

**2021 年信息学奥林匹克
中国国家集训队论文**

目 录

1	陈雨昕,《太阳神的宴会》命题报告	4
2	代晨昕,后缀树的构建	24
3	邓明扬,一类调整算法在信息学竞赛中的应用	34
4	丁晓漫,再探线性规划对偶在信息学竞赛中的应用	42
5	郭城志,浅谈信息学竞赛中的弦图问题	54
6	胡昊,浅谈 Lyndon 分解	64
7	蒋凌宇,信息学竞赛中构造题的常用解题方法	73
8	李白天,信息学竞赛中的生成函数计算理论框架	82
9	林昊翰,《逛公园》命题报告	109
10	林立,浅析一些维护二维点的算法	124
11	马耀华,浅谈超现实数与不平等博弈	134
12	潘佳奇,浅谈线性代数与图论的关系	159
13	彭思进,浅谈棋盘模型在计数问题中的应用	177
14	邱天异,对信息学比赛中选手分数数据的分析	212
15	钱易,浅谈亚 \log 数据结构在 OI 中的应用	239
16	施良致,对信息学竞赛中二维平面处理问题的总结和再优化	252
17	魏衍芑,《遇到困难睡大觉》命题报告	264
18	徐哲安,浅谈有限状态自动机及其应用	270
19	虞皓翔,再谈图连通性相关算法	293
20	叶卓睿,浅谈一类基于概率的约瑟夫问题	328
21	左骏驰,浅谈多项式牛顿迭代与拉格朗日反演在 OI 中的应用	338

22	张隽恺, 浅谈一类最小公倍数的求和问题及其拓展	352
23	周欣, 浅谈一类树分块的构建算法及其应用	362
24	周镇东, 浅谈一类树上路径相关问题	376

《太阳神的宴会》命题报告

福建省福州第一中学 陈雨昕

摘要

《太阳神的宴会》¹是我为 IOI2021 候选队员自主互测活动²命制的一道试题，本题要求选手探究一种等价关系下的字符串匹配。以往基于后缀数组的算法具有运行时间略高、不够直观的局限性。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，其中满分算法通过引入辅助序列描述等价类，并将转移不同的等价类分开，将后缀自动机的应用范围扩展到了等价关系上，作者称它为子形状自动机；将本题所要求的相似关系和拼接过程，对应到子形状自动机上，转化为自动机所对应的有向无环图的带权路径计数；在此基础上用合并相同转移的手段进一步优化，从而解决了本题。进一步地，本文指出，针对一类字符串上的等价关系，子形状序列自动机可用于处理字符串的本质不等价子串相关问题。

记号的约定

本文讨论的字符串与序列均为有限长的。

字符集为 Σ 的全体字符串记作 Σ^* ，空串记为 ϵ 。对于字符串 s ，记其长度为 $|s|$ ，从第一个字符到最后一个字符依次为 $s_1, s_2, \dots, s_{|s|}$ ，记作 $s = s_1 s_2 \cdots s_{|s|}$ 。字符串 s 和 t 的连接记作 st 。如果 $x, y, z \in \Sigma^*$ 使得 $s = xyz$ ，则称 x 为 s 的前缀， y 为 s 的子串， z 为 s 的后缀。 s 的子串 $s[l, r]$ 是指字符串 $s_l s_{l+1} \cdots s_r$ ($1 \leq l \leq r \leq |s|$)，特别地，对于不在上述范围内的 l, r ， $s[l, r] = \epsilon$ 。两个字符串 s, t 的最长公共前缀长度简称 LCP，记为 $\text{LCP}(s, t)$ 。字符串集合 S 的全体元素的最长公共前缀记为 $\text{LCP } S$ 。

空序列也记作 ϵ 。对于序列 a ，记其长度为 $|a|$ ，第一项到最后一项依次为 $a_1, a_2, \dots, a_{|a|}$ ，记作 $a = (a_1, a_2, \dots, a_{|a|})$ 。序列 a 和 b 的连接记作 $a \cdot b$ ，如无歧义也可记作 ab 。如果序列 x, y, z 使得 $a = xyz$ ，则称 x 为 a 的前缀， y 为 a 的连续子序列， z 为 a 的后缀。 a 的连续子序列 $a[l, r]$ 是指序列 $(a_l, a_{l+1}, \dots, a_r)$ ($1 \leq l \leq r \leq |a|$)，特别地，对于不在上述范围内的 l, r ， $a[l, r] = \epsilon$ 。两个序列 a, b 的 LCP 记为 $\text{LCP}(a, b)$ 。序列集合 A 的全体元素的最长公共前缀记为 $\text{LCP } A$ 。

¹<https://uoj.ac/problem/595>

²<https://uoj.ac/contest/58>

一个部分确定有限状态自动机简称部分 DFA，用五元组 $(Q, \Sigma, \delta, \bar{q}, F)$ 表示。其中， Q 是状态集合， Σ 是字符集， δ 是转移函数， \bar{q} 是初始状态， F 是终态集合。在部分 DFA 中，状态 q 的 α 转移 $\delta(q, \alpha)$ 可以存在，也可以不存在，如果不存在，记 $\delta(q, \alpha) = 0$ 。

对于命题 P ，记艾弗森括号 $[P] = 1$ 当且仅当 P 为真， $[P] = 0$ 当且仅当 P 为假。

1 试题

1.1 题目描述

设字符集 Σ 为前 k 个小写英文字母的集合，题中所有字符串均由 Σ 中的字符构成。

对于 Σ 上的置换 f 和 $s \in \Sigma^*$ ，令 s 作用 f 后的结果为 $f(s) = f(s_1)f(s_2)\cdots f(s_{|s|})$ 。

我们称字符串 s 与字符串 t 相似，当且仅当存在一个 Σ 上的置换 f ，使得 $f(s) = t$ ，记作 $s \sim t$ 。例如，字符串 aab 与 bba 相似，与 abb 不相似。

现给出 n 个字符串 S_1, S_2, \dots, S_n 。记 C_i 为全体与 S_i 的至少一个子串相似的字符串的集合。

称字符串 T 是好的，当且仅当存在字符串 T_1, T_2, \dots, T_n ，使得 $T_i \in C_i$ ，且 $T = T_1T_2\cdots T_n$ 。求有多少个好的字符串。

1.2 输入格式

第一行两个正整数 n, k 。

接下来 n 行，每行一个字符串，依次表示 S_1, S_2, \dots, S_n 。

1.3 输出格式

输出好的字符串的数量，对 998244353 取模。

1.4 样例 1

1.4.1 样例输入

2 2

aa

a

1.4.2 样例输出

11

1.4.3 样例解释

S_1 的子串有 ϵ, a, aa ，与它们之一相似的串有 ϵ, a, b, aa, bb 。

S_2 的子串有 ϵ, a ，与它相似的串有 ϵ, a, b 。

连接起来，好的串有： $\epsilon, a, b, aa, ab, ba, bb, aaa, aab, bba, bbb$ 。

1.5 数据规模与约定

简记 $L = \sum_{i=1}^n |S_i|$ 。

保证 $2 \leq k \leq 26$ ， S_i 非空， $L \leq 10^6$ 。

子任务一（2分）： $n = 1, k = 2, L \leq 1000$ 。

子任务二（7分）： $n = 1, L \leq 1000$ 。

子任务三（11分）： $n = 1, L \leq 10^5$ 。

子任务四（13分）： $k = 2, L \leq 1000$ 。

子任务五（17分）： $L \leq 1000$ 。

子任务六（31分）： $k \leq 5$ 。

子任务七（19分）：无特殊限制。

1.6 时空限制

时间限制：3s

空间限制：2GB

2 做题情况

本次互测活动共 3 题，本题为其中第一道。

活动面向全体 Universal OJ 用户，共 122 位选手参与了本次互测活动。

共 28 位选手在本题上得分：2 分 6 位、9 分 13 位、15 分 1 位、22 分 1 位、39 分 6 位，最高分为 50 分，1 位。

3 初步分析

首先我们说明：

性质 3.1. \sim 是一个等价关系。

证明. 自反性: 对于字符串 s , 取置换 $f = \text{id}_\Sigma$, 得 $s \sim s$ 。

对称性: 设 $s \sim t$ 。则 $|s| = |t|$, 且存在置换 f 满足 $\forall 1 \leq i \leq |s|, f(s_i) = t_i$ 。由于 f 是置换, 可取其逆置换 f^{-1} , 则 $f^{-1}(t_i) = s_i$ 。所以 $t \sim s$ 。

传递性: 设 $r \sim s, s \sim t$ 。则 $|r| = |s| = |t|$, 且存在置换 f, g 满足 $\forall 1 \leq i \leq |r|, f(r_i) = s_i, g(s_i) = t_i$ 。那么取置换的复合 $h = g \circ f$, 得 h 也是置换, 且 $\forall 1 \leq i \leq |r|, h(r_i) = t_i$ 。所以 $r \sim t$ 。□

其次, 题中字符串的相似关系具有以下性质:

性质 3.2. 若 $s \sim t$, 则 $\forall 1 \leq l \leq r \leq |s|, s[l, r] \sim t[l, r]$ 。

证明. 因为 $s \sim t$, 所以存在置换 $f, \forall 1 \leq i \leq |s| = |t|, f(s_i) = t_i$ 。 $|s[l, r]| = |t[l, r]| = r - l + 1$, 且 $\forall l \leq i \leq r, f(s_i) = t_i$ 。故 $s[l, r] \sim t[l, r]$ 。□

4 单字符串情形下的经典算法

对于只有一个字符串的情形, 只需求出 $|C_1|$ 。

4.1 算法一: $n = 1, k = 2, L \leq 1000$

在 $k = 2$ 的情形下, Σ 上只有两种置换, S_1 只有 $O(L^2)$ 种子串。因此 C_1 中所有可能出现的字符串可以直接枚举。考虑这些字符串的去重问题, 容易想到用 Trie 来解决。

因此得出一个简单的算法: 枚举 Σ 上的置换 f , 将 S_1 的所有后缀作用 f 后插入 Trie。最终 Trie 的结点数即为所求。时空复杂度 $O(L^2)$, 期望得分 2 分。

4.2 形状序列

当 k 更大时, 枚举置换将带来巨大的时空复杂度。因此, 我们要考虑一种更高效的方法来描述相似关系的等价类。

为了判定相似性, 我们试图将字符串对应到一种辅助序列上, 使得它记录字符间的相等关系, 而丢弃字符的具体取值。

4.2.1 形状序列的引入

定义 4.1. 对于字符串 s , 定义其形状序列为一个长度为 $|s|$ 的非负整数序列 a :

对于 $1 \leq i \leq |s|$,

- 若字符 s_i 在 s 中首次出现, 令 $a_i = 0$;
- 若字符 s_i 在 s 中并非首次出现, 那么在 $s[1, i-1]$ 中, 找到所有字符的最后一次出现位置, 从大到小记作 $j_1 > j_2 > \dots > j_z$ 。如果 $s_{j_x} = s_i$, 令 $a_i = x$ 。

由定义可得, 字符串的形状序列有以下性质:

性质 4.1. 对于字符串 s 和整数 $0 \leq r \leq |s|$, 如果 s 的形状序列为 a , 那么 $s[1, r]$ 的形状序列为 $a[1, r]$ 。

性质 4.2. 一个字符串的不同字符个数, 等于其形状序列中 0 的个数。

推论 4.1. 一个字符串的形状序列中 0 的个数不超过 k 。

以下记 $n_0(a)$ 表示非负整数序列 a 中 0 的个数。

性质 4.3. 设一个字符串的形状序列为 a , 则 $\forall 1 \leq i \leq |a|$, 有 $a_i \leq n_0(a[1, i-1])$ 。

接下来我们说明形状序列在相似关系方面的显著作用。

引理 4.1. 如果两字符串 s, t 相似, 那么 $\forall 1 \leq i < j \leq |s|$, 有 $s_i = s_j \iff t_i = t_j$ 。

证明. 由于 $s \sim t$, 可得 $|s| = |t|$, 且存在置换 f , 使得 $\forall 1 \leq i \leq |s|$ 有 $f(s_i) = t_i$ 。对于 $1 \leq i < j \leq |s|$, $s_i = s_j \implies t_i = f(s_i) = f(s_j) = t_j$ 。

由于 $t \sim s$, 同理可得 $t_i = t_j \implies s_i = s_j$ 。因此 $s_i = s_j \iff t_i = t_j$ 。 \square

定理 4.1. 两字符串 s, t 相似, 当且仅当 s 和 t 的形状序列相同。

证明. 先证必要性。对于 $1 \leq i \leq |s|$,

- 若 s_i 是首次出现的, 则由引理 4.1, 可得 t_i 也是首次出现的, 即 $a_i = b_i = 0$;
- 若 s_i 不是首次出现的, 那么在 $s[1, i-1]$ 中, 找到所有字符的最后一次出现位置, 从大到小记作 $j_1 > j_2 > \dots > j_z$ 。设 $s_i = s_{j_x}$, 由引理 4.1, 可得 $t_{j_x} = t_i$, 且 j_x 也是 $t[1, i-1]$ 中 t_{j_x} 的最后一次出现位置。所以 $a_i = b_i = x$ 。

再证充分性。设 s, t 具有相同的形状序列 a , 则 $|s| = |a| = |t|$ 。

设 a 所有为 0 的位置为 i_1, i_2, \dots, i_z 。根据形状序列的构造方式可知, s_{i_j} 互不相同, t_{i_j} 也互不相同。因此存在置换 f , 使得 $f(s_{i_j}) = t_{i_j}$ 。

以下利用数学归纳法证明: 对于任意的 $1 \leq i \leq |s|$, 上述 f 均满足 $f(s_i) = t_i$ 。

对于 $1 \leq i \leq |s|$, 假设对于任意的 $1 \leq j < i$, 均有 $f(s_j) = t_j$ 。

- 若 $a_i = 0$, $f(s_i) = t_i$ 。

- 若 $a_i = x > 0$, 那么在 $s[1, i-1]$ 中, 找到所有字符的最后一次出现位置, 从大到小记作 $j_1 > j_2 > \dots > j_z$ 。应用归纳假设, $f(s_i) = f(s_{j_x}) = t_{j_x} = t_i$ 。

综上所述, $\forall 1 \leq i \leq |s|$, $f(s_i) = t_i$ 。所以 $s \sim t$ 。

□

为了求一个字符串 s 的形状序列 a , 我们可以依次枚举 $i = 1, 2, \dots, |s|$, 同时维护 $last(\alpha)$ 表示字符 α 在 $s[1, i-1]$ 中最后一次出现的位置 (不存在设为 0)。若 $last(s_i) = 0$, 则 $a_i = 0$; 否则 $last(s_i)$ 是所有 $last(\alpha)$ ($\alpha \in \Sigma$) 中第 a_i 大的。因此, 一个字符串 s 的形状序列可以在 $O(k|s|)$ 的时间复杂度、 $O(k)$ 的空间复杂度内求出。

4.2.2 形状序列对应的字符串数

接下来我们讨论, 怎样的非负整数序列是一个字符串的形状序列:

定义 4.2. 对于非负整数序列 a , 称 a 是一个合法形状序列, 当且仅当 $n_0(a) \leq k$, 且 $\forall 1 \leq i \leq |a|$, 均有 $a_i \leq n_0(a[1, i-1])$ 。

由推论 4.1、性质 4.3 可得以下结论:

引理 4.2. 一个字符串的形状序列一定是合法形状序列。

接下来我们关注, 如何从形状序列还原回字符串。形状序列丢失了信息, 为了将这些信息补回, 我们引入以下概念:

定义 4.3. 对于字符串 s , 将其所含有的字符, 按照第一次出现位置从左到右的顺序写下来, 形成一个字符串 t , 称为 s 的提示串。

由定义可得以下结论:

引理 4.3. 字符串 s 的提示串长度等于其不同字符个数。

引理 4.4. 字符串 s 的提示串的字符两两不同。

现在我们可以根据形状序列和提示串还原一个字符串了。

引理 4.5. 设 a 是合法形状序列, 字符串 t 的字符两两不同, 且 $|t| = n_0(a)$ 。则存在唯一字符串 s 使得其形状序列为 a , 提示串为 t 。

证明. 以下设 $|t| = n_0(a) = z$ 。我们利用数学归纳法证明。

当 $|a| = 0$ 时, 可得 $|t| = 0$ 。有且只有 ϵ 满足条件, 引理显然成立。

对于正整数 N , 假设 $|a| = N-1$ 的情况下引理成立。当 $|a| = N$ 时:

首先可得字符串 s 的长度为 N 。根据性质 4.1, 可得 $s[1, N-1]$ 的形状序列为 $a[1, N-1]$ 。

- 若 $a_N = 0$, 则 s_N 在 s 中首次出现, 可得 $s_N = t_z$, 且 $s[1, N-1]$ 的提示串为 $t[1, z-1]$ 。对于 $a[1, N-1]$ 和 $t[1, z-1]$ 应用归纳假设, 可得存在唯一字符串 $s[1, N-1]$ 使得其形状序列为 $a[1, N-1]$, 提示串为 $t[1, z-1]$ 。因此引理成立。
- 若 $a_N = x \neq 0$, 那么 s_N 在 s 中非首次出现, $s[1, N-1]$ 的提示串为 t 。对于 $a[1, N-1]$ 和 t 应用归纳假设, 可得存在唯一字符串 $s[1, N-1]$ 使得其形状序列为 $a[1, N-1]$, 提示串为 t 。

根据性质 4.2, 在 $s[1, N-1]$ 中, 不同字符有 z 种。找到所有字符的最后一次出现位置, 从大到小记作 $j_1 > j_2 > \cdots > j_z$ 。因为 a 是合法形状序列, 可得 $x \leq z$ 。那么 $s_i = s_{j_x}$ 。因此引理成立。

综上所述, 引理成立。 □

最后我们得出以下定理:

定理 4.2. 对于合法形状序列 a , 恰有 $A_k^{n_0(a)}$ 个字符串的形状序列为 a 。其中 A_n^m 表示排列数 $\frac{n!}{(n-m)!}$ 。

证明. 满足 $n_0(a) = |t|$ 的合法形状序列 a 与字符两两不同的字符串 t 所构成的全体二元组 (a, t) 集合记作 P 。

根据性质 4.2、引理 4.2、引理 4.3、引理 4.4, 可构造映射 $f: \Sigma^* \rightarrow P$, 对于字符串 s , 设其形状序列为 a , 提示串为 t , 令 $f(s) = (a, t)$ 。根据引理 4.5, f 是可逆映射。

现在已知合法形状序列 a , 可得长度为 $n_0(a)$ 的、字符两两不同的字符串 t 恰有 $A_k^{n_0(a)}$ 个。因此恰有 $A_k^{n_0(a)}$ 个二元组 $(a, t) \in P$, 所以恰有 $A_k^{n_0(a)}$ 个字符串使得其形状序列为 a 。□

4.2.3 形状序列与子串

前缀的形状序列与原串的形状序列的关系, 已经在性质 4.1 中说明。

现在我们分析后缀的形状序列的性质。对于字符串 s 和整数 $1 \leq l \leq |s| + 1$, 设 s 的形状序列为 a , $s[l, |s|]$ 的形状序列为 b 。

依次考虑 $i = 1, 2, \dots, |s| - l + 1$:

- 若字符 s_{l+i-1} 在 $s[1, |s|]$ 中首次出现, 那么显然它也在 $s[l, |s|]$ 中首次出现, 因此 $a_{l+i-1} = b_i = 0$;
- 若字符 s_{l+i-1} 在 $s[1, |s|]$ 中并非首次出现, 那么在 $s[1, l+i-2]$ 中, 找到所有字符的最后一次出现位置, 从大到小记作 $j_1 > j_2 > \cdots > j_z$ 。设 $a_{l+i-1} = x$, 即 s_{l+i-1} 上次出现位置为 j_x 。

设 $n_0(b[1, i-1]) = z'$, 由性质 4.1 和性质 4.2, $s[l, l+i-2]$ 中有 z' 种不同字符。那么在 $s[l, l+i-2]$ 中, 所有字符的最后一次出现位置从大到小为 $j_1 > j_2 > \dots > j_{z'}$ 。所以若 $x > z'$, 则 s_{l+i-1} 在 $s[l, |s|]$ 中首次出现, 那么 $b_i = 0$; 否则 s_{l+i-1} 在 $s[l, |s|]$ 中并非首次出现, 所以 $b_i = x$ 。

因此我们得到结论:

性质 4.4. 对于字符串 s 和整数 $1 \leq l \leq |s| + 1$, 设 s 的形状序列为 a , $s[l, |s|]$ 的形状序列为 b 。

$$\text{对于 } 1 \leq i \leq |s| - l + 1, b_i = \begin{cases} a_{l+i-1}, & a_{l+i-1} \leq n_0(b[1, i-1]); \\ 0, & a_{l+i-1} > n_0(b[1, i-1]). \end{cases}$$

根据该性质, 我们可以遍历 $i = 1, 2, \dots, |s| - l + 1$ 递推确定 b_i , 它只和形状序列 a 而不和原字符串 s 有关。我们把子串看作后缀的前缀, 可作如下定义:

定义 4.4. 对于合法形状序列 a 和整数 $1 \leq l \leq r \leq |a|$, 令 a 关于区间 $[l, r]$ 的子形状序列为一个长度为 $r - l + 1$ 的序列 b , 满足:

$$\text{对于 } 1 \leq i \leq r - l + 1, b_i = \begin{cases} a_{l+i-1}, & a_{l+i-1} \leq n_0(b[1, i-1]); \\ 0, & a_{l+i-1} > n_0(b[1, i-1]). \end{cases}$$

a 关于区间 $[l, r]$ 的子形状序列记作 $\text{subshape}(a, l, r)$ 。

特殊地, 对于不在上述范围内的 l, r , $\text{subshape}(a, l, r) = \epsilon$ 。

对于整数 $1 \leq l \leq |a| + 1$, $\text{subshape}(a, l, |a|)$ 统称为 a 的后缀形状序列。

需要指出的是, 子形状序列不同于子序列和连续子序列, 它不仅要求截取连续的一段, 还需要对越界的项进行处理。

由性质 4.1、性质 4.4 立得:

性质 4.5. 设字符串 s 的形状序列为 a , 则子串 $s[l, r]$ 的形状序列为 $\text{subshape}(a, l, r)$ 。

现在讨论合法形状序列和其子形状序列的 0 的个数的关系。

性质 4.6. 对于合法形状序列 a 和整数 l, r , $n_0(\text{subshape}(a, l, r)) \leq n_0(a)$ 。

证明. 设 a 是字符串 s 的形状序列, 则 $\text{subshape}(a, l, r)$ 是 $s[l, r]$ 的形状序列。

所以 $n_0(\text{subshape}(a, l, r))$ 为 $s[l, r]$ 中不同字符数, 自然不超过 s 中不同字符数, 即 $n_0(a)$ 。

□

4.3 算法二: $n = 1, L \leq 1000$

将 S_1 的所有后缀的形状序列插入 Trie。对最终 Trie 的每个结点, 统计其代表的形状序列对应的字符串数。

时间复杂度 $O(L^2)$, 空间复杂度 $O(kL^2)$, 期望得分 9 分。

4.4 算法三： $n = 1, L \leq 10^5$

算法三基于后缀数组 [3]。

沿用后缀数组的思想，我们试图求出任意两个后缀的形状序列的 LCP，并比较字典序大小。

记 S_1 的形状序列为 A ，非空后缀 $S_1[l, L]$ 的形状序列为 $A^{(l)}$ 。设 $A^{(l)}$ 中所有为 0 的位置集合为 P_l ，那么对于 $1 \leq i \leq L - l + 1$ ， $A_i^{(l)} = \begin{cases} A_{l+i-1} & i \notin P_l; \\ 0 & i \in P_l. \end{cases}$

注意到， $i \in P_l$ 当且仅当 $S_{1, l+i-1}$ 在 $S_1[l, L]$ 中第一次出现，可以通过简单的倒序递推求出。

因此，为求出 $\text{LCP}(A^{(l)}, A^{(l')})$ 并比较 $A^{(l)}$ 和 $A^{(l')}$ 的字典序大小，我们可以按照如下步骤进行：

1. 利用后缀数组预处理 A 的任意两个后缀的 LCP；
2. 预处理 P_1, P_2, \dots, P_L ；
3. 设 $P_l \cup P_{l'}$ 含有元素 $i_1 < i_2 < \dots < i_z$ ，称它们为特殊点；
4. 对特殊点直接比较，对相邻两个特殊点间的连续子序列，利用后缀数组求它们的 LCP 后比较。

如果采用 $O(L)$ 时间预处理、 $O(1)$ 时间查询的后缀数组算法，以上算法预处理时间复杂度 $O(Lk)$ ，单次查询时间复杂度 $O(k)$ 。

将 $A^{(1)}, A^{(2)}, \dots, A^{(L)}$ 按照字典序从小到大排序，设排在第 i 位的形状序列是 $A^{(s_{a_i})}$ ， $A^{(i)}$ 排在第 rank_i 位。记 $h_i = \text{LCP}(A^{(s_{a_i})}, A^{(s_{a_{i+1}})})$ ($1 \leq i < L$)。根据后缀数组的相关结论，对于 S_1 的两个非空子串 $S_1[i, i+l-1]$ 和 $S_1[j, j+l-1]$ ($i \neq j$)，若不妨设 $\text{rank}_i < \text{rank}_j$ ，则它们相似当且仅当 $l \leq \min\{h_x \mid \text{rank}_i \leq x < \text{rank}_j\}$ 。

如果采用快速排序，由于单次比较时间为 $O(k)$ ，总时间为 $O(kL \log L)$ ，这是本算法的时间瓶颈。

现在我们可以开始统计了。先考虑怎么快速求一个合法形状序列的所有非空前缀对应的字符串数量。

对于合法形状序列 a ，设它的所有 0 的位置从小到大为 $i_1 < i_2 < \dots < i_z$ 。如果设 $i_{z+1} = |a| + 1$ ，那么 a 的所有长度位于区间 $[i_x, i_{x+1})$ 的前缀均含有 x 个 0，也就是对应 A_k^x 个字符串。因此它的所有非空前缀对应的字符串数量为：

$$\sum_{x=1}^z (i_{x+1} - i_x) A_k^x$$

记 $G(l, r) = |\{s \mid \exists r' \in [l, r], s \sim S_1[l, r']\}|$ ，它可以对 $\text{subshape}(A, l, r)$ 做上述算法求得，单次求算时间复杂度 $O(k)$ 。

现在我们着手解决原问题。

设 $T(i, j)$ 表示全体满足 $l \in \{sa_i, sa_{i+1}, \dots, sa_j\}$, $l \leq r \leq L$ 的子串 $s[l, r]$ 的形状序列集合, $F(i, j)$ 表示 $T(i, j)$ 总共对应的字符串数量。

当 $i = j$ 时 $F(i, j) = G(sa_i, L)$ 。

当 $i < j$ 时, 设 $h_i, h_{i+1}, \dots, h_{j-1}$ 中最小的是 h_m (若最小值有多个, 任取一个)。那么, $h_m = \text{LCP}\{A^{(sa_x)} \mid i \leq x \leq j\}$ 。所以 $u \in T(i, m) \cap T(m+1, j)$ 当且仅当 u 是 $A^{(sa_i)}$ 的长度不超过 h_m 的非空前缀。换句话说, 两部分形状序列的交集是 $T(sa_i, sa_i + h_m - 1)$ 。因此, 将两部分的贡献相加, 再扣除重复贡献, 我们可以写出如下的式子:

$$F(i, j) = F(i, m) + F(m+1, j) - G(sa_i, sa_i + h_m - 1)$$

按照这个式子递归求解 $F(1, L)$ 即为答案。由于 $1 \leq m < L$ 作为最小值点只能出现在一次递归中, 总共递归到的状态数是 $O(L)$ 级别的。

综上所述, 这个算法时间复杂度 $O(kL \log L)$, 空间复杂度 $O(kL)$, 期望得分 20 分。

5 设计新算法解决一般情况下的问题

当 $n \geq 2$ 时, 字符串相互影响较大, 并且难以化归到单字符串的情况下求解, 我们必须另寻出路。

我们考虑怎样的字符串是好的。

定义 5.1. 字符串 T 是 i -好的, 当且仅当存在字符串 T 的一个划分 $T_i T_{i+1} \dots T_n$, 使得 $\forall i \leq j \leq n, T_j \in C_j$ 。特别地, 字符串 T 是 $(n+1)$ -好的, 当且仅当 $T = \epsilon$ 。

引理 5.1. 对于 $1 \leq i \leq n$ 和字符串 T , 设 $T[1, r]$ 是 T 的最长前缀, 使得 $T[1, r] \in C_i$ 。则 T 是 i -好的, 当且仅当 $T[r+1, |T|]$ 是 $(i+1)$ -好的。

证明. 充分性显然, 只需证必要性。

假设 T 是 i -好的, 相应划分为 $T_i T_{i+1} \dots T_n$ 。

由 $T[1, r]$ 的最长性, 得 $|T_i| \leq r$ 。

对于子串 T_j ($i < j \leq n$), 若 $T_j = \epsilon$, 则令 $T'_j = \epsilon$; 否则若 T_j 对应的位置区间是 $[a, b]$, 则令 $T'_j = T[\max\{a, r+1\}, b]$, 它为 T_j 的子串, 由性质 3.2 及 $T_j \in C_j$ 自然可得 $T'_j \in C_j$ 。又 $T[r+1, |T|] = T'_{i+1} T'_{i+2} \dots T'_n$, 故 $T[r+1, |T|]$ 是 $(i+1)$ -好的。□

反复应用引理 5.1, 可得以下定理:

定理 5.1. 对于字符串 T , 设 $R_1 = T$.

对于 $1 \leq i \leq n$, 取 R_i 的最长前缀 $R_i[1, r]$, 使 $R_i[1, r] \in C_i$. 令 $R_{i+1} = R_i[r+1, |R_i|]$.

则 T 是好的, 当且仅当 $R_{n+1} = \epsilon$.

获取了一个字符串为好字符串的充要条件后, 我们需要建立模型来描述上述定理, 从而进行计数。

5.1 算法四: $k = 2, L \leq 1000$

不难想到一种思路, 即构造一个自动机使得其恰好接受好的字符串, 随后利用递推对其接受的字符串数计数。

对于每个字符串 S_i , 令部分 DFA $M_i = (Q_i, \Sigma, \delta_i, \bar{q}_i, Q_i)$ 为一棵 Trie。枚举 f , 将 S_i 的每个后缀作用 f 后插入到 M_i 上。则 M_i 接受的字符串集合恰为 C_i 。

我们现在考虑构造部分 DFA $M'_i = (Q'_i, \Sigma, \delta'_i, \bar{q}'_i, F'_i)$, 使得其恰好接受所有 i -好的字符串。

首先构造 M'_{n+1} 。这是容易的: $Q'_{n+1} = F'_{n+1} = \{\bar{q}'_{n+1}\}$, 无转移。

对于 $i = n, n-1, \dots, 1$, 我们递推地构造 M'_i 。根据引理 5.1, 我们应该在自动机 M_i 中接受尽量长的前缀, 并将剩余部分输入自动机 M'_{i+1} 中继续判定。

因此, 构造如下:

- $Q'_i = F'_i = Q_i \cup Q'_{i+1}$
- $\bar{q}'_i = \bar{q}_i$
- 对于 $q \in Q'_{i+1}$ 及 $\alpha \in \Sigma$, $\delta'_i(q, \alpha) = \delta'_{i+1}(q, \alpha)$ 。
- 对于 $q \in Q_i$ 及 $\alpha \in \Sigma$, $\delta'_i(q, \alpha) = \begin{cases} \delta_i(q, \alpha), & \delta_i(q, \alpha) \neq 0; \\ \delta'_{i+1}(\bar{q}'_{i+1}, \alpha), & \delta_i(q, \alpha) = 0. \end{cases}$

那么, 自动机 M'_1 接受的就是所有好的字符串。

记 $f(u)$ 表示在 M'_1 中状态 u 接受的字符串数, 进行递推。 $\sum_{u \in Q'_1} f(u)$ 即为所求。

这个算法, 总时空复杂度 $O(L^2)$, 期望得分 15 分。

5.2 算法五: $L \leq 1000$

根据之前的分析, 形状序列是我们判定字符串相似的有力工具。我们试图从形状序列入手。

首先, 我们将定义 5.1 与引理 5.1 改写 to 形状序列上。

定义 5.2. 合法形状序列 A 是 i -好的, 当且仅当存在整数 $0 = p_{i-1} \leq p_i \leq \dots \leq p_n = |A|$, 使得 $\forall i \leq j \leq n$, $\text{subshape}(A, p_{j-1} + 1, p_j)$ 是 S_j 的至少一个子串的形状序列。特别地, 合法形状序列 A 是 $(n+1)$ -好的, 当且仅当 $A = \epsilon$ 。

一个合法形状序列是好的, 当且仅当它是 1-好的。

对照定义 5.1 与定义 5.2, 可得:

引理 5.2. 一个字符串是 i -好的, 当且仅当其合法形状序列是 i -好的。

引理 5.3. 对于 $1 \leq i \leq n$ 和合法形状序列 A , 设 $A[1, r]$ 是 A 的最长前缀, 使得 $A[1, r]$ 是 S_i 的至少一个子串的形状序列。则 A 是 i -好的, 当且仅当 $\text{subshape}(A, r+1, |A|)$ 是 $(i+1)$ -好的。

我们沿用算法四的思路, 对于各字符串的所有子串的形状序列建立自动机。以下我们讨论的字符集均为 $\Sigma' = \{0, 1, \dots, k\}$ 。

对于每个字符串 S_i , 令部分 DFA $M_i = (Q_i, \Sigma', \delta_i, \bar{q}_i, Q_i)$ 为一棵 Trie, 将字符串 S_i 的所有后缀的形状序列插入到 M_i 上。则 M_i 能接受所有 S_i 的子串的形状序列。记 z_q 表示状态 q 接受的合法形状序列中 0 的个数。

对于 $a > z_q$, 令 $\delta_i(q, a) = \delta_i(q, 0)$ 。那么根据定义 4.4, 合法形状序列 A 的子形状序列 $\text{subshape}(A, l, r)$ 被自动机 M_i 接受, 当且仅当其连续子序列 $A[l, r]$ 被自动机 M_i 接受。

构造部分 DFA $M'_i = (Q'_i, \Sigma', \delta'_i, \bar{q}'_i, F'_i)$ 以接受所有 i -好的合法形状序列, 构造同算法四。那么一个合法形状序列是好的, 当且仅当它被 M'_1 接受。但是, M'_1 接受的序列并不一定是合法形状序列。

现在我们着手统计答案。记 $f(q)$ 表示形状序列被自动机 M'_1 的状态 q 接受的字符串数。考虑 q 接受的所有合法形状序列, 设为 A_1, A_2, \dots, A_m 。由性质 4.6, 有 $z_q \leq n_0(A_i)$ 。设 $n_0(A_i) = z$, 对每个整数 $0 \leq a \leq z$, 考虑接受合法形状序列 $A_i \cdot (a)$ 的状态及这一转移贡献的字符串数:

- $a = 0$, 那么 $A_i \cdot (a)$ 被状态 $\delta'_1(q, 0)$ 接受, 字符串数: $A_k^{z+1} = (k-z)A_k^z$ 。
- $1 \leq a \leq z_q$, 那么 $A_i \cdot (a)$ 被状态 $\delta'_1(q, a)$ 接受, 字符串数: A_k^z 。
- $z_q < a \leq z$, 那么 $A_i \cdot (a)$ 被状态 $\delta'_1(q, a) = \delta'_1(q, 0)$ 接受, 字符串数: A_k^z 。

那么 A_i 对 $f(\delta'_1(q, 0))$ 贡献了 $(k-z_q)A_k^{n_0(A_i)}$, 对 $f(\delta'_1(q, a))$ ($1 \leq a \leq z_q$) 贡献了 $A_k^{n_0(A_i)}$ 。结合 $f(q) = \sum_{i=1}^m A_k^{n_0(A_i)}$, 可得一个状态 q 对 $f(\delta'_1(q, 0))$ 贡献了 $(k-z_q)f(q)$, 对 $f(\delta'_1(q, a))$ ($1 \leq a \leq z_q$) 贡献了 $f(q)$ 。对于 $a > z_q$, 转移 $\delta'_1(q, a)$ 的贡献已经在 $\delta'_1(q, 0)$ 中计算过, 可以删去。至此, 我们可将自动机 M'_1 对应为一张加权有向无环图, 对其进行带权路径计数。 $\sum_{q \in Q'_1} f(q)$ 即为所求。

值得一提的是, 在删去多余的转移后, M'_i 事实上等同于:

- $Q'_i = F'_i = Q_i \cup Q'_{i+1}$
- $\bar{q}'_i = \bar{q}_i$
- 对于 $q \in Q'_{i+1}$ 及 $0 \leq a \leq z_q$, $\delta'_i(q, a) = \delta'_{i+1}(q, a)$ 。
- 对于 $q \in Q_i$ 及 $0 \leq a \leq z_q$, $\delta'_i(q, a) = \begin{cases} \delta_i(q, a), & \delta_i(q, a) \neq 0; \\ \delta'_{i+1}(\bar{q}'_{i+1}, 0), & \delta_i(q, a) = 0. \end{cases}$

对于每个转移 $\delta_i(q, a) \neq 0$, 若 $a = 0$, 其权值为 $k - z_q$, 否则其权值为 1。
这个算法, 时空复杂度为 $O(kL^2)$, 期望得分 39 分。

5.3 后缀自动机的扩展——子形状自动机

不难发现, 算法五的劣势在于自动机的状态数过多。很多状态彼此等效, 因此有巨大的优化空间。事实上, 对于合法形状序列 A , 本节将构造一种状态数与转移数均为 $O(k|A|)$ 的部分 DFA, 使得其恰好接受 A 的所有子形状序列。

既然我们可以建立后缀自动机 [1, 4] 来接受一个字符串的所有子串, 联想一下, 能否建立子形状自动机, 使得其接受一个合法形状序列的所有子形状序列呢?

5.3.1 子形状自动机的定义及其大小

现在设 A 为某合法形状序列。对于 A 的子形状序列 B , 设 B 在 A 中的右端点集合为 $R_A(B) = \{r \mid \text{subshape}(A, r - |B| + 1, r) = B\}$ 。特别地, 令 $R_A(\epsilon) = \{0, 1, \dots, |A|\}$ 。

若 A 的子形状序列 B, B' 满足 $|B| \leq |B'|$, 且 $R_A(B) \cap R_A(B') \neq \emptyset$, 则 B 是 B' 的后缀形状序列, $R_A(B) \subseteq R_A(B')$ 。因此, 所有子形状序列的右端点集合, 若相同的只保留一个, 则包含关系可构成一棵树。一个结点要么有至少两个子结点, 要么是一个元素出现的最深的集合。因此, 可得不同的右端点集合只有不超过 $2|A| + 1$ 种。

对于 A 的子形状序列 B, B' , 我们说 $B \equiv_A B'$, 当且仅当 $R_A(B) = R_A(B')$ 。显然 \equiv_A 是一个等价关系。在 \equiv_A 下, 子形状序列 B 所在的等价类记为 $[B]_A$ 。它有以下性质:

引理 5.4. 若 B, C, B' 是非负整数序列, $BC, B'C$ 都是 A 的子形状序列, $B \equiv_A B'$, 则 $BC \equiv_A B'C$ 。

推论 5.1. 若 B, C, B' 是非负整数序列, BC 是 A 的子形状序列, $B \equiv_A B'$ 且 $n_0(B) = n_0(B')$, 则 $B'C$ 也是 A 的子形状序列, $BC \equiv_A B'C$ 且 $n_0(BC) = n_0(B'C)$ 。

引理 5.5. 若 A 的子形状序列 B, B' 满足 $B \equiv_A B'$, 则 B 是 B' 的后缀形状序列, 或者反之。

因此, 存在部分 DFA $D_A = (Q, \Sigma', \delta, \bar{q}, Q)$, 使得其每个状态均形如 $q = (E, z)$, 其中 E 为 \equiv_A 下的一个等价类, z 为一个非负整数, 表示该状态接受 A 的子形状序列 B 当其仅当 $[B]_A = E$ 且 $n_0(B) = z$ 。对于 $q = (E, z)$, 记 $E_q = E, z_q = z$ 。构造如下 [2, 引理 2]:

- 初始状态 $\bar{q} = ([\epsilon]_A, 0)$ 。
- 对于状态 q 和状态 q' , 若存在序列 B 和非负整数 a 使得 B 被 q 接受, $B \cdot (a)$ 被 q' 接受, 则 $\delta(q, a) = q'$ 。

这个部分 DFA D_A 称作形状序列 A 的子形状自动机。

由于不同的右端点集合数为 $O(|A|)$ 级别, 0 的个数不超过 k , 所以 D_A 的状态数为 $O(k|A|)$ 。这个级别是容易达到的, 只需构造 $(0)^k(1)^{|A|-k}$ 。

以初始状态为根, 取 D_A 对应的有向无环图的一棵外向树。对于图的每条非树边 $\langle u, v \rangle$, 从根沿树边到 u , u 到 v , v 沿树边到任意一个叶结点的路径对应了 A 的至少一个后缀形状序列, 且一个后缀形状序列与其转移路径的第一条非树边 (如果存在) 对应。所以非树边至多 $|A|$ 条。因此, D_A 的转移数与状态数同阶。

5.3.2 子形状自动机的构造算法

现在我们给出 D_A 的构造算法。

对于每个状态, 定义其**代表元**为其接受的序列中最长的那一个。若 B 是 q 的代表元, 则 q 接受的所有字符串均为 B 的后缀形状序列。

因此对于一个状态 q , 设它的代表元为 B , 则令 $len_q = |B|$ 。对于 B 的后缀形状序列 $B_l = \text{subshape}(B, l, |B|)$, $|R(B_l)|$ 随 l 非严格递增 (由性质 4.5), $n_0(B_l)$ 随 l 非严格递减 (由性质 4.6)。于是可得: 存在非负整数 L 使得 q 接受了 B 的长度不小于 L 的所有后缀形状序列。对于 $q \neq \bar{q}$, $L \neq 0$, 则定义状态 q 的**后缀连接** $parent_q$ 是接受 B 的长度为 $L-1$ 的后缀形状序列的状态。特别地, 记 $parent_{\bar{q}} = 0$ 。

定义状态 q 的**后缀链** $SC(q)$ 是指从 q 开始沿后缀连接跳转所经过的状态序列。形式化地, 可递归定义: $SC(\bar{q}) = (\bar{q})$, $SC(q) = (q) \cdot SC(parent_q)$ 。那么, $SC(q)$ 接受了 q 的代表元的所有后缀形状序列。

考虑从空串开始, 每次在末尾添加一个字符, 并维护每个状态 q 对应的 $z_q, len_q, parent_q$ 。 D_ϵ 是平凡的。

设序列 B 、非负整数 a , 使得 $B' = B \cdot (a)$ 为合法形状序列, 假设现在已知 D_B , 考虑在其基础上构造 $D_{B'}$ 。则新加入的字符串为 B' 的所有后缀形状序列。

对于整数 $1 \leq l \leq |B'|$ 有:

$$\text{subshape}(B', l, |B'|) = \begin{cases} B_l \cdot (a), & a \leq n_0(B_l); \\ B_l \cdot (0), & a > n_0(B_l). \end{cases}$$

以下记 $tr(q, a) = \begin{cases} a, & a \leq z_q; \\ 0, & a > z_q. \end{cases}$

于是, 若 q 接受 B_l , 则 $subshape(B', l, |B'|)$ 应被 $\delta(q, tr(q, a))$ 接受。

以下设在自动机 D_B 中, B 由状态 $sink_B = ([B]_B, n_0(B))$ 接受。

增加一个字符的过程与一般的后缀自动机类似, 不同点为:

- 对于 $sink_B$ 的后缀链上的状态 q , 其对应转移的项为 $tr(q, a)$ 而不是 a 。
- 对于 $sink_B$ 的后缀链上的状态 q 、当前增加的状态 p , 若 $z_q + [tr(q, a) = 0] < z_p$, 则需要按照后缀自动机的拆分算法来拆分 p 。

明确起见, 我们给出伪代码:

Algorithm 1 *update*(B, a)

Require: 当前自动机为 B 的子形状自动机, $B \cdot (a)$ 为合法形状序列

Ensure: 修改后自动机为 $B' = B \cdot (a)$ 的子形状自动机

```

1:  $q := sink_B$ 
2: 新建状态  $p$ 
3:  $sink_{B'} := p$ 
4:  $z_p := z_q + [a = 0]$ 
5:  $len_p := len_q + 1$ 
6: while  $q \neq 0$  and  $\delta(q, tr(q, a)) = 0$  do
7:   if  $z_q + [tr(q, a) = 0] < z_p$  then
8:      $p := split\_new\_node(p, q, tr(q, a))$ 
9:   end if
10:   $\delta(q, tr(q, a)) := p$ 
11:   $q := parent_q$ 
12: end while
13: if  $q = 0$  then
14:    $parent_p := q_0$ 
15:   return
16: end if
17:  $r := \delta(q, tr(q, a))$ 
18: if  $len_r = len_q + 1$  then
19:    $parent_p := r$ 
20: else
21:    $parent_p := split\_old\_node(r, q, tr(q, a))$ 
22: end if

```

在算法 1 中，我们用到了 $split_new_node(p, q, c)$ 表示从新状态 p 中拆分出从 $\delta(q, c)$ 转移而来的区间。该算法实现如下：

Algorithm 2 $split_new_node(p, q, c)$

- 1: 新建状态 p'
 - 2: $z_{p'} := z_q + [c = 0]$
 - 3: $len_{p'} := len_q + 1$
 - 4: $parent_p := p'$
 - 5: **return** p'
-

在算法 1 中，我们用到了 $split_old_node(r, q, c)$ 表示从旧状态 r 中拆分出从 $\delta(q, c)$ 转移而来的区间。该算法实现如下：

Algorithm 3 $split_old_node(r, q, c)$

- 1: 新建状态 r'
 - 2: $z_{r'} := z_q + [c = 0]$
 - 3: $len_{r'} := len_q + 1$
 - 4: $parent_{r'} := parent_r$
 - 5: $parent_r := r'$
 - 6: **for all** $x \in \Sigma$ **do**
 - 7: $\delta(r', x) := \delta(r, x)$
 - 8: **end for**
 - 9: **while** $q \neq 0$ **and** $\delta(q, c) = r$ **do**
 - 10: $\delta(q, c) := r'$
 - 11: $q := parent_q$
 - 12: **end while**
 - 13: **return** r'
-

现在我们分析该算法的时间复杂度。注意到，如果使用数组保存各转移，空间复杂度为 $O(k^2|A|)$ ，任意单次操作时间为 $O(1)$ 。每次增量，至多拆分 k 个新状态，每次拆分需 $O(1)$ 时间；至多拆分一个旧状态，每次拆分中，复制信息和转移需 $O(k)$ 时间。故这些操作的总时间为 $O(k|A|)$ 。因此只需考虑新增或修改后缀链的转移的时间。

考虑 $|SC(sink_{B'})|$ 相对于 $|SC(sink_B)|$ 的变化。对于状态 $\bar{q} \neq p \in SC(sink_B)$ ，均存在 $q \in SC(sink_B)$ ，使得 $\delta(q, tr(q, a)) = p$ 。对于新增或修改后缀链的转移，其要么指向新状态，要么指向拆分出的旧状态，因此至多有 $k + 1$ 种不同的后继状态。因此若新增或修改了后缀链上 t 个状态的转移，则有

$$|SC(sink_{B'})| \leq |SC(sink_B)| - t + k + 1$$

初始 $|SC(\bar{q})| = 1$, 且 $|SC(sink_B)| > 0$ 恒成立。所以新增或修改后缀链的转移的总次数也为 $O(k|A|)$ 级别。

所以我们可得结论:

定理 5.2. 对于合法形状序列 A , 可以在 $O(k|A|)$ 时间、 $O(k^2|A|)$ 空间内, 构造一种状态数与转移数均为 $O(k|A|)$ 的部分 DFA, 使得其恰好接受 A 的所有子形状序列。

5.4 算法六: $k \leq 5$

记 S_i 的形状序列为 A_i , M_i 为 A_i 的子形状自动机, 其余部分同算法五。

优化后, M'_1 的状态数为 $O(\sum_{i=1}^n k|Q_i|) = O(kL)$, 每个状态的有效转移数均为 $O(k)$, 因此总时空复杂度 $O(k^2L)$, 期望得分 70 分。结合算法三, 期望得分 81 分。

5.5 合并相同转移

事实上, 子形状自动机还有优化的空间。

在合法形状序列 A 的子形状自动机 $D_A = (Q, \Sigma', \delta, \bar{q}, Q)$ 中, 若状态 $q \neq q'$ 但 $E_q = E_{q'}$, 则对于 $1 \leq a \leq \min\{z_q, z_{q'}\}$, 有 $E_{\delta(q,a)} = E_{\delta(q',a)}$ 。对于 $1 \leq a \leq z_q$, 若 $\delta(q,a)$ 存在, 则有 $z_{\delta(q,a)} = z_q$ 。

于是, 对于等价类 E , 若其中最长的序列为 B , 我们定义 $\delta(E, a) = [B \cdot (a)]_A$ 。特别地, 若 $B \cdot (a)$ 不是 A 的子形状序列, 则 $\delta(E, a) = 0$ 。于是, 对于 $1 \leq a \leq z_q$, 均有 $\delta(q, a) = (\delta(E_q, a), z_q)$ 。因此, 对于每个状态 $q = (E, z)$, 都只需要额外记录 $\delta(q, 0)$ 。

由于非空的等价类只有 $O(|A|)$ 个, 可得只需要 $O(k|A|)$ 空间来记录转移, 因此我们得到更强的结论:

定理 5.3. 对于合法形状序列 A , 可以在 $O(k|A|)$ 时间、 $O(k|A|)$ 空间内, 构造一种状态数与转移数均为 $O(k|A|)$ 的部分 DFA, 使得其恰好接受 A 的所有子形状序列。

5.6 算法七: 无特殊限制

上述优化无法直接用于算法五中构造的 M'_1 , 考虑进一步优化。仍然设 $f(q)$ 表示在自动机 M'_1 对应的加权有向无环图中, 从 \bar{q}'_1 到 q 的所有路径上, 各边权值积之和。

现在我们放弃显式建立自动机 M'_1 。对于 \equiv_{A_i} 的等价类 E , 统一考虑所有状态 (E, z) 在递推中对后继状态的贡献。

在原自动机中, 转移只有 $O(kL)$ 条, 这部分转移可以直接枚举。对于新加入的转移, 设 M_i 的状态 $q = (E, z)$ 和整数 $0 \leq a \leq z$ 使得 $M_i(q, a) = 0$, 对于这样的转移, 我们有:

- 若 $a = 0$, 则枚举状态直接转移, 总共有 $O(kL)$ 条转移。

- 若 $1 \leq a \leq z$, 这意味着 $\delta_i(E_q, a) = 0$, 则 q 对 $f(\delta'_{i+1}(\bar{q}'_{i+1}, 0))$ 贡献了 $f(q)$ 。可以换一个角度看: 对于每个等价类 E , 记 m_E 和 M_E 分别表示使得状态 (E, z) 存在的 z 的最小值和最大值。所有状态 (E, z) 总共对 $f(\delta'_{i+1}(q'_{i+1}, 0))$ 贡献了 $\sum_{z=\max\{m_E, a\}}^{M_E} f((E, z))$, 这可以用后缀和快速计算。所以这部分的时间复杂度优化到了 $O(kL)$ 。

这样, 这个算法的总时空复杂度降低到 $O(kL)$, 期望得分 100 分。

6 讨论与扩展

6.1 子形状自动机的优点

子形状自动机在处理子串的相似关系时, 较基于后缀数组的算法 (见算法三), 构建的时间复杂度更低, 空间复杂度不变。由于自动机的结构更加直观, 在面对一些问题如本题时, 容易设计出优秀的算法; 而本题特意抓住后缀数组算法的弱点来命题, 暴露出了后缀数组或后缀树在处理字符串匹配问题的场景中信息大量压缩、不够直观的问题。

6.2 子形状自动机的局限性

子形状自动机的各项复杂度仍然与字符集大小有关, 难以处理字符集较大的情况。不过, 基于后缀数组或后缀树的算法同样无法处理字符集较大的情况。

6.3 子形状自动机在其它一类关系下的应用

本题中, 从字符串对应到形状序列, 可以用自动机变换 [2]³ $M: \Sigma^* \rightarrow (\Sigma')^*$ 描述。

$M: \Sigma^* \rightarrow (\Sigma')^*$ 是自动机变换, 当且仅当存在四元组 $(Q, M_Q, M_{\Sigma'}, \bar{q})$, 其中, Q 是一个有限的状态集合, $M_Q: Q \times \Sigma \rightarrow Q$ 是转移函数, $M_{\Sigma'}: Q \times \Sigma \rightarrow \Sigma'$ 是变换函数, $\bar{q} \in Q$ 是初始状态, 满足 $\forall s \in \Sigma^*$, 存在状态序列 $(q_0, \dots, q_{|s|})$, 使得 $q_0 = \bar{q}$, $q_i = M_Q(q_{i-1}, s_i)$, $M(s)_i = M_{\Sigma'}(q_{i-1}, s_i)$, 且 $|M(s)| = |s|$ 。其中使得 $|Q|$ 最小的四元组 $(Q, M_Q, M_{\Sigma'}, \bar{q})$ 称为 M 的最小自动机 [2, 引理 2]。

本题主要利用了等价关系 \sim 的下列性质:

- 存在自动机变换 $M: \Sigma^* \rightarrow (\Sigma')^*$, 使得 $\forall s, t \in \Sigma^*$, $s \sim t \iff M(s) = M(t)$ 。仍然称 $M(s)$ 为 s 的形状序列, 如果 $a \in \text{Im}M$, 称 a 为合法形状序列。
- 一个合法形状序列的后缀, 可以按照长度分成 K 个区间, 每个区间均被 M 的最小自动机的同一个状态接受。

³这里与原文的定义略有区别。

对于满足这些性质的等价关系，利用本文的算法，均能在 $O(K|a|)$ 时间、 $O(K|a||\Sigma'|)$ 空间内建立合法形状序列 a 对应的子形状自动机，其状态数、转移数均为 $O(K|a|)$ 级别，从而解决这类子串等价关系问题。如果 $|\Sigma'|$ 较大，还可以利用可持久化线段树保存转移，时空复杂度均为 $O(K|a|\log|\Sigma'|)$ 。

这类性质不但包括本题中的相似，还包括例如：

例 6.1. 在字符集 Σ 上定义一个全序 \leq ，对于 $s, t \in \Sigma^*$ ， $s \sim t$ 当且仅当 $|s| = |t|$ 且 $\forall 1 \leq i, j \leq |s|$ ， $[s_i \leq s_j] = [t_i \leq t_j]$ 。

解. 对于字符串 s ，其形状序列为非负整数序列 a ，使得 $|a| = |s|$ 。设在 $s[1, i]$ 中出现的各字母中， s_i 为第 x 小的。若 s_i 在 $s[1, i-1]$ 中出现过，则 $a_i = 2x$ ，否则 $a_i = 2x-1$ 。 $\Sigma' = \{1, 2, \dots, 2k-1\}$ ， $K = |\Sigma| + 1$ 。

例 6.2. Σ 是 M 以内全体正整数集，对于 $s, t \in \Sigma^*$ ， $s \sim t$ 当且仅当 $|s| = |t|$ 且 $\forall 1 \leq i \leq |s|$ ， $s_i / \gcd(s_1, s_2, \dots, s_i) = t_i / \gcd(t_1, t_2, \dots, t_i)$ 。

解. 对于序列 $s \in \Sigma^*$ ，其形状序列为非负整数序列 a ，使得 $|a| = |s|$ ， $a_i = s_i / \gcd(s_1, s_2, \dots, s_i)$ ， $\Sigma' = \{1, 2, \dots, M\}$ ， $K = \lfloor \log_2 M \rfloor + 1$ 。

7 总结

7.1 命题思路

在 2019 年由日本经济新闻社与 AtCoder 共同主办的第二届全日统一编程之王决赛中，出现了一道新颖的试题。该题的研究对象是一类与其反串相似（定义同本题）的字符串，参考解法通过巧妙地扩展 Manacher 算法，并对这种情况下的 Manacher 算法的时间复杂度进行仔细的分析，最终得到了一种高效而简洁的解法。作者学习该题时很受启发，进一步地对类似问题进行了扩展研究。

于是，同一个字符串的两个子串的相似关系首先进入了作者的视线。针对此类问题，以往有一种基于后缀数组的算法。而作者对此类问题下后缀自动机的扩展应用展开了研究，通过引入辅助序列描述等价类、将转移不同的等价类分开，发现了在该类问题上相较于后缀数组更高效、更直观、更简便的数据结构——子形状自动机。

为了进一步体现子形状自动机建图直观的优势，作者将 2017 年山东一轮集训《字符串》⁴与子形状自动机结合，命制了《太阳神的宴会》一题。

本题在部分分的设计上，覆盖了单串情形的经典算法，选手思考单串情形，可以发现形状序列这一重要工具。在多串情形下建立自动机，对选手的知识和模型积累提出了一定要求，作者对不同层次的算法给出了多样的部分分，引导选手往正解的方向思考。

⁴<https://loj.ac/p/6071>

7.2 算法扩展

本文探究了一类基于自动机变换的等价关系，针对这类等价关系提出了子形状自动机及其构造算法，直观、快速、简便地处理了本质不等价子串的有关问题。在题中相似关系的特殊情形下，还通过合并相同转移节省了空间，在这一问题上取得了优于原有算法的进展。我希望《太阳神的宴会》这道题能够给大家带来更多关于子形状自动机以及字符串的等价类问题的思考。

致谢

感谢父母的养育之恩。

感谢中国计算机学会给予的交流机会。

感谢福州一中陈颖老师的关心与指导。

感谢清华大学陈俊锟学长、钟知闲学长、吴作同学长的关心与指导。

感谢清华大学陈俊锟学长、福州一中林昊翰同学和福州一中信息学竞赛组其他同学为本文审稿。

感谢 Universal OJ 为本次自主互测提供平台。

感谢互测的参赛选手参与本题的做题情况调查。

感谢各位的阅读。

参考文献

- [1] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M.T. Chen, and J. Seiferas. The smallest automation recognizing the subwords of a text. *Theoretical Computer Science*, pages 31–55, 1985.
- [2] A. Nerode. Linear automaton transformations. *Proceedings of The American Mathematical Society*, pages 541–541, 1958.
- [3] 刘汝佳、陈锋. 算法竞赛入门经典, pages 219–221. 清华大学出版社, 2012.
- [4] 张天扬. 后缀自动机及其应用. In 2015 年信息学奥林匹克中国国家集训队论文集, 2015.

后缀树的构建

广州市第二中学 代晨昕

摘要

后缀树是处理字符串匹配问题的利器。本文主要介绍它的两种在线构建方法，并给出了一些应用。

1 前言

后缀数据结构，如后缀自动机和后缀树，在若干年以前进入了国内信息学竞赛选手们的视野，现在已经有了广泛的应用。然而，不同于后缀自动机，国内鲜有介绍后缀树构建方法的资料，本文力图弥补这方面的空白。

本文第二、三节主要讲述关于字符串和后缀树的一些基础定义。第四节将介绍从后往前构造后缀树的方法。第五节将介绍从前向后构造后缀树的方法。第六节介绍了一些例题。第七节总结全文。

2 基础定义

记构建后缀树的母串为 S ，长度为 n 。

令 $S[i]$ 表示 S 中的第 i 个字符，其中 $1 \leq i \leq n$ 。

令 $S[l, r]$ 表示 S 中第 l 个字符至第 r 个字符组成的字符串，称为 S 的一个子串。

记 $S[i, n]$ 为 S 的以 i 开头的后缀， $S[1, i]$ 为 S 的以 i 结尾的前缀。

若 $str1$, $str2$ 为单一字符或字符串，令 $str1 + str2$ 表示将 $str1$ 和 $str2$ 拼接后得到的字符串。

如无特殊说明，本文默认字符串 S 的字符集为所有小写字母，即字符集大小为常数。

3 后缀树的定义

定义字符串 S 的**后缀 trie** 为将 S 的所有后缀插入至 trie 树中得到的字典树。在后缀 trie 中，节点 x 对应的字符串为从根节点走到 x 的路径上经过的字符拼接而成的字符串。记后

缀 trie 中所有对应 S 的某个后缀的节点为后缀节点。

容易看出后缀 trie 的优越性质：它的非根节点恰好能接受 S 的所有本质不同非空子串。但构建后缀 trie 的时空复杂度均为 $O(n^2)$ ，在很多情况下不能接受，所以我们引入后缀树的概念。

如果令后缀 trie 中所有拥有多于一个儿子的节点和后缀节点为关键点，定义只保留关键点，将非关键点形成的链压缩成一条边形成的压缩 trie 树为**后缀树 (Suffix Tree)**。

如果仅令后缀 trie 中所有拥有多于一个儿子的节点和叶结点为关键点，定义只保留关键点形成的压缩 trie 树为**隐式后缀树 (Implicit Suffix Tree)**。容易看出隐式后缀树为后缀树进一步压缩后得到的结果。

在后缀树和隐式后缀树中，每条边对应一个字符串；每个非根节点 x 对应了一个字符串集合，为从根节点走到 x 的父亲节点 fa_x 经过的字符串，拼接上 fa_x 至 x 的树边对应的字符串的任意一个非空前缀。

令 mx_x 表示 x 对应的字符串集合中最长的字符串， len_x 表示 mx_x 的长度。

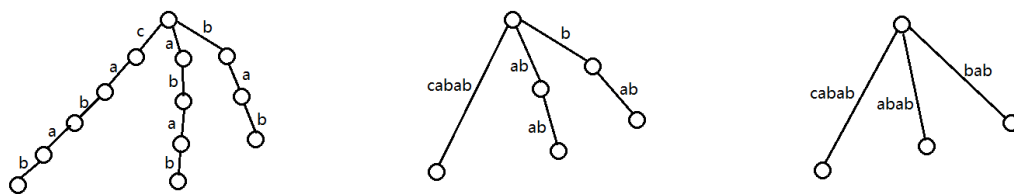


图 1: 字符串”cabab”对应的后缀 trie、后缀树和隐式后缀树

引理 3.1: 长度为 n 的字符串 S 构建的后缀树节点数不超过 $2n$ 。

证明. 考虑将 S 的后缀逐个插入至后缀 trie 中。从第二次插入开始，每次最多新增一个拥有多于一个儿子的节点和一个后缀节点，所以后缀 trie 中关键点个数最多为 $2n$ 个，得证。

4 支持从后往前添加字符的后缀树构建

一个广为人知的结论是：后缀自动机的 $parent$ 数组即为反串的后缀树^[1]，利用反串的后缀自动机来构建后缀树也成为了现在构建后缀树的主流算法。接下来将从后缀树的角度来描述此算法的过程。

4.1 一些定义

在后缀 trie 中，记 $trans_{x,c}$ 表示在节点 x 对应的字符串开头添加字符 c 后对应的节点，如果不存在设为空状态 $NULL$ 。

对 S 的某子串 str , 令 $leftpos_{str}$ 表示 str 在 S 中出现位置的左端点集合。

由后缀树的定义推出: 在后缀树上每个节点 x 对应的字符串集合的 $leftpos$ 集合相同, 可以记为 $leftpos_x$, 且不同的两个节点的 $leftpos$ 集合不相同。易得节点 x 对应的所有字符串开头添加字符 c 后得到的字符串的 $leftpos$ 集合都相同, 在后缀树中一定对应同一节点 y , 可以直接定义 $trans_{x,c} = y$ 。实际上, 这就是反串的后缀自动机的转移边。

4.2 算法过程

考虑增量法, 从后往前加入字符。假设已经维护好 $S[i+1, n]$ 的后缀树以及所有节点的 $trans$ 转移边, 现在需要在后缀树中加入后缀 i 。设 $S[i] = x$ 。

记录后缀 $i+1$ 对应的节点 $last$, 新增后缀节点 np 表示 $S[i, n]$ 。现在, 我们希望求出 $S[i, n]$ 在原树中出现过的最长前缀。由于 $S[i, n]$ 的前缀即为 $S[i+1, n]$ 的前缀在前端添加字符 x , 所以找出 $last$ 的深度最深的祖先 p 使得 $trans_{p,x}$ 存在, 那么 $x + mx_p$ 就是 $S[i, n]$ 在原树中出现过的最长的前缀。令 $q = trans_{p,x}$, 有两种情况:

1. $mx_q = x + mx_p$, 即 $len_q = len_p + 1$ 。此时在后缀树中会新增一条 np 至 q 的边, 将 fa_{np} 设为 q 即可。

2. $len_q > len_p + 1$ 。此时在 q 与 fa_q 之间需要新建节点 nq , 令 $len_{nq} = len_p + 1$, 并将 fa_{nq} 设为 fa_q , fa_{np} 和 fa_q 设为 nq 。同时, 我们还需要维护 $trans$ 的变化。 nq 的转移边可以直接复制 q 的转移边得到。同时, $trans_{p,x}$ 需要设为 nq ; 如果 $trans_{fa_p,x} = q$, 也需要重新设为 nq ……从 p 点不断往根移动, 不断修改转移边, 直到发现当前节点 $p1$ 添加字符 x 不会转移到 q 为止。

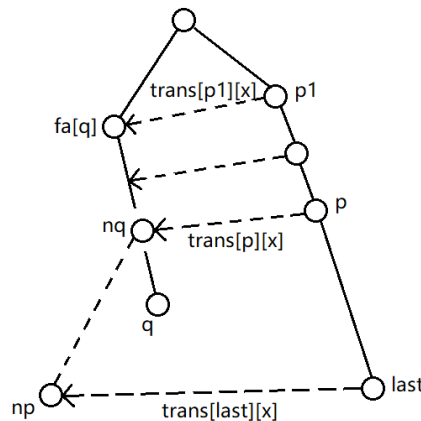


图 2: 情况 2 示意图

最后, 从 $last$ 到 p 之间的节点也需要新增到 np 的转移, 再将 $last$ 设为 np 即可。

4.3 算法时间复杂度证明

算法的正确性是显然的，但是时间复杂度仍有疑问。

引理 4.1: 所有 *trans* 转移边的个数为 $O(n)$ 级别（事实上，也与字符集大小无关）。
即证反串后缀自动机转移边数量为 $O(n)$ 级别。证明可见 [2]，这里不再赘述。

考虑算法过程，除了情况 2 中“将连向 q 的转移边重新设为 nq ”，其余步骤都伴随着新增节点或转移边，所以只需考虑这一步的时间复杂度即可。

设节点 x 的深度为 Dep_x 。考虑 $Dep_{fa_{nq}}$ 和 Dep_{p1} 。由于 fa_{nq} 的任何祖先（不包括 *root*）一定可以由某个 $p1$ 的祖先（包括 *root*）通过前端添加字符 x 转移而来，所以有 $Dep_{fa_{nq}} \leq Dep_{p1} + 1$ 。设转移边重定向进行了 k 次，那么有：

$$Dep_{np} = Dep_{fa_{nq}} + 2 \leq Dep_{p1} + 3 = Dep_p - k + 3 \leq Dep_{last} - k + 3$$

将每一步操作后的 Dep_{last} 设为势能函数。每次转移边重定向都伴随着 Dep 的减小，且 Dep 每次只会增加常数，通过势能分析可以得出这一步操作的总时间复杂度为均摊 $O(n)$ 。

综上所述，此算法构建后缀树的时间复杂度为 $O(n)$ 。

4.4 算法总结

当字符集大小视为常数时，本算法达到了 $O(n)$ 的时间复杂度下界，同时实现不算复杂，是离线构建后缀树的首选。然而，本算法只能支持在前端动态插入字符构造后缀树，在很多问题中，我们需要支持动态末端插入的构造方法，在下一节中将介绍另一种支持动态末端插入的后缀树构建算法——Ukkonen 算法 [3]。

5 支持从前往后添加字符的后缀树构建

Ukkonen 算法维护了 S 的隐式后缀树，并支持在末端插入字符。

称 S 的隐式后缀树为 $STree(S)$ 。

在 $STree(S)$ 中，每个叶结点对应了 S 的一个后缀。称在 $STree(S)$ 中不作为叶结点出现的后缀为 S 的隐式后缀，其余后缀为显式后缀。考虑隐式后缀具有什么样的性质。

引理 5.1: 若 $S[i, n]$ 为 S 的隐式后缀，则对于 $\forall j > i$ ， $S[j, n]$ 也为 S 的隐式后缀。

证明. 由 $S[i, n]$ 为隐式后缀可知，存在字符 c 使得 $S[i, n] + c$ 为 S 的子串，所以 $S[j, n] + c$ 也为 S 的子串，由隐式后缀树的定义可知 $s[j, n]$ 也不作为叶结点出现。证毕。

所以， S 的隐式后缀一定为 S 的后缀中最短的那些。在算法流程中，我们同时维护 S 的最长隐式后缀 $S[k, n]$ 在树中的位置，用二元组 $(active, remain)$ 表示，意为： $S[k, n]$ 在树中所属节点的父亲为 $active$ ，位置为从 $active$ 往下走 $remain$ 步。

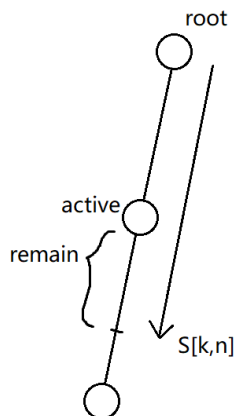


图 3: $(active, remain)$ 示意图

由于我们已经知道剩下的 $remain$ 个字符为 $S[n - remain + 1, n]$ ，所以我们不需刻意记录 $S[k, n]$ 位于 $active$ 的哪一条出边上，只需 $remain$ 即可确定位置。

为了完成本算法，我们还需引入一个概念：后缀链接。

5.1 后缀链接

引理 5.2: 对隐式后缀树中任意非叶非根节点 x ，在树中存在另一非叶节点 y ，使得 mx_y 为 mx_x 删去开头的字符。

证明. 令 str 表示 mx_x 删去开头字符形成的字符串。由隐式后缀树的定义可知，存在两个不同的字符 $c1, c2$ ，满足 $mx_x + c1$ 与 $mx_x + c2$ 均为 S 的子串。所以， $str + c1$ 与 $str + c2$ 也为 S 的子串，所以 str 在后缀 trie 中也对应了一个有分叉的关键点，即在 $STree$ 中存在 y 使得 $mx_y = str$ 。证毕。

由引理 5.2，我们可以对 $STree$ 中的所有非根非叶节点 x ，定义 $Link_x = y$ ， $Link_x$ 称为 x 的后缀链接 (Suffix Link)。

5.2 算法流程

为了构建 $STree(S)$, 我们从前往后加入 S 中的字符。假设当前已经建出 $STree(S[1, m])$ 且维护好了后缀链接。 $S[1, m]$ 的最长隐式后缀为 $S[k, m]$, 在树中的位置为 $(active, remain)$ 。设 $S[m + 1] = x$, 现在我们需要加入字符 x 。

此时, $S[1, m]$ 的每一个后缀都需要在末尾添加字符 x 。由于所有显式后缀都对应树中某个叶结点, 它们只会让叶结点对应父边的长度增加 1, 而不会改变树的形态, 这很方便我们记录。所以, 现在我们只用考虑隐式后缀末尾添加 x 对树的形态产生的影响。

首先考虑 $S[k, m]$, 有两种情况:

1、 $(active, remain)$ 位置已经存在 x 的转移。此时后缀树形态不会发生变化。由于 $S[k, m + 1]$ 已经在后缀树中出现, 所以对于 $j > k$, $S[j, m + 1]$ 也会在后缀树中出现, 此时只需将 $remain + 1$, 不需做任何修改。

2、 $(active, remain)$ 不存在 x 的转移。如果 $(active, remain)$ 恰好为 $STree$ 中的节点, 则此节点新增一条出边 x ; 否则需要对节点进行分裂, 在此位置新增一个节点, 并在新增节点处添加出边 x 。此时对于 $j > k$, 我们并不知道 $S[j, m]$ 会对后缀树形态造成什么影响, 所以我们还需继续考虑 $S[k + 1, m]$ 。考虑怎么求出 $S[k + 1, m]$ 在后缀树中的位置: 如果 $active$ 不为 $root$, 可以利用后缀链接, 令 $active = Link_{active}$; 否则, 令 $remain - 1$ 。最后令 $k + 1$, 再次重复这个过程。

上述过程可以用 `while` 循环来完成, 同时分裂节点时还应注意维护新增节点的后缀链接, 下面给出完整修改过程:

维护最后一次新增出边的节点 $last$, 初始设为 $NULL$ 。在循环过程中保证只有 $last$ 可能还未确定后缀链接。

`while` (还有尚未考虑的隐式后缀)

{

由于在上一次循环中对 $(active, remain)$ 作出了修改, 可能使 $remain$ 超出了 $active$ 的对应出边的长度, 所以需先在 $STree$ 中从 $active$ 往下搜索, 直到 $remain$ 不超出范围为止。由于保证当前隐式后缀在原 $STree$ 中出现过, 这一步很容易实现。

接下来分两类讨论:

1、 $(active, remain)$ 存在 x 的转移。此时, 如果 $last$ 存在, 由引理 5.2, $(active, remain)$ 位置必定恰好为 $STree$ 中某个非叶节点, 将 $Link_{last}$ 设为当前节点并将 $remain + 1$, 即可退出循环。

2、 $(active, remain)$ 不存在 x 的转移。此时, 视当前位置是否恰好在 $STree$ 中, 可能新增一个分裂节点。记此时 $(active, remain)$ 对应的节点为 p 。新增叶子结点 q , 并添加一条从 p 连向 q 的边。显然, 如果 $last$ 存在, 那么 $Link_{last} = p$, 再将 $last$ 设为 p 即可。

最后，如果还没有退出循环：如果 $active$ 不为 $root$ 则将 $active$ 设为 $Link_{active}$ ，否则将 $remain - 1$ 。

}

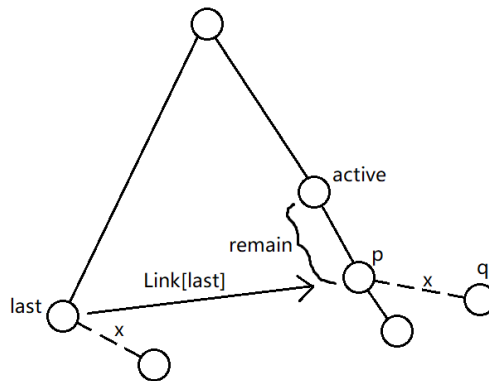


图 4: 过程示意图

至此，我们成功维护出了 $STree$ 形态的变化、所有后缀链接的变化和最长隐式后缀的变化。

5.3 算法时间复杂度证明

除了退出循环时，循环的每次运行都伴随着最长隐式后缀长度的减小，而最长隐式后缀长度只会增加 n 次，所以循环只会运行 $O(n)$ 次。

当 $remain$ 超出 $active$ 对应出边的长度时，我们会在 $STree$ 上往下搜索，但是每次移动都伴随着 $remain$ 的减小，而 $remain$ 总共只会增加 n 次，所以这一移动的时间复杂度也为均摊 $O(n)$ 。

综上所述，算法的时间复杂度为均摊 $O(n)$ 。

5.4 算法总结

由于 Ukkonen 算法只能处理出 S 的隐式后缀树，而隐式后缀树在一些问题中的功能可能不如后缀树强大，所以在需要时，可以在 S 的末端添加一个从未出现过的字符 $\#$ ，这时 S 的所有后缀可以和 $STree$ 的所有叶子一一对应。

Ukkonen 算法主要利用了后缀链接来优化每次寻找隐式后缀的过程，而且与上一节的算法相反，支持在字符串末端动态插入字符，时间复杂度也达到了 $O(n)$ 的下界。下面简单介绍上述算法的一些应用。

6 例题

6.1 例题一：SWERC 2015 Text Processor^[4]

6.1.1 题目大意

给定一个长度为 n 的小写字母字符串 S ，再给定 $1 \leq w \leq n$ ，对于所有 $i \in [1, n - w + 1]$ ，求出 $S[i, i + w - 1]$ 有多少本质不同子串。 $n \leq 10^5$ 。

6.1.2 题目分析

求区间本质不同子串个数是一个经典问题，可以利用后缀自动机和动态树得到时间复杂度为 $O(n \log^2 n)$ 的做法^[5]，实际上本题可以做到线性复杂度。

由于不同子串个数即为后缀 trie 节点数，如果我们对每个区间 $[i, i + w - 1]$ 建出后缀树，后缀树的边长总和即为答案。由于本题区间长度固定，可以考虑采用滑动窗口，每次在末尾添加一个字符，或在开头删掉一个字符，维护后缀树边长总和即可。

考虑利用 Ukkonen 算法维护后缀树。在开头删掉一个字符即为删掉当前深度最深的叶子，可以用队列维护当前所有叶子，快速找到当前需要删除的节点 x 。删去 x 以后，可能会使某个隐式后缀变为显式后缀，也可能使 fa_x 的儿子节点个数减少至一个，需要压缩掉 fa_x 节点。仔细维护当前 (*active, remain*) 的位置和边长总和的变化即可，时间复杂度为 $O(n)$ 。

6.2 例题二：北大集训 2018 Day3 close

6.2.1 题目大意

维护一个字符串 S ，支持双端插入，要求在线维护 S 的本质不同子串个数。

6.2.2 题目分析

令 n 为字符串 S 最后的总长。

出题人给出了利用平衡树维护后缀的时间复杂度为 $O(n \log^2 n)$ 的做法^[6]，实际上本题同样可以优化至 $O(n)$ ^[7]，下面简单介绍做法。

上文已经给出利用 *trans* 转移边和后缀链接 *Link* 维护插入的两种算法，容易想到同时维护 *trans* 和 *Link*，将两种算法结合。

由于末端插入时， S 的隐式后缀对后缀树的形态影响是难以预料的，所以仍然考虑维护 S 的隐式后缀树 $STree(S)$ 。首先需要解决的问题是：在隐式后缀树上，我们仍然可以类似的定义 *trans* 吗？

设 S 的最长隐式后缀为 $S[k, n]$ 。对 S 的一个子串 str ，定义 $leftpos'_{str}$ 表示 $leftpos_{str}$ 在 $[1, k-1]$ 中的位置。类似后缀树，在 $STree$ 中有：节点 x 代表的字符串集合的 $leftpos'$ 相同，可以记为 $leftpos'_x$ ，且不同节点的 $leftpos'$ 集合不相同。 x 代表的所有字符串 str 开头添加字符 c 后的 $leftpos'$ 集合相同的一个充分条件是：所有 str 的 $leftpos$ 集合在 $[1, k]$ 中出现的位置相同。所以， $trans_{x,c}$ 在 $STree$ 中几乎是良定义的，只有一种特殊情况：某些 $leftpos_{str}$ 中出现了位置 k ，而某些不出现位置 k 。这种节点在 $STree$ 中至多只有一个： $(active, remain)$ 所在的节点。

所以，在 $STree$ 中，我们仍然可以类似地定义 $trans$ 转移边：如果 x 代表的所有 str 在开头添加字符 c 后对应同一节点 y ，那么令 $trans_{x,c} = y$ ，否则令 $trans_{x,c} = NULL$ 。只有在 $(active, remain)$ 所处的树边上需要特判：在这条边上，位于 $(active, remain)$ 上方的位置在前端添加字符 $S[k-1]$ 后，会转移到 $S[k-1, n]$ 对应节点。

现在，可以开始考虑算法流程。

6.2.3 算法流程

首先考虑在前端添加字符。沿用 4.2 中的算法流程，注意特判 $(active, remain)$ 位置的转移边即可。如果此时新增了分裂节点 nq ，我们还需维护 $Link_{nq}$ ，显然 $Link_{nq} = p$ 。同时，可能会有一个显式后缀变为隐式后缀，简单讨论即可。

然后考虑在末端插入字符。沿用 5.2 中的算法流程。如果此时新增了节点，我们还需维护新增节点的转移边。

对于新增的叶子结点 q ，显然会有到上一个加入的叶子结点 q' 的转移。

对于新增的分裂节点，可以复制它的孩子的转移。

同时，对于上一个新增的分裂节点 $last$ ，可能会需要将一些转移边修改为它。可以发现，影响的所有节点恰好为从 $Link_{active'}$ 在 $STree$ 中寻找合适的 $active$ 的位置往下搜索时遍历到的所有节点 ($active'$ 为上一次 $active$ 所处的位置)。当 $active'$ 为 $root$ 时可能需要特殊判断。

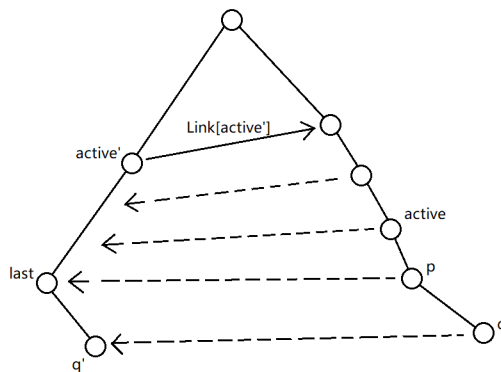


图 5: 虚线为需要修改的转移边

至此，我们完成了整个算法流程。将两种算法的时间复杂度证明结合到一起，可以得到本做法的时间复杂度也为 $O(n)$ 。

7 总结

在本文中，我们主要介绍了后缀树的动态前端插入、末端插入，并在例题中给出了将两种算法结合的支持动态双端插入的做法。

希望本文能激发读者的思考，让读者对后缀树和它的优美性质有更加深刻的理解，并带领读者领略到后缀树的魅力。

8 致谢

感谢中国计算机协会提供交流和学习的平台。

感谢国家级集训队高闻远教练的指导。

感谢广州市第二中学林盛华老师对我的支持与指导。

感谢赵雨扬前辈为本文提供思路。

感谢罗恺同学为本文验稿。

感谢给予过我帮助的老师与同学们。

参考文献

- [1] 范浩强，《后缀自动机与线性构造后缀树》
- [2] 陈立杰，《后缀自动机》，2012 冬令营营员交流。
- [3] E. Ukkonen, Constructing suffix trees on - line in linear time, in Algorithms, Software, Architecture. Information Processing 92, vol. I (J. van Leeuwen, ed.), Elsevier, 1992, pp. 484-492.
- [4] <https://codeforces.com/gym/101128>
- [5] 陈江伦，《后缀树结点数》命题报告及一类区间问题的优化，IOI2018 中国国家候选队论文集
- [6] <https://zhuanlan.zhihu.com/p/51880239>
- [7] <https://negiizhao.blog.uoj.ac/blog/4756>

一类调整算法在信息学竞赛中的应用

中国人民大学附属中学 邓明扬

摘要

调整是一种非完美算法，在许多情形下难以证明复杂度但却有着优越的表现。本文致力于研究一类调整算法在信息学竞赛中的应用，包括求解部分提交答案题，以及在随机数据下求解一些传统图论问题。

1 引言

组合优化题目在信息学竞赛中占了很大比重。一般地，组合优化问题有如下形式：一个问题有一些合法解和不合法解。每个合法解有一个对应的权值。你需要在所有合法解中找出权值最大的一个。

一种显然的做法是：先任取一个合法解，然后对合法解进行微调使得权值变大。一直操作直到无法进行。这一算法看似简单，但在许多问题中有出色的表现。

2 算法描述

本文中的调整算法有如下形式：

对一个组合优化问题，记 S 为所有可能状态的集合。 $\forall x \in S$, 定义 $F(x)$ 为 x 的权值。

$T \subset S$ 为合法状态的集合。你要求出 $x \in T$, 使得 $F(x)$ 取得最大值。¹

在实际问题中，集合 T 往往具有某种局部性质。一般地，对于 $x \in T$, 往往能找到 $N(x) \subset T$, 满足 $\forall y \in N(x)$, y 与 x 的差异较小。我们称 $N(x)$ 为 x 的一个“邻域”。

我们考虑一种显然的搜索算法。首先任取一个 $x \in T$, 一直重复以下过程直到超时：

1. 如果 $N(x)$ 中存在 y , 满足 $F(y) > F(x)$, 我们将 x 修改为 y , 并进入下一轮。
2. 如果不满足 1 中的条件, 但存在一些 $y \in N(x)$, 满足 $F(x) = F(y)$, 我们从中随机选取一个 y , 将 x 修改为 y 并进入下一轮。

我们称这一算法为调整法。容易发现，每一轮我们在当前解的局部邻域内寻找一个较优解并更新。最终我们将求出一个局部较优解。

¹本文中，如无特殊说明，默认最大值为最大值

这一简单的算法，在设计得当的情况下，有着优越的表现。我们将用许多具体的例子说明这一点。

3 匹配问题

匹配问题时常可在信息学竞赛中见到。对于经典的问题，例如一般图/二分图最大匹配，已经有了多项式时间的传统算法。然而调整法给出了一个在随机数据实践中表现优越的简洁解法。

除此之外，一些不常规的隐式匹配问题往往是 NPC 的，或是需要特殊性质下的特殊解法。而调整法在这些问题中仍然有突出的表现。

3.1 一般图最大匹配

一般图最大匹配是图论中的经典问题。利用带花树或 Tutte 矩阵，可以在 $O(n^6)$ 或 $O(n^3)$ 复杂度内求解。下面我们介绍一种在许多数据下运行表现良好的调整算法。

3.1.1 问题描述

一般图最大匹配指这样一个问题：

给定图 $G = (V, E)$ ，其中 V 是顶点集， E 是边集。求出最大的子集 $S \subset E$ ，满足 $\forall v \in V$ ， S 中以 v 为端点的边至多只有一条。

3.1.2 调整算法

以下是一种调整算法：

对于图 $G = (V, E)$ ，我们维护一个子集 $S \subset E$ ，表示当前匹配的边。每次随机选取 V 中一个未匹配的顶点 v 。如果 v 存在未匹配邻点，从中随机选取一个，将它与 v 的连边加入 S 。否则随机选取 V 的一个邻点 u ，将 u 和 v 匹配并将 u 原先匹配的边断开。

一直重复直到超时。

在这一过程中，我们一直在对 S 进行局部微调，其大小不减且有可能增大。这符合我们之前对调整算法的描述。

3.1.3 算法运行表现

这一算法十分简洁，但最坏复杂度不是多项式级的。然而，在针对这一算法的数据出现前，该算法能通过评测网站 uoj.ac 上一般图最大匹配的全部测试点。在随机图中，经过实验，我们发现该算法均在较少的轮次中就找到了最大匹配。

笔者测试了算法在 $|V| = 500$ 的随机图下的表现。具体随机方式为：先生成 $|E|$ ，然后在所有 $|V|$ 个点， $|E|$ 条边的图中等概率选取一个。

经过实验，当 $|E|$ 在 700 到 800 的范围内随机时所需的期望轮次较多。笔者在此范围内随机了 5000 张图，算法所需的最多轮次数为 1114612，平均轮次数约为 13367。

我们可以看出，该算法在随机数据下表现良好。

然而，对于一些特殊图，算法并不能在短时间内求解。例如，对于一张二分图 $V = (A_i, B_j | 1 \leq i \leq n, 1 \leq j \leq n+1)$ ，当 $n = 40$ 时算法需要很久才能求出完美匹配。

3.1.4 算法分析

算法在最坏数据下的期望运行时间是指数级的。

这里给出算法期望复杂度 $O(n^{n/2+4})$ 的证明。

考虑一个最大匹配 T 。设当前匹配为 S 。不妨当前匹配不是最大匹配。

我们考虑新建一个图 $G_1 = (V, E_1)$ ，一条边在 G_1 中当且仅当这条边出现在 T 和 S 恰好之一中。（即： G_1 为当前匹配与最大匹配的对称差。）对于 G_1 中的一条边，如果它在 T 中则将该边染为蓝色，如果它在 S 中则将该边染为红色。

由于每个点在 T 中和 S 中的度数均不超过 1，每个点在 G_1 中的度数均不超过 2。因此， G_1 由一些环和链构成。由于任何相邻两条边不同色， G_1 中没有奇环；又因为蓝色边比红色边多，因此存在一条链首尾均是蓝色，链上红蓝交替（用到相邻两条边不同色）。

此时任取一条这种链，该链上除了首尾点外均已匹配。期望至多 $O(n)$ 轮后，调整算法随机取点会取到这条链的链首或链尾。此时如果匹配了该点连接的红边，这条链的长度将减小 2。否则链长增加至多 1。由于长度减少的概率至少为 $1/n$ ，在期望 $O(n^{n/2+1})$ 次选取首尾之后长度会有连续的 $n/2$ 次减少，此时匹配数增加了 1。由于最大匹配的大小是 $O(n)$ 的，因此期望 $O(n^{n/2+1} * n * n)$ 轮之后可以得到最大匹配。由于每轮花费 $O(n)$ 的时间，算法的期望复杂度不超过 $O(n^{n/2+4})$ 。

这一理论上界非常巨大，但算法在极限数据下的运行速度的确很慢。但该算法便于实现，且在许多随机数据及非精心构造的数据下表现优越，因此很适用于解决现实生活中遇到的问题，以及信息学竞赛中的提交答案题或在一些特定模型下、数据有特殊性质的传统题。

对于一般图最大匹配，该算法的正确性有保证。我们也需要说明，后文中的许多调整算法笔者并不能证明其正确性，但同样在现实生活/提交答案题中有出色的表现。

3.2 隐式匹配问题

在信息学竞赛中，另有一大类问题与匹配相关，但常常无法转化成常规的一般图最大匹配。我们称其为隐式匹配问题。对于这类问题，前文提及的调整算法常常奏效。尽管复杂

度并没有严格的证明，但往往有出色的实际表现。后文将列举两道隐式匹配例题。

3.3 CF Round 562 E

3.3.1 问题描述

给定序列 x_i ($0 \leq i < 2^n$)，请你构造两个 0 到 $2^n - 1$ 的排列 p_i, q_i ($0 \leq i < 2^n$)，满足 $\forall 0 \leq i < 2^n$ ，有 $p_i \oplus q_i = x_i$ 或输出无解。

$$n \leq 12.$$

3.3.2 调整算法

容易发现，当 $\bigoplus_{0 \leq i < 2^n} x_i$ 不为 0 时，原问题无解。下面考虑 $\bigoplus_{0 \leq i < 2^n} x_i = 0$ 的情形。可以证明此时原问题均是有解的，证明参见参考文献 [2].

考察这样一种调整算法。

我们试图将 0 到 $2^n - 1$ 与 x_0 到 x_{2^n} 匹配，使得每一对匹配数的异或和不同。此时 x_0 到 x_{2^n} 所匹配的数，及其所处对子的异或和，构成了符合要求的两个排列。

我们参考最大匹配的调整做法，采取类似的解法：每次随机一个 0 到 $2^n - 1$ 中未匹配的数，如果与 x_0 到 x_{2^n} 中某个未匹配数匹配后与当前异或和均不同则将其匹配，否则随机匹配 x_0 到 x_{2^n} 中一个未匹配数，并将其异或和原先所处的对子断开。

3.3.3 算法运行表现

在比赛中，该算法可以在 265ms（约 1/5 时限）内通过原题，从表现上有接近 $O(n^2)$ 的运行效率。碍于水平所限，笔者无法证明其正确性、复杂度的下界也无法构造数据使其答案错误或超时。笔者猜想该算法在这一数据范围内无法卡掉，希望有意愿的同学能在未来给出证明。

3.4 JOI2020 制作团子

3.4.1 问题描述

这是一道提交答案题。

你是制作团子的专家。你现在有若干个团子和竹签，团子被整体摆放在一个 R 行 C 列的格子里，每个格子恰好有一个团子。团子颜色为粉色 (P), 白色 (W), 绿色 (G)。

你每次会选择三个连续的团子，这三个团子必须沿着竖直方向 (从上往下)，水平方向 (从左往右) 或者对角线方向 (从左上至右下，或从右上至左下)。例如，如果你选择了竖直方

向的三个团子，你会按照上-中-下的顺序依次将团子串到竹签上。一个团子只能被串在一根竹签上。

一串团子是漂亮的当且仅当竹签上串的团子的颜色依次为粉-白-绿或者绿-白-粉。

你想要制作尽量多的漂亮的团子。

其中最大的测试点大小约为 $500 * 500$ 。

3.4.2 调整算法

我们仍然考虑调整算法。每次我们随机考虑一个未匹配的 W 色点，并考虑以其为中心的竹签。如果存在一个竹签上不包含匹配点，我们在这样的竹签中随机选择一个添加。否则枚举以它为中心、且只与一根已有竹签冲突的竹签，我们有一半概率用新竹签替换原来的。

3.4.3 算法运行表现

该算法可以在十分钟左右获得 95 分，在一小时左右得到 99 到 100 分。鉴于其非常容易实现，这样的得分效率可以说很高。

4 NP 问题

在许多 NP 问题中，调整算法能给出优秀的解。下文将介绍图染色和有向图哈密顿链的调整解法。

4.1 图染色

4.1.1 问题描述

图染色问题是经典的 NP 问题。

给定一张图 $G = (V, E)$ ，让你构造一种方案将顶点用 k 种颜色染色，满足每条边的两端点不同色。

4.1.2 调整算法

我们需要最小化两 endpoint 同色的边的数目。当这一数目被减少到 0，我们便得到了合法的染色方案。

考察这一种调整算法：首先为每个点随机染一种颜色。之后每次随机一个点，考虑固定其余点的颜色不变，该点染不同颜色对同色边数量的贡献，在贡献最少的颜色中等概率选取一个作为该点的颜色。

4.1.3 算法运行表现

虽然得到染色方案很难，但一个染色是否合法是容易判定的，且生成一个可以 K 染色的数据并不困难。因此我们可以在随机生成数据下考虑调整算法的表现。

我们采取以下方式生成可三染色的图：首先固定 $|V|, |E|$ 。然后对于每个顶点，将其随机染成三种颜色之一，并随机生成 $|E|$ 条两端点不同色的边。

笔者生成了 120 张 $|V| = 500$ ， $|E|$ 在 0 到 45000 内随机的三染色图。这些图中，算法在 118 张上得到了正确的结果，调整的最大轮次数为 98587，平均轮次数为 5904.75。

4.1.4 算法分析

该算法的正确性并没有保证：因为存在一个染色方案，使得局部无法调整，但是全局并非最优。例如一个四元环，四个点颜色分别为黑黑白白，每个环上黑点外接一个白点、每个环上白点外接一个黑点。此时该染色无法调整，但是原图存在合法的二染色。

然而算法在随机数据下表现出色，因此可以常常应用于提交答案题或实际问题中。

4.2 有向图哈密顿链

4.2.1 问题描述

有向图哈密顿链也是经典的 NPC 问题。

给定一张有向图 $G = (V, E)$ ，请你构造一条经过每个点恰好一次的路径（起点终点不给定）。

4.2.2 调整算法

维护边的一个尽量大的子集，满足只考虑这些边时每个点出入度都不超过 1，且不构成圈。

如果子集大小达到 $n - 1$ ，则找到了一条哈密顿路。

考虑调整维护子集。按随机顺序考虑边，如果加入后不构成圈，且加入之后所有点度数均仍合法，则加入这条边。

否则如果不构成圈，但有一个点度数不合法，则以一半概率加入并把该点相连的与新加入边矛盾的边断掉。

使用 LCT 判断是否成圈。

4.2.3 算法运行表现

笔者采用以下方法生成了大量的图进行试验：

首先加入边 $(i, i + 1)$ ，其中 $1 \leq i \leq n - 1$ ，以保证存在哈密顿链。然后随机加入若干有向边，并将顶点重标号。

笔者按以上方法随机生成了 20 张 $|V| = 50000$ ， $|E|$ 在 $|V| - 1$ 到 $|V| + 200000$ 间等概率随机的图。其中有 1 张图该算法未能求出哈密顿链；对于其余的图，算法最多考虑的边数为 4170064，平均约为 2000000。换言之，在实验数据中百分之 95 的随机图均能在几秒内求解。

4.2.4 算法分析

该算法不一定是正确的。因为有可能加入了一些边后，局部无法调整，但是全局并非最优。例如：对于 5 个点的有向图，边集为 $\{(4, 1), (1, 3), (5, 2), (4, 2), (5, 1), (3, 4)\}$ ， $(5, 1, 3, 4, 2)$ 为一个合法的哈密顿链。然而，当选择了边集 $\{(4, 1), (1, 3), (5, 2)\}$ 后，无法进行任何调整。这也是实验部分一些图未能求解的原因。但此时改变随机顺序再次求解，往往能求出答案。

4.2.5 一些规约

事实上，给定起点终点的哈密顿路径问题及哈密顿圈问题均可归约到这一问题。

如果给定起点终点，只需新建两个点 U, V ；然后 U 到起点连一条边，终点到 V 连一条边

如果要求哈密顿圈，可以枚举一条边，转化成给定起点终点情形。

如果是无向图，只需正反边各加一遍即可。

4.3 NP 问题的近似解

调整法也可以用于求解许多 NP 问题的近似解。一般地，许多 NP 问题可以归约到每个变量取值在 $\{0, 1\}$ 中的线性规划。对于这种线性规划问题，常常有如下的近似算法：

首先忽略变量取值在 $\{0, 1\}$ 中的条件，弱化为变量取值在 $[0, 1]$ 中。弱化版本是一个正常的线性规划问题，可以用弱多项式的内点法求解。此时得到了一个解，但每个变量的取值是分数。我们想将这组解转化成一组整数解，并保持原先尽量多的性质。

转化成整数解的方法通常有以下两种：一种是对一个分数解中取值为 x 的变量，有 x 的概率将其转化为 1， $1 - x$ 的概率将其转化为 0。这一算法保证了各线性组合的期望。另一种是调整法，固定一些限制不变的情况下，调整剩余变量的取值直到剩余变量中有一些变为 1 或 0。这一算法能得到较优的近似比。算法中固定的限制往往是：在余下变量任意取的情形下，可能超额较多的限制。由于剩余变量越少，可能超额较多的限制也越少，通常可以选取近似比，使得固定的限制数小于变量数。此时调整总能进行。

这一类调整算法以及调整和概率方法的结合可以参见参考文献 [1]。

5 算法前景

5.1 实际应用

调整算法具有广泛的适用性，在许多题目中均可应用，并且能用于解决许多困难的问题。信息学竞赛中，用调整法解提交答案题往往能在短时间内取得大量分数；另外，对于部分传统题目，调整法也能取得不错的效果。

5.2 一些可能的优化

本文中提到的调整方法较为朴素简洁、容易实现。在此基础上，有一些可能的优化。

5.2.1 局部多次调整

由于目前的随机化算法每次在全局随机取一个变量调整，有可能一个刚被调整过的变量很快又被调整回去，加大了时间开销。因此一种可能的优化方式是：在一个调整过后将其固定，并在其附近寻找其影响到的位置再进行调整，直到该轮调整次数达到某一阈值后，将所有固定的变量改回不固定并开始下一轮调整。

5.2.2 调整与模拟退火

模拟退火是一类基于概率的调整算法，与文中提及调整算法的区别在于：就算解变的更劣，模拟退火也以一定概率接受新解。这样的算法不保证调整的单调性，但是扩大了调整的状态空间，因此也是一个可能的优化方向。

参考文献

- [1] Nikhil Bansal, On a generalization of iterated and randomized rounding. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1125–1135, 2019.
- [2] CodeForces Round 562 Editorial

再探线性规划对偶在信息学竞赛中的应用

南京外国语学校 丁晓漫

摘要

线性规划 (Linear Programming) 是数学规划的一个重要分支, 常用于解决各种最优化问题, 许多信息学竞赛中的模型均能用线性规划表示。直接求解线性规划复杂度一般较高, 而利用线性规划对偶原理对问题进行转化后能以更优秀的复杂度求解问题。本文介绍了线性规划对偶以及拉格朗日对偶, 并对二者的应用进行介绍和总结。

1 引言

在信息学竞赛中, 题目中给出线性的约束和线性的目标函数时, 就可以得到线性规划的模型。有些线性规划可以轻松转化成最短路、网络流等容易求解的问题, 但另一部分线性规划较难直接求解。此时使用线性规划对偶进行转化是另一个值得尝试的思路。

早在 2013 年, 正式 OI 比赛中就已经出现了利用线性规划对偶求解的问题¹。实际上, 2016 年的集训队论文中已经出现了线性规划以及对偶算法的介绍²。而在 2020 年, 浙江省选中又出现了可以运用这个算法的题目³, 且在近些年, 线性规划对偶在各种线上比赛中出现的频率也有所提升, 类型更加多样, 故作者认为有必要针对对偶算法再作一个整理和总结。

在第二节中, 作者简单介绍了线性规划模型和一般线性规划的通用解法。

在第三节中, 作者介绍了线性规划对偶、对偶原理和拉格朗日对偶。

在第四节中, 作者以例题为例, 详细介绍了第三节中算法的应用。

¹[ZJOI2013] 防守战线

²董克凡, 《浅谈线性规划与对偶问题》, 2016 年信息学奥林匹克中国国家集训队论文集

³[ZJOI2020] 序列

2 基本介绍

2.1 定义

对于一组变量 x_1, x_2, \dots, x_n 来说, 不等式 $\sum_{i=1}^n a_i x_i \leq, =$ 或 $\geq b$ 被称为一个线性约束, 而 $\sum_{i=1}^n c_i x_i$ 则是关于这组变量的线性目标函数。

规划问题可分为线性规划、整数规划、非线性规划、动态规划等, 而线性规划是在一组线性约束条件下, 求一线性目标函数最大或最小的问题。

2.2 标准型

为了方便表达一般的线性规划, 我们作出规定:

- 如果原模型中需要最小化 $\sum_{i=1}^n c_i x_i$, 则令 $c'_i = -c_i$, 目标即变成最大化 $\sum_{i=1}^n c'_i x_i$ 。
- 如果某条约束要求 $\sum_{j=1}^n a_{ij} x_j \geq b_i$, 同样令 $a'_{ij} = -a_{ij}$, 约束即变为 $\leq -b_i$; 如果某条约束要求 $\sum_{j=1}^n a_{ij} x_j = b_i$, 拆成 $\leq b_i$ 和 $\geq b_i$ 即可。
- 如果对于某个变量 x_i 没有约束, 引入两个变量 x' 和 x'' , 令 $x', x'' \geq 0$ 且 $x = x' - x''$, 容易验证和原模型等价。

经过上述转换, 任意模型就转换成了标准型, 可写成

$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{x} \\ \mathbf{A} \mathbf{x} \leq & \mathbf{b} \\ \mathbf{x} \geq & \mathbf{0} \end{aligned} \tag{1}$$

此处 \mathbf{c} 为长度为 n 的向量表示目标函数的系数, \mathbf{x} 为长度为 n 的向量表示变量, \mathbf{A} 为 $m \times n$ 的矩阵表示约束的系数, \mathbf{b} 为长度为 m 的向量表示约束中的常数。⁴

2.3 解法

目前为止, 线性规划问题还没有找到特别高效的算法。

多项式复杂度的算法有椭球算法和内点法等, 非多项式复杂度的算法有单纯形法⁵。由于后者实现难度较低, 且实际应用中远远达不到理论复杂度上限, 信息学竞赛中一般使用单纯形法求解。

⁴称两个长度都为 n 的向量 $\mathbf{x} \leq \mathbf{y}$ 当且仅当 $\forall_i x_i \leq y_i$

⁵可参考 Spyros Reveliotis of the Georgia Institute of Technology, An Introduction to Linear Programming and the Simplex Algorithm

3 线性规划对偶

3.1 标准型的对偶

由于任意线性规划都可以被转化成标准型，下面直接给出(1)的对偶，为

$$\begin{aligned} \min \mathbf{y}^T \mathbf{b} \\ \mathbf{A}^T \mathbf{y} \geq \mathbf{c} \\ \mathbf{y} \geq \mathbf{0} \end{aligned} \quad (2)$$

此处 \mathbf{y} 称作对偶变量。

同时，(1)也是(2)的对偶，故称二者互为对偶。

3.2 对偶定理

对偶定理即是线性规划对偶满足的性质。

对偶的基本思想是不断寻找 $\mathbf{c}^T \mathbf{x}$ 的上界。考虑任意长度为 m 的向量 \mathbf{y} 满足

$$\mathbf{A}^T \mathbf{y} \geq \mathbf{c}$$

，那么有

$$\begin{aligned} \mathbf{y}^T \mathbf{A} \geq \mathbf{c}^T \\ \mathbf{y}^T \mathbf{b} = \mathbf{y}^T \mathbf{A} \mathbf{x} \geq \mathbf{c}^T \mathbf{x} \end{aligned}$$

感性理解， y_i 是第 i 个线性约束的权重，这 m 个线性约束加权求和后每个变量的系数都 \geq 目标函数中的系数，那么目标函数的值肯定 \leq 加权后的约束值。

弱对偶定理。即是说，在(1)和(2)的定义下，有

$$\max \mathbf{c}^T \mathbf{x} \leq \min \mathbf{y}^T \mathbf{b}$$

强对偶定理。强对偶定理告诉我们，进一步地有

$$\max \mathbf{c}^T \mathbf{x} = \min \mathbf{y}^T \mathbf{b} \quad (3)$$

6

⁶强对偶定理的证明较为困难，可参考 Gärtner, Bernd; Matoušek, Jiří (2006). Understanding and Using Linear Programming. 在此略去。

3.3 拉格朗日对偶

3.3.1 一般的拉格朗日对偶

假设 $f(x), g(x)$ 均为关于 x 的函数, x 的定义域为 P 。此时要求

$$\begin{aligned} \max f(x) \\ g(x) \leq 0 \end{aligned} \quad (4)$$

我们引入拉格朗日乘子 λ , 记 $L(\lambda) = \max f(x) - \lambda g(x)$, 有

$$(4) \leq \min L(\lambda) \quad (5)$$

(注意此处的 x 和 λ 不一定是一个数, 也可能是向量)。

拉格朗日对偶的性质 考虑

$$\begin{aligned} &L(a\lambda_1 + (1-a)\lambda_2) \\ &= f(x^*) - (a\lambda_1 + (1-a)\lambda_2)g(x^*) \\ &= a(f(x^*) - \lambda_1 g(x^*)) + (1-a)(f(x^*) - \lambda_2 g(x^*)) \\ &\leq aL(\lambda_1) + (1-a)L(\lambda_2) \end{aligned}$$

也就是说 $L(\lambda)$ 关于 λ 是凸的, 可以通过二分斜率或三分求得最小值。

3.3.2 线性规划中的拉格朗日对偶

在线性规划中, 还是考虑(1), 令 $f(x) = \mathbf{c}^T \mathbf{x}$, $g(x) = \mathbf{A}\mathbf{x} - \mathbf{b}$, $\lambda = \mathbf{y}^T$ 。那么(5)即变成

$$\max \mathbf{c}^T \mathbf{x} \leq \min_{\mathbf{y} \geq 0} \max_{\mathbf{x} \geq 0} \mathbf{c}\mathbf{x} - \mathbf{y}^T (\mathbf{A}\mathbf{x} - \mathbf{b}) = \min_{\mathbf{y} \geq 0} \max_{\mathbf{x} \geq 0} (\mathbf{c} - \mathbf{y}^T \mathbf{A})\mathbf{x} + \mathbf{y}^T \mathbf{b} = (2) \quad (6)$$

上式中的最后一步是因为 \mathbf{x} 的系数为 $\mathbf{c} - \mathbf{y}^T \mathbf{A}$, 若 $\mathbf{c} - \mathbf{y}^T \mathbf{A} > \mathbf{0}$ 则(6)能取到 ∞ , 不符合 \min 的要求。

在线性规划条件下, 拉格朗日对偶和线性规划对偶本质相同, 但形式不同。如遇到拉格朗日对偶的 $\min \max$ 形式, 可以考虑转化成线性规划问题求解。

4 在信息学竞赛中的应用

4.1 转化成最小费用流模型

4.1.1 建立模型

对于边 $uv \in E$, 令变量 f_{uv} 为边 uv 的流量, 常量 c_{uv} 为流量限制, w_{uv} 为单位流量的代价, b_u 为 u 点的流量需求 (即要求流出的流量减去流进的流量为 b_u), 写成线性规划就是

$$\begin{aligned} \min & \sum_{uv} w_{uv} f_{uv} \\ & -f_{uv} \geq -c_{uv} \\ & \sum_v f_{vu} - \sum_v f_{uv} = -b_u \end{aligned} \quad (7)$$

令 z_{uv} 为 $f_{uv} \leq c_{uv}$ 的对偶变量, p_u 为 $\sum_v f_{vu} - \sum_v f_{uv}$ 的对偶变量, 写成对偶形式就是

$$\begin{aligned} \max & \sum_u -b_u p_u - \sum_{uv} c_{uv} z_{uv} \\ & p_v - p_u - z_{uv} \leq w_{uv} \end{aligned}$$

再整理一下并把 z_{uv} 消去, 可得

$$\min \sum_u b_u p_u + \sum_{uv} c_{uv} \max(0, p_v - p_u - w_{uv}) \quad (8)$$

这也告诉我们, 如果题目所求为 (8) 的形式, 就可以用最小费用流解决。

4.1.2 整数性的探讨

在实际应用中, 转化成 (8) 的形式之后, 有时题目会要求所有 p_u 都为整数。观察 (8) 的形式, 实际上有一个很强的结论:

关于整数性的引理. 在(8)中, 如果满足所有 w_{uv} 均为整数, 那么一定可以在所有 p_u 都为整数时取到最优解。也就是说, 每条边的单位流量代价都为整数的费用流的对偶一定有整数最优解。

证明. 考虑对于最优解所有 p_u 的本质不同的非零小数部分个数 k 进行归纳。

如果 $k = 0$, 结论成立;

如果 $k > 0$, 任选一个非零小数部分记为 x , 记 S 为所有满足 p_u 的小数部分为 x 的 u 的集合, 这里显然有 $|S| > 0$ 。考虑计算令所有 $u \in S$ 的 p_u 都 $+\epsilon$ (ϵ 为极小量) 对式子贡献的变化以及令所有 $u \in S$ 的 p_u 都 $-\epsilon$ 对式子贡献的变化之和。对于 $\sum_u b_u p_u$ 来说, 贡献变化之和显然为 $\epsilon b_u + (-\epsilon b_u) = 0$; 对于 $\sum_{uv} c_{uv} \max(0, p_v - p_u - w_{uv})$ 来说, 需要分类讨论:

- 如果 $u, v \in S$ ，显然两种情况下 $p_v - p_u$ 都不变，贡献变化之和为 0
- 如果 $u, v \notin S$ ，同样两种情况下 $p_v - p_u$ 都不变，贡献变化之和为 0
- 如果 $u \in S, v \notin S$ 或 $v \in S, u \notin S$ ，此时有 p_u 的小数部分 $\neq p_v$ 的小数部分。又有 w_{uv} 是整数，那么 $p_v - p_u - w_{uv} \neq 0$ 。如果 $p_v - p_u - w_{uv} < 0$ ，两种情况下变化后仍有 < 0 ，贡献变化之和为 0；如果 $p_v - p_u - w_{uv} > 0$ ，两种情况贡献变化之和为 $\epsilon c_{uv} + (-\epsilon c_{uv}) = 0$

综上，向两个方向变化的贡献变化之和 = 0，那么向某一个方向变化贡献一定不会变劣。不断向那个方向变化（ x 也向那个方向变化），直到 S 发生变化或者 x 变为整数。这两种情况下， k 的大小都会减小。根据归纳，结论成立。□

由于上述引理，在本节中可以略去对于整数性的讨论。

4.1.3 例：[ZJOI2013] 防守战线

题目描述 战线可以看作一个长度为 n 的序列，现在需要在这个序列上建塔来防守敌兵，在序列第 i 号位置上建一座塔有 C_i 的花费，且一个位置可以建任意多的塔，费用累加计算。有 m 个区间 $[L_1, R_1], [L_2, R_2], \dots, [L_m, R_m]$ ，在第 i 个区间的范围内要建至少 D_i 座塔。求最少花费。

数据范围 $n \leq 1000, m \leq 10000$

解题思路 令 p_i 为前 i 个位置建的塔总数，把问题用线性规划描述即为

$$\min p_i(C_i - C_{i+1})$$

$$p_{i+1} - p_i \geq 0$$

$$p_{R_i} - p_{L_i-1} \geq D_i$$

此时，可以将目标函数写成

$$\sum_v p_v(C_v - C_{v+1}) + \sum_v \infty \max(0, p_v - p_{v+1}) + \sum_i \infty \max(0, p_{L_i-1} - p_{R_i} + D_i)$$

，变成 (8) 中的形式。

4.1.4 例：[Aizu 2230] How to Create a Good Game

题目描述 给定 n 个点 m 条边的有向无环带权图。可以增长一些边的边权，使得点 0 和点 $n-1$ 的最长路不变。求出最多能增长多少边权。

数据范围 $n \leq 100, m \leq 1000$

解题思路 令 p_v 为点 0 到点 v 的最长路长度。对于边 uv ，记 w_{uv} 为原本的边权， x_{uv} 为边 uv 增长的边权， D 为原图上 0 到 $n-1$ 的最长路长度，把问题用线性规划描述即为：

$$\begin{aligned} \max \quad & \sum_{uv} x_{uv} \\ p_{n-1} - p_0 & \leq D \\ p_v - p_u & \geq w_{uv} + x_{uv} \\ x_{uv} & \geq 0 \end{aligned}$$

目标函数即为 $\min \sum_{uv} -x_{uv}$ 。又有 $-x_{uv} \geq p_u - p_v + w_{uv}$ ，故可以消去 x_{uv} ，将目标函数写成

$$\min \sum_{uv} \infty \max(0, p_u - p_v + w_{uv}) + p_u - p_v + w_{uv}$$

，变成 (8) 中的形式。

4.1.5 例：[Utpc2012.12] じょうしょうツリー

题目描述 给定 n 个节点的有根树，每个点有一个权值 c_i 。每次操作可以选一个给权值 $+1$ 或 -1 ，需要使用最少的次数使得对于任意 u, v 满足 u 是 v 的父亲都有 $c_u \geq c_v$ 。

数据范围 $n \leq 100000$

解题思路 令 p_v 为最终 v 的点权，把问题用线性规划描述即为：

$$\begin{aligned} \min \quad & \sum_v |p_v - c_v| \\ p_u & \geq p_v \end{aligned}$$

引入新变量 $p_0 = 0$ ，那么目标函数可以写成

$$\sum_v \max(0, p_v - p_0 - c_v) + \max(0, p_0 - p_v + c_v) + \sum_{uv} \infty \max(0, p_v - p_u)$$

，变成 (8) 中的形式。⁷

4.2 转化成动态规划模型

4.2.1 例：[XX Open Cup. GP of Moscow] Circles

题目描述 给定一个长度为 n 的非负整数序列 s_1, s_2, \dots, s_n ，称长度为 n 的非负序列（不一定是整数） x_1, x_2, \dots, x_n 是平衡的当且仅当对于任意 i 都有 $x_i + x_{i \bmod n+1} \leq s_i$ 。定义 $f(s_1, s_2, \dots, s_n)$

⁷直接最小费用流仍旧无法通过本题，需要根据费用流的特殊结构使用数据结构模拟费用流。由于和线性规划对偶无关，具体细节在此略去。

为所有平衡的序列的中 $x_1 + x_2 + \dots + x_n$ 的最大值。给定长度为 n 的非负整数序列 a_1, a_2, \dots, a_n 。对于所有 $3 \leq k \leq n$ ，求出 $f(a_1, a_2, \dots, a_k)$ 。

数据范围 $3 \leq n \leq 100000, 0 \leq a_i \leq 100000$ 。

解题思路 首先考虑如何计算 $f(s_1, s_2, \dots, s_n)$ ，写出线性规划：

$$\begin{aligned} \max \quad & \sum_i x_i \\ x_i + x_{i \bmod n+1} & \leq s_i \\ x_i & \geq 0 \end{aligned}$$

令 y_i 为 $x_i + x_{i \bmod n+1} \leq s_i$ 的对偶变量，那么对偶为：

$$\begin{aligned} \min \quad & \sum_i y_i s_i \\ y_i + y_{i \bmod n+1} & \geq 1 \\ y_i & \geq 0 \end{aligned}$$

发现对偶之后 y_i 的取值范围被限制在 $[0, 1]$ 。如果能证明 y_i 的整数性，问题会简单许多。实际上，除了 $y_1 = y_2 = \dots = y_n = 0.5$ 的情况外， y_i 一定为整数。

整数性的证明. 在环上，如果 $y_i + y_{i+1} < 1$ 就在 i 和 $i+1$ 之间分段，那么每一段一定都是 $a, 1-a, a, 1-a$ 交替的。考虑对 $0 < a < 1$ 的段数 k 进行归纳。

如果 $k = 0$ ，说明全部为 $0/1$ ，结论成立；

如果 $k = 1$ 且 n 为奇数，只可能是 $y_1 = y_2 = \dots = y_n = 0.5$ 的情况，结论成立；

如果 $k > 1$ 或 $k = 1$ 且 n 为偶数，任取一段，假设是 $a, 1-a, a, 1-a, \dots$ 交替的，如果段内奇数位置 s_i 之和 \leq 偶数位置 s_i 之和，令 a 加上 ϵ 贡献不变劣；否则令 a 减去 ϵ 贡献不变劣。如此不断调整，直到 $a = 0/1$ 或者和另外一段合并，这两种情况下 k 都会减小，根据归纳结论成立。 \square

有了上述结论，令 $dp[i][0/1][0/1]$ 表示 $y_1 = 0/1, y_i = 0/1$ 时的最小贡献，最后和全部为 0.5 取 \min 即可在线性时间内通过本题。

4.2.2 例：[ZJOI2020] 序列

题目描述 有一个长度为 n 的非负整数序列 a_1, a_2, \dots, a_n 。每一步你可以从以下三种操作中选择一种执行：

- 选择一个区间 $[l, r]$ ，将下标在这个区间里的所有数都减 1。

- 选择一个区间 $[l, r]$, 将下标在这个区间里且下标为奇数的所有数都减 1。
- 选择一个区间 $[l, r]$, 将下标在这个区间里且下标为偶数的所有数都减 1。

求最少需要多少步才能将序列中的所有数都变成 0。

数据范围 多组测试数据, 组数 $T \leq 10$, $n \leq 100000$ 。

解题思路 如果只有第一种操作, 容易发现答案为 $\sum_i \max(0, a_i - a_{i-1})$ 。那么记 x_i 为第 i 个数被第一种操作覆盖的次数, 就可以写出线性规划:

$$\begin{aligned} \min & \left(\sum_i \max(0, x_i - x_{i-1}) + \max(0, a_i - a_{i-2} - x_i + x_{i-2}) \right) \\ & x_i \leq a_i \\ & x_i \geq 0 \end{aligned} \tag{9}$$

(此处规定对于 $i \leq 0$, 有 $a_i = x_i = 0$ 。)

发现目标函数是最小费用流的形式, 故整数性得到证明。

上述线性规划目标函数和 0 取 \max 比较难处理, 考虑引入新变量 y_i 和 z_i , 有

$$\begin{aligned} \min & \left(\sum_i y_i + z_i \right) \\ & -x_i \geq -a_i \\ & y_i - x_i + x_{i-1} \geq 0 \\ & z_i + x_i - x_{i-2} \geq a_i - a_{i-2} \\ & x_i, y_i, z_i \geq 0 \end{aligned}$$

令 X_i 为 $-x_i \geq -a_i$ 的对偶变量, Y_i 为 $y_i - x_i + x_{i-1} \geq 0$ 的对偶变量, Z_i 为 $z_i + x_i - x_{i-2} \geq a_i - a_{i-2}$ 的对偶变量, 写出对偶:

$$\begin{aligned} \max & \left(\sum_i -a_i X_i - (a_i - a_{i-2}) Z_i \right) \\ & Y_i \leq 1 \\ & Z_i \leq 1 \\ & -X_i - Y_i + Y_{i+1} + Z_i - Z_{i+2} \leq 0 \\ & X_i, Y_i, Z_i \geq 0 \end{aligned}$$

将 X_i 消去, 得

$$\begin{aligned} \max & \left(\sum_i -a_i \max(0, -Y_i + Y_{i+1} + Z_i - Z_{i+2}) - (a_i - a_{i-2}) Z_i \right) \\ & Y_i \leq 1 \\ & Z_i \leq 1 \\ & Y_i, Z_i \geq 0 \end{aligned} \tag{10}$$

由于系数是整数的费用流的对偶也存在整数最优解，说明只需要考虑 $Y_i, Z_i \in \{0, 1\}$ 。

那么此时有 $Y_i, Z_i \in \{0, 1\}$ ， $dp[i][0/1][0/1][0/1]$ 表示考虑到第 i 个数， $Y_{i-1} = 0/1$ ， $Z_{i-2} = 0/1$ ， $Z_{i-1} = 0/1$ 的最大贡献，转移枚举 Y_i 和 Z_i 即可在线性时间内通过本题。

4.3 拉格朗日对偶的特殊应用

有两类拉格朗日对偶可以应用的问题：

- 利用拉格朗日对偶去掉一些约束
- 问题本身是 $\min \max$ 的形式，可以通过拉格朗日对偶转换成更简单的形式。

以下分别举例说明。

4.3.1 例：[POJ Monthly 2015.5] Min-Max

题目描述 定义函数 $F(x_1, x_2, \dots, x_n) = \sum_{i=1}^n \mu_i x_i$ ，这里 μ_i 是常数，满足 $\sum_{i=1}^n \mu_i = 1$ 且对于任意 i 有 $0 \leq \mu_i \leq 1$ 。已知 $F(p_1, p_2, \dots, p_n) = C$ ，找到可能的 μ_i 使得 $F(q_1, q_2, \dots, q_n)$ 取到最大值或者最小值。

数据范围 $n \leq 50000$

解题思路 以求最大值为例。令 $g(\mu) = \sum_i p_i \mu_i - C$ ，那么固定了 λ 后问题变成最大化

$$\sum_i (q_i - \lambda p_i) \mu_i + \lambda C$$

，只需要找到 $q_i - \lambda p_i$ 最大的 i 令 $\mu_i = 1$ 即可。⁸

4.3.2 例：[Utpc2012.10] きたまの逆襲

题目描述 给定一张二分图 (U, V) ，边 $u \rightarrow v$ 有边权 w_{uv} 。有 k 个互不相交的集合 U_i ，可以用 b_i 的代价让所有边 $u \rightarrow v$ 满足 $u \in U_i$ 的 $w_{uv} + 1$ 。希望最大化最小权完美匹配的权值 - 花费的代价。（这里完美匹配指匹配的大小 = $|V|$ ）。

数据范围 $|U| \leq 100, |V| \leq 1000$

⁸ 本题也可以直接线性规划对偶，对偶后线性规划的约束为半平面交，最优值一定在凸包上取到。两个做法本质一样。

解题思路 令 λ_i 为集合 U_i 加权的次数, f_{uv} 表示 $u \rightarrow v$ 这条边是否在 f 这个完美匹配中, 为那么可以写出目标函数:

$$\max_{\lambda_1, \dots, \lambda_k \geq 0} \min_{|f|=|V|} \sum_i \left(\sum_{u \in U_i} \sum_v (w_{uv} + \lambda_i) f_{uv} - b_i \lambda_i \right) \quad (11)$$

整理一下, 得到

$$\max_{\lambda_1, \dots, \lambda_k \geq 0} \min_{|f|=|V|} \sum_{uv} w_{uv} f_{uv} + \sum_i \left(\sum_{u \in U_i} \sum_v f_{uv} - b_i \right) \lambda_i \quad (12)$$

正好是 $\min \max$ 的形式。把 λ_i 看作拉格朗日乘子, $\sum_{u \in U_i} \sum_v f_{uv} - b_i \leq 0$ 看作约束, 那么 (12) 就是拉格朗日对偶后的形式, 对偶回去就是

$$\begin{aligned} \min_{|f|=|V|} \sum_{uv} w_{uv} f_{uv} \\ \sum_{u \in U_i} \sum_v f_{uv} \leq b_i \end{aligned}$$

由于 U_i 不相交, 约束即为对于 U_i 中的点流量总和要 $\leq b_i$, 要求最小费用的完美匹配, 直接最小费用流即可。

最后还剩下一个问题, 原题意中要求 λ_i 都为整数, 而 (11) 中并没有这一要求。想到之前已经说明拉格朗日乘子和对偶变量等价, 而根据上文中的费用流的关于整数性的引理, 整数费用流的对偶也一定有整数最优解, 那么也就说明一定可以在拉格朗日乘子为整数处取到最优解, 从而说明了本题做法的正确性。

4.4 总结

在本节中, 作者总结了三类线性规划对偶的应用。

转化成费用流模型, 模型较为固定且整数性有保证, 困难之处在于把题目中原来的目标函数和约束改写成我们需要的形式。有时转化成费用流后还需要数据结构优化。

转化成动态规划模型, 一般针对序列上的问题, 原问题目标函数的系数较小故对偶后可能的状态较少。此种转化较为困难的地方在于证明最优解的整数性, 需要具体问题具体分析, 常用调整法和归纳法证明。

利用拉格朗日对偶的特殊应用有两种, 其中第一种和线性规划对偶的做法本质相同, 而第二种将 $\min \max$ 的问题转化成只有 \min 或 \max 的问题, 简化了目标函数。同样, 需要注意整数性的分析。

在信息学竞赛中, 转化成其它模型 (如半平面交) 的线性规划对偶也有许多。线性规划将题目的约束和目标本质地、显式地表达出来, 而线性规划对偶有时能挖掘出原问题的优越性质, 从而高效地解决问题。

对于线性规划对偶的应用, 应该还有许多的发展空间。欢迎大家在这类问题上进行更多的思考和总结。

5 致谢

- 感谢中国计算机学会提供学习和交流的平台。
- 感谢国家集训队教练高闻远教练的指导。
- 感谢南京外国语学校的李曙老师的关心与指导。
- 感谢与我交流相关内容的同学们的帮助。
- 感谢父母的关心与支持。

参考文献

- [1] Wikipedia, Linear Programming
- [2] 董克凡,《浅谈线性规划与对偶问题》,2016年信息学奥林匹克中国国家队候选队员论文集
- [3] 岩田陽一,《双对性》,JOI春合宿2018
- [4] Spyros Reveliotis of the Georgia Institute of Technology.,An Introduction to Linear Programming and the Simplex Algorithm
- [5] Gärtner, Bernd; Matoušek, Jiří (2006). Understanding and Using Linear Programming.

浅谈信息学竞赛中的弦图问题

湖南省长沙市长郡中学 郭城志

摘要

弦图是一类特殊的图，很多一般图上的 NPC 问题在弦图上都有优秀的解法。本文介绍了一些与弦图有关的知识，并通过几个例题展示了这些知识在信息学竞赛中的应用。

1 引言

弦图出现在信息学竞赛中已经有至少十年，但是没有得到广泛的普及，近几年出现在信息学竞赛中的弦图问题非常少，许多选手对弦图的了解也止步于基本定义和最大势算法的记忆上。本文较详细地介绍了一些与弦图有关的知识，希望能够帮助大家更加了解弦图。

本文第 2 节介绍了一些用到的记号和定义。

第 3 节介绍了一些弦图的基础知识。

第 4 节介绍了弦图的团树，以及团树、子树图、弦图之间的关系。

第 5 节通过几道例题，展示了弦图知识在信息学竞赛中的一些应用。

2 定义与约定

如无特殊说明，本文中的图均指无重边、自环的无向图。

对于图 $G = (V, E)$ ，令 $n = |V|$ 表示点集的大小， $m = |E|$ 表示边集的大小。为了方便，有时会直接将点从 $1 \dots n$ 编号。

定义 2.1 (邻域). 对于任意 $v \in V$ ，记 v 的邻域 $N(v) = \{u | (u, v) \in E\}$ ，即与 v 有边直接相连的点集。

定义 2.2 (导出子图). 对于任意 $A \subseteq V$ ，令 A 在 G 上的导出子图为 $G' = (A, E')$ ，其中 $E' = \{(u, v) | (u, v) \in E, u, v \in A\}$ ，即点集 A 和 G 中两端都在点集 A 中的边组成的子图。

定义 2.3 (团). 对于任意集合 $A \subseteq V$ ， A 是一个团当且仅当 $\forall u, v \in A, u \neq v$ ，有 $(u, v) \in E$ ，即点集 A 中任意两个不同的点都有边相连。如果不存在 $A' \supset A$ 使得 A' 是一个团，则称 A 为极大团。

定义 2.4 (弦图). G 是弦图当且仅当对于 G 中任意一个长度大于 3 的简单环, 都存在环上不相邻的两个点之间有边. 环上不相邻的两个点之间的边也叫做弦.

3 基础知识

3.1 弦图的点割集

定义 3.1 (点割集). 对于任意集合 $A \subseteq V$, 定义 A 是 G 关于 u, v 两点的点割集, 当且仅当 $u, v \notin A$, 且 $V \setminus A$ 在 G 上的导出子图中 u, v 不连通. 如果不存在 $A' \subset A$ 满足 A' 也是 G 关于 u, v 的点割集, 则称 A 是极小点割集.

关于弦图的极小点割集, 有以下性质:

引理 3.1. 令 $G = (V, E)$ 是弦图, A 是 G 关于两个点 u, v 的极小点割集, 则 A 是一个团.

证明. 令 $V \setminus A$ 中 u, v 所在的连通分量分别为 V_1, V_2 .

显然, $\forall x \in A, N(x)$ 包含 V_1, V_2 中的点, 否则删去 x 可以得到一个更小的点割集, 与 A 是极小点割集矛盾.

那么, 对于任意 $x, y \in A (x \neq y), N(x)$ 包含 V_1, V_2 中的点, $N(y)$ 也包含 V_1, V_2 中的点. 令 x, y 之间在 V_1, V_2 内部的最短路径分别为 $x \rightarrow x_1 \rightsquigarrow y_1 \rightarrow y$ 和 $y \rightarrow y_2 \rightsquigarrow x_2 \rightarrow x$. 那么, $x \rightarrow x_1 \rightsquigarrow y_1 \rightarrow y \rightarrow y_2 \rightsquigarrow x_2 \rightarrow x$ 是 G 中一个长度 > 3 的环.

因为 G 是弦图, 所以该环上必然存在一条连接不相邻点的边. 可以发现, 如果该边不是 (x, y) , 那么 $x \rightarrow x_1 \rightsquigarrow y_1 \rightarrow y$ 和 $y \rightarrow y_2 \rightsquigarrow x_2 \rightarrow x$ 两条路径中至少有一条不是最短路径, 产生矛盾. 因此边 (x, y) 存在, 故结论成立. \square

3.2 弦图的单纯点

定义 3.2 (单纯点). 对于图 $G = (V, E)$ 和任意 $v \in V$, v 是单纯点当且仅当 $N(v)$ 是一个团.

利用引理 3.1, 我们可以证明一个结论:

引理 3.2. 所有弦图 $G = (V, E)$ 存在单纯点, 不是完全图的弦图存在两个不相邻的单纯点.

证明. 按 n 归纳. $n = 1$ 时显然成立.

假设结论对于更小的 n 都成立.

G 是完全图或 G 不是连通图的情况是平凡的.

假设 G 不是完全图且 G 是连通图. 任取两点 (u, v) 满足 $(u, v) \notin E$, 令 A 为 G 关于 u, v 的极小点割集. 根据引理 3.1, A 是一个团. 令 $G \setminus A$ 中 u, v 所在的连通分量分别为 V_1, V_2 .

令 $L = V_1 \cup A$ 。若 L 是一个团，那么 V_1 中任取一点都是 V_1 的导出子图的单纯点。否则， L 的导出子图中存在不相邻的单纯点 x, y ，它们不可能都在 A 中，因为 A 是一个团。也就是说， x, y 至少有一个点在 V_1 中，它是 V_1 的导出子图的单纯点。

同理， V_2 的导出子图中也存在单纯点。又因为 A 是点割集，所以 V_1, V_2 之间不可能有边，我们就找到了 G 的两个不相邻的单纯点。 \square

3.3 弦图的完美消除序列

定义 3.3 (完美消除序列). 若 G 是弦图，则 G 的完美消除序列是一个点集的排列 p_1, \dots, p_n ，满足 $\forall i \in [1, n]$ ， $\{p_i\} \cup (N(p_i) \cap \{p_{i+1}, \dots, p_n\})$ 是一个团。即对于排列中任意一个点，其自己和排列中在其后面的与其相邻的点是一个团。 $\forall i \in [1, n]$ ，在给定完美消除序列的前提下，定义 $C(p_i) = \{p_i\} \cup (N(p_i) \cap \{p_{i+1}, \dots, p_n\})$ 。

每个弦图都存在完美消除序列。对于给定的弦图 $G = (V, E)$ ，可以使用最大势算法求出 G 的一个完美消除序列。该算法的流程为：对每个点 i 设置一个变量 $label_i$ ，初始值为 0。执行 n 轮以下过程：找到 i 使得 $label_i$ 最大，且点 i 还未被加入完美消除序列中，将点 i 加入到完美消除序列的最前面，然后 $\forall j \in N(i)$ ，将 $label_j$ 增加 1。

下面，我们证明该算法的正确性。不失一般性地，设最大势算法求出来的序列为 $1, 2, \dots, n$ 。我们首先需要引理：

引理 3.3. 对于任意弦图 $G = (V, E)$ ，不存在元素两两不同的序列 $v_0, \dots, v_k (k \geq 2)$ 满足：

1. $v_i, v_j \in E$ 当且仅当 $|i - j| = 1$ 。
2. $\forall i \in [1, k], v_0 > v_i$ 。
3. $v_1 < v_k$ 。

证明. 假设存在这样的序列，那么有 $v_1 < v_k < v_0$ ，且 $(v_0, v_1) \in E, (v_0, v_k) \notin E$ 。后者说明，在最大势算法的过程中， v_0 对 $label_{v_1}$ 有贡献，而对 $label_{v_k}$ 无贡献。为了使 v_k 比 v_1 先加入到完美消除序列，必存在 $x > v_k$ ，满足 $(x, v_k) \in E, (x, v_1) \notin E$ 。

任取一个这样的 x ，并取最小的 $j \in [2, k]$ 满足 $(x, v_j) \in E$ 。那么 $(v_0, x) \notin E$ ，否则 $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_j \rightarrow x \rightarrow v_0$ 是一个长度 > 3 的无弦环。

如果 $x < v_0$ ，则 v_0, \dots, v_j, x 也是一个满足以上三条性质的序列，否则 x, v_j, \dots, v_0 也是满足性质的序列。每一种情况都使得序列最后一个元素增大，因此重复进行这个过程可以得到有无穷多个序列满足条件，产生矛盾。 \square

假设存在 u, v, w 满足 $u < v < w, (u, v) \in E, (u, w) \in E, (v, w) \notin E$ ，那么 w, u, v 就是一个满足引理 3.3 中性质的序列，产生矛盾。因此最大势算法求出的是一个完美消除序列。

3.4 弦图的判定

有一个重要的弦图判定定理：

定理 3.1. 图 $G = (V, E)$ 是弦图的充要条件是， G 存在完美消除序列。

证明. 若 G 是弦图，我们可以通过最大势算法构造 G 的一个完美消除序列。

若 G 不是弦图，则 G 中存在一个长度 > 3 的环，满足环上不存在弦。假设完美消除序列存在，考虑环上在完美消除序列中最前面的点 v ，其在环上与 v_1, v_2 直接相连。根据完美消除序列的定义， v_1, v_2 之间也应该有边直接相连，与环上不存在弦产生矛盾。 \square

那么，我们可以利用完美消除序列的存在性来判定一个给定的图 $G = (V, E)$ 是否是弦图。对 G 运行最大势算法，如果 G 是弦图，那么我们可以求出 G 的一个完美消除序列；如果 G 不是弦图，我们求出来的就不是完美消除序列。

设最大势算法求出的序列为 $1, \dots, n$ ，对于每个 i ，设 $C(i) = \{i, p_1, \dots, p_k\}$ ，其中 p 递增。若直接定义判定其是否是一个完美消除序列，我们需要判定 $C(i)$ 是否是一个团，这需要 $\Theta(k^2)$ 次判定，总复杂度 $O(nm)$ 。实际上，只需判定 p_1 是否与 $C(i)$ 中的其他点都有连边即可，因为假设我们是按 $n \dots 1$ 的顺序依次枚举每个 i 的，那么根据归纳的思想， $p_j, p_k (j, k > 1)$ 是否有连边的判定，已经在 p_1 处完成了。这样，复杂度就降为了 $O(n + m)$ 。

3.5 弦图的极大团

令完美消除序列为 $1 \dots n$ ，显然极大团必定是某个 $C(i)$ 。对于一个 i ， $C(i)$ 不是极大团当且仅当存在 $j < i$ 满足 $C(i) \subset C(j)$ 。假设存在这样的 j ，并存在 k 使得 $j < k < i$ 且 $(j, k) \in E$ ，那么 $C(i) \subset C(k)$ 同样成立。因此可以假设 $C(j) \setminus \{j\}$ 中最小的点为 i 。

这种情况下，判定 $C(i) \subset C(j)$ 是否成立，显然只需判定 $|C(i)| + 1$ 是否等于 $C(j)$ 。因为 i 从 1 取到 n ， $C(i) \setminus \{i\}$ 中最小的点总共只有 $O(n)$ 种，因此总复杂度为 $O(n + m)$ 。

3.6 弦图的色数/团数

定义 3.4 (色数). G 的色数指的是最小的正整数 k ，使得存在一种给 G 中每个点染上 $[1, k]$ 中的一种颜色的方案，满足每条边两端点颜色不同，记作 $\chi(G)$ 。

定义 3.5 (团数). G 的最大团指的是大小最大的 $A \subseteq V$ ，满足 A 是一个团。 G 的最大团的大小称为 G 的团数，记作 $\omega(G)$ 。

按照任意完美消除序列从后往前，给每个点染上未使用过的编号最小的颜色，使用的总颜色数 $k = \chi(G) = \omega(G)$ 。

证明. 显然 $\chi(G) \leq \omega(G)$ 。按照这种方法染色，显然 $k = \omega(G)$ 。而根据定义 $k \geq \chi(G)$ ，所以 $k = \chi(G) = \omega(G)$ 。 \square

3.7 弦图的最大独立集/最小团覆盖

定义 3.6 (最大独立集). G 的最大独立集指的是大小最大的 $A \subseteq V$, 满足 A 中任意两点没有边相连. G 的最大独立集的大小记作 $\alpha(G)$.

定义 3.7 (最小团覆盖). G 的最小团覆盖指的是用最少的团覆盖 G 中的所有点. 使用的团的数量记作 $\kappa(G)$.

按照任意完美消除序列从前往后考虑每个点, 若当前考虑的点与已被选入独立集中的点没有边相连, 就将当前点加入独立集, 最终得到的是最大独立集. 设最大独立集为 $\{v_1, \dots, v_k\}$, 则最小团覆盖为 $\{C(v_1), \dots, C(v_k)\}$.

证明. 首先, 这些团确实是 G 的一个最小团覆盖. 否则, 若 G 中一个点不在该团覆盖内, 说明完美消除序列中在其前面且与其有边的点都未被选择, 那么其应该被加入独立集内.

其次, 显然 $\alpha(G) \leq \kappa(G), k \leq \alpha(G), k \geq \kappa(G)$, 因此 $k = \alpha(G) = \kappa(G)$. □

4 子树图和团树

4.1 弦图的团树

定义 4.1 (团树). 对于图 $G = (V, E)$, 定义其团树为 $T = (\mu(G), E')$, 满足对于任意 $v \in V$, $\mu_v(G)$ 在 T 上的导出子图连通. 团树可能不存在.

每个弦图都存在团树. 下面我们将给出一种方法, 对于任意弦图 $G = (V, E)$, 构造出其团树.

因为团树是一棵树, 所以“团树上的团”这个描述没有意义. 以下为了更加直观, 我们把团树上的一个点, 即 G 中的一个团也称为团树上的团.

求出 G 的任意一个完美消除序列, 假设为 $1 \dots n$. 我们将按照完美消除序列从后往前增量构造, 即对于 $i = n \dots 1$, 依次构造出 $\{i \dots n\}$ 的导出子图的团树. 在构造过程中, 我们会令 G 中已加入的每个点 i 指向加入点 i 时团树上新产生的团.

假设我们已经对 $\{i+1 \dots n\}$ 的导出子图构造完毕, 现在需要加入点 i .

令 j 为 $C(i) \setminus \{i\}$ 中最小的点. 若不存在这样的 j , 则将 $\{i\}$ 作为一个新团加入团树, 与团树上任意一个团连边.

假设 j 存在. 若 $|C(i)| = |C(j)| + 1$, 并且 j 指向的团等于 $C(j)$, 直接将 i 加入该团, 即将团树上 $C(j)$ 替换成 $C(i)$.

否则, 将 $C(i)$ 作为一个新团加入团树中, 并与 j 指向的团连边.

容易发现算法流程中每个点 i 指向的团的大小是单调不降的, 且加入点 i 时其指向的团是 $C(i)$. 因此, 判定 j 指向的团是否等于 $C(j)$, 只需将其现在的大小与 $|C(j)|$ 比较即可. 因此, 该算法的时间复杂度为 $O(n + m)$.

下面, 我们证明该算法的正确性:

考虑加入点 i 时三种不同的情况。

1. 点 i 是孤立点。这种情况下显然正确。
2. $|C(i)| = |C(j)| + 1$, 并且 j 指向的团等于 $C(j)$ 。第一个条件说明 $C(i) = C(j) \cup \{i\}$, 因为根据完美消除序列的定义, $C(i) \setminus \{i\} \subseteq C(j)$ 。第二个条件说明, $C(j)$ 是 $\{i+1, \dots, n\}$ 的导出子图中的极大团。那么, $C(j)$ 不再是 $\{i, \dots, n\}$ 的导出子图中的极大团, 而 $C(i)$ 是新产生的一个极大团。而其他极大团在加入点 i 后仍然存在, 因为点 i 的加入只可能使得是 $C(i) \setminus \{i\}$ 子集的极大团变成非极大团, 而这些子集中除掉 $C(j)$ 以外的团在加入点 j 后必定已不是极大团。另一方面, 得到的新树与原团树相比, 区别仅仅在于某一个团内多出了新加入的点 i , 因此包含每个点的团连通这一性质仍然满足。
3. 其他情况。按照类似的分析, 所有 $\{i+1, \dots, n\}$ 的导出子图中的极大团都不会变成非极大团; 又因为 $C(i) \setminus \{i\} \subseteq C(j)$, 包含每个点的团连通这一性质仍然满足。

4.2 子树图与弦图的关系

定义 4.2 (交图). 考虑一个集合族 \mathcal{F} , 其交图 $G = (V, E)$ 是这样图: 每个点对应 \mathcal{F} 中的一个集合, 两个点之间有边当且仅当其所对应的集合交集非空。

定义 4.3 (子树). 对于树 $T = (V, E)$, 称 $V' \subseteq V$ 是 T 的子树, 当且仅当 V' 在 T 上的导出子图是连通图。

定义 4.4 (子树图). 对于一棵树的子树族 \mathcal{F} , 其交图称为子树图。

引理 4.1. 若图 $G = (V, E)$ 是子树图, 则 G 是弦图。

证明. 令 G 是树 T 上子树族 \mathcal{F} 的交图。我们将构造 G 的一个完美消除序列, 以此证明 G 是弦图。

设 A_i 为点 i 对应的 \mathcal{F} 中的子树。令 T 中任意一点为树根, 将 V 中所有点排序, 令排序后为 $1, \dots, n$, 满足 $\forall i < j, A_i$ 的根的深度不小于 A_j 的根的深度。因此, $\forall i < j$, 若 $A_i \cap A_j \neq \emptyset$, 则 A_j 必包含 A_i 的根。由此可得, $\forall i \in [1, n]$, 满足 $j > i$ 且 $A_i \cap A_j \neq \emptyset$ 的 A_j 一定两两有交, 故 $1, \dots, n$ 是 G 的一个完美消除序列。

□

4.3 团树与子树图的关系

引理 4.2. 若图 $G = (V, E)$ 是存在团树, 则 G 是子树图。

证明. 令其团树为 T 。令

$$\mathcal{F} = \{\mu_v(G) | v \in V\}$$

对于任意 $v \in V$ ，我们令 G 中的点 v 与 \mathcal{F} 中的元素 $\mu_v(G)$ 相对应。我们将证明 G 是 \mathcal{F} 的交图。

对于任意 $u, v \in V$ ，若 $\mu_u(G) \cap \mu_v(G) \neq \emptyset$ ，那么存在 G 中的极大团 A ，满足 $A \in \mu_u(G) \cap \mu_v(G)$ ，即 $A \in \mu_u(G), A \in \mu_v(G)$ 。因此 $u, v \in A$ ，即 $(u, v) \in E$ 。

另一方面，如果 $(u, v) \in E$ ，那么存在极大团 A 满足 $u \in A, v \in A$ （只需从 $\{u, v\}$ 开始任意扩展直到不能扩展为止，就可以得到一个这样的 A ）。因此 $A \in \mu_u(G) \cap \mu_v(G)$ ，即 $\mu_u(G) \cap \mu_v(G) \neq \emptyset$ 。

那么， $(u, v) \in E$ 当且仅当 $\mu_u(G) \cap \mu_v(G) \neq \emptyset$ ，即 G 是 \mathcal{F} 的交图。 \square

最后，根据本节内容，我们可以得到一个定理：

定理 4.1. 对于图 $G = (V, E)$ ，以下三个命题等价：

1. G 是子树图。
2. G 是弦图。
3. G 有团树。

5 应用

例题 1. 对于一个长度为 n 的字符串 s ，定义其 $next$ 数组为一个长度为 n 的整数数组，其中 $next_i = \max\{j | j < i, s[1 \dots j] = s[i - j + 1 \dots i]\}$ 。

现在给定一个长度为 n 的 $next$ 数组和字符集大小 k ，求有多少字符串 s ，满足 s 的 $next$ 数组为给定的 $next$ 数组。

$$n \leq 10^7$$

模拟 KMP 算法的过程，我们会得到若干条限制，每条限制形如 s 的某两个字符必须相等，或 s 的某两个字符必须不相等。

考虑一个 n 个点的无向图 G ，每个点对应 s 的一个字符。选择 s 每个位置填的字符就是给 G 中每个点染色；对于每一条限制，我们在两个字符对应的 G 中的点之间连形如两端点颜色必须相等/必须不相等的边。问题转化为：有多少种给 G 中每个点染上一种 $[1, k]$ 的颜色的方案，满足所有边的限制。

相等边是好处处理的，将每一个由相等边连成的极大连通块缩成一个点即可。设缩点后得到的新图是 G' 。现在的问题是，求 G' 有多少种染色方案，使得每条边两个端点的颜色不同。这是 NPC 问题，所以我们需要挖掘 G' 更多的性质。

分析 KMP 算法过程中求出的限制。考虑一棵树 T ，对于每个 $i \in [2, n]$ ， T 中点 i 的父亲是 $next_{i-1} + 1$ ，点 1 是树根。限制实际上就是，某一些点 i 有一个祖先 f_i ， i 的颜色必须和 f_i 的颜色相同，且和 i 到 f_i 路径上（不包括端点）的所有点颜色不同；另一些点与自己到根路径上（不包括自己）的所有点颜色不同。那么，一个相等边的极大连通块存在唯一一个块根，两个连通块有不等边当且仅当其块根有祖孙关系。

通过和引理 4.1 的证明类似的思想，容易发现， G' 是一个弦图，将所有相等块按照块根的编号从大到小排序就是一个完美消除序列。

怎样求弦图的 k 染色方案数呢？这是简单的：按完美消除序列从后往前依次决定每个点的颜色。假设当前考虑到了点 v ，根据完美消除序列的定义， $C(v) \setminus \{v\}$ 是一个团，因此 $C(v) \setminus \{v\}$ 中的点一定被染成了两两不同的颜色。那么，点 v 可染的颜色数就是 $k - |C(v)| + 1$ ，答案就是每个点可染颜色数的乘积。

时间复杂度 $O(n)$ 。

例题 2. 给定弦图 $G = (V, E)$ ，Alice 和 Bob 将在 G 上博弈。

第一轮，Alice 选择不超过 k 个点，设选中的点集为 A_1 ，然后 Bob 选择 $V \setminus A_1$ 中的一个点，设其为 v_1 。

第 $i (i > 1)$ 轮，Alice 选择不超过 k 个点，设选中的点集为 A_i ，然后 Bob 选择 $V \setminus A_i$ 中的一个点，设其为 v_i 。Bob 需要满足 v_{i-1} 和 v_i 之间存在一条不经过 $A_{i-1} \cap A_i$ 内的点的路径。

如果某一轮 Bob 无法选择 v_i ，则 Alice 获胜。如果游戏能无限进行下去，则 Bob 获胜。

求出最小的 k ，使得 Alice 能获胜。

$n, m \leq 10^5$

若 $k < \omega(G)$ ，显然 Bob 永远可以在最大团内选出一个点。

否则，考虑 G 的团树 T' ， G 中的每一个点对应 T' 中的一个子树 ($\mu_v(G)$)。原问题可以转化为一个树上的问题：给定树 T 和一个子树族 \mathcal{F} ，每一轮 Alice 选择 \mathcal{F} 中不超过 k 个子树 \mathcal{A}_i ，然后 Bob 选择一个不在 \mathcal{A}_i 中的子树 F_i ，满足存在一个子树序列，以 F_{i-1} 开头， F_i 结尾，相邻子树有交，且不包含 $\mathcal{A}_{i-1} \cap \mathcal{A}_i$ 中的子树。并且，对于 T 中的每个点， \mathcal{F} 中包含其的子树个数 $\leq k$ 。

第一步，Alice 可以选择所有包含 u 的子树，其中 u 是 T 中任意一个点。那么，删去点 u 后， T 将包含若干个连通分量，根据规则，Bob 必须选择一个完全在某个连通分量内的子树。假设 Bob 选择的子树所在的连通分量与 u 直接相连的点是 v 。那么，在第二步，Alice 可以选择所有包含 v 的子树，删去 v 后， v 所在的连通分量又分成若干个更小的连通分量。容易发现，Bob 再次选择的子树只能完全在这些更小的连通分量中……以此类推，最终 Bob 一定会无法选择子树。

所以，答案就是 $\omega(G)$ 。时间复杂度 $O(n + m)$ 。

例题 3. 给定弦图 $G = (V, E)$ ，求至少删去多少个点，才能使得 G 中不存在环。

$n, m \leq 10^5$

弦图删去若干个点后还是弦图，所以不存在环等价于每个极大团的大小都 ≤ 2 。

考虑求最多能保留的点数。建出 G 的团树 T' ，限制就是 T' 中的每个团内至多只能保留两个 G 中的点。

接下来，我们可以将原问题作和**例题 2**一样的转化，不过此处额外用到了一个团树的性质：每个点代表原图的一个极大团。问题转化为：给定树 T 和一个子树族 \mathcal{F} ，需要从 \mathcal{F} 中选出尽可能多的子树，满足对于 T 上的每个点 u ，选出的包含 u 的子树个数不超过 2。

转化后的问题可以用一个简单的树形 DP 解决。将 T 以任意点为根，设 S_u 表示 T 中所有到根的简单路径经过点 u 的点，设 $f_{u,i,j}$ 表示考虑完了 \mathcal{F} 中所有与 S_u 有交的子树，选出的包含 u 的子树分别为 i, j （其中 i, j 可以为 0，表示选出的包含 u 的子树个数小于 2），最多能选出的子树个数。DP 转移需要满足：对于一个 u 的儿子 v ，若 $i \neq 0$ 且子树 i 包含点 v ，则从 $f_{v,i',j'}$ 转移需要保证 $i' = i$ 或 $j' = i$ ；同样地，若 $j \neq 0$ 且子树 j 包含点 v ，则需要保证 $i' = j$ 或 $j' = j$ 。

根据团树的构建过程，可以得到 T' 上每个团的大小之和是 $O(m)$ 的，并且每个团的大小是 $O(\sqrt{m})$ 的（因为若存在一个大小为 t 的团，则团内有 $\Theta(t^2)$ 条边，而总边数只有 m ）。也就是说， \mathcal{F} 中每个子树的大小之和是 $O(m)$ 的，且对于 T 中每个点 u ， \mathcal{F} 中包含 u 的子树个数是 $O(\sqrt{m})$ 的。因此，DP 的总状态数是 $O(m\sqrt{m})$ 的，若精细实现使得每一次转移 $O(1)$ 完成，总时间复杂度即为 $O(m\sqrt{m})$ 。

6 总结

本文第 3 节讲述了一些弦图的基础知识，体现出了弦图有很多优秀的性质，一些在一般图上难以解决的问题在弦图上可以轻易解决。

第 4 节介绍的团树给出了一种化弦图为树的方法，利用团树我们可以更好地分析弦图的结构，解决更多的问题；同时也说明了只有弦图存在团树，该方式难以被扩展到一般图上解决问题。

第 5 节的几个例题介绍了第 3, 4 节知识的一些小应用。

事实上，因为作者的水平有限，这篇文章介绍的内容还相当浅，弦图还有很多可以被挖掘的东西。希望本文能够起到一个抛砖引玉的作用，吸引更多的读者来研究弦图，让更多优秀的弦图题出现在信息学竞赛中。

7 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练高闻远的指导。

感谢长郡中学谢秋锋老师的关心和指导。

感谢彭思进同学、高子翼同学对我的启发和对本文的帮助。

感谢父母对我的关心与支持。

参考文献

- [1] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. 3. *Combin. Theory Ser. 8*, 16:47-5G, 1974.
- [2] Spinrad, J.P. *Efficient Graph Representations*; Fields Institute Monographs, American Mathematical Society: Providence, RI, USA, 2003.
- [3] Wikipedia, Chordal_graph
- [4] OI Wiki, 弦图

浅谈 Lyndon 分解

雅礼中学 胡昊

摘要

字符串是信息学奥林匹克竞赛的重要考点, Lyndon 常用于与字典序最优化相关的问题。

1 基本定义

定义 $|S|$ 为字符串 S 的长度, S_i 为 S 中的第 i 个字符 ($i \in [0, |S|)$)。

定义两个字符串 S, T 相等, 则须满足 $|S| = |T|, \forall i \in [0, |S|), S_i = T_i$ 。

定义一个字符串 S 的一个子串 $S_{l..r}$ 为 S 中的第 l 个字符到第 r 个字符组成的字符串。

定义字符串 S 是另一字符串 T 的前缀, 当且仅当 $T_{0..|S|-1} = S$, 可以看出任意字符串 A 一定是 A 的前缀。

定义字符串 S 是另一字符串 T 的后缀, 当且仅当 $T_{|T|-|S|..|T|-1} = S$, 可以看出任意字符串 A 一定是 A 的后缀。

两个不相等的字符串 S, T 比较大小时, 若 T 是 S 的前缀, 则 $S > T$; 若 S 是 T 的前缀, 则 $S < T$; 否则找到第一个位置 p , 使得 $S_{0..p-1} = T_{0..p-1}, S_p \neq T_p$, 则 S, T 的大小关系与 S_p, T_p 的大小关系相同。

对于两个字符串 A, B , 定义 AB 为将 B 直接连接在 A 后得到的字符串, 即: $|AB| = |A| + |B|$,

$$(AB)_i = \begin{cases} A_i & i < |A| \\ B_{i-|A|} & i \geq |A| \end{cases}$$

根据上一条定义, 可以定义出 $S^k = S^{k-1}S, S^1 = S$ 。

2 后缀数组

后缀数组作为处理字符串问题的有力工具, 是理解 Lyndon 分解的重要前置知识, 本章将大致地提一下, 详细学习可以参考“参考文献 1”。

2.1 定义

将 S 的每一个后缀排序（它们肯定互不相同），可以得到后缀数组 $sa_{0\dots|S|-1}$ ，是一个 0 到 $|S|-1$ 的排列，满足 $\forall i \in [1, |S|), S_{sa_{i-1}\dots|S|-1} < S_{sa_i\dots|S|-1}$ 。

定义 rk 数组，满足 $rk_{sa_i} = i$ 。

2.2 后缀排序

后缀排序的常用方法为倍增排序。

根据字符串大小比较的定义，若 $S_{0\dots p} < T_{0\dots p}, p \leq q$ ，则 $S_{0\dots q} < T_{0\dots q}$ 。

于是，可以先仅取所有后缀的前 2^0 个字符进行比较，然后根据仅取前 2^i 个字符排序后结果进行双关键字排序，就可以得到取前 2^{i+1} 个字符排序后的结果。

双关键字排序时使用基数排序可以做到 $O(n \log n)$ 求后缀数组，使用 SA-IS 算法或 DC3 算法可以做到 $O(n)$ ，因为这不是本文重点，故不在此介绍。

3 Lyndon 分解

3.1 定义

对于一个字符串 S ，若 S 小于它的所有不为 S 的后缀，则称 S 为简单串（或 Lyndon 串）。

定义 S 的 Lyndon 分解为 $S = w_1 w_2 w_3 \dots w_m$ ，其中所有 w 均为简单串，且 $w_1 \geq w_2 \geq w_3 \geq \dots \geq w_m$ 。

Lyndon 分解是唯一的，且对于每个字符串都必定有 Lyndon 分解，下面将证明 Lyndon 分解是唯一的，并通过一个较差的算法（时间复杂度 $O(n \log n)$ ）来证明每一个字符串都有 Lyndon 分解。

3.2 简单串的性质

1. 字符串 S 为简单串，当且仅当 S 小于所有与 S 循环同构的串，即 $\forall i \in (0, |S|), S < (SS)_{i\dots i+|S|-1}$ 。

S 为简单串时，则 S 小于所有与 S 循环同构的串的证明：

若 $S = AB$ ，且 $S \geq BA$ ，则 $B > AB \geq BA$ ，不成立，则原命题成立。

S 小于所有与 S 循环同构的串，则 S 为简单串时的证明：

若 $S = AB$ ，且 $S \geq B$ ， B 取所有符合条件中最短的，则 $BA > AB \geq B$ ，则 B 是 A 的前缀或 A 是 B 的前缀。

若 B 是 A 的前缀, 则令 $A = BC$, 则 $S = BCB$, 则 $BBC > BCB \geq B$, 则 $BC > CB$, 又 $CBB > BCB$, 矛盾。

若 A 是 B 的前缀, 则令 $B = AC$, 则 $S = AAC$, 则 $ACA > AAC \geq AC$, 则 $CA > AC \geq C$, 又 B 取所有符合条件中最短的, 所以 $S = AAC < C$, 所以 $AC \geq C > AAC$, 所以 $C > AC$, 矛盾。

故原命题得证。

2. 若 Sa 为简单串, 其中 a 为一个字符, 则 $\forall b \geq a, Sb$ 为简单串, 证明:

对于 S 的每个非 S 后缀 T , 有 $Tb > Ta > Sa$, 因为 $|Tb| < |Sa|$, 所以 Tb, Sa 第一个不同的字符位置不为 $|Sa| - 1$, 所以将 a 换为 b 不等号保持不变, 即 $Tb \geq Sb$ 。

3.3 唯一性证明

若 $S = w_1w_2w_3 \cdots = w'_1w'_2w'_3 \cdots$, 不妨设 $w_1 \neq w'_1$ (找到第一个 $w_p \neq w'_p$ 的 p , 令 $S' = w_pw_{p+1} \cdots$, 可以转化为这种情况), 并假设 $|w_1| < |w'_1|$ (不然交换 w, w')。

令 $w'_1 = w_1A, w_2 = AB$, 则 $A > w_1A > w_1 \geq AB$, 可以推得 $A > AB$, 这显然是错误的, 所以可以推得: **Lyndon 分解是唯一的。**

3.4 $O(n \log n)$ 的分解方法

定义数组 a_i 为最小的 j , 大于 i 且 $S_{j \dots |S|-1} < S_{i \dots |S|-1}$, 若不存在这样的 j , 可以认为 $a_i = |S|$ 。那么, S 的 Lyndon 分解的第一项为 $S_{0 \dots a_0-1}$, 且后面 $m-1$ 项就是 $S_{a_0 \dots |S|-1}$ 的 Lyndon 分解。

3.4.1 正确性证明

证明可以分为 $S_{0 \dots a_0-1}$ 是简单串, 和 $S_{0 \dots a_0-1} \geq S_{a_0 \dots a_1-1}$ 两部分。

1. 令 $S = ABC$, 其中 $S_{0 \dots a_0-1} = AB$, 根据算法有 $C < ABC < BC$, 假设 $AB \geq BA$, 那么 B 一定是 AB 的前缀。

设 $AB = BD$, 则 $C < BDC < BC$, 则 $DC < C$ 。

因为算法得到 C 是比 S 小的最长的后缀, 与 $DC < C$ 矛盾, 所以原命题成立。

所以 $S_{0 \dots a_0-1}$ 是简单串。

2. 令 $S = ABC$, 其中 $S_{0 \dots a_0-1} = A, S_{a_0 \dots a_1-1} = B$, 根据算法有 $C < BC < ABC$, 假设 $A < B$, 那么 A 一定是 B 的前缀。

设 $B = AD$ ，则 $C < ADC < AADC$ ，则 $DC < ADC$ 。

因为算法得到 C 是比 ADC 小的最长的后缀，与 $DC < ADC$ 矛盾，所以原命题成立。

所以 $S_{0\dots a_0-1} \geq S_{a_0\dots a_1-1}$ 。

所以，我们证明了这个算法可以求出一个字符串的 Lyndon 分解，同时每一个字符串都必定有 Lyndon 分解。

3.4.2 算法伪代码

Algorithm 1: Algorithm based on SA

Input: 字符串 S

Output: S 的 Lyndon 分解

```

1  $n \leftarrow |S|$ ;
2  $sa \leftarrow \text{Sufsort}(S)$ ;
3  $X \leftarrow \{n\}$ ;
4 for  $i \in [0, n)$  do
5    $a_{sa_i} \leftarrow X.\text{upperbound}(sa_i)$ ;
6    $X.\text{insert}(sa_i)$ ;
7 end
8  $i \leftarrow 0$ 
9 while  $i < n$  do
10   $\text{Print}(S_{i\dots a_i-1})$ ;
11   $i \leftarrow a_i - 1$ ;
12 end

```

3.5 Duval 算法

定义字符串 S 为近似简单的，当且仅当 $S = www\dots w\bar{w}$ ，其中 \bar{w} 是 w 的前缀，且 w 是简单串，据此定义，简单串也是近似简单串。

在算法运行过程中， S 分为 3 部分 $s_1s_2s_3$ ，其中 s_1 是已经完成分解的部分， s_2 是近似简单串， s_3 是没有任何处理的部分。

我们可以用三个指针 i, j, k 来保存运行状态： i 为 s_2 的开始字符， j 为 s_3 的开始字符， k 则记录近似简单串 s_2 的情况： $k = j - |w|$ 。

Duval 算法即不断尝试往 s_2 的末尾加入 s_3 的首字符，根据 S_j 和 S_k 的大小关系，有：

- $S_j = S_k$

显然，将 S_{j+1} 加入 s_2 后， s_2 还是一个近似简单串。

可以令 $j \leftarrow j + 1, k \leftarrow k + 1$ 。

- $S_j > S_k$

令 $T = s_2 S_j = www \dots \bar{w} S_j = w^a \bar{w} S_j$, 因为 $\bar{w} S_j > w$, 将 T 循环位移 (位移长度在 $(0, |T|)$ 间), 则:

若循环向前移 $b|w|$ 步, 则 $T' = w^{a-b} \bar{w} S_j w^b > w^{a-b} ww^{b-1} \bar{w} S_j = T$ 。

若循环向前移 u 步, u 不是 $|w|$ 的倍数, 且 $u < |T| - |w|$, 因为 w 是简单串, 则 $(T')_{0, |w|-1} > w$, 则 $T' > T$; 当 $u > |T| - |w|$ 时, 因为 S_2 大于 w 的对应项, 所以 $T' > T$ 。

综上, 若循环位移长度在 $(0, |T|)$ 间, $T' > T$, 所以, 此时 $s_2 S_j$ 是一个简单串, 根据定义, 可以认为加入 S_j 后, s_2 依然是近似简单串。

可以令 $j \leftarrow j + 1, k \leftarrow i$ 。

- $S_j < S_k$

令 $T = s_2 S_j = w^a \bar{w} S_j$, 因为 $\bar{w} S_j < w$, 循环位移 $|w|$ 步后, 显然有 $T' < T$, 此时 T 不再是简单串。

因为 $\bar{w} S_j < w$, 考虑 $O(n \log n)$ 的算法的过程, 比 $w^a \bar{w} S_j$ 小的最长后缀为 $w^{a-1} \bar{w} S_j$, 所以前面的 a 个 w 可以加入 Lyndon 分解中。

修改 i 的值, 然后令 $j \rightarrow i + 1, k \rightarrow i$ (一个单独的字符一定是简单串, 所以直接令 s_2 为 s_1 后的第一个字符)。

3.5.1 算法伪代码

Algorithm 2: Duval Algorithm**Input:** 字符串 S **Output:** S 的 Lyndon 分解

```

1  $n \leftarrow |S|$ ;
2  $i \leftarrow 0$ ;
3 while  $i < n$  do
4    $j \leftarrow i + 1$ ;
5    $k \leftarrow i$ ;
6   while  $j < n$  do
7     if  $S_j = S_k$  then
8        $j \leftarrow j + 1$ ;
9        $k \leftarrow k + 1$ ;
10      Continue;
11     end
12     if  $S_j > S_k$  then
13        $j \leftarrow j + 1$ ;
14        $k \leftarrow i$ ;
15       Continue;
16     end
17     Break;
18   end
19   while  $i \leq k$  do
20     Print( $S_{i..i+(j-k)-1}$ );
21      $i \leftarrow i + (j - k)$ ;
22   end
23 end

```

3.5.2 复杂度证明

在一次外层循环内，向 s_1 中添加 u 个字符，那么 j 的值一定小于 $i + 2u$ ，内层循环次数不大于 $2u$ 。

对它求和，内层循环总的次数不大于 $2n$ ，外层循环次数不大于 n ，所以 Duval 算法的时间复杂度为 $O(n)$ 。

4 例题

4.1 例题 1

4.1.1 题面

给定长为 n 的字符串 S ，求出 S 的最小表示法。

4.1.2 题解

将 SS Lyndon 分解，找到分解后最后一个字符串，它的首字符为 S_p ，且 $p \in [0, |S|)$ ， S 的最小表示法就是 $S_{p \dots p+|S|-1}$ 。

考虑到 $SS = w_1 w_2 \dots w_m$ ，其中开头为 S_p 的为 w_i 。

因为若取 w_{i-1} 为开头，当 $w_{i-1} > w_i$ 时，显然取 w_i 为开头较优，当 $w_{i-1} = w_i$ 时，找到第一个 $j > i$ ，使得 $w_j \neq w_i$ ，显然 $w_j < w_i$ ，则 w_j 越靠前越优，故应选 w_i 作为开头。

时间复杂度 $O(n)$ 。

4.2 例题 2

4.2.1 题面

给定长度为 n 的字符串 S ，将 S 分为最多 k 个串 $c_1 c_2 \dots c_k$ ，求 $\max c_i$ 的最小值。

4.2.2 题解

考虑 S 的 Lyndon 分解，我们令 $S = w_1^m w_{m+1} \dots$ 。

如果 $k > m$ ，可以划分为 m 个 w_1 ，和 1 个 $w_{m+1} \dots$ ，此时答案是 w_1 。

如果 $k \leq m$ ，可以划分为 $k-1$ 个 $w_1^{\frac{m}{k}}$ ，和 1 个 $w_1^{\frac{m}{k}} w_{m+1} \dots$ ，此时答案为 $w_1^{\frac{m}{k}} w_{m+1} \dots$ 。

否则，可以额外花费一次划分，将最后的一个 $w_1^{\lfloor \frac{m}{k} \rfloor}$ ，和它后面的 $w_{m+1} \dots$ 划开，答案为 $w_1^{\lfloor \frac{m}{k} \rfloor}$ 。

时间复杂度 $O(n)$ 。

4.3 例题 3

4.3.1 题面

给定长度为 n 的字符串 S ，将 S 分为最多 k 个串 $c_1 c_2 \dots c_k$ ，求 $\max c_i$ 的最小值。

q 次询问，每次询问 S 的一个后缀，并重新给定 k 。

4.3.2 题解

用 $O(n \log n)$ 的 Lyndon 分解算法可以求出每个后缀的 Lyndon 分解的 w_1 ，并记录每个位置开始时，Lyndon 分解出的 w_1 的重复次数。

知道这两个信息，就可以和上题用相同的方法做出来了。

时间复杂度 $O(n \log n)$ 。

4.4 例题 4

4.4.1 题面

给定一个字符串 S ，求出 S 的每个前缀的最小后缀。

4.4.2 题解

将 S Lyndon 分解，令 $S = w_1 w_2 w_3 \dots w_m$ ，显然，前缀 $w_1 w_2 \dots w_k$ 的最小后缀为 w_k 。

但是如果前缀是 $w_1 w_2 \dots \bar{w}_k$ (\bar{w}_k 指 w_k 的前缀)，答案就不一定是 \bar{w}_k ，例如 aab 的 Lyndon 分解为 aab ，前缀 aa 的最小后缀为 a 而不是 aa 。

Duval 算法的运行过程，我们考虑怎么在处理 s_2 时求出每个前缀的最小后缀。

- 当 $s_2 = w$ 时，最小后缀显然为 w 。
- 当 $s_2 \neq w$ 时， s_2 为 w 重复若干次（最后一次不一定完整）得到。

如果起始字符不在 \bar{w} 内，那么它比 w 大，一定不优。

在 \bar{w} 内的情况，与去掉末尾 $|w|$ 个字符的情况相同，答案长度一样。

5 总结

Lyndon 分解可以作为处理字符串最优化的有力工具。

在上文中，简单介绍了求 Lyndon 分解的两种算法，并证明了它们正确性。

其中 $O(n \log n)$ 的算法思路简单，且能够求出每一个后缀的 Lyndon 分解，在例题 23 中有所说明。

其中 $O(n)$ 的算法复杂度优，代码简单，能够求出每一个前缀的 Lyndon 分解，在例题 4 中有所说明。

参考文献

- [1] 许智磊,《后缀数组》, IOI2004 国家集训队论文。
- [2] [cp-algorithms](#)

感谢

- [1] 雅礼中学的同学老师们为我提供的帮助。
- [2] CCF 给予的本次撰写论文的机会。

信息学竞赛中构造题的常用解题方法

西南大学附属中学校 蒋凌宇

摘要

构造题是近年的算法竞赛中常见的一种题目类型。本文对构造题中常见的几个解题方法进行了介绍，包括抽屉原理的运用、DFS 树的运用、递归法的运用等，并给出了例题和讲解。

1 引言

在近年的算法竞赛中，构造题的出现越来越频繁。不同于传统的计数、最优化等问题，构造题只要求选手给出一组满足约束条件的解，而不需要统计解的数量，或是寻找一组“最优”的解。然而，由于其模型繁多，涉及图论、数论、字符串等各领域，且常常难以发现，要解决起来并不容易。

本文对构造题中较常出现的一些解题思路进行了介绍，并给出了例题和讲解，希望对读者有所启发，在解决构造题时能更加得心应手。

2 抽屉原理

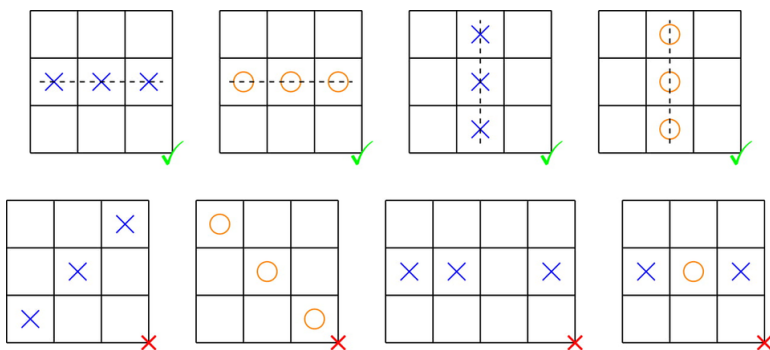
抽屉原理，或称为鸽巢原理，是组合数学中一个非常重要的原理。通常的表述是，若将 n 件物品放入 k 个抽屉，则其中一定有一个抽屉包含至少 $\lceil \frac{n}{k} \rceil$ 件物品，也一定有一个抽屉包含至多 $\lfloor \frac{n}{k} \rfloor$ 件物品。

在一些构造题中，常常会要求构造一个权值至少为（或不超过）某一个数的方案。很多时候，可以考虑找出若干个可行的方案，使得它们的权值之和是定值。假设找出了 k 个可行方案，其总权值和为 n ，由抽屉原理，这些方案中最小的权值一定不超过 $\lfloor \frac{n}{k} \rfloor$ ，最大的权值至少为 $\lceil \frac{n}{k} \rceil$ 。

2.1 Errich-Tac-Toe¹

2.1.1 题目大意

给定一张 n 行 n 列的棋盘，每个格子可能是空的或包含一个标志，标志有 X 和 O 两种。如果有三个相同的标志排列在一行或一列上的三个连续的位置，则称这个棋盘是一个胜局，否则称其为平局。



例如，上图第一行的局面都是胜局，而第二行的局面都是平局。

在一次操作中，你可以将一个 X 改成 O，或将一个 O 改成 X。

设棋盘中标志的总数为 k ，你需要用不超过 $\lfloor \frac{k}{3} \rfloor$ 次操作把给定的局面变成平局。

2.1.2 数据范围

$1 \leq n \leq 300$ 。

2.1.3 解题过程

不妨将行列都用 $0, 1, \dots, n-1$ 编号，将第 r 行第 c 列的格子记为 (r, c) 。

我们将所有格子分成 3 类，其中第 i ($0 \leq i < 3$) 类包含所有满足 $r+c \equiv i \pmod 3$ 的格子 (r, c) 。不难发现，在一行或一列上的连续三个格子包含第 0, 1, 2 类格子各一个。

由此，不难想到以下的几种操作方案：

- 将第 0 类格子上的 X 都改成 O，将第 1 类格子上的 O 都改成 X。
- 将第 1 类格子上的 X 都改成 O，将第 2 类格子上的 O 都改成 X。
- 将第 2 类格子上的 X 都改成 O，将第 0 类格子上的 O 都改成 X。

¹<https://codeforces.com/contest/1450/problem/C2>

显然这三种操作方案都能使得局面变成平局，而它们的操作次数的总和恰好是棋盘中标志的总数 k ，因此其中操作次数最少的方案的操作次数一定不超过 $\lfloor \frac{k}{3} \rfloor$ 。

2.2 Mine Sweeper II²

2.2.1 题目大意

扫雷地图是一张 n 行 m 列的网格，其中每个格子是地雷或空地。每个空地会显示一个数字代表与它相邻的雷的数量（两个格子相邻当且仅当它们共用一个顶点或一条边，不在边界上的格子与恰好 8 个格子相邻）。

在一次操作中，你可以将一个地雷改成空地，或将空地改成地雷。










给定两张扫雷地图 A , B ，你需要对 A 进行不超过 $\lfloor \frac{nm}{2} \rfloor$ 次操作，使得 A 所有空地上的数字之和等于 B 所有空地上的数字之和。

2.2.2 数据范围

$$1 \leq n, m \leq 1000。$$

2.2.3 解题过程

注意到一张地图所有空地上的数字之和等于相邻的（地雷，空地）的对数。这就意味着，如果将一张地图的所有地雷改成空地，所有空地改成地雷，其所有空地上的数字和不变。如下图所示。

	4		2		2
	6		2		3
		2	1	2	

由此，显然有以下两种方案：

- 将 A 改成 B 。
- 将 A 改成与 B 恰好相反，即若 B 的某个格子是地雷，则 A 对应的格子是空地，反之亦然。

²The 2020 ICPC Asia Shanghai Regional Contest, Problem B, <https://codeforces.com/gym/102900/problem/B>

由于每个格子只会在恰好一种方案中被修改，这两种方案的操作次数之和应为 nm 。因此取其中较少的一种，操作次数不超过 $\lfloor \frac{nm}{2} \rfloor$ 。

3 DFS 树

在解决一些图上的构造问题时，DFS 树往往有非常大的帮助。

一张图的 **DFS 树**是在对其进行深度优先遍历时，所形成的树结构。建立了 DFS 树后，图上的边可以分成四类：

- **树边**即每个点到其所有孩子结点的边，也即每个点第一次被访问时经过的边。
- **前向边**是每个点到其后代的边，不包括树边。
- **后向边**是每个点到其祖先的边。
- 其余边称为**横叉边**。

其中，前向边、后向边、横叉边统称为**非树边**。

在构造题中，通常我们用到的是无向图的 DFS 树。如果我们将每条边按照第一次经过时的方向进行定向，则无向图的 DFS 树满足所有非树边都是后向边。这个性质在解题过程中有非常大的作用。

3.1 Ehab's Last Corollary³

3.1.1 题目大意

给定一张 n 个点 m 条边的无向连通图，以及一个整数 k ，你需要

- 找到一个恰好 $\lfloor \frac{k}{2} \rfloor$ 个点的独立集，
- 或者找到一个长度不超过 k 的简单环。

3.1.2 数据范围

$$3 \leq k \leq n \leq 10^5, n - 1 \leq m \leq 2 \cdot 10^5。$$

³<https://codeforces.com/contest/1364/problem/D>

3.1.3 解题过程

记 $l = \lceil \frac{k}{2} \rceil$ 。

建出图的 DFS 树, 考虑每条非树边 (u, v) (正如上文所说, 它一定是后向边), 如果 $|\text{dep}_u - \text{dep}_v| < k$, 则取 (u, v) 加上 v 到 u 的树上路径即为一个长度不超过 k 的简单环。

否则, 考虑两种情况:

- 若 $m = n - 1$, 即图是一棵树。把所有点按照深度的奇偶性分成两个集合, 取其中较大的一个集合, 即为大小至少为 $\lceil \frac{n}{2} \rceil \geq l$ 的独立集, 取其中任意 l 个点即为所求。
- 若 $m > n - 1$, 这时 DFS 树上存在非树边, 但不满足 $|\text{dep}_u - \text{dep}_v| < k$, 意味着 DFS 树的深度至少为 k , 且任意一对深度差在 $[2, k)$ 中的点都不存在边相连。设深度最大的点为 x , 取 x 以及 x 的 $2, 4, \dots, 2l - 2$ 级祖先, 即为所求的独立集。

至此, 我们仅用一次 DFS, 在 $O(n + m)$ 的时间内解决了此题。

3.2 景点划分⁴

3.2.1 题目大意

给定一张 n 个点 m 条边的无向连通图, 以及三个整数 a, b, c , 满足 $a + b + c = n$ 。你需要将 n 个顶点分成三个集合 A, B, C , 大小分别为 a, b, c , 使得其中至少两个集合是连通的 (集合中的任意两个点能只经过该集合内的点互相到达)。有可能无解。

3.2.2 数据范围

$$3 \leq n \leq 10^5, 2 \leq m \leq 2 \cdot 10^5。$$

3.2.3 解题过程

不妨设 $a \leq b \leq c$, 则只需要集合 A, B 连通即可。假设是 A, C 连通, 我们可以通过将 C 中的一些顶点加入 B 中, 使得 C 仍然连通且大小变成 b , 因此仍然是合法的解。 B, C 连通的情况同理。

这时我们遇到了一些困难, 因为有可能无解, 而题目中并未给出、我们也并未发现有解的条件, 非常难以下手。因此我们不妨先来考察图是一棵树的情况。

当图是一棵树时, 集合 A, B 都是其中的子树, 因此一定存在一条边, 使得 A, B 处于边的两侧。显然, 我们只需要找到一条边, 使得其两侧的较小和较大的子树大小分别不小于 a ,

⁴IOI2019 第一试第二题, <https://loj.ac/p/3176>

b 即可。注意到一条边两侧较大的子树一定包含重心，我们可以考虑对重心进行一些分析。如果删去重心后最大的连通块大小小于 a ，则显然无解。否则，设这个连通块的大小为 x ，由重心的性质显然有 $x \leq n/2$ ，因此删去这棵子树后还剩 $n - x \geq n/2$ 个点。又由于 $b \leq n/2$ (因为 $b \leq c$)，因此这棵子树与重心之间的边就是我们要找的边。

回到一般的情况，我们建立图的 DFS 树。找到 DFS 树的重心，设为 u ，记 u 上方的子树为 T ， u 下方的子树为 S_1, S_2, \dots, S_k 。考虑几种情况：

- 如果 T 或某个 S_i 的大小不小于 a ，则我们可以用和树一样的方法构造一组解。
- 如果 T 和所有 S_i 的大小都小于 a ，我们就需要考虑无向图 DFS 树的性质。不同的 S_i 之间是没有边相连的，同时有一些 S_i 与 T 相连。如果所有与 T 相连的 S_i 加上 T 的大小之和小于 a ，则一定是无解的，因为这表示集合 A, B 都必须包含重心 u 。从 T 开始，我们依次加入与 T 相连的 S_i ，直到其大小不小于 a 。设得到的点集为 X ，则 X 是连通的，我们可以在其中选出 A 。同时由于 T 和所有 S_i 的大小之和都小于 a ， X 的大小不超过 $2a$ 。而 $2a + b \leq a + b + c = n$ ，因此我们在删除 X 之后，剩余的点数至少为 $n - 2a \geq b$ ，我们可以在其中选出集合 B 。

至此，我们在 $O(n + m)$ 的时间内完成了构造。

4 递归法

在一些构造题中，对于不同的输入，问题的结构有很大的相似性。在很多时候，这往往意味着我们的构造也具有很大的相似性，或是具有周期性。这时，我们往往可以通过递归的方式，对子问题进行构造，并在子问题的构造的基础上进行一些小的调整，来得到原问题的构造。

需要指出的是，递归可以作为一种思想，但在实际解题过程中可能有代码、时空复杂度高的缺点，需要选手灵活运用。

4.1 Baggage⁵

4.1.1 题目大意

有 $2n$ 个包裹，其中有 n 个 A 类包裹，和 n 个 B 类包裹，初始时它们的排列如下：

B A B A B A ... B A

⁵2014 ACM-ICPC World Finals, Problem A, <https://codeforces.com/gym/101221/problem/A>

这些包裹占据了编号为 1 到 $2n$ 的格子, 同时还有编号为 $-2n+1$ 到 0 的 $2n$ 个空格子可供使用。现在要将这些包裹重新排列, 使得它们形如

$$A A \dots A B \dots B B$$

即, 这些包裹占据了相邻的 $2n$ 个格子 (不一定是 1 到 $2n$), 且所有的 A 类包裹在所有的 B 类包裹的左边。

排列过程由若干次操作组成, 在每一次操作中, 可以选择相邻的两个包裹 (不能只选择一个), 并将它们移动至某两个相邻的空格中。

给定 n , 找到一个最短的操作序列。

4.1.2 数据范围

$$3 \leq n \leq 100。$$

4.1.3 解题过程

与通常的构造题不同, 本题要求的是最短的操作序列, 看起来难以下手。但经过一些尝试, 或是对 n 较小的情况进行搜索, 会发现它们的最短操作序列的长度都是 n 。

事实上, 证明操作次数不少于 n 是容易的: 考虑有多少对相邻的包裹的类型相同, 设这个个数为 d 。初始时 $d = 0$, 而在结束局面中 $d = 2n - 2$ 。在一次操作过程中, 取出包裹时不会 d 不会增加, 而在放回包裹时, 假设放的位置是 t 和 $t + 1$, 则只可能增加 $(t - 1, t)$ 和 $(t + 1, t + 2)$ 这两对相邻的包裹。同时, 容易发现第一次操作至多使得 d 增加 1, 因此总的操作次数不少于 $1 + \lceil (2n - 3)/2 \rceil = n$ 。

接下来, 我们就要尝试对所有 n 构造长度为 n 的操作序列。对于 n 较小 ($n \leq 7$) 的情况, 我们可以直接利用搜索求出操作序列。经过观察发现, 在 $n > 3$ 的情形中, 我们都是将这些包裹从编号为 1 到 $2n$ 的格子移至编号为 -1 到 $2n - 2$ 的格子。

因此, 我们可以定义函数 $\text{solve}(n, x)$ 表示将包裹从编号为 $x+1$ 到 $x+2n$ 的格子 (这些包裹形如 $B A B A \dots B A$) 移至编号 $x-1$ 到 $x+2n-2$ 的格子, 并排列成形如 $A A \dots A B \dots B B$ 。

通过尝试, 或是对 n 稍大一些的情形的观察, 对于 $n \geq 8$ 的情况, 我们可以构造出如下的操作序列 (其中 $_$ 表示空格子):

$$\begin{aligned} _ _ B A B A B A B A \dots B A B A B A \\ A B B A B A B A B A \dots B A B _ _ A \\ A B B A _ _ B A B A \dots B A B B A A \end{aligned}$$

在这里, 我们发现最后一行的红色部分正好符合 solve 函数的输入, 因此我们调用 $\text{solve}(n-4, x+4)$ 对其进行递归求解。

A B B A A A A ... B B B _ _ B B A A
 A _ _ A A A A ... B B B B B B A A
 A A A A A A A ... B B B B B B _ _

至此，我们便完成了长度为 n 的操作序列的构造，解决了此题。

4.2 Strange Housing⁶

4.2.1 题目大意

给定一张 n 个点 m 条边的无向图，你需要选择一个点集 S ，满足：

- 一条边 (u, v) 是开启的当且仅当 $u \in S$ 或 $v \in S$ ，则任意一对点都能只经过开启的边互相到达。
- 不存在一条边 (u, v) 满足 $u \in S$ 且 $v \in S$ 。

有可能无解。

4.2.2 数据范围

$$2 \leq n \leq 3 \cdot 10^5, 0 \leq m \leq 3 \cdot 10^5。$$

4.2.3 解题过程

显然如果原图不连通则一定无解。

我们不妨猜测当原图连通时一定有解，考虑归纳证明。当 $n = 1$ 时，显然 \emptyset 是合法的解。假设 $n = k - 1$ 时一定有解，考虑 $n = k$ 的情况。我们考虑删除一个非割点的点，容易发现这样的点是一定存在的。设这个点是 v ，则 $G \setminus \{v\}$ 的点数为 $k - 1$ ，且是连通图，由归纳假设，我们可以为其找到一组解 S' 。接下来，考虑两种情况：

- 若 G 中至少有一个与 v 相邻的点属于 S' ，则令 $S = S'$ 即为一组合法的解。
- 否则， G 中与 v 相邻的所有点都不属于 S' ，则令 $S = S' \cup \{v\}$ 即为一组合法的解。这是因为 G 是连通图，因此至少有一个点与 v 相邻。

至此，我们证明了有解当且仅当给定的图是连通图。然而，每次寻找一个非割点的复杂度太高，不能承受。事实上，观察归纳的过程，我们只需按照任意一个 DFS 序依次加入点即可。时间复杂度为 $O(n + m)$ 。

⁶<https://codeforces.com/contest/1470/problem/D>

5 总结

构造题的考察越来越频繁，但对许多选手来说，解决一道构造题并不容易。本文介绍了笔者在解题过程中总结的几个较常用的解题思路，希望能够让大家有所启发。

本文介绍的仅仅是构造题中的冰山一角，希望读者在训练过程中，也对构造题的解法进行归纳、总结，并分享给大家。同时也希望更多有趣的构造题能出现在算法竞赛中。

致谢

感谢中国计算机学会提供学习与交流的机会。

感谢国家集训队教练高闻远的指导与帮助。

感谢父母对我多年来的培养与关心。

感谢西南大学附属中学校潘玉斌教练的指导与帮助。

感谢给予我鼓励与帮助的老师以及同学。

信息学竞赛中的生成函数计算理论框架

北京大学附属中学 李白天

摘要

生成函数近年来在信息学竞赛的组合计数问题中扮演着越加重要的角色。其用途在问题处理时的公式推导，以及代码实现中的学问均已逐渐普及。本文旨在给出一套生成函数在信息学竞赛实战中处理问题的方法框架，同时梳理一些其中的重要算法。

1 概述

当我们得到一个组合计数问题相关的生成函数解后，其代数形式到答案计算的算法的转化过程仍有若干壁垒。究其原因，我们在进行计算时常常受到以下几个复杂对象的制约。

1.1 复合

相对于卷积，我们在形式幂级数问题上没有非常高效的处理一般复合问题的方法。多项式模复合问题在OI中已经被引入的复杂度最优的算法为 $\Theta((n \log n)^{3/2})$ ，且该算法具有较大的常数，在实战中往往 $\Theta(n^2)$ 次运算的一种“分块FFT”算法具有更好的表现。值得一提的是，该问题在理论界已有 $O(n^{1+o(1)})$ 的算法（见[1]），但尚不清楚在实现上是否优越。

1.2 生成函数方程

当组合对象的描述中进行自指时，生成函数方程便自然产生。在方程中种种运算的复合会使得计算更为困难。例如组合对象的笛卡尔积导出生成函数的乘积 $A(x) \times B(x)$ ，结构的复合导出生成函数的复合 $A(B(x))$ ，EGF中对位置的固定导出生成函数的微分 $A'(x)$ ，Pölya计数定理导出下标变换 $A(x^k)$ 等。

1.3 远处系数求值

远处系数求值意谓在提取第 n 次项系数时，往往 $\Omega(n)$ 的计算量都是不被允许的。而能在 $o(n)$ 时间内完成计算的问题要求确实比较苛刻，但目前仍有一些可探讨的空间。目前较为成熟的两类远处系数求值是线性递推数列和整式递推数列。

定理 1.1. 给定数列 a_n 的 $0 \sim k-1$ 项和定义在其上的线性递推式 b_1, \dots, b_k , 满足

$$a_n = \sum_{j=1}^k a_{n-j} b_j$$

存在算法在 $\Theta(k \log k \log n)$ 时间内计算 a_n 。

现在较为普及的线性递推算法由 Fiduccia 于 1985 年提出, 并不具有较好的常数。后文将介绍一种算法上更为简单, 且具有较好常数的线性递推算法。

定理 1.2. 给定数列 a_n 的 $0 \sim m-1$ 项和定义在其上的整式递推式 P_0, \dots, P_m , 满足

$$a_n = -\frac{1}{P_0(n)} \sum_{j=1}^m a_{n-j} P_j(n)$$

存在算法在 $\Theta(\sqrt{nd}(m^3 + m^2 \log(nd)))$ 时间内计算 a_n 。

1.4 线性变换

在对序列进行的线性变换中, 有一类变换是极常见的, 其变换可以用多项式的表示方法描述。围绕多项式的单项式系数表示和下降幂系数表示, 下列等式

$$f(x) = \sum_{n \leq N} a_n x^n = \sum_{n \leq N} b_n x(x-1)\dots(x-n+1)$$

中, 序列 a 与 b 的相互转化目前有 $\Theta(n \log^2 n)$ 的复杂度这一壁垒。但在各自的表示之下, 二者在进行其他运算时却常常具有不同的优势, 各自在一类问题上具有 $\Theta(n \log n)$ 或者 $\Theta(n)$ 的复杂度。

此外, 在处理线性变换时, 转置原理可以帮助我们简化问题。

转置原理 [2] 指出, 在考虑一类可表示为对输入向量 \mathbf{x} 进行矩阵乘法 $\mathbf{M}\mathbf{x}$ 的计算时, 可考虑先对 $\mathbf{M}^T \mathbf{y}$ 的计算设计算法, 然后将算法的各步骤转置改写。

2 算法的评定

2.1 系数

在信息学竞赛的组合计数问题中, 由于问题常常要求输出对某个数取模的结果, 而这一计算方式往往会影响算法的使用。其对计算复杂程度的影响大致可分为以下四级。

$$\mathbb{F}_{\text{NTT}} \subset \mathbb{F}_p \subset \text{GF}(p^k) \subset \text{Commutative Rings}$$

NTT 模数域 素数域 有限域 交换环

NTT 模数域 设取模的素数 p 满足 $p = m \times 2^k + 1$, 当 k 较大时, 由于 2^k 次单位根的存在, 快速数论变换可以在 $\Theta(n \log n)$ 时间内进行长度为 2 的幂, 且最长为 2^k 的数论意义下 DFT 结果。由此即可对于 $n \leq 2^k$, 在 $\Theta(n \log n)$ 的时间内完成两个 n 次多项式的乘法。由于其 DFT 的存在, 在这种模数下的算法往往有很大的, 根据 DFT 中间结果进行针对性常数优化的空间。

- **模998244353**: NTT 模数中最为著名的非 $\mathbb{F}_{998244353}$ 莫属, 有 $998244353 = 119 \times 2^{23} + 1$ 。

素数域 在题目中要求答案同余一个素数也较为常见, 其中又有大多数情况 p 很大 (例如 $10^9 + 7$), 通常是为了保证算法中可以较自由地进行除法。

有限域 对于一个 \mathbb{F}_p 上的 n 阶不可约多项式 f , $\mathbb{F}_p[x]/f$ 为一 p^n 阶有限域, 记为 $\text{GF}(p^n)$ 。

- **Nimber**: Nimber 是一种可以较为高效计算的有限域, 其出现时往往为 $\text{GF}(2^{32})$ 或 $\text{GF}(2^{64})$, 与计算机的无符号整数 `uint32` 或 `uint64` 交相呼应。
- **二次扩域**: 题目中较为常见的是二次扩域, 实际上就是对于模 p 下的二次非剩余 r , 模拟 $a + b\sqrt{r}$ 的运算。这实际上就等价于 $\mathbb{F}_p[x]/(x^2 - r)$, 由 r 是二次非剩余可知 $x^2 - r$ 是不可约多项式, 故前式是域。

交换环 交换环中不一定有逆, 因此并不总能进行除法。

- **模 n 运算**: $\mathbb{Z}/n\mathbb{Z}$ 在 n 是合数时不构成域。
- **多项式**: 若多项式的元素仅构成交换环, 或者运算时对一个可约多项式取模都会导致运算不构成域, 只构成交换环。

杂项 还有部分性质极差的运算, 它们并非本文想要讨论的重点, 仅简单提及。

- **矩阵环**: 矩阵的乘法是信息学竞赛中最常见的非交换运算。由于其非交换性, 在诸如生成函数乘法逆的计算时需要注意运算顺序, 而诸如生成函数 `exp` 等运算并不具有很好的性质, 尚未有复杂度优秀的算法。
- **最值与最值计数**: 最值与最值计数仅构成交换半环, 由于目前 `max-plus` 卷积尚未有 $O(n^{2-\epsilon})$ 的算法, 限于其算术性质与本文相离太远, 故仅在此提及。

在接下来的表述中, 我们默认计算时的系数至少为交换环, 记为 \mathbb{A} 。

2.2 序列变换

2.2.1 卷积

定义 2.1 (卷积下标系统). 我们称满足以下几条性质的集合 I 以及定义在其上的运算 \circ 构成下文要讨论的卷积下标系统:

- **结合律**: 即 $\forall i, j, k \in I$, 有 $(i \circ j) \circ k = i \circ (j \circ k)$ 。

- 交换律: 对于 $\forall i, j \in I$, 有 $i \circ j = j \circ i$ 。
- 单位元: 存在元素 $1 \in I$, 对于 $\forall i \in I$, 有 $1 \circ i = i$ 。
- 零元: 存在元素 $0 \in I$, 对于 $\forall i \in I$, 有 $0 \circ i = 0$ 。
- 后效律: 对于 $\forall i \in I \setminus \{0\}$, 对任意正整数 k 和 $j_1, \dots, j_k \in I \setminus \{0, 1\}$ 有 $i \circ j_1 \circ \dots \circ j_k \neq i$ 。

下文中有歧义时则用 $0_I, 1_I$ 表示 $0, 1$ 。在讨论卷积下标系统时, 默认以 I 表示 (I, \circ) 。在讨论算法时, 我们默认 I 是有限的。

定义 2.2 (生成函数). 对于以卷积下标系统 I 为下标的序列 $\{f_i \in \mathbb{A}\}_{i \in I \setminus \{0\}}$, 我们定义其生成函数 $F(X) = \sum_{i \in I} f_i X^i$, 我们记 X 为“未定元”, 满足:

- 对 $\forall a \in \mathbb{A}, i \in I$, 有 $aX^i = X^i a$ 。
- 对 $\forall i, j \in I$, 有 $X^i \cdot X^j = X^{i \circ j}$ 。
- 消去律: $X^{0_I} = 0_{\mathbb{A}}$, 即 X^{0_I} 作为生成函数对应的序列为 $\{f_i = 0_{\mathbb{A}}\}_{i \in I \setminus \{0\}}$ 。
- $X^{1_I} = 1_{\mathbb{A}}$ 。

容易看出, 前文所定义的零元刻画了在计算时溢出的部分。

我们记这样的全体生成函数 $F(X)$ 构成的环为 $\mathbb{G}[X]$ 。

我们所选取的生成函数进行乘法时, 未定元指标的运算自然地导出了对应系数序列的卷积。

定义 2.3 (卷积). 对于以卷积下标系统 I 为下标, 以 \mathbb{A} 为系数的序列, 我们定义其卷积 $c = a * b$:

$$c_k = \sum_{i \circ j = k} a_i b_j$$

定义 2.4 (截取). 称 I' 是 I 的一个截取, 当且仅当 $\{0, 1\} \subseteq I' \subseteq I$ 且 $\forall i \in I \setminus I', \forall j \in I, i \circ j \notin I' \setminus \{0\}$ 。从而定义运算 $\circ_{I'}$ 为

$$i \circ_{I'} j = \begin{cases} i \circ j & i \circ j \in I' \\ 0 & \text{else} \end{cases}$$

形象地看, 从 I 中截取的 I' 即为删去了 I 中的一部分「边界」, 仅保留剩余部分的信息。

2.2.2 在线算法

通过前文的铺垫，我们接下来叙述一个静态的算法与其在线形式的关联。

定义 2.5 (后效序). 定义 I 上的序关系 \leq : $i \leq j$ 当且仅当存在 $k \in I$, 使得 $i \circ k = j$. 我们称这一序关系为后效序。

引理 2.1. I 上的后效序构成偏序。且以 1 为最小元，以 0 为最大元。

读者不难根据偏序集的定义逐条验证此引理。

定义 2.6 (后效变换). 对于变换 $\Psi: \underbrace{\mathbb{A}^{\setminus\{0\}} \times \dots \times \mathbb{A}^{\setminus\{0\}}}_{n \uparrow} \rightarrow \mathbb{A}^{\setminus\{0\}}$, 我们称其为后效的, 当且仅当对于 $\forall i \in \setminus\{0\}$, 令 $J = \{j \mid j \leq i\}$, 任取 $a^{(1)}, \dots, a^{(n)}$, 记 $\psi = \Psi[a^{(1)}, \dots, a^{(n)}]$, 改变任何一者在 $\setminus J$ 中的取值, 都不会改变 ψ_i 。

引理 2.2. 卷积变换是后效的。

定义 2.7 (多项式左复合). 我们总是可以定义多项式与一个生成函数的复合。记多项式 $F(x) = \sum_{k=0}^n f_k x^k$ 和序列 g 对应的生成函数 $g(X)$ 。则复合 $h = f \circ g$ 定义为

$$h(X) = \sum_{k=0}^n f_k g(X)^k$$

引理 2.3. 多项式的左复合关于 g 是后效的。

定义 2.8 (在线算法). 对于一具有 n 个输入的后效变换 Ψ 以及对于一个保后效序的优先级 $\delta: \setminus\{0\} \rightarrow \mathbb{Z}_{\geq 0}$, 满足 $i < j \Rightarrow \delta(i) < \delta(j)$, 我们称一个算法是在线计算 Ψ 的当其能配合黑盒完成以下流程:

1. 令 $a^{(1)}, \dots, a^{(n)}$ 所有项均为 0。
2. 将 k 从 0 循环至最大的 δ 值, 记 $J_k = \{j \mid \delta(j) = k\}$ 。设此时 $\psi^{(k)} = \Psi[a^{(1)}, \dots, a^{(n)}]$, 对于全体 $j \in J_k$, 将 $\psi_j^{(k)}$ 汇报给黑盒。 J_k 中所有元素汇报结束后, 黑盒对于所有 $1 \leq i \leq n, j \in J_k$, 将 $a_j^{(i)}$ 赋值。

接下来, 我们不难注意到:

引理 2.4. 设问题规模为 n , 由一系列分别可在 $T_1(n), \dots, T_k(n)$ 时间内完成在线算法的后效变换 Ψ_1, \dots, Ψ_k 构成的表达式树, 不妨将其整体看做一个变换 Φ , 该变换是后效的, 且存在 $T_1(n) + \dots + T_k(n)$ 时间的在线算法。

2.3 线性算子

定义 2.9 (微分型算子). 对于生成函数环 $\mathbb{G}[X]$, 我们称作用在其上的一个线性算子 \mathfrak{D} 是微分型的, 当且仅当其满足条件 $\forall f, g \in \mathbb{G}[X] \Rightarrow \mathfrak{D}(fg) = f\mathfrak{D}g + g\mathfrak{D}f$.

这与我们常见的导数算子是符合的, 由此我们容易得到以下推论:

引理 2.5. 对于一个多项式 f 和一个 $\mathbb{G}[X]$ 上的生成函数 g , 有

$$\mathfrak{D}(f \circ g) = (f' \circ g)\mathfrak{D}g$$

引理 2.6. 对于微分型算子 \mathfrak{D}_1 和 \mathfrak{D}_2 , 其线性组合 $u\mathfrak{D}_1 + v\mathfrak{D}_2$ 也是微分型算子, 其中 $u, v \in \mathbb{G}[X]$.

而除了微分型算子, 还有以下两类算子是容易见到的。

定义 2.10 (下标变换算子). 对于 $n \in \mathbb{Z}_{\geq 0}$, 我们称作用在生成函数上的线性算子 \mathfrak{S}_n 满足 $\mathfrak{S}_n X^j = X^{nj} = X^{\underbrace{j \circ \cdots \circ j}_{n \uparrow}}$, 全体 \mathfrak{S}_n 构成下标变换算子。

这类算子广泛出现于 Pölya 计数中。

定义 2.11 (点乘算子). 对于数列 $\{a_i\}_{i \in \mathbb{N} \setminus \{0\}}$, 对应的线性算子 \mathfrak{d}_a 满足 $\mathfrak{d}_a X^i = a_i X^i$, 我们称之为点乘算子。

2.4 方程求解

经过前文的铺垫, 至此解出方程所有项系数的通用方法已经是图穷匕见了, 其两大主要方法为在线方法和牛顿迭代法。

在线方法 在线方法意为将方程计算的每一步运算都找到一个可以高效执行的在线算法, 且具有同一步调 (即定义 2.8 描述的 δ)。如此一来, 便可以将解方程转化为一个逐步进行的方程验证问题, 只需通过在线算法逐步验算出各项, 然后计算出使方程在该项系数满足时, 取到的「修正值」。

在线算法中最重要的对象无外乎为在线卷积, 而多项式的左复合通常需要通过其本身满足的微分方程, 将复合问题本身转化为一个带有微分型算子的解方程问题。

牛顿迭代法 牛顿迭代法主要通过对下标系统 I 的截取来分步降低问题的求解难度。对于涉及多项式左复合 $F(G)$ 的方程, 设 I 截取的部分 I' 已经算出答案, 对应为 G_0 , 则考虑将 $F(G)$ 在 G_0 处做 Taylor 展开, 即 $F(G) = \sum_{n \geq 0} \frac{F^{(n)}(G_0)}{n!} (G - G_0)^n$, 由于 $G - G_0$ 已知在 I' 下标处系数为 0, 往往能使得 $(G - G_0)^n$ 很快为 0, 通常会通过设置 I' 使得 $n \geq 2$ 时的项均消除, 便有 $F(G) = F(G_0) + F'(G_0)(G - G_0)$, 变成易于处理的线性情况。

牛顿迭代法在很多情况下较在线方法的复杂度略胜一筹，在部分简洁问题上亦表现出色。但实践中其表现有时不够令人满意，原因在于在解较为复杂的方程时，牛顿迭代过程会有若干子问题层层嵌套，虽在复杂度上没有影响，却会使得常数逐轮翻倍。

以我们熟悉的一元幂级数 \exp 的计算为例，采取诸如 [3] 中介绍的非常细致的优化可以做到约等于 $2\frac{7}{12}$ 次同长度多项式乘法的时间，但粗暴的实现可能会消耗 $7\frac{1}{2}$ 次同长度多项式乘法的时间甚至更大。在 OI 的实践范围内，基于半在线卷积的常见 $\Theta\left(\frac{n \log^2 n}{\log \log n}\right)$ 实现反而具有更高的效率。

3 普通生成函数

定义 3.1 (普通生成函数). 令 $I = \mathbb{Z} \cap [0, n] \cup \{0_I\}$, 且对于 $i, j \in I \setminus \{0_I\}$, 定义

$$i \circ j = \begin{cases} i + j & i + j \leq n \\ 0_I & \text{else} \end{cases}$$

定义在 (I, \circ) 上的生成函数即为我们计算时取的普通生成函数 (OGF)。

可见，这实际上就是对于形式幂级数 $\text{mod } x^{n+1}$ 。

3.1 卷积

定理 3.1 (快速 Fourier 变换). 对于形如模 M 运算的环，可以在 $\Theta(n \log n)$ 次运算内完成卷积的计算。

最常见的情况为 M 是 NTT 模数。而对一般的 M ，常使用三模数 NTT 或者拆系数 FFT 在较大常数的 $\Theta(n \log n)$ 内完成计算。

下文中，我们以 $M(n)$ 表示多项式乘法的复杂度。

3.1.1 半在线卷积

定义 3.2 (半在线卷积). 在计算序列 a, b 的卷积时，若算法仅关于序列 b 是在线的，序列 a 初始已经完全确定，那么称这一算法计算了半在线卷积。记其复杂度为 $S(n)$ 。

定理 3.2. 目前复杂度最好的半在线卷积经 [4] 分析为

$$S(n) = O\left(n \log n e^{2\sqrt{\log 2 \log \log n}}\right)$$

注意到对于 $\forall \epsilon > 0$, 都有 $S(n) = o\left(n \log^{1+\epsilon} n\right)$, 因此在理论上半在线卷积与卷积的复杂度是极为接近的。而在实际应用中较易实现的算法实为 $\Theta\left(\frac{n \log^2 n}{\log \log n}\right)$ 。在 [5] 中已有介绍，不予赘述。

定理 3.3. 在线卷积与半在线卷积等难。即记在线卷积的复杂度为 $R(n)$ ，那么有

$$R(n) = \Theta(S(n))$$

首先半在线卷积显然可以规约为在线卷积，接下来进行逆向规约。我们通过倍增来构造一种 $\Theta(S(n))$ 计算在线卷积的算法：

1. 令 $m = \lfloor n/2 \rfloor$ ，首先进行 $0 \sim m$ 下标部分的在线卷积。
2. 接下来考虑 $k > m$ 的部分，考虑卷积式 $c_k = \sum_{i=0}^k a_i b_{k-i}$ ，注意到 $\min(i, k-i) \leq \frac{k}{2}$ ，因此 $i, k-i$ 中必有一者 $\leq m$ 。
3. 因此考虑将剩下的计算分为三部分：
 - $i, j \leq m$ ，这部分可以立刻 $M(m)$ 算出。
 - $i \leq m, j > m$ ，这实际上是 $a_{0 \sim m}$ 与 $b_{m+1 \sim n}$ 部分的半在线卷积。
 - $j \leq m, i > m$ ，这实际上是 $b_{0 \sim m}$ 与 $a_{m+1 \sim n}$ 部分的半在线卷积。
4. 因此算法只需倍增地进行半在线卷积，有递归式 $R(n) = R(n/2) + \Theta(S(n))$ 。
5. 根据主定理，有 $R(n) = \Theta(S(n))$ 。

3.2 方程求解

3.2.1 牛顿迭代法

在一元生成函数中，只需截取 $I' = [0, \lfloor n/2 \rfloor] \cap \mathbb{Z} \cup 0_I$ ，即先计算出 $\text{mod } x^{\lfloor n/2 \rfloor + 1}$ 的结果，即可进行通常意义的牛顿迭代。

此外，这样的截取亦消解了下标变换算子带来的影响。对于 $\mathfrak{S}_k F(x) = F(x^k)$ ，若 $k = 1$ 则其就是 $F(x)$ ，否则 $F(x^k)$ 在 $0 \sim n$ 项的系数仅基于 $F(x)$ 在 $0 \sim \lfloor n/k \rfloor$ 项的系数，而它们已经完全确定，迭代过程中总是可以当做已经确定的常量。

3.2.2 在线方法

在线方法相较牛顿迭代法自由之处在于可以处理较为复杂的形式幂复合问题。简而言之，形式幂 $F(x)$ 若满足一个关于 $F, F', \dots, F^{(m)}$ 的多项式方程 $P(x, F, F', \dots, F^{(m)}) = 0$ ，可以基于这一方程，得到 $H = F \circ G$ 的一个多项式方程 $Q(H, H', \dots, H^{(m)}, G^{(0)}, \dots, G^{(m)}) = 0$ ，进而通过该方程得到的递推式在线求解 H 。具体地， Q 可以通过如下方式给出：

1. 依定义，有 $F(G(x)) = H(x)$ 。

2. 由 $H'(x) = F'(G(x))G'(x)$, 可将 $F'(G(x))$ 表示为 $\frac{H'}{G'(x)}$ 。
3. 由 $H''(x) = F''(G(x))G'^2(x) + F'(G(x))G''(x)$, 可将 $F''(G(x))$ 用 H' 和 H'' 表示。
4. 逐次递推, 可将 $F^{(k)}(G(x))$ 用 $H, H', \dots, H^{(k)}$ 表示。
5. 因此可将方程 P 直接带入 $G(x)$, 则有 $P(x, F(G(x)), F'(G(x)), \dots, F^{(m)}(G(x))) = 0$, 将各项均替换为由 H 表达的式子之后通分, 即得方程 Q 。

3.2.3 Lagrange 反演

对于一元生成函数来说, Lagrange 反演有效地将一个生成函数与其复合逆的系数建立起了联系。为体现形式的优美性, 我们需在形式 Laurent 级数下描述 Lagrange 反演。本文中只展示一个在特征为 0 的域上的证明方法, 在一般环上的通用证明可参看 [8, Sec. 1.2]。

定义 3.3 (形式 Laurent 级数). 记域 K 上的形式 Laurent 级数为 $K((x))$ 或 $K[[x]][x^{-1}]$, 即对于 $f(x) \in K((x))$, 若 $f \neq 0$, 则存在数列 $\{a_n\}_{n \geq n_0}$, 有

$$f(x) = x^{n_0} \left(\sum_{n \geq 0} a_{n+n_0} x^n \right)$$

其中有 $a_{n_0} \neq 0$ 。

此时对于 $k \in \mathbb{Z}$, 容易定义在 $K((x))$ 内的幂 f^k :

$$f(x)^k = x^{n_0 k} \left(\sum_{n \geq 0} a_{n+n_0} x^n \right)^k$$

引理 3.1 (形式留数). 对于幂级数 $F(x)$ 满足 $n_0 = 1$, 那么对于 $\forall k \in \mathbb{Z}$, 有

$$[x^{-1}]F'(x)F^k(x) = [k = -1]$$

证明. 考虑当 $k \neq -1$ 时, 我们有 $F'(x)F^k(x) = (\frac{1}{k+1}F(x)^{k+1})'$, 而 $(x^0)' = 0x^{-1}$, 故 -1 次项系数此时必然为 0。当 $k = -1$ 时设 $F(x) = a_1x + a_2x^2 + \dots$, 有

$$\begin{aligned} \frac{F'(x)}{F(x)} &= \frac{a_1 + 2a_2x + \dots}{a_1x + a_2x^2 + \dots} \\ &= x^{-1} \frac{1 + 2\frac{a_2}{a_1}x + \dots}{1 + \frac{a_2}{a_1}x + \dots} \end{aligned}$$

因此 $k = -1$ 时 -1 次项系数为 1。 □

定理 3.4 (Lagrange 反演). 对于幂级数 $F(x)$ 满足 $n_0 = 1$ 以及 $G(x)$ 满足 $F(G(x)) = x$ 是其复合逆, 那么对于 $n, k \in \mathbb{Z}$, 有

$$n[x^n]F(x)^k = k[x^{-k}]G(x)^{-n}$$

证明. 我们考虑带入复合关系 $F(G(x)) = x$, 有

$$\begin{aligned} F(G(x))^k &= x^k \\ (F^k)'(G)G' &= kx^{k-1} \\ \sum_i i([x^i]F^k(x))G^{i-1}G' &= kx^{k-1} \\ \sum_i i([x^i]F^k(x))G^{i-1-n}G' &= kx^{k-1}G^{-n} \\ [x^{-1}] \sum_i i([x^i]F^k(x))G^{i-1-n}G' &= [x^{-1}]kx^{k-1}G^{-n} \\ n[x^n]F^k &= [x^{-1}]kx^{k-1}G^{-n} \\ n[x^n]F^k &= k[x^{-k}]G^{-n} \end{aligned}$$

故原式得证. □

与这一富有对称性的形式相比, 这一公式在以往更加流行的版本则是一个复合形式。

引理 3.2. 对于幂级数 $H(x) \in K((x))$, 有

$$[x^n]H(F(x)) = \frac{1}{n}[x^{n-1}]H'(x) \left(\frac{x}{G(x)} \right)^n$$

读者只需确认 $H(x) = x^k$ 的情况即可验证, 在此不予赘述。

例题 1 (Slime and Sequences¹). 计算 $[z^n] \frac{t(e^{z(1-t)}-1)}{(1-z)(1-te^{z(1-t)})}$ 的系数同余 998244353, 保证 $1 \leq n \leq 10^5$ 。

解法 考虑 $\left([z^n] \frac{t(e^{z(1-t)}-1)}{(1-z)(1-te^{z(1-t)})} \right) + 1 = (1-t)[z^n] \frac{1}{(1-z)(1-te^{z(1-t)})}$, 用于简化表达式。

接下来我们令 $z = \frac{u}{1-t}$, 就有

$$[z^n] \frac{1}{(1-z)(1-te^{z(1-t)})} = (1-t)^n [u^n] \frac{1}{\left(1 - \frac{u}{1-t}\right)(1-te^u)}$$

接下来做分式分解

$$(1-t)[z^n] \frac{1}{(1-z)(1-te^{z(1-t)})}$$

¹来源: 许庭强与本人共同命制, <http://codeforces.com/problemset/problem/1349/F2>, 已省略得到生成函数的推导过程

$$\begin{aligned}
 &= [u^n] \frac{(1-t)^{n+2}}{\left(1 - \frac{t}{1-u}\right)(1-te^u)(1-u)} \\
 &= (1-t)^{n+2} [u^n] \left(\frac{-e^u}{(e^u u - e^u + 1)(1-te^u)} + \frac{\frac{1}{1-u}}{(e^u u - e^u + 1)\left(1 - \frac{t}{1-u}\right)} \right)
 \end{aligned}$$

由此，问题的关键转化为提取 $[u^n] \frac{-e^u}{(e^u u - e^u + 1)(1-te^u)}$ 部分，我们设 $F(u) = e^u - 1$ ，那么即可将 BGF 表为 $A(F)/(1-t(1+F))$ 的形式，由复合逆 $G = \ln(1+u)$ 可得：

$$\begin{aligned}
 [u^n] \frac{A(F)}{(1-t(1+F))} &= \frac{1}{n} [u^{n-1}] \left(\frac{A(u)}{1-t(1+u)} \right)' \left(\frac{u}{G(u)} \right)^n \\
 &= \frac{1}{n} [u^{n-1}] \left(\frac{tA(u)}{(1-t(1+u))^2} + \frac{A'(u)}{1-t(1+u)} \right) \left(\frac{u}{G(u)} \right)^n
 \end{aligned}$$

由此，关键在于计算 $A(u)$ 的系数表示，由 $A(F(u)) = \frac{-e^u}{(e^u u - e^u + 1)}$ 可知 $A(u) = \frac{-e^G}{(e^G G - e^G + 1)}$ 。至此，问题可以在 $\Theta(n \log n)$ 时间内解决。 \spadesuit

上述常称为的「扩展 Lagrange 反演」虽然已经很强大，但还是有所不便之处：若扩展到一般环 R 上，若 $F(x)$ 的一次项在 R 中可逆，可知 G 还是良定义的，但上述公式会需要 n 在 R 中可逆，不一定能帮我们计算出结果。最极端的情况便是这一公式无法告诉我们如何对于 $n=0, k < 0$ 的情况提取系数。因此，我们有时需要考虑一个变式：

引理 3.3 (另类 Lagrange 反演). 保持原先条件不变，有

$$[x^n] F^k = [x^{-k-1}] G' G^{-n-1}$$

证明. 我们在第一步不是求导，而是乘以 $G' G^{-n-1}$ ，就会有

$$\begin{aligned}
 F(G(x))^k &= x^k \\
 \sum_i ([x^i] F^k(x)) G^i &= x^k \\
 \sum_i ([x^i] F^k(x)) G' G^{i-n-1} &= x^k G' G^{-n-1} \\
 [x^{-1}] \sum_i ([x^i] F^k(x)) G' G^{i-n-1} &= [x^{-1}] x^k G' G^{-n-1} \\
 [x^n] F^k &= [x^{-1}] x^k G' G^{-n-1} \\
 &= [x^{-k-1}] G' G^{-n-1}
 \end{aligned}$$

□

这个形式虽然不再具有足够的对称性，却成功规避了除法。我们也可将其写作复合形式。

引理 3.4. 在上一引理的不同条件下，有

$$[x^n] H(F(x)) = [x^n] H(x) \left(\frac{x}{G(x)} \right)^{n+1} G'(x)$$

其证明依然大同小异，不予赘述。

例题 2 (简单的普及组计数²)。设幂级数 $F(x)$ 满足 $F = \frac{x}{(1-(m-1)F)^{k-1}}$ ，求 $[x^n] \frac{1}{1-mF}$ 同余质数 p ，保证 $1 \leq n, k \leq 10^9, 1 \leq m < p \leq 10^5$ 。

解法 可得 F 的复合逆为 $G = x(1 + (m-1)x)^{k-1}$ ，根据另类 Lagrange 反演，有

$$\begin{aligned} [x^n] \frac{1}{1-mF} &= [x^n] \frac{1}{1-mx} \left(\frac{x}{G(x)} \right)^{n+1} G'(x) \\ &= [x^n] \frac{1}{1-mx} (1-(m-1)x)^{-(k-1)(n+1)} \cdot (1-(m-1)x)^{k-2} (1-(m-1)kx) \\ &= [x^n] \frac{1}{1-mx} (1-(m-1)x)^{-kn+n-1} (1-(m-1)kx) \end{aligned}$$

由于模数很小，我们可以进行一个类似数位 DP 的过程，现在所求的答案形如 $[x^n] \frac{1}{1-mx} (1-(m-1)x)^m f(x)$ ，由 Lucas 定理 $\binom{np+n_0}{mp+m_0} \equiv \binom{n}{m} \binom{n_0}{m_0} \pmod{p}$ 可知，设 $m = m_1p + r$ 和 $n = n_1p + s$ 则有

$$\begin{aligned} & [x^n] \frac{1}{1-mx} (1-(m-1)x)^m f(x) \\ & \equiv [x^n] (1-mx)^{-p+(p-1)} (1-(m-1)x)^{m_1p+r} f(x) \\ & \equiv [x^{n_1p+s}] \frac{1}{1-mx^p} (1-(m-1)x^p)^{m_1} (f(x)(1-mx)^{p-1} (1-(m-1)x)^r) \\ & \equiv [x^{n_1}] \frac{1}{1-mx} (1-(m-1)x)^{m_1} f_1(x) \end{aligned}$$

其中 $f_1(x)$ 是 $f(x)(1-mx)^{p-1}(1-(m-1)x)^r$ 提取同余 p 余 s 的下标系数所得，可以在 $\Theta(p \cdot \deg f)$ 内计算，归纳可知在 $f \leftarrow f_1$ 这一迭代过程中，始终有 $\deg f \leq 1$ 。因此复杂度为单轮复杂度 $\Theta(p)$ 乘以迭代轮数 $\log_p n$ ，即 $\Theta(p \log_p n)$ 。♣

3.3 远处系数求值

3.3.1 线性递推

接下来我们介绍一种线性递推式求值的低位优先算法 (LSB-first, least significant bit first)。

不同于 Fiduccia 直接通过快速幂对于 $x^n \bmod Q(x)$ 的考虑，低位优先算法的主要思想是考虑将线性递推转化为生成函数的形式 $\frac{P(x)}{Q(x)}$ 。注意我们容易在 $M(k)$ 的时间内得到 $P(x)$ ， $(\sum_{i=0}^{k-1} f_i x^i) \cdot Q(x)$ 的 $0 \sim k-1$ 项即为所求。接下来我们考虑如下等式：

$$\frac{P(x)}{Q(x)} = \frac{P(x)Q(-x)}{Q(x)Q(-x)}$$

²来源：戴江齐，加强自 <https://acm.nflsoj.com/problem/308>，已省略得到生成函数的推导过程

我们注意记 $V(x) = Q(x)Q(-x)$, 分析系数可以发现 $V(x)$ 只有偶次项有值, 因此我们就得到了分解

$$\frac{P(x)}{Q(x)} = \frac{E(x^2)}{U(x^2)} + x \frac{O(x^2)}{U(x^2)}, \quad U(x^2) = Q(x)Q(-x)$$

因为这分别填满了二进制的 0 和 1 位, 所以我们只需递归到一侧即可。而 $n \leftarrow \lfloor n/2 \rfloor$ 。

因此每轮都在 $\Theta(M(k))$ 时间内完成计算, 进行 $\log_2 n$ 轮迭代后, 已经有 $n = 0$, 故算法复杂度为 $\Theta(M(k) \log n)$ 。

这一算法还有许多优化的空间, 在我们通常有 NTT 模数的情况下, 同样的 FFT 实现可以让我们的常数大约是原先方法的 $\frac{1}{3}$ 。受篇幅所限, 本文略过其优化细节。

这一过程每次仅用到了多项式乘法, 并不需要实现多项式求逆。总的来说, 本算法具有实现和效率方面的双重优越性。

4 多元幂级数

定义 4.1. 对于 $1 \leq j \leq k$ 设 $\mathfrak{G}_j[x_j]$ 是定义在 $I_j = \mathbb{Z} \cap [0, n_j - 1] \cup \{0_{I_j}\}$ 上的普通生成函数环, 其未定元为 x_j , 那么 $\mathfrak{G}_1[x_1] \times \cdots \times \mathfrak{G}_k[x_k]$ 是 k 元普通生成函数环, 也可写作 $\mathbb{A}[x_1, \dots, x_k]/(x_1^{n_1}, \dots, x_k^{n_k})$ 。接下来记 $n = n_1 \times \cdots \times n_k$ 。

4.1 卷积

若进行朴素的高维 DFT, 则每维都几乎要将数组倍长, 导致 $\Omega(n2^k)$ 的计算量和空间, 在高维情况是尤其不可接受的。

究其原因, DFT 若不将数组倍长, 则其所实际进行的为循环卷积, 下标溢出的部分会污染我们所求的答案。一个较为直接的想法是扩充一维, 用于确认实际的总度数 $\sum_{i=1}^k i_l$ 。如此一来, 我们得到了一个在 n_l 均较小时高效的算法:

插值转化 对于 $1 \leq l \leq k$, 若存在 $\alpha_0^{(l)}, \dots, \alpha_{m-1}^{(l)}$ 使得 $i \neq j \Leftrightarrow \alpha_i^{(l)} - \alpha_j^{(l)}$ 在 \mathbb{A} 中有逆元, 那么令

$$\tilde{F} = \sum_{i_1=0}^{n_1-1} \cdots \sum_{i_k=0}^{n_k-1} f_{i_1, \dots, i_k} x_1^{i_1} \cdots x_k^{i_k} t^{\sum_{l=1}^k i_l}$$

我们称 t 为占位元, 关于 t 的多项式称为占位多项式。称插值变换将 \tilde{F} 转化为 $\{f_{i_1, \dots, i_k}(t)\}$, 有

$$f_{i_1, \dots, i_k}(t) = \tilde{F}(\alpha_{i_1}^{(1)}, \dots, \alpha_{i_k}^{(k)}, t)$$

若需计算 $F \times G$, 只需将其插值转化为 \dot{f}, \dot{g} 后逐点乘, 关于 t 卷积, 再施插值转化的逆变换, 所得 \tilde{H} 满足

$$[x_1^{i_1} \cdots x_k^{i_k}](F \times G) = [x_1^{i_1} \cdots x_k^{i_k} t^{\sum_{l=1}^k i_l}] \tilde{H}$$

若记 $d = \sum_{l=1}^k (n_l - 1)$, 可见上述算法在最好情况下具有 $\Theta(M(n)d + M(d)n)$ 的复杂度, 但其条件苛刻, 且在 d 大时不具有优势。但通过另一种构造, 我们可以得到一种极优的复杂度:

定理 4.1. 高维卷积可在 $\Theta(kM(n))$ 时间内完成。

我们考虑将高维序列编码为非常规的进制数, 即将下标 (i_1, \dots, i_k) 映射到

$$i = i_1 + i_2 \cdot n_1 + \dots + i_k \cdot n_1 \cdots n_{k-1}$$

显见, 原本的下标运算对应于映射后未发生进位的加法。

定义 4.2 (进位占位元). 定义占位函数 χ 为

$$\chi(i) = \left\lfloor \frac{i}{n_1} \right\rfloor + \left\lfloor \frac{i}{n_1 n_2} \right\rfloor + \dots + \left\lfloor \frac{i}{n_1 \cdots n_{k-1}} \right\rfloor$$

由此函数衍生的进位占位元 t 得到其占位多项式 $\tilde{F} = \sum_i f_i x^i t^{\chi(i)}$ 。

引理 4.1. 对于 $i + j < n$, 占位函数满足

$$0 \leq \chi(i) + \chi(j) - \chi(i + j) \leq k - 1$$

其中 $0 \leq$ 取等当且仅当 i, j 相加时不进位。

证明. 只需注意到 $0 \leq \left\lfloor \frac{i}{n_1} \right\rfloor + \left\lfloor \frac{j}{n_1} \right\rfloor - \left\lfloor \frac{i+j}{n_1} \right\rfloor \leq 1$, 其中 $0 \leq$ 取等当且仅当最低位加法未进位。同理可知, χ 的求和式中每一项均有相同性质。 \square

因而我们可以直接计算 $\tilde{F}(x, t) \times \tilde{G}(x, t) \bmod (t^k - 1)$, 第 (i_1, \dots, i_k) 项即为 $x^i t^{\chi(i) \bmod k}$ 次项系数。此时我们可以对 x 进行 DFT, 然后对 t 维暴力进行 $\Theta(k^2)$ 的乘法 (因为 $k \leq \log_2 n$) 完成计算, 复杂度为 $\Theta(kM(n))$ 。

在某种意义上, 上述占位函数的构造其实是唯一的, 若占位函数满足 $\chi(i) = \sum_{j=1}^k i_j \cdot c_j$, 容易发现上述的取整符号也符合这一要求, 其线性组合亦满足。每一个进位要求相当于每次 i_j 减去 k_j , 而 i_{j+1} 加上 1 后 $\chi(i)$ 恰好变化量为 1, 这就要求了 $k_j \cdot c_j - c_{j+1} = 1$ 。总共对于 $1 \leq j \leq k-1$ 各有一个约束, 因此 χ 的自由度为 1, 对于任意 c , $\widehat{\chi}(i) = \chi(i) + c \cdot i$ 同样是一个满足要求的占位函数。此时前面给出的 $\chi(i)$ 就是一个特解。

容易发现, 前述的下标映射天然赋予了一个在线算法的计算顺序, 因此有

引理 4.2. 多元在线卷积可在 $O(kR(n))$ 完成。

这一计算顺序也可以直接作用于牛顿迭代法的计算。其正确性亦不难解释, 因为我们将高维卷积考虑为带有占位多项式的 $\sum_i f_i x^i t^{\chi(i)}$ 卷积时, 对其进行任何运算, 只会产生一些 $j < \chi(i)$ 的 $x^j t^j$ 项, 且这些项不会再对形如 $x^i t^{\chi(i)}$ 项有贡献。因此我们实际上就是牛顿迭代的时候只维护这个多项式的 $\chi(i)$ 所构成的上轮廓。

4.2 微分型算子

在对多元函数的某一元 x_j 求微分时, 会损失 $[x_j^0]$ 部分的所有信息, 但在许多计算中, 我们仅利用了引理 2.6 的性质, 因此取各元微分的线性组合 $\mathfrak{D} = \sum_j c_j \frac{\partial}{\partial x_j}$ 同样是可用的。但这样一来就让 $x_1^{i_1} \dots x_k^{i_k}$ 项的系数分散到了 k 个位置, 这同样不太利于计算。

究其原因, 我们其实最好有一个既是微分型, 又是点乘型的算子。而取 $\mathfrak{D} = \sum_j c_j x_j \frac{\partial}{\partial x_j}$ 则会满足这一条件, 它使得

$$\mathfrak{D}x_1^{i_1} \dots x_k^{i_k} = (c_1 i_1 + \dots + c_k i_k)x_1^{i_1} \dots x_k^{i_k}$$

事实上, 这也覆盖了所有情况。

引理 4.3. 一个线性算子既是微分型, 又是点乘型当且仅当其为 $\sum_j c_j x_j \frac{\partial}{\partial x_j}$ 的形式。

证明. \Leftarrow : 逐条验证即可。 \Rightarrow : 只需考虑令 $c_j = \frac{\mathfrak{D}x_j}{x_j}$ 即得。 \square

而在实践中, 以下两种 \mathfrak{D} 的取法是较为有用的。

- 取 $\mathfrak{D} = \sum_j x_j \frac{\partial}{\partial x_j}$, 这样使得点乘的取值最少。
- 取 $\mathfrak{D}x^i = ix^i$, 这样可以在实现中完全照搬一元多项式的写法。

4.3 多元 Lagrange 反演

当我们讨论多元 Lagrange 反演时, 首先注意到的就是无法对一个多元幂级数定义复合逆, 因为多元幂级数需要带入多个参数而非一个。

定义 4.3 (树形复合方程). 对于 $G_i \in R[[\mathbf{x}]]$, 其中 $\mathbf{x} = (x_1, x_2, \dots, x_n)$ 。且 $G_i(0) \neq 0$, 那么令 $\mathbf{F} = (F_1, F_2, \dots, F_m)$ 满足 $F_i = x_i G_i(\mathbf{F})$ 。记 \mathbf{F} 是由 \mathbf{G} 定义的一组树形复合方程。

称其为树形是很形象的, 若从 OGF 的角度审视, \mathbf{G} 可以认为是给出了一族有根树的生成关系, 它规定了对于第 i 种节点, 每种孩子集合的设置方案数。而 F_i 就是规定以第 i 种节点为根的有根树。

定理 4.2 (多元 Lagrange). 对于 $H \in R((\mathbf{x}))$ 和由 \mathbf{G} 定义的树形复合方程 \mathbf{F} , 记 $\mathbf{x}^{\mathbf{k}} = x_1^{k_1} x_2^{k_2} \dots x_n^{k_n}$ 。有

$$[\mathbf{x}^{\mathbf{k}}]H(\mathbf{F}) = [\mathbf{x}^{\mathbf{k}}]H\mathbf{G}^{\mathbf{k}} \left\| \delta_{i,j} - \frac{x_j}{G_i(\mathbf{x})} \frac{\partial G_i(\mathbf{x})}{\partial x_j} \right\|_3$$

这里仅给出一个在特征为 0 的域 K 上 $H \in K[[\mathbf{x}]]$ 时的证明, 通用证明仍参看 [8, Sec. 1.2]。

³ 其中 $\delta_{i,j} = [i = j]$, 而 $\|\cdot\|$ 表示对所括的矩阵取行列式。

证明. 只需证明 $H = \mathbf{x}^m$ 时的情况即可推及其线性组合, 即只需证明

$$[\mathbf{x}^k]\mathbf{F}^m = [\mathbf{x}^k]\mathbf{x}^m\mathbf{G}^k \left\| \delta_{i,j} - \frac{x_j}{G_i(\mathbf{x})} \frac{\partial G_i(\mathbf{x})}{\partial x_j} \right\|$$

将其看做 EGF, 则等式左侧描述的是 m_i 颗以第 i 种节点为根的有根树组成的森林, 且第 i 种节点共有 k_i 个. 而观察右侧, $\mathbf{x}^m\mathbf{G}^k$ 的意图无非是先令每个节点任选其孩子集合, 而这会导致一些有环的情况被统计进入, 应当枚举环进行容斥. 考虑设矩阵

$$\mathbf{M} = \left(\delta_{i,j} - \frac{x_j}{G_i(\mathbf{x})} \frac{\partial G_i(\mathbf{x})}{\partial x_j} \right)_{i,j=1}^n$$

那么将原图的 t 个点替换为一个长度为 t 的环的方法则会被 $\text{tr } \mathbf{M}^t$ 统计 t 次, 故单个环进行替代的生成函数由

$$C = \sum_{t \geq 1} \frac{1}{t} \text{tr } \mathbf{M}^t = \text{tr} \left(\sum_{t \geq 1} \frac{1}{t} \mathbf{M}^t \right) = \text{tr} (-\log(\mathbf{I} - \mathbf{M}))$$

给出. 故对其容斥的生成函数为 $\sum_{l \geq 0} \frac{(-C)^l}{l!} = \exp(-\text{tr}(-\log(\mathbf{I} - \mathbf{M}))) = \exp(\text{tr} \log(\mathbf{I} - \mathbf{M}))$. 矩阵的迹与行列式之间有关系 $\det \exp \mathbf{A} = \exp \text{tr } \mathbf{A}$, 带入 $\mathbf{A} = \log(\mathbf{I} - \mathbf{M})$ 既得容斥因子为 $|\mathbf{I} - \mathbf{M}|$, 故命题得证. \square

例题 3 (矩阵树⁴). 有一张图, 总共有 $n_1 + \dots + n_k$ 个节点, 我们依次称其为第 $1, 2, \dots, k$ 部分. 对于全体 i 部分到 j 部分的点对, 两两之间有 $a_{i,j}$ 条边. 输出这张图的生成树数量同余 998244353. 保证 $1 \leq n \leq 10^8, 1 \leq k \leq 300, 0 \leq a_{i,j} = a_{j,i} \leq 1$.

解法 我们考虑先写出一个生成函数的方程组来刻画这个问题, 设 $T_i(\mathbf{x}) = T_i(x_1, \dots, x_k)$ 表示以一个颜色为 i 的节点为根的生成树方案数的 EGF (除以 $n_1! \dots n_k!$) 那么易列方程

$$T_i(\mathbf{x}) = x_i \exp \left(\sum_j a_{i,j} T_j(\mathbf{x}) \right)$$

那么我们要的即为 $\frac{1}{n_1} \left[\frac{x_1^{n_1}}{n_1!} \dots \frac{x_k^{n_k}}{n_k!} \right] T_1(\mathbf{x})$, 根据多元拉格朗日反演, 答案转换为

$$\begin{aligned} & \frac{1}{n_1} \left[\frac{x_1^{n_1}}{n_1!} \dots \frac{x_k^{n_k}}{n_k!} \right] T_1(\mathbf{x}) \\ &= \frac{1}{n_1} \left[\frac{x_1^{n_1}}{n_1!} \dots \frac{x_k^{n_k}}{n_k!} \right] x_1 \exp \left(\sum_{i,j} n_i a_{i,j} x_j \right) \left\| \delta_{i,j} - a_{i,j} x_j \right\| \\ &= \left[\frac{x_1^{n_1-1}}{(n_1-1)!} \frac{x_2^{n_2}}{n_2!} \dots \frac{x_k^{n_k}}{n_k!} \right] \exp \left(\sum_{i,j} n_i a_{i,j} x_j \right) \left\| \delta_{i,j} - a_{i,j} x_j \right\| \end{aligned}$$

⁴来源: 本人命制的模拟赛题目

注意此时求行列式的第 j 列只和 x_j 有关, 因此我们在算行列式的时候显然在确认第 j 列选谁的时候就得到 x_j 次幂部分的系数, 设 $d_j = \sum_i n_i a_{i,j}$, 那么对于 $j \geq 2$, 我们可以直接将矩阵的第 (i, j) 项替换为 $\left[\frac{x_j^{n_j}}{n_j!} \right] \exp(d_j x_j) (\delta_{i,j} - a_{i,j} x_j) = \delta_{i,j} d_j^{n_j} - a_{i,j} n_j d_j^{n_j-1}$; 对于 $j = 1$, 替换为 $\left[\frac{x_1^{n_1-1}}{(n_1-1)!} \right] \exp(d_1 x_1) (\delta_{i,1} - a_{i,1} x_1) = \delta_{i,1} d_1^{n_1-1} - a_{i,1} (n_1-1) d_1^{n_1-2}$ 。⁵因此我们直接计算这个矩阵的行列式即可得到答案。◆

5 集合幂级数

定义 5.1 (集合幂级数的集合定义). 令 $I = 2^{\{1, \dots, n\}} \cup \{0\}$, 对于 $S, T \in I \setminus \{0\}$, 记集合的无交并 $S \sqcup T$ 为:

$$S \sqcup T = \begin{cases} S \cup T & S \cap T = \emptyset \\ 0 & \text{else} \end{cases}$$

易见定义在 (I, \sqcup) 上的生成函数就是经典的集合幂级数。

但我们在这里给出不同于前人的讨论方法来定义集合幂级数, 由此可以更加容易地叙述其代数性质。

定义 5.2 (集合幂级数的多元幂级数定义). 我们称 $\mathbb{A}[x_1, \dots, x_n]/(x_1^2, \dots, x_n^2)$ 是 n 元集合幂级数。另记为 $\mathbb{A}\{x_1, \dots, x_n\}$ 。

由此可见, 集合幂级数无非是多元幂级数中最为极端的情况。

5.1 卷积

定理 5.1. 当 \mathbb{A} 取任何环时, 计算集合幂级数卷积的复杂度可以做到 $\Theta(n^2 2^n)$ 。

我们可以照搬高维序列卷积的计算方法, 由于各维度都极小, 采取插值转化的复杂度与高维卷积的通法一致, 且具有更小的常数。注意到在任意环上总是存在 $0, 1$, 因此我们可以直接选取 $\alpha_0^{(i)} = 0, \alpha_1^{(i)} = 1$ 进行插值转化。事实上, 这就是经典的快速 Möbius 变换以及占位多项式的表述: 看做集合并卷积并且用集合大小进行占位。

定理 5.2. 计算集合幂级数在线卷积的复杂度可以做到 $\Theta(n^2 2^n)$ 。

这一算法最早出现于 [9]。我们令在线算法的优先级为 $\delta(S) = |S|$, 我们只需令 k 从小到大, 处理全体 $|S| = k$ 的系数时, 对应 $|S| = 0, \dots, k-1$ 部分的占位多项式已经完成计算, 可以在 $\Theta(k 2^n)$ 时间内计算出占位多项式在 t^k 位置的乘法, 然后通过 $\Theta(n 2^n)$ 时间仅对 t^k 部分进

⁵由于指数来源于对 \exp 的系数提取, 这里 $b < 0$ 时应当有 $a^b = 0$ 。

行 Möbius 逆变换，从而得到所有系数。得到后再通过 $\Theta(n2^n)$ 时间仅对 t^k 部分进行 Möbius 变换。由此，整个算法的复杂度为 $\Theta(n^22^n)$ 。

对于不太拘泥于运算顺序的半在线集合幂卷积来说，上述算法已经足够，[5] 中将其称为半半在线卷积。与之相对的全半在线卷积需保证按照集合所代表的二进制数从小到大算出各项的值。后文我们将看到在特征为 2 的环上这将能够用于求解指数生成函数的微分方程。

定理 5.3. 全半在线集合幂卷积可以在 $\Theta(n^22^n)$ 时间内完成。

这一算法的进行需要将占位多项式变换为插值形式，我们取 k 个插值点，那么插值转化后的乘法可以在 $\Theta(k2^n)$ 内完成。我们考虑从最高位开始进行分治，由于 Möbius 变换的每一位都可以在 $\Theta(k2^n)$ 内完成，我们在分治过程中每次仅对最高位进行变换即可，具体过程可参见下述伪代码。

Algorithm 1 Fully Relaxed Subset Convolution

Require: A number n , an array $g[1 \dots 2^n - 1]$, relaxing function ϕ will modify $f[i]$ when calling $\phi(i)$

Ensure: Evaluate $f[i]$ and call $\phi(i)$ in the order $i = 0 \dots 2^n - 1$

- 1: Find k elements $a_1, \dots, a_k \in \mathbb{A}$ such that $\forall i \neq j, a_i - a_j$ is invertible
- 2: Let $\mathbf{A} = (a_i^j)_{i,j=0}^{k-1}$ and precalculate \mathbf{A}^{-1}
- 3: Initialize $\mathbf{F}[0 \dots 2^n - 1]$ with $\mathbf{0}$
- 4: **for** $i = 0 \dots 2^n - 1$ **do**
- 5: Let $b = \text{popcount}(i)$
- 6: $\mathbf{G}[i] \leftarrow (g[i]\mathbf{A})_b$
- 7: **end for**
- 8: **for** $i = 0 \dots n - 1$ **do**
- 9: Do Möbius transform on $\mathbf{G}[2^i \dots 2^{i+1} - 1]$
- 10: **end for**
- 11: **function** DivideConquer(l, t) {solve $[l, l + 2^t)$ }
- 12: **if** $t = 0$ **then**
- 13: Let $b = \text{popcount}(l)$
- 14: $f[l] \leftarrow (\mathbf{F}[l]\mathbf{A}^{-1})_b$
- 15: $\phi(l)$
- 16: $\mathbf{F}[l] \leftarrow (f[l]\mathbf{A})_b$
- 17: **else**
- 18: **for** $i = l \dots l + 2^{t-1} - 1$ **do**
- 19: $\mathbf{F}[i + 2^{t-1}] \leftarrow \mathbf{F}[i + 2^{t-1}] + \mathbf{F}[i]$
- 20: **end for**

```

21:   DivideConquer( $l, t - 1$ )
22:   for  $i = l \dots l + 2^{t-1} - 1$  do
23:        $\mathbf{F}[i + 2^{t-1}] \leftarrow \mathbf{F}[i + 2^{t-1}] + \mathbf{F}[i] \cdot \mathbf{G}[i + 2^{t-1}]$ 
24:   end for
25:   DivideConquer( $l + 2^{t-1}, t - 1$ )
26: end if
27: end function
28: DivideConquer( $0, n$ )

```

整体复杂度为 $T(n) = 2T(n-1) + \Theta(k2^n) = \Theta(kn2^n)$ 。由于过程中得到的多项式最高次数为 $2n-1$ ，取 $k = 2n$ 已足够。如能够取单位根进行循环卷积，则取 $k \geq n$ 已足够。

5.1.1 形式幂转化

集合幂的形式幂转化由定理 5.1 使用的算法一脉相承，其思想自 [10] 引入之初便已成型。与一般高维卷积相反之处在于，插值转化后采取的占位多项式上的乘法不采取循环卷积，因而在进行多次变换之后所求的值不会被污染。因此，若集合幂级数的计算过程中有一连串计算，可始终在插值转化的状态下进行计算。后文中我们将会看到这一特点有时能够优化理论复杂度，此外在实际应用时，这也能够有效地减小算法的常数。

5.2 逐点牛顿迭代法

考虑为了计算得到某个集合幂 F 时，我们分解为计算 $\frac{\partial}{\partial x_n} F$ 和 $F|_{x_n=0}$ 。此时，前者等价于 $[x_n^1]$ 部分，后者等价于 $[x_n^0]$ 部分。

若我们确定了 $[x_n^0]$ 之后就能够通过 $\Theta(f(n))$ 的时间计算出 $[x_n^1]$ ，那么总复杂度也是 $\Theta(f(n))$ 的。（显然 $f(n) = \Omega(2^n)$ ）

我们称这一方法为集合幂上的逐点牛顿迭代法。

例题 4 (无根树计数). 给一对称矩阵 \mathbf{G} ，对于每个非空子集 S ，求和

$$\sum_T \prod_{(i,j) \in T} \mathbf{G}_{ij}$$

其中 T 枚举了以 S 为点集的所有生成树。

解法 我们不妨只考虑 n 号点，先计算出所有 $S \not\ni n$ 的情况，也即 $[x_n^0]F$ ，对于 $[x_n^1]F$ 部分，去掉 n 号点后的各部分为选择了一个根的树。对于一个子集 T ，若其自成一个连通块，那么可以任选其中一个节点为根，因此我们将 $[x^T][x_n^0]f$ 乘以 $\sum_{j \in T} \mathbf{G}_{jn}$ 得到 \widehat{F} ，那么有

$$[x_n^1]F = \exp \widehat{F}$$

集合幂级数 \exp 可在 $\Theta(n^2 2^n)$ 内计算，故该问题复杂度为 $\Theta(n^2 2^n)$ 。◆

5.2.1 复合

定义 5.3 (集合幂级数复合). 给出 \mathbb{A} 上的 n 次指数生成函数形式的多项式 $F = \sum_{k=0}^n f_k \frac{x^k}{k!}$ 和不含常数项的集合幂级数 $G \in \mathbb{A}\{x_1, \dots, x_n\}$ 。记其复合为

$$F \circ G = \sum_{k=0}^n f_k \frac{G^k}{k!}$$

注意当 G 不含常数项时, $\frac{G^k}{k!}$ 总是良定义的。因为任取 k 个非空不交集合 S_1, \dots, S_k , 记 $\sqcup_{j=1}^k S_j = S$, 此时 S_1, \dots, S_k 的 $k!$ 个置换均贡献给 x^S 项的系数。因此我们不妨直接定义 $[x^S] \frac{G^k}{k!}$ 为对于全体将 S 划分为 k 个无序非空集合 S_1, \dots, S_k 的方案, 对于 $\prod_{i=1}^k [x^{S_i}] G$ 求和。

定理 5.4. 集合幂级数复合可以在 $\Theta(n^2 2^n)$ 时间内完成计算。

考虑 $\frac{\partial}{\partial x_n} (F \circ G) = F'(G) \frac{\partial}{\partial x_n} G$, 故我们将其规约为 $n-1$ 规模的子问题, 但要计算 F 和 F' 对其复合的结果。

记 G_k 为 $G|_{x_n=\dots=x_{n-k+1}=0}$, 归纳可知, 对于 $0 \leq k \leq n$, 我们需要解决 $F, F', \dots, F^{(k)}$ 复合 G_k 这 $k+1$ 个规模为 $n-k$ 的复合问题。具体过程由下述伪代码给出:

Algorithm 2 EGF Composite Set Power Series

Require: A number n , EGF $F(x) = \sum_{k=0}^n f_n \frac{x^k}{k!}$, sequence $g[0 \dots 2^n - 1]$ denoting the set power serie

- 1: **for** $i = 0 \dots n$ **do**
 - 2: Let $h_{0,i} = [f_i]$
 - 3: **end for**
 - 4: **for** $k = 1 \dots n$ **do**
 - 5: **for** $j = 0 \dots n - k$ **do**
 - 6: Let $h_{k,j}[0 \dots 2^{k-1} - 1] = h_{k-1,j}$
 - 7: Let $h_{k,j}[2^{k-1} \dots 2^k - 1] = h_{k-1,j+1} * g[2^{k-1} \dots 2^k - 1]$
 - 8: **end for**
 - 9: **end for**
 - 10: **return** $h_{n,0}[0 \dots 2^n - 1]$
-

瓶颈为对于每个 k , 我们进行了 $n-k$ 次 k 元集合幂级数乘法, 其复杂度可由和式 $\Theta(\sum_{k=0}^n k^2 2^k (n-k))$ 表示。下面证明其复杂度就是 $\Theta(n^2 2^n)$ 。

证明. 经计算可得:

$$\sum_{k=0}^n k^2 2^k (n-k) = 2(-13 + 13 \cdot 2^n - 3n - 6 \cdot 2^n n + 2^n n^2)$$

由此, 该算法的复杂度为 $\Theta(n^2 2^n)$ 。 □

事实上,这一方法具有极好的常数。对于集合幂 \exp 来说,由于 $F' = F$, 我们可以在迭代过程中减少一些维护,这种方法在常数上较进行 2^n 次微分方程实现 $\Theta(n^2)$ 形式幂 \exp 的方法效率更高。实践显示,在部分根据 F 针对性设计算法的问题,若其方法稍微复杂些,通法也会显著具有更好的常数⁶。

引理 5.1. 集合幂级数复合在插值转化的情形下可以 $\Theta(2^n M(n))$ 完成计算。

为了达成这一理论复杂度,主要在于将上一算法的各步骤精细处理。首先,我们不需要得到 G_k , 而只需要得到 G_k 一个插值转化的结果,我们首先对 G 在 x_n 这一维解除插值,这只需要 $\Theta(n2^n)$ 的复杂度,便得到了 $\frac{\partial}{\partial x_n} G$ 和 G_0 的插值结果,对 G_0 继续递归即可得到所有 G_k 的插值结果,复杂度为 $\sum_{k=1}^n k2^k = \Theta(n2^n)$ 。反之,所有子问题的计算瓶颈也皆为多项式乘法,因此复杂度为 $\sum_{k=0}^n M(k)2^k(n-k) = \Theta(2^n M(n))$ 。

5.2.2 Tutte 多项式

定义 5.4. 对于无向图 $G = (V, E)$, 定义其 Tutte 多项式 为

$$T_G(x, y) = \sum_{A \subseteq E} (x-1)^{k(A)-k(E)} (y-1)^{k(A)+|A|-|V|}$$

其中 $k(E)$ 表示图 (V, E) 的连通分量数。

接下来的几条资料显示了 Tutte 多项式与几个经典的图上计数问题的联系。由于篇幅所限,下面的几条引理仅述而不证。证明可参看 [11, X.4]

定义 5.5 (色多项式). 对于无向图 $G = (V, E)$, 存在多项式 $P_G(k)$ 使得带入正整数 k 时, $P_G(k)$ 的值表示 G 的 k -染色方案。

引理 5.2. 色多项式可表为 Tutte 多项式:

$$P_G(c) = (-1)^{|V|-k(E)} c^{k(E)} T_G(1-c, 0)$$

引理 5.3. 对于一个无向图 $G = (V, E)$, 下式

$$(-1)^{|V|} P_G(-1) = T_G(2, 0)$$

为 G 的无环定向数量,即给每条边进行定向,使得图是有向无环图的方案数。

引理 5.4. 对于一个无向图 $G = (V, E)$, $T_G(0, 2)$ 为 G 的强连通定向数量,即给每条边进行定向,使得图是强连通图。

⁶例: 集合划分计数, <https://loj.ac/p/154>

由上述引理可见, Tutte 多项式统一了几个经典的图论计数问题, 只要 Tutte 多项式的计算得到解决, 则可以一并解决以上几个问题。

定理 5.5. 设 $n = |G|$, 带入一组 x, y , 对于 V 的全体子集 V' , 设其导出子图⁷为 $G' = (V', E')$, $T_{G'}(x, y)$ 可以在总共 $\Theta(n^2 2^n)$ 的时间内计算。

注意到指数总是非负的, 因此计算可以规避除法。接下来介绍一种在任意环 \mathbb{A} 上都可进行的算法。

易见若 G' 有多个连通分支, 则 $T_{G'}$ 等于各连通分支的 Tutte 多项式之乘积, 不妨先求出全体 G' 为连通图时的答案, 其余部分即可调用子问题的乘积。首先考虑 A 作为边集时 G' 的各连通分支, 对于连通分支 S , 我们设该分支内在 A 中的边集为 B , 其权值为

$$f_S(B) = (y - 1)^{|B| - (|S| - 1)}$$

不难发现按照这种定义, $(x - 1)^{k(A) - k(E)}(y - 1)^{k(A) + |A| - |V|}$ 即为各连通分支的 f 之乘积乘以 $(x - 1)^{t-1}$, 其中 t 为连通分支的数量。因此若对于使得 S 为连通分量的全体方案 B 的 f 值求和得到 g_S , Tutte 多项式的值即可表为

$$[X^V] \sum_{t \geq 1} (x - 1)^{t-1} \frac{(\sum_{S \neq \emptyset} g_S X^S)^t}{t!}$$

这可以直接通过调用定理 5.4 的算法解决。接下来只需考虑如何计算 g_S 。

考虑进行逐点牛顿迭代计算 g , 对于 $S \ni n$, 考虑删去 n 后 S 剩余节点构成的各连通分支, 每个连通分支至少有一条连向 n 的边在 B 中, 设总共向 n 连有 m 条边, 则有 $y^m - 1$ 种方案。故将所有 $S \not\ni n$ 的 g_S 乘以各自对应的 $y^m - 1$ 后对其生成函数进行集合幂 \exp , 即得各 $S \cup \{n\}$ 项的值。

综上所述, 算法的时间复杂度为 $\Theta(n^2 2^n)$ 。

5.3 复合方程

由于集合幂级数实为一种特殊的多元幂级数, 因而我们可以照搬多元幂级数中所定义的树形复合方程以及多元 Lagrange 反演。由此我们可以在 $\tilde{O}(2^n)$ 时间内计算集合幂的树形复合方程中的一项。下面我们记 $f^S = \prod_{i \in S} f_i$ 。

定理 5.6. 对于由 \mathbf{G} 定义的树形复合方程 \mathbf{F} , 经过 $\Theta(n^4 2^n)$ 的预处理, 给定 $H \in \mathbb{A}\{x_1, \dots, x_n\}$ 询问 $[x^S]H(\mathbf{F})$ 可在 $\Theta(|S|^2 2^{|S|})$ 时间内完成。

一般情况而言的算法还是较为臃肿, 但对于一些特殊情况, 我们能够做到更好的复杂度。

⁷仅保留两 endpoint 均在 V' 中的边

5.3.1 对称复合方程

定义 5.6 (对称复合方程). 存在集合幂级数 \mathcal{B} , 其有值项皆满足度数 ≥ 2 , 和一族指数生成函数 g_1, \dots, g_n 使得

$$G_i = g'_i \circ \frac{\partial}{\partial x_i} \mathcal{B}$$

时, 称由 \mathbf{G} 定义的树形复合方程是对称的。

引理 5.5. 对于上述对称复合方程 \mathbf{F} , 存在集合幂级数 \mathcal{C} 使得

$$g_i \circ \left(\frac{\partial}{\partial x_i} \mathcal{B} \right) \circ \mathbf{F} = \frac{\partial}{\partial x_i} \mathcal{C}$$

事实上, 上述引理具有一个很强的组合直观, 让我们来解释 \mathcal{B} 与 \mathcal{C} 的含义。

\mathcal{C} 的构成相当于对所有圆方树进行统计。其中对于 $[x^S]\mathcal{C}$ 而言, S 描述了圆方树的圆点点集, 而 g_i 是对于 i 号点的度数赋予的权重, 而 $[x^S]\mathcal{B}$ 则描述了一个方点的邻接点集为 S 时赋予的权重。

定理 5.7. 给出 g_i 后, 通过 \mathcal{B} 计算 \mathcal{C} 可在 $\Theta(n2^n M(n))$ 时间内完成计算。

这一算法首先由 [12] 提出。首先考虑一个朴素的情况, 即 $g_i = x$, 此时应有 $\mathcal{C} = \mathcal{B}$, 因为一颗圆方树此时每个圆点都为叶子。而若 g_1 是一般的, 则说明与 1 号节点相邻可以有多个方点, 此时有 $\frac{\partial}{\partial x_1} \mathcal{C} = g_1 \circ \frac{\partial}{\partial x_1} \mathcal{B}$ 而其余部分保持不变。

我们考虑进行一组变换 $\mathcal{B} = \mathcal{T}_0 \rightarrow \mathcal{T}_1 \rightarrow \dots \rightarrow \mathcal{T}_n = \mathcal{C}$ 。在过程中的意图为将 g_i 从皆为 x 逐个改为输入。因此第 i 步变换会根据 g_i 进行组合 i 号点相邻的方点, 就有

$$\begin{aligned} \frac{\partial}{\partial x_i} \mathcal{T}_i &= g_i \circ \frac{\partial}{\partial x_i} \mathcal{T}_{i-1} \\ \mathcal{T}_i|_{x_i=0} &= \mathcal{T}_{i-1}|_{x_i=0} \end{aligned}$$

此时 $\Theta(n^3 2^n)$ 复杂度已经昭然若揭, 而注意到中间的变换我们可以始终在插值转化的结果下进行计算, 在进行第 i 次变换的时候, 我们只需将第 i 维还原, 还原一维只需要 $\Theta(n2^n)$, 因此瓶颈为进行 n 次在插值转化下的集合幂级数复合问题。总共复杂度为 $n \cdot \Theta(n2^n M(n)) = \Theta(n^3 2^n M(n))$ 。

虽然在实践中 $M(n)$ 仅有必要实现为 $\Theta(n^2)$, 但减少 Möbius 变换的次数能够有效地降低算法的常数。

引理 5.6. 对于 V 的每个子集 V' 的导出子图的点双连通子图数量可在 $\Theta(n2^n M(n))$ 时间内计算。

定理 5.7 所描述的算法在提出之初便是为了解决此问题, 这一情况被命名为连通-点双连通变换。注意到设 \mathcal{B} 为点双连通子图的集合幂级数, 则 $g_i = \exp x$ 时, \mathcal{C} 为连通图的集

合幂级数。注意到定理 5.7 所描述的变换若每一步变换存在逆变换，则可以通过 \mathcal{C} 逐步还原得到 \mathcal{B} ，此处便有

$$\begin{aligned} \frac{\partial}{\partial x_i} \mathcal{T}_{i-1} &= \ln \circ \frac{\partial}{\partial x_i} \mathcal{T}_i \\ \mathcal{T}_{i-1}|_{x_i=0} &= \mathcal{T}_i|_{x_i=0} \end{aligned}$$

复杂度为 $\Theta(n2^n M(n))$ 。

例题 5 (数仙人掌⁸). 给出一个无向简单图 $G = (V, E)$ ，问有多少个边集的子集使得图连通，且每条边最多在一个简单环内。答案同余 998244353，保证 $n \leq 18$ 。

解法 考虑这样的图每个点双连通分量的结构，必然是一条边或一个回路。我们通过状态压缩 DP 不难在 $\Theta(k^2 2^k)$ 时间内计算出所有以 k 为编号最大的点的点集 S 上的回路数量。由此即可得到所有满足条件的点双连通分量的集合幂级数 \mathcal{B} ，然后取 $g_i = \exp x$ 得到的 \mathcal{C} 即为仙人掌的集合幂级数。复杂度瓶颈为计算点双连通-连通变换。◆

值得一提的是，本题还有一些与定理 5.6 的算法中处理相近方法的做法，与上述做法在实践中的复杂度均为 $\Theta(n^3 2^n)$ ，但上述调用通法的解法反而表现出更好的常数。

6 指数生成函数

如若 $1 \sim n$ 在所进行计算的环上均同态可逆，那么指数生成函数没有额外讨论的意义，因为我们可以直接将序列变换为 $\widehat{f}_n = \frac{f_n}{n!}$ 。但若 \mathbb{A} 是具有小质数因子的同余运算 $\mathbb{Z}/M\mathbb{Z}$ ，或者特征不为 0 且小于等于 n 的有限域之类，便不能直接除以 $n!$ 进行处理。我们首要需要解决的便是乘法如何进行。

6.1 卷积

定义 6.1 (二项卷积). 对于以 $\mathbb{Z}_{\geq 0}$ 为下标，以 \mathbb{A} 为系数的序列，我们定义其卷积 $c = a * b$:

$$c_k = \sum_{i=0}^k \binom{k}{i} a_i b_{k-i}$$

定理 6.1 (四模数 NTT). 在模 M 运算下，存在 $\Theta(n \log n \cdot \omega(M))$ ⁹ 复杂度的二项卷积算法。

我们考虑首先如何解决 $M = p^k$ 时的情况，然后可以用 CRT 合并各情况。

我们记 $v_p(n)$ 是 $n!$ 中 p 的质因子次数， p -阶乘为 $n!_p = p^{v_p(n)}$ ，反 p -阶乘为 $\overline{n!}_p = \frac{n!}{n!_p}$ 。

⁸来源：加强自 Ildar Gainullin, <https://loj.ac/p/6719>

⁹ $\omega(n)$ 指 n 的互异质因子数量

那么由定义显然 $\overline{n!}_p$ 还是同余 M 可逆的。我们先令 $\widehat{a}_n = a_n \cdot (\overline{n!}_p)^{-1} \pmod M$ ，我们可以得到

$$\widehat{c}_n \equiv \sum_k \left(\frac{n!_p}{k!_p(n-k)!_p} \right) \widehat{a}_k \widehat{b}_{n-k} \equiv \sum_k p^{v_p(n) - v_p(k) - v_p(n-k)} \widehat{a}_k \widehat{b}_{n-k} \pmod M$$

定理 6.2 (Kummer). $v_p(n) - v_p(k) - v_p(n-k)$ 就是在 p 进制下， n 减去 k 时所发生的退位次数。

由于 n 在 p 进制下最多只有 $\log_p n$ 位可退，根据上述定理我们知道 $p^d \leq n$ ，因此我们在不取模的情况下，可以得到 $\widehat{c}_n \leq n \cdot nM^2 = n^2M^2$ 。

虽然 p 在模 M 下不可逆，但是当 $p \leq n$ ，自然满足在我们选取的 NTT 模数下都可逆。因此，这一涉及除法的卷积式子，因为已经保证了结果是值域在 n^2M^2 内的整数，所以我们只需选取 NTT 模数进行卷积，之后用 CRT 合并即可。取 $n \leq 10^6, M \leq 10^9$ 的一般情况下，可得 $c_n \leq 10^{30}$ ，使用四个 NTT 模数进行合并足够。目前美中不足的是，在通常的 M 在 10^9 范围内，就已经不可避免地四模数 NTT 最后的 CRT 阶段使用 int128。

因此对于每个 $M = p^k$ 的情况，我们都可以通过四模数 NTT 进行计算，那么对于一般的 M 进行 CRT，算法的复杂度为 $\Theta(n \log n \cdot \omega(M))$ ，或者也可以解释为，结果值域为 $n^{1+\omega(M)}M^2$ 的卷积。

定理 6.3. 模 M 二项卷积可在 $\Theta(R(n) \cdot \omega(M))$ 时间内在线进行。

经过 CRT 转化，我们将在线二项卷积转化为 $\omega(M)$ 个易于进行的 $M = p^k$ 形式的在线二项卷积。通过前述的四模数 NTT 规约，我们将在线二项卷积转化为四个同步进行的在线卷积。

通过此法，我们也可以较为高效地完成二项运算下的初等函数复合。

定理 6.4. 对于多项式 $f(x)$ 和 EGF 形式的多项式 $g(x)$ ，设 $D = \frac{d}{dx}$ ，可以在 $\Theta(n \log n \cdot \omega(M))$ 的时间内计算求导算子对多项式的乘法 $g(D) \cdot f(x)$ 。

只需注意到将 $f(x)$ 的系数看做输入向量 \mathbf{f} ，那么这一变换是「将 f 与 g 做二项卷积」的转置算法，我们对二项卷积的过程进行转置计算即可。

引理 6.1. 对于多项式 $f(x)$ ，可以在 $\Theta(n \log n \cdot \omega(M))$ 时间内计算 $f(x+c)$ 同余 M 的各项系数。

只需注意到 $f(x+c) = e^{cD} \cdot f(x)$ 即可规约到上述算法。

引理 6.2. 已知多项式 $f(x)$ 在 $0, \dots, n$ 下的点值（且在同余 M 下存在且以此形式给出），可以在 $\Theta(n \log n \cdot \omega(M))$ 内求 $f(m), f(m+1), \dots, f(m+n)$ 在同余 M 下的点值。

考虑进行二项式反演 $g_n = \sum_k \binom{n}{k} f(k) (-1)^{n-k}$ 后即得 f 的下降幂表示

$$f(x) = \sum_{k \leq x} g_k \binom{x}{k}$$

因此我们首先可以通过一次二项卷积完成二项式反演，之后的第二次卷积是一个起点不太相同的二项卷积，但由于其中出现的组合数上标最大为 $n+m$ ，因此素因子幂次有 $p^k \leq n+m$ ，可知进行类似转化后的值域被控制在 $n \cdot (n+m)M^2 \leq nM^3$ 内，四模数 NTT 的转化仍是可用的。

7 总结

本文中，我们简要勾勒出了生成函数计算时问题的主要分支、计算时的不同难度层次、以及解决对应的问题的部分纲领。然而吾生也有涯，而知也无涯，本文的理论还有无数的空缺之处以及开放性问题尚待解决。希望能有兴趣的同学发挥愚公移山之精神，进一步完善生成函数计算的理论图景！

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢北大附中肖然老师的关心和指导。

感谢家人、朋友对我的支持与鼓励。

感谢赵雨扬前辈与我讨论以及给予的启发，以及其对于集合幂级数等相关部分前瞻性的理解。

感谢戴江齐同学与我讨论以及给予的启发。

参考文献

- [1] Kedlaya, Kiran & Umans, Christopher. (2008). Fast Polynomial Factorization and Modular Composition. *SIAM Journal on Computing*. 40. doi: 10.1137/08073408X
- [2] 陈宇, 赵雨扬, 曾致远. (2020). 转置原理的简单介绍. IOI2020 中国国家集训队论文集.
- [3] 赵雨扬. (2018 ~ 2020). 关于优化形式幂级数计算的 Newton 法的常数. <https://negiizhao.blog.uoj.ac/blog/4671>

- [4] Hoeven, Joris. (2007). New algorithms for relaxed multiplication. doi: 10.1016/j.jsc.2007.04.004
- [5] 罗煜翔。(2020)。浅谈 Nimber 和多项式算法。IOI2020 中国国家集训队论文集。
- [6] Bostan, A. & Mori, Ryuhei. (2020). A Simple and Fast Algorithm for Computing the N -th Term of a Linearly Recurrent Sequence. doi: 10.1137/1.9781611976496.14
- [7] 钟子谦。(2019)。两类递推数列的性质和应用。IOI2019 中国国家候选队论文集。
- [8] Goulden, Ian P., and David M. Jackson. (2004). Combinatorial Enumeration.
- [9] 洪华敦。(2018)。《「WC2018」州区划分》解题报告。
- [10] 吕凯风。(2015)。集合幂级数的性质与应用及其快速算法。IOI2015 中国国家候选队论文集。
- [11] Bollobás, Béla. (1998). Modern Graph Theory
- [12] 赵雨扬。(2019)。《点双连通生成子图计数》题解。由马耀华同学代为撰写：<https://loj.ac/d/2668>

《逛公园》命题报告

福建省福州第一中学 林昊翰

摘要

《逛公园》是我为 IOI2021 候选队员互测活动命制的一道试题，是一道“广义串并联图”相关的问题。本文先介绍了以往的通过收缩操作和建立串并联树进行动态规划的方法，但该方法扩展性较差，仅能用于规划类问题，无法完全解决这道题目。接下来引入了内部图和外部图的概念，说明了“内部图”的性质与树上问题时的“子树”有很大的相似性，并利用该性质在广义串并联图上进行类似边分治的算法来解决这个问题。我希望《逛公园》这道题能够带给大家更多关于广义串并联图的相关思考。

1 试题

1.1 题目描述

我们称满足对于任意 4 个节点都不存在 6 条两两除公共端点外没有公共点的路径连接这 4 个节点中的每一对节点的无向连通图为广义串并联图。

给定一张 n 阶简单（无自环无重边）广义串并联图 $G = (V, E)$ ，每条边有长度 l_i 。有 q 次询问，每种询问形如一下两种之一：

- 1、给定一个点集 S ，求 S 中每一对不同的无序点对 (S_i, S_j) 的最短路径长度和。
- 2、给定一个点集 S 和参数 v ，求 S 中有多少对不同的无序点对 (S_i, S_j) 满足其最短路径长度不超过 v 。

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 m 行，每行三个正整数，其中第 i 行为 x_i, y_i, l_i ，表示第 i 条道路的连接的两个点编号为 x_i, y_i ，长度为 l_i 。

第 $m + 2$ 行一个非负整数 q 。

接下来 q 组询问，每组形如以下两种之一：

1、为上述第一种询问。第一行两个正整数 $1\ k$ ，其中 k 表示选择的点集 S 的大小。接下来一行 k 个整数 S_1, S_2, \dots, S_k ，表示选择的点集 S 。

2、为上述第二种询问。第一行三个正整数 $2\ k\ v$ ，其中 k, v 分别表示选择的点集 S 的大小和给定的参数。接下来一行 k 个整数 S_1, S_2, \dots, S_k ，表示选择的点集 S 。

1.3 输出格式

输出 q 行，其中第 i 行表示第 i 个询问的答案。

1.4 样例输入

```
2 1
1 2 5
2
1 2
1 2
2 2 6
1 2
```

1.5 样例输出

```
5
1
```

1.6 样例解释

1,2 之间的边的长度 $l = 5$ 。

第一个询问相当于求 1,2 之间的最短路径长度。答案为 5。

第二个询问相当于求 1,2 之间的最短路径长度是否 ≤ 6 ，是则输出 1，否则输出 0。答案为 1。

1.7 数据范围

设 K 为所有询问给定的点集 S 的大小之和。

保证 $2 \leq n \leq 5 \times 10^5$ ， $1 \leq q, K \leq 5 \times 10^5$ 。

保证 $1 \leq l_i \leq 10^9$ ， $1 \leq v \leq 10^{18}$ 。

子任务 1 (5 分): $n, m, K \leq 3000$ ；

- 子任务 2 (5 分): $q = 1$, 仅有第一种询问, 给定的图为树;
- 子任务 3 (20 分): 给定的图为树;
- 子任务 4 (5 分): $q = 1, k_i = n$, 仅有第一种询问, 给定的图为仙人掌;
- 子任务 5 (20 分): 给定的图为仙人掌;
- 子任务 6 (25 分): $k_i = 2$;
- 子任务 7 (20 分): 无特殊限制。

1.8 时空限制

时间限制: 5s
空间限制: 1GB

2 一些约定

对于图 G , 用 $dis(x, y)$ 表示点 x 与点 y 之间的最短路径长度。
用中括号包起来的逻辑表达式 $[x]$ 的取值为: 若 x 成立 $[x] = 1$, 否则 $[x] = 0$ 。

3 初步分析

3.1 算法一

对于子任务 1, n, m, K 不超过 3000, 可以考虑对于每一对点预处理出它们之间的最短路径长度, 对于询问枚举每一对点并直接统计答案。

该算法时间复杂度为 $O(nm \log n + K^2)$ 。

期望得分 5 分。

该算法效率十分低下, 是因为该算法没有利用到 G 是广义串并联图的性质。

接下来先从更特殊的情况入手考虑这个问题。

4 树

4.1 算法二 树, 仅有一组的第一种询问

由于树上两点之间的简单路径唯一, 可以考虑每一条边被经过了多少次。

对于一条边 i , 它将树分成两颗子树, 若其中一颗子树有 a_i 个点在询问点集 S 中, 另一颗有 $b_i = k - a_i$ 个, 那么它就被经过了 $a_i \times b_i$ 次。

可以进行一次树形动态规划求出所有 a_i, b_i ，答案即为 $\sum_{i=1}^m (a_i \times b_i \times l_i)$ 。

时间复杂度 $O(n)$ 。

期望得分 5 分。

4.2 算法三 树，一般情况

此时这个问题是一个比较经典的树上简单路径类问题，有两种常见做法：虚树和点分治。由于虚树做法难以扩展到广义串并联图上，与本文内容关系不大，下面仅介绍点分治做法：

对于重心 u ，预处理出 u 到每一个点 x 的最短路长度 $dis(x, u)$ ，然后离线处理每一个询问：

询问一：对于每个点 x ，其贡献为 $dis(x, u) \times (k - c)$ 其中 k 为询问点集大小， c 为删去重心 u 后与 x 在同一连通块的询问点个数。

询问二：对于每个点 x ，其贡献为满足 $dis(x, u) + dis(y, u) \leq v$ 的，且删去重心 u 后与 x 不在同一连通块 y 的个数。考虑将所有满足条件的 y 减掉与 x 在同一连通块的 y 。统计所有满足条件的 y 可以通过按 $dis(x, u)$ 排序后双指针扫描实现，而与 x 在同一连通块的 y 则可以将每个连通块分别按 $dis(x, u)$ 排序并统计。（注意这样统计出来的答案为实际答案的两倍，最终答案要除以 2）。

然后删去重心 u ，对于剩下的图上的每个连通块分别递归下去。

时间复杂度为 $O((n + K) \log^2 n)$ 。

期望得分 25 分。

5 仙人掌

对于仙人掌上的问题，一种经典的方法是建立圆方树并转化为类似树上问题。

其中子任务 4 可以通过圆方树上动态规划解决，子任务 5 可以通过圆方树上点分治解决。

该部分内容与本文主题无关，因此不再赘述。相关内容可以参见陈俊锟在 IOI2017 候选队的论文《〈神奇的子图〉命题报告及其拓展》以及他的 UOJ 博客¹。

¹<http://immortalco.blog.uoj.ac/blog/1955>

6 广义串并联图

6.1 重要性质

对于一张图 G ，我们可以进行一些操作来缩小图的规模，我们称它们为收缩操作。

定义 6.1. 若图中存在点度为 1 的点，直接将其删除，称其为删 1 度点操作。

定义 6.2. 若图中存在一对重边，将其合并成为一条，合并成的新边的长度为这一对重边长度的较小值，称其为叠合重边操作。

定义 6.3. 若图中存在点度为 2 的点，如果与该点相连的两条边是一对重边，则先进行叠合重边操作；否则设与该点相连的两个点分别为 x, y ，删除这个点和与其相连的边，在 x, y 之间建立新边，新边的长度为被删除的两条边的长度之和，称其为缩 2 度点操作。

引理 6.1. 以上操作不会影响未被删除的点之间的最短路径长度。

证明.

1、对于删 1 度点操作，其它点对的最短路径不会经过 1 度点。

2、对于叠合重边操作，其他点对的最短路径在经过重边时一定选择长度较小的一条。

3、对于缩 2 度点操作，其他点对的最短路径若经过这个 2 度点，那么一定会同时经过与它相连的两条边。 \square

定理 6.1. 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。

定理 6.1 的证明十分繁琐，故略去，具体可以参见吴作同在 IOI2019 候选队的论文《〈公园〉命题报告》。

定理 6.2. 任意简单（无重边无自环）广义串并联图满足边数不超过点数的两倍。

证明. 由于每次缩 2 度点和删 1 度点操作都各减少一条边的一个点，叠合重边操作减少一条边，但每次缩 2 度点后至多进行一次叠合重边操作，因此任意无重边的广义串并联图满足边数不超过点数的两倍。 \square

6.2 建立串并联树

考虑将上述操作变成树形结构：

初始图中每一个点和每一条边均对应为串并联树的叶子节点，然后在收缩过程中建立这个串并联树：

1、删 1 度点操作：建立一个新的节点 v' ，将被删除点 x ，与其相连的边 a 和 a 的另一个端点 z 对应的树上节点 v_x, v_a, v_z 的父亲设为 v' ，然后将点 z 对应的树上节点设为 v' 。2、

叠合重边操作：建立一个新的节点 v' ，将两条重边 a, b 对应的树上节点 v_a, v_b 的父亲设为 v' ，然后将合并后的边对应的树上节点设为 v' 。

3、缩 2 度点操作：建立一个新的节点 v' ，将被删除点 x 和与其相连的两条边 a, b 对应的树上节点 v_x, v_a, v_b 的父亲设为 v' ，然后将建立的新边对应的树上节点设为 v' 。

这个串并联树中，每一个树上节点唯一对应图上的一个点或一条边，初始图 G 中每一个点或一条边唯一对应一个树上叶子节点，且每一个非叶子节点也对应一个收缩操作。

引理 6.2. 对于任意一个串并联树上非叶子节点 u ，都可以在不进行其它收缩操作的情况下将所有在点 u 的子树内（包括点 u ）的非叶子节点对应的收缩操作执行完毕。

证明. 考虑反证法：找到任意一个深度最大的不满足条件的点 u 。那么它子树内的操作都可以先执行，剩下点 u 对应的操作无法执行。

1、若 u 对应的是叠合重边操作，那么 u 的儿子对应需要被叠合的两条重边， u 子树内的操作执行完毕后这两条重边一定会出现， u 对应的操作一定可以执行，矛盾。

2、若 u 对应的是删 1 度点操作或缩 2 度点操作，如果 u 的儿子执行完毕后如果点 u 对应的操作无法执行，那么此时需要被收缩的点（设其为 x ）的点度不符合条件，需要再执行一些其他操作才能满足条件。

能够影响 x 点度的有叠合重边操作和删 1 度点操作，但不在点 u 的子树内的删 1 度点操作会导致点 x 对应的树上节点不在点 u 的子树内，不在点 u 的子树内叠合重边操作会导致与点 x 相连的边对应的树上节点不在点 u 的子树内，与它们是点 u 的儿子矛盾。

由此，引理 6.2 得证。 □

6.3 算法五 询问点集大小为 2

对于一个串并联树上节点 u ，我们定义它的内部图 $G'_u(V'_u, E'_u)$ 如下：

定义 6.4. 串并联树上节点 u 的内部图 G'_u 为初始图 G 的一个子图。

1、如果点 u 在图上对应的是一个点，那么对于初始图 G ，如果图上的一个点或一条边对应的树上叶子节点在点 u 的子树中，那么这个点或边属于 G'_u 。

2、如果点 u 在图上对应的是一条边，设这条边为 (x, y) ，那么对于初始图 G ，如果图上的一个点或一条边对应的树上叶子节点在点 u 的子树中，那么这个点或边属于 G'_u ，且点 x 和点 y 也属于 G'_u 。

定理 6.3. 对于一个树上节点 u 的内部图 $G'_u(V'_u, E'_u)$ ： $\forall (x, y) \in E'_u$ 有 $x \in V'_u, y \in V'_u$

证明. 对于一条在 G'_u 的边 (x, y) ，先执行点 u 子树内部的收缩操作，那么边 (x, y) 一定会被收缩掉。根据上述收缩操作，两个端点 x, y 要么是已经被收缩掉的点，要么（如果点 u 对应的是边）是点 u 的端点之一，也就一定属于 V'_u 。 □

询问点集大小为 2 时等价于多次询问两个点之间的最短路径长度，设这两个点分别为 f, g 。

考虑在串并联树上进行树形动态规划：

如果 u 对应一个点，那么记该点为点 A ，点 B 不存在。如果 u 对应一条边，那么记与它相邻的两个点分别为点 A 和点 B 。

设计如下的 DP 数组（以下的最短路径长度均为只保留点 u 的内部图后的最短路径长度）：

$fa[u]$ ：点 f 到点 A 的最短路径长度（若点 f 不在 u 的内部图中，或 u 对应一条边且点 f 为点 u 对应边的端点，该值为 $+\infty$ ）

$ga[u]$ ：点 g 到点 A 的最短路径长度（细节同上）

$fb[u]$ ：点 f 到点 B 的最短路径长度（若点 f 不在 u 的内部图中，或 u 对应一条边且点 f 为点 u 对应边的端点，或点 B 不存在，该值为 $+\infty$ ）

$gb[u]$ ：点 g 到点 B 的最短路径长度（细节同上）

$fg[u]$ ：点 f 到点 g 的最短路径（若不满足上述任一条件，该值均为 $+\infty$ ）

然后就能在串并联树上进行树形动态规划来解决一组询问（具体转移较为繁琐且与本文内容无关，在此不详细列出）。

对于多组询问的处理，考虑分析上面动态规划的转移形式，发现它可以写成类似矩阵的形式，只是将加法换成取 \max ，乘法换成加法。

由于加法， \max 运算满足结合律，且加法对 \max 运算满足分配率，所以这种新定义的矩阵依旧满足结合律^[5]。

对于一组询问 f, g ，我们仅需要得到 f, g 对应串并联树上节点到它们的 LCA 的转移矩阵乘积和 LCA 到根的转移矩阵乘积，采用树上倍增算法即可解决。

该算法复杂度为 $O((n+q)\log n)$ ，可以通过子任务 6，期望得分 25 分。

6.4 算法六 一般情况

由于串并联树是一个三叉树，可以直接进行边分治，且边分治仅会将树分成两个部分，比起会将树分成多个部分的点分治来说需要考虑的情况要简单得多，因此这里采用边分治而非点分治。

边分治时每次找到一条边并将树按照这条边分开。

考虑这条边的两个端点，它们一定是儿子-父亲关系，设其中的儿子为点 u ，那么就变成了每次考虑 u 的子树内到 u 的子树外的答案。

再定义外部图为：

定义 6.5. 一个串并联树上节点 u 的外部图为先进行所有在点 u 子树内的收缩操作后的图。

那么有：

定理 6.4. 对于一个串并联树上节点 u ，如果它对应一个点 x ，那么其内部图和外部图点集的交恰为 x ，如果它对应一条边 (x, y) ，那么其内部图和外部图点集的交恰为 x, y 。

证明.

1、如果 u 对应一个点 x ，那么点 u 对应到一个缩一度点操作，点 u 子树中对应的非 x 节点都已经被缩掉，属于内部图，而其它没被缩掉的点属于外部图。

2、如果 u 对应一条边 (x, y) ，那么点 u 子树中对应的节点都已经被缩掉，属于内部图，而其它没被缩掉的点属于外部图，且内部图额外加上了点 x 和点 y 。□

对于点 u 离线处理每一个询问：

我们称点 u 内部图和外部图点集的交为它的分割点，它有一到两个分割点，记为 p_1, p_2 (p_2 可能不存在)。

首先 $O(m \log n)$ 预处理出分割点到图中所有点的最短路径长度，如果该询问点集中包含分割点，暴力统计其答案并删除，这样询问中的点要么属于内部图，要么属于外部图。

然后统计所有询问中所有满足 x 属于内部图且 y 属于外部图的点对 (x, y) 的贡献：

由定理 6.4，它们之间的路径至少会经过 p_1, p_2 其中之一，分类讨论，得出其最短路径长度为：

$$dis(x, y) = \min\{dis(x, p_1) + dis(y, p_1), dis(x, p_2) + dis(y, p_2)\}$$

(如果 p_2 不存在，那么 $dis(x, p_2) + dis(y, p_2) = +\infty$)

这个 \min 取到哪一边和 $(dis(x, p_1) - dis(x, p_2)) + (dis(y, p_1) - dis(y, p_2))$ 的正负性有关，因此可以将属于外部图的点 y 分别按照 $dis(y, p_1) + dis(y, p_2)$ 排序。

对于属于内部图的点 x 。如果 $(dis(y, p_1) - dis(y, p_2)) \geq -(dis(x, p_1) - dis(x, p_2))$ ，那么就有 $dis(x, y) = dis(x, p_1) + dis(y, p_1)$ 。这样就只需要知道所有满足 $(dis(y, p_1) - dis(y, p_2)) \geq -(dis(x, p_1) - dis(x, p_2))$ 的相关信息，就能算出点 x 的贡献。具体地：

- 1、对于第一种询问，点 y 的个数及其 $dis(y, p_1)$ 的和。
- 2、对于第二种询问，满足 $dis(y, p_1) \leq v - dis(x, p_1)$ 的点 y 的个数。

这两者都是能比较简单地通过前缀和和双指针扫描直接计算的。

$(dis(y, p_1) - dis(y, p_2)) < -(dis(x, p_1) - dis(x, p_2))$ 的情况同理。

至此边分治的一步就完成了：我们解决了所有从内部图的点到外部图的点的贡献，也即 u 的子树内到 u 的子树外的贡献。

接下来我们就把剩下询问分成了两个部分：两者均属于外部图的点对之间的贡献和两者均属于内部图的点对之间的贡献。

由引理 6.1，外部图之间点对的最短路径长度与原图一致。

但对于内部图，两个点之间的最短路径是有可能经过不属于内部图的点的。

由定理 6.4，此时这条路径不属于内部图的部分一定是从 p_1 走到 p_2 或从 p_2 走到 p_1 (且 p_2 一定存在)，那么其长度也就一定是它们之间的最短路径 $dis(p_1, p_2)$ 。

只要在内部图中加上连接 p_1 和 p_2 ，长度为 $dis(p_1, p_2)$ 的无向边即可 (如果 p_2 不存在就什么也不干)。

然后我们就可以将整张图拆分成内部图和外部图两张相互独立的图，且它们内部的点对之间的最短路径长度不变，这样原问题也就被分成了在这两个图上的子问题。

由定义可知，外部图对应的串并联树为原串并联树删除 u 的子树（不包括 u ）后的树，内部图对应的串并联树为 u 的子树（包括 u ）再加上点 u 和新加的边叠合重边。

设原图的串并联树结点数为 N ，那么新图结点数就分别为 $x + 1$ 和 $N - x + 2$ ($\frac{N}{4} \leq x \leq \frac{3N}{4}$)。对于 $N \leq 10$ 的图可以暴力处理所有询问。

该算法时间复杂度为 $O((n + K) \log^2 n)$ ，期望得分 100 分。

7 扩展

7.1 点/边分治的一个扩展

由上述的算法五我们可以得到一个点/边分治算法的推广：

如果能够把一张点数为 N 的图分成若干个子图，每个子图点数不超过 $aN + c$ 满足 a, c 为固定常数且 $0 < a < 1$ 。每一个点和每条边都至少属于其中的一个子图，属于多个子图的点数量不超过某个较小的常数 k 。

那么我们可以把所有属于多个子图的点提取出来暴力处理，计算所有包含了这些点的路径的贡献，然后将这些点删除。每一个连通块往下分治，即可做到类似点/边分治的效果。

事实上，传统的点分治就是上述算法的 $k = 1$ 时的情况。

例 1. *Giant Penguin*²

给定一张连通无向图 $G = (V, E)$ ，每条边长度均为 1，这张图满足每个点至多属于 k 个简单环。你需要依次执行 q 个操作，每种操作为以下两者之一：

- 1、给定 v ，标记顶点 v ，保证点 v 之前没有被标记。
- 2、给定 v ，输出所有被标记的点到点 v 的最短路径长度的最小值，保证此时至少存在一个点被标记。

$$|V| \leq 10^5, |E| \leq 2 \times 10^5, q \leq 2 \times 10^5, k \leq 10。$$

这题有一个重要性质：这张图满足每个点至多属于 k 个简单环，因此只需要任取这张图的一个生成树，在这个生成树上进行点分治。

对于重心 u ，将它作为生成树的根，它就将这颗树分成了若干个子树。考虑所有不在生成树上的边，如果它连接了 u 的两个不同的子树，那么这条边提供了一个包含重心 u 的简单环，因此这种边不超过 k 条。

这 k 条边端点的点集并不超过 $2k$ ，加上重心 u 就是不超过 $2k + 1$ 个点，称这些点构成的点集为 S ，可以将 S 中的点一并删除，然后每个连通块往下递归。

²题目来源：300iq Contest 3

对于具体的操作，可以在提取出 S 后对每个 $x \in S, y \in V$ 求出 $dis(x, y)$ ，并建出点分治树，每个点分治树上结点 z 均有一个 S 集合 S_z ，并记 f_i ，初值均为 $+\infty$ 。

然后对于每一个具体操作：

1、对于 1 操作，找到满足的 $v \in S_z$ 点 z ，依次考虑点 z 以及其在点分治树上的祖先。当我们考虑点 z' 时，对于每一个 $x \in S_{z'}$ ，更新 $f_x = \min\{f_x, dis(v, x)\}$ 。

2、对于 2 操作，记本次操作的答案为 ans ，初值为 0。找到满足的 $v \in S_z$ 点 z ，依次考虑点 z 以及其在点分治树上的祖先。当我们考虑点 z' 时，对于每一个 $x \in S_{z'}$ ，更新 $ans = \min\{ans, f_x + dis(v, x)\}$ 。

即可解决该问题，时间复杂度为 $O(k(|V| + q) \log |V|)$ 。

7.2 KDtree 上启发式合并维护动态规划

对于广义串并联图上的动态规划问题，常常转化为串并联树上动态规划。

由内部图和外部图的性质，广义串并联图上的动态规划常常需要维护两个点上的状态信息，也就是串并联树上点 u 上的动态规划状态常常是二维的，形如 $f[u][i][j]$ 。

而处理树上动态规划时有一个很好用的工具：线段树合并，我们希望它能被扩展到二维情况。

例 2. 树上偏序³

给定一棵 n 个结点编号为 $1, 2, \dots, n$ 的有根树 T 以及 m 个三元组 (a, x, y) ，其中 a, x, y 均为正整数。称一个三元组 (a, x, y) 在结点 x 的子树中当且仅当树 T 中编号为 a_i 的结点在结点 x 的子树中。

求有多少个不同的非空三元组集合 S ，满足对于每一个结点 x ，要么不存在三元组 $(a, x, y) \in S$ 在 x 的子树中；要么存在一个三元组 $(a', x', y') \in S$ 在 x 的子树中且对于任意在 x 的子树中的三元组 $(a, x, y) \in S$ ，都有 $x \leq x', y \leq y'$ ，答案对 998244353 取模。

$n, m \leq 50000$ 。

7.2.1 朴素算法

设计动态规划，记 $f[i][j]$ 为在 i 的子树中且结点 i 选取的三元组 (a', x', y') 为第 j 个三元组的方案数（若子树中不存在三元组则 $j = 0$ ）。

合并两个点 u, v 的状态 $f[u][i], f[v][j]$ 到点 x 时：

- 1、如果 $x_i \geq x_j, y_i \geq y_j$ 或 $j = 0$ ，那么转移： $f[x][i] += f[u][i] * f[v][j]$ 。
- 2、否则如果 $x_i \leq x_j, y_i \leq y_j$ 或 $i = 0$ ，那么转移： $f[x][j] += f[u][i] * f[v][j]$ 。
- 3、若以上条件均不满足，那么这一对状态没有贡献。

对于一个结点 u 的状态，先合并它所有子树的状态作为 $f[x][j]$ 。

³题目来源：原创

然后对于所有三元组 (a_i, x_i, y_i) 满足 $a_i = u$ ，相当于点 x 的状态再合并上

$$f[n+i][j] = \begin{cases} 1; i \in 0, j \\ 0; otherwise \end{cases}$$

该算法时间复杂度为 $O(m^2)$ ，效率低下。

7.2.2 KDtree 上启发式合并

维护二维情况下的矩形信息一般使用 KDtree，于是考虑使用 KDtree 合并。二维上的合并十分复杂，可以使用启发式合并。

N 个点的静态 KDtree 单次查询复杂度为 $O(\sqrt{N})$ ，插入可以考虑采用分块重建：插入后每次查询都暴力求新插入的点的贡献，如果插入的点数超过 \sqrt{N} ，那么 $O(N)$ 暴力重构整个 KDtree，这样单次插入及询问复杂度均为均摊 $O(\sqrt{N})$ 。

启发式合并时每个点每次被合并其所在 KDtree 大小都会翻倍，设总点数为 n ，那么对于一个点，其被合并的代价不超过 $\sqrt{n} + \sqrt{\frac{n}{2}} + \sqrt{\frac{n}{4}} + \dots$ ，这是 $O(\sqrt{n})$ 级别的，那么总复杂度为 $O(n\sqrt{n})$ 。

此时合并两个点 u, v 的状态 $f[u][i], f[v][j]$ 到点 x 时：

1、我们对于每一个 $f[u][i]$ ，设向量

$$g[u][i] = \begin{bmatrix} f[u][i] & 0 \end{bmatrix}$$

2、接下来对于每一个 $f[v][j]$ ，对于所有位于 (x_j, y_j) 右下角的 (x_i, y_i) ， $g[u][i]$ 都右乘上矩阵

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

3、然后对于每一个 $f[v][j]$ ，求出所有位于 (x_j, y_j) 左上角的 (x_i, y_i) 的 $f[u][i]$ 的和 x' （也就是左上角 $g[u][i]$ 和 $[x' \ y']$ 中 x' 的值），然后在 KDtree 中加入

$$g[u][j] = \begin{bmatrix} 0 & x' \times f[v][j] \end{bmatrix}$$

4、转移所有完毕后将所有 $g[x][i]$ 右乘上矩阵

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

将其变回 $[f[x][i] \ 0]$ 的形式。

上述操作均可以在点数为 N 的 KDtree 中单次均摊 $O(\sqrt{N})$ 实现，因此总时间复杂度为 $O(m\sqrt{m})$ ，可以通过该例题。

7.3 带源汇串并联图计数

广义串并联图存在一个子集：**串并联图**：

定义 7.1. 如果一张图 G 可以仅通过缩 2 度点操作和叠合重边操作将 G 变成仅有两个点，且这两个点之间有且仅有一条边连接的图，则图 G 为串并联图。

引理 7.1. 一个点双连通的广义串并联图为串并联图。

证明. 考虑反证法。若一个广义串并联图不是串并联图，那么它的收缩操作中存在删 1 度点操作，且删除后点的数量不为 1。设这个收缩操作对应的树上结点为 u ，那么 u 的内部图与外部图的点集的交仅包含一个点，这个点为割点，与原图点双连通矛盾。 \square

引理 7.2. 任意广义串并联图的任意串并联树上结点 u 的内部图均连通。

证明. 考虑进行归纳，首先叶子结点的内部图显然连通。对于一个树上结点 u ，假设其儿子的内部图均连通，那么点 u 的内部图点集为儿子内部图点集的并。点 u 的儿子中至少有一个对应的是边，且对应边的儿子的内部图点集与其它儿子内部图点集均有交。因此点 u 的内部图连通。 \square

我们定义串并联图的源汇为：

定义 7.2. 对于一张串并联图 G ，如果 G 可以仅通过缩 2 度点操作和叠合重边操作将 G 变成仅有两个点 S 和 T ，且这 S, T 之间有且仅有一条边连接的图，则无序点对 (S, T) 为串并联图 G 的一组源汇。

引理 7.3. 对于一个串并联图 G 和一组源汇 (S, T) ，如果图 G 的进行收缩操作的最后一步为缩 2 度点操作，那么图 G 存在一个点 $x \notin S, T$ 使得若删除点 x ， S, T 不连通；否则无论删除图中任意一个点 $x \notin S, T$ ， S, T 均连通。

证明. 如果图 G 的进行收缩操作的最后一步为缩 2 度点操作，那么如果删除被它缩掉的点（显然不为 S 或 T ）， S, T 就不连通。否则图 G 的进行收缩操作的最后一步为叠合重边操作，那么设根结点的两个儿子（也是被叠合的一对重边对应的点）为 u, v ，如果删除任意一个点 $x \notin S, T$ ，如果 x 在 u 的内部图中， S, T 就可以通过 v 的内部图连通，反之同理。 \square

例 3. 带源汇串并联图计数⁴

有 n 个点的简单图（无重边无自环） $G = (V, E)$ ，编号为 $1, 2, \dots, n$ ，和图 G 的一组源汇 (S, T) 组成的三元组 (G, S, T) 。求共有多少个不同的三元组，对 998244353 取模。

两个三元组 (G, S, T) 和 (G', S', T') 不同当且仅当无序点对 (S, T) 与 (S', T') 不同，或图 G 中存在一条边 (x, y) 而图 G' 中不存在，或图 G' 中存在一条边 (x, y) 而图 G 中不存在。

$$2 \leq n \leq 10^5$$

⁴题目来源：原创

由引理 7.3 可得, 不存在三元组 (G, S, T) 使得图 G 的进行收缩操作的最后一步既可以为缩 2 度点操作, 也可以为叠合重边操作。

设 f_i 为 $i+2$ 个点 (即不包括源汇 i 个点), 最后一步为缩 2 度点操作, 源汇为 $(1, 2)$ 的三元组数 $(G, 1, 2)$, g_i 为 $i+2$ 个点, 最后一步为叠合重边操作的三元组数 $(G, 1, 2)$ 。

特别地, 我们设 $f_0 = 0, g_0 = 1$, 也就是说仅有两个点, 且这两个点之间有且仅有一条边连接的图归属于 g 。

记它们的指数型生成函数:

$$F(x) = \sum_{i=0}^{+\infty} \frac{f_i x^i}{i!}, G(x) = \sum_{i=0}^{+\infty} \frac{g_i x^i}{i!}$$

对于转移, 考虑将这张图“不断展开”, 形式化地说:

1、对于 f_i , 其最后一步为缩 2 度点操作。首先对最后缩成的边 x 执行“不断展开”, 将它“展开”成收缩前的样子, 它会展开成两条新的边的一个新的点, 设两条新的边分别为 x_1, x_2 。若 x_1 也是由缩 2 度点操作产生的, 那么对它继续执行“不断展开”操作, 递归下去; 否则将 x_1 原样保留。对 x_2 的操作同理。这样“不断展开”后原图就变成了 $j+1$ 个点和 j 条边串成的链 (j 为任意大于 1 的正整数), 其中每一条边都是初始就存在的或是由叠合重边操作产生的。

2、对于 g_i , 其最后一步为叠合重边操作。也是类似操作: 首先对最后缩成的边 x 执行“不断展开”, 将它“展开”成收缩前的样子, 它会展开成一对新的重边, 设它们分别为 x_1, x_2 。若 x_1 也是由叠合重边操作产生的, 那么对它继续执行“不断展开”操作, 递归下去; 否则将 x_1 原样保留。对 x_2 的操作同理。这样“不断展开”后原图就变成了 2 个点和 j 条重边形成的图, 其中每一条边都是初始就存在的或是由缩 2 度点操作产生的。

继续分析不断展开后这 j 条边的内部图, 这些内部图都是串并联图, 更具体地:

若 f_i 展开成了 $j+1$ 个点和 j 条边, 除源汇外有 $j-1$ 个点。其中每一条边都是初始就存在的或是由叠合重边操作产生的, 也就是说其中每一条边的内部图的源汇都已经固定为这条边的端点, 其他部分对应一个 g_x , 且这些内部图是有序拼接的。更具体地, 有:

$$\begin{aligned} F(x) &= \sum_{j=2}^{+\infty} x^{j-1} G^j(x) \\ &= \frac{xG^2(x)}{1-xG(x)} \end{aligned}$$

若 g_i 展开成了 2 个点和 j 条边, 这两个点分别为 a, b 。其中每一条边都是初始就存在的或是由叠合重边操作产生的, 也就是说其中每一条边的内部图的源汇都为这 2 个点, 其他部分对应一个 f_x , 但有可能有一条边对应初始就存在的边 (a, b) 。这些内部图是无序拼接的。更具体地, 有:

$$\begin{aligned}
 G(x) &= 1 + \sum_{j=1}^{+\infty} \frac{F(x)^j}{j!} + \sum_{j=2}^{+\infty} \frac{F(x)^j}{j!} \\
 &= \sum_{j=0}^{+\infty} \frac{2F(x)^j}{j!} - F(x) - 1 \\
 &= 2e^{F(x)} - F(x) - 1
 \end{aligned}$$

利用上述两个式子进行多项式牛顿迭代即可得出 $F(x), G(x)$ 的前 n 项, 而最终答案即为 $\binom{n}{2} \times (f_{n-2} + g_{n-2}) = \frac{n!}{2} \times ([x^{n-2}]F(x) + [x^{n-2}]G(x))$ 。

时间复杂度 $O(n \log n)$ 。

8 命题思路

吴作同在 IOI2019 的候选队论文《〈公园〉命题报告》中引入了“广义串并联图”的概念, 他给出的一种处理方法是通过收缩操作不断缩小问题规模并进行动态规划。同时, 为了支持快速修改参数, 也提到了建立表达式树⁵来将动态规划转化为树形结构然后链分治的方法。但是这种做法存在一定的局限性: 其只能用于处理较为简单的规划类问题。于是我想, 广义串并联图是否存在更好的性质。

我观察了收缩操作的逆操作的性质, 发现将收缩过程中的一个点或一条边进行逆操作后会展开成一个子图, 而缩 2 度点操作就是将两张这种图“串联”起来, 叠合重边操作就是将两张这种图“并联”起来。进一步地, 每一个串并联树上结点都对应着一个“内部图”, 而这个“内部图”与外界有连接的部分至多仅有两个点, 这一性质与树上的“子树”的性质十分相似。上述的动态规划本质上是将整个内部图的信息压缩到这两个点上。

这个新的发现不仅能够设计动态规划时给予更多的启示, 让动态规划的状态更加直观, 还能使得对于广义串并联图的分析不再局限于规划类问题。

接着我分析了一些树上和仙人掌上的经典问题, 考虑它们是否能通过这个性质扩展到广义串并联图上, 发现多源最短路和点分治类问题能够很好地被这个新的发现解决, 点分治类问题是强于多源最短路的, 于是我将点分治作为最终的解法, 将多源最短路作为子任务 6, 命制出了《逛公园》。

在子任务设置方面, 子任务 1,2,4 是常见的朴素算法, 而子任务 3,5 则给选手提供一个解决问题的关键方向———点分治。子任务 6 则是弱于正解的多源最短路问题, 也为选手提供另一个方向———收缩操作和动态规划。最终结合两个算法及内部图和外部图的相关性质即可得到子任务 7 的正解, 获得满分。

9 致谢

感谢中国计算机学会提供学习和交流的平台。

⁵即上文的“串并联树”

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高闻远提供的指导与帮助。
感谢福州一中的各位学长带给我的启发和指导。
感谢陈俊锟学长，吴作同学长为本文提供思路。
感谢福州一中的各位学长带给我的启发和指导。
感谢福州一中信息组的同学们为本文审稿。
感谢父母的养育之恩。

参考文献

- [1] 吴作同,《〈公园〉命题报告》,IOI2019 中国国家候选队论文
- [2] 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
- [3] Wikipedia, Series-parallel graph
- [4] Wikipedia, Biconnected component
- [5] 机械工业出版社,《线性代数》

浅析一些维护二维点的算法

中山纪念中学 林立

摘要

本文研究的是有矩形修改和矩形查询的维护二维点的经典问题，分析了三种不同的解法： $2-d$ 树、定期重构、平面分块，比较它们的理论复杂度以及实际测试中的耗时。并研究了该问题的强制在线动态加点版本，分析了两种不同的解法： $2-d$ 树、平面分块，比较它们的理论复杂度。虽然平面分块比 $2-d$ 树的理论时间复杂度要劣，但是实验中运行时间前者优于后者。

1 引言

本文第二节介绍了一道维护二维点的经典问题，第三、四、五节分别介绍了对于该经典问题的 $2-d$ 树解法、定期重构解法、平面分块解法，在第六节中比较这三种解法的优劣并将其拓展到 k 维，同时研究了有其他不同操作类型的维护二维点问题。第七节介绍了经典问题的强制在线动态加点版本及解法，并比较了几种方法的优劣。

2 经典问题

本节介绍一道经典维护二维点问题，在接下来的三节中将介绍三种不同的解法： $2-d$ 树、定期重构、平面分块。

2.1 问题描述

二维平面上有 n 个点，第 i 个点是 (i, p_i) ，初始权值是 w_i ，其中 p 是一个大小为 n 的排列。

一共要进行 m 次操作，操作有以下两种：

- 1 $l r d u k b$ 表示将在左下角是 (l, d) 右上角是 (r, u) 这个矩形内的点的权值乘 k 加 b 。
 - 2 $l r d u$ 表示询问左下角是 (l, d) 右上角是 (r, u) 这个矩形内的点的权值和对 2^{64} 取模。
- $1 \leq n, m \leq 10^5$

3 2-d 树做法

这道经典问题可以使用 2-d 树来解决。

接下来就先介绍 2-d 树，然后再讲述如何使用 2-d 树解决本题。

3.1 2-d 树简介

2-d 树就是 $k-d$ 树在维护二维点时的名称。

2-d 树就是一棵二叉树，其中每个叶子节点都是 2 维点。而每个非叶子节点视为隐式地将一个二维平面切分为两半。这个非叶节点的左子树表示左半平面上的点，而右子树表示右半平面上的点。

3.2 2-d 树的创建方法

有许多种建 2-d 树的方法，这里选取了一种相对常见的方法：

每次都是给出一个点集，然后建出这个点集的 2-d 树，一开始将 n 个点的点集递归下去，每递归到一个点集，就按如下方法顺序判断：

- 当点集中没有点时，建出一棵空的 2-d 树，直接返回。
- 当点集中只有一个点时，建出一棵只有该点的 2-d 树，直接返回。
- 当点集中有超过一个点时，需要选取一维来切分点集，而维度是循环地选择来切分¹，这里假定选择了用 x 坐标。

取出这个点集中 x 坐标是中位数的点²，接着将这个点作为切分点，将点集中的点分为 x 坐标小于切分点的和大于的。然后将小于的和大于的点集分别递归下去建出对应的 2-d 树，并且将这两棵 2-d 树分别设为切分点的左子树与右子树。

而其中选择中位数的点中 C++ 中可以使用 `nth_element` 函数。

对于一段长度为 n 的数组使用 `nth_element` 函数是 $O(n)$ 的时间，所以创建 2-d 树的时间复杂度是 $T(n) = 2T(n/2) + O(n) = O(n \log n)$ 。

3.3 2-d 树上的修改和查询

由于有矩形修改与矩形查询，2-d 树上每个点需要多维护一些信息。

每个点需要维护最小包含该点整个子树的矩形，子树内的信息和以及懒标记。

¹例如父亲节点选择了用 x 坐标，那么这次选择用 y 坐标，否则选择用 x 坐标

²如果有多个相同，将 y 坐标作为第二关键字

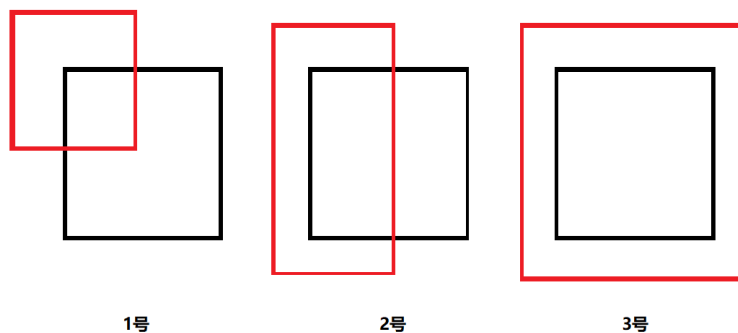
为了方便叙述，以下的“操作矩形”都是指当前的修改或询问的矩形， $2-d$ 树上的一个点的“最小包含矩形”是指最小的能够包含这个点子树内所有点的矩形。

而矩形修改/询问时，从 $2-d$ 树的根节点开始递归，每递归到一个点，就按如下方法顺序判断：

- 如果最小包含矩形与操作矩形不交，直接退出
- 如果操作矩形包含最小包含矩形，那么直接打上标记/询问，然后退出
- 递归进左子树与右子树

注意需要下传标记，以及判断当前节点是否也在操作矩形内。

3.4 $2-d$ 树上修改和查询操作的时间复杂度分析



如图所示将对应的操作矩形如上命名，其中红色矩形代表操作矩形，而黑色矩形代表当前递归到的 $2-d$ 树上的点的最小包含矩形。

考虑创建 $2-d$ 树时是先用 x 坐标切分，最坏情况下操作矩形与第一次切分线和第二次切分线都有交，那么这样就会分成四个 1 号操作矩形。

1 号操作矩形，最坏情况下两次切分会分成一个 1 号操作矩形，一个 3 号操作矩形以及两个 2 号操作矩形。

2 号操作矩形，最坏情况下两次切分只会分成两个 2 号操作矩形和 3 号操作矩形。

3 号操作矩形，是可以直接 $O(1)$ 打标记/询问。

令 $T(n)$ 为对大小为 n 的 $2-d$ 树进行一次 1 号矩形操作时间复杂度， $T_2(n)$ 为进行一次 2 号矩形操作的时间复杂度。

根据上述最坏情况下的切分方法，有

$$\begin{aligned} T_2(n) &= 2T_2(n/4) + O(1) \\ &= \sum_{i=0}^{\lfloor \log_4(n) \rfloor} O(2^i) \\ &= O(\sqrt{n}) \end{aligned}$$

$$\begin{aligned} T(n) &= T(n/4) + 2T_2(n/4) + O(1) \\ &= T(n/4) + O(\sqrt{n}) \\ &= \sum_{i=0}^{\lfloor \log_4(n) \rfloor} O\left(\sqrt{\frac{n}{4^i}}\right) \\ &= \sum_{i=0}^{\lfloor \log_4(n) \rfloor} O\left(\frac{\sqrt{n}}{2^i}\right) \\ &= O(\sqrt{n}) \end{aligned}$$

综上所述一次矩形操作的时间复杂度为 $O(\sqrt{n})$ 。

3.5 2-d 树做法

因为标记是可以被表示成 $kx + b$ 这样的形式，并且标记满足有结合律且能 $O(1)$ 计算，所以建出 2-d 树后按照 3.2 节所述方法做即可。

时间复杂度 $O(n \log n + m\sqrt{n})$ 。

4 定期重构

设阈值为 B ，考虑每 B 个操作就定期重构一次。

把 n 个点按照 B 个操作矩形的坐标分块，这样每个块对于这 B 个操作都是完全在操作矩形内或者完全不在操作矩形。对于每个块记录块内点的信息和以及懒标记，然后对于每个操作就直接打标记/询问其包含的所有块。由于块数最多为 $4B^2$ ，所以每个操作的复杂度为 $O(B^2)$ 。

B 个操作操作完之后，再将每个块的懒标记下传给对应的点上。

时间复杂度 $O(n + \frac{mn}{B} + mB^2)$ 。

当 B 取 $n^{1/3}$ 时最优，此时时间复杂度为 $O(n + mn^{2/3})$ 。

5 平面分块

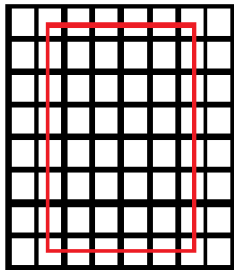


图 1. 一次矩形操作示意图

设阈值为 B ，考虑直接将 $n \times n$ 的平面分成 $B \times B$ 块，每个块都维护该块内的点的信息和以及懒标记。

如图 1 所示，其中红色矩形为操作矩形，黑色小矩形为平面分块后的每一块，每次操作都会包含一些完整的块以及一些不完整的块，对应的做法如下：

- 对于被完整包含的块，直接打标记/询问，由于总块数是 B^2 ，所以这里最多对 B^2 个块进行操作，这里复杂度为 $O(B^2)$ 。
- 对于不被完整包含的块，枚举每个块中的点，如果在操作矩形内，那么就做对应的操作。从图 1 可以看出这些块最多同属于两行整块与两列整块，又因为一行或者一列只有一个点，所以最多有 $4\frac{n}{B}$ 个点在这些块中，这里的复杂度为 $O(\frac{n}{B})$ 。

综上所述，总时间复杂度 $O(n + m(B^2 + \frac{n}{B}))$ 。

当 B 取 $n^{1/3}$ 时最优，此时时间复杂度为 $O(n + mn^{2/3})$ 。

6 算法小结

6.1 实验结果

对于 $2-d$ 树、定期重构、平面分块三种不同的解法，构造了不同类型的数据，测试了它们在这些数据下的运行时间。

在随机数据下平面分块算法远优于 $2-d$ 树，而定期重构因为重构的时间消耗稳定且较多导致随机数据下与 $2-d$ 树的运行时间差不多。

在针对 $2-d$ 树的数据下， $2-d$ 树的极大常数的显现出来，尽管 $2-d$ 树的理论时间复杂度是 $O(n \log n + m \sqrt{n})$ ，其在实际测试中需要足足 4s，远远大于定期重构和平面分块解法的运行时间。

	随机数据	针对 $2-d$ 树	针对平面分块	理论复杂度
$2-d$ 树	1.809s	4.125s	4.001s	$O(n \log n + m \sqrt{n})$
定期重构	1.834s	2.028s	1.993s	$O(n + mn^{2/3})$
平面分块	0.628s	1.734s	1.853s	$O(n + mn^{2/3})$

不同数据下不同做法的运行时间（数据规模： $n = m = 10^5$ ）

根据表格可以发现，在 $n = m = 10^5$ 的绝大多数情况下，平面分块算法都比 $2-d$ 树要优。

6.2 拓展性

6.2.1 拓展到 k 维

拓展到 k 维，三种解法都可行。

用 $2-d$ 树变成用 $k-d$ 树，而 $k-d$ 树就是在选择切分维度时从两维轮换变成 k 维轮换，维护方法和时间复杂度分析差不多，时间复杂度为 $O(n \log n + mn^{1-1/k})$ 。

定期重构就是每 $n^{1/(k+1)}$ 操作重构一次，维护方法和时间复杂度分析与二维差不多，时间复杂度为 $O(n + mn^{1-1/(k+1)})$ 。

平面分块就变成 k 维空间分块，就是每一位都分成 $n^{1/(k+1)}$ 块，维护方法和时间复杂度分析与二维差不多，时间复杂度为 $O(n + mn^{1-1/(k+1)})$ 。

由于 $k-d$ 树的大常数，在 k 维问题上应该还是 k 维空间分块算法实际时间上优秀。

6.2.2 强制在线

$2-d$ 树和平面分块解法都可行，定期重构就无法解决强制在线的一类问题。

在第七节介绍了该问题的强制在线动态加点的版本， $2-d$ 树和平面分块都可以解决该版本的问题，并且主要复杂度没有变化。

6.2.3 矩形最值操作

问题描述如下：

二维平面上有 n 个点，第 i 个点是 (i, p_i) ，初始权值是 w_i ，其中 p 是一个大小为 n 的排列。

一共要进行 m 次操作，操作有以下两种：

1 $l r d u x$ 表示将在左下角是 (l, d) 右上角是 (r, u) 这个矩形内的点的权值与 x 取 \max 。

2 $l r d u$ 表示询问左下角是 (l, d) 右上角是 (r, u) 这个矩形内的点的权值和对 2^{64} 取模。

$2-d$ 树做法：由于 $2-d$ 树的形态与线段树差不多，所以可以使用参考文献 [2] 中的方法，时间复杂度 $O(n \log n + m \sqrt{n})$ 。

定期重构做法：

- 重构后每个块内的点需要按照权值大小排序，要维护每个块的最小值，最小值的个数和非最小值总和。
- 每次矩形最值时，对于每个涉及到的块，修改其的最小值，当操作的 x 大于等于该块的严格次小值时，该块的最小值个数和非最小值总和就会改变，因为块内的点已经按权值排好序了，所以可以每个块用个指针指向严格次小值的位置，每次修改就判断是否要移动该指针。指针只会朝一个方向移动，每个块的变化量最多为块内点数，那么变化总量最多为 n 。
- 询问矩形和就直接枚举被包含的块求和即可。

其中排序部分可以使用以下方法：

- 使用快排，那么时间复杂度为 $O(n + m(n \log n)^{2/3})$ 。
- 由于总共最多只有 $n + m$ 不同的值，可以一开始离散，然后排序时就可以使用多关键字桶排，时间复杂度是 $O(n + m(n \log_n(n + m))^{2/3})$ 。
- 直接使用多关键字桶排，时间复杂度是 $O(n + m(n \log_n \max w)^{2/3})$ 。

平面分块做法：每块维护的东西都与定期重构做法一样，所以只用考虑不被修改矩形完整包含的块。

对于这种块只用考虑一次修改的影响对于权值顺序的影响，因为其他的东西都可以在块内点数常数倍时间内解决。由于修改只是取 \max ，所以可以将被修改的点在权值顺序去掉，然后再加入进去，这样可以在块内点数常数倍时间内解决。

每次修改不完整的块总共能变化量只会多出 $O(n^{2/3})$ ，所以时间复杂度不变，仍是 $O(n + mn^{2/3})$ 。

6.2.4 矩形最值 + 矩形加

问题描述如下：

二维平面上有 n 个点，第 i 个点是 (i, p_i) ，初始权值是 w_i ，其中 p 是一个大小为 n 的排列。

一共要进行 m 次操作，操作有以下三种：

1 $l r d u x$ 表示将在左下角是 (l, d) 右上角是 (r, u) 这个矩形内的点的权值与 x 取 \max 。

2 $l r d u x$ 表示将在左下角是 (l, d) 右上角是 (r, u) 这个矩形内的点的权值加上 x 。

$3\ l\ r\ d\ u$ 表示询问左下角是 (l, d) 右上角是 (r, u) 这个矩形内的点的权值和对 2^{64} 取模。

$2-d$ 树做法：由于 $2-d$ 树的形态与线段树差不多，所以可以使用参考文献 [2] 中的方法，时间复杂度 $O(n \log n + m \sqrt{n} \log n)$ ，但是根据参考文献 [2] 中所述暂时无法构造能够达到该复杂度的数据。

定期重构做法：同 6.2.3 中所述，不过由于多了矩形加，所以排列只能用多关键字桶排，时间复杂度 $O(n + m(n \log_n \max w)^{2/3})$ 。

平面分块做法：同 6.2.3 中所述，不过需要多处理矩形加时不被完整包含的块，由于本身每个块都维护了该块内点的权值顺序，每次矩形加就将修改的点按权值顺序提出来，那么两个有序的序列就能直接归并起来就能得到矩形加后的权值顺序序列。

时间复杂度不变，仍是 $O(n + mn^{2/3})$ 。

7 经典问题 2

比经典问题 1 多了动态加点操作以及强制在线。

7.1 问题描述

一共要进行 n 次加点和 n 次操作，第 i 次操作在第 i 次加点后，强制在线。

操作有以下两种：

$1\ l\ r\ d\ u\ k\ b$ 表示将在左下角是 (l, d) 右上角是 (r, u) 这个矩形内的点的权值乘 k 加 b 。

$2\ l\ r\ d\ u$ 表示询问左下角是 (l, d) 右上角是 (r, u) 这个矩形内的点的权值和。

$1 \leq n \leq 10^5$

7.2 $2-d$ 树 + 二进制分组

由于多了强制在线，就不能够直接预先建好 $2-d$ 树，此时考虑使用二进制分组。

我们将一个数拆分成 2 的次幂从大到小的形式。示例：

$$21 = 16 + 4 + 1$$

$$22 = 16 + 4 + 2$$

$$23 = 16 + 4 + 2 + 1$$

$$24 = 16 + 8$$

我们不妨用上述拆分方法对于加点操作分组，比如说，对于前 21 个加的点，我们会将前 16 个点分为第一组，第 17~20 的点分为第二组，第 21 个点单独作为第 3 组。我们对于每组

都使用 $2-d$ 树来维护。显然只会分出最多 $O(\log n)$ 组，修改和询问就直接对于每一组都做一遍，而最坏情况下是每一个 2 的次幂都有，此时时间复杂度为 $O(\sum_{i=0}^{\lfloor \log_2 n \rfloor} O(\sqrt{2^i})) = O(\sqrt{n})$ 。

而多加一个点时怎么做呢？事实上，我们只用考虑增加一个点之后原来的分组与新的分组的区别，而显然是一些分组和新加入的点合并成了一个新的分组。那么我们只需要将那些不存在了的分组下传 $2-d$ 树上的标记，得到每个点当前真实的权值，然后将这些点和新的点再建出一棵 $2-d$ 树。

我们来分析一下时间复杂度。我们考虑我们重建的所有组的元素总数。可以发现，当加入第 i 个点时，重建的元素个数是 $\text{lowbit}(i)$ ，也就是 i 的二进制表示下最右侧的 1 所代表的权值³。

可以推出重建需要的时间为：

$$\begin{aligned} & \sum_{i=1}^n O(\text{lowbit}(i) \log \text{lowbit}(i)) \\ &= \sum_{i=0}^{\lfloor \log_2 n \rfloor} \left\lfloor \frac{n}{2^{i+1} + 0.5} \right\rfloor \times O(2^i \times i) \\ &= \sum_{i=0}^{\lfloor \log_2 n \rfloor} O(n \times i) = O(n \log^2 n) \end{aligned}$$

最后总的时间复杂度为 $O(n \log^2 n + n \sqrt{n})$ 。

7.3 平面分块

由于多了强制在线和动态加点，不能够直接将平面切分好，所以考虑动态维护平面的切分。

我们考虑每一次加入一个点，就是在两维坐标各加入当前点的坐标的两维，然后将这个点加入到其对应的整块中。那么两维各维护一个块状链表，每次将一个点加入到对应的块上，如果当前块的大小等于阈值，那么将这个块均匀分裂成两半，对应的将平面上的整块分裂成两块。

如果我们将阈值设为 $O(n^{2/3})$ ，那么只会发生 $O(n^{1/3})$ 次分裂，每次分裂操作只会涉及该块内 $O(n^{2/3})$ 个点以及 $O(n^{1/3})$ 个平面上整块，所以分裂的总复杂度是 $O(n)$ 。而修改与查询操作与沿用第五节的做法即可。

总时间复杂度为 $O(n^{5/3})$ 。

7.4 算法小结

即使是在强制在线 + 动态加点的问题上，平面分块算法依旧比使用 $2-d$ 树时间上优秀。

³示例：24 = (11000)₂，所以 $\text{lowbit}(24) = (1000)_2 = 8$

8 总结

本文围绕一个维护二维点的经典问题，分析三种不同的解法： $2-d$ 树、定期重构、平面分块，比较了它们的理论时间复杂度和在试验中运行时间。

$2-d$ 树虽然是一种常见的数据结构，但是因为其极大的常数为人诟病。而平面分块算法能够在维护二维点集上优于 $2-d$ 树，即使理论复杂度更大，但因为常数更小也不失为一种好的做法。希望对读者有所帮助。

参考文献

- [1] k-d tree 的维基百科, https://en.wikipedia.org/wiki/K-d_tree
- [2] 吉如一,《区间最值操作与历史最值问题》, 2016 年信息学奥林匹克中国国家队候选队员论文集
- [3] 许昊然,《浅谈数据结构题的几个非经典解法》, 2013 年信息学奥林匹克中国国家队候选队员论文集

浅谈超现实数与不平等博弈

马耀华

摘要

本文介绍了超现实数与不平等博弈的相关理论，建立了博弈的数学模型，并引入公平博弈理论、热博弈理论、全部小博弈理论，同时介绍了相关理论在一些具体题目上的应用。

1 前言

公平博弈理论在目前的 OI 界已经非常普及，也出现了大量的题目。而近年来，更一般化的不平等博弈理论也越发受到 OI 界重视，在清华集训和若干国外比赛出现了相关的题目，甚至还由杜瑜皓学长在 WC 进行了分享。然而，由于种种原因，相关的基础理论在 OI 界还不够普及。本文力求从基础理论方面普及不平等博弈理论，希望能激发选手们对相关内容研究的热情。

本文第 2 节介绍了超现实数和博弈的定义，以及超现实数的一些基本性质；第 3 节介绍了博弈的一些基本性质；第 4 节介绍了公平博弈理论以及其在一般博弈理论中的推广；第 5 节介绍了热博弈；第 6 节介绍了全部小博弈。

2 超现实数与博弈

2.1 超现实数定义

我们先来定义超现实数。为了方便，在本文中，我们提到的“数”默认指超现实数。

定义 2.1. 令 L, R 为两个任意的数集合，且 L 中不存在 $\geq R$ 中某个元素的元素，则 $\{L|R\}$ 也是一个数。

所有的数都是由上面的构造得到的。

定义 2.2. 我们用 x^L 指代一个数 $x = \{L|R\}$ 的 L 中任意元素，用 x^R 指代 R 中任意元素。此外，我们还经常使用另一个记号 $x = \{a, b, c, \dots | d, e, f, \dots\}$ ，这里 $L = \{a, b, c, \dots\}, R = \{d, e, f, \dots\}$ 。

上述定义中，我们使用了 \geq ，但事实上我们还没有定义 \geq 。

定义 2.3.

1. $x \geq y$ 当且仅当不存在 $x^R \leq y$ 且不存在 $x \leq y^L$ 。
2. $x \leq y$ 当且仅当 $y \geq x$ 。

这里的 \geq, \leq 都是递归定义。

定义 2.4.

1. $x = y$ 当且仅当 $x \geq y$ 且 $y \geq x$ 。
2. $x > y$ 当且仅当 $x \geq y$ 且 $y \not\geq x$ 。
3. $x < y$ 当且仅当 $y < x$ 。

这样定义了超现实数的大小关系。

定义 2.5.

1. $x + y = \{x^L + y, x + y^L | x^R + y, x + y^R\}$ 。
2. $-x = \{-x^R | -x^L\}$ 。
3. $x - y = x + (-y)$ 。
4. $xy = \{x^L y + x y^L - x^L y^L, x^R y + x y^R - x^R y^R | x^L y + x y^R - x^L y^R, x^R y + x y^L - x^R y^L\}$ 。

这样就定义了超现实数的加减法，加法逆元，乘法。

2.2 数的实例

上述定义都是递归定义，而一开始我们没有任何数。

不过，我们可以取 $L = R = \emptyset$ ，这样我们得到了一个数 $\{\}$ ，我们称它为 0。容易验证，我们有 $0 \geq 0, 0 \leq 0, 0 = 0$ ，以及 $0 \not> 0, 0 \not< 0, -0 = 0$ 。

在图 1 的二叉树中，0 位于第 1 层。

接着，我们可以取 $L = 0, R = \emptyset$ ，这样我们得到了一个数 $\{0\}$ ，我们称它为 1。容易验证，我们有 $1 \geq 0$ 和 $1 > 0$ ，但没有 $0 \geq 1$ 和 $0 > 1$ 。我们还可以得到 $-1 = \{-0\} = \{\emptyset\}$ 。同样容易验证， $0 + 1 = 1 + 0 = 1, 0 + (-1) = (-1) + 0 = -1$ 。

我们还期望有 $1 + (-1) = 0$ ，这满足我们通常意义下的运算性质，那这是否成立呢？显然，有 $1 + (-1) = \{0 + (-1) | 1 + 0\} = \{-1 | 1\}$ ，看起来并不是 0。但 $1 \not\leq 0, 0 \not\geq -1$ ，于是我们有

$\{-1|1\} \geq 0$ 且 $0 \geq \{-1|1\}$, 这样, 按我们的定义, 确实有 $\{-1|1\} = 0$! 这里出现了超现实数与实数的最大区别: 两个形式不同的数, 它们的值可能是相等的。

以上的数在二叉树中位于第 2 层。

接着, 我们可以取 $L = 0, R = 1$, 构造出 $\frac{1}{2} = \{0|1\}$; 取 $L = -1, R = 0$, 构造出 $-\frac{1}{2} = \{-1|0\}$; 取 $L = 1, R = 0$, 构造出 $2 = \{1\}$; 取 $L = 0, R = -1$, 构造出 $-2 = \{0|-1\}$ 。同样可以验证, 这些数满足我们期望的通常意义下的一切性质。

以上的数在二叉树中位于第 3 层。

进一步, 二叉树的有限层中包含了所有形如 $\frac{p}{2^q}$ 的有理数 (p, q 是整数)。

那么 $\frac{1}{3}$ 呢? 我们可以类似实数定义的 Dedekind 分割, 用 $\frac{p}{2^q}$ 的无穷序列来逼近 $\frac{1}{3}$, 从而得到 $\frac{1}{3} = \{0, \frac{1}{4}, \frac{5}{16}, \frac{21}{64}, \dots | \frac{1}{2}, \frac{3}{8}, \frac{11}{32}, \frac{43}{128}, \dots\}$ 。类似地, 我们可以构造出所有实数。

我们甚至能得到无穷大 $\omega = \{0, 1, 2, \dots\}$ 和无穷小 $\frac{1}{\omega}$, 不过与本文主旨无关, 下面不会讨论。

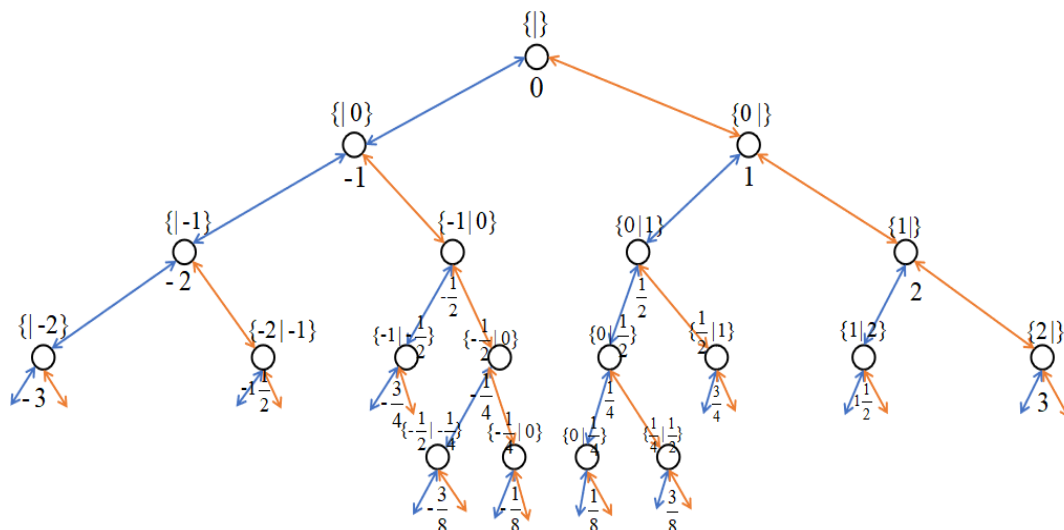


图 1: 超现实数的二叉树

2.3 博弈定义

类似地, 我们可以定义博弈。¹

定义 2.6. 令 L, R 为两个任意的博弈集合, 则 $\{L|R\}$ 也是一个博弈。

所有的博弈都是由上面的构造得到的。

¹本文中“博弈”的定义与通常中文语境中不同, 具体见下。

可以发现，博弈与超现实数的定义区别仅是我们去掉了“ L 中不存在 $\geq R$ 中某个元素的元素”这一限制。于是我们可以使用与超现实数相同的符号来描述博弈，同样定义博弈的大小关系，加法，加法逆元，乘法等。

两个博弈 G 和 H 若满足 $G = H$ ，我们只称它们值相等，而它们形式相等需要 L 和 R 中每个元素能对应。

我们可以将超现实数看作一种特殊博弈，称一个博弈 G 的值是数，若存在超现实数 x 使 $G = x$ 。

值得注意的是，这里我们所描述的博弈，仅仅是一个形式符号，还没有组合意义。我们将在下一节中详细分析它的组合意义。

2.4 基本性质

刚刚我们的例子中，隐含使用了实数的记号，那我们自然希望超现实数保持实数的性质。下面我们指出一些超现实数的性质，而其中的大部分性质是博弈同样具有的。

定理 2.1. 对一切数和博弈 x ，有：

1. $x \neq x^R$ 。
2. $x^L \neq x$ 。
3. $x \geq x$ 。
4. $x \leq x$ 。
5. $x = x$ 。

定理 2.2.

1. 若数或博弈 x 满足 $x \geq y$ 且 $y \geq z$ ，则 $x \geq z$ 。
2. 对于数 x ，有 $x^L < x < x^R$ 。
3. 对于任意两个数 x, y ，有 $x \geq y$ 或 $y \geq x$ 中至少一者成立。

(1) 告诉我们，对数和博弈来说， \geq 关系具有传递性，结合前面的 $x \geq x$ （反身性），以及 $x \geq y$ 且 $y \geq x$ 则有 $x = y$ （反对称性），可知 \geq 关系是一个偏序关系。

进一步，(3)告诉了我们对于超现实数来说， \geq 是一个全序关系。于是，超现实数是有序的。

定理 2.3. 对一切数和博弈 x, y, z ，有：

1. $x + 0 = x$ 。
2. $x + y = y + x$ 。
3. $(x + y) + z = x + (y + z)$ 。

也即，加法具有交换律，结合律，且 0 是加法单位元。

定理 2.4. 对一切数和博弈 x, y, z ，有：

1. $x \geq y$ 当且仅当 $x + z \geq y + z$ 。
2. $x > y$ 当且仅当 $x + z > y + z$ 。
3. $x = y$ 当且仅当 $x + z = y + z$ 。

也即，加法是保序的。

定理 2.5. 对一切数和博弈 x, y ，有：

1. $-(x + y) = (-x) + (-y)$ 。
2. $-(-x) = x$ 。
3. $x + (-x) = 0$ 。

定理 2.6.

1. 若 x, y 是数， $x + y$ 也是数。
2. 若 x 是数， $-x$ 也是数。

也即，超现实数关于加法和加法逆元运算是封闭的。

定理 2.7. 对一切数和博弈 x, y, z ，有：

1. $x0 = 0$ 。
2. $x1 = x$ 。
3. $xy = yx$ 。
4. $(xy)z = x(yz)$ 。
5. $(-x)y = x(-y) = -(xy)$ 。
6. $(x + y)z = xz + yz$ 。

也即, 0 是乘法零因子, 1 是乘法单位元, 且乘法满足交换律, 结合律, 与加法逆元交换, 且满足关于加法的分配律。

定理 2.8. 对于数 x, y , 我们有 xy 也是数, 且:

1. 若 $x = y$, z 是数, 则 $xz = yz$ 。
2. 若 $x, y > 0$, 则 $xy > 0$ 。
3. 若 $x \neq 0$, 则存在唯一的数 $y \neq 0$, 使 $xy = 1$, 记 $y = x^{-1}$ 为 x 的乘法逆元。

限于篇幅, 上述定理证明略去。

结合上述所有定理, 我们事实上得到了下述定理:

定理 2.9. 超现实数形成一个有序域, 记作超现实数域 **No**。

进一步地, 我们还可以证明下面的定理:

定理 2.10. 实数域 \mathbb{R} 是超现实数域 **No** 的子域。也即, 我们可以任意对值是实数的超现实数 (或博弈) 进行我们熟悉的实数运算, 而不必担心得到相异结果。

2.5 简单性法则

定理 2.11. 对于数 $x = \{x^L | x^R\}$, 若存在数 z 使得满足 $x^L \not\leq z \not\leq x^R$ 的限制, 且对某个 z 的形式 $z = \{z^L | z^R\}$ 不存在 z^L 和 z^R 满足相同限制, 则 $x = z$ 。

证明. 先证明 $x \geq z$ 。这等价于 $x^R \not\leq z$ 与 $x \not\leq z^L$ 。前者是显然满足的, 而后者由于 z^L 不满足相同限制, 且 $z^L < z$, 于是必有 $x^L \geq z^L$, 因此同样满足。同理可证 $z \geq x$, 那么就有 $x = z$ 。□

这引出了下面的重要推论:

推论 2.1. (简单性法则) 对于一个数 $x = \{x^L | x^R\}$, 若存在至少一个数 z 满足 $x^L < z < x^R$, 则其中最简单的数即为 x 的值。这里的“最简单”可以理解为在二叉树中所在层数最低的 (显然是唯一的)。

例如, $\{1\frac{1}{4} | 2\} = 1\frac{1}{2}$ 。这是由于 $1\frac{1}{2} \in (1\frac{1}{4}, 2)$, 且 $1\frac{1}{2}$ 比同样满足在 $(1\frac{1}{4}, 2)$ 之间的其他所有数 (例如 $1\frac{3}{4}$) 更简单。

3 博弈基础

3.1 博弈的组合意义

在上一节中，我们定义了博弈，但仅仅是作为一个计算符号来使用。事实上，博弈是有着组合意义的。

定义 3.1. 一个（双人）博弈有两位玩家，分别为左玩家和右玩家。在博弈的某一状态中，左右玩家分别有若干个（可能为 0 个）可能的行动，可以转移到另一状态。两位玩家在某个固定的初始状态开始，按最优策略交替行动，了解一切游戏信息，且行动必须是确定性的。达到终止状态，不能行动的则为输家。

这个定义事实上与我们上一节的定义是相同的。也即，对于一个博弈 $x = \{L|R\}$ ，我们认为 L 是左方行动后所达到的状态集合， R 是右方行动后所达到的状态集合。于是，我们所定义的博弈的运算和大小关系，以及一切相关性性质仍然成立。

特别需要指出的是，上节中我们定义了两个博弈的加法，这也是有组合意义的。

定义 3.2. 给定两个博弈 G 和 H ，我们定义它们的和博弈 $G+H$ 是这样一个博弈：有两个子博弈 G 和 H ，两位玩家每次只能恰好在其中一个行动，不能行动的则为输家。

显然，这与我们定义的博弈的加法是相同的，同样有 $G+H = \{G^L+H, G+H^L | G^R+H, G+H^R\}$ 。

在本文中，我们只讨论最简单的有限博弈，也即有下面的额外限制：

定义 3.3. 一个有限博弈是满足下述限制的博弈：仅有有限多个状态，每个状态能转移到的状态有限，且不存在一个长度无穷的双方交替行动的序列。

两个有限博弈的和仍是有限博弈。

可以证明，一个有限博弈不可能出现平局，且博弈结果只可能是以下四种之一：

1. 左方必胜
2. 右方必胜
3. 后手必胜
4. 先手必胜

下文中，我们所指的博弈，默认均指有限博弈。

此前我们定义的博弈大小关系比较复杂，不过我们可以得到下面的等价定义，这在实践中更加简便：

定理 3.1. 对博弈 G ，有：

1. G 左方必胜当且仅当 $G > 0$ 。
2. G 右方必胜当且仅当 $G < 0$ 。
3. G 后手必胜当且仅当 $G = 0$ 。
4. 其余情况 G 先手必胜，记作 $G \parallel 0$ 。

这蕴含 G 左方后手必胜当且仅当 $G \geq 0$ ，且 G 右方后手必胜当且仅当 $G \leq 0$ 。

证明. 考虑归纳。

$G > 0$ 当且仅当存在 $G^L \geq 0$ 且任意 $G^R \not\leq 0$ 。由归纳假设，这当且仅当左方先手行动后左方必胜，右方先手行动后左方也必胜，即左方必胜。

$G < 0$ 同理。

$G = 0$ 当且仅当任意 $G^L \not\leq 0$ ，任意 $G^R \not\geq 0$ 。由归纳假设，这当且仅当左方先手行动后右方必胜，右方先手行动后左方必胜，即后手必胜。 \square

定义 3.4.

1. $G \parallel > 0$ 当且仅当 $G > 0$ 或 $G \parallel 0$ ，也即 G 左方先手必胜。 $G \parallel > 0$ 当且仅当 $G \not\leq 0$ 。
2. $G \parallel < 0$ 当且仅当 $G < 0$ 或 $G \parallel 0$ ，也即 G 右方先手必胜。 $G \parallel < 0$ 当且仅当 $G \not\geq 0$ 。

结合定理 2.4 和定理 3.1，易得以下推论：

推论 3.1. 对于任意博弈 G 和 H ，有：

1. $G + (-H)$ 左方必胜当且仅当 $G > H$ 。
2. $G + (-H)$ 右方必胜当且仅当 $G < H$ 。
3. $G + (-H)$ 后手必胜当且仅当 $G = H$ 。
4. $G + (-H)$ 先手必胜当且仅当 $G \parallel H$ 。

类似可得 $G \geq H$ ， $G \leq H$ ， $G \parallel > H$ ， $G \parallel < H$ 等的等价定义。

这在实践中非常方便，下文我们都会采取这种方式验证博弈的大小关系。

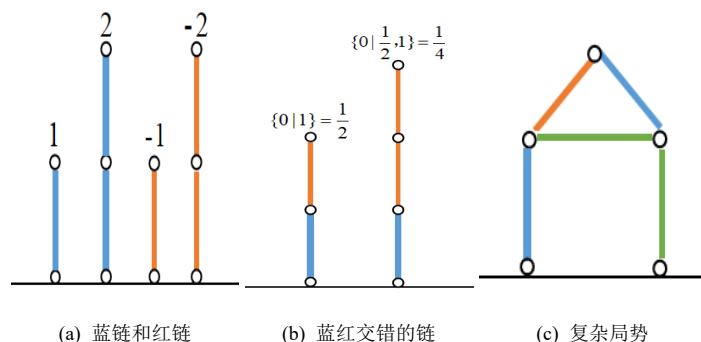


图 2: Hackenbush 局势示例

3.2 博弈的实例

我们介绍一个比较知名的博弈：**Hackenbush**（伐木游戏）。

Hackenbush 是在一个无向图上的游戏，其中某些点在地面上。边分为三类：蓝边，红边和绿边，其中蓝边只能由左方删去，红边只能由右方删去，绿边可以由双方删去。每次行动时，当前玩家需要删去一条他可删去的边，并将删去这条边后与地面不连通的连通块一并删去，不能行动的玩家判负。

图 2 给出了几个典型的 **Hackenbush** 局势：

图 2(a) 中，由 n 条蓝边组成的链博弈值为 n ，由 n 条红边组成的链博弈值为 $-n$ 。图 2(b) 中，左边底部一条蓝边，向上一条红边的链按定义博弈值为 $\{0|1\} = \frac{1}{2}$ ，类似可知道右边的链博弈值为 $\{0|\frac{1}{2}, 1\} = \frac{1}{4}$ 。图 2(c) 中给出了一个三种边混杂的复杂 **Hackenbush** 局势。

3.3 模糊博弈

我们称满足 $G \parallel 0$ 的博弈为模糊博弈。

定理 3.2.

1. 若 $G \gg 0$, $H \geq 0$, 则 $G + H \gg 0$.
2. 若 $G \ll 0$, $H \leq 0$, 则 $G + H \ll 0$.

证明.

1. $G \gg 0$ 意味着 G 左方先手必胜, $H \geq 0$ 意味着 H 左方后手必胜。那么在 $G + H$ 中, 左方先手时只需在 G 中走对应的必胜走法, 随后得到两个左方后手必胜的博弈, 且左方为后手, 显然左方必胜。
2. 与 (1) 同理。

□

一个典型的模糊博弈是仅有一条绿边的 Hackenbush，显然不论哪方砍去它后，得到了一个 0 状态，立刻获胜。我们称这样的博弈为 $* = \{0|0\}$ 。 $*$ 具有一些显然的性质，例如 $* = -*$ ， $\{*\} = 0$ ，且对于任意的正数 x ，有 $x \gg * , -x \ll *$ ，也即它的值非常接近于 0。

为了方便，我们引入一个记号，在不致混淆的情况下，对于任意的博弈 x ， $x* = x + *$ 。

我们另外定义 $\uparrow = \{0|*\}$ 以及 $\downarrow = \{*\|0\}$ ，显然 $\uparrow = -\downarrow$ 。容易验证， $\uparrow > 0, \downarrow < 0$ ，但对于任意的正数 x ，有 $x \gg \uparrow, -x \ll \downarrow$ ，也即它们的值非常接近于 0。同样可以验证， $\uparrow * \parallel 0, \downarrow * \parallel 0$ ，这意味着 $* \parallel \uparrow, * \parallel \downarrow$ 。

我们自然会尝试研究一类最“简单”的博弈，也即 $|L| = |R| = 1, L, R \in \{0, *, \uparrow, \downarrow\}$ 。上面我们已经列举了一些，另外我们还容易验证 $\{\downarrow|\uparrow\}, \{*\|\uparrow\}, \{\downarrow|*\}$ 均是后手必胜的博弈，于是它们的值都为 0。剩余的一些博弈更加复杂，我们留待下一小节讨论。

3.4 博弈的化简

我们前面说过，有很多形式不同的博弈具有相等的值。一个自然的想法是，给定一个复杂的博弈，我们希望能找到与它值相等的最简单博弈，这样在绝大部分情况下，我们都可以用这个值相等的简单博弈来替换它。

定理 3.3.

1. (优越) 若 $G = \{A, B, \dots | C, D, \dots\}$ ，则若 $B \leq A$ (称为 B 被 A 优越)，则 $G = \{A, \dots | C, D, \dots\}$ ，类似地，若 $D \geq C$ ，则 $G = \{A, B, \dots | C, \dots\}$ 。也即，我们可以删去被优越的行动而不改变值。
2. (逆转) 若 $G = \{A, B, \dots | C, D, \dots\}$ ，其中存在某个 $A^R \leq G$ (称 A 为可被逆转行动)， $A^{RL} = \{\alpha, \beta, \dots\}$ ，则 $G = \{\alpha, \beta, B, \dots | C, D, \dots\}$ ，类似地，若存在某个 $C^L \geq G$ ， $C^{LR} = \{\alpha, \beta, \dots\}$ ，则 $G = \{A, B, \dots | \alpha, \beta, D, \dots\}$ 。也即，我们可以删去被逆转的行动，并用走两步后的状态替换而不改变值。
3. (礼品马原理) 若 $G = \{A, B, \dots | C, D, \dots\}$ ，则若 $H \ll G$ (称 H 为“礼品马”)，则 $G = \{H, A, B, \dots | C, D, \dots\}$ ，类似地，若 $H \gg G$ ，则 $G = \{A, B, \dots | H, C, D, \dots\}$ 。也即，我们可以添上或删去“礼品马”作为行动而不改变值。

证明.

1. 考虑 $\{A, B, \dots | C, D, \dots\} - \{A, \dots | C, D, \dots\} = \{A, B, \dots | C, D, \dots\} + \{-C, -D, \dots | -A, \dots\}$ 。对于后手来说，先手大部分的行动都可以用对应的逆行动来抵消进而获胜。值得注意的是左方先手，且在 G 中选择 B ，但由于 $B \leq A$ ，此时右方后手选择 $-A$ 即可获胜。于是上述差博弈 = 0，也即两个博弈值相等。

2. 考虑 $\{A, B, \dots | C, D, \dots\} - \{\alpha, \beta, B, \dots | C, D, \dots\} = \{A, B, \dots | C, D, \dots\} + \{-C, -D, \dots | -\alpha, -\beta, -B, \dots\}$ 。对于后手来说，先手大部分的行动都可以用对应的逆行动来抵消进而获胜。值得注意的是左方先手，且在 G 中选择 A ，此时右方选择走到对应的 $A^R \leq G$ ，那么左方继续在 A^R 中采取行动则右方可直接抵消进而获胜，于是左方只能在另一博弈选择 $-C, -D$ 等，不妨设为选择 $-C$ ，由于 $G - C < 0$ ，且 $A^R \leq G$ ，则 $A^R - C < 0$ ，于是此时右方先手必胜；右方先手，且在第二个博弈中选择 α, β 等，不妨设为选择 α ，由于 $A^R - \alpha > 0$ ，且 $A^R \leq G$ ，则 $G - \alpha > 0$ ，于是此时左方先手必胜。于是上述差博弈 $= 0$ ，也即两个博弈值相等。
3. 考虑 $\{A, B, \dots | C, D, \dots\} - \{H, A, B, \dots | C, D, \dots\} = \{A, B, \dots | C, D, \dots\} + \{-C, -D, \dots | -H, -A, -B, \dots\}$ 。对于后手来说，先手大部分的行动都可以用对应的逆行动来抵消进而获胜。值得注意的是右方先手，在第二个博弈中选择 $-H$ ，但由于 $H < G$ ，显然此时左方先手必胜。于是上述差博弈 $= 0$ ，也即两个博弈值相等。

□

通过不断删去被优超的行动，以及删去被逆转的行动并用走两步后的状态替换，直到不能继续进行这个操作为止，我们可以完成对一个博弈的化简，称得到的新博弈为原博弈的最简形，显然最简形与原博弈值相等。

定理 3.4. 若两个博弈 G 和 H 均是自身最简形，且 $G = H$ ，则 G 和 H 形式相等。也即，博弈的最简形是唯一的。

证明. $G = H$ 意味着 $G + (-H) = 0$ 。考虑在博弈 $G + (-H)$ 中，左方先手在 G 中选择某个 G^L 。此时右方后手必胜，他的行动要么是由 G^L 走到某个 G^{LR} ，要么由 $-H$ 走到某个 $-H^L$ 。前者意味着 $G^{LR} \leq H = G$ ，也即 G^L 可被逆转，显然矛盾，于是一定有后者。而后者意味着 $G^L \leq H^L$ ，由对称性，又有某个 G^L 使 $H^L \leq G^L$ ，那么 $G^L \leq G^L$ ，由于没有被优超的选择，一定有 $G^L = H^L = G^L$ ，于是 H^L 是 H 中与 G^L 对应选择。也即 G^L 和 G^R 中每个行动都在 H^L 和 H^R 中有相应行动，由对称性知 H^L 和 H^R 中每个行动都在 G^L 和 G^R 中有相应行动，于是 G 和 H 形式相等。 □

对于博弈 $\ast = \{0|0\}$ ，用礼品马原理，在两边分别添上 $\uparrow \ast$ 与 $\downarrow \ast$ ，我们可以得到 $\ast = \{0, \uparrow 0\} = \{0|0, \downarrow\} = \{0, \uparrow 0, \downarrow\}$ 。由于 0 分别被 \uparrow 与 \downarrow 优超，我们有 $\{\uparrow \downarrow\} = \{\uparrow 0\} = \{0| \downarrow\} = \ast$ 。

对于博弈 $\{0| \uparrow\}$ ，我们可以通过差博弈验证它等于 $\uparrow \ast = 2 \uparrow \ast$ 。那么用礼品马原理，在左边添上 $\uparrow 2 \uparrow \ast$ ，由于 0 被 \uparrow 优超，我们有 $\{\uparrow \uparrow\} = \{0| \uparrow\} = \uparrow \ast$ 。

再考虑博弈 $\uparrow \ast = \uparrow \ast = \{\ast, \uparrow 0, \uparrow\}$ 。右边的 \uparrow 被 0 优超，可以删去，而左边的 $(\uparrow)^R = \ast$ ，其中 $\ast < \uparrow \ast$ ，于是可以被逆转，替换为 $(\ast)^L = 0$ ，那么就有 $\uparrow \ast = \{0, \ast|0\}$ 。

类似地，我们可以得到 $\{\downarrow \downarrow\} = \{\downarrow 0\} = \downarrow \ast$ ， $\downarrow \ast = \{0|0, \ast\}$ 。

$L \backslash R$	0	*	↑	↓
0	$* = \{0 0\}$	$\uparrow = \{0 *\}$	$\uparrow * = \{0 \uparrow\}$	$* = \{0 0\}$
*	$\downarrow = \{*\ 0\}$	$0 = \{\}$	$0 = \{\}$	$\{*\ \downarrow\}$
↑	$* = \{0 0\}$	$\{\uparrow *\}$	$\uparrow * = \{0 \uparrow\}$	$* = \{0 0\}$
↓	$\downarrow * = \{\downarrow 0\}$	$0 = \{\}$	$0 = \{\}$	$\downarrow * = \{\downarrow 0\}$

表 1: $|L| = |R| = 1, L, R \in \{0, *, \uparrow, \downarrow\}$ 的博弈的最简形

4 公平博弈与 Nimber

4.1 公平博弈定义

比起一般化的不平等博弈理论，公平博弈的理论更为大家熟知。

定义 4.1. 一个博弈 G 是公平的，若在 G 与 G 的任意后继状态中，双方的行动集合（ L 集合与 R 集合）均完全相同。

显然，任意公平博弈中，双方的胜负情况只取决于先后手。也即，任意公平博弈 G 只有可能是后手必胜或先手必胜，即 $G = 0$ 或 $G \parallel 0$ 。

定理 4.1. 任意公平博弈 G 有 $G + G = 0$ ，即 $G = -G$ 。

证明. 在 $G + G$ 中，后手可以采用模仿策略，若先手在某一侧行动，后手只需在另一侧作相同行动即可获胜。 □

4.2 公平博弈实例

我们之前介绍的 Hackenbush 中，若只有绿边，显然双方在任意后继状态的行动集合均完全相同，于是是公平博弈。特别地，一条绿边对应的 Hackenbush，也即 $* = \{0|0\}$ 是公平博弈，因为双方后继状态都只有 0。

另一类知名的公平博弈是 Nim 游戏：给定一堆或多堆石子，每次行动时玩家需要恰好从某一堆中取走任意非零个石子，不能取（也即每堆都为空）的玩家判负。

4.3 Nimber

我们来分析一下单一堆的 Nim 游戏：

若这堆没有石子，双方均不能行动，此时博弈值为 0。

若恰有 1 个石子，则双方行动后都会变为没有石子的 0 状态，于是此时博弈值 $* = \{0|0\}$ 。

对于有 $n > 1$ 个石子的状态，双方行动后可以变为 $0 \sim n - 1$ 个石子，那么如何描述它们呢？

定义 4.2. 对于 $n > 1$, 定义 $*_n = \{0, *, *_2, \dots, *_{n-1} | 0, *, *_2, \dots, *_{n-1}\}$ 。也即, $*_n$ 为恰有 n 个石子的单堆 Nim 游戏状态的博弈值。

显然 $*_n$ 是一个先手必胜态 (先手可以直接行动到 0 状态获胜), 于是有 $*_n \parallel 0$, 容易验证 $*_n = -*_n$, 且对于任意的正数 x , 有 $x \gg *_n, -x \ll *_n$, 也即它们的值非常接近于 0。

我们称 $0, *, *_2, \dots, *_n, \dots$ 为 Nimber。

我们发现, 仅用 Nimber 就可以描述任意公平博弈的值。

定理 4.2. 任意 (有限) 公平博弈的值一定是某个 Nimber $*_n$ 。

证明. 考虑归纳法。不妨设公平博弈 G 能转移到的任意公平博弈的值均已证明是某个 Nimber, 则有 $G = \{GS | GS\} = \{*_n, *_n, *_n, \dots | *_n, *_n, *_n, \dots\}$ 。令 $mex(S)$ 为有限非负整数集合 S 中最小没有出现的非负整数, 由于 G 是有限博弈, 能转移到的状态数目有限, 故 $n = mex(n_1, n_2, n_3, \dots)$ 一定存在, 记 $n = SG(G)$ 成为公平博弈 G 的 SG 值, 我们下面证明 $G = *_n$ 。

注意到按照 mex 的定义, $0 \sim *_{n-1}$ 均会出现在 GS 中, 且 $*_n$ 不会出现在 GS 中, 但可能有某些 $*_m (m > n)$ 出现在 GS 中。我们直接验证 $G + *_n = 0$, 也即 $G = *_n$ 。在公平博弈 $G + *_n = \{0, \dots, *_{n-1}, *_m, \dots | *0, \dots, *_{n-1}, *_m, \dots\} + \{0, \dots, *_{n-1} | 0, \dots, *_{n-1}\}$ 中, 先手大部分行动都可以被后手用对应相同行动抵消而获胜, 唯一值得讨论的是从 G 走到 $*_m$, 但由于 $*_m$ 后继状态包含 $*_n$, 后手可以走到 $*_n$ 而获胜。□

上述证明中, 我们不仅证明了定理 4.2, 事实上还给出了计算任意公平博弈 G 的值的算法: 只需计算出 $SG(G)$, 则 $G = *_n$ 。而计算 $SG(G)$ 时, 只需按定义递推, 计算每个状态后继状态 SG 值的 mex 即可。

4.4 公平博弈的和

对于单个的公平博弈, 计算其值通常是容易的。我们更关心的是若干个公平博弈的和。显然有限个公平博弈的和仍是公平博弈, 按定理 4.2, 若两个有限公平博弈的值分别是 Nimber, 则它们的和的值也是一个 Nimber, 那么如何计算这个值呢? 广为人知的 Sprague-Grundy 定理给出了计算方法:

定理 4.3. (Sprague-Grundy 定理) 值为 $*_n$ 的公平博弈 G 与值为 $*_m$ 的公平博弈 H 的和 $G+H$ 值为 $*_{n \oplus m}$ 。这里 \oplus 是二进制按位异或运算。也即, 两个 Nimber $*_n$ 与 $*_m$ 的和 $*_n + *_m = *_{n \oplus m}$ 。

限于篇幅, 上述定理证明略去。

4.5 综合运算

引入了 Nimber 后, 我们可以研究它们与数字, \uparrow, \downarrow 间的和博弈。由于 Nimber 的和还是 Nimber, 且 $\uparrow = -\downarrow$, 我们只需要研究形如 $a + *_b + c \uparrow (a \in \mathbb{R}, b \in \mathbb{N}, c \in \mathbb{Z})$ 这样的博弈。

对于 $x > 0$, 我们已知 $x \gg *_b, x \gg \uparrow, -x \ll *_b, -x \ll \downarrow$ 。于是若 $a > 0$, 显然有 $a + *_b + c \uparrow > 0$; 若 $a < 0$, 显然有 $a + *_b + c \uparrow < 0$ 。若 $b = 0$ 或 $c = 0$ 时也容易讨论。我们下面只讨论 $a = 0, b, c \neq 0$ 的非平凡情况。

我们已知 $\uparrow * \parallel 0, \downarrow * \parallel 0$, 且 $\{\uparrow \mid \uparrow\} = \uparrow * > 0, \{\downarrow \mid \downarrow\} = \downarrow * < 0$ 。那么一般地, 当 $c > 1$ 时 $* + c \uparrow > 0$; 当 $c < -1$ 时 $* + c \uparrow < 0$; 当 $-1 \leq c \leq 1$ 时 $* + c \uparrow \parallel 0$ 。

而在博弈 $\uparrow + *_n (n \geq 2)$ 中, 左方先手可以将 $*_n$ 变为 0 而获胜, 右方先手时, 若在 $*_n$ 中行动后左方同样行动可获胜, 而在 \uparrow 中行动后变为 $* + *_n = *_n \oplus 1 \parallel 0$, 于是左方仍然获胜。这样, 就有 $\uparrow + *_n (n \geq 2) > 0$, 同理有 $\downarrow + *_n (n \geq 2) < 0$ 。进一步地, 当 $c > 0$ 时 $*_n + c \uparrow > 0 (n \geq 2)$; 当 $c < 0$ 时 $*_n + c \uparrow < 0 (n \geq 2)$ 。

例题 1. 福若格斯²

有一种“跳青蛙”博弈: 在一个 5 格棋盘上, 初始时左边两格各有一只向右的青蛙, 右边两格各有一只向左的青蛙, 中间空出一格, 左方只能操作向右的青蛙, 右方只能操作向左的青蛙。每次行动为让己方的某只青蛙向对应方向行动: 若它前一格是唯一的空格, 可以直接走一格; 若它前一格是一只不同朝向的青蛙, 且再前一格是唯一空格, 可以跳过那只青蛙到达空格。显然, 任意时刻某方要么有唯一的行动方式, 要么不能行动。

有多次询问。每次给定一个大小为 m 的博弈可重集合, 其中每个博弈都是由初始状态经过若干次行动后可得到的状态 (但不保证是由双方交替行动所得), 共有 23 种不同状态。可重集里的每个博弈都可出现也可不出现, 你需要回答所有可能的情况的各自和博弈中, 左方必胜, 右方必胜, 后手必胜, 先手必胜的各有多少种, 答案对 998 244 353 取模。

询问组数 $T \leq 100$, 每组询问 $m \leq 10^6$, 且每种不同状态出现次数大致相同。

我们用一个长度为 5 的字符串表示一个博弈, 从左到右每个字符表示该格状态, L 表示有一只向右青蛙, R 表示有一只向左青蛙, $_$ 表示一个空格。由于博弈状态只有 23 种, 且每种状态下每方最多只有一种行动, 通过递推我们不难得到所有状态的博弈值:

从图中可看出, 所有状态的博弈值只可能是 $0, \pm 1, \pm \frac{1}{2}, *, \uparrow, \downarrow$ 。于是我们可以预见到若干个博弈的和一定是 $a + *_b + c \uparrow$ 形式。特别地, 这里 a 是 $\frac{1}{2}$ 的整数倍, 且 b 只能是 0 或 1。

由于不同类型博弈之间无关, 只需分别对值是数的博弈计算出 $a > 0, a = 0, a < 0$ 的方案数, 对值是 $*$ 的博弈计算出 $b = 0, b = 1$ 的方案数, 对值是 \uparrow 与 \downarrow 的博弈计算出 $c > 1, c < -1, -1 \leq c \leq 1$ 的方案数, 即可简单统计出答案。这些计算大部分是类似的, 我们只讨论最复杂的值是数的博弈的计数。我们可以将所有数 $\times 2$ 后变为整数, 方便计算。显然值为 0 的博弈不需要关注, 而对于其它绝对值为 x 的博弈, 是否出现会让和的博弈值改变 x 。那么容易将问题转化为给定 p 个 0/1 变量与 q 个 0/2 变量, 询问和 $>, <, =$ 某个给定常数 r 的方案数。这只需要枚举 q 个 0/2 变量中选了多少个 2, 对 p 个 0/1 变量中选择 1 的个数的方案数做前缀和即可, 每部分的系数都是一个组合数。

²来源: 清华集训 2017

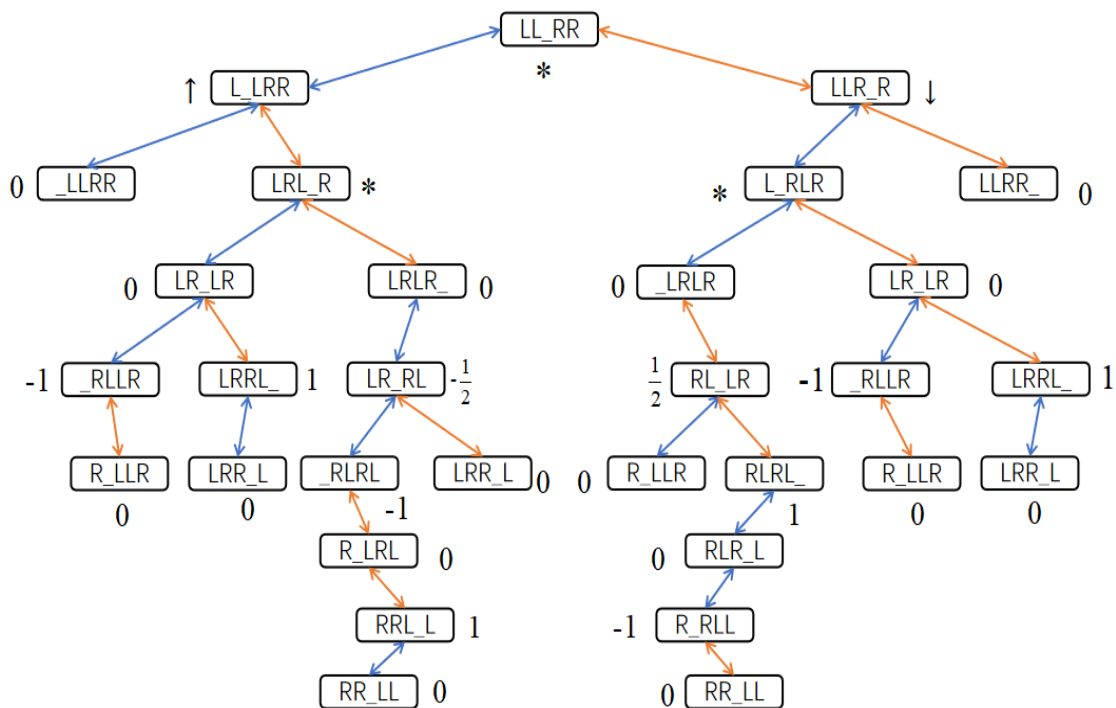


图 3: 例题 1 所有状态及对应博弈值

若预处理阶乘和阶乘逆，可 $O(1)$ 计算出组合数。因为 $p + q \leq m$ ，故总时间复杂度为 $O(Tm)$ 。但由于不同状态出现次数大致相同，因此 $p + q$ 大约只有 $\frac{8}{23}m$ ，实际运行效率很快。

5 热博弈

5.1 平移原理

定理 5.1. (平移原理) 对于一个值不是数字的博弈 $G = \{G^L|G^R\}$ 与值是数字的博弈 x ，我们有 $G + x = \{G^L + x|G^R + x\}$ 。也即，给定若干个博弈，若其中还有值不是数字的，则每位玩家的最佳走法都是在这些博弈中行动。

证明我们留待后文给出。

例题 2. もう、諦めない³

给定 n 个矩形，第 i 个大小为 $w_i \times h_i$ 。现在有一个博弈，每次玩家可以选择一个矩形，以及一个可以切的方向，均匀切成 $n > 1$ 份，要求切完后长宽仍是整数。对于第 i 个矩形，

³来源: Atcoder いろはちゃんコンテスト, Day4, Problem J

还会有参数 $a_i, b_i \in \{0, 1\}$ 。若 $a_i = 1$ ，则第 i 个矩形与它切出来的所有小矩形双方都可以在 w 这一维横着切，否则只有左方能横着切。若 $b_i = 1$ ，则第 i 个矩形与它切出来的所有小矩形双方都可以在 h 这一维竖着切，否则只有右方能竖着切。

现在给定 q 个询问，第 i 个询问为若仅留下第 $l \sim r$ 个矩形，左方先手能否获胜。

$n, q \leq 10^5, 1 \leq w, h \leq 10^5$ 。

首先显然需要算出每个矩形单独博弈时的博弈值。令 $\lambda(n)$ 为 n 的可重质因子个数， $\lambda^*(n)$ 为 n 的可重奇质因子个数，下面对 a, b 分类讨论。

若 $a = b = 1$ ，则这是一个公平博弈。注意到每次切割后的所有矩形都相同，因此出现奇数个时相当于只有 1 个，偶数个时直接变为 0。通过对 w 和 h 小时情况观察，我们可以猜测并证明如下结论：

引理 5.1. 当 $a = b = 1$ 时，一个 $w \times h$ 的矩形对应博弈值为：

$$\begin{cases} *_{\lambda^*(w) \oplus \lambda^*(h)}, & \text{if } w \times h \text{ is odd} \\ *_{(\lambda^*(w) \oplus \lambda^*(h)) + 1}, & \text{if } w \times h \text{ is even} \end{cases}$$

证明考虑归纳，只需注意到双方每次可以除去任意个数的奇质因子，且可能可以通过除去一个偶质因子得到 0 状态。这里略去具体证明。

若 $a = b = 0$ ，通过对 $\lambda(w)$ 和 $\lambda(h)$ 小时情况观察，我们可以猜测并证明如下结论：

引理 5.2. 令 w 的可重质因子降序排列为 $p_1 \geq p_2 \geq \dots \geq p_{\lambda(w)}$ ， h 的可重质因子降序排列为 $q_1 \geq q_2 \geq \dots \geq q_{\lambda(h)}$ 。则当 $a = b = 0$ 时，一个 $w \times h$ 的矩形对应博弈值为：

$$\begin{cases} \sum_{i=1}^{\lambda(w)-\lambda(h)} \prod_{j=1}^{i-1} p_j, & \text{if } \lambda(w) > \lambda(h) \\ - \sum_{i=1}^{\lambda(h)-\lambda(w)} \prod_{j=1}^{i-1} q_j, & \text{if } \lambda(w) < \lambda(h) \\ 0, & \text{if } \lambda(w) = \lambda(h) \end{cases}$$

证明考虑归纳，只需注意到双方每次选的 n 若质因子数目 > 1 一定不优，同时一定选择最大质因子。这里略去具体证明。

若 $a = 0, b = 1$ ，我们同样可以观察和猜测下述结论：

引理 5.3. 令 w 的可重质因子降序排列为 $p_1 \geq p_2 \geq \dots \geq p_{\lambda(w)}$ 。则当 $a = 0, b = 1$ 时，一个 $w \times h$ 的矩形对应博弈值为 $h \cdot (\sum_{i=1}^{\lambda(w)} \prod_{j=1}^{i-1} p_j) + *_{\lambda^*(h) + [2]h}$ 。

证明. 考虑归纳。

当 $h = 1$ 时由引理 5.2 易得。

当 $h > 1$ 时，令 $f(w) = \sum_{i=1}^{\lambda(w)} \prod_{j=1}^{i-1} p_j$ ，则 h 个 $w \times 1$ 的矩形博弈值即为 $h \cdot f(w)$ 。若先手在 h 这一维竖着切，根据归纳假设，行动后得到的博弈值一定形如 $hf(w) + *_k$ 。具体来说，若 n 为偶数，则 $k = 0$ ，否则 $k = \lambda^*(h) - \lambda^*(n)$ 。那么显然有右方先手行动后的博弈值集合 $\{hf(w), hf(w) +$

$\ast_1, \dots, hf(w) + \ast_{\lambda^*(h)+[2h]-1}$ }, 这个集合中的博弈值也是左方能达到的。而若左方在 w 这一维竖着切, 显然会发现得到的博弈值数字部分 $< hf(w)$, 会被优越, 可以不必考虑。那么最后的博弈值就是 $\{hf(w), hf(w) + \ast_1, \dots, hf(w) + \ast_{\lambda^*(h)+[2h]-1} | hf(w), hf(w) + \ast_1, \dots, hf(w) + \ast_{\lambda^*(h)+[2h]-1}\}$, 根据平移原理, 这就是 $hf(w) + \{0, \ast_1, \dots, \ast_{\lambda^*(h)+[2h]-1} | 0, \ast_1, \dots, \ast_{\lambda^*(h)+[2h]-1}\} = hf(w) + \ast_{\lambda^*(h)+[2h]}$ 。□

$a = 1, b = 0$ 的情况与 $a = 0, b = 1$ 类似。

这样, 我们计算出了每个矩形的博弈值, 发现都是 $a + \ast_b$ 的形式。那么一个区间的和博弈值仍然是一个 $a + \ast_b$ 的形式, 只需要预处理前缀和即可 $O(1)$ 查询。知道博弈值后, 通过它与 0 的大小关系就容易知道胜负了。

时间复杂度为 $O(n + \max\{w, h\} + q)$ 。

5.2 热博弈与转换

根据平移原理, 给定若干个不是数的博弈 G_1, G_2, \dots 与若干个是数的博弈 x_1, x_2, \dots , 则在和博弈 $G_1 + G_2 + \dots + x_1 + x_2 + \dots$ 中, 两位玩家都尽可能在 G_1, G_2, \dots 中行动而避开 x_1, x_2, \dots , 直到 G_1, G_2, \dots 等都变成确定的数。此时, 我们的和博弈变成了若干个数字之和, 显然只需要直接将数字加起来得到一个数字和, 再根据数字和与 0 的大小关系以及当前玩家就可以判定胜负了。

这给了我们启发: 每个不是数的博弈具有一定的热度, 因此是热博弈, 与之相对, 是数的博弈则是冷的。那么在一堆冷的和热的博弈的和中, 直观来看, 玩家通常会选择在较热的博弈中行动。

热博弈通常是很复杂的, 我们先研究其中最简单的一类:

定义 5.1. 称一个博弈 $\{x|y\}$ 为转换, 若 x, y 都是数且 $x \geq y$ 。

特别地, 当 $x = y$ 时, 显然有 $\{x|y\} = x\ast$; 当 $x = -y$ 时, 我们记 $\{x|y\} = \pm x$ 。

根据平移原理, 对于转换 $\{x|y\}$ 和数 z , 我们有 $\{x|y\} + z = \{x + z|y + z\}$ 。类似地, 我们有 $\{x|y\} + \ast = \{x\ast|y\ast\}(x \geq y)$ 和 $\{x\ast|y\ast\} + \ast = \{x|y\ast\}(x > y)$ 。

对于一个转换 $\{x|y\}$, 令 $u = \frac{1}{2}(x + y), v = \frac{1}{2}(x - y)$, 则有 $\{x|y\} = u \pm v$ 。我们认为一个转换的热度就是 v , 显然, v 越大转换就越热。那么对于若干个转换的和博弈 $\{x_1|y_1\} + \{x_2|y_2\} + \dots$, 由于平移原理, 双方需要恰好在每个转换中行动一次, 而为了获得优势, 显然双方都会选择当前剩余转换中热度最大的行动。这样, 我们就有了一个简单的算法判定转换的和博弈的胜负: 直接按转换热度降序排列, 则双方会依次选取最前面的。

5.3 停止值与模糊区间

为了研究更一般的热博弈, 我们引入停止值的概念。

定义 5.2. 对数 x , 定义 $L(x) = \{\{y|y < x\}|\{y|y \geq x\}\}, R(x) = \{\{y|y \leq x\}|\{y|y > x\}\}$ 。直观来看, $L(x)$ 比 x 小, 但比任何比 x 小的数大, $R(x)$ 比 x 大, 但比任何比 x 大的数小。显然, 对任意数 $x < y$, 有 $L(x) < x < R(x) < L(y) < y < R(y)$ 。

定义 5.3. 我们递归定义博弈 G 的左方停止值 $L(G)$ 和右方停止值 $R(G)$ 如下:

对于博弈 G , 令 $L = \max_{G^L} R(G^L)$ (不存在 G^L 设为 $-\infty$), $R = \min_{G^R} L(G^R)$ (不存在 G^R 设为 $+\infty$)。

若 $G = x$, 其中 x 是数, 则令 $L(G) = L(x), R(G) = R(x)$ 。

否则, 令 $L(G) = L, R(G) = R$ 。

显然, $L(-G) = -R(G), R(-G) = -L(G)$ 。

例如, $* = \{0|0\}$ 有 $L(*) = R(0), R(*) = L(0)$, $\uparrow = \{0|*\}$ 有 $L(\uparrow) = R(\uparrow) = R(0)$, $\downarrow = \{*\|0\}$ 有 $L(\downarrow) = R(\downarrow) = L(0)$ 。

直观来看, 对于通常博弈 G , $L(G)$ 和 $R(G)$ 就是博弈 G 中左方先手和右方先手的情况下, 双方轮流行动直到博弈值变成数时的值, 而最终数的符号 L 和 R 则表明了此时下一次行动的对象。

定理 5.2.

1. $L < R$ 当且仅当 G 的值是数。
2. 若 $L < R$, 则 G 的值即为满足 $L < x < R$ 的数中最简单的那个。
3. 若 $L \geq R$, 则对数 x , $x > L$ 当且仅当 $G < x$, $x < R$ 当且仅当 $G > x$, $R < x < L$ 当且仅当 $x \parallel G$ 。

证明. 考虑归纳。

当 G^L, G^R 中至少一者不存在时, G 的值显然是一个整数, 容易验证定理成立。

当 G^L, G^R 均存在时, 根据归纳假设, 不论 G^L, G^R 值是否是数, 都有 $\forall G^L < \| x$ 当且仅当 $x > R(G^L) = L$, 且 $\forall G^R \parallel > x$ 当且仅当 $x < L(G^R) = R$ 。

若 $L < R$, 则由于 x 是满足 $L < x < R$ 的数 x 中最简单的, 于是对某个 x 的形式 $x = \{x^L|x^R\}$ 有 $x^L < L < x < R < x^R$ 。那么在差博弈 $G - x$ 中, 左方先手走到某个 $G^L - x$ 或右方先手走到某个 $G^R - x$ 都显然必败, 且若左方先手走到 $G - x^R$, 右方走到 $G^R - x^R$, 由于 $x^R > R$ 于是 $G^R \leq x^R$, 右方必胜, 同理右方先手走到 $G - x^L$ 左方也必胜。这样不论谁先手都有 $G - x$ 后手必胜, 也即 $G = x$ 。

若 $L \geq R$, 则在差博弈 $G - x$ 中: 若 $x > L$, 右方先手走到 $G^R - x$ 可获胜, 且左方先手走到 $G^L - x$ 显然必败, 而走到 $G - x^R$ 右方可走到 $G^R - x^R$ 获胜, 于是右方必胜, 也即 $G < x$; 同理若 $x < R$ 左方必胜, 也即 $G > x$; 而若 $R < x < L$, 左方先手走到 $G^L - x$ 与右方先手走到 $G^R - x$ 都必胜, 也即 $G \parallel x$ 。那么不论 x 取值如何, 均没有 $G = x$, 也即 G 的值不是数。

由上述讨论知, $L < R$ 当且仅当 G 的值是数。 □

由定理 5.2 我们容易得到下述推论：

推论 5.1. 对于博弈 G, H 和数 x ，有：

1. 若 $G \geq H$ ，则 $L(G) \geq L(H), R(G) \geq R(H)$ 。特别地，若 $G = H$ ，则 $L(G) = L(H), R(G) = R(H)$ 。也即，博弈的左右停止值与形式无关，只与值有关。
2. $L(x + G) = x + L(G), R(x + G) = x + R(G)$ 。
3. $L(x - G) = x - L(G), R(x - G) = x - R(G)$ 。

现在我们可以给出平移原理的证明了：

证明. (平移原理) 考虑博弈 $G + x - \{G^L + x | G^R + x\}$ ，对于后手来说，先手大部分的行动都可以用对应的逆行动来抵消进而获胜。值得讨论的是左方先手从 x 走到 x^L 或右方先手从 x 走到 x^R 。对于前者来说，行动后得到的新博弈为 $(G + x^L) - \{G^L + x | G^R + x\}$ ，若行动后左方必胜，意味着新博弈 ≥ 0 ，即 $G + x^L \geq \{G^L + x | G^R + x\}$ ，于是 $L(G + x^L) \geq L(\{G^L + x | G^R + x\})$ ，但 $L(G + x^L) = L(G) + x^L < R(G^L) + x = R(G^L + x) = L(\{G^L + x | G^R + x\})$ ，矛盾。同理后者右方也无法获胜。

于是博弈 $G + x - \{G^L + x | G^R + x\}$ 后手必胜，即 $G + x = \{G^L + x | G^R + x\}$ 。 □

定理 5.2.2 可以认为是数的简单性法则在一般值为数的博弈的自然推广，使用起来非常方便。例如，对于博弈 $G = \{2 | -1\} | 0$ ，我们只需要注意到 $L(G) = L(-1), R(G) = L(0)$ ，立刻便可以知道 $G = -1$ 。

定理 5.2.3 事实上描述了与值非数博弈 G 模糊的所有数 x 的范围，我们记为 G 的模糊区间：

定义 5.4. 值非数博弈 G 的模糊区间是 $(L(G), R(G))$ ，一个数 $x \parallel G$ 当且仅当 $x \in (L(G), R(G))$ 。

5.4 平均值, 冷却与热图

一般的热博弈十分复杂，但利用下面介绍的平均值，我们可以得到它们的一个比较好的近似值。

定义 5.5. 我们递归定义一个博弈 G 冷却 $t \geq 0$ 度的值 G_t 如下：

$G_t = \{G^L_t - t | G^R_t + t\}$ ，但若存在某个最小的 t_0 使 G_{t_0} 无限接近于某个数 x (例如 x 或 x^*)，我们便定义 G 的平均值 $m(G) = x$ ，热度 $t(G) = t_0$ ，且定义所有 $t > t_0$ 均有 $G_t = x$ 。

显然，数 x 的平均值就是 x 本身，且热度为 0。

显然， $m(-G) = -m(G), t(-G) = t(G)$ 。

例如, $G = \{2| -1\}$ 有:

$$G_t = \begin{cases} \{2-t| -1+t\}, & 0 \leq t < \frac{3}{2} \\ \{2-\frac{3}{2}| -1+\frac{3}{2}\} = \frac{1}{2}*, & t = \frac{3}{2} \\ \frac{1}{2}, & t > \frac{3}{2} \end{cases}$$

于是它的平均值为 $\frac{1}{2}$, 热度为 $\frac{3}{2}$ 。

又例如, $*$ = $\{0|0\}$, \uparrow = $\{0|*\}$, \downarrow = $\{*\|0\}$ 的平均值和热度均为 0, 而 $G = \{\{2|1\}|0\}$ 的平均值和热度均为 $\frac{3}{4}$ 。

直接按定义计算平均值是麻烦的, 不过由定理 5.2 和平移原理, 我们只需要知道 $L(G_t)$ 和 $R(G_t)$, 并找出它们无限接近时最小的 t 即可, 而若我们知道了 $\max_{G^L} R(G^L_t)$ 和 $\min_{G^R} L(G^R_t)$ (可以归纳证明都是关于 t 的分段线性函数), 我们只需作一些平移。我们可以将 $L(G_t)$ 与 $R(G_t)$ 的函数图像同时画在一个图中, 从而得到博弈 G 的热图 (为了方便, 热图的坐标轴与通常意义相反)。

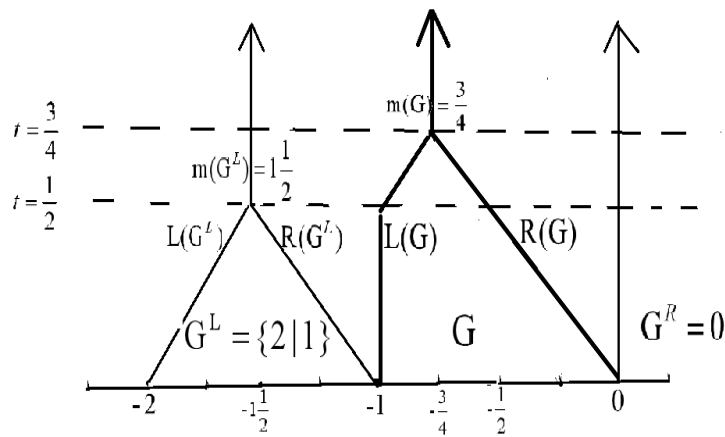


图 4: 博弈 $G = \{\{2|1\}|0\}$ 热图

容易看出, 博弈 G 的热图左侧部分斜率一定是 1 或 ∞ , 右侧部分斜率一定是 -1 或 ∞ , 最顶部一定是左右两侧交点处发起的一条竖直向上的射线, 射线的横坐标就是 $m(G)$, 而交点纵坐标就是 $t(G)$ 。

通过冷却定义, 对热图观察及归纳法, 我们容易得到下述结论:

引理 5.4. 对博弈 G 和数 x , 有:

1. $L(G_t)$ 与 $R(G_t)$ 的值即为热图上对应纵坐标处两条函数图像横坐标, 而它们的符号 L, R 满足下述法则: 在斜的部分靠外, 在直的部分靠内, 而在边界点符号与稍靠下部分相同。特别地, 取 $t = 0$, 由于斜率绝对值至少为 1, 于是 $L(G) < m(G) + t(G) + \epsilon, R(G) > m(G) - t(G) - \epsilon$, 其中 ϵ 是任意小的正数字。

2. 若 $G \geq x$, 则 $G_t \geq x$ 。
3. $(x + G)_t = x + G_t$ 。
4. $(x - G)_t = x - G_t$ 。
5. $(G_u)_v = G_{u+v}$ 。

作为推论, 我们可以得到下述重要定理:

定理 5.3. 对任意博弈 G, H , 均有 $(G + H)_t = G_t + H_t$ 。特别地, 取 $t = \infty$ 有 $m(G + H) = m(G) + m(H)$ 。

证明. 当 $G, H, G + H$ 中至少有一者是数时, 由引理 5.4.3 和 5.4.4 显然成立。

当 $G_t, H_t, (G + H)_t$ 均不是数时, 显然有:

$$\begin{aligned} G_t + H_t &= \{G_t^L + H_t - t, G_t + H_t^L - t | G_t^R + H_t + t, G_t + H_t^R + t\} \\ &= \{(G + H)_t^L - t | (G + H)_t^R + t\} \\ &= \{(G + H)_t^L | (G + H)_t^R\} \\ &= (G + H)_t \end{aligned}$$

其它情况可以用引理 5.4.5 转化为上述两种情况。 □

推论 5.2. 若 $G \geq H$, 则 $G_t \geq H_t$ 。特别地, 取 $G = H$ 则有 $G_t = H_t$, 于是 $m(G) = m(H), t(G) = t(H)$ 。也即, 博弈的平均值与热度与形式无关, 只与值有关。

证明. $G \geq H$ 有 $G - H \geq 0$, 于是 $G_t - H_t = (G - H)_t \geq 0$ 。 □

进一步地, 我们还有下述推论:

推论 5.3. 对博弈 G 和 $\forall k \in \mathbb{N}^*$, 有 $(kG)_t = kG_t$, 于是 $L(kG) < km(G) + t(G) + \epsilon, R(kG) > km(G) - t(G) - \epsilon$, 故 $km(G) - t(G) - \epsilon < kG < km(G) + t(G) + \epsilon$ 。

也即, 我们可以用 $km(G)$ 作为 kG 的近似值, 误差不会超过常数 $t(G) + \epsilon$, 这也解释了“平均值”的名字由来。

6 全部小博弈

6.1 全部小博弈与顺序和

我们先定义全部小博弈:

定义 6.1. 一个博弈是全部小的，若它的每个后继状态中，要么双方都不能行动，要么双方都能行动。公平博弈是一种特殊的全部小博弈。

显然，有限个全部小博弈的和还是全部小的。

例如，博弈 $0, *_n, n \uparrow, n \uparrow + *$ 都是全部小的。

定理 6.1. 对于全部小博弈 G 和任意数 $x > 0$ ，有 $-x < G < x$ 。

证明. 只需证明 $G+x > 0$ 和 $G-x < 0$ 。对于前者，在博弈 $G+x$ 中，左方只需不管 x 在 G 中行动，而右方在 x 中的行动只会增大 x ，同时由于 x 是全部小的，左方不能在 G 中行动仅当 $G=0$ ，此时显然有 $G+x=x > 0$ ，即左方必胜，于是前者成立。类似可知后者也成立。□

直观来看，全部小博弈的值的“绝对值”非常小。

我们再定义顺序和：

定义 6.2. 两个博弈 G 和 H 的顺序和 $G:H = \{G^L, G:H^L | G^R, G:H^R\}$ 。也即，对 G 的行动会令 H 消失，但对 H 的行动不会改变 G 。

显然， $-(G:H) = (-G):(-H)$ 。

例如，顺序和 $G:0 = G$ ，且 $0:H = \{0:H^L | 0:H^R\}$ ，于是可以归纳证明 $0:H = H$ 。

定理 6.2. 对于博弈 G, H, K ， $G:H$ 与 $G:K$ 大小关系同 H 与 K 大小关系相同。特别地，若 $H=K$ ，则 $G:H = G:K$ 。

证明. 考虑博弈 $G:H - G:K$ 。若左方在 $H-K$ 中某方先手时有必胜走法，则左方只需要按这个走法走，直到右方选择在某一侧 G 中行动。但左方此时显然只需在另一侧作相同行动，即可得到 0 状态而获胜。类似将左方换成右方也成立。□

值得注意的是，若 $G=H$ ， $G:K = H:K$ 未必成立。一个反例是 $\{-1|1\} = 0$ ，但 $\{-1:1\}:1 = \frac{1}{2} \neq 0:1$ 。

直观来看，在顺序和 $G:H$ 中， H 对局势影响远小于 G ：

定理 6.3. (Norton 引理) 若 G 与博弈 K 的任何一个后继状态值均不相同，则 $G:H$ 与 K 大小关系同 G 与 K 大小关系相同。

证明. 考虑博弈 $G:H - K$ 。若左方在 $G-K$ 中某方先手时有必胜走法，则左方只需要按这个走法走，直到右方选择在 H 中行动。但由于左方在右方行动前的 $G-K$ 中必胜，于是 $G-K \geq 0$ ，但 G 与 K 值不相同，于是 $G-K > 0$ ，则左方可以换成他先手必胜走法继续行动从而最终获胜。类似将左方换成右方也成立。□

6.2 全部小博弈实例

一个典型的无穷小博弈是仅由绿边组成的一棵倒着的有根树形态的 Hackenbush。由于一条绿边的博弈值为 $*$ ，于是一个这样的博弈 G 可以被写为 $* : H$ ，其中 H 是 G 去掉最底部绿边剩余部分。我们尝试求出 G 的博弈值，注意到 H 是若干棵有根树 Hackenbush H_1, H_2, \dots, H_k 的和博弈，因此若递归计算出 $H_1 = *_{n_1}, H_2 = *_{n_2}, \dots, H_k = *_{n_k}$ ，我们可以求出 H 的值 $*_n = *_{n_1 \oplus n_2 \oplus \dots \oplus n_k}$ ，由定理 6.2，我们有 $G = * : H = * : *_n = *_{n+1}$ 。

另一类全部小博弈是一种被称为“花”，接地部分是一条由绿边组成的链，链顶端只有蓝边或只有红边的 Hackenbush。若顶端只有蓝边，我们称为蓝花，若顶端只有红边，我们称为红花。

6.3 原子量

全部小博弈的和博弈值通常没有简单表示，但若我们只关心它的胜负关系，下面介绍的原子量理论可以提供帮助。

定义 6.3. 远星 \star 是一个 n 足够大的 $*_n$ 。这里的“足够大”的 n ，可以认为是比涉及的其他博弈后继状态中的任何 $*_k$ 的 k 都要严格大的任意整数。

定义 6.4. 我们递归定义一个全部小博弈 G 的原子量 G'' 如下：

$G'' = \{G^{L''} - 2|G^{R''} + 2\}$ ，除非如此定义得到的 G'' 是一个整数，且 $G > \star$ 或 $G < \star$ 。对于前者，我们令 G'' 为 $\ll G^{R''} + 2$ 的最大整数，对于后者，我们令 G'' 为 $\gg G^{L''} - 2$ 的最大整数。

例如， 0 的原子量显然是 0 ； $*$ 的原子量为 $\{0 - 2|0 + 2\} = 0$ ，类似地，一切 $*_n$ 的原子量均为 0 ； $\uparrow = \{0|*\}$ 的原子量直接计算是 $\{0 - 2|0 + 2\} = 0$ ，但由于 $\uparrow > \star$ ，于是真实原子量是 $\ll 2$ 的最大整数 1 ，类似地， $n \uparrow (n \in \mathbb{Z})$ 的原子量为 n ； $\uparrow * = \{0|\uparrow\}$ 的原子量直接计算是 $\{0 - 2|1 + 2\} = 0$ ，但由于 $\uparrow * > \star$ ，于是真实原子量是 $\ll 1 + 2$ 的最大整数 2 。

原子量未必是整数，甚至未必是数。例如，我们有 $\{\uparrow|\downarrow\}$ 的原子量是 $\{2 - 2|-2 + 2\} = *$ 。关于原子量，我们有下述定理：

定理 6.4. 对于全部小博弈 G, H ，有：

1. $(-G)'' = -(G'')$ 。
2. $(G + H)'' = G'' + H''$ 。
3. $G'' \geq 1$ 当且仅当 $G > \star$ 。
4. $G'' \leq -1$ 当且仅当 $G < \star$ 。

5. $-1 < G'' < 1$ 当且仅当 $G \star$ 。
6. 若 $G'' \geq 2$, 则 $G > 0$ 。
7. 若 $G'' \leq -2$, 则 $G < 0$ 。
8. 若 $G'' \gg 0$, 则 $G \gg 0$ 。
9. 若 $G'' \ll 0$, 则 $G \ll 0$ 。

限于篇幅, 上述定理证明略去。

我们最后给出一个原子量理论的应用实例。考虑 Hackenbush 中的花, 容易验证一朵蓝花的原子量为 1, 一朵红花的原子量为 -1 , 而一条由绿边组成的链的原子量则是 0。图 5(a) 中, 我们有总和原子量 = 2, 于是可以知道对应和博弈值 > 0 , 事实上不论哪方先手, 左方只需要在第一次行动中将红花的绿边砍去即容易获胜。图 5(b) 中, 我们有第一朵蓝花原子量 = 1, 而后面是一条较长的由绿边组成的链 (可被认为是 \star), 于是可以知道对应和博弈值 > 0 , 事实上若左方先手, 左方可以忽略蓝边而获胜, 若右方先手, 右方在忽略蓝边后的博弈中唯一获胜手段是在 \star 中行动, 此时左方只需砍去蓝边, 先后手便会交换从而获胜。

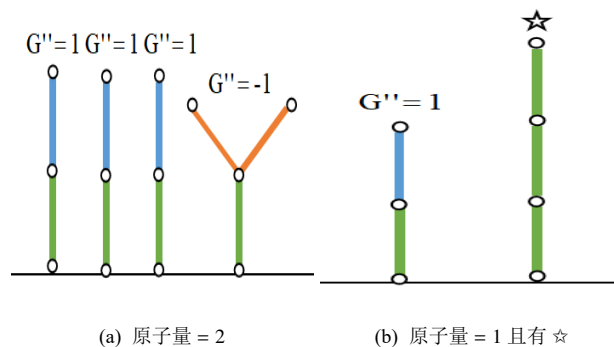


图 5: 花的和

7 总结

本文介绍了超现实数与不平等博弈的相关理论, 通过引入超现实数, 建立了较为严密的博弈数学模型, 在一定程度上给出了博弈问题的一些通用解法。

超现实数与不平等博弈的理论博大精深, 本文仅起到抛砖引玉的作用。例如, 本文仅研究了有限博弈, 没有涉及更一般的无限及带圈 (平局) 博弈的理论。希望本文能激发选手们对相关理论研究的热情, 也希望有更多利用超现实数与不平等博弈理论的有趣题目出现在 OI 中。

8 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢父母对我多年的培养与教导。

感谢国家集训队高闻远教练的指导。

感谢林盛华老师、陈旭龙老师对我的培养与教导。

感谢杜瑜皓学长、虞皓翔同学、代晨昕同学对本文的帮助。

感谢钟子谦学长、罗恺同学、张家瑞同学为本文验稿。

参考文献

- [1] John Conway. *On Numbers And Games*, 1976.
- [2] Elwyn R. Berlekamp, John Conway, Richard Guy. *Winning Ways For Your Mathematical Plays, Volume 1*, 1982.
- [3] 方展鹏. 浅谈如何解决不平等博弈问题. IOI2009 中国国家候选队论文集, 2009.
- [4] 贾志豪. 组合游戏略述——浅谈 SG 游戏的若干拓展及变形. IOI2009 中国国家候选队论文集, 2009.

浅谈线性代数与图论的关系

宁波市镇海中学 潘佳奇

摘要

本文将围绕着与图相关的矩阵介绍一部分与其相关的结论，以及在OI中常见的相关问题。

1 前言与约定

作者在平时的练习中，积累了一些图和矩阵之间关系的结论，并且得到了一些推广。

本文主要围绕这些矩阵的组合意义以及其特征系统的性质展开。因为这两者在OI中具有很大的实用意义：最优化问题以及计数问题。

1.1 内容简介

在第二节中主要介绍LGV引理的应用。LGV引理为有向无环图不交路径集合并计数提供了优美的方法，并且得到了特殊图最大流的应用。

在第三节中主要用LGV引理推导出Cauchy-Binet公式，并推导出矩阵树定理，同时得到了图的拉普拉斯矩阵与图的 k -生成森林之间的联系。

在第四节中主要通过运用矩阵树定理的证明过程得到了内向树计数以及特殊拟阵求交的方法。

在第五节中主要介绍邻接矩阵和拉普拉斯矩阵在图的积上的拓展，根据其特征值的变化可以得到许多有意思的结论。

1.2 符号与约定

对于一张图 $G = (V, E)$ ，每条边 $e \in E$ 有一个边权 ω_e ，其中所有的 ω_e 都属于同一个交换环（commutative ring¹）。为了方便描述，如无特殊说明，我们下面一致用实数域来讨论，至于交换环上的拓展，证明过程基本相同。如果一张图无权，则 $\omega_e = 1 (e \in E)$ 。我们通过函数 $V(G)$ 得到图的点集，函数 $E(G)$ 得到图的边集。

¹ https://en.wikipedia.org/wiki/Commutative_ring

我们约定矩阵 \mathbf{A} 的子式 $\mathbf{A}_{[S],[T]}$ 为选取 \mathbf{A} 所有 $a_{i,j}(i \in S, j \in T)$ 的元素得到的子矩阵。如果 $S = T$, 则称子式 $\mathbf{A}_{[S],[T]}$ 为 \mathbf{A} 的主子式 $\mathbf{A}_{[S]}$ 。

我们记 n 元置换的集合为 S_n 。 $N(\sigma)$ 是置换 σ 的逆序数。

对于两个序列 A, B , \sqcup 表示两个序列的拼接, 例如 $[1, 2] \sqcup [4] = [1, 2, 4]$ 。

对于一个矩阵 \mathbf{M} , 其行列式记为 $|\mathbf{M}|$ 。

2 LGV 引理

2.1 引理介绍

这一节中出现的路径都是指有向路径 (directed path²);

给定一张有向无环图 G , 对于图上任意一条路径 P , 定义 $\omega(P)$ 为路径上所有边的边权积。定义两顶点 u, v 之间的权值 $\omega(u, v)$, 为 u, v 之间所有路径 P 权值 $\omega(P)$ 的和, 即 $\omega(u, v) = \sum_{P:u \rightarrow v} \omega(P)$ 。

当边权都为 1 的时候, $\omega(u, v)$ 的意义就变成了 u 到 v 路径的个数。

给定起点元组 $S = (S_1, S_2, \dots, S_n)$ 和终点元组 $T = (T_1, T_2, \dots, T_n)$, 其中 S 和 T 的元素都属于顶点集合 V 。

定义 2.1.1 (n-path). 称一个路径 n 元组 $P = (P_1, P_2, \dots, P_n)$ 为 S, T 的 n -路径 (n -path), 当且仅当存在一个置换 $\pi, \forall 1 \leq i \leq n$, 路径 P_i 的起点是 S_i , 终点是 $T_{\pi(i)}$, 即 $P_i: S_i \rightarrow T_{\pi(i)}$ 。

显然对于一个 n -路径, 上述的置换 π 是唯一的。

类似地, 我们定义一个 n -路径的权值 $\omega(P) = \prod_{i=1}^n \omega(P_i)$ 。

定义 2.1.2 (permutation of n-path). 定义 n -路径到置换的映射 $\sigma(P)$ 为, 对于任意一个 n -路径 P , $\pi = \sigma(P)$ 为满足 $\forall i, P_i$ 的终点是 $T_{\pi(i)}$ 的置换。我们把 $\sigma(P)$ 称作 n -路径 P 的置换。

定义 2.1.3 (non-intersecting n-path). 对于一个 n -路径, 如果这个元组满足 $\forall 1 \leq i < j \leq n$, 路径 P_i 和 P_j 没有公共顶点, 那么我们称这个 n -路径为不交的 (non -intersecting)。

定义 2.1.4 (entangled n-path). 对于一个 n -路径, 如果它不是不交的, 那么我们称这个 n -路径为相交的 ($entangled$)。

引理 2.1.1 (Lindström-Gessel-Viennot).

对于图 G 和起终点集合 S 和 T , 记行列式

$$\mathbf{M} = \begin{vmatrix} \omega(S_1, T_1) & \omega(S_1, T_2) & \cdots & \omega(S_1, T_n) \\ \omega(S_2, T_1) & \omega(S_2, T_2) & \cdots & \omega(S_2, T_n) \\ \vdots & \vdots & \ddots & \vdots \\ \omega(S_n, T_1) & \omega(S_n, T_2) & \cdots & \omega(S_n, T_n) \end{vmatrix}$$

² [https://en.wikipedia.org/wiki/Path_\(graph_theory\)](https://en.wikipedia.org/wiki/Path_(graph_theory))

则

$$|\mathbf{M}| = \sum_{\substack{P \text{ is a} \\ \text{non-intersecting } n\text{-path}}} (-1)^{N(\sigma(P))} \omega(P)$$

证明 2.1.1 (Lindström–Gessel–Viennot Lemma). 对于 S 中存在相同元素或者 T 中存在相同元素, 证明是平凡的:

对于左式, \mathbf{M} 将有至少两行或两列相同, 此时行列式值为 0。对于右式, 对于每一个 n -路径, 都存在 $i \neq j$ 使得 P_i 和 P_j 的起点相同, 或终点相同, 此时不存在不交的 n -路径, 所以右式值为 0。

所以接下来只讨论 S 中元素两两不同, 且 T 中元素两两不同的情况。

$$\begin{aligned} |\mathbf{M}| &= \sum_{\pi \in S_n} (-1)^{N(\pi)} \prod_{i=1}^n \omega(S_i, T_{\pi(i)}) \\ &= \sum_{\pi \in S_n} (-1)^{N(\pi)} \prod_{i=1}^n \sum_{P: S_i \rightarrow T_{\pi(i)}} \omega(P) \\ &= \sum_{\pi \in S_n} (-1)^{N(\pi)} \sum_{P \text{ is an } n\text{-path with } \sigma(P)=\pi} \omega(P) \\ &= \left(\sum_{\substack{P \text{ is a} \\ \text{non-intersecting} \\ n\text{-path}}} (-1)^{N(\sigma(P))} \omega(P) \right) + \left(\sum_{\substack{P \text{ is an} \\ \text{entangled} \\ n\text{-path}}} (-1)^{N(\sigma(P))} \omega(P) \right) \end{aligned}$$

令相交的 n -路径 P 集合为 ζ , 则我们构造一个函数 $f(x) : \zeta \rightarrow \zeta$:

对于任意一个 n -路径 $P \in \zeta$, 设 $\pi = \sigma(P)$, 我们取出字典序最小的二元组 (i, j) , 满足 $i < j$ 并且 P_i, P_j 有公共顶点。

令 P_i 经过顶点序列为 $A = [\alpha_1, \alpha_2, \dots, \alpha_k]$, P_j 经过的顶点序列为 $B = [\beta_1, \beta_2, \dots, \beta_l]$ 。

记 $x = \min\{i | \exists j : \alpha_i = \beta_j\}$, $y = \min\{i | \beta_i = \alpha_x\}$ 。

则通过交换尾部得到 P'_i 的顶点序列 $A' = A[1 \dots x] \sqcup B[y + 1 \dots l]$, P'_j 的顶点序列 $B' = B[1 \dots y] \sqcup A[x + 1 \dots k]$ 。

则我们有 n -路径 $P' = (P_1, \dots, P_{i-1}, P'_i, P_{i+1}, \dots, P_{j-1}, P'_j, P_{j+1}, \dots, P_n)$, 易得 $\pi(P') = \pi' = (\pi(1), \dots, \pi(i-1), \pi(j), \pi(i+1), \dots, \pi(j-1), \pi(i), \pi(j+1), \dots, \pi(n))$, 且 P' 是相交的 n -路径, 即 $P' \in \zeta$ 。

我们令 $f(x)$ 为满足这样条件的函数: 对于所有这样的 n -路径 P , 都有 $f(P)$ 和按照上述过程得到的 P' 相等。

对于 A', B' , 相对于 A, B , 只修改了 $A_i (i > x)$ 和 $B_i (i > y)$ 的元素, 因此对于 P 和 P' 其得到的二元组 (i, j) 以及对应的 x, y 不会变化。所以可以得到 $f(f(P)) = P$, 即 f 是一个对

合函数 (Involution³)。

因为终点没有两个点相同, 则上述 π' 是 π 作用了一个奇置换得到的, 所以有

$$(-1)^{N(\pi')+N(\pi)} = -1$$

对于 $\omega(P'_i)\omega(P'_j)$, 容易得到

$$\begin{aligned} & \omega(P'_i)\omega(P'_j) \\ &= \left(\prod_{i=1}^x A_i \prod_{i=y+1}^l B_i \right) \left(\prod_{i=1}^y B_i \prod_{i=x+1}^k A_i \right) \\ &= \prod_{i=1}^k A_i \prod_{i=1}^l B_i \\ &= \omega(P_i)\omega(P_j) \end{aligned}$$

所以有

$$\begin{aligned} \omega(P') &= \omega(P'_i)\omega(P'_j) \prod_{\substack{1 \leq i \leq n \\ i \neq x, i \neq y}} \omega(P_i) \\ &= \omega(P_i)\omega(P_j) \prod_{\substack{1 \leq i \leq n \\ i \neq x, i \neq y}} \omega(P_i) \\ &= \prod_{1 \leq i \leq n} \omega(P_i) = \omega(P) \end{aligned}$$

因此有

$$\begin{aligned} & 2 \left(\sum_{\substack{P \text{ is an} \\ \text{entangled} \\ n\text{-path}}} (-1)^{N(\sigma(P))} \omega(P) \right) \\ &= \sum_{P \in \zeta} (-1)^{N(\sigma(P))} \omega(P) + (-1)^{N(\sigma(f(P)))} \omega(f(P)) \\ &= \sum_{P \in \zeta} ((-1)^{N(\sigma(P))} + (-1)^{N(\sigma(f(P)))}) \omega(P) \\ &= 0 \end{aligned}$$

由此得到

$$|\mathbf{M}| = \left(\sum_{\substack{P \text{ is a} \\ \text{non-intersecting} \\ n\text{-path}}} (-1)^{N(\sigma(P))} \omega(P) \right) + 0$$

□

³ [https://en.wikipedia.org/wiki/Involution_\(mathematics\)](https://en.wikipedia.org/wiki/Involution_(mathematics))

2.2 LGV 引理在网格图中的应用

在这里，我们特别地定义网格图是有向图。

定义 2.2.1 (square grid graph). 网格图 $S(n, m) = (V, E)$ 是一张有向图，其中 $V = \{(i, j) | 1 \leq i \leq n, 1 \leq j \leq m\}$, $E = \{((i, j), (i+1, j)) | 1 \leq i < n, 1 \leq j \leq m\} \cup \{((i, j), (i, j+1)) | 1 \leq i \leq n, 1 \leq j < m\}$ 。

容易得到，从 $u = (x, y)$ 走到 $v = (x', y')$ ($x \leq x', y \leq y'$) 的方案数 $\omega(u, v) = \binom{x'-x+y'-y}{x'-x}$ 。

例题 2.2.1 (Intersection is not allowed! ⁴) .

给出网格图 $S(N, N)$ ，给出 K 个起点 $(1, a_1), (1, a_2), \dots, (1, a_K)$ 和 K 个终点 $(N, b_1), (N, b_2), \dots, (N, b_K)$ 。

你要输出路径 K 元组 $P = (P_1, P_2, \dots, P_K)$ 的数量，满足 $\forall 1 \leq i \leq n$, P_i 的起点为 $(1, a_i)$ ，终点为 (N, b_i) ，且对于 $i \neq j$, P_i 和 P_j 没有公共点。

答案对 $10^9 + 7$ (一个质数) 取模。

保证 $1 \leq N \leq 10^5, 1 \leq K \leq 100, 1 \leq a_1 < a_2 < \dots < a_K \leq N, 1 \leq b_1 < b_2 < \dots < b_K \leq N$ 。

对照我们上面的定义，这道题实际上需要的是

$$|\mathbf{M}| = \sum_{\substack{P \text{ is a} \\ \text{non-intersecting } n\text{-path}}} \omega(P)$$

我们考虑在这道题的限制下，不相交的 n -路径的性质：

性质 2.2.1. 对于任意一个不相交的 n -路径 P ，都有 $\sigma(P) = (1, 2, \dots, K)$ 。

证明 2.2.1. 假设我们存在 $\pi = \sigma(P) \neq (1, 2, \dots, K)$ 的 n -路径 P ，找出任意的 i, j 满足 $1 \leq i < j \leq K$ ，使得 $\pi(i) > \pi(j)$ 。

此时有 $a_i < a_j$ 且 $b_{\pi(i)} > b_{\pi(j)}$ 。

假设 P_i 和 P_j 没有公共顶点。那么如果顶点 (x, y) 是 P_j 经过的顶点，那么 (x, y) 不可能是 P_i 经过的顶点。

如果 $(1, y)$ 不可能是 P_i 经过的顶点，那么 $(1, y+1)$ 不可能是 P_i 经过的顶点。

如果 $(x, y), (x-1, y+1)$ ($x > 1$) 不可能是 P_i 经过的顶点，那么 $(x, y+1)$ 不可能是 P_i 经过的顶点。

记 $f_x = \min\{k | \text{vertex } (x, k) \text{ is on } P_j\}$ ，那么 (x, f_x) 是 P_j 经过的顶点。

由网格图的性质，对于 $x > 1$ ，有 $f_x \geq f_{x-1}$ ，我们断言，对于 $\forall x, y$ ，满足 $1 \leq x \leq N, f_x \leq y$ ，有 (x, y) 不可能是 P_i 经过的顶点。

我们对下标的大小 x ($1 \leq x \leq N$) 进行归纳。

我们已经得到了 $x = 1$ 时， $(1, a_j)$ 不可能是 P_i 经过的顶点，因而得到了 $(1, j)$ ($j \geq f_1 = a_j$) 不可能是 P_i 经过的顶点，因此对于 $x = 1$ 成立。

⁴<http://acm.hdu.edu.cn/showproblem.php?pid=5852>

假设我们已经归纳完了 x , 对于 $x' = x + 1$, 我们已经得到 $(x', f_{x'})$ 不可能是 P_i 经过的顶点, 由归纳得, 对于 $j \geq f_{x'} \geq f_x$, 都有 (x, j) 不可能是 P_i 经过的顶点, 所以得到 (x', j) ($j \geq f_{x'}$) 不可能是 P_i 经过的顶点, 因此对于 $x' = x + 1$ 成立。

因为 $b_{\pi(i)} > b_{\pi(j)} \geq f_N$, $(N, b_{\pi(i)})$ 是 P_i 不可能经过的顶点, 与 P_i 经过了 $(N, b_{\pi(i)})$ 的条件矛盾, 因此 P_i 和 P_j 没有公共顶点的假设不成立。

所以 P_i 和 P_j 一定有公共顶点, 与任意两条路径不相交的条件矛盾, 因此 $\pi = \sigma(P) \neq (1, 2, \dots, K)$ 的假设不成立。

由此得到 $\forall \pi = \sigma(P)$, 都有 $\pi = (1, 2, \dots, K)$ 。

□

在这个性质下, $(-1)^{N(\sigma(P))}$ 的值只可能是 1, 所以我们只要通过构造行列式 \mathbf{M} , 就能通过预处理, 在 $O(n + K^3)$ 的时间复杂度下得到答案。

2.3 特殊有向无环图最大流

例题 2.3.1 (传播者⁵). 给定一个具有 n 层的分层图, 每一层有 k 个顶点, 第 i 层向第 $i + 1$ 层之间连了若干条有向边 ($1 \leq i \leq n - 1$)。

设 S, T 为两个点集, 满足 S 中所有顶点在第 l 层, T 中所有顶点在第 r 层, 且 $l < r$ 。

记 $\text{cut}(S, T)$ 为 S 和 T 的最小点割——即至少要去掉多少个顶点 (及其关联的边), 才能保证不存在一条从 s 到 t 的有向路径, 其中 $s \in S, t \in T$, 且 s, t 均未被去掉。

现在给定这张分层图, 有 q 次操作, 每次操作为改变一条边的存在状态或询问某两个点集的最小点割。

保证 $2 \leq n \leq 8192, 1 \leq k \leq 24, 1 \leq q \leq 8192, 1 \leq u, v \leq K, 1 \leq l < r \leq n$ 。

我们先考虑没有修改边, 且只有一次查询时的做法:

我们把 S 叫做源点集合, T 叫做汇点集合。由于有多个源点和汇点, 实际上由于条件是 S 和 T 里任意一对点连通, 所以我们可以建立超级源点 S' 和超级汇点 T' 。只要将原图的边集并上 $\{(S', s) | s \in S\}$ 和 $\{(t, T') | t \in T\}$, 问题就变成了:

要求 S' 和 T' 不能被去掉, 至少要去掉多少个顶点 (及其关联的边), 才能保证不存在一条从 S 到 T 的有向路径。

把最小点割转换到我们熟悉的最小割⁶。因为不能删边, 我们把边的边权看做 ∞ ; 因为点删了就不能连通, 我们把删点 u 看做删去 u_1 和 u_2 之间边权为 1 的边。

⁵ Author: 虞皓翔同学

⁶ 为了展示 LGV 引理在解决最大流问题的应用, 选择将问题转化为最大流问题, 而不是直接使用 Menger's theorem 将最小点割转换

于是，对于图中的有向边 (u, v) ，可以看做有向边 (u_2, v_1) 且边权为 ∞ ，对于图中的顶点 u 可以看做有向边 (u_1, u_2) 且边权为 1。由 Menger's theorem⁷，给定源汇点，一张图的最小割和最大流相等。所以只要求出源点 S'_2 到汇点 T'_1 的最大流，即可得到原题的答案。

再考虑存在修改边，多次查询时的做法：

因为点数有 $O(nk)$ ，边数有 $O(nk^2)$ ，跑 q 次最大流的复杂度是无法接受的。

考虑在这个模型中最大流的意义，就是从源点到汇点选择若干条有向路径 P_1, P_2, \dots, P_l ，使得对于 $1 \leq i < j \leq l$ ， P_i 和 P_j 没有公共的权值为 1 的边。

这和 n -路径的定义类似。实际上如果我们把所有权值为 1 的边看做点，在此基础上计算，这就将问题转化到了选择若干条点不相交的有向路径。在这道题中，具体地，如果对于三条有向边 $(u, v), (v, w), (w, r)$ 有公共点 v, w ，且 (u, v) 权值为 1，那么就相当于有向边 $((u, v), (w, r))$ 。

此时可以开始运用 LGV 引理。我们要求出源点出边的一个子集 S 和汇点入边的一个子集 T ，使得它们代表的点分别作为起点元组和终点元组时，存在至少一个不交的 n -路径。而根据 LGV 引理，我们可以用点对路径数来计算不交的 n -路径相关的值。但是由于有 $(-1)^{N(\sigma(P))}$ 项的干扰，我们计算出的行列式值可能是 0。为此，我们给每条边一个随机数，将点对路径数改为路径权值和。这样子，只要存在至少一个 n -路径，行列式就有很大概率不为 0。具体地，根据 Schwartz-Zippel lemma⁸，由于路径不会包含超过 n 条边，行列式的值是一个度最大不超过 n 的多项式。在模素数 P 意义下，如果存在至少一个 n -路径，行列式不为零的概率就 $\geq 1 - \frac{n}{P}$ 。

如果我们把点到点的方案数列成一个矩阵，那么问题实际上就是求出一个 $k \times k$ 的子矩阵，使得其行列式不为 0，在此基础上最大化 k ，相当于计算矩阵的秩。

这道题中，经过我们的不断转化后，我们得到的最终图的点对路径数，实际上就是题目给出的源点集合和汇点集合的点对路径数。我们可以通过矩阵乘法来计算这个方案。因为存在单点修改，所以我们需要使用线段树来维护区间积。时间复杂度 $O((n + q \log n)k^3)$ ，可以得到这道题的满分。

总结

一般地，在一张有向无环图中，如果边权都是 1，给这张图求最大流的时候，就可以使用同样的思想——将边看作点以将最大流转化为 n -路径相关的问题：

先给每条边一个随机权值来替换，然后列出一个和边有关的矩阵。如果源点有 n 条出边，汇点有 m 条入边，那么就能得到一个大小为 $n \times m$ 的矩阵 \mathbf{A} ：我们分别给源点的出边和

⁷ https://en.wikipedia.org/wiki/Menger's_theorem

⁸ https://en.wikipedia.org/wiki/Schwartz-Zippel_lemma

汇点的入边指定顺序, 其中 $a_{i,j}$ 表示所有从源点到汇点的有向路径 P 中, 满足源点的第 i 条出边和汇点的第 j 条出边都在路径 P 上的, $\omega(P)$ 的和。此时只要求出矩阵 \mathbf{A} 的秩就是最大流。

3 Kirchhoff's 矩阵树定理

3.1 Cauchy-Binet 公式

定理 3.1.1 (Cauchy-Binet). 给定一个 $n \times m$ 的矩阵 \mathbf{A} 和一个 $m \times n$ 的矩阵 \mathbf{B} , $\mathbf{C} = \mathbf{AB}$, 则:

$$|\mathbf{C}| = \sum_{|S|=n, S \subseteq \{1, 2, \dots, m\}} |\mathbf{A}_{n, [S]}| |\mathbf{B}_{[S], n}|$$

证明 3.1.1 (Cauchy-Binet formula). 考虑建图 $G(V, E)$, 其中 $V = L \cup R \cup D$ 组成。 $L = \{l_1, l_2, \dots, l_n\}$, $R = \{r_1, r_2, \dots, r_n\}$, $D = \{d_1, d_2, \dots, d_m\}$ 。

$E = E_L \cup E_R$ 。 $E_L = \{(l_i, d_j) | 1 \leq i \leq n, 1 \leq j \leq m\}$, 且 $\omega_{(l_i, d_j)} = a_{i,j}$ 。 $E_R = \{(d_i, r_j) | 1 \leq i \leq m, 1 \leq j \leq n\}$, 且 $\omega_{(d_i, r_j)} = b_{i,j}$ 。

那么容易发现, $c_{i,k} = \sum_{j=1}^m a_{i,j} b_{j,k} = \omega(l_i, r_k)$ 。

令起点元组 $S = (l_1, l_2, \dots, l_n)$, 终点元组 $T = (r_1, r_2, \dots, r_n)$, 由 LGV 引理得 $|\mathbf{AB}|$ 就是 S 到 T 不相交路径带权权值和。

我们枚举被经过的 d_i 集合 $D' = \{d'_1, d'_2, \dots, d'_n\} \subseteq \{1, 2, \dots, m\}$, 枚举路径的选择:

$$\begin{aligned} |\mathbf{AB}| &= \sum_{|D'|=n, D' \subseteq \{1, 2, \dots, m\}} \sum_{\sigma \in S_n} (-1)^{N(\sigma)} \sum_{\pi \in S_n} \prod_{i=1}^n (a_{i, d'_{\pi(i)}}) (b_{d'_{\pi(i)}, \sigma(i)}) \\ &= \sum_{|D'|=n, D' \subseteq \{1, 2, \dots, m\}} \sum_{\sigma \in S_n} \sum_{\pi \in S_n} (-1)^{N(\sigma\pi) + N(\pi)} \prod_{i=1}^n (a_{\pi^{-1}(i), d'_i}) (b_{d'_i, \sigma(\pi^{-1}(i))}) \\ &= \sum_{|D'|=n, D' \subseteq \{1, 2, \dots, m\}} \sum_{\sigma \in S_n} \sum_{\pi \in S_n} (-1)^{N(\sigma) + N(\pi)} \prod_{i=1}^n (a_{\pi(i), d'_i}) (b_{d'_i, \sigma(i)}) \\ &= \sum_{|D'|=n, D' \subseteq \{1, 2, \dots, m\}} \left(\sum_{\pi \in S_n} (-1)^{N(\pi)} \prod_{i=1}^n (a_{\pi(i), d'_i}) \right) \left(\sum_{\sigma \in S_n} (-1)^{N(\sigma)} \prod_{i=1}^n (b_{d'_i, \sigma(i)}) \right) \\ &= \sum_{|D'|=n, D' \subseteq \{1, 2, \dots, m\}} |\mathbf{A}_{n, [S]}| |\mathbf{B}_{[S], n}| \\ &= \sum_{|D'|=n, D' \subseteq \{1, 2, \dots, m\}} |\mathbf{A}_{n, [S]}| |\mathbf{B}_{[S], n}| \end{aligned}$$

□

3.2 定理介绍

定义 3.2.1 (Laplacian matrix). 对于一张无向图 $G = (V, E)$, 其中 $V = \{u_1, u_2, \dots, u_n\}$. 其拉普拉斯矩阵 (Laplacian matrix) \mathbf{L} 为:

$$l_{i,j} = \begin{cases} \sum_{(u_i,k) \in E} \omega_{(u_i,k)} & i = j \\ -\omega_{(u_i,u_j)} & i \neq j \text{ and } (u_i, u_j) \in E \\ 0 & \text{else} \end{cases}$$

性质 3.2.1. 拉普拉斯矩阵所有代数余子式的值都相等。

证明 3.2.1. 先证明 $C_{i,j}$ 和 $C_{i,k}$ 相等。不妨假设 $j < k$ 。

删去第 i 行后, 我们得到一系列列向量 r_1, r_2, \dots, r_n 。由 \mathbf{M} 矩阵的定义, 有 $\sum_{i=1}^n (r_i) = \mathbf{0}$, 即零向量。

我们记矩阵

$$\mathbf{A} = \begin{bmatrix} r_1 & r_2 & \cdots & r_{j-1} & r_{j+1} & \cdots & r_{k-1} & -r_j & r_{k+1} & \cdots & r_n \end{bmatrix}$$

则 \mathbf{A} 实际上就是由余子式 $\mathbf{M}_{i,k}$, 经过初等行变换 (即将除了 r_k 以外的所有 r_i 都加到 r_k 上) 得到的。

考虑将 $-r_j$ 取反, 并且通过交换两列移动到 r_{j+1} 的左边, 此时

$$\mathbf{A} = \mathbf{M}_{i,k} = (-1)^{1+(k-1)-(j+1)+1} \mathbf{M}_{i,j}$$

。

则有 $C_{i,j} = (-1)^{i+j} \mathbf{M}_{i,j} = (-1)^{i+j+k+j} \mathbf{M}_{i,k} = (-1)^{i+k} \mathbf{M}_{i,k} = C_{i,k}$ 。

对于 $C_{j,i}$ 和 $C_{k,i}$, 由于行向量和为零向量, 同理可得相等。

定理 3.2.1 (Kirchhoff's matrix tree). 对于一张简单无向图 $G = (V, E)$, 且 $\forall e \in E, \omega_e \geq 0$, 如果 $T = (V, E_T)$ 是 G 的一棵生成树, 记 $\omega(T) = \prod_{e \in E_T} \omega_e$ 。

我们记 \mathcal{T} 是 G 所有生成树的集合, 则对于 G 的拉普拉斯矩阵 \mathbf{M} 的任何一个代数余子式 $C_{i,j}$, 都有:

$$C_{i,j} = \sum_{T \in \mathcal{T}} \omega(T)$$

证明 3.2.2 (Kirchhoff's matrix tree theorem). 由于拉普拉斯矩阵所有代数余子式都相等, 我们不妨计算 $C_{1,1} = \mathbf{M}_{1,1}$, 即证明 $\mathbf{M}_{1,1} = \sum_{T \in \mathcal{T}} \omega(T)$ 。

边是一个二元组 $e = (u, v)$, 定义 $\zeta(e, x)$, 有 $\zeta(e, u) = v$, $\zeta(e, v) = u$ 。

记 $u_i < u_j$ 当 $i < j$, 记 $E = \{e_1, e_2, \dots, e_{|E|}\}$ 考虑关联矩阵 (Incidence matrix⁹) $\mathbf{A}_{|V|,|E|}$:

$$a_{i,j} = \begin{cases} \sqrt{\omega_{e_j}} & u_i \in e_j \text{ and } u_i < \zeta(e_j, u_i) \\ -\sqrt{\omega_{e_j}} & u_i \in e_j \text{ and } u_i > \zeta(e_j, u_i) \\ 0 & \text{else} \end{cases}$$

我们计算 $\mathbf{N} = \mathbf{A}\mathbf{A}^T$, 则 $N_{i,j} = \sum_{k=1}^{|E|} a_{i,k}a_{j,k}$.

当 $i = j$, 得到 $n_{i,j} = \sum_{(u_i, u_k) \in E} \omega_{(u_i, u_k)}$.

当 $i \neq j$, 得到 $n_{i,j} = -[(u_i, u_j) \in E] \omega_{(u_i, u_j)}$.

所以 $\mathbf{N} = \mathbf{M}$.

考虑 \mathbf{M} 的余子式 $\mathbf{M}_{1,1}$, 定义 \mathbf{B} 为 \mathbf{A} 删去第一行, 则方阵 $\mathbf{M}_{1,1} = \mathbf{B}\mathbf{B}^T$.

由 Cauchy-Binet 公式得到

$$|\mathbf{M}_{1,1}| = \sum_{|S|=n-1, S \subseteq \{1,2,\dots,|E|\}} |\mathbf{B}_{n-1,[S]}| |(\mathbf{B}^T)_{[S],n-1}|$$

考虑 $|\mathbf{B}_{n-1,[S]}|$, 根据行列式定义, 其意义是对于点 $u_2 \sim u_n$, 分别选一条相邻的边, 且所有边恰好被选一次. 如果 i 选择了 e_j , 则当做有向边 $(u_i, \zeta(e_j, u_i))$. 这样形成了一个有向图.

图中如果存在环, 则提取出一个环使得环上最小的点 (我们已经定义过 $<$) 最小, 将环上边反向, 此时对应着另一个选择方案. (如果把这样的对应关系看做函数 f , 则 f 是对合函数)

环的反向, 相当于作用上了一个循环排列. 我们考虑环反向之后的值:

若环长为奇数, 排列奇偶性不变, 考虑其 -1 变化了奇数个, 所以其权值积 v 和其对应的图权值积 v' 的关系为 $v = -v'$.

若环长为偶数, 排列奇偶性变化, 考虑其 -1 变化了偶数个, 所以其权值积 v 和其对应的图权值积 v' 的关系为 $v = v'$.

则出现环的权值都被两两抵消, 存在环的情况对行列式值没有贡献.

不存在环的情况下, 图只能是以 1 为根的内向树, 容易证明 $|\mathbf{B}_{n-1,[S]}|$ 的非零值至多有一个, 此时每条边选择的出边都是唯一的, 而对于 $|\mathbf{B}_{n-1,[S]}|^2$ 得到了该树的边权积.

也就是

$$|\mathbf{M}_{1,1}| = \sum_{|S|=n-1, S \subseteq E} [(V, S) \text{ is a tree}] \omega(S) = \sum_{T \in \mathcal{T}} \omega(T)$$

□

矩阵树定理在 OI 中十分普及, 因此不设置例题。

⁹ https://en.wikipedia.org/wiki/Incidence_matrix

3.3 k-生成森林

拉普拉斯矩阵的特征多项式也有一定的意义。

性质 3.3.1. 对于一张图 $G(V, E)$, 如果将其拉普拉斯矩阵 \mathbf{M} 特征值从大到小排序为 $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{|V|}$, 则有 $\lambda_{|V|} = 0$ 。

证明 3.3.1. 首先其特征值非负, 即矩阵 \mathbf{M} 是半正定的, 因为 $\mathbf{M} = \mathbf{A}\mathbf{A}^T$, 其中 \mathbf{A} 是其关联矩阵。

而由之前类似的分析很好得到 $|\mathbf{M}| = 0$, 所以一定存在 $\lambda = 0$, 实际上向量 $\mathbf{v} = [1 \ 1 \ \dots \ 1]^T$ 就是其特征向量。

□

我们定义 k -生成森林 (Spanning k -component forest) 为图的一个生成子图 (V, E) 使得这个子图有 k 个连通分量且不存在环。

记一张无向无权图 G 的 k -生成森林集合为 \mathcal{T}_k , 一个森林的权值 $Q(T)$ 为其每个连通分量顶点数量之积, 图 G 的拉普拉斯矩阵 \mathbf{M} 的特征多项式 (Characteristic polynomial¹⁰) 为 $P(x)$, 记 k -生成森林的带权和为 F_k , 则有:

$$F_k = \sum_{T \in \mathcal{T}_k} Q(T) = (-1)^{|V|-k} [x^k] P(x)$$

证明 3.3.2. 类似于矩阵树定理的证明过程, 对于我们构建的关联矩阵 \mathbf{A} 的子式 $\mathbf{A}_{[S],[T]}$ ($|S| = |T|$) 如果非零, 则需要满足不存在环, 此时每个连通分量都会向外连出一条边。

对于值非零的子式, 表示在 S 集合中的每个点选择一条出边, 不在 S 集合中的点成为了它所在的内向树的根, 形成弱连通块个数为 $|V| - |S|$ 的内向树森林。

每个子式对应唯一一个内向树森林。而从内向树森林也可以构造出唯一的子式。容易验证子式和内向树森林的对应关系构成了一个双射。

对于 S 的元素个数 $|S| = i$, 我们在计算所有主子式 $\mathbf{M}_{[S]}$ 之和的时候, 实际上就是在计算

$$\begin{aligned} v_i &= \sum_{\substack{|S|=i \\ S \subseteq \{1, 2, \dots, |V|\}}} M_{[S]} \\ &= \sum_{\substack{|S|=i \\ S \subseteq \{1, 2, \dots, |V|\}}} \sum_{\substack{|T|=|S| \\ T \subseteq \{1, 2, \dots, |E|\}}} (A_{[S],[T]})^2 \end{aligned}$$

因此在枚举所有主子式的时候, 等同于枚举所有生成内向树森林。对于一个生成森林 T , 在其每个连通分量任选一个顶点作为根, 都能得到一个内向树森林。每个生成森林 T 可以得到 $Q(T)$ 个不同的内向树森林, 这 $Q(T)$ 个内向树森林对应的子式的行列式的绝对值都是 1。

¹⁰ https://en.wikipedia.org/wiki/Characteristic_polynomial

容易证明，每个被枚举到的内向树森林恰好对应着一个生成森林，且一个生成森林能够得到的所有内向树森林都能被枚举到。所以每个生成森林一共被算了 $Q(T)$ 次，也就是 $F_k = \sum_{T \in \mathcal{T}_k} Q(T) = v_{|V|-k}$ 。

由特征多项式的性质，对于矩阵 \mathbf{M} 以及它的特征多项式 $P(x)$ ，满足 $(-1)^{|V|-k}[x^k]P(x)$ 等于 \mathbf{M} 所有 $|S| = k$ 的主子式 $\mathbf{M}_{[S]}$ 的和 $v_{|V|-k}$ 。

□

特殊地，有 $F_1 = \prod_{i=1}^{|V|-1} \lambda_i$ ，即生成树个数的 n 倍； $F_{|V|-1} = \sum_{i=1}^{|V|-1} \lambda_i$ ，即 $\text{trace}(\mathbf{M})$ ，矩阵的迹。

4 矩阵树定理拓展

我们注意到，在处理 $\mathbf{M} = \mathbf{A}\mathbf{A}^T$ 的时候，使用了两个关联矩阵 \mathbf{A} 。

实际上，我们可以同时使用关联矩阵 \mathbf{A} 和 \mathbf{B} ，使得它们承担不同的功能，也就是当计算 $\mathbf{A}\mathbf{B}^T$ 时，子式 $|\mathbf{A}_{n-1,[S]}||(\mathbf{B}^T)_{[S],n-1}|$ 中只要有一项判为 0，其值就为 0。

即使算出行列式的值可能没有很好的意义，我们还是可以在无向无权图中，利用随机权值，解决许多问题。

4.1 内/外向有根生成树

对于一张有向图 G ，定义内向有根生成树为 G 的一张子图 $G' = (V, E')$ ，使得 $|E'| = |V|-1$ ，且在 G' 任意一个点都可以通过有向边到达根。

对于外向有根生成树，采用类似的定义，表示根可以通过 G' 的有向边到达任意一个点。

我们只讨论内向树，因为外向树可以通过将所有边反向转换为内向树。

对于 $\mathbf{M} = \mathbf{A}\mathbf{B}^T$ ， \mathbf{A} 矩阵仍然承担着判断是否存在环的情况。将所有有向边看做无向边，类似关联矩阵的定义得到 \mathbf{A} ：

$$a_{i,j} = \begin{cases} \sqrt{\omega_{e_j}} & u_i \in e_j \text{ and } u_i \text{ is } e_j \text{'s head} \\ -\sqrt{\omega_{e_j}} & u_i \in e_j \text{ and } u_i \text{ is } e_j \text{'s tail} \\ 0 & \text{else} \end{cases}$$

对于 \mathbf{B} ，我们要限定边是原图中的有向边，因此 \mathbf{B} 为：

$$b_{i,j} = \begin{cases} \sqrt{\omega_{e_j}} & u_i \in e_j \text{ and } u_i \text{ is } e_j \text{'s head} \\ 0 & \text{else} \end{cases}$$

也就是只有边的起点能够选择这条边。

对于 \mathbf{M} 的值，非常容易计算：对于一条边 (u, v) 的贡献，会在 $m_{u,u}$ 处加 $\omega_{(u,v)}$ ，在 $m_{v,u}$ 处减去 $\omega_{(u,v)}$ 。

对应地，由于根 r 不能选择出边，我们选择计算代数余子式 $C_{r,r} = M_{r,r}$ ，这样子 \mathbf{A} 和 \mathbf{B} 都被删除第 r 行，表示 r 没有出边。

在使用 Cauchy - Binet 公式计算 $|\mathbf{A}_{n-1,[S]}|$ 和 $|\mathbf{B}_{n-1,[S]}|$ 时，如果 $|\mathbf{A}_{n-1,[S]}|$ 非零，那么值代表的意义就是以 r 为根的内向树，此时和 $|\mathbf{B}_{n-1,[S]}|$ 的值是相同的，也就是我们在计算 $|\mathbf{A}_{n-1,[S]}| |(\mathbf{B}^T)_{[S],n-1}|$ 时，得到的是编号在 S 中的边对应的内向树 T 的 $\omega(T)$ 。

也就是说，只要构造出了 \mathbf{M} ，计算 $C_{r,r}$ ，就能统计出以 r 为根的内向树个数。

例题 4.1.1 (无向图欧拉回路计数). 给出一个有向图 G ，求其欧拉回路 (Eulerian circuit¹¹) 方案数。

定理 4.1.1 (BEST). 对于一张有向图 G ，如果图弱连通，且对于所有点，入度等于出度，则其有欧拉回路。

则其不同的欧拉回路的数量是：

$$T(x) \prod_{u \in V} (\deg(u) - 1)!$$

其中 $T(x)$ 为以 x 为根的外向树的数量，则 x 只需要满足 $x \in V$ ， $\deg(u)$ 是 u 的入度大小。

□

对于任意一点为根的外向树数量的计算，我们可以直接使用这一小节的做法。因此我们得到一个 $O(n^3 + m)$ 的做法。

4.2 最大非二分子图

例题 4.2.1. 给定一张 n 个点 m 条边无向图，求最大的 m ，使得能够选出 m 个点 m 条边的子图 G ，使得对于 G 中任意一个连通块都不是二分图。

$$1 \leq n \leq 500, 0 \leq m \leq 10^6.$$

考虑矩阵 \mathbf{A} 满足：

$$a_{i,j} = \begin{cases} \alpha_j & u_i \in e_j \text{ and } u_i < \zeta(e_j, u_i) \\ \alpha_j & u_i \in e_j \text{ and } u_i > \zeta(e_j, u_i) \\ 0 & \text{else} \end{cases}$$

其中 α 是一个随机向量。

容易发现，和关联矩阵相比，我们将 1 和 -1 的系数改为了 1 和 1。

¹¹ https://en.wikipedia.org/wiki/Eulerian_path

在取 n 点 n 边的图时，如果图中存在孤立点，则其一定是二分图。则图一定是由若干基环树组成。此时每个点可以指定一条出边，正好和行列式意义对应上。

采用类似的分析方法，考虑贡献的抵消。

若环长为奇数，排列奇偶性不变，所以其权值积 v 和其对应的图权值积 v' 的关系为 $v = v'$ 。

若环长为偶数，排列奇偶性变化，所以其权值积 v 和其对应的图权值积 v' 的关系为 $v = -v'$ 。

于是我们成功地将所有偶环的贡献抵消。

因此只需要计算 \mathbf{A} 的秩即可。由于 m 很大，得到的是 $O(n^2m)$ 的做法（虽然可能存在优化）。

但是我们只要计算 \mathbf{AA}^T 的秩即可得到相同的结果，因为 $\text{rank}(\mathbf{AA}^T) = \text{rank}(\mathbf{A})$ 。

而类似于拉普拉斯矩阵， \mathbf{AA}^T 的值也十分好算，得到一个 $O(n^3 + m)$ 的做法。

4.3 拟阵交

例题 4.3.1. 有两个无向图，点数相同，边数也相同。对于编号 i ，它在 G_1 代表边 (x_1, y_1) ，在 G_2 代表边 (x_2, y_2) 。

我们想选出一个编号的集合，使得对于一个编号 i ，如果它在集合中出现了，那么它在 G_1 中会被连接，它在 G_2 中也会被连接。

要求选出最多的编号，使得两个图连上边后都不存在环。

使用类似的分析方法，先给所有边赋上随机权值。我们通过计算 G_1 的关联矩阵 \mathbf{A} 和 G_2 的关联矩阵 \mathbf{B} ，通过计算 \mathbf{AB}^T 的秩，即可得到一个 $O(n^3 + m)$ 的做法。

根据上面的经验，如果拟阵可以表示成矩阵的形式，即线性拟阵（Linear matroid¹²），则同样可以使用矩阵来计算两个拟阵的交大小：

对于表示成的列向量，我们将其乘上随机权值后排列成类似关联矩阵的形式，得到两个矩阵 \mathbf{A}, \mathbf{B} ，我们只需要计算 \mathbf{AB}^T 的秩即可得到拟阵交的大小。

¹² https://en.wikipedia.org/wiki/Matroid_representation

5 简单的图的积问题

5.1 Kronecker 积与图的 Cartesian 积的关系

定义 5.1.1 (Kronecker product). 给定两个矩阵 \mathbf{A}, \mathbf{B} , 其中 \mathbf{A} 是 $n \times m$ 的矩阵, 则定义 \mathbf{A} 和 \mathbf{B} 的 *Kronecker 积*¹³ 为:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \cdots & a_{1,m}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \cdots & a_{2,m}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1}\mathbf{B} & a_{n,2}\mathbf{B} & \cdots & a_{n,m}\mathbf{B} \end{bmatrix}$$

我们将其表示为分块矩阵的形式。

Kronecker 积有一个重要的性质:

性质 5.1.1 (Mixed-product).

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD})$$

5.2 张量积与 Kronecker 积

定义 5.2.1 (Tensor product of graphs). 对于两张图 $G_1(V_1, E_1), G_2(V_2, E_2)$, 定义其 *Tensor 积* 为:
 $G(V, E) = G_1 \times G_2$.

其中

$$V = \{(u, v) | u \in V_1, v \in V_2\}$$

$$E = \{((u_1, v_1), (u_2, v_2)) | (u_1, u_2) \in E_1, (v_1, v_2) \in E_2\}$$

如果将 (u, v) 按字典序排序, 那么记 G_1 邻接矩阵为 \mathbf{A}_1 , G_2 邻接矩阵为 \mathbf{A}_2 , G 的邻接矩阵为 \mathbf{A} , 由我们的定义, 则有 $\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{A}_2$ 。

对于 \mathbf{A}_1 的特征值 λ_i 和特征向量 \mathbf{x}_i , 以及 \mathbf{A}_2 的特征值 μ_j 和特征向量 \mathbf{y}_j 则有:

$$\begin{aligned} & (\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{x}_i \otimes \mathbf{y}_j) \\ &= (\mathbf{A}_1 \mathbf{x}_i) \otimes (\mathbf{A}_2 \mathbf{y}_j) \\ &= (\lambda_i \mathbf{x}_i) \otimes (\mu_j \mathbf{y}_j) \\ &= (\lambda_i \mu_j) \mathbf{x}_i \otimes \mathbf{y}_j \end{aligned}$$

也就是 $\lambda_i \mu_j$ 对应着特征向量 $\mathbf{x}_i \otimes \mathbf{y}_j$, 一共 $|V(G_1)||V(G_2)| = |V(G_1 \square G_2)|$ 个。

¹³ https://en.wikipedia.org/wiki/Kronecker_product

5.3 Cartesian 积与 Kronecker 和

定义 5.3.1 (Cartesian product of graphs). 对于两张图 $G_1(V_1, E_1), G_2(V_2, E_2)$, 定义其 *Cartesian 积*¹⁴ 为: $G(V, E) = G_1 \square G_2$.

其中

$$\begin{aligned} V &= \{(u, v) | u \in V_1, v \in V_2\} \\ E &= \{((u_1, v), (u_2, v)) | (u_1, u_2) \in E_1, v \in V_2\} \cup \\ &\quad \{((v, u_1), (v, u_2)) | (u_1, u_2) \in E_2, v \in V_1\} \end{aligned}$$

现在有两张图 G_1 和 G_2 首先, 我们先计算 $G_1 \square G_2$ 的邻接矩阵 A :

如果将 (u, v) 按字典序排序, 那么记 G_1 邻接矩阵为 A_1 , G_2 邻接矩阵为 A_2 , 由我们的定义:

对于边集 $\{((u_1, v), (u_2, v)) | (u_1, u_2) \in E_1, v \in V_2\}$, 实际上就是 $A_1 \otimes I_{|V(G_2)|}$ 。

对于边集 $\{((v, u_1), (v, u_2)) | (u_1, u_2) \in E_2, v \in V_1\}$, 实际上就是 $I_{|V(G_1)|} \otimes A_2$ 。

则 $A = A_1 \otimes I_{|V(G_2)|} + I_{|V(G_1)|} \otimes A_2$ 。

实际上, 对于 Cartesian 积与 Kronecker 积的关系, 我们称之为 Kronecker 和:

定义 5.3.2 (Kronecker sum). 对于 $n \times n$ 矩阵 A 和 $m \times m$ 矩阵 B , 定义其 *Kronecker 和*为:

$$A \oplus B = A \otimes I_m + I_n \otimes B$$

对于 A_1 的特征值 λ_i 和特征向量 x_i , 以及 A_2 的特征值 μ_j 和特征向量 y_j 则有:

$$\begin{aligned} & (A_1 \otimes I_{|V(G_2)|} + I_{|V(G_1)|} \otimes A_2)(x_i \otimes y_j) \\ &= (A_1 x_i) \otimes (I_{|V(G_2)|} y_j) + (I_{|V(G_1)|} x_i) \otimes (A_2 y_j) \\ &= \lambda_i x_i \otimes y_j + \mu_j x_i \otimes y_j \\ &= (\lambda_i + \mu_j) x_i \otimes y_j \end{aligned}$$

也就是 $\lambda_i + \mu_j$ 对应着特征向量 $x_i \otimes y_j$, 一共 $|V(G_1)||V(G_2)| = |V(G_1 \square G_2)|$ 个。

类似于邻接矩阵的分析方法, 对拉普拉斯矩阵分析可以得到相同的结果:

对于拉普拉斯矩阵 L_1 和 L_2 , 对于 $G_1 \square G_2$ 的拉普拉斯矩阵为 $L = L_1 \oplus L_2$ 。

同样, 其特征值也是所有 $\lambda_i + \mu_j$, 其特征向量是 $x_i \otimes y_j$ 。

5.4 有关图的积的问题

基于我们对其邻接矩阵特征值的分析, 可以联系上先前特征值和图意义的关系, 解决一些计数的问题, 尤其是生成树问题。

¹⁴ https://en.wikipedia.org/wiki/Cartesian_product_of_graphs

例题 5.4.1. 给定若干个图 G_1, G_2, \dots, G_k , 求图 $G = G_1 \times G_2 \times \dots \times G_k$ 中, 长度为 L 的起点和终点相等的不同路径数。答案对 998244353 取模。

$$\sum_{i=1}^k |V(G_i)| \leq 500, L \leq 10^{18}.$$

题目相当于求 G 的邻接矩阵 \mathbf{A} 的 L 次方的迹 $\text{trace}(\mathbf{A}^L)$ 。

实际上就是特征值 L 次求和:

$$\begin{aligned} & \sum (\lambda_i \mu_j \nu_k \dots)^L \\ &= \sum (\lambda_i^L \mu_j^L \nu_k^L \dots) \\ &= (\sum_i \lambda_i^L) (\sum_j \mu_j^L) (\sum_k \nu_k^L) \dots \end{aligned}$$

接下来的问题就是对于一张图, 计算其特征值 L 次方和, 即计算:

$$[x^L] \left(\sum_i \frac{1}{1 - \lambda_i x} \right)$$

令其特征多项式为 $f(x)$, 则 $g(x) = \prod_i (1 - \lambda_i x)$ 是将其系数翻转后得到的结果。

则

$$[x^L] \left(\sum_i \frac{1}{1 - \lambda_i x} \right) = [x^{L-1}] \frac{g'(x)}{g(x)}$$

以 $g'(x)$ 的系数为初值做线性递推即可。

对于特征多项式, 可以通过右乘初等行变换矩阵 \mathbf{P} , 同时左乘其逆 \mathbf{P}^{-1} , 直到得到一个 Hessenberg 矩阵¹⁵ \mathbf{H} 。由于其特殊的结构, 对于一个值 x 可以通过 $O(n^2)$ 时间复杂度求得行列式 $|x\mathbf{I} - \mathbf{H}|$ 的值。求出足够多点值后即可通过插值得到其特征多项式, 总时间复杂度 $O(n^3)$ 。

令 $n_i = |V(G_i)|$, 时间复杂度 $O(\sum_{i=1}^k (n_i^3 + n_i \log n_i \log L))$ 。

例题 5.4.2. 给定若干个图 G_1, G_2 , 求图 $G = G_1 \square G_2$ 的生成树个数。

$$|V(G_1)| + |V(G_2)| \leq 500.$$

很容易发现答案是:

$$\frac{1}{|V(G_1)||V(G_2)|} \prod_{i=1}^{|V(G_1)|} \prod_{j=1}^{|V(G_2)|} (\lambda_i + \mu_j)$$

计算出特征多项式后, 通过结式 (Resultant¹⁶) 即可在 $O(\max(|V(G_1)|, |V(G_2)|)^2)$ 的复杂度下得到答案。

很可惜由于即使在模意义下也难以求出所有特征值 (例如在模 5 意义下就无法解出 $a^2 \equiv 3 \pmod{5}$), 对于图任意多的情况, 作者并不知道有什么好的做法, 欢迎大家与作者讨论。

¹⁵ https://en.wikipedia.org/wiki/Hessenberg_matrix

¹⁶ <https://en.wikipedia.org/wiki/Resultant>

6 总结

关于这些与图相关的矩阵，还有很多有趣的性质。由于作者才疏学浅，所以介绍的内容较为简单。

OI 中这一类题目并不是很多，很大一部分都是矩阵树定理的直接应用。作者希望本文能起到抛砖引玉的作用，希望看到更多相关题目以及更多有意思的拓展出现。

7 参考文献

- [1] Wikipedia, the free encyclopedia, "Lindström–Gessel–Viennot lemma", https://en.wikipedia.org/wiki/Lindström–Gessel–Viennot_lemma
- [2] Wikipedia, the free encyclopedia, "Cauchy–Binet formula", https://en.wikipedia.org/wiki/Cauchy–Binet_formula
- [3] Wikipedia, the free encyclopedia, "Kirchhoff's theorem", https://en.wikipedia.org/wiki/Kirchhoff's_theorem
- [4] Wikipedia, the free encyclopedia, "BEST theorem", https://en.wikipedia.org/wiki/BEST_theorem
- [5] Wikipedia, the free encyclopedia, "Kronecker product", https://en.wikipedia.org/wiki/Kronecker_product
- [6] Biggs, N. (1993). Algebraic Graph Theory. Cambridge University Press.

8 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高闻远的指导。

感谢父母对我的培养和教育。

感谢学校的栽培，符水波老师和应平安老师的教导和同学们的帮助。

感谢翁伟捷同学，虞皓翔同学，钱易同学与我的交流和讨论，给了我不少启发。

感谢孙睿泽同学，宣毅鸣同学，施开成同学为本文审稿。

感谢各信息学奥赛网站上的出题人们提供了题目，给我带来了启发。

浅谈棋盘模型在计数问题中的应用

长郡中学 彭思进

摘要

本文介绍了组合计数问题中的棋盘模型，并简要探讨了其在若干计数问题中的应用。

目录

1 前言	2
2 约定	2
3 定义	3
4 基础定理	4
4.1 棋盘数与命中数的关系	4
4.2 展开定理	5
4.3 分离定理	5
5 禁区排列问题	6
6 Ferrers 棋盘的棋盘多项式	7
6.1 下降幂棋盘多项式分解定理	8
6.2 阶梯棋盘与斯特林数	10
6.3 棋盘等价类	13
6.4 分解定理的变体	14
7 排列降低计数	17
7.1 Foata 第一基本变换	18
7.2 k -超过数计数	20
7.3 X - Y 降低计数	21

8 Ferrers 棋盘的排列逆序对计数	23
8.1 k-棋盘数的 q-analogue 形式	23
8.2 q-analogue 形式下的分解定理	25
9 Ferrers 棋盘的排列环数计数	26
9.1 问题引入	26
9.2 有向图的覆盖多项式	27
9.3 Ferrers 棋盘覆盖多项式分解定理	28
9.4 分解定理在减 Ferrers 棋盘上的拓展	30
10 总结	33
11 致谢	33

1 前言

棋盘是一类重要的组合模型，其基本问题为计算在一个棋盘上放置若干个车使得任意两个车无法互相攻击的方案数。其简洁直观的特点使得其在大量计数问题中得到了应用，但在信息学竞赛中大部分题目仅考察了最基本的禁区排列问题。本文将在禁区排列问题的基础上介绍通过棋盘理论可以求解的其他计数问题。

2 约定

假定读者已经掌握多项式的基本运算与基本的组合计数知识。

对于一个多项式或幂级数 $f(x)$ ，约定 $[x^k]f(x)$ 为 $f(x)$ 的 x^k 项系数。

约定 $x^{\underline{n,m}} = x(x-m)(x-2m)\cdots(x-(n-1)m)$ 为 x 的 m 阶 n 次下降幂，特别地，约定 $x^{\underline{n}} = x^{\underline{n,1}}$ 为 x 的 n 次下降幂；约定 $x^{\overline{n,m}} = x(x+m)(x+2m)\cdots(x+(n-1)m)$ 为 x 的 m 阶 n 次上升幂，特别地，约定 $x^{\overline{n}} = x^{\overline{n,1}}$ 为 x 的 n 次上升幂。

约定 $\binom{n}{m}$ 表示组合数， $\left\{ \begin{matrix} n \\ m \end{matrix} \right\}$ 表示第二类斯特林数， $[n_m]$ 表示无符号第一类斯特林数， $\langle n_m \rangle$ 表示欧拉数¹。本文中仅考虑其中 n, m 为整数的情况。

约定 $\text{Sym}(n)$ 表示 n 阶对称群，即所有 n 阶排列构成的集合， \mathbb{N} 表示自然数集， \mathbb{Z} 表示整数集， \mathbb{N}_+ 表示正整数集， \mathbb{R} 表示实数集。

约定对于每个点入出度均不超过 1 的有向图 G ， $\text{cycle}(G)$ 表示图 G 的环数。

对于任意布尔表达式 λ ，定义符号 $[\lambda] = \begin{cases} 1, \lambda \text{ 为真} \\ 0, \lambda \text{ 为假} \end{cases}$

¹有关欧拉数的内容可参考 <https://www.luogu.com.cn/blog/Karry5307/eulerian-numbers>

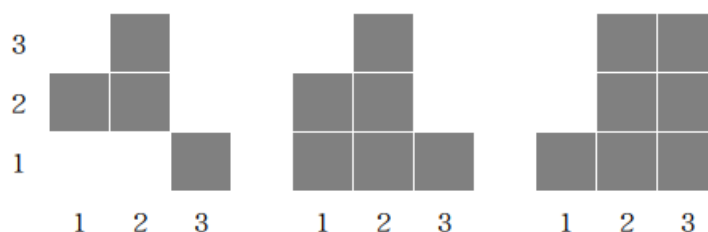
3 定义

定义 3.1 (棋盘). 定义棋盘为一个有限二元组集合 S 满足对于任意的 $(i, j) \in S$ 有 $i, j \in \mathbb{N}_+$ 。

对于棋盘 S 和整数 $n \geq \max_{(i,j) \in S} i, m \geq \max_{(i,j) \in S} j$, 可以对 S 给出直观描述: 考虑一个 $n \times m$ 的网格, 初始每个格子是白色的。对于每个属于 S 的二元组 (i, j) , 将棋盘从左至右第 i 列、从下至上第 j 行的格子染成黑色, 最后黑色格子组成的图形对应棋盘 S 。图 1 所示的是在 $n = m = 3$ 时三个棋盘的直观描述, 其中左棋盘对应 $\{(1, 2), (2, 2), (2, 3), (3, 1)\}$ 。

为了描述方便, 约定下文中除特殊声明外 $n = \max_{(i,j) \in S} i, m = \max_{(i,j) \in S} j$ 。

图 1: 棋盘、轮廓线棋盘、Ferrers 棋盘



定义 3.2 (轮廓线棋盘). 对于棋盘 S 和整数 n , 若 $(\max_{(i,j) \in S} i) \leq n$ 且对于每个 $1 \leq i \leq n$ 均存在非负整数 a_i 使得对于所有正整数 k , $(i, k) \in S$ 当且仅当 $k \leq a_i$, 则称 S 为轮廓线棋盘, 记 $S = F(a_1, a_2, \dots, a_n)$ 。特别地, 允许存在 $a_i = 0$, 此时第 i 列没有格子。

定义 3.3 (Ferrers 棋盘). 对于轮廓线棋盘 $S = F(a_1, \dots, a_n)$, 若 $a_1 \leq a_2 \leq \dots \leq a_n$, 则称棋盘 S 为 Ferrers 棋盘, 同时若不存在 $1 \leq i < n$ 满足 $a_i = a_{i+1}$ 则称棋盘 S 为严格增 Ferrers 棋盘。类似地, 若 $a_1 \geq a_2 \geq \dots \geq a_n$, 则称棋盘 S 为减 Ferrers 棋盘。

在图 1 中, 中间和右边的两个棋盘均是轮廓线棋盘, 而仅有最右边的棋盘是 Ferrers 棋盘, 它们的表示分别是 $F(2, 3, 1)$ 和 $F(1, 3, 3)$ 。

定义 3.4 (k -棋盘数²与棋盘多项式). 对于棋盘 S , 定义其子集 T 合法当且仅当对于任意两个不同的二元组 $(p_1, q_1), (p_2, q_2) \in T$, $p_1 \neq p_2$ 且 $q_1 \neq q_2$ 。定义 $R_k(S) = \{T \subseteq S \mid T \text{ 合法}, |T| = k\}$, 则棋盘 S 的 k -棋盘数 $r_k(S) = |R_k(S)|$, 棋盘 S 的棋盘多项式为 $F_S(x) = \sum_{k=0}^{|S|} r_k(S)x^k$ 。

定义 3.5 (k -命中数³). 对于 $n \in \mathbb{N}_+$, 定义棋盘 $B_n = \{(i, j) \mid 1 \leq i, j \leq n\}$ 。对于棋盘 $S \subseteq B_n$ 定义 $H_{k,n}(S) = \{T \in R_n(B_n) \mid |T \cap S| = k\}$, 则棋盘 S 的 k -命中数 $h_{k,n}(S) = |H_{k,n}(S)|$ 。

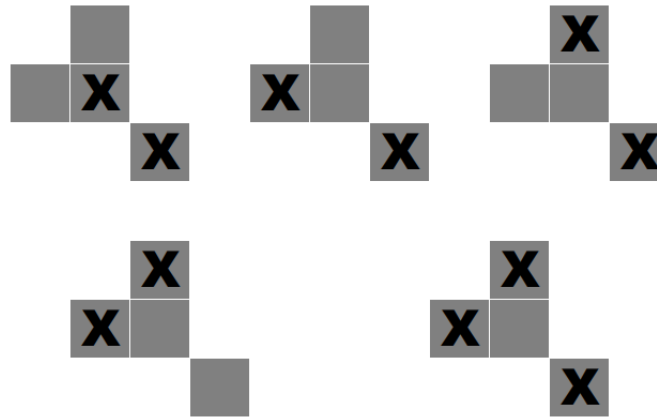
²笔者未在中文文献中找到“kth rook number”的翻译。

³笔者未在中文文献中找到“kth hit number”的翻译。

对应到直观描述上, k -棋盘数的组合意义即为在棋盘上放置 k 个车使得任意两个车不在同一行且不在同一列, 即互相不攻击的方案数; k -命中数的组合意义为所有在 B_n 上放 n 个车使得任意两个车不能互相攻击的方案中放在棋盘 S 上的车有 k 个的方案数。

对于图 1 中左边的棋盘 $S = \{(1, 2), (2, 2), (2, 3), (3, 1)\}$, 有 $r_0(S) = 1, r_1(S) = |S| = 4, r_2(S) = 4, r_3(S) = 1$, 而对于 $k \geq 4, r_k(S) = 0$; $h_{0,3}(S) = 1, h_{1,3}(S) = 3, h_{2,3}(S) = 1, h_{3,3}(S) = 1$ 。图 2 中描述了 $R_2(S)$ 与 $R_3(S)$ 中的元素, 其中放上了车的位置用 X 表示。

图 2: $R_2(S)$ 与 $R_3(S)$



4 基础定理

4.1 棋盘数与命中数的关系

定理 4.1 ([8]). 对于棋盘 $S \subseteq B_n$,

$$\sum_{k=0}^n h_{k,n}(S)x^k = \sum_{k=0}^n r_k(S)(n-k)!(x-1)^k \quad (1)$$

证明. 在定理中用 $x+1$ 替换 x 可以得到

$$\sum_{k=0}^n h_{k,n}(S)(x+1)^k = \sum_{k=0}^n r_k(S)(n-k)!x^k \quad (2)$$

对比两侧 x^i 项系数, 即需要说明 $\sum_{k=i}^n h_{k,n}(S)\binom{k}{i} = r_i(S)(n-i)!$ 。

考察 $\sum_{k=i}^n h_{k,n}(S)\binom{k}{i}$ 的组合意义, 即为: 对于所有命中数 $\geq i$ 的方案, 在所有命中的位置中选 i 个作为特殊的命中位置的方案数总和。交换求和顺序, 先选择 i 个特殊的命中位置再计算剩余部分的方案数, 可以得到相等的结果。选择这 i 个位置的方案数为 $r_i(S)$, 而剩余的 $n-i$ 个位置的选择没有其他限制, 方案数等于 $n-i$ 行和 $n-i$ 列匹配的方案数, 即 $(n-i)!$ 。

故 $\sum_{k=i}^n h_{k,n}(S) \binom{k}{i} = r_i(S)(n-i)!$, 即得 (2) 式和 (1) 式。 \square

定理 4.1 揭示了 $r_k(S)$ 和 $h_{k,n}(S)$ 的联系, 对于绝大多数的命中数计算问题, 需要通过定理 4.1 转化为棋盘数计算问题。

4.2 展开定理

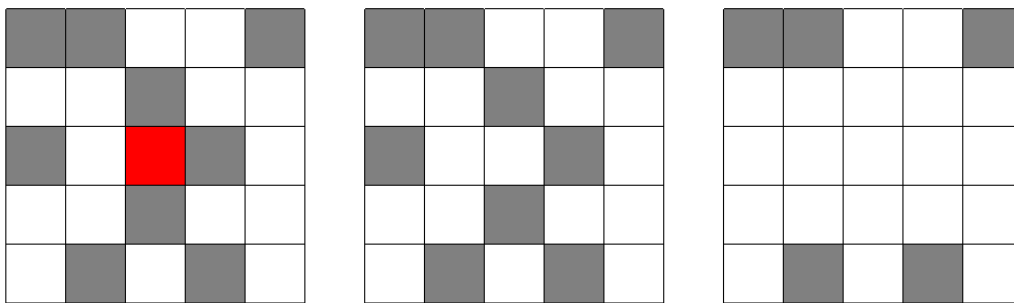
定理 4.2 (展开定理, [11]). 给定非空棋盘 S , 任取 $(p, q) \in S$, 设 $S_1 = S - \{(p, q)\}, S_2 = \{(p_1, q_1) \in S \mid p_1 \neq p, q_1 \neq q\}$, 有

$$F_S(x) = F_{S_1}(x) + xF_{S_2}(x) \tag{3}$$

证明. 对于确定的 k , 将 $R_k(S)$ 分为不包含 (p, q) 的方案集合 $R_{k,0}(S)$ 和包含 (p, q) 的方案集合 $R_{k,1}(S)$ 。 $R_{k,0}(S) = R_k(S_1)$, 而在 $R_{k-1}(S_2)$ 包含的方案中加入 (p, q) 可得 $R_{k,1}(S)$ 。 \square

直观理解即对于位置 (p, q) 枚举其是否放车, 若不放车则这个位置从 S 中删除; 否则这个车能够攻击到的所有位置都不能放车, 故被删除。如图 3 所示, 在左侧棋盘 S 中选择 $(3, 3)$ 作为位置 (p, q) , 那么 S_1 和 S_2 分别为中间和右侧的两个棋盘。

图 3: 展开定理图示



利用定理 4.2 可以得到易于编程实现的计算棋盘多项式的朴素算法, 但效率通常不高。

4.3 分离定理

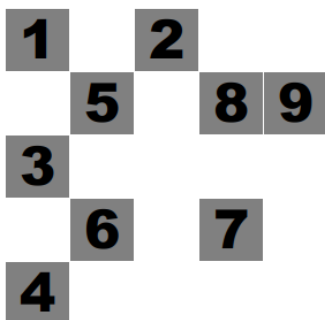
定理 4.3 (分离定理, [11]). 对于棋盘 S 和 $S_1 \subseteq S$, 设 $S_2 = S - S_1$, 若对于任意 $(p_1, q_1) \in S_1, (p_2, q_2) \in S_2$ 都有 $p_1 \neq p_2, q_1 \neq q_2$, 则

$$F_S(x) = F_{S_1}(x)F_{S_2}(x) \tag{4}$$

证明. 对于 S_1 的任一合法子集 T 和 S_2 的任一合法子集 \bar{T} , 由于 S_1 和 S_2 的行列不交, 因此 $T + \bar{T}$ 是 S 的合法子集. 同时 S 的合法子集 $T + \bar{T}$ 可恰好分为属于 S_1 的部分 T 和属于 S_2 的部分 \bar{T} . 故 S 的合法子集与 S_1, S_2 的合法子集构成的二元组一一对应, 则 $r_k(S) = \sum_{i=0}^k r_i(S_1)r_{k-i}(S_2)$, 即得式 (4). \square

图 4 中, 编号属于区间 $[1, 4]$ 的格子与属于区间 $[5, 9]$ 的格子行列无交, 因此可以分别算出它们的棋盘多项式最后卷积合并. 在部分特殊棋盘中, 通过定理 4.3 将大棋盘拆成若干个小棋盘有利于简化计算.

图 4: 分离定理图示



5 禁区排列问题

这一部分将介绍两个禁区排列问题。

例 5.1 (错排问题⁴). 给定 n , 求有多少个 n 阶排列 π 满足不存在 $1 \leq i \leq n$ 使得 $i = \pi_i$, 答案对大质数取模. $1 \leq n \leq 10^7$.

设 $S = \{(1, 1), (2, 2), \dots, (n, n)\}$, 则错排问题的解为 $h_{0,n}(S)$. 由定理 4.1 有 $h_{0,n}(S) = \sum_{i=0}^n r_i(S)(n-i)!(-1)^i$. 由于 S 中任意两个格子不在同行或同列, 因此 $r_i(S) = \binom{n}{i}$. 故有

$$h_{0,n}(S) = \sum_{i=0}^n (-1)^i \binom{n}{i} (n-i)! = n! \sum_{i=0}^n \frac{(-1)^i}{i!} \tag{5}$$

例 5.2 (夫妻分座问题⁵). 给定 n 和两个 n 阶排列 p, q , 对于 $k \in [0, n]$ 求有多少个 n 阶排列 π 满足 $\sum_{i=1}^n [\pi_i = p_i \text{ 或 } \pi_i = q_i] = k$, 答案对 998244353 取模. $1 \leq n \leq 2 \times 10^5$.

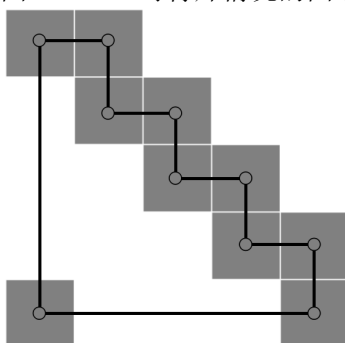
构造棋盘 $S = \{(i, j) \mid 1 \leq i, j \leq n, j = p_i \text{ 或 } j = q_i\}$, 则需要对 $k \in [0, n]$ 计算 $h_{k,n}(S)$. 仍使用定理 4.1 转化为计算 $F_S(x)$, 但这个棋盘并没有错排问题棋盘的显然的优秀性质.

不妨先考虑一类特殊情况: $n > 1, p_i = i, q_i = (i \bmod n) + 1$. 此时其直观描述如图 5.

⁴来源: 经典问题

⁵来源: 经典问题改编

图 5: $n = 5$ 时特殊情况的图示



对每个格子建一个点，将两个在同行或同列的格子间连一条边，如图 5，则其恰好构成长度为 $2n$ 的环，而 $r_k(S)$ 等于在这个环上选 k 个点使得任意两个点不相邻的方案数。

该问题在链上较容易解决，考虑断环成链。对于每个点，强制其不选并计算剩余 $2n - 1$ 个点构成的链上选择 k 个点满足任意两个点不相邻的方案数，即计算有多少个递增序列 $1 \leq a_1 < \dots < a_k \leq 2n - 1$ 满足对于所有 $1 \leq i < k, a_{i+1} \neq a_i + 1$ 。

对于合法序列 $A = \{a_1, \dots, a_k\}$ 设 $T(A) = \{t_1 = a_1 - 1, \dots, t_k = a_k - k\}$ ，则 $T(A)$ 需要满足严格递增且 $t_0 \geq 0, t_k \leq 2n - k - 1$ 。而满足该条件的序列 $T(A)$ 唯一对应一个合法序列 A ，故合法的 A 和 $T(A)$ 数量相等。而合法的 $T(A)$ 序列与在 $[0, 2n - k - 1]$ 中选 k 个数的方案一一对应，故其数量为 $\binom{2n-k}{k}$ 。

对于 $2n$ 个点均计算一次答案并求和，此时每个选择 k 个点的合法方案都会贡献 $2n - k$ 次，故 $r_k(S) = \frac{2n}{2n-k} \binom{2n-k}{k}$ 。特殊情况得到解决。

拓展到一般情况，类似图 5 的方式建立一张图，此时每个点的度数为 0 或者 2。值得注意的是不会存在度数为 1 的点。图会形成若干个连通块，连通块之间行列无交，根据定理 4.3 可以分别计算每个连通块的棋盘多项式然后卷积合并。

度数为 0 的点构成单独的连通块，棋盘多项式为 $x + 1$ 。

其他连通块每个节点度数为 2，故其构成一个环，棋盘多项式求解与特殊情况一致。

使用分治 FFT 合并每个连通块的棋盘多项式得到 $r_k(S)$ ，再使用 FFT 优化二项式反演得到 $h_{k,n}(S)$ 即可解决原问题。复杂度 $O(n \log^2 n)$ 。

特别地，笔者对于该问题的另一个特殊情况： $p_i = i, q_i = n - i + 1$ 进行了研究，得到了更优复杂度的做法，由于篇幅原因本文不进行阐述，具体内容可见参考文献 [13]。

6 Ferrers 棋盘的棋盘多项式

Ferrers 棋盘由于性质优秀，且与其他的组合工具有密切联系，故一直是棋盘理论中重要的研究对象。本节将会对 Ferrers 棋盘的棋盘多项式进行简单讨论。

6.1 下降幂棋盘多项式分解定理

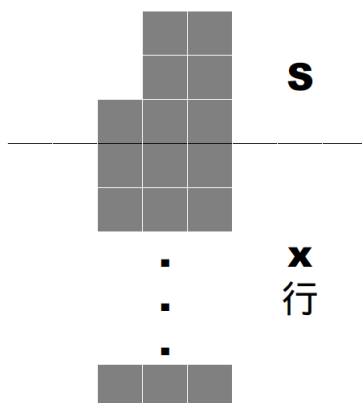
定理 6.1 (下降幂棋盘多项式分解定理, [6]). 对于 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n)$,

$$\sum_{k=0}^n r_k(S)x^{n-k} = \prod_{k=1}^n (x + a_k - k + 1) \tag{6}$$

证明. 式 (6) 两侧均为关于 x 的多项式, 故仅需要在无限多个点处证明式 (6) 成立即可证明式 (6) 恒成立. 考虑在 $x \in \mathbb{N}$ 处证明.

如图 6 所示, 设 S_x 为在棋盘 S 下方加入额外 x 行得到的棋盘. 考虑使用两种方法计算 $r_n(S_x)$:

图 6: $S = F(1, 3, 3)$ 时 S_x 所表示的棋盘



1. 从左往右确定每列的放车位置。

第一列有 $x + a_1$ 个位置可以放车; 第二列有 $x + a_2$ 个位置可以放车, 但由于第一列放了车且 $a_1 \leq a_2$, 因此恰好存在一个位置不能放车, 方案数为 $x + a_2 - 1$; 由于前两列的两个车一定不在同一行, 故第三列恰有两个位置无法放车, 方案数为 $x + a_3 - 2$

第 i 列放置时总会由于前 $i - 1$ 列放置的车导致 $i - 1$ 个位置无法放车, 方案数为 $x + a_i - (i - 1)$ 。每一列方案数是独立的, 与前面列的具体放置方案无关, 故

$$r_n(S_x) = \prod_{k=1}^n (x + a_k - (k - 1))$$

2. 枚举 S 的合法子集 T , 计算所有合法方案中与 S 交集为 T 的方案数。

设 $|T| = k$, 那么集合 T 中 k 个车对应的列不能再放车, 只需要确定剩下 $n - k$ 列的放车位置. 而由于 T 恰好是 S 与放车方案的交集, 故这 $n - k$ 列的车只能放在额外加入的 x 行内。

类似第一种方法的讨论，在这 $n - k$ 列内依次放车。第一列放车方案数为 x ，第二列由于第一列的车放车方案数为 $x - 1$ ，第三列放车方案数为 $x - 2 \cdots \cdots$ 总方案数为 x^{n-k} 。而选择合法子集 T 的方案数为 $r_k(S)$ ，故

$$r_n(S_x) = \sum_{k=0}^n r_k(S) x^{n-k}$$

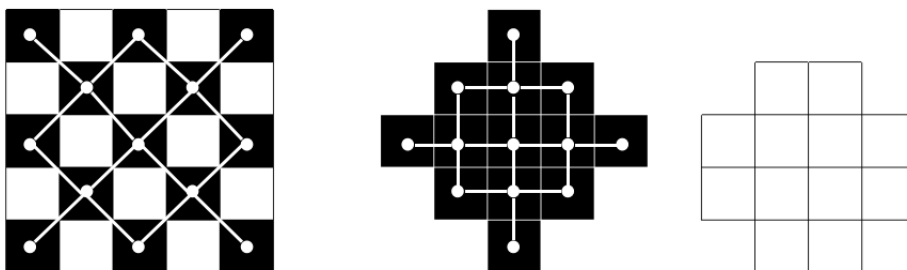
合并两式即得式 (6)。 □

一般的 Ferrers 棋盘可通过分治 FFT 计算式 (6) 右侧的多项式，然后通过普通多项式转下降幂多项式⁶得到棋盘多项式，复杂度 $O(n \log^2 n)$ 。

一个更好的做法是在分治 FFT 计算右侧多项式时直接维护下降幂多项式而不是普通多项式，在计算下降幂多项式乘法时使用卷积得到其在 $1, 2, \dots, n$ 处的点值，相乘后卷积还原⁷。复杂度仍为 $O(n \log^2 n)$ ，但其可以避免普通多项式转下降幂多项式中常数较大的多项式多点求值。

例题 6.1 (Gnutella Chessmaster⁸). 给定一个 $n \times n$ 的棋盘，对于 $k \in [1, 2n - 1]$ 求在棋盘上放 k 个国际象棋中的象使得它们两两互不攻击的方案数，对 998244353 取模。两个象 (x_1, y_1) 和 (x_2, y_2) 互相攻击当且仅当 $|x_1 - x_2| = |y_1 - y_2|$ 。 $1 \leq n \leq 10^5$ 。

图 7: $n = 5$ 时的黑白染色和旋转之后得到的黑白棋盘



对棋盘进行黑白染色，观察攻击方式可以知道两个位于不同颜色的象不会互相攻击。使用定理 4.3 将黑色格子和白色格子的方案数分开计算，最后使用卷积合并。

由于象攻击的方向是斜对角线不易处理，考虑将棋盘旋转 45° ，这样象攻击的方向与车相同，放置的限制也就是两个象不能在同行或同列。图 7 展示了 $n = 5$ 的旋转结果，为了更好地描述限制的对应关系，图 7 的黑色棋盘使用了图 5 中描述限制的方法。

⁶可参考 <https://www.luogu.com.cn/problem/P5383>

⁷可参考 <https://www.luogu.com.cn/problem/P5394>

⁸来源: Petrozavodsk Programming Camp Summer 2018 Day 3: MIPT Contest G

由于交换行列不会改变棋盘的棋盘多项式,因此可以先对所有行按照格子个数从大到小排序,再对所有列按照位置个数从小到大排序,可以发现 $n = 5$ 时黑色棋盘等于 $F(1, 1, 3, 3, 5)$, 而白色棋盘等于 $F(2, 2, 4, 4)$ 。

更一般的性质是: 棋盘大小为 $n \times n$ 时两个棋盘都可以通过交换行列变为每一列高度构成的可重集等于对应颜色所有对角线长度构成的可重集的 Ferrers 棋盘, 即黑色棋盘为 $F(1, 1, 3, 3, 5, 5, \dots)$, 白色棋盘为 $F(2, 2, 4, 4, 6, 6, \dots)$ 。由于两个棋盘求解仅有细节上的区别, 下文仅考虑黑色棋盘的求解。

对于黑色棋盘, 运用定理 6.1 得

$$\sum_{k=0}^n r_k(S)x^{n-k} = (x+1)^{\lceil \frac{n}{2} \rceil} x^{\lfloor \frac{n}{2} \rfloor} \quad (7)$$

朴素实现复杂度 $O(n \log^2 n)$ 。但注意到右式的单点点值可以 $O(\log n)$ 计算, 故可以以 $O(n \log n)$ 的复杂度计算出其在 $0, 1, \dots, n$ 处的点值然后通过卷积得到左侧多项式系数。复杂度优化为 $O(n \log n)$ 。

6.2 阶梯棋盘与斯特林数

定义 6.1 (阶梯棋盘). 对于 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n)$, 若对于所有 $1 \leq i \leq n$ 有 $a_i = i - 1$, 则称 S 为 n 阶阶梯棋盘, 记 S 为 St_n 。

图 8: St_6

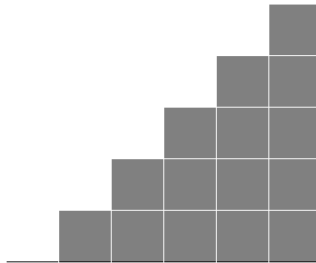


图 8 描述了 St_6 。对 St_n 应用定理 6.1 有

$$\sum_{k=0}^n r_k(St_n)x^{n-k} = x^n \quad (8)$$

有 $r_{n-k}(St_n) = \binom{n}{k}$, 即在 St_n 上放 $n - k$ 个车使得任意两个车不能互相攻击的方案数与将 n 个整数划分为 k 个无序集合的方案数一致。考虑在这两个问题之间建立一个双射。

对于集合划分 S_1, S_2, \dots, S_k , 将每个集合中的元素从小到大排序。设 $S_i = \{a_1, a_2, \dots, a_p\}$, 其中 $a_1 < a_2 < \dots < a_p$, 则在棋盘的 $(a_2, a_1), (a_3, a_2) \dots, (a_p, a_{p-1})$ 位置放车。这样得到的放车方案合法且放车数量为 $n - k$ 。

对于一个 $n - k$ 个车的合法放车方案, 若某个车放在 (p, q) 则将 p, q 所在的集合合并, 可以得到一个合法的集合划分。由于某个车放在 (p, q) 可推知 p 所在集合中比 p 小的最大元素恰好是 q , 因此若某个放车方案对应的划分中存在集合 $a_1 < a_2 < \cdots < a_p$ 则唯一对应的放车方案为 $(a_2, a_1), (a_3, a_2) \cdots, (a_p, a_{p-1})$ 。这样集合划分和放车方案的两个映射互为逆映射, 方案一一对应。

在另一类与棋盘相关的数上 n 阶阶梯棋盘与无符号第一类斯特林数也有关系。

定义 6.2. 设 $F_k(S)$ 表示在棋盘 S 上放置 k 个车使得没有两个车在同一列的所有方案构成的集合, $f_k(S) = |F_k(S)|$ 。

注意相比 $r_k(S)$, $f_k(S)$ 没有两个车不能在同一行的限制。有

定理 6.2. 对于轮廓线棋盘 $S = F(a_1, a_2, \cdots, a_n)$,

$$\sum_{k=0}^n f_{n-k}(S)x^k = \prod_{i=1}^n (x + a_i) \quad (9)$$

证明. 可以这样计算 $f_k(S)$: 选出放车的 k 列, 由于没有行限制因此第 i 列方案数是 a_i , 故 $f_k(S)$ 等于从 $\{a_1, a_2, \cdots, a_n\}$ 中选 k 个数相乘并求和, 即 $f_k(S) = [x^{n-k}] \prod_{i=1}^n (x + a_i)$ 。□

对 St_n 使用定理 6.2 有

$$\sum_{k=0}^n f_{n-k}(St_n)x^k = x^{\bar{n}} \quad (10)$$

此时 $f_{n-k}(St_n) = \begin{bmatrix} n \\ k \end{bmatrix}$, 即在 St_n 上放 $n - k$ 个车使得任意两个车不在同一列的方案数等于将 n 个数划分成 k 个圆排列的方案数。考虑在这两个问题之间建立一个双射。

对于某个划分为 k 个圆排列的方案, 对每个排列将最小的元素放在开头, 然后从 n 至 1 考虑。假设现在在考虑 i , 若其所在排列中仅有一个元素则删掉这个排列, 否则设其在该排列中恰好在它前面的元素为 v , 则在 (i, v) 放一个车并将 i 从该排列中删掉。这样得到的放车方案恰好放了 $n - k$ 个车, 满足每一列至多有一个车且在阶梯棋盘内。

以圆排列 (162)(35)(4) 举例说明操作过程:

1. 对于 6, 其前驱为 1, 因此加入车 (6, 1) 并删掉 6 得到 (12)(35)(4);
2. 对于 5, 其前驱为 3, 因此加入车 (5, 3) 并删掉 5 得到 (12)(3)(4);
3. 对于 4 和 3, 它们所在的排列的长度均为 1, 因此删掉它们, 剩余 (12);
4. 2 的前驱是 1, 故在 (2, 1) 上放车。最后 1 自成一个排列将其删掉, 过程结束。

故其对应的放车方案为 $\{(2, 1), (5, 3), (6, 1)\}$ 。

同时对于一个合法放车方案, 按列从小到大考虑, 若这一列没有车则新开一个以其为唯一元素的圆排列, 否则放在放车的行对应的数字后面, 可以得到 k 个圆排列的划分方案, 且两个映射互为逆映射。因此圆排列划分与不考虑行限制的放车方案是一一对应的。

对于两类斯特林数，有重要的斯特林反演公式：

$$\sum_{k=x}^y \begin{Bmatrix} y \\ k \end{Bmatrix} \begin{bmatrix} k \\ x \end{bmatrix} (-1)^{k-x} = [x=y] \quad (11)$$

上文给出了 $\begin{Bmatrix} n \\ k \end{Bmatrix}$ 和 $\begin{bmatrix} n \\ k \end{bmatrix}$ 在棋盘模型中的组合意义，那么我们可以考虑通过这样的组合意义证明斯特林反演公式。有

$$\begin{aligned} \sum_{k=x}^y \begin{Bmatrix} y \\ k \end{Bmatrix} \begin{bmatrix} k \\ x \end{bmatrix} (-1)^{k-x} &= \sum_{k=x}^y r_{y-k}(St_y) f_{k-x}(St_k) (-1)^{k-x} \\ &= \sum_{k=x}^y \sum_{P \in R_{y-k}(St_y)} \sum_{Q \in F_{k-x}(St_k)} (-1)^{|Q|} \end{aligned} \quad (12)$$

$x=y$ 时容易验证上式等于 1； $x \neq y$ 时，将所有 (k, P, Q) 三元组分成三类：

1. P 的第 y 列放了车；
2. P 的第 y 列没放车，但 Q 的第 k 列放了；
3. P 的第 y 列和 Q 的第 k 列都没放车。

考虑建立 1 类集合和 2 类集合的反号双射，即 1 类集合中的每个三元组与 2 类集合中的某个三元组一一对应，且它们对和式的贡献互为相反数。

具体地，对于某个属于 1 类集合的三元组 (k, P, Q) ，对棋盘 P, Q 进行如下操作：

1. 将 P 中第 y 列的车删除得到 P' ，并将 P' 上没有放车的行从下往上从 1 开始编号。记录 P 中第 y 列的车放在新编号下第 p 行。由于 P' 上有 $y-k-1$ 个车，故 $p \in [1, k]$ 。
2. 在 Q 中加入 $(k+1, p)$ 得到 Q' 。

则 (k, P, Q) 对应 $(k+1, P', Q')$ 。由于 $P' \in R_{y-k-1}(St_y)$, $Q' \in F_{k+1-x}(St_{k+1})$ ，且 P' 的第 y 列没有放车、 Q' 第 $k+1$ 列放了车，因此 $(k+1, P', Q')$ 属于 2 类集合。同时 $|Q'| - |Q| = 1$ 且该映射的逆映射容易构造，因此 1 类集合和 2 类集合间存在反号双射。

那么只需要计算 3 类集合的贡献，有

$$\begin{aligned} \sum_{k=x}^y \sum_{P \in R_{y-k}(St_y)} \sum_{Q \in F_{k-x}(St_k)} (-1)^{|Q|} &= \sum_{k=x}^y \sum_{P \in R_{y-k}(St_{y-1})} \sum_{Q \in F_{k-x}(St_{k-1})} (-1)^{|Q|} \\ &= \sum_{k=x-1}^{y-1} \sum_{P \in R_{(y-1)-k}(St_{y-1})} \sum_{Q \in F_{k-(x-1)}(St_k)} (-1)^{|Q|} \\ &= \sum_{k=x-1}^{y-1} \begin{Bmatrix} y-1 \\ k \end{Bmatrix} \begin{bmatrix} k \\ x-1 \end{bmatrix} (-1)^{k-(x-1)} \end{aligned} \quad (13)$$

即将 x, y 减去 1 答案不变。不断减直到 $x=0$ ，由于 $y > x$ 容易验证该式等于 0。那么通过棋盘的组合意义可以证明斯特林反演公式。

6.3 棋盘等价类

定义 6.3 (棋盘等价). 定义两个棋盘 S, T 等价当且仅当对于所有 $i \geq 0$, $r_i(S) = r_i(T)$ 。

对于 Ferrers 棋盘 $S = F(a_1, \dots, a_n)$, 由于在序列 $\{a_1, \dots, a_n\}$ 的开头加入若干个 0 或删除其开头的若干个 0 不会改变其 k -棋盘数, 故考虑调整其开头的 0 的数量, 即在棋盘前端插入或删除若干空列使得该序列的长度为 $|S| + 1$, 得到新的棋盘 $S' = F(a'_1, a'_2, \dots, a'_{|S|+1})$ 。此时定义 $B_S = \{b_{S,1} = a'_1, b_{S,2} = a'_2 - 1, \dots, b_{S,|S|+1} = a'_{|S|+1} - |S|\}$ 。注意序列 B_S 的长度总是 $|S| + 1$, 而与 S 的列数无关。有

定理 6.3 (Ferrers 棋盘的等价判定, [6]). 两个 Ferrers 棋盘 S, T 等价当且仅当 B_S 的元素构成的可重集与 B_T 的元素构成的可重集一致。

证明. 由定理 6.1, Ferrers 棋盘 $S = F(a_1, \dots, a_{|S|+1})$ 的下降幂棋盘多项式为 $\prod_{i=1}^{|S|+1} (x + a_i - i + 1)$ 。注意到 B_S 即 $\{a_i - i + 1 \mid 1 \leq i \leq |S| + 1\}$ 所构成的可重集, 故可以将棋盘 S 的下降幂棋盘多项式改写为 $\prod_i (x + i)^{cnt(B_S, i)}$, 其中 $cnt(B_S, i)$ 表示整数 i 在可重集 B_S 中的出现次数。

因此当 $B_S = B_T$ 时, 两个棋盘的下降幂棋盘多项式相等, 对比系数即得两个棋盘的 k -棋盘数相等; 同时由于将一个多项式分解为常数乘若干个一次项系数为一的一次式乘积的方式是唯一的, 故棋盘 S, T 的下降幂棋盘多项式相等可以推得 S, T 的下降幂棋盘多项式的一次式分解形式相等, 即 B 序列对应的可重集相等。□

由于棋盘等价具有传递性, 所以提出对棋盘等价类进行分析的想法是自然的。

首先对 Ferrers 棋盘 S 的序列 B_S 提出引理:

引理 6.1. 对于 Ferrers 棋盘 $S = F(a_1, \dots, a_{|S|+1})$ 和 $1 \leq i \leq |S| + 1$, $b_{S,i} \leq 0$ 。

证明. 假设存在 $1 \leq i \leq |S| + 1$ 使得 $b_{S,i} > 0$ 。此时有 $a_i \geq i$ 。由 Ferrers 棋盘的定義可得对于所有 $i \leq j \leq |S| + 1$, $a_j \geq a_i$, 故 $|S| = \sum_{k=1}^{|S|+1} a_k \geq \sum_{k=i}^{|S|+1} a_i \geq (|S| + 2 - i)i$ 。

而 $(|S| + 2 - i)i$ 是关于 i 的、二次项系数小于 0 的二次函数, 其在区间 $[1, |S| + 1]$ 的最小值在区间端点 1 或 $|S| + 1$ 处取到。而将 $i = 1$ 和 $i = |S| + 1$ 代入得到该式等于 $|S| + 1 > |S|$, 因此 $|S| \geq (|S| + 2 - i)i > |S|$, 产生矛盾。

故不存在 $b_{S,i} > 0$, 引理得证。□

由于 Ferrers 棋盘等价判定仅取决于序列 B 元素构成的可重集, 故对于 Ferrers 棋盘 S , 设 $k = -\min_{i=1}^{|S|+1} b_{S,i}$, 考虑将 B_S 中的元素构成的可重集表示为 $0^{c_0} (-1)^{c_1} (-2)^{c_2} \dots (-k)^{c_k}$ 的形式, 表示序列 B_S 中有 c_i 个 $-i$ 。值得注意的是, 由于对于所有 $2 \leq i \leq |S| + 1$, $b_{S,i} - b_{S,i-1} \geq -1$, 故对于所有 $0 \leq i \leq k$, c_i 不会等于 0。

定理 6.4 ([6]). 对于非空 Ferrers 棋盘 S , 恰好存在一个不存在空列的严格增 Ferrers 棋盘与 S 棋盘等价。

证明. 设 B_S 的元素构成的可重集为 $0^{c_0} \cdots (-k)^{c_k}$. 可以任意排列该可重集中的元素得到序列 B' , 只需 B' 满足 $b'_{i+1} - b'_i \geq -1$ 即对应一个 Ferrers 棋盘. 而让序列 B' 对应一个严格增 Ferrers 棋盘则需满足对于所有 $i \leq |S|$, $b'_i \neq -i + 1$ 等价于 $b'_i \leq b'_{i+1}$.

因此可以将 B' 分成前后两部分, 前一部分是从 0 到 $-k$ 的递减序列, 后一部分是从 $-k$ 到 0 的不降序列. 满足该条件的序列仅有一个, 而容易说明将该序列对应的棋盘的前面若干空列删掉之后得到一个严格增 Ferrers 棋盘. \square

定理 6.5 (棋盘等价类大小定理, [6]). 对于可重集 $T = 0^{c_0} \cdots (-k)^{c_k}$, 满足 B_S 的元素构成的可重集为可重集 T 的 Ferrers 棋盘 S 的数量为

$$\prod_{i=0}^{k-1} \binom{c_i + c_{i+1} - 1}{c_{i+1}} \tag{14}$$

证明. 考虑构造所有可能的序列 B_S . 先在序列中加入所有 0, 然后依次加入 $-1, -2, \dots, -k$.

加入 $-i$ 时, 由于序列相邻两项后一项减前一项不能小于 -1 且第一项必须为 0, 因此 $-i$ 的前驱一定是 $-i$ 或 $-i + 1$, 即有 c_{i-1} 个位置可将任意多 $-i$ 放在其后面. 使用隔板法, 放入 $-i$ 的方案数为 $\binom{c_{i-1} + c_i - 1}{c_i}$. 由于每个数的方案独立, 与之前的数在序列中的排列无关, 故总方案为每个数的方案的乘积, 即为式 (14). \square

6.4 分解定理的变体

这个部分将会定义两个与 k -棋盘数类似的量, 并研究在 Ferrers 棋盘上新的量对应的棋盘多项式的性质.

定义 6.4 (m 阶 k -棋盘数). 对于棋盘 S 和正整数 m , 自下而上将每 m 行分成一组, 则设棋盘 S 的 m 阶 k -棋盘数 $r_{m,k}(S)$ 表示在棋盘 S 上放 k 个车使得不存在两个车在同一个行组或同一列的方案数.

特别地 $r_{1,k}(S) = r_k(S)$. $m = 2$ 时, 对于图 9 中的两个放置, 左图放置合法而右图放置不合法, 因为右图两个车分别在 1, 2 行, 它们在同一个行组里. 而在 $m = 1$ 的情况下, 两个放置方案均是合法的.

图 9: $m = 2$ 时, 左图棋盘放置是合法的, 而右图是不合法的.



对于 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n)$, 将所有非空列按高度分成若干集合 S_1, S_2, \dots , 其中 $i \in S_p$ 等价于 $a_i \in [pm + 1, (p + 1)m]$, 也就是按照每一列能够放置的最高的行组进行分类。设 $\rho_m(S_i) = \sum_{p \in S} a_p - im$ 。

由于序列 a 单调不降, 因此非空的 S_i 会包含一段连续的整数, 且随着下标 i 的增加包含的列编号越来越大。

定理 6.6 (m 阶下降幂棋盘多项式分解定理, [1]). 对于 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n)$ 和正整数 m , 有

$$\sum_{k=0}^n r_{m,k}(S)x^{n-k,m} = \prod_{i=1}^n (x - (i-1)m + v_i) \quad (15)$$

其中若 $a_i = 0$ 则 $v_i = 0$, 否则设 $i \in S_p$, 若第 i 列是 S_p 的最后一列则 $v_i = pm + \rho_m(S_p)$, 否则 $v_i = pm$ 。

证明. 由于式 (15) 两边是关于 x 的多项式, 因此只需证明式 (15) 在 x 取无穷多个点处相等即可证明这两个多项式相等。考虑在 $m \mid x$ 的点 x 处证明之。

与图 6 一致地在 S 下面插入额外 x 行得到 S_x 。 S_x 的行组划分仅需在 S 的行组划分的基础上将额外加入的 x 行按顺序分成 $\frac{x}{m}$ 组。用两种方式计算 $r_{m,n}(S_x)$:

1. 枚举 S 的合法子集 T 并计算与 S 的交集为 T 的合法方案数。

设 $|T| = k$, 则满足条件的子集 T 的数量为 $r_{m,k}(S)$ 。 剩余 $n - k$ 列的方案可以一列列考虑, 类似定理 6.1 的证明, 得到其贡献为 $x^{n-k,m}$, 故

$$r_{m,n}(S_x) = \sum_{k=0}^n r_{m,k}(S)x^{n-k,m}$$

2. 从左到右考虑每个列集合的放车位置。

考虑 $S_p = [s, t]$ 时, 先放 s 至 $t - 1$ 列的车, 且仅考虑这 $t - s$ 列的车放在编号 $\leq pm$ 的行或额外加入的 x 行的情况。 对于 $s \leq j < t$, 第 j 列共有 $p + \frac{x}{m}$ 组可以放车, 其中有 $j - 1$ 个组已经放了车, 故方案数为 $pm + x - (j - 1)m$ 。

最后考虑第 t 列的放车时, 将 S_p 中所有的放车方案分为是否有车放在行组 $[pm + 1, (p + 1)m]$ 中的两种情况。

对于不放的情况, 只需在 s 至 $t - 1$ 列的选择上在第 t 列选择不冲突的 $\leq pm$ 的行或额外加入的 x 行, 方案数为 $pm + x - (t - 1)m$;

对于放的情况, 在 s 至 $t - 1$ 列的选择上, 选择一个位于 $[pm + 1, (p + 1)m]$ 行组且位于列集合 S_p 的位置放车。 若这个放车的位置不在第 t 列, 则将其所在列之前放入的车移动至第 t 列, 得到有车放在 $[pm + 1, (p + 1)m]$ 行组的合法方案。

容易证明可以对上述变换构建逆变换, 即二元组 (s 至 $t-1$ 列的选择方案, S_p 列集合、 $[pm+1, (p+1)m]$ 行组的选择方案) 与 S_p 中有车放在 $[pm+1, (p+1)m]$ 行组的方案一一对应。此时第 t 列选择的方案数为 $\rho_m(S_p)$ 。

故第 t 列的方案数为 $x + pm - (t-1)m + \rho_m(S_p)$ 。

任意两列的方案数互不影响, 故

$$r_{m,n}(S_x) = \prod_{i=1}^n (x - (i-1)m + v_i)$$

合并两式即得式 (15)。 □

定义 6.5 (α 权 k -棋盘数). 对于棋盘 S 和实数 α , 定义棋盘 S 的 α 权 k -棋盘数 $v_{\alpha,k}(S)$ 表示所有在棋盘 S 上放 k 个车使得不存在两个车在同一列的方案权值和。一个放车方案的权值等于其所有行的权值乘积, 一行的权值如下定义:

1. 若该行上放了不超过一个车, 则权值为 1;
2. 否则设其放了 p 个车, 其权值为 $\alpha(2\alpha-1)(3\alpha-2)\cdots(p\alpha-(p-1))$ 。

可以注意到的是 $v_{0,k}(S) = r_k(S), v_{1,k}(S) = f_k(S)$ 。

定理 6.7 (α 权下降幂棋盘多项式分解定理, [5]). 对于 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n)$ 和实数 α , 有

$$\sum_{k=0}^n v_{\alpha,k}(S) x^{n-k, 1-\alpha} = \prod_{i=1}^n (x + a_i - (i-1)(1-\alpha)) \tag{16}$$

证明. 将式 (16) 两侧看成关于 α 的 n 次多项式, 只需证明对于无穷多个 α 它们相等即可证明式 (16)。考虑在 $\alpha \in \mathbb{N}$ 处证明。

考察新问题: 给定棋盘 S , 从左至右考虑每一列是否放车, 若放车则在放车的行上方插入 α 个空行, 后面的列决策时可以选择这些新插入的空行, 同时在新插入的空行上放车也会插入空行。定义 $v'_{\alpha,i}(S)$ 表示新问题中放 i 个车的方案数, 考虑说明 $v'_{\alpha,i}(S) = v_{\alpha,i}(S)$ 。

建立从新问题方案到原问题方案的映射: 对于一个新问题方案, 设某个车在第 i 列, 放在棋盘 S 的第 j 行或由其直接或间接插入得到的行, 则在原问题棋盘中 (i, j) 位置放车。这样一个新问题解对应一个原问题解, 而对于一个原问题解, 某行有 $p > 1$ 个车时, 从左往右考虑, 第一个车方案数为 1、第二个车方案数为 α 、第三个车方案数为 $2\alpha-1\cdots\cdots$ 总方案数为 $\alpha(2\alpha-1)\cdots(p\alpha-(p-1))$, 与该行权值相等。故原问题的方案对应其权值个新问题的方案。因此原问题的方案权值和等于新问题的方案数, 即 $v'_{\alpha,i}(S) = v_{\alpha,i}(S)$ 。

转化为计算新问题之后证明过程与定理 6.1 基本一致, 在此不做赘述。 □

例题 6.2 (小 Q 的序列⁹). 给定长度为 n 的序列 a_0, a_1, \dots, a_{n-1} 和一个整数 α , 定义一个长度为 k 的序列 p_0, p_1, \dots, p_{k-1} 的权值为 $\prod_{i=0}^{k-1} (p_i + \alpha i)$. 对于每个 $0 \leq k \leq n$ 求序列 $\{a_i\}$ 的所有长度为 k 的子序列的权值和, 对 998244353 取模. $1 \leq n \leq 10^5$.

该问题的经典解法是求解微分方程. 接下来我们通过棋盘理论得到一个组方法.

当序列 $\{a_i\}$ 单调不降的时候, 所求解的正是 $F(a_0, \dots, a_{n-1})$ 的 $(\alpha+1)$ 权棋盘多项式. 考虑找到一个单调不降序列 $\{a'_i\}$ 使得两者的答案相等.

在模 $P = 998244353$ 意义下任选一个位置 i 进行操作 $a_i \leftarrow a_i + P$ 不会影响答案, 因此可以进行若干次该操作直到序列 $\{a_i\}$ 单调不降. 此时可以运用定理 6.7. 与此同时定理 6.7 右式中进行 $a_i \leftarrow a_i - P$ 不会影响答案, 故设长度为 k 的子序列权值和为 ans_k , 则

$$\sum_{k=0}^n ans_k x^{n-k-\alpha} \equiv \prod_{i=1}^n (x + a_i + (i-1)\alpha) \pmod{P}$$

由于该等式并不依赖于 P 的取值, 所以在不取模的意义下两者仍然是相等的.

特殊处理 $\alpha = 0$ 的情况, 对于其他情况代入 $x = -\alpha y$ 并化简, 有

$$\sum_{k=0}^n ans_k (-\alpha)^{n-k} y^{n-k} = \prod_{i=1}^n (-\alpha y + a_i + (i-1)\alpha)$$

仍可以使用传统方法计算 ans_k 的取值. 复杂度 $O(n \log^2 n)$.

7 排列降低计数

定义 7.1 (降低). 对于排列 $\pi = \pi_1 \pi_2 \dots \pi_n$, 定义

$$Des_\pi = \{1 \leq i < n \mid \pi_i > \pi_{i+1}\}, des_\pi = |Des_\pi|$$

$$Des_{\pi,k} = \{1 \leq i < n \mid \pi_i - \pi_{i+1} = k\}, des_{\pi,k} = |Des_{\pi,k}| (k > 0)$$

称 Des_π 为排列 π 的降低集合, 若 $x \in Des_\pi$ 则称 (π_x, π_{x+1}) 为一个降低对, des_π 称为 π 的降低; $Des_{\pi,k}$ 为排列 π 的 k -降低集合, 若 $x \in Des_{\pi,k}$ 则称 (π_x, π_{x+1}) 为一个 k -降低对, $des_{\pi,k}$ 称为 π 的 k -降低.

定义 7.2 (超过数). 对于排列 $\pi = \pi_1 \pi_2 \dots \pi_n$, 定义

$$Exc_\pi = \{1 \leq i \leq n \mid \pi_i > i\}, exc_\pi = |Exc_\pi|$$

$$Exc_{\pi,k} = \{1 \leq i \leq n \mid \pi_i - i = k\}, exc_{\pi,k} = |Exc_{\pi,k}| (k > 0)$$

称 Exc_π 为排列 π 的超过集合, 若 $x \in Exc_\pi$ 则称 (π_x, x) 为一个超过对, exc_π 称为 π 的超过数; $Exc_{\pi,k}$ 为排列 π 的 k -超过集合, 若 $x \in Exc_{\pi,k}$ 则称 (π_x, x) 为一个 k -超过对, $exc_{\pi,k}$ 称为 π 的 k -超过数.

⁹来源: LibreOJ 6703, 有改动

我们可以在 $\text{Sym}(n)$ 与 $R_n(B_n)$ 之间建立双射：对于排列 π 在所有 $(i, \pi_i), 1 \leq i \leq n$ 的位置放车。在这样的描述方式下，降低由于涉及相邻两个车的位置关系描述较困难，而超过数仅涉及一个车的行列关系更易描述。下面将要介绍的对排列的变换将在降低和超过数之间建立联系。

7.1 Foata 第一基本变换

定义 7.3 (Foata 第一基本变换). 对于排列 π ，对其进行 **Foata 第一基本变换**即依次进行以下操作得到另一个排列 π' ：

1. 将排列写成轮换形式，即写成 $(a_{1,1}, \dots, a_{1,k_1})(a_{2,1}, \dots, a_{2,k_2}) \dots$ 的形式满足 $\pi_{a_{i,j}} = a_{i,(j \bmod k_i)+1}$ 。
2. 将每个轮换中的最大值放到轮换末尾，然后按照轮换中的最大值将轮换从小到大排序；
3. 将每个轮换看成序列，翻转之后拼接起来得到 π' 。

以 $\pi = 61437258$ 举例说明。其轮换表示为 $(162)(34)(57)(8)$ 。将每个轮换的最大值放在轮换末尾得到 $(216)(34)(57)(8)$ ，再按照轮换最大值从小到大排序得到 $(34)(216)(57)(8)$ 。最后将每个轮换看做序列翻转后拼接得到 $\pi' = 43612758$ 。

可以通过建立逆变换的方式证明任意两个不同的排列经过该变换不会得到相同的排列：对于排列 π' ，按照前缀最大值将其分段，对于每一段翻转之后看做轮换形式最后还原排列。

如 $\pi' = 43612758$ ，其中 4, 6, 7, 8 是前缀最大值，故将其划分为 $[43][612][75][8]$ ，将每个段翻转之后看做轮换形式得到 $(34)(216)(57)(8)$ ，对应排列 $\pi = 61437258$ 。

不难证明这两个变换互为逆变换，因此这两个变换在 $\text{Sym}(n)$ 与 $\text{Sym}(n)$ 之间建立了双射。注意到 π 中 (j, i) 是一个 k -超过对当且仅当 π' 中 (j, i) 是一个 k -降低对，因此有

定理 7.1.

$$\sum_{\pi \in \text{Sym}(n)} x^{\text{des}_\pi} = \sum_{\pi \in \text{Sym}(n)} x^{\text{exc}_\pi} \tag{17}$$

$$\forall k > 0, \sum_{\pi \in \text{Sym}(n)} x^{\text{des}_{\pi,k}} = \sum_{\pi \in \text{Sym}(n)} x^{\text{exc}_{\pi,k}} \tag{18}$$

即在一定情况下，降低计数问题和超过数计数问题可以相互转化。由于超过数在棋盘模型中较容易描述，可使用棋盘模型计算超过数的分布，进而得到降低的分布。

例题 7.1. 给定 n 和 m 个二元组限制，每个二元组限制 (i, j) 均满足 $1 \leq i < j \leq n$ ，表示在排列中 i, j 相邻且 i 在 j 的前面。对 $k \in [m, n - 1]$ 求满足所有二元组限制的 n 阶排列中

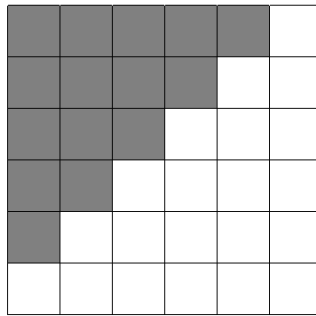
升高为 k 的排列数量, 对 998244353 取模。 $n \leq 10^6, n - m \leq 10^5$ 。定义 n 阶排列 π 的升高为 $\sum_{i=1}^{n-1} [\pi_i < \pi_{i+1}]$ 。¹⁰

由于上文中提到的是降低和超过数的关系, 所以先将计算对象转化为降低: 将 π 中的所有数 x 变为 $n - x + 1$, 然后将所有限制条件 (i, j) 变为 $(n - i + 1, n - j + 1)$ 。那么每一个限制 (p, q) 都有 $p > q$, 需要计算满足所有二元组限制的排列的降低分布。

考察 Foata 第一基本变换的逆变换过程考虑将问题转化为超过数的计算。由于二元组 (p, q) 有 $p > q$, 因此 q 一定不是前缀最大值, 在逆变换的分段过程中 p 和 q 一定在一段, 即逆变换得到的排列 π 一定有 $\pi_q = p$ 。除此以外没有其他限制条件。

故问题转化为给出若干形如 $\pi_p = q$ 的条件, 求满足条件的排列的超过数的分布。对于没有限制的情况, 由于排列与棋盘放车有一一对应, 考虑构建棋盘模型: 如图 10 所示, 考虑类似 S_n 的倒三角形式棋盘, 其中黑色格子构成棋盘 S_n , 则容易知道 $h_{k,n}(S_n)$ 为超过数为 k 的排列数。

图 10: $n = 6$ 时的倒三角棋盘。



对于有限制的情况, 在 S_n 基础上对于每个 $\pi_p = q$ 的条件将第 p 列和第 q 行删除, 完成后将没有被删除的行列按顺序进行重标号。则对于结果棋盘上的任一行, 在棋盘 S 中的二元组的行编号构成一段右端点为 $n - m$ 的区间, 且左端点随着列的增加单调不降。求出每一列的区间长度可以从后往前扫描, 复杂度 $O(m)$ 。

将棋盘旋转不会影响其棋盘多项式, 将其旋转 180° 之后得到一个 Ferrers 棋盘。使用定理 6.1 计算其棋盘多项式, 再使用定理 4.1 得到 $h_{k,n}(S)$ 。复杂度 $O(m + (n - m) \log^2(n - m))$ 。

例题中描述了如何使用倒三角棋盘计算满足条件的排列中超过数的分布。我们可以对例题中没有限制的特殊情况进行额外的讨论。

此时需要对每个 k 求 $\sum_{\pi \in \text{Sym}_n} [exc_\pi = k]$ 。由于 $\sum_{\pi \in \text{Sym}_n} [des_\pi = k] = \binom{n}{k}$, 故求的就是欧拉数。与此同时图 10 是求解 $\sum_{\pi \in \text{Sym}_n} [exc_\pi = k]$ 的倒三角棋盘 S_n , 有 $\binom{n}{k} = h_{k,n}(S_n)$ 。将 S_n 旋转

¹⁰来源: 原创

180° 之后得到 S_t_n ，而上文讨论了 $r_k(S_t_n) = \left\{ \begin{matrix} n \\ n-k \end{matrix} \right\}$ 。使用定理 4.1，有

$$\left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle = \sum_{i=k}^n (-1)^{i-k} \binom{i}{k} \left\{ \begin{matrix} n \\ n-i \end{matrix} \right\} (n-i)!$$

这样我们得到了欧拉数与第二类斯特林数之间的一个经典等式。

7.2 k-超过数计数

设 $P_{n,k,p} = \sum_{\pi \in S_n} [exc_{\pi,k} = p]$ 。 $P_{n,k,p}$ 相比超过数的分布计算简单一些，因为其棋盘的形式更简单：设棋盘 $S_{n,k} = \{(1, k+1), (2, k+2), \dots, (n-k, n)\}$ ，则 $P_{n,k,p} = h_{p,n}(S_{n,k})$ 。 $S_{n,k}$ 满足不存在两个格子在同行或同列，因此 $r_i(S_{n,k}) = \binom{n-k}{i}$ 。故

定理 7.2.

$$P_{n,k,p} = \sum_{i=p}^{n-k} (-1)^{i-p} \binom{i}{p} \binom{n-k}{i} (n-i)! \quad (19)$$

$P_{n,k,p}$ 还有一些有趣的性质：

定理 7.3. 对于 $0 \leq k < n$ ， $p \geq 1$ ，有

$$P_{n,k,p} = P_{n-1,k,p-1} + (p+1)P_{n-1,k,p+1} + (n-p-1)P_{n-1,k,p} \quad (20)$$

证明. 考虑在 $\pi \in \text{Sym}(n-1)$ 上按如下方式加入 n 得到 n 阶排列：先将 n 放在 π 最后，然后选择一个位置 $i \in [1, n]$ ，若 $i \neq n$ 则交换 π_i 和 n 得到排列 π' 。考虑该过程的三种结果：

1. $\pi'_{n-k} = n$ ，即 $exc_{\pi',k} = exc_{\pi,k} + 1$ 。此时 $exc_{\pi,k} = p-1$ ，且仅有 $i = n-k$ 可以达到该目标，方案数为 $P_{n-1,k,p-1}$ 。
2. $\pi'_n = i+k$ ，即 $exc_{\pi',k} = exc_{\pi,k} - 1$ 。此时 $exc_{\pi,k} = p+1$ ，且这 $p+1$ 个位置作为 i 都合法，方案数为 $(p+1)P_{n-1,k,p+1}$ ；
3. 两种情况都没有发生，即 $exc_{\pi',k} = exc_{\pi,k}$ 。此时 $exc_{\pi,k} = p$ ，且 $n-1-p$ 个没有贡献且不是 $n-k$ 的位置作为 i 均合法，方案数为 $(n-p-1)P_{n-1,k,p}$ 。

合并三者即可得到式 (20)。 □

作为定理 7.3 在 $p = 0$ 处的补充，有

定理 7.4. 对于 $0 \leq k < n$ ，

$$P_{n,k,0} = (n-1)P_{n-1,k,0} + (n-k-1)P_{n-2,k,0} \quad (21)$$

证明. 沿用定理 7.3 的增量计算方式。考虑两种 $exc_{\pi,k}$ 的可能：

1. $exc_{\pi,k} = 0$, 此时只需要 $i \neq n - k$ 就合法, 贡献为 $(n - 1)P_{n-1,k,0}$;
2. $exc_{\pi,k} = 1$, 此时仅有一个 i 合法。对于这样的排列 π , 在棋盘 $S_{n-1,k}$ 上将命中的格子所在行列删掉然后重标号行列, 该方案对 $P_{n-2,k,0}$ 产生贡献。同时对于 $P_{n-2,k,0}$ 的每个方案, 都有 $n-1-k$ 种方式加入被删掉的位置使得 $exc_{\pi,k} = 1$ 。故方案数为 $(n-k-1)P_{n-2,k,0}$ 。

将两者合并即可得到式 (21)。 □

定理 7.5. 对于 $0 \leq k < n$, 有

$$P_{n,k,0} = k! \sum_{r=0}^k \binom{k}{r} \binom{n-k}{k-r} P_{n-k,k-r,0} \quad (22)$$

证明. 使用棋盘理论解决。由于 $S_{n,k}$ 仅在前 $n - k$ 列存在元素, 因此考虑枚举最后 k 列的情况。

枚举最后 k 列有 r 个车放在 $1 \sim k$ 行, 删掉它们所在的行列时不会删掉 $S_{n,k}$ 中的格子, 而剩余 $k - r$ 个车删除行列时会删掉一个格子。选择不删的 r 行的方案数为 $\binom{k}{r}$; 选择删的 $k - r$ 行的方案数为 $\binom{n-k}{k-r}$; 给最后 k 列分配行的方案数为 $k!$ 。

删掉这 k 行 k 列后重编号行列, 此时还有 $n - k$ 列没有确定, 新的限制棋盘 S' 满足 $|S'| = n - r$, 由于 S' 仍满足任意两个格子不在同行或同列, 可以通过交换行列使得 $S' = S_{n-k,k-r}$, 故剩余的方案数为 $P_{n-k,k-r,0}$ 。

对于所有 r 将每一步的方案数相乘并求和, 可以得到式 (22)。 □

定理 7.6. 对于 $s \leq k < n$, 有

$$P_{n,k,s} = \binom{n-k}{s} P_{n-s,k,0} \quad (23)$$

证明. 这个的证明相对简单: 恰好有 s 个命中棋盘 $S_{n,k}$, 枚举这 s 个位置, 有 $\binom{n-k}{s}$ 种情况; 对于每种情况将 s 个车所在的行列删掉, 剩下 $n - k$ 列要选择, 有 $n - s - k$ 个限制的位置, 因此删掉行列之后的方案数就是 $P_{n-s,k,0}$ 。 □

7.3 X-Y 降低计数

定义 7.4. 给定数集 X, Y , 对于 n 阶排列 π , 设

$$Des_{\pi,X,Y} = \{1 \leq i < n \mid \pi_i > \pi_{i+1}, \pi_i \in X, \pi_{i+1} \in Y\}, des_{\pi,X,Y} = |Des_{\pi,X,Y}|$$

$$Exc_{\pi,X,Y} = \{1 \leq i < n \mid \pi_i > i, \pi_i \in X, i \in Y\}, exc_{\pi,X,Y} = |Exc_{\pi,X,Y}|$$

称 $Des_{\pi,X,Y}$ 为排列 π 的 $X - Y$ 降低集合, 若 $x \in Des_{\pi,X,Y}$ 则称 (π_x, π_{x+1}) 为一个 $X - Y$ 降低对, $des_{\pi,X,Y}$ 称为 π 的 $X - Y$ 降低; 称 $Exc_{\pi,X,Y}$ 为排列 π 的 $X - Y$ 超过集合, 若 $x \in Exc_{\pi,X,Y}$ 则称 (π_x, x) 为一个 $X - Y$ 超过对, $exc_{\pi,X,Y}$ 称为 π 的 $X - Y$ 超过数。

例如 $X = \{2, 3, 5\}, Y = \{1, 3, 4\}$ 时, 排列 53142 的 $X - Y$ 降低对有 $(5, 3)$ 和 $(3, 1)$ 。注意 $(4, 2)$ 虽然是降低对, 但 $4 \notin X$ 所以其不是 $X - Y$ 降低对。

通过 Foata 第一基本变换同样可以得到 $des_{\pi, X, Y}$ 和 $exc_{\pi, X, Y}$ 的双射关系:

定理 7.7. 对于 $X, Y \subseteq [1, n] \cap \mathbb{Z}$, 有

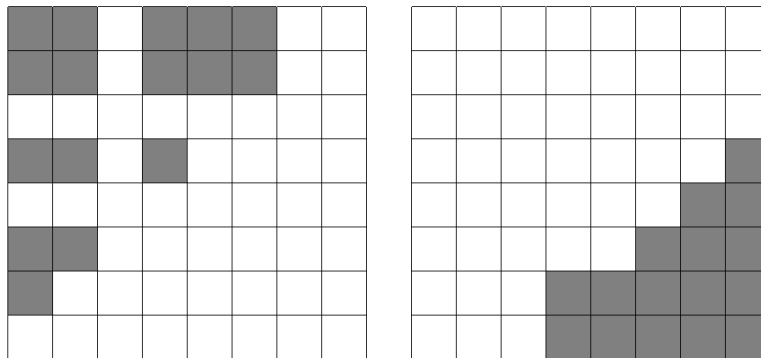
$$\sum_{\pi \in \text{Sym}_n} x^{des_{\pi, X, Y}} = \sum_{\pi \in \text{Sym}_n} x^{exc_{\pi, X, Y}} \quad (24)$$

计算 $exc_{\pi, X, Y}$ 仍然可以使用棋盘模型: 设 $S_{X, Y} = \{(i, j) \mid i \in Y, j \in X, i < j\}$, 则

$$\sum_{\pi \in \text{Sym}_n} [exc_{\pi, X, Y} = p] = h_{p, n}(S_{X, Y})$$

图 11 的左侧棋盘展示了 $X = \{2, 3, 5, 7, 8\}, Y = \{1, 2, 4, 5, 6\}$ 时的棋盘 $S_{X, Y}$ 。

图 11: $X = \{2, 3, 5, 7, 8\}, Y = \{1, 2, 4, 5, 6\}$ 时的棋盘 $S_{X, Y}$ 和交换行列后得到的 Ferrers 棋盘 $F(0, 0, 0, 2, 2, 3, 4, 5)$ 。



对于这样的棋盘, 一个特殊的性质是按照格子个数从小到大排序, 可以得到一个 Ferrers 棋盘, 如图 11 的右侧棋盘所示。因此我们仍然可以使用定理 6.1 和定理 4.1 来计算对于给定的 s , $\sum_{\pi \in \text{Sym}_n} [des_{\pi, X, Y} = s]$ 的值。

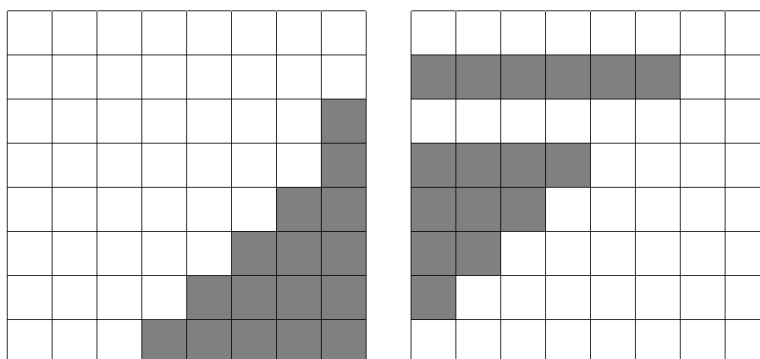
由定理 6.4, 每个 Ferrers 棋盘恰有一个无视空列情况下的严格增 Ferrers 棋盘与其对应, 而当该严格增棋盘的列高为 $a_1 < a_2 < \dots < a_p$ 时, 可以通过翻转并交换行列得到该棋盘等价于 $S_{\{a_1+1, a_2+1, \dots, a_p+1\}, \mathbb{N}}$ 。故有

定理 7.8. 对于任意数集 X, Y 均存在恰好一个数集 X' 使得

$$\sum_{\pi \in \text{Sym}_n} x^{des_{\pi, X, Y}} = \sum_{\pi \in \text{Sym}_n} x^{des_{\pi, X', \mathbb{N}}} \quad (25)$$

对于图 11 中的例子, 通过定理 6.4 的构造证明, 可以得到与 $F(2, 3, 4, 5, 5)$ 等价的棋盘是 $F(1, 2, 3, 4, 6)$ 。图 12 中给出了图 11 对应棋盘的等价严格增 Ferrers 棋盘和其翻转、交换行列之后得到的棋盘 $S_{\{2, 3, 4, 5, 7\}, \mathbb{N}}$ 。

图 12: 等价的严格增 Ferrers 棋盘和 $S_{\{2,3,4,5,7\},\mathbb{N}}$



最后, 在参考文献 [9] 中给出了计算 $\sum_{\pi \in \text{Sym}_n} [des_{\pi, X, Y} = s]$ 的公式, 但由于其证明没有使用棋盘理论, 故仅在此提出不做过多说明。感兴趣的读者可自行阅读参考文献 [9]。

8 Ferrers 棋盘的排列逆序对计数

在 Atcoder Grand Contest 中已经出现了轮廓线棋盘排列逆序对个数和的问题¹¹。其给出的标准算法不基于棋盘理论, 故在本文中不进行阐述, 但由于该算法需要使用到交换求和顺序的方式, 故拓展性不强。下面介绍的 q -analogue 理论虽然只能在 Ferrers 棋盘上进行计算, 但其基于对每个排列进行逆序对数的带权, 具有更强的拓展性。

8.1 k -棋盘数的 q -analogue 形式

q -analogue 理论是一类公式的拓展方式。它需要我们引进一些新的符号。

对于 $n \in \mathbb{R}$, 定义

$$[n]_q = \frac{1 - q^n}{1 - q}$$

其中 q 为形式元。特别地, 在 $n \in \mathbb{N}$ 时, 可以发现 $[n]_q = 1 + q + \dots + q^{n-1}$ 。

在后面的 q -analogue 等式中, 可以将 q 代入任一使得和式收敛的常数。特别地, 约定代入 $q = 1$ 时, 对于任意 $n \in \mathbb{R}$, $[n]_1 = n$ 。同时定义 q -analogue 下降幂

$$[n]_q^k = [n]_q [n-1]_q \cdots [n-k+1]_q$$

定义 8.1 (棋盘放车方案的权值¹²). 对于 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n) \subseteq B_n$ 和任一合法放车方案 $T \subseteq S$, 对于 T 中的每一个车 (i, j) , 在棋盘 S 中对所有满足 $i < k \leq n$ 的位置

¹¹Atcoder Grand Contest 023 E. Inversion

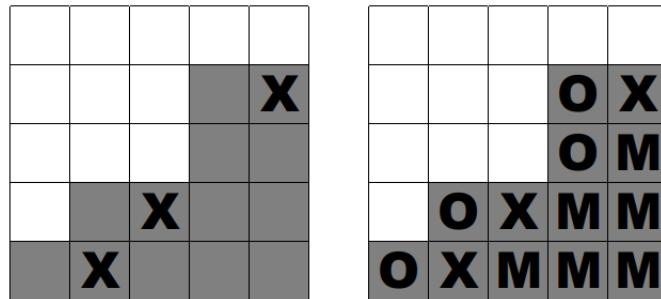
¹²大部分英文文献将其定义为逆序对数 inv , 但其与对应排列的逆序对数并不相同, 为了不产生混淆这里使用权值代替。

(k, j) 和满足 $1 \leq k < j$ 的位置 (i, k) 打上标记。设 $C_S(T)$ 为做完上述操作后 S 中没有被打标记且没有放车的格子数量，则定义 T 在棋盘 S 中的权值 $val_S(T) = |C_S(T)|$ 。

需要注意的是这里定义的权值与定义 6.5 中定义的权值不同。

如图 13 所示，左侧棋盘为棋盘 $S = F(1, 2, 2, 4, 4)$ 的一个放车的方案 $T = \{(2, 1), (3, 2), (5, 4)\}$ ，而右图展示了对其进行标记操作之后对应的棋盘，其中 M 表示被标记，O 表示既不被标记也没有放车。图上共有 4 个 O 因此 $val_S(T) = 4$ 。

图 13: 一个放车方案和其对应的标记和权值



定义 8.2 (正序对, 逆序对). 对于排列 $\pi \in \text{Sym}_n$, 定义

$$\text{coinv}_\pi = \{(i, j) \mid i < j, \pi_i < \pi_j\}, \text{inv}_\pi = \{(i, j) \mid i < j, \pi_i > \pi_j\}$$

分别表示排列 π 的正序对数和逆序对数。

对于排列 π 和其对应的棋盘放置方案 T , 若 $T \subseteq S$, 可以找到 $val_S(T)$ 与 $\text{coinv}(\pi)$ 之间的关系:

定理 8.1. 对于 n 阶排列 π 和 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n) \subseteq B_n$, 若 $T = \{(i, \pi_i) \mid 1 \leq i \leq n\} \subseteq S$, 则

$$val_S(T) + \sum_{i=1}^n n - a_i = \text{coinv}(\pi) \tag{26}$$

证明. 由于棋盘 S 是 Ferrers 棋盘, 所以棋盘 S 外的格子一定不会被标记, 所以 $val_S(T) + \sum_{i=1}^n n - a_i$ 表示的就是 B_n 中没有被标记的格子数量。

对于 B_n 中一个没有被标记且没有放车的格子 (i, j) , 由于 $|T| = n$, 故此时有 $\pi_i < j$, $\pi_j^{-1} > i$, 其中 π_j^{-1} 表示 j 在排列 π 中的位置。那么 (i, π_j^{-1}) 此时形成了一个正序对。与此同时一个正序对 (i, j) 也会对应一个没有被标记且没有放车的格子 (i, π_j) , 因此正序对集合与没有被标记的格子的集合一一对应, 故有式 (26)。□

同时由于有 $\text{coinv}(\pi) + \text{inv}(\pi) = \binom{n}{2}$, 因此 $\text{inv}(\pi)$ 与 $val_S(T)$ 的关系是简单的。故通过对 k -棋盘数的计算带权来做排列逆序对计数问题是很有发挥空间的。

定义 8.3 (k -棋盘数的 q -analogue 形式). 对于 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n) \subseteq B_n$, 定义 S 的 q -analogue k -棋盘数

$$[r_k(S)]_q = \sum_{T \in R_k(S)} q^{val_S(T)} \quad (27)$$

当 $q = 1$ 时, 其定义与 k -棋盘数 $r_k(S)$ 一致。这也是 q -analogue 理论在进行拓展的时候始终满足的一个特性: 当 $q = 1$ 时可以得到未进行拓展时就已经得出的定义和结论。

8.2 q -analogue 形式下的分解定理

对定理 6.1 在 q -analogue 形式下进行拓展。

定理 8.2 (q -analogue 下降幂棋盘多项式分解定理, [10]). 对于 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n) \subseteq B_n$, 有

$$\sum_{k=0}^n [r_k(S)]_q [x]_q^{n-k} = \prod_{i=1}^n [x + a_i - i + 1]_q \quad (28)$$

证明. 将式 (28) 左右看做关于 x 的 n 次多项式, 因此只需要在无限多个点 x 上证明式 (28) 成立即可证明式 (28) 恒成立。考虑在 $x \in \mathbb{N}$ 上证明。

如图 6 在棋盘 S 下面加上 x 行得到棋盘 S_x 。使用两种方式计算 $[r_n(S_x)]_q$:

1. 枚举 S 的合法子集 T , 计算所有合法方案中与 S 交集为 T 的方案权值和。

设 $|T| = k$, 由于额外加入的 x 行不会影响到棋盘 S 中的标记, 故 S 上没有被标记且没有放车的格子数量是 $val_S(T)$ 。

剩余 $n - k$ 列从左往右依次考虑。第一列有 x 个位置可以放车, 而决定了放车位置后在这 x 个位置中恰好在其上方的所有格子可以直接贡献到权值。因此从上往下在这些格子上放车权值会增加 $0, 1, 2, \dots, x - 1$, 因此第一列贡献是 $1 + q + \dots + q^{x-1} = [x]_q$ 。第二列有 $x - 1$ 个位置可以放车, 其中从上往下考虑在这些格子上放车权值会增加 $0, 1, 2, \dots, x - 2$, 因此第二列的贡献是 $[x - 1]_q$ 。

类似考虑, 第 i 列的贡献为 $[x - i + 1]_q$ 。故子集 T 的贡献为 $q^{val_S(T)} [x]_q^{n-k}$ 。对于 $|T| = k$ 的所有 T 求和, 它们的 $q^{val_S(T)}$ 的和为 $[r_k(S)]_q$, 故一个 k 的贡献为 $[r_k(S)]_q [x]_q^{n-k}$ 。对所有 $0 \leq k \leq n$ 求和, 有

$$[r_n(S_x)]_q = \sum_{k=0}^n [r_k(S)]_q [x]_q^{n-k}$$

2. 从左往右考虑每一列放车对权值的影响。

考虑到第 i 列时, 其有 $x + a_i - i + 1$ 个位置可以放车。与前面类似地, 当其确定了放在哪个位置之后, 在这 $x + a_i - i + 1$ 个格子中在其上方的所有格子可以直接贡献到权

值。故第 i 列的贡献为 $q^0 + \dots + q^{x+a_i-i} = [x + a_i - i + 1]_q$ ，总贡献为每一列的贡献和的乘积，即

$$[r_n(S_x)]_q = \prod_{i=1}^n [x + a_i - i + 1]_q$$

合并两式即得式 (28)。 □

定理 8.2 在代入 $q = 1$ 时即为定理 6.1。特别地，可以代入 $x = 0$ ，由于 $[0]_q = 0$ ，有

引理 8.1 ([4])。对于 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n) \subseteq B_n$ ，有

$$[r_n(S)]_q = \prod_{i=1}^n [a_i - i + 1]_q \tag{29}$$

这样就得到了 $\sum_{T \in R_n(B_n), T \subseteq S} q^{vals(T)}$ 的封闭形式，即可以得到所有满足对应棋盘放车方案落在 S 内的排列 π 的逆序对数分布。

由于该式对于任意常数 q 是收敛的，因此也可以在该式中代入 q 计算所有满足对应棋盘放置方案在 S 内的排列 π 的逆序对带权和。计算所有满足条件的排列的逆序对个数和则可以将式 (29) 两侧看成关于 q 的多项式，对 q 求导之后再代入 $q = 1$ 。

同时注意到 $coinv(\pi) = inv(\pi^r)$ ，其中 π^r 表示将 π 翻转得到的排列，故对于减 Ferrers 棋盘也有类似的公式。

9 Ferrers 棋盘的排列环数计数

9.1 问题引入

例题 9.1 (数圈圈¹³)。给定整数 n, y 和一个长度为 n 的整数序列 $A = \{a_1, a_2, \dots, a_n\}$ ，保证序列 A 单调不减或单调不增。

构建有向图 $G(V, E)$ ，其中 $V = \{1, 2, \dots, n\}$ ， $E = \{(i, j) \mid 1 \leq i, j \leq n, a_i \geq j\}$ 。注意图 G 中可能包含自环。

定义边子集 $T \subseteq E$ 合法当且仅当图 $G'(V, T)$ 中每个点的入度和出度不超过 1，自环对对应点的入度和出度均贡献 1。定义一个合法边子集 T 的权值为 $y^{\text{cycle}(T)}$ ，其中 $\text{cycle}(T)$ 表示图 $G'(V, T)$ 的环数，自环是一个环。

特别地，本题认为 $0^0 = 1$ 。

对于所有整数 $0 \leq k \leq n$ ，求所有大小为 k 的合法边子集的权值和，对 998244353 取模。
 $1 \leq n \leq 10^5, 0 \leq a_i \leq n, 0 \leq y < 998244353$ 。

¹³来源：2021 集训队互测 Round 1 C，<https://uoj.ac/problem/597>

本题为笔者为 2021 年集训队互测命制的题目，由于互测的整体题目难度较高，本题得分率较低，其中定理 6.1 的部分分也鲜有选手得到。下文将讨论的定理 9.2、定理 9.3 和定理 9.6 是解决该问题将会用到的定理。

9.2 有向图的覆盖多项式

直接使用棋盘多项式的相关算法计算排列环数相关问题不太现实，因为棋盘数和棋盘多项式并没有记录与环有关的信息。因此需要对棋盘多项式的定义进行拓展。

注意到可以对 B_n 的所有子集和 n 个点的允许有自环但不能有重边的有向图建立双射：对于 $S \subseteq B_n$ ，若 $(i, j) \in S$ 则在有向图中从 i 到 j 连一条边。因此可以借助有向图上的工具来解决棋盘问题。

定义 9.1 (有向图的覆盖多项式, [2]). 对于 n 个点的有向图 G ，定义其覆盖多项式

$$C(G, x, y) = \sum_{p \geq 0} \sum_{q \geq 0} c_G(p, q) x^p y^q \quad (30)$$

其中 $c_G(p, q)$ 表示用恰好 p 条链和恰好 q 个环覆盖所有点恰好一次的方案数。一个点可以构成一条链，一个自环可以构成一个环。两个覆盖方案不同当且仅当使用的边集不同。

举例说明：对于 $n = 2$ ， G 中包含边 $(1, 2), (2, 1)$ 的有向图 G ，有四种覆盖方案：

1. $\{1\}, \{2\}$ 两条链；
2. $\{1, 2\}$ 一条链；
3. $\{2, 1\}$ 一条链；
4. $\{1, 2, 1\}$ 一个环。

因此 $C(G, x, y) = x^2 + x^1 + x^1 + y^1 = x^2 + x + y$ 。

有向图的覆盖多项式与对应棋盘的棋盘多项式之间有重要的联系：

定理 9.1 ([2]). 对于棋盘 $S \subseteq B_n$ ，设有向图 G 为棋盘 S 对应的有向图，则

$$C(G, x, 1) = \sum_{k=0}^n r_k(S) x^{n-k} \quad (31)$$

证明. 对比其 x^{n-k} 次项系数，即需要证明

$$r_k(S) = \sum_{q \geq 0} c(G, n-k, q) \quad (32)$$

对于某个有 $n-k$ 条链的覆盖方案，恰好有 $n-k$ 个点没有出边。且每个点的入出度均不超过 1，故其对应的棋盘每行列仅有一个格子，总共占据 k 个格子，故其对应一个 k 个车

的合法放置方案。与此同时对于一个 k 个车的合法放置方案可以通过上文构建的双射得到一个 $n - k$ 个点没有出度的图，即恰好用 $n - k$ 条链覆盖的方案。

因此用 $n - k$ 条链、环数不限的方式覆盖图 G 的方案数等于在 S 上放 k 个车的合法方案数，此时有式 (32) 成立，故式 (31) 成立。□

这意味着可以通过覆盖多项式来求解棋盘多项式。同时可以发现覆盖多项式实际上是棋盘多项式的推广，相比棋盘多项式其在环数上也进行了记录，即 $c_G(p, q)$ 表示的就是在棋盘 S 上放 $n - p$ 个车使得对应到图上有 q 个环的方案数。

例题 9.2 (棋盘多项式计算). 给定 n 和棋盘 $S \subseteq B_n$ ，对于每个 $T \subseteq \{1, 2, \dots, n\}$ 求 $S_T = \{(i, j) \in S \mid i, j \in T\}$ 的棋盘多项式，系数对大质数取模。 $n \leq 15, |S| \leq n \times n$ 。

设棋盘 S 对应的有向图为 G ，问题即对于所有子集 T 计算 G 对点集 T 的导出子图的覆盖多项式在 $y = 1$ 时的形式。

由于覆盖的基本单元是链和环，故先计算 f_T, g_T 表示覆盖点集 T 的链和环的数量。计算 f 和 g 均可以使用动态规划。 f 的计算容易做到 $O(2^n n^2)$ ，对于 g 朴素的动态规划需要记录覆盖点集、起点、终点，转移时需要枚举出边，复杂度 $O(2^n n^3)$ ，但由于需要计算的是环，故可以改变状态为 $dp_{T,j}$ 表示覆盖点集 T 、起点为 T 编号最小的点、终点为 j 的一条链，最后考虑终点到起点是否有边。复杂度优化到 $O(2^n n^2)$ 。

设 $P_T = x f_T + g_T$ ， Q_T 表示 T 的所有划分的 P 乘积和，即

$$Q_T = \sum_{k \geq 0} \sum_{T_1 \neq \emptyset, T_2 \neq \emptyset, \dots, T_k \neq \emptyset} [\cup_{i=1}^k T_i = T] [\forall 1 \leq i < j \leq k, T_i \cap T_j = \emptyset] \prod_{i=1}^k P_{T_i} \quad (33)$$

则由定理 9.1 有 $[x^k] Q_T = r_{|T|-k}(S_T)$ 。

从 P 到 Q 的运算即为集合幂级数的 Exp 运算¹⁴。直接进行 Exp 则需要对二元多项式进行 Exp 不容易实现，可以考虑代入 $x = 0, 1, 2, \dots, n$ 计算出每个 Q_T 在该处的点值然后插值求解。复杂度 $O(2^n n^3)$ 。

9.3 Ferrers 棋盘覆盖多项式分解定理

上文中在棋盘 S 和 n 个点的有向图 G 之间的双射要求 $S \subseteq B_n$ ，但在部分情况下，棋盘 S 的行列差巨大，此时将其对应的有向图的点数拓展到行列最大值则覆盖多项式次数将会巨大。同时由于绝大部分点没有入度或没有出度，因此每一个覆盖的方案都会有较多的点单独构成一条链，这是没有意义的。因此对棋盘额外定义覆盖多项式是必要的。

定义 9.2 (棋盘与放车方案的对应图). 对于一个棋盘或放车方案 S ，按如下过程构造其对应图 G_S ：

¹⁴关于集合幂级数与其运算，可参考 [12] 和 <https://www.luogu.com.cn/blog/command-block/wei-yun-suan-juan-ji-yu-ji-kuo-zhan>

1. 图 G_S 包含编号属于 \mathbb{Z} 的所有点;
2. 对于 $(i, j) \in S$, 在图 G_S 的 i 与 j 之间连一条有向边, 除此之外没有其他边。

定义 9.3 (棋盘的覆盖多项式). 定义所有元素的列标号小于等于 n 的棋盘 S 的覆盖多项式

$$C(S, x, y) = \sum_{i=0}^n \sum_{j \geq 0} r_{i,j}(S) x^{n-i} y^j \quad (34)$$

其中 $r_{i,j}(S)$ 表示在棋盘 S 上放 i 个车的合法方案中对应图的环数为 j 的方案数。

考虑在覆盖多项式上拓展定理 6.1。

定理 9.2 (Ferrers 棋盘覆盖多项式分解定理, [3]). 对于 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n)$,

$$C(S, x, y) = \prod_{1 \leq i \leq n, a_i \geq i} (x + a_i - i + y) \prod_{1 \leq i \leq n, a_i < i} (x + a_i - i + 1) \quad (35)$$

证明. 可将等式两边看做关于 x 的 n 次多项式, 因此仅需要在无限多个点 x 处证明式 (35) 成立即可证明式 (35) 恒成立。考虑在 $x \in \mathbb{N}$ 处证明。

如图 6 在棋盘 S 下方加上额外 x 行得到 S_x , 新加入的行从上往下编号 0 至 $-x+1$, 即在 G_S 上 1 至 n 所有点向 $-x+1$ 至 0 的所有点连一条有向边得到 G_{S_x} 。使用两种方式计算 $\sum_{T \in \mathcal{R}_n(S_x)} y^{\text{cycle}(G_T)}$:

1. 枚举 S 的合法子集 T , 计算所有合法方案中与 S 交集为 T 的方案贡献和。

由于对应图上编号小于 0 的点没有出边, 故其环数固定为 $\text{cycle}(G_T)$ 。设 $|T| = k$, 则剩余 $n - k$ 列放车方案数为 x^{n-k} 。满足 $|T| = p, \text{cycle}(G_T) = q$ 的 T 个数为 $r_{p,q}(S)$, 对所有 p, q 求和即

$$\sum_{T \in \mathcal{R}_n(S_x)} y^{\text{cycle}(G_T)} = \sum_{p \geq 0} \sum_{q \geq 0} r_{p,q}(S) x^{n-p} y^q = C(S, x, y)$$

2. 从左往右填入每个车并考虑贡献。

填入第 i 个车时, 有 $x + a_i - i + 1$ 个位置可以放车。

若列 i 满足 $a_i \geq i$, 则此时恰好存在一个位置放车后有向图上立即得到一个环: 考察当前有向图中 i 所在链的起点 p , 若 i 没有入度则起点是 i , 由于 $p \in [1, i]$ 所以放在 (i, p) 总是合法的, 并会得到一个环; 除此之外没有其他位置可以成环。

若列 i 满足 $a_i < i$, 由于棋盘 S 是 Ferrers 棋盘, 因此在图上点 i 一定没有入度。而 $a_i < i$, 因此不存在位置放车之后得到一个环。

认为一条边在加入后成环时产生 y 的贡献, 否则产生 1 的贡献, 那么对于 $a_i \geq i$ 的列贡献是 $x + a_i - i + y$, 对于 $a_i < i$ 的列贡献是 $x + a_i - i + 1$ 。故

$$\sum_{T \in \mathcal{R}_n(S_x)} y^{\text{cycle}(G_T)} = \prod_{1 \leq i \leq n, a_i \geq i} (x + a_i - i + y) \prod_{1 \leq i \leq n, a_i < i} (x + a_i - i + 1)$$

合并两式即得式 (35)。 □

当 $y = 1$ 时，其得到定理 6.1。

9.4 分解定理在减 Ferrers 棋盘上的拓展

在减 Ferrers 棋盘上定理 9.2 不容易拓展。

在定理 9.2 的证明中，第一种算法可以直接沿用。对于第二种算法，在减 Ferrers 棋盘上一个自然的想法是从右往左加入依次考虑贡献，但在定理 9.2 的证明中，对于 $a_i \geq i$ 的情况总是存在一个位置放车之后可以成环，但减 Ferrers 棋盘上这不总是能够实现的：考察例子 $S = F(3, 2, 2)$ ，若第三列放在 $(3, 2)$ ，则第二列一定无法成环，因为 $(2, 3) \notin S$ 。

原因是对于减 Ferrers 棋盘， $(j, i)(j > i) \in S, (i, i) \in S$ 并不能推出 $(i, j) \in S$ ，而在 Ferrers 棋盘上 $(j, i)(j < i) \in S, (i, i) \in S$ 可以推出 $(i, j) \in S$ 。故考虑对 $(j, i)(j > i) \in S, (i, i) \in S$ 可以推出 $(i, j) \in S$ 的减 Ferrers 棋盘先进行讨论：

定义 9.4 (竖直型减 Ferrers 棋盘). 定义减 Ferrers 棋盘 $S = F(a_1, a_2, \dots, a_n)$ 是竖直型减 Ferrers 棋盘当且仅当对于任意 $i < j$ ， $(j, i) \in S$ 可以推出 $(i, j) \in S$ 。

对于这一类棋盘，上面提到的问题不存在，故有

定理 9.3 (竖直型减 Ferrers 棋盘覆盖多项式分解定理, [3]). 对于竖直型减 Ferrers 棋盘 $S = (a_1, a_2, \dots, a_n)$ ，有

$$C(S, x, y) = \prod_{1 \leq i \leq n, a_i \geq i} (x + a_i - (n - i) + y - 1) \prod_{1 \leq i \leq n, a_i < i} (x + a_i - (n - i)) \quad (36)$$

证明. 在定理 9.2 的证明基础上对第二种算法进行修改。从右往左依次考虑每一列的放车位置，第 i 列的方案数为 $x + a_i - (n - i)$ 。

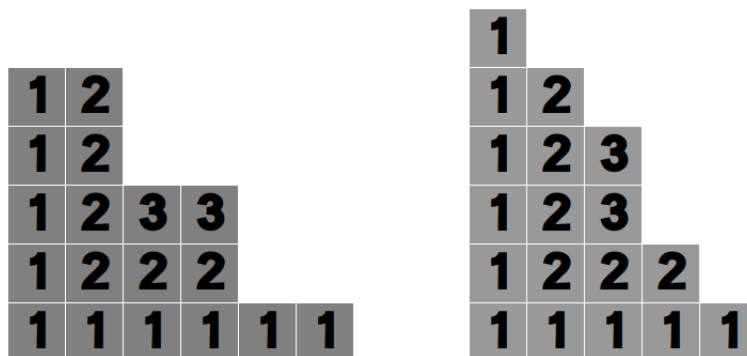
对于 $a_i < i$ 的列，由于其是减 Ferrers 棋盘，故图上点 i 没有入度。同时 i 也不能连向自己，所以不存在位置放车之后可以成环。

对于 $a_i \geq i$ 的列，若其在图上没有入度可以直接连成自环。否则设图上以 i 为终点的链从起点到终点依次为 $p_0, p_1, \dots, p_{k-1}, p_k = i$ 。设该序列的最大值为 p_v ，则 $v \neq k$ 。由 $(p_v, p_{v+1}) \in S, p_v > p_{v+1}$ 得 $(p_{v+1}, p_v) \in S$ ，即 $a_{p_{v+1}} \geq p_v$ 。而 $p_{v+1} \geq p_k$ ，故 $a_{p_k} \geq a_{p_{v+1}} \geq p_v \geq p_0$ ，故此时 (i, p_0) 可以放车。因此对于 $a_i \geq i$ 的情况存在恰好一个位置贡献为 y 。

故 $a_i \geq i$ 的列贡献为 $x + a_i - (n - i) + y - 1$ ， $a_i < i$ 的列贡献为 $x + a_i - (n - i)$ 。将所有列的贡献相乘即得到总贡献，即得式 (36)。 □

对任意减 Ferrers 棋盘 S ，对格子 $(i, j) \in S$ 编号 $\min(i, j)$ ，如图 14 的左棋盘所示。此时编号相同的格子构成 L 形。设编号为 i 的格子构成的 L 形下端长度为 q_i ，左端长度为 p_i ，如图 14 左图中 $p_1 = 5, q_1 = 6$ 。然后考虑对棋盘进行翻转：对于编号为 i 的 L 形，若 $p_i < q_i$ 则

图 14: 减 Ferrers 棋盘的编号与翻转操作实例



将 p_i 与 q_i 交换, 得到新的棋盘。图 14 的右图就是左图进行翻转得到的棋盘, 其中编号为 1, 3 的 L 形进行了翻转。

接下来将证明两个事实: 减 Ferrers 棋盘进行翻转操作之后得到的是一个竖直型减 Ferrers 棋盘; 翻转操作前后覆盖多项式不变。

首先对第一个事实进行证明。

引理 9.1. 棋盘 S 是减 Ferrers 棋盘当且仅当对所有在棋盘中的格子 (i, j) 编号 $\min(i, j)$ 后相同编号的格子构成一个 L 形, 且 $\forall p_i \neq 0, p_i > p_{i+1}, q_i > q_{i+1}$ 。

证明. 对于存在某一个编号不构成 L 形的棋盘, 一定存在某行或某列不满足恰有一段前缀的格子属于 S , 因此 S 一定不是减 Ferrers 棋盘。

考虑按照编号从大到小加入每一个 L 形并维护每一列的高度, 棋盘 S 是减 Ferrers 棋盘当且仅当在这个过程中每一个编号的 L 形加入后都得到减 Ferrers 棋盘。

设仅考虑编号 $\geq i + 1$ 的格子构成的棋盘为 $F(a_1, a_2, \dots, a_{q_{i+1}})$, 加入编号为 i 的 L 形相当于将序列 F 长度为 $q_i - 1$ 的前缀加一, 若序列 F 长度不够则在后面补充 0, 然后在序列最前面插入一个 p_i 。此时同时满足以下两个条件是新棋盘是减 Ferrers 棋盘的充要条件:

1. $q_i > q_{i+1}$, 若该条件不满足则第 q_{i+1} 列将不满足恰有一段行前缀属于 S ;
2. $p_i > p_{i+1}$, 若该条件不满足则序列 F 不满足单调不减。

即证明引理。 □

定理 9.4. 对减 Ferrers 棋盘进行翻转操作后得到的棋盘一定是减 Ferrers 棋盘。

证明. 由于减 Ferrers 棋盘编号后每个编号一定是 L 形, 因此只需要证明新的棋盘中 $p_i > p_{i+1}, q_i > q_{i+1}$ 即可。由于初始棋盘满足该条件, 因此若第 i 个和第 $i + 1$ 个 L 形均翻转或不翻转, 则条件显然成立。故仅需要考虑只有一个翻转的情况。

不妨假设编号为 i 的 L 形不翻转, 编号为 $i+1$ 的 L 形翻转, 另一种情况类似。设原棋盘第 $i, i+1$ 个 L 的左、下端长度为 $p_i, q_i, p_{i+1}, q_{i+1}$, 则 $p_i > q_i, p_{i+1} < q_{i+1}$ 。而 $p_i > q_i > q_{i+1}, q_i > q_{i+1} > p_{i+1}$, 因此这两个 L 形仍然满足条件。 \square

与此同时由于翻转后的棋盘满足 $p_i \geq q_i$, 所以其是竖直型减 Ferrers 棋盘。
 然后对第二个事实进行证明。我们不证明事实本身, 而证明容易推导出事实的结论。

定理 9.5. 对于减 Ferrers 棋盘 S 和其翻转得到的竖直型减 Ferrers 棋盘 S' , 设集合 P_S 表示 G_S 中的所有简单有向链和简单环构成的集合, 则存在一个双射

$$\phi : P_S \rightarrow P_{S'}$$

使得若 $T \in P_S$, 则

1. $\phi(T)$ 和 T 覆盖相同的点集;
2. $\phi(T)$ 是简单环当且仅当 T 是简单环。

一条简单有向链满足至少经过一个点且每个点至多经过一次, 一个简单环满足至少经过一个点且环上每个点的入出度均为 1。

证明. 考虑对 $v = \max_{p_i > 0} i$ 进行归纳。当 $v = 0$ 时, 两图都没有边, 因此 P 中仅有覆盖每个点的链, 此时 $\phi(\{v\}) = \{v\}$ 满足条件。

当结论对 0 至 $v-1$ 均成立时, 对于棋盘 S, S' , 由于将编号为 1 的格子删掉得到的两个棋盘 S_1, S'_1 满足 S'_1 是 S_1 翻转得到的, 因此存在一个双射 $\psi : P_{S_1} \rightarrow P_{S'_1}$ 满足题设条件。加入编号为 1 的格子可看做在对应图上插入编号为 1 的点, 原编号为正的点编号加一, 然后从 1 向一段正整数的前缀连边, 从另一段正整数的前缀向 1 连边, 自环不重复连接。

对于 $T \in P_S$, 若 $T \in P_{S_1}$ 则 $\phi(T) = \psi(T)$ 满足条件; 若 $T = \{1\}$ 则 $\phi(T) = T$ 。否则考虑 1 在 T 中的情况以及 S 变为 S' 的过程中是否翻转编号为 1 的 L 形, $\phi(T)$ 的取值如下表:

T 的形态	未翻转时 $\phi(T) =$	翻转时 $\phi(T) =$
$R_1, 1$	$\psi(R_1), 1$	$1, \psi(R_1)$
$1, R_1$	$1, \psi(R_1)$	$\psi(R_1), 1$
$1, R_1, 1$	$1, \psi(R_1), 1$	$1, \psi(R_1), 1$
$R_1, 1, R_2$	$\psi(R_1), 1, \psi(R_2)$	$\psi(R_2), 1, \psi(R_1)$

其中 R_1, R_2 为 P_{S_1} 中的某条简单有向链。用逗号将若干个路径或节点连接表示将它们拼接得到的简单有向链或简单环, 其中情况三表示的是简单环, 其余是简单有向链。

容易验证 ϕ 的一对原像和像满足同时不是环或同时是环, 且构成它们的点集是一致的, 故只需要证明这样的连边不会导致某条边不属于 S' 即可。

对于 R_1 或 R_2 大小 ≤ 1 的情况, 其在 ψ 下的像为其本身。在 L 形未翻转的情况下边没有发生改变, 不会导致连边不存在; 在 L 形翻转的情况下边的方向也发生了翻转, 其所对应的棋盘上的格子也做了翻转, 因此该边也是存在的。

对于 R_1 或 R_2 大小 > 1 的情况, 可以利用减 Ferrers 棋盘的直接推论“从 1 能直接到达以及能直接到达 1 的集合总是其他点的对应集合的超集”论证。由于情况较多仅举一例:

当 $T = 1, R_1$ 且编号为 1 的 L 形翻转时, 设 $\psi(R_1)$ 的终止节点为 w 且 $(w, 1) \notin G_S$, 即 $(1, w) \notin G_S$, 故 G_S 中 w 没有入度, 则 w 一定是 R_1 的起始节点。而 $T = 1, R_1$ 意味着 $(1, w) \in G_S$ 产生矛盾, 因此 $(w, 1) \in G_S$, 这条路径是合法路径。

对于其他情况可以类似证明。综合上面的讨论可以证明 ϕ 的合法性。 \square

综合上述两个事实, 有

定理 9.6. 对于任意减 Ferrers 棋盘, 存在一个竖直型减 Ferrers 棋盘与其拥有相同的覆盖多项式。

构造出对应的竖直型减 Ferrers 棋盘后计算出其覆盖多项式即可得到原减 Ferrers 棋盘的覆盖多项式。

该分解定理在轮廓线棋盘上使用会受到很大的限制, 在参考文献 [3] 中进行了一定的讨论, 但由于篇幅原因无法展现, 感兴趣的读者可自行阅读。

10 总结

本文对组合数学中的棋盘模型进行了简单的介绍, 通过对 Ferrers 棋盘的下降幂棋盘多项式的性质进行应用和推广得到了其在三种排列问题中的应用。

然而本文所介绍的棋盘理论知识仅仅是当下棋盘理论研究成果的冰山一角, 如: [7] 中利用棋盘模型得到了与二分图完美匹配相关的若干结论; [1] 中给出了定理 6.6 的 q -analogue 形式并利用其推导了拓展卡特兰数的若干性质; [10] 则利用棋盘理论对 $p - q$ 斯特林数给出了组合解释。这些问题由于本文篇幅有限, 且笔者能力有限没有找到它们在实际问题中的较好应用因而没有一一展开讨论。

希望本文可以起到抛砖引玉的作用, 让读者更好地认识棋盘理论, 更好地感受棋盘理论在计数问题中的强大应用。期待有更多使用棋盘模型的问题在信息学竞赛中出现。

11 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练高闻远的指导。

感谢家人对我的陪伴与支持。

感谢长郡中学谢秋锋老师的关心与指导。
感谢叶闻捷学长、郭晓旭前辈给予我写作本文的启发。
感谢长郡中学 2021 届信息组的同学给予我的帮助与陪伴。
感谢高子翼同学、谢睿杰同学为本文验稿并提出建议。
感谢其他给予我帮助的老师与同学。

参考文献

- [1] Kenneth Barrese, Nicholas Loehr, Jeffrey Remmel, and Bruce E. Sagan. m -level rook placements. *Journal of Combinatorial Theory*, 124:130–165, 2014.
- [2] F. R. K. Chung and R. L. Graham. On the cover polynomial of a digraph. *Journal of Combinatorial Theory Series B*, 65(2), 1995.
- [3] Morris Dworkin. Factorization of the cover polynomial. *Journal of Combinatorial Theory, Series B*, 71(1):17–53, 1997.
- [4] A. M Garsia and J. B Remmel. Q -counting rook configurations and a formula of frobenius. *Journal of Combinatorial Theory*, 41(2):246–275, 1986.
- [5] Jay Goldman and James Haglund. Generalized rook polynomials. *Journal of Combinatorial Theory*, 91(1-2):509–530, 2000.
- [6] Jay R. Goldman, J. T. Joichi, and Dennis E. White. Rook theory. i. rook equivalence of ferrers boards. *Proc. Amer. Math. Soc.*, 1975.
- [7] J. Haglund and J. B. Remmel. Rook theory for perfect matchings. *Advances in Applied Mathematics*, 27(2-3):438–481, 2001.
- [8] Jim Haglund, Ken Ono, and Lawrence Sze. Rook theory notes. 84(1):9–37, 1998.
- [9] John T. Hall and Jeffrey B. Remmel. Counting descent pairs with prescribed tops and bottoms. *Journal of Combinatorial Theory*, 115(5):693–725, 2008.
- [10] Jeffrey B. Remmel and Michelle L. Wachs. Rook theory, generalized stirling numbers and (p, q) -analogues. *The electronic journal of combinatorics*, 11(1), 2004.
- [11] 卢开澄, 卢华明. 组合数学. 第 4 版. 清华大学出版社, 2006.
- [12] 吕凯风. 集合幂级数的性质与应用及其快速算法. 2015 年信息学奥林匹克中国国家集训队论文集, 2015.

- [13] 彭思进. ioi2021 国家集训队第一阶段作业试题交流《permutation》解题报告, 2020.

对信息学比赛中选手分数数据的分析

江苏省天一中学 邱天异

摘要

本文建立了描述信息学比赛的数学模型，并基于该模型研究了过往比赛的选手分数数据。本文通过统计确定了同一名选手的得分波动所服从的分布，并基于此从联赛分数推算出了选手整体水平的分布情况。

1 引言

中国高中信息学竞赛的参赛人数和竞赛水平在最近十年中快速提高；这种迅猛的发展在让竞赛趋于繁盛的同时，也使得选手和教练对竞赛现状的认知难以跟上节拍。

这一情况引起了一些问题，例如：

- 选手对于自己所处的水平段认识不足，从而作出错误的学业规划。
- 出题人对于选手的水平认识不足，导致题目难度和部分分分配失当。
- 选手不了解对手的水平 and 发挥情况，导致选择了错误的考场策略。

本文将利用数学工具，基于过往比赛的选手分数数据来分析信息学竞赛的现状，以为上述问题的解决提供助力。

本文中用到的全部数据和计算程序可以在以下网址下载：

- <https://files.cnblogs.com/files/turboboost/qty-thesis-statdata.zip>
- <https://github.com/TianyiQ/ioi2021-thesis/blob/main/qty-thesis-statdata.zip>

正文分为三个部分：

第二节 建立用于描述信息学比赛的数学模型，作为后续分析的基础。

第三节 分析同一名选手的得分波动所服从的分布。

第四节 利用 NOIP 初赛、复赛的得分数据推算出信息学竞赛选手整体水平的分布情况。

由全文的目标决定，本文将不会对初中信息学竞赛进行研究，因此下文中在提到任何比赛时默认指面向高中生的比赛。

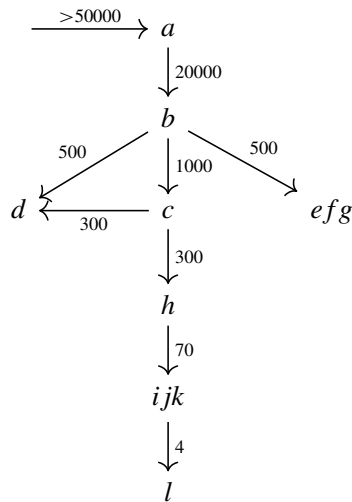


图 1: 信息学比赛间的关系

2 建立模型

2.1 赛程和赛制

在引入模型前，先对信息学竞赛的竞赛流程和比赛形式作简要介绍¹。

信息学竞赛是一系列比赛的统称。这些比赛整体上呈现“逐级递进”的关系，即下一层比赛的优胜者晋级上一层比赛。这些比赛按照级别从低到高，大致排列为²：

- a. 全国联赛 (NOIP/CSP) — 初赛
- b. 全国联赛 (NOIP/CSP) — 复赛
- c. 省队选拔赛
- d. 清华/北大学科营 (THUWC/PKUWC/THUSC/PKUSC)
- e. 亚太地区竞赛 (APIO)
- f. 国家队选拔赛 (CTSC/CTS) — 非正式选手
- g. 全国冬令营 (NOIWC) — 非正式选手
- h. 全国决赛 (NOI)
- i. 清华/北大集训 (CTT)
- j. 全国冬令营 (NOIWC) — 正式选手
- k. 国家队选拔赛 (CTSC/CTS) — 正式选手
- l. 国际奥林匹克竞赛 (IOI)

图 1 展示了这些比赛间的关系。箭头从低级别比赛指向高级别比赛，表示该低级别比赛的优胜者可以晋级对应的高级别比赛，箭头上标记的数值表示大致晋级人数。

¹赛程和赛制在近几年有小幅变化，本小节中会尽量兼顾新旧两套机制

²后文将用下表中的字母标号来代指对应的比赛

	时长	题数	题目类型	反馈机制	对应比赛
笔试	1~2h	数十	选择题、填空题	无反馈	a
COI 赛制	3~5h	3~4	编程题, 有多档部分分	无反馈	bcfghk
IOI 赛制	3~5h	3~4	编程题, 有多档部分分	多次提交、有反馈	deijl

表 1: 信息学比赛采用的赛制

赛制即比赛的进行方式和比赛规则。信息学竞赛中采用笔试、COI 赛制（机试）、IOI 赛制（机试）这三种不同的赛制，表 1 给出了每种赛制的特点和先前提到的比赛所分别采用的赛制。

2.2 数学模型

本小节中将建立用于描述一场信息学比赛的数学模型。

2.2.1 基本模型

为了更清晰地界定模型在现实中的适用范围，需要先明确：现实中怎样的对象能被称为一场“比赛”。

定义 2.1 (现实比赛). 一个**现实比赛**，即特定的人群在同样的规则下测试同一套题目的过程。一个现实比赛被参赛人群、规则和题目这三个要素所确定。

在这一定义下，每年中的 $a \sim l$ 这 12 个比赛，自然都是现实比赛。而且，不仅是包含两天考试的一场完整的比赛算作现实比赛，单独拿出其中一天也算现实比赛。

关于“参赛人群”这一概念需要注意两点：

- 参赛人群只是一个宽泛的范围，而不是具体的选手集合。例如我们可以规定参赛人群为“所有学习信息学的同学”，但这一规定并不关注张三、李四、王五是否是这个人群的成员。这样的规定不会给后续的分析带来不利影响，因为我们只关心关于比赛和人群的统计信息，而不关心每名选手的特点。

- 参赛人群不必囊括实际参赛的整个选手群体；例如在 NOIP 初赛中，“所有报名了初赛的女生”这一参赛群体依然能构成现实比赛。这一点对于后文中跨越不同比赛的分析大有帮助。

接下来定义从现实比赛抽象而来的数学模型。

定义 2.2 (理想比赛). **理想比赛** A 由二元函数 $H_A : [0, 1] \times \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$ 确定，其中 H_A 连续且满足

$$\int_0^1 \int_{-\infty}^{+\infty} H_A(x, \delta) d\delta dx = 1 \quad (1)$$

和

$$\int_{-\infty}^{+\infty} \delta \cdot \mathbf{H}_A(x_0, \delta) d\delta = 0, \quad \forall x_0 \in [0, 1] \quad (2)$$

此时我们把 \mathbf{H}_A 称为 A 的综合分布函数。

接下来将定义：一个理想比赛何时被认为“描述”了一个现实比赛。这也将同时表明综合分布函数的实际含义。

首先约定一下记号：

- $\Pr[A]$ 表示事件 A 发生的概率。
- $E[X]$ 表示随机变量 X 的期望值。

定义 2.3. 从现实比赛 B 可按如下方式确定一个理想比赛 A ：

步骤 1. 记 B 的参赛选手集合为有限集 S_B ，并在 B 的参赛人群（包括人群内部的具体构成）不变的情况下，假想参赛人数 $|S_B|$ 趋于无穷。我们之所以能够任意钦定 $|S_B|$ ，是因为——如先前所述—— B 的定义并未指明具体的选手集合。

步骤 2. 每一名参赛选手 p 在比赛 B 中的实际得分 $score_p$ 是一个随机变量，它被各种偶然因素（如临场发挥）所支配，但是它的分布可以由选手 p 和现实比赛 B 的三个要素完全确定。假想对每一名选手 p 计算其期望得分 $exscore_p = E[score_p]$ ，并取所有选手期望得分的最大值，记作 M_B 。由于参赛人数趋于无穷，每一个个人的特征可以忽略，故 $M_B = \max_{p \in S_B} exscore_p$ 仅由 B 确定。

步骤 3. 从 S_B 中等概率随机选取一名选手 p ，并：

- 定义 $[0, 1]$ 上的随机变量 $X_B = \frac{exscore_p}{M_B}$ 。³ 易见随机变量 X_B 的实际取值与考场上的偶然因素无关，而是由选取 p 的方式确定。
- 定义 \mathbb{R} 上的随机变量 $\Delta_B = \frac{score_p - exscore_p}{M_B}$ 。易见随机变量 Δ_B 的实际取值由选取 p 的方式和考场上的偶然因素（如选手临场发挥）共同确定。
- 请注意， X_B 和 Δ_B 的定义中所用的 p 是同一名随机选择的选手，而不是独立的两次选择。

步骤 4. 取 A 的综合分布函数 \mathbf{H}_A 为 X_B 与 Δ_B 的联合概率密度函数，从而确定 A 。换句话说，对所有 $x_0 \in [0, 1], \delta_0 \in \mathbb{R}$ ，需要满足⁴

$$\int_0^{x_0} \int_{-\infty}^{\delta_0} \mathbf{H}_A(x, \delta) d\delta dx = \Pr[(X_B \leq x_0) \wedge (\Delta_B \leq \delta_0)] \quad (3)$$

³ “将每名选手的分数除以最高分数”这一操作，类似于信息学比赛中计算标准分的方式。另外注意到：虽然 $\frac{exscore_p}{M_B} \leq 1$ ，但 $\frac{score_p}{M_B}$ 可以大于 1

⁴也可以直观地理解为 $\mathbf{H}_A(x_0, \delta_0) = \Pr[(X_B \approx x_0) \wedge (\Delta_B \approx \delta_0)]$ ，不写作 $X_B = x_0$ 是因为取等概率为 0

由(3)知定义 2.2 的等式(1)满足；由 $E[\Delta_B] = 0$ 知等式(2)满足。从而只要联合概率密度函数 H_A 存在且连续， A 就符合理想比赛的定义。

对于按上述方式得到的 A ，我们称 A 与 B 互相对应。如果按上述过程得到的 H_A 不连续或根本不存在，则认为不存在与 B 对应的 A 。

冗长的定义可以用一句话来作直观的总结： $H_A(x, \delta)$ 表示真实水平（即期望得分）约为 x （ $x \in [0, 1]$ 为按最高分折算后的标准分）、实际表现约为 $x + \delta$ （同样表示标准分）的选手的期望人数占总人数的比例；之所以实际表现会偏离真实水平——以及这里之所以说“期望人数”——是因为考场上的各种偶然因素为比赛结果带来了随机性。

可以看到，理想比赛这一模型只考虑了哪些结果可能出现，而未考虑哪种结果实际出现。而在现实中，能够获知的却只有实际出现的结果——和它恰恰相反。下面定义的概念将处理这一问题。

定义 2.4 (分数分布函数). 对理想比赛 A ，定义其分数分布函数 $C_A: \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$ 满足

$$C_A(s) = \int_0^1 H_A(x, s-x) dx, \quad \forall s \in \mathbb{R}$$

命题 2.5 (分数分布函数的实际含义). 对现实比赛 B 和与之对应的理想比赛 A ，假想比赛 B 的参赛人数 $|S_B|$ 趋于无穷，等概率随机选取选手 $p \in S_B$ ，则⁵：

$$\Pr \left[\frac{\text{score}_p}{M_B} \leq r \right] = \int_{-\infty}^r C_A(s) ds, \quad \forall r \in \mathbb{R}$$

证明.

$$\begin{aligned} \Pr \left[\frac{\text{score}_p}{M_B} \leq r \right] &= \Pr [X_B + \Delta_B \leq r] \\ &= \iint_{\{(x, \delta): x \in [0, 1], \delta \in \mathbb{R}, x + \delta \leq r\}} H_A(x, \delta) d(x, \delta) \\ &= \iint_{\{(x, s): x \in [0, 1], s \in (-\infty, r]\}} H_A(x, s-x) d(x, s) \\ &= \int_{-\infty}^r \left(\int_0^1 H_A(x, s-x) dx \right) ds \\ &= \int_{-\infty}^r C_A(s) ds \end{aligned}$$

□

⁵也就是说 C_A 为随机变量 $\frac{\text{score}_p}{M_B}$ 的概率密度函数。和先前类似，这里也可以直观理解为 $C_A(r) = \Pr \left[\frac{\text{score}_p}{M_B} \approx r \right]$

在上面四个定义中，涉及到现实情况的部分难免有模糊之处；实际应用中对这几条定义的执行，也不可避免地需要作近似处理。但即便如此，作出这些规定依然能极大地帮助我们厘清思路并发现隐含的前提。

2.2.2 特殊情况下的模型

在一场现实比赛 B 中，每一个选手 $p \in S_B$ 的实际得分相比真实水平的“得分偏移量” $\frac{\text{score}_p - \text{exscore}_p}{M_B}$ 都是一个随机变量。如果所有选手的“得分偏移量”独立同分布，对我们的模型意味着什么？

容易想到，此时随机变量 Δ_B 的概率分布就和任何一个选手的“得分偏移量”的概率分布完全相同。换句话说，在定义 2.3 的步骤 3 中，不论我们钦定选取哪一个 p ， Δ_B 取任何一个值的概率都是固定的，且恰好等于在不固定 p 的情况下 Δ_B 取这个值的概率。再换句话说⁶：

$$\Pr[(\Delta_B \leq \delta) | (X_B = x)] = \Pr[\Delta_B \leq \delta], \quad \forall (\delta \in \mathbb{R}, x \in [0, 1], \Pr[X_B = x] > 0)$$

即随机变量 X_B, Δ_B 独立。在研究这件事之前，我们需要一对新的定义。

定义 2.6 (期望值分布函数和偏移量分布函数). 对任意的理想比赛 A ：

- 定义其期望值分布函数 $X_A : [0, 1] \rightarrow \mathbb{R}_{\geq 0}$ 满足

$$X_A(x_0) = \int_{-\infty}^{+\infty} H_A(x_0, \delta) d\delta, \quad \forall x_0 \in [0, 1]$$

- 定义其偏移量分布函数 $\Delta_A : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$ 满足

$$\Delta_A(\delta_0) = \int_0^1 H_A(x, \delta_0) dx, \quad \forall \delta_0 \in \mathbb{R}$$

命题 2.7 (期望值分布函数和偏移量分布函数的实际含义). 对现实比赛 B 和与之对应的理想比赛 A ：

- X_A 为 X_B 的概率密度函数。换句话说⁷：

$$\Pr[X_B \leq x_0] = \int_0^{x_0} X_A(x) dx, \quad \forall x_0 \in [0, 1]$$

- Δ_A 为 Δ_B 的概率密度函数。换句话说⁸：

$$\Pr[\Delta_B \leq \delta_0] = \int_{-\infty}^{\delta_0} \Delta_A(\delta) d\delta, \quad \forall \delta_0 \in \mathbb{R}$$

⁶和之前类似，这里之所以不写 $\Delta_B = \delta$ ，是因为取等概率为 0

⁷也可以直观地理解为： $\Pr[X_B \approx x_0] = X_A(x_0)$

⁸也可以直观地理解为： $\Pr[\Delta_B \approx \delta_0] = \Delta_A(\delta_0)$

证明比较显然，这里略去。下面考虑 X_B, Δ_B 间的独立性带来的性质。

命题 2.8. 对现实比赛 B 和与之对应的理想比赛 A ，如果 X_B 与 Δ_B 独立，则：

$$H_A(x_0, \delta_0) = X_A(x_0)\Delta_A(\delta_0), \quad \forall (x_0, \delta_0) \in [0, 1] \times \mathbb{R} \quad (4)$$

更进一步，(4)是 X_B 与 Δ_B 独立的充要条件。

证明.

$$\begin{aligned} \Pr[(\Delta_B \leq \delta_0)|(X_B = x_0)] &= \Pr[\Delta_B \leq \delta_0], \quad \forall (\delta_0 \in \mathbb{R}, \Pr[X_B = x_0] > 0) \\ \Leftrightarrow \left(\int_{-\infty}^{\delta_0} H_A(x_0, \delta) d\delta \right) / X_A(x_0) &= \int_{-\infty}^{\delta_0} \Delta_A(\delta) d\delta, \quad \forall (\delta_0 \in \mathbb{R}, X_A(x_0) > 0) \\ \Leftrightarrow \frac{H_A(x_0, \delta_0)}{X_A(x_0)} &= \Delta_A(\delta_0), \quad \forall (\delta_0 \in \mathbb{R}, X_A(x_0) > 0) \\ \Leftrightarrow (4) \end{aligned}$$

最后一步中还需要特别考虑 $X_A(x_0) = 0$ 的情况，不难自行补全。 \square

定义 2.9 (简单理想比赛). 如果理想比赛 A 满足 (4) 式，则称它是简单的。

由命题 2.8，对于简单理想比赛 A ，从 X_A, Δ_A 可唯一确定 H_A ，进而能够确定 C_A 。

命题 2.10 (简单理想比赛的分数分布函数). 对简单理想比赛 A ：

$$C_A(s) = \int_0^1 X_A(x)\Delta_A(s-x)dx, \quad \forall s \in \mathbb{R}$$

证明显然，这里略去。

2.3 几个关键的假设

为了使得后续分析成为可能，我们还需要对真实情况作一些近似处理。近似处理的具体方式由本小节的几个假设给出。

假设 2.11. 对任何一个信息学（现实）比赛 B ，都存在符合定义 2.2 的理想比赛 A 与其对应。

假设 2.12. 对任何一个现实比赛，如果它的规则基于 COI 或 IOI 赛制，则它对应的理想比赛是简单的。

假设 2.13. 考虑所有基于 COI 或 IOI 赛制的现实比赛，考察它们对应的理想比赛的偏移量分布，这些分布应该是相似的，即它们应该有相同的形式，即使其中的参数可能有不同的取值。

在给出下一个假设之前，还需要定义一个概念。

定义 2.14 (缩放等价). 对简单理想比赛 A_1, A_2 ，当存在线性映射 $f(x) = \alpha x + \beta$ ($\alpha \in \mathbb{R}_{>0}, \beta \in \mathbb{R}$) 同时满足以下条件时，称 A_1, A_2 缩放等价，称 f 为 A_1, A_2 间的等价映射：

1. $f(1) = 1$
2. $\Delta_{A_2}(\alpha\delta) = \Delta_{A_1}(\delta) \cdot \alpha^{-1}, \quad \forall \delta \in \mathbb{R}$
3. $\bar{X}_{A_2}(f(x)) = \bar{X}_{A_1}(x) \cdot \alpha^{-1}, \quad \forall x \in \mathbb{R}$ ，其中

$$\bar{X}(x) = \begin{cases} 0 & x \notin [0, 1] \\ X(x) & x \in [0, 1] \end{cases}$$

命题 2.15 (缩放等价的实际含义). 对缩放等价的 A_1, A_2 及其等价映射 f ，有以下关系⁹：

1. $\int_{-\infty}^{\delta_0} \Delta_{A_1}(\delta) d\delta = \int_{-\infty}^{\alpha\delta_0} \Delta_{A_2}(\delta) d\delta, \quad \forall \delta \in \mathbb{R}$
2. $\int_{-\infty}^{x_0} \bar{X}_{A_1}(x) dx = \int_{-\infty}^{f(x_0)} \bar{X}_{A_2}(x) dx, \quad \forall x \in \mathbb{R}$

证明. 先来看关于 $\Delta_{A_1}, \Delta_{A_2}$ 的部分：

$$\begin{aligned} \int_{-\infty}^{\delta_0} \Delta_{A_1}(\delta) d\delta &= \int_{-\infty}^{\delta_0} \Delta_{A_2}(\alpha\delta) \alpha d\delta \\ &= \int_{-\infty}^{\alpha\delta_0} \Delta_{A_2}(\alpha\delta) \alpha d(\alpha\delta) \cdot \alpha^{-1} \\ &= \int_{-\infty}^{\alpha\delta_0} \Delta_{A_2}(t) dt \end{aligned}$$

对于 $\bar{X}_{A_1}, \bar{X}_{A_2}$ 同理，这里不再重复。 □

假设 2.16. 对现实比赛 B_1 (对应理想比赛 A_1) 和 B_2 (对应理想比赛 A_2)，如果 B_1, B_2 的规则都基于 COI 或 IOI 赛制，且它们的参赛人群相同，则 A_1, A_2 一定缩放等价。

⁹可以直观理解为：现实比赛 B_1 (对应于 A_1) 中的分数，经过 $f: x \mapsto \alpha x + \beta$ 的变换之后，变成了现实比赛 B_2 (对应于 A_2) 中的分数

对这些假设无法予以严格的证明，但在此可以列举一些感性的理由，来说明它们大体上是可靠的。

1. 如果假设所有比赛在考查角度上没有差异（因为我们只关心普遍的统计特征，所以这种假设是合理的），那么一名选手的解题能力（即，能够在比赛中解出多大难度的题目）就一定是固定的。

2. 当组题人为一场比赛选择题目、出题人为命制的题目设置部分分时，他们会有意识地给较难的任务设置较高的分值、给较简单的任务设置较低的分值，而具体多高、多低，则取决于他们心中作的判断。虽然不同的人可能作出不一样的判断，但这些判断应该大体上是“成比例”的。例如：张三认为算法 2 应当获得三倍于算法 1 的得分、李四认为算法 2 应当获得 2.5 倍于算法 1 的得分，这两种判断在比例上是大致相符的。

3. 综合 1 和 2，我们知道了：每个选手的能力可以看作是不变的；选手比赛中完成的任务难度与所获分数间的关系，这一关系在不同比赛之间应该是“成比例”的。所以只要选手集合不变，不同比赛的“选手期望得分构成的分布”也应该是“成比例”的（特别地，这两个分布的最大值也应该是相对应的，所以在定义 2.14 中要求 $f(1) = 1$ ）。这就为假设 2.16 关于期望值分布的部分和对 $f(1) = 1$ 的要求提供了依据。

4. 根据经验，一名选手考场发挥的稳定与否与水平高低等因素没有明显的相关性；所以虽然不同选手的稳定性存在差异，但是在样本很大时，这种差异不会给统计结果带来较大的系统性的偏差，因此我们近似地认为所有选手水平发挥的稳定性是相同的。又因为得分与实际表现出的能力是“成比例”的，所以所有选手比赛得分的稳定性也是相同的。这为假设 2.12、假设 2.13 和假设 2.16 关于偏移量分布的部分提供了依据。

5. 真实的比赛中“离散”的特性——比如选择题三分一道——可以在理想化的模型中忽略。这样在人数趋于无穷时，我们很容易想到：其各种统计数据会是“连续”的。因此 2.11 是一个很自然的假设。

6. 根据经验，在 COI 赛制中表现好的选手，在 IOI 赛制往往表现也很好；反之亦然。因此 COI/IOI 赛制间的差异至多会对选手期望得分的分布起到缩放的作用，而不会带来本质的改变。类似地，选手在 COI/IOI 赛制中发挥稳定性的差异，也只有量的差别而无质的差别。所以，认为 COI/IOI 赛制的比赛有着本质相同（即在缩放后完全相同）的期望值分布、偏移量分布，是合理的。

7. 假设 2.12 会带来一个问题：如果一名选手的期望得分十分接近 0，但他的分数波动的幅度仍被认为与其他选手相同，就会使他可能考出“负分数”，并使得分分布函数在负数处的点值非零。由于本文只研究近似的结果，且考虑到该现象并不会十分显著（因为一场比赛中只会有很少的选手期望得分接近 0），所以可以容忍这一不合理的现象。

	参赛总人数	正式选手人数	非正式选手人数	选拔人数
北大集训 2018	约 60	50	约 10	15
北大集训 2019	约 70	50	约 20	15
北大集训 2020	约 90	50	约 40	30

表 2: 三场比赛的参赛情况

3 偏移量分布的测量

由假设 2.13, COI/IOI 赛制下偏移量分布有一定的形式。本节中, 将利用过往比赛的分数数据得到偏移量分布的形式。

3.1 数据的获取

数据来自以下三场比赛:

- 2018 年北大集训 (字母标号 i)
- 2019 年北大集训 (字母标号 i)
- 2020 年北大集训 (字母标号 i)

选用它们的原因是, 北大集训包含连续进行的四场考试, 更多的考试场数使得我们能够更精确地估计每一名选手的期望分数。

这些比赛的参赛情况见表 2。

根据经验判断, 这三场比赛中并非所有选手都全情投入。因此为了保证数据可靠性, 对每场比赛只取总排名¹⁰中最靠前的 $1.5K \sim 2K$ 名选手的数据, 其中 K 表示当场比赛的选拔人数。具体地说: 北大集训 2018 取前 30 名、北大集训 2019 取前 30 名、北大集训 2020 取前 50 名。另外为保证比赛之间的统一性, 后文中在计算考试分数标准差时, 每场比赛只取总排名中前 30 名的分数。

3.2 数据的加工处理

三场比赛的参赛选手共计 110 人次, 我们将他们视为 110 名不同的选手。三场比赛共计 12 场考试, 我们将它们视为 12 个不同的现实比赛。参加这些现实比赛的共计 440 人次。

虽然这 12 个现实比赛的参赛人群是相同的 (国家集训队选手和精英培训选手), 但它们在题目难度等方面并不相同, 如果直接将它们的数据汇总起来的话, 会使得数据失去意义。为解决这一问题, 我们需对比赛得分进行变换。

¹⁰总排名中按每天标准分总和降序排列

命题 3.1. 对缩放等价的理想比赛 A_1, A_2 及其等价映射 $f(x) = \alpha x + \beta$, 有

$$\alpha = \frac{\text{Stddev}[C_{A_2}]}{\text{Stddev}[C_{A_1}]}$$

其中 $\text{Stddev}[F]$ 表示以 F 为概率密度函数的随机变量¹¹的标准差。

另外注意到由 $f(1) = 1$ 可得 $f(x) = 1 - \alpha(1 - x)$, 所以不必再考虑 β 的取值。

证明.

$$\begin{aligned} C_{A_1}(s) &= \int_0^1 X_{A_1}(x) \Delta_{A_1}(s-x) dx \\ &= \int_{-\infty}^{+\infty} X_{A_1}(x) \Delta_{A_1}(s-x) dx \\ &= \int_{-\infty}^{+\infty} (\bar{X}_{A_2}(\alpha x + \beta) \cdot \alpha) (\Delta_{A_2}(\alpha(s-x)) \cdot \alpha) dx \\ &= \int_{-\infty}^{+\infty} \alpha^2 \bar{X}_{A_2}(\alpha x + \beta) \Delta_{A_2}(\alpha s + \beta - (\alpha x + \beta)) d(\alpha x + \beta) \cdot \alpha^{-1} \\ &= \alpha \int_{-\infty}^{+\infty} \bar{X}_{A_2}(t) \Delta_{A_2}((\alpha s + \beta) - t) dt \\ &= \alpha C_{A_2}(\alpha s + \beta), \quad \forall s \in \mathbb{R} \end{aligned}$$

设连续型随机变量 Y_1 满足其概率密度函数为 C_{A_1} , Y_2 满足其概率密度函数为 C_{A_2} , 则 $\alpha Y_1 + \beta$ 与 Y_2 同分布。从而¹² $\text{Var}[Y_2] = \text{Var}[\alpha Y_1 + \beta] = \alpha^2 \cdot \text{Var}[Y_1]$, 于是 $\text{Stddev}[Y_2] = \alpha \cdot \text{Stddev}[Y_1]$ 。 \square

结合等价映射的实际含义和命题 3.1, 可以得到对前述 12 个现实比赛 $B_{1\dots 12}$ 的分数做变换的方法:

步骤 1. 记 $B_{1\dots 12}$ 对应的理想比赛为 $A_{1\dots 12}$ 。

步骤 2. 构造 $A'_{1\dots 12}$ 满足 A_i 与 A'_i 缩放等价, 且等价映射为 $f_i(x) = 1 - \frac{1-x}{c \cdot \text{Stddev}[C_{A_i}]}$ 。这里 $c = 4$ 为根据实际数据所选取的固定常数, 用来避免产生负分数。

步骤 3. 则 $A'_{1\dots 12}$ 这 12 个理想比赛完全相同 (即它们的综合分布函数相同), 且与 $A_{1\dots 12}$ 中的每一个缩放等价。

¹¹ 换句话说, 这样的随机变量 Y 满足 $\Pr[Y \leq t] = \int_{-\infty}^t F(s) ds, \quad \forall t \in \mathbb{R}$

¹² 这里 $\text{Var}[Y]$ 表示随机变量 Y 的方差

另外须注意, 根据定义 2.3 的步骤 2, 我们需要对每个现实比赛 B_i 确定选手期望分数的最大值 M_{B_i} 。这里可以用实际分数的最大值来近似地代替期望分数的最大值。

因为 $A'_{1\dots 12}$ 与 $A_{1\dots 12}$ 中的每一个缩放等价, 所以我们只需测量 $A'_{1\dots 12}$ 的偏移量分布, 即可得到结论。现在开始目标将转为测量 $A'_{1\dots 12}$ 的偏移量分布; 为便于表述, 记 $B'_{1\dots 12}$ 表示 $A'_{1\dots 12}$ 对应的现实比赛。

现在我们得到了 12 个完全相同的理想比赛 $A'_{1\dots 12}$, 和每个理想比赛对应的现实比赛的分数数据; 而因为 $A'_{1\dots 12}$ 完全相同, 所以所有这些分数数据可以直接合并。现在我们有了一个理想比赛 (记为 A' , 对应现实比赛 B') 和对应的 440 名选手的分数数据。原先的 110 名选手, 每人对应着 B' 中的 4 名选手。

对于 110 名选手中的每一位, 为了能够对比他在 B' 中的期望分数和他的四个“分身”的实际分数, 我们还需要估算前者的值。这里可以用该名选手在他所参加的 4 场现实比赛 B'_i 中的平均分, 来近似地代替在 B' 中的期望分数。

综上所述, 我们会按如下的流程来加工分数数据:

步骤 1. 对 12 场考试中的每一场, 将其中每一名选手的分数除以该场考试的最高分¹³, 并以此代替原始分数。

步骤 2. 对 12 场考试中的每一场, 计算总排名前 30 的选手的分数标准差 σ (这里的分数是指步骤 1 中得到的商), 然后将其中每个选手的分数 x 施以变换¹⁴ $x \mapsto 1 - \frac{1-x}{4\sigma}$, 并以此代替原始分数。

步骤 3. 对 110 名选手中的每一位, 计算他在 4 场考试中的平均分, 然后计算他在每场考试中的得分与这一平均分的差。

这样可以对每名选手计算出 4 个差值, 共计 440 个值, 每个值都表示一名选手在一场比赛中实际得分与期望得分的差距。这 440 个值即对应着随机变量 $\Delta_{B'}$ 的取值, 它们将会是下一小节的分析对象。

3.3 拟合的方法和结果

观察上一小节中获得的 440 个数值的分布情况, 发现:

- 整个分布大体上对称, 且以 0 为对称中心。
- 数值的分布中间稠密、两边稀疏, 所有数值的绝对值都小于 1。
- 分布的形状类似钟形曲线。

受此启发, 尝试用正态分布曲线来拟合这些数值。具体方法如下:

步骤 1. 对于 $t = -1.0, -0.9, \dots, 1.0$, 计算: 落在 $[t - 0.05, t + 0.05)$ 中的数值个数与总个数 440 的比值。这个比值记作 $c(t)$ 。

¹³即信息学比赛中计算标准分的过程

¹⁴除以标准差这一步的作用也可简单理解为, 消除题目区分度不同所带来的影响

步骤 2. 在平面直角坐标系中画出 $t - c(t)$ 散点图。

步骤 3. 选取合适的参数 $\sigma > 0$ ，以使得函数

$$f(t) = \int_{t-0.05}^{t+0.05} P_{\sigma^2}(x) dx$$

的图像与这些 $t - c(t)$ 数据点尽可能贴近。

这里 P_{σ^2} 表示期望值为 0、方差为 σ^2 的正态分布（用 $N(0, \sigma^2)$ 表示）的概率密度函数，满足

$$P_{\sigma^2}(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

再记

$$R_{\sigma^2}(t) = \int_{-\infty}^t P_{\sigma^2}(x) dx$$

为正态分布 $N(0, \sigma^2)$ 的累积分布函数，则有¹⁵ $R_{\sigma^2}(t) = \left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}\sigma}\right)\right)/2$ ，其中 erf 表示误差函数。

最后注意到

$$\int_{t-0.05}^{t+0.05} P_{\sigma^2}(x) dx = R_{\sigma^2}(t + 0.05) - R_{\sigma^2}(t - 0.05)$$

于是在进行拟合的过程中我们可以方便地计算这一定积分。

图 2 展示了拟合的结果。可以看到，除了约 3 个数据点以外，其余数据点均与曲线贴合紧密。为了验证这些数据是否确实服从正态分布，还需绘制 Q-Q 图来进行检验。

图 3 展示了所绘制的 Q-Q 图。注意，该图的坐标轴经过缩放，故坐标轴上标注的数值仅能代表相对的比例关系。

图 3 中有 440 个蓝色叉号，所有叉号的横坐标、纵坐标非严格递增。其中第 k 个叉号 ($1 \leq k \leq 440$) 对应着 440 个数值中的第 k 小值 val_k ，叉号的横坐标 x_k 等于对应的数值 val_k ，而叉号的纵坐标 y_k 等于：440 个服从正态分布 $N(0, \sigma^2)$ 的数值中，第 k 小值的期望；其中的 σ 是待定的参数。可以证明 y_k 满足 $R_{\sigma^2}(y_k) = \frac{k}{440+1}$ ，于是由这一关系可以求出 y_k 。

如果这 440 个数值服从 $N(0, \sigma^2)$ 的话，容易想到应该有 $x_k \approx y_k$ ，也就是说所有叉号落在直线 $y = x$ 附近。我们通过选取合适的 $\sigma > 0$ 来让叉号尽可能贴近直线 $y = x$ ，最终的结果就是图 3。可以看到，叉号与直线紧密贴合，所以这些数据确实服从正态分布¹⁶。又由假设 2.13，这一规律对任何 COI/IOI 赛制的比赛都成立。

¹⁵误差函数 erf 没有闭合形式，这个式子可以视为 erf 函数的定义式

¹⁶注意到，缩放坐标轴和改变 σ 的取值，这两种操作对图像的改变其实是完全相同的，所以缩放坐标轴不会影响结论的可靠性

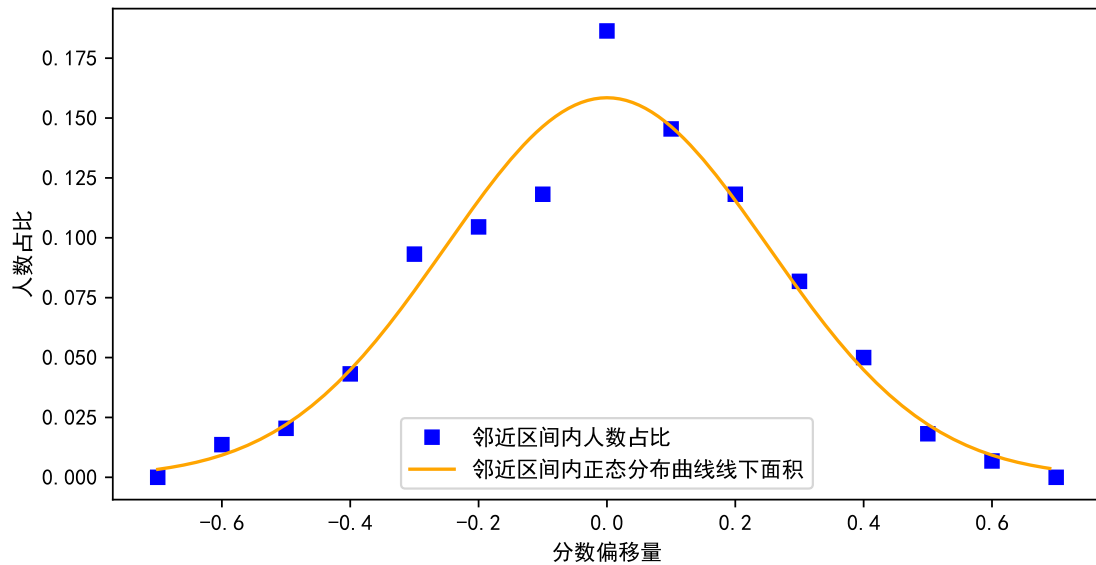


图 2: 散点图和拟合结果

定理 3.2 (偏移量分布的形式). 对任何基于 *COI* 或 *IOI* 赛制的现实比赛, 其对应的理想比赛的偏移量分布是期望值为 0 的正态分布。

4 选手整体水平的估计

本节将借助联赛初赛 (字母标号 *a*) 和联赛复赛 (字母标号 *b*) 的分数数据, 来估计全国信息学竞赛选手整体水平的分布情况。

选手的“水平”是一个模糊的概念; 为了将其量化, 我们将用一名选手在联赛复赛中的期望分数来衡量这名选手的水平。

虽然由假设 2.16, 不同年份的联赛复赛 (所对应的理想比赛) 是缩放等价的; 但它们毕竟不相同, 因此“在联赛复赛中的期望分数”这一概念需要澄清。4.1 小节将处理这一问题, 并完成对复赛分数数据的初步分析。接着, 4.2 小节将从分数数据中, 得到复赛在去除了初赛的筛选所带来的影响后, 其 (对应的理想比赛的) 分数分布函数的表达式。最后, 4.3 小节将从分数分布计算出对应的期望值分布, 这一分布即可体现全国选手整体水平的分布情况。

本节中会多次对现实情况作近似、作假设, 于是也会不可避免地带来可观的误差。因此, 本节的目标旨在估计而非精准计算, 所得的结果仅能反映趋势而不保证精确。

4.1 复赛分数数据的获取、加工和拟合

数据来自以下 6 场比赛:

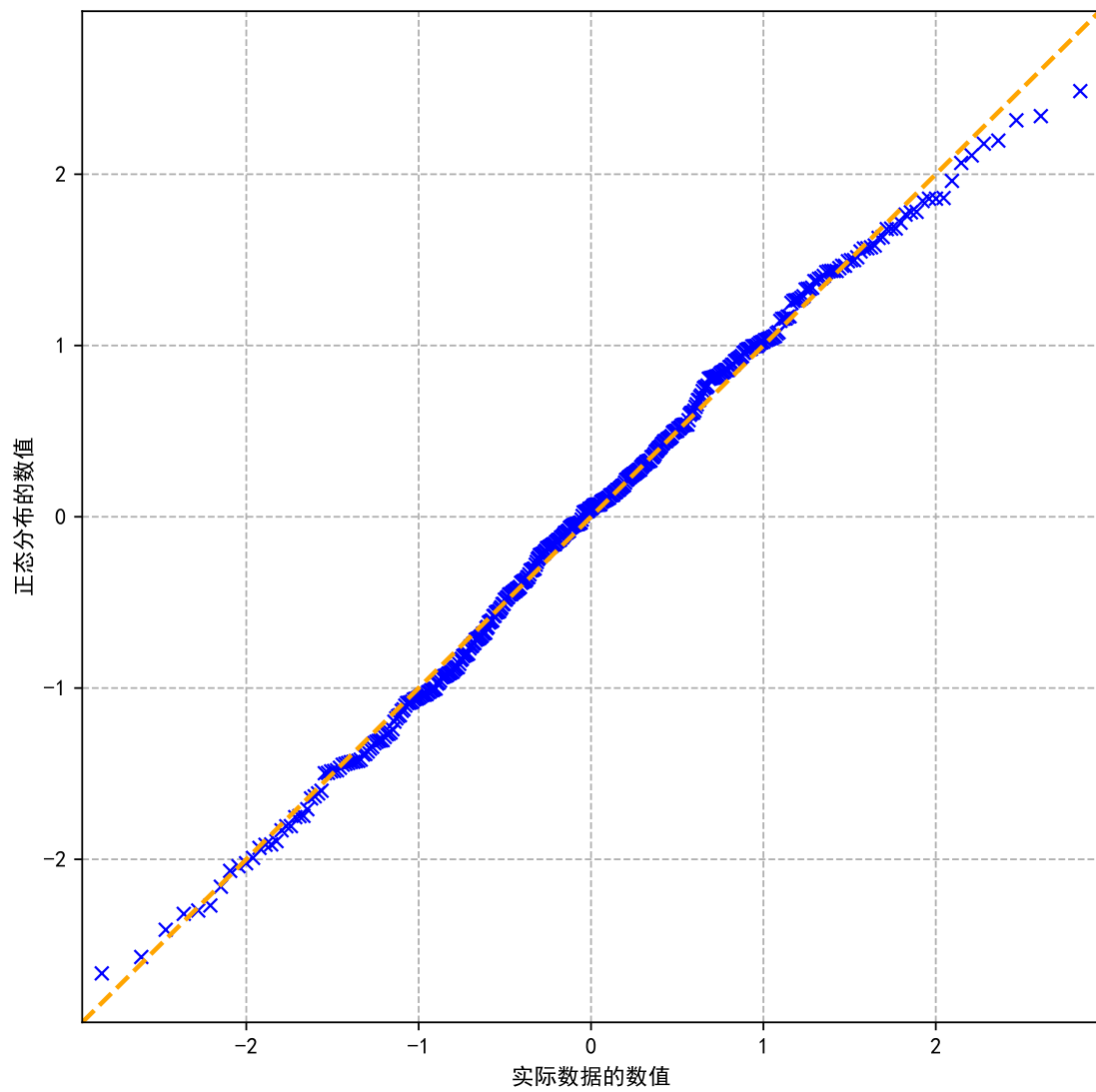


图 3: Q-Q 图，关于比赛分数差值数据和正态分布绘制

	参赛人数	获奖人数	满分	最高分	获奖分数线
NOIP2016 复赛	约 8300	约 5900	600	600	100
NOIP2017 复赛	约 10300	约 6600	600	600	80
NOIP2018 复赛	约 12900	约 8000	600	600	120
CSP2019 复赛	约 13900	约 8800	600	600	80
CSP2020 复赛	约 11000	约 6700	400	400	30
NOIP2020 复赛	约 4700	约 3600	400	400	60

表 3: 6 场比赛的相关数据

- NOIP2016 — 复赛 (字母标号 b)
- NOIP2017 — 复赛 (字母标号 b)
- NOIP2018 — 复赛 (字母标号 b)
- CSP2019 — 复赛 (字母标号 b)
- CSP2020 — 复赛 (字母标号 b)
- NOIP2020 — 复赛 (字母标号 b)¹⁷

之所以只采用 2016 年及以后的比赛, 是出于两个原因:

- 年代过于久远的比赛对当今的参考意义有限。
- 仅有的数据来源为 NOI 官网上的获奖名单公示, 故只能取得获奖选手的分数信息。而自 2016 年起 CCF 更改了获奖规则, 增加了获奖人数, 使得可以获取的数据量大了许多。

表 3 展示了关于这 6 场比赛的几项统计数据。

由假设 2.16, 这 6 场现实比赛 (所对应的理想比赛) 是缩放等价的。进而由命题 3.1, 这 6 场现实比赛所对应的理想比赛, 在对分数作变换 (变换方式见 3.2 小节) 后, 将成为完全相同的理想比赛。

于是, 与 3.2 小节类似, 数据加工将按以下步骤进行:

步骤 1. 将所有比赛中所有选手的分数除以当场比赛的最高分 (也就是计算标准分; 注意到最高分等于满分), 用以代替原始分数。

步骤 2. 去除所有 < 0.2 的分数。这是因为在这 6 场比赛中, 获奖分数线与最高分的商的最大值为 0.2; 这意味着分数低于 0.2 的选手中有一部分未能获奖, 于是这些选手中其余部分的数据也失去意义, 因此一并剔除。

步骤 3. 对每场比赛计算分数标准差 σ , 然后对分数作变换 $R: x \mapsto 1 - \frac{1-x}{5.52\sigma}$, 并用变换结果代替原分数。使用系数 5.52 的理由稍后说明。

步骤 4. 对每场比赛计算最低分, 取所得的 6 个最低分的最大值 T , 并剔除所有 $< T$ 的分数。这一步的理由与步骤 2 中的类似: 分数低于 T 的部分选手未能获奖, 故将这些选手连同已获奖的那些一并剔除。计算可得 $T \approx 0.200$, 与步骤 2 中的阈值保持一致; 这正是系数

¹⁷2020 年情况较为特殊, NOIP 与 CSP 同时存在, 且 NOIP 初赛与 CSP 初赛合并为同一场比赛

5.52 的主要作用。

步骤 5. 现在所有这些分数数据属于同一理想比赛 A (满足 A 与原先 6 个现实比赛所对应的理想比赛缩放等价), 将它们汇集起来即可。注意到我们所取得的并非完整的分数数据, 而只是 ≥ 0.2 的那一部分分数。

本节中我们约定使用理想比赛 A 作为衡量选手水平的标尺, 也就是说我们将用一名选手在 A 中的期望分数, 来代表该名选手的水平。后文中如果作为一个现实比赛提到“联赛复赛”, 则默认指 A 对应的现实比赛。

经过上述加工后, 我们得到了 26907 个落在 $[0.2, 1]$ 之中的分数数据。由命题 2.5, 这些数据应当服从 C_A 所描述的概率分布。

受数据限制, 我们暂时只关心 $[0.2, 1]$ 这一分数区间上的分布情况。即我们要确定函数 $F: [0.2, 1] \rightarrow \mathbb{R}_{\geq 0}$, 满足在任何一个区间 $[a, b]$ 上, F 的定积分在数值上约等于: 落在 $[a, b]$ 中的分数数值个数, 与数值总个数 26907 的比值。注意到在 F 和 C_A 之间有这一关系:

$$F(s) = C_A(s) / \int_{0.2}^1 C_A(t) dt, \quad \forall s \in [0.2, 1]$$

观察分数数据, 可以发现数值的分布左侧 (即靠近 0.2 一侧) 稠密、右侧 (即靠近 1 一侧) 稀疏, 于是尝试用递降的多项式函数、对数函数等常见函数, 以及截尾指数分布、截尾正态分布等“向左倾斜”的常见概率分布来拟合之。实验发现使用对数函数的拟合效果大幅好于其他方式, 以下将描述对数函数的拟合过程和结果。

步骤 1. 对于 $t = 0.25, 0.30, \dots, 0.95$, 计算: 落在 $[t - 0.05, t + 0.05]$ 中的数值个数与总个数 26907 的比值。这个比值记为 $c(t)$ 。

步骤 2. 在平面直角坐标系中画出 $t - c(t)$ 散点图。

步骤 3. 选取参数 α 并令 $F(x) = \alpha \log(x)$ 。注意到函数 F 在 $[0.2, 1]$ 上的定积分必须等于 1, 由此可唯一确定 α 的取值; 计算得到 $\alpha \approx -2.09156$ 。

步骤 4. 检查函数

$$G(t) = \int_{t-0.05}^{t+0.05} F(x) dx$$

的图像是否与 $t - c(t)$ 散点图吻合。

图 4 展示了拟合的结果。可以看到全部数据点与曲线贴合紧密, 且算得残差平方和约为 $3.3 \cdot 10^{-4}$, 显示出了较好的拟合效果。基于此, 我们确定取 $F(x) = \alpha \log(x)$, 其中 $\alpha \approx -2.09156$ 。

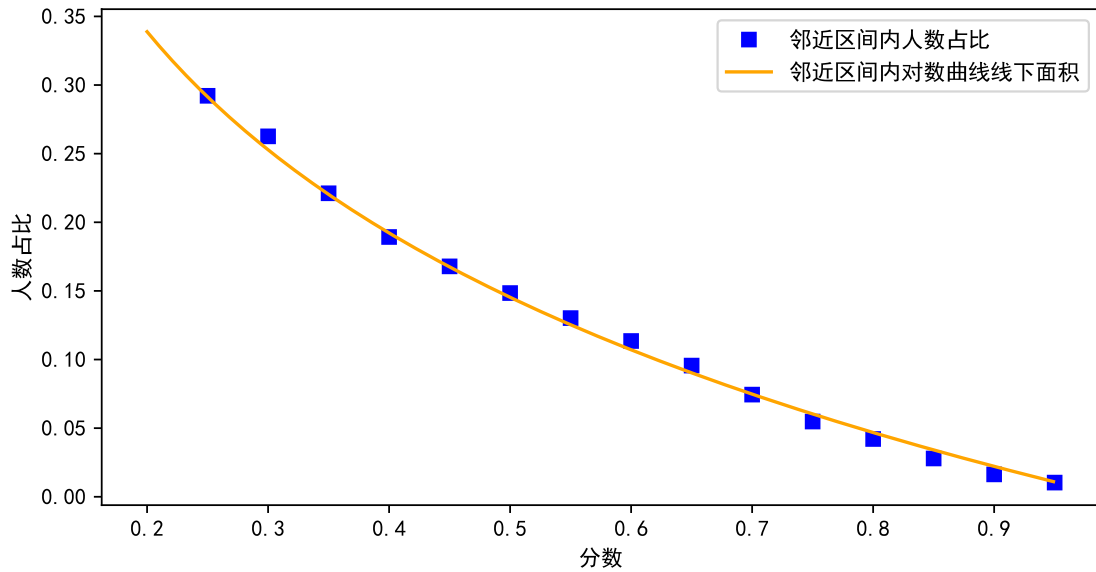


图 4: 散点图和拟合结果

我们推测在区间 $[0, 0.2)$ 上, 分数的分布依旧近似地符合对数曲线的形态。假定如此, 则分数区间 $[0.2, 1]$ 上的人数占总人数的比例应该等于

$$\frac{\int_{0.2}^1 -\log(x)dx}{\int_0^1 -\log(x)dx} \approx 0.478$$

进而根据分数区间 $[0.2, 1]$ 上的总人数为 26907, 可推算出 (前述 6 场比赛的) 参赛总人数约为 56300。该值与实际的复赛报名人数¹⁸61100 接近, 于是我们的假设得到验证。

由此可知 A 的分数分布函数的表达式同样形如 $C_A(s) = \beta \log(s)$, 其中系数 β 可由等式

$$\int_0^1 C_A(s)ds = 1$$

唯一确定; 计算得 $\beta = -1$ 。

命题 4.1. $C_A(s) = -\log(s)$, 其中 $s \in (0, 1]$ 且 A 是根据 4.1 小节中描述的过程所确定的理想比赛。

¹⁸这之中包括数千名缺考选手, 故真正参赛的人数会低于 61100

4.2 结合初赛的分析

这一小节将对于复赛所对应的理想比赛 A ，在去除了初赛的筛选性所带来的影响后，计算所得的新的理想比赛（记为 A' ）的分数分布函数。其中，4.2.1 小节将给出初赛分数数据的来源，4.2.2 小节将结合这些数据给出关于初赛的几个假设；第 4 节的其余部分都将依赖于这些假设。4.2.3 小节将计算初复赛（所对应理想比赛）的偏移量分布的标准差，以为 4.2.4 小节中 $C_{A'}$ 的计算做好准备。

4.2.1 初赛分数数据的获取

分数数据采用 NOIP2018 初赛北京赛区的成绩。该场比赛共 781 人获得非零分数（零分视为缺考），其中 536 人晋级复赛并获得非零分数；该场比赛满分 100 分，最高分 96 分，晋级分数线约为 35 分；全国最高分为 100 分。有关初赛的全部数据获取自官方网站上的成绩公示。

采用该场比赛的原因：后续分析需要分数表上包含选手姓名；而笔者所能找到的其他年份、其他省市的成绩公示，均未包含这一信息。

知道了选手姓名，我们就可以查询该名选手在 NOIP2018 复赛中的得分。通过这种方式，我们获得了 536 名晋级者的初赛和复赛分数。由于官方网站上的成绩公示仅包括获奖选手，这里所使用的北京选手复赛分数是按民间数据测试出的成绩。

除此以外，在 4.2.4 小节中，还将使用 CSP2019 初赛的全国分数数据，这些数据从官方网站上各省市发布的成绩公示汇总得到。CSP2019 初赛报名人数 48812 人，由于个别省份仅公示了晋级选手或未缺考选手的分数，最终收集到 47264 人的数据。

全国分数数据的来源之所以采用 CSP2019，是因为自 2019 年起才有完整的初赛分数公示。

4.2.2 关于初赛的几个假设

不同于复赛，信息学联赛的初赛是分省考试、分省排名的，这会给本文的分析带来很大困难。为了规避这一问题，我们作如下假设：

假设 4.2. 每一年的联赛初赛为全国统一考试、统一排名，全国范围内分数最高的若干名选手晋级复赛。

作出这一假设，意味着忽略不同省市间选手水平和竞争激烈程度上的差异，并用全国整体的选手水平和竞争激烈程度来代替之。即使如此，我们一般也并不能直接用一个地区的数据来“代表”全国的数据，而是只有在所研究的量与地域没有明显关联时（例如 4.2.3 小节中研究同一名选手的初赛得分与复赛得分间的关系）才能这样做。

在给出下一个假设前，先对 2018 北京初赛的分数做一点分析。

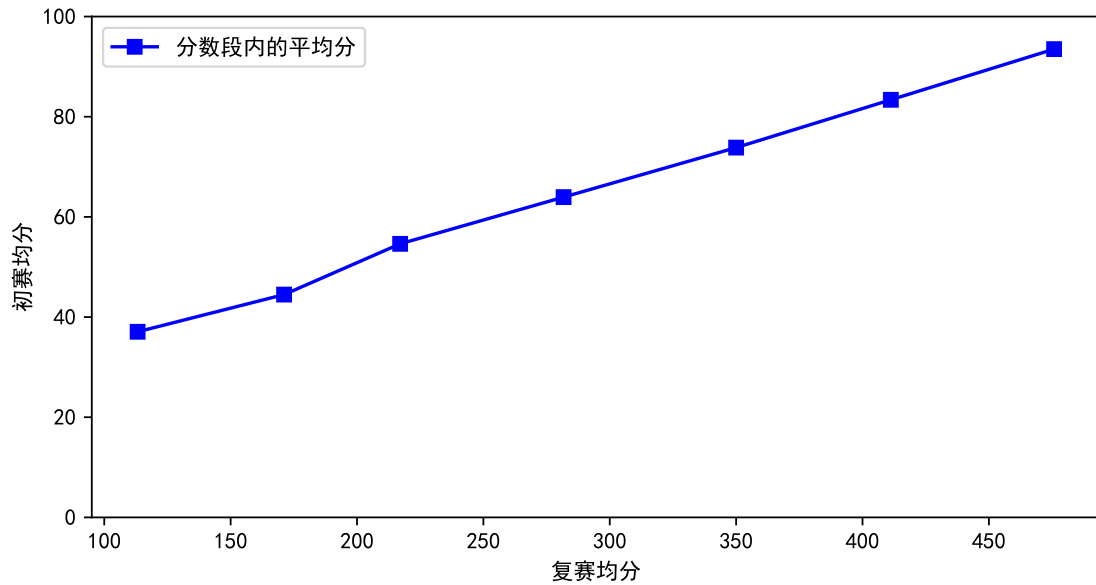


图 5: 平均分折线图

步骤 1. 将 536 名晋级选手按初赛分数分组：分数在 $[30, 40)$ 中的、在 $[40, 50)$ 中的、……、在 $[90, 100)$ 中的，分别分为一组，共计 7 组。

步骤 2. 对每一组计算初赛平均分和复赛平均分。

步骤 3. 对每一组，以复赛平均分为横坐标、初赛平均分为纵坐标，将数据点画在二维平面上，并将这些数据点连成折线图。

所得的折线图如图 5 所示。可以看到，这些数据点近似地连成一条直线；这提示我们，初赛分数与复赛分数之间存在一个线性的对应关系。

基于这一观察，我们作出如下假设：

假设 4.3. 记现实比赛 B_1 为联赛初赛，取参赛人群为实际晋级复赛的全体选手；记现实比赛 B_2 为联赛复赛；则 B_1, B_2 对应的理想比赛缩放等价。

注意：“取参赛人群为实际晋级复赛的全体选手”这一规定，只限制了参赛人群，而并未要求这些选手在理想比赛中的分数也一定得达到晋级的标准。也就是说，虽然我们只取那些在现实中达到了晋级分数线的选手，但在构造对应的理想比赛时，我们忽略现实中发生了什么，仍然只考察每名选手分数波动的概率分布和他的期望分数。

关于这一假设需要作几点说明：

1. 2.3 小节中，我们在为假设 2.16 予以辩护时，断言了“一名选手的水平是固定的，不会随比赛的改变而改变”。但是，由于考察内容的不同，一名选手在 B_1 和 B_2 中的能力差异可能较大，故上述断言在将初赛（即 B_1 ）加入考虑范围后似乎不再成立。

2. 为了使前述断言仍然成立, 在 4.2 小节内, 我们需暂时改变命题 2.3 中“期望得分”这一概念的所指, 将其改为: 一名选手在 B_1, B_2 中 (在按最高分和标准差折算后) 期望分数的平均值。这会改变从现实比赛构造理想比赛的方式, 并使得 B_1, B_2 所对应理想比赛的期望值分布和偏移量分布发生变化, 变化后 B_1, B_2 所对应的理想比赛分别记作 F_1, F_2 (所以 A 和 F_2 的区别, 就是概念更改前和更改后的区别)。显然, 此时 F_1, F_2 的期望值分布在经过缩放后是相同的。

3. 这样更改后, Δ_{B_2} 的值也发生了变化。原先 Δ_{B_2} 的取值等于选手实际表现与真实能力 (即期望表现) 的差; 现在它的值还要在此基础上加上选手在复赛单项上的能力与初、复赛综合能力的差。但是, 只要“单项能力减综合能力”这一随机变量服从正态分布, 新的 Δ_{B_2} 也一定服从正态分布——因为服从正态分布的独立随机变量之和依然服从正态分布。另一方面, 假如在原先定义下的随机变量 Δ_{B_1} 服从正态分布, 则对新的 Δ_{B_1} 可做与刚才类似的论证。

4. 关于 F_1, F_2 间的缩放等价性, 第 2 条对期望值分布的缩放等价予以了说明, 第 3 条对偏移量分布的缩放等价予以了说明; 这些说明都有一些感性的成分, 它们仅用作对假设 4.3 含义的澄清, 而并非尝试对其予以证明。需要注意, 由于我们对概念的修改, 关于 F_1, F_2 的期望值分布、偏移量分布所作的一切讨论, 在 4.2 小节之外均没有意义。但是 F_1, F_2 的分数分布不会受这一修改的影响, 故 4.2 小节计算出的分数分布函数会在后续分析中直接使用。

4.2.3 计算偏移量分布的参数

先前已经说明, 理想比赛 F_2 的偏移量分布为正态分布 $N(0, \sigma^2)$ 。这一小节将基于 4.2.1 小节中获得的数据, 来测量该分布的标准差 σ 。

引理 4.4. 独立随机变量 X_1, X_2 分别服从分布 $N(0, \sigma_1^2), N(0, \sigma_2^2)$, 则 $X_1 + X_2$ 服从分布 $N(0, \sigma_1^2 + \sigma_2^2)$ 。

证明见维基百科相应条目 [1], 这里不再重复。

由假设 4.3, 在对 F_1 作线性的缩放变换 T 之后, 可以使其与 F_2 相同; 此时两者的偏移量分布均为 $N(0, \sigma^2)$ 。又注意到对现实比赛 B_1, B_2 , 其对应的随机变量 $\Delta_{B_1}, \Delta_{B_2}$ 应当是独立的 (这里认为在定义 2.3 的步骤 3 中 B_1, B_2 共用同一个表示选手的随机变量 p), 所以由引理 4.4, $\Delta_{T[B_1]} + \Delta_{B_2}$ 服从正态分布 $N(0, 2\sigma^2)$ 。因此, 选手在 $T[F_1]$ 和 F_2 中的分数之差, 这一随机变量服从标准差为 $\sqrt{2}\sigma$ 的正态分布; 只要测出它的标准差, 即可得到 σ 的取值。

容易想到以下测量方式:

步骤 1. 对 F_1 的分数作线性变换, 使得变换后它的期望值、偏移量分布与 F_2 相同。

步骤 2. 对先前提到的 536 名选手, 计算每名选手在 F_2 中的分数和在变换后的 F_1 中的分数之差。

步骤 3. 这 536 个差值应该服从正态分布, 那么计算这些值的标准差即可。

但一个问题是，这 536 个差值并非真正服从正态分布。如果一名选手考出了大幅低于自己期望分数的分数，那么他进入这 536 人之列的机会就会大大降低；换句话说，这 536 个数据的取样方法是有选择性的，而且选择的方式倾向于实际分数高于期望分数的选手，因此这些数据不能代表整体的分布。

假如我们召集那些没有晋级的选手，让他们也参加复赛考试并记录他们的分数，再把这些分数和原有的 536 个数据汇总，就能获得完整、有代表性的数据。但实际上，我们也可以“假装”已经获得了未晋级选手的数据，并对全体数据进行分析；如果这个过程中“碰巧”没有用到任何一个未晋级选手的数据，我们事实上就在只凭借已有的 536 个数据的情况下完成了测量。以下给出一个这样的测量方式。

步骤 1. 对 F_1 的分数作线性变换，使得变换后它的期望值、偏移量分布与 F_2 相同。

步骤 2. 对先前提到的 536 名选手，计算每名选手在变换后的 F_1 中的分数和在 F_2 中的分数之差。（前者减后者）

步骤 3. 取出这些差值中前 35 大的值，则这些值可以视为：某一组服从正态分布的 781 个数（781 即初赛参赛人数），其中前 35 大的值。通过测量这些值可以得到正态分布的标准差。

在第 3 步中，之所以说这 35 个值为 781 个数中的最大值，是因为：

- 计算发现第 35 大的差值约等于 0.26，高于初赛晋级分数线经过变换后的值。又因为复赛分数不可能小于 0，所以任何一个未达到晋级分数线的选手，其两试分数差值不可能达到 0.26。（计算发现 35 人中最低的初赛分数为 53 分，比晋级分数线高出近 20 分）

- 因此，除了 536 名晋级选手之外，其余选手不可能进入 35 人之列，故只考虑已有的 536 个数据是充分的。

在前述过程的步骤 1 中需要作分数变换，以下给出具体步骤。

步骤 1. 对于复赛分数 $s \in [120, 600]$ （120 为官方分数公示所覆盖的最低分数），计算该分数在全国范围内的排名 c ，并将 s 映射到 $t \in [0, 1]$ ，满足

$$\frac{\int_t^1 -\log(x)dx}{\int_{0.186}^1 -\log(x)dx} = \frac{c}{N}$$

其中 $N = 8044$ 为复赛不低于 120 分的选手总数，0.186 为最低分数 120 依 4.1 小节中的变换 R 映射到的值。

步骤 2. 对于低于 120 分的复赛分数，我们将 $[0, 120)$ 均匀地映射到 $[0, 0.186)$ 上去。以上两个步骤所描述的映射方式，保证了分数分布呈对数曲线，与命题 4.1 一致。

步骤 3. 计算北京初赛排名前 25% 选手（共 195 名）的分数标准差 σ_1 ，再对映射后的复赛分数计算北京选手前 195 名的分数标准差 σ_2 。然后对于初赛分数 $s \in [0, 100]$ ，将其映射到 $1 - (1 - \frac{s}{100}) \cdot \sigma_2 / \frac{\sigma_1}{100}$ 。由命题 3.1，这一映射方式保证了映射后两个理想比赛相同。这一步中只取前 25% 的理由：对于初赛期望分数离晋级分数线较近的选手，这些选手中有相

当一部分未能进入复赛，故复赛在相应分数段的分布会比真实情况稀疏；只有把考察范围限制在分数足够高的选手，才能避免这一问题。

设得到的 35 个差值按降序排列为 d_1, \dots, d_{35} ，考虑如何由此推断全体差值的标准差 $\sqrt{2}\sigma$ 。

这里采用最大似然估计，即选取一个 σ 以最大化：在全体差值服从 $N(0, 2\sigma^2)$ 的条件下，测量得到 d_1, \dots, d_{35} 的概率。注意到这个概率实际上必定等于 0，但可以通过取极限规避这一问题。下式给出 σ 的计算方式，其中 u_1, \dots, u_{781} 表示随意排列的 781 个差值， v_k 表示 u_1, \dots, u_{781} 中的第 k 大值。

$$\begin{aligned}
& \lim_{\epsilon \rightarrow 0} \arg \max_{\sigma \in \mathbb{R}_{>0}} \Pr [v_k \in [d_k - \epsilon, d_k + \epsilon], \forall 1 \leq k \leq 35] \\
&= \lim_{\epsilon \rightarrow 0} \arg \max_{\sigma \in \mathbb{R}_{>0}} \binom{781}{35} \cdot 35! \cdot \prod_{k=1}^{35} (R_{2\sigma^2}(d_k + \epsilon) - R_{2\sigma^2}(d_k - \epsilon)) \cdot R_{2\sigma^2}(d_{35})^{746} \\
&= \lim_{\epsilon \rightarrow 0} \arg \max_{\sigma \in \mathbb{R}_{>0}} (2\epsilon)^{-35} \cdot \prod_{k=1}^{35} (R_{2\sigma^2}(d_k + \epsilon) - R_{2\sigma^2}(d_k - \epsilon)) \cdot R_{2\sigma^2}(d_{35})^{746} \\
&= \lim_{\epsilon \rightarrow 0} \arg \max_{\sigma \in \mathbb{R}_{>0}} \prod_{k=1}^{35} \frac{R_{2\sigma^2}(d_k + \epsilon) - R_{2\sigma^2}(d_k - \epsilon)}{2\epsilon} \cdot R_{2\sigma^2}(d_{35})^{746} \\
&= \arg \max_{\sigma \in \mathbb{R}_{>0}} \left(\prod_{k=1}^{35} R'_{2\sigma^2}(d_k) \right) \cdot R_{2\sigma^2}(d_{35})^{746} \\
&= \arg \max_{\sigma \in \mathbb{R}_{>0}} 746 \cdot \log(R_{2\sigma^2}(d_{35})) + \sum_{k=1}^{35} \log(P_{2\sigma^2}(d_k))
\end{aligned}$$

至此，问题完全转化为一个最优化问题。最优化方法采用 SciPy 提供的 BFGS 算法的实现 [2]，算得 $\sigma \approx 0.109$ 。

图 6 中的橙色曲线展示了差值的概率分布，35 条蓝色竖线表示实际测得最大的 35 个差值在其中的位置。

命题 4.5.

$$\Delta_{T[F_1]}(\delta) = \Delta_{F_2}(\delta) = P_{\sigma_F^2}(\delta), \quad \forall \delta \in \mathbb{R}$$

其中“缩放变换” T 满足 $T[F_1] = F_2$ ，常数 σ_F 约等于 0.109。

4.2.4 消除初赛对分数分布的影响

这一小节将计算初赛分数线映射为复赛分数后的值，并借助该值计算出：消除初赛的筛选性带来的影响后（即假想初赛并未淘汰一人，所有选手都晋级复赛），复赛的分数的分布函数。

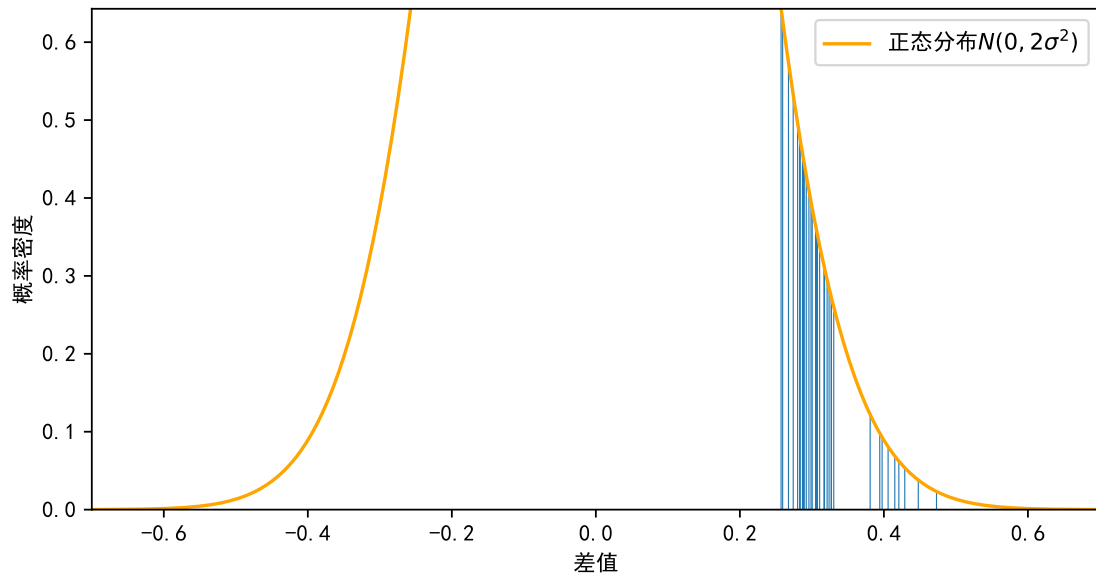


图 6: 差值的分布情况

在 4.2.1 小节中, 已经得到了 CSP2019 初赛参赛选手的分数数据。由表 3 中的数据知, 在 2016 到 2020 五年中, 平均每年的初赛晋级人数约为 11280; 因此我们选择 CSP2019 初赛全国第 11280 名的分数, 作为假设 4.2 中的“全国统一晋级分数线”。

这里之所以对晋级人数而不是晋级率取平均数, 是因为初赛的参赛选手总数受收费、政策等无关因素影响过大, 而晋级复赛的人数与复赛获奖的人数呈固定比例, 因而相对可靠。

最终算得分数线为 63.5 分。作为参照, CSP2019 初赛中, 浙江、山东、江苏实际的分数线¹⁹分别为 72.5, 60, 53。

下面将这一分数线映射为复赛分数。为了与 4.2.3 小节保持一致, 这里仍然使用同样的计算方式, 并同样采用北京的数据。

我们将计算 CSP2019 初赛中, 北京排名前 195 名的分数标准差 σ_1 , 再对 (按 4.2.3 小节中的方式) 映射后的 NOIP2018 复赛分数计算北京选手前 195 名的分数标准差 σ_2 。然后对于 CSP2019 初赛分数 $s \in [0, 100]$, 将其映射到 $1 - (1 - \frac{s}{100}) \cdot \sigma_2 / \frac{\sigma_1}{100}$ 。这一计算过程基于如下假设: 2019 年北京选手整体水平, 与 2018 年大体相同。

对分数线 63.5 施加上述变换, 得到其对应的复赛分数为 $h \approx 0.1156$ 。另一方面, 由命题 4.5 可知, 任何一名选手的初赛、复赛的 (变换后) 实际分数之差服从概率分布 $N(0, 2\sigma_F^2)$ 。从而, 如果假想所有初赛选手都参加了复赛, 则对于复赛实际分数为 t 的选手 p , 其初赛分数达到 63.5 的概率为 $1 - R_{2\sigma_F^2}(h - t)$ 。

需要注意, 这样得到的概率, 是在获得具体的期望值分布前的先验概率。假如已知全

¹⁹这里给出的是全省分数线, 省内各市的分数线可能高于全省分数线

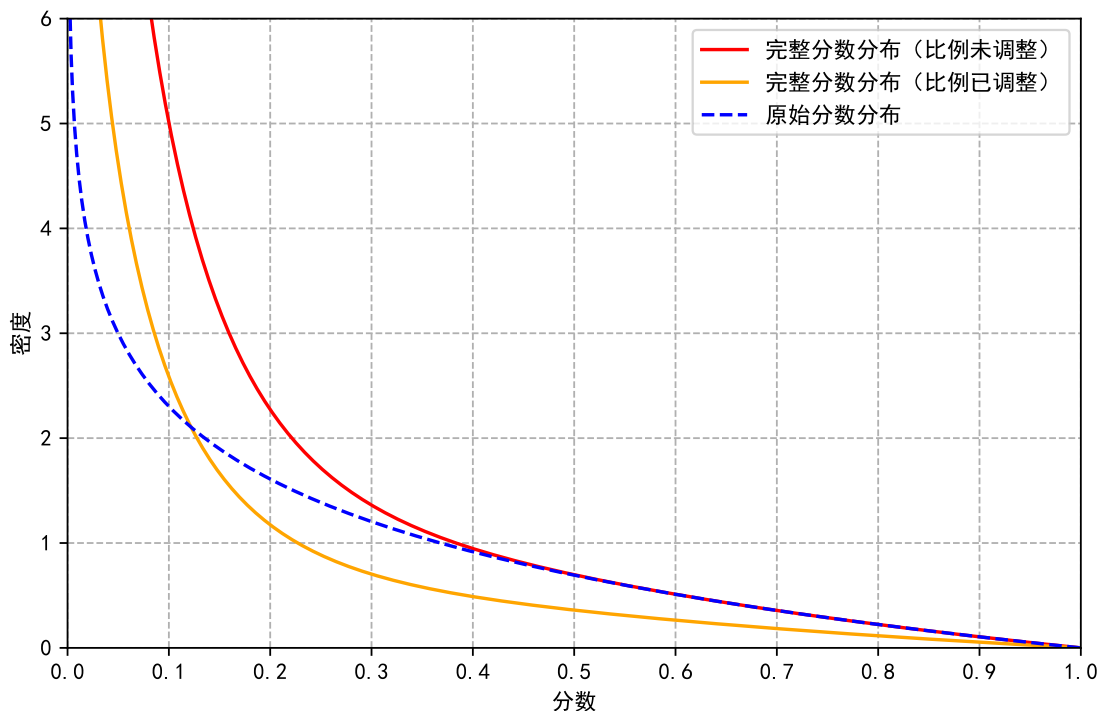


图 7: 消除初赛影响前后的分数分布函数

体选手的期望分数分布情况，我们可以用贝叶斯公式得到前述选手 p 的期望分数取每一个值的概率，进而得到 p 的初赛分数取每一个值的概率，也就是后验概率。简便起见这里采用先验概率，即使它相比后验概率略失精确。

记 F'_2 为现实比赛 B_2 在消除初赛的筛选性带来的影响后所对应的理想比赛，则由以上讨论可得：

$$C_{F'_2}(s) \propto \frac{C_{F_2}(s)}{1 - R_{2\sigma_F^2}(h-s)} = \frac{-\log(s)}{1 - R_{2\sigma_F^2}(h-s)}, \quad \forall s \in (0, 1]$$

上式中之所以使用“正比于”而不是“等于”，是因为分数分布函数表达的是分布“密度”，而不是样本“数量”。计算出对应的比例系数后得到：

$$C_{F'_2}(s) = \gamma \frac{-\log(s)}{1 - R_{2\sigma_F^2}(h-s)}, \quad \forall s \in (0, 1]$$

其中常数 $\gamma \approx 0.516$ ，它使得 $C_{F'_2}$ 在 $[0, 1]$ 上的定积分等于 1。

图 7 分别展示了以下三个函数的图像：

蓝色 $f(s) = -\log(s)$ ，即 $C_{F_2}(s)$ 或 $C_A(s)$ 。

红色 $f(s) = \frac{-\log(s)}{1 - R_{2\sigma_F^2}(h-s)}$

橙色 $f(s) = \gamma \frac{-\log(s)}{1 - R_{2\sigma_F^2}(h-s)}$ ，即 $C_{F'_2}(s)$ 。

由 4.2.2 小节中的讨论知, C_A 与 C_{F_2} 相同。同理, 如果定义 A' 为: 复赛在去除初赛影响后对应的理想比赛 (这里采用按原本的方式解读的定义 2.3), 则 $C_{A'}$ 亦与 C_{F_2} 相同。因此有以下命题:

命题 4.6.

$$C_{A'}(s) = \gamma \frac{-\log(s)}{1 - R_{2\sigma^2}(h-s)}, \quad \forall s \in (0, 1]$$

其中常数 $\gamma \approx 0.516$, 且 A' 为复赛在去除初赛影响后对应的理想比赛。

4.3 从分数分布还原期望值分布

在 4.2.4 小节中得到了 $C_{A'}(s)$ 的表达式; 这一小节将由此计算 $X_{A'}(x)$ 。

由假设 2.12, A' 是简单理想比赛; 又根据命题 2.10, 可从 $X_{A'}$ 和 $\Delta_{A'}$ 计算出 $C_{A'}$ 。这一小节将给出从 $C_{A'}$ 和 $\Delta_{A'}$ 逆推出 $X_{A'}$ 的方法。

根据定理 3.2, 存在 $\sigma > 0$ 使得

$$\Delta_{A'}(\delta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\delta^2}{2\sigma^2}\right), \quad \forall \delta \in \mathbb{R}$$

进而由命题 2.10 得

$$C_{A'}(s) = \frac{1}{\sqrt{2\pi}\sigma} \int_0^1 X_{A'}(x) \exp\left(-\frac{(s-x)^2}{2\sigma^2}\right) dx, \quad \forall s \in \mathbb{R} \quad (5)$$

在进行逆推之前, 先测量 σ 的值。我们获取了 CSP2019 复赛全体选手的民间分数, 并按以下步骤进行测量:

步骤 1. 将每名选手每一天的分数除以当天最高分 (两天最高分均为满分 300 分), 再将每一天的所有分数做变换, 以使得两天的分数分布分别呈对数曲线状。

步骤 2. 对每名选手计算两天分数之差, 计算所有这些差值的标准差 σ_0 。

与 4.2.3 小节中类似, 同一名选手的单日分数 (变换后的分数), 应该服从标准差为 $\sigma_1 = \frac{\sigma_0}{\sqrt{2}}$ 的正态分布。又因为一名选手在的总分 (变换后) 等于两天分数的平均值, 由引理 4.4 可以得到: 同一名选手在 CSP2019 中 (变换后) 的分数服从标准差为 $\sigma = \frac{\sigma_0}{\sqrt{2}}$ 的正态分布, 也就是说 $\sigma \frac{\sigma_0}{2}$ 。

最终算得 $\sigma \approx 0.078$ 。

从 $C_{A'}$ 和 $\Delta_{A'}$ 逆推出 $X_{A'}$ 难以精确地实现, 因此这里只能近似地计算 $X_{A'}$ 在许多个离散的点处的点值。

我们将区间 $(0, 1]$ 作 500 等分, 并设立 500 个未知数 $x_{1 \dots 500}$, 分别表示在 500 个分点处 $X_{A'}$ 的取值。为了得到等式, 我们将 5 中的定积分换成离散的求和, 并将 $x_{1 \dots 500}$ 代入。在作

了这样的近似之后，等式显然不再成立，因此我们改为了最小化所有这些等式两端之差的平方和。为了避免无意义的结果，我们额外加入了序列 $x_{1\dots 500}$ “光滑性”的限制。

上述问题最终归约到了一个二次规划模型的求解。最终算得的点值不宜在此处直接列举，可以在本文开头的链接中找到。

5 致谢

感谢中国计算机学会提供交流和学习的平台；

感谢国家集训队高闻远教练的指导；

感谢老师、教练们对我的指导；

感谢任清宇等同学与我讨论本文内容。

参考文献

- [1] Wikipedia: Sum of normally distributed random variables,
https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables
- [2] SciPy Documentation: `scipy.optimize.minimize`,
<https://docs.scipy.org/doc/scipy/reference/optimize.minimize-bfgs.html#optimize-minimize-bfgs>

浅谈亚 \log 数据结构在 OI 中的应用

宁波市镇海中学 钱易

摘要

数据结构是 OI 中的一类常考问题，本文介绍了压位 trie 与 vEB tree (van Emde Boas tree) 两种数据结构并且展示了其在部分问题中的应用。

引言

在 OI 中，数据结构一直是一个重要考点，而本文主要介绍了用来解决 Dynamic Predecessor Problem 的两种亚 \log 数据结构，目前在学术界的研究已经存在了一些比较优秀的算法，如 y-fast trie¹ 可以以 $O(n \log_2 \log_2 V)$ 的时间复杂度， $O(n)$ 的空间复杂度解决该问题，Fusion tree² 可以以 $O(n(\log_w n + \log_2 w))$ 的时间复杂度， $O(n)$ 的空间复杂度解决该问题。但是，这两种数据结构的常数因子非常大，且难以实现，往往不会在 OI 使用。这类问题大多数情况下，值域范围往往不会特别大，因此，本文介绍了在 OI 中更加实用的两种数据结构：压位 trie 与 vEB tree (van Emde Boas tree)。

本文第一节介绍了 Dynamic Predecessor Problem 以及常用于解决这个问题的一些 \log 数据结构，第二节介绍了压位 trie，第三节介绍了 vEB tree (van Emde Boas tree)，第四节对一些数据结构的运行效率进行了比较，第五节讲解了其部分应用。

本文涉及程序的运行时间，运行时间测试环境为 64 位 Ubuntu Linux 14.04 LTS x64 操作系统，CPU 为 Intel Core i5-6500 CPU @ 3.20GHz，8 GB 内存，GCC/G++ 版本为 4.8.4，并开启 -O2 优化，测试运行多次取平均值。

1 Dynamic Predecessor Problem

写一种数据结构，来维护一些数，其中需要支持以下操作：

1. 插入 x 数（若已有 x 则不进行此操作）；
2. 删除 x 数（若 x 不存在则不进行此操作）；

¹ 详见 Wikipedia: https://en.wikipedia.org/wiki/Y-fast_trie

² 详见 Wikipedia: https://en.wikipedia.org/wiki/Fusion_tree

3. 求 x 的前趋（前趋定义为小于 x ，且最大的数，若不存在则输出 -1 ）；
 4. 求 x 的后继（后继定义为大于 x ，且最小的数，若不存在则输出 -1 ）；
- 假设操作数量为 n ，保证 $1 \leq n, x \leq 10^7$ 且其均为整数。

时间限制：1s

空间限制：32MB

1.1 常见解法

1.1.1 平衡树

这一道题目要求支持插入删除前驱后继操作，所以可以直接使用平衡树来维护。

由于不要求第 K 大或求一个数排名，因此可以直接使用 STL 中的 `set` 来实现，然而由于平衡树效率较低，且使用空间较大，难以通过此题。

1.1.2 树状数组

由于本题值域不大，因此可以直接使用树状数组维护每种数字出现次数，然后采取在树状数组上跳跃的方法来求出前驱后继，时间复杂度 $O(\log_2 V)$ ，且常数较小，然而空间复杂度为 $O(V)$ ，其中 V 为值域，难以通过此题。尽管可以通过先分为若干小块的方法来压缩空间，但是代码较大，且运行速度较慢，意义不大。

1.1.3 整型压位

如果集合中的数 x 大小不大，满足 $0 \leq x < w$ ，假设 a_i 表示数 i 是否在集合之中，因此可以使用一个整型 $s = \sum_{i=0}^{w-1} a_i \times 2^i$ 来维护集合，对于插入、删除操作，可以直接使用位运算维护。而对于查询 x 的后继，则只需要返回 $ctz(s \gg x) + x$ ， $ctz(v)$ 表示 v 中第一个 1 所在的位，如果查询 x 的前驱，而对于查询 x 的前驱，则只需要返回 $w - 1 - clz(s \& (2^x - 1))$ ， $clz(v)$ 表示 v 中从高位到低位前缀 0 的个数，一般在实现之中，可以使用 GCC 内置的 `__builtin_ctzl`，`__builtin_clzl` 函数来实现。³ 所以如果集合中的数在 $[0, w)$ 中，以上操作均可 $O(1)$ 实现。

2 压位 trie

压位 trie，即 w 叉 trie，因为其较小的代码量与较小的常数，笔者一般使用其解决 Dynamic Predecessor Problem。压位 trie 各个操作时间复杂度均为 $O(\log_w V)$ ，且具有较强的可拓展性。

³如果不允许使用 GCC 的内置函数，我们也可以做到 $O(1)$ 的时间复杂度，详见：hgztrue，《位运算：进阶技巧（上）》，<https://zhuanlan.zhihu.com/p/70950198>

2.1 压位 trie 的结构

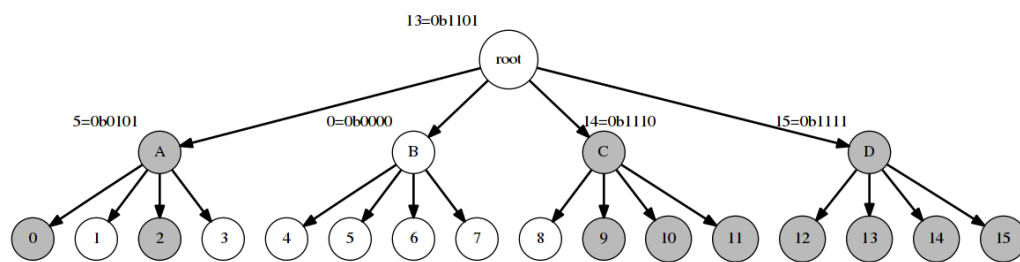
假设 $c = \log_2 w$ 。

一个大小为 2^k 的压位 trie 可以维护一个数字在 $[0, 2^k)$ 中的集合。

如果 $2^k \leq w$ ，显然可以直接使用一个整型压位来解决，因此我们接下来仅讨论 $2^k > w$ 的部分。

定义 $high(x) = \lfloor \frac{x}{2^{k-c}} \rfloor$ ， $low(x) = x \bmod 2^{k-c}$ ，即 x 的高位与低位。

对于一个大小为 2^k 的压位 trie，它将有 w 个大小为 2^{k-c} 的子压位 trie $A_i (0 \leq i < w)$ ，其中第 i 个子压位 trie 存储了高位为 i 的数的低位的集合，并且再使用一个整型 B 按位存储每个子压位 trie 中是否存在元素。



假设 $w = 4$ ，权值范围是 $[0, 16)$ ，维护的数字集合是 $\{0, 2, 9, 10, 11, 12, 13, 14, 15\}$ ，那么建出的压位 trie 如上图，其中节点旁边的数字代表其每个儿子中是否存在元素。

2.2 压位 trie 的插入操作

假设我们需要在一个大小为 2^k 的压位 trie 中插入元素 x 。

首先，必然需要将 $low(x)$ 插入到 $A_{high(x)}$ 之中，并且标记一下 $A_{high(x)}$ 中存在元素，可以发现的是，由于其每次会进入到一个大小为原来 $\frac{1}{w}$ 的子压位 trie 之中，因此其时间复杂度为 $O(\log_2 V / \log_2 w)$ ，即 $O(\log_w V)$ 。

2.3 压位 trie 的删除操作

假设我们需要在一个大小为 2^k 的压位 trie 中删除元素 x 。

首先，需要在 $A_{high(x)}$ 之中删除 $low(x)$ ，接着，可以通过 $A_{high(x)}$ 中是否存在一个非空子树来判断出其元素是否被删完了，如果删完了则标记一下这个子树之中不存在元素。时间复杂度可以用类似插入的分析方法得到，时间复杂度为 $O(\log_w V)$ 。

2.4 压位 trie 的查询前驱操作

我们在一个大小为 2^k 的压位 trie 中查询 x 的前驱，首先先尝试在 $A_{high(x)}$ 之中查询 $low(x)$ 的前驱，如果其存在，那么就得到了答案，如果不存在，那么尝试找到其之前一个存在元素

的子树 A_j , $0 \leq j < \text{high}(x)$ 且 j 最大, 如果存在这样一个子树, 就返回这个子树中最大值, 如果不存在, 那么其在这个压位 trie 中也不存在前驱。实现时可以用位运算技巧直接得到 j 的值。

对于查询一个子树里的最大值, 它最高位肯定是最大的, 因此仅需要找到 B 的最高位递归查询即可。

除了找最大值的部分, 就是访问一条根到叶子的链, 每个节点处花费的时间均为 $O(1)$, 求最大值也是访问一条到叶子的链, 因此时间复杂度为 $O(\log_w V)$ 。

2.5 压位 trie 的查询后继操作

可以发现, 压位 trie 的结构是对称的, 因此可以直接使用类似查询前驱的方法即可, 时间复杂度为 $O(\log_w V)$ 。

2.6 压位 trie 的空间复杂度

假设一颗大小为 2^k 的压位 trie 的整型个数为 $F(k)$ 。

容易发现 $F(i) = 1 (1 \leq 2^i \leq w)$, $F(k) = wF(k-c) + 1$, 容易得到空间复杂度为 $O(2^k)$, 使用一些技巧例如调整第一层子压位 trie 个数, 就可以达到 $O(\frac{2^k}{w})$ 的空间复杂度。

2.7 压位 trie 的实现

使用类似线段树的实现, 自顶向下搜索节点并且更新是一种可行的方法, 但是根据笔者的实现, 常数相对来说较大, 于是我们考虑改变实现的方法。

我们继续使用类似线段树的结构, 但是我们转而使用自底向上更新, 笔者这里使用了若干个数组分别记录每一层的信息, $A_{i,j}$ 维护了区间 $[j \times 2^i, (j+1) \times 2^i)$ 中的元素。

当插入时, 我们自底向上更新, 如果插入前该子树已经有数了, 就可以不必继续向上维护信息, 直接结束插入函数。

当删除时, 我们自底向上更新, 如果删除后子树之中还存在数, 我们也不必继续向上维护信息, 直接结束删除函数。

查询 x 的前驱我们仅仅需要往上寻找到第一个节点, 使得子树里存在值比 x 小, 然后寻找对应子树之中的最大值即可, 查询后继也可以使用类似的方法实现。

该实现方法剪枝较多, 所以在一般情况下表现非常优秀, 如果有更好的实现方法也欢迎与笔者交流。

2.8 压位 trie 的一些拓展

在部分题目中，对信息的维护可能比较复杂，如果继续使用 w 叉可能无法快速维护，例如我们需要支持插入一个元素，查询 $< x$ 的数的个数。

我们发现新的查询无法在之前的压位 trie 上直接实现的主要原因是，我们无法快速查询前 k 个儿子中一共有几个元素。

我们假设叉数为 B ，每当这个子树插入 B 个元素之后，我们重构一次，计算出每个儿子中有几个元素，并求出其前缀和，因此我们只需要快速计算最近 B 个插入的数中有几个 $< x$ 的。

我们考虑使用一种新的方法存储每个子树中的新元素个数，我们使用 1 的个数表示数字的值，由于其存在 B 个子树，所以我们使用 $B-1$ 个 0 当做数据的间隔，因此一个子树中每个儿子有几个数我们就可以使用一个整型压缩存储。而对于修改操作，我们要快速找到第 k 个子树对应的位置，这个就需要我们快速查询一个二进制第 k 个 0 的位置。我们可以使用 Method of Four Russians 来优化，即打出一张表，存储每一个不同的查询的答案。对于查询，我们也只要找到第 k 个 0 就知道前面有几个 1 了，我们就可以 $O(1)$ 支持一个子树元素个数加一，查询前 k 个子树的元素个数总和。如果我们取 $B = \frac{\log_2 n}{3}$ ，那么预处理复杂度将小于 $O(n)$ ，因此我们能以均摊 $O(\frac{\log_2 V}{\log_2 \log_2 n})$ 的时间复杂度解决这个问题，而我们也有一些手段将其变为严格 $O(\frac{\log_2 V}{\log_2 \log_2 n})$ 。

3 vEB tree

我们观察压位 trie 的时间复杂度，可以发现其叉数越大，效率越高，然而当叉数到了 w ，由于其整数操作复杂度不再是 $O(1)$ ，所以其时间可能不减反增。

我们发现其使用整数操作的几个地方主要有：在插入与删除时将一个位设为 0/1，在查询前驱后继时查询一个节点第一个编号 $< x$ 或 $> x$ 的儿子，我们发现第一个操作会进行 $O(\log_w v)$ 次，第二个操作只会进行 1 次！

因此如果我们使用 bitset 维护其儿子，可能会得到一个更快的数据结构。

而 vEB tree 则采取了一定的策略，使得其叉数变大且巧妙的保证了时间复杂度，可以在 $O(\log_2 \log_2 V)$ 的时间复杂度内完成插入、删除，查询一个数的前驱、后继。

3.1 vEB tree 的结构

一个大小为 2^k vEB tree 可以维护一个数字在 $[0, 2^k)$ 中的集合。

若 $k < 2$ ，则可以轻易使用 2^k 个 bool 值来 $O(1)$ 维护这个数据结构。否则我们令 $m = \lfloor \frac{k}{2} \rfloor$ 。我们将定义 2^{k-m} 个大小为 2^m 的子 vEB tree $A_i (0 \leq i < 2^{k-m})$ 。

我们定义 $high(x) = \lfloor \frac{x}{2^m} \rfloor$ ， $low(x) = x \bmod 2^m$ ，即 x 的高位与低位。对于这个 vEB tree 维护的集合，我们记录集合的最大值 max 与最小值 min （若集合为空则分别为 $-\infty$ 与 ∞ ），

并将除了最小值的元素插入其子 vEB tree 当中，对于元素 x ，它将在 $A_{high(x)}$ 这个子 vEB 中插入 $low(x)$ 。

容易发现，对于每个子 vEB tree A_i ，它存储了高位为 i 的数的低位，为了加速寻找，我们应该再定义一个大小为 2^{k-m} 的 vEB tree B ，其维护了出现的数的高位的集合，其可以用来加速我们的查询。

3.2 vEB tree 的简单操作

对于一个空的 vEB tree，我们插入一个元素仅仅需要将其 min 与 max 设置为要插入的 x 的数即可，时间复杂度 $O(1)$ 。

对于一个 vEB tree，可以通过检验其元素最大值与最小值是否相同来判断其集合大小是否为 1，数据复杂度 $O(1)$ 。

对于一个集合大小仅仅为 1 的 vEB tree，我们删除其最后一个元素仅需要将 min 与 max 分别设置为 ∞ 与 $-\infty$ 。

如果在 vEB tree A_x 中存在一个元素 y ，那就说明存在元素 $x \times 2^m + y$ ，所以可以快速计算一个子 vEB tree 中的值在这个 vEB tree 中对应的值。

快速完成这些操作对于 vEB tree 来说是不可缺少的。

3.3 vEB tree 的插入操作

假设我们要在一个大小为 2^k 的 vEB tree 中插入元素 x 。

如果该 vEB tree 为空，我们仅需要使用上述方法插入。

如果 x 比当前最小值 min 小，我们仅需要交换 x 与原来的 min ，并且在后续过程中插入原来的 min 。

如果 x 比当前最大值 max 大，我们仅需要更新 max 。

首先我们需要在 $A_{high(x)}$ 这个 vEB tree 中插入 $low(x)$ ，并且在 B 这个 vEB tree 中插入 $high(x)$ 。

容易发现的是，当原来的 $A_{high(x)}$ 非空时， B 中必然已经有元素 $high(x)$ ，应此仅仅需要在 $A_{high(x)}$ 中进行插入，否则 $A_{high(x)}$ 为空，进行插入仅仅需要设置其 min, max ，我们接下来仅仅需要在 B 中插入 $high(x)$ 。

容易发现，对于大小为 2^k 的 vEB tree，我们定义其插入一个元素的时间为 $F(k)$ 。

容易得到 $F(0) = O(1), F(1) = O(1)$ ，对于 $k > 1$ ， $F(k) = F(\lceil \frac{k}{2} \rceil) + O(1)$ ，解得 $F(k) = O(\log_2 k)$ ，而 k 取恰当的值就能以 $O(\log_2 \log_2 V)$ 的时间完成插入操作。

3.4 vEB tree 的删除操作

假设我们要在一个大小为 2^k 的 vEB tree 中删除元素 x 。

如果该 vEB tree 集合大小为 1，我们仅需要使用上述方法删除。

如果 x 是当前最小值 min ，那么我们只需要直接删除 min ，并且尝试求出新的 min ，显然的是，最小值的最高位必然是最小的，我们只需要找到 B 中的最小值并且在对应的子 vEB tree 中找出新的最小值，然后我们删除这个最小值并将 min 赋值为该值。

接下来，我们要在 $A_{high(x)}$ 这个 vEB tree 中删除 $low(x)$ ，并若 A 删除后为空就在 B 这个 vEB tree 中删除 $high(x)$ 。

如果 $A_{high(x)}$ 集合大小为 1，则可以 $O(1)$ 删除其中的元素并且在 B 中删除 $high(x)$ 。

如果 $A_{high(x)}$ 集合大小非 1，那么我们仅仅需要在 $A_{high(x)}$ 中删除 $low(x)$ 。

容易发现删除的时间复杂度与插入类似，同样是 $O(\log_2 \log_2 V)$ 。

3.5 vEB tree 的查询前驱操作

假设我们要在一个大小为 2^k 的 vEB tree 中查询 x 的前驱。

首先， x 的前驱与 x 高位相同当且仅当 $A_{high(x)}$ 中的 min 比 $low(x)$ 小，因此可以直接判断它前驱是否在 $A_{high(x)}$ 当中。如果前驱在 $A_{high(x)}$ 当中，我们直接在这个 vEB tree 中查询 x 的前驱。

否则， x 的前驱的高位一定是比 $high(x)$ 小的数中最大数，因此，我们在 B 中查询 $high(x)$ 的前驱 y ，而它前驱的低位必然是 A_y 中最大的，即 A_y 的 max 。

需要注意的是，如果 B 中的最小值如果大于等于 $high(x)$ ，那么 x 前驱将会是这个 vEB tree 的 min 。

3.6 vEB tree 的查询后继操作

假设我们要在一个大小为 2^k 的 vEB tree 中查询 x 的后继。

首先， x 的后继与 x 高位相同当且仅当 $A_{high(x)}$ 中的 max 比 $low(x)$ 大，因此可以直接判断它后继是否在 $A_{high(x)}$ 当中。如果后继在 $A_{high(x)}$ 当中，我们直接在这个 vEB tree 中查询 x 的后继。

否则， x 的后继的高位一定是比 $high(x)$ 大的数中最小数，因此，我们在 B 中查询 $high(x)$ 的后继 y ，而它后继的低位必然是 A_y 中最小的，即 A_y 的 min 。

需要注意的是，最小值不在这棵压位 trie 的子结构之中，所以如果 x 比这个压位 trie 中的最小值要大，那么需要直接返回最小值，不然就可能会返回错误的结果。

3.7 vEB tree 的一些简单优化

容易发现的是，当 $k < 2$ 时再使用 2^k 个 $bool$ 来维护集合时， vEB tree 深度常数可能会比较大。而当 $2^k \leq w$ 时，就可以使用一个整型轻松的维护一个大小为 2^k 的集合。如果 $w = 64$ ，我们维护一个大小为 2^{24} 的集合都仅需要递归两层即可。大大的优化了常数。

在部分使用 $w = 64 = 2^6$ 为测试机器的地方，我们定义大小为 2^{24} 的 vEB tree 的话基本可以满足大部分使用需求，且在其它操作中仅仅需要递归两层，大大提升了效率。

3.8 vEB tree 的实现

由于 vEB tree 的结构相对复杂，难以使用数组直接实现，故可以使用 C++ 中的 template 语法来定义不同大小的 vEB tree。并且特化了大小较小的 vEB tree，使用压位整型来实现，优化常数。使用此方法后代码难度相对降低，较为容易实现。如果有更加优秀的实现方法欢迎与笔者交流。

3.9 vEB tree 的空间复杂度

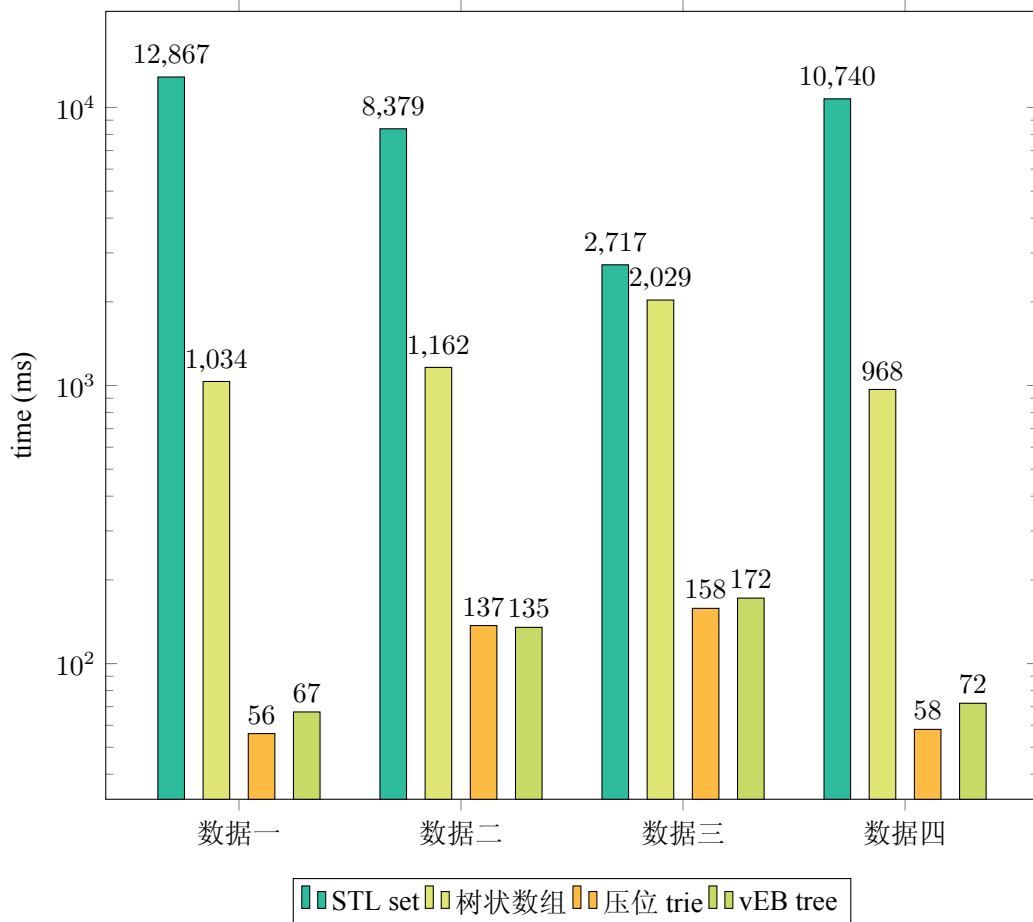
假设一个大小为 2^k 的 vEB tree 的中整型个数为 $F(x)$ 。

容易得到： $F(i) = 1(1 \leq 2^i \leq w)$, $F(i) = F(\lceil \frac{i}{2} \rceil) + 2^{\lceil \frac{i}{2} \rceil} F(\lfloor \frac{i}{2} \rfloor) + 2$ 。

我们可以证明 $F(x)$ 可能会达到 $O(\frac{2^k}{\sqrt{w}})$ ，不过可以通过调整 m 的大小即 vEB tree 的叉数能够得到 $O(\frac{2^k}{w})$ 的空间复杂度。

如果值域范围特别大，我们也可以考虑使用 hash 表等方法存储 vEB tree 以得到可以接受的空间复杂度。

4 对于部分数据结构效率的一些测试⁴



在所有数据中，操作个数均为 10^7 ，值域范围为 $[0, 2^{24}]$ 。

其中第一组数据为：插入 10^7 个不同的数字。

其中第二组数据为：插入 10^6 个不同的数字后进行 9×10^6 次前驱/后缀查询。

其中第三组数据为：插入 10^5 个不同的数字后进行 99×10^5 次前驱/后缀查询。

其中第四组数据为：插入 5×10^6 个不同的数字后再删除这 5×10^6 个数，顺序随机。

所有数字均随机生成。

从时间角度来说：可以发现 vEB tree 和压位 trie 在时间上还是远远快于其它两个 log 数据结构，在集合数字比较稀疏的情况下的查询甚至比树状数组快了 10 倍多。相对来说，vEB tree 的插入删除略慢于压位 trie，但是其查询效率还是可以的，但是这仅仅是随机数据，在更加强力的数据下 vEB tree 可能会有相对更高的查询效率。但是压位 trie 已经完全能够满足我们的使用需求。

从空间角度来说：四个数据结构使用的空间分别为：455MB, 64MB, 2.06MB, 2.03MB。

⁴测试代码可以在 <https://skip2004.blog.uoj.ac/blog/6551> 查看

可以见压位 trie 与 vEB tree 在操作数和值域基本同阶的情况下，空间也是非常占优势的。而树状数组的空间也相对来说较小。

从实现难度来说，笔者认为压位 trie, vEB tree 实现难度均在可接受范围之内，而压位 trie 相对来说更加容易实现，且在很多情况下效率更高。

总的来说，STL set 虽然容易实现，但是消耗时间空间都较大，笔者比较推荐使用树状数组或者压位 trie 来解决这一类问题。而在下面一些例题中可以看到，压位 trie 在实际中的应用还是比较广泛的。

5 简单应用

5.1 【北大集训 2018】ZYB 的游览计划

5.1.1 题目大意

有一棵 N 个顶点的树，和一个 $1 \sim N$ 的排列 P 。

定义一个整数区间 $[L, R]$ 的权值为从 1 号点出发遍历完 P_L, P_{L+1}, \dots, P_R （不一定按顺序）中的所有点最后回到 1 号点需要经过的最少边数。

现在将 $[1, N]$ 划分成 K 个区间，问权值和的最大值。

数据保证 $2 \leq K \leq N \leq N \times K \leq 2 \times 10^5$ 。

时间限制：7s

空间限制：512MB

5.1.2 做法

对于本题，我们首先可以使用决策单调性来优化 DP，由于内容和本文无关，这里略去不讲。我们要解决的问题是：维护一个集合 S ，支持往里面插入一个点与删除集合内的点，查询从 1 号点出发遍历完集合内所有点需要经过的最小边数，而其中的插入删除次数会达到 $O(nk \log_2 n)$ ，相对来说较大。

容易发现的是，我们按 dfs 序以此访问就是最优的答案了，因此我们需要维护 dfs 序中相邻两个点的距离之和以及 1 号点与 dfs 序最小和最大的点的距离。

我们在里面插入一个点 b ，首先求出这个点 dfs 序之前的点 a 和 dfs 序之后的点 c ，我们发现插入点 b 之后的答案会增加 $dist(a, b) + dist(b, c) - dist(a, c)$ 。

我们在里面删除一个点 b ，首先求出这个点 dfs 序之前的点 a 和 dfs 序之后的点 c ，我们发现删除点 b 之后的答案会减小 $dist(a, b) + dist(b, c) - dist(a, c)$ 。

如果其 dfs 序之前或者之后的点不存在，那么我们将其设置为 1 号点即可。

因此我们可以使用 vEB tree 来维护集合内点的 dfs 序集合，使用 ST 表 + 欧拉序求 lca 以及点对距离即可做到 $O(nk \log_2 n \log_2 \log_2 n)$ 的时间复杂度。

相对于原题给出的 $O(nk \log_2^2 n)$ 做法，上述做法的复杂度与常数更加优秀。笔者使用压位 trie 实现了该题目，经过测试可以在 150ms 左右内解决这个问题，远小于原题所要求的 7s 的时间限制。

5.2 【ZJOI 2019】语言

5.2.1 题目大意

有一棵 n 个顶点的树，和 q 条树上的链 (u_i, v_i) 。

对于一条链，链上的任意不同的两个点之间都可以展开贸易活动。

询问一共有多少点对之间可以开展贸易活动。

时间限制：3s

空间限制：512MB

5.2.2 做法

我们先假设一个城市能与自己展开贸易活动，然后考虑计算有序点对数量 (u, v) 为 ans ，那么答案就是 $\frac{ans-n}{2}$ 。

我们考虑计算可以和每个点开展贸易活动的点数。

容易发现，能和一个点展开贸易活动的点集（包括其自己）是覆盖它的若干条链的并，因此其为一个连通块，我们只需要包含它的链的端点，求出包含这些端点的最小连通块大小。

在常规做法之中，我们使用动态开点线段树来维护点集内点的 dfn 序集合，我们要求的连通块大小显然就是 dfn 序相邻的点距离加上 dfn 序最小的点与最大的点的距离之和的一半，然后线段树每个节点记录区间 dfn 最小与最大的点以及这个区间相邻的点的距离和，然后可以轻易上传信息，再使用线段树合并就可以达到 $O(n \log_2 n)$ 的时间与空间复杂度。

然而线段树做法时间空间常数都略大，我们考虑继续优化。

我们考虑使用动态开点压位 trie 维护 dfn 序集合。

使用一个数组 $ch[x][i]$ 表示一个压位 trie x 的第 i 个子压位 trie 的编号， $w[x]$ 表示压位 trie x 有哪些子压位 trie 非空，本题还需要记录子树最小最大值。

对于插入/删除一个节点，我们在压位 trie 上直接寻找要插入的儿子编号，与动态开点线段树类似，若不存在该儿子我们则新建一个节点，插入之后我们使用类似上一题的方法更新答案。

对于合并两个压位 trie，我们考虑使用类似线段树合并的方式实现：

若两个节点中有一个是空节点，则返回另外一个。

我们选取儿子个数大的节点作为新的根，并在之后把另外一个节点删除，在删除之前，我们需要将还有一个节点的儿子与这个节点的儿子合并。

对于它们的公共儿子，我们直接枚举并且合并，这里可以通过与运算来快速获得它们的公共儿子。

对于第一个节点没有但第二个节点有的儿子，直接暴力赋值。

我们考虑分析其时间复杂度。首先合并两个节点的公共儿子部分时间复杂度与线段树合并类似，每次合并都会少一个节点，所以这部分总时间复杂度是 $O(n \log_w n)$ 的。而对于暴力赋值对于第一个节点没有但第二个节点有的儿子的部分，可以发现第二个节点的儿子必定至少有一个元素，而赋值的子树是互不相交的，因此赋值次数不会超过两个压位 trie 大小的较小值，类似于启发式合并，这部分总复杂度为 $O(n \log_2 n)$ ，但常数非常小，复杂度瓶颈部分仅仅为一些赋值语句。

本题还需要在合并的同时改变答案，例如合并两个节点公共儿子时需要注意这个子树中第一个元素和最后一个元素改变后需要重新计算其与前驱后继的距离。

但是，它还存在一些弊端，其空间复杂度较大，需要 $O(n \log_w n)$ 个节点，每个节点需要 $O(w)$ 个整型来存储儿子编号，总空间复杂度为 $O(nw \log_w n)$ ，因此我们需要继续优化空间复杂度。

我们首先使用重链剖分，在树上 dfs 时候第一次往重儿子走，然后将当前 trie 设为重儿子的 trie，可以发现这样最多同时存在 $O(\log_2 n)$ 棵 trie，使用较好的实现，一棵 trie 中只会存在 $O(\frac{n}{w})$ 个节点，其占用的空间最多为 $O(n)$ ，使用内存回收⁵可以做到 $O(n \log_2 n)$ 的空间复杂度，且常数较小。

因此，我们使用压位 trie 得到了一个时间空间常数更加小的做法，并且这种压位 trie 合并的思想可以拓展到一些类似的题目之中。

总结

本文介绍了压位 trie 和 vEB tree 两种可以用于解决 Dynamic Predecessor Problem 的数据结构，并且给出了其在部分 OI 题目中的应用。并且本文介绍的以及其它亚 log 数据结构可能还有更多有趣的作用，故希望本文能起到抛砖引玉的作用，希望有兴趣的读者能继续研究，得到更多有趣的应用。

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高闻远的指导。

⁵内存回收时需要注意清空的时间复杂度，不要使用类似于 memset 的方式。

感谢父母对我的培养和教育。

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

感谢虞皓翔同学，潘佳奇同学，翁伟捷同学，戴江齐同学，与我交流讨论、给我启发。

感谢罗恺同学，施开成同学，孙睿泽同学为本文审稿。

参考文献

- [1] Wikipedia, the free encyclopedia, “Predecessor problem”, https://en.wikipedia.org/wiki/Predecessor_problem.
- [2] Wikipedia, the free encyclopedia, “Van Emde Boas tree”, https://en.wikipedia.org/wiki/Van_Emde_Boas_tree.
- [3] Wikipedia, the free encyclopedia, “Method of Four Russians”, https://en.wikipedia.wikimirror.org/wiki/Method_of_Four_Russians.
- [4] Peter van Emde Boas, *Preserving order in a forest in less than logarithmic time*, Proceedings of the 16th Annual Symposium on Foundations of Computer Science 10: 75-84, 1975
- [5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*
- [6] 中国计算机学会, 2018 全国信息学奥林匹克年鉴

对信息学竞赛中二维平面处理问题的总结和再优化

福建省厦门双十中学 施良致

摘要

本文总结了在信息学竞赛中对于二维平面问题处理的常见做法——KD-Tree 和区域树(线段树套线段树), 并利用分散层叠算法对于区域树进行改进, 优化其时间复杂度。在最后一节中, 笔者通过例题, 展示分散层叠算法改进区域树在信息学竞赛中解决二维平面问题的应用。

1 前言

二维平面处理问题在信息学竞赛中是比较经典的问题。对于此类问题, 在信息学竞赛中一般使用 KD-Tree, 树套树, 分块, CDQ 分治等算法来解决。然而, 这些算法的时间复杂度在一般情况下都不优于 $O(m \log^2 n)$ 。

本文将介绍一种在信息学竞赛中不常出现的新处理方法, 该算法在某些特定的情况下能将此类问题的时间复杂度优化至 $O(m \log n)$ 。此外, 本文还会通过例题展示其在信息学竞赛中的应用, 并且本文还会将其与选手熟知的算法进行对比, 分析该算法的优劣。

2 经典在线的二维问题

问题 1. 给定平面上 n 个点, 第 i 个点的坐标为 (x_i, y_i) , 其权值为 w_i 。接下来将进行 m 次查询, 每次查询一个左下角为 (x_1, y_1) , 右上角为 (x_2, y_2) 的矩形中的点的权值和, 查询强制在线。

上述问题为一个常见的二维平面处理问题。对于该问题, 下文将介绍三种解决方法。前两种分别为较为常见的 KD-Tree 和树套树, 第三种为分散层叠 (Fractional Cascading) 算法。

2.1 KD-Tree

对于此类在线的区间查询问题, 有经典的解决方法, 即 KD-Tree。

KD-Tree 是一种可以高效处理 k 维空间信息的数据结构, 其在处理二维问题上是非常有用的工具。

2.1.1 解决方案

KD-Tree 的构造方法为每次选择一个维度，并切分这个维度。

对于本问题，KD-Tree 的构造方法为交替选择维度（横向或纵向），然后按照此维度排序（按横坐标排序或按纵坐标排序）。如图所示*，排序后选取中点并按照中点的位置将平面切割开，此时所有点被分成两个集合，这两个集合分别递归构造子树。

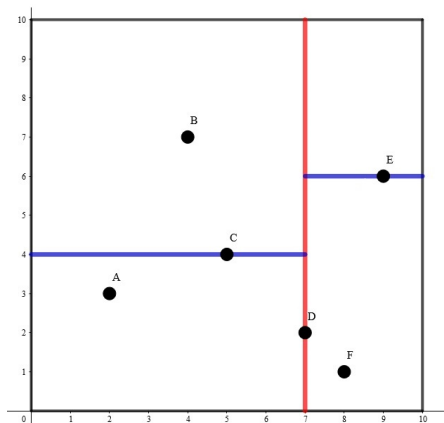


图 1: KD-Tree 的构造

按此方式构造后，KD-Tree 上的每个节点都对应平面上一个矩形区域。

查询时可以遍历整个 KD-Tree，对于每个节点的处理可以分为以下三种：

1. 当前矩形的边框或内部与查询矩形的边框相交：递归查询该节点的左子树和右子树。
2. 当前矩形被查询矩形相含：直接返回该节点对应子树中所有点的权值和。
3. 当前矩形与查询矩形相离：子树中所有点都不在查询矩形中，直接返回。

2.1.2 时间复杂度

定理 2.1. 对于点数为 n 的 KD-Tree，每次查询的时间复杂度为 $O(\sqrt{n})$ 的。

证明. 根据构造，不难得知，当某个节点对应的矩形的边框或内部与查询矩形的边框不相交时，均不用继续递归。当某个节点对应的矩形的边框或内部与查询矩形的边框相交时，均会继续递归。

因为每个节点的子节点最多两个，设有 x 个节点对应的矩形与查询矩形的边框相交，那么被访问的节点数会严格小于 $3x$ 。因此只需要证明 KD-Tree 上与查询矩形有交的矩形个数为 $O(\sqrt{n})$ 。□

定理 2.2. 对于任意一个查询矩形，在 KD-Tree 上与之有交的矩形为 $O(\sqrt{n})$ 。

*图片来源：<https://oi-wiki.org/ds/images/kdt1.jpg>

证明. 可以将矩形边框拆分成四条线段，而相交的矩形数不会超过分别和每条线段相交的矩形数目的总和。

接着将线段向两边延伸转换为直线，与线段相交的矩形数目不会超过与直线相交的矩形数目。

每次考虑两层节点，即把当前矩形划分成四份，设 $F(n)$ 表示 n 个节点的 KD-Tree 与一条直线相交的矩形数。□

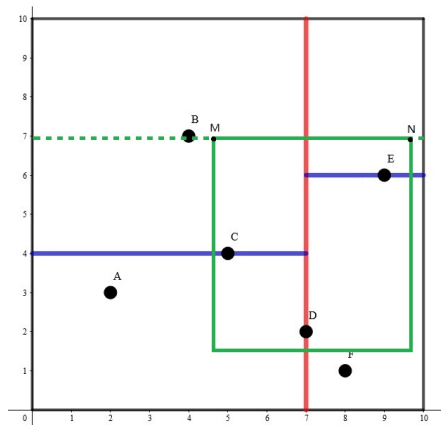


图 2: KD-Tree 复杂度分析

根据以上证明，可以得到: $F(n) = O(2) + 2F(n/4)$ 。

根据 Master Theorem 得到， $F(n) = O(\sqrt{n})$ 。

2.2 区域树

区域树 (Range Tree)，在信息学竞赛中一般被称为线段树套线段树。这是一种常见的对二维问题处理的方法。本文后面将会使用分散层叠算法对区域树进行优化。

2.2.1 解决方案

区域树的构造方法为将两个维度分开考虑。先对第一个维度建立一棵静态线段树，线段树上每个节点对应一段数据。对于这些数据，再按照第二维建立一棵静态线段树。

在查询时，可以在外层线段树上查询出第一个维度对应的查询区间，再在对应查询区间的内层线段树上查询第二个维度的信息。

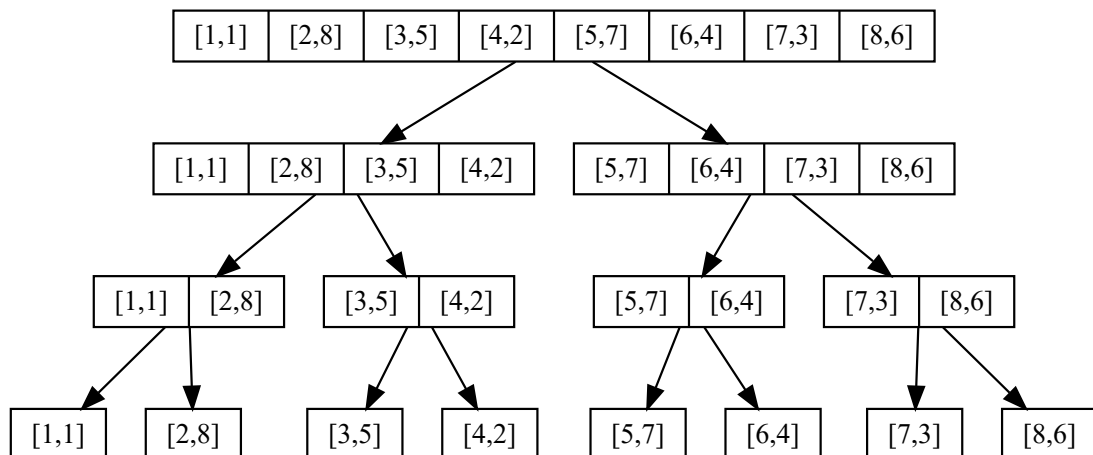


图 3: 区域树, 每个节点上存有一定数据, 每个节点上的数据独立建立线段树

2.2.2 时间复杂度

不难证明, 在外层线段树上会对应 $O(\log n)$ 个区间, 每一个区间在内层线段树上会对应 $O(\log n)$ 个区间, 因此时间复杂度为 $O(m \log^2 n)$ 。

2.3 分散层叠算法优化区域树

在信息学竞赛中, 对于此问题的在线算法大多都停留在时间复杂度为 $O(m \log^2 n)$ 或 $O(m \sqrt{n})$ 的做法上。事实上, 这类问题可以优化至时间复杂度为 $O(m \log n)$, 空间复杂度为 $O(n \log n)$ 。

学术界中存在一种名为分散层叠 (Fractional Cascading) 的算法, 在信息学竞赛中, 这个算法在 IOI2020 中国国家候选队论文《浅谈利用分散层叠算法对经典分块问题的优化》中出现过。该算法可以运用在对区域树的优化上, 并且可以将此类问题的时间复杂度优化至 $O(m \log n)$ 。在下文中 (除 2.3.1 外), 文中分散层叠算法的意思均为分散层叠算法优化区域树。

2.3.1 分散层叠

由于本文的目的为使用分散层叠算法对区域树进行优化, 因此本文并不详细讨论分散层叠算法的主要用途, 而只介绍该算法的一个用途。

对于两个数集 A, B , 保证 $B \subseteq A$, 此时是否可以通过构造映射 $A \rightarrow B$, 使得若知道了 A 中比 x 大的第一个数时, 可以在 $O(1)$ 的时间复杂度下得到 B 中比 x 大的第一个数。

不难想到, 只需要让 A 中的每个数映射到 B 中第一个大于等于自己的数上。

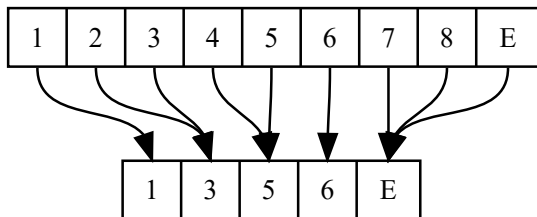


图 4: 分散层叠算法, E 为集合末尾的空节点

例如 $x = 4$, 在 A 集合中大于等于 4 的第一个数是 4, 并且由映射关系可以得知在集合 B 中大于等于 4 的第一个数是 5。

2.3.2 改进区域树结构

在 2.2 中有提到区域树的实现思想, 即将数据的两个维度分开处理, 先按第一个维度分治, 然后在每个节点上将所在节点的数据建立一棵线段树。

然而对于问题 1, 在第二维上其实并不需要使用线段树。如果先将每个节点上的数据按照第二维从小到大排序, 并且知道所查询节点上第二维符合询问的数据对应的区间, 那么只需要一个前缀和就可以解决权值和的问题。在这里, 分散层叠算法恰好可以解决定位区间的问题。

以 2.2.1 中的图 3 的区域树举例, 对于每个节点上的数据, 可以先按第二维从小到大排序, 然后对于上下两层按第二维的大小建立 2.3.1 中所描述的映射关系。

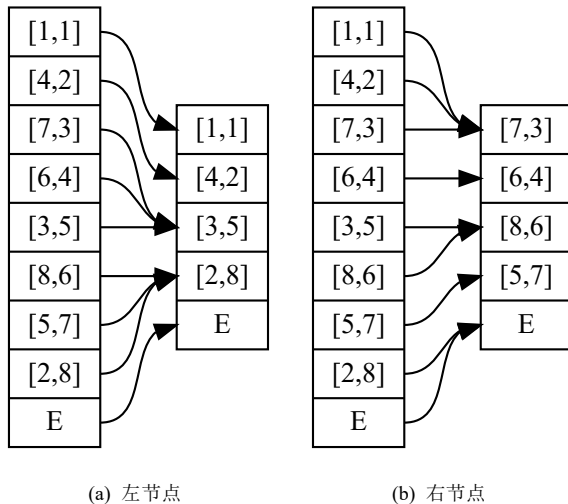


图 5: 第一层和第二层的节点数据之间的映射关系

以此类推, 每一层的节点中的数据都可以和下一层节点中的数据建立映射关系。这个

映射关系的建立可以用归并算法实现。

2.3.3 查询方式

与区域树查询数据的方法类似的，我们首先要在外层线段树上查询得到第一维符合条件的节点，而分散层叠算法可以在外层线段树节点移动的同时在 $O(1)$ 的时间复杂度下确定节点上第二维合法的数据所对应的区间。

假设查询第二维的范围为 $[l, r)$ ，则一个节点上第二维合法的数据为第一个大于等于 l 的数据到第一个大于等于 r 的数据之间的所有数据。而如果得到了大于等于 l 和大于等于 r 的第一个数据，其子节点上大于等于 l 和大于等于 r 的第一个数据也可以 $O(1)$ 得到。

以 2.2.1 中图 3 的区域树举例，我们希望查询第一维在 $[1, 6]$ ，第二维在 $[3, 8)$ 中的所有数据。

首先可以通过二分的方式在根节点中确定 $[3, 8)$ 的位置，即根节点中第二维大于等于 3 和 8 的第一个数据，在该例子中为 $[7, 3]$ 和 $[2, 8]$ 两个数据。

如图，在外层线段树查询时，总可以通过 $O(1)$ 的时间找到子节点中第二维在 $[3, 8)$ 中的数据。

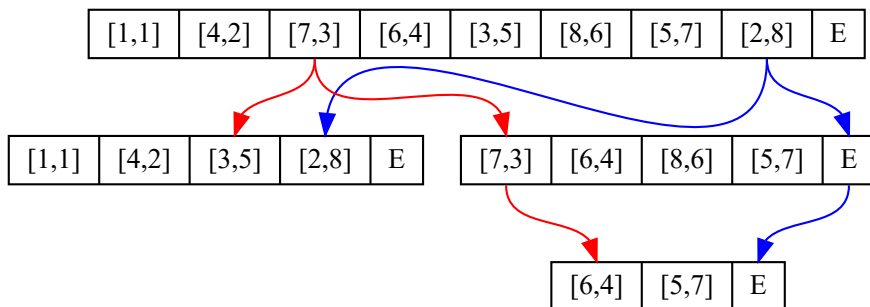


图 6: 分散层叠算法的查询方式

不难发现，对于每一个节点中的数据，某段区间的点的权值和可以用前缀和差分计算，这样是 $O(1)$ 的时间。因此，当外层线段树的搜索结束时，整个查询也就结束了。

2.3.4 空间复杂度分析

在 2.3.2 中介绍的这种算法的构造方法为归并算法，该算法总共的层数在 $O(\log n)$ 级别，每一层所有数据数目之和为 n 。

对于每一个数据，除了自身外，还需要多记入其在下层两个区间中对应的后继以及计算答案需要使用的和，但空间复杂度仍为 $O(n \log n)$ 。

2.3.5 时间复杂度分析

第一维的查找和线段树的时间复杂度是一致的，单次为 $O(\log n)$ 。但在第二维查找上，该算法在定位区间和计算答案上都是 $O(1)$ 的。

综上所述，对于每一次查找，该算法的时间复杂度为 $O(\log n)$ 。

2.4 三种算法的比较

2.4.1 从空间上分析

对于 KD-Tree，其空间复杂度为 $O(n)$ ，而区域树和分散层叠算法的空间复杂度均为 $O(n \log n)$ 。因此，KD-Tree 在处理此类问题时在空间复杂度上占有极大优势。

2.4.2 从时间上分析

对于 KD-Tree、区域树和分散层叠三种算法，其理论复杂度分别为 $O(m\sqrt{n})$ 、 $O((n+m)\log^2 n)$ 和 $O((n+m)\log n)$ 的。从理论上讲，分散层叠算法相对前两者在时间复杂度上占极大的优势。然而，在实际中并非如此。

通过实验分析，分散层叠算法在递归次数上有很大的优化。当 $m = 10^6$ 时，区域树的函数递归次数约为 6×10^8 次，KD-Tree 的函数递归次数约为 2×10^9 次，而分散层叠算法的递归次数只有 6×10^7 次。然而，分散层叠算法在移动指针、计算答案时都需要进行繁琐的操作，该操作成本使得分散层叠算法有一个较大的常数，这使得分散层叠算法在时间效率上表现较差。

不过尽管如此，在 n 和 m 较大时，分散层叠算法在运行时间上相对区域树和 KD-Tree 还是有较明显的优势。经过反复实验测试，对于问题 1，在 $n = 2 \times 10^5, m = 10^6$ 的随机数据下，KD-Tree 的运行时间最坏会达到 39s，区域树的平均运行时间为 15.7s，分散层叠算法的平均运行时间为 6.7s。由此可以看出，分散层叠算法虽然不能达到期望下比区域树快 10 ~ 20 倍的水平，但还是有明显的优势。

3 经典离线的二维问题

问题 2. 给定平面上 n 个点，第 i 个点的坐标为 (x_i, y_i) ，其权值为 w_i 。现在有 m 个询问，每个询问为查询一个以 (x_1, y_1) 为左下角，以 (x_2, y_2) 为右上角的矩形中的点的权值的最大值。

与问题 1 类似，该问题也可以使用 KD-Tree 和区域树解决，方法与问题 1 基本相同，时间复杂度也基本相同。而问题 1 中分散层叠算法利用了前缀和的性质，在这里失去了这个性质，所有没办法使用解决问题 1 的方法。因此这里将介绍另外一种方法。

3.1 问题转换

与问题 1 类似，在外层线段树查找时可以顺便得出每个节点合法数据的区间。而外层线段树上最多对应 $O(\log n)$ 个节点，此时每个节点中有一个询问，为询问这个节点的数据中某一段的最大值。

在线处理并不方便，因此考虑离线处理。不妨先把所有询问都记录下来，接着把这些询问按照其在外层线段树上对应的节点分组，把在同一个节点上的询问分成同一组。

分组后再按组回答询问。每一组询问都可以表示成下列形式：先给定一个长度为 k 的序列，再给定 q 个询问，每个询问为询问 $[l_i, r_i]$ 中数据权值的最大值。如果把所有组的 k 和 q 累加，可以得到 $O(\sum k) = O(n \log n)$, $O(\sum q) = O(m \log n)$ 。

3.2 解决新问题

由于每一组问题的每个询问都可以用一个二元组 (l_i, r_i) 表示，并且 $1 \leq l_i \leq r_i \leq k$ 。因此，可以先使用桶排序将 q 个询问按照 r_i 升序排序。

接着需要解决的问题是查询每个询问所对应的区间 $[l_i, r_i]$ 中数的最大值。这部分可以维护一个由 $[1, r_i]$ 中数据组成的单调递减队列，单调队列中下标大于等于 l_i 的第一个元素就是区间 $[l_i, r_i]$ 中的最大值。

如果使用二分查找则总时间复杂度为 $O(k + q \log k)$ ，这样和直接使用区域树并没有区别。不过注意到单调队列中的元素都“掌管”着一段连续的区间，所以可以使用并查集来优化，这样一来复杂度就下降到 $O(k + q\alpha(k))$ 。

如果把 $2n$ 个问题叠加起来，总的时间复杂度为 $O(n \log n + m\alpha(n) \log n)$ ，该复杂度略小于 $O((n + m) \log^2 n)$ 。

3.3 总结

通过该问题，可以得到分散层叠算法在处理问题的特点：对一个二维询问，将其分解为 $O(\log n)$ 个一维询问。对于每个一维询问，如果可以通过预处理或离线处理等方法使处理一个问题的时间复杂度低于 $O(\log n)$ ，则使用分散层叠算法解决该问题可以使其时间复杂度小于 $O(m \log^2 n)$ 。

4 分散层叠在信息学竞赛中的应用

该算法在处理二维问题上是一个非常有利的工具，但是在信息学竞赛中，由于数据范围过小、算法常数过大、适用条件过于严格等因素，很少出现此类算法的题目。本文中将会选择三道与信息学竞赛联系较为密切的例题来说明分散层叠如何在信息学竞赛中应用。

4.1 例题一

问题 3. 平面上有 n 个点，第 i 个点的坐标为 (x_i, y_i) 。接下来会在线的提出 m 个询问，每个询问会给出一个坐标 (X, Y) ，并询问 n 个点中与这个点距离最近的点的距离。本题中的距离为曼哈顿距离。数据范围： $n, m \leq 5 \times 10^5$ 。时间限制：4s。空间限制：192MB。

分析. 本题是经典的在线问题的一个变种。观察题目，注意到 $n, m \leq 5 \times 10^5$ 但时限只有 4s，因此应该需要一个 $O((n+m) \log n)$ 。

每次询问可以按照询问点 (x, y) 为原点将整个平面分成四份，分别为询问点的左上区域、左下区域、右上区域和右下区域。由于：

$$\text{dist}((x, y), (x_1, y_1)) = |x - x_1| + |y - y_1| \quad (1)$$

$$= \max\{x, x_1\} + \max\{y, y_1\} - \min\{x, x_1\} - \min\{y, y_1\} \quad (2)$$

因此对于四个区域中的点可以分别设立权值 w_i 为 $-x + y, -x - y, x + y, x - y$ ，此时问题转换为求矩形内的最小值。

如果使用 KD-Tree 算法，最后的时间复杂度为 $O(m\sqrt{n})$ ，并不被允许。

如果使用分散层叠算法，则至少需要记 2 个前缀最小值，2 个后缀最小值，2 种指针，因此空间复杂度至少为 $O(6n \log n)$ 。此算法无法在题目给定的空间限制下通过。

然而，对于曼哈顿距离还有一种常见的处理方法，就是把所有点 (x, y) 都变化为 $(x + y, x - y)$ ，此时原来两点的曼哈顿距离和现在两点的切比雪夫距离相同。

定理 4.1. 点 (x_1, y_1) 与点 (x_2, y_2) 的曼哈顿距离和点 $(x_1 + y_1, x_1 - y_1)$ 与点 $(x_2 + y_2, x_2 - y_2)$ 的切比雪夫距离相等。

注意到：

$$\begin{aligned} |x_1 - x_2| + |y_1 - y_2| &= \max\{x_1 - x_2, x_2 - x_1\} + \max\{y_1 - y_2, y_2 - y_1\} \\ &= \max\{x_1 - x_2 + y_1 - y_2, x_1 - x_2 + y_2 - y_1, x_2 - x_1 + y_1 - y_2, x_2 - x_1 + y_2 - y_1\} \\ &= \max\{\max\{x_1 + y_1 - x_2 - y_2, x_2 + y_2 - x_1 - y_1\}, \\ &\quad \max\{x_1 - y_1 - x_2 + y_2, x_2 - y_2 - x_1 + y_1\}\} \\ &= \max\{|(x_1 + y_1) - (x_2 + y_2)|, |(x_1 - y_1) - (x_2 - y_2)|\} \end{aligned}$$

故定理 3 得证。

因此，问题转换为求切比雪夫距离的最小值。

如果采用二分加判断矩形内是否存在点的方法，即使使用分散层叠算法，复杂度也会退化到 $O(m \log^2 n)$ ，这是不被允许的。

不过事实上，可以在查询的同时二分。

已知询问点为 (x, y) ，设在查找时第一维确定在 $[L, R]$ 的范围上，且满足 $x \notin [L, R]$ ，即 $x < L \leq R$ 或 $L \leq R < x$ 。为了方便，这里只讨论 $x < L \leq R$ ，对于 $L \leq R < x$ 的情况仅为对称情况，请读者自证。

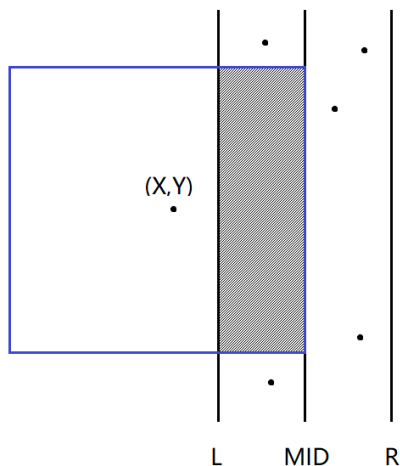


图 7: 例题一

如图，图中蓝色边框是以 (x, y) 为中心， $2(\text{MID} - x)$ 为边长的正方形。先考虑第一维在 $[L, \text{MID}]$ 中的点。不难得出，第一维在 $[L, \text{MID}]$ 并且第二维不在 $[x + y - \text{MID}, y + \text{MID} - x]$ 中的点（即位于阴影区域外的点）的切比雪夫距离都是由第二维决定的。因此，如果阴影区域内不存在点， $[L, \text{MID}]$ 就没有递归的必要了，直接计算完答案后递归 $[\text{MID}, R]$ 。

反之，如果阴影区域内存在点，这个点的切比雪夫距离一定小于等于 $\text{MID} - x$ ，这就意味着 $[\text{MID}, R]$ 中的点都不可能贡献答案（ $[\text{MID}, R]$ 中的点的切比雪夫距离都大于等于 $\text{MID} - x$ ）。所以此时直接递归 $[L, \text{MID}]$ 即可。

因此问题的关键就是快速判断阴影中是否存在点，以及如果阴影中不存在点，在阴影部分上下第二维离 y 最近的点是什么。

显然此时可以直接用分散层叠算法找到第二维坐标比 y 大的第一个点，这样可以解决上述两个问题。

这种算法除了指针和点的编号外其他都不需要存储，空间复杂度为 $O(3n \log n)$ ，时间复杂度为 $O(m \log n)$ ，可以通过本题。

□

4.2 例题二

问题 4. 给定一棵大小为 n 的有根树，点从 $1 \sim n$ 编号，树根为 1 号点。二维平面上有 m 个点，第 i 个点的坐标为 (x_i, y_i) ，其代表树上的一个点 s_i 。接下来将有 q 个询问，每个询问为

查询一个左下角为 (x_1, y_1) ，右上角为 (x_2, y_2) 的矩形中所有点的最近公共祖先。数据范围： $n, q \leq 5 \times 10^5, m \leq 5 \times 10^4$ 。时间限制：5s。空间限制：1024MB

分析. 本题没有强制在线，因此本题既可以使用在线的方法解决，也可以使用离线的方法解决。

对于在线算法，本题如果直接使用分散层叠，这样不方便直接求出区间 $[L, R]$ 中所有点的最近公共祖先。然而 m 只有 5×10^4 ，可以支持 $O(m \log^2 m)$ 的算法。因此可以先对外层线段树对应的每一个节点中的数据做一个 RMQ 预处理。这样在询问时就可以做到 $O(1)$ 了。时间复杂度为 $O((q+n) \log n + m \log^2 n)$ 。

对于离线算法，可以直接套用问题 2 的解决方案。先将询问离线，然后将询问按照 r_i 排序，最后使用单调队列加上并查集，时间复杂度为 $O((q+n)\alpha(m) \log n)$ 。不过该算法的理论复杂度并不如在线算法。□

4.3 例题三

问题 5. (2020 北大集训 树数术)

给定一棵大小为 n 的有根树，点从 $1 \sim n$ 编号，树根为 1 号点。有一个长度为 m 的整数序列 a ， $1 \leq a_i \leq n$ 。接下来有 q 个询问，每个询问会给出基于序列 a 的 k 段区间，并将这 k 段区间拼接成一个新的序列 b ，然后查询序列 b 中有多少个位置 i 满足对于所有 $1 \leq j \leq i$ ，都有 $\text{lca}(b_j, b_i) = b_i$ 。数据范围： $n, q, \sum k \leq 7 \times 10^5$ 。时间限制：4s。空间限制：1024MB。

分析. 本道题是经典离线问题的变种。

首先可以对问题进行一个简单的转换，不难发现对于每一个区间都相当于给定 l, r 和树上的一个点 x ，求区间中有多少个 i ，满足对于任意 $l \leq j \leq i$ ，都有 $\text{lca}(a_j, a_i) = a_i$ 并且 $\text{lca}(x, a_i) = a_i$ 。

先对序列中的每一个 i 都求一个数 c_i ， c_i 为满足对于任意的 $L \leq j \leq i$ 都有 $\text{lca}(a_j, a_i) = a_i$ 中最小的 L 。不妨把每个 i 都变成一个点 (i, c_i) ，这样一来就变成求有多少个点满足 $l \leq i \leq r, c_i \leq l, \text{lca}(a_i, x) = a_i$ 。

定理 4.2. 假设 h 为满足 $l \leq i \leq r, c_i \leq l$ 的所有 i 从小到大的序列，则一定满足 $\text{lca}(a_{h_i}, a_{h_{i+1}}) = a_{h_{i+1}}$ 和 $c_{h_{i+1}} \leq c_{h_i}$ 。

证明比较显然，请读者自证。

因此，如果把所有点 (i, c_i) 构建区域树（每个节点上的数据按照 c_i 从小到大排序），在第一维对应的某个合法节点上（即该节点对应的区间被 $[l, r]$ 包含），其中合法的点（即 $c_i \leq l$ ）一定满足前一个是后一个的祖先。同时，在这些点中满足是 x 的祖先的点一定是这些点的一个前缀。

此时如果使用分散层叠算法，可以得到一个 $O(\sum k \log^2 n)$ 的算法。即对于每一个询问，查出对应节点和该节点中对应的 $c_i \leq l$ 的点后，再在这些点上二分一下满足是 x 祖先的点。

事实上这个算法还可以优化。因为这些合法点构成的虚树一定是一条链，不妨记链的叶子为 y ，可以先把 x 变成 $\text{lca}(x, y)$ ，这样并不影响查询的正确性。然后把所有询问离线，使用桶排序让这些 $\text{lca}(x, y)$ 按照深度从深到浅排序，此时再询问就不需要二分，直接在每一个节点上维护一个指针表示在该节点上满足深度小于等于 $\text{lca}(x, y)$ 所对应的前缀。由于 $\text{lca}(x, y)$ 只会从深到浅，所以前缀只会缩短，指针只会向前移动，因此这部分复杂度是线性的。这样一来总时间复杂度就变成 $O((\sum k + n) \log n)$ 的了。 \square

4.4 总结

分散层叠算法因为本身对查询方式有极为特殊的要求，大多数问题都可以转换为经典在线问题和经典离线问题。即该问题的数据有可以预处理的特点，或该问题允许离线询问，并且在离线后总能按照某种顺序，用极短的时间回答每一个询问。

5 感谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队高闻远教练的指导。

感谢曾艺卿老师对我的培养与教导。

感谢父母对我的理解与支持。

感谢林荣恩同学、林弘扬同学、曾行周同学、任舍予同学对本文的帮助。

参考文献

- [1] 邓俊辉,《数据结构(C++语言版)(第3版)》,清华大学出版社。
- [2] IOI2020 中国国家候选队论文,《浅谈利用分散层叠算法对经典分块问题的优化》。
- [3] Bernard Chazelle and Leonidas J. Guibas, “Fractional Cascading: II. Applications”.

《遇到困难睡大觉》命题报告

成都市第七中学 魏衍芑

摘要

本文将主要介绍本人在校内训练中命制的一道传统题的解法和命题背景。

1 试题

题目描述

小 W 是个懒惰的 OIer，他正在打游戏。游戏有 n 个关卡，由于这是开放世界，他可以以任意顺序通关这个游戏。

每个关卡有一个轻松值和一个无聊值，越在后面打某个关卡，其轻松值就会越大。第 i 个关卡的初始轻松值是 a_i ，初始无聊值是 b_i 。这款游戏设计非常精妙，如果小 W 第 i 个通关的是 p_i 个关卡，那么他得到的轻松值是 $a_{p_i} + i \times k$ ，无聊值是 $b_{p_i} + i \times k$ 。而在整个游戏过程中，小 W 获得的轻松值是每个关卡轻松值的最小值，无聊值是每个关卡无聊值的最大值。

现在小 W 想要知道如何让自己获得的轻松值减去自己获得的无聊值最大。也即，找到一个排列 p ，使得 $\min_i(a_{p_i} + i \times k) - \max_i(b_{p_i} + i \times k)$ 最大。

输入格式

第一行一个正整数 n 表示游戏的关卡数，以及一个正整数 k ，意义如题面。

之后 n 行，每行两个正整数，分别表示 b_i, a_i （请注意这里的顺序）

输出格式

一行一个整数表示游戏过程中轻松值减去无聊值的最大值。

样例输入

2 10

0 15

5 20

样例输出

10

数据范围

对于所有数据，满足： $1 \leq a_i, b_i \leq 10^9, k \times n \leq 10^9, 1 \leq n \leq 10^5$

子任务 **1(5pts)**: $n \leq 20$

子任务 **2(5pts)**: $n \leq 100$

子任务 **3(10pts)**: $n \leq 1000$

子任务 **4(10pts)**: $n \leq 10000$

子任务 **5(20pts)**: $k \leq 100$

子任务 **6(10pts)**: $n \leq 50000$

子任务 **7(40pts)**: $n \leq 100000$

时间限制：**3s**

空间限制：**512M**

2 算法介绍

2.1 观察性质

首先不难发现的是答案只会与关卡的相对位置有关。那么不妨对所有关卡的下标做一个平移（形式化地说，可以让关卡位于下标 $t, t+1, \dots, t+n-1$ ，其中 t 是一个整数）。

枚举 $\max(i \times k + b_{p_i}) = M$ ，那么现在每个二元组 (a_i, b_i) 会有一个关于右端点位置的限制 $r_i = \lfloor \frac{M-b_i}{k} \rfloor$ 。由此同时不难发现关键的性质： r_i 的相对大小不随 M 改变。

为了方便，以下的下标都满足都满足对于任意 M ， r_i 始终单调不降（也即按照 $-b_i$ 排序二元组）。

2.2 算法一，算法二

如果将 $\max_i(b_{p_i} + i \times k)$ 所在的二元组移动到 0 下标，那么现在只会有 n 个不同的需要枚举的 M 。

现在的任务是在合法的限制 r 下将二元组填满 n 个连续的下标（必须包含 0），并且要让 $\min_i(a_{p_i} + i \times k)$ 最大。

考虑确定 t (t 的定义在前文已经给出) 后, 如何得到一个最优的排列:

每个限制是一段后缀, 所以可以考虑从后向前贪心, 维护当前位置能够填的二元组。每次贪心选择 a_i 最大的二元组。(以上贪心的复杂度是 $O(n \log n)$ 的)

但是同时我们可以得到另外一个本质相同的贪心方法: 将 a_i 从小到大排序, 依次确定这个二元组所在的位置, 从对应的 r_i 开始依次向前找第一个没有被占用的位置, 将这个二元组放在此位置上。

根据以上贪心, 可以得到以下结论:

Lemma 1. 对于任意 M , 选择最大的有解的 t 一定最优。

证明. 显然将 t 减少之后, 所有的 r_i 不会增加 (会有减少是因为如果 $r_i = t + n - 1$ 那么我们需要让 r_i 也跟着减少)。又不难发现的是第二个贪心的结果只与 r_i 有关, 又因为所有的 r_i 都不会增加, 所以显然不会让答案变得更优。□

由霍尔定理, 有 $t = \min(r_i - i + 1)$, 之后如果直接套用第一种贪心方案, 并用堆进行维护, 那么可以得到一个 $O(n^2 \log n)$ 的做法, 期望得分 20 分。

而第二种方案可以用并查集维护第一个没有被占用的位置, 所以可以做到 $O(n^2)$, 期望得分 30 分。

2.3 算法三

仔细考虑, 如果我们让 M 增加 k , 这个时候每个 r_i 也会增加 1, 不难发现 t 也会增加 1。不难发现这个时候 $\min_i(a_{p_i} + i \times k)$ 也只会增加 k 。

于是只需要考虑的是 $M \bmod k$ 的结果。

于是可以得到一个 $O(n \times k)$ 的做法, 结合上述算法, 期望得分 50 分。

2.4 算法四 - 第一部分

考虑二分答案 mid , 并且仍然考虑枚举 M (在 $[0, k)$ 的范围内), 看是否有解。

观察枚举 M 之后如何判断其是否有解:

首先类似 r_i 可以得到一个对左端点的限制 $l_i = \lceil \frac{mid - M - b_i}{k} \rceil$, 类似的, l_i 同样具有相对大小不变的性质。

由于已经确定了 t , 那么实际上的 l_i 是 $\max(\lceil \frac{mid - M - b_i}{k} \rceil, t)$, 实际上的 r_i 是 $\min(\lfloor \frac{M - b_i}{k} \rfloor, t + n - 1)$ 。

现在可以构造一个匹配的模型, 左右各 n 个点, 每个左边的点连向右边的一段区间 $[l_i + t + 1, r_i + t + 1]$ 。

关于判断匹配实际上可以用霍尔定理解决, 而由于每个左边的点连向右边的一个区间:

Lemma 2. 只需要检查是否存在一个区间 $[L, R] (L \leq R)$ 满足 $l_i \geq L, r_i \leq R$ 的二元组个数 $t > (R - L + 1)$, 就能知道是否存在完美匹配。

证明. 当找到一个左边的集合 s 的时候, 其连向的右边节点的集合 t 如果不是区间, 那么可以将 s 分成若干非空子集 s_1, s_2, \dots , 使得每个集合连向的右边节点的集合 t_i 都是 t 的一个极大区间。而如果 $|s| - |t| > 0$, 那么一定存在一个 $|s_i| - |t_i| > 0$ 。 \square

于是可以使用一个线段树解决此问题:

对于 R 做扫描线, 维护所有 $[L, R]$ 的答案, 遇到一个 $r_i = R$ 的二元组后将 $L \leq l_i$ 的所有位置区间加 1, 然后每次对前缀查询是否有 < 0 的位置即可。

只需要维护区间加以及区间的最小值即可完成这个问题。

但是这样做的复杂度有足足 $O(nk \log n \log A)$, 其中 A 是答案的范围 (在本题中是 10^9)

2.5 算法四 - 第二部分

现在考虑 M 从 0 变成 k 之后 l_i, r_i, t 的变化。

首先不难发现的是 l 和 r 的相对大小并不会改变 (前已证), 如果首先得出一个 l'_i 和 r'_i 以及 t' , 分别表示这些值在 l_i, r_i, t 的时候的值。

不难发现: $l'_i - 1 \leq l_i \leq l'_i, r'_i \leq r_i \leq r'_i + 1$, 并且都只会在 M 增加的过程中改变一次。以下我们分别记录其改变的时间戳 (记录为: tl_i, tr_i)。

仔细观察线段树在判断时的操作, 首先需要考虑的是一定是某一个 l_i 或者某一个 r_i 组成的区间, 并且每次操作影响到的区间是固定的。

这启发我们想办法将 k 次线段树的操作合并。

考虑一段区间 $[L, R]$, 现在是要检查 $R - L + 1 - s(L, R)$ 的权值是否小于 0, 其中 s 是区间内部的区间个数。而如果我们直接利用下标记录, 那么实际上是检查 $r_j - l_i + 1 - s(l_i, r_j)$ 。

不难发现, 由于 l, r 的相对大小不变, 改变的值只有 $r_j - l_i$ 。

考虑 $q(i, j) = r'_j - l'_i + 1 - s(l_i, r_j)$, 那么 $q(i, j) - 1 \leq r_j - l_i + 1 - s(l_i, r_j) \leq q(i, j) + 1$ 。

现在就是查询是否存在一个 M 使得对于所有 $1 \leq i, j \leq n, r_j - l_i + 1 - s(l_i, r_j) \geq 0$ 。

考虑维护 q , 进行分类讨论:

- 1 如果 $q(i, j) < -1$, 那么一定不存在合法的 k , 使得这个区间 $[l_i, r_j]$ 合法。
- 2 如果 $q(i, j) > 0$, 那么所有 k 都能使得这个区间 $[l_i, r_j]$ 合法。
- 3 如果 $q(i, j) = 0$, 那么当 $k < tr_j$ 并且 $k \geq tl_i$ 的时候, 这个区间不合法, 否则合法。
- 4 如果 $q(i, j) = -1$, 那么当 $k \geq tr_j$ 并且 $k < tl_i$ 的时候, 这个区间合法。

考虑后两个情况，我们发现在 tr_j 确定时，取最小的 tl 能获得最紧的限制。

同时由于不能出现 < -1 的情况，如果不存在情况 1，那么 $= 0$ 和 $= -1$ 的情况只能是最小和次小值。

可以考虑用类似第一部分中的线段树，每个结点维护区间最小值，次小值以及对应值下的最小 tl ，对 r' 做扫描线，在每次操作后询问最小值以及次小值，得到限制。

最终会有 $O(n)$ 个限制，对于 $q(i, j) = -1$ 得到的限制，可以轻松求出它们的交（是一个区间 $[L, R]$ ），然后对于 $q(i, j) = 0$ 的限制，可以从 L 扫到 R ，同时记录覆盖的最大右端点。

于是得到了在 $O(n \log n)$ 的时间完成对 mid 的判断。

总复杂度 $O(n \log n \log A)$ ，其中 A 是答案的范围。

3 命题契机与过程

本题是笔者在完成集训队作业中的《Scenery》一题时，错误地转化了题意得到的结果，当时笔者完成了一个 $O(n^2 \log n)$ 的做法（也即算法一）。

而之后在校内题目的命题过程中，笔者将题意中的“判断是否大于”改为了求最大，并且得到了算法四的做法。

4 总结与展望

本题是一个数据结构题，但其中涉及了二分图匹配的知的应用，这要求选手在完成此题时不仅对基础的数据结构技巧熟练掌握，还要对霍尔定理有一定了解。

完成本题需要从几个简单的结论出发，层层递进，不断发掘新的结论，将问题转化为二分图的判断问题并得到一个数据结构的解决方案。之后又要求选手对此数据结构的操作进行分析，从而得到在特殊情况下解决一组相似问题的方案。较长的解题步骤也要求选手有足够的耐心逐步分析。

5 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢蔺洋老师，林鸿老师，叶诗富老师以及父母多年来的关心和指导。

感谢王修涵同学为我验稿。

感谢 OI 路上支持和帮助我的所有人。

参考文献

- [1] 刘汝佳, 黄亮, 《算法艺术与信息学竞赛》, 清华大学出版社。
- [2] 刘汝佳, 《算法竞赛入门经典》, 清华大学出版社。

浅谈有限状态自动机及其应用

杭州学军中学 徐哲安

摘要

有限状态自动机是目前信息学竞赛中相对冷门的内容。随着时代的进步，有限状态自动机相关的内容越来越多地作为难题出现在各类算法竞赛中。此外，学习有限状态自动机相关的内容有助于我们深入理解某些问题与算法的本质。本文系统地简述了有限自动机的基础理论及算法，引出了一个正则表达式匹配的高效算法，以及有限状态自动机在信息学竞赛中的丰富应用。希望本文能推动有限状态自动机在信息学竞赛界的普及。

1 引言

计算理论是理论计算机科学的分支，也是一门本科生课程。计算理论研究在某种计算模型下，怎样的问题是可解决的，通过算法能多少有效地解决这个问题。目的是回答：计算机的基本能力和局限是什么？

有限状态自动机作为一种基础的计算模型，因其结构简单，相关理论成熟的特点，不仅在现实生活中有广泛的应用，而且更能协助我们解决各类信息学竞赛中的问题，具有广泛的应用前景。

但令人遗憾的是，目前在信息学竞赛界缺乏相关的系统的学习资料，目前仅有的资料是一些信息学竞赛活动中的讲课课件 [5, 6, 7]。由此，作者对有限状态自动机进行了深入的探究，将相关的理论加以梳理，并总结了有限状态自动机在信息学竞赛中的丰富应用，写作本文。

本文共由六个章节组成。

第二节主要介绍了两类有限状态自动机——确定性有限状态自动机与非确定性有限状态自动机，与正则语言。本节揭示了两类有限状态自动机在计算能力上的等价性，以及它们之间的不同之处，并介绍了它们计算的算法，提出了具有启发意义的优化方法。

第三节主要介绍了正则语言的另一种表达——正则表达式，以及正则语言具有的大量性质。

第四节主要介绍了确定性有限状态自动机的最小化，揭示了正则语言与确定性有限状态自动机更加本质的关系。本节也介绍了用于确定性有限状态自动机最小化的高效算法——Hopcroft 算法。

第五节主要介绍了有限状态自动机的相关理论在信息学竞赛中的应用。本节内容是作者深入探究有限状态自动机的相关理论，并对各类算法竞赛中的相关试题加以研究后整理得到的。作者引出了一个正则表达式匹配的高效算法，并提出了一些个人的见解。

第六节作简要的总结。

2 有限状态自动机与正则语言

2.1 确定性有限状态自动机

有限状态自动机 (*Finite State Machine*, FSM) 是最简单的一类计算模型，体现在它的描述能力与资源都极其有限。我们首先给出确定性有限状态自动机的形式化定义。

定义 2.1 确定性有限状态自动机 (*Deterministic Finite Automaton*, DFA) 是一个五元组 $(Q, \Sigma, \delta, q_0, F)$ ，其中

- Q 是一个有限状态集合；
- Σ 是一个有限字符集；
- $\delta: Q \times \Sigma \rightarrow Q$ 是转移函数；
- $q_0 \in Q$ 是开始状态；
- $F \subset Q$ 是接受状态集合。

我们可以使用状态图来直观地描述一个 DFA。

要让 DFA 发挥作用，我们也需要给出另一些定义。

定义 2.2 设 $M = (Q, \Sigma, \delta, q_0, F)$ 是一个 DFA， $w = w_1 w_2 \cdots w_n$ ($w_i \in \Sigma$) (也记作 $w \in \Sigma^*$) 是一个串，若存在 Q 中的状态序列 r_0, r_1, \cdots, r_n 满足

- $r_0 = q_0$ ；
- $\delta(r_i, w_{i+1}) = r_{i+1}$, $i \in \{0, 1, \cdots, n-1\}$ ；
- $r_n \in F$ 。

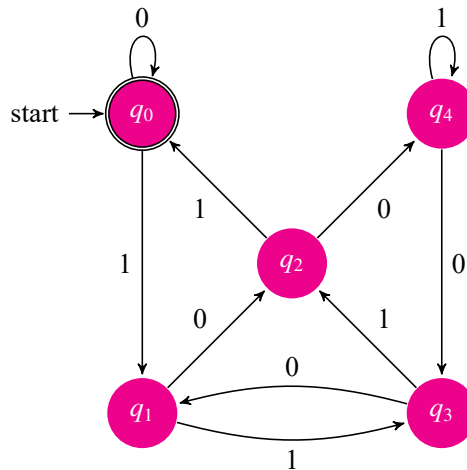
则称 M 接受 w 。

定义形式语言 (简称语言) 是任意 Σ 上串的集合 (集合大小可以为无限)。令语言 $L(M) = \{w \mid M \text{ 接受 } w\}$ ，则称 M 识别 $L(M)$ 。

如果一个语言能被某个 DFA 识别，则称它为正则语言 (*Regular Language*)。

为了方便说明，我们将求出输入串 w 在 DFA 中的状态序列，并判断其是否被接受的过程称为计算。

考虑这样一个例子：设计一个 DFA 能识别 5 的倍数的二进制串。注意到，我们只需要考虑已读入的串模 5 后的余数，由此构造状态 q_0, q_1, q_2, q_3, q_4 。若当前在状态 q_r 读入字符 c 后，转移到状态 $q_{(2r+c) \bmod 5}$ 即可。此外，开始状态和接受状态均为 q_0 。



2.2 非确定性有限状态自动机

非确定性是确定性的自然推广，在非确定性机中，任何一个点和某个转移字符可能存在多个后继。下面我们用 $\mathcal{P}(Q)$ 表示 Q 的幂集（所有子集的集合），令 $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ (ϵ 表示空串，即在串中能任意加删)，并给出非确定性有限状态自动机的形式化定义。

定义 2.3 非确定性有限状态自动机 (Nondeterministic Finite Automaton, NFA) 是一个五元组 $(Q, \Sigma, \delta, q_0, F)$ ，其中

- Q 是一个有限状态集合；
- Σ 是一个有限字符集；
- $\delta: Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$ 是转移函数；
- $q_0 \in Q$ 是开始状态；
- $F \subset Q$ 是接受状态集合。

我们同样给出 NFA 计算的形式化定义，需要注意，只要最终任何一个状态是接受状态，整个输入串就会被接受。

定义 2.4 设 $N = (Q, \Sigma, \delta, q_0, F)$ 是一个 NFA, 串 w 可以被表示为序列 $y_1 y_2 \cdots y_m$ ($y_i \in \Sigma_\epsilon$), 若存在 Q 中的状态序列 r_0, r_1, \cdots, r_m 满足

- $r_0 = q_0$;
- $r_{i+1} \in \delta(r_i, w_{i+1}), \quad i \in \{0, 1, \cdots, m-1\}$;
- $r_m \in F$.

则 N 接受 w 。

NFA 是 DFA 的扩展, 似乎一个显然的推论是 NFA 比 DFA 能力强, 能识别更多的语言类?

2.3 DFA 与 NFA 的等价性

实则不然, 下面我们将说明 DFA 与 NFA 的计算能力等价性。

定理 2.1 每一个 NFA 都等价于某一个 DFA, 反之亦然。两个机器等价, 即它们能识别的语言类相同。

证明. 显然每个 DFA 都可以直接转换为一个 NFA, 考虑将 NFA 转化为一个模拟它的 DFA。我们使用一种被称作**幂集构造** (*Powerset construction*) 的方法, 下面给出形式化描述。

设 $N = (Q, \Sigma, \delta, q_0, F)$, 定义 $E(q)$ 表示从状态 q 出发, 只沿 ϵ 转移能到达的状态集合。

我们构造 $M = (Q', \Sigma, \delta', E(q_0), F')$, 其中

- 状态集合: $Q' = \mathcal{P}(Q)$;
- 转移函数: $\delta' : Q' \times \Sigma \rightarrow Q'$;
- 其中 $\delta'(S, c) = \bigcup_{q \in S, q' \in \delta(q, c)} E(q')$;
- 终止状态集合: $F' = \{S \subset Q \mid S \cap F \neq \emptyset\}$ 。

显然计算的每一步, M 所在的状态对应 N 所处的状态集合。 □

鉴于 DFA 与 NFA 的计算能力等价性, 我们可以得到一个简单推论。

推论 2.1 一个语言是正则的, 当且仅当存在一个 NFA 能识别它。

虽然 NFA 与 DFA 能力相同, 但我们认为 NFA 是有用的。这是因为对于某些正则语言, 用 NFA 表示所需的状态数远小于 DFA 所需的状态数。而且, 我们也不难构造出一个状态数为 n 的 NFA 使得它对应的最小 DFA 状态数是 $\Theta(2^n)$ 的。

2.4 DFA 与 NFA 的计算

两类 FSM 的差异还体现在计算一个给定串的时间复杂度上。接下来，我们设串长为 n ，FSM 状态数为 s ，字符集大小为常数。

由于状态和转移的唯一性，显然 DFA 计算这个串的时间复杂度为 $O(n)$ 。为了方便处理 NFA 的转移，我们可以先将 ϵ 转移消除。在 NFA 的计算过程中，最坏可能包含所有 s 个状态，同时每个状态可能至多存在 s 个后继，所以其时间复杂度为 $O(ns^2)$ 。接下来，我们将给出两个有用的优化。

一是通过 **bitset** 来优化复杂度。具体来说，预处理出每个状态 u 沿 c 转移边会到达的集合，NFA 计算过程中求出新的状态集合就只需要将这些集合或起来即可。时间复杂度被优化为 $O(\frac{ns^2}{\omega})$ 。

二是使用 **Method of Four Russians**。将 NFA 的状态分块，设块大小为 T ，全体状态被分成了 $O(\frac{s}{T})$ 块。对于每个块，我们枚举 2^T 种状态子集，再枚举所有转移 c ，处理出这个集合沿 c 转移边所形成的新的状态集合。在 NFA 计算过程中，我们只需要利用每个块内预处理的结果就能求出新的状态集合。时间复杂度为 $O(\frac{s^2 2^T}{\omega T} + \frac{ns^2}{\omega T})$ ，取 $T = O(\log n)$ 即可做到 $O(\frac{ns^2}{\omega \log n})$ 的时间复杂度了。

3 正则表达式与正则语言

3.1 正则表达式

正则表达式是另一种常用的正则语言表达，下面给出形式化定义。

定义 3.1 对于一个正则表达式 (*Regular Expression*) R ，称 $L(R)$ 为正则表达式 R 对应的形式语言。正则表达式 R 可以是

1. $c (c \in \Sigma)$ ，表示语言 $L(R) = \{c\}$ ；
2. ϵ ，表示语言 $L(R) = \{\epsilon\}$ ；
3. \emptyset ，表示语言 $L(R)$ 为空语言；
4. $(R_1 + R_2)$ ，表示语言 $L(R) = L(R_1) \cup L(R_2)$ ；
5. $(R_1 R_2)$ ，表示语言 $L(R) = \{uv \mid u \in L(R_1), v \in L(R_2)\}$ ；
6. (R_1^*) ，表示语言 $L(R) = \{u_1 u_2 \cdots u_n \mid u_i \in L(R_1), n \in \mathbb{N}\} \cup \{\epsilon\}$ 。

3.2 正则表达式与 FSM 的等价性

看上去正则表达式与 FSM 相当地不同，但是，它们所能表述的语言类是相同的。

定理 3.1 一个语言是正则的，当且仅当可以用正则表达式表示它。

这个定理有两个方向，我们可以写作两个引理分别加以证明。

引理 3.1 如果一个语言可以用正则表达式描述，那么它是正则的。

证明. 发现我们只需要将一个正则表达式转化为一个 NFA 即可。我们可以使用 **Thompson 构造法** (*Thompson's construction*)，根据正则表达式 R 的构成，有如下六种情况：

1. 若 $R = c (c \in \Sigma)$,



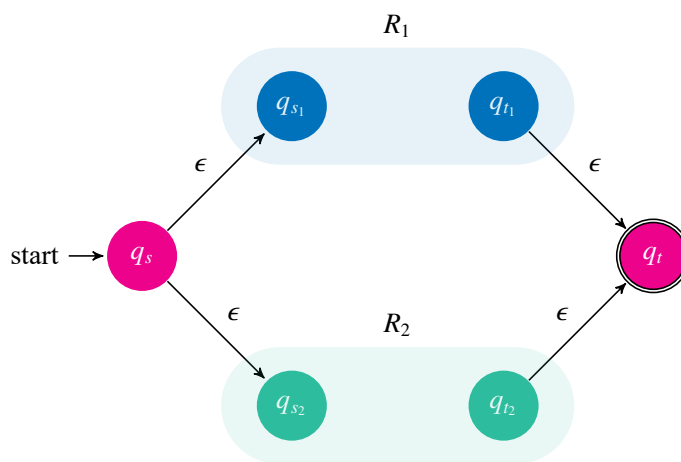
2. 若 $R = \epsilon$,



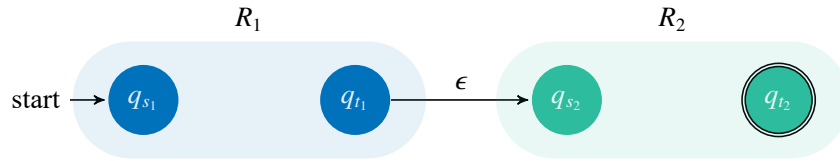
3. 若 $R = \emptyset$,



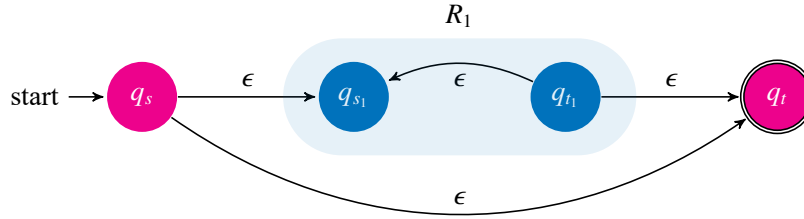
4. 若 $R = (R_1 + R_2)$,



5. 若 $R = (R_1R_2)$,



6. 若 $R = (R_1^*)$,



□

引理 3.2 如果一个语言是正则的，那么可以用正则表达式描述它。

证明. 显然我们只需要找到一种将一个 DFA 转化为正则表达式的方法，首先引入一种新型 FSM。

定义 3.2 广义非确定性有限状态自动机 (*Generalized Nondeterministic Finite Automaton, GNFA*) 是一个五元组 $(Q, \Sigma, \delta, q_s, q_t)$ ，其中

1. Q 是一个有限状态集合；
2. Σ 是一个有限字符集；
3. $\delta : (Q \setminus \{q_t\}) \times (Q \setminus \{q_s\}) \rightarrow \mathcal{R}$ 是转移函数，即每条边上是一个正则表达式；
4. $q_s \in Q$ 是开始状态；
5. $q_t \in Q$ 是接受状态。

如果串 w 可写作 $w = w_1 w_2 \cdots w_k$ ($w_i \in \Sigma^*$)，且存在状态序列 q_0, q_1, \cdots, q_k 使得

1. $q_0 = q_s$ 是开始状态；
2. $q_k = q_t$ 是接受状态；
3. $\forall i = 1, 2, \cdots, k, w_i \in L(\delta(q_{i-1}, q_i))$ 。

则称这个 GNFA 接受串 w 。

我们可以通过增加开始状态，接受状态，以及一些必要的转移箭头，使得一个 DFA 转化为等价的 GNFA。

假设当前的 GNFA 存在 k 个状态，且 $k > 2$ 。我们任取一状态 $q_r \in Q \setminus \{q_s\} \setminus \{q_t\}$ ，并构造一个新的 GNFA $(Q', \Sigma, \delta', q_s, q_t)$ 。其中 $Q' = Q \setminus \{q_r\}$ ，对于任意 $q_i \in Q' \setminus \{q_t\}, q_j \in Q' \setminus \{q_s\}$ ，我们令

$$\delta'(q_i, q_j) = ((\delta(q_i, q_r)\delta(q_r, q_r)^*\delta(q_r, q_j)) + (\delta(q_i, q_j)))$$

考虑串在原来 GNFA 中的路径会经过若干次 q_r ，我们通过构造使得它被转移边等效替代了。所以新的 GNFA 与原来的 GNFA 等价，并且状态数恰好减一。

通过不断减少 GNFA 的状态数，我们能使得最终的 GNFA 状态数为 2，此时 q_s 到 q_t 转移边上的正则表达式就是我们的目标。

□

建立起了正则表达式与正则语言之间的联系，就使得我们可以从代数视角考虑正则语言运算的性质。

3.3 正则语言的代数定律

在本小节中，我们不考虑具体的正则表达式，转而考虑以变量为参数的正则表达式（变量可以为任意正则语言）。运用正则表达式的代数定律有助于化简正则表达式。

定理 3.2 以下正则表达式的代数定律成立：

1. 并的交换律： $L + M = M + L$;
2. 并的结合律： $(L + M) + N = L + (M + N)$;
3. 连接的结合律： $(LM)N = L(MN)$;
4. \emptyset 是并运算的单位元： $\emptyset + L = L + \emptyset = L$;
5. ϵ 是连接运算的单位元： $\epsilon L = L\epsilon = L$;
6. \emptyset 是连接运算的零元： $\emptyset L = L\emptyset = \emptyset$;
7. 分配律： $L(M + N) = LM + LN, (M + N)L = ML + NL$;
8. 并的幂等律： $L + L = L$;
9. 闭包相关的定律： $(L^*)^* = L^*, \emptyset^* = \epsilon, \epsilon^* = \epsilon$ 。

以上定律的证明均较为简单，故在此略去。

由此延伸出这样一个问题：我们如何判断一个正则表达式的“定律”是否成立？

接下来给出了一个一般性的方法，但非常有趣的是，它紧密依赖于正则表达式的运算符性质，不能推广到含某些其他运算符（例如交运算）的表达式。我们给出一个重要引理：

引理 3.3 设 E 是包含变量 L_1, L_2, \dots, L_m 的正则表达式。将所有 L_i 替换为标志符 a_i ，得到形式正则表达式 $T(E)$ 。

令 $L_1 = L'_1, L_2 = L'_2, \dots, L_m = L'_m$ （注意 L_i 是变量，而 L'_i 是具体的语言），得到具体的正则表达式 E' 。我们可以按如下方式构造全体 $L(E')$ ：对于任意 $A = a_{j_1} a_{j_2} \dots a_{j_k} \in L(T(E))$ ，我们将 a_{j_i} 替换为 L'_{j_i} 中的任意元素，并将所有得到的串加入到 $L(E')$ ，记这些串的集合为 $w(A)$ 。

证明. 考虑对正则表达式进行归纳，不包含运算符的基础情况是显然的。

- 若 $E = F+G$ ，根据定义有 $T(E) = T(F)+T(G)$ 。对于任意串 $w \in w(A)$ ($w(A) \subset L(E')$)，存在对应的 $A \in L(T(E))$ 有 $A \in L(T(F))$ 或者 $A \in L(T(G))$ ，也即 $w \in L(F')$ 或者 $w \in L(G')$ 。
反之，对于 $w \in L(F')$ ，存在对应的 $A \in L(T(F))$ ，那么 $A \in L(T(E))$ 即 $w \in L(E')$ 。
- 若 $E = FG$ ，根据定义有 $T(E) = T(F)T(G)$ 。对于任意串 $w = uv$ ， $u \in w(A), v \in w(B)$ ($w(A) \subset L(F'), w(B) \subset L(G')$)，它们存在对应的 $A \in L(T(F)), B \in L(T(G))$ 有 $u \in L(F'), v \in L(G')$ 。
反之，对于 $u \in L(F'), v \in L(G')$ ，存在对应的 $A \in L(T(F)), B \in L(T(G))$ ，那么 $AB \in L(T(E))$ 即 $uv \in L(E')$ 。
- 若 $E = F^*$ ，其证明也是类似的，这里不再赘述。

□

如果扩展正则表达式引入交运算，可以发现上述的论证不再成立，也就不再适用接下来的定理。这里举出一个简单的例子，对于含变量的正则表达式 $E = L_1 \cap L_2$ ，有 $T(E) = a_1 \cap a_2 = \emptyset$ 。而如果我们令正则语言 $L'_1 = \{a, b\}, L'_2 = \{a, c\}$ ，带入 $L_1 = L'_1, L_2 = L'_2$ ，得到 $L(E') = \{a\}$ ，而使用上述过程会得到空语言。

定理 3.3 对于一对带有相同变量集合的正则表达式 E 和 F 。将每个变量替换为一个标识符，得到 $T(E), T(F)$ ，那么 $E = F$ 当且仅当 $L(T(E)) = L(T(F))$ 。

证明. 我们要证明：对于任意替换变量为具体语言的方案，均有 $L(E') = L(F')$ 当且仅当 $L(T(E)) = L(T(F))$ 。

假设对于所有替换选择，均有 $L(E') = L(F')$ 。那么替换为标识符也是一种替换选择，令 $E = T(E), F = T(F)$ ，即 $L(T(E)) = L(T(F))$ 。

假设 $L(T(E)) = L(T(F))$ ，根据引理 3.3，将 $L(T(E)), L(T(F))$ 中的标识符替换为对应的语言，就构造出了 $L(E')$ 和 $L(F')$ ，显然它们是相等的。 □

定理 3.3 为我们提供了一种判断两个含变量的正则表达式是否相同的方法。只需要不断将同一变量替换为未曾出现过的字符，将它们变成具体的正则表达式，再判断两个具体的正则表达式对应的语言是否相同。这个问题我们将在后续的章节中解决。

3.4 正则语言的封闭性

正则语言的封闭性是其重要的性质，这些性质使得我们能通过一定的运算，构造能够识别另一些语言的 FSM。简而言之，封闭性可以作为构造复杂 FSM 的工具。

定理 3.4 关于正则语言的封闭性，我们有：

1. 两个正则语言的并是正则的；
2. 两个正则语言的连接是正则的；
3. 正则语言的闭包是正则的；
4. 正则语言的补是正则的；
5. 两个正则语言的交是正则的；
6. 两个正则语言的差是正则的；
7. 正则语言的反转是正则的；
8. 正则语言的同态（相同串替代符号）是正则的；
9. 正则语言的逆同态是正则的。

这里简单说明这些性质的正确性。前三条性质就是正则表达式的运算符；性质四只需要将接受状态取作补集；性质五六可以构造两个 DFA 的笛卡尔积得到；性质七可以认为是反转所有转移边的方向，构造 NFA 得到。性质八九的证明较为繁琐，以上这些性质的详细证明都能在 [1] 中找到，此处略去。

3.5 正则语言的泵引理

引理 3.4 正则语言的泵引理 设 L 是正则语言，存在与 L 相关的常数 n 满足：对于任意 L 中的串 w ，如果 $|w| \geq n$ ，则我们能将 w 表示为 xyz ，满足：

1. $y \neq \epsilon$ ；
2. $|xy| \leq n$ ；

3. $\forall k \geq 0, xy^kz \in L$ 。

证明. 对于任意正则语言 L , 存在某个 DFA 使得 $L = L(A)$ 。假设 A 有 n 个状态, 考虑长度不小于 n 的串 $w = a_1a_2 \cdots a_m$ 其中 $m \geq n$ 且 a_i 均为输入符号。对于 $i = 0, 1, \dots, n$ 定义 p_i 为读入 w 前 i 个字符后所处的状态。

根据抽屉原理, 必然存在 $0 \leq i < j \leq n$ 使得 $p_i = p_j$, 我们构造

1. $x = a_1a_2 \cdots a_i$;
2. $y = a_{i+1}a_{i+2} \cdots a_j$;
3. $z = a_{j+1}a_{j+2} \cdots a_m$;

发现串 y 对应的路径构成了一个环, 如果我们在环上走 k 次, 就能构造出串 xy^kz 。 □

泵引理描述了正则语言一个共有的基本属性。对于一个语言, 如果我们能找到一个足够长的串使其不满足泵引理, 就足以说明这不是正则语言。泵引理是我们证明一个语言的非正则性的有力工具。不过需要注意的是, 泵引理的逆命题并不成立, 也即存在某些非正则语言满足泵引理。

4 DFA 的等价类与最小化

我们定义两个 DFA 等价当且仅当它们识别相同的正则语言, 显然, 全体 DFA 被划分了无穷多个等价类。本节介绍了一些通用的算法, 来寻找某个 DFA 所属等价类中的最小 DFA。

4.1 DFA 的最小化

定义 4.1 对于 DFA 的两个状态 p, q , 我们定义关系 $p \sim q$ 当且仅当: 对于任意输入串 $w = w_1w_2 \cdots w_k$ ($k \geq 0$), $\hat{\delta}(p, w) = \delta(\cdots \delta(\delta(p, w_1), w_2) \cdots, w_k)$ 是接受状态当且仅当 $\hat{\delta}(q, w)$ 是接受状态。

从直观上理解, 关系 $p \sim q$ 相当于不存在输入串 w 能区分状态 p 和 q 。稍加观察不难发现, 关系 \sim 构成了一个**等价关系** (自反性与对称性显然, 传递性易反证得到)。也就是说对于一个特定的 DFA, 等价关系 \sim 将 DFA 的状态划分为若干个等价类。

剔除初始状态无法到达的状态后, 假设我们得到了 DFA A 的等价类 S_1, S_2, \dots, S_m , 考虑构造一个新的 DFA B 。我们将 A 中的等价类 S_i 作为 B 中的状态 i , 那么 B 恰好包含 m 个状态一一对应 A 中的等价类。对于等价类 S_i , 若存在状态 $u \in S_i$ 和转移 c , 使得 $\delta_A(u, c) \in S_j$, 则对于任意状态 $v \in S_i$ 均有 $\delta_A(v, c) \in S_j$ (使用反证法易得)。那么, 我们不妨令 B 中的转移 $\delta_B(i, c) = j$ 。显然, 对于 S_i 中的所有状态, 要么均为接受状态, 要么均不是接受状态。若 S_i 中的状态均为接受状态, 我们令 B 中的状态 i 为接受状态, 否则不是接受状态。最后, 我们

令包含 A 的初始状态的等价类 S_i 对应的状态 i 为 B 的初始状态。不难验证，我们构造出的 DFA B 与 DFA A 等价。

那么，我们剩下的问题就是找出一个 DFA A 所有的等价类。

接下来的算法基于不断对等价类进行划分。我们扩展等价关系 \sim 的定义，令 $p \sim_k q$ 表示对于任意长度 $\leq k$ 的串 w ，均有 $\hat{\delta}(p, w)$ 是接受状态当且仅当 $\hat{\delta}(q, w)$ 是接受状态。不难发现 \sim_k 也是一个等价关系。由关系 \sim_k ，我们的 DFA A 被划分成了等价类集合 Π_k 。显然 $\Pi_0 = \{Q \setminus F, F\}$ ，我们也不难从 Π_k 推出 Π_{k+1} ，具体算法如下：

Algorithm 1: 等价类划分算法

Input: DFA $A = (Q, \Sigma, \delta, q_0, F)$

Output: 等价类集合 $\Pi_0, \Pi_1, \Pi_2, \dots$

```

1  $\Pi_0 \leftarrow \{Q \setminus F, F\}$ ;
2  $m \leftarrow 0$ ;
3 repeat
4    $\Pi_{m+1} \leftarrow \Pi_m$ ;
5   foreach  $c \in \Sigma$  do
6      $\Pi' \leftarrow \Pi_{m+1}$ ;
7     foreach  $G \in \Pi_{m+1} \wedge |G| > 1$  do
8       if  $\exists u, v \in G, \delta(u, c)$  与  $\delta(v, c)$  属于  $\Pi_m$  的不同组 then
9         根据转移后不同的组，对  $G$  进一步划分得到  $G^*$ ;
10         $\Pi' \leftarrow \Pi' \setminus \{G\} \cup G^*$ ;
11      end
12    end
13     $\Pi_{m+1} \leftarrow \Pi'$ ;
14  end
15   $m \leftarrow m + 1$ ;
16 until  $\Pi_m = \Pi_{m-1}$ ;
```

上述算法依次求出了等价类集合 $\Pi_0, \Pi_1, \dots, \Pi_m$ ，其中 $\Pi_{m-1} = \Pi_m$ 。由于划分方案仅依赖于前一个等价类集合，我们断言 $\Pi_{m-1} = \Pi_m = \Pi_{m+1} = \Pi_{m+2} = \dots$ ，根据定义， Π_m 就是我们所求得等价类集合。值得一提的是，如果我们稍加修改上述算法，每次找出任意一个能被分裂的集合并不断迭代，不难证明这同样也是正确（首先证明等价的一对状态不会被分开；再考虑最小的 k 使得存在一对处于同一等价类中且不满足 \sim_k 的状态，一定能继续划分）。

在具体实现的时候，对于每种转移 c ，我们遍历所有等价类集合，再遍历其中的元素。我们使用一个 Hash 表来维护当前等价类中的 u 对应的 $\delta(u, c)$ 所在 Π_m 中的等价类，据此来继续划分集合。该算法的时间复杂度为 $O(n^2|\Sigma|)$ 。值得注意的是，根据 DFA 随机生成的方式不同，该算法的平均迭代次数能达到 $O(\log n)$ 甚至更低。这是一个实践中非常优秀的算法。

4.2 Myhill-Nerode 定理

给定一个语言 L ，对于任意一对串 x, y ，我们定义关系 $x \equiv_L y$ 当且仅当：对于任意串 z ， $xz \in L$ 当且仅当 $yz \in L$ 。不难验证这是一个等价关系，也即 Σ^* 被划分为了若干等价类。我们有定理：

定理 4.1 Myhill-Nerode theorem 语言 L 是正则的，当且仅当关系 \equiv_L 对应的等价类个数有限。描述正则语言 L 的最小 DFA 唯一（忽略状态标号），且它的状态数量等于关系 \equiv_L 对应的等价类数量。

该定理包含两个方向，那么我们同样分作两个引理分别证明。

引理 4.1 对于任意正则语言 L ，都有关系 \equiv_L 对应的等价类个数有限，且不超过能识别 L 的最小 DFA 状态数。

证明. 取一个识别语言 L 的最小 DFA A ，假设关系 \equiv_L 对应的等价类个数大于 A 的状态数。根据抽屉原理，必然存在不属于同一个等价类的串 x, y 会到达 A 中的同一个状态，也即 $x \equiv_L y$ ，矛盾。 \square

引理 4.2 对于任意语言 L ，关系 \equiv_L 对应的等价类个数有限，我们都能构造唯一的 DFA 使得其状态数等于等价类个数。

证明. 考虑构造一个 DFA A 使得每个等价类恰好对应一个状态。DFA 的开始状态对应空串所属的等价类，DFA 的接受状态对应全体元素属于 L 的等价类。对于每个等价类和字符 c ，任意选择一个代表元求出沿 c 转移边到达的等价类，在这对等价类对应的状态之间加入 c 转移边。不难验证这个 DFA 识别语言 L 。

假设存在另一个与 A 不同构的 DFA B 识别 L 且它的大小不超过等价类个数。那么，必然存在不属于同一个等价类的串 x, y 会到达 B 中的同一个状态，也即 $x \equiv_L y$ ，矛盾。 \square

显然最小化后的 DFA 的状态一一对应等价类，所以最小化后的 DFA 就是最小 DFA。

Myhill-Nerode 定理也为我们提供了新的判断一个语言非正则性的方法，显然有推论

推论 4.1 对于一个语言 L ，若存在任意多个串 w_1, w_2, \dots 使得它们两两不等价，那么 L 不是正则语言。

由于最小 DFA 的唯一性，我们也找到了判断两个 DFA 是否等价的方法。只需将两个 DFA 分别最小化，判断是否同构即可。

4.3 Hopcroft 算法

Hopcroft 算法 是一个寻找 DFA 等价类更加高效的算法 [3]，基于启发式分裂来优化时间复杂度。算法描述如下：

Algorithm 2: Hopcroft's algorithm**Input:** DFA $A = (Q, \Sigma, \delta, q_0, F)$ **Output:** 等价类集合 P

```

1  $P \leftarrow \{F, Q \setminus F\};$ 
2  $W \leftarrow \{F\};$ 
3 while  $W \neq \emptyset$  do
4   从  $W$  中任取一个集合  $A$ , 并将其从  $W$  中删去;
5   foreach  $c \in \Sigma$  do
6      $X \leftarrow \{u \in Q \mid \delta(u, c) \in A\};$ 
7     foreach  $Y \in \{u \in P \mid u \cap X \neq \emptyset, u \setminus X \neq \emptyset\}$  do
8        $P \leftarrow P \setminus \{Y\} \cup \{Y \cap X\} \cup \{Y \setminus X\};$ 
9       if  $Y \in W$  then
10         $W \leftarrow W \setminus \{Y\} \cup \{Y \cap X\} \cup \{Y \setminus X\}$ 
11       else
12         if  $|Y \cap X| \leq |Y \setminus X|$  then
13            $W \leftarrow W \cup \{Y \cap X\};$ 
14         else
15            $W \leftarrow W \cup \{Y \setminus X\};$ 
16         end
17       end
18     end
19   end
20 end

```

如何证明上述算法的正确性？稍加修改后的等价类划分算法可以看做是以下过程的不断重复：选择等价类 Y, A 以及字符 c ，求出能沿着 c 转移边到达 A 的状态集合 X ，满足 $Y \cap X$ 与 $Y \setminus X$ 均非空，那么我们就将等价类 Y 拆分为两个等价类。我们称等价类 A 是划分等价类 Y 的**证据**。Hopcroft 算法实际上维护了一个集合 W 表示还未用于划分等价类的证据。显然用证据 A 将现有的等价类划分完后，证据 A 就失去了意义。

在我们将等价类 Y 划分为等价类 U, V 之后有两种情况。一种是 Y 还未作为证据使用，显然 Y 的作用可以被 U 和 V 共同作用代替。另一种是 Y 已经作为证据使用，我们可以证明，仅保留 U 或者 V 作为证据使用均能得到正确的结果。假设存在等价类 Z 使得 U, V 中仅有 U 能作为证据将其继续划分。那么必然存在一对状态 $a, b \in Z$ 和转移 c 使得 $\delta(a, c) \in U, \delta(b, c) \notin U$ 。如果 $\delta(b, c) \notin Y$ ，那么 Y 作为证据一定会将 Z 继续划分，否则 $\delta(b, c) \in V$ 即 V 同样能作为划分 Z 的证据。矛盾，故结论成立。

接下来的问题是，如何高效实现该算法？不难发现，时间复杂度的瓶颈在于快速寻找到

与 X 有交的 Y 。我们不妨遍历 X 中的所有元素，在其所属的等价类打上标记，最后将所有符合条件 Y 进一步处理。如果 $|Y \cap X|$ 较小，那么我们将 $Y \cap X$ 单独提出，并保留这些元素在原来的集合中（之后再次遍历到这个集合时就忽略这些元素，并重构整个集合），否则意味着遍历集合 Y 的时间复杂度上界为 $O(|X|)$ 。注意到时间复杂度依赖于所有 $|X|$ 与 $\min(|Y \cap X|, |Y \setminus X|)$ 之和，因为每个元素对时间复杂度产生贡献时，其所在等价类规模至少除以 2，故 Hopcroft 算法的时间复杂度为 $O(n \Sigma \log n)$ 。对于某些特定方式随机生成的 DFA，该算法的平均时间复杂度能达到 $O(n \Sigma \log \log n)$ 。

5 FSM 在 OI 中的应用

虽然 FSM 是 OI 中相对冷门的内容，但是学习 FSM 有助于我们深入了解一些问题的本质，以及提出更加优秀的算法。接下来，我们将略举几例来展现其丰富的应用。

5.1 DFA 的构造与设计

在 OI 中，我们有时会遇到这样的计数问题：“给定一个串，问其存在多少子串满足某条件”，或者“给定一个包含 0, 1, ? 的串，问存在多少种替换 ? 为 0 或者 1 的方案使其满足某条件”等等。如果满足条件的全体串构成了一个正则语言，我们往往能通过某种方式来构造一个 DFA，接着再通过 DP 解决。

为了解决这样的问题，我们需要设计一个 DFA 来识别特定的语言。没有什么机械的方法或者公式能帮助我们直接构造出一个 DFA，但是，在设计 DFA 的时候仍然有一般性的原则。

DFA 的局限在于它的状态数必须是有限的，而可能的输入种类却是无限的。这就需要我们用某些**关键信息**来描述状态，使得所有输入被划分为有限类。第二节提到的例子就是将已读入的数模 5 作为关键信息。

例题 5.1 Foreigner¹ 我们定义好数：

- 所有 1 至 9 的数是好的；
- 如果一个整数 $x \geq 10$ 是好的，它需要 $\lfloor x/10 \rfloor$ 是好的；除此之外，设 $\lfloor x/10 \rfloor$ 是第 k 个好数，那么 x 是好的仅当 $x \bmod 10 < k \bmod 11$ 。

前若干个好数为 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 21, 30, 31, 32...

给定数字串 s ，求有多少个 s 的子串不含前导 0 且是好数。

数据范围： $1 \leq |s| \leq 10^5$

¹Codeforces Round #549 (Div. 1), Problem D

考虑构造一个 DFA，使之能识别全体好数构成的集合。对于一个好数 x ，我们维护三个参数： a_x 表示小于等于 x 的好数个数， b_x 表示小于 x 且与 x 长度相同的好数个数， c_x 表示大于 x 且与 x 长度相同的好数个数。在 x 后追加一个数字得到 y ，我们不难判断 y 是否是一个好数，同时容易得到新的 a_y, b_y, c_y 。利用性质 $(0 + 1 + \dots + 10) \bmod 11 = 0$ ，我们只需要知道 a_x, b_x, c_x 模 11 的值。由此，我们可以构造一个包含 11^3 个状态的 DFA，在这个 DFA 上 DP 就能解决本题。值得注意的是，这个 DFA 有很多状态是不可达的，删去不可达的状态可以大大提高程序的效率。

事实上，上述做法构造的 DFA 包含很多冗余、重复的状态，我们能通过进一步分析问题性质得到更优的解法。方便起见，我们认为 0 也是好数，同时它的排名为 0。观察好数序列发现从 10 开始所有的好数都是在更小好数的末尾添加 0, 1, \dots , 9 得到的，同时能添加上的数字个数是 0, 1, \dots , 10 循环的。假设好数 x 与 $\lfloor x/10 \rfloor$ 的排名分别为 u, v ，那么有

$$u = 9 + 55\lfloor v/11 \rfloor + (0 + 1 + \dots + (v - 1) \bmod 11) + (x \bmod 10) + 1$$

发现我们只需要考虑模 11 的值，也即 $u \equiv v(v - 1)/2 + 10 + (x \bmod 10) \pmod{11}$ 。我们可以构造一个仅包含 11 个状态的 DFA。之后使用 DP 计数即可，时间复杂度为 $O(11n)$ 。

5.1.1 DP 套 DP

DP 套 DP 是陈立杰于 WC 2015 提出的一类问题的解决方法 [9]。基本的问题形式是：问存在多少种特定的对象（例如串，序列，矩阵等）满足某个特定的条件。如果判断该对象是否符合条件可以通过 DP 判断，那么我们可以“通过一个外层的 DP 来计算使得另一个 DP 方程（子 DP）最终结果为特定值的输入数。由于我们只对 DP 方程的最终结果感兴趣，我们并不需要记录这前 i 位都是什么，只需要记录对这前 i 位进行转移以后，DP 方程关于每个状态的值就可以了。也即外层 DP 的状态是所有子 DP 的状态的值”。

DP 套 DP 的内层可以看做是记录“全体 DP 状态及值”的 DFA。对于某些此类问题，其内层的 DP 仅有有限种情况（或者可以通过一些方法合并为有限种情况），这时全体符合条件的串构成了正则语言，可以使用 DFA 描述。对于其他此类问题，其内层 DP 可能有无限种情况（也即等价类个数随串长的增长而增长），如果我们固定一个上界 N ，仅考虑能正确识别长度不超过 N 的串，同样可以使用一个 DFA 来描述。

5.2 DFA 最小化：减少 DP 状态数

将 DP 套 DP 或者其它一些问题引入 DFA 的好处在于，我们可以使用有限状态自动机相关的理论来帮助我们解决问题。下面举出一个例子：

例题 5.2 Equanimous²

²XIX Open Cup Grand Prix of China, Problem E, 另见 Codeforces Round #472 (Div. 1), Problem F

定义 $f(n)$ 表示将十进制数 n 所有数码之间填入加号或者减号，最终得到的值的绝对值最小值。

T 组询问，给定 l, r ，对于所有 $k = 0, 1, \dots, 9$ ，求所有 m 之和满足 $l \leq m \leq r$ 且 $f(m) = k$ 。答案对 $10^9 + 7$ 取模。

数据范围： $1 \leq T \leq 10^4, 1 \leq l \leq r \leq 10^{100}$ 。

考虑构造一个 DFA 能对于给定的 m 计算 $f(m)$ 。令 $dp(i, j)$ 表示 m 的前 i 位插入加减号后能否变成绝对值为 j 的数。转移就是 $dp(i-1, j) \rightarrow dp(i, j+d_i), dp(i, |j-d_i|)$ ，其中 d_i 就是 m 的第 i 位数。不难发现，DP 过程中状态可能会达到 $9 \cdot \log m$ 的级别，时间复杂度无法承受。实际上，我们能设定一个阈值 K ，使得我们不需要考虑 $j \geq K$ 的 DP 信息，同样能够得到正确的结果。为了得到更加一般化的结论，我们使用 $w (w \geq 2)$ 替代题中的 9。

首先，我们需要用到以下引理

引理 5.1 对于任意两个大小为 n 的多重集，只包含元素 $1, 2, \dots, n$ ，一定能各自找到一个子集，使得这两个子集之和相同。

证明. 假设两个集合分别为 $\{a_1, a_2, \dots, a_n\}$ 与 $\{b_1, b_2, \dots, b_n\}$ 。设 $A_i = \sum_{j=1}^i a_j$ ， $B_i = \sum_{j=1}^i b_j$ 。不失一般性，设 $A_n \leq B_n$ 。

令 p_i 为满足 $B_{p_i} \geq A_i$ 的最小值。令 $R_i = B_{p_i} - A_i$ ，显然有 $R_i \in [0, n-1]$ 。

若存在 $R_i = 0$ ，那么子集 $a[1 \dots i], b[1 \dots p_i]$ 即符合条件。

否则根据抽屉原理，必然存在 $1 \leq i < j \leq n$ 使得 $R_i = R_j$ 。此时子集 $a[(i+1) \dots j], b[(p_i+1) \dots p_j]$ 符合条件。 \square

由此可以得到下述推论

推论 5.1 对于任意两个多重集 A, B ，只包含元素 $1, 2, \dots, w$ 且元素之和 $\geq w(w-1)$ ，一定能各自找到一个子集，使得这两个子集之和相同。

证明. 不妨设 $|A| \leq |B|$ 。对于满足上述条件的多重集 A, B ，若元素个数不超过 $w-1$ ，一定只能是 $w-1$ 个 w 。

若 A, B 均包含 $\geq w$ 个元素，那么根据引理 5.1，结论成立。

若 A, B 均只包含 $w-1$ 个元素，显然结论同样成立。

若 A 包含 $w-1$ 个 w ， B 包含 $\geq w$ 个元素。任取 B 中的 w 个元素 x_1, x_2, \dots, x_w ，构造 $s_i = (\sum_{j=1}^i x_j) \bmod w$ 。若存在 $s_i = 0$ ，取 $x[1 \dots i]$ 即可。否则必然存在 $1 \leq i < j \leq w$ 使得 $s_i = s_j$ ，取 $x[(i+1) \dots j]$ 即可。结论同样成立。 \square

接下来的定理给出了本题可行的阈值

定理 5.1 只保留上述 DP 在 $0 \dots w^2 - 1$ 处的信息，同样能得到正确结果。

证明. 设 $m = \overline{d_1 d_2 \dots d_k}$ ($0 \leq d_i \leq w$)，则上述定理等价于对于任意正整数 m ，我们能找到序列 s_0, s_1, \dots, s_k 使得：

- $s_0 = 0$;
- $|s_i - s_{i-1}| = d_i (i = 1, 2, \dots, k)$;
- $|s_k| = f(m)$;
- $|s_i| \leq w^2 - 1 (i = 1, 2, \dots, k)$.

设存在最小的 p 使得 $|s_p| \geq w^2$, 不失一般性, 设 $s_p \geq w^2$.

显然我们有 $|s_k| = f(m) \leq w$, 即 $s_p - s_k \geq w(w-1)$. 设 $d_{p+1}, d_{p+2}, \dots, d_k$ 中取减号的所有元素构成集合 B , 那么 B 中所有元素之和一定 $\geq w(w-1)$.

考虑 d_1, d_2, \dots, d_p 中取加号的元素 (包括 d_1) 依次为 $d_{t_1}, d_{t_2}, \dots, d_{t_q}$. 取最大的 l 满足 $\sum_{j=1}^q d_{t_j} \geq w(w-1)$. 令集合 $A = \{d_{t_1}, d_{t_2}, \dots, d_{t_q}\}$, 集合 A 中的所有元素之和 $\in [w(w-1), w^2-1]$.

对于集合 A, B , 我们应用推论 5.1, 一定能找到 $U \in A, V \in B$ 使得它们的元素之和相同.

之后将所有 U 中元素对应的加号改成减号, 将所有 V 中元素对应的减号改成加号. 不难发现 $|s_k|$ 仍然保持不变, 同时对于 $i = 1, 2, \dots, p-1$ 仍然有 $|s_i| \leq w^2 - 1$. 此外 $|s_p|$ 一定会变小.

不断重复上述操作, 此后一定会终止, 最终的序列就是符合要求的了.

□

实际上, 本题的阈值通过更加细致的分析能达到 $(9-1) \times 9 + 1 = 73$, 到达上界的方式为构造连续 8 个 9 和 9 个 8. 我们将状态看做一个长度为 73 的 bitset, 并搜索出所有可达的状态, 这样的状态数量只有数万个. 我们将状态对应的 f 值分为 10 类, 并运行 Hopcroft 算法, 最终的最小 DFA 只有 715 个状态. 通过 DP 预处理足够的信息, 我们能 $O(10 \cdot (\log l + \log r))$ 回答一次询问.

5.3 利用等价关系构造 DFA

例题 5.3 Median Replace Hard³

给定一个长度为 8 的二进制串 $P = P_0P_1 \dots P_7$. 定义一个长度为 n 的二进制串 X 是好的, 当且仅当能够通过执行 $(n-1)/2$ 次下述操作变为串 “1”:

- 选择 X 的连续三个比特 (X_i, X_{i+1}, X_{i+2}) , 将它们替换为 P 的第 $(X_i + 2X_{i+1} + 4X_{i+2})$ 个比特.

共 T 组询问. 给定一个包含 0, 1, ? 的串 S , 问存在多少个将 ? 替换为 0, 1 的方案, 使得最后的串为好串. 答案对 $10^9 + 7$ 取模.

数据范围: $1 \leq T \leq 256, 1 \leq |S|, \sum |S| \leq 300000, |S|$ 为奇数

³XX Open Cup Grand Prix of Tokyo, Problem J

实际上对于所有 256 个可能的 P ，我们都能构造一个 DFA 识别全体好串。找到这样的 DFA 之后，我们就能够通过一个简单的 DP 解决本题。

回忆 Myhill-Nerode 定理中的等价关系，我们认为串 x, y 等价当且仅当对于任意的串 z ，均有 xz 是好串当且仅当 yz 是好串。我们能根据等价类直接构造出 DFA。但是枚举所有的串 z 来判断是否等价是不可能的，考虑仅枚举长度不超过 L 的串 z 来判断等价性（同时通过记忆化 DFS 判断一个串是否为好串）。

不妨将所有等价类中任意长度最小的串视作代表元，我们可以使用 BFS 找出所有的代表元。先将空串入队，每次从队首弹出一个串 x ，并判断串 x 和已有的代表元是否等价。如果均不等价，那么它一定能成为某个等价类的代表元。假设存在某个等价类的代表元没有被找到，同时该串去掉最后一个字符得到串 t ，那么串 t 所在等价类的代表元加入这个字符就得到与之等价的串。观察这个过程，我们也能发现所有代表元形成了一个树形结构。

接下来的问题是我们怎样验证这个 DFA 是否符合条件？假设存在一个 DFA $d(k)$ 能正确识别长度不超过 k 的好串。据此可以构造出一个 NFA 能正确识别长度不超过 $k+2$ 的好串，再将其转化为 DFA $d(k+2)$ ，并最小化。如果 $d(k)$ 等价于 $d(k+2)$ ，我们就能得到 $d(k) = d(k+2) = d(k+4) = \dots$ ，这也就是我们所要求的 DFA。通过计算机验证，取 $L = 10$ 就能满足条件。最坏情况下 DFA 的大小为 35，可以通过本题。

5.4 OI 字符串理论中的 DFA

由于 DFA 自身的特性，它成为了信息学竞赛中的许多字符串算法的基础结构。Trie 是一个能识别多串的简单结构，AC 自动机在此基础上引入失配边，使之能不断回退最终找到匹配。序列自动机是贪心的产物，结构简单却能有效处理子序列相关的问题。回文树是解决回文串问题有力算法，进一步的讨论可以参见 [11]。

在 OI 字符串理论中，成果最为丰富，应用最为广泛的当属后缀数据结构。后缀 Trie 是最基本的结构，在此基础上进行压缩（缩去出入度均为 1 的点）就得到了后缀树，而后缀自动机就是最小化后的后缀 Trie。同时对后缀 Trie 进行最小化和压缩能够得到压缩后缀自动机等更进一步的结果。从 DFA 的角度思考，有助于我们更加深刻地理解这些结构。上述内容的进一步讨论可以参见 [8, 10, 12]。接下来的这道例题假定读者已经掌握了后缀自动机的基础知识。

例题 5.4 Beautiful Automata⁴

构造一个仅包含小写字母的串 s ，使得 s 的后缀自动机的转移图（仅保留其有向图的结构）与给定的 DAG 同构。如果不存在，输出 -1 ，否则输出字典序最小的解。

数据范围： $1 \leq n \leq 2000, 1 \leq m \leq 3000$ 。

显然只有开始状态入度为 0，记为 S 。根据后缀自动机的性质，出度为 0 的点只能有一个，记为 T 。也即图中的任意一个点都能从 S 到达，也都能到达 T 。因为后缀自动机中的一

⁴XIX Open Cup Grand Prix of Baltic Sea, Problem G

个节点 u 是 right 集合相同的串的集合，所以所有从 S 到 u 的路径长度互不相同且构成了一个区间。

假设所有从 S 到 T 的路径长度分别为 $K+1, K+2, \dots, n$ ，分别代表从 $1, 2, \dots, n-K$ 开始的后缀。因为 S 的转移边对应字符互不相同，所以我们根据这 $n-K$ 条路径的经过的第一条边，就能贪心确定 s 的前 $n-K$ 个字符。

剩下的 K 个字符在枚举 T 的后缀连接 P （即 parent 树上的父亲）后能唯一确定。我们要求从 S 到 P 的最长路恰好为 K ，对应 s 长度为 K 的后缀。假设存在一条 P 到 T 的路径长度为 l ，那么就有 $S[n-K+1:n] = S[n-K+1-l:n-l]$ 。由此便能唯一确定整个串 s 。

如何判断一个串 s 是否符合条件？考虑恢复 DAG 的转移边字符。观察到对于一个后缀自动机，我们从 T 出发的任意一条反向路径构成了 s 反串的前缀。所以我们可以从 T 开始 BFS，就能唯一确定 DAG 的转移边了。接着再求出 s 的真实后缀自动机，判断两者是否同构即可。

时间复杂度为 $O(nm|\Sigma|)$ 。

5.5 正则表达式匹配的算法

关于正则表达式一个常见的问题是，判断一个串是否属于它的正则语言（设正则语言规模为 s ，串长为 n ）。我们不难想到使用 Thompson 构造法将正则表达式转化为一个 NFA，变成 NFA 的计算问题。接下来有两个方向可以继续。

一个方向是将 NFA 通过幂集构造法转化为 DFA，计算时可以 $O(1)$ 完成一次转移。这在正则表达式规模较小的时候比较适用。但是，我们构造正则表达式 $(a+b)^*a(a+b)(a+b)(a+b)\dots$ 就能将对应状态数最少的 DFA 卡到指数级。所以这个做法的时间复杂度为 $O(s \cdot 2^s + n)$ 。

另一个方向是直接 NFA 上计算。我们套用前文的 NFA 计算算法，即可做到 $O(\frac{n \cdot s^2}{\omega \cdot \log n})$ 的时间复杂度。实际上，利用正则表达式转化后的 NFA 的特殊性，我们能做到更优的时间复杂度。

注意到 Thompson 构造法每次只会增加常数条转移边。如果我们不将 ϵ 转移边消去，总转移边数就是 $O(s)$ 的。我们可以借鉴幂集构造方法的思想，在算法的开始我们求出 $E(q_0)$ （即从初始状态沿 ϵ 转移到达的状态）作为初始的状态集合。在计算过程中，我们求出当前状态集合 S 中的元素，沿 c 转移边能够到达的新的状态集合 S' 。之后再求出沿 ϵ 转移边能到达的所有状态（这一部分可以通过队列实现 BFS 完成）。发现每一步转移只需要遍历所有 $O(s)$ 个状态与转移边，所以该算法的时间复杂度为 $O(ns)$ 。

5.5.1 基于 Method of Four Russians 的高效算法

事实上，我们做到更优的时间复杂度。由于下述算法较为复杂，本文仅做一个简单的介绍，感兴趣的读者可以参考 [4] 获取详细内容。

首先建出正则表达式的表达式树，一个性质是每个节点至多只有两个子节点。考虑将表达式树分块，设阈值为 K ，这里介绍一种分块的方法：每次从根节点开始，不断移向较大的一棵子树，直到子树的大小 $\leq K$ ，将该子树看做一个块，并使用一个节点来替代该子树。因为每个节点至多只有两个子节点，所以子树大小至多除 2，除根节点外，剩余的块大小一定在 $K/2$ 到 K 之间。表达式树被分作 $O(s/K)$ 块。我们建出正则表达式对应的 NFA，强制 Thompson 构造法中的每个运算符都新建两个节点（即 $R = (R_1R_2)$ 时，也新建一对节点），不难发现构造出 NFA 形成了嵌套结构，每块 NFA 通过一对节点与上一层的 NFA 产生联系。假设 $K \leq \omega$ ，我们可以使用一个二进制数维护每块 NFA 中的状态。

之后的转移分为两部分，首先考虑字符 c 的转移。根据 Thompson 构造法得到的 NFA 的性质，我们只需要考虑正则表达式树中的叶子节点。对于所有 $O(s/K)$ 个块和转移字符，我们预处理 $O(2^K)$ 种情况转以后得到的结果，就能在 $O(s/K)$ 内完成转移。

接下来考虑 ϵ 的转移。假设不存在闭包运算（即 $*$ 运算），不难发现我们的 NFA 构成了一个 DAG，使得我们能够按照拓扑序来处理转移。这种情况下，我们可以按照正则表达式树 DFS 序的顺序来处理，同样预处理所有 $O(2^K)$ 种情况，我们能 $O(1)$ 完成一次块间的转移。存在闭包运算的情况更为复杂，因为存在回退的 ϵ 边，不过我们有以下结论

引理 5.2 由 Thompson 构造法得到的 NFA 中，任意仅由 ϵ 组成的无环路径，至多经过一次闭包运算中的回退边。

相信读者不难独立给出这个结论的证明（只需对正则表达式进行归纳即可）。由于只会回退一次，我们可以同样通过预处理来加速回退边的转移。接着，只需要再次按正则表达式树的 DFS 序处理一遍即可。本算法可以看做是 Method of Four Russians 又一强大的运用，正则表达式匹配的时间复杂度被优化为了 $O(\frac{ns}{\log n})$ 。

例题 5.5 数字搜索⁵

给定一个字符集大小为 10 的正则表达式（长度为 s ），以及一个长度为 n 的字符串。

问该字符串存在多少前缀使得：该前缀存在一个后缀与正则表达式匹配？

数据范围： $1 \leq s \leq 500, 1 \leq n \leq 10^7$ 。

注意到本题与正则表达式匹配非常相似。同样考虑使用 NFA 来处理，我们只需要每一步转移之后，将开始状态加入状态集合。每次判断该前缀是否存在后缀与其匹配，判断状态集合是否包含接受状态即可。套用普通的正则表达式匹配算法即可做到 $O(ns)$ ，使用基于 Method of Four Russians 的高效算法即可做到 $O(\frac{ns}{\log n})$ 的时间复杂度。

本题在网络中现存的解题报告⁶与通过程序时间复杂度都是错误的。这些做法都是将当前的状态集合与转移对应的新的状态集合记录下来，再次访问到的时候就能 $O(\frac{s}{\omega})$ 完成一次转移。由于本题的测试数据强度较低，使得 NFA 在计算过程中有大量状态集合会重复出现。实际上，我们只需要构造正则表达式为 $(0+1)^*1(0+1)(0+1)\cdots(0+1)$ （总长度不超过 500），

⁵CTSC 2004

⁶参见 <https://www.docin.com/p-2262533912.html>，或 <https://www.docin.com/p-2268504832.html>

并使用强度较高的随机数生成器产生一个长度为 10^7 的随机 01 序列，就能使得所有通过程序无法在时限的十倍时间内计算出答案。

如果有读者提出了更加优秀的算法，欢迎与作者交流。

6 总结

本文系统地简述了有限自动机的基础理论及算法，并展现了相关理论在信息学竞赛中的丰富应用。

有限状态自动机是一个非常简单的结构，但是我们能发掘大量的性质与许多有趣的应用，这就是它的魅力所在。作者相信，有限状态自动机仍然有着巨大的潜力，等待我们去进一步探索更加有趣的理论与应用。希望本文能起到抛砖引玉的作用，吸引更多的选手来学习和研究有限状态自动机。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢徐先友老师的关心和指导。

感谢周航锐同学、唐靖哲前辈对我的启发和写作本文的帮助。

感谢陈立言同学、周航锐同学为本文验稿。

感谢父母对我多年来的关心与支持。

参考文献

- [1] Hopcroft J E, Motwani R, Ullman J D. Introduction to Automata Theory, Languages, and Computation, 3rd Edition[M]. Addison Wesley, 2006.
- [2] Sipser M. Introduction to the Theory of Computation[M]. Cengage learning, 2012.
- [3] Hopcroft J. An $N \log N$ Algorithm for Minimizing States in a Finite Automaton[M]. Academic Press, 1971.
- [4] Myers G. A four russians algorithm for regular expression pattern matching[J]. Journal of the ACM (JACM), 1992, 39(2): 432-448.
- [5] 乔明达. 有限状态自动机 [R]. NOI WC 讲课, 2014.
- [6] 杜瑜皓. FSM 相关问题选讲 [R]. ZJOI 讲课, 2015.

- [7] 茹逸中. 形式语言与自动机 [R]. NOI WC 讲课, 2016.
- [8] 张云帆. 从 DFA 到后缀自动机 [R]. APIO 讲课, 2018.
- [9] 陈立杰. 计数问题选讲 [R] NOI WC 讲课, 2015.
- [10] 徐翊轩. 浅谈压缩后缀自动机 [J]. IOI 中国国家候选队论文集, 2020.
- [11] 翁文涛. 回文树及其应用 [J]. IOI 中国国家候选队论文集, 2017.
- [12] Wikipedia. Suffix automaton[OL].
https://en.wikipedia.org/wiki/Suffix_automaton.
- [13] Wikipedia. Myhill-Nerode theorem[OL].
https://en.wikipedia.org/wiki/Myhill-Nerode_theorem.
- [14] Wikipedia. DFA minimization[OL].
https://en.wikipedia.org/wiki/DFA_minimization.

再谈图连通性相关算法

宁波市镇海中学 虞皓翔

摘要

本文介绍了图论中经典的连通性问题，针对 k -点/边连通和 2-点/边连通给出了对应的算法和应用，并引入了 3-点/边连通问题，并对其进行了初步介绍。

引言

图论是组合数学中一个历史悠久的分支，以图为研究对象。图的连通性是图论中基础且重要的理论，更是图论中的奠基石。

OI 中的图论主要以数据结构为主，对连通性等的考察并不多。笔者希望通过本文，结合图论、dfs 树、网络流等多种算法，使用了传统和较现代的图论理论，来对这些基本的问题做一个介绍。

1 相关定义

1.1 图

定义 1.1.1 (图). 图是一个二元组 $G = (V, E)$ ，其中 V, E 为集合或多重集，分别表示图的点集和边集， V 中的元素称为顶点， E 中的元素称为边。每条边是一个二元组 (u, v) ，其中 $u, v \in V$ 。若所有边均为无序二元组，则称 G 是无向图；若所有边均为有序二元组，则称 G 是有向图。

一般地，我们用 $V(G)$ 表示 G 的点集， $E(G)$ 表示 G 的边集。

定义 1.1.2 (自环, 重边, 简单图). 对无向图 $G = (V, E)$ ，若 $e = (u, v) \in E$ 满足 $u = v$ ，则称 E 是自环；若 E 中存在相同元素 (u, v) (注意 (u, v) 和 (v, u) 视为相同)，则称它们是一组重边；若 G 中不存在自环和重边，则称 G 是简单图。

定义 1.1.3 (关联, 相邻). 在无向图 G 中，若点 v 是 e 的一个端点，则称 v, e 是关联的，若两个顶点 u, v 是同一条边 e 的两个端点，则称 u, v 是相邻的。

定义 1.1.4 (邻域, 度). 对于一个顶点 $v \in V$, 所有与之相邻的顶点的集合 (多重集) 称为 v 的邻域, 用 $N(v)$ 表示; 与 v 关联的边的条数称作该顶点的度, 用 $d(v)$ 表示。对于无向图 G , 记 $\delta(G) = \min_{v \in G} d(v), \Delta(G) = \max_{v \in G} d(v)$ 分别表示 G 的最小度和最大度。

定义 1.1.5 (子图). 对无向图 $G = (V, E)$, 若存在另一张无向图 $H = (V', E')$ 满足 $V' \subseteq V$ 且 $E' \subseteq E$, 则称 H 是 G 的子图, 记作 $H \subseteq G$ 。

定义 1.1.6 (导出子图). 若 $H = (V', E')$ 是 $G = (V, E)$ 的子图, 且满足对 $\forall u, v \in V'$, 只要 $(u, v) \in E$ 就有 $(u, v) \in E'$, 则称 H 是 G 的导出子图, 也称诱导子图。

容易发现, 一个图的导出子图仅仅由它的点决定, 因此点集为 V' 的导出子图称为 (G 中) V' 导出的子图, 记作 $G[V']$ 。

1.2 连通性

定义 1.2.1 (途径、迹和路径).

- 对于无向图 G 中一个点和边的交错序列 $w = [v_0, e_1, v_1, e_2, v_2, \dots, e_k, v_k]$, 其中首尾是点, 且 $e_i = (v_{i-1}, v_i)$, 则称 w 是一条途径 (walk), 简记作 $w = v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_k$ 或 $w = v_0 \rightsquigarrow v_k$ 。
- 对于一条途径 w , 若 e_1, e_2, \dots, e_k 两两不同, 则称 w 是一条迹 (trail)。
- 对于一条迹 w , 若 $v_i = v_j (i \neq j)$ 蕴含 $\{i, j\} = \{0, k\}$, 则称 w 是一条路径 (path)。也就是说, 除了首末端点可以相同外, 其余所有的点都不能相同。

定义 1.2.2 (回路和圈). 闭合¹的迹称为回路 (circuit), 闭合的路径称为圈 (cycle), 也称环。

定义 1.2.3 (连通性). 对于一张无向图 $G = (V, E)$ 和 $u, v \in V$, 若存在途径 $u \rightsquigarrow v$, 则称 u, v 是连通的。若 G 中任意两个顶点之间都是连通的, 则称 G 是连通图, 或 G 的这一性质称作连通性。

定理 1.2.1. 对无向图 $G = (V, E)$ “连通”是 V 上的等价关系²。 □

根据定理 1.2.1, 我们可以根据顶点之间是否“连通”的关系将 V 划分³成若干个等价类 V_1, V_2, \dots, V_n 。其中每个等价类被称作 G 的一个连通分量, 也称连通块。

¹即 $v_0 = v_k$ 。

²即具有自反性、对称性和传递性的二元关系。

³“划分”即两两交集为空, 所有集合并集为 V 。

1.3 割集和连通度

定义 1.3.1 (点割集). 对于无向图 $G = (V, E)$, 其中 $u, v \in V$ 满足 $u \neq v$ 且 $(u, v) \notin E$. 若存在点集 $S \subseteq V \setminus \{u, v\}$ 使得 $G[V \setminus S]$ 中 u, v 不连通, 则称 S 是 u, v 的 (局部) 点割集. 若 S 是某对 (u, v) 的局部点割集, 则称 S 是 G 的 (全局) 点割集.

定义 1.3.2 (边割集). 对于无向图 $G = (V, E)$ 和 $u, v \in V (u \neq v)$, 如果存在边集 $F \subseteq E$ 使得在 $G \setminus F^4$ 中 u, v 不连通, 则称 F 是 u, v 的 (局部) 边割集. 若 F 是某对 (u, v) 的局部边割集, 则称 F 是 G 的 (全局) 边割集.

定义 1.3.3 (点连通度). 对于无向图 G 中一对不相邻的顶点 u, v , 定义 u, v 的 (局部) 点连通度为 u, v 的局部点割集大小的最小值, 记为 $\kappa(u, v)$. 若 G 是非完全图⁵, 则定义 G 的 (全局) 点连通度为所有全局点割集大小的最小值, 记为 $\kappa(G)$.

定义 1.3.4 (边连通度). 对于无向图 $G = (V, E)$ 和 $u, v \in V (u \neq v)$, 定义 u, v 的 (局部) 边连通度为 u, v 的局部边割集大小的最小值, 记为 $\lambda(u, v)$. 若 $|G| \geq 2$, 则定义 G 的 (全局) 边连通度为所有全局边割集大小的最小值, 记为 $\lambda(G)$. 若 $|G| = 1$, 则定义 $\lambda(G) = +\infty$.

定义 1.3.5 (k -连通性). 若 $\kappa(G) \geq k$ (或 $\lambda(G) \geq k$), 则称 G 是 k -点 (边) 连通图, G 的这一性质称作 k -点 (边) 连通性.

在大多数文献中, 若没有特别注明是点连通还是边连通, 则一般认为其是指点连通.

可以发现, 点连通性对图是否是简单图不敏感, 即去掉自环, 将重边缩为一条不影响两点间的点连通性.

相反, 边连通性对图是否是简单图是敏感的, 即统计边连通度时需要计算重边的重数.

2 k -连通性及其性质⁶

2.1 Menger 定理

关于点连通度和边连通度, 最重要的结论或许是下述定理:

定理 2.1.1 (Menger).

- 对于无向图 G 中一对不相邻的顶点 u, v , $\kappa(u, v) \geq k$ 当且仅当存在 k 条从 u 到 v 的路径, 两两之间的公共点只有 $\{u, v\}$.

⁴即图 $(V, E \setminus F)$ 的简记.

⁵对于完全图 K_n , 由于不存在全局点割集, 因此上述定义无效, 此时常见的定义由两种: $\kappa(K_n) = n - 1$ 或 $\kappa(K_n) = n$, 需根据上下文和相关文献合理选择.

⁶本节内容及复杂度分析中默认点和边不带权, 在边连通的分析中重边可以模拟边权的相关性质.

- 对于无向图 G 中的一对顶点 u, v , $\lambda(u, v) \geq k$ 当且仅当存在 k 条从 u 到 v 的路径, 两两之间没有公共边。 \square

事实上, Menger 定理是最大流最小割定理 [1] 的不带权版本, 其中点连通和边连通分别对应于限制点容量和边容量。

2.2 连通度的界

关于, 图的连通度有非常多的结论, 下面给出一个最基本的结论:

定理 2.2.1. 对于无向图 G 中一对不相邻的顶点 u, v , 有 $\kappa(u, v) \leq \lambda(u, v) \leq \min\{d(u), d(v)\}$ 。从而, 对于任意非完全图的图 G , 则 $\kappa(G) \leq \lambda(G) \leq \delta(G)$ 。

证明. 设 F 是 u, v 的局部边割集。

则对于 F 中的每一条边, 至少存在一个端点不是 u 或 v 。我们对所有 $|F|$ 条边都取这样一个点, 得到一个点集 S , 显然 $|S| \leq |F|$ 且 S 是 u, v 的局部点割集, 故 $\kappa(u, v) \leq \lambda(u, v)$ 。不妨设 $d(u) \leq d(v)$, 则与 u 关联的所有边构成 G 的一个边割集, 故 $\lambda(u, v) \leq \min\{d(u), d(v)\}$ 。

由全局连通度和最小度的定义知 $\kappa(G) \leq \lambda(G) \leq \delta(G)$ 。 \square

结合握手定理, 有 $|V|\delta(G) \leq \sum_{v \in V} d(v) = 2|E|$, 得

推论 2.2.1. 对于非完全图 $G = (V, E)$, 有 $\kappa(G) \leq \lambda(G) \leq \frac{2|E|}{|V|}$ 。 \square

在图中添加或删除一条点和边, 对图的连通度的影响如下:

定理 2.2.2.

- 点连通 (以下默认点连通有意义, 即图不是完全图)。

设 $\kappa(G) = A$, 则:

对于任意 $v \in G$, $\kappa(G \setminus \{v\}) \geq A - 1$ 。⁷

对于任意 $e \in G$, $\kappa(G \setminus \{e\}) \leq A$ 。

- 边连通。

设 $\lambda(G) = B < +\infty$, 则:

对于任意 $e \in G$, $B - 1 \leq \lambda(G \setminus \{e\}) \leq B$ 。 \square

这些结论可以通过定义不难得到。

⁷对于点集 $S \subseteq V(G)$, 用 $G \setminus S$ 表示导出子图 $G[V(G) \setminus S]$, 即删去图中若干个节点。

2.3 局部 k -边连通性

2.3.1 k -边连通关系

下面我们来讨论一下局部 k -边连通性。

对于无向图 G 和正整数 k ，我们在 V 上定义二元关系 ρ_k ，其中 $(a, b) \in \rho_k$ 当且仅当 $\lambda(a, b) \geq k$ ⁸，称该关系为图的 k -边连通关系。则：

定理 2.3.1. ρ_k 是 V 上的等价关系。

证明. 显然 ρ_k 具有自反性和对称性。只需证明 ρ_k 具有传递性。

设 $(a, b), (b, c) \in \rho_k$ ，即 $\lambda(a, b) \geq k, \lambda(b, c) \geq k$ 。

设 F 是 a, c 的局部边割集，则 $G \setminus F$ 中 a, c 不连通，于是由连通的传递性知， b 至少和其中之一不连通。不妨设 a 和 b 不连通。

于是 F 也是 a, b 的局部边割集，由定义知 $\lambda(a, c) \geq \lambda(a, b) \geq k$ ，即 $(a, c) \in \rho_k$ 。 \square

2.3.2 k -边连通分量

由定理 2.3.1， ρ_k 将 V 划分为若干个等价类 V_1, V_2, \dots, V_n 。称其中每个等价类为 G 的一个 k -边连通分量。

当 $k \geq 3$ 时，一个 k -边连通分量的导出子图并不一定是连通的，如图 1 中 $\{1, 2\}$ 是一个 3-边连通分量。

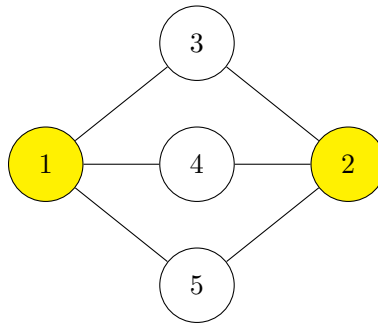


图 1

2.3.3 k -边连通判定

由定理 2.1.1，判断两个点 u, v 是否 k -边连通，只需要判断是否存在从 u 到 v 大小 $\geq k$ 的流即可。

因此，如果只需要判定一对顶点，那么使用最大流算法是一个不错的选择。

⁸特别地，当 $a = b$ 时人为规定 $\lambda(a, b) = +\infty$ 。

由于最大流算法具有很多不同的实现 [2], 需要合理衡量算法的时间复杂度和实现难度, 因此以下用 $O(\text{Flow}(v, e))$ 表示在一张 v 个点 e 条边的 (单位容量) 图做最大流的时间复杂度。

2.3.4 k -边连通分量划分

接下来考虑如何求出 G 的 k -边连通分量划分。

一个最简单的实现是对每一个点对之间做最大流, 时间复杂度 $O(|V|^2 \text{Flow}(|V|, |E|))$ 。

但实际上并不需要做那么多次最大流。

考虑取定一对顶点 u, v , 我们可以在 $O(\text{Flow}(|V|, |E|))$ 的时间内判断是否有 $\lambda(u, v) \geq k$ 。

- 若 $\lambda(u, v) \geq k$, 则 u 和 v 应在同一个 k -边连通分量中, 因此以下过程中无需再考虑 v 点。
- 若 $\lambda(u, v) < k$, 则说明存在一个局部边割集 F , 满足 u, v 在 $G \setminus F$ 中不连通。

于是, 令 C_u 为 u 所在的连通分量, $C_v = V \setminus C_u$ 。此时, 对于 $\forall u' \in C_u, v' \in C_v$, F 也是 u' 和 v' 的局部边割集, 即 $(u', v') \notin \rho_k$ 。

这说明, C_u 是若干个 k -边连通分量的并, C_v 也是若干个 k -边连通分量的并。也就是说, 我们将一个问题转化为了其子问题。

综上, 每次操作要么“删去”某个集合中的一个顶点, 要么将一个已知的集合拆分成若干个集合。当每个集合中的点都删得只剩下 1 个时, 我们就得到了 G 的 k -边连通分量划分。

于是我们只需要完成 $O(|V|)$ 次最大流, 时间复杂度 $O(|V| \text{Flow}(|V|, |E|))$ 。

2.4 边连通度

2.4.1 局部边连通度

判定是否为 k -边连通的一个自然推广就是计算取任意点对之间的局部边连通度。

在介绍局部边连通度的算法前, 再介绍最小边割集 (最小割) 的一个性质:

定理 2.4.1. 设 $G = (V, E)$, 有 $u, v \in V$ 。设 F 是 u, v 的最小割, 设 u, v 所在的连通分量为 S, T 。则对于 $\forall u' \in S, v' \in T$, 均有 $\lambda(u', v') \leq \lambda(u, v)$ 。

证明. 设 G 为 u, u' 的最小割, 其中 u, u' 所在的连通分量为 P, Q , 如图 2。

由最小割的 Submodular 性质知可以不妨设 $T \subseteq P$ 或 $T \subseteq Q$ 。

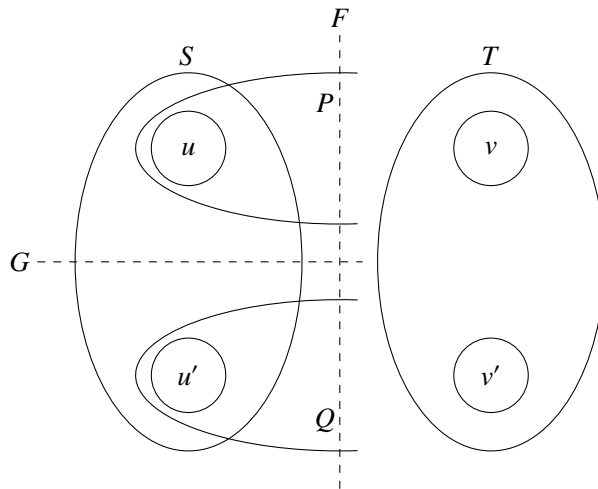


图 2

- 若 $T \subseteq P$, 则 G 也是 u', v' 的边割集, 因此 $\lambda(u', v') \leq |G| = \lambda(u, u')$ 。
- 若 $T \subseteq Q$, 则 G 也是 u, v 的边割集, 因此 $\lambda(u, v) \leq |G| = \lambda(u, u')$, 而 F 也是 u', v' 的边割集, 因此 $\lambda(u', v') \leq |F| = \lambda(u, v)$, 即 $\lambda(u', v') \leq \lambda(u, u')$ 。

综上, 有 $\lambda(u', v') \leq \lambda(u, u')$ 。 □

推论 2.4.1. 条件同定理 2.4.1, 有 $\lambda(u', v') \leq \min\{\lambda(u', u), \lambda(u, v), \lambda(v, v')\}$ 。 □

又由定理 2.3.1 知, $\lambda(a, c) \geq \min\{\lambda(a, b), \lambda(b, c)\}$, 结合推论 2.4.1 知

定理 2.4.2. 条件同定理 2.4.1, 有 $\lambda(u', v') = \min\{\lambda(u', u), \lambda(u, v), \lambda(v, v')\}$ 。 □

根据定理 2.4.2, 不难得到一个计算任意点对的局部边连通度的算法:

1. 定义函数 $EFT(U)$, 其中 $U \subseteq V$, 返回一棵带边权的树。
EFT(U) 的具体过程如下:
2. 若 $|U| = 1$, 返回树 (U, \emptyset) 。
3. 否则, 任取 U 中不同两点 u, v , 使用一次最大流算法得到 $\lambda(u, v)$ 以及对应的局部边割集, 设对应的两个连通分量为 S^*, T^* , 其中 $u \in S^*, v \in T^*$ 。
4. 令 $S = S^* \cap U, T = T^* \cap U$ 。
5. 令 $(S, E_S) = EFT(S), (T, E_T) = EFT(T)$ 。
6. 最后令边 $e = (u, v)$, 边权为 $\lambda(u, v)$, 函数返回树 $(U, E_S \cup E_T \cup e)$ 。

要注意一点的是,运行最大流算法时一定要在原图 G 上运行,不能取导出子图(见 2.3.2)。

上述过程中得到的树被称为**等价流树**(Equivalent-flow tree),下面证明等价流树的主要性质:

定理 2.4.3. 对于 $\forall u, v \in V (u \neq v)$, $\lambda(u, v)$ 等于 EFT(T) 中路径 $u \rightsquigarrow v$ 中边权最小的边的权值。

证明. 固定 G , (依照算法过程) 对 U 归纳证明。

设 EFT(S), EFT(T) 满足定理结论, 则对于 EFT(U) 中的点对, 只需考虑 $u' \in S, v' \in T$ 中的情形。

由**定理 2.4.2**, 知 $\lambda(u', v') = \min\{\lambda(u', u), \lambda(u, v), \lambda(v, v')\}$ 。

由归纳假设和树的性质知, $\lambda(u', v')$ 恰好等于路径 $u' \rightsquigarrow u \rightarrow v \rightsquigarrow v'$ 中边权最小的边的权值, 命题成立。 \square

不难证明这个算法的时间复杂度也是 $O(|V| \text{Flow}(|V|, |E|))$, 且预处理完毕后, 我们将一次局部边连通度的询问转化为树上权值 \min , 以达到 $O(\log |V|)$ 甚至 $O(1)$ 询问。

2.4.2 Gusfield 算法

Gusfield 算法是上述算法的一种实现, 由于它不需要递归, 因此显得较有优势。

该算法的流程如下:

1. 任取 $r \in V$, 令 $R = V \setminus \{r\}, E_T = \emptyset, B_r = R, B_v = \emptyset (v \neq r)$ 。
2. 若 $R = \emptyset$, 则返回树 (V, E_T) 。
3. 取 $v \in R$, 令 $R = R \setminus \{v\}$ 。
4. 设 u 满足 $v \in B_u$, 使用最大流算法得到 $\lambda(u, v)$ 以及对应的局部边割集, 设对应的两个连通分量为 S^*, T^* , 其中 $u \in S^*, v \in T^*$ 。
5. 令 $B_v = B_u \cap T^*, B_u = B_u \cap S^*$ 。
6. 最后令边 $e = (u, v)$, 边权为 $\lambda(u, v)$, 令 $E_T = E_T \cup \{e\}$, 回到步骤 2。

读者不难证明该算法和之前给出的算法是等价的。Gusfield 算法的时间复杂度仍为 $O(|V| \text{Flow}(|V|, |E|))$ 。

2.4.3 全局边连通度

对于一张图，它的全局边连通度等于所有局部边连通度的最小值，因此它可以在 $O(|V| \text{Flow}(|V|, |E|))$ 的时间内得到。

然而，如果我们只需要知道它的全局边连通度的话，有专门的算法来解决此类问题，其中比较著名的有确定性的 Stoer-Wagner 算法 [4] (时间复杂度 $O(|V|^2 \log |V| + |V| |E|)$) 和基于随机化的 Karger 算法 [3] (时间复杂度 $O(|V|^2 \log^3 |V|)$)。限于篇幅，这里不对这两种算法进行展开，感兴趣的读者可以见相关文献。

2.5 点连通度

与边连通度相对应的一个问题就是点连通度。

由图 3 可知， k -点连通性没有传递性。 $(\kappa(1, 2) = \kappa(2, 3) = 2, \kappa(1, 3) = 1)$ 。

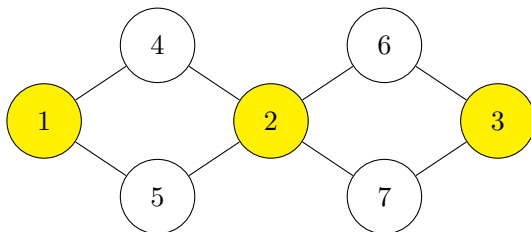


图 3

从而处理点连通度会比处理边连通度略显麻烦。

2.5.1 局部点连通度

计算两个点的点连通是容易的，由定理 2.1.1 知我们只需计算在限制点容量为 1 的条件下从 u 到 v 的最大流的大小。

对于限制了点容量，我们可以用拆点的思想，将一个点 v 拆成两个点 v, v' ，中间连一条 $v \rightarrow v'$ 的容量为 1 的边。对于原来的无向边 (u, v) ，转化成有向边 $u' \rightarrow v$ 和 $v' \rightarrow u$ ，最后使用一般的最大流算法即可。

如，图 3 拆点后就变成了图 4。

于是 $\kappa(u, v)$ 就等于从 u' 到 v 的最大流 (也等于从 v' 到 u 的最大流)。

该算法的时间复杂度为 $O(\text{Flow}(|V|, |E|))$ 。

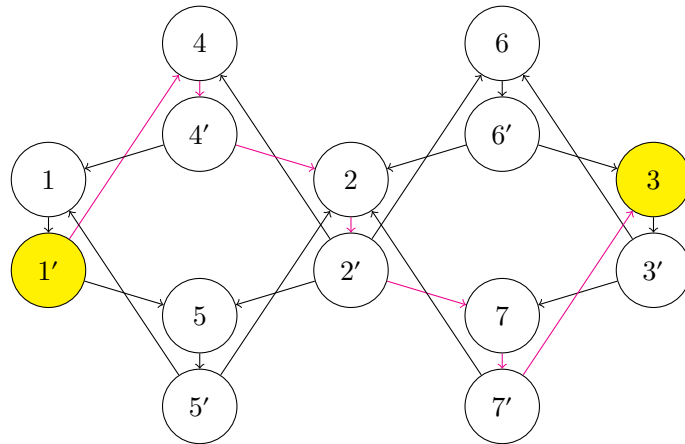


图 4

2.5.2 全局点连通度

由定义，

$$\kappa(G) = \min_{(u,v) \notin E} \kappa(u,v)$$

因此按照该定义直接计算，时间复杂度 $O(|V|^2 \text{Flow}(|V|, |E|))$ 。

和边连通度类似，我们并不需要运行 $O(|V|^2)$ 次最大流，因为有很多操作都是冗余的。

设 $G = (V, E)$ 是非完全图的连通无向简单图， $\kappa(G) = c$ ，则 $1 \leq c \leq |V| - 2$ 。

于是， G 存在大小为 c 的全局点割集，设为 S 。

设 $V = \{1, 2, \dots, |V|\}$ 。由于 $|S| = c$ ，因此必定存在元素 $v \in \{1, 2, \dots, c+1\}$ 满足 $v \notin S$ 。

又图 $G \setminus S$ 中有至少两个连通分量，从而存在 $u \in V \setminus S$ 且 u, v 在两个不同连通分量。显然 $(u, v) \notin E$ 。

于是 S 是 u, v 的一个局部点割集，从而 $\kappa(u, v) = c$ 。这说明：

定理 2.5.1. 若非完全图的连通无向简单图 $G = (V, E)$ 的点连通度为 c (即 $\kappa(G) = c$)，则对于 V 的任意一个 $c+1$ 元子集 X ，一定存在 $v \in X, u \in V, (u, v) \notin E$ 满足 $\kappa(u, v) = c$ 。 \square

于是我们得到了一个时间复杂度为 $O(\kappa(G) |V| \text{Flow}(|V|, |E|))$ 的算法：

1. 不妨设 $V = \{1, 2, \dots, |V|\}$ 。
2. 令 $u = 1, c = +\infty$ 。
3. 若 $u > c$ ，则返回 $\kappa(G) = c$ ，程序结束。
4. 否则，令 $U = \{v \mid u < v \wedge (u, v) \notin E\}$ ，使用网络流计算

$$c_u = \min_{v \in U} \kappa(u, v)$$

5. 令 $c = \min\{c, c_u\}$, $u = u + 1$, 回到步骤 2。

由推论 2.2.1, 它的时间复杂度也等于 $O(|E|\text{Flow}(|V|, |E|))$ 。

2.5.3 k -点连通分量

鉴于我们未对满足 $(u, v) \in E$ 的 u, v 定义 $\kappa(u, v)$ 。因此在本小节中, 我们额外给出它的定义:

定义 2.5.1. 对于无向简单图 $G = (V, E)$ 和 $(u, v) \in E$, 定义 $\kappa(u, v)$ 为从 u 到 v 的路径条数的最大值 (含路径 $u \rightarrow v$), 满足两两之间的公共点只有 $\{u, v\}$ (也就是说, 通过 Menger 定理来定义)。

为了区分起见, 我们将这个函数记为 κ^* , 即

$$\kappa^*(u, v) = \begin{cases} \kappa(u, v) & (u, v) \notin E \\ u \text{ 到 } v \text{ 的路径条数的最大值} & (u, v) \in E \end{cases}$$

下面我们给出 k -点连通分量的定义。

定义 2.5.2 (k -点连通分量). 对于无向简单图 $G = (V, E)$, 若集合 $S \subseteq G$ 满足 $\forall u, v \in S (u \neq v)$, 均有 $\kappa^*(u, v) \geq k$, 且不存在 S 的一个真超集⁹满足上述性质, 则称 S 是 G 的一个 k -点连通分量。

我们考虑建立一张辅助图 $H = (V, E_H)$, 其中 $E_H = \{(u, v) \mid u \neq v, \kappa^*(u, v) \geq k\}$ 。

当我们对边连通建图时, 由于 k -边连通是等价关系, 因此所得到的图为若干个不相交完全图的并, 从而 G 中的一个边连通分量对应辅助图中的一个连通块。

而我们对点连通建图时, 由于 k -点连通没有传递性, 因此不同的点连通分量之间可能有交集, 同样, 当 $k \geq 3$ 时, 一个 k -点连通分量并不一定能导出一个连通子图。

由定义知, G 的一个 k -点连通分量对应辅助图 H 中的一个极大团。因此 G 的 k -点连通分量的结构可以由 H 的极大团的结构来得到。

定理 2.5.2. n 阶无向图至多有 n 个 k -点连通分量。 □

对于建立辅助图的过程, 除了暴力对每一对点对使用局部点连通度的算法外, 笔者尚未获得低于 $O(|V|^2 \text{Flow}(|V|, |E|))$ 的算法, 如果有优于此复杂度的做法, 欢迎与笔者交流。

考虑 H 中两个极大团 A, B , 它们的交集大小是有限制的。事实上, 我们有如下结论:

定理 2.5.3. 对于 H 中任意两个不同的极大团为 A, B (对应 G 中两个不同 k -点连通分量), 则 $|A \cap B| \leq k - 1$ 。

⁹若 A 是 B 的真子集, 则称 B 是 A 的真超集。

证明. 若 $|A \cap B| \geq k$, 则由于 A, B 的极大性知 $A \not\subseteq B, B \not\subseteq A$, 从而存在 $u \in A \setminus B, v \in B \setminus A$, 满足 $(u, v) \notin H$.

由定义知 $\kappa^*(u, v) \leq k - 1$, 那么分两种情况讨论:

- $(u, v) \notin E$.

于是存在大小不超过 $k - 1$ 的集合 S 割掉 u, v , 从而存在 $x \in (A \cap B) \setminus S$.

因为 $\kappa^*(u, x) \geq k, \kappa^*(v, x) \geq k$. 因此在图 $G[V \setminus S]$ 中 u, x 连通, v, x 连通, 从而 u, v 连通, 矛盾.

- $(u, v) \in E$.

于是存在大小不超过 $k - 2$ 的集合 S , 使得 $G[V \setminus S]$ 中 (u, v) 是大小为 1 的边割集, 从而存在 $x \in (A \cap B) \setminus S$.

因为 $\kappa^*(u, x) \geq k$, 因此 $G[V \setminus S]$ 中 $\kappa^*(u, x) \geq 2$, 于是即使删去边 (u, v) 后 u, x 仍连通. 同理 v, x 也连通, 于是删去 (u, v) 后 u, v 连通, 矛盾.

即任意两个极大团的交至多有 $k - 1$ 个顶点. □

2.6 总结

本节主要介绍了对一般的正整数 k , k -点 (边) 连通性的判断和点 (边) 连通分量的一些可行方法.

事实上, 对于无向图边连通所建立的网络, 每条边权均为 1, 因此使用 Dinic 算法就有 $O(\text{Flow}(|V|, |E|)) = O(|E|^{\frac{3}{2}})$ ¹⁰. 当然, 如果我们只需判定最大流是否 $\geq k$, 第一项还可以对 k 取 \min , 即 $O(\min\{k, \sqrt{|E|}\} |E|)$.

而对于点连通所建立的网络流, 不难发现形如 x 的点只有一条出边, 形如 x' 的点只有一条入边, 因此此时使用 Dinic 算法就有 $O(\text{Flow}(|V|, |E|)) = O(\sqrt{|V|} \cdot |E|)$. 同样, 如果只需判定是否 $\geq k$, 复杂度可降为 $O(\min\{k, \sqrt{|V|}\} |E|)$.

下面将上述所有的算法列成一张表, 其中假设所有的网络流使用最常见的 Dinic 算法:

(注 1: 对于其中的某些问题, 学术界可能存在更优的做法, 不过由于实现过于复杂或实际意义不大等原因这里就不给出了. 本节讨论的算法都是相比之下较为实用的算法)

(注 2: 对于某些问题, 需要通过具体的 $k, |V|, |E|$ 来得出复杂度, 因为对应的界比较多 (如 $\kappa = O\left(\frac{|E|}{|V|}\right), \text{Flow}(|V|, |E|) = O(\min\{k, \sqrt{|V|}\} |E|)$, 等等。)

¹⁰特别地, 如果原无向图是简单图, 则还有一个更紧的上界: $O(\min\{|V|^{\frac{2}{3}}, \sqrt{|E|}\} |E|)$.

算法	点连通	边连通	其它算法
局部 k -连通判定 (单次)	$O(\min\{k, \sqrt{ V }\} E)$	$O(\min\{k, \sqrt{ E }\} E)$	-
局部连通度 (单次)	$O(\sqrt{ V } \cdot E)$	$O(E ^{\frac{3}{2}})$	-
全局 k -连通判定	$O(k \min\{k, \sqrt{ V }\} V E)$	$O(\min\{k, \sqrt{ E }\} V E)$	[边, 确定性] $O(V ^2 \log V + V E)$
全局连通度	$O(\sqrt{ V } \cdot E ^2)$	$O(V E ^{\frac{3}{2}})$	[边, 随机化] $O(V ^2 \log^3 V)$
k -连通分量划分	-	$O(\min\{k, \sqrt{ E }\} V E)$	-
局部 k -连通判定 (多次)	$O(\min\{k, \sqrt{ V }\} V ^2 E)$ - $O(1)$	$O(\min\{k, \sqrt{ E }\} V E)$ - $O(1)$	-
局部连通度 (多次)	-	$O(V E ^{\frac{3}{2}}) - O(1)$	-

3 1-连通与 2-连通

接下来我们着重讨论 k 较小的情形。

本文将讨论 $k = 1, 2, 3$ 的情形。对于 $k = 4$ 的情形，虽然学术界有对应的算法，不过由于实用性不大，而且实际表现并不见得比之前一般 k 的算法优秀，这里就暂且略去了。

$k = 3$ 的情形相较于 $k = 1, 2$ 更为复杂，因此本节先讨论 $k = 1, 2$ 的情形。

3.1 1-连通

由定义立即可知，对于无向图 G ， G 是连通图当且仅当 G 是 1-点连通图，当且仅当 G 是 1-边连通图。

因此问题转化为连通性的相关问题，使用 dfs、bfs 或并查集的技巧即可完成，这里不再赘述。

3.2 2-连通

2-连通分为两种：2-点连通和 2-边连通。在讨论 2-连通之前，我们有必要提及一下图的 dfs 生成树 (dfs 树)。

定义 3.2.1 (dfs 生成树). 对于一张连通无向图 $G = (V, E)$ ，任取一个点 r 为根进行深度优先搜索 (dfs)，则每个顶点 $v \in V \setminus \{r\}$ 在 dfs 上都有一个前趋顶点 (记为 p_v)，所有形如 (v, p_v) 的边构成一棵树，称为图 G (以 r 为根) 的一棵 dfs 生成树，简称 dfs 树，记为 T_r 。dfs 树上的边称为树边，其它的边称为非树边。

dfs 树可以看成无根树也可以看成有根树，不过为了方便起见一般看成以 r 为根的有根树。通常情况下，dfs 树可以看成外向树(根指向叶子)、内向树(叶子指向根)和无向树，具体要根据上下文决定。

对于不连通的图 G ，我们需要对每个连通分量进行 dfs，从而可以类似得到一个 dfs 森林。

下面的这个定理，是无向图 dfs 树的重要性质：

定理 3.2.1. 设 T_r 连通无向图 G 的一棵 dfs 树，则所有的非树边都连接 T_r 的某个顶点和其祖先顶点。 □

因为这个原因，我们也把无向图的非树边称为**返祖边**(返回边，回边，back-edge)。

3.3 2-边连通¹¹

首先来看 2-边连通，2-边连通也称为**边双连通**，2-边连通分量也称为**边双连通分量**。

3.3.1 2-边连通分量和桥边

定义 3.3.1 (桥边). 若单边集 $\{e\}$ 为连通图 G 的某个边割集，则称 e 是 G 的**桥边** (bridge)，也称**割边**。

2-边连通有如下独有的性质：

定理 3.3.1. 若 B 是连通图 G 的一个 2-边连通分量，则 $G[B]$ 是 2-边连通图。

证明. 取 $u, v \in B$ ，由 $\kappa(u, v) \geq 2$ 知存在从 u 到 v 两条边不相交的路径。

于是我们可以得到经过 u, v 的有向回路，从而回路上的所有点两两 2-边连通。于是 u, v 在 $G[B]$ 中连通且 2-边连通。 □

推论 3.3.1. B 是 G 的 2-边连通分量当且仅当 $G[B]$ 是 G 的极大 2-边连通子图。 □

考虑连通图 G 的 dfs 树，对于每条非树边 e ，由**定理 3.2.1** 知它是返祖边，连接某个点 v 和其祖先顶点 u 。对于这种情形，我们称 e **覆盖**了路径 $u \rightsquigarrow v$ 上所有的树边。

定理 3.3.2. 对于连通无向图 G ， e 是桥边当且仅当 e 是树边且 e 没有被任意一条返祖边覆盖。 □

¹¹本小节内容默认假设图是连通图。

于是可以使用树上差分的技巧计算出每条边是否被返祖边覆盖，来得到所有的桥边。

考虑所有被覆盖的树边构成的图，它们是一个森林。容易发现，这个森林中的每个连通分量对应 G 的一个 2-边连通分量。

从而找出所有桥边后，将它们去掉后，剩下的边 (或树边) 构成的连通分量，就是原图的 2-边连通分量。

上述算法的时间复杂度为 $O(|V| + |E|)$ 。

我们尝试发掘更深的性质。

考虑有根树上的一个连通子图 S ，则 S 中有唯一的顶点具有最小的深度。

于是，每个 2-边连通分量，对应到 dfs 树上是一个连通子图，于是它存在深度最小的顶点。

对于 $v \in G$ ，定义 low_v ，表示以 v 为根的子树的点 u 中，通过返祖边 (u, w) 能到达的点 w 中深度最小者。

根据上述定义可知，每个 2-边连通分量都有且仅有一个点满足 $v = low_v$ ，且这个 v 就是深度最小的顶点。

为了快速完成这个任务，同时避免记录 (深度, 标号) 二元组的麻烦，我们可以引入 dfs 序——dfs 时访问顶点的顺序。下文用 seq_i 表示 dfs 时访问的第 i 个顶点， id_i 或 dfn_i 表示顶点 i 访问的时刻 (即第几个被访问)。容易发现， seq_i 和 id_i 互为逆置换的关系。

dfs 序的一个重要性质是：

定理 3.3.3. 若 u 是 dfs 树中 v 的祖先，则 $id_u \leq id_v$ 。 □

注意到 low_v 一定是 v 的祖先，因此定义中可以被换成标号最小者。方便起见，不妨设 dfs 序就是 $1, 2, \dots, n$ 。

现在，考虑对每个 v 求出 low_v ，可以使用树形 DP：

- 考虑和 v 关联的所有返祖边，对于每个 (v, u) ，令 $low_v = \min\{low_v, u\}$ 。
- 考虑 v 的所有子节点，对于每个子节点 c ，令 $low_v = \min\{low_v, low_c\}$ 。

当遇到 $v = low_v$ 的顶点时，将以 v 为根的子树取出来，作为一个 2-边连通分量，并将该子树删去。同时，如果 v 存在父节点 p_v ，则边 (p_v, v) 是桥边。

这就是 **Tarjan 算法**，时间复杂度 $O(|V| + |E|)$ 。

3.3.2 应用——Robbins 定理

2-边连通图的一个实际应用是 Robbins 定理——2-边连通图的强连通定向。

定理 3.3.4 (Robbins). 无向简单图 G 能定向成强连通 (有向) 图的充分必要条件是， G 是 2-边连通图。

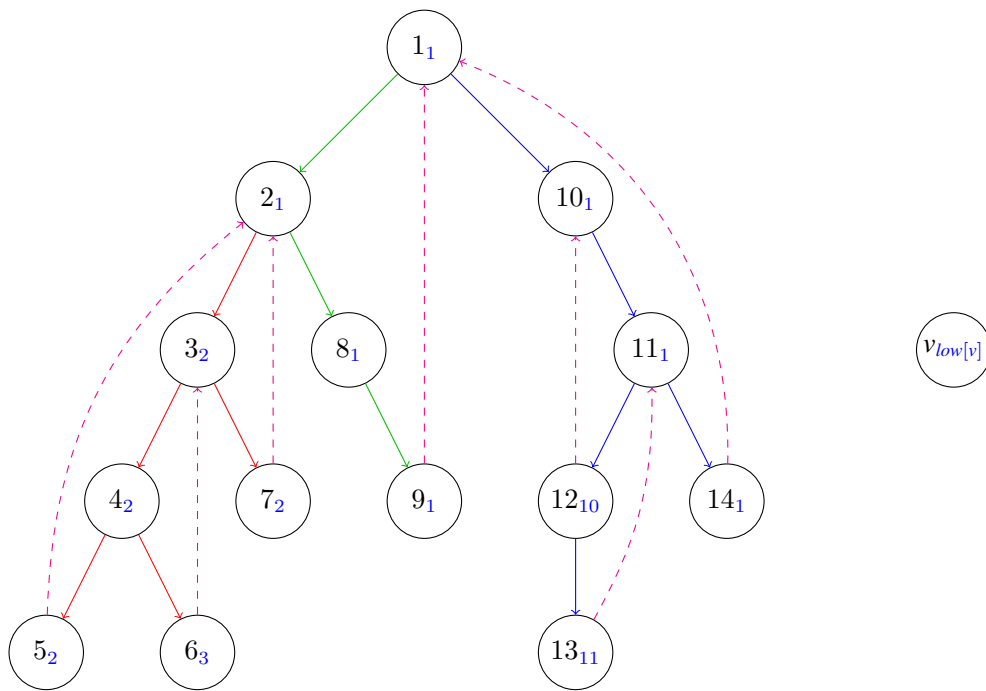


图 5

证明. 必要性: 若 G 不是 2-边连通图, 则 G 存在桥边 (u, v) , 于是无论如何定向, 都不可能同时存在 $u \rightsquigarrow v$ 的有向路径和 $v \rightsquigarrow u$ 的有向路径, 矛盾。

充分性: 设 G 是 2-边连通图, 设它的一棵 dfs 树为 T , 将所有树边按照外向树定向 (父节点指向子节点), 所有返祖边定向为“后代指向祖先”。下面证明这是一个强连通定向。

首先, 由外向树的性质知根可以通向所有节点, 因此只需要证明每个顶点 v 也可以通向根。

又因为 G 是 2-边连通图, 因此在 $Tarjan$ 算法中有 $\underbrace{low[low[\dots low[v]\dots]]}_{\infty} = 1$ 。¹²

而由 $Tarjan$ 算法和定向规则可知, v 可以通向 low_v , 从而可以不断通过此规则回到点 1 (即根节点), 证毕。 □

上述证明同时也给出了非常简单且容易实现的构造, 时间复杂度 $O(|V| + |E|)$ 。

3.4 2-点连通¹³

和边连通类似, 2-点连通也称为点双连通。

定义 3.4.1 (割点). 若单点集 $\{v\}$ 为连通图 G 的某个点割集, 则称 v 是 G 的割点 ($cut\ vertex$)。

¹²这里不妨设 dfs 序为 $1, 2, \dots, n$ 。

¹³本小节内容默认假设图是连通图。

3.4.1 2-点连通分量和点双连通分量

但是，由于对 2 阶完全图 K_2 点连通度的定义问题点双连通分量的定义会有少许区别。

为了方便起见，我们人为规定 K_2 是 2-点连通图，即 $\kappa(K_2) = 2$ 。

定义 3.4.2 (点双连通分量). 对于连通无向图 $G = (V, E)$ ，对于 $B \subseteq V$ ，若 $G[B]$ 是 G 的极大 2-点连通子图，则称 B 是 G 的点双连通分量。

实际上，当定义 $\kappa(K_2) = 1$ 时，点双连通分量的定义和 2-点连通分量是完全一致的。而当 $\kappa(K_2) = 2$ 时，所有的 2-点连通分量仍然是点双连通分量，多出来的点双连通分量均为二元集 $\{u, v\}$ ，满足 (u, v) 是 G 的桥边。当 $|V| > 1$ 时，所有点双连通分量都至少有两个点。

在后面的过程中，会逐渐发现，这么定义点双连通分量对整个算法流程是有益的。

定理 2.5.2 告诉我们， n 个点的图至多有 n 个 2-点连通分量。事实上，可以证明， n 个点的图也至多有 n 个点双连通分量。

3.4.2 边的等价性

由于点连通对点来说不是等价关系，但 2-点连通对边来说是一种等价关系：

在 E 上定义二元关系 ρ_c ，其中 $(e, f) \in \rho_c$ 当且仅当它们可以同时在一个圈中。

定理 3.4.1. $(e, f) \in \rho_c$ 当且仅当 e, f 在同一个点双连通分量中。

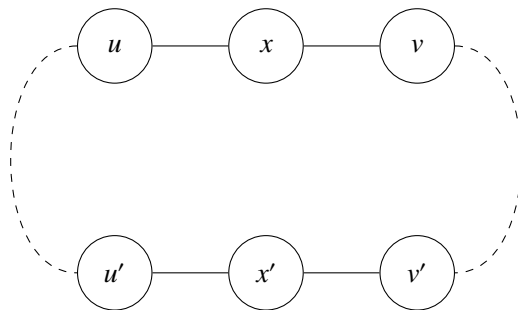


图 6

证明. 若 $(e, f) \in \rho_c$ ，则 e, f 同时在一个圈中，于是 e, f 的所有端点都是两两 2-点连通的，从而这些端点在同一个点双连通分量中。

考虑点双连通图中的两条边 $e = (u, v), f = (u', v')$ 。我们将 e 换成 (u, x) 和 (x, v) ， f 换成 (u', x') 和 (x', v') (裂边)。

则新图仍然是点双连通的，于是至少两条从 x 到 x' 的路径，它们拼起来就可以构成一个圈。由 x 和 x' 的构造方式知，这个圈经过边 e 和 f 。□

现在让我们回到 dfs 树。根据上述结论知，dfs 树上的 $n - 1$ 条树边可以根据 ρ_c 划分成若干个等价类。如图 5 中每种颜色的树边表示一个 ρ_c 的等价类。

考虑一个等价类，由之前的结论知它是一个点双连通分量中的所有树边，因此它必然对应 dfs 树上的一个连通块。

于是，我们对于每条返祖边 (v, u) (u 为 v 的祖先)，将这些路径上的所有边设为等价，最终树上每一种等价类对应的连通子图就是原图的一个点双连通分量。

对于一个点双连通分量 B ，在 dfs 树上仍然是一个连通子图，且其中有唯一顶点具有最小深度。

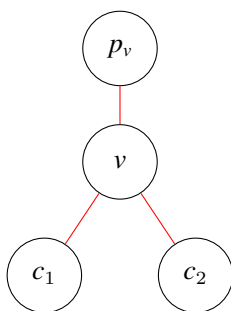


图 7

如图 7，设该顶点为 v ，则 v 的子节点中只有至多一个属于 B 。否则设 $c_1, c_2 \in B$ ，则边 (c_1, v) 和边 (c_2, v) 的等价，于是它们必然和 (v, p_v) 等价，这与 v 深度的最小性矛盾。

设满足该条件的唯一子节点为 u ，则必然有 $v \leq low_u$ ¹⁴。反之，如果一个顶点 v 和其子节点 u 满足 $v \leq low_u$ ，则此时 u 所在的子树 (的所有边) 连同边 (u, v) 共同构成了 ρ_c 的等价类，即我们得到了一个点双连通分量。于是我们将以 u 为根的子树删去，然后重复运行即可。

这就是 Tarjan 求点双连通分量的算法，时间复杂度 $O(|V| + |E|)$ 。

3.4.3 割点

如何判断一个点 v 是否为割点呢？我们有如下结论：

定理 3.4.2. 点 v 不是割点，当且仅当和 v 关联的所有边在同一个 ρ_c 等价类中 (颜色相同)。□

于是，如果这些边不全在同一个等价类中，则必然有一条边 (u, v) (u 为 v 的子节点) 与 (v, p_v) 不在一个等价类，从而存在 u 满足 $v \leq low_u$ 。

当然，有一个例外， v 是根节点。对于根节点的情况，可以注意到在 dfs 树中，与根节点关联的所有边两两不 ρ_c 等价，因此对于根节点只需要判断它是否有两棵及以上子树即可。

该算法的时间复杂度仍为 $O(|V| + |E|)$ 。

¹⁴当 (v, u) 为桥边时有 $v < low_u$ ，否则有 $v = low_u$ ，总之有 $v \leq low_u$ 。

3.4.4 应用——块割树和圆方树

不同的点双连通分量可能有交，因此我们在求解时不得不使用边的等价类来处理。

那点双连通分量的主角还是点，对那些点来说又有哪些性质呢？

首先，由定理 2.5.3 知，两个点双连通分量的交集至多只有 1 个点。事实上，如果两个点双连通分量的有交点 v ，则 v 必定是原图的割点。

定义 3.4.3 (块割树). 对连通无向图 G ，我们可以定义图 $H = (B \cup C, J)$ ，满足：

- B 是 G 中所有点双连通分量 (作为点集) 构成的集合。
- C 是 G 中所有割点构成的集合。
- $J = \{(b, c) \mid b \in B, c \in b \cap C\}$ 。

则称 H 为 G 的块割树¹⁵。

对于图 8 所示的这张图，一共有 5 个 2-点连通分量和 7 个点双连通分量。它的块割树如图 9：

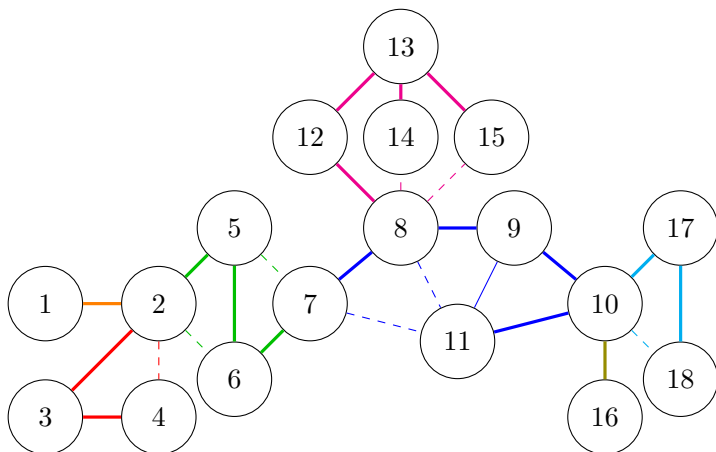


图 8

不难证明连通无向图的块割树一定是树。

得到一张图的块割树也是不难的。由于割点一定在块割树上，因此我们只需以任意一个割点为根进行 dfs，然后每次找到一个 (最浅点为 v) 的点双连通分量 B 时，将 B 中的所有割点连一条边向 B ，然后将 B 连向 v 。

不过在实际应用中，我们不仅仅只要割点，其它的点也需要考虑进来。因此通常会使用块割树的广义版本——圆方树。

¹⁵block-cut tree, 见 [7]。

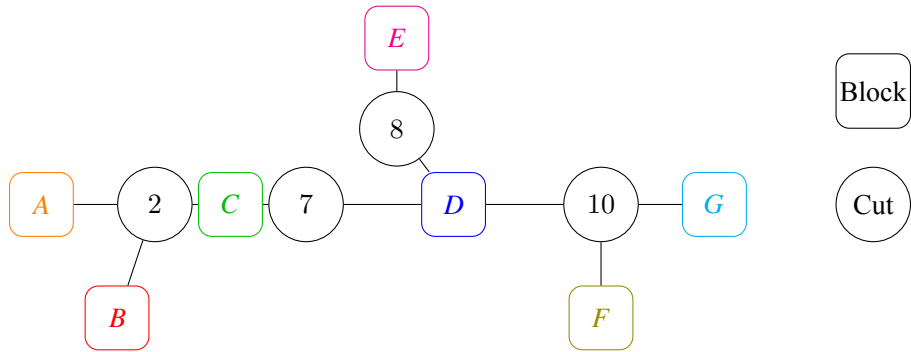


图 9

定义 3.4.4 (圆方树). 对连通无向图 $G = (V, E)$, 我们可以定义图 $H = (B \cup V, J)$, 满足:

- B 是 G 中所有点双连通分量 (作为点集) 构成的集合。
- $J = \{(b, v) \mid b \in B, v \in b\}$ 。

则称 H 为 G 的圆方树。其中 B 中的点称为方点 (块点), V 中的点称为圆点 (普通点)。

如, 图 8 的圆方树是图 10。

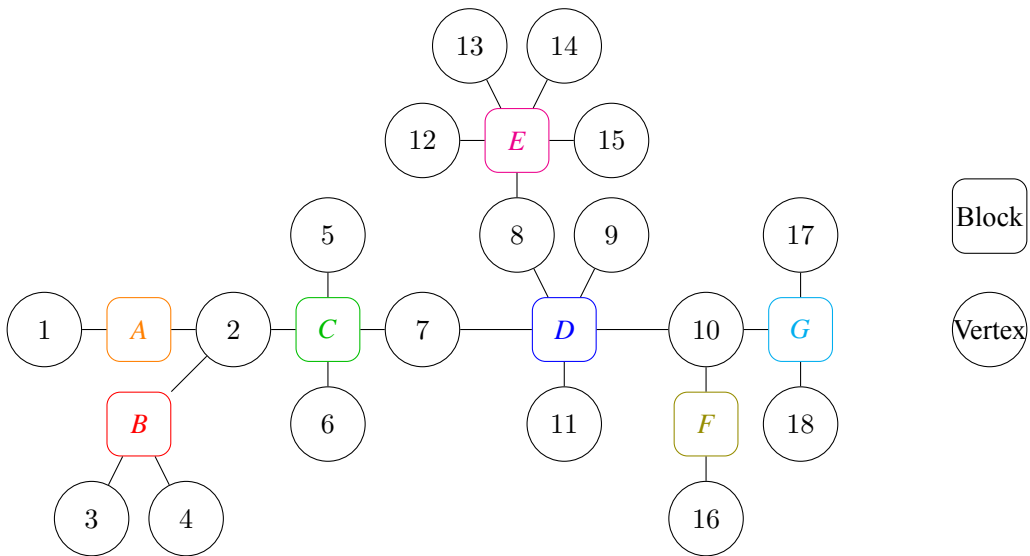


图 10

由定义可知,

定理 3.4.3. 块割树是圆方树中所有非叶节点的导出子图, 它们的每条边连接一个圆点和一个方点。 □

和块割树类似，圆方树也可以通过一次 Tarjan 算法在 $O(|V| + |E|)$ 时间内得到。圆方树可以将许多和图上路径有关的问题转化为对应的树上问题，从而使用传统的树上算法处理，是一个化“图”为“树”的利器。

4 3-连通及其应用

3-连通是图论中较为现代的理论，同样分为 3-边连通和 3-点连通。同样它们各有各的算法及应用。

4.1 3-边连通¹⁶

4.1.1 切边和切边对

定义 4.1.1 (切边, 切边对). 对于 2-边连通的无向图 G ，若某个二元边集 $\{e_1, e_2\}$ ($e_1 \neq e_2$) 是 G 的边割集，则称 (e_1, e_2) 是一对切边对 (cut pair)，其中 e_1, e_2 都被称为切边 (cut edge)。

切边对具有如下性质：

定理 4.1.1. 对于 2-边连通无向图 $G = (V, E)$ ， $e_1, e_2 \in E$ (e_1, e_2 不是同一条边)。则 (e_1, e_2) 为切边对当且仅当对于 G 中的每个圈 C ，要么 $e_1 \in C \wedge e_2 \in C$ ，要么 $e_1 \notin C \wedge e_2 \notin C$ (即 $(e_1 \in C) \Leftrightarrow (e_2 \in C)$ ¹⁷)。

证明. 充分性：若 e_1, e_2 满足对在所有圈中要么同时出现或同时不出现，这说明删去 e_1 后， e_2 不能在任何一个圈上。

从而 e_2 是 $G \setminus \{e_1\}$ 的桥边，即 (e_1, e_2) 是切边对。

必要性：若 (e_1, e_2) 是切边对，则 e_2 是 $G \setminus \{e_1\}$ 的桥边。

这说明，在 G 中通过 e_2 的圈必定通过 e_1 ，同理通过 e_1 的圈必定通过 e_2 ，即它们在所有圈中要么同时出现，要么同时不出现。 \square

4.1.2 切边等价

从而我们可以引入切边等价的概念：

定义 4.1.2 (切边等价). 对于 2-边连通无向图 $G = (V, E)$ 和 $e_1, e_2 \in E$ (e_1 和 e_2 不必不同)，如果对 G 中的每个圈 C ，都有 $(e_1 \in C) \Leftrightarrow (e_2 \in C)$ ，则称 e_1, e_2 切边等价，记作 $e_1 \stackrel{\sim}{\sim} e_2$ 。

由定理 4.1.1 可知，当 $e_1 \neq e_2$ 时， $e_1 \stackrel{\sim}{\sim} e_2$ 当且仅当 (e_1, e_2) 是切边对。

¹⁶本小节内容默认假设图是 2-边连通图，允许有重边，但不允许有自环。

¹⁷ \Leftrightarrow 表示逻辑等价，见 [9]。

定理 4.1.2. 切边等价是等价关系。

证明. 显然切边等价具有自反性和对称性。现在设 $e_1 \sim e_2, e_2 \sim e_3$, 则对于 G 中的每个圈 C , 都有 $(e_1 \in C) \Leftrightarrow (e_2 \in C), (e_2 \in C) \Leftrightarrow (e_3 \in C)$ 。由 \Leftrightarrow 的传递性知 $(e_1 \in C) \Leftrightarrow (e_3 \in C)$, 从而 $e_1 \sim e_3$ 。□

于是, “切边等价” 关系将边集 E 划分成若干个等价类 E_1, E_2, \dots, E_n 。

4.1.3 和 3-边连通的关系

定义 4.1.3 (收缩). 对于无向图 $G = (V, E)$ 和边 $e = (u, v) \in E$, 定义 G 对 e 收缩 (edge contraction) 所得的图为 $H = (V', E')$, 其中:

- 定义一个新的顶点 w , 则 $V' = (V \setminus \{u, v\}) \cup \{w\}$ 。
- 对于 E 中的边 (x, y) , 若 $x \notin \{u, v\} \wedge y \notin \{u, v\}$, 则 (x, y) 也在 E' 中; 否则不妨设 $x \in \{u, v\}$, 则将对应的边换成 (w, y) 。
- 最终去掉所得图中的所有自环。¹⁸

G 对 e 收缩的图记为 $G \cdot e$ 或 G/e 。

图 11 即为一次收缩操作的例子 (图来源 [10])。

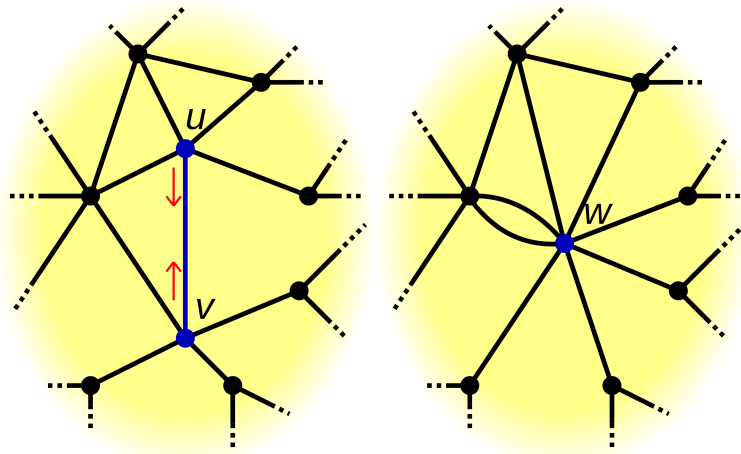


图 11

注意, 简单图经过收缩后可能成为非简单图 (如图 11)。

切边等价和 3-边连通的关系由下面两个定理建立:

¹⁸这是 3-边连通所需要的特殊要求, 也为了简化问题所考虑。

定理 4.1.3. 设 G 是 2-边连通无向图。若边 $e \in (u, v)$ 满足它所在的 \sim 等价类大小为 1, 则 u, v 在同一个 3-边连通分量中。

同时, 令 $H = G \cdot e$, 则 H 的 3-边连通分量划分就是将 G 的 3-边连通分量划分将 u, v 替换为 w 后的结果, 且 H 和 G 中对应边具有相同的切边等价关系。

证明. 上述定理包含三个命题, 我们逐一证明。

1. 若 $e = (u, v)$ 所在的等价类大小为 1, 说明 e 不是切边。

于是由切边的定义知 $\lambda(u, v) \geq 3$, 即 u, v 在同一个 3-边连通分量中。

2. 设 $\lambda(x, y) \geq 3$, 如果 $x \notin \{u, v\} \wedge y \notin \{u, v\}$, 由定理 2.1.1 知, 存在 3 条从 x 到 y 边不相交的路径。同时, 由收缩过程知对应的路径仍然存在 (注意收缩是保留重边的), 因此在 H 中仍然存在 3 条从 x 到 y 边不相交的路径。即 $\lambda(x, y) \geq 3 \Rightarrow \lambda(x', y') \geq 3$ 。当 $x, y \in \{u, v\}$ 时同理可证。

若 $\lambda(x', y') \geq 3$ 且 $\lambda(x, y) < 3$, 说明 x, y 存在大小为 2 的边割。由于 (u, v) 不可能作为切边, 因此这两条边在 H 中都是对应存在的。

3. 注意到 G 中的每个圈可以通过收缩操作对应到 H 中的每个圈, 反之亦然, 于是 G, H 具有相同的切边等价关系。

□

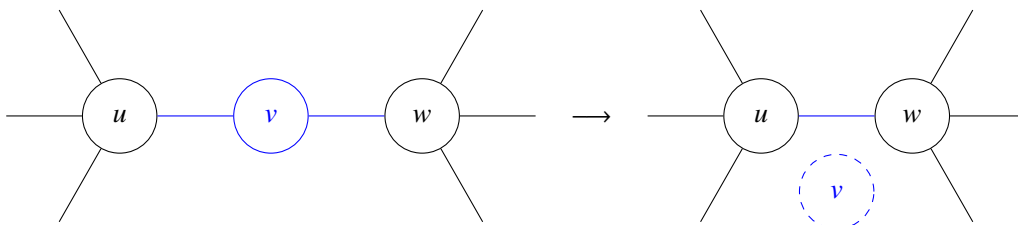


图 12

定理 4.1.4. 设 G 是 2-边连通无向图, 其中 $d(v) = 2$ 。设 $N(v) = \{u, w\}$ (其中 u 可以等于 w), $\{v\}$ 为一个单独的 3-边连通分量。

同时, 令 H 为在 $G \setminus \{v\}$ 中, 加入边 (u, w) 的图 (如果 $u = w$ 则无需添加自环)。则 H 的 3-边连通分量划分就是将 G 的 3-边连通划分将 $\{v\}$ 删去后的结果, 且对应边具有相同的切边等价关系。

证明. 同样依次证明上述三个命题。

1. 对于 $\forall x \in V \setminus \{v\}$, 由于 $(u, v), (w, v)$ 均为 v 和 x 的大小为 2 的点割集, 因此 $\lambda(v, x) \leq 2$ 。
于是 $\{v\}$ 自成一个 3-边连通分量。
2. 设 $\lambda(x, y) \geq 3$ ($x \neq v \wedge y \neq v$), 由定理 2.1.1 知存在 3 条从 x 到 y 边不相交的路径。
我们将 (u, v) 和 (v, w) 替换为 (u, w) 即可找到 3 条 H 中对应的路径, 反之亦然。故 $\lambda_G(x, y) \geq 3 \Leftrightarrow \lambda_H(x', y') \geq 3$ 。
3. G 中的每个圈仍然可以通过将 (u, v) 和 (v, w) 替换为 (u, w) 对应到 H 中的每个圈, 反之亦然, 故切边等价关系不变。

□

4.1.4 一个想法

反复利用定理 4.1.3 和定理 4.1.4, 我们可以将一张图的规模不断缩小, 从而得到原图的 3-边连通分量划分。

事实上, 这个思路是可行的, 我们需要依赖如下定理:

定理 4.1.5. 设 G 为 2-边连通无向图, 满足 $\delta(G) \geq 3$, 则必定存在一条边 (u, v) , 满足 (u, v) 不和其它任何边切边等价 (即等价类大小为 1)。

在证明这个定理前, 我们需要介绍切边等价和 dfs 树的关系。

设 T 为 2-边连通无向图 G 的 dfs 树, 由之前的部分知每条返祖边覆盖了若干条树边。我们对于每条边 e , 定义一个集合 $C(e)$:

- 若 e 是返祖边, 则 $C(e) = e$ 。
- 若 e 是树边, 在令 $C(e)$ 为所有覆盖它的返祖边的集合。形式化地,

$$C(e) = \{(u, v) \mid (u, v) \in G \setminus T; u, v \text{ 在 } T \setminus \{e\} \text{ 中不连通}\}$$

可以发现, $C(e)$ 中的元素全是返祖边。由于 G 是 2-边连通图, 因此由定理 3.3.2 知 $\forall e \in E, C(e) \neq \emptyset$ 。

事实上, 集合 $C(e)$ 和切边等价有着密切的关系:

定理 4.1.6. 对于 2-边连通无向图 $G = (V, E)$ 和 $e_1, e_2 \in E$, $e_1 \sim e_2$ 当且仅当 $C(e_1) = C(e_2)$ 。

证明. 考虑任意一条返祖边 (u, v) 。由定义知, 所有满足 $(u, v) \in C(e)$ 的边 e 构成 G 中的一个圈 C 。

如果 $e_1 \sim e_2$, 那么对于该圈 C , 有 $(e_1 \in C) \Leftrightarrow (e_2 \in C)$, 从而 $(u, v) \in C(e_1) \Leftrightarrow (u, v) \in C(e_2)$, 由 (u, v) 的任意性知 $C(e_1) = C(e_2)$ 。

反之, 若 $C(e_1) = C(e_2)$, 则对于每条只包含一条返祖边的圈 C , 均有 $(e_1 \in C) \Leftrightarrow (e_2 \in C)$ 。

注意到若对于边集 A, B , 成立 $(e_1 \in A) \Leftrightarrow (e_2 \in A)$, 则对于 $A \oplus B$ ¹⁹, 也成立 $(e_1 \in A \oplus B) \Leftrightarrow (e_2 \in A \oplus B)$ 。

由连通图的圈空间 [11] 的性质知, G 中的每一个圈均可以表示成若干个只包含一条返祖边的圈的对称差, 于是对于 G 中的每个圈 C , 都有 $(e_1 \in C) \Leftrightarrow (e_2 \in C)$, 即 $e_1 \sim e_2$ 。 □

引理 4.1.1. 设 G 为 2-边连通无向图。若树边 e_1, e_2 切边等价 ($e_1 \sim e_2$), 则这两条边一定是祖先-后代关系。

证明. 设 $e_1 = (u, p_u), e_2 = (v, p_v)$ 。由于 $e_1 \sim e_2$, 由定理 4.1.6 知 $C(e_1) = C(e_2) \neq \emptyset$ 。

任取 $e \in C(e_1)$, 知非树边 e 覆盖 e_1, e_2 , 而由定理 3.2.1 知 e 是返祖边, 从而 e_1, e_2 一定是祖先-后代关系。 □

引理 4.1.2. 设 \mathcal{P} 是 dfs 树上一条从根到某个顶点的路径。则以下情形不会出现: \mathcal{P} 中按顺序存在四条边 e_1, e_2, e_3, e_4 , 满足 $e_1 \sim e_3, e_2 \sim e_4$, 但 $e_1 \not\sim e_2$ 。

证明. 反设存在这种情况, 如图 14, 考虑 $e \in C(e_1)$, 知 $e \in C(e_3)$, 从而 e 覆盖 $e_1, e_3 \Rightarrow e$ 覆盖 e_2 , 即 $e \in C(e_2) \Rightarrow e \in C(e_4)$ 。

同理, $e \in C(e_4) \Rightarrow e \in C(e_1)$, 于是 $C(e_1) = C(e_4), e_1 \sim e_2$, 矛盾。 □

下面就可以证明定理 4.1.5 了。

证明 (4.1.5).

考虑所有 \sim 等价类中, 最浅的树边的上端顶点最深的那个等价类。设该等价类中最浅的树边为 $e = (v, p_v)$, 则依次证明:

- 树中没有其它的边 f 满足 $e \sim f$ 。

反之, f 在以 v 为根的子树中。设 $f = (u, p_u)$, 如果 $p_u \neq v$, 由引理 4.1.2 知中间其它边所在的等价类, 最浅的树边更加深。

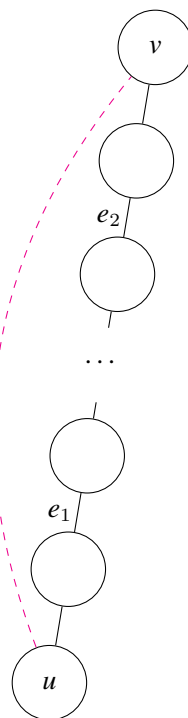


图 13

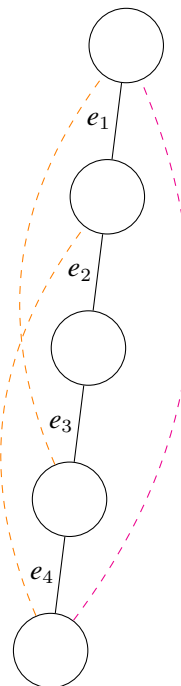


图 14

¹⁹⊕ 在这里表示集合的对称差。

如果 $p_u = v$, 由于 $d(v) \geq 2$, 因此 v 不能引其它返祖边, 则 v 必定还有其它子节点 y 。于是 (v, y) 所在等价类中, (v, y) 作为最浅树边, 与深度最大性矛盾。

- 没有返祖边 f 满足 $e \lesssim f$ 。

否则, 说明只有返祖边 $f = (a, b)$ (b 为 a 的祖先) 覆盖了 e 。若 $b \neq u$, 则 (b, p_b) 所在等价类的深度比 e 大, 若 $b = u$, 则有 $d(v) \geq 2$ 知 v 还有其它子节点 y , 于是 (v, y) 所在等价类的深度比 e 大, 矛盾。

于是 e 所在的等价类大小为 1, 结论成立。 \square

定理 4.1.5 告诉我们, 在任一时刻, 要么有一个点的度为 2, 要么存在一个大小为 1 的边等价类。对于这两种情况, 我们可以利用定理 4.1.4 和定理 4.1.3, 不断将问题转化为规模更小的子问题。

4.1.5 实现

下面介绍如何实现这个算法。

首先需要能够快速判断两条边是否切边等价, 由定理 4.1.6 知我们只需判断它们对应的 $C(e)$ 是否相同。

而获取每条边的 $C(e)$ 需要 $O(|E|)^{20}$ 的时间, 又只有树边定义的 $C(e)$ 元素个数才会 ≥ 1 , 从而获取所有边的 $C(e)$ 需要 $O(|V||E|)$ 的时间。

这个显然是我们无法接受的, 因此考虑利用经典的随机化技巧: 对每一条非树边 e , 随机一个 64 位权值 $w(e)$, 然后定义²¹

$$c(e) = \bigoplus_{f \in C(e)} w(f)$$

于是当 $C(e_1) = C(e_2)$ 时一定有 $c(e_1) = c(e_2)$, 而当 $C(e_1) \neq C(e_2)$ 时 $c(e_1) = c(e_2)$ 的概率只有 $\frac{1}{2^{64}}$ 量级, 可以忽略不计。

现在, 我们将问题转化为了对树上的一条链异或上一个数, 最终询问每条边的值, 这是一个静态问题, 可以使用树上差分的技巧在 $O(|V| + |E|)$ 时间内解决。

设 $G = (V, E)$ 是 2-边连通图, 算法的流程如下:

²⁰ 由于 G 是连通图, 因此 $|V| \leq 1 + |E|$ 。

²¹ \oplus 在这里表示两个数的按位异或。

1. 对于每条边 e , 求出 $c(e)$, 以判断两条边是否切边等价。
2. 令 $A = \{e \mid e \in E, e \text{ 所在的切边等价类的大小为 } 1\}$, $D = \{v \mid v \in V, d(v) = 2\}$ (以下默认边集是可重集, 点集不是可重集)。
3. 对于每个顶点 v , 维护其当前点度 d_v 和它所在的“3-边连通分量” L_v , 起初 $d_v = d(v)$, $L_v = \{v\}$, 再使用并查集维护辅助图 H 。
4. 若 $A = \emptyset$ 且 $D = \emptyset$, 则转到步骤 15。
5. 若 $A \neq \emptyset$, 则转到步骤 6, 否则转到步骤 10。
6. 任取 $e = (u_0, v_0) \in A$, 令 $A = A \setminus \{e\}$ 。
7. 设 u, v 分别为 u_0, v_0 在 (并查集) H 中所在的连通分量。
8. 若 $u = v$, 则说明这两边的连通分量已经处理, 因此直接令 $d_u = d_u - 2$ 即可。
否则, 令 $L_u = L_u \cup L_v, L_v = \emptyset, d_u = d_u + d_v - 2$, 在 H 中将 u, v 所在的连通分量合并。
9. 检查此时是否有 $d_u = 2$, 如果是则令 $D = D \cup \{u\}$, 回到步骤 4。
10. 任取 $x \in D$, 令 $D = D \setminus \{x\}$ 。
如果 $d_x \neq 2$, 则回到步骤 4。²²
11. 枚举 L_x 中的顶点, 来找到当前与它关联的两条边, 记为 (x, u) 和 (x, v) 。
12. 如果 u, v 已经在 H 中同一个连通分量中, 则回到步骤 4。
13. 将边 (x, u) 和 (x, v) 合并为 (u, v) 。
14. 检查此时 (u, v) 所在的切边等价类大小是否为 1, 如果是则令 $A = A \cup \{(u, v)\}$, 回到步骤 4。
15. 此时, 所有非空的 L_i 构成 G 的 3-边连通分量一个划分。

[这里](#)是上述算法的一个实现, 可以看出代码不是很长。

²²这是因为可能经过某些操作后原本度数为 2 的点的度数又不是 2 了。

4.1.6 演示

下面用粉色点表示 2 度点，不同颜色的边表示不同 \sim 等价类，黑色边表示大小为 1 的等价类，紫色表示已完成划分的 3-边连通分量。

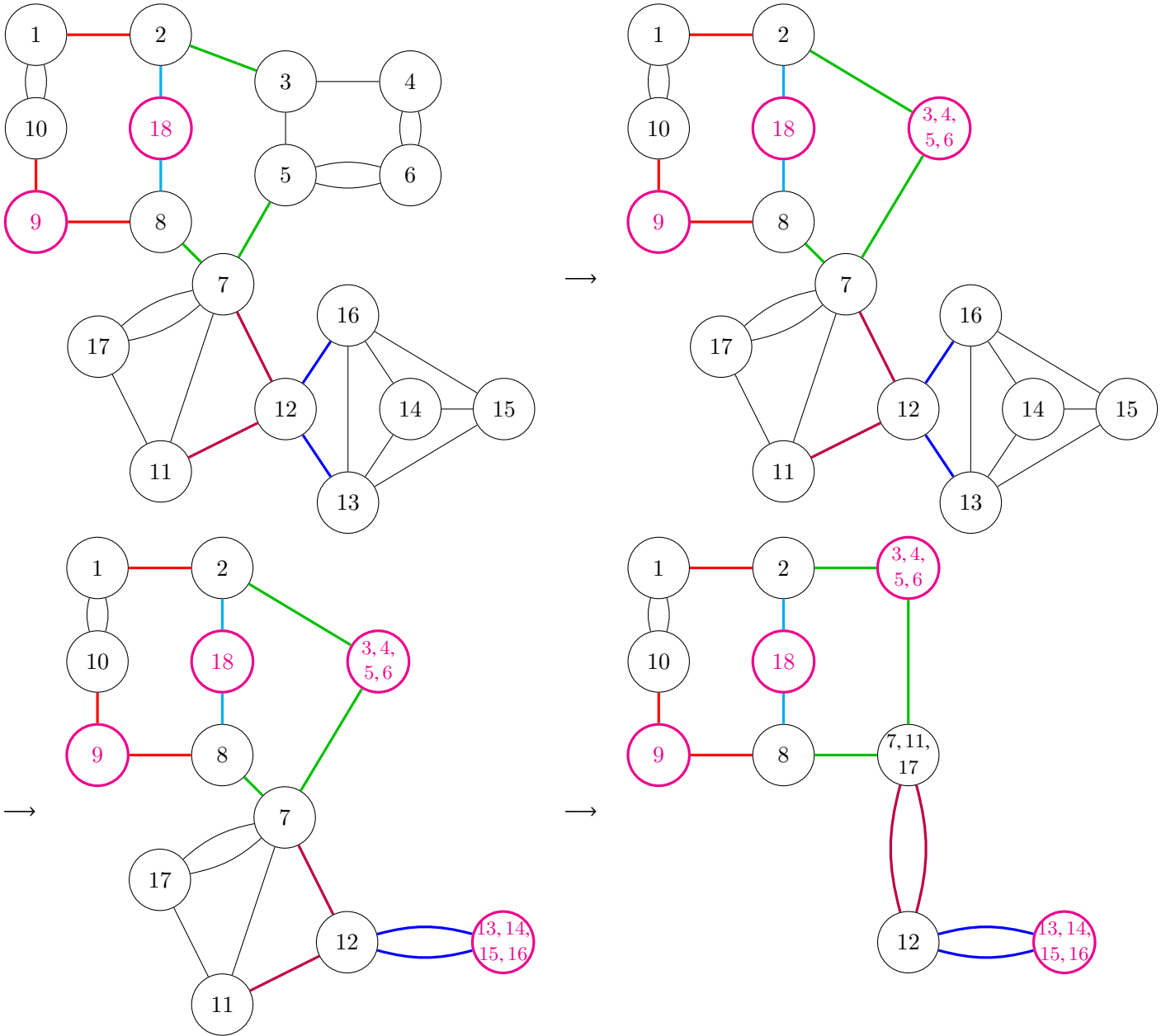


图 15

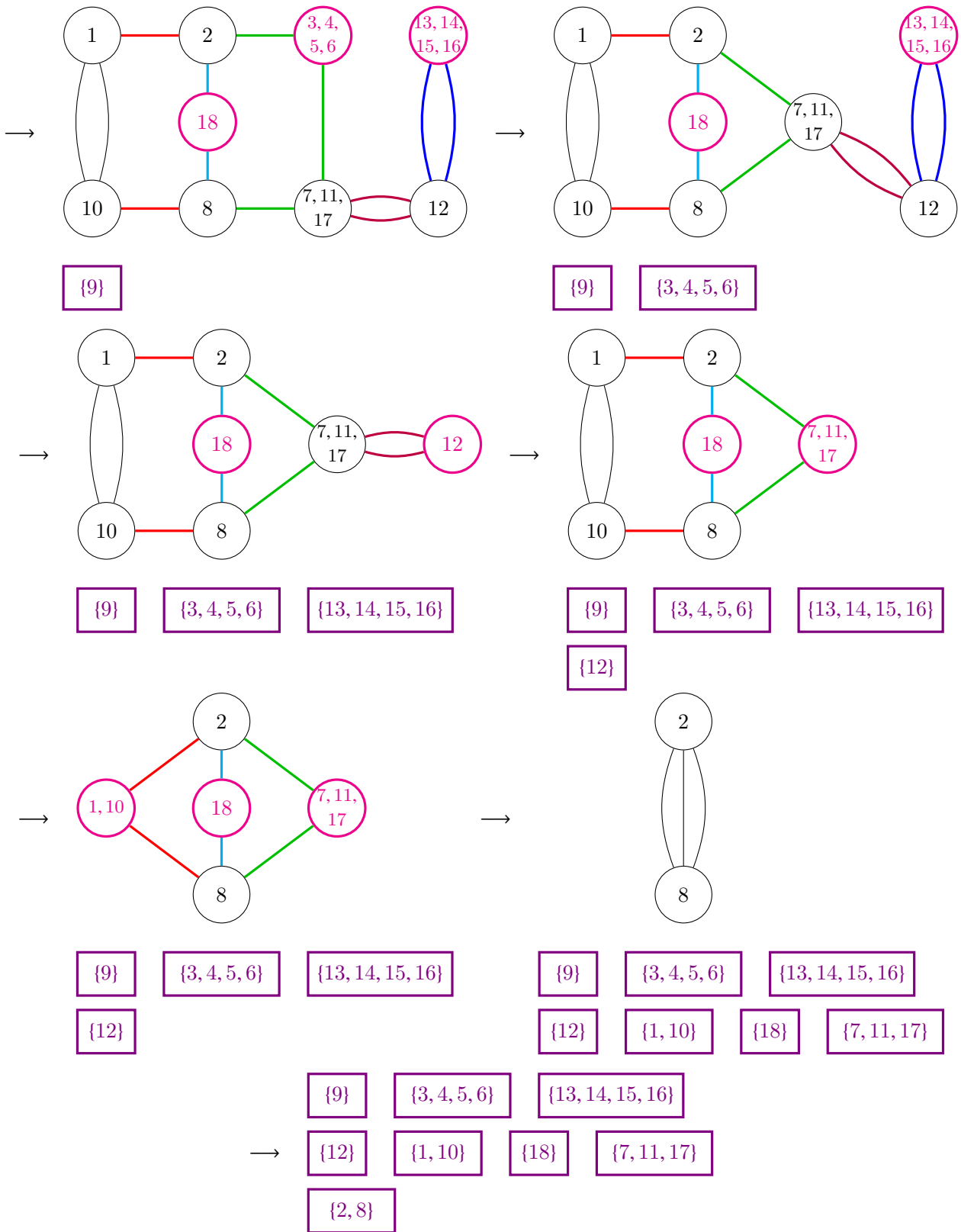


图 15 (续)

4.1.7 时间复杂度

分析上述算法的时间复杂度。

根据算法的流程可知，一旦应用[定理 4.1.3](#)或[定理 4.1.4](#)，图的点数就会减小 1，因此步骤 4 的运行总次数为 $O(|V|)$ 。

考虑步骤 8 中集合的合并，我们可以使用链表来维护每个 v 的 L_v ，因为我们只需要枚举 L_v 的所有顶点和合并两个集合。于是步骤 8 的总时间复杂度为 $O(|V|)$ 。

考虑步骤 11 中枚举顶点，注意到当我们枚举 L_x 的顶点后，点 x 就会被孤立出来，从而 L_x 中的点自成一个 3-边连通分量。这说明， G 中一个顶点至多被枚举到一次，故这部分的总时间复杂度为 $O\left(\sum_{v \in V} d(v)\right) = O(|E|)$ 。

所有步骤中并查集调用的总次数为 $O(|E|)$ ，故此部分时间复杂度为 $O(|E|\alpha(|V|))$ 。

剩下步骤的时间复杂度均为单次操作 $O(1)$ ，故总时间复杂度 $O(|V|)$ 。

综上，整个算法的时间复杂度为 $O(|E|\alpha(|V|))$ 。

事实上，3-边连通算法是有线性做法的[\[12\]](#)。不过限于篇幅等原因，这里就不再介绍了。上面的做法所需的引理较少，解法也比较自然，在实际测试中运行时间完全不逊于[\[12\]](#)中的方法。

4.2 3-点连通²³

4.2.1 引入

在 2-点连通时，我们引入了描述点双连通分量的树结构——块割树和圆方树。在研究 3-点连通时，我们也有与之对应的树结构——SPQR 树。

在引入 SPQR 树的定义之前，仍然可以利用[定理 2.5.3](#)，可得任意两个 3-点连通分量至多交于两个点。这两个点可以有边相连，也可以无边相连。

4.2.2 边的等价性

定义 4.2.1 (割点对). 对于 2-点连通无向简单图 (下略) G ，若某个二元点集 $\{u, v\}$ 是 G 的点割集，则称 (u, v) 是一对割点对 (split pair)。

在处理 2-点连通分量时，我们利用了边的等价性，而在 3-点连通分量时，边仍然具有类似的等价性。

我们取两个顶点 u, v ，考虑顶点 u, v 定义的局部关系 $\rho_{u,v}$ 。

²³ 本小节内容默认假设图是 2-点连通无向简单图。

对于 $e, f \in E$, 称 $(e, f) \in \rho_{u,v}$, 当且仅当存在一条经过 e, f 的路径, 除了路径端点外不能是 u 或 v 。

定理 4.2.1. $\rho_{u,v}$ 是等价关系。

证明. 若 $(e_1, e_2) \in \rho_{u,v}, (e_2, e_3) \in \rho_{u,v}$, 则我们将对应的两段路径接起来, 就得到了一个经过 e_1, e_3 的, 端点非 u, v 的路径了。 \square

考虑 $\rho_{u,v}$ 将 E 划分的等价类 E_1, E_2, \dots, E_n , 我们称如下两种情况为平凡情形:

1. $n = 1$ 。
2. $n = 2$, 且其中一个等价类只包含边 (u, v) 。

定理 4.2.2. 设 $|G| \geq 4$ 。对于 $u, v \in V$, $\rho_{u,v}$ 是平凡等价关系当且仅当 (u, v) 不是割点对。

证明. 若 u, v 是平凡等价关系, 则对于 $\forall a, b \in V \setminus \{u, v\}$, 显然 a, b 有不连接 u, v 的邻边 (否则与平凡矛盾), 于是这两条边必然是 $\rho_{u,v}$ 等价的, 即 a, b 在 $G \setminus \{u, v\}$ 中连通。

若 u, v 不是平凡等价关系, 于是存在两条不等价的边, 由定义知这两条边在 $G \setminus \{u, v\}$ 中不连通, 因此 u, v 是割点对。 \square

对于所有非平凡的等价关系, 我们通过取交得到一个全局关系 ρ_t : 对 $e_1, e_2 \in E$, 称 $(e_1, e_2) \in \rho_t$, 当且仅当对于所有非平凡的 $\rho_{u,v}$, 均有 $(e_1, e_2) \in \rho_{u,v}$ 。

图 16 是一个 ρ_t 对应的等价类划分的例子。

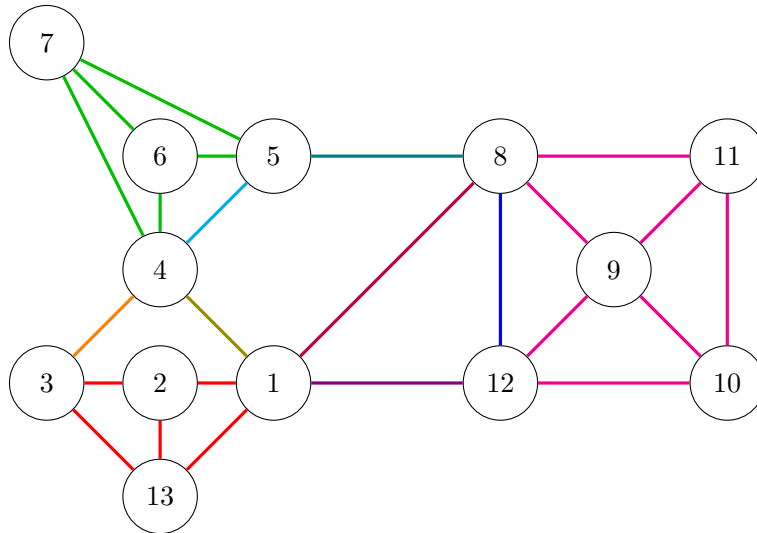


图 16

4.2.3 用割点对划分图

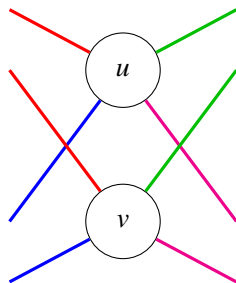


图 17

设 (u, v) 是一个割点对。设关系 $\rho_{u,v}$ 将边集划分为等价类 E_1, E_2, \dots, E_n , 设 A, B 为若干个等价类的并, 满足 $A \cap B = \emptyset, A \cup B = E, \min\{|A|, |B|\} \geq 2$ 。

定义两张新图 G_A, G_B , 其中 G_A 是包含 A 中所有边, 再额外加上边 (u, v) 得到的图, G_B 同理。我们加上的新边被称为**虚边** (virtual edge), 是用来表示划分过程的, 虚边是可以有重边的, 无论原图中是否存在对应的边。

不断执行这些上述划分操作, 直到无法操作为止, 这样我们就得到了一张图的**原始点三连通分量**。

注意到圈图 C_n 有很多划分的方法, 不过最终本质是一个 n 边形的三角剖分, 这是我们不希望看到的, 因此我们需要合并这些圈图。

具体地, 对于划分完毕后的两张小图 A, B , 如果它们具有**共同的虚边**, 那么我们定义 A, B 合并的结果是将它们的点集合并, 边集合并并去掉共同虚边。可以看出, **合并是划分的逆过程**。

我们将所有可合并的圈图合并成若干个大圈, 将所有可合并的偶极图²⁴合成一个大偶极图, 最终得到的结果就称为 G 的**点三连通分量**。

4.2.4 SPQR 树

所有的点三连通分量可以分为四类:

- Q (Trivial): 2 个点一对重边的图, 作为某些边界情况讨论。由于我们假设 G 为简单图, 因此这里我们忽略 Q 情形。
- S (Serial): 至少 3 个点的圈图。圈图中的每一条边可以是原图中的边, 也可以是**虚边**。
- P (Parallel): 至少 3 条边的偶极图。由于简单图的假设, 因此其中至多一条是原图的边, 其余的边均为**虚边**。

²⁴dipole graph, 见 [13]。

- R (Rigid): 3-点连通子图, 其中每一条边可以是原图中的边, 也可以是虚边。

定义 4.2.2 (SPQR 树). SPQR 树 $T = (\mathcal{V}, \mathcal{E})$, 其中 \mathcal{V} 表示所有点三连通分量的集合, 由之前的结论知可以分为四类。

对于 $u, v \in \mathcal{T}$, $(u, v) \in \mathcal{E}$ 当且仅当 u, v 具有共同的虚边, 即它们是可合并的。

定理 4.2.3. 对于 2-点连通无向图 G , 它的 SPQR 树是一棵树。 □

定理 4.2.4. 对于 2-点连通无向图 G , 它的 SPQR 树是唯一确定的。 □

4.2.5 构造

根据定义, 我们可以得到一个 $O(|V||E|^2)$ 的做法——不停寻找原图的割点对, 将原图划分成更小的三连通分量。

事实上, SPQR 树可以在线性时间内构造, 不过限于篇幅这里就不展开了, 有兴趣的读者可以见参考文献 [14] [15]。

4.2.6 应用

SPQR 树有较为广泛的应用, 下面举两个例子:

例. 给定 2-点连通无向简单图 G , 找出 G 的所有割点对。

考虑 G 的 SPQR 树, 注意到 (u, v) 是割点对当且仅当 $\rho_{u,v}$ 不平凡。而不平凡的 $\rho_{u,v}$ 在最终 SPQR 树的表现只有两种:

- 作为 SPQR 树中的一条虚边。
- 作为圈图的虚边被合并。

因此, 我们可以得到: (u, v) 是割点对, 当且仅当 u, v 是 SPQR 树上的虚边, 或 u, v 是 SPQR 树中 S 型顶点 (圈图) 中的不相邻顶点 (因为它们被合并了)。

例. 3-点连通平面图, 又称多面体图²⁵, 是表示凸多面体结构的图, 可以理解为凸多面体的球极投影。

定理 4.2.5 (Steinitz). G 为表示凸多面体结构的图, 当且仅当 G 是 3-点连通平面图。

定理 4.2.6. 若 G 是多面体图, 则当选定无穷平面后, 在拓扑意义下 G 具有唯一的平面嵌入。

这些定理, 和本文的关系不大, 可以见参考文献 [17] [18]。

²⁵polyhedral graph, 见 [16]

5 总结

本文介绍了图的连通性理论，从连通性的定义出发，介绍了 k -连通问题的一般解法，以及当 k 较小时的特殊做法和应用。本文挑选的做法都是在 OI 中较有实际应用的，希望能在信息学竞赛中有所普及。

同时，本文对大家熟悉 2-点/边连通问题作了更本质的讲解，并做了一个自然的推广——3-点/连通问题，且使用适量的插图来辅助理解。希望大家在阅读本文后，能对 2-连通问题有更深入的了解，对 3-连通问题有一定的认知。

不过，限于笔者水平有限，对于 k -点连通分量的问题，目前并没有较为高效实用的做法。希望本文能起到一个抛砖引玉的作用，吸引更多读者来研究图论以及组合数学类的问题。

感谢

感谢中国计算机学会提供学习和交流的平台。
感谢国家集训队高闻远教练的指导。
感谢父母对我的照顾与支持。
感谢符水波老师、应平安老师对我的关心与教导。
感谢潘佳奇、钱易等同学与我交流，给我启发。
感谢孙睿泽、宣毅鸣、翁伟捷同学为本文审稿。

参考文献

- [1] Wikipedia contributors. Max-flow min-cut theorem. *Wikipedia*, https://en.wikipedia.org/wiki/Max-flow_min-cut_theorem.
- [2] Wikipedia contributors. Maximum flow problem. *Wikipedia*, https://en.wikipedia.org/wiki/Maximum_flow_problem#Algorithms.
- [3] Wikipedia contributors. Karger's algorithm. *Wikipedia*, https://en.wikipedia.org/wiki/Karger's_algorithm.
- [4] Wikipedia contributors. Stoer–Wagner algorithm. *Wikipedia*, https://en.wikipedia.org/wiki/Stoer%E2%80%93Wagner_algorithm.
- [5] Shimon Even (1979), *Graph Algorithms*, Computer Science Press.
- [6] Dan Gusfield (1990), *Very Simple Methods for All Pairs Network Flow Analysis*, SIAM J. Computing.

- [7] Wikipedia contributors. Biconnected component. *Wikipedia*, https://en.wikipedia.org/wiki/Biconnected_component#Block-cut_tree.
- [8] Robert Tarjan (1972), *Depth-first search and linear graph algorithms*, SIAM J. Computing.
- [9] Wikipedia contributors. Logical equivalence. *Wikipedia*, https://en.wikipedia.org/wiki/Logical_equivalence.
- [10] Wikipedia contributors. Edge contraction. *Wikipedia*, https://en.wikipedia.org/wiki/Edge_contraction.
- [11] Wikipedia contributors. Cycle basis. *Wikipedia*, https://en.wikipedia.org/wiki/Cycle_basis.
- [12] Yung H. Tsin (2007), *A Simple 3-Edge-Connected Component Algorithm*. Theory of Computing Systems.
- [13] Wikipedia contributors. Dipole graph. *Wikipedia*, https://en.wikipedia.org/wiki/Dipole_graph.
- [14] Carsten Gutwenger, Petra Mutzel (2001), *A linear time implementation of SPQR-trees*, Lecture Notes in Computer Science.
- [15] John Hopcroft, Robert Tarjan (1973), *Dividing a graph into triconnected components*, SIAM J. Computing.
- [16] Wikipedia contributors. Polyhedral graph. *Wikipedia*, https://en.wikipedia.org/wiki/Polyhedral_graph.
- [17] Wikipedia contributors. Steinitz's theorem. *Wikipedia*, https://en.wikipedia.org/wiki/Steinitz's_theorem.
- [18] Saunders Mac Lane (1937), *A structural characterization of planar combinatorial graphs*, Duke Mathematical Journal.

浅谈一类基于概率的约瑟夫问题

杭州第二中学 叶卓睿

摘要

“一类基于概率的约瑟夫问题”是一类有趣的概率期望问题，在此之前，这类问题只零零散散地出现过两三次。

笔者对这类问题进行了深入研究，最终得到了较为系统的成果，本文将通过三道例题展示这类题目的各种技巧和思路，并对这些处理手段进行对比和分析，希望读者以后遇到这类问题时有所可循。

另外，本文也提供了一种很好的出题方向：对一个模型进行深入研究，对相关问题进行对比分析，从而获得新的问题和思路。

1 前言

本文介绍了处理“一类基于概率的约瑟夫问题”这类题目的各种技巧和思路，并对这些处理手段进行对比和分析。在此之前，这类问题只零零散散地出现过两三次。笔者对这类问题进行了深入研究，最终得到了较为系统的成果，这些在文中将会以三道例题的形式进行展示，希望读者以后遇到这类问题时有所可循。

本文第二节将介绍这类问题的定义，第三节将给出这类问题的通用观察和结论，第四、五、六节将通过三个例题对这类问题的解决办法进行较为全面的展示，第七节将对上述三个例题的解法和思路进行比较和分析。

2 定义

给一个 1 到 n 依次连接的环，有个指针从 1 开始移动，每次指针所在位置有 $p(p > 0)$ 的概率消失掉，然后指针向右移动，游戏在所有位置都消失时结束。

在此基础上需要求解一些概率期望问题。

为了方便描述，下文约定 $q = 1 - p$ 。

3 一些观察

引理1: 一个被指针经过 i 次的点, 存活概率为 q^i 。

证明. 转化问题, 认为一个点消失后不会真的从环上消失, 而是打上一个“删除”标记。那么一个点存活当且仅当每次经过都没有打“删除”标记, 故存活概率为 q^i 。□

引理2: 当指针落在位置 c 时, $[1, c-1]$ 中每个点已经消失的概率相同, $[c+1, n]$ 中每个点已经消失的概率也相同。

证明. 注意到此时 $[1, c-1]$ 中每个点经过次数相同, 由引理1, 它们的存活概率相同。 $[c+1, n]$ 也同理。□

4 一个简单的问题

例题1. 迫真大游戏¹

给一个 1 到 n 的环, 有个指针从 1 开始移动, 每次指针所在位置有 p 的概率消失掉, 然后指针向右移动。求每个点是最后一个消失的概率。

$$n \leq 2 \cdot 10^5$$

4.1 分析问题

我们关心的位置未必是起点, 考虑先让指针转几步使得它落在我们关心的位置上。

令 f_n 为环大小为 n 时, 1 号点最后消失的概率。尝试先求出 f_n 。

4.2 F 的求法

枚举第一个分身在第 $i+1$ 轮消失, 计算其概率, 可以得到

$$\begin{aligned} f_n &= \sum_{i=0}^{\infty} pq^i(1-q)^{n-1} \\ &= \sum_{i=0}^{\infty} pq^i \sum_{j=0}^{n-1} (-1)^j (1-p)^{ij} \binom{n-1}{j} \\ &= p \sum_{j=0}^{n-1} (-1)^j \binom{n-1}{j} \sum_{i=0}^{\infty} q^{(j+1)i} \\ &= p \sum_{j=0}^{n-1} (-1)^j \binom{n-1}{j} \frac{1}{1-q^{j+1}} \end{aligned}$$

¹CometOJ Contest 4, Problem E

将组合数用阶乘展开后这是个卷积的形式，可以 FFT 解决。

FFT 解决卷积问题的具体细节可以参考毛啸在 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》，这里不再赘述。

4.3 求解原问题

通过枚举指针移动过程中消失的位置个数，我们得到

$$ans_k = \sum_{i=0}^{k-1} \binom{k-1}{i} p^i q^{k-1-i} f_{n-i}$$

这也可以化为卷积的形式。综上所述，原问题可以通过 2 次 FFT 解决，总时间复杂度为 $O(n \log n)$ 。

5 对例题一进行改编

例题2. Game on a Circle²

给一个 1 到 n 的环，有个指针从 1 开始移动，每次指针所在位置有 p 的概率消失掉，然后指针向右移动。对于每个 i ，求 c 号点是第 i 个消失的概率。

$$n \leq 10^6$$

5.1 生成函数方法

我们试试直接用生成函数推式子。

令 a_i 为 c 号点是第 $i+1$ 个消失的概率，我们希望求出 $A(x) = \sum a_i x^i$ 。则有

$$\begin{aligned} A(x) &= \sum_{t=0}^{\infty} q^t p (q^{t+1} + (1 - q^{t+1})x)^{c-1} (q^t + (1 - q^t)x)^{n-c} \\ &= \sum_{t=0}^{\infty} q^t p (q^{t+1}(1-x) + x)^{c-1} (q^t(1-x) + x)^{n-c} \\ &= p \sum_i \sum_j \binom{c-1}{i} \binom{n-c}{j} \sum_{t=0}^{\infty} q^i (1-x)^{i+j} x^{n-1-i-j} q^{t(1+i+j)} \\ &= p \sum_i \sum_j \binom{c-1}{i} \binom{n-c}{j} q^i (1-x)^{i+j} x^{n-1-i-j} \frac{1}{1 - q^{1+i+j}} \end{aligned}$$

²2020 Multi-University Training Contest 3, Problem K

观察式子，可以记 $f_k = \sum_{i+j=k} \binom{c-1}{i} \binom{n-c}{j} q^i$ ，则有

$$A(x) = p \sum_i \frac{f_i}{1 - q^{i+1}} (1-x)^i x^{n-1-i}$$

由于 c 为常数， f_n 是一个卷积的形式，可以 FFT。现在考虑求解 $A(x)$ ，继续推导可知

$$[x^{n-i+j-1}]A(x) = p \sum_i \sum_j \binom{i}{j} (-1)^j \frac{f_i}{1 - q^{i+1}}$$

这也是卷积的形式，问题即可使用 2 次 FFT 得到解决。

5.2 优化

事实上我们可以做得更快。记 $F(x)$ 为 f_i 对应的普通型生成函数，我们发现 $F(x) = (1 + qx)^{c-1} (1+x)^{n-c}$ 。

通过对生成函数求导，可以得到

$$F'(x) = (c-1)q \frac{F(x)}{1+qx} + (n-c) \frac{F(x)}{1+x}$$

对比系数，得到 f_i 的递推式：

$$f_{i+1} = \frac{((c-1)q + n - c - (q+1)i) f_i + q(n-i) f_{i-1}}{i+1}$$

综上所述，我们只需要 1 次 FFT 就解决了这个问题，时间复杂度 $O(n \log n)$ 。

5.3 其它做法

事实上，使用 4.1 中提到的拆解过程的办法，可以做到 $O(n \log^2 n)$ ，由于复杂度较劣以及篇幅所限，这里不进行展示。

6 更复杂的问题

例题3. Eat Cards, Have Fun³

给一个 1 到 n 的环，第 i 个点上有数字 A_i ，有个指针从 1 开始移动，每次指针所在位置有 p 的概率消失掉然后加入序列 b 的末尾，然后指针向右移动。求排列 b 在所有 $[1, n]$ 的排列中，按字典序排序后的序号期望。

$$n \leq 300$$

³2018 Multi-University Training Contest 4, Problem H

6.1 官方题解做法

6.1.1 初步分析

考虑利用期望的线性性拆贡献。记排列为 P_n ，则这个排列的序号为 $Ans = \sum_i (n - i)! \sum_{j>i} [P_j < P_i] + 1$ 。

贡献还可以通过枚举位置 i ，以及在 i 消失之前 $[1, i - 1]$ 共消失了 a 个， $[i + 1, n]$ 共消失了 b 个， $[1, i - 1]$ 共消失了 c 个小于 P_i 的， $[i + 1, n]$ 共消失了 d 个小于 P_i 的来计算。

记 $dp_{i,a,b}$ 为在 i 消失之前 $[1, i - 1]$ 共消失了 a 个， $[i + 1, n]$ 共消失了 b 个的概率， l_i 为 $[1, i - 1]$ 中小于 P_i 的个数， r_i 为 $[i + 1, n]$ 中小于 P_i 的个数，则有

$$Ans = \sum_i \sum_a \sum_b \sum_c \sum_d (n - a - b - 1)! (A_i - c - d - 1) \binom{l_i}{c} \binom{i - 1 - l_i}{a - c} \binom{r_i}{d} \binom{n - i - r_i}{b - d} dp_{i,a,b} + 1$$

6.1.2 求解 dp 数组

首先考虑如何计算 $dp_{i,a,b}$ 。

$$\begin{aligned} dp_{i,a,b} &= \sum_{t=0}^{\infty} q^t p (q^{t+1})^{i-1-a} (1 - q^{t+1})^a (q^t)^{n-i-b} (1 - q^t)^b \\ &= \sum_{j=0}^a \sum_{k=0}^b \binom{a}{j} \binom{b}{k} (-1)^{j+k} q^{i-1-a+j} \sum_{t=0}^{\infty} q^{t(n+j+k-a-b)} \\ &= \sum_{j=0}^a \sum_{k=0}^b \binom{a}{j} \binom{b}{k} (-1)^{j+k} \frac{q^{i-1-a+j}}{1 - q^{n+j+k-a-b}} \end{aligned}$$

暴力计算时间复杂度为 $O(n^5)$ ，需要优化。

优化计算 定义

$$h_{1,a,b} = \sum_{j=0}^a \sum_{k=0}^b \binom{a}{j} \binom{b}{k} (-1)^{j+k} \frac{q^{a+j}}{1 - q^{n+j+k-a-b}}$$

则有

$$dp_{i,a,b} = pq^{i-1} h_{1,a,b}$$

定义

$$h_{2,a,b} = \sum_k \binom{b}{k} (-1)^k \frac{1}{1 - q^{n-a-b+k}}$$

则有

$$h_{1,a,b} = \sum_j \binom{a}{j} (-1)^j q^{j-a} h_{2,a-j,b}$$

那么，我们可以分别计算出 $h_{2,a,b}$, $h_{1,a,b}$, $dp_{i,a,b}$ ，这部分时间复杂度优化至了 $O(n^3)$ 。

6.1.3 原问题的求解

回到答案式子

$$Ans = \sum_i \sum_a \sum_b \sum_c \sum_d (n-a-b-1)! (A_i - c - d - 1) \binom{l_i}{c} \binom{i-1-l_i}{a-c} \binom{r_i}{d} \binom{n-i-r_i}{b-d} dp_{i,a,b} + 1$$

暴力计算仍然是 $O(n^5)$ 的。

优化计算 接下来我们定义一些辅助函数加速计算：

$$f_{i,0,a} = \sum_c \binom{l_i}{c} \binom{i-1-l_i}{a-c}$$

$$f_{i,1,a} = \sum_c \binom{l_i}{c} \binom{i-1-l_i}{a-c} c$$

$$g_{i,0,b} = \sum_d \binom{r_i}{d} \binom{n-i-r_i}{b-d}$$

$$g_{i,1,b} = \sum_d \binom{r_i}{d} \binom{n-i-r_i}{b-d} d$$

然后将 $A_i - c - d - 1$ 拆为 $(A_i - 1) - c - d$ 三部分分别算贡献，可以得到：

$$Ans = \sum_i \sum_a \sum_b ((A_i - 1)f_{i,0,a}g_{i,0,b} - f_{i,1,a}g_{i,0,b} - f_{i,0,a}g_{i,1,b})(n-a-b-1)! dp_{i,a,b} + 1$$

时间复杂度 $O(n^3)$ 。

6.2 新的思考角度

6.2.1 初步思路

根据期望的线性性，贡献可以拆成 $\sum_i \sum_j \sum_t [A_i > A_j] Pr(i \text{ 先于 } j \wedge i \text{ 在时刻 } t \text{ 消失}) (n-t)!$ 。

由引理2，我们发现固定 i, t 后，贡献只需与 j, i 的相对大小有关。

那么思路就很明显了，分 $i > j$ 和 $i < j$ 两类情况进行讨论。

6.2.2 动态规划算法

接下来我们先讨论 $i > j$ 的情况，可以设计 dp ： $f_{n,i}$ 表示长度为 n 的环， i 先于 $i-1$ 消失的所有情况下 $(n-t)!$ 的期望。

转移如下：

$$f_{n,i} = \begin{cases} qf_{n,n} + p(n-1)! & i = 1 \\ qf_{n,i-1} & i = 2 \\ qf_{n,i-1} + pf_{n-1,i-1} & i > 2 \end{cases}$$

按 n 从小到大进行求解，对于每个 n 直接高斯消元可以 $O(n^3)$ ，总复杂度 $O(n^4)$ 。

事实上对于每个 n ，dp 转移是环上高斯消元的形式。而环上高斯消元是可以做到 $O(n)$ 的，具体做法是：设 $x = f_{n,n}$ ，那么对于 $i \in [1, n]$ 可以分别把 $f_{n,i}$ 表示成 $k_i x + b_i$ 的形式，从而得到一个一元一次方程，解出 x 后即可计算出 $f_{n,i}$ 。

另一种情况 $i < j$ 类似，篇幅所限这里不再赘述。

这样时间复杂度为 $O(n^2)$ ，实现难度较低。

6.3 进一步优化

dp 转移式子最麻烦的一点在于有环，我们尝试先求解第一列。

6.3.1 dp 数组第一列的求解

记 $f_n = dp_{n,1}$ 。枚举有多少个是在 1 之后消失的：

$$\begin{aligned} f_n &= \sum_{i=0}^{n-2} \binom{n-2}{i} \sum_{t=0}^{\infty} q^t p q^t (q^t)^i (1-q^t)^{n-2-i} (i+1)! \\ &= \sum_{i=0}^{n-2} \binom{n-2}{i} \sum_{j=0}^{n-2-i} \binom{n-2-i}{j} (-1)^j p (i+1)! \sum_{t=0}^{\infty} q^{t(2+i+j)} \\ &= p \sum_{i=0}^{n-2} \binom{n-2}{i} \sum_{j=0}^{n-2-i} \binom{n-2-i}{j} (-1)^j (i+1)! \frac{1}{1-q^{2+i+j}} \\ &= p \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} \binom{n-2}{i+j} \binom{i+j}{i} (-1)^j (i+1)! \frac{1}{1-q^{2+i+j}} \end{aligned}$$

优化计算 为加速计算，我们令

$$g_k = \sum_{i+j=k} \binom{i+j}{i} (i+1)! (-1)^j$$

$$= \sum_{i+j=k} \frac{(i+j)! (i+1) (-1)^j}{j!}$$

这是一个卷积的形式，可以 FFT 解决。

于是有

$$f_n = p \sum_k \frac{1}{1-q^{2+k}} \binom{n-2}{k} g_k$$

$$= p \sum_k \left(\frac{1}{1-q^{2+k}} g_k \right) \binom{n-2}{k}$$

将组合数用阶乘展开后，这也是一个卷积的形式，可以 FFT 解决。

因此，我们可以 $O(n \log n)$ 地求出 $dp_{i,1}$ 。

6.3.2 dp 数组第 n 行的求解

回顾 dp 转移方程

$$f_{n,i} = \begin{cases} qf_{n,n} + p(n-1)! & i = 1 \\ qf_{n,i-1} & i = 2 \\ qf_{n,i-1} + pf_{n-1,i-1} & i > 2 \end{cases}$$

已知 $dp_{i,1}$ ，可以 $O(n)$ 地计算出 $dp_{i,2}$ ，这样做是为了解决 $i = 2$ 时转移比较特殊的问题。

记 $g_i = dp_{i,2}, h_i = dp_{n,i+2}$ 。计算每个 g_j 对 h_i 的贡献，观察 dp 转移式子可知：

$$h_i = \sum_j g_j \binom{i}{n-j} p^{n-j} q^{i+j-n}$$

将组合数用阶乘展开后，这也是一个卷积的形式，可以 FFT 解出 $f_{n,i}$ 。

6.3.3 回到原问题

考虑求解原问题， $i > j$ 的情况贡献为

$$Ans = \sum_{i>j} [A_i > A_j] f_{n,i}$$

，可以使用树状数组统计 $Ans = \sum_{i>j} [A_i > A_j]$ 。

$i < j$ 的部分使用类似的做法即可做到总时间复杂度 $O(n \log n)$ 。

引理3: 长度为 n 的环, 1 先于 2 消失的所有情况下 $(n-i)!$ 的期望与 1 先于 n 相同。

证明. 枚举 1 是在第 $i+1$ 次经过时消失的, 那么 $[2, n]$ 的每个点经过次数相同, 由引理1可知它们消失的概率相同, 因此在这个问题中是等价的。□

由引理3, $i < j$ 的情况下 dp 数组第一列和 $i > j$ 完全相同, 由此可以减小一定的代码量和常数。

7 对比与分析

例题一只需要仔细分析题目, 发现过程可以拆成独立的两步, 逐个解决即可, 算是这类问题中较为基础的情形。

在引理1的帮助下, 只使用数学推导手段就可以在一些问题(如例题二)上取得不错的效果, 其优势是思维难度不高, 不过缺点是推导、计算较为繁琐, 在一些情况下可能无法进行优化。

例题三很好地体现了代数推导的局限性: 不仅繁琐而且无法做到很优的复杂度。笔者通过对问题模型进行观察, 得到一个很强的性质, 从而得到一种 $O(n^2)$ 的简洁的 dp 做法。借鉴例题一的思想, 指针初始位于 1 的情况是容易解决的, 此后 dp 的转移无环, 因此可以使用常见的生成函数技巧进行优化, 从而做到 $O(n \log n)$ 的优秀复杂度, 这相对于官方题解给出的 $O(n^3)$ 是一个巨大的飞跃。

8 研究思路与成果

笔者最初在做 2018 年杭电多校联合训练第 4 场 H 题时思考出 $O(n^2)$ 的做法, 比官方题解的推式子做法更优秀。这启发我对这一问题模型进行研究。之后我发现这一模型存在一些基本结论和观察, 而且不同问题在解法上也有一定的联系。例如 CometOJ 第 4 场的 E 题就是先将指针所在位置移动到 1, 从而拆解问题, 事实上笔者也正是受这个思路启发, 才思考出了例题三最后一步的优化。在研究过程中, 笔者对例题一进行修改并进行了深入的思考, 最终命制了例题二, 此题虽和例题一题意比较相似, 做法却大相径庭, 最后几乎只需要一些推导的技巧就可以解决, 这说明代数推导在一些情况下也有用武之地, 选手在做这类题目时需要灵活选择解法, 不能思维定式化。

顺带一提, 例题二作为 2020 年杭电多校联合训练第 3 场 K 题使用, 场上共有 9 支队伍通过, 可见此题难度适中、区分度良好, 同时也说明选手对这类模型还不够熟悉。

9 总结

本文通过三个例题展示了解决“一类基于概率的约瑟夫问题”的各种思路 and 技巧，对这类问题进行了详细的分析研究，最终都得到了 $O(n \log n)$ 的优秀解法，希望读者遇到这类问题时有迹可循。

另外，对一个模型进行深入研究，对相关问题进行对比分析也提供了一种很好的出题方向，例如本文例题二就是这样命制的。

最后，希望本文起到抛砖引玉的作用，吸引更多读者来研究这一类问题，或是将本文提及的思想应用到更多其它问题上。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢杭州第二中学李建老师的关心和指导。

感谢国家集训队教练高闻远的指导。

感谢吴越学长、周欣同学对本文的帮助。

感谢父母对我的关心和照顾。

感谢所有帮助过我的人。

参考文献

- [1] Ronald L. Graham, Donald E. Knuth, Oren Patashnik. Concrete Mathematics .
- [2] 金策,《生成函数的运算与组合计数问题》, 2015年信息学奥林匹克中国国家队候选队员论文
- [3] 毛啸,《再探快速傅里叶变换》, 2016年信息学奥林匹克中国国家队候选队员论文

浅谈多项式牛顿迭代与拉格朗日反演在 OI 中的应用

华东师范大学第二附属中学 左骏驰

摘要

生成函数的计数问题是 OI 中的一类重要的问题。近年来,除了多项式乘法、多项式求逆、多项式求 \exp 、多项式求 \ln 等传统方法,牛顿迭代、拉格朗日反演等新的技术层出不穷。

因此,本文将围绕着牛顿迭代与拉格朗日反演这两个主题,浅析它们在 OI 中的应用。本文将先介绍牛顿迭代、拉格朗日反演的基本知识,然后介绍它们在 OI 题目中的应用,并将这些用法进行归纳、分类。接着本文介绍了牛顿迭代法的高级应用。由于牛顿迭代和拉格朗日反演都和多项式复合、复合逆息息相关,本文还简要介绍了求多项式复合、复合逆的方法。

1 前置知识

对于任意一个域 F 和其上的任意一个 $n+1$ 项的序列 $a_0, a_1, \dots, a_n \in F$, 我们记这个序列的普通型生成函数 (也称 **OGF**) 为:

$$F(x) = \sum_{i=0}^n a_i x^i \quad (1)$$

其指数型生成函数 (也称 **EGF**) 为:

$$G(x) = \sum_{i=0}^n \frac{a_i}{i!} x^i \quad (2)$$

而对于无穷序列 a_0, a_1, \dots 仍然可以定义其普通型和指数型生成函数:

$$F(x) = \sum_{i=0}^{+\infty} a_i x^i, \quad (3)$$

$$G(x) = \sum_{i=0}^{+\infty} \frac{a_i}{i!} x^i \quad (4)$$

对任意一个形如如下形式的级数:

$$F(x) = \sum_{i=-\infty}^{+\infty} a_i x^i \quad (5)$$

满足 a_{-1}, a_{-2}, \dots 这个数列在足够多项之后为 0, 则称 $F(x)$ 为形式 Laurent 级数。我们可以约定 $[x^n]F(x)$ 表示 a_n , 类似地, 我们也可以在上述定义上加减法、乘法、求导等运算。但在之后的论述中, “函数” 均指的是“形式幂级数”, 也就是不存在 x^{-1}, x^{-2}, \dots 项的形式 Laurent 级数。

对于生成函数 $F(x), G(x)$ (下面均约定它没有 x^{-1}, x^{-2}, \dots 这些项), 若 $F(0) \neq 0$ $F(x)G(x) = 1$, 则我们称 $G(x)$ 为 $F(x)$ 的逆, 记为: $G(x) = \frac{1}{F(x)}$ 。

对于生成函数 $F(x), G(x)$, 若 $[x^0]F(0) = 0, [x^1]F(0) \neq 0, F(G(x)) = G(F(x)) = x$, 则我们称 $G(x)$ 为 $F(x)$ 的复合逆。记为 $G(x) = F^{-1}(x)$ 。稍后将说明复合逆的存在性和唯一性。

对于两个有限次数的多项式 $F_1(x), F_2(x)$, 和非零多项式 $G(x)$, 若 $\frac{F_1(x)-F_2(x)}{G(x)}$ 也是一个多项式, 则称 $F_1(x) \equiv F_2(x) \pmod{G(x)}$ 。特别的, 若两个多项式模 x^n 同余, 则它们的 x^0, x^1, \dots, x^{n-1} 项系数相同。

在之后的介绍中, 我们约定 F 是一类性质比较好的域, 使得次数为 n 次的多项式乘法、求逆、求 \exp 、求 \ln 都可以通过 DFT 操作在 $O(n \log n)$ 的复杂度内完成。例如: $F = F_{998244353}$ 时, 因为 $998244353 = 7 \cdot 17 \cdot 2^{23} + 1$, F 存在 2^{23} 次单位根, 且加减乘运算均可以 $O(1)$ 实现, 不存在精度问题, 在大部分情况下可以支持除法。

2 多项式牛顿迭代

给定一个次数不大于 n 的多项式 $F(x)$, 我们要解出满足 $F(G(x)) = 0$ 的生成函数的前 n 项。

一种常见的方法是使用多项式牛顿迭代去倍增求解。

假设我们已经得到了 $F(A(x)) \equiv 0 \pmod{x^m}$, 试图找到 $B(x)$ 使得 $F(B(x)) \equiv 0 \pmod{x^{2m}}$ 。那么我们由泰勒展开的公式:

$$F(B(x)) = F(A(x)) + F'(A(x))(B(x) - A(x)) + \frac{F''(A(x))}{2!}(B(x) - A(x))^2 + \dots \quad (6)$$

考虑在 $\text{mod } x^{2m}$ 意义下, 我们有 $F(B(x)) \equiv F(A(x)) + F'(A(x))(B(x) - A(x)) \pmod{x^{2m}}$ 。

从而只需令 $B(x) = A(x) - \frac{F(A(x))}{F'(A(x))}$ 即可!

如果我们知道 $G(x)$ 在 $\text{mod } x$ 下的一个解, 那么通过如下迭代, 可以得到 $G(x) \pmod{x^2}$, $\text{mod } x^4$, $\text{mod } x^8, \dots$ 的结果。因此做 $O(\log n)$ 次迭代即可得到 $G(x) \pmod{x^{n+1}}$ 的结果!

如果 $F(G(x))$ 容易计算, 例如 $F(x)$ 是个低次多项式, 或者容易用 \ln, \exp 表示, 那么该算法的复杂度往往可以达到 $O(n \log n)$ 。

在某些情况下, F 不必局限于一个多项式。只要能够计算出 $G(x) \pmod{x}$ 的结果, 每次迭代的操作是有意义的, 那么 F 可以推广为形如 $F(G(x), x)$ 的二元函数。例如 $F(x) = (x+1)G(x) + 1, F(x) = (x+1)G(x)^2 + (x^2 + 2x + 2)G(x) + 1$ 等。而这时对 $F(G(x), x)$ 应该看作是对 $G(x)$ 求偏导!

此处我们先介绍基本的两种牛顿迭代，之后我们将介绍用牛顿迭代法解多项式微分方程的例子。

3 多项式的复合逆

对于 $[x^0]F(x) = 0, [x^1]F(x) \neq 0$ 的生成函数 $F(x)$ ，称满足 $G(F(x)) = x$ 的函数 G 为它的左逆元，称满足 $F(H(x)) = x$ 的函数 H 为它的右逆元。

设 $G(x) = a_0 + a_1x + a_2x^2 + \dots$ ，我们递推构造 a_0, a_1, a_2, \dots 。令 $a_0 = 0, a_1 = \frac{1}{[x^1]F(x)}$ 。比较 $G(F(x))$ 的 x^k 系数 ($k \geq 2$)，那么我们有：

$$[x^k]a_0 + [x^k]a_1F(x) + [x^k]a_2F^2(x) + \dots + [x^k]a_{k-1}F^{k-1}(x) + a_k([x^1]F(x))^k = 0 \quad (7)$$

这是因为 $F^{k+1}(x), F^{k+2}(x)$ 均不含 x^k 项系数，且 $F^k(x)$ 的系数就是 $([x^1]F(x))^k$ 。

因此，我们知道 $F(x)$ 的左逆存在且唯一。

另一方面，设 $H(x) = b_0 + b_1x + b_2x^2 + \dots, b_0 = 1, b_1 = \frac{1}{[x^1]F(x)}$ 。同样比较 $F(G(x))$ 的 x^k 系数，我们有：

$$[x^k]F(b_0 + b_1x + \dots + b_{k-1}x^{k-1}) + ([x^1]F(x))^k b_k = 0 \quad (8)$$

因此， $F(x)$ 的右逆存在且唯一。

且注意到 $G(x) = G(F(H(x))) = H(x)$ ，故 $F(x)$ 的左逆与右逆是相等的，我们统称为复合逆。在此我们证明了 $F(x)$ 的复合逆存在且唯一。

4 拉格朗日反演

拉格朗日反演公式: 若 $F(x)$ 是 $G(x)$ 的复合逆，则

$$[x^n]G(x) = \frac{1}{n}[x^{-1}] \left(\frac{1}{F(x)} \right)^n \quad (9)$$

推广形式为：

$$[x^n]H(G(x)) = \frac{1}{n}[x^{-1}]H'(x) \left(\frac{1}{F(x)} \right)^n \quad (10)$$

其中 $n \geq 1$ 。

下面给出证明：

设 $H(G(x)) = \sum_{i=0}^{+\infty} a_i x^i$ 。

那么我们有：

$$\sum_{i=0}^{+\infty} a_i F(x)^i = H(x) \quad (11)$$

对 (11) 两边求导得：

$$\sum_{i=1}^{+\infty} i a_i F(x)^{i-1} = H'(x) \tag{12}$$

(12) 两边除以 $F^n(x)$ ，那么我们发现， $\frac{1}{F^2(x)}, \frac{1}{F^3(x)}, \dots$ ，均不对 x^{-1} 项系数产生贡献，且 $F(x), F^2(x), \dots$ 也不对 x^{-1} 项系数产生贡献，故提取 x^{-1} 项系数，我们有：

$$n[x^n]H(G(x)) = [x^{-1}]H'(x)F(x) \tag{13}$$

这样即可证明出我们的命题。

5 组合计数问题分析

5.1 用拉格朗日反演求关于复合逆的式子的某一项

在这一部分，我们往往会把问题转换成：已知一个多项式 $F(x)$ 的复合逆 $G(x)$ ，想要求 $H(G(x))$ 中的某一项的值。我们常用的方法是使用拉格朗日反演，将问题转换为求 $F(x)^n$ 的问题！

例 1 推导含有 n 个点的生成树个数。

解： 这里我们给出一种与传统组合方法不同的方法。

设 $T(x)$ 表示含有 n 个顶点的有标号的有根树个数组成的 EGF。

那么我们删去根后，得到若干个有根树的划分，根据多项式 \exp 的组合定义，我们有：

$$T(x) = xe^{T(x)}$$

。

令 $F(x) = \frac{x}{e^x}$ ，从而 $T(x)$ 是 $F(x)$ 的复合逆。

则我们由拉格朗日反演

$$[x^n]T(x) \tag{14}$$

$$= [x^{-1}] \frac{1}{F^n(x)} \tag{15}$$

$$= [x^{-1}] \left(\frac{e^{nx}}{x^n} \right) = n^{n-1} \tag{16}$$

再将 n^{n-1} 除以 n ，即可得到 n 个顶点带标号生成树的个数为 n^{n-2}

例 2 给定 $\{1, 2, \dots, s\}$ 的一个子集 D 。对于一棵带有正整数点权的有根多叉树，如果它满足这样的性质，我们就称它为好的：点权为 1 的结点是叶子结点；对于任一点权大于 1 的结点 u ， u 的孩子数目 $\deg[u]$ 属于集合 D ，且 u 的点权等于这些孩子结点的点权之和。给出

一个整数 $s (1 \leq s \leq 10^5)$, 你需要求出根节点权值为 s 的好的多叉树个数, 这里一个点的儿子是有序的。我们只需要知道答案关于素数 $950009857 = 453 \cdot 2^{21} + 1$ 取模后的值。

解: 令权值为 n 的好的有根树个数的 OGF 为 $T(x)$, 那么我们会发现要么有根树要么恰好有 1 个顶点, 要么可以表示为若干棵子树序列。

因此, 我们有:

$$T(x) = x + \sum_{d \in D} T(x)^d \quad (17)$$

令 $F(x) = 1 - \sum_{d \in D} x^d$, 同样的, 我们会发现 $T(x)$ 是 $F(x)$ 的复合逆。

那么 $[x^s]T(x) = [x^{-1}] \frac{1}{F(x)^s}$ 。运用一次多项式 \ln 和多项式 \exp 即可得出原题的答案, 其时间复杂度为 $O(s \log s)$ 。

例 3 求出 n 阶点双、边双连通图的个数, 其中顶点有标号, 且图不含重边和自环。

解: 设 $C(x)$ 表示 n 阶连通无向图的个数乘以 n 的 EGF。 $A(x)$ 为 $n+1$ 阶的点双连通图个数的 EGF, $B(x)$ 为 n 阶边双连通图个数 EGF。这里 $C(x)$ 可以用多项式 \ln 的方法在 $O(n \log n)$ 的时间复杂度内求出。

先求 $A(x)$ 。我们用 $A(x)$ 去表示 $C(x)$ 。 $C(x)$ 的每一项的意义相当于是含有一个代表顶点的连通图, 可以类比有根树。

取这个代表顶点 u 并删去, 它和它所在的边, 设它构成若干个连通分量 $C_1, C_2, C_3, \dots, C_m$ 。

其中 $C_i \cup U$ 对应了一个 u 在原图中的点双连通分量 D_i 。再删去 D_i 内部的边之后, 对应了恰好 D_i 个连通分量。则 C_i 的选法除以 $|C_i|!$ 为: $[x^{|D_i|}]A(x) \cdot [x^{|C_i|}]C(x)^{|D_i|}$ 。

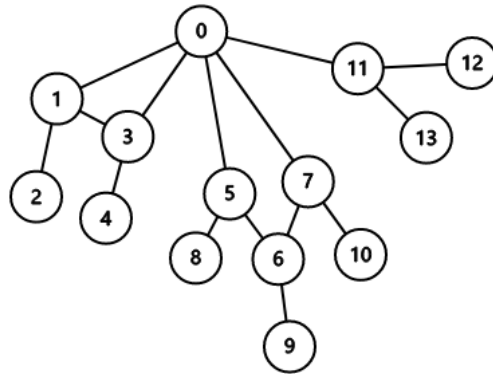


图 1:

如图所示，这里 0 视作代表顶点，三个点双连通分量的大小为 3, 4, 2。且删去一个点双连通分量内部的连边之后，会产生这个该点双连通分量大小 - 1 的不含 0 的连通块。

结合多项式 \exp 的组合意义，我们有：

$$C(x) = xe^{A(C(x))} \tag{18}$$

设 $H(x) = \ln \frac{C(x)}{x}$ ，则我们有：

$$A(x) = H(C^{-1}(x)) \tag{19}$$

对 (19) 式运用 (10) 式所述的扩展拉格朗日反演即可！

再求 $B(x)$ 。同样用 $B(x)$ 表示 $C(x)$ 。我们把 n 阶连通图的代表顶点 u 所在的极大边双连通分量 S 内部的边删去，这样会把图分割为若干个连通块，其中每个连通块与 S 的交集大小为 1。我们如果选取每个连通块的代表顶点为与 S 相交的点，那么将这个代表顶点的任何一条边删去后，图不连通。故这个连通块的取法的 EGF 应该为： $(xe^{C(x)})$ 。

下图展示了一种典型的情形。

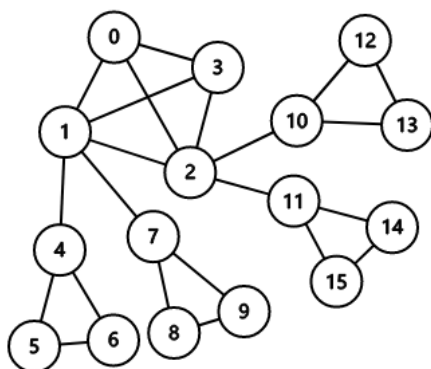


图 2:

与图 1 不同的是，代表顶点 0 恰好在一个边双内，而不能在多个边双。
 且由于边 $(1, 4), (1, 7), (2, 11), (2, 10)$ 都是桥，故 4, 5, 6 这三个点与 7, 8, 9 之间不能连边。
 在统计 1, 4, 5, 6, 7, 8, 9 这 7 个点的连接情况时，不能当成连通图的个数来计算，而是看成删去 1 后，代表顶点为 4 的一个连通块和代表顶点为 7 的一个连通块。

故我们按 u 所在的边双连通分量大小分类，可以得到：

$$C(x) = \sum_{i=1}^{+\infty} ([x^i]A(x)) (xe^{C(x)})^i \tag{20}$$

也就是说：

$$C(x) = B(xe^{C(x)}) \tag{21}$$

这里设 $G(x)$ 为 $xe^{C(x)}$ 的复合逆，就只需套用推广形式的拉格朗日反演来计算 $[x^n]C(G(x))$ 即可！

故我们均可以在 $O(n \log n)$ 的时间内计算点数为 n 的点双连通图、边双连通图的个数（最后答案都要除以 n ）。

例 4 (LOJ 6363)

解： 给定 $\{1, 2, \dots, n\}$ 的子集 S ，我们要求大小为 n 的所有极大点双连通分量都在 S 中的无向连通图的个数。 ($n, \sum_{x \in S} x \leq 100000$)

我们用例 3 的结果，对任意 $x \in S$ ，可以求出点数为 x 的点双连通图的个数 e_x 。

设 $E(x) = \sum_{i \in S} \frac{e_i}{i!} x^i$ 。设含有代表顶点的极大点双连通分量个数都在 S 中的图的 EGF 为 $F(x)$ 。

那么用类似例3的推导，我们有：

$$F(x) = xe^{E(F(x))} \tag{22}$$

则 $F(x)$ 为 $\frac{x}{e^{E(x)}}$ 的复合逆，使用简单形式的拉格朗日反演，即可重新推出满足条件的连通图的个数！

时间复杂度为 $O((n + \sum_{x \in S} x) \log n)$ 。

5.2 用复合逆求解关于多项式幂次的式子

在最近出现的很多拉格朗日反演的问题里面，很多都可以抽象地转换为：对任意 $i = 0, 1, 2, \dots, n$ ，求出

$$[x^k]F(x)G(x)^i \tag{23}$$

这里 $F(x), G(x)$ 都是次数为常数或者容易用 \exp, \ln 等基本函数表示的生成函数，但 G 可能会存在 $x^{-1}, x^{-2}, \dots, x^{-n}$ 项， $k = \Theta(n)$ 。

对于这一类问题，我们考虑引入一个新的变元 u 。则答案序列可以表示为：

$$F(x) + uF(x)G(x) + u^2F(x)G(x)^2 + \dots = \frac{F(x)}{1 - uG(x)} \tag{24}$$

现在考虑用拉格朗日反演提取 (23) 式中的 x^k 系数。先假设 $G(x)$ 存在复合逆 $H(x)$ ， $P(x) = F(H(x))$ 那么我们会得到：

$$[x^k] \frac{F(x)}{1 - uG(x)} = [x^k] \frac{P(G(x))}{1 - uG(x)} = \frac{1}{k} [x^{-1}] \frac{P'(x)(1 - ux) + uP(x)}{(1 - ux)^2} \left(\frac{1}{H(x)} \right)^k \tag{25}$$

因此，我们只需计算 $G(x)$ 的复合逆，以及 $F(H(x))$ ，就可以在 $O(n \log n)$ 的时间内求出结果。当 $F(x) = 1$ 时，这个问题可以在 $O(n \log n)$ 的代价内与复合逆相互转化。

另外，如果 $G(x)$ 不存在复合逆，则需要对 $G(x)$ 做一些处理。

若 $G(x)$ 的常数项不为 0，设 $G(x) = c + Q(x)$ 。则我们先求出 $[x^k]F(x)Q(x)^i$ ，然后注意到：

$$[x^k]F(x)(c + Q(x))^i = [x^k] \sum_{j=0}^i \frac{i!}{j!(i-j)!} c^j F(x) Q(x)^{i-j} \tag{26}$$

因此，只需要用 $O(n \log n)$ DFT 求一次卷积即可！

下面再处理 $G(x)$ 的最低次项不为 x^1 的问题。设 $G(x)$ 的最低次项为 $g_t x^t$ 。

那么对 $\frac{G(x)}{g_t x^t}$ 开 t 次方，可以将 $G(x)$ 表示为 $q_t Q(x)^t$ 。

故我们把问题转化为了求 $[x^k](F(x)Q(x)^{it})$ 。注意到我们只关心 $it \leq k$ 的那些项，因此用 $Q(x)$ 代替前述的 $G(x)$ 作讨论，将 n 与 k 取 \min ，即可做到类似的复杂度。

只要能够方便地求出复合逆和多项式复合,就可以在 $O(n \log n)$ 的时间内解决这个问题。而很多题目里面 $F(x), G(x)$ 都是可以用 \exp, \ln 以及幂次来表示的函数。

例 5, ZJOI2020 抽卡

当期卡池中共有 m 张不同的卡,每次抽卡, Bob 都可以等概率随机获得一张卡池中的卡。如果 Bob 抽到了一张他已经拥有的卡,那么什么事都不会发生,等于 Bob 浪费了这次抽卡机会。

Bob 是个谨慎的人,他想知道,如果他不停地抽卡直到抽到编号连续的 k 张卡时停止抽卡,期望需要抽多少轮。

输入这 m 张不同的卡的编号 $a_1, a_2, \dots, a_m \leq 2m$ 。

数据满足 $1 \leq m \leq 200000, 2 \leq k \leq m$ 。输出答案模 998244353 的结果。

解: 首先,对于一个选了 i 张卡牌,且没有 k 张卡牌编号连续的状态,容易计算出它出现的概率和出现它所用的期望轮数。因此我们只需对每个 i 求出含有 i 张卡牌,没有 k 张卡牌连续的编号总数。

事实上,我们只把 a_1, a_2, \dots, a_m 划分成若干连续段之后,就只需要求每个连续段所对应的答案,再用一次分治多项式乘法将它们乘起来即可。

对一个长度为 n 的连续段 $1, 2, \dots, n$ 。我们补充添加两张必定不选的卡牌 $0, n+1$,然后对与每张不选的卡牌(除了 $n+1$),记录它后面那张不选的卡牌到它的距离。题目条件等价于这些距离之和为 $n+1$,且每个距离都 $\leq k$ 。

因此,我们只需要对每个 i , 求出:

$$[x^{n+1}] \left(\frac{x - x^{k+1}}{1 - x} \right)^i \quad (27)$$

这就转为了前述的问题了!这里我们发现 $\frac{x - x^{k+1}}{1 - x}$ 是容易用牛顿迭代求出复合逆的一个多项式,因此可以在 $O(m \log m)$ 的时间内算出 (27) 式。而分治多项式乘法需要 $O(m \log^2 m)$ 的时间,故总复杂度为 $O(m \log^2 m)$ 。

然而,有些时候所求的式子需要经过变形,才能转换成形如 (23) 式的问题。

例 6, Codeforces Round #641 F2

定义一个长度为 n 的序列 p_1, p_2, \dots, p_n 是好的,当且仅当 p_1, \dots, p_n 都是 1 到 n 的整数,且对任意 $k > 1$,存在 $1 \leq i < j \leq n$ 使得 $p_i = k - 1, p_j = k$ 。

要求对于每个 $k = 1, 2, \dots, n$, 求出所有好的序列 k 出现的次数的总和。

解: 首先,我们把 $p_i = 1$ 的所有 i 从大到小写出来,再把 $p_i = 2$ 的所有 i 从大到小写出来,这样不断地写,写到 $p_i = n$ 的所有 i ,会得到一个排列 q_1, q_2, \dots, q_n 。

由题目的条件,好的序列由这样的排列唯一确定,这是因为所有 $p_i = j$ 的下标恰好构成 q_1, q_2, \dots, q_n 的一个连续递减的段。且 p_{q_i} 的值恰好是满足 $p_j < p_{j+1}, 1 \leq j < q_i$ 的 j 的个数加一。

设 $d_{i,j}$ 表示在长度为 $i+1$ 的排列中先选择 j 个相邻项要求前一项小于后一项,再填出这个排列的总方案数。把每个连续的小于号当成一段,就可以转换为把 $i+1$ 个数划分成

$i - j + 1$ 个集合。容易得到：

$$d_{i,j} = (i + 1)! [x^{i+1}] (e^x - 1)^{i-j+1} \tag{28}$$

求答案时，我们考虑每个位置的贡献，那么我们有：

$$Ans_{i+1} = \sum_{x=0}^{n-1} \sum_{y=i}^x (-1)^{y-i} \binom{y}{i} d_{x,y} \frac{n!}{(x+1)!} \tag{29}$$

$$= \frac{n!}{i!} \sum_{x=0}^{n-1} \sum_{y=i}^x (-1)^{y-i} \frac{y! d_{x,y}}{(y-i)! (x+1)!} \tag{30}$$

$$= \frac{n!}{i!} \sum_{y=i}^{n-1} \frac{(-1)^{y-i} y!}{(y-i)!} \sum_{x=y}^{n-1} [z^{x+1}] (e^z - 1)^{x-y+1} \tag{31}$$

只需对每个 y 求出 $\sum_{x=y}^{n-1} [z^{x+1}] (e^z - 1)^{x-y+1}$ ，即可用一次卷积求出答案。

设 $F(z) = \frac{e^z - 1}{z}$ ，则我们考虑：

$$\sum_{x=y}^{n-1} [z^{x+1}] (e^z - 1)^{x-y+1} = \sum_{x=y+1}^n [z^x] (e^z - 1)^{x-y} \tag{32}$$

$$= [z^y] \sum_{x=1}^{n-y} \left(\frac{e^z - 1}{z}\right)^x \tag{33}$$

$$= [z^y] \frac{1 - F(z)^{n-y+1}}{1 - F(z)} \tag{34}$$

$$= [z^y] \frac{1}{1 - F(z)} - [z^{n+1}] \frac{(zF(z))^{n-y+1}}{1 - F(z)} \tag{35}$$

注意到 $\frac{1}{1-F(z)}$ 可以用一次多项式求逆求出，而 $[z^{n+1}] \frac{(zF(z))^{n-y+1}}{1-F(z)}$ 就归约到了 (23) 式类型的问题了。

由于 $zF(z)$ 的复合逆 $G(z)$ 和 $\frac{1}{1-F(G(z))}$ 很容易求出，故该问题仍然可以在 $O(n \log n)$ 的时间内解决！

6 多项式牛顿迭代的高级应用

6.1 含有 $G(x^2), G(x^3), \dots$ 的多项式方程

在第二节中，我们介绍了求解关于 $G(x)$ 的方程 $F(G(x), x) = 0$ 的方法。但是，在和问题中（特别是同构意义下树的计数），我们得到了多项式方程可能含有 $G(x^2), G(x^3)$ 这些项。这个时候，我们假设得到了 $G(x) \bmod x^k$ 的结果，那么 $G(x^2), G(x^3), \dots$ 在 $\bmod x^{2k}$ 的结果已经确定了。故在推导迭代公式时，我们只需要把 $G(x^2), G(x^3), \dots$ 当作一个常数即可！

例 7. 无标号有根树计数

在同构意义下求所有含有 n 个顶点的有根树个数。

设 $T(x)$ 表示含有 n 个顶点的有根树的个数的 OGF。我们试图得到 $T(x)$ 的一个方程。

注意到所有含有 i 个顶点的，总顶点数为 j 的有根树可重集合的个数为：

$$[x^j] \left(\frac{1}{(1-x^i)^{[x^i]T(i)}} \right) \tag{36}$$

而 n 个顶点的同构意义下有根树的个数就是总共 $n-1$ 个顶点的有根树可重集合的个数。

因此由多项式乘法的组合意义，我们有：

$$T(x) = x \prod_{i=1}^n \frac{1}{(1-x^i)^{[x^i]T(x)}} \tag{37}$$

对上式两边求 \ln ，再结合泰勒展开式 $-\ln(1-x) = x + \frac{x^2}{2} + \frac{x^3}{3} + \dots$ 可得：

$$\ln \frac{T(x)}{x} = \sum_{i=1}^{+\infty} \frac{T(x^i)}{i} \tag{38}$$

也即：

$$T(x) = x \cdot \exp\left(\sum_{i=1}^{+\infty} \frac{T(x^i)}{i}\right) \tag{39}$$

假设得到了 $A(x) \equiv T(x) \pmod{x^m}$ 的值，希望找到 $B(x) \equiv T(x) \pmod{x^{2m}}$ 。

把上式右边对 $\sum_{i=1}^{+\infty} \frac{A(x^i)}{i}$

这一点泰勒展开，并忽略其二次、三次项，我们有：

$$B(x) \equiv x \cdot \exp\left(\sum_{i=1}^{+\infty} \frac{A(x^i)}{i}\right) (1 + B(x) - A(x)) \pmod{x^{2m}} \tag{40}$$

这是一个关于 $B(x)$ 的线性方程，很容易通过一次多项式求逆解出 $B(x)$ 。且由调和级数的性质，计算 $\sum_{i=1}^{+\infty} \frac{A(x^i)}{i}$ 的复杂度也是 $O(m \log m)$ 的。

总复杂度为 $O(n \log n)$ 。

6.2 多项式牛顿迭代解微分方程

这一节我们将讨论如何解一部分 $G'(x) = F(G(x), x)$ 形式的方程。也就是所已知 $[x^0]G(0)$ ，求 $G(x)$ 的前 n 项。其中 $F(G(x), x)$ 是关于 $G(x), x$ 的二元生成函数。

6.2.1 递推方法

假设我们得到了 $[x^0]G(x), [x^1]G(x), \dots, [x^{m-1}]G(x)$ ，那么

$$[x^m]G(x) = \frac{1}{m} [x^{m-1}]G'(x) = \frac{1}{m} [x^{m-1}](F(G(x), x)) \tag{41}$$

值得注意的是，这种方法在 $F(G(x), x)$ 极其特殊的时候，能够取得很好的效果。例如 $F(G(x), x) = (x^5 + x^4 + 4x^3 + 5x^2 + x + 4)G(x)$ 等。这样每次递推的代价就变成了常数级别的了。另外，有些问题中，可以把这样的递推用 cdq 分治 NTT 优化成 $O(n \log^2 n)$ 的复杂度。但是对于一般的问题来说，这样做往往不够高效。

6.2.2 牛顿迭代方法

首先，我们得到 $[x^0]G(x), [x^1]G(x)$ 。

假设得到了 $G(x) \bmod x^m$ 的结果 $A(x)$ ，试图求出 $G(x) \bmod x^{2m-1}$ 的结果 $B(x)$ 。

把 $F(G(x), x)$ 对 $G(x) = A(x)$ 作泰勒展开，并由 $G'(x) \equiv B'(x) \pmod{x^{2m-1}}$ ，我们有：

$$B'(x) \equiv \frac{\partial F}{\partial G}(A(x) - B(x), x) + F(A(x), x) \tag{42}$$

$$\equiv P(x)B(x) + Q(x) \pmod{x^{2m-1}} \tag{43}$$

其中 $P(x), Q(x)$ 是化简得到的结果。

我们设 $R'(x) = P(x)$ ，且 $[x^0]R(x) = 0$ ，那么我们有：

$$(B(x)e^{-R(x)})' \equiv e^{-R(x)}Q(x) \pmod{x^{2m-1}} \tag{44}$$

我们用一次多项式积分和多项式求逆，多项式除法即可求出 $B(x)$ 。

因此，若根据 $A(x)$ 求 $P(x), Q(x)$ 这一步可以做到 $O(m \log m)$ ，那么总复杂度也能做到 $O(m \log m)$ 了。

7 再探复合与复合逆

7.1 多项式复合

前述的内容主要用于计算特殊情况下 的多项式复合与复合逆的问题，拉格朗日反演往往只能计算关于复合逆的式子中的一项，而牛顿迭代往往只能快速计算特殊多项式的复合逆。接下来我们简要地介绍一般的多项式复合、复合逆的算法。

给定多项式 $F(x), G(x)$ ，其次数均不超过 n 。我们要在 $O(n^2)$ 时间内计算出 $F(G(x))$ 的前 n 项系数。这个算法的核心是大步小步思想。

令 $B = \lceil \sqrt{n} \rceil$ ，我们先求出 $G(x)^0, G(x)^1, \dots, G(x)^B$ ，复杂度为 $O(n^{1.5} \log n)$ 。

再求出 $G(x)^B, G(x)^{2B}, \dots, G(x)^{B^2}$ ，复杂度仍然为 $O(n^{1.5} \log n)$ 。

接下来，我们发现：

$$F(G(x)) = \sum_{i=0}^{B-1} G(x)^{iB} \sum_{j=0}^{B-1} [x^{iB+j}]G(x)^j \tag{45}$$

这样，我们只需要用预处理的结果在 $O(n^{1.5})$ 内计算 B 个 $\sum_{j=0}^{B-1} [x^{iB+j}]G(x)^j$ ，然后用 B 次卷积在 $O(n^{1.5} \log n)$ 的时间内计算出 $F(G(x))$ ！

总时间复杂度为 $O(n^{1.5} \log n + n^2)$ 。事实上，卷积的那一部分由于常数的原因，可能反而比 $O(n^2)$ 的那一部分慢。可以通过调整 B 的大小来获得实际上更好的效果。

多项式复合还有一个理论上更好的算法，称为 Brent-Kung 算法，其时间复杂度为 $O((n \log n)^{1.5})$ 。然而由于常数原因，这个算法在时间上往往不能和常数优秀的 $O(n^2)$ 算法拉开差距。

首先，我们令正整数 $m = \Theta(\frac{\sqrt{n}}{\log n})$ 。设 $G(x) = G_1(x) + G_2(x)$ ，这里 $G_1(x)$ 被 x^m 所整除。将 $F(G(x))$ 在 $G_1(x)$ 这一点展开，我们有：

$$F(G(x)) = F(G_2(x)) + \frac{F'(G_2(x))}{1!} G_1(x) + \frac{F''(G_2(x))}{2!} G_1(x)^2 + \dots \quad (46)$$

我们发现，上式只有前 $\lceil \frac{n}{m} \rceil$ 项对结果有贡献。

故我们只要计算出 $F(G_2(x)), F'(G_2(x)), F''(G_2(x)), \dots$ ，然后做 $O(\frac{n}{m})$ 次长度为 $O(n)$ 的卷积即可计算出 $F(G(x))$ 。后面这一步复杂度为 $O(\frac{n}{m} n \log n) = O((n \log n)^{1.5})$

先考虑计算 $F(G_2(x))$ 。首先将 $F(x)$ 的系数分为 m 个连续段，每一段长度为 $\Theta(\frac{n}{m})$ 。那么我们只需要计算一个 $\leq \frac{n}{m}$ 次的多项式 $A(x)$ 复合一次 $\leq m$ 次的多项式 $B(x)$ 的结果。

这里我们可以用分治 NTT 的方法计算。每次将 $A(x)$ 分为两个次数几乎相等的多项式 $x^k A_1(x) + A_2(x)$ 。递归计算 $A_1(B(x)), A_2(B(x)), B(x)^k$ ，用 $O(\deg A(x) \cdot \log n)$ 的代价计算出 $A(B(x)), B(x)^{\deg A(x)}$ 。故计算 $A(B(x))$ 可以用 $O(n \log^2 n)$ 的代价算出。计算 $F(G_2(x))$ 也可以用 $O(mn \log^2 n) = O((n \log n)^{1.5})$ 的代价来实现。

若得到了 $F^{(k)}(G_2(x))$ ，那么用一次求导可以求出 $F^{(k+1)}(G_2(x))G_2'(x)$ 。再用一次多项式求逆即可算出 $F^{(k+1)}(G_2(x))$ 了（计算两多项式除法时，先不断除以 x ）。这样我们又用 $O(\frac{n}{m} \log n) = O((n \log n)^{1.5})$ 的代价从 $F(G_2(x))$ 推到了 $F'(G_2(x)), F''(G_2(x)), \dots$ 这 $\Theta(\frac{n}{m})$ 个多项式。

综上，我们得到了一个复杂度为 $O((n \log n)^{1.5})$ 的多项式复合算法，这被称为 Brent-Kung 算法。

7.2 多项式复合逆

而对于计算 $F(x)$ 的复合逆的问题，我们把它看成多项式方程 $F(G(x)) - x = 0$ ，再使用牛顿迭代进行求解即可！这里我们需要通过 $F'(G(x)), F(G(x))$ 来进行迭代。

如果采用 $O((n \log n)^{1.5})$ 的多项式复合算法，那么求复合逆的时间复杂度可以表示为：

$$T(n) = T\left(\frac{n}{2}\right) + O((n \log n)^{1.5}) \quad (47)$$

由 Master 定理，这个算法的时间复杂度仍然为 $O((n \log n)^{1.5})$ 。

8 总结

本文介绍了多项式拉格朗日反演、牛顿迭代的基本理论，通过一定程度上的抽象，给出了比较通用的解法。

计数组合的理论博大精深，本文仅起到抛砖引玉的作用。特别遗憾的是，本文提出的算法有一定的局限性，例如不能计算模小素数、模合数的结果，不能解形式更复杂的微分方程等等。

希望本文能激发选手们对计数组合研究的热情，也希望这方面的知识能够更好、更全面地引入 OI 界。

9 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢教练金靖老师、吴申广老师对我的指导。

感谢父母对我的培育和教诲。

感谢所有帮助我的同学、老师。

参考文献

- [1] 金策,《生成函数的运算与组合计数问题》
- [2] <https://www.luogu.com.cn/blog/yurzhang/solution-p5373>, Brent-Kung 算法介绍
- [3] <http://codeforces.com/blog/entry/77284>, 例 5 的英文题解
- [4] <https://www.luogu.com.cn/blog/Caro23333/codeforces-round-641-div1f-slime-and-sequences-zhong-wen-ti-xie>, 例 5 的中文题解
- [5] https://blog.csdn.net/sslz_fsy/article/details/104846902,
- [6] 刘汝佳,《算法竞赛入门经典》,清华大学出版社。

浅谈一类最小公倍数的求和问题及其拓展

成都外国语学校 张隽恺

摘要

本文介绍了一类最小公倍数的求和问题及问题相对一般的情况，以及这类问题上一个通过合并无用状态减少状态数的一个动态规划做法。

1 前言

在信息学竞赛中，与最大公约数相关的问题出现频率较高。但与之相对的最小公倍数问题因为较为复杂而极少出现。本文将简单描述一类与最小公倍数相关问题及这类问题的拓展。

在本文第二节中，首先介绍了一个最小公倍数的求和问题以及几个朴素做法，然后描述了一个通过结合前几个做法的思路，并使用一些性质减少状态数的动态规划做法，接着分析了该算法的复杂度以及一些拓展。

在第三节中简要分析了这类问题拓展到更一般情况下的结果以及上一个算法此时的表现。

2 一类最小公倍数的求和问题

例题 1 给定正整数 n ，求

$$\sum_{S \subset \{1, 2, \dots, n\}} LCM(S)$$

$n \leq 2000$ ，答案对质数取模。

这里 $LCM(S)$ 表示集合 S 中的所有元素的最小公倍数。特别的，定义 $LCM(\emptyset) = 1$ 。

2.1 算法一

使用朴素的动态规划算法，设 $dp_{i,j}$ 表示前 i 个数组成的集合的子集中，子集内所有元素的最小公倍数为 j 的子集数量。

在从 dp_i 转移到 dp_{i+1} 时, 枚举 $i+1$ 是否在子集中即可。

设 S_n 为 $\{1, 2, \dots, n\}$ 的子集的最小公倍数的种类数, 则这个算法的时间复杂度为 $O(S_n * n * \log n)$ 或 $O(S_n * n)$ 。可以在 1 秒内通过 $n = 50$ 的数据。

2.2 算法二

2.2.1 定义

为了更好地描述接下来的算法, 在这里先进行一些定义。

设 $1, 2, \dots, n$ 中的所有质数从小到大排序后为 p_1, p_2, \dots, p_k 。定义正整数 a 能被 p_1, p_2, \dots, p_k 分解, 当且仅当将 a 质因数分解后, 得到的所有质因数都属于 $\{p_1, p_2, \dots, p_k\}$ 。对于能被 p_1, p_2, \dots, p_k 分解的 a , a 进行质因数分解的结果为 $a = p_1^{q_1} * p_2^{q_2} * \dots * p_k^{q_k}$, 定义 a 的分解式为一个 k 维向量 $F_a = (q_1, q_2, \dots, q_k)$ 。

对于两个 k 维向量 $F = (q_1, q_2, \dots, q_k), F' = (q'_1, q'_2, \dots, q'_k)$, 定义 $F \leq F'$ 当且仅当 $\forall i \in \{1, 2, \dots, k\}, q_i \leq q'_i$ 。

对于 $F = (q_1, q_2, \dots, q_k), F' = (q'_1, q'_2, \dots, q'_k)$, 定义它们的 \max 为

$$\max(F, F') = (\max(q_1, q'_1), \max(q_2, q'_2), \dots, \max(q_k, q'_k))$$

显然这样定义的 \max 满足交换律。使用类似的方式定义 \min :

$$\min(F, F') = (\min(q_1, q'_1), \min(q_2, q'_2), \dots, \min(q_k, q'_k))$$

定义 $\max(F_1, F_2, \dots, F_l) = \max(\max(F_1, \max(F_2, \dots, F_l)))$ 。

可以得到如下结论:

Lemma 2.1. 如果 a, b 都能被 p_1, p_2, \dots, p_k 分解, 则 $LCM(a, b)$ 能被 p_1, p_2, \dots, p_k 分解, 且 $F_{LCM(a,b)} = \max(F_a, F_b)$ 。

证明. 设 $F_a = (a_1, a_2, \dots, a_k), F_b = (b_1, b_2, \dots, b_k)$ 。则 $a = \prod_{i=1}^k p_i^{a_i}, b = \prod_{i=1}^k p_i^{b_i}$ 。因此 $LCM(a, b) = \prod_{i=1}^k p_i^{\max(a_i, b_i)}$ 。因此 $LCM(a, b)$ 能被 p_1, p_2, \dots, p_k 分解, 且 $F_{LCM(a,b)} = (\max(a_1, b_1), \dots, \max(a_k, b_k)) = \max(F_a, F_b)$ 。

□

扩展到多个数的情况, 可以得到:

Lemma 2.2. 如果 a_1, a_2, \dots, a_l 都能被 p_1, p_2, \dots, p_k 分解, 则 $LCM(\{a_1, a_2, \dots, a_l\})$ 能被 p_1, p_2, \dots, p_k 分解, 且 $F_{LCM(\{a_1, a_2, \dots, a_l\})} = \max(F_{a_1}, F_{a_2}, \dots, F_{a_l})$ 。

证明. 使用与 Lemma 2.1 相同的方法即可证明。

□

2.2.2 算法

显然 $1, 2, \dots, n$ 都能被 p_1, p_2, \dots, p_k 分解。根据 Lemma 2.2, 所有子集的最小公倍数也能被 p_1, p_2, \dots, p_k 分解, 且满足子集中所有数的最小公倍数等于 a 的子集个数等于满足子集中所有数的分解式的 \max 等于 F_a 的子集数量。

考虑如下引理:

Lemma 2.3. 对于分解式 F_a, F_b, F_c , $\max(F_a, F_b) \leq F_c$ 当且仅当 $F_a \leq F_c, F_b \leq F_c$ 。

证明. 设 $F_a = (a_1, a_2, \dots, a_k), F_b = (b_1, b_2, \dots, b_k), F_c = (c_1, c_2, \dots, c_k)$, 因为 $F_a \leq F_c, F_b \leq F_c$, 则 $\forall i \in \{1, 2, \dots, k\}, a_i \leq c_i, b_i \leq c_i$, 因此 $\max(a_i, b_i) \leq c_i$, 反之同理。因此引理成立。

□

Lemma 2.4. 设 $G = ([\log_{p_1} n], [\log_{p_2} n], \dots, [\log_{p_k} n])$, 则 $\forall i \in \{1, 2, \dots, n\}, F_i \leq G$ 。

证明. 若引理不成立, 则 $\exists a \in \{1, 2, \dots, n\}$ 满足 $F_a \not\leq G$ 。设 $F_a = (a_1, a_2, \dots, a_k)$, 则 $\exists r \in \{1, 2, \dots, k\}, a_r > [\log_{p_r} n]$ 。此时 $a = \prod_{i=1}^k p_i^{a_i} \geq p_r^{a_r} \geq p_r^{[\log_{p_r} n]+1} > n$, 与 $a \in \{1, 2, \dots, n\}$ 矛盾。因此引理成立。

□

Lemma 2.5. 对于所有满足 s 能被 p_1, p_2, \dots, p_k 分解, 且 $F_s \leq ([\log_{p_1} n], [\log_{p_2} n], \dots, [\log_{p_k} n])$ 的正整数 s , 存在一个集合 $S \subset \{1, 2, \dots, n\}$ 满足 $LCM(S) = s$ 。

证明. 设 $F_s = (q_1, q_2, \dots, q_k)$, 则 $\forall i \in \{1, 2, \dots, k\}, q_i \leq [\log_{p_i} n], p_i^{q_i} \leq n$ 。构造序列 $t_{1, \dots, k}, \forall i \in \{1, 2, \dots, k\}, t_i = p_i^{q_i}$, 则 $LCM(t_1, t_2, \dots, t_k) = \prod_{i=1}^k p_i^{q_i}$, 序列中元素构成的集合满足条件。

□

满足 $F \leq ([\log_{p_1} n], [\log_{p_2} n], \dots, [\log_{p_k} n])$ 且 F 每一维上的值均为非负整数的 F 共有 $\prod_{i=1}^k ([\log_{p_i} n] + 1)$ 个。根据 Lemma 2.5, 对于所有满足这个条件的 F , 都存在一个 $S \subset \{1, 2, \dots, n\}$ 满足 $LCM(S)$ 的分解式等于它。根据 Lemma 2.3 和 Lemma 2.4, $\forall S \subset \{1, 2, \dots, n\}, F_{LCM(S)} \leq ([\log_{p_1} n], [\log_{p_2} n], \dots, [\log_{p_k} n])$ 。因此最小公倍数的种类数 S_n 等于 $\prod_{i=1}^k ([\log_{p_i} n] + 1)$ 。

对于每一个满足这一条件的 F , 设 h_F 为满足子集中所有数的最小公倍数的分解式 F_{LCM} 满足 $F_{LCM} \leq F$ 的子集数量。设 ct_F 为 $\{1, 2, \dots, n\}$ 中满足 $F_i \leq F$ 的数 i 的数量。根据 Lemma 2.3, 满足条件的子集中的所有数 a 都满足 $F_a \leq F$, 且若子集中所有数 a 都满足 $F_a \leq F$, 则这个集合满足条件。因此 $h_F = 2^{ct_F}$ 。因为 $ct_F = \sum_{i=1}^n [F_i \leq F]$, 求出 ct 相当于求高维前缀和, 使用 [1] 中的做法可以以 $O(S_n * k)$ 的复杂度求出 ct 。

设 $H_{F'}$ 为子集中所有数的最小公倍数的分解式 $F_{LCM} = F'$ 的子集数量。通过 H 求 h 为高维前缀和的过程, 因此通过高维差分可以通过 h 求 H 。

这个做法的复杂度为 $O(S_n * k)$, 效率与上一个算法接近。

2.3 算法三

$1, \dots, n$ 中的所有数质因数分解后最多有一个大于 \sqrt{n} 的质因子。在动态规划时，只记录状态中所有不超过 \sqrt{n} 的质因子。将所有数按其大于 \sqrt{n} 的质因子分类，对每一类进行动态规划。这部分也可以使用算法二中的高维前缀和进行优化。

这个算法的具体细节与之后的算法无关，因此在这里不再展开。

2.4 算法四

2.4.1 算法

考虑算法一中的动态规划算法，设 $g_{i,F}$ 表示前 i 个数的集合的子集中，子集的最小公倍数的分解式为 F 的方案数。

定义 $g_{i,F}$ 对答案的贡献 $v_{i,F}$ 为：(设 $F = (q_1, q_2, \dots, q_k)$)

$$v_{i,F} = \sum_{S \subset \{i+1, i+2, \dots, n\}} LCM\left(\prod_{j=1}^k p_j^{q_j}, LCM(S)\right)$$

则有如下引理：

Lemma 2.6. 设问题的答案为 ans ，则

$$\forall i \in \{1, 2, \dots, n\}, \sum_F g_{i,F} * v_{i,F} = ans$$

证明.

$$\begin{aligned} ans &= \sum_{S \subset \{1, 2, \dots, n\}} LCM(S) \\ &= \sum_{S_1 \subset \{1, 2, \dots, i\}} \sum_{S_2 \subset \{i+1, i+2, \dots, n\}} LCM(LCM(S_1), LCM(S_2)) \\ &= \sum_{F=(q_1, q_2, \dots, q_k)} g_{i,F} \sum_{S_2 \subset \{i+1, i+2, \dots, n\}} LCM\left(\prod_{j=1}^k p_j^{q_j}, LCM(S_2)\right) \\ &= \sum_F g_{i,F} * v_{i,F} \end{aligned}$$

□

Lemma 2.7. 对于 $g_{i,F}$ ，设 $LCM(i+1, i+2, \dots, n) = s$, $F = (q_1, q_2, \dots, q_k)$, $F_s = (q'_1, q'_2, \dots, q'_k)$ 。若存在 $a \in \{1, 2, \dots, k\}$ 满足 $q_a > q'_a$ ，设 $F' = (q_1, q_2, \dots, q_{a-1}, q_a - 1, q_{a+1}, q_{a+2}, \dots, q_k)$ ，则 $v_{i,F} = p_a * v_{i,F'}$ 。

证明. 考虑集合 $S \subset \{i+1, i+2, \dots, n\}$ ，设 $LCM(S) = s'$, $F_{s'} = (s_1, s_2, \dots, s_k)$ 。由 2.3 有 $F_{s'} \leq F_s$ 。因此 $s_a \leq q'_a \leq q_a - 1$ 。

$$LCM\left(\prod_{j=1}^k p_j^{q_j}, s'\right) = \left(\prod_{j=1, j \neq a}^k p_j^{\max(q_j, s_j)}\right) * p_a^{\max(q_a, s_a)} = \left(\prod_{j=1, j \neq a}^k p_j^{\max(q_j, s_j)}\right) * p_a^{q_a}$$

$$LCM\left(\left(\prod_{j=1, j \neq a}^k p_j^{q_j}\right) * p_a^{q_a-1}, s'\right) = \left(\prod_{j=1, j \neq a}^k p_j^{\max(q_j, s_j)}\right) * p_a^{\max(q_a-1, s_a)} = \left(\prod_{j=1, j \neq a}^k p_j^{\max(q_j, s_j)}\right) * p_a^{q_a-1}$$

所以 $LCM\left(\prod_{j=1}^k p_j^{q_j}, s'\right) = p_a * LCM\left(\left(\prod_{j=1, j \neq a}^k p_j^{q_j}\right) * p_a^{q_a-1}, s'\right)$ 。对所有 S 求和即可得到 $v_{i,F} = p_a * v_{i,F'}$ 。

□

对于一个满足 Lemma 2.7 中条件的 F ，设 g'_i 为：

1. $g'_{i,F} = 0, g'_{i,F'} = g_{i,F'} + g_{i,F} * p_a$
2. 对于所有满足 $F'' \neq F, F'' \neq F'$ 的 F'' , $g'_{i,F''} = g_{i,F''}$

根据 Lemma 2.7 可得 $\sum_F g_{i,F} * v_{i,F} = \sum_F g'_{i,F} * v_{i,F}$ 。根据 Lemma 2.6，将 g_i 替换为 g'_i 不会改变答案。定义将 g_i 变为 g'_i 的过程为一次操作。对于任意的 g_i ，易证在有限次操作后可以使得所有非零的 $g_{i,F}$ 都满足 $F \leq F_{LCM(i+1, i+2, \dots, n)}$ 。

由 Lemma 2.3， $\{1, 2, \dots, i\}$ 的任意子集 S 满足 $F_S \leq F_{LCM(1, 2, \dots, i)}$ 。对于每一个 i ，在求出 g_i 后进行若干次操作，直到所有满足 $g_{i,F}$ 非零的 F 都满足 $F \leq F_{LCM(i+1, i+2, \dots, n)}$ 。这时满足 $g_{i,F}$ 非零的 F 满足 $F \leq F_{LCM(1, 2, \dots, i)}, F \leq F_{LCM(i+1, i+2, \dots, n)}$ 。在从 g_i 转移到 g_{i+1} 的过程中，满足 $g_{i+1,F}$ 非零的 F 一定满足 $F \leq F_{LCM(1, 2, \dots, i+1)}, F \leq F_{LCM(i+1, i+2, \dots, n)}$ 。记向量 $L_i = \min(F_{LCM(1, 2, \dots, i)}, F_{LCM(i+1, i+2, \dots, n)})$, $L'_i = \min(F_{LCM(1, 2, \dots, i)}, F_{LCM(i, i+1, \dots, n)})$ 。则可以得到这样一个算法：

维护若干状态 F 以及这些状态对应的 g_F ，依次考虑 $i = 1, 2, \dots, n$ ，对于每个 i 依次进行以下三步操作：

1. 将当前维护的 F 的范围变为所有满足 $F \leq L'_i$ 且 F 的每一维都是非负整数的 F 。显然 $L_{i-1} \leq L'_i$ ，因此这一步不会改变 g 。
2. 计算在子集中加入 i 的转移，进行 g 上的转移。
3. 通过进行若干次操作，使所有非零的 g_F 都满足 $F \leq F_{LCM(i+1, i+2, \dots, n)}$ ，将维护的 F 的范围变为所有满足 $F \leq L_{i+1}$ 且 F 的每一维都是非负整数的 F 。

根据 Lemma 2.7，在进行 i 的操作时，每一个操作后 $\sum_F g_F * v_{i,F}$ 不变。因此在进行上面的操作过程中，这个值始终等于答案。因为 $L_{n+1} = (0, 0, \dots, 0)$ ，进行所有操作后只有一个状态 $F = (0, 0, \dots, 0)$ ，显然 $v_{n+1,F} = 1$ ，因此这个状态对应的 g 即为问题的答案。

使用算法二中的方式优化转移。设 $f_F = \sum_{F' \leq F} g_{F'}$ ，即 f 为 g 进行高维前缀和后的结果。在这个算法中，任意时刻都存在一个 k 维向量 L''_i (等于 L_i 或 L'_i)，满足当前所有满足 g_F 非零的 F 都满足 $F \leq L''_i$ 。因此只需要维护满足 $F \leq L''_i$ 的 f_F 。在上面的过程中代替 g 维护 f ，依次分析每一步操作对 f 的影响 (设 f' 为操作后的 f)：

对于第一步操作， g 不会改变。设 $L_i = (l_1, l_2, \dots, l_k)$ ，因为所有满足非零的 g_F 都满足 $F \leq L_i$ ，因此对于任意状态 F ，有 $f'_F = f_{\min(F, L_i)}$ 。

对于加入一个数 i 的转移对 f 的影响, 根据 2.3 以及算法二中的分析, 可以得到:

$$f'_F = \begin{cases} 2 * f_F, & F_{i+1} \leq F \\ f_F, & F_{i+1} \not\leq F \end{cases} \quad (1)$$

对于第三步操作, 设 $L'_i = (l'_1, l'_2, \dots, l'_k)$, 将操作分为若干步, 每一步选择一维 a , 对所有第 a 维上的值等于 l'_a 的状态 F 进行操作, 使非零的 $g_{i,F}$ 都满足 F 第 a 维上的值小于 l'_a 。

考虑这样的一步操作对 g 的影响 (设 g' 为操作后的 g):

$$g'_{(q_1, q_2, \dots, q_k)} = \begin{cases} g_{(q_1, q_2, \dots, q_k)}, & q_a < l_a - 1 \\ g_{(q_1, q_2, \dots, q_k)} + p_a * g_{(q_1, q_2, \dots, q_{a-1}, q_a+1, q_{a+1}, \dots, q_k)}, & q_a = l_a - 1 \\ 0, & q_a = l_a \end{cases}$$

根据 f 的定义, 可以求出操作对 f 的影响:

$$f'_{(q_1, q_2, \dots, q_k)} = \begin{cases} f_{(q_1, q_2, \dots, q_k)}, & q_a < l_a - 1 \\ p_a * f_{(q_1, q_2, \dots, q_{a-1}, q_a+1, q_{a+1}, \dots, q_k)} - (p_a - 1) * f_{(q_1, q_2, \dots, q_k)}, & q_a = l_a - 1 \end{cases}$$

(在操作后所有值非零的状态第 a 维上的值都小于 l_a , 因此不需要保留满足 $q_a = l_a$ 的 f)

通过对操作次数归纳可得, 在进行 x 次这样的操作后, 得到的 f' 为:

$$f'_{(q_1, q_2, \dots, q_k)} = \begin{cases} f_{(q_1, q_2, \dots, q_k)}, & q_a < l_a - x \\ p_a^x * f_{(q_1, q_2, \dots, q_{a-1}, l_a, q_{a+1}, \dots, q_k)} - \sum_{i=0}^{x-1} p_a^i (p_a - 1) * f_{(q_1, q_2, \dots, q_{a-1}, l_a - x + i, q_{a+1}, \dots, q_k)}, & q_a = l_a - x \end{cases}$$

依次进行每一维上的操作即可。

通过这三步操作即可维护 F 从而求出答案, 接下来分析这个算法的实现与复杂度。

2.4.2 实现与复杂度分析

在算法的过程中, 任意时刻都存在一个各维的值都为非负整数的向量 $L'' = (l_1, l_2, \dots, l_k)$, 满足当前维护的所有 f_F 满足 $F \leq L''$ 。

定义一个从当前所有的状态 $\{(q_1, q_2, \dots, q_k) \mid \forall j \in \{1, 2, \dots, k\}, 0 \leq q_j \leq l_j, q_j \in \mathbb{N}\}$ 到 $\{0, 1, \dots, \prod_{j=1}^k (l_j + 1) - 1\}$ 的映射: $f1 : \{(q_1, q_2, \dots, q_k) \mid \forall j \in \{1, 2, \dots, k\}, 0 \leq q_j \leq l_j, q_j \in \mathbb{N}\} \rightarrow \{0, 1, \dots, \prod_{j=1}^k (l_j + 1) - 1\}$:

$$f1((q_1, q_2, \dots, q_k)) = \sum_{j=1}^k q_j * \left(\prod_{j'=j+1}^k (l_{j'} + 1) \right)$$

Lemma 2.8. $f1$ 是一个双射。

证明. 每一个 (q_1, q_2, \dots, q_k) 只会对应一个 $\{0, 1, \dots, \prod_{j=1}^k (l_j + 1) - 1\}$ 中的数。接下来证明 $\forall s \in \{0, 1, \dots, \prod_{j=1}^k (l_j + 1)\}$, 存在唯一一个满足 $\forall j \in \{1, 2, \dots, k\}, 0 \leq q_j \leq l_j, q_j \in \mathbb{N}$ 的 (q_1, q_2, \dots, q_k) 。

由 f_1 的定义可以得到 $(\sum_{j=1}^{k-1} q_j * (\prod_{j'=j+1}^{k-1} (l_{j'} + 1))) * (l_k + 1) + q_k = s$, 则 $q_k \equiv s \pmod{l_k + 1}$ 。因为 $q_k \in \{0, 1, \dots, l_k\}$, 所以存在唯一一个满足条件的 q_k 。 q_k 固定后, 剩余部分相当于 $\sum_{j=1}^{k-1} q_j * (\prod_{j'=j+1}^{k-1} (l_{j'} + 1)) = \frac{s - q_k}{l_k + 1}$, 且满足 $\frac{s - q_k}{l_k + 1} \in \{0, 1, 2, \dots, \prod_{j=1}^{k-1} (l_j + 1) - 1\}$ 。使用归纳法即可证明引理。□

根据 Lemma 2.8, 可以对于所有的 $s \in \{0, 1, \dots, \prod_{j=1}^k (l_j + 1)\}$, 记录 s 通过 f_1 对应的 F 对应的 f_F 。依次考虑关于 F 的三步操作:

设操作前的状态为 f , 操作后的状态为 f' 。第一步操作相当于对于每一个满足 $F \leq L'_i = (l'_1, l'_2, \dots, l'_k)$ 的 F , 设 $F = (q_1, q_2, \dots, q_k)$, 则 $f'_F = f_{(\min(q_1, l'_1), \min(q_2, l'_2), \dots, \min(q_k, l'_k))}$ 。

使用类似深度优先搜索的方式枚举 F 每一维上的值, 在枚举每一维上的值时, 可以求出 $(\min(q_1, l'_1), \min(q_2, l'_2), \dots, \min(q_k, l'_k))$ 每一维上的值以及这两个状态通过 f_1 对应的值。如果记录所有满足 $l'_i > 0$ 的维, 只枚举这些维上的值, 则枚举的复杂度为 $O(\prod_{j=1}^k (l'_j + 1))$ 。

对于后两个操作, 每一个 f_F 只会影响一个 f' 中的值。使用类似深度优先搜索的方式枚举每一个 F , 可以在这个过程中求出它影响的 f' 中的状态以及系数。这两步操作的复杂度同样为 $O(\prod_{j=1}^k (l'_j + 1))$ 。

因此这个实现的复杂度与每一个 L'_i 的 $\prod_{j=1}^k (l'_j + 1)$ 的和成正比。记这个值为 T_n 。可以得到 T_n 的大小 (所有大小保留三位有效数字):

n	50	100	200	500	1000
S_n	4.42×10^5	6.61×10^8	4.75×10^{15}	1.08×10^{31}	5.59×10^{53}
T_n	5.69×10^4	1.83×10^7	4.00×10^{10}	8.79×10^{19}	2.43×10^{33}

这样的效率不优于算法三, 但根据算法三中优化的方式, 可以对算法四进行修改:

将所有数按照它们的最大质因子从大到小排序 (最大质因子相同时从小到大排序), 按照这个顺序进行算法四中的做法。

按照最大质因子大小排序后进行算法四的做法, 设这时每一个 L'_i 的 $\prod_{j=1}^k (l'_j + 1)$ 的和为 T'_n , 有:

n	500	1000	2000	3000	5000	10^4
T'_n	1.35×10^5	1.44×10^6	2.45×10^7	1.97×10^8	3.60×10^9	4.66×10^{11}

T'_n 的增长速度仍然为指数级, 但 T'_n 的增长速度相对 S_n 较慢, 且在 $n \leq 2000$ 时 $O(T'_n)$ 的复杂度是可以接受的。因此算法四可以在 1 秒内通过 $n = 2000 \sim 2500$ 的数据。

实际上这个算法的复杂度并不是完全满的, 实际效率可以更快一点。

通过按这个顺序进行动态规划，可以得到相对小的状态数 T'_n 。但这样的状态数不是最优的，通过使用部分随机化算法 [2,3]，可以得到更小的状态数。例如作者使用随机化算法得到的部分状态数 T''_n ：

n	500	1000	2000	3000
T'_n	1.35×10^5	1.44×10^6	2.46×10^7	1.97×10^8
T''_n	1.25×10^5	1.35×10^6	2.34×10^7	1.93×10^8

但因为作者水平有限，并没有得到优于指数级复杂度的求出最优状态数的算法。在这里不再进行分析。

2.5 拓展

算法四可以拓展到相对一般的情况，即：

给定 n 个非负整数 a_1, a_2, \dots, a_n 以及一个满足 $\forall i \in \mathbb{N}^+, f(i) \neq 0$ 的积性函数 f ，求：

$$\sum_{S \subseteq \{1, 2, \dots, n\}} \left(\prod_{i \in S} a_i \right) * f(\text{LCM}(S)) \tag{2}$$

重新定义 $v_{i,F}$ ： $v_{i,(q_1, q_2, \dots, q_k)} = \sum_{S \subseteq \{i+1, i+2, \dots, n\}} \left(\prod_{i \in S} a_i \right) * f(\text{LCM}(\prod_{j=1}^k p_j^{q_j}, \text{LCM}(S)))$ 。

在这个定义下，Lemma 2.7 的结论变为 $v_{i,F} = \frac{f(p_i^{q_i})}{f(p_i^{q_i-1})} * v_{i,F'}$ ，(1) 中 $2 * f_F$ 变为 $(a_i + 1) * f_F$ 。剩余部分使用与算法四完全相同的方式推导即可。

例题 2 Math is Fun ¹

给一个长度为 n 的正整数序列 v ，定义一个序列的权值为序列中所有元素的最小公倍数的平方与序列中所有元素的最大公约数的乘积。求出这个序列所有 $2^n - 1$ 个非空子序列的权值之和模 $10^9 + 7$ 的结果。一共有 T 组数据。

$n \leq 100, v_i \leq 1000, T \leq 50$ 。时限 4 秒。

设 f_i 表示所有满足最大公约数等于 i 的非空子序列的序列元素最小公倍数的平方的和， g_i 表示所有满足最大公约数为 i 的倍数的非空子序列的序列元素最小公倍数的平方的和。

首先求出 g_i 。所有数的最大公约数为 i 的倍数当且仅当所有数都是 i 的倍数。因此 g_i 等于这些数的所有非空子序列的序列元素最小公倍数的平方和。显然若 $g | s_1, s_2, \dots, s_l$ ，则 $\text{LCM}(s_1, s_2, \dots, s_l) = \text{LCM}(\frac{s_1}{g}, \frac{s_2}{g}, \dots, \frac{s_l}{g}) * g$ 。则 g_i 等于只考虑序列中是 i 的倍数的数得到的序列，将这个序列所有元素除以 i 后，所有非空子序列的序列元素最小公倍数的平方和乘 i^2 的结果。

¹来源：Petrozavodsk Programming Camp, Summer 2016, Day 8, DPRK Contest

序列中所有元素都不会超过 $\frac{1000}{i}$ 。设数 a 出现了 c_a 次。令 $v_a = 2^{c_a} - 1$ ，再设函数 $f(n) = n^2$ ，则问题变为 (2) 的形式，可以使用算法四解决。计算所有 g_i 的复杂度与 $\sum_{i=1}^{1000} T'_{\frac{1000}{i}}$ 成正比。

然后通过 g_i 求出所有 f_i ，有 $g_i = \sum_{ij} f_j$ ，可以通过对 g 容斥求出 f 。这一部分的复杂度相对于上一部分可以忽略。

$\sum_{i=1}^{1000} T'_{\frac{1000}{i}} \leq 1.7 \times 10^6$ ，这个复杂度可以接受。作者的实现在此问题上运行时间约为 0.1 秒。

3 问题的拓展

对于更为一般的情况，可以将问题描述为如下形式：

给出 n 个 d 维向量 V_1, V_2, \dots, V_n 以及 n 个非负整数 a_1, a_2, \dots, a_n 。给出的所有向量满足每一维上的值都为非负整数。对两个向量取 \max 的定义同上一节。求：

$$\sum_{S \subset \{1, 2, \dots, n\}} \left(\prod_{i \in S} a_i \right) * f(\max_{i \in S} V_i)$$

其中 f 为一个值域为正整数的函数，满足 $f((q_1, q_2, \dots, q_d)) = \prod_{i=1}^d f_i(q_i)$ ，且所有 f_i 均为值域为正整数的函数。

上一个问题为这个问题的一个特例，满足给出的所有向量 $V = (q_1, q_2, \dots, q_d)$ 都满足 $\sum_{i=1}^d q_i * \log p_i \leq \log n$ 。

在不存在特殊限制的情况下，算法四中的状态数总和 T'_n 与 $S_n * n$ 接近。但在存在类似上一个限制的限制时，通过与算法四中类似的调整顺序的方式，得到的 T'_n 可以远小于 $S_n * d$ ，例如：

满足给出的所有向量 $V = (q_1, q_2, \dots, q_d)$ 都满足 $\sum_{i=1}^d q_i * i \leq m$ 时 ($d = m$ ，此处 S_n, T'_n 的定义同算法四中的定义，将所有向量按照最后一个非零位置排序)：

m	15	18	20	23	25
$S_n * d$	4.25×10^8	2.65×10^{10}	3.22×10^{11}	8.44×10^{12}	1.70×10^{14}
T'_n	7.63×10^5	9.31×10^6	4.99×10^7	5.83×10^8	2.96×10^9

在类似的问题上，算法四有优于算法一，二等做法的表现。与上一个问题相同，这里也存在使 T'_n 更小的向量顺序。

4 总结

本文描述的算法在动态规划的过程中，通过进行变换，将状态中不会被剩余的动态规划过程中改变的部分进行合并，从而达到减小状态数的结果。

由于最小公倍数问题的特殊性质，在改变动态规划的顺序后，该算法可以在这个问题上得到较优秀的结果。

同时，这个思想可以对更多的动态规划的问题进行优化。

感谢

感谢中国计算机学会提供学习和交流的平台。

感谢学校以及教练对我的支持。

感谢家人对我的关心。

参考文献

- [1] OI Wiki, "基于 DP 计算高维前缀和".
- [2] Wikipedia, "Simulated annealing".
- [3] 任一恒, 《非完美算法初探》, IOI2009 中国国家队候选队员论文.

浅谈一类树分块的构建算法及其应用

杭州第二中学 周欣

摘要

树分块算法是序列分块算法在树结构上的自然推广，具有广泛的应用场合。本文介绍了树分块的一类实现方式，并通过一些例题介绍了其应用。

引言

序列分块算法是一种朴素但有效的算法，通过与其它方法相结合，能够处理很多只靠传统树形数据结构不能处理的复杂问题，具有简洁高效的特点。

但是当我们试图将简洁高效的序列分块算法推广并用于处理树上问题的时候，由于树结构本身所具有的复杂性，无论是将树划分为若干个块的过程，还是成功划分之后借助分块结构处理问题的过程，相对于序列而言都增加了不少难度。

这也导致近年来OI中很多可以应用树分块算法解决的问题，为大家所熟知的解法都是借助一些通用手段¹来将问题简化，从而规避对复杂的树结构的直接处理。总之，近年来树分块算法在OI中的应用比较局限。

对此，笔者希望通过本文，介绍如何将序列分块算法推广到树结构上，以及树分块算法的一些应用。

本文的第一节会对树分块算法本身进行介绍，第二节讨论树分块算法在一些复杂度根号问题上的应用，第三节讨论非传统块大小的树分块在一些其它问题上的应用。

1 树分块算法简介

可以用于树分块构建的方法很多，本节中仅仅介绍一种基于 top cluster 理论的，适用面较广的做法。

¹比如沿时间轴定期重构

1.1 top cluster 概念的介绍

一个树簇 (cluster) 是树上的一个连通子图, 有至多两个点和全树的其他位置连接。
 这两个结点被称做界点 (boundary Node)。
 不是界点的点被称做内点 (internal node)。
 这两个界点之间的路径被称做簇路径 (cluster path)。

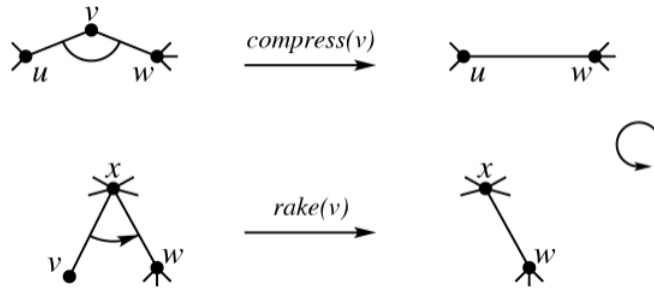


图 1

如上图图 1, 簇是可合并的, 并且簇的合并有两种方式。

如图所示, 第一种合并方式被命名为 $compress$, 要求对一个恰与两个簇相邻的点 v 执行, 效果为将与其相邻的两个簇 (u, v) 和 (v, w) 合并为一个新簇 (u, w) 。

第二种合并方式被命名为 $rake$, 要求对一个恰与一个簇相邻的点 v 执行, 记与其相邻的簇的另一个端点为 x , 与 x 相邻的除了 v 的另一个簇端点为 w , 则效果为将 (v, x) 合并至 (w, x) 中。

无论是第一种还是第二种合并, 执行完毕后点 v 都不再作为任何一个簇端点出现。显然, 通过不断执行合并操作可以将整棵树收缩为单个簇。

簇的合并过程构成一个二叉树的结构 (如图 2), 我们称之为 $top\ tree$ 。

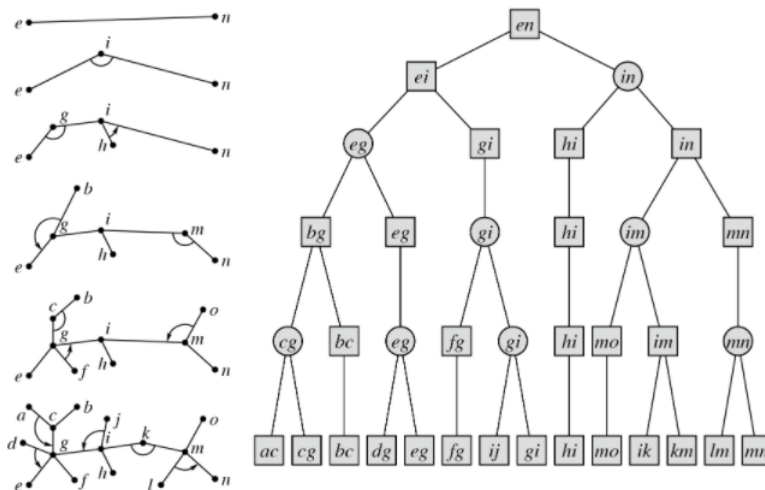


图 2

存在构造算法使得得到的 top tree 深度为 $O(\log n)$, 关于 top tree 的更多其它内容, 由于与主题无关, 这里不展开详细描述。²。

1.2 树分块的基本结构

序列分块可以看作是一种只有两层节点的特殊树形数据结构。当我们将序列分块推广到树结构上的时候, 自然的想法是, 寻找一个将树表示为多层树形结构的基底数据结构, 然后将节点强行拍扁成两层。

而上一节中 top tree 的概念, 恰好为我们提供了这样一种基底数据结构。

于是, 我们可以借用 top 簇的概念, 精确地描述树分块构造算法所应当解决的问题: 对于一棵给定的 n 个点的树和一个块大小 B , 要求将原树划分为 $O(\frac{n}{B})$ 个不交簇的并, 并且每个簇的大小均为 $O(B)$ 。

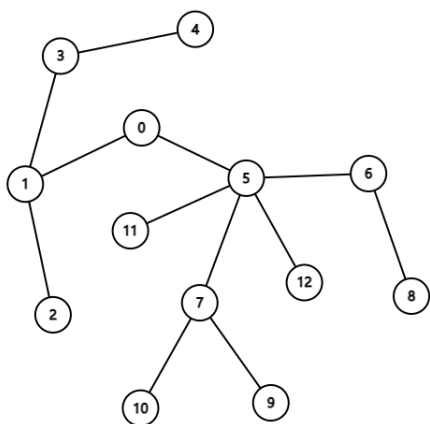


图 3

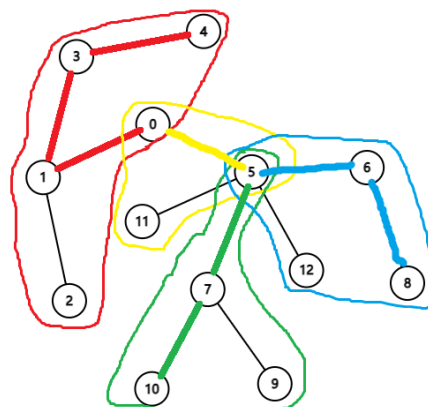


图 4

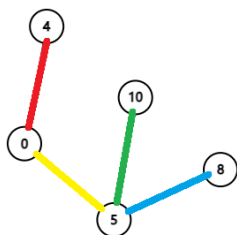


图 5

如图所示, 图 3 为原树。图 4 为对原树的一组划分, 每个簇都用彩线圈了起来, 簇路径用彩线描了一下。图 5 为将每个簇看作一条边后, 所有簇的界点所构成的新树, 之后我们称之为收缩树。

关于这组划分的选取, 在引入 top tree 的概念后, 一个自然的想法是, 先将原树建成一棵深度 $O(\log n)$ 的 top tree, 再在 top tree 的二叉树结构上截取大小 $O(B)$ 的子树。但是由于

²可以在参考文献 [1] 中了解更多

top tree 静态构建算法的复杂性，及其衍生的截取子树的复杂性，这并不是一个好的构造算法。下一小节中我们将介绍一个更简单方便的针对性构造算法。

1.3 一种比较易于实现的静态构造算法

首先，我们任选一个点为根，从该点开始运行 dfs 算法。dfs 过程中，我们需要确定原树的哪些点出现在最终的收缩树中；进而确定收缩树上的每条边包含原树的哪些边，或者说原树中的每条边，属于最终的收缩树的哪个簇。为此，dfs 的过程中我们需要维护一个栈用以存储暂时还未归类的边。

1.3.1 弹栈的过程

当前点 u 要结束 dfs 的时候，有一些栈中的边可能需要弹出来。具体的，出现如下三种情况时需要将栈中的边弹出来进行处理。

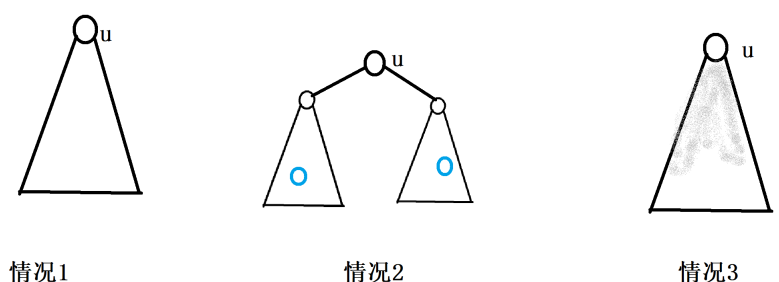


图 6

1. u 是全树的根。方便起见我们强制根节点出现在最终的收缩树中。
2. 存在 u 的至少两个不同子树，里面有已确定的簇的界点（图中蓝点）。由于一个簇至多拥有两个界点，所以此时点 u 不能为内点。
3. 栈中剩余边的数量（图中灰色部分）大于设定的阈值 B 。

至此，我们先分析一下弹栈的三种情况发生次数。情况三只会发生严格不超过 $\frac{n}{B}$ 次，而三种情况中涉及到的点，所构成的树结构中，所有不是根的非叶子节点儿子数量一定大于 1，且叶子节点数量一定不超过情况三发生次数。从而三种情况的总发生次数一定不超过 $2 \cdot \frac{n}{B}$ 。

1.3.2 子树与边的划分

下面要解决的问题是，如何合适地将 u 的不同子树分割为不同的簇，来满足最初的要求。

此时问题可以抽象为，给定一个数 m 以及两个长为 m 的序列 a_1, \dots, a_m 和 b_1, \dots, b_m 。其中 m 是点 u 的儿子数量，

a_i 是点 u 第 i 个儿子的子树中，尚未归类的边的数量。如果情况 3 均被正确地处理，则 a_i 恒不大于阈值 B 。

b_i 为一个布尔值，表示点 u 的第 i 个儿子的子树中是否存在已确定的界点。

我们要做的是，将序列切割为若干子段，使得每个子段中至多有一个 b_i 取值为真，且子段内的 a_i 和不超过 B 。

实际上，每次贪心截取序列的一段极长合法前缀，将其归为同一个簇，所得到分割方案，就是满足算法所要求的复杂度的。

一个小细节是，如果截取的这个子段里 b_i 全为 0，这样得到的簇会只有一个界点，如果想让最后得到的结构符合最初的严谨定义，需要额外地任意选取另一个界点。不过实践中有时可以忽略选取额外界点的步骤。

1.3.3 复杂度证明

首先，这样得到的每个簇大小显然不超过 B 。

接着，我们将前面提到的切割序列前缀，按照终止条件分为三类：

1. a_i 和将大于 B 。
2. 将出现多于一个的 b_i 取值为真。
3. 这次截取完毕后序列将为空。

我们从前往后贪心地将第一类情况与其邻居配对，则每对的 a_i 和一定大于 B ，从而对数不超过 $\frac{n}{B}$ ，从而归于这一部分的簇数量不超过 $2 \cdot \frac{n}{B}$ 。

之后我们仅需考虑第二类情况和第三类情况的次数，而这两者的出现，均与一次弹栈相对应，故而不超过 $4 \cdot \frac{n}{B}$ 。

于是，总的簇数量就不超过 $6 \cdot \frac{n}{B}$ 。由此，我们的算法成功地完成了最初的要求：对于一棵给定的 n 个点的树和一个块大小 B ，要求将原树划分为 $O(\frac{n}{B})$ 个不交簇的并，并且每个簇的大小均为 $O(B)$ 。

1.4 在动态树问题上的推广

本小节主要讨论如何在支持加边和删边的动态树问题上维护树分块的结构。

1.4.1 一种均摊复杂度的实现

每次执行加边或删除边的时候，我们暴力将涉及到的一个或两个簇划分为 $O(1)$ 子簇，以便操作涉及到的点成为界点，这是容易的。之后再处理一下操作边所构成的簇即可。

这样，一次操作后，每个簇的大小都仍然没有超过限制，并且簇的数量仅仅增加了 $O(1)$ 。对此，我们每 $O(\sqrt{n})$ 次操作后重构整棵树的分块结构，即可保证每个时刻的簇数量为 $O(\sqrt{n})$ 。

这样，我们就以单次均摊 $O(\sqrt{n})$ 的复杂度动态维护了树分块结构。这里假定我们需要维护的簇大小为 $O(\sqrt{n})$ ，具体实现的时候可以根据需要调整重构参数。

不过，在一些特殊场合，需要保证单次操作的最坏复杂度，均摊的算法不总是奏效。后文中我们将介绍一种可以保证单次最坏复杂度的算法。

1.4.2 伪簇的概念

在之前的讨论中，我们已经看到了簇概念的局限性。界点数量的限制导致我们无法保证每个块的大小下界，也为我们分析问题增加了困难。所以这里引入了伪簇的概念。

伪簇，定义为树上的一个连通边集。沿用前文中簇的定义，我们称伪簇中与外界相连的点为界点。但是和簇不同的是，伪簇不要求界点数量至多为 2，可以没有上限。

不过伪簇的结构并不利于信息维护，实践中我们可以保留界点及其虚树中的点，将单个伪簇建成点数至多为界点数量两倍的收缩树。由此，对于原树的一个伪簇划分，我们可以建出对应的簇划分。相应的，想要动态维护簇划分，我们只需以正确复杂度维护伪簇划分。伪簇的概念仅仅是为我们动态维护簇划分提供便利。

引理 1.4.1. 对于一棵树，以及一组分为 a 个伪簇的良好伪簇划分，界点数量不超过 a 。这里一组伪簇划分定义为良好的，当且仅当每个界点同时属于至少两个伪簇。

证明. 我们将伪簇记作圆点，界点记作方点。对于每个伪簇对应的圆点，我们向其所包含的界点对应的方点连边，这样可以得到一棵树。将任意一个点提根，则每个方点都有至少一个圆儿子（否则与“每个界点同时属于至少两个伪簇”矛盾）。从而方点数量不超过圆点数量，引理得证。 \square

推论 1.4.1. 对于一棵树，以及一组分为 a 个伪簇的良好伪簇划分，其对应的簇划分界点数量不超过 $2a$ 。

在下文的应用场合中，单个伪簇的大小总是有上界的。所以对于一个伪簇，将其建为收缩树的过程，只需在线性时间内完成即可，而且对每个簇的大小上下界没有额外要求。这样的构建是平凡的。

1.4.3 伪簇的合并与分裂

由于对于界点数量没有要求，所以对于两个有公共界点的伪簇来说，合并他们是平凡的，不像簇的合并受到诸多限制。

在合并过程中，为了保证单个伪簇的大小上界，我们需要进行分裂操作。实际上我们有如下引理：

引理 1.4.2. 对于一个大小为 n 的伪簇，存在算法将其划分为两个伪簇，使得较小的那个大小不小于 $\frac{n}{3}$ 。

证明. 我们将该伪簇的重心提根，记 m 为重心最大子树的大小。则 $m \leq \frac{1}{2}n$ 。

若 $m \geq \frac{1}{3}n$ ，则将重心最大子树分为一个伪簇，其它子树分为另一个即可。

否则我们按任意顺序将重心的子树加入第一个伪簇。当第一个伪簇的大小第一次大于等于 $\frac{1}{3}n$ 的时候，由于 m 的限制，其大小一定也不超过 $\frac{2}{3}n$ 。

此时我们终止加入进程，至此引理得证。 \square

1.4.4 树分块的动态维护

对于给定的森林和阈值 B ，我们可以维护森林的一组伪簇划分，并且每个伪簇均满足如下两种条件之一：

1. 每个伪簇的大小均在 $[0.5B, 2B]$ 之间。
2. 该伪簇所属连通块仅有一个伪簇。

具体的，对于一组合法的伪簇划分，如果我们要进行加边操作，如前所述，合并是平凡的，唯一问题是新生成的伪簇大小可能不合法。如果不合法，新生成的伪簇大小一定在 $(2B, 4B]$ 之间，我们只需对该伪簇执行至多 2 次引理 2.3.2 中提到的分裂算法，即可将其分为多个大小合法的子伪簇。

如果我们要进行删边操作，切割伪簇是平凡的，唯一的问题仍然是新生成的伪簇大小可能不合法。对此，我们暴力枚举新生成的伪簇的邻居伪簇并将其合并，如果合并后的新伪簇大小过大，像加边时一样进行切分即可。

这样，每次只会合并和分裂 $O(1)$ 个伪簇。由之前提到的伪簇划分与簇划分的对应关系与转化方式，我们得到的算法可以以 $O(B)$ 的复杂度动态维护树分块的结构。

2 树分块算法在一些根号复杂度问题上的应用

在当前的 OI 竞赛中，使用序列分块算法的题目常常具有 $O(n\sqrt{n}\text{poly}(\log n))$ 的复杂度。当序列分块推广到树上的时候，自然也会想到以 $O(\sqrt{n}\text{poly}(\log n))$ 作为块大小来处理问题。本节将介绍几道以 $O(\sqrt{n}\text{poly}(\log n))$ 为块大小的例题来帮助读者熟悉树分块的应用方式。

例题 1. 带 $link$, cut 操作的树上第 k 小值相关信息查询³

要求维护一个 n 个点的有根森林，每个点都有一个点权，要求支持 q 次操作，每次操作形如

1. 加入一条边
2. 删除一条边
3. 给定常数 c ，对一条链上的所有点，将其点权加等于 c 。
4. 对于指定的根和给定的常数 c ，对某点子树中的所有点，将其点权加等于 c 。
5. 给定常数 c ，对一条链上的所有点，求出其上点权不超过 c 的点的数量
6. 对于指定的根和给定的常数 c ，对一个子树内的所有点，求出点权不超过 c 的点的数量
7. 给定常数 k ，对一条链上的所有点，求出这些点权的第 k 小值
8. 对于指定的根和给定的常数 k ，对一个子树内的所有点，求出这些点权的第 k 小值

这里认为 n, q 同阶。

关于树结构的维护，如前所述，我们可以以单次 $O(B)$ 的复杂度动态维护块大小为 B 的树分块结构，后面将会分析 B 的最优取值。

关于点权信息的维护，在每个簇中，我们维护簇路径上的点权构成的有序数组，以及所有簇内点的点权构成的有序数组，还有修改操作带来的加法标记。

执行修改操作（同时包括点权修改和树形态修改）时。对于端点所处的簇，我们暴力重构的有序数组。为了避免复杂度增加 \log 因子，我们可以使用归并来代替重新排序。这一步复杂度 $O(B)$ 。对于其它簇，只需修改加法标记。对于涉及到的界点，由于数量有上界，可以暴力处理。这一步复杂度 $O(\frac{n}{B})$ 。

执行查询操作时，对于端点所处的簇和界点，我们可以暴力提取出涉及到的点的点权。这样，问题就变成了，在多个有序数组中计算常数 c 的排名，或者寻找第 k 小值。这两个子问题均可做到 $O(\frac{n}{B} \log n)$ 的复杂度。对于前者，在每个数组中二分查找即可。至于后者的做法，则可以在参考文献 [2] 中查阅。这一步复杂度为 $O(B + \frac{n}{B} \log n)$ 。

综上，取 $B = O(\sqrt{n \log n})$ 时本解法有最优复杂度 $O(n \sqrt{n \log n})$ 。

实际上本题的弱化版已多次在算法竞赛中出现。

Rujia Liu loves Wario Land!⁴ 即为本题去掉操作 2,4,6,7,8 的版本，并且操作 3 有额外的特殊性质。

Gty 的超级妹子树⁵ 即为本题去掉操作 4,5,7,8 的版本，并且操作 1,3 有额外的特殊性质。

³题目来源：经典问题

⁴题目可以在 <https://vjudge.net/problem/UVA-11998> 查看

⁵题目可以在 <https://www.luogu.com.cn/problem/P2166> 查看

带 Link、Cut 树上路径 k 小值⁶ 即为本题去掉操作 5,6 的版本, 并且出题人给出的参考算法复杂度为 $O(n\sqrt{n}\log n)$, 不够优秀。

May Holidays⁷ 即为本题去掉操作 1,2,4,5,7,8 的版本, 并且修改操作 3 中保证常数 $c = 1$, 询问操作 4,5 中保证常数 $c = 0$ 。不过由于修改的特殊性质, 本题可以做到更优复杂度, 在下一个例题中我们将进行一些探讨。

例题 2. *Simple Tree*⁸

给定一棵 n 个点的有根树, 点有点权。

现在有 q 次操作, 操作有 3 种:

1. 给定 x, y, w , 将 x 到 y 的路径上的点点权加上 w (其中 $w \in \{1, -1\}$)
2. 给定 x, y , 询问在 x 到 y 的路径上有多少个点点权 > 0
3. 给定 x , 询问在 x 的子树里的点有多少个点点权 > 0

实际上本题即为 May Holidays 一题增加操作 5 的版本, 并且要求复杂度 $O(n\sqrt{n})$ 。这里认为 n, q 同阶。

我们沿用上一个例题的做法, 不过还需要一些改动。

由于询问操作的特殊性, 我们对于每个有序数组额外维护一个指针指向第一个大于 0 的位置。修改的时候只需移动指针即可。

由于单次移动指针需要做到 $O(1)$ 复杂度, 而重复元素过多会影响暴力移动指针的复杂度。对此, 我们为每个簇的有序数组中的每个元素额外维护一个指针, 表示它加减 1 后指向的新元素位置。

这样, 每次修改操作时我们只需 $O(1)$ 维护有序数组中第一个大于 0 的位置, 即可省去查询时的二分查找, 取 $B = O(\sqrt{n})$ 即可做到 $O(n\sqrt{n})$ 的复杂度, 与出题人的官方做法⁹ 具有相同复杂度。

例题 3. [Ynoi2009] *rpdq*¹⁰ 给定一棵 n 个点的树和 q 次询问。每次询问给出 l, r , 要求回答 $\sum_{i=l}^r \sum_{j=i+1}^r dis(i, j)$, 其中 $dis(a, b)$ 表示编号为 a 和编号为 b 的点在树上的距离。要求复杂度 $O(n\sqrt{n})$, 这里认为 n, q 同阶。

我们将 1 号点提根, 则 $dis(a, b) = dep_a + dep_b - 2dep_{lca(a,b)}$, 这样问题可以转为计算 $\sum_{i=l}^r \sum_{j=i+1}^r dep_{lca(i,j)}$ 。

一个自然的想法是, 运用莫队算法计算上式的值, 将问题转为动态维护一个集合, 支持

⁶题目可以在 <https://immortalco.blog.uoj.ac/blog/670> 查看

⁷题目可以在 <https://codeforces.com/contest/966/problem/E> 查看

⁸题目可以在 <https://uoj.ac/problem/435> 查看

⁹可以在 [这里](#) 查看

¹⁰题目可以在 <https://www.luogu.com.cn/problem/P6778> 查看

1. 加入一个元素
2. 删除一个元素
3. 维护所有元素两两之间 lca 的深度和

这是经典问题，加入/删除元素时我们将该点到根的链上所有点的权都加/减 1，再求出该点到根的链上所有点的权和，即可更新答案。

链加链求和可以通过树链剖分来维护，这样我们就得到了一个 $O(n\sqrt{n}\log^2 n)$ 的做法。使用全局平衡二叉树¹¹可以将复杂度优化至 $O(n\sqrt{n}\log n)$ ，不过仍然不足以通过此题，还需要一些优化。

借助莫队二次离线的方法¹²，我们可以将链加的次数减少到 $O(n)$ 次，但是询问链和的次数仍然是 $O(n\sqrt{n})$ 。

询问和修改次数的不平衡为我们提供了一个切入口。序列问题中我们常常使用序列分块来平衡复杂度，类似的，这里我们可以使用树分块。

具体的，修改一个点后，我们可以在收缩树上暴力修改点权，并进行一次树上前缀和，这样就正确更新了所有界点的答案。同时，对于修改点所属的簇内部，我们也要进行一次树上前缀和，以便更新簇内点之间的贡献。由于原树边和收缩树的边都带权，实现的时候要注意细节。

这样，单次修改复杂度即为 $O(B + \frac{n}{B})$ ，查询复杂度显然可以做到 $O(1)$ 。由此，我们以 $O(n\sqrt{n})$ 复杂度解决了此题。

例题 4. [Ynoi2018] 馱作¹³

给定一棵 n 个点的树和 q 次询问。

每次询问给出参数 p_0, d_0, p_1, d_1 ，记 $S_i = \{u | \text{dis}(u, p_i) \leq d_i\}$ ，其中 $i \in \{0, 1\}$ 。

要求回答 $\sum_{a \in S_0} \sum_{b \in S_1} \text{dis}(a, b)$ ，其中 $\text{dis}(a, b)$ 表示编号为 a 和编号为 b 的点在树上的距离。

要求复杂度 $O(n\sqrt{n})$ ，这里认为 n, q 同阶。

本题处理起来比较复杂，我们先从一些相对简单的子问题入手。

子问题 1: 所有询问满足 $S_0 \cap S_1 = \emptyset$

解法 则存在一个点 c ，使得对于 $\forall a \in S_0, b \in S_1$ ， a, b 间的路径经过 c ，故而只需统计 S_0, S_1 的点数以及到 c 的距离和。

子问题 2: $q = 1$

解法 仿照上一道例题，我们将 $\text{dis}(a, b)$ 表示为 $\text{dep}_a + \text{dep}_b - 2\text{dep}_{\text{lca}(a, b)}$ 。

¹¹可以在参考文献 [3] 中查阅

¹²可以在参考文献 [2] 中查阅

¹³题目可以在 <https://www.luogu.com.cn/problem/P5399> 查看

这样我们只需求求两两点间 lca 的深度和。将所有 $a \in S_0$ 到根的路径上边权加上 1，求所有 $b \in S_1$ 到根的路径的边权和。时间复杂度 $O(n)$ 。

子问题 3: 所有询问中的 p_0, p_1 均相等，但是 d_0, d_1 可能不同

解法 枚举 d_0 的取值，类似情况 2 处理，处理出所有可能的询问的答案。之后还需要进行一次二维前缀和，时间复杂度 $O(n^2)$ 。

在解决完子问题后，我们会到原题。设定阈值 B 后，我们使用 2.3 中提到的构建算法求出原树的一组簇划分。

则原树的一个邻域¹⁴ 可以拆成每个簇中的邻域，且除了中心所在的簇，其余簇的邻域均以簇的界点为中心。

每次询问时，对于不同块中的邻域之间的距离和，可以在收缩树上进行树形 DP 统计，类似于情况 1。每次询问需要 $O(\frac{n}{B})$ 的时间。

对于同个块内两个以界点为中心的邻域间的距离和，每次询问在每个块中产生至多 1 个询问，可以最后离线计算，类似于情况 3。每个块需要 $O(B^2)$ 时间。

对于同个块内的两个邻域，如果至少一个邻域的中心不在界点上的情况，每次询问至多出现 2 次，类似于情况 2 处理。每次询问需要 $O(B)$ 时间。

这样总复杂度为 $O(nB + \frac{n^2}{B})$ ，取 $B = \sqrt{n}$ 时有最优复杂度 $O(n\sqrt{n})$ 。

点评 本题所涉及的信息询问结构非常复杂，但最后借助树分块对于树的结构分解，我们还是较好地解决了本题，体现了基于 top cluster 剖分的树分块算法的优势。特别是子问题 3 中，如果不是簇划分强制了界点数量至多为 2，那么进行前缀和运算的维数可能就不只 2 了，这也体现了伪簇划分在信息维护上的缺点。

例题 5. [Ynoi2014] 等这场战争结束之后¹⁵

给定一张 n 个点的图，每个点有点权，最开始没有边。之后需要进行 q 次操作，每次形如：

1. 添加一条 x 与 y 之间的双向边。
2. 回到第 x 次操作后的状态（注意这里的 x 可以是 0，即回到初始状态）。
3. 查询 x 所在联通块中点权第 y 小的值。

这里认为 n, q 同阶。

本题做法比较多，这里仅介绍一种常数比较优秀的 $O(n\sqrt{n})$ 做法。

首先我们可以使用离线算法，对于所有询问按照一定顺序建成操作树。

¹⁴对于点集 S ，如果存在点 p 和距离 d 使得 $S = \{x | \text{dis}(x, p) \leq d\}$ ，则称 S 为原树的一个邻域

¹⁵题目可以在 <https://www.luogu.com.cn/problem/P5064> 查看

然后在操作树上 dfs，同时维护并查集用于判断每次加边操作是否有效（ x 和 y 是否属于同一个连通块）。由于需要支持回溯操作，所以并查集需要按秩合并，这一步复杂度 $O(n \log n)$ 。

通过离散化的方法可以将点权值域压缩为 $O(n)$ 。

我们设定阈值 $B = \sqrt{n}$ ，将值域切成 \sqrt{n} 块，并对每组询问求出答案所属的值域块。具体的，我们需要对每个值域块都在操作树上进行一次 dfs，同时维护每个点所属连通块中，属于该值域块的点数。这一步复杂度为 $O(n \sqrt{n})$ 。

求出答案所属值域块后，我们还需要进一步求出答案的精确值。对此，我们以 $B = \sqrt{n}$ 为阈值将操作树进行分块。对于每个界点，我们可以 $O(n)$ 处理出该时刻整张图的连通信息，再以该界点为基础回答相邻簇的询问。

具体的，对于相邻簇的所有内点，其到界点的路径上至多只有 $O(\sqrt{n})$ 次加边操作，我们将这些加边操作涉及到的点取出来执行 bfs，就可以以 $O(\sqrt{n})$ 复杂度得到该内点处的每个点所属的连通块，之后再从小到大枚举值域块中的元素计算答案即可。

最后时间复杂度为 $O(n \sqrt{n})$ ，空间复杂度为 $O(n)$ ，常数较小。

点评 先前大家对定期重构的认识大多停留在序列或时间轴的线性结构上。这里借助树分块的技术将定期重构推广到了树上，从而得到了一个复杂度和常数都比之前算法更优秀的做法。这种推广的思路值得借鉴。

3 树分块算法在一些其它问题上的应用

序列分块的块大小并不总是 $O(\sqrt{n} \text{poly}(\log n))$ ，在一些特殊问题上特殊的块大小常常能出奇制胜，树分块也是。这里将通过一些例题来进行介绍。

例题 6. 线性树上并查集¹⁶

树上并查集是并查集的一个特殊情况：给定一棵树，每次操作形如将一个节点合并到父亲，每次询问形如查询一个点已合并的祖先。这里要求复杂度线性。

朴素的并查集算法¹⁷复杂度会带上 $\alpha(n)$ 的因子，不能满足我们的需求。

首先，我们以 $B = \lceil \log n \rceil$ 为阈值将原树分块。

对于每个簇内部，我们沿 dfn 序递推求出每个点到簇界点的点集，以一个 B 位的整数形式进行存储。为了后文处理方便，这里需要保证点按深度顺序递增存储。

修改的时候，对于每个簇我们维护已被合并至父亲的点集，同样以一个 B 位的整数来存储。对于收缩树，由于点数只有 $O(\frac{n}{B})$ ，我们可以直接维护并查集。这部分复杂度为 $O(\frac{n}{B} \alpha(n) + q) = O(n + q)$ 。

¹⁶ 题目来源：经典问题

¹⁷ 可以在参考文献 [4] 的第 21 章查阅

查询的时候，如果当前点尚未与所处簇的界点合并，则只需求出该点簇内祖先与簇内未合并点的交集中深度最大的点。这可以通过整数的按位与和 `hightbit` 查询来解决。由于所有涉及到的整数都不超过 $2^B = O(n)$ ，所以 `hightbit` 可以提前预处理。如果当前点已与所处簇的界点合并，则在收缩树的并查集上查询即可。

最后总复杂度为 $O(n + q)$ ，达到了线性的目标。

例题 7. 「十二省联考 2019」希望¹⁸ 给定一棵 n 个点的树，每条边长度均为 1 。求有多少个连通块满足，存在一个点，使得该连通块内的所有点到该点距离均不超过 L 。复杂度要求 $O(n)$ 。

记全集 $A = \{1, 2, \dots, n\}$ 。定义 $f(x, S)$ 为一个数组，其中 x 是原树中的一个点， S 是一个点集， $f(x, S)_i$ 定义为 $\{|T|T \text{ 是连通点集}, x \in T, T \subset S, \forall y \in T \text{ 有 } \text{dis}(y, x) \leq i\}$ 。

通过简单的容斥技巧，可以将问题转为对于所有有序二元组 (x, y) ，其中 x 和 y 在原树中相邻，计算 $f(x, A \setminus y)_{L-1}$ 。

对此，我们以 L 为阈值将原树分块，对于第 i 个簇，我们记簇的内点集合为 I_i ，通过长链剖分优化树上 DP，可以在 $O(L)$ 的复杂度内对簇的两个界点 b_0, b_1 求出 $f(b_0, I_i \cup \{b_0, b_1\})$ 和 $f(b_1, I_i \cup \{b_0, b_1\})$ 。由于与本文主题无关，长链剖分优化 DP 的具体细节就不在这里详细展开了。

通过收缩树上的简单 DP，我们可以对簇 i 的两个界点 b_0, b_1 ，求出 $f(b_0, T \setminus I_i)$ 和 $f(b_1, T \setminus I_i)$ 。

具体实现的时候，建议按照第一节中提到的簇的两种合并方式，`rake` 和 `compress`，相应地封装 f 数组的合并方式。

这里涉及到的关于 f 数组的运算均可在 $O(L)$ 的复杂度内完成，而总运算次数为 $O(\frac{n}{L})$ ，所以这部分总复杂度为 $O(n)$ 。

最后再通过一些简单的计算即可求出每个有序二元组的权值，进而求得答案。最后总复杂度 $O(n)$ 。

点评 本题的官方做法是做两遍长链剖分，一遍自底向上，一遍自顶向下，细节繁多且需要计算模意义下的乘法逆元。相比较而言本做法理解起来细节量更少，而且无需求逆，天然可推广至模数为合数的情况。

总结

在目前国内的 OI 竞赛中，出现过的树分块相关问题数量偏少，这与之前已有的树分块算法结构繁复、适用面窄有关。对此本文第一节引入 `top cluster` 的概念，介绍了一种新的对

¹⁸题目可以在 <https://loj.ac/p/3053> 查看

树分块思路，并针对静态问题和动态问题分别提供了一种构造维护树分块形态的算法，希望能够增加树分块算法的普及程度。

先前 OI 中的树分块相关问题多是借助通用化的手段，用序列问题的手法来处理。对此本文第二节通过数个例题展示了这种树分块算法在一些相关的问题上的优越性。其中例题 1 和例题 2 来源于对之前一系列类似问题的总结归纳。相较于先前依赖弱化问题具体性质的解法而言，这里给出的算法通用性更强，流程上也更加清晰。例题 3,4,5 则展示了在一些其它问题中这类树分块算法的应用。

本文第三节则列举了两个使用了非常规块大小的例题，警示我们有些时候除了分块之后的流程，分块大小的选取本身也是重要的，要具体问题具体分析。不过非常规大小树分块在当前 OI 竞赛中的出现频率还非常低，其应用还有待进一步发掘。

希望本文能起到抛砖引玉的作用，对 OI 中相关的树上的问题产生更多的启发。

感谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队高闻远教练的指导。

感谢父母对我的培养和教育。

感谢学校的栽培，李建老师和施琴儿老师的教导和同学们的帮助。

感谢李欣隆学长、蔡承泽学长与我交流讨论、给我启发。

感谢胡杨同学、吴与伦同学为本文审稿。

感谢其它所有本文所参考过的资料的提供者。

感谢各位百忙之中抽出宝贵的时间来阅读本文。

参考文献

- [1] 赵雨扬.《Top tree 相关东西的理论、用法和实现》, 2019, <https://negiizhao.blog.uoj.ac/blog/4912>
- [2] 李欣隆, 蔡承泽.《信息学竞赛中的抽象数据结构》. 2021.
- [3] 张哲宇.《浅谈树上分治算法》, IOI2019 中国国家队候选队员论文集。
- [4] Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein. 算法导论 [M]. 第 3 版. 机械工业出版社, 2009.

浅谈一类树上路径相关问题

杭州学军中学 周镇东

摘要

本文对一类树上路径相关问题做了总结，并罗列了许多相关例题。

在前言中，笔者大致描述了这类问题的特征。

在第二节中，介绍了解决此类问题常用的三种树上算法，以及这些算法起到的部分作用。

在第三节中，笔者给出了一道例题，并依次采用了第二节中提到的三种算法解决了这个问题。

在第四节中，笔者罗列了近年来的 NOI、WC、CTSC 等重要比赛的许多真题，并应用第二节提到的算法解决问题。

1 前言

近年来，一类树上路径相关的问题经常出现在各大比赛中，这类问题常具有以下特点：

- 给定一棵或多棵树
- 可能给定一些点对或一些路径
- 以某种给定的方式，求树上某些路径或点对的信息最值或求和

解决这类问题也存在一些常用的算法，例如树分治、虚树、线段树合并。

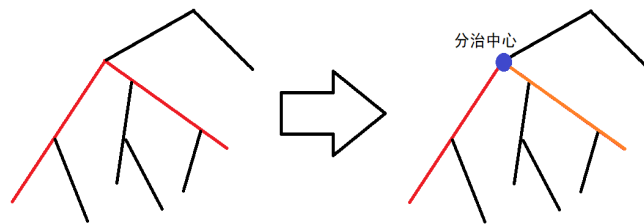
在本文中，将结合具体例题，总结出处理这些问题的常用方法。

如果没有特殊说明，在接下来的问题中，我们总是认为树的规模和询问次数（如果存在多次询问）的大小为 $O(n)$ 。

2 经典树上算法与其效果

2.1 树分治

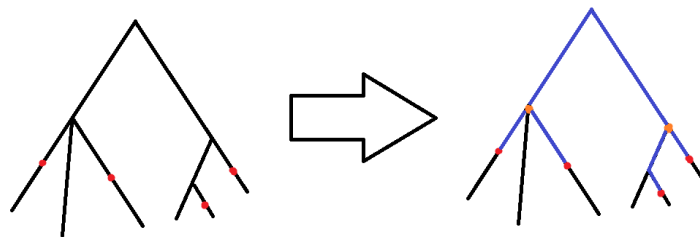
对于一棵树，以及树上的一些链，如果我们要计算的信息是可合并的，那么我们可以通过点分治或边分治将任意链信息拆分成两个从某个分治中心出发的前缀链信息。



因为这么做的代价往往只是在单次处理的时间复杂度的基础上乘上 $O(\log n)$ ，所以对于很多问题，我们都可以先考虑通过树分治尝试将问题简化，得出 $O(n \cdot \text{poly}(\log n))$ 的算法之后再考虑优化。

2.2 虚树

对于一个点集，保留原树上所有两侧均有该点集中的点的边后，删除孤立点，不断收缩所有二度点之后得到的树结构即虚树。因为该点集对应的虚树大小不超过点集大小的两倍，并包含了该点集中所有点，且有树结构，所以对于零散的点集，我们往往可以对其建虚树后再应用一般的树上算法。



有些时候，点集并不是直接给出的。例如，下面列举了两种虚树的使用方法：

- 有两棵树，对第一棵树做树分治，每次得到的点集对应到第二棵树上建虚树。

- 有一棵树及 $O(n)$ 条给定的路径，枚举路径最近公共祖先，并对同一最近公共祖先的所有路径端点建虚树。

在后面的例题中，这两种方法都会被用到。

2.3 线段树合并

对于一棵规模为 $O(n)$ 的树，若每个点初始有一个键值和一些对应的信息。若我们要不断去除某个叶子，并将该叶子的所有信息在不改变键值集的情况下以某种方式合并至父亲，并获取键值在某一个范围内的元素的信息或本次发生合并的点对信息，就可以使用线段树合并的方式实现。

如果按照树上深度优先遍历退出节点的顺序进行合并，线段树合并还能起到固定最近公共祖先的作用。这意味着，当我们依次把某个节点的所有子树信息合并到这个节点上时，每次合并时两个集合之间的点对的最近公共祖先都是这个节点，而线段树合并时还可以得出合并的两个集合中的点对的某些信息，这使得在某些情况下，我们可以把树上路径拆成三个点——两个端点、最近公共祖先来考虑。

若合并信息的复杂度为 $O(1)$ ，线段树合并的总时间复杂度一般仅为 $O(n \log n)$ ，而且应用灵活、容易实现，是非常实用的一种工具。

例如，在 NOI2020 中，“命运”¹ 这一题就考察了线段树合并的应用。

3 典型例题

Maximum and Minimum²

3.1 题目大意

对于一个有边权无向连通图 G 和两个节点 x, y ，我们定义 $f(G, x, y)$ 为图 G 中点 x 与点 y 间所有路径的权重的最小值。其中，一条路径的权值为路径上所有边权的最大值。

给定两个无向图 G_1, G_2 ，求 $\sum_{i=1}^n \sum_{j=i+1}^n f(G_1, i, j) f(G_2, i, j)$ 。

3.2 解题方法

首先，根据 f 函数的定义，我们可以只保留 G_1, G_2 最小生成树上的边，于是问题转化到了两棵树上。

¹供参考的题面链接：<https://uoj.ac/problem/559>

²<https://www.codechef.com/problems/MXMN>

3.2.1 树分治

容易发现，当 G 为一棵树时， $f(G, x, y)$ 就是求 x 至 y 的路径边权最大值，最大值是可合并的信息，所以我们可以先用边分治尝试一下。

对第一棵树进行边分治后，在第一棵树上，得到了由分治中心切割分治区域得出的两部分点集，我们尝试在这一次划分的过程中处理完所有横跨这两个点集的对。

我们设分治中心到点 x 的前缀边权 \max 为 w_x ，那么横跨两个点集的对 (x, y) 给答案的贡献即可被改写为 $\max(w_x, w_y)f(G_2, x, y)$ 。

3.2.2 虚树

我们将点集取出，并在第二棵树上建出虚树。

接下来考虑按照边权从小到大枚举虚树上的边，每次处理在第二棵树上所有以当前边为最大值的点对，即，按照权值顺序依次加入边，每次将边所在两侧集合合并。

按照加边顺序，我们可以得到一棵 Kruskal 重构树。

3.2.3 线段树合并

现在，有一个树形结构——Kruskal 重构树，且每个点有一个额外的权值——点 x 的权值为 w_x 。由于每次合并时的 $f(G_2, x, y)$ 被固定了，我们只需要对于所有满足以下条件的点对 (x, y) ，求出 $\max(w_x, w_y)$ 之和即可：

- x 和 y 处于分治中心两侧
- x 和 y 在本次合并之前处于不同集合

于是，对于每个集合，我们可以以权值为下标，维护区间元素个数和权值和。这样，我们在合并两个线段树的对应节点时，因为权值较大的处于右侧子树，所以只需要把左侧的元素个数乘上右侧权值和加入贡献即可。

3.2.4 时间复杂度

对于一个大小为 s 的点集的单次求解的时间复杂度为 $O(s \log s)$ ，由于我们在这以外还使用了边分治，所以总时间复杂度为 $O(n \log^2 n)$ 。³

³本题有树上启发式合并解法，但是与本文主题无关，故不作赘述

3.3 拓展

3.3.1 问题一

- 给定两棵树 T_1, T_2 ，定义 $g(T, x, y)$ 表示路径边权最大值与路径边权积的乘积。求

$$\sum_{i=1}^n \sum_{j=1}^n g(T_1, i, j) f(T_2, i, j)$$

解法显而易见。在边分治后，设点 x 到边分中心的边权乘积为 s_x ，原问题线段树维护了以节点的 w 值为下标的区间 w 值之和，现将其改为维护以节点的 w 值为下标的区间 $w \cdot s$ 值之和即可。

3.3.2 问题二

- 给定两棵树 T_1, T_2 ，求 $\sum_{i=1}^n \sum_{j=1}^n g(T_1, i, j) g(T_2, i, j)$ 。

由于第二棵树的路径边权乘积也被计入，所以 Kruskal 重构树和线段树合并的方式不再适用。

我们可以再次在虚树上使用边分治，于是该问题可以被化简成一个简单的可以使用线段树等数据结构解决的问题。

总时间复杂度为 $O(n \log^3 n)$ 。

虽然这个时间复杂度不如拓展问题一解法的时间复杂度，但是我们至少得出了一个 $O(n \cdot \text{poly}(\log n))$ 的算法。

3.4 小结

在这个问题中，我们多次运用了边分治化简题目，每一次边分治，都将一棵树以及边权信息转化成了每个点的一个权值。注意到，在原问题中，边分治后问题转化成一棵树和一些权值；在拓展问题二中，两次边分治之后，两棵树被转化成了每个点的若干个权值。

这说明，我们往往可以通过一次或多次点分治来很好的达到化简题目的效果。

4 更多应用

4.1 例 1

通道⁴

⁴来自 WC2018，参考题面链接：<https://uoj.ac/problem/347>

4.1.1 题目大意

给定三棵有非负边权的树，定义一棵树上两点距离为两点间最短路径边权和。问一对点在三棵树上的距离和的最大值为多少。

4.1.2 算法一

首先考虑在第一棵树上边分治，对题目进行简化。在第二棵树上建出虚树之后再次边分治简化题目。然后在第三棵树上再建虚树、再次边分治。

于是每个点获得了一个权值，代表它到三个分治中心的距离之和，而且我们知道每个点在三棵树上边分治时在哪一侧。

我们枚举其中一个点分别位于三次分治的哪一侧，另一个点得分别位于三次分治的另一侧，这种情况下的最优解即两边可能的点的权值最大值的乘积。

至此，我们通过不断边分治，非常快速且容易地得到了一个时间复杂度为 $O(n \log^3 n)$ 的算法。

4.1.3 算法二

考虑不做第三次分治，并对第三棵树做一些修改。

具体地，假设点 x 在前两棵树中与分治中心的距离之和为 d_x ，那么在第三棵树上，对于每个点 x ，我们新建一个点 x' ，并在 x 与 x' 之间连一条权值为 d_x 的边。

于是问题被转化成求“所有在前两棵树中均处于分治中心两侧的对”在第三棵树上的最远距离。

因为两个点集的最远点对之一的两 endpoint 一定分别处于这两个点集各自的最远点对中，所以我们可以轻松解决这部分求最远距离的问题。

因此，时间复杂度被降到了 $O(n \log^2 n)$ 。

4.1.4 算法三

考虑在算法二的基础上，不对第二棵树进行树分治。

在对第一棵树的分治过程中，我们将点集内的点分成了两种，这两种点分别位于分治中心两侧。

设 $d_k(x)$ 表示节点 x 在第 k 棵树中的深度，设 $dis_k(x, y)$ 表示点对 (x, y) 在第 k 棵树上的距离，设 $D(x)$ 表示节点 x 在第一棵树中与当前分治中心的距离。

设点对 (x, y) 在第二棵树上的最近公共祖先为 p ，则 x 与 y 在三棵树中的距离之和可以写为 $D(x) + D(y) + d_2(x) + d_2(y) - 2d_2(p) + dis_3(x, y)$ ，稍加整理得到

$$(D(x) + d_2(x)) + (D(y) + d_2(y)) + dis_3(x, y) - 2d_2(p),$$

也就是说，如果固定了 p ，剩下的问题仍然可以转化为在第三棵树上的最远点对问题。

注意到点集的最远点对是支持快速合并的，也就是说我们只需要在第二棵树中进行一次深度优先遍历，并不断将子树的点集向上合并，维护点集在第三棵树上的最远点对，计算合并时产生的新的不同种类点之间的最远点对。以这种方式合并时得到的新点对在第二棵树上的最近公共祖先是已知的，所以我们可以根据上式更新答案。

由于单次求点对距离的时间复杂度可以通过预处理降至 $O(1)$ ，且虚树复杂度瓶颈——按照 dfs 序排序部分可以在边分治处理子问题结束之后由子问题的结果向上使用归并的方式快速得到，所以我们得到了一个时间复杂度为 $O(n \log n)$ 的算法。

4.1.5 小结

在解决这个问题过程中，我们先通过多次边分治的方式，非常容易地得到了 $O(n \cdot \text{poly}(\log n))$ 的算法，在此基础上，再逐步优化，最终得到了优秀的解法。

这再次告诉我们遇到这类问题时，先尝试树分治往往能轻松地简化题目，并具有较强的启发意义。

4.2 例题 2

暴力写挂⁵

4.2.1 题目大意

给定两棵 n 个节点、边有权的树，定义 $depth_k(x)$ 为第 k 棵树上点 1 到点 x 的距离，定义 $LCA_k(x, y)$ 表示第 k 棵树上点 x 和点 y 的最近公共祖先，定义点对 (x, y) 间的“距离”为

$$depth_1(x) + depth_1(y) - depth_1(LCA_1(x, y)) - depth_2(LCA_2(x, y))$$

求“距离”值最大的点对的“距离”值。

4.2.2 简单转化

定义 $dis_k(x, y)$ 表示点对 (x, y) 在第 k 棵树上的距离。

则原“距离”式可以转化为

$$\frac{1}{2}dis_1(x, y) + \frac{1}{2}depth_1(x) + \frac{1}{2}depth_1(y) - depth_2(LCA_2(x, y))$$

如果边权均非负，那么最远点对信息可以合并，我们可以采用类似于通道这一题中的方法来解决。

⁵来自 CSTC2018，参考题面链接：<https://uoj.ac/problem/400>

但是这里边权可能为负数。

接下来，这题有两个思路：

- 根据之前的经验，我们先对第一棵树进行边分治。
- 在第二棵树上，最近公共祖先似乎占据了重要的位置，这让我们不禁想起了线段树合并。

4.2.3 算法一

首先，我们对第一棵树进行边分治，假设第一棵树上节点 x 到当前分治中心的距离为 $D(x)$ ，则原式可以进一步转化为

$$\frac{1}{2}(\text{depth}_1(x) + D(x) + \text{depth}_1(y) + D(y)) - \text{depth}_2(\text{LCA}_2(x, y))$$

直接在第二棵树上建出虚树，然后在树上 dfs，并记录子树内节点 $\text{depth}_1(x) + D(x)$ 的最大值，在合并子树的时候更新答案即可。

时间复杂度瓶颈在于建虚树前的按照 dfs 序排序这一部分，而这一部分可以用在通道一题中使用的归并方式快速解决。

于是我们轻松地得到了一个 $O(n \log n)$ 的算法。

4.2.4 算法二

我们考虑在第二棵树上进行深度优先遍历，并不断合并子树信息。

注意到，边分树也是一个深度为 $O(\log n)$ 的二叉树，和线段树非常相似。如果把边分树看做线段树，我们可以在上面进行类似的合并操作——我们对第一棵树建立边分树，并记录每个分治中心到对应范围内每个点的距离。由于单次合并前第二棵树上的最近公共祖先已经确定，我们只需要在边分树结构里记录每一个边分中心每一侧节点的 $\text{depth}_1(x) + D(x)$ 的最大值，在合并的时候更新答案即可。

时间复杂度仍然是 $O(n \log n)$ 。

4.3 例 3

情报中心⁶

⁶来自 NOI2018，参考题面链接：<https://uoj.ac/problem/397>

4.3.1 题目大意

给定一棵 n 个节点的树，边有权。

给定 m 条路径，每条路径有一个权重。

对于所有至少有一条边相交的路径对，求它们路径并所有边权之和减去这两条路径各自的权值得到的结果的最大值。

或者判定没有合法的路径对。

边权和路径权值非负。

4.3.2 对边分治效果的分析

由于边分治的过程中，“有交路径对”这个关键限制并没有得到化简，故这题中不适合在第一步进行边分治。

4.3.3 算法一

我们称一条路径的 LCA 为该路径两端点的最近公共祖先。

按照路径相交的方式，我们将相交路径对分成以下两类讨论：

- 两条路径的 LCA 不同
- 两条路径的 LCA 相同

设第 i 条路径的两端点为 x_i, y_i ，其 LCA 为 p_i ，权值为 v_i 。

设点 x 和点 y 的最近公共祖先为 $LCA(x, y)$ 。

设原树上两点 (x, y) 间距离为 $dis(x, y)$ ，一个点 x 的深度为 d_x 。

考虑第 i 条路径和第 j 条路径，如果 $p_i \neq p_j$ ，则两条路径的交必然是一条其中一端点是另一点祖先的链，我们不妨设 x_i, x_j 均位于这条链底端节点的子树中，则这两条路径的结果为

$$(dis(x_i, y_i) - v_i) + (dis(x_j, y_j) - v_j) - d_{LCA(x_i, x_j)} + \max(d_{p_i}, d_{p_j})$$

若我们把 $(dis(x_i, y_i) - v_i)$ 看做路径 i 的新权值，把 d_{p_i} 看做路径 i 的键值，再设法固定 $LCA(x_i, x_j)$ ，就可以发现线段树合并能解决这个问题。

如果 $p_i = p_j$ ，我们不妨先枚举 LCA，并对同一 LCA 的所有路径端点建虚树，将问题等价转化成了 $p_i = p_j = 1$ 的版本。

由于这两条路径有交，所以节点 1 必然存在一个子树，满足这两条路径恰好各有一个端点在这个子树内，且其他的端点都在这个子树外，我们不妨设在这个子树内的点分别为 x_i, x_j 。

则这两条路径的权值为

$$\frac{1}{2}((dis(x_i, y_i) - 2v_i + d_{x_i}) + (dis(x_j, y_j) - 2v_j + d_{x_j}) - 2d_{LCA(x_i, x_j)} + dis(y_i, y_j))$$

我们可以进行树形 DP，一侧用于固定 $LCA(x_i, x_j)$ ，另一侧维护与 y_i, y_j 相关的最远点对。

至此，这两部分都已经解决，时间复杂度为 $O(n + m \log n)$ 。

4.3.4 小结

在第一种情况中，我们发现了线段树合并的经典形式。

在第二种情况中，我们采用了枚举 LCA 并建虚树的方式有效地化简了题目。这种方法非常常用，例如对于 ZJOI2019 中的语言⁷ 一题，也可以使用类似的思路。

5 总结

通过上述例题，可以体会到树分治和虚树在简化树上问题时起到的巨大作用，而线段树合并往往解决了问题的最后一步。

在处理这些问题时，我们往往可以先尝试使用树分治或者虚树简化问题，然后尝试将问题转化成线段树合并或者其他算法可以解决的问题。

这些树上算法运用灵活，效果显著，可以处理很多问题。在本文中，笔者挖开了这类问题的冰山一角，希望借此帮助大家熟练掌握此类问题，也希望有更多更加巧妙的相关的问题出现。

感谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队高闻远教练的指导。

感谢徐先友老师对我的培养与指导。

感谢周航锐同学为本文审稿。

感谢父母对我的理解与支持。

参考文献

[1] 张哲宇. 浅谈树上分治算法. IOI2019 中国国家候选队论文集, 2019

⁷<https://uoj.ac/problem/470>

[2] 陈俊锟, 吕欣. 《通道》试题讲评. 2018

[3] 陈俊锟, 吕欣. 《暴力写挂》试题讲评. 2018

[4] 陈俊锟, 吕欣, 杨景钦, 王逸松. 《情报中心》试题讲评. 2018