全国青少年信息学奥林匹克竞赛

IOI2019 中国国家候选队论文集

教练：张瑞喆

2019 年 5 月

目 录

两类递推数列的性质和应用

福州第三中学 钟子谦

摘 要

线性递推数列和整式递推数列是数学中常见的两类递推数列，本文介绍了这两类递推数列的定义、性质和有关算法，并展示了它们在信息学竞赛中的一些应用。

前言

线性递推数列被引入算法竞赛界已经有至少五年，但是一直没有得到特别广泛的普及。整式递推数列是线性递推数列的一个自然的拓展，近两年才被引入信息学竞赛。本文希望能够系统介绍这两类数列的性质和在信息学竞赛中的用途，使读者在思考有关问题时有迹可循。

本文首先在第 1 节介绍了线性递推数列，接下来在第 2 节介绍了整式递推数列。对于这两类数列，本文介绍了它们的定义、性质、有关算法和实际例题，对于线性递推数列本文还介绍了一些与线性代数相关的应用。

1 线性递推数列

1.1 定义

定义 1.1. 我们称长度有限的数列为**有限数列**，长度无限的数列为**无限数列**。

定义 1.2. 我们称形式幂级数 F 最高次项的次数为形式幂级数 F 的**次数**，记为 $\deg(F)$（可能为 ∞）。特别地，我们定义零多项式的次数为负无穷大（$-\infty$）。

定义 1.3. 对于有限数列 $\{a_0, a_1, a_2 \cdots a_{n-1}\}$，我们定义它的**生成函数**为多项式 $A(x) = \sum_{i=0}^{n-1} a_i x^i$。对于无限数列 $\{a_0, a_1, a_2 \cdots\}$，我们类似地定义它的**生成函数**为形式幂级数 $A(x) = \sum_{i=0}^{\infty} a_i x^i$。

定义 1.4. 对于无限数列 $\{a_0, a_1, a_2 \cdots\}$ 和有限非空数列 $\{r_0, r_1, r_2 \cdots r_{m-1}\}$，若对于任意 $p \geq m-1$，有 $\sum_{k=0}^{m-1} a_{p-k} r_k = 0$，则称数列 r 为数列 a 的**线性递归式**。若 $r_0 = 1$，我们称数列 r 为数列 a 的**线性递推式**。我们称存在线性递推式的无限数列为**线性递推数列**。

对于有限数列 $\{a_0, a_1, a_2 \cdots a_{n-1}\}$ 和有限非空数列 $\{r_0, r_1, r_2 \cdots r_{m-1}\}$，类似地，若对于任意 $m-1 \le p \le n-1$，有 $\sum_{k=0}^{m-1} a_{p-k} r_k = 0$，则称数列 r 为数列 a 的**线性递归式**。若 $r_0 = 1$，我们称数列 r 为数列 a 的**线性递推式**。

我们称这个线性递推式的**阶数**为它的长度减一，称数列 a 阶数最小的线性递推式为数列 a 的**最短线性递推式**。

1.2 基本性质和判定方法

在生成函数的观点下看线性递归式，我们有如下结论：

定理 1.1. *对于无限数列 $\{a_0, a_1, a_2 \cdots\}$ 和有限非空数列 $\{r_0, r_1, r_2 \cdots r_{m-1}\}$，设数列 a 和数列 r 所对应的生成函数为 A 和 R，数列 r 为数列 a 的线性递归式等价于存在次数不超过 $m-2$ 的多项式 S 满足 $AR + S = 0$。*

对于有限数列 $\{a_0, a_1, a_2 \cdots a_{n-1}\}$ 和有限数列 $\{r_0, r_1, r_2 \cdots r_{m-1}\}$，设数列 a 和数列 r 所对应的生成函数为 A 和 R，数列 r 为数列 a 的线性递归式等价于存在次数不超过 $m-2$ 的多项式 S 满足 $AR + S \equiv 0 \pmod{x^n}$。

证明. 下面证明无限数列的情况，有限数列的情况也是类似的。

对于 $k \ge m-1$，考察两侧 x^k 次项的系数，我们有 $[x^k](A(x)R(x)) = \sum_{j=0}^{m-1} r_j a_{k-j} = 0$。只需要取适当的 S 使得低次项系数为 0 即可。 □

接下来我们介绍几种常见的判定线性递推数列的方法。

推论 1.1. *对于无限数列 $\{a_0, a_1, a_2 \cdots\}$，设数列 a 所对应的生成函数为 A，a 为线性递推数列当且仅当存在常数项为 1 的多项式 R 和多项式 S 满足 $A = \frac{S}{R}$。数列 a 的最短线性递推式阶数就是对于这样的 R 和 S，$\max(\deg(R), \deg(S)+1)$ 的最小可能值。*

证明. 由定理1.1移项即得。 □

定理 1.2. *对于一个 $n \times n$ 的矩阵 M，无限数列 $\{I, M, M^2, M^3 \cdots\}$ 是一个线性递推数列，它的最短线性递推式阶数不超过 n。*

证明. 考虑矩阵 M 的特征多项式 p，它满足 $\deg(p) = n$，$[x^n]p(x) = 1$。由 Cayley-Hamilton 定理，我们有 $p(M) = 0$。该定理的证明可参见参考文献 [2]，由于和本文主题关系不大，这里略去。

设 $p(x) = \sum_{i=0}^{n} c_{n-i} x^i$，$p(M) = 0$ 即 $\sum_{i=0}^{n} c_{n-i} M^i = 0$，两边乘 M^j 得 $\sum_{i=0}^{n} c_{n-i} M^{i+j} = 0$，即 $\sum_{i=0}^{n} c_i M^{j+n-i} = 0$。所以 $\{c_0, c_1 \cdots c_n\}$ 即为数列 $\{I, M, M^2, M^3 \cdots\}$ 的一个阶数为 n 的线性递推式，该数列的最短线性递推式阶数不超过 n。 □

线性递推数列还满足以下的封闭性。

定理 1.3. 对于线性递推数列 $\{a_0, a_1, a_2 \cdots\}$、线性递推数列 $\{b_0, b_1, b_2 \cdots\}$、有限数列 $\{t_0, t_1 \cdots t_{m-1}\}$、常数 p，我们有：

- $\{t_i\}_{i=0}^{m-1}\{a_i\}_{i=0}^{\infty}$ 是线性递推数列。
- $\{a_i p\}_{i=0}^{\infty}$ 是线性递推数列。
- $\{a_{i+1}\}_{i=0}^{\infty}$ 是线性递推数列。
- $\{a_i + b_i\}_{i=0}^{\infty}$ 是线性递推数列。
- $\{\sum_{j=0}^{i} a_j b_{i-j}\}_{i=0}^{\infty}$ 是线性递推数列。
- $\{a_i b_i\}_{i=0}^{\infty}$ 是线性递推数列。

以上引理的证明较为简单，使用推论1.1和定理1.2即可证明，由于篇幅所限这里略去。

1.3 有关算法

1.3.1 求出一个数列的最短线性递推式

我们先考虑有限数列的情况，即我们要对一个有限数列求出最短线性递推式。假设运算均在一个域上进行。

一种简单的做法是使用高斯消元消元出最短线性递推式，对于长度为 n 的有限数列复杂度为 $O(n^3)$，而使用 Berlekamp-Massey 算法可以将时间复杂度降到 $O(n^2)$。

对于一个有限数列 $\{a_0, a_1, a_2 \cdots a_{n-1}\}$，Berlekamp-Massey 算法会对于它的每个前缀 $\{a_0, a_1, a_2 \cdots a_i\}$ 求出这一前缀的最短线性递推式 $r^{(i)}$，设 $|r^{(i)}| - 1 = l_i$，即 l_i 为线性递推式 $r^{(i)}$ 的阶数。我们显然有 $r^{(-1)} = \{1\}$，$l_{i-1} \le l_i$。

引理 1.1. 如果 $r^{(i-1)}$ 不是 $\{a_0, a_1, a_2 \cdots a_i\}$ 的最短线性递推式，那么

$$l_i \ge \max(l_{i-1}, i + 1 - l_{i-1})$$

证明. 反证法，假设 $l_i \le i - l_{i-1}$。设 $r^{(i-1)} = \{p_0, p_1 \cdots p_{l_{i-1}}\}$，$r^{(i)} = \{q_0, q_1 \cdots q_{l_i}\}$。

那么由 $r^{(i)}$ 的定义，对于 $l_i \le j \le i$ 我们都有 $a_j = -\sum_{t=1}^{l_i} q_t a_{j-t}$。

那么我们有

$$\begin{aligned} -\sum_{j=1}^{l_{i-1}} p_j a_{i-j} &= \sum_{j=1}^{l_{i-1}} p_j \sum_{k=1}^{l_i} q_k a_{i-j-k} \quad (i - l_{i-1} \ge l_i) \\ &= \sum_{k=1}^{l_i} q_k \sum_{j=1}^{l_{i-1}} p_j a_{i-j-k} \\ &= -\sum_{k=1}^{l_i} q_k a_{i-k} = a_i \end{aligned}$$

所以 $r^{(i-1)}$ 就是 $\{a_0, a_1 \cdots a_i\}$ 的最短线性递推式，矛盾。

所以 $l_i \geq i + 1 - l_{i-1}$，又由单调性即证。 □

事实上，引理1.1中的等号是可以取到的，以下我们给出这样的一种方案。

考虑定理1.1，令 A 为数列 a 的生成函数，$R^{(i)}$ 为 $r^{(i)}$ 的生成函数，那么就存在多项式 $S^{(i)}$ 使得 $AR^{(i)} \equiv S^{(i)} \pmod{x^{i+1}}$，其中 $\deg(S^{(i)}) \leq l_i - 1$。

考虑由 $R^{(i-1)}$ 推出 $R^{(i)}$。如果我们仍然有 $AR^{(i-1)} \equiv S^{(i-1)} \pmod{x^{i+1}}$，那么令 $R^{(i)}$ 等于 $R^{(i-1)}$ 即可。否则，我们有 $AR^{(i-1)} \equiv S^{(i-1)} + dx^i \pmod{x^{i+1}}$。

考虑上一次增长递推式的情形，设当时存在 $p < i$ 和 c 使得 $AR^{(p-1)} \equiv S^{(p-1)} + cx^p \pmod{x^{p+1}}$，那么将两侧乘上 $x^{i-p}dc^{-1}$，我们有 $Ax^{i-p}dc^{-1}R^{(p-1)} \equiv x^{i-p}dc^{-1}S^{(p-1)} + dx^i \pmod{x^{i+1}}$。

将上述两式相减我们就有 $A(R^{(i-1)} - x^{i-p}dc^{-1}R^{(p-1)}) \equiv S^{(i-1)} - x^{i-p}dc^{-1}S^{(p-1)} \pmod{x^{i+1}}$，所以我们令 $R^{(i)}$ 等于 $R^{(i-1)} - x^{i-p}dc^{-1}R^{(p-1)}$ 即可。

可以发现，我们这样构造出的 R 和 S 一定满足 $l_i = \max(l_{i-1}, i + 1 - l_{i-1})$。考虑归纳证明，由于在 p 处增长了递推式，由归纳假设我们就有 $l_p = p + 1 - l_{p-1}$，则 $l_i = \max(l_{i-1}, i - p + l_{p-1}) = \max(l_{i-1}, i - l_p + 1)$。

我们只需要从小到大枚举 i 并如上计算 $R^{(i)}$ 和 l_i 即可，如果 $l_i > l_{i-1}$ 就对 p 和 c 进行更新。时间复杂度 $O(n^2)$。

对于无限数列的情况，我们有如下定理。

定理 1.4. *对于线性递推数列 $\{a_0, a_1, a_2 \cdots\}$，若它的最短线性递推式阶数不超过 s，那么 $\{a_0, a_1, a_2 \cdots a_{s+s-1}\}$ 的最短线性递推式即为 a 的最短线性递推式。*

证明. 取最小的 $t \geq s+s$，满足 $\{a_0, a_1, a_2 \cdots a_{s+s-1}\}$ 的最短线性递推式 r 不是 $\{a_0, a_1, a_2 \cdots a_t\}$ 的最短线性递推式，由引理1.1，我们就有 $\{a_0, a_1, a_2 \cdots a_t\}$ 的最短线性递推式长度至少为 $t + 1 - l_t \geq t + 1 - s \geq s + s + 1 - s > s$，矛盾。 □

所以如果我们知道数列 a 最短线性递推式阶数的上界 s，我们只需要求出这个数列长度为 $2s$ 的前缀并求出它的最短线性递推式即可。

1.3.2 求出一个线性递推数列的某一项

假设我们有一个线性递推数列 $\{a_0, a_1, a_2 \cdots\}$ 满足线性递推式 $\{r_0, r_1 \cdots r_{m-1}\}$，考虑如何对于 $k \geq 0$ 求出 a_k。

考虑设 $G(F) = \sum_{i=0}^{\infty} a_i [x^i] F(x)$，那么我们就是要求 $G(x^k)$。

注意到对于多项式 a, b，我们有 $G(a+b) = G(a) + G(b)$。由线性递推式的定义，对 $t \geq 0$ 我们有 $\sum_{i=0}^{m-1} a_{m-1-i+t} r_i = 0$，即 $G((\sum_{i=0}^{m-1} x^{m-1-i} r_i) x^t) = 0$。所以设 $S(x) = \sum_{i=0}^{m-1} x^{m-1-i} r_i$，我们就有 $G(S(x)x^t) = 0$ $(\forall t \geq 0)$，又因为 $G(a+b) = G(a) + G(b)$，我们就有对任意多项式 $T(x)$，$G(S(x)T(x)) = 0$。

考虑把 x^k 和 $S(x)$ 做带余除法，即设 $x^k = S(x)U(x) + R(x)$，其中 U,R 为多项式且 $\deg(R) < \deg(S)$。我们有 $G(S(x)U(x)) = 0$，所以 $G(x^k) = G(x^k - S(x)U(x)) = G(R(x)) = G(x^k \bmod S(x))$。我们只需要类似快速幂地倍增 k，每次把多项式对 $S(x)$ 取模。$x^k \bmod S(x)$ 的次数不超过 $m-2$，我们再由定义带入 a 的前 $m-1$ 项求出 G 即可。

求两个次数 $O(m)$ 的多项式取模结果在模域下可以做到 $O(m\log(m))$ 的时间复杂度（可以参见参考文献 [4]），那么这个问题就可以在 $O(m\log(m)\log(k))$ 的时间复杂度内解决。

1.4 常见应用

由于定理1.2，线性递推数列在与线性代数有关的问题中十分常见，本节提出一些常见的应用。下文中无特别说明，均假设运算在模某个大质数 p 下进行。

1.4.1 求向量列和矩阵列的最短递推式

考虑如何求出 n 维行向量列 $\{t_0,t_1,t_2\cdots\}$ 的线性递推式。假设考虑在模 p 意义下随机一个 n 维列向量 v，转而计算 $\{t_0v,t_1v,t_2v\cdots\}$ 这个标量序列的最短线性递推式。

由 Schwartz-Zippel 引理（可参见参考文献 [5]），我们可以推出有至少 $1-\frac{n}{p}$ 的概率，$\{t_0v,t_1v,t_2v\cdots\}$ 的最短线性递推式就是 $\{t_0,t_1,t_2\cdots\}$ 的最短线性递推式，所以我们只需要用前述的方法求出最短线性递推式即可。

求矩阵列的最短递推式也是类似的，对于 n 行 m 列的矩阵列 $\{t_0,t_1,t_2\cdots\}$ 的线性递推式，考虑在模 p 意义下随机一个 n 维列向量 u 和一个 m 维行向量 v，转而计算 $\{ut_0v,ut_1v,t_2v\cdots\}$ 这个标量序列的最短线性递推式。由 Schwartz-Zippel 引理，我们可以类似地推出有至少 $1-\frac{n+m}{p}$ 的概率它们的最短线性递推式相同。

1.4.2 求矩阵的最小多项式

$n\times n$ 的矩阵 M 的最小多项式是次数最小的使得 $f(M)=0$ 的多项式 f。

类似定理1.2的证明，考虑 $\{I,M,M^2\cdots\}$ 的线性递推式 $\{r_0,r_1,r_2\cdots r_m\}$，那么我们有 $\sum_{i=0}^{m} r_{m-i}M^i = 0$，所以 $f(x) = \sum_{i=0}^{m} r_{m-i}x^i$ 即为矩阵 M 的一个零化多项式。所以矩阵 M 的最小多项式就对应着 $\{I,M,M^2\cdots\}$ 的最短线性递推式，而由定理1.2它的阶数不超过 n，我们使用上一节中的方法求出最短线性递推式即可。

具体地，对于 n 维随机向量 u,v 我们需要求出 $\{uv,uMv,uM^2v\cdots uM^{2n}v\}$。注意到 $uM^{i+1} = (uM^i)M$，而向量乘上矩阵是可以在 $O(n^2)$ 时间内计算的，所以我们可以在 $O(n^3)$ 时间内从前往后递推出 $\{u,uM,uM^2\cdots uM^{2n}\}$，接下来再把每一项乘上 v 即可。时间复杂度 $O(n^3)$。

如果 M 是稀疏矩阵，假设其中有 e 个非零位置，那么向量乘上 M 的结果就可以在 $O(n+e)$ 的时间内计算，我们就可以在 $O(n(n+e))$ 的时间内求出 $\{uv,uMv,uM^2v\cdots uM^{2n}v\}$，

从而在 $O(n(n+e))$ 的时间内求出它的最小多项式。

1.4.3 优化动态规划

一类常见的动态规划问题具有如下的递推关系式：$f(i,j) = \sum_{t=0}^{m-1} f(i-1,t)c(t,j)$ $(i \geq 1, j \in [0,m))$，给定 $f(0,j)$ $(j \in [0,m))$，需要求出 $f(n,j)$ $(j \in [0,m))$。

记 $F(i)$ 为 $f(i,j)$ $(j \in [0,m))$ 所对应的行向量，C 为 c 所对应的 $m \times m$ 矩阵，那么我们有 $F(i) = F(i-1)C$ $(i \geq 1)$，所以 $F(n) = F(0)C^n$。常见的优化方法是使用矩阵乘法快速幂求出 C^n，之后乘上 $F(0)$ 得到结果，复杂度 $O(m^3 \log(n))$。

由定理1.2我们有 $\{C^n\}_n$ 是线性递推数列，所以 $\{F(n)\}_n$ 也是线性递推数列，用上述的方法求出 $\{F(0), F(1), F(2) \cdots\}$ 的最短线性递推式后我们再用节1.3.2中的方法求出 $F(n)$ 即可，时间复杂度 $O(m^3 + m\log(m)\log(n))$（求 $\{F(0), F(1), F(2) \cdots F(n+n-1)\}$ 需要 $O(m^3)$ 的时间复杂度）。

1.4.4 解稀疏线性方程组

有一个 $n \times n$ 的满秩矩阵 A 和一个长度为 n 的行向量 b，我们需要求出一个长度为 n 的行向量 x 满足 $Ax = b$。

由于 A 是满秩的，我们有 $x = A^{-1}b$，其中 A^{-1} 表示 A 的逆矩阵。

考虑求出 $\{b, Ab, A^2b, A^3b \cdots\}$ 的最短递推式 $\{r_0, r_1, r_2 \cdots r_{m-1}\}$，由定理1.2这样的递推式一定存在且阶数不超过 n，那么我们有 $\sum_{i=0}^{m-1} A^i b r_{m-1-i} = 0$，注意到 $r_{m-1} \neq 0$（否则去掉 r_{m-1} 即为一个更短的递推式），我们在两边乘上 A^{-1} 就有 $\sum_{i=0}^{m-1} A^{i-1} b r_{m-1-i} = 0$，所以 $A^{-1}b = -\frac{1}{r_{m-1}}(\sum_{i=0}^{m-2} A^i b r_{m-2-i})$。

瓶颈在于求 $\{b, Ab, A^2b, A^3b \cdots A^{2n}b\}$。假设 A 中有 e 个非零位置，那么我们从前往后递推地求出这个向量序列的时间复杂度为 $O(ne)$，总的时间复杂度就为 $O(n(n+e))$。

1.4.5 求稀疏矩阵行列式

对于输入的 $n \times n$ 的满秩矩阵 A，我们需要求出 $\det(A)$。

注意到我们可以快速地求出稀疏矩阵的最小多项式（见节1.4.2），而当矩阵的每个特征值的几何重数均为一时最小多项式就是特征多项式。对于 n 阶矩阵，特征多项式的常数项乘上 $(-1)^n$ 即为行列式（因为行列式即全部特征值的乘积）。

由于 A 不一定满足该性质，考虑将输入矩阵乘上一个新的矩阵 B。我们取 B 为一个 $n \times n$ 的模 p 意义下的随机对角矩阵，那么我们可以证明 AB 有至少 $1 - \frac{2n^2-n}{p}$ 的概率满足该性质[1]。求出 $\det(AB)$ 后注意到 $\det(AB) = \det(A)\det(B)$，而 $\det(B)$ 就是对角线上各个元

[1]事实上，所有特征值的代数重数均为 1 的概率至少为 $1 - \frac{2n^2-n}{p}$，证明可参见 [7] 定理 4.3

素的乘积，相除即可得到 $\det(A)$ 的值。

设 A 中有 e 个非零位置，该做法的时间复杂度即为 $O(n(n+e))$。

1.4.6 求稀疏矩阵秩

对于输入的 $n \times m$ 的矩阵 A，我们需要求出 $rank(A)$。

注意到一个 $n \times n$ 矩阵的秩即为矩阵的阶数 n 减去特征值 0 的代数重数，所以如果我们求出了这个矩阵的特征多项式 $f(x)$，只需要除去其中的所有因子 x，剩余多项式的次数就是矩阵的秩。如果矩阵的特征多项式等于最小多项式乘上一个 x 的次幂，那么我们只需要求出最小多项式并除去其中的所有因子 x 即可。

考虑先把矩阵 A 转化为一个 $n \times n$ 的对称矩阵。我们在模 p 意义下随机生成一个 $m \times m$ 的对角矩阵 D_1，矩阵 AD_1A^T 就是一个 $n \times n$ 的对称矩阵，并且它的秩有至少 $1 - \frac{n^2}{p}$ 的概率与矩阵 A 相同[2]。接着我们在模 p 意义下随机生成一个 $n \times n$ 的对角矩阵 D_2，$D_2AD_1A^TD_2$ 这一矩阵就有至少 $1 - \frac{4n^2}{p}$ 概率满足特征多项式等于最小多项式乘上一个 x 的次幂[3]。我们求出它的最小多项式并计算即可。

求最小多项式时我们需要对一个向量乘上 $D_2AD_1A^TD_2$，由于 D_2, D_1, A, A^T 都是稀疏的，我们只需要依次乘入即可。设 A 中有 e 个非零位置，该做法的时间复杂度为 $O(n(n+e))$。

1.5 例题

例题 1. *Gapless Filling with Tetrominoes*[4]

有一个 n 行 4 列的网格，你需要用 n 个俄罗斯方块（即大小为四的格子联通块）来覆盖它，每个格子必须被覆盖恰好一次。输出方案数对输入的质数 p 取模的值。$1 \le n \le 10^9$，$2 \le p \le 10^9 + 9$。

考虑状态压缩动态规划，记 $f[t][S]$ 表示当前考虑了前 t 行，最后三行的格子中当前没被覆盖的集合为 S，转移时可以枚举最下方的格子在行 t 中的俄罗斯方块集合，$f[n][\{\}]$ 的值即为所求的答案。

由于第二维的状态数较多，官方题解的做法是找出所有可达的状态并使用矩阵乘法转移。类似节1.4.3，由定理1.2我们有 $\{f[u][\{\}]\}_{u=0}^{\infty}$ 为线性递推数列，我们使用节1.3.1中的算法求出它的最短线性递推式并使用节1.3.2中的算法转移即可，这个做法明显不劣于官方题解的做法（如果官方题解的做法将第二维大小压缩为了 k，那么我们就得到了一个 $O(k)$ 阶线性递推式）。实际测试中求出了一个 34 阶递推式。

[2]设 $N(A)$ 为 A 的零空间，首先注意到 $N(A) \subseteq N(AD_1A^T)$，然后取 $N(AD_1A^T)$ 的一组基并利用 Schwartz-Zippel 引理即可证明其有高概率在 $N(A)$ 内，所以就有 $N(AD_1A^T) = N(A)$

[3]证明可参见 [7] 定理 4.7

[4]来源：Bytedance Camp 2019 Day3 Division A, Problem G, 有改动

例题 2. *Expected Value*[5]

有一张简单无向连通平面图，有一枚棋子开始在 1 号点，每步棋子会从与当前点相连的边中等概率选择一条移到出边指向的点，求棋子到达 n 号点的期望步数，对输入的质数 p 取模输出。$1 \le n \le 2000$，$10^9 < p < 1.01 \times 10^9$。

由欧拉定理的推论，我们有该图的边数 m 不超过 $3n-6$。[8]

我们记 f_x 为从 x 点出发到达 n 点的期望步数，d_x 为点 x 的度数，那么我们就有

$$f_x = \begin{cases} 1 + \sum_{(x,y)\in E} \frac{f_y}{d_x} & (x \neq n) \\ 0 & (x = n) \end{cases}$$

其中 E 为图的边集。注意到这个方程组对应的系数矩阵只有 $O(n+m)$ 个位置非零，我们套用节1.4.4中的做法即可，复杂度 $O(n^2)$。

另外本题也有一个不同的做法，可以参见参考文献 [9]。

2 整式递推数列[6]

2.1 定义

我们同定义1.1一样定义**有限数列**、**无限数列**、形式幂级数的**次数**和**生成函数**。设运算在域 $\mathbb{K}$ 上进行。

定义 2.1. *对于无限数列 $\{a_0, a_1, a_2 \cdots\}$ 和有限非空多项式数列 $\{P_0, P_1, P_2 \cdots P_{m-1}\}$，若 P_0 非零且对于任意 $p \ge m-1$，有 $\sum_{k=0}^{m-1} a_{p-k} P_k(p) = 0$，则称数列 P 为数列 a 的*__整式递推式__*。我们称存在整式递推式的无限数列为__整式递推数列__。*

对于有限数列 $\{a_0, a_1, a_2 \cdots a_{n-1}\}$ 和有限非空多项式数列 $\{P_0, P_1, P_2 \cdots P_{m-1}\}$，若 P_0 非零且对于任意 $m-1 \le p \le n-1$，有 $\sum_{k=0}^{m-1} a_{p-k} P_k(p) = 0$，则称数列 P 为数列 a 的__整式递推式__。

我们称整式递推式 $\{r_0, r_1, r_2 \cdots r_{m-1}\}$ 的__阶数__为 $m-1$，__次数__为 $max_{i=0}^{m-1} \deg(r_i)$。

定义 2.2. *我们称一个形式幂级数 $A(x)$ 为微分有限的[7]当且仅当存在多项式数列 $\{Q_0(x), Q_1(x) \cdots Q_{m-1}(x)\}$，满足 $Q_{m-1} \neq 0$ 且 $\sum_{i=0}^{m-1} Q_i(x) A^{(i)}(x) = 0$（其中 $A^{(i)}(x)$ 为 $A(x)$ 关于 x 的 i 阶导数）。我们称一个形式幂级数 $A(x)$ 为代数形式幂级数当且仅当它在 $\mathbb{K}(x)$[8]上是代数的（即 $A(x)$ 为系数在 $\mathbb{K}(x)$ 内的多项式方程的根）。*

[5]来源：Ildar Gainullin Contest 1, Problem E, 有简化

[6]P-recursive sequences

[7]D-finite

[8]$\mathbb{K}$ 上的有理分式域

2.2 基本性质和判定方法

我们首先给出整式递推数列生成函数的性质。

定理 2.1. *设一个数列 $\{a_0, a_1, a_2 \cdots\}$ 的普通生成函数为 $A(x)$，a 为整式递推数列当且仅当 A 为微分有限的。*

证明. 充分性：由于 $x^j A^{(i)}(x) = \sum_{n=0}^{\infty} a_{n+i-j} x^n \prod_{t=1}^{i}(n+t-j)$（视 a 中负下标的位置值为 0），而 $Q_i(x)A^{(i)}(x) = \sum_{j=0}^{\deg(Q_i(x))} x^j A^{(i)}(x)[x^j]Q_i(x)$，将之带入 $\sum_{i=0}^{m-1} Q_i(x)A^{(i)}(x) = 0$，对于充分大的 n，我们在两边取 x^n 项系数即可得到一个 a 的整式递推式。

必要性：把递推式写成 $\sum_{k=0}^{m-1} a_{p+k} P'_k(p) = 0 \ (\forall p \geq 0)$，对于每个多项式 $P'_i(n)$，我们可以把它从高次项到低次项逐位表示为 $\{\prod_{t=0}^{j-1}(n+i-t)\}_{j=0}^{\infty}$ 的线性组合，即写成 $P'_i(n) = \sum_{j=0}^{\deg(P'_i(n))} c_{i,j} \prod_{t=0}^{j-1}(n+i-t)$，而我们有 $x^i \sum_{n=0}^{\infty} a_{n+i} x^n \prod_{t=0}^{j-1}(n+i-t) = \sum_{n=i}^{\infty} a_n x^n \prod_{t=0}^{j-1}(n-t) = x^j A^{(j)}(x) - \sum_{n=0}^{i-1} a_n x^n \prod_{t=0}^{j-1}(n-t)$，那么 $0 = x^m \sum_{p=0}^{\infty} x^p \sum_{k=0}^{m-1} a_{p+k} P'_k(p) = \sum_{k=0}^{m-1} x^{m-k} \sum_{j=0}^{\deg(P'_k(n))} c_{k,j} x^k \sum_{p=0}^{\infty} x^p a_{p+k} \prod_{t=0}^{j-1}(n+k-t) = \sum_{k=0}^{m-1} x^{m-k} \sum_{j=0}^{\deg(P'_k(n))} c_{k,j}(x^j A^{(j)}(x) - \sum_{n=0}^{k-1} a_n x^n \prod_{t=0}^{j-1}(n-t))$，整理后即可得到一个形如 $\sum_{i=0}^{m-1} Q'_i(x)A^{(i)}(x) = S(x)$ 的等式，其中 S 为一个多项式。若 $S = 0$ 即证，否则我们将两侧求 $\deg(S(x)) + 1$ 阶导，由链式法则即得。 □

接下来我们考虑一种常见的生成函数：代数形式幂级数。我们可以证明代数形式幂级数均为微分有限的，事实上我们有更强的结论定理2.2。为了证明该结论，我们先引入两条引理。

引理 2.1. *形式幂级数 $u(x)$ 微分有限当且仅当 $\dim_{\mathbb{K}(x)} span_{\mathbb{K}(x)}\{u(x), u'(x), ...\}$ 有限。*

证明. 充分性：设 $\dim_{\mathbb{K}(x)} span_{\mathbb{K}(x)}\{u(x), u'(x), ...\} = d$，那么 $u, u', \cdots u^{(d)}$ 在 $\mathbb{K}(x)$ 上线性相关，取一个这样的和为 0 的非零组合，并乘上公分母使得分母均为 1 即证。

必要性：对于 $\sum_{i=0}^{m-1} Q_i(x)u^{(i)}(x) = 0$，我们有 $u^{(m-1)}(x) = -\sum_{i=0}^{m-2} u^{(i)}(x)\frac{Q_i(x)}{Q_{m-1}(x)}$，所以 $u^{(m-1)}(x) \in span_{\mathbb{K}(x)}\{u(x), u'(x) \cdots u^{(m-2)}(x)\}$。接下来对 $t \geq m-1$ 归纳，假设我们在 $\mathbb{K}(x)$ 中用 $u, u', \cdots u^{(t-1)}$ 表出了 $u^{(t)}$，我们在两侧由链式法则关于 x 求导并移项，就可以把 $u^{(t+1)}$ 写成 $u, u', \cdots u^{(t)}$ 的线性组合，由此可以归纳出 $u^{(t+1)}(x) \in span_{\mathbb{K}(x)}\{u(x), u'(x) \cdots u^{(m-2)}(x)\}$。所以 $span_{\mathbb{K}(x)}\{u(x), u'(x), ...\} \subseteq span_{\mathbb{K}(x)}\{u(x), u'(x) \cdots u^{(m-2)}(x)\}$，维数有限。 □

引理 2.2. *以 x 为变量的代数形式幂级数 u 关于 x 的微分 u' 也是代数形式幂级数。*

证明. 设 $P(x,u)$ 为 u 在 $\mathbb{K}(x)$ 上的最小多项式，那么我们就有 $P(x,u) = 0$，把两侧关于 x 求偏导，由链式法则我们有：

$$0 = \frac{d}{dx}P(x,u) = \left.\frac{\partial P(x,y)}{\partial x}\right|_{y=u} + u' \left.\frac{\partial P(x,y)}{\partial y}\right|_{y=u}$$

那么我们就有 $u' = -\frac{\left.\frac{\partial P(x,y)}{\partial x}\right|_{y=u}}{\left.\frac{\partial P(x,y)}{\partial y}\right|_{y=u}}$，所以它在 $\mathbb{K}(x)$ 上也是代数的。 □

定理 2.2. *设 u 为微分有限的形式幂级数，v 为一个代数形式幂级数，那么如果 $u(v(x))$ 为形式幂级数*[9]*，那么它是微分有限的。*

证明. 设 $y = u(v(x))$，考虑证 $y, y', y'' \cdots$ 在 $\mathbb{K}(x)$ 中线性相关。

由归纳法和链式法则容易证明，对于每个非负整数 i，我们都有 $y^{(i)}$ 可以表示为 $u(v(x)), u'(v(x)) \cdots$ 系数在 $K[v', v'', v''' \cdots]$ 中的线性组合。由引理2.2的证明我们有 $v^{(i)} \in \mathbb{K}(x, v)$，所以 $K[v', v'', v''' \cdots] \subseteq \mathbb{K}(x, v)$，所以 $y, y', y'' \cdots \in span_{\mathbb{K}(x,v)}\{u(v(x)), u'(v(x)), ...\}$。设 $span_{\mathbb{K}(x,v)}\{u(v(x)), u'(v(x)), ...\} = V$。

由于 u 微分有限，我们有 $\dim_{\mathbb{K}(x)} span_{\mathbb{K}(x)}\{u(x), u'(x), ...\}$ 有限（引理2.1），所以 $\dim_{\mathbb{K}(x,v)} V$ 有限，所以 $d = \dim_{\mathbb{K}(x)} V = [\mathbb{K}(x, v) : \mathbb{K}(x)] \dim_{\mathbb{K}(x,v)} V$ 有限，那么 $dim_{\mathbb{K}(x)}\{y, y', y'' \cdots y^{(d)}\} \le d$ 也有限，由引理2.1即证。 □

类似线性递推数列，整式递推数列也满足以下的封闭性。

定理 2.3. *对于整式递推数列 $\{a_0, a_1, a_2 \cdots\}$、整式递推数列 $\{b_0, b_1, b_2 \cdots\}$、有限数列 $\{t_0, t_1 \cdots t_{m-1}\}$、常数 p，我们有：*

- $\{t_i\}_{i=0}^{m-1}\{a_i\}_{i=0}^{\infty}$ *是整式递推数列。*
- $\{a_i p\}_{i=0}^{\infty}$ *是整式递推数列。*
- $\{a_{i+1}\}_{i=0}^{\infty}$ *是整式递推数列。*
- $\{a_i + b_i\}_{i=0}^{\infty}$ *是整式递推数列。*
- $\{\sum_{j=0}^{i} a_j b_{i-j}\}_{i=0}^{\infty}$ *是整式递推数列。*
- $\{a_i b_i\}_{i=0}^{\infty}$ *是整式递推数列。*

证明. 前三条由定义式不难证明。

第四条：设 u 和 v 分别为 a 和 b 的生成函数，那么 $\{a_i + b_i\}_{i=0}^{\infty}$ 的生成函数就是 $u + v$。对于形式幂级数 t，记 $V_t = span_{\mathbb{K}(x)}\{t, t', t'' \cdots\}$，那么 $V_{u+v} \subseteq V_u \oplus V_v$，其中 $\oplus$ 表示直和，我们就有 $\dim_{\mathbb{K}(x)} V_{u+v} \le \dim_{\mathbb{K}(x)}(V_u \oplus V_v) \le \dim_{\mathbb{K}(x)} V_u + \dim_{\mathbb{K}(x)} V_v$（引理2.1）有限。

第五条：同样设 u 和 v 分别为 a 和 b 的生成函数，那么 $\{\sum_{j=0}^{i} a_j b_{i-j}\}_{i=0}^{\infty}$ 的生成函数就是 uv。对于形式幂级数 t，记 $V_t = span_{\mathbb{K}(x)}\{t, t', t'' \cdots\}$。我们定义线性变换 $\varphi : V_u \otimes V_v \to \mathbb{K}((x))$ 满足 $\varphi(y \otimes z) = yz$（$y \in V_u, z \in V_v$），那么我们使用积法则对 uv 求导，不难归纳得到 $V_{uv} \subseteq im(\varphi)$，所以我们有 $\dim_{\mathbb{K}(x)} V_{uv} \le \dim_{\mathbb{K}(x)}(V_u \otimes V_v) \le \dim_{\mathbb{K}(x)} V_u \dim_{\mathbb{K}(x)} V_v$（引理2.1）有限。

第六条的证明较为复杂，可以参见 [10] 的定理 6.4.12。 □

[9] 当 $v(0) \neq 0$ 时，$u(v(x))$ 的常数项可能不存在或不收敛

2.3 有关算法

2.3.1 求出一个数列的整式递推式

由于整式递推式的次数未知，所以一般我们会枚举或推导出整式递推式的次数 s，然后再求出该次数下阶数最小的整式递推式。

回顾定义，阶数 m 的整式递推式 $\{P_0, P_1, P_2 \cdots P_m\}$ 满足对任意 $m \le p \le n-1$ 有 $\sum_{k=0}^{m} a_{p-k}P_k(p) = 0$，考虑设 $Q_i(x) = P_i(x+m)$，它就满足对任意 $0 \le p \le n-m-1$ 有 $\sum_{k=0}^{m} a_{p+k}Q_k(p) = 0$。假设阶数不超过 m_0，我们记 $t_{k,u} = [x^u]Q_k(x)$，就得到了 $(m_0+1)(s+1)$ 个变量。我们求出前 $n \ge (s+2)(m_0+1)$ 项，接下来取 $0 \le p \le n-m_0-1$ 列方程，就得到了 $n-m_0$ 个方程。需要注意的是，原矩阵不满秩（将 $\{Q_0, Q_1, Q_2 \cdots Q_m\}$ 均乘上一个常数显然仍然满足条件），而我们实际需要的是一个 $Q_m \neq 0$ 的解，所以我们可以按 $t_0, t_1 \cdots t_{m_0}$ 的顺序作为主元消除，并给第一个自由元赋上非零值，取它所在的 t_x 的 x 作为阶数。求出 $\{Q_0, Q_1, Q_2 \cdots Q_m\}$ 后用二项式定理即可还原回 $\{P_0, P_1, P_2 \cdots P_m\}$，时间复杂度 $O(n^2ms)$。

在随机数据的情况下，取足够大的 n 该算法就有很大概率求出正确的整式递推式。

2.3.2 求出一个整式递推数列的某一项

假设有一个整式递推数列 $\{a_0, a_1, a_2 \cdots\}$ 满足一个阶数为 m 的整式递推式 $\{P_0, P_1 \cdots P_m\}$，已知 $a_0, a_1 \cdots a_{m-1}$，设这个整式递推式的次数为 d，考虑如何对于 $n \ge 0$ 求出 a_n。

首先假设 $P_0(x)$ 对 $m \le x \le n$ 的整数 x 均不为 0（否则无法求 $P_0(x)$ 的逆，从而无法求出 a_x）。对 $p \ge m$，我们有 $a_p = -\frac{1}{P_0(p)} \sum_{j=1}^{m} a_{p-j}P_j(p)$。我们从前往后递推即可求出 a_n，时间复杂度 $O(nmd)$（这里忽略了求逆元的复杂度）。

当 m 和 d 远小于 n 时，我们还有更加优秀的做法。考虑先求出 $\prod_{i=m}^{n} P_0(i)a_n$，再除以 $\prod_{i=m}^{n} P_0(i)$ 即可得到 a_n。

当 $n \le m-1$ 时我们直接输出 a_n 即可，下设 $n \ge m$，我们设 $u_i = \{a_j \prod_{t=0}^{i-1} P_0(t+m)\}_{j=i}^{i+m-1}$，那么 $u_{n-m+1} = \{a_j \prod_{t=0}^{n-m} P_0(t+m)\}_{j=n-m+1}^{n}$，所以只需要求出 u_{n-m+1} 就求出了 $\prod_{i=m}^{n} P_0(i)a_n$。设 $n' = n-m+1$。

注意到 $P_0(p)a_p = -\sum_{j=1}^{m} a_{p-j}P_j(p)$，所以我们可以简单地用 u_i 递推出 u_{i+1}，考虑把这种关系用矩阵刻画。设 $M(i)$ 为一个满足 $u_iM(i) = u_{i+1}$ 的矩阵，那么我们可取

$$M(i) = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & -P_m(i+m) \\ P_0(i+m) & 0 & \cdots & 0 & 0 & -P_{m-1}(i+m) \\ 0 & P_0(i+m) & \cdots & 0 & 0 & -P_{m-2}(i+m) \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & P_0(i+m) & 0 & -P_2(i+m) \\ 0 & 0 & \cdots & 0 & P_0(i+m) & -P_1(i+m) \end{bmatrix}$$

那么 $u_{n'}=u_0\prod_{i=0}^{n'-1}M(i)$，我们只需要能求出 $\prod_{i=0}^{n'-1}M(i)$ 即可。

考虑把 $M(x)$ 当做一个以 x 为自变量的函数，返回一个 $m\times m$ 的矩阵，每个元素为次数不超过 d 的多项式。考虑设一个阀值 S，我们求出 $\prod_{i=0}^{S-1}M(Sx+i)$ 这一矩阵，它的每个元素是次数不超过 dS 的多项式。我们求出这个矩阵在 $x=0,1\cdots\lfloor\frac{n'-S}{S}\rfloor$ 处的取值（即每个元素在这些位置取点值），相乘之后再乘上若干项零散的 M（$\prod_{i=S\lfloor\frac{n'}{S}\rfloor}^{n'-1}M(i)$），我们就得到了欲求的 $\prod_{i=0}^{n'-1}M(i)$。

注意到我们并不需要多项式的每项系数，而只需要点值，所以考虑只进行点值计算，而不还原成每项系数。设 $T_w(x)=\prod_{i=0}^{w-1}M(x+i)$，考虑对于某些 w 求出 $T_w(0),T_w(S),T_w(2S)\cdots T_w(dwS)$。有了这 $dw+1$ 个点值，矩阵每个位置上的 dw 次多项式就唯一确定了。欲求的即为 $w=S$ 的情形。

考虑倍增 w 的值，假设有了 $T_w(0),T_w(S)\cdots T_w(dwS)$，考虑如何求出 $T_{2w}(0),T_{2w}(S)\cdots T_{2w}(2dwS)$。注意到 $T_{2w}(x)=T_w(x)T_w(x+w)$，所以我们只需要求出 $T_w((dw+1)S),T_w((dw+2)S)\cdots T_w(2dwS)$ 和 $T_w(w),T_w(w+S),T_w(w+2S)\cdots T_w(w+2dwS)$，我们就可以直接相乘得到 $T_{2w}(0),T_{2w}(S)\cdots T_{2w}(2dwS)$（这里是矩阵乘法）。

我们这里实际上需要做一个类似多项式平移的操作。考虑这样一个问题：假设对于 d 次多项式 h 我们已知 $h(0),h(1)\cdots h(d)$，考虑如何求 $h(k),h(1+k)\cdots h(d+k)$。考虑进行拉格朗日插值，对于每个 $m\in[0,d]$ 我们有：

$$\begin{aligned}h(m+k)&=\sum_{i=0}^{d}h(i)\prod_{j=0,i\neq j}^{d}\frac{m+k-j}{i-j}\\&=\left(\prod_{j=0}^{d}(m+k-j)\right)\left(\sum_{i=0}^{d}\frac{h(i)}{i!(d-i)!(-1)^{d-i}}\cdot\frac{1}{m+k-i}\right)\end{aligned}$$

求 $\prod_{j=0}^{d}(m+k-j)=\prod_{j=m+k-d}^{m+k}j$ 只需要预处理 $[k-d,k+d]$ 的前缀积，后面一部分是一个卷积的形式，所以我们可以运用快速傅里叶变换在 $O(d\log(d))$ 内完成这个过程。

我们考虑矩阵中的每个元素，我们令 $H(x)$ 为 $T_w(Sx)$ 中的这一元素，那么就相当于给定了 $H(0),H(1),H(2)\cdots H(dw)$，求 $H(dw+1),H(dw+2)\cdots H(2dw)$ 和 $H(\frac{w}{S}),H(\frac{w}{S}+1)\cdots H(\frac{w}{S}+2dw)$，调用上面的过程即可。

由 $T_w(0),T_w(S),T_w(2S)\cdots T_w(dwS)$ 求出 $T_{w+1}(0),T_{w+1}(S),T_{w+1}(2S)\cdots T_w(d(w+1)S)$ 很容易，只需要添上缺少的项即可。那么我们类似快速幂不断倍增 w，就可以求出 $T_S(0),T_S(S),T_S(2S)\cdots T_S(dS^2)$。

取 S 为满足 $dS\geq\frac{n'-S}{S}$ 的最小 S，我们有 $S=O(\sqrt{\frac{n}{d}})$，由主定理可算出时间复杂度为 $O\left(\sqrt{nd}\,(m^3+m^2\log(nd))\right)$。

当 $P_0\neq 1$ 时我们还需要将求出的 $\prod_{i=m}^{n}P_0(i)a_n$ 除以 $\prod_{i=m}^{n}P_0(i)$。$\prod_{i=m}^{n}P_0(i)$ 也可以类似地求，即考虑数列 $a_0=1$，$a_t=a_{t-1}P_0(m+t-1)$ $(t\geq 1)$，a_{n-m+1} 即为 $\prod_{i=m}^{n}P_0(i)$。重新调用上述过程即可，由于阶数只有 1 不是复杂度的瓶颈。总的时间复杂度就为 $O\left(\sqrt{nd}\,(m^3+m^2\log(nd))\right)$。

2.4 例题

例题 3. *Chef and Same Old Recurrence* 2[10]

给定常量 K, A, B，定义递推数列 $\{a_i\}_{i=1}^{\infty}$ 满足：

$$\begin{cases} a_1 = K \\ a_n = Aa_{n-1} + B\sum_{i=1}^{n-1} a_i a_{n-i} \ (n > 1) \end{cases}$$

你需要求出 $\{a_1, a_2 \cdots a_N\}$ 对 10^9+7 取模的值。由于输出规模较大，你只需要输出取模后的值的异或和。

$1 \le N \le 10^7$，$1 \le B, K < 10^9+7$，$0 \le A < 10^9+7$。

设 $a_0 = 0$，$\{a_i\}_{i=0}^{\infty}$ 的生成函数为 g，那么由递推式不难得出 $g = Kx + Axg + Bg^2$，所以 g 在 $\mathbb{K}(x)$ 上是代数的，那么 g 就是微分有限的，从而 a 是一个整式递推数列。我们使用节2.3.1中的方法即可求出递推式并进行递推，时间复杂度 $O(N)$。本题也有本质上相同的初等推导方法，但是较为繁琐。

例题 4. *Jump Jump Jump*[11]

平面上有一只兔子，它从 $(0,0)$ 出发开始进行跳跃。有 k 个互不相同的向量 $(dx_1, dy_1), (dx_2, dy_2) \cdots (dx_k, dy_k)$，每次兔子会从中均匀独立地随机选择一个，假设它现在在 (x, y) 并选择了向量 (dx_c, dy_c)，它会跳到 $(x + dx_c, y + dy_c)$。

对于每个正整数 x，在平面上 (x, x) 的位置均有一个陷阱。一旦兔子掉进了陷阱，它就出不来了。

兔子会不断地跳跃，直至掉入陷阱。给定正整数 n，对每个 $[1, n]$ 的正整数 x 输出兔子掉进陷阱 (x, x) 的概率。对 998244353 取模。

k 个向量互不相同，$\forall 1 \le i \le n$，dx_i, dy_i 均为 $[0, 3]$ 的整数且 dx_i, dy_i 不同时为 0。$1 \le n \le 10^5$，$1 \le k \le 15$，保证答案在模 998244353 意义下存在。

我们有如下定理：

定理 2.4. *若 $F(s,t)$ 是一个关于 s 和 t 的有理形式幂级数（即一个可以表示为两个关于 s 和 t 的多项式的商的形式幂级数），那么 $diag\ F = \sum_n x^n [s^n t^n] F(s,t)$ 为关于 x 的代数形式幂级数。*

该定理的证明较为复杂，可参见 [10] 定理 6.3.3。

考虑先求出 p_t 表示假设没有陷阱，兔子可以无限地走下去，兔子到过 (t, t) 的概率。设 $G(s,t) = \frac{1}{k}\sum_{i=1}^{k} s^{dx_i} t^{dy_i}$，设 $F(s,t)$ 表示兔子到过的位置的生成函数，即 $[s^x t^y] F(s,t)$ 表

[10] 来源：Codechef JADUGAR2, 有改动

[11] 来源：Grand Prix of Zhejiang, Problem J, 有改动

示兔子到过 (s,t) 的概率，我们就有 $F(s,t)=\sum_{i=0}^{\infty}G(s,t)^i=\frac{1}{1-G(s,t)}$。那么由上述定理，$diag\ F=\sum_n x^n[s^nt^n]F(s,t)$ 为关于 x 的代数形式幂级数，而 $p_t=[x^t]diag\ F$，所以由定理2.2，p 为整式递推数列。我们使用动态规划求出 p 的前若干项并用节2.3.1中的方法解出这个整式递推数列的整式递推式，即可递推出 $p_1,p_2\cdots p_n$。

设 h_t 表示兔子掉进陷阱 (t,t) 的概率，那么 p_t-h_t 的值即为如果无限走可以到达 (t,t)，但实际上在此之前被别的陷阱困住的概率。假设它实际上被陷阱 (s,s) 困住，概率即为 h_sp_{t-s}，所以我们有 $h_t=p_t-\sum_{s=1}^{t-1}h_sp_{t-s}$。设 $P=\sum_{i=1}^{\infty}x^ip_i$，$H=\sum_{i=1}^{\infty}x^ih_i$，我们就有 $H=P-HP$，所以 $H=\frac{P}{1+P}$。我们已知 $P \bmod x^{n+1}$，所以我们使用多项式求逆求出 $\frac{1}{1+P} \bmod x^{n+1}$，并把它乘上 $P \bmod x^{n+1}$ 就得到了欲求的 $H \bmod x^{n+1}$。多项式求逆和多项式乘法可以在 $O(n\log(n))$ 时间内计算 [4]。

例题 5. 简单计数[12]

求 n 个点的不同带标号有向无环图数量，其中每条边有 k 种颜色之一，图中的每个点有不超过 1 条出边，且每个点的入边条数在集合 S 中。对 998244353 取模输出。

两个图不同当且仅当存在一条从某个点 a 到某个点 b 的有向边，它只在恰好一个图中出现，或在两个图中都出现但颜色不同。

$1\le n\le 9\times 10^8$，$1\le k\le 10^7$，$0\in S$，S 集合中所有元素为 $[0,3]$ 的整数。

考虑所有 n 个点的带标号有根森林，其中每个点的儿子数量在 S 集合中，我们把每个点的儿子向这个点连边，容易发现这与题目中所描述的有向图一一对应，那么我们要计数的就是每个点的孩子个数在 S 中、每条边有 k 种颜色的 n 个点的带标号有根森林个数。

记 f_n 为 n 个点的合法有根树个数，$g_{a,n}$ 为 n 个点的恰好由 a 棵树组成的合法有根森林个数，h_n 表示 n 个点的合法有根森林个数。对于一个数列 $\{a_0,a_1\cdots a_t\}$，我们定义它的指数生成函数为形式幂级数 $A(x)=\sum_{i=0}^{t}a_i\frac{x^i}{i!}$。设 $\{f_n\}_{n=0}^{\infty}$、$\{g_{a,n}\}_{n=0}^{\infty}$、$\{h_n\}_{n=0}^{\infty}$ 的指数生成函数分别为 $F(x)$、$G_a(x)$ 和 $H(x)$。

由指数生成函数的有关知识（可参见 [12]），不难得出 $G_a(x)=\frac{F(x)^a}{a!}$，$F(x)=\sum_{t\in S}xk^tG_t(x)=\sum_{t\in S}\frac{xk^tF(x)^t}{t!}$，$H(x)=\sum_{t=0}^{\infty}\frac{F(x)^t}{t!}=e^{F(x)}$，而答案就是 $n![x^n]H(x)$。由于 $F(x)=\sum_{t\in S}\frac{xk^tF(x)^t}{t!}$，$F(x)$ 为代数形式幂级数，而 e^x 微分有限，所以由定理2.2，$H(x)=e^{F(x)}$ 为微分有限的，所以 $\{[x^t]H(x)\}_{t=0}^{\infty}$ 为整式递推数列，由定理2.3，$\{t![x^t]H(x)\}_{t=0}^{\infty}$ 也为整式递推数列。我们用节2.3.1中的方法消元出整式递推式，再使用节2.3.2中的方法即可求出 $n![x^n]H(x)$。

3 总结

线性递推数列和整式递推数列是两类非常常见的数列，在数学中有广泛的应用，但在信息学竞赛中目前较不普及。本文介绍了这两类常见递推数列的定义、性质、有关算法和

[12]来源：2019 年集训队互测，有改动

应用，希望能帮助大家对此类问题有所了解，也希望更多有趣的此类问题能出现在信息学竞赛中。

这一领域中仍有许多问题亟待解决，由于作者水平有限，希望本文能够起到抛砖引玉的作用，吸引更多的读者来研究此类问题。

感谢

感谢中国计算机学会提供学习和交流的平台。
感谢国家集训队张瑞喆教练的指导。
感谢黄志刚老师、宋立林老师、黄晓燕老师、魏丽真老师对我的培养与教导。
感谢杜瑜皓前辈、毛啸前辈、王修涵同学、朱震霆同学对本文的帮助。
感谢孔朝哲同学、刘承奥同学、曾耀辉前辈、张晴川前辈、吴航前辈为本文验稿。
感谢父母对我的理解与支持。

参考文献

[1] 毛啸. 关于数列递归式的一些研究. IOI2017 中国国家候选队论文集, 2017

[2] Wikipedia contributors. Cayley-Hamilton theorem. Wikipedia, https://en.wikipedia.org/wiki/Cayley-Hamilton_theorem

[3] Massey J. Shift-register synthesis and BCH decoding. IEEE transactions on Information Theory, 1969

[4] 彭雨翔. Introduction to Polynomials. WC2016, 2016

[5] Wikipedia contributors. Schwartz-Zippel lemma. Wikipedia, https://en.wikipedia.org/wiki/Schwartz-Zippel_lemma

[6] Wiedemann, Douglas. Solving sparse linear equations over finite fields. IEEE transactions on information theory 32.1, 1986

[7] Chen, Li, et al. Efficient matrix preconditioners for black box linear algebra. Linear Algebra and its Applications 343, 2002

[8] Wikipedia contributors. Planar graph. Wikipedia, https://en.wikipedia.org/w/index.php?title=Planar_graph

[9] 王修涵. 浅谈图模型上的随机游走问题. IOI2019 中国国家候选队论文集, 2019

[10] Stanley RP. Enumerative combinatorics. Volume 62 of Cambridge Studies in Advanced Mathematics, 1999

[11] Min-25. Find Linear Recurrence with Polynomial Coefficients. https://min-25.hatenablog.com/entry/2018/05/10/21280

[12] 金策. 生成函数的运算与应用. WC2015 营员交流, 2015

浅谈图模型上的随机游走问题

成都市第七中学 王修涵

摘 要

随机游走问题是信息学竞赛中一类常见的问题，在实际生活中也有广泛的应用，例如谷歌的网页排名算法。本文围绕这个问题，从网格图，稀疏图，一般图三个角度分别进行了一定研究，介绍了几种解决这类问题常用的方法，并对比了它们在解决各种问题的优缺点，希望读者遇到这类问题时有迹可循。

1 引言

在近年的算法竞赛中，随机游走问题的考察逐渐增多。这类问题变化繁多，解决方法因题而异，常常让人难以下手。

本文主要从网格图、稀疏图、一般图三个角度进行探究，介绍了解决这类问题的各种方法，并对比了它们在解决各种问题时的优缺点，希望读者遇到这类问题时有迹可循。

本文第二节将介绍随机游走问题的定义。第三节将介绍在网格图上的随机游走问题的两种时间复杂度分别为 $O(n\sqrt{n})$ 和 $O(n^2)$ 的方法。第四节将介绍在稀疏图上对单一起点终点求解随机游走问题的一种时间复杂度为 $O(n^2)$ 的方法。第五节将介绍在一般图上对所有点对作为起点终点求解随机游走问题的时间复杂度为 $O(n^3)$ 的方法。第六节对这些方法的优缺点进行比较。第七节总结全文。

2 定义

给定一张有向简单图 $G=(V,E)$（$V=\{v_1,v_2,\cdots,v_{|V|}\}$）和起点 $v_s\in V$，终点 $v_t\in V$，每条边 $e=(v_x,v_y)$ 有正权值[13] w_e，满足 $\forall v_x\in V\setminus\{v_t\},\sum_{(v_x,v_y)\in E}w_{(v_x,v_y)}=1$，且对于任意点 v_x 都存在一条从 v_x 出发到达 v_t 的路径。有一枚棋子从起点出发，每秒从当前所在点 v_x 以 w_{v_x,v_y} 的概率选择出边 (v_x,v_y) 并走向 v_y，到达终点则停止，求期望花费时间[14]。

事实上，这个问题也可以写成矩阵的形式。定义矩阵 P：

[13]若权值为 0 则可以认为这条边不存在。

[14]为了方便描述，本文中的定义与实际随机游走问题的定义略有不同。

$$P_{x,y}=\begin{cases}w_{(v_x,v_y)} & (v_x,v_y)\in E\text{且}x\neq t\\ 0 & (v_x,v_y)\notin E\text{或}x=t\end{cases}$$

要求的答案即为：

$$\sum_{k\geq 0}k\times(P^k)_{s,t}$$

其中 $(P^k)_{s,t}$ 表示走了 k 步第一次到达终点的概率。当图有限且所有点都能到达终点时，由 P 的定义可以证明其特征值都小于 1 ，所以答案一定是收敛的。

为了方便描述，在本文中如无特殊说明，均用 n 代指 $|V|$ ，m 代指 $|E|$ 。

另外，在本文中，稀疏图指边数和点数同阶的图。

3 网格图

例题 1. *Circles of Waiting*[15]

有一枚棋子起始被放在平面直角坐标系的 $(0,0)$ 点。每秒棋子会随机移动。假设它当前在 (x,y) ，它下一秒有 p_1 的概率移动到 $(x-1,y)$ ，p_2 的概率移动到 $(x,y-1)$ ，p_3 的概率移动到 $(x+1,y)$ ，p_4 的概率移动到 $(x,y+1)$ 。保证 $p_1+p_2+p_3+p_4=1$ 。求期望经过多少时间它会移动到一个离原点的欧几里得距离大于 R 的位置。

$0\leq R\leq 50, p_1,p_2,p_3,p_4>0$ ，答案对 10^9+7 取模。

3.1 朴素做法

记 $f(i,j)$ 表示在棋子在 (i,j) 时移动到一个离原点的欧几里得距离大于 R 的位置的期望时间，转移方程为：

$$f(i,j)=\begin{cases}p_1f(i-1,j)+p_2f(i,j-1)+p_3f(i+1,j)+p_4f(i,j+1)+1 & i^2+j^2\leq R\\ 0 & i^2+j^2>R\end{cases}$$

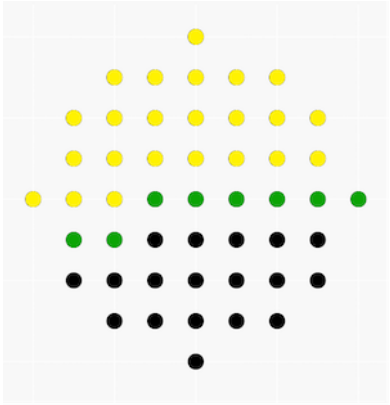
由于转移并不存在拓扑序，需要使用高斯消元求解。时间复杂度 $O(R^6)$ ，无法通过本题。

3.2 直接消元法

注意到需要消元的方程的系数大多数都是 0 ，消元时只对值非 0 的位置进行计算就可以降低复杂度 [1]。

[15] Tinkoff Internship Warmup Round 2018 and Codeforces Round #475(Div. 1), Problem E

考虑消元的过程，将方程按照在坐标系中从上到下，同一层中从左到右的顺序进行消元，将已经消元过的方程染成黄色，与黄色点相邻的点染成绿色，其余点染成黑色，如下图：



接下来我们要对下一个绿色格子对应的方程进行消元。在这个方程中，只有绿色格子和它下方的第一个黑色格子对应的变量系数可能不为 0；而只有绿色格子和它下方的第一个黑色格子对应的方程中，当前格子对应的变量系数可能不为 0。

注意到绿色格子只有 $O(R)$ 个，所以单个方程消元的时间复杂度为 $O(R^2)$。一共只有 $O(R^2)$ 个方程，所以时间复杂度降低为 $O(R^4)$，可以通过本题。

3.3 主元法

方程和变量都有 $O(R^2)$ 个，如果能将规模缩小至 $O(R)$，那么朴素的高斯消元就能通过了。

将每行从左到右第一个格子对应的变量设为主元，共 $2R+1$ 个，设法将其他格子对应的变量用关于这些主元的线性函数表示。从左到右逐列考虑，对于当前列的每个格子 (i,j)，注意到 $f(i,j), f(i-1,j), f(i,j-1), f(i,j+1)$ 都是已知的关于主元的线性函数，将转移方程移项，有：

$$f(i+1,j)=\frac{f(i,j)-p_1f(i-1,j)-p_2f(i,j-1)-p_4f(i,j+1)-1}{p_3}$$

这样我们就能得到 $f(i+1,j)$ 关于主元的线性函数表示。如果 $(i+1,j)$ 已经离原点欧几里得距离超过 R 了，则可以得到一个方程：$f(i+1,j)=0$。最终，我们会得到 $2R+1$ 个方程，对这些方程进行高斯消元即可。

在递推关于主元的线性函数的阶段，共有 $O(R^2)$ 个变量，递推单个变量需要花费 $O(R)$ 的时间；之后，我们将问题的规模缩减到了 $O(R)$。两部分的时间复杂度均为 $O(R^3)$，总的时间复杂度也为 $O(R^3)$，可以通过本题。

3.4 两种做法的对比

我们从多种方面对比两种做法：

从时间复杂度方面，主元法在网格图上的最坏时间复杂度为 $O(n\sqrt{n})$（当网格图长和宽都为 $O(\sqrt{n})$ 级别时时间复杂度最高），直接消元法在网格图上最坏时间复杂度为 $O(n^2)$，主元法较优。

从精度方面，对于一些需要进行实数计算而不是取模的题目，直接消元法的精度优于主元法。

从适用性方面，两种做法适用于不同的方面。

当网格图中存在障碍或者走某些边的概率为 0 时，对于每个障碍或者概率为 0 的边主元法需要增加一个主元，当障碍或者概率为 0 的边的数量多于 $O(R)$ 时，主元法的时间复杂度会增加，而直接消元法的时间复杂度仍然不变。

但主元法还可以做类似于网格图的转移方程的消元，例如 $f(i,j) = p_1 f(i+1,j) + p_2 f(i,j+1) + p_3 f(pre(i,j)) + 1$，其中 $pre(i,j) = (x,y)(x \le i, y \le j)$ 是问题给定的值，而直接消元法的复杂度分析在这种模型中并不适用。

除此以外，网格图邻接矩阵行列式的计算，也不能使用主元法，只能用直接消元法来优化时间复杂度。

综上，两种做法各有所长，需要根据具体题目分析采用不同的做法。

4 稀疏图

例题 2. *Expected Value*[16]

给定一张简单无向连通平面图 $G = (V, E)$，有一枚棋子起始被放在 v_1，每秒棋子会从与当前点相连的边中等概率选择一条走到出边指向的点，求到达 v_n 的期望时间。

$n \le 2000$，答案对 p 取模，p 是在区间 $[10^9, 1.01 \times 10^9]$ 内随机生成的一个质数。

一个结论是，平面图的边数和点数是同阶的。具体地，我们有：

定理 4.1. 一个 $n(n \ge 3)$ 个点的平面图，其边数 m 不超过 $3n - 6$ *[2]*。

由于本题的做法能拓展到任意稀疏图上，所以接下来的讨论只基于本题边数和点数同阶这个条件，而不需要平面图的其他性质。

4.1 基础知识

定义 4.1. 所有满足 $p(A) = 0$ 的多项式 $p(\lambda)$ 称为矩阵 A 的零化多项式。

[16]Ildar Gainullin Contest 1, Problem E, 有改动

定义 4.2. 记 I_n 表示 n 阶单位矩阵，定义一个 $n \times n$ 矩阵 A 的特征多项式为 $p(\lambda) = \det(\lambda I_n - A)$，其中 det 表示一个矩阵的行列式。

不难发现，一个 n 阶矩阵 A 的特征多项式的次数不超过 n。

定理 4.2. （凯莱-哈密顿定理 *[3]*）任意矩阵的特征多项式是它的零化多项式。

所以，一个 n 阶矩阵的次数最小的零化多项式的次数也不超过 n。

4.2 求解原问题

注意到，期望走的时间 $E(t) = \sum_{i \geq 0} \Pr[t > i]$，如果我们能求出走了 i 步还没有结束的概率，对所有 $i \geq 0$ 求和即为答案。

记 $f(i, j)$ 表示走了 i 步，当前停留在 v_j，且没有走到过 v_n 的概率，那么有：

$$f(i, j) = \sum_{(v_k, v_j) \in E} \frac{f(i-1, k)}{deg_k} (j \neq n)$$

其中 deg_k 表示 v_k 的度数。

注意到 f 的转移与 i 无关，可以认为一次转移是乘上了一个矩阵，即 $f_{i+1} = f_i M$。由于 M 的最小零化多项式次数不超过 n，所以 f 的最短递推式长度也不超过 n，故 $\Pr[t > i] = \sum_{j=1}^{n-1} f(i, j)$ 的最短递推式长度也不超过 n。我们可以在 $O(nm)$ 的时间求出 $\Pr[t > 0], \Pr[t > 1], \cdots, \Pr[t > 3n]$，然后使用 Berlekamp-Massey 算法 [4]，在 $O(n^2)$ 的时间内求解出 $\Pr[t > i]$ 的最短递推式。

考虑求一个 k 阶线性递推序列 a 的生成函数。不妨设 $i \geq i_0$ 时 $a_i = \sum_{j=1}^{k} c_j a_{i-j}$，记 a 和 c 的生成函数为 $A(x)$ 和 $C(x)$，那么 $A(x) = A(x)C(x) + A_0(x)$，其中 $A_0(x)$ 是由 $i < i_0$ 的项决定的。

回到原问题，由于我们能求出 $\Pr[t > i]$ 的最短递推式，则我们可以求出 $C(x)$ 和 $A_0(x)$（定义与上一段相同），移项得 $A(x) = \frac{A_0(x)}{1 - C(x)}$。我们要求的是 $\sum_{i \geq 0} [x^i] A(x)$，不难发现这个值就等于 $A(1)$，将 $x = 1$ 带入原问题求解即可。由于模数是随机质数，可以认为分母不会为 0。

这样，我们就在 $O(nm + n^2)$ 的时间复杂度内解决了本题。由于点数与边数同阶，所以本题中时间复杂度可以认为是 $O(n^2)$。

另外本题也有一个不同的做法，可以参见参考文献 [5]。

5 一般图

例题 3. *Frank*[17]

[17] 2018 ACM-ICPC Asia Nanjing Regional Contest, Problem F, 有改动

给定一张简单强连通有向图 $G=(V,E)$ ，对于所有 $1\le s\le n, 1\le t\le n, s\ne t$ 回答下面的问题：

有一枚棋子起始被放在 v_s ，每秒棋子会从当前点的出边中等概率选择一条走到出边指向的点，求到达 v_t 的期望时间。

$3\le n\le 400$ 。

5.1 分析和转化

记 $p_{i,j}$ 表示棋子在 v_i 时，选择出边 (v_i,v_j) 走到 v_j 的概率，特别地，当出边不存在时概率为 0 。记 $f_{i,j}$ 表示 v_i 随机游走到 v_j 的期望时间，特别地，$f_{i,i}=0$ 。当 $i\ne j$ 时，转移方程为：

$$f_{i,j}=1+\sum_{1\le k\le n} p_{i,k}f_{k,j}$$

当 $i=j$ 时，记 g_i 表示从 v_i 开始随机游走，第一次回到 v_i 的期望时间，那么：

$$f_{i,i}=1-g_i+\sum_{1\le k\le n} p_{i,k}f_{k,i}$$

为了方便观察，我们将转移方程写成矩阵的形式。记 P 表示这个图的转移矩阵，F 表示答案矩阵，I 表示 n 阶单位矩阵，J 表示 n 阶全 1 矩阵，G 是一个 n 阶矩阵，满足 $G_{i,i}=g_i$ ，其他位置为 0 ，则：

$$F=J-G+PF$$

如果我们能求出 G，那么我们只需要解方程 [6]：

$$(I-P)F=J-G$$

5.2 G 的求法

定义 5.1. *定义一个 n 阶转移矩阵 P 的稳态分布 [7] 为一个 n 维向量 π ，满足 $\sum_{i=1}^{n}\pi_i=1, \pi P=\pi$ 。其中，π 每一维的值都在区间 $[0,1]$ 内。*

我们很容易找出稳态分布的实际意义。如果某个时刻棋子有 π_i 的概率停留在 v_i ，则在之后的任意时刻，棋子仍然满足这个概率分布。我们可以在 $O(n^3)$ 的时间通过高斯消元解方程来求出 π ，那么 π 与 G 有什么关系呢?

定理 5.1. *对于任意 $1\le i\le n$ ，有 $\pi_i g_i=1$ 。*

证明. 由 $F = J - G + PF$ ，移项得：

$$G = PF + J - F$$

两边同时在左边乘上 π 有：

$$\pi G = \pi PF + \pi J - \pi F$$

由 π 的定义有 $\pi P = \pi$ ，故：

$$\pi G = \pi J$$

所以：

$$\pi_i g_i = \sum_{j=1}^{n} \pi_j = 1$$

原命题得证。 □

所以，通过引入稳态分布，我们可以在 $O(n^3)$ 的时间内求解 G 。

5.3 求解原问题

在解方程的过程中，我们发现一个问题：$(I - P)$ 并不满秩，不能通过乘逆矩阵的方法求解。

定义 5.2. *定义一个有向图 $G = (V, E)$ 的以 $v_r \in V$ 为根的有向生成树是 G 的一个子图 $T = (V, A)$ ，满足：*

1) 对于任意 $v_i \neq v_r$ ，v_i 的出度为 1 。

2) v_r 的出度为 0 。

3) T 中不存在环。

引理 5.1. *（有向图上的矩阵树定理 [8]）对于一个有向图 G ，记 D 表示其出度矩阵，即 $D_{i,i} = d_i, D_{i,j} = 0 (i \neq j)$ ，其中 d_i 表示 v_i 的出度，记 A 表示其邻接矩阵，则其以 v_r 为根的有向生成树个数为 $D - A$ 去掉第 r 行第 r 列后的行列式。*

定理 5.2. *对于一个强连通图 $G = (V, E)$ 的转移矩阵 P ，$(I - P)$ 的秩为 $n - 1$ 。*

证明. 因为对矩阵某一行乘上一个非零常数其秩不改变，所以我们将 $(I-P)$ 的第 i 行乘上 v_i 的出度，得到一个新的矩阵 L ，只需证明 L 的秩为 $n-1$ 即可。

由于 L 每行的和均为 0 ，对 L 的所有列向量求和，会得到零向量，即这些向量线性相关，所以 L 的秩不为 n 。

不难发现 L 等于图 G 的出度矩阵减去其邻接矩阵，由引理 5.1 得 L 去掉第 i 行第 i 列后行列式表示以 v_i 为根的有向生成树个数。

由于 G 是强连通的，所以以任意点为根的有向生成树个数均不为 0 ，即 L 去掉第 i 行第 i 列之后仍然满秩。

因为加上一列秩不会变小，所以 L 去掉第 i 行后所有行向量线性无关。故 L 的秩为 $n-1$ 。 □

回到原问题，考虑求解原问题中的方程。为了方便，我们将方程写成 $AX=B$ 的形式，其中 A,B 已知，需要求解 X 。由于 A 不满秩，解有无数个，我们首先求出一组特解。

将 A 和 B 一起做高斯消元。把 A 的前 $n-1$ 行消成只有主对角线和第 n 列有值的形式，最后一行消成全 0 ，即下列形式：

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & a_1 \\ 0 & 1 & 0 & \cdots & 0 & a_2 \\ 0 & 0 & 1 & \cdots & 0 & a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & a_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} X = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & \cdots & b_{1,n-1} & b_{1,n} \\ b_{2,1} & b_{2,2} & b_{2,3} & \cdots & b_{2,n-1} & b_{2,n} \\ b_{3,1} & b_{3,2} & b_{3,3} & \cdots & b_{3,n-1} & b_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ b_{n-1,1} & b_{n-1,2} & b_{n-1,3} & \cdots & b_{n-1,n-1} & b_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

令 $X_{n,i}=0$ ，可以解出一组特解，记为 Y 。接下来我们需要将特解调整为真正的解。

注意到 $X_{i,i}=0$ ，考虑组合意义有 $Y_{i,j}=1+Y_{j,j}+P_{i,k}X_{k,j}$ ，不难解出 $X_{i,j}=Y_{i,j}-Y_{j,j}$ 。

最终，我们在 $O(n^3)$ 的时间复杂度内解决了这个问题。

6 比较和讨论

从时间复杂度方面，网格图上存在 $O(n\sqrt{n})$ 的主元法，在稀疏图上的算法则是 $O(n^2)$ 的，在一般图上，复杂度更高，为 $O(n^3)$ 。

从求解问题的角度方面，网格图上的算法能对图中所有点作为起点的情况求解，而稀疏图上的算法只能对单个起点、单个终点的情况快速求解，一般图上的算法则是对所有点对作为起点和终点的情况求解。如果网格图、稀疏图上的算法要对多个终点求解，或者稀疏图上的算法要对多个起点求解，则需要花费额外的时间。

从适用范围方面，因为网格图是稀疏图的子集，而稀疏图是一般图的子集，所以网格图上的算法适用范围最小，而一般图上的算法适用范围最大。

从精度方面，由于 Berlekamp-Massey 算法的精度非常差，所以在实数计算的题目中，一般不能使用稀疏图上的算法，而网格图和一般图上的算法都是可以选用的。

综上所述，几种算法各有优劣，使用哪种算法往往因题而异，选手求解这类问题时，应该分析题目性质后选择最适合该题目的方法。

7 总结

本文研究了图模型上的随机游走问题，分别在网格图、稀疏图、一般图上介绍了几种不同的算法。注意到这些算法大多是基于矩阵的基础上讨论的，这说明矩阵是解决随机游走问题的一种有力工具。对于随机游走问题，转化成矩阵形式后，往往就能更好下手解决问题了。

同时，本文中提到的算法不仅能用在随机游走上，还能在别的问题上给我们一些启发：网格图中的直接消元法启发我们对于一些有特殊性质的稀疏矩阵，可以通过一些剪枝来降低复杂度；主元法可以用于一类解方程问题；稀疏图中提到的算法则启示我们利用线性递推求解一些矩阵相关的问题；一般图中的稳态分布是解决马尔科夫链相关问题的有力工具，它还有很多性质值得我们发掘。

事实上随机游走问题还有很多值得研究的地方，希望本文起到抛砖引玉的作用，吸引更多的读者来研究这一类问题。

8 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢成都七中蔺洋老师和张君亮老师的关心和指导。

感谢国家集训队教练张瑞喆的指导。

感谢杨景钦学长对我的帮助。

感谢陈松扬前辈，唐靖哲前辈，杜瑜皓前辈对我的启发和写作本文的帮助。

感谢陈松扬前辈，唐靖哲前辈，房励行同学为本文验稿。

参考文献

[1] Romanov, V. ”Editorial Tinkoff Internship Warmup Round 2018 and Codeforces Round #475 (Div. 1 + Div. 2)”, *Codeforces*, https://codeforces.com/blog/entry/58991

[2] Wikipedia contributors. ”Planar graph”. *Wikipedia*, https://en.wikipedia.org/wiki/Planar_graph

[3] Hamilton, W. R. (1853). *Lectures on Quaternions*. Dublin.

[4] Massey, J. L. (1969), "Shift-register synthesis and BCH decoding", *IEEE Trans. Inf. Theory*, IT-15 (1): 122–127, doi:10.1109/TIT.1969.1054260

[5] 钟子谦. 两类递推数列的性质和应用, IOI2019 中国国家候选队论文集

[6] Tang, J. "2018-2019 ACM-ICPC, Asia Nanjing Regional Contest (Online Mirror on Gym)", *Codeforces*, https://codeforces.com/blog/entry/63057

[7] Wikipedia contributors. "Stationary Distribution". *Wikipedia*, https://en.wikipedia.org/wiki/Stationary_Distribution

[8] Chaiken, S.; Kleitman, D. (1978), "Matrix Tree Theorems", *Journal of Combinatorial Theory, Series A*, **24** (3): 377–381, doi:10.1016/0097-3165(78)90067-5, ISSN 0097-3165

《小水题》命题报告

南京外国语学校 杨骏昭

摘 要

本文介绍了作者命制的候选队互测第五场的第三题。这是一道以维护水槽的水位变化为背景的一道数据结构题。此题需要先将水位变化这个物理模型转化为易于维护的数学模型，然后再使用数据结构技巧优化并解决问题。

1 题目大意

1.1 题目描述

有 n 个底面积相同的水槽排成一排，相邻两个水槽共用一个隔板，第 i 个水槽与第 $i+1$ 个水槽之间的隔板高度为 b_i 。最左边与最右边的隔板高度可以视为无限大。初始时所有水槽是静止的，第 i 个水槽初始水位是 a_i 。你需要维护以下两种操作：

- 给定 x 和 h ，在第 x 个水槽和第 $x+1$ 个水槽之间的隔板的高度 h 处钻一个小孔。这之后一些水槽的水位会发生变化，你应该一直等到这些水槽恢复静止。
- 给定 x ，询问此时第 x 个水槽的水位。

1.2 输入格式

从标准输入读入数据。
第一行两个整数 n,q ，分别表示水槽数和操作数。
第二行 n 个由空格隔开的实数 a_i ，表示初始水位。
第三行 $n-1$ 个由空格隔开的实数 b_i ，表示初始隔板高度。
接下来 q 行每行描述一个操作。
1 x h 表示第一种操作，保证 $1 \le x < n$ 。注意 h 是实数。
2 x 表示第二种操作，保证 $1 \le x \le n$ 。
输入中出现的所有实数最多有七位小数。

1.3 输出格式

输出到标准输出中。

对于第二种操作，每一行输出答案。

最后一行输出 n 个空格隔开的实数，表示操作结束后的每个水槽的水位。

绝对误差在 10^{-7} 内即可接受。

1.4 提示与附加说明

在这题中，我们使用一个理想模型。即我们假设水的体积不变，并且忽视水的表面张力、摩擦力等。你可以认为钻孔之后的水位变化是符合生活常识的。你可以认为，对于两个相邻的水位 a_i, a_{i+1} ，以及它们之间隔板存在一个高度为 h 的小孔（或初始隔板高度为 h ），如果有 $a_i > h, a_i > a_{i+1}$ ，那么将会有水从 i 流向 $i+1$ 。对称地，如果有 $a_{i+1} > h, a_{i+1} > a_i$ ，那么将会有水从 $i+1$ 流向 i 。水从 x 流向 y 是指 a_x 不断减少，a_y 不断增加，并且保持它们的和不变。

1.5 限制与约定

对于所有数据，$1 \le n, q \le 100000, 0 \le a_i, h \le 100$ ，对于 $1 \le i < n$ 有 $b_i \ge \max(a_i, a_{i+1})$ 。输入中出现的所有实数最多有七位小数。

- 子任务 1（1 分）：$1 \le n, q \le 10$ ；
- 子任务 2（10 分）：$1 \le n, q \le 300$ ；
- 子任务 3（10 分）：$1 \le n, q \le 2000$ ，对于 $i > 1$ ，满足 $a_i = 0$ ；
- 子任务 4（10 分）：$1 \le n, q \le 2000$ ；
- 子任务 5（30 分）：对于 $i > 1$ ，满足 $a_i = 0$ ；
- 子任务 6（39 分）：无特殊限制。

时间限制：4s

空间限制：1024MB

2 解法分析

2.1 算法一

这是一道数据结构题。首先考虑按照题意直接模拟。题目需要维护水从高处往低处流的过程。但是不同于一般的数据结构题，事情并没有这么简单。因为题目的描述并不是程式化的语言，所以我们需要先建立起模型。

由于隔板只与它的初始高度和钻孔高度的最小值有关，所以下文中将隔板高度（也就是 b_i ）直接视为这个最小值。

题目中的提示已经给出了一种可能的模型。每次只考虑相邻的两个水槽，如果它们处于不平衡状态，那么就将它们调整为平衡状态，直到不存在两个相邻的不平衡的水槽为止。

这个模型并不易于维护。考虑如果有三个水槽，水从第一个水槽流向后两个水槽，最终三个水槽水位都应该相同，但是由于每次只调整相邻两个水槽的水位，无法在有限步内使它们水位相同，这个过程可能会无限地进行下去。但是由于我们并不需要准确答案，只需要保证九位精度（包含个位和十位，以及小数点后的七位）即可，所以我们可以在两个高度十分接近时不继续做下去。于是我们就有了算法一：

算法一：设置 $eps = 10^{-8}$ 。对于任意 $1 \le i < n$ ，我们称它为不平衡状态当且仅当 $a_i \ge \max(b_i, a_{i+1}) + eps$ 或 $a_{i+1} \ge \max(b_i, a_i) + eps$ 。对于每次修改，我们不断找出不平衡状态并进行调整，直到没有不平衡状态为止。如果是 $a_i \ge \max(b_i, a_{i+1}) + eps$ ，那么变化水量就为 $\min(a_i - b_i, \frac{a_i - a_{i+1}}{2})$ ，否则变化水量就是 $min(a_{i+1} - b_i, \frac{a_{i+1} - a_i}{2})$ 。

算法复杂度： $O(q \times$ 与精度有关的大常数$)$

期望得分：可以通过子任务一。期望得分 1 分。

2.2 算法二

我们需要一个时间复杂度与精度无关的多项式时间的算法。我们可以观察出一些性质：

- 每次修改隔板高度后，考虑与隔板相邻的两个水槽的水位，水只会从高处往低处流。所以要么什么都不发生，要么水从左往右流，要么水从右往左流。
- 水从左往右流与水从右往左流是对称的。**下文分析性质时将不妨设水是从左往右流的，而讨论如何维护信息时再考虑两个方向的影响。**

算法一主要问题在于无法解决多个水槽的平衡问题。我们考虑改进算法一。我们称一段区间 $[l, r]$ 是**连通的**当且仅当同时满足下列三个条件：

- $1 \le l \le r \le n$
- 对于 $l \le i \le r$ ，满足 $a_i = a_l$ 。

- 对于 $l \le i < r$ ，满足 $b_i < \max(a_i, a_{i+1})$ 。

注意到当水位发生变化时，连通器内每个水槽的水位是一起变化，即增加和减少同样体积的水。我们基于连通区间设计出算法二：

算法二：我们将上一个做法中考虑两个相邻的水槽改为考虑两个相邻的连通区间，然后让水从高处往低处流，直到连通的条件被破坏或他们之间不再存在高度差。具体实现中我们需要支持：

- 合并两个连通区间。
- 将一个连通区间分裂为两个连通区间。
- 对连通区间实行整体加减。
- 询问连通区间内的隔板高度的最大值。

算法复杂度：容易发现每次调整后处于高处的连通区间的左端点或右端点至少会往右移一位，所以每次操作后只需要调整 $O(n)$ 次。所以每次修改操作复杂度为 $O(n\times$调整复杂度$)$ 。

根据实现的不同，每次调整的时间复杂度可以做到 $O(n)$ 或 $O(\log n)$ 或 $O(1)$ 。这里 $O(1)$ 可以用单调队列实现，并且每次修改操作需要额外的 $O(n)$ 复杂度。总时间复杂度可以做到 $O(qn^2)$ 或 $O(qn\log n)$ 或 $O(qn)$ 。

期望得分：根据实现的不同，可以通过前两个子任务或通过前四个子任务，期望得分 11 到 31 分。

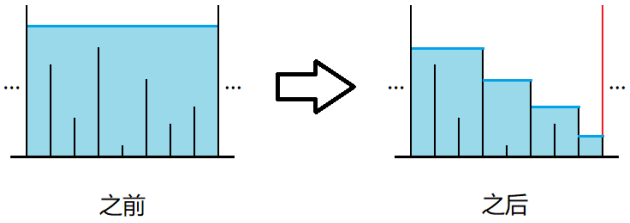
2.3 算法三

算法二连通区间的总合并或分裂次数可以达到平方级别，无法通过后两个子任务。注意虽然算法二中仍然存在冗余操作（考虑一个水槽呈阶梯状，左方的水可能会顺着阶梯流下去直到右下方，而中间的那些连通区间并没有发生改变），但是我们可以构造方案使得每次操作要么合并 $O(n)$ 个连通区间，要么将一个连通区间分裂为 $O(n)$ 个连通区间，也就是说存在一种方案使得相邻两次操作的连通区间个数的差的绝对值之和达到平方级别。

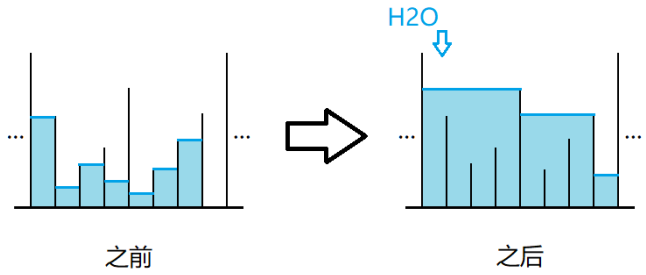
我们用另一个角度思考问题。可以发现如果在一次操作中按时间维护水槽的变化情况将会十分麻烦，我们考虑如何直接算出变化后的结果。

经过冷静分析后，有以下三种情形需要处理：

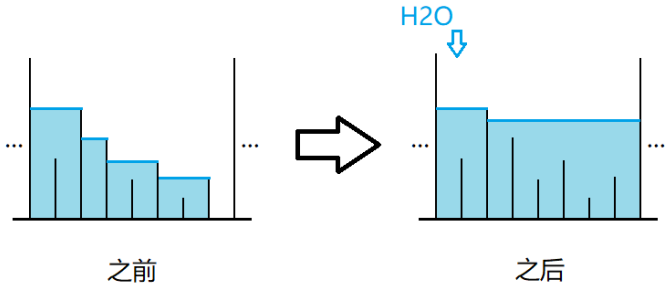
一、水流走后留下了一段“阶梯”。



二、水流经过时填满了一些“坑”。



三、水流下来被隔板挡住，形成了“湖”。



我们可以将整个变化过程按照三个图分为三个阶段。假设我们将第 x 个水槽与第 $x+1$ 个水槽之间的隔板高度修改为了 b'_x 。设 $h = \max(b'_x, a_{x+1})$ 。此时水从左往右流需要满足 $a_x > h$ 。

- 第一阶段：假设第 $x+1$ 个水槽的水位是负无穷，我们要计算出有多少水会流走。注意只有连通区间内的水位会产生变化。我们设连通区间的左端点为 l 。设流走的水的体积为 V 。(参考图 1)
- 第二阶段：假设 V 体积的水从第 l 个水槽往右流，找到一个最大的 r ，使得填满 l 到 r 的坑所需的体积小于等于 V 。(参考图 2)
- 第三阶段：设填坑剩下的水的体积为 V' 。我们需要将剩下来的水填入靠右的几个水

槽。相当于找到一个 h' ，使得 $[l,r]$ 区间中所有水位小于 h 的水槽填补到 h' 的水位所需要的体积恰好为 V'。（参考图 3）

注意这三个阶段不仅需要查询，同时也需要修改水位。虽然这几个阶段中的操作看起来有点难优化，但是仔细思考后发现他们有着相似之处。我们考虑下面这个操作：

对于一段区间 $[l,r]$ ，设 $FL(l,r,h)=\sum_{i=l}^{r}\max(\max_{j=i}^{r-1}b_j,h)$ 。对称地，设 $FR(l,r,h)=\sum_{i=l}^{r}\max(\max_{j=l}^{i-1}b_j,h)$。令 $ML(l,r,h)$ 表示这样一个修改操作：对于 $l\le x\le r$ ，令 $a_x=\max(\max_{i=x}^{r-1}b_i,h)$。对称地，$MR(l,r,h)$ 表示这样一个修改操作：对于 $l\le x\le r$ ，令 $a_x=\max(\max_{i=l}^{x-1}b_i,h)$。

如果考虑将这个隔板高度的后缀 max（或前缀 max）在图中表示出来的话，这个类似轮廓线一样的东西我们称之为**轮廓**，轮廓线的水位和，也就是轮廓线下的水的体积和称为这个区间的**轮廓水位**。这个函数的实际意义是（以 FL 为例），假设区间内初始没有水，区间两侧隔板高度为无限大，我们往最左侧的水槽中缓慢加水，直到最右侧的水位到达 h 后所需的水的体积。

此时三个阶段的过程就可以用轮廓水位来形式化地描述了。我们用 $SUM(l,r)$ 来表示 $[l,r]$ 第 l 个水槽到第 r 个水槽的水位和。

- 第一阶段：求出连通区间的左端点 l 。计算出 $V=SUM(l,x)-FL(l,x,h)$。执行 $ML(l,x,h)$ 操作。
- 第二阶段：找到一个最大的 r ，使得 $FL(l,r,a_r)-SUM(l,r)\le V$ ，令 $V'=V-FL(l,r,a_r)+SUM(l,r)$ 。执行 $ML(l,r,a_r)$ 操作。
- 第三阶段：找到一个最大的 h ，使得 $FL(l,r,h)-SUM(l,r)\le V'$ 。执行 $ML(l,r,h)$ 操作。

这里第三阶段使用了 FL 函数关于 h 的单调性，即对于 $h\le h'$ ，有 $FL(l,r,h)\le FL(l,r,h')$ 。

我们发现可以把这三个过程优美地合并成一个过程：

- 求出连通区间的左端点 l 。找出最大的 r ，再找出最大的 h ，使得 $FL(l,r,h)\le SUM(l,r)$ 。然后执行 $ML(l,r,h)$ 。

注意这里必须先保证 r 最大，再保证 h 最大，不然会出现不平衡状态。而这里 FL 函数还有一个单调性，也就是关于 r 的单调性，即对于 $r\le r'$ ，有 $FL(l,r,a_r)\le FL(l,r',a_{r'})$ 。

我们发现如果我们可以找到一个支持快速询问 FL、FR，支持快速进行 ML、MR 操作的数据结构，只需要两个二分就可以解决问题了！

算法三：使用上文提到的方法。$O(n)$ 暴力求解 FL、FR，暴力修改 ML、MR。

算法复杂度：由于每次修改操作需要二分，所以复杂度 $O(qn\log n)$ 。注意第三阶段的二分可以直接对值域二分，也可以用它是分段函数的性质按这个区间内的水位二分。如果是前者复杂度分析中的 $O(\log n)$ 可能要变为 $O(\text{二分值域次数})$ 。为了避免精度误差，这个二分次数应该足够大（60 左右）。（注意之后算法的复杂度分析中也存在这个问题）

期望得分：可以通过前四个子任务。期望得分 31 分。

2.4 算法四

特殊性质部分满足初始只有第一个水槽中有水。我们发现这部分子任务有一个优美的性质：a 数组是不增的，也就是对于任意 $1 \le i < n$ ，有 $a_i \ge a_{i+1}$ 。这个结论可以由我们最开始的模型得来，调整两个相邻的水槽时，它们的相对大小不会发生改变，所以若初始时满足这个性质，以后一定都满足这个性质。

我们还可以发现有水的区间一定只有前 K 个，并且对于任意 $1 \le i < K$ ，一定有 $a_i \ge b_i$ 。我们经过冷静分析发现之前的三个阶段可以化简为：

- 第一阶段：求出连通区间的左端点 l 。计算出 $V = SUM(l,x) - FL(l,x,h)$ 。执行 $ML(l,x,h)$ 操作。
- 第二阶段：找到一个最大的 h ，使得 $FL(l,r,h) - SUM(l,r) \le V'$ 。执行 $ML(l,r,h)$ 操作。若 h 大于 b_K ，我们将 K 增加一，然后对 b_K 进行修改，重复这个过程。

而由于这里的第二阶段满足 a 数组递减，且对于任意 $l \le i < r$ ，一定有 $a_i \ge b_i$ （其实这个性质在上文中第三阶段中也照样存在），所以这里的 FL 可以直接二分计算，即计算大于 h 的和与小于 h 的个数。

这里还有一种更简单的做法。由于水的总体积 V 固定，我们可以不维护真实的水位。我们只需要知道最右边的水槽是哪个以及最右边水槽的高度即可推算出所有水槽的水位。我们考虑只维护隔板高度，查询时再找出最右边有水的水槽 r 与最右边水槽的水位 h 。由于上文提到的单调性，我们只需要求出：

- 找出最大的 r ，再找出最大的 h ，使得 $FL(1,r,h) \le V$ 。

这样我们只需要支持 FL 操作即可。但是我们下文仍介绍一个接近标算的做法，即如何维护 FL、FR、ML、MR。

我们发现我们需要维护的是区间信息与区间操作，不难想到使用线段树。

首先线段树肯定需要维护区间内水位和。我们还需要支持这段区间对于任意 h 的 FL 和 FR 函数的快速询问，以及对于任意 h 的 ML 和 MR 的修改，我们无法使用传统的线段树解决问题。

我们考虑线段树节点 $[l,r]$ 维护 $[l,r]$ 的水槽信息，以及中间的 $r-l$ 个隔板信息。假设儿子节点已经可以维护水位 SUM，隔板 MAX，FL 与 FR 操作。设 $m=\lfloor\frac{l+r}{2}\rfloor$，那么 $[l,r]$ 的儿子节点为 $[l,m]$ 和 $[m+1,r]$ 。记 $HMAX(l,r)=\max_{i=l}^{r-1} b_i$ 。

我们发现 $FL(l,r,h)=FL(m+1,r,h)+FL(l,m,\max(h,\max(HMAX(m+1,r),b_m)))$ 。直接递归算下去是 $O(r-l)$ 的，但是如果能只选择一边递归算下去即可做到 $O(\log(r-l))$ 。

若 $h\geq HMAX(m+1,r)$ ，那么有 $FL(m+1,r,h)=(r-m)h$ ，此时只要递归算另一边就行了。若 $h<HMAX(m+1,r)$ ，注意到此时必须要快速算 $FL(l,m,\max(h,\max(HMAX(m+1,r),b_m)))$ ，而 $\max(h,\max(HMAX(m+1,r),b_m))=\max(HMAX(m+1,r),b_m)$ ，这是一个与 h 无关的常数。所以我们需要在线段树节点中记录 $FL(l,m,\max(HMAX(m+1,r),b_m))$ 。而维护这个值需要调用子节点的 FL 函数，不过只要子树内所有节点都维护了这个值，那么任意 FL 函数就都可以在 $O(\log n)$ 内算出。

考虑如何进行修改。注意到 ML 和 MR 操作本质是一个区间的修改操作，所以标记互相之间不兼容，可以直接覆盖，无需考虑合并标记。我们对线段树打 ML 或 MR 标记时同时记录参数 h 的值。打标记时考虑快速修改区间信息，注意到更新水位 SUM 只需要查询 FL 和 FR 函数即可。这种标记也很容易下传。

我们获得了一个能支持水位和、支持 FL、FR、ML、MR 操作的线段树。由于它的 update 函数（合并左右儿子信息）与 pushdown 函数（标记下传）以及单个区间的 FL、FR、ML、MR 操作都是 $O(\log n)$ ，所以对于任意区间操作的复杂度是 $O(\log^2 n)$ 。

算法四：使用上文提到的方法。$O(\log^2 n)$ 求解 FL，修改 ML。也可以直接用 FL 询问最右水槽及最右水槽水位。

算法复杂度：由于每次修改操作需要二分，所以复杂度 $O(q\log^3 n)$ 。如果第二阶段中将二分改为在线段树上二分，可以将复杂度降到 $O(q\log^2 n)$ 。

期望得分：可以通过前五个子任务（需要注意常数问题）。期望得分 31 到 61 分。

2.5 算法五、算法六

其实算法四中的数据结构已经可以用来解决整个问题了。但是我们还剩下一个问题：

- 求出连通区间的左端点 l 。找出最大的 r ，再找出最大的 h ，使得 $FL(l,r,h)\leq SUM(l,r)$ 。

算法五：使用上文提到的方法。$O(\log^2 n)$ 求解 FL、FR，修改 ML、MR，$O(\log^3 n)$ 进行二分 + 线段树上查询 FL、FR。

算法复杂度：总复杂度 $O(q\log^3 n)$ 。

期望得分：可以通过所有子任务。（需要注意常数问题）期望得分 31 到 100 分。

这里找出最大的 h 可以仿照上文的算法做到 $O(\log^2 n)$ 。关键在于如何找出最大的 r 。直接二分是 $O(\log^3 n)$ 的。我们发现它本质要求任意区间右边接一个线段树节点，询问

新区间的 FL 或 FR 函数。我们可以考虑直接新建一个线段树节点存储这个新区间的区间信息，注意每个节点需要存储它的左右儿子。我们发现合并两个区间后询问 FL 和 FR 仍然是 $O(\log n)$ 的，于是我们在线段树上二分时维护这个合并出的新节点即可将复杂度降到 $O(\log^2 n)$ 。

算法六：使用上文提到的方法。$O(\log^2 n)$ 求解 FL、FR，修改 ML、MR，$O(\log^2 n)$ 进行线段树上二分。

算法复杂度：总复杂度 $O(q \log^2 n)$ 。

期望得分：可以通过所有子任务。期望得分 100 分。

2.6 其他算法

由于本题的特殊性质，还有一些其他的算法未讨论，这里就略提一下。

算法七：暴力模拟时将水一格一格推进，维护一个单调队列。

算法复杂度：总复杂度 $O(nq)$ 。

期望得分：可以通过前四个子任务。期望得分 31 分。

算法八：上文中三个阶段中的第二阶段，可以直接进行暴力修改，当线段树节点区间内值单调时直接进行区间赋值操作。可以证明访问节点个数均摊 $O(\log n)$ ，但是由于线段树 pushdown 操作需要 $O(\log n)$ ，所以总复杂度不变。

算法复杂度：总复杂度 $O(q \log^2 n)$ 。

期望得分：可以通过所有子任务。期望得分 100 分。

3 测试数据设置

这题共 82 个测试点。共 11 种不同类型的随机数据，3 种满足特殊性质的数据，以及 4 种构造数据。其中构造数据兼顾了测试时间复杂度与算法精度。最终数据由使用 long double 的标程生成答案。

这里只提一下某种对于精度要求非常高的数据的构造方法。

设 m 为一个参数（约为 n 的三分之一），设 $S = \sum_{i=1}^{m} i$ 。由于本题输入中所有数字扩大 10^7 倍后都为整数，所以下文中直接用整数来表示扩大后的数。

将整个水槽分为三部分，第一部分负责提供水，第二部分是一个阶梯，第三部分负责接收水，第二部分水位高度依次递减。每一部分恰好 m 个水槽，并且设地平线为 S。

初始第一、二部分水位与隔板高度相同，第三部分没有水。每次将第一部分最右边的隔板高度降低恰好 S ，再将第三部分最左边的隔板降低到 S。保证这 S 体积水在第一次操作后会将第二部分水槽填平，第二次操作后会全部流入第三部分最左边的水槽内的地平线下，并且不留下坑。每次修改后询问第三部分的最左边的水槽的水位。

最后说一下为什么本题要求值域小于 100 ，并且要求七位绝对误差。首先我们可以改变水槽的隔板高度，对水位进行类似取出小数部分的操作，所以此时使用绝对误差或相对误差比较是本质相同的。而由于初始水量不变，如果有维护相同水位的连通块的算法，值域小的话就无法方便地构造出连通块数量变化大的数据，所以作者选择了使用比较九位有效精度。而这对于 double 来说精度要求相对较高，实现程序时应尽量使用误差小的浮点数运算方法。

4 难度估计

第一个子任务只需要简单的分析与模拟，预计做了这道题的选手都可以通过。

第二个子任务选手需要一些思考即可通过，预计 80% 的选手都可以通过。

第三、四个子任务选手需要对暴力算法进行简单的优化，或直接使用效率高的暴力算法，预计 60% 的选手都可以通过。

第五个子任务选手需要对这题进行深入地分析与研究，在仔细思考后得出做法，具有一定的思维难度。代码难度一般。预计 40% 的选手都可以通过。

第六个子任务选手需要对这题进行进一步地分析、研究、优化，代码难度较高。预计 20% 的选手可以通过。

本题除了具有思维难度、代码难度外，还要求选手对浮点数运算的精度误差进行仔细地分析与处理。若选手没有对操作进行仔细地分析，可能还会得出较复杂的做法。总体来说这题难度较高。

5 总结

这道题最后使用的线段树不同于普通的线段树。普通的线段树的每个节点本质是维护了信息的集合，每个节点的信息可以单独取出。而本题的线段树不仅维护了信息的集合，它还依靠二叉树形成了一个递归结构，对一个节点无论修改还是查询都要依赖于它的子节点。而我们在查询有特殊需要的时候还可以取出若干个节点形成一个二叉树外的递归结构，这与可持久化线段树的思想有些类似。

这道题正解所需要的知识只涉及了线段树，但是却考察了选手的模型转化能力、思维能力、问题分析能力、代码能力。虽然这是一道数据结构题，但是不同于传统的数据结构题，不仅考察了选手对数据的处理能力，对思维也有较高的要求。这道题并没有考察选手没有认真研究过、或者没有完全掌握的数据结构，而是考察了我们选手最了解的、最基础的、NOIP 选手都会用的线段树，这说明了一道好题并不需要依靠冷门的数据结构或知识，或者单纯依靠庞大的代码量来提高它的门槛。出题人希望通过这道题起到一个抛砖引玉的作用，希望能看到大家想出更多样、更有趣的数据结构题。

6 感谢

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张瑞喆的指导。
感谢南京外国语学校的李曙老师的关心与指导。
感谢王泽远同学与我讨论算法，帮忙验题。
感谢陈孙立同学为本文审稿。
感谢这些年陪伴我共同前行的同学们。
感谢所有培养我、指导我的老师们。
感谢父母对我的关心与支持。

参考文献

[1] Codechef Count on a Treap, Problem code : COT5

[2] 2017 multi-university training contest 5 - 1005.

浅谈图的点着色问题

广东省中山市中山纪念中学 高嘉煊

摘 要

本文将介绍图着色问题中的重要内容—点着色问题。点着色问题有两个重要的部分：色数与色多项式。本文会介绍关于色数的相关定理以及一些用于判断 k-可着色以及求图的色数的实用算法；同时，本文会介绍计算图的色多项式的删除收缩算法，并提出此算法在稀疏图和稠密图上的优化。

引言

图着色问题作为图论中的重要内容，从古至今一直是人们乐于研究的热门问题，如广为人知的四色定理。同时，图着色在调度问题，寄存器分配以及规划问题上有着重要的应用。

图着色问题有几种类型：点着色、边着色以及一些其他的着色问题，其中点着色问题是图着色问题中十分重要的一种。虽然在信息学竞赛中，也有着与点着色相关的问题，如计算一个图的色数，计算一个图的 k-着色数，但是这一类的题目的数据范围往往很小，通常是使用集合幂级数[18]作为标准解法。因此，作者就点着色问题进行了更深入的研究，希望能让点着色问题得到更多人的关注。

本文分成三部分：

在第一部分中，本文将给出文中将用到的一些记号以及定义。

在第二部分中，本文将围绕点着色问题中的色数给出相关的定理以及实用的算法。

在第三部分中，本文将介绍计算图的色多项式的删除收缩[19] 算法，并给出作者关于此算法所做的优化。

1 定义与约定

为了方便描述，以下给出一些定义。

[18]详情参见国家集训队 2015 论文集吕凯风的《集合幂级数的性质与应用及其快速算法》

[19]原文为：Deletion-Contraction Algorithm

如果没有特殊说明，本文中的图均指无重边无自环的无向图。

对于 $G=(V,E)$，本文中将使用 $n=|V|$ 来表示点集 V 的大小，用 $m=|E|$ 来表示边集 E 的大小。

定义 1.1. （点着色）点着色指的是在一张无向图 $G=(V,E)$ 中，给每个节点 x 分配一个颜色 $c(x)$，使得着色方案满足 $\forall(u,v)\in E, c(u)\neq c(v)$。

定义 1.2. （合法 k-着色方案）在无向图 G 中一种使用不超过 k 种颜色的着色方案被称作合法 k-着色方案；如果无向图 G 存在 k-着色方案，那么称 G 为**可 k-着色的**。

定义 1.3. （独立集）无向图 $G=(V,E)$ 的一个独立集定义为 V 的一个子集 S，满足 S 中的节点两两不相邻。形式化地，I 是 G 的一个独立集，当且仅当 $I\subseteq V$ 且 $\forall u,v\in I,(u,v)\notin E$

推论 1.1. 在无向图 G 的一个合法 k-着色方案中，任意一种颜色的节点集合都为 G 的一个独立集。

定义 1.4. （双连通图）定义无向图 G 是双连通的，当且仅当去掉 G 中任意一个节点之后，G 仍然是连通的。

定义 1.5. （双连通分量）无向图 G 的一个双连通分量定义为 G 中的一个极大双连通子图；G 中任意两个不同的双连通分量只包含至多一个相同节点，这样的节点被称为**割点**，一个双连通图中不存在割点。

定义 1.6. （导出子图）在无向图 $G=(V,E)$ 中，对于点集 $S\subseteq V$，S 在 G 上的导出子图定义为由 S 中的节点以及 E 中两端都是 S 中节点的边组成的子图，记作 $G[S]$。

定义 1.7. （团）一个无向图 $G=(V,E)$ 是团当且仅当 V 中任意两个不同的节点之间都有边相连，形式化地，$G=(V,E)$ 是团当且仅当 $\forall u,v\in V,(u,v)\in E$；用 K_n 表示一个大小为 n 的团。

定义 1.8. （最大团与团数）无向图 $G=(V,E)$ 的最大团，定义为一个最大的 V 的子集 S，使得 S 在 G 上的导出子图是团；同时，G 的最大团的大小被称为 G 的团数，记作 $\omega(G)$

定义 1.9. 在无向图 $G=(V,E)$ 中，用 $\Delta(G)$ 表示 G 中的最大度数，用 $\Delta(x)$ 表示节点 x 的度数。

为了方便描述，在本文中，将使用正整数对不同的颜色进行编号，在一种 k-合法着色方案中，用 $1..k$ 对这些颜色编号。

2 色数

2.1 定义

定义 2.1. （色数）定义无向图 G 的色数为一个最小的正整数 k，满足 G 是可 k-着色的；G 的色数也被记作 $\chi(G)$。

2.2 色数的界

关于色数 $\chi(G)$ 的界存在着许多有趣的定理以及结论，这里给出其中一些基本的结论。

结论 2.1. 在无向图 $G=(V,E)$ 上，色数不小于团数，即 $\chi(G)\geq\omega(G)$。

结论 2.2. 在无向图 $G=(V,E)$ 上，$\chi(G)(\chi(G)-1)\leq 2m$。

证明. 对于 G 的一个着色方案，如果存在两种颜色 p,q，使得不存在 $(u,v)\in E$ 满足 u 的着色为 p 且 v 的着色为 q，那么将所有颜色为 p 的点的颜色替换成颜色 q，得到的着色方案仍然是合法的。进而可以说明，在一个 $\chi(G)$-着色方案中，边数至少为 $\frac{\chi(G)(\chi(G)-1)}{2}$，即 $\chi(G)(\chi(G)-1)\leq 2m$，证毕。 □

2.3 贪心着色

通过以下贪心着色算法，我们可以认识到 $\chi(G)\leq\Delta(G)+1$。

贪心着色算法的算法流程如下：

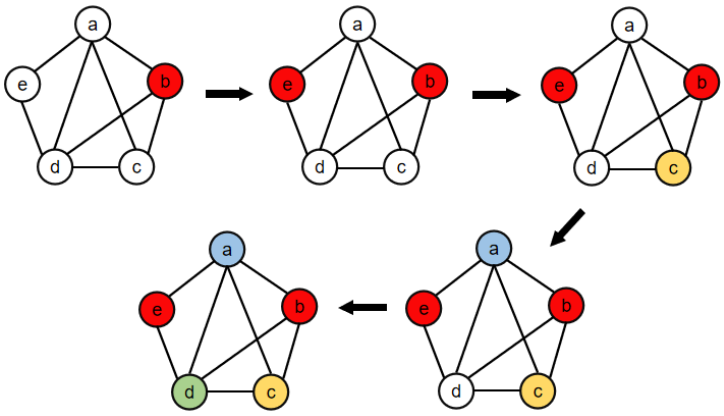
首先，选择一个序列 a，序列 a 是 G 中所有节点的一个排列，并且所有节点初始被设置为未着色的。

然后，按照从前到后的顺序，枚举 a 中每个节点，对于节点 x，记 c 为编号最小的满足此时与 x 相邻的节点中没有节点的着色为 c 的颜色，使用颜色 c 对节点 x 进行着色。

容易知道，每个节点的颜色编号必定小于等于与之相邻的点中排在它前面的点的数量加一。

按照上述过程，即可得到一个使用的颜色数不超过 $\Delta(G)+1$ 的着色方案。

举个例子，如下图中，着色序列为 $[b,e,c,a,d]$，红/黄/蓝/绿色的颜色编号分别为 $1,2,3,4$：



以上的贪心着色也被称为序列着色算法，选择的序列 a 对于算法的结果起到了至关重要的作用。

2.4 平面图

定义 2.2. （平面图）无向图 G 是平面图，当且仅当可以将 G 画在平面上，使得所有边只在节点处相交。

在平面图的点着色问题中，四色定理占据着极其重要的地位，其内容是：

定理 2.1. （四色定理）任意平面图都存在合法 4-着色方案。

由于四色定理不是本文的重点，故不在此花费过多篇幅介绍。

下面给出求平面图 6-着色方案的方法，首先有结论：

结论 2.3. 平面图中至少存在一个点的度数不超过 5。

请读者自行证明。

有了这个结论之后，不难得到一个对平面图 $G = (V, E)$ 求平面图 6-着色方案的算法：

先找到图中一个度数不超过 5 的节点 x，在 $G - x$ 中求出合法的 6-着色方案后，找到 x 相邻的节点中没有被使用的编号最小的颜色 c，使用颜色 c 对 x 进行着色，如此即可求得一个合法 6-着色方案。

2.5 弦图

定义 2.3. （弦图）无向图 G 是弦图，当且仅当对于所有 G 中的大小超过 3 的环 C，环 C 上都存在两个在环上不相邻的节点之间有边相连（这条边也被称作**弦**）。

定义 2.4. （完美消除序列）弦图 $G=(V,E)$ 中的一个完美消除序列定义为一个由 V 中所有节点排列而成的序列 P，其满足对于所有节点 v，与 v 相邻的节点中在序列 P 中排在 v 后面的所有节点与 v 在 G 上的导出子图是一个团。

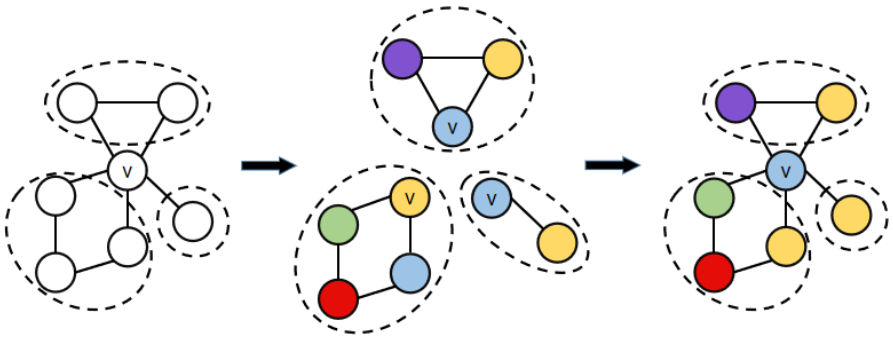
在弦图 $G=(V,E)$ 中，将 G 的完美消除序列翻转得到序列 P，根据序列 P 执行 2.3 节中的贪心着色算法，即可得到一个合法 $\omega(G)$-着色方案，由于对于任意图 $\chi(G)\geq\omega(G)$，那么可以知道，对于任意弦图 $G=(V,E)$ 都有 $\chi(G)=\omega(G)$。

2.6 Brooks 定理

定理 2.2. （*Brooks* 定理）当 $G=(V,E)$ 不是完全图且 G 不是奇环的时候，$\chi(G)\leq\Delta(G)$。

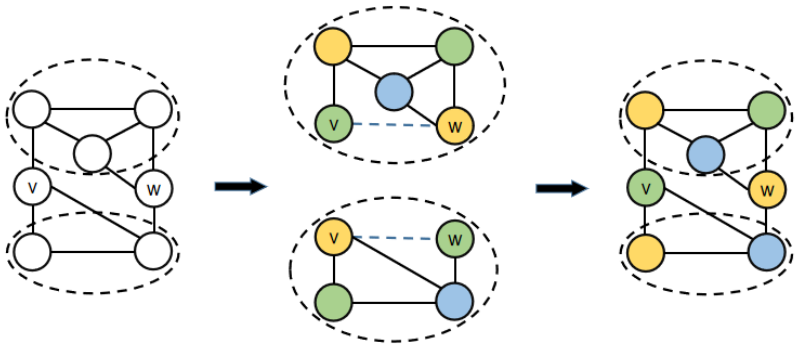
证明. 以下的证明将采用归纳法，记 d 为无向图 $G=(V,E)$ 的最大度数 $\Delta(G)$，记 n 为点数，即 $|V|$。对于 d，假设在 d 更小的情况下定理成立，$d\leq 2$ 时显然成立；对于 d,n，假设在 n 更小的情况下定理成立。归纳证明将从 $n=d+1$ 开始，显然这种情况下是成立的，因为 $G\neq K_{d+1}$，那么一定可以找到两个不相邻的节点，可以对这两个节点使用同样的颜色进行着色，就可得到一种合法的 d-着色方案。因此，下面的证明中将假设 $n\geq d+2$ 以及 $d>2$。

Case 1. G 不是一个双连通分量，即存在割点 v。记 $C_1,\ldots,C_t$ 为 $G-v$ 中的连通块的点集，根据归纳，可以对点集 $C_1\cup\{v\},\ldots,C_t\cup\{v\}$ 在 G 上的导出子图进行 d-着色，通过颜色的替换，可以把这些 d-着色方案合并起来得到 G 上的一个合法 d-着色方案。

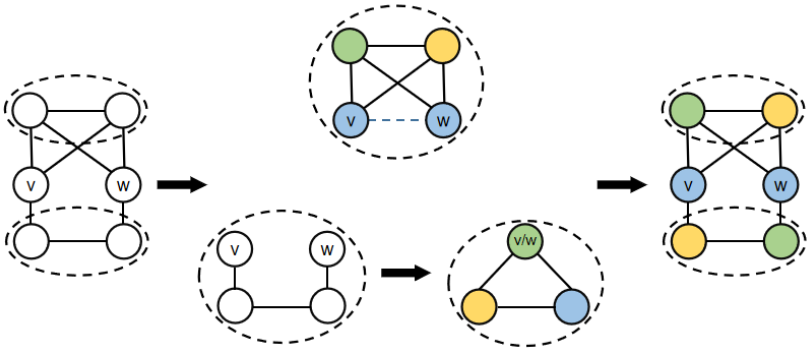


Case 2. G 中存在两个不相邻的点 v,w，满足 $G-v-w$ 是不连通的。记点集 A 为 $G-v-w$ 的其中一个连通块的点集，记点集 $B=V\backslash(A\cup\{v,w\})$。那么 v,w 都至少有一条连向点集 A 和 B 中节点的边。记 G_1 为点集 $A\cup\{v,w\}$ 在 G 上的导出子图，G_2 为点集 $B\cup\{v,w\}$ 在 G 上的导出子图。

若 G_1+vw 以及 G_2+vw 都不是 K_{d+1}，那么根据归纳可以对这两幅图求出合法 d-着色方案，通过替换颜色可以合并这两种方案（如下图）：



否则，假设 G_1+vw 是 K_{d+1}（A 在 G 上的导出子图是 K_{d-1}），那么可以把 v 和 w 都染上颜色 1，A 中的点依次染上颜色 2..d；通过合并 G_2+vw 中的 v 和 w 两个节点得到 G_3，根据归纳，G_3 存在合法 d-着色方案，替换颜色即可与前面的方案合并（如下图）。G_2+vw 是 K_{d+1} 的情况类似。



Case 3. G 中既不存在割点，也不存在两个点 v,w 使得 $G-v-w$ 是不连通的。记一个度数最大的节点为 x，在 x 的相邻节点中找到两个不相邻的节点 a,b，那么 $G-a-b$ 一定是连通图，在 $G-a-b$ 中以 x 为起点执行广度优先遍历 (bfs) 得到序列 P，将序列 P 翻转然后将 a,b 加到序列最前面得到序列 Q，根据这个序列执行在 2.3 节中的贪心着色算法，即可得到一个合法的 d-着色方案。

因为开头是 a,b，那么 a,b 的颜色都为 1；对于 $V\backslash\{a,b,x\}$ 中的任意节点，都有至少一个相邻节点在 Q 中排在它后面；对于 x，由于 a,b 的颜色都为 1，那么一定可以在 x 点使用编号不超过 d 的颜色。 □

2.7 判断 k-可着色

本节会给出一些判断无向图 $G=(V,E)$ 是否 k-可着色的算法。

对于 $k=2$

判断一个图是否 2-可着色，即判断这个图是否是二分图，直接对图进行黑白染色即可，时间复杂度 $O(n+m)$。

对于 $k=3$

判断一个图是否 3-可着色，可以使用钟知闲在 2017 年国家集训队论文《浅谈信息学竞赛中的独立集问题》提到的用于寻找图的极大独立集的 Bron-Kerbosch 算法，时间复杂度是 $O(3^{\frac{n}{3}}n^2)$

对于任意 k

在 k 任意的情况下，存在使用动态规划和极大独立集搜索算法的做法，其时间复杂度为 $O(2.445^n)$，作者这里给出使用集合幂级数判断图是否 k-可着色的算法。

集合并卷积

集合并卷积的定义如下：

对于两个关于 x 集合幂级数 f 和 g，设 $f=\sum_{S\subseteq 2^U} f_S x^S, g=\sum_{S\subseteq 2^U} g_S x^S$，定义 $f\cdot g=h$，其中 h 是一个集合幂级数，且 h_S 满足：

$$h_S=\sum_{L\subseteq 2^U}\sum_{R\subseteq 2^U}[L\cup R=S]f_L g_R$$

定义集合幂级数 f 的**莫比乌斯变换**为：

$$\hat{f}_S=\sum_{T\subseteq S} f_T$$

使用分治算法可以求出集合幂级数 $\hat{f}$。

反过来，可以定义 $\hat{f}$ 的**莫比乌斯反演**为 f。

通过容斥原理可以得到：

$$f_S=\sum_{T\subseteq S}(-1)^{|S|-|T|}\hat{f}_T$$

对 (1) 式两端同时求莫比乌斯反演可以得到：

$$\begin{aligned}\hat{h}_S&=\sum_{L\subseteq 2^U}\sum_{R\subseteq 2^U}[L\cup R\subseteq S]f_L g_R\\&=\left(\sum_{L\subseteq 2^U}[L\subseteq S]f_L\right)\left(\sum_{R\subseteq 2^U}[R\subseteq S]g_R\right)\\&=\hat{f}_S\hat{g}_S\end{aligned}$$

那么，对 f,g 分别求莫比乌斯变换得到 $\hat{f},\hat{g}$，即可求出 $\hat{h}$，再对 $\hat{h}$ 求莫比乌斯反演得到 h。

时间复杂度 $O(n2^n)$

在信息学竞赛的题目中，往往都会给出一个模数 P，集合幂级数的系数是在模 P 的整数环上进行运算的。

■

考虑使用集合并卷积来判断无向图 G 是否可以被 k-着色。

由点着色的定义可以知道每种颜色的节点集合都是独立集，那么判断无向图 $G=(V,E)$ 是否可以被 k-着色等价于判断能否在无向图 $G=(V,E)$ 中找到 k 个独立集 $P_1,\ldots,P_k$，使得 $P_1\cup P_2\cup\cdots\cup P_k=V$

那么可以得到如下算法：

对于无向图 $G=(V,E)$，定义集合幂级数 f，其中：

$$f=\sum_{S\subseteq V}[S\text{是}G\text{中的独立集}]x^S$$

求出 f 的莫比乌斯变换 $\hat{f}$

求出 $\hat{g}$，其中 $\hat{g}_S=\hat{f}_S^k$

对 $\hat{g}$ 求莫比乌斯反演得到 g，若 g_V 为 0，那么说明 G 不存在合法 k-着色方案，否则说明 G 可以被 k-着色。

以上算法的时间复杂度是 $O(2^n n)$。

2.8 求图的色数

在求无向图 $G=(V,E)$ 的色数的时候，可以沿用判断图能否被 k-着色中使用集合并卷积的思路。

算法的大体部分是相同的，不同点在于从小到大枚举 k 再判断 G 能否被 k-着色。

算法的流程如下：

对于无向图 $G=(V,E)$，定义关于 x 的集合幂级数 f，其中：

$$f=\sum_{S\subseteq V}[S\text{是}G\text{中的独立集}]x^S$$

求出 f 的莫比乌斯变换 $\hat{f}$。

从小到大枚举 k，记集合幂级数 $\hat{g}$，其中 $\hat{g}_S=\hat{f}_S^k$，使用容斥原理在 $O(2^n)$ 时间内计算出 $g_V=\sum_{S\subseteq V}(-1)^{|V|-|S|}\hat{g}_S$，若 $g_V\neq 0$，那么就可以认为 G 的色数 $\chi(G)=k$。

以上算法的时间复杂度是 $O(2^n n)$

3 色多项式

3.1 定义与约定

定义 3.1. （k-着色数）定义无向图 G 的 k-着色数为使用 k 种颜色，对 G 进行合法着色的方案数，两个方案不同当且仅当存在一个点在两种方案中的颜色不同。

定义 3.2. （色多项式）定义无向图 G 的色多项式 $P(G,x)$ 为一个关于 x 的多项式，当使用 k 种颜色进行着色时，将 $x=k$ 代入 $P(G,x)$ 得到的数值为 G 的 k-着色数。

定义 3.3. （k-独立集划分）定义无向图 $G=(V,E)$ 的一个 k-独立集划分方案为：将点集 V 划分成 k 个非空集合 $S_1,S_2,\ldots,S_k$，每个节点都属于恰好一个集合，并且满足 $\forall i\in[1,k]$，S_i 都是 G 的独立集。两种 k-独立集划分方案是不同的，当且仅当，存在两个节点 u,v，满足这两者在一个方案中处于同一集合而在另一方案中处于不同的集合。

定义 3.4. （独立集划分序列）定义无向图 $G=(V,E)$ 的独立集划分序列为一个序列 h，其中 h_k 表示 G 的 k-独立集划分方案数；容易知道，$h_n=1$ 且 $\forall i>n, h_i=0$。

定义 3.5. （独立集划分多项式）对于无向图 G，记其独立集划分序列为 h，定义无向图 G 的独立集划分多项式为 $H(G,x)=\sum_i h_i\cdot x^i$。

一般情况下，一定规模的图中色多项式的系数会很大，在信息学竞赛中，一般考虑的是这些系数对某模数取模的结果（例如 998244353 或 10^9+7），故在之后的论述中，我们假定所有的运算都是在模某足够大的质数 M 的意义下进行的。

3.2 性质

性质 3.1. 对于无向图 G，其色多项式 $P(G,x)=\sum_i a_i\cdot x^i$ 和独立集划分序列 h 之间存在着如下关系：

$$P(G,x)=\sum_i h_i\cdot x^{\underline{i}}$$

其中 $x^{\underline{i}}$ 表示 x 的 i 下降幂，即 $x^{\underline{i}}=\prod_{j=1}^{i}(x-i+j)$

由于 $h_n=1$ 且对于 $i(i>n)$，都有 $h_i=0$，那么多项式 $P(G,x)$ 的次数等于 n。

性质 3.2. 无向图 G 的色数为最小的使得 $P(G,k)$ 非零的整数 k，形式化的，$\chi(G)=\min\{k\in\mathbb{N}:P(G,k)>0\}$

由色数的定义可知性质 3.2. 的正确性。

性质 3.3. 对于无向图 $G=(V,E)$，$P(G,-1)\times(-1)^{|V|}$ 等于 G 的无环定向[20]方案数，其中无环定向指的是给 G 中的每条边确定一个方向得到有向图 G'，使得 G' 是有向无环图的方案数，两种方案不同当且仅当有一条边 (u,v) 在两种方案中的方向不同。

证明略。

性质 3.4. 对于无向图 $G=(V,E)$，假设 G 中有 c 个连通块 $G_1,\ldots,G_c$，那么在 G 的色多项式 $P(G,x)$：

- $x^0,\ldots,x^{c-1}$ 这些项的系数都是 0。
- $x^c,\ldots,x^n$ 这些项的系数都是非零的，且它们的正负性交替变化。
- x^n 这一项的系数为 1
- x^{n-1} 这一项的系数为 $-|E|$。
- $P(G,x)=P(G_1,x)P(G_2,x)\cdots P(G_c,x)$

性质 3.5. G 是树当且仅当 $P(G,x)=x(x-1)^{n-1}$。

3.3 特殊图的色多项式

在一些特殊的图中，色多项式是可以直接求出的：

完全图 K_n	$x(x-1)(x-2)\ldots(x-(n-1))$
无边图 $\overline{K}_n$	x^n
有 n 个节点的树	$x(x-1)^{n-1}$
简单环 C_n	$(x-1)^n+(-1)^n(x-1)$

在弦图中，除了色数 $\chi(G)$ 等于团数 $\omega(G)$，弦图的色多项式也是可以直接计算的，对于弦图 $G=(V,E)$，求出 G 的一个完美消除序列 P，记 x_i 表示节点 P_i 相邻的节点中在 P 中排在它后面的节点数量，那么 $P(G,x)=(x-x_1)(x-x_2)\ldots(x-x_n)$。

3.4 集合幂级数

求图的色数可以使用集合幂级数，求图的色多项式也可以使用集合幂级数。

首先考虑如何计算用 k 种颜色对图 $G=(V,E)$ 进行着色的方案数。

定义关于 x 的集合幂级数 f，其中 $f=\sum_{S\subseteq V}[S\text{是}G\text{中的独立集}]\cdot u^{|S|}\cdot x^S$

[20]原文为：Acyclic orientation

注意，这里的集合幂级数与 2.8 节中不同的地方在于，f_S（x^S 的系数）不再是单纯的数，而是一个关于 u 的多项式，与其相关的运算是在模 u^{n+1} 意义下进行的。

求出 f 的莫比乌斯变换 $\hat{f}$。

计算集合幂级数 $\hat{g}$，其中 $\hat{g}_S = \left(\hat{f}_S\right)^k$。

计算 $\hat{g}$ 的莫比乌斯反演 g，g_V 中 u^n 的系数即为用 k 种颜色对 G 进行着色的方案数。

取 $k = 1..n$，用以上算法可以在 $O(2^n n^3)$ 的时间复杂度内计算出使用 $1..n$ 种颜色对 G 进行着色的方案数，记 $w(k)$ 表示用 k 种颜色进行着色的方案数。

由于 G 的色多项式 $P(G,x)$ 是一个次数为 n 且常数项为 0 的多项式，前面得到的 $w(1..n)$ 实际上是 $P(G,x)$ 在 x 取值为 $1..n$ 时对应的点值，故可以使用拉格朗日插值求出 $P(G,x)$。

拉格朗日插值

对于一个关于 x 的多项式 $F(x)$，其次数为 c，已知 $F(x)$ 上 $c+1$ 个两两不同的点 $(x_0,y_0),(x_1,y_1),\ldots,(x_c,y_c)$，那么：

$$F(x) = \sum_{i=0}^{c} y_i \cdot \prod_{i \neq j} \frac{x - x_j}{x_i - x_j}$$

■

使用拉格朗日插值，可以利用点值 $(0,0),(1,w(1)),(2,w(2)),...,(n,w(n))$ 计算出 $P(G,x)$

3.5 删除收缩算法

定义 3.6. （边收缩）在图 $G=(V,E)$ 中，对边 $e=(u,v)$ 进行收缩指的是：将 e 从 G 中去掉，并将节点 u 和节点 v 合并成一个新的节点 w，与 w 相连的边对应着一条原本与 u 或 v 相连的边。

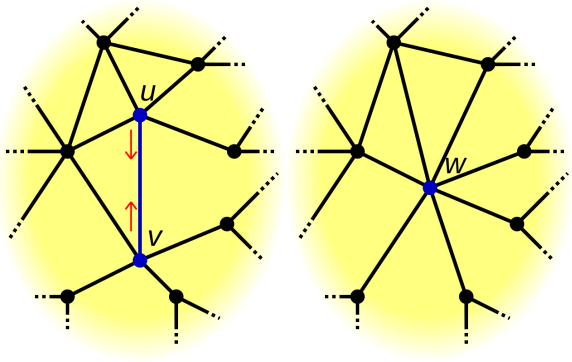
更形式化地，在 $G=(V,E)$ 中，对边 (u,v) 进行收缩得到一个新的图 $G'=(V',E')$。

首先将 u,v 合并成一个新的节点 w，那么点集 $V'=\{w\}\cup(V\backslash\{u,v\})$。

然后，在边集 E' 中，对于所有满足 $x\notin\{u,v\}$且$y\notin\{u,v\}$ 的 x,y，$(x,y)\in E'$ 当且仅当 $(x,y)\in E$；对于所有满足 $x\notin\{u,v\}$ 的 x，$(w,x)\in E'$ 当且仅当 $(u,x)\in E$ 或 $(v,x)\in E$。

如下图[21]。

[21]图片来自 Wikipedia 中的 Contraction 词条



顾名思义，删除收缩算法的内容是基于边的删除以及收缩的：

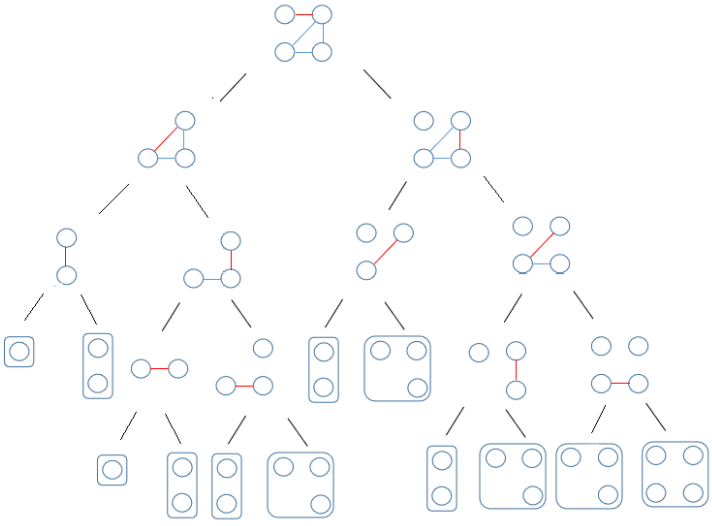
对于 G 中的一条边 (u,v)，有

$$P(G,x) = P(G-uv,x) - P(G/uv,x) \tag{1}$$

其中 $G-uv$ 表示将边 (u,v) 从 G 中删掉得到的图，G/uv 表示在 G 上对边 (u,v) 进行收缩得到的图。

直接根据上述等式进行计算，就可以得到最基本的删除收缩算法。

举个例子，下图中给出了对一个图执行删除收缩算法的过程：



由于上述算法递归的形式与斐波那契数列[22]类似，在最坏情况下，其复杂度为：

$$\left(\frac{1+\sqrt{5}}{2}\right)^{|V|+|E|} \in O(1.62^{|V|+|E|})$$

[22]斐波那契数列满足 $f_0 = 1, f_1 = 1, f_i = f_{i-1} + f_{i-2} (i \geq 2)$

遗憾的是，在大部分情况下，这个算法的效率远远劣于使用集合幂级数进行计算的复杂度。

下面将对删除收缩算法进行优化。

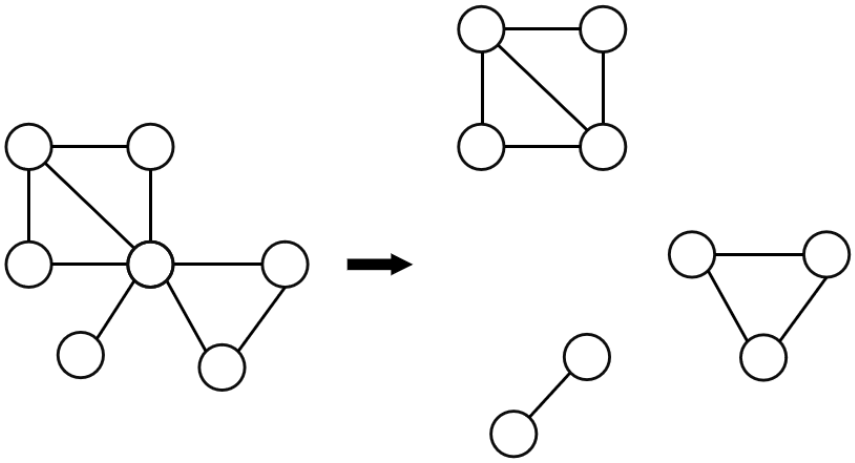
3.6 删除收缩算法在稀疏图上的优化

3.6.1 双连通分量拆分

对于 $G=(V,E)$，假设在 G 中有 k 个双连通分量 $C_1,C_2,\ldots,C_k$，并且 G 中有 s 个连通块，那么：

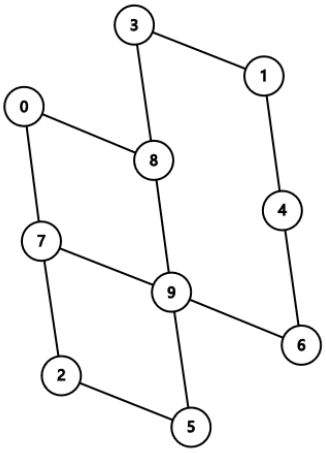
$$P(G,x)=\frac{1}{x^{k-s}}\prod_{i=1}^{k}P(C_i,x)$$

对于 G，先用 tarjan 算法将 G 拆分成若干双连通分量，求出每个双连通分量的色多项式之后，用上述式子计算 G 的色多项式。



3.6.2 二度路径

通过观察，容易发现，在稀疏图中，有许多节点的度数都不超过 2!



当节点 u 在 G 中的度数为 0 时，$P(G,x) = P(G-u,x) \cdot x$

当节点 u 在 G 中的度数为 1 时，$P(G,x) = P(G-u,x) \cdot (x-1)$

对于度数为 2 的节点，为了减少运算次数，考虑对多条边同时进行删除收缩，并根据前面总结出的特殊图上的色多项式进行快速计算。

为了更好的阐述，引入“二度路径”的概念。

定义 3.7. （二度路径）G 中的一条二度路径定义为一条除了起点以及终点以外的节点在 G 中的度数都是 2 的简单路径。形式化的，一个长度为 k 的路径 $P = (p_1, p_2, \ldots, p_{k+1})$ 是一条二度路径，当且仅当其满足：

- P 是一条简单路径。
- $\forall i \in [2,k]$，$\Delta(p_i) = 2$

不妨先来回顾一下链与环的情况。

一条长度为 l 的链的色多项式是：$x(x-1)^{l-1}$

一个长度为 l 的环的色多项式是：$(x-1)^l + (-1)^l (x-1)$

对于一条长度为 k 的二度路径 $P = (p_1, p_2, \ldots, p_{k+1})$，如果已经确定了 p_1 和 p_{k+1} 的颜色，那么 $p_2, \ldots, p_k$ 的着色方案数是多少呢?

分成两种情况来讨论：

情况 1. 对 p_1 和 p_{k+1} 着色的颜色相同。

这种情况下可以视为求一个长度为 k 的环的色多项式，即 $(x-1)^k + (-1)^k (x-1)$，由于已经确定了 p_1 和 p_{k+1} 的着色，那么对应的方案数为：

$$\frac{(x-1)^k + (-1)^k (x-1)}{x}$$

情况 2. 对 p_1 和 p_{k+1} 着色的颜色不同。

这种情况可以看成是用长度为 k 的链的着色方案数减去长度为 k 的环的着色方案数，即：

$$x(x-1)^k-(x-1)^k-(-1)^k(x-1)=(x-1)^{k+1}-(-1)^k(x-1)$$

由于已经确定了 p_1 和 p_{k+1} 的着色，那么 $p_2,\ldots,p_k$ 的着色方案数是：

$$\frac{(x-1)^k-(-1)^k}{x}$$

■

那么，对于一条二度路径 P，记 G_1 表示将 G 中 P 上除了起点和终点之外的节点都删掉得到的图，G_2 表示将 G 中 P 上的所有边进行收缩得到的图。

根据上面两种情况，可以得到：

$$\begin{aligned}P(G,x)&=\frac{(x-1)^k-(-1)^k}{x}(P(G_1,x)-P(G_2,x))+\frac{(x-1)^k+(-1)^k(x-1)}{x}\cdot P(G_2,x)\\&=\frac{(x-1)^k-(-1)^k}{x}\cdot P(G_1,x)+(-1)^k\cdot P(G_2,x)\end{aligned}$$

实际上，$P(G,x)=P(G-uv,x)-P(G\backslash uv,x)$ 是这个式子在 $k=1$ 时的情况。

在 G 中，首先找到一条极长的二度路径 P，对 P 使用上述式子进行计算即可。

3.7 删除收缩算法在稠密图上的优化

在稠密图 $G=(V,E)$ 中，对于一条边 $(u,v)\notin E$，有以下等式：

$$P(G,x)=P(G+uv,x)+P(G\backslash uv,x)\tag{2}$$

(2) 式实际上是 (1) 式的变形。

为了更好的利用稠密图的特殊性质，考虑首先求出 G 的独立集划分多项式 $H(G,x)$，然后根据性质 3.1 利用 $H(G,x)$ 求出 $P(G,x)$。

对于 $H(G,x)$，有：

$$H(G,x)=H(G+uv,x)+H(G\backslash uv,x)$$

为了方便描述，记 $\overline{H}(G,x)=H(\overline{G},x)$。

3.7.1 割点

与稀疏图中的情况不同的是，这里将会对稠密图的补图的割点进行拆分。

对于 $\overline{G}$ 中的一个割点 u，记 G_1 为 $\overline{G}$ 中一个包含 u 的双连通分量，G_2 为 $\overline{G}$ 中除去 $G_1 - u$ 以外的节点在 $\overline{G}$ 上的导出子图。

那么：

$$H(G,x) = \overline{H}(G_1,x)\overline{H}(G_2-u,x) + \overline{H}(G_2,x)\overline{H}(G_1-u,x) - x\cdot\overline{H}(G_2-u,x)\overline{H}(G_1-u,x)$$

需要注意的是，上式与 3.6.1 节中得到的结果有很大区别，其优点在于优化了 (2) 式中与点数、边数有关的递归形式。

3.7.2 二度路径

在稠密图的补图中，有许多度数为 2 的节点，受到 3.6.2 节的启发，可以考虑对补图中一条极长的二度路径进行优化。

记简单路径 $P = (p_1, p_2, \ldots, p_{k+1})$ 为 $\overline{G}$ 中一条极长的二度路径，记 $u = p_1, v = p_{k+1}$

当 $k = 1$ 时，那么直接对 G 计算：

$$H(G,x) = H(G+uv,x) + H(G\backslash uv,x)$$

当 $k = 2$ 时，直接对 (u, p_2) 计算 $H(G,x) = H(G+up_2,x) + H(G\backslash up_2,x)$。

当 $k > 2$ 时，首先在 G 上对边 (u, p_2) 计算 $H(G,x) = H(G+up_2,x) + H(G\backslash up_2,x)$；在得到的两个图中再对 (p_k, v) 计算之前的式子。

在上述得到的无向图 G' 中，会出现 $\overline{G'}$ 的某一连通块为链的情况。

对于一个点数为 r 补图为的链 L_r 的图，其独立集划分多项式为：

$$H(\overline{L_r},x) = \sum_{i=\lceil \frac{r}{2} \rceil}^{r} \binom{i}{r-i} \cdot x^i$$

可以看出，在稀疏图中与在稠密图的补图中进行关于二度路径的优化有着很大的不同，前者的情况较为简单，而后者则相对复杂，仅仅只是提出了一个对边的计算次序，形式也并没有前者优美，本质上是为了尽量分割出补图为链的连通块，然后利用式子来减少运算次数。

3.8 删除收缩算法的其他优化

在删除收缩算法中，对于一些如环、树之类的特殊图，可以直接计算出其色多项式。

在实际应用中，常常会使用到图的同构判定相关的做法来减少删除收缩算法的计算次数。

总结

针对色数，本文给出了色数在图中的一些基本的界，给出了判断 k-可着色以及计算图的色数的算法。

针对色多项式，本文给出了色多项式的性质以及计算色多项式的删除收缩算法，并针对稀疏图以及稠密图给出了对其所做的优化，为了更好的阐述，文中引入了”二度路径”的概念，在针对稠密图所做的优化中，文中引入了”独立集划分序列”以及”独立集划分多项式”的概念以对稠密图进行优化。

经典的使用集合幂级数计算色多项式的算法由于其时间效率以及空间效率只能应用于点数较小的图中，而删除收缩算法针对稀疏图所做的优化使得在点数更大的图中计算色多项式成为了可能。

点着色问题中仍然有更多有趣的内容，希望本文可以引起读者的思考，能让更多人关注点着色问题。

致谢

感谢中国计算机学会提供学习与交流的机会。
感谢张瑞喆教练的指导与帮助。
感谢父母对我多年来的培养与关心。
感谢中山纪念中学的宋新波老师，熊超老师多年来给予的关心与帮助。
感谢中山纪念中学曹天佑同学，杭州学军中学范致远张哲宇同学为本文验稿。
感谢给予我鼓励与帮助的老师以及同学。

参考文献

[1] 图着色问题的英文维基百科，
https://en.wikipedia.org/wiki/Graph_coloring

[2] 色多项式的英文维基百科，
https://en.wikipedia.org/wiki/Chromatic_polynomial

[3] 钟知闲，《浅谈信息学竞赛中的独立集问题》，IOI2017 国家集训队论文。

[4] 吕凯风，《集合幂级数的性质与应用及其快速算法》，IOI2015 国家集训队论文。

浅谈格路计数相关问题

南京外国语学校 戴言

摘 要

格路模型是信息学竞赛中一种常见的问题模型，其中较为常见的一种是 Dyck 路计数的相关问题。作者注意到在算法竞赛中，选手通常采用生成函数或动态规划方法解决这类问题。作者经过对这一类问题的认真研究，发现这类问题可以通过构造双射快速解决。本文给出了两种特殊的 Dyck 路的计数方法，以及他们在有峰个数限制时的计数方法。同时，作者也研究了一些不相交格路的相关计数问题，并给出了这两个问题之间的联系。

1 引言

格路问题是组合数学中的经典模型，在信息学竞赛中十分常见。而其中非常多的问题都以 Dyck 路计数作为问题模型，例如 NOI 2018 第一天第二题「冒泡排序」[23]。

而构造双射是组合计数中的常见做法，与信息学竞赛选手常用的生成函数或动态规划算法相比，更加快速、优美，且实现难度较低，时间复杂度在多数情况下也可以得到优化。但目前的 OI 圈中，这种解题思想较为少见。

第 2 节中，作者主要讨论了两种特殊 Dyck 路：n,m 互质时的 (n,m)-Dyck 路，以及 n 阶 t-Dyck 路的计数，并给出了他们在限制峰的个数时的计数方法。

第 3 节中，作者主要讨论了不相交 Dyck 路，不相交自由路，以及不接触自由路的计数问题，并给出了第 2 节分与第 3 节讨论问题之间的一个联系。

第 4 节中，主要给出了 Dyck 路计数在实际生活中的一个应用。

2 Dyck 路

2.1 格路

定义 2.1. *在平面直角坐标系中，横坐标和纵坐标都是整数的点称为***格点***，***平面格路***是指从一个格点到另一格点只走格点的路，格路的***长度***是指其所走的路的步数。*

[23]http://uoj.ac/problem/394

2.2 自由路

定义 2.2. 对于一条从 $(0,0)$ 到 (n,m) 的格路，若其只使用了上步 $U=(0,1)$，水平步 $L=(1,0)$，则我们称其为(n,m) **自由路（*free path*）**。

定理 2.1. 记 $\mathcal{F}(n,m)$ 为 (n,m) 自由路的集合，$F(n,m)=\#\mathcal{F}(n,m)$ 为 (n,m) 自由路的数量，即 $\mathcal{F}(n,m)$ 的元素个数。则 (n,m) 自由路的个数为：

$$F(n,m)=\binom{n+m}{n} \tag{3}$$

证明. 我们可以发现，一条 $(0,0)$ 到 (n,m) 的自由路可以唯一地对应到一个含有 n 个水平步 L 和 m 个上步 U 的序列。而每个这样的序列可以唯一对应上一条这样的自由路。因此，(n,m) 自由路的条数等于从 $n+m$ 个位置中选出 n 个 L 的方案数，即 $\binom{n+m}{n}$。 □

2.3 (n,m)-Dyck 路的计数

定义 2.3. 对于一条从 $(0,0)$ 到 (n,m) 的自由路，若其始终不经过对角线 $y=\frac{m}{n}x$ 下方，则我们称之为(n,m)***-Dyck* 路**。

记 $\mathcal{D}(n,m)$ 为 (n,m)-Dyck 路的集合，$D(n,m)=\#\mathcal{D}(n,m)$ 为 (n,m)-Dyck 路的数量。

我们考虑一条 (n,m)-Dyck 路对应的 LU 序列。

定义 2.4. 对于从 $(0,0)$ 到 (n,m) 的 2 条格路 P,Q，其中 $P=u_1u_2\cdots u_{n+m}$,$Q=v_1v_2\cdots v_{n+m}$ $(u_i,v_i\in\{L,U\},i=1,2,\cdots,n+m)$。若 $\exists i$，$u_{i+1}\cdots u_{n+m}u_1\cdots u_i=v_1v_2\cdots v_{n+m}$，则我们称格路 P,Q **等价**。将 P 的等价格路全集记为 $[P]$。

定义 2.5. 对于任意格路 P,记 $P_k=u_{k+1}u_{k+2}\cdots u_{n+m}u_1\cdots u_k$,则 $[P]=\{P_k\mid k=1,2,3,\cdots,n+m\}$。定义 P 的**周期**为使得 $P=P_k$ 的最小数 k，用 $period(P)$ 表示，则显然有 $\#[P]=period(P)$。

接下来，我们给出如下两条引理：

引理 2.1. 若 $(n,m)=1$，即 n,m 互质，则 $\forall P\in\mathcal{F}(n,m)$，有：

$$period(P)=n+m \tag{4}$$

证明. 显然有 $period(P)\le n+m$。下面用反证法证明：$period(P)=n+m$。

若 $period(P)<n+m$，设为 r。设 $P=u_1u_2\cdots u_{n+m}$，$n+m=a\cdot r+b$（$b<r$）。则由周期的定义可知，$P=P_r=P_{2r}=\cdots=P_{ar}$。

若 $b\neq 0$，则有 $P=P_t$，而 $t<r$，因此这与 $period(P)$ 的定义矛盾。故 $b=0$，即 $n+m=a\cdot r$。因为 $r<n+m$，所以 $a>1$。

设 $u_1u_2\cdots u_r$ 中有恰好 x 个 L, y 个 U, 则有: $n=a\cdot x$, $m=a\cdot y$。因此有 $(n,m)=a\neq 1$, 矛盾。

故 $period(P)$ 不可能小于 $n+m$。因此 $period(P)=n+m$。 □

引理 2.2. 当 n 和 m 互质时 [P] 有且仅有一条 (n,m)-*Dyck* 路。

证明. 首先证明其存在性:

若 P 不是一条 (n,m)-Dyck 路，则其一定至少存在一个点在对角线下方。设这些点当中离对角线最远的点是 v，从这一点断开，将 P 分为两条子路 L_1，L_2。则 P 可以表示为 $P=L_1L_2$。构建一条新的格路: $\tilde{P}=L_2L_1$, 则显然 $\tilde{P}$ 是一条 (n,m)-Dyck 路。故存在性得证。

然后证明其唯一性:

由上述过程，我们可以得到我们要证明在对角线下方，且距离对角线最远的点有且只有一个即可。考虑反证:

如果有 2 个点，设为 $v_1(x_1,y_1)$，$v_2(x_2,y_2)$（$0<x_1<x_2<n$，$0<y_1<y_2<m$）。则这 2 个点形成的直线与对角线 $y=\frac{m}{n}x$ 平行。则有他们的斜率相等，即:

$$k=\frac{y_2-y_1}{x_2-x_1}=\frac{m}{n}$$

而这与条件 n 和 m 互质矛盾。故唯一性得证。 □

由上述两个引理可以立即得出:

定理 2.2. 当 (n,m) 互质时，(n,m)-*Dyck* 路的数量为:

$$D(n,m)=\frac{1}{(n+m)}\binom{n+m}{n} \tag{5}$$

2.4 有 k 个峰的 (n,m)-Dyck 路计数

定义 2.6. 对于一条从 $(0,0)$ 到 (n,m) 的自由路中的连续两步，若其为 UL，则我们称之为一个**峰**（***peak***）；若其为 LU，则我们称之为一个**谷**（***valley***）。

定理 2.3. 记 $\mathcal{F}(n,m;k)$ 为所有有恰好 k 个峰的 (n,m) 自由路的集合, $F(n,m;k)=\#\mathcal{F}(n,m;k)$。则从 $(0,0)$ 到 (n,m) 的 k 个峰的自由路数量为:

$$F(n,m;k)=\binom{n}{k}\binom{m}{k} \tag{6}$$

证明. 我们将峰 UL 之间的格点叫做**峰点**。任取一条格路 $P\in\mathcal{F}(n,m;k)$，设 P 的 k 个峰点分别为 $(x_1,y_1),(x_2,y_2),\cdots,(x_k,y_k)$。则峰点坐标满足:

$$0\leq x_1<x_2<\cdots<x_k\leq n-1, 0\leq y_1<y_2<\cdots<y_k\leq m$$

我们考虑先选出 $x_1, x_2, \cdots, x_k$，然后再选出 $y_1, y_2, \cdots, y_k$。可知满足条件的 k 个整点有 $\binom{n}{k}\binom{m}{k}$ 个。

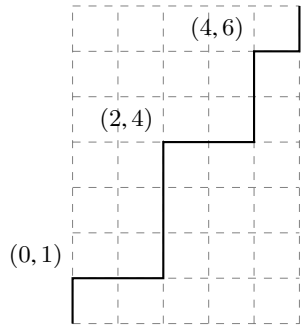


图 1: $n = 5$，$m = 7$，选出的峰点坐标为：$(0,1), (2,4), (4,6)$

在选出这些点后，我们用连续的 U 或者连续的 L 连接这 k 个点，即可得到一条满足条件的自由路。

容易发现，任意一条自由路都可以唯一对应到这些峰点上，而任意一组峰点选择都可以唯一确定一条自由路。换言之，自由路与峰点选择是一一对应的关系，故自由路条数即为峰点选择的方案数 $\binom{n}{k}\binom{m}{k}$。 □

定理 2.4. *记 $\mathcal{F}^{UL}(n,m;k)$ 为所有有恰好 k 个峰，且首步为 U，末步为 L 的 (n,m) 自由路的集合，$F^{UL}(n,m;k) = \#\mathcal{F}^{UL}(n,m;k)$。则有：*

$$F^{UL}(n,m;k) = \binom{n-1}{k-1}\binom{m-1}{k-1} \tag{7}$$

证明. $\mathcal{F}^{UL}(n,m;k)$ 中的元素均以 U 起始、以 L 结束，因此第一个峰的横坐标以及最后一个峰的纵坐标已经确定。故选出 k 个整数点的方法共有 $\binom{n-1}{k-1}\binom{m-1}{k-1}$ 种。

我们类似定理 2.3 中的证明进行构造，即可得到一条以 U 起始、以 L 的自由路。故自由路条数即为峰点选择的方案数 $\binom{n-1}{k-1}\binom{m-1}{k-1}$。 □

定理 2.5. *记 $\mathcal{D}(n,m;k)$ 为所有有恰好 k 个峰的 (n,m)-Dyck 路的集合，$D(n,m;k) = \#\mathcal{D}(n,m;k)$。则当 n,m 互质时，恰有 k 个峰的 (n,m)-Dyck 路的个数为：*

$$D(n,m;k) = \frac{1}{k}\binom{n-1}{k-1}\binom{m-1}{k-1} \tag{8}$$

证明. 对任意的格路 P，若其有 k 个峰，则其必然有 $k-1$ 个谷，设为 $v_1, v_2, \cdots, v_{k-1}$。我们把 P 在谷点断开，则其可分为 k 条子路：$L_1, L_2, \cdots, L_k$，且每条子路都包含恰好 1 个峰。有 $P = L_1 L_2 \cdots L_k$。

对于任意一条 Dyck 路，在其 k 个峰点处都可以映射为一条自由路：对第 i 个峰，若所在子路为 L_i，则构建 $\tilde{P} = L_{i+1}L_{i+2}\cdots L_kL_1\cdots Li$。因此，每条 Dyck 路的每个峰都可以唯一对应恰好一个 $\mathcal{F}^{UL}(n,m;k)$ 中的元素。

而对于任意一条自由路 $\tilde{P} \in \mathcal{F}^{UL}(n,m;k)$，我们同样将其按照谷点拆开。考虑其在对角线下方最远的点，其必然是谷点，设为 v_j。则我们把 $\tilde{P}$ 在 v_j 处断开为 L_1, L_2，设 $P = L_2L_1$。

则由引理 2.2 知，$P = L_2L_1 \in \mathcal{D}(n,m;k)$。因此每个 $\mathcal{F}^{UL}(n,m;k)$ 中的元素都可以唯一对应上一条 Dyck 路的一个峰。故有

$$D(n,m;k) = \frac{1}{k}F^{UL}(n,m;k) = \frac{1}{k}\binom{n-1}{k-1}\binom{m-1}{k-1}$$

□

2.5 t-Dyck 路计数

定义 2.7. *对于一条从 $(0,0)$ 到 (n,m) 的自由路，若其始终不经过对角线 $y = t\cdot x$ 下方，则我们称之为**t-Dyck 路**。特别地，若 $m = t\cdot n$，则我们称之为**n 阶 t-Dyck 路**。*

注意因为 $(n, t\cdot n) = n \neq 1$，所以 n 阶 t-Dyck 路并不满足上述定理的使用条件。

定理 2.6. *有 k 个峰的 n 阶 t-Dyck 路的个数是：*

$$D(n,tn;k) = \frac{1}{k}\binom{n-1}{k-1}\binom{tn}{k-1} \tag{9}$$

证明. 记 $m = t\cdot n + 1$，则显然 $(n,m) = 1$。代入式 8 得：

$$\begin{aligned} D(n,tn+1;k) &= \frac{1}{k}\binom{n-1}{k-1}\binom{tn+1-1}{k-1} \\ &= \frac{1}{k}\binom{n-1}{k-1}\binom{tn}{k-1} \end{aligned}$$

考虑 $\mathcal{D}(n,tn+1;k)$ 中的任一元素 P，因为 $\frac{tn+1}{n} > t \geq 2$，所以 P 的前两步必然都是上步 U。我们设其去掉第一个上步后得到的路为 P'。

因为在 $y = \frac{tn+1}{n}x$ 与 $y = tx$ 之间没有整点，所以去掉第一个 U 不会使 P' 到 $y = tx$ 下方。因此有 $P' \in \mathcal{D}(n,tn;k)$。

类似地，可以得到：$\forall P' \in \mathcal{D}(n,tn;k)$，有 $P = UP' \in \mathcal{D}(n,tn+1;k)$。因此，有 $\mathcal{D}(n,tn;k)$ 与 $\mathcal{D}(n,tn+1;k)$ 中的元素一一对应。故有：

$$D(n,tn;k) = D(n,tn+1;k) = \frac{1}{k}\binom{n-1}{k-1}\binom{tn}{k-1}$$

□

定理 2.7. *n 阶 t-Dyck 路的个数是：*

$$D(n,tn)=\frac{1}{tn+1}\binom{tn+n}{n} \tag{10}$$

证明.

$$\begin{aligned}D(n,tn)&=\sum_{k=1}^{n}D(n,tn;k)\\&=\sum_{k=1}^{n}\frac{1}{k}\binom{n-1}{k-1}\binom{tn}{k-1}\\&=\frac{1}{tn+1}\binom{tn+n}{n}\end{aligned}$$

□

事实上，从 $(0,0)$ 到 (n,m) 的恰有 k 个峰的 t-Dyck 路个数由 I.P. Goulden 和 L.G. Serrano 在参考文献 3 中给出：

定理 2.8. *从 $(0,0)$ 到 (n,m) 的有 k 个峰的 t-Dyck 路的个数是：*

$$D_t(n,m;k)=\frac{m+1-tn}{n}\binom{n}{k}\binom{m}{k-1} \tag{11}$$

定理 2.9. *从 $(0,0)$ 到 (n,m) 的 t-Dyck 路的个数是：*

$$D_t(n,m)=\frac{m-tn+1}{n+1}\binom{n+m}{n} \tag{12}$$

由于这两个定理的证明较为复杂，在此略过。有兴趣的读者可以自行查阅资料。

2.6 Dyck 路的另一种等价定义

n 阶 Dyck 路是在 x 轴上方以上步 $U_1=(1,1)$，下步 $D=(1,-1)$，从 $(0,0)$ 到 $(2n,0)$ 的格路。

n 阶 t-Dyck 路是在 x 轴上方以上步 $U_t=(1,t)$，下步 $D=(1,-1)$，从 $(0,0)$ 到 $((t+1)n,0)$ 的格路。

n 阶 t-Dyck 路的全体记做 $\tilde{\mathcal{D}}_t(n)$，$\tilde{D}_t(n)=\#\tilde{\mathcal{D}}_t(n)$。

将有 k 个峰的 t-Dyck 路的全体记为 $\tilde{\mathcal{D}}_t(n;k)$，$\tilde{D}_t(n;k)=\#\tilde{\mathcal{D}}_t(n;k)$。

$\tilde{\mathcal{D}}_t(n)$ 和 $\mathcal{D}(n,tn)$ 之间存在双射。构造双射的一个方法如下：

对于 P，我们将其倒序得到 P'，然后把 L 替换为 U_t，U 替换为 D 即可。例如：

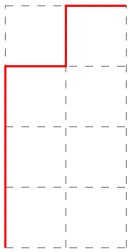


图 2: P

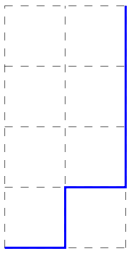


图 3: P'

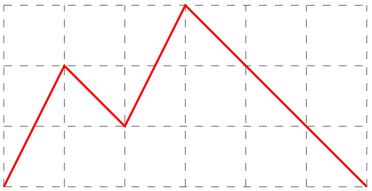


图 4: $\tilde{P}$

其中 P 是 $\mathcal{D}(2,4)$ 中的一条 $(2,4)$-Dyck 路 $UUULUL$。则令 $P' = LULUUU$，$\tilde{P} = U_tDU_tDDD$ 即可。

3 不相交格路

3.1 n 阶不交 Dyck 路计数

定义 3.1. *从 $(0,0)$ 到 (n,n) 的两条 Dyck 路 P、Q 称为一个n* **阶 *Dyck* 路对**。*若 Q 始终* **不穿过** *P，则 (P,Q) 是一个* **不交 *Dyck* 路对**，*否则称为* **相交 *Dyck* 路对**。

定理 3.1. *n 阶不交 Dyck 路对数为*

$$\begin{vmatrix} C_n & C_{n+1} \\ C_{n+1} & C_{n+2} \end{vmatrix} \tag{13}$$

其中 C_n 为卡特兰数的第 n 项。[24]

证明. 对于任意一个 n 阶 Dyck 路对 (P,Q)，构造映射 $(P,Q) \to (\tilde{P},\tilde{Q})$，其中 $\tilde{P} = UUPLL$，$\tilde{Q} = Q$：

则显然 (P,Q) 相交当且仅当 $(\tilde{P},\tilde{Q})$ 相交。因此，不交路对 (P,Q) 的个数即为 Dyck 路对 $(\tilde{P},\tilde{Q})$ 的个数去掉相交路对 $(\tilde{P},\tilde{Q})$ 的结果。显然，前者即为 $C_{n+2}\cdot C_n$，考虑如何计算后者。

对任意一个相交路对 $(\tilde{P},\tilde{Q})$，设第一个交点是 x，则 x 将 $\tilde{P}$ 和 $\tilde{Q}$ 分割为 2 条子路，记做 $\tilde{P}_1$，$\tilde{P}_2$ 以及 $\tilde{Q}_1$，$\tilde{Q}_2$。

令映射 $\theta(\tilde{P},\tilde{Q}) = (\tilde{P}_1\tilde{Q}_2, \tilde{Q}_1\tilde{P}_2)$，其中 $\tilde{P}_1\tilde{Q}_2$ 是从 $(0,0)$ 到 $(n+1,n+1)$ 的一条 $n+1$ 阶 Dyck 路，$\tilde{Q}_1\tilde{P}_2$ 是从 $(1,1)$ 到 $(n+2,n+2)$ 的一条 $n+1$ 阶 Dyck 路。

而因为在同一个 $(n+2)\times(n+2)$ 方格中，从 $(0,0)$ 到 $(n+1,n+1)$ 的一条 $n+1$ 阶 Dyck 路和从 $(1,1)$ 到 $(n+2,n+2)$ 的一条 $n+1$ 阶 Dyck 路必然有交点。我们只要从第一个交点处断开，然后做同样的操作即可得到 $\tilde{P}$ 以及 $\tilde{Q}$，所以我们可以得出 θ 是一个双射。

[24] C_n 的通项公式为：$C_n = \frac{1}{n+1}\binom{2n}{n}$。

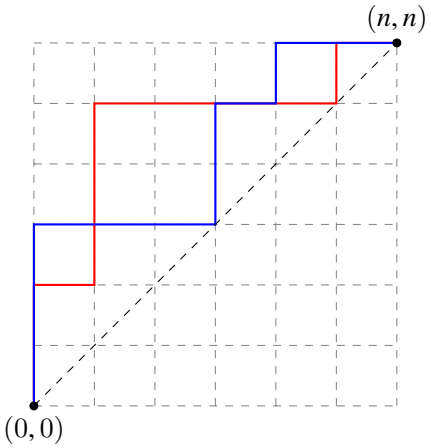


图 5: (P, Q)

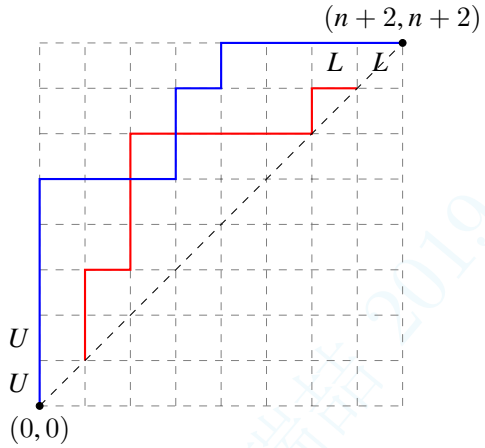


图 6: $(\tilde{P}, \tilde{Q})$

因此，相交路对 $(\tilde{P}, \tilde{Q})$ 的个数即为 $(\tilde{P}_1\tilde{Q}_2, \tilde{Q}_1\tilde{P}_2)$ 的个数，即 $C_{n+1} \cdot C_{n+1} = C_{n+1}^2$。故我们可以得到 n 阶不交 Dyck 路对个数即为：

$$C_{n+2} \cdot C_n - C_{n+1}^2 = \begin{vmatrix} C_n & C_{n+1} \\ C_{n+1} & C_{n+2} \end{vmatrix}$$

□

事实上，该定理由 M. DeSainte-Catherine 和 G. Viennot 在参考文献 4 中拓展到 k 条 n 阶不交 Dyck 路的情况：

定理 3.2. *k 条 n 阶不交 Dyck 路对数为*

$$\det(C_{n-i-j})_{i,j=1}^{k} = \begin{vmatrix} C_{n-2} & C_{n-3} & \cdots & C_{n-k} & C_{n-k-1} \\ C_{n-3} & C_{n-4} & \cdots & C_{n-k-1} & C_{n-k-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ C_{n-k-1} & C_{n-k-2} & \cdots & C_{n-2k+1} & C_{n-2k} \end{vmatrix} \tag{14}$$

3.2 不交自由路对计数

定义 3.2. *设 P、Q 是两条自由路，如果 Q 始终不穿越 P，则称 (P, Q) 是从 $(0, 0)$ 到 (n, m) 的两条**不交自由路对**，用 F_{nc} 表示（non-crossing）。如果 P、Q 除了起点以及终点之外没有公共点，则称 (P, Q) 是一个不接触自由路对，用 F_{nt} 表示（non-touching）。*

定理 3.3. *从 $(0, 0)$ 到 (n, m) 的不接触自由路对个数是*

$$F_{nt}(n, m) = \frac{1}{n}\binom{n+m-1}{n-1}\binom{n+m-2}{n-1} \tag{15}$$

证明. 对于任意一条不接触自由路对 (P,Q)，不妨设 P 在 Q 上方。则 P 第一步必然是 U，最后一步必然是 L。Q 则刚好相反：Q 第一步必然是 L，最后一步必然是 U。

去掉 P 和 Q 的首末 2 步，得到一组自由路对 $(\tilde{P},\tilde{Q})$（即 $P=U\tilde{P}L$，$Q=L\tilde{Q}U$）。根据定义，P,Q 是不含公共点的，因此 $\tilde{P},\tilde{Q}$ 是不交自由路对。显然不交自由路对 $(\tilde{P},\ \tilde{Q})$ 和不接触自由路对 (P,Q) 一一对应。

因为不交自由路对 $(\tilde{P},\ \tilde{Q})$ 的数量等于自由路对 $(\tilde{P},\ \tilde{Q})$ 的个数减去相交自由路对 $(\tilde{P},\ \tilde{Q})$ 的个数，而前者为 $F(n-1,m-1)^2=\binom{n+m-2}{n-1}^2$。考虑如何计算相交自由路对 $(\tilde{P},\ \tilde{Q})$ 的个数。

设 $\tilde{P}$ 与 $\tilde{Q}$ 的第一个相交点为 x，则 x 将 $\tilde{P}$ 和 $\tilde{Q}$ 分割为 2 条子路，记做 $\tilde{P}_1$，$\tilde{P}_2$ 以及 $\tilde{Q}_1$，$\tilde{Q}_2$。则格路 $\tilde{P}_1\tilde{Q}_2$ 为从 $(0,1)$ 到 $(n,m-1)$ 的自由路，格路 $\tilde{Q}_1\tilde{P}_2$ 为从 $(1,0)$ 到 $(n-1,m)$ 的自由路。

类似定理 3.1 的证明可以发现，映射 $\theta(\tilde{P},\tilde{Q})=(\tilde{P}_1\tilde{Q}_2,\tilde{Q}_1\tilde{P}_2)$ 为双射。所以相交自由路对 $(\tilde{P},\ \tilde{Q})$ 的个数等于自由路对 $(\tilde{P}_1\tilde{Q}_2,\ \tilde{Q}_1\tilde{P}_2)$ 的个数，即 $F_{touching}(\tilde{P}\tilde{Q})=F(n,m-2)\cdot F(n-2,m)=\binom{n+m-2}{n-2}^2$。因此，我们得到：

$$\begin{aligned}F_{nt}(n,m)&=\binom{n+m-2}{n-1}^2-\binom{n+m-2}{n}\binom{n+m-2}{n-2}\\&=\frac{1}{n}\binom{n+m-1}{n-1}\binom{n+m-2}{n-1}\end{aligned}$$

□

定理 3.4. 从 $(0,\ 0)$ 到 $(n,\ m)$ 的不交自由路对个数是

$$F_{nc}(n,m)=\frac{1}{n+1}\binom{n+m+1}{n}\binom{n+m}{n}\tag{16}$$

证明. 类似定理 3.3 的证明，我们考虑构建一个从 $(0,0)$ 到 $(n+1,m+1)$ 的不接触自由路对 $(\tilde{P},\tilde{Q})$：令 $\tilde{P}=UPL,\tilde{Q}=LQU$，可得到双射：$\theta^{-1}(\tilde{P},\tilde{Q})=(P,Q)$。因此：

$$\begin{aligned}F_{nc}(n,m)&=F_{nt}(n+1,m+1)\\&=\frac{1}{n+1}\binom{n+1+m+1-1}{n+1-1}\binom{n+1+m+1-2}{n-1}\\&=\frac{1}{n+1}\binom{n+m+1}{n}\binom{n+m}{n}\end{aligned}$$

□

3.3 不接触自由路对与 k 个峰的 Dyck 路的关系

我们可以发现，从 $(0,0)$ 到 $(k,n-k+1)$ 的不接触自由路对个数与 k 个峰的 n 阶 Dyck 路是一样的[25]，均为

$$\frac{1}{k}\binom{n}{k-1}\binom{n-1}{k-1}$$

事实上，我们可以证明这两者间存在双射：

证明. 对于任意一个从 $(0,0)$ 到 $(k,n-k+1)$ 的不接触自由路对 (P,Q)，将其展开为 UL 序列：

$$P = U\underbrace{U\cdots U}_{i_1}L\underbrace{U\cdots U}_{i_2}L\cdots L\underbrace{U\cdots U}_{i_k}L$$
$$Q = L\underbrace{U\cdots U}_{j_1}L\underbrace{U\cdots U}_{j_2}L\cdots L\underbrace{U\cdots U}_{j_k}U$$

则有：

$$\sum_{t=1}^{k} k^{i_t} = n-k$$
$$\sum_{t=1}^{k} j_t = n-k$$

令 $R=\rho(P,Q)=\underbrace{U\cdots U}_{i_1+1}\ \underbrace{L\cdots L}_{j_1+1}\underbrace{U\cdots U}_{i_2+1}\ \underbrace{L\cdots L}_{j_2+1}\cdots\underbrace{U\cdots U}_{i_k+1}\ \underbrace{L\cdots L}_{j_k+1}$，则 R 是一个有着 k 个峰的 n 阶格路。而因为 (P,Q) 是不接触自由路，因此：

$$i_1+1>j_1$$
$$i_1+i_2+1>j_1+j_2$$
$$\vdots$$
$$\sum_{t=1}^{k} i_t+1>\sum_{t=1}^{k} j_t$$
$$\sum_{t=1}^{k} i_t\geq\sum_{t=1}^{k} j_t$$

故 R 是 Dyck 路。一个通过 P,Q 构造 R 的例子如图所示：

[25]即为 Narayana 数 $N(n,k)$（见参考文献 10 中的序列 A001263）。

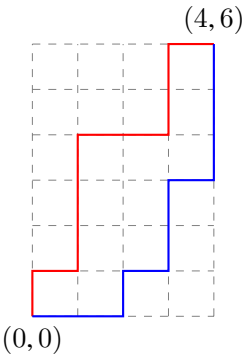


图 7: $P = ULUUULLUUL$, $Q = LLULUULUUU$

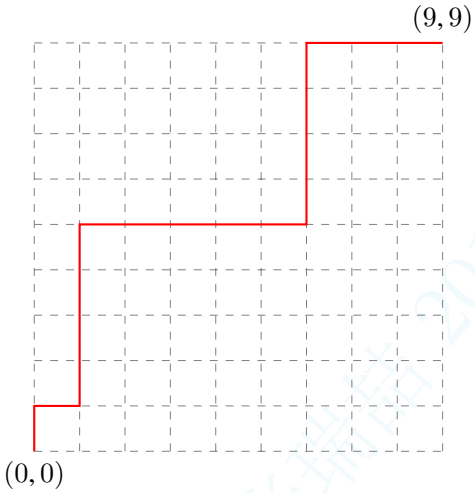


图 8: $\{i\} = \{0, 3, 0, 2\}$, $\{j\} = \{0, 1, 2, 2\}$, $R = ULUUUULLULLLUUULLL$

而 ρ 的逆映射的构造以及证明是类似的，因此 ρ 是一个双射。 □

4 实际应用

直方图是统计学中的一种常见工具，它可以显示统计对象波动的状态，从而较直观地传递有关过程统计对象状况的信息。而通过研究统计对象波动状况之后，就能掌握过程的状况，更好地进行改进。直方图的形态除了人工分析外，也可以转化为例如 Dyck 路的格路，从而提升分析效率。

4.1 构造双射

定义 4.1. 对于一个 *Dyck* 路 P，定义 P 中每一步的**高度**为这一步的两个的端点中，纵坐标的较大值。定义 P 的每一步的高度组成的序列为 P 的**高度序列**。

我们考虑构造从 Dyck 路到直方图的映射 φ。考虑 P 的高度序列中，每个极长的相同高度的连续段。设某个极长段的高度为 h，长度为 c，则我们把它映射到 $\lfloor \frac{c}{2} \rfloor$ 个高度为 h 的列。这样得到的序列即为 $\varphi(P)$ 的高度序列。

例如，对于如图所示的 Dyck 路径 $P = UDUDUUUDUDDUDUUDUDDUDDUDUUDD$，其高度序列为：11111233332222333322211112221。我们可以得到对应直方图 $\varphi(P)$ 的高度序列：113322332112。

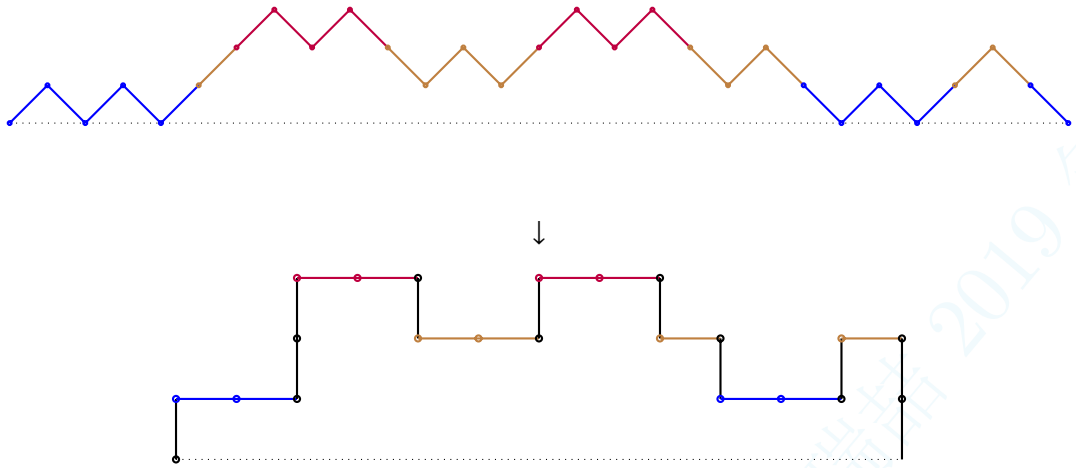


图 9: 一个 Dyck 路以及其对应的直方图

接下来，我们考虑该映射的逆映射 φ^{-1}。设直方图 B 的高度序列为 $h_0^{a_0}h_1^{a_1}\cdots h_{k+1}^{a_{k+1}}$，通过在合适的位置插入 $a_i=0$，我们可以假设 $h_0=h_{k+1}=0$，且 $\forall 0\le i\le k$，均有 $|h_i-h_{i+1}|=1$。对于 $i=1,2,\cdots,k$，我们把 $h_i^{a_i}$ 依次作如下替换，即可得到 Dyck 路 $\varphi^{-1}(B)$：

$$\begin{cases}(UD)^{a_i}U & ,\text{if } h_{i-1}<h_i\wedge h_i<h_{i+1}\\ (UD)^{a_i} & ,\text{if } h_{i-1}<h_i\wedge h_i>h_{i+1}\\ (DU)^{a_i} & ,\text{if } h_{i-1}>h_i\wedge h_i<h_{i+1}\\ (DU)^{a_i}D & ,\text{if } h_{i-1}>h_i\wedge h_i>h_{i+1}\end{cases}$$

例如，对于上面的例子中的直方图 $B=\varphi(P)$，其高度序列 $1133223321\,12$ 可以被改写为 $0^0 1^2 2^0 3^2 2^2 3^2 2^1 1^2 2^1 1^0 0^0$，从而得到对应的 Dyck 路：

$$\varphi^{-1}(B)=(UD)^2UU(UD)^2(DU)^2(UD)^2(DU)D(DU)^2(UD)D=P$$

因此 φ 是一个双射。

4.2 发掘性质

我们先定义一些直方图以及 Dyck 路上面的统计数据：

对于直方图 $B\in\mathcal{B}$，我们可以将其看作由**横步** $H=(1,0)$，**上步** $U=(0,1)$，以及**下步** $D=(0,-1)$ 组成的格路。记 $\#H(B),\#U(B),\#D(B)$ 分别表示 B 中 H,U,D 步的个数。定义 B 的**半周长（semiperimeter）**为上步与横步的个数和，记作 $\mathrm{sp}(B)=\#H(B)+\#U(B)$。定义 B 的**面积（area）**为 B 与 x 轴围成的封闭区域的面积，记作 $\mathrm{area}(B)$。

定义一个**峰（peak）**为 $UH^{\ell}D(\ell \geq 1)$ 的一次出现。记 $\mathrm{pk}(B)$ 表示 B 中峰的个数。类似地，定义一个**谷（valley）**为 $DH^{\ell}U(\ell \geq 1)$ 的一次出现，记 $\mathrm{val}(B)$ 表示 B 中谷的个数。因为峰和谷总是交替出现，且第一个和最后一个一定都是峰，所以 $\mathrm{pk}(B) = \mathrm{val}(B) + 1$。设 $\mathrm{shv}(B)$ 表示 B 中谷的高度之和，$\#H_1(B)$ 表示 B 中高度为 1 的谷的个数。

对于 Dyck 路 $P \in \mathcal{D}$，定义**峰（peak）**为 UD 的一次出现，**回归（return）**为一个使得 Dyck 路接触到 x 轴的下步。记 $\Lambda(P)$ 和 $\mathrm{ret}(P)$ 分别表示 Dyck 路 P 中，峰和回归的个数，设 $\mathrm{shp}(P)$ 表示 P 中峰的高度之和。

对于直方图 B 或 Dyck 路 P，定义 $\mathrm{ini}_U(B)$（或 $\mathrm{ini}_U(P)$）和 $\mathrm{fin}_D(B)$（或 $\mathrm{fin}_D(P)$）分别表示初始的上步 U 以及结束的下步 D 的个数。

例子 1. 对于图 4.1 中的直方图 B，我们有：$\#H(B) = 12$，$\#H_1(B) = 4$，$\#U(B) = 5$，$\#D(B) = 5$，$\mathrm{sp}(B) = \#H(B) + \#U(B) = 17$，$\mathrm{ini}_U(B) = 1$，$\mathrm{fin}_D(B) = 2$，$\mathrm{area}(B) = 24$，$\mathrm{pk}(B) = 3$，$\mathrm{shv}(B) = 3$。

我们可以得出如下定理：

定理 4.1. *对于 $P \in \mathcal{D}$ 以及 $B = \varphi(P) \in \mathcal{B}$，有以下结论成立：*

- $\mathrm{sl}(P) = \mathrm{sp}(B) - \mathrm{pk}(B)$；
- $\Lambda(P) = \#H(B) - \mathrm{val}(B)$；
- $\mathrm{shp}(P) = \mathrm{area}(B) - \mathrm{shv}(B)$；
- $\mathrm{ini}_U(P) = \mathrm{ini}_U(B)$；
- $\mathrm{ini}_D(P) = \mathrm{ini}_D(B)$；
- *若 P 的高度最大值为 1，则 $\mathrm{ret}(P) = \#H_1(B)$，否则 $\mathrm{ret}(P) = \#H_1(B) + 1$。*

这些结论的证明均不是非常复杂，由映射的定义稍加推导即可证明，故在此略去，有兴趣的读者可以自行推导。

下表整理了本节我们定义的记号以及它们间的关系：

直方图		Dyck 路		关系
$\mathrm{sp}(B)$	B 的半周长	$\mathrm{sl}(P)$	P 的半长	$\mathrm{sl}(P) = \mathrm{sp}(B) - \mathrm{pk}(B)$
$\mathrm{pk}(B)$	B 中峰的数量	$\Lambda(P)$	P 中峰的数量	$\Lambda(P) = \#H(B) - val(B)$
$\mathrm{val}(B)$	B 中谷的数量	$\mathrm{ret}(P)$	P 中回归的数量	$\mathrm{ret}(P) = \#H_1(B) + 1$（当 P 的高度不为 1 时）
$\mathrm{shv}(B)$	B 中谷的高度和	$\mathrm{shp}(P)$	P 中峰的高度和	$\mathrm{shp}(P) = \mathrm{area}(B) - \mathrm{shv}(B)$
$\mathrm{ini}_U(B)$	初始的上步 U 的个数	$\mathrm{ini}_U(P)$	初始的上步 U 的个数	$\mathrm{ini}_U(P) = \mathrm{ini}_U(B)$
$\mathrm{ini}_D(B)$	结束的下步 D 的个数	$\mathrm{ini}_D(P)$	结束的下步 D 的个数	$\mathrm{ini}_D(P) = \mathrm{ini}_D(B)$

5 感谢

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张瑞喆的指导。
感谢南京外国语学校李曙老师给予的支持与指导。
感谢帮助过我的老师、同学、朋友们。
感谢我的父母对我无微不至的关心与照顾。

参考文献

[1] Blanco S A, Petersen T K. Counting Dyck paths by area and rank[J]. Annals of Combinatorics, 2014, 18(2): 171-197.

[2] Chen W Y C, Pang S X M, Qu E X Y, et al. Pairs of noncrossing free Dyck paths and noncrossing partitions[J]. Discrete Mathematics, 2009, 309(9): 2834-2838.

[3] Goulden I P, Serrano L G. Maintaining the spirit of the reflection principle when the boundary has arbitrary integer slope[J]. Journal of Combinatorial Theory, Series A, 2003, 104(2): 317-326.

[4] DeSainte-Catherine M, Viennot G. Enumeration of certain Young tableaux with bounded height. InCombinatoire énumérative 1986 (pp. 58-67). Springer, Berlin, Heidelberg.

[5] Deutsch E, Elizalde S. A bijection between bargraphs and Dyck paths[J]. Discrete Applied Mathematics, 2018, 251: 340-344.

[6] 卢青林, 常海廷. 广义 Dyck 路径加性参数的计数 (英文)[J]. 徐州师范大学学报 (自然科学版), 2009 (1): 3.

[7] 孙学芝. 两类广义 Dyck 路上的一些计数问题 [D]. 华东师范大学, 2014.

[8] 孙怡东, 贾藏芝. 峰严格递增的 Dyck 路的计数 (英文)[J]. 数学研究与评论, 2007 (2): 6.

[9] 张晓光. 组合数学中的 K-路径问题研究 [D]. 大连理工大学, 2008.

[10] Sloane, N. J. A. (2002). On-Line Encyclopedia of Integer Sequences. http://oeis. org.

算法竞赛中一些数论问题的推广与高斯整数初探

西北工业大学附属中学 李佳衡

摘 要

本文将从一道校内 NOIP 模拟赛题出发，简要研究整数整除理论在数论结构上的某些特点，并尝试将整除理论移植到高斯整数上，进而将算法竞赛中一些常见的数论问题推广到高斯整数的范围内。

1 预备知识

为方便文章讨论与读者阅读，本节中将给出本文中用到的抽象代数中的一些概念及其简单的性质。已经具备相关知识的同学可以跳过对应的部分。

1.1 二元运算与代数系统

定义 1.1.1 (二元运算)**.** *设 X 为一给定集合，则称映射 $f : X^2 \to X$ 为集合 X 上的一个二元运算。即对于任意有序对 $(a,b) \in X^2$，存在一个确定的 $h \in X$ 与之对应，记作*

$$h = f(a,b)$$

或

$$h = a \circ b$$

定义 1.1.2 (交换律与结合律)**.** *对于定义在集合 X 上的二元运算 $\circ$，如果对任意的 $a,b \in X$ 都有*

$$a \circ b = b \circ a$$

则称这个二元运算是交换的。若对任意的 $a,b,c \in X$ 都有

$$(a \circ b) \circ c = a \circ (b \circ c) \tag{17}$$

则称这个二元运算是结合的。

定义 1.1.3. 设集合 $S \subset X$, $\circ$ 是 X 上的二元运算，且对任意的 $a,b \in S$ 都有

$$a \circ b \in S$$

则称集合 S 对运算 $\circ$ 封闭。

定义 1.1.4 (代数系统)**.** 在其上定义了若干二元运算 $\circ, \star, \cdots$ 的集合 X 称为一个代数系统。例如上述代数系统可记作 $(X, \circ, \star, \cdots)$。

1.2 半群与群

定义 1.2.1 (半群与交换半群)**.** 我们称代数系统 $(X, \circ)$ 是一个半群，当且仅当二元运算 $\circ$ 是结合的。特别地，如果半群中的二元运算 $\circ$ 是交换的，则称 $(X, \circ)$ 为交换半群。

定义 1.2.2 (半群的幺元素、幺半群)**.** 半群 $(X, \circ)$ 中的元素 e 如果满足：对任意的 $a \in X$，必有

$$a \circ e = e \circ a = a \tag{18}$$

则称 e 为半群 $(X, \circ)$ 的幺元素。有幺元素的半群称为幺半群。

定理 1.2.1. 半群中的幺元素要么不存在，要么有且仅有一个。

Proof.

不妨设 $e \neq e'$ 均为半群 $(X, \circ)$ 的幺元素，则由式 (18) 可知

$$e = e \circ e' = e'$$

这与 $e \neq e'$ 矛盾。因此幺元素如果存在必然唯一。 □

定义 1.2.3 (可逆元素)**.** 设幺半群 $(X, \circ)$ 的幺元素为 e，对于元素 $a \in X$，若存在 $b \in X$ 使得

$$a \circ b = b \circ a = e \tag{19}$$

则称 a 为可逆元素（或单位元素），并把 b 称作 a 的逆元素，记作 $b = a^{-1}$。

定理 1.2.2. 对于半群中的每一个可逆元素，其逆元素唯一。

Proof.

不妨设 $b \neq b'$ 同时为半群 $(X, \circ)$ 中元素 a 的逆元素，则由式 (17) (18) (19) 可知

$$b' = (b \circ a) \circ b' = b \circ (a \circ b') = b$$

这与 $b \neq b'$ 矛盾。因此逆元素如果存在必然唯一。 □

定义 1.2.4 (群与交换群)**.** 设 $(X, \circ)$ 是一个半群。如果它的每个元素都可逆，则称 $(X, \circ)$ 是群。特别地，如果二元运算 $\circ$ 是交换的，则称之为交换群。

1.3 环

定义 1.3.1 (环与交换环)**.** 设 X 是一个给定的集合，在其上定义了两种二元运算 $\oplus$ 和 $\odot$。代数系统 $(X,\oplus,\odot)$ 称为环，如果它满足以下条件：

(1) $(X,\oplus)$ 是一个交换群，通常称作环的加法群

(2) $(X,\odot)$ 是一个半群，通常称作环的乘法半群

(3) 对于任意的 $a,b,c\in X$，有

$$a\odot(b\oplus c)=(a\odot b)\oplus(a\odot c)$$
$$(b\oplus c)\odot a=(b\odot a)\oplus(c\odot a)$$

特别地，如果运算 $\odot$ 也是交换的，则称这个代数系统为交换环。

定义 1.3.2 (零元素)**.** 如果环 $(X,\oplus,\odot)$ 的加法群 $(X,\oplus)$ 含有幺元素 o，则称 o 为环 X 的零元素。

定义 1.3.3 (幺元素)**.** 如果环 $(X,\oplus,\odot)$ 的乘法半群 $(X,\odot)$ 含有幺元素 e，则称 e 为环 X 的幺元素。

定义 1.3.4 (单位元素)**.** 环 $(X,\oplus,\odot)$ 的乘法半群 $(X,\odot)$ 的单位元素 ϵ 称为环 X 的单位元素。

定义 1.3.5 (相伴数)**.** 设 ϵ 为环 $(X,\oplus,\odot)$ 的单位元素，对于 $a,b\in X$，若有 $a\odot\epsilon=b$，则称 a 和 b 互为相伴数，记作 $a\sim b$。

显然相伴关系是一个等价关系（即具有对称性、传递性和反身性）。

定义 1.3.6 (零因子)**.** 记环 $(X,\oplus,\odot)$ 的零元素为 o，再设 $o\neq a,b\in X$，若有 $a\odot b=o$ 成立，则称 a 是 X 的左零因子，b 是 X 的右零因子。显然在交换环中，左右零因子是一致的。

定义 1.3.7 (整环)**.** 设 X 是至少有两个元素的交换环，若 X 有幺元素，无零因子，则称 X 为整环。

2 一些记号与约定

在本文中，如无特殊说明，我们不对集合 M 和建立在 M 上的代数系统加以区分。其具体含义可以根据语境推知。

为了叙述简洁，在不引起歧义的情况下，用 o,e,ϵ 分别表示对应代数系统中的零元、幺元和单位元。

$\mathbb{Z}[i]$ 表示集合 $\{a+bi\mid a,b\in\mathbb{Z}\}$，其中 $i^2=-1$ 为虚数单位，并称 $\mathbb{Z}[i]$ 中的数为高斯整数。类似的记号还有 $\mathbb{Q}[i]$ 等。

3 问题引入

3.1 题目描述

给出两个非零数 $x,y \in \mathbb{Z}[i]$，你需要求出所有形如 $ax+by$（其中 $a,b \in \mathbb{Z}[i]$）的非零复数中模长最小的并将其输出。

保证 x,y 的实部与虚部的绝对值不超过 10^9。

3.2 题目讨论

该题来自于我校 2018 年 NOIP 模拟赛。

原题有一档 x,y 虚部均为 0 的部分分，而 $ax+by$ 的形式则让人很容易联想到 Bezout 定理。于是对于这一档部分分输出两个有理整数的最大公约数即可。

现在问题从一维的有理整数拓展到二维的 $\mathbb{Z}[i]$，一个很自然的想法是将两维分别讨论。很遗憾的是，这样做是错的。例如：对于 $x=1+2i,\ y=3+i$，如果两维分别取最大公约数得到的答案为 $1+i$；但注意到 $y=(1-i)x$，因此有

$$|ax+by| = \left|ax+(1-i)bx\right| \geq |x| = \sqrt{5}$$

显然 $|1+i| = \sqrt{2} < \sqrt{5}$，矛盾。

另外一个想法便是能否依照有理整数那样，在 $\mathbb{Z}[i]$ 上建立起一套“整除理论”，从而利用类似于 Bezout 定理的结论解决问题。

为此，我们需要先了解有理整数中的整除理论，并在充分了解其数论结构之后尝试将其推广。

4 有理整数 $\mathbb{Z}$ 中的整除理论

虽然本文的绝大部分读者已经了解整数的整除理论，但为了文章论述的完整性，本节将不加证明地列举整数整除理论的一些基本概念及其简单性质。

由定义易知 $(\mathbb{Z},+,\times)$ 是一个整环。为了与后文统一，本节中对一些概念的叙述将使用整环的语言。

定义 4.1 (整除)**.** *设 $a,b \in \mathbb{Z}$ 且 $a \neq 0$，若存在 $x \in \mathbb{Z}$ 使得 $b=ax$，则称 a 整除 b，记作 $a \mid b$。同时称 a 是 b 的约数，b 是 a 的倍数。*

对于 $0 \neq b \in \mathbb{Z}$，显然 ϵ 和 $\pm b$ 都是 b 的约数，称它们为 b 的显然约数，b 的其他约数称作非显然约数。

定义 4.2 (不可约数)**.** *设 $p \in \mathbb{Z},\ p \neq 0,\pm 1$，如果 p 没有非显然约数，就称 p 是 $\mathbb{Z}$ 中的不可约数。*

定义 4.3 (素数)**.** 若 $p \in \mathbb{Z}$ 满足对任意的 $a,b \in \mathbb{Z}$ 且 $p \mid ab$ 一定有 $p \mid a$ 或 $p \mid b$，则称 p 是 $\mathbb{Z}$ 中的素数。

众所周知，在 $\mathbb{Z}$ 中不可约数与素数是一致的，因此对它们不作区分[26]。在后文的叙述中，为避免混淆，将 $\mathbb{Z}$ 中的素数称为有理素数。

定理 4.1 (带余除法)**.** 设 $a,b \in \mathbb{Z}$ 且 $a \neq 0$，那么一定存在唯一的一对 $q,r \in \mathbb{Z}$，使得：

$$b = qa + r, \qquad 0 \leq r < |a|$$

定义 4.4 (公约数)**.** 设 $a,b \in \mathbb{Z}$，若 $d \in \mathbb{Z}$, $d \mid a$, $d \mid b$，就称 d 是 a,b 的公约数。

定义 4.5 (最大公约数)**.** 设 $a,b \in \mathbb{Z}$ 且不全为 0, a,b 的公因数中的最大者称为 a 和 b 的最大公因数，记作 $\gcd(a,b)$。

定理 4.2 (Bezout 定理)**.** 设 $a,b \in \mathbb{Z}$，则存在 $x,y \in \mathbb{Z}$ 使得 $ax + by = \gcd(a,b)$。

定理 4.3 (算术基本定理)**.** 设 $a \in \mathbb{Z}$, $a \neq 0, \pm 1$，那么 a 一定可以唯一地表示为：

$$a = \epsilon p_1^{m_1} \cdots p_s^{m_s}$$

其中 $p_1 < p_2 < \cdots < p_s$ 是不同的正有理素数。

由于相伴关系是一个等价关系，因此可以将 $\mathbb{Z}$ 中的全体元素按相伴关系划分成两两不相交的等价类，并在每个等价类中指定一个元素为代表，这样就得到 $\mathbb{Z}$ 的一个代表集合（如 $\mathbb{N}$）。由于相伴元素具有相同的整除性质，在讨论仅和整除有关的性质时，可以只在某个代表集合中考虑。

5　整除理论体系的数论结构

不难发现，第 4 节中提到的概念与性质大多建立在 $(\mathbb{Z},+,\times)$ 是一个整环的基础上，并没有用到整数集 $\mathbb{Z}$ 的特性（如离散性、大小关系等）。由此，设 $(M,\oplus,\odot)$ 是一个整环，我们可以将第 4 节中的概念进行推广。

为了方便，对于 $\alpha,\beta \in M$，我们用 $\alpha\beta$ 表示 $\alpha \odot \beta$。

定义 5.1 (整除)**.** 设 $\alpha,\beta \in M$ 且 $\alpha \neq o$，若存在 $\tau \in M$ 使得 $\beta = \alpha\tau$，则称 α 整除 β，记作 $\alpha \mid \beta$。同时称 α 是 β 的约数，β 是 α 的倍数。

对于 $o \neq \beta \in M$，显然 ϵ 和 $\epsilon\beta$ 都是 β 的约数，称它们为 β 的显然约数，β 的其他约数称作非显然约数。

[26] 但事实上，“素数”与“不可约数”在数论中的确是两个不同的概念。

定义 5.2 (不可约数)**.** 设 $\xi \in M, \xi \neq o, \epsilon$，如果 ξ 没有非显然约数，就称 ξ 是 M 中的不可约数。

定义 5.3 (素数)**.** 设 $\xi \in M$，如果对任意 $\alpha, \beta \in M$ 且 $\xi \mid \alpha\beta$，都有 $\xi \mid \alpha$ 或 $\xi \mid \beta$，则称 ξ 为素数。

定理 5.1. 所有的素数都是不可约数。

Proof.

假设素数 $\xi = \alpha\beta$ 不是不可约数，其中 $\alpha, \beta \neq \epsilon$，由素数的定义可知必有 $\xi \mid \alpha$ 或 $\xi \mid \beta$ 成立。

不妨 $\xi \mid \alpha$，由此及 $\alpha \mid \xi$ 可知 $\xi \sim \alpha$，进而有 $\beta = \epsilon$，由此产生矛盾，故所有素数都是不可约数。 □

定义 5.4 (公约数)**.** 设 $\alpha, \beta \in M$，若 $\delta \in M, \delta \mid \alpha, \delta \mid \beta$，就称 δ 是 α, β 的公约数。

我们在尝试推广最大公约数的概念的时候出现了问题：整环 M 的元素之间不一定定义有大小关系。因此我们需要用另一种方式来描述所谓“最大”公约数。

定义 5.5 (最大公约数)**.** 设 $\alpha, \beta \in M$ 且不全为 o，若存在 $\Delta \in M$ 满足 $\Delta \mid \alpha, \Delta \mid \beta$，且对于 α, β 的任一公约数 δ 都有 $\delta \mid \Delta$，就称 Δ 是 α, β 的最大公约数，记作 $\gcd(\alpha, \beta)$。

这里需要指出，如此定义并没有保证 Δ 的存在性 [27]，但是，可以证明，如果 Δ 存在，那么它必然在相伴意义下唯一。

定义 5.6 (可分解整环)**.** 如果对于任一 $\alpha \in M, \alpha \neq o, \epsilon$，$\alpha$ 一定可且只可表为 M 中有限个不可约数的乘积：

$$\alpha = \xi_1 \cdots \xi_r \tag{20}$$

则称 M 是可分解整环。其中“只可表为”指的是不存在这样的表法：对于任意给定的正整数 n, α 可表为 n 个非单位元素的乘积，且其中存在非不可约数。

定理 5.2. 如果对任一 $o \neq \alpha \in M$, α 的互不相伴约数只有有限个，那么 M 是可分解整环。

Proof.

设 $n(\alpha)$ 表示 α 的不相伴约数个数。显然当 $\alpha \neq 0, \epsilon$ 时总有 $n(\alpha) \geq 2$。

当 $n(\alpha) = 2$ 时，α 为不可约数，结论成立。

假设当 $2 \leq n(\alpha) < k$ 时结论都成立，那么当 $n(\alpha) = k$ 时，α 一定不是不可约数，即存在 $\alpha = \beta\gamma$ 且 $\beta, \gamma \neq \epsilon$，显然 $n(\alpha) > n(\beta), n(\gamma)$，由归纳假设知 β, γ 都可表为不可约数的乘积，因此 α 也可表为不可约数的乘积，结论成立。 □

对于可分解整环，有以下命题：

[27] 事实上，在一些整环中的确存在 Δ 不存在的情况。

命题 1. 设 $\alpha,\beta \in M$ 且不全为 o，则 $\Delta = \gcd(\alpha,\beta)$ 存在，且存在 $x,y \in M$ 使得 $\alpha x \oplus \beta y = \Delta$。

命题 2. M 中的不可约数都是素数。

命题 3. 分解式 *(20)* 在不计次序和相伴的意义下是唯一的，即设 M_0 是 M 的一个代表集合，α 一定可以唯一地表为（不计次序）：

$$\alpha = \epsilon \eta_1^{n_1} \cdots \eta_s^{n_s}$$

其中 $\eta_j \in M_0$ 是两两不同的不可约数。

定义 5.7 (唯一分解整环)**.** 我们称命题 *3* 成立的可分解整环为唯一分解整环。

定理 5.3. 若 M 中命题 *2* 成立，设 $\xi_1,\xi_2 \in M$ 是不同的素数，$\alpha \in M$ 且 $\xi_1 \mid \alpha$, $\xi_2 \mid \alpha$，则 $\xi_1\xi_2 \mid \alpha$。

Proof.

由 $\xi_1 \mid \alpha$，设 $\alpha = \xi_1\beta$，显然 $\xi_2 \nmid \xi_1$，由命题 2 得 $\xi_2 \mid \beta$，因此 $\xi_1\xi_2 \mid \alpha$。 □

定理 5.4. 命题 *1* ~ 命题 *3* 两两等价。

这个定理证明请查阅参考文献 [1] 的相关内容。

6 高斯整数 $\mathbb{Z}[i]$ 中的整除理论

显然 $(\mathbb{Z}[i],+,\times)$ 是一个整环，我们可以引入上一节的定义 5.1 至 5.5，在此不再赘述。

定义 6.1 (范数)**.** 设 $\alpha = a + bi \in \mathbb{Q}[i]$，我们把 $N(\alpha) = a^2 + b^2$ 称作 α 的范数。

定理 6.1. 设 $\alpha,\beta \in \mathbb{Q}[i]$，则 $N(\alpha\beta) = N(\alpha)N(\beta)$，若 $\alpha \mid \beta$ 则 $N(\alpha) \mid N(\beta)$。

定理 6.2. 设 $n \in \mathbb{N}$, $\mathbb{Z}[i]$ 中范数等于 n 的元素个数有限。

上面两个定理是显然的，因此在这里不做证明。由定理 6.2 及定理 5.2 可得：

定理 6.3. $\mathbb{Z}[i]$ 是可分解整环。

定理 6.4 (带余除法)**.** 设 $\alpha,\beta \in \mathbb{Z}[i]$ 且 $\alpha \neq 0$，那么一定存在 $\eta,\gamma \in \mathbb{Z}[i]$ 使得：

$$\beta = \eta\alpha + \gamma, \qquad 0 \le N(\gamma) < N(\alpha) \tag{21}$$

Proof.

设 $\tau = r + si \in \mathbb{Q}[i]$ 且 $\beta = \tau\alpha$，取 $u, v \in \mathbb{Z}$ 分别为离 r, s 最近的整数，则有 $|r-u| \le \frac{1}{2}$ 以及 $|s-v| \le \frac{1}{2}$。

令 $\eta = u + vi$，则

$$N(\tau - \alpha) = |r-u|^2 + |s-v|^2 \le \frac{1}{2}$$

又因为 $\gamma = (\tau - \eta)\alpha$，所以有

$$N(\gamma) = N(\tau - \eta)N(\alpha) \le \frac{1}{2}N(\alpha) < N(\alpha)$$

□

定理 6.5. $\mathbb{Z}[i]$ 中命题 *1* 成立，即对于不全为 0 的 $\alpha, \beta \in \mathbb{Z}[i]$，$\Delta = \gcd(\alpha, \beta)$ 一定存在，且存在 $x, y \in \mathbb{Z}[i]$ 使得

$$\alpha x + \beta y = \Delta$$

Proof.

设 $S = \{\alpha x + \beta y \mid x, y \in \mathbb{Z}[i]\}$，设其中范数最小的非零元素为 δ，由 $\delta \in S$ 可推出对于任一 $\eta \mid \alpha, \eta \mid \beta$ 都有 $\eta \mid \delta$。对于 S 中任一元素 v，由定理 6.4，将 v 表示为：

$$v = \eta\delta + \gamma, \qquad 0 \le N(\gamma) < N(\delta)$$

由 $N(\delta)$ 的最小性可推出 $N(\gamma) = 0$ 即 $\gamma = 0, \delta \mid \gamma$。由定义 $\delta = \gcd(\alpha, \beta)$, $\gcd(\alpha, \beta)$ 存在，且 $\delta \sim \Delta$，可得 $\Delta \in S$。 □

由此及定理 5.4 得：

定理 6.6. $\mathbb{Z}[i]$ 是唯一分解整环。

定理 6.7. 设 $n \in \mathbb{N}$, $\mathbb{Z}[i]$ 中范数等于 n 的个数

$$E(n) = 4\sum_{d \mid n} \chi(d)$$

其中 [28]

$$\chi(d) = \begin{cases} (-1)^{\frac{(d-1)}{2}}, & 2 \nmid d, \\ 0, & 2 \mid d. \end{cases}$$

定理 6.8. $\xi \in \mathbb{Z}[i]$ 是素数，当其仅当 ξ 满足以下条件之一：

(1) $N(\xi) = 2$;

[28] 对数论函数理论比较熟悉的同学可以看出这里的 χ 其实是模 4 意义下的 Dirichlet 非主特征，不过这并不在本文的讨论范围内。这里仅指出这个函数是一个完全积性函数，证明从略。

(2) $N(\xi)=p$，其中 $p\equiv 1 \pmod 4$ 是正有理素数；

(3) $\xi\sim p$，其中 $p\equiv 3 \pmod 4$ 是正有理素数。

Proof.

充分性：(1) 的情况可以直接验证；(2) 的情况可以由定理 6.1 和定义 5.2 推出；对于 (3) 的情况，不妨设 $\xi=\alpha\beta$ $(\alpha,\beta\neq\epsilon)$，则有 $p^2=N(\xi)=N(\alpha)N(\beta)$，由此及 $N(\alpha),N(\beta)>1$ 得 $N(\alpha)=N(\beta)=p$，根据定理 6.7 不存在范数为 p 的高斯整数，由此推出矛盾，故 ξ 为素数。

必要性：若 ξ 是素数，由定理 4.3 可知：

$$\xi\bar{\xi}=N(\xi)=q_1\cdots q_r$$

其中 q_j 是正有理素数。由 ξ 是素数可得，存在一个 q_j（不妨设为 q_1），使得 $\xi\mid q_1$，进而有 $N(\xi)\mid N(q_1)=q_1^2$，因此有 $N(\xi)=q_1$ 或 $N(\xi)=q_1^2$。

若 $N(\xi)=q_1$ 必然有 (1) 或 (2) 成立；若 $N(\xi)=q_1^2$，由此及 $\xi\mid q_1$ 推出 $\xi\sim q_1$，所以 q_1 是 $\mathbb{Z}[i]$ 中的素数，由此及定理 6.7 知 $q_1\equiv 3 \pmod 4$ 即 (3) 成立。 □

7 一些常用算法的推广

从上面的讨论可以看出，$\mathbb{Z}[i]$ 与 $\mathbb{Z}$ 在数论结构上有很多的相似之处，由此我们可以尝试将 $\mathbb{Z}$ 中的一些常用且仅与数论结构有关的算法推广到 $\mathbb{Z}[i]$ 中。

7.1 Euclid 算法

在 $\mathbb{Z}$ 中，对于两个不全为 0 的数 a,b，我们可以使用 Euclid 算法在 $O(\log|a|)$ 的时间内求出 a 和 b 的最大公约数。下面我们仿照 $\mathbb{Z}$ 中的 Euclid 算法，解决 $\mathbb{Z}[i]$ 中的最大公约数问题。

对于不全为 0 的 $\alpha,\beta\in\mathbb{Z}[i]$，由定理 6.4，我们可以将其写成：

$$\beta=\eta\alpha+\gamma,\qquad 0\le N(\gamma)\le\frac{1}{2}N(\alpha) \tag{22}$$

并记 $\gamma=\beta \bmod \alpha$。

引理 7.1.1. $\alpha,\beta\in\mathbb{Z}[i]$, $\alpha\neq 0$，有 $\gcd(\alpha,\beta)=\gcd(\beta \bmod \alpha,\alpha)$。

Proof.

我们将 α,β 写成式 22。设 $\Delta=\gcd(\alpha,\beta)$，则由 $\Delta\mid\alpha$, $\Delta\mid\beta$，以及 $\gamma=\beta-\eta\alpha$ 可知 $\Delta\mid\gamma$。

另一方面，对于任意 $\delta \mid \alpha$, $\delta \mid \gamma$，由于 $\beta = \eta\alpha + \gamma$，有 $\delta \mid \beta$，因此 $\delta \mid \Delta$。由定义有 $\gcd(\beta \bmod \alpha, \alpha) = \Delta = \gcd(\alpha, \beta)$。 □

于是我们仍可以按照 $\mathbb{Z}$ 中 Euclid 算法的过程，按照上式递归处理，边界仍为

$$\gcd(0, \beta) = \beta$$

由式 (22)，对 (α, β) 的每次递归后会使两数较小的范数减少至少一半，因此这个算法的时间复杂度为 $O(\log N(\alpha))$。

至此，由定理 6.5 及本小节内容，我们已经可以在 $O(\log N(x))$ 的时间内解决第 3 节中的问题。接下来，我们再尝试对一些其它常见算法进行推广。

7.2 标准分解

众所周知，任何一个有理整数 $n \neq 0$ 都可以表示为若干有理素数的乘积，同时有很多算法可以解决 $\mathbb{Z}$ 中的素因子分解，即标准分解问题。

本节中我们将从 $\mathbb{Z}$ 中最广为人知的试除法出发，寻找 $\mathbb{Z}[i]$ 中的标准分解算法。

记 $\alpha \in \mathbb{Z}[i]$，我们希望将其表示为

$$\alpha = \xi_1 \cdots \xi_r \tag{23}$$

的形式，其中 $\xi_1, \cdots, \xi_r$ 为高斯素数。

7.2.1 试除法

由式 (23) 及定理 6.1 可知

$$N(\alpha) = N(\xi_1) \cdots N(\xi_r)$$

进而可推知：α 的所有素因子中，范数超过 $\sqrt{N(\alpha)}$ 的至多有一个。

因此我们可以仿照 $\mathbb{Z}$ 中的做法，按照范数顺序依次用所有范数不超过 $\sqrt{N(\alpha)}$ 的高斯整数试除，最终剩下的结果要么为 ϵ，要么为高斯素数。

由于范数不超过 $\sqrt{N(\alpha)}$ 的高斯整数有 $O\left(\sqrt{N(\alpha)}\right)$ 个，因此这个算法的时间复杂度为 $O\left(\sqrt{N(\alpha)}\right)$。

7.2.2 算法改进

对于 $0 \neq \alpha = a + bi \in \mathbb{Z}[i]$，记 $d = \gcd(a, b)$，显然 $d \mid \alpha$，记

$$\alpha_1 = \frac{\alpha}{d} = a_1 + b_1 i$$

由 $\gcd(a_1, b_1) = 1$ 知 α_1 没有非显然的有理整数因子。

下面考虑分解 α_1，这需要我们寻找一个 α_1 的素因子 β。

记 $N(\alpha_1)$ 的标准分解为

$$N(\alpha_1)=\prod_{i=1}^{k}p_i^{r_i}$$

其中 $p_1<p_2<\cdots<p_k$ 为正有理素数。

当 $p_1=2$ 时，由 α_1 没有有理整数因子可知，取 $\beta=1+i$ 符合要求。

当 $p_1\equiv 1 \pmod 4$ 时，记 γ_1,γ_2 为满足 $N(\gamma_1)=N(\gamma_2)=p_1$ 的两个互不相伴的高斯整数，由定理 6.8 知 γ_1,γ_2 均为高斯素数。进而由 p_1 的最小性可知 $\beta=\gamma_1$ 或 $\beta=\gamma_2$ 中至少有一个满足条件。

当 $p_1\equiv 3 \pmod 4$ 时，不妨设 $\alpha_1=\beta\alpha_2$，于是有

$$p_1 \mid N(\alpha_1)=N(\beta)N(\alpha_2)$$

由此及 p_1 为有理素数知必然有 $p_1\mid N(\beta)$ 或 $p_1\mid N(\alpha_2)$。不妨设 $p_1\mid N(\beta)$，由定理 6.8 知 $\beta=p_1$，这与 α_1 无非显然的有理整数因子矛盾。

如上述构造出 β 后，记 $\alpha_1=\beta\alpha_2$，再对 α_2 重复上述操作直到只剩下 ϵ。

最后将 d 在 $\mathbb{Z}$ 中标准分解，并将所有模 4 不为 3 的素因子在 $\mathbb{Z}[i]$ 中分解即可。

现在问题转化为有理素数 $p\equiv 1 \pmod 4$ 在 $\mathbb{Z}[i]$ 中的分解。

引理 7.2.2.1. *设有理素数 $p\equiv 1 \pmod 4$，有理整数 $z^2\equiv -1 \pmod p$ 且 $0\le z<p$，记 $\xi=\gcd(p,z+i)$，则有 $\xi\overline{\xi}=p$。*

Proof.

原命题等价于证明 $N(\xi)=\xi\overline{\xi}=p$。

由已知，有 $\xi\mid p$ 以及 $\xi\mid z+i$，于是有

$$N(\xi)\mid \gcd(N(p),N(z+i))=p$$

由题设知存在 $x,y\in\mathbb{Z}$，使得 $p=x^2+y^2=(x+yi)(x-yi)$，由此可知 $\gcd(x,y)=\gcd(x,p)=1$ 以及

$$0\equiv x^2+y^2\equiv x^2-z^2y^2\equiv (x+yz)(x-yz) \pmod p$$

如果 $p\mid x-yz$，不妨设 $x=yz+kp\ (k\in\mathbb{Z})$，于是有

$$x+yi=(yz+kp)+yi=y(z+i)+kp$$

又注意到 $(x+yi)\mid p$ 以及 $\gcd(x+yi,y)=1$，故有 $(x+yi)\mid(z+i)$，进而 $(x+yi)\mid\xi$，于是 $N(\xi)=p$。

如果 $p\mid x+yz$，同理可知 $N(\xi)=p$。

□

对于每个需要分解的 p，我们可以利用随机数寻找[29]一个模 p 的二次非剩余 c，取 $z \equiv c^{\frac{p-1}{4}} \pmod{p}$ 即可满足 $z^2 \equiv -1 \pmod{p}$[30]，然后按照引理执行 Euclid 算法即可。

设被分解数为 α，其范数 $N(\alpha) = n$。在这个算法中，除去有理整数在 $\mathbb{Z}$ 中的分解外，只对其部分素因子执行了快速幂和 Euclid 算法，而素因子的个数为 $O(\log n)$，故这部分用时为 $O(\log^2 n)$。

本算法的时间复杂度瓶颈在于对有理整数在 $\mathbb{Z}$ 中的分解。若使用 Pollard's Rho 算法，则总时间复杂度为 $O\left(n^{\frac{1}{4}}\right)$。

8　数论函数求和问题在高斯整数 $\mathbb{Z}[i]$ 中的简要推广

在算法竞赛中，数论函数求和问题是一类重要的问题，下面我们仿照 $\mathbb{Z}$ 中的做法，将一些常见的数论函数求和算法推广至 $\mathbb{Z}[i]$ 中。

为了方便，我们定义这样的两个集合

$$\begin{aligned} D^+ &= \{a+bi \mid a \in \mathbb{N}^+, b \in \mathbb{N}\} \\ D &= D^+ \cup \{0\} \end{aligned}$$

显然 D 为 $\mathbb{Z}[i]$ 的一个代表集，D^+ 为 D 中非零元素组成的集合，以下所有求和号枚举的高斯整数均为 D 中的元素。再定义

$$\begin{aligned} T(n) &= \{\alpha \in D \mid N(\alpha) = n\} \\ S(n) &= \{\alpha \in D \mid 1 \le N(\alpha) \le n\} \end{aligned}$$

仿照数论函数的原有定义，在 $\mathbb{Z}[i]$ 中给出以下定义：

定义 8.1 (数论函数)**.** *我们称函数 $f : D^+ \to \mathbb{C}$ 为 $\mathbb{Z}[i]$ 中的数论函数。*

定义 8.2 (积性函数)**.** *设 f 为 $\mathbb{Z}[i]$ 中的数论函数，如果满足对任意 $\alpha,\beta \in D^+$ 且 $\gcd(\alpha,\beta) = 1$，都有*

$$f(\alpha\beta) = f(\alpha)f(\beta)$$

则称 f 为 $\mathbb{Z}[i]$ 中的积性函数。

定义 8.3 (完全积性函数)**.** *设 f 为 $\mathbb{Z}[i]$ 中的积性函数，如果满足对任意 $\alpha,\beta \in D^+$，都有*

$$f(\alpha\beta) = f(\alpha)f(\beta)$$

则称 f 为 $\mathbb{Z}[i]$ 中的完全积性函数。

[29] 由二次剩余的性质，随机到一个二次非剩余的期望次数是 2 次。

[30] 这一结论可以由二次剩余的 Euler 判别法和 Fermat 小定理推出，在此不做详细叙述。

定义 8.4 (Dirichlet 卷积)**.** *对于数论函数 f 和 g，定义其 Dirichlet 卷积*

$$(f*g)(\alpha)=\sum_{\delta|\alpha}f(\delta)g\left(\frac{\alpha}{\delta}\right)$$

特别地，当 $g(\alpha)=c$ 为常数函数时，这个卷积可以记作 $f*c$。

定理 8.1. *设 $\alpha\in D^+$，则对于 $\beta\in S(n)$, $\alpha\beta$ 遍历所有 $S(N(\alpha)n)$ 中所有 α 的倍数。*

8.1 线性筛法

在 $\mathbb{Z}$ 中，我们可以使用 Euler 筛法在 $O(n)$ 的时间内求出不超过 n 的所有正有理素数，以及积性函数在不超过 n 的所有正有理整数处的值。在 $\mathbb{Z}[i]$ 中，我们可以使用类似的方法，求出范数不超过 n 的所有高斯素数，以及 $\mathbb{Z}[i]$ 中积性函数再范数不超过 n 的高斯整数处的值。

$\mathbb{Z}$ 中 Euler 筛法的过程是这样的：从 2 到 n 枚举有理整数 i，如果 i 未被更新，则记录 i 为有理素数，再依次枚举不超过 i 的有理素数 p，更新 $i\times p$ 的答案，当 $p\mid i$ 时，更新答案后停止 p 的枚举。

在这个算法中，每个有理非素数 i 只会被作为 $\frac{i}{p}\times p$ 更新一次，其中 p 为 i 的最小素因子，因此这个算法的时间复杂度为 $O(n)$。

在 $\mathbb{Z}[i]$ 中，我们仿照上面的做法，我们首先需要对所有范数不超过 n 的高斯整数按照范数排序，再按照范数从 2 到 n 枚举高斯整数 α（范数相同的 α 顺序任意），如果 α 未被更新，则记录 α 为高斯素数，再依次枚举 α 及之前的高斯素数 ξ，更新 $\alpha\times\xi$ 的答案，当 $\xi\mid\alpha$ 时，更新答案后停止 ξ 的枚举。

同样地，在这个算法中，每个高斯非素数 α 只会被作为 $\frac{\alpha}{\xi}\times\xi$ 更新一次，其中 ξ 为 α 在排序中最靠前的素因子，由于范数不超过 n 的高斯整数有 $O(n)$ 个，该算法的复杂度为 $O(n)$。

8.2 一种基于 Dirichlet 卷积构造的快速求和法

在 $\mathbb{Z}$ 中，对于某些积性函数，可以通过构造 Dirichlet 卷积在低于线性的时间内计算其前缀和，这种方法在国内算法竞赛界被称为“杜教筛”。下面我们尝试将这种方法扩展到 $\mathbb{Z}[i]$ 中。

在 $\mathbb{Z}$ 中，该算法大致过程如下：设

$$F(n)=\sum_{i=1}^{n}f(i)$$

有

$$\sum_{i=1}^{n}(f*g)(i)=\sum_{i=1}^{n}g(i)F\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$$

变形得

$$g(1)F(n)=\sum_{i=1}^{n}(f*g)(i)-\sum_{i=2}^{n}g(i)F\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$$

如果可以构造 g 使得 $f*g$ 和 g 都易于求和，直接按照上式记忆化递归，可以得到时间复杂度为 $O\left(n^{\frac{3}{4}}\right)$ 的算法。如果设置一个阈值 B，先对 1 至 B 使用线性筛预处理，大于 B 的部分按照上式递归，时间复杂度为

$$\begin{aligned}&O(B)+\sum_{i=1}^{\left\lfloor\frac{n}{B}\right\rfloor}O\left(\sqrt{\frac{n}{i}}\right)\\=\;&O(B)+O\left(\int_1^{\frac{n}{B}}\sqrt{\frac{n}{x}}\mathrm{d}x\right)\\=\;&O\left(B+\frac{n}{\sqrt{B}}\right)\end{aligned}$$

取 $B=n^{\frac{2}{3}}$ 即可达到理论最优复杂度 $O\left(n^{\frac{2}{3}}\right)$

在 $\mathbb{Z}[i]$ 中，我们也可以使用类似的算法：设

$$F(n)=\sum_{\alpha\in S(n)}f(\alpha)$$

有

$$\sum_{\alpha\in S(n)}(f*g)(\alpha)=\sum_{i=1}^{n}\sum_{\alpha\in T(i)}g(\alpha)F\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$$

变形得

$$g(1)F(n)=\sum_{\alpha\in S(n)}(f*g)(\alpha)-\sum_{i=2}^{n}\sum_{\alpha\in T(i)}g(\alpha)F\left(\left\lfloor\frac{n}{i}\right\rfloor\right)$$

只要构造出合适的 g，按照与 $\mathbb{Z}$ 中相同的方法，即可得到复杂度同样为 $O\left(n^{\frac{2}{3}}\right)$ 的求和算法。

在 $\mathbb{Z}[i]$ 中，$\sum_{\alpha\in S(i)}(f*g)(\alpha)$ 和 $\sum_{\alpha\in T(i)}g(\alpha)$ 的计算可能需要高于 $O(1)$ 的时间，如果能在 $O\left(\sqrt{i}\right)$ 的时间内计算，则我们可以线性筛预处理 $i=1$ 至 B 的答案，大于 B 的答案使用 $O(\sqrt{i})$ 的做法预处理，仍可以达到 $O\left(n^{\frac{2}{3}}\right)$ 的总复杂度。

8.3 一种基于扩展 Eratosthenes 筛法的快速求和法

在 $\mathbb{Z}$ 中，还有一种适用范围较广的、基于扩展 Eratosthenes 筛法的积性函数求和算法，在国内算法竞赛界一般被称为“Min_25 筛”。

在以下讨论中，我们认为所有高斯整数是有序的，对于范数不同的高斯整数，范数小者在前；对于范数相同的高斯整数，实部小的在前。

仿照 $\mathbb{Z}$ 中的做法：

$$g_{n,m} = \sum_{\substack{2\le N(\alpha)\le n \\ \alpha \text{ 不含 } m \text{ 及之前的素因子}}} f(\alpha)$$

$$h_n = \sum_{\alpha \text{ 为 } n \text{ 及之前的素数}} f(\alpha)$$

则仍有：

$$g_{n,m} = \sum_{\substack{\xi\in S(\sqrt{n}) \\ \xi \text{ 为 } m \text{ 之后的素数} \\ N(\xi^e)\le n}} f(\xi^e)\left([e>1] + g_{\left\lfloor\frac{n}{N(\xi^e)}\right\rfloor,p}\right) + h_n - h_m$$

为了计算 h，我们还需要完全积性函数 $f'(\xi) = f(\xi)$ 当 ξ 为素数（也可以将 f 拆分为若干个完全积性函数 f' 之和，最后相加），设 $\eta_1,\cdots,\eta_r$ 依次为范数不超过 $\sqrt{n}$ 的素数

$$h'_{i,n} = \sum_{\substack{2\le N(\xi)\le n \\ \xi \text{ 为素数或没有 } \eta_i \text{ 及之前的素因子}}} f'(\xi)$$

则有

$$h'_{i,n} = h'_{i-1,n} - f'(\eta_i)\left(h'_{i-1,\left\lfloor\frac{n}{N(\eta_i)}\right\rfloor} - h'_{i-1,N(\xi_{i-1})}\right)$$

边界为

$$h'_{0,n} = \sum_{\alpha\in S(n)} f'(\alpha) - f'(1)$$

计算完 h' 后，即可由最终的 h' 得到要求的 h（范数相同的高斯素数处要进行特殊处理）。

由定理 6.8，高斯素数的范数总为有理素数或有理素数的平方，且每个有理素数对应不超过 $O(1)$ 个高斯素数，因此范数不超过 n 的高斯素数个数为 $O(\pi(n))$ 个，其中 $\pi(n)$ 表示不超过 n 的有理素数个数。因此，只要 h'_0 可以快速求得，就可以使用与 $\mathbb{Z}$ 中相同的方法，以相同的复杂度解决 $\mathbb{Z}[i]$ 中的问题，时间复杂度仍为 $O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$。

对于 h'_0 的初值，可能也需要如上一小节所说，用 $O\left(n^{\frac{2}{3}}\right)$ 的时间计算，不影响总时间复杂度。

9 一些常见数论函数与例题

在本节中，对于 $\alpha,\beta\in\mathbb{Z}[i]$，我们说的 $\gcd(\alpha,\beta)$ 指两者的最大公约数在 D 中的相伴数。

首先，我们仿照 $\mathbb{Z}$ 中的定义，定义一些常用数论函数：

单位函数 $\epsilon(\alpha) = [\alpha = 1]$，即当 $\alpha = 1$ 时 $\epsilon(\alpha) = 1$，否则 $\epsilon(\alpha) = 0$。

幂函数 $\mathrm{Id}_k(\alpha) = \alpha^k$。特别地，当 $k = 1$ 时亦记作 $\mathrm{Id}(\alpha)$。

Mobius 函数，设 $\alpha=\prod_{i=1}^{r}\xi_i^{k_i}$，其中 $\xi_1,\cdots,\xi_r$ 为不同的素数，则

$$\mu(\alpha)=\begin{cases}0 & ,\exists k_i>1\\ (-1)^r & ,\text{otherwise}\end{cases}$$

与 $\mathbb{Z}$ 中类似，仍有性质：

定理 9.1. 设 $\alpha\in D^+$，则

$$\sum_{\delta|\alpha}\mu(\delta)=\epsilon(\alpha)$$

定义 9.1 (同余)**.** 设 $\alpha,\beta,\gamma\in\mathbb{Z}[i]$ 且 $\gamma\neq 0$, α 与 β 同余当且仅当 $\gamma\mid(\alpha-\beta)$，记作 $\alpha\equiv\beta\pmod{\gamma}$。

定义 9.2 (剩余类)**.** 设 $0\neq\gamma\in\mathbb{Z}[i]$，我们可以将 $\mathbb{Z}[i]$ 按照模 γ 是否同余分为两两不相交的等价类，这些等价类称为模 γ 的剩余类，记这些剩余类的个数为 $R(\gamma)$。

定义 9.3 (完全剩余系)**.** 设 $0\neq\gamma\in\mathbb{Z}[i]$，对模 γ 的每一个剩余类取定一个元素作为代表，这样得到的 $R(\gamma)$ 个元素称为模 γ 的一个完全剩余系。

定理 9.2. 设 $\gamma=a+bi\in\mathbb{Z}[i]$ $(a^2+b^2\neq 0)$，记 $g=\gcd(a,b)$，则

$$x_{m,n}=m+ni,\qquad 0\le m<\frac{N(\gamma)}{g},0\le n<g$$

是模 γ 的一个完全剩余系。

Proof.

先证明 $x_{m,n}$ 两两不同余。

假设 $x_{m,n}\equiv x_{m',n'}\pmod{\gamma}$，则设

$$\eta\gamma=x_{m,n}-x_{m',n'}=(m-m')+(n-n')i$$

由 $g\mid\gamma$ 可知 $g\mid(m-m)+(n-n')i$，进而有 $g\mid(n-n')$，由 n 的范围知 $n=n'$。

再设 $\eta=c-di$，则 $\eta\gamma=(ac+bd)+(bc-ad)i$，因此

$$\frac{a}{g}\cdot d=\frac{b}{g}\cdot c,\qquad \gcd\left(\frac{a}{g},\frac{b}{g}\right)=1$$

由整数的性质有 $\frac{a}{g}\mid c$，因此有 $\frac{cg}{a}=\frac{dg}{b}=k\in\mathbb{Z}$，于是

$$m-m'=ac+bd=k\cdot\frac{a^2+b^2}{g}$$

即 $\frac{N(\gamma)}{g}\mid(m-m')$，由 m 的范围知 $m=m'$。因此有 $x_{m,n}=x_{m',n'}$。

再证明对于任意的 $\alpha=x+yi\in\mathbb{Z}[i]$，都存在 $x_{m,n}$ 与 α 同余。

由带余除法，存在 $q_2 \in \mathbb{Z}, 0 \le n < g$，使得

$$y = q_2 g + n$$

以及 $q_1 \in \mathbb{Z}, 0 \le m < \frac{N(\gamma)}{g}$，使得

$$x - q_2(ax_0 - by_0) = q_1 \cdot \frac{N(\gamma)}{g} + m$$

其中 (x_0, y_0) 是 $ay_0 + bx_0 = g$ 的任意一组整数解。于是有

$$\alpha = m + ni + q_1 \cdot \frac{N(\gamma)}{g} + q_2\theta$$

其中 $\theta = (ax_0 - by_0) + gi$。

又由于

$$\begin{aligned}\theta &= (ax_0 - by_0) + gi = (ax_0 - by_0) + (ay_0 + bx_0)i \\ &= (a + bi)(x_0 + y_0 i)\end{aligned}$$

于是有 $\alpha \equiv m + ni \pmod{\gamma}$。

□

由定理 9.2 我们可以得到 $R(\alpha) = N(\alpha)$。

定义 9.4 (既约剩余系)**.** *设 $0 \neq \gamma \in \mathbb{Z}[i]$，模 γ 的一个完全剩余系中所有满足 $\gcd(\alpha, \gamma) = 1$ 的元素 α 组成模 γ 的一个既约剩余系。*

我们定义 Euler 函数 $\varphi(\gamma)$ 表示模 γ 的既约剩余系大小，与 $\mathbb{Z}$ 中类似，仍有性质：

定理 9.3. *设 $\alpha \in D^+$，则*

$$\sum_{\delta|\alpha} \varphi(\delta) = N(\alpha)$$

Proof.

考虑 α 的一个完全剩余系中的每个数 β，取 $\Delta = \gcd(\alpha, \beta)$，则有 $\gcd\left(\frac{\beta}{\Delta}, \frac{\alpha}{\Delta}\right) = 1$，考虑 α' 和 β' 分别为 $\frac{\alpha}{\Delta}$ 和 $\frac{\beta}{\Delta}$ 在 D 中的相伴元素，则所有这样的 β' 组成了 α' 的一个既约剩余系，因此 α 所有因数既约剩余系的大小之和与 α 的完全剩余系大小相等。 □

定理 9.4. *设 $\alpha \in D^+$ 且*

$$\alpha = \prod_{i=1}^{r} \xi_i^{k_i}$$

其中 $\xi_1, \cdots, \xi_r$ 为互不相同的高斯素数，则有

$$\varphi(\alpha) = \prod_{i=1}^{r} N(\xi_i^{k_i}) \left(1 - \frac{1}{N(\xi_i)}\right)$$

Proof.

由定理 9.3 得 $\varphi * 1 = N$，因此 $\varphi = N * \mu$ 为积性函数，对于 $\beta = \eta^c$，其中 η 为高斯素数，有：

$$\varphi(\beta) = \varphi(\eta^c) = N(\eta^c) - N(\eta^{c-1}) = N(\eta^c)\left(1 - \frac{1}{N(\eta)}\right)$$

由积性函数性质知定理成立。 □

以下例题均改编自有理整数数论中的常见问题。

例 1. 给定正整数 k，每次询问给定正整数 n, m，求

$$\sum_{\alpha \in S(n)} \sum_{\beta \in S(m)} \gcd(\alpha, \beta)^k$$

不超过 2000 次询问，$n, m, k \le 5 \times 10^5$，答案对 $10^9 + 7$ 取模。

$$\begin{aligned}
& \sum_{\alpha \in S(n)} \sum_{\beta \in S(m)} \gcd(\alpha, \beta)^k \\
= & \sum_{\alpha \in S(n)} \sum_{\beta \in S(m)} \sum_{\substack{\delta \mid \alpha \\ \delta \mid \beta}} \epsilon\left(\gcd\left(\frac{\alpha}{\delta}, \frac{\beta}{\delta}\right)\right) \delta^k \\
= & \sum_{\alpha \in S(n)} \sum_{\beta \in S(m)} \sum_{\substack{\delta \mid \alpha \\ \delta \mid \beta}} \delta^k \sum_{\tau \mid \delta} \mu(\tau) \\
= & \sum_{\Delta \in S(n)} C\left(\left\lfloor \frac{n}{N(\Delta)} \right\rfloor\right) C\left(\left\lfloor \frac{m}{N(\Delta)} \right\rfloor\right) \sum_{\delta \mid \Delta} \delta^k \mu\left(\frac{\Delta}{\delta}\right) \\
= & \sum_{i=1}^{n} C\left(\left\lfloor \frac{n}{i} \right\rfloor\right) C\left(\left\lfloor \frac{m}{i} \right\rfloor\right) \sum_{\Delta \in T(i)} \sum_{\delta \mid \Delta} \delta^k \mu\left(\frac{\Delta}{\delta}\right)
\end{aligned}$$

其中 $C(m)$ 表示 D^+ 中范数不超过 m 的元素个数。

C 和式子后半部分函数的值可以用 $O((n+m)\log k)$ 的时间预处理，对于每组数据枚举 $\left\lfloor \frac{n}{i} \right\rfloor$ 和 $\left\lfloor \frac{m}{i} \right\rfloor$ 的值计算即可，每组数据时间复杂度为 $O\left(\sqrt{n} + \sqrt{m}\right)$。

例 2. 每次询问给定正整数 n，求

$$\sum_{\alpha \in S(n)} \mu(\alpha) \text{ 和 } \sum_{\alpha \in S(n)} \varphi(\alpha)$$

不超过 10 次询问，$n \le 2^{32} - 1$。

我们使用构造 Dirichlet 卷积的方法求和：由上面的定理，我们有 $\mu * 1 = \epsilon$ 和 $\varphi * 1 = N$，其中 ϵ 的前缀和始终为 1，可以快速计算；计算常数函数和范数 N 的前缀和时，对于 $x^2 + y^2 \le m$，我们可以用 $O(\sqrt{m})$ 的时间枚举 x，然后对于每个 x，累加 1 至 $\left\lfloor \sqrt{m - x^2} \right\rfloor$，1 和 $x^2 + y^2$ 的前缀和即可，根据上面的方法，每次询问总复杂度为 $O\left(n^{\frac{2}{3}}\right)$。

例 3. 给定正整数 n，求

$$\sum_{\alpha\in S(n)}\sum_{\beta\in S(n)}\frac{\alpha\beta}{\gcd(\alpha,\beta)}$$

$n\le 10^{10}$。

$$\begin{aligned}
&\sum_{\alpha\in S(n)}\sum_{\beta\in S(n)}\frac{\alpha\beta}{\gcd(\alpha,\beta)}\\
=\ &\sum_{\Delta\in S(n)}\Delta^2F\left(\frac{n}{N(\Delta)}\right)\sum_{\delta|\Delta}\frac{1}{\delta}\mu\left(\frac{\Delta}{\delta}\right)\\
=\ &\sum_{i=1}^{n}F\left(\left\lfloor\frac{n}{i}\right\rfloor\right)^2\sum_{\Delta\in T(i)}\Delta^2\sum_{\delta|\Delta}\frac{1}{\delta}\mu\left(\frac{\Delta}{\delta}\right)
\end{aligned}$$

其中 $F(m)$ 表示 D^+ 中范数不超过 m 的所有高斯整数之和，可以在 $O\left(n^{\frac{2}{3}}\right)$ 时间内预处理，如果设 $(u\cdot v)(\alpha)=u(\alpha)v(\alpha)$，则后面的函数为 $f=\mathrm{Id}_2\cdot(\mathrm{Id}_{-1}*\mu)$，可以构造 Dirichlet 卷积 $f*\mathrm{Id}_2=\mathrm{Id}$ 计算，总时间复杂度为 $O\left(n^{\frac{2}{3}}\right)$。

10 总结

本文通过分析 $\mathbb{Z}$ 与 $\mathbb{Z}[i]$ 的结构，利用它们相似的性质，将 Euclid 算法、标准分解以及一些常用的数论函数求和方法推广至高斯整数中。实际上，仅与数论结构有关的算法大都可以如此推广，同时还有很多代数系统都可以进行类似的推广，感兴趣的读者可以自行尝试。

感谢

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张瑞喆的帮助和指导。
感谢本校刘利丽老师和各位学长的帮助和指导。
感谢本校王成瑞学长为撰写本文提供的帮助。
感谢本校刘承奥、和嘉晅、杜俊辰、王泽宇学长为本文验稿。

参考文献

[1] 潘承洞, 潘承彪, 《代数数论（第二版）》, 哈尔滨工业大学出版社。

[2] 潘承洞, 潘承彪, 《初等数论（第三版）》, 北京大学出版社。

[3] 任之洲, 《积性函数求和的几种方法》, 2016 年信息学奥林匹克中国国家队候选队员论文集。

[4] 朱震霆, 《一些特殊的数论函数求和问题》, IOI2018 中国国家候选队论文集。

[5] https://math.stackexchange.com/questions/5877

《基础圆方树练习题》命题报告

杭州学军中学 范致远

摘 要

本文介绍了作者在此次集训队互测中命制的一道传统数据结构题。

本题需要选手对动态树问题有比较深刻的了解，同时又需要一定的代码能力，是一道考察选手数据结构水平的好题。

本文提出了一种新的数据结构—ABF 树，相较于前人的结果，可以在一些问题上得到更低的时间复杂度。

1 题目大意

如果一个无向连通图的任意一条边最多属于 k 个不同的简单环，我们就称之为 k-仙人掌。

如果一个无向图的每个连通块都是个 k-仙人掌，且不存在自环，我们就称之为 k-沙漠。

要求维护一个 k-沙漠，支持加边、删边、对某两点之间最短路或某个子仙人掌进行修改或询问。

根为 x 时，子仙人掌 y 的定义是，删掉 x 到 y 的所有简单路径上的边后，y 所在的连通块。

2 数据范围

n 为结点数，m 为操作数。

对于所有数据：$1 \le n \le 50000$；$1 \le m \le 250000$。

特殊性质 A：加边与删边在其他操作之前。

特殊性质 B：没有子仙人掌询问、最短路加与子仙人掌加。

特殊性质 C：没有子仙人掌询问与子仙人掌加。

分数	测试集编号	k 的范围	特殊性质
6	1	$k=0$	AB
6	2		AC
6	3		A
5	4		B
5	5		C
5	6		
6	7	$k=1$	AB
6	8		AC
6	9		A
5	10		B
5	11		C
5	12		
6	13	$k=8$	AB
6	14		AC
7	15		A
5	16		B
5	17		C
5	18		

3 部分分算法

通过选取不同的部分分算法加以结合，可以得到多种不同的部分分。由于篇幅限制，这里只讨论少数高分做法。

3.1 算法一

对于 $k=0$ 的测试集，即，图一直是一个森林。这个子问题可以使用 Self-adjusting top trees 做。

具体细节可以参考 [1]，这里不再赘述。

时间复杂度为 $O(m\log n)$。

期望得分 33 分。

3.2 算法二

对于 $k = 1$，且满足特殊性质 C 的测试集。这个子问题与“动态仙人掌 III”[31] 一致，使用该题的做法可以解决此子问题。

具体细节可以参考《动态仙人掌 III》的题解 [32]，这里不再赘述。

时间复杂度为 $O(m \log n)$。

期望得分 36 分。

3.3 算法三

对于 $k = 1$，且满足特殊性质 A 的测试集。这个子问题“【清华集训 2015】静态仙人掌”[33] 相似，可以使用链剖圆方树解决。

关于圆方树的具体细节，可以参考 [2]，这里不再赘述。

时间复杂度为 $O(m \log n)$。

期望得分 44 分。

3.4 算法四

对于 $k = 1$ 的测试集。这个子问题与“动态仙人掌 IV”[34] 一致，可以使用 top cactus 解决。

具体细节可以参考 [3]，这里不再赘述。

时间复杂度为 $O(m \log^2 n)$。

期望得分 66 分。

4 标准算法

显然，这些结构都不足以维护 k-仙人掌，为了解决原问题，我们需要引入一个新结构维护 k-仙人掌。

由于这个结构是使用 AAA 树维护圆方树的产物，我们把这个数据结构称作”ABF 树”。

4.1 AAA 树

在展示 ABF 树的结构之前，先来探讨一下 AAA 树：

[31] http://uoj.ac/problem/65

[32] https://blog.csdn.net/VFleaKing/article/details/80747834

[33] http://uoj.ac/problem/158

[34] http://uoj.ac/problem/106

注：后文中的 AAA 树与 [4] 中的稍有不同，这里的 AAA 树可能是 SATT 的真 · 魔改版，结点的命名也参考了 SATT。

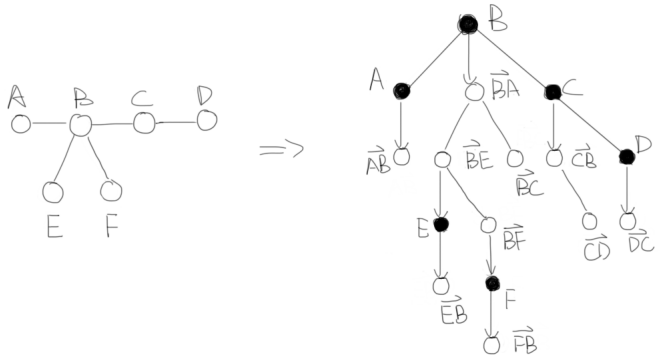
4.1.1 结构

AAA 树是一种基于 LCT 的数据结构，为了维护子树信息，在每个结点上额外加一棵辅助树维护其**所有**的出边。

AAA 树有两种结点：compress 与 rake。每个 compress 结点表示原树中的一个结点，每个 rake 结点表示原树中的一条**有向**出边（即，每条边会由两个 rake 结点维护）。所有的原子信息（即，不可拆分的信息本身）都放在 compress 结点上。

每个 compress 结点都有三个指针，分别为 l、r 与 c。l 与 r 指向了其在 compress 树中的孩子，c 指向了维护其所有出边的 rake 树。

每个 rake 结点也有三个指针，分别为 l、r 与 c。l 与 r 指向了其在 rake 树中的孩子，c 指向了维护以该出边为根的实链的 compress 树，特殊的，如果这条出边指向了该结点的父亲或是实孩子，则 c 为空。



如图，其中黑点为 compress 结点，白点为 rake 结点，向左的边为 l 指针，向右的边为 r 指针，带箭头的边为 c 指针。

为了方便维护，每个 compress 结点都有两个指针 p 与 s，分别表示该点的父边、实孩子边在 rake 树中是哪个结点。例如，B 结点的 p、s 指针分别指向 BA、BC。这两个指针仅起标记作用，不参与信息的处理。

另外，这个结构本身并不能维护边上的信息，当边上有信息时，需要在边上建辅助点。

4.1.2 维护

基于链剖分的动态树有两个核心操作：

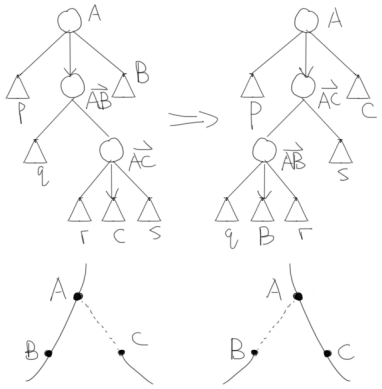
翻转（reverse）

翻转一棵 compress 树这个操作会在换根时用到。依然使用懒标记进行维护，当下传翻转标记时，不仅需要交换 l、r 指针，还需要交换 p、s 这两个指针。

切换虚实（splice）

切换孩子的虚实这个操作会在 expose 操作提取链时用到。

当要把结点 A 的实孩子由结点 B 切换为结点 C 时，先将结点 AB 与结点 AC 上旋至顶，然后把结点 C 移出来：



显然，AAA 树的 expose 操作与 LCT 中的 expose 操作除 splice 操作本身外完全一致，这里不再赘述。

考虑如何支持一些其他的基本操作：

上传信息

对于 compress 结点，信息分为两种：在链上的结点的信息和 $info_r$，在子树中但不在链上的结点的信息和 $info_v$。

$info_r$ 为 l 与 r 的 $info_r$，自身的信息之和。

$info_v$ 为 l、r 及 c 的 $info_v$ 之和。

对于 rake 结点，信息只有一种：在子树中的结点的信息和 $info_v$。

$info_v$ 为 l 与 r 的 $info_v$，c 的 $info_r$ 及 $info_v$ 之和。

下传标记

对于 compress 结点，标记分为两种：对链上的结点的修改 tag_r，对在子树中但不在链上的结点的修改 tag_v。

tag_r 会向 l 与 r 的 tag_r 下传，并影响其自身的原子信息。

tag_v 会向 l、r 及 c 的 tag_v 下传。

对于 rake 结点，标记只有一种：对子树中结点的修改 tag_v。

tag_v 会向 l 与 r 的 tag_v，c 的 tag_r 及 tag_v 下传。

提取链

当要提取结点 A 与结点 B 之间的链时，先对结点 A 进行 expose 操作后翻转结点 A，再对结点 B 进行 expose 操作。这时，以结点 A 为根的 compress 树对应了结点 A 与结点 B 之间的最短路。可以直接对其询问，或使用标记对其进行修改。

提取子树

当要提取以结点 A 为根的子树 B 时，先对结点 A 进行 expose 操作后翻转结点 A，再对结点 B 进行 expose 操作。这时，结点 B 及以结点 $B.c$ 为根的子树对应了以结点 A 为根的子树 B。可以直接对其询问，或使用标记对其进行修改。

加边

当要在结点 A 与结点 B 之间加一条边时，先对结点 A 进行 expose 操作后翻转结点 A，再对结点 B 进行 expose 操作，将结点 B 的 l 指针指向结点 A。在结点 A 的 rake 树中插入结点 AB 并将结点 A 的 s 指针指向它，在结点 B 的 rake 树中插入结点 BA 并将结点 B 的 p 指针指向它。

删边

当要删除结点 A 与结点 B 之间的边时，先对结点 A 进行 expose 操作后翻转结点 B，再对结点 B 进行 expose 操作，清空结点 B 的 l 指针。删去结点 A 的 rake 树中的结点 AB 并清空结点 A 的 s 指针，删去结点 B 的 rake 树中的结点 BA 并清空结点 B 的 p 指针。

实现上述所有操作，就可以得到一棵 AAA 树了。

4.1.3 分析

由于 AAA 树的所有操作都是基于 expose 操作的，expose 操作是基于 splice 操作的。容易证明 AAA 树单次操作时间复杂度的一个上界—均摊 $O(\log^2 n)$：最多 $O(\log n)$ 次 splice 操作，每次 splice 操作的时间复杂度为均摊 $O(\log n)$。

为了得到更紧的上界，需要进一步发掘 splice 操作的性质：

套用 LCT 的分析，依然以子树大小的对数和作为势能函数。

为了方便描述，令 $|x|$ 为结点 x 在 splice 操作前的子树大小，而 $|x'|$ 为结点 x 在 splice 操作后的子树大小。

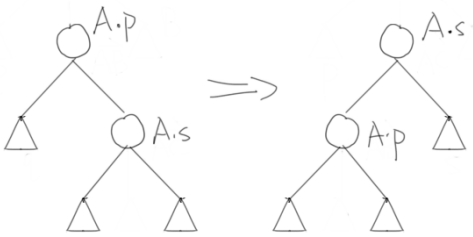
引理 1. *AAA 树中 splice 操作的摊还代价小于等于* $3(\log(|A'|) - \log(|C|)) + O(1)$。

证明. 将 C 移出来这个操作只会改变结点 AB 与结点 AC 这的势能。它的摊还代价为：

$$\begin{aligned}\Phi' - \Phi + c &= (\log(|AB'|) + \log(|AC'|)) - (\log(|AB|) + \log(|AC|)) + O(1)\\ &\le 3(\log(|A'|) - \log(|AC|)) + O(1)\end{aligned}$$

当没有 reverse 的情况下，结点 AB 不用上旋（因为它一直在树顶），而上旋结点 AC 的摊还代价显然小于等于 $3(\log(|AC|) - \log(|C|)) + O(1)$。

当存在 reverse 时，结点 AB 可能并不在树顶—它先作为 A 的父边（p）下旋，而后因为 reverse 操作而成为了实边（s）。为此，在势能函数中额外添加一项：$\log(|A.c|) - \log(|A.p|) + dis(A.c, A.p)$，即，将原来的结点 $A.p$ 作为 $A.s$ 时**单旋**上来所需的代价。显然，这个东西只有当旋转结点 $A.s$ 使得结点 $A.p$ 进入它的子树时才会改变，而这只会发生一次：



这次单旋的摊还代价为：

$$\begin{aligned}\Phi' - \Phi + c &= (\log(|A.s'|) + \log(|A.p'|) - \log(|A.p'|)) - (\log(|A.s|) + \log(|A.p|) - \log(|A.p|)) + O(1)\\ &\le 3(\log(|A.s'|) - \log(|A.s|)) + O(1)\end{aligned}$$

显然，这并不影响 splay 的势能分析。因此，将结点 AB 上旋的摊还代价是 $O(1)$。

splice 操作的摊还代价为上述三个代价之和，因此，AAA 树中 splice 操作的摊还代价为：

$$\begin{aligned}\Phi' - \Phi + c &\le (3(\log(|A'|) - \log(|AC|)) + O(1)) + (3(\log(|AC|) - \log(|C|)) + O(1)) + O(1)\\ &= 3(\log(|A'|) - \log(|C|)) + O(1)\end{aligned}$$

□

于是就可以得到 AAA 树的 expose 操作的时间复杂度了：

引理 2. *AAA 树中 expose 操作的时间复杂度为均摊 $O(\log n)$。*

证明. 我们称当前操作的结点为活动结点。在 splice 操作之后，活动结点从 C 变为了 A，因此 splice 操作摊还代价中的第一项小于等于活动结点势能的改变的 3 倍。由于活动结点的势能不会超过 $O(\log n)$，因此第一项的总时间复杂度为均摊 $O(\log n)$。

expose 操作中最多会执行 $O(\log n)$ 次 splice 操作，因此第二项的总时间复杂度为均摊 $O(\log n)$。

综上，AAA 树中 expose 操作的时间复杂度为均摊 $O(\log n)$。 □

与 LCT 类似，AAA 树的所有操作都是基于 expose 操作的，于是就可以得到 AAA 树的总时间复杂度了：

定理 1. *n 个结点的 AAA 树进行 m 次操作的时间复杂度为 $O((n+m)\log n)$。*

证明. 在 AAA 树中提取链、提取子树、加边、删边，都可以通过 $O(1)$ 次 expose 操作后再修改 $O(1)$ 个结点实现。

势能函数在任何时候都是大于等于 0，小于等于 $O(n\log n)$ 的。

因此 n 个结点的 AAA 树进行 m 次操作的时间复杂度为 $O((n+m)\log n)$。 □

4.2 ABF 树

对于 $k=1$ 的情况，图的每个双连通分量都是一个环。同样的，当 k 是一个常数的时候，图中的每个双连通分量的结构也是非常简单的：

引理 3. *若双连通图 G 的任意一条边最多属于 k 个不同的简单环，则 G 中最多有 $k+1$ 条连接着两个大于等于三度的点且仅由二度点构成的链。*

证明. 考虑逆否命题：若双连通分量连通图 G 有 k 条连接着两个大于等于三度的点且仅由二度点构成的链，则每条边至少属于 $k-1$ 个不同的简单环。

不失一般性的考虑 G 中的边 uv，将其中加入一个点 w，并将其拆为两条边 uw、vw。

由 Ear decomposition 的存在性可知，存在一种从 w 开始得到 G 的连链方法，满足任意时刻 G' 都是双连通的。（只需要以 w 为根跑 Ear decomposition 的构造即可）

每当从 S 新加一条链 ab 得到 S' 时，由双连通分量的性质可知，S 中一定存在一条从 a 经过 w 到达 b 的简单路径。于是 S' 一定会比 S 多出一个经过 w 也就是原图中 uv 的简单环。

得到图 G 需要连 $k-1$ 条链，因此 G 中每条边至少属于 $k-1$ 个简单环。 □

由于信息是可加的，因此在更新信息的时候，链可以缩成一个点来维护。于是可以使用圆方树来维护 k-沙漠，对于图中的每个双连通分量，记录其中的三度点以及连接它们的链。每个方点在更新的时候需要用到的信息是 $O(k)$ 的。

据此，我们就可以设计维护 k-沙漠的数据结构了：

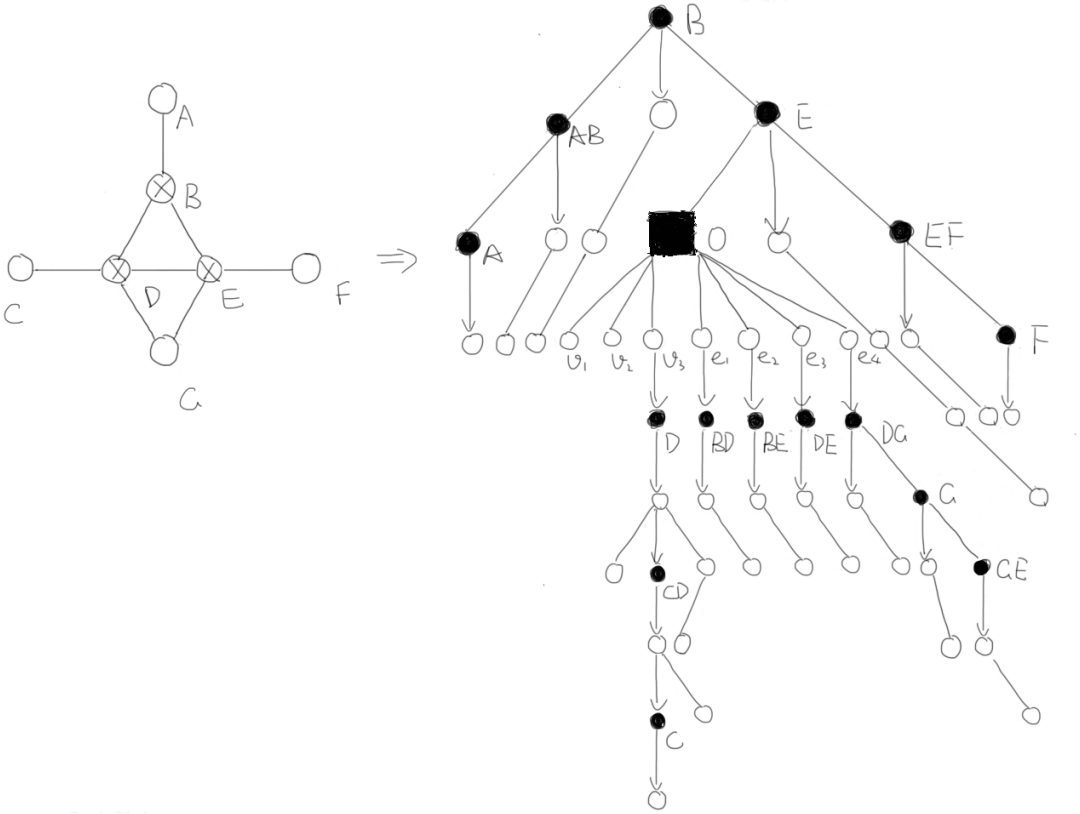
4.2.1 结构

ABF 树是一种基于 AAA 树的数据结构，依然采用链剖分的方式维护树的形态。为了维护双连通分量，在 AAA 树上增加一种新结点维护双连通分量。由于维护 k-沙漠时不可避免的需要用到边上的信息，接下来的 AAA 树默认已经在边上增加了辅助点。

ABF 树除了 compress 与 rake 之外，还有另一种结点—twist。twist 结点作为 compress 结点的变种出现在 compress 树中，用于维护一个双连通分量。为了保证 ABF 树的复杂度，任何时刻所有 twist 结点在 compress 树中均为**叶子**结点。

每个 twist 结点都有两个指针 l、r 与两个指针表 v 与 e，l 与 r 指向了其在 compress 树中的孩子，每个 v 中的指针指向了双连通分量中的一个关键点，每个 e 中的每个指针指向了一条连接着关键点的链。这里的关键点指哪些我们在双连通分量中关心的点，即，在双连通分量内部有三个出度的点、双连通分量的父亲连着的点，双连通分量的实孩子连着的点。

为了方便维护，v 与 e 中的指针并不会直接指向它们维护的 compress 结点，它们会指向一个空的 rake 结点，而这个 rake 结点的 c 指针会指向其所维护的 compress 结点。与 AAA 树中类似，如果需要维护的结点是双连通分量的父亲连着的点或是双连通分量的实孩子连着的点，则这个 rake 结点的 c 指针为空。



如图，其中黑圆点为 compress 结点，黑方点为 twist 结点，白点为 rake 结点，向左的边为 l 指针，向右的边为 r 指针，带箭头的边为 c 指针。

为了方便维护，每个 twist 结点都有两个指针 p 与 s，分别表示该点的父亲连着的点、实孩子连着的点在指针表 v 中是哪个结点。例如，图中 O 的 p、s 指针分别指向 v_1、v_2。这两个指针仅起标记作用，不参与信息的处理。

为了记录图的结构，e 表中的每个 rake 结点也有两个指针 p 与 s，分别表示该链连接

的端点。例如，图中 e_4 的 p、s 指针分别指向 $v3$、$v2$。这两个指针也仅起标记作用，不参与信息的处理。

当某个 compress 结点 A 是 twist 结点 B 的前驱时，A 的 n 指针可以指向其 rake 树中任何一条连向 B 的出边，当 A 是 B 的后继时它的 p 指针也是如此。

4.2.2 维护

与 AAA 树相似，这里只需要考虑 reverse 与 splice 操作对 twist 结点的影响。

基于链剖分的动态树有两个核心操作：

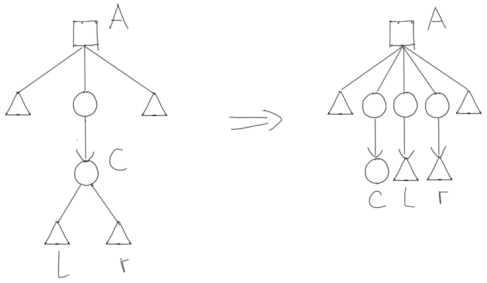
翻转（reverse）

当下传 twist 结点的翻转标记时，依然只需要交换 p、s 这两个指针即可。

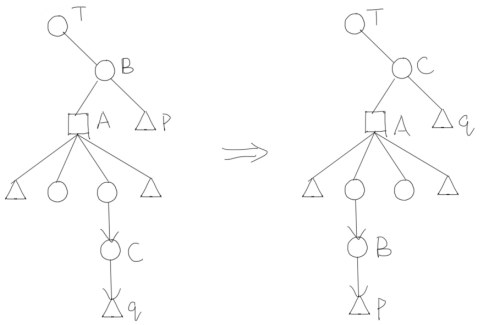
切换虚实（splice）

考虑如何将 twist 结点 A 的实孩子由结点 B 切换为结点 C：

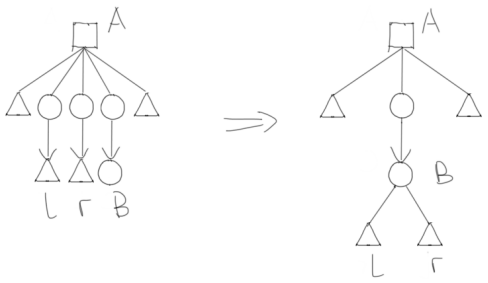
第一步：结点 C 可能并不是结点 A 的关键点，结点 C 可能位于某条链中。在这种情况下，先将结点 C 上旋，而后将其变为关键点：



第二步：将结点 A 在 compress 树中的前驱与后继都转到顶（当然，结点 A 可能是某条实链的顶，这样就没有前驱了），然后将结点 C 移出来并将结点 B 移进去：



第三步：结点 B 可能不再是结点 A 的关键点了（它本身是个二度点），将其与和它相连的两条链合并即可：



显然，ABF 树的 expose 操作与 AAA 中的 expose 操作除 splice 操作本身外完全一致，这里不再赘述。

考虑如何支持一些其他的基本操作，由于 ABF 树比起 AAA 树只是增加了 twist 结点，因此只考虑 twist 结点：

上传信息

twist 结点的信息分为两种：在最短路上的结点的信息和 $info_r$，在子树中但不在最短路上的结点的信息和 $info_v$。

$info_r$ 为 l 与 r 的 $info_r$，在 v、e 中处于最短路上的结点或链的 $info_r$ 之和。

$info_v$ 为 l 与 r 的 $info_v$，在 v、e 中处于最短路上的结点或链的 $info_v$，在 v、e 中不在最短路上的结点或链的 $info_r$ 及 $info_v$ 之和。

下传标记

twist 结点的标记分为两种：对最短路上的结点的修改 tag_r，对在子树中但不在最短路上的结点的修改 tag_v。

tag_r 会向 l 与 r 的 tag_r，在 v、e 中处于最短路上的结点或链的 tag_r 下传。

tag_v 会向 l 与 r 的 tag_v，在 v、e 中处于最短路上的结点或链的 tag_v，在 v、e 中不在最短路上的结点或链的 tag_r 及 tag_v 下传。

提取最短路

在 ABF 树中提取最短路与在 AAA 树中提取链除 expose 操作本身外完全一致，这里不再赘述。

提取子仙人掌

在 ABF 树中提取子仙人掌与在 AAA 树中提取子树除 expose 操作本身外完全一致，这里不再赘述。

加边

当连接两个连通分量时，ABF 树中的加边操作与 AAA 树中的加边操作完全一致，这里不再赘述。

当对一个连通分量内部进行操作时，将最短路上的所有双连通分量提取出来，缩掉连接它们的链后，新建一个大的双连通分量。

删边

当删边后会将一个连通分量分为两个时，ABF 树中的删边操作与 AAA 树中的删边操作完全一致，这里不再赘述。

当对一个双连通分量内部进行操作时，删去该边后跑一次 tarjan，对于得到的所有双连通分量，将它们按照顺序依次连起来。

实现上述所有操作，就可以得到一棵 ABF 树了。

4.2.3 分析

与 AAA 树相似，容易证明 ABF 树单次操作时间复杂度的一个上界—均摊 $O(\log^2 n + k \log k \log n)$：最多 $O(\log n)$ 次 splice 操作，每次 splice 操作的时间复杂度为均摊 $O(\log n + k \log k)$（每次 splice 时需要跑一次最短路）。

为了得到更紧的上界，需要进一步发掘 splice 操作的性质：

依旧以子树大小的对数和作为势能函数（twist 结点下方紧接着的 rake 结点是辅助点，不计入此处的势能函数）。

为了方便描述，令 $|x_0|$ 为结点 x 在 splice 操作前的子树大小，$|x_1|$ 为结点 x 在 splice 操作第一步后的子树大小，$|x_2|$ 为结点 x 在 splice 操作第二步后的子树大小，$|x_3|$ 为结点 x 在 splice 操作后的子树大小。

引理 4. *ABF 树中 splice 操作的摊还代价小于等于* $9(\log(|T_3|) - \log(|C_0|)) + O(k \log k)$。

证明. 当 splice 操作的对象是一个 compress 结点时，显然成立。考虑对 twist 结点进行 splice 操作：

当结点 B 与结点 C 都是大于等于三度的结点时，只需要执行 splice 操作中的第二步。

将结点 C 移出来这个操作只会改变 A、B 与 C 这三个结点的势能。它的摊还代价为：

$$\begin{aligned}\Phi' - \Phi + c &= (\log(|A_2|) + \log(|B_2|) + \log(|C_2|)) - (\log(|A_1|) + \log(|B_1|) + \log(|C_1|)) + O(1) \\ &\le 3(\log(|T_2|) - \log(|C_1|)) + O(1)\end{aligned}$$

由于结点 A 是一个叶子，因此它的前驱与后继都是它的祖先，因此将这两个点上旋的摊还代价为：

$$\begin{aligned}\Phi' - \Phi + c &\le 3(\log(|T_2|) + \log(|C_2|) - \log(|A_1|) - \log(|A_1|)) + O(1) \\ &\le 6(\log(|T_2|) - \log(|A_1|)) + O(1)\end{aligned}$$

于是第二步的摊还代价为：

$$\Phi' - \Phi + c \le 9\log(|T_2|) - 6\log(|A_1|) - 3\log(|C_1|) + O(1)$$

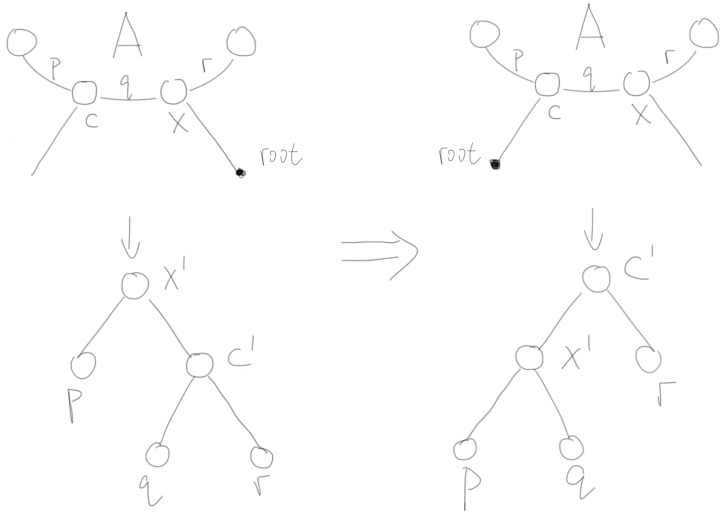
当结点 B 或结点 C 为二度点时，就需要考虑第一、第三步操作的摊还代价了。

在第一步中，结点 C 上旋的代价为 $3(\log(|C_1|) - \log(|C_0|)) + O(1)$。考虑计算因关键点改变而产生的势能变化，注意到可以改变 splice 操作中三步的顺序：先执行第二步中对结点 C 的修改，接着将结点 C 变为非关键点，再将结点 B 变为关键点，最后执行第二步中对结点 B 的修改。这样并不会影响 splice 操作的时间复杂度，因此可以据此分析：

在结点 B 变为关键点前，新建一个虚拟结点 B' 继承结点 B 的所有信息。将结点 B' 视为普通的 *compress* 结点加入势能函数，并在势能函数中添加一项：$-\log(|B'|)$，即，结点 B 变为关键点后其子树大小的对数的相反数。于是，把结点 B 变成关键点的摊还代价为：

$$\Phi' - \Phi + c = (\log(|B'_3|) + \log(|B'_3|) - \log(|B'_3|)) - (\log(|B_2|)) + O(1) = O(1)$$

在结点 C 变为非关键点前，一般不会修改其对应的虚拟结点 C'。这时，把结点 C 变为非关键点的摊还代价也为 $O(1)$。只有当两个二度关键点在同一条链上，且在 twist 结点处进行了 reverse 操作时，结点 C' 才会发生修改。由于翻转只有在 splice 操作中才会体现，于是它的势能改变可以在此统计：



这时，可视为对结点 C' 执行了一次上旋，这次上旋的代价小于等于 $\log(|A_1|) - \log(|C_1|) + O(1)$。

于是第一、第三步的摊还代价之和为：

$$\begin{aligned}\Phi' - \Phi + c &\le 3(\log(|C_1|) - \log(|C_0|)) + \log(|A_1|) - \log(|C_1|) \\ &\le 3(\log(|A_1|) - \log(|C_0|)) + O(1)\end{aligned}$$

每次 splice 操作还需要跑一次最短路，由于图中最多有 $O(k)$ 条边，这次最短路的时间复杂度为 $O(k \log k)$。

splice 操作的摊还代价为上述所有代价之和，因此，ABF 树中 splice 操作的摊还代价为：

$$\begin{aligned}\Phi' - \Phi + c &\le 9\log(|T_2|) - 6\log(|A_1|) - 3\log(|C_1|) + 3(\log(|A_1|) - \log(|C_0|)) + O(1) + O(k \log k) \\ &\le 9(\log(|T_3|) - \log(|C_0|)) + O(k \log k)\end{aligned}$$

□

于是就可以得到 ABF 树的 expose 操作的时间复杂度了：

引理 5. *ABF 树中 expose 操作的时间复杂度为均摊 $O(k \log k \log n)$。*

证明. 我们称当前操作的结点为活动结点。在 splice 操作之后，活动结点从 C 变为了 T，因此 splice 操作摊还代价中的第一项小于等于活动结点势能的改变的 9 倍。由于活动结点的势能不会超过 $O(\log n)$，因此第一项的总时间复杂度为均摊 $O(\log n)$。

expose 操作中最多会执行 $O(\log n)$ 次 splice 操作，因此第二项的总时间复杂度为均摊 $O(k \log k \log n)$。

综上，ABF 树中 expose 操作的时间复杂度为均摊 $O(k \log k \log n)$。 □

与 AAA 树类似，ABF 树的所有操作也都是基于 expose 操作的，于是就可以得到 ABF 树的总时间复杂度了：

定理 2. *n 个结点的 ABF 树进行 m 次操作的时间复杂度为 $O(mk \log k \log n + n \log n)$。*

证明. 在 ABF 树中提取最短路、提取子仙人掌、加边、删边，都可以通过 $O(1)$ 次 expose 操作后再修改 $O(1)$ 个结点实现。

势能函数在任何时候都是大于等于 0，小于等于 $O(n \log n)$ 的。

因此 n 个结点的 ABF 树进行 m 次操作的时间复杂度为 $O(mk \log k \log n + n \log n)$。 □

原问题可以直接使用 ABF 树维护。

时间复杂度为 $O(mk \log k \log n)$。

类似的，“动态仙人掌 IV”也可以使用 ABF 树维护。

时间复杂度为 $O(m \log n)$。

5 总结

本题是一道数据结构题，从已有问题出发，以 LCT、圆方树等基础数据结构为基础，提出了 ABF 树这一数据结构，考察了选手的建模能力与代码能力。

通过对 ABF 树进行细致分析，可以得到较为优美的时间复杂度。

部分分涵盖了原问题多种不同的子问题，难度不尽相同，有良好的区分度。

希望本题能对大家有所启发，加强对基础数据结构的灵活运用。

致谢

感谢中国计算机学会提供交流和学习的平台。

感谢徐先友老师对我的指导和帮助。

感谢王逸松前辈，陈俊锟前辈，赵雨扬同学为本文提供思路。

感谢周任飞同学，郎思轲同学本文的撰写给予帮助。

感谢张哲宇同学，高嘉煊同学为本文验稿。

参考文献

[1] Tarjan, Robert E., and Renato F. Werneck. “Self-adjusting top trees.”

[2] 陈俊锟，《神奇的子图》命题报告及其拓展，2017 年国家集训队论文

[3] 王逸松，仙人掌相关算法及其应用，2015 年国家集训队论文

[4] 黄志翱，浅谈动态树的相关问题及简单拓展，2014 年国家集训队论文

《整点计数》命题报告以及对高斯整数的若干研究

江苏省常州高级中学 徐翊轩

摘 要

本文从一道经典的信息学竞赛问题出发，阐述了笔者的候选队互测题《整点计数》命题的背景、思路以及题目的各种解法，对解决问题所需要的勾股数、高斯整数、数论筛法等知识点进行了系统的介绍，并对相关定理给出了形式化的证明。同时，将解决本题的思路进行一定的延伸与拓展，我们还能够得到一个收敛于圆周率的级数。

1 前言

数学是解决信息学问题的基本工具，是信息学竞赛中考查的重点，而整数数论就是其中一个重要的分支。在信息学竞赛中，有关整数数论的考点常常以筛法、莫比乌斯反演、对于积性函数的研究等形式出现，通常具有一定难度和深度。

笔者在对于一道经典问题的研究中得到启发，得到了该问题的一个拓展性较强的做法，并由此命制了笔者的集训队互测题《整点计数》。《整点计数》的各种解法涉及到勾股数、高斯整数、数论筛法等多个在信息学竞赛中出现频率不高的知识点。

笔者希望，本文能够在介绍《整点计数》命题的背景和解法的同时，将笔者的相关研究以论文的形式留于信息学竞赛界，起到抛砖引玉的作用。

本文的结构如下：

第 2 节介绍了笔者所说的经典问题，也即笔者命题的契机。

第 3 节介绍了《整点计数》的题面以及数据范围。

第 4 节挖掘了题目的一些较为显然的性质，给出了一个简单的暴力解法。

第 5 节介绍了勾股数的相关性质，并给出了题目的一个可行的部分分解法。

第 6 节深入挖掘问题本质，结合高斯整数、数论筛法，给出了题目的正确解法。

第 7 节对解决本题的思路进行了一定的拓展，得到了一个收敛于圆周率的级数。

第 8 节对前文用到的一些定理进行了补充证明。

第 9 节对题目的考点、难点、区分度设计以及全文进行了分析和总结。

2 一道经典问题

2.1 试题描述

本题来源于 HAOI2008《圆上的整点》[35]。

对于一个圆心在原点处，半径为给定值 r $(1 \le r \le 2 \times 10^9)$ 的圆，求其圆周上的整点个数，其中整点定义为横坐标、纵坐标均为整数的点。

2.2 传统解法

题目中 r 的范围较大，不方便我们直接枚举计数，需要进行一些数学推导。

不难发现，对于任意的 r ，坐标轴上都存在恰好 4 个符合要求的整点，并且各象限内满足要求的整点数相等，因此，不妨考虑计算第一象限内的整点数 Cnt ，再通过简单计算就可以得到答案，为 $4Cnt+4$。

设平面上某一整点的坐标为 (x,y) $(x,y \in \mathbb{N}^+)$ ，由勾股定理，该整点在圆周上的充分必要条件应为 $x^2+y^2=r^2$，稍加变形，有 $(r-x)(r+x)=y^2$。

令 $d=gcd(r-x,r+x)$ ，则有 $d^2 \times \frac{r-x}{d} \times \frac{r+x}{d} = y^2$，且 $\frac{r-x}{d}$ 与 $\frac{r+x}{d}$ 互质，因此，$\frac{r-x}{d}$ 与 $\frac{r+x}{d}$ 均为完全平方数，记 $\frac{r-x}{d}=u^2, \frac{r+x}{d}=v^2$ $(u<v)$ 。

此时，我们就可以用 u,v,d 来表示 r,x,y 了，有

$$2r = d \times (v^2+u^2), 2x = d \times (v^2-u^2), y = duv$$

注意到 d 是 $2r$ 的因数，并且 $u^2 < \frac{2r}{d}$ ，我们可以直接枚举 d 和 u ，并计算得到 $v=\sqrt{\frac{2r}{d}-u^2}$ ，再验证是否有 $u<v, gcd(u,v)=1$ 即可。

在上面的做法中，可能枚举到的 (d,u) 的个数约为

$$\sum_{i=1}^{\lfloor\sqrt{r}\rfloor} \sqrt{i} + \sum_{i=1}^{\lfloor\sqrt{r}\rfloor} \sqrt{\frac{r}{i}} = \Theta(\int_1^{\lfloor\sqrt{r}\rfloor} (\sqrt{x}+\sqrt{\frac{r}{x}})\,dx) = \Theta(r^{\frac{3}{4}})$$

计算上求解 $gcd(u,v)$ 的时间复杂度，上述做法总的时间复杂度为 $O(r^{\frac{3}{4}} Log\ r)$ 。

2.3 小结

本节给出了一道经典问题的传统解法，笔者也正是在这个解法的基础上展开了对这个问题的研究。虽然该传统解法效率不高，可拓展性也比较差，但其中用到的分解因数、整除分析等数学方法是具有相当的启发意义的，将会在下文被广泛应用。

[35] https://www.lydsy.com/JudgeOnline/problem.php?id=1041

3 《整点计数》

3.1 题目描述

对于给定的 N,k,P ，计算 $\sum_{i=1}^{N} f(i)^k \bmod P$ 。

其中 $f(x)$ 表示以原点为圆心，x 为半径的圆上整点的个数，例如 $f(1)=4, f(5)=12$ 。

3.2 输入格式

一行三个正整数 N,k,P 。

3.3 输出格式

一行一个整数 Ans ，表示答案对 P 取模的结果。

3.4 样例输入

5 1 998244353

3.5 样例输出

28

3.6 样例解释

不难发现，样例输入要求计算距原点 5 单位内，且至原点距离为整数的整点个数对 998244353 取模的结果。

形如 $(x,0),(-x,0),(0,x),(0,-x)$ 的符合要求的整点有 $4\times 5=20$ 个；

除了以上整点以外，还有 $(3,4),(-3,4),(3,-4),(-3,-4),(4,3),(-4,3),(4,-3),(-4,-3)$ 共 8 个符合要求的整点，总计 28 个。

3.7 数据范围与约定

对于所有测试数据，保证 $1\le N,k\le 10^{11}, 10^8\le P\le 10^9+9$ 。

详细的数据范围见下表。

测试点编号	分值	N	k	P
1	3	$\le 10^3$	≤ 5	P 是质数
2		$\le 8\times 10^3$		
3	6	$\le 2\times 10^5$		
4		$\le 5\times 10^5$		
5	5	$\le 3\times 10^6$		
6		$\le 2\times 10^7$		
7				
8				
9	8	$\le 2\times 10^8$	$=1$	
10				
11			$=2$	
12				
13	1	$\le 10^9$	$=15$	P 是 2 的次幂
14			$=18$	
15	4		$\le 10^9$	P 是质数
16		$\le 10^{10}$		
17				
18				
19	6	$\le 10^{11}$	$\le 10^{11}$	没有额外的限制
20				

4 一些暴力解法

4.1 算法一

考虑如何对于给定的 x ，计算 $f(x)$ ，即以原点为圆心，x 为半径的圆上整点的个数。

我们可以枚举这个圆上某一点的横坐标 i $(-x \le i \le x)$ ，并由勾股定理计算其对应的纵坐标 j 的绝对值 $|j| = \sqrt{x^2 - i^2}$ 。

通过判断 $|j|$ 是否为正整数或 0 ，我们可以确定 j 的整数解的个数。若 $|j|$ 为正整数，j 有 2 个整数解；若 $|j|$ 为 0 ，j 有 1 个整数解；否则，j 没有整数解。由此，我们可以得知以原点为圆心，x 为半径的圆上横坐标为 i 的整点的个数，从而累加得到 $f(x)$ 。

由此，我们得到了一个时间复杂度为 $O(N)$ 的计算 $f(N)$ 的算法。

多次调用这个算法，按照题目中的式子计算 $\sum_{i=1}^{N} f(i)^k \mod P$ ，就可以得到本题的一个时间复杂度为 $O(N^2)$ 的算法。可以通过测试点 1 ~ 2 ，期望得分 6 分。

4.2 算法二

由算法一中的 $f(x)$ 计算方式，我们可以尝试计算一些数对应的 $f(x)$ 的值：

$f(1)=4, f(2)=4, f(3)=4, f(4)=4, f(5)=12$

$f(6)=4, f(7)=4, f(8)=4, f(9)=4, f(10)=12$

$f(15)=12, f(25)=20, f(45)=12, f(65)=36, f(100)=20$

虽然 $f(x)$ 的数值分布没有明显的规律，但我们可以注意到所有 $f(x)$ 均是 4 的倍数。

我们可以在几何上直观地理解这一点：如果一个点 (x,y) 是整点 $(x>0,y>0)$，那么将其绕原点逆时针旋转 $90°$，$180°$，$270°$ 得到的点一定也是整点，且到原点距离与其到原点距离相等，因此 $f(x)$ 一定是 4 的倍数。

那么，我们可以进一步得到 $f(x)^k$ 一定是 4^k 即 2^{2k} 的倍数，当 $k\geq 15$ 时，$f(x)^k$ 一定是 2^{30} 的倍数，对 10^9+9 以内的某一个 2 的次幂取模的结果为 0 。

因此，直接输出 0 可以通过测试点 $13\sim 14$，期望得分 2 分。

结合算法一可以得到 8 分。

5 勾股数解法

5.1 算法三

算法二中对 $f(x)$ 在数值上的观察还让我们发现了一个事实：$f(x)$ 一般不大，难以达到 $O(x)$ 的级别，这意味着 $\sum_{i=1}^{N} f(i)$ 不会达到 $O(N^2)$ 的级别，如果我们可以得到一种高效的找到所有距原点距离为整数的点的方法，我们就可以得到本题的一种 $O(\sum_{i=1}^{N} f(i))$ 的算法。

可以发现，对于到原点距离为整数 z 的整点 (x,y) $(x>0,y>0)$，有 $x^2+y^2=z^2$，即 (x,y,z) 是一组**勾股数**。因此，有 $f(x)=4+4\times g(x)$，其中 $g(x)$ 表示以 x 为最大数的勾股数的对数。

定义 5.1：对于勾股数 (a,b,c) $(0<a,b<c,a^2+b^2=c^2)$，若 $gcd(a,b)=1$，那么，我们称勾股数 (a,b,c) 为一组**原生勾股数**，否则，我们称勾股数 (a,b,c) 为一组**派生勾股数**。

可以发现，任意一组派生勾股数 (x,y,z) 一定可以唯一地表示为某一组原生勾股数 (a,b,c) 的正整数倍。即对于一组派生勾股数 (x,y,z)，存在唯一的一组原生勾股数 (a,b,c) 和正整数 k $(k\geq 2)$，满足 $x=ka,y=kb,z=kc$ 。

那么，只要我们能够迅速地找到所有的原生勾股数，将它们乘上一系列的正整数，就可以迅速地找到所有的勾股数，从而计算 $g(x)$ 。

关于原生勾股数的生成，有以下经典引理。

若读者曾了解过则可自行跳过以下证明部分。

引理 5.1.1：对于任意的原生勾股数 (a,b,c) $(0<a,b<c,a^2+b^2=c^2,gcd(a,b)=1)$，存在正整数 n,m $(n<m,n+m\equiv 1\ (mod\ 2),gcd(n,m)=1)$，满足 $a=2mn,b=m^2-n^2,c=m^2+n^2$ 或 $a=m^2-n^2,b=2mn,c=m^2+n^2$ 。

证明：若 a,b 均为奇数，则有 $a^2 \equiv b^2 \equiv 1\ (mod\ 4)$ ，那么 $c^2 \equiv a^2 + b^2 \equiv 2\ (mod\ 4)$ ，而 2 不是模 4 的二次剩余，因此，a,b 中必然存在一个奇数、一个偶数。

不失一般性地，我们假设 $a = 2k$ ，那么 $(2k)^2 = 4k^2 = c^2 - b^2 = (c+b)(c-b)$ 。

由于 $(c+b)(c-b)$ 必须为偶数，所以 c 和 b 应当同为奇数。

记 $x = \frac{c+b}{2}, y = \frac{c-b}{2}$ ，显然，x,y 均为正整数。

若存在质数 p ，使得 $p|x, p|y$ ，那么 $p|x+y(=c), p|x-y(=b)$ ，从而 $p|c, p|b$ ，从而 $p|a$ ，这与 $gcd(a,b) = 1$ 矛盾，因此，不存在这样的质数 p ，即 $gcd(x,y) = 1$ 。

记 $xy = k^2 = p_1^{k_1} p_2^{k_2} p_3^{k_3} ... p_s^{k_s}$ ，其中 $p_1, p_2, p_3, ..., p_s$ 均为质数，$k_1, k_2, k_3, ..., k_s$ 均为正偶数。又由 $gcd(x,y) = 1$ ，可知 x,y 均为完全平方数。

令 $m = \sqrt{x}, n = \sqrt{y}$ ，$b = x - y = m^2 - n^2, c = x + y = m^2 + n^2, a = \sqrt{c^2 - b^2} = 2mn$ 。

并且，由于 $gcd(x,y) = 1$ ，$gcd(m,n) = 1$ 。又由于 $gcd(a,b) = 1$ ，即 $gcd(2mn, m^2 - n^2) = 1$ ，m,n 奇偶性不同，即 $n + m \equiv 1\ (mod\ 2)$ 。

引理 5.1.2：对于任意的正整数 n,m $(n < m, n + m \equiv 1\ (mod\ 2), gcd(m,n) = 1)$ ，$(2mn, m^2 - n^2, m^2 + n^2), (m^2 - n^2, 2mn, m^2 + n^2)$ 均为原生勾股数。

证明：首先，$(2mn)^2 + (m^2 - n^2)^2 = (m^2 + n^2)^2$ 恒成立。

因此，$(2mn, m^2 - n^2, m^2 + n^2), (m^2 - n^2, 2mn, m^2 + n^2)$ 均为勾股数。

若存在质数 p $(p \neq 2)$ ，使得 $p|2mn, p|m^2 - n^2$ ，那么 $p|m, p|n$ ，与 $gcd(m,n) = 1$ 矛盾，并且由于 $n + m \equiv 1\ (mod\ 2)$ ，$m^2 - n^2$ 是奇数，因此 $gcd(2mn, m^2 - n^2) = 1$ ，所以 $(2mn, m^2 - n^2, m^2 + n^2), (m^2 - n^2, 2mn, m^2 + n^2)$ 均为原生勾股数。

以上两个引理给出了一种经典的高效生成所有原生勾股数的算法，我们只需枚举所有符合条件的 m,n 即可，注意到本题中需要考虑的勾股数需满足 $c = m^2 + n^2 \leq N$ ，因此，m,n 的范围均为 $O(\sqrt{N})$ 级别的。

生成所有范围内的原生勾股数，再找到所有对应的派生勾股数，我们可以得到本题的一个时间复杂度为 $O(\sum_{i=1}^{N} g(i))$ 的算法，当 $N = 2 \times 10^7$ 时，找到的勾股数的对数（不记 a,b 的顺序）有 $49149097 \approx 5 \times 10^7$ 对。可以通过测试点 $1 \sim 8$ ，期望得分 38 分，结合算法二可以得到 40 分。

6 标准解法

6.1 算法四

算法一中提到的计算 $f(x)$ 的方法时间复杂度为 $O(x)$ ，并且难以拓展，我们需要一种更高效的计算 $f(x)$ 的方法。

当我们考虑二维平面上整点的问题的时候，整点 (x,y) 不仅可以代表二维平面上的一个点，同时，它也可以代表复平面上的一个**高斯整数**[36] $x + yi$ 。

[36]有关高斯整数的概念和术语将在第 8 节为不熟悉的读者进行补充说明

在本题中，我们要求整点 (x,y) 到原点的距离 $\sqrt{x^2+y^2}$ 为整数，另一种看待这个距离的方式是计算高斯整数 $x+yi$ 与其共轭复数 $x-yi$ 的乘积 $(x+yi)(x-yi)=x^2+y^2$ ，可以发现，得到的数值恰好是整点 (x,y) 到原点的距离的平方。

这一观察帮助我们将问题转化为了一个质因数分解的问题，也就是说，$f(x)$ 的值实际上等于与其共轭复数的乘积等于 x^2 的高斯整数的个数。

想要解决这个新问题，我们需要明确一点：类似于整数，高斯整数环是唯一分解整环[37]。这意味着高斯整数同样可以像整数的唯一分解一样，写成一系列不能够继续分解的**高斯质数**的积。

回顾整数的唯一分解，对于任意整数，例如整数 42 ，我们可以唯一地将其表示为若干质数的乘积：如

$$42=2\times3\times7$$

这样的表示“几乎”是唯一的，除非我们允许负数在这里出现，例如

$$42=2\times3\times7=(-2)\times3\times(-7)$$

和整数类似，对于任意高斯整数，例如 $15+20i$ ，我们同样可以以唯一地将其表示为若干高斯质数的乘积：如

$$15+20i=(2-i)(2+i)(3+4i)$$

这样的表示“几乎”是唯一的，除非我们允许一些因子成为它们的相反数，例如

$$15+20i=(2-i)(2+i)(3+4i)=(-2+i)(-2-i)(3+4i)$$

或者，我们允许一些因子乘上 i 和 $-i$ ，例如

$$15+20i=(i\times(2-i))(-i\times(2+i))(3+4i)=(1+2i)(1-2i)(3+4i)$$

在本题中，我们需要分解一系列的完全平方数，如果我们知道质数如何分解为高斯整数，我们就可以将需要分解的完全平方数分解为高斯整数。

费马平方和定理[38]：奇质数 p 可以表示为两个正整数 a,b 的平方和 $p=a^2+b^2$ ，当且仅当 $p\equiv1\ (mod\ 4)$ ，并且这样的表示若存在，在不计较排列顺序的情况下，是唯一的。

结合高斯整数唯一分解的性质，费马平方和定理实际上表明了：对于一个 $4k+1$ 型的质数 p ，我们可以找到一对共轭的高斯整数 $a+bi,a-bi$ ，使得它们的乘积为 p ，若不考虑将这两个因子取相反数，或者分别乘上 i 和 $-i$ 的情况，这样的高斯整数对是唯一的，同时，这一对高斯整数也是高斯质数；而任意一个 $4k+3$ 型的质数 p 不能被分解为一对共轭的高斯整数的乘积，因而 $4k+3$ 型的质数是高斯质数。

[37]这一点是不显然的，但为了不妨碍读者阅读的连续性，笔者将在第 8 节进行补充证明

[38]费马平方和定理的证明同样被放在了本文的第 8 节

质数 2 同样可以被分解为两个共轭的高斯质数的乘积 $(1-i)(1+i)$ 。

现在，我们已经可以将任意整数分解为高斯质数的乘积了，剩余的唯一问题在于如何计算 $f(x)$ 。

注意到算法二中的观察“如果一个点 (x,y) 是整点 $(x>0,y>0)$ ，那么将其绕原点逆时针旋转 $90°$ ，$180°$ ，$270°$ 得到的点一定也是整点，且到原点距离与其到原点距离相等”，用高斯整数的观点看待，就是任意高斯整数 $x+yi$ 在乘以 $i,-1,-i$ 后，其在复平面上到原点的距离不变，我们认为这样的高斯整数本质相同[39]，接下来我们讨论如何计算将 x^2 分解为本质不同的共轭高斯整数的乘积的方案数，即 $\frac{f(x)}{4}$ 。

我们先来考虑一个简单的情况，计算 $\frac{f(225)}{4}$ ，其中 $225=3^2\times5^2=3^2(2-i)^2(2+i)^2$ 。

由前文的讨论，不同的质数将分解为不同的高斯整数，而最终分解的得到的一对高斯整数是共轭的，这表明各个质数必须独立地分解为一对共轭的高斯整数的乘积。因此分解 225 的方案数 $\frac{f(225)}{4}$ 实际上是分解 $9,25$ 的方案数的乘积，即 $\frac{f(9)}{4}\times\frac{f(25)}{4}$ 。

分解一个 $4k+3$ 型质数 p 的 k 次幂 p^k 是容易的，当且仅当 k 是偶数，存在 1 种合法的分解方式，即 $p^k=p^{\frac{k}{2}}\times p^{\frac{k}{2}}$ ，在本题中，我们需要分解的数均为完全平方数，因此 $4k+3$ 型质数对分解的方案数没有影响。

考虑分解一个 $4k+1$ 型质数 p 的 k 次幂 p^k 的方案数，由上文的讨论，$p^k=(a+bi)^k(a-bi)^k$ 。那么 p^k 可以被分解为 $(a+bi)^x(a-bi)^{k-x}\times(a+bi)^{k-x}(a-bi)^x$ $(x\in\{0,1,2,...,k\})$ ，共 $k+1$ 种方案。

2 是一个特殊的质数，因为其分解而成的高斯质数 $(1-i),(1+i)$ 本质相同，因此，2 对分解的方案数同样没有影响。

至此，我们得到了一个能够在分解质因数的时间复杂度内计算 $f(x)$ 的算法，令 $x^2=p_1^{k_1}p_2^{k_2}p_3^{k_3}...p_s^{k_s}$ ，其中 $p_1,p_2,p_3,...,p_s$ 均为质数，$k_1,k_2,k_3,...,k_s$ 均为正偶数。那么有

$$f(x)=4\times\prod_{i=1}^{s}(1+[p_i\equiv1\ (mod\ 4)]\times k_i)$$

借助欧拉质数筛求出的每一个数 x 的最小质因子，在花费 $O(N)$ 的时间预处理后，我们可以在 $O(LogN)$ 的时间内分解一个整数 N ，因此可以得到本题的一种时间复杂度为 $O(NLogN)$ 的算法，可以通过测试点 $1\sim8$ ，期望得分 38 分。

并且，该算法较之算法三的优势在于，分解质因数本身不依赖于 $O(N)$ 级别的空间，并且处理 $f(i)$ 的过程是互相独立的，这表明我们可以预先花一定的时间计算 $\sum_{i=1}^{x}f(i)^k$ 的值 $value$ ，将其保存在代码中，这样我们在计算 $\sum_{i=1}^{N}f(i)^k$ $(N\geq x)$ 时只需要计算 $value+\sum_{i=x+1}^{N}f(i)^k$ 即可。

测试点 $9\sim12$ 中，k 的值只有 $1\sim2$ ，因此 $\sum_{i=1}^{N}f(i)^k$ 的真实值是在 64 位整型的范围内的，假设我们设定一个阈值 α ，预先处理 $N=\alpha,2\alpha,3\alpha,...,n\alpha$ 时，问题的答案，将它们保存在代码中，在实际运行过程中，我们便只需要再额外计算 $O(\alpha)$ 个 $f(x)$ 的值即可给出答案。

[39]事实上，这种关系称为相伴

若取 $\alpha = 4 \times 10^5$ ，那么我们需要在程序中储存 $\frac{2\times10^8}{4\times10^5} = 500$ 个预先处理的 $k = 1, 2$ 的答案，大约占 $12kb$ 的代码长度，是可以接受的。

此部分的时间复杂度为 $O(\alpha * factorize(N))$ ，其中 $factorize(N)$ 表示将 N 分解质因数的时间复杂度，可以通过测试点 $9 \sim 12$ ，总计可以通过测试点 $1 \sim 12$ ，期望得分 70 分，结合算法二可以得到 72 分。

6.2 算法五

在算法四的推导过程中，有一处至关重要的观察**“各个质数必须独立地分解为一对共轭的高斯整数的乘积”**这表明若取 $g(x) = \frac{f(x)}{4}$ ，$g(x)$ 是**积性函数**。

因此，若将原题的计算式稍加变形：$\sum_{i=1}^{N} f(i)^k = 4^k \times \sum_{i=1}^{N} g(i)^k$ ，我们要做的就是对积性函数 $h(x) = g(x)^k$ 求前缀和，考虑使用选手熟知的筛法解决这个问题。

令 $cnt_{0,x}$ 表示 x 以内 $4k+1$ 型质数的个数，$cnt_{1,x}$ 表示 x 以内 $4k+3$ 型质数的个数。无论使用洲阁筛，还是 $Min25$ 筛，我们都需要对于每一个不同的 $x = \lfloor \frac{N}{i} \rfloor$ ，求解 $cnt_{0,x}, cnt_{1,x}$ 的值。

我们先通过线性筛得到 $\sqrt{N}$ 以内所有的质数 $prime_i$ 以及 $prime_i$ 以内 $4k+1, 4k+3$ 型质数的个数 $pre_{0,i}, pre_{1,i}$ 。

定义

$$getcnt_0(N, i) = \sum_{j=1}^{N} [j \text{ is a prime or } Min_j > prime_i] \times [j \equiv 1 \ (mod\ 4)]$$

$$getcnt_1(N, i) = \sum_{j=1}^{N} [j \text{ is a prime or } Min_j > prime_i] \times [j \equiv 3 \ (mod\ 4)]$$

其中 Min_x 表示 x 最小的质因数。

直观地来说，$getcnt_0(N, i), getcnt_1(N, i)$ 表示的就是 N 以内的在埃拉特斯特尼筛算法进行第 i 轮后尚未被筛去的 $4k+1, 4k+3$ 型的数的个数。注意由于这里我们不需要考虑偶数，因此，我们认为在埃拉特斯特尼筛算法的第 1 轮，即 $prime_i = 2$ 时，没有数会被筛去。

一个合数 N 一定存在一个 $\sqrt{N}$ 以内的质因数，因此 $getcnt_0(x, Cnt), getcnt_1(x, Cnt)$ 即为所求的 $cnt_{0,x}, cnt_{1,x}$ ，其中 Cnt 为 $\sqrt{N}$ 以内的质数个数。

考虑如何通过 $getcnt_0(*, i-1), getcnt_1(*, i-1)$ 求出 $getcnt_0(*, i), getcnt_1(*, i)$ 。

若 $prime_i^2 > N$ ，那么埃拉特斯特尼筛算法的第 i 轮将不会筛去任何数，有

$$getcnt_0(N, i) = getcnt_0(N, i-1), \quad getcnt_1(N, i) = getcnt_1(N, i-1)$$

若 $prime_i^2 \le N$ ，考虑将埃拉特斯特尼筛算法的第 i 轮筛去的数，即以 $prime_i$ 为最小质因子的合数，从 $getcnt_0(N, i-1), getcnt_1(N, i-1)$ 中删除。

由于 $prime_i^2 \le N$ ，有 $\lfloor \frac{N}{prime_i} \rfloor \ge prime_i$ ，因此 $\lfloor \frac{N}{prime_i} \rfloor$ 以内最小质因数大于等于 $prime_i$ 的 $4k+1, 4k+3$ 型的数的个数分别为

$$getcnt_0(\lfloor \frac{N}{prime_i} \rfloor, i-1) - pre_{0,i-1}, \quad getcnt_1(\lfloor \frac{N}{prime_i} \rfloor, i-1) - pre_{1,i-1}$$

因此，若 $prime_i \equiv 1 \ (mod\ 4)$ ，有

$$getcnt_0(N,i) = getcnt_0(N,i-1) - (getcnt_0(\lfloor \frac{N}{prime_i} \rfloor, i-1) - pre_{0,i-1})$$

$$getcnt_1(N,i) = getcnt_1(N,i-1) - (getcnt_1(\lfloor \frac{N}{prime_i} \rfloor, i-1) - pre_{1,i-1})$$

否则，即 $prime_i \equiv 3 \ (mod\ 4)$ ，有

$$getcnt_0(N,i) = getcnt_0(N,i-1) - (getcnt_1(\lfloor \frac{N}{prime_i} \rfloor, i-1) - pre_{1,i-1})$$

$$getcnt_1(N,i) = getcnt_1(N,i-1) - (getcnt_0(\lfloor \frac{N}{prime_i} \rfloor, i-1) - pre_{0,i-1})$$

总结起来即为

$$getcnt_0(N,0) = \lfloor \frac{N-1}{4} \rfloor, \quad getcnt_1(N,0) = \lfloor \frac{N+1}{4} \rfloor$$

$$getcnt_0(N,i) = \begin{cases} getcnt_0(N,i-1) & prime_i^2 > N \\ getcnt_0(N,i-1) & \\ -(getcnt_0(\lfloor \frac{N}{prime_i} \rfloor, i-1) - pre_{0,i-1}) & prime_i^2 \le N, prime_i \equiv 1 \ (mod\ 4) \\ getcnt_0(N,i-1) & \\ -(getcnt_1(\lfloor \frac{N}{prime_i} \rfloor, i-1) - pre_{1,i-1}) & prime_i^2 \le N, prime_i \equiv 3 \ (mod\ 4) \end{cases}$$

$$getcnt_1(N,i) = \begin{cases} getcnt_1(N,i-1) & prime_i^2 > N \\ getcnt_1(N,i-1) & \\ -(getcnt_1(\lfloor \frac{N}{prime_i} \rfloor, i-1) - pre_{1,i-1}) & prime_i^2 \le N, prime_i \equiv 1 \ (mod\ 4) \\ getcnt_1(N,i-1) & \\ -(getcnt_0(\lfloor \frac{N}{prime_i} \rfloor, i-1) - pre_{0,i-1}) & prime_i^2 \le N, prime_i \equiv 3 \ (mod\ 4) \end{cases}$$

$$cnt_{0,x} = getcnt_0(x, Cnt), \quad cnt_{1,x} = getcnt_1(x, Cnt)$$

以上算法可以在 $O(\frac{N^{\frac{3}{4}}}{LogN})$ 的时间复杂度内计算得出对于每一个不同的 $x = \lfloor \frac{N}{i} \rfloor$ ，$cnt_{0,x}, cnt_{1,x}$ 的值，接下来，我们只需利用得到的值计算答案即可。以出题人的做法为例，以下内容推导得出了一个 $Min25$ 筛的做法。

定义

$$s(N,i) = \sum_{j=1}^{N} [Min_j \ge prime_i] \times h(j)$$

即所有满足最小质因子大于等于 $prime_i$ 的 h 值之和。

由定义，所求的 $\sum_{i=1}^{N} h(i) = s(N,2) + h(1) + [N \geq 2] \times h(2)$ 。

借助上面的计算结果，我们已经可以快速计算范围内 $4k+1, 4k+3$ 型质数的个数，因此 $s(N,i)$ 中质数的贡献能够被轻松计算：

$$3^k \times (cnt_{0,N} - pre_{0,i-1}) + (cnt_{1,N} - pre_{1,i-1})$$

接下来考虑答案中合数的贡献，我们枚举这个合数的最小质因子 $prime_j$ 及其出现次数 e ，由于 h 为积性函数，我们可以得到合数的贡献为

$$\sum_{j=i}^{prime_j^2 \leq N} \sum_{e=1}^{prime_j^{e+1} \leq N} (h(prime_j^e) \times s(\lfloor \frac{N}{prime_j^e} \rfloor, j+1) + h(prime_j^{e+1}))$$

将 $h(x)$ 函数的值展开即

$$\sum_{j=i}^{prime_j^2 \leq N} \begin{cases} \sum_{e=1}^{prime_j^{e+1} \leq N} ((2e+1)^k \times s(\lfloor \frac{N}{prime_j^e} \rfloor, j+1) + (2e+3)^k) & prime_j \equiv 1 \ (mod\ 4) \\ \sum_{e=1}^{prime_j^{e+1} \leq N} (s(\lfloor \frac{N}{prime_j^e} \rfloor, j+1) + 1) & prime_j \equiv 3 \ (mod\ 4) \end{cases}$$

总结起来即为

$$s(N,i) = \begin{cases} 0 & prime_i > N \\ 3^k \times (cnt_{0,N} - pre_{0,i-1}) + (cnt_{1,N} - pre_{1,i-1}) & \\ + \sum_{j=i}^{prime_j^2 \leq N} \sum_{e=1}^{prime_j^{e+1} \leq N} (h(prime_j^e) \times s(\lfloor \frac{N}{prime_j^e} \rfloor, j+1) + h(prime_j^{e+1})) & prime_i \leq N \end{cases}$$

求得 $\sum_{i=1}^{N} h(i)$ 后，最终答案即为 $4^k \times \sum_{i=1}^{N} h(i)$ 。

该部分的时间复杂度为 $O(Min25(N))$ ，在 $N \leq 10^{11}$ 的数据范围下，其运行效率与时间复杂度为 $O(\frac{N^{\frac{3}{4}}}{LogN})$ 的洲阁筛相当，可以通过全部的测试点，期望得分 100 分。

7 一些拓展

可以发现，题目中的函数 $f(x)$ 计算的始终是长度为整数的圆上的整点个数，而许多整点距离原点的距离有可能不是整数，例如 $(1,1)$ 距离原点的距离就是 $\sqrt{2}$ ，因此，它不会被任何一个 $f(x)$ 统计在内。

定义 $f'(x)$ 表示以原点为圆心，$\sqrt{x}$ 为半径的圆上整点的个数，不难发现，$f'(x^2) = f(x)$，$f'(x)$ 可以视作 $f(x)$ 的一个更加一般化的定义。

在算法四的讨论中，我们同样可以得到 $f'(x)$ 的计算方法：令 $x = p_1^{k_1} p_2^{k_2} p_3^{k_3} ... p_s^{k_s}$ ，其中 $p_1, p_2, p_3, ..., p_s$ 均为质数，$k_1, k_2, k_3, ..., k_s$ 均为正整数。那么有

$$f'(x) = 4 \times \prod_{i=1}^{s} ([p_i = 2] + [p_i \equiv 1 \ (mod\ 4)] \times (k_i + 1) + [p_i \equiv 3 \ (mod\ 4)] \times [k_i \equiv 0 \ (mod\ 2)])$$

定义函数 $\chi(x)$ $(x \in \mathbb{N}^+)$ ，满足

$$\chi(x) = \begin{cases} 1 & x \equiv 1 \ (mod\ 4) \\ -1 & x \equiv 3 \ (mod\ 4) \\ 0 & x \equiv 0 \ (mod\ 2) \end{cases}$$

可以发现，对于 $4k+1$ 型的质数 p_i ，有

$$\chi(p_i^x) = 1 \ (x \in \mathbb{N})$$

因此

$$\sum_{j=0}^{k_i} \chi(p_i^j) = k_i + 1 \quad (p_i \equiv 1 \ (mod\ 4))$$

对于 $4k+3$ 型的质数 p_i ，有

$$\chi(p_i^x) = \begin{cases} 1 & x \in \mathbb{N}, x \equiv 0 \ (mod\ 2) \\ -1 & x \in \mathbb{N}, x \equiv 1 \ (mod\ 2) \end{cases}$$

因此

$$\sum_{j=0}^{k_i} \chi(p_i^j) = [k_i \equiv 0 \ (mod\ 2)] \quad (p_i \equiv 3 \ (mod\ 4))$$

对于质数 $p_i = 2$ ，有

$$\chi(p_i^x) = \begin{cases} 1 & x = 0 \\ 0 & x \in \mathbb{N}^+ \end{cases}$$

因此

$$\sum_{j=0}^{k_i} \chi(p_i^j) = 1 \quad (p_i = 2)$$

综上，不同于前文需要对不同质因数进行分类的计算 $f'(x)$ 的方法，利用函数 $\chi(x)$ ，我们**甚至可以省去对于不同质因数的分类**，有

$$f'(x) = 4 \times \prod_{i=1}^{s} \sum_{j=0}^{k_i} \chi(p_i^j)$$

同时，注意到 $\chi(x)$ 同时也是一个**完全积性函数**，即对于任意的 $a, b \in \mathbb{N}^+$ ，均有 $\chi(a) \times \chi(b) = \chi(a \times b)$ 。结合上面 $f'(x)$ 的表达式，可以发现，$\sum$ 符号内枚举了各个质因数的各个指数，而 $\prod$ 符号则将不同的质因数的各种情况相互组合了起来，因此，该表达式同样也可以写作

$$f'(x) = 4 \times \sum_{i|x} \chi(i)$$

至此，我们得到了一个形式简单得惊人的 $f'(x)$ 的表达式。

现在，让我们来考虑一个问题：给定一个半径为 R 的圆，它内部存在多少个整点?

一方面，其内部的整点个数大约为其面积的大小，也即 πR^2 。这虽然只是一个近似的估计，但当 R 趋向于正无穷的时候，其误差相较于真实值的比值将会越来越小，以至于可以忽略不计。

另一方面，我们同样可以将 $f'(x)$ 从 1 至 R^2 求和，其内部的整点数应当约为[40]

$$\sum_{i=1}^{R^2} f'(i) = \sum_{i=1}^{R^2} 4 \times \sum_{j|i} \chi(j)$$

考虑交换此式的求和顺序，先枚举 j ，则可将其变形为

$$4 \times \sum_{j=1}^{R^2} \chi(j) \times \lfloor \frac{R^2}{j} \rfloor$$

忽略掉一些在 R 趋向于正无穷时细微的误差，我们可以得到一个大致成立的等式

$$\pi R^2 \approx 4 \times \sum_{j=1}^{R^2} \chi(j) \times \frac{R^2}{j} = 4R^2 \times \sum_{j=1}^{R^2} \frac{\chi(j)}{j}$$

并且，可以证明，当 R 趋向于正无穷时，等式两端的误差可以达到任意小。

此时，我们将等式两端的 R^2 除去，就会得到

$$\pi = 4 \times \sum_{i \in \mathbb{N}^+} \frac{\chi(i)}{i} = 4 \times (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots)$$

这正是一个令人难以置信的收敛于 π 的无穷级数。

8 补充证明

8.1 高斯整数

首先，我们需要为对高斯整数不熟悉的读者介绍有关高斯整数的概念和术语。

定义 8.1.1：若复数的实部和虚部均为整实数，则称其为**高斯整数**，即

$$\mathbb{Z}[i] = \{a + bi | a, b \in \mathbb{Z}, i^2 = -1\}$$

$\mathbb{Z}[i]$ 上所定义的加法和乘法就是普通的复数加法和乘法，高斯整数构成一个整数环。

定义 8.1.2：定义高斯整数 $a+bi$ 的**范数**为该高斯整数与其共轭的高斯整数的乘积，即

$$N(a + bi) = (a + bi)(a - bi) = a^2 + b^2$$

[40]此处用“约为”的原因是忽略了原点处的一个整点，但同样但当 R 趋向于正无穷的时候，其误差可以忽略不计

注意到高斯整数本身是不可比较大小的，但有了范数的定义，我们就可以对高斯整数进行某种意义上的定序。

定义 8.1.3：若高斯整数 $a+bi$ 满足 $N(a+bi)=1$ ，则称其为一个**单位数**。

由定义，一共有 4 个单位数，它们分别是 $1,-1,i,-i$ ，它们也是仅有的可逆高斯整数。

若两个高斯整数之间只相差一个单位数因子，我们称这两个高斯整数**相伴**[41]。

定义 8.1.4：对于高斯整数 X,Y ，若存在高斯整数 Z ，使得 $X=YZ$ ，则称 Y **整除** X ，记做 $Y\mid X$ 。

可以发现，若 Y 整除 X ，那么 $N(Y)$ 同样整除 $N(x)$ 。

定义 8.1.5：对于高斯整数 X,Y,g ，若 $g\mid X,g\mid Y$ ，则称 g 是 X,Y 的一个**公因数**。若 g 是 X,Y 的一个公因数，且对于 X,Y 的任意一个公因数 d ，都有 $d\mid g$ ，则称 g 是 X,Y 的**最大公因数**。

最大公因数可以看做范数最大的公因数。

如果两个高斯整数的最大公因数为单位数，则称这两个高斯整数**互质**。

定义 8.1.6：对于不是单位数的高斯整数 X ，若不存在不是单位数的高斯整数 Y,Z 使得 $X=YZ$ ，则称 X 是一个**高斯质数**。

由定义，我们可以写出范数最小的 4 个高斯质数：$1+i,1-i,-1+i,-1-i$ 。

注意普通的质数可能不是高斯质数，例如 $2=(1+i)(1-i)$ 。

8.2 唯一分解

在第 6 节中，我们提到了高斯整数的唯一分解性，我们接下来要证明这一点[42]。

引理 8.2.1（带余除法）：对于任意高斯整数 α,β ，且 $\beta\neq 0$ ，存在高斯整数 γ,λ ，使得

$$\alpha=\gamma\beta+\lambda,\ N(\lambda)<N(\beta)$$

证明：考虑复数 $\frac{\alpha}{\beta}=a+bi\ (a,b\in\mathbb{Q})$ ，取整数 c,d 满足 $|a-c|\le 0.5,|b-d|\le 0.5$ ，令 $\gamma=c+di$，则可代入计算 λ ，有

$$\begin{aligned}\lambda&=\alpha-\gamma\beta\\&=\beta(\frac{\alpha}{\beta}-\gamma)\\&=\beta((a-c)+(b-d)i)\end{aligned}$$

[41]在第 6 节中，为了方便读者快速理解算法，有相伴关系的两个高斯整数被称为“本质相同”

[42]不是所有环上都存在唯一分解性，例如偶数环上，$60=2\times 30=6\times 10$ ，其分解是不唯一的

那么

$$\begin{aligned} N(\lambda) &= N(\beta((a-c)+(b-d)i)) \\ &= N(\beta)N((a-c)+(b-d)i) \\ &= N(\beta)((a-c)^2+(b-d)^2) \\ &\le N(\beta)(0.5^2+0.5^2) \\ &< N(\beta) \end{aligned}$$

从而引理 8.2.1 得证。

引理 8.2.2（裴蜀等式）：对于任意非零的高斯整数 α,β ，令 d 为它们的最大公约数，那么存在高斯整数 γ,δ ，满足

$$\alpha\gamma+\beta\delta=d$$

证明：引理 8.2.1 的证明过程同时给出了一种进行高斯整数带余除法的方式，因此，我们可以通过**辗转相除法**来求得一对高斯整数的最大公因数。并且，我们将辗转相除的过程倒过来，逐步代入，可以构造得到一组满足引理 8.2.2 的等式，从而引理 8.2.2 得证。

引理 8.2.3（欧几里得引理）：若某个高斯质数 π 整除两个高斯整数的乘积 $\alpha\beta$ ，那么该高斯质数至少整除其中一个乘数，即 $\pi\mid\alpha$ 和 $\pi\mid\beta$ 中至少有一个成立。

证明：假设 $\pi\nmid\alpha$ ，我们只需证明此时 $\pi\mid\beta$ 即可证明引理。

由于 π 为高斯质数，且 $\pi\nmid\alpha$ ，有 π 和 α 的最大公因数为单位数，不妨令为 1 。

那么存在高斯整数 γ,δ ，满足

$$\pi\gamma+\delta\alpha=1$$

从而有

$$\pi\gamma\beta+\delta\alpha\beta=\beta$$

又因为 $\pi\mid\pi\gamma\beta,\pi\mid\delta\alpha\beta$ ，因此 $\pi\mid\beta$ ，引理 8.2.3 得证。

定理 8.2（唯一分解定理）：任何范数大于 1 的高斯整数均可以分解为有限个高斯质数的乘积，若不考虑单位数因子以及因子的排列顺序，这样的分解是唯一的。

证明：考虑数学归纳法，范数为 2 的高斯整数均为高斯质数，唯一分解定理在范数小于等于 2 的高斯整数范围内成立。假设唯一分解定理对于范数小于 $N(\alpha)$ 的高斯整数都成立，且 α 存在两种不同的分解为高斯质数乘积的方式

$$\alpha=\pi_1\pi_2\ldots\pi_s=\pi'_1\pi'_2\ldots\pi'_t$$

可知 $\pi_s\mid\pi'_1\pi'_2\ldots\pi'_t$ ，由欧几里得引理，π_s 至少整除 $\pi'_1,\pi'_2,\ldots,\pi'_t$ 中的一个，并且 $\pi'_1,\pi'_2,\ldots,\pi'_t$ 均为高斯质数，因此 π_s 至少与 $\pi'_1,\pi'_2,\ldots,\pi'_t$ 中的一个相伴。

不失一般性地，我们假设 π_s 与 π'_t 相伴，那么 $\pi_1\pi_2\ldots\pi_{s-1}$ 与 $\pi'_1\pi'_2\ldots\pi'_{t-1}$ 相伴，且 $N(\pi_1\pi_2\ldots\pi_{s-1})<N(\alpha)$ 。因此对于此数唯一分解定理成立，所以唯一分解定理对于范数等于 $N(\alpha)$ 的高斯整数也成立。

综上，唯一分解定理在高斯整数中成立。

8.3 费马平方和定理

这一小节中，我们将证明第 6 节中用到的费马平方和定理。

引理 8.3.1：设 p 是质数，那么

$$(p-1)! \equiv -1 \ (mod \ p)$$

证明：容易验证，$p=2$ 时，引理 8.3.1 成立，考虑 p 是奇质数的情况。

考虑乘法逆元不为本身的数，这样的数与其乘法逆元在 $(p-1)!$ 内两两配对，对于在模 p 意义下的 $(p-1)!$ 没有贡献。

剩余的数满足 $x^2 \equiv 1 \ (mod \ p)$ ，解得 $x \equiv 1 \ (mod \ p)$ 或 $x \equiv p-1 \ (mod \ p)$ 。

因此

$$(p-1)! \equiv 1 \times (p-1) \equiv -1 \ (mod \ p)$$

引理 8.3.2：如果质数 p 满足 $p \equiv 1 \ (mod \ 4)$ ，那么存在整数 x ，满足 $p \mid x^2+1$ 。

证明：由引理 8.3.1，有 $(p-1)!+1 \equiv 0 \ (mod \ p)$

注意到 $x \times y \equiv (p-x) \times (p-y) \ (mod \ p)$ ，如果 $p-1$ 是 4 的倍数，我们就可以将 $1 \times 2 \times \cdots \times (p-2) \times (p-1)$ 分成两半，前一半都两两配对相乘，对应地后面那一半也两两配对相乘，后面那一半就与前面那一半同余，即

$$1 \times 2 \equiv (p-1) \times (p-2) \ (mod \ p)$$

$$3 \times 4 \equiv (p-3) \times (p-4) \ (mod \ p)$$

$$\cdots$$

$$\frac{p-3}{2} \times \frac{p-1}{2} \equiv (p-\frac{p-3}{2}) \times (p-\frac{p-1}{2}) \ (mod \ p)$$

因此

$$1 \times 2 \times 3 \times \cdots \times \frac{p-1}{2} \equiv \frac{p+1}{2} \times \frac{p+3}{2} \times \frac{p+5}{2} \times \cdots \times (p-1) \ (mod \ p)$$

从而

$$(\frac{p-1}{2}!)^2 + 1 \equiv (p-1)! + 1 \equiv 0 \ (mod \ p)$$

引理 8.3.2 得证。

定理 8.3（费马平方和定理）：奇质数 p 可以表示为两个正整数 a,b 的平方和 $p=a^2+b^2$ ，当且仅当 $p \equiv 1 \ (mod \ 4)$ ，且这样的表示若存在，在不计排列顺序的情况下，是唯一的。

证明：费马平方和定理包含三点：(1)、$4k+3$ 型的质数不能写成两个正整数的平方和；(2)、$4k+1$ 型的质数可以写成两个正整数的平方和；(3)、$4k+1$ 型的质数写成两个正整数平方和的方式是唯一的。

由于对于任意整数 x ，x^2 模 4 只可能为 0 或 1 ，因此，两个正整数的平方和模 4 的余数不可能是 3 ，从而 $4k+3$ 型的质数不能写成两个正整数的平方和，第 (1) 点成立。

由引理 8.3.2，存在整数 x ，满足 $p \mid x^2+1$ ，即 $p \mid (x+i)(x-i)$ 。

假设一个 $4k+1$ 型的质数 p 是一个高斯质数，那么，根据引理 8.2.3，有 $p \mid x+i$ 或 $p \mid x-i$ 。但 $\frac{x\pm i}{p} = \frac{x}{p} \pm \frac{1}{p}i$ ，并非高斯整数，因此假设不成立，p 不是高斯质数。

那么，设 $p = uv$ $(u, v \in \mathbb{Z}[i], N(u), N(v) > 1)$ ，有

$$p^2 = N(p) = N(u)N(v)$$

由于 p 是质数，只可能 $N(u) = N(v) = p$ ，并且 u 和 v 的乘积为一个整数，因此 u 和 v 共轭。设 $u = a+bi, v = a-bi$ $(a, b \in \mathbb{Z})$，那么

$$p = (a+bi)(a-bi) = a^2+b^2$$

定理的第 (2) 点成立。

假设 p 存在两种不同的表示为两个正整数平方和的方式

$$p = a^2+b^2 = c^2+d^2$$

那么 $(a+bi)(a-bi) = (c+di)(c-di)$ ，有 $a+bi \mid (c+di)(c-di)$ 。

若 $a+bi$ 为高斯质数，那么有 $a+bi \mid c+di$ 或 $a+bi \mid c-di$ ，对应地，有 $a-bi \mid c-di$ 或 $a-bi \mid c+di$ ，又因为 $N(a+bi) = N(a-bi) = N(c+di) = N(c-di) = p$ ，$a+bi, a-bi$ 和 $c+di, c-di$ 两两相伴，$p = a^2+b^2 = c^2+d^2$ 实际上是同一种表示。

若 $a+bi$ 不是高斯质数，那么令 $a+bi = uv$ $(u, v \in \mathbb{Z}[i], N(u), N(v) > 1)$ ，有

$$p = (a+bi)(a-bi) = uv \times \overline{uv} = (u\overline{u}) \times (v\overline{v})$$

p 被分解为了两对共轭的，范数大于 1 的高斯整数的乘积，因而也可以被分解为两个大于 1 的整数的乘积，这与 p 是质数矛盾，因此 p 不存在两种不同的表示为两个正整数平方和的方式，定理的第 (3) 点成立。

至此，费马平方和定理得证，之前所用到的一些结论也全部证明完毕。

9 总结

《整点计数》一题综合考察了勾股数、高斯整数、数论筛法等知识点，并需要用到在程序中储存已知数据、观察函数规律等得分技巧，是考察选手综合素质的一道好题。

标准解法中所用到的拓展数域，将问题转化为分解质因数的思想是本题的核心，也是难点所在，选手需要有足够敏锐的洞察能力和强大的思维能力才能够发现标准解法，并解决问题。

非标准解法中，通过简单的暴力解法难以获得可观的分数；但利用题目性质的勾股数解法、以及去除繁琐的数论筛法的高斯整数解法，均可以获得十分可观的分数。并且各部

分数据均存在合理的梯度，为笔者没有想到的解法或是实现欠佳的解法提供了更好的得分空间。

互测现场的得分情况如下：4 位同学获得了 100 分，3 位同学获得了 88 分，另外各有 1 位同学获得了 38 分，40 分，70 分，2 位同学没有产生提交，因此没有得分。可见本题是一道区分度良好的题。

笔者希望，通过本文，在介绍本题的背景与解法的同时，向读者展现高斯整数理论的魅力，在不久的将来，相关研究成果也能够越来越多地出现在信息学竞赛中。

感谢

感谢中国计算机学会提供学习和交流的平台。

感谢江苏省常州高级中学的曹文老师，吴涛老师多年来给予我的关心和指导。

感谢家人、朋友对我的支持与鼓励。

感谢帮助过我的老师、同学们。

参考文献

[1] 维基百科 https://en.wikipedia.org/wiki/Gaussian_integer

[2] 维基百科 https://en.wikipedia.org/wiki/Pythagorean_triple

[3] 维基百科 https://en.wikipedia.org/wiki/Fermat%27s_theorem_on_sums_of_two_squares

[4] 苏剑林《从费马大定理谈起（三）：高斯整数》https://spaces.ac.cn/archives/2811

[5] 苏剑林《从费马大定理谈起（四）：唯一分解整环》https://spaces.ac.cn/archives/2819

[6] 苏剑林《从费马大定理谈起（七）：费马平方和定理》https://spaces.ac.cn/archives/2886

[7] 朱震霆《一些特殊的数论函数求和问题》2018 国家集训队论文集。

[8] 3Blue1Brown《Pi hiding in prime regularities》
https://www.patreon.com/posts/new-video-pi-in-11267915

浅谈树上分治算法

杭州学军中学 张哲宇

摘 要

本文简述了广为人知的边分治、点分治和全局平衡二叉树。

引言

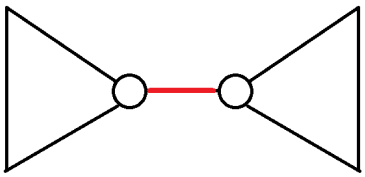
分治算法的基本思想是将一个规模非常大的问题分解为若干个规模较小的子问题，这些子问题相互独立且与原问题性质相同。求出子问题的解，就可间接得到原问题的解。

一般分治常用于链上（序列上），运用于树上的时候就需要一些技巧与套路了。

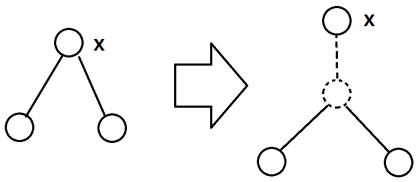
1 边分治

1.1 算法简介

对于一类询问树上路径的问题，可以选择一条边，处理完经过这条边的所有路径后，将这条边删除，递归剩下两棵较小的树。



为了保证算法的效率，需要每次删除一条边后剩下的较大树尽可能小，不难想到当度数均为常数的时候，该算法的复杂度为 $O(n\log n)$ 。



当点 x 的度数特别大的时候，可以加入一个空点和一条空边使得 x 的度数减一。
所以按照上图方法总可以加入 $O(n)$ 个空点和空边使得所有点度数为常数。

1.2 例题

1.2.1 题面描述

给出两棵点数均为 n 的树，求两个点，最大化他们在两棵树上的距离和。
$n \le 10^5$，边权可以为负

1.2.2 题解

对第一棵树进行边分治，之后的问题相当于给定两个集合，要求从两个集合中各取一个点，最大化他们的树上距离。建出虚树即可动态规划。

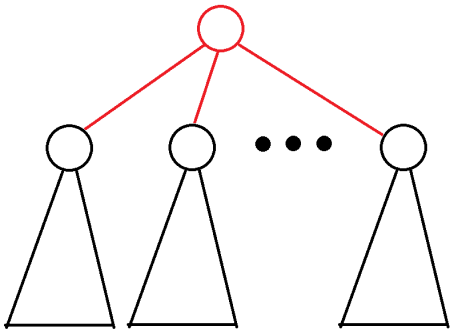
1.3 总结

因为每次只需要考虑两个子树之间的事情，往往考虑起来比较简单。
但因为增加了空点和空边所以有些题并不适合。

2 点分治

2.1 算法简介

对于一类询问树上路径或连通块的问题，可以选择一个点，处理完所有包含这个点的所有路径或连通块后，将这个点删除，递归剩下若干棵较小的树。



由于较小的树的大小至少除以二，故最多只会递归 $O(\log n)$ 层。

2.2 例题

2.2.1 题面描述

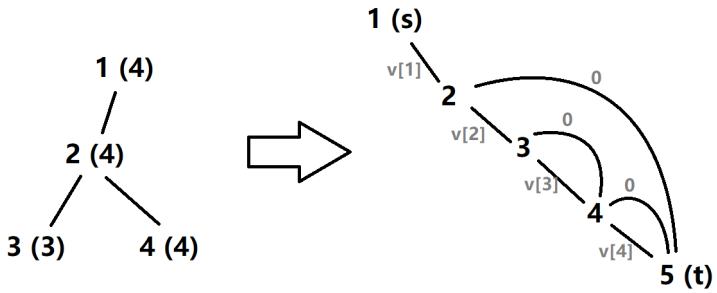
n 个点的树，每个点有一个非负点权，连通块的权值为点权和，求权值第 k 大的联通块。

$n \le 10^5$, $k \le 10^5$

2.2.2 题解

点分治之后，子问题增加了一个连通块必须包含根的限制，那么就很好解决了。

按 dfs 序重标号，在 i 号点上有两种决策，要么选择该点，到 $i+1$ ，要么割掉以 i 为根的子树，到 i 的后戳 $+1$ 。（约定俗成：前戳为 dfs 入栈时间，后戳为 dfs 出栈时间）



那么，一个连通块就可以与一条路径一一对应了。

现在就变成了 $O(n \log n)$ 个点，$O(n \log n)$ 条边的 DAG ，求 k 短路的经典问题。

时间复杂度：$O(n\log n + k\log(n\log n) + k\log k)$

2.3 例题：再谈 1.2

2.3.1 希望用点分治尝试一下

显然点分治和边分治极其相似，不妨用点分治做一下边分治的题。

2.3.2 遇到了一些问题

因为边权有负，求最远点除了建虚树之外没有什么好的方法。

而现在，删除一个点之后会分成 $O(n)$ 个集合，集合个数不再是常数，直接影响了算法复杂度。

2.3.3 解决方法

考虑每次选择最小的两个集合，求解只有这两个集合的答案，然后将这两个集合合并。这样每个集合只会被反复计算 $O(\log n)$ 次。乍一看，时间复杂度非常大。

2.3.4 仔细分析

凭什么点分治要平白无故地比边分治多一个 log ？

你可以拿起笔，在草稿纸上涂涂画画，写下一堆形如“$\log n - \log s$ ”的东西，然后拍案而起：“这样做复杂度是 $O(n\log n)$ 的！”

你也可以平心静气，突然发现，点分的时候每次选最小的两个集合合并，其本质就是边分治。

2.4 例题

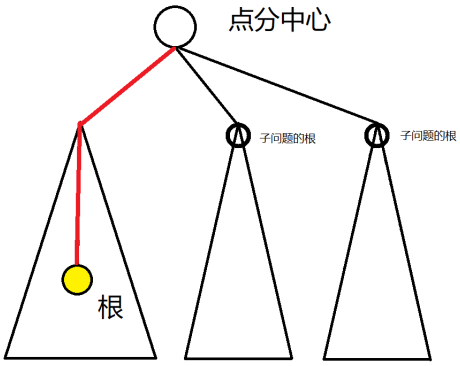
2.4.1 题面描述

给出一棵 n 个点的有根树，每个结点上有一个一次多项式。求每个结点到根的多项式乘积的和。

$n \le 10^5$

2.4.2 题解

点分治，在每个子问题中，假设我们已经求出了点分中心到根的多项式的乘积，原问题就可以变成若干个规模至少除以二的子问题。



问题就在于如何快速地求出红色路径的多项式乘积。

考虑运用已知的信息。当我求当前问题的路径的时候，找到该路径上的下一个点分中心，就可以直接得到下一个点分中心到根的路径的乘积。以此类推，可以得到 $O(\log n)$ 个多项式，其中第 i 个多项式长度级别不会超过 $\frac{n}{2^i}$ 。显然把他们卷起来的时间复杂度是 $O(n \log n)$ 。

那么，原问题的时间复杂度为 $O(n \log^2 n)$ 。

2.5 总结

处理树上强制包含根会非常好做的问题。

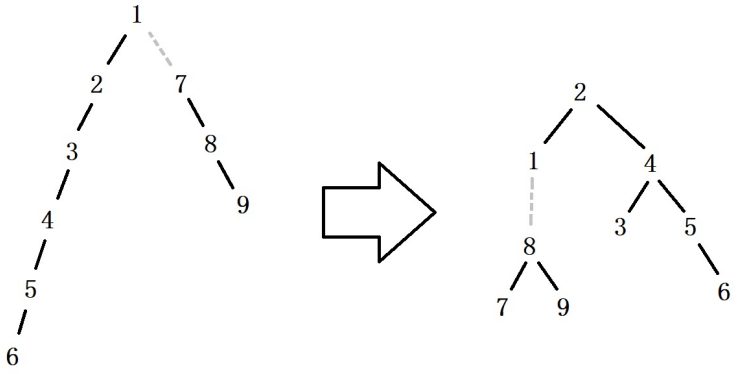
本身常数不小，但是当内层嵌套高复杂度的算法的时候常数很小。

3 全局平衡二叉树

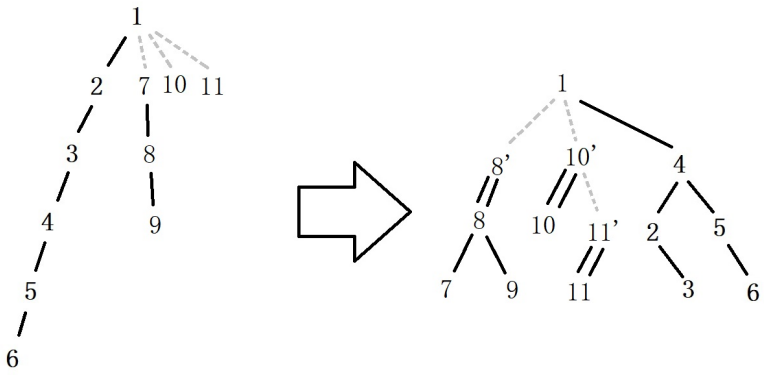
3.1 算法简介

对于有根树上的每个点，选择其子树大小最大的孩子作为其重儿子，其间的边称为重边，只关注重边的话，就将树剖成了若干条链。

每条链用一棵二叉树来维护，要注意这个二叉树的 mid 不是链上的 mid ，而是整棵子树的 mid 的话，整棵树的深度就是 $O(\log n)$ 的了。



有些时候，当虚孩子的数量多的时候，把虚孩子按大小建平衡二叉树，可以方便维护很多信息。



为了方便后文的叙述，定义一些东西。

全局平衡二叉树上一个子树中，到根只经过实边的点集称为子树实部，其余的称为子树虚部。

子树实部即对应树上的一条路径，这样的路径称为一个段。显然树上任意一条路径可以划分为 $O(\log n)$ 个段。

3.2 例题：再谈 2.4

3.2.1 优化树链剖分

这题有树链剖分的简单做法，时间复杂度 $O(n \log^3 n)$ 。

对于当前问题的最上一条重链，进行分治，将下一半的答案乘上上一半的乘积再加上上一半的答案。

把树链剖分链上部分改成全局平衡二叉树的分治方法就可以轻松做到 $O(n\log^2 n)$ 。

3.2.2 对比点分治做法

不难发现，这样的分治顺序，实际上是要求点分中心一定要在最浅的重链上。

与朴素点分治不同的是，他需要两次分治，才能严格把问题规约为原来的一半。但与之而来的关于链上信息的收益是巨大的。

3.3 例题

3.3.1 题面描述

给出一棵 n 个点的有根树，q 次询问距离点 p 距离不超过 d 且不在路径 $u-v$ 上的点权 max。

$n \le 10^5$, $q \le 10^5$, 树边长度均为 1

3.3.2 题解

预处理每个子树虚部的信息。

考虑 p , u , v 和他们所在重链底和根，把这些点之间的 $O(\log n)$ 个段找出来，然后 $O(1)$ 查询每个子树虚部的答案。

子树实部的答案以及个别特殊点单独处理。

个别单点会涉及到一个点的所有虚子树中去掉 $O(1)$ 个虚子树之后的信息，由于 max 不可减，对于度数大的点非常困难。对虚孩子也建了二叉树的力量就体现出来的。

总时间复杂度 $O((n+q)\log n)$

3.4 总结

虽然功能很强大，但常数略大，需要注意。

全局平衡二叉树不仅可以看作一个分治结构，也可以看作一种数据结构，但这不是本文的重点。

后记

如果你不是熟练的 oi 选手，没有关系，百度上有本文所需的所有前置知识。

本文中没有复杂度分析，因为分析的方法很多，无论你是推式子的 $\log n - \log s$ 选手，还是运算次数每加一规模减半的毛估估大师，都可以轻松得到正确的复杂度。

然而这并没有什么用，因为理论复杂度与运行效率没有直接关系，全局平衡二叉树只能出成交互题卡卡交互次数。树链剖分还是得学的，点分治也是得学的。

~~出题人一般是比较懒的，只会造菊花加链加二叉树。~~

参考文献

[1] 董永建，宋新波，徐先友等，《信息学奥赛一本通 (C++ 版)》，科学技术文献出版社。

感谢

[1] 王逸松、茹逸中与范致远为本文提供的帮助。

[2] CCF 给予的本次撰写论文的机会。

[3] 徐先友教练的审稿与帮助。

《组合数求和》命题报告

南京外国语学校 吴思扬

摘 要

本文将介绍本人在 2019 年候选队互测第三轮中命制的《组合数求和》一题，其考查了二项式定理、卢卡斯定理、中国剩余定理等关于组合数学、数论方面的知识；形式幂级数、等比数列求和、取模意义下组合数求值等技巧；扩展欧几里得、快速傅里叶变换等算法，要求选手有扎实的基本功和良好的思维能力。

除此之外，近年来计算模意义下答案的问题在编程竞赛中越来越常见，也出现了不少解决此类问题的算法，但是其中一些算法往往仅能在模数为质数的情况下发挥较大作用。本文也希望起到抛砖引玉的效果，引发大家对于一些计算模合数意义下答案的问题的思考。

1 试题

1.1 题面描述

定义 $f(j) \equiv \sum_{i=0}^{n-1} \binom{i\cdot d}{j} \pmod{M}$，$0 \le f(j) < M$，其中 n, d, M 为给定值。

现在给定 m，输出 $f(0)\ xor\ f(1)\ xor\ f(2)\ xor\ \cdots\ xor\ f(m-1)$ 的值。

其中 $\binom{n}{m}$ 为组合数 n 选 m，即 $\binom{n}{m} = \begin{cases} \frac{n!}{m!\cdot(n-m)!}, & 0 \le m \le n \\ 0, & otherwise \end{cases}$。$xor$ 表示异或和。

1.2 输入格式

一行四个整数 n, m, d, M，由空格隔开。

1.3 输出格式

一行一个整数表示答案。

1.4 样例输入

3 2 3 998244353

1.5 样例输出

10

1.6 样例解释

$f(0) \equiv C_0^0 + C_3^0 + C_6^0 \equiv 1 + 1 + 1 \equiv 3 \pmod{998244353}$
$f(1) \equiv C_0^1 + C_3^1 + C_6^1 \equiv 0 + 3 + 6 \equiv 9 \pmod{998244353}$
$f(0)\ xor\ f(1) = 3\ xor\ 9 = 10$

1.7 限制与约定

本题采用捆绑测试。

对于所有测试包均满足 $1 \le d \le 100$, $1 \le m \cdot d \le 3 \times 10^6$, $1 \le n \cdot d \le 10^9$, $10^8 \le M \le 10^9$。

对于所有测试包，空间限制均为 512MB。对于第 1, 2, 3, 4 测试包，时间限制为 1s；对于第 5, 6, 7, 8, 9, 10 测试包，时间限制为 3s。

测试包编号	测试包分值	其它约定
1	4	$n \cdot d,\ m \le 2000$
2	10	$m \le 400$
3	6	$m \le 8000,\ M = 998244353$
4	6	$m \le 8000$
5	7	$d = 1$
6	22	$gcd(d,\ M) = 1$
7	7	$d = 2$
8	7	$d = 4$
9	8	d 为质数
10	23	–

2 约定与记号

对于一元多项式 $F(x) = \sum_{i=0}^{n} f_i \cdot x^i$，记 n 为这个多项式的次数，$[x^i]F(x)$ 为其次数为 i 的单项式的系数，若不存在则为 0，即 $[x^i]F(x) = \begin{cases} f_i, & 0 \le i \le n \\ 0, & otherwise \end{cases}$。

令 $(n)_k = \Pi_{i=1}^{k} (n - i + 1)$。

令 $N = n \cdot d$，其中 n, d 为题面中的变量。

3 算法讨论

3.1 算法 1

按照题意计算出需要用到的所有组合数在对 M 取模意义下的值，并对其求和计算出 f 的值，再对 f 求异或和得到答案。计算组合数 $\binom{n}{m}$ 可以用其递归公式计算。递归公式如下：

$$\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}, \ \forall n, m, \ 1 \le m \le n-1$$

边界为：

$$\binom{n}{0} = \binom{n}{n} = 1, \ \forall n, \ n \ge 0$$

时间复杂度 $O(Nm)$，可以通过第一个测试包。

3.2 算法 2

定理 1. 二项式定理

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} \cdot x^i \cdot y^{n-i}$$

当 $y = 1$ 时，有 $(x+1)^n = \sum_{i=0}^{n} \binom{n}{i} \cdot x^i$。

若将其看作关于 x 的一元多项式，则有：

$$f(j) \equiv \sum_{i=0}^{n-1} \binom{i \cdot d}{j} \equiv \sum_{i=0}^{n-1} [x^j](x+1)^{i \cdot d} \pmod{M}$$

令 $C_n(x) = \sum_{i=0}^{n-1} (x+1)^{i \cdot d}$，则 $f(j) \equiv [x^j]C_n(x) \pmod{M}$。于是问题变成如何求 $C_n(x)$ 的前 m 项。

考虑使用倍增算法进行求解。令 $D_n(x) = (x+1)^{n \cdot d}$，则有：

$$C_{n+1}(x) = C_n(x) \cdot (x+1)^d + (x+1)^0, \ D_{n+1}(x) = D_n(x) \cdot (x+1)^d$$

$$C_{2n}(x) = C_n(x) \cdot (1 + D_n(x)), \ D_{2n}(x) = D_n(x)^2$$

由于只要求 $C_n(x)$ 的前 m 项，故所有多项式乘法运算可以在 $\bmod x^m$ 意义下进行。

如果采用 $O(m^2)$ 的方法计算两个多项式的乘法，时间复杂度为 $O(m^2 \ log_2 \ N)$，可以通过第二个测试包。

多项式乘法可以通过快速傅里叶变换来优化，总时间复杂度为 $O(m \ log_2 \ m \ log_2 \ N)$。具体实现方法可以参考毛啸在 2016 年信息学奥林匹克中国国家队候选队论文中所写的《再探

快速傅里叶变换》，由于其方法与本文其它内容关联不大，限于篇幅不再鳌述。如果实现模数为 998244353 情况下的数论变换可以通过第三个测试包，如果使用任意模数卷积可以通过前四个测试包。

3.3 算法 3：$d = 1$ 时的做法

当 $d = 1$ 时，$f(j) \equiv \sum_{i=0}^{n-1} \binom{i}{j} \equiv \sum_{i=j}^{n-1} \binom{i}{j} \pmod{M}$。

定理 2.

$$\binom{n+1}{k+1} = \binom{k}{k} + \binom{k+1}{k} + \cdots + \binom{n}{k}$$

证明. 根据组合数递归式 $\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$, $\forall n, m,\ 1 \le m \le n-1$ 有：

$$\begin{aligned}
\binom{n+1}{k+1} &= \binom{n}{k} + \binom{n}{k+1} \\
&= \binom{n}{k} + \binom{n-1}{k} + \binom{n-1}{k+1} \\
&= \binom{n}{k} + \binom{n-1}{k} + \binom{n-2}{k} + \binom{n-2}{k+1} \\
&= \cdots \\
&= \binom{n}{k} + \binom{n-1}{k} + \cdots + \binom{k+1}{k} + \binom{k+1}{k+1} \\
&= \binom{n}{k} + \binom{n-1}{k} + \cdots + \binom{k+1}{k} + \binom{k}{k}
\end{aligned}$$

□

故 $f(j) \equiv \binom{n+1}{j+1} \pmod{M}$。问题转化为如何在对 M 取模意义下求出组合数的值。

观察要求的组合数，其形式为 $\binom{n}{j}$，其中 n 很大，不适合再用**算法 1** 的方法求解。但是 j 相对较小，因此考虑按照 $\binom{n}{j} = \frac{(n)_j}{j!}$ 来计算。由于 $gcd(j!, M)$ 可能不是 1，因此 $j!$ 有可能在对 M 取模意义下不存在逆元。我们对 M 分解质因数得 $M = \Pi_{i=1}^{k} p_i^{q_i}$，将 $\frac{(n)_j}{j!}$ 写成 $\frac{A \cdot \Pi_{i=1}^{k} p_i^{a_i}}{B \cdot \Pi_{i=1}^{k} p_i^{b_i}}$ 的形式，保证 $\gcd(B, M) = 1$，即 B 的逆元在对 M 取模意义下存在，又由于 $\forall i,\ a_i \ge b_i$，故可以通过计算 $A \cdot B^{-1} \cdot \Pi_{i=1}^{k} p_i^{a_i - b_i}$ 得到 $\binom{n}{j}$ 的值，其中 $B \cdot B^{-1} \equiv 1 \pmod{M}$。$A$, B, a_i, b_i 的值可以按照 j 从小往大的顺序，每次在 $\binom{n}{j-1}$ 的基础上修改得到。时间复杂度 $O(m \cdot (log_2 N + log_2 M))$，可以通过第五个测试包。

3.4 算法 4

与算法二一样，考虑如何求 $C_n(x)$。为了方便，在下文中，若无特殊说明，均用 $C(x)$ 来代替 $C_n(x)$。

我们有：

$$C(x) = (x+1)^{0\cdot d} + (x+1)^{1\cdot d} + \cdots + (x+1)^{(n-1)\cdot d} \tag{24}$$

$$(x+1)^d \cdot C(x) = (x+1)^{1\cdot d} + (x+1)^{2\cdot d} + \cdots + (x+1)^{(n-1)\cdot d} + (x+1)^{n\cdot d} \tag{25}$$

与等比数列求和类似，(25) − (24) 为：

$$[(x+1)^d - 1] \cdot C(x) = (x+1)^{n\cdot d} - (x+1)^{0\cdot d}$$

即：

$$C(x) = \frac{(x+1)^{n\cdot d} - 1}{(x+1)^d - 1}$$

可以发现 $[x^0]((x+1)^{n\cdot d} - 1) = [x^0]((x+1)^d - 1) = 0$，即分子分母都可以写成 x 乘上多项式的形式，因此可以对分子分母同时约去 x。

令 $A(x) = \sum_{i=1}^{n\cdot d} \binom{n\cdot d}{i} \cdot x^{i-1}$，$B(x) = \sum_{i=1}^{d} \binom{d}{i} \cdot x^{i-1}$，那么

$$C(x) = \frac{A(x)}{B(x)},\ B(x) \cdot C(x) = A(x)$$

当 $d = 1$ 时，可以得到与**算法 3** 一样的解法。

从 $B(x) \cdot C(x) = A(x)$ 可以得到：

$$\sum_{j=0}^{d-1} ([x^j]B(x)) \cdot ([x^{i-j}]C(x)) = [x^i]A(x)\ ,\ \forall\ 0 \le i < N \tag{26}$$

3.4.1 算法 4.1：$gcd(d, M) = 1$ 时的做法

由 (26) 得：

$$[x^i]C(x) = \frac{[x^i]A(x) - \sum_{j=1}^{d-1} ([x^j]B(x)) \cdot ([x^{i-j}]C(x))}{[x^0]B(x)}\ ,\ \forall\ 0 \le i < m \tag{27}$$

由于 $[x^0]B(x) = \binom{d}{1} = d$，并且 $gcd(d, M) = 1$，所以 $[x^0]B(x)$ 在对 M 取模意义下的逆元存在，因此直接使用 (27) 按照 i 从小到大的顺序对 $[x^i]C(x)$ 进行求解即可。

$A(x)$, $B(x)$ 的计算方法与**算法 3** 一样，时间复杂度为 $O(m \cdot (log_2 N + log_2 M))$。计算 $[x^i]C(x)$ 一项的时间复杂度为 $O(d)$，故总时间复杂度为 $O(m \cdot (log_2 N + log_2 M + d))$。可以通过第三、第五、第六个测试包。

3.4.2 算法 4.2：$d = 2$ 时的做法

将 M 分解成 $2^a \cdot b$ 的形式，其中 $b \equiv 1 \pmod 2$。对每个 $f(j)$ 求出其对 2^a 取模意义下的值和对 b 取模意义下的值之后再使用中国剩余定理 (CRT) 算法就能求出 $f(j)$ 对 M 取模意义下的值。

计算对 b 取模的值可以使用**算法 4.1**，问题变成如何求对 2^a 取模下的值。

当 $d = 2$ 时，$B(x) = x + 2$，将其带入到 (26) 可得：

$$2 \cdot [x^i]C(x) + [x^{i-1}]C(x) = [x^i]A(x)\ ,\ \forall\ i \geq 0$$

即：

$$[x^i]C(x) = [x^{i+1}]A(x) - 2 \cdot [x^{i+1}]C(x)\ ,\ \forall\ i \geq 0 \tag{28}$$

将右侧 $C(x)$ 的项不断用 (28) 带入可得：

$$\begin{aligned}
[x^i]C(x) &= [x^{i+1}]A(x) - 2 \cdot [x^{i+1}]C(x) \\
&= [x^{i+1}]A(x) - 2 \cdot [x^{i+2}]A(x) + 4 \cdot [x^{i+2}]C(x) \\
&= [x^{i+1}]A(x) - 2 \cdot [x^{i+2}]A(x) + 4 \cdot [x^{i+3}]A(x) - 8 \cdot [x^{i+3}]C(x) \\
&= \cdots \\
&= \sum_{j \geq 0} (-2)^j \cdot [x^{i+j+1}]A(x)
\end{aligned}$$

当 $j \geq a$ 时，$(-2)^j \cdot [x^{i+j+1}]A(x) \equiv 0 \pmod{2^a}$，故：

$$[x^i]C(x) \equiv \sum_{j=0}^{a-1} (-2)^j \cdot [x^{i+j+1}]A(x) \pmod{2^a} \tag{29}$$

使用 (29) 就可以计算出 $[x^i]C(x)$ 对 2^a 取模意义下的值。计算一项所需时间为 $O(a)$ 即 $O(log_2\ M)$。结合计算对 b 取模的部分，总时间复杂度为 $O(m \cdot (log_2 N + log_2 M + d))$，结合**算法 4.1** 可以通过第三、第五到第七个测试包。

3.4.3 算法 4.3：$d = 4$ 时的做法

与**算法 4.2** 一样，将 M 分解成 $2^a \cdot b$ 的形式，对 b 取模意义下的值使用**算法 4.1** 计算，考虑怎么算对 2^a 取模意义下的值。

当 $d = 4$ 时，$B(x) = 4 + 6x + 4x^2 + x^3$，将其带入到 (26) 可得：

$$4 \cdot [x^i]C(x) + 6 \cdot [x^{i-1}]C(x) + 4 \cdot [x^{i-2}]C(x) + [x^{i-3}]C(x) = [x^i]A(x)\ ,\ \forall\ i \geq 0$$

即：

$$[x^i]C(x) = [x^{i+3}]A(x) - 4\cdot[x^{i+3}]C(x) - 6\cdot[x^{i+2}]C(x) - 4\cdot[x^{i+1}]C(x),\ \forall\ i \geq 0 \quad (30)$$

可以发现对于 (30) 的右侧来说，$C(x)$ 的每一项系数 $\equiv 0 \pmod 2$。与**算法 4.2** 中得到 (29) 的方式类似，我们称一轮操作为将式子右侧的每个 $C(x)$ 项用 (30) 带入，即用 (30) 的右式表示。如果操作前右侧式子中 $C(x)$ 每一项系数都是 2^i 的倍数，那么一轮操作后式子右侧 $C(x)$ 的每一项系数都将是 2^{i+1} 的倍数。如果对 (30) 进行 $a-1$ 轮操作，$C(x)$ 的每一项系数将会 $\equiv 0 \pmod{2^a}$，此时，式子右侧就是若干 $A(x)$ 项的系数的和，问题就得到了解决。

迭代操作会进行 a 轮，迭代后的式子长度可以达到 $O(ad)$，对每一项用 (30) 展开需要 $O(d)$ 的复杂度，故计算迭代的式子需要复杂度为 $O(a^2d^2)$，即 $O((log_2M)^2\cdot d^2)$。之后计算 $[x^i]C(x)$ 每一项需要时间复杂度为 $O(log_2M\cdot d)$，再加上计算对 b 取模意义下的结果，总时间复杂度为 $O(m\cdot(log_2N + log_2M\cdot d) + (log_2M)^2\cdot d^2)$。结合**算法 4.1** 和**算法 4.2** 可以通过第三、第五到第八个测试包。

3.4.4 算法 4.4：d 是质数时的做法

与前两个算法（**算法 4.2** 和**算法 4.3**）类似，可以将 M 分解成 $d^a\cdot b$ 的形式，其中 $gcd(b,\ d) = 1$，对模 d^a 和模 b 意义下分别求解。模 b 意义下的做法可以使用**算法 4.1**，考虑对 d^a 取模怎么做。

由 (26) 得：

$$([x^{d-1}]B(x))\cdot([x^i]C(x)) = [x^{i+d-1}]A(x) - \sum_{j=0}^{d-2}([x^j]B(x))\cdot([x^{i+d-1-j}]C(x)),\ \forall\ i \geq 0 \quad (31)$$

观察前两个算法（**算法 4.2** 和**算法 4.3**），都是对 (28), (30) 右侧的每一项 $C(x)$ 不断地用 (28), (30) 迭代下去，直到系数在模意义下都为 0 为止。那么对于 d 是质数的情况，只要能保证有限轮迭代之后能使右式在对 d^a 取模意义下只剩下若干个 $A(x)$ 项求和，那么就可以类似地通过每次将 (31) 右侧的一个 $C(x)$ 项用 (31) 右式替代来做。

定理 3. *卢卡斯定理*

对于非负整数 m, n 以及质数 p：

$$\binom{m}{n} \equiv \prod_{i=0}^{k}\binom{m_i}{n_i} \pmod p$$

其中：

$$m = m_kp^k + m_{k-1}p^{k-1} + \cdots + m_1p + m_0,\ 0 \leq m_i < p$$

$$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0,\ 0 \le n_i < p$$

由卢卡斯定理可知，$\forall\ 0 < i < d$, $\binom{d}{i} \equiv \binom{1}{0} \cdot \binom{0}{i} \equiv 0 \pmod{d}$，即 $B(x)$ 的系数除了 $[x^{d-1}]B(x)$ 以外其余均是 d 的倍数。那么若迭代前 $C(x)$ 的系数均为 d^i 的倍数的话，迭代后的系数将会为 d^{i+1} 的倍数，因此经过 $a - 1$ 轮迭代后 $C(x)$ 的系数都会为 0。

总时间复杂度计算与**算法 4.3** 类似，为 $O(m \cdot (log_2 N + log_2 M \cdot d) + (log_2 M)^2 \cdot d^2)$。结合**算法 4.1** 和**算法 4.3** 可以通过第三、第五到第九个测试包。

3.4.5 算法 4.5：满分做法

将 M 分解质因数得 $M = \Pi_{i=1}^{k} p_i^{q_i}$，对每个 $p_i^{q_i}$ 计算取模意义下的值。问题变成如何对一个质数的次幂 p^q 进行求解。

当 $gcd(d,\ p) = 1$ 时，仍然使用**算法 4.1**。

当 $gcd(d,\ p) > 1$ 即 $d \equiv 0 \pmod{p}$ 时，仍然是想通过对 (31) 迭代使得右式 $C(x)$ 的系数在取模意义下为 0。当 $i \equiv 0 \pmod{p}$ 时，由卢卡斯定理 $\binom{d}{i} \equiv \binom{\frac{d}{p}}{\frac{i}{p}} \cdot \binom{0}{0} \equiv \binom{\frac{d}{p}}{\frac{i}{p}} \pmod{p}$，其结果不一定为 0，因此不能直接套用**算法 4.4** 的解法了。

用 (31) 迭代失败的原因主要是其迭代式子存在一个次数比 i 高的 $C(x)$ 项，其系数不一定为 p 的倍数，导致迭代后，新产生的系数不一定能使得所有系数都是 p 的更高次幂的倍数。如果我们选择系数不是 p 的倍数的 $C(x)$ 项中次数最高的那一项放在等式左侧，即找到最小的 t 使得 $[x^t]B(x) \not\equiv 0 \pmod{p}$，那么 (26) 可以写成：

$$\begin{aligned}([x^t]B(x)) \cdot ([x^i]C(x)) \ =\ & [x^{i+t}]A(x) \\ & - \sum_{j=0}^{t-1} ([x^j]B(x)) \cdot ([x^{i+t-j}]C(x)) \\ & - \sum_{j=t+1}^{d-1} ([x^j]B(x)) \cdot ([x^{i+t-j}]C(x))\end{aligned} \quad , \forall\ i \ge 0 \tag{32}$$

其中当 $j < t$ 时 $[x^j]B(x)$ 为 p 的倍数，当 $j > t$ 时则不一定。

只要把 (32) 中次数比 i 高的 $C(x)$ 项的系数在模 p^q 意义下变成 0，就可以与**算法 4.1** 类似地按照从小到大的顺序求解每一个需要的 $[x^i]C(x)$ 了。因此，只需要对次数比 i 高的 $C(x)$ 项迭代。如果直接用 (32) 迭代，在对次数为 j 的项进行迭代的时候，有可能向次数为 k, $(i < k < j)$ 的项产生一个非 p 倍数的系数，这样就不一定能保证有限轮迭代之后可以结束。而这样的问题只会单向产生（向次数较低的项产生），那么很自然的想法是，假设当前次数比 i 高的每一个 $C(x)$ 项的系数都是 p^a 的倍数，我们每次找到次数最高的系数不是 p^{a+1} 的倍数的 $C(x)$ 项，对其进行迭代，直到所有 $C(x)$ 项的系数都是 p^{a+1} 的倍数为止，我们称这个过程叫做一轮，不难发现，一轮操作中需要迭代的项的次数是单调变低的，故一轮操作时间复杂度为 $O(l)$, 其中 l 为操作前式子右侧 $C(x)$ 项个数。那么进行 $q-1$ 轮操

作后，次数比 i 高的项的系数都变成了 0。在迭代中 $[x^i]C(x)$ 由于每次加上的都是一个为 p 的倍数的数，所以其仍然与 p^q 互质，逆元仍然存在，只用将等式两边同时乘上这个逆元就得到了 $[x^i]C(x)$ 关于 $A(x)$ 以及 $[x^j]C(x),\ (j<i)$ 的转移。

计算 $A(x)$ 时间复杂度为 $O(m\ \cdot\ (log_2N\ +\ log_2M))$；计算对所有 $p^q,\ (gcd(d,\ p)\ =\ 1)$ 乘积取模的值时，可以先算出乘积 T，使用**算法 4.1** 计算对 T 取模意义下的值，时间复杂度为 $O(md)$；计算对所有 $p^q,\ (gcd(d,\ p)\ \neq\ 1)$ 乘积取模的值时，需要对每个 p^q 分别进行计算，对于每一个 p^q 时间复杂度为 $O(mdq\ +\ d^2q^2)$；用中国剩余定理算法计算最终答案时间复杂度为 $O(m\ \cdot\ log_2M^2)$。故总时间复杂度为 $O(m\ \cdot\ (log_2N\ +\ d\cdot log_2M)\ +\ d^2\cdot(log_2M)^2)$，可以通过所有测试点。

4 数据生成方式

容易发现输入的 n 与 m 是否随机和本题并无太大关联，因此 n 和 m 均为随机生成。

d 和 M 只与质因数分解有关，于是本人通过在满足部分测试包特殊限制的前提下，根据质因数组成方式的不同（譬如质数、单个质数的幂次、1、多个质数乘积等）进行生成。

5 命题思路

命制此题的思路来源于 tourist 在 VK Cup 2017 Round 3 中出的 F 题 Test Data Generation，本人在做那题时首先将问题转化成了求 $\sum_{i=0}^{n}\ \sum_{j=0}^{m}\ \binom{2i}{2j}\ (\mathrm{mod}\ M)$ 的形式，之后想出了**算法 4.2**，用其过了那题，而官方题解所给出的与**算法 2** 十分类似。我的做法在单次求解上述式子的值时优于题解，但在那题中需要计算 n 取值为 $N,\ \frac{N}{2},\ \frac{N}{4},\ \cdots$ 式子的值，这就导致了我的算法需要做 $log_2\ N$ 次，而倍增算法可以同时处理 $log_2\ N$ 个询问，故时间复杂度一样。但我的做法仍然在常数方面优于题解，因此截至 2019 年 3 月，我的提交在那题中仍为运行时间最短的提交。

多次询问不能体现出我算法的优越性，于是我便首先将题目修改成了只询问一个 n 的问题。

然而仅仅是对 $f(j)$ 求和可以通过插值等技巧解决，于是便又将题目修改成求每个 $f(j)$ 的值，为了避免输出量过大，采用了输出异或和的方式。

在想出了 $d\ =\ 2$ 的做法后，我又想尝试解决 d 为任意值的问题。我起初仍然想像**算法 4.2**、**算法 4.3** 和**算法 4.4** 那样不断迭代，但是由于存在非 p 的倍数的系数，我并不知道迭代过程什么时候结束。于是便有了找到最小的 i 使得 $\binom{d}{i}$ 不为 p 的倍数，对 $C(x)$ 中次数比它高的部分进行迭代的想法，之后又观察到了其只会向次数较低的 $C(x)$ 项贡献非 p 倍数的系数，于是就想到了从右到左依次迭代的方法，即**算法 4.5**，也就得到了本题。

6 考点、难点以及区分度设计

本题考查了二项式定理、卢卡斯定理、中国剩余定理等关于组合数学、数论方面的知识；形式幂级数、等比数列求和、取模意义下组合数求值等技巧；扩展欧几里得、快速傅里叶变换等算法。

如果知道组合数、取模、异或和等知识就能做出第一个测试包。此部分意在区分选手是否知道组合数、取模、异或和等基础知识，以及对题意的理解。

通过组合数的推导可以做出 $d = 1$ 的测试包。此部分意在区分选手的数学推导能力。

要做出 N 较大的测试包需要想到二项式定理，通过形式幂级数将问题转化成多项式求和，此为本题的第一个难点。在此基础上不难得出**算法 2**，此部分意在区分选手是否知晓二项式定理、倍增等算法以及能否熟练地使用形式幂级数来帮助解决问题，同时还考察了对多项式乘法的优化。

在想到多项式求和的基础上，观察求和多项式的每一项均为 $(x+1)^d$ 的幂，联想数学中等比数列求和的技巧，将问题转化为多项式除法，此为本题第二个难点。在此基础上如果理解多项式除法则不难得出 $gcd(d, M) = 1$ 问题的解法，此部分意在区分选手对求和技巧以及对多项式除法的理解。

$d = 2, 4$ 以及 d 为质数的测试包意在引导选手解决较为特殊的问题，通过中国剩余定理 (CRT) 将问题分成模数与 d 互质以及模数为 d 的质因数次幂两个问题，其中模数与 d 不互质的情况为本题的第三个难点。此部分意在区分选手对于中国剩余定理 (CRT) 和迭代算法的掌握。

会了 d 较为特殊的情况之后，通过分析之前的迭代算法为何不能进一步解决一般情况的问题，挖掘转移式的性质，从而得到最终的正解，此为本题最后的难点。此部分意在区分选手是否对转移式以及之前做法有深入的理解。

在实际互测中，有 6 位选手提交了此题，除最后一个测试包外其它测试包均有选手通过，具体通过情况如下：

测试包编号	1	2	3	4	5	6	7	8	9	10
通过人数	4	1	3	1	3	2	1	1	1	0

7 总结

总的来说，本题所需知识点、算法难度不高，大部分算法难度与 NOI 级别相近；正解的过程也并不多，代码量并不大，但思维难度不小，思路新颖，具有很大的启发意义。作为一道 OI 题，在部分分的设计上鼓励了多种方法解决此问题，并且引导选手从特殊情况开始一步步走向正解。

除此之外，近年来计算模意义下答案的问题在编程竞赛中越来越常见，也出现了不少解决此类问题的算法，但是其中一些算法往往仅能在模数为质数的情况下发挥较大作用，譬如 Berlekamp-Massey 算法。此类算法在模数为合数时，由于乘法逆元不一定存在，导致在

算法中不一定能进行除法操作，从而引发一系列问题。常见的处理办法有：使用诸如 double 等实数类型存储变量；使用高精度将有理数转化为分数的形式进行存储。而前者可能会存在精度误差，当运算较多时会导致结果出错，后者虽然保证了结果的正确性，但是时间复杂度太高，有时会难以接受。本题中采用的思想为先对模数分解质因数，将其表示成若干个不同的质数的幂的乘积的形式，如果能对每一个质数的幂分别进行求解，就可以使用中国剩余定理算法得到最终的答案。求解对质数的幂 p^q 取模意义下的答案时，往往可以尝试先解决更低的幂的情况，譬如 p^{q-1}，再在其基础上修改得到最终答案。这种思想也在其它算法中有所体现，譬如 Reeds–Sloane 算法。本文也希望起到抛砖引玉的效果，引发大家对于一些计算模合数意义下答案的问题的思考。

8 鸣谢

感谢中国计算机学会提供学习和交流的平台。
感谢父母和老师多年来的关心和指导。
感谢温州中学孔朝哲同学验题并为本文审稿。
感谢其他同学与我交流讨论。

参考文献

[1] 毛啸，《再探快速傅里叶变换》，国家集训队 2016 论文集。

[2] 汪乐平，《生成函数，多项式算法与图的计数》，WC2019 第一课堂课件。

[3] wikipedia, Binomial coefficient, https://en.wikipedia.org/wiki/Binomial_coefficient。

[4] wikipedia, Chinese remainder theorem, https://en.wikipedia.org/wiki/Chinese_remainder_theorem。

[5] wikipedia, Lucas's theorem, https://en.wikipedia.org/wiki/Lucas%27s_theorem。

[6] wikipedia, Reeds–Sloane algorithm, https://en.wikipedia.org/wiki/Reeds–Sloane_algorithm。

浅谈一类简洁数据结构

成都市第七中学 王思齐

摘 要

简洁数据结构是一种空间常数较小的数据结构。本文介绍了一类简洁的数据结构，并给出了一些他们的应用。

1 前言

如何空间高效的表示数据与查询数据，是计算机科学中一个重要的问题。对于表示数据，最常见的方法是进行压缩，但是在压缩之后，对原始数据的查询通常会更加困难。而如果直接存储原始数据，再用普通的数据结构维护，空间上则不够高效。

针对这些问题，空间高效的数据结构应运而生。常见的有下面几种：（设原数据需要 n bits 表示）

- 压缩数据结构（Compressed data structure），使用的存储空间取决于特定数据，通常和所表示的数据的信息熵有关。
- 简洁数据结构（Succinct data structure），使用 $n + o(n)$ bits。
- 隐式数据结构（Implicit data structure），使用 $n + O(1)$ bits。
- 紧凑数据结构（Compact data structure），使用 $O(n)$ bits。

本文介绍了一种简洁的处理 01 串中 *rank* 和 *select* 问题的数据结构，然后以这种数据结构为基础，引入了处理树的数据结构和小波树。

2 01 串

考虑给出一个长为 n 的 01 串 S，要求支持两种操作：

- $rank_v(k)$：求位置 k 及之前有多少个 v。

- $select_v(k)$：求第 k 个 v 的位置。

显然，如果某个位置 i 的值是 v，那么 $select_v(rank_v(i)) = i$。

这个问题显然有空间 $O(n\log n)$，时间 $O(1)$ 的前缀和做法以及空间 n，时间 $O(n)$ 的暴力做法。但是更优的方法可以做到空间 $n + O(\frac{n\log\log n}{\log n})$，时间 $O(1)$。

下面首先介绍一种空间 $O(n)$，时间 $O(1)$ 的做法，再介绍上述更优的做法。

2.1 空间 $O(n)$，时间 $O(1)$ 的做法

对于 *rank* 操作，首先把 S 以 $\frac{\log n}{2}$ 为大小分块，然后存储每个块尾位置的前缀和，这部分需要 $\frac{n}{\frac{\log n}{2}} \cdot \log n = 2n$ 空间。

对于块内，可以直接存储所有长度不超过 $\frac{\log n}{2}$ 的 01 串的和，需要 $\sqrt{n} \cdot \log\log n$ 空间。

查询时可以先查出所在连续块部分的和，不满一块的部分可以通过两次查表求出。

对于 $select_1$ 操作，首先可以对询问的值按 $\log n$ 分为大块，即求出每个 $select_1(k \cdot \log n)$ 的结果，这部分需要 $\frac{n}{\log n} \cdot \log n = n$ 空间。

如果某个大块对应原串所在的区间长度大于 $\log^2 n$，那么这样的块个数不会超过 $\frac{n}{\log^2 n}$，可以直接存储所有这样的块的答案，共需要 $O(\frac{n}{\log^2 n} \cdot \log n \cdot \log n) = O(n)$ 空间。

对于其他的大块，在块内继续按照 $\log\log n$ 分为小块，存储每个位置在这一小块内的答案。由于对应到原串的区间长度不超过 $\log^2 n$，表示每个答案也只需要 $O(\log\log n)$ 空间。那么这部分一共需要 $O(\frac{n}{\log n} \cdot \frac{\log n}{\log\log n} \cdot \log\log n) = O(n)$ 空间。

对于长度不超过 $\frac{\log n}{2}$ 的区间，可以直接存储所有这样的区间的所有 *select* 操作的答案，共需要 $O(\sqrt{n} \cdot \log n \cdot \log\log n)$ 空间。

两次分块之后，如果某个小块对应原串所在的区间长度不超过 $\frac{\log n}{2}$，可以直接查上面的表；否则这样的块个数显然是 $O(\frac{n}{\log n})$，可以直接存储这个块所有位置在所在 $\log n$ 大块中的答案，需要 $O(\frac{n\cdot(\log\log n)^2}{\log n})$ 空间。

那么最终一共需要 $O(n + \sqrt{n} \cdot \log\log n + \sqrt{n} \cdot \log n \cdot \log\log n + \frac{n\cdot(\log\log n)^2}{\log n}) = O(n)$ 空间，而每个操作的分类讨论次数是固定的，所以均为 $O(1)$ 时间。

2.2 空间 $n + O(\frac{n\log\log n}{\log n})$，时间 $O(1)$ 的做法

首先把 S 以 $\log^2 n$ 为大小分块，然后存储每个块尾位置的前缀和，这样共需要 $\frac{n}{\log^2 n} \cdot \log n = \frac{n}{\log n}$ 空间。

接下来再把每个块分为大小为 $\log n$ 的小块，然后存储每个小块尾到大块头的区间和，这样一共会存储 $\frac{n}{\log n}$ 个值，每个值大小为 $\log^2 n$，共需要 $\frac{n\cdot\log\log n}{\log n}$ 空间。

最后再用一个数组存储所有长度不超过 $\frac{\log n}{2}$ 的 01 串的和，需要 $\sqrt{n} \cdot \log\log n$ 空间。

对于 *rank* 操作，首先定位 k 所在的大块，然后定位 k 所在的小块，求出前面两个前缀和之后，最后不满 $\log n$ 的部分可以通过两次查表求出。

对于 *select* 操作，可以类似的采用多次分块的方法，具体可以参考 [3]，此处由于篇幅所限，不再阐述。

2.3 实现

前面两种做法在复杂度上取得了较好的成果，但是实际实现还依赖于常数的大小。

01 串可以直接每 64 位用一个 64 位整数储存。

对于 *rank* 操作，可以以 64 的倍数为大小分块，存储块间前缀和。不满一块的部分可以直接调用 CPU 指令计算。可以通过调节块大小达到时空平衡。

对于 *select* 操作，一种方法是重用 *rank* 操作的表，在里面先二分，然后再处理不满一块的部分。这种方法时间复杂度较高，而空间使用较少。

另一种方法和 2.1 中的类似，但是为了减小询问常数，只进行一次分块。对于映射到原串长度较短的，可以直接暴力计算，较长的则仍然存储答案。

3 树

本节会介绍三种表示树的简洁数据结构以及他们的应用。这三种数据结构分别是 BP（Balanced Parentheses）、LOUDS（Level-ordered Unary Degree Sequence）和 DFUDS（Depth-first unary degree sequence）。

这三种数据结构都把树编码为长为 $2n + O(1)$ 的 01 串或括号序列，然后再用前文提到的 01 串数据结构，或是 [4] 中提到的括号序数据结构去维护。

这三种数据结构在某些操作均有 $O(1)$ 的时间复杂度，而在另一些操作上则各不相同。而有另外一些数据结构，可以同样在 $2n + o(n)$ 的空间内进行这些操作、以及更多的操作，具体可以参考 [6,7,8]，这里就不再赘述。

3.1 BP

BP 即用括号表示树。

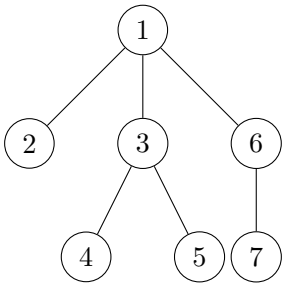


图 10: 一棵树

如图 10 所对应的括号序是：

括号序	(( ) (( ) ( )) (( )))
对应点的编号	1 2 2 3 4 4 5 5 3 6 7 7 6 1

当去掉根节点的两个括号之后，显然，长为 $2n-2$ 的平衡括号序列（两种括号个数相同）和 n 个点的树是一一对应的。而长为 $2n-2$ 的括号序列个数 $C_{n-1}=\frac{1}{n}\binom{2n-2}{n-1}$。根据斯特林公式，$\log C_n$ 趋近于 $2n$。那么使用 $2n+o(n)$ bit 的空间是简洁的。

根据 [4] 中提到的方法，可以在 $O(1)$ 时间，$2n+o(n)$ 空间内求出一个长为 $2n$ 的括号序列中，某个左括号对应的右括号、某个右括号对应的左括号、某个括号外层的括号。

使用这些操作，可以 $O(1)$ 的实现找第一个儿子、最后一个儿子、兄弟节点、父亲，以及求子树大小和点的深度等操作。

3.2 LOUDS

LOUDS 即为按照 bfs 序将度数编码为一进制。

首先给树添加一个新的根节点，其儿子只包含原树的根。然后将这棵树按照 bfs 序编号，并把每个节点的编码依次连接，得到这棵树的编码。每个节点的编码取决于它的儿子个数，有 x 个儿子则是 x 个 1 之后加上一个 0。

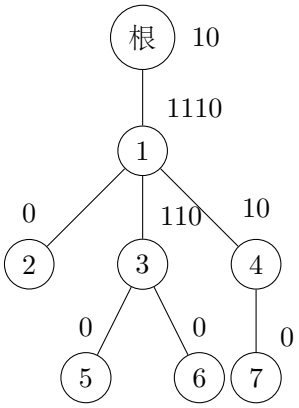


图 11: 一棵树及每个点的编码

图 11 所示的树的编码就是 10.1110.0.110.10.0.0.0（点只起到方便读者查看的作用）。每个节点恰好对应编码中的一个 1。而节点编号和编码中的位置的对应关系如下：

- $v = rank_1(i)$：求位置 i 对应的节点编号 v。
- $i = select_1(v)$：求编号为 v 的节点所在的位置 i。

而树上的节点相关操作也可以容易的实现：

- $firstchild(i) = select_0(rank_1(i)) + 1$：求位置 i 对应的节点的第一个儿子所在的位置。
- $lastchild(i) = select_0(rank_1(i) + 1) - 1$：求位置 i 对应的节点的最后一个儿子所在的位置。
- $parent(i) = select_1(rank_0(i))$：求位置 i 对应的节点的父亲所在的位置。
- $child(i, k) = firstchild(i) + k - 1$：求位置 i 对应的节点的第 k 个儿子所在的位置。
- $degree(i) = lastchild(i) - firstchild(i) + 1$：求位置 i 对应的节点的儿子个数。

3.3 DFUDS

DFUDS 即为按照 dfs 序将度数编码为一进制。
一棵树的 DFUDS 按下面的方式给出：

- 对于只有一个节点的树，其编码为 ()。

- 对于超过一个节点的树，首先去掉其所有子树的编码中，最左边的左括号。然后其编码为 ((...((()，其中左括号的个数为儿子个数 +1，后面再依次接上每个儿子的编码。

如果去掉最左边的左括号，那么每个点新加入的编码和 LOUDS 相同（1 对应左括号，0 对应右括号）。

如图 10 中的树对应的编码为 (.(((().).(().).).().)（每个点的编码可见图 11）。

不难发现，DFUDS 一定是一个合法的括号序列。

在实际构建时，可以直接 DFS 一遍，然后依次加入每个点的儿子个数个左括号和一个右括号。最后再添上最前面的左括号。具体算法见下。

算法 1 求一棵树的 DFUDS 编码

Require: 树 T

Ensure: 编码 Result

```
Result ← ""
procedure Dfs(x)
    Result ← Result + "(" × x.children_count + ")"
    for y ∈ x.children do
        Dfs(y)
    end for
end procedure
Dfs(T.root)
Result ← "(" + Result
```

为了快速进行树上操作，我们需要使用下面这些 2.2 中提到的操作，以及 [4] 中的对括号序的操作进行维护。

- $rank_v(k)$：求位置 k 及之前有多少个 v，其中 v 为 *open* 或 *close*，表示左括号和右括号。
- $select_v(k)$：求第 k 个 v 的位置。
- $findclose(k)$：求位置 k 对应的右括号，保证位置 k 是左括号。
- $findopen(k)$：求位置 k 对应的左括号，保证位置 k 是右括号。
- $enclose(k)$：求位置 k 外层的左括号。

在 DFUDS 中，我们把每个节点和每个右括号一一对应，对应关系如下：

- $v = rank_{close}(i)$：求位置 i 对应的节点编号 v。

- $i = select_{close}(v)$：求编号为 v 的节点所在的位置 i。

容易看出，每个节点的编号就是它在 dfs 序上的位置。
各种操作的实现如下：

- $pre(k) = select_{close}(rank_{close}(k) - 1)$：求前一个右括号的位置。
- $degree(k) = k - select_{close}(rank_{close}(k) - 1)$：求位置 k 对应的节点的儿子个数。
- $child(i,k) = select_{close}(rank_{close}(findclose(k - i)) + 1)$：求位置 k 对应的节点的第 i 个儿子。
- $parent(k) = select_{close}(rank_{close}(findopen(pre(k))) + 1)$：求位置 k 对应的节点的父亲。
- $subtreesize(k) = (findclose(enclose(pre(k))) - pre(k))/2 + 1$：求位置 k 的子树大小，这个操作在 k 为根时需要特判。

3.4 应用

在很多需要表示树的场景下，都可以使用这三种数据结构优化。

例如 Trie，Trie 构建时需要使用指针结构，但是在构建完之后可以不用指针，只保留树结构。下面给出两种优化方法。

设字符集为 $\sum$，Trie 的节点数为 n。

方法一为，对于 Trie 的每个节点，可以添加一些虚节点，将其子节点补满到 $|\sum|$ 个。然后直接对这个树编码，可以根据一个节点的儿子个数来判断其是否是虚节点。

这个做法的空间复杂度为 $O(n|\sum|)$，查找某个字符对应的子节点的时间复杂度为 $O(1)$。

方法二为，直接把原 Trie 编码，然后记录下每个节点对应的字符。查找某个字符对应的子节点时，可以二分或者遍历，时间复杂度分别为 $O(\log|\sum|)$ 和 $O(\frac{|\sum|}{w})$，其中 w 为计算机位宽，后者在 $|\sum|$ 较小时可近似为 $O(1)$。

这个做法的空间复杂度为 $O(n)$。

4 小波树

考虑给出一个长为 n 的串 S，字符集为 $\sum$，要求支持两种操作：

- $rank_v(k)$：求位置 k 及之前有多少个 v。
- $select_v(k)$：求第 k 个 v 的位置。

小波树（Wavelet Tree）可以在 $(1+o(1))n\log|\sum|$ 的空间、单次询问 $O(\log|\sum|)$ 的时间下实现这两个操作。

对于一个串，我们把它的字符集分为两部分（通常直接按照字符大小均分），并把第一部分中的字符替换为 0，第二部分中的字符替换为 1。然后将原串中属于这两部分的字符依次拿出，得到两个新串，再对这两个新串递归建树，直到字符集大小为 1 时停止。

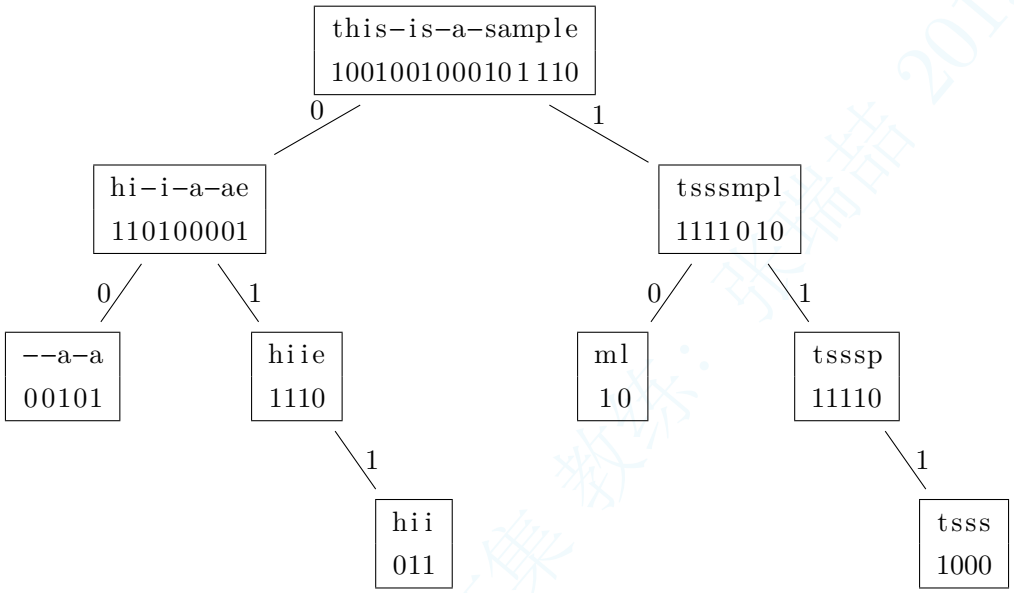


图 12: 小波树的示例

如图 12 是一个原串为 this-is-a-sample，字符集为 $\{-, \mathrm{a}, \mathrm{e}, \mathrm{h}, \mathrm{i}, \mathrm{l}, \mathrm{m}, \mathrm{p}, \mathrm{s}, \mathrm{t}\}$ 的小波树。

这样建出的树的节点个数为 $|\sum|-1$，最大深度为 $\lceil\log|\sum|\rceil$，所有节点维护的 01 串总长不超过 $n\lceil\log|\sum|\rceil$。

在每个节点的 01 串上快速的维护 $rank_0, rank_1, select_0, select_1$ 操作之后，我们要求的操作可以这样进行：

- 对于 $rank_v(k)$ 操作，可以找到当前节点上，v 是被划为了 0 还是 1。设 v 被划为了 x，可以求出当前节点的 01 串的 $rank_x(k)$，最后可以递归到该侧子节点继续询问。

  如在图 12 的根节点询问 $rank_a(10)$，可以发现 a 被划分到 0，而在该 01 串上查询 $rank_0(10)=7$，所以可以递归到左儿子继续查询 $rank_a(7)$。

- 对于 $select_v(k)$ 操作，可以和 $rank$ 操作相反的处理。首先找到 v 所在的最底层节点，并找到该节点上 v 被划分为的值 x，然后求出该节点的 01 串的 $select_x(k)$，最后递归到父亲节点处理。

如在图 12 中询问 $select_a(1)$，可以发现 a 所在的最底层节点是 −−a−a，而在该节点询问 $select_1(1) = 3$，那么可以递归为在该节点的父亲查询 $select_0(3) = 6$，继续递归到父亲节点为 $select_0(6) = 9$，最终我们就查询到了 $select_a(1) = 9$。

下面是这两个操作的伪代码：

算法 2 小波树的 *rank* 和 *select* 操作

Require: 根节点 T，字符集为 $1\ |\sum|$

```
function Rank(T, v, k, l, r)
  if l=r then
    return k
  end if
  mid ← (l+r)/2
  if v ≤ mid then
    return Rank(T.left, v, T.rank_0(k), l, mid)
  else
    return Rank(T.right, v, T.rank_1(k), mid + 1, r)
  end if
end function

function Select(T, v, k, l, r)
  if l=r then
    return k
  end if
  mid ← (l+r)/2
  if v ≤ mid then
    return T.select_0(Select(T.left, v, k, l, mid))
  else
    return T.select_1(Select(T.right, v, k, mid + 1, r))
  end if
end function

function rank(v, k)
  return Rank(T, v, k, 1, |∑|)
end function

function select(v, k)
```

30: **return** Select$(T, v, k, 1, |\sum|)$
31: **end function**

利用小波树，还可以完成下面这些操作：

- $quantile(l, r, k)$：求 $[l, r]$ 的第 k 大值。可以求出 $[l, r]$ 中 0 的个数，然后确定第 k 大值是被划分了 0 还是 1，最后递归到该侧子节点继续询问。
- $rangefreq(l, r, x, y)$：求 $[l, r]$ 中，值域在 $[x, y]$ 中的个数。可以类似线段树的递归询问。
- $nextvalue(l, r, x)$：求 $[l, r]$ 中，大于 x 的最小值。可以先确定 x 是被划分了 0 还是 1，然后递归到该侧子节点处理。

这些操作都可以在 $\log|\sum|$ 的时间内完成。而更多的同时在区间和值域做出限制的问题也可以轻松实现。

5 总结

本文由 01 串引入了处理树的数据结构和小波树。对于 01 串和树，在本文提到的数据结构之外，还有更多复杂度更低、或者能实现更多功能的数据结构。

这三种数据结构均有广泛的应用，尤其是在压缩算法、数据库、字符串索引等场景应用较多。

在信息学竞赛中，由于通常内存限制较大，就不需要使用这些数据结构；而如果要针对性的考察这些数据结构，现有的评测技术又难以精确的限制内存，所以这些数据结构目前考察较少。但是这些数据结构在常数优化时还是有不少的用途；而如果要针对性考察，可以使用自定义的语言来精确的限制内存（如在 IOI2019 集训中出现的 Potato 语言）。

感谢

感谢中国计算机学会提供学习和交流的平台。
感谢成都七中张君亮、蔺洋老师多年以来的关心和指导。
感谢为本文审稿和提出建议的老师、同学们。
感谢其他帮助过我的老师、同学们。

参考文献

[1] G. Jacobson, Succinct Static Data Structures, Tech. report CMU-CS-89-112, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1989.

[2] G. Jacobson, Space-efficient static trees and graphs, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, pp. 549–554, 1989.

[3] A. Golynski, Optimal lower bounds for rank and select indexes, Theoret. Comput. Sci., 387, pp. 348–359, 2007.

[4] J. I. Munro and V. Raman, Succinct representation of balanced parentheses, static trees and planar graphs, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, pp. 118–126, 1997.

[5] D. Benoit, E. D. Demaine, J. I. Munro, and V. Raman, Representing trees of higher degree, in Proceedings of the 6th Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Comput. Sci. 1663, Springer-Verlag, Berlin, pp. 169–180, 1999.

[6] R. F. Geary, R. Raman and V. Raman. Succinct ordinal trees with level-ancestor queries. In Proc. 15th ACM-SIAM SODA, pp. 1–10, 2004.

[7] M. He, J.I. Munro, S.S. Rao, Succinct ordinal trees based on tree covering, in: Proceedings of the International Conference on Automata, Language and Programming, LNCS, vol. 4596, pp. 509–520, 2007.

[8] A. Farzan, R. Raman, and S. S. Rao. Universal succinct representations of trees? In Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP). 451–462, 2009.

[9] H. Zhang, D. G. Andersen, M. Kaminsky, A. Pavlo, H. Lim, V. Leis, and K. Keeton. SuRF: Practical Range Query Filtering with Fast Succinct Tries. In Proceedings of the 2018 International Conference on Management of Data (SIGMOD). ACM, 323 – 336, 2018.

[10] R. Grossi, A. Gupta, and J. S. Vitter, High-order entropy-compressed text indexes, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, pp. 841–850, 2003.

[11] M. Burrows and D.J. Wheeler, A block-sorting lossless data compression algorithm. Digital SRC Reports 124, 1994.

子串周期查询问题的相关算法及其应用

南京外国语学校 陈孙立

摘 要

本文介绍了子串周期查询问题的解决方法，以及解决这个问题所使用的到的一些结论和字符串工具——基本子串字典 (Dictionary of Basic Factors)，接着介绍了前述结论和工具的一些应用。

本文第二节引入了一些基础定义和记号。

本文第三节证明了三个重要引理，引入基本子串字典。

本文第四节描述了解决子串周期查询问题的方法。

本文第五节介绍了基本子串字典的一些应用。

1 引言

现阶段计算机科学界对字符串的研究越来越注重于对周期，即“重复模式”的性质的探究。这一方向也具有很大的现实意义，如语音识别、DNA 匹配等，它们都需要对大量重复的字符串模式的精巧刻画。

子串周期查询问题是一个描述简单但是做法复杂的经典问题，由此还可以延伸到应用更广的内部模式匹配 (Internal Pattern Matching) 问题。本文着重介绍了它的解决办法，以及一些扩展。

2 记号、基础定义与问题描述

2.1 记号与定义

令 S 为一个字符串，$|S|$ 为其长度，记 $S[i]$ 为 S 的第 i 个字符，$S[l,r]$ 为把 S 的第 l 个字符到第 r 个字符取出组成的新字符串，若 $l > r$ 则 $S[l,r] = \emptyset$ 。

记 $pref(S,x) = S[1 \ldots x]$ 为 S 的长度为 x 的前缀，类似地，$suf(S,x) = S[|S|-x+1 \ldots |S|]$ 为 S 的长度为 x 的后缀。

若 $T[x \ldots x+|S|-1] = S$ ，我们说 S 在 T 中的 x 位置出现，x 是 S 在 T 中的出现位置或匹配位置。

定义 1：若串 S 和整数 x 满足 $1 \le x \le |S|$，且对于所有的 $1 \le i \le |S| - x$，都有 $S[i] = S[i+x]$，则称 x 为串 S 的周期或周期长度。特别地，若 x 还是 $|S|$ 的因数，则称 x 是 S 的整周期，称 S 是整周期串。

显然，若 x 是 S 的周期，则 $kx(k \in \mathbb{Z}^+, k \cdot |x| \le |S|)$ 也是 S 的周期，x 也是 $pref(S,y), y \ge x$ 的周期。

定义 2：$per(S)$ 表示所有 S 的周期组成的集合，$minper(S)$ 表示 $per(S)$ 中的最小值。不难发现对于非空串 S，$|S|$ 一定是 S 的周期，因此 $|per(S)| > 0$。

定义 3：若串 S 和整数 x 满足 $0 \le x < |S|$，且满足 $pref(S,x) = suf(S,x)$，则称 $pref(S,x)$ 是 S 的 border。

推论 1：$|S| - x$ 是 S 的周期当且仅当 $pref(S,x)$ 是 S 的 border。

证明：根据 border 和字符串相等的定义可得：$pref(S,x)$ 是 S 的 border 当且仅当对于所有的 $1 \le i \le |pref(S,x)|$ 都有 $pref(S,x)[i] = suf(S,x)[i]$，也即 $S[i] = S[|S| - x + i]$，与周期的定义相符。 □

推论 2：若 p 是 S 的周期，且某个满足 $r - l + 1 \ge p$ 的子串 $S[l \ldots r]$ 有周期 q，并满足 $q \mid p$，则 q 也是 S 的周期。

此推论比较显然。

2.2 问题描述

给定一个字符串 S，以及若干次询问。每个询问给出 l, r 两个数，要求给出 $per(S[l \ldots r])$。

通过下文将要给出的结论，可以用一种简洁的方式去表示 $per(S[l \ldots r])$，花费 $O(\log n)$ 的空间，而不是最坏情况下的 $O(|S|)$，并且给出一个预处理需要 $O(n \log n)$ 的时间与空间，每次回答询问需要 $O(\log n)$ 的时间与空间的算法。

3 解决问题的准备工作

3.1 关于“重复”的一些性质

关于字符串周期有一个重要的引理，叫做 Periodicity Lemma。它本身的证明比较繁琐，下面我们证明它的弱化版本。

引理 1（Weak Periodicity Lemma）：若 $p, q \in per(S)$，且 $p + q \le |S|$，则 $\gcd(p,q) \in per(S)$。

证明：不妨假设 $p < q$。对于任意 $1 \le i \le |S|$，若 $p < i \le |S| - q + p$，则 $S[i] = S[i-p] = S[i+q-p]$，若 $i \le p$，则 $S[i] = S[i+q] = S[i+q-p]$，那么可得 $q - p$ 也是

$|S|$ 的周期。这里用到了周期的定义和 $p+q \le |S|$ 的条件。而又有 $p+(q-p) \le |S|$，所以可以用辗转相除法一直得到 $\gcd(p,q)$ 是 $|S|$ 的周期。 □

推论 3：若 $p = minper(S) \le \frac{|S|}{2}$，则对于任意 $x \in per(S), x+p \le |S|$，有 $p|x$。

证明：若 $p \nmid x$，根据**引理 1** 有 $\gcd(p,x) \in per(S)$，而 $p < x$，所以 $\gcd(p,x) < p$，与 $p = minper(S)$ 矛盾。 □

以下把 Weak Periodicity Lemma 简写为 WPL。

引理 2：若串 S,T 满足 $2|S| \ge |T|$，则 S 在 T 中的出现位置构成一个等差数列。

证明：若 S 在 T 中出现了一次或两次，则引理已证完，因此只需考虑 S 在 T 中至少出现了 3 次。

对于两次出现位置 $p < q$，由于 S 是 $T[p \dots q+|S|-1]$ 的 border，有 $q-p$ 是 $T[p \dots q+|S|-1]$ 的周期。同时根据 $1 \le p,q \le |T|-|S|+1$，有 $p+|S|-1 \ge |S| \ge |T|-|S| \ge q$，因此 $q-p < |S|$，则 $q-p$ 也是 S 的周期。

那么我们考虑 S 在 T 中的前两次出现位置 $p_1 < p_2$ 和某一次出现位置 $q > p_2$，有 $p_2-p_1, q-p_2 \in per(S)$。又有 $q-p_1 \le |T|-|S| \le |S|$，根据 WPL 可得 $r = \gcd(p_2-p_1, q-p_2) \in per(S)$。同时，可以发现 $r \le \min(p_2-p_1, q-p_2) \le \frac{q-p_1}{2} \le \frac{|S|}{2}$。令 $r' = minper(S)$，那么根据**推论 3** 有 $r' \mid r$。再根据**推论 2**，得到 r' 是 $T[p_1 \dots p_2+|S|-1]$ 的周期。假设 $r' < r$，根据前面的结论，S 在 p_1+r' 位置同样出现，和 p_2 是第二次出现位置矛盾。因此 $r = r' = minper(S)$。

至此，再使用一次**推论 3**，就可以得到 $r \mid q-p_2$。另一方面，若令 q 是 S 在 T 中的最后一次匹配位置，则任何满足 $x = p_1 + kr \le q, k \in \mathbb{Z}$ 的 x，根据上面的结论，都是 S 的出现位置。综合这两个结论，就证明了**引理 2**。 □

不仅如此，我们还得到了：若 S 在 T 中至少出现三次，则对应等差数列的公差为 $minper(S)$。

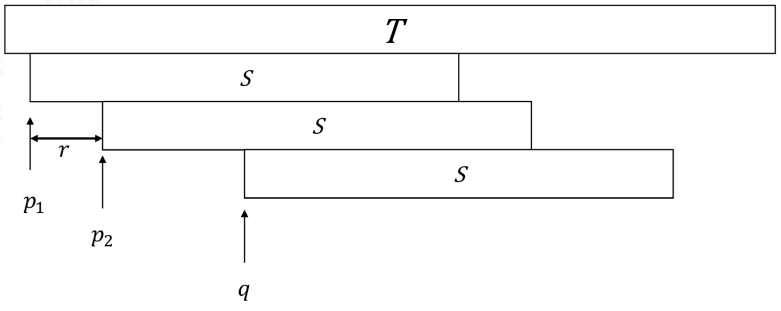


图 13: **引理 2** 示意图

引理 3：串 S 的所有不小于 $\frac{|S|}{2}$ 的 border 长度构成一个等差数列。

证明：只需证明 S 的所有不大于 $\frac{|S|}{2}$ 的周期构成一个等差数列，而这个结论可以由**推论 3** 直接得到。 □

至此，解决子串周期查询问题必须的三个的引理证明完毕。

3.2 数据结构：基本子串字典

解决子串周期查询问题还需要一种被称为基本子串字典的数据结构。

定义 4：一个串 S 的基本子串字典 (Dictionary of Basic Factors)，称为 $\mathbf{DBF}(S)$，它由 $\lfloor \log_2 |S| \rfloor + 1$ 个数组组成。具体地，第 t 个数组用 $Name_t$ 表示，长度为 $|S| - 2^t + 1$，满足 $Name_t(i) \le Name_t(j)$ 当且仅当 $S[i \dots i + 2^t - 1] \le S[j \dots j + 2^t - 1]$。一般情况下，可以再额外限制每个 $Name_t$ 中的值为正整数，且值域恰好为 $\{1, 2 \dots k\}$，其中 $k \in \mathbb{Z}$。

不难发现，数组 $Name_t$ 可以由 $Name_{t-1}$ 得到。因此要求出串 S 的基本子串字典，直接使用类似 Manber & Myers 的倍增求后缀数组算法即可，复杂度 $O(n \log n)$。

S	a	b	a	c	a	b	c	a
$Name_0$	1	2	1	3	1	2	3	1
$Name_1$	1	3	2	5	1	4	5	/
$Name_2$	1	4	3	5	2	/	/	/
$Name_3$	1	/	/	/	/	/	/	/

图 14: S = abacabca 的基本子串字典

仅有 $Name$ 数组可能还不够，一个简单的扩展是把每个 $Name_t$ 数组按值分开，即对于每个 x 维护一个有序表，记录 $Name_t[i] = x$ 的所有的 i。不难发现这一步也可以在构建基本子串字典的过程中完成，复杂度依然是 $O(n \log n)$。

后面将会看到，为了达到目前最优的一次查询 $O(\log n)$，还要对基本子串字典的再次扩展。

4 子串周期查询问题的解决

4.1 基本算法

定义 5：对于两个串 S,T 满足 $|S|=|T|$，定义 $PS(S,T)=\{k \mid pref(S,k)=suf(T,k)\}$，$LargePS(S,T)=\{k \mid k\in PS(S,T), k\geq \frac{|S|}{2}\}$。

推论 4：$LargePS(S,T)$ 的元素构成一个等差数列。

证明：考虑其中最大的元素 p，则对于任意 $q\in LargePS(S,T)$，有 $pref(S,q)$ 是 $pref(S,p)$ 的前缀，$suf(T,q)$ 是 $suf(T,p)$ 的后缀，因此 $pref(S,q)$ 是 $pref(S,p)$ 的 border。根据**引理 3**，其中的元素构成等差数列。 □

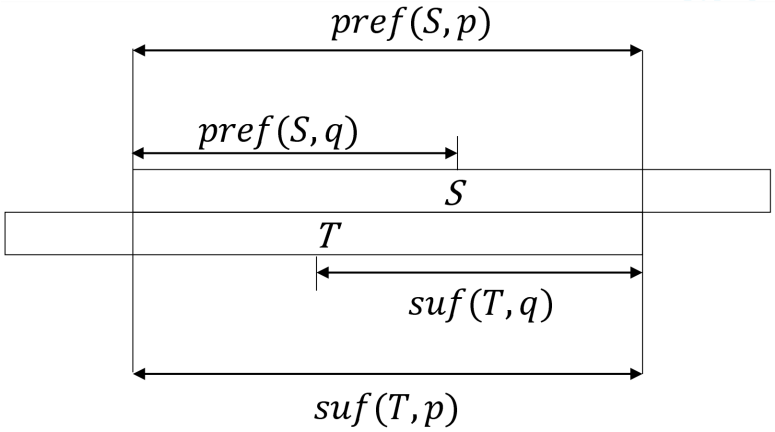


图 15: **推论 4** 示意图

推论 5：对于一个串 S 和整数 $k\leq \frac{|S|}{2}$，所有满足 $k\leq x<2k$ 的 S 的 border 长度 x 构成等差数列。

证明：上述 x 组成的集合恰好为 $LargePS(pref(S,2k),suf(S,2k))$。 □

有了这个结论，可以把 S 的 border 分为 $O(\log|S|)$ 类，第 i 类为长度在 $[2^{i-1},2^i)$ 中的所有 border，但是若 $|S|$ 不是 2 的幂次，最后一个区间要修改为 $[2^{\lfloor\log_2|S|\rfloor},|S|)$。在每个区间中的 border 长度均可以用一个等差数列表示，因此得到了一个空间 $O(\log|S|)$ 的表示 S 的 border 长度，也即 $per(S)$ 的方法。

对于 $[2^{i-1},2^i)$ 类型的区间，我们要求的就是 $LargePS(pref(|S|,2^i),suf(|S|,2^i))$，为此，使用下面的引理。引理的证明比较显然。

引理 4:对于 $2^{k-1}\leq x<|S|=|T|=2^k$，$x\in LargePS(S,T)$ 当且仅当 $suf(pref(S,x),2^{k-1})=suf(T,2^{k-1})$，且对应地 $pref(suf(T,x),2^{k-1})=pref(S,2^{k-1})$。

有了这个引理，我们先求出 $pref(|S|,2^{i-1})$ 在 $suf(|S|,2^i)$ 的所有出现位置，以及 $suf(|S|,2^{i-1})$ 在 $pref(|S|,2^i)$ 的所有出现位置，均使用等差数列表示。为了得到 $LargePS(pref(|S|,2^i),suf(|S|,2^i))$

，把前述的两个等差数列移位之后求交即可。

求 $pref(|S|, 2^{i-1})$ 在 $suf(|S|, 2^i)$ 的所有出现位置，即求出第一次、第二次和最后一次出现位置。由于这两个串都是母串的某一个子串，只要实现 $succ(S, i)$ 和 $pred(S, i)$ 操作，即找到 i 位置之后或之前（包括其本身）的 S 串的第一次出现位置。具体地，假设母串是 T，前者是 $S_1 = T[l \dots r]$，后者是 $S_2 = T[l' \dots r']$，其中 $2(r - l + 1) = r' - l' + 1$。令 $s = succ(S_1, l'), d = succ(S_1, s + 1) - s, t = pred(S_1, r' - |S_2| + 1)$，我们要求的等差数列即是首项是 s，末项是 t，公差是 d 的等差数列。

为此，我们构建出母串的基本子串字典，并在 $Name_{i-1}$ 中找到 $pref(|S|, 2^{i-1})$ 所在的有序表，对于每一次 $succ$ 或 $pred$ 查询，对这个有序表二分即可。

对于 $[2^{\lfloor \log_2 |S| \rfloor}, |S|)$ 类型的区间，和上面类似，不过在两个等差数列求交过程结束之后，还要对结果截取小于 $|S|$ 的一段。

至此，子串周期查询问题的基本算法框架就完成了。

4.2 等差数列求交及优化

我们用 (s, d, k) 三个参数描述一个等差数列 $s, s + d, \dots, s + (k - 1)d$。考虑求 (s_1, d_1, k_1) 和 (s_2, d_2, k_2) 两个等差数列的公共部分，即求出所有 $s_1 + xd_1 = s_2 + yd_2$ 的整数解，并适当截取某一段。对于一般的问题，只能使用扩展欧几里得算法解这个丢番图方程，复杂度 $O(\log \max(|d_1|, |d_2|))$。

然而，在本问题中，可以证明如下引理，使得我们可以 $O(1)$ 对等差数列求交。

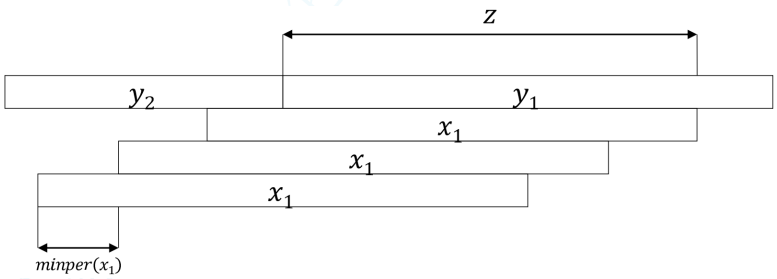


图 16: **引理 5** 示意图

引理 5：若四个字符串满足 $|x_1| = |y_1| \geq |x_2| = |y_2|$，且 x_1 在 y_2y_1 中至少匹配 3 次，y_1 在 x_1x_2 中至少匹配 3 次，则 $minper(x_1) = minper(y_1)$。

证明：用反证法，假设 $minper(x_1) \neq minper(y_1)$。若 $minper(x_1) > minper(y_1)$，考虑 x_1 在 y_2y_1 中最后一次匹配位置，令其与 y_1 的重叠部分为 z。根据**引理 2**，可知 x_1 在 y_2y_1 中相邻两次匹配的间距恰好为 $minper(x_1)$，又有 $|y_2| \leq |x_1|$，可得 $|y_2| + |z| \geq |x_1| + 2minper(x_1) \geq |y_2| + 2minper(x_1)$，即 $|z| \geq 2minper(x_1)$。

此时可以发现，z 有周期 $minper(x_1)$ 和 $minper(y_1)$ ，而 $|z| \geq 2minper(x_1) \geq minper(x_1)+minper(y_1)$ ，根据 WPL，z 有周期 $d = \gcd(minper(x_1), minper(y_1)) \mid minper(x_1)$ ，因此 d 也是 x_1 的周期，与 $minper(x_1)$ 产生了矛盾。

而当 $minper(x_1) < minper(y_1)$ 时产生矛盾的过程也几乎一致，只有“翻转”意义下的区别，故不再重复。上述矛盾表明 $minper(x_1) = minper(y_1)$ 。 □

若 $k_1 < 3$ 或 $k_2 < 3$ ，可以直接枚举对应等差数列里的所有元素并判断是否在另一个等差数列内，否则根据**引理 5**，$d_1 = d_2$ ，可以直接 $O(1)$ 地求出等差数列的交。

综合两部分，我们得到了本问题 $O(1)$ 求等差数列交的方法。

4.3 *succ* 和 *pred* 查询的优化

首先分析一下当前能达到的复杂度。预处理的时间已经达到了 $O(n \log n)$ ，子串周期查询分为 $O(\log |S|)$ 个 *LargePS* 询问，而每个 *LargePS* 询问需要 $O(1)$ 次 *succ* 和 *pred* 询问和一次等差数列求交。后一部分在上一节中已经优化为 $O(1)$ ，然而前半部分仍需要二分，每次 $O(\log |S|)$ ，总复杂度达到每次查询 $O(\log^2 n)$ 。同时，空间复杂度为 $O(n \log n)$ 。

如果考虑真正实现 *succ* 和 *pred* 功能，似乎只有对下标列表二分一种可行的方案，但是在子串周期查询中，我们用到它们是为了求出现位置的等差数列，似乎有些大材小用。因此考虑从问题本身出发，找到这两个操作的替代品。

在这个问题中，*succ* 和 *pred* 并不需要被完整地实现，因为我们的查询形如“$pref(|S|, 2^{i-1})$ 在 $suf(|S|, 2^i)$ 的所有出现位置”，也就是我们只关心“在某个长度不超过 2^{i-1} 的区间内的某个长度为 2^{i-1} 的串的所有出现位置”这样一类问题。

考虑再次扩展基本子串字典的内容。在第一次扩展中，我们在 $Name_t$ 中把所有长度为 2^t 的子串中相等的串的所有出现位置保存在同一个有序表内，并在这个表内二分实现 *succ* 查询。在此基础上，我们把母串 S 每 2^t 个字符开辟一个段，也就是 $[1, 2^t], [2^t+1, 2^{t+1}], [2^{t+1}, 3 \cdot 2^t], \ldots, [k \cdot 2^t, (k+1) \cdot 2^t]$ 的段。对于每一段，保存当前有序表中所有存在于这一段里的下标。

具体地，我们对每个本质不同的长度为 2^t 的串 T 保存 $seg(T, x), 1 \leq x \leq \lceil \frac{|S|}{2^t} \rceil$ ，表示 T 在母串 S 的所有在区间 $[2^t \cdot (x-1)+1, 2^t \cdot x]$ 的出现位置。根据**引理 2**，每个 $seg(T, x)$ 可以用一个等差数列表示。若其表示的等差数列为空，则 $seg(T, x) = \emptyset$ 。

如果直接这样保存空间会达到 $O(|S|^2 \log |S|)$ 级别，但是可以注意到并不需要对所有的 $seg(T, x)$ 开辟空间。对于一个固定的 t ，满足 $seg(T, x) \neq \emptyset$ 的不会超过所有串 T 的出现次数之和，而后者等于 $|S| - 2^t + 1$ 。因此，整个基本子串字典中非空的 $seg(T, x)$ 的个数是 $O(|S| \log |S|)$ 的。

为了使空间复杂度和非空 *seg* 个数同级，我们需要对每个 T 维护所有非空的 $seg(T, x)$ 的 x 的值，并支持快速查询某个数是否在这个集合内。不难发现，可以使用哈希表完成询问操作。

最后，如果我们要查询 $S[i \ldots i+2^t-1]$ 在区间 $[j, j+2^t-1]$ 的所有出现位置，只要找到 $seg(S[i \ldots i+2^t-1])$ 中所有和 $[j, j+2^t-1]$ 有交的区间，这样的区间显然最多只有两个，

取出对应的 $seg(T,x)$ 值，并适当截取。由于保证最终答案也是一个等差数列，只需要简单地对截取过后的 $seg(T,x)$ 直接合并。

这样，我们成功地把算法中用到的 *succ* 和 *pred* 查询替代为在 *seg* 的哈希表中的每次期望 $O(1)$ 查询。

综合前面的分析，子串周期查询问题做到了期望 $O(n\log n)$ 的时间空间预处理，一次查询期望 $O(\log n)$ 的复杂度。

5 基本子串字典的扩展与应用

本节主要介绍基本子串字典的一些扩展和应用。由于基本子串字典和后缀数组具有很多联系，也可以看作后缀数组向基本子串，即长度为 2 的幂次的子串方向的应用。

5.1 子串比较

基本子串字典的一个基础应用，是可以 $O(1)$ 比较两个子串的大小。尽管这个问题可以被后缀数组、高度数组和区间最值查询 (RMQ) 做到相同的复杂度，甚至可以做到更优的 $O(|S|)$ 预处理，但是基本子串字典的做法简单，在我看来有一定的启发性，而且可扩展性要强于 RMQ 做法。

5.1.1 问题描述

给定串 S ，每次查询给出 i,j,i',j' ，要求比较 $S[i\dots j]$ 和 $S[i'\dots j']$ 的大小。

5.1.2 做法

如果 $j-i\neq j'-i'$ ，那么两个子串不可能相等，可以把较长的子串的后面若干位删去使得和较短的子串长度相同，并做一次新询问。若新询问的回答是不相等，则原询问的回答和新询问的回答相同，否则原询问的回答是较长的子串更大。

现在假设 $j-i=j'-i'$ 。找到正整数 m 满足 $2^m<j-i+1\le 2^{m+1}$ ，那么两个子串的比较可以由比较 $S[i\dots i+2^m-1]$ 与 $S[i'\dots i'+2^m-1]$ ，以及比较 $S[j-2^m+1\dots j]$ 与 $S[j'-2^m+1\dots j']$ 完成。然而前两者可以通过比较 $Name_t[i]$ 与 $Name_t[j]$ 和比较 $Name_t[j-2^m+1]$ 与 $Name_t[j'-2^m+1]$ 完成。

综上所述，得到了一个 $O(n\log n)$ 预处理，$O(1)$ 查询的做法。

5.1.3 简单扩展

把原问题改为：给定一棵大小为 n 的 trie 树，定义 $up(i,l)$ 表示 i 节点向上走 l 条边，并把每条边上的字符依次连接形成的字符串。每次询问给出 i,l,i',l' ，比较 $up(i,l)$ 和 $up(i',l')$ 的大小。

此时倍增求后缀数组的算法仍然适用，但是线性求高度数组的算法却失效了。一般遇到这种情况，只能使用常数较大的二分 + 字符串哈希做法，或是较难理解的广义后缀树（后缀自动机）。

然而，基于基本子串字典的做法仍然有效，甚至还可以和 RMQ 做法结合，得到新的求高度数组的方法：由于二分 + 哈希中的哈希仅仅是用来判断字符串是否相等的，可以直接把其替换为基于基本子串字典的字符串比较，得到更优的常数。

当然，可以发现上述所有算法不可避免地需要快速求某个点 x 的 k 级祖先 $anc(x,k)$ 。记录 $f(i,k)=anc(i,2^k)$ ，并用转移式 $f(i,k)=f(f(i,k-1),k-1)$ 递推得到所有的 f 值，这样 anc 查询可以在 $O(\log n)$ 时间内解决，不过这样每次询问的复杂度增大为 $O(\log n)$ 。要做到 $O(1)$ 查询，还需要结合长链剖分 +ladder 的做法，具体做法可参考相关资料。

5.2 整周期相关

在一些有关整周期的问题中，幂串 (String Powers) 是一个重要的研究对象。

定义 6：如果一个字符串 T 有周期 $\frac{|T|}{k},k\in\mathbb{Z},k>1$ ，则称 T 是一个 k 次幂串。此时 T 可以表示为 $\left[pref(T,\frac{|T|}{k})\right]^k$ 。若还满足 $k\cdot minper(T)=|T|$ ，则称 T 是一个本原 k 次幂串。特别地，当 $k=2$ 时，T 是平方串（本原平方串），当 $k=3$ 时，T 是立方串（本原立方串）。如果没有满足条件的 k ，则称 T 是本原串。

5.2.1 整周期的简单性质

性质 1：如果 T 既是 k_1 次幂串，又是 k_2 次幂串，则 T 是 $\frac{k1\cdot k2}{\gcd(k_1,k_2)}$ 次幂串。

性质 2：T 是本原串当且仅当 T^k 是本原 k 次幂串。特别地，当 $k=2$ 时，T 是本原串当且仅当 T^2 中 T 只作为前缀和后缀出现。

这两个性质容易使用 WPL 证明。

5.2.2 k 次幂子串查找

给定一个字符串 S ，找到一个 S 的子串 $S[l\ldots r]$ 满足其是 k 次幂串。我们直接给出做法，并说明其是正确的。注意算法可能会给出多个相同的 k 次幂子串。

Algorithm 1 Detect k-th Powers

Require: S, k

Require: Function $IsPower_k(pos, root)$

Ensure: Report if S contains a k-th power

```
Name ← DBF(S)
for t = 0 to ⌊log2 |S|⌋ do
    Prev ← (0, 0, ... 0)
    for i = 1 to |S| − 2^t + 1 do
        name ← Name_t(i)
        pos ← Prev[name]
        root ← i − pos
        if IsPower_k(pos, root) then
            Report that S contains a k-th power S[pos ... pos + k · root − 1]
        end if
        Prev[name] ← j
    end for
end for
if No k-th power detected then
    Report that S does not contain a k-th power
end if
```

算法中的函数 $IsPower_k(pos, root)$ 返回 $S[pos \dots pos + k \cdot root - 1]$ 是否是一个 k 次幂串。这可以通过判断 $S[pos \dots pos + (k-1) \dots root - 1]$ 和 $S[pos + root - 1 \dots pos + k \cdot root - 1]$ 是否相等来解决，从而可以套用之前的 $O(1)$ 判断方法。

显然如果算法回答存在 k 次幂子串则一定是正确的，我们只需说明如果算法回答不存在，则 S 中一定没有 k 次幂子串。

定理 1：当 $k = 2$ 时，若 S 存在平方子串，则 **Algorithm 1** 一定会找到所有长度最短的平方子串。

引理 6：若 $w = S[i \dots j] = S[i' \dots j']$，且区间 $[i, j]$ 和 $[i', j']$ 的交非空，则 S 中存在一个长度不超过 $2|w| - 1$ 的平方串。

证明：令 $l = \min(i, i'), r = \max(j, j')$。考虑串 $S[l \dots r]$，w 是它的 border，$r - l + 1 - |w|$ 是它的周期。而 $r - l + 1 < 2|w|$，因此 $S[l \dots l - 1 + 2 \cdot (r - l + 1 - |w|)]$ 是一个平方串。 □

定理 1 的证明：令 $S[j \dots j + 2 \cdot r - 1]$ 为某个长度最短的平方子串。令 $v = S[j \dots j + r - 1]$，找到 s 满足 $2^s \le |v| < 2^{s+1}$，那么考虑 **Algorithm 1** 运行到 $t = s, i = j + r$ 时的情况。由于 $S[j \dots j + 2^s - 1] = S[i \dots i + 2^s - 1]$，可以得到 $i > pos \ge j$，因此只需证明 $pos = j$ 即可。

令 $w = S[i \dots i + 2^s - 1]$。假设 $pos > j$，则有 $S[i \dots i + 2^s - 1] = S[pos \dots pos + 2^s - 1] = S[j \dots j + 2^s - 1] = w$。由于 $j - i = |v| < 2^{s+1} = 2|w|$，区间 $[pos, pos + 2^s - 1]$ 必定和 $[i, i + 2^s - 1]$ 与 $[j, j + 2^s - 1]$ 其中之一有交。根据**引理 6** 就得到了一个长度比 $2|v|$ 更短的平方串，产生了矛盾。 □

定理 2：当 $k \ge 3$ 时，**Algorithm 1** 会找到 S 中的所有本原 k 次幂子串。

证明：令 $S[j \dots j + k \cdot r - 1]$ 为某个本原 k 次幂子串。令 $v = S[j \dots j + r - 1]$，找到 s 满足 $2^{s-1} < |v| \le 2^s$，那么考虑 **Algorithm 1** 运行到 $t = s, i = j + r$ 时的情况。由于 $S[j \dots j + 2^s - 1] = S[i \dots i + 2^s - 1]$，可以得到 $i > pos \ge j$，因此只需证明 $pos = j$ 即可。

考虑 $S[j \dots j + 2 \cdot r - 1] = v^2$，若 $i > pos > j$ 则 v 在 pos 位置再次出现，根据**性质 1** 和**性质 2**，v 不是本原串，与 v^k 是本原 k 次幂串矛盾。 □

综合**定理 1** 和**定理 2**，我们就证明了 **Algorithm 1** 的正确性。

5.2.3 Runs 和 Cubic Runs

与整周期有关的另一个重要概念是 Runs 的概念。

定义 7：字符串 S 的一个 Run 是一个三元组 (i, j, l) 满足 $l \in per(S[i \dots j])$，$l \le \frac{j-i+1}{2}$，且 $S[i-1 \dots j]$ 和 $S[i \dots j+1]$ 均未定义或不满足前述条件。若 $l \le \frac{j-i+1}{3}$，则三元组 (i, j, p) 被称为一个 Cubic Run。

根据 WPL 也不难得到下述结论：字符串 S 的所有 Run 对应的区间 $[i, j]$ 没有包含关系。

可以把 Run 理解成至少重复两次，且无法向左右扩展的周期子串。下面的重要定理说明了 Run 个数的上界。定理的证明可以参考相关资料。

定理 3：(The Runs Theorem) 一个字符串 S 至多有 $|S|$ 个 Run。

求字符串 S 的所有 Runs 的最优复杂度是 $O(|S|)$，不过需要使用后缀数组的线性构造，以及基于 Method of Four Russians 的、线性预处理 $O(1)$ 查询的 RMQ，因而十分繁琐，不适合在算法竞赛中实现。这个问题有较为容易理解且易于实现的 $O(|S|\log|S|)$ 的做法，有兴趣的读者可以阅读参考文献。

在一些问题中，我们关心的并不是所有的 Runs，而是具有更强性质的 Cubic Runs。由于 Cubic Runs 是 Runs 的子集，所以 Cubic Runs 的个数也至多是 $|S|$。不难发现，任何一个本原立方串都恰好被一个周期与其相等的 Cubic Run 包含，且一个 Cubic Run 一定包含至少一个本原立方串。这提示我们可以通过求出本原立方串来解决这类问题。我们不加证明地给出下面一个定理，这使得我们可以直接用 **Algorithm 1** 来求出所有的本原立方串。

定理 4： **Algorithm 1** 中求出的所有 k 次幂串均是本原的。

求出了所有的本原立方串，就容易得到所有 Cubic Runs 了，只要每次取出出现位置最早的本原立方串，并尽量扩展得到一个 Cubic Run 即可。

6 总结

在本文中，我们完整地解决了子串周期查询问题这一经典问题。问题虽然解决了，但是它的做法，以及所用到的三个引理和基本子串字典这个数据结构，还值得更多的研究。

希望本文能起到一个抛砖引玉的作用，带领读者初步领略有关字符串周期的魅力，并更加深入地了解不仅限于本文的相关知识和研究成果。

感谢

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张瑞喆的指导。
感谢南京外国语学校李曙老师的指导和支持。
感谢杨骏昭和王泽远同学帮我审稿。
感谢所有曾经给予我帮助的老师和同学。
感谢我的父母一直以来的全力支持和鼓励。

参考文献

[1] 金策，2017 年 CCFNOI 冬令营《字符串算法选讲》

[2] Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń. Efficient Data Structure for the Factor Periodicity Problem

[3] Maxime Crochemore, Costas Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, Krzysztof Stencel, Tomasz Waleń. New Simple Efficient Algorithms Computing Powers and Runs in Strings

[4] Manber Udi, Myers Gene. Suffix Arrays: a new method for on-line string searches

[5] Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, Kazuya Tsuruta. The "Runs" Theorem

[6] 徐明宽，IOI2018 中国国家候选队论文《非常规大小分块算法初探》

《公园》命题报告

福建省福州第一中学 吴作同

摘 要

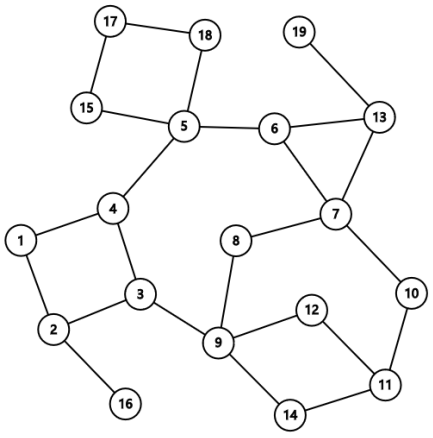
《公园》是我在 2019 年集训队互测中命制的一道试题。本文引入了“广义串并联图”的概念，并证明了任意广义串并联图可以通过“删 1 度点操作”、“缩 2 度点操作”、“叠合重边操作”使图转化为只包含一个节点的图。本文介绍了本题在不同的特殊条件下的解决思路与算法选择，其中正解算法为了支持快速修改参数，建立了表达式树来表示本题中广义串并联图的动态规划的过程，使用矩阵的形式表示转移，并用链分治线段树来加速修改。我希望《公园》这道题能够带给大家更多关于广义串并联图以及缩 2 度点操作的相关思考。

1 试题

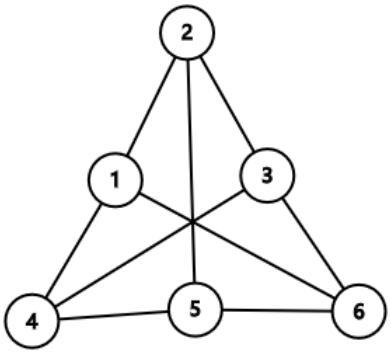
1.1 题目大意

我们称满足对于任意 4 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为广义串并联图。

下图是一张广义串并联图：



下图不是一张广义串并联图：



因为对于第 $1,4,5,6$ 号节点，存在六条两两没有公共边的路径 $1\to 4, 4\to 5, 5\to 6, 6\to 1, 4\to 3\to 6, 1\to 2\to 5$ 连接这四个节点的每一对节点。

给定一张 n 阶简单（无自环无重边）广义串并联图 $G=(V,E)$ 。每个节点有两种点权 b_i 和 w_i ，每条边有两种边权 s_i 和 d_i 。要求把每个节点染成黑白两色之一，求一种权值最大的染色方案的权值。定义染色之后节点 i 的颜色为 c_i ，即若 $c_i=0$ ，表示节点 i 为黑色；若 $c_i=1$ ，表示节点 i 为白色。一种染色方案的权值定义为

$$\sum_{v\in V}(b_v\cdot[c_v=0]+w_v\cdot[c_v=1])+\sum_{e=(u,v)\in E}(s_e\cdot[c_u=c_v]+d_e\cdot[c_u\neq c_v])$$

其中约定 $[x]$ 的含义为，若表达式 x 为真则 $[x]=1$ ，否则 $[x]=0$ 。

要求支持动态修改某个节点的两种点权或某条边的两种边权，在所有修改前与每次修改后输出答案。修改共 Q 次。

1.2 输入格式

第一行两个正整数 n,m ，其中 n 表示图 G 的节点数， m 表示图 G 的边数。

接下来 n 行，每行两个非负整数，其中第 $i(1\le i\le n)$ 行表示 b_i,w_i 。

接下来 m 行，每行四个正整数 x_i,y_i,s_i,d_i ，其中第 $i(1\le i\le m)$ 行表示第 i 条边的两个端点的编号为 x_i,y_i 且第 i 条边的两种边权为 s_i,d_i 。

第 $n+m+2$ 行一个非负整数 Q 。

接下来 Q 行每行三个正整数 x,a,b 表示一次修改，若 $x\le n$ 则表示把 b_x 改为 a ，把 w_x 改为 b ，否则表示把 s_{x-n} 改为 a ，把 d_{x-n} 改为 b 。

1.3 输出格式

输出 $Q+1$ 行，其中第 1 行表示所有修改之前问题的答案，第 $i(2\le i\le Q+1)$ 行表示第 $i-1$ 次修改后问题的答案。

1.4 样例输入

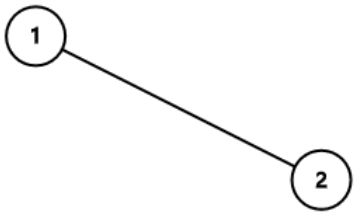
```
2 1
2 3
4 7
1 2 5 7
1
1 2 6
```

1.5 样例输出

```
16
18
```

1.6 样例解释

公园路线图如下图：



修改前，$b_1 = 2, w_1 = 3, b_2 = 4, w_2 = 7, s_1 = 5, d_1 = 7$ 。
最优染色方案为 $c_1 = 0, c_2 = 1$ ，权值为 $b_1 + w_2 + d_1 = 16$ 。
修改后，$b_1 = 2, w_1 = 6, b_2 = 4, w_2 = 7, s_1 = 5, d_1 = 7$ 。
最优染色方案为 $c_1 = 1, c_2 = 1$ ，权值为 $w_1 + w_2 + s_1 = 18$ 。

1.7 数据规模与约定

保证 $2 \le n \le 10^5, Q \le 10^5$ 。
保证 $x_i, y_i \le n$ ， $x \le n + m$ ， $b_i, w_i, s_i, d_i, a, b \le 10^6$ 。

子任务	分值	图的特殊形态	n	Q	其它限制
1	2	树		$=0$	
2	3		≤ 17	≤ 17	
3	7	仙人掌		$=0$	
4	5			$=0$	
5	13		≤ 1000	≤ 1000	
6	19	仙人掌			$b_i = w_i = 0, x > n$
7	17				
8	23	树			
9	11				

树即无环的连通图。

仙人掌即每条边属于不超过 1 个简单环的连通图。

留空部分表示没有额外限制。

1.8 时空限制

时间限制：6s

空间限制：1GB

2 初步分析

2.1 算法一

问题需要求出一种权值最大的染色方案。显然对于一种确定的染色方案，可以通过枚举边统计贡献的方式在 $O(m)$ 的时间内计算出染色方案的权值。

对于子任务 2 ，n, Q 都不超过 17 。对于一次询问，染色方案的情况总数较少，只有不超过 $2^{17} = 131072$ 种，所以可以考虑枚举所有的染色方案，分别求出权值后取最大值。

该算法时间复杂度为 $O(2^n \times Qm)$ 。题目中并没有给出 m 的取值范围，但是可以证明 m 是 $O(n)$ 级别的，证明见 5.3.1 。

期望得分 3 分。

该算法效率低下，是因为该算法没有利用到 G 是广义串并联图的性质，并且对于每一次修改都要重新计算问题的答案，没有利用到修改前后大量数据的相似性。

接下来考虑先从一些特殊的情况入手，来分析这个问题。

3 树上的问题

3.1 算法二 树，没有修改

对于此类树上最优化问题，通常可以使用树形动态规划的方式解决。

将无根树转为以 1 为根的有根树，记 $f(i,j)(j\in\{0,1\})$ 为 $c_i=j$ 时节点 i 子树的答案。

设 $C(u)$ 为 u 的子节点集合，则有

$$f(u,0)=b_u+\sum_{v\in C(u)}\max\{f(v,0)+s_{(u,v)},f(v,1)+d_{(u,v)}\}$$

$$f(u,1)=w_u+\sum_{v\in C(u)}\max\{f(v,0)+d_{(u,v)},f(v,1)+s_{(u,v)}\}$$

答案为 $\max\{f(1,0),f(1,1)\}$ 。

时间复杂度 $O(n)$ 。

可以通过子任务 1 ，期望得分 2 分。

3.2 算法三 树，带修改

考虑利用算法二的动态规划算法。

根据转移方程，可以发现只有被修改的节点/边到树的根的一条链上的节点的 f 值[43]可能会被修改。对于这类问题，可以考虑使用基于链的分治。

在树上，我们把度数为 1 的非根节点称为**叶子**或**叶子节点**，否则称为**非叶子节点**。

修改点的 f 值或修改边的父亲节点的 f 值可以直接计算，其余部分可以利用树链剖分[44] 拆成 $O(\log n)$ 条重链，然后只需考虑如何快速维护重链上的 f 值。

设 son_u 为节点 u 的重子节点。

[43]我们把一个节点 u 的 $f(u,0)$ 和 $f(u,1)$ 的值统称为节点 u 的 f 值。

[44]本文中所有提到的“树链剖分”为轻重链剖分，是摘要中提到的“链分治”的表现形式。具体地，定义点 u 的**重子节点**为 $C(u)$ 中满足 $size(v)$ 最大的 v ，若有多个则取编号最小的，其中 $size(v)$ 表示以 v 为根的子树中节点的个数，并把一个节点与其重子节点之间的连边称为**重边**，把树上不是重边的边称为**轻边**。定义**重链**为所有重边的边导出子图的一个连通块或是满足 u 与 u 的父亲节点之间的边为轻边的叶子节点 u 所构成的只包含一个节点的图。定义点 u 的**轻子节点**为 $C(u)$ 中不是 u 的重子节点的节点。

对于非叶子节点 u ，我们改写一下转移方程：

$$\begin{aligned} f(u,0) &= b_u + \sum_{v\in C(u)} \max\{f(v,0)+s_{(u,v)}, f(v,1)+d_{(u,v)}\} \\ &= b_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+s_{(u,v)}, f(v,1)+d_{(u,v)}\} \\ &\quad + \max\{f(son_u,0)+s_{(u,son_u)}, f(son_u,1)+d_{(u,son_u)}\} \\ f(u,1) &= w_u + \sum_{v\in C(u)} \max\{f(v,0)+d_{(u,v)}, f(v,1)+s_{(u,v)}\} \\ &= w_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+d_{(u,v)}, f(v,1)+s_{(u,v)}\} \\ &\quad + \max\{f(son_u,0)+d_{(u,son_u)}, f(son_u,1)+s_{(u,son_u)}\} \end{aligned}$$

使用已更新的 son_u 的 f 值更新 u 的 f 值时，只有最后一项中的 $f(son_u,0)$ 和 $f(son_u,1)$ 可能改变，其余都可视为常数。于是转移方程可以写成：

$$\begin{aligned} f(u,0) &= \max\{f(son_u,0)+A_u, f(son_u,1)+B_u\} \\ f(u,1) &= \max\{f(son_u,0)+C_u, f(son_u,1)+D_u\} \end{aligned}$$

其中

$$\begin{aligned} A_u &= b_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+s_{(u,v)}, f(v,1)+d_{(u,v)}\} + s_{(u,son_u)} \\ B_u &= b_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+s_{(u,v)}, f(v,1)+d_{(u,v)}\} + d_{(u,son_u)} \\ C_u &= w_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+d_{(u,v)}, f(v,1)+s_{(u,v)}\} + d_{(u,son_u)} \\ D_u &= w_u + \sum_{v\in C(u)\wedge v\neq son_u} \max\{f(v,0)+d_{(u,v)}, f(v,1)+s_{(u,v)}\} + s_{(u,son_u)} \end{aligned}$$

于是重链上的转移可以用类似矩阵的形式来表示。

3.2.1 矩阵表示

令

$$f_u = \begin{bmatrix} f(u,0) \\ f(u,1) \end{bmatrix} = \begin{bmatrix} \max\{f(son_u,0)+A_u, f(son_u,1)+B_u\} \\ \max\{f(son_u,0)+C_u, f(son_u,1)+D_u\} \end{bmatrix}$$

观察表达式的形式，发现它很类似一个矩阵乘法的结果，只是原本矩阵乘法中元素的加法在这里为 max 运算，原本矩阵乘法中元素的乘法在这里为加法运算。

由于加法、max 运算满足交换律、结合律且加法对 max 运算满足分配律，所以这样新定义的矩阵乘法仍然满足结合律 [3]。

本题中所有矩阵乘法都在该定义下进行。

令

$$g_u = \begin{bmatrix} A_u & B_u \\ C_u & D_u \end{bmatrix}$$

则有

$$\begin{aligned} g_u f_{son_u} &= \begin{bmatrix} A_u & B_u \\ C_u & D_u \end{bmatrix} \begin{bmatrix} f(son_u, 0) \\ f(son_u, 1) \end{bmatrix} \\ &= \begin{bmatrix} \max\{f(son_u, 0) + A_u, f(son_u, 1) + B_u\} \\ \max\{f(son_u, 0) + C_u, f(son_u, 1) + D_u\} \end{bmatrix} \\ &= f_u \end{aligned}$$

3.2.2 分治

初始状态下，每个 A_u, B_u, C_u, D_u 的值，可以在预处理时利用算法二计算出。之后当且仅当修改 u 到某个子节点之间的边的边权，或修改 u 或 u 的某个轻子节点的子树内的点的点权时，A_u, B_u, C_u, D_u 的值才需要更新。由于一个点到根的简单路径上只有 $O(\log n)$ 条轻边，所以一次修改只有 $O(\log n)$ 个点的 A_u, B_u, C_u, D_u 被更新。但是为了更新一个点 u 的 A_u, B_u, C_u, D_u 的值或计算答案（即计算 $f(1,0)$ 与 $f(1,1)$ 的值），就需要计算出某一条重链的顶端节点[45]的 f 值。我们可以用线段树来维护该 f 值。

有了 3.2.1 中提到的的矩阵乘法以及结合律，就可以使用线段树来维护重链上一段区间内的矩阵 g 的乘积，在修改某个点 u 的 g_u 时，直接在对应的线段树上更新即可。

注意 g_u 对叶子节点 u 是没有定义的，所以建线段树时只需要维护非叶子节点的信息。

因此重链顶端节点 u 的 f 值可以按如下方法计算：

若 u 是叶子，则 $f_u = \begin{bmatrix} b_u & w_u \end{bmatrix}^T$ [46]，否则可以利用线段树维护的 g 矩阵计算。

设重链上的节点为 $u_1, u_2, \cdots, u_l$，其中 $u_1 = u$，且对于 $1 \le i < l$，u_i 是 u_{i+1} 的父亲节点，且 u_l 一定为叶子[47]。

那么有

$$f_{u_1} = g_{u_1} g_{u_2} \cdots g_{u_{l-1}} f_{u_l}$$

[45] 重链的顶端节点即重链上距离根最近的节点。底端节点即重链上距离根最远的节点。

[46] 符号 ^{T} 表示矩阵的转置。

[47] 根据树链剖分的方法可知，重链的底端一定是叶子。

于是直接在线段树上查询整条重链上的 g 矩阵的乘积，再右乘上 f_{u_l} 即可得到 f_u 的值。

总结一下，在一次修改中，这个算法的过程是这样的：

1、修改被修改的点或边的权值，令 u 为被修改边的父亲节点或被修改点。

2、如果 u 不是叶子，更新 g_u ，并更新对应重链上的线段树。

3、令 u 为原来 u 对应重链的顶端节点。

4、重新计算 f_u 的值并令 v 为 f_u 。

5、若 u 为根，返回 v 作为答案，否则令 u 为 u 的父亲节点。

6、利用计算好的 v 的值更新 g_u 。

7、跳转到第 3 步。

在一次修改中，需要进行 $O(\log n)$ 次线段树单点修改操作，所以该算法的时间复杂度为 $O(n+Q\log^2 n)$ 。

可以通过子任务 $1,8$ ，期望得分 25 分。

4 仙人掌上的问题

4.1 算法四 仙人掌，没有修改

对于仙人掌上的问题，一种经典的方法是仙人掌上的动态规划。

时间复杂度 $O(n)$ ，可以通过子任务 $1,3$ ，获得 9 分。

该部分内容与本文主题无关，因此不再赘述。相关内容可以参考陈俊锟的论文《〈神奇的子图〉命题报告及其拓展》。

子任务 6 有一个“所有点权均为 0 ”的限制。这个限制可以对解题有什么样的帮助呢？先分析一下树上的情况。

4.2 算法五 树，点权均为 0

当点权均为 0 时，染色方案的权值只和每条边两端的点的颜色是否相等有关。

我们称两端节点颜色相同的边为同色边，两端节点颜色不同的边为异色边。

有一种思路是，枚举每条边是同色边还是异色边，然后判断是否存在一种染色方案满足条件。

对于树上的情况，假设我们已经确定了每条边是同色边还是异色边，那么我们可以令 $c_1=0$ ，然后进行一次深度优先搜索。对于每个非根节点，可以根据父亲节点的颜色以及与父亲节点之间的边是同色边还是异色边来确定自己的颜色。这证明了对于任意一种枚举的情况，均存在一种染色方案满足条件。所以对于树上点权均为 0 的问题，我们可以对每条边贪心地取对答案贡献较大的情况，答案为

$$\sum_{e\in E}\max\{s_e,d_e\}$$

接下来考虑仙人掌上点权均为 0 的问题。

4.3 算法六 仙人掌，点权均为 0

仍然可以使用深度优先搜索来判断是否存在一种满足条件的染色方案。搜索出任意一棵 DFS 树，按照 4.2 中的方法确定每个点的颜色，再判断所有的非树边是否均满足条件。

可以发现存在合法的染色方案，当且仅当不存在一个简单环使得环上异色边的数量为奇数。

由于仙人掌上每条边最多属于一个简单环，所以可以把每个简单环以及割边独立计算出答案再加起来。

对于割边，直接把 $\max\{s_i, d_i\}$ 计入答案即可。对于环，考虑先把所有边的 $\max\{s_i, d_i\}$ 计入答案，若异色边的数量为偶数，则取到理论最大值并且合法；否则可以找到使 $|s_i - d_i|$ 最小的一条边，把它两端颜色的异同性反转，即可得到最优解。修改的时候把上一次询问的答案加上修改边所在环或割边的答案的增量即可。

时间复杂度 $O((n+Q)\log n)$ 。

可以通过子任务 6 ，期望得分 19 分。

结合以上所有树和仙人掌的部分（算法三、算法四、算法六）可以得到 51 分。

5 较为一般的图的形态

广义串并联图的性质为，对于任意 4 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。之前的算法对于图的形态都有更高的要求，所以不适合用于这种较一般的情况，但是仍然可以从树与仙人掌出发，寻找一些更为通用的做法。为了方便分析，这里暂且不考虑需要修改的情况。

5.1 树上问题的另一种解决方法

我们先来看一道例题。

例题 1. 树上最大权独立集问题[48]

给定一棵 n 个节点的树 T ，节点 i 有点权 a_i 。求一个点集的子集 S 使得 S 中任意两个节点不相邻且 S 中节点的权值和最大。

数据范围 $1 \le n \le 10^5, 1 \le a_i \le 10^9$

时间限制 1 秒

空间限制 128 MB

[48]题目来源：经典问题

当然，这道题有类似算法二的动态规划算法，但是我们现在考虑另一种算法。

当所有 a_i 均为 1 时，有一种经典的贪心做法：

每次找到一个度数不超过 1 的节点 u ，把它加入 S（即贪心地认为 u 一定在 S 中），并在 T 中删去与 u 相邻的点 v（如果存在这样的 v ，根据题意若 $u \in S$ 则必有 $v \notin S$ ）以及 u 本身，直到 T 中不存在任何节点，此时 S 就为一组最优解。

这个算法的正确性证明如下：

证明. 假设存在一个最优解为 $S^* \neq S$ 。

当 a_i 均为 1 时，S 中节点的权值和最大相当于 $|S|$ 最大。

如果 u 的度数为 0 ，且不在 S^* 中，那么显然可以把 u 加入 S^* 来增加 S^* 的大小，所以 S^* 不是问题的最优解，与假设矛盾。

如果 u 的度数为 1 ，且不在 S^* 中，若 u 的相邻节点 $v \notin S^*$ ，则把 u 加入 S^* 同样可以得到一组更优的解，与假设矛盾；若 $v \in S^*$ ，则可以把 S^* 中的 v 替换成 u ，仍然是一组合法的最优解。

于是可以求出删去 u 和 v（如果存在）后的问题的最优解 S' ，然后令 $S = S' \cup \{u\}$ 就可以得到原问题的最优解。递归边界是图为空，此时显然 $S = \varnothing$ 就是问题的最优解。

这样每次操作至少会删去一个点。而树的非空子图一定存在一个度数不超过 1 的节点，因为树的非空子图的每个连通块必然是树，并且根据抽屉原理，树一定存在一个度数不超过 1 的节点。所以经过有限次操作后，算法一定会结束。 □

但是，如果去掉 a_i 均为 1 的限制，这个贪心算法就是错误的，原因在于在上面的证明中把 v 换成 u 时，可能会使解变劣，但是证明的其余部分仍然适用。

于是可以考虑改进一下这个贪心算法，在把 u 加入 S 后给一个撤销该操作并把 v 加入 S 的机会。这相当于之后可以考虑把 v 加入 S ，但是需要额外花费 a_u 的代价把原来已经在 S 中的 u 移出 S 。这一步可以通过把 a_v 改为 $a_v - a_u$ 来实现。若 $a_v < a_u$ ，那么最优解中一定包含 u 而不包含 v ；因为若包含 v ，则把 v 替换为 u 后可以得到更优的解。于是在这种情况下可以直接把 u 加入 S 并删去 v 。改进后的算法可以处理一般的情况。

把改进后的算法用在本题的树上问题上也是类似的。若遇到一个度数为 1 的节点 u ，那么找到它的相邻节点 v 。若把 v 染成黑色，那么 u 以及边 (u,v) 最多可以贡献的权值为 $\max\{s_{(u,v)} + b_u, d_{(u,v)} + w_u\}$ ；若把 v 染成白色，那么 u 以及边 (u,v) 最多可以贡献的权值为 $\max\{d_{(u,v)} + b_u, s_{(u,v)} + w_u\}$ 。于是可以把 b_v 改为 $b_v + \max\{s_{(u,v)} + b_u, d_{(u,v)} + w_u\}$ ，把 w_v 改为 $w_v + \max\{d_{(u,v)} + b_u, s_{(u,v)} + w_u\}$ ，并删去节点 u 和边 (u,v) 。这样操作后与操作前的问题的答案相等，但从问题规模上前者比后者少了一个节点。我们称这个把一个 1 度点[49]删去并修改其相邻点的点权的操作为**删 1 度点操作**。

一棵点数大于 1 的树一定有 1 度点，并且删去一个 1 度点后仍为一棵树。所以不断地对一棵树进行删 1 度点操作，可以使树转化为只包含一个节点的图，且答案不变。而仅包

[49]本文中把“度数为 k 的节点”简称为“ k 度点”

括单个点 u 的图的答案显然为 $\max\{b_u, w_u\}$ ，从而一个树上的静态问题可以在 $O(n)$ 的时间内得到解决。

删 1 度点操作对图的整体形态并没有要求，只要图中有一个度数为 1 的节点就可以进行该操作。

解决了树上的问题，现在可以去仙人掌上寻找更多的启发。

5.2 在仙人掌上的应用

这里不加证明地给出一些要用到的关于点双连通分量的知识：

定理 5.1. *对于任意的非空无向图 G ，一定存在一个 G 的点双连通分量 B ，使得 B 中只有不超过 1 个节点是 G 的割点。其中，若 B 中没有 G 的割点，则有 $B = G$ 。*

定理 5.2. *若一个点双连通分量不为 K_2 [50]，则该点双连通分量中至少有一个简单环。*

引理 5.1. *在仙人掌上的每个点双连通分量要么是 K_2 ，要么是一个简单环。*

引理 5.2. *对于一个不是 K_2 的点双连通分量中的任意一个点 u ，一定存在一个简单环 C 使得 u 在 C 上。*

树上的问题可以使用删去 1 度点（叶子）的方式简化问题，那么仙人掌能否用类似的方法呢？

首先若存在度数为 1 的节点，则同样可以使用删 1 度点操作。

否则可以考虑定理 5.1 ，找到一个点双连通分量 B 使得 B 中**只有不超过一个节点**是 G 的割点；再结合引理 5.1 ，B 一定是一个简单环。

此时 B 在图中的地位与度数不超过 1 的节点在树上的地位类似。那么类比树上的做法，能否将 B 转化为一个节点，来减少 G 中点双连通分量的数量呢？

答案是肯定的。

由于点双连通分量 B 是一个环，所以 B 中的节点中除了割点（如果割点存在），度数均为 2 。

在遇到 1 度点时，我们根据其相邻点的染色方案计算其带来的贡献，在遇到 2 度点时同样可以使用这个方法。

设 2 度点 x 的相邻两个节点（编号）为 u, v 。

当 $c_u = 0, c_v = 0$ 时，x 和与 x 相邻的两条边对答案的最大贡献为

$$w_{00} = \max\{s_{(u,x)} + b_x + s_{(x,v)}, d_{(u,x)} + w_x + d_{(x,v)}\}$$

当 $c_u = 0, c_v = 1$ 时，x 和与 x 相邻的两条边对答案的最大贡献为

$$w_{01} = \max\{s_{(u,x)} + b_x + d_{(x,v)}, d_{(u,x)} + w_x + s_{(x,v)}\}$$

[50] 定义 K_n 为包含 n 个节点的完全图。

当 $c_u = 1, c_v = 0$ 时，x 和与 x 相邻的两条边对答案的最大贡献为

$$w_{10} = \max\{d_{(u,x)} + b_x + s_{(x,v)}, s_{(u,x)} + w_x + d_{(x,v)}\}$$

当 $c_u = 1, c_v = 1$ 时，x 和与 x 相邻的两条边对答案的最大贡献为

$$w_{11} = \max\{d_{(u,x)} + b_x + d_{(x,v)}, s_{(u,x)} + w_x + s_{(x,v)}\}$$

由于该贡献同时与 c_u, c_v 的取值有关，所以不能像删 1 度点时那样考虑更新 u, v 的点权，而是可以考虑新建一条连接 u, v 的边，在边上记录 c_u, c_v 的每种取值的情况下对答案的贡献。

我们用 $(u, v, (w_{00}, w_{01}, w_{10}, w_{11}))$ 表示该边对答案的贡献为 $w_{c_u c_v}$ 。我们称它的边权为

$$\begin{bmatrix} w_{00} & w_{01} & w_{10} & w_{11} \end{bmatrix}^T$$

输入中给出的边是 $(u, v, (s, d))$ 的形式。为了使格式一致，可以把它改为 $(u, v, (s, d, d, s))$ 。

我们将把一个 2 度点和其相邻两条边转化为一条边的操作称为**缩 2 度点操作**。

可以注意到，缩 2 度点操作不会使 u, v 的度数发生改变。

于是对于一个 n $(n \geq 3)$ 元环，可以通过使用一次缩 2 度点操作使其转化为一个 $(n-1)$ 元环。特殊地，对于一个 3 元环，使用一次缩 2 度点操作后会得到两条重边。

由于贡献是可以叠加的，所以若遇到两条重边 $(u, v, (w_{00}, w_{01}, w_{10}, w_{11}))$ 和 $(u, v, (w'_{00}, w'_{01}, w'_{10}, w'_{11}))$ ，可以将它们合并为一条边

$$(u, v, (w_{00} + w'_{00}, w_{01} + w'_{01}, w_{10} + w'_{10}, w_{11} + w'_{11}))$$

我们称把两条重边合并为一条边的操作为**叠合重边操作**[51]。

设 B 上有 k 个节点，因为 B 中只有一个 G 上的割点，而其他点都是 2 度点，且缩 2 度点操作和叠合重边操作不会使 2 度点的度数变得大于 2 ，所以在经过 $(k-2)$ 次缩 2 度点操作和一次叠合重边操作之后， B 就转化成了 K_2 。再进行一次删 1 度点操作，G 中就减少了一个点双连通分量。

所以经过 $O(n)$ 次操作之后，可以把 G 转化为一个节点。

这样就可以在 $O(n)$ 的时间内解决仙人掌上的静态问题。

5.3 广义串并联图

下面证明，不仅是仙人掌，任意广义串并联图都可以在若干次缩 2 度点、叠合重边、删 1 度点的操作后变为一个只包含一个节点的图。

[51]重边的方向有可能是相反的，即可能遇到 $(u, v, (w_{00}, w_{01}, w_{10}, w_{11}))$ 和 $(v, u, (w'_{00}, w'_{01}, w'_{10}, w'_{11}))$ ，此时只要改变其中一条边的方向再按上述方法叠合即可，叠合后的边为 $(u, v, (w_{00} + w'_{00}, w_{01} + w'_{10}, w_{10} + w'_{01}, w_{11} + w'_{11}))$ 。

5.3.1 证明

考虑证明它的逆否命题，即证明任意一个不能用上述操作使其变为一个只包含一个节点的图的无向连通图不是广义串并联图。

这里先证明两个结论：

定理 5.3. *若一张无向连通图 G 中存在 3 个不同的 1 度点 x,y,z，则一定存在一个点 $u \notin \{x,y,z\}$ 使得存在 3 条两两没有公共边的简单路径满足其中一个端点均为 u 且另一个端点分别为 x,y,z。*

证明. 先求出 G 的任意一棵生成树 T。

由于 x,y,z 是 1 度点，所以在 T 中 x,y,z 一定是 1 度点。

由于任意图中度数为奇数的点必有偶数个，所以 G 中一定存在一个异于 x,y,z 的节点。

取一个异于 x,y,z 的点作为根，把 T 转化为有根树，令 u 为 x 和 y 的最近公共祖先，即令 $u = \mathrm{LCA}(x,y)$。因为 x,y,z 均为叶子且不为根，所以有 $u \neq x, u \neq y, u \neq z$。

若 z 不在 u 的子树中或 $\mathrm{LCA}(x,z), \mathrm{LCA}(y,z)$ 均为 u，那么在 T 上 u 到 x,y,z 的三条路径两两没有公共边。

否则 $\mathrm{LCA}(x,z)$ 和 $\mathrm{LCA}(y,z)$ 恰有一个不为 u。

不失一般性，假设 $w = \mathrm{LCA}(x,z) \neq u$，那么 y 不在 w 的子树中，又因为 $\mathrm{LCA}(x,z) = w$，所以在 T 上 w 到 x,y,z 的三条路径两两没有公共边。 □

定理 5.4. *对于一个无向图 G，若进行若干次删 1 度点操作，缩 2 度点操作以及叠合重边操作后得到的图不是广义串并联图，那么 G 也不是广义串并联图。*

证明. 考虑还原所进行的操作。

设操作后的图中 4 个使图不满足广义串并联图的性质的节点为 a,b,c,d，6 条两两边不相交的路径为 $p(a,b), p(a,c), p(a,d), p(b,c), p(b,d), p(c,d)$。

删 1 度点操作的逆操作为选择一个点 $u \in V$，加入新点 v 和边 (u,v)。

叠合重边操作的逆操作为选择一条边 $e = (u,v) \in E$，加入一条新的重边 (u,v)。

由于这两个逆操作不会删除图中的节点或边，所以若删 1 度点操作或叠合重边操作后的图不是广义串并联图，那么操作前的图也不是广义串并联图。

缩 2 度点操作的逆操作为选择一条边 $e = (u,v) \in E$，删去 e 并加入新点 w 以及新边 $(u,w), (w,v)$。

若还原时没有删去这 6 条路径中任何一条路径上的边，则原来的 $p(a,b), p(a,c), p(a,d), p(b,c), p(b,d), p(c,d)$ 同样能够作为判断操作前的图不是广义串并联图的依据。否则不失一般性，假设要删去 $p(a,b)$ 上的一条边。那么可以在 $p(a,b)$ 中删去那条边并加入逆操作需要加入的两条边和一个节点，这样仍然满足存在 $p(a,b), p(a,c), p(a,d), p(b,c), p(b,d), p(c,d)$ 两两没有公共边。

因此，若任意多次操作之后的图不是广义串并联图，那么操作之前的图也一定不是广义串并联图。 □

一张简单无向图不能再进行操作，当且仅当每个连通块是一个 0 度点或一个所有点的度数都大于等于 3 的图。下面给出定理：

定理 5.5. *任意一张所有点的度数都大于等于 3 的简单无向连通图，一定不是广义串并联图。*

证明. 假设有一个满足所有点的度数都大于等于 3 的简单无向连通图 G 。

根据定理 5.1，可以在 G 中找到一个点双连通分量 B ，使得 B 中只有不超过 1 个节点使得该节点是 G 的割点。

这个点双连通分量 B 一定不是 K_2 ，否则 G 中就会存在一个 1 度点。

若 B 中有 G 的割点，根据引理 5.2，B 中一定存在一个包含该唯一割点的简单环，设其为 C 。

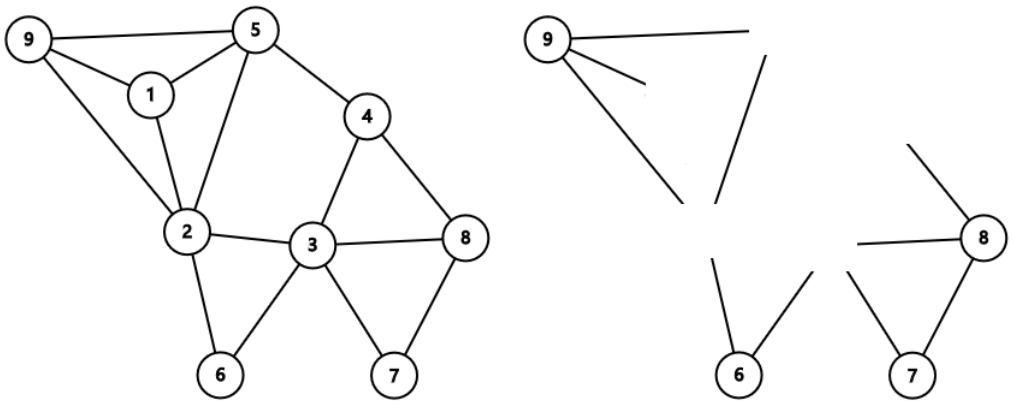
否则根据定理 5.2，B 中一定存在一个简单环。任取一个 B 中的简单环，并设为 C 。

设 C 上的点依次为 $v_0, v_1, \cdots, v_{l-1}$ ，即 v_0 和 v_{l-1} 相邻，对于所有的 $0 \le i < l-1$ ，v_i 与 v_{i+1} 相邻。

设 $V' = V \setminus V_C, E' = E \setminus E_C$ ，在 $V' \cup E'$ 上定义二元关系 $\sim$ ：对于 $u, e \in V' \cup E'$ ，$u \sim e$ 当且仅当 $u \in V', e \in E'$ ，且 u 是 e 的一个端点。再定义 $V' \cup E'$ 上的等价关系 $\leftrightarrow$ 是二元关系 $\sim$ 的最小等价关系。直观地说，如果 a, b 是 G 中的顶点或边，且不在 C 中，那么 $a \leftrightarrow b$ 当且仅当从 a 出发可以经过一系列不在 C 中的点和边到达 b ，这类似图中的连通性。

我们称等价关系 $\leftrightarrow$ 的等价类为**片**。对于片 U ，定义它的**点集**为 $U \cap V$ ，它的**边集**为 $U \cap E$ ，它对 C 的**联系点集**为 $A(U) = \{v \in V_C \mid \exists u \in V,\ e = (u, v) \in U \cap E\}$ ，直观地说，在 G 中删去所有在 C 上的顶点和边（但是那些不在 C 上的边即使一个或两个顶点被删去也应该保留），“连通”的部分就是片；$A(U)$ 就是片 U 的没有或者只有一个端点的边所缺少的端点集合。

以下是一个例子：（左边的图删去环 $1 \to 2 \to 3 \to 4 \to 5 \to 1$ 后会得到右边的图）



右图中共有 4 个片，这 4 个片包含的边的数量分别为 $1,2,3,4$ ，联系点集的大小分别为 $2,2,3,2$ 。

由于 G 中每个节点的度数都大于等于 3 ，所以对于环上的任意一个点 v_i ，至少存在一个片 U 使得 $v_i \in A(U)$ 。

若 B 中存在 G 的割点则设该割点为 v_k （此时割点唯一且在 C 上），若不存在割点则 $k=-1$ 。

若存在一个 U 使得 $|A(U)|=1$ ，由于删去其中的唯一元素之后 G 不连通，即它是 G 的割点，所以对于所有满足 $|A(U)|=1$ 的 U ，均满足 $A(U)=\{v_k\}$ （显然当 $k=-1$ 时 v_k 不存在，这也意味着当 $k=-1$ 时不存在 U 使得 $|A(U)|=1$ ）。

接下来分三种不同的情况讨论说明 G 一定不是广义串并联图。

情况 1. 存在一个 U 使得 $|A(U)|\geq 3$ 。

假设 U 中有一条边 e 满足其两个端点均不在 U 中，那么 e 与 V' 中的任意一个元素 u 均不可能满足 $u\sim e$ 。自然地，对于 $V'\cup E'$ 中的任意一个元素 a ，均不满足 $e\leftrightarrow a$ 。所以有 $U=\{e\}$ ，且 $A(U)$ 中的元素为 e 的两个端点，与 $|A(U)|\geq 3$ 矛盾。因此，U 中不可能有一条边 e 满足 e 的两个端点均不在 U 中。

可以考虑把 $A(U)$ 中的所有元素都加入 U ，得到一张图 $U'=((U\cap V)\cup A(U),U\cap E')$ 。在 U' 中，$U\cap V$ 是连通的，这是因为 U 是等价关系 $\leftrightarrow$ 的等价类，而 U 中存在一个端点不属于 U 的边不传导 U 的点集在 U' 中的连通性。

在 U' 中，删去若干条与 $A(U)$ 的元素相邻的边使得 $A(U)$ 的元素的度数均为 1 。此时 $A(U)$ 中的点仍然通过一条 U' 中的边与 U 的点集连通，因此 U' 是连通图。

这样根据定理 5.3 ，设 $x,y,z\in A(U)$ ，U 中一定存在一个节点 u 使得 U' 中有 3 条两两没有公共边的路径分别连接 u 与 x ，u 与 y ，u 与 z 。又因为 x,y,z 在同一个环 C 上，所以在 C 上存在 3 条两两没有公共边的路径分别连接 x 与 y ，y 与 z ，z 与 x 。因为 U' 与 C 没有公共边，所以找到了 4 个节点 x,y,z,u 使得存在 6 条两两没有公共边的路径分别连接这 4 个节点的每一对，即说明了 G 不是广义串并联图。

情况 2. 对于所有的 U 都有 $|A(U)|\leq 2$ ，并且存在 U_1,U_2 使得 $A(U_1)=\{v_x,v_y\},A(U_2)=\{v_z,v_w\}$ ，其中 $x<z<y<w$ 。

此时在环 C 上可以找到 4 条两两没有公共边的路径连接 v_x 与 v_z ，v_z 与 v_y ，v_y 与 v_w ，v_w 与 v_x ，且 G 中存在一条连接 v_x 与 v_y 的路径满足其边集包含于 U_1 的边集，存在一条连接 v_z 与 v_w 的路径满足其边集包含于 U_2 的边集。又因为 U_1 的边集、U_2 的边集、C 的边集两两不交，所以找到了 6 条两两没有公共边的路径连接 4 个节点 x,y,z,w 中的每一对，即说明了 G 不是广义串并联图。

情况 3. 对于所有的 U 都有 $|A(U)|\leq 2$ ，并且存在 U_0,i 使得 $A(U_0)=\{v_i,v_{i+1}\}$ 。

以下说明，情况 $1,2,3$ 涵盖了所有可能情况。

如果不满足情况 1 ，我们有以下事实：

1. 对于环上的任意一个点 v_i ，至少存在一个片 U 使得 $v_i \in A(U)$ ；

2. 所有满足 $|A(U)| = 1$ 的 U 均满足 $A(U) = \{v_k\}$ ；

3. 简单环 C 至少包含 3 个节点；

4. 不存在任何 U 使得 $A(U) \geq 3$ 。

综合上述四个事实，我们知道一定能找到一个片 U 使得 $|A(U)| = 2$ 。

假设 $A(U) = \{v_x, v_y\}(x < y)$ ，由于环可以按照顺时针或逆时针来定向，所以可以不失一般性地假设 $k \notin (x, y)$ 。

下面证明若不满足情况 1 和情况 2 ，则一定存在 U_0, i 使得 $A(U_0) = \{v_i, v_{i+1}\}$ 。

若 $y - x = 1$ ，则取 $U_0 = U, i = x$ 可以满足要求。

否则找到 U_1, z 使得 $A(U_1) = \{v_{x+1}, v_z\}$ 。根据事实 *1,2,4* ，且 $k \notin (x, y)$ ，这样的 U_1, z 一定存在。

若 $z = x$ ，则取 $U_0 = U_1, i = x$ 可以满足要求；若 $z \notin [x, y]$ ，则 U 与 U_1 满足情况 2 ，矛盾。

所以若 $z \neq x$ ，则必有 $z \in (x + 1, y]$ 。

令新的 $x' = x+1, y' = z$ ，则有 $y'-x' = y'-x-1 \leq y-x-1$ ，并且仍有 $y' > x', k \notin (x', y')$ 。

每一次 x, y 到 x', y' 的迭代都会使 $y - x$ 的值减小，所以经过有限次迭代后一定能够找到满足条件的 U_0, i 。

所以若不满足情况 1 和情况 2 ，则一定存在 U_0, i 使得 $A(U_0) = \{v_i, v_{i+1}\}$ 。

令 $G' = (V_{G'}, E_{G'})$ 为 U_0 加入节点 v_i, v_{i+1} 以及边 (v_i, v_{i+1}) 后的无向图。那么有 $deg(v_i) \geq 2, deg(v_{i+1}) \geq 2, \forall u \in U_0, deg(u) \geq 3$ 。

所以，G' 中不存在度数不超过 1 的点，G 中 2 度点的个数不超过 2 ，并且若存在 2 个 2 度点，则这 2 个 2 度点必然相邻。

可以证明 $V_{G'} \subseteq V_B$ 成立：若 $k \neq -1$ ，假设存在 $u \in V_{G'}$ 且 $u \notin V_B$ ，则连接 u 和 B 中任意一点的任意一条路径必然经过 v_k ，而 $v_k \in V_C$ ，所以有 $A(U_0) = \{v_k\}$ ，与 $|A(U_0)| = 2$ 矛盾；若 $k = -1$ ，则根据定理 5.1 有 $B = G$ ，所以有 $V_{G'} \subseteq V = V_B$ 。

下面说明 G' 是点双连通图，即 G' 不存在割点。

首先 v_i 和 v_{i+1} 不是 G' 的割点，否则 U_0 不连通。

其次 G' 中其余节点也不是 G' 的割点，否则它同时也是 G 的割点。已知当 $k = -1$ 时 B 中没有 G 的割点，而 $k \neq -1$ 时 B 中只有 v_k 是 G 的割点，然而因为 $V_{G'} \cap V_C = \{v_i, v_{i+1}\}$ ，所以一定有 $v_k \notin V_{G'} \setminus \{v_i, v_{i+1}\}$ 。

所以，G' 不存在割点，即 G' 是点双连通图。

可以通过枚举得到，对于所有节点数不超过 3 的简单图，均不能同时满足以上两个性质[52]。所以 G' 至少包含 4 个节点。

考虑对 G' 进行缩 2 度点、叠合重边操作。缩 2 度点操作不会影响与 2 度点相邻的两个点 u, v 的度数，而之后的叠合重边操作（如果需要）只会使 u, v 的度数减少 1 。

[52]这两个性质分别为“是一个点双连通图”和“不存在度数不超过 1 的点且 2 度点的数量不超过 2 。

所以，若 G' 只有一个 2 度点，那么在操作后，G' 中不存在度数不超过 1 的节点，并且 2 度点数量仍然不超过 2 ，并且若有 2 个 2 度点，则这 2 个 2 度点必然相邻。

若 G' 有两个相邻的 2 度点 x,y ，设与 x 相邻的另一个节点为 z ，与 y 相邻的另一个节点为 w ，则 $z \neq w$ ，否则有 z 的度数为 2 或 z 是 G' 的割点。不失一般性，假设接下来对 x 进行缩 2 度点操作，由于 y 与 z 之间没有边，所以不会进行叠合重边操作，y 与 z 的度数均不变，所以操作之后 G' 中只有 1 个 2 度点，没有 1 度点。

接下来证明经过一次缩 2 度点以及一次叠合重边操作（如果缩 2 度点之后出现重边）之后，G' 仍是点双连通图。

设操作之后的图为 G'' 。

判断一个图是点双连通图的依据是图中不存在任何一个点 u 使得删去 u 后图不连通。

由于重边不会改变点的连通性，所以叠合重边操作不会对图的点双连通性产生影响。

现在考虑缩 2 度点操作。设 G' 中操作的 2 度点为 x ，与 x 相邻的两个节点为 y,z 。

考虑在 G'' 的边 (y,z) 中“插入”一个节点 x ，即加入节点 x 后把边 (y,z) 替换为边 (y,x) 和边 (x,z) ，这样 G'' 就变成了 G' 。

所以证明删去 G'' 中任何一个节点，G'' 均连通，相当于证明删去 G' 中任何一个不为 x 的节点，G' 均连通。

因为 G' 是点双连通图，所以删去 G' 中任意一个节点 G' 均连通，所以 G'' 是点双连通图。

所以在操作过程中的任一时刻，G' 是点双连通图，且 G' 中不存在度数不超过 1 的点，且 G' 中 2 度点的数量不超过 2 。

每一次缩 2 度点操作会使 G' 的点数减少 1 。假设某一时刻 G' 的点数为 3 ，又因为 G' 是点双连通图，所以此时 G' 一定是 K_3 。K_3 有 3 个 2 度点，而任一时刻 G' 中的 2 度点数量不超过 2 ，所以 G' 不能够转化为一个不超过 3 个节点的图。又因为任意时刻 G' 中均没有 1 度点，所以在经过足够但有限多次（可能为 0 次）缩 2 度点操作和叠合重边操作后，G' 的每个节点的度数都大于等于 3 。

由于最初的 G' 中只有 2 个节点在 C 上，而 C 至少有 3 个节点，所以 G' 的点数一定比 G 小。把操作后每个节点的度数都大于等于 3 的 G' 作为新的 G 迭代下去，G' 仍然能被归为情况 1,2,3 中的一种情况，其中 G' 满足情况 1 或情况 2 时 G' 不是广义串并联图，而满足情况 3 时 G' 的点数一定会减小，但 G' 的点数一定大于等于 4 ，故经过有限次迭代之后，一定能够找到 6 条两两没有公共边的路径连接 4 个顶点中的每一对顶点。根据定理 5.4 ，得出 G 不是广义串并联图。

因此，一张所有点的度数都大于等于 3 的简单无向连通图，一定不是广义串并联图。

□

显然，一个连通图进行缩 2 度点操作、叠合重边操作、删 1 度点操作均不能使其变为一个不连通的图。

根据定理 5.4 和定理 5.5 ，可以得出，任意一个不能用缩 2 度点操作、叠合重边操作、删 1 度点操作使其变为一个只包含一个节点的图的无向连通图不是广义串并联图。

由于删 1 度点操作与缩 2 度点操作会使图的节点数和边数均减少 1 ，并且叠合重边操作只可能在缩 2 度点操作之后执行并使边数减少 1 ，所以可以证明，任意广义串并联图可以在 $O(n)$ 次缩 2 度点、叠合重边、删 1 度点的操作后变为一个只包含一个节点的图；此外还可以推出，简单广义串并联图的边数是 $O(n)$ 级别的，更具体地，有 $m \le 2n$ 。

5.4 算法七 广义串并联图，没有修改

有了上面的结论，就可以直接使用 5.2 中的算法通过子任务 $1,3,4$ 。每次修改后暴力重新计算，就可以通过子任务 $2,5$ 。

时间复杂度 $O(nQ)$ 。

期望得分 30 分。

6 考虑修改

现在有了一种能够在线性时间内解决一组询问的算法，那么可以考虑在此之上进行一些优化，使其能更高效地支持修改。

先来考虑一种较为简单的情况：子任务 7 中所有点的点权均为 0 ，此时无需考虑点权对答案造成的影响。

6.1 算法八 广义串并联图，点权均为 0

考虑需要使用的所有操作。

叠合重边操作是把两条重边的边权对应相加，并换成一条新的边。

缩 2 度点操作是把一个 2 度点的相邻两条边，经计算后换成一条新的边。由于点权均为 0 ，所以 2 度点的点权对新边的边权没有影响。

注意到当点权为 0 时 $w_{00} = w_{11}, w_{01} = w_{10}$ ，所以可以如题目描述中的那样，只用 s_e 和 d_e 来表示一条边 e 的边权。

对于删 1 度点操作，由于可以通过确定 1 度点的颜色来最大化其相邻边的贡献，所以可以把 1 度点相邻的边的 $\max\{s_i, d_i\}$ 直接计入答案。

由于叠合重边操作与缩 2 度点操作都是把两条边的权值通过计算变为一条新的边的权值，所以可以用运算的形式来表示它们。

6.1.1 定义运算

设一条边 e 的边权为 $v_e = \begin{bmatrix} s_e & d_e \end{bmatrix}^T$ 。

对于两条边的边权，定义 $+$ 运算的结果为叠合重边操作后的边权：

$$\begin{bmatrix} s_1 \\ d_1 \end{bmatrix} + \begin{bmatrix} s_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} s_1 + s_2 \\ d_1 + d_2 \end{bmatrix}$$

定义 $*$ 运算的结果为缩 2 度点操作后的边权：

$$\begin{bmatrix} s_1 \\ d_1 \end{bmatrix} * \begin{bmatrix} s_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} \max\{s_1 + s_2, d_1 + d_2\} \\ \max\{s_1 + d_2, d_1 + s_2\} \end{bmatrix}$$

可以发现，$+$ 运算和 $*$ 运算均满足交换律。

对于删 1 度点操作，可以定义初始答案 $ans = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$，然后遇到 1 度点时令 ans 为 $ans * v_e$（其中 e 为 1 度点相邻的边）。

由于每条边的边权都可以由最初的边权用 $+$ 运算和 $*$ 运算表示，所以整个问题的答案也可以用一个表达式来表示。于是可以建立这个表达式的表达式树，使问题重新变为树形动态规划的形式，进而用链分治的方法优化修改。

6.1.2 建立表达式树

以下是建立表达式树的过程：

定义 op_v 为表达式树上非叶子节点 v 上的运算符。

首先对于每一条边 e 新建表达式树上的对应节点 $v_e = \begin{bmatrix} s_e & d_e \end{bmatrix}^T$ [53]，对初始答案新建节点 $ans = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$。

然后使用算法七来对图 G 进行操作，直到 G 只剩下一个节点。在操作的过程中：

1. 若需要进行缩 2 度点操作，则在操作后对新建的边 e' 建立表达式树上的对应节点 $v_{e'}$，令原有的两条边在表达式树上的对应节点的父亲节点为 $v_{e'}$，并令 $op_{e'} =$ ‘$*$’；

2. 若需要进行叠合重边操作，则在操作后对新建的边 e' 建立表达式树上的对应节点 $v_{e'}$，令原有的两条边在表达式树上的对应节点的父亲节点为 $v_{e'}$，并令 $op_{e'} =$ ‘$+$’；

3. 若需要进行删 1 度点操作，则在操作后建立表达式树上的节点 v，令 ans 与删去的边在表达式树上的对应节点的父亲节点为 v，并定义 $op_v =$ ‘$*$’，最后令 ans 为 v。

6.1.3 分治

考虑沿用算法三的分治。

在表达式树中，每个非叶子节点都有两个子节点，所以设 $c_{v,0}, c_{v,1}$ 为 v 的两个子节点，设 sn_v 为 v 的重子节点的参数编号，即 $c_{v,sn_v} = son_v$ 为 v 的重子节点。

[53] 为叙述方便，这里不区分一个节点和其对应的权值。

为了快速维护重链上的信息，仍然沿用算法三中 g_v 的定义，即

$$f_v = g_v f_{son_v}$$

对于非叶子节点 v ，设 $son_v = \begin{bmatrix} x & y \end{bmatrix}^T, c_{v,1-sn_v} = \begin{bmatrix} z & w \end{bmatrix}^T$ 。

1. 若 $op_v = '+'$ ，则有

$$v = \begin{bmatrix} x+z \\ y+w \end{bmatrix}$$

所以有

$$g_v = \begin{bmatrix} z & -\infty \\ -\infty & w \end{bmatrix}$$

2. 若 $op_v = '*'$ ，则有

$$v = \begin{bmatrix} \max\{x+z, y+w\} \\ \max\{x+w, y+z\} \end{bmatrix}$$

所以有

$$g_v = \begin{bmatrix} z & w \\ w & z \end{bmatrix}$$

然后就可以套用算法三来解决问题了。唯一不同的是，算法三中的 g 不能暴力计算，必须通过维护增量的方式来维护 g 的值，而本算法中 g 的值可以暴力计算。

该算法时间复杂度为 $O(n + Q\log^2 n)$ ，可以通过子任务 $6, 7$ 。

期望得分 36 分。

最后可以开始考虑点权对问题的影响。

6.2 算法九 广义串并联图，带修改

考虑在算法八的基础上进行改进。

此时缩 2 度点操作和删 1 度点操作均需要考虑到点权，并且边权中需要记录 $w_{00}, w_{01}, w_{10}, w_{11}$ 四个值，即记

$$v_e = \begin{bmatrix} w_{00} \\ w_{01} \\ w_{10} \\ w_{11} \end{bmatrix}$$

此外，还需要记录点权，对于节点 u 记

$$v_u = \begin{bmatrix} b_u \\ w_u \end{bmatrix}$$

算法八中的 $+$ 运算和 $*$ 运算已经不适用，所以需要重新定义一下运算。

6.2.1 扩展运算

同样地，记 $+$ 运算的结果为叠合重边操作产生的新边的边权，即

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} a+x \\ b+y \\ c+z \\ d+w \end{bmatrix} \tag{33}$$

记 $*$ 运算的结果为缩 2 度点操作产生的新边的边权，而此时该边权与操作删去的 1 个节点和 2 条边的权值均有关。设 $e_0 = (u_0, u_1, (a,b,c,d)), e_1 = (u_1, u_2, (x,y,z,w)), u_1 = (\alpha, \beta)$ ，则有

$$*\left(\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, \begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}\right) = \begin{bmatrix} \max\{a+\alpha+x, b+\beta+z\} \\ \max\{a+\alpha+y, b+\beta+w\} \\ \max\{c+\alpha+x, d+\beta+z\} \\ \max\{c+\alpha+y, d+\beta+w\} \end{bmatrix} \tag{34}$$

对于删 1 度点操作，需要定义 $\oplus$ 运算的结果为操作修改的点（即被删去的点的相邻点）的点权，设 $e_0 = (u_0, u_1, (a,b,c,d)), u_0 = (\alpha, \beta), u_1 = (\gamma, \delta)$ 且 u_1 为操作选择的 1 度点，则有

$$\oplus\left(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, \begin{bmatrix} \gamma \\ \delta \end{bmatrix}\right) = \begin{bmatrix} \max\{\alpha+a+\gamma, \alpha+b+\delta\} \\ \max\{\beta+c+\gamma, \beta+d+\delta\} \end{bmatrix} \tag{35}$$

此外，还需要考虑将一条边反向[54]，即把 $(u, v, (w_{00}, w_{01}, w_{10}, w_{11}))$ 变为 $(v, u, (w_{00,}, w_{10}, w_{01}, w_{11}))$ ，所以定义 R 运算为

$$R\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} a \\ c \\ b \\ d \end{bmatrix} \tag{36}$$

6.2.2 建立表达式树

建立表达式树的方法与算法八类似，但不同的是，这里的 $*$ 运算和 $\oplus$ 运算是不满足交换律的，所以需要考虑子节点的顺序。此时定义 $c_{v,0}, c_{v,1}, c_{v,2}$ 依次为 $*$ 运算或 $\oplus$ 运算的三个参数，即 $v = *(c_{v,0}, c_{v,1}, c_{v,2})$ 或 $v = \oplus(c_{v,0}, c_{v,1}, c_{v,2})$ 。

使用算法七来对图 G 进行操作，直到 G 只剩下一个节点。在操作的过程中：

[54]在某些实现方法中需要进行该运算，也存在一些实现方法不需要进行该运算，而是使用一些别的方法来解决关于边的方向的问题。我的做法使用了该运算，所以我把它写在了这里。

1. 若需要进行缩 2 度点操作，则在操作后对新建的边 e' 建立表达式树上的对应节点 $v_{e'}$，令操作中删去的一个节点和两条边在表达式树上的对应节点的父亲节点为 $v_{e'}$，确定 $c_{v_{e'},0}, c_{v_{e'},1}, c_{v_{e'},2}$ 的值，并令 $op_{e'} =$ '$*$'；

2. 若需要进行叠合重边操作，则在操作后对新建的边 e' 建立表达式树上的对应节点 $v_{e'}$，令原有的两条边在表达式树上的对应节点的父亲节点为 $v_{e'}$，并令 $op_{e'} =$ '$+$'；

3. 若需要进行删 1 度点操作，则在操作后对修改的点 u' 建立表达式树上的对应节点 $v_{u'}$，令与操作相关的两个节点和一条边在表达式树上的对应节点的父亲节点为 $v_{u'}$，确定 $c_{v_{u'},0}, c_{v_{u'},1}, c_{v_{u'},2}$ 的值，并令 $op_{u'} =$ '$\oplus$'；

4. 注意操作的时候需要考虑边的方向，若需要改变边的方向，则在操作后建立表达式树上的节点 $v_{e'}$，令需要改变方向的边在表达式树上的对应节点的父亲节点为 $v_{e'}$，并令 $op_{e'} =$ 'R'。

6.2.3 分治

同样地，只要能够计算出 g 矩阵，就可以运用算法八的方法进行分治。

对于非叶子节点 v，

1. 若 $op_v =$ '$+$'，设

$$\begin{aligned} son_v &= \begin{bmatrix} x & y & z & w \end{bmatrix}^T \\ c_{v,1-sn_v} &= \begin{bmatrix} a & b & c & d \end{bmatrix}^T \end{aligned}$$

根据 (33)，

$$v = \begin{bmatrix} x+a \\ y+b \\ z+c \\ w+d \end{bmatrix}$$

所以

$$g_v = \begin{bmatrix} a & -\infty & -\infty & -\infty \\ -\infty & b & -\infty & -\infty \\ -\infty & -\infty & c & -\infty \\ -\infty & -\infty & -\infty & d \end{bmatrix}$$

2. 若 $op_v =$ 'R'，根据 (36) 则有

$$g_v = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ -\infty & -\infty & 0 & -\infty \\ -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & -\infty & 0 \end{bmatrix}$$

3. 若 $op_v =$ ‘$*$’，设

$$\begin{aligned} c_{v,0} &= \begin{bmatrix} a & b & c & d \end{bmatrix}^T \\ c_{v,1} &= \begin{bmatrix} \alpha & \beta \end{bmatrix}^T \\ c_{v,2} &= \begin{bmatrix} x & y & z & w \end{bmatrix}^T \end{aligned}$$

根据 (34)，

$$v = \begin{bmatrix} \max\{a+\alpha+x, b+\beta+z\} \\ \max\{a+\alpha+y, b+\beta+w\} \\ \max\{c+\alpha+x, d+\beta+z\} \\ \max\{c+\alpha+y, d+\beta+w\} \end{bmatrix}$$

3.1. 若 $sn_v = 0$，则有

$$g_v = \begin{bmatrix} \alpha+x & \beta+z & -\infty & -\infty \\ \alpha+y & \beta+w & -\infty & -\infty \\ -\infty & -\infty & \alpha+x & \beta+z \\ -\infty & -\infty & \alpha+y & \beta+w \end{bmatrix}$$

3.2. 若 $sn_v = 1$，则有

$$g_v = \begin{bmatrix} a+x & b+z \\ a+y & b+w \\ c+x & d+z \\ c+y & d+w \end{bmatrix}$$

3.3. 若 $sn_v = 2$，则有

$$g_v = \begin{bmatrix} a+\alpha & -\infty & b+\beta & -\infty \\ -\infty & a+\alpha & -\infty & b+\beta \\ c+\alpha & -\infty & d+\beta & -\infty \\ -\infty & c+\alpha & -\infty & d+\beta \end{bmatrix}$$

4. 若 $op_v =$ ‘$\oplus$’，设

$$\begin{aligned} c_{v,0} &= \begin{bmatrix} \alpha & \beta \end{bmatrix}^T \\ c_{v,1} &= \begin{bmatrix} a & b & c & d \end{bmatrix}^T \\ c_{v,2} &= \begin{bmatrix} \gamma & \delta \end{bmatrix}^T \end{aligned}$$

根据 (35)，

$$v = \begin{bmatrix} \max\{\alpha+a+\gamma, \alpha+b+\delta\} \\ \max\{\beta+c+\gamma, \beta+d+\delta\} \end{bmatrix}$$

4.1. 若 $sn_v = 0$ ，则有

$$g_v = \begin{bmatrix} \max\{a+\gamma, b+\delta\} & -\infty \\ -\infty & \max\{c+\gamma, d+\delta\} \end{bmatrix}$$

4.2. 若 $sn_v = 1$ ，则有

$$g_v = \begin{bmatrix} \alpha+\gamma & \alpha+\delta & -\infty & -\infty \\ -\infty & -\infty & \beta+\gamma & \beta+\delta \end{bmatrix}$$

4.3. 若 $sn_v = 2$ ，则有

$$g_v = \begin{bmatrix} \alpha+a & \alpha+b \\ \beta+c & \beta+d \end{bmatrix}$$

这里构造 g 矩阵的方式类似于构造一般的线性关系的转移矩阵。在代码实现的过程中，可以不需要对 sn_v 进行分类讨论来计算 g_v 的值，具体方式详见我的代码[55]的 tran 函数。

其余的步骤，均与算法八相同，这里不再重复描述。

该算法的时间复杂度为 $O(n + Q\log n)$ ，可以通过全部测试点，期望得分 100 分。

6.3 优缺点

6.3.1 优势

通过缩小问题规模的方式来解决问题，会使动态规划的过程更加直观，从而减少一些思维量。

相比于一般的链分治算法，使用表达式树上链分治的方法，不需要考虑一个节点有多个轻子节点的情况，从而降低代码实现的难度。

6.3.2 局限性

使用表达式树上链分治的方法通常只能支持全局或子树的查询，而难以支持链上问题的查询，且该方法通常需要在线段树上维护矩阵的乘积，这会带来不小的常数因子。

7 拓展

对于一些问题，需要统计最优方案的数量。此时可以把最优方案数同时记录在矩阵的元素中。令矩阵中的每个元素均为一个二元组 (a, cnt) ，其中 a 表示最优值，cnt 表示最优

[55] https://loj.ac/submission/433979

方案的数量，并定义

$$\begin{aligned}(a_1, cnt_1) + (a_2, cnt_2) &= (\max\{a_1, a_2\}, cnt_1[a_1 \geq a_2] + cnt_2[a_2 \geq a_1]) \\ (a_1, cnt_1) \times (a_2, cnt_2) &= (a_1 + a_2, cnt_1 \times cnt_2)\end{aligned}$$

不难发现，这里定义的加法和乘法满足交换律、结合律，并且乘法对加法满足分配律，所以能够使用矩阵乘法的形式来转移。

所以若《公园》或类似的题要求输出最优方案数（对大质数取模），那么仍然可以使用树链剖分线段树维护矩阵乘积的方法。

8 命题思路

这道题最初源于我对独立集问题的一些思考。在搜索最大独立集[56] 时有一个剪枝：若存在一个度数为 1 的节点，那么一定存在一个最优解包含这个度数为 1 的节点，于是可以删去这个度数为 1 的节点和其相邻节点，缩小问题规模。那么放在最大权独立集问题[57]上是否也有类似的方法呢?

可以发现，若 1 度点的点权大于等于其相邻节点的点权，那么前面所述的结论[58]仍然正确。否则我们可以先假定与那个 1 度点相邻的边不存在，并且选择该点。那么在之后的过程中若选择了与原来那个 1 度点相邻的节点，就意味着不能选择该 1 度点，于是可以把相邻节点的点权减去 1 度点的点权来处理这种情况。

钟知闲在 IOI2017 候选队的论文《浅谈信息学竞赛中的独立集问题》中提到了一种先搜索度数大于等于 3 的节点的算法，该算法的复杂度很大程度取决于枚举的度数大于等于 3 的节点个数。我发现通过删去 1 度点的方式可以在一定程度上减少需要枚举的度数大于等于 3 的节点的个数（如树上的独立集问题不需要枚举任何度数大于等于 3 的节点），于是想到，能否对度数为 2 的节点做类似的操作，来进一步减少需要枚举的度数大于等于 3 的节点个数，从而增加算法效率甚至优化复杂度呢？通过类似于删 1 度点方法的研究，我发现这是可以做到的，于是我设计了一个更加优秀一些的最大权独立集算法。

接着我想，这个算法在图满足什么性质的时候，不需要枚举任何度数大于等于 3 的节点呢？首先我就想到了仙人掌，显然仙人掌是满足条件的，但是我觉得，满足条件的图不仅仅有仙人掌，可以考虑更一般的情况。通过研究发现，这个算法不需要枚举任何度数大于等于 3 的节点，当且仅当图的每个连通块都是广义串并联图。这让我产生了《公园》命题的最初构想。

[56]最大独立集问题即给定图 $G = (V, E)$ ，求一个最大的 V 的子集 S 使得 E 中任意一条边的两个端点中均有至少一个不属于 S

[57]最大权独立集问题即给定图 $G = (V, E)$ 和权值函数 $w : V \to \mathbb{R}^+$ ，求一个 V 的子集 S 使得 E 中任意一条边的两个端点中均有至少一个不属于 S 且 $\sum_{u \in S} w(u)$ 最大

[58]即“一定存在一个最优解包含这个度数为 1 的节点”

然后我认为最大权独立集问题是一个经典问题，直接作为集训队互测的题目，在新意和难度上都有一些欠缺，于是就去思考如何去加强、改造这个问题。首先我想到的是把贡献答案的方式放到边上，这样会使题目看起来不那么经典，但是难度仍然偏低。然后考虑到现在出现了许多在树上甚至是仙人掌上动态修改，询问全局最优解的题目，于是我也试图在关于广义串并联图的动态修改上进行突破。

刚开始我考虑的是利用其对偶图的形态（类似一棵树）用链分治树链剖分线段树维护缩 2 度点后边的信息（此时我的问题保证了图点双连通并且没有点权）。但我发现若给出了对偶图，该问题就直接变成了一个树上问题，因为不需要把原图建出来就可以解决问题；若不给出对偶图，得到对偶图的形态是一个较为困难的过程。于是我把思路转向了直接维护缩 2 度点时图上信息（本题中即边权）的变化过程。我发现这样做就是在建立一棵表达式树。有了之前的思考，我很轻松地解决了问题。

但是我还不够满意，因为这棵表达式树其实表示的直接就是原图的串并联结构，而对于这一类的图，直接计算一些只和边有关的答案（比如计算串并联电路中两点之间的等效电阻）是一件相对容易的事情，因为“串并联”本身描述的就是边的关系。我为了使问题更一般化，开始考虑与点有关的问题。我发现建立表达式树并链分治的方法在加上点权后仍然适用，并且需要加上一定量的细节处理。于是《公园》就变成了最后的这个形式。

在子任务设置方面，子任务 $1,2,3$ 供选手分析朴素算法，同时为子任务 $4,5$ 提供思路。子任务 $4,5$ 与子任务 8 分别代表了解决问题的两个关键方向，子任务 $6,7$ 则从无需考虑点权这一特殊角度来简化问题细节，从而更方便地推出子任务 9 的标准解法。

9 总结

《公园》是一道具有一定难度的题目。本题设置了三方面的部分分，来引导选手走向正解，同时也为选手提供了多种突破的思路；但这也意味着若要拿较多的部分分，选手需要写多份的代码。正确通过此题需要清晰的编程思路和适当的处理细节的能力。

10 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢福州一中的陈颖老师给予的关心和指导。

感谢国家集训队教练张瑞喆提供的指导与帮助。

感谢清华大学的董克凡、陈俊锟、钟知闲学长，北京大学的杨昊翔学长给予的关心，教导与鼓励。

感谢福州一中的陈彦谔、陈雨昕同学在证明上的一些帮助。

感谢福州一中信息组的陈鸿基、陈彦谔、陈雨昕等同学为本文审稿。

参考文献

[1] Wikipedia, Biconnected component

[2] Wikipedia, Series-parallel graph

[3] 机械工业出版社, 《线性代数》

[4] 漆子超, 《分治算法在树的路径问题中的应用》

[5] 钟知闲, 《浅谈信息学竞赛中的独立集问题》, IOI2017 中国国家候选队论文

[6] 陈俊锟, 《〈神奇的子图〉命题报告及其拓展》, IOI2017 中国国家候选队论文

浅谈可追溯化数据结构

温州中学 孔朝哲

摘 要

本文主要介绍可追溯化数据结构，首先介绍了并查集的可追溯化，然后介绍了队列、栈、双端队列的可追溯化，接着介绍了优先队列的部分可追溯化，最后以优先队列的完全可追溯化为例介绍了一种通用的完全可追溯化方法。

1 引言

现在有一个数据结构，支持一些修改操作和询问操作。

我们在进行了一些修改操作之后，发现某一次修改操作输入错了，希望修改这个修改操作，然后继续。

因此就有了可追溯化数据结构。

2 定义

对于一个数据结构，

维护一个操作序列，支持在某个位置插入一个操作，或删除一个操作，

以及对按顺序执行这些操作后的状态进行一些询问。

实现这些功能，就称为原数据结构的**部分可追溯化 (Partial Retroactivity)**。

如果还支持对操作序列的任意一个前缀进行询问，就称为原数据结构的**完全可追溯化 (Full Retroactivity)**。

3 可追溯化并查集

下面先以并查集的可追溯化为例。

维护一个并查集的操作序列，支持以下操作：

1. Insert(t, union(x,y)) 在第 t 次操作后插入合并 x, y 所属集合的操作。

为了简化问题，这里不考虑删除操作。
(有删除操作时的部分可追溯化实际上就是动态图连通性问题)

3.1 部分可追溯化并查集

部分可追溯化并查集要求回答以下询问：

1. Query_sameset(x,y) 询问当前时刻 x 和 y 是否属于同一个集合。

这里认为所有操作发生的时刻互不相同且按操作序列上的顺序递增，当前时刻为 $+\infty$ 。

3.1.1 分析

union 操作具有交换律，因此直接上并查集就好了。

3.2 完全可追溯化并查集

完全可追溯化并查集要求回答以下询问：

1. Query_sameset(t,x,y) 询问在 t 时刻 x 和 y 是否属于同一个集合。

3.2.1 分析

lct 维护以时刻为关键字的最小生成树即可。

4 可追溯化队列

维护一个队列的操作序列，支持以下操作：

1. Insert(t, enqueue(x)) 在第 t 次操作后插入将元素 x 入队操作。
2. Insert(t, dequeue()) 在第 t 次操作后插入出队操作。
3. Delete(t) 删除第 t 次操作。

4.1 部分可追溯化队列

部分可追溯化队列要能回答以下询问：

1. Query(front()) 询问当前时刻队列的下一个将被弹出的元素。
2. Query(back()) 询问当前时刻队列中最后一个被加入的元素。

4.1.1 分析

不难发现队列的操作都是可以合并的，无需考虑一个 enqueue(x) 操作和一个 dequeue() 操作的顺序。可以先将所有的 enqueue(x) 都做一遍，再做所有的 dequeue() 操作，所得到的队列仍然一样。

那么就只用维护 enqueue(x) 操作的先后顺序，即每次高效地在第 t 个数后插入一个数 x 就行了。

平衡树？

太难写、复杂度太高。

其实链表就行了。

将 enqueue(x) 操作按照时间顺序存在链表里。并维护两个指针 F 和 B 分别表示队头元素和队尾元素，这样回答 Query(front()) 和 Query(back()) 只用分别返回 F 和 B 的值就行了。

enqueue(6) enqueue(5) enqueue(4) enqueue(3) enqueue(2) enqueue(1) dequeue() dequeue() 的例子如下：

一次 Insert(t, enqueue(x)) 对链表的修改：

1. 插入元素 x。
2. 若其成为了新的队尾则更新 B。
3. 如果在 F 后插入元素，则将 F 指向 F 的后继，否则不变。

一次 Delete(t) 操作对链表的修改，t 为 enqueue(x) 操作：

1. 删除元素 x。
2. 若其原来为队尾则更新 B。
3. 如果在 F 后删除元素，则将 F 指向 F 的前驱，否则不变。

一次 Insert(t, dequeue()) 操作对链表的修改：
将 F 指向 F 的前驱。
一次 Delete(t) 操作对链表的修改，t 为 dequeue() 操作：
将 F 指向 F 的后继。
一次操作复杂度是 $O(1)$ 的。

4.2 完全可追溯化队列

完全可追溯化队列要能回答以下询问：

1. Query(t, front()) 询问 t 时刻后队列的下一个将被弹出的元素。
2. Query(t, back()) 询问 t 时刻后队列中最后一个被加入的元素。

4.2.1 分析

由于 enqueue(x) 操作和 dequeue() 操作是独立的，所以维护两棵平衡树 T_e 和 T_d 将两种操作分开处理。

T_e 按时间顺序存下所有 enqueue(x) 操作。

T_d 按时间顺序存下所有 dequeue() 操作。

修改操作直接在对应的平衡树上进行对应的操作即可，复杂度 $O(log\ m)$。

回答 Query(t, back()) 询问：

在 T_e 中找到 t 之前的最后一个操作。

回答 Query(t, front()) 询问：

先在 T_d 中找到 t 之前有多少个 dequeue() 操作，设次数为 k。

在 T_e 中找到第 $k + 1$ 次插入的值，可以用平衡树上二分。

一次询问复杂度是 $O(log\ m)$ 的。

5 可追溯化栈

维护一个栈的操作序列，支持以下操作：

1. Insert(t, push(x)) 在第 t 次操作后插入将元素 x 入栈操作。
2. Insert(t, pop()) 在第 t 次操作后插入出栈操作。
3. Delete(t) 删除第 t 次操作。

由于部分可追溯化和完全可追溯化的单次操作复杂度都是 $O(log\ m)$ 的，所以下面只考虑完全可追溯化栈。

5.1 完全可追溯化栈

完全可追溯化栈要能回答以下询问：

1. Query(t, top()) 询问 t 时刻后的栈顶元素。

5.1.1 分析

考虑什么样的元素才能是最后的 top()。

可以发现一个操作 Insert(t', push(x)) 要是最后的 top() 元素当且仅当 $(t', t]$ 中的 push 操作数量与 pop 操作数量一样。

所以问题就变成了如何快速找到这样一个操作。

用平衡树来解决。用一个平衡树按时间顺序维护所有操作，push(x) 操作记为 $+1$，pop() 操作记为 -1。

这样问题就变成了求最后一个后缀和为 1 的点。

由于数值只有 ± 1，故区间 $[l_t, r_t]$ 内的后缀和肯定是一段连续的值 $[l_x, r_x]$，所以对平衡树每个节点记录 *min* 和 *max* 表示子树内后缀和的最小值、最大值，以及子树内权值和 *sum*，查找后缀和为 1 的点就可以在平衡树上二分了。

单次修改和询问的复杂度是 $O(logm)$ 的。

6 可追溯化双端队列

维护一个双端队列的操作序列，支持以下操作：

1. Insert(t, pushL(x)) 在第 t 次操作后插入将元素 x 从左端加入双端队列操作。
2. Insert(t, pushR(x)) 在第 t 次操作后插入将元素 x 从右端加入双端队列操作。
3. Insert(t, popL()) 在第 t 次操作后插入弹出左端第一个数操作。
4. Insert(t, popR()) 在第 t 次操作后插入弹出右端第一个数操作。
5. Delete(t) 删除第 t 次操作。

由于部分可追溯化和完全可追溯化的单次操作复杂度都是 $O(log\ m)$ 的，所以下面只考虑完全可追溯化双端队列。

6.1 完全可追溯化双端队列

完全可追溯化双端队列要能回答以下询问：

1. Query(t, left()) 询问 t 时刻后双端队列中最左端元素的值。
2. Query(t, right()) 询问 t 时刻后双端队列中最右端元素的值。

6.1.1 分析

既然有了完全可追溯化栈，能否将双端队列拆成两半，对 pushL(x) 和 popL() 操作维护一个左端栈，对 pushR(x) 和 popR() 操作维护一个右端栈?

由于双端队列一侧的操作如果不合法了会影响到另一侧，所以最后一个后缀和为 1 的点不一定就是答案，譬如说下面这个例子：

pushR(1) popL() pushL(2) Query(t, right())

当然，这个错误算法也给了我们一些启发。一个值肯定是被一次 pushL(x) 操作或者一次 pushR(x) 操作加入双端队列中的，如果能得到 pushL(x) 操作中最有可能是答案的和 pushR(x) 操作中最有可能是答案的再判断一下就行了。

考虑一种手写双端队列的方法。建一个数组 a，初始时 $L = 1, R = 0$。

1. pushL(x) 操作对应 a[- -L] = x
2. pushR(x) 操作对应 a[++R] = x
3. popL() 操作对应 ++L
4. popR() 操作对应 - -R

这样一次询问只要知道 i 和 a[i] 就行了。

维护两棵平衡树 T_l 和 T_r，T_l 按时间存下 pushL(x) 操作和 popL() 操作，权值分别为 $+1$ 和 -1，T_r 一样。求 t 时刻的 i 的值只用在 T_l 或 T_r 中求一下前缀和就行了。

a[i] 的值肯定是最后一次 pushL(x) 后 $L = i$ 的操作或者最后一次 pushR(x) 后 $R = i$ 的操作。这两个都可以在平衡树上二分出后缀和为 k 的点，取两者较晚的一次操作即可。

单次修改和询问的复杂度是 $O(logm)$ 的。

可以发现回答询问时并没有用到最左端点和最右端点这个限制，所以回答双端队列中任意一个元素的值也是可以的。

7 可追溯化优先队列

维护一个优先队列（堆）的操作序列，支持以下操作：

1. Insert(t, insert(k)) 在第 t 次操作后插入将元素 k 入堆操作。
2. Insert(t, delete_min()) 在第 t 次操作后插入弹出最小元素的操作。
3. Delete(t) 删除第 t 次操作。

7.1 部分可追溯化优先队列

部分可追溯化优先队列要能回答以下询问：

1. Query() 询问当前时刻队列的最小元素。

7.1.1 分析

记 Q_t 表示 t 时刻队列内的元素，我们只要能够询问 Q_{now} 即可。

考虑 Insert(t, insert(k)) 对 Q_{now} 的影响。可以发现他相当于是在 Q_{now} 中插入了

$$\max\{k, k' \mid k' \text{ deleted at time} \geq t\}$$

问题在于删除了哪些元素并不好进行维护。

我们称一个时刻 t 是一个**桥 (bridge)**，当且仅当 $Q_t \subseteq Q_{now}$。

若 t' 是 t 时刻前的第一个桥，则：

$$\begin{aligned} &\max\{k' \mid k' \text{ deleted at time} \geq t\} \\ &= \max\{k' \notin Q_{now} \mid k' \text{ inserted at time} \geq t'\} \end{aligned} \tag{37}$$

类似地，我们考虑 Insert(t, delete_min()) 的影响。若 t' 是 t 后面的第一个桥，则影响即为删除

$$\min\{k' \in Q_{now} \mid k' \text{ inserted at time} \leq t'\} \tag{38}$$

而对于撤销一个 insert(k) 操作，如果 k 在 Q_{now} 中，那么影响就是删除 k，否则就是删除

$$\min\{k' \in Q_{now} \mid k' \text{ inserted at time} \leq t'\}$$

类似地，撤掉一个 delete_min() 的操作的影响即为

$$\max\{k' \notin Q_{now} \mid k' \text{ inserted at time} \geq t'\}$$

于是我们可以进行 Q_{now} 的维护。

考虑使用平衡树维护 Q_{now}。为了处理修改，我们需要另外的平衡树。

我们用另外一棵平衡树维护插入。对于每个节点 u，我们存下

$$\max\{k' \notin Q_{now} \mid k' \text{ inserted in } u\text{'s subtree}\}$$

以及

$$\min\{k' \in Q_{now} \mid k' \text{ inserted in } u\text{'s subtree}\}$$

我们还需要一个平衡树用来求出桥。我们存储 Insert 的操作，并如下赋权：

1. 对于操作 insert(k)，且 $k \in Q_{now}$，我们赋权 0；
2. 对于操作 insert(k)，且 $k \notin Q_{now}$，我们赋权 $+1$；
3. 对于操作 delete_min()，我们赋权 -1。

我们在每个节点上存储子树和，前缀和的最小值，以及前缀和的最大值。

因为桥等价于前缀和等于 0，所以我们对于每次修改，可以通过树上二分在 $O(\log_2 m)$ 的时间内找到 t 前面的一个桥。

在求出桥之后，我们可以通过 1 式和 2 式来求出这次操作对 Q_{now} 的影响，从而实时维护 Q_{now}。

求出对 Q_{now} 的影响后，可以同时维护另外两棵平衡树的信息。

时间复杂度：单次修改、询问均为 $O(\log_2 m)$。

7.2 完全可追溯化优先队列

堆的完全可追溯化比较困难，因此使用的方法都是比较通用的，适用范围比较广的方法。

有一个通用的方法可以将 full 的复杂度做到 partial 的复杂度乘上 $O(\sqrt{m})$ 。

对操作序列分块，每 $O(\sqrt{m})$ 个操作分成一块。

对于每个块，将这个块之前的所有操作组成的操作序列用 partial 的方法进行维护。

插入，删除和询问都可以非常暴力的做。

通过一个叫 hierarchical checkpointing 的方法，可以做到单次插入删除 $O(\log^2 m)$ ，询问 $O(\log^2 m \log\log m)$ 。

用一棵替罪羊树维护整个操作序列。

对于每个结点，对这个结点代表的子树组成的操作序列，维护部分可追溯化堆，将维护的结果记作两棵平衡树 Q_{now} 和 Q_{del} ，分别包含没被删除的元素和被删除的元素 (如果某次删除的元素的不存在，则认为是无穷大)。

插入结点时对每个祖先都执行一次插入。

如果导致一个子树不平衡则需要进行重构 (重构的实现后面会讲)。

删除是类似的。

为了实现删除，可以打删除标记，也可以只在叶节点存储信息，就是类似线段树的结构。

询问时，需要合并 $O(\log m)$ 个部分可追溯化堆，然后对合并的结果进行查询。

引理：将两个部分可追溯化堆 Q_1, Q_2 合并的结果记作 Q_3 ，则有

$$Q_{3,now} = Q_{2,now} \bigcup max\text{-}A\{Q_{1,now} \cup Q_{2,del}\}$$

$$Q_{3,del} = Q_{1,del} \bigcup min\text{-}D\{Q_{1,now} \cup Q_{2,del}\}$$

这里 $A = |Q_{1,now}|$, $D = |Q_{2,del}|$ ， $max\text{-}C\{S\}$ 表示集合 S 中前 C 大的元素。

为了快速合并，我们用多棵平衡树的并来表示 Q_{now} 和 Q_{del} 。

定义可并的部分可追溯化堆 Q^k ，表示 Q^k_{now} 和 Q^k_{del} 都是用不超过 k 棵平衡树表示的。

那么如果现在要合并 Q^k_1 和 Q^k_2 ，

令 $T = Q_{1,now} \bigcup Q_{2,del}$ ，记 $T_i (1 \le i \le 2 \times k)$ 为 T 中第 i 棵平衡树，

令 $x = max\text{-}A\{T\}$ ，

将 T_i 根据 x 分裂成两棵平衡树 $T_{i,<}$ 和 $T_{i,>}$

则 $Q_{3,now} = Q_{2,now} \bigcup T_{1,>}, \cdots, T_{2k,>}$ $Q_{3,del} = Q_{1,del} \bigcup T_{1,<}, \cdots, T_{2k,<}$

这样我们就由 Q^k_1 和 Q^k_2 得到了 Q^{3k}_3 。

时间复杂度 $O(k \log m)$。

求解 $max\text{-}A\{T\}$ 有点麻烦，这里不再赘述，有兴趣的同学可以自己去看论文。

假设现在要合并 k 个部分可追溯化堆 $Q_{1..k}$ ，记合并的结果为 Q_* 。

如果直接分治，使用上面的方法合并，那么 $Q_{*,now}$ 和 $Q_{*,now}$ 会用 $O(3^{\log_2 k}) = O(k^{\log_2 3})$ 棵平衡树来表示。

引理：存在 a_i 和 a'_i ，使得

$$Q_{*,now} = \bigcup_i max\text{-}a_i Q_{i,now} \bigcup_i max\text{-}a'_i Q_{i,del}$$

因此 $Q*, now$ 和 $Q*, del$ 都可以用 $2k$ 棵平衡树来表示。

因此合并 k 个部分可追溯化堆的时间为 $O(k \log k \log m)$ 。

询问时的复杂度为 $O(\log^2 m \log \log m)$ 。

关于子树重构，只需要使用归并操作来合并 $Q_{1,now}$ 和 $Q_{2,del}$ 。对于大小的 m 的子树重构的复杂度为 $O(m \log m)$ 。

所以插入删除的复杂度为均摊 $O(\log^2 m)$ 。

8 鸣谢

感谢中国计算机学会提供学习和交流的平台。

感谢舒春平老师对我的培养和指导。

感谢吴思扬同学，戴言同学对本文的帮助。

感谢父母对我的理解和关心。

参考文献

[1] ERIK D. DEMAINE, JOHN IACONO and STEFAN LANGERMAN. Retroactive Data Structures.

[2] Erik D. Demaine , Tim Kaler, Quanquan Liu, Aaron Sidford, and Adam Yedidia. Polylogarithmic Fully Retroactive Priority Queues via Hierarchical Checkpointing.

浅谈杨氏矩阵在信息学竞赛中的应用

成都市第七中学 袁方舟

摘 要

杨氏矩阵又称杨表，是一个特殊的矩阵。本文将介绍一些关于杨表的性质，使得杨表可以和许多组合问题建立联系，在组合计数问题的求解中发挥作用。希望本文能够让读者深入了解杨表，体会到杨表的美妙之处。

1 前言

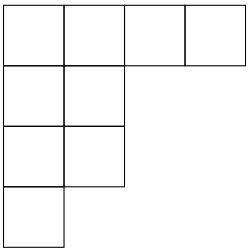
在数学中，杨表 (Young tableaux)，又称杨氏矩阵，是组合表示理论和舒伯特演算领域的常用工具。在对称群和一般线性群性质的研究中，杨表提供了一个方便的方式来描述的它们的群表示。杨表由剑桥大学数学家阿尔弗雷德 · 杨在 1900 年提出。接着于 1903 年被弗罗贝尼乌斯应用于对称群的研究中。

在信息学竞赛中，很早便出现了考察杨表的钩子公式的题目。遗憾的是，许多选手对于杨表的认识只止步于基本定义和钩子公式的套用上。本文将从杨表和排列的对应关系讲起，第 4 节将谈论杨表和对称矩阵的关系，第 5 节的内容是杨表和信息学竞赛中耳熟能详的问题——最长上升子序列问题的联系。第 6 节则会谈论广为人知的杨表钩子公式以及其证明。第 7 节是杨表和网格图路径的一些联系。第 8 节是对半标准杨表计数公式的讲解。同时，本文也会将杨表和信息学竞赛中的题目联系起来，展示他们之间的关联。

2 定义

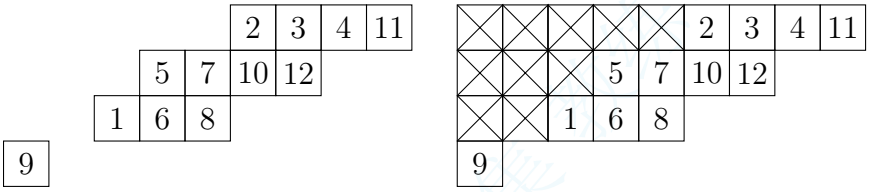
令 $\lambda = (\lambda_1, \ldots, \lambda_m)(\lambda_1 \geq \lambda_2 \geq \ldots \lambda_m > 0)$ 是 N 的一个整数拆分，记为 $\lambda \vdash N$。则一个形状为 λ 的杨图 (young diagram) 为一个 m 行，第 i 行有 λ_i 列的表格。一个形状为 λ 的标准杨表 (standard young tableaux) 则是将 1 到 N 这 N 个正整数填到杨图中，并使得每一行从左往右和每一列从上往下都是严格递增的。若满足同一行中的数字**非严格递增**，且同一列中的数字严格递增的杨表被称作是半标准杨表 (semistandard young tableaux)。将杨表中每个数字出现的次数记录下来，可得到一个序列，该序列被视为杨表的“权重”。比如，标准杨表的权重便是 $(1, 1, \ldots, 1)$，因为在标准杨表中，1 到 N 的正整数恰好各出现一次。

如下图，分别是大小为 9 的杨图和标准杨表。



1	4	7	8
2	5		
3	9		
6			

对于两个整数拆分 $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$ 和 $\mu = (\mu_1, \mu_2, \ldots, \mu_{m'})$，设对于每个 i 都有 $\mu_i \leq \lambda_i$（设 $i > m$ 时 $\lambda_i = 0$，$i > m'$ 时 $\mu_i = 0$）。那么一个形状为 λ/μ 的斜杨图 (skew young tableaux) 就是 λ 的杨图中扣去 μ 的杨图。同理，若满足同一行中的数字**非严格递增**，且同一列中的数字严格递增，则该斜杨表被称作半标准的；若满足同一行中的数字严格递增，且同一列中的数字严格递增，且表内 1 到 N 各出现一次，则该斜杨表被称作标准的。如下图所示，是一个形状为 $(9,7,5,1)/(5,3,2)$ 的标准斜杨表。



			2	3	4	11		
		5	7	10	12			
	1	6	8					
9								

3 杨表和排列的对应关系

排列总是有一些难以捉摸的组合性质，比如“一个排列的最长上升子序列”。从这一节开始，我们将会看到杨表和排列的对应关系，以及排列性质是如何借用杨表直观表现出来的。

下面的算法，被称作 RSK 算法，由 Robinson,Schensted and Knuth 提出。这个算法提供了一个将杨表和排列联系起来的途径。本节讨论的杨表都是标准杨表。

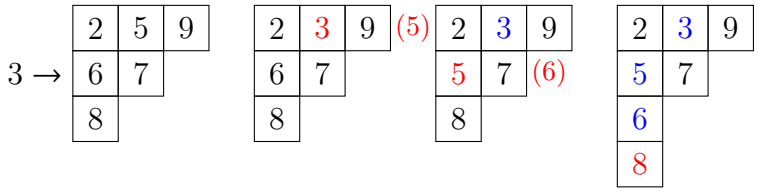
插入算法 令 S 是一个杨表，定义 $S \leftarrow x$ 表示将 x 从第一行插入杨表中，具体做如下操作：

(1) 在当前行中找到最小的比 x 大的数 y。

(2) 如果找到了就用 x 去替换 y，移到下一行，令 $x \leftarrow y$ 重复操作 (1)。

(3) 如果找不到的话就把 x 放在该行末尾并退出。记 x 在第 s 行 t 列，(s,t) 必定是一个边角。一个格子 (s,t) 是边角当且仅当 $(s+1,t)$ 和 $(s,t+1)$ 都不存在格子。

例如，将 3 插入杨表 $(2,5,9)(6,7)(8)$ 中：



同理，定义对列插入算法，描述的便是把 x 从第一列开始插入，仿照上述过程构建的杨表。

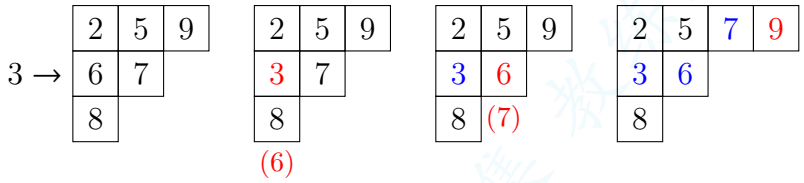
对列插入算法 令 S 是一个杨表，定义 $x \to S$ 表示将 x 从第一列插入杨表中，具体做如下操作：

(1) 在当前列中找到最小的比 x 大的数 y。

(2) 如果找到了就用 x 去替换 y，移到下一列，令 $x \leftarrow y$ 重复操作 (1)。

(3) 如果找不到的话就把 x 放在该列末尾并退出。记 x 在第 s 行 t 列，(s,t) 必定是一个边角。

例如，将 3 使用对列插入算法插入杨表 $(2,5,9)(6,7)(8)$ 中：



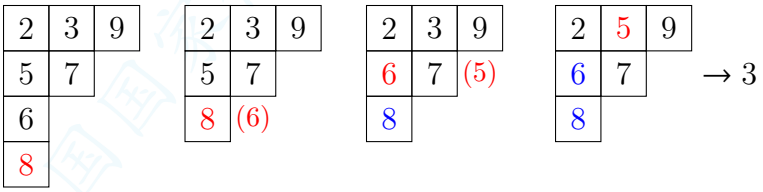
删除算法 输入 s,t，表示将杨表 S 的第 s 行第 t 列删去。注意 (s,t) 一定是一个边角。令 $x = S_{s,t}$，具体做如下操作：

(1) 若现在是第一行，退出并删除 x。

(2) 在上一行中找到最大的比 x 小的数 y。

(3) 用 x 替换 y，并移到上一行，令 $x \leftarrow y$ 重复操作 (2)。

我们试图对杨表 $(2,3,9)(5,7)(6)(8)$ 执行删除 $(4,1)$ 操作



引理 3.1. *$S \leftarrow x$ 和 $x \to S$ 一定是一个杨表。*

证明. 考虑 $S \leftarrow x$($x \to S$ 同理) 首先，我们可以注意到如果有两个连续的行长度相等，那么必定不会使第二行变长。这是因为第二行的末尾一定比第一行所有数大。再来考虑递增的性质，可以发现每行仍然是递增的。由于列递增，每次替换后只会往下或者左下走，可以发现它一定会比上一行同列的数大，且比下一行同列的数小。于是列递增也是成立的。 □

引理 3.2. *对 S 运行删除算法后一定是一个杨表。*

证明. 首先，由于删除的是边角，所以同样不会有第一行比第二行短的情况。再来考虑递增的性质，可以发现每行仍然是递增的。由于列递增，每次替换后只会往上或者右上走，可以发现它一定会比上一行同列的数大，且比下一行同列的数小。于是列递增也是成立的。 □

引理 3.3. *上述删除算法是插入算法的逆运算。逆运算指的是如果记下每次插入算法返回的 $(s_1,t_1),(s_2,t_2)\dots(s_n,t_n)$ 等，那么反过来，对这个操作之后的杨表依次删除 $(s_n,t_n)\dots(s_2,t_2),(s_1,t_1)$，得到的杨表和操作之前的相同。*

证明. 考虑对于一个杨表 S，插入一个数 x_1 返回 (s,t)，然后立刻删去 (s,t)。我们要证的是，假设插入的时候每行操作的 x 分别是 $x_1,x_2,\dots x_m$，那么删除的时候每行操作的 x 必定也是 $x_1,\dots x_m$。考虑归纳，首先第 m 行一定都是 x_m。假设插入算法到第 i 行时，要插入的数为 x_i，令在第 $i-1$ 行插入的数为 x_{i-1}。注意到 x_i,x_{i-1} 在第 $i-1$ 行里满足 $\dots S_{i-1,k}<x_{i-1}<x_i<S_{i-1,k+2}\dots$ 的关系（如果定义超出右边界的位置为 inf，超出左边界的位置为 $-\inf$）。那么在删除算法到第 i 行时，找最大的比 x_i 小的数就一定能找到 x_{i-1}。所以命题得证。 □

定义 3.1. *对于一个排列 $X=(x_1,x_2\dots x_k)$，定义 P_X 为杨表 $(\dots(x_1\leftarrow x_2)\leftarrow x_3\dots)\leftarrow x_k$。定义 Q_X 为*记录表*，即在对 P_X 插入 x_i 时将下标的 i 插入相应位置 (注意，这里没有运行任何算法) 并维持 Q_X 和 P_X 的形状相同。显然，Q_X 也是一个杨表。*

如下图所示

$X=1,5,3,2,6,7,4\rightarrow$ $P_X=$

1	2	4	7
3	6		
5			

$Q_X=$

1	2	5	6
3	7		
4			

定理 3.1. *(Robinson–Schensted correspondence)*

一个 1 到 n 个数的排列 $X=x_1,\dots,x_n$ 和一对相同形状的标准杨表是一一对应的。也就是说，

$$\sum_{\lambda\vdash n} f_\lambda^2 = n!$$

证明. 首先，对于一个排列 X，通过插入算法一定可以找到唯一一对对应的标准杨表 P_X 和 Q_X。反过来，对于一对标准杨表 P_X 和 Q_X，一定可以通过每次将 Q_X 中最大值所在坐标输入删除算法得到唯一一个排列。由于插入算法和删除算法互为逆算法，所以可以得到一一对应关系。 □

4 杨表和对称矩阵

一个大小为 $W\times H$ 的非负整数矩阵 A，可以看做是一些数对 (i,j)，其中 (i,j) 出现 A_{ij} 次。定义数对之间的比较 $(a,b)\le(c,d)$ 为真当且仅当 $a<c$ 或 $a=c$ 且 $b\le d$ 时成立。那

么一个矩阵就可以唯一表达成一个数对的排列，也就是一个序列对，如下图所示。这让我们想到杨表也可以和矩阵有对应关系。杨表的对称性可以从中直观地感受到。

$$\begin{pmatrix}0&1&0\\0&0&2\\1&1&0\end{pmatrix}\to\begin{pmatrix}1&2&2&3&3\\2&3&3&1&2\end{pmatrix}$$

下面对于这个序列对进行讨论。仿照杨表和排列的对应关系，如果直接对着这个序列对运行 RSK 算法可以得到如下的基本定理

定理 4.1. *一对满足 $(u_1,v_1)\le(u_2,v_2)\le\cdots\le(u_k,v_k)$ 的相同长度序列对*

$$\begin{pmatrix}u_1&u_2&\dots&u_{k-1}&u_k\\v_1&v_2&\dots&v_{k-1}&v_k\end{pmatrix}$$

和一对相同形状的半标准杨表 (P_X,Q_X) 一一对应。此时 P_X 是对第二行运行插入算法后得到的杨表，即 $(\dots(v_1\leftarrow v_2)\leftarrow v_3\dots)\leftarrow v_k$，而 Q_X 每次记录下的则是 u_i，如下图所示。

$$\begin{pmatrix}1&2&2&3&3\\2&3&3&1&2\end{pmatrix}\to\begin{array}{|c|c|c|}\hline1&2&3\\\hline2&3\\\cline{1-2}\end{array}\quad\begin{array}{|c|c|c|}\hline1&2&2\\\hline3&3\\\cline{1-2}\end{array}$$

我们试图对 RSK 算法进行更深刻的观察。考虑如下这样一个序列对

$$X=\begin{pmatrix}u_1&u_2&\dots&u_{k-1}&u_k\\v_1&v_2&\dots&v_{k-1}&v_k\end{pmatrix}$$

且满足 $u_1\le u_2\le\cdots\le u_k$。考虑这个序列对里哪些数会被放在 P_X 和 Q_X 的第一行中。记 $P_X^{(i,j)}$ 为 P_X 的第 i 行第 j 列的数，同理定义 $Q_X^{(i,j)}$。对于 $P_X^{(1,1)}$，首先会变成 v_1，此时 $Q_X^{(1,1)}$ 被赋值为 u_1；然后 $P_X^{(1,1)}$ 每次遇到比他小的都会变化，$Q_X^{(1,1)}$ 则一直不变。这启发我们得到一个结论

引理 4.1. *如果把 X 分成若干组 $A_X^{(i)}$，定义 $A_X^{(i)}=\{(u_{a_{(i,1)}},v_{a_{(i,1)}}),\dots,(u_{a_{(i,k_i)}},v_{a_{(i,k_i)}})\}$，其中 $a_{(x,y)}$ 是指 $A_X^{(x)}$ 的第 y 个数对原序列对中的位置。第 1 组 $A_X^{(1)}=\{(u_{a_{(1,1)}},v_{a_{(1,1)}}),\dots,(u_{a_{(1,k_1)}},v_{a_{(1,k_1)}})\}$ 是记录下从 1 开始的一个序列，满足每个 $v_{a_{(1,i)}}$，在序列对中都严格小于它前面的所有数，即 $\forall j,1\le j\le a_{(1,i)}-1,v_j>v_{a_{(1,i)}}$。然后把这个序列删去后重复操作，得到 $A_X^{(2)},A_X^{(3)}\dots$，直到序列为空。那么对于所有 t，有 $P_X^{(1,t)}=v_{a_{(t,k_t)}},Q_X^{(1,t)}=u_{a_{(t,1)}}$*

$$X=\begin{pmatrix}1&3&5&6&8&8\\7&9&2&5&2&7\end{pmatrix}\to\begin{array}{c}A_X^{(1)}=\{(1,7),(5,2)\}\\A_X^{(2)}=\{(3,9),(6,5),(8,2)\}\\A_X^{(3)}=\{(8,7)\}\end{array}\to\begin{array}{c}P_X^{(1,*)}=\{2,2,7\}\\Q_X^{(1,*)}=\{1,3,8\}\end{array}$$

有了这个引理后，我们可以得到一个 P_X 和 Q_X 的关系。

定理 4.2. 定义 X 为序列对满足 $(u_1,v_1)\le(u_2,v_2)\le\cdots\le(u_k,v_k)$，那么定义 X^{-1} 为将这个序列对按照 (v_i,u_i) 排序之后的序列对。

$$X=\begin{pmatrix}u_1&u_2&\dots&u_{k-1}&u_k\\v_1&v_2&\dots&v_{k-1}&v_k\end{pmatrix}\to X^{-1}=\begin{pmatrix}v_{i_1}&v_{i_2}&\dots&v_{i_{k-1}}&v_{i_k}\\u_{i_1}&u_{i_2}&\dots&u_{i_{k-1}}&u_{i_k}\end{pmatrix}((v_{i_1},u_{i_1})\le(v_{i_2},u_{i_2})\le\cdots\le(v_{i_k},u_{i_k}))$$

那么有 $(P_X,Q_X)=(Q_{X^{-1}},P_{X^{-1}})$，如下图所示。

$$X=\begin{pmatrix}2&3&4&5&5&5&6\\1&5&3&2&6&7&4\end{pmatrix}\to\quad P_X=\begin{array}{|c|c|c|c|}\hline1&2&4&7\\\hline3&6\\\cline{1-2}5\\\cline{1-1}\end{array}\qquad Q_X=\begin{array}{|c|c|c|c|}\hline2&3&5&5\\\hline4&6\\\cline{1-2}5\\\cline{1-1}\end{array}$$

$$X^{-1}=\begin{pmatrix}1&2&3&4&5&6&7\\2&5&4&6&3&5&5\end{pmatrix}\to\ P_{X^{-1}}=\begin{array}{|c|c|c|c|}\hline2&3&5&5\\\hline4&6\\\cline{1-2}5\\\cline{1-1}\end{array}\qquad Q_{X^{-1}}=\begin{array}{|c|c|c|c|}\hline1&2&4&7\\\hline3&6\\\cline{1-2}5\\\cline{1-1}\end{array}$$

证明. 首先我们考虑第一行。对于 $A_X^{(1)}$，其中的元素满足

$$\begin{pmatrix}u_{a_{(1,1)}}<u_{a_{(1,2)}}<u_{a_{(1,3)}}<\cdots<u_{a_{(1,k_1)}}\\v_{a_{(1,1)}}>v_{a_{(1,2)}}>v_{a_{(1,3)}}>\cdots>v_{a_{(1,k_1)}}\end{pmatrix}$$

注意到数对比较的定义，所以 A 中不会存在相等的 u_i。那么 $A_{X^{-1}}^{(1)}$ 则是

$$\begin{pmatrix}v_{a_{(1,k_1)}}<v_{a_{(1,k_1-1)}}<v_{a_{(1,k_1-2)}}<\cdots<v_{a_{(1)}}\\u_{a_{(1,k_1)}}>u_{a_{(1,k_1-1)}}>u_{a_{(1,k_1-2)}}>\cdots>u_{a_{(1)}}\end{pmatrix}$$

很容易证明 $A_X^{(1)}$ 和 $A_{X^{-1}}^{(1)}$ 是同构的，即 X 和 X^{-1} 中被选入 $A^{(1)}$ 的数对是一样的。首先 $A_{X^{-1}}^{(1)}$ 不会比 $A_X^{(1)}$ 多一个或者少一个，然后可以利用这两个的对称性易证不可能会有元素被替代。

假设存在 $(v_{a(1,x)},u_{a(1,x)}),(v_{a(1,y)},u_{a(1,y)})(x>y),(a,b)$ 满足 $v_{a(1,x)}<a<v_{a(1,y)},u_{a(1,x)}>b$ 且 $b\le u_{a(1,y)}$。那么在 A_X 中就会有 $b\le u_{a(1,y)}$ 且 $a<v_{a(1,y)}$，这样 A_X 一定会优先选择 (b,a)，矛盾。考虑 A 的构造算法，通过归纳可以发现 $A_X=A_{X^{-1}}$。由于 P,Q 的第一行分别会取右下角和左上角，那么我们就证明了 P_X,Q_X 的第一行分别和 $Q_{X^{-1}},P_{X^{-1}}$ 相等。

可以发现在取走属于第一行的数之后，这些集合变成了

$$\begin{pmatrix}u_{a_{(i,2)}}&u_{a_{(i,3)}}&\dots&u_{a_{(i,k_i)}}\\v_{a_{(i,1)}}&v_{a_{(i,2)}}&\dots&v_{a_{(i,k_i-1)}}\end{pmatrix},\begin{pmatrix}v_{a_{(i,k_i-1)}}&v_{a_{(i,k_i-2)}}&\dots&v_{a_{(i,1)}}\\u_{a_{(i,k_i)}}&u_{a_{(i,k_i-1)}}&\dots&u_{a_{(i,2)}}\end{pmatrix}$$

这样我们就得到了被挤到第二行的数对。请注意，这个 A_X 并不是第二行的 A_X，但是我们可以发现这两个序列对仍然是符合关系的。所以可以继续归纳下去，证明完毕。 □

4.1 对称矩阵

定理 4.3. 一个 $W\times H$ 的矩阵 M，假设每行的和分别是 $(r_1,r_2,\ldots,r_W)$，每列的和分别是 $(c_1,c_2,\ldots,c_H)$。定义 X_M 为矩阵 M 对应的序列对，定义 P_M,Q_M 为 P_{X_M},Q_{X_M}。那么 M 和一对半标准杨表 P_M,Q_M 一一对应，其中 P_M 中 i 出现了 c_i 次，Q_M 中 i 出现了 r_i 次。

这个定理很容易说明，考虑上述对于矩阵的编码即可。

定理 4.4. 对于一个 $W\times H$ 的矩阵 M，定义 M^T 为一个 $H\times W$ 的矩阵满足 $\forall i,j,M^T_{ij}=M_{ji}$。定义**对称矩阵**指一个 $n\times n$ 的矩阵 M 满足 $M=M^T$。假设每行的和分别是 $(r_1,r_2,\ldots,r_W)$，那么对称矩阵 M 和一个半标准杨表 P_M 一一对应，其中 P_M 中 i 出现了 r_i 次。

$$\begin{pmatrix}0&1&0&0&0\\1&0&0&2&0\\0&0&0&0&0\\0&2&0&0&0\\0&0&0&0&1\end{pmatrix}\rightarrow\quad X=\begin{pmatrix}1&2&2&2&4&4&5\\2&1&4&4&2&2&5\end{pmatrix}\rightarrow P_X=Q_X=\begin{array}{|c|c|c|c|}\hline 1&2&2&5\\\hline 2&4&4\\\cline{1-3}\end{array}$$

证明. 定义 M 对应的排列对为 X_M。那么考虑 X_M 和 X_{M^T} 之间的关系。可以发现 $X_M^{-1}=X_{M^T}$。

$$M=\begin{pmatrix}0&1&0\\0&0&2\\1&1&0\end{pmatrix}\rightarrow X_M=\begin{pmatrix}1&2&2&3&3\\2&3&3&1&2\end{pmatrix}\rightarrow X_M^{-1}=\begin{pmatrix}1&2&2&3&3\\3&1&3&2&2\end{pmatrix}$$

$$M^T=\begin{pmatrix}0&0&1\\1&0&1\\0&2&0\end{pmatrix}\rightarrow X_{M^T}=\begin{pmatrix}1&2&2&3&3\\3&1&3&2&2\end{pmatrix}$$

由定理 4.2 可得，X_M 和 X_M^{-1} 的杨表对是反过来的。这说明如果 X 满足 $X_M=X_M^{-1}$ 的话，X_M 就和一个杨表 P_X 一一对应。表现到矩阵上就是满足 $M=M^T$ 的矩阵 M 与一个半标准杨表一一对应。于是证毕。 □

定理 4.5. 在对称矩阵 M 中，M 的 **迹** 与其对应的半标准杨表 P_M 中长度为奇数的列数相等。一个矩阵的 **迹** 记做 $tr(M)$，且 $tr(M)=\sum_{i=1}^{n}M_{ii}$。

证明. 对于一个对称矩阵 M 转化而来的排列 X，每个 $A_X^{(i)}$ 一定是回文的，这是因为如果 (u,v) 在 $A_X^{(i)}$ 出现，那么 (v,u) 也会出现。设 $u<v$。假设 A_X 选择了 (u,v)，然后它选择了一个 (a,b) 而不是选择 (v,u)，那么 (a,b) 满足 $u<a<v$ 且 $v>b,b<u$。那么它前面一定会存在一个 (b,a) 满足 $a<v,b<u$，所以 A_X 会选择 (b,a) 而不是选择 (u,v)，矛盾。

一个形如 (u,u) 的对只能在每个 $A_X^{(i)}$ 中出现一次。所以一定会有 $tr(M)$ 个 $A_X^{(i)}$ 长度为奇数。由归纳就可以证明结论。 □

4.2 对合排列

如果一个 $n\times n$ 的 01 矩阵满足每行每列都存在 1，且每行每列的和都是 1，那么这个矩阵就暗示着一个排列。这样的一个矩阵对应的杨表就是标准杨表。

定义 4.1. *对于一个 1 到 n 的排列 $X=x_1,\ldots,x_n$，定义 X^{-1} 为一个排列，满足 $\forall i, X^{-1}_{x_i}=i$。定义满足 $X=X^{-1}$ 的排列为对合排列(involution permutation)。*

可以发现，这个操作正好就是矩阵的翻转操作。所以我们可以得到如下的结论：

定理 4.6. *大小为 n 的标准杨表个数为对合数(involution numbers)。对合数指的是满足 $X=X^{-1}$ 的排列 X 数量。且大小为 n 的标准杨表和每个这样的排列一一对应。*

这个定理给出了大小为 n 的标准杨表个数公式：$a(n)=(n-1)a(n-2)+a(n-1), a(0)=1, a(1)=1$。这是对合数的计数公式，由于每个对合排列都是由一些大小为 2 的环和一些点构成，每次枚举最后一个点和谁配对就可以得到这个公式。

定理 4.7. *对于对合排列 X，X 中满足 $X_i=i$ 的 i 的数量和它所对应标准杨表奇数列的个数相等。*

$$X=1,5,3,6,2,4,8,7\rightarrow\ P_X=Q_X=$$

1	2	4	7
3	6	8	
5			

这就是定理 4.5 在排列上的体现。

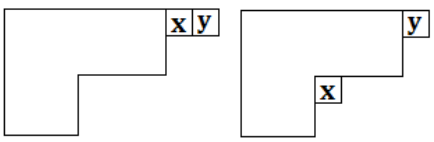
5 杨表与最长上升子序列

定义一个序列的 LIS(Longest Increasing Subsequence) 为其最长上升子序列，同理，定义 LDS(Longest Decreasing Subsequence) 为其最长下降子序列。

定理 5.1. *P_X 中第一行长度即为排列 X 的 LIS 长度。*

注意，P 的第一行并不一定是 LIS 本身，所以不能直接利用杨表性质做“LIS 划分”之类的问题。

引理 5.1. $(x\rightarrow S)\leftarrow y=x\rightarrow(S\leftarrow y)$



分 3 种情况讨论，分别是 S 最大,x 最大或是 y 最大。y 最大的情况，可以由图知；同理可得 x 最大的情况。S 最大的情况可以通过讨论最大值的位置解决，感兴趣的可以查看完整证明过程 [10]，这里不再赘述。

引理 5.2. *对于一个排列 X 和它产生的杨表 P_X，若 X^R 是 X 的翻转，那么 X^R 产生的杨表 P_{X^R} 即为 P_X 交换行列得到。*__注意，这个引理对 Q_X 不成立__

$X = 1,5,7,2,8,6,3,4 \rightarrow$ $P_X =$

1	2	3	4
5	6	8	
7			

$X^R = 4,3,6,8,2,7,5,1 \rightarrow$ $P_{X^R} =$

1	5	7
2	6	
3	8	
4		

证明. 定义 $P(x_1 \dots x_n) = (\dots(x_1 \leftarrow x_2)\dots) \leftarrow x_n, P'(x_1 \dots x_n) = x_1 \rightarrow (\dots(x_{n-1} \rightarrow x_n)\dots)$。那么 $P(x_1 \dots x_n)$ 就是 X 生成的杨表，$P'(x_1 \dots x_n)$ 就是 X_R 生成的杨表翻转行列得到。首先可以注意到 $x_1 \leftarrow x_2 = x_1 \rightarrow x_2$。如果 $x_1 < x_2$ 则它们都是 $x_1 x_2$，否则它们都是 $\begin{matrix} x_2 \\ x_1 \end{matrix}$，这说明 $n \le 2$ 时 P, P' 相等。设 $P(x_1 \dots x_n) = P'(x_1 \dots x_n)$ 对于该引理成立，那么

$$\begin{aligned} P(x_1 \dots x_n, x_{n+1}) &= P'(x_1 \dots x_n) \leftarrow x_{n+1} \\ &= (x_1 \rightarrow P'(x_2 \dots x_n)) \leftarrow x_{n+1} \\ &= x_1 \rightarrow (P'(x_2 \dots x_n) \leftarrow x_{n+1}) (\text{引理 5.1}) \\ &= x_1 \rightarrow (P(x_2 \dots x_n) \leftarrow x_{n+1}) \\ &= x_1 \rightarrow P'(x_2 \dots x_{n+1}) (\text{由前提条件可得}) \\ &= P'(x_1 \dots x_{n+1}) \end{aligned}$$

□

定理 5.2. *杨表 P_X 中的第一列长度即为排列 X 的 LDS 长度。*

证明. 由定理 5.1 和引理 5.2 立刻可以得到，因为把序列翻转后原先的最长上升子序列就变成了最长下降子序列 □

5.1 最长的 k-LIS 子序列

定义 k-LIS 序列为 LIS 长度不超过 k 的序列。同理，定义 k-LDS 序列表示 LDS 长度不超过 k 的序列。显然，最长的 1-LIS 子序列就是该序列的 LDS，这正是杨表的第一列；我

们于是可以作出如下猜想：是否前 k 列长度就是最长的 k-LIS 子序列长度呢？实际上，这是正确的。

引理 5.3. *对于序列中三个连续的数 $x,y,z(x<y<z)$，如果它们在序列中不是按照 $(x,y,z)(z,y,x)$ 出现的话，交换 x,z 后序列的最长 k-LIS 子序列长度不变。*

证明. 考虑 (z,x,y)。注意到交换 x,z 不会让答案增加。令 G 为原排列的最长 k-LIS 子序列。如果 G 不同时包含 x,z，那么交换 x,z 后长度不会变短。若 G 同时包含 z,x，那么 G 一定包含 y。否则把 x 换成 y 就可以得到一个更长的 k-LIS 子序列。这时可以发现交换 x,z 后不会增加 G 中 LIS 长度。

所以 $(z,x,y)\to(x,z,y)$ 不会使答案变化。同理，还有 $(x,z,y)\to(z,x,y),(y,x,z)\to(y,z,x),(y,z,x)\to(y,x,z)$。 □

引理 5.4. *对于一个排列 X 和它产生的 m 行杨表 P，令排列 X^* 为 $(P_{m,1}\ldots,P_{m,\lambda_m},P_{m-1,1}\ldots,P_{1,1}\ldots P_{1,\lambda_1})$（即将杨表从下往上每行依次写在后面）。那么 X 一定可以通过引理 5.3 中的交换操作转化成 X^*。*

对于这个定理，考虑归纳，模拟杨表插入算法即可证明。这里不再赘述。

定理 5.3. *对于杨表 P，前 k 列的长度总和就是这个排列最长 k-LIS 子序列长度。同理，前 k 行的长度总和就是这个排列的最长 k-LDS 子序列长度。*

证明. 由引理 5.4 可得，每个排列都可以化成 $(P_{m,1}\ldots.P_{m,\lambda_m},P_{m-1,1}\ldots.,P_{1,1}\ldots P_{1,\lambda_1})$ 的形式。所以最长 k-LIS 子序列长度（另一个结论可以由翻转排列得到）可以表示成

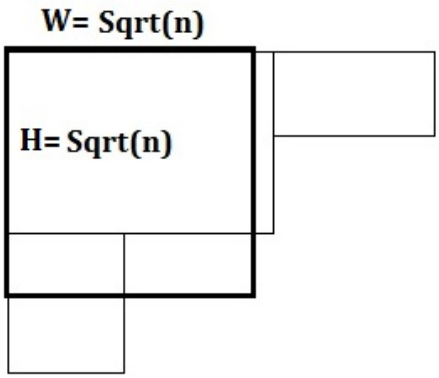
$$F(k)=\sum_{i=1}^{m}\min(k,\lambda_i)$$

可以很快发现这个东西就等于前 k 列的长度总和。 □

例题 1. *(CTSC2017 最长上升子序列)*

有一个长为 n 的数列 b，对于序列 $B_m=(b_1,b_2\ldots,b_m)$，设 C 是 B_m 的子序列，且 C 的最长上升子序列的长度不超过 k，则询问 C 的长度最大值。Q 次询问 k,m，$Q\le 2\times10^5,k\le m\le n\le 50000$。

分析先思考对于一个询问怎么做，多个询问考虑使用扫描线的方法。这样我们就需要维护每个前缀的杨表。如果使用以上结论，可以发现问题变成了如何快速维护杨表前 k 列的长度之和。如果直接维护，复杂度是 $O(n^2\log n)$ 的不能接受。考虑维护前 $\sqrt{n}$ 列和前 $\sqrt{n}$ 行。可以发现，杨表一定不会完全覆盖这个 $W\times H$ 的矩形。如果 $K\le W$，那么可以直接得答案；如果 $K>W$，那么大于 W 的部分一定在 H 行内。所以可以考虑如何同时维护前 $\sqrt{n}$ 列和前 $\sqrt{n}$ 行。由引理 5.2 可得，将这个排列翻转一下就可以得到杨表的翻转，所以只需要再同时维护 $-A_i$ 即可，复杂度为 $O(n\sqrt{n}\log n)$。



5.2 LIS 计数

定理 5.4. *最长上升子序列长度为 α，最长下降子序列长度为 β 的排列数就是对于列数 α，行数 β 的杨表数平方和。即：*

$$A = \sum_{\text{行数为}\beta,\text{列数为}\alpha,\lambda\vdash n} f_\lambda^2$$

证明. 使用定理 5.1(首行 LIS)，定理 5.2(首列 LDS) 和定理 3.1(杨表和排列对应) 即可证明。 □

如果这个序列里允许重复数字的出现，则依照 RSK 算法的运行可以得到如下的结果。

定理 5.5. *最长**非严格上升**子序列长度为 α，最长**严格**下降子序列长度为 β 的排列数是对于列数 α，行数 β 的相同形状半标准杨表数和标准杨表数的乘积。即：*

$$A = \sum_{\text{行数为}\beta,\text{列数为}\alpha,\lambda\vdash n} g_{\lambda/\mu} f_\lambda$$

其中 $g_{\lambda/\mu}$ 表示权重为 μ 的半标准杨表数。特别地，当 $\mu = (1,1,\dots 1)$ 时 $g_{\lambda/\mu} = f_\lambda$。

例题 2. *(BJWC2018 最长上升子序列)*

现在有一个长度为 n 的随机排列，求它的最长上升子序列长度的期望。为了避免精度误差，你只需要输出答案模 998244353 的余数。($n \le 28$)

分析令 $E(n)$ 为随机序列中最长上升子序列的期望长度。即

$$E(n) = \frac{1}{n!} \sum_{w\text{是一个长为}n\text{的排列}} LIS(w)$$

这道题的标算使用了 $O(n^2 2^n)$ 的状压 DP。考虑状压每个长度最小结尾数组（就是杨表第一行）即可，由于本题 $n \le 28$ 太大还需要本地打表。实际上，使用定理 5.4，可以发现我们只需要枚举每种形状的杨表并计算它们的个数就可以了。杨表形状数是 $p(n)$，其中 $p(n)$ 为 n 的整数拆分数。那么 $O(n^2 p(n))$ 的时间得出结果。$p(n)$ 的增长速度远小于 2^n，甚至能做到 $1s$ 内跑出 $n \le 63$ 的答案。

6 杨表与钩子公式

从本节开始，我们将要看到杨表的一系列计数公式，以及它们是如何使用各种组合事物推演而来的。这一节讲的就是众所周知的钩子公式，钩子公式在代数，概率论等方面都有应用。这里也给出了标准杨表钩子公式的一个证明；实际上，对于斜杨表的钩子公式也是存在的 [13]，它使用了一个“Excited tableaux”概念来定义。

令 $\lambda = (\lambda_1, \ldots, \lambda_m)$ 为 n 的一个整数拆分。在形状为 λ 的杨图中，定义 $h_\lambda(i,j)$ 为形如 $(a,b)(a = i, b \geq j$ 或 $a \geq i, b = j)$ 的格子数量。这个 h 被称作为钩子 (hook) 函数。正因为这个函数的计数区域形似一个钩子，所以才称作为钩子公式。那么形状为 λ 的标准杨表数量为

定理 6.1. *(Frame, Robinson and Thrall formula)*

$$f_\lambda = \frac{n!}{\prod h_\lambda(i,j)} \tag{39}$$

如果只使用 λ 表达，那么钩子公式可以写成

$$n! \frac{\prod_{1 \leq j < k \leq m} (\lambda_j - j - \lambda_k + k)}{\prod_{i=1}^{m} (\lambda_i + m - i)!}$$

6.1 钩子公式的一个证明

我们考虑使用归纳法证明。令 λ 为 n 的一个整数拆分，μ 为 $(n-1)$ 的一个整数拆分。记 $\mu \to \lambda$ 当且仅当形状为 λ 的杨图包含形状为 μ 的杨图。那么我们要证明就是

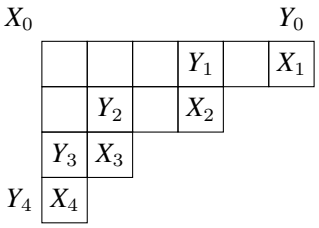
$$\frac{n!}{\prod_{s \in \lambda} h_\lambda(s)} = \sum_{\mu \to \lambda} \frac{(n-1)!}{\prod_{s \in \mu} h_\mu(s)}$$

或者是

$$\sum_{\mu \to \lambda} \frac{\prod_{s \in \lambda} h_\lambda(s)}{\prod_{s \in \mu} h_\mu(s)} = n \tag{40}$$

对于一个 i 行 j 列的格子 $c = (i,j)$，令 $ct(c) = i - j$。对于 $(6,4,2,1)$ 的杨图，每个格子的 ct 为

0	-1	-2	-3	-4	-5
1	0	-1	-2		
2	1				
3					



令 λ 里有 m 个边角，设这些边角从上往下为 $X_i=(\alpha_i,\beta_i)(i\in[1,m])$。令 $Y_i=(\alpha_i,\beta_{i+1})(i\in[0,m])$。这里我们令 $\alpha_0=\beta_{m+1}=\beta_0=0,X_0=(\alpha_0,\beta_0)=(0,0)$，注意这个点是在图外的。再令 $A(c)$ 为格子 c 往下走最远的点，$B(c)$ 为格子 c 往右走最远的点。那么 $h_\lambda(c)=ct(A(c))-ct(B(c))+1$。

令 $x_i=ct(X_i),y_i=ct(Y_i)$，我们随即发现

$$\sum_{i=0}^{m}x_i=\sum_{i=0}^{m}y_i \tag{41}$$

这是因为 $\sum_{i=0}^{m}(x_i-y_i)=\sum_{i=0}^{m}(\alpha_i-\beta_i-(\alpha_i-\beta_{i+1}))=\beta_{m+1}-\beta_0=0$。

我们的证明分为 3 个步骤，分别是 (4),(5),(6) 这三个等式

$$\begin{aligned}\sum_{\mu\to\lambda}\frac{\prod_{s\in\lambda}h_\lambda(s)}{\prod_{s\in\mu}h_\mu(s)} &= -\sum_{i=1}^{m}\frac{\prod_{j=0}^{m}(x_i-y_j)}{\prod_{j=1,j\neq i}^{m}(x_i-x_j)} &(42)\\ &= -\frac{1}{2}\sum_{i=0}^{m}(x_i^2-y_i^2) &(43)\\ &= n &(44)\end{aligned}$$

对 (4) 式的证明 设 $\mu^{(i)}$ 是通过加一个边角 X_i 变成 λ 的 μ，这其中只有和 X_i 同行或同列的格子的 h_λ 值会发生变化。对于同列的格子，注意到只有形如 $L_j=(\alpha_j,\beta_i)(1\le j<i)$ 和 $M_j=(\alpha_j+1,\beta_i)(0\le j<i)$ 的格子会对答案造成影响。可以得到

$$\begin{aligned}h_\lambda(M_j) &= ct(A(M_j))-ct(B(M_j))+1=x_i-y_j\\ h_{\mu^{(i)}}(L_j) &= ct(A(L_j))-ct(B(L_j))+1=x_i-x_j\end{aligned}$$

同理对于同行的格子令 $L_j=(\alpha_i,\beta_j)(i<j\le m)$ 和 $M_j=(\alpha_i,\beta_{j+1}+1)(i\le j\le m)$，则

$$\begin{aligned}h_\lambda(M_j) &= y_j-x_i\\ h_{\mu^{(i)}}(L_j) &= x_j-x_i\end{aligned}$$

所以有

$$\begin{aligned}\sum_{\mu\to\lambda}\frac{\prod_{s\in\lambda}h_\lambda(s)}{\prod_{s\in\mu}h_\mu(s)} &= \sum_{i=1}^{m}\frac{\prod_{s\in\lambda}h_\lambda(s)}{\prod_{s\in\mu^{(i)}}h_{\mu^{(i)}}(s)}\\ &= \sum_{i=1}^{m}\frac{\prod_{j=0}^{m}h_\lambda(M_j)}{\prod_{j=1,j\neq i}^{m}h_{\mu^{(i)}}(L_j)}\\ &= \sum_{i=1}^{m}\frac{\prod_{j=0}^{i-1}(x_i-y_j)\prod_{j=i}^{m}(y_j-x_i)}{\prod_{j=1}^{i-1}(x_i-x_j)\prod_{j=i+1}^{m}(x_j-x_i)}\\ &= -\sum_{i=1}^{m}\frac{\prod_{j=0}^{m}(x_i-y_j)}{\prod_{j=1,i\neq j}^{m}(x_i-x_j)}\end{aligned}$$

对 (5) 式的证明 熟悉拉格朗日插值的人可能会很熟悉这个形式。这里我们沿用多项式的想法来证明它。定义

$$\begin{aligned} P(t) &= -\sum_{i=1}^{m} \frac{\prod_{j=0}^{m}(x_i - y_j)}{\prod_{j=1,i\neq j}^{m}(x_i - x_j)} \prod_{j=1,j\neq i}^{m}(t - x_j) \\ Q(t) &= \prod_{j=0}^{m}(t - y_j) \end{aligned}$$

可以发现 $P(t)$ 是一个 $m-1$ 次多项式，而我们的式子就是它的 t^{m-1} 项系数，而 $Q(t)$ 是一个 $m+1$ 次多项式。现在考虑 $P(t)+Q(t)$ 这个多项式的性质，可以发现它是一个 $m+1$ 次多项式，而且在 $t = x_s$ 的时候为 0。注意到 x_i 互不相同，那么也就是说，存在一个 α 使得

$$\begin{aligned} P(t) + Q(t) &= (t-\alpha)\prod_{i=1}^{m}(t - x_i) \\ \Rightarrow P(t) &= (t-\alpha)\prod_{i=1}^{m}(t - x_i) - \prod_{j=0}^{m}(t - y_j) \\ &= (-\alpha - \sum_{i=1}^{m} x_i + \sum_{i=0}^{m} y_i)t^m + (\alpha \sum_{i=1}^{m} x_i + \sum_{1\le i<j\le m} x_i x_j - \sum_{0\le i<j\le m} y_i y_j)t^{m-1} + \dots \end{aligned}$$

由于 $P(t)$ 是一个 $m-1$ 次多项式，所以它的 t^m 的系数应该为 0。由 (3) 式可得，$\alpha = 0$。注意到 $x_0 = 0$，也就是说，$P(t)$ 的 t^{m-1} 项系数可以被写成

$$\sum_{0\le i<j\le m} x_i x_j - y_i y_j$$

我们马上发现

$$\begin{aligned} \sum_{0\le i<j\le m} x_i x_j - y_i y_j &= \frac{1}{2}((\sum_{i=0}^{m} x_i)^2 - (\sum_{i=0}^{m} y_i)^2 - \sum_{i=0}^{m} x_i^2 - \sum_{i=0}^{m} y_i^2) \\ &= -\frac{1}{2}\sum_{i=0}^{m}(x_i^2 - y_i^2) \end{aligned}$$

对 (6) 式的证明 这个结论的证明很简单，将 x_i, y_i 用 α_i, β_i 带入可得

$$\begin{aligned} -\frac{1}{2}\sum_{i=0}^{m}(x_i^2 - y_i^2) &= -\frac{1}{2}\sum_{i=0}^{m}((\alpha_i - \beta_i)^2 - (\alpha_i - \beta_{i+1})^2) \\ &= \sum_{i=0}^{m}(\alpha_i\beta_i - \alpha_i\beta_{i+1}) \end{aligned}$$

而这个式子即计算图大小的式子，于是得证，即钩子公式得证。

6.2 在杨图上的随机游走

定义在杨图上的随机游走过程为：从一个位置开始，每次选择同行靠右或是同列靠下（当然不能选自己）的格子，然后转移到那里，直到到达一个边角为止。

定理 6.2. *(Hook Walk)*

在杨图 $\lambda=(\lambda_1,\lambda_2,\ldots,\lambda_m)$ 中，从 *(1,1)* 随机游走到一个边角 *(r,s)* 的概率是

$$\frac{1}{n}\prod_{i=1}^{r-1}\frac{h_\lambda(i,s)}{h_\lambda(i,s)-1}\prod_{j=1}^{s-1}\frac{h_\lambda(r,j)}{h_\lambda(r,j)-1}$$

如果对于每行每列分别设一个权重，随机选择时会通过这个权重来定义每个位置的概率，具体就是该位置的权重/所有能选的位置权重之和，其中如果这个格子是在下面的那么权重定义为该行的权重，否则定义为该列的权重。

定理 6.3. *(Weighted Hook Walk)*

在杨图 $\lambda=(\lambda_1,\lambda_2,\ldots,\lambda_m)$ 中，定义 $\lambda'=(\lambda'_1,\ldots)$，满足 $\lambda'_i=|\{j|\lambda_j\geq i\}|$。定义 $y_1,y_2,\ldots,y_m$ 为每列的权重，$x_1,x_2,\ldots,x_{\lambda'_1}$ 为每行的权重，那么从 *(1,1)* 随机游走到一个边角 *(r,s)* 的概率是

$$\frac{x_r y_s}{\sum_{(p,q)\in[\lambda]}x_p y_q}\prod_{i=1}^{r-1}\left(1+\frac{x_i}{x_{i+1}+\cdots+x_r+y_{s+1}+\cdots+y_{\lambda_i}}\right)\times\prod_{j=1}^{s-1}\left(1+\frac{y_j}{x_{r+1}+\cdots+x_{\lambda'_j}+y_{j+1}+\cdots+y_s}\right)$$

对于这个公式的证明，可以查看参考文献 [5]。

7 杨表与网格图路径

在网格图上，一条路径 P 是从一个整点 $\mathcal{A}=(A_1,A_2)$ 到另一个整点 $\mathcal{E}=(E_1,E_2)$，规定每次只能往右或者往上走。这两点之间所有路径可以表示为 $P(\mathcal{A}\to\mathcal{E})$。两条路径相交则是指这两条路径上有公共点。

显然的，从 (A_1,A_2) 到 (E_1,E_2) 的路径条数为

$$|P(\mathcal{A}\to\mathcal{E})|=\binom{E_1+E_2-A_1-A_2}{E_1-A_1}$$

7.1 卡特兰数

定理 7.1. $2\times n$ 的标准杨表个数为卡特兰数：$C_n=\frac{1}{n+1}\binom{2n}{n}$

证明. C_n 的组合意义是 (0,0) 到 (n,n)，每次让横坐标加 1 或纵坐标加 1，问有多少种方法使得过程中横坐标一定不小于纵坐标。假设 X_i 是横坐标到达 i 的时间，Y_i 是纵坐标到达 i 的时间。那么一定有 $Y_1<Y_2<\cdots<Y_n, X_1<X_2<\cdots<X_n$，而且有 $\forall i, X_i<Y_i$。这其实就是一个 $2\times n$ 的标准杨表。 □

例题 3. *(LOJ 6051「雅礼集训 2017 Day11」PATH)*

给定 n 和 $\{a_i\}$，满足 $a_1 \geq a_2 \geq \cdots \geq 0$，求出在 n 维空间中从 $(0,0\ldots.0)$ 走到 $(a_1,a_2,\ldots,a_n)$，每一步使某一维坐标增加 1 的方案中随机选出一种，满足经过的所有点 $(x_1,x_2,\ldots,x_n)$ 都满足 $x_1 \geq x_2 \cdots \geq x_n$ 的概率，答案模 1004535809 输出。$a_i, n \leq 500000$。

分析同样的，如果记录每一维到达 i 的时间，那么可以发现每条路径可以转化为一个 $\lambda = (a_1,a_2\ldots a_n)$ 的标准杨表。使用钩子公式，如果令 $r_i = \lambda_i + n - i$，式子可以转化成：

$$Ans = \frac{\frac{n!}{\prod_{i=1}^{n} r_i!}\prod_{1\leq j<k\leq n}(r_j - r_k)}{\frac{n!}{\prod_{i=1}^{n} a_i!}} = \prod_{i=1}^{n}\frac{a_i!}{r_i!}\prod_{1\leq j<k\leq n}(r_j - r_k)$$

直接计算的话复杂度为 $O(n^2)$。但是可以发现 $r_i \leq n + \max a_i$，所以可以使用 fft 计算出每种 $r_j - r_k$ 出现了多少次，然后使用快速幂计算。所以总复杂度为 $O(n\log n)$。

7.2 非交叉网格路径

定理 7.2. *(Lindstrom–Gessel–Viennot lemma)*

平面上有 $2n$ 个点 $(A_1^{(1)},A_2^{(1)})\ldots(A_1^{(n)},A_2^{(n)})(E_1^{(1)},E_2^{(1)})\ldots(E_1^{(n)},E_2^{(n)})$。令 $\mathcal{A}_i = (A_1^{(i)},A_2^{(i)}), \mathcal{E}_i = (E_1^{(i)},E_2^{(i)}), P_i = P(\mathcal{A}_i \to \mathcal{E}_i)$。设 $A_1^{(1)} \leq A_1^{(2)} \leq \cdots \leq A_1^{(n)}, A_2^{(1)} \geq A_2^{(2)} \geq \cdots \geq A_2^{(n)}, E_1^{(1)} \leq E_1^{(2)} \leq \cdots \leq E_1^{(n)}, E_2^{(1)} \geq E_2^{(2)} \geq \cdots \geq E_2^{(n)}$，对于一个集合 $S = (P_1,P_2,\ldots P_n)$，每个 P_i 里选择一条路径使得路径两两不相交的方案数为

$$Ans = \det(f(\mathcal{A}_i,\mathcal{E}_j))_{i,j=1}^{n}$$

其中 $f(x,y)$ 指从 x 到 y 的方法数。如果只有必须向右或向上走的限制，那么答案就是

$$\det\left(\binom{E_1^{(j)} + E_2^{(j)} - A_1^{(i)} - A_2^{(i)}}{E_1^{(j)} - A_1^{(i)}}\right)_{i,j=1}^{n}$$

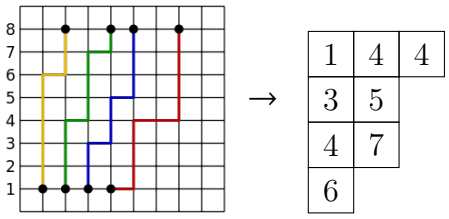
证明. 对这个行列式的证明可以这样理解：首先我们选择用容斥来解决这个问题。对于从 A_i, A_j 开始的路径，如果它们相交了，就将这两个路径在最后的相交点进行反转。考虑最后每个 A_i 匹配了哪个 E_j，这是一个排列 σ（不等式的关系保证了如果 $i<j, \sigma(i) > \sigma(j)$ 那么它们必定相交）。每种排列的容斥系数即 $\mathrm{sgn}(\sigma)$(-1 的逆序对数量次方)，即：

$$\sum_{\sigma}\mathrm{sgn}(\sigma)\prod_{i} f(\mathcal{A}_i,\mathcal{E}_{\sigma(i)})$$

这正是行列式的定义式。 □

非交叉网格路径可以和杨表建立双射关系。

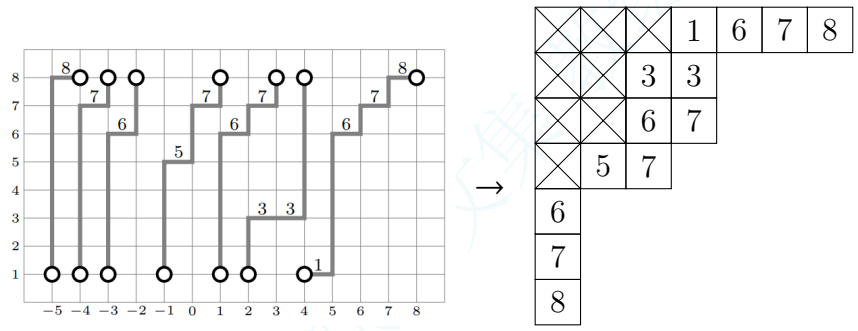
定理 7.3. 一个半标准杨表 $\lambda = (\lambda_1,\ldots,\lambda_m)(\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m)$ 且数都在 $[1,n]$ 内，可以和一个 $\mathcal{A} = \{(-i,1)\}, \mathcal{E} = \{(\lambda_i - i, n\}$ 的不相交交叉路径集合建立双射关系。



证明. 将从 $(-i,1)$ 出发的路径视作杨表的第 i 行，每次往右走视为在这一行的末尾写下一个数字，且这个数字为这一步所在纵坐标。我们要证明的是半标准杨表的性质：一行内非严格递增，一列内严格递增。第一个性质直接可以由该路径的性质得到。若假设第二个性质不成立，那么必定会产生交点，这是因为同列处横坐标相差不超过 1。反过来，也可以证明一个半标准杨表一定能写成若干个不相交路径的形式。 □

上述定理稍微拓展一下可以得到斜杨表和非交叉路径的关系

定理 7.4. *一个半标准斜杨表 λ/μ 且数都在 $[1,n]$ 内，可以和一个 $\mathcal{A}=\{(\mu_i-i,1)\},\mathcal{E}=\{(\lambda_i-i,n\}$ 的不相交交叉路径集合建立双射关系。*



7.3 行列式公式

利用上述杨表和非交叉网格路径的联系，我们得到了杨表计数的行列式公式

定理 7.5. *(Aitken's Formula)*
标准斜杨表 λ/μ 的个数为

$$f_{\lambda/\mu}=(\sum_i \lambda_i-\mu_i)!\left|\frac{1}{(\lambda_i-i-\mu_j+j)!}\right|_{i,j=1}^{|\lambda|}$$

这里，定义 $\frac{1}{a!}$ 在 $a<0$ 的时候等于 0，在 $a=0$ 的时候等于 1。

证明. 考虑杨表和非交叉网格路径的对应关系。只考虑其中一个对应排列 σ。标准斜杨表的定义在非交叉网格路径里的体现就是对于每个纵坐标，只存在一个路径在该路径上走了一步。这就相当于将 $\sum_i \lambda_i-\mu_i$ 个纵坐标分配给各个路径，其中第 i 个路径分配的坐标个数

为 $\lambda_i - i - \mu_{\sigma(i)} + \sigma(i)$。当然，这个值小于 0 的时候答案定义为 0。所以这个对应排列的方案数就是

$$(\sum_i \lambda_i - \mu_i)! \prod_i \frac{1}{(\lambda_i - i - \mu_{\sigma(i)} + \sigma(i))!}$$

直接用行列式表达即可得到原式。 □

特殊的，当 $\mu = (0, 0 \ldots, 0)$ 的时候我们可以得到对于标准杨表计数的行列式公式，可以看到的是这个行列式和钩子公式可以联系起来。

定理 7.6. *(Frobenius Determinantal Formula)*

标准杨表 λ 的个数为

$$f_\lambda = (\sum_i \lambda_i)! \left| \frac{1}{(\lambda_i + j - i)!} \right|_{i,j=1}^{|\lambda|}$$

例题 4. *(Euler numbers)*

长为 $2n$ 的排列中，计算满足 $a_1 < a_2 > a_3 < \cdots > a_{2n-1} < a_{2n}$ 的方案数。

分析 如果在一个 $(2n-1+2, 2n-2+2, \ldots, n+2)/(2n-1, 2n-2, \ldots, n)$ 的斜杨表里填上数，那么这样的斜杨表刚好可以对应一个满足上述条件的排列。具体方法就是把这个斜杨表从下到上从左到右依次写出。那么答案就是

$$(2n)! \det\left(\frac{1}{(2j-2i+2)!} \right)_{i,j=1}^{n}$$

对于这个问题，这个答案可能并不鼓舞人心；因为欧拉数已经可以做到 $O(n \log n)$ 求出。不过，这个思路还可以求一些匪夷所思的东西，这里限于篇幅不再说明。

为了继续介绍下面的定理，我们需要了解一些补充定义。

7.4 Hankel Determinants

现在有一个序列 $\{a_i\} = a_0, a_1, \ldots$，那么它的 Hankel 矩阵被定义为

$$\left(a_{i+j-2}\right)_{i,j=1}^{n} = \begin{pmatrix} a_0 & a_1 & \ldots & a_{n-1} \\ a_1 & a_2 & \ldots & a_n \\ \ldots & \ldots & \ldots & \ldots \\ a_{n-1} & a_n & \ldots & a_{2n-2} \end{pmatrix}$$

关于它的行列式，有非常多的研究。这里介绍一个广为人知的引理。

引理 7.1.

$$\det\left((X_i + A_{n-1}) \cdots (X_i + A_{j+1})(X_i + B_j) \cdots (X_i + B_1)\right)_{i,j=0}^{n-1} = \prod_{0 \le i < j \le n-1} (X_i - X_j) \prod_{1 \le i \le j \le n-1} (B_i - A_j)$$

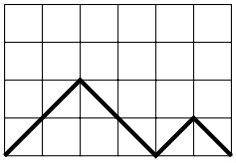
利用这个引理可以推得一个和卡特兰数 C_n 相关的定理 [17]。

定理 7.7.

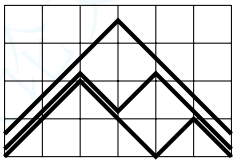
$$\det\left(C_{\alpha_i+j}\right)_{i,j=0}^{n-1} = \prod_{0\le i<j\le n-1}(\alpha_j-\alpha_i)\prod_{i=0}^{n-1}\frac{(i+n)!\,(2\alpha_i)!}{(2i)!\alpha_i!\,(\alpha_i+n)!}$$

7.5 k-Dyck Path

一个长为 $2n$ 的 Dyck Path 是从 $(0,0)$ 到 $(2n,0)$ 的一条路径，每次横坐标一定增加 1，纵坐标增加 1 或减少 1，但是一定不会越过 x=0。如下图所示。长为 $2n$ 的 Dyck Path 条数就是卡特兰数第 n 项。



定义一个“k-Dyck Path”为 k 条相同起点和终点的 Dyck Path 的集合。假设第 i 条依次经过的点为 $P_i = ((1,x_{(i,1)}),\ldots,(2n,x_{(i,2n)}))$（起点不算），那么第 i 条路径不能越过第 $i-1$ 条路径，即有 $\forall i>1, j\in[1,2n], x_{(i,j)} \ge x_{(i-1,j)}$。

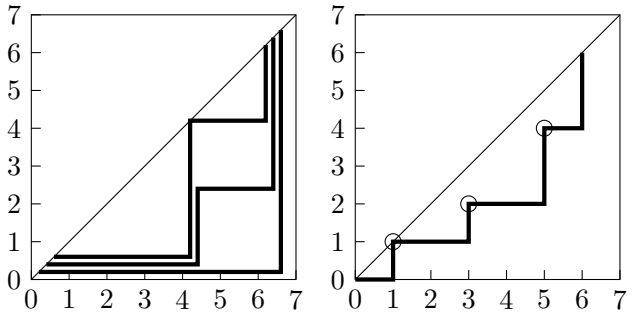


我们要提到的定理如下

定理 7.8. **列数**不超过 $2k$ 的，**元素**都在 $[1,n]$ 内的且**每行大小为偶数**的半标准杨表和长度均为 $2n+2$ 的 *k-Dyck Path* 形成双射关系，且计数公式如下

$$b_{n,k} = \prod_{1\le i\le j\le n}\frac{2k+i+j}{i+j}$$

考虑矩阵和 k-Dyck Path 的关系。如果把 k-Dyck Path 旋转一下，就成为了从 $(0,0)$ 到 $(n+1,n+1)$ 的 k 条路径，且每条路径只能往右和上走，且不能越过前一条路径（第一条路径不能越过 $x=y$）。



如果对于每个路径记下其每次转向的“拐角”，即这条路径每次从上方向转为右方向的地方，如上图所示。我们发现其实每条 Dyck Path 就是一个严格上升序列。

推论 7.1. *设矩阵 M 满足当 $j > i$ 时 $M_{ij} = 0$（这样的矩阵也叫做下三角矩阵）。定义 M 的 LDS 为权值最大的一条路径 $(x_1, y_1)(x_2, y_2)\dots(x_m, y_m)(x_i > x_{i+1}$ 或 $x_i = x_{i+1}$ 且 $y_i < y_{i+1})$，权值的定义是 $\sum_{i=1}^{m} M_{x_i y_i}$。满足 LDS(M)≤ k 的下三角矩阵 M 和 k-Dyck Path 一一对应。*

由参考文献 [1] 可得如下定理

定理 7.9. 元素都在 $[1, n]$ 内的且**每行大小为偶数**的半标准杨表和一个 $n \times n$ 的对称矩阵 M' 有着双射关系。且杨表的列数和 $LDS(M')$ 相等。

易证满足 LDS(M')≤ $2k$ 的对称矩阵 M' 和满足 LDS(M)≤ k 的下三角矩阵 M 一一对应。这样我们就成功证明了双射关系。如果我们把 k-Dyck Path 稍微平移一下，使得第 i 条 Dyck Path 的初始点为 $(-2i + 2, 0)$，终点为 $(2n + 2i, 0)$，那么我们要计算的就是这样的 k 条不相交路径数量。使用公式可以得到答案为

$$b_{n,k} = \det(C_{n+i+j-1})_{i,j=1}^{k} = \prod_{1 \le i \le j \le n} \frac{2k + i + j}{i + j}$$

其中 C_m 是卡特兰数第 m 项。套用上节的公式即可得到结果。

7.6 Bender-Knuth Conjecture

我们考虑一个元素都在 $[1, n]$ 内，列数不超过 k 的半标准杨表。它的个数被证明和 $n \times n$ 的元素在 $[0, k]$ 内的对称矩阵满足每行每列都非严格递增的数量相同 [11]。下列公式是 Bender 和 Knuth 在研究平面图划分时所猜想的公式，后来被证明。由于这个公式没有组合证明，所以不再赘述，有兴趣者可以查阅文献。

$$a_{n,k} = \prod_{1 \le i \le j \le n} \frac{k + i + j - 1}{i + j - 1}$$

8 半标准杨表的计数公式

半标准杨表 (semistandard young tableaux) 是列严格递增，行非严格递增的杨表。对于半标准杨表的计数有如下的公式。

定理 8.1. *若表内数为 $[1,n]$ 中的整数，那么半标准杨表 $\lambda=(\lambda_1,\ldots,\lambda_m)$ 的个数为*

$$\prod_{(i,j)\in\lambda}\frac{n+j-i}{h_\lambda(i,j)}$$

也可以只用 λ_i 表示

$$\prod_{1\le i<j\le n}\frac{\lambda_i-\lambda_j+j-i}{j-i}$$

其中定义 $i>|\lambda|$ 时 $\lambda_i=0$。

为了证明这个公式，我们需要一些补充定义。

8.1 对称多项式和交错多项式

在数学里，对称多项式 (Symmetric Polynomial) 是指对于所有的 n 个参数，无论怎样排列，这个多项式都是不变的。比如 $f(x_1,x_2,x_3)$ 若是一个对称多项式，那么 $f(x_1,x_2,x_3)=f(x_1,x_3,x_2)=f(x_2,x_1,x_3)=f(x_2,x_3,x_1)=f(x_3,x_1,x_2)=f(x_3,x_2,x_1)$ 对于所有 x_1,x_2,x_3 成立。

对于一个多项式 $f(x_1,\ldots,x_n)$，如果满足对于排列 σ，$f(x_{\sigma(1)}\ldots x_{\sigma(n)})=\mathrm{sgn}(\sigma)f(x_1,\ldots.x_n)$，那么这个多项式被称作为交错多项式 (Alternating Polynomial)。两个交错多项式相乘可以得到对称多项式，而一个交错多项式和一个对称多项式相乘可以得到一个交错多项式。

范德蒙德多项式定义为 $v_n(x_1,\ldots,x_n)=\prod_{1\le i<j\le n}(x_j-x_i)$ 是范德蒙德矩阵的行列式，是一个最基本的交错多项式。

引理 8.1. *所有的交错多项式都可以被表示为 $A\cdot v_n$ 的形式。这也就是说，所有的形如 $f(x_1,\ldots,x_n)$ 的交错多项式都有一个因式为 $v_n(x_1,\ldots,x_n)$。*

证明. 对于交错多项式 $f(x_1,\ldots,x_n)$，如果把 $x_i=x_j$ 带进去，可以得到 $f(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n)=f(x_1,\ldots,x_j,\ldots,x_i,\ldots,x_n)=-f(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n)$，即 $f(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n)=0$，所以 (x_i-x_j) 是该多项式的一个因式。所以 $v_n(x_1,\ldots,x_n)$ 一定是一个因式；又由交错多项式和对称多项式得交错多项式，证毕。 □

8.2 表内数 $\leq n$ 的半标准杨表计数

为了说明定理的正确性，我们需要介绍一个东西与半标准杨表建立双射。

定义 8.1. *Gelfand–Tsetlin patterns* 是一个包含**正整数**的数字三角形，第 i 行有 i 个数。表示为 $G(g_{ij})_{1\leq j\leq i\leq n}$。且这些数满足 $g_{(i+1)j}\leq g_{ij}<g_{(i+1)(j+1)}$。如下所示。

$$\begin{array}{ccccccccccc} & & & 3 & & & & \lambda^{(1)}=(2) & & & \\ & & 3 & & 4 & & & \lambda^{(2)}=(2,2) & & 1\ \ 1\ \ 3 & \\ & 1 & & 4 & & 6 & \rightarrow & \lambda^{(3)}=(3,2,0) & \rightarrow & 2\ \ 2\ \ 4 & \\ 1 & & 2 & & 6 & & 7 & \lambda^{(4)}=(3,3,0,0) & & 5\ \ 5 & \\ 1 & & 2 & & 5 & & 7\ \ 8 & \lambda^{(5)}=(3,3,2,0,0) & & & \end{array}$$

引理 8.2. 定义 $x_j=\lambda_{n+1-j}+j(1\leq j\leq n)$，规定大于 $|\lambda|$ 的都为 0。

那么，可以通过如下的算法将一个最后一行为 $(x_1,\dots x_n)$ 的 *Gelfand–Tsetlin patterns* 和一个形状为 λ，数在 $[1,n]$ 内的半标准杨表建立双射。

1. 对于第 i 行，$\lambda^{(i)}=(g_{ii}-i,\dots,g_{i2}-2,g_{i1}-1)$，除掉 $\lambda^{(i)}$ 末尾的 0。可以发现 $\lambda^{(i)}$“包含”$\lambda^{(i-1)}$，即 $\lambda^{(i-1)}$ 每一项都不会超过 $\lambda^{(i)}$

2. 从 (1) 开始，每次把数 i 放在 $\lambda^{(i)}$ 比 $\lambda^{(i-1)}$ 多的地方。

有了这个引理后，对杨表的计数就变成了对这个数字三角形的计数。

定理 8.2. 对于 $n\geq 1$ 且 $1\leq x_1<\cdots<x_n$，最后一行为 $(x_1\dots.x_n)$ 的 *Gelfand–Tsetlin patterns* 数量为

$$\prod_{1\leq i<j\leq n}\frac{x_j-x_i}{j-i}$$

证明. 考虑使用归纳证明。定义 $f_n(x_1,\dots x_n)=\prod_{1\leq i<j\leq n}\frac{x_j-x_i}{j-i}$。显然 $n=1$ 成立。定义

$$\begin{aligned} \mathrm{S}_{i=l}^{r}f(i) &= \sum_{i=l}^{r-1}f(i)(l<r) \\ \text{自然的，就有}\mathrm{S}_{i=l}^{r}f(i) &= -\mathrm{S}_{i=r}^{l}f(i),\mathrm{S}_{i=l}^{l}f(i)=0 \\ \text{那么}\mathrm{S}_{i=a}^{b}f(i)+\mathrm{S}_{i=b}^{c}f(i) &= \mathrm{S}_{i=a}^{c}f(i) \\ W_n(x_1,\dots x_n) &= \mathrm{S}_{y_1=x_1}^{x_2}\dots\mathrm{S}_{y_{n-1}=x_{n-1}}^{x_n}f_{n-1}(y_1,\dots y_{n-1}) \end{aligned}$$

那么我们要证明的就是 W_n 等同于 f_n。首先我们注意到，在上述定义下有 $W_n(x_1,\dots,x_i,x_{i+1},\dots n)=-W_n(x_1,\dots,x_{i+1},x_i,\dots n)$。

$$\begin{aligned} -W_n(x_1,\dots,x_{i+1},x_i,\dots n) &= -\mathrm{S}_{other\ y}\dots\mathrm{S}_{y_{i-1}=x_{i-1}}^{x_{i+1}}\mathrm{S}_{y_i=x_{i+1}}^{x_i}\mathrm{S}_{y_{i+1}=x_i}^{x_{i+2}}f_{n-1}(y_1,\dots y_{n-1}) \\ &= \mathrm{S}_{other\ y}\dots\mathrm{S}_{y_{i-1}=x_{i-1}}^{x_{i+1}}\mathrm{S}_{y_i=x_i}^{x_{i+1}}\mathrm{S}_{y_{i+1}=x_i}^{x_{i+2}}f_{n-1}(y_1,\dots y_{n-1}) \\ &= \mathrm{S}_{other\ y}\dots\mathrm{S}_{y_i=x_i}^{x_{i+1}}(\mathrm{S}_{y_{i-1}=x_{i-1}}^{x_i}+\mathrm{S}_{y_{i-1}=x_i}^{x_{i+1}})(\mathrm{S}_{y_{i-1}=x_i}^{x_{i+1}}+\mathrm{S}_{y_{i-1}=x_{i+1}}^{x_{i+2}})f_{n-1}(y_1,\dots y_{n-1}) \end{aligned}$$

由于 f 是个交错多项式，把上式展开后便可以发现有 3 项都是 0，易得结论。所以我们证明了 W_n 是交错多项式，即 W_n 含有 $\prod_{1\le i<j\le n}(x_j-x_i)$ 因式。

我们又可以发现，f_{n-1} 是一个每项的总次数都是 $M=\frac{(n-1)(n-2)}{2}$ 的多项式，即 $f_{n-1}(y_1,\dots y_{n-1})$ 的每项都形如 $c\prod_{j=1}^{n-1}y_j^{a_j}(\sum a_j=M)$。且有 $\mathrm{S}_{y_1=x_1}^{x_2}f_{n-1}(y_1,\dots y_{n-1})$ 是一个关于 $x_1,x_2,y_2,\dots,y_{n-1}$ 的 $M+1$ 次多项式，且最高次项是形如 $-\frac{c}{a_1+1}x_1^{a_1+1}\prod_{j=2}^{n-1}y_j^{a_j}$ 和 $\frac{c}{a_1+1}x_2^{a_1+1}\prod_{j=2}^{n-1}y_j^{a_j}$。

由于多一个 S 就多一个次数，所以我们只需要关心最高项的次数。这样可以得到 W_n 的最高次是 $\frac{n(n-1)}{2}$。又由于 W_n 含有 $\prod_{1\le i<j\le n}(x_j-x_i)$ 因式，这个因式的最高次刚好就是 $\frac{n(n-1)}{2}$。这就说明所有低于这个次数的项一定会被消去，即

$$W_n(x_1\dots x_n)=c\prod_{1\le i<j\le n}(x_j-x_i)=\prod_{j=1}^{n-2}\frac{1}{j!}\mathrm{S}_{y_1=x_1}^{x_2}\dots\mathrm{S}_{y_{n-1}=x_{n-1}}^{x_n}\prod_{1\le i<j\le n-1}(y_j-y_i)$$

注意到 $\prod_{1\le i<j\le n-1}(y_j-y_i)$ 里一个形如 $\prod_{i=1}^{n-2}y_i^{a_i}$ 的多项式，它的次数一定是 $0,1,\dots,n-2$ 的一个排列。而它产生的最高次项系数为 $\prod_{i=1}^{n-2}\frac{1}{i+1}=\frac{1}{(n-1)!}$。可以得到 $c=\frac{1}{(n-1)!}\prod_{j=1}^{n-2}\frac{1}{j!}=\prod_{j=1}^{n-1}\frac{1}{j!}$。易得原式，证毕。 □

所以我们可以得到原先的公式。

$$\prod_{i\le i<j\le n}\frac{\lambda_{n+1-j}+j-\lambda_{n+1-i}-i}{j-i}=\prod_{i\le i<j\le n}\frac{\lambda_j-j-\lambda_i+i}{j-i}=\prod_{(i,j)\in\lambda}\frac{n+j-i}{h_\lambda(i,j)}$$

例题 5. *(CodeChef BillBoards(BB))*

有一排长为 n 的格子，大厨需要在里面放置一些物品，使得任意连续 m 个格子中一定有 K 个物品。大厨想要放的物品数**尽量少**，问有多少种方法，答案对 10^9+7 取模。$(K\le m\le 50, m\le n\le 10^9)$

分析如果 $m|n$，那么最小值就是 $\frac{n}{m}$。将这些格子分成 $\frac{n}{m}$ 段，每段长为 m。记下每段中的 1 的位置，如下图 (n=25,m=5,K=3)

00111 01011 01101 10110 11100 →

3	2	2	1	1
4	4	3	3	2
5	5	5	4	3

可以发现在这个表中，每列从上往下是递增的，每行从左往右则必须是非增的，否则会不满足题目要求；反之则一定满足题目要求。把这个表转一下就变成了半标准杨表，只需要对半标准杨表计数即可。复杂度为 $O(m^2)$ 或是 $O(m\log(10^9+7))$。

剩下的问题就是当 $m\nmid n$ 时怎么办。令 $p=n \mod m$。如果 $p\le m-k$，那么每块的前 p 个必须是 0；如果 $p\ge m-k$，那么每块的后 $m-p$ 个一定是 1（注意这个时候要多加 1 块）。这样就把这个问题转化了。

9 行数限制的杨表计数

9.1 LIS 小于等于 K 的排列计数

定义 9.1.

$$u_k(x) = \sum_{\lambda\vdash n,\lambda_1\le k} f_\lambda^2 \tag{45}$$

$$U_k(x) = \sum_{n\ge 0} u_k(n)\frac{x^{2n}}{n!^2} \tag{46}$$

$$I_i(2x) = \sum_{n\ge 0}\frac{x^{2n+i}}{n!(n+i)!} \tag{47}$$

其中 I_i 被称作第一类修正贝塞尔函数 *(hyperbolic Bessel function of the first kind)*，这里有 $I_k(2x)=I_{-k}(2x)$（这个函数有很多性质）。

定理 9.1. *(The Gessel-Bessel identity)*[7]

$$U_k(x)=\det(I_{i-j}(2x))_{i,j=1}^{K}$$

当 $k=2$ 的时候就有 (注意 $I_1=I_{-1}$)

$$\begin{aligned} U_2(x) &= \begin{vmatrix} I_0(2x) & I_1(2x) \\ I_1(2x) & I_0(2x)\end{vmatrix} = I_0(2x)^2-I_1(2x)^2 \\ \Rightarrow u_2(n) &= \frac{1}{n+1}\binom{2n}{n}\end{aligned}$$

而当 $k=3$ 的时候，有

$$u_3(n)=\frac{1}{(n+1)^2(n+2)}\sum_{j=0}^{n}\binom{2j}{j}\binom{n+1}{j+1}\binom{n+2}{j+2}$$

可惜的是，$k>3$ 的时候并没有如此“优美”的形式。不过，可以证明 $u_k(n)$ 是 $P-cursive$ 的。即一定存在某个多项式 $p_1,..,p_m$ 满足 $\sum_{i=1}^{m}u_k(n+i)p_i(n)=0$。下面列出了 4,5 的例子。

$$(n+4)(n+3)^2u_4(n)=(20n^3+62n^2+22n-24)u_4(n-1)-64n(n-1)^2u_4(n-2)$$

$$\begin{aligned}(n+6)^2(n+4)^2u_5(n)=&(375-400n-843n^2-322n^3-35n^4)u_5(n-1)\\&+(259n^2+622n+45)(n-1)^2u_5(n-2)\\&-225(n-1)^2(n-2)^2u_5(n-3)\end{aligned}$$

例题 6. 求有多少长度为 $N(3\le N\le 500)$ 的排列可以表示为 3 个上升子序列。答案对 10^9+7 取模。

分析 普通的做法是使用 $dp[i][j][k]$ 表示表示现在有 i 个数, 形成了三个上升子序列, 其中最大的子序列末尾显然是第 i 大的数, 第二大的子序列末尾是第 j 大的数, 第三大的子序列末尾是第 k 大的数这样，这个 DP 优化之后可以做到 $O(N^3)$。稍加思考后可以发现这就是 LIS ≤ 3 的排列个数问题。如果使用杨表的性质则可以做到 $O(N^2)$，即枚举三行的形态。如果使用上述方法，则不仅能做到 $O(n)$，甚至还能做到 $O(\sqrt{n})$ 的优秀复杂度。

9.2 行数小于等于 K 的杨表

定义 9.2.

$$\begin{aligned} y_k(n) &= \sum_{\lambda \vdash n, \lambda_1 \leq k} f_\lambda \\ v_{2k}(n) &= \sum_{\lambda \vdash n, \lambda_1 \leq k} f_{2\lambda'} \\ z_k(n) &= \sum_{\lambda \vdash n, \lambda'_1 \leq k} f_{2\lambda'} \end{aligned}$$

这里定义 λ' 满足 $\lambda'_i = |\{j|\lambda_j \geq i\}|$（$\lambda'$ 也可以理解成将杨表行列翻转之后的 λ），$2\lambda' = (2\lambda'_1, 2\lambda'_2, \dots)$。

在 k 比较小的时候，有如下比较简洁的公式。

定理 9.2. [8]

$$\begin{aligned} y_2(n) &= \binom{n}{\lfloor n/2 \rfloor} \\ y_3(n) &= \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{2i} C_i \\ y_4(n) &= C_{\lfloor (n+1)/2 \rfloor} C_{\lceil (n+1)/2 \rceil} \\ y_5(n) &= 6 \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{2i} C_i \frac{(2i+2)!}{(i+2)!(i+3)!} \end{aligned}$$

这里 C_m 指卡特兰数第 m 项。

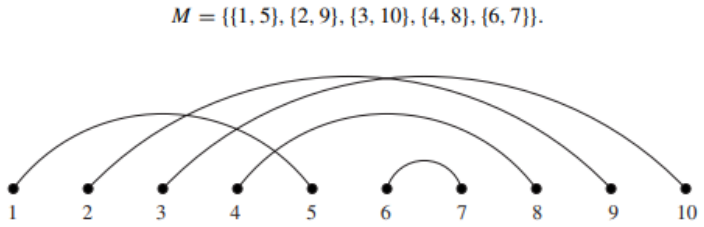
定义 9.3. 定义它们的指数生成函数如下。

$$\begin{aligned} Y_k(n) &= \sum_{n \geq 0} y_k(n) \frac{x^n}{n!} \\ V_{2k}(n) &= \sum_{n \geq 0} v_{2k}(n) \frac{x^n}{n!} \\ Z_k(n) &= \sum_{n \geq 0} z_k(n) \frac{x^n}{n!} \end{aligned}$$

定理 9.3.

$$\begin{aligned}
Y_{2k}(x) &= \det(I_{i-j}+I_{i+j-1})_{i,j=1}^{k} \\
Y_{2k+1}(x) &= e^{x}\det(I_{i-j}-I_{i+j})_{i,j=1}^{k} \\
V_{2k}(x) &= \det(I_{i-j}-I_{i+j})_{i,j=1}^{k} \\
Z_{2k}(x) &= \frac{1}{4}\det(I_{i-j}+I_{i+j-2})_{i,j=1}^{k}+\frac{1}{2}\det(I_{i-j}-I_{i+j})_{i,j=1}^{k-1} \\
Z_{2k+1}(x) &= \frac{1}{2}e^{x}\det(I_{i-j}-I_{i+j-1})_{i,j=1}^{k}+\frac{1}{2}e^{-x}\det(I_{i-j}+I_{i+j-1})_{i,j=1}^{k}
\end{aligned}$$

例题 7. 一排 n 个点，每个点最多只能和一个其他点进行匹配。对于匹配 $(a_1,b_1)(a_2,b_2)\dots(a_k,b_k)$，如果存在 $a_1<a_2<\cdots<a_k<b_1<\cdots<b_k$，那么这些匹配被称作为 *k-crossing*。对于匹配 $(a_1,b_1)(a_2,b_2)\dots(a_k,b_k)$，如果存在 $a_1<a_2<\cdots<a_k<b_k<b_{k-1}<\cdots<b_1$，那么这些匹配被称作为 *k-nesting*。比如 *(1,5)(2,8)(3,6)(4,7)* 存在一个 *3-crossing(1,5)(3,6)(4,7)* 和一个 *2-nesting(2,8)(3,6)*。一个排列被称作 *k-noncrossing* 当且仅当这个排列中不存在 *k-crossing*。同理定义*-nonnesting*。现在分别询问对于一个长为 n 的排列，*k-noncrossing* 的排列个数和 *k-nonnesting* 的排列个数。



分析 先解决 k-nonnesting（可以证明 k-nonnesting 答案和 k-noncrossing 相同，实际上若令 i-nonnesting,j-noncrossing 的长度为 n 的排列个数为 $f_n(i,j)$，那么 $f_n(i,j)=f_n(j,i)$）考虑对于匹配 $(a_1,b_1)\dots(a_m,b_m)$，对于每对都交换。这正是对合排列，我们在之前已经知道每个对合排列都对应一个杨表。易得一个 k-nonnesting 排列，其 LDS 一定小于等于 $2k-1$；而一个非 k-nonnesting 排列，其 LDS 一定大于等于 $2k$。这说明问题的答案就是“行数不超过 2k-1”的杨表个数。

10 总结

由于作者无法读到系统介绍杨表的资料，所以这篇文章可能略显得有点杂乱。实际上这篇文章介绍的内容还相当浅（如果有兴趣翻看参考文献的话会发现有的长达几百页）。许多内容由于需要大量前置知识或说明过程非常冗长所以略去不提。希望这篇文章能够让同

学们真正地认识杨表，能给同学们打开一扇窗户，能让同学们看到在信息学竞赛之外还有更广阔的天地。

11 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢成都七中蔺洋老师和张君亮老师的关心和指导。
感谢国家集训队教练张瑞喆的指导。
感谢神树大人的指引，以及杨乾澜学长和郑钧天同学的验稿。

参考文献

[1] Burge, William H. "Four correspondences between graphs and generalized Young tableaux." Journal of Combinatorial Theory, Series A 17.1 (1974): 12-30.

[2] Bandlow, Jason. "An elementary proof of the hook formula." the electronic journal of combinatorics 15.1 (2008): 45.

[3] Wikipedia, Lindstrom–Gessel–Viennot lemma

[4] Knuth, Donald Ervin. The art of computer programming: sorting and searching. Vol. 3. Pearson Education, 1997.

[5] Ciocan-Fontanine I, Konvalinka M, Pak I. The weighted hook length formula[J]. Journal of Combinatorial Theory, Series A, 2011, 118(6): 1703-1717.

[6] Wikipedia, Involution (mathematics)

[7] Gessel, Ira M. "Symmetric functions and P-recursiveness." J. Comb. Theory, Ser. A 53.2 (1990): 257-285.

[8] Stanley, Richard P. "Increasing and decreasing subsequences and their variants." International Congress of Mathematicians. Vol. 1. 2007.

[9] Romik, Dan. The surprising mathematics of longest increasing subsequences. Vol. 4. Cambridge University Press, 2015.

[10] Schensted, Craige. "Longest increasing and decreasing subsequences." Canadian Journal of Mathematics 13 (1961): 179-191.

[11] Andrews G. Plane partitions. II. The equivalence of the Bender-Knuth and MacMahon conjectures[J]. Pacific Journal of Mathematics, 1977, 72(2): 283-291.

[12] N. J. A. Sloane,OEIS, The Online Encyclopedia of Integer Sequences.

[13] Konvalinka, Matjaz. "A bijective proof of the hook-length formula for skew shapes." Electronic Notes in Discrete Mathematics 61 (2017): 765-771.

[14] Greene, Curtis. "An extension of Schensted's theorem." Young Tableaux in Combinatorics, Invariant Theory, and Algebra. Academic Press, 1982. 39-50.

[15] Krattenthaler, Christian. The major counting of nonintersecting lattice paths and generating functions for tableaux. Vol. 552. American Mathematical Soc., 1995.

[16] Narayanan, Hariharan. "On the complexity of computing Kostka numbers and Littlewood-Richardson coefficients." Journal of Algebraic Combinatorics 24.3 (2006): 347-354.

[17] Krattenthaler C. Determinants of (generalised) Catalan numbers[J]. Journal of Statistical Planning and Inference, 2010, 140(8): 2260-2270.

[18] 金策的 CTSC2017 讲题课件