



IOI2022国家集训队 论文集

中国计算机学会
2022年2月



2022 年信息学奥林匹克中国国家集训队论文集

目 录

浅谈一类特殊 DAG 上的路径信息维护问题 (戴江齐)	1
更一般的 DFT (周航锐)	13
浅谈一种基于比较的贪心思想在信息学竞赛中的应用 (张隽恺)	28
浅谈平面图相关算法 (唐绍轩)	49
浅谈求交互参数类交互问题 (胡昊)	72
浅谈 OI 中的近似算法 (许庭强)	82
图染色问题初探 (彭博)	99
浅谈信息学竞赛中的空间优化问题 (陈知轩)	118
浅谈一类蒙日矩阵 (张景行)	129
浅谈一类哈希表的复杂度分析 (陈于思)	145
线性代数在 OI 中的应用 (常瑞年)	159
浅谈与 Lyndon 理论有关的字符串组合问题 (万成章)	182
《可爱多的字符串》命题报告 (冯施源)	204
浅谈 OI 中的模拟费用流问题 (陈扩文)	213



浅谈一类特殊 DAG 上的路径信息维护问题

南京外国语学校 戴江齐

摘要

在 OI 中,不少和数据结构有关的问题都可以看作 DAG¹ 上的路径问题。本文介绍了三种可以在 $O(n\text{poly}(\log S))$ 时间内维护 DAG 路径信息的算法(其中 n 为输入规模, S 为 DAG 的路径总数, $\text{poly}(x)$ 表示关于 x 的多项式)并探索了这些算法在数据结构、字符串和数论等方面的应用。

引言

在 OI 中,很多问题都可以归约到和 DAG 路径相关的模型上,包括但不限于图论、数据结构和字符串问题。在一般的 DAG 上快速维护路径信息是不现实的,这是因为描述一条路径就需要 $\Omega(n)$ 个 bit; 因此,本文着眼于路径总数较少的 DAG (有不少较为实用的 DAG 模型满足这一性质,如后缀自动机)。本文主要介绍了可以以 $O(\text{polylog} S)$ (其中 S 为路径总数) 时间进行 DAG 路径信息维护的三种算法: DAG 链剖分、可持久化平衡树、平衡树分裂归并。其中, DAG 链剖分的实现较为简洁,且修改操作比较容易,是 OI 中相对更实用的算法;可持久化平衡树拥有最优的时间复杂度和较强的可拓展性,但空间复杂度和实现难度较高;平衡树分裂归并的时间复杂度仍是 open 的(目前仅有 $O(\log S \log q)$ 的上界证明但无法构造出使上界紧的数据),但由于它修改和查询的关系与前两种算法相反(这意味着查询是离线的而 DAG 可以强制在线给出),也有一些难以替代的应用。

本文第一节介绍了 DAG 路径信息维护的基本模型;第二节依次介绍了 DAG 链剖分、可持久化平衡树和平衡树分裂归并三种算法;第三节介绍了这些算法在一些场景下的应用。

1 基本模型

给定一张 n 个点的 DAG (每个点的出边有顺序),将结点 $i(1 \leq i \leq n)$ 指向的结点依次排列,记为序列 G_i 。结点 i 有正整数权值 c_i , 它的第 j 条出边有边权 j 。

¹DAG 即有向无环图,为 Directed Acyclic Graph 的缩写

将图上所有起点为 1、终点出度为 0 的路径按照边权的字典序排成一排（容易发现排列方式唯一），记为序列 $P_i (1 \leq i \leq S)$ （其中 S 为满足条件的路径总数）。

现有 q 次查询，每次给定正整数 k ，求 P_k 的所有结点权值和。

保证 DAG 的边数不超过 $2n$ ， $1 \leq n \leq 10^5$ ， $1 \leq k \leq 10^{18}$ ， $1 \leq c_i \leq 10^9$ 。

2 解法

由于 $k \leq 10^{18}$ ，只需保留 DAG 的前 10^{18} 条路径。事实上，可以通过 $O(n)$ 的 DP 求出每个点出发到零出度点的路径个数（并和 10^{18} 取 $\min$ ），然后仅保留 DP 值小于 10^{18} 的点；此外，还需复制字典序第 10^{18} 小的路径上经过的点，并对它们的出边分别保留一个前缀指向前一类点。这样，只需考虑 $S \leq 10^{18}$ 的情况。

通过二度化，容易发现只需考虑每个点的出度是 0 或 2 的情形。

以下分析中认为 $S \geq n$ 。

2.1 DAG 链剖分

若 DAG 是一棵有向树（或森林，即每个点入度不超过 1），则显然可以使用重链剖分维护链上信息。

我们将这个做法推广到更一般的 DAG 上：对每个出度非零的结点 i ，记以它为起点的所有路径个数为 f_i ，并定义它的重后继 $next_i$ 为 G_i 中 f 值最大的结点（若有多个则任取一个）。类似地，记 h_i 为以 i 为终点的路径个数，并将指向 i 的结点中 h 值最大的记为 i 的重前驱 pre_i 。

称边 $u \rightarrow v$ 为重边当且仅当 $v = next_u$ 且 $u = pre_v$ ；我们仅保留重边，则剩余的图由若干条不交的链（记为重链）构成。

考虑 DAG 上的任一路径 $p_1 \rightarrow p_2 \rightarrow \dots \rightarrow p_k$ 。显然 f_{p_i} 递减， h_{p_i} 递增；而对于每条不是重边的边 $p_i \rightarrow p_{i+1}$ ， $f_{p_i} > 2f_{p_{i+1}}$ 和 $2h_{p_i} < h_{p_{i+1}}$ 中至少有一条成立，故路径上至多有 $O(\log S)$ 条轻边。

因此，我们可以从 1 出发，每次二分出被查询的路径和当前重链的公共部分，再对重链预处理前缀和，即可在 $O(n + q \log S \log n)$ 时间（ $O(n)$ 空间）内解决上文中的问题（瓶颈是在 f 数组中二分）。

2.1.1 优化

事实上，参考树剖中的全局平衡二叉树技巧 [1]，算法的复杂度可以进一步优化至 $O(n + q \log S)$ ：

对于固定的重链 $q_1 \rightarrow q_2 \rightarrow \dots \rightarrow q_k$, 记 $w_i = f_i - f_{i+1}$ (这里 $f_{k+1} = 0$; 为叙述方便, 假定 $q_i = i$)。

考虑按以下方式对整条链建立二叉树结构: 对于区间 $[l, r]$ ($l < r$), 记其子树根节点为满足 $l \leq i < r$ 且 $2 \sum_{j=l}^i w_j \geq \sum_{j=l}^r w_j$ 的最小的 i (若不存在则取 $i = l$), 并令左右子树分别为 $[l, i]$ 和 $[i+1, r]$ 建树的结果。并对每个 u , 预处理它在二叉树上和节点 k 的 LCA。

记 u 为进入当前重链的位置, v 为退出的位置, 预处理出每个点和重链终点 k 的 LCA, 并从 $LCA(u, k)$ 向下二分, 则在二叉树上二分的复杂度即为 $LCA(u, k)$ 和 v 在二叉树上的距离。

考虑从 v 开始, 每次向上走 2 步, w_v 的子树和至少会翻倍 (否则子树 v 会被切开; v 是叶子的情况例外), 而 $LCA(u, k)$ 往下走 1 步后子树和不会超过 f_u 。因此, 二分的复杂度为 $O(\log \frac{f_u}{f_v - f_{v+1}})$ 。

考虑询问路径上的所有重链, 则当前重链的 $f_v - f_{v+1}$ 大于等于下一条重链的 f_u 。因而二分的总复杂度为 $O(\log S)$ 。

使用 splay 和 treap 等平衡树也能得到相同的复杂度, 此处不再赘述。

2.1.2 2.1.1 优化的推广

使用分治 (即基于二区间合并的 $O(n \log n) - O(1)$ 区间半群信息) 结合 DP 预处理可以在 $O(n \log n + q \log S)$ 时间内支持求 $P_l, P_{l+1}, \dots, P_r$ 结点权值和的最大值 (以及其它一些较为复杂的 P 上区间信息) 的查询。具体地, 对每个点求出以它为起点的路径的最大权值, 再对每条重链记录上述信息的区间最值即可。

对于有单点修改和简单链修改、查询一条链信息的问题, 也可以将带权线段树推广从而使单次操作复杂度降至 $O(\log S)$ 。

除 $w_i = f_i - f_{i+1}$ 外, 再记 $x_i = h_i - h_{i-1}$ (这里 $h_0 = 0$)。

考虑按以下方式建二叉树: 对于区间 $[l, r]$ ($l < r$), 记其子树根节点为满足 $l \leq i < r$ 且

$$\begin{cases} 2 \sum_{j=l}^i w_j \geq \sum_{j=l}^r w_j & d \equiv 1 \pmod{2} \\ 2 \sum_{j=l}^i x_j \geq \sum_{j=l}^r x_j & d \equiv 0 \pmod{2} \end{cases} \quad (1)$$

(其中 d 为当前节点深度) 的最小的 i (若不存在则取 $i = l$), 并令左右子树分别为 $[l, i]$ 和 $[i+1, r]$ 建树的结果。则在二叉树上二分的复杂度即为 u 和 v 在二叉树上的距离, 其中 u 为进入当前重链的位置, v 为退出重链的位置。

注意到 u 到 $LCA(u, v)$ 的距离不超过 $O(\log \frac{h_v}{h_u - h_{u-1}})$: 这是因为从 u 开始每向上走 4 步 h_u 的子树和至少翻倍 (中途有 2 个点 d 为偶数, 进而可以套用 2.1.1 的结论)。对 v 再进行类似分析, 可知区间查询 (修改) 的复杂度不超过 $O(\log \frac{f_u}{f_v - f_{v+1}} + \log \frac{h_v}{h_u - h_{u-1}})$, 从而所有重链上的总复杂度不超过 $O(\log S)$ 。

此外, 对于带单点修改的可减信息, 关于 f 和 h 分别建立带权线段树也可做到单次查询 $O(\log S)$ 。

不过, 由于常数较大, 若维护的信息较为简单, 该算法比起 BIT 并没有显著优势。

2.2 可持久化平衡树

不妨按 DAG 的拓扑序逆序考虑原问题。对每个结点 i , 考虑以它开头的所有路径构成的序列 Q_i 。

显然, 对于出度为 0 的结点 i , $Q_i = \{(i)\}$; 对于出度为 2 的结点 i , $Q_i = (i) \otimes (Q_{G_{i,1}} \oplus Q_{G_{i,2}})$ (其中 $A \oplus B$ 为 A 和 B 的拼接, $a \otimes B$ 为在 B 的每个元素开头插入 a 得到的序列)。

我们仅维护 Q_i 中每条路径的结点权值和。这样, 只需在 $O(\log S)$ 内进行以下几种操作:

1. 拼接两个长度不超过 S 的整数序列, 得到一个新序列;
2. 将一个序列的所有元素加上一个定值;
3. 给定 k , 查询一个序列的第 k 项。

考虑使用平衡树维护。由于自复制操作的存在, 我们需要一种不带 parent 指针且树高严格 $O(\log S)$ 的平衡树。此处以 WBLT (Weight Balanced Leafy Tree [2]) 为例 (WBLT 中已经维护了 size 信息故相对方便; AVL 树、红黑树等亦可)。

具体地, 取一个平衡常数 $\alpha \in (0, 1 - \frac{\sqrt{2}}{2})$, WBLT 是这样的一棵二叉树: 每个叶节点表示一个元素 (这也是 Leafy 的含义), 每个非叶节点均有左右子树且左右子树的 size 均不低于总 size 的 α 倍; 每个节点处可以记录一些区间信息。这里, 我们用叶节点表示路径, 并在每个节点处维护一个表示子树加的 tag。

对于操作 1, 只需将两棵 WBLT 进行合并操作。具体地, 考虑合并两棵 WBLT A 和 B (不妨设 $|A| \geq |B|$)。

1. 若 $|A|$ 和 $|B|$ 的比值不超过 $t = \frac{1-\alpha}{\alpha}$, 则可以直接新建一个以 A 和 B 为左右子树的点作为答案。

2. 否则, 若 A 的右子树和 B 合并后 size 不超过 A 左子树的 t 倍, 可以直接将 A 的右子树和 B 合并然后递归;

3. 否则, 可将 A 的左子树和 A 的右儿子的左子树合并 (这一部分两边 size 之比不超过 t), 并将 A 的右儿子的右子树和 B 合并 (这一部分可以递归), 再将两次合并的结果合并 (这一步两部分的 size 之比也是 $O(1)$)。

归纳易证合并两棵 size 比值是常数的 WBLT 复杂度为 $O(1)$ 。由于其它递归都是单边的, 合并的总复杂度不超过 $O(\log S)$ (进一步地, 不会超过 $O(\log \frac{|A|}{|B|})$)。另外, 本题中可以利用信息可减、可交换的性质进行适当的标记永久化, 从而减小 pushdown 次数的常数。

对于操作 2, 只需修改根节点的 tag; 对于操作 3, 自顶向下递归即可。

该算法的时间复杂度为 $O((n+q) \log S)$, 空间复杂度为 $O(n \log S)$ 。

2.2.1 WBLT 解法的推广

事实上,除合并外,WBLT(和其它一些平衡树)也支持分裂操作:给定一棵 WBLT A 和整数 $k \in (0, |A|)$,返回两棵 WBLT B 和 C 分别表示 A 的前 k 个元素和后 $|A| - k$ 个元素构成的 WBLT。

具体地,直接自顶向下递归可以将 A 分成若干个深度递增的子树(相当于 B)和若干个深度递减的子树(相当于 C),再将它们自底向上合并即可得到 B 和 C 。由于合并 X 和 Y ($|X| \geq |Y|$) 的复杂度为 $O(\log \frac{|X|}{|Y|})$,容易证明分裂的算法复杂度为 $O(\log S)$ 。

换言之,除了序列合并、全局修改和单点查询,可持久化平衡树也支持序列截取(当然还有区间修改和区间查询,这里的区间反映在原问题 1. 上是路径序列 P 的区间)。

当然,由于结构较为复杂,可持久化平衡树很难支持对 DAG 点权的修改操作。

2.2.2 相关讨论

可以发现,每个点出度为 0 或 2 的 DAG 在结构上等价于一棵可持久化 Leafy Tree。

这意味着无特殊性质且操作次数较少时可以直接维护 DAG 实现序列合并和序列截取,而用于维护平衡性的操作可以省去。

具体地,定义结点 v 的深度 dep_v 为以 v 为起点的最长路径边数。将 A, B 合并为 C 时,只需从 C 向 A, B 连边,则 $\text{dep}_C = \max(\text{dep}_A, \text{dep}_B) + 1$;将 A 分裂为 B, C 的时间复杂度不会超过 dep_A (容易证明递归分裂的过程中每个深度的点只会在前后访问各一次),而通过沿着 B 和 C 的分界线从后往前合并,我们可以保证 $\max(\text{dep}_B, \text{dep}_C) \leq \text{dep}_A$ 。因此 $O(q)$ 次序列合并和序列截取后 dep 不会超过 q ,进而总复杂度为 $O(q^2)$ (q 为操作次数)。这也达到了理论下界,因为 q 步后的路径个数可以达到 $\exp(\Theta(q))$;而事实上可持久化平衡树的优势在于 DAG 较为特殊的情形。

此外,2.1.2 中得到的 DAG 链剖分套带权线段树结构也可看作是和原 DAG 在路径序列的意义下等价但更“平衡”的一个 DAG。

2.3 平衡树分裂归并

考虑将原问题离线,并在结点 1 上维护所有查询的 k 构成的集合。

按拓扑序从前往后遍历所有结点:扫到结点 i 时,先将 i 上的查询答案增加 c_i ,若 i 出度为 2 再将这些查询全部移动到 i 指向的结点上(其中后半一半需要将对应的 k 减去 $f_{G_{i+1}}$)。

考虑如何维护这个过程。我们需要维护若干个非严格递增序列,支持以下三种操作:(序列被操作后就会消失)

1. 将一个序列划分成 $\geq k$ 和 $< k$ 形成两个新的序列;
2. 将一个序列的所有元素加上 k ;
3. 将两个序列归并得到一个新的序列。

用平衡树维护这些序列, 则操作 1 和 2 是平凡的。

对于操作 3, 考虑归并两个序列 $A_{1\dots p}$ 和 $B_{1\dots s}$ 得到序列 $C_{1\dots p+s}$ 。将 $1, 2, \dots, p+s$ 划分成 t 个极长连续段 $[l_i, r_i]$ 使得 $C[l_i \dots r_i]$ 在归并前属于同一个序列 (A 或 B)。则从前往后二分和自顶向下递归都能得到简单 $O(t \log q)$ 复杂度做法。

接下来, 我们证明 t 的总和不会超过 $O((n+q) \log S)$:

对于序列 $A_{1\dots p}$, 定义 A 的势能为 $\sum_{i=1}^{p-1} \log(2 + A_{i+1} - A_i)$ 。显然总势能非负且不会超过 $O(q \log S)$, 操作 1 最多让势能变化 $O(\log S)$ 而操作 2 不会改变势能, 故只需证操作 3 中势能至少减少 $\Omega(t) - O(\log S)$:

考虑序列 $A_{1\dots p}$ 和 $B_{1\dots s}$ 归并得 $C_{1\dots p+s}$, t 个极长区间依次对应的 C 值为 $[l_i, r_i]$ 。则 $r_i \leq l_{i+1}$ 且 $r_i < l_{i+2}$ 。

记 $d_i = l_{i+1} - r_i$, 注意到势能的变化量

$$\begin{aligned} \Delta &= \sum_{i=1}^{t-1} \log(2 + l_{i+1} - r_i) - \sum_{i=1}^{t-2} \log(2 + l_{i+2} - r_i) + O(\log S) \\ &\leq \frac{1}{2} \sum_{i=1}^{t-2} (\log(2 + d_i) + \log(2 + d_{i+1}) - 2 \log(2 + \max(d_i + d_{i+1}, 1))) + O(\log S) \\ &\leq \frac{1}{2} \sum_{i=1}^{t-2} (\log(2 + \min(d_i, d_{i+1})) - \log(2 + \max(2 \min(d_i, d_{i+1}), 1))) + O(\log S) \\ &\leq O(\log S) - \Omega(t) \end{aligned}$$

t 的总和不会超过势能总增加量, 故有上界 $O((n+q) \log S)$ 。

由此, 我们得到了一个 $O((n+q) \log S \log q)$ 时间 $O(n+q)$ 空间的做法。

2.4 推广与讨论

回顾 2.2.1 中可持久化平衡树支持的四种操作 (序列合并、序列截取、全局加、单点查询)。考虑将这些操作看作一张 DAG, 并按倒序用平衡树维护所有查询。可以发现, 瓶颈依然是平衡树的分裂、全局加和归并。换言之, 可持久化平衡树和平衡树分裂合并所解决的问题在一定程度上互为转置。

遗憾的是, 平衡树分裂合并已知的最优复杂度上界为 $O(\log S \log q)$ (虽然也并未构造出达到该上界的数据), 而可持久化平衡树为 $O(\log S)$ 。事实上, 在与本文有关的问题里, 该算法仅在对空间复杂度有特别要求或操作 DAG 中的某些信息强制在线给出时 (如分裂不是分成 $\geq k$ 和 $< k$ 的而是分成前 k 项和其余项) 优于可持久化平衡树。

3 应用

3.1 对序列操作过程的维护

根据 2.2.2, 将 DAG 看作可持久化 Leafy Tree 可以方便地实现序列截取和序列合并等操作。

3.1.1 例题 1

有 n 个序列, 初始时第 i 个序列仅包含一个数 i 。

现需进行 q 次操作, 每次为以下三种之一:

1. 新建一个序列, 它等于某两个已有序列的拼接;
 2. 新建两个序列, 分别为某个已有序列的前 k 项和其余项构成的序列 (其中 k 为不超过序列长度的高精度数)。
 3. 输出某个序列的逆序数, 对 998244353 取模。
- 保证 $1 \leq n \leq 100, 1 \leq q \leq 600$ 。

3.1.2 例题 1 解法

假定没有操作 2, 则显然可以维护每个序列中每个数的出现次数和总逆序数, 单次操作复杂度 $O(n)$ 。

考虑使用 2.2.2 中的做法: 对于操作 2, 我们自顶向下递归分裂, 再自底向上逐个合并 (过程中需要维护每个结点的 size), 则每次分裂操作可以被改写为 $O(q)$ 次合并操作; 而合并的复杂度为 $O(\frac{q}{w} + n)$ (这是因为 size 有 $O(q)$ 个 bit; 通常认为 w 取 32 或 64)。

总复杂度为 $O(nq^2 + \frac{q^3}{w})$ 。

3.2 在后缀自动机上的应用

在 OI 字符串理论中, 后缀自动机 (SAM) 是一种高效而实用的算法。

具体地, 对于字符集 Σ 上的字符串 S , 考虑 S 的子串 T 在 S 中出现的右端点集合, 记作 R_T 。对于不同的 T , R_T 之间的关系只能是包含或相离, 故本质不同的 R_T 只有 $O(|S|)$ 种且构成树形结构, 记作树 P (实为将二度点压缩后的反串后缀树)。

定义自动机 A , 其点集为 $\{R_T | T \text{ 为 } S \text{ 的子串}\}$, 边集为 $\{(R_T, a) \rightarrow R_{Ta} | a \in \Sigma \text{ 且 } Ta \text{ 为 } S \text{ 的子串}\}$ 。

由 SAM 相关性质知 A 的边数为 $O(|S|)$ 且存在 $O(|S|)$ 的算法构造出树 P 和自动机 A 。

注意到 S 的本质不同子串可以和 A 上从空串出发的路径一一对应。由此, 我们将本质不同子串信息转化成了自动机上路径信息, 从而可以高效地处理一些从后缀树角度看比较棘手的问题。

3.2.1 例题 2

题目来源为 UOJ Long Round #1 D 题² (略去了不必要的细节)。

给定字符串 S 和数组 a, b 。对于 S 的子串 T , 记 T 在 S 中出现的右端点集合为 R_T , 并定义 $f(T) = |R_T| \sum_{i \in R_T} b_i$ 。

对每个 i , 试求 $\sum_{j=i}^{|S|} f(S[i \cdots j])$ 。

保证 $1 \leq |S| \leq 5 \times 10^5$, 字符集 26。

3.2.2 例题 2 解法

建立 S 的反串后缀树 P 和后缀自动机 G 。注意到对 P 的每个结点 (也即 G 的每个结点), 它的 f 值是个定值; 而查询恰好是求 G 的一条路径上 f 的和。

使用 DAG 链剖分并对每条重链维护前缀和即可解决问题。

瓶颈是求询问路径和当前重链的公共长度, 此处用 $O(|S| \log |S|) - O(1)$ 的 ST 表 LCP 即可。

3.2.3 例题 2 讨论

事实上, SAM 的极长路径只有 $O(|S|)$ 条 (对应 S 的所有后缀)。基于这一性质, 存在更简单的算法。

将 SAM 的所有路径展开, 会得到一棵只有 $O(|S|)$ 个叶节点的树 (实为 S 的正串后缀树)。每次查询相当于求这棵树的一条链上所有结点权值和; 另一方面, 这些结点也都对应着 SAM 结点。

将这棵树的二度点压缩并考虑压缩操作在自动机上的体现。具体地, 这相当于删去自动机上所有出度为 1 的点, 并对所有指向它们的边, 把边的终点改为原终点所指向的点 (递归进行直到终点出度不是 1 为止), 再将边上的字符改为一个字符串。

通过在边上记录数组, 压缩后的自动机上仍可以维护区间和; 而将自动机展开后, 会直接得到一棵 $O(|S|)$ 个点的压缩正串后缀树。直接在这棵树上进行前缀和处理询问即可。

总复杂度 $O(|S| + q)$ 。

值得一提的是, 原串和反串的压缩后缀自动机结点集合是本质相同的 (详见 [3]), 这无疑为处理和左右端点均相关的子串问题提供了一个新的视角。

不过该算法也具有局限性: 压缩后缀自动机的结构依赖正反串的对称性, 因此可拓展性不如 DAG 剖分。

² 【ULR #1】打击复读, <https://uoj.ac/problem/577>

3.2.4 例题 3

给定一棵 n 个点的 trie S 。定义 S 的子串为一条自顶向下的路径对应的字符串。

q 次查询，每次给定 k ，求 S 中第 k 小的本质不同子串或报告不存在。（为避免过大的输出量，只需输出该字符串的所有字符 ASCII 码之和）

保证 $1 \leq n, q \leq 10^5$ ， $1 \leq k \leq 10^{10}$ ，字符集 26。

3.2.5 例题 3 解法

对原树建 SAM。具体地，反串后缀树 T 被定义为所有点到根的路径对应的串构成的 trie，而后缀自动机 G 的 right 集合与 T 的结点一一对应。

注意到串上 SAM 构造算法的复杂度分析不再成立，且此时 SAM 的边数可能达到 $n|\Sigma|$ ，其中 Σ 为字符集。根据 [4] [5] [6]，直接在 trie 上 BFS 并使用串上 SAM 算法是 $n|\Sigma|$ 的，不过笔者并不知晓完整证明；当然，也可以用推广的倍增 SA 计算 T ，再用 T 计算 G ，总复杂度 $O(n \log n + n|\Sigma|)$ （进一步地，利用 DC3 [7] 算法在树上的推广可以去掉 $\log$ 因子）。

构造出 G 后，问题即被转化为第一节经典模型，使用第二节中介绍的三种算法均可解决。以 DAG 剖分为例，其总复杂度为 $O(n \log n + n|\Sigma| + q \log^2 n)$ 或 $O(n \log n + n|\Sigma| + q \log n)$ 。

3.2.6 例题 4

题目来源为 UOJ 候选队互测 #1 C 题³。

给定长为 n 的字符串 S 和 m 个 S 的子串作为模式串。

有 q 次查询，每次求 S 的一个子串中所有模式串出现次数之和。

保证 $1 \leq n \leq 4 \times 10^5$ ， $1 \leq m \leq 10^6$ ， $1 \leq q \leq 10^5$ ，字符集 26。

3.2.7 例题 4 解法

对 S 建立后缀自动机 G 和反串后缀树 P ，并对 G 进行 DAG 剖分。

对于 S 的子串 T ， T 的后缀中模式串的个数是由 T 在 SAM 上对应的结点 v 以及 v 所对应的模式串决定的。进一步地，只需将 P 中 v 到根的路径上模式串个数 a_v 减去 v 上长度比 $|T|$ 大的模式串个数即可。

对于每次查询，可以把子串定位到自动机的路径上，并计算路径上每个结点的贡献之和。

考虑路径定位：对原串和 DAG 重链进行 $O(n \log n)$ 的预处理再 $O(1)$ 求子串 LCP 可以做到 $O((n + q) \log n)$ 。

³ 【候选队互测】广为人知题，<https://uoj.ac/problem/697>



考虑答案统计：对于第一种贡献，对 DAG 重链记 a_i 的前缀和即可；对于第二种贡献，可以将问题转化成每条重链上的斜 45 度平行四边形数点，在坐标变换后即是普通的二维数点。

时间复杂度为 $O((n+m)\log n + q\log^2 n)$ 或 $O((n+m+q)\log n)$ （使用带权线段树和较优的 LCP 的实现）。

3.3 万能欧几里得算法

网格图上有一条直线 $y = \frac{b}{a}x$ ($a, b \in \mathbb{N}^+$)。考虑该直线上一动点沿 $x+y$ 正方向移动一个区间，每经过一条横线就在纸上记录一个 A ，每经过一条竖线就记录一个 B （若经过格点，则先记 A 再记 B ），最终会得到一个由 $\{A, B\}$ 构成的序列（如 $a=3, b=2$ 时序列为 $ABBAB$ ）。

不妨将 A 和 B 各代入一个半群元素（即 $O(1)$ 可合并的信息），则万能欧几里得算法的目标是求出整个序列对应的半群元素。

先假定动点的移动范围为 $(0, 0) \rightarrow (a, b)$ （左闭右开）。考虑算法流程如下：

不妨设 $a > b$ 。记 $t = \lfloor \frac{a}{b} \rfloor$ ，则容易发现每个 A 后面都有 t 个 B 。考虑将原序列中的所有 AB^t 替换为 A' ，这相当于对平面和直线作用了坐标变换 $(x', y') = (x, y - tx)$ 。因此，只需令 $A' = AB^t$ ， $a' = a - tb$ ，即可将原问题归约到规模更小的问题。重复迭代 $O(\log a)$ 次即可解决原问题。

考虑用 DAG 描述上述算法。具体地，维护一张 DAG，初始时有两个点 A, B 。每次建立一个点 C ，向 A 和 B 都连边，接着若 $a > b$ 则令 $B \leftarrow C, a \leftarrow a - b$ 否则令 $A \leftarrow C, b \leftarrow b - a$ ，直到 $a = b$ 为止。

注意到 DAG 的边数可能达到辗转相减的复杂度即 $O(a)$ 。这里，我们只需用倍增将连续的 $a \leftarrow a - b$ （或 $b \leftarrow b - a$ ）操作优化，即可将图的规模降至 $O(\sum \log \frac{\max(a_i, b_i)}{\min(a_i, b_i)}) = O(\log(a+b))$ （其中 a_i, b_i 为每次坐标时的 a, b ）。

万能欧几里得过程产生的 AB 序列即是这张 DAG 按 DFS 顺序展开后所有路径终点构成的序列，因而一般情况下的万欧算法相当于这张 DAG 对应的可持久化 Leafy Tree 上的区间查询。容易发现，很多经典的类欧几里得问题（如求 $\sum_{i=1}^n ai \bmod b$ 或 $\min_{i=1}^n ai \bmod b$ ）都可以转化成该问题；进一步地，借助这张 DAG，我们可以解决一些和欧几里得过程有关的较为复杂的问题。

3.3.1 例题 5

此题为 LOJ 上的万能欧几里得模板题⁴。

给定正整数 n, m, l 和大小不超过 10 的方阵 A, B 。求 $\sum_{i=0}^{l-1} A^i B^{\lfloor \frac{l-i}{m} \rfloor}$ 。

$1 \leq n, m, l \leq 10^{18}$ 。

⁴<https://loj.ac/p/6440>



3.3.2 例题 5 解法

考虑直线 $y = \frac{im}{n}x$ 对应的 AB 序列。

维护矩阵 M ，初始时 $M = I$ 。从序列的第二个位置开始，每遇到一个 A 就令 $M = MB$ ；每遇到一个 B 就将答案加上 M 并令 $M = AM$ ，则最终答案即为所求。注意到序列的一个区间起到的作用可以用矩阵 U, V, W 表示：记进入区间前 A, B 的幂次为 X, Y ，则经过区间会令答案加上 XUY 并令 $X = XV, Y = WY$ 。由此，区间信息可以被 $O(1)$ 描述，进而直接套用万能欧几里得算法的区间版本即可解决原问题。

3.3.3 例题 6

题目来源为 CTT 2021 Day 4 T3（略去了不必要的细节）。

考虑一张 n 个点的无向图，其中 i 和 $(i + d) \bmod n$ 之间连边。第 i 个点有点权 a_i ，初始时 $a_i = w_{i \bmod m}$ 。

你需要进行 q 次对点权的单点修改，每次修改后求出图的最大权独立集大小。

$1 \leq d < n \leq 10^{16}$, $1 \leq m \leq 2 \times 10^4$, $1 \leq w_i \leq 400$, $1 \leq q \leq 10^5$ 。

3.3.4 例题 6 解法

容易发现，图由 $\gcd(n, d)$ 个环构成，而本质不同的环只有 m 种。

进一步地，将所有被修改过至少一次的点删去后，图由 $O(q)$ 条链和 m 种环构成。只需预处理出每条链和每种环的答案并求和（这里对每条链需要求出一个 2×2 矩阵表示首尾选或不选的最大权独立集），再关于关键点维护线段树即可。

考察一个环上的信息。具体地，在网格图上作线段 $y = \frac{d}{n}x (x \in [0, n))$ 并对它构造万能欧几里得对应的序列和 DAG；另维护一个数 x ，不同的 x 对应不同的环。遍历序列并维护一个 2×2 矩阵，每遇到一个 A 令 $x = (x - n) \bmod m$ ，每遇到一个 B 令 $x = (x + d) \bmod m$ ，同时将当前的 2×2 矩阵与 w_x 对应的信息合并。则操作完毕后的矩阵即是环的答案（不同的 x 初值对应不同的环）；若仅进行一个区间的操作，会得到链的答案。

序列的区间信息可以被定义为 m 个 2×2 矩阵表示 x 初值依次为 $0, 1, \dots, m-1$ 时作用这个点所对应的操作后得到的 2×2 矩阵，以及一个数 d 表示 x 经过该区间后的增量。在欧几里得算法对应的 DAG 上 DP 可以 $O(m \log n)$ 求出所有结点对应的区间信息，从而立即得到环的答案。

链的答案相当于对 DAG 路径信息的区间查询；由于该 DAG 仅有 $O(\log n)$ 个点，直接自顶向下 DFS 即可达到 $O(q \log n)$ 的复杂度（具体实现参考支持求区间矩阵乘积乘向量的线段树；由于 x 的初值确定，这一部分的复杂度不带 m ）。

总复杂度 $O((m + q) \log n)$ 。



总结

本文介绍了 DAG 链剖分、可持久化平衡树和平衡树分裂归并三种算法；并且给出了其在部分 OI 题目中的应用。事实上，笔者猜测 DAG 路径维护的模型背后可能有其它更为优美、深刻的性质，文中三种算法的更多拓展及它们之间的联系也有待进一步挖掘。故希望本文能起到抛砖引玉的作用，也希望有兴趣的读者能继续研究。

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杨耀良的指导。

感谢父母对我的培养和教育。

感谢学校的栽培，李曙老师、张超老师的教导和同学们的帮助。

感谢杜瑜皓学长、李欣隆学长、周欣学长、钱易同学等与我交流讨论、给我启发。

感谢钱易同学、罗恺同学、丁晓漫同学、许庭强同学为本文审稿。

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] 张哲宇. 《浅谈树上分治算法》. IOI 2019 国家集训队论文, 2019
- [2] 王思齐. 《Leafy Tree 及其实现的加权平衡树》. IOI 2018 国家集训队论文, 2018
- [3] 徐翊轩. 《浅谈压缩后缀自动机》. IOI2020 国家集训队论文, 2020
- [4] 刘研绎. 《后缀自动机在字典树上的拓展》. IOI2015 国家集训队论文, 2015
- [5] J. A. Blumer. *Algorithms for the directed acyclic word graph and related structures*. Ph.D. Thesis, Denver University, 1985
- [6] A. Blumer, J. Blumer, D. Haussler, R. M. McConnell, A. Ehrenfeucht. *Complete inverted files for efficient text retrieval and analysis*. J. ACM 34(3)(1987), pp. 578-595
- [7] J. Karkkainen, P. Sanders, S. Burkhardt. *Linear work suffix array construction*. J. ACM 53(6)(2006), pp. 918-936



更一般的 DFT

杭州学军中学教育集团文渊中学 周航锐

摘要

离散傅里叶变换在信息学竞赛中应用非常广泛, 本文对传统 DFT 进行推广, 以略高的复杂度完成更一般的二元关系定义的卷积。

1 相关定义及说明

1.1 卷积和基础运算

定义 1.1. 对于一个集合 G 和一个二元关系 $\varphi: G \times G \rightarrow G$, 映射 $A, B: G \rightarrow \mathbb{C}$, 定义卷积 $C = A \times B$ 满足:

$$C(v) = \sum_{\varphi(x,y)=v} A(x)B(y)$$

两个映射的和 $C = A + B$ 满足:

$$C(v) = A(v) + B(v)$$

一个映射和一个复数的数乘 $B = cA$ 满足:

$$B(v) = cA(v)$$

本文只讨论 (G, φ) 构成有限半群的情形。对于映射 $A: G \rightarrow \mathbb{C}$, 本文 A_x 与 $A(x)$ 意义相同。

1.2 DFT

定义 1.2. 对于大小序列¹ $(a_i)_{i=1}^k$, 定义它对应的矩阵序列集为 R , 满足:

¹这是本文的特殊定义

- $R = \{(m_i)_{i=1}^k | m_i \in M_{a_i}\}$, M_n 包含所有 $n \times n$ 的复数矩阵。
- $C = A + B$ 满足 $C_i = A_i + B_i$ 。
- $C = AB$ 满足 $C_i = A_i B_i$ 。
- $B = cA$ 满足 $B_i = cA_i$ 。

其中 $A, B, C \in R$ 是大小序列相同的矩阵序列, c 是一个复数。

定义 1.3. 对于某个卷积关系, 定义它对应的 DFT 为可逆线性变换 $\mathcal{F} : (G \rightarrow \mathbb{C}) \rightarrow R$, 其中 R 是某个大小序列 $(a_i)_{i=1}^k$ 定义的矩阵序列, 并且满足:

- $\mathcal{F}(A) \times \mathcal{F}(B) = \mathcal{F}(AB)$
- $\mathcal{F}(A) + \mathcal{F}(B) = \mathcal{F}(A + B)$
- $c\mathcal{F}(A) = \mathcal{F}(cA)$
- $\sum_{i=1}^k a_i^2 = |G|$

可以认为 $\mathcal{F}$ 是一个同构。

对于矩阵序列的乘法, 我们可以做到 $O(\sum_{i=1}^k a_i^w)$ 。其中 w 表示 $n \times n$ 矩阵乘法的复杂度为 $O(n^w)$, 目前已知 $w \leq 2.3728596$ [5]。

$(G \rightarrow \mathbb{C})$ 存在一组基 $\{A_x\} (x \in G)$, 其中 A_x 为 $t \mapsto [t = x] = \begin{cases} 1 & t = x \\ 0 & t \neq x \end{cases}$ 。

因为每个元素都可以被分解成基的线性组合, 因此只需要关心 $\mathcal{F}(A_x)$ 即可。定义 $\mathcal{F}_x = \mathcal{F}(A_x)$ 。

引理 1.1. DFT 关系 $\mathcal{F}$ 与满足如下条件的 $\{\mathcal{F}_x\}$ 一一对应:

- $\mathcal{F}_x \times \mathcal{F}_y = \mathcal{F}_{\varphi(x,y)}$ 。
- $\mathcal{F}_x$ 构成 R 的一组基。

证明. 对于一个 DFT 关系 $\mathcal{F}$, 由于同构, 显然它得到的 $\{\mathcal{F}_x\}$ 合法。

对于一组 $\{\mathcal{F}_x\}$, 我们可以做如下构造: $\mathcal{F}(A) = \sum_{x \in G} A(x) \mathcal{F}_x$ 。

对于乘法:

$$\begin{aligned}
 & \mathcal{F}(A) \times \mathcal{F}(B) \\
 &= \sum_{x \in G} \sum_{y \in G} A_x B_y \mathcal{F}_x \times \mathcal{F}_y \\
 &= \sum_{v \in G} \sum_{\varphi(x,y)=v} A_x B_y \mathcal{F}_x \times \mathcal{F}_y \\
 &= \sum_{v \in G} \mathcal{F}_v \sum_{\varphi(x,y)=v} A_x B_y \\
 &= \mathcal{F}(A \times B)
 \end{aligned}$$

对于加法及数乘，显然满足。

对于双射，可以有 $\{\mathcal{F}_x\}$ 构成一组基得到。

□

因此直接在已知 $\{\mathcal{F}_x\}$ 的情况下，DFT 可以看成行向量乘变换矩阵，复杂度为 $O(|G|^2)$ 。

如果需要 IDFT（即 DFT 的逆变换），求出逆矩阵需要 $O(|G|^n)$ 的预处理复杂度。

2 传统 DFT 和循环群

2.1 序列卷积

对于一般的序列卷积，对应的二元关系为： $(\mathbb{Z}, +)$ 。但是在实际操作时，一般取 2^k 满足 2^k 超过序列长度 n ，把二元关系转化为 $(\mathbb{Z}_{2^k}, +)$ 后操作，其中 $\mathbb{Z}_p$ 表示阶为 p 的循环群。

2.2 循环群

引理 2.1. 对于循环群 $\mathbb{Z}_p$ ，一个它对应的 DFT 为：

$$\mathcal{F}_i = \{[1], [\omega_p^i], [\omega_p^{2i}], \dots, [\omega_p^{(p-1)i}]\}$$

其中 ω_p 为一个 p 次本原单位根。

读者可以自行验证，这个构造符合上述条件。

2.3 FFT 与 chirp-Z 变换

对于循环群 $\mathbb{Z}_p$ 满足 $p = 2^k$ ，可以使用 FFT 算法进行优化，使 DFT 做到 $O(p \log p)$ 。

对于更一般的循环群 $\mathbb{Z}_p$ ，可以使用 chirp-Z 变换后，转为上面一种情况，同样做到 $O(p \log p)$ 的复杂度，具体的：

$$\begin{aligned}\mathcal{F}(A)_i &= \left[\sum_{j=0}^{p-1} A_j \omega_p^{ij} \right] \\ &= \left[\frac{1}{\omega_p^{\binom{i}{2}}} \sum_{j=0}^{p-1} \frac{A_j}{\omega_p^{\binom{j}{2}}} \omega_p^{\binom{i+j}{2}} \right]\end{aligned}$$

这是一个减法卷积的形式，可以翻转后转化为序列卷积，进而转化为 $\mathbb{Z}_{2^k}$ 上的问题。时间复杂度 $O(p \log p)$ 。

因此对于循环群，我们都可以在 $O(|G| \log |G|)$ 的复杂度内完成 DFT，并且矩阵乘法的复杂度显然为 $O(|G|)$ 。

3 直和及对应的 DFT

3.1 直和

定义 3.1. 对于二元关系 $(G_1, \times)$ 和 $(G_2, \times)$ ，定义它们的直和 $(G, \times)$ 满足：

- $G = \{(x, y) | x \in G_1, y \in G_2\}$
- $(x_1, y_1) \times (x_2, y_2) = (x_1 \times x_2, y_1 \times y_2)$

3.2 DFT 的构造

首先证明一个引理。

引理 3.1. 对于有限维线性空间， $\otimes$ 表示张量积， $V = V_1 \otimes V_2 = \text{span}\{x | x = a \otimes b, a \in V_1, b \in V_2\}$ ， V_1 的一组基为 $(a_i)_{i=1}^n$ ， V_2 的一组基为 $(b_j)_{j=1}^m$ ，则 V 的一组基为 $c = \{x | x = a_i \otimes b_j, i \in [1, n], j \in [1, m]\}$ 。

证明. 显然 c 是 V 的一组生成元，我们只需证明其中元素线性无关即可。

考虑反证法，设 $c_{(i-1)m+j} = a_i \otimes b_j$ ，不妨令 $\sum_{i=1}^n \sum_{j=1}^m c_{(i-1)m+j} k_{i,j} = \mathbf{0}$ ， $k_{i,j}$ 不全为 0。

令 $S_i = \sum_{j=1}^m k_{i,j} b_j$ ，则 $\sum_{i=1}^n S_i \otimes a_i = \mathbf{0}$ 。

根据 a 构成一组基可得， $\forall i, S_i = \mathbf{0}$ 。

又根据 b 构成一组基可得，若 $S_i = \mathbf{0}$ ，则 $k_{i,j} = 0 (j \in [1, m])$ 。

因此 $k_{i,j}$ 均为 0，矛盾。 $\square$

定义 3.2. 对于大小序列分别为 $(a_i)_{i=1}^n, (a'_i)_{i=1}^m$ 的矩阵序列 R 和 R' ，定义他们的张量积 $R \otimes R'$ 满足：

$$(R \otimes R')_{(i-1)m+j} = R_i \otimes R'_j$$

定理 3.1. 若 $G_1 \rightarrow \mathbb{C}$ 和 $G_2 \rightarrow \mathbb{C}$ 均存在 DFT, 那么 $G \rightarrow \mathbb{C}$ 也存在 DFT。

证明. 设两个 DFT 分别为 $\mathcal{A}, \mathcal{B}$, 对应的大小序列分别为 $(a_i)_{i=1}^n, (b_i)_{i=1}^m$ 。

设 $G \rightarrow \mathbb{C}$ 对应的 DFT 为 $\mathcal{F}$, 对应的大小序列为 $(c_i)_{i=1}^{nm}$ 。

可以如下构造:

- $c_{(i-1)m+j} = a_i b_j | i \in [1, n], j \in [1, m]$
- $\mathcal{F}_{(x,y),(i-1)m+j} = \mathcal{A}_{x,i} \otimes \mathcal{B}_{y,j}$, 其中 $\otimes$ 表示张量积。

对于加法和数乘, 显然成立。

由于张量积的性质: $(x_1 \otimes y_1)(x_2 \otimes y_2) = (x_1 x_2) \otimes (y_1 y_2)$, 其中 x_1, x_2 为 $n \times n$ 的矩阵, y_1, y_2 为 $m \times m$ 的矩阵, 因此对于乘法:

$$\begin{aligned} & \mathcal{F}_{(x_1, y_1), v} \times \mathcal{F}_{(x_2, y_2), v} \\ &= (\mathcal{A}_{x_1, a} \otimes \mathcal{B}_{y_1, b})(\mathcal{A}_{x_2, a} \otimes \mathcal{B}_{y_2, b}) \\ &= (\mathcal{A}_{x_1, a} \mathcal{A}_{x_2, a}) \otimes (\mathcal{B}_{y_1, b} \mathcal{B}_{y_2, b}) \\ &= \mathcal{F}_{(x_1, y_1) \times (x_2, y_2), v} \end{aligned}$$

其中 $v = (a-1)m + b, a \in [1, n], b \in [1, m]$ 。

由引理 3.1, $\{\mathcal{F}_v\}$ 构成一组基。

□

有推论

引理 3.2. 如果有

$$G \rightarrow \mathbb{C} = (G_1 \times G_2 \times \dots \times G_k) \rightarrow \mathbb{C}$$

且 $G_i \rightarrow \mathbb{C} (i \in [1, k])$ 有 DFT, 可以构造出 $G \rightarrow \mathbb{C}$ 的 DFT。

3.3 更优的复杂度

引理 3.3. G 对应的变换矩阵是 G_1 对应的变换矩阵和 G_2 对应的变换矩阵的张量积。

考虑 G DFT 的构造过程, 显然可以得到。

定理 3.2. 对于 $G \rightarrow \mathbb{C} = (G_1 \times G_2) \rightarrow \mathbb{C}$, 映射 $a: G \rightarrow \mathbb{C}$, 对 a 应用 G 对应的 DFT 等价于先后应用 G_1, G_2 对应的 DFT。具体的:

- 对于 $x \in G_1$, 令 G'_x 满足 $G'_{x,y} = G_{(x,y)}$, 对于 G'_x 分别应用 G_2 上的 DFT, 记结果为 H_x 。

- 令 G_2 对应的矩阵序列集为 R , 对应的大小序列为 $(b_i)_{i=1}^k$, 对于 $i \in [1, k]$, 令 H'_i 满足 $H'_{i,x} = H_{x,i}$, 对 H'_i 分别应用 G_1 上的 DFT, 注意这里映射变为了 $G_1 \rightarrow M_{b_i}$, 但是过程相似。

考虑变换矩阵的性质即可。

同样有推论:

引理 3.4. 如果有

$$G \rightarrow \mathbb{C} = (G_1 \times G_2 \times \dots \times G_k) \rightarrow \mathbb{C}$$

那么对 a 应用 G 对应的 DFT 等价于先后应用 $G_i (i \in [1, k])$ 对应的 DFT (可能需要划分 G)。

引理 3.5. 这么做的复杂度是

$$T = O\left(|G| \sum_{i=1}^k \frac{T_i}{|G_i|}\right)$$

其中 T_i 表示 G_i 上 DFT 的复杂度。

可以得到推论:

- 如果 $T_i = O(|G_i| \log |G_i|)$, 则 $T = O(|G| \log |G|)$ 。
- 如果 $T_i = O(|G_i|^2)$, 则 $T = O(|G| \sum |G_i|)$
- 若 $|G_i|$ 均是常数, 可以认为 $T = O(k|G|) = O(|G| \log |G|)$ 。

举个例子:

n 维 FWT 是 n 个 $\mathbb{Z}_2 \rightarrow \mathbb{C}$ 的直和, 复杂度为 $O(|G| \log |G|) = O(n2^n)$ 。

例题 1. (Numbers²) 给定 n, k 和长为 k 的数组 $m, p (\sum_{i=1}^k p_i = 1)$, 定义状态是一个 k 元组 c , 初始均是 0, 每次操作有 p_i 的概率让 $c_i := (c_i + 1) \bmod m_i$ 。求第一次回到初始状态时, 期望经过本质不同状态数量。 $T = \prod m_i \leq 5 \times 10^5$, 保证数据随机, 不需要考虑 corner case。

解法 令 $G = \mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \dots \times \mathbb{Z}_{m_k}$, 0 为初始状态。

考虑计算每个状态被经过的概率, 下面假设计算状态 $x \in G$ 被经过的概率。

假定状态到达初始状态或 x 时, 立刻结束操作。令 f_i 表示到达初始状态或 x 前, 经过状态 i 的期望次数。特别的, 强制令 $f_0 = 1, f_x = 0$, 对于其他状态 y 有:

²来源: 张隽恺, <https://ioihw21.duck-ac.cn/problem/101>

$$f_y = \sum_{i=1}^k f_{d(y,i)} p_i$$

其中 $d(y,i)$ 表示将状态 y 的第 i 个元素减一。

直接高斯消元即可做到 $O(T^4)$ 。

引入变量 d_0, d_x , 满足:

$$\begin{aligned} f_0 &= \sum_{i=1}^k f_{d(0,i)} p_i - d_0 = 1 \\ f_x &= \sum_{i=1}^k f_{d(x,i)} p_i - d_x = 0 \end{aligned}$$

验证发现 d_x 即为所求。

令 F 为映射 $x \mapsto f_x$, P 为映射:

$$x \mapsto \begin{cases} p_i & \text{if } x_j = 1 \Leftrightarrow i = j \\ 0 & \text{otherwise} \end{cases}$$

类似的, 对于 d_0 和 d_x , 定义 $D: y \mapsto d_y$ 。

那么上述方程可以写为:

$$F = F \times P - D$$

$$f_0 = 1$$

$$f_x = 0$$

而 $F = F \times P - D$ 等价于 $\mathcal{F}(F) = \mathcal{F}(F) \times \mathcal{F}(P) - \mathcal{F}(D)$ 。这等价于 T 个方程:

$$\mathcal{F}_i(\mathcal{P}_i - 1) = \mathcal{D}_i(i \in G)$$

其中 $\mathcal{F}_{g_i}(g_i \in G)$ 和 $\mathcal{F}(F)_i(i \in [1, |G|])$ 一一对应, $\mathcal{P}, \mathcal{D}$ 类似定义。

可以发现 $\mathcal{P}_0 = \sum p_i = 1$ (实际上, 对于有限群, DFT 结果均存在累和), 即对应方程为 $\mathcal{D}_0 = d_0 + d_x = 0$ 。且由于数据随机, $\mathcal{P}_i(i \neq 0)$ 为 0 的概率很小。

定义 $t(a, b)$ 为: $\mathcal{F}(C)_b = \sum C_a t(a, b)$, 即变换矩阵的系数。

那么有 $t(a, b) = t(b, a), t(a, b)^{-1} = t(a^{-1}, b)$ 。

而 $f_0 = 1$ 等价于 $\sum \mathcal{F}_i = |G|$, $f_x = 0$ 等价于 $\sum \frac{\mathcal{F}_i}{t(i, x)} = 0$ 。

此时我们就把原来的复杂方程组转化为了如下形式:

$$\sum \mathcal{F}_i = |G| \quad (1)$$

$$\sum \frac{\mathcal{F}_i}{t(x, i)} = 0 \quad (2)$$

$$\begin{aligned} \mathcal{F}_i(\mathcal{P}_i - 1) &= \mathcal{D}_i = d_x(t(x, i) - 1) \\ \Rightarrow \mathcal{F}_i &= \frac{t(x, i) - 1}{\mathcal{P}_i - 1} d_x \quad (i \in G \setminus \{0\}) \end{aligned} \quad (3)$$

将方程 3 代入方程 1,2 可以得到:

$$\begin{aligned} \mathcal{F}_0 + \left(\sum \frac{1}{\mathcal{P}_i - 1} - \sum \frac{1}{t(x, i)(\mathcal{P}_i - 1)} \right) d_x &= 0 \\ \Rightarrow \mathcal{F}_0 &= \left(\sum \frac{1}{t(x, i)(\mathcal{P}_i - 1)} - \sum \frac{1}{\mathcal{P}_i - 1} \right) d_x \\ \mathcal{F}_0 + \sum \frac{t(x, i) - 1}{\mathcal{P}_i - 1} d_x &= |G| \\ \Rightarrow \left(\sum \frac{1}{t(x, i)(\mathcal{P}_i - 1)} - \sum \frac{1}{\mathcal{P}_i - 1} \right) d_x + \sum \frac{t(x, i) - 1}{\mathcal{P}_i - 1} d_x &= |G| \\ \Rightarrow \left(\sum \frac{1}{t(x, i)(\mathcal{P}_i - 1)} + \sum \frac{t(x, i)}{\mathcal{P}_i - 1} - 2 \sum \frac{1}{\mathcal{P}_i - 1} \right) d_x &= |G| \end{aligned}$$

直接按照最后式子计算即可做到 $O(|G|^2)$ 。

考虑优化, 瓶颈在于左侧三部分求和。

对映射 $x \mapsto \frac{1}{\mathcal{P}_x - 1}$ 进行 DFT, 第 1,2 部分分别为结果的第 x, x^{-1} 项。第 3 部分是个常数, 于是我们将复杂度优化为 DFT 的复杂度即 $O(T \log T)$, 可以通过本题。 ■

本题是 DFT 的经典应用, 通过 DFT 将复杂的卷积方程转为简单的点乘方程, 再利用偏移量和特殊值推出的方程解出偏移量, 进而解出答案。

4 有限交换半群 DFT

4.1 有限交换群 DFT

4.1.1 有限交换群分解

定义 4.1. 若有限交换群 G 的一组元素 $[a_1, a_2, \dots, a_m]$ 满足:

- $\forall x \in G, \exists (k_i)_{i=1}^m, \prod_{i=1}^m a_i^{k_i} = x$
- $\forall (k_i)_{i=1}^m, \prod_{i=1}^m a_i^{k_i} = e \Rightarrow a_i^{k_i} = e (i \in [1, m])$

则这组元素是 G 的一个分解。

引理 4.1. $[a_1, a_2, \dots, a_m]$ 是 G 的一个分解等价于:

$$\langle a_1 \rangle \times \langle a_2 \rangle \times \dots \times \langle a_m \rangle \simeq G$$

考虑两个性质, 证明较为简单。

接下来我们将证明任意有限交换群均存在分解, 同时给出构造。

引理 4.2. 任意有限交换群 G 都能被分解为有限交换 p 群直和。

证明. 令 $o(x)$ 表示 x 的阶。设 G 的阶为 $p^k m$, $p \in \mathbb{P}$, $p \nmid m$, $G_{p^k} = \{x | o(x) \mid p^k\}$, $G_m = \{x | o(x) \mid m\}$ 。

接下来我们证明: $G_{p^k} \times G_m \simeq G$ 。

构造映射 $\phi: (x, y) \mapsto xy$ 。

显然这是一个群同态, 我们只需证明它满射且 $\text{Ker} \phi = \{e\}$ 即可。

对于任一元素 x , 由拉格朗日定理, $o(x) \mid p^k m$, 设 $o(x) = p^{k'} m' (p \nmid m')$, 因此 $o(x^{p^k}) = m' \mid m$, $o(x^{tm}) = p^{k'} \mid p^k$ 。

因为 $(p^k, m) = 1$, 一定存在 $sp^k + tm = 1$, 于是 $x = x^{sp^k + tm} = x^{tm} x^{sp^k}$, ϕ 是一个满射。

若 $x \in G_{p^k}, y \in G_m, xy = e$, 因为 $o(x) = o(x^{-1}) = o(y)$, 有 $o(x) \mid (p^k, m) = 1$, 因此 $x = y = e$ 。这就证明了 $\text{Ker} \phi = \{e\}$ 。

因此 ϕ 是一个群同构。如此我们便可以不断分解, 把 G 分解为若干有限交换 p 群直和。

□

4.1.2 有限交换 p 群分解

引理 4.3. 对于有限交换 p 群 G , 其中一个阶最大的元素为 a , 令 $H = \langle a \rangle$, 则 $\forall xH \in G/H, \exists y \in xH, o(xH) = o(y)$ 。

证明. 令 $o(a) = p^t, o(x) = p^s, o(xH) = p^k$, 显然有 $t \geq s \geq k$ 。

考虑 $x^{p^k}H = (xH)^{p^k} = H$, 于是 $x^{p^k} \in H$ 。不妨设 $x^{p^k} = a^r$ 。

$$o(x^{p^k}) = \frac{o(x)}{(o(x), p^k)} = p^{s-k}$$

$$o(a^r) = \frac{o(a)}{(o(a), r)}$$

$$\Rightarrow (o(a), r) = p^{t-s+k}$$

不妨设 $r = up^{t-s+k}$

构造: $y = xa^i \in xH$, 其中 $i = -up^{t-s}$ 。

$$\begin{aligned}
 & (yH)^{p^k} \\
 &= y^{p^k} H \\
 &= x^{p^k} a^{-up^{j-s+k}} H \\
 &= H
 \end{aligned}$$

□

引理 4.4. 任意有限交换 p 群 G 都可以被分解。

证明. 考虑归纳法, 设 $|G| = p^n$, 并且任意群 $G' (|G'| < p^n)$ 均存在分解。

找到阶最大的元素 $a \in G, o(a) = p^k$, 若 $k = n$, 则 $[a]$ 即是分解。

令 $H = \langle a \rangle, G' = G/H$, 根据归纳假设, G' 存在分解, 不妨设为 $[a_1H, a_2H, \dots, a_mH]$, 由引理 4.3, 设 $o(a_i) = o(a_iH) (i \in [1, m])$ 。

构造: $[a, a_1, a_2, \dots, a_m]$ 即为一个分解。

(D1) 考虑任意 $x \in G, xH \in G/H$ 。

$$xH = \prod_{i=1}^m (a_iH)^{k_i} = \left(\prod_{i=1}^m a_i^{k_i} \right) H$$

因此 $x^{-1} \prod_{i=1}^m a_i^{k_i} \in H$, 设为 a^k 。

则 $x = a^{-k} \prod_{i=1}^m a_i^{k_i}$ 。

(D2) 考虑 $a^k \prod_{i=1}^m a_i^{k_i} = e$, 则 $a^{-k} \prod_{i=1}^m a_i^{k_i} \in H$ 。

于是 $\prod_{i=1}^m (a_iH)^{k_i} = H$, 则 $(a_iH)^{k_i} = H (i \in [1, m])$ 。

根据 a_i 的性质有 $a_i^{k_i} = e (i \in [1, m])$, 则 $a^{-k} = e$ 即 $a^k = e$ 。

□

定理 4.1. 任意有限交换群都可以被分解。

根据引理 4.2 和引理 4.4 显然。

4.1.3 较为简单的实现

上述证明实际上给出了一个构造过程, 但这仍过于复杂。

定理 4.2. 对于交换群 G , 维护集合 B , 初始为空集, $S = \text{span}(B)$, 定义 $f(x) = \min\{k | x^k \in S\}$ 。不断进行操作:

- 若 $G = S$, 退出。
- 找出所有 $x \in G, x \notin S, o(x) = f(x)$ 中, $f(x)$ 最大值对应的元素 a 。

• $B := B + a, S = \text{span}(B)$ 。

得到的 B 是一个分解。

考虑这个更强的命题：

引理 4.5. 对于 $n \geq 0$ ，某一时刻 $B = [a_1, a_2, \dots, a_n]$ ，则存在分解 $[a_1, a_2, \dots, a_m] (m \geq n)$ 满足 $o(a_i) = p_i^{k_i} (n < i \leq m, p_i \in \mathbb{P}, k_i \in \mathbb{Z}^+)$ 。

证明. 考虑归纳，对 $n = 0$ 显然成立。

设第 n 个元素加入前合法分解为 $B' = [a'_1, \dots, a'_{m'}]$ 。

对于元素 x ，若 $x = \prod_{i=1}^{m'} a_i^{t_i}$ ，则

$$f(x) = \text{lcm}_{i=n}^{m'} \frac{o(a'_i)}{(o(a'_i), t_i)}$$

考虑 $f(x)$ 取最大值时，对于每一种出现的 p ，存在 i 使得 $(o(p^{k_i}), t_i) = 1$ 且 $k_i = \max_{p_j=p} k_j$ ，并且构造 $x = \prod_{i=n}^{m'} a'_i$ ，则有 $o(x) = f(x), x \notin S$ 且 $f(x) = \max_{y \in G \setminus S} \{f(y)\}$ 。

构造： $B = [a'_1, a'_2, \dots, a'_{n-1}, x] + R$ ，其中 R 是 B' 删去前 $n-1$ 项，并对于每种出现的 p 删除一个 i 后的结果。

不难证明 $\text{span}(B)$ 中包含删去的元素。因此 $\text{span}(B) = G$ ，考虑 B 中元素的阶，显然这是一组分解。 $\square$

引理 4.6. 得当的实现下，这么做的复杂度是 $O(|G|^2)$ 。

证明. 复杂度瓶颈在于计算 $f(x)$ 。

而第 n 轮，有：

$$\begin{aligned} f(x) &= \text{lcm}_{i=n}^m \frac{o(a_i)}{(o(a_i), t_i)} \\ &\leq \prod_{i=n}^m o(a_i) \\ &= \frac{|G|}{\prod_{i=1}^{n-1} o(a_i)} \\ &\leq \frac{|G|}{2^{n-1}} \end{aligned}$$

因此第 n 轮计算 $f(x)$ 的复杂度不超过 $O(\frac{|G|^2}{2^n})$ ，总复杂度为 $O(|G|^2)$ 。 $\square$

4.1.4 有限交换半群

定理 4.3. $G \rightarrow \mathbb{C}$ 存在 DFT 等价于 G 满足循环律。

证明及构造 [6] 不详细展开。

5 对称群

5.1 群表示

定义 5.1. 定义两个线性变换的乘法为两个线性变换的相继作用（即矩阵乘法）。则对于 n 维线性空间 V ，全部非奇异线性变换（即行列式不为 0）构成 n 维一般线性群，记作 $GL(V)$ ，其子群 $L(V)$ 称为 V 上的线性变换群。

定义 5.2. 对于群 G ，如果存在一个从 G 到线性空间 V 上的线性变换群 $L(V)$ 的同态映射 ρ ，则同态映射 ρ 称为群 G 的一个线性表示， V 为表示空间， V 的维数称为表示的维数，记为 d_ρ 。

定义 5.3. 若群 G 在空间 V 中的两个表示 ρ, ρ' 分别取基：

$$\mathbf{e} = (e_1, e_2, \dots, e_n), \mathbf{e}' = (e'_1, e'_2, \dots, e'_n)$$

令 $\mathbf{e} = \mathbf{e}'X$ ，若 $\det X \neq 0$ 且 $\forall g \in G, \rho(g) = X^{-1}\rho'(g)X$ ，称 ρ, ρ' 等价。

定义 5.4. 若 ρ 是群 G 在线性空间 V 上的一个表示， V 如果不存在子空间 $W \subset V$ 满足 $W \neq 0, W \neq V, \forall g \in G, \mathbf{x} \in W, \rho(g)\mathbf{x} \in W$ ，称 ρ 是 G 的一个不可约表示。

记群 G 的所有不等价不可约表示构成的集合为 $\mathcal{R}(G)$ 。

5.2 对称群 DFT

接下来将给出对群 S_n 进行 DFT 的算法。限于篇幅及笔者水平，将不给出具体证明和实现 [1] [2] [3]。

定理 5.1.（勃恩赛德定理）对于有限群 G ，有：

$$\sum_{\rho \in \mathcal{R}(G)} d_\rho^2 = |G|$$

这个定理证明 DFT 后，信息量足够。

引理 5.1. 对于有限群 G ，以下是一个合法的 DFT 构造：

- $a_i = d_{\rho_i}$
- $\mathcal{F}(A)_i = \sum_{g \in G} A_g \rho_i(g)$

这说明了群表示和 DFT 之间有非常密切的联系。如果已知 $\mathcal{R}(G)$ ，则可以在 $O(|G|^2)$ 时间内完成 DFT，但是遗憾的是，目前不存在一个通用的算法可以求出任意有限群 G 对应的 $\mathcal{R}(G)$ 。

引理 5.2. 在上述构造下, IDFT 的另一种计算方式为:

$$A_g = \frac{1}{|G|} \sum_{\rho_i \in \mathcal{R}(G)} d_{\rho_i} \text{Tr}\{\rho(g^{-1})\mathcal{F}(G)_i\}$$

引理 5.3. $\mathcal{R}(S_n)$ 和 n 的划分间存在一一对应关系。

接下来将给出一个称为杨氏半正规表示 (Young's seminormal representation) 的构造。

定义 5.5. 对于两个形状相同的 n 阶标准杨表 T_1, T_2 , 定义它们的末标号序为: 若 n 在 T_1, T_2 中的行号不同, 则 $T_1 < T_2$ 当且仅当 n 在 T_1 中的行号更小, 否则令 T'_1, T'_2 分别为 T_1, T_2 删去元素 n 后的杨表, $T_1 < T_2$ 当且仅当 $T'_1 < T'_2$ 。

定义 5.6. 对于一个杨表 T , 元素 t_1, t_2 所在行分别为 x_1, x_2 , 所在列分别为 y_1, y_2 , 则轴向距离 $d(t_1, t_2) = x_2 - x_1 + y_1 - y_2$ (注意没有绝对值)。

引理 5.4. n 阶杨表的形状和 n 的划分存在一一对应。

考虑杨表每一行长度即可。

定义 5.7. 对于划分 $\lambda \vdash n$, 对应的杨表按照末标号序排为 T 。记 T_i 上 a, b 的轴向距离为 $d_i(a, b)$, 接下来给出 $\rho((t, t+1))$ 的构造, 记 $\rho((t, t+1)) = a$ 。

- a 的维数为 k 。
- $a_{i,i} = d_i(t, t+1)^{-1}$
- 若 T_i 交换元素 t 和 $t+1$ 后仍为合法标准杨表, 设为 T_j , 则:

$$a_{i,j} = \begin{cases} 1 - d_i(t, t+1)^{-2} & i < j \\ 1 & i > j \end{cases}$$

- 其余位置均设为 0。

对于任意排列 p , 由冒泡排序可证, 可以分解为若干 (t_i, t_i+1) 的累乘, 合并出 $\rho(p)$ 即可。

可以证明, 这是一组不可约不等价表示。直接计算, DFT 的代价为 $O(|G|^2)$ 。

5.3 更优的复杂度

考虑 S_n 的子群 $H = \{\pi | \pi_n = n\} \simeq S_{n-1}$, 记 $\{s_i H | s_i \in S_n\}$ 构成 S_n 的一组陪集划分。

将 DFT 定义式变形, 有:

$$\mathcal{F}(A)_i = \sum_{g \in G} A_g \rho_i(g) = \sum_{t=1}^n \rho_i(s_t) \sum_{x \in H} A'_{t,x} \rho_i(x)$$

其中 $A'_{t,i} = A_{s_t x}$ 。

定理 5.2. 对于 $x \in H$, $\rho_i(x)$ 的形式为:

$$\begin{bmatrix} \rho'_{j_1}(x) & 0 & \cdots & 0 \\ 0 & \rho'_{j_2}(x) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \rho'_{j_k}(x) \end{bmatrix}$$

其中 ρ'_{j_i} 表示, 划分 (或杨表形状) λ_i 删去一个元素后, 可以得到 $n-1$ 阶杨表形状 λ'_{j_i} 。

考察 n 在杨表中的位置, n 位置相同的在 T 中一定连续。由于 $\pi_n = n$, 删去 n 并不影响构造过程。归纳证明即可。

因此对于 $A' : S_n \rightarrow \mathbb{C}$, 可以将它分解为 $(A'_i)_{i=1}^n$ 并递归为 $S_{n-1} \rightarrow \mathbb{C}$ 的子问题, 再通过 n 次矩阵序列乘法合并。

引理 5.5. $C_t = \rho((t, t+1))(t \in [1, n-2])$ 和 $D_t = \sum_{x \in H} A'_{t,x} \rho_i(x) (t \in [1, n])$ 中, 非零元素个数之和是 $O(n!)$ 的。

证明. 考虑每一种划分 (或杨表形状) $\lambda \vdash n-1$, 它至多对每个 C_t 和 D_t 中贡献一次, 即最多出现 $2n-2$ 次。

而我们知道 $\sum_{\lambda \vdash n-1} d_{\rho_\lambda}^2 = (n-1)!$, 因此总贡献不超过 $O(n!)$ 。 $\square$

对于 s_i , 我们挑选 $s_i = (n, n-1, \dots, i)$, 则 $s_i = s_{i+1} \times (i, i+1)$, 只需 n 次矩阵序列乘法即可求出 $\rho(s_i)$, 注意由于引理 5.5, 矩阵乘法的总复杂度不超过 $O(n!n)$ 。

因此, 对 S_n DFT 的复杂度为 $T(n) = nT(n-1) + O(n!n^2) = O(n!n^3)$, 显著优于暴力的 $O((n!)^2)$ 。

这个算法通过类似 FFT 的分治, 把不同情况递归到相同子问题从而降低复杂度, 对于更一般的群, 如果可以找到类似的构造, 也可以在较低复杂度完成 DFT。

6 总结

DFT 在 OI 中已经被广泛应用, 但是长久以来 DFT 的应用都仅限于序列卷积, 近年来扩展到了集合幂级数。但是对于更一般的卷积定义, OI 界并没有过多的研究。本文展示了不交换的群同样可以在较高的复杂度下完成卷积, 并指出 DFT 与群表示的关系。笔者仅在文中展现了对称群的 DFT, 并没有展现它的应用, DFT 更深层的意义及更多样的应用有待感兴趣的同学研究。

致谢

感谢中国计算机学会提供学习和交流的平台。



感谢徐先友老师的关心和指导。

感谢许庭强同学，马耀华学长关于内容的帮助。

感谢许庭强同学，罗恺同学，金策学长，徐寅展学长为本文验稿。

参考文献

- [1] Michael Clausen; Ulrich Baum. Fast fourier transforms for symmetric groups: theory and implementation. Math. Comp. 61 (1993), 833-847, 1993.
- [2] Risi Kondor. A c++ library for fast fourier transforms on the symmetric group.
- [3] Persi Diaconis; Daniel Rockmore. Efficient computation of the fourier transform on finite groups. J. Amer. Math. Soc. 3 (1990), 297-332, 1990.
- [4] Audrey Terras. Fourier Analysis on Finite Groups and Applications. Cambridge University Press, 1999.
- [5] Josh Alman; Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms, 2020.
- [6] 刘承奥. 浅谈 dft 在信息学竞赛中的应用. IOI2018 中国国家候选队论文集, 2018.



浅谈一种基于比较的贪心思想在信息学竞赛中的应用

成都外国语学校 张隽恺

摘要

本文介绍了一种基于比较的贪心思想在信息学竞赛中的应用。本文首先讲述了问题形式、该方法的过程以及使用该方法需要的一种条件，接着引入了一类树上形式与上一类问题类似的问题，并给出了这类问题基于上述贪心思想的算法。最后本文对信息学竞赛中的满足上述条件的常见模型和算法部分性质进行了分析。

1 引言

在信息学竞赛中，贪心思想的应用是极其重要的一个部分。在贪心思想中，存在一类通过分析在解中交换两个元素对答案的影响，从而得到最优解满足的性质的方法，这种方法通常被称为“Exchange Arguments”。这类方法在十年前就已经在信息学竞赛中出现，近年来也出现了一些该算法的拓展例子。但关于该算法的分析和研究仍然较少。本文将对这种算法进行较为详细的分析。

本文的第一部分通过一个例子引入了该方法，接着分析了该方法可以得到正确的最优解的一种充分条件。第二部分介绍了这种问题的一个经典拓展——一类在之前的问题上加入一种形如有根树的限制的问题，接着给出了一种与之前类似但更强的条件，以及问题满足该条件时问题的两种算法。两部分分别对几种常见的模型进行了讨论，并分析了该部分算法的部分性质及应用。

2 一类与排列相关的最优化问题及 Exchange Arguments 方法

2.1 问题引入

在本文的这一部分中，主要讨论如下形式的问题：

给定 n 个元素 $x_1, \dots, x_n$ ，以及一个定义域为这些元素的序列，值域为有序集合的函数 F 。求出对于所有的 n 阶排列 p ，如下表达式的最小值：

$$F(\{x_{p_1}, x_{p_2}, \dots, x_{p_n}\})$$

例如如下问题:

例题 1.¹

给出 n 个二元正整数对 (a_i, b_i) , 将它们按照任意顺序排成一列。定义排成一列的代价为每个二元组的 a 乘上序列中这个二元组之后的所有二元组的 b 之和的总和。求最小的代价。

$$n \leq 10^6, 1 \leq a_i, b_i \leq 10^6$$

这可以看成上述问题的一种情况, 其中 $F(\{(a_1, b_1), (a_2, b_2), \dots, (a_k, b_k)\}) = \sum_{1 \leq i < j \leq k} a_i b_j$, 显然实数中可以进行比较。

分析最优的排列满足的性质。考虑一个排列 $(p_1, \dots, p_n)$ 以及在这个排列中交换两个相邻位置 $(i, i+1)$ 得到的排列, 即 $(q_1, \dots, q_n) = (p_1, \dots, p_{i-1}, p_{i+1}, p_i, p_{i+2}, \dots, p_n)$, 则可以得到:

$$F(\{(a_{p_1}, b_{p_1}), \dots, (a_{p_n}, b_{p_n})\}) - F(\{(a_{q_1}, b_{q_1}), \dots, (a_{q_n}, b_{q_n})\}) = a_{p_i} b_{p_{i+1}} - a_{p_{i+1}} b_{p_i} \quad (1)$$

因而如果 $a_{p_i} b_{p_{i+1}} - a_{p_{i+1}} b_{p_i} > 0$, 则 $F(\{(a_{q_1}, b_{q_1}), \dots, (a_{q_n}, b_{q_n})\}) < F(\{(a_{p_1}, b_{p_1}), \dots, (a_{p_n}, b_{p_n})\})$, 则 $(p_1, \dots, p_n)$ 不是最优解。因此只有满足 $\forall 1 \leq i < n, a_{p_i} b_{p_{i+1}} - a_{p_{i+1}} b_{p_i} \leq 0$, 即 $\frac{a_{p_i}}{b_{p_i}} \leq \frac{a_{p_{i+1}}}{b_{p_{i+1}}}$ 的排列可能是最优解。

如果一个排列 p 满足这样的限制, 则 p 满足 $\forall 1 \leq i < j \leq n, \frac{a_{p_i}}{b_{p_i}} \leq \frac{a_{p_j}}{b_{p_j}}$ 。因此对于任意数 v , 所有满足 $\frac{a_i}{b_i} = v$ 的 i 在排列中一定构成连续的一段。不同的段在排列中按照 $\frac{a_i}{b_i}$ 从小到大排序。对于任意的 i, j , 如果 $\frac{a_i}{b_i} = \frac{a_j}{b_j}$, 则由 1 式, 如果排列中 i, j 相邻, 交换它们后得到的排列的 F 保持不变。因此可以得到, 对于排列中满足内部所有 $\frac{a_i}{b_i}$ 相等的一段, 这一段内任意交换不改变 $F(\{(a_{p_1}, b_{p_1}), \dots, (a_{p_n}, b_{p_n})\})$ 。因此所有满足上述条件的排列 p 的 $F(\{(a_{p_1}, b_{p_1}), \dots, (a_{p_n}, b_{p_n})\})$ 相同, 从而它们都是最优解。

因此对于这个问题, 只需要将所有的二元组按照 $\frac{a_{p_i}}{b_{p_i}}$ 排序, 即可得到一种最优的排成一列的方式。时间复杂度为排序的时间复杂度 $O(n \log n)$ 。

2.2 问题分析与 Exchange Arguments 方法

考虑分析上一个问题中的做法。在这个问题中, 使用的做法可以看作对于每两个元素 x_i, x_j 间建立了比较关系, 比较关系可以看成 $F(\{x_i, x_j\}) \leq F(\{x_j, x_i\})$ 。在这道题中, 这样定义的比较关系以及原问题中满足一些特殊的性质, 使得将所有元素按照比较关系排序可以得到一种最优解。但对于更一般的情况, 上述做法显然不再成立。事实上对于一些 F , 上述形式的问题目前只有指数级的算法。因此考虑分析上一个问题的性质, 得到使得如下算法可以找到最优解的一种条件:

¹题目来源: 经典问题

设给出的所有元素构成的集合为 S ，定义 S 上的一种二元比较关系 $\leq$ ，将所有元素按照比较关系排序。

在上一个问题中，不难发现存在如下性质：

性质 2.1. $\forall a, b \in S, a \leq b, b \leq a$ 至少有一者成立。即比较关系满足强完全性。

性质 2.2. $\forall a, b, c \in S$ ，如果 $a \leq b, b \leq c$ ，则 $a \leq c$ 。即比较关系满足传递性。

性质 2.3. $\forall a, b \in S$ ，如果 $a \leq b$ ，则对于任意一个包含 $\{a, b\}$ 作为连续子序列的元素序列 $\{s_1, \dots, s_{k-1}, a, b, s_{k+2}, \dots, s_l\}$ ，都有：

$$F(\{s_1, \dots, s_{k-1}, a, b, s_{k+2}, \dots, s_l\}) \leq F(\{s_1, \dots, s_{k-1}, b, a, s_{k+2}, \dots, s_l\})$$

存在如下结论：如果 F 满足存在一种 S 上的一种二元比较关系 $\leq$ 满足性质 2.1, 2.2, 2.3，则在上述算法中使用这种比较关系，上述算法可以得到问题的一个最优解。

证明. 由假设，则 $\leq$ 构成 S 上的一个弱序关系 (Weak Ordering[1]，也被称为优先关系)。因此，设 $|S| = n$ ，存在一种 S 中元素排成序列 $\{x_1, \dots, x_n\}$ 的方式，满足 $\forall 1 \leq i < n, x_i \leq x_{i+1}$ 。对于一个满足弱序关系的比较关系 $\leq$ ，使用基于比较的排序算法，即可在 $O(n \log n)$ 次比较中得到一个这样的序列。

此时有如下引理：

Lemma 2.1. 如果 S 中元素的一种排列 $\{x_1, \dots, x_n\}$ 满足 $\forall 1 \leq i < n, x_i \leq x_{i+1}$ ，且 F 满足性质 2.3，则不存在一个 S 中元素的排列 $\{y_1, \dots, y_n\}$ ，使得 $F(\{y_1, \dots, y_n\}) < F(\{x_1, \dots, x_n\})$ 。即 $F(\{x_1, \dots, x_n\})$ 是所有可能的排列对应的函数值中最小的。

证明. 由 $\leq$ 的传递性，有 $\forall 1 \leq i < j \leq n, x_i \leq x_j$ ，设 $\{y_1, \dots, y_n\} = \{x_{p_1}, \dots, x_{p_n}\}$ 。对 $\{x_{p_1}, \dots, x_{p_n}\}$ 进行 n 轮交换，第 i 轮交换通过若干次交换 x_i 以及与其相邻的一个元素的顺序，将 x_i 交换至第 i 个位置， n 轮交换后即可得到 $\{x_1, \dots, x_n\}$ ，可以得到第 i 轮交换中的每次交换一定是交换 x_i 以及 x_i 的前一个元素，且设前一个元素为 x_t ，则一定有 $t > i$ ，从而 $x_i \leq x_t$ 。由性质 2.3，这样交换相邻两个元素后， F 不会增大。因此进行所有交换后 F 不会增大，从而有 $F(\{x_1, \dots, x_n\}) \leq F(\{x_{p_1}, \dots, x_{p_n}\})$ 。因此引理成立。 $\square$

由此，满足 $\forall 1 \leq i < n, x_i \leq x_{i+1}$ 的排列 $\{x_1, \dots, x_n\}$ 是问题的一种最优解。 $\square$

这也证明了上一个问题解法的正确性。

2.3 部分例子与对应模型

2.3.1 一些例子

还有其它一些问题满足类似的形式，下面列举了几个例子。

例题 2.²

给出 n 个包含小写字母的字符串 $s_1, \dots, s_n$ 。找到一个 n 阶排列 p , 将 $s_{p_1}, s_{p_2}, \dots, s_{p_n}$ 顺序拼接得到 S , 使 S 的字典序最小。

$$\sum |s_i| \leq 5 \times 10^5$$

此时设 F 为将字符串顺序拼接的结果, 字符串间的字典序比较显然合法。此时仍然令 $x \leq y$ 当且仅当 $F(\{x, y\}) \leq F(\{y, x\})$, 则此时定义的比较关系为: 对于字符串 s, t , $s \leq t$ 当且仅当 $s+t$ 的字典序小于等于 $t+s$ 的字典序 (这里 $s+t$ 表示字符串 s, t 按顺序拼接得到的结果)。显然 $\leq$ 满足性质 2.1。字典序的比较关系满足在两个长度相同的字符串开头或结尾增加相同字符不影响字典序大小关系, 因此 $\leq$ 满足性质 2.3, 只需证明比较关系满足传递性, 即性质 2.2。此时有如下引理:

Lemma 2.2. 记 S^∞ 为将字符串 S 无限重复所得到的字符串, 则 $S+T$ 的字典序小于等于 $T+S$ 的字典序当且仅当 S^∞ 的字典序小于等于 T^∞ 的字典序。

该引理的证明较为复杂, 且与本文其余部分关系不大, 故在此略去证明。

字典序的比较关系具有传递性, 因此使用上述做法得到的 $\leq$ 关系具有传递性, 因此比较关系满足上述三条性质。使用上述做法, 使用后缀数组或类似的算法支持预处理后 $O(1)$ 询问两个给出串的后缀的最长公共前缀, 即可对于两个给出的字符串 S, T 在 $O(1)$ 的复杂度内比较 $S+T$ 与 $T+S$ 的字典序大小。后缀数组的实现时间复杂度为 $O((\sum |s_i|) \log(\sum |s_i|))$ 。

例题 3.³

有 n 块硬盘, 需要对这些硬盘进行格式化。第 i 块硬盘在格式化前容量为 a_i , 格式化后容量为 b_i 。初始时每块硬盘中都装满了数据, 但在格式化一块硬盘的过程中, 需要将这块硬盘上的所有数据移动到其他硬盘或者存储空间中。数据可以进行任意转移, 任意分割。求最少需要多少的额外存储空间, 才能格式化所有的硬盘并保留数据。

$$n \leq 10^6, 0 \leq a_i, b_i \leq 10^9$$

设第 i 次格式化的硬盘为第 p_i 块。则在格式化第 p_i 块硬盘前, 需要的额外空间为 $\max(0, (\sum_{j=1}^{i-1} b_{p_j}) - (\sum_{j=1}^i a_{p_j}))$ 。因此问题相当于给定 n 对 (a_i, b_i) , 找到它们的一个排列, 最大化 $\min_{i=1}^n \{(\sum_{j=1}^{i-1} b_i) - (\sum_{j=1}^i a_j)\}$ 。记 $F((a_1, b_1), \dots, (a_n, b_n)) = -\min_{i=1}^n \{(\sum_{j=1}^{i-1} b_j) - (\sum_{j=1}^i a_j)\}$, 即可看成最小化 F 。

沿用之前的方法, 定义 $x \leq y$ 当且仅当 $F(\{x, y\}) \leq F(\{y, x\})$, 对于两个元素 $(a_1, b_1), (a_2, b_2)$, $(a_1, b_1) \leq (a_2, b_2)$ 当且仅当 $\min\{-a_1, b_1 - a_1 - a_2\} \geq \min\{-a_2, b_2 - a_1 - a_2\}$ 。考虑这样定义的 $\leq$ 关系是否满足上述性质。对于性质 2.3, 设最后的序列为 $(a_1, b_1), \dots, (a_n, b_n)$, 交换的元素为

² 题目来源: 经典问题

³ 题目来源: 2016 ACM-ICPC World Finals L Swap Space

$(a_t, b_t), (a_{t+1}, b_{t+1})$, 且 $(a_t, b_t) \leq (a_{t+1}, b_{t+1})$, 则:

$$\begin{aligned}
 F((a_1, b_1), \dots, (a_n, b_n)) &= -\min_{i=1}^n \left(\sum_{j=1}^{i-1} b_j - \left(\sum_{j=1}^i a_j \right) \right) \\
 &= -\min \left(\sum_{j=1}^{t-1} b_j - \left(\sum_{j=1}^t a_j \right) + \min \{-a_t, b_t - a_t - a_{t+1}\}, \min_{1 \leq i \leq n, i \neq t, t+1} \left(\sum_{j=1}^{i-1} b_j - \left(\sum_{j=1}^i a_j \right) \right) \right) \\
 F((a_1, b_1), \dots, (a_{t-1}, b_{t-1}), (a_{t+1}, b_{t+1}), (a_t, b_t), (a_{t+2}, b_{t+2}), \dots, (a_n, b_n)) \\
 &= -\min \left(\sum_{j=1}^{t-1} b_j - \left(\sum_{j=1}^t a_j \right) + \min \{-a_{t+1}, b_{t+1} - a_{t+1} - a_t\}, \min_{1 \leq i \leq n, i \neq t, t+1} \left(\sum_{j=1}^{i-1} b_j - \left(\sum_{j=1}^i a_j \right) \right) \right)
 \end{aligned}$$

因为 $\min \{-a_t, b_t - a_t - a_{t+1}\} \geq \min \{-a_{t+1}, b_{t+1} - a_{t+1} - a_t\}$, 所以上式中前者小于等于后者, 因此性质 2.3 成立。

分析此时大小关系的性质, 即 $\min \{-a_1, b_1 - a_1 - a_2\} \geq \min \{-a_2, b_2 - a_1 - a_2\}$ 。如果这个不等式成立, 则说明不等式左侧的两个元素都大于等于右侧的某一个元素, 因此上述条件等价于满足以下两者之一:

1. 如果都大于等于第一个元素, 则相当于 $a_1 \leq a_2$ 且 $b_1 - a_1 \geq 0$ 。
2. 如果都大于等于第二个元素, 则相当于 $b_1 \geq b_2$ 且 $b_2 - a_2 \leq 0$ 。

由此可以考虑将所有元素按照 $b_i - a_i$ 的符号, 即 $\text{sgn}(b_i - a_i)$ ⁴ 分成三类。此时有如下引理:

- Lemma 2.3.**
1. 如果 $\text{sgn}(b_1 - a_1) > \text{sgn}(b_2 - a_2)$, 则 $\min \{-a_1, b_1 - a_1 - a_2\} \geq \min \{-a_2, b_2 - a_1 - a_2\}$ 成立。
 2. 如果 $\text{sgn}(b_1 - a_1) = \text{sgn}(b_2 - a_2) = 1$, 则 $\min \{-a_1, b_1 - a_1 - a_2\} \geq \min \{-a_2, b_2 - a_1 - a_2\}$ 当且仅当 $a_1 \leq a_2$ 。
 3. 如果 $\text{sgn}(b_1 - a_1) = \text{sgn}(b_2 - a_2) = 0$, 则 $\min \{-a_1, b_1 - a_1 - a_2\} \geq \min \{-a_2, b_2 - a_1 - a_2\}$ 成立。
 4. 如果 $\text{sgn}(b_1 - a_1) = \text{sgn}(b_2 - a_2) = -1$, 则 $\min \{-a_1, b_1 - a_1 - a_2\} \geq \min \{-a_2, b_2 - a_1 - a_2\}$ 当且仅当 $b_1 \geq b_2$ 。

$${}^4 \text{sgn}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$$

证明. 如果 $\text{sgn}(b_1 - a_1) > \text{sgn}(b_2 - a_2)$, 则 $b_1 - a_1 \geq 0, b_2 - a_2 \leq 0$ 都成立. 如果 $a_1 \leq a_2, b_1 \geq b_2$ 都不成立, 则有 $b_1 - a_1 \leq b_2 - a_2$, 但这与 $\text{sgn}(b_1 - a_1) > \text{sgn}(b_2 - a_2)$ 矛盾, 因此此时上述两种条件一定有一种得到满足. 对于 sgn 相同的情况, 由上述两种条件可以直接推出结论. $\square$

由引理 2.3 不难发现, 如果按照 $\text{sgn}(b_i - a_i)$ 将所有元素分成三类, 则在每一类内部, $\leq$ 都定义了合法的弱序关系. 但此时在不同类之间, $\leq$ 定义的关系不一定满足传递性. 例如, $(2, 2) \leq (1, 1) \leq (1, 2)$, 但 $(2, 2) \not\leq (1, 2)$. 因此这样的比较关系不满足性质 2.2.

但对于不同类之间的情况, 如果 $\text{sgn}(b_1 - a_1) > \text{sgn}(b_2 - a_2)$, 则一定有 $(a_1, b_1) \leq (a_2, b_2)$. 因此考虑使用如下方式定义 $\leq$:

$(a_1, b_1) \leq (a_2, b_2)$ 当且仅当满足如下两个条件之一:

1. $\text{sgn}(b_1 - a_1) > \text{sgn}(b_2 - a_2)$.
2. $\text{sgn}(b_1 - a_1) = \text{sgn}(b_2 - a_2)$ 且 $F((a_1, b_1), (a_2, b_2)) \leq F((a_2, b_2), (a_1, b_1))$

容易验证使用这种方式定义的 $\leq$ 满足三条性质. 因此使用这样定义的 $\leq$ 排序, 即可使用上述算法求解. 时间复杂度 $O(n \log n)$.

这个例子说明, 由于性质 2.3 中给出的是蕴涵关系而不是双向关系, 因而不一定可以直接使用 $F((x, y)) \leq F((y, x))$ 定义比较关系. 但由性质 2.3, 如果 $x \leq y$ 则一定有 $F((x, y)) \leq F((y, x))$, 因此从这个比较关系出发仍然是一种好的思路.

在上一个例子中, 不难发现如果没有 $a_i, b_i \geq 0$ 的条件, 上述结论仍然成立. 此时它可以表示下面这个例子:

例题 4.⁵

有 n 个箱子, 第 i 个箱子有重量 w_i 和承载量 $v_i (w_i, v_i > 0)$, 将它们堆成一列使得每一个箱子上的箱子总重量都不大于它的承载量.

$$n \leq 10^6$$

对于一个箱子的序列 $\{(w_1, v_1), \dots, (w_n, v_n)\}$, 它满足要求当且仅当 $\min_{i=1}^n \{v_i - \sum_{j=1}^{i-1} w_j\} \geq 0$. 考虑最大化 $\min_{i=1}^n \{v_i - \sum_{j=1}^{i-1} w_j\} = \min_{i=1}^n \{\sum_{j=1}^{i-1} (-v_j - w_j) - \sum_{j=1}^i (-v_j)\}$. 则问题可以看成上一个例题中的形式, 其中 $a_i = -v_i, b_i = -v_i - w_i$, 此时所有的 $b_i - a_i < 0$. 因此一种最优方案为将所有箱子按照 b_i 从大到小排序, 即按照 $v_i + w_i$ 从小到大排序.

类似地, 对于例题 1 中的问题, 如果将限制改为 $a, b \geq 0$ 或者 $a \geq 0$ 或者类似的条件, 则对一些特殊元素进行处理后, 可以通过类似于比较 $a_i b_j, a_j b_i$ 的大小关系来比较 $(a_i, b_i), (a_j, b_j)$, 使得比较关系满足性质 2.1, 2.2, 2.3. 因此这种问题仍然可以使用上述算法求解. 但如果没有限制, 变为 a, b 为任意实数, 则不难找到一组反例使得三条性质一定不能同时成立, 因而此时的问题不再可以使用之前的算法.

⁵ 题目来源: 改编自经典问题

2.3.2 部分模型

上文中给出的三个例子分别对应三种不同的元素、函数以及元素间的比较关系，这里将一组元素、函数以及比较关系称为模型，则上述例子对应了三种模型：

1. 元素形如 $(a, b) (a, b \in \mathbb{R}, a \geq 0)$, $F(\{(a_1, b_1), (a_2, b_2), \dots, (a_k, b_k)\}) = \sum_{1 \leq i < j \leq k} a_i b_j$, 比较 F 的方式为实数比较。元素间具体的比较方式见上。
2. 元素形如 $(a, b) (a, b \in \mathbb{R})$, $F(\{(a_1, b_1), \dots, (a_n, b_n)\}) = -\min_{i=1}^n \{(\sum_{j=1}^{i-1} b_j) - (\sum_{j=1}^i a_j)\}$, 比较 F 的方式为实数比较。元素间具体的比较方式见例题 3。
3. 元素形如字符串 s , $F(\{s_1, \dots, s_n\}) = s_1 + \dots + s_n$, 比较 F 的方式为字典序比较。比较两个元素 s, t 的方式为比较 $s+t, t+s$ 的字典序。

上述几种模型（尤其是前两种）在信息学竞赛中有较为广泛的应用。还有很多问题可以转化为上面的几种模型或者其它满足上述性质的模型（例如， $F(\{(a_1, b_1), \dots, (a_n, b_n)\}) = \sum_{i=1}^n a_i \prod_{j=i+1}^n b_j$, 其中 $a_i, b_i > 0$ ），在此不再展开。

2.4 一些该方法的其它应用

使用上述算法中的性质和处理方式，可以解决更多的问题。例如如下一类问题：

给定 n 个元素 $x_1, \dots, x_n$ ，以及一个定义域为这些元素的序列的函数 F 。该函数满足上述三条性质。选出 k 个元素，将它们任意排列成 $(y_1, \dots, y_k)$ ，最小化：

$$F(\{y_1, \dots, y_k\})$$

按照上述算法，可以建立 x 间的弱序关系，不妨设 $x_1 \leq x_2 \leq \dots \leq x_n$ 。设选择的元素为 $x_{s_1}, \dots, x_{s_k} (s_1 < s_2 < \dots < s_k)$ ，则对于选择元素的集合为 $\{x_{s_1}, \dots, x_{s_k}\}$ 的情况，由上述算法，序列 $\{x_{s_1}, \dots, x_{s_k}\}$ 是这种情况中的一种最优方案。因此一定存在一种最优方案是 $\{x_1, \dots, x_n\}$ 的子序列。根据这个性质，可以将按顺序选择若干个元素的问题变为选取序列的一个子序列的问题。这在一些情况下可以降低问题的难度，例如：

例题 5.⁶

有 n 个物品，第 i 个物品有非负费用 c_i 和价值 v_i ，两个人进行如下博弈：

1. 第一个人选择一个物品，付出 c_i 的代价。第一个人也可以选择直接让博弈结束。
2. 第二个人可以选择删除这个物品，这会使得博弈回到第一步（但第一个人已经付出的代价不会改变），第二个人最多可以进行 k 次这种操作。第二个人也可以选择不上操作，此时第一个人获得 v_i 的收益，博弈结束。

⁶ 题目来源：2014-2015 ACM-ICPC, Central Europe Regional Contest L The Imp

第一个人的总收益为收益减去付出的所有代价。第一个人希望最大化收益，第二个人希望最小化它。求双方最优操作下博弈的结果。

$$n \leq 1.5 \times 10^5, k \leq 9$$

如果第一个人选择不拿物品，则收益为 0。如果第一个人有更优的策略，则他一定会在博弈结束时得到一个物品的收益。此时第一个人的策略可以看成，选择一个由 $k+1$ 个物品组成的序列 $\{s_1, \dots, s_{k+1}\}$ ，第 i 次操作时选择第 s_i 个物品。如果第一个人的策略固定，则容易得到第二个人任意操作时，第一个人的最小总收益为：

$$\min_{i=1}^{k+1} \{v_{s_i} - \sum_{j=1}^i c_{s_j}\}$$

考虑对于第一个人所有的策略，上式可能的最大值。此时，第一个人通过选择这种策略，可以使总收益不小于这个值。同时，不存在一种方案使得 $\min$ 更大，因此对于第一个人的某种策略，一定存在一个物品使得如果博弈在他选择这个物品时结束，则他的总收益小于等于这个值，因此第二个人一定可以使得总收益不超过这个值。从而游戏的结果即为这个值，问题变为最大化这个值。这与例题 4 中的情况类似，由例题 4 做法可以得到一种最优方案是将所有物品按照 v_{s_i} 从小到大排序。将所有物品按照价值从小到大排序，则第一个人的策略一定是选择排序后的一个子序列。设 $f_{i,j}$ 表示在排序后 $[i, n]$ 这段后缀中选择 j 个物品，上式的最大值，则容易得到 $f_{i,j} = \max\{f_{i+1,j}, \min\{v_i - c_i, f_{i+1,j-1} - c_i\}\}$ 。该算法时间复杂度为 $O(nk)$ 。

2.5 总结

该算法的出发点在于分析在一组方案中交换相邻两个元素后，当前方案权值的变化，由此得到最优的方案需要满足的性质。因此它被称为“Exchange Argument”。而该算法的实质在于找到一种比较方式构成序关系，使得只要交换一对相邻且满足序关系中后者小于等于前者的元素，则交换后的方案的优秀度不会变差，因而可以推出满足序关系的序列一定是最优方案。直接使用交换后是否不会变差作为比较方式可能会出现问题，但可以发现从交换的比较关系出发是一种优秀的方案。上文中给出了几个可以使用这种算法的经典例子。这些例子之间看似没有关联，但它们都可以使用同一种方法求解，这说明这种算法存在广泛的应用。

同时该算法的思想，即分析最优方案中局部的性质从而推出最优方案中整体的性质，是一种分析问题性质的有力做法。关于该思想的更多分析在本文中不再展开。

3 问题在有根树上的拓展以及解法

3.1 问题引入

在这一部分中，主要讨论下面给出的一类问题，这类问题的形式与本文上一部分中给出的问题形式类似，但加入了一种与有根树相关的，类似偏序的限制。定义 y 在排列 p 中比 x 更先出现，当且仅当 $\exists 1 \leq i < j \leq n, p_i = y, p_j = x$ 。则这类问题可以表示为：

给定 n 个元素 $x_1, \dots, x_n$ ，以及一个定义域为这些元素的序列，值域为有序集合的函数 F 。再给出一棵 n 个点的有根树 T ，其中 1 为根。记点 $i (2 \leq i \leq n)$ 的父亲为 fa_i 。在接下来的部分中，为了使描述更简洁，本文使用“点 i 对应元素 p_i ”这种方式来描述元素与树中点的关系。

称一个排列满足 T 的限制，当且仅当 $\forall 2 \leq i \leq n, fa_i$ 在排列中比 i 更先出现。求出对于所有满足 T 的限制的 n 阶排列 p ，如下表达式的最小值：

$$F((x_{p_1}, x_{p_2}, \dots, x_{p_n}))$$

例如下述问题：

例题 6.⁷

给出一棵 n 个点的有根树，其中 1 为根。点 i 上有数字 $v_i (v_i \in \{0, 1\})$ 。现在需要将这些点排成一行 $p_1, \dots, p_n$ ，满足对于任意一个点 x ，在排列中 x 除去自身外的所有祖先都比 x 更先出现。最小化序列 $\{v_{p_1}, v_{p_2}, \dots, v_{p_n}\}$ 的逆序对数（满足 $1 \leq i < j \leq n$ 且 $v_{p_i} > v_{p_j}$ 的对数）。

$$n \leq 2 \times 10^5$$

在本题中， $v_{p_i} > v_{p_j}$ 当且仅当 $v_{p_i} = 1, v_{p_j} = 0$ ，因此如果将 $v_i = 1$ 的元素看作上一部分中第一种模型中的 $(1, 0)$ ，将 $v_i = 0$ 的元素看作 $(0, 1)$ ，则该问题可以被表述为上述形式。

如果不存在 T 的限制，则由上一部分的结果，只需要问题满足存在一种元素间的比较关系满足上一部分给出的三条性质，就可以使用 Exchange Arguments 方法求解。但在加入树的限制后，即使 F 满足这一条件，也可能不存在时间复杂度较优的算法，例如如下模型：

给出 n 个数 $v_1, \dots, v_n$ ，将它们排序，使得前 k 个数的和最大。

定义 $v_i \leq v_j$ 即为在实数意义的进行比较。这显然满足性质 2.1, 2.2，不难证明这样的比较关系也满足性质 2.3。因此 F 满足上文提出的限制。但考虑如下问题：

3.1.1 一种反例

给一棵 n 个点的有根树，点有点权。给定正整数 k ，选择一个树上包含根的大小为 k 的连通块，使得连通块内点权和最大。

⁷题目来源：AGC023F 01 on Tree

该问题等价于选择一个 n 个点的排列, 满足 T 的限制, 并最大化排列中前 k 个点的点权和。没有树上限制时该问题等价于上述模型。该问题可以使用动态规划的方式以 $O(n^2)$ 的时间复杂度解决, 且如果树形如根节点有三个子树、每个子树都是一条链, 则问题等价于给出三个任意序列 a, b, c , 求出 $\max_{i_1+i_2+i_3=k} a_{i_1} + b_{i_2} + c_{i_3}$, 因而这个问题难以在低于 $O(n^2)$ 的时间复杂度内解决。沿用上一部分中的思路考虑也难以解决问题。这说明如果希望使用上一部分中基于交换的贪心思路解决这类问题, 则需要 F 满足更强的性质以及对问题进一步的分析。

3.2 对树上问题的分析

设问题中存在 $\leq$ 满足性质 2.1, 2.2, 2.3, 则可以得到如下引理:

Lemma 3.1. 设元素为 $x_1, \dots, x_n$, 有根树的根为 1。设 v 满足 x_v 是 $x_2, \dots, x_n$ 中的最小元素: $\forall i \in \{2, \dots, n\}, x_i \leq x_v$, u 是 v 在树上的父亲, 则存在一种最优的排列, 使得 u, v 在排列中相邻, 即 $\exists i \in \{1, \dots, n-1\}, p_i = u, p_{i+1} = v$ 或者 $p_{i+1} = u, p_i = v$ 。

证明. 对于任意一个满足 T 的限制的排列 p , 设 a, b 满足 $p_a = u, p_b = v$, 易得 $a < b$ 。因为 p_1 一定是 T 的根, 因而 $\forall i \in \{a+1, \dots, b-1\}, x_v \leq x_{p_i}$ 。由性质 2.3, 有:

$$\begin{aligned} F(\{x_{p_1}, \dots, x_{p_{b-1}}, x_{p_b}, \dots, x_{p_n}\}) &\geq F(\{x_{p_1}, \dots, x_{p_b}, x_{p_{b-1}}, \dots, x_{p_n}\}) \geq \dots \\ &\geq F(\{x_{p_1}, \dots, x_{p_a}, x_{p_b}, x_{p_{a+1}}, \dots, x_{p_{b-1}}, x_{p_{b+1}}, \dots, x_{p_n}\}) \end{aligned}$$

如果 $\exists t \in \{a+1, \dots, b-1\}$ 使得 x_{p_t} 是 x_v 的祖先, 则因为 $p_t \neq v, u$, 而 u 是 v 的父亲, 因此 p_t 是 u 的祖先且 $p_t \neq u$, 与 $p_1, \dots, p_n$ 满足 T 的限制矛盾。因此不存在 $t \in \{a+1, \dots, b-1\}$ 使得 p_t 是 v 的祖先, 由于其它点对间的相对顺序不变, 因而 $p_1, \dots, p_a, p_b, p_{a+1}, \dots, p_{b-1}, p_{b+1}, \dots, p_n$ 满足 T 的限制。因此对于任意一个满足 T 的限制的排列 p , 都存在一个满足 T 的限制, 且 x_u, x_v 在排列中相邻的排列 q , 使得 $F(\{x_{q_1}, \dots, x_{q_n}\}) \leq F(\{x_{p_1}, \dots, x_{p_n}\})$ 。因此存在一种最优的排列方式满足 u, v 在排列中相邻。 $\square$

由引理 3.1, 只考虑所有 u, v 相邻的满足 T 的限制的排列时得到的最优解是原问题的一个最优解。考虑在树上将 u, v 合并为一个点, 这个点代表两个元素组成的序列 $\{x_u, x_v\}$ 。

此时问题相当于, 给出一棵 $n-1$ 个点的有根树 T' , 每个点对应一个元素组成的序列 s_i , 选择一个满足 T 的限制的 $n-1$ 阶排列 p , 将所有序列按照排列顺序拼接后得到序列 $S = s_{p_1} + s_{p_2} + \dots + s_{p_n}$, 最小化 $F(S)$ 。不难发现, 在新的问题中, 所有满足 T' 限制的排列拼接出的序列即为所有原问题中满足 T 的限制, 且 x_u, x_v 在排列中相邻的排列所得到的序列。因此, 新问题的最优解对应原问题的一个最优解。

新的问题与之前的问题形式相同, 但每个点不再对应一个元素, 而对应一个元素组成的序列。因而如果希望继续使用上述引理, 则需要将元素间的比较关系拓展为元素构成的序列间的比较关系, 即如下条件:



设所有给出的元素组成的集合为 S ，所有由 S 中元素构成的非空序列组成的集合为 T 。存在一种 T 中元素间的二元比较关系 $\leq$ ，满足如下三条性质：

性质 3.1. $\forall a, b \in T, a \leq b, b \leq a$ 中至少有一者成立。

性质 3.2. $\forall a, b, c \in T$ ，如果 $a \leq b, b \leq c$ ，则 $a \leq c$ 。

性质 3.3. $\forall a, b \in T$ ，如果 $a \leq b$ ，则对于任意由 S 中元素组成的序列 s_1, s_2 ，下列不等式都成立：

$$F(s_1 + a + b + s_2) \leq F(s_1 + b + a + s_2)$$

如果问题满足上述性质，则存在两种可以在 $O(n \log n)$ 次序列比较， $O(n)$ 次序列合并中得到问题的一个最优解的算法。下一部分将会给出对两种算法的分析。

3.3 树上问题的两种算法

接下来的算法都在问题满足上述性质的基础上考虑。此时序列满足和元素相同的比较性质，可以将问题中每个点对应一个元素 x_i 变为每个点对应一个序列 $s_i = \{x_i\}$ ，因此接下来只考虑序列间的比较和合并。

3.3.1 算法一

在 T 满足上述性质的基础上继续考虑上述做法，此时在引理 3.1 中将元素变为序列，引理仍然成立。因此在一个点对应一个元素组成的序列时，也可以进行上文中合并两个点的操作。上述每次操作都会在树上将两个点以及它们对应的序列合并，合并 $n-1$ 次后，有根树变为一个点，由引理这个点对应的序列即为一种最优解。通过上述步骤，可以得到如下算法：

进行 $n-1$ 次操作，每次找到除去根节点对应的序列外，其余点对应序列中最小的序列。设这个点为 x ，将 x 对应的序列拼接到 x 父亲的序列的末尾，并在树上将 x 与 x 的父亲合并为一个点。进行 $n-1$ 次操作后，根节点对应的序列即为一种最优解。

使用引理 3.1 容易证明，每一次合并点后，新得到的问题的最优解对应上一个问题的一个最优解，因此上述算法的正确性得以保证。

在实现时，如果使用树上并查集维护缩点，将并查集中一个连通块的信息记录在连通块的根处，使用任意一种支持插入以及删除最小值的数据结构（例如 C++ 中的 `priority_queue`）维护除去根节点外所有点对应的序列，则上述算法需要进行 $O(n \log n)$ 次序列比较， $O(n)$ 次序列合并，其余部分时间复杂度为 $O(n \log n)$ 。

3.3.2 算法二

通过从不同的角度分析问题,可以得到另外一种解决这类问题的算法。这个算法基于如下两个引理:

Lemma 3.2. 称一个排列 p 满足序列 $\{v_1, \dots, v_k\}$ 的顺序关系, 当且仅当在排列中 v_1 比 v_2 更先出现, v_2 比 v_3 更先出现, $\dots$, v_{k-1} 比 v_k 更先出现。

设有根树 T 中节点 u 有若干个儿子节点 $v_1, \dots, v_k$, 且这些儿子节点都是叶子节点。则如果满足 $s_{v_1} \leq s_{v_2} \leq \dots \leq s_{v_k}$, 则存在一种满足 T 的限制的最优的元素排列, 其满足序列 $\{v_1, \dots, v_k\}$ 的顺序关系。

为了证明这个引理, 这里给出另一个引理:

Lemma 3.3. 设有根树中 u 有两个儿子节点 v_1, v_2 , 如果 $s_{v_1} \leq s_{v_2}$, 对于任意一种满足 T 要求的排列 p , 设 $p_i = v_1, p_j = v_2$, 则如果 $j \leq i$, 则可以通过只改变排列中 $p_j, p_{j+1}, \dots, p_i$ 的值, 得到另外一个排列 q , 满足 $F(s_{q_1} + \dots + s_{q_n}) \leq F(s_{p_1} + \dots + s_{p_n})$ 且在排列 q 中 v_1 比 v_2 先出现。

证明. 因为 $s_{v_1} \leq s_{v_2}$, 由性质3.1,3.2, 可以得到不等式 $s_{p_{j+1}} + \dots + s_{p_{i-1}} \leq s_{v_2}, s_{v_1} \leq s_{p_{j+1}}, \dots, s_{p_{i-1}}$ 中至少有一个成立。

如果前者成立, 由性质 3.3, 可以得到:

$$\begin{aligned} & F(s_{p_1} + \dots + s_{p_{j-1}} + s_{v_2} + s_{p_{j+1}} + \dots + s_{p_{i-1}} + s_{v_1} + s_{p_{i+1}} + \dots + s_{p_n}) \\ & \geq F(s_{p_1} + \dots + s_{p_{j-1}} + s_{p_{j+1}} + \dots + s_{p_{i-1}} + s_{v_2} + s_{v_1} + s_{p_{i+1}} + \dots + s_{p_n}) \\ & \geq F(s_{p_1} + \dots + s_{p_{j-1}} + s_{p_{j+1}} + \dots + s_{p_{i-1}} + s_{v_1} + s_{v_2} + s_{p_{i+1}} + \dots + s_{p_n}) \end{aligned}$$

因为 u, v 为叶子节点且父亲相同, 不难得到 $p_1, \dots, p_{j-1}, p_{j+1}, \dots, p_{i-1}, v_1, v_2, p_{j+1}, \dots, p_n$ 满足 T 的限制, 不难发现这个排列满足引理的要求。后者成立的情况同理, 因此引理成立。 $\square$

由这个引理, 可以给出前一个引理证明:

证明. 对于任意一个由 $\{v_1, \dots, v_k\}$ 中元素组成, 且元素两两不同的序列 $\{v_{i_1}, \dots, v_{i_l}\}$, 定义 $g(\{v_{i_1}, \dots, v_{i_l}\})$ 为所有满足 T 的限制, 且满足序列 $\{v_{i_1}, \dots, v_{i_l}\}$ 的顺序关系的排列 p 中, $F(s_{p_1} + \dots + s_{p_n})$ 的最小值。由于 $v_1, \dots, v_k$ 为父亲相同的叶子节点, 因此不难发现对于任意满足条件的序列, 都存在至少一种合法的排列, 因此 g 是良定义的。

考虑两个满足 $s_{v_x} \leq s_{v_y}$ 的元素 x, y 。对于任意一个满足条件的序列 $\{v_{i_1}, \dots, v_{i_l}\}$, 如果 $\exists t \in \{1, \dots, l-1\}$ 使得 $i_t = y, i_{t+1} = x$, 令 i' 为在序列 $\{i_1, \dots, i_l\}$ 中交换 i_t, i_{t+1} 的结果, 考虑 $g(\{v_{i_1}, \dots, v_{i_l}\})$ 与 $g(\{v_{i'_1}, \dots, v_{i'_l}\})$ 间的大小关系。

对于任意一组在计算 $g(\{v_{i_1}, \dots, v_{i_l}\})$ 时满足条件的排列 p , 不难发现在 p 中, v_y 比 v_x 更先出现, 且设 $p_j = v_y, p_i = v_x$, 则 $v_{i_1}, \dots, v_{i_l}$ 没有在 $p_{j+1}, \dots, p_{i-1}$ 中出现。由引理 3.3, 存在一个在 p 上只改变 $p_j, \dots, p_i$ 的值, 且使得 v_x 比 v_y 更先出现的排列 q , 使得 q 满足 T 的限制

且 $F(s_{q_1} + \dots + s_{q_n}) \leq F(s_{p_1} + \dots + s_{p_n})$ 。此时 q 满足序列 $\{v_{i_1}, \dots, v_{i_{l-1}}, v_x, v_y, v_{i_{l+1}}, \dots, v_{i_l}\}$ 的顺序关系。因此 q 是在计算 $g(\{v_{i_1}, \dots, v_{i_{l-1}}, v_x, v_y, v_{i_{l+1}}, \dots, v_{i_l}\})$ 中一种满足要求的排列。从而可以得到 $g(\{v_{i_1}, \dots, v_{i_{l-1}}, v_x, v_y, v_{i_{l+1}}, \dots, v_{i_l}\}) \leq g(\{v_{i_1}, \dots, v_{i_l}\})$ 。

此时如果将 g 看作上一部分中算法需要最小化的函数, 则当前的 $\leq$ 关系满足性质 2.3, 因为 $\leq$ 满足性质 3.1, 3.2, 所以 $\leq$ 一定满足性质 2.1, 2.2。因此由上一部分中的 Exchange Arguments 方法, 可以得到如果 $s_{v_1} \leq \dots \leq s_{v_k}$, 则 $g(\{v_1, \dots, v_k\})$ 是所有 k 阶排列 p 中 $g(\{v_{p_1}, \dots, v_{p_n}\})$ 的最小值。因此存在一种最优的排列 p 满足序列 $\{v_1, \dots, v_k\}$ 的顺序关系。因此引理成立。 $\square$

另一个引理为:

Lemma 3.4. 设在有根树 T 中, 点 u 的所有儿子节点为 $v_1, \dots, v_k$, 且 u 的所有儿子节点都是叶子节点。假设 $s_{v_1} \leq s_{v_2} \leq \dots \leq s_{v_k}$ 。如果 $s_{v_1} \leq s_u$, 则对于任意一个满足 T 的限制, 且满足序列 $\{v_1, \dots, v_k\}$ 的顺序关系的排列 p , 都存在一个排列 q 满足 T 的限制, 且满足序列 $\{v_1, \dots, v_k\}$ 的顺序关系, 并且 $F(s_{q_1} + \dots + s_{q_n}) \leq F(s_{p_1} + \dots + s_{p_n})$ 。

证明. 设在排列 p 中 $p_j = u, p_i = v_1$, 则 $j \leq i$ 。如果 $s_{p_i} \leq s_{p_j}$, 则不等式 $s_{p_{j+1}} + \dots + s_{p_{i-1}} \leq s_{v_2}, s_{v_1} \leq s_{p_{j+1}}, \dots, s_{p_{i-1}}$ 中至少有一者成立。

如果前者成立, 则由性质 3.3 可以得到:

$$\begin{aligned} & F(s_{p_1} + \dots + s_{p_{j-1}} + s_{p_j} + s_{p_{j+1}} + \dots + s_{p_{i-1}} + s_{p_i} + s_{p_{i+1}} + \dots + s_{p_n}) \\ & \geq F(s_{p_1} + \dots + s_{p_{j-1}} + s_{p_{j+1}} + \dots + s_{p_{i-1}} + s_{p_j} + s_{p_i} + s_{p_{i+1}} + \dots + s_{p_n}) \end{aligned}$$

因为 p 满足序列 $\{v_1, \dots, v_k\}$ 的顺序关系, 因而 $v_2, \dots, v_k$ 在排列中都在 v_1 之后出现。不难得到 $p_1, \dots, p_{j-1}, p_{j+1}, \dots, p_{i-1}, u, v_1, p_{j+1}, \dots, p_n$ 满足 T 的限制, 且满足序列 $\{v_1, \dots, v_k\}$ 的顺序关系。后者成立的情况同理。因此引理成立。 $\square$

由上述引理, 可以推出如下引理:

Lemma 3.5. 对于 T 中的任意点 u , 设 u 子树中所有点构成集合 $subtree_u$, 将 $subtree_u$ 中元素分为若干个序列 $t_1, \dots, t_k$ (每个 $subtree_u$ 中元素在所有序列中正好出现一次), 设 $t_i = \{t_{i,1}, \dots, t_{i,l_i}\}$, 令 $c_i = s_{t_{i,1}} + \dots + s_{t_{i,l_i}}$, 则存在一种方式满足如下条件:

1. $c_1 \leq c_2 \leq \dots \leq c_k$
2. 如果在原问题中删去 u 的子树, 并在树中向 u 的父亲加入 k 个儿子节点 $v_1, \dots, v_k$, 它们依次对应序列 $c_1, \dots, c_k$, 则新问题中所有满足序列 $\{v_1, \dots, v_k\}$ 的顺序关系的排列中的最优解对应原问题的一组最优解。

证明. 在树上进行归纳。对于叶子节点 u , 令 $k = 1, d_1 = \{u\}$, 此时显然成立。

对于非叶子节点 u , 设它有 m 个儿子节点为 $v_1, \dots, v_m$ 。由归纳假设, 它的每个儿子都满足引理。因而对于每个儿子 v_i , 都存在满足引理条件的点的序列 $t_{i,1}, \dots, t_{i,k_i}$ 以及对应元素

的序列 $c_{i,1}, \dots, c_{i,k_i}$ 。对每个子树使用第二条条件后, 书中 u 的子树变为 u 有 $\sum_{j=1}^m k_i$ 个儿子节点 $v'_{1,1}, \dots, v'_{1,k_1}, \dots, v'_{m,1}, \dots, v'_{m,k_m}$, 这些节点都是叶子节点, 且它们对应上文中的那些序列。因为 $c_{i,1} \leq \dots \leq c_{i,k_i}$, 因而使用归并的方式可以将所有儿子节点排成一列, 使得在这个排列中, 对于任意一个儿子节点 v_i , 排列都满足 $\{v'_{i,1}, \dots, v'_{i,k_i}\}$ 的顺序限制, 且排列中所有点对应的序列在比较关系下单调不降: 对于排列中两个相邻的元素 $v'_{i,a}, v'_{j,b}$, 满足 $c_{i,a} \leq c_{j,b}$ 。记 $l = \sum_{j=1}^m k_i$, 当前点的排列为 $p_1, \dots, p_l$, 点对应的序列为 $s_{p_1}, \dots, s_{p_l}$ 。由引理 3.2, 存在当前问题的一组最优解, 使得解的排列满足 $\{p_1, \dots, p_l\}$ 的顺序限制。又因为 $\{p_1, \dots, p_l\}$ 满足所有 $\{v'_{i,1}, \dots, v'_{i,k_i}\}$ 的顺序限制, 从而对每个子树考虑第二条性质, 可以得到满足 $\{p_1, \dots, p_l\}$ 的顺序限制的最优解对应原问题的一组最优解。

考虑这样一组最优解的性质, 如果 $s_{p_1} \leq s_u$, 则由引理 3.4, 存在一种排列满足 $\{p_1, \dots, p_l\}$ 的顺序限制, 满足 u, p_1 在排列中相邻, 且这组解的 F 不大于最优解的 F , 因此这也是一组最优解。考虑在树中将节点 p_1 与 u 合并, 将对应的序列进行拼接 (拼接后的 $s'_u = s_u + s_{p_1}$)。则不难得到新问题中满足 $\{p_2, \dots, p_l\}$ 的顺序限制的解对应原问题中一组满足 $\{p_1, \dots, p_l\}$ 的顺序限制的解, 且原问题中任意一组满足 u, p_1 在排列中相邻的解都对应一组新问题的解。因此由引理 3.2, 新问题中存在一组满足 $\{p_2, \dots, p_l\}$ 的顺序限制的最优解, 从而这组解对应之前问题中所有满足 $\{p_1, \dots, p_l\}$ 的顺序限制的排列的最优解, 从而对应原问题的一组最优解。

不断进行上面的操作, 直到当前排列中的第一个点满足 $s_{p_i} \not\leq s_u$ 或者所有叶子节点都被合并。对于第二种情况, 取 $k = 1, d_1, c_1$ 为之前合并的结果即可满足引理条件。对于第一种情况, 设合并了 $p_1, \dots, p_{i-1}$, 则此时满足 $s_u \leq s_{p_i} \leq \dots \leq s_{p_k}$ 。考虑令 t_1 为当前 u 对应的点序列, $t_2, \dots$ 为 $p_i, \dots, p_k$ 对应的点序列, 则这满足第一条要求。同时因为 $p_i, \dots, p_k$ 都是叶子节点且父亲为 u , 因此在第二条要求中的新问题中, 所有满足序列 $\{u, p_i, \dots, p_k\}$ 的顺序关系的排列都可以对应此时一种满足序列 $\{p_i, \dots, p_k\}$ 的顺序关系的最优解, 由上述分析可得这个最优解对应一组原问题的最优解, 因此新问题满足这个条件的最优解对应一组原问题的最优解, 引理成立。□

使用上述引理证明中的构造方法, 即可得到第二种算法:

考虑对于每个点 u 求出引理 3.5 中该点的所有 c_i 按顺序排列构成的序列 S_u 。对于一个 u , 进行如下过程:

首先对于 u 的每个儿子 v_i 求出 S_{v_i} , 接着将所有的 S_{v_i} 按照序列的比较关系归并得到 S_u 。

令 $C = \{x_u\}$, 重复进行下一步:

找到 S_u 中的第一个元素, 设为 c 。如果 S_u 为空或者 $c \not\leq C$ 则结束过程。否则, 从 S_u 中删去 c , 并将序列 c 拼接在 C 后得到新的 C 。

在重复的过程结束后, 将 C 加入 S_u 的开头, 结束当前点的构造过程。

可以在树中进行深度优先搜索, 自下向上求出所有的 S_u , 求出根节点的 S 后, 由引理 3.5 可知, 将 S 中所有序列拼接即可得到问题的一组最优解。该算法需要 $O(n)$ 次对两个 S

进行归并, $O(n)$ 次询问/删除 S 的第一个序列以及 $O(n)$ 次合并序列。如果使用平衡树或类似的数据结构维护 S , 则该算法需要 $O(n \log n)$ 次比较序列, 与上一个算法的复杂度相同。

3.4 对具体模型的分析

对于满足存在一种比较关系满足性质 3.1, 3.2, 3.3 的问题, 上述两种算法都可以使用 $O(n \log n)$ 次序列比较以及 $O(n)$ 次序列合并中解决问题。接下来考虑哪些模型满足这一条件。这里对上一部分中提到的三种模型进行分析:

3.4.1 第一种模型

在第一种模型中, 元素形如 $(a, b) (a, b \in \mathbb{R}, a \geq 0)$, $F(\{(a_1, b_1), \dots, (a_n, b_n)\}) = \sum_{1 \leq i < j \leq n} a_i b_j$ 。考虑对于两个序列 $c_1 = \{(a_1, b_1), \dots, (a_n, b_n)\}$, $c_2 = \{(a'_1, b'_1), \dots, (a'_m, b'_m)\}$, $F(c_1 + c_2)$ 与 $F(c_2 + c_1)$ 的关系 (+ 表示序列拼接)。则不难得到如下结果:

$$F(c_1 + c_2) - F(c_2 + c_1) = \left(\sum_{i=1}^n a_i \right) * \left(\sum_{i=1}^m b'_i \right) - \left(\sum_{i=1}^m a'_i \right) * \left(\sum_{i=1}^n b_i \right)$$

因此可以将序列看成 $(\sum a_i, \sum b_i)$, 这样得到的结果仍然是一个元素。因此考虑使用上一部分中这种模型的比较方式对两个序列的 $(\sum a_i, \sum b_i)$ 进行比较, 比较的结果即为序列比较的结果。不难发现这样比较满足性质 3.1, 3.2, 3.3。

3.4.2 第二种模型

在第二种模型中元素形如 $(a_i, b_i) (a, b \in \mathbb{R})$, $F(\{(a_1, b_1), \dots, (a_n, b_n)\}) = -\min_{i=1}^n \{(\sum_{j=1}^{i-1} b_j) - (\sum_{j=1}^i a_j)\}$ 。此时从 F 进行分析可以得到如下结论:

对于序列中两个位置连续的元素 $(a_1, b_1), (a_2, b_2)$, 在序列中将它们合并为一个元素 $(-\min(-a_1, b_1 - a_1 - a_2), b_1 + b_2 - a_1 - a_2 - \min(-a_1, b_1 - a_1 - a_2))$ 不会改变 F 的值。

归纳可得, 在计算 F 时任意一个元素组成的序列都可以被合并为一个等价的元素。因此在性质 3.3 中可以在不等式两侧分别将序列 s, t 合并为与之等价的元素, 从而可以使用等价元素的比较结果来比较两个序列。由于元素间存在一种比较关系满足性质 2.1, 2.2, 2.3, 因此使用这种比较关系得到的序列间比较关系满足性质 3.1, 3.2, 3.3。

3.4.3 第三种模型

对于第三种模型, 序列中若干个字符串进行拼接后仍然是字符串, 因此上述性质显然成立, 但对字符串进行拼接及比较的问题则较为复杂。这部分的具体细节在本文中不再展开。



但对于这部分开头给出的反例,不存在一种 $\leq$ 关系满足上述条件。例如,令 $k=2$,则 $F(\{3,3\}) > F(\{4,1\})$,但 $F(\{4,3,2\}) < F(\{4,4,1\})$,由性质 3.1, $\{3,3\}, \{4,1\}$ 中必定有一者小于等于另外一者,但两种情况都与性质 3.3 矛盾。因此上述条件一定不能被满足,该问题不能使用上述算法解决。

3.5 算法性质及应用

对于上述前两种模型,它们有较好的性质:在合并两个序列 c_1, c_2 时,只需要知道关于两个序列的 $O(1)$ 级别的信息,即可求出合并后序列的信息以及 $F(c_1 + c_2)$ 。例如,对于第一种模型,只需要知道序列 c 的 $\sum a_i, \sum b_i, F(c)$,即可 $O(1)$ 求出合并后的这些信息,而在第二种模型中只需要知道等价元素 (a, b) 。因为这些原因,对于这两种模型对应的问题,上述两种算法可以在 $O(n \log n)$ 的时间复杂度内得到答案。其中第一种模型对应上文给出的例题 6,而下面这个题目对应第二种模型:

例题 7.⁸

有一棵 n 个点的树,树上除去节点 1 外的每个节点有一个权值 v_i 。有一个人初始在点 1,这个人有一个值 hp ,初始为 0。这个人可以沿着树边移动,他第一次到达一个点 i 时, hp 会变为 $hp + v_i$,如果 $hp < 0$ 则他会失败。求他能否在不失败的情况下到达一个给定的点 t 。
 $n \leq 10^5, |v_i| \leq 10^9$

考虑加入一个叶子节点连向 t ,这个节点的权值为 $+\infty$,则如果能到达 t ,则从 t 走到这个叶子节点后,一定可以经过其它所有节点。因此问题变为能否在不失败的情况下经过所有节点。如果维护一个序列,在第一次经过某个节点时将节点加入这个序列,则得到的序列是一个满足给出的树 T 的限制的排列,且经过所有节点时 v 的变化过程即为按照排列顺序经过每个点时, v 的变化过程。

此时问题变为找到一个满足 T 的限制的排列,使得按照排列顺序经过每个点,任意时刻 $v \geq 0$ 。考虑最大化 v 在所有时刻的最小值,则问题变为第二种模型的形式,其中一个大于等于 0 的元素 x 对应 $(0, x)$,小于 0 的元素 x 对应 $(-x, 0)$ 。使用上述算法求解,时间复杂度为 $O(n \log n)$ 。

3.5.1 求一组最优解

上一部分提到,在前两种模型中,只需要维护序列的部分信息就可以比较序列并维护答案,考虑在这种情况下求出方案。上述两种算法中,都进行了 n 次合并序列操作,每次操作形如将序列 s_i 拼接到序列 s_j 末尾。在算法的过程中记录这些操作,执行完整个算法后,通过这些合并操作的信息,不难在 $O(n)$ 的时间复杂度内还原出算法求出的方案。

⁸题目来源:2013-2014 ACM-ICPC, Central Europe Regional Contest E Escape

3.5.2 求“每个点子树问题的答案”

考虑下面这类形式的问题：

给出一个上述形式的问题，称一个以 u 为根的子树的子问题为：只保留树中 u 子树内的点作为新的树 T' ，以及这些点对应的元素，以 u 为 T' 的根，其余问题形式与之前的问题相同。

对于树中每个点 u ，求出以 u 为根的子树的子问题的答案。

下面是一个这种问题的例子：

例题 8.⁹

给定一棵 n 个点的有根树，1 为根，点有点权 v_i 。你有若干枚棋子，可以进行如下操作：

1. 选择一个上面有棋子的点，将这个点上的所有棋子拿回。
2. 选择一个上面没有棋子，且它的儿子上都有棋子的点 i ，向这个点上放 v_i 个棋子。

对于每个点，求出如果需要在该点上放上棋子，则初始时最小需要多少枚棋子。

$$n \leq 2 \times 10^5, 1 \leq v_i \leq 10^9$$

分析题目不难得到如下性质：

1. 如果需要在 i 上放上棋子，则过程中只会向 i 的子树中放棋子，且正好会向 i 子树中的每个点放一次棋子。
2. 存在一种最优的操作方式，使得对于任意点 x ，在向 x 上放上棋子后，下一步操作为拿回 x 儿子上的棋子。

这里略去证明。通过上述性质，设 $p_1, \dots, p_k$ 为操作中所有向点上放棋子的操作的点按顺序构成的序列，则对于向 i 上放棋子的操作，存在一种最优解使得 p 是 i 子树中所有点的一个排列。该排列需要满足，对于树上任意一个点 u ，设 u 的父亲为 v ，如果 u, v 都在排列中出现，则在排列中 u 一定在 v 之前出现。同时，设 s_i 为点 i 所有儿子的 v 之和，则由第二条性质，在点 p_i 放下棋子后，此时总共放下的棋子数量为 $(\sum_{j=1}^i v_{p_j}) - (\sum_{j=1}^{i-1} s_{p_j})$ ，因此相当于最小化 $\max_i ((\sum_{j=1}^i v_{p_j}) - (\sum_{j=1}^{i-1} s_{p_j}))$ 。这与上述第二种模型类似，唯一的区别是树上一条边限制的方向与之前相反。

注意到 $(\sum_{j=1}^i v_{p_j}) - (\sum_{j=1}^{i-1} s_{p_j}) = (\sum_{i=1}^k v_{p_i} - s_{p_i}) + (\sum_{i=1}^{k-i+1} s_{p_{k-i}}) - (\sum_{i=1}^{k-i} v_{p_{k-i}})$ ，因此将排列翻转后的 F 仍然与之前形式相同，而翻转后树的限制变为 u 的父亲 v 在排列中一定在 u 之前出现，这与上文中满足 T 的限制等价。因此通过上述方式，问题被转化为对于树中每个点 u ，求出以 u 为根的子树的子问题的答案。

⁹题目来源：UOJ418 【集训队作业 2018】三角形

使用算法二中的思路容易解决这种问题。在算法二中, 由引理 3.5 不难得到如下性质: 记 S_u 为引理 3.5 中 u 构造的所有序列构成的序列, 将 S_u 内元素按顺序拼接, 得到的序列即为一种以 u 为根的子树的子问题的最优解。

因此, 如果在算法二中使用平衡树或者类似的数据结构维护 S_u , 即可同时在平衡树中维护以 u 为根的子树的子问题的最优解。这样的过程复杂度与算法二相同。

使用算法一的思路也可以解决这个问题, 但需要进行更多的分析, 可以得到如下引理:

Lemma 3.6. 对于有根树 T 中的任意点 u , 设 u 的儿子节点构成集合 son_u , 则 $\forall S \in son_u$, 令 $U(u, S)$ 为 u 以及 S 中所有点在树中的子树内的所有节点构成的集合, 则:

对于任意一种在 T 中使用算法一求出的最优排列 p , 存在一种在只保留树中 $U(u, S)$ 的导出子图, 且以 u 为根的树上使用算法一求出的排列 q , 使得 p 满足 $\{q_1, \dots\}$ 的顺序关系。

证明. 对 T 的节点数进行归纳, 只有一个节点时显然成立。

对于其余情况, 考虑在 T 中进行算法一过程中的第一次合并操作。如果这次合并操作没有涉及到属于 $U(u, S)$ 的点, 则操作后 $U(u, S)$ 不变, 这部分中使用算法一求出的最优排列不变, 由归纳假设可知此时结论成立。如果这次合并只影响到了 u (例如 u 与父亲节点合并或者某个不在 $U(u, S)$ 内的儿子节点与 u 合并), 则合并后 $U(u, S)$ 在树中仍然有引理中要求的结构, 且在合并后 $U(u, S)$ 的问题中, 树结构不变, 但根节点对应的序列会改变。但可以发现算法一中根节点对应的序列不影响算法的过程, 因此此时结论仍然成立。

否则, 合并操作一定合并了两个属于 $U(u, S)$ 的节点。设合并点为 u 与 u 的父亲节点 fa_u , 则 s_u 为 T 中除去根节点外所有节点对应序列的一个最小值, 因此它也是 $U(u, S)$ 除去根节点 u 外所有节点对应序列的一个最小值。因此存在在保留 $U(u, S)$ 的导出子图中以 u 为根得到的有根树中进行算法一的一种方式, 使得第一步合并 u 与 fa_u 。由归纳假设可知此时结论成立。因此引理成立。 $\square$

由引理 3.6, 对于任意一种算法一求出的最优排列 p , 对于任意树中节点 x , 在 p 中只保留属于 x 子树内的节点得到的排列即为对 x 子树内的问题使用算法一得到的一种最优解。因此通过这个最优排列, 可以知道在所有子树内的问题中元素一定满足一种顺序。此时可以使用支持合并的数据结构, 在树中进行深度优先搜索, 自下向上维护每个节点的一种最优解。一种实现方式为使用动态开点线段树进行维护, 合并时使用线段树合并, 该算法的复杂度也与算法一相同。

由引理 3.5 不难发现, 算法二求出的最优排列 p 也满足这一性质: 对于任意树中节点 x , 在 p 中只保留属于 x 子树内的节点得到的排列即为对 x 子树内的问题的一种最优解。但如果最优排列 p 不是由上述算法求出, 而是任意一个最优的排列, 则容易构造出上述性质的反例。因此上述结论只对本文中给出的两种算法求出的最优解有保证。

3.5.3 部分其它应用

最后, 这里再给出一个与算法二的做法有关的题目:

例题 9. ¹⁰

有 n 堆石子排成一列, 第 i 堆石子有 p_i 个。对于每个 i , 求出如下问题的答案:

你需要将所有石子合并成一堆, 每次你可以选择相邻的两堆石子, 将它们合并成一堆, 代价为两堆石子数量的和。有一个额外限制: 每次选择的两堆石子中, 必须包含原来的第 i 堆石子。求出最小总代价。

$$n \leq 2 \times 10^5, 1 \leq a_i \leq 10^6$$

考虑从位置 x 开始的问题, 可以看成每次将一堆石子并入第 x 堆, 设第 $i-1$ 次加入了第 p_i 堆石子 ($p_1 = x$), 则 p 构成一个排列, 序列的限制相当于 $\forall i \in \{1, 2, \dots, k-1\}$, $i+1$ 在排列中比 i 先出现, 且 $\forall i \in \{k+1, \dots, n\}$, $i-1$ 在排列中比 i 先出现。因此考虑一条 $1-2-\dots-n$ 的链, 设这条链以 k 为根得到的有根树为 T_k , 则从位置 x 开始的问题中, 合并顺序的限制等价于 p 满足 T_k 的限制。

容易发现总代价只和 p 有关, 且代价为 $\sum_{i=1}^n a_{p_i} * (n-i+1)$ 。如果减去一个定值 $\sum_{i=1}^n a_i$, 则上述代价相当于 $\sum_{1 \leq i < j \leq n} a_{p_i}$ 。因此如果令点 i 对应元素 $(a_i, 1)$, 上述代价即为第一种模型中的形式。从而可以使用上述算法, 在 $O(n \log n)$ 的时间复杂度内求出单个问题的答案。

如果使用算法二, 则该算法求出以 x 为根的答案的步骤为:

求出在引理 3.5 中, $1-2-\dots-(x-1)$ 的链, 以 $x-1$ 为根时根节点得到的所有序列按顺序组成的序列 L_{x-1} , 以及 $(x+1)-\dots-n$ 的链, 以 $x+1$ 为根时根节点得到的序列 R_{x+1} 。由引理 3.2, 3.4 可以得到, 设将 L_{x-1} , R_{x+1} 按照比较关系归并后得到的序列为 S'_x , 则将 S'_x 中所有序列按顺序拼接, 再在得到的序列开头加入 x 后, 得到的方案对应原问题的一组解, 且是由引理 3.5 转换后问题的一组最优解, 因此这组解是原问题的一组最优解。

考虑 L_x 与 L_{x+1} 间的差别, 根据上述算法, 求 L_{x+1} 时, $x+1$ 作为根, 只有一个儿子 x , 且对 x 的子树进行算法二可以得到 L_x 。因此算法二中会进行如下操作: 令 $C = \{(v_x, 1)\}$, $L_{x+1} = L_x$, 取出 L_{x+1} 中第一个元素 c , 如果 $c \leq C$ 则将 c 从 L_{x+1} 中删去, 并将 c 拼接到 C 末尾。重复执行上一步直到不满足条件, 最后将 C 加入 L_{x+1} 开头。

在 x 从 1 依次增加到 n 时, 对 L 进行的操作为向开头插入或删除序列并查询开头的序列。显然插入及删除的次数为 $O(n)$ 次。对 R 也只会进行 $O(n)$ 次向开头插入或删除的操作。只需要在每次插入或删除后维护当前 L, R 按比较关系归并的一种结果, 就可以维护答案。考虑使用平衡树维护一种归并的结果, 删除开头序列时可以直接在平衡树中删除。在开头插入序列时, 设操作为插入到 L 开头, 插入序列为 s , 则由构造方式, 插入的序列一定小于等于当前 L 中的所有序列。从而如果将这个序列插入到平衡树中最后一个满足 $t \leq s, s \not\leq t$ 的元素 t 后, 则这种方式既保证了有序性, 也保证了 L 中所有元素在当前序列中按照顺序排列, 因此插入后仍然是一种归并的结果。在平衡树中维护答案, 因为序列比较, 拼接, 合并答案可以 $O(1)$ 维护, 因而时间复杂度为 $O(n \log n)$ 。

¹⁰ 题目来源: Yuhao Du Contest 7 H Heavy Stones

本题中只要求出答案,而如果两个序列 a, b 满足 $a \leq b, b \leq a$, 则在方案中交换相邻的这两个序列一定不改变 F , 因此在插入时, 对于满足 $a \leq b, b \leq a$ 的序列 a, b 按照任意顺序排列都不会改变答案。但如果按照任意顺序插入, 虽然答案一定与最优解答案相同, 但当前维护的方案不一定满足树的限制。

3.6 总结

这类问题有着与上一部分中问题类似的结构, 但在上一种问题上额外增加了对排列的限制, 对排列的限制可以看作一个有根树的形式。从需要满足的条件以及满足条件下的解法上看, 两种问题都有很强的关联。但由于加入了限制, 通过一些例子可以发现只满足之前的要求并不能解决问题。本文对上一部分中的要求加强, 将比较元素变为了比较序列。在此基础上, 本文给出了两种从不同角度出发的算法。第一种算法从局部单个点的情况入手, 通过将比较关系下最小的序列所在的点与它的父亲合并得到规模更小的问题。第二种算法则是先解决每个儿子子树内的问题, 通过若干性质, 将儿子子树内问题的解合并得到自身子树内问题的解。两种算法可以以相同的时间复杂度解决问题。从两个算法的解题过程出发, 可以得到更多的应用, 本文对一些应用进行了分析, 但因为笔者水平有限, 本文只举出了笔者所知的部分应用。

4 后记及展望

在信息学竞赛中, 贪心思想有着广泛的应用, 有很多与贪心思想有关的值得深入研究的问题。本文所述的相关算法在信息学竞赛中已经有了多次出现, 但这类问题的详细分析并不简单。由于笔者水平有限, 还有许多与这类算法相关的问题笔者没有得到很好的答案, 例如下列问题:

1. 本文中两部分分别对比较关系提出了三条性质的限制, 是否存在比这三条性质更弱/更简单, 但也能使得本文所述算法得到最优解的性质?
2. 除了上文提到的三种模型及其变形, 是否存在更多满足上述性质的模型? 模型是否存在更加本质的描述?

希望本文能够起到抛砖引玉的作用, 希望感兴趣的读者能够进行进一步研究, 从而得到这类问题更多更有趣的做法与应用。笔者也希望看到上述问题更好的结果。

感谢

感谢中国计算机学会提供学习和交流的平台。



感谢国家集训队教练杨耀良的指导。

感谢父母对我的培养和教育。

感谢学校对我的栽培。

感谢李翔教练对我多方面的指导，感谢成都外国语学校的教练对我的教导。

感谢戴江齐同学与我交流讨论，给我启发。

感谢刘一平、金天、宋佳兴等多位同学为本文审稿。

感谢温铠瑞学长在我学习信息学竞赛的过程中给我的鼓励和帮助。

参考文献

- [1] Wikipedia, "Weak Orderings", https://en.wikipedia.org/wiki/Weak_ordering
- [2] Errichto, "Lecture #3 — Exchange arguments (sorting with dp)", <https://codeforces.com/blog/entry/63533>
- [3] AtCoder Grand Contest 023 Editorial, <https://atcoder.jp/contests/agc023/editorial>
- [4] CERC 2013: Presentation of solutions, <https://cerc.tcs.uj.edu.pl/2013/data/cerc2013solutions.pdf>



浅谈平面图相关算法

山东省平邑第一中学 唐绍轩

摘要

平面图是图论中一类特殊且重要的图。本文主要介绍了平面图的有关算法。首先讨论了对偶图的求法，接着给出了两种平面图的判定算法，之后从点着色以及完美匹配计数两个方面展示了平面图的特殊性质。在介绍算法的同时穿插了相关例题，给出了这些算法在具体题目上的应用。

1 引入

平面图在信息学竞赛中引入较早，也在各大比赛中出现过不少次。但平面图的相关知识在选手中的普及度并不高，大多数选手对平面图的印象仅停留在对偶图。希望本文能将更多的平面图相关知识带入大家的视野。

本文第二节介绍了平面图的定义和基础性质；第三节介绍了对偶图的性质和求法；第四节介绍了 DMP 算法和 Tarjan 算法这两种平面图的判定算法；第五节介绍了平面图点着色，主要给出了三角剖分图点着色的性质；第六节介绍了求解平面图完美匹配计数问题的 FKT 算法。

2 基础知识

2.1 定义

定义 2.1 (平面图). 若图 G 能被画在平面上且不同的边仅在端点处相交，则称图 G 为平面图。画出的没有边相交的图称为 G 的平面表示或平面嵌入。

定义 2.2 (面). 一个图的平面嵌入会将整个平面划分成若干个互不连通的区域，每个区域称为一个面。无界的区域称作外部面，有界的区域称作内部面。包围某个面的所有边构成了该面的边界。

注意，实际上外部面和内部面并无本质区别。我们总可以通过球极投影将平面图画在球面上，而此时每个面的地位相等。

通过以上讨论,我们反过来可以知道正多面体的点和棱构成了平面图,而正多面体上的面就是图上的面。

定义 2.3 (对偶图). 设 G 为平面图的一个平面嵌入, 定义它的对偶图 G^* 为:

1. 在 G 中的每一个面取一个点, 作为 G^* 的顶点;
2. 对于 G 中的每一条边 e , 都对应一条 G^* 中的边 e^* 连接与 e 相邻的两个面中的顶点, 且 e^* 在平面中穿过 e 。

注意, 无论平面嵌入 G 是不是简单图, 它的对偶图 G^* 都可能有自环和重边。具体来说, 若 G 中有割边 e , 则 G^* 中会有自环 e^* , 若 G 中的某两个面的公共边界有多条边, 则 G^* 中会有这两个面对应点之间的重边。

容易发现, 对于一个连通的平面嵌入 G , 它的对偶的对偶都是其本身, 也即 $(G^*)^* = G$ 。

2.2 欧拉公式

定理 2.1 (欧拉公式). 对于一个连通的平面嵌入 G , 它的点数 V , 边数 E , 面数 F 有如下关系:

$$V - E + F = 2$$

证明. 任取连通图 G 的生成树 T , 依次删除每条非树边, 每次都会令边数 E 与面数 F 恰好减少 1, 故删除过程中 $V - E + F$ 保持不变。删除之后留下的生成树 T 显然满足 $V = E + 1, F = 1$, 故自然有 $V - E + F = 2$ 。

□

注意到, 若平面图 G 由两个不同的连通块 G_1, G_2 构成, 那么有 $V = V_1 + V_2, E = E_1 + E_2, F = F_1 + F_2 - 1$ 。此时我们可以得到 $V - E + F = 3$ 。

更一般地, 我们有 $V - E + F = C + 1$, 其中 C 为连通块数。

引理 2.1. 对于一个简单的连通的平面嵌入 G , 若 $V \geq 3$ 则有 $E \leq 3V - 6$ 。

证明. 若 $E \leq 2$ 条件显然成立, 否则由于图 G 为简单图, 所以 G 中每个面的边界上至少有三条边, 而每条边只与两个面相邻, 故 $3F \leq 2E$, 将欧拉公式带入即得。

□



3 对偶图

可以发现, 一个平面嵌入 G 的对偶图 G^* 实际上给出了 G 的很多有用信息。其中较为常见的是以下结果:

定义 3.1. 将无向图 $G(V, E)$ 的点集 V 划分成两个非空集合 V_1, V_2 , 则其中一个端点在 V_1 , 另一个端点在 V_2 的所有边构成了图 G 的一个割集。若一个割集的任意真子集都不是割集, 则称为极小割。

引理 3.1. 连通平面嵌入 G 的每个极小割与对偶图 G^* 的每个简单环一一对应。特别地, G 的最小割对应了 G^* 中的最小环。

如果两个点 s, t 在平面嵌入 G 的一个面 f 的边界上, 那么将 s, t 之间连一条边 e , 把 f 分割成面 f_1 与面 f_2 , 再在对偶图上跑含有边 e^* 的最小环, 也即去掉边 e^* 后 f_1 到 f_2 的最短路, 就可以算出 s 和 t 之间的最小割。这相比使用最大流求出最小割在时间复杂度上有较大优越性。

例题 1 (网格图删边连通性¹)。给定一个 $n \times m$ 的网格图 G , 进行 q 次操作, 操作有以下两种:

1. 删除两个点 u, v 之间的边, 保证在本次操作之前 u, v 之间有边。
2. 询问两个点 u, v 之间是否连通。

$1 \leq n \times m, q \leq 10^6$, 操作强制在线。

解法

若直接使用无向图动态连通性的算法, 时间复杂度为 $O((nm + q) \log^2 nm)$, 无法通过此题。

考虑若某个时刻的删边改变了连通性, 那么必定有某个点集 S , 由 S 与 $V - S$ 导出的割集中的边全部被删除。由引理 3.1 可知此时对偶图中新形成了至少一个由删除的边构成的环。

网格图的对偶图容易求出, 那么使用并查集维护对偶图的连通性, 对偶图中两个点之间有边 e^* 当且仅当原图中的边 e 被删除了。

那么每次删除边 $e: (u, v)$ 时都在对偶图上加入对应的边 e^* , 若形成了环就说明原图中删去这条边分成了两个连通块。

从 u, v 同时开始 BFS, 每次两边分别扩展一个点, 直到遍历完某个连通块为止。假设分成的两个连通块为 V_1, V_2 , 那么该步的复杂度为 $O(\min(V_1, V_2))$, 容易得到此处的总复杂度为 $O(nm \log nm)$ 。

¹bzoj4423, 题面有改动



将遍历完的那个连通块染成一种新的颜色,那么查询时直接比较两个点的颜色是否相同即可。

时间复杂度 $O(q + nm \log nm)$ 。

此题存在 $O(q + nm)$ 做法,详见参考文献 [2]。

■

3.1 对偶图的求法

给定一张 n 个点的连通平面嵌入 G , 具体来说, 我们给定 G 中每个点在平面上的位置, 边即为连接某两个点的线段。我们需要求出它的对偶图 G^* 。

将每条无向边 (u, v) 拆成两条有向边 $u \rightarrow v$ 和 $v \rightarrow u$, 并对每个点的出边极角排序。

从某条边 $e: u \rightarrow v$ 出发, 每次找出以 v 为中心将边 e 顺时针旋转后碰到的第一条边 e' , 再从 e' 继续往下找, 直到形成了一个环。可以发现, 这个环就对应了 G 的一个面 f 。对于内部面, 环上的边均为逆时针方向, 而对于外部面, 环上的边均为顺时针方向。

找出所有面后, 对于图 G 中的边 $e: (u, v)$, 将 e 两侧的面在对偶图上对应的点之间连边即可。

时间复杂度 $O(n \log n)$, 瓶颈在于给边极角排序。

3.2 点定位

给定一张 n 个点的连通平面嵌入 G , 并且现在有 m 个平面上的点, 需要快速求出这些点在 G 上的哪个面。

如果暴力对于每个面都跑一遍判定点在多边形中的算法, 时间复杂度为 $O(nm)$ 。

首先跑一遍求对偶图把每条边两侧的面找出来。接着对横坐标扫描线, 在每个时刻 t 维护 $x = t$ 这条直线穿过的边的相对位置, 那么操作有插入某条边, 删除某条边, 以及查询当前某个点所在的面。这可以使用平衡树维护, 每次查询时只需找出在这个点上下最近的两条边, 就可以确定该点所在的面。

时间复杂度 $O((n + m) \log n)$ 。

具体实现时可以用 set 来维护边的相对顺序, 只需在写比较函数时比较当前时刻两条边的大小即可。由于两条边不会在除了端点以外的地方交叉, 那么边的相对顺序不变, 所以不会出现错误。

例题 2 (WC2013 平面图²)。给定一张 n 个点的连通平面嵌入 G , 边带权。有 q 次查询, 每次给定两个平面上的点 x, y , 请求出从 x 走到 y 穿过的边的边权最大值最小是多少, 走的过程中不能经过外部面。

$1 \leq n, q \leq 10^5$, 坐标范围 $[0, 10^7]$ 。

²<https://uoj.ac/problem/57>



解法

首先利用上面介绍的算法求出 G 的对偶图，并将询问的点所在的面找出来。

题目中要求不能走外部面，考虑如何判定一个面 f 是否为外部面。一个简单的方法是按照围住 f 的有向边利用叉积算 f 的面积。因为只有外部面中有向边为顺时针方向，那么 f 的面积算出来是负数时就表明它是外部面。

删掉外部面在对偶图 G^* 上的点后，剩下的问题变成了在图 G^* 上，多次询问两个点之间最大边权最小的路径。这是一个经典问题，找出最小生成树后倍增处理两点间的最大边权即可。

时间复杂度 $O((n+q)\log n)$ 。

■

4 平面图判定

现在我们来考虑怎样判定一个给定的图是不是平面图。我们有以下熟知的定理。

定理 4.1 (库拉托夫斯基定理). 一个简单图是平面图当且仅当它并不包含一个是 K_5 或 $K_{3,3}$ 的分割的子图。

其中一个图的分割指将这个图进行若干次以下操作：

- 选定某条边 (u, v) ，新建点 x ，将边 (u, v) 替换为边 (u, x) 与边 (x, v) 。

简单来说，就是在这个图的边上插入若干个二度顶点。

如果直接按照该定理进行判定，所需的时间复杂度甚至是指数级的，这显然不可接受。

所以我们需要更高效的算法解决这一问题。

4.1 简化

为了之后的叙述方便，我们在此对问题进行一些简化。

设要判定的图为 $G(V, E)$ 。注意到自环和重边对图的可平面性没有影响，所以不妨设 G 为简单图。

现在说明只需考虑 G 中的每个点双连通分量。首先 G 的每个连通块之间互不影响。对于其中某个连通分量，若它有割点 v ，则可以将 G 从割点 v 分割成若干个子图 $G_1, G_2, \dots, G_k$ ，注意点 v 会出现在每个子图 G_i 中。若这些图都是可平面的，则可将这些图中的点 v 都放到外部面，再把点 v 拼接起来即可得到 G 的一个平面嵌入。



4.2 DMP 算法

首先, 一个粗略的想法是我们逐步确定点和边在平面上的位置。设目前确定了子图 $H(V', E')$ 在平面上的位置, 那么 G 会被 H 划分成若干还未确定的小块, 具体如下。

定义 4.1 (段). 定义图 G 被子图 H 划分的段为:

- 若边 $e = (u, v)$ 满足 $u \in H, v \in H, e \notin H$, 则点 u, v 与边 e 构成了一个段。
- 考虑 G 删去 H 中的点后构成的连通分量 $G_1, G_2, \dots, G_k$, 对于某个 G_i , G_i 连到 H 的边及其端点与 G_i 本身构成了一个段。

定义段的附着点为同时在该段和子图 H 中的点, 段的边界为附着点组成的点集, 若段中某条简单路径包含的附着点恰为它的起点和终点, 则称为附着路径。

由于每个段都是 G 的连通子图, 所以每个段一定位于 H 的某个面内部, 否则会导致交叉。这也就是说, 每个段的边界一定位于 H 的面上, 记完全包含段 A 的边界的面的集合为 $F(A)$ 。算法如下:

1. 开始时将 H 置为 G 中任意一个简单环, 求出 G 的所有段。
2. 若存在某个段 A 满足 $|F(A)| = 0$, 那么 G 不可平面, 结束算法。
3. 若存在某个段 A 满足 $|F(A)| = 1$, 那么这个段所在的面 f 唯一确定, 可以将 A 嵌入面 f 中。
4. 否则, 任选一个段 A 以及它能嵌入的面 f 。
5. 我们任意选定 A 中的一条附着路径 P 嵌入面 f , 将它分割成两个面, 接着更新 H 以及 G 被 H 划分出的段。
6. 若 G 中没有任何段, 那么 G 可平面, 结束算法, 否则回到 2. 继续执行。

若直接实现这个算法, 时间复杂度会达到 $O(n^3)$ 。

考虑优化一些实现上的细节。我们维护出每个段 A 对应的集合 $F(A)$, 那么每次找出一个段 A 和面 f 的时间复杂度为 $O(n)$, 接着会删除面 f , 插入分割后的两个面 f_1, f_2 , 删除段 A , 插入若干个新段。

那么只需考虑以下两个问题:

- 对于一个特定的面 f , 对所有段 A 求出是否有 $f \in F(A)$ 。
- 对于一个特定的段 A , 对所有面 f 求出是否有 $f \in F(A)$ 。

注意到面上的总点数和段的总附着点数都为 $O(n)$, 那么以上两个问题也可以在 $O(n)$ 的时间内解决。

总共循环的次数为 $O(n)$, 所以总时间复杂度为 $O(n^2)$ 。

下面证明这个算法的正确性。只需证 G 为平面图时, 算法能给出某个平面嵌入即可。

定义 4.2. 定义两个段 S, T 为冲突段当且仅当满足以下条件:

- $F(S) \cap F(T) \neq \emptyset$
- 对于任意 $f \in F(S) \cap F(T)$, 存在两个附着路径 $P \subseteq S, Q \subseteq T$, 满足 P 和 Q 不能同时平面嵌入到 f 中。

引理 4.1. 对于冲突段 S, T , 满足 $|F(S)| \geq 2, |F(T)| \geq 2$, 则有 $F(S) = F(T)$ 且 $|F(S)| = 2$ 。

证明. 反证法, 假设 $F(S) \neq F(T)$, 则可以取出三个互不相同的面 f, g, h , 满足 $f \in F(S), g \in F(T), h \in F(S) \cap F(T)$ 。由于 S, T 是冲突段, 所以可以找到两条附着路径 $P \subseteq S, Q \subseteq T$, 满足它们在面 h 中无法同时嵌入。

由于 $f \in F(S)$, 所以 P 可以在面 f 中嵌入, 同理 Q 可以在面 g 中嵌入。此时两条路径同时嵌入在了 h 的外面, 而这实际上等价于它们能同时嵌入到 h 里面, 等价性可以直接由以 h 内任意一点做圆反演得到。这与冲突段定义矛盾。

所以 $F(S) = F(T)$, 当 $|F(S)| > 2$ 时, 也可以选出三个互不相同的面 f, g, h 做以上讨论, 故 $|F(S)| = 2$ 。 □

接着定义图 C , C 的点集为图 G 的所有段构成的集合, C 中两个点有边当且仅当这两个段是冲突段。

引理 4.2. 当所有段 A 均满足 $|F(A)| \geq 2$ 时, 图 C 是二分图。

证明. 反证法, 假设存在奇环, 不妨设为 $A_1, A_2, \dots, A_k$, 对于任意 $1 \leq i \leq k$, A_i 到 $A_{i \bmod k+1}$ 之间有边。

由引理 4.1, 可知 $F(A_i)$ 均相同且大小为 2。不妨设 $F(A_i) = \{f, g\}$, 由于 A_i 和 $A_{i \bmod k+1}$ 冲突, 它们不能嵌入在同一个面中, 所以嵌入到 f 和嵌入到 g 实际构成了这个奇环的二染色, 但奇环不能二染色, 矛盾。 □

定理 4.2. DMP 算法是正确的。

证明. 若该算法给出了一个平面嵌入, 那么 G 自然是平面图。设 G 的平面嵌入为 G' , 我们将其作为目标平面嵌入。

开始选定简单环时, 若环的方向与 G' 不同, 则将 G' 翻转依然可以得到 G 的平面嵌入。

若存在某个段 A 满足 $|F(A)| = 1$, 那么选出的附着路径 P 的嵌入方法实际上是唯一的, 故一定和 G' 的嵌入方法相同。

否则, 所有段 A 都满足 $|F(A)| \geq 2$, 只需讨论 A 中附着路径 P 的嵌入方法与 G' 不同的情况。考虑图 C 里 A 所在的连通分量 M , 只有 M 里的段可能与 P 冲突。设 M 中的点为 $A_1, A_2, \dots, A_k$ 。

由引理 4.1 和引理 4.2 可知 M 是二分图且 $F(A_i)$ 均相同且大小为 2, 于是在 G' 里将 M 中所有段所嵌入的面改为另一个面, 那么嵌入依然合法, 并且此时 P 的嵌入方法与 G' 的嵌入方法相同。

□

4.3 Tarjan 算法

为了方便判定一个平面嵌入是否合法, 这里给出以下引理。

引理 4.3. 对于一个平面嵌入 G , 若两点 x, y 之间有三条路径 p_1, p_2, p_3 , 这三条路径满足除了起点和终点之外两两不交。设 $(x, v_1), (x, v_2), (x, v_3)$ 分别为 p_1, p_2, p_3 上的第一条边, $(w_1, y), (w_2, y), (w_3, y)$ 分别为 p_1, p_2, p_3 上的最后一条边。若 x 周围三条边顺时针排列为 $(x, v_1), (x, v_2), (x, v_3)$, 则 y 周围三条边顺时针排列为 $(y, w_1), (y, w_3), (y, w_2)$ 。

证明. 该引理为约旦曲线定理的直接推论, 但约旦曲线定理的证明较为复杂, 详见 [5]。

□

我们一般使用引理 4.3 的逆否命题, 即两点 x, y 之间有不交的两条路径, 但不满足顺时针排列的结论, 那么 G 不是平面嵌入。

在介绍算法之前, 此处先给出一道与算法本身有关的例题。

例题 3 (HNOI2010 平面图判定加强版). 给定一个 n 个点 m 条边的简单无向图 G , 并给出图 G 的一个哈密顿回路, 请判定该无向图是否可平面。

$$1 \leq n \leq 5 \times 10^5, 1 \leq m \leq 10^6$$

解法

设哈密顿回路为 $v_1, v_2, \dots, v_n$, 对于任意 $1 \leq i \leq n$, v_i 到 $v_{i \bmod n+1}$ 之间有边。为了叙述方便, 不妨设 $v_0 = v_n, v_{n+1} = v_1$ 。

记 $a \rightarrow b$ 为一条原图中的边, $a \rightarrow b$ 为 a, b 在哈密顿回路中不经过 (v_1, v_n) 对应的路径。

考虑其余的边如何嵌入, 对于一条边 (v_i, v_j) , 不妨设 $i > j$, 这条边要么在 $v_1, v_2, \dots, v_n$ 这条链左侧, 要么在这条链右侧。若在左侧, 则 v_i 周围的顺时针排列为 $(v_i, v_{i-1}), (v_i, v_j), (v_i, v_{i+1})$, 若在右侧, 则 v_i 周围的顺时针排列为 $(v_i, v_{i+1}), (v_i, v_j), (v_i, v_{i-1})$ 。

不同侧的边互不影响, 只需考虑同一侧的边之间的影响。

引理 4.4. 图 G 中的平面嵌入合法当且仅当不存在同一侧的两条边 $(v_i, v_j), (v_k, v_l)$, 满足 $j < l < i < k$ 。

证明. 若存在这样两条边 $(v_i, v_j), (v_k, v_l)$, 则存在三条 v_i 到 v_l 的路径 p_1, p_2, p_3 , 其中 $p_1: v_i \rightarrow v_l, p_2: v_i \rightarrow v_k \rightarrow v_l, p_3: v_i \rightarrow v_j \rightarrow v_l$, 利用引理 4.3 可以得出该平面嵌入不合法。

若不存在这样的边, 则每一侧中的边 (v_i, v_j) 对应的区间要么不交, 要么一个包含另一个, 所以只要每次找出最长的区间对应的边, 将其画的足够靠近链 $v_i \rightarrow v_j$ 就不会与之前的边有交。

□

考虑建立图 G' , G' 中的点为 G 中哈密顿回路之外的边, G' 中的边表示根据上述引理, 这两条 G 中的边不能在同一侧。

那么一个合法的平面嵌入就对应了 G' 中的一个二染色, 即将 G' 的每个点标为 L 和 R 表示 G 中对应的边在左侧还是右侧。

如果暴力建出图 G' 后二分图染色可以做到时间复杂度 $O(m^2)$ 。

考虑扫描线, 按照边 $(v_i, v_j), i > j$ 中 i 从大到小排序, 若 i 相同按照 j 从小到大排序。维护两个栈 L, R , 分别表示当前在左侧的边与当前在右侧的边。

按照扫描线的顺序, 我们只需判断是否有某个已加入的 (v_k, v_l) , 满足 $j < l < i$, 所以栈中每条边的信息只需维护 l 的值。我们时刻保持 L, R 关于 l 的单调性, 即从栈底到栈顶的每条边的 l 单调不降。

对于一条边 $(v_i, v_j), i > j$ 我们要做如下流程:

1. 若栈 L 和栈 R 中元素的 l 满足 $l \geq i$ 直接弹出, 直到栈中所有元素均有 $l < i$ 。因为按照扫描线的顺序, 之后不可能满足 $j < l < i$ 这一限制。
2. 考虑将边 (v_i, v_j) 加入栈 L 中, 为此需要将栈 L 中与这条边冲突的移到栈 R 。
3. 注意到将一些边移到栈 R 可能会导致新的冲突, 于是再将原来栈 R 中的产生冲突的移到栈 L 。
4. 若此时仍不满足, 则无解, 否则回到 1 继续加入下一条边。

为了讨论该算法的正确性以及快速处理冲突时的转移, 新建栈 B , B 中的每个元素代表了加入当前这些边后 G' 的连通分量。所以 B 中每个元素中只要确定一条边就可以确定这个连通分量的所有边, 并且不同的连通分量互不影响。

实际上我们也保持着栈 B 关于 l 的单调性, 具体来说, 从栈底到栈顶任意两个相邻元素都满足前一个的 l 的最大值都小于等于后一个的 l 的最小值, 并且 B 中的元素在栈 L, R 中都恰好占据了一个区间。为了支持两个连通分量的快速合并以及弹出末尾元素, 每个连通分量 b 额外记录两个双向链表 L_b, R_b 分别表示按照栈 L 和 R 中的顺序串起 b 在 L 和 R 中的元素。

有了这个栈，我们可以将算法流程修改如下：

1. 若栈 B 中元素的链表末端的 l 满足 $l \geq i$ 直接弹出，直到栈中所有元素对应的连通分量的边均有 $l < i$ 。
2. 考虑若 B 的栈顶元素的 l 的最小值都比 j 严格大，则需要将整个连通分量移至 R 。
3. 若此时连通分量中既有 L 又有 R ，报告无解，否则先将该连通分量移至 R ，再从栈中弹出，并回到 2 处执行。
4. 否则若栈顶元素的 l 的最大值比 j 严格大，检查是否应该互换该连通分量中的 L, R 并执行，若均不行则报告无解，之后将其从栈中弹出。
5. 将所有弹出的元素以及目前考虑的这条边按顺序连成同一个连通分量并加入栈中，然后回到 1 继续加入下一条边。

注意到这里的做法类似单调栈，所以 L, R, B 都能保持单调性。此外，我们发现在这个算法流程中栈 L, R 是不必要的，只需维护每个连通块中的链表即可替代它们。由于每次弹栈都会带来一次连通分量的合并，所以弹栈次数为 $O(n)$ ，故总时间复杂度为 $O(n)$ 。

Tarjan 算法的基本思想依然是先找到一个环，并将图划分成若干个段。然后对于每个段分别递归下去判定是否平面，再将段之间的限制用类似例题的手段解决。

与前面算法相同，我们假设判定的图 G 为点双连通图。首先，对图 G 进行 DFS，记 DFS 树为 P ，那么图 G 的边被分为两类：树边和返祖边。我们将边定向，具体来说，树边从父亲指向孩子，返祖边从孩子指向祖先，除非特殊说明，否则只能按照定向后的方向走。

记 u 到 v 的树边为 $u \rightarrow v$ ， u 到 v 的返祖边为 $u \rightsquigarrow v$ ，树上 u 到 v 的路径为 $u \longrightarrow v$ 。为了叙述方便，我们将 u 重标号为 u 的 DFS 序。

记 low_u 为从 u 出发，经过若干条定向后的树边，再经过至多一条返祖边能到达的点标号的最小值， low'_u 为从 u 出发，经过若干条定向后的树边，再经过至多一条返祖边能到达的点标号的严格次小值。特别地，若 $low_u = u$ ，则定义 $low'_u = u$ ，这也是 $low_u = low'_u$ 的唯一情况。 low_u 和 low'_u 都可以在 DFS 的过程中顺便求出来。

接下来，我们要将图 G 划分成若干条路径，再将这些路径按照特定顺序嵌入到平面中。为了找到这个路径划分，我们将每条边给定边权。

定义 4.3. 定义一条边 (u, v) 的边权 $\phi((u, v))$ 为：

$$\phi((u, v)) = \begin{cases} 2v & u \rightsquigarrow v \\ 2low_v & u \rightarrow v \wedge low'_v \geq u \\ 2low_v + 1 & u \rightarrow v \wedge low'_v < u \end{cases}$$

这些边权的意义会在之后的引理中得到体现。

接下来，我们对每个点的出边按照边权从小到大排序。注意，为了不影响总复杂度，这里需要使用桶排解决，时间复杂度 $O(n)$ 。

接着，我们再对图 G 进行 DFS。假设当前到了点 u ，我们按照排序后的顺序遍历 u 的出边，若经过的是树边 $u \rightarrow v$ ，将该边加入当前路径，并递归到 v 搜索。否则 $u \rightarrow v$ 为返祖边，将该边作为当前路径的最后一条边，之后新建一条路径作为当前路径。

算法结束后，图 G 被划分成了若干条路径，且每条边恰好在一个路径中。对于一条划分出的路径，它一定是先经过若干条树边，再经过一条返祖边得到的。我们记 u 到 v 的一条划分出的路径为 $u \Rightarrow v$ 。

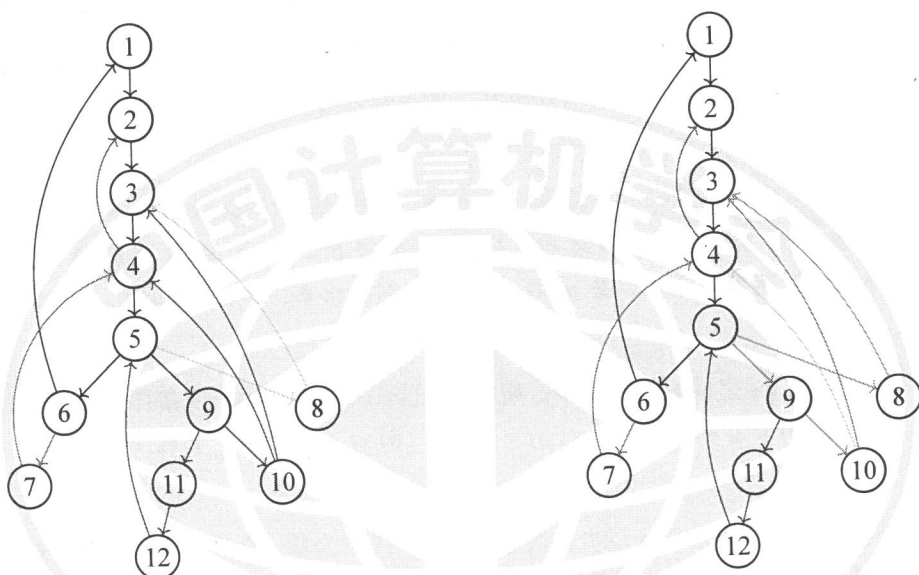


图 1: 段与划分路径

下面来分析划分出的路径含有的性质。

引理 4.5. 对于一条划分出的路径 $p: s \Rightarrow t$ ，考虑在开始划分这条路径时所有还未经过的返祖边，那么 t 即为 s 的后代中能通过这些返祖边到达的编号最小的点。特别地，如果 v 是路径 p 上的点且 $v \neq s, v \neq t$ ，则 $t = low_v$ 。

证明. 对于一条边 e ，它的边权实际上可以表示从它开始，最多经过一条返祖边（包括 e 本身）能到达的编号最小的点，并且每次访问的是还未访问过的最小边权的边，故引理显然成立。 $\square$

引理 4.6. 对于一条划分出的路径 $p: s \Rightarrow t$ ，那么 t 是 s 的祖先（可以为本身），也即 $t \rightarrow s$ 合法。若 p 是划分出的第一条路径，那么 p 是一个简单环，否则 p 是一个简单路径。若 p 不是第一条路径，那么 p 恰好包含两个点 s, t ，满足它出现在之前被划分出的路径中。

证明. 若 p 仅包含一个返祖边, 那么 t 显然是 s 的祖先, 且 $s \neq t$. 否则考虑 p 中的第一条边 $s \rightarrow v$, 此时 v 是第一次被经过, 根据引理 4.5, 有 $t = \text{low}_v$, 因为 G 为点双连通图, 所以 t 是 s 的祖先, 且 $t = s$ 当且仅当 $s = 1$. 实际上只有 t 可能与路径中其它结点重复, 注意到只有第一条路径满足 $s = 1$, 那么 p 是第一条路径时为简单环, 否则 t 是 s 的祖先且 $s \neq t$, 那么 p 为简单路径. 由 DFS 的性质, 对于一个点 $v \neq 1$, v 第一次被经过一定在某条路径 $p': s' \rightarrow t'$ 的中间, 即 $v \neq s', v \neq t'$, 并且所有在路径中间的点一定是第一次被经过, 故 p 中恰好包含两个不是第一次被经过的点.

□

引理 4.7. 对于两条划分出的路径 $p_1: s_1 \Rightarrow t_1, p_2: s_2 \Rightarrow t_2$, 若 p_1 在 p_2 之前被划分出来, 并且 s_1 是 s_2 的祖先 (可以为本身), 那么 $t_1 \leq t_2$.

证明. p_2 中所含的返祖边从 s_1 的后代出发, 且开始划分 p_1 时这条边还未经过, 由引理 4.5 可知 $t_1 \leq t_2$.

□

引理 4.8. 对于两条划分出的起点终点相同的路径 $p_1: s \Rightarrow t, p_2: s \Rightarrow t$, 设 v_1 为 p_1 的第二个点, v_2 为 p_2 的第二个点, 若 p_1 在 p_2 之前被划分出来, $v_1 \neq t$ 且 $\text{low}'_{v_1} < s$, 那么 $v_2 \neq t$ 且 $\text{low}'_{v_2} < s$.

证明. 因为 p_1 在 p_2 之前被划分出来, 所以 $\phi((s, v_1)) \leq \phi((s, v_2))$, 而条件保证了 $\phi((s, v_1)) = 2t + 1$ 且 $\phi((s, v_2)) \leq 2t + 1$, 故 $\phi((s, v_2)) = 2t + 1$, 即 $v_2 \neq t$ 且 $\text{low}'_{v_2} < s$.

□

我们尝试将划分出的路径逐步嵌入至平面中. 令 c 为第一条路径, 由引理 4.6 可知 c 是一个简单环. c 中含有若干条树边 $1 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n$ 以及一条返祖边 $v_n \rightarrow 1$. 沿用 DMP 算法中的定义, G 被 c 划分成了若干个段, 每个段要么是一条返祖边 (v_i, u) , 要么是一条树边 (v_i, u) 加上 u 子树内的边以及从 u 子树发出的返祖边, 所以一条从环 c 中出发且不是环 c 中的边 (v_i, u) 恰好对应了一个 G 中的段.

对于每一段, 它里面的所有被划分出的路径被划分的时间一定是一个连续的区间. 而且在第一条路径划分之后, DFS 的过程导致了进入每个段的顺序是按照 v_i 递减的.

由约旦曲线定理, 每个段要么在链 $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n$ 左侧, 要么在链右侧. 类似上面例题解法中的性质, 边 (v_i, u) 对应的段若在左侧, 则 v_i 周围顺时针排列为 $(v_i, v_{i-1}), (v_i, u), (v_i, v_{i+1})$, 若在右侧, 则 v_i 周围顺时针排列为 $(v_i, v_{i+1}), (v_i, u), (v_i, v_{i-1})$.

对于一个段 S , 设它里面划分出的第一条路径为 p , 假设所有在 p 之前的路径都已经嵌入至平面中, 那么以下引理可以给出 p 能嵌入在某一侧的充要条件, 这对之后的算法至关重要.

引理 4.9. 路径 $p: v_i \Rightarrow v_j$ 能被嵌入在左 (右) 侧当且仅当没有满足 $v_j < v_k < v_i$ 的返祖边 (x, v_k) 被嵌入到左 (右) 侧.

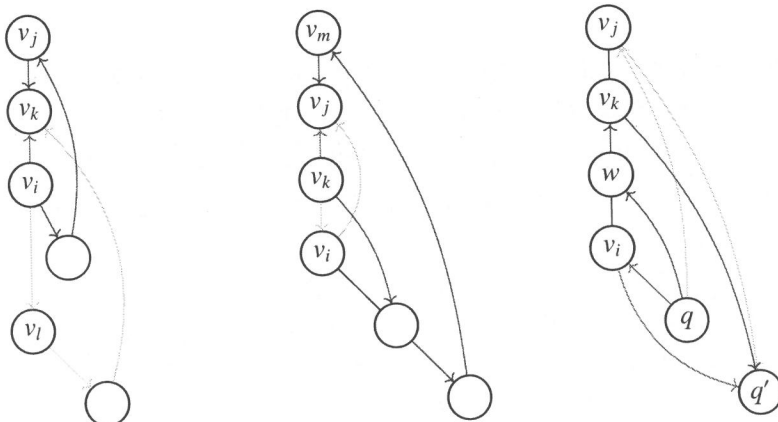


图 2: 证明图例

证明. 若没有满足 $v_j < v_k < v_i$ 的返祖边, 则目前嵌入的边中没有进入或离开 v_i 与 v_j 之间的边, 于是只要将路径 p 足够靠近 v_i 与 v_j 之间的链, 那么嵌入一定合法。

考虑若存在 $v_j < v_k < v_i$ 的返祖边 (x, v_k) , 下证路径 p 无法平面嵌入。不妨设 p 要嵌入到左侧。注意, 以下分类讨论中构造的路径均忽略原先的定向。

设 (x, v_k) 所在的段为 S' , 所在划分出的路径为 $p' : o \Rightarrow v_k$ 。设 S' 中第一条划分出的路径为 $v_l \Rightarrow v_m$, 由划分顺序, 可知 $v_l \geq v_i$ 。

1. 若 $v_l > v_i$, 构造三条 v_i 到 v_k 的路径 p_1, p_2, p_3 , 其中 $p_1 : v_i \rightarrow v_k, p_2 : v_i \rightarrow o \Rightarrow v_k, p_3 : v_i \Rightarrow v_j \rightarrow v_k$, 由引理 4.3 可知 p 无法嵌入至左侧。

2. 若 $v_l = v_i$, 由引理 4.7 可知 $v_m \leq v_j$ 。

2.1. 若 $v_m < v_j$, 构造三条 v_k 到 v_j 的路径 p_1, p_2, p_3 , 其中 $p_1 : v_k \rightarrow v_j, p_2 : v_k \rightarrow v_i \Rightarrow v_j, p_3$ 为段 S' 中任意一条 v_k 到 v_m 的路径加上 $v_m \rightarrow v_j$, 由引理 4.3 可知 p 无法嵌入至左侧。

2.2. 若 $v_m = v_j$, 设 y, z 分别为 p, p' 上的第二个顶点, 因为段 S' 中包含返祖边 (x, v_k) , $z \neq v_m$ 并且 $low'_z < v_i$, 而且 p' 在 p 之前被划分出来, 由引理 4.8 可知 $y \neq v_j$ 且 $low'_y < v_i$ 。因为 $low_y = v_j$ 所以 $low'_y > v_j$ 。记 $w = low'_y$, 路径为 $o' \Rightarrow w$ 。

设 q 为路径 $v_i \Rightarrow v_j$ 与路径 $v_i \rightarrow o' \Rightarrow w$ 的交中编号最大的点, 那么可以将这两条路径分别拆成两段, 即 $v_i \Rightarrow v_j = v_i \rightarrow q \rightsquigarrow v_j$ 与 $v_i \rightarrow o' \Rightarrow w = v_i \rightarrow q \rightsquigarrow w$, 其中 $u \rightsquigarrow v$ 指代 u 到 v 的一条特定的路径。类似地定义 q' 为路径 $v_l \Rightarrow v_m$ 与路径 $v_l \rightarrow o \Rightarrow v_k$ 的交中编号最大的点, 并将这两条路径拆成两段。

构造三条 q 到 q' 的路径 p_1, p_2, p_3 , 其中 $p_1 : q \rightarrow v_i \rightarrow q', p_2 : q \rightsquigarrow v_j \rightsquigarrow q', p_3 : q \rightsquigarrow w \rightarrow v_k \rightsquigarrow q'$, 由引理 4.3 可知 p 无法嵌入至左侧。

□

可以看出,该引理和上面的例题中给出的引理 4.4 十分相似。实际上,在上面的例题经过扫描线之后,这两条引理对于一条路径 $p: v_i \Rightarrow v_j$ 来说完全相同。

问题在于,此处的路径 p 为段 S 第一条划分出的路径,而段 S 的其它路径并未纳入考虑。幸运的是,加入路径 p 后,可以将 p 以及 $v_j \rightarrow v_i$ 视作环 c ,递归到 S 中进行判定,因为 v_j 为 S 中返祖边能到达的最小位置,若 p 不会与之前嵌入的路径产生矛盾,则 S 中所有路径均不会与之前的路径产生矛盾,故只需判定 S 内部能否嵌入即可。

所以,我们只需执行上面例题的算法,并在成功插入路径 p 至 L 后做以下操作。

首先,在栈 R 中置入一个栈底标记,之后递归判定段 S 是否合法。若段 S 合法,那么此时栈 B 中会多出若干个连通块。由于段 S 只能位于环 c 的一侧,所以栈 B 中段 S 的连通块需要合并成一个连通块,并且都在 L 中。所以若栈 B 内段 S 的连通块里同时含有 L 和 R 则无解,否则将其弹出,顺便回收掉栈 R 的标记,将这些连通块连同路径 p 一起组成一个大的连通块,再插入栈 B 中。栈 R 中栈底标记的作用在于方便判定哪些连通块在段 S 中以及防止递归时与前面的元素产生连边。

综上所述,我们得到了一个时间复杂度 $O(n)$ 的平面图判定算法。

5 平面图点着色

着色问题是图论中一类重要的问题。在平面图中,我们有广为人知的四色定理。

定义 5.1 (k -着色). 对于一张无向图 G ,若将每个点 u 上分配一个颜色 c_u ,使得对于任意边 (u,v) ,有 $c_u \neq c_v$,则称这种分配方式为一种着色方案。若图 G 存在一种使用不超过 k 种颜色的着色方案,则称这种方案为 k -着色方案,图 G 为可 k -着色的。满足图 G 是可 k -着色的最小的 k 称为图 G 的色数 $\chi(G)$ 。

定理 5.1 (四色定理). 任意平面图 G 都是可 4-着色的。

证明见 [8]。四色定理的证明过程给出了一个在 $O(n^2)$ 时间构造平面图的 4-着色方案的算法。但其过于复杂,故不在此介绍。

实际上,贪心的着色方法能给出一个 6-着色方案。为此,我们需要以下引理。

引理 5.1. 平面图中一定存在某个点 u ,使得点 u 的度数不超过 5。

证明. 假设所有点度数均 ≥ 6 ,那么有 $3V \leq E$,这与引理 2.1 矛盾。

□

那么对于平面图 G ,我们将点 u 从图 G 中删去,得到图 G' 。我们先将图 G' 递归下去着色,最后再将点 u 的颜色置为没有在与它相连的点中出现过的颜色即可。

稍微改进一下这个算法可以给出一个 5-着色方案。我们只需讨论点 u 度数为 5, 且与它相连的点颜色互不相同的情形。

不妨设 u 连向的点按逆时针顺序分别为 v_1, v_2, v_3, v_4, v_5 , 其中点 v_i 的颜色为 i 。

考虑所有颜色为 1, 3 的点的导出子图 H , 若 v_1 与 v_3 不连通, 则可将 v_1 所在连通块中的颜色翻转 (1 变为 3, 3 变为 1), 此时我们可以将点 u 的颜色置为 1。

否则, 考虑 v_1 到 v_3 在 H 中的一条简单路径与 u 形成的环, 由约旦曲线定理, 这个环将平面分成了内部和外部。此时考虑所有颜色为 2, 4 的点的导出子图 H' , 那么 v_2 和 v_4 必定不连通。类似地, 我们将 v_2 所在连通块颜色翻转, 则可将点 u 的颜色置为 2。

综上所述, 这个算法可以在 $O(n^2)$ 时间内给出平面图的 5-着色方案。

即使平面图都为可 4-着色的且判定可 2-着色十分容易, 判定某张平面图是否可 3-着色仍为 NPC 问题。所以接下来我们关心一类限制更强的图。

定义 5.2 (三角剖分图). 定义平面嵌入 G 为三角剖分图当且仅当 G 的每个内部面边界上的边数均为 3。

对于三角剖分图, 我们可以有效判定它是否可 3-着色。

引理 5.2. 三角剖分图 G 可 3-着色当且仅当 G 不含奇轮子图。其中奇轮子图指的是一个奇环和一个特定点, 该特定点向环上所有点连边。

证明. 首先奇轮子图显然无法 3-着色, 只需证不含奇轮子图的三角剖分图可 3-着色。

考虑归纳, $n \leq 3$ 时显然成立。若 G 不为点双连通图, 则显然可以将其拆成若干个点双分别染色, 最后合并。下设图 G 为点双连通图。

我们尝试找到某个点, 使得删掉这个点后 G 依然点双连通。考虑外部面边界构成的环 c , 在 c 上任意选定一个外部面边界上的点 u , 设 u 的出边极角排序后为 $(u, v_1), (u, v_2), \dots, (u, v_k)$, 由于 G 是三角剖分图, 所以 v_i, v_{i+1} 之间一定存在边 (否则 $(u, v_i), (u, v_{i+1})$ 所在的面边数 > 3)。

删掉 u 点后外部面边界的环由 $c: \dots \rightarrow v_k \rightarrow u \rightarrow v_1 \rightarrow \dots$, 变为了 $c': \dots \rightarrow v_k \rightarrow v_{k-1} \rightarrow \dots \rightarrow v_2 \rightarrow v_1 \rightarrow \dots$, 若没有从 u 点出发的环 c 的弦, 则 c' 依然是简单环, 所以删掉 u 后图 G 为点双连通图。

否则设弦为 (u, v) , 它将图 G 分成了两个部分, 这两个部分的公共点只有 u, v 。根据归纳假设, 这两个部分都有 3-着色方案, 将其中一个部分的方案中的颜色重标号一下即可得出图 G 的 3-着色方案。

接着, 我们对于所有满足 $1 < i < k$ 的 v_i 考虑, 由之前的讨论, 与 v_i 相邻的点 w_j 之间必定可以连成一条链, 而所有 w_j 的颜色必须与 v_i 不同, 故 w_j 的颜色相间, 又因为图 G 中不含奇轮子图, 为了不使 w_j 和 u 以及 v_i 形成奇轮子图, 所以序列 w_j 的长度必定为奇数, 所以序列 w_j 两端的点颜色必定相同。而两端的点实际上恰为 v_{i-1} 和 v_{i+1} 。至此, 我们发现 v_i 的颜色也是相间的, 所以直接将 u 点染为与它们都不同的颜色即可。

□

例题 4 (MIN&MAX II³). 对于元素互不相同的长为 n 的整数序列 a , 我们建立一个无向简单图 $G(a)$, 有 n 个结点从左到右排成一排, 编号为 $1 \sim n$, 每个点向它左右最近的比它大的点以及比它小的点连边, 这里比较两个点的大小指的是这两个点 a_i 的大小。对于序列 a , 记 $a[l:r]$ 表示 a 中子区间 $[l, r]$ 构成的序列。

给定一个长为 n 的排列 a , q 次询问, 每次询问某个区间 $[l, r]$ 中的所有子区间 $[l', r']$ 中 $\chi(G(a[l':r']))$ 的最大值以及达到这个最大值的子区间数。

$$1 \leq n, q \leq 3 \times 10^5$$

解法

令第 i 个结点的坐标为 (i, a_i) , 边为两点之间的线段。

对于任意序列 a , $G(a)$ 是平面图。假设有两条边 $(i, j), (k, l)$ 交叉, 不妨设 $i < j, a_i < a_j$ 且点 j 是 i 右边最近的比它大的点。由于这两条边交叉, 不妨设 $a_k > a_l, a_k > a_i, a_l < a_j$, 因为 j 是 i 右面最近的比它大的点, 所以 $k < i$ 或 $k > j$ 。

1. 若 $l < i$ 或 $l > j$, 则 k, l 必定分别在区间 $[i, j]$ 两侧, 此时 k 到 l 以及 l 到 k 一定会分别被 i, j 两个点挡住, 从而边 (k, l) 不可能存在。从 x 到 y 被某个点 u 挡住是指 $x < u < y$ 且 $a_x - a_u$ 和 $a_x - a_y$ 同号, 那么 x 不可能向 y 连边。
2. 若 $i < l < j$, 则有 $a_l < a_i$, 那么若 $k < i$, 则 k 到 l 以及 l 到 k 一定会被 i 点挡住, 若 $k > j$, 若 $a_k > a_j$ 会被 j 点挡住, 否则根本不会出现交叉。

此处给出一个结论, 对于任意点 u , 设与它相连的边极角排序后为 $(u, v_1), (u, v_2), \dots, (u, v_k)$, 那么对于任意两条相邻的边 $(u, v_i), (u, v_{i \bmod k+1})$, v_i 与 $v_{i \bmod k+1}$ 有边的充要条件是在以 u 为原点的坐标系中这两个点不在两个不同且不相邻的象限中。结论的证明需要进行较为繁琐但并不困难的分类讨论, 此处略去。

对于任意序列 a , $G(a)$ 是三角剖分图。假设有一个边数 ≥ 4 的面, 边界为环 $c: v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_l \rightarrow v_1$, 不妨设 v_1 为所有 v_i 的最小值, 那么由上述结论可知 v_2 与 v_l 之间必定有边, 矛盾。

接着, 对于一个点 u , 存在以 u 为特定点的奇轮子图当且仅当以 u 为原点的四个象限均有点且 u 的度数为奇数。这是上述结论的简单推论。

注意到 $G(a[l:r])$ 其实就是 $G(a)$ 中区间 $[l, r]$ 中的点的导出子图, 那么不同的奇轮子图最多只会有 n 个, 可以直接预处理出来。

那么我们有以下判定方法。

1. $\chi(G(a[l:r])) = 1$ 当且仅当 $l = r$ 。

³<https://loj.ac/p/554>

2. $\chi(G(a[l;r])) = 2$ 当且仅当 $l < r$ 且 $a[l;r]$ 单调上升或者单调下降。否则一定存在某个 $l < i < r$ 满足 $a_{i-1} < a_i, a_{i+1} < a_i$ 或 $a_{i-1} > a_i, a_{i+1} > a_i$, 那么 $i-1, i, i+1$ 形成了三元环, 无法 2-着色。
3. $\chi(G(a[l;r])) = 3$ 当且仅当 $a[l;r]$ 不单调并且 $G(a[l;r])$ 中没有奇轮子图。
4. $\chi(G(a[l;r])) = 4$ 当且仅当 $G(a[l;r])$ 有奇轮子图。

注意到当区间左端点固定时, 右端点往右移动时图的色数单调不降。于是只需处理出每个左端点, 图的色数变化的位置, 之后从左往右扫描线, 用线段树维护即可。

时间复杂度 $O((n+q)\log n)$ 。

6 平面图完美匹配计数

求一张图的完美匹配计数是很困难的, 甚至对于二分图而言, 这个问题都是 #P-complete 的。但 FKT 算法能在多项式时间复杂度内算出平面图的完美匹配个数。

FKT 算法的主要思想是, 将完美匹配计数转化成计算一个反对称矩阵的 pf, 而后者可以转化为计算行列式, 从而得到一个高效的算法。

我们首先需要找到给定平面图的一个平面嵌入 G 。实际上, 之前提到的两个判定平面图的算法均给出了一个平面嵌入的方法。

首先若 G 的点数是奇数显然无解, 否则记 G 的点数为 $2n$ 。若 G 不连通, 则对每个连通块分别执行以下算法, 下设 G 连通。定义 G 的邻接矩阵 $\{G_{i,j}\}$, 若 i 和 j 之间有边则 $G_{i,j} = 1$, 否则 $G_{i,j} = 0$ 。那么 G 的完美匹配数 $\text{PerfMatch}(G)$ 为

$$\text{PerfMatch}(G) = \frac{1}{2^n n!} \sum_p \prod_{i=1}^n G_{p_{2i-1}, p_{2i}}$$

其中 p 取遍所有 $1 \sim 2n$ 的排列。前面的系数是由于每一种完美匹配都恰好对应 $2^n n!$ 种不同的排列 p 。

对于一个矩阵 A , 我们定义 $\text{pf}(A)$ 为

$$\text{pf}(A) = \frac{1}{2^n n!} \sum_p \text{sgn}(p) \prod_{i=1}^n A_{p_{2i-1}, p_{2i}}$$

设 $\sigma(p)$ 表示排列 p 的逆序数, 那么定义 $\text{sgn}(p) = (-1)^{\sigma(p)}$ 。

注意到这两个式子十分相似, 我们想到假如可以构造一个矩阵 A , 它相比邻接矩阵只改变了某些位置的符号, 使得这些符号能与前面的 $\text{sgn}(p)$ 抵消掉, 那么就可以通过计算 pf 来求出完美匹配数了。

我们先将平面嵌入 G 的每条边定向, 之后构造矩阵 A 为

$$A_{i,j} = \begin{cases} 1 & i \rightarrow j \in E' \\ -1 & j \rightarrow i \in E' \\ 0 & i \rightarrow j, j \rightarrow i \notin E' \end{cases}$$

其中 E' 为图 G 的边集 E 定向以后的边集。容易发现矩阵 A 为反对称矩阵, 即 $\forall i, j, A_{i,j} = -A_{j,i}$ 。

6.1 关键引理

引理 6.1. 若平面嵌入 $G(V, E)$ 的定向满足, 对于任意一个内部面 f , f 的边界上都有奇数条顺时针方向的边, 那么每一个完美匹配向 $\text{pf}(A)$ 的贡献均相等, 都为 1 或都为 -1。

证明. 记 $f(p) = \text{sgn}(p) \prod_{i=1}^n A_{p_{2i-1}, p_{2i}}$ 。

首先对于一个完美匹配 M , 证明它对应的任意两个排列 p, p' , 都有 $f(p) = f(p')$ 。

- 对于 p 中相互匹配的元素 p_{2i-1}, p_{2i} , 记 p'' 为在 p 中交换 p_{2i-1}, p_{2i} 得到的排列, 由于 $\text{sgn}(p'') = -\text{sgn}(p)$ 且 $A_{p_{2i}, p_{2i-1}} = -A_{p_{2i-1}, p_{2i}}$, 所以 $f(p'') = f(p)$ 。
- 对于 p 中相互匹配的两对元素 $(p_{2i-1}, p_{2i}), (p_{2j-1}, p_{2j})$, 记 p'' 为在 p 中交换 p_{2i-1}, p_{2j-1} 并交换 p_{2i}, p_{2j} 得到的排列, 那么 $\text{sgn}(p'') = \text{sgn}(p)$, 所以 $f(p'') = f(p)$ 。

容易发现, 这两种交换操作一定能使 p 变为 p' , 故 $f(p) = f(p')$ 。

定义两个集合 A, B 的对称差 $A \Delta B = (A \setminus B) \cup (B \setminus A)$ 。对于大小为 k 的边集 $E = (E_{i,0}, E_{i,1})$, 记 $g(E) = \prod_{i=1}^k A_{E_{i,0}, E_{i,1}}$ 。

之后考虑两个完美匹配 M, M' 以及它们对应的两个排列 p, p' , 求出这两个完美匹配的对称差 $M \Delta M'$, 那么有

$$\frac{g(M')}{g(M)} = \frac{g(M' \setminus M)}{g(M \setminus M')} = \frac{g(M' \setminus M)g(M \setminus M')}{g(M \setminus M')^2} = g(M' \setminus M)g(M \setminus M') = g(M \Delta M')$$

考虑图 $G'(V, M \Delta M')$, 那么每个点的度数要么为 0, 要么为 2。于是整张图由若干个环以及若干个孤立点组成。进一步地, 对于每个环, 完美匹配 M 中的边与 M' 中的边交替出现, 于是每个环一定为偶环。

对于每个偶环分别考虑, 设当前的偶环为 $c: v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k \rightarrow v_1$, 下面证明, 每个偶环中顺时针的边数 a 必为奇数。

首先, 记严格在偶环内部的点数为 v , 边数为 e , 面数为 f 。考虑对偶环内部 (包括偶环自身) 使用欧拉定理, 可得 $(v+k) - (e+k) + (f+1) = 2$, 整理得 $v - e + f = 1$ 。

记 $C(u)$ 为面 u 内顺时针方向的边数, 由条件知 $C(u) \equiv 1 \pmod{2}$ 。考虑将所有偶环内部的面 u 的 $C(u)$ 求和, 这实际上对应了偶环中所有顺时针方向的边数 a 加上严格在偶环内部的边数 e 。原因在于偶环中的边被算进去当且仅当它在某个面中为顺时针, 那么在偶环中自



然也是顺时针，而偶环内部的边无论如何定向，都会被恰好一侧的面看作顺时针的边。所以我们有 $\sum_u C(u) = a + e$ 。在 $\text{mod } 2$ 意义下继续推导

$$f \equiv \sum_u C(u) = a + e \equiv a + v + f - 1 \pmod{2}$$

整理得

$$a \equiv 1 - v \pmod{2}$$

由于完美匹配 M 交替选择了偶环中的边，而这分离了偶环的内部与外部，所以内部的点数 v 必须为偶数才能保证存在这样的完美匹配，故 a 为奇数。

设 M'' 为只在 M 中改变所有偶环中的边是否选择后得到的完美匹配，那么 $c = M \Delta M''$ 。

不妨设排列 p 形如 $v_1, v_2, \dots, v_k, p_{k+1}, \dots, p_{2n}$ ， p'' 形如 $v_2, v_3, \dots, v_k, v_1, p_{k+1}, \dots, p_{2n}$ ，则有 $\text{sgn}(p'') = -\text{sgn}(p)$ ， $g(M'') = g(M)g(M \Delta M'') = g(M)g(c) = -g(M)$ ，所以 $f(p) = f(p'')$ 。

经过若干次操作以后，一定能使 p 变为 p' ，故 $f(p) = f(p')$ 。

□

6.2 定向方法

现在需要找出满足引理 6.1 条件的一种定向。

引理 6.2. 给定平面嵌入 G 以及对偶图 G^* ，对于任意一棵 G 的生成树 T_1 ，所有不在 T_1 中的边 e 的对偶边 e^* 以及 G^* 中所有点构成的子图 T_2 是 G^* 的一棵生成树。

证明. 首先 T_2 无环，否则任取 T_2 的一个简单环，根据引理 3.1，这对应了图 G 中的一个割，与图 G 中含有的生成树 T_1 矛盾。

其次 T_2 连通，否则任取两个连通块，它们之间的边均不在 T_2 中，根据引理 3.1，这对应了图 G 中的一个环，与 G 中含有的生成树 T_1 矛盾。

所以 T_2 是 G^* 的一棵生成树。

□

那么我们执行以下算法：

1. 找出平面嵌入 G 的对偶图 G^* 。
2. 找出图 G 的任意一棵生成树 T_1 ，并对其中的边任意定向。
3. 找出 G 里所有不在 T_1 中的边，根据引理 6.2，它们的对偶边构成了 G^* 的生成树 T_2 。
4. 找出一个 T_2 中的不代表外部面的叶子 v 以及它相连的边 e^* 。

5. 注意到叶子 v 对应的面中仅含一条边 e 还未确定方向。若现在有奇数条顺时针的边，则将 e^* 定向为逆时针，否则将 e^* 定向为顺时针。
6. 将 v 与 e^* 从 T_2 中删除。若 T_2 中仅含一个外部面对应的点，则结束算法，否则回到 4 继续执行。

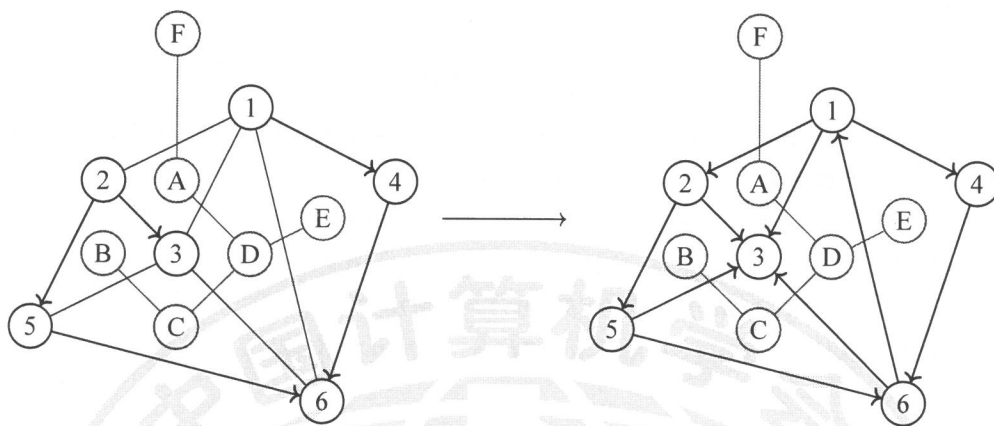


图 3: 定向示例

注意到我们每次删除一个点后，对应的内部面中所有边均完成定向且恰好有奇数条顺时针的边。所以在算法结束时，我们得到了一种满足引理 6.1 的定向。

6.3 计算 pf

引理 6.3. 对于一个反对称矩阵 A 和任意矩阵 B ，有

$$\text{pf}(A)^2 = \det(A)$$

$$\text{pf}(BAB^T) = \det(B)\text{pf}(A)$$

证明见 [11]。

由引理 6.1, $\text{PerfMatch}(G) = |\text{pf}(A)|$ 那么我们可以直接算出 A 的行列式，再开根即可。时间复杂度 $O(n^3)$ 。

但我们通常是在 $\mathbb{F}_p$ 上做运算，而此时开根会得出两个解。我们无法区分这两个数在 $\mathbb{F}_p$ 下的正负性，这带来了一些麻烦。

实际上，求解过程引入了两个正负号。第一个在于我们不确定每个完美匹配给答案贡献的是 1 还是 -1，所以我们需要通过求解一般图完美匹配的一个解来判定贡献的正负号。第二个在于算 pf 时我们通过引理 6.3 得到了答案的平方，而开根时会带来正负号。

为了避免开根,可以通过在引理 6.3 中令矩阵 B 为初等行变换,之后便可以执行一个类似高斯消元的过程,具体可见参考文献 [12] [13]。参考文献 [14] 给出了在任意整环上时间复杂度 $O(n^3)$ 的计算 pf 的算法。

可以发现,尽管 FKT 算法本身比较简洁,但在最后的符号处理上却较为繁琐。在出题时,可以考虑通过计算两个完美匹配组成有序对的方案数等方式来避免正负号的判断。

6.4 例题

例题 5. Alice 和 Bob 在下棋。有一个 $n \times m$ 的棋盘,棋盘的每个位置有一个权值 $a_{i,j}$,开始时棋盘中有 k 个格子上有一个棋子。Alice 和 Bob 每人轮流选择棋盘的一个没有棋子的格子,并放上棋子, Alice 先手,无棋可下的人输。Bob 选择的格子必须与 Alice 上次选择的格子有公共边。

设一步棋的精彩度为这步棋所在位置的权值与对手上一步棋所在位置的权值之积,特别地,第一步棋的精彩度为 0。整局棋的精彩度为每步棋的精彩程度之和。

如果两个人的策略都为随机选择一个合法位置,请计算所有可能的 Bob 胜利的局面中,整局棋精彩度的平均数。局面不同当且仅当在某一步棋走的不同。答案对 998244353 取模,保证 Bob 胜利的局面数不为 998244353 的倍数。

$$1 \leq n \times m \leq 400, 0 \leq k \leq nm$$

解法

建立图 G , 点由所有开始没有棋子的格子组成,若两个格子间有公共边,则将格子对应的点连边。设 G 的点数为 $2t$ 。

考虑将 Bob 选择的格子与 Alice 上次选择的格子匹配,则 Bob 胜利时会给出图 G 的一个完美匹配。

反过来,考虑一个完美匹配 $M = \{M_{i,0}, M_{i,1}\}$ 对答案产生的贡献。首先,一个完美匹配会产生 $2^t t!$ 个不同的局面。接着考虑每一手棋在这些局面下对答案的贡献,我们发现只需计算以下两个式子的答案 p_M, q_M 以及完美匹配数即可。

$$p_M = \sum_{i=1}^t a_{M_{i,0}} a_{M_{i,1}}$$

$$q_M = \sum_{i=1}^t \sum_{j=i+1}^t (a_{M_{i,0}} + a_{M_{i,1}})(a_{M_{j,0}} + a_{M_{j,1}})$$

设 s 表示 G 中所有 a_i 的和。可以观察到 $2(p_M + q_M) + \sum_{i=1}^{2t} a_i^2 = s^2$, 于是只需计算 p_M 。

考虑如何计算 p_M 。设 $U_i(x) = a_{M_{i,0}} a_{M_{i,1}} x + 1$, 只需算 $[x^1] \prod_{i=1}^t U_i(x)$ 。

注意到 G 的边如果带权,则 FKT 算法给出的是每个完美匹配边权乘积的和。那么将边 (u, v) 的边权设为 $a_u a_v x + 1$, 同时在 $\text{mod } x^2$ 的意义下求 pf, 即可算出所有完美匹配中 $[x^1] \prod_{i=1}^t U_i(x)$ 的和, 记为 p 。



此时，我们又碰到了符号的问题。当然我们可以通过上述符号处理得出答案，但有些繁琐。

首先通过求行列式求出 pf 的平方。设完美匹配数为 u ，那么第一个行列式算出的答案为 $(px + u)^2 \equiv 2pux + u^2 \pmod{x^2}$ ，虽然我们没法得到 p 和 u 分别是什么，但我们可以得知 $\frac{p}{u} = \frac{2pu}{2 \cdot u^2}$ 。

可以发现，答案能表示为 $c_1 \cdot \frac{p}{u} + c_2$ ，其中 c_1, c_2 为容易计算的常数。

时间复杂度 $O(n^3)$ 。

■

7 总结

对偶图是研究平面图的一个重要工具。本文简要介绍了对偶图中环与割的对应，以及对偶图的求法和点定位的方法。

平面图判定是平面图中的关键问题。本文中给出了两种算法，分别是 DMP 算法与 Tarjan 算法。这两种算法不仅可以判定一张图是否可平面，而且对于平面图可以构造具体的平面嵌入。

平面图的特殊性质使得它在某些一般图中难以解决的问题上有较为优秀的解法。对于点着色问题，本文给出了三角剖分图色数的求法。对于完美匹配计数问题，本文给出了 FKT 算法。

平面图中还有一些问题例如子图同构问题在学术界已经有较为成熟的算法。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平面图的相关问题与算法。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练杨耀良的指导。

感谢平邑一中刘闽波老师的关心和指导。

感谢常瑞年同学和戴江齐同学为本文验稿。

感谢其他同学与我交流讨论。

感谢父母对我的关心与支持。

参考文献

- [1] Wikipedia, the free encyclopedia, "Planar graph", https://en.wikipedia.org/wiki/Planar_graph



- [2] Łącki, J., & Sankowski, P. (2017). Optimal Decremental Connectivity in Planar Graphs. *Theory Comput Syst* 61, 1037–1053. doi: 10.1007/s00224-016-9709-x
- [3] 钱桥. (2013). 平面图的处理方法. NOI WC 讲课.
- [4] Kohnert, A. (2004). Algorithm of Demoucron, Malgrange, Pertuiset.
- [5] Cairns, S. S. (1951). An Elementary Proof of the Jordan-Schoenflies Theorem. *Proceedings of the American Mathematical Society*, 2(6), 860–867. doi: 10.2307/2031698
- [6] Hopcroft, J., & Tarjan, R. (1974). Efficient Planarity Testing. *Journal of the ACM (JACM)*, 21(4), 549–568. doi: 10.1145/321850.321852
- [7] Wikipedia, the free encyclopedia, "FKT algorithm", https://en.wikipedia.org/wiki/FKT_algorithm
- [8] Robertson, N., & Sanders, D., & Seymour, P., & Graham, Communicated. (1996). A New Proof Of The Four-Colour Theorem. *Electron. Res. Announc. Amer. Math. Soc.* 2., 17–25. doi: 10.1090/S1079-6762-96-00003-0
- [9] 高嘉煊. (2019). 浅谈图的点着色问题. IOI2019 中国国家候选队论文集.
- [10] 刘承奥. (2018). MIN&MAX II 题解. <https://loj.ac/d/322>
- [11] Haber, H. E. (2015). Notes on antisymmetric matrices and the pfaffian.
- [12] 李白天. (2020). 斜对称矩阵 Pfaffian 的正确求法. https://blog.csdn.net/EI_Captain/article/details/111016740
- [13] 林昊翰. (2020). Matching 解题报告. IOI2021 集训队第一阶段作业.
- [14] Galbiati, G., & Maffioli, F. (1994). On the computation of pfaffians. *Discrete Applied Mathematics*, 51(3), 269–275. doi: 10.1016/0166-218X(92)00034-J

浅谈求交互参数类交互问题

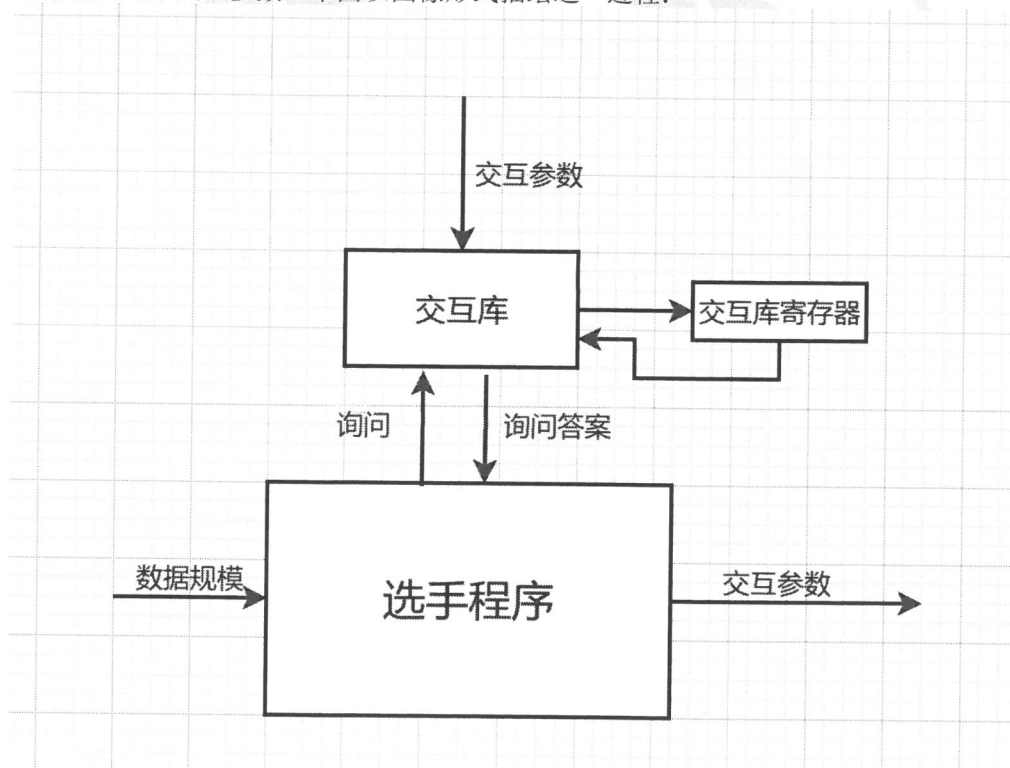
雅礼中学 胡昊

摘要

交互题正在逐渐成为当今 OI 的重要考点，它的考法包括求交互参数、博弈等考法。我的论文将介绍求交互参数类问题的考法，三种解决方案与优化方法。

1 交互参数类问题的基本考法

交互库有一个隐藏的交互参数，你可以通过不断向交互库提出询问来试图得到这个交互参数并输出这个交互参数，下面以图像形式描绘这一过程：



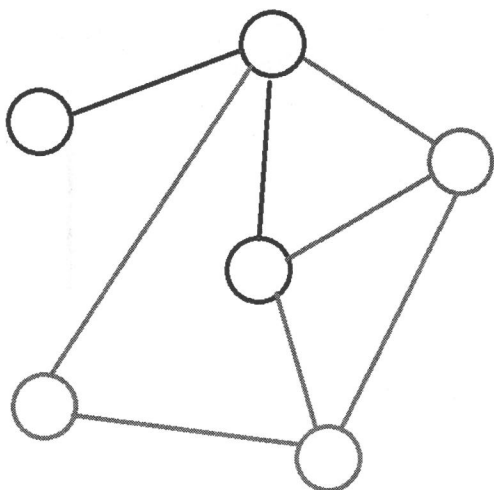
2 逐步增加方便利用的已知信息

2.1 简述

在一道题的开始，可以认为已知信息为空，或者选定一个元素作为初始的已知信息，然后在后面的每一次操作中，目标都是将可以利用的已知信息变多。

在某一道题中，你的目标是求出隐藏的一个序列，题目保证了可以通过前面的若干元素加以询问得到后面的元素，这样我们可以从序列的开头向后依次确定，便于利用的信息就是这个序列的一个前缀；但是如果你知道零散几个位置的值，并不方便利用，这就不是方便利用的信息。

又比如说在另一道题中，你的目标是求出一个隐藏的无向图，我们可以找一个点作为初始连通块，然后不断通过询问扩大，在这里便于利用的信息就是你找到了一个点集的导出子图；但是如果你知道零散几条边，在后面的询问中不好利用，甚至会重复地求出，这就不是方便利用的信息。下面就是在黑色联通块内加入红色的点的实例：



2.2 例题

下面来看几道例题：



2.2.1 例题 1: 组合动作¹

交互库有一个长度为 n 字符集大小为 4 的字符串 S , 你需要通过询问求出并输出这个字符串。每次可以询问一个长度小于 $4n$ 的字符串 T , 会返回最大的 v , 使得 S 的长度为 v 的前缀是 T 的子串。特别的, S 的第一个字符在 S 中仅出现一次。

如果我们知道了 S 的一个前缀 S_p , 那么如果询问 $S_p a$ 返回 True, 那么 $S_p a$ 是 S 的一个更长的前缀。于是, 可以将 S_p 视为我们所说的便于利用的性质。

先通过 2 次询问确定下第一位, 然后用刚才的方法不断地得到 S 的前缀。

题目的性质保证了他便于优化, 但因为适用范围不广, 就在这里简述了: 通过一次询问 $S_p a S_p b a S_p b b S_p b c$ 就可以确定 S 的下一位, 这样就是 $n + O(1)$ 次询问了。

2.2.2 例题 1 改编:

交互库有一个长度为 n 字符集大小为 2 的字符串 S , 你的目标是通过询问求出并输出它。你每次可以询问一个字符串, 交互库会返回它是否是 S 的子串。

如果我们用和上题相同的方法, 可以但也只能得到 S 的一个后缀 (在 S 的后缀后面加 0 或 1 都会返回 False, 可以以此判断是否是 S 的后缀), 我们可以通过在这个字符串前面添加元素, 来得到整个串。

套用上题的优化方法, 但如果询问 $S'0$ 返回 False, 就认为 S' 的下一位是 1 的话, 可能会出现: 此时是序列末尾。每隔 $\sqrt{n}$ 次判断一下现在是否是序列末尾就可以做到 $n + O(\sqrt{n})$ 次询问内求解了。

2.2.3 例题 2: 自然公园²

交互库有一个无向图, 你要求出并输出每一条边。你每次可以询问两个点和一个点集, 会返回这两个点在这个点集的导出子图内是否处于同一连通块。

我们选定任意一个点作为初始点集, 不断找到一个与当前点集相连的点 (如果仅通过当前点集的点能到达点集内任意一点, 就相邻), 并找到连至点集内的所有边, 将这个点加入点集。

¹International Olympiad in Informatics(IOI) 2018

²Japanese Olympiad in Informatics Spring Camp(JOISC) 2017



2.2.4 例题 3: So Mean³

交互库有一个排列, 保证第一个数小于 $\frac{n}{2}$, 你需要求出并输出这一个排列。每次可以询问若干位置, 会返回这些位置上的数的平均值是否是整数。

通过观察发现, 我们询问 $\{l, l+1, l+2, \dots, r\} - \{x\}$ 时, 交互库返回的是 x 是否是 l 到 r 中的最大或最小值, 我们就可以不断的找出未知位置中的最大与最小值。

2.2.5 例题 4: 即时战略⁴

交互库有一棵树, 你需要求出并输出每一条边。每次可以询问 x, y , 交互库会返回 x, y 路径上距离 x 距离为 1 的点。

我们可以挑任意一个点为根, 对于剩下每一个点都可以通过不断的询问找到根到它的路径, 这样就可以还原出树的形态了。

2.3 优化方案

在很多题目中, 最为朴素的算法是不足以通过的, 我们需要找到优化的方案。

要找到优化方案, 要先对先前的算法有更深入的理解, 通过归纳总结, 多数情况下我们使用的算法就是重复的执行下面两步:

1. 找到一个未知的元素。
2. 找到这个元素与已知信息中元素的关系。

就如之前展示的例子, 先找到那个红色的点, 然后找到它和已知的黑点的关系——那些红色的边。在这之后, 红点和黑点间的所有信息就都被我们知道了。通过不断重复就可以求出最终的答案。

很多时候, 这两个步骤是可以优化的, 和优化数据结构题类似, 可以先找到复杂度的瓶颈, 然后针对这一瓶颈套用常用的 $\log$ 或者根号算法: 分治、分块等。

例如, 若瓶颈在于第一步, 可以试着对没有确定的部分进行分治, 将它们根据先前询问得到的特性分成若干部分, 然后分别处理; 如果瓶颈在于第二步; 则可以试着将已经确定的部分划分为若干部分, 依次判断是否与将要加入的元素存在关系, 然后仅保留存在关系的部分。

³codeforces 1299 E

⁴Winter Camp(WC) 2018



2.4 优化实例

下面将展示几个优化的例子：

2.4.1 例题 2 优化 1

这道题中，第一步是找到与已知点集相邻的一个点，第二步是找到这个点已知点集的所有边。课件在这道题中，两个步骤都是复杂度的瓶颈。第一步的优化将在后文中展示，这里先介绍第二步的优化。

不妨假设一个点为根，便于寻找点 u 和已知集合的关系。这里有一个引理：

对于联通点集 S ，点 x, y, z ，满足 $x, y \in S, z \notin S$ ：若满足 $S - \{x\}$ 是联通的，若 z 在仅经过 S 中的点能到达 y ，但在仅经过 $S - \{x\}$ 中的点不能到 y ，则 x, z 有边直接相连。

证明是：若 x 与 z 无直接边相连，考虑 z 到 y 的第一步 $z \rightarrow r$ ，因为 $S - \{x\}$ 联通，所以 r 可以在不经过 x 的情况下到达 y ，与假设矛盾。

利用这一个引理，可以通过二分快速找到一条与 u 相连的边：任意找到一个已知点集的生成树，在它的先序遍历上二分即可，先序遍历的前缀一定组成连通块，与引理的要求相符。

2.4.2 例题 3 优化

在这一道题的解法中，并没有用上元素间的关系，仅仅是找到有集合中最大/最小这个性质的元素，所以可以认为第一步是复杂度的瓶颈，针对这个步骤，可以找到优化：

在第一次询问后，可以将所有元素分为奇数和偶数（询问 1 和这个元素即可），就将数据规模变为了 $\frac{1}{2}$ ；类似的，再根据 $\text{mod} 4$ 分为四个部分，数据规模就变为了 $\frac{1}{4}$ ……

这样就不断让数据规模变为一半，复杂度满足递归式 $T(n) = 2T(\frac{n}{2}) + O(n)$ ，根据主定理有： $T(n) = O(n \log n)$ 。

2.4.3 例题 4 优化

先前算法的瓶颈是：操作次数很多都浪费在了在已知信息中询问。

观察到点分治算法的过程与这道题的询问完全相符，所以可以通过点分治快速到达已知信息组成的树的叶子。这样这个瓶颈的复杂度就被降到 $\log$ 级别了。

3 尝试找到唯一的前驱

接下来我们介绍第二种方法，这一种方法是针对每一个元素的：找到前驱，让每一个点都与它的前驱相连，这样就可以得到一条链。在序列上，一个数的前驱可以认为是它前面

那一个数，在一棵树上，在可以认为是一个点的父亲。

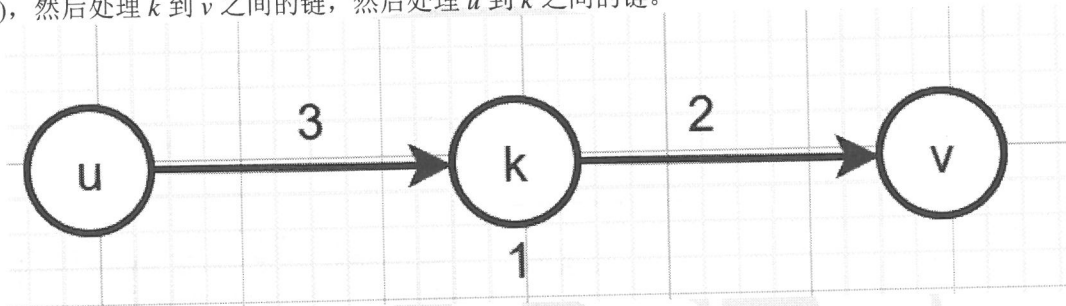
归根到底，这种方法目的在与找到一条链，而之前介绍的方法目的是找出一个连通块。

下面这两种情况可以优先考虑使用这种方法：需要找到一条链，链的末端是已知的但无法由末端回溯；或是答案是一条链，且其它方法不能较好的处理。

有时并不能直接的找到一个点 u 的前驱 $p(u)$ ，但是能找到 $p^k(u)$ ，只要满足以下条件也可以在链长次数内还原出链：

- 如果存在 $p(u)$ ，就一定能找到一个 $p^k(u)$ 。
- 对于任意 $v = p^x(u), x \neq 1$ ，一定能找到 $p^y(u), 1 \leq y < x$ 。

即可以找到任意 u 前面的点，能找到任意 u, v 中间的点，就可以还原出链：先找到 $k = p^x(u)$ ，然后处理 k 到 v 之间的链，然后处理 u 到 k 之间的链。



3.1 例题

3.1.1 例题 2 优化 2

对于任意一个点找到一条到已知连通块的链，然后依次加入这个已知连通块即可。要找到 $p^k(u)$ ，类似优化 1 中的二分即可。

3.1.2 例题 5：整数⁵

交互库有排列 $p: [0, n) \rightarrow [0, n)$ ，并内置了一个整数 I ，你可以询问 $Q(i)$ ，交互库会将 I 加上 2^{p_i} ，并返回 I 中 1 的个数，最终你需要找出交互库的排列。

如果将每一个数都与恰好比它小 1 的数相连，这一道题的目的可以看做是找到一条链。

连续的询问 $Q(i), Q(i)$ 可以知道 I 的 2^{p_i} 位是否为 1，并将 2^{p_i+1} 位变为 1，随机排列 q ，询问 $Q(q_i), Q(q_i)$ ，每次可以以 $\frac{1}{2}$ 的概率去掉一个数 u 错误的前驱 v ，在 $C \log n$ 次即可以将

⁵集训队互测 2021，作者：孙嘉伟

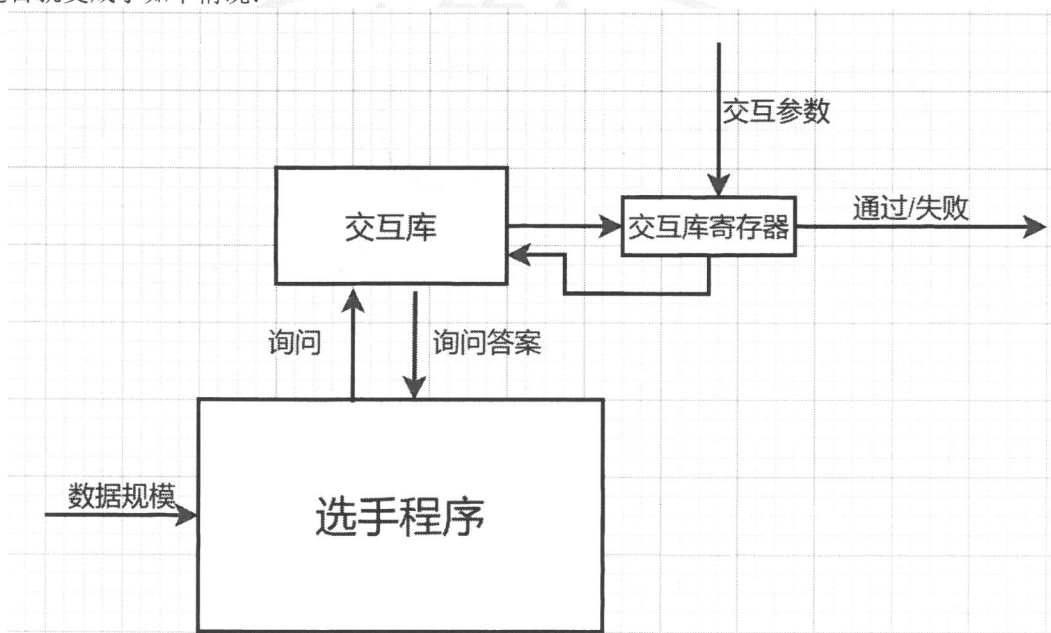
所有数所有错误的前驱去掉，最后只剩下它正确的那一个前驱，依次链接就成了答案对应的链。

最后需要把 I 的前 n 变为 0，加上 2×2^0 即可，寻找 0 的位置可以通过 $Q(0), Q(1), \dots, Q(n-1), Q(0), Q(1), \dots, Q(n-1)$ 做到。

4 巧妙处理交互库寄存器

交互库有时还存在一个寄存器（如例题 5），所以连续两次相同的询问可能会出现不同的答案，这会对分析造成极大的影响。

还有一种情况，交互库寄存器基于交互参数，最终对交互库寄存器的终态有要求（当然往往这个与求交互参数是等效的，因为知道交互参数的情况下这类题将变得特别简单），此时题目就变成了如下情况：



4.1 尽量控制交互库寄存器的信息至理想状态

一开始交互库寄存器的信息往往可以认为是毫无规律的，在经过若干操作时会变得有部分良好的性质，在通过分析返回值得到这些性质时，可以想着怎么去操作使得他们能够保留下来。

更进一步，当寄存器的部分内容是自行选择写入时，对每一个写入元素代表状态明确区分有利于后面操作的分析。

可以说，这一类题目最重要的就是对未知的寄存器的把控。



4.2 例题

4.2.1 例题 6: Indiana Jones and the Uniform Cave⁶

你现在在一个洞穴里。这个洞穴有 n 个房间。房间是不可区分的。每个房间都引出 m 个单向道路，终点可以是自己也可以是其它房间。这些单向道路的入口均匀地分布在房间的墙壁上，且每条单向道路也是不可区分的。保证整个有向图是强连通的。

每个房间有一个石子，这也是你区分房间和道路的唯一工具。一开始石子是在这个房间的某一个通道的入口前，并且是放在中央的。你每到一个房间，可以选择将石子移动到某个通道前，把它放在通道左边或者右边（不能是中间），然后再从某个通道走出去。你不可以把石子带出房间。你一开始在某一个房间，你的目的是便利这个洞穴的所有边。

这道题的寄存器就是石头，一开始石头的摆放是很有规律的：全在中间，可以认为这就是良好的状态，自然的想法就是后续操作也以一定规律摆放石头，这样就可以方便把控寄存器的内容。

因为石头是唯一一个提供的信息，不妨把一个的出边认为是石头所在的边上，并且我们希望从一个点走出后可以回到这一个点上，所以不妨尽可能的让所有已知的节点和他们的出边组成一个内向森林，据此可以对石头的位置进行明确分类：

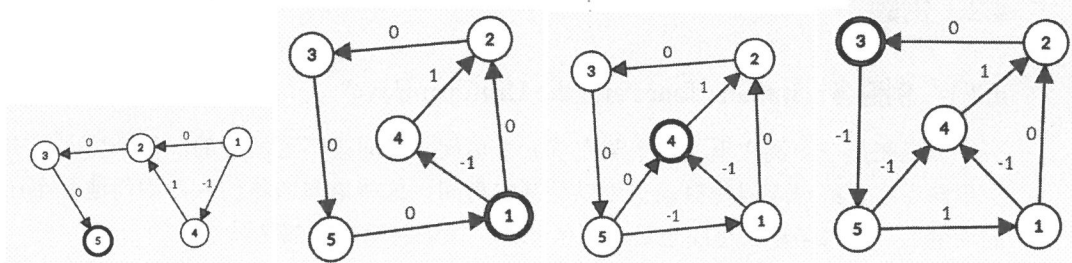
- 中间：第一次经过。
- 左边：确定的边。
- 右边：不能确定到哪的边。

右边会形成一条链，表示这些链上的点还没有找出所有的边。而沿左边最终都会到一个右边的点。

于是，这些石头就可以在已知的信息内指路，当确定一条边后，就继续尝试在顺时针的下一条边行走。当一个点的所有边都被走过，就恢复为左边，并放置将石头在能形成经过右边数量最多的环上。最后走到右链的末端。

下面就是一个实例，加粗表示当前节点，1 表示左边 0 表示右边，-1 表示这条边上不放石头，为了使图示更美观，部分边被省略。

⁶2016-2017 ACM-ICPC Northeastern European Regional Contest (NEERC 16)



4.2.2 例题 7: Rotary Laser Lock⁷

你有一个锁叫做旋转激光锁：

- 锁由编号为 0 到 $n-1$ 的 n 个同心环组成。最里面的环是环 0，最外面的环是环 $n-1$ 。
- 锁平均分成了 $n \times m$ 部分。每一个环都包含一个圆弧，正好覆盖了 m 个相邻的部分（不同的若干个环可以覆盖同一段圆弧）。
- 你每次可以转动一个环一部分的长度，此时交互库会返回锁有多少个部分没有被任何圆环覆盖。

你需要通过若干次转动来得到与输出每一个圆环的当前位置。

对所有圆环依次执行：旋转一周，找到返回的最大值，然后旋转至返回值最大时位置最大处。

这样就一定可以让所有圆环都有另一圆环与它重合。这就是一个非常好的性质。

然后将所有重叠的圆环视作一个，这样就不会破坏那个性质了，重复上述过程，最终在 $\log$ 次内停止。

5 参考文献

英文翻译参考自：

<https://loj.ac/p/2398>

<https://ioihw20.duck-ac.cn/problem/271>

<https://www.luogu.com.cn/problem/CF1428H>

⁷codeforces 1428 H



6 致谢

- 感谢 CCF 提供的交流与学习机会。
- 感谢廖晓刚老师对我的支持与指导。
- 感谢毛啸学长为本文提出的修改意见。
- 感谢褚轩宇、朱添翼同学为本文提供部分例题。
- 感谢集训队的队员们提供的高质量互测题。
- 感谢其它曾给予我帮助的老师同学们。





浅谈 OI 中的近似算法

中国人民大学附属中学 许庭强

摘要

在 OI 中的很多题目里,选手只需要输出与答案相对误差不超过 ε 的结果即可通过,其中 ε 有 $10^{-9}, 10^{-4}, 10^{-2}$ 等取值。本文旨在介绍一些近似算法,其给出的答案与精确值的相对误差或绝对误差不超过给定的范围。利用近似算法,一类返回实数的求值类问题可以被很好地解决;一部分 NP-hard 问题,也能在较大甚至任意的 ε 误差限制条件下,在多项式复杂度内得到解决;另外,一些多项式复杂度的问题同样可以在允许很小的误差之后获得更小的多项式复杂度做法。

1 前言

在 OI 中,很多答案为实数的求值类问题以 10^{-9} 的相对或绝对误差为限制,来避免 C++ 中 double 的精度误差。但事实上,有些题目可以通过泰勒展开、数值积分等方法近似答案,以部分时间为代价换取足够的精度,来达到直接求解难以企及的复杂度,而这种技巧尚未得到普及。本文将在第二章中介绍这些算法。

另外,对于一些 NP 问题,多年以来,学术界中致力于获得相对更优的解;而本文引入的一些算法可以很好地解决部分此类问题。其他随机算法,如调整算法,也能求出相对较优的解,但解的优秀程度不能对所有数据都有所保证;相应地,其正确性也无法得到证明。然而,本文给出的算法可以保证对于所有数据都能得到相对较优的答案。本文将在第三章中展示这些算法。

本文的第四章描述了一些图上的近似算法。这些算法所解决的问题本身可以用其他多项式复杂度算法精确求解;但近似算法可以以一定误差为代价,极大地降低求解的复杂度。

2 精度近似

2.1 数值积分

在很多题目中,所求答案可以经过某些过程转化为求一次数值积分。即存在一个可积函数 $f(x)$ 与区间 $[A, B]$, 题目需要求出以下积分式的值:

$$\int_A^B f(x)dx$$

大多数题目中的 $f(x)$ 为线段长度或图形面积等信息，所以我们在这里默认 $f(x)$ 连续，在除有限个位置外任意阶可导，且在 $[A, B]$ 内大于等于 0。有时 f 会比较复杂，难以直接求出其积分值，这时就需要近似求解。本文给出以下方法来近似求出对应的积分值。

2.1.1 黎曼和近似

根据定积分的定义，取一正整数 N ，将区间等分成 N 份。对于每个区间，用区间中点处对应的 f 的取值来近似整个区间 f 的平均值，并以此来近似出整个区间的积分值。形式化地说，对固定正整数 N 求出：

$$I_N = \sum_{i=0}^{N-1} \frac{B-A}{N} f\left(A + \frac{i(B-A)}{N} + \frac{1}{2N}\right)$$

根据定积分定义，在 N 趋于无穷大时，求得的值无限接近于真实积分值。相应的，算法要支付 $O(N)$ 次求值的代价。这时估计每个 N 对应的误差就变得更加重要。以下为 f 在若干常见函数下时的情况：

1. 一次函数： $f(x) = ax + b$ ，没有误差。
2. 二次函数： $f(x) = ax^2$ ，相对误差为

$$\frac{|I_N - \frac{1}{3}a(B^3 - A^3)|}{\frac{1}{3}a(B^3 - A^3)} = \frac{(B-A)^2}{4(B^2 + AB + A^2)} \cdot \frac{1}{N^2} = O\left(\frac{1}{N^2}\right)$$

3. 三角函数： $f(x) = \cos x$ ，相对误差为

$$\frac{|I_N - (\sin B - \sin A)|}{\sin B - \sin A} = \frac{\frac{B-A}{N} - 2 \sin \frac{B-A}{2N}}{2 \sin \frac{B-A}{2N}} = O\left(\frac{1}{N^2}\right)$$

另外，我们有以下定理：

定理 2.1. 如果 f 在 $[A, B]$ 内任意阶可导，黎曼和近似的绝对误差在最坏情况下不超过 $\frac{K(B-A)^3}{24N^2}$ ，其中 K 为 $|f^{(2)}(x)|$ 在 $[A, B]$ 中的最大值。

证明在这里不进行展开，具体可见 [4]。

下面给出一道用黎曼和近似积分的例题：

例题 1 (差不多得了¹)。给定 n 个二维平面上的整点，对每个点求到所有其他点的距离之和，允许相对误差范围 $\varepsilon = 10^{-4}$ ，保证 $n \leq 2 \times 10^5$ ，坐标绝对值不超过 10^9 。

¹来源：邓明扬命制的模拟赛题目

解法 将两个点 $(x_1, y_1), (x_2, y_2)$ 之间的距离 $l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ 看做

$$\frac{1}{4} \int_0^{2\pi} |l \cos x| dx$$

即为对所有角度的线段投影长度求积分。这时就可以交换求和顺序，取 N 个均匀角度 $\alpha_1, \alpha_2, \dots, \alpha_N$ 。考虑某一条线段，对每个 $1 \leq i \leq N$ ，取出该线段投影到 $y = \tan \alpha_i \cdot x$ 这条直线上后的长度，求和算出的值即为上述积分式的黎曼和近似。这样，就可以对每个 $1 \leq i \leq N$ ，将所有点投影到直线上，算出和每个点相连的线段投影长度之和。这是一维的问题，排序后容易做到线性。

根据上文的分析，可以知道在所有不包含 $|\cos x|$ 的不可导点的区间内，误差为 $O(\frac{1}{\sqrt{N}})$ 。对于 $\cos x = 0$ 附近的位置，由于有 $\sin x < x$ ，周围 $\frac{B-A}{N}$ 长度的区间中积分为 $O(\frac{1}{\sqrt{N}})$ 量级。同时取出的中点值域也在 $[0, \frac{B-A}{N}]$ 之间，所以这一段中近似出的值也为 $O(\frac{1}{\sqrt{N}})$ 量级。因此，造成的绝对误差不超过 $O(\frac{1}{\sqrt{N}})$ 量级，从而对总体相对误差的影响不超过 $O(\frac{1}{\sqrt{N}})$ 。

取 $N = \Theta(\varepsilon^{-\frac{1}{2}})$ 即能获得满足题目要求的精度。

时间复杂度为 $O(n\varepsilon^{-\frac{1}{2}} \log n)$ 。

由于大部分函数误差关于 N 均为多项式量级，此方法需要在 ε^{-1} 值较小时才会有比较可观的实现效率。

2.1.2 辛普森积分

我们尝试用二次函数来近似被积函数 $f(x)$ 。当 $f(x) = ax^2 + bx + c$ 时，有

$$\begin{aligned} \int_A^B f(x) dx &= \frac{1}{3}a(B^3 - A^3) + \frac{1}{2}b(B^2 - A^2) + c(B - A) \\ &= \frac{B-A}{6}(2a(A^2 + AB + B^2) + 3b(A+B) + 6c) \\ &= \frac{B-A}{6} \left(f(A) + 4f\left(\frac{A+B}{2}\right) + f(B) \right) \end{aligned}$$

这样，辛普森积分用 $(A, f(A)), (\frac{A+B}{2}, f(\frac{A+B}{2})), (B, f(B))$ 三个点决定的二次函数来估计 $f(x)$ 在 $[A, B]$ 上的积分值，如下：

定义 2.1. 函数 $f(x)$ 在区间 $[A, B]$ 中的辛普森积分为

$$\frac{B-A}{6} \left(f(A) + 4f\left(\frac{A+B}{2}\right) + f(B) \right)$$

辛普森积分的近似效果比中点更好，具体来说：

定理 2.2. 将区间 $[A, B]$ N 等分，用辛普森积分近似每一段区间，如果 f 在 $[A, B]$ 内均任意阶可导，其近似绝对误差不超过 $\frac{M(B-A)^5}{2880N^4}$ ，其中 M 为 $|f^{(4)}(x)|$ 在 $[A, B]$ 中的最大值。

证明在这里不进行展开，具体可见 [4]。

2.1.3 自适应辛普森积分

在 $f(x)$ 存在与二次函数相差过多的区间（如不可导点）时，在其附近近似产生的误差会极大增加，这导致我们需要进一步缩小辛普森积分的区间长度。但在所有位置上均缩小区间长度的代价太大，这时我们可以用自适应辛普森积分解决这一问题。在自适应辛普森中，如果可以大致判断辛普森积分误差很小，那么直接返回辛普森积分的值，否则递归到两个区间中继续运算。伪代码如下所示：

Algorithm 1 ADAPTIVESIMPSONINTEGRAL

Require: An interval $[A, B]$, a small number ε , a function f which will return $f(x)$ and a function ϕ which will return the simpson integral from l to r when calling $\phi(l, r)$

```

1: Let  $mid = \frac{A+B}{2}$ 
2: if  $|\phi(A, mid) + \phi(mid, B) - \phi(A, B)| < \varepsilon$  then
3:   return  $\phi(A, B)$ 
4: else
5:   Let  $left = \text{AdaptiveSimpsonIntegral}(A, mid)$ 
6:   Let  $right = \text{AdaptiveSimpsonIntegral}(mid, B)$ 
7:   return  $left + right$ 
8: end if
```

自适应辛普森算法能解决很多 OI 中的计算几何问题，并有着不错的表现。

2.2 泰勒展开

对于一个实解析的函数 f ，我们观察 $f(x)$ 在 x_0 展开的泰勒级数：

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + \cdots$$

其中 ξ 为 x 与 x_0 之间的某个位置。

引理 2.1. 如果 $\exists M > 0, \forall n \geq 0, \left| \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \right| < M \cdot 2^{-n}$ ，那么只需计算泰勒展开的前 $\Theta(\log \varepsilon^{-1})$ 项，即可使绝对误差不超过 ε 。

证明. 由于函数实解析，泰勒级数一定收敛到 $f(x)$ 。考虑 $N_0 = \lceil \log \varepsilon^{-1} + \log M \rceil + 1$ 之后的项，其绝对值总和 $< M \frac{\varepsilon}{M} = \varepsilon$ 。所以计算前 $N_0 = \Theta(\log \varepsilon^{-1})$ 项即可。□

容易得到以下直接推论：

引理 2.2. 如果 $\exists M > 0, \forall n \geq 0, \left| \frac{f^{(n)}(x_0)}{n!} \right| < M$ 且 $|x - x_0| < \frac{1}{2}$ ，则只需计算泰勒展开的前 $\Theta(\log \varepsilon^{-1})$ 项，即可使绝对误差不超过 ε 。

由于大部分常见函数均满足导数有界, 且该引理使用起来相对方便, 我们下面会经常使用该引理。将函数用泰勒展开的好处是我们把一个相对复杂的函数变成了一个简单的多项式, 而多项式的性质是很好的。以下几道例题展示了这种方法的实用性。

例题 2 (获取名额²)。给定正整数序列 $a_1, a_2, \dots, a_n$, q 次查询 $\prod_{i=l}^r (1 - \frac{a_i}{x})$, 允许绝对误差范围 $\varepsilon = 10^{-6}$, 保证 $n, q \leq 6 \times 10^5, a_i < x \leq 10^9$ 。

解法 注意到如果在区间 $[l, r]$ 内有至少 $\log \varepsilon^{-1}$ 个 a_i 满足 $\frac{a_i}{x} \geq \frac{1}{2}$, 那么所求一定不超过 ε , 输出 0 即可。可以暴力求出区间大于等于 $\frac{1}{2}$ 这部分的贡献。那么不妨只考虑所有 $\frac{a_i}{x} < \frac{1}{2}$ 的 a_i 。

将所求式子改写为

$$\exp\left(\sum_{i=l}^r \ln\left(1 - \frac{a_i}{x}\right)\right)$$

将 $\ln(1-x)$ 在 $x=0$ 处展开, 得到泰勒级数

$$\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \dots$$

由于 $|x| < \frac{1}{2}$, 由引理, 只需展开到 $\Theta(\log \varepsilon^{-1})$ 项即可。那么每个 $\ln(1 - \frac{a_i}{x})$ 均可近似成一个关于 $\frac{a_i}{x}$ 的不超过 $\Theta(\log \varepsilon^{-1})$ 次的多项式。所以此时区间 $[l, r]$ 的多项式就可以按次数合并了。求答案时可简单预处理出所有 $\frac{a_i^k}{k}$ 的前缀和, 最后分别求出区间和再乘上 x^{-k} 叠加即可。时间复杂度 $O(n \log^2 \varepsilon^{-1} + n \log n)$ 。

此题对 $\geq \frac{1}{2}$ 的部分做了特殊处理, 对 $< \frac{1}{2}$ 的部分使用泰勒展开优化, 做到了直接求值难以做到的复杂度。但在省掉一个 n 的同时, 付出了 $\log^2 \varepsilon^{-1}$ 的额外代价。

例题 3 (Game of Chance³)。给定正整数序列 $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n$, 对每个 j 求 $\sum_{i=1}^n \frac{b_j}{b_j + a_i}$, 允许误差范围 10^{-9} , 保证 $n \leq 3 \times 10^5, a, b \leq 10^9$ (经过部分转化)。

解法 设函数 $f(x) = \frac{1}{1+x}$, 查询即为求所有 $\sum_{i=1}^n f(\frac{a_i}{b_j})$ 的值。将 $f(x)$ 在 $x_0 = \frac{1}{2}$ 处展开, 若 $\frac{a_i}{b_j} \leq 1$, 则可用以下展开式的前 $\Theta(\log \varepsilon^{-1})$ 项来估计 $f(\frac{a_i}{b_j})$ 的值。

$$\frac{1}{1+x} \approx \sum_{n=0}^{\infty} (-1)^n \left(\frac{2}{3}\right)^{n+1} \left(x - \frac{1}{2}\right)^n$$

截取前 $\Theta(\log \varepsilon^{-1})$ 项, 将 $(x - \frac{1}{2})^n$ 用二项式定理展开, 可以得到一个关于 x , 次数不超过 $\Theta(\log \varepsilon^{-1})$ 的多项式。我们要一次性对所有满足 $\frac{a_i}{b_j} \leq 1$ 的 a_i 求和。把 $\frac{1}{b_j}$ 次数相同的项合

²来源: <http://zijian-lv.com/problem/34>

³来源: <https://codeforces.com/problemset/problem/1578/G>

并；只需要求出所有 $a_i \leq b_j$ 中 $a_i^k (k \leq \Theta(\log \varepsilon^{-1}))$ 之和即可。用 $O(n \log \varepsilon^{-1})$ 的复杂度预处理出所有 a_i^k ，求答案时求出每项系数乘上对应的 b_j^k ，就能求出答案。

我们至此解决了 $a_i \leq b_j$ 的这部分，但另一部分还仍未解决。在 $\frac{a_i}{b_j}$ 远大于 1 时，泰勒展开出的式子并不收敛。所以我们需要另辟蹊径。

我们换一种角度看 $\frac{b_j}{b_j+a_i}$ 。 $\frac{b_j}{b_j+a_i}$ 可以写成

$$\frac{\frac{b_j}{a_i}}{1 + \frac{b_j}{a_i}}$$

此时注意到有 $0 \leq \frac{b_j}{a_i} \leq 1$ ，这启示我们对 $\frac{x}{1+x}$ 在 $x_0 = \frac{1}{2}$ 处展开。将 $\frac{x}{1+x}$ 在 $x_0 = \frac{1}{2}$ 展开得到：

$$\frac{x}{1+x} \approx \frac{1}{3} + \sum_{n=1}^{\infty} (-1)^{n+1} \left(\frac{2}{3}\right)^{n+1} \left(x - \frac{1}{2}\right)^n$$

与刚才类似，我们截取前 $\Theta(\log \varepsilon^{-1})$ 项后二项式展开得到一个多项式。需要对 $a_i > b_j$ 的所有 a_i 在 $\frac{b_j}{a_i}$ 位置求和。预处理出所有的 a_i^k ，求出每项系数乘上对应的 b_j^k 即可。在刚开始对 a 和 b 排序需要 $O(n \log n)$ 。总时间复杂度 $O(n \log n + n \log \varepsilon^{-1})$ 。

此题以 a_i 和 b_j 的大小关系分了两部分，分别用不同的函数泰勒展开求解。这是为了解决泰勒级数的收敛问题，本质是将 $f(x)$ 分成了 $0 \leq x \leq 1$ 与 $x > 1$ 两部分。对于 $0 \leq x \leq 1$ 的部分可以直接用 $f(x)$ 在 $x_0 = \frac{1}{2}$ 处展开的多项式求解。对于 $x > 1$ 的部分用 $f\left(\frac{1}{x}\right)$ 在 $x_0 = \frac{1}{2}$ 处展开的多项式求解。这能解决绝大部分对 $f(x)$ 求和的求值类问题，只需要额外花费 $O(\log \varepsilon^{-1})$ 的代价（有时为分出两部分需要加上 $O(\log n)$ 的代价）。

3 部分 NP 问题的近似解

有很多 NP 问题均有近似算法，如背包问题、最大独立集问题等。但有些 NP 问题也被证明不存在有效的近似算法，如 2-SAT 解计数等。我们下面给出部分可以近似求解的 NP 问题。

以下用 $(1 + \varepsilon)$ -近似来表示相对误差为 ε 的近似。

3.1 背包问题

考虑一个经典的 01-背包问题：

例题 4. 有 n 个物品，每个物品有体积 w_i ，权值 v_i 。有一个背包容量为 W ，至多装体积总和不超过 W 的物品，求装的物品权值和最大值。保证 $w_i \leq W$ 。

众所周知, 这是一个 NP 问题, 所以我们在关于 n 的多项式复杂度内解决该问题。但我们有对任意 $\varepsilon > 0$ 的多项式 $(1 + \varepsilon)$ -近似算法。

解法 设所有物品的权值最大值为 V 。将每个物品的权值改为 $v'_i = \lfloor \frac{w_i}{\varepsilon V} \rfloor$ 。可以用一次简单 dp 求出 f_j 的值, 其中 f_j 表示选取物品 v'_i 总和为 j 时重量总和的最小值。由于所有物品权值和不超过 $n^2 \varepsilon^{-1}$, 该 dp 时间复杂度为 $O(n^3 \varepsilon^{-1})$ 。我们证明如下引理:

引理 3.1. 取出 $f_j \leq W$ 的最大 j , 其对应的选取方案与最优解相对误差不超过 ε 。

证明. 设实际上的最优解取出的物品集合为 S_0 , dp 出的最优方案取出的物品集合为 S_1 。首先有 $\sum_{x \in S_0} w_x, \sum_{y \in S_1} w_y \leq W$ 。并且满足如下两个条件:

$$\begin{aligned} \sum_{y \in S_1} v_y &\leq \sum_{x \in S_0} v_x \\ \sum_{y \in S_1} \left\lfloor \frac{nv_y}{\varepsilon V} \right\rfloor &\geq \sum_{x \in S_0} \left\lfloor \frac{nv_x}{\varepsilon V} \right\rfloor \end{aligned}$$

其中第一个条件由 S_0 的最优性给出, 第二个条件由 dp 出 S_1 的最优性给出。可以由第二个条件推出以下不等式:

$$\begin{aligned} \sum_{y \in S_1} \frac{nv_y}{\varepsilon V} &\geq \sum_{y \in S_1} \left\lfloor \frac{nv_y}{\varepsilon V} \right\rfloor \geq \sum_{x \in S_0} \left\lfloor \frac{nv_x}{\varepsilon V} \right\rfloor > \sum_{x \in S_0} \left(\frac{nv_x}{\varepsilon V} - 1 \right) \\ \sum_{x \in S_0} v_x - \sum_{y \in S_1} v_y &< \frac{\varepsilon V}{n} \sum_{x \in S_0} 1 \leq \varepsilon V \end{aligned}$$

由于 $\sum_{x \in S_0} v_x \geq V$, 所以相对误差满足

$$\frac{\sum_{x \in S_0} v_x - \sum_{y \in S_1} v_y}{\sum_{x \in S_0} v_x} < \varepsilon$$

证毕。 □

这样我们就给出了一个 $O(n^3 \varepsilon^{-1})$ 时间复杂度的 $(1 + \varepsilon)$ -近似算法。 ■

实际上, 关于 01 背包的 $(1 + \varepsilon)$ -近似问题, 目前的最优算法的复杂度远远低于 $O(n^3 \varepsilon^{-1})$ [3]。



3.2 随机采样

在一些 NP 问题中, 我们经常可以用随机采样的方式来估计某个值的大小。以下基本模型展示了随机采样方法的优秀程度。

定理 3.1. 有一个随机数生成器, 返回的数字在 $[A, B]$ 之间, 期望为 E_0 , 方差为 σ^2 。那么随机 N 次, 求出返回数的平均值, 与 E_0 的绝对误差期望为 $O(\sigma N^{-\frac{1}{2}})$ 。

证明. 设随机 N 次返回的结果为 $a_1, a_2, \dots, a_N$, 平均值即为 $\frac{a_1 + a_2 + \dots + a_N}{N}$ 。

先求出 $E\left(\left(\frac{a_1 + a_2 + \dots + a_N}{N} - E_0\right)^2\right)$ 的值。

$$\begin{aligned} E\left(\left(\frac{a_1 + a_2 + \dots + a_N}{N} - E_0\right)^2\right) &= \frac{1}{N^2} E\left(\left(\sum_{i=1}^N (a_i - E_0)\right)^2\right) \\ &= \frac{1}{N^2} \left(\sum_{i=1}^N E((a_i - E_0)^2) + \sum_{1 \leq i < j \leq N} 2E(a_i - E_0)E(a_j - E_0) \right) \\ &= \frac{1}{N^2} \cdot N \cdot \sigma^2 = \frac{\sigma^2}{N} \end{aligned}$$

所以绝对误差期望为 $O(\sigma N^{-\frac{1}{2}})$ 。

□

进一步的, 在算法竞赛中, 不仅需要误差期望不大, 还需要让误差较小的概率非常大, 所以我们展示如下定理:

定理 3.2. 有一个随机数生成器, 返回的数字在 $[A, B]$ 之间, 期望为 E_0 , 方差为 σ^2 。那么随机 N 次, 求出返回数的平均值, 设与 E_0 的绝对误差为 δ , 那么几乎一定有 $\delta < O(\sigma N^{-\frac{1}{2}})$ 。

证明. 设绝对误差为 $\delta = \left| \frac{a_1 + a_2 + \dots + a_N}{N} - E_0 \right|$ 。考虑对任意常数 c , 求出

$$\Pr(\delta > c \cdot \sigma N^{-\frac{1}{2}})$$

由 Hoeffding's inequality [7],

$$\begin{aligned} &\Pr(\delta N > c \cdot \sigma N^{\frac{1}{2}}) \\ &\leq 2 \exp\left(-\frac{2(c-1)^2 \sigma^2 N}{N(B-A)^2}\right) \\ &= 2 \exp\left(-2(c-1)^2 \cdot \frac{\sigma^2}{(B-A)^2}\right) \end{aligned}$$

是一个与 N 无关的常数且随 c 增大指数级缩小。证毕。

□

该定理告诉我们,如果要求出 E_0 的近似值,满足绝对误差不超过 ε ,则只需让 $N = \Theta(\sigma^2 \varepsilon^{-2})$ 即可。

进一步的,为使用方便,给出以下直接推论:

引理 3.2. 有一个随机数生成器,返回的数字在 $[A, B]$ 之间,期望为 E_0 。那么随机 N 次,求出返回数的平均值,设与 E_0 的绝对误差为 δ ,那么几乎一定有 $\delta < O((B-A)N^{-\frac{1}{2}})$ 。

证明. 由于 $\sigma \leq B-A$, 该式显然成立。 □

例题 5 (Number of Close Strings⁴). 有 k 个长度为 n 的 01 串, 和一个正整数 m 。求有多少个长度为 n 的 01 串, 满足和至少一个给出的 01 串最多 m 位不同。满足相对误差在 $\varepsilon = 10^{-2}$ 内即可。保证 $1 \leq n, k \leq 50, 1 \leq m \leq n$ 。

解法

设 S_i 为和第 i 个串至多 m 位不同的 01 串集合, 设 $S = S_1 \cup \dots \cup S_k$, 所求即为 $|S|$ 。显然有 $|S_1| = |S_2| = \dots = |S_k|$, 设为 M 。

设 $S = \{a_1, a_2, \dots, a_{|S|}\}$, b_i 为 a_i 在 $S_1, \dots, S_k$ 中的出现次数。设 $S_r = \{a_{r_1}, a_{r_2}, \dots, a_{r_M}\} (1 \leq r \leq k)$ 。构造一个序列 $x_1, x_2, \dots, x_{kM}$, 其中 $x_{(r-1)M+s} = \frac{1}{b_{r_s}} (1 \leq r \leq k, 1 \leq s \leq M)$ 。

显然有 $\sum_{i=1}^{kM} x_i = |S|$, 即要求出的值。但 kM 过大, 无法枚举计算。不过由于 kM 已知, 我们只需估计出 $E(x_i)$ 即可。由于等概率随机其中一个 x_i 是容易的, 考虑随机采样 N 次。由引理 3.2, 绝对误差以极大概率不超过 $O(N^{-\frac{1}{2}})$ 。因所求平均值在 $[\frac{1}{k}, 1]$ 中, 相对误差不超过 $O(kN^{-\frac{1}{2}})$ 。取 $N = \Theta(k^2 \varepsilon^{-2})$ 即可。

加上计算 x_i 与随机的代价, 时间复杂度为 $O(k^3 \varepsilon^{-2})$ 。 ■

4 若干图上问题的近似算法

近似算法应用范围广泛, 在一些能用多项式复杂度解决的问题中, 如果允许部分相对或绝对误差, 也能做到更优秀的复杂度。本文给出一些图上问题的近似算法。

例题 6 (想法⁵)。有一张有向无环图 $G = (V, E)$, 求每个点能到达的节点个数, 允许相对误差范围 $\varepsilon = \frac{1}{4}$ 。保证 $|V| \leq 10^5, |E| \leq 10^6$ 。

解法

为了解决这一题目, 我们先介绍如下引理:

⁴来源: <https://codeforces.com/gym/101385/problem/A>

⁵来源: <https://loj.ac/p/2229>

引理 4.1. k 个 $[0, 1]$ 中的等概率随机变量, 其最小值期望为 $\frac{1}{k+1}$ 。

证明. 期望为

$$\int_0^1 (1-x)^k dx = \frac{1}{k+1}$$

□

所以我们每次给每个点赋一个 $[0, 1]$ 之间的随机权值, 求出所有点能到的点中权值最小值。设第 i 个点的答案是 ans_i 。采样 N 次之后, 对每个点求出这些最小值的平均值, 因为我们期望其是 $\frac{1}{\text{ans}_i+1}$, 所以我们认为 ans_i 约为该平均值取倒数后减一的值。我们来分析这么做的误差。

引理 4.2. 相对误差几乎一定不超过 $O(N^{-\frac{1}{2}})$ 。

证明. 设最小值的平均值为 M , $k = \text{ans}_i$, 先分析 M 与 $\frac{1}{k+1}$ 的绝对误差。此时仅仅使用引理 3.2 是不够的, 我们需要求出定理 3.2 中用到的方差的值。

$$\begin{aligned} \sigma^2 &= E\left(\left(M - \frac{1}{k+1}\right)^2\right) \\ &= E\left(M^2 - \frac{2M}{k+1} + \frac{1}{(k+1)^2}\right) \\ &= \int_0^1 (1-\sqrt{x})^k dx - \frac{1}{(k+1)^2} \\ &= \frac{k}{(k+1)^2(k+2)} = \Theta(k^{-2}) \end{aligned}$$

所以 $\sigma = \Theta(k^{-1})$ 。由定理 3.2, 该绝对误差几乎一定不超过 $O(k^{-1}N^{-\frac{1}{2}})$ 。那么, 设 $M = \frac{1}{k+1} + \delta$, 几乎一定有 $|\delta| < O(k^{-1}N^{-\frac{1}{2}})$ 。

接下来求 $|\frac{1}{M} - k - 1|$ 的值。

$$\begin{aligned} \left|\frac{1}{M} - k - 1\right| &= \frac{(k+1)^2|\delta|}{1 + (k+1)\delta} \\ &= O(kN^{-\frac{1}{2}}) \end{aligned}$$

所以相对误差几乎一定不超过 $O(N^{-\frac{1}{2}})$ 。

□

取 $N = \varepsilon^{-2}$ 即可。时间复杂度 $O((|V| + |E|)\varepsilon^{-2})$ 。

■

定义 4.1. 一个问题精确答案为正整数 x , 我们称只需求出一个满足 $x \leq y \leq x+t$ 的 y 的问题为该问题的 surplus- t 近似。

例题 7. 给定一张无向无权简单图 $G = (V, E)$, 保证存在环, 以 surplus-2 近似求图上最小简单环的长度。保证 $|V| \leq 8000$ 。

解法

固定一个点 v , 从 v 开始 bfs, 找到到每个点的最短距离并求出最短路是否唯一。维护一个正整数 t , 初始为 n 。

1. 如果两个相邻点到 x 最短路长度相同, 设均为 d , 令 $t \leftarrow \min(t, 2d+1)$ 。
2. 如果某个点到 x 最短路不唯一, 设最短路长度为 d , 令 $t \leftarrow \min(t, 2d)$ 。

设最小环长度为 q , 我们证明如下引理:

引理 4.3. 最终 t 的值不会小于图中最小环长度, 且如果最小环经过 v 或 v 的邻域, 则 $t \leq q+2$ 。

证明. 如果一个环经过的所有边构成一棵树, 我们称其为退化环, 否则为非退化环。显然如果存在一个长度为 k 的非退化环, 那么一定存在一个长度不超过 k 的简单环。我们先证明命题的前一半。

1. 对第一种情况, 设两点为 a, b , 则 $x \rightarrow a \rightarrow b \rightarrow x$ 为一个长度为 $2d+1$ 的环。由于该环为非退化环, 一定存在一个简单环长度不超过 $2d+1$ 。
2. 对第二种情况, 设点为 a , 则 x 先用第一种最短路到 a 再用第二种最短路回到 x 为一个长度为 $2d$ 的环。由于该环为非退化环, 一定存在一个简单环长度不超过 $2d$ 。

这样我们就证明了最终 t 的值不会小于图中最小环长度。再证明命题的后一半。

如果最小环经过 v , 那么考虑环在与 v 对称的 1 或 2 个点, 分别能够对应前文中的第 2 或 1 种情况, 这时 $t \leq q$ 。

如果最小环经过 v 的邻域, 设经过了点 u , 与 v 相邻, 那么分别考虑不同环长奇偶性的情况。如果环长为偶数, 考虑最小环中 u 对面的点, 设为 a 。那么会在第二种情况中将 a 识别为最短路不唯一, 由于 v 到 x 的距离不超过 $\frac{q}{2}+1$, 有 $t \leq q+2$ 。

如果环长为奇数, 考虑最小环中 u 对面的两个点, 设为 a, b 。设到 v 的最短路分别为 d_a 和 d_b 。考虑 $v \rightarrow a \rightarrow u \rightarrow v$ 这个环。

1. 如果该环为退化环, 那么一定 v 到 a 的一条最短路经过 u , 所以有 $d_a = \frac{q+1}{2}$ 。
2. 如果该环为非退化环, 那么由于最小环长度为 q , 有 $d_a+1+\frac{q-1}{2} \geq q$, 即 $d_a \geq \frac{q-1}{2}$ 。同样因为 $v \rightarrow u \rightarrow a$ 为一条 v 到 a 的路径, 所以 $d_a \leq \frac{q+1}{2}$ 。

所以 $\frac{q-1}{2} \leq d_a \leq \frac{q+1}{2}$, 同理 $\frac{q-1}{2} \leq d_b \leq \frac{q+1}{2}$ 。再分两种情况讨论 d_a 与 d_b 的关系。

1. 如果 $d_a = d_b$, 那么 a, b 会在第一种情况中被考虑到, 所以 $t \leq d_a + d_b + 1 \leq q + 2$ 。
2. 如果 $d_a \neq d_b$, 不妨设 $d_a = \frac{q-1}{2}, d_b = \frac{q+1}{2}$, 点 b 会在第二种情况中被考虑到, 所以 $t \leq 2d_b \leq q + 1$ 。

这样我们就完成了对 $t \leq q + 2$ 的证明。

□

这样如果最小环经过 v 或其邻域, 输出 t 即可。否则我们就可以将这个点及其邻域删去, 再找一个新的 v , 反复 bfs, 对所有 bfs 得到的 t 取最小值, 输出即可。因为每个 t 始终不会小于真实答案, 并且在将所有点删去的过程中, 一定有一次删除的 v 在最小环上或最小环周围。我们断言如下引理成立。

引理 4.4. 如果每次 v 为当前度数最大的点, 时间复杂度为 $O(|V|^2)$ 。

证明. 每次 bfs 时间复杂度为 $O(|V| + |E|)$, 但因 v 为度数最大的点, v 的度数不低于 $\frac{2|E|}{|V|}$, 所以删除的点数不少于 $\frac{2|E|}{|V|}$ 。设每一次删除时边数分别为 $m_1, m_2, \dots, m_s$, 由于最多删除 $|V|$ 个点, 有

$$\sum_{i=1}^s \frac{2m_i}{|V|} \leq |V|$$

所以时间复杂度不超过

$$O(|V|^2 + \sum_{i=1}^s m_i) = O(|V|^2)$$

□

这样我们就以 $O(|V|^2)$ 的时间复杂度内求出了图中 surplus-2 的最小环长度。

■

例题 8.⁶ 给定一张无向无权简单图 $G = (V, E)$, 保证存在环, 以 surplus-1 近似求图上最小简单环的长度。保证 $|V| \leq 8000$ 。

解法

观察上一个题目的做法, 注意到如果最小环经过 v , 那么求出的答案是准确的。进一步的, 如果是 surplus-1 近似, 我们只需找到最小的 d , 使得存在某点或某两个相邻点满足条件。这样答案一定为 $2d$ 或 $2d + 1$ 。

但此时一次 bfs 需要 $O(m)$ 的代价, 不可接受, 需要进一步优化。我们给出如下算法:

⁶来源: 王泽远命制的模拟赛题目

Algorithm 2 ALMOSTMINIMUMCIRCUIT

Require: A graph $G = (V, E)$ and a given vertex v .

```

1:  $L_0 \leftarrow v, t \leftarrow n$ 
2:  $vis[v] \leftarrow \text{true}$ 
3: for  $u \neq v, u \in V$  do
4:    $vis[u] \leftarrow \text{false}$ 
5: end for
6: for  $i \geq 0$  do
7:    $L_{i+1} \leftarrow \emptyset$ 
8:   for  $u \in L_i$  do
9:     for  $(u, x) \in E$  do
10:       $E \leftarrow E \setminus \{(u, x)\}$ 
11:      if  $vis[x] = \text{false}$  then
12:         $vis[x] \leftarrow \text{true}$ 
13:         $L_{i+1} \leftarrow L_{i+1} \cup \{x\}$ 
14:      else
15:        return  $\text{dis}(v, u) + \text{dis}(v, x) + 1$ 
16:      end if
17:    end for
18:  end for
19: end for
    
```

引理 4.5. 该算法复杂度为 $O(|V|)$ 。

证明. 由于在返回前每个点最多被经过一次, 显然复杂度为 $O(|V|)$ 。 $\square$

引理 4.6. 如果最小环经过 v , 该算法能给出最小环长度的 *surplus-1* 近似。进一步的, 如果最小环长度为偶数, 那么能返回其精确值。

证明. 显然 $\forall v_0 \in L_i, \text{dis}(v, v_0) = i$ 。设在扫到 (u, x) 这条边时返回。分两种情况讨论。

1. 如果 $x \in L_i$, 那么 $\text{dis}(v, x) = \text{dis}(v, u)$, 所以此情况对应上题中的第一种情况。同时可以注意到这找到的一定是满足这种情况的最小的 d 。
2. 如果 $x \notin L_i$, 那么就有 $x \in L_{i+1}$, 所以至少有两种不同最短路到 x , 对应上题中的第二种情况。这找到的也一定是满足这种情况的最小的 d , 但注意此时对应的 $d = i + 1$ 。

因此, 如果最小环长度 q 为偶数, 在 $L_1, \dots, L_{\frac{q}{2}-1}$ 中一定不会出现第一种情况。那么在 $i = \frac{1}{2} - 1$ 时, 会找到这个环, 返回精确答案。

如果 q 为奇数, 在 $i = \frac{q-1}{2}$ 时会找到这个环, 但此时不能保证之前没有遇到第二种情况。所以返回的答案一定为 q 或 $2 \cdot (\frac{q-1}{2} + 1) = q + 1$ 之一, 为合法的 *surplus-1* 近似。 $\square$



这样我们就以 $O(|V|)$ 的复杂度内求出了经过一个点的最小环长度。对每个点均运行一次即可。时间复杂度 $O(|V|^2)$ 。注意这里已经达到了复杂度下界, 因为 $|E|$ 可以为 $\Theta(|V|^2)$ 量级。

例题 9 (毛估估就行⁷)。给定一张无向无权简单图 $G = (V, E)$, q 次查询两点之间 surplus-2 最短路长度。保证 $|V| \leq 8000, q \leq 10^6$ 。

解法 考虑例题 7 中用到的技巧, 选择一个点 v , bfs 算出 v 到其他点的距离。那么如果对于两个点 x, y , 最短路经过了 v 或 v 的邻域, 那么就可以用 $d_x + d_y$ 来近似该最短路长度, 其中 d_x, d_y 分别为 v 到 x, y 的距离。容易发现这是 surplus-2 的。

类似的, 每次选择度数最大的点作为 v , 再删除 v 及其邻域。一直操作直到所有点都被删除, 每次查询时求出 $\min_v (d_x + d_y)$ 即可。这样就给出了一个 $O(|V|^2 + q|V|)$ 的做法。

由于 q 过大, 并且查询复杂度为删除次数, 考虑缩小删除次数。设只选 B 个 v 删除。设剩余的图为 $G' = (V', E')$ 。现在只需支付 $O(qB)$ 的代价后求出 G' 中两两的 surplus-2 距离即可。

注意到, 由于每次删除时 v 的度数单调不增, 所以剩余的点度数均不超过 $\frac{|V|}{B}$, 这意味着 $|E'| \leq O(\frac{|V|^2}{B})$ 。那么简单从所有点开始 bfs 即能做到 $O(\frac{|V|^3}{B})$ 的时间复杂度。

总时间复杂度为 $O(\frac{|V|^3}{B} + qB)$ 。取 $B = \Theta(|V|^{\frac{2}{3}} q^{-\frac{1}{3}})$ 即可做到 $O(|V|^{\frac{5}{3}} q^{\frac{1}{3}})$ 的时间复杂度。由于常数非常小, 可以通过此题。

例题 10。给定一张无向无权简单图 $G = (V, E)$, 求出两两之间 surplus-2 最短路长度。

解法

设 $n = |V|, m = |E|$ 。如果只用上一题的方法, 把 q 当成 n^2 , 我们只能做到 $O(n^{\frac{5}{3}})$ 的复杂度。目前学术界最优秀的做法为 $\tilde{O}(n^{\frac{7}{3}})$, 见 [1], 我们在这里去掉其中的 polylog 因子, 给出一个 $O(n^{\frac{7}{3}})$ 的做法。

仔细回顾上一题的做法, 发现第二步中允许误差的性质并没有被利用。我们只是在一张点数为 n , 边数为 m 的图上做了 n 次 bfs, 用 $O(nm)$ 的时间复杂度来求出两两之间的距离。这里给出一个 $O(n^{\frac{3}{2}} m^{\frac{1}{2}})$ 的做法来求出两两之间 surplus-2 的距离。这时如果取 $B = n^{\frac{1}{2}}$, 我们就能以 $O(n^2 B + n^{\frac{3}{2}} B^{-\frac{1}{2}}) = O(n^{\frac{7}{3}})$ 的时间复杂度内解决问题。

我们给出如下算法:

Algorithm 3 ALLPAIRSALMOSTSHORTESTPATH

Require: A graph $G = (V, E)$

- 1: Let $s = \sqrt{\frac{m}{n}}, t = \lceil \frac{n}{s} \rceil$
 - 2: Let $S = \{u \mid \deg(u) \geq s\}, T = \emptyset, E' = \emptyset$
-

⁷来源: 本人命制, <https://uoj.ac/problem/695>

```

3: for  $i = 1 \dots t$  do
4:   Call a vertex  $v$  cover a vertex  $u$  iff  $u = v$  or  $(u, v) \in E$ 
5:   Find a vertex  $v$  which covers the maximum number of vertices in  $S$ 
6:   Let  $S_0 = \{u \mid u \in S \wedge v \text{ covers } u\}$ 
7:    $T \leftarrow T \cup v, S \leftarrow S \setminus S_0$ 
8:   for  $u \in S_0$  do
9:      $E' \leftarrow E' \cup \{(u, v)\}$ 
10:  end for
11: end for
12: for  $(u, v) \in E, u \in S$  or  $v \in S$  or  $\deg(u) < s$  or  $\deg(v) < s$  do
13:    $E' \leftarrow E' \cup \{(u, v)\}$ 
14: end for
15: for  $v \in T$  do
16:   Run bfs from vertex  $v$  on  $G$  to find the distance between  $v$  and other vertices
17: end for
18: for  $v \in V$  do
19:   Run bfs from vertex  $v$ , using edge set  $E'$  and weighted edges  $\forall u \in T, (v, u, \text{dis}(v, u))$ 
20:   The results are the surplus-2 distances between  $v$  and other vertices
21: end for
    
```

需要证明该算法给出了正确的 surplus-2 近似, 并且时间复杂度不超过 $O(n^{\frac{3}{2}}m^{\frac{1}{2}})$ 。

引理 4.7. 该算法给出了正确的 surplus-2 近似最短路。

证明. 设 S_1 为 $\{u \mid \deg(u) \geq s\} \setminus S$, 即被 cover 的那些度数大于等于 s 点的集合。考察原图中 u, v 两点间的最短路。

如果最短路上没有 S_1 中的点, 那么这条路径上的所有边都在 E' 中, bfs 时能求出精确答案。否则考察 u 到 v 路径中最后一个属于 S_1 的点, 设为 x 。由于 $x \in S_1$, 一定有一个 T 中的点 cover 了 x , 设为 y , 所以 $(y, x) \in E'$ 。由于只有两端都在 S_1 中的边不在 E' 中, 所以 u 到 v 路径中 x 到 v 的部分一定均在 E' 中。

通过 $(u, y, \text{dis}(u, y)), (y, x)$ 与 x 到 v 的路径, u 能走到 v , 而因为 $\text{dis}(u, y) + 1 \leq \text{dis}(u, x) + 2$, 并且 bfs 出的最短路一定不会更短, 所以这一定能返回一个合法的 surplus-2 近似距离。□

引理 4.8. $|E'| = O(ns)$

证明. 把 E' 分成三部分的并, 第一部分为至少一端度数 $< s$ 的边, 第二部分为至少一端在 S 中的边, 第三部分为伪代码第 9 行中加入的边。设边集分别为 E'_1, E'_2, E'_3 , 有 $E' = E'_1 \cup E'_2 \cup E'_3$, 即有 $|E'| \leq |E'_1| + |E'_2| + |E'_3|$ 。显然 $|E'_3| \leq n$ 。分别证明 $|E'_1|, |E'_2| = O(ns)$ 。

因为 E'_1 中的边至少一端度数小于 s , 所以

$$|E'_1| \leq \sum_{\deg(u) < s} \deg(u) \leq ns$$

对于 E'_2 , 考虑反证法。如果 $|E'_2| > ns$, 那么对于任意一条在 E'_2 中的边 (u', v') , 不妨设 $u' \in S$, 会有 v' 能够 cover u' 。容易发现这些点对互不相同, 所以会产生至少 $|E'_2|$ 对能够 cover 的点对。由抽屉原理, 一定存在一个点能够 cover 至少 s 个 S 中的点。因此, 在伪代码中找的 t 轮中, 由最大性, 每次找的点能够 cover 的 S 中点数至少为 s , 由 $t = \left\lceil \frac{n}{s} \right\rceil$, 总共至少 cover 了 n 个点, 而初始 $|S|$ 至多为 n 。所以最后 S 一定为空集, 此时 $|E'_2| = 0$, 矛盾!

所以 $|E'| \leq |E'_1| + |E'_2| + |E'_3| \leq ns + ns + n = O(ns)$ 。 $\square$

引理 4.9. 该算法的复杂度为 $O(n^{\frac{3}{2}}m^{\frac{1}{2}})$ 。

证明. 对于求出 T 的部分, 复杂度为 $O(n|T|) = O(nt) = O(n^2s)$ 。对于从 T 中的每个点运行 bfs 的部分, 复杂度为 $O(m|T|) = O\left(\frac{mn}{s}\right)$ 。对于从 V 中每个点运行 bfs 的部分, 由于 $E' = O(ns)$ 且带权边均与起点相连, 每次 bfs 可以做到 $O(ns)$ 的时间复杂度。所以该部分复杂度为 $O(n^2s)$ 。

总复杂度为 $O(n^2s + \frac{mn}{s}) = O(n^{\frac{3}{2}}m^{\frac{1}{2}})$ 。 $\square$

这样, 综上所述, 我们找到了一个 $O(n^{\frac{3}{2}}m^{\frac{1}{2}})$ 时间复杂度内的 surplus-2 近似算法。再结合前面, 就有了一个 $O(n^{\frac{7}{3}})$ 的 surplus-2 近似两两最短路算法。 $\blacksquare$

注意到该做法比较容易实现, 并且有着不错的常数表现, 可以被应用到 OI 中。

5 总结

近似算法在 OI 中有着广阔的应用前景。无论是已经被广泛应用的自适应辛普森积分, 还是尚未得到普及的 NP 或图上近似算法, 都可以在 OI 比赛中占据一席之地。同时, 对于解题者来说, 它与动态规划、贪心等算法类似, 难以找到一个通用的方法, 也就更考验选手的组合技巧和思维能力, 这与算法竞赛的初衷相契合。

但是, 近似算法题目的一个缺点是数据不容易构造, 因为要判断选手代码的正确性往往需要用到精确答案的值, 而精确值是更加难以求出的。然而, 如果题目精度要求很高, 可以将有概率把误差稍大于规定误差的答案判定为正确作为代价, 来保证标准算法能够通过。这在大部分精度要求较高的题目中是一个可取的方案。

笔者在本文中展示的近似算法只是该领域的冰山一角。事实上, 还有更多的问题, 在允许误差后, 能够用各种不同的近似算法加以解决。而这些问题也有待感兴趣的同学们进一步探讨研究。甚至在学术界, 近似算法也是一个广阔而未被开发完全的新兴领域。或许看到这里的你, 将来也可以用自己的名字来命名一种全新的近似算法。



致谢

感谢中国计算机学会提供学习和交流的平台。

感谢人大附中叶金毅老师的关心和指导。

感谢家人、朋友对我的支持与鼓励。

感谢邓明扬学长、周航锐同学、李白天同学、戴江齐同学与我讨论以及给予我的启发。

感谢周航锐同学、陈于思同学、杨钧天同学、吴航同学为本文审稿。

参考文献

- [1] Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. SIAM Journal on Computing, 29(5):1740–1759, 2000.
- [2] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. SIAM Journal on Computing, 7(4):413–423, 1978.
- [3] Ce Jin. An Improved FPTAS for 0-1 Knapsack. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019), volume 132 of Leibniz International Proceedings in Informatics (LIPIcs), pages 76:1–76:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [4] Edward C. Fazekas Jr. and Peter R. Mercer. Elementary proofs of error estimates for the midpoint and simpson's rules. Mathematics Magazine, 82(5):365–370, 2009.
- [5] Wikipedia contributors. Approximation algorithm — Wikipedia, the free encyclopedia, 2021.
- [6] Wikipedia contributors. Knapsack problem — Wikipedia, the free encyclopedia, 2021.
- [7] Wikipedia contributors. Hoeffding's inequality — Wikipedia, the free encyclopedia, 2022.



图染色问题初探

广州大学附属中学 彭博

摘要

图染色问题在计算机科学中是一个非常重要且有趣的问题。在这篇论文中，作者将介绍若干个不同的染色问题，分为离线问题和在线问题两类，并给出它们的一些比较优秀的解法。

1 引言

求一般图的最小色数是 NP 完全问题，甚至三染色也是 NP 完全问题。因此，有更多的工作放在求出次优解上，也就是在多项式时间内最小化染色所用的色数。

在这篇论文中，作者将介绍在一些特殊图上能达到的比较好的结果，并且分为离线问题和在线问题两类。离线指的是一开始可以获得图的所有信息，而在线则只能一次获得一个点的信息，且必须立刻给出染色。两个方面各有不少有趣的成果。

2 记号和定义

对图对象（如点和边）进行某些限制（如相邻点颜色不能相同）下的染色问题，称为图染色问题。最经典的模型是对点染色，另外也有一些其他模型。在这一节中，我们给出一些常用的记号和定义。

Definition 2.1. 若无特殊说明，我们的研究对象是简单无向图 $G = (V, E)$ ，并且默认 $n = |V|$ ，且所有点按照 $1, 2, \dots, n$ 进行编号。

Definition 2.2.

$$N(v) = \{u \mid (u, v) \in E\}, d(v) = |N(v)|$$

Definition 2.3.

$$D(S) = \sum_{v \in S} d(v), N(S) = \bigcup_{v \in S} N(v)$$

注意并不一定有 $D(S) = |N(S)|$ 。

Definition 2.4 (离线图染色 (点染色)). 给定一个简单无向图 $G = (V, E)$, 构造一个映射 $f: V \rightarrow \mathbb{Z}_+$, 使得 $\forall (u, v) \in E, f(u) \neq f(v)$, 并最小化 $\max_{v \in V} f(v)$ 。

Definition 2.5 (在线图染色 (点染色)). 维护一个简单无向图 $G = (V, E)$, 初始 $V = E = \emptyset$ 。交互进行 n 轮, 在第 i 轮中会向图中加入编号为 i 的点, 以及 i 向所有编号小于 i 的点连的边。算法仍需构造一个映射 $f: V \rightarrow \mathbb{Z}_+$, 但加入编号为 i 的点时需立即给出 $f(i)$ 并在之后不能修改。仍需满足 $\forall (u, v) \in E, f(u) \neq f(v)$, 并最小化 $\max_{v \in V} f(v)$ 。

若没有特别说明, 默认交互库可以通过之前的输出来构造此轮的输入, 但图的总点数给定。

3 离线图染色问题

在某些极特殊的图上 (比如弦图), 存在多项式算法能找到最优解, 这里不展开。

3.1 边染色

Definition 3.1 (离线图染色 (边染色)). 给定一个简单无向图 $G = (V, E)$, 构造一个映射 $f: E \rightarrow \mathbb{Z}_+$, 使得 $\forall (u, v), (u, w) \in E \wedge v \neq w, f((u, v)) \neq f((u, w))$, 并最小化 $\max_{(u, v) \in E} f((u, v))$ 。

由于任取一个点 v 都会导致 $d(v)$ 条边的颜色两两不能相同, 所以边染色的一个显然的下界是 $\max_{v \in V} d(v)$ 种颜色。

Theorem 3.1 (Vizing's Theorem). 边染色的最少颜色数必然是 $\max d(v)$ 或 $\max d(v) + 1$ 。

证明. 对边数归纳证明。从空图开始, 每次新增一条边并把它染色。设现在准备加入的边是 (v, w_1) 。

因为有 $d_{\max} + 1$ 种颜色, 所以每个点都有至少一种没有被相邻的边用过的颜色。设 v 的空余颜色是 a , w_1 以及接下来可能出现的 w_i 现在的空余颜色是 b_i 。

如果 $a = b_1$ 则直接染色即可。否则, 从 w_1 开始可以走出唯一一条 a/b_1 的交错路径。

如果这条 a/b_1 路径的末尾不是 v , 那么只需要把这条路径上的 a, b_1 颜色交换, 即可使得 w_1 的空余颜色为 a , 然后将新边染色为 a 。否则, 设这条路径的倒数第二个点为 w_2 , 则 (v, w_2) 的颜色为 b_1 。

把边 (v, w_2) 抹去, 然后把 (v, w_1) 染色为 b_1 。现在要加入的边变为 (v, w_2) 了。

对 $w_2, w_3, \dots$ 继续进行这个过程, 过程中如果某个点的交错路径不会走回 v 则立刻可以染色成功。否则, 因为颜色是有限的, 所以一定会在某个时刻, 做到 w_j 时会发现 b_j 在之前出现过了, 设 $b_j = b_i$ 。

因为 (v, w_j) 原先的颜色是 b_{j-1} , 因此 $b_j \neq b_{j-1}$, 所以 $i < j - 1$ 。



图 1: 括号外是现在的颜色, 括号内是原来的颜色

考虑从 v 开始的 b_i/a 交错路径, 会经过 w_i , 走到 w_{i+1} , 然后断掉。由于 w_j 现在没有 b_i 颜色的邻边, 所以这条交错路径不会经过 w_j 。

因此从 w_j 出发的 a/b_i 路径不会走到 v , 将这条路径反色之后把 (v, w_j) 染色为 a 即可。□

3.2 平面图染色

Definition 3.2 (平面图). 一个简单无向图 $G = (V, E)$ 是平面图, 当且仅当可以把它画在平面上, 使得边互不相交。

四色定理告诉我们, 任意一个平面图都可以被四染色, 但是这个定理并没有给出构造。这里描述并证明四色定理的一个弱化版本——五色定理。

Theorem 3.2. (五色定理) 一个简单平面图 G 必然存在一种至多用五种颜色的染色方案。

在给出算法之前, 先证明一个简单引理。

Lemma 3.1. 任意一个非空简单平面图 G 必然存在一个点度数不超过 5。

证明. 设 V, E, C, F 分别为图的点数、边数、连通块数、面数, 不妨设 $V \geq 3$ 。

根据欧拉定理, $E = V + F - C - 1 \leq V + F - 2$, 而注意到一个面的边界上至少有三条边, 一条边只会在至多两个面的边界上, 因此 $F \leq \frac{2}{3}E$, 所以 $E \leq 3V - 6$ 。

所以度数之和为 $2E \leq 6V - 12 < 6V$, 所以一定存在一个度数不超过 5 的点。□

然后可以证明五色定理。

证明. 对图的点数进行归纳。

由 Lemma 3.1, 图中必然存在一个度数不超过 5 的点 x 。把 x 从图中删去, 由归纳假设可知剩下的图可以找到一个只用 5 种颜色的染色方案。现在要把 x 加回来。

如果 $d(x) < 5$, 或是与 x 相邻的点中存在同色点, 那么容易直接对 x 染色。否则, 设与 x 相邻的点按顺时针方向分别是 a_1, a_2, a_3, a_4, a_5 , 并不妨假设 a_i 的颜色是 i 。

类似 Theorem 3.1 的证明, 考虑从 a_1 开始, 通过 $1/3$ 交错路径能遍历到的所有点组成的集合 S 。如果 $a_3 \notin S$, 那么把 S 内所有点的颜色 $1, 3$ 互换, 然后即可把 x 染色为 1 。否则, 我们知道存在一条 $1/3$ 交错路径能从 a_1 走到 a_3 。

此时再对 a_2, a_4 进行类似的讨论。但是如果 a_2 能通过一条 $2/4$ 交错路径来到 a_4 , 那么这条路径与前面的 $1/3$ 路径必然相交, 且不可能有公共点, 与平面图性质矛盾。因此把从 a_2 开始的 $2/4$ 交错路径反色之后就可以把 x 染色为 2 。□

3.3 三分图染色

三分图听起来与二分图类似, 但染色的难度却天差地别, 这就使得三分图染色成为一个非常有趣的问题。在这一小节里, 我们介绍由 [2] 发明的算法。

Definition 3.3 (三分图). 一个简单无向图 $G = (V, E)$ 被称为三分图, 当且仅当可以把 V 划分成三个集合 V_1, V_2, V_3 , 使得它们在 G 中的导出子图都是独立集。

3.3.1 朴素算法 [7]

注意到, 任取一个点 v , $N(v)$ 的导出子图必然是二分图, 否则与三分图性质矛盾。而二分图是可以 $O(n)$ 确定一个染色方案的。

因此, 只要有 $d(v)$ 较大的点, 就可以只用两种颜色把 $N(v)$ 从图中删去。而较小的点显然可以贪心。

这启发我们使用根号分治: 设定阈值 B , 每次取出一个 v 使得 $d(v) > B$, 然后就可以用 2 种颜色删除至少 B 个点。当所有点的度数都小于 B 时则可以直接使用贪心算法。用的颜色数不超过 B 。总色数不超过 $2\frac{n}{B} + B = O(\sqrt{n})$ 。

3.3.2 问题分析

把朴素算法的基本思路提取出来, 并找到可以推广的地方。我们希望每次能在多项式时间内把问题化归到一个更小的三分图, 并且化归的过程用掉的额外颜色数相对于消掉的点数很少。

定义一个函数 $f(n) = n^\alpha \log^\beta n$ ($0 < \alpha < 1$) 表示目标颜色数, 每次如果能用 $O(1)$ 种颜色消掉 $\Omega(n/f(n))$ 个点, 那么根据主定理可知最终用的颜色数为 $O(f(n))$ 。

受到朴素算法的启发, 我们可以考虑三种进展:

- 进展一: 找到一个点集 V_1 , 使得 V_1 的导出子图是二分图, 且 $|V_1| = \Omega(n/f(n))$ 。
- 进展二: 找到一个点集 V_2 , 使得 V_2 的导出子图是二分图, 且 $|N(V_2)| = O(f(n)|V_2|)$ 。
- 进展三: 找到点 u, v , 在任意一种三染色方案中 u, v 都必须同色。

进展一直接提供了用 $O(1)$ 种颜色消掉 $\Omega(n/f(n))$ 的方法；进展三可以让我们把两个点合并而不需担心丢失合法方案；进展二的意义则在于我们可以把 $V_2 \cup N(V_2)$ 一起删除，这样删至空图时所有 V_2 拼起来就获得了一个进展一。

这三种进展每一种单独出现都可以找到一个 $O(f(n))$ 的染色方案，而事实上，它们不管以任何顺序出现，都可以导向一个 $O(f(n))$ 的染色方案。

Lemma 3.2. 只要存在一个多项式复杂度的算法 $\mathcal{A}$ ，对任何一个三分图 G' 都可以找到某一种进展，那么就存在一个多项式复杂度的算法 $\mathcal{B}$ 能对任何一个三分图 G 给出一个 $O(f(n))$ 的染色方案。

证明. 显然， $\mathcal{A}$ 在 G 的任意子图上返回进展三都可以推出 u, v 在 G 的所有三染色方案里同色，此时 $\mathcal{B}$ 只需要合并两个点然后重新开始。因此可以认为 $\mathcal{A}$ 在 G 的子图上只返回进展一和进展二。

考虑怎么把进展二合并为进展一。每次获得进展二时把 $V_2 \cup N(V_2)$ 从 V 中删除，那么会有两种情况：

1. 在 V 的大小减半之前获得了一个进展一。此时直接返回这个进展一即可。
2. 在 V 的大小减半之前只获得了进展二。那么把所有进展二的 V_2 并起来即可获得一个进展一。

所以总是可以获得进展一。 $\square$

因此，接下来我们只需要关心怎样获得进展，而不需要关心进展的类型和顺序。

3.3.3 性质挖掘

我们尝试特判掉一些简单的情况，来给图增加一些性质。

首先，如果 $d(v) = O(f(n))$ 那么获得一个进展二，所以我们可以认为每个点的度数都至少是 $\Omega(f(n))$ 。

接下来，对于某对 u, v ，设 $S = N(u) \cap N(v)$ ，那么会有两种情况。

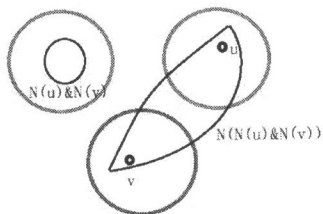


图 2: uv 不同色

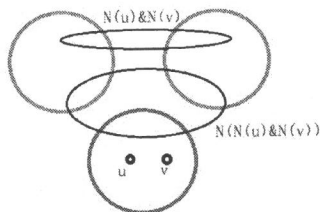


图 3: uv 同色

假设 $|S| = \Omega(n/f(n)^2)$ ，对 $N(S)$ 分类讨论

1. 如果 $N(S)$ 是一个二分图, 那么考虑它的大小。如果 $|N(S)| \geq f(n)|S| = \Omega(n/f(n))$ 则获得了一个进展一, 否则 $|N(S)| \leq f(n)|S|$ 则获得进展二。
2. 否则, 由上图可知 u, v 在任何一种三染色方案中都必须是同色的。此时获得了一个进展三。

因此, 任意两个点的共同邻居个数是 $O(n/f(n)^2)$ 。

3.3.4 随机三分图

三分图天然有三个很大的独立集, 但是能否从中提取出一个较大的独立集呢?

我们先考虑随机三分图的情况, 每个点的颜色随机, 邻居也是随机抽取。设三种颜色的点集分别是 red, blue, green。

扩展我们的视野, 从一个点 v 的直接邻居 $N(v)$ 扩展到其二阶邻居 $N(N(v))$ 。不妨设 $v \in \text{red}$, 那么 $N(v)$ 应当均匀分布在 blue + green 里, 于是 $N(N(v))$ 就应该有大约一半的点都是红色的, 即最大独立集大约占了所有点的一半。

此时我们可以应用一个最小点覆盖的近似算法¹, 它可以多项式时间内求出大小至多为 $2 - \frac{\log \log n}{2 \log n}$ 倍最优解的点覆盖, 称这个算法为 BE/MS 算法。

由于 $N(N(v))$ 的最大独立集至少占了一半, 那么最小点覆盖大小就不会超过 $\frac{1}{2}|N(N(v))|$, 所以 BE/MS 算法求出的点覆盖大小不会超过 $(1 - \frac{\log \log n}{4 \log n})|N(N(v))|$, 那么我们就求出大小至少为 $\frac{\log \log n}{4 \log n}|N(N(v))|$ 的独立集了。

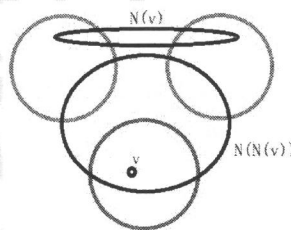


图 4: 一阶和二阶邻居

3.3.5 最坏情况

我们给出两个额外的定义。

Definition 3.4.

$$d_T(v) = |N(v) \cap T|, D_T(S) = \sum_{v \in S} d_T(v)$$

然而, 在最坏情况下这样的 v 是否能存在呢?

不妨假设 $D(\text{red})$ 在三种颜色中是最大的, 就可以推出

¹由 [1] 和 [5] 分别独立发现

$$D_{\text{red}}(V - \text{red}) \geq \frac{1}{2} D(V - \text{red})$$

这个形式与 $D_{\text{red}}(N(v)) \geq \frac{1}{2} D(N(v))$ 非常类似，我们尝试左右分别求和来证明存在一个 v 满足这一点。

然后还需要考虑怎么把度数转为点数，即把 $D_{\text{red}}(N(v))$ 转化为 $|N(N(v)) \cap \text{red}|$ 。

试着把 $D_{\text{red}}(N(v)) \geq \frac{1}{2} D(N(v))$ 的左右同时对所有 $v \in \text{red}$ 求和

$$\begin{aligned} \sum_{v \in \text{red}} LHS &= \sum_{u \in V - \text{red}} d_{\text{red}}(u)^2 \\ \sum_{v \in \text{red}} RHS &= \frac{1}{2} \sum_{u \in V - \text{red}} d_{\text{red}}(u) d(u) \end{aligned}$$

图 5: 红色点集的度数较大

我们发现每个 u 都多乘了 $d_{\text{red}}(u)$ ，从而 $D_{\text{red}}(V - \text{red}) \geq \frac{1}{2} D(V - \text{red})$ 不能直接推出 $\sum LHS \geq \sum RHS$ ，而事实上在度数相差很大时后者确实可能不成立。

由于我们无法求出 red ，所以我们应该尝试控制 $d(u)$ 。只要能令 $d(u)$ 非常接近，就可以根据“平方的期望大于等于期望的平方”来证明 $\sum LHS \geq \sum RHS$ 。

另外，即使存在 $D_{\text{red}}(N(v)) \geq \frac{1}{2} D(N(v))$ ，也不能从度数占一半推出点数占一半。所以度数并不能直接反映到点数。

可以发现，矛盾的根源是度数的不均匀。只要能保证度数相近，就可以找到很好的解决方案。所以我们再给出几个定义

Definition 3.5.

$$\delta = \delta(n) = 1/(5 \log n)$$

$$I_j = \{v \in V \mid d(v) \in [(1 + \delta)^j, (1 + \delta)^{j+1})\}$$

$$N_i(S) = \{v \in N(S) \mid d_S(v) \in [(1 + \delta)^i, (1 + \delta)^{i+1})\}$$

3.3.6 两个定理

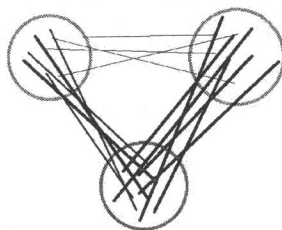
Theorem 3.3. 给定一个图 G ，最小度数为 d ，再给一个独立集 R 和某个 $0 \leq \lambda \leq 1$ 使得 $D_R(V - R) \geq \lambda D(V - R)$ 。那么可以找到某个 $v \in R$ 和某个 I_j ，使得

$$|N(v) \cap I_j| \geq \delta^2 d / \log_{1+\delta} n \quad (1)$$

$$D_R(N(v) \cap I_j) \geq \lambda(1 - 3\delta) D(N(v) \cap I_j) \quad (2)$$

设 $B = I_j \cap (V - R)$ 。也就是说，如果用 B 替换 $V - R$ ， $N(v)$ 仍然足够大，并且此时 $N(v)$ 就有接近一半的边连向 R 。

设 $S = N(v) \cap I_j$ 为 Theorem 3.3 找到的集合，我们再来关注 S 的邻居，分别考虑每一个 $N_i(S)$ 。



Theorem 3.4. 给定一个图 G ，一个点集 R ，和 $\lambda' \in [0, 1]$ 。对于任意满足 $D_R(S) \geq \lambda' D(S)$ 的集合 S ，都存在一个 $i < \log_{1+\delta} n$ ，使得

$$D_{N_i(S) \cap R}(S) \geq \delta D_R(S) / (\log_{1+\delta} n) \quad (3)$$

$$|N_i(S) \cap R| / |N_i(S)| \geq (1 - 2\delta)\lambda' \quad (4)$$

设 $T = N_i(S)$ 。也就是说，如果用 T 替换 V ， $D_R(S)$ 不会减小太多，并且 $N(S)$ 接近至少 λ' 的点都属于 R 。

如果我们暂时认为这两个定理是正确的，就可以获得一个很强大的推论。

Corollary 3.1. 给定一个 n 个点的三色图 G ，使得

1. 任意两个点不会共有超过 s 个邻居。
2. G 的最小度数 $d_{\min} \geq 10(\log_{1+\delta} n)/\delta$ 。

那么存在 $v \in V, i, j \in [0, 5 \log^2 n]$ 使得集合 $T = N_i(N(v) \cap I_j)$ 的大小为 $\Omega(d_{\min}^2 / (s \log^7 n))$ ，且至少 $\frac{1}{2}(1 - \frac{1}{\log n})$ 的点都在 G 的某个三染色方案中是红色。

证明. 取 $R = \text{red}, \lambda = 1/2$ ，则由 Theorem 3.3 可知存在 v, I_j 满足 (1) 和 (2)。设 $S = N(v) \cap I_j$ 。再令 $\lambda' = \frac{1}{2}(1 - 3\delta)$ ，则由 Theorem 3.4 可知存在 i 满足 (3) 和 (4)。设 $T = N_i(S)$ ，则

$$\begin{aligned} D_{T \cap R}(S) &\geq \delta D_R(S) / (\log_{1+\delta} n) && (\text{Theorem 3.4, equation (3)}) \\ &\geq \delta [\lambda(1 - 3\delta)D(S)] / (\log_{1+\delta} n) && (\text{Theorem 3.3, equation (2)}) \\ &\geq \delta \lambda(1 - 3\delta) [d_{\min}|S|] / (\log_{1+\delta} n) && (D(S) \geq d_{\min}|S|) \quad (*) \\ &\geq \delta^3 \lambda(1 - 3\delta) d_{\min}^2 / (\log_{1+\delta} n)^2 && (\text{Theorem 3.3, equation (1)}) \\ &= \Omega(\delta^5 d_{\min}^2 / (\log^2 n)) && (\log_{1+\delta} n = O(\frac{1}{\delta} \log n)) \\ &= \Omega(d_{\min}^2 / (\log^7 n)) && (\delta = \frac{1}{5 \log n}) \end{aligned}$$

由于任意两个点的共同邻居不超过 s 个，而 $S \subset N(v)$ ，所以 T 中的点除了 v 以外在 S 里的邻居不超过 s 。所以 $|T| \geq |T \cap R| \geq (D_S(T \cap R) - d_S(v))/s \geq (D_{T \cap R}(S) - |S|)/s$ 。

另外，由 (*)，且 $d_{\min} \geq 10(\log_{1+\delta} n)/\delta$ ，可得 $|S| \leq \frac{1}{2} D_{T \cap R}(S)$ 。所以

$$|T| \geq \frac{1}{2} D_{T \cap R}(S) = \Omega(d_{\min}^2 / (s \log^7 n))$$

并且 T 满足

$$\begin{aligned}
 |T \cap R|/|T| &\geq (1 - 2\delta)\lambda' && (\text{Theorem 3.4, equation (4)}) \\
 &\geq \frac{1}{2}(1 - 5\delta) \\
 &= \frac{1}{2}\left(1 - \frac{1}{\log n}\right)
 \end{aligned}$$

□

接下来我们考虑如何证明 Theorem 3.3 和 Theorem 3.4。

3.3.7 定理的证明

直观理解, I_j, N_i 都只把所有点分成了 $5\log^2 n$ 份, 所以肯定存在某一份里的点数和边数足够多。但是为了保证这一份质量还足够好 (比如说大部分边都连向 R), 就需要找到数量和质量之间的平衡。

Lemma 3.3. 有 b 个球, 其中 r 个是红色。把它们放在 k 个盒子里, 那么任取一个 $\epsilon \in [0, 1)$, 都存在某个盒子, 使得其中有至少 $\epsilon r/k$ 个红球, 且这个盒子中红球占比超过 $(1 - \epsilon)r/b$ 。

证明. 把小于 $\epsilon r/k$ 个红球的盒子全部丢掉, 这样至多丢掉 ϵr 个红球。剩下的红蓝球比例至少是 $(r - \epsilon r)/(b - \epsilon r) > (1 - \epsilon)r/b$, 所以一定还剩下一个合法的盒子。 □

证明 (Theorem 3.3). 证明分为三步。

首先要找到一个 I_j , 设 $B = I_j \cap (V - R)$, 使得 B 有与 $V - R$ 类似的性质。

然后, 因为我们需要 $N(v) \cap I_j$ 足够大, 所以需要把不合法的 v 从 R 中筛选掉。设筛选之后剩下 R' , 需要证明 R' 有与 R 类似的性质。

最后, 因为度数已经被非常紧地限制住了, 可以通过两边求和来证明存在一个 $v \in R'$ 使得 $N(v) \cap B$ 满足条件。注意 $N(v) \subseteq V - R$, 所以 $N(v) \cap B = N(v) \cap I_j$ 。

考虑怎么找到合法的 I_j 。把每一个 I_j 看做一个盒子, 并且对于每一个 $v \in I_j \cap (V - R)$, 往第 j 个盒子中放 $d(v)$ 个球, 其中 $d_R(v)$ 个是红色的。那么在引理中令 $\epsilon = \delta$, 就一定存在某个 j_0 , 如果令 $B = I_{j_0} \cap (V - R)$, 则

$$D_R(B) \geq \delta D_R(V - R) / (\log_{1+\delta} n) \quad (5)$$

$$D_R(B) \geq \lambda(1 - \delta)D(B) \quad (6)$$

也就是说, 无论是从大小方面还是占比方面, $D_R(I)$ 都与 $D_R(V - R)$ 非常类似。

接下来, 我们令 $R' = \{v \in R \mid |N(v) \cap B| \geq \delta^2 d / \log_{1+\delta} n\}$, 也就是把连向 B 的边数不足的点删去了 (剩下的点都满足 (1) 的性质)。为了确保删去的不是太多, 注意到

$$\begin{aligned}
 D_{R'}(B) &\geq D_R(B) - |R|\delta^2 d / \log_{1+\delta} n \\
 &\geq D_R(B) - D_R(V-R)\delta^2 / \log_{1+\delta} n & (D_R(V-R) \geq |R|d) \\
 &\geq D_R(B) - (D_R(B)(\log_{1+\delta} n) / \delta) (\delta^2 / \log_{1+\delta} n) & \text{(equation (5))} \\
 &\geq D_R(B)(1 - \delta) \\
 &\geq \lambda(1 - 2\delta)D(B) & \text{(equation (6))}
 \end{aligned}$$

现在, 我们只需要从 R' 中找到一个 v , 使得 $D_{R'}(N(v) \cap B) \geq \lambda(1 - 3\delta)D(N(v) \cap B)$ 。只需要把两边求和:

$$\begin{aligned}
 LHS &= \sum_{v \in R'} D_{R'}(N(v) \cap B) = \sum_{w \in B} d_{R'}(w)^2 \\
 RHS &= \sum_{v \in R'} \lambda(1 - 3\delta)D(N(v) \cap B) = \lambda(1 - 3\delta) \sum_{w \in B} d_{R'}(w)d(w)
 \end{aligned}$$

因为 $d(w) \in [(1 + \delta)^{j_0}, (1 + \delta)^{j_0+1}]$, 所以

$$\begin{aligned}
 RHS &\leq \lambda(1 - 3\delta)(1 + \delta)^{j_0+1} \sum_{w \in B} d_{R'}(w) \\
 &< \lambda(1 - 2\delta)(1 + \delta)^{j_0} D_{R'}(B) \\
 LHS &= |B| \cdot \left(\frac{1}{|B|} \sum_{w \in B} d_{R'}(w)^2 \right) \\
 &\geq \frac{1}{|B|} \left(\sum_{w \in B} d_{R'}(w) \right)^2 \\
 &= \frac{1}{|B|} D_{R'}(B)^2
 \end{aligned}$$

并且 $\lambda(1 - 2\delta)(1 + \delta)^{j_0}|B| \leq \lambda(1 - 2\delta)D(B) \leq D_{R'}(B)$, 所以 $LHS \geq RHS$ 。

所以一定存在一个 $v \in R'$, 使得 $D_{R'}(N(v) \cap B) \geq \lambda(1 - 3\delta)D(N(v) \cap B)$ ((2) 得证)。□

证明 (Theorem 3.4). 只需要直接应用引理即可。

把每个 $N_i(S)$ 看做一个盒子, 对于每个 $v \in N_i(S)$ 往 $N_i(S)$ 对应的盒子里放 $d_S(v)$ 个球, 并且这 $d_S(v)$ 个球是红色的当且仅当 $v \in R$ 。那么在引理中令 $\epsilon = \delta$, 就可以找到一个 i_0 使得

$$\begin{aligned}
 D_{N_{i_0}(S) \cap R}(S) &\geq \delta D_R(S) / (\log_{1+\delta} n) & ((4) \text{ 得证}) \\
 D_{N_{i_0}(S) \cap R}(S) / D_{N_{i_0}(S)}(S) &\geq (1 - \delta)\lambda' & (7)
 \end{aligned}$$

又因为 $N_{i_0}(S)$ 的点向 S 连的边数相差不超过 $1 + \delta$ 倍, 所以

$$\begin{aligned}
 |N_{i_0}(S) \cap R|/|N_{i_0}(S)| &\geq D_{N_{i_0}(S) \cap R}(S)/D_{N_{i_0}(S)}(S)/(1+\delta) \\
 &\geq (1-\delta)\lambda'/(1+\delta) \\
 &\geq (1-2\delta)\lambda'
 \end{aligned}
 \tag{equation (7)}$$

□

3.3.8 算法描述

Algorithm 3.1. 输入一个三分图 $G = (V, E)$ ，返回一个进展。

1. 如果存在 $v \in V$ 使得 $d(v) < f(n)$ ，或是存在 $u, v \in V, u \neq v$ 使得 $|N(u) \cap N(v)| \geq n/f(n)^2$ ，那么根据 3.3.3 可以直接返回一个进展。
2. 否则，对于每一个 $v \in V, i \in [0, 5 \log^2 n], j \in [0, 5 \log^2 n]$ ，对 $T = N_i(N(v) \cap I_j)$ 运行 BE/MS 算法。
3. 如果 BE/MS 算法在某个 T 上找到了一个 $\Omega(f(n)^4/(n \log^8 n))$ 的独立集那么返回进展一。

我们只需要证明算法的第三步总是能返回一个进展一。

证明. 根据算法的第一步，图的最小度数至少是 $f(n)$ ，且任意两个点的共同邻居不超过 $n/f(n)^2$ 个。代入到上面的推论里，我们就一定能找到一个集合 T 使得 $|T| = \Omega(f(n)^4/(n \log^7 n))$ 且存在某个三染色方案使得 T 的至少 $\frac{1}{2}(1 - \frac{1}{\log n})$ 的点是红色的。

那么 T 的导出子图的最小点覆盖大小必然不超过 $\frac{1}{2}(1 + \frac{1}{\log n})|T|$ ，所以 BE/MS 算法找到的点覆盖大小就不会超过

$$(2 - \frac{\log \log |T|}{2 \log |T|}) \cdot \frac{1}{2}(1 + \frac{1}{\log |T|})|T| = (1 - \Omega(\frac{1}{\log |T|}))|T|$$

从而找到了一个大小为 $\Omega(\frac{|T|}{\log |T|}) = \Omega(f(n)^4/(n \log^8 n))$ 的独立集。

□

令 $f(n)^4/(n \log^8 n) = n/f(n)$ ，即得 $f(n) = n^{2/5} \log^{8/5} n$ 。

综上，我们在多项式时间内可以找到一个使用 $O(n^{2/5})$ 种颜色的染色方案。

4 在线图染色问题

4.1 树染色

我们首先通过一个问题来展示在线和离线的巨大差别。

Theorem 4.1. 保证图的最终形态是一棵树时,任何在线算法在最坏情况下都需要至少 $\lfloor \log n \rfloor + 1$ 种颜色。

证明. 任取一个在线算法 $\mathcal{A}$ 。

归纳证明, 无论现在已经确定的图是怎样, 总是可以另外新增一个点数为 2^k 的树, 使得 $\mathcal{A}$ 在里面使用了至少 $k + 1$ 种颜色。

$k = 0$ 的情况是显然的。

假设我们对于 $k = 0, \dots, k = i$ 都已经构造了一棵树, 分别是 $T_0, \dots, T_k$ 。那么我们可以从每一棵树中选出一个点, 使得选出来的这些点的颜色两两不同。此时只需要新增一个点与选出来的这些点连边, 就可以逼迫 $\mathcal{A}$ 使用第 $k + 2$ 种颜色了。这样我们就成功构造出了 T_{k+1} 。 $\square$

事实上, 要构造出使用不超过 $\lfloor \log n \rfloor + 1$ 种颜色的算法也非常简单。直接使用贪心即可。

4.2 特殊图染色

在这一小节中, 我们介绍几种具有特殊性质的图上的染色算法。

4.2.1 区间图

Definition 4.1. 一个简单无向图 $G = (V, E)$ 被称为区间图, 当且仅当可以有一个对应关系, 每个点 v 对应一个数轴上的一个区间 $[l_v, r_v]$, 且两个点之间有边当且仅当它们对应的区间有交。

Theorem 4.2 ([4]). 对于任意一个区间图 G , 存在一个在线算法可以用不超过 $3\omega(G) - 2$ 种颜色, 其中 $\omega(G)$ 是 G 的最大团大小。

我们先给出这个算法, 然后证明其正确性。

Algorithm 4.1 (solve(k)). 输入一个最大团大小不超过 k 的区间图, 在线返回一个染色方案。

1. 若 $k = 1$ 那么直接全部同色即可。否则新建一个 solve($k-1$)。
2. 维护两个图 G_1, G_2 , 分别表示送入 solve($k-1$) 的图, 和在这里要处理的图, 初始都为空。新增三种颜色给 G_2 使用。
3. 每次加入一个点 v , 直到没有新的点。进行如下步骤:
 - (a) 若 $\omega(G_1 + v) \leq k - 1$ 则把 v 加入 G_1 并送进 solve($k-1$) 处理。(区间图是弦图, 所以最大团可以在多项式时间内求出来)

(b) 否则, 把 v 加入 G_2 并在三种颜色中用最小的可用颜色染色。

我们需要证明三种颜色足够给 G_2 进行贪心染色。

证明. 由区间图的性质, 存在某一种合法的点到区间的映射, 设 i 对应的区间是 $[l_i, r_i]$ 。

为了区分图中的点和数轴上的点, 用 $1, 2, \dots, n, i, j, k$ 来表示图中的点, 用 x, y, z 表示数轴上的点。

容易发现, 一个区间图的最大团大小即为

$$\max_x \sum_{i=1}^n [l_i \leq x \leq r_i]$$

以这个为基础, 分析 G_2 里的区间的性质。

因为 G_2 的点都是无法被加入 G_1 的, 所以对于每一个 $i \in G_2$, 都存在一个 $x \in [l_i, r_i]$ 使得 x 在 G_1 中被 $k-1$ 个区间覆盖。把某个这样的点设为 x_i 。

考虑如果存在 $i, j \in G_2, i \neq j$ 使得 $l_i \leq l_j \leq r_j \leq r_i$, 那么 x_j 就会在 G_1 被覆盖 $k-1$ 次而在 G_2 被覆盖两次, 从而使得 $\omega(G_1 + G_2) > k$, 矛盾。因此不存在相互覆盖的区间。

考虑如果 G_2 存在三元环, 因为没有相互覆盖的情况, 就必然形如 $l_i \leq l_j \leq l_k \leq r_i \leq r_j \leq r_k$ 。但是容易发现此时 x_j 必然被 $[l_i, r_i]$ 或 $[l_k, r_k]$ 覆盖, 从而还是会使得 $\omega(G_1 + G_2) > k$, 矛盾。因此不存在三元环。

因此容易发现 G_2 的每一个点的度数都不会超过 2, 所以在 G_2 中贪心只需要使用不超过三种颜色。□

有趣的是, $3\omega(G) - 2$ 同样也是一个下界。

Theorem 4.3 ([4]). 对于任意一个在线算法 $\mathcal{A}$, 它在区间图的最坏情况至少需要使用 $3\omega(G) - 2$ 种颜色。

证明. 我们强制这个算法只能使用 $3\omega(G) - 3$ 种颜色, 并证明构造过程中会出现一个点无法被染色。

为了方便, 首先定义缩点操作: 选取一个区间 $[x, y]$, 使得所有点的区间要么包含 $[x, y]$, 要么与 $[x, y]$ 不交。那么把 $[x, y]$ 缩成一个点的意思就是强制接下来加入的点的区间也要么包含 $[x, y]$, 要么与其不交, 也就是把 $[x, y]$ 以内的区间看成一个整体。那么缩起来得到的新点就可以被看做是同时拥有区间内的所有颜色, 并且含有一个大小与区间内最大团相等的团。

我们归纳证明, 对于每一个 $1 \leq i \leq \omega(G)$, 都可以构造出任意多的点, 使得每个点包含的团的大小至多为 i , 且存在一个大小为 $3i - 2$ 的颜色集合 C , 使得 C 存在于每一个点中。

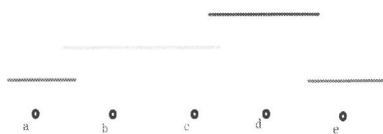
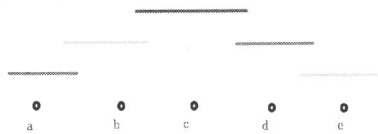
对于 $i = 1$, 我们可以先给出 N 个互不相交的区间, 那么必然存在一种颜色被至少 $N/(3\omega(G) - 3)$ 个区间使用。通过缩点, $i = 1$ 的情况得证。

现在从 i 推到 $i + 1$ 。

我们取出最左边的 5 个点，按顺序设为 a, b, c, d, e 。新增两个区间 I_1, I_2 分别恰好覆盖 a, e 。

如果 I_1, I_2 颜色相同，那么令 I_3 覆盖 b, c 且与 I_1 有交，再令 I_4 覆盖 d 且与 I_3, I_2 有交，就可以发现 I_2, I_3, I_4 分别被染了三种不同的颜色，且都不在 C 中。

否则，令 I_3 覆盖 d 且与 I_2 有交。如果 I_3 的颜色与 I_1 不同，那么已经获得了三种不在 C 中的颜色。否则可以如上构造。


图 6: I_1, I_2 颜色相同

图 7: I_1, I_3 颜色相同

设新出现的颜色是 (x, y, z) 。对每连续 5 个点都这样构造，那么由于 (x, y, z) 可能的取值是有限的，一定存在一组 (x_0, y_0, z_0) 出现了足够多次。分别缩点即可。

由此即从 i 推到了 $i+1$ 。显然推到 $i = \omega(G)$ 时算法就必须使用至少 $3\omega(G) - 2$ 种颜色了。□

4.2.2 不存在 C_3, C_5 的图

Theorem 4.4 ([3]). 若保证最终给出的图中不存在长度为 3 或为 5 的简单环，那么存在一个算法使用不超过 $2\lfloor \sqrt{n} \rfloor$ 种颜色。

证明. 我们先描述这个算法，然后证明其正确性。

Algorithm 4.2. 输入一个不存在长度为 3 或为 5 的简单环的图，在线返回一个染色方案。

1. 设阈值 $B = \lfloor \sqrt{n} \rfloor$ ，维护集合 $W_i, B < i \leq 2B$ ，初始全部为空。
2. 每次加入一个点 v ，直到没有新的点。进行如下步骤：
 - (a) 尝试贪心地用 $[1, B]$ 中的颜色染色。
 - (b) 如果失败了，但是存在某个 W_i 使得 $v \in N(W_i)$ ，那么就把 v 染成 i 。
 - (c) 否则，找到第一个 $W_i = \emptyset$ ，把 v 染成 i ，并令 $W_i = \{u \mid u \in N(v), \text{col}(u) \leq B\}$ 。

显然 $|W_i| \geq \sqrt{n}$ ，并且 $\forall j < i, W_j \cap W_i = \emptyset$ 。

我们先证明第二步不会导致一个不合法的染色方案。

证明. 假设染色过程中出现了两个点 $x < y$ 的颜色都是 c , 并且 x, y 有边相连. 那么 y 显然只能是在第二步染色.

如果 x 是第三步染色, 那么存在 $u \in W_c$ 使得 $x, y \in N(u)$, 因此存在 C_3 , 矛盾.

否则 x 肯定也是在第二步染色. 为了不出现 C_3 , x, y 不能在 W_c 中有共同的邻点, 那么就存在 $u, v \in W_c, u \in N(x), v \in N(y), u \neq v$, 且 u, v 同时是创建 W_c 的点的邻点, 所以存在 C_5 , 矛盾. $\square$

然后证明用到的颜色数不超过 $2B$. 注意到 $|W_i| \geq B$, 并且 $\forall i \neq j, W_j \cap W_i = \emptyset$, 所以最终只会不超过 $\lfloor n/B \rfloor = B$ 个 W_i 非空. 因此总颜色数不超过 $2B$. $\square$

4.3 k 分图染色

4.3.1 二分图染色

在考虑更一般的图之前, 我们先考虑二分图的情况.

因为树也是二分图, 所以 Theorem 4.1 告诉我们最坏情况下使用 $o(\log n)$ 种颜色的算法并不存在. 这里给出一个使用不超过 $2 \log n$ 种颜色的算法:

Algorithm 4.3. 输入一个二分图, 在线返回一个染色方案.

1. 对于每一个连通块都维护一个 k , 使得左部点没有用过 $k-1$, 右部点没有用过 k , 且一共只用了 $1, \dots, k$ 这些颜色.
2. 每次加入一个点 v , 直到没有新的点. 进行如下步骤:
 - (a) 考虑 v 合并的若干个连通块, 设其中最大和次大的 k 为 k_1, k_2 .
 - (b) 新的连通块使用 $k' = \max(k_1, k_2 + 2)$.
 - (c) 新点视其在左边还是右边染色 k' 或 $k' - 1$.

算法的正确性不难证明, 而使用的颜色数可以这样分析: 设 f_k 表示使得这个算法造出 k 的最小点数, 则 $f_k = \min(f_{k-1} + f_{k-2}, 2f_{k-2}) + 1 \geq 2^{k/2}$, 因此使用的颜色数不超过 $2 \log n$.

4.3.2 k 分图染色

在这个小节中, 我们认为点数 n 和色数 k 都是已知的, 且图也是提前确定的. 在这里, 我们介绍由 [6] 发明的算法.

考虑一般的 k 分图染色问题。一种思路是, 任取一个 v 都有 $N(v)$ 的导出子图是 $k-1$ 分图, 如果能用尽可能少的点将 k 分图拆解为若干个 $k-1$ 分图染色的子问题, 那么子问题就会更加简单, 从而导出一个较优的染色方案。

如何找到这样一个分割? 我们可以使用“部分贪心算法”, 设立一个阈值 s , 然后用 s 种颜色进行贪心染色。设那些无法被染色的点是 R , 而被染了第 i 种颜色的点集是 C_i , 那么 R 中的任意一个点显然都与至少一个 C_i 中的点相邻。因此任意一个 C_i 都可以把 R 拆成 $|C_i|$ 个 $k-1$ 分图。

在离线的情况下, 可以选取最小的 $|C_i|$, 那么就有 $|C_i| \leq n/s$, 递归到不超过 n/s 个 $k-1$ 分图染色问题。

通过合理地选择 $s = S(n, k)$, 离线时可以使用不超过 $(1 + O(\frac{\log k}{k}))n^{(k-2)/(k-1)}$ 种颜色。但是这与目前已知的最优算法有着较大距离 (在三分图上就几乎退化为朴素的 $O(\sqrt{n})$ 度数分治), 所以不展开分析。

考虑把这个算法转为在线。在转化的过程中, 会遇到如下两个问题:

1. 在动态加点的过程中我们无法知道哪个 C_i 最小, 也就无法保证 $|C_i| \leq n/s$ 。
2. 我们会需要在线向子问题加点, 而子问题无法预知 n 是多少, 也就无法确定 $S(n, k)$ 。

对于第一个问题, 我们可以随机选取一个 C_i , 这样获得的期望大小是 n/s 。而对于第二个问题, 可以强行按照 s 把点分块, 这样新增的子问题个数不会超过 n/s 个, 但成功把每个子问题的大小控制在 s 以内。后面会有更详细的描述。

Theorem 4.5. 存在一个随机化在线算法 $\mathcal{A}$, 在输入数据提前确定时期望使用 $O(k2^k n^{\frac{k-2}{k-1}} (\log n)^{\frac{1}{k-1}})$ 种颜色, 其中点数 n 和色数 k 不需要提前给出。

我们先认为 n, k 已知, 最后再处理 n, k 未知的情况。

定义一个可以在线处理点数为 n 色数为 k 的图的算法。其描述如下:

Algorithm 4.4 (solve(n, k)). 输入一个不超过 n 个点的 k 分图, 在线返回一个染色方案。

1. 若 $k \leq 2$ 则直接使用 *Algorithm 4.3*。
2. 设定阈值 $s = S(n, k)$, 其中 $S(n, k)$ 是某个函数。新增 s 种之前从未用过的颜色, 设为 $1, 2, \dots, s$ 。在 $1, 2, \dots, s$ 里均匀随机选取一个 r 。
3. 维护图 G , 初始为空。
4. 每次加入一个点 v , 直到没有新的点。进行如下步骤:
 - (a) 如果 v 在 G 中可以使用这 s 种颜色进行贪心染色, 那么选取可用的最小颜色进行染色, 并把 v 加入到 G 中。若 v 染色为 r 则新增一个 solve($s, k-1$)。

(b) 否则, v 必然与某一个颜色为 r 的点 u 相邻。把 v 送入 u 对应的 $\text{solve}(s, k-1)$ 里处理。如果那里面已经有 s 个点了那么新增一个 $\text{solve}(s, k-1)$ 。

令 $S(n, k) = \lceil 2^k n^{(k-2)/(k-1)} (\log n)^{1/(k-1)} \rceil$, 设 $A(n, k)$ 表示 Algorithm 4.4 在所有 n 个点的 k 分图上期望使用的颜色数的最大值。

Theorem 4.6. $k \geq 2$ 时, $A(n, k) \leq k 2^k n^{\frac{k-2}{k-1}} (\log n)^{\frac{1}{k-1}}$ 。

证明. 对 k 进行归纳。当 $k = 2$ 时直接使用 Algorithm 4.3 即可。接下来设 $k > 2$ 。

设 G 是 Algorithm 4.4 期望使用颜色数最大的图。设 C_i 是被贪心染色为 i 的点集, R 是无法被贪心染色的点集, $s = S(n, k)$ 。

对于某一个颜色 i , 设 C_i 把 R 拆成了 $G_1^i, \dots, G_{n_i}^i$, 那么由于每次需要 s 个点才能新增一个子问题, 所以 $n_i \leq |C_i| + n/s$ 。

设 $E(G_j^i)$ 表示第 i 种颜色拆出的第 j 个子图, 算法期望使用的颜色数, 那么有 $E(G_j^i) \leq A(s, k-1)$ 。由此可以得到

$$\begin{aligned} A(n, k) &\leq s + \frac{1}{s} \sum_{i=1}^s (E(G_1^i) + \dots + E(G_{n_i}^i)) \\ &\leq s + \frac{A(s, k-1)}{s} \sum_{i=1}^s n_i \\ &\leq s + \frac{A(s, k-1)}{s} \sum_{i=1}^s (|C_i| + n/s) \\ &\leq s + \frac{2n}{s} A(s, k-1) \end{aligned}$$

设 $c = 2^k, s' = s/c$ 。那么进一步把 s 展开, 利用归纳假设把 $A(s, k-1)$ 展开, 可得

$$\begin{aligned} A(n, k) &\leq s + \frac{2n}{s} A(s, k-1) \\ &\leq s + \frac{2n}{s} (k-1) 2^{k-1} s^{\frac{k-3}{k-2}} (\log s)^{\frac{1}{k-2}} \\ &= s + \frac{n}{s'} (k-1) s^{\frac{k-3}{k-2}} (\log s)^{\frac{1}{k-2}} \\ &\leq s + \frac{n}{s'} (k-1) c^{\frac{k-3}{k-2}} s^{\frac{k-3}{k-2}} (\log n)^{\frac{1}{k-2}} \\ &< s + c(k-1) n s' \frac{-1}{k-2} (\log n)^{\frac{1}{k-2}} \\ &\leq s + c(k-1) n^{1-\frac{1}{k-1}} (\log n)^{\frac{1}{k-2} - \frac{1}{(k-2)(k-1)}} \\ &= s + c(k-1) n^{\frac{k-2}{k-1}} (\log n)^{\frac{1}{k-1}} \\ &= k 2^k n^{\frac{k-2}{k-1}} (\log n)^{\frac{1}{k-1}} \end{aligned}$$

□

最后, 考虑 n, k 未知应该怎么处理。我们可以维护 n', k' 表示对点数和色数的估计, 并假设真实的点数和色数就是 n', k' 。

每当点数超出 n' 时就把 n' 提高一倍, 并换成一组全新的颜色和全新的一个 $\text{solve}(n', k')$ 。由于有 $n^{\frac{k}{2}}$ 这一项存在, 倍增并不会对渐进复杂度造成影响。

一旦 Algorithm 4.4 进行到 $\text{solve}(n, 2)$ 并发现这里的图并不是二分图, 就说明 k' 必然小于此时的 k 。同样只需要把 k' 加一, 然后换成新颜色和新的 $\text{solve}(n', k')$ 即可。由于有 2^k 这一项存在, 这也不会影响渐进复杂度。

这样就完成了 Theorem 4.5 的证明。

5 总结

图染色问题有多种变体, 而对于每种变体都可以找到很多有趣且大不相同的算法。本文介绍了多种算法, 希望能够起到抛砖引玉的作用, 启发读者进行进一步的研究, 得到更多有趣的做法和应用。

感谢

感谢中国计算机学会提供学习和交流的平台。

感谢广州大学附属中学的王晓鹏老师、蔡子健老师的关心与指导。

感谢国家集训队教练杨耀良的指导。

感谢彭思进同学为本文验稿。

感谢父母对我的关心和照顾。

References

- [1] R. Bar-Yehuda and S. Even. "A Local-Ratio Theorem for Approximating the Weighted Vertex Cover Problem". In: Analysis and Design of Algorithms for Combinatorial Problems. 1985.
- [2] Avrim Blum. "New Approximation Algorithms for Graph Coloring". In: J. ACM (1994).
- [3] Henry A Kierstead. "On-line coloring k -colorable graphs". In: Israel Journal of Mathematics (1998).
- [4] Henry A Kierstead and William T Trotter. "An extremal problem in recursive combinatorics". In: Congressus Numerantium (1981).
- [5] Burkhard Monien and Ewald Speckenmeyer. "Ramsey Numbers and an Approximation Algorithm for the Vertex Cover Problem". In: Acta Inf. (1985).



- [6] S. Vishwanathan. “Randomized online graph coloring”. In: Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science. 1990.
- [7] Avi Wigderson. “Improving the Performance Guarantee for Approximate Graph Coloring”. In: J. ACM (1983).





浅谈信息学竞赛中的空间优化问题

浙江省乐清市知临中学 陈知轩

摘要

本文介绍了几种常见的数据处理模型, 以及针对这些模型的一些优化空间的技巧, 分析了它们的优化效果以及适用范围, 并且对于几道例题进行了具体分析。

1 引言

在信息学竞赛中, 题目通常对于时间复杂度有着较高要求, 有时甚至会采取牺牲空间换取时间更优的做法。而在部分题目中, 严格的复杂度要求同样考察选手设计算法的能力。本文针对几种常见的数据处理模型介绍了几种简单优化算法空间复杂度的思路, 并且对于几个经典的问题提供了具体优化方法。

2 模型 1 DAG 上的静态数据处理

在许多问题中, 需要处理的数据可以分为若干个块, 数据块之间存在着依赖关系, 而最终的答案仅与最后一个数据块有关。这样的模型可以描述为:

1. 有 n 个数据块, 第 i 个数据块记作 $data_i$, 其规模为 L_i 。
2. 有若干条依赖关系, 用有向边 (j, i) 表示计算 $data_i$ 依赖于 $data_j$, 这些有向边构成一个 DAG。此外, 每一个数据块还可能依赖于输入的数据。
3. 输入数据的存储消耗可以忽略, 且答案仅与 $data_n$ 有关。

对于这样的模型, 最简单的处理方式预先申请 n 个长度为 $\max\{L_i\}$ 的数组, 或者对于每一个数据块分别申请内存。

注意到当一个数据块的所有依赖项被处理完后, 便可以将其释放。可以按照某一个拓扑序 (如 $1, 2, \dots, n$) 依次处理每一个数据块, 在处理时动态申请内存, 在数据块不再有效后释放内存。对于一般情况, 这样的方法难以估计其空间复杂度, 且依赖于选取的拓扑序。此外, 动态申请和释放内存还会带来较大的常数消耗。

而对于某些特殊的情况, 存在着能在优化空间的同时保持时间常数的简单技巧。



2.1 特殊情况 1 滚动

滚动通常适用于满足以下条件的情况：

1. 每一个数据包的规模 L_i 相同，设为 L 。
2. 对于每一个 i ，其依赖的 j 均满足 $i - m \leq j < i$ ，其中 $m(m < n)$ 是一个常数。

此时，通过滚动可以将空间复杂度降为 $O(mL)$ 。

2.2 例题 1 球染色

2.2.1 题面描述

n 个球排成一排，一开始均为白色，将第 i 个染色的代价为 a_i 。求最小的染色代价使得任意相邻 m 个球中至少有两个球被染色。

2.2.2 数据规模与约定

$$n \leq 10^5, m \leq 10^3, c_i \leq 10^9。$$

2.2.3 分析

令 $f_{i,j}$ 表示最后一个被染色的球为 i ，上一个被染色的球为 $i - j$ 的最小代价，则 $f_{i,j} = \min\{f_{i-j,k} \mid j + k \leq m + 1\}$ 。

用简单的前缀和处理这个 dp，时间复杂度和空间复杂度均为 $O(nm)$ 。

将每一个 f_i 看成规模为 n 的数据块，容易发现在处理 f_i 时，只有 $f_j, j \geq i - m$ 的部分存在着有效的 dp 转移，因此可以使用滚动的方式优化。在具体实现时，可以将 f_i 存储在第 $i \bmod m$ 个数组中。

用类似方式的实现循环队列，可以优化 SPFA 等算法。

2.3 特殊情况 2 树形合并

树形合并通常指满足以下条件的情况：

1. 有向边构成一棵以某个块 u 为根的外向树。
2. 对于每一个数据块 $data_i$ ，其规模 L_i 为其所有子节点规模 L_j 之和。

维护树形合并的空间复杂度可以优化至 $O(L_u)$ 。



2.4 例题 2 树形背包

2.4.1 题面描述

给定一棵无根树，结点从 $1 \sim n$ 标号，每一个点有点权 a_i ，定义一个连通块的权为其中所有点点权的乘积。对于每一个 i ，求所有恰好包含 i 个点且经过 1 号点的连通块的权之和 $\bmod 10^9 + 7$ 。

2.4.2 数据规模与约定

$$n \leq 8000, 1 \leq a_i < 10^9 + 7.$$

2.4.3 分析

令 $f_{i,j}$ 表示连通块中深度最小的结点为 i ，且包含 j 个点的连通块的权值之和。每次对于一条边 (u,v) 进行合并， $f'_{u,i} = \sum f_{u,j} f_{v,i-j}$ 。

朴素地实现树形背包，时间和空间复杂度均为 $O(n^2)$ 。

2.4.4 优化

注意到数据 f_u 的规模为 u 节点子树大小，是典型的树形合并模型。

在每一次将 f_v 的信息合并给 f_u 之后， f_v 的信息都不再有效，如果将 f_u 的规模看成与其有关的节点个数，在任何时刻每一个节点最多与一个数据块有关。即在同一时刻最多有 n 个有效的 dp 值。

为了避免动态申请内存，可以将 f_u 的值存储在 f_u 目前已经被遍历的子节点对应的 dfs 序区间。合并 f_u, f_v 信息时，恰好对应 dfs 序上相邻的两段。

此外，归并排序也是一个典型的例子，但是其维护的数据恰好可以存储在对应的排序区间上，因此无需显式地采取上述方式进行优化。由于可能无法实现原地合并，这类方法还需要考虑合并时所需的临时空间（也为 $O(L_u)$ ）。

3 模型 2 带查询的静态数据的处理

一个静态数据处理问题，常常可以描述为以下模型：

1. 需要处理的数据满足上文提到的 **模型 1**，设采取 **模型 1** 的方式进行优化得到的空间复杂度为 $\text{process}(L_i)$ 。
2. 有 m 个询问，询问之间无互相影响。每一个询问操作可以分解为若干 $O(c)$ 个子问题，每一个子问题只涉及某一数据块的信息。



由于需要处理询问,无法像模型 1 一样在数据块无效后释放空间,达到理想的 $O(\text{process}(L_i))$ 空间。因而直接处理上述模型并且在线回答所有询问所需的空间复杂度为 $O(\sum L_i)$ 。

在原问题支持离线的情况下,针对 n, L_i, m, c 的量级,以及数据的具体特征(数据的递推关系、子问题间的依赖和决策关系),可以选择不同的方式进行空间复杂度的优化。下面介绍几种可以在尽量不改变算法本质(即尽量保持时间复杂度不变)的情况下,优化空间复杂度的技巧。

3.1 技巧 1 分离处理

在 c 量级较小,且子问题之间相对独立时,可以用以下方法将空间复杂度转化为 $O(\text{process}(L_i) + n + m \cdot c + k)$ 。

1. 离线所有询问的子问题,将所有 $O(mc)$ 个子问题分配到其所在数据块的对应的链表上。
2. 按照模型 1 中的方式处理每一个数据块,然后将当前层对应链表上的若干个询问取出,依次进行回答。

3.2 例题 3 ST 表

3.2.1 题面描述

给定 n 个数 $a_1, a_2, \dots, a_n$ 和 m 个查询 l_i, r_i , 每次求区间 $[l_i, r_i]$ 最大的数。

3.2.2 数据规模与约定

$$n \leq 10^6, m \leq 5 \cdot 10^6.$$

3.2.3 朴素算法

用经典的 Sparse Table 维护 RMQ 问题。时间复杂度为 $O(n \log n + m)$, 空间复杂度为 $O(n \log n)$ 。

3.2.4 优化

将倍增数组的 $L = \log n$ 层看成数据块,每一层的规模均为 $O(n)$ 。每一个询问可以分解为 $O(1)$ 个子问题。注意到:

1. 倍增问题的第 i 个数据块仅与第 $i-1$ 个块有关,因此用滚动的方式可以得到 $\text{process}(L_i) = n$ 。



2. 询问的子问题完全独立。

因此可以采用以上的优化方式，将空间复杂度优化至 $O(n + m)$ 。

RMQ 问题有时间和空间上更优的算法，但是与本文的优化方向无关，因此不作提及。

3.3 技巧 2 并行处理

分块处理基于询问的相对独立性，而当询问的子问题之间存在着依赖或者决策关系时，有时可以采用并行处理的方式。这样的询问的子问题通常满足：每一个子问题的参数基于于上一个子问题的结果进行决策得到。

这样的询问有一个最简单的例子：二分答案。并行处理二分答案的方式也常被称作一起二分，其目的常常是避免数据的可持久化操作。

3.4 例题 4 区间第 k 大

3.4.1 题面描述

给定 n 个数 a_i 和 m 个查询 l_i, r_i, k_i ，对于第 i 个查询回答区间 $[l_i, r_i]$ 中第 k_i 大的数。

3.4.2 数据规模与约定

$n, m \leq 10^6, 1 \leq a_i \leq n, k_i \leq r_i - l_i + 1$ ，认为 n, m 同阶。

3.4.3 算法 1

区间第 k 大是一个经典问题，核心思路是在前缀 r_i 和前缀 $l_i - 1$ 对应的值域线段树上进行二分，第 i 个数据块存储了前缀 $1 \sim i$ 的值域线段树，规模为 $O(n)$ ，无法暴力存储。而注意到 $data_i$ 与 $data_{i-1}$ 只有一条链上的 $O(\log n)$ 个值不同，因此可以通过共用信息的方式将总存储消耗压缩至 $O(n \log n)$ ，即可持久化线段树。

3.4.4 算法 2

根据并行优化的思路，考虑离线处理所有询问。将所有询问二分的当前状态用三个参数 (L_i, R_i, M_i) 描述，按照轮次一同处理。在二分过程的每一轮，需要计算区间 $[L_i, R_i]$ 中 $\leq M_i$ 的数的个数。

用朴素的二维数点操作处理一轮操作，然后根据结果决策下一轮的二分参数，在 $\log n$ 轮内可以完成所有询问。

时间复杂度为 $O(n \log^2 n)$ ，空间复杂度为 $O(n)$ 。



3.4.5 算法 3

注意到 **算法 2** 虽然通过并行处理降低了空间复杂度，但牺牲了时间复杂度。而本节的优化思路的一大核心是：不改变算法本质，尽量保持时间复杂度。

重新考虑 **算法 1** 的核心思路：在前缀 r_i 和前缀 $l_i - 1$ 对应的值域线段树上进行二分，根据子节点的权值决策下一轮的查询节点。考虑并行处理所有询问，可以用线段树节点位置 p_i 描述当前询问子问题的状态，每一轮的子问题涉及到 $l_i - 1$ 和 r_i 两个数据块上的一个元素。由于不需要进行可持久化操作，递推处理数据块的空间复杂度为 $\text{process}(L_i) = n$ ，时间复杂度为 $O(n \log n)$ ，而查询对应节点的权值并作出决策的时间为 $O(1)$ 。

以上算法和 **算法 1** 有着相同的本质，但其数据处理上的冗余导致了时间复杂度的提高。考虑进一步将数据分块，将线段树上第 i 层的节点看做一个数据块，则在询问的第 i 轮，只涉及每一个前缀的第 i 个块的数据。此时再递推处理每一个前缀的数据块，每次只需要修改特定的一个位置，故单层时间复杂度为 $O(n)$ 。

在实际实现时，可以预处理叶节点的编号，通过位运算找到其 i 级祖先的位置；也可以直接抛弃线段树原有的编号结构，直接通过位运算进行快速定位。

此外，线段树上二分也可以替换为树状数组上倍增。此时，可以将数据按照 lowbit 分块，并行二分的每一轮也只涉及一个特定的块。

3.5 技巧 3 压缩子问题

当 c 的量级较大时，采用分块处理的方式在压缩了数据存储的空间的同时，额外产生的存储询问子问题的空间消耗过大。此时可以观察分离子问题的过程，根据其特点，对于子问题的存储进行压缩。

3.6 例题 5 区间凸包

3.6.1 题面描述

给定 n 个点 (x_i, y_i) 。有 m 个查询，对于第 i 个询问回答 $\max\{k_i \cdot x_j + y_j \mid l_i \leq j \leq r_i\}$ 。

3.6.2 数据规模与约定

$n, m \leq 10^6, x_i, y_i, k_i \leq 10^9$ ，认为 n, m 同阶。

3.6.3 朴素算法

分治结合李超树可以用 $O(n)$ 空间解决此题，但由于其时间复杂度较劣，这里不作阐述。



用线段树维护凸包，归并预处理出线段树区间对应的凸包。每一个询问可以分解为 $O(\log n)$ 个线段树结点上的凸包查询，如果用二分查询凸包则时间复杂度为 $O(n \log^2 n)$ 。也可以离线将询问按照 k_i 排序，查询时在凸包上维护最优指针，时空复杂度均为 $O(n \log n)$ （实际上凸包点数的上限还与值域有关）。

3.6.4 优化

分析朴素算法：

1. 线段树上的信息可以描述为 $2n - 1$ 个数据块，递推处理这些数据块的过程与处理归并排序相同，容易得到 $\text{process}(L_i) = n$ 。

2. 询问的子问题之间独立，但是直接带入分块优化的方法会导致存储询问子问题所需的空间过大。

而将每一个询问分解为子问题的过程就是一次线段树上的区间查询操作，而线段树上查询操作在递归的每一个时刻只需要用 $O(1)$ 个参数即可记录。因此可以考虑用递归过程的参数来代替子问题的直接存储，从而实现压缩的效果。具体的，可以描述为：

1. 依次遍历线段树的每一个节点，同时维护递归至当前节点的询问集合。
2. 在当前节点，对于询问集中的每一个询问进行决策，判断其是否进入左右儿子节点、是否在当前节点进行查询。
3. 当结束了两个子节点的处理后，归并子节点的凸包，完成在这个节点的查询。
4. 回撤进入该节点时询问集合的变化。

这里的压缩是通过将询问和预处理的过程整合做到的，因此不改变原算法的本质，时间复杂度为 $O(n \log n)$ ，空间复杂度为 $O(n)$ 。

此外，用非递归操作维护区间查询（ZKW 线段树）还能得到更为简单的实现：

1. 预先将所有询问按照权值排序，处理出区间询问开始时所在的线段树叶节点。
2. 按照深度对于线段树分块，自下而上处理所有块。
3. 对于每一层，先将所有查询的节点向上移动一层，然后在当前层寻找所需的部分进行查询。
4. 归并当前层的凸包，回到第 3 步。

实际上，这种方法还可以支持修改（插入点）。

4 针对特殊序列问题的分块方法

对于序列上的区间问题，选取合理的分块方式有时可以减小空间消耗。不同于一般的分块算法，用于优化内存时的分块算法往往块长或者块数较小。

4.1 顶层分块

顶层分块类似一般的分块算法,将原始序列分成较少的块,块内为子问题,但是在优化内存时往往块数较少。其大致可以描述为以下模型:

1. 原序列长度为 n , 有 m 个区间操作。
2. 对于长度为 n 的序列处理全局操作的时间、空间复杂度为 $T(n), M(n)$ 。
3. 对于长度为 n 的序列处理区间操作的时间、空间复杂度为 $T'(n), M'(n)$ 。

直接处理的时空复杂度即 $T'(n), M'(n)$ 。

若将原序列均分为 S 段,则将原区间问题分解为 S 个全局操作和 $O(1)$ 个区间操作,时空复杂度为 $S \cdot T(\frac{n}{S}) + T'(\frac{n}{S})$ 。

每一个块作为独立的子问题,离线依次对于每一个块处理,因此空间复杂度为 $M(\frac{n}{S}) + M'(\frac{n}{S})$ 。

4.2 例题 5 区间凸包

4.2.1 解法 2

朴素算法可以归纳为 $T(n) = O(1), M(n) = O(n), T'(n) = O(\log n), M(n) = O(n \log n)$ 。

取 $S = \log n$, 则新的时间复杂度为 $S \cdot T(\frac{n}{S}) + T'(\frac{n}{S}) = \log n$, 与原算法相同; 而新的空间复杂度降低为 $M(\frac{n}{S}) + M'(\frac{n}{S}) = O(n)$ 。

4.3 底层分块

底层分块通常指以下模型:

1. 设原序列长度为 n 。先将原序列分成长度为 S 的块。
2. 处理出每一个块内信息合并后的结果,将原序列转化为长度为 $\frac{n}{S}$ 的新序列。
3. 在新序列上套用原先的数据结构。

在新序列上套用数据结构的空间占用缩小为 $\frac{1}{S}$, 每次处理单点或者区间操作时,需要进入 $O(1)$ 个区间暴力修改,即每次操作的时间复杂度为 $O(\log \frac{n}{S} + f(S))$, 其中 $f(S)$ 为暴力处理块的复杂度。

4.4 例题 6 动态 dp

4.4.1 题面描述

给定长度为 n 的序列 $a_1, a_2, \dots, a_n$, 求在其中选择若干个数,使得相邻两个被选择的数下标差不超过 k 。设一个方案的权值为所有被选择的数的积,求所有合法方案的权值和



$\text{mod } 10^9 + 7$ 。

维护 m 个修改，每次将 a_x 改为 y ，每次修改后输出新的答案。

4.4.2 数据规模与约定

$$n \leq 10^5, m \leq 10^4, k \leq 8, 1 \leq a_i, y < 10^9 + 7。$$

4.4.3 朴素算法

这是经典的矩阵维护动态 dp 模型。设 f_i 表示最后一个选择的数为 i ，则 $f_i = \sum_{i-k \leq j < i} f_j \cdot a_i$ 。在线段树上每一个节点 $[l, r]$ 维护一个矩阵 $mat_{i,j}$ 表示由 f_{l-i} 转移到 f_{r+1-j} 的系数。维护单点修改的时间为 $O(m \cdot k^3 \log n)$ ，预处理时间为 $O(nk^3)$ ；空间复杂度为 $O(n \cdot k^2)$ 。

4.4.4 优化

暴力重构块内信息需要做 k 次线性的 dp，复杂度为 $O(S \cdot k)$ 。取 $S = k^2 \log n$ 进行底层分块，则空间复杂度为 $O(n(1 + \frac{k^2}{S})) = O(n(1 + \frac{1}{\log n})) = O(n)$ ，时间复杂度为 $O(\frac{n}{S} \cdot S \cdot k + \frac{n}{S} k^3 + m(k^3 \log n + S \cdot k)) = O(nk + mk^3 \log n)$ 。在这道题中底层分块可以减少空间消耗以及预处理的时间，相应的增加单点修改的常数，在实际应用中 S 取小于 $k^2 \log n$ 可能更优。

此外，bitset 优化、高精度压位等方法也可以认为是特殊的底层分块。

5 针对 trie 树的优化

5.1 朴素的 trie 树模型

设一棵 trie 树由 n 个串 $s_1, s_2, \dots, s_n$ 构建，设字符集为 Σ ，串总长为 m 。

朴素实现的 trie 树包含 m 个结点，且每一个结点需要维护 $|\Sigma|$ 个后继结点，空间复杂度为 $O(m \cdot |\Sigma|)$ 。

5.2 链表 trie

当字符集较大时，对于每一个结点存储 $|\Sigma|$ 个后继往往存在着很大的冗余，此时可以考虑用链表代替数组存储后继节点。

对于所有的 trie 树节点，所有这样的后继只有 m 个，因此存储链表的空间复杂度为 $O(m)$ 。而此时寻找后继需要遍历链表，因此建立 trie 树的时间复杂度复杂度为 $O(m \cdot |\Sigma|)$ ，实际上常数非常小。

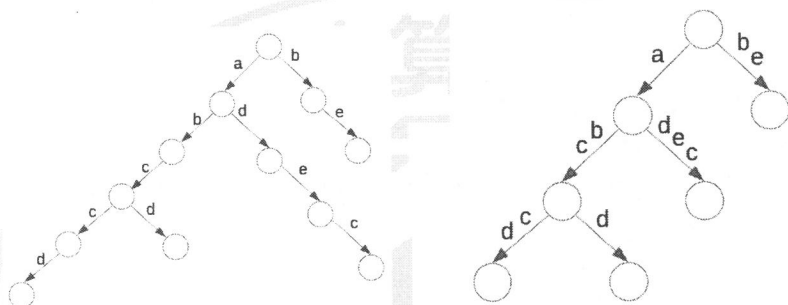
如果先对于所有 trie 进行字典序排序,再插入 trie 树时就只需要比较链表末的后继节点是否匹配,这样插入的时间复杂度为 $O(m \log n + m)$ (良好的实现可以到达 $O(m)$ 排序)。处理匹配时可以用类似的方式,对于每一个节点维护一个指针来完成均摊线性的匹配。

使用平衡树代替链表可以将查询的复杂度降至 $O(\log |\Sigma|)$,但相应地会带来更大的内存占用以及更高的实现难度,且在常见情况 ($|\Sigma| \leq 26$) 时难以带来更优的效率。

5.3 压缩 trie

压缩 trie 算法将普通 trie 树上的长链压缩成一个结点,保留原先的祖先关系,从而减少使得 trie 树的结点数量将低至 $O(n)$ 。

下图是对于 4 个串 abcd,adec,abccd,be 建立的普通 trie 和压缩 trie。



与普通 trie 一样,对于每一个压缩 trie 的结点,都需要维护每一个字符对应的后继结点,此外再为每一个结点维护其父边对应的压缩字符串。

在进行 trie 树插入和访问时,先通过开头的字符找到对应的后继结点,然后对于当前串和后继结点的串比较。如果出现不匹配则新建一个结点,将新插入的串和当前的结点作为其子结点。每一次插入一个串最多额外产生一个结点,因此总结点数 $\leq 2n - 1$ 。因此维护压缩 trie 的空间复杂度为 $O(m + n \cdot |\Sigma|)$ 。

用于处理子串问题的后缀树就是一种特殊的压缩 trie,也就是原串的所有后缀建立的 trie 树。由于这棵压缩 trie 上的每一段压缩串都是原串的子串,因此不需要额外使用 $O(m)$ ($m = n^2$) 的空间存储。

5.3.1 应用 动态开点线段树

基于值域,支持单点修改的动态开点线段树其本质是一棵字符集为 $\{0, 1\}$ 的 trie 树,每次插入的字符串实际上就是对应下标的二进制表示。对于 n 次单点修改,朴素的动态开点线段树空间复杂度为 $O(n \log v)$,其中 v 为值域。

在数值可以 $O(1)$ 存储的情况下,记录 trie 树的每一个结点对应的串只需要 $O(n)$ 的空间,用压缩 trie 进行存储的空间复杂度为 $O(n)$ 。同样的方法在处理线段树合并时也适用,但是



在实现时需要讨论更复杂的情况。

6 总结

本文介绍了对于静态数据处理以及查询、序列区间操作、trie 树等几类问题的优化空间复杂度的思路，并且在几个经典的例题中提供了具体的应用示例。

空间的优化在实际生活的应用中往往占据着非常重要的地位，而在算法竞赛中常常不作为考察点。希望本文可以激发读者对于算法的空间优化的兴趣，设计出更多优异的算法，在信息学竞赛的考题中引入空间优化的观念。

7 致谢

感谢中国计算机协会提供交流和学习的平台。

感谢国家级集训队杨耀良教练的指导。

感谢张超老师对我的支持与指导。

感谢南京外国语学校戴江齐同学、西安交通大学周品良同学为本文审稿。

感谢给予过我帮助的老师与同学们。

参考文献

[1] 刘汝佳,《算法竞赛入门经典》,清华大学出版社。

[2] Kurt Maly, *Compressed Tries*, 1976.