$  表示簇内距离簇  $x$  的左端点或右端点距离  $\leq d$  的簇的信息, 要求信息的一端为对应左右端点, 且当  $d \geq$  簇的左右端点距离时信息两端点分别为该簇的左右端点, 发现 compress 和 rake 类型节点可以进行来自左右儿子的信息合并, 因此可以不使用簇运算回答每次关于根的询问。

当询问点为原树任意点  $x$  时, 我们选择该点在原树上的任意一条与之相连的边, 找到一个满足条件的簇使得该簇是该边在 Top Tree 对应节点的祖先且点  $x$  的范围为  $d$  可以完全包含该簇信息, 要求该簇在 Top Tree 上的深度最小, 此时答案为点  $x$  到该簇的两个端点再向该范围外进行拓展的簇的信息合并。

<sup>3</sup>来源: <https://uoj.ac/problem/766>



设簇类型  $g_{x,0/1,d}$  表示簇外距离簇  $x$  的左端点或右端点距离  $\leq d$  的簇的信息，其中  $d$  的上界为簇  $x$  在 Top Tree 上父节点的大小，当簇  $x$  不为 Top Tree 的根时，父亲节点大小  $\geq d$ ，因此可以使用类似换根 dp 的方式通过簇自身的  $f$  和父亲节点对应簇的  $g$  维护出簇自身的  $g$ ，然后通过一个簇的  $f$  和  $g$  的信息即可在至多 1 次簇运算内快速回答询问。

按照上述方式预处理，由于该 Top Tree 深度为  $O(\log n)$ ，即节点子树大小的和的级别为  $O(n \log n)$ ，因此预处理簇运算次数为  $O(n \log n)$ ，可以通过本题。

■

### 4.3 总结

Top Tree 功能强大，从上述例题中可以发现 Top Tree 在处理相关问题时的特殊的结构和特殊的信息合并方式，进一步提升了对于树上范围相关问题的处理能力。

## 5 总结

本文介绍了树链剖分和基础 Top Tree 在树上范围相关问题中的运用，事实上很多树上问题经典算法在树上范围相关问题中具有广泛运用的潜力，笔者希望本文能够激发读者思考，启发读者对树上范围相关问题进行更加深入的研究。

## 致谢

感谢中国计算机学会提供学习和交流的平台。  
感谢长沙市长郡中学谭献龙老师的关心和指导。  
感谢父母对我的支持和鼓励。  
感谢国家集训队杨耀良教练、彭思进教练的指导。  
感谢肖岱恩同学对本文的帮助。

## 参考文献

- [1] OI-wiki 贡献者. 树链剖分 — OI-wiki, 2023.
- [2] OI-wiki 贡献者. 点分树 — OI-wiki, 2023.
- [3] 朱羿恺. 浅谈一类树上统计相关问题. IOI2023 中国国家候选队论文集, 2023.
- [4] 程思元. 浅谈静态 top tree 在树和广义串并联图上的应用. IOI2023 中国国家候选队论文集, 2023.



# 《世界沉睡童话》命题报告

中国人民大学附属中学 叶李蹊

本文介绍了作者给 IOI2024 国家集训队互测命制的一道题《世界沉睡童话》。

## 1 题目

本题加强自 [BalticOI 2016 Day2] C. Swaps<sup>1</sup>。

### 1.1 题目大意

给定一棵大小为  $n$  的有标号有根树  $T$ ，其中结点  $Rt$  是  $T$  的根。

设最初结点  $u$  的点权是  $a_u$ ，保证  $a_1 \sim a_n$  构成一个  $1 \sim n$  的排列。

注意一个结点的儿子之间是有顺序的。设结点  $u$  有  $k_u$  个儿子，它们从左到右依次称为  $ch(u, 1), ch(u, 2), \dots, ch(u, k_u)$ 。

由于结点的儿子之间有顺序，以  $Rt$  为根 DFS 这棵树将得到唯一一个 DFS 序列  $s$ ，其中结点  $s_i$  第  $i$  个被访问到。

接下来进行  $n-1$  次删边操作，第  $i$  次删边操作删去  $(s_{i+1}, Fa(s_{i+1}))$ 。在此之前，你需要选择是否交换  $a_{s_{i+1}}$  与  $a_{Fa(s_{i+1})}$ 。

当所有边尽数被删去以后，设最终结点  $u$  的点权是  $a'_u$ 。定义  $S$  表示所有可能的排列  $a'_1 \sim a'_n$  构成的集合。

最后给定一个  $1 \sim n$  的排列  $p$ ，你需要在下列三个小问中选择一个回答：

1. 判定  $p$  是否  $\in S$ ；
2. 求出  $S$  中字典序大于  $p$  的最小排列  $q$ ；
3. 求出  $S$  中字典序小于  $p$  的排列个数，对 998244353 取模。

### 1.2 数据范围

以下是各个子任务的具体情况以及特殊性质：

<sup>1</sup><https://loj.ac/p/2785>

| 子任务编号 | 分值 | $n =$             | 树的形态  | 特殊性质             | 子任务依赖      |
|-------|----|-------------------|-------|------------------|------------|
| 1     | 10 | 20                |       |                  |            |
| 2     | 10 | $8 \times 10^4$   | 完全二叉树 | DFS 序 $1 \sim n$ |            |
| 3     | 10 | $8 \times 10^4$   | 完全二叉树 |                  | 2          |
| 4     | 10 | $8 \times 10^4$   | 二叉树   | DFS 序 $1 \sim n$ | 2          |
| 5     | 10 | $8 \times 10^4$   | 二叉树   |                  | $2 \sim 4$ |
| 6     | 10 | $8 \times 10^4$   |       | DFS 序 $1 \sim n$ | 2,4        |
| 7     | 10 | $8 \times 10^4$   |       |                  | $1 \sim 6$ |
| 8     | 10 | $1.2 \times 10^5$ |       |                  | $1 \sim 7$ |
| 9     | 10 | $1.6 \times 10^5$ |       |                  | $1 \sim 8$ |
| 10    | 10 | $2 \times 10^5$   |       |                  | $1 \sim 9$ |

- 二叉树：保证  $T$  中每个结点的儿子个数都  $\leq 2$ 。
- 完全二叉树：保证  $T$  与一棵大小同样为  $n$  的完全二叉树同构，并且  $Rt$  对应完全二叉树的根。
- DFS 序  $1 \sim n$ ：按照题目描述中所述方法对  $T$  进行 DFS，第  $i$  个被访问到的结点一定是标号为  $i$  的结点，即  $s_i = i$ 。

### 1.3 评分标准

在一个测试点中：

- 如果你正确回答了第 1 小问，将获得该测试点满分 10% 的分数；
- 如果你正确回答了第 2 小问，无论第 1 小问是否回答、回答正确与否，都将获得该测试点满分 70% 的分数；
- 如果你正确回答了第 3 小问，无论第 1,2 小问是否回答、回答正确与否，都将获得该测试点满分 100% 的分数。

## 2 第一小问的 $O(n)$ 算法

### 2.1 定义与记号

- 记  $b_x$  表示初始时点权为  $x$  的结点的标号，则  $b$  与  $a$  互为逆排列。

- 再记  $t_i$  表示标号为  $i$  的结点在 DFS 的过程中第几个被访问到, 则  $t$  与  $s$  也互为逆排列。

**定义 2.1** (染色方案). 假如一条边在被删去之前交换了两端点的点权, 我们把这条边染黑; 否则把这条边染白。我们记  $c_u$  表示  $(u, Fa(u))$  这条边的颜色:  $c_u = 1$  表示黑色,  $c_u = 0$  表示白色。

- 做出  $n-1$  次选择一共有  $2^{n-1}$  种方案, 一一对应  $2^{n-1}$  种不同的染色方案。我们用染色方案代指选择方案。
- 根据一种染色方案交换点权, 称最终点权构成的排列  $p$  为该种染色方案导出的结果。

## 2.2 不同染色方案导出的结果一定不同

**结论 2.1.**  $|S| = 2^{n-1}$ 。即任意两种不同的染色方案导出的结果也一定不同。

证明. 两种染色方案是不同的, 说明至少存在一条边  $(u, Fa(u))$ , 在第一种方案中  $c_u = 0$ , 在第二种方案中  $c_u = 1$ 。

连接  $u$  子树和外部的途径只有  $(u, Fa(u))$  一条。我们将  $u$  子树内结点的初始点权标为 0,  $u$  子树外的标为 1。

如果交换  $(u, Fa(u))$ , 最终  $u$  子树内就会恰有一个 1; 否则就会全部是 0。因此交换  $(u, Fa(u))$  与不交换  $(u, Fa(u))$ , 最终点权肯定不会完全相同。  $\square$

## 2.3 第一小问的做法

我们可以由此得出第 1 小问的做法: 给定一个排列  $p$ , 我们先 DFS 一遍, 对每个结点  $u$  求出  $u$  子树内所有  $t_{b_{p_i}}$  的最大值、最小值。

若最小值  $= t_u$  且最大值  $= t_u + \text{siz}(u) - 1$ , 则说明  $(u, Fa(u))$  必须染白; 否则说明  $(u, Fa(u))$  必须染黑。

得到每条边的颜色之后, 我们再检查一遍交换过后的结果是否确实  $= p$  即可。时间复杂度  $O(n)$ , 可以获得 10 分。

**定义 2.2** (部分染色方案). 在一个部分染色方案当中, 一条边可以被确定染白, 也可以被确定染黑, 或者尚未确定被染成什么颜色。对于第三种情况, 我们记  $c_u = ?$ 。

**定义 2.3** (自由度). 定义一个部分染色方案的自由度等于未染色的边的总个数  $cur$ , 那么将该部分染色方案完善成完整的染色方案的方式共有  $2^{cur}$  种, 且这  $2^{cur}$  种完整的染色方案导出的结果一定两两不同。



### 3 点权的运输和追溯

#### 3.1 点权的运输

固定一个染色方案，考虑交换点权的过程。

**定义 3.1** (始发地和目的地). 对于同一个权值  $x$ ，如果最初结点  $u$  拥有点权  $x$ ，最终结点  $v$  拥有点权  $x$ ，我们称权值  $x$  从  $u$  运输到了  $v$ ，称  $v$  是  $u$  的目的地， $u$  是  $v$  的始发地，记  $\text{destination}(u) = v$ ， $\text{origin}(v) = u$ 。

**定义 3.2** (运输路线). 在交换点权的过程中， $x$  一定依次经过从  $u$  到  $v$  的树链上所有的结点各一次，我们称树链  $u \rightsquigarrow v$  是权值  $x$  的运输路线。

设树链  $u \rightsquigarrow v$  上的结点依次为  $h_1 \sim h_m$ 。拿出其中一个  $h_i$  来看：

- $h_i$  的  $k_{h_i} + 1$  条邻边被删去的顺序为：  
 $(h_i, \text{Fa}(h_i)), (h_i, \text{ch}(h_i, 1)), (h_i, \text{ch}(h_i, 2)), \dots, (h_i, \text{ch}(h_i, k_{h_i}))$ 。
- $h_i$  作为一个枢纽结点，起到中转权值的作用：在权值  $x$  首次到达结点  $h_i$  之后，找到  $h_i$  的邻边中第一条尚未被删去的黑边，那么  $x$  将被运往这条边的另一个端点。

#### 3.2 运输路线的正序描述

钦定  $\text{destination}(u) = v$ ，以下是  $\text{destination}(u) = v$  对染色方案的限制：

- 所有  $(h_i, h_{i+1})$  必须染黑；
- 对于  $h_1$ ， $(h_1, h_2)$  是  $h_1$  的邻边中第一条黑边；
- 对于  $h_m$ ， $(h_{m-1}, h_m)$  是  $h_m$  的邻边中最后一条黑边；
- 对于中间的  $h_i$ ， $(h_{i-1}, h_i)$  和  $(h_i, h_{i+1})$  是  $h_i$  的邻边中相邻两条黑边，其中  $(h_{i-1}, h_i)$  在前， $(h_i, h_{i+1})$  在后。

例外：当  $u = v$  时要求  $u$  的所有邻边染白。

#### 3.3 运输路线的倒序描述

钦定  $\text{origin}(u) = v$ ，以下是  $\text{origin}(u) = v$  对染色方案的限制：

- 所有  $(h_i, h_{i+1})$  必须染黑；

- 对于  $h_1$ ,  $(h_1, h_2)$  是  $h_1$  的邻边中最后一条黑边;
- 对于  $h_m$ ,  $(h_{m-1}, h_m)$  是  $h_m$  的邻边中第一条黑边;
- 对于中间的  $h_i$ ,  $(h_{i-1}, h_i)$  和  $(h_i, h_{i+1})$  是  $h_i$  的邻边中相邻两条黑边, 其中  $(h_i, h_{i+1})$  在前,  $(h_{i-1}, h_i)$  在后。

例外: 当  $u = v$  时要求  $u$  的所有邻边染白。

**结论 3.1.** 钦定  $origin(u) = v$  对染色方案的限制是两个集合  $E_0, E_1$ , 要求  $E_0$  中的边全部染白,  $E_1$  中的边全部染黑。 $E_0, E_1$  的限制既是充分条件, 也是必要条件。

证明. 假如  $E_0, E_1$  中的限制全部满足, 可以确保  $a_u$  的运输路线是  $u \rightsquigarrow v$ .  $\square$

### 3.4 追溯过程的描述

固定一个染色方案, 我们追溯出  $origin(u)$  的值。

我们知道  $(h_i, h_{i+1})$  是  $h_i$  的邻边中位于  $(h_{i-1}, h_i)$  之前的第一条黑边。利用这一点, 我们可以从  $h_1 = u$  开始, 顺藤摸瓜追溯出  $h_1 \sim h_m$ 。

注意到如果  $h_{i+1}$  是  $h_i$  的儿子, 那么追溯到  $h_{i+1}$  就结束了! 因此在  $h_m$  之前,  $h_{i+1}$  一直是  $h_i$  的父亲。

经过整理, 我们得到以下描述:

- 倒序访问  $u$  到其所有儿子的连边, 然后访问  $(u, Fa(u))$ ;
- 倒序访问  $Fa(u)$  到其所有位于  $u$  左侧儿子的连边, 然后访问  $(Fa(u), anc(u, 2))$ ;
- 倒序访问  $anc(u, 2)$  到其所有位于  $Fa(u)$  左侧儿子的连边, 然后访问  $(anc(u, 2), anc(u, 3))$ ;

以此类推。追溯过程结束的条件有两个:

1. 访问一个结点到其儿子的连边  $(u, v)$  时, 如果  $c_v = 1$ , 则追溯过程在  $v$  结束;
2. 访问一个结点到其父亲的连边  $(u, Fa(u))$  时, 如果  $c_u = 0$ , 则追溯过程在  $u$  结束。

称第一种边的预期颜色是白色, 第二种边的预期颜色是黑色。

设将要访问的边依次为  $e_1 \sim e_m$ , 它们的预期颜色分别为  $c_{e_1}^* \sim c_{e_m}^*$ 。访问到第一条  $c_{e_i} \neq c_{e_i}^*$  的边时, 追溯过程结束,  $origin(u)$  是这条边靠下的端点。

由于  $(Rt, Fa(Rt))$  一定是白边, 不符合预期颜色黑色, 故而追溯过程一定会结束。

**定义 3.3 (追溯过程).** 总结上述的讨论, 我们定义针对  $origin(u)$  的追溯过程是一个二元组序列  $(e_1, c_{e_1}^*) \sim (e_m, c_{e_m}^*)$ , 按照访问的顺序依次表示每条边靠下的端点以及每条边的预期颜色。



## 4 第二、三小问的 $O(n^2)$ 算法

### 4.1 逐步确定染色方案

固定  $p_u = x$  意味着钦定  $\text{origin}(u) = b_x$ 。

我们维护一个部分染色方案，初始时所有  $c_u = ?$ 。随着对排列  $p$  的扫描，逐步确定染色方案。

我们需要提供以下接口：

- 操作  $\text{fix}(u, c)$  表示将  $(u, \text{Fa}(u))$  染成颜色  $c$ ；
- 操作  $\text{assign}(u, x)$  表示固定  $p_u = x$ ，即钦定  $\text{origin}(u) = b_x$ 。  
 $\text{assign}(u, x)$  可能与现有的部分染色方案矛盾。假如  $\text{assign}(u, x)$  希望将确定的白边染黑，或者将确定的黑边染白，那么  $\text{assign}(u, x)$  就是矛盾的。  
 对于没有矛盾的  $\text{assign}(u, x)$ ，记  $\text{reduce}(u, x)$  表示  $\text{assign}(u, x)$  将会确定的无色边个数，即自由度的减小量。
- 记  $\text{valid}(u)$  表示当前执行  $\text{assign}(u, x)$  不会导致矛盾的  $x$  构成的集合。

### 4.2 第二小问的做法

首先求出  $p$  和  $q$  不相等的最高位是哪一位，我们希望这个最高位越靠后越好。

从 1 到  $n$  依次考虑每个结点。考虑到  $u$  的时候：

1. 如果  $\max \text{valid}(u) > p_u$ ，说明  $p_u$  可以  $\neq q_u$ ；
2. 执行  $\text{assign}(u, p_u)$ 。假如  $\text{assign}(u, p_u)$  矛盾，立刻跳出循环。

记  $\lambda$  表示最大的  $u$  使得  $p_u$  可以  $\neq q_u$ （假如  $\lambda$  不存在则输出  $-1$ ）。重置所有  $c_u = ?$ ，再次从 1 到  $n$  依次考虑每个结点。考虑到  $u$  的时候：

1. 对于  $u < \lambda$ ，直接执行  $\text{assign}(u, p_u)$ ；
2. 对于  $u = \lambda$ ，求出  $\text{valid}(u)$  中  $> p_u$  的最小元素  $x$ ，然后执行  $\text{assign}(u, x)$ ；
3. 对于  $u > \lambda$ ，记  $x = \min \text{valid}(u)$ ，执行  $\text{assign}(u, x)$ 。

唯一  $u = \lambda$  的那一步可以直接暴力。因此后续数据结构优化需要解决的问题有三个：

1. 执行  $\text{assign}(u, x)$ ；
2. 求出  $x = \min \text{valid}(u)$ ；
3. 求出  $x = \max \text{valid}(u)$ 。

### 4.3 第三小问的做法

维护当前自由度  $\text{cur}$  和累计答案  $\text{ans}$ ，初始时  $\text{cur} = n - 1$ ， $\text{ans} = 0$ 。

从 1 到  $n$  依次考虑每个结点。考虑到  $u$  的时候：

1. 对于  $\text{valid}(u)$  中每个  $< p_u$  的  $x$ ，将  $2^{\text{cur}-\text{reduce}(u,x)}$  计入答案；
2. 执行  $\text{assign}(u, p_u)$ 。假如  $\text{assign}(u, p_u)$  矛盾，立刻跳出循环。

对于第三小问，后续数据结构优化需要解决的问题有两个：

1. 执行  $\text{assign}(u, x)$ ；
2. 求出  $\text{contribution}(u) = \sum_{x \in \text{valid}(u)} [x < p_u] 2^{\text{cur}-\text{reduce}(u,x)}$ 。

### 4.4 上述做法的暴力实现

针对  $\text{origin}(u)$  进行追溯。设访问到的边靠下的端点依次为  $e_1 \sim e_m$ ，它们的预期颜色分别为  $c_{e_1}^* \sim c_{e_m}^*$ 。

- 如果  $c_{e_i} \neq ?$  并且  $c_{e_i} = c_{e_i}^*$ ，那么称  $e_i$  是绿色的；
- 如果  $c_{e_i} \neq ?$  并且  $c_{e_i} \neq c_{e_i}^*$ ，那么称  $e_i$  是红色的；
- 如果  $c_{e_i} = ?$ ，那么称  $e_i$  是未染色的；

暴力找出  $e_1 \sim e_m$ 。如果追溯过程在  $e_i$  结束，说明  $\text{origin}(u) = e_i$ ，即  $a'_u = a_{e_i}$ 。

- $\text{assign}(u, x)$  设  $a_{e_i} = x$ 。如果这样的  $i$  不存在则  $\text{assign}(u, x)$  是矛盾的。否则将  $e_i$  染成红色，并将  $e_1 \sim e_{i-1}$  染成绿色。
- $\text{reduce}(u, x)$  等于  $e_1 \sim e_i$  这  $i$  条边中未染色的个数。
- $\text{valid}(u)$  找到第一条红色边  $e_i$ 。那么  $\text{valid}(u)$  等于  $e_1 \sim e_i$  中所有未染色边构成的集合，再并上  $e_i$ 。

这里染红一条边指的是通过  $\text{fix}$  操作将其染成不等于预期颜色的那种黑白颜色。染绿指的是染成预期颜色。

对于完全二叉树的特殊情形，时间复杂度  $O(n \log n)$ ，可以获得 30 分。

## 4.5 DFS 序 $1 \sim n$ 的情况

对于 DFS 序  $1 \sim n$  的特殊情况，我们可以随着  $1 \sim n$  的扫描，实时维护针对  $\text{origin}(u)$  进行追溯的过程。

**定义 4.1** (追溯过程的差异度). 我们认为针对  $\text{origin}(u)$  的追溯过程是一个二元组序列  $(e_1, c_{e_1}^*) \sim (e_m, c_{e_m}^*)$ 。定义针对  $\text{origin}(u)$  的追溯过程  $A$  和针对  $\text{origin}(u+1)$  的追溯过程  $B$  的差异度  $\text{diff}(u, u+1)$  等于这两个序列不完全一致的前缀长度之和，即  $|A| + |B| - 2|LCS(A, B)|$ 。给定两个栈  $A, B$ ，通过弹出栈顶和压入栈中两种操作从  $A$  得到  $B$  所需要的最小操作次数等于  $\text{diff}(A, B)$ 。

**结论 4.1.**  $\sum_{u=1}^{n-1} \text{diff}(u, u+1) = O(n)$ 。

证明. 我们可以用下列三种操作按 DFS 序遍历一棵树：

- 下降操作：从  $u$  走到  $\text{ch}(u, 1)$ 。变化量是  $k_u + (k_{\text{ch}(u, 1)} + 1)$ 。
- 转换操作：从  $\text{ch}(u, i)$  走到  $\text{ch}(u, i+1)$ 。变化量是  $(k_{\text{ch}(u, i)} + 1) + (k_{\text{ch}(u, i)} + 2)$ 。
- 上升操作：从  $\text{ch}(u, k_u)$  走到  $u$ 。变化量是  $(k_{\text{ch}(u, k_u)} + 1) + 1$ 。

由于每个结点的度数对总差异度之和的贡献次数  $\leq 3$ ，因此总差异度之和是  $O(n)$  的。□

直接使用线段树维护对  $\text{origin}(u)$  进行追溯的过程，时间复杂度  $O(n \log n)$ ，可以获得 30 分，结合完全二叉树 50 分。

可以使用一个栈维护所有  $c_{e_i} = ?$  的边，将时间复杂度优化到  $O(n)$ 。假如这么做，需要将前缀信息拆成后缀信息处理。

## 5 第二小问的 $O(n \log^2 n)$ 算法

### 5.1 定义与记号

记  $\text{top}(u)$  表示  $u$  所在的重链顶端， $\text{bottom}(u)$  表示  $u$  所在的重链底端。

**定义 5.1** (二元组序列  $\text{circle}(u)$ ). 我们定义  $\text{circle}(u)$  是一个长度为  $k_u + 1$  的二元组序列。按照如下的过程：

- 倒序访问  $u$  到其所有儿子的连边，然后访问  $(u, \text{Fa}(u))$ ；

其中  $u$  到其所有儿子的连边的预期颜色都是白色，而  $(u, \text{Fa}(u))$  的预期颜色是黑色。



**定义 5.2** (二元组序列  $\text{trace}(u, v)$ ). 对于树上的一条从下到上的直链  $u \rightsquigarrow v$ , 我们定义  $\text{trace}(u, v)$  是一个二元组序列。按照如下的过程:

- 倒序访问  $\text{Fa}(u)$  到其所有位于  $u$  左侧儿子的连边, 然后访问  $(\text{Fa}(u), \text{anc}(u, 2))$ ;
- 倒序访问  $\text{anc}(u, 2)$  到其所有位于  $\text{Fa}(u)$  左侧儿子的连边, 然后访问  $(\text{anc}(u, 2), \text{anc}(u, 3))$ ;
- 以此类推, 直到访问完  $(v, \text{Fa}(v))$  为止。

在这个过程中, 所有结点到其儿子的连边预期颜色都是白色, 所有结点到其父亲的连边预期颜色都是黑色。

**定义 5.3** (二元组序列  $\text{chain}(u)$ ). 最后我们对于每个重链顶端结点  $u$ , 定义  $\text{chain}(u) = \text{trace}(\text{bottom}(u), u)$ 。

## 5.2 用重链剖分拆分追溯过程

针对  $\text{origin}(u)$  的追溯过程可以拆成两个部分:  $\text{circle}(u)$  和  $\text{trace}(u, \text{Rt})$ 。

我们知道  $u \rightsquigarrow \text{Rt}$  的直链可以划分成  $\leq \log n$  条轻边和重链前缀。按照重链剖分, 我们把  $\text{trace}(u, \text{Rt})$  拆分成  $\leq \log n$  段  $\text{trace}(u, p)$  和  $\text{trace}(p, q)$  的并, 其中  $p = \text{Fa}(u)$ ,  $q = \text{top}(u)$ 。我们有:

- $\text{trace}(u, p)$  一定是  $\text{circle}(p)$  的一个后缀, 并且所有边的预期颜色也和  $\text{circle}(p)$  相一致;
- $\text{trace}(p, q)$  一定是  $\text{chain}(q)$  的一个后缀, 并且所有边的预期颜色也和  $\text{chain}(u)$  相一致;

我们称这样的后缀叫做重合子段, 于是我们把针对  $\text{origin}(u)$  的追溯过程拆成了  $O(\log n)$  个重合子段。

## 5.3 用数据结构维护重合子段

对于每个  $\text{circle}(u)$  和每个重链顶端的  $\text{chain}(u)$ , 我们各开一个数据结构来维护。

由于每条边只会有效染色一次, 我们可以暴力定位那些需要被染色的边进行染色。所有  $n$  个过程总共只会染色  $n - 1$  条边。

以  $\text{trace}(p, q)$  为例, 假设  $\text{trace}(p, q)$  的第一条边在  $\text{chain}(q)$  中位于位置  $l$ , 那么这个重合子段就是后缀  $[l, m]$ 。既然预期颜色一致, 我们可以直接称呼边的红色和绿色, 用一条边的红色和绿色代指这条边的黑色和白色。

我们用数据结构维护重合子段:



- 我们开一个 set 维护所有无色边的位置, 用于  $\text{assign}(u, x)$  定位所有无色边进行染色;
- 我们开一个 set 维护所有红色边的位置, 用于寻找  $l$  之后第一条红色边的位置;
- 再开一棵线段树维护满足  $c_{e_i} = 1$  或  $c_{e_i} = ?$  之一的  $a_{e_i}$  的区间  $\max$ 、区间  $\min$ , 用于求出  $\max \text{ valid}(u)$  和  $\min \text{ valid}(u)$ 。

处理一个重合子段的时间复杂度是  $O(\log n)$ 。由于重合子段的个数也是  $O(\log n)$  的, 单次操作的时间复杂度是  $O(\log^2 n)$ 。我们可以正确回答第二小问, 时间复杂度  $O(n \log^2 n)$ , 可以获得 70 分。

使用全局平衡二叉树可以优化到  $O(n \log n)$ 。

## 6 第三小问的 $O(n\sqrt{n})$ 算法

### 6.1 单个数据结构的任务

我们将单个数据结构的任务做一个总结:

- 维护两个序列  $a_{e_1} \sim a_{e_m}$ 、 $c_{e_1} \sim c_{e_m}$ , 其中  $a_{e_1} \sim a_{e_m}$  的值不会变化, 所有  $c_{e_i}$  初值为 ?。
- 每次修改操作会将某个  $c_{e_i}$  变为 0/1。
- 每次查询操作一个后缀  $[l, m]$  和一个权值  $x$ , 查询  $\sum_{i=l}^m [a_{e_i} < x] w'_i \prod_{j=l}^{i-1} w_j$ 。其中:
  - 对于  $c_{e_i} = ?$ :  $w_i = w'_i = \frac{1}{2}$ ;
  - 对于  $c_{e_i} = 0$ :  $w_i = 1$ ,  $w'_i = 0$ ;
  - 对于  $c_{e_i} = 1$ :  $w_i = 0$ ,  $w'_i = 1$ 。
- 另外还需要返回  $\prod_{j=l}^m w_j$ , 以供后续查询使用。

注意到单个数据结构的任务可以归约  $x < A$  乘  $y < B$  和问题, 因此追求一个  $O(n\sqrt{n})$  算法是明智的。



## 6.2 静态情况

### 6.2.1 数点问题

静态情况指的是  $c_{e_i}$  具有 0/1/? 初值, 但是不会变化的情况。

开一棵线段树维护序列  $a_{e_1} \sim a_{e_m}$ 。在每个线段树结点  $u$  上记一个 vector  $M_u$  表示  $a_{e_{l_u}} \sim a_{e_{r_u}}$  排序后的结果, 那么  $M_u$  可以由  $M_{lch(u)}$  和  $M_{rch(u)}$  直接归并得到。

查询时在  $[l, m]$  分解为的  $O(\log n)$  个线段树结点上各自二分即可。

假如采用主席树的办法, 这种办法没有希望扩展到动态情况。

### 6.2.2 带权问题

定义一个  $i$  在  $[l_u, r_u]$  中的贡献是  $w'_i \prod_{j=l_u}^{i-1} w_j$ 。

那么再开一个与  $M_u$  等长的 vector  $F_u$  表示  $[l_u, r_u]$  中所有  $a_{e_i} < x$  的  $i$  的贡献之和, 再记录  $\prod_{i=l_u}^{r_u} w_i$  的值, 即可支持 pushup。

可以看出线段树的结构处理带权问题游刃有余。

## 6.3 动态情况

**引理 6.1** (重链剖分的性质)。对于任意一个结点  $u \rightsquigarrow Rt$  的直链, 我们可以把  $u \rightsquigarrow Rt$  分解为  $k \leq \log n$  段重链前缀。假设这  $k$  段重链的顶端按照从上到下的顺序依次为  $z_1, z_2, \dots, z_k$ , 那么一定有  $siz(z_i) \leq \frac{n}{2^{i-1}}$ 。

根据这个性质我们可以得到, 跳到从上往下数第  $i$  条重链的时候, 所访问的两个数据结构所维护的序列长度必然分别  $\leq \frac{n}{2^{i-1}}$ 。因此为了得到一个  $O(n\sqrt{n})$  做法, 只需要能在  $O(\sqrt{n})$  的时间复杂度内处理单个数据结构的修改查询即可。

### 6.3.1 AKEE 线段树

以下线段树结构的名字叫 AKEE 线段树<sup>2</sup>:

- 记一个线段树结点的长度  $\text{len}(u) = r_u - l_u + 1$ ;
- 设置一个阈值  $B$ , 只对  $\text{len}(u) \leq B$  的线段树结点维护出  $M_u$ 。

<sup>2</sup><https://www.cnblogs.com/dmoransky/p/14957063.html>



- 修改的时间消耗来源于不断向上 pushup。由于单次修改重构的区间长度和  $= B + \frac{B}{2} + \frac{B}{4} + \cdots = 2B$ ，总复杂度  $O(B)$ 。
- 查询只在当前节点完全包含于查询区间且  $\text{len}(u) \leq B$  时，在该点上相应查询。访问的结点分为两种：
  1. 父亲已经完全包含于查询区间，但是父亲长度  $> B$  暴力递归下来的结点。至多  $O(\frac{n}{B})$  个。
  2. 位于查询区间的两侧，父亲并不属于查询区间的结点。至多  $O(\log n)$  个。

我们把访问到的结点个数  $O(\frac{n}{B} + \log n)$  乘上暴力二分的  $O(\log n)$  得到在单个数据结构上查询的时间复杂度  $O(\frac{n \log n}{B} + \log^2 n)$ 。

在  $O(\log n)$  个数据结构上查询的总时间复杂度  $O(\frac{n \log n}{B} + \log^3 n)$ 。取  $B = O(\sqrt{n \log n})$  平衡得时间复杂度  $O(n \sqrt{n \log n})$ ，并不令人满意。

AKEE 线段树是很好实现的，只需要在 pushup 和 query 的时候加一个 if 条件就行。相比于直接分块，按照线段树结点维护的区间 len 根号分治并不会带来任何复杂度优化，这样的线段树结构唯一的作用在于简化代码。

常数正常的话可以获得 70 ~ 90 分。

### 6.3.2 分散层叠技术

最后我们用分散层叠技术 [1] 优化 AKEE 线段树。

我们希望利用  $x$  在父亲结点  $u$  上二分的结果  $O(1)$  确定出在两个子结点  $\text{lch}(u)$  和  $\text{rch}(u)$  上二分的结果。

- 对于长度  $\leq B$  的结点，我们只需要保留归并的过程即可做到。具体来说我们在归并  $M_u$  的时候顺便记录两个 vector  $L_u, R_u$ ，表示归并到这个位置的时候左右子结点两个指针分别指到哪了，就可以  $O(1)$  确定。
- 对于长度  $> B$  的结点，我们扩展定义  $M_u$  表示  $M_{\text{lch}(u)}$  和  $M_{\text{rch}(u)}$  各取  $1/3$  的元素归并起来的结果。我们将  $M_{\text{lch}(u)}$  和  $M_{\text{rch}(u)}$  分别每隔 3 个元素取一个元素，然后归并起来。这样只需要暴力往后找  $\leq 2$  个元素就可以定位到，仍然是  $O(1)$  的。

于是只需要在线段树的根处做一次二分即可。

- 对于  $\text{len} > B$  的部分，单次修改重构的区间长度和  $= B + \frac{2B}{3} + \frac{4B}{9} + \cdots = 3B$ ，时间复杂度  $O(B)$ 。



- 查询操作递归到长度  $\leq B$  的结点时应该已经知道了二分位置，单个复杂度  $O(\frac{n}{B} + \log n)$ ，一次询问总计  $O(\frac{n}{B} + \log^2 n)$ 。

取  $B = O(\sqrt{n})$  平衡得时间复杂度  $O(n\sqrt{n})$ ，可以获得满分 100 分。

随着对排列  $p$  的扫描，上述做法在线统计出了每个  $i$  的贡献（一个  $i$  的贡献指的是满足  $q_i < p_i$  且  $\forall j < i$  都有  $q_j = p_j$  的合法排列  $q$  的个数）。

同时也有一种离线的办法可以避免分散层叠。既然已经知道了完整的排列  $p$ ，我们可以把整个统计过程中所有的染色操作和查询操作全部离线下来。这样一来，每个数据结构只需要用相对简单的序列分块实现即可。

## 7 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练彭思进的指导。

感谢梁霄教练、叶金毅教练、谷多玉教练对我的培养与教导。

感谢许庭强学长、向昱帆学长、黄洛天学长、董亦骐学长、王梓霖学长、李春进学长、严思珩同学对我的帮助与启发。

感谢黄洛天学长对本文提出的建议。

感谢父母对我的陪伴与支持。

## 参考文献

- [1] 蒋明润. 浅谈利用分散层叠算法对经典分块问题的优化. 2020  
年信息学奥林匹克中国国家队候选队论文集, 2020.



# 顶点带权的图匹配问题

华东师范大学第二附属中学 柯绎思

## 摘要

本文介绍了一种时间复杂度为  $O(m\sqrt{n})$  的一般图最大匹配算法以及一种时间复杂度为  $O(m\sqrt{n}\log n)$  的一般图最大点权匹配算法。

## 1 引言

本文研究了顶点带权的图上的匹配问题。一个匹配的权值是所有被它覆盖的点的点权之和，本文将研究如何求出图中权值最大的匹配。在信息学竞赛中，图的最大匹配以及边上带权的最大权匹配问题已经有广泛的研究与应用，但是关于最大点权匹配的研究却不多。

如果令一条边的权值为它所连接的两个点的点权之和，那么最大点权匹配就转化为了最大权匹配。然而事实上，最大点权匹配是显著易于最大权匹配的，图的最大点权匹配与最大匹配密切相关。本文将给出一个时间复杂度为  $O(m\sqrt{n}\log n)$  的最大点权匹配算法。

本文的第二节给出所需的定义；第三节先介绍一个时间复杂度为  $O(m\sqrt{n})$  的一般图最大匹配算法，作为最大点权匹配算法的前置步骤；第四节先介绍在点权非负情况下的二分图的最大点权匹配算法，然后把一般图上的最大点权匹配问题转化为二分图上的问题，最后说明处理负点权的方法。

## 2 记号与约定

对于一张图  $G$ ，记  $V$  表示其点集， $E$  表示其边集，记作  $G = (V, E)$ 。对于一个二分图  $G$ ，如果其左部点为  $V_1$ ，右部点为  $V_2$ ，边集为  $E$ ，有时也记作  $G = (V_1, V_2, E)$ 。

图  $G$  的一个匹配  $M$  为边集  $E$  的一个子集，满足  $M$  中任意两条边没有公共点，称  $|M|$  为匹配的大小。与  $M$  中至少一条边相邻的点构成的集合称为  $M$  所覆盖的点集。

$G$  的所有匹配中大小最大的称为  $G$  的最大匹配。

如果没有额外说明，用  $w_u$  表示点  $u$  的点权。一个匹配的权值为所有被它覆盖的点的点权之和。

$G$  的所有匹配中权值最大的称为  $G$  的最大点权匹配。

对于两个集合  $S, T$ , 用  $S \setminus T$  表示在  $S$  中且不在  $T$  中的元素构成的集合,  $S \oplus T$  表示  $S$  与  $T$  的对称差, 即  $(S \setminus T) \cup (T \setminus S)$ 。

在下文中, 如果没有额外说明, 用  $G$  表示当前所讨论的图,  $V$  表示其点集,  $E$  表示其边集, 记  $n = |V|$ ,  $m = |E|$ 。

### 3 一般图最大匹配

#### 3.1 匹配与增广路

**定义 3.1** (交错路). 对于一个匹配  $M$ , 如果一条简单路径交替地经过不在  $M$  中的边和在  $M$  中的边, 那么称这条路径为 (关于  $M$  的) 交错路。

**定义 3.2** (增广路). 对于一个匹配  $M$ , 如果一条交错路连接两个不在  $M$  中的顶点, 则称这条路径为 (关于  $M$  的) 增广路。

在没有歧义的情况下, 对于一条路径  $P = (v_1, v_2, \dots, v_k)$ , 我们同时让  $P$  代表它所经过的边集  $\{(v_1, v_2), \dots, (v_{k-1}, v_k)\}$ 。

由增广路的定义可以立刻得到, 设  $M$  为匹配,  $P$  为关于  $M$  的增广路, 则  $M \oplus P$  也是匹配, 且  $|M \oplus P| = |M| + 1$ 。

**引理 3.1.** 设  $M, N$  为两组匹配,  $|M| = r, |N| = s$  且  $r > s$ , 则  $M \oplus N$  至少包含  $r - s$  条两两不相交的关于  $N$  的增广路。

**证明.** 考虑子图  $\bar{G} = (V, M \oplus N)$ , 由于  $M$  和  $N$  是匹配,  $\bar{G}$  中每个点最多与一条  $M \setminus N$  和一条  $N \setminus M$  中的边相邻。因此  $\bar{G}$  的每个连通块一定是

- (i) 孤立点;
- (ii) 一个长度为偶数的环, 上面的边交替地属于  $M \setminus N$  和  $N \setminus M$ ;
- (iii) 一条简单路径, 上面的边交替地属于  $M \setminus N$  和  $N \setminus M$ 。

对于一个连通块  $C_i = (V_i, E_i)$ , 记  $\delta_i = |M \cap E_i| - |N \cap E_i|$ , 则  $\delta_i \in \{-1, 0, 1\}$  且  $\sum_i \delta_i = r - s$ 。

因此至少有  $r - s$  个连通块  $C_i$  满足  $\delta_i = 1$ , 这些连通块就是  $r - s$  条两两不相交的关于  $N$  的增广路。  $\square$

**推论 3.1.**  $M$  是最大匹配当且仅当图中不存在关于  $M$  的增广路。

**推论 3.2.** 设  $M$  为匹配,  $|M| = r$ , 最大匹配的大小为  $s > r$ , 则存在一条关于  $M$  的增广路, 其长度  $\leq 2 \left\lfloor \frac{r}{s-r} \right\rfloor + 1$ 。



证明. 令  $N$  为最大匹配, 则  $M \oplus N$  包含至少  $s - r$  条两两点不相交的关于  $M$  的增广路, 而这些路径包含至多  $r$  条  $M$  中的边, 因此有一条路径包含不超过  $\lfloor \frac{r}{s-r} \rfloor$  条  $M$  中的边, 即长度  $\leq 2 \lfloor \frac{r}{s-r} \rfloor + 1$ .  $\square$

设  $M$  为匹配, 所有关于  $M$  的增广路中长度最短的称为  $M$  的最短增广路。

**引理 3.2.** 设  $M$  为匹配,  $P$  是  $M$  的最短增广路,  $P'$  是  $M \oplus P$  的增广路, 则  $|P'| \geq |P| + |P \cap P'|$ 。

证明. 令  $N = M \oplus P \oplus P'$ , 则  $N$  为匹配且  $|N| = |M| + 2$ , 因此  $N \oplus M$  包含两条点不相交的关于  $M$  的增广路, 设为  $P_1, P_2$ 。那么有  $P \oplus P' = M \oplus N$ , 故  $|P \oplus P'| = |M \oplus N| \geq |P_1| + |P_2| \geq 2|P|$ 。又因为  $|P \oplus P'| = |P| + |P'| - |P \cap P'|$ , 所以  $|P'| \geq |P| + |P \cap P'|$ 。  $\square$

由引理 3.2 可以设想以下计算方法: 从初始的空匹配  $M_0 = \emptyset$  出发, 每次找到  $M_i$  的最短增广路  $P_i$  并令  $M_{i+1} = M_i \oplus P_i$ , 得到序列  $M_0, M_1, \dots$ , 直到没有增广路。

**推论 3.3.** 上述过程中  $|P_{i+1}| \geq |P_i|$ 。

**推论 3.4.** 对于任意  $i \neq j$ , 如果  $|P_i| = |P_j|$ , 那么  $P_i$  和  $P_j$  点不相交。

**定理 3.1.** 设  $s$  为最大匹配的大小, 序列  $|P_0|, |P_1|, \dots, |P_{s-1}|$  中本质不同的元素个数不超过  $2\sqrt{s} + 1$ 。

证明. 令  $r = \lfloor s - \sqrt{s} \rfloor$ , 则  $|M_r| = r$ , 由推论 3.2,  $|P_r| \leq 2 \frac{\lfloor s - \sqrt{s} \rfloor}{s - \lfloor s - \sqrt{s} \rfloor} + 1 \leq 2 \lfloor \sqrt{s} \rfloor + 1$ 。因此  $|P_0|, \dots, |P_r|$  只能取不超过  $2 \lfloor \sqrt{s} \rfloor + 1$  的奇数, 共  $\lfloor \sqrt{s} \rfloor + 1$  个, 而  $|P_{r+1}|, \dots, |P_{s-1}|$  也只有  $s - r - 1 \leq \sqrt{s}$  个, 故总共不同的取值不超过  $2\sqrt{s} + 1$  个。  $\square$

由定理 3.1 可以得到以下算法: 初始匹配  $M = \emptyset$ , 每一轮中假设当前  $M$  的最短增广路长度为  $l$ , 找到一组极大的增广路集合  $\{Q_1, Q_2, \dots, Q_k\}$ , 满足  $|Q_i| = l$ ,  $Q_i$  两两点不相交, 令  $M \leftarrow M \oplus Q_1 \oplus \dots \oplus Q_k$ 。

下面几部分将展示一个算法, 其能在  $O(n + m)$  的时间复杂度内找到一组极大的最短增广路集合。

## 3.2 基本框架

固定当前的一组匹配  $M$ , 以下的交错路和增广路都是关于  $M$  的。

对于一个点  $u$ , 定义  $evenlevel_u$  表示从一个未匹配点出发, 经过长度为偶数的交错路到达  $u$  的最短距离, 如果不存在这样的路径, 那么  $evenlevel_u = +\infty$ 。类似地定义  $oddlevel_u$ 。定义  $level_u = \min(evenlevel_u, oddlevel_u)$ 。如果  $level_u$  为偶数, 则称  $u$  为外部点, 否则称  $u$  为内部点。

如果一个点  $u$  是外部点, 则称其另一 level 为  $oddlevel_u$ , 否则称其另一 level 为  $evenlevel_u$ 。

定义原图  $G = (V, E)$  的 BFS 图  $\tilde{G} = (V, \tilde{E})$  为一个有向图，其点集仍为  $V$ 。对于一条边  $(u, v) \in E$ ，当  $v$  为外部点时， $(u, v) \in \tilde{E}$  当且仅当  $(u, v)$  为匹配边且  $\text{oddlevel}_u + 1 = \text{level}_v$ ；当  $v$  为内部点时  $(u, v) \in \tilde{E}$  当且仅当  $\text{evenlevel}_u + 1 = \text{level}_v$ 。

**定义 3.3** (花、花托). 一个长度为奇数的简单环  $(b_1, b_2, \dots, b_{2n+1})$ ，如果对于任意  $i = 1, \dots, n$ ， $(b_{2i}, b_{2i+1}) \in M$ ，则称  $B = \{b_2, b_3, \dots, b_{2n+1}\}$  为一个花，其花托  $\text{base}(B)$  为  $b_1$ 。

注意花  $B$  的花托并不属于  $B$ 。

在没有歧义的情况下，我们同时让  $B$  代表边集  $\{(b_1, b_2), \dots, (b_{2n}, b_{2n+1}), (b_{2n+1}, b_1)\}$ 。

图 1 展示了由  $\{b_2, \dots, b_{2n+1}\}$  构成的花，其花托为  $b_1$ 。

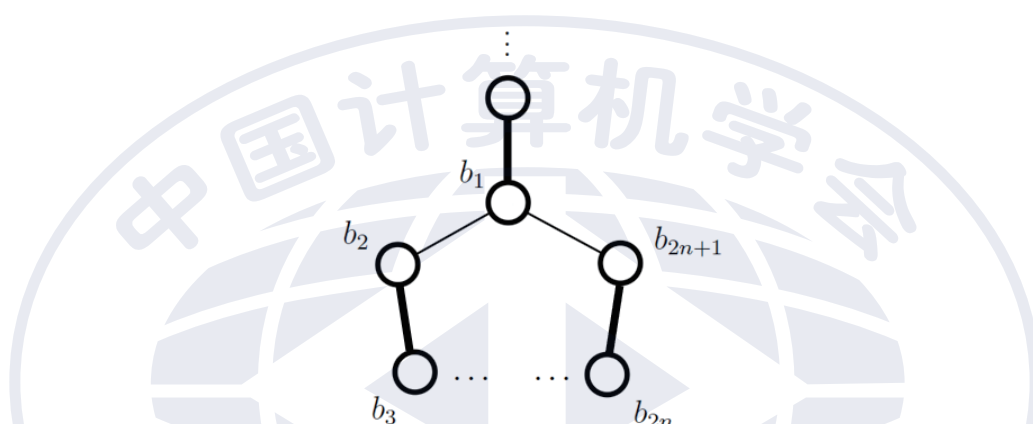


图 1:  $(b_1, \dots, b_{2n+1})$  构成的花

对于一个花  $B$ ，设  $d$  表示从一个未匹配点出发，在不经过  $B$  中点的情况下，经过长度为偶数的交错路到达  $\text{base}(B)$  的最短距离，如果不存在这样的路径，那么  $d = +\infty$ 。定义  $\text{tenacity}(B) = 2d + \text{奇环长度}$ 。

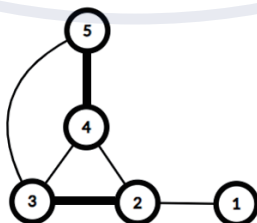


图 2: 一个例子

例如图 2 中，对于花  $B = \{2, 3\}$ ， $\text{base}(B) = 4$ ，虽然存在 1 到 4 的长度为偶数的交错路，但是它必须经过 2 和 3，因此  $\text{tenacity}(B)$  还是  $+\infty$ 。

对于一条增广路  $P$ ，定义  $\text{tenacity}(P)$  为其长度。

对于点  $u$ , 定义  $tenacity(u) = evenlevel_u + oddlevel_u$ 。

对于匹配边  $(u, v)$ , 定义  $tenacity(u, v) = oddlevel_u + oddlevel_v + 1$ ; 对于非匹配边  $(u, v)$ , 定义  $tenacity(u, v) = evenlevel_u + evenlevel_v + 1$ 。

**定义 3.4 (桥).** 对于一条边  $(u, v)$ , 如果  $tenacity(u, v) < +\infty$ , 则称  $(u, v)$  为桥 (**bridge**)。

该算法首先会从所有未匹配点开始 BFS, BFS 的过程中会计算出所有点  $u$  的  $level_u$ 。在第  $k$  层 BFS 中, BFS 会枚举所有  $evenlevel$  或  $oddlevel$  为  $k$  的点, 用它们更新  $level$  为  $k + 1$  的点。

在此过程中会发现一些桥, 第  $k$  层 BFS 结束后, 对于所有  $tenacity$  为  $2k + 1$  的桥  $(s, t)$ , 调用  $BLOSS-AUG(s, t)$ 。

$BLOSS-AUG(s, t)$  会尝试找到一条长度为  $2k + 1$  的包含边  $(s, t)$  的增广路。如果无法找到, 那么它会找到一个  $tenacity$  为  $2k + 1$  的花, 之后会对所有花中的点更新其另一  $level$ , 在此过程中又会发现一些桥。

如果在第  $k$  层找到了一条增广路, 首先会还原出完整的增广路, 然后对增广路上所有的点  $u$  调用  $ERASE(u)$ 。  $ERASE(u)$  会把  $u$  标记为删除, 并进一步删除无法从未匹配点通过 BFS 图到达的点。在所有  $tenacity$  为  $2k + 1$  的桥处理完之后, 算法结束。

### 3.3 BFS

BFS 的目的是正确计算出所有点的  $level$ 。

在第  $k$  层 BFS 中, 如果  $k$  为奇数, 那么 BFS 会枚举所有  $oddlevel_u = k$  的点  $u$ , 找到其匹配点  $v$ 。如果  $oddlevel_v < +\infty$ , 说明  $(u, v)$  为桥, 将其加入  $bridge(tenacity(u, v))$ 。否则  $evenlevel_v = k + 1$ , 将  $(u, v)$  加入 BFS 图。

如果  $k$  为偶数, 那么 BFS 会枚举所有  $evenlevel_u = k$  的点  $u$ , 再遍历所有边  $(u, v)$ , 满足  $(u, v)$  不是匹配边。如果  $evenlevel_v < +\infty$ , 说明  $(u, v)$  为桥, 将其加入  $bridge(tenacity(u, v))$ 。否则, 如果  $oddlevel_v < k + 1$ , 这时将  $u$  加入  $anomaly(v)$ , 这步的作用之后会说明。如果以上情况都不满足, 就有  $oddlevel_v = k + 1$ , 并将  $(u, v)$  加入 BFS 图。

注意上述过程中要求  $v$  不能被标记为删除, 且当  $k$  为奇数时  $u$  不在任何花中。

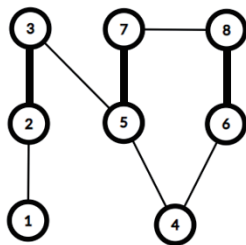


图 3: 一个例子

例如图 3 中,  $level$  为 0 的点有 1、4,  $level$  为 1 的点有 2、5、6,  $level$  为 2 的点有 3、7、8。当遍历到边  $(3,5)$  时, 会把 3 加入  $anomaly(5)$ 。

### 3.4 BLOSS-AUG

在第  $k$  层 BFS 结束后, 对所有  $tenacity = 2k+1$  的桥  $(s,t)$ , 算法会调用  $BLOSS-AUG(s,t)$ 。 $BLOSS-AUG(s,t)$  的目的是找到一条包含  $(s,t)$  且长度为  $2k+1$  的增广路, 如果这样的增广路不存在, 那么它会找到一个包含  $(s,t)$  且  $tenacity = 2k+1$  的花。

该部分接下来所指的“路径”都是 BFS 图上的路径, 因而一定是交错路。

先假设在此之前还未发现过花, 那么如果存在增广路, 其一定是由两条从未匹配点出发, 分别到达  $s$  和  $t$ , 且点不相交的路径构成的。想要找到这两条路径, 首先得解决这样的问题:

**问题 1.** 在一个有向图  $H$  中, 给定两个点  $S$  和  $T$ , 判断是否存在两条  $S$  到  $T$  的路径, 满足它们除了端点外点不相交。

由最大流最小割定理, 存在满足条件的路径当且仅当不存在点  $u \neq S, T$ , 使得删掉点  $u$  后  $S$  不可达  $T$ 。换句话说, 如果建出以  $S$  为根的支配树 [1], 那么存在满足条件的路径的充要条件就是  $T$  的直接支配点为  $S$ 。

回到原算法, 如果建一个超级源  $S$  指向所有未匹配点, 并把  $s$  和  $t$  指向一个超级汇  $T$ , 那么就转化为问题 1。

为了找到满足条件的路径, BLOSS-AUG 会进行 Double Depth First Search(DDFS)。DDFS 会在 BFS 图的反图上同时从  $s$  和  $t$  生长出两棵不交的 DFS 树  $T_l, T_r$ 。用  $x$  和  $y$  分别代表当前  $T_l$  和  $T_r$  的搜索中心, 初始时  $x = s, y = t$ 。

对于每个  $T_l$  中的点, 将其打上 left 标记; 对于每个  $T_r$  中的点, 将其打上 right 标记。对于在  $T_l$  或  $T_r$  中的点  $u$ , 记  $par_u$  表示  $u$  在 DFS 树上的父节点。过程中还要维护一个变量  $dcv$ (Deepest Common Vertex), 初始时未赋值, 其作用稍后会说明。

每一时刻, 如果  $level_x \geq level_y$ , DDFS 就会调用 LDFS, 否则调用 RDFS。LDFS 会找一条未使用过的边  $(z,x)$ , 如果  $z$  未被打过标记, 那么  $z$  加入  $T_l$ , 令  $par_z \leftarrow x, x \leftarrow z$ , 并退出。否则如果  $z = y$ , 就让  $dcv \leftarrow y$ 。如果  $x$  未能找到出边, 那么进行回溯, 令  $x \leftarrow par_x$ 。但是如果  $x$  无法回溯了, 说明无法找到另一个比  $dcv$  深的点, 此时强制将  $dcv$  加入  $T_l$ , 从  $T_r$  中删除, 令  $x \leftarrow dcv, y \leftarrow par_y$ 。如果此时  $y$  也无法回溯, 就说明不存在两条点不相交的路径。

为了防止  $x$  和  $y$  重复回溯, 还要维护两个变量  $barrier_x$  和  $barrier_y$ , 初始时分别为  $s$  和  $t$ 。每次  $x \leftarrow dcv$  时就将  $barrier_x$  设为  $x$ , 如果  $x = barrier_x$  就说明  $x$  无法回溯。

为了在  $x \leftarrow dcv$  后更新  $par_x$ , 可以维护一个  $lstx$  在每次  $dcv \leftarrow y$  时令  $lstx \leftarrow x$ , 这样  $par_x$  就可设为  $lstx$ 。

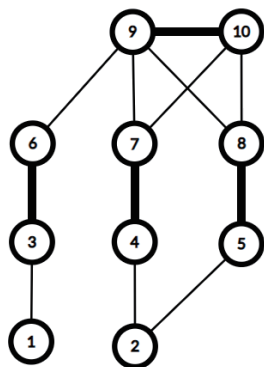


图 4: 一个例子

例如在图 4 中，如果调用  $\text{BLOSS-AUG}(9,10)$ ，以下过程可能会发生：

- (1) 初始化  $x = \text{barrierox} = 9, y = \text{barriery} = 10$ 。
- (2) LDFS 把  $x$  沿  $(7,9)$  移到 7。
- (3) RDFS 把  $y$  沿  $(8,10)$  移到 8。
- (4) LDFS 把  $x$  沿  $(4,7)$  移到 4。
- (5) RDFS 把  $y$  沿  $(5,8)$  移到 5。
- (6) LDFS 把  $x$  沿  $(2,4)$  移到 2。
- (7) RDFS 遍历到边  $(2,5)$ ，但 2 已经被标上 left，由于  $x = 2$ ，把  $d_{cv}$  设为 2。
- (8) 5 已经没有其他入边，一直回溯到 10。
- (9) RDFS 遍历到边  $(7,10)$ ，但 7 已经被标上 left。
- (10)  $y = \text{barriery}$  无法回溯，把  $y$  和  $\text{barriery}$  设为 2，把  $x$  设为 4。
- (11)  $x$  一直回溯到 9。
- (12) LDFS 遍历到边  $(8,9)$ ，但 8 已经被标上 right。
- (13) LDFS 把  $x$  沿  $(6,9)$  移到 6，并最终移到 1。

附录 A.1 展示了 DDFS 过程的伪代码，注意此时还没有考虑花的存在。

如果 DDFS 结束后  $x$  和  $y$  都是未匹配点，说明找到了从  $x$  到  $s$  以及从  $y$  到  $t$  的点不相交的路径，下一部分将会说明怎么还原增广路。

否则 DDFS 未能找到两条点不相交的路径。把  $d_{cv}$  上的 left 或 right 标记去掉，并从  $T_l$  或  $T_r$  中删掉。需要注意到此时有以下事实：

性质 3.1. 如果建立超级源  $S$  指向所有未匹配点, 并以  $S$  为根建立支配树, 那么  $dcv$  就是  $s$  和  $t$  在支配树上的最近公共祖先。

性质 3.2. 记  $B$  为除了  $dcv$  以外所有满足  $dcv$  可达  $u$  且  $u$  可达  $s$  或  $t$  的点  $u$  构成的集合, 那么任何从未匹配点到  $B$  中某个点的路径都一定经过  $dcv$ 。

性质 3.3.  $T_l$  和  $T_r$  分别是以  $s$  和  $t$  为根的树, 且  $T_l$  和  $T_r$  的点集构成  $B$  的一个划分。

性质 3.4. 对于每一个  $B$  中的点  $u$ , 都存在一个以  $dcv$  作为花托且  $tenacity$  为  $2k+1$  的花包含  $u$ 。

性质 3.5.  $dcv$  为外部点。

可以推广一下花的定义, 把  $B$  看成一个花,  $base(B) = dcv$ , 记  $peakL(B) = s$ ,  $peakR(B) = t$ 。  $B$  中每一个点  $u$  的另一  $level$  就是  $2k+1 - level_u$ 。 由于 BFS 图上不存在边  $(u, v)$  满足  $u \notin B \cup \{base(B)\}, v \in B$ , 可以把  $B$  中所有点都缩到  $base(B)$  上, 也就是对于 BFS 图上每条满足  $u \in B, v \notin B$  的边  $(u, v)$ , 将其替换为  $(base(B), v)$ 。

后续其他的 BLOSS-AUG 过程中都可以在修改过的 BFS 图中进行, 在寻找新的花时需要跳过以前已经在花中的点, 因为这些点的  $evenlevel$  与  $oddlevel$  都已经确定。可以再注意到以下事实:

性质 3.6. 对于一个点  $u$ ,  $tenacity(u) < +\infty$  当且仅当  $u$  在一个花中。

性质 3.7. 一个点最多只在一个花中。

记  $blo_u$  表示  $u$  所属的花, 如果  $u$  不在任何花中, 则  $blo_u = \text{undefined}$ 。 任何时刻, 对于一条原本 BFS 图中的边  $(u, v)$ , 设  $u'$  为不断地令  $u \leftarrow base(blo_u)$  直到  $u$  不属于任何花后的值, 那么  $(u, v)$  现在会变为  $(u', v)$ , 记  $base^*(u) = u'$ ,  $base^*$  可以用并查集维护。

考虑了花的 DDFS 过程也只需对原算法稍加修改: 初始时令  $x = base^*(s)$ ,  $y = base^*(t)$ ; 之后每次遍历到一条未使用过的边  $(z, x)$  时, 只需先令  $z \leftarrow base^*(z)$  再执行后续操作即可。

修改完花  $B$  中的点的另一  $level$  后, 可能会产生一些新的桥。新产生的桥只有两种情况: 两端都在  $B$  中, 或恰好有一端在  $B$  中。如果桥的两端都在  $B$  中, 那么它已经没有任何作用, 因此不用处理。对于恰好有一端在  $B$  中的桥  $(u, v)$ , 设  $u \in B, v \notin B$ , 那么  $(u, v)$  为非匹配边, 因此  $u$  为内部点,  $evenlevel_v > oddlevel_u$ , 故  $v \in anomaly(u)$ 。反之, 对于任何  $v \in anomaly(u)$ ,  $(u, v)$  都能构成一个桥。

因此, 在找到一个花之后, 要对其中每个点  $u$ , 遍历所有  $v \in anomaly(u)$ , 若  $tenacity(u, v) = 2k' + 1$ , 则将其加入  $bridge(2k' + 1)$ 。由于  $evenlevel_v > oddlevel_u$ , 因此  $2k' + 1 = evenlevel_v + evenlevel_u + 1 = evenlevel_v + 2k + 1 - oddlevel_u + 1 > 2k + 1$ , 第  $k$  层找到的桥的  $tenacity$  一定  $> 2k + 1$ 。



### 3.5 还原增广路

如果在 BLOSS-AUG( $s, t$ ) 过程中, 两个搜索中心  $x, y$  都走到了未匹配点, 说明找到了两条不相交的从  $x$  到  $s$  和  $y$  到  $t$  的路径。下面说明如何还原  $x$  到  $s$  的增广路,  $y$  到  $t$  的增广路可用类似的方法还原。

首先, 从  $x$  出发顺着  $par$  数组还原出一个原始序列  $a_1, a_2, \dots, a_k$ 。由于花的存在,  $a_i$  到  $a_{i+1}$  不一定直接有边相连, 而且  $a_k = base^*(s)$  也不一定为  $s$ 。为了处理这种情况, 在 DDFS 过程中还要额外维护一个数组  $top$ , 它记录了从  $par_u$  遍历到  $u$  所经过的边的原本的起点, 也即如果 BFS 图上的边  $(v, par_u)$  满足  $base^*(v) = u$ , 那么  $top_u = v$ 。特别地, 令  $top_{base^*(s)} = s$ ,  $top_{base^*(t)} = t$ 。

接下来, 对于每个  $a_i$ , 如果  $blo_{a_i} \neq \text{undefined}$ , 就将其替换为  $b_1, b_2, \dots, b_l$ , 满足  $b_1 = a_i$ ,  $b_l = top_{a_i}$ ,  $b_{i-1} = base(blo_{b_i})$  ( $i = 2, \dots, l$ )。之后, 对于修改后的每个  $a_i$ , 如果  $blo_{a_i} \neq \text{undefined}$ , 就将  $a_i$  替换为  $OPEN(a_i)$ 。

$OPEN(u)$  会返回从  $base(blo_u)$  到  $u$  且长度为偶数的交错路 (不包含  $base(blo_u)$ )。记  $B = blo_u$ 。如果  $u$  为外部点, 那么直接沿着 BFS 图走到  $base(B)$  即可。为了方便, 可以在 DDFS 过程中记  $dw_u$  表示  $u$  在 BFS 图上任何一条入边的起点。顺着  $dw$  数组还原出一个原始序列  $a$ , 之后类似上一段的方法还原增广路即可。

如果  $u$  为内部点, 不妨假设  $u$  上的标记为  $left$ , 那么路径一定是先从  $base(B)$  走到  $peakR(B)$ , 再从  $peakL(B)$  走到  $u$ 。用  $par$  数组分别还原出这两段的原始序列  $a_l, a_r$ , 然后分别用上述方法还原出增广路再拼接起来即可。

### 3.6 ERASE

BLOSS-AUG 在找到一条增广路  $P$  之后, 首先令当前的匹配  $M$  变成  $M \oplus P$ , 然后为了保证找到的所有增广路点不相交, 要对  $P$  中每个点  $u$  调用  $ERASE(u)$ 。 $ERASE(u)$  先把  $u$  标记为删除, 之后会遍历  $u$  在 BFS 图上的每一条出边  $(u, v)$ , 判断  $v$  在 BFS 图上是否有还未删除的前驱, 如果没有则调用  $ERASE(v)$ 。判断  $v$  是否有还未删除的前驱可以通过记录  $deg_v$  表示  $v$  有多少未被删除的前驱。这种删除方式保证了对于任何未删除的点  $u$ , BFS 图上都存在一条从未匹配点到点  $u$  的路径, 上面不经过任何已经被删除的点。

### 3.7 正确性与复杂度说明

**引理 3.3.** 如果一个花或增广路  $S$  包含点  $u$ , 那么  $tenacity(u) \leq tenacity(S)$ ; 如果  $S$  包含边  $(u, v)$ , 那么  $tenacity(u, v) \leq tenacity(S)$ 。

证明. 只证明  $S$  包含  $u$  的情况,  $S$  包含  $(u, v)$  的情况类似。如果  $S$  为增广路, 那么  $u$  往两个方向走就是两条长度不同奇偶的交错路, 因此  $tenacity(u) \leq tenacity(S)$ 。如果  $S$  为花, 那么  $u$  沿着

环的两个方向走到  $base(S)$ ，再从  $base(S)$  沿着最短的不经过  $S$  中其他点的长度为偶数的交错路走到一个未匹配点，也是两条长度不同奇偶的交错路，因此  $tenacity(u) \leq tenacity(S)$ 。□

设最短增广路的长度为  $L$ 。

**引理 3.4.** 对于一个点  $u$ ，如果  $tenacity(u) \leq L$ ，那么一定存在一个花或增广路  $S$  使得  $S$  包含  $u$  且  $tenacity(S) = tenacity(u)$ 。

**引理 3.5.** 对于一条边  $(u, v)$ ，如果  $tenacity(u, v) \leq L$ ，那么一定存在一个花或增广路  $S$  使得  $S$  包含  $(u, v)$  且  $tenacity(S) = tenacity(u, v)$ 。

**引理 3.6.** 所有点的  $oddlevel$  和  $evenlevel$  都能被正确计算。

证明. 对 BFS 的层数归纳，同时证明以上三个引理。假设当前在第  $k$  层，所有  $level_u < k$  的点  $u$  的  $level$  都已经被正确计算，且所有  $tenacity(u) < 2k + 1$  的点  $u$  的另一  $level$  已经被正确计算。由于 BFS 枚举所有  $oddlevel$  或  $evenlevel$  为  $k - 1$  的点来更新  $level$  为  $k$  的点，因此  $level_u = k$  的点  $u$  的  $level$  一定能被正确计算。

对于一个  $tenacity$  为  $2k + 1$  的桥  $(s, t)$ ，由算法的过程可知一定能找到一个包含  $(s, t)$  的增广路或花  $S$  满足  $tenacity(S) = 2k + 1$ 。而且在第  $k$  层找到的桥要么其  $tenacity > 2k + 1$ ，要么已经被包含在一个花中，因此所有桥都能被找到。

对于一个  $tenacity$  为  $2k + 1$  的点  $u$ ，考虑  $u$  在 BFS 图上的入边  $(v, u)$ 。不妨假设  $u$  为外部点，则  $oddlevel_v = level_u - 1$ ，故  $tenacity(v, u) = tenacity(u)$ 。由上一段的说明知点  $u$  一定在一个  $tenacity$  为  $2k + 1$  的增广路或花中。如果  $u$  在花中，那么其另一  $level$  能被正确计算；否则  $u$  一定会被删除，没有影响。□

**引理 3.7.** 每一次 BLOSS-AUG 中被标记了  $left$  或  $right$  的点在这一次 BLOSS-AUG 结束后要么在花中，要么被标记为删除。

证明. 如果这次 BLOSS-AUG 找到了一个花，那么所有被标记了  $left$  或  $right$  的点一定都在花中。如果找到了一条增广路，对于一个被标记了  $left$  或  $right$  的点  $u$ ，如果  $u$  在增广路中，那么  $u$  一定会被删除。否则，不妨假设  $u$  被标记了  $left$ ，那么 DDFS 一定在某个时刻从  $u$  回溯到了  $par_u$ 。这说明所有从未匹配点到  $u$  的路径都要经过  $d_{cv}$ 。而  $d_{cv}$  一定在增广路中，因此  $d_{cv}$  会被删除，这进一步导致  $u$  被删除。□

由以上引理可知，每个点的  $evenlevel$  和  $oddlevel$  都能够被正确计算，每一个  $tenacity \leq L$  的桥都能被找到，因此该算法一定能找到一组极大的长度为  $L$  的增广路集合。

在 BFS 过程中，每个点至多被访问两次，每条边至多被访问四次。在 BLOSS-AUG 过程中，每条边只会被访问一次，一个点一旦被标记过  $left$  或  $right$  后就不会再被访问。而还原增广路只需访问增广路上的所有点。因此，可以得到

**定理 3.2.** 该算法能够在  $O(n + m)$  的时间复杂度内找到一组极大的最短增广路集合。



再结合定理 3.1, 不断调用该算法, 直到没有增广路, 调用的轮数是  $O(\sqrt{n})$  的, 因此

**定理 3.3.** 一般图最大匹配可以在  $O(m\sqrt{n})$  的时间复杂度内计算出来。

附录A.2展示了该算法的完整伪代码。

## 4 图的最大点权匹配

接下来本文将讨论图的最大点权匹配问题。现在先假设所有点的点权均是非负的, 本节的第三部分将说明处理负点权的方法。

首先给出一些点权非负时的结论。

**引理 4.1.** 一定存在一种最大点权匹配, 使得它同时也是最大匹配。

证明. 假设不成立, 取出一个大小最大的最大点权匹配  $M$ , 再任取一个最大匹配  $M'$ , 考虑子图  $\tilde{G} = (V, M \oplus M')$ , 其中一定存在一个连通块  $C$ , 满足  $C$  是一条简单路径, 上面的边交替地属于  $M'$  和  $M$ , 且开头和结尾都属于  $M'$ 。因此  $C$  是一条关于  $M$  的增广路, 令  $M \leftarrow M \oplus C$ , 匹配的大小增加, 而点权之和不降, 矛盾!  $\square$

**引理 4.2.**  $G$  的一个最大匹配  $M$  是最大点权匹配的充要条件是不存在连接未匹配点  $u$  和匹配点  $v$  的长度为偶数的交错路满足  $w_u > w_v$ 。

证明. 充分性: 假设  $M$  不是最大点权匹配, 任取一个最大点权匹配  $M'$ , 考虑子图  $\tilde{G} = (V, M \oplus M')$ 。  $\tilde{G}$  中一定存在连通块  $C$  满足  $C$  是一条长度为偶数的简单路径, 上面的边交替地属于  $M$  和  $M'$ 。设  $C$  的两端是  $u$  和  $v$ , 则  $u, v$  恰好一个属于  $M$ , 一个属于  $M'$ , 不妨设  $v$  属于  $M$ , 记  $\delta(C) = w_u - w_v$ 。由于  $M'$  的权值大于  $M$  的权值, 一定存在一个  $C$  满足  $\delta(C) > 0$ 。于是就有一条连接未匹配点  $u$  和匹配点  $v$  的长度为偶数的交错路满足  $w_u > w_v$ , 矛盾!

必要性: 假设存在一条这样的交错路, 那么沿着该交错路进行调整就可以使得  $u$  在匹配中,  $v$  不在匹配中, 点权之和变大, 矛盾!  $\square$

定义点集  $V$  中的序关系:  $u < v$  当且仅当  $w_u > w_v$  或  $w_u = w_v$  且  $u < v$ 。

定义序关系  $<$  下的字典序: 对于大小相等的集合  $S, T$ ,  $S$  的字典序小于  $T$  当且仅当  $S \oplus T$  在  $<$  下的最小元素属于  $S$ 。定义两个匹配的字典序为它们所覆盖的点集的字典序。

**引理 4.3.** 在所有最大匹配中, 字典序最小的最大匹配一定是最大点权匹配。

证明. 设  $M$  为字典序最小的最大匹配。任取一个最大点权匹配  $M'$ , 如果  $M = M'$ , 那么  $M$  就是最大点权匹配。否则, 设  $M \oplus M'$  在  $<$  下的最小元素为  $u$ 。那么  $u$  在  $M$  中, 且一定存在一条长度为偶数的简单路径以  $u$  为起点, 且上面的边交替地属于  $M$  和  $M'$ , 并且其结尾  $v$  不在  $M$  中。故一定  $u < v$ 。

如果  $w_u > w_v$ , 那么将  $M'$  沿着该路径进行调整, 可以使得  $u$  在  $M'$  中而  $v$  不在  $M'$  中,  $M'$  的权值严格增加, 矛盾! 因此一定  $w_u = w_v$ , 还是将  $M'$  沿着该路径进行调整,  $M'$  依旧是最大点权匹配, 并且  $M \oplus M'$  中在  $<$  下的最小元素严格变大。不断如此调整, 最终能使得  $M = M'$ , 因此  $M$  是最大点权匹配。  $\square$

接下来先解决二分图的最大点权匹配。

## 4.1 二分图最大点权匹配

**引理 4.4.** 在二分图  $G = (V_1, V_2, E)$  中, 任意两组最大匹配  $M_1, M_2$ , 设  $V'_1$  为所有  $V_1$  中被  $M_1$  覆盖的点,  $V'_2$  为所有  $V_2$  中被  $M_2$  覆盖的点, 则一定存在一个最大匹配恰好覆盖  $V'_1 \cup V'_2$ 。

证明. 按如下方法构造一个匹配  $M$ : 首先令  $M = M_1 \cap M_2$ 。接下来考虑子图  $\bar{G} = (V_1, V_2, M_1 \oplus M_2)$ , 其中的连通块要么是孤立点, 要么是长度为偶数的环, 要么是长度为偶数的路径。

对于一个环  $C$ , 上面的边交替地属于  $M_1$  和  $M_2$ , 则  $M_1 \cap C$  和  $M_2 \cap C$  都可以覆盖  $C$  中的所有点, 不妨将  $M_1 \cap C$  加入  $M$ 。

对于一个路径  $C$ , 上面的边也交替地属于  $M_1$  和  $M_2$  且长度一定是偶数, 因此点数是奇数。如果  $C$  中属于  $V_1$  的点更多, 则将  $M_1 \cap C$  加入  $M$ , 可以发现这样  $C$  中唯一无法被  $M$  覆盖的点属于  $V_1$  且不被  $M_1$  覆盖; 如果  $C$  中属于  $V_2$  的点更多, 则将  $M_2 \cap C$  加入  $M$ , 原因也是类似的。  $\square$

由引理 4.4, 二分图的最大点权匹配关于左部点和右部点是独立的。可以先假设右部点的点权全部为 0, 求出此时的最大点权匹配; 再假设左部点的点权全部为 0, 求出此时的最大点权匹配。之后通过引理 4.4 的证明过程还原出真实的最大点权匹配即可。

因此接下来就讨论只有左部点带权的情况。

对于一个  $S \subseteq V$ , 称  $S$  为可匹配集当且仅当存在一个匹配能够覆盖  $S$  中的点。由引理 4.3, 只需求出左部点中字典序最小的最大可匹配集即可。根据 [2] 中的结论, 如果能在  $T(n, m)$  的时间复杂度内计算出  $n$  个点  $m$  条边的二分图的最大匹配, 那么就能在  $T(n, m) \log n$  的时间复杂度内计算出其左部点中字典序最小的最大可匹配集。

Hopcroft 与 Karp [3] 已经给出了时间复杂度为  $O(m\sqrt{n})$  的二分图最大匹配算法, 因此

**定理 4.1.** 二分图最大点权匹配能够在  $O(m\sqrt{n} \log n)$  的时间复杂度内计算出来。

## 4.2 一般图最大点权匹配

首先求出原图的一个最大匹配  $M$ , 然后不断地找到图中  $tenacity < +\infty$  的花  $B$ , 将  $B$  中的点全部合并到  $base(B)$  上。这一步能通过上一节算法中的 BFS 过程实现, 或者也可以通过熟知的带花树算法 [4] 解决。设处理完之后形成的图为  $G^* = (V^*, E^*)$ , 则  $V^*$  由所有不在任何花中的点构成, 原图中的边  $(u, v)$  在  $G^*$  中就变成  $(base^*(u), base^*(v))$ 。



对于  $V^*$  中的点  $u$ , 定义其权值  $w_u^* = \min_{base^*(v)=u} w_v$ 。

**引理 4.5.** 对于任何  $G^*$  的匹配  $\tilde{M}^*$ , 可以将其对应成  $G$  的匹配  $\tilde{M}$ , 满足对于每个  $(u, v) \in \tilde{M}$ , 要么  $base^*(u) = base^*(v)$ , 要么  $(base^*(u), base^*(v)) \in \tilde{M}^*$ ; 对于每个  $(u, v) \in \tilde{M}^*$ , 一定存在一个  $(u', v') \in \tilde{M}$  满足  $base^*(u') = u, base^*(v') = v$ 。并且  $\tilde{M}$  的权值为  $\sum_{u \in V} w_u - \sum_{u \in V^*} w_u^* + (\tilde{M}^* \text{ 的权值})$ 。

证明. 对于  $V^*$  中的一个点  $u$ , 记  $B = \{v : base^*(v) = u\}$ 。如果  $u$  未被  $\tilde{M}^*$  覆盖, 那么对于每个  $B$  中的点  $v$ , 都可以通过依次展开合并到  $u$  上的花使得恰好  $B \setminus \{v\}$  中的点之间构成完美匹配, 可以让  $v$  是  $B$  中点权最小的。反之, 如果  $u$  被  $\tilde{M}^*$  覆盖, 那么设该匹配边  $e$  在原图中的端点是  $v$ , 也可以使得  $B \setminus \{v\}$  中的点之间构成完美匹配, 而  $v$  被  $e$  覆盖。故  $\tilde{M}$  的权值是  $\sum_{u \in V} w_u - \sum_{u \in V^*} w_u^* + (\tilde{M}^* \text{ 的权值})$ 。□

下文将把  $\tilde{M}$  与  $\tilde{M}^*$  称为对应的匹配,  $(u, v)$  与  $(base^*(u), base^*(v))$  称为对应的边。特别地, 令当前的最大匹配  $M$  对应的匹配为  $M^*$ 。

对于  $V^*$  中的每个点  $u$ , 如果存在从未匹配点到  $u$  的长度为偶数的交错路, 则将  $u$  加入集合  $S$ ; 如果存在从未匹配点到  $u$  的长度为奇数的交错路, 则将  $u$  加入集合  $T$ 。

**引理 4.6.** 集合  $S$  与集合  $T$  的交为空。

证明. 假设存在  $u \in S \cap T$ , 这说明  $tenacity(u) < +\infty$ , 因此由引理 3.4,  $u$  一定在某个增广路或花中。但由于  $M$  为最大匹配, 不存在增广路, 因此  $u$  一定在某个花中, 这与  $u \in V^*$  矛盾! □

**引理 4.7.** 对于每个  $T$  中的点  $u$ ,  $u$  不是任何  $tenacity(B) < +\infty$  的花  $B$  的花托。

证明. 假设  $u = base(B)$  且  $tenacity(B) < +\infty$ , 那么存在从未匹配点到  $u$  的长度为偶数的交错路, 与  $u \in T$  矛盾! □

接下来, 构造二分图  $G' = (S, T, E')$ , 其中  $E'$  由  $E^*$  中所有一端在  $S$  中, 一端在  $T$  中的边构成。对于一条  $G$  中的交错路  $P = (a_1, a_2, \dots, a_k)$ , 定义  $b(P)$  为  $(base^*(a_1), \dots, base^*(a_k))$  将相邻的相同点合并后的路径。

**引理 4.8.** 对于  $M^*$  中的边  $(u, v)$ , 如果  $u, v \in S \cup T$ , 那么  $(u, v) \in E'$ 。

证明. 假设  $(u, v) \in M^*$  不在  $E'$  中。如果  $u, v \in T$ , 那么  $(u, v)$  在原图中的对应边也是  $(u, v)$ , 且存在从未匹配点到  $u$  和  $v$  的长度为奇数的交错路, 故  $(u, v)$  是桥, 由引理 3.4  $(u, v)$  在某个增广路或花中, 矛盾! 如果  $u, v \in S$ , 由于花中的点一定与同一个花中另一个点匹配, 故  $(u, v)$  在原图中的对应边也是  $(u, v)$ , 从未匹配点到  $u$  的长度为偶数的交错路一定经过  $(u, v)$ , 因此一定存在从未匹配点到  $v$  的长度为奇数的交错路, 与  $v \in S$  矛盾! □

由引理 4.8,  $M^*$  在  $S \cup T$  中的部分是  $G'$  的匹配。

**引理 4.9.** 若  $\tilde{M}^*$  为  $G' = (S, T, E')$  的一个最大匹配, 设其对应的  $G$  中的匹配为  $\tilde{M}$ , 则对于任何  $G$  中关于  $\tilde{M}$  的交错路  $P$ ,  $b(P)$  都是  $G'$  中关于  $\tilde{M}^*$  的交错路。

证明. 首证明所有未被  $\tilde{M}$  覆盖的点都在  $S$  中。这是由于  $T$  中的点都不是未匹配点, 故  $\tilde{M}^*$  覆盖了  $T$  中所有点, 因此  $G'$  的最大匹配的大小就是  $|T|$ , 所以  $\tilde{M}^*$  也一定覆盖了  $T$  中所有点, 即  $\tilde{M}$  覆盖了  $T$  中所有点。

现在设  $P$  从未匹配点  $s$  出发, 那么  $base^*(s) \in S$ 。假设  $b(P)$  不是  $G'$  中关于  $\tilde{M}^*$  的交错路, 那要么  $b(P)$  中有连续两条非匹配边, 要么  $b(P)$  中的某条边不在  $E'$  中。令  $e$  为最靠前的不符合条件的边: 第一种情况中,  $e$  为第二条非匹配边; 第二种情况中  $e$  为不在  $E'$  中的边。令  $u$  为  $e$  的起点。

如果  $u \in T$ , 由引理 4.7,  $b(P)$  中与  $u$  相邻的两条边在  $P$  中的对应边也是相邻的。因此  $e$  一定是匹配边且  $e$  的终点在  $S$  中, 与以上两种情况均矛盾!

如果  $u \in S$ , 那么  $u$  在  $b(P)$  中的前一条边如果存在的话一定在  $\tilde{M}^*$  中, 因此  $u$  在  $b(P)$  中不可能与两条非匹配边相邻。那么只有可能  $e$  不在  $E'$  中, 但这说明  $e$  的终点  $v$  也在  $S$  中, 故  $(u, v)$  在  $G$  中的对应边是桥, 由引理 3.4 它在某个增广路或花中, 矛盾!

综上,  $b(P)$  一定是  $G'$  中关于  $\tilde{M}^*$  的交错路。  $\square$

对于一些  $V^*$  中的点  $u$ ,  $u$  即不在  $S$  中也不在  $T$  中, 那么这些  $u$  之间有完美匹配, 设它们在原图中的完美匹配为  $M_R$ 。

**引理 4.10.** 设  $\tilde{M}^*$  为二分图  $G' = (S, T, E')$  的最大点权匹配, 其在  $G$  中对应的匹配是  $\tilde{M}$ , 则  $\tilde{M} \cup M_R$  是  $G$  的最大点权匹配。

证明. 只用说明  $\tilde{M} \cup M_R$  满足引理 4.2 的条件。考虑一条连接  $u$  和  $v$  的交错路  $P$ , 其中  $u$  为未匹配点,  $v$  为匹配点。由于不存在从未匹配点向  $M_R$  所覆盖的点的交错路,  $u$  和  $v$  一定都属于  $S \cup T$ 。由引理 4.9,  $b(P)$  是  $G'$  中关于  $\tilde{M}^*$  的交错路。那么  $base^*(u)$  在  $G'$  中是未匹配点,  $base^*(v)$  在  $G'$  中是匹配点。由引理 4.5 求解  $\tilde{M}$  的过程可知  $w_{base^*(u)}^* = w_u$ 。并且由于  $\tilde{M}^*$  是  $G'$  的最大点权匹配, 一定有  $w_{base^*(u)}^* \leq w_{base^*(v)}^*$ 。故  $w_u = w_{base^*(u)}^* \leq w_{base^*(v)}^* \leq w_v$ 。  $\square$

由引理 4.10, 求解  $G$  的最大点权匹配时可以先求  $G$  的最大匹配, 然后求出二分图  $G'$ , 之后调用上一部分的二分图最大点权匹配算法求出  $G'$  的最大点权匹配  $M^*$ , 然后通过引理 4.5 的过程求出  $G$  的最大点权匹配  $M$ 。而一般图最大匹配可以在  $O(m\sqrt{n})$  的时间复杂度内计算出, 二分图最大点权匹配能够在  $O(m\sqrt{n}\log n)$  的时间复杂度内计算出, 因此

**定理 4.2.** 一般图最大点权匹配能够在  $O(m\sqrt{n}\log n)$  的时间复杂度内计算出来。

### 4.3 有负点权的情况

对于每个点权为负的点  $u$ , 假设  $w_u = -a$ , 其中  $a > 0$ , 那么只需新建点  $u'$  连向  $u$ , 令  $u'$  的点权为  $a$ ,  $u$  的点权变为 0。如果  $u$  和其他点匹配, 那么  $u'$  就不在匹配中, 否则  $u'$  一定和  $u$  匹配。因此修改后的图的最大点权匹配恰好比原图的最大点权匹配大  $a$ 。





该变换在点数至多翻倍的代价下将有负点权的情况转化为点权非负的情况。

## 5 总结

本文主要介绍了一种较为高效的一般图最大匹配算法以及一般图最大点权匹配算法。由这两个算法的复杂度以及过程可以看出，图的最大点权匹配与图的最大匹配几乎是等难的。

在信息学竞赛中，题目往往不会直接输入一张图来求最大点权匹配，而是通过一些特殊的性质将图给出，图中的边数通常很大。此时需要具体分析图的性质，灵活运用各种结论或方法。

事实上，本文中的许多引理以及证明过程中所体现的思想都能广泛应用于这些题目中，有些性质由于与本文所介绍的算法关系不大因此未在正文中提及，例如如下结论就可直接类似引理 4.3 的证明过程得到：

**性质 5.1.** 令  $V'$  为点集  $V$  的子集， $S$  为  $V'$  中字典序最小的最大可匹配集， $S$  中的元素为  $a_1 < \dots < a_k$ 。设  $T$  为  $V'$  中任意可匹配集， $T$  中的元素为  $b_1 < \dots < b_l$ ，则  $a_i \leq b_i (i = 1, \dots, l)$ 。

限于篇幅本文不再用具体题目举例说明。

希望本文能吸引感兴趣的读者研究和发展相关理论。

## 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢教练金靖老师对我的指导。

感谢父母对我的培育和教诲。

感谢所有帮助我的同学、老师。

## 参考文献

- [1] 陈孙立，《浅谈支配树及其应用》，IOI2020 中国国家集训队论文
- [2] 柯绎思，《浅谈一些二分图匹配相关问题》，IOI2023 中国国家集训队论文
- [3] Hopcroft, J. E., Karp, R. M. An  $n^{5/2}$  Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. on Comput.* 2 (1973), pp. 225-231.
- [4] Wikipedia, Blossom algorithm, [https://en.wikipedia.org/wiki/Blossom\\_algorithm/](https://en.wikipedia.org/wiki/Blossom_algorithm/)

- [5] Micali, S., and Vazirani, V. (1980), An  $O(\sqrt{|V|}|E|)$  algorithm for finding maximum matching in general graphs, *Proc. 21st Ann. Symp. on Foundations of Computer Science*, IEEE, pp. 17-27.
- [6] Spencer, T. H., and Mayr, E. W. Node weighted matching. In *Proceedings of the 11th Colloquium on Automata, Languages and Programming* (London, UK, 1984), Springer-Verlag, pp. 454-464.

## 附录 A

### A.1

---

**Algorithm 1** DDFS

---

**Input:**  $s, t$ 

```
1: $x \leftarrow s; y \leftarrow t$
2: $barrier_x \leftarrow s; barrier_y \leftarrow t$
3: $blossom \leftarrow \text{false}$
4: while $level_x > 0$ or $level_y > 0$ do
5: if $level_x \geq level_y$ then
6: $flag \leftarrow \text{false}$
7: for (z, x) 为 BFS 图上未使用过的边且 z 未删除 do
8: 把 (z, x) 标记为使用过
9: if z 未被标记过 left 或 right then
10: 把 z 标记为 left
11: $par_z \leftarrow x; x \leftarrow z; flag \leftarrow \text{true}$
12: break
13: else if $z = y$ then
14: $dcv \leftarrow y; lstx \leftarrow x$
15: end if
16: end for
17: if $flag = \text{true}$ then
18: continue
19: else if $x = barrier_x$ then
20: if $y = barrier_y$ then
21: $blossom \leftarrow \text{true}$
22: break
23: else
24: $y \leftarrow par_y$
25: $x \leftarrow dcv; barrier_x \leftarrow dcv$
26: $par_x \leftarrow lstx$
27: 把 x 标记为 left
28: end if
29: else
30: $x \leftarrow par_x$
31: end if
```

```
32: else
33: $flag \leftarrow \text{false}$
34: for (z, y) 为 BFS 图上未使用过的边且 z 未删除 do
35: 把 (z, y) 标记为使用过
36: if z 未被标记过 left 或 right then
37: 把 z 标记为 right
38: $par_z \leftarrow y; y \leftarrow z; flag \leftarrow \text{true}$
39: break
40: else if $z = x$ then
41: $dcv \leftarrow x; lsty \leftarrow y$
42: end if
43: end for
44: if $flag = \text{true}$ then
45: continue
46: else if $y = \text{barriery}$ then
47: if $x = \text{barrierx}$ then
48: $blossom \leftarrow \text{true}$
49: break
50: else
51: $x \leftarrow par_x$
52: $y \leftarrow dcv; \text{barriery} \leftarrow dcv$
53: $par_y \leftarrow lsty$
54: 把 y 标记为 right
55: end if
56: else
57: $y \leftarrow par_y$
58: end if
59: end if
60: end while
```

## A.2

### Algorithm 2 一般图最大匹配

```
1: function MaximumMatching($G = (V, E)$)
2: for u in V do
3: $mch_u \leftarrow \text{undefined}$
4: end for
5: repeat
6: {初始化}
7: 清空 BFS 图中所有边
8: for u in V do
9: $\text{evenlevel}_u \leftarrow +\infty; \text{oddlevel}_u \leftarrow +\infty$
10: $\text{blo}_u \leftarrow \text{undefined}; \text{deg}_u \leftarrow 0$
11: $\text{anomaly}(u) \leftarrow \text{empty}$
12: 清除 u 上所有标记
13: end for
14: for i from 0 to $|V| - 1$ do
```

```
15: $que(i) \leftarrow \text{empty}$ {在第 i 层 BFS 需要被遍历的点}
16: $bridge(i) \leftarrow \text{empty}$
17: end for
18: {开始算法}
19: call BFS(G)
20: until 无法找到增广路
21: end function
22:
23: function BFS($G = (V, E)$)
24: for u in V do
25: if $mch_u = \text{undefined}$ then
26: $evenlevel_u \leftarrow 0; level_v \leftarrow 0$
27: 把点 u 加入 $que(0)$
28: end if
29: end for
30: for i from 0 to $|V| - 1$ do
31: if i 为奇数 then
32: for u in $que(i)$ 且 $blo_u = \text{undefined}$ do
33: $v \leftarrow mch_u$
34: if $oddlevel_v < +\infty$ then
35: 把 (u, v) 加入 $bridge(oddlevel_u + oddlevel_v + 1)$
36: else
37: $evenlevel_v \leftarrow i + 1; level_v \leftarrow i + 1$
38: 把 v 加入 $que(i + 1)$
39: 把 (u, v) 加入 BFS 图; $deg_v \leftarrow deg_v + 1$
40: end if
41: end for
42: else
43: for u in $que(i)$ do
44: for (u, v) in E 且 (u, v) 不是匹配边且点 v 未删除 do
45: if $evenlevel_v < +\infty$ then
46: 把 (u, v) 加入 $bridge(evenlevel_u + evenlevel_v + 1)$
47: else
48: if $oddlevel_v = +\infty$ then
49: $oddlevel_v \leftarrow i + 1; level_v \leftarrow i + 1$
50: 把 v 加入 $que(i + 1)$
51: end if
52: if $oddlevel_v = i + 1$ then
53: 把 (u, v) 加入 BFS 图; $deg_v \leftarrow deg_v + 1$
54: else
55: 把 u 加入 $anomaly(v)$
56: end if
57: end if
58: end for
59: end for
60: end if
61: for (u, v) in $bridge(2i + 1)$ do
```

```
62: call BLOSS-AUG(u, v, i)
63: end for
64: if 发现增广路 then
65: break
66: end if
67: end for
68: end function
69:
70: function BLOSS-AUG(s, t, k)
71: if s 或 t 已被删除 then
72: return
73: end if
74: $x \leftarrow base^*(s); y \leftarrow base^*(t)$
75: if $x = y$ then
76: return
77: end if
78: {初始化}
79: 把 x 标记为 left; 把 y 标记为 right
80: $par_x \leftarrow undefined; par_y \leftarrow undefined$
81: $top_x \leftarrow s; top_y \leftarrow t$
82: $dcv \leftarrow undefined; barrier_x \leftarrow x; barrier_y \leftarrow y$
83: blossom $\leftarrow false$
84: {Double DFS}
85: while $level_x > 0$ or $level_y > 0$ do
86: flag $\leftarrow false$
87: if $level_x \geq level_y$ then
88: for (v, x) 为 BFS 图上未使用过的边且 v 未删除 do
89: 把 (v, x) 标记为使用过
90: $u \leftarrow base^*(v); dw_x \leftarrow v$
91: if u 未被标记过 left 或 right then
92: 把 u 标记为 left
93: $par_u \leftarrow x; top_u \leftarrow v; x \leftarrow u; flag \leftarrow true$
94: break
95: else if $u = y$ then
96: $dcv \leftarrow y; lstx \leftarrow x; topx \leftarrow v$
97: end if
98: end for
99: if flag = true then
100: continue
101: else if $x = barrier_x$ then
102: if $y = barrier_y$ then
103: blossom $\leftarrow true$
104: break
105: else
106: $y \leftarrow par_y; x \leftarrow dcv; barrier_x \leftarrow dcv$
107: $par_x \leftarrow lstx; top_x \leftarrow topx$
108: 把 x 标记为 left
```

```
109: end if
110: else
111: $x \leftarrow par_x$
112: end if
113: else
114: for (v, y) 为 BFS 图上未使用过的边且 v 未删除 do
115: 把 (v, y) 标记为使用过
116: $u \leftarrow base^*(v); dw_y \leftarrow v$
117: if u 未被标记过 left 或 right then
118: 把 u 标记为 right
119: $par_u \leftarrow y; top_u \leftarrow v; y \leftarrow u; flag \leftarrow true$
120: break
121: else if $u = x$ then
122: $d_{cv} \leftarrow x; lst_y \leftarrow y; top_y \leftarrow v$
123: end if
124: end for
125: if $flag = true$ then
126: continue
127: else if $y = barrier_y$ then
128: if $x = barrier_x$ then
129: blossom $\leftarrow true$
130: break
131: else
132: $x \leftarrow par_x; y \leftarrow d_{cv}; barrier_y \leftarrow d_{cv}$
133: $par_y \leftarrow lst_y; top_y \leftarrow top_y$
134: 把 y 标记为 right
135: end if
136: else
137: $y \leftarrow par_y$
138: end if
139: end if
140: end while
141: if blossom = true then
142: {找到一个花}
143: $b \leftarrow d_{cv}$; 清除 b 上的 left 或 right 标记
144: 令 B 为所有在这一轮中被标记了 left 或 right 的点构成的集合
145: $base(B) \leftarrow b$
146: for u in B do
147: if BFS 图中存在边 (v, u) 使得 $base^*(v) = b$ then
148: if u 被标记了 left then
149: $parL(B) \leftarrow u; topL(B) \leftarrow v$
150: else
151: $parR(B) \leftarrow u; topR(B) \leftarrow v$
152: end if
153: end if
154: end for
155: if $base^*(s) = b$ then
```



```
156: $parL(B) \leftarrow \text{undefined}; topL(B) \leftarrow s$
157: end if
158: if $base^*(t) = b$ then
159: $parR(B) \leftarrow \text{undefined}; topR(B) \leftarrow t$
160: end if
161: for u in B do
162: $blo_u \leftarrow B$
163: if u 为外部点 then
164: $oddlevel_u \leftarrow 2k + 1 - evenlevel_u$
165: else
166: $evenlevel_u \leftarrow 2k + 1 - oddlevel_u$
167: 把 u 加入 $que(evenlevel_u)$
168: for v in $anomaly(u)$ do
169: 把 (u, v) 加入 $bridge(evenlevel_u + evenlevel_v + 1)$
170: end for
171: end if
172: end for
173: else
174: {找到增广路}
175: $Path \leftarrow []; A \leftarrow []$
176: $u \leftarrow x$
177: while $u \neq \text{undefined}$ do
178: $A \leftarrow A + \text{JMP}(u, top_u)$
179: $u \leftarrow par_u$
180: end while
181: $Path \leftarrow \text{FINDPATH}(A, \text{undefined}); A \leftarrow []$
182: $u \leftarrow y$
183: while $u \neq \text{undefined}$ do
184: $A \leftarrow A + \text{JMP}(u, top_u)$
185: $u \leftarrow par_u$
186: end while
187: $Path \leftarrow Path + \text{reverse}(\text{FINDPATH}(A, \text{undefined}))$
188: 沿着 $Path$ 进行增广
189: for u in $Path$ do
190: if u 未被删除 then
191: call $\text{ERASE}(u)$
192: end if
193: end for
194: end if
195: end function
196:
197: function $\text{JMP}(u, v)$
198: {返回序列 $a_1, a_2, \dots, a_n$ 满足 $a_1 = u, a_n = v$ 且 $a_{i-1} = base(blo_{a_i})$ }
199: $A \leftarrow []$
200: while $v \neq u$ do
201: $A \leftarrow [v] + A$
202: $v \leftarrow base(blo_v)$
```

```
203: end while
204: $A \leftarrow [u] + A$
205: return A
206: end function
207:
208: function FINDPATH(A, B)
209: $Path \leftarrow []$
210: for u in A do
211: if $blo_u \neq B$ then
212: $Path \leftarrow Path + OPEN(u)$
213: else
214: $Path \leftarrow Path + [u]$
215: end if
216: end for
217: return $Path$
218: end function
219:
220: function OPEN(u)
221: $B \leftarrow blo_u$
222: $A \leftarrow []$; $Path \leftarrow []$
223: if u 为外部点 then
224: while $u \neq base(B)$ do
225: if $blo_u \neq B$ then
226: $Path \leftarrow OPEN(u) + Path$
227: $u \leftarrow base(blo_u)$
228: else
229: $Path \leftarrow [u] + Path$
230: $u \leftarrow dw_u$
231: end if
232: end while
233: else
234: $mark \leftarrow$ 点 u 上的标记
235: while $u \neq undefined$ do
236: $A \leftarrow A + JMP(u, top_u)$
237: $u \leftarrow par_u$
238: end while
239: $Path \leftarrow FINDPATH(A, B)$; $A \leftarrow []$
240: $u \leftarrow base(B)$
241: if $mark = left$ then
242: $A \leftarrow JMP(u, topR(B))$
243: $u \leftarrow parR(B)$
244: else
245: $A \leftarrow JMP(u, topL(B))$
246: $u \leftarrow parL(B)$
247: end if
248: while $u \neq undefined$ do
249: $A \leftarrow A + JMP(u, top_u)$
```



```
250: $u \leftarrow par_u$
251: end while
252: 删掉 A 开头的 $base(B)$
253: $Path \leftarrow \text{FINDPATH}(A, B) + \text{reverse}(Path)$
254: end if
255: return $Path$
256: end function
257:
258: function ERASE(u)
259: 把 u 标记为删除
260: for BFS 图上所有出边 (u, v) do
261: $deg_v \leftarrow deg_v - 1$
262: if $deg_v = 0$ then
263: call ERASE(v)
264: end if
265: end for
266: end function
```



# 浅谈再谈格路计数

北京市十一学校 吴畅

## 1 前言

计数是算法竞赛中的一大重要板块，而格路计数是其中较为常见的模型之一。在计算机科学界和数学界，格路计数也得到了广泛的研究。

笔者注意到，目前大多数选手对格路计数的了解，仍只停留在反射容斥和 Dyck 路上；另一方面，也鲜有比较复杂的格路计数题目。事实上，许多相关结论和算法是非常适合放入算法竞赛中的，笔者的集训队互测题就是一次尝试。为此，笔者撰写本论文，旨在向读者展示格路计数更广阔、更有前景的一面。

本文主要分为不相交格路、边界限制、特征限制三章，基本涵盖了算法竞赛中较为常见的格路计数模型，将一些原本零碎的结论进行了统一。另外，相比于代数推导，本文更注重组合上的一些较为直观的方法，这也是算法竞赛更常考察的。

## 2 基本定义与约定

**定义 2.1** (步长, 步长集合). 称  $\mathbb{Z}^d$  上的一个向量为一种步长，若干步长构成的集合为一个步长集合。

**定义 2.2** (格路). 考虑  $\mathbb{Z}^d$  上的序列  $P = (P_0, P_1, \dots, P_l)$ ，以及步长集合  $\mathbb{S}$ ，若满足  $\forall 0 \leq i < l, \overrightarrow{P_i P_{i+1}} \in \mathbb{S}$ ，则称  $P$  是步长集合  $\mathbb{S}$  下的一条长度为  $l$  的格路。

**定义 2.3** (简单步, 简单格路). 若  $\mathbb{Z}^d$  上的步长  $s = (s_1, s_2, \dots, s_d)$  满足  $\forall 1 \leq i \leq d, s_i \in \{0, 1\}$  且  $\sum_{i=1}^d s_i = 1$ ，则称  $s$  是简单步。步长集合由  $\mathbb{Z}^d$  上所有简单步构成的格路，称为简单格路。

为了方便，将从  $A$  到  $E$ ，长度为  $m$ 、步长集合为  $\mathbb{S}$ 、约束集合为  $R$  的所有格路构成的集合表示为以下记号

$$L_m(A \rightarrow E; \mathbb{S} | R)$$

有时该记号中的若干部分会被省略，请留意上下文中的陈述。通常来讲，简单格路会省略步长集合和长度，自由（无限制）格路会省略约束集合。例如  $L((0, 0) \rightarrow (1, 1) | x \geq y) = \{((0, 0), (1, 0), (1, 1))\}$ 。

另外，若  $A$  和  $E$  是两个长度相同的格点序列，那么对应的记号表示的是每对起点终点之间的路径形成的路径组的集合。

当我们想带权计数时，使用记号

$$GF(\mathcal{M}; w) := \sum_{x \in \mathcal{M}} w(x)$$

其中  $w$  是定义在集合  $\mathcal{M}$  上的函数。通常情况下  $\mathcal{M}$  是路径或路径组的集合，而  $w$  同时定义在边、路径和路径组上：路径的权值为所有边权的乘积；路径组的权值为所有路径权值的乘积。

### 3 不相交格路

对于有向无环图  $G$  上的两条路径  $P = (P_0, P_1, \dots, P_{l_P})$  和  $Q = (Q_0, Q_1, \dots, Q_{l_Q})$ , 若满足  $\forall 0 \leq i \leq l_P, 0 \leq j \leq l_Q, P_i \neq Q_j$ , 则称  $P$  和  $Q$  不相交。

若简单格路组成的  $r$  元组  $(P_1, P_2, \dots, P_r)$  两两不相交, 则称这是一组不相交格路。反之, 称其为一组相交格路。

本节将会介绍几种方法, 用于解决不相交格路相关的若干计数问题。事实上, 这些方法大多可以在添加一些限制或减弱一些要求后, 扩展到更一般的情形中, 而在不相交格路中会被更自然的满足。

#### 3.1 行列式与 Pfaffian

首先介绍一些本章中涉及的线性代数工具, 主要围绕行列式与 Pfaffian 展开。

为此, 先明确一些记号: 记  $\mathfrak{S}_n$  表示全体长度为  $n$  的排列构成的集合。对序列  $\mathbf{v} = (v_1, v_2, \dots, v_n)$  以及  $\sigma \in \mathfrak{S}_n$ , 令  $\mathbf{v}_\sigma = (v_{\sigma(1)}, v_{\sigma(2)}, \dots, v_{\sigma(n)})$ 。对  $\sigma \in \mathfrak{S}_n$ , 令  $\text{sgn } \sigma$  表示  $(-1)^{\text{inv } \sigma}$ , 其中  $\text{inv } \sigma$  表示  $\sigma$  的逆序对数。

**定义 3.1 (行列式).** 令  $\mathbf{A} = (a_{i,j})_{n \times n}$  是一个方阵, 定义  $\mathbf{A}$  的行列式为

$$\det(\mathbf{A}) = \sum_{\sigma \in \mathfrak{S}_n} (\text{sgn } \sigma) \prod_{i=1}^n a_{i, \sigma(i)}$$

由于行列式是较为常用的概念, 关于它的更多信息这里不再赘述。

**定义 3.2 (完美匹配).** 称一个长度为  $2n$  的排列  $\pi \in \mathfrak{S}_{2n}$  是完美匹配, 当且仅当其满足

- $\forall 1 \leq i \leq n, \pi_{2i-1} < \pi_{2i}$
- $\forall 1 \leq i < n, \pi_{2i-1} < \pi_{2i+1}$

对所有  $1 \leq i \leq n$ , 称  $\pi_{2i-1}$  和  $\pi_{2i}$  是一对匹配的元素。

记  $\mathfrak{M}_{2n}$  表示全体长度为  $2n$  的完美匹配构成的集合。对  $\pi \in \mathfrak{M}_{2n}$ , 若  $\pi$  中的匹配  $(w, y)$  和  $(x, z)$  满足  $w < x < y < z$  或  $w > x > y > z$ , 则称它们相交。这里  $\pi$  是一个排列, 因此我们可以如上定义  $\text{inv } \pi$  和  $\text{sgn } \pi$ 。

值得一提的是,  $\text{inv } \pi$  和  $\pi$  中相交的匹配对数量的奇偶性相等, 这是不难验证的。另外, 我们还有一种递归的方法计算  $\text{inv } \pi$  的奇偶性:

取  $\pi$  中任意一对匹配的元素  $(i, j)$ , 将它们删去, 其余元素在下标和值域上填补空缺。可以得到一个  $\pi' \in \mathfrak{M}_{2n-2}$ , 那么  $\text{inv } \pi \equiv \text{inv } \pi' + j - i - 1 \pmod{2}$ 。

**定义 3.3 (反对称矩阵).** 令  $\mathbf{A} = (a_{i,j})_{n \times n}$  是一个方阵, 若  $\mathbf{A}^T = -\mathbf{A}$ , 则称  $\mathbf{A}$  是一个反对称矩阵。

可以发现, 反对称矩阵满足主对角线上为 0, 关于主对角线对称的元素互为相反数。因此, 我们可以只用主对角线上方的元素  $(a_{i,j})_{1 \leq i < j \leq n}$  来描述一个  $n \times n$  的反对称矩阵  $\mathbf{A}$ 。

**定义 3.4 (Pfaffian).** 令  $n$  是偶数,  $\mathbf{A} = (a_{i,j})_{1 \leq i < j \leq n}$  是一个反对称矩阵, 定义  $\mathbf{A}$  的 Pfaffian 为

$$\text{Pf}(\mathbf{A}) = \sum_{\pi \in \mathfrak{M}_n} (\text{sgn } \pi) \prod_{i=1}^{n/2} a_{\pi(2i-1), \pi(2i)}$$

注意到行列式和 Pfaffian 的定义十分相似, 接下来我们将列出一些 Pfaffian 的性质, 说明二者之间的联系。

**命题 3.1.** 假设  $n$  是偶数,  $A$  是大小为  $n$  的反对称矩阵,  $X$  是大小为  $n$  的方阵

(a).  $\text{Pf}(1)_{1 \leq i < j \leq n} = 1$ , 等价地,  $\sum_{\pi \in \mathfrak{M}_n} \text{sgn } \pi = 1$ 。

(b).  $\text{Pf}(X^T A X) = \det(X) \text{Pf}(A)$ 。

(c).  $\text{Pf}(A)^2 = \det(A)$ 。

对于给定的反对称矩阵  $A$ , 为了计算  $\text{Pf}(A)$ , 你可能会考虑用 (c) 转化成求  $\det(A)$ 。可惜在一般交换环上开根号并不是一件方便的事情。而事实上, 我们只需要令 (b) 中的  $X$  是初等行变换矩阵, 就可以得到与  $\det$  类似的消元过程, 更详细的内容此处不再赘述。复杂度同高斯消元。

## 3.2 起点与终点固定

接下来我们将简述广为人知的 LGV 引理。引理中的  $L_G(\dots)$  含义与  $L(\dots)$  基本相同, 唯一的区别是语境不再是格路而是有向无环图  $G$  上的路径。

**引理 3.1 (LGV 引理).** 令  $G = (V, E)$  是一张有向无环图,  $\mathbf{A} = (A_1, A_2, \dots, A_n)$ ,  $\mathbf{E} = (E_1, E_2, \dots, E_n)$  均是节点组成的序列, 即  $\forall 1 \leq i \leq n, A_i, E_i \in V$ 。每条边  $e \in E$  被赋予了边权  $w(e)$ , 一条路径  $P$  的权值为  $w(P) = \prod_{e \in P} w(e)$ , 一个路径组  $\mathbf{P} = (P_1, P_2, \dots, P_n)$  的权值为  $\prod_{i=1}^n w(P_i)$ 。那么我们有

$$\sum_{\sigma \in \mathfrak{S}_n} (\text{sgn } \sigma) GF(L_G(\mathbf{A}_\sigma \rightarrow \mathbf{E} \mid \text{non-intersecting}); w) = \det_{1 \leq i, j \leq n} (GF(L_G(A_j \rightarrow E_i); w))$$

证明. 将等号右边的行列式展开得到

$$\det_{1 \leq i, j \leq n} (GF(L_G(A_j \rightarrow E_i); w)) = \sum_{\substack{(\sigma, \mathbf{P}), \\ \sigma \in \mathfrak{S}_n, \mathbf{P} \in L_G(\mathbf{A}_\sigma \rightarrow \mathbf{E})}} (\text{sgn } \sigma) w(\mathbf{P})$$

接下来, 我们将构造一个对合  $\phi$ , 定义在二元组  $(\sigma, \mathbf{P})$  上, 其中  $\mathbf{P} \in L_G(\mathbf{A}_\sigma \rightarrow \mathbf{E})$  是相交的路径组, 满足符号相反、权值不变。也就是说, 令  $(\sigma_\phi, \mathbf{P}_\phi) = \phi((\sigma, \mathbf{P}))$ , 那么有  $\text{sgn } \sigma_\phi = -\text{sgn } \sigma$  且  $w(\mathbf{P}_\phi) = w(\mathbf{P})$ 。



假如这样的  $\phi$  存在, 将会使所有  $(\sigma, \mathbf{P})$  的贡献 (其中  $\mathbf{P}$  是相交的) 两两抵消。于是只剩下了不交路径组的贡献, 刚好和原式等号左边等价, 立刻证明该引理。现在我们来构造一个这样的对合。

取任一  $G$  的拓扑序  $\mathcal{A}$ 。对  $(\sigma, \mathbf{P})$ , 其中  $\mathbf{P}$  是相交路径组, 我们将每个交点描述为  $(M, i, j)$  的形式, 表示  $P_i$  和  $P_j$  在  $M$  处有一交点。找到所有这样的三元组中字典序最小的一个 (记为  $(M, i, j)$ ), 其中第一维的比较按照  $\mathcal{A}$  上的排名进行。接下来我们交换  $P_i, P_j$  在  $M$  前的子路径, 便得到了需要构造的  $\phi$ 。如图 1 所示。

不难发现, 进行完这个变换后,  $P_i$  和  $P_j$  仍相交于  $A$ 。因此, 我们在  $(\sigma, \mathbf{P})$  上进行两次映射  $\phi$ , 会回到  $(\sigma, \mathbf{P})$ , 也就是说  $\phi$  是一个对合。另一方面, 设变换后的二元组为  $(\sigma_\phi, \mathbf{P}_\phi)$ , 那么可以发现:  $\sigma_\phi$  只是将  $\sigma$  中的第  $i$  个元素和第  $j$  个元素进行了交换, 符号取反; 而由于组成路径的可重边集在变换后不变,  $w(\mathbf{P}_\phi) = w(\mathbf{P})$ 。至此验证了  $\phi$  是满足要求的对合。  $\square$

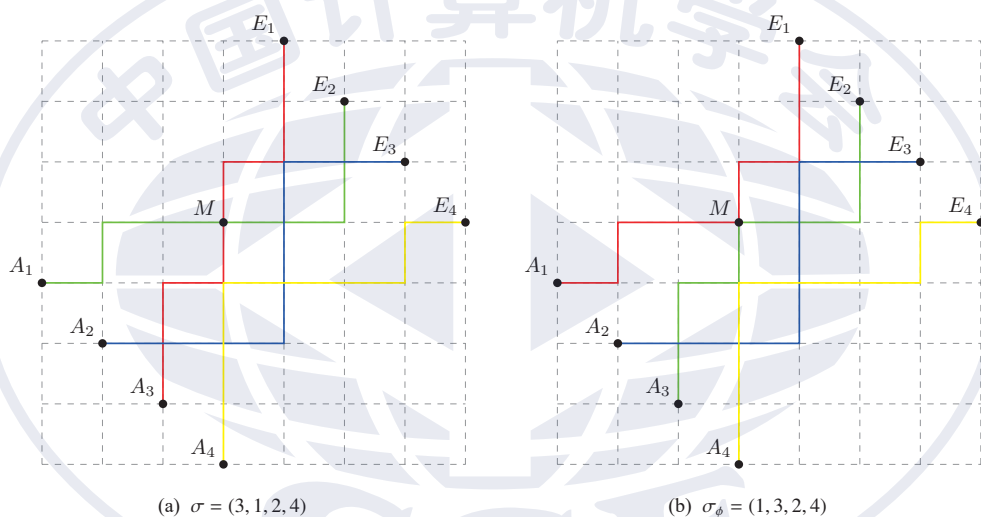


图 1: 格路上的 LGV—— $\phi$  中对应的一对  $(\sigma, \mathbf{P})$

LGV 帮我们抵消了相交格路的贡献, 但是仍只能求出所有不相交格路带符号的求和。

假如 LGV 引理中的有向无环图  $G$  以及其上的顶点集合  $\mathbf{A}, \mathbf{E}$  满足,  $\forall i < j, k < l$ , 任意  $P \in L_G(A_i \rightarrow E_l)$  和  $Q \in L_G(A_j \rightarrow E_k)$  相交, 那么称  $\mathbf{A}$  和  $\mathbf{E}$  是  $G$ -协调的。

可以发现, 如果添加了  $\mathbf{A}$  和  $\mathbf{E}$  是  $G$ -协调的这个条件, 那么等号左边不为 0 的项只有  $\sigma = (1, 2, \dots, n)$ 。我们便刚好可以求出, 这种匹配方式对应的不交路径组数量。

**定理 3.1.** 令  $G$  是一张有向无环图,  $\mathbf{A} = (A_1, A_2, \dots, A_n), \mathbf{E} = (E_1, E_2, \dots, E_n)$  是  $G$ -协调的节点序列,  $w$  是边权函数 (同样定义在路径和路径组上)。

$$GF(L_G(\mathbf{A} \rightarrow \mathbf{E} \mid \text{non-intersecting}); w) = \det_{1 \leq i, j \leq n} (GF(L_G(A_j \rightarrow E_i); w))$$

在简单格路的环境下, 这个条件更容易被满足, 例如  $A_i = (1, a_i), E_i = (n, e_i)$ 。事实上, 绝大多数 LGV 的自然出现就是在格路计数中。

### 3.3 起点固定、终点不固定

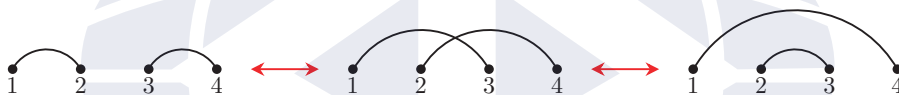
#### 3.3.1 终点在集合内任选

LGV 引理只能解决路径数量、起点数量和终点数量都相等的情形，也就是路径的起点集合和终点集合恰好为给出的两个集合。

接下来我们考虑，给定有向无环图  $G$  和起点集合  $\mathbf{A} = (A_1, A_2, \dots, A_n)$ ，终点集合  $\mathbf{E} = (E_1, E_2, \dots, E_m)$ ，满足  $n \leq m$  且  $\mathbf{A}$  和  $\mathbf{E}$  是  $G$ -协调的，每个起点  $a \in \mathbf{A}$  需要选一个终点  $e \in \mathbf{E}$ ，求所有选法的不相交路径的方案数。

**引理 3.2.** 令  $n, i, j$  是三个正整数，满足  $1 \leq i < j \leq n$ 。对完美匹配  $\pi \in \mathfrak{M}_n$ ，满足在  $\pi$  中  $i$  和  $j$  不匹配，令  $\pi'$  表示交换  $\pi$  中  $i$  和  $j$  的位置，然后在保持匹配关系不变的情况下，重排形成的新完美匹配。假如对所有  $i < k < j$ ，在  $\pi$  中  $k$  和  $i, j$  均不匹配，那么  $\text{sgn } \pi = -\text{sgn } \pi'$ 。

证明. 根据引理中所给的条件， $i, j$  及其匹配元素的变换情况，只可能为下图两种中的一种。用 3.1 节中提到的方法比较  $\text{inv } \pi$  和  $\text{inv } \pi'$  的奇偶性，也就是分别删去  $i, j$  及其匹配元素并检查贡献的不同。不难验证，在两种情况下，该引理均是正确的。



□

**定理 3.2.** 令  $G$  是一张有向无环图， $\mathbf{A} = (A_1, A_2, \dots, A_{2n})$ ,  $\mathbf{E} = (E_1, E_2, \dots, E_m)$  是  $G$ -协调的节点序列， $w$  是边权函数。

$$\sum_{\mathbf{E}' \subseteq \mathbf{E}, |\mathbf{E}'| = 2n} GF(L_G(\mathbf{A} \rightarrow \mathbf{E}' \mid \text{non-intersecting}); w) = \text{Pf}_{1 \leq i < j \leq 2n} (Q_G(i, j; w))$$

其中  $Q_G(i, j; w)$  表示对所有不交路径对  $(P', P'')$ ，满足起点分别为  $A_i$  和  $A_j$ ，终点均为  $\mathbf{E}$  中的点， $\sum_{(P', P'')} w(P')w(P'')$ 。

证明. 我们希望仿照 LGV 引理的证明，构造对合  $\phi$  将二元组  $(\pi, \mathbf{P})$  中  $\mathbf{P}$  相交的那些消掉。

取任一  $G$  的拓扑序  $\mathcal{A}$ 。对于一个相交路径组  $\mathbf{P}$ ，我们将每个交点描述为  $(M, i, j)$  的形式，表示  $P_i$  和  $P_j$  在  $M$  处有一交点。找到所有这样的三元组中字典序最小的一个（记为  $(M, i, j)$ ），其中第一维的比较按照  $\mathcal{A}$  上的排名进行。交换  $P_i, P_j$  在  $M$  前的子路径；同时交换  $i, j$  在  $\pi$  中的位置，保持元素的匹配关系不变的情况下重排。便得到了需要构造的  $\phi$ 。

这个对合和 LGV 引理中构造的十分相似，不难发现  $w(\mathbf{P})$  不变，接下来只需要说明符号取反。首先，由于  $Q_G(i, j; w)$  的定义保证了匹配的起点对应的路径不交，因此  $i$  和  $j$  在  $\pi$  中不匹配。

其次, 假设存在  $i < k < j$  和  $i, j$  其一匹配, 不妨设  $k$  和  $i$  匹配。记  $P_i, P_k$  对应的的终点编号分别为  $id_i, id_k$ ,  $P_i, P_j$  分别被  $M$  分割成  $AB$  和  $CD$ 。根据假设  $P_k$  和  $A, B$  均不交, 而由于  $M$  是拓扑序最小的交点,  $P_k$  和  $C$  也不相交。由于  $\mathbf{A}$  和  $\mathbf{E}$  是  $G$ -协调的, 必然有  $id_k > id_i$ , 否则它们不可能不交。现在考虑路径  $P_k$  以及  $CB$ , 由于  $CB$  的终点  $id_i$  在  $id_k$  前面,  $P_k$  和  $CB$  必然相交, 而前面推出了  $P_k$  和  $C, B$  均不相交, 矛盾。

这样一来,  $\phi$  对  $\pi$  的影响, 也就是交换  $i$  和  $j$  的位置, 满足引理 3.2 中的条件, 因此符号必然取反。由于  $A, C$  上不存在其它的交点, 也容易说明再进行一次变换会回到初始状态, 进而  $\phi$  是对合。

最后, 待验证的是, 每组合法的不交路径恰好贡献 1。由命题 3.1(a) 这也是显然的。□

假如路径数量 (同起点集合) 是奇数, 我们只需向  $G$  加一个没有任何邻边的虚点  $v$ , 同时将  $v$  添加到  $\mathbf{A}$  和  $\mathbf{E}$  的末尾。这样一来, 起点集合的  $v$  只能和终点集合的自己以唯一的方式匹配, 答案不变但转化为了偶数的情况。

### 3.3.2 混合情况

**定理 3.3.** 令  $G$  是有向无环图,  $\mathbf{A} = (A_1, A_2, \dots, A_n), \mathbf{E} = (E_1, E_2, \dots, E_m), \hat{\mathbf{E}} = (\hat{E}_1, \hat{E}_2, \dots, \hat{E}_k)$  是三个节点序列, 满足  $n \geq m$  且  $n + m$  是偶数, 另外  $(A_1, A_2, \dots, A_m)$  和  $\mathbf{E}$  是  $G$ -协调的,  $(A_{m+1}, A_{m+2}, \dots, A_n)$  和  $\hat{\mathbf{E}}$  是  $G$ -协调的。  $w$  是边权函数。

$$GF(L_G(\mathbf{A} \rightarrow \mathbf{E} \cup \hat{\mathbf{E}} \mid \text{non-intersecting}); w) = (-1)^{\binom{m}{2}} \text{Pf} \begin{pmatrix} Q & H \\ -H^T & 0 \end{pmatrix}$$

其中  $L_G(\mathbf{A} \rightarrow \mathbf{E} \cup \hat{\mathbf{E}} \mid \text{non-intersecting})$  表示所有起点序列为  $\mathbf{A}$  的长度为  $n$  的不交路径组  $\mathbf{P}$ , 满足  $\forall 1 \leq i \leq m, P_i$  的终点属于  $\mathbf{E}$ ;  $\forall m < i \leq n, P_i$  的终点属于  $\hat{\mathbf{E}}$ 。  $Q = (Q_G(i, j; w))_{1 \leq i, j \leq n}$  是一个反对称矩阵, 任意  $i < j$ ,  $Q_G(i, j; w)$  表示对所有不交路径对  $(P', P'')$ , 满足起点分别为  $A_i$  和  $A_j$ , 终点均为  $\hat{\mathbf{E}}$  中的点,  $\sum_{(P', P'')} w(P')w(P'')$ 。  $H = (GF(L_G(A_i \rightarrow E_j); w))_{1 \leq i \leq n, 1 \leq j \leq m}$  是一个  $n \times m$  的矩阵。

事实上, 该定理是 minor summation formula 的特殊情况 [1], 后者的其中一种形式在下面给出:

**定理 3.4.** 令  $m, n, p$  是正整数, 满足  $n + m$  是偶数且  $0 \leq n - m \leq p$ 。令  $M$  是一个  $n \times p$  的矩阵,  $H$  是一个  $n \times m$  的矩阵,  $A = (a_{i,j})_{1 \leq i, j \leq p}$  是一个反对称矩阵。那么我们有

$$\sum_K \text{Pf}(A_K^K) \det(M_K; H) = (-1)^{\binom{m}{2}} \text{Pf} \begin{pmatrix} MAM^T & H \\ -H^T & 0 \end{pmatrix}$$

其中  $K$  枚举所有  $1, 2, \dots, p$  的  $n - m$  元子集,  $A_K^K$  表示  $A$  选取  $K$  对应的行列得到的反对称矩阵,  $M_K$  表示  $M$  选取  $K$  对应的列得到的矩阵。

### 3.4 起点和终点不固定

最后我们来看起点和终点都不固定的情况。此时答案还由路径组中的路径数量决定，我们考虑直接求出答案关于路径数量的生成函数。

**定理 3.5.** 令  $G$  是一张有向无环图， $\mathbf{A} = (A_1, A_2, \dots, A_n)$ ,  $\mathbf{E} = (E_1, E_2, \dots, E_m)$  是节点序列，满足  $\mathbf{A}$  和  $\mathbf{E}$  是  $G$ -协调的， $w$  是边权函数。

$$\sum_{s \geq 0} x^s \sum_{\mathbf{A}' \subseteq \mathbf{A}, \mathbf{E}' \subseteq \mathbf{E}, |\mathbf{A}'| = |\mathbf{E}'| = s} GF(L_G(\mathbf{A}' \rightarrow \mathbf{E}' \mid \text{non-intersecting}); w)$$

当  $n$  是奇数时

$$= \text{Pf}_{1 \leq i < j \leq n+1}((-1)^{i+j-1} + x^2 Q_G(i, j; w))$$

其中  $Q_G(i, j; w)$  在  $j \leq n$  时与定理 3.2 的定义一致，而  $Q_G(i, n+1; w) = x^{-1} \sum_{E \in \mathbf{E}} GF(L(A_i \rightarrow E); w)$ ;

当  $n$  是偶数时

$$= \text{Pf}_{1 \leq i < j \leq n+2}((-1)^{i+j-1} + x^2 Q_G(i, j; w))$$

其中  $Q_G(i, j; w)$  在  $j \leq n+1$  时定义同上，而  $Q_G(i, n+2; w) = 0$ 。

证明. 观察  $(-1)^{i+j-1} + x^2 Q_G(i, j; w)$ ，其意义即为是否同时选择这对匹配的两个起点。如果不选择这对匹配，由前文我们知道它会对符号产生影响  $(-1)^{i+j+1}$ ，这里刚好将其消去；如果选择则乘上  $x^2$  及不交路径数。

那么我们可以将 Pfaffian 展开，得到的每一项还是形如  $(\pi, \mathbf{P})$ ，满足在  $\pi$  中匹配的起点同时属于或者不属于  $\mathbf{P}$  的起点。于是我们在定理 3.2 的证明中构造的对合  $\phi$  可以沿用至此，将相交路径的方案数抵消。

但是若干个匹配包含的起点数量一定是偶数，如果我们想选奇数个起点，需要添加一个虚点  $A_{n+1}$  和其中一个起点  $A_i$  匹配，匹配被选上的贡献就是  $x \sum_{E \in \mathbf{E}} GF(L(A_i \rightarrow E); w)$ 。这便对应了  $n$  是奇数时的答案。

当  $n$  是偶数时，为了让斜对称矩阵的大小变成偶数，我们还需要再添加一个空点。它和任何其它起点被同时选上的系数都是 0。这对应了  $n$  是偶数时的答案。

在两种情况中，都容易验证合法方案的贡献恰好为 1。这样便证明了该定理。□

对定理 3.5 的计算工作可以简化为求  $\text{Pf}(\mathbf{A} + \mathbf{B}x + \mathbf{C}x^2)$ ，其中  $\mathbf{A}, \mathbf{B}, \mathbf{C}$  都是大小相同（不超过  $n+2$ ）的反对称矩阵。

在具体的题目中，例如笔者的集训队互测题<sup>1</sup>，往往会将答案对大质数取模。一个直接的想法是插值，这样我们只需要计算  $O(n)$  次常系数矩阵的 Pfaffian，但是复杂度略高。另一个想法是转化为求  $\det(\mathbf{A} + \mathbf{B}x + \mathbf{C}x^2)$ ，其经典做法是先将  $\mathbf{A}$  消成单位矩阵，然后转化成求另一个常系数矩阵的特征多项式，复杂度是  $O(n^3)$  的。更详细的做法可以参考该题题解以及网上的其它资料。

<sup>1</sup>网格图最大流计数，<https://qoj.ac/problem/7648>



## 4 边界限制

边界限制，顾名思义，计数和给定边界不交的格路数量。

本节将会从 Ballot Problem 出发，讨论了直线边界下的若干问题，以及两个重要方法：折线法与旋转法，然后介绍了处理更一般边界的较为优秀的算法。

### 4.1 斜率为 1 的直线

#### 4.1.1 Ballot Problem 与折线法

Ballot Problem 是指如下的组合问题：

- 在一场选举中，候选人  $A$  获得了  $n$  票，候选人  $B$  获得了  $m$  票，满足  $n > m$ 。求有多大的概率使得在计票过程中  $A$  的票数一直高于  $B$ ？

不难发现，我们可以将其转化为格路计数模型。

首先  $A$  必须获得第一票。接下来，考虑一条从原点出发的路径，每当  $A$  获得一票就向右走一步，每当  $B$  获得一票就向上走一步。那么  $A$  的票数始终高于  $B$  的限制就变为，路径不能碰到  $y = x + 1$ 。

**定理 4.1.** 从  $(0, 0)$  到  $(n, m)$ ，始终不与  $y = x + 1$  相交的格路数量为

$$|L((0, 0) \rightarrow (n, m) \mid x \geq y)| = \binom{n+m}{n} - \binom{n+m}{n+1} = \frac{n+1-m}{n+1} \binom{n+m}{n}$$

证明. 利用容斥原理，我们只需计算总的数量  $\binom{n+m}{n}$  减去碰到过  $y = x + 1$  的数量。

折线法，又名反射法、反射原理等等。它正是计算“碰到某条直线的格路数量”的利器，接下来我们将阐述其思想。

对于一条和  $y = x + 1$  有交的路径  $P$ ，我们可以找到其和  $y = x + 1$  最早的交点  $A$ 。沿着  $y = x + 1$  将  $P$  在  $A$  之前的部分反射（对称）过去，便得到了一条从  $(-1, 1)$  到  $(n, m)$  的格路。如图 2(a) 所示，红色路径在  $A$  之前的部分被反射变为蓝色路径。

如此，我们构造了映射  $\phi : L((0, 0) \rightarrow (n, m) \mid \text{Intersecting with } y = x + 1) \rightarrow L((-1, 1) \rightarrow (n, m))$ ，接下来我们想证明  $\phi$  是双射，只需对每个像找到唯一的原像与之对应。这一点是容易的，因为任意一条  $L((-1, 1) \rightarrow (n, m))$  中的路径至少和  $y = x + 1$  有一个交点，我们找到最早的交点做一次逆变换即可。

综上，我们得到

$$\begin{aligned} |L((0, 0) \rightarrow (n, m) \mid x \geq y)| &= |L((0, 0) \rightarrow (n, m))| - |L((0, 0) \rightarrow (n, m) \mid x < y \text{ at least once})| \\ &= |L((0, 0) \rightarrow (n, m))| - |L((-1, 1) \rightarrow (n, m))| \\ &= \binom{n+m}{n} - \binom{n+m}{n+1} \end{aligned}$$

□



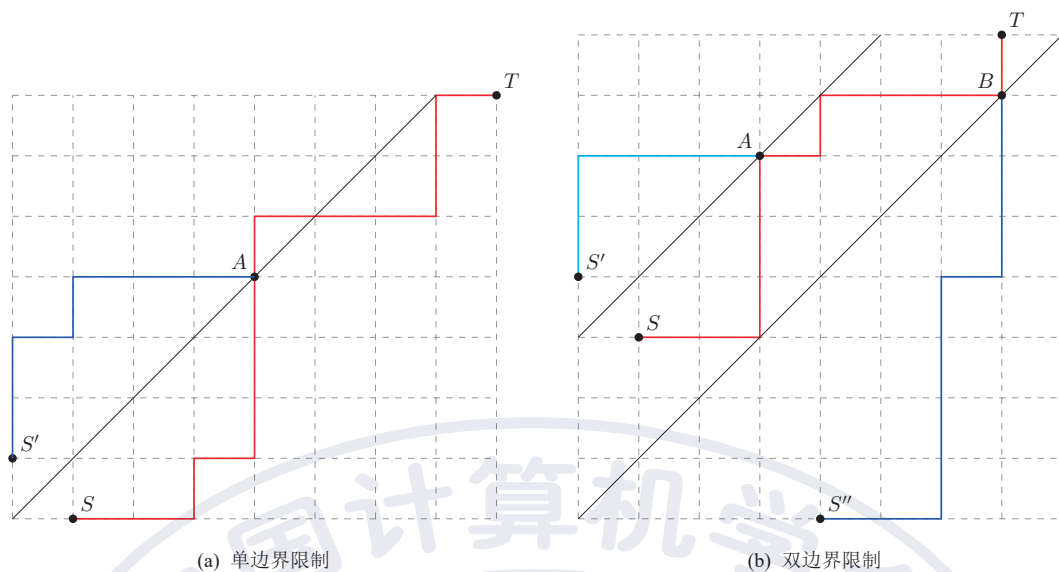


图 2: 折线法

借助定理 4.1, 我们可以立刻得到 Ballot Problem 的答案:  $\frac{n-m}{n+m}$ 。

进一步, 如果起点和边界的位置关系任意, 不难发现折线法仍然适用。

**定理 4.2.** 令  $n, m$  是非负整数,  $b$  是负整数, 满足  $m > n + b$ 。从  $(0, 0)$  到  $(n, m)$ , 始终不与  $y = x + b$  相交的格路数量为

$$|L((0, 0) \rightarrow (n, m) \mid y > x + b)| = \binom{n+m}{n} - \binom{n+m}{n-b}$$

#### 4.1.2 Dyck path 与卡特兰数

由定理 4.1 容易得到

**推论 4.1.** 从  $(0, 0)$  到  $(n, n)$ , 始终不与  $y = x + 1$  相交的格路数量为

$$|L((0, 0) \rightarrow (n, n) \mid x \geq y)| = \frac{\binom{2n}{n}}{n+1}$$

这样的格路被称为 **Dyck 路**, 其数量便是卡特兰数。它们的身影出现在组合数学的各个角落。

卡特兰数有各种各样组合意义、递推式、渐进估计。由于不是本文重点, 此处略去。

Dyck 路有许多扩充定义 (generalized Dyck path), 我们将在 4.2.1 节讨论其中一种。





### 4.1.3 双边界限制与反射容斥

接下来我们考虑被两条斜率为 1 的直线限制的格路计数。形式化的, 给定  $n, m, l, r \in \mathbb{Z}$ , 满足  $l < 0 < r, n + l < m < n + r$ , 求从  $(0, 0)$  到  $(n, m)$ , 始终不与  $y = x + l$  和  $y = x + r$  相交的格路数量。

反射容斥的想法是, 先将计算的贡献容斥成若干部分, 然后用折线法一一处理。

对于一条格路  $P$ , 维护一个字符串  $s(P)$ , 每当它碰到  $y = x + l$ , 就往  $s(P)$  末端添加一个  $L$ ; 每当他碰到  $y = x + r$ , 就往  $s(P)$  末端添加一个  $R$ 。那么我们要求的就是  $|L((0, 0) \rightarrow (n, m) \mid s(P) = \emptyset)|$ 。为了方便, 用  $\emptyset$  表示空字符串; 记  $S \subseteq T$  表示  $S$  是  $T$  的子序列; 记  $\#(S) = \sum_{P \in L((0, 0) \rightarrow (n, m))} [S \subseteq s(P)]$ 。

首先我们审视折线法可以处理怎样的计数对象。类似 4.1.1 节中所述, 我们可以找到路径第一次和某边界的交点并将其翻转过去。其数量分别对应了  $\#(L)$  和  $\#(R)$ 。

然而当我们两次将路径沿着同一直线反射后, 它会变会最初的形态, 这显然不是我们想要的。尽管不能多次将路径沿着同一直线反射, 但是可以沿着两条直线交替反射:

当我们将路径  $P$  沿着  $y = x + r$  反射一次后, 假设找到的交点在  $A$ , 那么反射后  $P$  在  $A$  之前的部分一定和  $y = x + l$  无交。这样我们就可以进行沿着  $y = x + l$  进行第二次反射, 使得这次反射找到的交点  $B$  在  $A$  之后。如图 2(b) 所示。

由此一来, 我们可以计数这样的对象:  $\#(L), \#(R), \#(LR), \#(RL), \#(LRL), \#(RLR) \dots$ , 接下来尝试用它们凑出所求, 考虑如下的容斥

$$[s = \emptyset] = [\emptyset \subseteq s] - [L \subseteq s] - [R \subseteq s] + [LR \subseteq s] + [RL \subseteq s] - [LRL \subseteq s] - [RLR \subseteq s] + \dots$$

该式的正确性是容易证明的。而等号右边的形式是折线法所擅长的, 我们有

$$|L((0, 0) \rightarrow (n, m) \mid s(P) = \emptyset)| = \#(\emptyset) - \#(L) - \#(R) + \#(LR) + \#(RL) - \#(LRL) - \#(RLR) + \dots$$

至此我们完成了反射容斥的工作, 代入自由格路的计算式立刻得到

**定理 4.3.** 令  $n, m$  是非负整数,  $l, r$  是整数, 满足  $l < 0 < r, n + l < m < n + r$ , 从  $(0, 0)$  到  $(n, m)$ , 始终不与  $y = x + l$  和  $y = x + r$  相交的格路数量为

$$|L((0, 0) \rightarrow (n, m) \mid x + l < y < x + r)| = \sum_{k \in \mathbb{Z}} \left( \binom{n+m}{n-k(r-l)} - \binom{n+m}{n-k(r-l)+r} \right)$$

可以发现, 当枚举的  $k$  过大时组合数恒为 0, 因此我们只需要计算  $O(\frac{n+m}{k})$  个  $O(n+m)$  量级的组合数。如果答案对大质数取模, 我们可以用一些多项式技术做到  $\tilde{O}(\sqrt{n+m})$ , 由于不是本文重点, 故略去。

## 4.2 斜率为有理数的直线

对于最一般的边界  $y = kx + b$ , 其中  $k, b \in \mathbb{R}$ , 折线法不再适用, 似乎目前也没有较好的解决办法。本节将讨论一些特殊情况, 它们的答案存在着简洁的公式, 且均满足  $k \in \mathbb{Q}$ 。



### 4.2.1 $(n, m)$ -Dyck 路

**定义 4.1.** 若一条从  $(0, 0)$  到  $(n, m)$  的格路, 始终不经过  $y = \frac{m}{n}x$  上方, 则称之为一条  $(n, m)$ -Dyck 路。

记  $D(n, m) := |L((0, 0) \rightarrow (n, m) \mid y \leq \frac{m}{n}x)|$ , 即  $(n, m)$ -Dyck 路的数量。

若  $n, m$  互质,  $D(n, m)$  有简单的计算公式。我们先来证明一个引理。

**引理 4.1 (Spitzer's Lemma).** 若和为 0 的实数序列  $a_1, a_2, \dots, a_n$ , 满足其所有真子区间的和都不为 0, 那么  $a$  恰好存在一种循环移位使得任意前缀非负。

证明. 记  $s_i = \sum_{j=1}^i a_j$ , 由真子区间非 0 的条件易知  $s_1, s_2, \dots, s_n$  两两不同。

假设某种循环移位  $a_p, \dots, a_n, a_1, \dots, a_{p-1}$  ( $1 \leq p \leq n$ ) 满足条件, 等价于  $\forall 1 \leq i \leq n, s_i \geq s_{p-1}$ , 也即  $s_{p-1} = \min_{i=1}^n s_i$ 。因此恰有一个  $p$  满足条件。□

**定理 4.4.** 任意  $n, m \in \mathbb{Z}^+$ , 满足  $(n, m) = 1$ ,  $(n, m)$ -Dyck 路的数量为:

$$D(n, m) = \frac{1}{n+m} \binom{n+m}{n}$$

证明. 对于一条从  $(0, 0)$  到  $(n, m)$  的简单格路  $P$ , 构造一个长度为  $n+m$  的序列  $A$ 。如果  $P$  的第  $i$  步是向上就令  $A_i := -n$ , 否则  $A_i := m$ 。

可以发现  $\sum_{i=1}^{n+m} A_i = 0$ ; 如果  $P$  是合法的  $(n, m)$ -Dyck 路, 那么  $A$  的任意前缀非负。因此根据 Spitzer's Lemma,  $P$  恰好存在一个循环同构合法。对于所有简单格路, 将它们按循环同构关系分为  $\frac{1}{n+m} \binom{n+m}{n}$  类, 每个等价类中恰有一条路径合法。□

对于  $(n, m)$  不互质的情况, 这个做法会出现两个问题: 路径可能在中途和  $y = \frac{m}{n}x$  相交, 以及循环同构的等价类大小不一定是  $n+m$ 。

针对这两个问题, 考察一条与  $y = \frac{m}{n}x$  相交了  $k$  次 (包含终点不含起点) 的路径, 且其最短循环节为  $l$ 。那么其每个循环节内恰好有  $\frac{kl}{n+m}$  个交点, 也就是说这  $l$  条不同的路径需要总共贡献  $kl$  次, 那么平均下来每条路径需要贡献  $\frac{k}{n+m}$  次, 而在 Spitzer's Lemma 引理中只贡献了  $\frac{1}{n+m}$  次。

记  $d = (x, y), n' = \frac{n}{d}, m' = \frac{m}{d}$ 。构造生成函数  $F, G$ , 其中  $[x^i]F(x) = \frac{1}{i(n'+m')} \binom{i(n'+m')}{in'}$  (特别的,  $[x^0]F(x) = 0$ );  $[x^i]G(x)$  表示从  $(0, 0)$  到  $(in', im')$ , 除起点终点外严格在  $y = \frac{m}{n}x$  下方的格路数量, 那么我们有

$$\ln(1 - G) = - \sum_{i \geq 1} \frac{G^i}{i} = -F$$

所求即为  $\frac{1}{1-G} = \exp F$ 。

另外, 当  $n, m$  有整除关系时,  $D(n, m)$  有更简单的算法, 将在下一节介绍。

## 4.2.2 广义 Ballot Problem

Ballot Problem 要求我们对起点在原点，边界斜率为 1 的直线计数。广义 Ballot Problem 中起点仍在原点，但边界斜率为任意正整数，终点在边界上方。

形式化的，给定正整数  $n, m, \mu$ ，满足  $m \geq \mu n$ ，求  $|L((0,0) \rightarrow (n,m) \mid y \geq \mu x)|$ 。

让我们尝试沿用之前的证明思路。先在路径开头加一步  $(0,1)$ ，现在起点变为  $(0,-1)$ ，那么问题转化为以下引理。

**引理 4.2 (Cycle Lemma).** 令  $n, m, \mu$  是满足  $m \geq \mu n$  的正整数。对于任一含  $m$  个 1 和  $n$  个 2 的长度为  $n+m$  的序列  $p_1, p_2, \dots, p_{m+n}$ ，其恰好存在  $m - \mu n$  个循环同构使得任意真前缀中  $\#(1) > \mu \#(2)$ ，即 1 的个数大于 2 的个数的  $\mu$  倍。

**证明.** 将序列  $p$  首尾相接连成一个环。进行  $n$  次操作，每次找到一个 2，使得它前面有连续的  $\mu$  个 1，将这  $\mu+1$  个元素同时删去。最终得到一个长度为  $m - \mu n$  的全 1 序列，可以发现序列中的每个 1 恰好对应了原问题中的一个合法的循环同构。  $\square$

有了这个引理，我们立刻得到广义 Ballot Problem 的答案：

**定理 4.5.** 任意  $n, m, \mu \in \mathbb{Z}^+$ ，满足  $m \geq \mu n$ ，从  $(0,0)$  到  $(n,m)$ ，始终不低于  $y = \mu x$  的格路数量为

$$|L((0,0) \rightarrow (n,m) \mid y \geq \mu x)| = \frac{m - \mu n + 1}{m + 1} \binom{n+m}{n}$$

可以看到，这个问题中折线法难以沿用。接下来我们将介绍另一种方法——旋转法，它的核心思想和折线法十分相似，可以解决一类边界斜率为正整数且过起点的格路计数问题。

仍然考虑容斥，转化为存在  $y < \mu x$  的路径数量。回顾折线法的想法：找到最小的非法点，将其之前的路径进行变换，双射到另一些自由路上。

我们如法炮制，取路径上第一个满足  $y < \mu x$  的点  $A$ ，那么存在唯一的  $1 \leq b \leq \mu$  使得  $A$  在  $y = \mu x - b$  上。

接下来，我们将所有非法路径按照  $b$  分成总共  $\mu$  类，试图构造每一类与  $L((1,-1) \rightarrow (n,m))$  的双射。

双射的构造方式是，将  $A$  前一条边  $BA$  由横向变为竖向，然后将之前的部分旋转 180 度接在前面。如右图所示，将  $SB$  旋转为  $B'S'$ 。

一方面，对映射后的路径寻找第一个不低于  $y = \mu x - b$  的点，可以找到  $A$ ；另一方面，取任意一条  $(1,-1)$  到  $(n,m)$  的自由路，类似的找到  $A$  进行逆变换，可以得到原来那类路径中的某一条。因此该映射是一一映射。那么定理 4.5 的正确性便呼之欲出了。

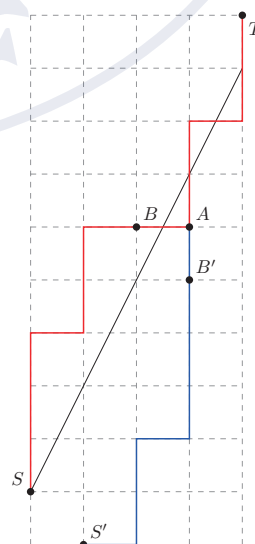


图 3: 旋转法



## 4.3 更一般的边界

### 4.3.1 反射容斥的推广

对反射容斥比较理想的推广，需要借助 root system 和 Weyl group 的概念进行刻画。这里只给出一些简化后的、最表层的定义和结论，感兴趣的读者可以阅读 [2]。

在  $\mathbb{R}^d$  上，令  $\mathcal{H}$  是一个超平面构成的有限集合， $W$  是将  $\mathcal{H}$  中所有元素看成边界，生成的反射群。注意到  $W$  中的一些反射变换并不沿着  $\mathcal{H}$  中的超平面进行，那么记  $\mathcal{R}$  表示  $W$  中的所有元素对应的超平面的集合。

$\mathcal{R}$  中的超平面将  $\mathbb{R}^d$  分割为许多区域， $\mathbb{R}^d - \bigcup_{H \in \mathcal{R}} H$  中的连通块被称为腔。不难发现，对于任意腔  $C$ ，取  $w \in W$ ，那么  $w(C)$  两两不同且覆盖了全部的腔。

经过如上的若干定义，我们已经具备了描述更一般的反射容斥的工具。然而，为了使反射容斥适用，步长集合也需要加以约束，使得路径无法在不接触边界的情况下越过边界。为此，我们通过定义  $k_H$  和  $r_H$  来实现这一点，其中  $H \in \mathcal{H}$ 。

**定理 4.6.** 令  $C$  是反射群  $W$  对应的一个腔，由超平面集合  $\mathcal{H}$  生成。取步长集合  $\mathbb{S}$ ，使得对任意  $w \in W$ ， $w(\mathbb{S}) = \mathbb{S}$ ，且对任意  $H \in \mathcal{H}, s \in \mathbb{S}$ ，内积  $\langle s, r_H \rangle$  的值为 0 或  $\pm k_H$ 。其中  $r_H$  是一和  $H$  垂直的非零向量， $k_H$  是固定常数，它们都只和  $H$  有关。

对于  $C$  中任意两个整点  $A, E$ ，满足对任意  $H \in \mathcal{H}$ ， $\langle A, r_H \rangle$  和  $\langle E, r_H \rangle$  都是  $k_H$  的整数倍。从  $A$  到  $E$ ，恰好  $m$  步，且始终在  $C$  内部的格路数量为

$$|L_m(A \rightarrow E; \mathbb{S} \mid \text{inside } C)| = \sum_{w \in W} (\det w) |L_m(w(A) \rightarrow E; \mathbb{S})|$$

下面给出两个具体例子来理解定理 4.6。

考虑在  $\mathbb{Z}^2$  中，步长集合为  $\{(1, -1), (1, 1), (F, 0)\}$ ，且始终不低于  $x$  轴的一类格路。当  $F = 1$  时，对应的格路称为 **Motzkin 路**。当  $F = 2$  时，对应的格路称为 **Schröder 路**。

**推论 4.2.** 令  $a, c$  是整数， $b, d$  是非负整数，从  $(a, b)$  到  $(c, d)$  的 Schröder 路的数量为

$$|L((a, b) \rightarrow (c, d); S = \{(1, -1), (1, 1), (2, 0)\} \mid y \geq 0)| = \sum_{k=0}^{(c-a)/2} \binom{c-a-k}{k} \left( \binom{c-a-2k}{(c+d-2k-a-b)/2} - \binom{c-a-2k}{(c+d-2k-a+b+2)/2} \right)$$

证明. 如果没有  $y \geq 0$  的限制，可以直接枚举  $(2, 0)$  走了多少步，而剩下的步长和简单步本质相同，只需将坐标系旋转 45 度。那么我们有

$$|L((a, b) \rightarrow (c, d); S)| = \sum_{k=0}^{(c-a)/2} \binom{c-a-k}{k} \binom{c-a-2k}{(c+d-2k-a-b)/2}$$

应用反射容斥，将问题转化为从  $(a, b)$  到  $(c, d)$  的自由 Schröder 路数量，减去从  $(a, b)$  到  $(c, -d-2)$  的自由 Schröder 路数量，代入上式即可完成证明。□

上面的推论借助 Schröder 路的例子, 揭示了这类格路均适用反射容斥解决, 其本质在于较为特殊的  $(F, 0)$  步长与边界  $y = -1$  平行, 对应了  $\langle s, r_H \rangle = 0$  的情况。另外, 此时可以取  $r_H = (0, 1), k_H = 1$ , 那么所有满足  $\langle s, r_H \rangle = k_H$  的步长都可以成对 (和经过反射的步长) 加入进步长集合, 即  $(x, 1)$  和  $(x, -1)$ 。

另一个例子是高维的 ballot problem, 即各坐标之间始终单调的简单格路。

**推论 4.3.** 令  $d$  是正整数,  $\mathbf{a} = (a_1, a_2, \dots, a_d)$  和  $\mathbf{e} = (e_1, e_2, \dots, e_d)$  是  $\mathbb{Z}^d$  中的两格点, 且满足  $\forall 1 \leq i < d, a_i < a_{i+1}, e_i < e_{i+1}$ 。从  $\mathbf{a}$  走到  $\mathbf{e}$ , 且始终保持  $x_1 < x_2 < \dots < x_d$  的路径数量为

$$|L(\mathbf{a} \rightarrow \mathbf{e} \mid x_1 < x_2 < \dots < x_d)| = \left( \sum_{i=1}^d (e_i - a_i) \right)! \det_{1 \leq i, j \leq d} \left( \frac{1}{(e_i - a_j)!} \right)$$

证明. 在该情境中,  $\mathcal{H} = \{x_i = x_{i+1} \mid 1 \leq i < d\}, \mathcal{R} = \{x_i = x_j \mid 1 \leq i < j \leq d\}$ , 而  $W$  对应了全体  $\{1, 2, \dots, d\}$  的置换, 因此根据定理 4.6 我们有

$$\begin{aligned} & |L(\mathbf{a} \rightarrow \mathbf{e} \mid x_1 < x_2 < \dots < x_d)| \\ &= \sum_{\sigma \in \mathfrak{S}_d} (\text{sgn } \sigma) |L(\mathbf{a}_\sigma \rightarrow \mathbf{e})| \\ &= \sum_{\sigma \in \mathfrak{S}_d} (\text{sgn } \sigma) \frac{\left( \sum_{i=1}^d (e_i - a_i) \right)!}{\prod_{i=1}^d (e_i - a_{\sigma(i)})!} \\ &= \left( \sum_{i=1}^d (e_i - a_i) \right)! \det_{1 \leq i, j \leq d} \left( \frac{1}{(e_i - a_j)!} \right) \end{aligned}$$

□

### 4.3.2 阶梯型格路计数

至此, 本章的所有内容仍局限在边界由若干超平面组成的情况。接下来, 我们将考虑一些更一般的情况。尽管此时不一定有简洁的结论, 但仍可能存在一些复杂度优秀的算法。

首先考虑阶梯型格路计数。形式化的, 给定单调不降的非负整数序列  $a_0, a_1, \dots, a_n$ , 计数从  $(0, 0)$  到  $(n, m)$  的简单路径, 使得路径上任意点  $(x, y)$  满足  $y \leq a_x$ 。

最直接的想法是 dp, 设  $f_{i,j}$  表示从  $(0, 0)$  走到  $(i, j)$  且不出阶梯的路径数量, 转移是  $O(1)$  的, 总复杂度  $O(nm)$ 。

当  $m$  较大时, 另一种思路是容斥。将对路径的阶梯型约束看成  $n+1$  个障碍点  $(i, a_i)$ , 钦定其中一些需要被经过, 每个被钦定的位置贡献一个  $-1$  的系数。据此不难设计一个状态和转移均为  $O(n)$  的 dp。转移系数为组合数, 还要预处理  $O(m)$  规模的阶乘及其逆元。总复杂度  $O(n^2 + m)$ 。

这样的容斥思路可扩展性强, 可用于障碍点形态任意的格路计数。假设障碍点有  $k$  个, 那么时间复杂度为  $O(k^2 + n + m)$ 。然而在这个情形下, 我们还有一个更优秀的做法。



转化一下计数的对象，变为求有多少个非负整数构成的不降序列  $b_0, b_1, \dots, b_{n-1}, b_n = m$ ，满足  $\forall 0 \leq i \leq n, b_i \leq a_i$ 。接下来，我们不对不超过  $a$  的限制容斥，而是对  $b$  不降的限制容斥。

钦定若干个位置满足  $b_i > b_{i+1}$ 。考虑一个被钦定的极长连续段  $b_l > b_{l+1} > \dots > b_r (r \neq m)$ ，那么这一段的方案数为  $\binom{a_l}{r-l+1}$ ，也是  $dp$  最主要的转移系数。而这样的系数在  $l$  固定时有递推关系，因此总复杂度  $O(n^2)$ 。

最后，如果  $n, m$  同阶，阶梯型格路计数的一个经典解决办法是分治 FFT。

我们的目标是对最朴素的  $dp$  进行优化。如图 4 的阶梯型网格，现在我们知道了最下层每个点的  $dp$  值，想要转移出最右边每个点的  $dp$  值。

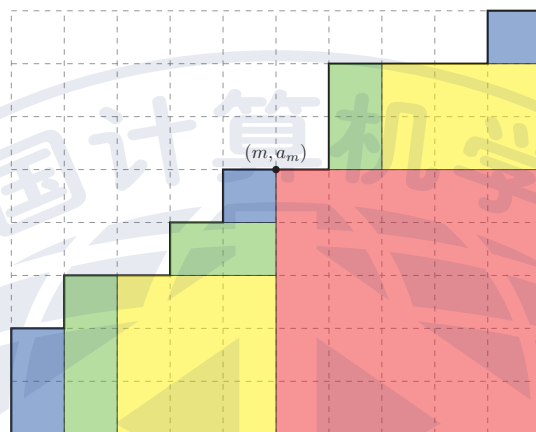


图 4: 阶梯型网格

首先找到横坐标的中点  $m$ ，那么  $(m, a_m)$  右下角的矩形（红色矩形）完全在阶梯内部。注意到，如果我们求出了红色矩形的左边界和下边界上的  $dp$  值，只需要一次卷积即可求出右边界和上边界的  $dp$  值，因为转移系数是组合数。

那么我们先解决  $(m, a_m)$  左下的子问题，做一次卷积，然后解决  $(m, a_m)$  右上的子问题，就可以解决原问题了。将这个过程不断递归下去，也就是分治，便得到了完整的算法。

分析一下这个做法的时间复杂度。首先，由于我们每次都是从中点将问题分开，因此只会进行  $O(\log n)$  层。每一次卷积的复杂度和对应矩形周长有关，假设周长为  $L$ ，那么使用 FFT 做卷积的复杂度为  $O(L \log L)$ 。注意到每一层的所有矩形（图中同色即同层）边长之和不超过  $2(n + m)$ 。因此当  $n, m$  同阶时，总复杂度为  $O(n \log^2 n)$ 。

### 4.3.3 障碍点任意

如果障碍点的形态分布没有任何约束，前文的若干做法将不再适用。仅剩的手段便是利用容斥的思想，钦定一些障碍点必须经过，然后进行  $dp$ 。

重剑无锋，这样的做法不依赖于障碍点的性质，具有很强的扩展性。甚至在一些障碍点也不固定的情形下，仍可以做到优秀的复杂度。





## 5 特征限制

本章将讨论三个常见的路径特征，计数这些特征值固定时的格路数量，或是格路的特征值之和。在此基础上还可能涉及不相交格路或边界限制。

### 5.1 与直线的交点

给定一条直线  $l$ ，那么这条直线会经过若干整点，设这些整点构成的集合为  $\mathcal{L}$ 。对于一条路径  $P = (P_0, P_1, \dots, P_l)$ ，计算  $\sum_{i=0}^l [P_i \in \mathcal{L}]$ ，称为  $P$  与该直线的交点数量，记为  $I_l(P)$ ，当  $l$  的含义明确且唯一时会将其省略。

#### 5.1.1 斜率为 1

当直线斜率为 1 时，我们想计算与其相交固定次数的格路数量，由以下定理给出：

**定理 5.1.** 令  $n, m, b$  是三个非负整数，满足  $m \geq n + b$ ， $t$  是不超过  $n$  的正整数。计数从  $(0, 0)$  到  $(n, m)$  的路径  $P$ ，满足  $P$  与  $y = x + b$  恰好相交  $t$  次。

$$|L((0, 0) \rightarrow (n, m) \mid I(P) = t)| = 2^{t-1} \binom{n+m-t+1}{m} - 2^t \binom{n+m-t}{m}$$

证明. 不难发现我们只需证明

$$|L((0, 0) \rightarrow (n, m) \mid I(P) \geq t)| = 2^{t-1} \binom{n+m-t+1}{m}$$

对于一条路径  $P$  满足  $I(P) \geq t$ ，我们找到  $P$  和  $y = x + b$  的前  $t$  个交点  $A_1, A_2, \dots, A_t$ 。交点之间的路径全部在  $y = x + b$  一侧，由于有反射构造的双射，我们只需求  $A_t$  前均满足  $y \leq x + b$  的路径数量，再乘上  $2^{t-1}$  即可。

此时显然有，对任意  $1 \leq i < t$ ， $A_i$  后的第一步是  $(1, 0)$ 。我们将这  $t-1$  步  $(1, 0)$  从  $P$  中删去，其余每步依次拼接并保持起点不动，得到一条新的路径，其起点和终点分别为  $(0, 0)$  和  $(n-t+1, m)$ 。如图 5 所示。

于是我们构造了一个待求路径集到  $L((0, 0) \rightarrow (n-t+1, m))$  的映射。可以发现这是双射，只需依次找到  $A_1, A_2, \dots, A_t$  并将删去的  $(1, 0)$  换元即可。由此不难得到待证等式。□

定理 5.1 有趣的一点在于，结果和  $b$  无关。回顾  $b$  的定义，它的限制只有  $0 \leq b < m - n$ 。也就是说，夹在  $(0, 0)$  和  $(n, m)$  之间的直线，他们都有相同的答案。事实上，我们也可以通过折线法构造双射来证明这一点，此处不再赘述。

还有一个小问题，如果  $m < n + b$ ，只需通过折线法将终点反射至  $y = x + b$  上方，便转化为了定理 5.1 适用的情形。

有了如上的结论，我们还可以计数所有路径和  $y = x + b$  的交点数量之和。

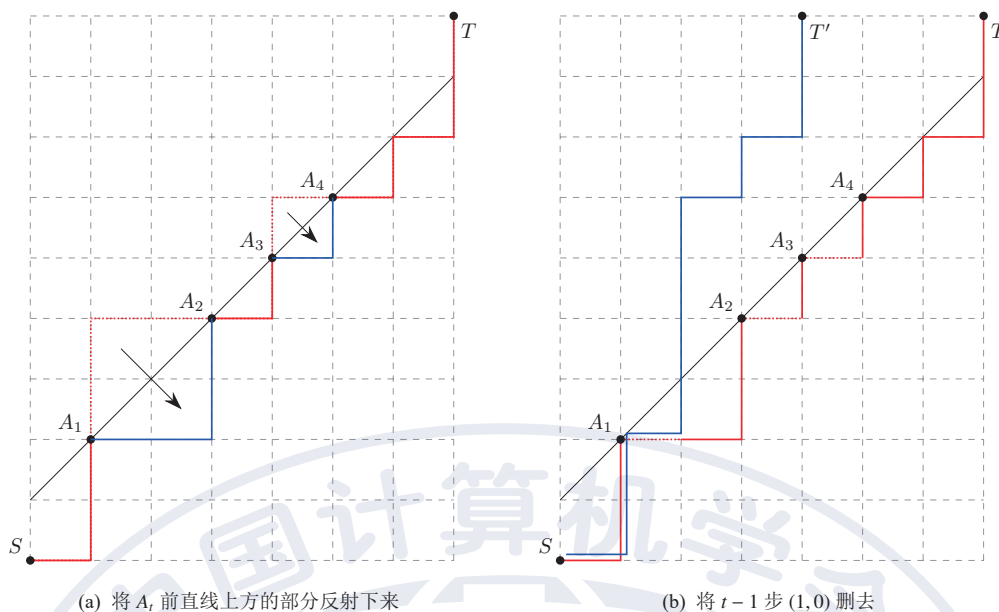


图 5: 双射的构造,  $t=4$

**推论 5.1.** 令  $n, m, b$  是三个非负整数, 满足  $m \geq n + b$ . 从  $(0, 0)$  到  $(n, m)$  的所有路径与  $y = x + b$  的交点个数之和为

$$GF(L((0, 0) \rightarrow (n, m); I(P))) = \sum_{i=0}^n 2^i \binom{n+m-i}{m} = \sum_{i=m+1}^{n+m+1} \binom{n+m+1}{i}$$

第一个等号显然, 第二个等号只需要将  $2^i$  二项式展开即可。可以发现, 最终结果的形式是一个经典问题——组合数前缀和。

### 5.1.2 斜率为正整数

**定理 5.2.** 令  $n, m, b$  是三个非负整数,  $k$  是正整数, 满足  $m \geq kn + b$ ,  $t$  是不超过  $n$  的正整数。计数从  $(0, 0)$  到  $(n, m)$  的路径  $P$ , 满足  $P$  与  $y = kx + b$  恰好相交  $t$  次。

$$|L((0, 0) \rightarrow (n, m) \mid I(P) = t)| = (k+1)^{t-1} \binom{n+m-t+1}{m} - (k+1)^t \binom{n+m-t}{m}$$

证明. 我们仍然只需证明

$$|L((0, 0) \rightarrow (n, m) \mid I(P) \geq t)| = (k+1)^{t-1} \binom{n+m-t+1}{m}$$

然而, 当我们想沿用之前的方法时, 无法简单的将两个整交点之间的路径, 分为在直线上下的两部分。因为路径有可能和直线在非格点处相交。



如果某个整交点  $A_i$  之后的第一步是  $(1, 0)$ , 那么下一次相交一定在整点处, 也就是  $A_{i+1}$ 。这提示我们, 不高于  $y = kx + b$  的情形是好算的, 接下来我们通过旋转法将其它的情况转化过来。

观察相邻的两个整交点  $A_i, A_{i+1}$  之间的部分。假设  $A_i$  后的第一步是  $(0, 1)$ , 找到路径上  $A_i$  之后第一个满足  $y \leq kx + b$  的格点  $A'$ , 那么存在唯一的  $0 \leq b' < k$  使得  $A'$  在  $y = kx + b - b'$  上, 我们将这样的路径按照  $b'$  分为  $k$  类。

任取这  $k$  类中的某一类, 对其中所有路径进行操作: 将  $A_i A'$  旋转  $180^\circ$ 。接下来我们说明这个操作构造了, 这一类路径和不高于直线的路径之间的双射。

一方面, 如果对操作后的路径寻找  $A_i$  之后第一个不低于  $y = kx + b - b'$  的点, 那么会找到  $A'$ ; 另一方面, 任意一条不高于直线的子路径, 将第一个不低于  $y = kx + b - b'$  的点看作  $A'$ , 容易证明对其进行逆变换后, 会让  $A_i$  和  $A'$  之间的路径高于  $y = kx + b$ , 进而对应了原来那类路径中的一条。至此我们构造了所需的双射。

现在我们可以将  $A_i$  和  $A_{i+1}$  之间的方案数, 看成不高于  $y = kx + b$  的方案数的  $k + 1$  倍。因此最终只需要计数  $A_i$  之前均不高于  $y = kx + b$  的路径数量, 只需对每个  $1 \leq i < t$  删去  $A_i$  后第一步  $(1, 0)$ , 双射到另一些自由路上。  $\square$

**推论 5.2.** 令  $n, m, b$  是三个非负整数,  $k$  是正整数, 满足  $m \geq kn + b$ 。从  $(0, 0)$  到  $(n, m)$  的所有路径与  $y = kx + b$  的交点个数之和为

$$GF(L((0, 0) \rightarrow (n, m); I(P))) = \sum_{i=0}^n (k+1)^i \binom{n+m-i}{m} = \sum_{i=m+1}^{n+m+1} k^{i-m-1} \binom{n+m+1}{i}$$

## 5.2 拐点

对于一条从  $(a, b)$  到  $(c, d)$  的简单格路  $P$ , 每走一步  $(1, 0)$  我们就写一个 E, 每走一步  $(0, 1)$  我们就写一个 N, 最终会得到恰好有  $c - a$  个 E 和  $d - b$  个 N 的字符串。

考虑这个字符串中的一个 EN 子串, 在路径上对应的是连续的  $(1, 0)$  和  $(0, 1)$ 。我们称这两步之间的点, 也就是前一步的终点、后一步的起点, 是一个 EN 拐点。记一条路径  $P$  的 EN 拐点个数是  $EN(P)$ 。同样我们可以定义 NE 拐点与  $NE(P)$ 。

本章研究的是, 在某种拐点的数量一定的条件下, 若干经典问题的再探讨。

### 5.2.1 无限制

首先考虑没有限制的情况: 从  $(0, 0)$  到  $(n, m)$ , 计数某种拐点的数量固定的格路。

为了更好的刻画拐点的形态, 我们引入拐点信息的概念。

**定义 5.1 (拐点信息).** 令路径  $P$  有  $\ell$  个 EN 拐点, 依次设为  $(p_1, q_1), (p_2, q_2), \dots, (p_\ell, q_\ell)$ , 那么长度均为  $\ell$  的序列对  $(p, q)$  称为  $P$  的 EN 拐点信息。同理可定义 NE 拐点信息。

可以看到拐点信息实质上是, 将所有拐点的两维坐标分别组成了两个序列。由于本节讨论的都是简单格路, 因此无论是对 EN 拐点信息还是 NE 拐点信息,  $p$  和  $q$  均单调递增。进一步, EN 拐点信息具有以下约束:

$$\begin{aligned} 1 \leq p_1 & p_2 \dots p_\ell \leq n \\ 0 \leq q_1 & q_2 \dots q_\ell < m \end{aligned}$$

另一方面, 任意一组满足条件的序列对  $(p, q)$  都对应了一条路径。那么我们可以直接对满足条件的  $(p, q)$  计数。如果限制的是 NE 拐点, 也有相同的结果。

**定理 5.3.** 令  $n, m, \ell$  是非负整数。从  $(0, 0)$  到  $(n, m)$ , 有  $\ell$  个 EN 拐点或有  $\ell$  个 NE 拐点的格路数量为

$$|L((0, 0) \rightarrow (n, m) \mid \text{EN}(P) = \ell)| = |L((0, 0) \rightarrow (n, m) \mid \text{NE}(P) = \ell)| = \binom{n}{\ell} \binom{m}{\ell}$$

### 5.2.2 斜率为 1 的边界

添加一个斜率为 1 的直线的边界限制, 仍然固定 EN 拐点或 NE 拐点计数。

我们已经知道, 反射法是解决此类问题的利器。接下来会直接对拐点信息进行类似的变换, 实质上也是将路径的反射描述为了更直观的形式。

先考虑只有一个边界, 固定 EN 拐点的版本: 计数从  $(0, 0)$  到  $(n, m)$ , 始终不低于  $y = x + b$  且有  $\ell$  个 EN 拐点数量的格路。

仍然考虑容斥, 那么限制转化为路径上存在过低于  $y = x + b$  的点。可以发现, 路径  $P$  满足这个条件等价于  $P$  存在一个 EN 拐点低于  $y = x + b$ 。

记最后一个低于  $y = x + b$  的拐点编号为  $I$ 。考虑将拐点信息双射到如下序列对。

$$\begin{aligned} -b \leq q_1 - b & \dots q_I - b & p_I & p_{I+1} & \dots p_\ell \leq n \\ b < & p_1 + b & \dots p_{I-1} + b & q_{I+1} & \dots q_\ell < m \end{aligned}$$

由于假设保证了  $q_I < p_I + b < p_{I+1} + b \leq q_{I+1}$ , 不难验证这两个序列各自都是单调递增的。另外有  $(p_I, p_{I-1} + b)$  低于  $y = x + b$ , 而  $I$  之后的信息都没有变化, 因此我们仍能在新序列对上, 从后往前定位到  $I$ , 将其映射回来。特别的, 如果找不到这样的位置,  $I = 1$ 。也就是说, 所求与满足下述约束条件的单增序列对  $(\bar{p}, \bar{q})$  双射。

$$\begin{aligned} -b \leq \bar{p}_0 & \bar{p}_1 & \bar{p}_2 & \dots & \bar{p}_\ell \leq n \\ b < & \bar{q}_2 & \dots & \bar{q}_\ell < m \end{aligned}$$

**定理 5.4.** 令  $n, m, \ell$  是非负整数,  $b$  是非正整数, 满足  $m \geq n + b$ 。从  $(0, 0)$  到  $(n, m)$ , 始终不低于  $y = x + b$  且有  $\ell$  个 EN 拐点的格路数量为

$$|L((0, 0) \rightarrow (n, m) \mid y \geq x + b, \text{EN}(P) = \ell)| = \binom{n}{\ell} \binom{m}{\ell} - \binom{n+b+1}{\ell+1} \binom{m-b-1}{\ell-1}$$

类似的, 也可以对 NE 拐点数量进行限制。

**定理 5.5.** 令  $n, m, \ell$  是非负整数,  $b$  是非正整数, 满足  $m \geq n + b$ . 从  $(0, 0)$  到  $(n, m)$ , 始终不低于  $y = x + b$  且有  $\ell$  个 NE 拐点的格路数量为

$$|L((0, 0) \rightarrow (n, m) \mid y \geq x + b, \text{NE}(P) = \ell)| = \binom{n}{\ell} \binom{m}{\ell} - \binom{n + b - 1}{\ell} \binom{m - b + 1}{\ell}$$

如果变为双边界限制, 只需类似的将反射容斥的过程应用于拐点信息上即可:

**定理 5.6.** 令  $n, m, l, r, \ell$  是整数, 满足  $l \leq 0 \leq r, n + l \leq m \leq n + r$ . 从  $(0, 0)$  到  $(n, m)$ , 始终夹在  $y = x + l$  和  $y = x + r$  之间, 且恰有  $\ell$  个 EN 拐点的格路数量为

$$\begin{aligned} & |L((0, 0) \rightarrow (n, m) \mid x + l \leq y \leq x + r, \text{EN}(\cdot) = \ell)| \\ &= \sum_{k \in \mathbb{Z}} \left( \binom{m - k(r - l)}{\ell + k} \binom{n + k(r - l)}{\ell - k} - \binom{m - k(r - l) - r - 1}{\ell + k} \binom{n + k(r - l) + r + 1}{\ell - k} \right) \end{aligned}$$

### 5.2.3 斜率为正整数的边界

进一步, 如果斜率不为 1, 事情将变得复杂起来。在这里, 我们只讨论一种简单的情况: 计数从  $(0, 0)$  到  $(n, m)$ , 始终不低于  $y = \mu x$  且有  $\ell$  个 EN 拐点数量固定的格路。

经过了上一章中旋转法的讨论, 我们可以预见拐点信息的结构, 很有可能也是分为  $\mu$  类, 分别构造双射。

对于一条不合法的路径  $P$ , 显然其必定存在一个低于  $y = \mu x$  的拐点, 记第一个这样的拐点编号为  $I$ 。显然存在唯一的  $X \in \mathbb{Z}$  和  $r \in [1, \mu] \cap \mathbb{Z}$ , 满足  $q_I = \mu X - r$ , 那么  $I$  也是第一个满足  $y \leq \mu x - r$  的点。我们将这些不合法的  $P$  按照对应的  $r$  分为  $\mu$  类, 实际上也就是按照  $q_I \bmod \mu$  分类。

接下来对拐点信息进行变换, 也就是双射到如下的序列对。

$$\begin{array}{ccccccccccc} 0 \leq & X - p_{I-1} & \dots & X - p_I & X & p_{I+1} & \dots & p_{\ell+1} & \leq n \\ 1 \leq & q_I - q_{I-1} & \dots & q_I - q_I & q_{I+1} & \dots & q_{\ell} & & < m \end{array}$$

这个映射的一个特点是将  $I$  之前的部分进行了一定程度的“翻转”。这并不是巧合, 而是旋转法中路径旋转 180 度产生的效果。由于保证了  $\mu p_{I-1} - r < q_{I-1} < q_I = \mu X - r \leq \mu p_I - r$ , 可以验证这两个序列满足限制并且单调递增。

$$\begin{array}{ccccccc} 0 \leq & \bar{p}_1 & \dots & \bar{p}_{\ell-1} & \bar{p}_{\ell} & \bar{p}_{\ell+1} & \leq n \\ 1 \leq & \bar{q}_1 & \dots & \bar{q}_{\ell-1} & & & < m \end{array}$$

当我们想换元拐点信息时, 只需找到第一个  $I$  满足  $(\bar{p}_I, \bar{q}_I)$  在  $y = \mu x - r$  上方。不难验证, 这样找到的  $I$  和上述映射是同一个, 那么只需要做逆变换即可。

**定理 5.7.** 令  $n, m, \ell$  是非负整数,  $\mu$  是正整数, 满足  $m \geq \mu n$ . 从  $(0, 0)$  到  $(n, m)$ , 始终不低于  $y = \mu x$  且恰有  $\ell$  个 EN 拐点的格路数量为

$$|L((0, 0) \rightarrow (n, m) \mid y \geq \mu x, \text{EN}(P) = \ell)| = \binom{n}{\ell} \binom{m}{\ell} - \mu \binom{n+1}{\ell+1} \binom{m-1}{\ell-1}$$



类似的，也可以对 NE 拐点数量进行限制。

**定理 5.8.** 令  $n, m, \ell$  是非负整数， $\mu$  是正整数，满足  $m \geq \mu n$ 。从  $(0, 0)$  到  $(n, m)$ ，始终不低于  $y = \mu x$  且恰有  $\ell$  个 NE 拐点的格路数量为

$$|L((0, 0) \rightarrow (n, m) \mid y \geq \mu x, \text{NE}(P) = \ell)| = \binom{n-1}{\ell-1} \binom{m+1}{\ell} - \mu \binom{n}{\ell} \binom{m}{\ell-1}$$

### 5.2.4 不相交格路

**定理 5.9.** 令  $n, \ell$  是非负整数， $\mathbf{A} = (A_1, A_2, \dots, A_n)$ ,  $\mathbf{E} = (E_1, E_2, \dots, E_n)$  是格点序列，其中  $A_i = (a_1^{(i)}, a_2^{(i)})$ ,  $E_i = (e_1^{(i)}, e_2^{(i)})$ ，满足  $a_1, e_1$  单增， $a_2, e_2$  单减。那么从  $\mathbf{A}$  到  $\mathbf{E}$  的不交路径组，满足共有  $\ell$  个 EN 拐点的方案数为

$$\begin{aligned} & |L(\mathbf{A} \rightarrow \mathbf{E} \mid \text{non-intersecting}, \text{EN}(\mathbf{P}) = \ell)| \\ &= \sum_{\ell_1 + \dots + \ell_n = \ell} \det_{1 \leq i, j \leq n} \left( \binom{e_1^{(j)} - a_1^{(i)} - i + j}{\ell_i} \binom{e_2^{(j)} - a_2^{(i)} + i - j}{\ell_i + i - j} \right) \end{aligned}$$

其中对不交路径组  $\mathbf{P}$ ， $\text{EN}(\mathbf{P})$  表示  $\sum_{P \in \mathbf{P}} \text{EN}(P)$ 。

甚至还可以在有边界限制的情况下，求固定拐点总数的不相交路径的方案数。限于篇幅，此处不再深入讨论。

## 5.3 area(P) 与 maj(P)

### 5.3.1 $q$ -模拟

一个定理、等式或者表达式的  $q$ -模拟是指，引入形式元  $q$ ，以适当的方式将原本的内容替换为和  $q$  有关的表达式，使得当  $q \rightarrow 1$  时通常会得到原来的定理、等式或者表达式。它是一类公式的拓展方法，也可以研究组合统计量的分布。

本节的内容和  $q$ -模拟相关性较高，这里先给出一些基本的定义和结论。

**定义 5.2** ( $q$ -整数、 $q$ -阶乘、 $q$ -组合数). 对非负整数  $m \leq n$ ,

$$[n]_q = \sum_{i=0}^{n-1} q^i$$

称为  $q$ -整数；

$$[n]_q! = \prod_{i=1}^n [i]_q$$

称为  $q$ -阶乘；



$$\begin{bmatrix} n \\ m \end{bmatrix}_q = \frac{[n]_q!}{[m]_q! [n-m]_q!}$$

称为  $q$ -组合数。

**命题 5.1** ( $q$ -帕斯卡恒等式、 $q$ -二项式定理). 对正整数  $m \leq n$ ,

$$\begin{bmatrix} n \\ m \end{bmatrix}_q = q^{n-m} \begin{bmatrix} n-1 \\ m-1 \end{bmatrix}_q + \begin{bmatrix} n-1 \\ m \end{bmatrix}_q = \begin{bmatrix} n-1 \\ m-1 \end{bmatrix}_q + q^m \begin{bmatrix} n-1 \\ m \end{bmatrix}_q$$

称为  $q$ -帕斯卡恒等式;

$$\prod_{i=0}^{n-1} (1 + q^i x) = \sum_{i=0}^n q^{\binom{i}{2}} \begin{bmatrix} n \\ i \end{bmatrix}_q x^i$$

称为  $q$ -二项式定理。

### 5.3.2 $\text{area}(P)$

考虑一条从  $(0,0)$  到  $(n,m)$  的格路  $P$ , 记它与  $x=n, y=0$  围成的区域面积为  $\text{area}(P)$ , 我们希望将  $\text{area}(P)$  作为特征进行计数。

**定理 5.10.** 令  $n, m$  是非负整数, 从  $(0,0)$  到  $(n,m)$  的格路, 以  $\text{area}(P)$  作为特征的  $q$ -counting 为

$$GF(L((0,0) \rightarrow (n,m)); q^{\text{area}(P)}) = \begin{bmatrix} n+m \\ m \end{bmatrix}_q$$

证明. 显然该式在  $nm=0$  成立, 考虑归纳。假如该式对  $n < N$  或  $n \leq N, m < M$  成立, 下证该式对  $n=N, m=M$  也成立。为了方便, 记  $\text{area}(n,m) = GF(L((0,0) \rightarrow (n,m)); q^{\text{area}(P)})$ 。根据  $q$ -帕斯卡恒等式, 我们有

$$\begin{aligned} \text{area}(n,m) &= q^m \text{area}(n-1, m) + \text{area}(n, m-1) \\ &= q^m \begin{bmatrix} n+m-1 \\ m \end{bmatrix}_q + \begin{bmatrix} n+m-1 \\ m-1 \end{bmatrix}_q \\ &= \begin{bmatrix} n+m \\ m \end{bmatrix}_q \end{aligned}$$

□

### 5.3.3 $\text{maj}(P)$

另一个相关的特征是  $\text{maj}(P)$ , 其含义为  $P$  的所有 EN 拐点在  $P$  中的下标之和。

$\text{maj}(P)$  与  $\text{area}(P)$  的相关性在以下命题中体现:

命题 5.2. 对非负整数  $m \leq n$ ,

$$GF(L((0,0) \rightarrow (n,m)); q^{\text{area}(P)}) = GF(L((0,0) \rightarrow (n,m)); q^{\text{maj}(P)})$$

这意味着在自由格路中, 以  $\text{area}(P)$  和以  $\text{maj}(P)$  作为特征的  $q$ -counting 的结果是相同的。  
接下来让我们增加一个边界:

定理 5.11. 令  $n, m$  是非负整数, 从  $(0,0)$  到  $(n,m)$  的格路, 始终不低于  $y = x$ , 以  $\text{maj}(P)$  作为特征的  $q$ -counting 为

$$GF(L((0,0) \rightarrow (n,m) \mid y \geq x); q^{\text{maj}(\cdot)}) = \begin{bmatrix} n+m \\ n \end{bmatrix}_q - q \begin{bmatrix} n+m \\ n-1 \end{bmatrix}_q$$

证明. 由于我们已经知道自由路的答案, 只需将所有经过  $y = x - 1$  的路径的贡献减掉。

考虑构造一个双射  $\phi$ , 将这样的所有不合法路径  $P$  映射到从  $(0,0)$  到  $(n-1, m+1)$  的自由路, 满足  $\text{maj}(P) = \text{maj}(\phi(P)) + 1$ 。显然, 如果这样的  $\phi$  存在, 那么立刻证明了该定理。

$\phi$  的构造如下: 找到  $P$  中  $y - x$  最小的位置  $A$ , 如有多个选最靠前的一个。记  $A$  的前一个位置为  $B$ , 显然  $BA$  步长为  $(1,0)$ 。那么将这一步变为  $(0,1)$ , 便得到了  $\phi(P)$ 。

可以发现, 如果我们寻找  $\phi(P)$  中  $y - x$  最小的位置, 如有多个选最靠后的一个, 那么将会找到  $B$ 。如此一来所有满足条件的自由路都存在唯一的原像, 因此  $\phi$  是双射。

还需验证  $\text{maj}(P) = \text{maj}(\phi(P)) + 1$  总是成立。首先  $A$  显然是一个 EN 拐点, 若  $B = (0,0)$ ,  $\phi$  恰好删掉了一个下标为 1 的 EN 拐点, 显然是正确的; 否则,  $B$  的前一步也是  $(1,0)$ , 那么  $A$  将不再是 EN 拐点而  $B$  将成为 EN 拐点, 二者下标刚好差 1。  $\square$

值得一提的是, 若在定理 5.11 中满足  $n = m$ , 等号右边化为  $\frac{1}{[n+1]_q} \begin{bmatrix} 2n \\ n \end{bmatrix}_q$ , 称为  $q$ -卡特兰数。

## 6 总结

本文既包含算法竞赛中数次考察、但没有得到广泛普及的知识点, 如 LGV 引理、广义 Dyck 路、与直线的交点限制; 还包含一些在算法竞赛中少见, 但是非常适合放入的结论与算法, 如起点终点不固定的不相交路径、高维 ballot problem、拐点限制。

总的来说, 本文不算全面, 只是泛泛的介绍了格路计数中较为初等的工作。于其中, 我们也可以窥见若干内容的共性。例如 LGV 引理和反射容斥, 都需要构造双射或者对合来抵消一些贡献, 推论 4.3 正同时可以被两种方法解决。

仅以本文抛砖引玉, 希望读者可以对格路计数作进一步的思考和研究。期待有更多格路计数相关的问题在算法竞赛中出现。



## 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢北京市十一学校的汪星明老师和韩思瑶老师的关心与教导。

感谢家人的关心和支持。

感谢林卓铨、武林、周康阳等同学与我交流讨论。

## 参考文献

- [1] Masao Ishikawa and Masato Wakayama. Minor summation formulas of pfaffians, survey and a new identity. Adv. Stud. Pure Math, 28:133–142, 2000.
- [2] Ira M Gessel and Doron Zeilberger. Random walk in a weyl chamber. Proceedings of the American Mathematical Society, 115(1):27–31, 1992.
- [3] Miklós Bóna. Handbook of enumerative combinatorics, volume 87. CRC Press, 2015.
- [4] Ian P Goulden and Luis G Serrano. Maintaining the spirit of the reflection principle when the boundary has arbitrary integer slope. Journal of Combinatorial Theory, Series A, 104(2):317–326, 2003.
- [5] Johannes Furlinger and Josef Hofbauer. q-catalan numbers. Journal of Combinatorial Theory, Series A, 40(2):248–264, 1985.
- [6] John R Stembridge. Nonintersecting paths, pfaffians, and plane partitions. 1990.
- [7] Christian Krattenthaler. The enumeration of lattice paths with respect to their number of turns, 1997.
- [8] Yukiko Fukukawa. Counting generalized dyck paths. arXiv: Combinatorics, 2013.
- [9] George E Andrews, Christian Krattenthaler, Alan Krinik, et al. Lattice Path Combinatorics and Applications, volume 58. Springer, 2019.
- [10] Katherine Humphreys. A history and a survey of lattice path enumeration. Journal of statistical planning and inference, 140(8):2237–2254, 2010.



# 连分数与欧几里得算法在 OI 中的应用

华东师范大学第二附属中学 柯怿憬

## 摘要

连分数是实数作为有理数的特定收敛序列的表示，并且与欧几里得算法密切相关。本文介绍了它们在一系列数论问题以及多项式环上的应用。

## 1 引言

连分数是实数作为有理数的特定收敛序列的表示，可以有效解决有理数逼近以及与格点有关的数论问题。此外，连分数与欧几里得算法密切相关。本文将介绍它们在 OI 中的应用。

第二节将介绍连分数与欧几里得算法的性质及应用。

第三节将介绍多项式环上的欧几里得算法及其应用。

## 2 连分数

定义连分数为形如  $a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \dots}}$  的表达式，其中  $a_0 \in \mathbb{Z}$ ，对于  $i \geq 1$ ，有  $a_i \in \mathbb{Z}^+$ ，并将其记为  $[a_0; a_1, \dots]$ 。

对于实数  $r$ ，考虑计算其连分数表示，则有

$$\begin{aligned} r &= s_0 \\ s_0 &= \lfloor s_0 \rfloor + \frac{1}{s_1} \\ s_1 &= \lfloor s_1 \rfloor + \frac{1}{s_2} \\ &\vdots \\ s_{n-1} &= \lfloor s_{n-1} \rfloor + \frac{1}{s_n} \end{aligned}$$

其中对于  $i \in \{1, \dots, n\}$ ，有  $s_i > 1$ 。

令  $a_i = \lfloor s_i \rfloor$ , 则  $r = [a_0; a_1, \dots]$ , 对于  $i \geq 1$ , 有  $a_i \in \mathbb{Z}^+$ . 若存在  $N$  使得  $s_N$  为整数, 则  $r$  为有理数, 对于  $n > N$ ,  $s_n$  与  $a_n$  未定义。

称  $[a_0; a_1, \dots]$  为  $r$  的连分数表示。

## 2.1 渐近分数

定义  $r$  的第  $i$  个渐近分数为  $r_i = [a_0; a_1, \dots, a_i]$ , 记  $r_i = \frac{p_i}{q_i}$ , 其中  $p_i, q_i$  互素。大于  $r$  的渐近分数称为上渐近分数, 小于  $r$  的渐近分数称为下渐近分数。

引理 2.1.

$$\frac{p_n x + p_{n-1}}{q_n x + q_{n-1}} = a_0 + \frac{1}{a_1 + \frac{1}{\dots + \frac{1}{a_n + \frac{1}{x}}}} \quad (2.1)$$

证明. 对  $n$  归纳,  $n = 1$  时是容易的。对于  $n \geq 1$ , 令  $x \leftarrow a_{n+1}$ , 得  $\frac{p_{n+1}}{q_{n+1}} = \frac{a_{n+1}p_n + p_{n-1}}{a_{n+1}q_n + q_{n-1}}$ 。令  $x \leftarrow a_{n+1} + \frac{1}{x}$ , 则有

$$\begin{aligned} a_0 + \frac{1}{a_1 + \frac{1}{\dots + \frac{1}{a_{n+1} + \frac{1}{x}}}} &= \frac{p_n(a_{n+1} + \frac{1}{x}) + p_{n-1}}{q_n(a_{n+1} + \frac{1}{x}) + q_{n-1}} \\ &= \frac{(a_{n+1}p_n + p_{n-1})x + p_n}{(a_{n+1}q_n + q_{n-1})x + q_n} \\ &= \frac{p_{n+1}x + p_n}{q_{n+1}x + q_n} \end{aligned}$$

□

引理 2.1 表明

定理 2.1.  $p_n, q_n$  具有递推式

$$\begin{pmatrix} p_{n+1} & p_n \\ q_{n+1} & q_n \end{pmatrix} = \begin{pmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{pmatrix} \begin{pmatrix} a_{n+1} & 1 \\ 1 & 0 \end{pmatrix} \quad (2.2)$$

为了形式上一致, 补充定义  $r_{-2} = \frac{0}{1} = \frac{p_{-2}}{q_{-2}}, r_{-1} = \frac{1}{0} = \frac{p_{-1}}{q_{-1}}$ , 则有

$$\begin{pmatrix} p_0 & p_{-1} \\ q_0 & q_{-1} \end{pmatrix} = \begin{pmatrix} p_{-1} & p_{-2} \\ q_{-1} & q_{-2} \end{pmatrix} \begin{pmatrix} a_0 & 1 \\ 1 & 0 \end{pmatrix}$$

在无歧义的情况下, 默认  $a_i$  是  $r$  或  $\frac{p}{q}$  的连分数表示的第  $i$  项,  $r_i = \frac{p_i}{q_i}$  是其第  $i$  个渐近分数。若定义了连分数  $[a_0; a_1, \dots]$ , 则默认定义  $r_i = \frac{p_i}{q_i}$  为其第  $i$  个渐近分数。

推论 2.1.

$$p_{n+1}q_n - p_nq_{n+1} = \det \begin{pmatrix} p_{n+1} & p_n \\ q_{n+1} & q_n \end{pmatrix} = (-1)^n \quad (2.3)$$

证明. 只需注意到  $\det \begin{pmatrix} p_{-1} & p_{-2} \\ q_{-1} & q_{-2} \end{pmatrix} = 1$  而  $\det \begin{pmatrix} a_n & 1 \\ 1 & 0 \end{pmatrix} = -1$ . □

推论 2.1 表明, 按照式 (2.2) 递推得到的  $p_n, q_n$  确实是互素的。

**例题 2.1.** 给定  $A, B, C \in \mathbb{Z}$ , 求  $x, y \in \mathbb{Z}$ , 使得  $Ax + By = C$ 。

**解法.** 求出  $\frac{A}{B}$  的连分数表示  $[a_0; a_1, \dots, a_N]$ 。由式 (2.3),  $p_N q_{N-1} - p_{N-1} q_N = (-1)^{N-1}$ 。由于  $\frac{A}{B} = \frac{p_N}{q_N}$ , 所以  $A q_{N-1} - B p_{N-1} = (-1)^{N-1} \gcd(A, B)$ , 由此可以解出  $x, y$ 。 □

## 2.2 误差与余项

定义  $r$  的第  $i$  个余项为  $s_i = [a_i; a_{i+1}, \dots]$ , 则有

**性质 2.1.**  $r = [a_0; a_1, \dots, a_{n-1}, s_n]$

**性质 2.2.**  $s_n = \lfloor s_n \rfloor + \frac{1}{s_{n+1}} = a_n + \frac{1}{s_{n+1}}$

**引理 2.2.**

$$r - r_n = \frac{(-1)^n}{q_n(q_n s_{n+1} + q_{n-1})} \quad (2.4)$$

**证明.**

$$\begin{aligned} r - r_n &= [a_0; a_1, \dots, a_n, s_{n+1}] - r_n \\ &= \frac{p_n s_{n+1} + p_{n-1}}{q_n s_{n+1} + q_{n-1}} - \frac{p_n}{q_n} \\ &= \frac{p_{n-1} q_n - p_n q_{n-1}}{q_n(q_n s_{n+1} + q_{n-1})} \\ &= \frac{(-1)^n}{q_n(q_n s_{n+1} + q_{n-1})} \end{aligned}$$

□

由引理 2.2 可得

**推论 2.2.** 对于  $r_n \neq r$ , 当  $n$  为偶数时,  $r_n < r$ ; 当  $n$  为奇数时,  $r_n > r$ 。

**引理 2.3.** 当  $n$  为偶数时,  $r_{n+2} > r_n$ ; 当  $n$  为奇数时,  $r_{n+2} < r_n$ 。

**证明.**

$$\begin{aligned} r_{n+2} - r_n &= \frac{a_{n+2} p_{n+1} + p_n}{a_{n+2} q_{n+1} + q_n} - \frac{p_n}{q_n} \\ &= \frac{a_{n+2}(p_{n+1} q_n - p_n q_{n+1})}{q_n q_{n+2}} \\ &= \frac{a_{n+2}(-1)^n}{q_n q_{n+2}} \end{aligned}$$

□



结合推论 2.2 与引理 2.3，立刻得到

**定理 2.2.**  $r_0 < r_2 < \cdots < r_{2k} < \cdots < r < \cdots < r_{2k+1} < \cdots < r_3 < r_1$

## 2.3 欧几里得算法

欧几里得算法又称辗转相除法，对于整数  $p, q$ ，令

$$u_{-2} = p$$

$$u_{-1} = q$$

$$u_n = u_{n-2} \bmod u_{n-1}$$

设  $\frac{p}{q} = [a_0; a_1, \cdots, a_N]$ ，则最终  $u_{N-1} = \gcd(p, q)$ ， $u_N = 0$ 。由于  $u_{n+2} < \frac{u_n}{2}$ ，因此  $N = O(\log \min(p, q))$ 。

定义  $(p, q)$  的第  $i$  个欧几里得项为  $u_i$ 。

定义  $(p, q)$  的第  $i$  个欧几里得阵为  $M_i = \begin{pmatrix} p_{i+1} & p_i \\ q_{i+1} & q_i \end{pmatrix}$ 。

在无歧义的情况下，默认  $u_i$  是  $(p, q)$  的第  $i$  个欧几里得项， $M_i$  是其第  $i$  个欧几里得阵。

**性质 2.3.**  $s_n = \frac{u_{n-2}}{u_{n-1}}, a_n = [s_n] = \left\lfloor \frac{u_{n-2}}{u_{n-1}} \right\rfloor$

**性质 2.4.**  $u_n = u_{n-2} - a_n u_{n-1}$

证明. 归纳易得。 □

**定理 2.3.**

$$M_n \begin{pmatrix} u_n \\ u_{n+1} \end{pmatrix} = \begin{pmatrix} p \\ q \end{pmatrix} \quad (2.5)$$

证明. 只需注意到

$$\begin{aligned} \begin{pmatrix} p_{n+1} & p_n \\ q_{n+1} & q_n \end{pmatrix} \begin{pmatrix} u_n \\ u_{n+1} \end{pmatrix} &= \begin{pmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{pmatrix} \begin{pmatrix} a_{n+1} & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} u_n \\ u_{n+1} \end{pmatrix} \\ &= \begin{pmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{pmatrix} \begin{pmatrix} a_{n+1}u_n + u_{n+1} \\ u_n \end{pmatrix} \\ &= \begin{pmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{pmatrix} \begin{pmatrix} u_{n-1} \\ u_n \end{pmatrix} \end{aligned}$$

$$\text{且 } M_{-2} \begin{pmatrix} u_{-2} \\ u_{-1} \end{pmatrix} = \begin{pmatrix} p \\ q \end{pmatrix}. \quad \square$$

**推论 2.3.**

$$pq_n - p_nq = (-1)^n u_n \quad (2.6)$$

证明. 由式 (2.3),  $M_n^{-1} = \begin{pmatrix} p_{n+1} & p_n \\ q_{n+1} & q_n \end{pmatrix}^{-1} = (-1)^n \begin{pmatrix} q_n & -p_n \\ -q_{n+1} & p_{n+1} \end{pmatrix}$ , 结合式 (2.5) 即证.  $\square$

定义  $M_n$  对  $\begin{pmatrix} p \\ q \end{pmatrix}$  的变换为将  $\begin{pmatrix} p \\ q \end{pmatrix}$  变为  $M_n^{-1} \begin{pmatrix} p \\ q \end{pmatrix}$ , 即变为  $\begin{pmatrix} u_n \\ u_{n+1} \end{pmatrix}$ , 记作  $\begin{pmatrix} p \\ q \end{pmatrix} \xrightarrow{M_n} \begin{pmatrix} u_n \\ u_{n+1} \end{pmatrix}$ .

### 2.3.1 快速在线逆元

这是欧几里得算法的一个应用。

**引理 2.4.** 设  $M$  为整数矩阵, 且  $|\det(M)| = 1$ 。若  $p_1, p_2, q_1, q_2 \in \mathbb{Z}$  满足  $\begin{pmatrix} p_2 \\ q_2 \end{pmatrix} = M \begin{pmatrix} p_1 \\ q_1 \end{pmatrix}$ , 则  $\gcd(p_1, q_1) = \gcd(p_2, q_2)$ 。

证明. 设  $x_2, y_2 \in \mathbb{Z}$  满足  $p_2 x_2 + q_2 y_2 = \gcd(p_2, q_2)$ , 令  $\begin{pmatrix} x_1 & y_1 \end{pmatrix} = \begin{pmatrix} x_2 & y_2 \end{pmatrix} M$ , 则

$$\begin{aligned} p_1 x_1 + q_1 y_1 &= \begin{pmatrix} x_1 & y_1 \end{pmatrix} \begin{pmatrix} p_1 \\ q_1 \end{pmatrix} = \begin{pmatrix} x_2 & y_2 \end{pmatrix} M \begin{pmatrix} p_1 \\ q_1 \end{pmatrix} \\ &= \begin{pmatrix} x_2 & y_2 \end{pmatrix} \begin{pmatrix} p_2 \\ q_2 \end{pmatrix} = \gcd(p_2, q_2) \end{aligned}$$

因此  $\gcd(p_1, q_1)$  整除  $\gcd(p_2, q_2)$ 。由  $|\det(M)| = 1$  知  $M^{-1}$  也为整数矩阵, 同理得  $\gcd(p_2, q_2)$  整除  $\gcd(p_1, q_1)$ 。综上,  $\gcd(p_1, q_1) = \gcd(p_2, q_2)$ 。  $\square$

**问题 2.1.** 多次询问  $(p, q)$ , 满足  $1 \leq p, q \leq N$ , 求  $x, y \in \mathbb{Z}$  使得  $px + qy = \gcd(p, q)$ 。

考虑设定阈值  $B$ , 对于每个  $(p, q)$  满足  $1 \leq p, q \leq B$ , 预处理出  $(p, q)$  的答案。再对于每个  $(p, q)$  满足  $1 \leq q \leq p \leq B$ ,  $p \geq \sqrt{B}$ , 分别预处理  $k$  使得  $u_{k-1} \geq \sqrt{B} > u_k$ , 其中  $\{u_i\}$  为  $(p, q)$  的欧几里得项, 并计算其第  $k-1$  个欧几里得阵  $M_{k-1}$ 。这部分的时间复杂度为  $O(B^2)$ 。

**性质 2.5.** 对每个  $(p, q)$ , 预处理得的  $M_{k-1}$  中所有元素的值均不超过  $\sqrt{B}$ 。

证明. 结合式 (2.5) 与  $p, q \leq B$ 、 $u_{k-1} \geq \sqrt{B}$  易得。  $\square$

对于一组询问  $(p, q)$ , 不妨设  $p \geq q$ 。若  $p \leq B$ , 则答案已经预处理出。否则, 将  $\frac{p}{q}$  分解为  $\frac{p_0 + \lceil \frac{p}{B} \rceil p_1}{q_0 + \lceil \frac{p}{B} \rceil q_1}$ , 其中  $0 \leq p_0, q_0 < \lceil \frac{p}{B} \rceil$ 。由于  $q_1 \leq p_1 \leq B$ , 设  $(p_1, q_1)$  预处理得到的欧几里得项为  $u_{k-1}, u_k$ , 欧几里得阵为  $M_{k-1}$ , 则

$$M_{k-1}^{-1} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{bmatrix} \frac{p}{B} \\ \frac{p}{B} \end{bmatrix} M_{k-1}^{-1} \begin{pmatrix} p_1 \\ q_1 \end{pmatrix} + M_{k-1}^{-1} \begin{pmatrix} p_0 \\ q_0 \end{pmatrix} = \begin{bmatrix} \frac{p}{B} \\ \frac{p}{B} \end{bmatrix} \begin{pmatrix} u_{k-1} \\ u_k \end{pmatrix} + M_{k-1}^{-1} \begin{pmatrix} p_0 \\ q_0 \end{pmatrix}$$

令  $\begin{pmatrix} \hat{p} \\ \hat{q} \end{pmatrix} = M_{k-1}^{-1} \begin{pmatrix} p \\ q \end{pmatrix}$ , 则由引理 2.4,  $\gcd(p, q) = \gcd(\hat{p}, \hat{q})$ , 所以由  $(\hat{p}, \hat{q})$  的答案可以得到  $(p, q)$  的答案。

由性质 2.5,  $|\hat{q}| \leq \left\lfloor \frac{p}{B} \right\rfloor u_k + \sqrt{B}(p_0 + q_0) \leq \frac{3p}{\sqrt{B}}$ , 原问题被规约为了一个规模是原本  $\frac{3}{\sqrt{B}}$  的问题。因此单次询问的时间复杂度为  $O\left(\frac{\log N}{\log B}\right)$ 。

取  $B = \Theta(N^{\frac{1}{k}})$ , 则可以得到一个  $O(N^{\frac{1}{k}})$  预处理,  $O(k)$  查询的算法。

**问题 2.2.** 多次询问  $a, p$ , 满足  $1 \leq a < p \leq N$ , 求  $a$  在模  $p$  意义下的逆元。

解法. 使用问题 2.1 的解法即可。 □

## 2.4 最佳逼近

引理 2.5.

$$|p_{n+1} - q_{n+1}r| < |p_n - q_n r| \quad (2.7)$$

证明. 由式 (2.4), 可得  $|p_n - q_n r| = \frac{1}{q_n s_{n+1} + q_{n-1}}$ 。因此

$$\begin{aligned} |p_{n+1} - q_{n+1}r| &< \frac{1}{q_{n+1} + q_n} \\ &= \frac{1}{(a_{n+1} + 1)q_n + q_{n-1}} \\ &< \frac{1}{s_{n+1}q_n + q_{n-1}} = |p_n - q_n r| \end{aligned}$$

□

**定理 2.4.** 若  $\hat{q} \geq 2$ , 则  $\frac{\hat{p}}{\hat{q}}$  为  $r$  的渐近分数的充要条件为, 对任意  $(a, b) \neq (\hat{p}, \hat{q})$ ,  $1 \leq b \leq \hat{q}$ , 皆有  $|a - br| > |\hat{p} - \hat{q}r|$ 。

证明. 分两步证明。

(i) 若  $\frac{\hat{p}}{\hat{q}}$  不是  $r$  的渐近分数, 则存在  $(a, b) \neq (\hat{p}, \hat{q})$ ,  $1 \leq b \leq \hat{q}$ , 使得  $|a - br| \leq |\hat{p} - \hat{q}r|$ 。

若  $r = \frac{p}{q}$  为有理数且  $\hat{q} \geq q$ , 则取  $a = p, b = q$  即可。

否则, 设  $q_n \leq \hat{q} < q_{n+1}$ , 取  $x, y$  使得  $\begin{pmatrix} \hat{p} \\ \hat{q} \end{pmatrix} = \begin{pmatrix} p_{n+1} & p_n \\ q_{n+1} & q_n \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ , 则

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} p_{n+1} & p_n \\ q_{n+1} & q_n \end{pmatrix}^{-1} \begin{pmatrix} \hat{p} \\ \hat{q} \end{pmatrix} = (-1)^n \begin{pmatrix} q_n & -p_n \\ -q_{n+1} & p_{n+1} \end{pmatrix} \begin{pmatrix} \hat{p} \\ \hat{q} \end{pmatrix}$$

因此  $x, y$  均为整数。

- 若  $x = 0$ , 则  $\begin{pmatrix} \hat{p} \\ \hat{q} \end{pmatrix} = y \begin{pmatrix} p_n \\ q_n \end{pmatrix}$ , 取  $a = p_n, b = q_n$  即可。
- 否则, 由条件知  $x, y \neq 0$  且  $x, y$  异号, 又由于  $p_n - q_n r$  与  $p_{n+1} - q_{n+1} r$  异号, 因此

$$|\hat{p} - \hat{q}r| = |x| \cdot |p_{n+1} - q_{n+1}r| + |y| \cdot |p_n - q_n r| > |p_n - q_n r|$$

取  $a = p_n, b = q_n$  即可。

(ii) 若  $\frac{\hat{p}}{\hat{q}}$  是  $r$  的渐近分数, 则对任意  $(a, b) \neq (\hat{p}, \hat{q}), 1 \leq b \leq \hat{q}$ , 皆有  $|a - br| > |\hat{p} - \hat{q}r|$ 。

设  $\hat{p} = p_n, \hat{q} = q_n$ 。不妨只考虑  $b < q_n$  的情形, 由 (i) 知存在  $k < n$  使得  $|p_k - q_k r| \leq |a - br|$ , 结合式 (2.7), 有

$$|a - br| \geq |p_k - q_k r| > |p_n - q_n r| = |\hat{p} - \hat{q}r|$$

□

定义  $r$  的中间分数为形如  $[a_0; a_1, \dots, a_{i-1}, t]$  且  $1 \leq t \leq a_i$  的数。大于  $r$  的中间分数称为上中间分数, 小于  $r$  的中间分数称为下中间分数。

**引理 2.6.** 对于  $r$  的上中间分数  $\frac{\hat{p}_1}{\hat{q}_1}$  与  $\frac{\hat{p}_2}{\hat{q}_2}$ , 若  $\hat{q}_1 < \hat{q}_2$ , 则  $\frac{\hat{p}_1}{\hat{q}_1} > \frac{\hat{p}_2}{\hat{q}_2}$ 。  
对于  $r$  的下中间分数  $\frac{\hat{p}_1}{\hat{q}_1}$  与  $\frac{\hat{p}_2}{\hat{q}_2}$ , 若  $\hat{q}_1 < \hat{q}_2$ , 则  $\frac{\hat{p}_1}{\hat{q}_1} < \frac{\hat{p}_2}{\hat{q}_2}$ 。

证明. 不妨只考虑上中间分数的情形。

设  $\frac{\hat{p}_1}{\hat{q}_1} = \frac{t_1 p_m + p_{m-1}}{t_1 q_m + q_{m-1}}, \frac{\hat{p}_2}{\hat{q}_2} = \frac{t_2 p_n + p_{n-1}}{t_2 q_n + q_{n-1}}$ , 则要么  $m < n$ , 要么  $m = n$  且  $t_1 < t_2$ , 由定理 2.2, 均可得  $\frac{\hat{p}_1}{\hat{q}_1} > \frac{\hat{p}_2}{\hat{q}_2}$ 。 □

定义  $\frac{\hat{p}}{\hat{q}}$  为  $r$  的最佳逼近, 当且仅当对任意  $(a, b) \neq (\hat{p}, \hat{q}), 1 \leq b \leq \hat{q}$ , 皆有  $\left| r - \frac{a}{b} \right| > \left| r - \frac{\hat{p}}{\hat{q}} \right|$ 。

定义  $\frac{\hat{p}}{\hat{q}}$  为  $r$  的上最佳逼近, 当且仅当  $\frac{\hat{p}}{\hat{q}} \geq r$ , 且对任意  $(a, b) \neq (\hat{p}, \hat{q}), 1 \leq b \leq \hat{q}, \frac{a}{b} \geq r$ , 皆有  $\left| r - \frac{a}{b} \right| > \left| r - \frac{\hat{p}}{\hat{q}} \right|$ 。类似地定义下最佳逼近。

**引理 2.7.** 对于  $u = (x_1, y_1), v = (x_2, y_2), w = (x, y), x_1, x_2, x, y_1, y_2, y \in \mathbb{Z}, x_1, x_2, x \geq 1, \frac{y_1}{x_1} < \frac{y_2}{x_2}$  且  $u \times v = 1$ , 设  $w = au + bv$ , 则  $\frac{y_1}{x_1} < \frac{y}{x} < \frac{y_2}{x_2}$  的充要条件为  $a, b$  均为正整数。

证明. 只需注意到  $a = a \cdot u \times v = w \times v$  且  $b = b \cdot u \times v = u \times w$ 。而  $\frac{y_1}{x_1} < \frac{y}{x} < \frac{y_2}{x_2}$  当且仅当  $u \times w > 0$  且  $w \times v > 0$ 。 □

由引理 2.7 立刻得到

**推论 2.4.** 对于  $\frac{y_1}{x_1}, \frac{y_2}{x_2}$ ,  $x_1, x_2, y_1, y_2 \in \mathbb{Z}$ ,  $x_1, x_2 \geq 1$ , 若  $\frac{y}{x}$  满足  $x, y \in \mathbb{Z}$ ,  $x \geq 1$  且  $\frac{y_1}{x_1} < \frac{y}{x} < \frac{y_2}{x_2}$ , 则  $x \geq x_1 + x_2$ 。

**引理 2.8.** 设  $r$  的中间分数  $\frac{\hat{p}}{\hat{q}} = \frac{tp_n + p_{n-1}}{tq_n + q_{n-1}}$ ,  $0 \leq t < a_{n+1}$ , 则对任意  $(a, b) \neq (\hat{p}, \hat{q})$ ,  $\hat{q} \leq b < \hat{q} + q_n$ ,  $\frac{a}{b}$  与  $\frac{\hat{p}}{\hat{q}}$  位于  $r$  同侧, 皆有  $\left| r - \frac{a}{b} \right| \geq \left| r - \frac{\hat{p}}{\hat{q}} \right|$ 。

证明. 考虑反证, 假设  $\left| r - \frac{a}{b} \right| < \left| r - \frac{\hat{p}}{\hat{q}} \right|$ , 设  $\frac{a}{b}$  介于中间分数  $\frac{t'p_m + p_{m-1}}{t'q_m + q_{m-1}}$  与  $\frac{(t'+1)p_m + p_{m-1}}{(t'+1)q_m + q_{m-1}}$  之间, 则结合引理 2.6 得  $(t'+1)q_m + q_{m-1} \geq \hat{q} + q_n$ 。

令  $u = (t'q_m + q_{m-1}, t'p_m + p_{m-1})$ , 由于  $u \times (u + (q_m, p_m)) = 1$ , 应用推论 2.4, 得  $b > (t'+1)q_m + q_{m-1} \geq \hat{q} + q_n$ : 矛盾!  $\square$

**定理 2.5.**  $\frac{\hat{p}}{\hat{q}}$  是  $r$  的上(下)最佳逼近, 当且仅当  $\frac{\hat{p}}{\hat{q}}$  是  $r$  的上(下)中间分数或  $\frac{\hat{p}}{\hat{q}} = r$ 。

证明. 不妨只考虑  $\frac{\hat{p}}{\hat{q}} > r$  的情形。同样分两步证明。

(i) 若  $\frac{\hat{p}}{\hat{q}}$  不是  $r$  的上中间分数, 则存在  $(a, b) \neq (\hat{p}, \hat{q})$ ,  $1 \leq b \leq \hat{q}$ ,  $\frac{a}{b} \geq r$ , 使得  $\left| r - \frac{a}{b} \right| \leq \left| r - \frac{\hat{p}}{\hat{q}} \right|$ 。

若  $r = \frac{p}{q}$  为有理数且  $\hat{q} \geq q$ , 则取  $a = p, b = q$  即可。

否则, 设  $tq_n + q_{n-1} \leq \hat{q} < (t+1)q_n + q_{n-1}$ ,  $n$  为偶数,  $0 \leq t < a_{n+1}$ 。由推论 2.2,  $\frac{tp_n + p_{n-1}}{tq_n + q_{n-1}}$  为  $r$  的上中间分数。取  $a = tp_n + p_{n-1}, b = tq_n + q_{n-1}$ , 应用引理 2.8, 有  $\left| r - \frac{a}{b} \right| \leq \left| r - \frac{\hat{p}}{\hat{q}} \right|$ 。

(ii) 若  $\frac{\hat{p}}{\hat{q}}$  是  $r$  的上中间分数, 则对任意  $(a, b) \neq (\hat{p}, \hat{q})$ ,  $1 \leq b \leq \hat{q}$ ,  $\frac{a}{b} \geq r$ , 皆有  $\left| r - \frac{a}{b} \right| > \left| r - \frac{\hat{p}}{\hat{q}} \right|$ 。

不妨只考虑  $b < \hat{q}$  的情形, 由 (i) 知存在上中间分数  $\frac{\tilde{p}}{\tilde{q}}$  使得  $\tilde{q} \leq b < \hat{q}$  且  $\left| r - \frac{\tilde{p}}{\tilde{q}} \right| \leq \left| r - \frac{a}{b} \right|$ 。结合引理 2.6, 有

$$\left| r - \frac{a}{b} \right| \geq \left| r - \frac{\tilde{p}}{\tilde{q}} \right| > \left| r - \frac{\hat{p}}{\hat{q}} \right|$$

$\square$

**推论 2.5.**  $r$  的最佳逼近一定是  $r$  的中间分数。

**例题 2.2.** 给定有理数  $\frac{A}{B}$  与正整数  $N$ , 求  $(x, y)$  使得  $1 \leq x \leq N$  且  $\left| \frac{A}{B} - \frac{y}{x} \right|$  最小。

**解法.** 只需考虑  $\frac{A}{B}$  的横坐标最接近  $N$  的上中间分数与下中间分数即可。  $\square$

定义  $\frac{\hat{p}}{\hat{q}}$  为  $r$  的严格上最佳逼近, 当且仅当  $\frac{\hat{p}}{\hat{q}} > r$ , 且对任意  $(a, b) \neq (\hat{p}, \hat{q})$ ,  $1 \leq b \leq \hat{q}$ ,  $\frac{a}{b} > r$ , 皆有  $\left| r - \frac{a}{b} \right| > \left| r - \frac{\hat{p}}{\hat{q}} \right|$ 。类似地定义严格下最佳逼近。

与上(下)最佳逼近不同, 严格上(下)最佳逼近去除了  $y = rx$  上的整点。若  $r$  为无理数, 严格上(下)最佳逼近显然与上(下)最佳逼近等价。

**定理 2.6.**  $\frac{\hat{p}}{\hat{q}}$  是  $r = \frac{p}{q}$  的严格上(下)最佳逼近, 当且仅当  $\frac{\hat{p}}{\hat{q}}$  是  $r$  的上(下)中间分数, 或  $\frac{\hat{p}}{\hat{q}} = \frac{\tilde{p} + tp}{\tilde{q} + tq}$ , 其中  $\frac{\tilde{p}}{\tilde{q}}$  是  $r$  的横坐标最大的上(下)中间分数,  $t \in \mathbb{Z}^+$ 。

证明. 略, 类似定理 2.5 的证明可以得证。□

**例题 2.3 (Good Bitstrings<sup>1</sup>).** 按如下方式定义  $\text{gen\_string}(a, b)$  函数:

```
string gen_string(int64_t a, int64_t b) {
 string res;
 int64_t ia = 0, ib = 0;
 while (ia + ib < a + b) {
 if ((__int128)ia * b <= (__int128)ib * a) {
 res += '0';
 ia++;
 } else {
 res += '1';
 ib++;
 }
 }
 return res;
}
```

一个 01 串  $s$  是好的, 当且仅当存在正整数  $x, y$  使  $s = \text{gen\_string}(x, y)$ 。

给定正整数  $A, B$ , 求  $\text{gen\_string}(A, B)$  有多少前缀是好的。

**解法.** 问题等价于求有多少个  $(x, y)$  满足  $1 \leq x \leq A, 1 \leq y \leq B$  且对任意  $1 \leq x' \leq x, x' + y' < x + y$ , 有  $\left\lfloor \frac{y'}{x'} < \frac{B}{A} \right\rfloor = \left\lfloor \frac{y'}{x'} < \frac{y}{x} \right\rfloor$ 。

可以发现, 若  $\frac{y}{x} < \frac{B}{A}$ , 则  $(x, y)$  合法当且仅当  $\frac{y}{x}$  是  $\frac{B}{A}$  的严格下最佳逼近; 若  $\frac{y}{x} \geq \frac{B}{A}$ , 则  $(x, y)$  合法当且仅当  $\frac{y}{x}$  是  $\frac{B}{A}$  的上最佳逼近, 或  $(y, x) = t(p_n, q_n)$ , 其中  $\frac{p_n}{q_n}$  是  $\frac{B}{A}$  的渐近分数,  $\frac{p_n}{q_n} \geq \frac{B}{A}$  且  $tq_n \leq q_n + q_{n+1}$ 。两种情况都容易统计。□

## 2.5 几何性质

**定理 (Pick [1]).** 对于简单格点多边形, 设其面积为  $A$ , 边界上有  $B$  个整点, 内部有  $I$  个整点, 则有  $A = I + \frac{1}{2}B - 1$ 。

<sup>1</sup>USACO 2023 Open, Platinum



对于  $r$ , 定义  $r_i = (q_i, p_i)$ , 则有  $r_{n+1} = a_{n+1}r_n + r_{n-1}$ 。

**定理 2.7.**  $r_{-2}, r_0, \dots, r_{2k}, \dots$  构成  $y = rx$  下方的上凸包。

$r_{-1}, r_1, \dots, r_{2k+1}, \dots$  构成  $y = rx$  上方的下凸包。

且两个凸包之间的空间中没有整点。

**证明.** 由定理 2.2 易得该图形为凸包。

观察  $(0, 0), r_n, r_{n+2}$  围成的三角形, 其面积为  $\frac{1}{2}|r_n \times r_{n+2}| = \frac{1}{2}|r_n \times (a_{n+2}r_{n+1} + r_n)| = \frac{a_{n+2}}{2}$ , 其边界上的整点为  $(0, 0)$  与  $tr_{n+1} + r_n$ , 其中  $0 \leq t \leq a_{n+2}$ , 共  $a_{n+2} + 2$  个。结合 Pick 定理, 知该三角形内部没有整点。  $\square$

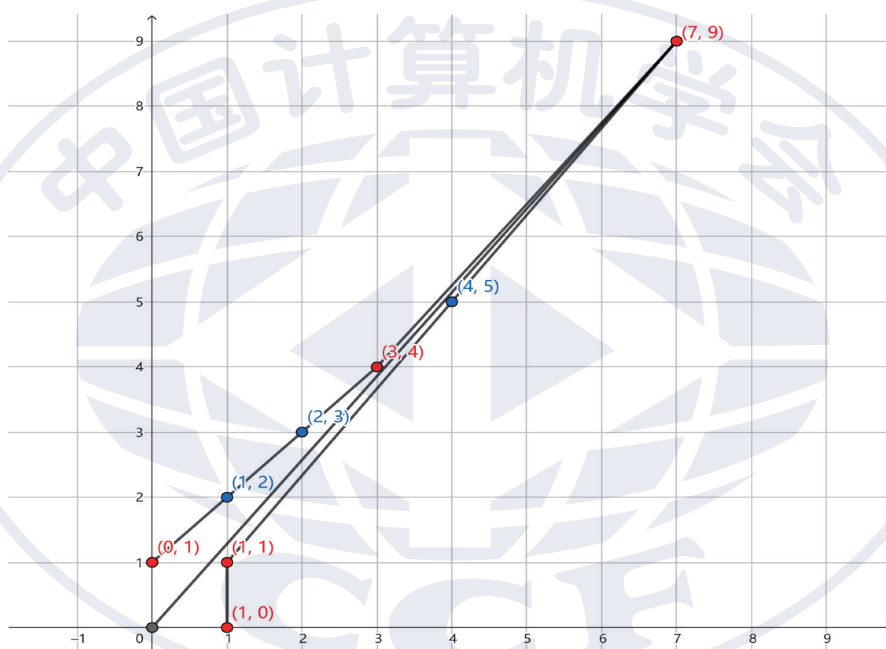


图 2.1:  $r = \frac{7}{9}$  时的例子

如图 2.1, 红点的斜率为  $r$  的渐近分数, 蓝点的斜率为  $r$  其他的中间分数, 上渐近分数与下渐近分数对应的格点分别构成凸包。

### 2.5.1 正比例函数下方的凸包

**问题 2.3.** 给定正整数  $A, B, N$ , 对于所有整点  $(x, y)$  满足  $0 \leq x \leq N$ ,  $y \leq \frac{A}{B}x$ , 求它们构成的上凸包。

按横坐标从小往大构建凸包, 假设当前凸包的最后一个点为  $(x_k, y_k)$ , 设下一个点为  $(x_k + \Delta x, y_k + \Delta y)$ , 则需要满足  $\Delta x \leq N - x_k$  且  $\frac{\Delta y}{\Delta x}$  最大。注意到一定有  $\frac{\Delta y}{\Delta x} \leq \frac{A}{B}$ , 因此只需最小化  $\left| \frac{A}{B} - \frac{\Delta y}{\Delta x} \right|$ 。现在该问题转化为了例题 2.2 的形式, 所以  $\frac{\Delta y}{\Delta x}$  一定是  $\frac{A}{B}$  的中间分数。  
可以写出如下伪代码:

---

**Algorithm 1** Convex below  $y = \frac{A}{B}x$

---

**Input:**  $A, B, N$

**Output:**  $y = \frac{A}{B}x$  下方的凸包  $\{(x_i, y_i)\}$

---

```

1: $k \leftarrow 0$
2: $(x_0, y_0) \leftarrow (0, 0)$
3: 计算 $\frac{A}{B}$ 的渐近分数 $\frac{p_{-2}}{q_{-2}}, \dots, \frac{p_n}{q_n}$
4: for i from n down to -2 do
5: if $Aq_i \geq Bp_i$ then
6: while $x_k + q_i \leq N$ do
7: if $i = n$ then
8: $(\Delta x, \Delta y) \leftarrow (q_i, p_i)$
9: else
10: $t \leftarrow \left\lfloor \frac{N - q_i - x_k}{q_{i+1}} \right\rfloor$
11: $(\Delta x, \Delta y) \leftarrow (q_i, p_i) + t(q_{i+1}, p_{i+1})$
12: end if
13: $c \leftarrow \left\lfloor \frac{N - x_k}{\Delta x} \right\rfloor$
14: $(x_{k+1}, y_{k+1}) \leftarrow (x_k, y_k) + c(\Delta x, \Delta y)$
15: $k \leftarrow k + 1$
16: end while
17: end if
18: end for
19: return $\{(x_0, y_0), \dots, (x_k, y_k)\}$

```

---

注意到凸包中每增加一个点,  $N - x_k$  至少减半, 因此有

**性质 2.6.** 凸包上的点数为  $O(\log N)$  级别。

### 2.5.2 一次函数下方的凸包

**问题 2.4.** 给定正整数  $A, B, C, N$ , 对于所有整点  $(x, y)$  满足  $0 \leq x \leq N$ ,  $y \leq \frac{Ax + C}{B}$ , 求它们构成的凸包。

**定理 2.8.** 设问题 2.4 所求的凸包为  $\{(x_i, y_i)\}$ , 则  $\frac{y_{k+1} - y_k}{x_{k+1} - x_k}$  一定是  $\frac{A}{B}$  的中间分数。

证明. 设  $(\hat{x}, \hat{y})$  为满足  $0 \leq x \leq N$ ,  $y \leq \frac{Ax + C}{B}$  且  $By - Ax$  最大的点, 则  $(\hat{x}, \hat{y})$  一定在凸包上。考虑从  $(\hat{x}, \hat{y})$  分别向前和向后构建凸包。

向前构建凸包时, 假设当前凸包的第一个点为  $(x_k, y_k)$ , 设前一个点为  $(x_k - \Delta x, y_k - \Delta y)$ , 则需要满足  $\Delta x \leq x_k$  且  $\frac{\Delta y}{\Delta x}$  最小。注意到一定有  $\frac{\Delta y}{\Delta x} \geq \frac{A}{B}$ , 因此只需最小化  $\left| \frac{A}{B} - \frac{\Delta y}{\Delta x} \right|$ 。

向后构建凸包时, 假设当前凸包的最后一个点为  $(x_k, y_k)$ , 设后一个点为  $(x_k + \Delta x, y_k + \Delta y)$ , 则需要满足  $\Delta x \leq N - x_k$  且  $\frac{\Delta y}{\Delta x}$  最大。注意到一定有  $\frac{\Delta y}{\Delta x} \leq \frac{A}{B}$ , 因此也只需最小化  $\left| \frac{A}{B} - \frac{\Delta y}{\Delta x} \right|$ 。

两种情况都转化为了例题 2.2 的形式, 所以  $\frac{\Delta y}{\Delta x}$  一定是  $\frac{A}{B}$  的中间分数。□

回到原问题, 依旧考虑按横坐标从小到大构建凸包。可以写出如下伪代码:

---

**Algorithm 2** Convex below  $y = \frac{Ax + C}{B}$

---

**Input:**  $A, B, C, N$

**Output:**  $y = \frac{Ax + C}{B}$  下方的凸包  $\{(x_i, y_i)\}$

---

```

1: $k \leftarrow 0$
2: $(x_0, y_0) \leftarrow (0, \lfloor \frac{C}{B} \rfloor)$
3: 计算 $\frac{A}{B}$ 的渐近分数 $\frac{p_{-2}}{q_{-2}}, \dots, \frac{p_n}{q_n}$
4: for i from -2 to $n - 2$ do
5: if $Aq_i < Bp_i$ then
6: while $y_k + p_{i+2} \leq \frac{A(x_k + q_{i+2}) + C}{B}$ do
7: $t \leftarrow \max \left(\left\lceil \frac{B(y_k + p_i) - A(x_k + q_i) - C}{Aq_{i+1} - Bp_{i+1}} \right\rceil, 0 \right)$
8: $(\Delta x, \Delta y) \leftarrow (q_i, p_i) + t(q_{i+1}, p_{i+1})$
9: if $x_k + \Delta x > N$ or $A\Delta x \geq B\Delta y$ then
10: break
11: end if
12: $c \leftarrow \min \left(\left\lfloor \frac{N - x_k}{\Delta x} \right\rfloor, \left\lfloor \frac{A x_k - B y_k + C}{B \Delta y - A \Delta x} \right\rfloor \right)$
13: $(x_{k+1}, y_{k+1}) \leftarrow (x_k, y_k) + c(\Delta x, \Delta y)$
14: $k \leftarrow k + 1$
15: end while
16: end if
17: end for
18: for i from n down to -2 do
19: if $Aq_i \geq Bp_i$ then
20: while $x_k + q_i \leq N$ do

```

---

```

21: if $i = n$ then
22: $(\Delta x, \Delta y) \leftarrow (q_i, p_i)$
23: else
24: $t \leftarrow \left\lfloor \frac{N - q_i - x_k}{q_{i+1}} \right\rfloor$
25: $(\Delta x, \Delta y) \leftarrow (q_i, p_i) + t(q_{i+1}, p_{i+1})$
26: end if
27: $c \leftarrow \left\lfloor \frac{N - x_k}{\Delta x} \right\rfloor$
28: $(x_{k+1}, y_{k+1}) \leftarrow (x_k, y_k) + c(\Delta x, \Delta y)$
29: $k \leftarrow k + 1$
30: end while
31: end if
32: end for
33: return $\{(x_0, y_0), \dots, (x_k, y_k)\}$

```

性质 2.7. 凸包上的点数为  $O(\log N)$  级别。

证明. 设  $(\hat{x}, \hat{y})$  为满足  $0 \leq x \leq N$  且  $By - Ax$  最大的点, 将凸包从  $(\hat{x}, \hat{y})$  划分为前后两部分, 分别应用性质 2.6 即可。□

例题 2.4 (Crime and Punishment<sup>2</sup>). 给定正整数  $A, B, N$ , 求  $(x, y)$  使得  $x, y \leq 0$ ,  $Ax + By \leq N$  且  $Ax + By$  最大。

解法. 令  $x = \left\lfloor \frac{N}{A} \right\rfloor - x'$ , 则问题转化为要求  $0 \leq x' \leq \left\lfloor \frac{N}{A} \right\rfloor$ ,  $By - Ax' \leq N \bmod A$ , 求  $By - Ax'$  的最大值。用斜率为  $\frac{A}{B}$  的直线去切  $y = \frac{Ax + N \bmod A}{B}$  下方的凸包即可。□

例题 2.5 (Maximum Sine<sup>3</sup>). 给定  $l, r, p, q$ , 令  $f(x) = \left| \sin\left(\frac{p}{q}\pi x\right) \right|$ . 求最小的整数  $x \in [l, r]$  使得  $f(x)$  最大。

解法. 问题可以转化为求  $x$  使得  $\frac{2px + q}{2q}$  最接近整数, 只需求出  $(2px + q) \bmod 2q$  的最大值与最小值。

考虑给定  $A, B, C, l, r$ , 如何求  $\min_{x \in [l, r]} \{(Ax + C) \bmod B\}$ . 设  $Ax + C = By + r$ , 则要求  $By \leq Ax + C$ , 最小化  $Ax - By$ . 因此只需用斜率为  $\frac{A}{B}$  的直线去切横坐标在  $[l, r]$  中,  $y = \frac{Ax + C}{B}$  下方的凸包即可。□

<sup>2</sup>Timus Online Judge 1430

<sup>3</sup>CF1182F

### 3 多项式环上的欧几里得算法

对于多项式  $A, B$ , 若多项式  $Q, R$  满足  $A = BQ + R$  且  $\deg R < \deg B$ , 则记  $\left\lfloor \frac{A}{B} \right\rfloor = Q$ ,  $A \bmod B = R$ 。

对于多项式  $F$ , 定义  $F^R = x^{\deg F} F(x^{-1})$ 。

**引理 3.1.** 对于多项式  $A, B$ ,  $\deg A \geq \deg B$ , 有

$$\left\lfloor \frac{A}{B} \right\rfloor^R \equiv A^R (B^R)^{-1} \pmod{x^{\deg A - \deg B + 1}}$$

证明. 设  $\deg A = a, \deg B = b, \deg R = r$ , 则有  $r < b$ , 因此  $a - r \geq a - b + 1$ 。

$$\begin{aligned} A &= BQ + R \\ \Rightarrow BQ &= A - R \\ \Rightarrow B^R Q^R &= A^R - x^{a-r} R^R \\ \Rightarrow Q^R &= A^R (B^R)^{-1} - x^{a-r} (B^R)^{-1} R^R \\ \Rightarrow Q^R &\equiv A^R (B^R)^{-1} \pmod{x^{a-b+1}} \end{aligned}$$

□

**问题 3.1.** 给定多项式  $P, Q$ , 求  $\gcd(P, Q)$ 。

考虑欧几里得算法, 计算  $(P, Q)$  的欧几里得项, 最终  $u_{N-1} = \gcd(P, Q)$ ,  $u_N = 0$ 。

设  $\max(\deg P, \deg Q) = n$ , 若朴素实现该算法, 由于  $N = O(n)$ , 时间复杂度为  $O(n^2)$ , 考虑如何更加高效地计算。

#### 3.1 Half-GCD

类似有理数域, 同样地定义多项式环上的连分数。

**性质 3.1.** 对于多项式  $P, Q$ ,  $\deg P \geq \deg Q$ , 设  $(P, Q)$  的欧几里得项为  $\{u_i\}$ ,  $\frac{P}{Q} = [a_0; a_1, \dots]$ , 则有

$$\deg p_n = \deg a_0 + \dots + \deg a_n \quad (3.1a)$$

$$\deg q_n = \deg a_1 + \dots + \deg a_n \quad (3.1b)$$

$$\begin{aligned} \deg u_n &= \deg P - \deg a_0 - \dots - \deg a_{n+1} \\ &= \deg P - \deg p_{n+1} \\ &= \deg Q - \deg q_{n+1} \end{aligned} \quad (3.1c)$$

证明. 由定理 2.1、性质 2.3 易得。

□

**定理 3.1.** 对于多项式  $P_1, Q_1$ ,  $\deg P_1 \geq \deg Q_1$ , 设  $(P_1, Q_1)$  的欧几里得项为  $\{u_i^{(1)}\}$ ,  $\frac{P_1}{Q_1} = [a_0; a_1, \dots]$ 。若  $k$  满足  $\deg u_{k-1}^{(1)} \geq \frac{\deg P_1}{2}$ , 则对任意多项式  $P_0, Q_0$  以及  $t > \max(\deg P_0, \deg Q_0)$ ,  $a_0, a_1, \dots, a_k$  恰好是  $\frac{P_0 + x^t P_1}{Q_0 + x^t Q_1}$  的连分数表示的前  $k+1$  项。

证明. 令  $P = P_0 + x^t P_1, Q = Q_0 + x^t Q_1$ , 设  $(P, Q)$  的欧几里得项为  $\{u_i\}$ 。

对  $k$  归纳, 若  $\deg u_{k-1}^{(1)} \geq \frac{\deg P_1}{2}$ ,  $a_0, \dots, a_{k-1}$  恰好是  $\frac{P}{Q}$  的连分数表示的前  $k$  项, 由

$$\begin{aligned} \deg P_1 &\leq 2 \deg u_{k-1}^{(1)} \\ \Rightarrow \deg Q_0 + \deg P_1 &< 2 \deg u_{k-1}^{(1)} + t \\ \Rightarrow \deg Q_0 + (\deg P_1 - \deg u_{k-2}^{(1)}) &< \deg u_{k-1}^{(1)} - (\deg u_{k-2}^{(1)} - \deg u_{k-1}^{(1)}) + t \\ \Rightarrow \deg Q_0 + \deg p_{k-1} &< \deg u_{k-1}^{(1)} - \deg a_k + t \end{aligned}$$

得  $\deg(Q_0 p_{k-1}) < \deg(x^t u_{k-1}^{(1)}) - \deg a_k$ 。

同理  $\deg(P_0 q_{k-1}) < \deg(x^t u_{k-1}^{(1)}) - \deg a_k$ 。

由式 (2.6)

$$\begin{aligned} u_{k-1} &= (-1)^{k-1} (P q_{k-1} - Q p_{k-1}) \\ &= (-1)^{k-1} (P_0 q_{k-1} - Q_0 p_{k-1} + x^t (P_1 q_{k-1} - Q_1 p_{k-1})) \\ &= (-1)^{k-1} (P_0 q_{k-1} - Q_0 p_{k-1}) + x^t u_{k-1}^{(1)} \end{aligned}$$

因此  $u_{k-1}^R \equiv (x^t u_{k-1}^{(1)})^R \equiv (u_{k-1}^{(1)})^R \pmod{x^{\deg a_k + 1}}$ 。

类似地, 有  $u_{k-2}^R \equiv (u_{k-2}^{(1)})^R \pmod{x^{\deg a_k + 1}}$ 。

结合引理 3.1, 有  $\left\lfloor \frac{u_{k-2}}{u_{k-1}} \right\rfloor = \left\lfloor \frac{u_{k-2}^{(1)}}{u_{k-1}^{(1)}} \right\rfloor = a_k$ 。 □

由定理 3.1, 只需对  $P, Q$  的高位执行欧几里得算法, 便可得  $\frac{P}{Q}$  的连分数表示的前若干项, 进而得到  $(P, Q)$  的前若干个欧几里得项。

据此, 可以设计函数 Half-GCD( $P, Q$ ), 其中  $P, Q$  为多项式,  $\deg P \geq \deg Q$ 。Half-GCD( $P, Q$ ) 求解  $\frac{P}{Q}$  的连分数表示的前  $k+1$  项  $\{a_0, \dots, a_k\}$  以及第  $k-1$  个欧几里得阵  $M_{k-1}$ 。令  $\begin{pmatrix} P \\ Q \end{pmatrix} \xrightarrow{M_{k-1}} \begin{pmatrix} u_{k-1} \\ u_k \end{pmatrix}$ , 要求  $k$  满足  $\deg u_{k-1} \geq \frac{\deg P}{2} > \deg u_k$ 。

先取定参数  $m_1$ , 将  $\frac{P}{Q}$  分解为  $\frac{P_0 + x^{m_1} P_1}{Q_0 + x^{m_1} Q_1}$ 。调用 Half-GCD( $P_1, Q_1$ ) 得到  $\{a_0, \dots, a_k\}$  以及  $M_{k-1}$ 。由定理 3.1,  $a_0, \dots, a_k$  也是  $\frac{P}{Q}$  的连分数表示的前  $k+1$  项, 因此  $M_{k-1}$  也是  $(P, Q)$  的第  $k-1$  个欧几里得阵, 令  $\begin{pmatrix} P \\ Q \end{pmatrix} \xrightarrow{M_{k-1}} \begin{pmatrix} u_{k-1} \\ u_k \end{pmatrix}$ 。



若  $u_k < \frac{\deg P}{2}$ , 则问题已经解决。否则, 计算  $a_{k+1} = \left\lfloor \frac{u_{k-1}}{u_k} \right\rfloor$ ,  $u_{k+1} = u_{k-1} \bmod u_k$ 。再取定参数  $m_2$ , 将  $\frac{u_k}{u_{k+1}}$  分解为  $\frac{P'_0 + x^{m_2} P'_1}{Q'_0 + x^{m_2} Q'_1}$ 。再次调用  $\text{Half-GCD}(P'_1, Q'_1)$  得到  $\{a'_0, \dots, a'_{k'}\}$  以及  $M'_{k'-1}$ 。  $M'_{k'-1}$  也是  $(u_k, u_{k+1})$  的第  $k' - 1$  个欧几里得阵, 令  $\begin{pmatrix} u_k \\ u_{k+1} \end{pmatrix} \xrightarrow{M'_{k'-1}} \begin{pmatrix} u'_{k'-1} \\ u'_{k'} \end{pmatrix}$ 。取  $m_2 = \deg P - \deg u_k$ , 即可保证

$$\begin{aligned} \deg u'_{k'} &< m_2 + \frac{\deg P'_1}{2} = \frac{\deg u_k + m_2}{2} = \frac{\deg P}{2} \\ \deg u'_{k'-1} &\geq m_2 + \frac{\deg P'_1}{2} = \frac{\deg u_k + m_2}{2} = \frac{\deg P}{2} \end{aligned}$$

注意到

$$\begin{pmatrix} P \\ Q \end{pmatrix} \xrightarrow{M_{k-1}} \begin{pmatrix} u_{k-1} \\ u_k \end{pmatrix} \xrightarrow{\begin{pmatrix} a_{k+1} & 1 \\ 1 & 0 \end{pmatrix}} \begin{pmatrix} u_k \\ u_{k+1} \end{pmatrix} \xrightarrow{M'_{k'-1}} \begin{pmatrix} u'_{k'-1} \\ u'_{k'} \end{pmatrix}$$

因此只需返回  $\{a_0, \dots, a_{k+1}, a'_0, \dots, a'_{k'}\}$  以及  $M_{k-1} \begin{pmatrix} a_{k+1} & 1 \\ 1 & 0 \end{pmatrix} M'_{k'-1}$ 。

设  $\deg P = n$ , 则时间复杂度为  $T(n) = T(n - m_1) + T(m_1) + M(n)$ , 其中  $M(n)$  为两个次数为  $O(n)$  的多项式相乘的复杂度, 这里认为  $M(n) = O(n \log n)$ 。取  $m_1 = \left\lfloor \frac{n}{2} \right\rfloor + 1$ , 则  $T(n) = O(n \log^2 n)$ 。

可以写出如下伪代码:

---

**Algorithm 3** Half-GCD

---

**Input:**  $P, Q$

**Output:**  $\left( \{a_0, \dots, a_k\}, \begin{pmatrix} p_k & p_{k-1} \\ q_k & q_{k-1} \end{pmatrix} \right)$  s.t.  $\deg u_{k-1} \geq \frac{\deg P}{2} > \deg u_k$

```

1: if $2 \deg Q < \deg P$ then
2: return $\left(\{ \}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right)$
3: end if
4: $m_1 \leftarrow \left\lfloor \frac{\deg P}{2} \right\rfloor + 1$
5: $P_1 \leftarrow \left\lfloor \frac{P}{x^{m_1}} \right\rfloor$
6: $Q_1 \leftarrow \left\lfloor \frac{Q}{x^{m_1}} \right\rfloor$
7: $(\{a_0, \dots, a_k\}, M) \leftarrow \text{Half-GCD}(P_1, Q_1)$
8: $\begin{pmatrix} u_{k-1} \\ u_k \end{pmatrix} \leftarrow M^{-1} \begin{pmatrix} P \\ Q \end{pmatrix}$
9: if $2 \deg u_k < \deg P$ then
10: return $(\{a_0, \dots, a_k\}, M)$
11: end if
```

---

---

```

12: $(a_{k+1}, u_{k+1}) \leftarrow \text{divmod}(u_{k-1}, u_k)$
13: $m_2 \leftarrow \deg P - \deg u_k$
14: $P'_1 \leftarrow \left\lfloor \frac{u_k}{x^{m_2}} \right\rfloor$
15: $Q'_1 \leftarrow \left\lfloor \frac{u_{k+1}}{x^{m_2}} \right\rfloor$
16: $(\{a'_0, \dots, a'_{k'}\}, M') \leftarrow \text{Half-GCD}(P'_1, Q'_1)$
17: return $\left(\{a_0, \dots, a_{k+1}, a'_0, \dots, a'_{k'}\}, M \begin{pmatrix} a_{k+1} & 1 \\ 1 & 0 \end{pmatrix} M' \right)$

```

---

利用 Half-GCD, 问题 3.1 可以较为高效地求解。下面的伪代码将求解  $\frac{P}{Q}$  的连分数表示  $[a_0; a_1, \dots, a_N]$  以及  $M_{N-1}$ 。

---

**Algorithm 4** GCD

---

**Input:**  $P, Q$

**Output:**  $\left( \{a_0, \dots, a_N\}, \begin{pmatrix} p_N & p_{N-1} \\ q_N & q_{N-1} \end{pmatrix} \right)$

```

1: $N \leftarrow -1$
2: $M \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
3: while $Q \neq 0$ do
4: if $\deg Q \leq \deg P < 2 \deg Q$ then
5: $(\{a'_0, \dots, a'_k\}, M') \leftarrow \text{Half-GCD}(P, Q)$
6: $N \leftarrow N + k + 1$
7: $(a_{N-k}, \dots, a_N) \leftarrow (a'_0, \dots, a'_k)$
8: $M \leftarrow MM'$
9: $\begin{pmatrix} P \\ Q \end{pmatrix} \leftarrow (M')^{-1} \begin{pmatrix} P \\ Q \end{pmatrix}$
10: else
11: $N \leftarrow N + 1$
12: $(a_N, R) \leftarrow \text{divmod}(P, Q)$
13: $M \leftarrow M \begin{pmatrix} a_N & 1 \\ 1 & 0 \end{pmatrix}$
14: $(P, Q) \leftarrow (Q, R)$
15: end if
16: end while
17: return $(\{a_0, \dots, a_N\}, M)$

```

---

设  $P$  和  $Q$  的次数都是  $O(n)$  级别, 则每次调用 Half-GCD, 多项式的次数至少减半, 因此时间复杂度仍是  $O(n \log^2 n)$ 。

若已知  $\frac{P}{Q}$  的连分数表示  $[a_0; a_1, \dots]$ , 结合式 (2.2)、式 (2.5), 可以利用多项式分治乘法在  $M(n) \log n = O(n \log^2 n)$  的时间复杂度内计算  $\{p_i\}, \{q_i\}$  或  $\{u_i\}$  中的某一项。

下面介绍该算法的几个应用。

### 3.2 多项式模求逆

**问题 3.2.** 给定一个次数为  $n$  且  $n$  次项系数为 1 的多项式  $M$  和一个次数小于  $n$  的多项式  $F$ , 保证  $M$  和  $F$  没有公因式, 求一个次数小于  $n$  的多项式  $G$  使得  $FG \equiv 1 \pmod{M}$ 。

**解法.** 求出  $\frac{M}{F}$  的渐近分数  $\left\{\frac{p_0}{q_0}, \dots, \frac{p_N}{q_N}\right\}$ , 由式 (2.3),  $Fp_{N-1} - Mq_{N-1} = c$ ,  $c$  为一个数, 取  $G = \frac{1}{c}p_{N-1}$  即可。  $\square$

### 3.3 有理分式逼近

**定理 3.2.** 对于多项式  $A, B$ ,  $\deg A < \deg B$ , 设  $(B, A)$  的欧几里得项为  $\{u_{-2}, u_{-1}, \dots, u_N\}$ ,  $\frac{B}{A}$  的渐近分数为  $\left\{\frac{p_0}{q_0}, \dots, \frac{p_N}{q_N}\right\}$ 。若多项式  $\hat{p}, \hat{u}$  满足  $\deg \hat{p} + \deg \hat{u} < \deg B$  且  $A\hat{p} \equiv \hat{u} \pmod{B}$ , 则存在  $n$  和多项式  $C$ , 使得  $\hat{u} = (-1)^{n+1}C \cdot u_n, \hat{p} = C \cdot p_n$ 。

**证明.** 若  $\hat{u} = 0$ , 则  $n = N$ 。否则, 设  $\mu = \deg \hat{u}, \nu = \deg B - \mu - 1$ , 由  $\deg u_{-1}, \dots, \deg u_{N-1}$  严格递减且  $\deg u_{N-1} = \deg \gcd(A, B) \leq \deg \hat{u} = \mu$ , 知存在  $n$  使得  $\deg u_{n-1} > \mu \geq \deg u_n$ 。由式 (3.1c), 得  $\deg p_n \leq \nu < \deg p_{n+1}$ 。

结合式 (2.6) 与条件, 知存在  $\hat{q}$  使得

$$\begin{cases} Bq_n - Ap_n = (-1)^n u_n & (3.2a) \\ B\hat{q} + A\hat{p} = \hat{u} & (3.2b) \end{cases}$$

将 (3.2a) 两边同乘  $\hat{p}$ , (3.2b) 两边同乘  $p_n$ , 得

$$\begin{cases} B\hat{p}q_n - A\hat{p}p_n = (-1)^n \hat{p}u_n & (3.3a) \\ Bp_n\hat{q} + A\hat{p}p_n = p_n\hat{u} & (3.3b) \end{cases}$$

因此  $(-1)^{n+1}\hat{p}u_n \equiv p_n\hat{u} \pmod{B}$ 。又由

$$\deg(\hat{p}u_n) = \deg \hat{p} + \deg u_n \leq \nu + \mu < \deg B$$

$$\deg(p_n\hat{u}) = \deg p_n + \deg \hat{u} \leq \nu + \mu < \deg B$$

可得  $(-1)^{n+1}\hat{p}u_n = p_n\hat{u}$ , 结合 (3.3a)、(3.3b), 进而有  $\hat{p}q_n = -p_n\hat{q}$ 。

由  $p_n, q_n$  无公因式, 知存在多项式  $C$  使得  $\hat{p} = C \cdot p_n, \hat{q} = -C \cdot q_n$ , 结合 (3.2a)、(3.2b), 得  $\hat{u} = (-1)^{n+1}C \cdot u_n$ 。  $\square$

对于多项式  $A$ , 若一个不超过  $\mu$  次的多项式  $P$  和一个不超过  $\nu$  次的多项式  $Q$  满足  $AQ \equiv P \pmod{x^{\mu+\nu+1}}$ , 且  $\deg Q$  是所有满足该条件的  $Q$  中最小的, 则称  $\frac{P}{Q}$  是  $A$  的一个  $[\mu/\nu]$  阶帕德逼近, 记作  $A_{\mu/\nu} = \frac{P}{Q}$ 。

**问题 3.3.** 给定整数  $\mu, \nu$  和一个不超过  $\mu + \nu$  次的多项式  $A$ , 求  $A$  的  $[\mu/\nu]$  阶帕德逼近。

**解法.** 设  $(x^{\mu+\nu+1}, A)$  的欧几里得项为  $\{u_{-2}, u_{-1}, \dots, u_N\}$ ,  $\frac{x^{\mu+\nu+1}}{A} = [a_0; a_1, \dots, a_N]$ 。由定理 3.2, 答案一定为  $\frac{(-1)^{n+1}u_n}{p_n}$  的形式。

由性质 3.1,  $\deg p_n + \deg u_n = \mu + \nu + 1 - \deg a_{n+1}$ , 因此对任意  $\deg p_n \leq k < \deg p_{n+1}$ ,  $A$  的  $[(\mu + \nu - k)/k]$  阶帕德逼近都是  $\frac{(-1)^{n+1}u_n}{p_n}$ 。由于  $0 = \deg p_{-1} < \dots < \deg p_N = \mu + \nu + 1 - \deg \gcd(x^{\mu+\nu+1}, A)$ , 若  $\mu < \deg \gcd(x^{\mu+\nu+1}, A)$ , 则取  $n = N$ ; 否则取  $n$  使得  $\deg p_n \leq \nu < \deg p_{n+1}$  即可。□

### 3.4 最短线性递推式

**问题 3.4.** 给定一个不超过  $n$  次的多项式  $A$ , 求多项式  $P, Q$  使得  $\deg Q$  最小且满足  $Q$  非零,  $\deg P < \deg Q$ ,  $AQ \equiv P \pmod{x^{n+1}}$ 。

**引理 3.2.** 问题 3.4 的答案一定是  $A$  的  $[\mu/\nu]$  阶帕德逼近, 其中  $\mu + \nu = n$ 。

**证明.** 设  $(x^{n+1}, A)$  的欧几里得项为  $\{u_i\}$ ,  $\frac{x^{n+1}}{A}$  的渐近分数为  $\left\{\frac{p_i}{q_i}\right\}$ 。

对任一组解  $(P, Q)$ , 取  $k$  使得  $\deg p_k \leq \deg Q < \deg p_{k+1}$ , 下证一定有  $\deg u_k < \deg p_k$ 。考虑反证, 假设  $\deg u_k \geq \deg p_k$  即  $\deg p_k + \deg p_{k+1} \leq n + 1$ , 注意到

$$\deg(Qu_k) = \deg Q + \deg u_k < \deg p_{k+1} + \deg u_k = n + 1$$

$$\deg(Pp_k) = \deg P + \deg p_k < \deg Q + \deg p_k < \deg p_{k+1} + \deg p_k \leq n + 1$$

类似定理 3.2 可得存在多项式  $C$  使得  $P = (-1)^{k+1}C \cdot u_k$ ,  $Q = C \cdot p_k$ , 则有  $\deg P = \deg u_k - \deg C \geq \deg p_k - \deg C = \deg Q$ : 矛盾!

因此  $((-1)^{k+1}u_k, p_k)$  也是问题 3.4 不劣的一组解。□

由引理 3.2, 只需找到最小的  $k$  满足  $\deg u_k < \deg p_k$ ,  $\frac{(-1)^{k+1}u_k}{p_k}$  即为问题 3.4 所求。

**问题 3.5.** 给定  $a_0, \dots, a_n$ , 求  $r_1, \dots, r_d$ , 使得  $d$  最小, 并且对任意  $m \geq d$ , 有

$$a_m = \sum_{i=1}^d r_i a_{m-i}$$

解法. 令  $A = \sum_{i=0}^n a_i x^i$ , 注意到若多项式  $B, R$  满足  $R$  的常数项为 1 且  $AR \equiv B \pmod{x^{n+1}}$ , 则由  $R$  可以得到一个合法的长为  $\max(\deg R, \deg B + 1)$  的递推式。

要限制  $R$  的常数项非零比较麻烦, 考虑令  $\tilde{A} = \sum_{i=0}^n a_i x^{n-i}$ 。若多项式  $\tilde{B}, \tilde{R}$  满足  $\tilde{R}$  的首项系数为 1,  $\deg \tilde{B} < \deg \tilde{R}$  且  $\tilde{A}\tilde{R} \equiv \tilde{B} \pmod{x^{n+1}}$ , 则取  $r_i = -[x^{\deg \tilde{R}-i}] \tilde{R}$  可以得到一个长为  $\deg \tilde{R}$  的递推式。现在没有了常数项非零的限制, 问题便转化为求解次数最小的  $\tilde{R}$ , 即问题 3.4。□

### 3.5 多项式的结式

对于一个  $n$  次多项式

$$A = \sum_{i=0}^n a_i x^i = a_n \prod_{i=1}^n (x - \alpha_i)$$

和一个  $m$  次多项式

$$B = \sum_{i=0}^m b_i x^i = b_m \prod_{i=1}^m (x - \beta_i)$$

定义  $(A, B)$  的结式为

$$\text{Res}(A, B) = a_n^m b_m^n \prod_{i=1}^n \prod_{j=1}^m (\alpha_i - \beta_j)$$

易得

性质 3.2.

$$\text{Res}(A, B) = a_n^m \prod_{i=1}^n B(\alpha_i) \quad (3.4)$$

性质 3.3.

$$\text{Res}(A, B) = (-1)^{nm} \text{Res}(B, A) \quad (3.5)$$

性质 3.4. 设  $R = B \bmod A$ ,  $\deg R = r$ , 则

$$\text{Res}(A, B) = a_n^{m-r} \text{Res}(A, R) \quad (3.6)$$

证明. 设  $B = AQ + R$ , 则

$$\begin{aligned} \text{Res}(A, B) &= a_n^m \prod_{i=1}^n B(\alpha_i) \\ &= a_n^m \prod_{i=1}^n (A(\alpha_i)Q(\alpha_i) + R(\alpha_i)) \\ &= a_n^m \prod_{i=1}^n R(\alpha_i) \\ &= a_n^{m-r} \text{Res}(A, R) \end{aligned}$$

□

结合性质 3.3 与性质 3.4, 可以得到

推论 3.1. 设  $R = A \bmod B$ ,  $\deg R = r$ , 则

$$\text{Res}(A, B) = (-1)^{nm} b_m^{n-r} \text{Res}(B, R) \quad (3.7)$$

问题 3.6. 给定多项式  $A, B$ , 求  $\text{Res}(A, B)$ 。

解法. 设  $(A, B)$  的欧几里得项为  $\{u_{-2}, u_{-1}, \dots, u_N\}$ , 若  $\deg u_{N-1} > 0$ , 则  $\text{Res}(A, B) = 0$ 。

否则, 令  $d_i = \begin{cases} \deg u_i & (i < N) \\ 0 & (i = N) \end{cases}$ ,  $\hat{u}_i$  为  $u_i$  的首次项系数, 则由式 (3.7)

$$\text{Res}(A, B) = \prod_{i=0}^N (-1)^{d_{i-2}d_{i-1}} \hat{u}_{i-1}^{d_{i-2}-d_i}$$

在执行 Half-GCD 的过程中额外维护出每个  $u_i$  的首次项系数即可。□

## 4 总结

本文介绍了连分数与欧几里得算法的许多性质与应用。在一些关于有理数逼近或格点相关的最值问题中, 利用连分数相关性质, 往往能简洁而直观地解决问题, 而其他方法显得略有繁琐。

本文还介绍了 Half-GCD 算法, 利用 Half-GCD, 能够高效地执行多项式环上的欧几里得算法。随后本文列举了该算法的若干应用。

本文也有许多未提及的内容, 如无理数的连分数表示、连分数与 Stern-Brocot 树和 Farey 序列之间的联系、多项式欧几里得算法更深入的性质与应用等。希望本文能够激发读者的思考, 起到抛砖引玉的作用。

## 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢教练金靖老师的关心与指导。

感谢父母对我的培育和教诲。

感谢所有帮助我的老师、同学。

## 参考文献

[1] Pick, G. Geometrisches zur Zahlentheorie. *Sitzber. Lotos (Prague)*, 19:311–319, 1899.



- [2] Robert J. McEliece and James B. Shearer. A Property of Euclid's Algorithm and an Application to Padé Approximation. *SIAM Journal on Applied Mathematics*, 34(4):611–615, 1978.
- [3] Algorithms for Competitive Programming. Continued fractions. <https://cp-algorithms.com/algebra/continued-fractions.html>.
- [4] Codeforces. Half-gcd algorithm. <https://codeforces.com/blog/entry/101850>.





# 浅谈复杂度及其在解决问题方面的应用

南京外国语学校 杨敏行

## 摘要

计算复杂度是计算机科学中一个重要的概念。本文引入了更加泛用的多元复杂度概念，介绍了数种较为常见的限制对应的复杂度，列举了各种数据范围下常见的复杂度范围，并引入了精细复杂性理论的概念。

## 1 引言

本文介绍了复杂度及其在信息学竞赛中的若干较为重要的应用。

第二节介绍了复杂度的概念，并扩展一元复杂度得到了更泛用的多元复杂度概念。

第三节列举了各种数据规模下的常见复杂度。

第四节介绍了数种有可能成为解题突破点的特殊限制。

第五节引入了精细复杂性理论，并粗浅介绍数个经典问题现有的复杂度下界。

## 2 一元复杂度

### 2.1 引入复杂度的原因

设计算法是为了解决问题：不仅是为了得到正确的答案，更希望可以更快速地得到正确的答案。因此，我们有寻找效率更优异的算法的需要。

在现代计算机领域，通常认为四则运算、取模、左右移、与或非、异或、输入输出、赋值等操作，如果所操作的变量不过大（何种范围被认为是不过大的将在下文讨论），则可以认为只需一次操作即可完成。本文中理想地认为所有上述运算的效率相等，所以一份算法的效率就可以被看作是其执行的操作数目。

一般来说，通过阅读代码，可以计算或估计得到操作数目。但是，随着输入规模的变化，操作数目也会相应的变化，原本操作数目较少的算法可能反而操作数目变多了。计算理论方面往往认为，输入规模更大时的算法效率是更重要的，因此，设计一个通过算法在  $n$  充分大时的表现来衡量其效率高低的方式是有必要的。

由此,可以定义一个函数  $f(n)$ ,表示算法的输入是  $n$  时,所执行的操作数目。于是,比较算法的效率的问题便被转化为比较函数规模的数学问题了。

## 2.2 一元复杂度

**定义 2.1** (一元复杂度). 对于两个函数  $f(n), g(n)$ , 如果存在实数  $M > 0$  以及一个  $N > 0$ , 使得  $\forall n \geq N$ , 都有  $|f(n)| \leq M|g(n)|$ , 此时称  $g$  是  $f$  的渐进上界, 使用 *big-O-notation* 记号可记作  $f = O(g)$ 。此时, 认为函数  $f$  有着  $g$  或者  $O(g)$  的复杂度。

注意到  $O(g)$  其实指代了所有满足上述条件的  $f$  构成的集合, 因此正确的写法其实应该是  $f \in O(g)$ , 但因为历史习惯原因, 通常采取  $f = O(g)$  的写法, 但是这里的  $=$  符号仍然表示  $\in$ , 因此没有反向的  $O(g) = f$  成立。

**定理 2.1.** 易验证一元复杂度有着如下定理:

1. 传递性: 若  $f = O(g), g = O(h)$ , 则  $f = O(h)$ 。
2. 线性性: 若  $f = O(h), g = O(h)$ , 那么  $\lambda f + \mu g = O(h)$ , 其中  $\lambda, \mu \in \mathbb{R}$ 。
3. 乘法结合性: 若  $f = O(F), g = O(G)$ , 那么  $f \times g = O(F \times G)$ 。
4. 常底数对数的互换: 由换底公式, 底数为常数的  $O(\log_a n)$  可以互换底数, 也即  $\log_b n = \frac{\log_a n}{\log_a b}$ , 而  $\log_a b$  是在渐进复杂度中无影响的常数。因此, 在研究底数为常数的对数时, 我们常常忽略底数, 统一记作  $O(\log n)$ 。

**定义 2.2.** 粗略定义一些常见的复杂度级别:

- 对数多项式复杂度 ( $O(\text{polylog}(f))$ ): 复杂度是关于  $\log f$  的多项式。
- 多项式复杂度 ( $O(\text{poly}(f))$ ): 复杂度是关于  $f$  的多项式。
- 指数复杂度: 复杂度是  $a^f$ , 其中  $a$  是一个大于 1 的实数。

易证, 在  $n$  充分大时, 对数多项式复杂度总是优于多项式复杂度, 而多项式又总是优于指数。同理, 三类内部又都有大小比较关系, 即低次多项式优于高次多项式, 小底数指数复杂度又优于大底数的复杂度。除此之外, 还有一些无法归入上述分类的复杂度, 例如  $\log \log n$  或者  $n!$ , 但是通过基本的极限运算, 比较复杂度是容易的。

如果  $f = O(g)$ , 那么说明在  $n$  充分大时,  $f$  的效率不劣于  $g$  的效率。如果同时有  $g = O(f)$ , 则表明  $f, g$  的效率大体是相同的, 可记作  $f = \Theta(g)$ , 同时亦有  $g = \Theta(f)$ , 此时可称  $f, g$  同阶。此时决定两个算法效率的就是  $f, g$  前面的常数, 但是只要常数区别不大一般认为二者效率相近。否则, 如果没有  $g = O(f)$ , 则表明  $f$  的效率严格优于  $g$ , 尽管这个优势可能要在  $n$  充分大时才能展现。

除此之外, 还有一种复杂度分析中较为常见的概念, 即 *soft-O-notation*。

定义 2.3 (软复杂度<sup>1</sup>). 如果  $f = O(g(n)\text{polylog}(g(n)))$ , 则可记  $f = \tilde{O}(g(n))$ 。[1] 中提到了软复杂度的另一种定义 ( $O(g(n)\text{polylog}(n))$ ), 但是为了有  $O(n^a \beta^n)$  可以被写作  $\tilde{O}(\beta^n)$ , 所以采取第一种写法。

软复杂度将在后文得到应用。

目前已经创造了衡量函数效率的基础工具; 但是其还太过粗糙, 难以进行一些精细的分析: 可以在下方的例子中看到这一点。

例 2.1. 给定  $n$  个整数  $\{a_1, a_2, \dots, a_n\}$ , 对其求和。

一个简单的想法是, 用一个初始为零的变量维护结果, 依次将这个变量同每个  $a_1, a_2, \dots, a_n$  相加, 即为结果。一共进行了  $n$  次加法, 因此这个算法的复杂度是  $O(n)$  的。

但是, 注意到前面提到的, 四则运算等操作在变量“不过大”时只需一次操作; 而如果值域超过了  $2^{32}, 2^{64}, 2^{128}$  等电脑常见的类型大小, 计算机就无法仅使用一次运算来执行上述操作。很明显, 此时上述算法的复杂度并非只与  $n$  相关: 这些整数的值域也是一个无法忽略的维度。此时,  $f$  就不再是一元函数: 其是二元、乃至多元函数。

- 另一种较常见定义是, 令  $n$  是“输入规模”: 输入一个值域为  $A$  的元素需要  $\log_2 A$  个 bit, 进而在此种意义下复杂度关于输入规模是严格  $O(n)$  的。但是, 当问题的输入更加繁杂时, 这样分析出的结果常常是混乱不堪的。因此, 将更多变量纳入考虑的范围的确是有必要的。

## 2.3 多元复杂度

定义 2.4 (邻域). 对于  $a \in \mathbb{R} \cup \{\pm\infty\}$ ,  $a$  的一个邻域是指:

- $(a - \delta, a + \delta)$ , 如果  $a \in \mathbb{R}$ ; 其中  $\delta$  是任何正实数。
- $(N, +\infty)$ , 如果  $a = +\infty$ ; 其中  $N$  是任何实数。
- $(-\infty, N)$ , 其中  $a = -\infty$ ; 其中  $N$  是任何实数。

定义 2.5 (多元复杂度). 对于两个多元函数  $f(x_1, x_2, \dots, x_n), g(x_1, x_2, \dots, x_n)$ , 若对于确定的一组  $y_1, y_2, \dots, y_n \in \mathbb{R} \cup \{\pm\infty\}$ , 有:

- 存在一个实数  $M > 0$ , 使得存在  $y_1, \dots, y_n$  的邻域, 对于该邻域中的任何  $z_1, \dots, z_n$ , 都有  $|f(z_1, \dots, z_n)| \leq M|g(z_1, \dots, z_n)|$ 。

<sup>1</sup>笔者自译

此时, 称  $g$  是  $f$  在  $y_1, y_2, \dots, y_n$  处的渐进上界, 记作  $f = O(g)$  或者  $f \in O(g)$ 。

同时, 一元复杂度的对数多项式复杂度、多项式复杂度和软复杂度的概念也可以简单推广到多元复杂度。

**定理 2.2.** 如果所有函数的自变量  $\{x_1, \dots, x_n\}$  都相同, 且自变量都趋于同一组  $\{y_1, \dots, y_n\}$ , 那么易验证多元复杂度仍满足一元复杂度的定理 2.1。

可以发现, 一元复杂度是多元复杂度令变量为  $n$ , 且  $n \rightarrow +\infty$  时的特殊形式。

使用多元复杂度的概念, 我们可以重新描述各种基础运算的复杂度:

**定理 2.3.** 如果变量的值域是  $V$ , 那么加减、与或非、异或、输入输出、赋值等操作的复杂度是  $O(\log_{2^w} V)$ , 而乘除、取模等操作, 使用 FFT 算法 [2] 可以在  $O(\log_{2^w} V \log_2 \log_{2^w} V)$  内解决。

其中,  $w$  是计算机的字长, 也即计算机可以使用一次操作处理值域为  $2^w$  的运算。常见的  $w$  有 32, 64, 128 等, 随着计算机技术的发展, 将来可能会出现更大的  $w$ 。将  $w$  也看作变量, 按照传统, 忽略常数底数并把变量底数使用换底公式摘出来, 则有加减类运算的复杂度  $O(\frac{\log V}{w})$ , 乘除类运算的复杂度  $O(\frac{\log V}{w} \log \frac{\log V}{w})$ 。

通过多元复杂度的新工具, 我们可以得到一个考虑了所有影响因素的描述例 2.1 中所述算法的复杂度, 也即令  $A = \sum_{i=1}^n a_i$ , 那么复杂度是  $O(\frac{n \log A}{w})$ 。

在上述例子中,  $n, A, w$  都是趋于无穷的。但是, 现实生活中, 不仅有趋于无穷的变量, 还有变量有趋于其它值的必要, 在接下来的例子中可以得见。这也是需要通过邻域的概念来拓展复杂度应用范围的原因。

**例 2.2 (二分法).** 设  $\phi(x)$  是一个  $[0, n]$  上严格递增的连续函数。若  $\phi(0) < 0 < \phi(n)$ , 现在要以不超过  $\epsilon$  的误差找到  $\phi(x)$  的零点。

令零点在  $[l, r]$  范围内, 令  $mid = \frac{l+r}{2}$ , 检验  $\phi(mid)$ , 如果  $\phi(mid) = 0$ , 那么因为函数严格增所以零点即为  $mid$ , 否则若  $\phi(mid) > 0$  则零点在  $[l, mid]$  内, 否则即  $\phi(mid) < 0$  则零点在  $[mid, r]$  内。重复执行上述流程, 直到  $r-l \leq \epsilon$ , 则此时误差已不超过  $\epsilon$ 。

易验证这个算法一共需要调用  $O(\log(\frac{n}{\epsilon}))$  次  $\phi$ , 其中  $n \rightarrow +\infty, \epsilon \rightarrow 0$ 。

注意该算法的复杂度并非上式: 因为单次调用  $\phi(x)$  的复杂度并非  $O(1)$ 。

## 2.4 简化复杂度的方法

复杂度不仅仅可以描述在变量趋于目标值时算法的操作数目, 也可以直接代入变量的范围, 得到一个对操作数目的估计。这其实是复杂度最常见的应用, 因为最常见的变量目标值是无穷, 而在实际应用的时候我们往往并不关心  $n$  非常大时算法的表现。





比如说,在一个问题中,如果有  $n \leq 10^5$ ,那么基本可以立刻放弃所有复杂度为  $O(n^2)$  的做法,并且认为所有复杂度为  $O(n \log n)$  的算法基本上都是可以通过的。复杂度就像一个标签,它帮助我们快速过滤掉一批算法,并判定另一些算法有成为正解的可能。

标签之间也有好坏之分:例如  $O(n)$  就明显比  $O(\frac{n \log A}{w})$  要更加简洁。而假如在给定的数据范围下,  $\frac{\log A}{w}$  一项的值不算很大,那么把它看成一个常数也未尝不可。比如说,在例 2.1 中,一个常见的数据范围是  $n \leq 10^5, A \leq 10^9, w = 32$ ,那么就有  $O(\frac{n \log A}{w}) \in O(\frac{n \log 10^9}{32}) = O(n)$ 。因此,下文中默认值域不过大,进而忽略  $\frac{\log A}{w}$  一项。

从另一个角度说,也有  $O(\frac{n \log A}{w}) \in O(\frac{10^5 \log 10^9}{32}) = O(1)$ 。但是,此时这个  $O(1)$  的标签就不是一个合适的标签,因为向其中代入  $n, w, A$  的取值后得到的结果是 1,与真实的操作数 ( $n$  次,也即  $10^5$  次操作) 大相径庭。

这说明,将复杂度式子中一些量看成常数有助于简化问题,但是如果不分青红皂白就将一切变量都看作常数,则并非一个好的近似。

同样的简化复杂度形式的操作还有,将一些变量用其它变量表示(比如说,简单图的边数  $m$  数目不超过  $n^2$ ,于是涉及到  $m$  的复杂度  $f(m)$  可以被写作  $f(n^2)$ );将数据范围相同的变量合并(比如说一道题元素个数  $n$  和询问次数  $q$  的范围都是  $10^5$ ,那么它们就可以统一以  $n$  表示)。这些操作以减少复杂度的泛用性和严谨性为代价,大大降低了复杂度描述的繁琐度。在下一节里,我们将看到这样简化在解题方面的意义。

### 3 从数据范围到复杂度

#### 3.1 前提

信息学竞赛遇到的问题与实际生活中遇到的问题相比,区别之一是竞赛的问题总是有解的。这意味着存在一个算法可以在题目的数据范围下通过问题,于是数据范围也可以成为辅助我们寻找答案的工具之一。

从数据范围到算法,我们需要复杂度作为桥梁。所以我们才需要适当地对复杂度加以简化,否则枚举各种复杂得毫无意义的复杂度是舍本逐末的。

出于同样的道理,我们仅考虑最基础的场合,即认为输入仅有一维  $n$  是重要的。至于更多变量的场合,如果不存在乘积量,可以尝试对于每个变量分开分析,因为复杂度具有线性性。否则,可以尝试把乘积整体当作变量分析。如果上述分析无法成立,则这种过于复杂的场合或许并不是根据数据范围反推复杂度的合适情景。另外,下文仅提供思路的参考,具体问题还是应该具体分析。





## 3.2 常见数据范围对应的复杂度

范围 ( $n \leq 1 \sim 12$ ). 可能对应着一些比较高的复杂度, 例如  $4^n, 5^n$  或者  $Bell(n), n!$  等, 其中  $Bell(n)$  指代着 Bell 数 [3], 即等价类计数。当然, 这些复杂度都可以是软复杂度。

范围 ( $n \leq 15 \sim 17$ ). 可能对应着各种大到小  $poly(n)$  系数的  $\tilde{O}(3^n)$ , 或者具有较大  $poly(n)$  系数的  $\tilde{O}(2^n)$ 。

范围 ( $n \leq 20$ ).  $2^n$ , 带有一般来说不超过二次的  $poly(n)$  系数。需要注意的是, 也有一种偶见的  $\binom{n}{n/2}$  的复杂度, 比如说如下例题:

**例 3.1** (神隐<sup>2</sup>). 交互库有一棵  $n \leq 2^{17}$  个点  $n-1$  条边的树, 第  $i$  条边是  $(u_i, v_i)$ 。初始不知道树的任何信息, 你需要在不超过 20 次询问中, 确定树上的所有边 (边的顺序任意)。你可以使用的询问是, 给定若干  $i$ , 交互库返回在原树中保留给定的边、删去其它边后, 各个极大连通块包含的点集。询问之间独立。

对于一对距离恰为 2 的点对  $x, y$ , 设其经过一个点  $z$ 、两条边  $i, j$  连通。此时, 如果  $i, j$  总是或同时保留, 或同时被删去, 则  $x, y, z$  要么在同一连通块中、要么散在三个连通块里, 它们之间的连边方式无法被确定。因此, 如果我们构建一个算法, 其中对于所有  $i, j$ , 都存在一次询问使得  $i$  出现  $j$  不出现, 并保证所有边在所有询问中出现次数都相同 (设为  $K$ ), 那么对于直接连通的点对, 它们应该恰在同一个连通块中  $K$  次, 非直接连通的点对, 在同一个连通块中的次数是小于  $K$  的。

一个朴素的想法是, 询问  $2 \lceil \log_2 n \rceil$  次, 第  $2i+1$  次询问所有二进制下第  $i$  位为 1 的边, 第  $2i+2$  次询问所有二进制下第  $i$  位为 0 的边。这个算法满足上述要求, 但是操作数目有些过大。

设总询问数目是  $T$ , 则算法的本质是为所有边分配两两不同的  $T$  位二进制标号, 其中恰有  $K$  位为 1, 然后第  $i$  次询问所有标号的第  $i-1$  位为 1 的元素。所有可行的标号数目是  $\binom{T}{K}$ , 则只要标号数目不小于边数就可以为所有边分配标号。二项式系数的性质使得  $K = \lfloor T/2 \rfloor$  时的二项式系数最大, 而取  $T = 20, K = 10$  时即有  $\binom{T}{K} \geq n-1$ 。

除此之外, 还要实现寻找那些在  $K$  次询问中连通的点对的算法, 这部分因为并不简单且与本文关联不大而略去。

这类方法常见于恰有  $K$  位为 1 的  $T$  位二进制数数目的最大值, 在一些状态压缩 DP 的问题, 例如一双木棋<sup>3</sup> 中也有出现, 可以减少状态数目。

同时, 这道例题也给我们启示: 数据范围不一定总是标志着复杂度, 也可能启示了某些量的级别。对这些量级的敏感度有时可以成为我们解决问题的关键。

<sup>2</sup>UNR#6 D2T2; <https://uoj.ac/problem/751>

<sup>3</sup>九省联考 2018, <https://www.luogu.com.cn/problem/P4363>



范围 ( $n \leq 26 \sim 28$ ). 这是  $O(2^n)$  复杂度的上限。需要注意的是, 有时复杂度可以是  $\sum_{i=0}^n 2^i = O(2^n)$ 。

这里记录一道笔者认为很有趣的该数据范围下的例题:

**例 3.2** <sup>(4)</sup>. 给定  $m$  个  $n$  位二进制数, 求有多少个  $n$  位二进制数可以表示成其中若干个数的或。  $n \leq 28, m \leq \min(2^n, 5 \times 10^5)$ 。

将一个二进制数  $a$  看成集合幂级数 (见 [4])  $1 + x^a$ , 可以使用一种对一次集合幂级数的特殊 FWT 卷积算法做到  $O(n2^n)$ 。但是这无法通过。

思考上述算法的劣势是什么: 其可以对于每个二进制数求出有多少种表示为若干个数的或的方法, 而这是强于原问题的。

一般地, 若仅关心每个数的存在性而不关心其计数, 可以尝试 *bitset* 或压位优化。首先, 要将问题转化为一个较为简单的形式: 一个数  $b$  能被表出, 显然当且仅当所有是其子集的  $a$  或运算的结果是其本身。因此定义  $f(a)$  为,  $a$  若在给定的  $m$  个数中出现则为  $a$ , 否则为 0。然后对  $f$  作高维前缀或 (同样见 [4]) 变换, 即枚举关键位  $i$ , 然后枚举  $a$ , 如果  $a$  第  $i$  位为零, 则令  $b := a \vee 2^i$ , 然后令  $f(b) := f(b) \vee f(a)$ 。最后只要对  $f(b) = b$  的个数计数即可。

尝试使用该流程判定对于第  $i$  位为 1 的  $b$ , 是否有最终的  $f(b)$  在第  $i$  位上为 1。假如某个  $b$  关于所有的  $i$  验证都通过, 那么其就满足  $f(b) = b$ 。

在  $i$  固定时,  $f(a)$  可以从一个  $n$  位二进制数变成一个 0/1 量  $g(a)$ , 表示  $a$  是否在给定的  $m$  个数中出现。然后需要判定  $b$  的子集内元素, 是否存在至少一个  $a$  满足  $g(a) = 1$  且  $a$  的第  $i$  位为 1, 这等价于对  $g$  高维前缀或, 但是枚举转移的关键位时要跳过第  $i$  位不处理 (读者可以自行验证正确性)。这可以压位优化: 用一个 *unsigned long long* 存储低 6 位的值, 然后在作前缀或时分当前处理的位是高 22 位还是低 6 位处理即可。复杂度  $O\left(\frac{n^2 2^n}{w}\right)$ , 其中  $w$  是压位位数, 本题为 64。

虽然其比一开始的  $O(n2^n)$  要优秀, 但是还是无法通过。但是, 注意到对  $i$  的转移涉及到全体  $j \neq i$ , 这适用于一个经典的缺一分治的技巧: 分治时维护  $g$  数组, 递归时将当前区间分成两半: 右侧区间中的元素对左侧区间中元素会产生贡献, 所以枚举右侧区间中元素, 用其对  $g$  转移, 然后递归进入左侧区间, 返回时撤销右侧区间元素的影响; 然后递归进入右侧区间时, 就用左侧区间中元素产生影响, 返回时再撤销。可以发现, 这样处理确实会对于每个  $i$  让所有  $j \neq i$  被贡献恰一次。由此, 可以将复杂度优化至  $O\left(\frac{n \log n 2^n}{w}\right)$ 。

范围 ( $n \leq 30 \sim 60$ )。这些数据范围往往不会有多项式复杂度。例如 *DP on DP*、划分数 *DP*、剪枝后的搜索、*meet in middle* 等算法是此区间内较主流的计算。

同时, 对于所有的非多项式复杂度算法, 永远都存在一种搜索所有本质不同的状态并通过压缩、分类等操作将状态数削减至可以承受的范围的方法。分析此种方法的复杂度往往

<sup>4</sup>非公开题

是毫无意义且与实际大相径庭的, 此时最佳策略莫过于亲自写一份暴力得到状态数: 可能会发现真实的状态数其实出乎意料得小。大部分 *DP on DP* 题都能在这种策略下找到思路。

范围 ( $n \leq 50 \sim 80$ ). 可能存在一些具有高次多项式复杂度 (例如  $\tilde{O}(n^6)$  或者  $\tilde{O}(n^7)$ ) 的 *DP* 在此范围内。需要注意的是, 高次多项式复杂度往往可能拥有一个较小的常数 (因为循环嵌套时, 内层循环的起讫值依赖于外层循环, 进而会产生一个小于 1 的常数), 所以此时在理论的大  $O$  复杂度内代入  $n$  得到的结果和实际运行的效率可能会有较大的分野, 因此有时不能因为理论复杂度大就贸然放弃一个算法的可能性。

范围 ( $n \leq 100$ ). 随着计算机性能的提高, 近年来  $n \leq 100$  的数据范围往往已经对应着  $O(n^4)$  的复杂度了。

范围 ( $n \leq 200 \sim 500$ ). 可能对应着带有不同级别  $\text{polylog}(n)$  的  $\tilde{O}(n^3)$ 、 $O(n^{3+\epsilon})$  (其中  $\epsilon \in (0, 1)$ )、直到严格的  $O(n^3)$ 。需要注意的是,  $n \leq 150 \sim 200$  可能有一种特殊的技巧, 将在下文提到。

范围 ( $n \leq 1000 \sim 10000$ ). 带有不同级别的  $\text{polylog}(n)$  的  $\tilde{O}(n^2)$ 、 $O(n^{2+\epsilon})$ 、直到严格  $O(n^2)$ 。除此之外, 在  $n \leq 1000$  左右, 有一些复杂度为  $\frac{n^3}{w}$  的使用 *bitset* 的做法, 或者复杂度为  $\frac{n^3}{\log n}$  的做法。

**例 3.3** (Density of subarrays<sup>5</sup>). 给定一个值域为  $[1, c]$  中整数的序列  $a$ 。定义一个序列的密度为最大的  $p$ , 满足所有长度为  $p$ 、值域为  $[1, c]$  中整数的序列 (共  $c^p$  个) 都是其子序列。对于  $i \in [0, n]$ , 求出  $a$  有多少个子序列, 满足其密度为  $i$ 。  $c \leq n \leq 1000$ 。

简单分析可得, 一个序列的密度可以表示为: 不断删去其最短的包含  $[1, c]$  中所有数至少一次的前缀直至无法再删, 删去的前缀数目。

令  $f_{i,j}$  表示后缀  $[i, n]$  的所有子序列 (为避免重复计算, 此处强制  $a_i$  必须包含于子序列中) 中, 密度恰为  $j$  的方案数。转移考虑枚举删去的前缀, 令  $g(i, k)$  为区间  $[i, k]$  作为删去的最短前缀的方案数: 这需要保证  $[i, k]$  中所有元素出现至少一次, 且  $a_k$  仅出现一次。令  $b_t$  为区间中元素  $t$  出现的次数, 则  $g(i, k) = 2^{b_{a_i}-1} \prod_{t \neq a_i, t \neq a_k} (2^{b_t} - 1)$ , 可以通过用桶维护  $b$  来在  $O(n^2)$  的复杂度求出  $g$ 。这样之后, 就可以以  $f_{i,j} = \sum_{k=i}^n f_{k+1,j-1} \times g(i, k)$  来进行转移, 初态为  $f_{n+1,0} = 1$ 。

这个算法的复杂度可以做到  $O(\frac{n^3}{c})$ , 因为  $j$  这一维显然只需枚举到  $\frac{n}{c}$ 。另外, 额外存在一种 *DP*, 令  $h_{i,j,k}$  表示  $[1, i]$  的前缀的所有子序列, 密度恰为  $j$ , 且最后剩下的无法被删去的后缀中包含的  $[1, c]$  中元素的状态压缩后是  $k$  的方案数, 转移在  $k$  中所有元素全都出现时令  $j$  增加并清空  $k$  即可。此时的复杂度为  $O(\frac{n^2 2^c}{c})$ 。在  $c \leq \log_2 n \leq 10$  时使用第二种算法, 否则使用第一种算法, 可以达到平衡态。复杂度上限其实是  $O(\frac{n^3}{\log n})$ 。

<sup>5</sup>Codeforces Round 559 (Div. 1); <https://codeforces.com/problemset/problem/1158/F>

这种设计两种不同的算法来分别处理某个量较大或较小场合的方法，也是一种常见的思想。或许更为人们熟知的是两个算法在根号处取得平衡的场合（即根号分治），本题则是另一种在对数处取得平衡的场合（即对数分治）。

范围 ( $n \leq 5 \times 10^4$ )。这是一个几乎所有  $\tilde{O}(n^{1+\epsilon})$  都可以通过的复杂度级别；虽然有一些例如  $O(n\sqrt{n}\log^2 n)$  的复杂度或许不太可行。在  $5 \times 10^4 \sim 5 \times 10^5$  的范围内， $O(\log n)$  和  $O(n^{0.25})$  的表现通常是相近的，虽然前者具有较优的理论复杂度，但是后者（通常在  $O(n\log^2 n)$  和  $O(n\sqrt{n})$  作比较时）往往有着较小的常数，综合而言，二者效果一般大致相同。同时，像  $O(n\sqrt{n}\log^2 n)$  或  $O(n\log^4 n)$  之类的复杂度，其对应的算法即使效率或许可以承受，代码的复杂程度也是不可接受的。

另外，存在一些  $O(\frac{n^{t=1/2/3}}{\log n})$  的算法，比如说 *The Method of Four Russians*[5]（中文文献可见 [6]）。 $t=2$  时的该技巧有可能在这种数据范围下应用。

范围 ( $n \leq 10^5$ )。例如  $n\sqrt{n}\log n, n\log^3 n$  这样的复杂度有可能通过这一级别，前提常数必须较小。另外，此数据范围下一个不可忽略的算法是  $\frac{n^2}{w}$  的 *bitset* 算法。同时，这往往是 *KDT* 能处理的数据范围的极限：*KDT* 具有过大的常数。

范围 ( $n \leq 2 \times 10^5$ )。  $n\sqrt{n}\log n$  有可能通过本数据范围。常数中等的  $n\sqrt{n}$  亦可。某些常数很大的  $n\log^2 n$  可能只能通过这种范围。

范围 ( $n \leq 5 \times 10^5$ )。常数很小的  $n\sqrt{n}$  有可能可以通过。常数适中的  $n\log^2 n$ ，以及 *LCT* 这种常数较大的  $n\log n$  也会有这种数据范围。

范围 ( $n \leq 10^6$ )。常数较小的  $n\log^2 n$ （例如二分、树状数组、排序、分治的  $\log n$ ），常数不过大的  $n\log n$ 。

范围 ( $n \leq 2 \times 10^6 \sim 5 \times 10^6$ )。常数中等至很小的  $n\log n$ 。也可能是某些亚  $n\log n$  做法，例如  $n\sqrt{\log n}$ ，或者  $O(n\alpha(n))$  的并查集。

范围 ( $n \leq 10^7 \sim 5 \times 10^7$ )。往往意味着  $O(n)$  的复杂度，其中有可能同时包含着对空间的限制。

范围 ( $n \leq 10^8 \sim 10^9$ )。往往任何低于  $O(n)$  的复杂度都可以通过，但一般  $\text{polylog}(n)$  不会只出这么小的复杂度。某些分块打表的做法也可能通过这种数据范围。筛法是这种数据范围的常客：其中  $O(n^{2/3})$  的杜教筛往往在这种范围下出现。

范围 ( $n \leq 10^{10} \sim 10^{11}$ )。一般是 *min25* 筛的范围，其它算法出这个级别的比较少见。

范围 ( $n \leq 10^{12} \sim 10^{14}$ )。常见筛法中能通过这种范围的或许只有  $O(\sqrt{n})$  的 *powerful number* 筛。其它  $\tilde{O}(\sqrt{n})$  的算法也可能出这个数据范围。

范围 (更大的  $n$ )。此时往往不会有  $n^\alpha$  的复杂度（但或许可能会出现  $n^{1/3}$  之类），更常见的还是  $\text{polylog}(n)$ ，例如数位 *DP*。这种数据范围下，一种可行的分析方式是取  $n' = \log n$  然后对  $n'$  展开如上文的分析。





总结：上述数据范围对应的复杂度仅供参考。同时，经常也能见到一些题目，如果复杂度并非题目考察的关键， $n \log n$  的复杂度只出  $n \leq 10^5$  的数据范围也是有可能的：也即，复杂度可以向下兼容数据范围，但不能向上。同时，一些很复杂的复杂度可能是光靠猜想不可能得到的，不能想当然地认为复杂度只有上述列的几种。笔者希望达到的效果是，看到  $10^6$  不去考虑根号算法，看到  $10^5$  记得试一试 bitset 优化，看到 40 想想 meet in middle，而不是遇到问题就来查表。

## 4 一些特征条件对应的复杂度

有一些很有特征的条件，常常对应着一些比较优秀的复杂度。

### 4.1 很小的 $K$

一些问题要求统计  $n$  个元素中，所有大小不超过  $K$  且额外满足某些特殊性质的元素集合，对应的某种函数的最值。如果这个  $K$  很小，一种常见的方法是对元素随机染色，计算不同颜色间的上述集合的最值。答案集合期望在不过多次随机内被正确染色，进而被求出。

**例 4.1 (Storm<sup>6</sup>)**. 给定一张  $n$  点  $m$  边的无向图，点带非负点权  $v_i$ ，边带非负边权  $e_i$ ，选择不超过  $K$  条边，最大化  $\sum_{i \in I(S)} v_i - \sum_{i \in S} e_i$ ，其中  $S$  是选择的边集， $I(S)$  是与  $S$  中所有边端点构成的集合。数据范围： $2^K(n+m) \leq 10^6, 0 \leq v_i, e_i \leq 10^8$ 。

经过分析后，可以得到存在一组最优解  $S$ ，其中选中的边构成若干无公共点的“菊花图”。此时对原图随机黑白染色，那么有至少  $\frac{1}{2^{K-1}}$  的概率，选中的边被成功黑白染色，然后在二分图上处理这个问题可以使用网络流做法，与本文关系不大就此略去。

这种随机染色方法更多的应用可见 [7]。

### 4.2 $n \times m \leq \text{定值}$

这是一类常见限制，令  $n \times m \leq C$ ，则  $\min(n, m) \leq \sqrt{C}$ 。于是，算法可能包含一些与  $\sqrt{C}$  有关的复杂度，例如  $\tilde{O}(2^{\sqrt{C}})$  或者  $O(C\sqrt{C})$  之类。

此时，解法往往与  $n, m$  二维构成的网格图有关：设计算法分别从  $n, m$  二维进行处理。有时  $n, m$  二维是对称的因此只需设计一个算法，但更多的时候是不对称的。

$n, m$  二维有时不是显式的：可能需要一些推理才能得到网格图的结构，例如如下的例题：

<sup>6</sup>2020-2021 集训队作业，<https://loj.ac/p/3400>

例 4.2 (Dance<sup>7</sup>). 数轴有  $n$  个点, 坐标分别为  $x_1, \dots, x_n$ . 现在对每个点, 将其坐标增加  $d$  或减少  $d$  (不能不操作), 然后使用总代价最小的线段覆盖坐标变动后的所有点。一段线段  $[l, r]$  的代价为  $a + b(r - l)$ .  $n \leq 100, 1 \leq x_i \leq 100, d \leq 50, 1 \leq a, b \leq 10^6$ .

问题等价于使用若干线段, 使得  $x_i \pm d$  至少有一个被覆盖。将所有点事先平移  $-d$  后, 这等价于  $x_i$  和  $x_i + 2d$  至少一个被覆盖。此时处理后, 值域是  $x_i + 2d \leq 200$  的级别, 以  $V$  表示值域。观察发现, 一个点  $x$  涉及到的点仅有  $x \pm 1, x \pm 2d$ , 这暗含了网格图的模式: 一行  $2d$  个点, 那么绝大多数点, 与它相关的只有上下左右四个位置 (每行首尾的点例外)。

然后就可以尝试从网格图的行、列两个方向分析, 设计两个算法。从行的角度分析, 就是扫过数轴时直接状压前  $2d$  个位置有无被覆盖的信息, 复杂度  $O(V2^{2d})$ 。从列的角度, 就是  $O(2^{V/2d})$  枚举第一列方案, 然后对于每种方案  $O(V2^{V/2d})$  在列上做轮廓线 DP, 复杂度  $O(V2^{V/d})$ 。分析可知在  $d \leq 10$  时采取行算法, 在  $d > 10$  时采取列算法, 达到最优复杂度。其中, 列上轮廓线 DP 的部分也存在复杂度更优的网络流做法, 但并非本文重点。

### 4.3 对状态数而非复杂度分析

有两个方向的复杂度是较难分析的: 一是 NP-Hard 问题, 二是数论方向。前者前文已经提到数次, 此处我们考虑后者。数论方向有很多量是难以用复杂度刻画的, 比如说因数数目函数  $d(n)$ , 或者不同质因数数目函数  $\omega(n)$ 。诚然, 存在一些可行的复杂度描述 (对于任意  $\varepsilon > 0$ , 当  $n \rightarrow +\infty$  时  $d(n) = O(n^\varepsilon)$  [8];  $\omega(n) = O(\frac{\log n}{\log \log n})$  [9]), 不过这对于我们解决  $10^{18}$  内的问题是没什么帮助的: 此时  $d(n)$  表现得更像  $n^{1/3}$  甚至更高的次数, 而  $\omega(n) \leq 15$ 。比起分析复杂度, 我们更应该通过搜索、打表等方式, 处理出我们关心的量的级别。时刻牢记, 目标是把操作数压缩在可行的范围内, 而不是用繁琐、复杂的式子来描述复杂度。

以此题验证这一点。

例 4.3 (Burenka, an Array and Queries<sup>8</sup>). 给定一个值域为  $[1, m]$  中整数的、长度为  $n$  的数列,  $q$  次询问, 每次给定区间  $[l_i, r_i]$ , 计算  $p = \prod_{i=l_i}^{r_i} a_i$ , 求  $[1, C]$  中与  $p$  互质的元素数目。数据范围:  $n, q \leq 10^5, m \leq 2 \times 10^4, C \leq 10^5$ 。

对于一次询问, 假设我们已经通过某种操作求出  $p$  包含的质数集合  $S$ , 则可以通过简单容斥得到答案为  $\sum_{T \subseteq S} (-1)^{|T|} \left\lfloor \frac{C}{\prod_{x \in T} x} \right\rfloor$ 。

这种与整除相关的问题, 往往可以将所有质因子分成  $\leq B$  的小因子和  $> B$  的大因子, 其中  $B$  是一个稍后确定的阈值。因为只有  $\prod_{x \in T} x \leq C$  的  $T$  才是有价值的  $T$ , 所以  $T$  中大因子的

<sup>7</sup>XVIII Open Cup named after E.V. Pankratiev, Grand Prix of Ukraine, <https://qoj.ac/problem/6372>

<sup>8</sup>Codeforces Round 814 (Div. 1); <https://codeforces.com/problemset/problem/1718/F>



数目不会太多：具体而言，其不会超过  $\lfloor \log_B C \rfloor$ 。令  $R$  为  $S$  中小因子集合，则我们可以尝试枚举大因子选中了集合  $U$ ，然后答案为  $\sum_U \sum_{V \subseteq R} (-1)^{|U|+|V|} \left\lfloor \frac{C / \prod_{x \in U} x}{\prod_{x \in V} x} \right\rfloor$ 。

$$\text{令 } f(R, C') = \sum_{V \subseteq R} (-1)^{|V|} \left\lfloor \frac{C'}{\prod_{x \in V} x} \right\rfloor, \text{ 然后上式写成 } \sum_U (-1)^{|U|} f\left(R, \left\lfloor \frac{C}{\prod_{x \in U} x} \right\rfloor\right)。$$

如果我们对所有可能的  $(R, C')$  都预处理好  $f(R, C')$ ，那么每次询问时就只需查询不同的  $U$  数目个  $f$  值即可：这是一个在预处理和查询之间的平衡思想。预处理的复杂度是  $O(2^{\pi(B)} \sqrt{C})$ ，其中  $\pi(B)$  为不超过  $B$  的质数数目， $\sqrt{C}$  则是因为整除分块/数论分块 [10] 思想：不同的  $\lfloor C/x \rfloor$  仅有  $O(\sqrt{C})$  个。在不会占用过多时间的情况下，预处理的复杂度最多可以支持  $B = 41$ ，此时  $\pi(B) = 13$ 。此时，大因子的  $U$  数目经实践验证，在可承受范围之内。

到这里本题还剩下最后一步，即对于每次询问求出  $P$ 。因为与本文关系不大就此略去。（但是，上述算法的正确性依赖于可以在合理的操作次数内求出  $P$ ；所以正确的思维顺序应该是，先寻找求出  $P$  的方法，在验证其效率后再进行前述的分析；如果  $P$  无法快速求出，那么上述分析无异于空中楼阁，就必须另寻他解了）

#### 4.4 $\sum_{i=1}^n a_i = A$ 的启示

$\sum_{i=1}^n a_i = A$  的条件能带来很多有趣的结论：

- 不同的  $a_i$  数目是  $O(\sqrt{A})$  的。
- 上一条可以扩展为，所有的  $a_i$  可以分为两类： $\leq \sqrt{A}$  的，和  $O(\sqrt{A})$  个大于  $\sqrt{A}$  的。
- 如果额外有  $a_i \leq n$ （例： $a_i$  是简单图的边数），那么  $\min a_i$  是  $O(\sqrt{A})$  的。

这几条限制都有着很多用处。第一条可以用于构建压缩 trie (Radix Tree, Radix 树 [11])：用长度集合为  $a_i$  的串建立的压缩 Trie 的深度是  $\sqrt{A}$  级别的，进而一些暴力算法可能有较优的复杂度；第二条是很经典的根号分治的思想，在包括众数问题、图论等情形都经常被应用；第三条体现在降低一些搜索问题的复杂度，例如如下例题：

**例 4.4 (Skeleton Dynamization<sup>9</sup>)**. 给定一张  $n$  个点  $m$  条边的简单图，点的编号为  $1 \sim n$ 。找到一种为每个点赋以  $(i, j)$  二元组的方式，使得：

- 存在一组  $T, S$ ，使得对于所有的  $1 \leq i \leq T, 1 \leq j \leq S$ ， $(i, j)$  二元组恰在一个点上，且：
- 若  $i_1 \neq i_2$ ，则  $(i_1, j_1)$  与  $(i_2, j_2)$  所在的点有边当且仅当  $|i_1 - i_2| = 1$  且  $j_1 = j_2$ 。

<sup>9</sup>2020 China Collegiate Programming Contest, Weihai Site, F; <https://codeforces.com/gym/102798/problem/F>



- 对于  $i_1 \neq i_2$  和  $j_1 \neq j_2$ ,  $(i_1, j_1), (i_1, j_2)$  所在的点间与  $(i_2, j_1), (i_2, j_2)$  所在的点间要么同时有边、要么同时无边。

你需要给出任一最大化  $T$  的方案。  $n \leq 10^5, m \leq 2 \times 10^5$

假如通过某种方法, 确定一条边两侧的点  $(u, v)$  分别是  $(i_0, j_0)$  与  $(i_0 + 1, j_0)$ , 则可以:

- 对所有点求出到  $u, v$  的最短路, 则所有到  $u$  更近的点的  $i \in [1, i_0]$ , 所有到  $v$  更近的点的  $i \in [i_0 + 1, T]$ , 这将所有点分为两个点集  $A, B$ 。显然,  $A$  里与  $B$  中点有边的点构成了所有  $i = i_0$  的点集, 而  $B$  里所有与  $A$  中点有边的点构成了  $i = i_0 + 1$  的点集。于是我们现在确定了  $i_0, i_0 + 1$  两层点的点集。
- 与  $i_0$  层中的点有边、且不在  $i_0$  也不在  $i_0 + 1$  的点, 是  $i_0 - 1$  层的点。同理, 该分析可以在  $i, i + 1$  层确定时求出  $i - 1$  层, 也可以在  $i - 1, i$  确定时求出第  $i + 1$  层。一层层确定点, 于是可以将整张图分层。最后就验证该分层是否合法即可。

此部分复杂度  $O(n + m)$ 。

现在主要问题的寻找一条跨层的  $(i_0, j_0), (i_0 + 1, j_0)$ 。此时直接使用上述  $m$  条边的简单图, 度数最小的点的度数是  $O(\sqrt{m})$  级别的结论即可, 即找到度数最小点并枚举其相邻的哪条边是跨层边。总复杂度  $O(\sqrt{m}(n + m))$ 。

## 5 精细复杂性问题初探

除了从数据范围来推理复杂度, 还可以从待解决的的问题来推理复杂度。在思考问题的时候, 一个常见的思路是将原问题转化成其它的问题; 而转化至的问题有可能是一些经典的、有着复杂度下界的问题。因此, 牢记这些经典问题的下界, 也可以助力迅速判定原问题转化的思路是否正确。

研究一些问题复杂度下界的理论, 被称作 Fine Grained Complexity 问题, 即精细复杂性问题。本节将介绍数个精细复杂性理论中较为常用的结论, 并引入归约矩乘的概念, 以便选手在解题时自行判定问题是否可解。

本章专注于实用解的概念, 基本不关注在信息学竞赛方面几乎没有实用性的做法。因此, 一些问题给出的复杂度可能与现在提出的最优复杂度不同。相反, 此处认为 bitset 优化是很有价值的, 并会特别指出某些算法存在 bitset 优化的解法。

### 5.1 矩阵乘法及相关问题

两个  $n \times n$  矩阵的乘法是精细复杂性问题中极其基础的一个环节。截至完稿日, 现行复杂度最优的矩阵乘法已可以在  $O(n^{2.37188})$  [12] 的复杂度内解决, 将来也必然存在更优秀的算



法。但是，在信息学竞赛方面，基本上任何  $\tilde{O}(n^{3-\epsilon})$  的算法都实现过于复杂或常数过大，基本不实用。因此，以下我们认为：

**定理 5.1** (矩阵乘法的复杂度). 竞赛方面，认为两个  $n \times n$  的矩阵乘法的实用复杂度是  $O(n^3)$ 。特别地， $0/1$  矩阵的乘法（其中  $0/1$  间的乘法看作与，加法看作或）可以用 *bitset* 优化至  $O(\frac{n^3}{w})$ ，其中  $w$  是计算机字长，一般取 64。

有若干问题都是可以归约至矩阵乘法的 [13][14]，例如行列式、求逆、有向图上传递闭包、高斯消元。因此，这些问题同样只有  $O(n^3)$  的实用解，其中传递闭包更常见的是 *bitset* 优化解法。

除此之外，还可以将其它一些问题归约为矩阵乘法，以证明问题的一个下界。这种方法被称作归约矩乘。

**例 5.1** (链上数颜色<sup>10</sup>). 给定一棵  $n$  个点、点上带颜色树，多次询问链上不同颜色数目。

令  $m = \sqrt{n/2}$ 。考虑自根挂下  $2m$  条长度为  $m$  的链，令  $A_{i,j}$  表示第  $i$  条链上是否有第  $j$  种颜色， $B_{j,i}$  表示第  $i+m$  条链上是否有第  $j$  种颜色（其中  $1 \leq i, j \leq m$ ），则一切  $A, B$  矩阵都可以被构造。而对第  $i$  条链底点和第  $j$  条链底点的询问，就等价于询问  $A, B$  两矩阵乘积中下标为  $i, j$  处的值，询问  $m^2$  次即可问出整个矩阵。因此，如果原问题中点数和询问数同阶，则原问题的最劣情况不弱于计算  $m \times m$  矩阵乘法，那么其不存在低于  $O(m^3) = O(n\sqrt{n})$  的实用解。

## 5.2 其它经典问题的复杂度下界

以下问题都是精细复杂性问题的经典结论，可在 [15] 自行学习。

**例 5.2** (OV 问题). 给定  $n$  个  $m$  bit 二进制数，询问是否存在两个二进制数与运算的结果是 0。

结论：当  $m = O(n)$  时，OV 问题不存在低于  $O(n^3)$  的实用解（但显然可以 *bitset* 优化）。特别地，当  $m = O(\log n)$  时，在竞赛的数据范围内有显然的  $O(n^2)$  解法（此时的值域仅有  $O(n)$ ）。

**例 5.3** (LCS 问题). 给定两个长度为  $n$  的串，寻找其最长公共子序列。

结论：不存在低于  $O(n^2)$  的解。特别地，存在  $O(\frac{n^2}{w})$  的 *bitset* 做法，读者可自行查找<sup>11</sup>的题解。

**例 5.4** ((min, +) 卷积). 给定长度为  $n$  的序列  $a, b$ ，计算  $c_k = \min_{i+j=k} \{a_i + b_j\}$ 。

结论：不存在低于  $O(n^2)$  的解。特别地，如果  $a, b$  满足凸性，可以用类 *Minkowski* 和做法做到  $O(n)$ 。

<sup>10</sup><https://www.spoj.com/problems/COT2/>

<sup>11</sup><https://loj.ac/p/6564>

例 5.5 (KSUM 问题). 在长为  $n$  的数列中, 寻找大小恰为  $K$  的零和子集。

$3SUM$ 、 $4SUM$  都只有  $O(n^2)$  做法。还有一些问题与之等价, 例如长为三的等差数列计数、三点共线计数、三线共点计数、 $a_i + a_j = a_u + a_v$  计数等。

需要注意的是, 存在与  $A = \max a$  有关的高速做法, 例如朴素的  $FFT$  卷积做法, 当  $K = O(1)$  时, 结合容斥可以做到  $O(A \log A)$ 。

例 5.6 (APSP 问题). 全源最短路问题, 其复杂度下界是  $O(n^3)$  的  $Floyd$  算法。特别地, 使用  $bfs$ , 可以达到  $O(n^2w)$  的时间复杂度, 其中  $w$  是最大边权。

一个与其类似的问题是矩阵的  $(\min, +)$  乘法, 复杂度下界也是  $O(n^3)$ 。

## 总结

各种信息学竞赛入门材料中, 往往会介绍复杂度的概念, 也即本文中提到的“一元复杂度”。然而, 当  $O(nm)$ ,  $O((n+q) \log n)$  这种明显并非一元复杂度的概念被频繁使用时, 笔者意识到在复杂度的定义方面, 当前的教材或许稍有欠缺。借由数学分析相关知识, 笔者引入了多元复杂度的概念, 对各种情形下的复杂度进行了更准确的定义。

算法竞赛的问题总是有解的, 这意味着一定程度上“先射箭再画靶”, 即先假设有解并排除掉那些与数据范围相悖过大的方法, 是可以提高解题效率的。无论是从数据范围倒推算法, 还是验证归约而来的问题是否可在合理复杂度内解决, 本身都是“先射箭再画靶”。然而在解决开放性问题时, 这或许就不是一条很好的思路了。

最后, 精细复杂性问题为计算理论界较为前沿的内容, 笔者对此也没有过多了解, 在这里提出权当抛砖引玉, 读者可自行查阅相关资料更深入了解。

## 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢南京外国语学校张超老师、李曙老师、史钊镭老师对我的教导。

感谢程思元同学、魏佳泽同学、闫陈效同学、汤智铖同学、刘承奥学长与我的沟通交流。

感谢杜冠成同学、蒋承佑同学的验稿。

感谢父母对我的关心和支持。

## 参考文献

- [1] [https://en.wikipedia.org/wiki/Big\\_O\\_notation#Extensions\\_to\\_the\\_Bachmann.E2.80.93Landau\\_notations](https://en.wikipedia.org/wiki/Big_O_notation#Extensions_to_the_Bachmann.E2.80.93Landau_notations)



- [2] 毛嘯. 再探快速傅里叶变换. IOI2016 中国国家候选队论文集, 2016.
- [3] [https://en.wikipedia.org/wiki/Bell\\_number](https://en.wikipedia.org/wiki/Bell_number)
- [4] 吕凯风. 集合幂级数的性质与应用及其快速算法. IOI2015 中国国家候选队论文集, 2015.
- [5] Arlazarov, V.; Dinic, E.; Kronrod, M.; Faradžev, I. (1970), "On economical construction of the transitive closure of a directed graph", Dokl. Akad. Nauk SSSR 194 (11). Original title: "Об экономном построении транзитивного замыкания ориентированного графа", published in Доклады Академии Наук СССР 134 (3), 1970.
- [6] 徐明宽. 非常规大小分块算法初探. IOI2017 中国国家候选队论文集, 2017.
- [7] 李白天. 外代数算法. <https://www.cnblogs.com/Elegia/p/exterior-algebraic-algorithm.html>, 2023.
- [8] Apostol, Tom M. (1976), Introduction to analytic number theory, Undergraduate Texts in Mathematics, New York-Heidelberg: Springer-Verlag, ISBN 978-0-387-90163-3, MR 0434929, Zbl 0335.10001
- [9] Ramanujan. (1917), The normal number of prime factors of a number  $n$ . <http://ramanujan.sirinudi.org/Volumes/published/ram35.pdf>
- [10] <https://oi-wiki.org/math/number-theory/sqrt-decomposition/>
- [11] [https://en.wikipedia.org/wiki/Radix\\_tree](https://en.wikipedia.org/wiki/Radix_tree)
- [12] Duan, Ran; Wu, Hongxun; Zhou, Renfei (2022), Faster Matrix Multiplication via Asymmetric Hashing, arXiv:2210.10173
- [13] [https://link.springer.com/chapter/10.1007/978-3-662-03338-8\\_16](https://link.springer.com/chapter/10.1007/978-3-662-03338-8_16)
- [14] V. Strassen. Gaussian elimination is not optimal. Numer. (MATH), 13:354–356, 1969
- [15] <https://people.csail.mit.edu/virgi/6.s078/>





# 浅谈一种互质数对与最大公约数的维护算法

南京外国语学校 杜冠成

## 摘要

本文首创了一种互质数对与最大公约数的维护算法,并将后者命名为 Dynamic-gcd algorithm,同时结合例题对这两个算法的应用加以阐述,以体现其广泛的应用前景。

## 1 定义与约定

$Prime$  为所有质数构成的集合。

$P(x)$  表示  $x$  的最大质因子。

## 2 互质数对的维护

### 2.1 引入

给定  $n$ , 你需要对初始所有元素均为 1 的序列  $a_1, a_2, \dots, a_n$  进行若干次修改, 使得  $\forall i \in [1, n]$ , 存在一个时刻满足  $\forall a_j = [\gcd(i, j) = 1]$ 。

### 2.2 维护方式

注意到,  $i$  与  $j$  不互质, 当且仅当  $\exists p \in Prime, p|i, p|j$ 。

考虑列出  $n$  以内的所有质数, 并采用暴力搜索的方式, 对于每个质数  $p$ , 考虑  $p$  是否能整除  $i$ , 并将其作为两个分支继续搜索下去。

具体来说, 我们在搜索的过程中, 维护  $x_0, i_0$  及序列  $a$ , 表示目前考虑到了第  $x_0$  小的质数, 且钦定需要整除  $i$  的所有质数乘积为  $i_0$ 。初始  $x_0 = i_0 = 1$  且  $a$  中所有元素均为 1。令  $p_0$  为第  $x_0$  小的质数, 在  $i_0 p_0 \leq n$  的前提下, 若钦定  $p_0$  整除  $i$ , 则将  $a_{p_0}, a_{2p_0}, \dots$  赋值为 0, 并向分支  $(x'_0, i'_0) = (x_0 + 1, i_0 p_0)$  继续搜索; 然后, 从  $(x'_0, i'_0)$  回溯后, 撤销对  $a_{p_0}, a_{2p_0}, \dots$  的修改, 并向分支  $(x'_0, i'_0) = (x_0 + 1, i_0)$  搜索。

可以发现, 若在搜索过程中, 所有钦定整除  $i$  的质数构成的集合恰好与某个  $i$  的质因子集合相同, 则该时刻满足条件。换言之, 若对于每个  $i$ , 求出  $i$  的所有质因子的乘积  $t_i$ , 则当



$i_0 = t_i$  时, 该时刻恰好满足关于  $i$  的要求。注意到这个时刻总存在, 因此前述算法的正确性得到了保障。

注意到, 当  $i_0 p_0 > n$  时, 此后总有  $i_0 p_0 > n$ , 那么不会再额外钦定任意质数整除  $i$ 。此时直接返回即可。

以下是函数  $\text{dfs}(x_0, i_0)$  的伪代码:

---

**Algorithm 1** 互质数对的维护

---

```

1: $p_0 :=$ the x_0 -th smallest prime
2: if $i_0 p_0 > n$ then
3: for each $t(i) == i_0$ do
4: claim requirements of i have been satisfied
5: end for
6: else
7: $\text{dfs}(x_0 + 1, i_0)$
8: for each j which is a multiple of p_0 do
9: $a_j := 1$
10: end for
11: $\text{dfs}(x_0 + 1, i_0 p_0)$
12: for each j which is a multiple of p_0 do
13: revoke operations on a_j to recover
14: end for
15: end if

```

---

## 2.3 时间复杂度分析

首先,  $\text{dfs}$  本身的时间复杂度为  $\Theta(n)$ , 的瓶颈在于枚举  $p_0$  倍数的部分。

注意到  $i_0 p_0$  的最大质因子为  $p_0$ , 而  $\text{dfs}$  过程中所有  $i_0 p_0$  两两不同, 因此时间复杂度为  $O\left(\sum_{i=1}^n \frac{n}{P(i)}\right)$ 。

Erdos, Ivic, Pomerance 的论文中给出<sup>1</sup>

$$\sum_{i=1}^n \frac{1}{P(i)} = n\delta(n) \left( 1 + O\left(\frac{\log \log n}{\log n}\right)^{\frac{1}{2}} \right)$$

这个算法的实际运行效率相当优秀, 在  $n \leq 10^6$  时, 其时间复杂度可以看做  $\Theta(n\sqrt{n})$ 。

---

<sup>1</sup>[1] P. Erdos, A. Ivic and C. Pomerance, On sums involving reciprocals of the largest prime factor of an integer. Glasnik Matematički. 21(41) 1986, 283-300.

## 2.4 例题 1

### 2.4.1 题目大意

给定长度为  $n$  的字符串  $s$ ，记  $s[l, r]$  ( $1 \leq l \leq r \leq n$ ) 为  $s$  第  $l, l+1, \dots, r$  的个字符依次拼接而成的串。

我们称非空字符串  $t$  是「被提及的」，当且仅当  $\exists 1 \leq l \leq r \leq n, \gcd(l, r) = 1, s[l, r] = t$ 。

求所有被提及的不同非空字符串  $t$  的长度和。

保证  $1 \leq n \leq 10^5$ ， $s$  中只有小写字母。<sup>2</sup>

### 2.4.2 题目解法

考虑建出 SAM，令节点  $u$  对应的子串长度在  $L(u), R(u)$  间，子树内所有 endpos 组成集合  $E$ ，则对答案的贡献为

$$\sum_{i=L(u)-1}^{R(u)-1} [\exists x \in E, \gcd(i, x) = 1](i+1)$$

考虑全局地枚举每个  $i$ ，计算  $i+1$  对答案的贡献次数。

对于所有与  $i$  互质的数  $k$ ，若前缀  $[1, k]$  在 SAM 上对应的节点为  $t$ ，则对于  $t$  到根路径上的每个点  $u$ ，只要  $i \in [L(u)-1, R(u)-1]$ ，就能产生贡献。

考虑套用互质数对的维护算法，爆搜  $i$  各质因子构成的集合  $S$ ，同时维护所有  $t$  构成的集合  $T$ ，以及所有  $i$  的贡献次数  $c_i$ 。每当钦定  $p_0 | i$  时， $\forall p_0 | k_0$ ，记前缀  $[1, k_0]$  在 SAM 上对应的节点为  $t_0$ ，若  $t_0 \in T$ ，则将  $t_0$  剔除出  $T$ 。对于每个  $S$ ，枚举所有质因子集合  $= S$  的  $i$ ，更新答案即可。

具体来说，当  $t_0$  将被剔除出  $T$  时，若对  $c$  有影响，会存在一条链，链上所有的  $u$  由合法变为不合法。由于链上各点的  $[L, R]$  是连续的，对  $c$  的影响始终是区间减，直接根号平衡即可。

时间复杂度  $\Theta(n\sqrt{n})$ 。

## 3 最大公约数的维护

### 3.1 引入

给定  $n$ ，你需要对初始所有元素均为 1 的序列  $a_1, a_2, \dots, a_n$  进行若干次修改，使得  $\forall i \in [1, n]$ ，存在一个时刻满足  $\forall a_j = \gcd(i, j)$ 。

<sup>2</sup>2023 ICPC Asia Xian Regional Contest C

### 3.2 维护方式

注意到,  $\gcd(i, j)$  为所有满足  $p^\alpha | i, p^\alpha | j (p \in \text{Prime}, \alpha \in \mathbb{N}^+)$  的  $p$  的乘积。

类似互质数对的维护, 我们列出  $n$  以内的所有质数, 并暴力搜索。对于每个质数  $p$ , 枚举  $i$  质因数分解后  $p$  的最高幂次  $\beta$ , 将其作为多个分支继续搜索下去。在此过程中, 维护  $x_0, i_0$  及序列  $a$ , 表示考虑到了第  $x_0$  小的质数, 而  $i_0$  表示所有  $p^\beta$  的乘积。

在枚举  $\beta$  的过程中, 每当  $\beta$  增大 1 后, 若  $i_0 p^\beta \leq n$ , 则将所有满足  $p^\beta | j$  的  $a_j$  乘  $p$ , 向分支  $(x_0 + 1, i_0 p^\beta)$  继续搜索。在  $(x_0, i_0)$  的所有分支处理完毕后, 对于所有满足  $p | j$  的  $a_j$ , 撤销对  $a_j$  的所有乘  $p$  操作。

根据算术基本定理,  $\forall i \in [1, n]$ , 都存在某个时刻满足  $i_0 = i$ , 算法的正确性得到了保障。2.2 节提出的减枝, 同样适用于此处。

以下是函数  $\text{dfs}(x_0, i_0)$  的伪代码:

---

**Algorithm 2** 最大公约数的维护

---

```
1: $p_0 :=$ the x_0 -th smallest prime
2: if $i_0 p_0 > n$ then
3: claim requirements of $i = i_0$ have been satisfied
4: else
5: $\text{dfs}(x_0 + 1, i_0)$
6: for each $\beta \in \mathbb{N}^+$ satisfying $i_0 p_0^\beta \leq n$ do
7: for each j which is a multiple of p_0^β do
8: $a_j := p a_j$
9: end for
10: $\text{dfs}(x_0 + 1, i_0 p_0^\beta)$
11: end for
12: for each j which is a multiple of p_0 do
13: revoke operations on a_j to recover
14: end for
15: end if
```

---

可以看出, 此算法能够通过一定次数的修改, 动态维护每一对  $n$  以内正整数的最大公约数, 因此我将其命名为 Dynamic-gcd algorithm。

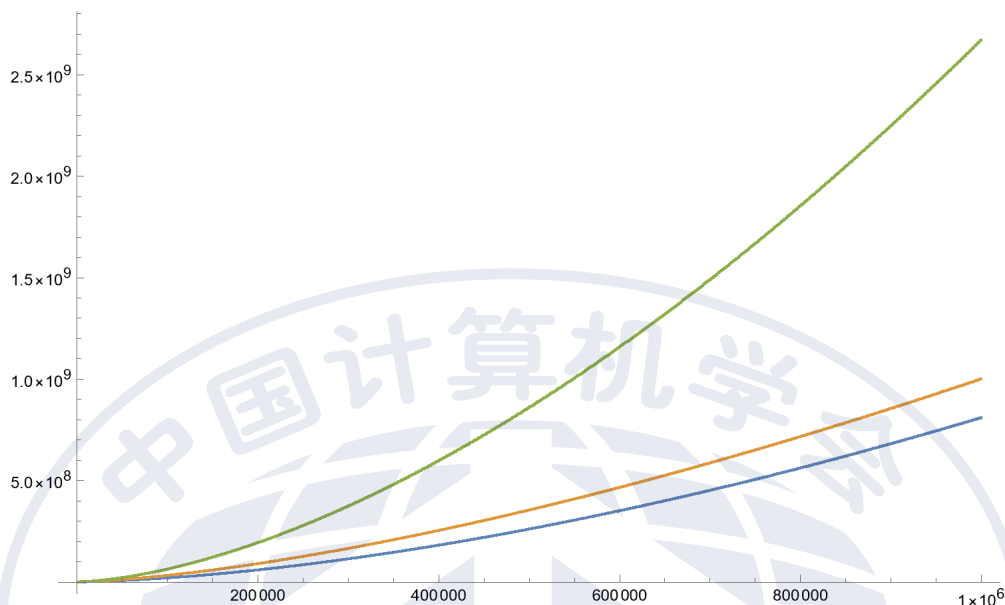
### 3.3 时间复杂度分析

$\text{dfs}$  本身的时间复杂度是  $\Theta(n)$ , 复杂度的瓶颈在于枚举  $p_0^\beta$  的倍数。

注意到  $i_0 p_0^\beta$  的最大质因子为  $p_0$ , 而  $\text{dfs}$  过程中所有  $i_0 p_0^\beta$  两两不同, 因此时间复杂度仍为  $O\left(\sum_{i=1}^n \frac{n}{P(i)}\right)$ 。

在实际运行效率方面, 当  $n \leq 10^6$  时, 其时间复杂度可看做  $\Theta(n\sqrt{n})$ , 但常数较大。

下图蓝线、绿线分别为互质数对维护算法、Dynamic-gcd algorithm 中  $a_j \leftarrow 1$  和  $a_j \leftarrow pa_j$  的执行次数关于  $n$  的函数图像, 黄线表示  $f(x) = x\sqrt{x}$  的函数图像。



## 3.4 例题 2

### 3.4.1 题目大意

给定长度为  $n$  的序列  $A$ ,  $T$  次询问, 每次给定正整数  $x, l, r$ , 求  $A_{\gcd(x,l)}, A_{\gcd(x,l+1)}, \dots, A_{\gcd(x,r)}$  的最大子段和。

保证  $1 \leq T \leq 10^6, 1 \leq n \leq 5 \times 10^4, 1 \leq x, l, r \leq n, l \leq r$ 。

### 3.4.2 题目解法

考虑 Dynamic-gcd algorithm, 在维护  $a$  的同时, 维护序列  $b$ , 满足  $b_i = A_{a_i}$ 。每次对  $a$  作单点修改后, 对  $b$  也进行单点修改。

将所有询问统一离线处理后, 问题转化为:  $O(n\sqrt{n})$  次单点修改,  $T$  次查询区间最大子段和。使用线段树维护即可, 时间复杂度  $\Theta((n\sqrt{n} + T) \log n)$ 。

## 4 总结

本文提出了互质数对的维护算法，与最大公约数的维护算法 Dynamic-gcd algorithm，通过例题可以看出，它们可以通过较少次数的修改，动态维护每一对  $n$  以内正整数间互质性、最大公约数的信息，在与互质、gcd 有关的数论、数据结构等题型中，均有广阔的应用前景。

笔者希望本文能起到抛砖引玉的作用，吸引读者对数论信息的动态维护进行深入研究。

## 5 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢南京外国语学校张超老师对我的教导。

感谢家人对我的关心与支持。

## 参考文献

[1] P. Erdos, A. Ivic and C. Pomerance, On sums involving reciprocals of the largest prime factor of an integer. Glasnik Matematicki. 21(41) 1986, 283-300.



# 浅谈计算理论与 OI 中的难解问题

南京外国语学校 闫陈效

## 摘要

本文从图灵机的定义出发，证明了一系列问题的 NP 完全性，说明了为什么 NP 完全问题是困难的，并帮助 OI 选手在思考题目的过程中更快地识别 NP 完全问题，提高思考效率。

## 1 引言

### 1.1 内容简介

计算理论是理论计算机科学的重要分支。本文围绕计算理论中的基本计算模型——图灵机，严格定义了计算和计算复杂性等概念，引入了几个问题类，使用归约的方法展示了如何证明问题的困难性，并给出了一系列经典的 NPC 问题。

本文共由五个章节组成。

第二节从图灵机出发，介绍了一些计算模型的概念和性质。

第三节介绍了时间复杂性，围绕 P、NP 等几个问题类进行了讨论。

第四节从 SAT 问题出发，通过归约证明了一系列 NPC 问题。

第五节作简要的总结。

### 1.2 记号与约定

在不引起歧义的情况下，统一使用记号  $\perp$  表示不存在、未定义、空白字符等含义。

记  $\Sigma$  为一个固定的有限集合，称为**字符集**，其中的元素称为**字符**。无需关心  $\Sigma$  的具体值，但默认其包含一些特殊字符 0, 1，且不包含特殊字符  $\perp$ 。另外，记  $\Sigma_0 = \Sigma \cup \{\perp\}$ 。

称有限的字符序列为**字符串**，即  $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$  中的元素，其中  $\Sigma^n$  表示笛卡尔积。

设字符串  $x \in \Sigma^*$ 。记  $|x|$  表示  $x$  的**长度**，满足  $x \in \Sigma^{|x|}$ 。记  $x_i$  表示  $x$  的第  $i$  个字符，其中  $1 \leq i \leq |x|$ 。

对于集合  $S$ ，记  $\mathcal{P}(S) = \{T \mid T \subseteq S\}$  表示  $S$  的**幂集**。

记  $u = O(v)$  表示存在  $c > 0$  使得  $u \leq cv$ ， $u = \Omega(v)$  表示  $v = O(u)$ ， $u = \Theta(v)$  表示  $u = O(v)$  且  $u = \Omega(v)$ ， $u = \text{poly}(v)$  表示存在  $c > 0$  使得  $u = O(v^c)$ 。





## 2 计算模型

### 2.1 问题

**定义 2.1 (问题).** 一个  $\Sigma^* \rightarrow \Sigma^*$  的函数  $f$  称为一个问题。特殊地, 如果值域  $f(\Sigma^*) \subseteq \{0, 1\}$ , 则称  $f$  是一个判定性问题。

可以体会该定义的合理性: 计算机的任务就是输入一个字符串, 并根据计算意图得到一个输出字符串, 而  $f$  则给出了该计算意图。

**定义 2.2 (语言).** 一个字符串集合  $L \subseteq \Sigma^*$  称为一个语言。

明显地, 判定性问题和语言形成双射, 只要取语言  $L$  的特征函数

$$f: \Sigma^* \rightarrow \{0, 1\}, x \mapsto \begin{cases} 1, & x \in L; \\ 0, & x \notin L \end{cases}$$

即可。下文不再仔细区分这两者。

### 2.2 图灵机

**定义 2.3 (图灵机<sup>1</sup>).** 一台  $k$  带图灵机是一个二元组  $M = (Q, \delta)$ , 其中:  $Q$  是有限集合, 称为状态集, 至少包含初始状态  $S$  和停机状态  $H$ ;  $\delta$  是一个  $(Q \setminus \{H\}) \times \Sigma_0^k \rightarrow Q \times \Sigma_0^k \times \{-1, 0, 1\}^k$  的函数, 称为转移函数。

接下来描述  $k$  带图灵机  $M$  的运行规则。

假想有  $k$  条两端无限延伸、线性排列着方格的纸带, 初始时将输入字符串写上去, 然后机器的读写头读取当前  $k$  个字符, 根据转移函数自动修改字符和状态并左右移动读写头, 一直如此运行直到停机。

形式化地, 图灵机先接受一个输入  $x \in \Sigma^*$ 。接下来的运行时间分为若干个时刻, 时刻  $t$  (其中  $t \in \mathbb{N}$ ) 由一个三元组  $\sigma_t = (q_t, A_{t,1 \sim k}, h_{t,1 \sim k})$  描述, 称为  $M$  在读入  $x$  后时刻  $t$  的格局。其中,  $A_{t,i}$  表示第  $i$  条纸带上的内容, 它是一个  $\mathbb{Z} \rightarrow \Sigma_0$  的映射;  $h_{t,i} \in \mathbb{Z}$  表示第  $i$  条纸带上的读写头的位置。

时刻 0 的格局  $\sigma_0$  由

$$q_0 = S, \quad h_{0,i} = 1, \quad A_{0,i}(j) = \begin{cases} x_j, & i = 1, 1 \leq j \leq |x|; \\ \perp, & \text{otherwise} \end{cases}$$

<sup>1</sup>图灵机有多种等价定义, 本文选取了较为便于说明问题的一种

初始化, 其中  $\perp$  是空白字符。对于时刻  $t+1$ , 格局  $\sigma_{t+1}$  依赖于  $\sigma_t$ , 具体地: 设转移指令  $\Delta_t = \delta(q_t, (A_{t,h_{t,i}})_{i=1}^k) = (q', (A'_i)_{i=1}^k, (d_i)_{i=1}^k)$ , 则  $\sigma_{t+1}$  由

$$q_{t+1} = q', \quad h_{t+1,i} = h_{t,i} + d_i, \quad A_{t+1,i}(j) = \begin{cases} A_{t,i}(j), & j \neq h_{t,i}; \\ A'_i, & j = h_{t,i} \end{cases}$$

给出; 特殊地, 如果  $q_t = H$ , 则图灵机在时刻  $t$  停机, 对  $t_0 > t$  都有  $\sigma_{t_0}$  不存在, 记作  $\sigma_{t_0} = \perp$ 。我们将格局  $\sigma_t$  由转移指令  $\Delta_t$  到达格局  $\sigma_{t+1}$  的过程记作  $\sigma_t \xrightarrow{\Delta_t} \sigma_{t+1}$ ; 特殊地,  $\perp \xrightarrow{\perp} \perp$ 。

如果  $M$  在读入  $x$  后运行至停机, 设停机时刻的第  $k$  条纸带为  $A_H$ , 如果存在  $r \in \mathbb{N}$  使得  $A_H(i) = \perp$  当且仅当  $i < 1$  或  $i > r$ , 则记字符串  $M(x) = A_H(1)A_H(2) \cdots A_H(r)$  表示计算结果。对于其他情况 (要么不停机, 要么停机时刻第  $k$  条纸带格式错误), 一律记  $M(x) = \perp$ 。

称  $M$  计算了问题  $f$ , 当且仅当对  $x \in \Sigma^*$ ,  $f(x) = M(x)$  总成立。如果存在图灵机计算问题  $f$ , 则称  $f$  是可计算的。特殊地, 如果  $f$  是判定性问题, 设其对应语言  $L$ , 则说图灵机判定  $L$  或  $L$  是可判定的。

## 2.3 非确定图灵机

**定义 2.4** (非确定图灵机). 一台  $k$  带非确定图灵机是一个二元组  $N = (Q, \delta)$ , 其中:  $Q$  是有限集合, 称为状态集, 至少包含初始状态  $S$  和停机状态  $H$ ;  $\delta$  是一个  $(Q \setminus \{H\}) \times \Sigma_0^k \rightarrow \mathcal{P}(Q \times \Sigma_0^k \times \{-1, 0, 1\}^k)$  的函数, 称为转移函数。

作为图灵机的推广概念, 非确定图灵机的定义只有  $\delta$  的上域不一样: 原先是一个转移指令, 现在变成了一个转移指令的集合。非确定图灵机运行时, 每一时刻都应从当前转移指令的集合里任意选一个, 并按照上文中图灵机的描述去执行。如果认为图灵机的每个格局都只有不超过一个后继格局, 则其构成链状 (有限或无限长); 那么非确定图灵机的格局则构成树状, 有些路径有限长 (停机于叶子节点), 有些路径无限长 (不停机)。

在本文中, 我们只讨论非确定图灵机计算判定性问题的情形。对于非确定图灵机  $N$  和输入  $x$ , 考虑转移指令序列  $\{\Delta_i\}_{i \geq 0}$ , 如果  $\sigma_0 \xrightarrow{\Delta_0} \sigma_1 \xrightarrow{\Delta_1} \sigma_2 \xrightarrow{\Delta_2} \cdots \xrightarrow{\Delta_{n-1}} \sigma_n \xrightarrow{\Delta_n} \cdots$  且对所有  $\Delta_i \neq \perp$  都有  $\Delta_i \in \delta(q_i, (A_{i,h_{i,i}})_{i=1}^k)$ , 则称  $\Delta$  是  $N$  的一条转移路径, 记  $N(x; \Delta)$  为计算结果。

那么, 如果存在转移路径  $\Delta$  使  $N(x; \Delta) = 1$ , 则记  $N(x) = 1$ , 否则  $N(x) = 0$ 。称  $N$  计算判定性问题  $f$  或判定其对应语言  $L$ , 当且仅当对任意  $x \in \Sigma^*$  总成立  $f(x) = N(x)$ 。直观上说, 非确定图灵机认为  $x \in L$  当且仅当至少一种转移路径认为  $x \in L$ 。

## 2.4 时间函数

**定义 2.5** (完全图灵机<sup>2</sup>). 如果图灵机  $M$  输入任何  $x \in \Sigma^*$  后都停机, 则称  $M$  为一台完全图灵机。类似地, 如果非确定图灵机  $N$  输入任何  $x \in \Sigma^*$  后对任意转移路径都停机, 则称  $N$  为一

<sup>2</sup>这并不是通用术语, 仅限于本文使用



台完全非确定图灵机。

明显地，一台图灵机是完全的当且仅当它计算了某个问题；但是对非确定图灵机却不然。我们认为这样总是停机的（非确定）图灵机的性质是较好的，下文的讨论也大多是围绕它们。

**定义 2.6 (时间函数).** 对于完全图灵机  $M$ ，记  $\mathbb{N} \rightarrow \mathbb{N}$  的时间函数  $T_M$ ，使得  $T_M(n)$  为  $M$  接受任意长度不超过  $n$  的输入后最大的停机时刻，此时称  $M$  在  $T_M(n)$  时间内停机。对完全非确定图灵机  $N$  有类似定义， $T_N(n)$  为  $N$  接受任意长度不超过  $n$  的输入后的任意转移路径中最大的停机时刻。

一般地，我们讨论时间函数的渐近级别。如果  $T_{M_1}(n) = O(T_{M_2}(n))$ ，则认为  $M_1$  计算得大约不慢于  $M_2$ ；如果  $T_{M_1}(n) = \Theta(T_{M_2}(n))$ ，则认为它们的计算时间差不多。

## 2.5 编码

（非确定）图灵机的输入和输出只能是一个基于有限字符集的字符串，这是为了在理论上方便讨论而做的约束。然而，很多时候我们希望以自然数、数据结构（如树、图）等等非字符串的对象作为输入和输出，此时就要给它们编码，即构造一个到  $\Sigma^*$  的单射。

长度为  $n$  的字符串是有限的，所以字符串的总数是可数的，即  $|\Sigma^*| = \aleph_0$ 。所以，能给某类对象编码的一个必要条件是它的数量是可数的。除此之外，还需要保证凭图灵机的计算能力能成功解读输入的编码的信息。事实上，通常不必对这点做过多担心，因为图灵机的计算能力与现实世界中基于冯·诺依曼结构的计算机相当（本文不对此进行说明），而读者应当非常熟悉如何使用后者编程处理各种数据。

对于（非确定）图灵机  $M$ ，我们记  $M(a, b, c, \dots) = (A, B, C, \dots)$  为将一系列非字符串的数据编码成字符串输入给  $M$  后的非字符串计算结果的编码。不必过于严格与仔细地讨论具体的编码方式，以及  $M$  如何识别编码，这通常并不重要。如果该编码不是到  $\Sigma^*$  的满射，并且希望  $M$  总是停机，此处默认  $M$  需要额外检查输入是否合法（即是否对应某个原始数据），但这也通常是平凡的。

**定理 2.1.** 存在一个易于解读的对全体（非确定）图灵机的编码。

证明. 对于大小相同的两个有限集  $Q_1, Q_2$ ，以  $Q_1$  为状态集的图灵机总是等价于某个以  $Q_2$  为状态集的图灵机，所以我们只需关心状态集的大小  $n$ 。确定  $n$  后，不同的转移函数的数量是有限的，所以（非确定）图灵机的总数是可数的。

接下来讨论编码方式。首先确定状态集  $Q$  和字符集  $\Sigma_0$  内元素的一个排列顺序，那么转移函数就可以表示为一个自然数序列到自然数序列（的集合）的映射，按字典序拼接（配合有限个分隔符）即可。对一个自然数序列编码，只要将每个自然数用二进制写出，在中间插入分隔符即可。□

## 2.6 通用图灵机

有了对（非确定）图灵机  $M$  的编码，自然会考虑将其输入给某个（非确定）图灵机，让它来模拟  $M$ 。这个能模拟所有  $M$  的机器称为**通用（非确定）图灵机**。下面将说明它的存在性和模拟效率。

**定理 2.2.** 存在一个单带图灵机  $U$ ，使得对所有图灵机  $M$  和  $x \in \Sigma^*$ ，有  $U(M, x) = M(x)$ ，并且如果  $M(x)$  的计算在时刻  $T$  内停机，则  $U(M, x)$  的计算在时刻  $\text{poly}(|M|T)$  内停机（其中  $|M|$  表示  $M$  编码的长度）。

证明. 设  $M$  是  $k$  带图灵机，首先需要考虑如何用  $U$  的一条纸带存储  $k$  条纸带的信息。一个巧妙的方法是将  $\mathbb{Z}$  分成任意的  $m$  个剩余类，这样就可以达到独立的  $m$  条纸带的效果，在剩余类之间跳转可以用  $O(m)$  个状态来控制。此处，我们花 1 个剩余类存储  $M$  的编码信息以随时调用，花  $2k$  个剩余类信息存储纸带，其中每个纸带占用连续的两个剩余类，一个存储字符信息，另一个存储 0 或 1 表示读写头是否位于此处。

这样，在  $M(x)$  计算中的时刻  $t$ ， $U$  纸带上的有效位置被包含在长度为  $O(|M|+kt) = O(|M|t)$ （总可以假定  $k \leq |M|$ ）的一段区间上，该时刻  $U$  对  $M$  的模拟容易在这个长度的多项式时间内完成，来回扫描并用状态记录信息即可。那么总时间就是  $\sum_{t=1}^T \text{poly}(|M|t) = \text{poly}(|M|T)$ 。□

**定理 2.3.** 存在一个单带非确定图灵机  $U$ ，使得对所有非确定图灵机  $N$  和  $x \in \Sigma^*$ ，有  $U(N, x) = N(x)$ ，并且如果  $N(x)$  的计算中的所有转移路径都在时刻  $T$  内停机，则  $U(N, x)$  的计算中的所有转移路径都在时刻  $\text{poly}(|N|T)$  内停机。

证明. 大致与定理 2.2 同理。在模拟  $N$  从一个格局转移到某个子格局时， $U$  的中间过程采用确定性的转移（即转移指令集合大小为 1）。□

## 2.7 不同模型的比较

### 2.7.1 单带与多带

明显地， $k+1$  带（非确定）图灵机看上去应该完全强于  $k$  带（非确定）图灵机，因为前者可以选择不在第 2 条纸带上工作，进而直接转化为后者。对于  $k=1$  情况稍有微妙，因为单带（非确定）图灵机的输入带和输出带合为一体，但也可以通过将第一条带复制到第二条带来简单地解决。

现在的一个问题是， $k$  带（非确定）图灵机是否能转化到  $k+1$  带（非确定）图灵机呢？事实上，之前对通用图灵机的讨论已经给出了答案。

**定理 2.4.** 如果完全  $k$  带（非确定）图灵机  $M_k$  计算问题  $f$ ，则存在完全单带（非确定）图灵机  $M_1$  计算  $f$ ，且  $T_{M_1}(n) = \text{poly}(T_{M_k}(n))$ 。

证明. 考虑让  $M_1$  输入  $x$  后, 将  $M_k$  的编码也复制进去 (注意, 我们讨论的  $M_k$  是常量), 然后转接到通用 (非确定) 图灵机  $U$  的算法即可。由于  $|M_k|$  是常量, 它在  $\text{poly}$  里可以忽略。□

至此, 我们知道带数不同的 (非确定) 图灵机所能计算的问题集合, 即 **计算能力**, 是等价的; 同时, 计算时间也可以多项式级别地相互限制。于是, 我们能够使用更便于构造算法的多带 (非确定) 图灵机来研究单带 (非确定) 图灵机的性质。

### 2.7.2 图灵机与非确定图灵机

图灵机是特殊的非确定图灵机, 所以后者的计算能力包含前者。反过来也有类似结论: 图灵机的计算能力等价于完全非确定图灵机。

**定理 2.5.** 如果完全非确定图灵机  $N$  计算问题  $f$ , 则存在图灵机  $M$  计算  $f$ , 且  $T_M(n) = \text{poly}(n) \exp O(T_N(n))$ 。

证明. 我们只考虑  $N$  是单带非确定图灵机, 随后多带的情况由定理 2.4 可立刻得到。

注意到  $N(x)$  的计算中的转移情况是一棵树, 每个节点的儿子数量有常数上界, 且深度最多是  $T_N(|x|) + 1$ , 于是节点数是  $\exp O(T_N(|x|))$ 。考虑用  $M$  模拟 BFS 遍历这棵树。令  $M$  是双带的, 第一行原样存储当前  $N$  的纸带, 第二行维护队列。队列存储每个节点代表的格局, 其大小是  $O(|x|)$  的。队列操作容易在多项式时间内完成。□

需要注意的是, 存在一些非确定图灵机能计算但是图灵机计算不了的问题, 本质原因是非确定图灵机不会因为不停机而得到  $\perp$ 。

## 3 时间复杂性

### 3.1 几个问题类

**定义 3.1.** 记  $\mathbf{P}$  为所有满足  $T_M(n) = \text{poly}(n)$  的图灵机  $M$  计算的判定性问题集合;

记  $\mathbf{EXP}$  为所有满足  $T_M(n) = \exp \text{poly}(n)$  的图灵机  $M$  计算的判定性问题集合;

记  $\mathbf{NP}$  为所有满足  $T_N(n) = \text{poly}(n)$  的非确定图灵机  $N$  计算的判定性问题集合;

记  $\mathbf{NEXP}$  为所有满足  $T_N(n) = \exp \text{poly}(n)$  的非确定图灵机  $N$  计算的判定性问题集合。

我们已经证明: 不同带数的 (非确定) 图灵机  $M_1, M_2$  计算相同的问题集合, 且前者的时间函数顶多是后者的多项式, 反之亦然。于是便有了上述不依赖于带数的定义。

**定理 3.1.**  $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP} \subseteq \mathbf{NEXP}$ 。

证明. 由于图灵机是特殊的非确定图灵机, 显然  $\mathbf{P} \subseteq \mathbf{NP}$  和  $\mathbf{EXP} \subseteq \mathbf{NEXP}$ 。而定理 2.5 则说明  $\mathbf{NP} \subseteq \mathbf{EXP}$ 。□



关于是否有  $P = NP$  成立，这是一个极其困难的问题，至今<sup>3</sup>没有被证明或否定。

### 3.2 用图灵机描述 NP

**定义 3.2** (验证机). 对于完全图灵机  $M$  和语言  $L$ ，如果  $x \in L$  当且仅当存在  $c \in \Sigma^*$  使得  $M(x, c) = 1$ ，则称  $M$  是  $L$  的一个验证机。这样的  $c$  为  $x \in L$  提供了证明，称  $c$  为  $x$  的证书。验证机  $M$  的时间函数  $T_M^L(n)$  定义为对任意  $|x| \leq n$  和任意  $c$ ， $M(x, c)$  的计算的最大停机时刻。

**定理 3.2.** 语言  $L \in NP$  当且仅当存在一个  $L$  的验证机  $M$ ，使得  $T_M^L(n) = \text{poly}(n)$ 。

证明. 首先，对于在  $\text{poly}(|x|)$  时间内停机的验证机  $M$ ，长度超过  $\text{poly}(|x|)$  的证书都是无效的，因为此时  $M$  甚至不能完整地读取这些证书。这里默认  $M$  必须在面对这些证书时及时掐停自己，这是不难做到的。

这样，有效的证书只有  $\text{exp poly}(|x|)$  个，那就可以用一台每条转移路径都在  $\text{poly}(|x|)$  时间内停机的非确定图灵机  $N$  来枚举它们，然后接入  $L$  的验证算法。

反过来，如果有多项式时间的非确定图灵机  $N$  判定  $L$ ，可以构造  $M$  使得它将证书  $c$  看作  $N(x)$  的某条转移路径，一路模拟下去并输出该转移路径的计算结果。□

通俗地说，一个语言能被非确定图灵机在多项式时间内判定，等价于能被图灵机在多项式时间内验证。非确定图灵机的非确定性在这里用于枚举证书。

### 3.3 神谕与归约

**定义 3.3** (神谕图灵机). 神谕图灵机是图灵机的一种扩展，记作  $M^?$ ，其中  $?$  处可以填入一个问题  $f$ 。 $M^f$  比普通图灵机额外多一条纸带（称为神谕带）和两个状态  $q_{\text{query}}, q_{\text{answer}}$ ，当进入  $q_{\text{query}}$  状态时，如果神谕带上从位置 1 往后到第一个空白字符之前的字符串为  $x$ ，则该步计算使神谕带变为  $f(x)$ （从位置 1 开始，没覆盖到的位置变为空白字符），并进入  $q_{\text{answer}}$  状态。

**定义 3.4** (归约). 对于问题  $f$  和  $g$ ，如果存在神谕图灵机  $M^g$  计算  $f$ ，则称  $f$  可图灵归约到  $g$ ，记  $f \leq_T g$ ；如果满足  $T_{M^g}(n) = \text{poly}(n)$ ，则称  $f$  可库克归约到  $g$ ，记  $f \leq_C g$ 。

**定理 3.3.** 如果  $f \leq_T g \leq_T h$ ，则  $f \leq_T h$ ；对库克归约类似。

**定理 3.4.** 如果  $f \leq_T g$ ，则  $g$  可计算蕴含  $f$  可计算。如果  $f \leq_C g$ ，则  $g$  多项式时间可计算蕴含  $f$  多项式时间可计算；特殊地，如果  $f$  和  $g$  都是判定性问题，则  $g \in P$  蕴含  $f \in P$ 。

证明. 先考虑图灵归约。设图灵机  $M'$  计算  $g$ ，我们只需要将神谕图灵机  $M^g$  转化为一个能接入子程序  $M'$  的图灵机  $M$  即可。这一点不难做到：只要将  $M'$  的纸带全部置于  $M$  下方，进

<sup>3</sup>笔者撰写本文的时间是 2024 年 1 月



入  $q_{\text{query}}$  后只用新增的纸带跑  $M'$  的算法,  $M'$  停机后将最后一条纸带的内容复制到神谕带并清空新增纸带, 然后进入  $q_{\text{answer}}$  即可。

对于库克归约,  $M^g$  输入  $x$  后任意时刻神谕带上的字符串长度都不超过  $T_{M^g}(|x|)$ , 且进入  $q_{\text{query}}$  最多  $T_{M^g}(|x|)$  次, 因此  $M$  的计算时间不超过  $O(T_{M^g}(|x|)T_{M'}(T_{M^g}(|x|))) = \text{poly}(|x|)$ 。□

归约为问题难度的比较提供了工具: 通过给出假如能够计算问题  $g$  (即拥有一个  $g$  的神谕) 之后如何能够计算问题  $f$ , 说明  $f$  不难于  $g$ 。

### 3.4 NP-Hard 和 NPC 问题

**定义 3.5.** 记 **NP-Hard** 为满足以下条件的问题  $f$  的集合: 对每个  $g \in \text{NP}$  都有  $g \leq_c f$ 。记  $\text{NPC} = \text{NP} \cap \text{NP-Hard}$ , **NPC** 问题即 **NP** 完全问题。

这就是说, **NP-Hard** 问题在多项式时间计算的意义下不弱于所有 **NP** 问题。而 **NPC** 问题本身又是 **NP** 问题, 所以是 **NP** 问题中最难的一类。

很难想象 **NPC** 问题的存在, 但下文将会给出很多实例。

**定理 3.5.** 对于  $f \in \text{NP-Hard}$ , 如果又有  $f \in \text{P}$ , 则  $\text{P} = \text{NP}$ 。

根据  $\text{P} \stackrel{?}{=} \text{NP}$  问题的困难性, 如果证明了某个问题在 **NP-Hard** 中, 则可以认为该问题不太可能有多项式时间的算法——即使有, 也是极其困难的。

OI 中的问题几乎都要求给出多项式时间的算法 (除非数据范围非常小, 这是容易识别的)。选手在思考问题的过程中, 如果将原问题泛化得到了一个 **NP-Hard** 问题, 便可以尽快放弃这一思路了。这对提高选手思考问题的效率很有帮助。

### 3.5 不可计算问题

图灵机的计算能力虽然强大, 但仍然几乎计算不了所有问题。

一方面, 我们可以比较图灵机与问题的数量。前者的数量是  $\aleph_0$ , 而即便是判定性问题的数量也有  $2^{\aleph_0} = 2^{\aleph_0} = \aleph_1$ 。这就是说, 图灵机到它所计算的问题的映射不可能是满射, 并且存在不可数多的不可计算问题。

另一方面, 我们也可以构造出不可计算问题的实例。

#### 3.5.1 对角线法

考虑语言  $L_0 = \{x \in \Sigma^* \mid M_x = \perp \vee M_x(x) = 0\}$ , 其中  $M_x$  表示编码为  $x$  的图灵机,  $M_x = \perp$  表示该图灵机不存在。

现在证明  $L_0$  不可能被任何图灵机判定。假设有图灵机  $M$  判定  $L_0$ ，设  $x$  是  $M$  的编码。如果  $x \in L_0$ ，根据  $L_0$  的定义有  $M(x) = 0$ ，矛盾；如果  $x \notin L_0$ ，根据  $L_0$  的定义有  $M(x) \neq 0$ ，也矛盾。

类似地，对非确定图灵机也可以构造出相应例子，因为其也能编码。

这相当于将一个无限矩阵  $M_i(j)$  的对角线翻转，从而让任何一台图灵机不可能匹配这个矩阵的任意一行。这种构造思路称为**对角线法**。

### 3.5.2 停机问题

有了上面的例子  $L_0$ ，现在可以讨论著名的停机问题：

$$\text{HALT}(M, x) = \begin{cases} 1, & \text{图灵机 } M \text{ 输入 } x \text{ 后停机;} \\ 0, & \text{otherwise.} \end{cases}$$

事实上，有  $L_0 \leq_T \text{HALT}$ 。可以构造神谕图灵机  $M^{\text{HALT}}$ ，使得它输入  $x$  后，先调用一下  $\text{HALT}(M_x, x)$  看看是否停机，如果不停机则肯定  $x \notin L_0$ ，否则接入通用图灵机模拟  $M_x(x)$  的计算即可。这样， $M^{\text{HALT}}$  能判定  $L_0$ 。

因此， $\text{HALT}$  也不可计算。

## 4 一些常见的 NPC 或 NP-Hard 问题

### 4.1 布尔可满足性问题

#### 4.1.1 基本概念

**定义 4.1** (布尔函数和表达式). 一个  $\{0, 1\}^n \rightarrow \{0, 1\}$  的函数  $f$  称为**布尔函数**。对于  $n$  个命题变项  $x_1 \sim x_n$ ，用析取运算符  $\vee$ 、合取运算符  $\wedge$ 、否定运算符  $\neg$  和命题变项  $x_i$  构成的表达式称为**布尔表达式**  $\phi$ ，它能诱导出一个布尔函数  $f_\phi$ 。称  $\phi$  是**可满足的**当且仅当存在  $x \in \{0, 1\}^n$  使得  $f_\phi(x) = 1$ ，进一步地，如果这样的  $x$  唯一则称  $\phi$  是**唯一可满足的**。 $\phi$  的大小  $|\phi|$  定义为它所包含的二元运算符（即  $\wedge$  和  $\vee$ ）的个数。

**定义 4.2** (范式). 某个命题变项  $x_i$  或其否定  $\neg x_i$  称为一个**文字**。若干个文字的合取称为**合取式**，若干个文字的析取称为**析取式**。若干个合取式的析取称为**析取范式**，若干个析取式的合取称为**合取范式**。

**定理 4.1.** 每个  $n$  元布尔函数  $f$  都对应某个大小为  $O(2^n n)$  的析取范式和合取范式。

证明. 析取范式取  $\bigvee_{x'_i \in \{x_i, \neg x_i\}, f(x'_1, x'_2, \dots, x'_n) = 1} \bigwedge_{i=1}^n x'_i$ ，合取范式取  $\bigwedge_{x'_i \in \{x_i, \neg x_i\}, f(x'_1, x'_2, \dots, x'_n) = 0} \bigvee_{i=1}^n \neg x'_i$  即可。这两者分别称为  $f$  的**主析取范式**和**主合取范式**。□

定义 4.3 (SAT 和  $k$ SAT). 语言 SAT 为所有可满足的合取范式的集合,  $k$ SAT 在此基础上还要求合取范式内的每项析取式都包含不超过  $k$  个文字。

#### 4.1.2 用合取范式描述图灵机

定理 4.2. 对于完全单带图灵机  $M$ , 对每个  $x \in \Sigma^*$  都存在一个合取范式  $\phi_{M,x}$  使得  $M(x) = 1$  当且仅当  $\phi_{M,x}$  可满足当且仅当  $\phi_{M,x}$  唯一可满足, 且  $|\phi_{M,x}| = \text{poly}(T_M(|x|))$ 。

证明.  $M(x) = 1$  等价于存在一个格局序列  $\sigma$ , 使得  $\sigma_t$  能转移到  $\sigma_{t+1}$ , 且  $\sigma_0$  是初始格局, 某个  $\sigma_T$  停机并输出 1。一个想法是, 将一个格局序列编码为一个 01 串  $a$ , 构造一个合取范式  $\phi$  判断  $a$  是否是合法的格局序列, 具体地,  $a$  合法当且仅当  $f_\phi(a) = 1$ 。这样,  $\phi$  的可满足性就和  $M(x) = 1$  等价了。另外, 实际上  $M(x)$  计算中的格局序列是确定的, 所以最多存在一个  $a$  使  $f_\phi(a) = 1$ , 因此  $\phi$  可满足又等价于唯一可满足。

先讨论如何编码格局序列。令  $m = T_M(|x|)$ , 那么格局序列的有效范围只有  $\sigma_{0 \sim m}$ , 而在  $m$  步内只能移动到  $[-m+1, m+1]$  内的位置, 所以纸带内容也只要记录  $O(m)$  大小的信息。记  $q'_t$  表示  $t$  时刻的状态转为二进制后的长度  $O(\log |Q|)$  的 01 串;  $A'_{t,i}$  表示有效范围内  $t$  时刻纸带  $i$  上的字符转为二进制, 再加上一位表示读写头是否位于此处后的长度  $O(\log |\Sigma|)$  的 01 串。注意  $|Q|$  和  $|\Sigma|$  都是常数, 因此整个编码的大小是  $O(m^2)$  的。

现在开始构造  $\phi_{M,x}$ 。总的来说, 分为三个部分  $\phi_{M,x} = \phi_{\text{start}} \wedge \phi_{\text{halt}} \wedge \phi_{\text{transit}}$ , 每一部分都是一个合取范式。

$\phi_{\text{start}}$  检查  $\sigma_0$  的编码是否是初始格局, 这是极其容易的, 只要  $O(m)$  的大小。

$\phi_{\text{halt}}$  检查是否存在某个  $q'_t$  是 H 的编码, 且对应  $A'_t$  描述的字符串是  $\cdots \perp \perp 1 \perp \perp \cdots$ , 这也很容易, 只要  $O(m^2)$  的大小。

$\phi_{\text{transit}}$  检查  $\sigma_t$  到  $\sigma_{t+1}$  的转移是否成立。检查  $q'_t$  只要看  $q'_{t-1}$  和  $t-1$  行的读写头处的字符是否能转移到  $q'_t$ , 容易写成一个合取范式。检查  $A'_{t,i}$  时, 它是否正确只与  $q'_{t-1}$  和  $A'_{t-1,i-1}, A'_{t-1,i}, A'_{t-1,i+1}$  有关, 依然容易写成一个常数大小的合取范式 (如果不易直接写成合取范式, 只要先写成任意布尔表达式, 再通过定理 4.1 转为合取范式即可, 其大小依然是常数)。

综上所述, 我们得到了一个关于  $m$  多项式大小的  $\phi_{M,x}$ 。 □

#### 4.1.3 SAT 和 3SAT 的 NP 完全性

定理 4.3.  $\text{SAT} \in \text{NPC}$ 。

证明. 易知 SAT 是 NP 的, 我们只要构造一个验证机, 输入合取范式和所有命题变项的取值, 在多项式时间内给出计算结果即可, 这是容易的。

而要证  $\text{SAT} \in \text{NP-Hard}$ , 需要让所有  $f \in \text{NP}$  都库克归约到它。设  $M$  是  $f$  的一台多项式时间验证机, 考虑将  $M(x, c) = 1$  用  $\phi_{M,(x,c)}$  描述 (这需要对定理 5.2 中的构造做一点修改:

因为验证机的时间函数只与  $|x|$  有关, 所以应该令  $m = T_M^f(|x|)$  而非  $m = T_M^f(|(x, c)|)$ 。而为了表示存在  $c$  使得  $\phi_{M(x,c)}$  可满足, 只需要将这个构造中的  $\phi_{\text{start}}$  部分中关于  $c$  的限制删掉即可。

容易确认由  $M$  和  $x$  构造合取范式的过程可以由一台多项式时间的图灵机  $M'$  完成, 那么给它加上神谕 SAT, 在构造完合取范式之后调用一次, 即可得到  $f(x)$ 。所以  $f$  能库克归约到 SAT, 即  $\text{SAT} \in \text{NPC}$ 。□

SAT 是第一个被发现的 NPC 问题, 其地位非常重要。SAT 本身的证明不依赖于某个特定问题的归约, 而其他 NPC 问题的证明基本上都直接或间接来自于 SAT 到它的库克归约。

**定理 4.4.**  $3\text{SAT} \in \text{NPC}$ 。

证明. 显然  $3\text{SAT} \in \text{NP}$ 。要证  $3\text{SAT} \in \text{NP-Hard}$ , 根据库克归约的传递性, 只要证  $\text{SAT} \leq_c 3\text{SAT}$ 。

现在的任务是将一个析取式转化为一个多项式大小的等值的合取范式, 使得其内部每个析取式的文字数量不超过 3。考虑析取式  $\phi_1, \phi_2$ , 有  $(y \vee \phi_1) \wedge (\neg y \vee \phi_2) = \phi_1 \vee \phi_2$ , 这可以将一个  $k = k_1 + k_2$  个文字的析取式拆成两个分别有  $k_1 + 1$  和  $k_2 + 1$  个文字的析取式。这样, 可以每次拆两个文字出来, 即  $x_1 \vee x_2 \vee x_3 \vee \cdots \vee x_k = (y \vee x_1 \vee x_2) \wedge (\neg y \vee x_3 \vee \cdots \vee x_k)$ , 使主式的文字数量减少 1, 拆  $k - 3$  次即可。□

#### 4.1.4 例题

**例题 4.1** (游戏<sup>4</sup>)。有一个长度为  $n$  的数组  $a$ , 每个位置  $a_i$  的值可以取的范围是  $A_i$ ,  $A_i$  的值可能是  $\{1, 2\}$ ,  $\{1, 3\}$ ,  $\{2, 3\}$  或  $\{1, 2, 3\}$ 。另有  $m$  个限制, 每个限制形如  $(x, u, y, v)$ , 表示若  $a_x = u$  则必须  $a_y = v$ 。问是否有满足条件的数组  $a$ 。

$n \leq 5 \times 10^4$ ,  $m \leq 10^5$ ,  $A_i = \{1, 2, 3\}$  的数量不超过  $d = 8$ 。

设命题变项  $x_{i,j}$  表示是否有  $a_i = j$ 。需要用合取范式描述  $m$  个限制, 以及  $a_i$  恰好取一个值。 $a_i$  至多取一个值可以通过两两不同值不能同时取描述, 至少取一个值则是一个包含  $|A_i|$  个文字的析取式。

如果不存在  $A_i = \{1, 2, 3\}$ , 那就转化为了 2SAT 问题, 否则是 3SAT, 后者没有多项式时间的算法。那么对于  $A_i = \{1, 2, 3\}$  的情况, 考虑暴力枚举  $3^d$  种可能, 每种可能跑 2SAT 即可, 时间复杂度  $O(3^d(n + m))$ 。一个优化是可以只枚举  $a_i \in \{1, 2\}$  和  $a_i = 3$  这  $2^d$  种可能, 依然可以跑 2SAT, 时间复杂度  $O(2^d(n + m))$ 。

## 4.2 一些图论问题

图相关定义此处不进行赘述。将图  $G = (V, E)$  编码为字符串时, 默认需要包含  $V$  中元素的名称信息, 这使得编码长度总是  $\Omega(|V|)$  的 (而由  $|E| \leq |V|^2$ , 编码长度自然有上界  $O(|V|^2)$ )。

<sup>4</sup>来源: NOI 2017

### 4.2.1 独立集与团问题

**定义 4.4 (IS 和 Clique).** 对于无向图  $G = (V, E)$ , 如果点集  $S \subseteq V$  内的点两两没有连边, 则称  $S$  为独立集, 如果两两有连边, 则称  $S$  为团。语言 IS 为所有  $(G, k)$  的集合使得  $G$  中有至少  $k$  个点的独立集, Clique 类似, 但将独立集改为团。

**定理 4.5.** IS, Clique  $\in$  NPC。

证明. 两者在 NP 中是明显的, 只要枚举每个点是否属于团或独立集, 对应的 01 串作为证书验证即可。

现在证明  $\text{SAT} \leq_c \text{IS}$ 。对于合取范式  $\phi = \bigwedge_{i=1}^k \bigvee_j x_{i,j}$  (其中  $x_{i,j}$  是文字), 构造图  $G = (V, E)$ , 其中  $V = \{(i, j)\}$  表示文字集合,  $E = \{((i_1, j_1), (i_2, j_2)) \mid i_1 = i_2 \vee x_{i_1, j_1} = \neg x_{i_2, j_2}\}$ 。此时,  $k$  个点的独立集一定是每个析取式内各取一个文字, 且相互不矛盾, 因此  $\phi$  可满足当且仅当  $G$  有至少  $k$  个点的独立集。

显然有  $\text{IS} \leq_c \text{Clique}$ , 因为独立集和补图的团等价。  $\square$

所以, 最大独立集问题和最大团问题是 NP-Hard 问题, 因为 IS 和 Clique 可以库克归约到它们。但它们本身不是 NP 问题, 因为不是判定性问题。

### 4.2.2 点覆盖问题

**定义 4.5 (VC).** 对于无向图  $G = (V, E)$ , 如果点集  $S \subseteq V$  使得每条边都有至少一个端点在  $S$  中, 则称  $S$  为点覆盖。语言 VC 为所有  $(G, k)$  的集合使得  $G$  中有至多  $k$  个点的点覆盖。

**定理 4.6.** VC  $\in$  NPC。

证明. 显然 VC  $\in$  NP, 只要枚举每个点是否属于点覆盖, 验证即可。

如果  $S$  是一个点覆盖, 则  $V \setminus S$  是一个独立集, 反之亦然。因此  $\text{IS} \leq_c \text{VC}$ 。  $\square$

最小点覆盖问题是 NP-Hard 问题。

### 4.2.3 图染色问题

**定义 4.6 (Color 和 kColor).** 对于无向图  $G = (V, E)$ , 一个  $V \rightarrow \{0, 1, \dots, k-1\}$  的函数  $c$  如果满足对任意  $(x, y) \in E$  都有  $c(x) \neq c(y)$ , 则称为  $G$  的一个  $k$  染色。语言 Color 为所有  $(G, k)$  的集合使得  $G$  存在  $k$  染色, kColor 为所有存在  $k$  染色的  $G$  的集合。

**定理 4.7.** Color  $\in$  NPC, 且对  $k \geq 3$  有 kColor  $\in$  NPC。



证明.  $\text{Color}, k\text{Color} \in \mathbf{NP}$  是明显的, 可以枚举可能的染色  $c$  进行验证, 即使是对于  $\text{Color}, c$  的编码长度是  $O(n \log k)$ , 这依然是输入长度的多项式。

现在证明  $\text{SAT} \leq_c 3\text{Color}$ 。对于合取范式  $\phi = \bigwedge_{i=1}^k \bigvee_j x_{i,j}$ , 考虑构造图  $G = (V, E)$  使得  $G$  存在 3 染色当且仅当  $\phi$  可满足。先构造一个三元环  $T, F, R$ , 如果存在 3 染色, 则三元环上的节点颜色一定互不相同, 不妨设  $c(F) = 0, c(T) = 1, c(R) = 2$ 。对于每个命题变项  $x_i$ , 新增两个点  $a_i, b_i$ , 表示  $x_i$  的两种取值, 连边  $(a_i, R), (b_i, R)$ , 则必有  $\{c(a_i), c(b_i)\} = \{0, 1\}$ , 表示  $x_i$  实际取颜色为 1 的那个值。

可以构造一种类似逻辑门的组件: 对于已有的两个点  $u, v$ , 新增一个三元环  $i_1, i_2, o$  ( $i_1, i_2$  可类比输入端,  $o$  可类比输出端), 连边  $(u, i_1), (v, i_2)$ , 达到的效果是如果  $c(u) = c(v)$  则必有  $c(o) = c(u)$ , 否则  $c(o)$  不受限制。基于此, 对于析取式  $\bigvee_j x_{i,j}$ , 重复使用多次该组件, 可以将  $x_{i,j}$  对应的节点们连成一个大组件, 当所有  $x_{i,j} = 0$  时输出端强制颜色为 0, 否则可以不为 0, 那么只要将这个输出端与  $F$  相连即可。这样, 我们成功构造了一个图来描述  $\phi$ 。

对于  $k \geq 3$ , 容易知道  $3\text{Color} \leq_c k\text{Color}$ , 只要新增一个大小为  $k-3$  的团并连向原来的所有点即可归约。  $3\text{Color} \leq_c \text{Color}$  更是极为明显。  $\square$

注意到图染色问题中的  $k$  是比前面列举的几个问题的  $k$  更强的存在: 当  $k$  是常数时, 图染色问题依然 **NP-Hard**, 但独立集问题之类就可以直接  $O(n^k)$  暴力了。

#### 4.2.4 团覆盖问题

定义 4.7 (CC). 对于无向图  $G = (V, E)$ , 语言 CC 为所有  $(G, k)$  的集合使得  $V$  能划分为  $k$  个团,  $k\text{CC}$  为所有  $G$  的集合使得  $V$  能划分为  $k$  个团。

定理 4.8.  $\text{CC} \in \mathbf{NPC}$ , 且对  $k \geq 3$  有  $k\text{CC} \in \mathbf{NPC}$ 。

证明. 图染色相当于独立集覆盖, 即其补图的团覆盖。  $\square$

### 4.3 集合的两种覆盖问题

#### 4.3.1 集合覆盖问题

定义 4.8 (SC). 语言 SC 为所有  $(S, k)$  的集合, 其中  $S$  是元素为有限集的有限集, 使得存在  $T \subseteq S$  满足  $|T| \leq k$  且  $\bigcup_{U \in T} U = \bigcup_{U \in S} U$ 。

定理 4.9.  $\text{SC} \in \mathbf{NPC}$ 。

证明. 显然  $\text{SC} \in \mathbf{NP}$ , 枚举  $T \subseteq S$  验证即可。

容易发现点覆盖是集合覆盖的特例, 满足  $\bigcup_{U \in S} U$  中的每个元素都恰好在两个  $U \in S$  中出现。因此  $\text{VC} \leq_K \text{SC}$ 。  $\square$

最小集合覆盖问题是 **NP-Hard** 问题。



### 4.3.2 精确覆盖问题

**定义 4.9 (EC).** 语言 EC 为所有  $S$  的集合, 其中  $S$  是元素为有限集的有限集, 使得存在  $T \subseteq S$  满足  $\sqcup_{U \in T} U = \bigcup_{U \in S} U$ , 其中  $\sqcup$  表示无交并。

**定理 4.10.**  $EC \in NPC$ 。

证明. 显然  $EC \in NP$ , 枚举  $T \subseteq S$  验证即可。

考虑证明  $SAT \leq_c EC$ 。对于合取范式  $\phi = \bigwedge_{i=1}^k \bigvee_j x_{i,j}$ , 对每个命题变项  $x_u$  新增两类集合  $T_u = \{x_u\} \cup \{(i, j) \mid x_{i,j} = \neg x_u\}$ ,  $F_u = \{x_u\} \cup \{(i, j) \mid x_{i,j} = x_u\}$ , 并且此后不会新增任何包含  $x_u$  的集合, 这就是说  $T_u$  和  $F_u$  恰好要选一个, 表示  $x_u$  的取值。再对每个文字  $x_{i,j}$  新增两类集合  $\{(i, j)\}$  和  $\{i, (i, j)\}$ 。至此, 我们可以证明  $S$  有精确覆盖当且仅当  $\phi$  可满足, 因为  $i$  能被选则至少需要一个文字  $x_{i,j}$  符合它的取值。□

### 4.4 子集和问题

**定义 4.10 (SS).** 语言 SS 为所有  $(S, s)$  的集合, 其中  $S$  是自然数的可重集,  $s$  是自然数, 使得存在  $T \subseteq S$  满足  $\sum_{x \in T} x = s$ 。

**定理 4.11.**  $SS \in NPC$ 。

证明. 显然  $SS \in NP$ , 枚举  $T \subseteq S$  验证即可。

考虑证明  $EC \leq_c SS$ 。对于有限集  $T$ , 令  $A = \bigcup_{U \in T} U$ , 记  $n = |A|, m = |T|$ 。考虑任一  $A \rightarrow \{0, 1, \dots, n-1\}$  的映射  $f$ , 令  $S = \{\sum_{x \in U} (m+1)^{f(x)} \mid U \in T\}$ ,  $s = \frac{(m+1)^n - 1}{m+1}$ , 则  $S$  存在子集和为  $s$  当且仅当  $T$  存在精确覆盖。这个构造中,  $S$  的编码长度为  $O(mn \log m)$ , 是  $T$  编码长度的多项式。□

需要注意, 这个问题涉及到了自然数。通过 DP 可以获得  $O(|S|s)$  的算法, 这并不说明它有多项式时间的算法, 因为  $s$  的编码只有  $O(\log s)$  长, 因此  $s$  可能是输入长度的指数级别。这种关于输入自然数的值域的多项式时间称为伪多项式时间。

### 4.5 01 背包问题

**定理 4.12 (01Knapsack).** 语言 01Knapsack 为所有  $(n, w_{1 \sim n}, v_{1 \sim n}, W, V)$  的集合, 其中  $w_i, v_i, W, V$  是自然数, 满足存在序列  $x_{1 \sim n} \in \{0, 1\}$  使得  $\sum_{i=1}^n x_i w_i \leq W$  且  $\sum_{i=1}^n x_i v_i \geq V$ 。

**定理 4.13.** 01Knapsack  $\in NPC$ 。

证明. 显然 01Knapsack  $\in NP$ , 枚举  $x_{1 \sim n}$  验证即可。

对于自然数的可重集  $S$  和自然数  $s$ , 令  $n = |S|$ ,  $w_i = v_i = S_i$  (其中  $S_i$  是  $S$  中的第  $i$  个元素),  $W = V = s$ , 则得到  $SS \leq_c 01Knapsack$ 。□



01 背包问题的最优化版本是 **NP-Hard** 问题。

与子集和类似，01 背包问题（的最优化版本）也有  $O(nW)$  的伪多项式时间算法。

## 5 总结

计算理论的内容博大精深，因笔者水平有限，本文所展示的只是其中的基础知识与冰山一角。

问题类 **NP** 和 **NP-Hard** 能帮助我们判断 OI 中的问题哪些是几乎不可做的。事实上，能做到这点的不仅有 **NP** 相关问题类，还有如 **FP**、**#P** 和 **#PC** 等问题类描述计数问题的复杂性，以及精细复杂性理论给出了多项式级别的复杂度下界，等等。

希望本文能起到抛砖引玉的作用，吸引更多的选手来学习和研究计算理论。

## 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢张超老师、李曙老师、史钊镭老师对我的指导。

感谢杨敏行同学、程思元同学、李白天学长与我的沟通交流。

感谢父母对我多年来的关心与支持。

## 参考文献

- [1] 徐哲安. 浅谈有限状态自动机及其应用 [J]. IOI 中国国家候选队论文集, 2021.
- [2] Arindama Singh. Elements of Computation Theory [M]. Germany: Springer London, 2009.
- [3] Michael Sipser. Introduction to the Theory of Computation [M]. India: Cengage Learning India Private Limited, 2012.
- [4] 傅育熙. 计算复杂性理论 [M]. 北京: 清华大学出版社, 2023.
- [5] Wikipedia. Nondeterministic Turing machine [OL].  
[https://en.wikipedia.org/wiki/Nondeterministic\\_Turing\\_machine](https://en.wikipedia.org/wiki/Nondeterministic_Turing_machine).
- [6] Wikipedia. Karp's 21 NP-complete problems [OL].  
[https://en.wikipedia.org/wiki/Karp%27s\\_21\\_NP-complete\\_problems](https://en.wikipedia.org/wiki/Karp%27s_21_NP-complete_problems).



# 浅谈格算法在整系数多项式分解中的应用

北京大学附属中学 谢梓涵

## 1 前言

整系数多项式分解是一个经典问题。该问题给出一个整系数本原多项式  $f = \sum_{i=0}^n a_i x^i$ ，要求将  $f$  分解为若干个  $\mathbb{Z}[x]$  下不可约多项式的乘积。本文将引入 Lenstra-Lenstra-Lovasz 格基约化算法和 [4] 中给出的多项式分解算法，并在  $O(n^{12} + n^9(\log |f|)^3)$  时间内解决整系数多项式分解问题。

## 2 约定和记号

对于一个向量  $v = (v_1, v_2, \dots, v_n)$ ，定义  $|v|$  为它的欧几里得长度。形式化地，定义：

$$|v| = \sqrt{\sum_{i=1}^n v_i^2}$$

对多项式  $f(x) = \sum_{i=0}^n a_i x^i$ 。定义  $|f|$ ：

$$|f| = \sqrt{\sum_{i=0}^n a_i^2}$$

对于两个向量  $v_1$  和  $v_2$ ，定义  $v_1 \cdot v_2$  为它们的点积。

对于矩阵  $M$ ，定义  $M_{i,j}$  表示  $M$  的第  $i$  行第  $j$  列的元素。

在第 4 节中，我们用  $p$  表示一个质数，用  $k$  表示一个正整数，用  $\mathbb{Z}/p^k\mathbb{Z}$  表示整数模  $p^k$  的环，用  $\mathbb{F}_p$  表示域  $\mathbb{Z}/p\mathbb{Z}$ 。

对于一个多项式  $g(x) = \sum_{i=0}^n a_i x^i$  其中  $\forall 0 \leq i \leq n, \text{ s.t. } a_i \in \mathbb{Z}$ ，我们定义  $(g \bmod p)(x) = \sum_{i=0}^n (a_i \bmod p) x^i$ 。

对于两个多项式  $f, g \in \mathbb{Z}[x]$ ，我们记  $g \mid f$  表示  $g$  在  $\mathbb{Z}[x]$  上整除  $f$ ；记  $g \mid f \pmod{p}$  表示  $g \bmod p$  在  $\mathbb{F}_p[x]$  上整除  $f \bmod p$ ；记  $g \mid f \pmod{p^k}$  表示  $g \bmod p^k$  在  $(\mathbb{Z}/p^k\mathbb{Z})[x]$  上整除  $f \bmod p^k$ ，并同理用  $\nmid$  表示不整除。

对于集合  $S$ ，用  $\#S$  表示  $S$  中元素个数。

### 3 LLL 格基约化算法

**定义 3.1 (格).** 对于  $n$  维实数向量空间  $\mathbb{R}^n$  的一组基  $B = \{b_1, b_2, \dots, b_n\}$ , 定义格 (Lattice)  $\mathcal{L}$  是  $B$  的所有整系数线性组合构成的集合:

$$\mathcal{L} = \left\{ \sum_{i=1}^n z_i b_i : z_i \in \mathbb{Z}, b_i \in B \right\}$$

我们称  $B$  构成  $\mathcal{L}$  的一组基底, 或  $B$  张成  $\mathcal{L}$ 。

**定义 3.2 (格的行列式).** 对于对于  $n$  维实数向量空间  $\mathbb{R}^n$  的一组基  $B = \{b_1, b_2, \dots, b_n\}$  张成的格  $\mathcal{L}$ , 定义  $\det \mathcal{L} = |\det(b_1, b_2, \dots, b_n)|$ , 其中所有  $b_i$  被写作列向量。该值是一个与  $B$  的选取无关的值, 证明见 [3, Sect. 1.2]。

**定义 3.3 (格的最短向量).** 对于格  $\mathcal{L}$ , 定义其最短向量  $\lambda(\mathcal{L})$  是  $\mathcal{L}$  中欧几里得长度最短的向量。

对于  $\mathcal{L}$  求  $\lambda(\mathcal{L})$  是经典的最短向量问题 (SVP), 目前已被证明是 NP-Hard 问题。LLL 算法能够在关于格的维数  $n$  的多项式时间复杂度内求出一个对于整点格的 SVP 的答案的估计。具体来说, 对任意常数  $c > \sqrt{4/3}$ , LLL 算法能够求出向量  $v$  满足  $|v| \leq c^{n-1} \lambda(\mathcal{L})$ 。

#### 3.1 $n = 2$ 的情形: 格基约化

**定义 3.4 (二维格基底的不可约性).** 如果一组二维的基底  $(b_1, b_2)$  满足:

1.  $|b_1| \leq |b_2|$ ;
2.  $b_2$  关于  $b_1$  的正交投影系数  $\mu_{1,2} = \frac{b_1 \cdot b_2}{|b_1|^2}$  满足  $|\mu_{1,2}| \leq 1/2$ 。

则称该基底是不可约的 (reduced)。

**定理 3.1.** 对于二维格  $\mathcal{L}$  的一组不可约的基底  $(b_1, b_2)$ , 满足  $b_1$  是  $\mathcal{L}$  的最短向量。

证明. 设  $v$  是  $\mathcal{L}$  中任意向量, 则存在不同时为 0 的整数  $z_1, z_2$  满足  $v = z_1 b_1 + z_2 b_2$ 。于是有:

$$\begin{aligned} |v|^2 &= z_1^2 |b_1|^2 + 2z_1 z_2 b_1 \cdot b_2 + z_2^2 |b_2|^2 \\ &\geq z_1^2 |b_1|^2 - 2|z_1||z_2||b_1 \cdot b_2| + z_2^2 |b_2|^2 \\ &= z_1^2 |b_1|^2 - 2\mu_{1,2}|z_1||z_2||b_1|^2 + z_2^2 |b_2|^2 \\ &\geq z_1^2 |b_1|^2 - |z_1||z_2||b_1|^2 + z_2^2 |b_2|^2 \quad (\text{由定义, } |\mu_{1,2}| \leq \frac{1}{2}) \\ &\geq z_1^2 |b_1|^2 - |z_1||z_2||b_1|^2 + z_2^2 |b_1|^2 \quad (\text{由定义, } |b_1| \leq |b_2|) \\ &= (|z_1| - |z_2|)^2 |b_1|^2 \end{aligned}$$

由于  $z_1, z_2 \in \mathbb{Z}$  且不同时为 0, 因此若  $|z_1| \neq |z_2|$ , 则  $(|z_1| - |z_2|)^2 \geq 1$ ; 否则  $|z_1| = |z_2| \neq 0$ , 则  $|z_1 z_2| \geq 1$ , 从而有  $(|z_1| - |z_2|)^2 + |z_1 z_2| \geq 1$ , 从而  $|v|^2 \geq |b_1|^2$ , 即  $b_1$  是  $\mathcal{L}$  的最短向量, 命题得证。□

由上述定理, 对于  $n = 2$  的格, 我们只要求出一组不可约的基底, 就可以求出 SVP 的解。高斯在 19 世纪初提出了一种求解二维 SVP 的算法, 见下文 **Algorithm 1**。

---

**Algorithm 1** 高斯格基约化算法

---

**Input:**  $b_1, b_2$

**Output:**  $b_1, b_2$  张成的格  $\mathcal{L}$  的一组不可约的基底  $b'_1, b'_2$

```

1: repeat
2: if $|b_1| > |b_2|$ then
3: swap b_1 and b_2
4: end if
5: $r \leftarrow \mu_{1,2}$ rounded to the nearest integer
6: $b_2 \leftarrow b_2 - rb_1$
7: until $|b_1| \leq |b_2|$
8: return b_1, b_2

```

---

### 3.2 $n \geq 3$ 的情形: Gram-Schmidt 正交化

我们在  $n = 2$  的情形中可以注意到向量  $b_2^* = b_2 - \mu_{1,2}b_1$  满足  $b_2^* \perp b_1$ 。但是如果  $\mu_{1,2} \notin \mathbb{Z}$ , 那么我们得到的  $b_2^*$  并不在格  $\mathcal{L}$  中。我们只有将  $b_2$  改变为  $b_2 + zb_1$  ( $z \in \mathbb{Z}$ ) 才不会改变  $\mathcal{L}$  的结构。据此, 我们发现当  $|\mu_{1,2}| \leq 1/2$  时,  $b_2$  是在所有  $b_2 + zb_1$  ( $z \in \mathbb{Z}$ ) 里面最接近  $b_2^*$  的。

我们尝试在  $n$  更大的情况下延续该思路: 使得基底中的向量尽量接近正交关系。为了做到这一点, 我们引入 **Gram-Schmidt 正交化**。

**定理 3.2** (Gram-Schmidt 正交化). 对于  $\mathbb{R}^n$  的一组基底  $\{b_1, b_2, \dots, b_n\}$ , 定义:

$$\begin{cases} b_1^* = b_1 \\ b_k^* = b_k - \sum_{i < k} \mu_{i,k} b_i^* \quad (k > 1, \mu_{i,k} = \frac{b_k \cdot b_i^*}{|b_i^*|^2}) \end{cases}$$

则  $\{b_1^*, b_2^*, \dots, b_n^*\}$  是  $\mathbb{R}^n$  的一组正交基底。

证明. 考虑归纳, 当  $n = 2$  时:

$$\begin{aligned} b_1^* \cdot b_2^* &= b_1 \cdot (b_2 - \mu_{1,2}b_1) \\ &= b_1 \cdot (b_2 - \frac{b_1 \cdot b_2}{|b_1|^2} b_1) \\ &= b_1 \cdot b_2 - \frac{b_1 \cdot b_2}{|b_1|^2} |b_1|^2 \\ &= 0 \end{aligned}$$

现假设  $n = k$  时命题成立。 $n = k + 1$  时, 由归纳假设,  $\forall 1 \leq i < j \leq k, \text{s.t. } \mathbf{b}_i^* \perp \mathbf{b}_j^*$ 。同时, 对任意  $1 \leq j \leq k$ :

$$\begin{aligned} \mathbf{b}_{k+1}^* \cdot \mathbf{b}_j^* &= \left( \mathbf{b}_{k+1} - \sum_{i=1}^k \mu_{i,k+1} \mathbf{b}_i^* \right) \cdot \mathbf{b}_j^* \\ &= \mathbf{b}_{k+1} \cdot \mathbf{b}_j^* - \sum_{i=1}^k \mu_{i,k+1} \mathbf{b}_i^* \cdot \mathbf{b}_j^* \\ &= \mathbf{b}_{k+1} \cdot \mathbf{b}_j^* - \mu_{j,k+1} |\mathbf{b}_j^*|^2 \quad (\text{由归纳假设, } \forall 1 \leq i, j \leq k, i \neq j, \text{s.t. } \mathbf{b}_i^* \cdot \mathbf{b}_j^* = 0) \\ &= \mathbf{b}_{k+1} \cdot \mathbf{b}_j^* - \frac{\mathbf{b}_{k+1} \cdot \mathbf{b}_j^*}{|\mathbf{b}_j^*|^2} |\mathbf{b}_j^*|^2 \\ &= 0 \end{aligned}$$

从而  $\forall 1 \leq i < j \leq k + 1, \text{s.t. } \mathbf{b}_i \cdot \mathbf{b}_j = 0$ , 对  $n$  归纳即得证。  $\square$

至此, 我们得到了一个在任意空间里面正交化一组基底的方式。但是和  $n = 2$  的情况一样, 使用这个正交化方法得到的基底很可能不出现在原基底张成的格内, 在这些情况下只能获得一个接近正交基底的结果。因此, 类似  $n = 2$  的情况, 我们同样需要定义一个新的不可约标准来告诉我们什么样的基底是“足够接近正交基底”的。

**定义 3.5** ( $n$  维格基底的 LLL 不可约性). 设  $y$  是一满足  $1/4 < y < 1$  的常数,  $B = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n\}$  是格  $\mathcal{L}$  的一组基底,  $B^* = \{\mathbf{b}_1^*, \mathbf{b}_2^*, \dots, \mathbf{b}_n^*\}$  是该基底通过 Gram-Schmidt 正交化生成的正交基底。记  $\mu_{i,j} = \frac{\mathbf{b}_j \cdot \mathbf{b}_i^*}{|\mathbf{b}_i^*|^2}$ 。如果  $y, B, B^*$  满足如下条件:

1.  $\forall 1 \leq i < j \leq n, \text{s.t. } |\mu_{i,j}| \leq 1/2$
2.  $\forall 2 \leq i \leq n, \text{s.t. } |\mathbf{b}_i^* + \mu_{i-1,i} \mathbf{b}_{i-1}^*|^2 \geq y |\mathbf{b}_{i-1}^*|^2$

则称格  $\mathcal{L}$  的基底  $B$  是 LLL 不可约的。

**定理 3.3.** 设  $n$  维格  $\mathcal{L}$  的基底  $B = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n\}$  是 LLL 不可约的, 则有:

$$\forall 1 \leq i \leq j \leq n, \text{s.t. } |\mathbf{b}_i|^2 \leq \left( \frac{4}{4y-1} \right)^{i-1} |\mathbf{b}_j^*|^2$$

证明. 由定义, 有  $|\mathbf{b}_i^* + \mu_{i-1,i} \mathbf{b}_{i-1}^*|^2 \geq y |\mathbf{b}_{i-1}^*|^2$ 。展开左侧并移项可得  $|\mathbf{b}_i^*|^2 \geq (y - \mu_{i-1,i}^2) |\mathbf{b}_{i-1}^*|^2$ 。又由  $|\mu_{i-1,i}| \leq 1/2$  可得:

$$\forall 2 \leq i \leq n, \text{s.t. } |\mathbf{b}_i^*|^2 \geq \left( y - \frac{1}{4} \right) |\mathbf{b}_{i-1}^*|^2$$

归纳即得:

$$\forall 1 \leq i \leq j \leq n, \text{s.t. } |\mathbf{b}_i|^2 \leq \left( \frac{4}{4y-1} \right)^{j-i} |\mathbf{b}_j^*|^2$$



由 Gram-Schmidt 正交化, 对所有  $1 \leq i \leq n$  有:

$$\mathbf{b}_i = \mathbf{b}_i^* + \sum_{j=1}^{i-1} \mu_{j,i} \mathbf{b}_j^*$$

对该式两侧平方, 由于  $\mathbf{b}_j^*$  两两垂直, 所以对任意  $x \neq y$ , 有  $\mathbf{b}_x^* \cdot \mathbf{b}_y^* = 0$ 。得到:

$$\begin{aligned} |\mathbf{b}_i|^2 &= |\mathbf{b}_i^*|^2 + \sum_{j=1}^{i-1} \mu_{j,i}^2 |\mathbf{b}_j^*|^2 \\ &\leq |\mathbf{b}_i^*|^2 + \sum_{j=1}^{i-1} \frac{1}{4} \left( \frac{4}{4y-1} \right)^{i-j} |\mathbf{b}_i^*|^2 \\ &= \left( 1 + \frac{1}{4y-1} \sum_{j=0}^{i-2} \left( \frac{4}{4y-1} \right)^j \right) \cdot |\mathbf{b}_i^*|^2 \\ &= \left( 1 + \frac{1}{5-4y} \left( \left( \frac{4}{4y-1} \right)^{i-1} - 1 \right) \right) \cdot |\mathbf{b}_i^*|^2 \quad (\text{等比数列求和公式}) \\ &\leq \left( \frac{4}{4y-1} \right)^{i-1} |\mathbf{b}_i^*|^2 \quad \left( \frac{1}{4} < y < 1 \Rightarrow \frac{1}{5-4y} < 1 \right) \end{aligned}$$

从而, 对所有  $1 \leq i \leq j \leq n$ , 有:

$$|\mathbf{b}_i|^2 \leq \left( \frac{4}{4y-1} \right)^{i-1} |\mathbf{b}_i^*|^2 \leq \left( \frac{4}{4y-1} \right)^{i-1} \left( \frac{4}{4y-1} \right)^{j-i} |\mathbf{b}_j^*|^2 = \left( \frac{4}{4y-1} \right)^{j-1} |\mathbf{b}_j^*|^2$$

即得证。 □

**定理 3.4.** 设  $n$  维格  $\mathcal{L}$  的基底  $B = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n\}$  是 LLL 不可约的, 则对任意一组线性无关的向量  $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_t \in \mathcal{L}$  且  $\forall 1 \leq j \leq t, \mathbf{v}_j \neq \mathbf{0}$ , 则对所有  $1 \leq j \leq t$  有:

$$|\mathbf{b}_j|^2 \leq \left( \frac{4}{4y-1} \right)^{n-1} \max_{1 \leq l \leq t} |\mathbf{v}_l|^2$$

证明. 设  $\mathbf{v}_j = \sum_{i=1}^n r_{i,j} \mathbf{b}_i = \sum_{i=1}^n r'_{i,j} \mathbf{b}_i^*$ , 其中  $\forall 1 \leq i \leq n, 1 \leq j \leq t, \text{s.t. } r_{i,j} \in \mathbb{Z}, r'_{i,j} \in \mathbb{R}$ 。由  $\forall 1 \leq j \leq t, \mathbf{v}_j \neq \mathbf{0}$  可得对  $1 \leq j \leq t$ ,  $r_{i,j}$  不全为 0, 设  $i_M(j)$  是最大的满足  $r_{i,j} \neq 0$  的下标  $i$ 。由 Gram-Schmidt 正交化:

$$\sum_{i=1}^n r_{i,j} \mathbf{b}_i = \sum_{i=1}^n r_{i,j} \left( \mathbf{b}_i^* + \sum_{l=1}^{i-1} \mu_{l,i} \mathbf{b}_l^* \right) = \sum_{l=1}^{n-1} \left( r_{l,j} + \sum_{i=l+1}^n r_{i,j} \mu_{l,i} \right) \mathbf{b}_l^*$$

由此可得对  $1 \leq j \leq t$ ,  $r_{i_M(j),j} = r'_{i_M(j),j}$ 。又由  $r_{i_M(j),j} \neq 0, r_{i_M(j),j} \in \mathbb{Z}$ , 可得  $|r'_{i_M(j),j}| \geq 1$ 。从而有:

$$|\mathbf{v}_j|^2 = \sum_{i=1}^n r_{i,j}^2 |\mathbf{b}_i^*|^2 \geq r_{i_M(j),j}^2 |\mathbf{b}_{i_M(j)}^*|^2 \geq |\mathbf{b}_{i_M(j)}^*|^2$$

我们将  $v$  重新排序使得  $i_M(1) \leq i_M(2) \leq \cdots \leq i_M(t)$ 。由  $v_{1,2,\dots,t}$  线性无关可得  $\forall 1 \leq j \leq t, \text{ s.t. } j \leq i(j)$ 。又由定理 3.3, 得:

$$|b_j|^2 \leq \left(\frac{4}{4y-1}\right)^{i_M(j)-1} |b_{i_M(j)}^*|^2 \leq \left(\frac{4}{4y-1}\right)^{n-1} |b_{i_M(j)}^*|^2 \leq \left(\frac{4}{4y-1}\right)^{n-1} |v_j|^2$$

即得证。  $\square$

注意到如果我们在定理 3.4 里面取  $t = 1$ , 即可得到:

$$\forall v \in \mathcal{L}, v \neq 0, \text{ s.t. } |b_1|^2 \leq \left(\frac{4}{4y-1}\right)^{n-1} |v|^2$$

因此, 对于格  $\mathcal{L}$ , 我们只需要给出一组 LLL 不可约的基底, 就可以完成本节开头处提到的估计。LLL 算法即为一种将任意基底约化为 LLL 不可约基底的算法。

该算法的主要思路为在第一个不满足 LLL 不可约的定义的地方通过交换相邻向量进行调整。具体来说, 该算法设定一个位置  $k$ , 表示:

1.  $\forall 1 \leq i < j < k, \text{ s.t. } |\mu_{i,j}| \leq 1/2$
2.  $\forall 2 \leq i < k, \text{ s.t. } |b_i^* + \mu_{i-1,i} b_{i-1}^*|^2 \geq y |b_{i-1}^*|^2$

初始  $k = 2$ , 此时不保证任何条件。接下来不断进行迭代, 直到  $k = n + 1$  也就是得到一个完整的 LLL 不可约基底。在一次迭代中, 算法首先通过调整使得  $|\mu_{k-1,k}| \leq 1/2$ , 然后分两种情况进行讨论。

情况 1.  $k \geq 2$  同时  $|b_k^* + \mu_{k-1,k} b_{k-1}^*|^2 < y |b_{k-1}^*|^2$ , 此时我们交换  $b_{k-1}$  和  $b_k$ , 并对应更新受影响的  $b^*$  和  $\mu$ 。完成更新后, 将  $k$  减小 1。

情况 2.  $k = 1$  或者  $|b_k^* + \mu_{k-1,k} b_{k-1}^*|^2 \geq y |b_{k-1}^*|^2$ , 此时我们首先对  $j$  从  $k-1$  到 1 调整, 使得  $|\mu_{j,k}| \leq 1/2$ 。注意到当我们用  $b_j$  调整  $b_k$  时, 对任意  $j < l < k$ ,  $\mu_{l,k}$  均不受到影响, 因此可以调整使得对所有  $j < k$ , 有  $|\mu_{j,k}| \leq 1/2$ 。调整完成后, 将  $k$  增加 1。

实现细节上, 在情况 1 交换向量  $b_{k-1}$  和  $b_k$  时, 我们设交换完成后  $b_i, b_i^*, \mu_{i,j}$  分别变为  $c_i, c_i^*, \nu_{i,j}$ , 那么我们首先有  $c_{k-1} = b_k, c_k = b_{k-1}$ 。由 Gram-Schmidt 正交化可得:

$$\begin{aligned} c_{k-1}^* &= b_k - \sum_{i=0}^{k-2} \mu_{i,k} b_i^* \\ &= b_k^* + \sum_{i=0}^{k-1} \mu_{i,k} b_i^* - \sum_{i=0}^{k-2} \mu_{i,k} b_i^* \\ &= b_k^* + \mu_{k-1,k} b_{k-1}^* \end{aligned}$$

$$\begin{aligned}
 v_{k-1,k} &= \frac{c_k \cdot c_{k-1}^*}{|c_{k-1}^*|^2} \\
 &= \frac{b_{k-1} \cdot (b_k^* + \mu_{k-1,k} b_{k-1}^*)}{|c_{k-1}^*|^2} \\
 &= \frac{|b_{k-1}^*|^2}{|c_{k-1}^*|^2} \mu_{k-1,k} \quad (b_{k-1} \cdot b_k^* = 0, b_{k-1} \cdot b_{k-1}^* = |b_{k-1}^*|^2) \\
 c_k^* &= b_{k-1}^* - v_{k-1,k} c_{k-1}^*
 \end{aligned}$$

对于  $i < k-1$ , 由  $\mu$  的定义不难得到  $v_{i,k-1} = \mu_{i,k}$ ,  $v_{i,k} = \mu_{i,k-1}$ 。对于  $i > k$  可直接将  $c_{k-1}^*$  和  $c_k^*$  代入  $\mu$  的定义式得到:

$$\begin{aligned}
 v_{k-1,i} &= \frac{b_i \cdot c_{k-1}^*}{|c_{k-1}^*|^2} = \frac{|b_i^*|^2}{|c_{k-1}^*|^2} \mu_{k,i} + \frac{|b_{k-1}^*|^2}{|c_{k-1}^*|^2} \mu_{k-1,k} \mu_{k-1,i} \\
 v_{k,i} &= \frac{b_i \cdot c_k^*}{|c_k^*|^2} \\
 &= \frac{b_i \cdot (b_{k-1}^* - v_{k-1,k} b_k^* - \mu_{k-1,k} v_{k-1,k} b_{k-1}^*)}{|c_k^*|^2} \\
 &= \frac{|b_{k-1}^*|^2}{|c_k^*|^2} (1 - \mu_{k-1,k} v_{k-1,k}) \mu_{k-1,i} - \frac{|b_k^*|^2}{|c_k^*|^2} v_{k-1,k} \mu_{k,i}
 \end{aligned}$$

调整  $\mu_{i,j}$  时, 我们采用与上一部分相同的记号。取  $r$  为最接近  $\mu_{i,j}$  的整数, 则有  $c_j = b_j - r b_i$ ,  $v_{i,j} = \mu_{i,j} - r$ 。显然所有的  $b^*$  都不受影响。对于  $i < l < j$ ,  $\mu_{l,j}$  也不受影响。因此剩下的受影响的只有所有满足  $l < i$  的  $\mu_{l,j}$ 。代入定义式可得:

$$v_{l,j} = \frac{(b_j - r b_i) \cdot b_l^*}{|b_l^*|^2} = \mu_{l,j} - r \mu_{l,i}.$$

这里我们注意到我们在算法初始化完成之后的迭代过程中并不需要保存所有的  $b_i^*$ , 我们只需要保存所有的  $b_i, |b_i^*|^2, \mu_{i,j}$  就可以计算出所有的  $c_i, |c_i^*|^2$  和  $v_{i,j}$ 。

完整算法的形式化的描述可参考下文 **Algorithm 2**。我们在其中用  $B_i$  表示  $|b_i^*|^2$ 。

---

**Algorithm 2** Lenstra-Lenstra-Lovasz 格基约化算法

---

**Input:**  $n \geq 2$ ,  $n$  维格  $\mathcal{L} \subset \mathbb{Z}^n$  的一组基底  $b_{1,2,\dots,n}$ ,  $1/4 < y < 1$

**Output:**  $\mathcal{L}$  的一组不可约的基底  $b'_{1,2,\dots,n}$

- 1: **for**  $i = 1, 2, \dots, n$  **do**
  - 2:    $b_i^* \leftarrow b_i$
  - 3:   **for**  $j = 1, 2, \dots, i-1$  **do**
  - 4:      $\mu_{j,i} \leftarrow (b_j^* \cdot b_i) / B_j^2$
  - 5:      $b_i^* \leftarrow b_i^* - b_j^* \mu_{j,i}$
  - 6:   **end for**
  - 7: **end for**
-

```
8: $k \leftarrow 2$
9: while $k \leq n$ do
10: if $|\mu_{k-1,k}| > 1/2$ then
11: $r \leftarrow \mu_{k-1,k}$ rounded to the nearest integer
12: for $j = 1, 2, \dots, k-2$ do
13: $\mu_{j,k} \leftarrow \mu_{j,k} - r\mu_{j,k-1}$
14: end for
15: $\mu_{k-1,k} \leftarrow \mu_{k-1,k} - r$
16: $b_k \leftarrow b_k - rb_{k-1}$
17: end if
18: if $k \geq 2$ and $|b_k^* + \mu_{k-1,k}b_{k-1}^*|^2 < y|b_{k-1}^*|^2$ then
19: $C_{k-1} \leftarrow B_k + \mu_{k-1,k}^2 B_{k-1}^2$
20: $v_{k-1,k} \leftarrow B_{k-1}\mu_{k-1,k}/C_{k-1}$
21: $C_k \leftarrow (1 - \mu_{k-1,k}v_{k-1,k})^2 B_{k-1} + v_{k-1,k}^2 B_k$
22: for $i = 1, 2, \dots, k-2$ do
23: $v_{i,k-1} \leftarrow \mu_{i,k}$
24: $v_{i,k} \leftarrow \mu_{i,k-1}$
25: end for
26: for $i = k+1, k+2, \dots, n$ do
27: $v_{k-1,i} \leftarrow (B_k\mu_{k,i} + B_{k-1}\mu_{k-1,k}\mu_{k-1,i})/C_{k-1}$
28: $v_{k,i} \leftarrow ((1 - \mu_{k-1,k}v_{k-1,k})B_{k-1}\mu_{k-1,i} - B_kv_{k-1,k}\mu_{k,i})/C_k$
29: end for
30: swap b_{k-1} and b_k
31: $B_{k-1} \leftarrow C_{k-1}, B_k \leftarrow C_k$
32: for $1 \leq i < j \leq n, \{i, j\} \cap \{k-1, k\} \neq \emptyset$ do
33: $\mu_{i,j} \leftarrow v_{i,j}$
34: end for
35: $k \leftarrow k-1$
36: else
37: for $j = k-2, k-3, \dots, 1$ do
38: if $|\mu_{j,k}| > 1/2$ then
39: $r \leftarrow \mu_{j,k}$ rounded to the nearest integer
40: for $l = 1, 2, \dots, j-1$ do
41: $\mu_{l,k} \leftarrow \mu_{l,k} - r\mu_{l,j}$
42: end for
43: $\mu_{j,k} \leftarrow \mu_{j,k} - r$
44: $b_k \leftarrow b_k - rb_j$
```

```

45: end if
46: end for
47: $k \leftarrow k + 1$
48: end if
49: end while
50: return $b_{1,2,\dots,n}$

```

显然, 该算法终止时一定会求出一组 LLL 不可约的基底。但是该算法的时间复杂度, 甚至于该算法是否一定会结束都不是很显而易见。

### 3.3 LLL 算法的时间复杂度分析

为了证明该算法的时间复杂度, 我们对  $0 \leq i \leq n$  引入势能量  $d_i$  和总势能量  $D$ , 如下:

**定义 3.6.** 定义  $d_0 = 1$ 。设  $B = \{b_1, b_2, \dots, b_n\}$  是  $\mathbb{R}^n$  的一组基底, 对  $1 \leq i \leq n$ , 定义矩阵  $M_i$ :

$$M_i = \begin{bmatrix} b_1 \cdot b_1 & b_1 \cdot b_2 & b_1 \cdot b_3 & \cdots & b_1 \cdot b_i \\ b_2 \cdot b_1 & b_2 \cdot b_2 & b_2 \cdot b_3 & \cdots & b_2 \cdot b_i \\ b_3 \cdot b_1 & b_3 \cdot b_2 & b_3 \cdot b_3 & \cdots & b_3 \cdot b_i \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_i \cdot b_1 & b_i \cdot b_2 & b_i \cdot b_3 & \cdots & b_i \cdot b_i \end{bmatrix}$$

定义  $d_i = \det M_i$ , 总势能量  $D = \prod_{i=0}^n d_i$ 。

**定理 3.5.** 对所有  $1 \leq i \leq n$ , 有:

$$d_i = \prod_{j=1}^i |b_j^*|^2$$

证明. 注意到  $b_1 = b_1^*$ 。对  $M_i$  依次进行如下初等变换:

1. 将  $M_i$  的第 2 行整体减去第 1 行整体乘  $\mu_{1,2}$ ;
2. 将  $M_i$  的第 3 行整体减去第 1 行整体乘  $\mu_{1,3}$  和第 2 行整体乘  $\mu_{2,3}$ ;
3. 以此类推, 直到将  $M_i$  的第  $i$  行整体减去第 1 行整体乘  $\mu_{1,i}$ 、第 2 行整体乘  $\mu_{2,i}$ 、第 3 行整体乘  $\mu_{3,i}$ 、……、第  $i-1$  行整体乘  $\mu_{i-1,i}$ 。

由 Gram-Schmidt 正交化, 该过程之后矩阵  $M_i$  将变为:

$$M'_i = \begin{bmatrix} b_1^* \cdot b_1 & b_1^* \cdot b_2 & b_1^* \cdot b_3 & \cdots & b_1^* \cdot b_i \\ b_2^* \cdot b_1 & b_2^* \cdot b_2 & b_2^* \cdot b_3 & \cdots & b_2^* \cdot b_i \\ b_3^* \cdot b_1 & b_3^* \cdot b_2 & b_3^* \cdot b_3 & \cdots & b_3^* \cdot b_i \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_i^* \cdot b_1 & b_i^* \cdot b_2 & b_i^* \cdot b_3 & \cdots & b_i^* \cdot b_i \end{bmatrix}$$

不难证明对任意  $1 \leq i < j \leq n$ , 有  $b_i \cdot b_j^* = 0$ , 因此  $M'_i$  是满足  $\det M_i = \det M'_i$  的上三角矩阵:

$$M'_i = \begin{bmatrix} b_1^* \cdot b_1 & b_1^* \cdot b_2 & b_1^* \cdot b_3 & \cdots & b_1^* \cdot b_i \\ & b_2^* \cdot b_2 & b_2^* \cdot b_3 & \cdots & b_2^* \cdot b_i \\ & & b_3^* \cdot b_3 & \cdots & b_3^* \cdot b_i \\ & & & \ddots & \vdots \\ & & & & b_i^* \cdot b_i \end{bmatrix}$$

考虑其转置  $M_i'^T$ , 如下:

$$M_i'^T = \begin{bmatrix} b_1 \cdot b_1^* & & & & \\ b_2 \cdot b_1^* & b_2 \cdot b_2^* & & & \\ b_3 \cdot b_1^* & b_3 \cdot b_2^* & b_3 \cdot b_3^* & & \\ \vdots & \vdots & \vdots & \ddots & \\ b_i \cdot b_1^* & b_i \cdot b_2^* & b_i \cdot b_3^* & \cdots & b_i \cdot b_i^* \end{bmatrix}$$

对其执行同样的一系列初等变换即得到:

$$M_i'' = \begin{bmatrix} b_1^* \cdot b_1^* & & & & \\ & b_2^* \cdot b_2^* & & & \\ & & b_3^* \cdot b_3^* & & \\ & & & \ddots & \\ & & & & b_i^* \cdot b_i^* \end{bmatrix}$$

于是有:

$$d_i = \det M_i = \det M'_i = \det M_i'^T = \det M_i'' = \prod_{j=1}^i |b_j^*|^2$$

即得证。  $\square$

**定理 3.6** (LLL 算法的迭代次数). 设 LLL 算法的输入为一个  $n$  维整点格  $\mathcal{L} \subset \mathbb{Z}^n$  的基底  $B = \{b_1, b_2, \dots, b_n\}$ . 设  $V \in \mathbb{R}$  满足对任意  $1 \leq i \leq n$ , 有  $|b_i^*|^2 \leq V$ , 则 LLL 算法总共迭代  $O(n^2 \log V)$  次。

证明. 首先考虑使得算法中的  $k$  减少 1 的迭代次数。

将  $k$  减少 1 时, 新的  $b_{k-1}$  变为原来的  $b_k^* + \mu_{k-1,k} b_{k-1}^*$ , 并且满足  $|b_k^* + \mu_{k-1,k} b_{k-1}^*| < y|b_{k-1}|$ . 由  $d_i = \prod_{j=1}^i |b_j^*|^2$ , 可得在  $k$  减少 1 之后  $d_{k-1}$  至少减少到了原来的  $y$  倍。

对于  $i < k-1$ , 显然  $d_i$  不变. 对于  $i \geq k$ , 交换  $b_{k-1}$  和  $b_k$  对  $M_i$  的影响是交换相邻两行再交换相邻两列, 于是  $d_i = \det M_i$  不变. 因此在一次  $k$  减少 1 的迭代之后,  $D$  一定至少减少到原来的  $y$  倍。

考虑到初始状态下,  $\forall 1 \leq i \leq n, \text{ s.t. } |b_i^*|^2 = O(V)$ , 因此  $\forall 1 \leq i \leq n, \text{ s.t. } d_i = O(V^i)$ , 从而  $D = O(V^{\frac{n(n+1)}{2}})$ 。



算法流程中, 所有  $b_i^* \neq 0$ , 所以所有  $d_i > 0$ 。又由于  $\mathcal{L} \subset \mathbb{Z}^n$ , 因此对于任意  $1 \leq i \leq n$ ,  $M_i$  中的所有数都是整数, 因此所有  $d_i$  都是整数, 从而  $D$  是整数, 因此有  $D \geq 1$ 。因此使得  $k$  减少 1 的迭代只会进行  $O(\log D) = O(n^2 \log V)$  次。

由于算法开始时  $k = 2$ , 结束时  $k = n + 1$ , 因此使  $k$  增加 1 的迭代次数只会比使  $k$  减少 1 的迭代次数多  $O(n)$  次, 从而总迭代次数为  $O(n^2 \log V)$ , 即得证。□

考虑到初始化需要  $O(n^3)$  次代数运算, 每次情况 1 的迭代需要  $O(n)$  次代数运算, 每一次情况 2 的迭代都要进行  $O(n^2)$  次代数运算, 因此我们的总代数运算次数就是  $O(n^4 \log V)$ 。但是我们的算法中涉及到实数运算和大数运算, 因此我们的总时间复杂度还需要考虑到代数运算的复杂度, 也就是实数的精度和所有数的上界。

**定理 3.7.** 设 LLL 算法的输入满足与定理 3.6 中相同的条件, 则 LLL 算法过程中涉及到的所有数都是有理数, 且都能够用分子、分母均为  $O(n \log V)$  位二进制整数的分数表示。

证明. 首先我们分析分母。对于  $b_i$ , 显然其所有位的值都保持为整数。对于  $|b_i^*|^2$ , 有  $d_{i-1}|b_i^*|^2 = d_i \in \mathbb{Z}$ 。对于  $\mu_{i,j}$ , 我们首先证明  $d_{i-1}b_i^* \in \mathbb{Z}^n$ , 然后即可得到  $d_i\mu_{i,j} = d_{i-1}b_i^* \cdot b_j \in \mathbb{Z}$ 。该式对  $i = 1, 2$  显然成立, 接下来默认  $i \geq 3$ 。

设  $\lambda_{i,j} \in \mathbb{R}$  满足:

$$b_i^* = b_i - \sum_{j=1}^{i-1} \lambda_{i,j} b_j$$

则对  $1 \leq j < i$ ,  $\lambda_{i,j}$  是如下线性方程组的解:

$$\begin{cases} \sum_{j=1}^{i-1} \lambda_{i,j} b_j \cdot b_1 = b_i \cdot b_1 \\ \sum_{j=1}^{i-1} \lambda_{i,j} b_j \cdot b_2 = b_i \cdot b_2 \\ \dots \\ \sum_{j=1}^{i-1} \lambda_{i,j} b_j \cdot b_{i-1} = b_i \cdot b_{i-1} \end{cases}$$

不难发现该方程组的系数矩阵是  $M_{i-1}$ 。记  $A_{i-1,j,k}$  为元素  $M_{i-1,j,k}$  的代数余子式, 那么由代数余子式的定义, 我们对任意  $1 \leq l < i$  有:

$$\sum_{j=1}^{i-1} \frac{A_{i-1,l,j}}{d_{i-1}} b_j \cdot b_l = 1$$

对任意  $1 \leq l, l' < i, l \neq l'$  有:

$$\sum_{j=1}^{i-1} \frac{A_{i-1,l',j}}{d_{i-1}} b_j \cdot b_l = 0$$

又由克拉默法则,  $d_{i-1} \neq 0$ , 于是该方程组有唯一解, 据此我们可以构造出该方程组的解。对  $1 \leq j < i$ , 有:

$$\lambda_{i,j} = \frac{1}{d_{i-1}} \sum_{l=1}^{i-1} A_{i-1,l,j} b_i \cdot b_l$$

由于  $M_{i-1}$  的所有元素均为整数，因此其所有代数余子式也均为整数，因此我们对  $1 \leq j < i$  有  $d_{i-1}\lambda_{i,j} \in \mathbb{Z}$ ，从而有  $d_{i-1}\mathbf{b}_i^* \in \mathbb{Z}^n$ 。

因此，对任意  $\mathbf{b}_i, |\mathbf{b}_i^*|^2, \mu_{i,j}$ ，其分母的进制长度均为  $O(\log d_i) = O(n \log V)$ 。

接下来分析分子。由于已经证明了分母的量级，我们只需要证明这些数值本身的量级即可证明分子的量级。

对于  $|\mathbf{b}_i^*|^2$ ，初始时有  $|\mathbf{b}_i^*|^2 \leq |\mathbf{b}_i|^2 \leq V$ ，且该量只在交换  $\mathbf{b}_{k-1}$  和  $\mathbf{b}_k$  时发生改变。根据上述讨论， $|\mathbf{c}_{k-1}^*|^2 < \gamma |\mathbf{b}_{k-1}^*|^2$ ；同时注意到  $\mathbf{c}_k^*$  是  $\mathbf{b}_{k-1}^*$  在  $\mathbb{R}\mathbf{c}_{k-1}^*$  上的正交投影，因此有  $|\mathbf{c}_k^*|^2 \leq |\mathbf{b}_{k-1}^*|^2$ ，也就是  $\max_{1 \leq i \leq n} |\mathbf{b}_i^*|^2$  在算法过程中单调不减，因此在整个算法过程中有  $|\mathbf{b}_i^*|^2 \leq V$ 。

在每一次迭代开始时（也就是 **Algorithm 2** 执行到第 10 行时），我们对  $|\mathbf{b}_i|^2$  和  $\mu_{i,j}$  有如下上界：

1. 对所有  $i \neq k$ ，有  $|\mathbf{b}_i|^2 \leq nV$ 。
2.  $|\mathbf{b}_k|^2 \leq n^2(4V)^n$ 。
3. 对所有  $1 \leq i < j < k$ ，有  $|\mu_{i,j}| \leq \frac{1}{2}$ 。
4. 对所有  $1 \leq i < j, j > k$ ，有  $|\mu_{i,j}| \leq \sqrt{nV^i}$ 。
5. 对所有  $1 \leq i < k$ ，有  $|\mu_{i,k}| \leq 2^{n-k} \sqrt{nV^{n-1}}$ 。

其中算法设定的  $k$  满足的条件保证了不等式 3。同时对于  $i < k$ ，我们可以直接由不等式 3 证明不等式 1：

$$|\mathbf{b}_i|^2 = |\mathbf{b}_i^*|^2 + \sum_{j=1}^{i-1} \mu_{j,i}^2 |\mathbf{b}_j^*|^2 \leq |\mathbf{b}_i^*|^2 + \sum_{j=1}^{i-1} \frac{1}{4} V = \frac{nV}{4} \leq nV$$

同理可由不等式 5 证明不等式 2。同时，由不等式 1 可以证明不等式 4：

$$|\mu_{i,j}|^2 = \frac{(\mathbf{b}_j \cdot \mathbf{b}_i^*)^2}{|\mathbf{b}_i^*|^4} \leq \frac{|\mathbf{b}_j|^2 |\mathbf{b}_i^*|^2}{|\mathbf{b}_i^*|^4} = |\mathbf{b}_j|^2 \cdot \frac{d_{i-1}}{d_i} \leq nV \cdot d_{i-1} \leq nV^i$$

接下来，为了证明上面的五个不等式，我们只需要证明不等式 1 在  $i > k$  时成立，以及证明不等式 5 成立即可。算法初始时我们有  $|\mathbf{b}_i|^2 \leq V$ ，并且可以用与用不等式 1 证明不等式 4 相同的方法证明初始时有  $|\mu_{i,j}| \leq \sqrt{nV^i}$ ，因此初始时不等式 1、5 均成立，我们只需要在假设一次迭代开始时不等式 1、5 均成立的条件下证明迭代结束后不等式 1、5 均保持成立即可证明整个算法过程中不等式 1、5 保持成立。

考虑  $k$  的变化。对于不等式 1，注意到进行一次情况 1 后，向量集合  $\{\mathbf{b}_i | i \neq k\}$  保持不变；进行一次情况 2 后，向量集合  $\{\mathbf{b}_i | i > k\}$  变为原来的一个子集，又由于不等式 1 对  $i < k$  成立，因此不等式 1 对  $i \neq k$  保持成立。

对于不等式 2，注意到在一次情况 2 的过程中，我们在将  $k$  增加 1 之前并不会改变  $\mu_{i,k+1}$ ，从而一次情况 2 结束后的不等式 5 可以由情况 2 开始前的不等式 4 推出，从而不等式 5 在一

次情况 2 后保持成立。对于情况 1，考虑到  $k$  减少 1，为了证明不等式 5 保持成立我们应该分析  $v_{i,k-1}$  的值。在算法过程中，从迭代开始到情况 1 结束，我们首先将  $\mu_{i,k}$  改为  $\mu_{i,k} - r\mu_{i,k-1}$ ，其中  $r$  是最接近  $\mu_{k-1,k}$  的整数；然后将  $v_{i,k-1}$  赋值为  $\mu_{i,k}$ 。从而有：

$$|v_{i,k-1}| = |\mu_{i,k} - r\mu_{i,k-1}|$$

注意到  $|r| \leq 2|\mu_{k-1,k}|$  且  $\mu_{i,k-1} \leq 1/2$ ，有：

$$|v_{i,k-1}| \leq |\mu_{i,k}| + |\mu_{k-1,k}|$$

对右侧的两个值同时应用不等式 5，有：

$$|v_{i,k-1}| \leq 2^{n-k+1} \sqrt{nV^{n-1}} = 2^{n-(k-1)} \sqrt{nV^{n-1}}$$

注意到  $k$  减少了 1，所以上式就是情况 1 结束后的不等式 5。所以我们证明了不等式 5 在情况 1 之后保持成立，即不等式 5 保持成立。因此在每次迭代开始时，所有变量的分子的二进制长度都是  $O(n \log V)$  的。

接下来我们需要对算法迭代过程中的值进行分析。由于调整  $\mu_{k-1,k}$  和情况 1 的过程中，任意一个变量都只经过了  $O(1)$  次代数运算，所以在这些过程中，上面的界自然成立。我们只需要对情况 2 过程中的调整进行分析。注意到调整  $\mu_{j,k}$  时， $\mu_{1,k}, \mu_{2,k}, \dots, \mu_{j-1,k}$  中的最大值最多翻倍，因此整个情况 2 的过程中， $\mu$  的最大值最多翻  $2^{k-1}$  倍。所以这个过程分子的二进制长度仍然是  $O(n \log V)$  的。这样我们就完成了对于分子量级的证明。  $\square$

这样，我们就可以证明 LLL 算法的时间复杂度上界为  $O(n^6(\log V)^3)$ ，也可以使用快速大整数乘除法做到  $O(n^{5+\epsilon}(\log V)^{2+\epsilon})$ ，其中  $\epsilon$  是任意大于 0 的实数。

## 4 整系数多项式分解

### 4.1 整系数多项式分解与格的联系

假设我们已经指定了多项式  $f, h \in \mathbb{Z}[x]$ （其中  $f$  不是常数）、质数  $p$  和正整数  $k$  满足：

1.  $h$  是首一多项式；
2.  $h \mid f \pmod{p^k}$ ；
3.  $h \bmod p$  在  $\mathbb{F}_p[x]$  上不可约；
4.  $(h \bmod p)^2 \nmid f \pmod{p}$ 。

我们记  $n = \deg f$ ， $l = \deg h$ ，则我们有  $1 \leq l \leq n$ 。此时我们有如下定理：

**定理 4.1.**  $f$  在  $\mathbb{Z}[x]$  内存在不可约因子  $h_0$ , 其中  $h \bmod p$  整除  $h_0 \bmod p$ , 且  $h_0$  被唯一确定到  $\pm 1$ 。

证明. 设  $f(x) = \prod_{i=1}^m (a_i(x))^{b_i}$ , 其中所有  $a_i$  在  $\mathbb{Z}[x]$  上不可约。若不存在  $i$  满足  $h \mid a_i \pmod{p}$ , 则  $h \nmid f \pmod{p}$ , 进而  $h \nmid f \pmod{p^k}$ , 矛盾。若存在  $1 \leq i < j \leq m$  满足  $h \mid a_i \pmod{p}$  且  $h \nmid a_j \pmod{p}$ , 则  $(h \bmod p)^2 \mid f \pmod{p}$ , 矛盾。因此存在唯一的  $1 \leq i \leq m$  满足  $h \mid a_i \pmod{p}$ , 由  $\mathbb{Z}[x]$  是唯一因子分解整环,  $a_i$  被唯一确定到  $\pm 1$ , 即得证。  $\square$

**定理 4.2.** 对于多项式  $g \in \mathbb{Z}[x]$ , 如果  $g \mid f$ , 那么  $h \mid g \pmod{p}$ 、 $h \mid g \pmod{p^k}$ 、 $h_0 \mid g$  三个命题等价。其中  $h_0$  的定义同定理 4.1。

证明. 命题 2、3 成立都可以简单推得命题 1 成立, 只需假定命题 1 成立, 证明命题 2、3 成立即可。

现假定  $h \mid g \pmod{p}$ 。如果  $h_0 \nmid g$ , 则  $h_0 \nmid (f/g)$ , 从而  $h_0 \nmid (f/g) \pmod{p}$ 。又由于  $h \mid g \pmod{p}$ , 因此  $(h \bmod p)^2 \mid f \pmod{p}$ , 矛盾。于是命题 3 得证。

由于  $h \bmod p$  在  $\mathbb{F}_p[x]$  上不可约, 又由于  $h_0 \nmid (f/g)$ , 从而  $h \bmod p$  和  $(f/g) \bmod p$  在  $\mathbb{F}_p[x]$  上互质。从而存在  $\lambda_1, \mu_1 \in \mathbb{Z}[x]$  使得:

$$\lambda_1 h + \mu_1 (f/g) \equiv 1 \pmod{p}$$

于是存在  $v_1 \in \mathbb{Z}[x]$  使得:

$$\lambda_1 h + \mu_1 (f/g) = 1 - v_1 p$$

两侧同时乘  $(1 + v_1 p + v_1^2 p^2 + \cdots + v_1^{k-1} p^{k-1})g$  得到:

$$(1 + v_1 p + v_1^2 p^2 + \cdots + v_1^{k-1} p^{k-1})g \lambda_1 h + (1 + v_1 p + v_1^2 p^2 + \cdots + v_1^{k-1} p^{k-1})\mu_1 f = 1 - v_1^k p^k$$

也就是存在  $\lambda_2, \mu_2 \in \mathbb{Z}[x]$  使得:

$$\lambda_2 h + \mu_2 f \equiv g \pmod{p^k}$$

由于左侧在  $(\mathbb{Z}/p^k\mathbb{Z})[x]$  上被  $h \bmod p^k$  整除, 因此右侧也被整除, 于是命题 2 得证。从而三个命题互相等价。  $\square$

从这里开始, 我们固定一个正整数  $m \geq l$ 。我们可以建立一个所有次数  $\leq m$  的多项式与向量空间  $\mathbb{R}^{m+1}$  的双射:

$$f = \sum_{i=0}^m a_i x^i \mapsto (a_0, a_1, \cdots, a_m)$$

不难发现我们在第 2 节中定义的  $|f|$  就等于其双射到的向量的欧几里得长度。从而我们可以讨论由多项式构成的格和基底。在后面讨论多项式构成的格和基底时, 我们默认使用该双射将其对应到向量空间中的格和基底。

现在考虑所有满足如下条件的多项式  $g$ :

1.  $g \in \mathbb{Z}[x]$  且  $\deg g \leq m$ ;

2.  $h \mid g \pmod{p^k}$ ;

设这些  $g$  构成集合  $\mathcal{L}$ 。

**定理 4.3.**  $\mathcal{L}$  是  $\mathbb{R}^{m+1}$  中的一个格, 且  $B = \{p^k x^i \mid 0 \leq i < l\} \cup \{hx^j \mid 0 \leq j \leq m-l\}$  是  $\mathcal{L}$  的一组基底。

证明. 由于  $B$  内所有多项式的次数互不相同, 所以  $B$  内所有元素线性无关。设  $B$  张成格  $\mathcal{L}'$ 。考虑  $B$  的任意整系数线性组合:

$$g(x) = \sum_{i=0}^{l-1} a_i p^k x^i + \sum_{i=0}^{m-l} b_i h x^i$$

则有:

$$g \equiv \sum_{i=0}^{m-l} b_i h x^i \pmod{p^k}$$

于是有  $h \mid g \pmod{p^k}$ 。同时显然有  $g \in \mathbb{Z}[x]$  且  $\deg g \leq m$ 。从而我们证明了  $\mathcal{L} \subset \mathcal{L}'$ 。

考虑任意  $g \in \mathcal{L}$ :

$$g(x) = h(x) \cdot \left( \sum_{i=0}^{m-l} c_i x^i \right) + \sum_{i=0}^m d_i p^k x^i$$

显然  $h(x) \cdot \left( \sum_{i=0}^{m-l} c_i x^i \right)$  可以被表示为  $B$  的整系数线性组合, 我们只需要对于任意  $0 \leq i \leq m$  证明  $p^k x^i$  可以被表示为  $B$  的整系数线性组合即可。

考虑归纳, 命题对于  $i \leq l-1$  显然成立。现假设命题对于  $i \leq q$  成立, 考虑  $i = q+1 \leq m$  的情况。注意到  $h$  是首一多项式, 于是有:

$$p^k x^{q+1} = p^k h(x) x^{q+1-l} - \sum_{i=0}^{l-1} p^k [x^i] h(x) x^{q+1-l+i}$$

由归纳假设, 右侧可以被表示为  $B$  的整系数线性组合。归纳即证得对于任意  $0 \leq i \leq m$ ,  $p^k x^i$  可以被表示为  $B$  的整系数线性组合。

于是任意  $g \in \mathcal{L}$  可被表示为  $B$  的整系数线性组合, 于是我们证明了  $\mathcal{L}' \subset \mathcal{L}$ 。

综上, 我们证明了  $\mathcal{L} = \mathcal{L}'$ , 即  $\mathcal{L}$  是由  $B$  张成的格。  $\square$

**定理 4.4.** 取多项式  $b \in \mathcal{L}$ 。如果  $p^{kl} > |f|^m |b|^n$ , 则  $h_0 \mid b$ 。其中  $h_0$  的定义同定理 4.1。

证明.  $b = 0$  时显然成立, 接下来仅考虑  $b \neq 0$  的情况。此时我们设  $g = \gcd(f, b)$ 。考虑到  $g \mid f$ , 由定理 4.2, 只需证明  $h \mid g \pmod{p}$ , 即有  $h_0 \mid g$ , 从而有  $h_0 \mid b$ 。

反证, 假设  $h \nmid g \pmod{p}$ , 那么由于  $h \pmod{p}$  在  $\mathbb{F}_p$  上不可约, 可得  $h \pmod{p}$  和  $g \pmod{p}$  在  $\mathbb{F}_p$  上互质, 即存在  $\lambda_3, \mu_3, \nu_3 \in \mathbb{Z}[x]$  使得  $\lambda_3 h + \mu_3 g = 1 - p\nu_3$ 。同时可以导出  $h \mid (f/g) \pmod{p}$ 。

记  $e = \deg g, m' = \deg b$ 。显然  $0 \leq e \leq m' \leq m$ 。对于多项式  $f$ , 定义  $R(f)$  为  $f - (f \bmod x^e)$ 。定义  $\mathcal{M}$ :

$$\mathcal{M} = \{R(\lambda f + \mu b) | \lambda, \mu \in \mathbb{Z}[x], \deg \lambda < m' - e, \deg \mu < n - e\}$$

我们可以证明  $\mathcal{M}$  是空间  $\sum_{i=e}^{n+m'-e-1} \mathbb{Z}x^i$  内的一个格, 且

$$B = \{R(x^i f) | 0 \leq i < m' - e\} \cup \{R(x^i b) | 0 \leq i < n - e\}$$

是  $\mathcal{M}$  的一个基底。

考虑到  $R$  满足对于任意多项式  $s, t \in \mathbb{Z}[x]$ , 有  $R(s) + R(t) = R(s + t)$ , 因此  $\mathcal{M}$  中任意元素可被表示为  $B$  内所有元素的整系数线性组合, 以及  $B$  内的元素的任意整系数线性组合均在  $\mathcal{M}$  中存在都是显然的, 我们只需证明  $B$  内的元素线性无关即可。

考虑  $B$  内的元素的整系数线性组合一定可以被表示为  $R(\lambda f + \mu b)$ 。若  $R(\lambda f + \mu b) = 0$ , 则  $\deg(\lambda f + \mu b) < e = \deg g$ 。但是  $g \mid (\lambda f + \mu b)$ , 所以  $\lambda f + \mu b = 0$ 。两侧同时除  $g$  得到  $\lambda(f/g) + \mu(b/g) = 0$ 。由于  $f/g$  和  $b/g$  互质, 因此有  $(b/g) \mid \lambda$ 。又由于  $\deg \lambda < m' - e = \deg(b/g)$ , 所以  $\lambda = 0$ , 所以  $\lambda = \mu = 0$ 。如此即可证明  $B$  内元素线性无关, 并证明  $\mathcal{M}$  是  $B$  张成的格。

由 Hadamard 不等式, 我们有:

$$\begin{aligned} \det \mathcal{M} &\leq \prod_{i=0}^{m'-e-1} |R(x^i f)| \cdot \prod_{i=0}^{n-e-1} |R(x^i b)| \\ &\leq \prod_{i=0}^{m'-e-1} |x^i f| \cdot \prod_{i=0}^{n-e-1} |x^i b| \\ &= |f|^{m'-e} |b|^{n-e} \\ &\leq |f|^m |b|^n \\ &< p^{kl} \end{aligned}$$

同时, 由 [3, Chap. I, Theorem I.A], 我们可以选定一个  $\mathcal{M}$  的基底  $B' = \{b_e, b_{e+1}, \dots, b_{n+m'-e-1}\}$  对所有  $e \leq i \leq n + m' - e - 1$  满足  $\deg b_i = i$ 。此时有  $\det \mathcal{M}$  等于所有  $b_i$  的首项系数的乘积。同时我们可以证明: 对于  $e \leq i \leq e + l - 1$ ,  $b_e$  的首项系数是  $p^k$  的倍数。又由  $g \mid b$  和  $h \mid (f/g) \pmod{p}$  可分别导出  $e \leq m'$  和  $l \leq n - e$ , 可得  $e + l - 1 \leq n + m' - e - 1$ 。于是可证明  $\det \mathcal{M} \geq p^{kl}$ , 结合上述由 Hadamard 不等式导出的不等式即得到矛盾。

为了证明对于  $e \leq i \leq e + l - 1$  有  $b_e$  的首项系数是  $p^k$  的倍数, 我们定义  $\mathcal{M}'$ :

$$\mathcal{M}' = \{\lambda f + \mu b | \lambda, \mu \in \mathbb{Z}[x], \deg \lambda < m' - e, \deg \mu < n - e\}$$

显然对任意  $s \in \mathcal{M}$ , 存在  $t \in \mathcal{M}'$ , 满足  $\deg s = \deg t$  且  $R(t) = s$ 。因此我们只需证明对任意  $t \in \mathcal{M}'$ ,  $\deg t < e + l$ , 满足  $t$  的所有系数是  $p^k$  的倍数, 即可证明上述待证结论。

为证明该结论, 考虑任意  $t \in \mathcal{M}'$ ,  $\deg t < e + l$ 。由  $\mathcal{M}'$  定义可得  $g \mid t$ 。对  $\lambda_3 h + \mu_3 g = 1 - p v_3$  的两侧同时乘  $(1 + v_3 p + \dots + v_3^{k-1} p^{k-1})(t/g)$  可得存在  $\lambda_4, \mu_4 \in \mathbb{Z}[x]$  满足:

$$\lambda_4 h + \mu_4 t \equiv t/g \pmod{p^k}$$



由  $\mathcal{L}$  的定义可得  $h \mid b \pmod{p^k}$ ; 又由  $t = \lambda f + \mu b, h \mid f \pmod{p^k}, h \mid b \pmod{p^k}$  可得  $h \mid t \pmod{p^k}$ 。由上式可得  $h \mid (t/g) \pmod{p^k}$ 。又由  $\deg(t/g) < e + l - e = l = \deg h$  可得  $t/g \equiv 0 \pmod{p^k}$ , 即  $t \equiv 0 \pmod{p^k}$ , 即  $t$  的所有系数是  $p^k$  的倍数。

综上, 原命题得证。  $\square$

**定理 4.5.** 设  $\{b_1, b_2, \dots, b_{m+1}\}$  是  $\mathcal{L}$  的一个 LLL 不可约的基底, 且有如下不等式:

$$p^{kl} > \left(\frac{4}{4y-1}\right)^{mn/2} \left(\frac{2m}{m}\right)^{n/2} |f|^{m+n}$$

则  $\deg h_0 \leq m$  当且仅当:

$$|b_1| < \left(\frac{p^{kl}}{|f|^m}\right)^{1/n}$$

证明. 由  $|b_1|$  上的不等式限制推得  $\deg h_0 \leq m$  由定理 4.4 是显然的。对于另一个方向, 假定有  $\deg h_0 \leq m$ , 那么由  $h|h_0 \pmod{p}$ , 在定理 4.2 中取定  $g = h_0$  即得  $h|h_0 \pmod{p^k}$ 。由定义, 有  $h_0 \in \mathcal{L}$ 。在定理 3.4 中取定  $v = h_0$  即得:

$$|b_1| \leq \left(\frac{4}{4y-1}\right)^{m/2} |h_0|$$

又由 [5] 中的结果, 有:

$$|h_0| \leq \left(\frac{2m}{m}\right)^{1/2} |f|$$

结合以上不等式, 有:

$$|b_1| \leq \left(\frac{4}{4y-1}\right)^{m/2} \left(\frac{2m}{m}\right)^{1/2} |f| = \left(\left(\frac{4}{4y-1}\right)^{mn/2} \left(\frac{2m}{m}\right)^{n/2} |f|^n\right)^{1/n} < \left(\frac{p^{kl}}{|f|^m}\right)^{1/n}$$

即得证。  $\square$

**定理 4.6.** 在定理 4.5 的基础上, 假设存在  $1 \leq j \leq m+1$  使得

$$|b_j| < \left(\frac{p^{kl}}{|f|^m}\right)^{1/n}$$

设  $t$  为满足上述条件的  $j$  的最大值, 则有  $\deg h_0 = m+1-t, h_0 = \gcd(b_1, b_2, \dots, b_t)$ , 且上述条件对所有  $1 \leq j \leq t$  成立。

证明. 设集合  $S$  为所有满足上述不等式的  $j$  构成的集合。由定理 4.4 可得对所有  $j \in S$ , 有  $h_0 \mid b_j$ 。设  $h_1 = \gcd\{b_j | j \in S\}$ , 则有  $h_0 \mid h_1$ 。

考虑到对所有  $j \in S$ , 有  $h_1 \mid b_j$ , 那么这些  $b_j$  都在由基底  $\{h_1, h_1 x, \dots, h_1 x^{m-\deg h_1}\}$  张成的  $m - \deg h_1 + 1$  维的格内。又由于所有  $b$  线性无关, 所以  $\#S \leq m - \deg h_1 + 1$ 。

同时, 在定理 3.4 中取  $t = m - \deg h_0 + 1, v_1 = h_0, v_2 = h_0 x, \dots, v_t = h_0 x^{t-1}$ , 结合 [5] 中的结果和定理 4.5 的证明, 有:

$$\forall 1 \leq j \leq m - \deg h_0 + 1, \text{ s.t. } |b_j| \leq \left( \frac{4}{4y-1} \right)^{m/2} \binom{2m}{m}^{1/2} |f| < \left( \frac{p^{kl}}{|f|^m} \right)^{1/n}$$

因此  $\{1, 2, \dots, m - \deg h_0 + 1\} \subset S$ , 即  $\#S \geq m - \deg h_0 + 1$ 。又因为  $h_0 | h_1$ , 所以  $\deg h_0 \leq \deg h_1$ 。综合以上不等式可得  $\#S = m - \deg h_0 + 1 = m - \deg h_1 + 1$ , 从而有  $\deg h_0 = \deg h_1 = m + 1 - t$  和  $S = \{1, 2, \dots, t\}$ 。

同时我们可以证明  $h_1$  是本原多项式: 对  $j \in S$ , 设  $d_j$  为  $b_j$  的容度。由于  $h_0$  是首一多项式, 同时由定理 4.4, 可得  $h_0 | (b_j/d_j)$ 。由定理 4.2 可得  $(b_j/d_j) \in \mathcal{L}$ 。但是同时  $b_j$  属于  $\mathcal{L}$  的一个基底, 因此  $d_j = 1$  且  $b_j$  是本原多项式。又由于  $h_1$  是  $b_j$  的因子, 从而  $h_1$  也是本原多项式。又由于  $\deg h_0 = \deg h_1$  且  $h_0 | h_1$ , 可得  $h_0 = \pm h_1$ 。综上, 原命题得证。□

## 4.2 分解算法

### 4.2.1 判定是否有 $\deg h_0 \leq m$

假设已经指定了多项式  $f \in \mathbb{Z}[x]$ 、 $n = \deg f$ , 并指定了质数  $p$ 、正整数  $k$  和多项式  $h \in (\mathbb{Z}/p^k\mathbb{Z})[x]$ 、 $l = \deg h$ , 和正整数  $m \geq l$ , 其中  $f, h, p, k$  满足 4.1 节开头处给定的条件, 且有

$$p^{kl} > \left( \frac{4}{4y-1} \right)^{mn/2} \binom{2m}{m}^{n/2} |f|^{m+n}$$

成立。我们将给出一个算法判断是否有  $\deg h_0 \leq m$ , 并且在有  $\deg h_0 \leq m$  的情况下求出  $h_0$ 。

该算法是定理 4.5 和定理 4.6 的直接应用, 具体见下 **Algorithm 3**。

---

**Algorithm 3** 整系数多项式分解: 判定是否有  $\deg h_0 \leq m$

---

**Input:**  $n, f, p, k, h, l, m, y$

**Output:** 求出  $h_0$  或报告  $\deg h_0 > m$

$B \leftarrow \{p^k x^i | 0 \leq i < l\} \cup \{h x^j | 0 \leq j \leq m - l\}$

Run **Algorithm 2** with input  $(m+1, \mathcal{L}, B, y)$ , store the result in  $B$

if  $|b_1|^n |f|^m \geq p^{kl}$  then

    return  $\deg h_0 > m$

else

$t \leftarrow 1$

$h_0 \leftarrow b_1$

    while  $|b_{t+1}|^n |f|^m < p^{kl}$  do

$t \leftarrow t + 1$

$h_0 \leftarrow \gcd(h_0, b_t)$

```

end while
return h_0
end if

```

其中多项式 gcd 的计算使用 [1, Sect. 4.6.1] 中提到的子结式算法。

**定理 4.7.** *Algorithm 3* 需要  $O(m^4 k \log p)$  次算术运算, 且涉及到的数的二进制长度均为  $O(mk \log p)$ 。

证明. 注意到  $h$  的任意系数在  $[0, p^k]$  范围内, 因此有  $|h| \leq 1 + lp^{2k}$ , 即  $O(\log |h|) = O(k \log p + \log l)$ 。又由  $l \leq m, l \leq n$  和  $p^{kl} > \left(\frac{4}{4y-1}\right)^{mn/2}$ , 可得  $O(\log l) = O(k \log m)$ 。因此由定理 3.6 和定理 3.7, LLL 算法的部分满足上述估计。计算 gcd 的部分容易验证满足同样的估计。□

#### 4.2.2 确定 $h_0$

下面我们给出一个算法在给定了  $f, h, n, l, p, y$  的情况下确定  $h_0$ , 其中  $f, h, n, l, p$  需要在  $k = 1$  的情况下满足 4.1 节开头处提到的四个条件。我们在该算法中需要对于正整数  $k \geq 1$ , 求出多项式  $h'$  使得  $h' \equiv h \pmod{p}$  且  $h' | f \pmod{p^k}$ 。这个问题可以应用 Hensel 引理解决, 具体见 [1, Exercise 4.6.2.22]。

我们首先特殊处理  $l = n$  的情况, 此时必有  $h_0 = f$ 。对于  $l < n$  的情况, 我们首先求一个  $k$  使得对最大的  $m = n - 1$  有  $p^{kl} \leq \left(\frac{4}{4y-1}\right)^{mn/2} \binom{2m}{m}^{n/2} |f|^{m+n}$  成立, 然后倍增地枚举  $m$  直到  $m = n - 1$ 。过程中如果求出  $h_0$  则返回之; 否则有  $\deg h_0 > n - 1$ , 即  $h_0 = f$ 。具体见下 **Algorithm 4**。

---

**Algorithm 4** 整系数多项式分解: 确定  $h_0$

---

**Input:**  $f, h, n, l, p, y$

**Output:**  $h_0$

if  $l = n$  then

return  $f$

end if

$k \leftarrow 1, m \leftarrow n - 1$

while  $p^{kl} \leq \left(\frac{4}{4y-1}\right)^{mn/2} \binom{2m}{m}^{n/2} |f|^{m+n}$  do

$k \leftarrow k + 1$

end while

Calculate  $h'$  satisfying  $h' \equiv h \pmod{p}$  and  $h' | f \pmod{p^k}$  with [1, Exercise 4.6.2.22]

$u \leftarrow$  the greatest integer for which  $l \leq (n - 1)/2^u$

for  $i = u, u - 1, \dots, 0$  do

Run **Algorithm 3** with input  $(n, f, p, k, h, l, m = \lfloor (n - 1)/2^i \rfloor, y)$

```
if $\deg h_0 \leq m$ then
 return h_0
end if
end for
return f
```

**定理 4.8.** 设  $m_0 = \deg h_0$ , 则上述算法需要进行  $O(m_0 n^3(n^2 + n \log |f| + \log p))$  次算术运算, 过程中涉及到的数的二进制长度均为  $O(n^3 + n^2 \log |f| + n \log p)$ 。

证明. 注意到

$$p^{(k-1)l} \leq \left(\frac{4}{4y-1}\right)^{(n-1)n/2} \binom{2n-2}{n-1}^{n/2} |f|^{2n-1}$$

同时由  $p^{k-1} \leq p^{(k-1)l}$  可得:

$$k \log p = (k-1) \log p + \log p = O(n^2 + n \log |f| + \log p)$$

由于调用一次 **Algorithm 3** 需要  $O(m^4 k \log p)$  次运算, 同时算法过程中是倍增枚举  $m$ , 因此调用 **Algorithm 3** 的总次数就是  $O(m_0^4 k \log p) \leq O(m_0 n^3 k \log p)$ 。数值大小显然和 **Algorithm 3** 是同级的, 即  $O(m_0 k \log p) \leq O(nk \log p)$ 。代入  $O(k \log p) = O(n^2 + n \log |f| + \log p)$  即可得到命题中的估计。

使用 Hensel 定理的部分的运算量显然不会超过  $O(n^3 k)$ , 且运算过程均为模  $p^k$ , 因此数值大小也不超过上述估计, 即原命题得证。□

### 4.2.3 完整算法

现在我们将指定本原多项式  $f \in \mathbb{Z}[x]$ ,  $n = \deg f$ , 并给出一个算法求出  $f$  在  $\mathbb{Z}[x]$  上的分解。

考虑  $f$  和  $f$  的导数的结式  $R(f, f')$ 。我们首先考虑  $R(f, f') \neq 0$  的情况。此时我们取最小的质数  $p$  使得  $R(f, f') \not\equiv 0 \pmod{p}$ 。由多项式判别式的定义可以得到此时  $f \bmod p$  的次数仍然是  $n$ , 且  $f$  在模  $p$  意义下没有重根。于是我们使用 [1, Sect. 4.6.2] 提到的 Berlekamp 分解算法求出  $(f \bmod p)$  在  $\mathbb{F}_p$  上的分解之后, 得到的所有因子  $h$  都满足 4.1 开头处要求的  $h \nmid f \pmod{p}$ 。

接下来设  $(f \bmod p)$  的所有因子构成集合  $S$ , 重复以下流程: 每次从  $S$  中选出一个元素  $h$ , 将其整体乘  $h$  的首项在模  $p$  意义下的乘法逆元使得  $h$  是首一多项式, 并使用 **Algorithm 4** 求出其对应的  $h_0$ , 并从  $S$  中删去所有满足  $h_1 | h_0 \pmod{p}$  的  $h_1$ , 直到将  $S$  删空为止。如此就可以得到多项式  $f$  在  $\mathbb{Z}[x]$  上的分解。

接下来考虑  $R(f, f') = 0$  的情况。此时设  $g = \gcd(f, f')$ , 那么  $(f/g)$  在  $\mathbb{Z}[x]$  上没有重复因子, 即  $R((f/g), (f/g)') \neq 0$ 。于是通过上述流程分解  $(f/g)$ , 同时注意到  $g$  只包含  $f$  在  $\mathbb{Z}[x]$

上的重复因子, 于是有  $g \mid (f/g)$ 。同时我们已经得到了  $(f/g)$  的分解, 所以使用  $(f/g)$  的所有因子依次进行试除即可。

该算法过程很直接, 因此这里略去伪代码。算法正确性是显然的, 而算法的复杂度需要进一步讨论。

**定理 4.9.** 上述算法需要  $O(n^6 + n^5 \log |f|)$  次算术运算, 涉及到的数的二进制长度均为  $O(n^3 + n^2 \log |f|)$ 。

证明. 首先考虑  $R(f, f') = 0$  时, 由于  $(f/g)$  是  $f$  的因子, 由 [5] 的结果, 有  $\log |(f/g)| = O(n + \log |f|)$ 。不难发现将其代入命题中给出的估计不会产生任何影响。同时子结式算法的开销显然在上述估计范围内。于是只需考虑  $R(f, f') \neq 0$  的情况。

对于  $R(f, f') \neq 0$  的情况, 我们首先分析  $p$  的上界。注意到  $p$  是最小的不整除  $R(f, f')$  的质数, 于是有

$$\prod_{q < p, q \text{ is prime}} q \leq |R(f, f')|$$

同时由 [2, Sect. 22.2], 可以证明对任意  $p > 2$ , 存在正常数  $A$  满足:

$$\prod_{q < p, q \text{ is prime}} q > e^{Ap}$$

又注意到  $|f'| \leq n|f|$ , 由 Hadamard 不等式, 可得:

$$R(f, f') \leq n^n |f|^{2n-1}$$

综上所述有  $p = 2$  或  $e^{Ap} < n^n |f|^{2n-1}$ , 即

$$p < \max \left\{ 3, \frac{n \log n + (2n-1) \log |f|}{A} \right\}$$

由上述分析, 可以发现我们能够在  $O(n(\log n + \log |f|))$  次二进制长度为  $O(n(\log n + \log |f|))$  的数的运算内求出  $p$ , 满足命题中的估计; 同时对 **Algorithm 4** 的运算量的估计中包含  $\log p$  的项都可以被其他项吸收。

Berlekamp 分解算法的运算次数和过程量的数量级都显然满足命题中的估计。考虑调用 **Algorithm 4** 的部分, 设每次求出的  $h_0$  的度数为  $m_0$ , 则这次调用需要  $O(m_0 n^3 (n^2 + n \log |f|))$  次运算, 涉及到的数的二进制长度均为  $O(n^3 + n^2 \log |f|)$ 。考虑到  $\sum m_0 = n$ , 总运算次数即为  $O(n^6 + n^5 \log |f|)$ 。该算法的其他部分显然满足同样的估计, 于是原命题得证。□

综上所述, 我们得到了一个时间复杂度为  $O(n^{12} + n^9 (\log |f|)^3)$  的整系数本原多项式分解算法。同样的, 我们可以使用快速大整数乘除法将复杂度降低至  $O(n^{9+\epsilon} + n^{7+\epsilon} (\log |f|)^{2+\epsilon})$ , 其中  $\epsilon$  是任意大于 0 的实数。



## 参考文献

- [1] Knuth D.E. The art of computer programming, Vol. 2, Seminumerical algorithms. Addison-Wesley Professional, 1981.
- [2] Hardy G.H., Wright E.M. An introduction to the theory of numbers. Oxford: Oxford University Press, 1979.
- [3] Cassels J.W.S. An introduction to the geometry of numbers. Berlin, Heidelberg, New York: Springer, 1971.
- [4] A.K.Lenstra, H.W.Lenstra Jr., L.Lovász. Factoring polynomials with rational coefficients. Mathematische Annalen, 261:515–534, 1982.
- [5] Mignotte M. An inequality about factors of polynomials. Math. Comp., 28:1153–1157, 1974.





# 浅谈图的列表着色问题

成都市第七中学 钟自厚

## 摘要

本文简述了关于每个点可染颜色带限制的图染色问题的一些结论和猜想，并展示了组合问题证明中强大的概率方法，最后给出了信息学竞赛中的一些例题。

## 1 引言

图染色是图论领域非常重要的问题之一。图染色的一个经典变体是给每个点可染的颜色施加限制，并求最少需要的颜色数。

本文主要介绍列表着色，这是一种带限制图染色问题，导出了图的 *choosability* 属性。

对于图  $G$ ，记  $V(G)$  为它的点集，二元组集合  $E(G)$  为它的边集（若  $G$  是有向图则二元组有序）。记  $L(G)$  为它的线图 (Line Graph)，即以  $E(G)$  作为顶点集合的图。

我们下文提到的染色默认相邻点（边）颜色不同 (Proper Coloring)。记  $\chi(G)$  为  $G$  的色数 (Chromatic Number)，表示给点染色最少需要的颜色数。记  $\chi'(G) = \chi(L(G))$  为  $G$  的边色数 (Edge Chromatic Number / Chromatic Index)，表示给边染色最少需要的颜色数。

对于图  $G$  和函数  $f: V(G) \rightarrow \mathbb{N}^*$ ，称  $G$  是  $f$ -choosable 的，当无论如何给每个点  $v \in V(G)$  分配大小为  $f(v)$  的整数集合  $S(v)$ ，都存在染色  $c: V(G) \rightarrow \mathbb{Z}$  满足  $\forall v \in V(G), c(v) \in S(v)$ 。称  $G$  是  $k$ -choosable 的，当  $G$  对于常值函数  $f(v) = k$  是  $k$ -choosable 的。

定义图  $G$  的列表着色数 (List Chromatic Number)  $\chi_\ell(G)$  为最小的正整数  $k$  满足  $G$  是  $k$ -choosable 的。类似于边色数，记  $G$  的边列表着色数 (Edge-List Chromatic Number) 为  $\chi'_\ell(G) = \chi_\ell(L(G))$ 。

有向图  $D$  的核 (Kernel) 为  $K \subseteq V(D)$  满足

- $\forall v_1, v_2 \in K, (v_1, v_2) \notin E(D)$
- $\forall v \notin K, \exists k \in K, (v, k) \in E(D)$

## 2 列表着色猜想

根据定义，显然有  $\chi_\ell(G) \geq \chi(G)$  及  $\chi'_\ell(G) \geq \chi'(G)$ 。

我们很容易想到，先考虑何时会出现  $\chi_\ell(G) > \chi(G)$ 。这一情况是存在的，例如二分图  $K_{2,4}$  的色数显然为 2，但下面的例子说明它不是 2-choosable 的。

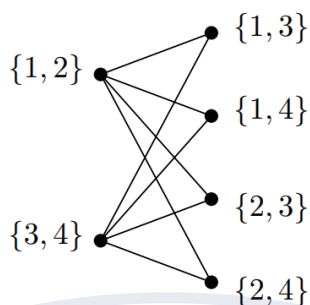


图 1: 对  $K_{2,4}$  进行列表着色的例子

更进一步地，对于任意  $k \geq 2$ ，都能构造二分图使得列表着色数至少为  $k$ 。这就说明， $\chi_\ell(G)$  相比于  $\chi(G)$  可以无限大 [1]。

Bollobas 和 Harris 提出猜想 [2]，在线图中上述两个值总是相等。

**猜想 1 (List Coloring Conjecture).** 对于任意图  $G$ ， $\chi'_\ell(G) = \chi'(G)$ 。

图染色 Brooks 定理指出，非完全图或奇环的连通图的色数不超过最大度数。我们有这一定理的 choosability 版本。

**定理 1 ([3]).** 非完全图或奇环的连通图的列表着色数，不超过最大度数。

从这一定理，我们可以得到平面图是 6-choosable 的。这是因为平面图（的所有子图）都总存在度数不超过 5 的点，我们可以不断删除这些点并进行归纳。1994 年 C. Thomassen 进一步证明了每个平面图都是 5-choosable 的 [4]，这个界是紧的。

目前，列表着色猜想在一些特殊图上已经得到证明。例如  $G$  的线图是完美图 (Perfect Graph) 时 [5]，即  $G$  的每个双连通分类都是二分图、 $K_4$  或  $K_{1,1,n}$ 。由于篇幅所限，下面仅给出对于  $G$  是二分图的证明。

顺带一提，Jeff Kahn 在 2000 年证明了列表着色猜想当  $\chi'_\ell(G) \rightarrow +\infty$  时渐进成立 [6]。

### 3 二分图的证明

数学家 Jeff Dinitz 在 1979 年提出了一个有趣的猜想：对于  $n$  阶拉丁方，每个格子给出大小为  $n$  的候选数集合，则总是存在给每个格子填数的合法方案。

1993 年，Jeannette Janssen 解决了 Dinitz 猜想的弱化版 [7]，即每个格子给出大小为  $n+1$  的集合。1995 年，Fred Galvin 证明了 Dinitz 猜想 [8]，并且将猜想推广到任意二分图。

下面，我们给出 Galvin 的证明。这一证明非常巧妙。

### 3.1 核的性质

记  $d_G^+(v)$  为  $v$  的出度。

**引理 1.** 设  $G$  为无向图,  $\{S_v\}_{v \in V(G)}$  为染色列表。若  $G$  存在定向  $D$  使得  $\forall v \in G, d_G^+(v) < |S_v|$  且  $D$  的每个导出子图都有核, 那么  $G$  可以用  $\{S_v\}$  染色。

**证明.** 对  $|V(G)|$  归纳。显然  $|V(G)| = 1$  时引理成立。

对于  $V(G) > 1$ , 设对所有点的数量小于  $|V(G)|$  的图引理成立。取  $\alpha$  为任意出现在至少一个列表的颜色, 考虑所有对应列表包含  $\alpha$  的点导出的子图, 根据假设存在  $U$  为其核。

设  $G'$  为  $G \setminus U$ ,  $S'_v = (S_v \setminus \{\alpha\})_{v \in G'}$ 。对于  $v \in V(G')$ , 若  $\alpha \notin S_v$  则  $d_{G'}^+(v) < |S_v| = |S'_v|$ , 否则根据核的定义  $d_G^+(v) < d_G^+(v), |S_v| = |S'_v| + 1 \implies d_{G'}^+(v) < |S'_v|$ 。

给所有  $U$  中的点染上  $\alpha$ , 根据归纳假设  $G'$  也有列表着色, 因此引理成立。  $\square$

### 3.2 边染色方案构造定向

设  $G = (X \cup Y, E(G))$  为二分图,  $c$  是  $G$  的一种边染色方案。定义对  $L(G)$  的定向  $D(G, c)$  为, 对于任意两个在  $G$  中共享顶点的边  $e_1, e_2$ , 不妨设  $c(e_1) < c(e_2)$ , 用如下方式给  $(e_1, e_2) \in E(L(G))$  定向:

- 若  $e_1, e_2$  只在  $X$  共享顶点, 定向为  $e_1 \rightarrow e_2$ 。
- 若  $e_1, e_2$  只在  $Y$  共享顶点, 定向为  $e_2 \rightarrow e_1$ 。
- 若  $e_1, e_2$  是重边, 那么  $L(G)$  会产生两条重边, 分别定向为  $e_1 \rightarrow e_2$  和  $e_2 \rightarrow e_1$ 。

**引理 2.** 对于二分图  $G$ , 设  $k = \chi'(G) = \max_{v \in V(G)} d_G(v)$ ,  $c : E(G) \rightarrow [1, k] \cap \mathbb{Z}$  为一种染色方案, 那么  $\forall v \in V(D(G, c)), d_{D(G, c)}^+(v) < k$ 。

**证明.** 考虑  $e \in V(D(G, c))$ , 设  $c(e) = \alpha$ ,  $e' \in V(D(G, c))$  为与  $e$  共享顶点的边且  $(e, e')$  定向为  $e' \rightarrow e$ , 根据  $D(G, c)$  的构造总是有  $c(e') \in \{1, 2, \dots, i-1, i+1, \dots, k\}$ , 所以  $d_{D(G, c)}^+(e) \leq |\{1, 2, \dots, i-1, i+1, \dots, k\}| < k$ 。  $\square$

### 3.3 稳定婚姻

考虑稳定婚姻 (Stable Matching) 模型。我们先明确稳定婚姻的定义: 对于二分图  $G$ , 每个点  $v \in V(G)$  有对所有邻边的排列称为偏好列表  $L_v$ , 则总是存在稳定匹配  $M \subseteq E(G)$ 。匹配  $M$  是稳定的当且仅当  $\forall e \notin M, \exists e' \in M$  且  $e, e'$  共享顶点  $v \in V(G)$ ,  $v$  相比  $e$  更偏好  $e'$ 。

**引理 3.** 对于二分图  $G = (X \cup Y, E(G))$ , 设  $k = \chi'(G) = \max_{v \in V(G)} d_G(v)$ ,  $c : E(G) \rightarrow [1, k] \cap \mathbb{Z}$  为一种染色方案, 那么  $D(G, c)$  的每个导出子图都有核。

证明. 在  $V(G)$  上定义偏好列表, 若  $(e_1, e_2) \in E(L(G))$  定向为  $e_1 \rightarrow e_2$ , 那么对于  $v \in V(G)$  令  $v$  相比  $e_1$  更偏好  $e_2$ . 求解  $V(G)$  的稳定婚姻就可以给出  $D(G, c)$  的核  $W$ , 对于导出子图也是类似的。

我们反证  $W$  必定是核。若  $W$  不是核, 则  $\exists e \notin W$  满足  $e$  不存在到  $W \in V(L(G))$  中的点的出边。显然  $e$  与  $W$  中的至少一个边  $e_1$  共享顶点 (否则不是极大匹配)  $v$ , 那么  $D(G, c)$  中有定向  $e_1 \rightarrow e$ , 且  $v$  相比  $e_1$  更偏好  $e$ 。若  $e$  不与  $W$  中的另一个边  $e_2$  共享顶点则  $W$  不是稳定匹配, 否则同理可推出  $e$  与  $e_2$  的公共顶点  $w$  相比  $e_2$  更偏好  $e$ ,  $W$  仍然不是稳定匹配。□

于是我们构造的定向  $D(G, c)$  符合引理 1 的假设。证毕。

## 4 概率方法

概率方法已经广泛应用于图染色问题。

概率方法是一种非常强大的工具。简而言之就是说, 如果一个集合不包含某个性质的对象这一事件的概率小于 1, 那么这个集合必然包含某个满足性质的对象。

下面是一个应用概率方法的例子。

**定理 2 ([9]).** 设  $G$  的最大度数为  $\Delta$ , 列表着色数为  $r \geq \Delta$ 。设  $d : E(G) \rightarrow [1, r] \cap \mathbb{Z}$  为边的禁止颜色。那么设  $t$  为最小的整数满足  $t! > n$ , 存在染色  $c : E(G) \rightarrow [1, r+t] \cap \mathbb{Z}$  满足  $\forall e \in E(G), c(e) \neq d(e)$ 。

证明. 固定用  $[1, r]$  中颜色的一个染色方案。考虑随机打乱边的颜色, 那么对于某个点  $v \in V(G)$ , 至少有  $t$  条与  $v$  相邻的边染了禁止颜色的概率不超过

$$\binom{\Delta}{t} \frac{1}{r(r-1) \cdots (r-t+1)} \leq \frac{1}{t!} < \frac{1}{n}$$

将每个点的概率相加小于 1, 所以必然存在某个排列使得每个点都至多有  $t-1$  条相邻的边违反了限制。设  $H$  为此时所有违反限制的边形成的子图, 那么  $H$  中点的最大度数小于  $t$ 。若  $H$  中有重边, 交换它们的颜色即可减少违反限制的边数。因此可以假设  $H$  是简单图, 根据 Vizing 定理,  $H$  中的边可以用  $[r+1, r+t]$  中颜色染色。□

另一个值得研究的问题是完全  $k$  分图 (Complete  $k$ -Partite Graph) 的列表着色数。记  $K_{m \times r}$  为一个完全  $r$  分图, 其中每部分都有  $m$  个点。平凡的情形是  $\chi_\ell(K_{m \times 1}) = 1, \chi_\ell(K_{r \times 1}) = r$ 。

下面给出了对其余情形  $\chi_\ell(K_{m \times r})$  只相差常数倍的估计。

注意下面的  $\log$  默认以  $e$  为底。

**定理 3 ([10]).** 存在正常数  $C$  使得对任意正整数  $m \geq 2$  和  $r$  都有

$$\chi_\ell(K_{m \times r}) \leq Cr \log m$$

证明. 显然  $\chi_\ell(K_{m \times r}) \leq rm$ . 所以选择  $c \geq 4$  即可无需考虑  $m \leq c$  的情况。

设  $V_1, V_2, \dots, V_r$  ( $|V_i| = m$ ) 是  $K = K_{m \times r}$  的每部分的点集. 对于每个点  $v \in V(K)$ , 设  $S(v)$  为大小不小于  $cr \log m$  (我们之后会确定  $c$  的值) 的集合. 我们需要证明存在用  $S(v)$  给每个点染色的方案. 因为  $\chi_\ell(K_{m \times r})$  单调递增, 不妨假设  $r$  是 2 的幂。

**情形一:  $r \leq m$ .**

设  $S = \cup_{v \in V(K)} S(v)$ . 设  $R = [1, r] \cap \mathbb{Z}$ ,  $f: S \rightarrow R$  是一个随机函数, 对每个  $c \in S$  满足  $f(c)$  的值互相独立且是  $R$  上的均匀分布。

我们只适用满足  $f(c) = i$  的  $c$  来染  $V_i$  中的点, 可以避免相邻的点颜色相同. 那么说明对每个  $i$  ( $1 \leq i \leq r$ ) 和  $v \in V_i$ , 都能找到至少一个  $c \in S(v)$  满足  $f(c) = i$  即可完成证明。

固定  $i$  和  $v \in V_i$ , 计算没有颜色  $c \in S(v)$  满足  $f(c) = i$  的概率并进行放缩得

$$\left(1 - \frac{1}{r}\right)^{|S(v)|} \leq \left(1 - \frac{1}{r}\right)^{cr \log m} \leq e^{-c \log m} \leq \frac{1}{m^c} < \frac{1}{rm}$$

注意我们已经默认条件  $m > c \geq 4$  及  $r \leq m$  成立。

有  $rm$  种选择  $i, v$  的方式, 由于不合法的概率之和小于 1, 所以必然存在某个  $f$  合法。

**情形二:  $r > m$ .**

沿用  $S$  和  $R$  的定义. 再定义  $R_1 = [1, \frac{r}{2}] \cap \mathbb{Z}$ ,  $R_2 = [\frac{r}{2} + 1, r] \cap \mathbb{Z}$ 。

设  $f: S \rightarrow \{1, 2\}$  是一个随机函数. 我们令满足  $f(c) = 1$  的颜色  $c$  只染  $\cup_{i \in R_1} V_i$  中的点,  $f(c) = 2$  的颜色  $c$  只染  $\cup_{i \in R_2} V_i$  中的点。

对每个点  $v \in V(K)$ , 定义  $S^0(v) = S(v)$ , 若  $v \in \cup_{i \in R_1} V_i$  则  $S^1(v) = S^0(v) \cap f^{-1}(1)$  (即所有颜色为 1 的  $c$  组成的集合), 否则  $S^1(v) = S^0(v) \cap f^{-1}(2)$ 。

于是问题转化为对两个完全  $\frac{r}{2}$  分图用  $S^1(v)$  染色. 设  $s_0 = cr \log m$  为原问题中每个点列表的大小, 不难说明对所有  $v \in V(K)$  几乎有

$$|S^1(v)| \geq \frac{1}{2}s_0 - \frac{1}{2}s_0^{\frac{2}{3}}$$

设  $s_1 = \min_{v \in V(K)} |S^1(v)|$ , 那么

$$s_1 \geq \frac{1}{2}s_0 - \frac{1}{2}s_0^{\frac{2}{3}}$$

我们通过对颜色的折半划分, 将证明  $\chi_\ell(K_{m \times r}) \leq s_0$  规约为证明  $\chi_\ell(K_{m \times \frac{r}{2}}) \leq s_1$ 。

不断重复这一规约过程. 设  $j$  为最小的整数满足  $\frac{r}{2^j} \leq m$ . 我们可以得到一个序列

$$\begin{aligned} s_0 &= cr \log m \\ s_{i+1} &\geq \frac{1}{2}s_i - \frac{1}{2}s_i^{\frac{2}{3}}, \quad \text{for } 1 \leq i < j \end{aligned}$$

当  $c$  充分大时我们可以说明 (这里忽略了一些繁琐的细节)

$$s_j \geq \frac{s_0}{2^{j+1}}$$

为了证明  $\chi_\ell(K_{m \times \frac{r}{2j}}) \leq s_j$ , 注意到  $\frac{r}{2j} \leq m$ , 利用情形一的结论得

$$s_j \geq \frac{s_0}{2^{j+1}} \geq \frac{c}{2} \cdot \frac{r}{2^j} \log m$$

综上, 可以找到充分大的  $c$  使结论成立。□

**推论 1.** 存在正常数  $C_1, C_2$  使得对任意正整数  $m, r \geq 2$  都有

$$C_1 r \log m \leq \chi_\ell(K_{m \times r}) \leq C_2 r \log m$$

证明. 定理 3 的证明对下界也是适用的。□

这一定理有一个有趣的推论。记  $G^c$  表示  $G$  的补图 (Complement Graph)。

**推论 2.** 存在正常数  $C$  使得对任意正整数  $n$ , 存在  $n$  个点的图  $G$  满足

$$\chi_\ell(G) + \chi_\ell(G^c) \leq C \sqrt{n \log n}$$

证明. 设  $m = \sqrt{n \log n}, r = \frac{n}{m} = \sqrt{\frac{n}{\log n}}, G = K_{m \times r}$ 。根据推论 1,  $\chi_\ell(G) = O(r \log m) = O(\sqrt{n \log n})$ 。 $G^c$  是  $r$  个不交的大小为  $m$  的团, 故  $\chi_\ell(G^c) = m = O(\sqrt{n \log n})$ 。所以  $\chi_\ell(G) + \chi_\ell(G^c) = O(\sqrt{n \log n})$ 。□

另一个有趣的研究方向是关于随机图的。例如, 考虑一个经典的模型, 有一张  $n$  个点的无向图  $G_{n, \frac{1}{2}}$ , 其中每条边的出现概率都是  $\frac{1}{2}$  (实际上概率是  $\frac{1}{2}$  还是  $p$  不影响后续推导)。Bollobas 指出图  $G_{n, \frac{1}{2}}$  的色数 [11] 几乎必然 (Almost Surely, 即概率  $\rightarrow 1$  当  $n \rightarrow +\infty$ ) 是

$$\frac{(1 + o(1))n}{2 \log_2 n}$$

$G$  几乎必然不存在大小超过  $2 \log_2 n$  的独立集是一个经典结论。取  $r = \frac{n}{\log_2 n}, m = 2 \log_2 n$ , 那么  $G$  几乎必然可以用  $r$  种颜色染色且每种颜色出现次数不超过  $m$ 。所以  $G$  几乎必然是  $K_{m \times r}$  的子图, 可以直接对  $\chi_\ell(G(n, \frac{1}{2}))$  给出上界

$$\chi_\ell\left(G\left(n, \frac{1}{2}\right)\right) \leq \chi_\ell(K_{m \times r}) = O(r \log m) = O\left(\frac{n \log \log n}{\log n}\right)$$

神奇的是, 我们也能给出精确的估计。

**定理 4.** 图  $G = G_{n, \frac{1}{2}}$  的列表着色数几乎必然是

$$\frac{(1 + o(1))n}{2 \log_2 n}$$





证明。注意下面给出的证明主要目的是展示思路，没有进行严谨的数学证明。

首先根据 Bollobas 的结论， $G$  的列表着色数几乎必然不小于  $\frac{(1+o(1))n}{2\log_2 n}$ 。Bollobas 还说明了  $G$  几乎必然满足任意至少  $\frac{n}{\log^2 n}$  个  $V(G)$  中的点可以选出  $q = 2(1+o(1))\log_2 n$  个点的独立集。

现在考虑对于每个点  $v \in V(G)$  都给出了大小为  $\frac{n}{q} + \frac{n}{\log^2 n} = \frac{(1+o(1))n}{2\log_2 n}$  的集合  $S(v)$ 。一旦某种颜色  $c$  在至少  $\frac{n}{\log^2 n}$  个  $S(v)$  中出现，找到这些  $v$  的一个独立集，将独立集中的点染色为  $c$  并在之后忽略颜色  $c$ 。

最终，有至多  $\frac{n}{q}$  个颜色被移除。那么每个  $S(v)$  包含至少  $\frac{n}{\log^2 n}$  种颜色，且每种颜色出现在至多  $\frac{n}{\log^2 n}$  个  $S(v)$  中。根据 Hall 定理，一定有解。□

笔者仅仅是抛砖引玉，期待读者可以发现概率方法的更多应用场景。

## 5 例题

**例题 1 (染色<sup>1</sup>)**. 有一天，*master* 给了 *pupil* 一张简单图，并且对每个点分别给了一个大小为 2 的颜色集合，*pupil* 只能从这个集合中选一种颜色给这个点染色。*master* 希望 *pupil* 的染色方案使得没有两个有边相连的点被染了相同的颜色。现在 *pupil* 想知道，是否无论 *master* 的颜色集合是什么，他均有办法按照要求染色。

很明显，本题就是要求我们判断给定的图是否是 2-choosable 的。

定义一个连通图的 2-Core 为不断删除度数为 1 的点得到的子图。

**定理 5 ([3])**. 一个简单图是 2-choosable 的，当且仅当它的每个连通块的 2-Core 是孤立点或偶环，或存在源点和汇点使得它们中间恰好有三条长为偶数的不交路径且至少两条路径长度为 2。

证明是进行细致的分类讨论，本文不再重复。感兴趣的读者可以参考原论文。

注：可以发现 2-choosable 的条件已经很复杂了。事实上，3-choosable 问题是 NP Hard 的 [12]。

**例题 2 (彩虹航线<sup>2</sup>)**. 给定二分图  $G = (A \cup B, E)$  和不小于最大度数的整数  $k$ ，每条边给出大小为  $k$  的候选颜色集合，求一种边染色方案。

本题是笔者以列表着色为背景，命制的集训队互测题。

思路同第 3 节内容，证明已经给出了答案的求解方式。

<sup>1</sup>2018 北京省队选拔

<sup>2</sup>2023-2024 集训队互测 Round 16



## 6 总结

本文从列表着色猜想切入，介绍了由这一猜想延伸得到的结论。但限于篇幅，还有一些学术界的研究成果并未提及，感兴趣的读者可以深入了解。

值得一提的是，定理的证明涉及到了多种类型的方法，希望读者可以通过这些方法了解图论领域的常见证明思路。

希望算法竞赛未来可以引入更多有趣的图论模型，拓宽选手的视野。

## 7 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢成都市第七中学蔺洋、林鸿、叶诗富老师对我的指导。

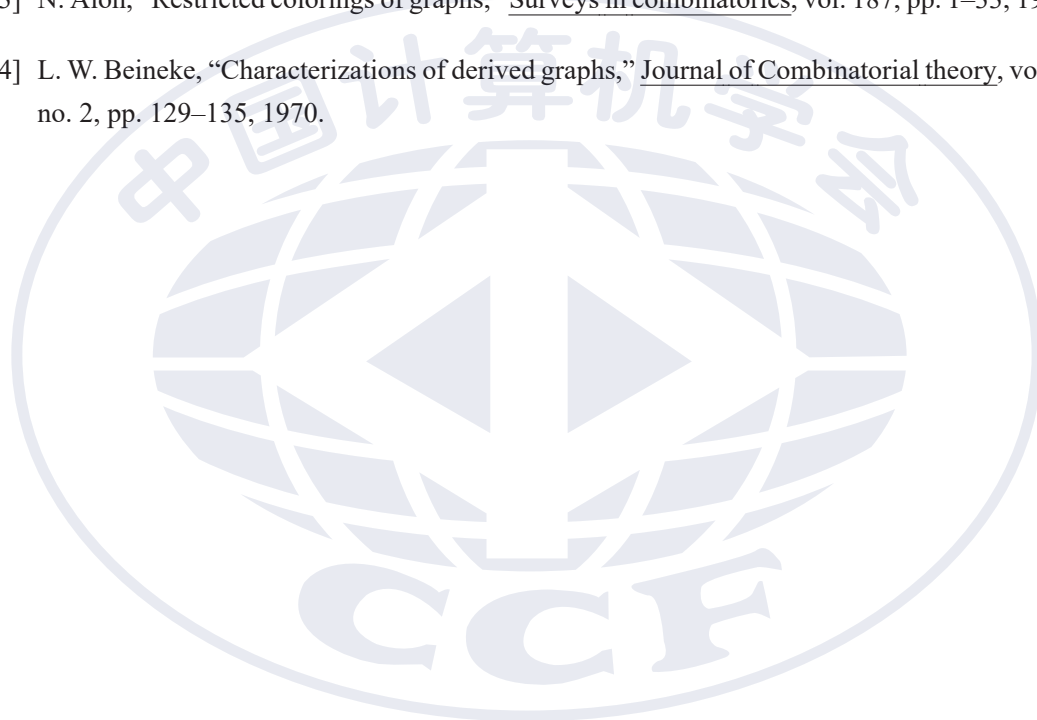
感谢家人、朋友对我的支持与鼓励。

## 参考文献

- [1] P. Erdős, “On a combinatorial problem. 11,” Acta Math. Acad. Sci. Hungar, vol. 15, pp. 445–447, 1964.
- [2] B. Bollobás and A. J. Harris, “List-colourings of graphs,” Graphs and combinatorics, vol. 1, no. 1, pp. 115–127, 1985.
- [3] P. Erdos, A. L. Rubin, and H. Taylor, “Choosability in graphs,” Congr. Numer, vol. 26, no. 4, pp. 125–157, 1979.
- [4] C. Thomassen, “Every planar graph is 5-choosable,” Journal of Combinatorial Theory Series B, vol. 62, no. 1, pp. 180–181, 1994.
- [5] D. Peterson and D. R. Woodall, “Edge-choosability in line-perfect multigraphs,” Discrete mathematics, vol. 202, no. 1-3, pp. 191–199, 1999.
- [6] J. Kahn, “Asymptotics of the list-chromatic index for multigraphs,” Random Structures & Algorithms, vol. 17, no. 2, pp. 117–156, 2000.
- [7] J. Janssen, “The dinitz problem solved for rectangles,” Bulletin of the American Mathematical Society, vol. 29, no. 2, pp. 243–249, 1993.
- [8] F. Galvin, “The list chromatic index of a bipartite multigraph,” Journal of Combinatorial Theory, Series B, vol. 63, no. 1, pp. 153–158, 1995.



- [9] R. Häggkvist and A. Chetwynd, “Some upper bounds on the total and list chromatic numbers of multigraphs,” Journal of graph theory, vol. 16, no. 5, pp. 503–516, 1992.
- [10] N. Alon, “Choice numbers of graphs: a probabilistic approach,” Combinatorics, Probability and Computing, vol. 1, no. 2, pp. 107–114, 1992.
- [11] B. Bollobás, “The chromatic number of random graphs,” Combinatorica, vol. 8, no. 1, pp. 49–55, 1988.
- [12] S. Gutner, “Choice numbers of graphs,” arXiv preprint arXiv:0802.2157, 2008.
- [13] N. Alon, “Restricted colorings of graphs,” Surveys in combinatorics, vol. 187, pp. 1–33, 1993.
- [14] L. W. Beineke, “Characterizations of derived graphs,” Journal of Combinatorial theory, vol. 9, no. 2, pp. 129–135, 1970.



# 由《蚩》出发举例分析观察法在信息学竞赛中的应用

大连市第二十四中学 李嘉佑

## 摘要

观察归纳部分特性或得出结论是解决众多信息学竞赛问题的关键步骤。然而，对于观察法应用技巧的深入研究目前还比较匮乏。为了更为深入地介绍观察法在信息学竞赛中的应用技巧，本文以 IOI2024 中国国家集训队队员互测试题之一《蚩》的解题思路为出发点，通过几道经典例题的解题思路提炼并分析“观察循环节、递推式、通项公式”与“观察有效状态数与具体运算次数”这两种观察角度在信息学竞赛中的应用。最后，本文通过对比这些观察角度的异同点，归纳出不同观察角度的应用场景，体现出观察法巧妙减少比赛中非必要思考、节约解题时间的独特价值。

## 1 《蚩》命题报告

### 1.1 简要题意

给定  $a_1 \dots a_n$ 。两人在玩取石子游戏。

他们面前有  $\sum a_i$  堆石子，大小分别为  $1 \dots a_1, 1 \dots a_2, \dots, 1 \dots a_n$ 。

两人轮流操作，每次从任意一堆石子中（石子堆当前大小记为  $x$ ）删除至多  $\sqrt{x}$  枚石子，无法操作者负。

求先手是否有必胜策略。多测。

### 1.2 数据范围

对于所有测试数据保证： $1 \leq k \leq 5$ ， $1 \leq n$ ， $\sum n \leq 10^5$ ， $1 \leq a_i < 2^{60}$ 。

每个子任务的具体限制见下表：

| 子任务编号 | $n$                       | $k$               | $a_i$          | 特殊性质 | 分值 |
|-------|---------------------------|-------------------|----------------|------|----|
| 1     | $\sum n \leq 10^5$        | $k = 1$           | $a_i < 2^{60}$ |      | 5  |
| 2     | $\sum n \leq 10^5$        | $2 \leq k \leq 5$ | $a_i < 2^{16}$ |      | 5  |
| 3     | $\sum n \leq 10^5$        | $2 \leq k \leq 5$ | $a_i < 2^{22}$ |      | 10 |
| 4     | $\sum n \leq 10^3, n = 2$ | $2 \leq k \leq 3$ | $a_i < 2^{32}$ | A    | 10 |
| 5     | $\sum n \leq 10^3$        | $2 \leq k \leq 5$ | $a_i < 2^{32}$ | A    | 10 |
| 6     | $\sum n \leq 10^5$        | $k = 2$           | $a_i < 2^{60}$ | A    | 10 |
| 7     | $\sum n \leq 10^5$        | $k = 3$           | $a_i < 2^{60}$ | A    | 10 |
| 8     | $\sum n \leq 10^3$        | $k = 2$           | $a_i < 2^{32}$ |      | 10 |
| 9     | $\sum n \leq 10^3$        | $k = 3$           | $a_i < 2^{32}$ |      | 10 |
| 10    | $\sum n \leq 10^5$        | $1 \leq k \leq 5$ | $a_i < 2^{60}$ |      | 20 |

特殊性质 A: 保证  $2 \mid n$ , 且  $a_{2k-1} = a_{2k} - 1$  ( $1 \leq k \leq \frac{n}{2}$ )。

### 1.3 解题思路

#### 1.3.1 记号与约定

下文中, 我们将使用如下符号:

- $\oplus$  表示异或运算;
- $V$  表示  $\max a_i$ ;
- $g(x)$  表示一排  $x$  个棋子的 Grundy value<sup>1</sup>;
- $G(x)$  表示  $g(x)$  的前缀异或和, 即  $\bigoplus_{i=1}^x g(i)$ ;
- $\text{lowbit}(x) = \max 2^k$  ( $k \in \mathbb{N}, 2^k \mid x$ )。

#### 1.3.2 一些特殊情况

$k = 1$  时, 显然  $g(x) = x$ , 此时容易  $\Theta(1)$  单点求出  $G(x)$ 。

下文我们将不再考虑  $k = 1$  的情况, 以规避一些边界问题。

<sup>1</sup>[https://en.wikipedia.org/wiki/Sprague-Grundy\\_theorem](https://en.wikipedia.org/wiki/Sprague-Grundy_theorem)

### 1.3.3 $g(x)$ 的递推式

容易得出一个  $\Theta(V\sqrt[k]{V})$  的算法来预处理出  $g(0) \dots g(V)$ 。  
观察得出的结果，可以注意到：

引理 1.1.

$$g(x) = \begin{cases} \sqrt[k]{x} & \sqrt[k]{x} \in \mathbf{Z} \\ g(x - \lceil \sqrt[k]{x} \rceil) & \sqrt[k]{x} \notin \mathbf{Z} \end{cases}$$

证明. 我们将归纳地给出证明。显然引理对  $g(0), g(1)$  成立。

设  $t > 1$ ，若上式对所有  $g(y)$  ( $0 \leq y < t$ ) 成立，我们将证明上式对  $g(t)$  成立。

记  $S(x) = \{g(y) \mid x - \sqrt[k]{x} \leq y < x\}$ ，记  $r = \lceil \sqrt[k]{t} \rceil$ 。

根据 Sprague-Grundy theorem,  $g(x) = \text{mex } S(x)$ 。

容易证明,  $\forall t - r \leq p < q < t$ ,  $q - p \leq \sqrt[k]{q}$ , 即  $g(p) \neq g(q)$ 。

由于引理对所有  $g(y)$  ( $0 \leq y < t$ ) 成立, 所以  $\forall y < t$ ,  $g(y) < r$ 。

记  $T = \{g(y) \mid t - r \leq y < t\}$ , 则有  $T = [0, r) \cap \mathbf{Z}$ 。

若  $\sqrt[k]{t} \in \mathbf{Z}$ , 则  $S(t) = T$ ,  $g(t) = r$ ;

若  $\sqrt[k]{t} \notin \mathbf{Z}$ , 则  $S(t) = T \setminus \{g(t - r)\}$ ,  $g(t) = g(t - r)$ 。

故引理对  $g(t)$  也成立。 □

### 1.3.4 对 $g(x)$ 进行单点求值

记  $g(r, d) = g(r^k + d)$  ( $0 \leq d < (r+1)^k - r^k$ )。

根据引理 1.1,  $g(r, *)$  有长为  $r+1$  的循环节, 故我们只需要考虑  $d \leq r$  的情况。

针对这种情况, 有:

引理 1.2.

$$g(r, d) = \begin{cases} r & d = 0 \\ d - 1 & 2 \nmid k, 1 \leq d \leq r \\ \lfloor \frac{r}{2^{\text{lowbit}(r)}} \rfloor & 2 \mid k, d = 1 \\ g(r - \lfloor \frac{d}{2} \rfloor, d \bmod 2) & 2 \mid k, 2 \leq d \leq r \end{cases}$$

证明. 由于  $r^k - (r-1)^k \equiv (-1)^{k+1} \pmod{r}$ , 所以有:

$$g(r, d) = \begin{cases} r & d = 0 \\ g(r - 1, (d - 1 + (-1)^{k+1}) \bmod r) & 1 \leq d \leq r \end{cases}$$

尝试递归下去即可。 □



### 1.3.5 对 $G(x)$ 进行单点求值

注意到,  $(x+1)^k - x^k \equiv (-1)^{k+1} \pmod{x+1}$ , 且  $\frac{(x+1)^k - x^k - (-1)^{k+1}}{x+1} \equiv kx \pmod{2}$ , 故有:

$$\bigoplus_{i=x^k}^{(x+1)^k-1} g(i) = \left( [2 \nmid xk] \bigoplus_{i=0}^x i \right) \oplus ([2 \mid k] g(x, x)) \oplus ([2 \nmid k] x)$$

根据上式不难在  $O(\log V)$  的时间复杂度内求出  $\bigoplus_{i=0}^{x^k-1} g(i)$ 。

而  $\bigoplus_{i=x^k}^{x^k+d} g(i)$  ( $d < (x+1)^k - x^k$ ) 也可以在应用一次引理 1.2 后在  $O(\log V)$  的时间复杂度内计算出来。故我们可以在  $O(\log V)$  的时间复杂度内对  $G(x)$  进行单点求值。具体解决方案在官方解题报告 [2] 中进行了详细的讨论。

## 2 观察法

### 2.1 观察法在《蚩》中的作用

事实上, 如果无法注意到引理 1.1 (即  $g$  的循环节与递推式), 本题是几乎无从下手的。而一旦注意到引理 1.1 这一切入点, 就可以从循环节入手来思考如何求解本题。而这一观察是较难以被直接推导所替代的。观察法在本题中得到了非常好的体现。下面我们将对这类可以简化思考或推导过程的观察进行分类举例分析。

### 2.2 观察循环节、递推式或通项公式

**例题 2.1** (Colouring Game<sup>2</sup>). 两人在一初始仅包含 R B 两种字符的字符串上玩游戏。两人轮流操作。先手一次操作可将任意一对包含 R 的相邻字符变为 WW; 后手一次操作可将任意一对包含 B 的相邻字符变为 WW。无法操作者负, 问谁有必胜策略。字符串长度不超过  $5 \times 10^5$ 。

在原串所含 R B 数目不等时, 显然所含字符多者必胜。否则, 两人的最优策略之一必定是优先将相邻的 RB 或 BR 变为 WW, 先不能如此操作者必败。

于是, 问题变为: 给定若干字符串, 两人轮流操作, 一次操作可以从某个串中删掉一对相邻字符并将该串从删去字符的位置分裂成两个字符串, 无法操作者负, 问谁有必胜策略。

我们很难规避掉计算新问题的 Grundy value。设  $g(x)$  表示一个长为  $x$  的字符串在新游戏中的 Grundy value, 则有  $g(x) = \text{mex}_{y+z=x-2} g(y) \oplus g(z)$ 。

<sup>2</sup>Codeforces 1704F



我们看似只能以  $\Omega(n^2)$  的时间复杂度朴素递推计算  $g(1) \dots g(n)$ 。不过，如果我们观察该序列的前若干项，就可以发现其在  $g(100)$  后有长度为 34 的循环节。

对此归纳证明是很容易的。于是，我们得到了一个  $O(n)$  计算  $g(1) \dots g(n)$  的算法。

本题中的观察与《蚱》较为类似，都发现了一些较难直接推导、较易归纳证明的性质，且发现的性质大大优化了计算所需时间复杂度。

**例题 2.2** (六元不定方程<sup>3</sup>)。给定整数  $N, r$ ，求有多少六元有序数组  $(a, b, c, a', b', c')$  满足同余方程  $ab + a'b' \equiv bc + b'c' \equiv ca + c'a' \equiv r \pmod{N}$ ，答案对 998244353 取模。其中  $a, b, c, a', b', c' \in \{0, 1, \dots, N-1\}$ 。  $0 \leq r < N \leq 10^{18}$ ， $N \geq 2$ ， $\mu(N) \neq 0$ 。

不难发现，该问题对  $N$  的所有质因数独立。换句话说，设答案为  $f(N, r)$ ，记  $N$  的质因数分解为  $N = \prod_{i=1}^k p_i$  ( $p_i \in \mathbf{Prime}$ )，则  $f(N, r) = \prod f(p_i, r \bmod p_i)$ 。于是我们只需要快速求出  $N \in \mathbf{Prime}$  时的答案即可。

该问题的官方解法 [3] 是利用 Legendre 符号严谨地推出此时答案的表达式。不过，如果我们观察  $N$  较小时的答案，也很容易不经过任何复杂推导就猜到答案的表达式。下面假设  $n$  是一个质数。

我们首先观察在  $n$  较小时  $f(n, 0) \dots f(n, n-1)$  的列表：

| $n$      | $f(n, 0) \dots f(n, n-1)$     |
|----------|-------------------------------|
| 2        | [20, 8]                       |
| 3        | [73, 28, 28]                  |
| 5        | [489, 124, 124, 124, 124]     |
| 7        | [1009, 344, ..., 344]         |
| $\vdots$ | $\vdots$                      |
| 31       | [89281, 29792, ..., 29792]    |
| 37       | [245161, 50652, ..., 50652]   |
| 41       | [334641, 68920, ..., 68920]   |
| 43       | [238393, 79508, ..., 79508]   |
| 47       | [311329, 103824, ..., 103824] |

首先可以看出， $\forall 1 \leq x < y < n$ ， $f(n, x) = f(n, y)$ 。假设这一结论是正确的，我们就只关心  $f(n, 0)$  与  $f(n, 1)$  了。不难发现  $f(n, 1)$  的取值很接近立方数。更具体地， $f(n, 1) = n^3 + [n \bmod 4 = 3] - [n \bmod 4 = 1]$ 。

此外， $f(n, 0)$  的增长速率也并不高，不妨猜想  $f(n, 0)$  也是  $\Theta(n^3)$  的。且  $f(n, 1)$  的表达式讨论了  $n \bmod 4$  的余数，这启发我们将  $n$  根据  $n \bmod 4$  进行分类。对  $n$  进行分类后，

<sup>3</sup>IOI2023 中国国家集训队互测, d3t3

我们通过差分或插值或将一个比较大的  $f(n, 0)$  视为平衡  $n$  进制数的方式，都可以猜想出

$$f(n, 0) = \begin{cases} 5n^3 - 6n^2 + 3n - 1 & n \equiv 1 \pmod{4} \\ 20 & n = 2 \\ 3n^3 - 3n + 1 & n \equiv 3 \pmod{4} \end{cases}。$$

本题中，我们通过观察发现了一些在小范围内得以验证的简洁性质。虽然未能给出这些性质的证明，但在信息学竞赛中（尤其是一些 IOI 赛制的比赛中）已经足够我们通过本题了。

在信息学竞赛中，观察循环节、递推式或通项公式是一种很有效的规避复杂推导的手段，有时也是解决问题（归纳证明）的必要步骤。而且，这类观察的成本通常较低，故在竞赛中是一种比较值得尝试的方法。

在这部分观察中，比较常见的手段包括但不限于：

- 观察答案级别：
  - 若  $\forall n, f(n) = \Theta(1)$  则可能存在循环节；
  - 若  $\forall n, f(n) = \Theta(\text{poly } n)$  则  $f(n)$  可能是关于  $n$  的多项式；
  - 否则，答案可能是指数级别，且可能可以用一个递推式表示出来。
- 观察数列的差分或前缀和；
- 观察数列与二项式系数或斯特林数的关系（特别是，观察其与杨辉三角的关系）；
- Berlekamp-Massey algorithm；
- 拉格朗日插值。

## 2.3 观察有效状态数或具体运算次数

**例题 2.3** (移除石子<sup>4</sup>)。给定  $n, k, l_{1..n}, r_{1..n}$ 。求有多少序列  $a_{1..n}$ ，满足  $\forall i, l_i \leq a_i \leq r_i$ ，以及如下条件：假设初始时有  $n$  堆石子，大小分别为  $a_1 \dots a_n$ ；则存在一种向其中一些石子堆中再添加恰好共  $k$  枚石子的方案，使得你可以通过“移除一堆石子中的至少两枚石子”“从一个长度不少于三的区间中的每个石子堆分别移除一枚石子”这两种操作将所有石子移除。多测。  $T \leq 10, n \leq 10^3, k \leq 100, 0 \leq l_i \leq r_i \leq 10^9$ 。

首先考虑  $l_i = r_i$  的情况，即已知  $a_{1..n}$ ，判断其是否满足条件。我们不难发现，除  $a = [0, 0, \dots, 0]$  以及  $a = [1, 1, 1]$  这两个特例外，存在一种添加恰好共  $k$  枚石子的方案，等价于存在一种添加至多共  $k$  枚石子的方案。这一结论在参考文献 [5] 中进行了较为详细的证明。

<sup>4</sup>第 39 届全国青少年信息学奥林匹克竞赛 (NOI 2022), d1t2



于是,我们可以设  $f_{i,x,y}$  表示考虑  $a_{1\dots i}$ , 有  $x$  个区间至少包含  $i+1$ ,  $y$  个区间至少包含  $i+2$ , 最少需要在  $a_{1\dots i}$  中添加多少石子。此时经过一些分析或实验可以发现,我们只需要考虑  $x < 3, y < 3$  的状态(其余状态均可调整为  $x < 3, y < 3$  的状态)。

此时,我们设  $g_{i,v[0\dots 2][0\dots 2]}$  表示  $f_{i,x,y} = v_{x,y}$  的  $a_{1\dots i}$  的数目,则可以用 DP of DP 的方式解决本题。可能的  $v_{0\dots 2,0\dots 2}$  看上去很多(应当有  $\Omega(k^9)$  种),不过,我们可以通过实验发现,事实上可能的  $v_{0\dots 2,0\dots 2}$  只有不超过  $10^4$  种,足以通过本题。(这是可以理解的,因为直观上这九个元素之间不会相差很多。)这一解法也是本题最为经典的解法。

在这道题目中,我们在得到  $l_i = r_i$  时的解法后,发现其难以扩展到  $l_i \neq r_i$  的情况。此时,我们通过观察发现状态数并不多,于是得到了一个 DP of DP 的解法。这提醒我们,在对状态有较严格的限制时,有效状态数可能远低于总状态数。在这种情况下,观察有效状态数可能会为我们提供一些新的思路。

**例题 2.4 (Kangaroo<sup>5</sup>)**. 求有多少  $1\dots n$  的排列  $p_{1\dots n}$ , 使得  $p_1 = s, p_n = t$ , 且  $\forall 1 < i < n, [p_i < p_{i-1}] = [p_i < p_{i+1}]$ 。答案对  $10^9 + 7$  取模。 $n \leq 2 \times 10^3$ , 时间限制 1s。

该问题的经典解法 [1] 是一个时间复杂度  $O(n^2)$  的动态规划。这里将介绍一种时间复杂度  $O(n^3)$  的解法,并分析其最坏情况下的运算次数。

考虑以 1 在排列中的位置  $x$  为分界,将排列拆成  $p_{1\dots x}$  与  $p_{x\dots n}$  两部分。

设  $f_{i,j}$  表示有多少  $1\dots i$  的排列  $p_{1\dots i}$ , 满足  $p_1 = 1, p_i = j$ , 且  $\forall 1 < x < i, [p_x < p_{x-1}] = [p_x < p_{x+1}]$ 。 $f_{1\dots n,1\dots n}$  是比较好求以  $\Theta(n^2)$  的时间复杂度求出的。

接下来,不妨设  $s < t$ , 则我们只关心  $(1, s), (s, t), (t, n)$  三个区间内分别有多少值在排列中位于 1 左侧。直接枚举这三个量,即可用求得的  $f$  数组计算出答案,得到一种  $O(n^3)$  的解法。

此时,可以注意到,这三个量的和不会超过  $2 \times 10^3$ 。于是我们发现,这一解法是通过本题的,运算次数约为  $\left(\frac{2000}{3}\right)^3$  级别,虽然其复杂度偏高。

**例题 2.5 (序列<sup>6</sup>)**. 给定  $P, q$ 。  $q$  次给定  $n, m$ , 求满足  $a_i \in [0, m]$  且  $\nexists 1 \leq i < j < k \leq n, a_k < a_i < a_j$  的序列  $a_{1\dots n}$  的数目。答案对  $P$  取模。 $n \leq 100, m \leq 8 \times 10^4$ 。

令  $N$  为  $n$  的最大值,该问题的经典解法 [4] 的时间复杂度为  $O(N^4 + qN)$ 。其瓶颈之一在于利用动态规划求出  $f_{0\dots N,0\dots N}$ , 其中  $f_{i,j}$  表示长为  $i$ 、元素是  $[0, j]$  中整数的满足条件的序列数目。这一数组的经典求法对序列的形态进行了分析,这里将给出一种无需分析性质的  $\Theta(N^5)$  求出  $f_{0\dots N,0\dots N}$  的方式,并分析其最坏情况下的运算次数。

设  $g_{i,j,k}$  表示长为  $j$ , 元素是  $0\dots i$  中整数(且要求  $0\dots i$  均出现过), 且恰有  $k$  个位置可以插入  $i+1$  的满足条件的序列数目。考虑枚举第一个  $i+1$  插在第几个可以插入的位置,再枚举在除序列末尾外的其他位置插入了多少个  $i+1$ , 即可利用差分及前缀和进行优化,在  $\Theta(N^5)$  的时间复杂度内求出  $g_{0\dots N,0\dots N,0\dots N}$ , 从而求出  $f_{0\dots N,0\dots N}$ 。

<sup>5</sup>The 23rd Central European Olympiad in Informatics (CEOI 2016), d1t2

<sup>6</sup>IOI2024 中国国家集训队互测, d2t1



此时可以发现, 由于我们对状态的限制较为严格, 绝大部分  $g_{i,j,k}$  的值为 0。而我们只需要对  $g_{i,j,k} \neq 0$  的状态进行转移。经过实际测试, 我们发现, 这一解法在  $\Theta(N^5)$  瓶颈处只需进行不超过  $2.6 \times 10^8$  次运算。而其他部分的时间复杂度均为  $O(N^4 + qN)$ , 足以通过本题。

在这两道题目中, 我们都使用了复杂度偏高的解法, 却均能在原数据范围内的任意测试数据下通过。这启发我们, 解法的效率并非只与时间复杂度有关。受限于现代计算机的运算能力, 有时我们无法区分“高复杂度小常数”解法与“低复杂度大常数”解法之间的效率差异。观察解法所需的具体运算次数有时可以帮助我们利用一些复杂度偏高的解法通过一些题目, 而这在信息学竞赛中很可能为我们节约大量的思考时间。

让我们试着分析一下观察有效状态数与观察具体运算次数的异同。状态数较少通常会导致运算次数也很少, 且大部分情况下状态数的最大值与运算次数的最大值都可以通过实验得出。从这个角度看, 这两种观察是相似的。不过, 观察有效状态数更常作为必要的解题步骤, 而观察具体运算次数更常用来优化一些复杂度偏高的解法。同时, 我们通常会在解题的过程中观察状态数, 而在一些复杂度偏高的算法成型后观察其具体运算次数。从这个角度看, 这两种观察又有一定的区别。换句话说, 这两种观察在本质上类似, 只是常见应用场景有所不同。

## 2.4 总结

一般来说, 在一些关键信息可以写成数列或矩阵的形式时, 我们可以试着观察其循环节、递推式或通项公式; 而观察有效状态数、具体运算次数则常常在我们尝试优化一些较差的解法时进行。两种观察都在尝试减少我们所关心的信息量: 前者尝试用循环节、递推式或通项公式来直接表示一个较复杂的序列或矩阵; 后者尝试直接抛弃掉一些我们不关心的状态, 或跳过一些不必要的运算。

从某种角度来说, 观察法常常用于我们对问题的解法已经有了大致猜想的情况。这种情况下, 我们可以用观察法来解决一些我们在解题过程中遇到的细节问题, 如: 若某序列比较难以求出, 则考虑观察循环节等; 若总状态数偏多, 则考虑观察有效状态数; 若复杂度偏高, 则考虑观察具体运算次数。

## 3 总结与展望

观察或猜想在信息学竞赛题目中常常作为必要步骤出现。在观察或猜想并非必要步骤的题目中, 比较巧妙的观察或猜想有时也能为我们打开思路, 或者帮助我们获得一些分数。而且, 信息学竞赛有样例、可以“对拍”、很少要求证明的特性, 使得观察法常常可以极低的成本获得极高的收益。

本文尝试着总结了两类常见的观察对象。笔者希望借此将观察、猜想的方法进行初步系统化, 让我们在解决相关问题时可以更容易地发现一些性质。同时, 也希望本文可以起



到抛砖引玉的作用，吸引更多读者对如何有技巧地观察、如何有章法地思考进行探究。

## 4 致谢

感谢父母的培育与支持；

感谢张新钢老师对我的教导；

感谢中国计算机学会提供学习和交流的平台；

感谢王艺霏、牛俊博、张力文、范绪杰、彭思进、苏泓燃为本文验稿并提出宝贵意见；

感谢尤俊翔、艾恒宇、杜茂森对本文结构提出的宝贵意见；

感谢王孜研、肖子尧、江城为本文提供的宝贵意见。

## 参考文献

- [1] Adrian Panaete and Ionel-Vasile Pit-Rada. ceoi 2016 kangaroo solution, 2016.
- [2] 李嘉佑. ioi2024 中国国家集训队作业《蚩》解题报告, 2023.
- [3] 柯绎思. ioi2023 中国国家集训队作业《六元不定方程》解题报告, 2022.
- [4] 梁城玮. ioi2024 中国国家集训队作业《序列》解题报告, 2023.
- [5] 魏佳泽. p8497 [noi2022] 移除石子, 2022.





# 浅谈 $q$ -analogue

湖南省长沙市第一中学 李铭乐洋

## 摘要

本文从  $q$ -模拟出发, 围绕 statistic 展开, 介绍一些经典问题的  $q$ -模拟。在此过程中介绍了多例经典的双射构造。

## 目录

### 1 引言

在组合数学中, 计数函数的  $q$ -analogue 通常是  $q$  的多项式。当  $q = 1$  时, 该多项式简化为所讨论的函数。

近些年来, 与  $q$ -模拟有关的题目不断地在算法竞赛中出现。但笔者注意到, 目前大多数选手对  $q$ -模拟的认知与应用仍局限于处理矩阵的秩相关问题, 与  $q$ -模拟有关的题目也主题单一。

因此, 笔者撰写此文, 旨在向读者展示  $q$ -模拟更丰富的内容, 也希望读者可以通过此文深入对  $q$ -模拟的了解。

本文在第一节中, 对基本的  $q$ -模拟给出了定义, 主要介绍了与组合数有关的  $q$ -模拟。在第二节中, 介绍了  $q, t$ -卡特兰数。第三节中, 介绍了对称函数与  $q$ -复合的相关模型。

### 2 约定

出于严谨性问题考虑, 本文中出现的名词多使用英文原文。

### 3 基本 $q$ -analogues

#### 3.1 $q$ -factorial 与排列

在  $q$ -analogue 中, 我们有时将  $q$  视作满足  $0 < q < 1$  的实参数。

**定义 3.1.** 对于实数  $x$ , 我们定义其  $q$ -analogue 为:

$$[x] = \frac{1 - q^x}{1 - q} \quad (1)$$

根据洛必达法则, 当  $q \rightarrow 1$  时  $[x] \rightarrow x$ 。

**定义 3.2.** 对于正整数  $n$ , 我们定义  $n!$  的  $q$ -analogue 为:

$$[n]! = \prod_{i=1}^n [i] = \prod_{i=1}^n \frac{1 - q^i}{1 - q} \quad (2)$$

即是  $q$ -factorial。

接下来我们来研究  $[n]!$  与排列的关系。对于长度为  $n$  的序列  $\pi$  我们定义:

**定义 3.3.** 我们记  $\text{inv}(\pi)$  为满足  $1 \leq i < j \leq n, \pi_i > \pi_j$  的  $(i, j)$  的对数。即是  $\pi$  的 **inversion statistic**。

**定义 3.4.** 我们记  $\text{maj}(\pi)$  为满足  $1 \leq i < n, \pi_i > \pi_{i+1}$  的所有  $i$  的和 (我们称这些位置为 **descent**)。即是  $\pi$  的 **major index statistic**。

当  $\pi$  取遍所有  $n$  阶排列时, 我们有:

**定理 3.1.**

$$\sum_{\pi} q^{\text{inv}(\pi)} = [n]! = \sum_{\pi} q^{\text{maj}(\pi)} \quad (3)$$

证明. 考虑归纳证明两个等号。

对于  $\sum_{\pi} q^{\text{inv}(\pi)} = [n]!$ 。考虑任一  $n-1$  阶排列  $\pi'$  以及一个  $[1, n]$  之间的正整数  $k$ , 我们可以通过向  $\pi'$  中第  $k$  个数前侧加入  $n$  得到一  $n$  阶排列  $\pi$ , 并将这个结果记做  $\pi'(k)$ 。不难发现合法的  $\pi'(k)$  和  $n$  阶排列  $\pi$  一一对应。

而这里有  $\text{inv}(\pi) = \text{inv}(\pi') + (k-1)$ 。因此可以得到:

$$\sum_{\pi} q^{\text{inv}(\pi)} = \sum_{\pi'} q^{\text{inv}(\pi')} (1 + q + q^2 + \cdots + q^{n-1}) = [(n-1)]! \cdot [n] = [n]! \quad (4)$$

对于  $\sum_{\pi} q^{\text{maj}(\pi)} = [n]!$ 。考虑任一  $n-1$  阶排列  $\pi'$ , 我们找到所有 descent  $i_1 < i_2 < \cdots < i_s$ 。然后, 我们发现:

$$\text{maj}(\pi'(n)) = \text{maj}(\pi') \quad (5)$$

$$\text{maj}(\pi'(i_k + 1)) = \text{maj}(\pi') + (s - k + 1) \quad (6)$$

$$\text{maj}(\pi'(1)) = \text{maj}(\pi') + s + 1. \quad (7)$$

接着对于小于  $n-1$  的剩下的位置  $j_1 < j_2 < \cdots < j_{n-2-s}$ , 有:

$$\text{maj}(\pi'(j_1 + 1)) = \text{maj}(\pi') + s - (j_1 - 1) + j_1 + 1 = \text{maj}(\pi') + s + 2 \quad (8)$$

此处,  $s - (j_1 - 1)$  是  $j_1$  后侧的 descent 的个数, 它们将后移一位。同时,  $j_1 + 1$  这个位置有了新的 descent。

将上面的结果接着扩展可以得到:

$$\text{maj}(\pi'(j_k + 1)) = \text{maj}(\pi') + s - (j_k - k) + j_k + 1 = \text{maj}(\pi') + s + (k + 1) \quad (9)$$

于是, 我们同样得到了:

$$\sum_{\pi} q^{\text{maj}(\pi)} = \sum_{\pi'} q^{\text{maj}(\pi')} (1 + q + q^2 + \cdots + q^{n-1}) = [(n-1)!] \cdot [n] = [n]! \quad (10)$$

□

接下来我们介绍关于定理 3.1 的另一种证明。我们构造映射  $\pi \rightarrow \phi(\pi)$ , 具体过程如下:

对于  $n \leq 2$ , 令  $\phi(\pi) = \pi$ 。

接下来我们一个一个地将数字填入  $\phi(\pi)$ 。最开始设  $\phi^{(1)}(\pi) = \pi_1, \phi^{(2)}(\pi) = \pi_1 \pi_2$  (此处表示拼接)。然后, 对于  $3 \leq k \leq n$  的  $\phi^{(k)}$ , 若  $\pi_{k-1} > \pi_k$ , 则在  $\phi^{(k-1)}$  中所有大于  $\pi_k$  的元素后侧划线 (同时在 1 前侧划线)。否则, 有  $\pi_{k-1} < \pi_k$ , 则在  $\phi^{(k-1)}$  中所有小于  $\pi_k$  的元素后侧划线 (同时在 1 前侧划线)。

对于任意相邻两条线之间的段, 我们称做 block。我们令  $\phi^{(k)}$  为将  $\phi^{(k-1)}$  的所有 block 实行“将最后一个元素移动到最前面”的操作后, 再在最后加入  $\pi_k$  的结果。

结果即是  $\phi^{(n)}$ 。

**命题 3.1.**  $\text{maj}(\pi) = \text{inv}(\phi(\pi))$

证明. 考虑归纳证明。对于  $k \leq 2$  的  $\phi^{(k)}$  的  $\text{inv}(\phi^{(k)})$  与  $\pi$  中 index 严格小于  $k$  的 descent 的 index 之和总是相等。而对于  $3 \leq k \leq n$ , 在  $k-1$  满足要求的情况下, 有如下两种情况:

若  $\pi_{k-1} > \pi_k$ , 则 descent  $k-1$  将被计算。而考虑一个 block (设其大小为  $x$ ), 最后侧的元素会贡献一个 inversion。将其移动到 block 中最前侧后, 又会贡献  $x-1$  个 inversion。因此总计贡献  $k-1$  个 inversion。

若  $\pi_k < \pi_{k-1}$ , 则 descent 不变。而考虑一个 block (设其大小为  $x$ ), 最后侧往前移, 会导致贡献变少  $x-1$ 。而 block 中剩下  $x-1$  个元素都会与  $\pi_k$  产生 inversion。于是贡献抵消了。

因此, 每轮加数后, 上述两个量总是相同的。

显而易见, 我们很容易设计  $\phi$  的逆映射  $\phi^{-1}$  满足  $\phi^{-1}(\phi(\pi)) \rightarrow \pi$ 。于是  $\pi, \phi(\pi)$  是一一对应的, 因此定理 3.1 再次得证。□

### 3.2 Gaussian polynomial

我们记 **partition** 为序列  $\lambda$  满足  $\lambda_1 \geq \lambda_2 \geq \dots$ , 接着记  $\ell(\lambda)$  为最大的  $x$  使得  $\lambda_x > 0$ , 然后记  $|\lambda|$  为  $\sum \lambda_i$ 。

对于一个 partition 定义其 **Ferrers graph** 为: 共  $\ell(\lambda)$  行, 从下往上第  $i$  行长度为  $\lambda_i$ , 且所有行均左对齐的图形。接着, 我们定义一个 partition  $\lambda$  的 **conjugate partition**  $\lambda'$  为: 以  $\lambda$  的 Ferrers graph 沿  $y = x$  翻转后得到的 Ferrers graph 还原出的 partition。

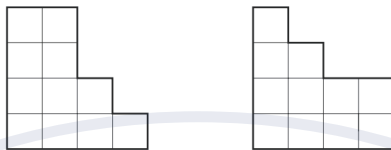


图 1: conjugate partition 示例

上图左侧是 partition  $(4, 3, 2, 2)$  的 Ferrers graph。而右侧即是其 conjugate partition  $(4, 3, 2, 2)' = (4, 4, 2, 1)$ 。

**定义 3.5.** 对于  $n, k \in \mathbb{N}, 0 \leq k \leq n$  定义 **Gaussian polynomial** 为:

$$\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n \\ k \end{bmatrix}_q = \frac{[n]!}{[n-k]![k]!} = \frac{(1-q^n)(1-q^{n-1})\dots(1-q^{n-k+1})}{(1-q^k)(1-q^{k-1})\dots(1-q)} \quad (11)$$

这是  $q$ -**binomial coefficients** 的特殊情况。而后者有:

**定义 3.6.** 对于  $x \in \mathbb{R}$  定义  $q$ -binomial coefficients 为:

$$\begin{bmatrix} x \\ k \end{bmatrix} = \frac{(q^{x-k+1}; q)_k}{(q; q)_k} \quad (12)$$

其中  $(a; q)_k = (a)_k = (1-a)(1-qa)\dots(1-q^{k-1}a)$ , 被称做  $q$ -**rising factorial**。

而事实上 Gaussian polynomial 是  $q$  中的多项式, 因为我们有:

**定理 3.2.** 对于  $n, k \in \mathbb{N}$  有:

$$\begin{bmatrix} n+k \\ k \end{bmatrix} = \sum_{\lambda \subseteq n^k} q^{|\lambda|} \quad (13)$$

其中  $\lambda \subseteq n^k$  即  $\lambda_1 \leq n, \ell(\lambda) \leq k$ 。

证明. 考虑归纳。我们令  $P(n, k) = \sum_{\lambda \subseteq n^k} q^{|\lambda|}$ , 即有:

$$P(n, k) = \sum_{\lambda \subseteq n^k, \lambda_1 = n} q^{|\lambda|} + \sum_{\lambda \subseteq n^k, \lambda_1 < n} q^{|\lambda|} = q^n P(n, k-1) + P(n-1, k) \quad (14)$$

边界条件  $P(n, 0) = P(0, k) = 1$  是显而易见的。

另一方面：

$$q^n \begin{bmatrix} n+k-1 \\ k-1 \end{bmatrix} + \begin{bmatrix} n-1+k \\ k \end{bmatrix} = q^n \frac{[n+k-1]!}{[k-1]![n]!} + \frac{[n+k-1]!}{[k]![n-1]!} \quad (15)$$

$$= \frac{[n+k-1]!}{[k]![n]!} \left( q^n \frac{1-q^k}{1-q} + \frac{1-q^n}{1-q} \right) \quad (16)$$

$$= \frac{[n+k]!}{[k]![n]!} \quad (17)$$

□

令  $\alpha = (\alpha_0, \dots, \alpha_s) \in \mathbb{N}^{s+1}$ 。我们记由  $\alpha_0$  个 0、 $\alpha_1$  个 1 ...  $\alpha_s$  个  $s$  组成的多重集为  $\{0^{\alpha_0} 1^{\alpha_1} \dots s^{\alpha_s}\}$ 。再记  $n = \sum \alpha_i$  表示可重集大小。而对于其我们有  $q$ -multinomial coefficient 即：

$$\begin{bmatrix} n \\ \alpha_0, \dots, \alpha_s \end{bmatrix} = \frac{[n]!}{[\alpha_0]! \dots [\alpha_s]!} \quad (18)$$

接下来我们进一步扩展之前通过研究排列得到的性质：

**定义 3.7.** 我们有：

$$\sum_{\sigma \in M_a} q^{\text{inv}(\sigma)} = \begin{bmatrix} n \\ \alpha_0, \dots, \alpha_s \end{bmatrix} = \sum_{\sigma \in M_a} q^{\text{maj}(\sigma)} \quad (19)$$

其中  $M_a$  表示多重集  $\{0^{\alpha_0} 1^{\alpha_1} \dots s^{\alpha_s}\}$  的排列形成的集合。

证明. 对于  $\sigma \in M_a$  我们生成排列：将  $\sigma$  中所有的 0 按照 index 从小到大用  $0, \dots, \alpha_0 - 1$  替换，将所有 1 按照 index 从小到大用  $\alpha_0, \dots, \alpha_0 + \alpha_1 - 1$  替换... 由此得到的排列记做  $\pi$ 。

此处的证明与定理 3.1 的情况是基本相同的。以我们对定理 3.1 的第二种证明用到的映射  $\pi \rightarrow \phi(\pi)$  的说明为例：在  $\phi^{(k-1)} \rightarrow \phi^{(k)}$  时，所有  $> \pi_k$  的元素相对位置不变，所有  $< \pi_k$  的元素相对位置也不变。因此，在  $\phi(\pi)$  中， $0, \dots, \alpha_0 - 1$  的出现 index 是递增的， $\alpha_0, \dots, \alpha_0 + \alpha_1 - 1$  的出现 index 也是递增的。由此， $\phi(\pi)$  也唯一对应了一个  $M_a$  中的元素。□

### 3.3 The Catalan numbers and Dyck paths

我们记所有  $(0, 0)$  到  $(n, m)$  的格路集合为  $L_{n,m}$ ，以及  $L_{n,m}^+ \subseteq L_{n,m}$  为那些永远不低于  $y = \frac{n}{m}x$  的格路。对于任意的  $\pi \in L_{n,m}^+$  我们称  $\pi$  是一个 **Dyck path**，又称作 **Catalan path**。

我们记  $C_n = \frac{1}{n+1} \binom{2n}{n}$  为 **Catalan number**（即 Catalan 数）。Catalan 数的一种组合解释是  $L_{n,n}^+$  中的元素个数，即  $|L_{n,n}^+| = C_n$ 。

对于  $\pi \in L_{n,n}^+$ ，我们给出如下定义：

**定义 3.8.** 记  $\sigma(\pi)$  为：初始为空，从  $(0,0)$  开始，每走一步 North（即  $(1,0)$ ）就往后加入字符 0，每走一步 East（即  $(0,1)$ ）就往后加入字符 1。最终走到  $(n,n)$  结束时得到的序列。

**定义 3.9.** 记  $a_i(\pi)$  为：第  $i$  行中，在  $\pi$  右侧、在  $y = x$  左侧的完整格子个数。这里我们还将  $a_i(\pi)$  称做  $\pi$  的第  $i$  行的长度。同时， $\pi$  的 **area statistic** 被定义为  $\sum a_i(\pi)$ （记做  $\text{area}(\pi)$ ）。

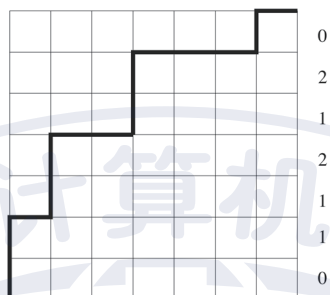


图 2: area statistic 示例

上图每行右侧的值即为对应的  $a_i(\pi)$ 。

接下来我们给出  $C_n$  的一个  $q$ -analogue:

**定理 3.3.** 我们有:

$$\sum_{\pi \in L_{n,n}^+} q^{\text{maj}(\sigma(\pi))} = \frac{1}{[n+1]} \begin{bmatrix} 2n \\ n \end{bmatrix} \quad (20)$$

证明. 考虑所有非法路径  $L_{n,n}^- = L_{n,n} \setminus L_{n,n}^+$ 。对于任一  $\pi \in L_{n,n}^-$ ，我们找到其经过的格点中， $j-i$  最小的（如有多个，选择最早的一个），记为  $P$ 。

那么到达  $P$  的一步一定是 East。将这一步 East 变成 North，后面的部分平移，得到了一个  $\pi' \in L_{n-1,n+1}$ 。可以看到此时  $\text{maj}(\sigma(\pi'))$  相比  $\text{maj}(\sigma(\pi))$  少了 1。

事实上，对于任一  $\pi' \in L_{n-1,n+1}$ ，只要找到  $j-i$  最小的点（如有多个，选择最晚的一个）。下一步必然是 North，将其改为 East，后面的部分平移即可。

那这样  $L_{n-1,n+1}$  和  $L_{n,n}^-$  中的路径就是一一对应的了，即有:

$$\sum_{\pi \in L_{n,n}^-} q^{\text{maj}(\sigma(\pi))} = \sum_{\pi' \in L_{n-1,n+1}} q^{\text{maj}(\sigma(\pi'))+1} = q \begin{bmatrix} 2n \\ n-1 \end{bmatrix} \quad (21)$$



于是就有：

$$\sum_{\pi \in L_{n,n}^+} q^{\text{maj}(\sigma(\pi))} = \sum_{\pi \in L_{n,n}} q^{\text{maj}(\sigma(\pi))} - \sum_{\pi \in L_{n,n}^-} q^{\text{maj}(\sigma(\pi))} \quad (22)$$

$$= \begin{bmatrix} 2n \\ n \end{bmatrix} - q \begin{bmatrix} 2n \\ n+1 \end{bmatrix} \quad (23)$$

$$= \frac{1}{[n+1]} \begin{bmatrix} 2n \\ n \end{bmatrix} \quad (24)$$

□

然后来看另一个  $C_n$  的  $q$ -analogue。

**定义 3.10.**

$$C_n(q) = \sum_{\pi \in L_{n,n}^+} q^{\text{area}(\pi)} \quad (25)$$

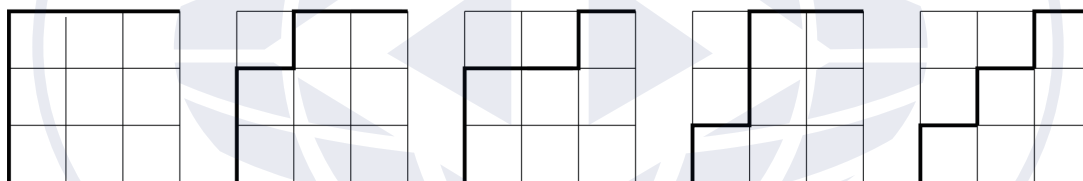


图 3: 从左到右 area 依次为 3, 2, 1, 1, 0

如上图可以得到  $C_3(q) = 1 + 2q + q^2 + q^3$ 。

其存在递归关系：

**命题 3.2.** 我们有：

$$C_n(q) = \sum_{k=1}^n q^{k-1} C_{k-1}(q) C_{n-k}(q), n \geq 1 \quad (26)$$

证明. 考虑  $(0,0)$  出发后, 第一次碰到  $y=x$  的位置为  $(k,k)$ 。那么  $(0,1)$  到  $(k-1,k)$  的路径可以被  $q^{k-1}C_{k-1}(q)$  描述,  $(k,k)$  到  $(n,n)$  的路径可以被  $C_{n-k}(q)$  描述。□

### 3.4 The $q$ -Vandermonde convolution

我们记下式为 基本超几何级数 (basic hypergeometric series):

$${}_p\phi_p \left( \begin{matrix} a_1, a_2, \dots, a_{p+1} \\ b_1, \dots, b_p \end{matrix}; q; z \right) = \sum_{k=0}^{\infty} \frac{(a_1)_k \cdots (a_{p+1})_k}{(q)_k (b_1)_k \cdots (b_p)_k} z^k \quad (27)$$

下面的结果被称为 Cauchy's  $q$ -binomial series :

**定理 3.4.** 在  $|z| < 1, |q| < 1$  时我们有:

$${}_1\phi_0 \left( \begin{matrix} a \\ - \end{matrix}; q; z \right) = \sum_{k=0}^{\infty} \frac{(a)_k}{(q)_k} z^k = \frac{(az)_{\infty}}{(z)_{\infty}} \quad (28)$$

其中  $(a; q)_{\infty} = (a)_{\infty} = \prod_{i=0}^{\infty} (1 - aq^i)$ 。

证明. 令:

$$F(a, z) = \sum_{k=0}^{\infty} \frac{(a)_k}{(q)_k} z^k \quad (29)$$

可以得到:

$$F(a, z) - F(a, qz) = \sum_{k=0}^{\infty} \frac{(a)_k}{(q)_k} z^k (1 - q^k) = (1 - a)zF(aq, z) \quad (30)$$

以及:

$$F(a, z) - F(aq, z) = -azF(aq, z) \quad (31)$$

合并可得:

$$F(a, z) = \frac{(1 - az)}{(1 - z)} F(a, qz) \quad (32)$$

我们不断进行上述代换过程。当进行次数  $n \rightarrow \infty$  时:

$$F(a, z) = \lim_{n \rightarrow \infty} \frac{(az; q)_n}{(z; q)_n} F(a, q^n z) \quad (33)$$

$$= \frac{(az; q)_{\infty}}{(z; q)_{\infty}} F(a, 0) \quad (34)$$

$$= \frac{(az; q)_{\infty}}{(z; q)_{\infty}} \quad (35)$$

□

当我们将  $a$  替换为  $q^a$  并令  $q \rightarrow 1$  就得到了:

$$\sum_{n=0}^{\infty} \frac{a(a+1) \cdots (a+n-1)}{n!} z^n = \frac{1}{(1-z)^a} \quad (36)$$

接下来是几个推论:

**推论 3.1.** 以下的结果被称为  $q$ -binomial theorem :

$$\sum_{k=0}^n q^{\binom{k}{2}} \begin{bmatrix} n \\ k \end{bmatrix} z^k = (-z; q)_n \quad (37)$$

$$\sum_{k=0}^{\infty} \begin{bmatrix} n+k \\ k \end{bmatrix} z^k = \frac{1}{(z; q)_{n+1}} \quad (38)$$

证明. 证明第一个等式, 只需要往定理 3.4 中带入  $a = q^{-n}, z = -zq^n$ :

$$\sum_{k=0}^{\infty} \frac{(q^{-n})_k}{(q)_k} (-zq^n)^k = \frac{(-z)_{\infty}}{(-zq^n)_{\infty}} = (-z)_n \quad (39)$$

这是因为事实上我们有反转公式:

$$\begin{bmatrix} n \\ k \end{bmatrix} = (-1)^k q^{kn - \binom{k}{2}} \begin{bmatrix} k - n - 1 \\ k \end{bmatrix} \quad (40)$$

只需要考虑后者即可。

而证明第二个等式, 只需要往定理 3.4 中带入  $a = q^{n+1}$ :

$$\sum_{k=0}^{\infty} \frac{(q^{n+1})_k}{(q)_k} z^k = \frac{(q^{n+1}z)_{\infty}}{(z)_{\infty}} = \frac{1}{(z)_{n+1}} \quad (41)$$

□

对于任意函数  $f(z)$ , 记  $f(z)|_{z^k}$  为其麦克劳林级数中  $z^k$  的系数。

**推论 3.2.** 我们有:

$$\sum_{k=0}^h q^{(n-k)(h-k)} \begin{bmatrix} n \\ k \end{bmatrix} \begin{bmatrix} m \\ h-k \end{bmatrix} = \begin{bmatrix} m+n \\ h \end{bmatrix} \quad (42)$$

证明. 根据  $q$ -binomial theorem 有:

$$q^{\binom{h}{2}} \begin{bmatrix} m+n \\ h \end{bmatrix} = \prod_{k=0}^{m+n-1} (1 + zq^k)|_{z^h} \quad (43)$$

$$= \prod_{k=0}^{n-1} (1 + zq^k) \prod_{j=0}^{m-1} (1 + zq^n q^j)|_{z^h} \quad (44)$$

$$= \left( \sum_{k=0}^n q^{\binom{k}{2}} \begin{bmatrix} n \\ k \end{bmatrix} z^k \right) \left( \sum_{j=0}^m q^{\binom{j}{2}} \begin{bmatrix} m \\ j \end{bmatrix} (zq^n)^j \right) |_{z^h} \quad (45)$$

$$= \sum_{k=0}^h q^{\binom{k}{2} + \binom{h-k}{2} + n(h-k)} \begin{bmatrix} n \\ k \end{bmatrix} \begin{bmatrix} m \\ h-k \end{bmatrix} \quad (46)$$

其中有：

$$\binom{k}{2} + \binom{h-k}{2} + n(h-k) - \binom{h}{2} = (n-k)(h-k) \quad (47)$$

□

**推论 3.3.** 我们有：

$$\sum_{k=0}^h q^{(m+1)k} \begin{bmatrix} n-1+k \\ k \end{bmatrix} \begin{bmatrix} m+h-k \\ h-k \end{bmatrix} = \begin{bmatrix} m+n+h \\ h \end{bmatrix} \quad (48)$$

证明. 根据  $q$ -binomial theorem 有：

$$\begin{bmatrix} m+n+h \\ h \end{bmatrix} = \frac{1}{(z)_{m+n+1}} |z^h| \quad (49)$$

$$= \frac{1}{(z)_{m+1}} \frac{1}{(zq^{m+1})_n} |z^h| \quad (50)$$

$$= \left( \sum_{k=0}^{\infty} \begin{bmatrix} m+k \\ k \end{bmatrix} z^k \right) \left( \sum_{j=0}^{\infty} \begin{bmatrix} n-1+j \\ j \end{bmatrix} (zq^{m+1})^j \right) |z^h| \quad (51)$$

$$= \sum_{k=0}^h q^{(m+1)k} \begin{bmatrix} m+h-k \\ h-k \end{bmatrix} \begin{bmatrix} n-1+k \\ k \end{bmatrix} \quad (52)$$

□

上述推论指出的恒等式都是以下结果的特例，称  $q$ -Vandermonde convolution：

**定理 3.5.** 我们有：

$${}_2\phi_1 \left( \begin{matrix} a, q^{-n} \\ c \end{matrix}; q; q \right) = \frac{(c/a)_n}{(c)_n} a^n \quad (53)$$

为了证明上面的等式，我们需要先做一些铺垫：

**引理 3.1.** 在  $|z| < 1, |b| < 1$  时我们有：

$${}_2\phi_1 \left( \begin{matrix} a, b \\ c \end{matrix}; q; z \right) = \frac{(b, az; q)_{\infty}}{(c, z; q)_{\infty}} {}_2\phi_1 \left( \begin{matrix} c/b, z \\ az \end{matrix}; q; b \right) \quad (54)$$

其中  $(a, b; q)_k = (a; q)_k (b; q)_k$ 。

证明. 根据定理 3.4 我们有:

$${}_2\phi_1\left(\begin{matrix} a, b \\ c \end{matrix}; q; z\right) = \sum_{k=0}^{\infty} \frac{(a, b; q)_k}{(q, c; q)_k} z^k \quad (55)$$

$$= \frac{(b)_{\infty}}{(c)_{\infty}} \sum_{k=0}^{\infty} \frac{(a)_k}{(q)_k} \frac{(cq^k)_{\infty}}{(bq^k)_{\infty}} z^k \quad (56)$$

$$= \frac{(b)_{\infty}}{(c)_{\infty}} \sum_{k=0}^{\infty} \frac{(a)_k}{(q)_k} z^k \sum_{j=0}^{\infty} \frac{(c/b)_j}{(q)_j} (bq^k)^j \quad (57)$$

$$= \frac{(b)_{\infty}}{(c)_{\infty}} \sum_{j=0}^{\infty} \frac{(c/b)_j}{(q)_j} b^j \sum_{k=0}^{\infty} \frac{(a)_k}{(q)_k} (zq^j)^k \quad (58)$$

$$= \frac{(b)_{\infty}}{(c)_{\infty}} \sum_{j=0}^{\infty} \frac{(c/b)_j}{(q)_j} b^j \frac{(azq^j)_{\infty}}{(zq^j)_{\infty}} \quad (59)$$

$$= \frac{(b, az)_{\infty}}{(c, z)_{\infty}} \sum_{j=0}^{\infty} \frac{(z, c/b)_j}{(az, q)_j} b^j \quad (60)$$

$$= \frac{(b, az)_{\infty}}{(c, z)_{\infty}} {}_2\phi_1\left(\begin{matrix} c/b, z \\ az \end{matrix}; q; b\right) \quad (61)$$

在  $|b| < 1, |c/ab| < 1$  时我们有:

$${}_2\phi_1\left(\begin{matrix} a, b \\ c \end{matrix}; q; c/ab\right) = \frac{(b, c/b; q)_{\infty}}{(c, c/ab; q)_{\infty}} {}_1\phi_0\left(\begin{matrix} c/ab \\ - \end{matrix}; q; b\right) \quad (62)$$

用定理 3.4 就可以得到:

$${}_2\phi_1\left(\begin{matrix} a, b \\ c \end{matrix}; q; c/ab\right) = \frac{(c/a, c/b; q)_{\infty}}{(c, c/ab; q)_{\infty}} \quad (63)$$

也就有了:

$${}_2\phi_1\left(\begin{matrix} a, q^{-n} \\ c \end{matrix}; q; cq^n/a\right) = \frac{(c/a; q)_n}{(c; q)_n} \quad (64)$$

□

上面的等式是  $q$ -Vandermonde convolution 的一个变式. 为了得到定理 3.5, 将上式展

开得到：

$${}_2\phi_1\left(\begin{matrix} a, q^{-n} \\ c \end{matrix}; q; cq^n/a\right) = \sum_{k=0}^{\infty} \frac{(a; q)_k (q^{-n}; q)_k}{(q; q)_k (c; q)_k} (cq^n/a)^k \quad (65)$$

$$= \sum_{k=0}^{\infty} \frac{(a; q)_k (q^{-n}; q)_k}{(q; q)_k (c; q)_k} (cq/aq^{-n})^k q^{-k} \quad (66)$$

$$= \sum_{k=0}^{\infty} \frac{(a^{-1}; q^{-1})_k (q^n; q^{-1})_k}{(q^{-1}; q^{-1})_k (c^{-1}; q^{-1})_k} q^{-k} \quad (67)$$

$$= {}_2\phi_1\left(\begin{matrix} a^{-1}, q^n \\ c^{-1} \end{matrix}; q^{-1}; q^{-1}\right) \quad (68)$$

这是因为我们有：

$$(a; q)_k a^{-k} = (a^{-1} - 1) \cdots (a^{-1} - q^{k-1}) \quad (69)$$

$$= (-1)^k q^{\frac{k(k-1)}{2}} (1 - a^{-1}) \cdots (1 - a^{-1} q^{-(k-1)}) \quad (70)$$

$$= (-1)^k q^{\frac{k(k-1)}{2}} (a^{-1}; q^{-1})_k \quad (71)$$

那么现在有：

$${}_2\phi_1\left(\begin{matrix} a^{-1}, q^n \\ c^{-1} \end{matrix}; q^{-1}; q^{-1}\right) = \frac{(c/a; q)_n}{(c; q)_n} \quad (72)$$

将变量全部代换为  $x \leftarrow x^{-1}$ ，就得到了我们想要的结果：

$${}_2\phi_1\left(\begin{matrix} a, q^{-n} \\ c \end{matrix}; q; q\right) = \frac{(a/c; q^{-1})_n}{(c^{-1}; q^{-1})_n} \quad (73)$$

$$= \frac{(c/a; q)_n}{(c; q)_n} a^n \quad (74)$$

## 4 $q, t$ -Catalan numbers

### 4.1 The bounce statistic

这一节我们将介绍一个关于 Dyck path 的新 statistic: bounce statistic。

**定义 4.1.** 对于路径  $\pi \in L_{n,n}^+$  定义其 bounce path 为如下算法生成的路径：

从  $(0,0)$  开始，跟着  $\pi$  走直到遇见一个 E，转向并走到  $y = x$ 。此时再次转向，跟着  $\pi$  走直到再次遇见某个 E，重复上述过程。

可以将 bounce path 想象成从  $(0,0)$  出发、初始向 N 走的、碰到  $\pi$  或  $y = x$  会反弹的小球走出的路径。



我们记下所有碰到  $y = x$  的位置：

$$(0, 0), (j_1, j_1), \dots, (j_b, j_b) = (n, n) \quad (75)$$

对此，我们定义 bounce statistic  $\text{bounce}(\pi)$  为：

$$\text{bounce}(\pi) = \sum_{i=1}^{b-1} n - j_i \quad (76)$$

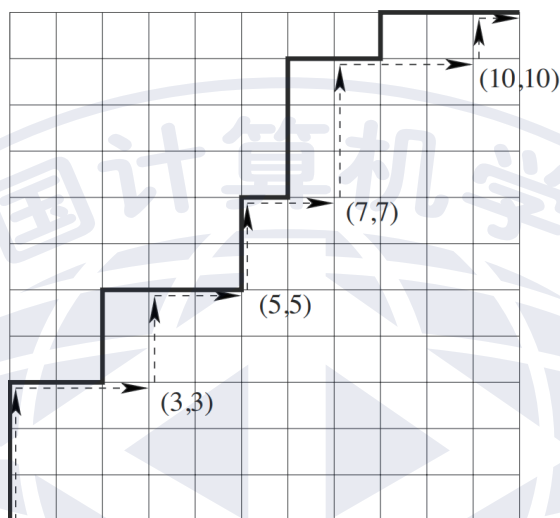


图 4: bounce path 示例

我们称  $b$  为反弹次数，称  $j_1$  为第一次反弹的长度，称  $j_2 - j_1$  为第二次反弹的长度.. 剩下的以此类推。

**定义 4.2.** 我们定义：

$$F_n(q, t) = \sum_{\pi \in L_{n,n}^+} q^{\text{area}(\pi)} t^{\text{bounce}(\pi)} \quad (77)$$

**定义 4.3.** 记  $L_{n,n}^+(k)$  为满足  $\pi \in L_{n,n}^+$  且恰好以  $k$  个 N 开头，然后紧接着下一步是 E 的那些  $\pi$  的集合。

特别地，对于  $L_{0,0}^+(k)$ 。当  $k = 0$  时其仅包含空路径，否则为空。

我们定义：

$$F_{n,k}(q, t) = \sum_{\pi \in L_{n,n}^+(k)} q^{\text{area}(\pi)} t^{\text{bounce}(\pi)} \quad (78)$$

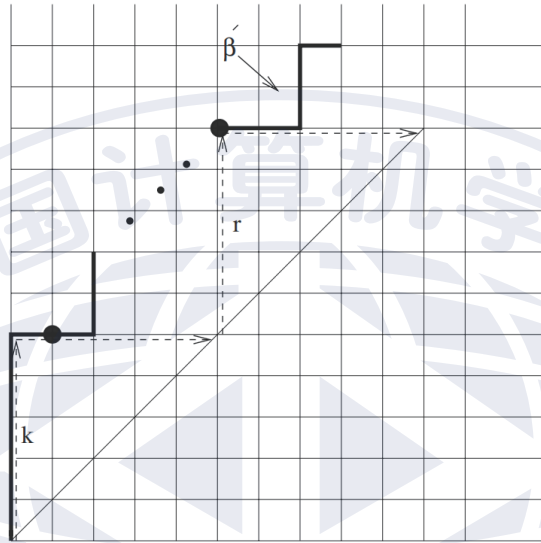
特别地， $F_{n,0}$  仅在  $n = 0$  时取 1，否则取 0。

那么我们有如下递归关系：

**定理 4.1.** 我们有：

$$F_{n,k}(q, t) = \sum_{r=0}^{n-k} \begin{bmatrix} r+k-1 \\ r \end{bmatrix}_q t^{n-k} q^{\binom{k}{2}} F_{n-k,r}(q, t) \quad (79)$$

证明. 考虑反弹至少两次的  $\beta \in L_{n,n}^+(k)$ , 其满足第一次反弹长度为  $k$ , 第二次反弹长度为  $r$ ——这意味着其经过点  $(1, k), (k, k+r)$ , 即下图中的两个大点：



将  $\beta$  分成两个部分, 第一个部分是  $(0,0)$  到  $(k, k+r)$ , 第二个部分是  $(k, k+r)$  到  $(n, n)$ 。

对于第二个部分, 我们在其前面加入  $r$  个  $N$  并记其为  $\beta'$ 。此时有  $\beta' \in L_{n-k, n-k}^+(r)$ , 同时也有  $\text{bounce}(\beta') = \text{bounce}(\beta) + n - k$ 。

而面积可以拆成三部分:  $x = k$  右侧的面积,  $y = k$  下侧的面积以及剩余面积。第一部分只和  $\beta'$  有关系。第二部分一定是  $\binom{k}{2}$ 。第三部分, 只要考虑  $(1, k)$  走到  $(k, k+r)$  的方案。

根据定理 3.2, 所有方案的面积和为  $\begin{bmatrix} r+k-1 \\ r \end{bmatrix}_q$ 。因此有：

$$q^{\text{area}(\beta)} = q^{\text{area}(\beta')} q^{\binom{k}{2}} \begin{bmatrix} r+k-1 \\ r \end{bmatrix}_q \quad (80)$$

于是就有：

$$F_{n,k}(q, t) = \sum_{r=0}^{n-k} \sum_{\beta' \in L_{n-k, n-k}^+(r)} q^{\text{area}(\beta')} q^{\binom{k}{2}} \begin{bmatrix} r+k-1 \\ r \end{bmatrix}_q t^{\text{bounce}(\beta')} t^{n-k} \quad (81)$$

$$= \sum_{r=0}^{n-k} \begin{bmatrix} r+k-1 \\ r \end{bmatrix}_q t^{n-k} q^{\binom{k}{2}} F_{n-k,r}(q, t) \quad (82)$$

而对于  $n = k$  的情况（即唯一需要考虑反弹恰好一次的情况）上述结果是显然的。  $\square$

**推论 4.1.** 我们有：

$$F_n(q, t) = \sum_{b=1}^n \sum_{\alpha_1 + \dots + \alpha_b = n, \alpha_i > 0} t^{\sum_{j=2}^b (j-1)\alpha_j} q^{\sum_{j=1}^b \binom{\alpha_j}{2}} \prod_{i=1}^{b-1} \begin{bmatrix} \alpha_i + \alpha_{i+1} - 1 \\ \alpha_{i+1} \end{bmatrix}_q \quad (83)$$

## 4.2 $t = 1$ 与 $t = 1/q$ 的特殊情况

对于  $t = 1$ ，我们已经在前面得到了结果。即：

$$F_n(q, 1) = C_n(q) \quad (84)$$

而关于  $F_n(q, 1/q)$  有如下结果：

**定理 4.2.** 对于  $1 \leq k \leq n$  有：

$$q^{\binom{n}{2}} F_{n,k}(q, 1/q) = \frac{[k]}{[n]} \begin{bmatrix} 2n - k - 1 \\ n - k \end{bmatrix}_q q^{(k-1)n} \quad (85)$$

证明.  $n = k$  时上述结果是显然的。 $n < k$  时，我们对  $n$  进行归纳。 $n = 0$  时上述结果是显然的。对于  $n > 0$ ，我们参考定理 4.1 的证明做出如下推理：

$$q^{\binom{n}{2}} F_{n,k}(q, 1/q) = q^{\binom{n}{2} - \binom{n-k}{2}} \sum_{r=1}^{n-k} \begin{bmatrix} r + k - 1 \\ r \end{bmatrix}_q q^{\binom{k}{2} - \binom{n-k}{2}} q^{\binom{n-k}{2}} F_{n-k,r}(q, 1/q) \quad (86)$$

$$= q^{\binom{n}{2} + \binom{k}{2} - \binom{n-k}{2} - \binom{n-k}{2}} \sum_{r=1}^{n-k} \begin{bmatrix} r + k - 1 \\ r \end{bmatrix}_q \frac{[r]}{[n-k]} \begin{bmatrix} 2(n-k) - r - 1 \\ n - k - r \end{bmatrix}_q q^{(r-1)(n-k)} \quad (87)$$

$$= q^{n(k-1)} \sum_{r=1}^{n-k} \begin{bmatrix} r + k - 1 \\ r \end{bmatrix}_q \frac{[r]}{[n-k]} \begin{bmatrix} 2(n-k) - 2 - (r-1) \\ n - k - 1 - (r-1) \end{bmatrix}_q q^{(r-1)(n-k)} \quad (88)$$

$$= q^{n(k-1)} \frac{[k]}{[n-k]} \sum_{u=0}^{n-k-1} \begin{bmatrix} u + k \\ u \end{bmatrix}_q \begin{bmatrix} 2(n-k) - 2 - u \\ n - k - 1 - u \end{bmatrix}_q q^{u(n-k)} \quad (89)$$

应用  $q$ -binomial theorem 替换上式的后半部：

$$q^{n(k-1)} \frac{[k]}{[n-k]} \frac{1}{(zq^{n-k})_{k+1}} \frac{1}{(z)_{n-k}} \Big|_{z^{n-k-1}} = q^{n(k-1)} \frac{[k]}{[n-k]} \frac{1}{(z)_{n+1}} \Big|_{z^{n-k-1}} \quad (90)$$

$$= q^{n(k-1)} \frac{[k]}{[n-k]} \begin{bmatrix} 2n - k - 1 \\ n \end{bmatrix}_q \quad (91)$$

$\square$

**推论 4.2.** 我们有：

$$q^{\binom{n}{2}} F_n(q, 1/q) = \frac{1}{[n+1]} \begin{bmatrix} 2n \\ n \end{bmatrix}_q \quad (92)$$

证明. 我们有:

$$F_{n+1,1}(q, t) = t^n F_n(q, t) \quad (93)$$

这是因为:

$$q^{\binom{n+1}{2}} F_{n+1,1}(q, 1/q) = \frac{[1]}{[n+1]} \left[ \begin{matrix} 2(n+1)-2 \\ n+1-1 \end{matrix} \right]_q \quad (94)$$

$$= \frac{[1]}{[n+1]} \left[ \begin{matrix} 2n \\ n \end{matrix} \right]_q \quad (95)$$

$$= q^{\binom{n+1}{2}-n} F_n(q, 1/q) \quad (96)$$

$$= q^{\binom{n}{2}} F_n(q, 1/q) \quad (97)$$

□

### 4.3 对称性问题与 $\text{dinv}$ statistic

**定理 4.3.** 我们有:

$$\sum_{\pi \in L_{n,n}^+} q^{\text{area}(\pi)} = \sum_{\pi \in L_{n,n}^+} q^{\text{bounce}(\pi)} \quad (98)$$

证明. 对于  $\pi \in L_{n,n}^+$ , 其  $a_i(\pi)$  需要满足的必要条件是  $a_i(\pi) \leq a_{i-1}(\pi) + 1$ . 于是, 我们转而枚举  $a_i(\pi) = j$  的次数, 分别记为  $b_0, b_1, \dots, b_k$ . 接着考虑它们组合的方案数计算如下:

先将  $b_0, b_1$  组合, 方案数为  $\binom{b_0+b_1-1}{b_1}$ . 然后将  $b_2$  加入, 不能出现 0 后面紧邻一个 2 的情况. 方案数为  $\binom{b_1+b_2-1}{b_2}$  .. 以此类推.

因此, 我们可以得到:

$$\sum_{\pi \in L_{n,n}^+} q^{\text{area}(\pi)} = \sum_{k=1}^n \sum_{b_1+\dots+b_k=n, b_i>0} q^{\sum_{i=2}^k b_i(i-1)} \prod_{i=1}^{k-1} \binom{b_i+b_{i+1}-1}{b_{i+1}} \quad (99)$$

结果与  $F_n(q, 1)$  一致.

□

事实上, 我们存在:

**定理 4.4.**

$$\sum_{\pi \in L_{n,n}^+} q^{\text{area}(\pi)} t^{\text{bounce}(\pi)} = \sum_{\pi \in L_{n,n}^+} q^{\text{bounce}(\pi)} t^{\text{area}(\pi)} \quad (100)$$

即  $F_n(q, t) = F_n(t, q)$ 。证明复杂<sup>1</sup>，本文篇幅有限，这里就不展开了。

我们接下来介绍另外一个关于  $q, t$ -Catalan 的 Dyck path 的 statistic。如下所示：

**定义 4.4.** 对于  $\pi \in L_{n,n}^+$  有：

$$\text{dinv}(\pi) = |\{(i, j) : 1 \leq i < j \leq n, a_i(\pi) = a_j(\pi)\}| + \quad (101)$$

$$|\{(i, j) : 1 \leq i < j \leq n, a_i(\pi) = a_j(\pi) + 1\}| \quad (102)$$

**定义 4.5.** 我们有：

$$\sum_{\pi \in L_{n,n}^+} q^{\text{dinv}(\pi)} t^{\text{area}(\pi)} = \sum_{\pi \in L_{n,n}^+} q^{\text{bounce}(\pi)} t^{\text{area}(\pi)} \quad (103)$$

证明. 我们构造 Dyck path 的双射  $\zeta$  使其满足：

$$\text{dinv}(\pi) = \text{area}(\zeta(\pi)) \quad (104)$$

$$\text{area}(\pi) = \text{bounce}(\zeta(\pi)) \quad (105)$$

对于  $\pi \in L_{n,n}^+$ ，令  $b-1$  为其最长行的长度（即  $\max a_i(\pi)$ ），记  $c_0, \dots, c_{b-1}$  为每个长度对应的出现次数。

我们构造对应路径  $\pi'$  使其有恰好  $b$  次反弹，且反弹长度依次为  $c_0, \dots, c_{b-1}$ 。对于每个  $0 \leq i < b-1$  的  $(c_i, c_i + c_{i+1})$  处，我们构造其与  $(c_{i-1}, c_{i-1} + c_i)$ （当  $i=0$  即  $(0,0)$ ）之间的路径：从上到下考虑所有  $a_i(\pi)$ ，若是  $i+1$  则加入一个 S (South step)，若是  $i$  则加入一个 W (West step)，其他情况不做考虑。

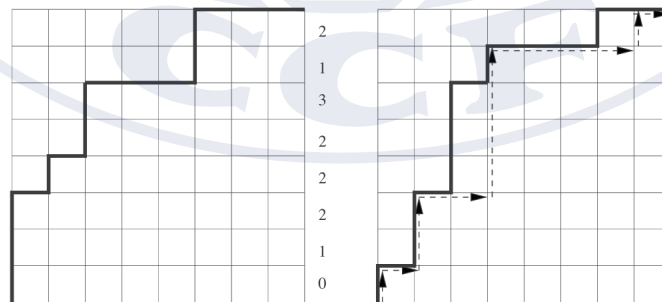


图 5: 左侧的  $\pi$  对应了右侧的  $\pi'$

$\zeta$  很显然是个双射：对于  $\pi'$ ，我们通过反弹次数得到了  $b-1$ ，并且得到了每个长度的出现次数。此时再看具体路径，对于  $0 \leq i < b-2$  可以得到仅有  $i, i+1$  的子序列。而组合原路径的方案是唯一的，这和定理 4.3 的证明是对应的。

<sup>1</sup>读者可以参考 [2]

我们将  $\text{area}(\pi')$  按照 bounce path 分成两部分：下侧是  $\binom{c_0}{2} + \dots + \binom{c_{b-1}}{2}$ ，对应了  $\text{dinv}(\pi)$  中的  $\{(i, j) : 1 \leq i < j \leq n, a_i(\pi) = a_j(\pi)\}$ 。上侧，分析每一段，不难发现也恰好对应了  $\{(i, j) : 1 \leq i < j \leq n, a_i(\pi) = a_j(\pi) + 1\}$ 。

因此  $\text{area}(\pi') = \text{dinv}(\pi)$ 。另一方面，我们有：

$$\text{bounce}(\pi') = \sum_{i=1}^{b-1} ic_i = \text{area}(\pi) \quad (106)$$

□

**推论 4.3.** 我们有：

$$F_{n,k}(q, t) = \sum_{\pi \in L_{n,n}^+, \pi \text{ has exactly } k \text{ rows of length } 0} q^{\text{dinv}(\pi)} t^{\text{area}(\pi)} \quad (107)$$

证明. 定义 4.3 的限制通过  $\zeta$  对应的限制即为有恰好  $k$  行长度为 0 (即第一次反弹长度为  $k$ )。□

## 5 $q$ -functional composition

### 5.1 基础对称函数

对于多元函数  $f(x_1, \dots, x_n)$ ，称其是对称的，当且仅当：对于任意排列  $\sigma$ ，令  $\sigma f = f(x_{\sigma_1}, \dots, x_{\sigma_n})$ ，那么  $\sigma f = f$  必然存在。

**定义 5.1.** 我们记  $\Lambda$  为所有  $x_1, \dots, x_n$  的对称函数的环。同时记  $\Lambda^n$  为其由  $n$  次齐次对称函数组成的向量空间。

最基本的对称函数是单项式对称函数，其取决于一组变量以及一个 partition  $\lambda$ 。记  $n$  的所有 partition 组成的集合为  $\text{Par}(n)$ ，并以  $\lambda \vdash n$  表示  $\lambda$  属于  $\text{Par}(n)$ 。

**定义 5.2.** 我们记  $m_\lambda(x_1, \dots, x_n)$  为满足指数的可重集为  $\lambda$  的不同单项式的和。特别地，记  $m_\lambda(X_n) = m_\lambda(x_1, \dots, x_n)$ 。以及  $m_\lambda(X) = m_\lambda(x_1, x_2, \dots)$ ，可以根据上下文省略  $X$ 。

下面是几个例子：

$$m_{1,1} = \sum_{i < j} x_i x_j \quad (108)$$

$$m_{2,1,1}(X_3) = x_1^2 x_2 x_3 + x_1 x_2^2 x_3 + x_1 x_2 x_3^2 \quad (109)$$

$$m_2(X) = \sum_i x_i^2 \quad (110)$$

可以发现，任意  $\Lambda$  中的元素，都可以被一些  $m_\lambda$  线性组合得到。

我们记  $1^n$  为由  $n$  个 1 组成的 partition。函数  $m_{1^n}$  被称作第  $n$  个 **elementary symmetric function**，有如下定义：



定义 5.3. 我们记  $e_n = m_{1^n}$ , 满足:

$$\prod_{i=1}^{\infty} (1 + zx_i) = \sum_{n=0}^{\infty} z^n e_n \quad (111)$$

同时也有如下几个值得关注的对称函数:

定义 5.4. 我们记  $p_n = m_n$ , 称 **power-sum symmetric functions**。

定义 5.5. 我们记  $h_n$  为 **complete homogeneous symmetric functions**。有  $h_n = \sum_{\lambda \vdash n} m_{\lambda}$ , 其满足:

$$\frac{1}{\prod_{i=0}^{\infty} (1 - zx_i)} = \sum_{n=0}^{\infty} z^n h_n \quad (112)$$

另外, 对于  $\lambda \vdash n$ , 有如下定义:

定义 5.6. 我们记  $e_{\lambda} = \prod_i e_{\lambda_i}$ 。

定义 5.7. 我们记  $p_{\lambda} = \prod_i p_{\lambda_i}$ 。

定义 5.8. 我们记  $h_{\lambda} = \prod_i h_{\lambda_i}$ 。

举个例子:

$$e_{2,1} = \left( \sum_{i < j} x_i x_j \right) \left( \sum_k x_k \right) = m_{2,1} + 3m_{1,1,1} \quad (113)$$

$$p_{2,1} = \left( \sum_i x_i^2 \right) \left( \sum_j x_j \right) = m_3 + m_{2,1} \quad (114)$$

$$h_{2,1} = \left( \sum_i x_i^2 + \sum_{i < j} x_i x_j \right) \left( \sum_k x_k \right) = m_3 + 2m_{2,1} + 3m_{1,1,1} \quad (115)$$

定义 5.9. 以下是两个在 partition 上简单的函数:

$$n(\lambda) = \sum_i (i-1)\lambda_i \quad (116)$$

$$z_{\lambda} = \prod_i i^{m_i} m_i! \quad (117)$$

其中  $m_i = m_i(\lambda)$  表示  $\lambda$  中  $i$  的出现次数。

接下来的命题揭示了上述三个对称函数的关系:

命题 5.1.

$$h_n = \sum_{\lambda \vdash n} \frac{p_{\lambda}}{z_{\lambda}} \quad (118)$$

$$e_n = \sum_{\lambda \vdash n} \frac{(-1)^{n-\ell(\lambda)} p_{\lambda}}{z_{\lambda}} \quad (119)$$

证明. 对于前者, 我们考虑  $\lambda$  的每个组成部分, 列出  $h_n$  的关系式:

$$\sum_{n=0}^{\infty} z^n h_n = \prod_{i=1}^{\infty} \exp\left(\frac{z^i}{i} \sum_{j=0}^{\infty} x_j^i\right) \quad (120)$$

即:

$$\ln\left(\sum_{n=0}^{\infty} z^n h_n\right) = \sum_{i=1}^{\infty} \frac{z^i}{i} \sum_{j=0}^{\infty} x_j^i \quad (121)$$

而讨论定义 5.5 给出的等式可以得到:

$$\ln\left(\sum_{n=0}^{\infty} z^n h_n\right) = \ln\left(\frac{1}{\prod_{i=0}^{\infty} (1 - zx_i)}\right) \quad (122)$$

$$= \sum_{i=0}^{\infty} \sum_{j=1}^{\infty} \frac{x_i^j z^j}{j} \quad (123)$$

对于后者, 证明过程是类似的。  $\square$

**定义 5.10.** 对于  $f \in \Lambda$ , 称其  $n$  阶 **principal specialization** 为  $f(1, q, \dots, q^{n-1})$ 。

下面的等式揭示了本节与前文的联系。

**定理 5.1.** 我们有:

$$e_m(1, q, \dots, q^{n-1}) = q^{\binom{m}{2}} \begin{bmatrix} n \\ m \end{bmatrix}_q \quad (124)$$

$$h_m(1, q, \dots, q^{n-1}) = \begin{bmatrix} n-1+m \\ m \end{bmatrix}_q \quad (125)$$

$$p_m(1, q, \dots, q^{n-1}) = \frac{1 - q^{nm}}{1 - q^m} \quad (126)$$

证明.  $e_m, h_m$  的等式只需要将定义 5.3、定义 5.5 的等式与  $q$ -binomial theorem 联系即可。  
 $p_m$  的等式是显然的。  $\square$

## 5.2 对称函数与 $q$ -functional composition

**定理 5.2.** 我们定义  $q$ -functional composition 为:

$$(F \circ_q G)(z) = \sum_n f_n G(z) G(qz) \cdots G(q^{n-1}z) \quad (127)$$

其中  $F = \sum_n f_n z^n$ 。同时:

$$F \circ_q G = z \text{ and } G \circ_{q^{-1}} F = z \quad (128)$$

是等价的。我们称  $F, G$  互为  $q$ -复合逆。也就有：

$$(\Phi \circ_{q^{-1}} F) \circ_q G = \Phi = (\Phi \circ_q G) \circ_{q^{-1}} F \quad (129)$$

对所有  $\Phi$  都成立。

我们记  $H(z) = \sum_{i=0}^{\infty} z^i h_i$ ，与之对应的就有  $1/H(-z) = \sum_{i=0}^{\infty} z^i e_i$ 。

考虑  $F(z) = zH(-z)$ ，我们记  $G(z)$  满足  $F \circ_q G = z$ ，有：

$$G(z) = zH^*(zq; q) = \sum_{k=0}^{\infty} q^k h_k^*(q) z^{k+1} \quad (130)$$

其中：

$$H^*(z; q) = \sum_{k=0}^{\infty} h_k^*(q) z^k \quad (131)$$

即有：

$$G \circ_{q^{-1}} F = z = \sum_{k=0}^{\infty} q^{-\binom{k}{2}} h_k^*(q) z^{k+1} H(-z) H(-q^{-1}z) \cdots H(-q^k z) \quad (132)$$

$h_n^*(q), n \geq 0$  的解是唯一的：

**命题 5.2.** 我们有：

$$h_n^*(q) = \sum_{\lambda \subseteq \delta_n} q^{\binom{n}{2} - |\lambda|} e_{\beta(\lambda)} \quad (133)$$

其中  $\beta(\lambda)$  为  $0 \leq i < n$  的  $m_i(\lambda)$  形成的可重集。 $m_0(\lambda)$  认为是  $\delta_n$  剩余的空行数。即有  $|\beta(\lambda)| = n$ 。而  $\delta_n$  为由从下往上长度分别为  $n, n-1, \dots, 1$  的  $n$  行组成的 Ferrers graph。

证明. 将式 (132) 除  $zH(-z)$ ，然后用  $zq$  代换  $z$  得到：

$$1/H(-zq) = \sum_{k=0}^{\infty} q^k h_k^*(q) zH(-z) q^{-1} zH(-q^{-1}z) \cdots q^{1-k} zH(-q^{1-k}z) \quad (134)$$

$$= H^*(zq; q) \circ_{q^{-1}} (zH(-z)) \quad (135)$$

应用定理 5.2：令  $F \leftarrow zH(-z), \Phi \leftarrow H^*(zq; q)$ ，可以得到：

$$H^*(zq; q) = (1/H(-zq)) \circ_q (zH^*(zq; q)) \quad (136)$$

比较  $z^n$  的系数，可以得到  $h_n^*(q), n > 0$  的递推式：

$$h_n^*(q) = \sum_{k=1}^n q^{\binom{k}{2}} e_k \sum_{n_1 + \dots + n_k = n-k, n_i \geq 0} \prod_{i=1}^k q^{(i-1)n_i} h_{n_i}^*(q) \quad (137)$$

接下来我们证明以上递推式与式 (133) 是等价的:  $n = 0$  时显然等价;  $n > 0$  时, 考虑归纳。

对于  $\lambda$ , 我们记其最大一个非零的位置是  $n - k$ 。与  $\delta_n$  对比, 我们认为  $\lambda$  上面还有  $k$  个 0 行。然后, 我们将  $\delta_n$  的左下角与坐标系  $(0, 0)$  对齐: 那么限制就是  $y = n - x$ 。将  $\lambda$  的左下角与坐标系  $(0, 0)$  对齐后不能与限制线相交。

然后, 我们对于  $1 \leq i \leq k$  记  $p_i$  为最大的满足  $\lambda_k > n - i - k$  的  $k$ 。如果不存在, 令  $p_i = 0$ 。可以将  $p_i$  看作  $y = n - i - x$  往下扫, 扫到的  $y$  最大的行的标号。

记  $p_0 = 0$ 。 $p_i$  一定是单调不降的, 同时  $p_k = n - k$ 。那么我们令:

$$p_i = n_1 + \cdots + n_i \quad (138)$$

即  $n_1 + \cdots + n_k = n - k$ 。

此时,  $m_0(\lambda) = k$ , 那么  $y = n - k$  上侧的贡献就是  $q^{\binom{k}{2}} e_k$ 。接下来, 我们考虑每个行标号区间  $(p_0, p_1], (p_1, p_2], \dots, (p_{k-1}, p_k]$  的贡献。

对于  $(p_i, p_{i+1}]$ , 我们称  $\binom{n}{2} - |\lambda|$  为“空格子个数”。此时, 限制线下移了  $i$  格, 因此最后就需要加上  $i \cdot n_{i+1}$  个空格子。

然后, 从  $p_{i+1}$  行最后一个格子 (记为  $(p_{i+1}, t)$ ) 的右上角出发, 导出一条斜率为  $-1$  的线  $l$ 。行坐标在  $(p_i, p_{i+1}]$  中且列坐标在  $t + 1, t + n_{i+1}$  的子区域内,  $\lambda$  一定在  $l$  下方。观察这个子区域,  $(p_i, p_{i+1}]$  可能的长度与贡献和与  $h_{n_{i+1}}^*(q)$  是对应的。

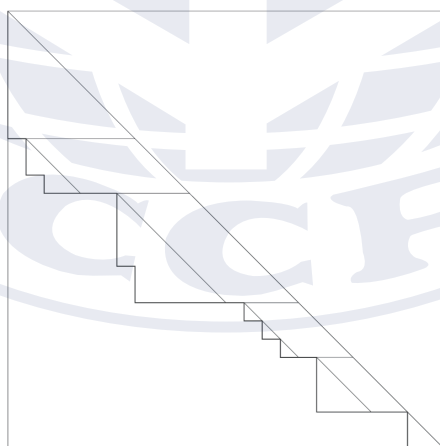


图 6: 划分示例

□

推论 5.1.

$$\sum_{k=0}^{\infty} e_k z^k = \sum_{k=0}^{\infty} q^{-\binom{k}{2}} h_k^*(q) z^k H(-q^{-1}z) \cdots H(-q^k z) \quad (139)$$

**推论 5.2.** 我们定义  $h_n^*(\mathbf{c}, q)$  满足:

$$\sum_{k=0}^{\infty} e_k c_k z^k = \sum_{k=0}^{\infty} q^{-\binom{k}{2}} h_k^*(\mathbf{c}, q) z^k H(-q^{-1}z) \cdots H(-q^k z) \quad (140)$$

有:

$$h_n^*(\mathbf{c}, q) = \sum_{k=0}^n c_k \sum_{\lambda \subseteq \delta_n, m_0(\lambda)=k} q^{\binom{n}{2}-|\lambda|} e_{\beta(\lambda)} \quad (141)$$

证明. 我们有:

$$\sum_{k=0}^{\infty} e_k c_k z^k q^k = \sum_{k=0}^{\infty} q^k h_k^*(\mathbf{c}, q) z H(-z) q^{-1} z H(-q^{-1}z) \cdots q^{1-k} z H(-q^{1-k}z) \quad (142)$$

$$= H^*(zq, \mathbf{c}; q) \circ_{q^{-1}} F \quad (143)$$

其中

$$H^*(z, \mathbf{c}; q) = \sum_{k=0}^{\infty} h_k^*(\mathbf{c}, q) z^k \quad (144)$$

即有:

$$H^*(zq, \mathbf{c}; q) = \left( \sum_{k=0}^{\infty} e_k c_k z^k q^k \right) \circ_q G \quad (145)$$

同样比较  $z^n$  的系数, 可以得到  $h_n^*(\mathbf{c}, q), n > 0$  的递推式:

$$h_n^*(\mathbf{c}, q) = \sum_{k=1}^n q^{\binom{k}{2}} e_k c_k \sum_{n_1+\cdots+n_k=n-k, n_i \geq 0} \prod_{i=1}^k q^{(i-1)n_i} h_{n_i}^*(q) \quad (146)$$

接下来的证明过程与命题 5.2 一致。  $\square$

## 6 总结

本文介绍了  $q$ -二项式系数、 $q$ -范德蒙德卷积以及  $q$ -复合, 并着重介绍了  $q$ -卡特兰数以及  $q, t$ -卡特兰数。在此过程中, 多次通过 statistic 给出了许多有趣的结论, 同时介绍了对应的双射构造。

本文的内容不算全面, 仅是简单地介绍了  $q$ -模拟最基本的内容。限于篇幅, 笔者对包括  $q$ -拉格朗日反演在内的更多内容不在本文范围内感到遗憾。

不过, 笔者希望本文能够起到抛砖引玉的作用, 吸引更多的读者来探索、思考与研究相关问题。笔者期待有更多的  $q$ -模拟相关的问题出现在算法竞赛中。



## 致谢

感谢父母、朋友对我的支持与鼓励。  
感谢中国计算机学会提供学习和交流的平台。  
感谢曹利国老师多年以来的关心和教导。  
感谢国家集训队杨耀良教练和彭思进教练的指导。  
感谢时庆钰学长、林卓铖同学与我交流讨论。  
感谢胡一凡同学为本文验稿。

## 参考文献

- [1] G. Gasper and M. Rahman. Basic hypergeometric series, second ed. 2004.
- [2] J. Haglund. Catalan paths and  $q, t$ -enumeration. 2014.
- [3] M. Haiman. Conjectures on the quotient ring by diagonal invariants. Journal of Combinatorial Theory, pages 17–76, 1994.
- [4] D. Stanton. Recent results for the  $q$ -lagrange inversion formula. pages 525–536, 1988.





# 浅谈最小比率环及其应用

南京外国语学校 魏佳泽

## 摘要

本文介绍了最小比率环及其思想的应用，并利用最小比率环在一类最短路问题上获得了比已知解法更优的算法。

## 1 前言

比率问题是一类经典问题，有一套成熟的解法。最小比率环的思想在信息学竞赛中大量使用，却很少有资料对其进行总结。为此，文章介绍了最小比率环及其思想在各种问题上的应用。

文章第2节介绍了最小比率环的相关定义，第3节介绍了最小比率环的求解方法，第4节以最短路问题为例介绍了最小比率环的应用，第5节介绍了几个其它类型的比率问题的解法。

## 2 定义

假定读者已经掌握了最基本的图论术语，文章只对核心定义进行阐述。

在不引起歧义时，记  $n$  表示图的点数， $m$  表示图的边数。

**定义 2.1 (途径).** 连接一串结点的序列称为途径 (Walk)，用点序列  $v_{0..k}$  和边序列  $e_{1..k}$  描述，其中  $e_i = (v_{i-1}, v_i)$ 。

**定义 2.2 (途径长度).** 定义途径  $W$  的长度为边数  $k$ ，记作  $|W|$ 。

**说明:** 除 4.2 小节涉及无穷长途径以外，文章其它所有地方讨论的途径均为有限长，包括但不限于无特殊说明的，由途径引出的定义及相关定理。

**定义 2.3 (途径权值).** 定义途径  $W$  的权值为其所有边的权值之和  $\sum_{i=1}^k w(e_i)$ ，记作  $w(W)$ 。

以上是途径权值的最常见定义，不符合该定义的情况会进行特殊说明，例如途径长度无穷长的情况。

**定义 2.4** (环, 负环与零环). 除起点等于终点以外, 所有点互不相同的途径称为**环**或**圈** (Cycle)。权值为负数的环称为**负环**。权值为 0 的环称为**零环**。

**定义 2.5** (路径). 所有点互不相同的途径称为**路径** (Path)。

**特别注意:** 在信息学竞赛中, 通常使用“路径”代指“途径”, 用“环”代指“起点等于终点的途径”, 这两组概念的区别均仅在于途径上的点是否可重。由于这种代指已被广泛采用, 为避免繁琐的说明, 文章接下来将立刻采用同样的代指。当需要强调路径上的点互不相同或路径上除了起点等于终点以外其它点互不不同时, 文章将相对应地称呼为**简单路径**和**简单环**。

**定义 2.6** (比率). 定义一个非空途径  $W$  的比率 (Ratio) 为它的权值除以长度, 记作  $r(W) = \frac{w(W)}{|W|}$ 。

**定义 2.7** (最小比率环和最小环比率). 称图上比率最小的环为**最小比率环**, 记作  $C_{\min}(G)$ , 对应权值称为**最小环比率**, 记作  $r_{\min}(G)$ 。类似定义**最大比率环**  $C_{\max}(G)$  和**最大环比率**  $r_{\max}(G)$ 。

**定义 2.8** (路径的连接). 若路径  $P'$  的终点等于  $Q$  的起点, 则将  $P'$  的终点和  $Q$  的起点连接在一起得到的路径  $P$  称为  $P'$  和  $Q$  的**连接**, 记作  $P = P'Q$ 。

**定义 2.9** (环的重复). 对于环  $C$ , 将  $k$  个  $C$  连接在一起得到的环称为  $C$  的  $k$  **重复**, 记作  $C^k$ 。特别地, 将无穷多个  $C$  连接在一起得到的路径称为环的**无限重复**, 记作  $C^\infty$ 。

### 3 最小比率环的求法

#### 3.1 分数规划

给出定义在集合  $S_0 \subset \mathbb{R}^n$  上的实函数  $f, g, h_{1..m}$ , 设  $S = \{x \in S_0 : h_j(x) \leq 0, j = 1, \dots, m\}$ , 最大化  $\frac{f(x)}{g(x)}, x \in S$  的问题称为**分数规划**。

信息学竞赛中研究的分数规划一般形如给定一组  $(a_i, b_i)$ , 求一组  $c_i \in \{0, 1\}$  最小化  $\frac{\sum a_i c_i}{\sum b_i c_i}$ , 其中  $b_i > 0$  且要求  $c_i$  不全为 0, 即选择一些物品 (不可以不选), 使得选中物品的  $a_i$  之和除以  $b_i$  之和最小。方便起见, 文章称  $a_i$  为**权值**,  $b_i$  为**权重**,  $\frac{\sum a_i w_i}{\sum b_i w_i}$  为**比率**。因系数  $c_i$  只有 0 和 1 两种取值, 问题也称 0/1 分数规划。

0/1 分数规划的常用解法是二分法, 通过二分答案将最优化问题转化为判定性问题。具体地, 二分答案  $mid$ , 并比较答案与  $mid$  的大小。这相当于检查是否存在一组方案  $c_i \in \{0, 1\}$  使得

$$\frac{\sum a_i c_i}{\sum b_i c_i} \leq mid$$

变形后得

$$\sum c_i (a_i - mid \cdot b_i) \leq 0$$

若无特殊限制, 则只需检查是否存在  $a_i - mid \cdot b_i \leq 0$ , 即  $\frac{a_i}{b_i} \leq mid$ 。这很容易理解: 如果任意选择均合法, 选择比率最小的物品一定最优, 再选其它任何物品均会让比率变大。

因此, 题目会对  $c_i$  的选择进行限制, 只有满足特殊条件的方案才合法。基于不同的限制条件, 0/1 分数规划引出了最小比率环, 最小比率背包, 最小比率生成树, 最大密度子图等一系列比率问题。文章重点研究最小比率环, 其它比率问题将一并在文末提及。

## 3.2 二分法

**例题 3.1** (最小圈<sup>1</sup>)。给定一张边带实数权的有向图, 求最小环比率, 或报告无解。

$2 \leq n \leq 3 \times 10^3$ ,  $1 \leq m \leq 10^4$ ,  $|w(u, v)| \leq 10^7$ 。

写成分数规划的形式, 元素是图上的每一条边  $e \in E$ , 每个元素的权值  $a_i$  为其边权  $w(e)$ , 权重  $b_i = 1$ , 要求选出的所有元素构成环。

对于最小比率环问题, 可以借助分数规划的思路, 使用二分法求解。二分答案  $mid$ , 比较最小环比率  $r_{\min}(G)$  和  $mid$  的大小。

一个环的比率反映了其上所有边的边权的平均值。如果  $r_{\min}(G) < mid$ , 说明最小比率环的边权平均值小于  $mid$ 。此时将图上所有边的边权减去  $mid$ , 最小比率环变为负环; 否则所有环的边权平均值不小于  $mid$ , 操作不会导致任何负环。

将所有边的边权减去  $mid$  并使用 Bellman-Ford 算法或 Shortest Path Faster Algorithm (SPFA) 算法求负环。若图上存在负环, 则  $r_{\min}(G) < mid$ , 否则  $r_{\min}(G) \geq mid$ 。算法的时间复杂度为  $O(nm(\log \frac{w}{\epsilon}))$ 。

**引理 3.1.** 若一张图存在负环 (零环), 则存在简单负环 (零环)。

证明. 假设图有负环  $C$ , 则  $C$  的长度有限。

若  $C$  不是简单环, 则将  $C$  拆成两个非空环  $C_1, C_2$ , 方法是将重复出现的点的相邻两次出现对应的环  $C_1$  从  $C$  中取出, 剩余部分头尾相接得到另一个环  $C_2$ 。

因为  $w(C) = w(C_1) + w(C_2) < 0$ , 所以  $C_1$  和  $C_2$  至少一个是负环。根据一个非简单负环, 总能得到长度更小的负环。

不断进行上述操作, 因负环长度不小于 1, 所以过程总会停止。停止时对应的环一定是简单负环, 否则与过程停止矛盾。

同理可证若存在零环, 则存在简单零环。□

引理 3.1 是使用 BF 算法或 SPFA 算法求负环的正确性保证。

**定理 3.1.** 总存在最小比率环是简单环。

<sup>1</sup><https://www.luogu.com.cn/problem/P3199>, HNOI2009

证明. 将图上所有边的权值减去  $r_{\min}(G)$ , 则图上存在零环。由引理 3.1, 存在简单零环, 对应原图比率为  $r_{\min}(G)$  的简单最小比率环。

该定理也可以通过糖水不等式

$$\min\left(\frac{a}{b}, \frac{c}{d}\right) \leq \frac{a+c}{b+d} \leq \max\left(\frac{a}{b}, \frac{c}{d}\right) \quad (b, d > 0)$$

结合比率的定义和拆分非简单环的思想证明。  $\square$

关于二分法的精度问题: 直接在实数域上二分并不能求出精确解, 只在题目限制较松, 仅要求答案保留一定位小数时有效。

当边权是非负整数时 (若边权有负数, 则将所有边权全部加上边权最小值的绝对值), 若题目要求用分数表示的精确解, 可以考虑在有理数域上二分。一般的有理数域二分法超出了本文的讨论范围, 感兴趣的读者可自行学习 Stern-Brocot 树。由于问题形式限制了  $n$  的规模不会太大, 甚至只是检查负环的时间复杂度就已经达到了  $O(nm)$ , 这使得可以根据本题的特殊情况设计相对应的最优秀解法。

答案可表示为分数  $\frac{x}{y}$ , 其中  $0 \leq x \leq nV$ ,  $1 \leq y \leq n$ ,  $V$  是边权的值域。由于分母不会超过  $n$ , 所以考虑实数二分, 但用有理数表示边界。当  $mid' = \frac{L+R}{2}$  的分母大于  $n$  时, 寻找距离  $mid'$  最近的分母不超过  $n$  的有理数作为  $mid$ 。寻找最接近的有理数可以枚举每个分母计算, 时间复杂度  $O(n)$ 。也可以使用 Stern-Brocot 树做到  $O(\log^2 n)$ 。

### 3.3 Karp 算法

Richard M. Karp 在 1977 年提出了快速求解有向强连通图最小平均权回路 (最小比率环) 的算法。不同于分数规划的二分法, Karp 算法是最小比率环问题的特化解法。它利用了有向图的性质, 因此时间复杂度更优秀。

任取起点  $s$ , 设  $F_k(v)$  为从  $s$  到  $v$  经过  $k$  条边的最短路, 若不存在则为  $+\infty$ 。

引理 3.2. 若  $r_{\min}(G) = 0$ , 则

$$\min_{v \in V} \max_{k=0}^{n-1} \frac{F_n(v) - F_k(v)}{n-k} = 0$$

证明. 因为图中无负环, 所以  $s$  到  $v$  存在一条边数不超过  $n-1$  的最短路。设  $s$  到  $v$  的最短路权值为  $\pi(v)$ , 则  $F_n(v) \geq \pi(v)$  且  $\min_{k=0}^{n-1} F_k(v) = \pi(v)$ 。于是

$$\max_{k=0}^{n-1} \frac{F_n(v) - F_k(v)}{n-k} = \frac{F_n(v) - \pi(v)}{n-k} \geq 0$$

只需证明存在  $v$  使得  $F_n(v) = \pi(v)$ 。设  $C$  是一个零环,  $v$  是  $C$  上任意一点。设  $P(v)$  是  $s$  到  $v$  的最短路, 则  $P(v)C, P(v)CC, \dots$  也是  $s$  到  $v$  的最短路, 因为它们的权值为  $w(P(v)) = \pi(v)$ 。

最短路的每个前缀都是  $s$  到对应点的最短路, 所以  $P(v)C^n$  的长度为  $n$  的前缀是  $s$  到某个点  $v'$  的边数为  $n$  的最短路, 则  $v'$  满足  $F_n(v') = \pi(v')$ 。  $\square$

因此, 考虑

$$f(G) = \min_{v \in V, F_n(v) \neq +\infty} \max_{k=0}^{n-1} \frac{F_n(v) - F_k(v)}{n-k}$$

设  $\lambda = r_{\min}(G)$ , 将所有边的边权减去  $\lambda$  得到新图  $G'$ , 则  $r_{\min}(G') = 0$  且

$$f(G') = \min_{v \in V, F_n(v) \neq +\infty} \max_{k=0}^{n-1} \frac{(F_n(v) - \lambda n) - (F_k(v) - \lambda k)}{n-k} = f(G) - \lambda$$

根据引理 3.2,  $f(G') = r_{\min}(G') = 0$ , 所以  $f(G) = f(G') + \lambda = r_{\min}(G)$ 。一张有向强连通图的最小环比率可按照  $f$  的定义计算求出。

递推计算  $F_k(v)$  的时间复杂度为  $O(nm)$ , 根据  $F_k(v)$  计算  $f(G)$  的时间复杂度为  $O(n^2)$ , 算法的总时间复杂度为  $O(nm)$ 。

若图不是强连通图, 使用 Tarjan 算法做强连通分量分解, 并对每个强连通分量运行 Karp 算法。另一种解法: 从证明可以看出, 只要  $s$  可达最小比率环, 算法就能求出正确结果。因此, 新建超级源点  $s'$  向每个点连权值为 0 的边, 这不改变答案, 但使得  $s'$  一定可达最小比率环, 前提是图上存在环。

## 4 在最短路问题上的应用

最小比率环的思想可应用于一类最短路问题: 给定起点  $s$ , 求从  $s$  出发到每个点  $v$  经过恰好  $k$  条边的最短路权值。不妨规定  $s = 1$ 。

根据路径长度是否有限, 可将问题分为两类, 文章将分别进行研究。

### 4.1 路径长度有限

路径长度有限, 即  $k$  为给定值。

#### 4.1.1 矩阵快速幂解法

限制经过  $k$  条边且  $k$  较大的图论问题, 经典解法是转化为线性代数的矩阵语言。

构造  $n \times n$  的矩阵  $A_{i,j}$ , 满足第  $i$  行第  $j$  列的值为边  $i \rightarrow j$  的权值, 若不存在则为  $+\infty$ , 若有重边则取最小权。定义  $(\min, +)$  矩阵乘法  $A \times B = C$  为

$$C_{i,j} = \min(A_{i,k} + B_{k,j})$$

容易验证这样定义的矩阵乘法满足结合律, 则问题相当于计算

$$\begin{bmatrix} 0 & +\infty & +\infty & \cdots & +\infty \end{bmatrix} \times A^k$$

结果是一个  $n$  维行向量, 第  $v$  个位置的值为从  $s$  到  $v$  经过恰好  $k$  条边的最短路权值。若值为  $+\infty$ , 说明这样的路径不存在。时间复杂度  $O(n^3 \log k)$ 。



### 4.1.2 最小比率环解法

**例题 4.1** (矩阵快速幂<sup>2</sup>). 给定一张边带整数权的有向图, 可能有重边和自环. 求从 1 出发到每个点经过恰好  $k$  条边的路径权值的最小值对 998244353 取模后的结果. 若路径不存在则输出 -1.

$$1 \leq n \leq 300, 1 \leq m \leq 2n, 1 \leq k \leq 10^{64}, 1 \leq w(u, v) \leq 10^{18}.$$

这是笔者自主命制的集训队互测题.  $O(n^3 \log k)$  矩阵快速幂解法在本题并不可行, 因为精确值的数量级达到了  $10^{82}$ . 选手必须使用高精度维护权值, 导致算法常数过大, 无法通过本题.

问题可以做到几乎和  $k$  无关.

**定义 4.1** (点的最小比率环). 定义点  $v$  的最小比率环为所有经过点  $v$  的简单环中, 比率最小的一个, 记作  $C_{\min}(v)$ , 对应权值记作  $r_{\min}(v)$ . 类似定义点  $v$  的最大比率环  $C_{\max}(v)$ , 对应权值记作  $r_{\max}(v)$ .

**注意:** 必须要求环为简单环. 考虑图  $G = \{(1, 2, 1), (2, 2, 0), (2, 1, 1)\}$ ,  $C_{\min}(1) = 1 \rightarrow 2 \rightarrow 1$ . 若不要求环为简单环, 假设有限长的  $C_{\min}(1)$  存在, 则往环上添加自环  $(2, 2)$  可得比率更小的环, 与  $C_{\min}(1)$  的最小比率性质矛盾. 这样的情况在全局最小比率环上不会发生, 但证明全局最小比率环一定可以是简单环的调整法不再适用于限制经过某一点的最小比率环, 因为调整后无法保证新的最小比率环满足限制.

**定义 4.2** (点集的最小比率环). 定义点集  $V$  的最小比率环为经过  $V$  中任意一点的最小比率简单环, 记作  $C_{\min}(V)$ , 对应权值记作  $r_{\min}(V)$ . 类似定义点集  $V$  的最大比率环  $C_{\max}(V)$ , 对应权值记作  $r_{\max}(V)$ .

若最小比率环不存在 (唯一的可能是不存在经过  $v$  的环), 则认为  $r_{\min}(v) = +\infty$ .

当  $k \geq 2n^2$  时, 有以下结论:

**引理 4.1.** 考虑任意一条  $s$  到  $v$  的经过  $k$  ( $k \geq 2n^2$ ) 条边的最短路  $P(v)$ . 若  $P(v)$  存在, 设  $P(v)$  经过的所有点的集合为  $S$ .

那么, 总存在一条  $s$  到  $v$  的经过  $k$  条边的最短路  $P'(v)$  形如: 从  $s$  出发, 先走不超过  $n^2$  条边到达某个点集  $S$  的最小比率环  $C = C_{\min}(S)$ , 然后在  $C$  上不断绕整圈直到剩余步数不超过  $n^2$ , 最后走到  $v$ , 同时满足  $C$  上所有点均属于  $S$ .

证明. 调整法.

设取到最小比率环对应的点为  $x \in S$ .

任取  $x$  在路径中的某一次出现, 将路径分为前半部分和后半部分. 若前半部分的步数超过  $n$ , 根据抽屉原理, 可以在前  $n$  步找到一个简单环. 以此类推, 可以在前  $n^2$  步找到  $n$  个

<sup>2</sup><https://www.luogu.com.cn/problem/P10000>, 集训队互测 2023 第八场



独立的简单环。因为  $1 \leq |C| \leq n$ ，根据抽屉原理，可以选出这  $n$  个环的非空子集，使得它们包含的边数之和是  $|C|$  的倍数。

根据  $C$  的最小比率性质， $C$  的比率不大于经过路径上任意一点的简单环，故不大于路径上的任意简单环。将这些环替换为若干圈  $C$  的重复，路径权值和不会增大。不断替换直到前半部分的步数不大于  $n^2$ 。

类似处理后半部分步数大于  $n^2$  的情况。

替换过程中可能改变  $S$ ，因为删去简单环时可能导致  $S$  减小，且  $C$  上的所有点并不一定均属于  $S$ 。但如果替换后不满足终止条件，那么新的  $r_{\min}(S)$  一定更大：假设调整前后  $r_{\min}(S)$  不变，因为  $k > 2n^2$ ，所以调整时至少加入了一个  $C_{\min}(S)$ ，满足终止条件。因为  $r_{\min}(S)$  仅有不超过  $n$  种可能的取值 ( $r_{\min}(v)$  唯一)，所以调整总会结束。□

根据引理 4.1 设计如下解法：

预处理  $F(u, v, k)$  表示从  $u$  到  $v$  经过恰好  $k$  条边的最短路权值，其中  $0 \leq k \leq n$ ，从而对每个点  $v$  求出  $r_{\min}(v)$  的精确值  $\frac{w(C_{\min}(v))}{|C_{\min}(v)|}$ ，前提是  $r_{\min}(v)$  存在。

预处理  $f_{i,v}$  表示  $s$  走  $i$  步到  $v$  的最小权值，其中  $1 \leq i \leq n^2$ ， $1 \leq v \leq n$ 。当  $k \leq 2n^2$  时，将  $f$  预处理到  $i \leq k$  即可直接求出答案。

可以认为路径在第  $n^2 - n < i \leq n$  步进入最小比率环，因此枚举  $n^2 - n < i \leq n$ ， $1 \leq v \leq n$  且  $r_{\min}(v)$  存在，表示  $s$  走  $i$  步进入经过  $v$  的最小比率环。通过在环上绕圈直到剩余步数不超过  $n^2$ ，根据  $|C_{\min}(v)|$  求出剩余步数  $d$ ，根据  $w(C_{\min}(v))$  求出路径权值，记录在  $g_{d,v}$  中。最后根据  $g$  的所有信息递推回  $g_0$  求出答案。

$f_i \rightarrow f_{i+1}$  和  $g_{i+1} \rightarrow g_i$  单次转移的复杂度为  $O(m)$ ，总时间复杂度  $O(n^2m)$ 。特别地，枚举  $n^2 - n < i \leq n$ ， $1 \leq j \leq n$  的时间复杂度为  $O(n^2)$ ，所以结论较弱使得这部分带更多  $n$  的算法也可以勉强通过，例如额外枚举环长  $L$ 。

**注意：**因为  $k$  过大，所以需要  $\frac{ak+b}{b}$  的形式储存答案。同时，代码需要大量使用 `__int128`，常数较大。经笔者测试，将前后不在最小比率环上的路径边数的阈值设为  $\frac{n^2}{2}$  也能得到正确结果，但  $\frac{n^2}{3}$  则不行。感兴趣的读者可以尝试将阈值卡满，或证明路径两端不在最小比率环上的路径边数不超过  $\frac{n^2}{2}$ 。

该算法给出了在  $(\min, +)$  矩阵乘法下， $O(n^2c)$  计算向量乘以矩阵高次幂的结果，其中  $c$  是矩阵非空元素个数，复杂度和矩阵的幂次几乎无关。相较于常见的  $O(n^3 \log k)$  矩阵快速幂，该算法在处理稀疏矩阵时表现更优秀。这也是本题要求  $m \leq 2n$  的原因。

**例题 4.2 (Shortest Path<sup>3</sup>)**. 给定一张边带整数权的无向图，求 1 到  $n$  的长度为  $1 \sim x$  的最短路权值之和，若最短路不存在则权值为 0。答案对 998244353 取模。

$1 \leq n \leq 2 \times 10^3$ ， $0 \leq m \leq 5 \times 10^3$ ， $1 \leq x \leq 10^9$ ， $1 \leq w(i) \leq 10^9$ 。可能存在自环和重边。

沿用上一题的结论，将无向边拆成两条有向边，可以证明引理 4.1。

<sup>3</sup><https://www.luogu.com.cn/problem/P9675>，ICPC2022 济南区域赛



在引理 4.1 的证明过程中, 如果调整终止时最后一步调整的  $|C| > 2$ , 因为本题是无向边, 所以可以证明总存在长度不超过 2 的  $C$  满足引理条件。因此, 最优路径的中途一定是在某条边上来回走, 或者绕着自环转圈。后一种情况可视作前一种情况。

进一步地,  $|C| \leq 2$  使得路径前后部分的长度不超过  $2n$ , 因为只要两个简单环就一定可以进行  $C$  的替换。

对  $k \leq 4n$ , 直接求答案。对  $k > 4n$ , 枚举每条边考虑贡献: 预处理从 1 和  $n$  走  $k$  步到  $v$  的最短路权值, 可以对每个点  $v$  求出从 1 到  $n$  中途经过  $v$  且恰好走  $4n$  和  $4n+1$  步的最短路权值  $F_{4n}(v)$  和  $F_{4n+1}(v)$ 。

于是, 边  $(u, v)$  对  $k > 4n$  的答案的贡献为

$$\begin{cases} F_{4n}(v) + (k - 4n)w(u, v), & (2 \mid k) \\ F_{4n+1}(v) + (k - 4n - 1)w(u, v), & (2 \nmid k) \end{cases}$$

分成  $k$  是奇数或偶数两种情况之后, 答案来自每条边的贡献关于  $k$  是一个等差数列, 于是答案关于  $k$  可表示为若干等差数列的  $\min$ 。

求若干等差数列的  $\min$  的和的方法有很多, 例如求出凸壳后使用等差数列求和公式, 或者使用李超树维护每个  $k$  的答案, 然后二分出答案的每一段等差数列。

时间复杂度  $O(nm)$ 。

## 4.2 路径长度无限

求任意长度路径的权值最大值, 此时路径可以经过无限条边。为了避免路径权值无穷大的情况, 一般会设置衰减系数  $0 < \rho < 1$ , 表示每经过一条边, 所有边的边权就会乘以  $\rho$ 。

**注意:** 本小节讨论涉及“无穷”的概念。为保证严谨性, 需要使用一些微积分方法。

无穷长路径的权值定义涉及无穷级数。根据  $0 < \rho < 1$  可使用 Dirichlet-Abel 判别法证明级数收敛于定值, 该值即无穷长路径的权值。

无穷长路径的权值的最大值不是良定义的概念, 因为无穷长路径的数量也是无穷多的, 而无穷数集就算有上界, 也不一定存在最大值。因此, 笔者认为接下来讨论的两道例题的提问方式均不严谨。虽然可以证明一定能取到最大值, 但将“最大值”改为“上确界”会更合理一些。

### 4.2.1 倍增法

**例题 4.3** (幸福路径<sup>4</sup>)。给定一张点带实数权的有向图, 路径  $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$  的权值为  $\sum_{i=0}^k \rho^i w(v_i)$ , 求从  $v_0$  出发的路径权值的最大值, 四舍五入至小数点后一位。

$1 \leq n \leq 100, 1 \leq m \leq 10^3, 0 < \rho < 1 - 10^{-6}, 0 \leq w(v) \leq 100$ 。

<sup>4</sup><https://www.luogu.com.cn/problem/P4308>, CTSC2011

将点权转化为边权, 设原图每条边的权值为终点权值, 一条路径的权值为其所有边的权值加上路径起点的权值, 路径起点产生的贡献  $w(v_0)$  放在最后算。

题目仅要求答案保留一位小数, 所以不需要算出精确值。当路径经过足够多条边之后, 剩余部分的贡献可忽略不计。具体地, 第  $k$  条边之后的贡献不超过

$$\epsilon(k) = \sum_{i \geq k} \rho^i \max w(v) = \frac{\rho^k}{1 - \rho} \max w(v)$$

因为  $\rho < 1 - 10^{-6}$  且  $\max w(v) \leq 100$ , 所以当  $k = 2^{26}$  时,  $\epsilon(k) \leq 10^{-21}$ , 精度已经足够。也就是说, 令  $k = 2^{26}$  转化为路径长度有限的情况, 求得的答案与真实答案的差值不超过  $10^{-21}$ 。

**注意:** 由于最优路径可以只经过有限条边, 所以需要添加权值为 0 的自环, 以保证最优路径总可以经过无限条边。

虽然转移无法写成矩阵乘法的形式, 但可以借鉴矩阵快速幂的思路, 使用倍增法求解问题。设  $f_{k,u,v}$  表示从  $u$  出发经过  $2^k$  条边到  $v$  的最长路, 若不存在则为  $-\infty$ , 初始  $f_{0,u,v} = w(u, v)$  或  $-\infty$  (若  $(u, v) \notin E$ ), 有转移方程

$$f_{k+1,u,v} = \max_{p=1}^n (f_{k,u,p} + \rho^{2^k} f_{k,p,v})$$

算法的时间复杂度为  $O(n^3 \log k)$ 。

可以看出, 倍增法只在精度要求不高的情况下适用。当衰减系数  $\rho$  和权值  $w$  为有理数时, 可以证明答案也是有理数, 此时就需要使用最大比率环法计算答案了。

#### 4.2.2 最大比率环解法

**例题 4.4** (大航海时代<sup>5</sup>)。给定一张点、边均带整数权的有向图, 路径  $P = (v_0, e_1, \dots, e_k, v_k)$  的权值为

$$w(P) = q \cdot \left( \sum_{i=0}^k p^i (1 - p) \cdot w(v_i) \right) - q \cdot \left( \sum_{i=1}^k p^i w(e_i) \right)$$

其中  $p = \frac{y}{y+z}$ ,  $y, z, q$  为给定正整数。

对每个点  $s$ , 求从  $s$  出发的路径权值最大值, 用分数表示答案。

$1 \leq n \leq 50$ ,  $1 \leq m \leq 500$ ,  $1 \leq y, z, q, w(i), w(u, v) \leq 10^4$ 。

本题是上一题的加强版。

题目同时有点权和边权, 考虑将点权转化到边权上。设边  $(u, v)$  的新权值  $w'(u, v) = (1 - p)w(v) - w(u, v)$ , 则路径权值改写为

$$q \cdot \left( (1 - p)w(v_0) + \sum_{i=1}^k p^i w'(e_i) \right)$$

<sup>5</sup><https://www.luogu.com.cn/problem/P5822>, L&K Round 3, 题意经过一定程度的抽象

其中  $(1-p)w(v_0)$  只和起点有关, 可以放在最后统计, 这样就和上一题一样只剩边权了。

考虑最优路径的形态, 本题中的“最优”即“最大”。有了路径长度有限的基础, 自然的想法是路径会在经过  $O(\text{poly}(n))$  条边之后不断地在某个环上绕圈。

本题中, 特殊定义环的比率为由它不断重复得到的无限长路径的权值, 即

$$r(C) = w(C^\infty) = w(C) + p^{|C|}w(C) + p^{2|C|}w(C) + \cdots = \frac{w(C)}{1-p^{|C|}}$$

和上一题一样, 为每条边添加权值为 0 的自环不影响答案, 但使得最优路径总可以经过无限条边。在此基础上, 设  $F(s)$  表示从  $s$  出发的路径权值的最大值 (实际应定义为上确界)。

因为所有路径的权值有限, 根据实数的确界原理, 所有路径的权值在实数范围内存在上确界, 即  $F(s)$  存在。

**引理 4.2.**  $F(s)$  一定会由某条从  $s$  出发的  $\rho$  形路径取到, 即先走一段不经过重复点的路径, 然后在某个简单环上不断绕圈。这同时说明从  $s$  出发的路径权值存在最大值。

证明. 考虑  $s$  的所有出边  $(s, u)$ , 因路径长度无限, 故

$$F(s) \geq (1-p)w(s) + p(w(u, s) + F(u))$$

且存在  $u$  使得

$$F(s) = (1-p)w(s) + p(w(u, s) + F(u))$$

根据上确界的定义使用  $\epsilon - \delta$  语言容易证明该结论。

设  $out_s$  为任意一个这样的  $u$ , 则  $out_s$  存在。从  $s \rightarrow out_s$  连边得到内向基环树, 则从每个点出发下一步最优的点是  $out_s$ 。由此可知从每个点出发的最优路径是沿基环树不断移动, 即  $F(s)$  可由某条从  $s$  出发的  $\rho$  形路径取到, 否则在第一个不同的位置调整, 会导出与  $F$  值最优性的矛盾。  $\square$

**引理 4.3.** 从  $s$  出发最优路径的环部分总可以是经过某个点的最大比率环。

证明. 考虑从  $s$  出发沿内向基环树边移动得到的无限长路径  $P_s$ 。由引理 4.2,  $P_s$  是从  $s$  出发权值最大的路径, 且  $P_s$  可表示为简单路径  $P'_s$  接上无穷多个简单环  $Q_s$ 。

设  $Q_s$  的起止点为  $v$ , 设经过  $v$  的最大比率环为  $C_v = C_{\max}(v)$ 。

根据路径权值的定义可知若  $A$  长度有限,  $B$  长度任意, 则  $w(B) \geq w(B')$  当且仅当  $w(AB) \geq w(AB')$ , 所以比较两条路径的权值时, 如果它们有一段公共前缀, 则将这段前缀去掉不影响大小关系。

由  $P_s$  的最优性可知  $w(P'_s Q_s^\infty) \geq w(P'_s C_v^\infty)$ , 于是  $w(Q_s^\infty) \geq w(C_v^\infty)$ , 即  $r(Q_s) \geq r(C_v)$ 。但  $C_v$  是  $v$  的最大比率环且  $Q_s$  经过  $v$ , 所以  $r(C_v) \geq r(Q_s)$ 。

因此  $r(C_v) = r(Q_s)$ , 即  $Q_s$  是  $v$  的最大比率环。  $\square$



根据引理 4.3 设计如下算法:

首先预处理  $f(k, u, v)$  表示从  $u$  出发恰好走  $k$  条边的最长路权值, 其中  $0 \leq k \leq n$ 。由  $f$  可以算出经过每个点  $v$  的最大比率环以及对应比率  $r_v = r_{\max}(v)$ 。

对每个起点  $s$ , 枚举  $v$ , 本质上是在枚举  $n$  个可能最大比率环之一, 再枚举  $s \rightarrow v$  的简单路径长度  $k$ , 则对应权值为  $q \cdot (f(k, s, v) + p^k \cdot r_v)$ 。因此

$$ans(s) = q \cdot \max_{v=1}^n \max_{k=0}^{n-1} (f(k, s, v) + p^k \cdot r_v)$$

时间复杂度  $O(n^2 m)$ 。

关于维护精确值的问题: 本题需要使用高精度分数类维护所有值, 导致复杂度多带一个  $n^2$  因子。由于高精度 gcd 耗时极其严重, 所以无法保证在高精度分数运算过程中及时化简。为减小常数, 在预处理  $f$  时, 注意到  $f(k, u, v)$  的分母一定是  $(y+z)^{k+1}$  的因数, 所以可设  $f'(k, u, v) = f(k, u, v) \cdot (y+z)^{k+1}$ , 此时计算和维护  $f$  就只涉及高精度整数类了。仍需一定常数优化才能通过本题, 如使用压位高精度, 尽可能减少高精度运算等。

## 5 其它比率问题

处理信息学竞赛常见的比率问题的通用手段是二分法, 二分之后再根据问题的特性使用针对性算法解决。

### 5.1 最小比率背包

**例题 5.1** (Talent Show<sup>6</sup>). 给定权值为  $t_i$ , 权重为  $w_i$  的  $n$  个物品, 求选出若干物品的最大比率, 要求选中物品的权重之和不小于  $W$ 。答案保留三位小数。

$$1 \leq n \leq 250, 1 \leq W \leq 10^3, 1 \leq w_i \leq 10^6, 1 \leq t_i \leq 10^3.$$

二分答案  $mid$ , 判定能否选出若干物品使得选中物品的  $\sum t_i - mid \cdot \sum w_i \geq 0$  且  $\sum w_i \geq W$ 。

在对重量总和提出一定要求的限制下最小或最大化权值总和, 这是经典的 01 背包, 直接 DP 即可。时间复杂度  $O(nW \log \frac{t}{\epsilon})$ 。

### 5.2 最小比率生成树

**例题 5.2** (Desert King<sup>7</sup>). 给定平面上  $n$  个点的坐标  $(x_i, y_i)$  和对应高度  $z_i$ , 对  $u \neq v$ , 定义无向边  $(u, v)$  的权值  $a_e$  为  $|z_u - z_v|$ , 权重  $b_e$  为  $u, v$  之间的欧几里得距离  $\sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}$ , 求选出若干条边的最大比率, 要求选中的边恰好构成原图的生成树。边可以在平面上任意相交。答案保留三位小数。

<sup>6</sup><https://www.luogu.com.cn/problem/P4377>, USACO2018 OPEN Gold

<sup>7</sup><http://poj.org/problem?id=2728>





$$2 \leq n \leq 10^3, 0 \leq x_i, y_i < 10^4, 0 \leq z \leq 10^7.$$

二分答案  $mid$ , 判定能否选出若干条边使得它们构成生成树且  $\sum a_e - mid \cdot b_e \geq 0$ 。

令一条边的权值为  $a_e - mid \cdot b_e$ , 问题转化为求最小生成树。因为本题是完全图, 所以使用 Prim 算法即可。时间复杂度  $O(n^2 \log \frac{n}{\epsilon})$ 。

### 5.3 最大密度子图

**定义 5.1** (导出子图). 对图  $G = (V, E)$ , 定义  $V' \subseteq V$  的导出子图为  $V'$  和两端均属于  $V'$  的边构成的边集  $E'$  组成的图  $G[V'] = (V', E')$ 。

**定义 5.2** (图密度). 对非空图  $G = (V, E)$ , 定义  $G$  的密度 (Density) 为边数除以点数, 记作  $den(G) = \frac{|E|}{|V|}$ 。

**例题 5.3** (Everywhere is Sparser than Whole (Judge)<sup>8</sup>). 给定一张  $N$  个点,  $M = DN$  条边的图  $G = (V, E)$ , 判断对任意非空  $V' \subset V$ , 是否均有  $den(G[V']) < D$ , 即对所有非平凡 (非空且不等于  $G$ ) 的导出子图, 其密度是否均小于  $D$ 。

$1 \leq N, D, M \leq 5 \times 10^4$ . 保证图无重边和自环。

设  $V'$  的导出子图的边集为  $E(V')$ , 则问题等价于检查是否存在非空  $V' \subset V$  使得  $|E(V')| - D \cdot |V'| \geq 0$ 。

每选一个点有  $D$  的代价, 每选一条边有 1 的贡献, 但要求其两端的点也被选择。为最大化权值, 最优方案不会出现一条边的两端被选择, 但它没有被选择的情况, 即选中的边集等于选中点集的导出子图的边集。

这是最大权闭合子图的模型, 解法为: 构建有向二分图  $G' = (L, R, E)$ ,  $G$  的每条边对应  $G'$  的一个左部点, 每个点对应  $G'$  的一个右部点。默认选择所有边, 不选所有点, 初始权值为  $M$ 。从源点  $S$  向每个左部点连容量为 1 的边, 每个左部点向对应的两个右部点连容量为  $+\infty$  的边, 每个右部点向汇点  $T$  连容量为  $D$  的边, 用割表示一组方案: 左边的边被割掉表示不选择对应的边产生 1 的代价, 右边的边被割掉表示选择对应的点产生  $D$  的代价, 不存在  $S \rightarrow T$  的通路表示不存在一条被选择的边, 它的任意一端不在闭合子图内。割的流量即产生的总代价。

最大化权值, 即最小化总代价。而最小割等于最大流, 所以当上述网络流模型的  $S, T$  最大流小于  $M$  时, 说明最大权值大于 0,  $G$  存在密度大于  $D$  的导出子图。求最大流使用 Dinic 算法, 可以证明时间复杂度为  $O(M^{1.5})$ 。

问题出在最大流等于  $M$  的情况。此时  $S$  和左部点之间所有边的流量为零, 从  $S$  出发 BFS 得到的割对应了所有边都不选择的  $V' = \emptyset$  的不合法情况; 右部点和  $T$  之间所有边的流量为零, 从  $T$  出发 BFS 得到的割对应了所有点都选择的  $V' = V$  的不合法情况。

<sup>8</sup>[https://atcoder.jp/contests/arc161/tasks/arc161\\_f](https://atcoder.jp/contests/arc161/tasks/arc161_f), ARC161F



已知  $G$  没有密度大于  $D$  的导出子图，但无法判断是否存在密度等于  $D$  的非平凡导出子图。有两种解决方法。

**强连通分量解法：**最大流等于  $M$ ，说明存在每个点匹配  $D$  条边的完美匹配。对于每条边，从它匹配的点指向另一个点，给所有边定向得到有向图  $\vec{G}$ ，则  $G$  存在非平凡的密度等于  $D$  的导出子图当且仅当  $\vec{G}$  不是强连通图。求完美匹配的时间复杂度为  $O(M^{1.5})$ ，求强连通分量的时间复杂度为  $O(M)$ ，总时间复杂度  $O(M^{1.5})$ 。

证明. 充分性：若  $\vec{G}$  不是强连通图，则对  $\vec{G}$  做强连通分量分解，至少存在一个强连通分量（强连通分量缩点后出度为零的点）使得其中所有点的出边均指向强连通分量内部。因为每个点恰有  $D$  条出边，所以强连通分量对应  $G$  的密度等于  $D$  的非平凡导出子图。

必要性：若  $G$  存在密度为  $D$  的非平凡导出子图  $G[V'] = (V', E')$ ，则因为  $E'$  中的每一条边在  $\vec{G}$  上都只能从  $V'$  指向  $V'$ ，而  $|E'| = |V'|D$ ，所以  $\vec{G}$  不存在从  $V'$  指向  $V \setminus V'$  的边，即  $\vec{G}$  不强连通。□

**网络流解法：**考虑钦定第一条边是否选择。

- 如果选第一条边，相当于钦定不割掉这条边，即令  $S \rightarrow L_1$  的容量为无穷大。此时只要存在割掉左边任意一条边的最小割，就能说明  $G$  存在选中第一条边的密度等于  $D$  的非平凡导出子图。因为容易证明此时最大流（最小割）仍为  $M$ ，且  $S \rightarrow L_1$  没有被割说明第一条边被选中，而某条  $S \rightarrow L_i$  被割保证存在一条边没有被选中。

**引理 5.1.**  $G$  存在选中第一条边的密度等于  $D$  的非平凡导出子图，当且仅当存在  $S$  在残量网络上不可达的左部点  $L_i$ 。

证明. 充分性：将  $S$  在残量网络上可达的点集作为割的左部点集，其余作为右部点集，则得到的割为最小割且  $S \rightarrow L_1$  没有被割但  $S \rightarrow L_i$  被割，从而  $G$  存在选中第一条边的密度等于  $D$  的非平凡导出子图。

必要性：如果  $G$  存在选中第一条边的密度等于  $D$  的非平凡导出子图，那么根据导出子图和割的对应关系，存在割掉  $S \rightarrow L_i$  且流量为  $M$  的最小割。考虑任意割掉  $S \rightarrow L_i$  的最小割，这说明存在一种划分点集的方案，使得左部点集包含  $S$ ，右部点集包含  $L_i$ ，且中间所有边均不在残量网络上（最小割割边必然满流），这说明  $S$  在残量网络上不可达  $L_i$ 。□

- 如果不选第一条边，相当于钦定割掉这条边，即令  $S \rightarrow L_1$  的容量为 0。容易证明此时最大流（最小割）为  $M - 1$ 。假设新图的最小割不包含  $S \rightarrow L_1$ ，那么这个割在  $G'$  上的流量也为  $M - 1$ ，矛盾。此时只要找到一组割掉右边任意一条边的最小割，就能说明  $G$  存在不选第一条边的密度等于  $D$  的非平凡子图，因为此时的最小割对应  $G'$  流量为  $M$  的最小割， $S \rightarrow L_1$  被割说明不选第一条边，割掉  $R_i \rightarrow T$  说明选中了点  $i$ 。类

似地, 可以证明  $G$  存在不选第一条边的密度等于  $D$  的非平凡子图, 当且仅当存在右部点  $R_i$  在残量网络上不可达  $T$ 。

当最小割流量小于  $M$  时, 不存在一个割使得全割左边或全割右边, 所以不需要特殊判断最小割小于  $M$ , 即最大权闭合子图权值大于 0 的情况, 而直接使用上述最小割等于  $M$  的算法即可。

这个做法只需要两次形式类似的网络流, 且可以证明在改造后的图上运行 Dinic 算法求最大流的时间复杂度依然为  $O(M^{1.5})$ 。

**例题 5.4 (Hard Life<sup>9</sup>).** 求密度最大的非空导出子图。

$1 \leq n \leq 100, 0 \leq m \leq 10^3$ 。保证图无重边和自环。

二分答案  $mid$ 。虽然无法直接套用上一题的做法, 因为不能保证  $M = n \cdot mid$ , 但其中的思想还是可以借鉴的。

将  $\frac{|E|}{|V|} \geq mid$  转化为  $|E| - mid \cdot |V| \geq 0$ , 建出最大权闭合子图模型, 若最大流小于  $|E|$ , 说明存在密度大于  $mid$  的子图, 反之则不存在。至于是否存在密度等于  $mid$  的子图则并不重要, 这一点和上一题不一样。

求出最大密度  $den_{\max}$  后还要构造方案。为了避免求出空图作为方案, 需要令  $mid = den_{\max} - \epsilon$  而非  $den_{\max}$  再求一次最大权闭合子图以构造方案。因为若密度不同则相差至少  $\frac{1}{n^2}$ , 所以令  $\epsilon$  取小于  $\frac{1}{n^2}$  的值即可。

本题涉及实数权网络流, 所以 Dinic 算法的复杂度只能分析到  $O(|V|^2|E|) = O(m^3)$ 。但网络流常数很小, 就算结合二分也是可以接受的。最大密度子图还有一种点数  $O(n)$  的网络流建图方法, 感兴趣的读者可自行查找资料。

## 6 总结

本篇文章介绍了最小比率环的定义和求解方法, 并基于最小比率环的性质, 提出了更快解决固定起点和步数时的最短路问题: 去掉了常用矩阵快速幂算法的  $\log$  因子。文章系统地阐述了路径长度无限时的最长路问题, 并给出了解决方法。此外, 文章对一些常见比率问题进行了汇总。

因笔者水平有限, 未能对比率问题进行更深层次的研究, 仅是将它们汇总。希望文章提出的算法能够在更多场合找到用武之地, 也希望文章能够对比率问题的研究提供帮助。

## 致谢

感谢中国计算机学会提供学习和交流的平台。

<sup>9</sup><https://www.luogu.com.cn/problem/UVA1389>



感谢南京外国语学校张超老师，李曙老师，史钊镭老师对我的指导。

感谢集训队教练彭思进和杨耀良的指导。

感谢程思元同学和我关于论文内容进行了大量讨论。

感谢汤智铖同学和郭羽冲同学为本文审稿。

感谢家人对我的关心与支持。

## 参考文献

- [1] R. M. Karp, *A Characterization of the Minimum Cycle Mean in a Digraph*. Discrete Mathematics 23 (1978) 309-311.







---

全国青少年信息学奥林匹克竞赛 (CCF NOI)

E-mail: [noi@ccf.org.cn](mailto:noi@ccf.org.cn)

网址: <http://www.noi.cn>

---

