



中國計算機學會
CHINA COMPUTER FEDERATION

IOI2025中国国家集训队 论文集





中國計算機學會
CHINA COMPUTER FEDERATION

IOI2025中国国家集训队 论文集



2025 年国际信息学奥林匹克竞赛中国国家集训队论文集

目 录

浅谈连续段及相关问题	刘恒熙	(1)
OI 题目中凸性的证明方法	苟竞予	(22)
浅谈树上邻域问题的一种基于长链剖分的解法	刘海峰	(49)
转置原理在一类动态规划问题中的应用	徐骁扬	(61)
浅谈线性基在 OI 中的应用	陈昕阳	(73)
序列相关问题的一些思路总结	范斯喆	(101)
浅谈 Thue-Morse 序列的性质及其应用	郑 钧	(115)
浅谈矩阵乘法与归约	陈 诺	(139)
一些经典问题模小质数幂的做法	武 林	(150)
浅谈一类实现简易的动态树型结构信息维护方法	肖岱恩	(161)
浅谈一类图上邻域相关问题	焦思源	(169)
浅谈一类基于线段树分治的信息维护算法	黄佳旭	(179)
浅谈连续型随机变量的相关问题	周桓毅	(187)
《无限地狱》命题报告	陈旭磊	(207)
《数据结构》解题报告	朱屹帆	(213)
特殊等价关系下的字符串问题	章弥炫	(219)
浅谈无精度误差的计算几何	姚 熙	(228)
浅谈 OI 中的大容量背包问题	方心童	(243)
浅谈一类树上换根问题	赵海鲲	(251)
一类由数据结构解决的树上字符串问题	李静榕	(272)
浅谈一类提取有理分式系数的方法	黄建恒	(282)
浅谈张量及其在OI的运用	魏忠浩	(293)
浅谈一类线性同余不等式问题	杨鑫和	(314)

浅谈连续段及相关问题

宁波市镇海中学 刘恒熙

摘要

连续段是信息学竞赛中常见的研究对象。本文将与连续段有关的一系列问题转化为了统一的形式，探究了区间集的性质，给出了针对一类特殊区间集的算法，并用区间集上的算法解决了一部分问题。除此以外，本文通过归约，证明了一部分问题的困难性。

1 引言

信息学竞赛的问题中，常常出现“元素的值形成一个区间”“ $r - l = \max - \min$ ”等描述，这类问题通常与连续段有关。在完成 [CCPC Final 2021] Tree Partition 一题时，笔者发现存在更优做法，于是笔者思考了更加一般的情况，得到了本文中的一些结果。

笔者希望通过本文，让读者了解连续段问题的本质，熟悉连续段的一些性质，并且能利用区间集上的算法解决一些问题。

2 约定

如果没有特殊说明，出现的数都是整数。

对于 $l, r \in \mathbb{Z}, l \leq r$ ，区间 $[l, r]$ 表示 l, r 之间的整数构成的集合，即 $[l, r] = \{i \mid i \in \mathbb{Z}, l \leq i \leq r\}$ 。当 $l < r$ 时，称 $[l, r]$ 为非平凡区间。区间 $[l, r]$ 的长度为其中元素个数，即 $r - l + 1$ 。

对于一个排列 p ， p^{-1} 表示排列 p 的逆，即 $p_{p_i}^{-1} = i$ 。

对于一个排列 p ， $p_{[l, r]}$ 表示 $p_l, \dots, p_r$ 构成的集合，即 $p_{[l, r]} = \{p_i \mid i \in [l, r]\}$ 。

如果没有特殊说明，排列 p 的大小为 n ， p 的下标范围为 $[1, n]$ ，区间 $[l, r]$ 满足 $1 \leq l \leq r \leq n$ 。

用 U 表示被 $[1, n]$ 包含的区间构成的集合，即 $U = \{[l, r] \mid 1 \leq l \leq r \leq n\}$ 。

用 2^A 表示集合 A 的幂集，即 $2^A = \{B \mid B \subseteq A\}$ 。

3 连续段

3.1 引入

定义 3.1 (排列的连续段). 对于一个排列 p , 若区间 $[l, r]$ 满足 $p_{[l, r]} \in U$, 则称 $[l, r]$ 是 p 的连续段。

在信息学竞赛中, 常常有关于连续段及相关拓展的问题。可能的拓展有把排列的区间改为树的连通子树, 单个排列改为多个排列等。这些拓展通常都能用统一的形式表示。

3.2 统一形式

假设有 m 个结构和 n 个元素, 设 n 个元素构成的集合为 A ¹, 每个结构上包含 n 个元素, 并且定义了每个 A 的子集是否“连通”, 要求出关于在 m 个结构上都“连通”的 A 的子集的信息。

将在 m 个结构上都“连通”的 A 的子集看作广义的连续段。

在一个结构上定义 A 的每个子集是否“连通”等价于一个结构有一个包含所有“连通”子集的连通集 $C \subseteq 2^A$ 。

因此问题又可以表述为: 有 n 个元素, 设这些元素构成的集合为 A , 还有 m 个 2^A 的子集 $C_1, \dots, C_m$, 要求出关于 $\bigcap C_i$ 的信息。

在信息学竞赛的问题中, 常见的结构有:

- 排列 (链)²: 排列 p 的连通集通常定义为 $\{p_{[l, r]} \mid [l, r] \in U\}$, 也就是 p 的区间。
- 树: 树 $T = (V, E)$ ($V = A$) 的连通集通常定义为 $\{S \mid S \subseteq V, T[S] \text{ 是连通图}\}$ ³, 该定义等价于 $\{S \mid S \subseteq V, \{(u, v) \in E \mid u, v \in S\} = |S| - 1\}$ 。
- 图: 图 $G = (V, E)$ ($V = A$) 的连通集通常定义为 $\{S \mid S \subseteq V, G[S] \text{ 是连通图}\}$ 。

可能的结构组合有:

- 2 条链 (即排列的连续段)
- 更多条链
- 链, 树
- 链, 多棵树

¹为了方便, 设 n 个元素为 $1, 2, \dots, n$, 此时 $A = [1, n]$ 。

²排列 p 的连通集与链 $T = (V, E), V = [1, n], E = \{(p_i, p_{i+1}) \mid i \in [1, n-1]\}$ 的连通集相同, 因此可以看成链。

³ $G[S]$ 表示图 $G = (V, E)$ 点集为 S 的导出子图, 即 $G[S] = (S, \{(u, v) \in E \mid u, v \in S\})$ 。

- 链, 图
- 2 棵树

对于不含链的组合, 例如 2 棵树的组合, 计算连续段的数量就是相当困难的。

对于含有链的组合, 连续段数量不超过 $O(n^2)$, 相应的问题也相对容易。不妨将元素按照某条链上顺序将元素重标号为 $1, 2, \dots, n$ 。重新标号后, 所有连续段都是区间, 因此我们需要考察区间集的性质。

4 区间集

定义 4.1 (正规). 若由集合族⁴ S 满足 $S \subseteq U, [1, n] \in S, \forall i \in [1, n], \{i\} \in S$, 则称 S 正规。

定义 4.2 (I 类封闭). 若由正规集合族 S 满足 $\forall A, B \in S, A \cap B \neq \emptyset \implies A \cup B \in S$, 则称 S I 类封闭。

定义 4.3 (II 类封闭). 若由正规集合族 S 满足 $\forall A, B \in S, A \cap B \neq \emptyset \implies A \cap B \in S$, 则称 S II 类封闭。

定义 4.4 (III 类封闭). 若由正规集合族 S 满足 $\forall A, B \in S, A \cap B \neq \emptyset \wedge A \not\subseteq B \wedge B \not\subseteq A \implies A \setminus B \in S$, 则称 S III 类封闭。

1. 若干个正规集合族的交是正规集合族。
2. 若干个 I 类封闭集合族的交是 I 类封闭集合族。
3. 若干个 II 类封闭集合族的交是 II 类封闭集合族。
4. 若干个 III 类封闭集合族的交是 III 类封闭集合族。

上述四条结论可以由定义直接得到。

4.1 I, II 类封闭区间集

定义 4.5 (极小区间). 设区间集 S 是 I, II 类封闭⁵区间集。对于 $i \in [1, n-1]$, 若区间 $[l, r]$ 包含 $[i, i+1]$ 并属于 S , 且所有这样的区间都包含 $[l, r]$, 即 $[l, r] \supseteq [i, i+1], [l, r] \in S, \forall [l', r'] \in S, [l', r'] \supseteq [i, i+1] \implies [l', r'] \supseteq [l, r]$, 则称 $[l, r]$ 是 S 上包含 $[i, i+1]$ 的极小区间, 记作 $M_S(i) = [\text{ml}_S(i), \text{mr}_S(i)]$ 。

把 $n-1$ 个极小区间构成的序列称作 S 的极小区间序列, 记作 M_S 。

⁴集合族指以集合为元素的集合。

⁵指同时满足 I 类封闭和 II 类封闭。下文的 I, II, III 类封闭同理。

定理 4.1. 设区间集 S 是 I, II 类封闭区间集。对于所有 $i \in [1, n-1]$, 存在 S 上包含 $[i, i+1]$ 的极小区间。

证明. 对于 I, II 类封闭区间集 S 和 $i \in [1, n-1]$, 取 $[l, r] = \bigcap_{[x,y] \supseteq [i,i+1] \wedge [x,y] \in S} [x,y]$, 由于 $[1, n] \in S$, 等式右边至少有一个区间被求交, 显然 $[l, r]$ 满足 $[l, r] \supseteq [i, i+1]$ 和 $[l', r'] \supseteq [i, i+1] \implies [l', r'] \supseteq [l, r]$ 。只需说明 $[l, r] \in S$ 。对等式右边的求交依次应用 S II 类封闭的性质, 可得 $[l, r] \in S$ 。□

定理 4.2. 设区间集 S 是 I, II 类封闭区间集。对于非平凡区间 $[l, r]$ ($l < r$), $[l, r] \in S$ 的充分必要条件是 $\forall i \in [l, r-1], M_S(i) \subseteq [l, r]$ 。

证明. 设区间集 S 是 I, II 类封闭区间集, $[l, r]$ ($l < r$) 是非平凡区间。

充分性: 若 $\forall i \in [l, r-1], M_S(i) \subseteq [l, r]$, 有 $[l, r] = \bigcup_{i=l}^{r-1} M_S(i)$ 。对于 $i \in [l+1, r-1]$, 当取 $M_S(i)$ 和 $\bigcup_{j=i}^{r-1} M_S(j)$ 的并时, 有 $i \in M_S(i), i \in \bigcup_{j=i}^{r-1} M_S(j)$, 因此可以依次应用 S I 类封闭的性质, 得到 $[l, r] \in S$ 。

必要性: 若 $[l, r] \in S$, 根据极小区间的定义, 对于所有的 $i \in [l, r-1]$, 显然 $[l, r] \supseteq [i, i+1]$, 因此 $M_S(i) \subseteq [l, r]$ 。□

由上述定理可知, I, II 类封闭区间集可以由 $n-1$ 个极小区间确定。

4.2 I, II, III 类封闭区间集

引理 4.1. 设区间集 S 是 I, II, III 类封闭区间集。若 i, j ($1 \leq i < j \leq n-1$) 满足 $i+1 \leq ml_S(j) \leq mr_S(i) \leq j$, 则 $ml_S(j) = i+1, mr_S(i) = j$ 。

证明. 设区间集 S 是 I, II, III 类封闭区间集, i, j ($1 \leq i < j \leq n-1$) 满足 $i+1 < ml_S(j) \leq mr_S(i) \leq j$ 。

根据极小区间的定义, $[ml_S(i), mr_S(i)], [ml_S(j), mr_S(j)] \in S$ 。

因为 S 是 III 类封闭的, $[ml_S(i), mr_S(i)] \setminus [ml_S(j), mr_S(j)] = [ml_S(i), ml_S(j)-1] \in S$ 。

有 $[ml_S(i), ml_S(j)-1] \in S, [ml_S(i), ml_S(j)-1] \supseteq [i, i+1]$, 与 $[ml_S(i), mr_S(i)]$ 是包含 $[i, i+1]$ 的极小区间矛盾。

因此 $ml_S(j) = i+1$ 。同理, $mr_S(i) = j$ 。□

取 T 为所有满足包含 $[i, i+1]$ 的极小区间不被其他极小区间真包含的 i 构成的集合, 即 $T = \{i \in [1, n-1] \mid \neg(\exists j \in [1, n-1], M_S(i) \subsetneq M_S(j))\}$ 。

引理 4.2. 对于 $i \in [1, n-1]$, 若存在 $t \in T$, 使得 $M_S(i) \supseteq [t, t+1]$, 则 $i \in T$ 。

证明. 假设 $i \notin T$, 那么存在 $j \in [1, n-1]$, 使得 $M_S(i) \subsetneq M_S(j)$ 。

因为 $M_S(t)$ 是包含 $[t, t+1]$ 的极小区间, 且 $M_S(i) \supseteq [t, t+1]$, 所以 $M_S(t) \subseteq M_S(i)$ 。

又因为 $M_S(i) \subsetneq M_S(j)$, 有 $M_S(t) \subseteq M_S(i) \subsetneq M_S(j)$, 与 $t \in T$ 矛盾。□

引理 4.3. 对于所有 $i \in [1, n-1]$, 存在 T 中的元素 t , 使得 $M_S(t)$ 包含 $[i, i+1]$ 。

证明. 对于 $i \in [1, n-1]$, 取 $M_S(j)$ ($j \in [1, n-1]$) 中包含 $[i, i+1]$ 且长度最大的, 设其为 $M_S(t)$ 。

若 $M_S(t)$ 被 $M_S(u)$ 真包含, 则 $M_S(u)$ 包含 $[i, i+1]$ 且长度更大, 因此 $M_S(t)$ 不被其他极小区间真包含, 即 $t \in T$ 。□

引理 4.4. 若存在 $i, j \in T, i \neq j$ 使得 $M_S(i) = M_S(j)$, 则 $\forall t \in T, M_S(t) = [1, n]$ 。

证明. 设 $i, j \in T, i \neq j$ 满足 $M_S(i) = M_S(j)$ 。

若 $ml_S(i) \neq 1$, 取 $k = ml_S(i) - 1$ 。

- 若 $mr_S(k) \geq i+1$, 则 $M_S(k) \supseteq [i, i+1]$, 因为 $[ml_S(i), mr_S(i)]$ 是包含 $[i, i+1]$ 的极小区间, $M_S(k) \supseteq M_S(i)$, 又因为 $k \in M_S(k), k \notin M_S(i)$, 所以 $M_S(k) \supsetneq M_S(i)$, 与 $i \in T$ 矛盾。
- 若 $mr_S(k) \leq i$, 则 $k \leq ml_S(i) \leq mr_S(k) \leq i$, 根据引理 4.1, 有 $mr_S(k) = i$; 而 $k \leq ml_S(j) \leq mr_S(k) \leq j$, 根据引理 4.1, 有 $mr_S(k) = j$, 矛盾。

因此 $ml_S(i) = 1$ 。同理, $mr_S(i) = n$ 。

根据 T 的性质, $\forall t \in T, \neg(M_S(t) \subsetneq M_S(i) = [1, n])$, 即 $\forall t \in T, M_S(t) = [1, n]$ 。□

引理 4.5. 当 $M_S(t), t \in T$ 互不相同时, 不存在 $i, j \in T, i \neq j$ 使得 $M_S(i) \supseteq [j, j+1]$ 。

证明. 若存在 $i, j \in T, i \neq j$ 使得 $M_S(i) \supseteq [j, j+1]$, 根据极小区间的定义, 有 $M_S(i) \supseteq M_S(j)$ 。又因为 $M_S(i) \neq M_S(j)$, 所以 $M_S(i) \supsetneq M_S(j)$, 与 $j \in T$ 矛盾。故不存在 $i, j \in T, i \neq j$ 使得 $M_S(i) \supseteq [j, j+1]$ 。□

引理 4.6. 当 $M_S(t), t \in T$ 互不相同时, 设 $T = \{t_1, t_2, \dots, t_k\}, t_0 = 0 < t_1 < t_2 < \dots < t_k < t_{k+1} = n$, 有 $M_S(t_i) = [t_{i-1} + 1, t_{i+1}]$ 。

证明. 根据引理 4.5, 有 $M_S(t_i) \subseteq [t_{i-1} + 1, t_{i+1}]$ 。

只需证明 $ml_S(t_i) = \begin{cases} 1 & i = 1 \\ t_{i-1} + 1 & i \leq 2 \end{cases}$, 同理可得 $mr_S(t_i)$ 的部分。

若 $ml_S(t_1) > 1$, 则有 $[1, 2]$ 未被任何一个 $M_S(t_i)$ 包含, 与引理 4.3 矛盾, 因此 $ml_S(t_1) = 1$ 。

对于 $i \in [2, k]$, 若 $mr_S(t_{i-1}) < ml_S(t_i)$, 则 $[ml_S(t_i)-1, ml_S(t_i)]$ 未被任何一个 $M_S(t_i)$ 包含, 与引理 4.3 矛盾, 因此 $t_{i-1} + 1 \leq ml_S(t_i) \leq mr_S(t_{i-1}) \leq t_i$, 根据引理 4.1, 可得 $ml_S(t_i) = t_{i-1} + 1$ 。□

定理 4.3. 设区间集 S 是 I, II, III 类封闭区间集。

当 $n \geq 2$ 时, 区间 $[1, n]$ 总能被划分成 k 个子区间 $[l_1, r_1], [l_2, r_2], \dots, [l_k, r_k]$ ($k \geq 2, l_1 = 1, r_k = n, l_i \leq r_i, r_i + 1 = l_{i+1}$), 使得:

- 对于所有的 $i \in [1, k]$, 子区间 $[l_i, r_i] \in S$ 。

- 所有 S 中的元素要么被某个子区间 $[l_i, r_i]$ 包含，要么可以表示成若干个完整子区间的并，即 $S \subseteq \bigcup_{i=1}^k \{[l, r] \mid [l, r] \subseteq [l_i, r_i]\} \cup \{[l_i, r_j] \mid 1 \leq i \leq j \leq k\}$ 。
- 所有可以表示成若干个完整子区间的并的连续区间构成的集合满足以下两个条件之一：
 - 包含所有可以表示成若干个完整子区间的并的区间，即 $S \supseteq \{[l_i, r_j] \mid 1 \leq i \leq j \leq k\}$ 。将这类划分称作合划分。
 - 仅包含所有完整子区间与 $[1, n]$ ，且 $k \geq 3$ ⁶，即 $S \cap \{[l_i, r_j] \mid 1 \leq i \leq j \leq k\} = \{[l_i, r_i] \mid i \in [1, k]\} \cup [1, n], k \geq 3$ 。将这类划分称作析划分。

证明. 取 $T = \{t_1, t_2, \dots, t_k\} = \{i \in [1, n-1] \mid \neg(\exists j \in [1, n-1], M_S(i) \subseteq M_S(j))\}$, $t_0 = 0 < t_1 < t_2 < \dots < t_k < t_{k+1} = n$ 。

区间 $[1, n]$ 可以被划分为 $k+1$ 个子区间 $[t_0+1=1, t_1], [t_1+1, t_2], \dots, [t_k+1, t_{k+1}=n]$ 。

对于子区间 $[t_i+1, t_{i+1}]$ ，由引理 4.2，有 $\forall j \in [t_i+1, t_{i+1}-1], M_S(j) \subseteq [t_i+1, t_{i+1}]$ 。根据定理 4.2，可得 $[t_i+1, t_{i+1}] \in S$ 。

- 若 $M_S(t), t \in T$ 互不相同，根据引理 4.6，有 $M_S(t_i) = [t_{i-1}+1, t_{i+1}]$ ，根据定理 4.2，该划分满足 $S \supseteq \{[l_i, r_j] \mid 1 \leq i \leq j \leq k\}$ ，即该划分是合划分。
- 否则 $k \geq 2, \forall t \in T, M_S(t) = [1, n]$ ，根据定理 4.2，该划分满足 $S \cap \{[l_i, r_j] \mid 1 \leq i \leq j \leq k\} = \{[l_i, r_i] \mid i \in [1, k]\} \cup [1, n]$ ，即该划分是析划分。

以上两种情况都有 $M_S(t_i) \supseteq [t_{i-1}+1, t_{i+1}]$ 。设 $[l', r'] \in S \setminus \bigcup_{i=1}^k \{[l, r] \mid [l, r] \subseteq [l_i, r_i]\}, t_i + 1 \leq l' \leq t_{i+1} \leq t_j + 1 \leq r' \leq t_{j+1}$ ，根据定理 4.2， $[l', r'] \supseteq M_S(t_{i+1}), [l', r'] \supseteq M_S(t_j)$ ，即 $[l', r'] = [t_i+1, t_{j+1}]$ 。因此 $S \subseteq \bigcup_{i=1}^k \{[l, r] \mid [l, r] \subseteq [l_i, r_i]\} \cup \{[l_i, r_j] \mid 1 \leq i \leq j \leq k\}$ 。

综上所述，该划分满足条件。 $\square$

定理 4.4. 定理 4.3 中的划分方式唯一。

证明. 设区间集 S 上区间 $[1, n]$ 有两种划分方式，划分 i 为 $[l_{i,1}, r_{i,1}], [l_{i,2}, r_{i,2}], \dots, [l_{i,k_i}, r_{i,k_i}]$, ($i \in [1, 2], k_i \geq 2$)。

找到最小的 i ，使得 $[l_{1,i}, r_{1,i}] \neq [l_{2,i}, r_{2,i}]$ ，不妨设 $r_{1,i} < r_{2,i}$ 。

- 若划分 1 是合划分：
 - 若 $i = 1$ ，对于划分 2， $[r_{1,i}+1, n]$ 即不被某个子区间 $[l_{2,j}, r_{2,j}]$ 包含，也不能表示成若干个完整子区间的并，矛盾。

⁶若不要 $k \geq 3$ ，则 $k = 2$ 的划分可能同时满足合划分和析划分的要求。

- 若 $i \geq 2$ ，对于划分 2， $[1, r_{1,i}]$ 即不被某个子区间 $[l_{2,j}, r_{2,j}]$ 包含，也不能表示成若干个完整子区间的并，矛盾。
 - 若划分 1 是析划分， $[l_{2,i}, r_{2,i}]$ 即不被某个子区间 $[l_{1,i}, r_{1,i}]$ 包含，也不是 $[1, n]$ ，矛盾。
- 故原假设不成立。 $\square$

4.3 析合树

析合树由刘承奥在 WC2019 营员交流提出并给出线性构造算法，可以参见 [3]。这里的定义与原营员交流课件的析合树定义略有不同，但本质相同。

定义 4.6 (析合树). 区间集的析合树是含有以下三类节点的有根树：

- 叶子节点： (L, i) ⁷代表区间 $[i, i]$ 。
- 合点： (J, l, r) 代表区间 $[l, r]$ 。
- 析点： (C, l, r) 代表区间 $[l, r]$ 。

区间集 S 的析合树需要满足以下限制：

- 根节点代表区间 $[1, n]$ 。
- 所有节点代表的区间属于区间集 S 。
- 非叶节点至少有 2 个子节点，并且这些子节点代表的区间构成该节点代表的区间的划分。特别地，析点至少有 3 个子节点。
- 对于非叶节点 (c, u, v) ($c \in \{J, C\}$)，设其子节点代表的区间分别为 $[l_1, r_1], [l_2, r_2], \dots, [l_k, r_k]$ ($k \geq 2, l_1 = u, r_k = v, l_i \leq r_i, r_i + 1 = l_{i+1}$)。 S 中所有在 $[u, v]$ 内的区间要么被某个子区间 $[l_i, r_i]$ 包含，要么可以表示成若干个完整子区间的并，即 $S \cap \{[l, r] \mid [l, r] \subseteq [u, v]\} \subseteq \bigcup_{i=1}^k \{[l, r] \mid [l, r] \subseteq [l_i, r_i]\} \cup \{[l_i, r_j] \mid 1 \leq i \leq j \leq k\}$ 。
- 对于合点 (J, u, v) ，设其子节点代表的区间分别为 $[l_1, r_1], [l_2, r_2], \dots, [l_k, r_k]$ ($k \geq 2$)。 S 包含所有可以表示成若干个完整子区间的并的区间，即 $S \supseteq \{[l_i, r_j] \mid 1 \leq i \leq j \leq k\}$ 。
- 对于析点 (C, u, v) ，设其子节点代表的区间分别为 $[l_1, r_1], [l_2, r_2], \dots, [l_k, r_k]$ ($k \geq 3$)。 S 中所有可以表示成若干个完整子区间的并的区间仅有 $[l_1, r_1], [l_2, r_2], \dots, [l_k, r_k], [u, v]$ ，即 $S \cap \{[l_i, r_j] \mid 1 \leq i \leq j \leq k\} = \{[l_i, r_i] \mid i \in [1, k]\} \cup [u, v]$ 。

⁷L 代表 leaf, J 代表 join, C 代表 cut。

定理 4.5. 对于 I, II, III 类封闭区间集 S , S 的析合树存在且唯一。

证明. 当 $n = 1$ 时, 仅包含一个叶子节点 $(L, 1)$ 的析合树满足条件, 并且显然不存在其他析合树满足条件。

假设对于 $n \leq N$ ($N \geq 1$), 结论成立。当 $n = N + 1 \geq 2$ 时, 按照定理 4.3 描述的方式, 把区间 $[1, n]$ 划分为 k 个子区间 $[l_1, r_1], [l_2, r_2], \dots, [l_k, r_k]$ ($k \geq 2, l_1 = 1, r_k = n, l_i \leq r_i, r_i + 1 = l_{i+1}$)。容易验证对于所有 $i \in [1, k]$, $S \cap \{[l, r] \mid [l, r] \subseteq [l_i, r_i]\}$ 在每个元素都进行平移后满足 I, II, III 类封闭, 根据归纳假设, k 个子集在每个元素都进行平移后均存在唯一的析合树。

- 若 S 包含所有可以表示成若干个完整子区间的并的区间, 建立合点 $(J, 1, n)$ 。
- 否则 S 中所有可以表示成若干个完整子区间的并的区间仅有 $[l_1, r_1], [l_2, r_2], \dots, [l_k, r_k], [1, n]$, 建立析点 $(C, 1, n)$ 。

将 k 个子集的析合树的每个节点平移回原区间后的根节点作为新建节点的 k 个子节点, 容易验证这样构造出的析合树满足条件, 故 S 的析合树存在。

当 $n = N + 1 \geq 2$ 时, 析合树的根节点的若干个子节点代表的区间构成定理 4.3 描述的划分方式, 根据定理 4.4, 划分方式唯一, 因此对于 S 的所有析合树, 根节点的类型和子节点代表的区间都相同。根据归纳假设, 每个子树都唯一, 故 S 的析合树唯一。 $\square$

定理 4.6. 对于区间集 S , S 的析合树存在的充分必要条件是 S I, II, III 类封闭。

证明. 充分性: 已由定理 4.5 给出。

必要性:

设区间集 S 存在析合树。

由于 S 的析合树中一定存在 n 个叶子节点 (L, i) ($i \in [1, n]$) 和代表 $[1, n]$ 的根节点, $[1, n] \in S, \forall i \in [1, n], \{i\} \in S$, 即 S 正规。

对于两个交集不为空的区间 $[l_1, r_1], [l_2, r_2]$, 设 u 为代表的区间包含 $[l_1, r_1]$ 且深度最大的节点, v 为代表的区间包含 $[l_2, r_2]$ 且深度最大的节点。

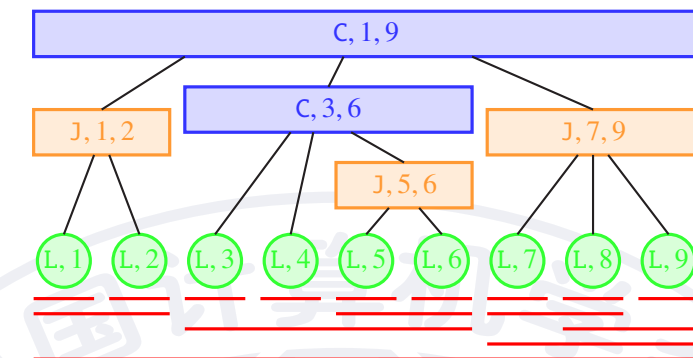
取 $i \in [l_1, r_1] \cap [l_2, r_2]$ 。因为 u, v 都是 (L, i) 的祖先, 所以 u, v 必为祖先后代关系。不妨设 $u \neq v$ 时 u 是 v 的祖先。

- 若 $u = v$, 根据 u 的类型分类讨论。
 - 若 u 是叶子节点或析点, $[l_1, r_1], [l_2, r_2]$ 都是 u 代表的区间, $[l_1, r_1] \cup [l_2, r_2] = [l_1, r_1] \cap [l_2, r_2] = [l_1, r_1] = [l_2, r_2] \in S$ 。
 - 若 u 是合点, $[l_1, r_1] \cap [l_2, r_2], [l_1, r_1] \cup [l_2, r_2]$ 都是 u 代表的区间被划分成的若干个子区间的并, 根据合点的性质, $[l_1, r_1] \cap [l_2, r_2], [l_1, r_1] \cup [l_2, r_2] \in S$ 。
- 若 u 是 v 的祖先, $[l_1, r_1] \cup [l_2, r_2] = [l_1, r_1] \in S$ 。

因此 S I, II 类封闭。

若 $[l_1, r_1], [l_2, r_2]$ 互不包含, $u = v$ 且 u 是合点, $[l_1, r_1] \setminus [l_2, r_2]$ 是 u 代表的区间被划分成的若干个子区间的并, 根据合点的性质, $[l_1, r_1] \setminus [l_2, r_2] \in S$ 。

因此 S III 类封闭。 $\square$



$$S = \{[1, 9], [1, 2], [3, 6], [5, 6], [7, 9], [7, 8], [8, 9]\} \cup \{[i, i] \mid i \in [1, 9]\}$$

图 1: 区间集 S 的析合树

4.4 析合树的构造

在实际问题中, 区间集的极小区间序列通常不易快速求出, 因此我们引入伪极小区间序列。

定义 4.7 (伪极小区间序列). 区间序列 $([l_1, r_1], \dots, [l_{n-1}, r_{n-1}])$ 是 I, II 类封闭区间集 S 的伪极小区间序列当且仅当 $\forall i \in [1, n-1], [l_i, r_i] \supseteq [i, i+1], \forall 1 \leq u \leq v \leq n, [u, v] \in S \iff (\forall i \in [u, v-1], [l_i, r_i] \subseteq [u, v])$ 。

换句话说, 伪极小区间序列具有定理 4.2 描述的极小区间的性质。

4.4.1 朴素算法

给定 I, II, III 类封闭区间集 S 的伪极小区间序列, 可以用贪心合并的方法构造 S 的析合树。

Algorithm 1 Naive Algorithm for Constructing Cut-Join Tree from a Pseudo Minimal Interval Sequence

Require: $([l_1, r_1], \dots, [l_{n-1}, r_{n-1}])$ is a pseudo minimal interval sequence of S

Ensure: Stack s eventually contains the root node of the cut-join tree

```

1:  $s \leftarrow$  empty stack
2: for  $i = 1$  to  $n$  do
3:    $u \leftarrow \text{Node}(L, i, i, \emptyset)$  {denotes type, left, right, child, respectively}
4:    $\text{sufmin} \leftarrow i$ 
5:    $\text{sufmax} \leftarrow i$ 
6:    $m \leftarrow |s|$ 
7:   for  $j = m$  to  $1$  do
8:      $\text{sufmin} \leftarrow \min\{\text{sufmin}, l_{s_j.\text{right}}\}$ 
9:      $\text{sufmax} \leftarrow \max\{\text{sufmax}, r_{s_j.\text{right}}\}$ 
10:    if  $\text{sufmin} \geq s_j.\text{left}$  and  $\text{sufmax} \leq i$  then
11:      if  $j = |s|$  then
12:        if  $s_j.\text{type} = J$  and  $\text{sufmin} \geq s_j.\text{child}_{|s_j.\text{child}|}.\text{left}$  then
13:           $s_j.\text{child}.\text{append}(u)$ 
14:           $s_j.\text{right} \leftarrow i$ 
15:           $u \leftarrow s_j$ 
16:        else
17:           $v \leftarrow \text{Node}(J, s_j.\text{left}, i, (s_j, u))$ 
18:           $u \leftarrow v$ 
19:        end if
20:      else
21:         $v \leftarrow \text{Node}(C, s_j.\text{left}, i, (s_j, s_{j+1}, \dots, s_{|s|}, u))$ 
22:         $u \leftarrow v$ 
23:      end if
24:      while  $|s| \geq j$  do
25:         $s.\text{pop}()$ 
26:      end while
27:    end if
28:  end for
29:   $s.\text{push}(u)$ 
30: end for

```

可以发现该算法新建的节点代表的区间总在 S 中。对于 S 的析合树中的一个非叶节点，该算法不会将不同子树的节点合并，并且能正确地建立合点与析点，因此该算法总能给出正确的结果。

4.4.2 线性算法

由于析合树的节点总数是 $O(n)$ 的，朴素算法的瓶颈是寻找新建的点。

朴素算法中，若 $\text{sufmax} > i$ ，则对于当前的 i 不可能再新建节点，因此可以直接退出内层循环。

若 $l_{s_j.\text{right}} < s_j.\text{left}$ ，则 s_j 不可能再作为某次新建节点的左端点，因此之后可以不再判断。利用上述优化思想，可以得到线性算法。

Algorithm 2 Linear-Time Algorithm for Constructing Cut-Join Tree from a Pseudo Minimal Interval Sequence

Require: $([l_1, r_1], \dots, [l_{n-1}, r_{n-1}])$ is a pseudo minimal interval sequence of S

Ensure: Stack s eventually contains the root node of the cut-join tree

```

1:  $l_n \leftarrow 1$ 
2:  $s \leftarrow$  empty stack
3:  $t \leftarrow$  empty stack
4: for  $i = 1$  to  $n$  do
5:    $u \leftarrow \text{Node}(L, i, i, \emptyset)$ 
6:   while  $t$  is not empty and  $r_{t.\text{top}()} \leq i$  do
7:      $j \leftarrow |s|$ 
8:     while  $s_j.\text{left} > t.\text{top}()$  do
9:        $j \leftarrow j - 1$ 
10:    end while
11:    if  $j = |s|$  then
12:      if  $s_j.\text{type} = J$  and  $l_{s_j.\text{right}} \geq s_j.\text{child}_{|s_j.\text{child}|}.\text{left}$  then
13:         $s_j.\text{child}.\text{append}(u)$ 
14:         $s_j.\text{right} \leftarrow i$ 
15:         $u \leftarrow s_j$ 
16:      else
17:         $v \leftarrow \text{Node}(J, s_j.\text{left}, i, (s_j, u))$ 
18:         $u \leftarrow v$ 
19:      end if
20:    else
21:       $v \leftarrow \text{Node}(C, s_j.\text{left}, i, (s_j, s_{j+1}, \dots, s_{|s|}, u))$ 
22:       $u \leftarrow v$ 
23:    end if
24:    while  $|s| \geq j$  do
25:       $s.\text{pop}()$ 
26:    end while
27:     $t.\text{pop}()$ 
28:  end while

```

```

29:   s.push(u)
30:   t.push(u.left)
31:   while t.top() > l_i do
32:     r_i ← max{r_i, r_{t.top()}}
33:     t.pop()
34:   end while
35:   r_{t.top()} ← max{r_{t.top()}, r_i}
36: end for

```

可以验证上述线性算法与朴素算法等价，因此该算法总能给出正确的结果。

4.5 求右端点对应的左端点处信息之和

给定区间集 S 和 n 个信息 $a_1, a_2, \dots, a_n$ ，信息的合并满足半群的性质⁸，需要对每个 $r \in [1, n]$ 求出 $\prod_{l \in [1, r], [l, r] \in S} a_l$ 。

对于 I, II, III 类封闭的区间集，可以建立析合树后通过一次 DFS 解决。

对于 I, II 类封闭的区间集，若信息的合并满足群的性质⁹，存在线性的算法解决该问题。

Algorithm 3 Linear-Time Algorithm for Finding the Product of the Information at the Left Endpoint Corresponding to the Right Endpoint

Require: $([l_1, r_1], \dots, [l_{n-1}, r_{n-1}])$ is a pseudo minimal interval sequence of S , the multiplication of a_i satisfies the properties of a group

Ensure: $b_i = \prod_{j \in [1, i], [i, j] \in S} a_j$

```

1: for i = 1 to n - 1 do
2:   c_i ← 0
3: end for
4: t ← empty stack
5: for i = n - 1 to 1 do
6:   t.push(i)
7:   while t.top() < r_i - 1 do
8:     c_{t.top()} ← i
9:     c.pop()
10:  end while
11: end for
12: s ← empty stack
13: s.push(0)

```

⁸即封闭性、结合律。将信息的合并记作 $\times$ 。

⁹即除半群的性质外，还要求存在单位元，每个元素都有逆元。将单位元记作 1， a 的逆元记作 a^{-1} 。

```

14: p_0 = 1
15: b_1 ← a_1
16: for i = 1 to n - 1 do
17:   p_i ← p_{s.top()} × a_i
18:   s.push(i)
19:   while s.top() > l_i do
20:     ERASE(s.top())
21:     s.pop()
22:   end while
23:   v ← QUERY(c_i)
24:   b_{i+1} ← p_v^{-1} × p_{s.top()} × a_{i+1}
25: end for

```

解释 该算法中的 c_i 是最大的 $k \in [1, i]$ ，使得 $r_k > i + 1$ （若不存在这样的 k ，则 $c_i = 0$ ），伪极小区间序列中 r 的限制转化为 $j > c_i$ ，而 l 的限制转化为了弹出 s 顶端的元素。该算法需要一个数据结构：

- 维护长度为一个 $n + 1$ 的 01 串 s ，初始全为 1。
- ERASE(x): 把 s_x 改成 0。
- QUERY(x): 查询最大的 y ，使得 $y \leq x, s_y = 1$ 。

数据结构的实现 在 Word RAM 模型下，可以使用压位并查集实现该数据结构。大致方法是将串 s 每 w 位分为一块，块内用位运算 $O(1)$ 处理，块间使用路径压缩、启发式合并的并查集处理。根据 [2]，复杂度为 $O(n\alpha(n, n/w)) = O(n)$ 。

上述算法与通过析合树解决该问题，都是在线的，即可以依次求出 $r = 1, 2, \dots, n$ 的答案，在求 $r = r_0$ 的答案时，不需要知道 $a_{r_0+1}, \dots, a_n$ 。

5 对连续段的进一步分析

链（排列）的连通集是正规、I 类封闭、II 类封闭、III 类封闭的。

树的连通集是正规、I 类封闭、II 类封闭的。

图的连通集是正规、I 类封闭的。

由于若干个满足正规、I 类封闭、II 类封闭、III 类封闭中的某个性质的集合族的交仍满足该性质，可以直接得到一些结构组合的性质。

5.1 2 条链

5.1.1 性质

将一条链上的元素依次重标号为 $1, 2, \dots, n$, 可得 2 条链的连续段等价于排列的连续段。类似地, 含有链的组合都可以进行类似的转化。

2 条链的连续段集是正规、I 类封闭、II 类封闭、III 类封闭的, 这意味着可以使用析合树来刻画排列的连续段。

排列 p 的连续段集的一个伪极小区间序列是 $([\min_{j \in A_i} p_j^{-1}, \max_{j \in A_i} p_j^{-1}])_{i \in [1, n-1]}$, 其中 $A_i = [\min\{p_i, p_{i+1}\}, \max\{p_i, p_{i+1}\}]$ 。

5.1.2 复杂性

定理 5.1. 对于一棵析合树, 若其所有析点的子节点个数都至少为 4, 则存在一个排列 p , 使得 p 的连续段集的析合树为该析合树。

定义 $\text{combine}(p, q_1, q_2, \dots, q_{|p|})$ 为给每个 q_i 整体加上一个数并按顺序拼接, 且 q_i 间的相对大小与 p 匹配的唯一排列。例如 $\text{combine}((1, 3, 2), (1, 2), (2, 1), (1)) = (1, 2, 5, 4, 3)$ 。

定义 $\text{flip}(p) = (|p| + 1 - p_i)_{i \in [1, |p|]}$ 。例如 $\text{flip}((1, 4, 2, 3)) = (4, 1, 3, 2)$ 。

对于一棵满足条件析合树, 可按照如下方式构造 p :

- 若根是叶子节点, 令 $p = (1)$ 。
- 否则设各个子树按照此方式构造出的排列为 $p_1, \dots, p_k$ 。

– 若根是合点, 令

$$p = \text{combine}((1, \dots, k), \text{flip}(p_1), \dots, \text{flip}(p_k)).$$

– 若根是析点, 令

$$p = \text{combine} \left(\begin{cases} (2, 4, \dots, k, 1, 3, \dots, k-1) & 2 \mid k \\ (2, 4, \dots, k-1, 1, k, 3, \dots, k-2) & 2 \nmid k \end{cases}, \text{flip}(p_1), \dots, \text{flip}(p_k) \right).$$

5.2 更多条链

5.2.1 性质

定义 5.1 (多个排列的连续段). 对于由 m 个排列构成的序列 $p = (p_1, \dots, p_m)$, 若区间 $[l, r]$ 是所有 p_i 的连续段, 则称 $[l, r]$ 是 p 的连续段。

k ($k \geq 3$) 条链的连续段等价于 $k-1$ 个排列的连续段。

k 条链的连续段集是正规、I 类封闭、II 类封闭、III 类封闭的, 这意味着仍然可以使用析合树来刻画多个排列的连续段。

排列构成的序列 p 的连续段集的一个伪极小区间序列是其中各个排列 p_i 的伪极小区间序列对应位取并得到的序列。

5.2.2 复杂性

定理 5.2. 对于任意一棵析合树, 存在 2 个排列 p_1, p_2 , 使得 (p_1, p_2) 的连续段集的析合树恰为该析合树。

由于所有 I, II, III 类封闭区间集都对应唯一一棵析合树, 该定理事实上指出了 2 个排列的连续段集可能为所有的 I, II, III 类封闭区间集。

对于一棵满足条件析合树, 可按照如下方式构造排列对 (p, q) :

- 若根是叶子节点, 令 $(p, q) = ((1), (1))$ 。
- 否则设各个子树按照此方式构造出的排列对为 $(p_1, q_1), \dots, (p_k, q_k)$ 。

– 若根是合点, 令

$$p = \text{combine}((1, \dots, k), \text{flip}(p_1), \dots, \text{flip}(p_k)),$$

$$q = \text{combine}((1, \dots, k), \text{flip}(q_1), \dots, \text{flip}(q_k)).$$

– 若根是析点, 令

$$p = \text{combine} \left(\begin{cases} (2, 3, 1) & k = 3 \\ (2, 4, \dots, k, 1, 3, \dots, k-1) & k \geq 4, 2 \mid k \\ (2, 4, \dots, k-1, 1, k, 3, \dots, k-2) & k \geq 4, 2 \nmid k \end{cases}, \text{flip}(p_1), \dots, \text{flip}(p_k) \right),$$

$$q = \text{combine} \left(\begin{cases} (3, 1, 2) & k = 3 \\ (2, 4, \dots, k, 1, 3, \dots, k-1) & k \geq 4, 2 \mid k \\ (2, 4, \dots, k-1, 1, k, 3, \dots, k-2) & k \geq 4, 2 \nmid k \end{cases}, \text{flip}(q_1), \dots, \text{flip}(q_k) \right).$$

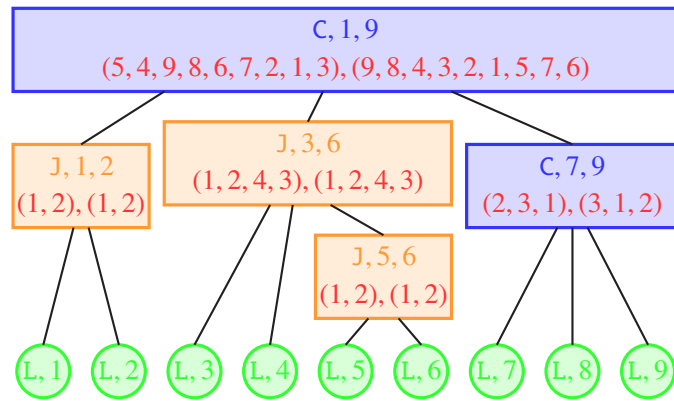


图 2: 对给定析合树构造 2 个排列

5.3 链，树

5.3.1 性质

定义 5.2 (树的连续段). 对于一棵树 $T = (V, E), V = [1, n]$, 若区间 $[l, r]$ 满足树 T 点集为 $[l, r]$ 的导出子图是连通图, 则称 $[l, r]$ 是 T 的连续段。

“链，树”组合的连续段等价于树的连续段。

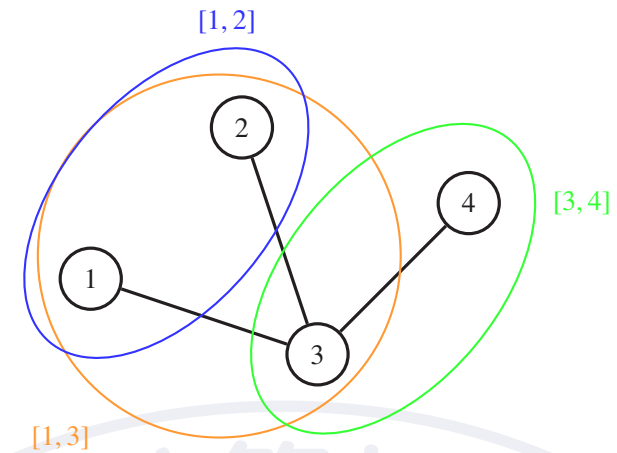
“链，树”组合的连续段集是正规、I 类封闭、II 类封闭的, 这意味着可以使用 $n-1$ 个极小区间来刻画树的连续段。

树的连续段集的一个伪极小区间序列是 $(\min \text{path}(i, i+1), \max \text{path}(i, i+1))_{i \in [1, n-1]}$, 其中 $\text{path}(i, i+1)$ 表示树上 i 到 $i+1$ 的路径的点集。

5.3.2 复杂性

树的连续段集不是 III 类封闭的。

考虑图 $T = (V, E), V = \{1, 2, 3, 4\}, E = \{(1, 3), (2, 3), (3, 4)\}$, 区间 $[1, 3]$ 和 $[3, 4]$ 都是其连续段, 而 $[1, 3] \setminus [3, 4] = [1, 2]$ 不是其连续段。



$[1, 3], [3, 4]$ 都是树 T 的连续段, 而 $[1, 2]$ 不是。

图 3: 树 $T = (\{1, 2, 3, 4\}, \{(1, 3), (2, 3), (3, 4)\})$

5.4 链，多棵树

5.4.1 性质

定义 5.3 (多棵树的连续段). 对于由 m 棵树构成的序列 $T = (T_1, \dots, T_m), V(T_i) = [1, n]$, 若区间 $[l, r]$ 是所有 T_i 的连续段, 则称 $[l, r]$ 是 T 的连续段。

“链，多棵树”组合的连续段等价于多棵树的连续段。

“链，多棵树”组合的连续段集是正规、I 类封闭、II 类封闭的, 这意味着可以使用 $n-1$ 个极小区间来刻画多棵树的连续段。

树构成的序列 T 的连续段集的一个伪极小区间序列是其中各个树 T_i 的伪极小区间序列对应位取并得到的序列。

5.4.2 复杂性

对于由 $n-1$ 个区间构成的序列 $A = (A_1, \dots, A_{n-1}), A_i = [u_i, v_i]$, 若对于所有 $i \in [1, n-1]$, 都有 $A_i \supseteq [i, i+1], \forall j \in [u_i, v_i-1], A_j \subseteq A_i$, 则称 A 是合法的。

由极小区间的定义及引理 4.2 可知, I, II 类封闭区间集的极小区间序列 M_S 是合法的。

定理 5.3. 对于任意一个由 $n-1$ 个区间构成的合法序列 A , 都存在 2 棵树 T_1, T_2 , 使得 (T_1, T_2) 的连续段集合的极小区间序列恰为 A 。

由于所有 I, II 类封闭区间集都对应一个极小区间序列, 而极小区间序列都是合法的, 该定理事实上指出了 2 棵树的连续段集可能为所有的 I, II 类封闭区间集。

对于一个极小区间序列 $([l_1, r_1], \dots, [l_{n-1}, r_{n-1}])$ ，令

$$V = [1, n], E_1 = \{(l_i, i+1) \mid i \in [1, n-1]\}, E_2 = \{(r_i, i) \mid i \in [1, n-1]\}, T_1 = (V, E_1), T_2 = (V, E_2).$$

可以验证 (T_1, T_2) 的连续段集合的极小区间序列恰为 $([l_1, r_1], \dots, [l_{n-1}, r_{n-1}])$ 。

5.5 链，图

5.5.1 性质

定义 5.4 (图的连续段). 对于一张图 $G = (V, E)$, $V = [1, n]$ ，若区间 $[l, r]$ 满足图 G 点集为 $[l, r]$ 的导出子图是连通图，则称 $[l, r]$ 是 G 的连续段。

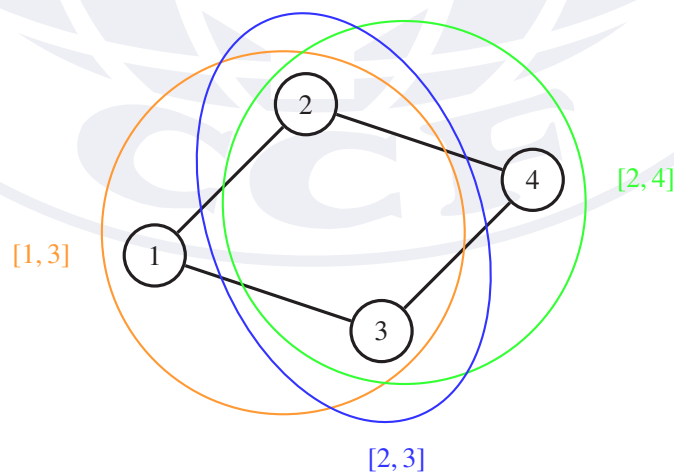
“链，图”组合的连续段等价于图的连续段。

“链，图”组合的连续段集是正规、I 类封闭的。

5.5.2 复杂性

图的连续段集不是 II 类封闭的。

考虑图 $G = (V, E)$, $V = \{1, 2, 3, 4\}$, $E = \{(1, 2), (2, 4), (4, 3), (3, 1)\}$ ，区间 $[1, 3]$ 和 $[2, 4]$ 都是其连续段，而 $[1, 3] \cap [2, 4] = [2, 3]$ 不是其连续段。



$[1, 3], [2, 4]$ 都是图 G 的连续段，而 $[2, 3]$ 不是。

图 4: 图 $G = (\{1, 2, 3, 4\}, \{(1, 2), (2, 4), (4, 3), (3, 1)\})$

5.6 2 棵树

5.6.1 性质

2 棵树的连续段集是正规、I 类封闭、II 类封闭的。但由于没有链的存在，不能重标号元素，使得连续段为区间。

5.6.2 复杂性

对 2 棵树的连续段计数是 #P-complete 的。

根据 [1]，对给定偏序集上的反链计数是 #P-complete 的。因此只需将反链计数多项式时间归约到 2 棵树的连续段计数，即可证明对 2 棵树的连续段计数是 #P-complete 的。

设给定偏序集为 $(S, \leq)$ ，为了方便起见，设 $S = \{1, 2, \dots, n\}$ 。

令 $\{x_{i,1}, x_{i,2}, \dots, x_{i,k_i}\} = \{j \in S \mid i \leq j, i \neq j\}$ 。

构造点集

$$V = \{a\} \cup \{b_i \mid i \in S\} \cup \{c_i \mid i \in S\} \cup \{d_i \mid i \in S\} \cup \{e_{i,j} \mid i, j \in S, i \neq j, i \leq j\}.$$

令 $(h_{i,1}, h_{i,2}, \dots, h_{i,k_i+4}) = (a, b_i, d_i, e_{i,x_{i,1}}, e_{i,x_{i,2}}, \dots, e_{i,x_{i,k_i}}, c_i)$ 。

构造边集

$$E_1 = \bigcup_{i \in S} \{(h_{i,j}, h_{i,j+1}) \mid j \in [1, k_i + 3]\},$$

$$E_2 = \bigcup_{i \in S} \{(a, d_i), (a, c_i), (c_i, b_i)\} \cup \bigcup_{i, j \in S, i \neq j, i \leq j} \{(e_{i,j}, b_j)\}.$$

2 棵树 $T_1 = (V, E_1), T_2 = (V, E_2)$ 的连续段数量减去 $|V|$ 即为给定偏序集 $(S, \leq)$ 的反链数量。

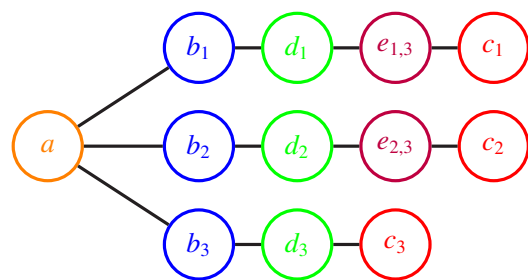
这是因为所有非平凡连续段¹⁰与反链存在一一对应关系。

连续段的形式 所有非平凡连续段都能表示成 $a \cup \{b_i \mid i \in A\} \cup \{c_i \mid i \in A\} \cup \{d_i \mid i \in A\} \cup \{e_{i,j} \mid i \in A, j \in S, i \neq j, i \leq j\}$ 的形式，其中 $\emptyset \neq A \subseteq S$ 。

连续段对应到反链 非平凡连续段 $a \cup \{b_i \mid i \in A\} \cup \{c_i \mid i \in A\} \cup \{d_i \mid i \in A\} \cup \{e_{i,j} \mid i \in A, j \in S, i \neq j, i \leq j\}$ 对应反链 $\{i \mid i \in A, \forall j \leq i, j \notin A\}$ 。

反链对应到连续段 反链 A 对应 非平凡连续段 $a \cup \{b_i \mid i \in B\} \cup \{c_i \mid i \in B\} \cup \{d_i \mid i \in B\} \cup \{e_{i,j} \mid i \in B, j \in S, i \neq j, i \leq j\}$ ，其中 $B = \{i \mid \exists j \in A, j \leq i\}$ 。

¹⁰指至少含有 2 个元素的连续段。



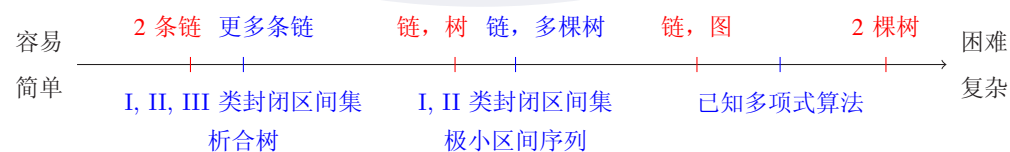
$$S = \{1, 2, 3\}, \leq = \{(1, 3), (2, 3), (1, 1), (2, 2), (3, 3)\}$$

图 5: 偏序集 $(S, \leq)$ 和对应的 2 棵树

上述过程把含有 n 个元素偏序集的反链计数问题，归约到了 2 棵 $O(n^2)$ 个点的树的连续段计数问题，因此对 2 棵树的连续段计数是 #P-complete 的，这意味着如果存在该问题的多项式算法，那么 $P = NP$ 。

6 总结

本文中的问题和结构的复杂性或困难性可以大致表示为：



本文指出了一类连续段问题的本质——若干个结构上“连通”的子集构成的集合的交。本文通过极小区间序列刻画了 I, II 类封闭的区间集 S ，并给出了对某个右端点 r 求出所有满足 $[l, r] \in S$ 的左端点 l 处的信息之和的线性算法，但需要被维护的信息存在逆元，较为局限。由于笔者能力有限，没有找到对于半群信息的算法或证明其不存在。此外，本文还给出了区间集 S 有析合树的充分必要条件，即 S I, II, III 类封闭。最后，本文将区间集 S 上

的算法用于解决连续段问题，并证明了一些连续段问题复杂性或困难性，例如计算 2 棵树的连续段的数量是 #P-complete 的。

希望本文能让读者对连续段有更深刻的理解，并作进一步的研究。

7 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练杨耀良、彭思进、张艺缤的指导。

感谢父母对我的培养和教育。

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

感谢镇海中学的同学们为本文验稿。

感谢刘承奥与我交流，为本文提供启发。

参考文献

- [1] Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12, 1983.
- [2] Jan van Leeuwen Robert E. Tarjan. Worst-case analysis of set union algorithms. *Journal of the ACM (JACM)*, 31:245 – 281, 1984.
- [3] 刘承奥. 简单的连续段数据结构. *WC2019 营员交流*, 2019.

OI 题目中凸性的证明方法

重庆市南开中学校 苟竞予

摘要

在本文中归纳总结了一些 OI 题目中的凸性的证明方法，将证明方法分为了三个大类，分别是规约为常见模型、代数方法、针对几何题的方法，并分别列举了一些例题。

1 前言

凸性在竞赛题目中很常见，研究凸性的文献和资料也已经有很多。在 2023 年国家集训队论文集，郭羽冲的论文 [3] 总结了一些常见的利用凸性解决问题的算法和技巧，并辅以一些例题。该论文主要通过多个例题讲述具体的算法设计，展示出作者丰富的技巧。通过网络检索或者在各大题库查找，也能找到非常多与凸性相关的题目和题解。确实，在信息学竞赛中，很多题目都需要选手发现题目中的凸性，并利用凸性解决问题。近年来这类题目日益增多，涉及到了很多种算法和技巧。笔者注意到，无论是这类题目的题解，还是选手对于这类题目的讨论，一般都不太重视凸性的证明。笔者查阅大量相关资料后发现大部分资料的关注点都集中在如何利用凸性解题上，即算法设计层面，而题目中的凸性如何证明却常常被忽略或过于简略。基于这一现象，笔者结合自己做题积累的经验，将证明凸性的方法分为了三个大类，以例题的方式展现，主要从分析题目性质的角度出发，着重探讨如何严谨的证明题目中的凸性。

2 基本概念

2.1 凸函数

函数的凸性是一种带有几何意义的数学性质。凸函数和凹函数的定义，在一些不同的中文数学教材中完全相反，而以英文为主的外文资料中有统一的定义。外文资料中凸函数定义为：

定义 2.1 (Convex Function). 定义在某个向量空间的凸子集 C 上，对于 $\forall a, b \in C$ 满足

$$f\left(\frac{a+b}{2}\right) \leq \frac{f(a)+f(b)}{2}$$

特别的，如果定义域为实数区间，则函数图像为一条开口向上的曲线，曲线上方的区域是凸集。凹函数的定义与凸函数定义的不等号相反，其函数图像为开口向下的曲线。

中文数学教材中有一些与国际保持一致，也有一部分与之恰恰相反。笔者认为这可能是因为汉字中“凸”和“凹”这两个字的语义造成的，“凸”让人产生一种函数曲线开口向下的直觉，而“凹”则让人觉得曲线开口向上。对于这些定义中的争议，笔者建议各位读者在阅读相关文献时保持谨慎的态度，先确认作者对凸函数究竟如何定义。另外笔者倡议各位读者如果要自己进行写作，最好在合适的位置给出明确的解释，并在同一语境下保证定义的一致性，以避免给他人造成阅读障碍。

在本文中，函数凸性延续了参考文献 [3] 的定义，将具有凸性的函数分为上凸函数和下凸函数两类。具体的：

定义 2.2 (上凸函数). 函数 $f(x)$ 是上凸函数，当且仅当对于定义域内的 $\forall a, b$ 都满足

$$f\left(\frac{a+b}{2}\right) \geq \frac{f(a)+f(b)}{2}$$

定义 2.3 (下凸函数). 函数 $f(x)$ 是下凸函数，当且仅当对于定义域内的 $\forall a, b$ 都满足

$$f\left(\frac{a+b}{2}\right) \leq \frac{f(a)+f(b)}{2}$$

2.2 凸序列

序列可以视为定义域内的元素都是整数的函数。对于序列这样的函数，凸性的定义可以等价的写为任意连续 3 项之间的大小关系。

定义 2.4 (上凸序列). 序列 $\{a_n\}$ 是上凸的，当且仅当

$$\forall i, 2a_i \geq a_{i-1} + a_{i+1}$$

定义 2.5 (下凸序列). 同理，序列 $\{a_n\}$ 是下凸的，当且仅当

$$\forall i, 2a_i \leq a_{i-1} + a_{i+1}$$

当序列项数 ≥ 3 时以上定义才会生效，而项数 < 3 的序列讨论凸性没有意义。然而，在实际问题中往往存在边界情况，因此序列项数 < 3 也很常见。一般情况下，可以认为项数 < 3 的序列既是上凸的，也是下凸的。

序列 $\{a_n\}$ 的差分序列定义为 $\{b_n\}$ ，满足 $b_i = a_i - a_{i-1}$ 。

序列是上凸的，等价于差分序列是单调递减的。这可以通过将凸序列的定义稍加变形得到。

$$2a_i \geq a_{i-1} + a_{i+1}$$

$$\Leftrightarrow a_i - a_{i-1} \geq a_{i+1} - a_i$$

$$\Leftrightarrow b_i \geq b_{i+1}$$

同理，序列下凸等价于差分序列单调递增。

序列 $\{a_n\}$ 的二阶差分序列 $\{c_i\}$ 定义为 $\{b_n\}$ 的差分序列，即 $c_i = b_i - b_{i-1}$ 。则序列上凸等价于二阶差分序列 $c_i \leq 0$ ，序列下凸等价于二阶差分序列 $c_i \geq 0$ 。

3 规约为常见算法模型证明

竞赛题目中要证明凸性，可以优先考虑将题目归结为比较常见的算法模型。与凸性密切相关的算法模型有费用流模型和四边形不等式模型，二者都有丰富的理论系统可以直接套用。于是，只要能够将题目中的模型规约到费用流或者四边形不等式的模型上，证明凸性就变得轻而易举。

3.1 费用流

竞赛中很多问题可以用费用流模型来理解，可能转化为建图后用费用流算法求解，也可能用数据结构或者其他手段模拟费用流来更高效的解决。OI 中建图求解费用流一般用 SSP 算法 [2]，即每次找最短路进行增广，直到图上不存在增广路为止。以最小费用最大流问题为例，随着算法的进行，图中流量不断增加，增广路的长度也在不断增加。如果将总费用写成关于流量的函数，则该函数的差分单调不降，所以是下凸函数。同理，最大费用最大流问题费用是关于流量的上凸函数。

例题 3.1 (House Deconstruction 简化版¹)。大小为 x 的环上有 n 个人 m 座房子，其中 $n < m$ 。给每个人分配一座房子，代价为环上的距离。求最小总代价。

数据范围 $1 \leq n < m \leq 2 \times 10^5$ ， $n + m \leq x \leq 10^9$ 。

算法 1 考虑 $n = m$ 时如何快速计算。若钦定没有人经过位置 1 和 x 之间的边，则此时的环等价一条链。设计 c_i 为前 i 个位置人的数量减去前 i 个位置的房子数量。容易注意到，一条边不会有两入分别从两个方向经过，若有，则交换这两个人的房子不劣。

因此，此时对应的代价即为 $\sum_{i=1}^{x-1} |c_i|$ 。

考虑钦定有 k 人经过走过了边 $x \rightarrow 1$ ，或者 $k < 0$ 表示有 $-k$ 个人经过边 $1 \rightarrow x$ 时，怎么计算代价。

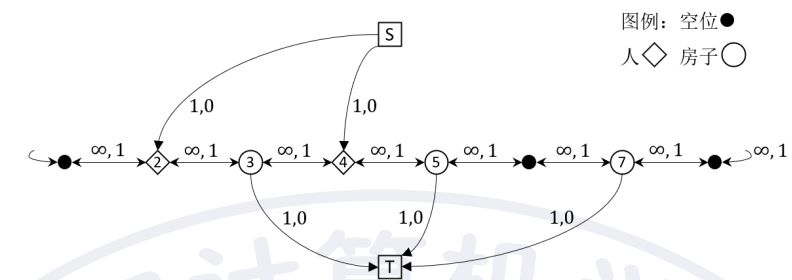
当 $k > 0$ 时，等价于环最后加一个点 $x+1$ ， x 到 $x+1$ 有一条边，而 $x+1$ 上额外增加 k 个房子，并在 1 处额外增加 k 个人。当 $k < 0$ 同理。

令此时的代价为 $h(k) = \sum_{i=1}^x |k + c_i|$ 。注意到 $h(k)$ 表达式中每个绝对值均为关于 k 的下凸函数，所以 $h(k)$ 也是下凸函数。本题答案为 $h(k)$ 的最小值，直接对 k 三分即可求解。时间复杂度 $O((n + m) \log(n + m))$

¹来源：The 3rd Universal Cup. Stage 0: Trial Contest，原题除了求最小总代价，还要计算方案数

算法 2 考虑直接建模为有源汇最小费用最大流：源点向每个人的位置连一条容量为 1 费用为 0 的边，每个房子的位置向汇点连一条容量为 1 费用为 0 的边，而环上相邻两点间正向与反向分别连一条容量为 $+\infty$ 费用为 1 的边。参考图 1

图 1: 费用流建图示例。环长 $x = 8$ ，在位置 2, 4 各有一人，在位置 3, 5, 7 各有一座房子。



令 $g(k)$ 表示钦定费用流图中点 x 和点 1 之间两条边流量之差为 k 时 ($k > 0$ 则方向为 $x \rightarrow 1$ 流量为 k ，否则为 $1 \rightarrow x$ 流量为 $-k$)，原图的最小费用最大流。

引理 3.1. 函数 $g(k)$ 在定义域 $[-n, n]$ 上是一个下凸函数。

证明. 等价于一个有源汇有上下界最小费用可行流问题，其中 $x \rightarrow 1$ 流量的上下界限制为 $[-n, n]$ 。

不妨先令 $x \rightarrow 1$ 的流量为 $-n$ ，等价于钦定 $1 \rightarrow x$ 的流量为 n ，并在当前图上先跑有源汇有上下界最小费用可行流，令此时的费用为 y 。现在可以考虑 k 从 $[-n, n]$ 的过程中 $x \rightarrow 1$ 的费用变化，这等价于当 $k \in [-n, 0]$ 时每增加 1 流量费用减少 1；当 $k \in [0, n]$ 时每增加 1 流量费用增加 1。

因此可以在残量网络上新建一条 $x \rightarrow 1$ 的边，容量为 n ，费用为 1。考虑当 k 变成 $k+1$ 时：

- 若 $k < 0$ ，则一定会在 $1 \rightarrow x$ 的边退回 1 流量；
- 若 $k \geq 0$ ，则一定会在 $x \rightarrow 1$ 的边增加 1 流量。

因此，直接添加一条超级源至位置 x 的容量为 $2n$ 费用为 0 的边，设其为 e ，并添加一条位置 1 至超级汇的容量为 $2n$ 费用为 0 的边。在当前图上再次跑上下界最小费用可行流，则 $g(k)$ 的答案就是 e 流量为 $k+n$ 时的最小费用，加上原图的费用 y 。根据费用流的性质可知，当 k 增加 1 时 $g(k)$ 增加量单调不降，因此 $g(k)$ 时下凸函数，引理 3.1 得证。□

算法 3 考虑结合算法 1 与算法 2。显然算法 1 的 $h(k)$ 与算法 2 的 $g(k)$ 相等。所以我们仍对 k 三分，接下来则只需解决固定 k 时的问题。修改 c_i 的定义，令 c_i 为前 i 个位置人的

数量减去前 i 个位置中有人匹配的房子数量。我们考虑利用 dp 解决此问题，设 $dp(i, j)$ 表示考虑了前 i 个位置， $c_i + k = j$ 的最小代价。初始值为 $dp(0, k) = 0$ ，转移如下：

$$dp(i, j) = \begin{cases} dp(i-1, j-1) + |j| & \text{位置 } i \text{ 是人} \\ \min(dp(i-1, j+1), dp(i-1, j)) + |j| & \text{位置 } i \text{ 是房子} \\ dp(i, j) = dp(i-1, j) + |j| & \text{位置 } i \text{ 为空} \end{cases}$$

最终的答案即为 $dp(x, k)$ 。

注意到 $dp(i, j)$ 对于固定的 i ，关于 j 是下凸的。程序实现为了方便，可以把状态改为 $dp(i, j)$ 表示前 i 个位置选了 j 个房子的最小代价。这时最小值的位置 j 一定是单调不降的，可以用数组维护最小值右侧的斜率的差分。对于房子只需插入一个 0 的斜率，连续的空位置和人可以合并处理。

算法时间复杂度 $O(m \log n)$ ，空间复杂度 $O(n + m)$ 。

3.2 四边形不等式

定义 3.1 (四边形不等式). 如果对于 $a \leq b \leq c \leq d$ ，二元函数 w 满足以下条件则称为满足四边形不等式：

$$w(a, c) + w(b, d) \leq w(a, d) + w(b, c)$$

四边形不等式在各种 OI 学习资料，例如 OI-wiki [1] 等，都已经有很多非常详细的讲述和论证。本文中不再详细描述，只将几个常用的题目模型和与之对应的结论罗列出来，具体证明可以参考其他资料和文献。在证明题目凸性时，如果要将问题规约到四边形不等式模型，核心在于找到题目中满足四边形不等式的代价函数 w 。

例题 3.2 (划分问题). 将 $1, 2, \dots, n$ 划分成多段，段 $l, \dots, r$ 的代价为 $w(l, r)$ ，求最小总代价。

DP 转移方程形如

$$f(i) = \min_{1 \leq j \leq i} f(j-1) + w(j, i)$$

只要 $w(j, i)$ 满足四边形不等式，则状态转移具有决策单调性。用这个结论可以将 DP 用二分队列优化到 $O(n \log n)$ 时间复杂度。

例题 3.3 (段数划分问题). 将 $1, 2, \dots, n$ 划分成 m 段，段 $l, \dots, r$ 的代价为 $w(l, r)$ ，求最小总代价。

DP 转移方程形如

$$f(i, k) = \min_{1 \leq j \leq i} f(j-1, k-1) + w(j, i)$$

只要 $w(j, i)$ 满足四边形不等式，则 $f(i, k)$ 是关于 k 的下凸函数。这个结论是 wqs 二分算法的基础，将计算 $f(n, m)$ 的时间复杂度优化到 $O(n \log n \log W)$ ，其中 W 是某个与 $w(j, i)$ 值域相关的常数。

例题 3.4 (Bar Cover²). 给一个长度为 n 的整数序列 $a_1, \dots, a_n$ ，一张卡片可以覆盖连续 k 个数，卡片不能重叠。对 $i = 1, 2, \dots, \lfloor \frac{n}{k} \rfloor$ 分别计算恰好用 i 张卡片时被覆盖所有数字之和的最大值。

数据范围 $1 \leq n \leq 2 \times 10^5$ ， $1 \leq k \leq \min(5, n)$ ， $|a_i| \leq 10^9$ 。

为方便描述，设使用 i 张卡片的答案为 $f(i)$ ，其中 $0 \leq i \leq \lfloor \frac{n}{k} \rfloor$ 。

引理 3.2. $f(i)$ 是上凸函数。即对于 $0 < i < \lfloor \frac{n}{k} \rfloor$ 都有 $f(i-1) + f(i+1) \leq 2f(i)$ 。

本题可以直接利用题目中的组合意义证明，也可以套用四边形不等式模型来证明凸性。以下是用组合意义证明：

证明. 设 $f(i-1)$ 对应最优方案的 $i-1$ 张卡片起点位置分别是 $b_1, b_2, \dots, b_{i-1}$ ，设 $f(i+1)$ 对应最优方案的 $i+1$ 张卡片起点位置分别是 $c_1, c_2, \dots, c_{i+1}$ 。它们满足相邻两项之差至少为 k ，即 $b_j + k \leq b_{j+1}$ ， $c_j + k \leq c_{j+1}$ 。

将 $b_1, \dots, b_{i-1}$ 和 $c_1, \dots, c_{i+1}$ 归并，设归并后的序列为 $d_1, d_2, \dots, d_{2i}$ 。按归并后的下标奇偶性重新划分成两个子序列 $D_1 = (d_1, d_3, d_5, \dots, d_{2i-1})$ 和 $D_2 = (d_2, d_4, d_6, \dots, d_{2i})$ 。序列 D_1 和 D_2 也满足相邻两项之差至少为 k ，即 $d_j + k \leq d_{j+2}$ 。这是因为：

- 如果 d_j 和 d_{j+2} 来自归并前的同一个序列则相差至少为 k ；
- 如果来自不同序列，则考虑中间项 d_{j+1} ，无论 d_{j+1} 与 d_j 和 d_{j+2} 中哪一项来自归并前的同一个序列，都会相差至少为 k 。又因为 $d_j \leq d_{j+1} \leq d_{j+2}$ ，从而 $d_j + k \leq d_{j+2}$ 。

也就是说序列 D_1 和 D_2 分别对应了合法方案，方案的卡片数量等于 i 。上述过程并未增删或移动卡片，所以这两个方案的数字之和的和等于 $f(i-1) + f(i+1)$ 。而 $f(i)$ 是 i 张卡片的最优方案的代价，必然不小于这两个方案中的任意一个，所以 $f(i-1) + f(i+1) \leq 2f(i)$ 。引理 3.2 得证。

可以参照图片 2 中的实例理解上述过程。

□

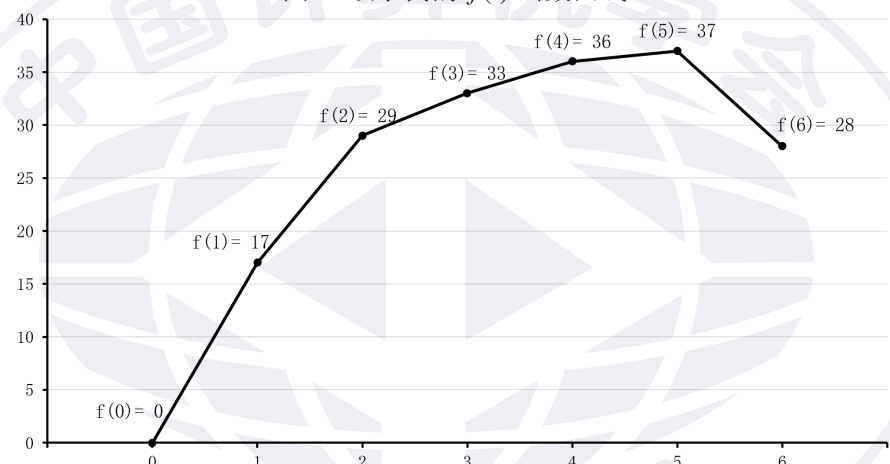
以下是利用四边形不等式模型证明：

²来源：Atcoder ABC383 G

图 2: 组合意义证明 $f(3) + f(5) \leq 2f(4)$ 的一个示例

下标 i	1	2	3	4	5	6	7	8	9	10	11	12
序列 a_i	-1	5	7	2	-8	6	-2	-1	10	7	2	1
$f(3)$ 最优方案	-1	5	7	2	-8	6	-2	-1	10	7	2	1
$f(5)$ 最优方案	-1	5	7	2	-8	6	-2	-1	10	7	2	1
归并	$\{b\} = (2, 6, 9)$ $\{c\} = (1, 3, 6, 9, 11)$ $\{d\} = (1, 2, 3, 6, 6, 9, 9, 11)$											
再划分	$D_1 = (1, 3, 6, 9)$ $D_2 = (2, 6, 9, 11)$											
D_1 的方案	-1	5	7	2	-8	6	-2	-1	10	7	2	1
D_2 的方案	-1	5	7	2	-8	6	-2	-1	10	7	2	1
$f(4)$ 最优方案	-1	5	7	2	-8	6	-2	-1	10	7	2	1

图 3: 该示例的 $f(i)$ 函数曲线



证明. 设 $w(l, r)$ 表示在 $a_l, \dots, a_r$ 范围内放置一张卡片时的最大值。则

$$w(l, r) = \begin{cases} -\infty & r - l + 1 < k \\ \max_{l \leq p \leq r-k+1} (a_p + \dots + a_{p+k-1}) & r - l + 1 \geq k \end{cases}$$

上式中的 ∞ 在实际程序中设置远大于 $|a_i|$ 值域上限的常数。设 $b_p = a_p + \dots + a_{p+k-1}$ ，其中 $1 \leq p \leq n - k + 1$ 。考虑 $\Delta = w(l, r - 1) + w(l + 1, r) - w(l + 1, r - 1) - w(l, r)$ 的正负性，设 b_{p_1} 和 b_{p_2} 分别是 $b_l, \dots, b_{r-k+1}$ 中的最大和次大

- 若 $l < p_1 < r - k + 1$ ，则这四个区间的代价均为 b_{p_1} ，从而 $\Delta = 0$ ；
- 若 $p_1 = l$ 或者 $p_1 = r - k + 1$

- 若 $l < p_2 < r - k + 1$ ，则四个区间有两个代价为 b_{p_1} ，另外两个为 b_{p_2} ，从而 $\Delta = 0$ ；
- 若 p_2 等于 l 和 $r - k + 1$ 中的另一个，则 $w(l + 1, r - 1)$ 的值为第三大的 b_p ，此时 $\Delta \geq 0$ ；

由 $\Delta \geq 0$ 得 $w(l, r)$ 满足四边形不等式。也就是说，本题符合“段数划分问题”的 DP 模型，所以划分 i 段的总代价关于 i 是上凸函数。□

在确定了此题的答案具有凸性之后，可以利用凸性解决问题。

算法 1 使用分治算法，将子问题 (l, r) 设置为：对每个子序列 $a_{l+x}, \dots, a_{r-y}$ 分别计算原答案的凸序列 $f_{x,y}(i)$ ，其中 $0 \leq x, y \leq k - 1$ 。每个子问题共 $O(k^2)$ 个答案序列，每个答案序列的长度为 $O(\frac{r-l+1}{k})$ 。分治时先递归求解子问题 (l, mid) 和 $(mid + 1, r)$ ，然后将它们的答案合并为子问题 (l, r) 的答案，合并时需要用 k^3 次凸序列 $(\max, +)$ 卷积，即闵可夫斯基和，时间复杂度 $O(k^3 \cdot \frac{r-l+1}{k})$ 。算法总时间复杂度 $O(k^2 n \log n)$ ，空间复杂度 $O(kn \log n)$ 。

由四边形不等式，本题的 DP 算法还具有决策单调性，算法也可以基于这一点设计。

算法 2 设 $f(p, i)$ 表示 $a_1, \dots, a_p$ 放置 i 张卡片的最大总和。转移方程

$$f(p, i) = \max \begin{cases} f(p - 1, i) \\ f(p - k, i - 1) + b_{p-k+1} \end{cases}$$

根据决策单调性，存在整数 i_p ，当 $i \leq i_p$ 时状态从 $f(p - 1, i)$ 转移，当 $i > i_p$ 时状态从 $f(p - k, i - 1) + b_{p-k+1}$ 转移。用可持久化平衡树维护 DP 状态数组，对每个 p 用一棵平衡树 $Tree(p)$ 存储 $i = 0, 1, 2, \dots$ 的状态 $f(p, i)$ 。那么状态转移可以视为取 $Tree(p - 1)$ 的前 i_p 个节点，再取 $Tree(p - k)$ 的从第 i_p 起的后面所有节点，打标记后再拼接在一起。可以先二分 i_p 的值，在平衡树上查询 $f(p - 1, i)$ 和 $f(p - k, i - 1) + b_{p-k+1}$ 比较谁更优来验证 i_p 偏小或者偏大。每个 p 的状态转移需要 $O(\log^2 n)$ 时间复杂度求 i_p ，然后 $O(\log n)$ 的时间复杂度构建 $Tree(p)$ 。算法总时间复杂度 $O(n \log^2 n)$ ，空间复杂度 $O(n)$ 。

算法 2 的时间复杂度与 k 无关，对于 k 数据范围稍大的情况仍然可以处理，适用范围比算法 1 更广。但因为其常数较大，所以在本题 $k \leq 5$ 的数据范围下比算法 1 稍慢。

4 基于代数方法的证明

凸性的定义本身具备代数形式，因此有些题可以采用代数方法证明凸性。这样的证明方法一般来说不容易拓展到其他题目，只能作为个例。本章展示两道例题的代数证明过程。

4.1 一个例题

例题 4.1 (高性能计算机³)。把 n_A 个 A 类任务和 n_B 个 B 类任务分配给一台计算机的 p 个计算节点。第 i 个计算节点有四个参数 $t_i^A, t_i^B, k_i^A, k_i^B$ ，表示这个节点连续执行 x 个 A 类任务需要 $t_i^A + k_i^A \cdot x^2$ 纳秒，连续执行 x 个 B 类任务需要 $t_i^B + k_i^B \cdot x^2$ 纳秒。所有计算节点同时启动，求完成所有任务最少需要多少纳秒。

数据范围 $1 \leq n_A, n_B \leq 60$ ， $1 \leq p \leq 20$ ， $1 \leq t_i^A, t_i^B \leq 1000$ ， $1 \leq k_i^A, k_i^B \leq 50$ 。

值得注意的是此题年代久远，设计算法需要以当时的评测机性能为参考。

算法 1 在不分析题目性质的情况下，可以分两步进行 DP，先计算每一个计算节点的信息，再算将它们合并到一起。第一步设 $f(i, a, b, c)$ 表示第 i 个计算节点解决 a 个 A 类任务和 b 个 B 类任务，且最后一个任务的类型是 c 的最短用时。其中 $1 \leq i \leq p$ ， $1 \leq a \leq n_A$ ， $1 \leq b \leq n_B$ ， $c \in \{0, 1\}$ 。状态转移需要枚举最后一段任务的数量：

$$\begin{aligned} f(i, a, b, 0) &= \min_{1 \leq x \leq a} (f(i, a-x, b, 1) + t_i^A + k_i^A \cdot x^2) \\ f(i, a, b, 1) &= \min_{1 \leq x \leq b} (f(i, a, b-x, 0) + t_i^B + k_i^B \cdot x^2) \end{aligned}$$

接下来不妨设 $f(i, a, b) = \min(f(i, a, b, 0), f(i, a, b, 1))$ 。第二步设 $g(i, a, b)$ 表示前 i 个计算节点解决 a 个 A 类任务和 b 个 B 类任务的最短用时。状态转移需要枚举第 i 台机器解决的两类任务数量 x 和 y ：

$$g(i, a, b) = \min_{0 \leq x \leq a} \min_{0 \leq y \leq b} \max(g(i-1, a-x, b-y), f(i, x, y))$$

该算法第一部分 DP 的时间复杂度 $O(pn^3)$ ，第二部分 DP 的时间复杂度高达 $O(pn^4)$ ，整个算法的空间复杂度为 $O(pn^2)$ 。以现代评测机的运行效率，这个算法可以轻松通过本题；但如果以 2001 年的评测机效率为准，则有一定的超时风险。算法时间复杂度的瓶颈在于第二部分 DP，所以需要针对第二部分优化。

算法 2 发现 $f(i, a, b)$ 固定 a 时是 b 的单谷函数。如果 A 类任务数量固定的情况下，每个计算节点能在一定时间内解决的 B 类任务数量是一个连续区间，即固定 a, t 时满足 $f(i, a, b) \leq t$ 的 b 是一段连续区间。如果增大时间限制 t ，区间范围扩大。将算法 1 的第二部分 DP 改成二分答案：二分时间限制 ans ，先利用第一阶段 DP 结果 $f(i, a, b)$ 算出第 i 台机器在 ans 时间内必须恰好解决 a 个 A 类任务的情况下至少和至多可以解决多少个 B 类任务，记为 $h(i, a)$ 。

$$h_{\min}(i, a) = \min\{b \mid f(i, a, b) \leq ans\}$$

$$h_{\max}(i, a) = \max\{b \mid f(i, a, b) \leq ans\}$$

再设 $g(i, a)$ 表示在 ans 时间内前 i 个计算节点必须恰好解决 a 个 A 类任务的同时最少和最

³来源：WC2001

多可以解决多少个 B 类任务。二分要验证 n_B 在 $g(p, n_A)$ 的区间内。

$$\begin{aligned} g_{\min}(i, a) &= \min_{0 \leq x \leq a} (g_{\min}(i-1, a-x) + h_{\min}(i, x)) \\ g_{\max}(i, a) &= \max_{0 \leq x \leq a} (g_{\max}(i-1, a-x) + h_{\max}(i, x)) \end{aligned}$$

该算法将第二部分优化到 $O(pn^2 \log ans)$ 。虽然效率提高，但遗留了一个关键问题：为什么 $f(i, a, b)$ 固定 a 时是 b 的单谷函数？如果用程序打表观察这个性质很容易被发现，但是要证明这个性质却并不容易。而且事实上，本题中隐藏的性质并不止于此，还有更多可以挖掘的性质，以凸性为主，单谷性的证明需要依赖这些隐藏的凸性。

因为只考虑单个计算节点，以下为方便描述将 $f(i, a, b)$ 简记为 $f(a, b)$ ，省略了计算节点的编号 i 。 $f(a, b)$ 的最优策略将 a 个 A 类任务和 b 个 B 类任务分别划分成连续段，设连续段的长度分别为 $a_1, a_2, \dots, a_r$ 和 $b_1, b_2, \dots, b_s$ ，其中 $|r-s| \leq 1$ ， $a_1 + \dots + a_r = a$ ， $b_1 + \dots + b_s = b$ 。

引理 4.1. 最优策略中不存在 $|a_{j_1} - a_{j_2}| \geq 2$ ，也不存在 $|b_{j_1} - b_{j_2}| \geq 2$ 。

证明. 由于对称性，只需证明不存在 $|a_{j_1} - a_{j_2}| \geq 2$ ，用反证法证明。假设存在 $a_{j_1} - a_{j_2} \geq 2$ ，若将这两个连续段的长度调整 $a'_{j_1} = a_{j_1} - 1$ ， $a'_{j_2} = a_{j_2} + 1$ ，其他连续段长度均不变。新方案与原方案代价之差为

$$\begin{aligned} \Delta &= k_i^A \cdot ((a_{j_1} - 1)^2 + (a_{j_2} + 1)^2 - a_{j_1}^2 - a_{j_2}^2) \\ &= k_i^A \cdot (2 - 2(a_{j_1} - a_{j_2})) \\ &\leq k_i^A \cdot (2 - 2 \times 2) \\ &< 0 \end{aligned}$$

即调整后的新方案比原方案更优，这与原方案是最优策略的假设相矛盾。因此最优策略中一定不存在 $|a_{j_1} - a_{j_2}| \geq 2$ ，由对称性，也一定不存在 $|b_{j_1} - b_{j_2}| \geq 2$ 。引理 4.1 得证。□

由引理 4.1 可知，最优策略的连续段长度满足所有 a_j 至多相差 1，所有 b_j 也至多相差 1。基于这一性质，其实已经不用 DP 就可以计算 $f(a, b)$ 的值了。枚举 A 类任务的连续段数量 $r = 1, 2, \dots, a$ ，则 A 类任务连续段的长度为 $\lfloor \frac{a}{r} \rfloor$ 和 $\lfloor \frac{a}{r} \rfloor + 1$ ，两种长度的连续段数量分别为 $r - (a \bmod r)$ 和 $a \bmod r$ 。不妨设 $a = \alpha \cdot r + \beta$ ，其中 $0 \leq \beta < r$ 。则 a 个 A 类任务分成 r 段的的最小总代价为

$$f_A(a, r) = r \cdot t_i^A + k_i^A \cdot (r - \beta) \cdot \alpha^2 + k_i^A \cdot \beta \cdot (\alpha + 1)^2$$

显然 B 类任务分成 s 段的最小总代价 $f_B(b, s)$ 具有相同的形式。

函数 $f_A(a, r)$ 形式较为复杂，这使得发觉其性质的难度很高。此时不妨先采用不严谨的方式进行观察，确定可能的性质明确研究方向后再尝试严谨证明。将划分连续段限制放宽，

允许连续段的长度不必是整数，从而使函数定义域从整数区间变成实数区间。 r 个“连续段”长度均为 $\frac{a}{r}$ ，新的代价函数为

$$\begin{aligned} f_A^*(a, r) &= r \cdot t_i^A + k_i^A \cdot r \cdot \left(\frac{a}{r}\right)^2 \\ &= r \cdot t_i^A + k_i^A \cdot a^2 \cdot \frac{1}{r} \end{aligned}$$

根据均值不等式， $f_A(a, r) \geq f_A^*(a, r)$ ，取等条件当且仅当 $\frac{a}{r}$ 是整数。

观察 $f_A^*(a, r)$ 的表达式可以发现：

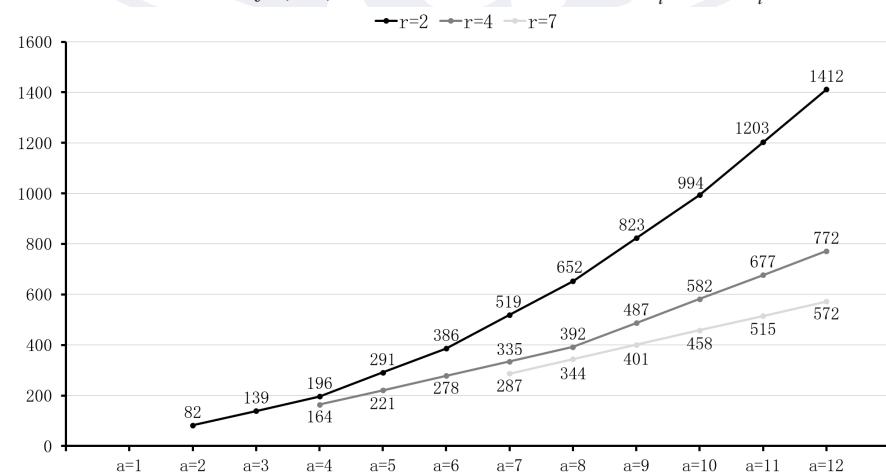
- 固定 r 时，函数 $f_A^*(a, r)$ 是关于 a 的下凸函数，且单调递增；
- 固定 a 时，函数 $f_A^*(a, r)$ 是关于 r 的下凸函数。
- 设 $r_0 = a \cdot \sqrt{\frac{k_i^A}{t_i^A}}$
 - 若 $r_0 \leq a$ ，则函数 $f_A^*(a, r)$ 在 $r \in [1, r_0]$ 单调递减，在 $r \in [r_0, a]$ 单调递增，在 $r = r_0$ 处时取得最小值；
 - 否则函数 $f_A^*(a, r)$ 在定义域内 $r \in [1, a]$ 单调递减，在 $r = a$ 处取得最小值。

事实上函数 $f_A(a, r)$ 也满足上述下凸的性质，且取得最小值的点为 r_0 取整（向上或向下均有可能）。该证明过程需要用到较多的分类讨论，较为繁琐。

引理 4.2. $f_A(a, r)$ 是关于 a 的下凸函数，且单调递增。

图 4 展示了 $f_A(a, r)$ 关于 a 的函数曲线，图中三种情况均为下凸函数。

图 4: 不同情况下 $f_A(a, r)$ 关于 a 的函数曲线示例 ($t_i^A = 22, k_i^A = 19$)



证明. 假设 r 固定，考虑 a 增加 1 时 $f_A(a, r)$ 变成 $f_A(a+1, r)$ 。显然，最优划分方案是将一个长度为 α 的连续段变成 $\alpha+1$ ，其他连续段长度都不变。这个操作显然使代价增大，所以即 $f_A(a, r)$ 在固定 r 时关于 a 时单调递增的。考虑增加量：

$$\begin{aligned} \Delta(a) &= f_A(a, r) - f_A(a-1, r) \\ &= k_i^A \cdot (\alpha+1)^2 - k_i^A \alpha^2 \\ &= k_i^A \cdot (2\alpha+1) \\ &= k_i^A \cdot \left(2\left\lfloor \frac{a}{r} \right\rfloor + 1\right) \\ &> 0 \end{aligned}$$

即增加量 $\Delta(a)$ 关于 a 单调不降且恒大于零，所以 $f_A(a, r)$ 关于 a 是单调递增的下凸函数。□

引理 4.3. $f_A(a, r)$ 是关于 r 在每个整除分块 $\lfloor \frac{a}{\alpha+1} \rfloor < r \leq \lfloor \frac{a}{\alpha} \rfloor$ 内是一次函数。

证明. 如果 r 和 $r+1$ 在同一个整除块内，设 $a = \alpha \cdot r + \beta = \alpha' \cdot (r+1) + \beta'$ ，则 $\alpha = \lfloor \frac{a}{r} \rfloor = \lfloor \frac{a}{r+1} \rfloor = \alpha'$ ， $\beta' = \beta - \alpha$ 。函数 $f_A(a, r)$ 的变化量：

$$\begin{aligned} \Delta(r) &= f_A(a, r+1) - f_A(a, r) \\ &= ((r+1) \cdot t_i^A + k_i^A \cdot (r+1 - \beta + \alpha) \cdot \alpha^2 + k_i^A \cdot (\beta - \alpha) \cdot (\alpha+1)^2) \\ &\quad - (r \cdot t_i^A + k_i^A \cdot (r - \beta) \cdot \alpha^2 + k_i^A \cdot \beta \cdot (\alpha+1)^2) \\ &= t_i^A + k_i^A \cdot (\alpha+1) \cdot \alpha^2 + k_i^A \cdot (-\alpha) \cdot (\alpha+1)^2 \\ &= t_i^A - k_i^A \cdot \alpha \cdot (\alpha+1) \end{aligned}$$

当 r 在同一个整除块内时 $\Delta(r)$ 是定值，所以每一个整除块内 $f_A(a, r)$ 是一次函数，斜率为 $\Delta(r)$ 。□

用直线 L_α 表示该一次函数，解析式为：

$$\begin{aligned} L_\alpha(r) &= t_i^A \cdot r + k_i^A \cdot ((r - \beta) \cdot \alpha^2 + \beta \cdot (\alpha+1)^2) \\ &= t_i^A \cdot r + k_i^A \cdot ((r - a + \alpha \cdot r) \cdot \alpha^2 + (a - \alpha \cdot r) \cdot (\alpha+1)^2) \end{aligned}$$

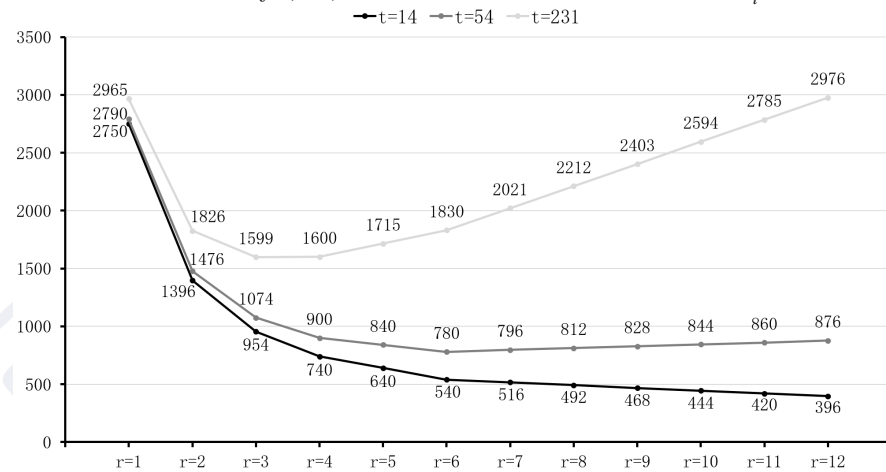
注意到这样的定义，即使 α 的整除块内只有一个整数 r ，或者没有整数 r ，也可以确定一条直线。

每个整除块的 α 随着 r 增大而减小，所以随着 r 增大，每个整除块的一次函数的斜率随着 r 增大而增大。如果要进一步证明 $f_A(a, r)$ 是下凸函数，则需要证明每段整除块之间的连接处也满足下凸性质。

引理 4.4. 对于 $1 \leq r \leq a$ 有 $f_A(a, r) = \max_{1 \leq \alpha \leq a} L_\alpha(r)$ ，即 $f_A(a, r)$ 是所有直线的半平面交，是关于 r 的下凸函数。

图 5 展示了 $f_A(a, r)$ 关于 r 的函数曲线，图中三种情况均为下凸函数。

图 5: 不同情况下 $f_A(a, r)$ 关于 r 的函数曲线示例 ($a = 12, k_i^A = 19$)



证明. 设 $r_0 - 1$ 是整除块 $\lfloor \frac{a}{\alpha+1} \rfloor < r \leq \lfloor \frac{a}{\alpha} \rfloor$ 的右端点，即 $r_0 - 1 = \lfloor \frac{a}{\alpha} \rfloor$ 。 r_0 是下一个整除块 $L_{\alpha'}$ 的左端点，其中 α' 满足 $\alpha > \alpha'$ 。 下证：在 r_0 处 $L_\alpha < L_{\alpha-1} < \dots < L_{\alpha'}$ 。 设 $\alpha \geq \delta > \alpha'$

$$\begin{aligned} L_\delta(r_0) &= t_i^A \cdot r_0 + k_i^A \cdot ((r_0 - a + \delta \cdot r_0) \cdot \delta^2 + (a - \delta \cdot r_0) \cdot (\delta + 1)^2) \\ &= t_i^A \cdot r_0 + k_i^A \cdot (-r_0 \cdot \delta \cdot (\delta + 1) + a \cdot (2\delta + 1)) \end{aligned}$$

那么

$$\begin{aligned} L_\delta(r_0) - L_{\delta-1}(r_0) &= k_i^A \cdot (-r_0 \cdot \delta \cdot (\delta + 1) + a \cdot (2\delta + 1)) \\ &\quad - k_i^A \cdot (-r_0 \cdot (\delta - 1) \cdot \delta + a \cdot (2\delta - 1)) \\ &= k_i^A \cdot 2 \cdot (-r_0 \cdot \delta + a) \\ &< 0 \end{aligned}$$

同理可以证明，在 $r_0 - 1$ 处 $L_\alpha > L_{\alpha-1} > \dots > L_{\alpha'}$ 。 因此， $L_\alpha, L_{\alpha-1}, \dots, L_{\alpha'}$ 这些直线任意两条的交点都在开区间 $(r_0 - 1, r_0)$ 内，当 $r \leq r_0 - 1$ 时 $L_\alpha > L_{\alpha-1} > \dots > L_{\alpha'}$ ，当 $r \geq r_0$ 时 $L_\alpha < L_{\alpha-1} < \dots < L_{\alpha'}$ 。 结合这些直线斜率单调递增的性质，这样就证明了 $f_A(a, r)$ 是所有直线 L_α 的半平面交。 并由此推论 $f_A(a, r)$ 是关于 r 的下凸函数，引理 4.4 得证。 $\square$

值得一提的是，事实上 $f_A(a, r)$ 可以 DP 计算，且符合四边形不等式中“段数划分问题”的模型，因此可以跳过前几个引理直接证明 $f_A(a, r)$ 关于 r 是下凸函数。 但这样的证明相当于将四边形不等式理论当做一个黑盒使用，对函数本身的性质的分析并不够深入，无法说明连续段长度至多相差 1，也无法得到函数的解析式，更无法说明函数最小值在 r 满足什么条件时取得。 然而，这些结论恰恰又是本文的后续证明中需要用到的。

现在已经深度挖掘了函数 $f_A(a, r)$ 的性质，即仅考虑一种任务时任务数、段数、总时间之间的关系，可以开始着手研究两种任务整体的情况，也就是

$$f(a, b) = \min_{|r-s| \leq 1} (f_A(a, r) + f_B(b, s))$$

考虑 $f(a, b)$ 取得最小值时候的段数的二元组 $\langle r, s \rangle$ 。 根据引理 4.4 可以知道 $f_A(a, r)$ 和 $f_B(b, s)$ 分别是关于 r 和 s 的下凸函数，设二者取得极小值的点分别为 r_0 和 s_0 。 若 $k_i^A < t_i^A$ 则 $r_0 \approx a \cdot \sqrt{\frac{k_i^A}{t_i^A}}$ (向上或向下取整)，否则 $r_0 = a$ 。 s_0 同理。 在增设限制 $|r - s| \leq 1$ 后， $f(a, b)$ 取得整体最优的 $\langle r, s \rangle$ 分为以下三种情况：

1. $r < s$ ：因为 $\langle r, s \rangle$ 比 $\langle r+1, s \rangle$ 优，即 $f_A(a, r) \leq f_A(a, r+1)$ ，或者 $r = a$ ，所以 $r_0 \leq r$ 。 因为 $\langle r, s \rangle$ 比 $\langle r, s-1 \rangle$ 优，所以 $s \leq s_0$ 。 也就是 $r_0 \leq r < s \leq s_0$ 。
2. $r = s$ ：因为 $\langle r, s \rangle$ 比 $\langle r \pm 1, s \rangle$ 和 $\langle r, s \pm 1 \rangle$ 都要优，所以必然是 $f_A(a, r)$ 和 $f_B(b, s)$ 同时取到了各自的极小值，即 $r_0 = r = s = s_0$ 。
3. $r > s$ ：与情况 1 同理， $s_0 \leq s < r \leq r_0$ 。

固定 b 考虑 a 从 1 开始逐渐增大，则 s_0 固定而 r_0 逐渐增大。 故取得整体最优的 $\langle r, s \rangle$ 初始为情况 1，某个时刻变成情况 2，再某个时刻变成情况 3。 且随着 a 每次增加 1， r 每次至多增加 1。

引理 4.5. 若 $f(a, b) \geq f(a+1, b)$ ，则 $f(a, b)$ 最优方案的 A 类任务划分段数 $r = a$ 。

证明. 反之，考虑 $r < a$ 的时候。

若 $f(a, b)$ 变成 $f(a+1, b)$ 时，最优段数 (r, s) 变成 (r', s') ，其中 $r \leq r' \leq r+1$ ， $s \leq s' \leq s+1$ 。 考虑中间状态 $f_A(a, r') + f_B(b, s')$ ，即任务数量为 a, b 但钦定段数为 (r', s') 的答案。 中间状态与 $f(a, b)$ 相比不优。 而 $f(a+1, b)$ 与中间状态相比， B 类任务方案不变， A 类任务方案中某个连续段长度加 1。 因此 $f(a, b) \leq f_A(a, r') + f_B(b, s') < f(a+1, b)$ 。 即当 $r < a$ 时 $f(a, b) < f(a+1, b)$ 。

如果 $f(a, b) \geq f(a+1, b)$ ，则只可能是在 $f(a, b)$ 最优方案满足 $r = a$ 时。 引理 4.5 得证。 $\square$

引理 4.6. 当最优方案满足 $r = a$ 时，固定 b ，则 $f(a, b)$ 是关于 a 的下凸函数。

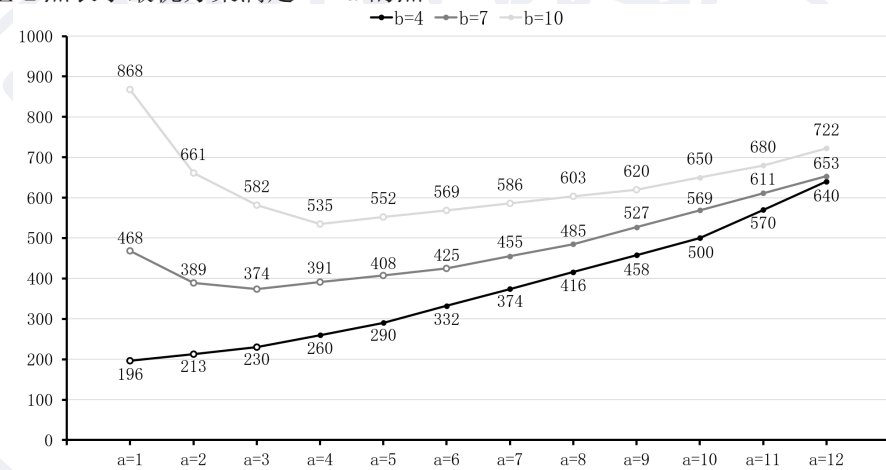
证明. 此时 $f(a, b) = f_A(a, a) + f_B(b, a + 1)$ 。其中 $f_A(a, a) = (t_i^A + k_i^A) \cdot a$ ，是关于 a 的一次函数， $f_B(b, a + 1)$ 已经证明是关于 a 的下凸函数。所以 $f(a, b)$ 在该情况下是下凸函数。

□

由此引理 4.5 和引理 4.6 可以得到推论：固定 b 时， $f(a, b)$ 可以视为一段下凸函数和一段递增函数的拼接，所以 $f(a, b)$ 是单谷函数。至此，算法 2 中用到的结论终于得到可严格的证明。

图 6 展示了 $f(a, b)$ 关于 a 的函数曲线，图中三种情况均为单谷函数。每条曲线上的符合 $r = a$ 的空心点满足下凸性质；而从最后一个空心点起，曲线单调递增。每条曲线都呈现为单谷函数。

图 6: 不同情况下 $f(a, b)$ 关于 a 的函数曲线示例，参数 $t_i^A = 16$ ， $k_i^A = 14$ ， $t_i^B = 19$ ， $k_i^B = 16$ 。图中空心点表示最优方案满足 $r = a$ 的点。



4.2 另一个例题

例题 4.2 (Dedenne⁴). 单词是不含连续 0 的 01 串，词典是任意两个单词不形成前缀关系的单词集合。一个 01 串如果是词典中 k 个单词的前缀则代价为 $\sum_{j=1}^k (1 + \lfloor \log_2 j \rfloor)$ ，词典的代价为所有 01 串的代价之和（含空串）。求 n 个单词的词典的代价的最小值。

数据范围 $1 \leq n \leq 10^{15}$ ，多组数据 $1 \leq t \leq 50000$ 。

算法 1 题目中的代价函数 $w(k) = \sum_{j=1}^k (1 + \lfloor \log_2 j \rfloor)$ 可以在 $O(\log k)$ 时间复杂度内计算。词典的代价显然和词典的 Trie 树有关，每个对代价有贡献的 01 串与词典的 Trie 树节点一一对应。设答案为 $f(n)$ ，考虑 Trie 树根节点左子树包含 k 个叶子节点，右子树包含 $n - k$ 个叶

⁴来源：XX Open Cup named after E.V. Pankratiev, Grand Prix of Warsaw

子节点，可以列出 DP 状态转移方程：

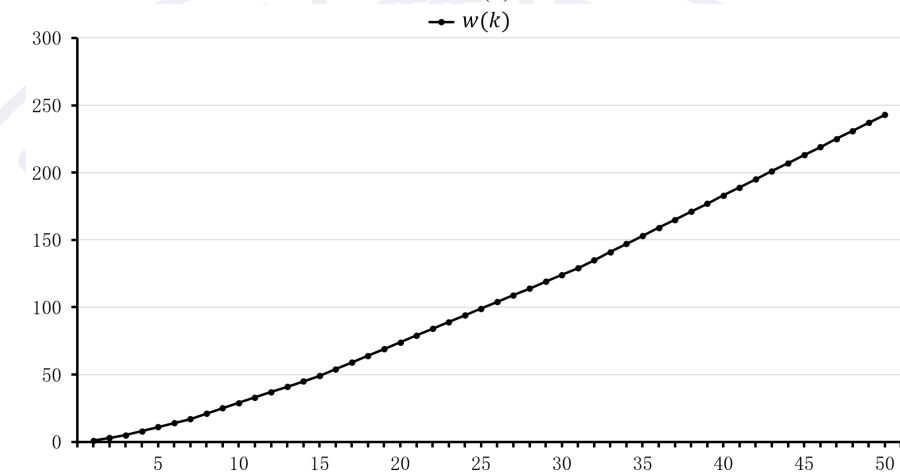
$$f(1) = w(1) = 1$$

$$f(n) = w(n) + \min_{1 \leq k \leq n-1} (w(k) \cdot [k > 1] + f(k) + f(n - k))$$

时间复杂度 $O(n^2)$ 。对这个算法进行优化需要分析性质。

引理 4.7. 代价函数 $w(k)$ 是下凸函数。

图 7: 函数 $w(k)$ 的曲线



证明. $w(k) - w(k - 1) = 1 + \lfloor \log_2 k \rfloor$ 。因为 $w(k)$ 差分序列单调不降，所以 $w(k)$ 是下凸函数。

□

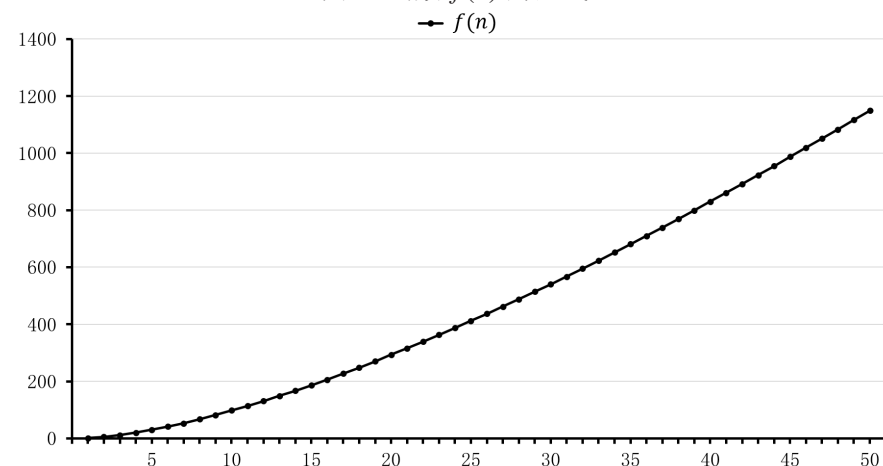
引理 4.8. DP 状态 $f(n)$ 是单调递增的下凸函数。

引理 4.9. 状态 $f(n)$ 的 DP 转移具有决策单调性，最优决策点 k_n 满足 $0 \leq k_n - k_{n-1} \leq 1$ 。

证明. 将两个引理一起证明，证明采用数学归纳法，两个引理的证明过程需要相互用到对方作为归纳假设。

由于最优转移点为 $k = 1$ 时 DP 转移方程具有特殊性，所以数学归纳法需要验证足够项数直到最优转移点 $k_n > 1$ 。

图 8: 函数 $f(n)$ 的曲线



n	$f(n)$	k_n	$f(n) - f(n-1)$
1	1		
2	5	1	4
3	11	1	6
4	20	1	9
5	30	2	10
6	41	3	11

表中列举了 $n \leq 6$ 时的 DP 状态 $f(n)$ 、最优转移点 k_n 、以及 DP 状态的差分序列，在这个范围内引理成立。设 $n_0 \geq 6$ ，假设引理 4.8 和引理 4.9 的结论对于 $2 \leq n \leq n_0$ 都成立，下证 $n = n_0 + 1$ 时结论同样成立。

设 $w(n)$ 的差分序列为 $dw(n)$ ，则 $dw(n)$ 单调不降。设 $f(n)$ 的差分序列为 $df(n) = f(n) - f(n-1)$ ，则在引理结论成立的范围内 $df(n)$ 单调不降。

先证明 $n_0 + 1$ 处的决策单调性，即证 $0 \leq k_{n_0+1} - k_{n_0} \leq 1$ 。

在计算 $f(n_0)$ 时，决策点 k_{n_0} 最优，那么从转移点 $k \neq k_{n_0}$ 转移与从最优转移点转移的代价之差大于零：

$$\begin{aligned} \Delta(n_0, k, k_{n_0}) &= (w(k) + f(k) + f(n_0 - k)) \\ &\quad - (w(k_{n_0}) + f(k_{n_0}) + f(n_0 - k_{n_0})) \\ &\geq 0 \end{aligned}$$

在计算 $f(n_0 + 1)$ 时，若 $k < k_{n_0}$ ，考虑从 k 转移和从 k_{n_0} 转移的代价之差：

$$\begin{aligned} \Delta(n_0 + 1, k, k_{n_0}) &= (w(k) + f(k) + f(n_0 + 1 - k)) \\ &\quad - (w(k_{n_0}) + f(k_{n_0}) + f(n_0 + 1 - k_{n_0})) \\ &= (w(k) + f(k) + f(n_0 - k)) - (w(k_{n_0}) + f(k_{n_0}) + f(n_0 - k_{n_0})) \\ &\quad + (f(n_0 + 1 - k) - f(n_0 - k)) - (f(n_0 + 1 - k_{n_0}) - f(n_0 - k_{n_0})) \\ &= \Delta(n_0, k, k_{n_0}) + df(n_0 + 1 - k) - df(n_0 + 1 - k_{n_0}) \\ &\geq 0 \end{aligned}$$

即从 $k < k_{n_0}$ 的任何转移点进行状态转移都不如从 $k = k_{n_0}$ 转移优，所以 $k_{n_0+1} \geq k_{n_0}$ 。

同理，若 $k > k_{n_0} + 1$ ，考虑从 k 转移和从 $k_{n_0} + 1$ 转移的代价之差：

$$\begin{aligned} \Delta(n_0 + 1, k, k_{n_0} + 1) &= (w(k) + f(k) + f(n_0 + 1 - k)) \\ &\quad - (w(k_{n_0} + 1) + f(k_{n_0} + 1) + f(n_0 - k_{n_0})) \\ &= (w(k - 1) + f(k - 1) + f(n_0 + 1 - k)) \\ &\quad - (w(k_{n_0}) + f(k_{n_0}) + f(n_0 - k_{n_0})) \\ &\quad + (w(k) - w(k - 1)) - (w(k_{n_0} + 1) - w(k_{n_0})) \\ &\quad + (f(k) - f(k - 1)) - (f(k_{n_0} + 1) - f(k_{n_0})) \\ &= \Delta(n_0, k - 1, k_{n_0}) + dw(k) - dw(k_{n_0} + 1) + df(k) - df(k_{n_0} + 1) \\ &\geq 0 \end{aligned}$$

即从 $k > k_{n_0} + 1$ 的任何转移点进行状态转移都不如从 $k = k_{n_0} + 1$ 转移优，所以 $k_{n_0+1} \leq k_{n_0} + 1$ 。结合起来，就得到了 $0 \leq k_{n_0+1} - k_{n_0} \leq 1$ ，即引理 4.9 在 n_0 处也成立。

接下来证明 $n_0 + 1$ 处的凸性，即 $df(n_0 + 1) \geq df(n_0)$ 。

若 $k_{n_0+1} = k_{n_0}$ ，则

$$\begin{aligned} df(n_0 + 1) &= f(n_0 + 1) - f(n_0) \\ &= (w(n_0 + 1) + w(k_{n_0}) + f(k_{n_0}) + f(n_0 + 1 - k_{n_0})) \\ &\quad - (w(n_0) + w(k_{n_0}) + f(k_{n_0}) + f(n_0 - k_{n_0})) \\ &= dw(n_0 + 1) + df(n_0 + 1 - k_{n_0}) \end{aligned}$$

若 $k_{n_0+1} = k_{n_0} + 1$ ，则

$$\begin{aligned} df(n_0 + 1) &= f(n_0 + 1) - f(n_0) \\ &= (w(n_0 + 1) + w(k_{n_0} + 1) + f(k_{n_0} + 1) + f(n_0 - k_{n_0})) \\ &\quad - (w(n_0) + w(k_{n_0}) + f(k_{n_0}) + f(n_0 - k_{n_0})) \\ &= dw(n_0 + 1) + dw(k_{n_0} + 1) + df(k_{n_0} + 1) \end{aligned}$$

所以

$$df(n_0 + 1) = \min \begin{cases} dw(n_0 + 1) + df(n_0 + 1 - k_{n_0}) \\ dw(n_0 + 1) + dw(k_{n_0} + 1) + df(k_{n_0} + 1) \end{cases}$$

同理 $df(n_0)$ 也可以写成上述形式

$$df(n_0) = \min \begin{cases} dw(n_0) + df(n_0 - k_{n_0-1}) \\ dw(n_0) + dw(k_{n_0-1} + 1) + df(k_{n_0-1} + 1) \end{cases}$$

观察比较 $df(n_0 + 1)$ 和 $df(n_0)$ 的公式，发现其中取 $\min$ 的两种情况可以对应比大小

$$\begin{aligned} dw(n_0 + 1) + df(n_0 + 1 - k_{n_0}) &\geq dw(n_0) + df(n_0 - k_{n_0-1}) \\ dw(n_0 + 1) + dw(k_{n_0} + 1) + df(k_{n_0} + 1) &\geq dw(n_0) + dw(k_{n_0-1} + 1) + df(k_{n_0-1} + 1) \end{aligned}$$

所以有 $df(n_0 + 1) \geq df(n_0)$ ，即 $f(n)$ 在 n_0 处也满足凸性。

根据数学归纳法，引理 4.8 和引理 4.9 得证。

□

算法 2 根据引理 4.2，算法 1 可以加上决策单调性优化，时间复杂度降低到 $O(n)$ 。

算法 2 已经比算法 1 有了大幅优化，但仍然无法通过本题 $n \leq 10^{15}$ 的数据范围。如果仍然采用传统思路优化 DP，则必然会求出 n 以内的所有状态，时间复杂度至少为 $O(n)$ ，即算法 2 已经无法再进一步优化了。那么如果要解决本题，必须找寻其他思路，避免求出所有 DP 状态。

算法 3 观察状态转移方程，发现其中取 $\min$ 的部分是三个关于 k 的下凸函数 $w(k)$ ， $f(k)$ 和 $f(n - k)$ 相加。下凸函数相加得到的函数也必然是下凸函数，所以可以在计算 $f(n)$ 时用三分法找出最优的 k 。三分法需要知道 $f(k)$ 和 $f(n - k)$ 的值，递归处理。打表发现最优的 $k_n \approx \frac{n}{3}$ ，这与观察词典的 Trie 树的直觉相符合。所以这样的递归方式计算 $f(n)$ ，其递归深度约为 $\log_{\frac{2}{3}} n = O(\log n)$ 。

算法 3 时间复杂度分析较为困难，且有很多地方可以加一些优化，例如基于 $k_n \approx \frac{n}{3}$ 优化三分法的初始范围等。可以验证，算法 3 的实际运行效率远高于 $O(n)$ 的算法 2，但仍然还不够。

引理 4.10. $w(n)$ 的渐进上下界是 $w(n) = \Theta(n \log n)$ 。

证明. 可以直接根据函数 $w(n)$ 的定义式进行代数不等式缩放。

渐进上界：

$$\begin{aligned} w(n) &= \sum_{j=1}^n (1 + \lfloor \log_2 j \rfloor) \\ &\leq n \cdot (1 + \lfloor \log_2 n \rfloor) \\ &= O(n \log n) \end{aligned}$$

渐进下界：

$$\begin{aligned} w(n) &= \sum_{j=1}^n (1 + \lfloor \log_2 j \rfloor) \\ &\geq \sum_{j=n/2}^n (1 + \lfloor \log_2 j \rfloor) \\ &\geq \frac{n}{2} \cdot (1 + \lfloor \log_2 \frac{n}{2} \rfloor) \\ &= \Omega(n \log n) \end{aligned}$$

综上 $w(n) = \Theta(n \log n)$ 。

□

引理 4.11. $f(n)$ 的渐进上下界是 $f(n) = \Theta(n \log^2 n)$ 。

证明. 直接利用状态转移方程不方便证明，但可以通过修改状态转移来分别确定 $f(n)$ 的渐进上界和渐进下界。

对于渐进上界，假设 DP 转移的不取最优的 k_n ，而取 $k_n^* = \lfloor \frac{n}{2} \rfloor$ ，设这样转移的状态记为 $f^*(n)$ 。显然 $f(n) \leq f^*(n)$ 。在不考虑 n 的奇偶性的情况下，根据状态转移列出：

$$\begin{aligned} f^*(n) &= w(n) + w\left(\frac{n}{2}\right) + 2f^*\left(\frac{n}{2}\right) \\ &= \Theta(n \log n) + 2f^*\left(\frac{n}{2}\right) \end{aligned}$$

将上式迭代 $\log_2 n$ 次展开并求和可得 $f^*(n) = \Theta(n \log^2 n)$ ，所以 $f(n) = O(n \log^2 n)$ 。

对于渐进下界，假设 DP 转移方程不含 $w(k)$ 这一项，新的状态记为 $f_*(n)$ ，转移方程 $f_*(n) = w(n) + \min_k (f_*(k) + f_*(n - k))$ 。显然 $f(n) \geq f_*(n)$ 。采用与引理 4.2 相同的方法可以证明 $f_*(n)$ 也是下凸函数。由于 $f_*(k) + f_*(n - k)$ 是关于 k 的下凸函数，且对称轴为直线 $k = \frac{n}{2}$ ，所以此状态转移方程的最优转移点就是 $k = \lfloor \frac{n}{2} \rfloor$ 。在不考虑 n 的奇偶性的情况下，根据状态转移列出：

$$\begin{aligned} f_*(n) &= w(n) + 2f_*\left(\frac{n}{2}\right) \\ &= \Theta(n \log n) + 2f_*\left(\frac{n}{2}\right) \end{aligned}$$

与渐进上界的 $f^*(n)$ 同理可得 $f_*(n) = \Theta(n \log n)$ ，所以 $f(n) = \Omega(n \log^2 n)$ 。

综上 $f(n) = \Theta(n \log^2 n)$ ，引理 4.11 得证。

□

引理 4.12. $f(n)$ 的差分序列 $df(n)$ 的渐进上下界为 $df(n) = \Theta(\log^2 n)$ 。

证明. 利用 $df(n)$ 的单调性构建估值：

$$f(n) - f(1) = \Theta(n \log^2 n) - 1$$

$$f(n) - f(1) = \sum_{k=2}^n df(k)$$

$$\leq (n-1) \cdot df(n)$$

$$f(2n) - f(1) = \sum_{k=2}^{2n} df(k)$$

$$\geq \sum_{k=n+1}^{2n} df(k)$$

$$\geq n \cdot df(n)$$

所以

$$df(n) \geq \frac{1}{n-1} \cdot \Theta(n \log^2 n)$$

$$= \Theta(\log^2 n)$$

$$df(n) \leq \frac{1}{n} \cdot \Theta((2n) \log^2(2n))$$

$$= \Theta(\log^2 n)$$

证得渐进上下界 $df(n) = \Theta(\log^2 n)$ 。

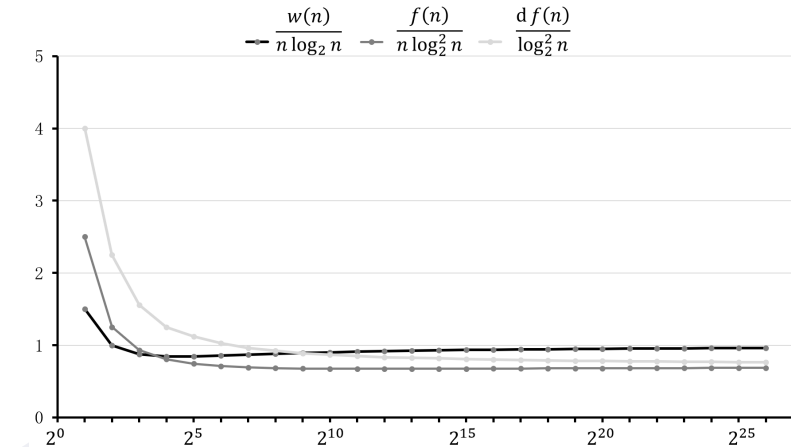
□

图 9 展示了引理 4.10、引理 4.11 和引理 4.12 中的结果。

算法 4 注意到 $df(n) = \Theta(\log^2 n)$ 而且单调递增，那么从 $df(2)$ 到 $df(n)$ 这 $n-1$ 项必然被划分为 $O(\log^2 n)$ 个连续段。考虑从小到大枚举整数 δ ，计算 $\delta = df(n)$ 的最大整数 n 。差分 $< \delta$ 的连续段端点已经算出，可以倍增的假设差分 $= \delta$ 的连续段右端点是 n 然后用算法 3 的三分法进行验证。为了避免验证 $f(n)$ 时递归求 $f(k)$ 和 $f(n-k)$ ，可以将倍增的倍数设为 $\frac{3}{2}$ ，这样只需查找 k 和 $n-k$ 属于哪一段即可验证，每次验证的时间复杂度不超过连续段的数量，即 $O(\log^2 n)$ 。求出差分 $= \delta$ 的连续段端点时间复杂度为 $O(\log^3 n)$ ，算法总时间复杂度为 $O(\log^5 n)$ 。

算法 5 经程序验证，本题数据范围 $n \leq 10^{15}$ 下 $\delta_{\max} < 1850$ 。所以实际做题时可以先在本地使用算法 4 求出所有连续段端点，然后在最终提交的程序中打表存储。

图 9: 函数 $w(n)$ ， $f(n)$ ， $df(n)$ 与渐进上下界的比值曲线，水平坐标轴为对数刻度



5 计算几何问题中的凸性证明

计算几何题目中常常包含凸多边形、凸包等，或者是以圆为主的二次曲线，这些几何图形都具有凸性。如果题目中的核心问题是伴随着这些几何图形而生的，则很可能也具有隐藏的凸性。这类题目的凸性证明则主要围绕几何图形本身的凸性，并可能需要用到少量数学技巧。

根据笔者的经验，题目本身如果是关于凸多边形，在证明前最好先考虑接下来要证明的性质是否对于任何凸的几何图形都成立，如果是，则在证明过程中要尽量避免使用多边形这一性质，从而避免或减少分段函数所引发的分类讨论。本章展示一个笔者认为具有代表性的例题。

5.1 一道例题

例题 5.1 (Easy Geometry ⁵). 坐标系中有一个 n 个顶点的凸多边形，在凸多边形内部构造一个面积尽量大的矩形，矩形的边必须平行于坐标轴。参考图 10。

数据范围 $3 \leq n \leq 10^5$ ， $|x_i|, |y_i| \leq 10^9$ ，坐标都是整数。

算法 1 三分套三分，外层三分矩形的左右宽度 w ，内层三分矩形的左边界坐标 l 。确定的对于 l 和 $r = l + w$ 求最大矩形面积，只需求出上边界和下边界的位置 u, d ，可以二分查找到 l, r 分别位于上轮廓和下轮廓的哪条边。算法总时间复杂度 $O(n + \log^3 n)$ ，其中 $O(n)$ 是读入和预处理的时间复杂度。

算法本身不难，但算法的正确性取决于两层三分法的函数是否是单峰函数。事实上，这两层三分法针对的都是上凸函数。

⁵来源：ICPC 2013-2014 NEERC

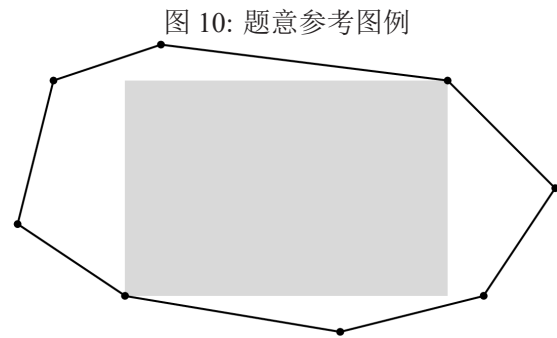


图 10: 题意参考图例

设 $s_w(l)$ 表示固定宽度 w 后左边界为 l 时的最大矩形面积。设 $h_w(l)$ 表示固定宽度 w 后左边界为 l 时的最大矩形高度。二者显然有以下关系：

$$s_w(l) = w \times h_w(l)$$

引理 5.1. $s_w(l)$ 是关于 l 的单峰函数。

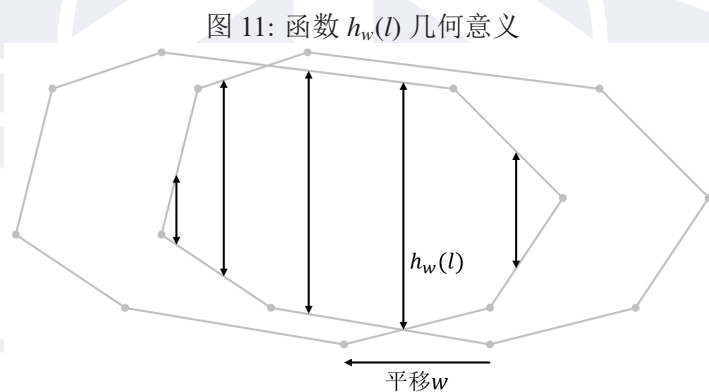


图 11: 函数 $h_w(l)$ 几何意义

证明. 函数 $h_w(l)$ 有如图 11 所示的几何意义：将原凸多边形自身和自身向左平移 w 的相交区域，直线 $x = l$ 被相交区域截取部分的长度。当相交区域非空时，相交区域必然是凸多边形。因此被直线 $x = l$ 截取部分长度 $h_w(l)$ 必然是关于 l 的上凸函数。而 $s_w(l)$ 等于上凸函数 $h_w(l)$ 乘常数 w ，必然也是上凸函数。

□

设 $S(w)$ 表示宽度为 w 的最大矩形面积。设 $H(w)$ 表示宽度为 w 时矩形的最大高度。显然有以下关系：

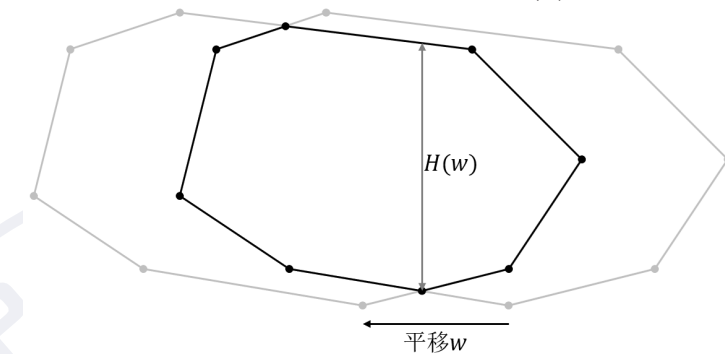
$$H(w) = \max_l h_w(l)$$

$$S(w) = w \times H(w)$$

引理 5.2. $H(w)$ 非严格单调递减。

证明. 根据几何意义， w 增大时的相交区域缩小，缩小后的相交区域被完全包含在原相交区域内。所以缩小后 $H(w)$ 对应的线段必然也在原相交区域内，因此 $H(w)$ 非严格单调递减。参考图 12 所示。

图 12: w 增大，相交区域缩小，函数 $H(w)$ 非严格递减



□

引理 5.3. $S(w)$ 和 $H(w)$ 都是上凸函数。

证明. 设原多边形上轮廓和下轮廓的纵坐标分别为上凸函数 $u(x)$ 和下凸函数 $d(x)$ ，那么 $h_w(l)$ 可以写成

$$\begin{aligned} h_w(l) &= \min(u(l), u(l+w)) - \max(d(l), d(l+w)) \\ &= \min \begin{cases} u(l) - d(l) \\ u(l) - d(l+w) \\ u(l+w) - d(l) \\ u(l+w) - d(l+w) \end{cases} \end{aligned}$$

其中取 $\min$ 的四个表达式均为关于 l 的上凸函数。

设实数 $0 < a < b$ 。设 $w = a$ 时 $h_w(l)$ 在 l_a 处取得最大值，设 $w = b$ 时 $h_w(l)$ 在 l_b 处取得最大值。那么

$$H(a) = h_a(l_a)$$

$$H(b) = h_b(l_b)$$

再设 $c = \frac{a+b}{2}$ ， $l_c = \frac{l_a+l_b}{2}$ 。当 $w = c$ 时 $h_w(l)$ 不一定在 l_c 处取得最大值，即

$$H(c) \geq h_c(l_c)$$

对于 $h_c(l_c)$ ，按取 $\min$ 的四种情况分别考虑，并利用函数 $u(x)$ 的上凸性质和 $d(x)$ 的下凸性质：

- 函数 $u(l) - d(l)$ ：与 w 无关，所以显然

$$2(u(l_c) - d(l_c)) \geq u(l_a) + u(l_b) - d(l_a) - d(l_b)$$

- 函数 $u(l) - d(l + w)$ ：

$$\begin{aligned} 2(u(l_c) - d(l_c + c)) &= 2u\left(\frac{l_a + l_b}{2}\right) - 2d\left(\frac{l_a + l_b}{2} + \frac{a + b}{2}\right) \\ &\geq u(l_a) + u(l_b) - d(l_a + a) - d(l_b + b) \end{aligned}$$

- 函数 $u(l + w) - d(l)$ ：

$$\begin{aligned} 2(u(l_c + c) - d(l_c)) &= 2u\left(\frac{l_a + l_b}{2} + \frac{a + b}{2}\right) - 2d\left(\frac{l_a + l_b}{2}\right) \\ &\geq u(l_a + a) + u(l_b + b) - d(l_a) - d(l_b) \end{aligned}$$

- 函数 $u(l + w) - d(l + w)$ ：

$$\begin{aligned} 2(u(l_c + c) - d(l_c + c)) &= 2u\left(\frac{l_a + l_b}{2} + \frac{a + b}{2}\right) - 2d\left(\frac{l_a + l_b}{2} + \frac{a + b}{2}\right) \\ &\geq u(l_a + a) + u(l_b + b) - d(l_a + a) - d(l_b + b) \end{aligned}$$

结合四种情况得

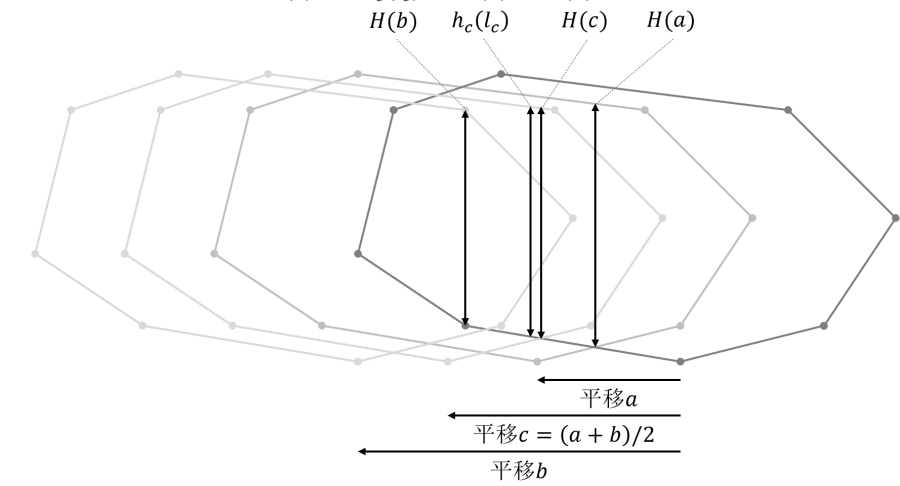
$$2h_c(l_c) \geq h_a(l_a) + h_b(l_b)$$

所以

$$\begin{aligned} 2H(c) &\geq 2h_c(l_c) \\ &\geq h_a(l_a) + h_b(l_b) \\ &= H(a) + H(b) \end{aligned}$$

所以函数 H 是上凸函数。以上过程可以参考图 13。

图 13: 图中展示 $2H(c) \geq 2h_c(l_c) \geq H(a) + H(b)$ 中的四个值之间的关系



又因为 $S(w) = w \times H(w)$ ，所以

$$\begin{aligned} 2S(c) &= c \cdot 2H(c) \\ &\geq c \cdot (H(a) + H(b)) \\ &= \frac{1}{2}(a + b)(H(a) + H(b)) \\ &= a \cdot H(a) + b \cdot H(b) + \frac{1}{2}(b - a)(H(a) - H(b)) \\ &\geq a \cdot H(a) + b \cdot H(b) \\ &= S(a) + S(b) \end{aligned}$$

所以 $S(w)$ 是关于 w 的上凸函数。

□

6 总结

凸性是数学性质，OI 题目中证明凸性与数学上的与凸性相关的证明题其实没有本质区别。如果一定要找出区别，则数学证明题的证明过程就是题目的答案，证明完成时问题就解决了；而 OI 题目的凸性证明仅仅只是为算法服务，证明完成只是解题过程的一个开始，后面还需要设计算法、优化算法、编写程序等很多环节。

本文阐述了三类凸性的证明方法，这是笔者自己学习积累的经验，并不代表证明凸性的方法只有这三类。笔者对于这三类证明方法只是基本掌握，不能保证遇到类似问题一定能够给出严谨的证明，事实上，在编写本论文时笔者多次遇到障碍，不知证明如何继续或者发现证明有漏洞需要修改，所幸这些困难最终都得以克服。学术研究是无止境的，笔者希

望以这篇论文作为抛砖引玉，期待以后更多有能力的竞赛选手提出其他证明方法。

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的指导。

感谢家人对我的陪伴与支持。

感谢董又铭老师、李翔老师、杜瑜皓、蒋凌宇在论文上为我提供的帮助。

感谢我的高中重庆市南开中学校、初中重庆南开（融侨）中学校对我的栽培。

感谢其他给予我帮助的老师与同学。

参考文献

- [1] 四边形不等式优化. <https://oi-wiki.org/dp/opt/quadrangle/>, 2024.
- [2] 费用流. <https://oi-wiki.org/graph/flow/min-cost/>, 2024.
- [3] 郭羽冲. 浅谈函数的凸性在 oi 中的应用. 2023 年信息学奥林匹克中国国家集训队论文集, 2023.

浅谈树上邻域问题的一种基于长链剖分的解法

中山纪念中学 刘海峰

摘要

本文从长链剖分角度出发，介绍了静态情况下长链剖分对树上常见的四种邻域问题的处理方法，其中包括了离线情况和在线情况，并利用长链剖分将部分关于邻域的问题在线性复杂度内转化为了序列上的区间查询问题和树上的树链查询问题。

1 前言

静态树上邻域问题常见的处理方法是点分治和静态 Top tree。长链剖分作为一个常用技巧，通常用于优化一些和深度相关的问题，而邻域的结构也和深度相关，所以理论上也可以尝试长链剖分。本文将探讨长链剖分对于各种静态树上邻域问题的处理。

2 前置知识

基本的树论知识。

3 一些约定 & 定义

下文中的树默认是一个 n 个点的有根树，其中根节点为 1，并且每条边的距离都是 1，并且无修改操作。

为了方便，下文所称的“半群”在非特殊说明下都默认为是交换半群，“群”在非特殊说明下都默认是交换群。

定义 3.1 (深度). 定义一个节点的深度为该节点到根节点的树上最短距离 +1。

设 fa_x 表示 x 的父亲节点。

3.1 长链剖分基本定义

定义 3.2 (高度). 定义一个节点的高度为该节点子树内距离该节点最远的点到该节点的距离 +1，叶子的高度是 1， x 节点的高度记作 h_x 。

定义 3.3 (长儿子). 定义一个节点的长儿子所有儿子中高度最大的儿子，如果有多个可以任选一个，如果是叶子则无长儿子。

定义 3.4 (短儿子). 定义一个节点的短儿子为不是长儿子的那些儿子构成的集合。

定义 3.5 (长链). 对于所有非叶节点只保留到长儿子的边之后构成的那些链（可以为单点）称之为是长链。定义长链的长度为其包含的节点数。

定理 3.1 (长链剖分基本定理). 设 $f(x)$ 为点 x 的短儿子的高度和。 $\sum_{i=1}^n f(i) = n - \text{根节点的高度}$ 。

证明. 对 $\sum_{i=1}^n f(i)$ 进行贡献拆解之后可以发现，只有根节点所在的长链没有被算进去，其它的长链都恰好被算了一次。□

3.2 邻域相关定义

注意，本部分很多定义的名字是作者自己取的。

定义 3.6 (邻域). 定于二元组 (x, d) 表示的邻域为树上距离 $x \leq d$ 的点构成的点集。

定义 3.7 (内邻域). 定义二元组 (x, d) 表示的内邻域为 x 子树内距离 $x \leq d$ 的点构成的点集。有些人也会把这个称之为是有向邻域。

定义 3.8 (外邻域). 定义二元组 (x, d) 表示的外邻域为 x 子树外距离 $x \leq d$ 的点构成的点集。

定义 3.9 (链邻域). 定义三元组 (x, y, d) （要求 y 是 x 的祖先）表示的链邻域为所有节点满足：

1. 在 x 子树外且在 y 子树内。
2. 存在 x 到 y 路径上的一个点到该点的距离 $\leq d$ 。

构成的点集。

接下来的两节，我们将处理离线情况下的各种邻域的群和半群信息查询。

4 群信息查询

为了方便，不妨假设本部分我们将讨论求各种邻域的点权和的做法。本部分将会在离线的前提下把这些各种邻域问题做到线性。

4.1 内邻域

我们对于一条长度为 m 、链顶节点为 x 的长链，我们先预处理出 x 子树内距离 $x = i$ 的那些点的权值和。根据长链剖分基本定理，这部分直接暴力处理时间复杂度为线性。

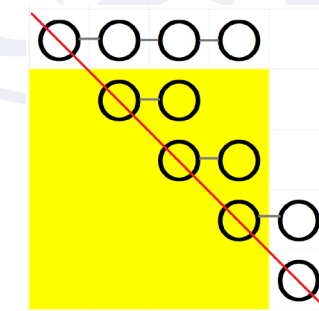
然后现在我们对于每条长链进行考虑，我们对于一个点 x ，我们预处理出一个长度为其所有短儿子高度的最大值的数组，其中第 i 项表示所有短儿子的子树中距离 $x = i$ 的点的点权和。

然后我们就可以把树看作是若干条长链，然后每个点除了所在长链外还可能挂了一条额外的链表示短儿子内部的信息，并且点数之和是 $O(n)$ 的，于是我们可以对于每条长链分别进行了处理。

我们直接从链底向上扫（不妨假设链低编号为 m ，链顶编号为 1），然后维护当前距离链顶为 i 的点权和，可以发现，扫到 i 号点的时候会更改 $i-1$ 以及向后的若干个位置，并且更改量总和是 $O(n)$ 的。然后我们把询问挂到对应的长链的点上查询，相当于若干次查区间和，可以用线段树等数据结构进行维护。但注意到我们可以维护后缀和，所以我们可以直接做到线性。

考虑到我们本质上要做的是，给定一个类似如下图的结构，然后查询形如给定一个点，求这个点左下方的点权和。

图 1: 一个例子，黄色区域表示查询的区域，红线上的点为长链上的点，灰色的链表示每个点挂的额外的链



一个想法是，对于查询的 (x, y) （表示第 x 行第 y 列），我们先快速求出其下方第一个和他所在的列相同并且有值的位置 (x', y) ，然后提前预处理这类位置的答案，该位置向上到原位置显然就是 $[x, x')$ 这些行的所有点的权值和。预处理所有有值位置的答案是容易的，考虑对于 (x, y) 如何找到其下方第一个和他所在的列相同并且有值的位置，我们将强制在线的问题转化成该问题，该问题的解决方法将在第六节内讲述。

4.2 邻域

外邻域可以转化成邻域减去内邻域，所以我们只需要考虑邻域的情况。

对于一个邻域 (x, d) ，如果将其变成 $(fa_x, d-1)$ ，那么 x 子树外的点所否在这个邻域内的情况是没有改变的，如果 $d \geq h_x + 1$ ，那么邻域 $(x, d), (fa_x, d-1)$ 都包含了 x 子树内的所有点，那么我们转化成求 $(fa_x, d-1)$ 是不会改变答案的。

我们不断父亲直到不能跳为止。

直接跳可以采取倍增，不过复杂度是 $O(\log n)$ 的，不优美，然而该问题做起来比较困难，我们实际上可以适当放宽限制。

定义 4.1 (跳链操作). 对于 (x, d) ，找到 x 最近的一个祖先 y 满足 y 所在的长链长度 $\times 2 \geq d$ ，然后跳到 y 位置，并将 d 减去跳的深度差，可以发现，得到的 (y, d') 满足 $d' \leq 2 \times y$ 所在长链长度。

接下来我们将证明跳链操作换成一步一步向上跳，即 (x, d) 每次变成 $(fa_x, d-1)$ 直到 $x = y$ 之前时都有 $d \geq h_x + 1$ 。

证明. 注意到 x 子树中任何一个点到 x 的距离均 $< x$ 的高度，而对于 x 到 y 路径上除 y 之外的点 z ，都满足其所在的长链长度 $\times 2 \geq d$ ，也就是说从 x 跳到 z 时， d 的减小少量小于 z 所在长链长度，所以满足条件。 $\square$

该问题本质上来说就是给定 x, k ，找到最近的祖先 y 满足 y 所在长链长度 $\geq k$ 。考虑允许离线的情况下怎么做，我们 DFS 并开 n 个栈，第 i 个栈表示当前 $d = i$ 的时候 x 应该跳到哪一条长链上。DFS 过程中我们优先 DFS 长儿子，当 DFS 到一个短儿子的时候，设这个短儿子所在的长链编号为 k ，长度为 m ，我们在第 $1 \sim m$ 个栈中压入 k ，在完成这个短儿子的 DFS 之后再弹出第 $1 \sim m$ 个栈的栈顶。然后我们就可以快速知道最后这个点会在哪条长链上。

我们在 DFS 过程中再维护一点简单的信息就可以确定 (x, d) 邻域经过跳链操作之后得到的结果。

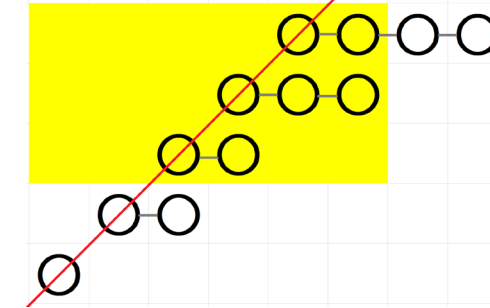
然后对于每条长链的顶端节点 x ，如果这条长链长度为 m ，则我们再预处理出外邻域 $(x, 1), (x, 2), \dots, (x, 2m)$ 的答案。

如果我们能够预处理出来，那么可以发现，处于该长链的所有满足 $d \leq 2 \times$ 该长链长度的外邻域询问都可以像内邻域那样用同样的方式进行处理，具体来说，我们把贡献拆成链顶子树内的和链顶子树外的，而链顶子树外的（也就是链顶的一个外邻域）已经被预处理了，而链顶子树内的情况我们可以像内邻域那样求出该链链顶子树内的答案，如图：

显然，我们只查询了删去 x 这个短儿子时 fa_x 的其它短儿子的大小为 $1 \sim O(m)$ 内邻域半群信息，这个可以直接暴力做。

再拼上预处理的链顶节点的一个外邻域答案即可。现在变成了如何处理顶端节点的外邻域答案，简单观察一下，对于一个顶端节点 x 和其父亲节点 y ，外邻域 (x, d) 可以转化成

图 2: 一个例子，各种颜色所表示的含义同图 1



外邻域 $(y, d-1)$ 再加上链邻域 $(x, y, d-1)$ （本质上是 y 子树除去 x 子树之后的一个内邻域），可减情况下这个可以通过容斥来实现。

可以发现，预处理的整个过程复杂度都是 $O(n)$ 的，对于 Q 次询问，离线下来并处理的复杂度是 $O(n+Q)$ 的。

4.3 链邻域

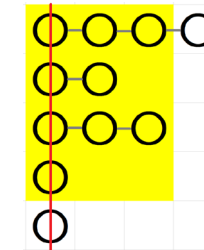
做法和邻域类似。设 $A(x, d)$ 为链邻域 $(x, 1, d)$ 拼上内邻域 (x, d) 的权值和。

可以发现，如果 $d > x$ 所在长链长度，那么有 $A(x, d) = A(fa_x, d)$ 。于是我们类似邻域那样做一个跳链操作，只不过此时要求的是找到最近的祖先 y 满足 $d \leq y$ 所在长链长度。

然后对于每一条长链，设链顶节点为 x ，链长为 m ，我们预处理出链邻域 $(x, 1, 1), (x, 1, 2), \dots, (x, 1, m)$ 的值，具体思路还是将链邻域 $(x, 1, i)$ 转化成链邻域 $(fa_x, i, 1)$ 再加上内邻域 (fa_x, i) 减去内邻域 $(x, i-1)$ 。

如果查询的链邻域 $(x, 1, d)$ 满足 $d \leq x$ 所在长链长度，设 y 为 x 所在长链链顶节点，那么我们可以将询问拆成链邻域 (x, y, d) 和 $(y, 1, d)$ ，其中链邻域 $(y, 1, d)$ 已经被预处理，对于链邻域 (x, y, d) ，也就是查询 x 到其长链链顶这部分的贡献，本质上就是查询图所示的结构点权和：

图 3: 一个例子，各种颜色所表示的含义同图 1



于是我们可以先求出 $A(x, d)$ 再减去内邻域 (x, d) 即可，复杂度依然是线性。

5 半群信息查询

本部分我们的目标是在 $O(n + Q)$ （其中 Q 为询问次数）的复杂度下将内邻域和外邻域交换半群信息查询问题转化为序列上的区间半群信息查询¹，将链邻域交换半群信息查询转化为树上链半群信息查询。显然，对于内邻域和外邻域的半群查询，这是不弱于区间半群信息查询的，对于链邻域的半群信息查询，这是不弱于树链半群信息查询的。

5.1 内邻域

回顾一下内邻域查询的本质（详见图 1），我们考虑如何在不用减法的情况下解决问题。

对于查询的 (x, y) （表示第 x 行第 y 列），我们先快速求出其下方第一个和他所在的列相同并且有值的位置 (x', y) ，然后提前预处理这类位置的答案，显然预处理这部分的答案是不需要利用可减性的，至于如何找到对应的 x' ，在允许离线的前提下，直接从下往上扫即可，对每个 y 用桶记录当前访问过的最小的 x 即可，然后再求出第 $[x, x']$ 行的半群信息，合并起来就是答案。

5.2 邻域

我们同可减情况一样做跳链操作，考虑做完操作之后相当于查对应的外邻域和内邻域拼起来的结果。

5.3 外邻域

我们先回到可减情况下预处理，可以发现，预处理的时候有一个部分用到了减法。具体来说，对于一条长度为 m 、链顶节点为 x 的长链，我们预处理了 fa_x 去掉 x 之后的 $1 O(m)$ 的内邻域半群信息，更加具体的来说，是查询链邻域 $(x, fa_x, 1), (x, fa_x, 2), \dots, (x, fa_x, O(m))$ 的半群信息。

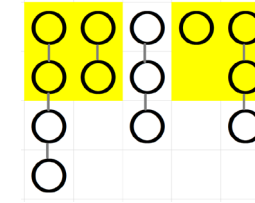
这个其实可以不用可减性的，设 y 为 fa_x 的长儿子，预处理出内邻域 $(y, 1), (y, 2), \dots, (y, O(m))$ 的半群信息，然后我们只需要考虑 fa_x 的其余的短儿子的内邻域半群信息了，虽说比较简单但还是值得一提，具体来说，把 fa_x 的所有短儿子拎出来，然后进行若干次如图 4 所示的查询：

可以看出，最直接的方法是拆成前后缀来查，这里比较简单，直接暴力即可。

回到可减的情况下邻域的做法，具体来说，如果外邻域 (x, d) 满足 $d \leq 2 \times$ 当前点所在长链长度，那就直接做，不然还是和之前那样，设 x 最后会跳到长链 i 上，那么考虑找到最近的祖先 y 满足 y 所在长链长度 $\times 2 \geq d$ ，设 $d' = d - x, y$ 高度差，我们先求出外邻域 (y, d') 的值，我们

¹<https://www.luogu.com.cn/problem/P11265>

图 4: 一个例子，各种颜色所表示的含义同图 1

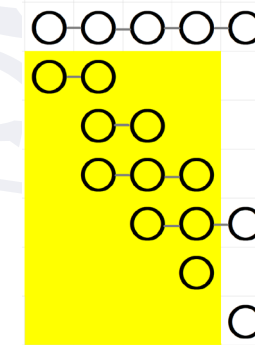


还要预处理出 y 的所有儿子中是 x 祖先的那个儿子（设为 z ），这个可以询问离线然后一次 DFS 直接算出。

于是我们现在还要计算 y 子树删除 z 子树后大小为 d' 内邻域的半群信息，其实就是链邻域 (z, y, d') ，处理方法也比较类似，我们先处理掉 y 的长儿子的贡献，然后考虑短儿子的贡献，查询内容同图 3 但不保证 d' 和 z 所在长链长度同阶，不过我们可以用类似 5.1 中讲述的方法进行处理。

当然，这个做法实现起来比较麻烦，有一种稍微简单一点的做法，具体来说，对于一条长链，我们对其所有短儿子进行编号，满足深度浅的短儿子编号更小，深度相同可以随意钦定编号，然后我们对于 (x, d) ，我们先直接找到 y, z, d' （含义同上），并求出 z 所在的儿子的编号，然后算出 $> z$ 的短儿子子树的贡献（具体来说，对于每个短儿子，求出其子树内到 y 距离 $\leq d'$ 的点的半群信息）、 $< z$ 的短儿子的贡献（两侧是对称的）以及 y 所在的链顶的子树外的贡献（这部分显然已经被预处理了），我们只考虑 $> z$ 部分的贡献，这里放一个查询的例子：

图 5: 一个例子，这里只画了所有短儿子对应的贡献



查询本质上和 5.1 所述情况相同。

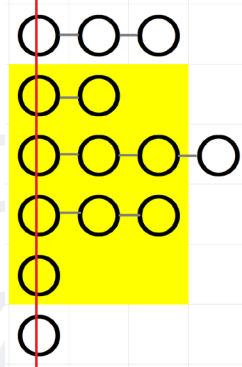
然后如果从 a 跳到了 b ，我们还需要加上 b 子树除去 a 子树的半群信息，这个直接转到 DFS 序上就是两个区间的半群信息查询。

于是综上，我们成功把离线内邻域和外邻域的半群信息查询通过 $O(n + Q)$ 的预处理转化成了序列上的半群信息查询。

5.4 链邻域

注意到如果是到根的链的话做法本质上和外邻域是相同的。
先考虑一个简单一点的情况，如果 x, y 在同一条长链内该怎么做。
类似图 1，画出对应的结构和查询区域，如图 6：

图 6: 一个例子，各种颜色所表示的含义同图 1



可以发现，因为信息不可减，我们还是可以像内邻域那样，假设是查询第 $[l, r]$ 行前 d 列的半群信息，我们可以找到第 d 列第 l 行向下第一个有值的位置和第 r 行向上第一个有值的位置，因为有值的位置的总数量为 $O(n)$ 的，所以我们可以对每一列，用数据结构维护所有有值的位置之间的区间半群信息，这个是容易的。

考虑设 g_x 为 x 的所有短儿子的高度最大值，如果 x 不存在短儿子则 $g_x = 0$ 。

考虑构造一个该树的生成森林，具体的，该生成森林有 n 层，所有的边所在的两个端点都在同一层中，对于原树上的点 x ，在该生成森林的第 $1 \sim g_x$ 层各有一个编号为 x 的点，可以发现，总点数 $\leq n$ 。对于 x ，对于第 i 层的点 x （显然有 $g_x \geq i$ ），找到 x 在原树上最近的祖先 y 满足 $g_y \geq i$ ，然后向 y 对应的第 i 层的点连边。

对于第 i 层一条 x 连向 y 的边，设其边权为链邻域 (x, y, d) 的半群信息。显然一种想法是暴力处理，不过后文将提到更优秀的处理方法，我们这里先考虑求出该信息之后该怎么做。

对于链邻域 (x, y, d) ，我们还是先做跳链操作，使得 x 所在长链长度 $\geq d$ ，跳过的部分的贡献可以类似 5.4 那样进行计算。

然后不妨假设现在的询问满足 x 所在长链长度 $\geq d$ ，我们令 z 为 x 所在长链链顶节点，然后先快速求出 (x, fa_z, d) （链邻域 (x, fa_z, d) 可以转化为链邻域 (x, z, d) 拼上链邻域 (z, fa_z, d) ，前者是好做的，后者在外邻域部分已经被预处理了），然后变成求 (fa_z, y, d) 。

下文不妨假设我们求的依然是 (x, y, d) ，但 $g_x \geq d$ 并且 (x, y) 不在同一条长链上。

我们再找到 y 到 x 路径上第一次走到的短儿子 k ，那么 $g_{fa_k} \geq d$ ，于是我们可以通过实现链半群信息查询快速求出 (x, fa_k, d) ，然后此时 fa_k, y 处于同一条长链上了，再使用之前处于同一条链时的做法即可。

接下来我们将讨论生成森林中所有边查询的半群信息的求法。

可以发现，如果第 i 层中 x 向 y 连边了，那么 x, y 之间的其它点的 g 值均 $\leq i$ 。

不妨假设 x, y 在同一条长链上（如果 x, y 不在同一条长链上，那么 y 显然就是 x 的链顶节点的父亲），该问题本质上就是维护这么一个数据结构：

- 对一个前缀每个位置加上一个指定的半群，并对这个前缀以后的部分再加上固定的半群。
- 查询一个前缀的所有位置半群信息，并将查询的位置清空。

并且保证总修改量是 $O(n)$ 的，我们考虑懒标记，对于一个对后缀加半群的操作，我们直接在对应位置打上一个懒标记，然后在做前缀操作的时候，我们把懒标记进行后推，这样复杂度就是线性了。

6 其它的情况

对于部分问题，信息是不可拆解的，换句话讲，我们对于子树 x ，我们只能维护 x 子树内距离 $x \leq i$ 的那些点构成的信息，而不能单独维护距离 $x = i$ 的那些点的信息，在一些转移能够快速合并并且满足结合律的情况下，该问题还是可以轻松计算的。下面这题是一个例子。

例题 6.1.² 给定一棵点带权的树，每次询问给定 x, d ，求内邻域 (x, d) 的最大权独立集。

我们还是跟之前一样，对于每条长链，对于长链范围内的每个 i 求出距离链顶 $\leq i$ 的点构成的邻域的信息，具体来说，对于每个 x ，我们先求出 x 除去长儿子的子树之后剩余的子树的所有内邻域的信息，然后再由长儿子的 dp 进行转移。

具体来讲，在一条长链上，我们修改到链顶节点距离为 i 的点时，相当于对 i 开始的一段前缀（设长度为 j 、即对 $i \sim i + j - 1$ 进行暴力修改），然后再对 $i + j$ 以及之后的部分做一个乘上同一个矩阵，类似于 5-4，这个可以通过懒标记来实现，每次直接向后推懒标记即可。

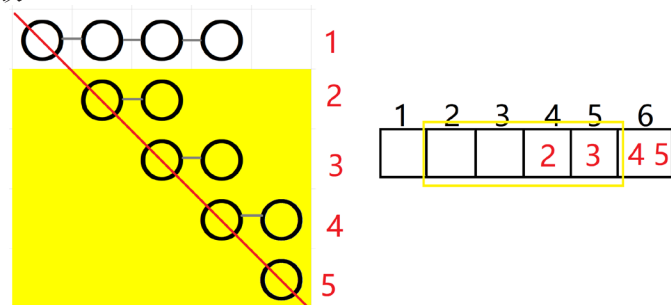
然后是查询，内邻域具体查询示意图详见图 1。

可以发现，所有有值的位置的信息左下方的信息都可以在对应的修改过程中得出。然后同样的，转化成一个区间的矩阵乘法再加上一个有值的位置的左下方的信息。

事实上，还可以从另一个角度思考该问题的做法，具体来说，我们对一个位置进行查询的时候，需要讲这个位置前面的懒标记全部推到这个位置后面，当然我们还可以直接查该位置的值以及该位置之前的所有懒标记按顺序合并之后的结果。注意到一个懒标记相当于一个点的除去长儿子的子树之后剩余的子树的信息，所以本质和上述做法相同。为了让读者更好理解，这里放一张示意图：

² 本题无来源

图 7: 一个例子, 右图表示从第五行扫到第二行时懒标记所处的位置, 数组开了六个位置是为了方便标记存放



在该问题中, 扫到第二行时直接查询 $(2, 5)$ 位置的结果, 因为前面存在没有处理的懒标记, 所以查询到的时 $(4, 5)$ 位置的值, 此时还要再算上第三行和第二行对答案的贡献。

对于其余的邻域情况类似, 不过比较繁琐, 所以这里留作读者自行思考。

7 强制在线

本部分需要读者掌握在线 $O(n) - O(1)$ 最近公共祖先以及在线 $O(n) - O(1)$ 树上 k 级祖先。本部分将讨论强制在线情况下之前所属的各种邻域问题的解决方法。

7.1 序列前驱问题

接下来我们先探讨一个简单的问题:

给定正整数序列 $a_1, \dots, a_n$, 有若干次询问, 每次询问给定 i, x , 你需要找到最大的 j 满足 $j \leq i, a_j \geq x$ 或报告不存在, 强制在线。设 $S = \sum_{i=1}^n a_i$, 要求 $O(n + S)$ 预处理 $O(1)$ 查询。

允许离线情况下, 我们只需要对下标扫描线, 用桶维护最后依次访问到 i 的位置即可, 扫到 i 的时候暴力更新 $1 \sim a_i$ 这些位置。

对于在线的情况, 我们考虑建一个 S 个点的图, 其中对于 $i = 1 \sim n$, 建立 a_i 个点, 分别为 $(i, 1), (i, 2), \dots, (i, a_i)$ 。

然后对于每个存在的点 (x, y) , 找到最大的 z 满足 $z \leq x$ 并且 $(z, y + 1)$ 存在, 然后 (x, y) 向 $(z, y + 1)$ 连边, 如果不存在则不连边。不难发现, 这构成了一棵内向树森林。

不难发现, 对于每次询问, 我们只需要找到 $(i, 1)$ 的 $x-1$ 级祖先即可, 这可以用 $O(n) - O(1)$ 树上 k 级祖先来实现。

7.2 跳链方面的问题

之前的问题无论是外邻域还是链邻域, 都需要做一个操作: 对于 x , 找到最近的祖先 y 满足 y 所在的长链长度大于等于一个指定的数 k 。该问题离线下的做法已经在 4.2 中讲述, 接下来尝试在线做。

具体来说, 我们对于一个已经长链剖分好的树, 我们将每个长链看作一个点, 长链 a 向长链 b 连边当且仅当 a 链顶的父亲节点为于长链 b 上。

事实上, 做到这里, 该问题已经变成了树上前驱问题, 具体做法根序列前驱问题相同, 对于点 x , 设其代表的长链长度为 a_x , 则建立节点 $(x, 1), (x, 2), \dots, (x, a_x)$, 然后对于 (x, y) , 找到 x 最近的祖先 (可以是 x) z 满足 $(z, y + 1)$ 存在, 然后 (x, y) 向 $(z, y + 1)$ 连边, 然后查询的时候找到 x 所在长链 i , 求 $(i, 1)$ 的 $k - 1$ 级祖先即可。

注意我们现在求出的内容是 x 会一直跳直到跳到某条置顶的长链的链上节点时会停止, 为了求出这个点是哪个点, 我们可以求出该链链底节点与 x 的 LCA。

7.3 各种邻域问题求法

同样的, 我们的目标是用 $O(n) - O(1)$ 的复杂度把问题转化成在线区间半群信息查询和在线树链半群信息查询。

回到内邻域问题的求法, 我们的目标相当于是对于查询 (x, y) , 如何找到其下方第一个和他所在的列相同并且有值的位置。

考虑该问题可以转化为树上 k 级祖先问题, 和 7.1 的做法类似。具体来说, 对于每个有值的点 (x, y) , 找到最小的 i 满足 $(i, y + 1)$ 有值且 $i \geq x$ 并向 $(i, y + 1)$ 连边 (如果不存在这样的 i 则不连边)。这样的话, 对于查询, 我们可以先找到 (x, x) (显然一定有值), 然后查询其 $y - x$ 级祖先即可, 使用在线 $O(n) - O(1)$ 树上 k 级祖先可以解决此问题。剩余部分都可以通过序列上和树上的半群信息查询实现。

对于外邻域和链邻域问题, 我们还是先处理好跳链操作。可以发现, 外邻域和链邻域的查询都可以用类似的方法来做强制在线。

8 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢中山纪念中学和三鑫学校提供优秀的信息学学习环境。

感谢家人、朋友、教练对我的支持与鼓励。

感谢赵海鲲同学、许淇文同学、庞佳一同学为本文提出宝贵意见。

参考文献

[1] 《算法艺术与信息学竞赛》刘汝佳

转置原理在一类动态规划问题中的应用

华中师范大学第一附属中学 徐骁扬

摘要

转置原理 (Transposition Principle)，是一种利用矩阵转置改写线性变换算法的思想，可以有效优化部分线性变换算法问题的时空复杂度。本文将其引入了动态规划 (Dynamic Programming, DP) 问题的求解中，将转置原理与部分线段树上操作相结合，并借由两道例题具体说明了利用转置原理在解决线段树优化动态规划问题中的优势，提供了一种新的求解动态规划问题的思路。

1 引言

转置原理自上个世纪起被用于优化线性变换算法，其将线性变换算法和该算法转置紧密联系在一起。在此之前，大多数线性变换算法与该算法转置都是分别进行优化的（例如多项式乘法和它的转置）。而转置原理可以说明，这两个问题是在几乎相同的时空复杂度之内解决。因此，在解决与优化线性变换算法的同时，也能够解决与优化该线性变换算法的转置，这样能够简化部分问题的求解。

对于较为简单的问题，直接优化是较为容易的，所以转置原理的优势在于较为困难的问题中。又因为转置原理只能解决线性变换算法相关问题，这导致目前使用转置原理解决的算法问题是较少的，且基本都与生成函数和多项式运算相关。

而转置原理在线段树合并优化 DP 问题上的应用还未见于文献，因此本文将对此问题展开研讨。

2 转置原理

2.1 简介

定义 2.1 (线性变换算法). 对于给定的 $m \times n$ 矩阵 A ，对于任意的 $n \times 1$ 向量 a 均可计算并输出乘积 $b = Aa$ 的算法，被称为针对矩阵 A 的**线性变换算法**。

很显然，朴素的矩阵乘法就是一种线性变换算法，但是线性变换算法还包括其他的实现形式。

在针对矩阵 A 的线性变换算法中， $\mathbf{a}$ 可以是 $\mathbb{R}^n$ 内的任意向量，与算法的具体操作无关。

例如对于后缀和算法，就有矩阵 A 为上三角矩阵 $\begin{bmatrix} 1 & 1 & \dots & 1 \\ & 1 & \dots & 1 \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix}$ 。算法的实现和向量 $\mathbf{a}$

无关，只与矩阵 A 有关。

定义 2.2 (线性变换算法的转置). 对于针对给定矩阵 A 的线性变换算法 f ，称针对转置矩阵 A^T 的线性变换算法 g 为算法 f 的转置。即算法 g 为对于任意 $m \times 1$ 向量 $\mathbf{a}'$ 均可计算并输出乘积 $\mathbf{b}' = A^T \mathbf{a}'$ 的算法。

作为算法 f 的转置的输入， $\mathbf{a}'$ 是一组与 $\mathbf{a}$ 没有数值关联的向量。

仍以后缀和算法为例，有 A 的转置 $A^T = \begin{bmatrix} 1 & & & \\ \vdots & \ddots & & \\ 1 & \dots & 1 & \\ 1 & \dots & 1 & 1 \end{bmatrix}$ ，这个算法与向量 $\mathbf{a}'$ 的取值无

关。容易看到该算法为前缀和算法。在这种意义上，可以认为后缀和算法的转置是前缀和算法。

根据转置原理 [1]，可以将一个线性变换算法改写成其转置后的算法，同时保持时空复杂度不变。因此通过对算法转置进行优化，即可实现对原算法的优化。

在上面的例子中，如果后缀和算法是难以优化的，但是前缀和算法是很容易优化的，就可以通过将前缀和算法优化后进行转置，来实现对后缀和算法的优化。

2.2 算法改写

为了便于处理，可以通过对 A 添加若干行和若干列，使得其变成一个可逆的方阵。具体的，通过添加 n 行和 m 列，构造出矩阵 $\begin{bmatrix} A & E_m \\ E_n & O \end{bmatrix}$ (其中 E_n 表示 n 阶单位矩阵， O 为零矩阵)。同时向量 $\mathbf{a}$ 会添加 m 行，但这 m 个值不由输入给出，可以设置为 0，不会影响 $\mathbf{b}$ 前 m 行的值，而 $\mathbf{b}$ 的后 n 行刚好就是原先的 $\mathbf{a}$ 。容易发现，额外添加的部分不会影响算法的正确性和时空复杂度。在未特别说明的情况下均默认矩阵 A 是可逆的。

定义 2.3. 定义如下的三种矩阵为初等矩阵：

1. 把单位矩阵中的第 i 行和第 j 行交换得到的矩阵。
2. 单位矩阵以数 $k \neq 0$ 乘第 i 行得到的矩阵。
3. 以 k 乘单位矩阵的第 j 行加到第 i 行得到的矩阵。

根据可逆矩阵的性质 [2]，可以将给定的任意可逆方阵 A 拆成若干个初等矩阵的乘积，即存在初等矩阵 $V_1, V_2, \dots, V_k$ ，使得 $A = V_1 V_2 \dots V_k$ 。

在此列举将三种初等矩阵左乘到 $\mathbf{a} = [a_1, a_2 \dots a_n]^T$ 上的操作：

1. 交换 a_i 与 a_j ，将这个操作对应的矩阵记为 $P_1(a_i, a_j)$ ，简记为 $a_i \leftrightarrow a_j$ 。
2. a_i 变为 $k \times a_i$ ，将这个操作对应的矩阵记为 $P_2(a_i, k)$ ，简记为 $a_i \leftarrow k \times a_i$ 。
3. a_i 加上 $k \times a_j$ ，将这个操作对应的矩阵记为 $P_3(a_i, a_j, k)$ ，简记为 $a_i \leftarrow a_i + k \times a_j$ 。

所以对于所有的线性变换算法，都可以表示成以上三种操作的组合。

又因为 $A^T = V_k^T V_{k-1}^T \dots V_1^T$ ，所以如果存在求解 $\mathbf{b}' = A^T \mathbf{a}'$ 的算法，其每一步都可以被表示成初等矩阵，那么就可以通过逆序执行所有的步骤的转置，得到 $\mathbf{b} = A \mathbf{a}$ 的算法。

接下来考察三种初等矩阵的转置左乘到 $\mathbf{a}$ 上的结果：

1. $P_1(a_i, a_j)$ ，其转置数值上等于 $P_1(a_i, a_j)$ 。
2. $P_2(a_i, k)$ ，其转置数值上等于 $P_2(a_i, k)$ 。
3. $P_3(a_i, a_j, k)$ ，其转置数值上等于 $P_3(a_j, a_i, k)$ 。

发现对于前两种初等矩阵，其转置后效果不变；而第三种则交换了 i 和 j 。

因此，将指定线性变换算法改写成该算法的转置时，需要将所有的操作改写成对应转置并逆序执行。对于前两种初等矩阵对应操作，也就是交换两个数和让一个数扩大常数倍，都是不变的；而对于第三种初等矩阵对应操作，让一个数加上另一个数的若干倍 $x \leftarrow x + k \times y$ 都应变成 $y \leftarrow y + k \times x$ 。

3 转置原理与动态规划

3.1 基本思想

使用转置原理求解问题，需要先将问题描述成一个线性变换算法，并根据 A 矩阵的具体含义，得到线性变换算法 $\mathbf{b}' = A^T \mathbf{a}'$ 的实际含义。如果转置后的新问题可以被使用动态规划解决，那么原问题便可以通过将动态规划的过程转置之后进行求解。

3.2 动态规划优化

事实上，对于绝大多数可以通过转置去解决的动态规划问题，是可以不使用转置原理而直接设计状态和转移。但有些情况下直接去设计会比较困难，尤其是在需要使用 DP 优化的问题上，就会显得繁琐与不自然。在 DP 问题需要使用较复杂的算法或者数据结构优化的情况下，使用转置原理来解决此类问题，将会在算法设计方面带来一定程度上的便利。

能够进行转置的前提是动态规划问题可以被写成线性变换算法，目前相关问题主要涉及如下两类知识：

- 生成函数与多项式

- 线段树

针对前者的研究较为广泛 [3]，而对于后者的研究目前较少，所以本文将着重研究转置原理在需要线段树优化 DP 的问题中的应用。

4 线段树上部分操作的转置

由于要讨论使用线段树优化 DP 的算法的转置，那么就需要优先考虑对于线段树上部分操作的转置分别对应的是什么。在实施转置前，必须将线段树算法描述成线性变换算法的形式。在具体问题中，需要将所有输入的数据合理地分配设计到 A 或 $\mathbf{a}$ 中，使得相关线段树算法变成线性变换算法。

4.1 区间加单点查询

维护一个长度为 n 的序列 a_i ，初始全部为 0，需要支持如下两种操作：

1. $l\ r\ x$ ，让下标在区间 $[l, r]$ 的 a_i 增加 x 。
2. $q\ y$ ，查询 a_q 的值，存储到 y 中。

这个问题相当于输入 l, r, x, q 四类数，将结果输出到 y 中。

操作 1 的输入记为 l_i, r_i, x_i ($i = 1, 2 \dots n_1$)，操作 2 的输入输出记为 q_i, y_i ($i = 1, 2 \dots n_2$)。

对于每一次操作 2，可以找到一个集合 S_i ，有 $y_i = \sum_{j \in S_i} x_j$ 。这个集合 S_i 就是所有满足在第 i 个操作 2 之前执行，且 $q_i \in [l_j, r_j]$ 的操作 1 的下标 j 构成的集合。

记矩阵 $A = (a_{ij})_{n_2 \times n_1}$ ，其中 $a_{ij} = [j \in S_i]$ 。则线段树算法可以被描述成 $\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n_2} \end{bmatrix} = A \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_1} \end{bmatrix}$ ，将

x_i 看作输入，而将 l_i, r_i 和 q_i 设计到 A 中，则区间加单点查询算法就是针对矩阵 A 的线性变换算法。

4.1.1 暴力操作的转置

为了方便处理，将 A 扩张成一个 $(n + n_1 + n_2)$ 阶方阵，同样 $\mathbf{a}$ 和 $\mathbf{b}$ 变成了 $(n + n_1 + n_2) \times 1$ 的向量。其中 $\mathbf{a}$ 中的元素依次为：输入的 n_1 个 x_i ，用来存储序列 a 的 n 个 a_i ，存储答案 n_2 个 y_i 。这里约定输入向量的后 $n_2 + n$ 个数初始时均为 0。

尝试对于每一次操作构造出一个矩阵 B_i ，使得最终有 $A = B_{n_1+n_2} B_{n_1+n_2-1} \dots B_1$ 。

1. $l_i\ r_i\ x_i$ 操作：在这一次操作中，会让 $a_{l_i}, a_{l_i+1} \dots a_{r_i}$ 增加 x_i ，写出该操作对应矩阵为 $P_3(a_{l_i}, x_i, 1) P_3(a_{l_i+1}, x_i, 1) \dots P_3(a_{r_i}, x_i, 1)$ 。

2. $q_i\ y_i$ 操作：在这一次操作中，会让 y_i 赋值乘 a_{q_i} ，由于初始时钦定 $y_i = 0$ ，也就是执行操作 $y_i \leftarrow y_i + a_{q_i}$ 。对应矩阵等于 $P_3(y_i, a_{q_i}, 1)$

考察 A 的转置 A^T ，由于有 $A^T = B_1^T B_2^T \dots B_{n_1+n_2}^T$ ，所以可以考虑每一个操作 B_i 的转置 B_i^T 。即考察上面两种操作的转置：

1. $l_i\ r_i\ x_i$ 操作：原操作矩阵为 $P_3(a_{l_i}, x_i, 1) P_3(a_{l_i+1}, x_i, 1) \dots P_3(a_{r_i}, x_i, 1)$ ，那么其转置就是 $P_3(x_i, a_{r_i}, 1) P_3(x_i, a_{r_i-1}, 1) \dots P_3(x_i, a_{l_i}, 1)$ 。还原其具体含义就是，让 x_i 依次加上 $a_{r_i}, a_{r_i-1} \dots a_{l_i}$ 的值。
2. $q_i\ y_i$ 操作：原操作矩阵为 $P_3(y_i, a_{q_i}, 1)$ ，那么其转置就是 $P_3(a_{q_i}, y_i, 1)$ 。还原其具体含义就是，让 a_{q_i} 加上 y_i 的值。

通过分析，可以让新问题输入在 y_i 处输入任意值，所有 a_i 与 x_i 初始 0。那么 A^T 得到的线性变换算法实际含义如下：

有一个长度为 n 的序列，其所有元素初始为 0，支持两种操作：

1. $l_i\ r_i\ x_i$ 操作：将 $a_{l_i}, a_{l_i+1} \dots a_{r_i}$ 的和赋值到 x_i 。
2. $q_i\ y_i$ 操作：让 a_{q_i} 增加 y_i 。

容易看到，以上对应的是单点加区间和操作。

4.1.2 线段树上操作的转置

实现方式使用标记永久化，也就是对于每一个节点维护 s_{pos} 表示对应区间的所有 a_i 都会增加 s_{pos} 。

将 A 扩张成一个 $(n_1 + 2n - 1 + n_2)$ 阶方阵。对于输入的向量 $\mathbf{a}$ ，相较于上面的暴力算法，前 n_1 和后 n_2 位的含义不变，仍然为 x_i 与 y_i ，只是将中间原本的 n 个数 a_i 替换成了 $2n - 1$ 个线段树节点 s_{pos} 。其中所有 s_{pos} 和 y_i 初始为零。

- $l_i\ r_i\ x_i$ 操作：将 $[l_i, r_i]$ 划分成若干个线段树上对应区间，记这些区间对应的节点集合为 S_i ，对于所有的 $pos \in S_i$ ，让 s_{pos} 增加 x_i 。该操作对应的矩阵为 $\prod_{pos \in S_i} P_3(s_{pos}, x_i, 1)$ （这 $|S_i|$ 个矩阵的对应的修改操作两两独立，所以这些矩阵之间的相对顺序不影响矩阵乘法计算结果）。
- $q_i\ y_i$ 操作：找到线段树上包含 q_i 的所有区间，记这些区间对应的节点集合为 S'_i ，对于所有 $pos \in S'_i$ ，让 y_i 增加 s_{pos} 。该操作对应的矩阵为 $\prod_{pos \in S'_i} P_3(y_i, s_{pos}, 1)$ （同上可知这 $|S'_i|$ 个矩阵之间的相对顺序不影响矩阵乘法计算结果）。

考虑将这两个操作进行转置：

- $l_i\ r_i\ x_i$ 操作：原操作矩阵为 $\prod_{pos \in S_i} P_3(s_{pos}, x_i, 1)$ ，其转置为 $\prod_{pos \in S_i} P_3(x_i, s_{pos}, 1)$ 。还原其具体含义为，让 x_i 依次加上所有 $pos \in S_i$ 的 s_{pos} 的值。
- $q_i\ y_i$ 操作：原操作矩阵为 $\prod_{pos \in S'_i} P_3(y_i, s_{pos}, 1)$ ，其转置为 $\prod_{pos \in S'_i} P_3(s_{pos}, y_i, 1)$ 。还原其具体含义为，让所有 $pos \in S'_i$ 的 s_{pos} 增加 y_i 的值。

分析这两种操作的实际意义，让新问题输入在 y_i 处输入任意值，所有 s_{pos} 与 x_i 初始 0，这就是区间和以及单点加操作。发现这和不使用线段树优化，而得到暴力区间加单点查的转置得到的结果是一致的。

4.2 求解线段树上操作转置的两种方式

基于以上分析，可以看到对于线段树上的操作，有两种求解其转置的方式：

1. 通过分析在线段树上进行的操作直接得到其转置。
2. 先找到暴力操作的转置，然后对其使用线段树优化。

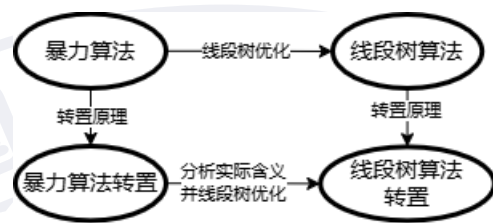


图 1: 求解线段树上操作的转置的两种途径

这两种流程分别对应了图 1 中两种从左上角到右下角的路径。

第一种方式在分析线段树算法的转置时会较为复杂，但适用于线段树上的任意操作；而使用第二种方式，在原算法较简单时是容易的，但在原算法较为复杂时，分析实际含义并进行优化相比第一种方式可能更为困难。例如在下面的部分例子中，使用第二种方法得到暴力算法的转置之后，不仅不好使用线段树优化，同时还需要进行一些额外的证明才能够保证算法的时空复杂度和正确性。所以在实际算法中使用时需要仔细比较两种方式的优劣，并选择更为合适的方式。

在下文中，分析转置算法时，将不再具体给出每一步操作的对应的矩阵，而是会直接使用前文中简记的三种符号 $a_i \leftrightarrow a_j$, $a_i \leftarrow k \times a_i$, $a_i \leftarrow a_i + k \times a_j$ 来表示三种初等矩阵。

4.3 线段树合并的转置

本节暂时不关注数值计算，只考虑将线段树合并的操作进行逆序。在实际问题中，一般会进行一次正常的线段树合并的过程，然后再逆序过来处理，所以这个过程将会和撤销非常相似，本文将这个过程称为线段树分裂。这里定义的线段树分裂和一般的线段树分裂的定义有一定的出入。线段树分裂一般指将线段树上的所有节点按照值域不交的两部分（若值域为 $[1, n]$ ，一般分裂成 $[1, k]$ 和 $(k, n]$ 两部分），而此处线段树合并的撤销分裂出的两棵线段树值域可能相交，这取决于被撤销合并的两棵线段树的值域。

考虑对两棵线段树 T_1 和 T_2 进行合并，在选出两棵树的根节点 rt_1 和 rt_2 后进行如下合并操作：

- 如果 rt_1 或 rt_2 为空，则直接返回不为空的一个（如果二者均为空则返回空结点）。
- 否则，将 rt_1 和 rt_2 的左右儿子分别合并作为 rt_1 新的左右儿子，更新 rt_1 的信息并返回 rt_1 。

这样可以合并得到新的线段树 T 。

考虑将 T 分裂成 T_1 和 T_2 的过程。会先对 T 的根 rt 进行分裂：

- 如果 rt 是通过合并中的 1 情况得到的，那么就将其还原到对应子树的儿子。
- 否则，递归将 rt 的左右儿子分别分裂到 rt_1 和 rt_2 的左右儿子上，更新 rt_1 和 rt_2 的信息。

不难发现，由于合并的时空复杂度均为 $O(n \log n)$ ，若需要维护撤销，可以在进行线段树合并时额外记录每一次合并被修改的信息，在撤销时还原对应合并修改过的所有信息即可，这样处理时空复杂度仍保持为 $O(n \log n)$ 。

4.4 线段树合并优化 MAX 卷积

4.4.1 MAX 卷积

如无特殊说明，下文中所有小写字母表示的数列将使用其对应大写字母表示其形式幂级数，例如： $F = \sum_{i=0}^n f_i x^i$, $D_i = \sum_{i=0}^n d_{i,j} x^j$ 。

定义 4.1 (形式幂级数的 MAX 卷积运算). 称 H 为形式幂级数 F 和 G 的 MAX 卷积，当且仅当对于所有 $k = 0, 1 \dots n$ ，有 $h_k = \sum_{\max(i,j)=k} f_i \times g_j$ 。记为 $H = F \otimes G$ 。

4.4.2 MAX 卷积的转置

将 MAX 卷积描述称线性变换算法的形式：认为 F 对应输入向量 a ，由 G 生成矩阵 A ， H 对应计算结果 b 。在此定义下，记 MAX 卷积 $H = F \otimes G$ 的转置为 $F = H \otimes^T G$ 。

从定义的角度考虑 MAX 卷积的转置的实际含义。MAX 卷积的操作可以被认为是枚举所有的 i, j ，令 $k = \max(i, j)$ ，执行操作 $h_k \leftarrow h_k + f_i \times g_j$ 。那么对于一对 i, j 的操作 $h_k \leftarrow h_k + f_i \times g_j$ 其转置即为 $f_i \leftarrow f_i + h_k \times g_j$ 。

将所有的操作再整合到一起，则 MAX 卷积的转置对应操作为：枚举所有的 i, j ，令 $k = \max(i, j)$ ，执行操作 $f_i \leftarrow f_i + h_k \times g_j$ 。可以得到若 $F = H \otimes^T G$ ，则有 $f_i = \sum_{j=0}^n h_{\max(i,j)} \times g_j$ ；

讨论 i 和 j 的大小关系，也可以将转移写为 $f_i = \sum_{j=0}^i h_i \times g_j + \sum_{j=i+1}^n h_j \times g_j$ 。

4.4.3 MAX 卷积的计算

考虑如下计算 $H = G \otimes F$ 的算法：从 0 到 n 枚举 i ，同时实时维护 $sf = \sum_{j=0}^{i-1} f_j$ 和 $sg = \sum_{j=0}^{i-1} g_j$ ，讨论 f_i 和 g_i 是否为 0。

- 如果 $f_i = g_i = 0$, 有 $h_i = 0$ 。
- 如果 $f_i = 0, g_i \neq 0$, 有 $h_i = sf \times g_i$, 同时 $sg \leftarrow sg + g_i$ 。
- 如果 $f_i \neq 0, g_i = 0$, 有 $h_i = sg \times f_i$, 同时 $sf \leftarrow sf + f_i$ 。
- 如果 $f_i \neq 0, g_i \neq 0$, 有 $h_i = sf \times g_i + sg \times f_i + f_i \times g_i$, 同时 $sf \leftarrow sf + f_i, sg \leftarrow sg + g_i$ 。

4.4.4 线段树合并优化 MAX 卷积原理

定理 4.1. 若 $H = F \otimes G$, 则对于所有 $k = 0, 1 \dots n$, $h_k \neq 0$ 的必要条件是 $f_k \neq 0$ 或 $g_k \neq 0$ 。

证明. 反证法. 若 $f_k = g_k = 0$, 又因为 $h_k = \sum_{\max(i,j)=k} f_i \times g_j$, 而对于所有满足 $\max(i,j) = k$ 的 i, j 而言, $i = k$ 和 $j = k$ 至少有一个成立, 那么 $f_i = 0$ 和 $g_j = 0$ 至少有一个成立. 那么就有 $f_i g_j = 0$ 恒成立, 则 $h_k = \sum_{\max(i,j)=k} f_i \times g_j = 0$, 与 $h_k \neq 0$ 矛盾。□

而在树形 DP 问题中, 若有 DP 数组 $d_{i,j}$, 且转移过程如下:

1. 将 D_i 中的 $O(1)$ 位初始化为非 0 数。
2. 依次枚举每一个儿子 v , 令 $D_i \leftarrow D_i \otimes D_v$ 。

如果记 S_i 为 D_i 初始非零位置的集合, 在每一次合并的过程中, 让 $S_i \leftarrow S_i \cup S_v$, 那么 D_i 中的非零位集合一定是 S_i 的子集。所以可以用线段树单独维护出这 $|S_i|$ 个位置的值, 其余位置节点为空表示这些位置值为 0。

那么就可以将上面的 MAX 卷积计算方法放到树上, 对 i 和 v 的两棵线段树进行合并, 假设当前要合并的节点为 pos_1 和 pos_2 。为了与前文标记保持一致, 记 $F = D_i, G = D_v$, 同样维护 sf 和 sg (含义与上面一致)。记 pos_1 节点对应区间所有 f_i 的和为 f_{pos_1} , g_{pos_2} 同理。

- 如果 pos_1 和 pos_2 均为空, 返回空。
- 如果 pos_1 不为空, pos_2 为空, 令 $sf \leftarrow sf + f_{pos_1}$, 给 pos_1 打上乘 sg 的标记, 返回 pos_1 。
- 如果 pos_1 为空, pos_2 不为空, 令 $sg \leftarrow sg + g_{pos_2}$, 给 pos_2 打上乘 sf 的标记, 返回 pos_2 。
- 如果 pos_1 和 pos_2 均不为空, 如果 pos_1 和 pos_2 已经是叶子了, 直接和上面一样计算这个位置的值即可; 否则递归到左右儿子分别处理 (优先遍历左儿子), 然后再更新当前点的值。

这一过程和普通线段树合并的时空复杂度完全一致, 即 $O(n \log n)$ 。

4.4.5 线段树合并优化 MAX 卷积转置

在上述 DP 中, 由于卷积为 $D_i \leftarrow D_i \otimes D_v$, 如果将 $dp_{i,j}$ 均看作变量则该 DP 算法将不再是线性变换算法, 所以需将其看作常数, 同时额外维护一个数组 $c_{i,j}$, 考虑更为复杂的转移:

$D_i \leftarrow D_i \otimes D_v, C_i \leftarrow C_i \otimes D_v + C_v \otimes D_i$ 。这样就可以将问题看作关于 c 的线性变换算法, 原先的 d 数组被看作常数。

单独考虑其中 $C_i \leftarrow C_i \otimes D_v$ 的转置: $C_i \leftarrow C_i \otimes^T D_v$ 。

考虑上面合并的逆序过程, 就是要将合并后的线段树中由原属于 v 的部分剥离出去。将 C_i 记作 F, D_v 记作 G , 发现 sg 由于维护的是 D_v , 而 D_v 在此处为常数, 所以含义不变, 只需要重新考虑 sf 。

假设当前访问到的节点为 pos , 记当前的 pos 是由 pos_1 与 pos_2 合并得来的。其中 pos_1 对应将要计算出的 C_i 对应的线段树节点。

- 如果 pos 为空, 直接返回。
- 如果 pos_1 不为空, pos_2 为空, 对于 pos_1 节点区间内的每一个 f_i , $f_i \leftarrow sg \times f_i + sf$ 。
- 如果 pos_1 为空, pos_2 不为空, 运算不与 f 或 sf 相关, 不在此处讨论。
- 如果 pos_1 和 pos_2 均不为空, 如果 pos_1 和 pos_2 已经是叶子了, 那么有 $sf \leftarrow g_i \times f_i, f_i \leftarrow f_i \times (sg + g_i) + sf$ 同时进行; 否则递归到左右儿子分别处理 (优先遍历右儿子), 然后再更新当前点的值。

如果先考虑 MAX 卷积的转置, 然后再进行线段树优化, 也可以得到类似的结果。但是, MAX 卷积的转置并不保证 D_i 的非零位的集合是 S_i 的子集。是可以证明只有在 S_i 中的这些位置才会有意义, 也就是使用线段树只维护这些位置的值是正确的。

5 例题

5.1 例题 1

题目来源 NOI2024 D2T2 登山¹

题目大意 给定一个以 1 为根的树, 第 i 个点的父亲为 p_i , 记一个点的深度 d_i 为其到 1 节点的边数。每一个点有对合法登山路径进行限制的参数 l_i, r_i, h_i 。

定义一条从 x 节点开始的合法登山路径对应一个登山序列 $a_1 = x, a_2, a_3 \dots a_m = 1$, 其满足对于所有 $1 \leq i < m$, a_{i+1} 为 a_i 的 k ($l_{a_i} \leq k \leq r_{a_i}$) 级祖先或 $p_{a_{i+1}} = a_i$, 且对于所有 $1 \leq i < j \leq m$ 且 $p_{a_j} \neq a_{j-1}$, 有 $d_{a_j} < d_{a_i} - h_{a_i}$ 。

记 f_x 为从 x 开始走到 1 的合法登山路径条数, 两条登山路径不同当且仅当登山序列不同, 求 $f_2, f_3 \dots f_n$ 。

解法 记 $A = [f_2, f_3 \dots f_n]^T$, 令 $\mathbf{a} = [1]$, 相当于要求向量 $\mathbf{b} = A\mathbf{a}$ 。

考虑它的转置问题: 假设现在有系数向量 $\mathbf{a}' = [v_2, v_3 \dots v_n]^T$, 求解 $\mathbf{b}' = F^T \mathbf{a}'$ 。

则问题转化为: 定义一条合法登山路径的权值为其起点 s 处的权值 v_s , 求所有合法登山路径的权值和。

¹<https://www.noi.cn/zxzy/lnzl/jszl/2024-08-21/829056.shtml>

根据登山路径的定义，发现每一次冲刺到的点必然是之前冲刺到的点的祖先，所以可以设计 DP 状态 g_x 表示所有冲刺到 x 的路径的权值和。

设计辅助状态 $c_{i,j}$ 表示在以 i 为根的子树中，向上冲刺到深度为 j 的祖先处的路径数， $t_{i,j}$ 表示在以 i 为根的子树中，可以向上冲刺到深度为 j 的祖先处的路径的权值之和。

记 S_x 为 x 的儿子集合，那么就有 $g_x = v_x + \sum_{i \in S_x} t_{i,d_x}$ ，

$$c_{x,j} = \begin{cases} [d_x - r_x \leq j \leq d_x - l_x] + \sum_{i \in S_x} c_{i,j} & , j < d_x - h_x \\ 0 & , j \geq d_x - h_x \end{cases} \quad (1)$$

$$t_{x,j} = \begin{cases} g_x + \sum_{i \in S_x} t_{i,j} & , j < d_x - h_x \\ 0 & , j \geq d_x - h_x \end{cases} \quad (2)$$

最终答案就是 g_1 。

具体在单个节点 x 处的操作如下：

1. $c_{x,j} = \sum_{i \in S_x} c_{i,j}$, $t_{x,j} = \sum_{i \in S_x} t_{i,j}$ 。
2. 令 $g_x \leftarrow v_x + t_{x,d_x}$ 。
3. 对于 $c_{x,j}$ ，对于 $j \in [d_x - r_x, d_x - l_x]$ 的范围内的加上 1； $j \geq d_x - h_x$ 的部分清空。
4. 对于 $t_{x,j}$ ，对于 $j < d_x - h_x$ 的部分加上 $c_{x,j}g_x$ ； $j \geq d_x - h_x$ 的部分清空。

使用线段树合并优化 c, t 数组的维护，可以在 $O(n \log n)$ 的时空复杂度内解决该问题。

考虑该问题的转置。 c 数组在整个过程中只作为系数出现，所以针对该数组的维护直接撤销即可。

初始时有 $d_{1,0} = 1$ 。将上面在单个节点 x 处的操作转置处理：

1. 令 $g_x \leftarrow \sum_{j < d_x - h_x} c_{x,j} t_{x,j}$ 。
2. 令 $f_x \leftarrow g_x$, $t_{x,d_x} \leftarrow g_x$ 。
3. 对于每一个儿子 i 有 $t_{i,j} \leftarrow t_{x,j}$ 。

最终 f_x 就是答案。使用线段树合并以及撤销线段树合并，可以在 $O(n \log n)$ 的时间复杂度内解决问题。

直接从实际含义的角度观察这个 DP，发现 $f = g$ ，同时 $t_{x,j}$ 就是 x 的深度为 j 的祖先 u 的 g_u 。

根据 $c_{i,j}$ 的含义“从 i 出发，休息若干次之后冲刺到其深度为 j 的祖先处的方案数”，可知 $g_x = \sum_{y \in Fa_x} g_y c_{x,d_y}$ ，其中 Fa_x 表示 x 在树上所有祖先构成的集合，这样也可以直接得到上述做法。

在这道题中，直接考虑原问题的求解可能会易于使用转置原理求解，但在考虑对于系数 c 的维护去设计优化可能会有一定的障碍，通过转置去考虑这个问题不失为一种有效的方法。

5.2 例题 2

题目大意 给定一棵树，点 i 有点权 p_i ，定义一个连通子图的权值为其包含的所有点中的点权最大值。对于每一个点 k ，求树上包含 k 的所有连通子图的权值之和。

解法 记 $a_{i,j}$ 表示包含第 i 个节点且最大点权为 j 的连通子图数量，矩阵 $A = (a_{i,j})$ 。记系数向量 $\mathbf{a} = [1, 2, \dots, n]^T$ 。原题意就是要求解 $\mathbf{b} = \mathbf{A}\mathbf{a}$ 。

考虑其转置问题，假设有一系数向量 $\mathbf{a}' = [v_1, v_2 \dots v_n]^T$ ，考虑求解 $\mathbf{b}' = \mathbf{A}^T \mathbf{a}'$ 。通过分析可以得到转置后的问题题意如下：给每一个点定义一个重量 v_i ，定义一个连通子图的重量为其所有点的重量之和。对于 $i = 1, 2 \dots n$ ，求所有权值为 i 的连通子图的重量之和。

这个问题是容易使用 DP 解决的：

将树定义为以 1 为根的有根树，定义一个点的深度为其到 1 号点路径上的点数量，定义一个连通子图的根为其中深度最小的点，不难证明这样的有且仅有一个。

记 $f_{i,j}$ 表示以 i 为根且权值为 j 的连通子图数量， $g_{i,j}$ 表示以 i 为根且权值为 j 的连通子图的重量之和。

考虑对于 i 求解 F_i 和 G_i ，假设已经对于 i 的每一个儿子 v 求出了其 F_v 和 G_v ：

- 初始时有 $F_i = x^{p_i}, G_i = v_i x^{p_i}$ ，表示在只考虑节点 i 时的 F_i 和 G_i 。
- 考虑将每一个儿子 v 逐个合并进来，此时以 i 为根的连通子图可能包含了 v 子树内的一个连通子图，而这个连通子图必然时以 v 节点为根的，因此可以得到转移： $F_i \leftarrow F_i \otimes (1 + F_v)$, $G_i \leftarrow G_i \otimes (1 + F_v) + G_v \otimes F_i$ 。

最终答案需要统计权值为所有连通子图的重量之和，也就是对 G_i 求和，在每一次合并完成之后令 $y \leftarrow y + G_i$ 。

将 i 及其儿子 v 合并的过程是 MAX 卷积，是可以使用线段树合并优化的。这样可以在 $O(n \log n)$ 的时间复杂度内解决这个问题，最终答案存储在数组 y 中。

考虑将上面的过程逆序过来，将线段树合并变成线段树分裂，同时逆序实现所有的转移。发现 F 的所有转移都和 $\mathbf{a}'$ 无关，所以将其看作系数，只需要做撤销还原即可。

转置问题中最终答案存储在 y 数组中，则对于原问题应有 $y = \sum_{i=1}^n i x^i$ 。

逆序考虑在 i 节点执行的所有操作的逆：

- $G_i \leftarrow G_i + y$ 。
- 将所有的儿子逆序分裂出来： $G_v \leftarrow G_i \otimes^T F_i$, $G_i \leftarrow G_i \otimes^T (F_v + 1)$ 。
- 分裂结束后有 $v_i \leftarrow [x^{p_i}]G_i$ 。

那么最终的答案就是 y 数组。

在线段树分裂的过程中处理 $\otimes^T$ 根据上一节的讨论时可以做到 $O(n \log n)$ ，即就能够在 $O(n \log n)$ 的时空复杂度下解决原问题。

同样，可以得到原问题的 DP g 实际含义： $g_{i,j}$ 表示“对于一个以 i 为根的连通子图，和所有包含 fa_i 但不包含 i 的连通子图（可以为空）合并在一起的连通子图的权值之和”。在 $f_{i,j}$ 仍然沿用上面的定义的情况下，可以得到完全相同的 DP 转移。但是在设计状态，得到转移，以及优化转移复杂度的过程会更加繁琐。

第二道例题相较于第一道例题，矩阵 A 更加复杂，此时直接考虑原问题就会较为复杂，也难以很自然地想到对于转移的优化，尤其是对于 MAX 卷积的转置 $\otimes^T$ 的计算与优化。此时使用转置原理去求解该问题则具有一定的优势。

6 总结

本文介绍的转置原理，并研究了线段树上部分操作的转置，结合两道较为典型的题目，展示了转置原理在部分可以被描述为线性变换算法的线段树合并（分裂）优化动态规划问题中的应用，可以大幅降低这些问题在动态规划状态设计、动态规划转移设计以及动态规划转移优化的难度，并且能够较为容易地实现与线段树合并一致的时空复杂度（一般为 $O(n \log n)$ ）。

7 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢国家集训队的教练们对我的帮助与指导。
感谢向期中老师、胡伟栋老师对我的关心与指导。
感谢家人对我的支持与鼓励。
感谢张睿宸、吴瑞祺等同学为本文验稿并提出建议。

参考文献

- [1] É.Schost A.Bostan, G.Lecerf. Tellegen's principle into practice. Proceedings of the 2003 international symposium on Symbolic and algebraic computation, pages 37–44, 2003.
- [2] Gilbert Strang. 线性代数（第 5 版）. 清华大学出版社, 2019.
- [3] 陈宇. 转置原理的简单介绍. 国家集训队 2020 论文集, 2020.

浅谈线性基在 OI 中的应用

杭州第二中学 陈昕阳

摘要

信息学竞赛中的线性基是解决一类子序列异或和问题的利器，且有着许多不同方向的扩展。本文先详细介绍了线性基最基础的构建方法与应用以及各种常见拓展，再在带删线性基这一方向提出了一些独到的方法与见解，解决了这一方向之前未能解决的一些问题。

1 导引

1.1 前言与定义

本文用 $\oplus$ 符号表示异或运算。

在 OI 中，一个由非负整数序列 S 的线性基通常指将这个序列内的所有元素视为二进制数（通常有较小的位数上界 m ，即所有元素在 $[0, 2^m)$ 之间，本文中一般默认 $m \leq 30$ 即对这些二进制数进行异或运算是 $O(1)$ 的），目标是构造另一个由 m 位二进制数序列 T ，使得：

1. T 所有子序列（同时注意空子序列也是一种子序列）元素的异或和互不相同。
2. S 所有子序列元素的异或和视为不可重集后，与 T 所有子序列元素异或和构成的集合完全相同。
3. T 中所有元素最高位严格递减。

这样做的基本想法是借由性质二通过研究 T 子序列的异或和，来研究 S 的子序列异或和。而 T 因为满足这三点限制，具有更好的性质，研究 T 子序列异或和可能比研究 S 子序列异或和更简单。值得注意的是，这样的序列 T 并不唯一，存在很多不同但在应用上可能相互等价的选法。

实际上 OI 中的线性基与线性代数有很深的联系，此处谈论的 m 位二进制数可以等同为长度为 m 的系数在 $\mathbb{F}_2$ 中的向量，而要求构造的线性基就是给定的由 m 位系数在 $\mathbb{F}_2$ 中的向量构成的序列 S ，张成的线性空间的一组基。不过这样忽略了限制三，再认为要求取出的基不允许有两个向量首个非零项值相同即可保持限制三。

而希望对一些向量张成的线性空间取一组基，方法当然是执行著名的高斯消元法，实际上这与后文介绍的线性基构建算法是完全等价的。这表明从线性代数的角度观 OI 中的线性基，其实在有相关知识的情况下会发现无非是给线性代数中的一些内容换了个名称，本质并无差别。

但考虑到大多数 OI 选手理应没有学习过线性代数相关知识，且实际上在大多数情况下引入线性代数的语言也并不能显著简化问题，只是徒增需要线性代数相关知识的门槛，本文会尽量避免使用线性代数的语言，而尽量使用虽然不甚严谨规范，但在 OI 中更广泛流传的语言来做描述。

1.2 导引

本文除本章引入与导引外，正文一共四章。其中前两章内容较为基础，也比较广为人知，有很多相关资料；后两章内容较为困难，且目前已有的介绍较少，其中最后一章内容均为原创，对多个现有的经典问题给出了复杂度更优秀的解法。

第二章会介绍线性基基本的构造方法，以及构造出线性基后的一些经典应用。

第三章则着眼于对于构建出线性基后容易解决的问题，如果每次都构造线性基时间开销太大该如何优化。即介绍了若干基于线性基性质的常见优化技巧。

第四章从更易懂的角度介绍了正交线性基，它是刻画点值序列以及执行线性基求交的强大工具。

第五章介绍了动态线性基，重点在于如何对线性基执行删除操作。介绍了本文原创的两种带删线性基，并展示了带删线性基与数据结构结合后可以发挥的强大功能。

2 构造算法与基本应用

2.1 基本性质观察

正如 1.1 所说，线性基最重要的思想是要研究 S 所有子序列异或和的性质，可以借由另一个具有更好性质，且所有子序列异或和构成的不可重集与 S 所有子序列异或和构成的不可重集完全相同的序列 T 来研究。为此，可以考虑研究对一个序列 S 做怎样的操作，其所有子序列异或和构成的可重集不发生改变，在这接下来的几节中这样的相等性质称为等价。

定理：从 S 内任意选出 $1 \leq i \neq j \leq |S|$ ，从 S 中删去 S_j 加入 $S_i \oplus S_j$ ，得到的序列 S' 与 S 等价。

证明：如果选出的子序列不包含 S_j ，显然 S' 中存在对应的完全相同的子序列；如果选出的子序列同时包含 S_i, S_j ，在 S' 中存在对应的子序列中删去 S_i, S_j 但加入了 $S_i \oplus S_j$ 的方案，异或和是相同的；如果选出的子序列包含 S_j 而不包含 S_i ， S' 中存在对应子序列删去 S_j ，

加入 $S_i \oplus S_j$ 以及 S_j ，异或和不变。这给出了 S 子序列到 S' 子序列的对应方案，反方向的对应也容易构造，因此构建了子序列之间保持异或和不变的双射，显然具有等价性。

2.2 等价序列的增量构造法

有了基本的给序列内一个元素异或上序列中另一个元素，得到序列与原序列等价的性质，先介绍一个与常用的线性基的增量构造法略有不同的构造等价序列的增量构造法：

- 增量构造 $S'[1, i]$ 与 $S[1, i]$ 等价，且满足 S' 内 0 只出现在一段后缀， S' 内非零元素最高位单调严格递减。
- 初始 $i = 0$ ，考虑将 i 增加 1 时如何从 $S'[1, i]$ 构造到 $S'[1, i + 1]$ ，初始令 $S'_{i+1} = S_{i+1}$ 。
- 维护指针 j 初始指向 1，当 $j = i + 1$ 时结束。否则考察若 S'_j 的最高位严格小于 S'_{i+1} ，将 S'_{i+1} 插入到 S'_j 前面并结束将 i 增加 1 的增量构造；接下来若 S'_j 的最高位与 S'_{i+1} 相同，将 S'_{i+1} 异或上 S'_j 并将 j 增加 1；否则说明 S'_j 的最高位严格高于 S'_{i+1} ，直接将 j 增加 1 即可。

可以发现这样确实达成了目标，略微需要注意的一点是当到达 $j = i + 1$ 时的停止状态时， S'_{i+1} 是否为 0 都有可能。

S' 非零元素最高位单调递减，0 只出现在一段后缀的性质非常有趣。从 S' 中选取子序列考察异或和，异或和的值不与选出了多少个后缀 0 有关。而因为非零部分最高位单调递减，不难发现从非零部分任取子序列，得到的异或和都不相同。证明考虑任取两个不同的子序列，考察最靠前的出现于一个子序列而未出现于另一子序列的元素即可做到，两个子序列的异或和在该元素最高位上必然不相同。

这表明了假设 S' 中有 c 个 0， S' 所有子序列的异或和构成的可重集可以将看作是 S' 中非零元素所有子序列的异或和构成的不可重集复制 2^c 份拼成一个可重集。

这一结论将在后面被反复用到。

2.3 线性基的增量构造法

2.2 构造出的等价序列 S' 和在 1.1 中提及的目标序列 T 具有相仿、但是略微不同的性质。实际上注意到只要去掉 S' 中所有后导 0 就得到了一个可行的序列 T 即构造出了一组 S 的线性基。但注意在一些应用中实际上 S' 的后导 0 数量 c 也可能被用到，我们知道它代表了 S 子序列异或和构成的多重集每个元素的重复出现次数。

仿照等价序列的增量构造法，就得到了流传最广的线性基的增量构造法，这里介绍的版本记录了 c ：

- 换一种方式描述等价序列。用值 c 记录后导 0 数量，对于 S' 中非零元素，假设其最高位是 p ，将其的值记录在 w_p ，不难发现不会产生冲突。如果 S' 中不存在非零元素最高位为 p ， $w_p = 0$ 。
- 初始化 $c = 0$ ， $w_0 \sim w_{m-1}$ 均为 0。
- 增量构造加入 S_{i+1} 时，指针 j 初始指向 $m-1$ 。如果 S_{i+1} 不包含第 j 位，直接将 j 减 1，如果减完后 $j = -1$ ，将 c 加一并退出；否则如果 $w_j = 0$ ，将 w_j 设为 S_{i+1} 并退出；对于剩下的 $w_j \neq 0$ 的情况，将 S_{i+1} 异或上 w_j ，再将 j 减 1，如果减完后 $j = -1$ 同理将 c 加 1 并退出。

可以发现这样只是换了一种方式描述了等价序列 S' ，本质上没有区别。

增量加入一项 S_{i+1} 的时间复杂度为 $O(m)$ ，因此构造序列 S 的线性基的时间复杂度为 $O(|S|m)$ ，构造线性基的额外空间复杂度显然为 $O(m)$ 。

2.4 使用线性基求最大异或和子序列

假设已经对序列 S 得到了一组线性基 $w_0 \sim w_{m-1}$ ，根据 2.2 的性质求 S 的最大异或和子序列相当于求 $w_0 \sim w_{m-1}$ 的最大异或和子序列。

方便起见，将 $w_0 \sim w_{m-1}$ 中的所有非零元素按顺序为 $a_1 \sim a_k$ 。选择一个后缀 $a[p, k]$ ，考察两种 $a_p \sim a_k$ 的不同子序列 P, Q 。根据等价序列的性质， P 与 Q 的异或和必然不相同（2.2 节）。更具体地说， P 与 Q 的异或和一定在某个不低于 a_p 最高位的位上不相同，而 $a_1 \sim a_{p-1}$ 的最高位都一定严格小于 a_p 的最高位，因此如果要将 P, Q 拓展为一组 $a_1 \sim a_k$ 的子序列，无论怎么拼上一组 $a_1 \sim a_{p-1}$ 的子序列， P 与 Q 中的异或和较小者都不可能反而变得异或和较大。

这表明要求 $a_p \sim a_k$ 的最大异或和子序列，只需要先求出 $a_{p+1} \sim a_k$ 的最大异或和子序列，再考虑要不要额外选上 a_p 这一项元素即可。

算法流程可以简化为：

- 维护最优答案 ans 初始为 0，指针 j 初始指向 $m-1$ 。
- 只要 $j \geq 0$ ，将 ans 设为 $\max(ans, ans \oplus w_j)$ ，然后将 j 减一。

最后的 ans 即为目标答案，该算法时间复杂度为 $O(m)$ 。

可以发现即使 ans 的初始值并非 0 而是某个给出的值 v ，即要求的是异或和再异或上给定值 v 后，得到权值最大的子序列的权值，上述的分析也仍然完全可行，因而算法也仍然正确。

2.5 使用线性基求异或和为某个定值的子序列个数

根据 2.2 的性质，如果 S 序列存在子序列异或和为目标值，子序列个数为 2^c ，否则为 0。

而要判断 S 序列是否有子序列异或和为目标值 x ，只需要尝试向 S 的线性基内插入 x ：如果插入失败表明向 S 序列加入 x 后，其所有子序列异或和构成的不可重集没有扩大，因此 x 本来就作为 S 内某个子序列异或和出现；如果插入成功，同理表示不可重集扩大了，因此 x 并未作为 S 内某个子序列异或和出现。

该算法时间复杂度为 $O(m)$ 。

实际上该问题可视作 2.6 的一个特例。

2.6 使用线性基求异或和小于某个定值的子序列个数

仍然利用 2.2 的性质，将问题转化为求 w 所有非零项的异或和小于某个定值 lim 的子序列个数，最后将答案乘上 2^c 。设 w 非零项依次为 $a_1 \sim a_k$ 。

仍然考察后缀 $a[p, k]$ ，考虑取出的 a 的子序列在 $a[p, k]$ 内的部分的异或和 cur ，将 cur 和 lim 的不低于 a_p 最高位的所有位做比较：

- cur 较大， $a_1 \sim a_{p-1}$ 最高位都严格小于 a_p 的最高位，即使考虑它们也无法改变 cur 在某个高位超过 lim 的事实。
- lim 较大，同理任意选择 $a_1 \sim a_{p-1}$ 的一个子序列都合法。
- cur 与 lim 在这些位上完全相同，那么可以认为目标变为数 $a_1 \sim a_{p-1}$ 有多少个子序列异或和异或上 cur 低于 a_p 最高位的部分的结果，小于 lim 低于 a_p 最高位的部分，递归到更小的子问题。

由此导出了以下算法：

- 维护当前异或和 cur 初始为 0，当前答案 ans 初始也为 0，指针 j 初始指向 $m-1$ ，重复执行以下操作直到 $j < 0$ 或退出：
 - 如果 lim 在第 j 位上为 1
 - $w_j \neq 0$ ，先将 ans 加上 $2^{\sum_{k=0}^{j-1} [w_k \neq 0]}$ 。如果 cur 在第 j 位上为 0，将 cur 异或上 w_j 。无论如何将 j 减 1。
 - $w_j = 0$ ，如果 cur 在第 j 位上为 0 将答案加上 $2^{\sum_{k=0}^{j-1} [w_k \neq 0]}$ 并退出，否则将 j 减 1。
 - 如果 lim 在第 j 位上为 0
 - $w_j \neq 0$ ，如果 cur 在第 j 位上为 1，将 cur 异或上 w_j 。无论如何将 j 减 1。

– $w_j = 0$ ，如果 cur 在第 j 位上为 1 则退出，否则将 j 减 1。

- 最后将 ans 乘上 2^c 作为答案。

这些情形的分类讨论无非是基于总可以剪枝掉选择 w 非零项一段后缀的子序列时，选出的异或和与限制在已经考虑的高位上并非完全相同的情况，只留下两者完全相同的情况继续递归考虑。

该算法的时间复杂度在合理预处理后为 $O(m)$ 。

对这个算法稍加修改，利用同样的思想即可在 $O(m)$ 时间内求出异或和第 k 小的子序列的异或和值，在此不详细展开。

2.7 利用线性基解决无向图上最大 XOR 和路径问题

2.4 ~ 2.6 的应用都局限于研究序列上子序列异或和相关问题，在这一章的最后，我们来看一个更加有趣的应用：

例题 2.1.¹

给定一张边带权的无向连通图 $G = (V, E)$ ，边权在 $[0, 2^m)$ 之间。求所有从 1 到 n 的路径（不要求简单）中，经过的所有边边权异或和最大的一条的边权异或和，注意一条边在异或和中被计算的次数是它被经过的次数。

$|V|, |E| \leq 10^5, m \leq 60$ 。

这个问题当然与无向带权图上的最短路问题非常不同，这是因为一条路径再拓展一步，即额外加上一条边之后，边权异或和可能减小也可能增大，类似 Dijkstra 的分析完全失效。

一个观察是，对于同一条路径正反走两次对异或和不会产生任何影响，虽然走完之后所在节点也不会发生变化。但其实可以任选一个走过路径上的节点，再选择一条包含这个节点的回路，在正反走这条路径中首次经过这个节点时，将回路插入到剩余的需要经过的边前面，就将异或和异或上了该回路的边权异或和。

而给定的图是无向连通图，因此任取一个回路，总能走到回路上的任意一个点再走回来，因此假设当前找到了一条 $1 \rightarrow n$ 的路径权值异或和为 val ，存在一个图上的回路边权异或和为 x ，总能找到一条权值为 $val \oplus x$ 的 $1 \rightarrow n$ 的路径。

定理：将无向图上所有回路边权异或和去重后记录为序列 S ，任取一条 $1 \rightarrow n$ 的路径边权异或和为 v ， $1 \rightarrow n$ 的最大异或和路径权值为 S 中任选子序列，子序列内元素异或和异或上 v 的最大值。

证明：设这样算出的答案为 ans' ，真实答案为 ans 。 $ans \geq ans'$ 是显然的，因为确实可以由 v 对应的路径拼上 S 内某个子序列对应的回路找到一条权值为 ans' 的 $1 \rightarrow n$ 路径。 $ans \leq ans'$ 则是考虑，先走最优路径再沿着给出的 $1 \rightarrow n$ 路径返回（即倒着走，无向图因此

¹来源：[WC2011] 最大 XOR 和路径 <https://www.luogu.com.cn/problem/P4151>

合法)，再沿着给出的 $1 \rightarrow n$ 路径走到 n ，前两部分构成一条回路，因此权值为 ans 的方案被考虑到了。

于是问题转化为求出 S 序列的任意线性基即可，但按照定义求解需要枚举无向图上的所有回路，这有无穷多条，当然不可行。但注意到只需要求出 S 序列的线性基，实际上可以只考察很少的回路：

定理：考虑任意生成树，只考虑所有非树边 (u, v) 加上 u 到 v 简单路径构成的回路，边权异或和构成的线性基与原图所有回路构成的线性基等价。

证明：显然这样求出的线性基 $basis'$ 不可能比原图所有回路边权异或和构成的线性基 $basis$ 大，于是只需要证不存在某个原图上的回路，其异或和不能写成 $basis'$ 内一些元素的异或和（从而将其插入 $basis'$ 会扩大 $basis'$ ），这只需要真的给出一组写成 $basis'$ 内一些元素异或和的方案。考虑任取根 r ，记生成树上 r 到 u 简单路径边权异或和为 d_u ，任意回路边权异或和也可以表示为对其包含的所有边（经过多次计入多次） (u_i, v_i, w_i) 将 $d_{u_i} \oplus d_{v_i} \oplus w_i$ 全部异或起来后得到的结果。对于树边，结果为 0，对于非树边，结果这条非树边所对应的回路（被考虑过的）的边权异或和。

于是只需要先 $O(|V| + |E|)$ 求出任意生成树并找到任意一条 $1 \rightarrow n$ 的路径求出边权异或和。然后将 $O(|E|)$ 条非树边对应的回路的边权异或和插入线性基，再最后做一次 2.4 的查询即可，时间复杂度为 $O(|V| + |E|m + m)$ 。

2.8 本章小结

本章先指出了线性基原理中最重要的基本性质，利用基本性质立即推导出了等价序列的增量构造法，并指出最常用的线性基的增量构造法只是给等价序列的增量构造法换了一种写法。

接下来 2.4 ~ 2.6 展示了线性基最基本的三个应用，实际上这三个应用基本的想法都是一致的：利用线性基内元素最高位严格递减的特性，在某个位处考虑选出的方案在不低于这个位的部分的权值，而这个权值不受线性基内最高位更低的元素的影响，由此可以直接减枝掉很多局面。利用这一基本思想还可以解决更多更复杂的问题。

2.7 是非常经典的图上最大 XOR 和路径问题，本文给出做法并详细论证了做法的正确性。

3 常见优化技巧

3.1 线性基合并

考虑这样一个问题：假设现在已知序列 S 的一组线性基是 S' ，序列 T 的一组线性基是 T' 。现在要将序列 S 和序列 T 拼接在一起，能否在只知道 S', T' 的情况下求出 $S + T$ （表示

序列拼接) 的线性基?

从线性基实际上是等价序列去掉所有后导 0 的这一点来看,做法是显然的:直接将 S', T' 作为序列拼接起来,求这个序列的线性基得到的就是 $S + T$ 的线性基。这是因为 S' 拼上若干个 0 后与 S 等价, T' 拼上若干个 0 后与 T 等价,那么显然 $S' + T'$ 再拼上若干个 0 与 $S + T$ 等价,于是直接对 $S' + T'$ 构造等价序列即可。

如果额外记录了 c 值,合并起来后的 c 值也是容易计算的。

一次线性基合并需要将 $O(m)$ 个元素插入线性基,因此时间复杂度为 $O(m^2)$ 。

3.2 结合数据结构

针对序列 S 构造的线性基(在不加下文所介绍的其他的优化前),只能处理类似询问从整个序列中任取一个子序列,子序列权值异或和最大是多少之类的问题,并且还不支持修改。如果要面对区间询问或带修,结合数据结构是一种可行的思路。

例题 3.1.²

维护一个长度为 n 值域在 $[0, 2^m)$ 之间的非负整数序列 a 。支持 q 次操作分为两种:

- 修改操作: 将某个 a_x 修改为 y , 保证 $y \in [0, 2^m)$ 。
- 查询操作: 给出区间 $[l, r]$, 查询从 a 的子区间 $a[l, r]$ 中任取一个子序列能得到的异或和最大值。

$n, q \leq 5 \times 10^4, m \leq 30$ 。

考虑对原序列建立线段树,在对应区间为 $[l, r]$ 的节点维护 $a[l, r]$ 构成的任意一组线性基。注意到线性基可以 $O(m^2)$ 合并,于是可以简单的把一个序列的线性基看作是一种满足结合律的信息,用 $O(n + q \log n)$ 次信息合并来保证每次询问时能顺利求出 $a[l, r]$ 的任意一组线性基,进而回答询问。

该算法的时间复杂度为 $O((n + q \log n)m^2)$,实际上可以分析到略好的上界但在此不赘述。

可以发现这个算法的时间复杂度并不优秀,本章后面几节会介绍在与这个问题相似的一些更为简单的问题上该如何优化复杂度,在第五章会真正介绍如何优化这个问题的时间复杂度降低至 $O(nm + qm(\log n + m))$ 。

3.3 维护等价序列优化复杂度

如果要维护序列 S 的线性基但这难以做到,可以考虑维护与 S 等价的序列构成的线性基。这在静态的时候当然没有帮助,但如果要针对序列 S 进行一些修改,有时会很有效果。

²来源: 经典问题, 出处不明

例题 3.2.³

维护一个长度为 n 值域在 $[0, 2^m)$ 之间的非负整数序列 a 。支持 q 次操作分为两种:

- 修改操作: 给出区间 $[l, r]$ 和值 $v \in [0, 2^m)$, 将 $a_l \sim a_r$ 全部异或上 v 。
- 查询操作: 给出区间 $[l, r]$, 查询从 a 的子区间 $a[l, r]$ 中任取一个子序列能得到的异或和最大值。

$n, q \leq 5 \times 10^4, m \leq 30$ 。

设 $b_i = a_i \oplus a_{i-1}$, 可以发现 $a[l, r]$ 等价于 $a_l + b[l + 1, r]$ 。而对 $a_l \sim a_r$ 全部异或上 v 的操作对 b 的影响只是将 b_l, b_{r+1} 异或上了 v , 可以视为是对 b 进行两次单点修改。于是用线段树维护 b 序列做 3.2 的问题每次查询出 $b[l + 1, r]$ 构成的任意线性基,再将 a_l 插入该线性基即可解决此问题,需要略微注意的一点是还需要支持对 a 进行区间异或,单点查询,但这也不难做到,时间复杂度仍为 $O((n + q \log n)m^2)$ 。

注意不要将此题视作孤例,如果希望以“维护等价序列”作为做法命题,很容易就能得到以下题目:

例题 3.3.⁴

维护一个长度为 n 值域在 $[0, 2^m)$ 之间的非负整数序列 a , 给定一个常数 k 。支持 q 次操作分为两种:

- 修改操作: 给出区间 $[l, r]$ 和值 $v \in [0, 2^m)$, 对于 $\forall l \leq i \leq r$ 且满足 $i \bmod k = l \bmod k$, 将 a_i 异或上 v 。
- 查询操作: 给出区间 $[l, r]$, 查询从 a 的子区间 $a[l, r]$ 中任取一个子序列能得到的异或和最大值。

$n, q \leq 5 \times 10^4, k, m \leq 30$ 。

做法大同小异,维护 $b_i = a_i \oplus a_{i-k}$ 即可,时间复杂度为 $O((n + q \log n)m^2 + qkm)$ 。实际上如果是给 $[l, r]$ 内编号模 k 在某个每次修改给出的集合 S 中的下标做异或 v 的操作,时间复杂度也可以做到 $O((n + q \log n + qk)m^2)$ 。

³来源: [Ynoi2013] 无力回天 NOI2017 <https://www.luogu.com.cn/problem/P5607>

⁴来源: 原创题

3.4 前缀线性基

前缀线性基的想法可以从两个动机看出。

- 最常用的构造序列 S 的线性基的方法是增量构造法，每次考察 S 的首个尚未被考虑过的元素，考虑加入其对线性基的影响，而我们知道要么没有影响，要么只是将某一项值由 0 填成某个非零值。所以如果记录了每个非零值是在哪一次插入时被填上的，实际上自然就可以得到 S 所有前缀的线性基信息了： $S[1, p]$ 的线性基信息就等于 S 整体的线性基信息，只保留被填上时间不晚于 p 的非零值。
- 实际上对 S 任意重排后，建出的线性基都彼此等价。因此可以给 S 内的每个值额外分配一个优先级，钦定按照优先级从高到低的顺序增量构造，将值插入线性基。

现在考虑这样一个问题：需要动态维护 (t_i, a_i) 二元组集合 S ，你需要在每次向 S 内额外插入一个二元组后，求出 $f(S)$ 表示把所有二元组按照优先级 t_i 从大到小排序后，依次将 a_i 插入到线性基，线性基的形态。保证 t_i 互不相同。

我们先给出以下在 2.2 中描述的算法基础上稍加修改后的算法，再说明其正确性：

- 在维护 w_i 的基础上记录 p_i 表示占领第 i 位的二元组的优先级，初始均为 $-\infty$ 。
- 向 S 内插入 (t_i, a_i) 时，指针 j 初始指向 $m - 1$ 。如果 a_i 不包含第 j 位，直接将 j 减 1，如果减完后 $j = -1$ ，将 c 加一并退出。否则如果 $p_j < t_i$ ，交换 w_j, a_i ，再交换 p_j, t_i 。无论如何，将 a_i 异或上 w_j ，再将 j 减 1，如果减完后 $j = -1$ 将 c 加 1 并退出。

正确性证明：交换相当于保证了一定优先级较高的二元组占领第 i 位，将优先级较低的二元组的 a 异或上优先级较高的在“按照优先级降序插入”的题设下显然也确保了等价性。该算法的时间复杂度仍为 $O(m)$ 。

3.5 前缀线性基的多种应用

前缀线性基应用广泛，在此节暂且用以下三个例题说明，后文还有和其他技巧结合的例题：

例题 3.4.⁵

给定长度为 n 值域为 $[0, 2^m)$ 的非负整数序列 a 。 q 次询问区间 $[l, r]$ 查询从 a 的子区间 $a[l, r]$ 中任取一个子序列能得到的异或和最大值。

$n, q \leq 5 \times 10^5, m \leq 30$ 。

⁵来源：CF1100F Ivan and Burgers <https://codeforces.com/problemset/problem/1100/F>

这是 3.2 例题无修改的弱化版，考虑要回答询问 $[l, r]$ ，只需要将 $\forall 1 \leq i \leq r, (i, a_i)$ 插入集合 S 然后保留只 $f(S)$ 中 $p_i \geq l$ 的部分就得到了 $a[l, r]$ 的线性基信息。而对 (i, a_i) 二元组序列的每个前缀求其 f 显然就是上面描述的问题，因此得到了 $O((n + q)m)$ 的算法。

例题 3.5.⁶

给定 n 个点每个点带 $[0, 2^m)$ 内非负整数点权的树 T 。 q 次询问 u, v 查询 u 到 v 简单路径上所有点点权构成的序列中任取一个子序列能得到的异或和最大值。

$n \leq 5 \times 10^5, q \leq 10^5, m \leq 30$ 。

这是上一个例题上树的情形。任意定根 r ，先考虑 u, v 有祖先后代关系时该如何获取 u 到 v 简单路径上点权构成序列的线性基信息。不妨设 u 是 v 的祖先，可以发现将 v 祖先中所有点 w 的 (dep_w, a_w) 插入集合 S 然后只保留 $f(S)$ 中 $p_i \geq dep_u$ 的部分就得到了目标信息，其中 dep_p 表示节点 p 到根简单路径边数。注意到动态维护 $f(S)$ 的是一个增量算法，且该算法的时间复杂度不依赖于均摊，因此从一个局面增量到不同局面的时间复杂度仍然正确（也就是可以把一个点的信息复制给它的所有子节点，再插入它子节点对应的那个二元组，也仍然完全可行）。因此该子问题仍然可以 $O((n + q)m)$ 解决。

对于不存在祖先后代关系的情形，只要求一次 LCA 将两条有祖先后代关系的链的线性基信息 $O(m^2)$ 合并即可。

因此时间复杂度为 $O(nm + qm^2)$ ，LCA 可能有额外时间开销但这里不关心。

例题 3.6.⁷

维护元素在 $[0, 2^m)$ 内的非负整数集合 S ，支持插入、删除元素，每次操作后求选出一个子集异或和的最大值。

操作次数 $q \leq 5 \times 10^5, m \leq 30$ 。

注意到本题允许离线，因此可以求出每次插入 a_i 的操作中， a_i 在第 t_i 次操作被删除，不存在则为 $+\infty$ 。那么进行前 q' 次操作后的线性基信息可以看作是将前 q' 次操作中的所有插入操作看作二元组 (t_j, a_j) 插入集合 S ，询问 $f(S)$ 中 $p_i > q'$ 的部分。

因此该问题也在 $O(qm)$ 的时间内得以解决。

本节的第三个例题表明实际上使用前缀线性基的技巧就能一定程度上实现动态线性基，但实际上这种实现方法是非常“脆弱”的。首先这个算法是离线的，难以优化到在线。另外为了实现这种虚假的在线，这一算法还已经“消耗”掉了线性基内可以控制优先级的自由元，这导致就算不要求强制在线，这个算法也不可能做到对一个序列单点修改，支持查询其任意前缀的线性基信息。在第五章会介绍真正的动态线性基，然而这一算法就算不考虑其复杂性，也仍然有其自身的缺陷。

⁶来源：SCOI2016 幸运数字 <https://www.luogu.com.cn/problem/P3292>

⁷来源：经典问题，出处不明

3.6 线性基的进一步消元

在建出线性基 $w_0 \sim w_{m-1}$ 之后，其实还可以进行进一步消元以保证若 $w_i > 0$ ，所有 $\forall i < j \leq m-1$ 的 w_j 不包含第 i 位。

做法是简单的：升序枚举 $i \in [0, m)$ ，若 $w_i > 0$ ，再升序枚举 $j \in (i, m)$ ，若 w_j 含有第 i 位，将 w_j 异或上 w_i 。

该做法的正确性可以归纳证明：假设在前 p 轮结束后，保证了 $\forall 0 \leq i \leq p$ 的位 i 满足这一性质。那么若 $w_{p+1} > 0$ ，第 $p+1$ 轮无疑保证了第 $p+1$ 位也满足这一性质，且没有破坏 $[0, p]$ 这些位的性质。

做这样的进一步消元有时并不必要，但有时很有帮助，可以通过以下例题看出：

例题 3.7.⁸

维护一个长度为 n 值域在 $[0, 2^m)$ 内的非负整数序列 a 以及一个元素同样是 $[0, 2^m)$ 内非负整数的集合 S_0 。初始 a 给定 S_0 为空集，支持 q 次操作分为三种：

- 对 a 的修改操作：给出 x, v ，将 a_x 异或上 v ， v 仍然 $\in [0, 2^m)$ 。
- 对 S_0 的修改操作：给出 v ，将 v 插入 S_0 。
- 查询操作：定义 $g(x, S) = \max_{T \subseteq S} (x \oplus \text{xor}(T))$ ，其中 $\text{xor}(T)$ 表示 T 集合内所有数的异或和。给出 $[l, r]$ 求 $\sum_{i=l}^r f(\oplus_{j=l}^i a_j, S_0)$ ，答案对 2^{64} 取模。

$$n \leq 5 \times 10^5, q \leq 10^5, m \leq 64。$$

先假设 S_0 是某个一开始就给定的集合，考虑如何回答查询操作。首先对 S_0 建线性基并执行上述的进一步消元，然后我们考虑如果询问的是 $g'(x, S) = \min_{T \subseteq S} (x \oplus \text{xor}(T))$ 该如何处理：实际上由于对于某个 $w_i > 0$ 的 i ，由于线性基内除了 w_i 之外不存在其他数包含位 i ，那么只要 x 包含位 i ，在最小异或值方案中就必须选 w_i ，否则就一定不选，不然一定不优。由此便可看出最小异或值方案是唯一的使得所有 $w_i > 0$ 的位 i 一定在异或和不被包含的选子集的方案。注意到每次用到的 x 实际上是若干数 $x_1 \sim x_k$ 的异或和，如果对每个 x_i 都求出 $g'(x_i, S)$ 并将所有 $g'(x_i, S)$ 异或起来，其对应的选取子集的方式就是这组方案。即我们证明了 $g'(x, S) = \oplus_{i=1}^k g'(x_i, S)$ 。

但实际上需要研究的是 $g(x, S)$ ，不过实际上可以注意到 $g(x, S) \oplus g'(x, S) = \oplus_{i=0}^{m-1} w_i$ 。这是因为最大异或值方案反过来是使得所有 $w_i > 0$ 的位 i 一定在异或和中被包含的取子集的方案，所以对于同一个 w_j ，其一定恰在最大异或和方案与最小异或和方案中被取一次。

于是询问可以这样回答：维护 $b_i = g'(a_i, S_0)$ ， $\sum_{i=l}^r (\oplus_{j=l}^i b_j) \oplus bas$ 即为答案，其中 bas 表示 $\oplus_{i=0}^{m-1} w_i$ 。

⁸来源：2023-2024 集训队互测 Xor Master <https://qoj.ac/contest/1417/problem/7765>

回到原问题，对 a 做单点修改操作相当于单点修改 b ，花费 $O(m)$ 时间即可求出修改后的结果。对 S_0 做修改操作如果没有扩大线性基，所有 b 均不变，否则一个暴力的想法是直接重新 $O(nm)$ 求出 $b_1 \sim b_n$ ，这种事件只会发生 m 次。问题可以转化为维护一个长度为 n 的序列，有 m 次整体重建， q 次单点修改以及查询一个区间所有前缀的异或和再异或上给定值 C 。该部分做法与本文主题无关但可以在 $O(nm + q \log^2 n)$ 内解决。

但这样只能得到 $O(nm^2 + q \log^2 n)$ 的解法，实际因为 $O(nm^2)$ 太大无法通过。

注意到瓶颈在于 $O(m)$ 次在 S_0 对应线性基扩大时直接重新计算了所有 $b_1 \sim b_n$ ，实际上执行线性基的进一步消元之后，求 $g'(x, S_0)$ 不必再按照从高位到低位的顺序将结果向额外异或上线性基内元素的方案取 $\min$ ，而是任意顺序都可行。同时如果线性基扩大导致线性基内出现了最高位为第 p 位的元素，注意到实际上知道线性基扩大前的 b_j ，而 b_j 在线性基内其他 $w_i > 0$ 的 i 位上一定均为 0，于是只需要检查 b_j 是否包含第 p 位即可。这样重新计算 $b_1 \sim b_n$ 的时间复杂度降为 $O(n)$ 。线性基扩大后再次执行进一步消元也只需要试着消掉 w_p 的更低位，并用 w_p 去消掉 $\forall p < k < m$ 的 w_k 的第 p 位，于是进一步消元也是 $O(m)$ 的。

于是该做法的时间复杂度被优化到了 $O(nm + q \log^2 n + m^2)$ 。

实际上 $g'(x, S), g(x, S)$ 之间的结论不用进一步消元的方法也能证明，但做最后的时间复杂度的优化，进一步消元是必不可少的。

3.7 前缀线性基结合数据结构

让我们考虑 3.2 的另一个相仿的问题。

例题 3.8.⁹

维护 n 个序列 $S_1 \sim S_n$ ，初始均为空序列。支持 q 次操作分为两种：

- 插入操作：向 S_x 的末尾插入 y ，保证 $y \in [0, 2^m)$ 。
- 查询操作：给出区间 $[l, r]$ ，查询将 $S_l \sim S_r$ 这些序列全部按顺序拼接起来后，从中任取一个子序列能得到的异或和最大值。

$$n, q \leq 2 \times 10^5, m \leq 30。$$

直接沿用 3.2 的做法当然并没有什么问题，但实际上利用前缀线性基的性质，可以做得更好：

建线段树，对每个线段树上的节点，如果它是一个左儿子，维护对应区间的后缀线性基；如果它是一个右儿子，维护对应区间的前缀线性基；对根什么都不维护。

查询时，考虑何时查询区间首次被 mid 切开，设此时线段树节点管辖 $[l, r]$ ，查询区间是 $[L, R]$ ，需要的线性基信息可以由 $[L, mid], [mid + 1, R]$ 做一次 $O(m^2)$ 的合并得到，而这两段

⁹来源：P 哥的桶 <https://www.luogu.com.cn/problem/P4839>

区间分别是左儿子对应区间的后缀和右儿子对应区间的前缀，信息都可以通过维护的前/后缀线性基 $O(m)$ 提取出来。找查询区间首次被 mid 切开的位置其实是求一次 LCA，理论上不是瓶颈。

修改时，对所有管辖被修改位置的线段树上节点，向这个节点的前/后缀线性基做一次插入即可。

这个算法相比 3.2 的做法，单次修改复杂度降为 $O(m \log n)$ ，单次查询复杂度降为 $O(m^2)$ 。

该例题又一次证明了前缀线性基的强大功能，但要使用前缀线性基，得先拆出“前缀询问”的形式。

但如果修改并非向某个序列末尾插入元素，而是修改某个序列中的某个元素（即 3.2 中的问题）。如果还想使用类似的方法，会需要解决 3.5 中提到的“对一个序列单点修改，支持查询其任意前缀的线性基信息”这样的问题，而这一问题在目前的讨论中，还暂时没有较好的方法。

3.8 本章小结

本章主要聚焦于对于需要用线性基解决的题目，如何快速求出需要的线性基信息。3.1 ~ 3.6 介绍了线性基合并、使用数据结构维护信息、转而维护更易维护的等价序列、前缀线性基、对线性基进行进一步消元这些常用技巧。这些常用技巧有时需要配合使用，这正是 3.7 的内容。

3.2 使用数据结构维护信息的方法基于 3.1 的线性基合并。这一方法能力非常强，但线性基合并单次 $O(m^2)$ 的代价太高往往难以接受。3.5 和 3.7 描述了如何使用 3.4 介绍的前缀线性基的技巧尽量少用或不用线性基合并来降低复杂度，这一优化方式非常常见。

3.3 描述的维护等价序列的方法是一种转化来简化问题的手法，在转化完之后仍然需要使用剩下几节的方法维护。

3.6 对线性基进行进一步消元的技巧在一些情况下也可以优化时间复杂度，但相对不太常用。

本章最后遗留下来的问题是，如何延续 3.7 中的思路优化 3.2 中的问题，这将在第五章得到解答。

4 正交线性基

4.1 准备工作

事实上正交线性基与线性代数的联系就更为紧密了，但避免从线性代数的角度介绍仍然是可行的，为此需要先介绍一些概念，但它们在 OI 中仍然是为人熟知的。

本章会区分原序列 S （不定长）和一些需要用到的，定长为 2^m 描述子序列异或和分布信息的序列，称后者为点值序列，虽然有时用到的点值序列可能找不到其对应的原序列是什么。

需要用到下面四个定义：

- 序列 S 对应的点值序列：对于某个序列 S ，定义其对应的点值序列是长度为 2^m ，编号在 $[0, 2^m)$ 中的序列 $v(S)$ ，其中 $v(S)_i$ 表示 S 子序列中异或和为 i 的子序列数量。
- 点值序列的异或卷积：对于两个点值序列 a, b ，定义它们的异或卷积 $a \times b$ 结果也是一个点值序列 c ，满足 $c_k = \sum_{0 \leq i, j < 2^m, i \oplus j = k} a_i b_j$ 。
- 点乘、正交：对于两个 m 位二进制数 i, j ，定义 $i \odot j = \text{popcnt}(i \& j) \bmod 2$ ，如果 $i \odot j = 0$ 称 i 和 j 正交。
- FWT 变换：对于一个点值序列 a ，定义对其进行 FWT 变换后的结果 $\hat{a}$ 是一个点值序列，满足 $\hat{a}_i = \sum_{j=0}^{2^m-1} (-1)^{i \odot j} a_j$ 。

可以发现按照异或卷积的定义，事实上对任意原序列 S ，有 $v(S) = \prod_{i=1}^{|S|} v(S_i)$ ，这里连乘记号指异或卷积。以下针对点值序列整体的连乘记号均指异或卷积，如果取出点值序列具体某一项进行连乘，当然就是正常乘法。

容易验证点乘与异或运算满足分配律即 $i \odot (j \oplus k) = (i \odot j) \oplus (i \odot k)$ 。

对于 FWT 变换，需要知道对 $\hat{a}$ 再进行一次 FWT 得到的结果是给 a 每一项系数乘上 2^m 。以及如果 $c = a \times b$ ，那么 $\hat{c}$ 等于 $\hat{a}$ 和 $\hat{b}$ 逐点相乘，下面记 p, q 两个点值序列逐点相乘的结果为 $p \cdot q$ 。这两点性质的证明在此略去。

4.2 点值序列 FWT 变换后的结果

现在我们来研究对于序列 S ，其点值序列 $v(S)$ 进行 FWT 后得到的 $\hat{v}(S)$ 的第 k 项的值是什么。

那么由于 $v(S) = \prod_{i=1}^{|S|} v(\{a_i\})$ ，根据 FWT 的性质 $\hat{v}(S)_k = \prod_{i=1}^{|S|} \hat{v}(\{a_i\})_k$ ，而 $\hat{v}(\{a_i\})_k = (-1)^{0 \odot k} + (-1)^{a_i \odot k} = 1 + (-1)^{a_i \odot k}$ 。因此当且仅当 $\forall 1 \leq i \leq |S|, a_i \odot k = 0$ ，此时这个连乘积项非零，值为 $2^{|S|}$ ，剩下的情况 $\hat{v}(S)_k$ 都必然为 0。

也就是说， $\hat{v}(S)_k$ 在 k 与 S 内所有元素正交时为 $2^{|S|}$ ，否则为零。

这为研究点值序列提供了另一种角度：如果能找出所有的 k 使得 k 与 S 内所有元素正交， S 的点值序列 $v(S)$ 就是将所有这些 k 的位置值设为 $2^{|S|}$ ，剩余位置设为 0 得到的点值序列进行一次 FWT 后得到的点值序列，每项除以 2^m 。

而要研究 k 是否与 S 内所有元素正交，还有以下非常好用的结论： $k \odot x = 0$ 且 $k \odot y = 0$ 充要于 $k \odot x = 0$ 且 $k \odot (x \oplus y) = 0$ ，这用点乘对异或的分配性即可证明。因此研究 k 是否与

S 内所有元素正交，等价于研究 k 是否与 S 的任意等价序列正交，等价于 k 研究 k 是否与 S 的任意线性基内所有元素正交。

新角度的好处在于，一些情况下与所有 S 内元素正交的 k 非常少，从而这样构造出的点值序列非常稀疏，而稀疏性是非常有用的。最为简单的例子就是通过 S 构造出的线性基为满线性基的情况，此时如果对线性基 3.6 提到的进行进一步消元将得到 $w_i = 2^i$ 的线性基，此时 k 与所有 2^i 正交，不难发现 k 只能为 0。

可以通过以下例子来看出新角度非常有用：

例题 4.1.¹⁰

给定长为 n 值域为 $[0, 2^m)$ 的非负整数序列 a ，对 $\forall 0 \leq i \leq m$ 求其有多少个子序列异或和的 popcnt 为 i 。

$n \leq 2 \times 10^5, m \leq 53$ 。

当然只需要在 a 的任意一组线性基上考虑原问题，最后将答案每项乘以 2^c 即可。记 a 线性基内非零元素有 s 个，对于 $s \leq \frac{m}{2}$ 时可以直接枚举线性基的所有子序列共 2^s 种，接下来考虑 s 较大的情况。

那么要研究的无非是线性基对应的点值序列在 popcnt 上的分布，使用新角度把点值序列视作与线性基内所有元素正交的 k 位置上为 2^s ，剩余位置为 0 的点值序列 FWT 后除以 2^m 的结果。可以发现其实只需要知道所有 occ_p 表示与线性基内所有元素正交的 k 中， popcnt 为 p 的恰好有 occ_p 个，实际上就已经可以推导出答案。

这是因为最后要求的是这些 k 构造出的点值序列 FWT 后得到的新点值序列在 popcnt 上的分布，那么对于一个 $\text{popcnt}(k) = p$ 的 k ，考虑其对于所有其为 1 的 p 个位上有 a 个位置也为 1，其为 0 的 $m - p$ 个位上有 b 个位置为 1 的所有 k' ， $(-1)^{k \odot k'}$ 的贡献之和，可以发现这样的 k' 有 $\binom{p}{a} \binom{m-p}{b}$ 种，且 $(-1)^{k \odot k'}$ 固定为 $(-1)^a$ ，并且这些 k' 每个的 popcnt 均为 $a + b$ ，

所以可以看作是它为 ans_{a+b} 这一项提供了 $\frac{\binom{p}{a} \binom{m-p}{b} (-1)^a}{2^m}$ 的贡献。因此在求出所有 occ_p 之后，可以 $O(m^3)$ 解决原问题。

直觉上来说，当 s 较大，与线性基内所有 s 个元素同时正交的 k 会比较少，因此暴力枚举所有这些 k 的时间复杂度应该还可以接受，但问题在于，如何快速找出所有这些 k ？这就需要使用正交线性基来解决了。

4.3 正交线性基的构造

延续 4.2 的内容，先对 a 已有的这组线性基进行进一步消元，接下来我们声称：对于所有 $w_i = 0$ 的位 i 任意选取 0 或 1，这样一共 2^{m-s} 种方案，每种方案恰好对应一种将所有 $w_i > 0$ 的位填上 0 或 1 的方案，得到一个数 k 与线性基内所有元素全部正交。

¹⁰来源：CF1336E2 Chiori and Doll Picking <https://codeforces.com/problemset/problem/1336/E2>

假设 k 的所有 $w_j = 0$ 的位已经全部选好。考虑某个 $w_i > 0$ 的位，因为 k 要与 w_i 正交，且 w_i 满足其不包含除自己以外的任何 $w_j > 0$ 的位 j ，所以要判断 k 与 w_i 是否正交，现在唯一未知的项就是 k 在位 i 上的取值，取值方案显然只有一种：如果 k 在 $w_j = 0$ 的位上与 w_i 有奇数个位置重合，那么 k 必须包含位 i ，否则必须不包含位 i 。

这也可以看作是，对每个 $w_j = 0$ 的位 j ，如果 k 选上了位 j ，且 $j < i < m$ 的 w_i 也包含位 j ，每多一个这样的位 j ， k 是否包含位 i 的包含性会发生一次翻转。

既然如此，我们可以写下一个编号为 $0 \sim m-1$ 值域为 $[0, 2^m)$ 的序列 w' ， $w'_i \neq 0$ 当且仅当 $w_i = 0$ ，且 $w_i = 0$ 时， $w'_i = 2^i + \sum_{j=i+1}^{m-1} [w_j \& 2^i = 2^i] 2^j$ 。如果希望 k 包含某个子集的 $w_i = 0$ 的位 i ，最终对应的与线性基内所有元素正交的 k 就是这个子集的位 i 对应的 w'_i 值的异或和。

这样的 w' 被称为 a 对应线性基的正交线性基，它的特点是其所所有子序列异或和构成的集合，恰好等于与 a 线性基内所有元素正交的 k 构成的集合。

如果已知 a 的一组线性基，构造其正交线性基的复杂度当然不会超过 $O(m^2)$ ，更精细的实现可以做到 $O(sm)$ ，其中 s 还是表示 a 的这组线性基的非零元素个数。

需要格外注意的一点是，构建正交线性基前必须对 a 的线性基进行进一步消元，否则求出的结果可能不正确。

于是 4.2 中的问题也得到了完美解答：只需构造出 a 对应线性基的正交线性基，枚举正交线性基的所有子序列并求出异或和，即可快速找出所有的 k ，这样做的时间复杂度为 $O(2^{m-s})$ ，平衡两部分做法可以在 $O(nm + 2^{\frac{m}{2}} + m^3)$ 时间复杂度内解决原问题。

4.4 线性基求交

在一些题目中，需要快速判断多个序列是否均有子序列异或和为某个给定值 x ，此时需要所谓“线性基求交”，实际上使用正交线性基相关技巧可以很好地解决这一类问题。

形式化地：给定序列 $S_1, S_2, \dots, S_k$ ，判断值 x 是否同时作为 $S_1 \sim S_k$ 的某个子序列的异或和出现。

这相当于判断所有 $v(S_1) \cdot v(S_2) \cdot v(S_3) \dots v(S_k)$ ， x 项系数是否为零。但 $v(S_i)$ 全部逐点相乘对应的含义我们并不熟悉，我们熟悉的是将所有 $v(S_i)$ 做异或卷积得到的 $\prod_{i=1}^k v(S_i)$ ，这无非是将所有 S_i 拼成一个大序列后，大序列的点值序列。

根据 FWT 的性质，任取点值序列 $a_1 \sim a_k$ ， $\forall j \in [0, 2^m)$ ，有 $(\text{FWT}(\prod_{i=1}^k a_i))_j = \prod_{i=1}^k (\hat{a}_i)_j$ 。考

虑代入 $a_i = \hat{v}(S_i)$ ，那么 $(\text{FWT}(\prod_{i=1}^k \hat{v}(S_i)))_j = \prod_{i=1}^k 2^m v(S_i)_j$ ，这是因为对 $v(S_i)$ 做两次 FWT 后每项变为原来的 2^m 倍。

注意到 $\hat{v}_i$ 这个点值序列实际上可以找到其对应的原序列：就是 S_i 的任意一组正交线性基。

这就凑出了我们想要的 $v_1 \cdot v_2 \cdot v_3 \dots v_k$ 的形式，这表明它 $v_1 \cdot v_2 \cdot v_3 \dots v_k$ 的第 j 项不为零，当且仅当将所有 $\hat{v}_i$ ，即 S_i 的正交线性基所有元素拼接成大序列后，大序列的正交线性基（因

为外面还有一层 FWT 变换) 存在子序列异或和为 j 。

换句话说, 将一些线性基求交, 等价于将它们全部正交之后求并, 将结果再正交回来。

下面一道例题展示了线性基求交的应用:

例题 4.2.¹¹

给定 n 个元素是 $[0, 2^m)$ 之间非负整数的可重集 $S_1 \sim S_n$, q 次询问每次给出 x , 要求找到编号 i 最小的可重集 S_i 使得 S_i 中不存在子集异或和为 x , 不存在输出 $n+1$ 。

$n \leq 10^5, |S_i| \leq 64, m \leq 64, q \leq 10^6$ 。

对 $S_1 \sim S_n$ 分别求出一组正交线性基 $S'_1 \sim S'_n$, 询问相当于找出最小的 i 使得 $v(S_i)$ 的第 x 项系数为 0, 即最小的 i 使得 x 与 S'_i 中的某个元素不正交。这是因为 $v(S_i)$ 第 x 项为 0 等价于 $\hat{v}(S_i)$ 做 FWT 后第 x 项为 0, 等价于 x 不与 $\hat{v}(S_i)$ 这个点值序列的原序列所有元素正交, 等价于 x 不与 S_i 的正交线性基 S'_i 内所有元素正交。

从小到大扫描 i , 维护线性基 ans , 将 S'_i 内所有元素插入 ans , 询问的相当于是 ans 内何时首次出现与 x 不正交的元素, 只要记录 ans 每个元素被插入的时间显然即可 $O(qm)$ 回答询问。

建立需要做 $\sum |S_i|$ 次线性基插入, 求 n 次正交线性基, 并将 $O(nm)$ 个元素插入线性基 ans , 因此时间复杂度为 $O(nm^2 + \sum |S_i|m)$ 。

于是该问题在 $O(nm^2 + \sum |S_i|m + qm)$ 时间内得到完美解决。

4.5 本章小结

本章从一个更 OI 的角度介绍了正交线性基并展示了其两个角度的应用: 刻画点值序列以及线性基求交。其中 4.1 是一些定义与准备工作; 4.2 通过讨论点值序列进行 FWT 变换后的结果, 展现了正交线性基确实非常必要; 4.3 介绍了构建正交线性基的算法; 4.4 介绍了线性基求交。

本章内容相对第二三章较少, 这是因为正交线性基难度较高, 此前受到的研究与关注较少, 事实上正交线性基结合上第三章提到的常见优化技巧, 应该还可以解决一些更复杂、更为有趣的问题, 读者阅读完本文之后可以自行思考研究。

5 带删线性基

5.1 具有缺陷的实现方法

本节中我们来考虑如何解决以下问题:

¹¹来源: 候选队互测 2022 枪打出头鸟 <https://uoj.ac/problem/698>

例题 5.1.¹²

动态维护 (t_i, a_i) 二元组集合 S , 其中 t_i 是优先级, a_i 是 m 位二进制数。初始 S 为空, 需要支持 n 次操作, 分为三种:

- 插入操作: 向 S 内新插入一个二元组 (t_i, a_i) 。
- 删除操作: 删除 S 内一个已有的二元组, 保证存在。
- 查询操作: 给出 lim , 询问如果按照任意顺序写下 S 内所有 $t_i \leq lim$ 的 a_i , 得到的序列的所有子序列元素异或和种类数 (即构成的集合大小) $\bmod 998244353$ 。

$n, m \leq 2000$ 。

在 3.5 介绍用前缀线性基实现“脆弱”的动态线性基时就已提到过, 即使这个问题不强制在线, 用前缀线性基也必然无法解决。如果附上每个二元组被删除的时间变为三元组, 额外信息有 t_i 对应的优先级信息和“被删除的时间”两维, 询问相当于希望求一个前缀矩形, 即 x, y 坐标均小于等于对应上界的所有点的线性基信息, 显然无论对点集如何排序都不能总用一段前缀表示这个矩形信息。

为了避免标号上的麻烦, 可以按照插入操作的顺序给每个被插入的二元组一个递增的标号, 插入操作视作是将一个二元组序列中 a_i 为全零二进制数的位置修改为任意值, 删除操作视作是将一个二元组序列中一个在此前操作被修改为任意值的位置的 a_i 修改回全零二进制数。

解决这个问题大体上的思路仍然是以 t_i 为优先级建前缀线性基, 方便起见假设 t_i 互不相同, 因为实际上对于 t_i 相同的值任意给这些 t_i 定序以区分, 这样求出的结果都是正确的。

难点无疑在于删除操作, 考虑通过维护额外信息来支持删除操作。

额外对二元组序列的每个位置 i 维护 $resp_i$ 为 n 位二进制数, 表示 a_i 如果要写作一些线性基中的元素的异或和, 应包含原二元组序列的哪些元素。以及对线性基的每一位维护 $wresp_i$ 也是 n 位二进制数, 表示这个值要写成一些线性基中元素的异或和, 应包含原二元组序列哪些元素。

有必要做概念上的明确: 线性基数组上每一位的值事实上是不直接对应原序列的值, 而是原序列一些元素的异或和。假设我们按优先级 t_i 从小到大顺序插入序列中所有元素, 会有一些元素插入成功, 最后填上了线性基数组的某一位, 接下来我们称所有这样插入成功的元素的原始值 a_i 构成的集合为一组 **标准基**。那么原序列所有子集的异或和都可以被唯一表示为标准基的一个子集的异或和, $resp_i, wresp_i$ 就分别状压了 a_i 和 w_i 进行这样分解后的子集结果, w_i 按惯例指线性基中控制第 i 位的值。

在本节的问题中, 只要顺利维护了每个 a_i 是否在标准基内, 显然就可以回答所有询问, 因此可以使用以下带有缺陷的实现方法。

¹²来源: 原创题

要删除 (t_i, a_i) ，如果 a_i 不在标准基内很简单，直接删除即可。假设 a_i 在标准基内，本来在标准基内的元素不会变得不在标准基内，只可能有唯一一个 t_j 最小但仍大于 t_i 的 j ，如果其的标准基表示中包含 i ，其会被加入标准基。

这两点通过考虑， (t_i, a_i) 在标准基内当且仅当 a_i 不能写成一些 $t_j < t_i$ 的 a_j 的异或和即可看出。

而找到这个 j ，只需要枚举所有不在标准基内的元素 a_j ，找到 t_j 最小的 $resp_j$ 含有 i 的 j 即可。

如果顺利找到了，需要在标准基中用 a_j 换掉 a_i 。这会对所有 $resp_k, wresp_k$ 产生一些影响：记 $msk = 2^i \oplus resp_j$ ，对于所有 $resp_k$ 和 $wresp_k$ ，如果其本来含有 i 这一位，需要异或上 msk 。而 w 数组在本节讨论范围内可以认为没有发生变化。

剩下的情况是任何 $resp_j$ 都不包含 i 。这说明除了 i 以外，任何 a_j 在标准基内的表示都不需要 i ，所以可以直接将 a_i 从标准基中删去，从而也不需要修改 $resp$ 序列。但 $wresp$ 序列和 w 序列仍然会受到影响。

考虑找到 $wresp$ 最低的一位 $wresp_q$ 包含 i ，将高于 q 的所有包含 i 的 $wresp$ 位的值异或上 $wresp_q$ ， w 也做相应处理。这样就保证了所有 w 序列包含的值都在删去 a_i 后仍然可以被表示为 a 序列的子集异或和，并且注意到因为 q 是最低的 $wresp$ 包含 i 的位，所以更高的 w 序列的位的值不会受影响（在先前的处理中只异或上了 w_q ，而 w_q 的最高位是 q ，不会违背线性基的定义），最后将 $w_q, wresp_q$ 置零即可。

接下来考虑插入操作，因为标准基必须是按优先级 t_i 从小到大顺序插入时插入成功的 a_i 构成的集合，所以注意如果直接向线性基插入 a_i 失败了，不代表 a_i 不在标准基中：把 a_i 写成标准基子集异或和的形式，如果这个子集中 t_j 最大的项的 $t_j > t_i$ ，那么应该把 a_j 从标准基中删去，将 a_i 加入标准基。这样做的正确性分析和删除时的正确性分析一致，执行这样的操作对 $resp, wresp$ 会产生与删除部分类似的影响，处理方法也是一致的。把 a_i 写成标准基子集异或和的形式这一步通过 $wresp$ 数组即可完成。

审视插入和删除过程，可以发现执行了 $O(m)$ 次长度为 m 的二进制数的异或操作，以及 $O(n+m)$ 次长度为 n 的二进制数的异或操作，其余部分代价不超过 $O(n+m)$ 。注意由于该算法维护的二进制数长度是 n 或 m 的，而 n, m 显著比计算机字长 w 大，所以应当认为对这些二进制数进行异或运算的时间复杂度是 $O((n+m)/w)$ 而非 $O(1)$ ，因此插入和删除的时间复杂度均为 $O((n+m)^2/w)$ 。

于是在 $O((n+m)^2/w)$ 时间内解决了这一问题。

这种带删线性基的最大缺陷在于单次插入和删除的时间开销带有关于序列长度 n 的因子，且这实际上是完全无法避免的，在 n, m 不同阶时这会导致这种带删线性基几乎完全无法使用。

但就算抛开这个缺陷不谈，这种实现方法仍然还有巨大的缺陷：这样实现的带删线性基的“前缀”性是不完全的。如果将询问的内容改为“所有子序列异或和的最大值”，这一实现方法实际上就失效了：维护的 w 事实上仍然不带有“前缀”的性质，仍然是一种全局

信息。这种实现方法只保证了能够维护顺利维护所有使得线性基扩张的位置，只有在这一信息上才具有“前缀”性。

虽然有这两个缺陷，这种带删线性基也已经可以解决一些以往无法解决的问题，除本节例题以外最简单的例子就是即使目标是维护所有子序列异或和最大值，只要不管 lim 的限制即保持 lim 足够小，这个算法就完全是正确的，并且是在线算法，不是 3.5 中描述的“虚假”的前缀线性基可以媲美的。

除此之外，用它还可以解决一些更为复杂的问题，将在 5.2 中介绍。

5.2 带删前缀线性基结合数据结构

回顾 3.2 中描述的问题：

例题 5.2.

维护一个长度为 n 值域在 $[0, 2^m)$ 之间的非负整数序列 a 。支持 q 次操作分为两种：

- 修改操作：将某个 a_x 修改为 y ，保证 $y \in [0, 2^m)$ 。
- 查询操作：给出区间 $[l, r]$ ，查询从 a 的子区间 $a[l, r]$ 中任取一个子序列能得到的异或和最大值。

$n, q \leq 5 \times 10^4, m \leq 30$ 。

3.7 中描述的问题及其解法，可以认为是解决了本题保证每次修改前 a_x 均为 0 的特殊情形，现在延续 3.7 中的思路并使用 5.1 中介绍的（即使是带有缺陷的）带删线性基，可以给出该问题时间复杂度为 $O(nm + qm(\log n + m))$ 的解法。

既然要使用带删线性基，迫在眉睫的问题就是如何解决其时间开销带有其维护的序列长度 n 作为因子的问題。回顾 3.7 中的做法，实际上是对线段树上每个管辖区间 $[l, r]$ 的节点，维护两个前缀线性基，分别维护二元组序列 $\forall l \leq i \leq r, (i, a_i)$ 与 $\forall l \leq i \leq r, (-i, a_i)$ 。现在我们指出最关键的一点：线段树是分治结构，其每个节点的信息可以由其子节点信息合并而来，反映到这个问题上，就是父节点 u 上的前缀线性基可以认为维护的二元组序列并非所有 $\forall l \leq i \leq r$ 的 (i, a_i) ，而是其两个子节点 $ls(u), rs(u)$ 的标准基序列，而后的长度受到线性基大小上界的控制，至多只有 $2m$ 。

于是现在终于得以将 3.7 的算法拓展到可以将某个非零元素修改为其他任意值的情况。对线段树上所有节点维护正反两个前缀带删线性基，对于非叶节点，认为维护的二元组序列是其两个子节点的标准基序列；而对于叶节点，这个线性基只包含一个元素当然可以直接得到。

所有结构的建立是简单的，叶子节点可以直接建立，非叶子节点将左右儿子的标准基的各个元素依次插入即可。

修改时，叶节点的标准基的变化可以直接得到，将这些变化记录为若干次向其两个方向的标准基内插入或删除二元组的事件。如果对于某个子节点已经得知了所有这些事件，在其父节点的带删前缀线性基看来，相当于父节点的带删前缀线性基要维护的二元组序列发生了一些单点修改，执行这些单点修改又会导致其标准基发生若干次插入和删除事件，不断向上传递影响即可。事实上对一个序列的值做单点修改，对这个序列的标准基产生的影响当然只是 $O(1)$ 次插入和删除。所以把每次记录下来的插入和删除事件做一些抵消（要插入和删除相同的值可以都消去）就可以保证插入和删除事件每层都只进行 $O(1)$ 次。

查询时，像 3.7 中的做法一样找到询问区间 $[l, r]$ 被首次切开的位置 mid ，相当于要查询 $[L, mid], [mid + 1, R]$ 两段区间的线性基信息。比如 $[mid + 1, R]$ 的部分，其实直接取出 $[mid + 1, r]$ 这一线段树节点维护的标准基中，所有编号 $\leq R$ 的元素，插入一个新的线性基即可。这里使用 5.1 中介绍的“前缀”性不完全的带删线性基确实就足以解决问题，且就算换成“前缀”性完全的带删线性基也必然不会优化复杂度，因为最后无论如何都要执行一次 $O(m^2)$ 的线性基合并，而取出标准基的不超过 m 个元素重新插入到一个线性基中的时间开销也只有 $O(m^2)$ 。

于是修改最终被拆成 $O(m)$ 次对带删线性基进行插入与删除操作，且保证了用到的带删线性基维护的序列长度也都是 $O(m)$ 的，因此插入的时间复杂度为 $O(m \log n)$ 。查询时求 LCA 的代价可以忽略不计，时间开销是从左右两边各找出至多 m 个标准基元素插入到用于求答案的线性基中，时间复杂度为 $O(m^2)$ 。

但不幸的是这样实现会导致建立的时间复杂度以及空间复杂度都较高，为此还可以加上一个优化：底层分块，将线段树上所有管辖的区间 $[l, r]$ 满足 $r - l + 1 \leq m$ 的节点视为叶节点，在线段树上去掉其所有后代节点。修改直接 $O(m^2)$ 重建唯一的一个这样的伪叶节点的标准基信息然后再不断传递影响，查询时如果询问区间不被某个伪叶节点管辖的区间完全包含那么就完全不受影响，否则说明询问区间长度不超过 m ，直接暴力建立询问区间的线性基即可 $O(m^2)$ 回答询问。

而底层分块降低了建立的时间复杂度以及该算法的空间复杂度，首先建立所有伪叶节点的标准基信息当然时间复杂度为 $O(nm)$ ，对于剩下的非伪叶节点，其维护的二元组序列长度为 $O(m)$ ，因此建立时间复杂度为 $O(m^2)$ ，而线段树上维护区间长度超过 m 的节点只有 $O(\frac{n}{m})$ 个，因此建立的时间复杂度为 $O(nm)$ 。而该算法只维护了 $O(\frac{n}{m})$ 个带删前缀线性基，且每个带删前缀线性基维护的序列长度为 $O(m)$ ，因此空间复杂度是 $O(n)$ 的。

综上所述，该问题可以在 $O(nm + qm(\log n + m))$ 时间， $O(n)$ 空间内在线解决。相比 3.2 中描述的传统算法在时间复杂度和空间复杂度上都远更优秀，唯二的缺点在于思路与实现较为复杂，以及常数较劣。

5.3 具有完全前缀性的实现方法

接下来我们来介绍具有完全前缀性的实现方法。这一实现方法与带有缺陷的实现方法具有一定相似性，仍然是通过维护 $resp, wresp$ 这些额外信息来试图实现删除操作，但区别

在于其对线性基的形态做了更严格的约束，首先要求其具有前缀性，在具有前缀性的基础上，还规定了更严格的限制，以保证按任意顺序插入二元组得到的线性基形态（主要指的是 $w, wresp$ ）完全一致，而 5.1 中带有缺陷的实现方法对 $w, wresp$ 的限制实际上是非常宽松的。

另外，一般的前缀线性基实际上如果按照不同顺序插入二元组，得到的线性基形态也可能不相同。如果希望验证这一点，读者可以自行编写代码按照随机顺序将二元组插入前缀线性基，得到的线性基事实上几乎总是不相同。

然后，利用这个重要的“完全一致”性，可以认为在删除操作中，被删除的二元组实际上是最后一个被插入的二元组，从而认为本次删除操作其实是在撤销最后一次插入操作。但与真正的撤销最后一次插入操作相比，常规的“在插入时记录一些信息作为线索，以帮助此后的撤销”这样的方法完全不适用，因此在删除过程中只能通过维护的额外信息 $resp, wresp$ 作为线索来尝试还原这最后一次插入中究竟进行了哪些操作。

方便起见，本节只考虑以下问题：

例题 5.3.¹³

维护长度为 n ，由 n 位二进制数组成的序列 a ，支持 q 次操作分为两种：

- 修改操作：给出位置 p 以及 n 位二进制数 val ，将 a_p 修改为 val 。
- 查询操作：给出位置 p ，查询 $a[1, p]$ 所有子序列中，异或和最大的子序列。

$n, q \leq 2000$ 。

既然我们希望按任意顺序插入二元组得到的线性基形态完全一致，总得先明确这样的形态是什么。这里我们就希望得到的线性基形态是按照编号 $i = 1 \sim n$ 的顺序按照 2.3 中描述的最朴素的方式将 a_i 插入线性基，从而得到的 w 数组以及对应的 $wresp$ 方案。

但这种描述的性质太差了，实际上可以注意到这样做与下面更复杂但性质更好的描述等价：

仍然按照 $i = 1 \sim n$ 的顺序考虑，于是在考虑 a_i 前，所有 $\forall 1 \leq j < i$ 的 a_j 是如何插入线性基的已经确定。对于 a_i ，考虑将其异或上 $a[1, i)$ 的一个子序列使得得到的值 val 的最高位尽可能小，当然这样的选择方案可能有很多种， val 的值也不确定。如果 val 可以为 0，那么 a_i 当然不在标准基中，在 w 中也不存在其对应的元素，其对于 w 没有影响。否则设 val 的最高位为 p ，我们对选取 $a[1, i)$ 子序列的方式做出限制：首先必须选在标准基内的元素，其次选出的标准基内的元素必须作为某个 $p < k < n$ 的 $wresp_k$ 的最高位出现。可以发现，这样就限定了选出的 $a[1, i)$ 子序列是唯一的， w_p 必须是 a_i 异或上这样的唯一子序列的异或和， $wresp_p$ 同样描述对应的方案。因为这里的描述涉及 $wresp_k$ 的最高位，下面记 $wresp_k$ 的最

¹³来源：原创题

高位为 idx_k 。特别地，如果 $w_k = 0$ ，认为 $idx_k = +\infty$ 。可以发现 idx_k 其实就表明了 w_k 首次被填上值时，对应的 i 是多少。

仍然将修改操作视作一次插入操作（保证 a_p 修改前全 0）和一次删除操作（认为修改为的值 val 全零），既然希望将删除视作撤销最后一次插入，先考虑插入操作如何进行。

现在的线性基形态是限制更严格的前缀线性基，因此插入操作大体上与前缀线性基的插入是类似的。维护 $cur, cresp, q$ 分别表示当前要插入的值，这个值真正作为原序列子序列异或和时的表示方法，以及当前要插入的值对应的下标。初始 cur 为要插入的二进制数， $cresp = 2^p$ ， $q = p$ 。降序考虑 i 从 $n-1$ 到 0：

- 若 cur 的第 i 位为 0，那么注意 $idx(i, n)$ 构成的集合中，只有 p 被加入了集合，以及 q 被从集合中删除（如果 $p = q$ ，即无事发生）。那么如果 $wresp_i$ 是包含 q 的，应该用 $cresp$ 所对应的包含 p 和 q ，且第 $i \sim n-1$ 位均为零的子序列异或和表达式从 $wresp_i$ 中消去 q ， w_i 也相应要异或上 cur ，来保持更严格的限制。如果 $wresp_i$ 包含 q ，隐含了 $q < idx_i$ ，所以异或之后 idx_i 当然不会发生变化。

注意这一条体现了限制比前缀线性基更严格之处。

- 接下来是 cur 的第 i 位为 1 的情况。和前缀线性基一样，这里要根据 idx_i 和 q 的大小关系来考虑要不要交换 cur, w_i 以及 $cresp, wresp_i$ ，还有 q, idx_i 。无论是否交换，将 cur 异或上 w_i ， $cresp$ 异或上 $wresp_i$ 。这里的疑惑可能在于如果 $q > idx_i$ ，那么不需要交换，但需不需要和 cur 的第 i 位为 0 中的情况一样检查是否 $wresp_i$ 是否包含 q 。但再回顾一下定义： idx_i 是 $wresp_i$ 的最高位，即可发现这当然不需要检查。

如果这一步结束之后发现 $q = +\infty$ ，需要立即退出。

在退出之后，如果 $q = +\infty$ 说明只有 p 被加入了标准基，标准基中没有发生任何删除。于是不需要检查所有不在标准基内的元素关于标准基内元素的表法是否有一些变得不合法。否则对于 $1 \leq q \leq n$ 的情况，说明 p 被加入了标准基， q 本来在标准基中但现在被删除（ $p = q$ 会对应一个插入了 p 但它不在标准基中的无意义情况，可能会比较奇怪，判掉就好了），通过 $cresp$ 就得到了 $resp_q$ 这一将 q 表示为一些标准基内元素的表法，同时还需要检查所有不在标准基内的元素，如果其 $resp_i$ 包含 q ，需要异或上 $cresp$ 来消去 q 。

接下来考虑删除操作，因为希望将其视作撤销最后一次插入，我们需要先还原在最后一次插入结束后， $cur, cresp, q$ 三者的具体值。首先需要区分的两种情况是插入后标准基中发生了删除的情况，与没有发生删除的情况。这一步与 5.1 中找出删除后哪个元素被加入标准基的方式是完全一致的：枚举 $\forall p < i \leq n$ ，若 a_i 不在标准基内且 $resp_i$ 包含 p ，取其中最小的 i 就是最后一次插入结束后的 q ，即唯一一个从标准基内删除的元素，如果不存在说明插入结束后 $q = +\infty$ 。如果 $1 \leq q \leq n$ ，顺便也得知道了 $cur = 0$ ， $cresp$ 是 $resp_q$ 翻转第 q 位后的结果； $q = +\infty$ 时由于循环在某个 i 就做了立刻退出，我们还希望还原这个 i 。

在这里需要注意，对于 $1 \leq q \leq n$ 的情况需要做没有本质困难但必不可少的对 $resp_i$ 修改的撤销。正着做的过程是给所有 $resp_i$ 含有 q 的 $resp_i$ 异或上 $cresp$ ，同时注意到 $cresp$ 必

然含有 p ，且 $resp_i$ 在异或前必然不含 p 。于是倒着做的过程就是给所有 $resp_i$ 含有 p 的 $resp_i$ 异或上 $cresp$ 。

在这里需要介绍一个重要性质：在最后一轮插入 a_p 时， cur 任何时刻都一定包含 p 这一位，而在插入 a_p 前所有 $resp_i, wresp_i$ 都一定不包含 p 这一位，所以插入后所有 $resp_i, wresp_i$ 是否包含 p 这一位实际上是非常重要的标识符，其起到了提示插入过程，作为线索的作用。观察插入过程，每当 $cresp, wresp_i$ 发生交换时， $wresp_i$ 会从本来不含 p 这一位变得含有 p 这一位。另一种让 $wresp_i$ 含有 p 这一位的方式是在插入到第 i 位时 cur 的第 i 位为 0，此时如果 $wresp_i$ 包含当时的 q ， $wresp_i$ 会被异或上 $cresp$ 从而含有 p 这一位。 $wresp_i$ 只能通过这两种方式含有 p 这一位。

那么在 $q = +\infty$ 时还原退出的 i 实际上就非常简单了：退出的 i 一定是最后一次 $cresp, wresp_i$ 发生交换的位置，于是 $wresp_i$ 一定含有第 p 位，而 $\forall 0 \leq j < i$ 的 $wresp_j$ 当然不含有第 p 位。得到退出的 i 之后，在这里做一些处理之后，可以认为 $1 \leq q \leq n$ 和 $q = +\infty$ 两种情况接下来没有区别。

接下来考虑如何在每一步过程中倒推，设现在已知 $cur, cresp, q$ 是插入过程中进行完第 i 轮之后的值，考虑如何通过 $w_i, wresp_i$ 还原出 $cur', cresp', q'$ 是插入过程中进行完第 i 轮之前的值，并将 $w_i, wresp_i$ 也还原到进行完第 i 轮之前。一个非常平凡的情况是 $w_i = 0$ ，那么在插入前一定也 $w_i = 0$ ，且插入过程对 $w_i, wresp_i$ 没有发生任何影响，于是可以直接跳过这一位。

下面所有带引号的表示执行插入操作第 i 轮前的值，不带引号的表示执行完插入操作第 i 轮后的值。

我们先假设已经通过某种手段，判断出了进行插入的第 i 轮前 cur' 的第 i 位是 0 还是 1，并分别考虑在这两种情况下如何还原。

- cur' 的第 i 位是 0：那么需要判断出插入的第 i 轮前，是否 $wresp'_i$ 包含 q' ，从而 $wresp'_i$ 和 w'_i 分别异或上了 $cresp', cur'$ 。但其实注意到如果异或了，那么 $wresp_i$ 必然含有 p 这一位，否则必然不含，于是这部分的还原可以完成。
- cur' 的第 i 位是 1：那么需要判断插入的第 i 轮中，是否发生了交换。还是从标识符的角度考虑，如果交换了必然 $wresp_i$ 含有 p ，否则必然不含，于是这部分的还原也可以完成。

现在最终的困难在于如何确定插入的第 i 轮前 cur' 的第 i 位的值。我们依次检查以下几点：

- 如果 $cresp$ 包含 idx_i 这一位，可以发现必然 cur' 的第 i 位为 1。但对于 $cresp$ 不包含 idx_i 的情况，有可能是一般的 cur' 的第 i 位为 0 的情况；也可能是 cur' 的第 i 位为 1，但 $q' < idx_i$ 于是发生了交换，而本来的 $wresp'_i$ 就是包含 q' 的，交换后异或反而消除了。

- 不然，如果 $wresp_i$ 不包含 p ，可以发现 cur' 的第 i 位必然是 0。因为现在假设 cur' 的第 i 位为 1，必然是发生了交换的分支，而这种情况必然 $wresp_i$ 包含 p 。
- 如果还是无法区分，假设 cur' 的第 i 位为 0，那么必然使用 $cresp'$ 执行了对于 q' 的消去，此时隐含了 $q' < idx'_i$ ，于是 $cresp = cresp'$ 在 $idx_i = idx'_i$ 及之后的位上均必然为 0。而假设 cur' 的第 i 位是 1，那么必然发生了 q' 与 idx'_i 之间， $cresp'$ 与 $wresp'_i$ 之间的交换，于是 $cresp = wresp'_i$ 在 idx'_i 这一位上必然为 1，而 $idx'_i > idx_i = q'$ ，因此 $cresp$ 一定在某个 idx_i 之后的位上为 1。只需检查 $cresp$ 的最高位是否不低于 idx_i 即可分辨。

现在所有需要的信息均得到了还原，因而删除操作可以被顺利地视作撤销最后一次插入操作并执行。无论是插入操作和删除操作，时间瓶颈均为执行 $O(n)$ 次长度为 n 的二进制数的异或操作，因而单点修改操作可在 $O(n^2/w)$ 时间内完成。

既然顺利维护了比一般前缀线性基限制更强的线性基形态，查询操作也可以在 $O(n^2/w)$ 时间内轻易解决。事实上将查询的内容换成其他线性基可维护的经典信息该算法也完全可以胜任。

由于建立还需要 $O(n^3/w)$ 时间，综上所述，该例题在 $O((n+q)n^2/w)$ 时间内得以在线解决。

5.4 本章小结

本章聚焦于在线性基相关问题中极其常见的带删线性基问题，重点讨论了维护的集合大小与需要维护的二进制数长度同阶的情形。事实上带删线性基在第三章就已被讨论过：

- 3.2 的例题在询问均查询一段前缀时便可视作和 5.3 中例题类似的带删线性基问题，但当时讨论的设定是需要维护的集合大小远大于需要维护的二进制数长度，因而线性基合并的时间开销虽然较大但仍可接受，因此当时使用的方法其实是避免执行删除，利用线段树的结构全部转化为线性基合并。
- 3.5 中的例题三也是带删线性基问题的形式，但该例题并没有强制占用掉线性基的“前缀”性，即查询时总查询整个集合，且也并不强制在线，因此可以利用前缀线性基的技巧以删除时间为优先级避免真的执行删除操作。

所以此前的方法均为试图寻求转化，避免直面删除操作。但事实上这些转化的方法大多非常脆弱：对于全部转化为合并的方法，就算抛开转化为合并本身乘上的 $\log n$ 倍的操作代价不谈，在维护的二进制数位长较大时急剧增加的合并代价也会使这种方法完全无法使用（合并需要 $O(m^2)$ 次长度为 m 的二进制数运算，而常规插入只需要 $O(m)$ 次长度为 m 二进制数运算）；而对于使用前缀线性基技巧，以删除时间为优先级建立前缀线性基的方法，简单加上强制在线即可让这种方法完全束手无策。

因此本章在 5.1 和 5.3 介绍了两种功能强大的带删除线性基。因为“前缀”性是线性基所具有的非常重要的性质，还着重考虑了“前缀”性在两种带删除线性基上能否得以保持。其中 5.1 一节介绍的带删线性基原理、实现较为简单，但具有缺陷，只具有部分“前缀”性。5.3 一节介绍的带删线性基基于 5.1 中介绍的带有缺陷的版本并加以改进，使其具有完全“前缀”性，但原理、实现更为复杂。

但需要注意的是，我认为带删线性基单次操作的时间开销无法避免地与其维护的元素个数有关，这导致其在传统的需要维护的集合大小远大于要维护的二进制数长度的情形下并不优秀，因此类似的方法此前也并未引起广泛关注。

具体而言，我尝试在互联网上搜索带删线性基，搜索到的关于带删线性基的内容只有 UOJ453 这道例题及其衍生博客，其中使用的带删线性基与 5.1 一节中介绍的类似，但其不具有任何“前缀”性。我在这种带删线性基基础上加以改进，先后发明了 5.1,5.3 中介绍的两种具有部分“前缀”性和完全“前缀”性的带删线性基，应当在一定程度上填补了这部分研究的空白。

本章剩下的 5.2 一节介绍了如何使用 5.1 中介绍的具有部分“前缀”性的带删线性基配合线段树优化 3.2 中提到的经典问题，得到的新解法相比传统做法少一个 $\log$ 因子，该做法（据我所知的范围内）同样首次由我提出。

本章正文只有三节，相比二三四章均更少，且均为本文原创内容，难以在网络上搜索到相关资料。这表明了带删线性基这一方向受到的研究仍然较少，希望读者在这一方向多多思考，创造出更多成果！

6 总结

本文依次介绍了线性基基本的构建方法与基础应用、线性基上的常见优化技巧、正交线性基、带删线性基。前三部分主要对信息学竞赛中已有的理论做规范化的梳理整合，最后一部分在带删线性基这一此前较少受到研究的领域提出了一些全新的理论，对一些经典问题做出了优化。

希望本文能起到抛砖引玉的作用，给与读者在这一方向一些启发，也希望未来能出现更多线性基的有趣题目与拓展。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练彭思进、杨耀良的指导。

感谢父母对我的培养和教育。

感谢学校的栽培，李建老师的教导和同学们的帮助。

感谢章绍嘉、汪苏轶同学与我进行交流讨论，并为本文审稿。

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] 苏焜, 浅谈「正交线性基」<https://www.luogu.com/article/qb29lo53>
[2] 吴与伦, 动态线性基学习笔记 <https://mem.ac/oi/algorithm/dynamic-linear-basis/>

序列相关问题的一些思路总结

浙江省诸暨市海亮高级中学 范斯喆

摘要

本文主要对信息学竞赛中与序列相关的问题展开研究, 列举了一些常见的解题思想与分析技巧, 并结合例题阐述了其作用与运用场合。

1 前言

信息学竞赛中的问题千变万化, 解法复杂多样, 这使许多选手面对问题时不知如何下手。然而, 不同的问题背后也有其共通之处。如果能提炼出解题过程背后的思想与技巧, 在今后的解题中将大有裨益。本文将对关于序列问题的一些技巧试着进行总结。

关于序列的问题一般有两类: 一种是已知某序列, 对在其上定义的某种组合对象进行统计; 另一种则是已知某序列满足的性质, 对所有这样的序列进行统计。第二节和第三节分别归纳了这两类问题的一些思想方法。

其中, 也有一些问题涉及到序列上的操作。思考这样的问题时, 需要对其进行适当的转化, 或找到操作中的关键组合量, 使之容易被描述。第四节讲述了若干种针对操作特点的解题思路。

2 针对已知序列

2.1 统计区间贡献

一般来说, 对于统计区间贡献的问题, 最直接的想法就是考察单个区间的贡献是否具有可减性。

如果贡献具有可减性, 例如区间和、区间乘积等, 就可以用前缀和作差, 化为两端点处的贡献, 然后对每个位置作为端点的情况统计另一个端点的贡献总和。

如果贡献不具有可减性, 则可以尝试扫描线或分治。这两种方法都比较广为人知。

例题 2.1.1 (大海的深度¹) 给定长度为 n 的序列 $a_1, a_2, \dots, a_n$ 。有 m 次操作，每次给出 x, y ，表示将 a_x 修改为 y 。求每次操作后 a 序列所有区间的最小值之和。

$$1 \leq n, m \leq 2 \times 10^5, 1 \leq a_i, y \leq 10^6, 1 \leq x \leq n。$$

本题的区间贡献形式为区间最小值，不满足可减性。有修改，扫描线维护较为困难。因此将所有操作离线，然后分治统计答案。

分治时需要计算所有跨过分治中心的贡献。区间最小值较难处理，考虑枚举值域上的点 x ，并求出最小值大于等于 x 的区间个数。

枚举 x ，对每个时刻 i 维护 p_i ， q_i 和 w_i 三个数组。其中 p_i 表示第 i 时刻分治中心左边最近的 $\leq x$ 的数的距离， q_i 表示第 i 时刻分治中心右边最近的 $\leq x$ 的数的距离， w_i 表示第 i 时刻的贡献总和。

x 增大时， p 和 q 的变化均为区间取 $\min$ ， w_i 则要变为 $w_i + p_i \times q_i$ 。这两种操作均可用 Segment Tree Beats 在 $O(n \log n)$ 的时间复杂度内维护。算上外层分治，总复杂度为 $O(n \log^2 n)$ 。

另外，本题也存在 $O(n \sqrt{n})$ 的算法（见参考文献 [1]），利用了“操作序列分块”这一种数据结构技巧。

对于有关区间最值的问题，利用笛卡尔树也是一种常见的思路。这是因为笛卡尔树的性质： $a_l, a_{l+1}, \dots, a_r$ 中的最小值等于 l 和 r 在小根笛卡尔树上的 LCA 对应的 a 值。

例题 2.1.2 (Subsequence²) 给定长度为 n 的序列 $a_1, a_2, \dots, a_n$ 。要选出 m 个位置 $b_1, b_2, \dots, b_m$ 满足 $1 \leq b_1 < b_2 < \dots < b_m \leq n$ ，定义一个选择方案的价值为

$$\sum_{i=1}^m (m \cdot a_{b_i}) - \sum_{i=1}^m \sum_{j=1}^m \min(a_{\min(b_i, b_j)}, a_{\min(b_i, b_j)+1}, \dots, a_{\max(b_i, b_j)})$$

求最大价值。

$$1 \leq n, m \leq 4000, 1 \leq a_i < 2^{31}。保证所有 a_i 互不相同。$$

本题中较难处理的部分显然是区间最小值的和。考虑建出笛卡尔树，于是一对 (b_i, b_j) 会贡献到它们的 LCA 处。那么可以在每个点处统计以该点为 LCA 的所有 (b_i, b_j) 的贡献。

因此考虑在笛卡尔树上进行动态规划。设 $dp_{l,r,i}$ 表示在下标 $[l, r]$ 内选 i 个数能得到的最大价值， p 表示 $a_l, a_{l+1}, \dots, a_r$ 中最小元素的下标，则转移方程如下：

- 如果不选 p ，则有 $dp_{l,p-1,j} + dp_{p+1,r,i-j} - 2 \times j \times (i-j) \times a_p \rightarrow dp_{l,r,i}$ ；
- 如果选 p ，则有 $dp_{l,p-1,j} + dp_{p+1,r,i-j-1} + m \times a_p - (2 \times (j+1) \times (i-j) - 1) \times a_p \rightarrow dp_{l,r,i}$ 。

这是一个树形背包，因此时间复杂度为 $O(n^2)$ 。

¹UOJ892

²CF1580D

2.2 统计二元组贡献

一般的思路是：枚举二元组的其中一个量，计算所有合法的另一个量的贡献。

在这类问题中，有时要利用贡献的单调性，排除不影响最优解的情况。此处举一例加以说明。

例题 2.2.1 (Classical³) 给定 n 个整数 $a_1, a_2, \dots, a_n$ ，求两两的最小公倍数的最大值。即

$$\max_{1 \leq i < j \leq n} \text{lcm}(a_i, a_j)$$

$$2 \leq n \leq 10^5, 1 \leq a_i \leq 10^5。$$

首先注意到

$$\text{lcm}(a_i, a_j) = \frac{a_i \times a_j}{\text{gcd}(a_i, a_j)}$$

因此考虑枚举 $g = \text{gcd}(a_i, a_j)$ ，每次加入所有是 g 倍数的数。问题变为求 $\text{gcd}(a_i, a_j) = 1$ 情况下 $a_i \times a_j$ 的最大值。

排序并删除重复的数，然后从大到小枚举 a_i 。容易发现，如果此时找到了 $a_j > a_i$ 且 $\text{gcd}(a_i, a_j) = 1$ ，则所有满足 $a_i < a_k < a_j$ 的 a_k 对答案都不会有贡献了。

因此，可以用一个栈维护当前可能贡献到答案的 a_j 。每枚举一个 a_i ，就尝试弹出栈顶，如果栈中剩下的元素均不和 a_i 互质就放弃弹栈。

那么问题变为：在栈中插入和删除元素，询问是否存在与 a_i 互质的数。

“是否存在”并不方便统计，考虑改求“存在几个与 a_i 互质的数”。运用莫比乌斯反演，不难得到

$$\sum_{x \in S} [\text{gcd}(x, a_i) = 1] = \sum_{x \in S} \sum_{d | \text{gcd}(x, a_i)} \mu(d) = \sum_{d | a_i} \mu(d) \sum_{x \in S} [d | x]$$

因此维护每个数的倍数有多少个在栈中即可。

设 $A = \max_{i=1}^n a_i$ ，则单次时间复杂度为 $O(\sum_{i=1}^A d(i)) = O(A \log A)$ 。总复杂度为 $O(A \log^2 A)$ 。

考虑优化，注意到如果答案为 $\text{lcm}(a_i, a_j)$ ，则必存在 $x | a_i, y | a_j$ ，满足 $\text{gcd}(x, y) = 1, x \cdot y = \text{lcm}(a_i, a_j)$ 。因此事先将所有 a_i 的约数加入序列，即可做到 $O(A \log A)$ 。

3 针对待序列

对于涉及“待确定的序列”的问题，解题关键一般在于找到合理的生成顺序。换句话说，应当以什么样的顺序去确定序列中的每个元素？

比较直接的想法是按下标从小到大确定所有值，也就是令状态为序列的一个前缀。

³CF1285F

然而，在许多问题中，前缀（或后缀）对未确定部分的影响并不容易表示成较少的几个量。因此需要考虑其他的生成顺序。

同时，在确定序列时，还有两种视角：

1. “绝对”插入，也就是每次加入新元素时，放在序列中的一个空位；
2. “相对”插入，也就是每次加入新元素时，放在已有序列中的两个数之间（或者边界）。

多数问题中，两种视角往往只有一种易于分析，很少成为思考瓶颈，但仍在此列出。更详细的分析可见参考文献 [2]。

3.1 按值确定

与按下标确定对应，也有一种思路是按值的大小顺序确定序列中的所有元素。

例题 3.1.1 (波浪⁴) 定义一个排列 $p_1, p_2, \dots, p_n$ 的价值为

$$\sum_{i=1}^{n-1} |p_{i+1} - p_i|$$

给定 n, m ，求长度为 n ，价值为 m 的排列个数。

$1 \leq n \leq 100, 0 \leq m \leq n^2$ 。

由于绝对值的性质，确定排列中相邻两个数的大小关系会很有用。考虑将 1 到 n 从小到大依次插入到排列中，依此设计动态规划。

因为代价和相邻的数字有关，因此考虑“相对”插入，记录当前插入的数形成了几个连续段。且由于边界处并不会计算贡献，还要记录目前未填的边界位置数。

具体地，设 $dp_{i,j,k,s}$ 表示目前已经填入了 1 到 i ，形成了 j 个连续段，边界位置还剩 k 个未填，已插入的数的总贡献为 s 的方案数。通过分类讨论，得出从 $dp_{i-1,j,k,s}$ 开始的转移：

- 一边与边界相邻，一边与空位相邻。转移到 $dp_{i,j+1,k-1,s-i}$ ，方案数为 k ；
- 一边与边界相邻，一边与连续段相邻。转移到 $dp_{i,j,k-1,s+i}$ ，方案数为 k ，要求 $j > 0$ ；
- 两边均与空位相邻。转移到 $dp_{i,j+1,k,s-2i}$ ，方案数为 $j-1+k$ ；
- 一边与连续段相邻，另一边与空位相邻。转移到 $dp_{i,j,k,s}$ ，方案数为 $2(j-1)+k$ ，要求 $j > 0$ ；
- 两边均与连续段相邻。转移到 $dp_{i,j-1,k,s+2i}$ ，方案数为 $j-1$ 。

这样就可以以 $O(n^4)$ 的时间复杂度解决这个问题。

还有一类问题同样是按值确定，但并不需要依次决定所有数的位置，只需考虑一部分关键位置，其余位置则可以用数学方法或预处理确定。

⁴ZJOI2012

例题 3.1.2 (Emotional Fishermen⁵) 给定 n 个整数 $a_1, a_2, \dots, a_n$ 。要求重排 a 中的所有数，使得对任意 $2 \leq i \leq n$ ，设

$$b_i = \max_{j=1}^{i-1} a_j$$

则须有 $2a_i \leq b_i$ 或 $2b_i \leq a_i$ 。问重排的方案数对 998244353 取模后的结果。

$2 \leq n \leq 5 \times 10^3, 1 \leq a_i \leq 10^9$ 。

本题的条件显然与前缀最大值有关。对于重排后的序列，考虑其中的相邻两个前缀最大值 a_i 和 a_j ，则须有 $2a_i \leq a_j$ ，且对于所有满足 $i < k < j$ 的 k 都有 $2a_k \leq a_i$ 。受到该性质启发，考虑对所有前缀最大值进行 DP。

记 $\#_I$ 表示在区间 I 中的数个数。设 f_i 表示从前往后确定，当前前缀最大值为 a_i 时，插入所有 $[1, a_i/2]$ 的数的方案数。

枚举上一个前缀最大值 a_j 。注意到，如果已经填入 a_j 和 $[1, a_j/2]$ 中的数，则 a_i 必须放在最靠前的空位， $(a_j/2, a_i/2] \setminus \{a_j\}$ 中的数则可以在剩余空位中任意插入。因此

$$f_i = \sum_{2a_j \leq a_i} f_j \cdot \frac{(\#_{(a_j/2, a_i/2]} - 2)!}{(\#_{(a_i/2, +\infty)} - 1)!}$$

用前缀和优化即可做到 $O(n)$ 的时间复杂度。

本题也可以从后往前确定序列，状态和转移较为类似，只是插入方式由“绝对”插入变为了“相对”插入。

3.2 同时按值和下标确定

既然存在按下标和按值确定的方法，那么是否存在同时按两者确定序列的方法呢？答案是可以的。

例题 3.2.1 (Permutation Oddness⁶) 定义一个排列 $p_1, p_2, \dots, p_n$ 的价值为

$$\sum_{i=1}^n |i - p_i|$$

给定 n, m ，求长度为 n ，价值为 m 的排列个数。

$1 \leq n \leq 50, 0 \leq m \leq n^2$ 。

为方便叙述，令 p' 为 p 的逆排列，即 $p'_{p_i} := i$ 。

绝对值的贡献与 i 和 p_i 的大小有关。因此可以从小到大枚举 i ，并确定 p_i 和 p'_i 与 i 的大小关系。

⁵CF1437F

⁶ABC134F

考虑排列的组合意义，本质上是将 n 个元素和 n 个位置进行匹配。针对这一点尝试设计动态规划。

设 $dp_{i,j,s}$ 表示已经确定所有满足 $k \leq i$ 且 $p_k \leq i$ 的 (k, p_k) ，且这样的 k 有 j 个，所有 $\leq i$ 的数对对价值的贡献总和为 s 的方案数。从 $dp_{i-1,j,k}$ 开始的转移有：

- i 号元素和 i 号位置匹配。转移到 $dp_{i,j+1,s}$ ，方案数为 1；
- 两者匹配的对象都在 i 之后。转移到 $dp_{i,j+2,s+2i}$ ，方案数为 $(i-1-j)^2$ ；
- 两者匹配的对象都在 i 之前。转移到 $dp_{i,j,s-2i}$ ，方案数为 1；
- 两者匹配的对象一个在 i 之前，一个在 i 之后。转移到 $dp_{i,j+1,s}$ ，方案数为 $2(i-1-j)$ 。

总时间复杂度 $O(n^4)$ 。

4 针对序列操作

4.1 分析操作本身

在有关操作的问题中，操作本身的性质无疑是十分重要的。

下面将介绍几种思路。这些思路较为一般化，不仅在与序列操作相关的问题中有用武之地，在与其他结构上的操作相关的问题中也能大展身手。

4.1.1 减少操作可能性

如果问题中存在多种操作，应当优先考虑这些操作能否互相替代，排除一些无必要性的操作方案。这样有时能大大减少操作的可能性，更方便分析性质。

例题 4.1.1 (移除石子 $k=0$ 且 $l_i=r_i$ 的部分⁷) 给定长度为 n 的序列 $a_1, a_2, \dots, a_n$ 。你可以进行如下两种操作：

1. 选择一个 $a_i \geq 2$ ，将 a_i 减去至少 2；
2. 选择一个区间 $[l, r]$ 满足 $r-l \geq 2$ 且 $a_l, a_{l+1}, \dots, a_r$ 均大于 0，将 $a_l, a_{l+1}, \dots, a_r$ 减去 1。

求你能否使得所有 a_i 变为 0。

要求时间复杂度为 $O(n)$ 。

观察到两种操作的自由度都非常高。考虑能否等效替代，从而排除部分可能性。易得：

1. 显然可以优先进行操作二，最后进行所有操作一；

⁷NOI2022

2. 每个 a_i 只需要进行至多一次操作一。也就是说，只要操作二能够将所有 a_i 变为不是 1 的数，就成功了；
3. 操作二的区间长度至少为 3，而一次长度为 6 及以上的操作二能拆成若干次长度为 3, 4, 5 的操作二的并。因此，操作区间长度可以被限定为 3, 4, 5；
4. 不会进行多次相同的操作二，因为可以用几次操作一代替。

至此已经可以设计 DP。设 $dp_{i,j,k}$ 表示已经决策了左端点在 1 到 i 之间的操作二，其中 j 个的右端点被钦定大于 i ，其中 k 个的左端点为 i （右端点须大于 i ）。转移较为容易。

考虑 j 和 k 的范围。由于左端点与右端点在 i 两侧的操作二只可能有 $[i-3, i+1], [i-2, i+1], [i-2, i+2], [i-1, i+1], [i-1, i+2], [i-1, i+3]$ 六种，所以 $j \leq 6, k \leq 3$ 。

不过状态可以继续缩减。注意到如果从某个位置开始同时有长度为 4 和 5 的操作二，可以改为一次操作一，以及两次长度为 3 和 4 的操作二。进一步，枚举所有包含这六个区间中至少三个的方案，发现都可以被若干次操作一和不超过两次操作二替代。于是 $j \leq 2, k \leq 2$ 仍然能产生所有有必要性的操作方案，状态数减少为 $9n$ 。

本题之后的 DP 套 DP 与主题无关，故不赘述（见参考文献 [3]）。值得注意的是，DP 套 DP 对内层 DP 的状态数要求非常高，因此在思考的前期就应当以最优状态数为目标，也就是要尽可能减少操作的可能性。

4.1.2 逆操作

当问题中的操作性质不够优秀时，可以试着描述逆操作，也许对解题有更大的帮助。

例题 4.1.2 (基础循环结构练习题⁸) 有一个长度为 n 的序列 $a_1, a_2, \dots, a_n$ ，其中 $a_i = i$ 。给定一个目标序列 $b_1, b_2, \dots, b_n$ ，你需要在进行不超过 2001 次操作后，将序列 a 变为序列 b 。

你可以进行如下两种操作：

1. 选取一个非负整数 x ，对所有 i ，将 a_i 变为 $a_i + x$ ；
2. 选取一个非负整数 x ，对所有 i ，将 a_i 变为 $a_i \bmod x$ 。

$1 \leq n \leq 1000, 0 \leq b_i < 10^5$ 。

本题的操作看似简单，实际上却不容易分析。倒过来考虑整个过程，操作变为：

1. 选取一个非负整数 x ，对所有 i ，将 a_i 变为 $a_i - x$ ，要求 x 小于等于 a 中的最小值；
2. 选取一个非负整数 x ，对所有 i ，将 a_i 变为 $a_i + s \cdot x$ ，其中 s 是对每个 i 独立选取的任意非负整数。要求 x 大于 a 中的最大值。

⁸第五届图灵杯趣味网络国际邀请赛高级组第二题

新的问题是从序列 b 出发，要变为 $a_i = i$ 的序列。

注意到将 a_i 变大的操作较为自由，尝试考虑这样的操作能做到什么。通过一些模拟和思考，发现可以将最小值增大到比原先的最大值大 1。

具体地，设原本最小值为 mn ，最大值为 mx 。则先进行操作一，选择 mn 。然后进行操作二，选择 $mx - mn + 1$ ，只增大 mn 而不增大其他数。

不断重复这个过程，可以解决 b_i 递增的情况。而 b_i 不递增的情况也很好处理：一开始进行一次操作二，选择一个充分大的 S ，将 a_i 变为 $a_i + i \cdot S$ 。这样就转化为了递增的情况。

最后将所有操作步骤倒过来，得到原问题的解。需操作 $2n + 1$ 次，在限制范围内。

4.1.3 研究操作功能

当操作的动机不够明显时，有一种思考方式较为常用：考虑多次操作能实现的功能。譬如，上一个例题中的操作二自由度很高，但与要达成的目标却没有什么直接联系。这时可以和第一种操作绑定，形成一个“将最小值增大到比原先的最大值大 1”的操作，提升了与操作目标的相性。

不过要注意，将多次操作绑定在一起也是一种弱化，在需要优化时要追本溯源，回过头思考原先操作的性质。

例题 4.1.3 (Circular Xor Reversal⁹) 有一个长度为 n 的环形序列 $a_0, a_1, \dots, a_{n-1}$ ， $a_i = 2^i$ 。你需要通过若干次操作将 a 翻转，也就是使其变为 $a_i = 2^{n-i-1}$ 。

你可以进行以下操作至多 2.5×10^5 次：选择 $[0, n-1]$ 中的整数 i ，将 a_i 的值改为 $a_i \oplus a_{(i+1) \bmod n}$ 。其中， $x \oplus y$ 表示 x 与 y 的异或值。

$1 \leq n \leq 400$ 。

为方便叙述，下文中的下标均对 n 取模。

翻转序列等价于 $\lfloor \frac{n}{2} \rfloor$ 次交换操作。但“相邻两个数的异或”和交换相去甚远，难以分析。

不过，由异或可以联想到一种经典的交换两个变量的方式：依次执行 $x := x \oplus y, y := x \oplus y, x := x \oplus y$ 。这种方法的正确性容易验证。

想交换 a_i 和 a_j ，只需用操作实现 $a_i := a_i \oplus a_j$ 。通过手动模拟不难得出一种构造：

1. 从 $j-1$ 开始逆序操作，到 i 为止；
2. 从 $i+1$ 开始顺序操作，到 $j-1$ 为止；
3. 从 $j-2$ 开始逆序操作，到 i 为止；
4. 从 $i+1$ 开始顺序操作，到 $j-2$ 为止。

记上面的一系列操作为 $f(i, j)$ 。设 l 表示 i 到 j 的距离，即 $(j-i) \bmod n$ 。则执行 $f(i, j)$ 需要的操作次数为 $4l-4$ 。

⁹CF1748F

按照上文中的方法，交换 a_i 与 a_j 需要 $4n + 4l - 12$ 次操作。然而如果 $l > n-l$ ，可以改为交换 a_j 与 a_i 。于是只需要 $4n + 4 \min(l, n-l) - 12$ 次操作，总共需要 $2.5n^2 + O(n)$ 次操作。

尝试优化。注意到当 $i > 0$ 时，做完 $f(i, n-i-1)$ 的前三步刚好也做完了 $f(i-1, n-i)$ 的第一步。所以可以通过合理的操作顺序去除重复的操作过程。分析得到总操作次数为 $1.25n^2 + O(n)$ 次，足以通过。

4.2 寻找组合量

在一些问题中，操作对序列本身的影响可能不容易刻画，直接对序列展开分析比较困难。遇到这种情况，有一种常见的思路是：寻找一种组合量来对序列进行补充描述，或是替代原序列参与操作。

4.2.1 逆序对

在涉及交换操作的问题中，考虑逆序对数往往能够切中问题的要害。从这个角度入手，有时能够得到判断无解的条件，有时可以给出操作次数的下界，从而确定思路。

例如，有以下两个经典结论：

1. 有一个序列，需要通过交换相邻两个数将其升序排序，最少操作次数等于逆序对数；
2. 有一个元素互不相同的序列，交换其中任意两个元素，会改变逆序对数的奇偶性。

例题 4.2.1 (The same permutation¹⁰) 有一个长度为 n 且元素互不相同的序列 $a_1, a_2, \dots, a_n$ 。一种交换操作是：选择两个数 i, j ，满足 $1 \leq i < j \leq n$ ，交换 a_i 和 a_j 。

易知有 $\frac{n(n-1)}{2}$ 种本质不同的交换操作。要求合理安排这些交换操作的次序，使得依次执行这些操作后的序列与原序列相同。

$1 \leq n \leq 1000$ 。

每次交换操作都将改变序列中逆序对数的奇偶性。那么有解的必要条件即 $\frac{n(n-1)}{2}$ 是偶数。因此 $n \bmod 4$ 为 2 或 3 时一定无解。

接下来，针对 $n \bmod 4$ 为 0 或 1 的情况，考虑四个一组分组，不难枚举得出构造方案（见参考文献 [4]）。这部分内容与主题无关，故略去。

4.2.2 构造新序列

另一种常见的技巧是依据原序列构造一个新序列。具体的构造方法较为繁多，但主要方向是让操作变得尽可能简洁。下面以两个例题作为例子，列举一些常用的构造手段。

¹⁰CF804E

例题 4.2.2 (方差¹¹) 给定长度为 n 的序列 $a_1, a_2, \dots, a_n$, 保证 $1 \leq a_1 \leq a_2 \leq \dots \leq a_n$ 。

你可以进行如下操作：选择一个 i , 满足 $1 < i < n$, 将 a_i 改为 $a_{i-1} + a_{i+1} - a_i$ 。

求在任意次操作之后, 该数列的方差最小值乘 n^2 后是多少。其中方差定义为：数列中每个数与平均值的差的平方的平均值, 即

$$D = \frac{1}{n} \sum_{i=1}^n \left(a_i - \frac{1}{n} \sum_{j=1}^n a_j \right)^2$$

要求时间复杂度为 $O(nA \min(n, A))$, 其中 $A = \max_{i=1}^n a_i$ 。

首先对方差公式进行变形, 以得到更易计算的形式。不难得出

$$n^2 D = n \sum_{i=1}^n a_i^2 - \left(\sum_{i=1}^n a_i \right)^2$$

问题中的操作比较奇怪, 难以描述。考虑构造序列 $b_i = a_i - a_{i-1}$, 则通过模拟容易验证: 选择 i 操作对 b 序列的影响为交换 b_i 和 b_{i+1} 。于是最终序列的差分数组必然是 b 的任意一个排列。

接下来观察方差最小时 b 序列的性质。通过组合意义与数学推导都不难发现差分数组应当形成单谷, 即先减后增。

所以可以将 b 从小到大排序, 然后用 DP 决策每个 b_i 放在当前序列的左边还是右边。

设 $dp_{i,j}$ 表示已经放入 b_1 到 b_i , 此时 a 之和为 j 时, a 的平方和的最大值。考虑转移:

- 将 b_i 放在左边, 则 $dp_{i-1,j} + 2 \cdot j \cdot b_i + i \cdot b_i^2 \rightarrow dp_{i,j+i \cdot b_i}$;
- 将 b_i 放在右边, 则 $dp_{i-1,j} + s_i^2 \rightarrow dp_{i+1,j+s_i}$ 。

其中 $s_i = \sum_{j=1}^i b_j$ 。

时间复杂度为 $O(n^2 A)$, 但注意到不为 0 的 b_i 不超过 A 个, 所以可以做到 $O(nA \min(n, A))$ 。

例题 4.2.3 (千百¹²) 给定长度为 n 的环形序列 $a_0, a_1, \dots, a_{n-1}$ 。为方便叙述, 下文中的下标均对 n 取模。

你需要使得对于所有 $0 \leq i < n$, $a_i \cdot a_{i+1} < 0$ 。

你可以进行的操作是：选择一个 i 满足 $0 \leq i < n$, 将 a_{i-1} 和 a_{i+1} 同时减去 a_i , 然后使 a_i 变为 $-a_i$ 。

有 m 次修改, 每次给定 p, v , 将 a_p 和 a_{p+1} 同时加上 v 。修改之间互不影响。

每次修改后, 求你要达成目标最少需进行几次操作。

$3 \leq n \leq 10^5, 1 \leq m \leq 10^5, \sum_{i=1}^{n-1} a_i \leq 10^8$ 。对于每次询问, $0 \leq p < n, |v| \leq 10^8$ 。

¹¹NOIP2021

¹²互测 2022

首先 n 为奇数时显然无解。

观察单次操作的性质, 其相当于让 a_{i-1}, a_i, a_{i+1} 分别减去 $a_i, 2a_i, a_i$ 。

注意到奇数项与偶数项之和不变。进一步, 考察每个前缀的奇数项之和与偶数项之和的差, 即

$$b_i = \sum_{j=0}^i (-1)^j a_j$$

则一次操作对会将 b_{i-1}, b_i, b_{i+1} 变为 b_i, b_{i-1}, b_{i+1} , 也就是交换了 b 中的相邻两个数。而对最终结果的要求等价于要求 b 具有严格单调性。

考虑到 a 是环形序列, 不妨断环为链, 并将序列向两边无限循环延展。令所有下标模 n 相同的位置为一个族, 则操作转化为每次可以交换两个相邻的族, 需要给序列排序。

现在的问题与经典的排序交换次数问题十分类似。类似逆序对计数, 只需统计两两族之间需要交换几次, 通过代数推导容易将问题变为二维偏序的形式。

而单次修改对 b 的影响为: 将一个族内的所有数加上相同的一个数。于是只需解决在线的二维偏序问题, 使用树套树或离线分治均可做到 $O(n \log^2 n)$ 。

4.2.3 维护二元组集合

有一部分特殊的问题, 涉及到类似“将排列变为逆排列”的操作。当出现这种操作时, 考虑维护 (i, a_i) 形成的二元组集合, 则这种操作相当于交换所有二元组的两个元素, 有时会有更好的性质。

例题 4.2.4 (Latin Square¹³) 给定一个 $n \times n$ 的矩阵 a , 每行每列都是一个 1 到 n 的排列。有 m 次操作, 操作有如下六种:

1. 将矩阵的每行向右循环移位一格;
2. 将矩阵的每行向左循环移位一格;
3. 将矩阵的每列向下循环移位一格;
4. 将矩阵的每列向上循环移位一格;
5. 将矩阵的每行变为其逆排列;
6. 将矩阵的每列变为其逆排列。

求最后的矩阵。

$1 \leq n \leq 1000, 1 \leq m \leq 10^5$ 。

考虑维护三元组集合 $(i, j, a_{i,j})$ 。则六种操作对某个三元组 (x, y, z) 的影响为:

¹³CF1458C

1. 将 (x, y, z) 变为 $(x, (y + 1) \bmod n, z)$;
2. 将 (x, y, z) 变为 $(x, (y - 1) \bmod n, z)$;
3. 将 (x, y, z) 变为 $((x + 1) \bmod n, y, z)$;
4. 将 (x, y, z) 变为 $((x - 1) \bmod n, y, z)$;
5. 将 (x, y, z) 变为 (x, z, y) ;
6. 将 (x, y, z) 变为 (z, y, x) ;

于是维护操作所需的信息量大幅减少。只需维护一个大小为 3 的排列，以及每个位置的变化量，即可 $O(1)$ 维护单次操作。

所有操作结束后可以 $O(n^2)$ 复原矩阵，总时间复杂度为 $O(n^2 + m)$ 。

4.2.4 建图

以序列为依据建图，在很多问题中都是一种思考策略。最常见的建图手段便是置换环。遇到排列问题，考虑置换环有时能得到十分优雅简洁的结论。

例题 4.2.5 (Inverse Transformation¹⁴) 有一个长度为 n 的排列 $a_1, a_2, \dots, a_n$ 。定义排列的一次变化为将所有 a_i 同时变为 a_{a_i} 。

定义 $\sigma(x) = a_x$ ， $f(x)$ 为最小的正整数 m 使得 $\sigma^m(x) = x$ ，其中 $\sigma^m(x) = \sigma(\sigma^{m-1}(x))$ 。

现在给出初始排列变化 k 次之后的排列。求一个合法的初始排列，使得 $\sum_{i=1}^n \frac{1}{f(i)}$ 最小。

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

考虑建图，对每个 i ，从 i 向 a_i 连边。此图中每个点的入度和出度均为 1，因此其是由若干个环构成。

本题中定义的函数 $f(x)$ 看上去和此图有密切的联系。仔细思考不难得出 $f(x)$ 等于 x 所在的环的大小。因此 $\sum_{i=1}^n \frac{1}{f(i)}$ 就是图中环的个数。

接下来从置换环的视角考虑排列的变化。注意到一次变化后，大小为偶数的环会分裂为两个大小相等的环，而大小为奇数的环不会分裂。

将分裂的过程倒过来思考。为了让初始排列的环数尽量小，应当尽可能合并最终图中的环。对于大小为奇数的环，可以将 2^i ($i \leq k$) 个合并在一起。对于大小为偶数的环，则只能一次合并 2^k 个，如果数量不是 2^k 的倍数则无解。

知道了原排列中环的构成之后，环上元素的具体顺序可以用数学方法简单求出。

时间复杂度为 $O(n)$ 。

当然，也有许多其他的建图方法。同之前一样，这些方法的关键在于通过建图更好地描述操作的性质，不可生搬硬套。下面列举两个例题，希望读者能领悟到这类方法的精髓。

¹⁴CF1787F

例题 4.2.6 (排序大师¹⁵) 给定长度为 n 的排列 $a_1, a_2, \dots, a_n$ 。你需要将 a 排序。一次操作为：选定 $1 \leq i \leq j < k \leq l \leq n$ ，交换 $a_{i \sim j}$ 和 $a_{k \sim l}$ 。也就是令序列变为

$$a_1, a_2, \dots, a_{i-1}, a_k, a_{k+1}, \dots, a_l, a_{j+1}, a_{j+2}, \dots, a_{k-1}, a_i, a_{i+1}, \dots, a_j, a_{l+1}, a_{l+2}, \dots, a_n$$

求最小的操作次数，并给出一种方案。

$1 \leq n \leq 2000$ 。

本题需要最小化操作次数，这意味着我们需要先对操作次数下界进行分析。

观察到操作对 $a_i \rightarrow a_{i+1}$ 这样的边只改变了 4 条，因此考虑以此为依据建图。然而这样建图后所有状态都是一条链，不利于分析。

尝试调整。在排列最前面加入一个 0，最后面加入一个 $n + 1$ ，再按照 $a_i + 1 \rightarrow a_{i+1}$ 建图。此时最终状态对应 $n + 1$ 个自环。

考察操作对图的影响，不难得出一次操作相当于以下两步：交换 a_i 和 a_k 的入边，再交换 $a_j + 1$ 和 $a_l + 1$ 的出边。

由于一次交换至多让环数增加或减少 1，因此一次操作会让环数改变不超过 2，且不改变其奇偶性。因此，如果原排列的环数和 $n + 1$ 的奇偶性不同，则无解。否则需要找到一种每次将环数增加 2 的操作方案。

之后的构造通过手动模拟不难得出，故不在此赘述。

例题 4.2.7 (Flip and Reverse¹⁶) 给定长度为 n 的 01 序列 $a_1, a_2, \dots, a_n$ 。

你可以进行任意次操作，每次操作选择一个 0 和 1 的个数相同的区间，将其中所有数反转（1 变为 0，0 变为 1），然后再将其翻转。

求可能的最终序列中字典序最小的。

$1 \leq n \leq 5 \times 10^5$ 。

对于 01 序列（或者括号序列），一种常见的思路是将 0 视为 +1，1 视为 -1，然后做前缀和。设求出的前缀和为序列 s ，则一次操作等价于选择 $s_{l-1} = s_r$ 的 $[l, r]$ ，然后翻转 $s_l, s_{l+1}, \dots, s_{r-1}$ 。

对所有 $0 \leq i < n$ ，从 s_i 到 s_{i+1} 连一条边，建出一张图。在此图上考虑单次操作的影响。

不难发现，一个 0 和 1 的个数相同的区间，就是图上的一个环。一次操作相当于将这个环上的所有边的方向反转。

而若干次操作后的 s 序列，则是将边视为无向边后，图中的一条欧拉路径。

注意到 a 序列的字典序最小，等价于 s 序列的字典序最小。于是问题变为求字典序最小的欧拉路径，贪心即可。时间复杂度为 $O(n)$ 。

¹⁵THUPC2024 初赛

¹⁶CF1458D

5 总结

本文对信息学竞赛中与序列问题相关的思想与技巧进行了初步的总结，并给出了一些例题用于解说。其中不少的思路都能应用到其他问题中，具有一定的可拓展性。

信息学竞赛中的问题浩如烟海。限于篇幅和笔者水平，这样的总结难以做到面面俱到。希望本文可以起到抛砖引玉的作用，也希望今后的选手能通过学习本文中的一些思想与技巧提升自己的解题水平。

6 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢家人对我的培养、关心和支持。

感谢包括黄志刚老师、傅振滔老师、陈惇显老师在内的海亮信奥教练团队对我的指导。

感谢曹立学长对本文提出的宝贵意见。

感谢我学习信息学竞赛的途中遇到过的所有人。

参考文献

- [1] 周康阳. UOJ NOI Round #8 Day2 题解. <https://zhoukangyang.blog.uoj.ac/blog/9191>, 2024.
- [2] 曹立. dp 题方法总汇. <https://www.luogu.com.cn/article/ki71nw88>, 2023.
- [3] 魏佳泽. P8497 [NOI2022] 移除石子. <https://www.luogu.com.cn/article/wb0zxabp>, 2022.
- [4] saliii. Codeforces Round #411 Editorial. <https://codeforces.com/blog/entry/51846>, 2017.
- [5] 曹立. 构造题方法总汇. <https://www.luogu.com.cn/article/o9mj50m1>, 2023.

浅谈 Thue-Morse 序列的性质及其应用

浙江省衢州第二中学 郑钧

摘要

围绕 Thue-Morse 序列，本文引入几种 Thue-Morse 序列的等价定义，探究了 Thue-Morse 串的字符串性质、Thue-Morse 生成函数的性质，以及其在等幂和问题、无平方无限串构造问题等方面的应用。

1 Thue-Morse 序列

Thue-Morse (TM) 序列，也被称为 Prouhet-Thue-Morse 序列，是在 $\{0, 1\}$ 上定义的无穷序列，于 1851 年由 Prouhet 首次研究。尽管该序列形式较为简单，但在数论、单词组合学和组合博弈论等领域都有不同的应用，可见其有着不少奥妙的性质。

由于 Thue-Morse 序列仅有 $\{0, 1\}$ 两种值，在一些问题中将其看成一个无限字符串往往会更加直观，故其也被称为 Thue-Morse 串或 Thue-Morse 无限串。下面是 Thue-Morse 串的前若干项。

01101001100101101001011001101001...

2 Thue-Morse 序列的定义

对于 Thue-Morse 序列，存在若干等价的定义方式。研究不同的问题时，借助合适的定义方式可以使问题的形式更加清晰。研究同一个问题时，使用不同的定义方式也可以得到不同的解决方法。在这一节，将会列举几种常见的定义方式。

2.1 奇偶校验定义

定义 2.1. 若 $T_i (i \in \mathbb{N})$ 表示 Thue-Morse 序列的第 i 项，则 T_i 等于 i 在二进制下数位和模 2 的值。

例子 由于 $(11)_{10} = (1011)_2$, 故 $T_{11} = (1 + 0 + 1 + 1) \bmod 2 = 1$ 。

方便起见, 下文中的 T 均表示 Thue-Morse 序列。

由该定义可得, 在二进制下, $2n$ 仅比 n 末尾多了一个 0, 并不影响数位和, 故 $T_{2n} = T_n$; 同理, $T_{2n+1} \neq T_n$ 。

这种新的定义方式称为递归式定义。

2.2 递归式定义

定义 2.2. 定义在 $\{0, 1\}$ 上的序列 T , 若满足 $T_0 = 0, T_{2n} = T_n, T_{2n+1} \neq T_n$, 则 T 即为 Thue-Morse 序列。

由该定义, 可以自然地推出态射法定义。

2.3 态射法定义

令 μ 表示一个态射变换, 满足 $\mu(0) = 01, \mu(1) = 10$ 。

且 $\mu(w_1 w_2 \cdots w_n) = \mu(w_1) \mu(w_2) \cdots \mu(w_n)$, 则 $\mu^k(0)$ 的前若干项为:

$$\begin{aligned}\mu^0(0) &= 0 \\ \mu^1(0) &= 01 \\ \mu^2(0) &= 0110 \\ \mu^3(0) &= 01101001 \\ &\vdots\end{aligned}$$

于是, 态射法定义也呼之欲出。

定义 2.3. Thue-Morse 序列等于 $\mu^\infty(0) = \mu(\mu(\cdots \mu(0) \cdots))$, 即让 0 作用 μ 无限次。

对比 $S_n, \mu(S)_{2n}, \mu(S)_{2n+1}$ 可以发现, $\mu(S)_{2n} = S_n, \mu(S)_{2n+1} \neq S_n$, 正好是定义 2.2 的形式。

另外, 需要证明 $\mu^k(0)$ 是 $\mu^{k+1}(0)$ 的前缀, 以保证 $\mu^\infty(0)$ 是良定义的 (或者说 Thue-Morse 序列是态射变换 $0 \rightarrow 01, 1 \rightarrow 10$ 的不动点)。证明是简单的, 此处略去。

2.4 反转法定义

沿用态射法定义的思路, 若已经求出了 $\{T_i\}_{i=0}^{2^n-1}$, 考虑如何求出 $\{T_i\}_{i=0}^{2^{n+1}-1}$ 。由定义 2.1 可以得到: 对于 $i = 0, 1, \cdots, 2^n - 1$, $T_{i+2^n} \neq T_i$ 。故 $\{T_i\}_{i=2^n}^{2^{n+1}-1}$ 即为 $\overline{\{T_i\}_{i=0}^{2^n-1}}$, 其中 $\bar{a}$ 表示将 a 序列 0, 1 反转得到的序列。因此, $\{T_i\}_{i=0}^{2^{n+1}-1} = \{T_i\}_{i=0}^{2^n-1} \overline{\{T_i\}_{i=0}^{2^n-1}}$, 其中 ab 表示序列 a, b 顺次拼接得到的序列。这就是反转法的定义方式。

定义 2.4. 令 $TM_0 = \{0\}$, $TM_{2^{n+1}} = TM_{2^n} \overline{TM_{2^n}}$ 。则 Thue-Morse 序列就等于 TM_{2^∞} 。

3 Thue-Morse 序列与等幂和问题

在进一步探究 Thue-Morse 序列的性质之前, 先通过 Thue-Morse 序列在等幂和问题中的应用, 感受 Thue-Morse 序列优美的对称性。

3.1 等幂和问题

给定正整数 n, k , 求两个不交的大小为 n 的集合 A, B , 使得对于 $i = 0, 1, \cdots, k$ 满足:

$$\sum_{a \in A} a^i = \sum_{b \in B} b^i$$

3.2 基于 Thue-Morse 序列构造的解

由于 Thue-Morse 序列生成方式 (反转法) 的高度对称性, 对于等幂和问题的 n 为 2 的非负整数幂的情况, 可以借助 Thue-Morse 序列构造出一组优美的解。

对于 $k \in \mathbb{N}$, 定义集合 $X_k = \{n | n \in [0, 2^{k+1}) \cap \mathbb{N}, T_n = 0\}, Y_k = \{n | n \in [0, 2^{k+1}) \cap \mathbb{N}, T_n = 1\}$ 。

首先证明 $|X_k| = |Y_k|$, 根据 Thue-Morse 序列反转法的定义方式, 发现 TM_{2^1} 中 0、1 的个数是相等的, 而若 TM_{2^n} 中 0、1 的个数相同, 则 $TM_{2^{n+1}} = TM_{2^n} \overline{TM_{2^n}}$ 中 0、1 的个数也是相等的。所以, 对于 $n \in \mathbb{N}^*$, TM_{2^n} 中 0、1 的个数是相等的, 即 $|X_k| = |Y_k| = 2^k$ 。

定理 3.2.1. 对于任意度数不超过 k 的多项式 f , 有:

$$\sum_{n \in X_k} f(n) = \sum_{n \in Y_k} f(n)$$

例子 $X_2 = \{0, 3, 5, 6\}, Y_2 = \{1, 2, 4, 7\}$, 当 $f(x) = x^2 - x + 1$ 时:

$$\begin{aligned}\sum_{n \in X_2} f(n) &= f(0) + f(3) + f(5) + f(6) = 1 + 7 + 21 + 31 = 60 \\ \sum_{n \in Y_2} f(n) &= f(1) + f(2) + f(4) + f(7) = 1 + 3 + 13 + 43 = 60\end{aligned}$$

证明. 当 $k = 0$ 时, f 即为常数多项式, 显然满足条件。

当 $k \geq 1$ 时, 假设 $k-1$ 满足条件, 设多项式 $g(n) = f(n+2^k) - f(n)$ 。容易发现, g 的 k 次项系数为 0, 故 $\deg g \leq k-1$, 所以有:

$$\begin{aligned}
 \sum_{n \in X_{k-1}} g(n) &= \sum_{n \in Y_{k-1}} g(n) \\
 \sum_{n \in X_{k-1}} f(n+2^k) - f(n) &= \sum_{n \in Y_{k-1}} f(n+2^k) - f(n) \\
 \sum_{n \in X_{k-1}} f(n+2^k) + \sum_{n \in Y_{k-1}} f(n) &= \sum_{n \in Y_{k-1}} f(n+2^k) + \sum_{n \in X_{k-1}} f(n) \\
 \sum_{n \in (Y_k \setminus Y_{k-1})} f(n) + \sum_{n \in Y_{k-1}} f(n) &= \sum_{n \in (X_k \setminus X_{k-1})} f(n) + \sum_{n \in X_{k-1}} f(n) \\
 \sum_{n \in Y_k} f(n) &= \sum_{n \in X_k} f(n)
 \end{aligned}$$

因此 k 也满足条件。

□

备注 这是 Christopher 给出的证明¹，较为简洁优美，值得学习。

对于等幂和问题，实际上就是 $f(n) = n^i$ 的情况。

例题 3.1 (愤怒的小 N^2)。给定正整数 $n(\log_2 n < 5 \times 10^5)$ 和多项式 $f(\deg f \leq 500)$ ，其中 n 以二进制形式给出，即给定集合 S 满足 $n = \sum_{k \in S} 2^k$ 。求：

$$\sum_{i=0}^{n-1} T_i f(i) \pmod{(10^9 + 7)}$$

解法 首先，为了更接近等幂和问题，可以将答案转化成：

$$\frac{\sum_{i=0}^{n-1} f(i) - \sum_{i=0}^{n-1} (-1)^{T_i} f(i)}{2}$$

$\sum_{i=0}^{n-1} f(i)$ 仅需使用经典的拉格朗日插值法求出，接下来考虑 $\sum_{i=0}^{n-1} (-1)^{T_i} f(i)$ 如何计算。

对于 n 的每个为 1 的二进制位 k ，要求出：

$$\sum_{i=s}^{s+2^k-1} (-1)^{T_i} f(i) = \sum_{i=0}^{2^k-1} (-1)^{T_{i+s}} f(i+s)$$

其中， $s = \sum_{k' \in S \wedge k' > k} 2^{k'}$ 。所以， $(-1)^{T_{i+s}}$ 可以拆分为 $(-1)^{T_s} (-1)^{T_i}$ 。故贡献转化为：

$$\sum_{i=0}^{2^k-1} (-1)^{T_{i+s}} f(i+s) = (-1)^{T_s} \left(\sum_{i \in X_{k-1}} f(i+s) - \sum_{i \in Y_{k-1}} f(i+s) \right)$$

容易发现， $f(i+s)$ 的度数也不超过 $\deg f$ 。故当 $\deg f \leq k-1$ 的时候，贡献为 0。

而对于 $\deg f > k-1$ 的情况，利用二项式定理可以递推求解，但这和本文内容并无关联，且难度较低，此处略去。

至此，复杂度做到了 $O(\log n + \deg^3 f)$ ，即可通过此题。

¹Christopher Williamson, [An Overview of the Thue-Morse Sequence](#), 2012

²[NOI Online 2021 提高组] 愤怒的小 N

3.3 扩展

进一步扩展此类构造方式。具体地，对于不小于 2 的整数 b ，设 $s_b(n)$ 表示 n 在 b 进制下各数位之和模 b 的值。对于 $k \in \mathbb{N}$ ， $i = [0, b) \cap \mathbb{N}$ ，定义集合 $S_{k,i} = \{n | n \in [0, b^{k+1}) \cap \mathbb{N}, s_b(n) = i\}$ 。

定理 3.3.1. 对于任意度数不超过 k 的多项式 f ，有：

$$\sum_{n \in S_{k,0}} f(n) = \sum_{n \in S_{k,1}} f(n) = \cdots = \sum_{n \in S_{k,b-1}} f(n)$$

证明. 当 $k = 0$ 时，显然满足条件。

当 $k \geq 1$ 时，对于任意常数 C_1, C_2 ，设 $g(n) = f(n+C_1) - f(n+C_2)$ ，有 $\deg g \leq k-1$ ，故：

$$\begin{aligned}
 \sum_{n \in S_{k-1,0}} g(n) &= \cdots = \sum_{n \in S_{k-1,b-1}} g(n) \\
 \sum_{n \in S_{k-1,0}} f(n+C_1) - f(n+C_2) &= \cdots = \sum_{n \in S_{k-1,b-1}} f(n+C_1) - f(n+C_2)
 \end{aligned}$$

则对于任意的 $i = 1, 2, \dots, b-1$ ，满足：

$$\begin{aligned}
 &\sum_{n \in S_{k,0}} f(n) - \sum_{n \in S_{k,i}} f(n) \\
 &= \sum_{t=0}^{b-1} \sum_{n \in S_{k-1,t}} (f(n + ((0-t) \bmod b) \times b^k) - f(n + ((i-t) \bmod b) \times b^k)) \\
 &= \sum_{t=0}^{b-1} \sum_{n \in S_{k-1,0}} (f(n + ((0-t) \bmod b) \times b^k) - f(n + ((i-t) \bmod b) \times b^k)) \\
 &= \sum_{t=0}^{b-1} \sum_{n \in S_{k-1,0}} f(n + ((0-t) \bmod b) \times b^k) - \sum_{t=0}^{b-1} \sum_{n \in S_{k-1,0}} f(n + ((i-t) \bmod b) \times b^k) \\
 &= \sum_{p=0}^{b-1} \sum_{n \in S_{k-1,0}} f(n + p \times b^k) - \sum_{q=0}^{b-1} \sum_{n \in S_{k-1,0}} f(n + q \times b^k) \\
 &= 0
 \end{aligned}$$

第一步的含义是枚举第 $0, 1, \dots, k-1$ 位数值之和模 b 的值，反推出第 k 位的值。第二步则是对于固定的 t ， $((0-t) \bmod b) \times b^k$ 和 $((i-t) \bmod b) \times b^k$ 都是常数。应用上述结论，可将 $S_{k-1,t}$ 改写为固定的 $S_{k-1,0}$ 。最后拆开即可发现结果为 0。因此， k 也满足条件。

□

4 Thue-Morse 的字符串性质

在这一节中，主要探究 Thue-Morse 序列作为字符串的性质，所以接下来将 T 看作一个无限字符串。

4.1 一些记号

记 ε 表示空字符串。

记 Σ^* 表示字符集为 Σ 的有限非空字符串构成的集合。

记 Σ^∞ 表示字符集为 Σ 的有限非空字符串和无限字符串构成的集合。

对于 $w \in \Sigma^*$ ，记 $|w|$ 表示 w 的长度，记 w^R 表示将 w 翻转得到的字符串。

对于 $w \in \Sigma^\infty$ ，记 $\bar{w}$ 表示将 w 中 0,1 翻转得到的字符串。

对于 $u \in \Sigma^*, v \in \Sigma^\infty$ ，记 uv 表示将 u, v 顺次拼接得到的字符串。

对于 $w \in \Sigma^\infty$ ，记 $w(i)$ 表示 w 的编号为 i 的字符，从 0 开始编号，即：

- 若 $w \in \Sigma^*$ ，则 $w = w(0)w(1) \cdots w(|w| - 1)$ ；
- 若 $w \in \Sigma^\infty \setminus \Sigma^*$ ，则 $w = w(0)w(1)w(2) \cdots$ 。

对于 $w \in \Sigma^\infty$ ，记 $w(l:r)$ 表示 w 的编号为 l 到 r 的子串，即 $w(l:r) = w(l)w(l+1) \cdots w(r)$ 。

特别地，对于 $w \in \Sigma^\infty \setminus \Sigma^*$ ， $w(l:\infty)$ 表示 $w(l)w(l+1)w(l+2) \cdots$ 。

特别地，对于空字符串 ε ，有：

- $|\varepsilon| = 0$ ， $\varepsilon^R = \varepsilon$ ， $\bar{\varepsilon} = \varepsilon$ ；
- $\varepsilon u = u$ ， $u\varepsilon = u$ ， $\varepsilon\varepsilon = \varepsilon$ 。

此处列举一些下文会用到的事实：

- $(w^R)^R = \overline{(\bar{w})} = w$ ；
- $\overline{w^R} = (\bar{w})^R$ ；
- $|w_1w_2 \cdots w_n| = |w_1| + |w_2| + \cdots + |w_n|$ ；
- $(w_1w_2 \cdots w_n)^R = w_n^R w_{n-1}^R \cdots w_1^R$ 。

4.2 Thue-Morse 子串结构

首先，回到定义 2.4，可以得到 TM_{2^n} 的回文性质：

- $TM_{2^{2n}} = TM_{2^{2n}}^R$ ；
- $TM_{2^{2n+1}} = \overline{TM_{2^{2n+1}}^R}$ 。

使用归纳法证明，此处给出简略的过程：

$$\begin{aligned} TM_{2^{2n}}^R &= (TM_{2^{2n-1}} \overline{TM_{2^{2n-1}}})^R = \overline{TM_{2^{2n-1}}^R} TM_{2^{2n-1}}^R = TM_{2^{2n-1}} \overline{TM_{2^{2n-1}}} = TM_{2^{2n}} \\ \overline{TM_{2^{2n+1}}^R} &= \overline{(TM_{2^{2n}} \overline{TM_{2^{2n}}})^R} = \overline{(TM_{2^{2n}}^R TM_{2^{2n}})^R} = TM_{2^{2n}}^R \overline{TM_{2^{2n}}^R} = TM_{2^{2n}} \overline{TM_{2^{2n}}} = TM_{2^{2n+1}} \end{aligned}$$

接着，可以发现 Thue-Morse 子串的两条较为明显的性质。

定理 4.2.1. 对于 $w \in \{0,1\}^*$ ，若 w 为 T 的子串，则 $\bar{w}$ 也为 T 的子串。

证明. 考虑使用定义 2.4 帮助证明。找到一个足够大的 n ，使得 w 是 TM_{2^n} 的子串（容易证明这是一定能够找到的），则 $\bar{w}$ 也是 $\overline{TM_{2^n}}$ 的子串。而 $TM_{2^{n+1}} = TM_{2^n} \overline{TM_{2^n}}$ ，故 $\bar{w}$ 是 $TM_{2^{n+1}}$ 的子串。所以 $\bar{w}$ 是 T 的子串。□

定理 4.2.2. 对于 $w \in \{0,1\}^*$ ，若 w 为 T 的子串，则 w^R 也为 T 的子串。

证明. 同样考虑使用定义 2.4 帮助证明。找到一个足够大的 n ，使得 w 是 TM_{2^n} 的子串。由于 $TM_{2^{2n}} = TM_{2^{2n}}^R$ ，故 w 也是 $TM_{2^{2n}}^R$ 的子串，同时翻转两者，得到 w^R 是 $TM_{2^{2n}}$ 的子串。所以 w^R 是 T 的子串。□

由定理 4.2.1、4.2.2 可得，对于 $w \in \{0,1\}^*$ ， w 、 w^R 、 $\bar{w}$ 和 $\bar{w}^R$ 要么同时是 T 的子串，要么同时不是。

在进一步探究 Thue-Morse 串的子串结构之前，需要先了解准 TM 前缀串和 TM 拆分的概念。

4.2.1 准 TM 前缀串和 TM 拆分

定义 4.1 (准 TM 前缀串). 对于 $w \in \{0,1\}^*$ ，称 w 为准 TM 前缀串当且仅当 w 为 T 或 $\bar{T}$ 的前缀。

关于准 TM 前缀串，有如下两个明显的事实：

- 对于 $w \in \{0,1\}^*$ ，若 w 为准 TM 前缀串，则 $\bar{w}$ 也为准 TM 前缀串。
- 对于 $u, v \in \{0,1\}^*$ ，若 uv 为准 TM 前缀串，则 u 也为准 TM 前缀串。

接着，容易得到准 TM 前缀串的性质定理和判定定理。

定理 4.2.3 (准 TM 前缀串的性质定理). 对于 $w \in \{0,1\}^*$ ，若 w 为准 TM 前缀串，则 $\forall i, j \in [0, |w|) \cap \mathbb{N}$ ， $w(i) \oplus w(j) = T(i) \oplus T(j)$ 。其中， $\oplus$ 表示按位异或运算，下同。

证明. 若 w 为 T 的前缀, 则 $w(i) = T(i)$, $w(j) = T(j)$; 若 w 为 $\bar{T}$ 的前缀, 则 $w(i) = T(i) \oplus 1$, $w(j) = T(j) \oplus 1$. $\square$

定理 4.2.4 (准 TM 前缀串的判定定理). 对于 $w \in \{0, 1\}^*$, 若 $\forall i = 0, 1, \dots, |w| - 1$, 满足 $w(0) \oplus w(i) = T(i)$, 则 w 为准 TM 前缀串。

证明. 若 $w(0) = 0$, 则 $w(i) = T(i)$; 若 $w(0) = 1$, 则 $w(i) = \overline{T(i)}$. $\square$

定义 4.2 (TM 拆分). 对于 $w \in \{0, 1\}^*$, 称二元组 (u, v) 为 w 的一个 TM 拆分, 当且仅当 $uv = w$, 且 u^R 和 v 均为准 TM 前缀串。

接下来, 就能够精确刻画 Thue-Morse 串的子串形态了。

定理 4.2.5 (Thue-Morse 子串形态). 对于 $w \in \{0, 1\}^*$, w 为 T 的子串当且仅当 $|w| = 1$ 或 w 存在至少一个 TM 拆分。

证明. $|w| = 1$ 的情况显然成立, 接下来考虑 $|w| > 1$ 的情况。

必要性 若 w 为 T 的子串, 不妨设 $w = T(l : r)$, 在 $(l, r] \cap \mathbb{N}$ 中找到 lowbit^3 最大的 i . 令 $u = T(l : i - 1)$, $v = T(i : r)$, 设 $\text{lowbit}(i) = k$. 由于 $\text{lowbit}(i + 2^k) > k$, 故 $r < i + 2^k$. 所以对于 $j = 0, 1, \dots, r - i$, 有 $j < 2^k$, 因此 $v(j) = T(i + j) = T(i) \oplus T(j)$. 由定义 4.1 可得, v 是准 TM 前缀串。同理, $\text{lowbit}(i - 2^k) > k$, 故 $l \geq i - 2^k$. 所以对于 $j = 0, 1, \dots, i - l - 1$, 有 $j < 2^k$. 故 $u^R(j) = T(i - 1 - j) = T(i - 1) \oplus T(j)$, 可得 u^R 是准 TM 前缀串。所以, (u, v) 是 w 的一个 TM 拆分。

充分性 若 $u^R, v \in \{0, 1\}^*$, 满足 $w = uv$, 且 u^R 和 v 均为准 TM 前缀串。找到最小的满足 $2^k \geq \max(|u|, |v|)$ 的正奇数 k , 设 $s = T(0 : 2^k - 1)$. 则 u, v 都是 s 或 $\bar{s}$ 的前缀。由定义 2.4 可知, $s = TM_{2^k}$. 由于 k 为奇数, 故 $\bar{s} = s^R$. 将 $T(0 : 7 \times 2^k - 1)$ 用 $s, \bar{s}$ 拼接表示, 如图 1。

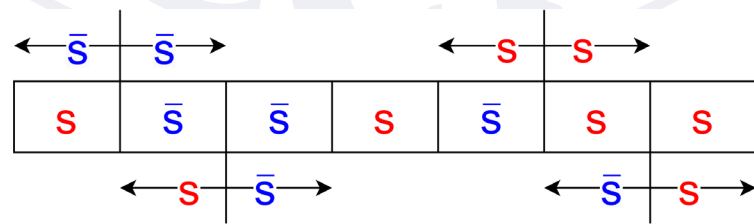


图 1: $T(0 : 7 \times 2^k - 1) = s\bar{s}\bar{s}s\bar{s}s s$

容易发现, 无论 u^R, v 是 s 还是 $\bar{s}$ 的前缀, 图 1 中的四个位置, 一定有一个匹配的。故 $w = uv$ 一定是 $T(0 : 7 \times 2^k - 1)$ 的子串。 $\square$

³对于 $n \in \mathbb{N}^*$, $\text{lowbit}(n)$ 表示 n 在二进制下最低一位 1 表示的数值。特别地, 此处定义 $\text{lowbit}(0) = \infty$ 。

备注 事实上, 在充分性的证明中, 无需限制 k 为奇数。当 k 为偶数时, 图 1 所示的四个拆分点, 也是包含所有情况的, 只不过此时的 $s^R = s, \bar{s}^R = \bar{s}$, 略有不同, 但证明过程一致。另外, 由该定理可以得到, 长度为 n 的 Thue-Morse 子串个数不超过 $4n$ 。

例子 对于 $w = T(10 : 19) = 0101101001$, w 共有两个 TM 拆分, 分别是 $(01, 01101001)$ 和 $(010110, 1001)$ 。

4.2.2 Thue-Morse 子串的 TM 拆分个数

更进一步地, 考虑如果 $w \in \{0, 1\}^*$ 有两个 TM 拆分, 不妨设 $w = xyz$, $x, y, z \in \{0, 1\}^*$, w 的两个 TM 拆分为 (x, yz) 和 (xy, z) . 则 yz 和 $(xy)^R = y^R x^R$ 均为准 TM 前缀串。故 y 和 y^R 均为准 TM 前缀串。考虑 y 应该满足什么条件。

引理 4.2.6. 对于 $w \in \{0, 1\}^*$, 若 w, w^R 均为准 TM 前缀串, 则 $|w|$ 为 2 的非负整数幂。

证明. 对于 $i = 0, 1, \dots, |w| - 1$, 不妨设 $w(i) = x \oplus T(i)$, $w^R(i) = y \oplus T(i)$, 故 $x \oplus T(i) = w(i) = w^R(|w| - 1 - i) = y \oplus T(|w| - 1 - i)$. 即 $T(i) \oplus T(|w| - 1 - i)$ 为定值 $x \oplus y = T(|w| - 1)$ 。

假设 $|w|$ 不为 2 的非负整数幂, 故 $|w| - 1 > 0$. 接着, 将 $|w| - 1$ 二进制拆分为 $(a_0, a_1, \dots, a_n = 1)$, 满足 $|w| - 1 = \sum_{i=0}^n a_i 2^i$, $a_i \in \{0, 1\}$. 由于 $|w|$ 不为 2 的非负整数幂, 故 $a_0, a_1, \dots, a_n$ 不全为 1. 所以, 一定可以找到一个最大的 k 满足 $a_k = 0$. 由于 k 是最大的一个, 故 $a_{k+1} = 1$. 考虑 $T(2^k) \oplus T(|w| - 1 - 2^k)$ 是否等于 $T(|w| - 1)$. 下面给出 $|w| - 1$ 和 $|w| - 1 - 2^k$ 的二进制拆分:

$$|w| - 1 : (a_0, a_1, \dots, a_k = 0, a_{k+1} = 1, \dots, a_n = 1)$$

$$|w| - 1 - 2^k : (a_0, a_1, \dots, a'_k = 1, a'_{k+1} = 0, \dots, a_n = 1)$$

发现两者除了 $k, k+1$ 位不同外, 剩余位完全相同。于是, 可以得到 $T(|w| - 1) = T(|w| - 1 - 2^k)$, 而 $T(2^k) \oplus T(|w| - 1 - 2^k) = 1 \oplus T(|w| - 1) \neq T(|w| - 1)$, 矛盾。 $\square$

所以, 对于 $w \in \{0, 1\}^*$ 的两个 TM 拆分, 其中间重叠部分的长度为 2 的非负整数幂。

至此, 利用这个性质, 已经能够分析出 $w \in \{0, 1\}^*$ 的 TM 拆分个数不超过 3. 但这个上界实际上为 2。

定理 4.2.7. 对于 $w \in \{0, 1\}^*$, 其 TM 拆分个数不超过 2。

证明. 假设 w 有 3 个 TM 拆分, 此时不妨设 $w = uvxy$, $u, v, x, y \in \{0, 1\}^*$, w 的 3 个 TM 拆分分别为 (u, vxy) 、 (uv, xy) 和 (uvx, y) . 则由引理 4.2.6 可知, $|v|$ 、 $|x|$ 和 $|vx| = |v| + |x|$ 均为 2 的非负整数幂, 故 $|v| = |x| = 2^k, k \in \mathbb{N}$. 由于 vx 为准 TM 前缀串, 应用定理 4.2.3 可得, $v(0) \oplus x(0) = T(0) \oplus T(|v|) = 1$, 即 $v(0) \neq x(0)$. 同理, 由 vxy, xy 为准 TM 前缀串可以分别得到 $v(0) \neq y(0)$ 、 $x(0) \neq y(0)$. 故 $v(0)$ 、 $x(0)$ 和 $y(0)$ 两两不同, 由抽屉原理可知, 这是不可能的。 $\square$

4.2.3 Thue-Morse 子串有 2 个 TM 拆分结构

定理 4.2.8. 对于 $x, y, z \in \{0, 1\}^*$, (x, yz) 、 (xy, z) 均为 $w = xyz$ 的 TM 拆分当且仅当 $|x|, |z| \leq |y|$, 且 $(xy)^R$ 和 yz 均为准 TM 前缀串。

证明. 分别考虑必要性和充分性。

必要性 一方面, 若 (x, yz) 、 (xy, z) 均为 w 的 TM 拆分。显然 $(xy)^R$ 、 yz 均为准 TM 前缀串, 且 $|y|$ 为 2 的非负整数幂, 设 $|y| = 2^k, k \in \mathbb{N}$, 若 $|z| > |y| = 2^k$, 则和定理 4.2.7 的证明类似, 有 $y(0) \neq z(0), y(0) \neq z(2^k), z(0) \neq z(2^k)$, 导出矛盾, 故 $|z| \leq |y|$, 同理也可以得到 $|x| \leq |y|$ 。

充分性 另一方面, 若 $|x|, |z| \leq |y|$, 且 $(xy)^R$ 和 yz 均为准 TM 前缀串。可得 $|y|$ 为 2 的非负整数幂, 设 $|y| = 2^k, k \in \mathbb{N}$, 对于 $i = 0, 1, \dots, |z| - 1$, 应用定理 4.2.3 可得, $y(i) \oplus z(i) = T(i) \oplus T(2^k + i) = 1$, 故 $z = \overline{y(0, |z| - 1)}$ 。因为 y 为准 TM 前缀串, 故 $y(0, |z| - 1)$ 、 $z = \overline{y(0, |z| - 1)}$ 均为准 TM 前缀串。同理, 也可以得到 x^R 为准 TM 前缀串, 故 (x, yz) 、 (xy, z) 均为 w 的 TM 拆分。□

4.2.4 Thue-Morse 子串结构的应用

例题 4.1 (基础 01? 练习题⁴)。对于 $S \in \{0, 1\}^*$, 记 $g(S)$ 表示 S 中有多少子串同时也是 Thue-Morse 串的子串 (子串相同但出现位置不同算多个)。对于 $S \in \{0, 1, ?\}^*$, 记 $h(S)$ 表示将 S 中所有 ? 替换为 0 或 1 后得到的串 S' 的 $g(S')$ 之和。给定 $S \in \{0, 1, ?\}^*$, m 次询问, 每次查询给出 l, r , 求 $h(S_l S_{l+1} \dots S_r) \bmod 998244353$ 。数据范围: $|S| \leq 5 \times 10^4, m \leq 2 \times 10^5$ 。

解法 首先, 考虑 $l = 0, r = |S| - 1$ 的情况。对于一个 Thue-Morse 串的子串, 为了仅计算一次贡献, 可以计算长度为 1 的子串个数 + 所有子串 TM 拆分数之和 - TM 拆分数为 2 的子串个数。

计算所有子串 TM 拆分数之和, 需要枚举拆分数在 S 中对应的位置, 向左向右匹配 T 或 $\overline{T}$, 计算合法左右端点个数。匹配部分只需要使用倍增维护 (由定义 2.4 可得, $TM_{2^{n+1}} = TM_{2^n} \overline{TM_{2^n}}, \overline{TM_{2^{n+1}}} = \overline{TM_{2^n}} TM_{2^n}$)。

计算 TM 拆分数为 2 的子串个数, 需要枚举两个 TM 拆分重叠部分的长度 2^k , 再枚举该段在 S 中的位置, 向左向右匹配至多 2^k 个位置, 计算左右端点的个数。

最后, 加上区间询问, 只需要使用扫描线、历史和线段树就能够维护了, 具体内容参见笔者的解题报告。

⁴[2024-2025 集训队互测 Round 1 T2] 基础 01? 练习题, 这是笔者根据上述定理设计的题目。

4.3 Thue-Morse 串的重叠性质及其应用

4.3.1 无重叠串的定义

定义 4.3. 对于 $w \in \Sigma^\infty$, 称 w 是无重叠的, 当且仅当 w 中不存在两个相交的相同子串。

然而, 有另一个更为形式化的定义方式。

定义 4.4. 对于 $w \in \Sigma^\infty$, 称 w 是无重叠的当且仅当 $\forall a \in \Sigma, u \in \Sigma^* \cup \{\varepsilon\}$, $auaua$ 不为 w 的子串。

这两种定义方式是等价的:

- 一方面, 若有交的两个子串, 交的大小大于 1, 则可以缩减为 1, 如图 2,

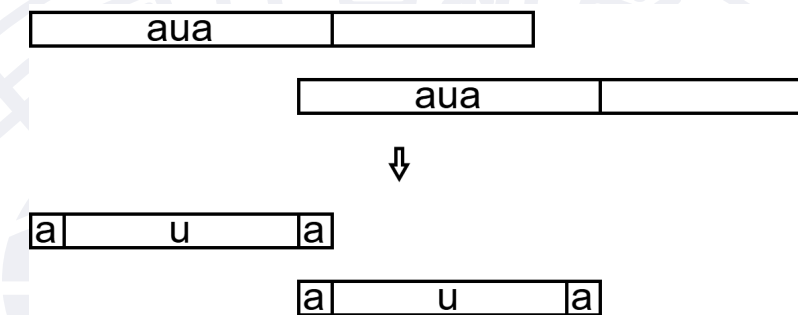


图 2: 重叠子串缩减过程示意图

故有重叠的字符串一定可以找到形如 $auaua$ 的子串;

- 另一方面, $auaua$ 中包含两个重叠的 aua 子串, 无重叠串显然不能包含 $auaua$ 。

4.3.2 Thue-Morse 串的重叠性质

定理 4.3.1. T 是无重叠的, 即 Thue-Morse 串是无重叠的。

这个定理的证明并不是那么容易, Christopher 给出了一种证明方式⁵。但笔者提供了另外一种截然不同的使用代数分析的证明方式。

证明. 假设 T 存在一个重叠子串 $auaua(a \in \{0, 1\}, u \in \{0, 1\}^* \cup \{\varepsilon\})$ 。

由定理 4.2.5 可得, $auaua$ 至少有一个 TM 拆分 (x, y) 。由定理 4.2.2 可知, 若 $|x| > |y|$, 则翻转该重叠子串得到 $au^R au^R a$, 也是 T 的一个重叠子串, 找到该子串 (y^R, x^R) 这组 TM 拆分,

⁵Christopher Williamson, [An Overview of the Thue-Morse Sequence](#), 2012

就转化为了 $|x| < |y|$ 的情况。所以接下来只需考虑 $|x| < |y|$ 的情况（由于 $|auaua|$ 为奇数，不存在 $|x| = |y|$ 的情况）。

设拆分点将 u 拆成两个字符串： $u = vw$ ，其中 $v, w \in \Sigma^* \cup \{\varepsilon\}$ 。则子串 $auaua = avwvwa$ 的 TM 拆分即为 $(av, wvwa)$ 。注意，此时 v, w 可能为空字符串。

由定理 4.2.1 可知，若 $wvwa$ 是 $\bar{T}$ 的前缀，则将子串 $auaua$ 反转得到 $\overline{auaua}$ ，也是 T 的重叠子串。此时的 TM 拆分是 $(\overline{av}, \overline{wvwa})$ ，其中 $\overline{wvwa}$ 是 T 的前缀，就转化为了 $wvwa$ 是 T 的前缀的情况。所以接下来只需考虑 $wvwa$ 是 T 的前缀的情况，如图 3。

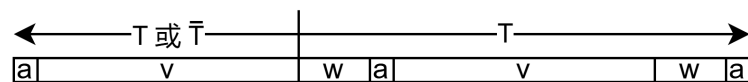


图 3: $auaua$ 的 TM 拆分 $(av, wvwa)$ 示意图

设 $n = |wv| = |u| + 1$ ，则 $wa = T(0, |wa| - 1) = T(n, n + |wa| - 1)$ ，故对于 $n \in \mathbb{N}^*$ ，设 $f(n) = \text{LCP}(T, T(n : \infty)) \geq |wa|$ ，其中 $\text{LCP}(u, v)$ 表示 u, v 的最长公共前缀的长度。

同时，由 TM 拆分的定义可知， $(av)^R$ 是准 TM 前缀串。对于 $n \in \mathbb{N}^*$ ，设 $g(n)$ 表示最大的正整数 k 满足 $T(n - k : n - 1)^R$ 为准 TM 前缀串。由于 $T(n - |av|, n - 1)^R = (av)^R$ 是准 TM 前缀串，所以 $g(n) \geq |av|$ 。

所以， $n = |u| + 1 \geq 1$ ，满足 $f(n) + g(n) \geq |wa| + |av| = n + 1$ 即 $f(n) + g(n) > n$ 。接下来逐步证明： $\forall n \in \mathbb{N}^*$ ， $f(n) + g(n) \leq n$ ，从而导出矛盾。

引理 4.3.2. 当且仅当 n 为 2 的非负整数幂时， $g(n) = n$ 。

证明. 一方面，由引理 4.2.6 可得，若 $g(n) = n$ ，则 n 为 2 的非负整数幂。

另一方面，若 n 为 2 的非负整数幂，则对于 $i = 0, 1, \dots, n-1$ ，都有 $T(n-1) \oplus T(n-1-i) = T(i)$ 由定理 4.2.3 可得， $g(n) = n$ 。□

引理 4.3.3. $\forall n \in \mathbb{N}^*$ ， $g(n) \neq n-1$ 。

证明. 当 $i = n-1$ 时，一定满足 $T(n-1) \oplus T(n-1-i) = T(i)$ ，故 $g(n) \neq n-1$ 。□

引理 4.3.4. 对于 $n \in \mathbb{N}^*$ ， $f(2n) = 2f(n)$ 。

证明. 一方面，由于 $\text{LCP}(T, T(n : \infty)) = f(n)$ ，故 $T(f(n)) \neq T(n + f(n))$ 。则 $T(2f(n)) = T(f(n)) \neq T(n + f(n)) = T(2n + 2f(n))$ ，即 $\text{LCP}(T, T(2n : \infty)) \leq 2f(n)$ ，也即 $f(2n) \leq 2f(n)$ 。

另一方面， $\forall i = 0, 1, \dots, f(n) - 1$ ，已知 $T(i) = T(n + i)$ ，故 $T(2i) = T(2n + 2i)$ ， $T(2i + 1) = T(2n + 2i + 1)$ 。即 $\forall i = 0, 1, \dots, 2f(n) - 1$ ， $T(i) = T(2n + i)$ ，也即 $f(2n) \geq 2f(n)$ 。□

引理 4.3.5. 对于 $n \in \mathbb{N}^*$ ， $g(2n) = 2g(n)$ 。

证明. 考虑使用定理 4.2.3、4.2.4 和定义 2.2 辅助证明。

一方面，由于 $T(n - g(n) : n - 1)^R$ 为准 TM 前缀串，则 $\forall i = 0, 1, \dots, g(n) - 1$ ， $T(n-1) \oplus T(n-1-i) = T(i)$ 。所以 $T(2n-1) \oplus T(2n-1-2i) = T(2i)$ ， $T(2n-1) \oplus T(2n-1-(2i+1)) = T(2i+1)$ 。即 $\forall i = 0, 1, \dots, 2g(n) - 1$ ， $T(2n-1) \oplus T(2n-1-i) = T(i)$ ，也即 $g(2n) \geq 2g(n)$ 。

另一方面，若 $g(n) = n$ ，则 $g(2n) \leq 2n = 2g(n)$ ；若 $g(n) \leq n-1$ ，则 $T(n-1) \oplus T(n-1-g(n)) \neq T(g(n))$ 。所以 $T(2n-1) \oplus T(2n-1-2g(n)) \neq T(2g(n))$ ，使用定理 4.2.4 的逆否命题可得， $g(2n) \leq 2g(n)$ 。□

引理 4.3.6. $\forall n \in \mathbb{N}$ ， $f(2n+1) \leq 2$ 。

证明. 假设 $f(2n+1) > 2$ 。则 $T(2n+2) = T(1) = 1$ 且 $T(2n+3) = T(2) = 1$ 。由定义 2.2 可得， $T(2n+2) \neq T(2n+3)$ ，矛盾。□

定理 4.3.7. $\forall n \in \mathbb{N}^*$ ， $f(n) + g(n) \leq n$ 。

证明. 分为如下几种情况讨论：

$n = 1, 2$ $f(n) = 0, g(n) = n$ 。

$n \geq 3$ 且 n 为奇数 显然 n 不是 2 的非负整数幂。由于 $g(n) \leq n$ ，由引理 4.3.2、4.3.3 可得， $g(n) \neq n, n-1$ ，则 $g(n) \leq n-2$ 。再由引理 4.3.6 得到， $f(n) + g(n) \leq 2 + (n-2) = n$ 。

$n \geq 3$ 且 n 为偶数 $f(n) + g(n) = 2f(\frac{n}{2}) + 2g(\frac{n}{2}) \leq 2 \times \frac{n}{2} = n$ 。□

至此，得到了 $\forall n \in \mathbb{N}^*$ ， $f(n) + g(n) \leq n$ 后，回到定理 4.3.1 的证明。发现原假设不成立，即 T 是无重叠的，原定理得证。□

4.3.3 Thue-Morse 串的非重叠性质应用

定义 4.5 (无立方串). 对于 $w \in \Sigma^\infty$ ，称 w 是无立方的，当且仅当 w 中不存在连续的三个相同的子串。形式化地， $\forall u \in \Sigma^*$ ， uuu 不为 w 的子串。

推论 4.3.8. T 是无立方的，即 Thue-Morse 串是无立方的。

证明. 显然，立方子串一定也是重叠子串，而 T 是无重叠的，可以得出 T 也是无立方的。□

定义 4.6 (无平方串). 对于 $w \in \Sigma^\infty$ ，称 w 是无平方的，当且仅当 w 中不存在连续的两个相同的子串。形式化地， $\forall u \in \Sigma^*$ ， uu 不为 w 的子串。

根据无重叠、无平方、无立方的定义，可以发现无平方的限制包含无重叠的限制、无重叠的限制包含无立方的限制。

当 $|\Sigma| \leq 2$ 时，容易发现是不存在无限的无平方串的。

而当 $|\Sigma| = 3$ 时，想要构造出一个无限的无平方串，似乎并不那么容易。但利用 Thue-Morse 串的无重叠性，就能够轻松构造出无限的无平方串了：

- 设 $X = \{n | T(n) = 0\} = \{x_0 < x_1 < x_2 < \dots\}$ 。
- 考虑 $\Sigma = \{0, 1, 2\}$ 时的一个无限字符串 $c \in \Sigma^\infty$ ，满足 $c(n) = x_{n+1} - x_n - 1$ 。
- 由于 T 是无重叠的，故 T 中不存在连续的 3 个 1，即 $c(n) = x_{n+1} - x_n - 1 \leq 2$ ，确保 c 的字符集大小为 3。
- 下面给出 c 的前若干项⁶：

210201210120210201202101...

另外， c 实际上是态射变换 $2 \rightarrow 210, 1 \rightarrow 20, 0 \rightarrow 1$ 的不动点。这也是 c 被称为“Ternary Thue-Morse sequence”的原因。

使用不同的态射变换可以得到很多与 Thue-Morse 序列性质相似的序列（或称为无限串），例如态射变换 $0 \rightarrow 01, 1 \rightarrow 0$ 可以得到斐波那契无限串。这些序列都具有很多优美的性质，有兴趣的读者可以自行探究，此处并不提及。

定理 4.3.9. c 是无平方的。

证明. 假设 c 有平方子串。设 c 中包含 uu 子串，其中 $u \in \{0, 1, 2\}^*$ 。则可知， T 中包含如下子串：

$01^{u(0)}01^{u(1)}0 \dots 01^{u(|u|-1)}01^{u(0)}01^{u(1)}0 \dots 01^{u(|u|-1)}0$

其中 1^k 表示 k 个 1 拼接得到的字符串。特别地， $1^0 = \varepsilon$ 。

观察这个子串，容易发现它是一个重叠子串，与定理 4.3.1 矛盾。

□

例题 4.2 (红⁷). 在 $\Sigma = \{0, 1, 2\}$ 上构造一个 $w \in \Sigma^*$ ，满足 $|w| = n$ 且 w 包含 m 个平方子串（子串相同但出现位置不同算多个），或判断无解。数据范围： $0 \leq n, m \leq 10^5$ 。

⁶OEIS A036577, <https://oeis.org/A036577>

⁷[2024 牛客暑期多校训练营 5 F] 红, <https://ac.nowcoder.com/acm/contest/81600/F>

解法 取出无平方串 c 的前 k 项，扩展为长度为 $l_0, l_1, \dots, l_{k-1}$ 的连续段，即构造字符串 $w = c(0)^{l_0}c(1)^{l_1} \dots c(k-1)^{l_{k-1}}$ 。则 w 的平方子串个数是 $\sum f_i$ ，其中 f_i 表示长度为 i 的连续段的平方子串个数。

考虑如何求出包含 m 个平方子串最短的字符串长度。可以使用背包解决，由于 f_n 是 $O(n^2)$ 级别的，故复杂度是 $O(m\sqrt{m})$ 。

最后，在包含 m 个平方子串的最短字符串末尾不断插入一个 $l_k = 1$ 的连续段，直到字符串总长达到 n ，就能够得到符合条件的构造。

另外，这样的构造方法，构造不出来的 (n, m) 是否一定无解？实际上是的，证明参见此题的题解⁸。

4.4 Thue-Morse 串的最短 n -特征子串（前缀）

对于一些问题（如例题 4.1），研究对象仅限于 $w \in \Sigma^\infty$ 的所有长度不超过 n 的子串。当 w 的结构具有一定性质时，仅需保留一个 w 的子串 u ，将 u 当作 w 处理即可得到相同的结果。

定义 4.7 (n -特征子串). 对于 $w \in \Sigma^\infty$ ， u 为 w 的非空子串，若所有 w 的长度不超过 n 的子串都是 u 的子串，则称 u 为 w 的 n -特征子串。

定义 4.8 (最短 n -特征子串). 对于 $w \in \Sigma^\infty$ ，若 u 是 w 的 n -特征子串，且不存在 $v \in \Sigma^*$, $|v| < |u|$ ， v 也是 w 的 n -特征子串，则称 u 为 w 的最短 n -特征子串。

在探究 Thue-Morse 串的最短 n -特征子串之前，先考虑更简单的问题：Thue-Morse 串的最短 n -特征前缀。

4.4.1 Thue-Morse 串的最短 n -特征前缀

定义 4.9 (n -特征前缀). 对于 $w \in \Sigma^\infty$ ， u 为 w 的非空前缀，若所有 w 的长度不超过 n 的子串都是 u 的子串，则称 u 为 w 的 n -特征前缀。

定义 4.10 (最短 n -特征前缀). 对于 $w \in \Sigma^\infty$ ，若 u 是 w 的 n -特征前缀，且不存在 $v \in \Sigma^*$, $|v| < |u|$ ， v 也是 w 的 n -特征前缀，则称 u 为 w 的最短 n -特征前缀。

由定义容易发现，Thue-Morse 串的最短 n -特征子串可能有多个，但其最短 n -特征前缀是唯一的。故对于 $n \in \mathbb{N}^*$ ，只需求出 $\text{pre}(n)$ 表示 Thue-Morse 串的最短 n -特征前缀的长度。

显然，对于 Thue-Morse 串，其 n -特征前缀，仅需考虑长度恰为 n 的 Thue-Morse 子串。

⁸2024 牛客暑期多校资料汇总帖, <https://ac.nowcoder.com/discuss/1295959>

$\text{pre}(n)$ 的前若干项如下表所示。

n	1	2	3	4	5	6	7	8	9	10
$\text{pre}(n)$	2	7	8	15	16	29	30	31	32	57

例子 对于 $n = 3$ 的情况, T 共有 6 个本质不同的长度为 3 的子串: 001、010、011、100、101 和 110, 它们都是 $T(0 : 7) = 01101001$ 的子串, 但 001 不是 $T(0 : 6) = 0110100$ 的子串。故 $h(3) = 8$ 。

从定理 4.2.5 的充分性证明中可以得到, $\text{pre}(n)$ 是 $O(n)$ 量级的。但实际上, 有精确计算 $\text{pre}(n)$ 的办法。

定理 4.4.1.

$$\text{pre}(n) = \begin{cases} 2 & n = 1 \\ 7 & n = 2 \\ 3 \times 2^{\lceil \log_2(n-1) \rceil} + n - 1 & n \geq 3 \end{cases}$$

证明. $n = 1, 2$ 的情况是简单的, 接下来考虑 $n \geq 3$ 的情况。

设 $k = \lceil \log_2(n-1) \rceil \geq 1$, 则 $\text{pre}(n) = 3 \times 2^k + n - 1$ 。

一方面, 考虑任意长度为 n 的 Thue-Morse 子串 w , 找到 w 的一个 TM 拆分 (u, v) 。将 $T(0 : 4 \times 2^k - 1)$ 用 $s = T(0 : 2^{k-1} - 1)$ 表示, 如图 4。

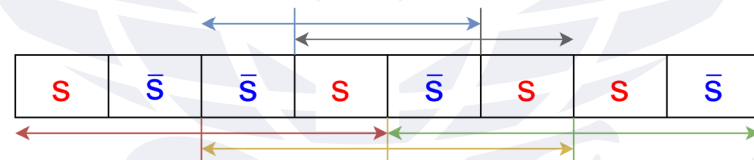


图 4: $T(0 : 4 \times 2^k - 1) = s \bar{s} s \bar{s} s s \bar{s} \bar{s}$

由于 $|u| \leq n - 1 \leq 2^k = 2|s|$, 同理 $|v| \leq 2|s|$, 故图 4 中的 5 个位置, 一定有一个向左匹配 u^R 、向右匹配 v 的 (当 u^R 是 $(s\bar{s})^R$ 的前缀、 v 是 $s\bar{s}$ 的前缀时, 由于 $|u| + |v| = n \leq 2^k + 1$, 故 u, v 一定有一个长度不超过 $|s| = 2^{k-1}$, 所以蓝色和灰色两个位置中一定有一个能够匹配)。考虑最坏情况, 即 (u, v) 匹配在绿色位置的情况, 则 $\text{pre}(n) \leq 3 \times 2^k + |v| \leq 3 \times 2^k + n - 1$ 。

另一方面, 考虑上述不等式取等时的条件, 即当 $w = T(2^{k-1} - 1)T(0 : n - 2)$ 时, 尝试求出其首次出现的末尾下标的下限。在此, 引入定理 4.3.1 证明中的 $f(n)$, 为方便读者阅读接下来的证明过程, 给出 $f(n)$ 的前若干项。

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f(n)$	0	0	2	0	1	4	0	0	1	2	0	8	0	0	2

设 $w = T(p - 1 : p + n - 2)$, 则 $T(p - 1) = T(2^{k-1} - 1)$ 且 $T(p : p + n - 2) = T(0 : n - 2)$, 故 $f(p) \geq n - 1 > 2^{k-1}$ 。设 $p = (2a + 1) \times 2^b$, 其中 $a, b \in \mathbb{N}$ 。由引理 4.3.4、4.3.6 可知, $f(p) = 2^b f(2a + 1) > 2^{k-1}$, 故 $2^{b+1} > 2^{k-1}$ 即 $b \geq k - 1$ 。接下来讨论 $f(2a + 1)$ 和 b 的取值。

情况一: $f(2a + 1) = 0$ 则 $f(p) = 0$, 不满足。

情况二: $f(2a + 1) = 1$ 由上表知, $2a + 1 \geq 5$, $f(p) = 2^b > 2^{k-1}$ 即 $b \geq k$ 。故 $p = (2a + 1) \times 2^b \geq 5 \times 2^k$ 。

情况三: $f(2a + 1) = 2, b \geq k$ 由上表知, $2a + 1 \geq 3$, 故 $p = (2a + 1) \times 2^b \geq 3 \times 2^k$ 。

情况四: $f(2a + 1) = 2, b = k - 1$ 若 $2a + 1 = 3$, 则 $T(p - 1) = T(3 \times 2^{k-1} - 1) = \overline{T(2^{k-1} - 1)}$, 不满足。故由上表得到 $2a + 1 \geq 15$, 则 $p = (2a + 1) \times 2^b \geq 15 \times 2^{k-1}$ 。

所以 $p \geq 3 \times 2^k$ 。此时 $w = T(p - 1 : p + n - 2)$ 的右端点 $p + n - 2 \geq 3 \times 2^k + n - 2$, 即 $\text{pre}(n) \geq 3 \times 2^k + n - 1$ 。

□

4.4.2 Thue-Morse 串的最短 n -特征子串

和上一小节类似地, 对于 $n \in \mathbb{N}^*$, 设 $\text{sub}(n)$ 表示 Thue-Morse 串的最短 n -特征子串的长度。

$\text{sub}(n)$ 的前若干项如下表所示。

n	1	2	3	4	5	6	7	8	9	10
$\text{sub}(n)$	2	5	8	14	16	26	28	30	32	50

例子 10011 是 Thue-Morse 串的一个 2-特征子串。容易证明, 不存在长度更短的 2-特征子串, 故 $\text{sub}(2) = 5$ 。

在定理 4.4.1 的证明中, 若把图 4 的红色部分的向左匹配的长度缩减为 $n - 1$, 就能对于 $n \in \mathbb{N}^*$ 且 $n \geq 3$ 的情况, 构造出 Thue-Morse 串的长度为 $2^{\lceil \log_2(n-1) \rceil + 1} + 2(n - 1)$ 的 n -特征子串。即 $\forall n \in \mathbb{N}^*$ 且 $n \geq 3$, $\text{sub}(n) \leq 2^{\lceil \log_2(n-1) \rceil + 1} + 2(n - 1)$ 。

对比上表, 发现对于 $n = 3, 4, \dots, 10$, 均满足 $\text{sub}(n) = 2^{\lceil \log_2(n-1) \rceil + 1} + 2(n - 1)$ 。

定理 4.4.2.

$$\text{sub}(n) = \begin{cases} 2 & n = 1 \\ 5 & n = 2 \\ 2^{\lceil \log_2(n-1) \rceil + 1} + 2(n - 1) & n \geq 3 \end{cases}$$

备注 对于 $n \in \mathbb{N}^*$ 且 $n \geq 3$ 时 $\text{sub}(n) \geq 2^{\lceil \log_2(n-1) \rceil + 1} + 2(n-1)$ 的证明, 由于其较为复杂, 此处略去, 有兴趣的读者可以自行证明。

5 Thue-Morse 生成函数

5.1 生成函数的引入

首先, 分别列出 $T_n = 0$ 和 $T_n = 1$ 的集合:

$$A = \{n \in \mathbb{N} | T_n = 0\} = \{0, 3, 5, 6, \dots\}$$

$$B = \{n \in \mathbb{N} | T_n = 1\} = \{1, 2, 4, 7, \dots\}$$

接着, 分别写出 A, B 的生成函数 $A(x), B(x)$:

$$A(x) = \sum_{a \in A} x^a = 1 + x^3 + x^5 + x^6 + \dots$$

$$B(x) = \sum_{b \in B} x^b = x + x^2 + x^4 + x^7 + \dots$$

非常容易地, 可以得到:

$$A(x) + B(x) = \frac{1}{1-x} \quad (1)$$

同时, 由定义 2.2 可以得到:

$$A(x) - B(x) = (1-x)(A(x^2) - B(x^2)) \quad (2)$$

更进一步地, 依次将 (2) 中的 x 替换为 $x^2, x^4, x^8, \dots$ 可以得到:

$$A(x) - B(x) = (1-x)(A(x^2) - B(x^2))$$

$$A(x^2) - B(x^2) = (1-x^2)(A(x^4) - B(x^4))$$

$$A(x^4) - B(x^4) = (1-x^4)(A(x^8) - B(x^8))$$

$\vdots$

$$A(x^{2^n}) - B(x^{2^n}) = (1-x^{2^n})(A(x^{2^{n+1}}) - B(x^{2^{n+1}}))$$

于是, 有:

$$A(x) - B(x) = \prod_{k=0}^{n-1} (1-x^{2^k})(A(x^{2^n}) - B(x^{2^n}))$$

由于 $[x^0](A(x) - B(x)) = 1$, 所以, 当 n 趋于 ∞ 时, 可以得到:

$$A(x) - B(x) = \prod_{k=0}^{\infty} (1 - x^{2^k}) \quad (3)$$

另外, 从 $A(x), B(x)$ 的定义, 可以得到:

$$A(x) - B(x) = \sum_{n=0}^{\infty} (-1)^{T(n)} x^n$$

对于 (3) 的推导, 也可以从定义 2.1 直接出发, (3) 中的 k 即为枚举的 n 的第 k 位, 若第 k 位为 0, 则对 n 的贡献为 0, 对 $(-1)^{T(n)}$ 的贡献为 1; 否则对 n 的贡献为 2^k , 对 $(-1)^{T(n)}$ 的贡献则为 -1 。而这也正是 (3) 的直观理解。

5.2 等幂和问题的另一种证明方式

考虑前文所说的集合 X_k, Y_k , 容易发现, 两者 m 次方和的差为:

$$\begin{aligned} \sum_{x \in X_k} x^m - \sum_{y \in Y_k} y^m &= \sum_{n=0}^{2^{k+1}-1} (-1)^{T(n)} n^m \\ &= \sum_{n=0}^{2^{k+1}-1} (-1)^{T(n)} \left[\frac{x^m}{m!} \right] e^{nx} \\ &= \left[\frac{x^m}{m!} \right] \sum_{n=0}^{2^{k+1}-1} (-1)^{T(n)} e^{nx} \\ &= \left[\frac{x^m}{m!} \right] \prod_{n=0}^k (1 - e^{2^n x}) \end{aligned}$$

容易发现, $[x^0](1 - e^{2^n x}) = 0$, 而共有 $k+1$ 个常数项为 0 的多项式相乘, 故当 $m \leq k$ 时, 最终结果为 0。

这和上文推出的结论完全一致。

5.3 一个有趣的自然数划分问题

5.3.1 问题描述

将自然数集合 $\mathbb{N}$ 划分为两个集合 A, B , 使得对于任意 $n \in \mathbb{N}$, 将 n 表示成 A 中不同的两个数之和的方案数等于其在 B 中的方案数。

5.3.2 A, B 是一组解

由 (1) 和 (2) 可以得到:

$$A(x) - B(x) = \frac{A(x^2) - B(x^2)}{A(x) + B(x)}$$

整理一下，可以得到：

$$A(x)^2 - A(x^2) = B(x)^2 - B(x^2) \quad (4)$$

考虑一下 $[x^n]A(x)^2 - A(x^2)$ (resp. $[x^n]B(x)^2 - B(x^2)$) 的实际含义，即为将 n 表示成 A (resp. B) 中不同的两个数之和的方案数。

5.3.3 A, B 是唯一解

设存在另一组划分方案集合 C, D ，不妨设 $0 \in C$ ，同理设出它们的生成函数：

$$C(x) = \sum_{c \in C} x^c \quad D(x) = \sum_{d \in D} x^d$$

则也可以得到：

$$\begin{aligned} C(x) + D(x) &= \frac{1}{1-x} \\ C(x)^2 - C(x^2) &= D(x)^2 - D(x^2) \end{aligned}$$

容易发现，这和 (1)、(2) 是等价的，且 $[x^0](C(x) - D(x)) = 1$ 。故和推导 $A(x) - B(x)$ 类似地，可以得到：

$$C(x) - D(x) = \prod_{k=0}^{\infty} (1 - x^{2^k})$$

故 $C(x) = A(x), D(x) = B(x)$ ，说明该问题仅有唯一一组解 (A, B) 。

6 其他性质及应用

6.1 Shevelev 的猜想

定理 6.1.1 (Shevelev 的猜想⁹)。记 $\{u(n)\}_{n=0}^{\infty} = \{(-1)^{T(n)}\}_{n=0}^{\infty}$ ，令 a 为正整数。令 $B_a = \{\ell_0 < \ell_1 < \ell_2 < \dots\}$ ， $C_a = \{m_0 < m_1 < m_2 < \dots\}$ ，满足 $u(\ell + a) = -u(\ell)$ ， $u(m + a) = u(m)$ 。

记 $\{\beta_a(n)\}_{n=0}^{\infty}, \{\gamma_a(n)\}_{n=0}^{\infty}$ 为 ± 1 序列，满足：

$$\beta_a(n) = u(\ell_n), \gamma_a(n) = u(m_n)$$

⁹Vladimir Shevelev, Equations of the Form $t(x+a) = t(x)$ and $t(x+a) = 1 - t(x)$ for Thue-morse Sequence, 2012

那么 $\beta_a(n), \gamma_a(n)$ 具有最小的长度为 $k = 2 \text{lowbit}(a)$ 的循环节，且 $\beta_a(n) = -\gamma_a(n)$ 。

更进一步地，若 $u(a) = +1$ ，则 $\{\gamma_a(n)\}_{n=0}^{k-1} = \{u(n)\}_{n=0}^{k-1}$ ，否则 $u(a) = -1$ ，则 $\{\gamma_a(n)\}_{n=0}^{k-1} = \{-u(n)\}_{n=0}^{k-1}$ 。

例子 对于 $a = 6$ 的情况， $u(a) = +1$ ，有：

$$B_a = \{4, 5, 10, 11, 14, 15, 20, 21, 28, 29, \dots\}$$

$$C_a = \{0, 1, 2, 3, 6, 7, 8, 9, 12, 13, \dots\}$$

$$\beta_a = \{-1, +1, +1, -1, -1, +1, +1, -1, \dots\}$$

$$\gamma_a = \{+1, -1, -1, +1, +1, -1, -1, +1, \dots\}$$

证明。先考虑证明定理中 γ_a 的部分，接下来对 a 的奇偶性分类讨论。

a 为奇数 即需证明：当 $u(a) = +1$ 时， $\gamma_a = \{+1, -1, +1, -1, \dots\}$ ； $u(a) = -1$ 时， $\gamma_a = \{-1, +1, -1, +1, \dots\}$ 。

考虑序列 $\{w_a(n)\}_{n=0}^{\infty}$ ，其中 $w_a(n) = u(a+n) + u(n)$ 。显然，当 $n \in C_a$ 时， $w_a(n) = 2u(n) = \pm 2$ ，否则 $w_a(n) = 0$ 。换句话说，序列 w_a 是序列 $2\gamma_a$ 插入了一些 0 得到的。

那么 $\gamma_a = \{+1, -1, +1, -1, \dots\}$ 等价于 w_a 的前缀和仅有 $\{+2, 0\}$ 两种取值；

而 $\gamma_a = \{-1, +1, -1, +1, \dots\}$ 等价于 w_a 的前缀和仅有 $\{-2, 0\}$ 两种取值。

因此，设：

$$S_a(n) = \sum_{i=0}^{n-1} w_a(i) = \sum_{i=0}^{n-1} u(a+i) + \sum_{i=0}^{n-1} u(i)$$

由定义 2.2 可得， $u(2n) + u(2n+1) = 0$ ，故对 n 的奇偶性进行分类讨论来化简 $S_a(n)$ 。

n 为偶数：

$$S_a(n) = \sum_{i=0}^{n-1} u(a+i) + \sum_{i=0}^{n-1} u(i) = u(a) + u(a+n-1) = u(a) \pm 1$$

n 为奇数：

$$S_a(n) = \sum_{i=0}^{n-1} u(a+i) + \sum_{i=0}^{n-1} u(i) = u(a) + u(n-1) = u(a) \pm 1$$

所以，当 $u(a) = +1$ 时， $S_a(n) \in \{+2, 0\}$ ；当 $u(a) = -1$ 时， $S_a(n) \in \{-2, 0\}$ 。

a 为偶数 容易发现

$$2n \in C_{2a} \iff u(2n) = u(2n+2a) \iff u(n) = u(n+a) \iff n \in C_a$$

$$2n+1 \in C_{2a} \iff u(2n+1) = u(2n+1+2a) \iff -u(n) = -u(n+a) \iff n \in C_a$$

因此, 若 $C_a = \{\ell_0, \ell_1, \ell_2, \dots\}$, 则 $C_{2a} = \{2\ell_0, 2\ell_0+1, 2\ell_1, 2\ell_1+1, 2\ell_2, 2\ell_2+1, \dots\}$, 所以有:

$$\gamma_{2a}(2n) = \gamma_a(n)$$

$$\gamma_{2a}(2n+1) = -\gamma_a(n)$$

因此, 若 γ_a 满足上述性质, 则 γ_{2a} 也同样满足。

最后, 将 $u(n+a) + u(n)$ 替换为 $u(n+a) - u(n)$ 后, 采取上述相同的步骤就能得出 $\beta_a = -\gamma_a$ 。□

备注 笔者在阅读了 Allouche 的证明方式¹⁰后, 认为其利用生成函数考虑前缀和的方式非常巧妙。受其启发, 笔者得出了上述更为直接的证明方式。

6.2 哈希冲突

在 OI 中, 传统的自然溢出的字符串哈希方式, 由于其不需要进行取模运算, 故拥有简短的实现和相比于大质数取模较快的运行效率, 是广为人知的一种哈希方式。

但是, 这种哈希方式, 即模数 $p = 2^k$, 底数为 b , 对于字符串 $w \in \Sigma^*$ 的哈希结果为:

$$\sum_{i=0}^{|w|-1} w(i)b^i \mod p$$

的哈希方式, 是可以利用 Thue-Morse 序列构造 hack 的。

b 为偶数 由于 $b^k \mod p = 0$, 故这种情况非常容易 hack。

b 为奇数 下面, 考虑定义 2.4 中的 TM_{2^n} 和 $\overline{TM_{2^n}}$ 的哈希值:

$$W_0 = \sum_{i=0}^{2^n-1} T(i)b^i \quad W_1 = \sum_{i=0}^{2^n-1} (1-T(i))b^i$$

考虑 $f(n) = W_1 - W_0$ 的值:

¹⁰Jean-Paul Allouche, Thue, Combinatorics on words, and conjectures inspired by the Thue-Morse sequence, J. de Théorie des Nombres de Bordeaux, 2015

$$\begin{aligned} f(n) &= W_1 - W_0 = \sum_{i=0}^{2^n-1} (-1)^{T(i)} b^i = \prod_{i=0}^{n-1} (1 - b^{2^i}) \\ &= (1-b) \prod_{i=1}^{n-1} [(1-b^{2^{i-1}})(1+b^{2^{i-1}})] \\ &= f(n-1)(1-b) \prod_{i=0}^{k-2} (1+b^{2^i}) \end{aligned}$$

由于 $1-b, 1+b^{2^i}$ 都是偶数, 故 $f(n) = a2^n f(n-1)$, 即 $f(n)$ 的质因子中至少有 $\frac{n(n+1)}{2}$ 个 2。所以, hack 自然溢出, 即 $p = 2^{64}$ 的情况, 只需令 n 取 11 就能使 $W_0 \equiv W_1 \pmod{p}$ 。

7 总结

本文首先介绍了 Thue-Morse 序列的四种等价定义方式。接着讨论了 Thue-Morse 序列和其扩展序列在等幂和问题的应用。其次通过引入准 TM 前缀串和 TM 拆分的概念, 深入探究了 Thue-Morse 子串结构和 Thue-Morse 串的无重叠性质, 并依此构造出字符集大小为 3 的无限无平方串、分析 Thue-Morse 串的最短 n -特征子串(前缀)长度。然后推导了 Thue-Morse 生成函数的性质并介绍了其部分应用。最后探究了一个关于 Thue-Morse 序列的优美结论——Shevelev 猜想, 介绍了 Thue-Morse 序列在 OI 中 hack 传统的自然溢出的字符串哈希方式的应用。

希望本文上述的一些思路方法可以使读者对 Thue-Morse 序列有更深入的理解。

8 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢衢州二中叶敏红老师的关心和指导。

感谢家人、朋友对我的支持与鼓励。

感谢方心童同学与我讨论、给予我启发并为本文刊误。

9 参考文献

[1] OEIS A010060, <https://oeis.org/A010060>

[2] OEIS A036577, <https://oeis.org/A036577>

[3] Christopher Williamson, An Overview of the Thue-Morse Sequence, 2012

- [4] Vladimir Shevelev, Equations of the Form $t(x + a) = t(x)$ and $t(x + a) = 1 - t(x)$ for Thue-Morse Sequence, 2012
- [5] Jean-Paul Allouche, Thue, Combinatorics on words, and conjectures inspired by the Thue-Morse sequence, J. de Théorie des Nombres de Bordeaux, 2015
- [6] Hs-black, <https://www.cnblogs.com/Hs-black/p/12219270.html>, 2020
- [7] Ricardo Astudillo, On a Class of Thue-Morse Type Sequences, 2003
- [8] Hieu D. Nguyen, A Generalization of the Digital Binomial Theorem, 2015

浅谈矩阵乘法与归约

杭州学军中学教育集团文渊中学 陈诺

摘要

矩阵乘法的时间复杂度优化是一个经典问题，目前的最优算法可以在 $O(n^{2.371339})$ 的时间复杂度内解决 $n \times n$ 的整数矩阵乘法。与矩阵乘法的归约可以用作一些问题时间复杂度上下界的证明。本文将介绍归约的定义，一些经典问题与矩阵乘法之间的归约，以及用矩阵乘法得到的一些经典问题的更优时间复杂度算法。

1 定义与约定

1.1 问题

对于一个问题，我们可以将其记作函数 A ，其中 x 代表输入的变量， $A(x)$ 代表问题的答案。解决问题的过程即为设计一个算法来计算 $A(x)$ 。

对于 OI 中的问题，我们认为 $A(x)$ 都是确定型图灵机上的算法。

在本文中，我们用一个算法使用的计算时间来比较算法的优劣和问题的难度，一个问题的难度可以定义为目前已知最优算法的复杂度。

1.2 归约

对于两个问题 A 和 B ，我们可以通过找到一个便于计算的函数 $f(x)$ 满足 $A(x) = B(f(x))$ ，来证明一个问题 B 比问题 A 更困难。

因为我们可以通过使用解决 B 的算法来解决 A ，如果计算 $f(x)$ 的复杂度低于计算 $B(x)$ 的复杂度，则忽略计算 $f(x)$ 的复杂度，可以说明问题 A 比问题 B 更简单，解决 A 的时间复杂度必定不超过解决 B 的时间复杂度。

我们用 $A \leq B$ 来说明 A 可以归约到 B 。

1.3 双向归约

如果 $A \leq B$ 且 $B \leq A$ ，容易发现它们的难度是相等的，即复杂度也是相等的。

1.4 矩阵乘法

对于 $N \times N$ 的矩阵 A, B , 我们定义其乘积 $A \times B$ 为 $N \times N$ 的矩阵 C , 满足 $C_{i,j} = \sum A_{i,k} \cdot B_{k,j}$ 。这里的运算 $(+, \cdot)$ 可以对于不同的矩阵元素集合定义, 有以下常见的矩阵乘法:

- 整数矩阵乘法: 元素集合是 $\mathbb{Z}$, 运算是加法和乘法。
- 实数矩阵乘法: 元素集合是 $\mathbb{R}$, 运算是加法和乘法。
- 模意义矩阵乘法: 元素集合是 $\mathbb{Z}/m$, 运算是模意义的加法和乘法。
- 01 矩阵乘法: A, B 的元素集合是 $\{0, 1\}$, C 的元素集合是 $\mathbb{Z}$, 运算是加法和乘法。
- bool 矩阵乘法: 元素集合是 $\{0, 1\}$, 运算是逻辑或和逻辑与。

如果我们认为两种运算都是 $O(1)$ 的, 整数矩阵乘法目前的最优时间复杂度是 $O(n^\omega)$, 其中 ω 的值截止笔者提交论文时约为 2.371339。¹

在 OI 中, 时间复杂度低于 $O(n^3)$ 的矩阵乘法往往由于其过大的时间常数与实现难度而 not practical。所以我们通常认为, 对于不弱于规模为 $\sqrt{n}$ 的矩阵乘法的问题, $O(n^{1.5})$ 的算法可以作为其 practical 的最优解。

因此, 对于下文提到的, 能归约到矩阵乘法的问题, 实际上在各大 OJ 中代码和运行时间最短的解都是 $O(n^{1.5})$ 的。在现代 OI 中, 由于评测机性能的发展和对时空限制更宽裕的资源, 选手通常很难直接通过数据范围来推测一道题目预期解的时间复杂度。如果选手能够快速将一个问题归约到矩阵乘法, 就可以先开始考虑 $O(n^{1.5})$ 或更高时间复杂度的算法。

2 矩阵乘法内部的归约

2.1 01 矩阵乘法到整数矩阵乘法

我们用 $a = \sum a_i 2^i$ 表示一个非负整数的二进制分解形式, 其中 $a_i \in \{0, 1\}$ 。

注意到 $a \cdot b = \sum_{i,j} 2^{i+j} a_i b_j$, 因此值域不超过 2^k 的整数矩阵乘法可以被拆分成 k^2 次 01 矩阵乘法结果的和, 而整数矩阵乘法显然强于 01 矩阵乘法, 所以在 k 是常数的情况下, 我们认为 01 矩阵乘法和整数矩阵乘法存在双向归约。

2.2 bool 矩阵乘法到 01 矩阵乘法

此处展示一个单向归约的例子, 显然 01 矩阵乘法强于 bool 矩阵乘法, 但不存在从 01 矩阵乘法向 bool 矩阵乘法的归约, 所以无法说明 bool 矩阵乘法的最优时间复杂度低于 01 矩阵乘法。但据笔者所知, 目前 bool 矩阵乘法的最优复杂度也为 $O(n^\omega)$ 。

¹Wikipedia, Computational complexity of matrix multiplication, https://en.wikipedia.org/wiki/Matrix_multiplication

3 几类归约到矩阵乘法的技巧

3.1 静态问题: 构造集合信息交

有 n 个集合 S 和 k 个集合 T , 每个集合都是 $\{1, 2, \dots, m\}$ 的子集。

我们想要对于每个 i, j 求出 $|S_i \cup T_j|$ 。

构造矩阵 $A_{i,j} = [j \in S_i]$, $B_{i,j} = [i \in T_j]$, 不难发现 $|S_i \cup T_j| = (A \times B)_{i,j}$ 。

记 $N = \max(n, m, k)$, 这个问题等价于解决规模为 N 的 01 矩阵乘法, 时间复杂度 $O(N^\omega)$ 。

类似地, 我们可以对每个长度为 n 的块构造两种长度为 n 的信息作为矩阵 A, B , 其中 $A_{i,j}$ 代表第 i 块的第 j 个一类信息, $B_{i,j}$ 代表第 i 块的第 j 个二类信息, 使得第 i 块和第 j 块之间产生的信息为 $C_{i,j} = \sum_k A_{i,k} B_{k,j}$ 。

此时如果我们需要知道每两块之间的信息, 就需要知道 $A \times B$ 的结果, 于是就可以将规模为 n 的矩阵乘法归约到该问题。

因此, 我们也可以认为, 对于两个大小 $\geq n$, 任意预处理后合并代价也不低于 $O(n)$ 的信息上的区间查询, 时间复杂度的下界是 $O(n^\omega)$ 。

例题 3.1. (Count on a tree II²) 给定一棵 N 个点的树, 每个结点有权值 a_i 。有 Q 次询问, 每次要求出 (x, y) 路径上所有结点中出现过的权值数量。

在本文中的所有例题, 我们认为 N, Q 是同阶的。下文起会同时用 N 代表二者。

我们可以令树的形态为菊花套链, 让根引出 $n+k$ 条链分别代表 S_i 和 T_i , 询问 nk 次这两条链和根组成的链即可。

原问题的规模为 $(n+k)m$ 个点和 nk 次询问, 令 n, m, k 同阶则可以证明, 原问题规模为 N 时, 规模为 $\sqrt{N}$ 的矩阵乘法可以归约到该问题。

因此, 该问题的复杂度下界是规模为 $\sqrt{N}$ 的 01 矩阵乘法的复杂度下界, 即 $O(N^{\frac{\omega}{2}})$ 。

例题 3.2. (小 Z 的袜子³) 给定长度为 N 的排列 a 。有 Q 次询问, 每次询问给定 l, r , 要求出存在多少对整数 (x, y) 满足 $l \leq x < y \leq r$ 且 $a_x = a_y$ 。

此时区间信息使我们不方便直接构造两个集合 (S, T) 之间的贡献。但由于区间相等对存在可差分性, 我们可以考虑将序列进行分块, 用容斥原理把区间信息转化为块间贡献。

我们将序列分为 n 块, 每块的元素都是 $\{1, 2, \dots, n\}$ 的子集。考虑记 $[l, r]$ 块对应的区间的答案为 $F(l, r)$, 可以发现第 x 块和第 y 块之间的贡献可以用 $F(x, y) - F(x, y-1) - F(x+1, y) + F(x+1, y-1)$ 计算。

此时我们已经可以通过构造询问求出所有满足 $l \leq r$ 的 $F(l, r)$ 了, 而对于 $l > r$ 的情况也是平凡的, 只需要倍长序列即可。

²<https://www.spoj.com/problems/COT2/>

³<https://www.luogu.com.cn/problem/P1494>

因为此处需要求解的块间信息和前文的问题是一样的，所以规模为 $\sqrt{N}$ 的整数矩阵乘法也可以归约到区间相等对计数。

因此，该问题的复杂度下界也是规模为 $\sqrt{N}$ 的 01 矩阵乘法的复杂度下界，即 $O(N^{\frac{3}{2}})$ 。

例题 3.3. (*Yuno loves sqrt technology III*⁴) 给定长度为 N 的排列 a 。有 Q 次询问，每次询问给定 l, r ，要求出区间 $[l, r]$ 中，所有数出现次数的最大值。

事实上区间众数的信息已经两个集合 (S, T) 之间的贡献。但由于区间相等对存在可差分性，我们可以考虑将序列进行分块，用容斥原理把区间信息转化为块间贡献。

我们将序列分为 $2n$ 块，奇数块的元素是 $\{1, 2, \dots, n\}$ 的子集 $S_1, S_2, \dots, S_n$ ，偶数块是奇数块的补集。考虑记 $[2l, 2r-1]$ 块对应的区间的答案为 $(r-l+1) + [S_r \subset S_l]$ ，可以平凡地转化为 $[S_l \cap S_r \neq \emptyset]$ ，即 bool 矩阵乘法。

因此，该问题的复杂度下界是规模为 $\sqrt{N}$ 的 bool 矩阵乘法的复杂度下界，目前为 $O(N^{\frac{3}{2}})$ 。

3.2 动态问题：构造任意子集信息

前文中，我们使用将序列分块的方法构造块间信息为块内信息的矩阵乘法结果。对于有修改的问题，同样可以使用序列分块的方法，来构造对指定子集的信息查询。

我们可以对每个长度为 n 的块构造两种长度为 n 的信息作为矩阵 A ，再构造 n 个块的 n 个子集 B ，其中 A_{ji} 代表第 i 块的第 j 个信息， B_{ji} 代表第 i 块是否在第 j 个询问集合内，使得第 j 个询问集合对应元素的第 i 类信息的和为 $C_{i,j} = \sum_k A_{i,k} B_{k,j}$ 。

此时如果我们需要知道每个集合的信息，就需要知道 $A \times B$ 的结果，于是就可以将规模为 n 的矩阵乘法归约到该问题。

因此，我们也可以认为，对于大小为 n ，预处理后对所有可能的查询总信息量不低于 $O(n)$ 的，修改可以撤销和覆盖为平凡信息上的信息，对该信息对应的序列进行区间修改查询，时间复杂度的下界是 $O(n^\omega)$ 。

例题 3.4. (区间加全局数零) 给定长度为 N 的序列 a 。有 Q 次操作，操作有修改和询问两类。每次修改给定 l, r, v ，需要将区间 $[l, r]$ 加上整数 v 。每次询问给定 l, r ，要求出存在多少位置 x 满足 $a_x = 0$ 。

我们将序列分为 n 块 $[c_1, c_2, \dots, c_n]$ ，每块的元素都是 $\{1, 2, \dots, n\}$ 的子集；再构造 n^2 个询问，每 n 个询问为一组，每组都取出一个 $\{c_1, c_2, \dots, c_n\}$ 的子集 q_i ，并依次询问 q_i 中 $1, 2, \dots, n$ 的出现次数。

我们说明要获取以上信息只需要 $O(n^2)$ 次操作：

- 对于每组询问，如果该块不在子集中，将其加上 $+\infty$ ，这需要 $O(n)$ 次操作。

⁴<https://www.luogu.com.cn/problem/P5048>

- 将全局每次减去 1，并查询全局 0 的个数，这需要 $O(n)$ 次操作。
- 撤回每组询问的修改。
- 由于共有 n 个询问组，我们只需要 $O(n^2)$ 次操作。

构造 $A_{i,j} = [i \in c_j]$ ， $B_{i,j} = [q_j \in c_i]$ ，不难发现我们要求出矩阵 $C = A \times B$ 。因此，规模为 $\sqrt{N}$ 的 01 矩阵乘法可以归约到该问题，该问题的复杂度下界也是规模为 $\sqrt{N}$ 的 01 矩阵乘法的复杂度下界，即 $O(N^{\frac{3}{2}})$ 。

例题 3.5. (行加列求和问题) 给定二维平面上的 N 个点 (i, j) ，点权初始为 0，有 Q 次操作，操作有修改和询问两类。每次修改给定 x ，需要将 $i = x$ 的所有点的点权加 1。每次询问给定 y ，要求出 $i = y$ 的所有点的点权和。

不妨将平面上的稀疏点集限制到 $n \times n$ 的稠密点集。

我们称 (i, j) 上的点为第 i 行第 j 列上的点，构造每行的点集 c_i 是 $\{1, 2, \dots, n\}$ 的子集；再构造 n^2 个修改和 n^2 个询问，每 n 个修改询问为一组，每组都取出一个 $\{c_1, c_2, \dots, c_n\}$ 的子集 q_i 进行修改，并依次询问每一列的点权和。

构造 $A_{i,j} = [i \in c_j]$ ， $B_{i,j} = [q_j \in c_i]$ ，不难发现我们要求出矩阵 $C = A \times B$ 。因此，该问题的复杂度下界也是规模为 $\sqrt{N}$ 的 01 矩阵乘法的复杂度下界，即 $O(N^{\frac{3}{2}})$ 。

4 一些问题之间的归约

4.1 矩形逆序对与区间逆序对的归约

例题 4.1. (时代的眼泪⁵) 给定长度为 N 的排列 a 。有 Q 次询问，每次询问给定 l, r, u, d ，要求出存在多少对整数 (x, y) 满足 $l \leq x < y \leq r$ 且 $u \leq a_x \leq a_y \leq d$ 。

区间逆序对归约到矩形逆序对是平凡的，我们来介绍一个分治算法将其归约到区间逆序对问题⁶。

对平面建树套树，保留至少存在一个点的矩形，可以证明只有 $O(npolylog(n))$ 个这样的矩形，它们内部的点数之和也是 $O(npolylog(n))$ 。对于一个询问，将其定位到树套树上的 $O(\log^2 n)$ 个子矩形，我们发现所有的点对可以分为以下三类：

- 两个点在同一矩形内。
- 两个点不在同一矩形内，其横坐标在树套树其中一维的同一区间内。

⁵出自 NOI 2020

⁶<https://www.luogu.com.cn/problem/P6774>

- 两个点不在同一矩形内，其纵坐标在树套树其中一维的同一区间内。
- 两个点在树套树中横纵坐标均不交的两个矩形中。

统计第一类和第四类的逆序对数是可以的在 $O(n \text{polylog}(n))$ 的时间内完成的，我们可以用容斥来简化第二类和第三类的贡献。对于每一行和每一列，我们计算整行和整列内的逆序对，并发现其减去同一矩形内的贡献就可以得到前三类贡献的和。

于是我们只需要考虑求 $O(n \log n)$ 次树套树上每一行或一列的区间逆序对，由于所有行的点数之和是 $O(n \log n)$ 的，可以将该问题归约到区间逆序对。事实上，如果我们对这个问题进行更深入的分析，我们事实上是要对每个长度为 L 的线段树结点（其最多只包含 L 个点）求 L 次区间逆序对，此时调用矩阵乘法的时间复杂度为 $T(n) = O(n^\omega) + 2T(\frac{n}{2}) = O(n^\omega)$ ，所以该归约不需要忽略 $\text{polylog}(n)$ 的时间复杂度。

4.2 区间查询问题的双向归约 [2]

定义 4.1. 非正交半平面多项式：定义 $F(x, y) = \sum P_i(x, y)[A_i x + B_i y \geq 0]$ ，其中 $P_i(x, y)$ 为关于 x, y 的常数次多项式； $F(x, y)$ 是非正交半平面多项式当且仅当存在一组 (A_i, B_i) 满足 $A_i \neq 0$ 且 $B_i \neq 0$ 。

定义 4.2. 区间查询问题：给定长度为 N 的序列 a 和项数不超过 $\text{polylog}(N)$ 的非正交半平面多项式 $F(x, y)$ 。有 Q 次询问，每次询问给定 $l \leq r$ ，要求出所有满足 $l \leq x < y \leq r$ 的整数对 (x, y) 的 $F(a_x, a_y)$ 之和；我们称 $F(x, y) = [x = y]$ 的问题为区间相等对问题，即例题 3.2。

定义 4.3. 二区间查询问题：给定长度为 N 的序列 a 和项数不超过 $\text{polylog}(N)$ 的非正交半平面多项式 $F(x, y)$ 。有 Q 次询问，每次询问给定 $l_1 \leq r_1 < l_2 \leq r_2$ ，要求出所有满足 $l_1 \leq x \leq r_1 < l_2 \leq y \leq r_2$ 的整数对 (x, y) 的 $F(a_x, a_y)$ 之和；我们称 $F(x, y) = [x = y]$ 的问题为二区间相等对问题。

如果我们将每一对 (i, j) 的贡献写在二维平面上，两个问题分别代表着查询一个斜边在主对角线上的等腰直角三角形，和一个不跨过主对角线的矩形。

我们来尝试证明，对于同一个 $F(x, y)$ ，若 a_i 的值域为 $\text{poly}(n)$ 级别，这两个问题可以在忽略 $\text{polylog}(n)$ 意义的时间复杂度下双向归约到区间相等对问题。

引理 4.1. 对于一个有 c 项， $P(x, y)$ 次数不超过 d 的非正交半平面多项式 $F(x, y) : \mathbb{Z}^2 \rightarrow \mathbb{Z}$ ，有 $k = O(c \cdot d \cdot \log^{d+1} U)$ 满足存在一组长度为 k 的，可以在常数时间内计算的函数序列 g_i, h_i ，和系数序列 α_i ，使得对于 $1 \leq x, y \leq U$ 均有 $F(x, y) = \sum_{i=1}^k \alpha_i \cdot [g_i(x) = h_i(y)]$ 。 [1]

引理 4.2. 对于一个 $P(x, y)$ 次数不超过 d 的非正交半平面多项式 $F(x, y) : \mathbb{Z}^2 \rightarrow \mathbb{Z}$ ，有 $k = O(d^2)$ 满足存在一组长度为 k 的，可以在常数时间内计算的函数序列 g_i, h_i ，和系数序列 α_i ，使得对于 $1 \leq x, y \leq U$ 均有 $[x = y] = \sum_{i=1}^k \alpha_i \cdot F_i(g_i(x), h_i(y))$ ，且 $F_i \in \{F, f(x, y) = x \cdot y\}$ 。

这两个问题建立了非正交半平面多项式 $F(x, y)$ 与相等函数 $[x = y]$ 的转化关系，其中定理 4.2 是定理 4.1 的反演形式。这两个定理的证明较为复杂，故不在本文中描述，可以阅读参考文献中的证明。

定理 4.1. 二区间查询问题和二区间相等对问题存在双向归约。

证明. 在使用 $\text{polylog}(N)$ 个函数描述另一种函数之后，对于任意一个二区间查询问题，我们只需要先对于每个 t 计算出 $g_t(a_i)$ 和 $h_t(a_i)$ ，再计算这 k 个区间的二区间相等对问题，最后将每个问题的答案乘上 α_i 并求和即可。类似地，我们也可以证明另一个方向的归约（ $f(x, y) = x \cdot y$ 的问题是平凡的），所以二区间查询问题和二区间相等对问题存在双向归约。 $\square$

定理 4.2. 区间查询问题和二区间查询问题存在双向归约。

证明. 将区间查询问题归约到二区间查询问题：记第一个问题的区间答案为 $f([l, r])$ ，第二个问题的答案为 $f([l_1, r_1], [l_2, r_2])$ ，我们可以发现 $f([l_1, r_1], [l_2, r_2]) = f([l_1, r_2]) - f([l_1, l_2]) - f([r_1, r_2]) + f([r_1, l_2])$ 。

将二区间查询问题归约到区间查询问题：因为 $f([l, r]) = f([1, r]) - f([1, l]) - f([1, l], [l, r])$ ，所以我们只需要快速求出 $f([1, i])$ 即可。运用定理 4.1 可以发现，只需要使用哈希表或平衡树解决这 $\text{polylog}(N)$ 个前缀相等对计数即可。 $\square$

综上，我们可以获取一个直观的结论。

定理 4.3. 任意区间查询问题和二区间相等对问题存在忽略 $\text{polylog}(N)$ 时间复杂度的双向归约，它们均可以被大小为 $\sqrt{N}$ 的 01 矩阵乘法归约。

我们可以列举一些常见的二元函数，发现许多非平凡的信息都可以表示成非正交半平面多项式，从而可以与二区间相等对归约：

- $[x = y]$ 可以表示为 $1 \cdot [x - y \geq 0] + 1 \cdot [-x + y \geq 0] - 1$ 。
- $[x > y]$ 可以表示为 $1 \cdot [x - y > 0]$ 。
- $\max(x, y)$ 可以表示为 $x \cdot [x - y > 0] + y \cdot [-x + y \geq 0]$ 。
- $|x - y|$ 可以表示为 $(x - y) \cdot [x - y > 0] + (y - x) \cdot [-x + y \geq 0]$ 。

4.3 三元环计数与二区间相等对的归约

定义 4.4. 三元环计数问题：给定一张 N 个点， M 条边的无向图，你需要对于每条边 (v, w) 求出存在多少个点 u ，满足 $(u, v), (u, w)$ 两个点对之间都有边连接。在下文中，我们用 M 来代表该问题的规模。

定理 4.4. 三元环计数问题可以归约二区间相等对问题。

证明. 我们枚举每一条边 (u, v) , 考虑求出 u 的所有出边和 v 的所有出边的交, 即为 (u, v) 所在的所有三元环。对于每个点, 令 E_i 为 i 的出边序列, 将序列拼接后 u 的所有出边和 v 的所有出边的交即为一个二区间相等对查询。

用 M 次二区间相等对查询求出所有点所在的三元环数即可。□

定理 4.5. 二区间相等对问题可以归约三元环计数问题。

证明. 使用 4.1 中的树套树分治, 可以将 $O(N)$ 个询问变为 $O(N \log^2 N)$ 个对两个线段树区间的询问, 故只需考虑求出 $O(N \log^2 N)$ 对线段树区间之间的贡献。

构造带权三分图 $G = (U \cup V \cup W, E_{U,V} \cup E_{V,W} \cup E_{U,W})$, 其中 V, W 均为序列在线段树上的 $O(N)$ 个区间, U 为值域中的 $O(N)$ 个值。

对于 $E_{U,V} \cup E_{U,W}$, 我们从区间 $v_{l,r} \in V$ 向 $u_i \in U$ 连边的边权等于 i 在 $[l, r]$ 中的出现次数。由于线段树所有结点的长度之和是 $O(N \log N)$ 的, 所以此部分边数也是 $O(N \log N)$ 的。

对于 $E_{V,W}$, 我们则对所有至少被询问一次的线段树区间对连边, 边数是 $O(N \log^2 N)$ 的。

不难发现该图具有 $O(N \text{polylog}(N))$ 个点和边。我们对该图求解带权三元环计数问题后, 只需要对每个询问枚举所有线段树区间之间的贡献, 求和即可, 因此二区间相等对问题可以归约到带权三元环计数问题。对于三分图上的带权三元环计数问题, 我们用 2.1 中的方法将其进行二进制拆分, 因为值域不超过 $\log N$ 位, 因此三元环计数问题也可以归约到该问题。□

定理 4.6. 三元环计数问题和二区间相等对问题存在忽略 $\text{polylog}(N)$ 时间复杂度的双向归约, 它们均可以从大小为 $\sqrt{N}$ 的 01 矩阵乘法归约得到。

5 使用矩阵乘法改良时间复杂度

归约矩阵乘法可以帮我们得到一个复杂度下界, 也可以帮我们得到一个复杂度上界。对于一些存在 $O(n^{1.5})$ 的经典算法的题目, 我们可以使用矩阵乘法来得到更优的算法。

对于这一类算法, 我们往往将序列分块后, 对值域进行阈值分治, 从而对 B 个块和 B 个值进行矩阵乘法。我们的目标通常是在进行 $O(B^\omega)$ 的矩阵乘法后, 每个询问可以用 $O(\frac{N}{B})$ 的时间复杂度求解。

5.1 区间逆序对问题

例题 5.1. (*Yuno loves sqrt technology II*⁷) 给定长度为 N 的排列 a 。有 Q 次询问, 每次询问给定 l, r , 要求求出存在多少对整数 (x, y) 满足 $l \leq x < y \leq r$ 且 $a_x > a_y$ 。

⁷<https://www.luogu.com.cn/problem/P5047>

将序列分为 B 个大小为 $\frac{N}{B}$ 的块, 任意询问显然可以在 $O(\frac{N}{B})$ 的时间内被转化到若干个整块之间的询问。

整块内部的贡献同样也可以 $O(\frac{N^2}{B})$ 进行预处理, 只需要考虑高效求出任意两个整块之间的贡献。

将值域同样分为 B 个大小为 $\frac{N}{B}$ 的块, 对于不在同一值域块的部分, 我们记 $F_{x,y}$ 为第 x 个序列块中在第 y 个值域块中的数的个数, 记 $G_{y,z}$ 为第 z 个序列块中在前 $y-1$ 个值域块的数的个数, 不难发现 $(F \times G)_{i,j}$ 即为第 i 块向第 j 块的贡献, 这是一个规模为 B 的整数矩阵乘法, 所以此处时间复杂度为 $O(B^\omega)$ 。

而在同一值域块的部分, 注意到只存在 $O(\frac{N^2}{B})$ 个二元组, 所以可以直接枚举所有的二元组计算贡献。

所以我们可以 $O(B^\omega + \frac{N^2}{B}) = O(N^{\frac{2\omega}{\omega+1}})$ 的时间复杂度内解决这个问题。因为区间逆序对与矩形逆序对双向归约, 可以得到矩形逆序对的同时时间复杂度算法。

同理, 因为相等对可以通过所有二元组减去顺序对和逆序对的方法计算, 区间相等对也可以通过这个方法在 $O(N^{\frac{2\omega}{\omega+1}})$ 的时间复杂度内解决。因为区间相等对与所有区间查询问题双向归约, 所以同样可以得到对所有区间查询问题的 $O(N^{\frac{2\omega}{\omega+1}} \text{polylog}(N))$ 时间复杂度算法。

5.2 三元环总数计数问题

定义 5.1. 三元环总数计数问题: 给定一张 N 个点, M 条边的无向图, 你需要求出存在多少个无序三元组 (u, v, w) , 满足 $(u, v), (v, w), (w, u)$ 三个点对之间都有边连接。在下文中, 我们用 M 来代表该问题的规模。

显然三元环总数计数问题可以归约到三元环问题, 得到时间复杂度 $O(M^{\frac{2\omega}{\omega+1}} \text{polylog}(M))$ 的做法, 下文介绍一种可以消去时间复杂度中 $\text{polylog}(M)$ 的做法。

对度数进行阈值分治, 记度数最大的 B 个点的点集为 V , 考虑构建图的邻接矩阵 G , 对于每条边求出 $G_{i,j}^2$ 的和即可算出 $u, v, w \in V$ 的三元环数量, 时间复杂度 $O(B^\omega)$ 。

对于剩下的点, 因为其度数不超过 $O(\frac{M}{B})$, 可以直接枚举两条边来计算 $u \notin V$ 的三元环, 时间复杂度 $O(\frac{M^2}{B})$ 。

故总时间复杂度为 $O(B^\omega + \frac{M^2}{B}) = O(M^{\frac{2\omega}{\omega+1}})$ 。

5.3 行加列求和问题

题目见例题 3.5。

我们只保留点数最多的前 B 行 B 列并构建新矩阵, 对于其余行列, 由于其点数不超过 $O(\frac{N}{B})$, 可以暴力枚举每个点所在的行被操作的次数, 或暴力将对应列的答案加上 1, 时间复杂度 $O(\frac{N^2}{B})$ 。我们对选中行列的点重新离散化, 得到的新点集 $V \subset \{1, 2, \dots, B\} \times \{1, 2, \dots, B\}$ 。

将所有操作分为 B 个长度为 $\frac{N}{B}$ 的块，将贡献分为块内和块间两种。对于块内的贡献，可以对于每个询问遍历所有块内的操作，时间复杂度 $O(\frac{N^2}{B})$ 。对于块间的贡献，构造 $A_{i,j} = [(i,j) \in V]$, $B_{i,j}$ 为第 j 块的操作中对第 i 行的修改次数，不难发现我们可以对矩阵 $C = A \times B$ 进行前缀和后求出所有块间贡献，时间复杂度为 $O(B^\omega)$ 。

总时间复杂度为 $O(B^\omega + \frac{N^2}{B}) = O(N^{\frac{2\omega}{\omega+1}})$ 。

5.4 链上颜色数问题

题目见例题 3.1。

定义 5.2. 对于无向树 T ，定义 T 上的一个簇为三元组 $C = (V, E, B)$ ，其中 (V, E) 连通且是 T 的子图， $B = \{x \mid \exists (x, y) \in T, (x, y) \notin E\} \setminus V$ 且 $|B| \leq 2$ 。我们称 B 中的元素为 C 的界点。

不难发现一个边集 E 要么没有对应的簇，要么存在唯一的簇 C 。

定义 5.3. Top Tree: 对于无向树 T ，可以构造一棵新的有根二叉树 T' 满足每个结点是一个簇 C_i ，根结点对应的点集为 T 的点集，每个叶结点都有 $|E| = 1$ ，非叶结点的边集为其孩子的边集的并，构造一棵深度可以在 $O(n)$ 时间内完成。[3]

定义 5.4. Top Cluster: 对于无向树 T ，可以将其边集划分至 $O(B)$ 个簇 C_i ，使得每个簇的大小均不超过 $O(\frac{|V|}{B})$ 。我们可以通过在 $Top\ Tree$ 上不断合并且保持子树大小不超过 $O(\frac{|V|}{B})$ 来快速构建 $Top\ Cluster$ 。也存在更简洁且便于实现的方法：在树上不断递归，直到找到一棵只有根的子树大小 $\geq B$ 的点。令一个簇只包含根结点，贪心加入所有子树，在簇的大小 $> 2B$ 时完成该簇的构建，再开始构建下一个簇，最后将该子树删除，只保留根结点即可。

我们先将树三度化，再使用树分块算法将树分为 $O(B)$ 个簇。容易发现此时有 $O(B)$ 个界点，树上任意点对的路径均可以划分为 $O(\frac{N}{B})$ 个散点和一条界点之间路径的并。假如我们已经求出所有界点之间路径上的颜色数量， $O(1)$ 判断每个散点颜色是否出现过是平凡的。

而对于界点之间路径上的颜色数量，我们使用以子树内界点数量为权重的边分治来快速计算。由于我们只需要递归子树内有界点的结点，此时的边分治仍然至少将问题的规模变为原来的 $\frac{2}{3}$ 。我们取出 B 种出现次数最多的颜色，在一次分治中的问题中，可以类似前文构造矩阵乘法来计算这 B 种颜色在所有界点路径上的出现次数，记当前分治子树内有 M 个界点，时间复杂度为 $T(M) = O(M^\omega) + M \cdot \frac{N}{B} + \frac{3}{2}T(\frac{2}{3}M)$ ，因此 $T(B) = O(B^\omega + N \log N)$ 。

而对于出现次数较少的颜色，我们可以直接对每种颜色建立虚树，并暴力求出虚树上被这些点分割的至多 $O(\frac{N}{B})$ 个连通块的两两贡献，我们可以将连通块处理为 DFS 序上的空间，再将 DFS 序对 B 个界点离散化，最后用二维前缀和求出答案即可，时间复杂度为 $O(\frac{N^2}{B})$ 。

因此，该问题的总时间复杂度为 $O(B^\omega + \frac{N^2}{B} + N \log N) = O(N^{\frac{2\omega}{\omega+1}})$ 。

6 总结

本文介绍了归约的定义和一些矩阵乘法归约到数据结构问题的技巧和例子，也介绍了其他不同类型问题之间的归约，还展示了一些用矩阵乘法来加速部分经典问题求解的算法。

同时，上文中也仍有一些尚未能与矩阵乘法归约的问题，和一些未能通过矩阵乘法得到低于 $O(n^{1.5})$ 的时间复杂度的问题，希望读者能够继续探索这类问题的更优复杂度算法。

7 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练张艺缤、彭思进、杨耀良的指导。

感谢学军中学徐先友等老师的关心与指导。

感谢李欣隆，周康阳，时庆钰学长对本文的指导和建议。

感谢家人对我的陪伴与支持。

感谢信友队的同学、学长给予我帮助与陪伴。

感谢其他给予我帮助的老师与同学。

参考文献

- [1] Przemysław Uznański Karim Labib and Daniel WollebGraf. Hamming distance completeness. Annual Symposium on Combinatorial Pattern Matching, 2019.
- [2] Adam Polak Lech Duraj, Krzysztof Kleiner and Virginia Vassilevska Williams. Equivalences between triangle and range query problems. Symposium on Discrete Algorithms, 2020.
- [3] Renato F. Werneck Robert E. Tarjan. Self-adjusting top trees. Symposium on Discrete Algorithms, 2005.

一些经典问题模小质数幂的做法

北京市十一学校 武林

摘要

模小质数幂的计数问题在数论中占据重要地位。本文以总结模小质数幂计数问题的经典方法为目标，采用生成函数和数论方法对相关问题进行整理与分析。研究围绕经典问题展开，阐述了不同方法在解题过程中的具体运用。

1 约定

下文讨论的特殊模数默认是 p^k ，满足 $p \in \mathbb{P}, k \in \mathbb{Z}^+$ 且 p 比较小。

记 $v_p(n)$ 为最大的满足 $p^k \mid n$ 的自然数 k 。

记 $f_p(n)$ 为 $n!$ 去掉所有的 p 因子后的值。

2 多项式点值操作

特殊模数下的多项式点值操作是一种常见问题，本节将介绍几个与点值相关的问题与解法。

引理 1 (勒让德公式)。

$$v_p(n!) = \sum_{i=1}^{\infty} \left\lfloor \frac{n}{p^i} \right\rfloor$$

引理 2. 对于任意整数 n 与自然数 k ，有 $k! \mid n^k$ 。

证明. 考虑对于每个质数 p 证明 $v_p(n^k) \geq v_p(k!)$ 。

类似于勒让德公式，我们有

$$v_p(n^k) = \sum_{i=1}^{\infty} \left\lfloor \frac{n}{p^i} \right\rfloor - \left\lfloor \frac{n-k}{p^i} \right\rfloor$$

那么有

$$v_p(n^k) - v_p(k!) = \sum_{i=1}^{\infty} \left\lfloor \frac{n}{p^i} \right\rfloor - \left\lfloor \frac{n-k}{p^i} \right\rfloor - \left\lfloor \frac{k}{p^i} \right\rfloor$$

求和式的每一项形如 $\lfloor a+b \rfloor - \lfloor a \rfloor - \lfloor b \rfloor \geq 0$ ，故最终结果也非负。

□

2.1 多项式插值

多项式插值是这样的一个问题：给定满足 $\deg < n$ 的多项式 $F(x)$ 经过的点 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ ，保证 $x_1, x_2, \dots, x_n$ 互不相同，要求还原出 $F(x)$ 。

在实数域上，我们总能还原出唯一的 $F(x)$ ，但是在模 p^k 意义下并不一定能还原。

虽然我们不能还原 $F(x)$ ，但是在给定的 x_i 满足一些限制的时候我们总是可以求出 $F(x)$ 的任意点值。

定理 1. 给定满足 $\deg < n$ 的多项式 $F(x)$ 在 $1, 2, \dots, n$ 处的整数点值 $y_1, y_2, \dots, y_n$ 对 p^k 取模的结果，对于任意整数 x_0 我们总能唯一确定 $F(x_0)$ 对 p^k 取模的结果。

证明. 考虑拉格朗日插值公式，我们有

$$F(x) = \sum_{i=1}^n y_i \prod_{j \neq i} \frac{x-j}{i-j}$$

因此有

$$\begin{aligned} F(x_0) &= \sum_{i=1}^n y_i \prod_{j \neq i} \frac{x_0-j}{i-j} \\ &= \sum_{i=1}^n (-1)^{n-i} y_i \frac{(x_0-1)^{i-1} (x_0-i-1)^{n-i}}{(i-1)!(n-i)!} \\ &= \sum_{i=1}^n (-1)^{n-i} y_i \frac{(x_0-1)^{i-1}}{(i-1)!} \frac{(x_0-i-1)^{n-i}}{(n-i)!} \end{aligned}$$

由引理 2，这两个分式都是整数，因此 $F(x_0)$ 总是 y 的一个整系数线性组合，所以利用 y 对 p^k 取模的余数即可还原出唯一的 $F(x_0)$ 。

□

更为一般地，如果给定的 x_i 形成一个区间，我们也可以确定任意 $F(x_0)$ 的值。

同时上面的做法并不要求 $F(x)$ 是一个整系数多项式，例如求 $F(x) = \frac{x(x+1)}{2}$ 模 2^k 的点值也是可行的。

例题 1. 自然¹

给定 n, m ，求 $\sum_{0 \leq i < n} i^m$ 对 2^k 取模的值。原题中 $k = 64$ 。

考虑 $\sum_{0 \leq i < n} i^m$ 是一个关于 n 的 $m+1$ 次多项式。

当 $m < k$ 时，直接插值的复杂度是 $O(k \log m)$ 的（使用快速幂求点值）。

当 $m \geq k$ 时，所有偶数项都为 0。设 $t = \lfloor (n-1)/2 \rfloor$ 考虑奇数项的贡献

$$\begin{aligned} \sum_{0 \leq i \leq t} (2i+1)^m &= \sum_{0 \leq i \leq t} \sum_{0 \leq j \leq m} \binom{m}{j} (2i)^j \\ &= \sum_{0 \leq i \leq t} \sum_{0 \leq j < k} \binom{m}{j} (2i)^j \end{aligned}$$

因此答案还是可以写成关于 t 的 k 次式，复杂度还是 $O(k \log m)$ 的。

2.2 多项式多点求值

多项式多点求值是这样的一个问题：给定 $\deg = m$ 的多项式 $F(x)$ 和 $x_1, x_2, \dots, x_n$ ，要求求出 $F(x_1), F(x_2), \dots, F(x_n)$ 。

一般在模意义下考虑的时候，我们认为多项式系数和 x_i 都是整数。

在一般情况下，我们需要比较复杂的方法才能做到较低的复杂度，不过基于模数 p^k 的性质，我们有一些更简单的做法

引理 3. 对于任意整数 x ，我们总有 $x^{p^k} \equiv 0 \pmod{p^k}$ 。

证明. 由 2，我们有 $(pk)! \mid x^{p^k}$ ，而 $v_p((pk)!) \geq k$ ，故得证。□

由这个引理，如果要对整系数多项式 $F(x)$ 做多点求值，我们可以直接求出它对 x^{p^k} 取模的结果，然后暴力计算的复杂度就变为 $O(pk(n+m))$ 。

更方便地，我们可以直接求出 $F(x)$ 在 $0, 1, \dots, pk-1$ 处的点值，然后用上述拉插求点值的方法求出需要的每个点值。

¹loj 6894, <https://loj.ac/p/6894>

事实上，最小的满足 x^d 模 p^k 恒为 0 的 d 可以分析到 $(p-1)k$ 左右，可以进一步减少需要的点值数量。

例题 2. 【模板】线段树²

有一个长度为 n 的序列 $a_1, a_2, \dots, a_n$ ，保证 a_i 是奇数，有两种操作

1. 给定 l, r, x ，将 $a_l, a_{l+1}, \dots, a_r$ 加上 x ，保证 x 是偶数；
2. 给定 l, r ，求 $a_l, a_{l+1}, \dots, a_r$ 的乘积，答案对 2^k 取模；

下面的做法不依赖对于 a_i 和 x 奇偶性的限制。

考虑使用线段树，在节点 (l, r) 上维护函数 $f(x) = \prod_{i=l}^r (x + a_i)$ ，需要支持查询点值。使用上述的优化方法，可以只记录 $O(k)$ 个连续点值。

合并左右儿子信息时直接对位乘，单次复杂度 $O(k)$ 。查询复杂度也是 $O(k)$ 。打标记时暴力求出新的 $O(k)$ 个点值，单次复杂度 $O(k^2)$ 。

可以做到总复杂度 $O(nk^2 \log n)$ 。

例题 3. Bears and Juice³

这里仅讨论该题做法中与本文相关的一个优化，在该做法中，我们最后需要对于 $1 \leq i \leq q$ 的每个 i 求出

$$\sum_{j=0}^{\min(m, n-1)} \binom{n}{j} i^j$$

对 2^k 取模的结果。

朴素做法复杂度至少为 $O(mq)$ ，考虑同样应用上述优化方法，对于前 $O(k)$ 个 i 的答案暴力求，对于后面的 i 使用拉插，可以做到 $O((m+q)k)$ 的总复杂度。

2.3 组合数求值 [1][2]

组合数求值是这样的一个问题：给定 n, m ，求出 $\binom{n}{m}$ 对 p^k 取模的结果。

对于组合数求值，我们只需要能把 $n!$ 拆成 $p^a \times b$ ，其中 $(b, p) = 1$ 即可。

a 就是 $v_p(n!)$ ，容易 $O(\log_p n)$ 直接计算，因此只需计算 b 。

不妨设 $n!$ 去掉所有的 p 因子后为 $f_p(n)$ ，那么有

$$f_p(n) = f_p(\lfloor n/p \rfloor) \prod_{1 \leq i \leq n, p \nmid i} i$$

²2023 北京市大学生程序设计竞赛, <https://qoj.ac/contest/1464/problem/7992>

³Codeforces 643F, <https://codeforces.com/problemset/problem/643/F>

设 $n = mp + v$ ，那么后面的部分可以如下拆开

$$\left(\prod_{0 \leq i < m} \prod_{1 \leq j < p} (pi + j) \right) \prod_{1 \leq i \leq v} (mp + i)$$

$mp + 1$ 到 $mp + v$ 的乘积可以暴力计算，复杂度 $O(p)$ ，考虑前面的部分如何计算。

设

$$F_m(x) = \prod_{0 \leq i < m} \prod_{1 \leq j < p} \left(\frac{i}{j}x + 1 \right)$$

这里 $\frac{1}{j}$ 是 j 模 p^k 下的逆元对应的整数，因此 $\frac{i}{j}$ 也是一个整数。原式可以写成 $(p-1)!^m F_m(p)$ 。

那么我们只需求出 $F_m(p) \bmod p^k$ ，因此只需要关注 $[x^0]F_m(x), [x^1]F_m(x), \dots, [x^{k-1}]F_m(x)$ 。

设 $G_m(x) = \ln F_m(x)$ ，我们有

$$\begin{aligned} G_m(x) &= \ln F_m(x) \\ &= \sum_{0 \leq i \leq m} \sum_{1 \leq j < p} \sum_{k \geq 1} \frac{(-1)^{k-1} i^k}{k j^k} x^k \\ &= \sum_{k \geq 1} \frac{(-1)^{k-1}}{k} x^k \sum_{0 \leq i \leq m} \sum_{1 \leq j < p} \frac{i^k}{j^k} \\ &= \sum_{k \geq 1} \frac{(-1)^{k-1}}{k} x^k \left(\sum_{0 \leq i \leq m} i^k \right) \left(\sum_{1 \leq j < p} j^{-k} \right) \end{aligned}$$

我们知道 $[x^k]G_m(x)$ 是一个关于 m 的 $k+1$ 次多项式。利用 $F_m(x) = \exp G_m(x)$ ，我们有

$$\begin{aligned} F'_m(x) &= G'_m(x)F_m(x) \\ [x^t]F_m(x) &= \frac{1}{t} \sum_{0 \leq i < t} (t-i)([x^i]F_m(x))([x^{t-i}]G_m(x)) \end{aligned}$$

依据上式以及 $[x^0]F_m(x) = 1$ ，我们可以归纳证明 $[x^t]F_m(x)$ 是一个关于 m 的不超过 $2t$ 次的多项式，且依据 $F_m(x)$ 的定义易得每个 $[x^t]F_m(x)$ 都是整数。

因此

$$\sum_{0 \leq i < k} p^i [x^i]F_m(x)$$

是一个整数，是关于 m 的不超过 $2k-2$ 次的多项式，且它与 $F_m(p)$ 模 p^k 同余。

因此 $F_m(p) \bmod p^k$ 可以通过对 $F_0(p), F_1(p), \dots, F_{2k-2}(p)$ 做连续点值拉插得到，且这些点值可以 $O(pk)$ 暴力预处理。

因此计算 $\frac{f_p(n)}{f_p(\lfloor n/p \rfloor)}$ 可以做到 $O(pk)$ 预处理， $O(p+k+\log n)$ 单次查询。这里的 $O(\log n)$ 来自计算 $(p-1)!$ 的快速幂，如果把幂次加起来最后再计算可以做到 $O(pk+\log n)$ 预处理， $O(p+k)$ 查询。

计算 $f_p(\lfloor n/p \rfloor)$ 时直接递归，总复杂度 $O((p+k)\log_p n + pk + \log n)$ 。

3 多项式求系数

许多计数问题最后都会转化为多项式求一项系数的问题，本节会介绍一些在特殊模数下求多项式系数的做法。

3.1 短多项式高次幂远处系数⁴

短多项式高次幂远处系数是这样的一个问题：给定一个 n 次多项式 $F(x) = \sum_{i=0}^n a_i x^i$ ，求出 $[x^t]F^m(x) \bmod p^k$ ，其中 n 比较小，但是 m, t 都非常大。

下文将分两步解决这个问题，先解决 $k=1$ 的情况然后推广到 $k>1$ 。

3.1.1 $k=1$ 时

引理 4. 对于质数 p 和整系数多项式 $F(x)$ ，有 $F^p(x) \equiv F(x^p) \pmod{p}$ 。

证明. 考虑多项式系数 $\binom{p}{a_0, a_1, \dots, a_m}$ ， $v_p\left(\binom{p}{a_0, a_1, \dots, a_m}\right)$ 只有在 $\exists i, a_i = p$ 的时候才为 0，且此时一定有 $\binom{p}{a_0, a_1, \dots, a_m} \equiv 1 \pmod{p}$ ，因此 $F^p(x)$ 的每一项形如 $(a_i x^i)^p$ 。

由费马小定理，有 $a^p \equiv a \pmod{p}$ ，因此 $F^p(x) \equiv F(x^p) \pmod{p}$ 。

□

考虑解决以下不弱于 $k=1$ 的一个问题，求 $[x^t]F^m(x)G(x) \bmod p$ 的值。

我们把 m 和 t 拆分为 $m = a_1 p + b_1$ ， $t = a_2 p + b_2$ ，运用以上引理，可以得到

⁴LibreOJ NOI Round #2 简单算术，<https://loj.ac/p/577>

$$\begin{aligned} [x^t]F^m(x)G(x) &= [x^t]F^{a_1p+b_1}(x)G(x) \\ &= [x^t]F^{a_1}(x^p)\left(F^{b_1}(x)G(x)\right) \end{aligned}$$

设 $F^{b_1}(x)G(x)$ 把系数按照次数 $\bmod p$ 的余数拆分的结果为 $\sum_{0 \leq r < p} x^r H_r(x^p)$, 那么有

$$\begin{aligned} [x^t]F^m(x)G(x) &= [x^{a_2p+b_2}]F^{a_1}(x^p)\left(x^{b_2}H_{b_2}(x^p)\right) \\ &= [x^{a_2}]F^{a_1}(x)H_{b_2}(x) \end{aligned}$$

设 $\max(\deg F(x), \deg G(x)) = n$, 那么有

$$\deg H_{b_2}(x) \leq \frac{\deg F^{b_1}(x)G(x)}{p} \leq n$$

因此我们可以把原问题转化为一个 n 不变, 且 m, t 都除以了 p 的子问题。因为过程中 $F(x)$ 不发生改变, 所以可以预处理 $F(x)$ 的 $0 \sim p-1$ 次幂, 这样预处理复杂度为 $O(np^2)$, 求解复杂度为 $O(n^2 \log_p m)$ 。

3.1.2 $k > 1$ 时⁵

在 $k=1$ 的时候我们通过引理 4 来减小 m , 那么在 $k > 1$ 的时候是否有类似的结论呢? 事实上是有的。

引理 5. 对于质数 p 和整系数多项式 $F(x)$, 有 $F^{p^k}(x) \equiv F^{p^{k-1}}(x^p) \pmod{p^k}$ 。

证明. 考虑归纳证明, 在 $k=1$ 的时候结论成立。对于 $k > 1$, 设 $F^{p^{k-1}}(x) \equiv F^{p^{k-2}}(x) + p^{k-1}G(x) \pmod{p^k}$, 我们有

$$\begin{aligned} F^{p^k}(x) &= \left(F^{p^{k-1}}(x)\right)^p \\ &= \sum_{0 \leq i < p} \binom{p}{i} F^{(p-i)p^{k-2}}(x) p^{(k-1)i} G^i(x) \end{aligned}$$

当 $i > 0$ 的时候, $i(k-1) + v_p\left(\binom{p}{i}\right) \geq k$, 因此有 $F^{p^k}(x) \equiv F^{p^{k-1}}(x^p) \pmod{p^k}$ 。

□

那么令 $G_0(x) = F^{n \bmod p^{k-1}}(x)G(x)$, $F_0(x) = F^{p^{k-1}}(x)$, $m' = \left\lfloor \frac{m}{p^{k-1}} \right\rfloor$, 要求的是 $[x^t]F_0^{m'}(x)G_0(x)$ 。

⁵<https://www.cnblogs.com/leukocyte/p/14552848.html>

仍然设 $m' = a_1p + b_1, t = a_2p + b_2$, 我们有

$$\begin{aligned} [x^t]F_0^{m'}(x)G_0(x) &= [x^{a_1p+b_1}]F_0^{a_2p+b_2}(x)G_0(x) \\ &= [x^{a_1p+b_1}]\left((F_0(x))^p\right)^{a_2}F_0^{b_2}(x)G_0(x) \\ &= [x^{a_1p+b_1}]\left((F(x^p))^{p^{k-1}}\right)^{a_2}F_0^{b_2}(x)G_0(x) \\ &= [x^{a_1}]\left((F(x))^{p^{k-1}}\right)^{a_2}H_{b_2}(x) \\ &= [x^{a_1}]F_0^{a_2}(x)H_{b_2}(x) \end{aligned}$$

容易发现之前的复杂度分析仍然适用, 只是 n 变成了 $p^{k-1}n$, 因此复杂度为 $O(n^2 p^{2k})$ 预处理, $O(n^2 p^{2k-2} \log_p m)$ 求解。

3.2 短多项式复合逆远处系数⁶

短多项式复合逆远处系数是这样的一个问题: 给定一个多项式 $F(x) = \sum_{1 \leq i \leq n} f_i x^i$, 满足 $f_1 \neq 0$, 求出满足 $F(G(x)) = x$ 的多项式 $G(x)$ 的 x^t 次系数 $\bmod p^k$ 的值。其中 n 比较小但 t 很大。

求解复合逆单项系数一般会使用拉格朗日反演, 其形式是

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

但是这个式子需要对 n 求逆, 在模 p^k 的情况下不一定可行。考虑使用另类拉格朗日反演 [3]

$$\begin{aligned} [x^n]G^m(x) &= [x^{-m-1}]F'(x)F^{-n-1}(x) \\ &= [x^{n-m}]F'(x)\left(\frac{F(x)}{x}\right)^{-n-1} \end{aligned}$$

这样可以直接套用短多项式高次幂远处系数的做法, 不过需要注意的是因为次数是负数, 所以递归的终止状态形如 $[x^0]A^{-1}(x)B(x)$ 。

3.3 二项卷积 [4]

二项卷积是这样的一个问题: 给定两个序列 $a_0, a_1, \dots, a_n$ 和 $b_0, b_1, \dots, b_n$, 求出序列 $c_i = \sum_{0 \leq j \leq i} \binom{i}{j} a_j b_{i-j}$ 。

⁶2020-2021 集训队作业, 带加强和多项式, <https://loj.ac/p/3398>

二项卷积通常的解法是把二项式系数拆成 $\frac{i!}{j!(i-j)!}$ ，然后做 $\frac{a_i}{i!}$ 和 $\frac{b_i}{i!}$ 的普通卷积（或者说，把 a, b 转为对应的 EGF）。

和之前类似的，在 $p \leq n$ 的时候我们想要避免求逆，考虑把组合数拆成 $f_p(n)$ 和 p 因子

$$\binom{i}{j} = p^{\nu_p(i!) - \nu_p(j!) - \nu_p((i-j)!)} \frac{f_p(i)}{f_p(j)f_p(i-j)}$$

记 $\hat{a}_i = \frac{a_i}{f_p(i)} \bmod p^k$ ，那么有

$$\hat{c}_i = \sum_{0 \leq j \leq i} p^{\nu_p(i!) - \nu_p(j!) - \nu_p((i-j)!)} \hat{a}_i \hat{b}_{i-j}$$

定理 2 (Kummer 定理). $\nu_p(i!) - \nu_p(j!) - \nu_p((i-j)!)$ 等于 p 进制下 i 减去 j 发生的退位次数。

由 Kummer 定理， $\nu_p(i!) - \nu_p(j!) - \nu_p((i-j)!)$ 不会超过 $d = \lfloor \log_p(n) \rfloor$ ，因此在直接使用上式不取模计算 $\hat{c}_i$ 的时候最终答案不超过 $n^2 p^{2k}$ 。

因此直接用 $O(1)$ 个 NTT 模数做 NTT 求出上式在对 p^k 取模前的值对每个 NTT 模数取模的答案，最后 CRT 合并回去即可。

因为 p 在选取的大质数模数下总是有逆元，所以可以直接做普通卷积，复杂度 $O(n \log n)$ 。

3.4 多项式指数函数⁷

多项式指数函数一般指给定一个常数项为 0 的 n 次多项式 $F(x)$ ，求出 $G(x) = \exp F(x)$ 的前 $n+1$ 项系数。

一般我们会用牛顿迭代来求解 $G(x)$ 。考虑 $G(x)$ 满足的方程 $H(G(x)) = \ln G(x) - F(x) = 0$ ，设我们已经求得前 m 项系数，即 $G_0(x) = G(x) \bmod x^m$ ，考虑如何求出 $G(x) \bmod x^{2m}$ 。

把 $H(G(x))$ 在 $G_0(x)$ 处泰勒展开得到

$$H(G(x)) = \sum_{i \geq 0} \frac{H^{(i)}(G_0(x))}{i!} (G(x) - G_0(x))^i = 0$$

因为 $x^m \mid G(x) - G_0(x)$ ，所以 $H(G(x)) \equiv H(G_0(x)) + H^{(1)}(G(x) - G_0(x)) \equiv \ln G_0(x) - F(x) + \frac{G(x) - G_0(x)}{G_0(x)} \bmod x^{2m}$ 。

⁷指数公式，<https://loj.ac/p/6737>

因此有 $G(x) \equiv G_0(x)(1 + F(x) - \ln G_0(x)) \bmod x^{2m}$ 。

在模数有小质因子的时候，求 $\ln G_0(x)$ 会出现问题。因为一般求 $\ln G_0(x)$ 的方法是求导完再积分回去，即对 $\frac{G'_0(x)}{G_0(x)}$ 积分。

首一多项式求逆可以照常进行，唯一的问题出现在积分的时候需要对系数除以次数加一，在模数有小质因子的时候这不一定可行。

但是一般实际求解 $G(x) = \exp F(x)$ 的时候，我们最终需要得到的并不是 $[x^i]G(x)$ ，而是 $i![x^i]G(x)$ ，即 $G^{(i)}(0)$ 。

因此仍然考虑牛顿迭代，但是过程中维护 $G^{(i)}(0)$ 而非 $[x^i]G(x)$ 。那么原先的求导和积分会转变为乘除 x ，普通卷积会转变为二项卷积。套用上面二项卷积的做法即可做到 $O(n \log n)$ 的复杂度。

4 总结

本文围绕模小质数幂的计数问题展开研究，通过整理经典问题的解法，总结出生成函数和数论方法在这类问题的具体应用。研究利用小质数的性质优化了一些计数问题的复杂度，并在模小质数幂组合数求值问题上提供了一种新的做法。

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、彭思进、张艺缤的指导。

感谢父母对我多年的培养与教育。

感谢汪星明老师对我的培养与教导。

感谢吴畅、董雨润、刘子渊学长对本文提供的宝贵建议。

感谢赵海鲲、朱鹏睿同学与我讨论。

感谢其他给予我帮助的老师与同学。

参考文献

- [1] 唐靖哲. 组合数求模. <https://blog.csdn.net/skywalkert/article/details/52553048>, 2020.
- [2] 唐靖哲. 组合数求模 (续). <https://blog.csdn.net/skywalkert/article/details/104681101>, 2020.
- [3] 李白天. 信息学竞赛中的生成函数计算理论框架. 国家集训队 2021 论文集, 2021.
- [4] 李白天. 任意模数二项卷积. <https://entropyincreaser.github.io/posts/arbitrary-mod-binomial-convolution>, 2020.

浅谈一类实现简易的动态树型结构信息维护方法

长郡中学 肖岱恩

摘要

动态树型结构信息维护是一类重要的数据结构问题，该问题存在两种常见的自然对数解法，分别是伸展树维护实链剖分的均摊解法和重量平衡树维护动态绝对重链剖分的严格解法，本文将介绍这两种做法及其实现方式。

1 前言

动态树型结构信息维护问题是指在支持动态断边连边的森林中，动态修改连通块和链的节点信息，同时查询动态连通块和链的信息并的相关问题，在二十世纪就得出了单次操作最坏自然对数的理论解决方案，但不易实现且常数较大，本文会从理论出发，介绍两种实现简易的动态树型结构信息维护方法。

2 约定

假定读者已经掌握伸展树、重量平衡树、重链剖分的基本概念。

3 定义

定义 3.1 (树). 定义树是一个无向连通无环简单图。

定义 3.2 (森林). 定义森林是由一棵或多棵树组成的无向无环简单图。

为了方便维护，我们需要将树视为有根树，以下定义定义在有根树上。

定义 3.3 (父亲). 定义一个节点的父亲是该节点到根路径上经过的第一个节点，根节点没有父亲。

定义 3.4 (儿子). 定义一个节点的儿子是以该节点为父亲的所有节点。

定义 3.5 (绝对重儿子). 定义**绝对重儿子**为一个节点的所有儿子中，子树大小超过其一半的儿子，以下简称**重儿子**，容易发现一个节点最多只有一个重儿子。

定义 3.6 (绝对轻儿子). 定义**绝对轻儿子**为一个节点的所有儿子中，子树大小不超过其一半的儿子，以下简称**轻儿子**，一个节点可能只有轻儿子。

定义 3.7 (重边). 定义**重边**为一个节点和其重儿子的连边。

定义 3.8 (轻边). 定义**轻边**为一个节点和其轻儿子的连边。

定义 3.9 (双半群). 定义双半群模型由两种数据类型 (D, O) 组成。

D, O 上定义了抽象的运算 $\times: O \times D \rightarrow D$;

O 上定义了抽象的运算 $\times: O \times O \rightarrow O$;

D 上定义了抽象的运算 $+: D + D \rightarrow D$;

ϵ_D 是 D 中的一个特殊的元素，称为单位元;

ϵ_O 是 O 中的一个特殊的元素，称为单位元;

这些操作满足性质:

1. 对任意 $a, b, c \in D$, 有 $a + b = b + a$, $(a + b) + c = a + (b + c)$, $a + \epsilon_D = \epsilon_D + a = a$;
2. 对任意 $u, v, w \in O$, 有 $(u \times v) \times w = u \times (v \times w)$, $u \times \epsilon_O = \epsilon_O \times u = u$;
3. 对任意 $u, v \in O$, $a, b \in D$, 有 $(u \times v) \times a = u \times (v \times a)$, $u \times (a + b) = (u \times a) + (u \times b)$, $\epsilon_O \times a = a$, $u \times \epsilon_D = \epsilon_D$ 。

4 问题引入

维护一个森林，每个节点拥有信息 d_x ，支持以下三种操作：

1. 动态加入或删除一条边，保证图依旧是森林；
2. 给定 $o \in O$ 和一个极大连通块或链的点集 S , $\forall x \in S, d_x \leftarrow o \times d_x$;
3. 给定一个极大连通块或链的点集 S , 求 $\sum_{x \in S} d_x$ 。

5 带权伸展树 (splay) 势能分析

势能是用来分析均摊复杂度的常用做法，一次操作的代价可以认为是势能的增长值加操作的运算次数，进而得出算法的均摊复杂度。

记 $F(k) = \lceil \log_2 k \rceil$ 。

定义 5.1 (带权伸展树). 定义一棵带权伸展树是一棵伸展树，每个节点带有额外的权值，对于每个节点 x ，用 $size_x$ 表示子树大小和子树内所有节点额外权值的总和。

以下用中间的儿子表示该节点所带的额外的权值，并不是真实的儿子。

定义 5.2 (势能). 定义带权伸展树上的节点 x 的势能为 $h_x = F(size_x)$ 。

定义全局总势能为所有节点的势能总和: $\varphi = \sum_x h_x$ 。

一个节点的势能可以理解为提前预支用于均摊的代价，我们可以在操作过程中加入或取出来平衡复杂度。

一次操作的势能增长值为 $\Delta\varphi = \varphi - \varphi'$ ，即新势能总和减原势能总和。

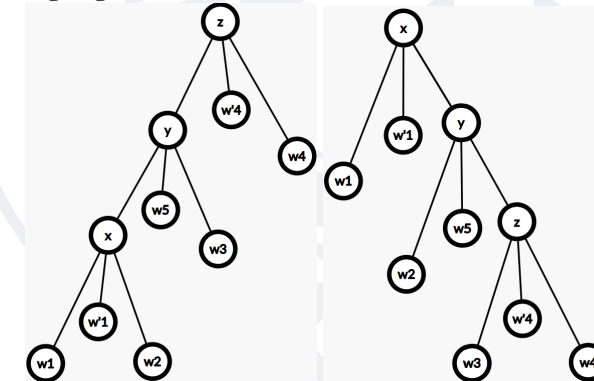
一次操作的代价是 $\Delta\varphi + cost$ ，其中 $cost$ 表示操作的运算次数。

如果能将某一系列操作的代价和控制一定的数量级，我们就能够分析得出该操作的均摊复杂度和操作代价相同。

以下进行带权伸展树操作的势能分析。

操作 5.1 (Zig-Zig 和 Zag-Zag).

Zig-Zig 操作前后图片如下：



容易发现 $(w_1, w'_1), (w_4, w'_4)$ 所属子树完全等价，所以以下用 w_x 表示 $w_x + w'_x$ 。

设 n 为操作前的 $size_z$ ，操作前势能为 $F(w_1 + w_2 + 1) + F(n - w_4 - 1) + F(n)$ ，操作后势能为 $F(w_3 + w_4 + 1) + F(n - w_1 - 1) + F(n)$ 。

势能的增长值为 $\Delta\varphi = F(w_3 + w_4 + 1) + F(n - w_1 - 1) - F(w_1 + w_2 + 1) - F(n - w_4 - 1)$ ，我们认为操作的代价是 1，所以总代价为 $\Delta\varphi + 1$ 。

注意到 $\max\{F(w_3 + w_4 + 1), F(n - w_1 - 1)\} \leq F(n)$, $F(w_1 + w_2 + 1) \leq F(n - w_4 - 1)$ ，故有 $\Delta\varphi \leq 2(F(n) - F(w_1 + w_2 + 1))$ 。

此时分两种情况：

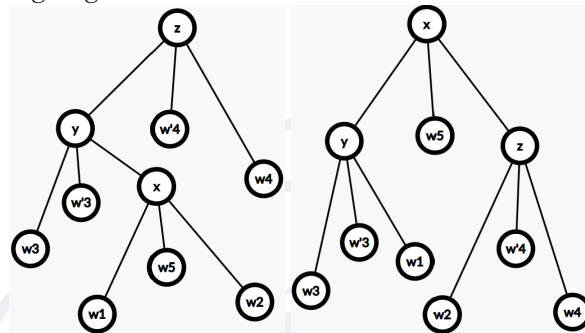
1. $w_3 + w_4 \leq w_1 + w_2$ ，则 $F(w_3 + w_4 + 1) \leq F(n) - 1$, $\Delta\varphi + 1 \leq 2(F(n) - F(w_1 + w_2 + 1)) \leq 3(F(n) - F(w_1 + w_2 + 1))$ 。

2. $w_3 + w_4 > w_1 + w_2$, 则 $F(w_1 + w_2 + 1) \leq F(n) - 1, F(n) - F(w_1 + w_2 + 1) \geq 1, \Delta\varphi + 1 \leq 2(F(n) - F(w_1 + w_2 + 1)) \leq 3(F(n) - F(w_1 + w_2 + 1))$ 。

无论是哪种情况，都有代价不超过新树势能减原树势能的 3 倍的性质，注意 Zag-Zag 操作与本操作对称，具有同样的性质。

操作 5.2 (Zig-Zag 和 Zag-Zig).

Zig-Zag 操作前后图片如下：



容易发现 $(w_3, w'_3), (w_4, w'_4)$ 所属于子树完全等价，所以以下用 w_x 表示 $w_x + w'_x$ 。

设 n 为操作前的 $size_z$ ，操作前势能为 $F(w_1 + w_2 + w_5 + 1) + F(n - w_4 - 1) + F(n)$ ，操作后势能为 $F(w_1 + w_3 + 1) + F(w_2 + w_4 + 1) + F(n)$ 。

势能的增长值为 $\Delta\varphi = F(w_1 + w_3 + 1) + F(w_2 + w_4 + 1) - F(w_1 + w_2 + w_5 + 1) - F(n - w_4 - 1)$ ，我们认为操作的代价是 1，所以总代价为 $\Delta\varphi + 1$ 。

注意到 $(w_1 + w_3 + 1) + (w_2 + w_4 + 1) \leq n, F(w_1 + w_2 + w_5 + 1) \leq F(n - w_4 - 1)$ ，故有 $F(w_1 + w_3 + 1) + F(w_2 + w_4 + 1) \leq 2F(n) - 1, \Delta\varphi + 1 \leq 2(F(n) - F(w_1 + w_2 + w_5 + 1)) \leq 3(F(n) - F(w_1 + w_2 + w_5 + 1))$ 。

也即存在代价不超过新树势能减原树势能的 3 倍的性质，注意 Zag-Zig 操作与本操作对称，具有同样的性质。

操作 5.3 (Zig 和 Zag). 这是每次伸展操作的最后一步才可能出现的操作，显然其代价不超过新树势能减原树势能的 3 倍再加一，这说明伸展操作的次数本身并不能利用势能进行均摊。

以上我们说明了带权伸展树每次伸展操作的代价为结束势能减初始势能的 3 倍再加一。

6 实链剖分

将一棵有根树剖分成若干条实链，用于维护信息，每个节点最多有一个实儿子，和实儿子相连的边是实边，和虚儿子相连的边是虚边，由实边相连的边组成的链被称为实链。

使用伸展树 (splay) 维护每一条实链，每个节点的左侧是实链上深度比自己小的点，右侧是深度比自己大的点。

每个节点会记录 splay 上的父亲，splay 根节点的父亲是树上该实链链顶的父亲。

为了维护信息或改变树的形态，需要进行以下关键操作：

操作 6.1 (splice(x)). 表示将 x 所在实链的链顶变为实儿子。

分为以下三个步骤：

1. 对父亲进行 splay 操作；
2. 断掉父亲在 splay 上的右儿子（即当前的实儿子）；
3. 将父亲的右儿子变为自己所在的 splay 的根节点。

实现时步骤 2,3 可以合并为直接更改父亲的右儿子。

容易发现，步骤 2 只会减少势能，所以不用考虑其对势能的影响。

操作 6.2 (access(x)). 表示将 x 到根的路径变成一条实链。

我们需要先将 x 的实儿子断掉，然后不断进行 splice(x)，直到 x 和根节点在同一条实链为止。

考虑分析每次 splice(x) 时步骤 1 的复杂度。

我们利用带权伸展树势能进行分析，每一个节点带有其虚子树大小之和的权重，发现我们伸展前后的子树带权大小总是不断变大，形如 $\Delta\varphi = h(x_2) - h(x_1) + h(x_4) - h(x_3) + \dots + h(x_{2k}) - h(x_{2k-1})$ ，其中 $h(x_i) \leq h(x_{i+1})$ ，故有 $\Delta\varphi \leq h(x_{2k}) - h(x_1)$ ，即 $O(\log n)$ ，步骤 3 的分析类似。

但是我们需要考虑伸展或更改儿子自带的 $O(1)$ 常数的影响，也就是代价还需要加上 splice 的次数。

考虑给每条虚重边（既是虚边，又是重边）分配 1 点势能。

对于一次 splice 操作，如果打通的实边是重边，那么我们可以获得原本虚重边的势能，没有代价；否则，我们最多会进行 $O(\log n)$ 次付出 $O(1)$ 的代价。

综上所述，一次 access 操作的代价不超过 $O(\log n)$ ，即复杂度是 $O(\log n)$ 的。

我们可以通过 access 操作完成所有树形态变化以及链信息的维护，所有操作的复杂度都是均摊 $O(\log n)$ 的，包括连边、断边、换根、修改、查询等。

7 维护子树的方法

使用仅由叶子代表信息的伸展树 (splay-leafy-tree, SLT) 来维护虚儿子^[1]，对于一个节点，其所有虚儿子可以使用 SLT 进行信息维护，此时会产生若干用于合并的节点，这些结

点依旧带有 $F(size_x)$ 的势能，然后一个节点除了左右儿子之外，还具有一个中儿子表示其所有虚儿子构成的 SLT 的根节点。

我们需要对之前的 splice 操作进行分析，分为以下两种情况：

1. 父亲具有实儿子（右儿子）

此时让右儿子代替自身（新的实儿子）在 SLT 上的位置，伸展 SLT 上的父亲。

2. 父亲不具有实儿子（右儿子）

此时需要将 SLT 上自己的父亲删除，自己的兄弟代替其位置，然后对自己的爷爷（二级祖先）进行伸展。

这两种情况对势能的影响不超过 $2(h(fa_x) - h(x))$ ，和之前 access 的分析一样，具有子树带权大小不断增大的性质，复杂度是 $O(\log n)$ 。

对于 access 开始的断掉当前实儿子操作，只需要将实儿子插入自己的中儿子即可，显然对势能的影响不超过 $O(\log n)$ ，复杂度也是 $O(\log n)$ 。

以上我们分析了 access 的时间复杂度为均摊 $O(\log n)$ 的。

信息维护方法如下：

1. 对于每个实链 splay 上的节点，我们维护 $d1, d2, o1, o2$ ，分别表示其子树内的链和、虚子树和、链懒标记、虚子树懒标记；
2. 对于每个虚子树 SLT 上的节点，我们维护 $d1, o1$ ，分别表示子树和、子树懒标记。

于是我们成功地在 $O((n + q) \log n)$ 的均摊时间复杂度解决了提出的问题。

8 带权重平衡树

使用仅由叶子代表信息的带权重平衡树（weight-balanced-leafy-tree, WBLT^[2]）来维护实链和虚儿子信息。

记 ls_x 为节点 x 的左儿子， rs_x 为节点 x 的右儿子。

这里简要介绍一下合并两棵 WBLT L, R 的过程，不妨设 $size_L \leq size_R$ ，否则对称处理，分为以下几种情况：

1. $\frac{size_R}{size_L + size_R} \leq \alpha$ 或 R 是一个叶子节点：此时直接新建节点，将两棵树串起来；
2. $\frac{size_{rs_R}}{size_L + size_R} \geq 1 - \alpha$ 或 ls_R 是一个叶子节点：此时递归合并 L 和 ls_R ，然后返回 R ；
3. 否则，递归合并 $(L, ls_{ls_R}), (rs_{ls_R}, rs_R)$ ，然后串起来。

进行少量分类讨论可以说明合并的时间复杂度是 $O(\log(size_R) - \log(size_L) + 1)$ 的，并且所有节点都具有向上跳 3 步 $size$ 至少变大 $\frac{1}{\alpha}$ 倍的性质。

一般来说， α 可以取值 0.65 ~ 0.85 左右，效率与数据有关，具有一定常数。

9 动态绝对重链剖分

容易发现，直接使用 WBLT 代替 splay 和 SLT 是可行的。

这时我们依旧会用到 splice 次数的均摊性质，当然如果遇到类似“THUPC2024 初赛 种苹果”这样的题目，只需要具有重量平衡性质，我们可以这样做。

考虑进行绝对重链剖分^[3]来保证 splice 的次数。

绝对重链剖分是一种实链剖分，满足任意操作结束之后，所有重边都是实边（如果一个节点没有重儿子，实边可能是轻边或者没有实边）。

在一次 access 操作之后，我们成功地进行了信息的修改查询，然而此时很可能不满足绝对重链剖分的性质。

考虑操作影响到的 $O(1)$ 条实链，如果对于一个非链底的节点，其具有重儿子且不是当前的实儿子，我们需要将当前实边断掉，在他的中儿子中找到可能存在的重儿子，然后合并。

对于维护这条实链的 WBLT，我们在上面二分，找到第一个大小超过一半的位置，此处断开的边就是从上往下的第一条轻实边（既是轻边，又是实边）。

我们先将维护实链的树从此处分裂，然后查询断开边的父亲的中儿子子树，扫描前 $O(1)$ 层节点，看能否找到一个叶子节点，大小超过了原树上父亲的子树大小的一半。

如果能找到，说明这个叶子节点所对应的儿子就是重儿子，从中儿子子树中切开，分裂成 $O(1)$ 个子树，和原来的轻实儿子合并，这里合并多个子树的方法可以通过不断合并当前最小的两个子树来实现，因为子树个数是 $O(1)$ 的，所以使用优先队列或者排序后利用单调性来查找最小值都可以。

如果没找到，直接将轻实儿子所在树和中儿子合并。

设父亲所在树为 x ，轻实儿子所在树为 y ，重儿子所在树为 z 。

考虑和重儿子的实链合并的复杂度，此时的代价为 $O(|\log(size_x) - \log(size_z)|)$ 。

因为 $size_y \leq \min\{size_x, size_z\}$ ，所以单次的代价不超过 $O(\log(\max\{size_x, size_z\}) - \log(size_y))$ ，我们发现每次新的 $size_x, size_z$ 都不会超过原来的 $size_y$ ，所以沿用 6.2 的分析，可以得出总代价不会超过 $O(\log n)$ 。

当然我们还需要处理链底的重儿子，只需要在上面的操作结束之后，扫描链底节点中儿子的前 $O(1)$ 层，看能不能找到重儿子，如果找到了就将其分裂出来，和自己的实链合并即可。

这样可以做到单次操作严格 $O(\log n)$ 的时间复杂度，对虚子树合并的部分操作复杂度分析较为复杂，这里不再过多讨论，容易发现这种方法也能够做到普通（即非叶子节点总是存在重儿子）的动态重链剖分，但是那样分类讨论较多。

因为这种方法具有重量平衡的特点，可以维护类似“CTS2022 WBTT”的树形块状信息，并且易于实现。

10 总结

本文对动态树型结构信息维护问题提出了两种自然对数解决方法，通过实链剖分和动态绝对重链剖分分别得出了均摊正确和严格正确的维护方式，且均能在有限的时间内得出实现。

然而，严格正确的做法的实现方式依旧较为繁杂且常数较大，希望本文可以起到抛砖引玉的作用，让读者更进一步地得出更优秀的解决方法。

11 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢家人对我的陪伴与支持。

感谢长郡中学陈奕哲老师的关心与指导。

感谢杨鑫和同学为本文验稿并提出建议。

感谢其他给予我帮助的老师与同学。

参考文献

- [1] F7487. Satt(self-adjusting top tree) 学习笔记. <https://f7487.blog.uoj.ac/blog/7490>, 2021.
- [2] 王思齐. Leafy tree 及其实现的加权平衡树. 2018 年信息学奥林匹克中国国家队候选队论文集, 2018.
- [3] zx2003. Weight balanced top tree. <https://zx2003.blog.uoj.ac/blog/7884>, 2022.

浅谈一类图上邻域相关问题

焦作市第一中学 焦思源

摘要

本文主要通过一类经典的图上邻域维护问题，深入分析得到了更本质的一个做法，之后又从另一方面对该问题进行了拓展，并通过相关例题具体介绍了一些应用。

1 前言与约定

数据结构理论在序列上、树上都有了很深入的研究与考察，在一些具有特殊性质的图，如仙人掌图，广义串并联图上也有通过向树进行转化形成的做法。但对于一般图上的问题，由于结构的复杂通常难以快速维护。本文从一个经典的问题引入，通过分析特殊情况下的做法得到了一些更加深入的理解。

本文中图定义为无重边无自环的图，用 $G = (V, E)$ 来表示一个图 G 的点集合为 V ，边集合为 E 。

一个结点的邻域定义为与其直接相连的点构成的集合，记作 $R(x)$ 。

一个结点的度数定义为其邻域大小，记作 $\deg_x$ 。

2 从一类经典问题引入

例题 1 (项目管理¹)。给定一个 n 个点 m 条边的无向连通图，有 q 次操作，操作有两种：

1. 给结点 x 的权值加上 v 。
2. 查询所有和结点 x 有条边相连的结点的权值之和。

解法 该问题的经典做法是对度数根号分治，取阈值 B ，以下将 $\deg_i \leq B$ 的点记作轻点， $\deg_i > B$ 的点记作重点。因为所有点的度数之和为 $2m$ ，故图上的重点个数不超过 $\lfloor \frac{2m}{B} \rfloor$ 。

查询一个点时：

1. 如果查询的点 x 是轻点，那么直接枚举 x 的邻域进行求和，时间复杂度 $O(B)$ 。

¹ 题目来源：HDU 4858

2. 如果查询的点 x 是重点, 那么我们考虑维护 s_x 表示重点 x 的邻域和, 在进行修改操作时枚举所有与其相邻的重点 y 并修改其 s_y , 时间复杂度 $O(\frac{m}{B})$ 。

结合上述做法, 我们取 $B = \sqrt{2m}$ 可以得到时间复杂度为 $O(q\sqrt{m})$ 的做法。

■

2.1 一类特殊图上的做法

注意到上述做法只和图的总边数有关, 与图的具体形态无关, 但是在一些特殊图上却存在更优复杂度的做法。

2.1.1 树

考虑当我们把上面的问题变成树上, 我们考虑如下做法: 以任意结点为根求出每个结点的父亲结点, 记作 fa_x 。我们对于每个结点维护其儿子结点的权值和 s_x , 那么修改操作只会影响 s_{fa_x} , 查询操作只需要在 s_x 的基础上加上 fa_x 的权值即可。

该做法的时间复杂度为 $O(n+q)$ 。

2.1.2 仙人掌

定义 2.1 (边仙人掌). 定义无向连通图是边仙人掌当且仅当每条边最多在一个简单环里。

定义 2.2 (点仙人掌). 定义无向连通图是点仙人掌当且仅当每个点最多在一个简单环里。

我们将仙人掌图上的简单环使用 tarjan 算法缩点, 可以证明形成了一棵树。这样修改操作时与 x 在不同环上邻域内的点用树的方法维护, 同环上的邻域内的点最多只有 2 个, 直接查询即可。

时间复杂度依旧为 $O(n+m+q)$ 。

2.1.3 广义串并联图

定义 2.3 (广义串并联图). 定义无向连通图是广义串并联图当且仅当其不存在与 K_4 同胚的子图, 即不存在四个点 a, b, c, d 使得这些点之间存在六条除顶点外不相交的路径。

广义串并联图可以通过以下的收缩操作来缩小图的规模。

定义 2.4. 删一度点操作指将图中度数为 1 的点 x 删除。

定义 2.5. 缩二度点操作对于图中度数为 2 的点 x , 如果其连接的点分别是 y, z , 那么将这两条边删去并添加新边 (y, z) 。

定义 2.6. 叠合重边操作对于图中连接同一对顶点的多条边, 将其缩成一条边。

引理 2.1. 一个广义串并联图可以通过上述三种操作缩成一个点。

证明可见吴作同 2019 年的候选队论文《〈公园〉命题报告》。上述缩点过程同样可以建出一个树结构, 对树结构使用上述算法即可解决, 在此不多描述。

时间复杂度依旧为 $O(n+m+q)$ 。

上面三种特殊图都可以做到 $O(n+m+q)$ 的复杂度, 引导我们考虑这些做法有什么共同点。

3 稀疏图上的做法

3.1 图的密度

定义 3.1 (图的密度). 对于一个无向连通图 $G = (V, E)$, 定义其密度 $\rho(G) = \frac{|E|}{|V|}$ 。

定义 3.2 (最大密度子图). 对于一个无向连通图 $G = (V, E)$, 定义其最大密度子图为一个子图 $G' = (V', E')$ 使得 G' 的密度最大。

定义 3.3 (图的最大密度). 对于一个无向连通图 $G = (V, E)$, 定义其最大密度为最大密度子图的密度, 记为 $\rho_{\max}(G)$ 。

引理 3.1. 对于一个图 $G = (V, E)$, 一定存在一个最大密度子图 $G' = (V', E')$ 使得 G' 连通且 G' 是 V' 的导出子图。

证明. 若 G' 不是 V' 的导出子图, 则将导出子图中未选的边选入 E' 一定会使得 $|E'|$ 增大, 即密度增大。若 G' 可以划分成若极大干连通子图 $G_1, G_2, \dots, G_k$, 不妨设 $\rho(G_1) \leq \rho(G_2) \leq \dots \leq \rho(G_k)$, 那么容易证明 $\rho(G_k) \geq \rho(G')$, 故选择 G_k 依旧是一个最大密度子图。□

我们以下认为最大密度子图 $G' = (V', E')$ 满足 G' 连通且 G' 是 V' 的导出子图。我们看一下之前的三种特殊图的最大密度。

3.1.1 树

树的任意非空连通子图的密度均为 $\frac{|V'|-1}{|V'|} < 1$, 故树的最大密度 < 1 。

3.1.2 仙人掌

由于仙人掌图有不超过 n 个简单环, 故仙人掌图的边数 $< 2n$, 其密度 < 2 , 因为仙人掌图的最大密度子图还是仙人掌图, 故仙人掌图的最大密度 < 2 。

3.1.3 广义串并联图

我们可以归纳证明广义串并联图的密度也 < 2 ：

只有一个点时，密度为 0。

使用删一度点操作后，点数和边数都减少一，故如果操作之后密度 < 2 ，那么原本也小于 < 2 。

使用缩二度点操作之后，点数减少 1，如果因此产生了重边，边数最多减少 2，同样会保持密度 < 2 的性质。

故一个广义串并联图的密度总是 < 2 。

这启发我们从图的最大密度角度考虑原问题。

3.2 基于最大密度的做法

引理 3.2. 对于一个最大密度为 $\rho_{\max}(G)$ 的图 G ，其任意子图内都存在至少一个度数不超过 $2\rho_{\max}(G)$ 的结点。

证明. 我们考虑反证法，如果一个子图 (V', E') 中所有结点的度数均 $> 2\rho_{\max}(G)$ ，那么该图的总度数 $> 2\rho_{\max}(G)|V'|$ ，边数 $> \frac{2\rho_{\max}(G)|V'|}{2} = \rho_{\max}(G)|V'| \geq |E'|$ ，矛盾。 $\square$

需要注意的一点是上述条件并不是充要的。

以下我们记 $d_{\min}(G)$ 表示其所有非空子图的最小度数的最大值，那么可以得出 $\rho_{\max}(G) \leq d_{\min}(G) \leq 2\rho_{\max}(G)$ 。

引理 3.3. 对于图 $G = (V, E)$ ， $d_{\min}(G) \leq r$ 的充要条件为，每次删除 G 中度数 $\leq r$ 的节点，可以将 G 删空。

证明. 必要性：因为对于 G 的任意非空子图 G' 都存在一个度数 $\leq d_{\min}(G)$ 的结点，故总是可以找到度数 $\leq r$ 的结点将其删去，然后归纳。

充分性：考虑反证，如果 $d_{\min}(G) > r$ 即存在一个非空子图 G' ，满足 G' 内所有点的度数均 $> r$ ，删除 G' 外的点不影响这些点内部的度数，故那么这些点我们无论如何也无法删除。 $\square$

我们通过如下的一道题介绍基于上述结论的一个做法。

例题 2 (The Last Cumulonimbus Cloud²). 给定一个 $d_{\min}(G) \leq 10$ 的图，支持两种操作：

1. 给结点 x 的权值加上 v 。
2. 查询所有和结点 x 有边相连的结点的权值之和。

²题目来源：2024 西安邀请赛，笔者命制的一道测试题

根据上述引理，该图可以通过每次删掉度数不超过 10 的点将图删空，这样可以得到每个点的删除顺序。我们考虑对于每条边，把它定向为从先删除的点指向后删除的点，那么对于这个有向图来说，每个点的出边个数均不超过 $d_{\min}(G)$ 。

我们对于每个结点，维护其所有入边连向的点的权值和 s_x ：

1. 对于修改操作，只需要修改 x 本身的权值，以及 x 出边连向的点 y 的 s_y 。

1. 对于查询操作，只需要在 s_x 的基础上，加上 x 出边连向的的权值和即可。

两部分操作复杂度均为 $O(d_{\min}(G))$ ，故复杂度不超过 $O(n + m + qd_{\min}(G))$ 。

因为 $\rho_{\max}(G) \leq d_{\min}(G) \leq 2\rho_{\max}(G)$ ，所以我们就得到了一个时间复杂度为 $O(n + m + q\rho_{\max}(G))$ 的简洁在线做法。

3.3 允许离线处理时的更优做法

我们进一步考虑上述做法的本质，其核心思路为：给每条边定向，每个点仅维护其入边的信息。

那么如果允许离线，我们可以求出 1,2 操作作用在结点 x 上的总次数 c_x ，那么一条边定向产生的代价就是其入边方向上点的操作次数 c_x ，故最优情况下我们让每条边从 c 小的一端指向 c 大的一端，这样子就可以得到该做法下的最优复杂度，总代价为 $\sum_{(x,y) \in E} \min(c_x, c_y)$ 。

我们接下来分析该做法的最坏复杂度，该问题为将 q 分配给每个点的 c 使得 $\sum_{(x,y) \in E} \min(c_x, c_y)$ 最大。这里先认为每个点的权值 c 可以是任意非负实数，要求整数的情况是近乎相同的。

假设最优情况下，被分配了权值的点集合是 V_1 ，我们考虑 V_1 的导出子图 $G_1 = (V_1, E_1)$ 。

引理 3.4. $\sum_{(x,y) \in E} \min(c_x, c_y)$ 最大时， V_1 内部所有点的权值均相等，即每个点的 c 都等于 $\frac{q}{|V_1|}$ 。

证明. 否则，我们可以用如下方式计算总代价：设当前权值的最小值为 v ，将 $|E_1|v$ 加入总代价里，然后将所有结点的 c 减掉 v ，将 $v > 0$ 的结点构成的导出子图 $G_2 = (V_2, E_2)$ 进行递归计算。

当所有结点的权值不全相等时，其至少会递归一次，我们设前两次操作的最小值分别是 v_1, v_2 ，点数分别是 n_1, n_2 ，边数分别是 m_1, m_2 。

- 当 $\frac{m_1}{n_1} > \frac{m_2}{n_2}$ 时，我们考虑将两次操作变为一次操作，将可以分配的总权值 $n_1v_1 + n_2v_2$ 平均分配到这 n_1 个点上，此时新的 $v = \frac{n_1v_1 + n_2v_2}{n_1} = v_1 + \frac{n_2}{n_1}v_2$ ，新的贡献为 $(v_1 + \frac{n_2}{n_1}v_2)m_1 = v_1m_1 + v_2\frac{n_2m_1}{n_1}$ ，因为 $\frac{n_2m_1}{n_1} > m_2$ 故一定更优。
- 当 $\frac{m_1}{n_1} \leq \frac{m_2}{n_2}$ 时，我们考虑将 G_2 作为最终分配了权值的点集，即将 G_1 中不在 G_2 内的点的权值均分给 G_2 。此时新的 $v = \frac{n_1v_1 + n_2v_2}{n_2} = \frac{n_1}{n_2}v_1 + v_2$ ，新的贡献为 $(\frac{n_1}{n_2}v_1 + v_2)m_2 = v_2m_2 + v_1\frac{n_1m_2}{n_2}$ ，因为 $\frac{n_1m_2}{n_2} \geq m_1$ 故一定不劣。

综上，当至少有两种不同权值时我们一定可以将权值数减少，归纳下去即可证明最优情况下所有结点的权值均相等。□

引理 3.5. $\sum_{(x,y) \in E} \min(c_x, c_y)$ 最大时， G_1 为原图的最大密度子图。

证明. 因为所有结点的 c_x 均相等，那么代价即可表示为 $\frac{q}{|V|}|E_1| = q\rho(G_1)$ ，故只需要最大化 $\rho(G_1)$ 即可最大化总代价。□

因此该做法的最终复杂度上界为 $O(n + m + q\rho_{\max}(G))$ ，虽然和在线的做法相同，但是因为很难卡满且常数极小，所以实际运行效率是要高于在线算法的。

3.4 稠密图的情况

我们之前讨论了 $\rho_{\max}(G)$ 较小，即稀疏图的情况，接下来我们考虑以下该做法在稠密图下的情形。

引理 3.6. 一个图 $G = (V, E)$ 的 $d_{\min}(G) \leq \sqrt{2m}$ 。

这直接说明了该做法任何情况下都不劣于传统的做法，不过我们也可以很容易构造一个图卡到上界，故该做法在稠密图也不能表现出更好的情况。

3.5 有向图的情况

这里有向图的情况指每次修改时限定仅对 x 的入边连向的点或者出边连向的点进行操作。注意到我们的算法里的定向本质只是把每个点的邻边进行了划分，那么有向图的情况我们可以把边看成无向边，之后在枚举到一条边的时候判断是不是我们想要的方向即可。

4 另一个角度的拓展

事实上，本文开头提到的经典问题可以进一步加强。

4.1 一道例题

例题 3 (雨旸风簋³)。给定一个 n 个点 m 条边的无向图 G ，每个点有权值 a_x ，支持两种操作：

1. l, r, v 对于 $i \in [l, r]$ ，将 $x \in R(i)$ 的 a_x 加上 v 。

³题目来源：笔者命制的一道测试题

2. l, r 查询 $\sum_{i \in [l, r]} \sum_{x \in R(i)} a_x$

在本题中，传统的根号分治做法不再可行，我们需要进一步观察问题的本质。

4.1.1 转化为序列问题

定义 4.1 (邻域序列). 我们定义图 G 的邻域序列 $seq(G)$ 为按照如下方式生成的点序列：枚举 $i = 1 \rightarrow n$ ， $\forall j \in R(i)$ ，将 j 加入 $seq(G)$ 的末尾。

因为每条边会在端点各被算一次，故 $|seq(G)| = 2m$ 。

由于我们是按照编号顺序加入 $seq(G)$ 的，故修改查询的区间 l, r 内所有点的邻域也可以表示成 $seq(G)$ 上的一个区间 l', r' ，那么我们可以转化为如下的问题。

例题 4. 给定一个长度为 $2m$ ，值域为 $[1, n]$ 的序列 s ，对于每个 $i \in [1, n]$ 的点维护权值 a_i ，支持两种操作。

1. l, r, v 对于 $i \in [l, r]$ ，将 a_{s_i} 加上 v 。

2. l, r 查询 $\sum_{i \in [l, r]} a_{s_i}$ 。

该问题显然不弱于原问题，我们不妨解决这个问题。

4.1.2 序列分块

取块长 B ，把邻域序列 s 每 B 个位置分成一个块，对于一个修改或查询的区间 $[l, r]$ ，我们称完整包含于 $[l, r]$ 的块为整块，有交的不被包含的块为散块，那么一次操作涉及的整块个数 $\leq \frac{2m}{B}$ ，散块大小不超过 $2B$ 。

设一共分成了 C 个块。

我们将修改操作对询问操作产生的贡献分开计算。

- 修改的散块对询问的散块的贡献

因为涉及的散块大小是 $O(B)$ 的，暴力做复杂度就是 $O(B)$ 的。

- 修改的整块对询问的贡献

预处理 $f_{i,j}$ 表示将序列的第 i 个块内所有点的权值都 $+1$ 后，序列里前 j 个点的权值和，并在修改时维护每个块 i 总共被加的数之和 tag_i ，那么对于一个询问操作 $[l, r]$ 来说由于贡献的可减性，该部分的贡献为：

$$\sum_{i=1}^C tag_i (f_{i,r} - f_{i,l-1})。$$

预处理的复杂度为 $O(\frac{m^2}{B})$ ，修改和查询的复杂度都是 $O(\frac{m}{B})$ 。

- 修改的散块对询问的整块的贡献

和上一情况类似地预处理 $g_{i,j}$ 表示将序列里前 i 个点的权值都 +1 后，序列里第 j 个块内所有点的权值和。

在修改时枚举所有块 i ，如果散块区间为 l', r' ，则记录对第 i 个块的贡献 $v(g_{r',i} - g_{l'-1,i})$ ，询问时枚举所有块求和即可，复杂度同上。

取 $B = \sqrt{2m}$ 可以得到时间复杂度为 $O((m+q)\sqrt{m})$ ，空间复杂度为 $O(m\sqrt{m})$ 的做法。

进一步的，离线询问对于每个块分别处理可以将空间复杂度降为线性。

据此我们在原本做法相同的时间复杂度内解决了这个加强的问题。

4.1.3 矩乘归约

首先对上述问题稍作修改，考虑下面的问题：

例题 5. 给定 n 个点 m 条边的有向图 G_1, G_2 ，每个点有权值 a_x ，记 $R_1(i), R_2(i)$ 分别表示 G_1, G_2 上结点 i 的出边连向的点构成的集合，支持两种操作：

1. l, r, v 对于 $i \in [l, r]$ ，将 $x \in R_1(i)$ 的 a_x 加上 v 。

2. l, r 查询 $\sum_{i \in [l, r]} \sum_{x \in R_2(i)} a_x$

在有向图上之前的分块做法依旧是成立的，而两个图只需要将 f, g 数组的含义稍作修改即可，在此不多描述，我们以下默认该问题可以在 $O((m+q)\sqrt{m})$ 的复杂度内解决。

取 $n = \sqrt{m}$ ，考虑大小为 $n \times n$ 的 01 矩阵 A, B ，我们要计算 $C = AB$ 。

如果 $A_{i,j} = 1$ 就在 G_1 中连有向边 $i \rightarrow j$ ，如果 $B_{i,j} = 1$ 就在 G_2 中连有向边 $j \rightarrow i$ ，那么 $C_{i,j} = \sum_{k=1}^n A_{i,k} B_{k,j}$ 可以通过这样的方式求出：先使用 1 操作给区间 $[i, i]$ 进行修改，之后查询区间 $[j, j]$ 即可得出上式，之后撤销操作即可。

因此该问题的复杂度不低于解决矩阵乘法的时间，故我们的做法是较为满意的。

4.2 另一道例题

例题 6 (旅行⁴). 给定一个 n 个点 m 条边的无向图 $G = (V, E)$ ，有 q 次查询操作，每次给出一个区间 $[l, r]$ ，请计算有多少长度为 k 的序列 $x_1, x_2, \dots, x_k$ ，满足：

- $x_1, x_k \in [l, r]$ 。
- $\forall i \in [1, k-1], (x_i, x_{i+1}) \in E$ 。

其中 $k \in [2, 4]$ 是一个常数。

⁴题目来源：笔者命制的一道测试题

4.2.1 $k = 2$

此时等价于询问有多少边的两端点都在区间 $[l, r]$ 内，这是一个二维数点问题，可以离线扫描线，用树状数组维护。

4.2.2 $k = 3$

一个直接的想法是借用上一题，转化为将区间内的点的邻域 +1，查询区间内的点的邻域和，直接用上一题的分块做法就可以做到 $O((m+q)\sqrt{m})$ 的做法。

但为了后续的拓展，这里我们换一个做法，考虑离线使用莫队解决这个问题，在移动端点时将该点邻域里的点权计入答案，再给这些点点权 +1。

但是因为莫队算法可能会反复加入一个点，如果该点的度数较大则会破坏复杂度，故我们考虑对邻域序列 $seq(G)$ 使用莫队算法，这样就变成单点加，单点查的问题，使用普通莫队的复杂度分析可以得到 $O(m\sqrt{q})$ 的做法。

4.3 $k = 4$

考虑二次离线莫队。

上述莫队做法中移动一个点的代价实际上就是区间内对一个单点的贡献，将其差分为两个前缀的贡献。那么现在的问题形如，在 $seq(G)$ 上， $O(m)$ 次将一个点的邻域内的点权 +1， $O(m\sqrt{q})$ 次查询一个点的点权。

延续最开始的经典问题的解法，对度数分治，取阈值 B ，当询问的点度数 $\leq B$ 时直接枚举出边查询，否则在修改时计算贡献。

视 m, q 同阶，取 $B = m^{\frac{1}{4}}$ 可以得到 $O(m^{\frac{7}{4}})$ 复杂度的做法。

使用稀疏图上的定向做法也可以做到同等复杂度。

5 总结

本文从一个经典的图上邻域问题出发，研究了其在稀疏图上的更优做法，并说明了该做法不劣于经典的做法。之后又从另一个角度对该问题进行了加强，得到了一个更优的做法。不过因为邻域并未很好的利用图上的各种结构，因此我们的最终还是转化为可以维护的序列问题进行解决。

希望能对读者有所启发。

致谢

感谢中国计算机学会提供交流和学习的平台。

感谢父母对我的培养和教育。

感谢刘小刚老师的教导和焦作一中同学们的帮助。

感谢王向前，支扬帆学长，周泽坤，吴彦儒同学为本文审稿。

参考文献

- [1] 吴作同. 《〈公园〉命题报告》. IOI2019 中国国家候选队论文集.



浅谈一类基于线段树分治的信息维护算法

福建师范大学附属中学 黄佳旭

摘要

本文简要介绍了一类通过线段树分治与其他数据结构相结合，用于维护连通块信息的离线算法，以及离线维护可持久化信息的算法。

1 线段树分治简介

线段树分治通常用于离线维护一系列操作。在处理“插入信息”相对简单而“删除信息”较为困难的问题时，通过应用时间线段树分治的方法，可以有效避免执行“删除信息”的操作。

例题 1.1 (「离线可过」动态图连通性¹). 维护一张 n 个点的简单无向图，有 m 次操作，支持加边删边，询问两点间连通性。

我们对 m 次操作建立时间线段树，每条边的存在时间是一段区间。将这些区间挂到时间线段树上，然后遍历整棵时间线段树，同时用可撤销并查集维护连通性。这样我们一共有 $O(m \log m)$ 次加边操作，每次调用并查集的时间复杂度为 $O(\log n)$ ，因此总时间复杂度为 $O(m \log m \log n)$ 。此外，我们将 $O(m \log m)$ 次加边操作存到了线段树的节点上，因此空间复杂度为 $O(m \log m)$ 。

2 连通块整体信息的维护

例题 2.1 (Communication Towers²). 有一张 n 个点 m 条边的无向图，第 i 个点只在第 L_i 到 R_i 这段时间出现。你需要对于每个点 i ，判断是否存在一个时刻 x ，使得 i 和 1 联通。 $n \leq 2 \times 10^5, m \leq 4 \times 10^5, 1 \leq L_i \leq R_i \leq 2 \times 10^5$ 。

上述问题可以转化为：在每个时刻，对节点 1 所在的连通块内的所有点的点权 +1，最后求出有哪些点的点权大于 0。

¹来源：<https://loj.ac/p/121>

²来源：<https://codeforces.com/problemset/problem/1814/F>

2.1 算法 1

考虑在可撤销并查集中使用懒标记。在将 y 集合合并到 x 集合时，我们记录 x 集合的加法标记为 tag_1 ，在撤销合并操作时，设当前 x 集合的加法标记为 tag_2 ，则我们下传到 y 集合的加法标记应为 $tag_2 - tag_1$ 。

然而，该算法具有一定的局限性，仅适用于信息存在逆元的情况。下面介绍一种适用范围更广的算法。

例题 2.2. 给定一个包含 n 个点和 m 条边的无向图，其中第 i 个点的点权为 a_i 。接下来 q 个时刻，每个时刻有一个操作。

第 i 条边连接 x_i, y_i ，仅在时刻 $[l_i, r_i]$ 存在 ($1 \leq l_i \leq r_i \leq q$)。

每个时刻的操作是以下之一：

1. 给出 x ，询问当前时刻 x 所在连通块的所有点中点权的最大值。
2. 给出 x, w ，将当前时刻 x 所在连通块的所有点的点权对 w 取 $\max$ 。

2.2 算法 2

由于可撤销并查集具有“栈”的后进先出结构，那么我们单独考虑一个以 x 为代表元的集合的操作，与 x 合并的集合按照合并顺序构成了一个序列，将集合 y 合并到集合 x 时，将 y 加入集合序列的末尾。

可以发现，我们要支持的就是以下操作：

1. 在序列末尾插入一个数。
2. 删除序列末尾的数。
3. 对序列中的所有数进行修改（打上取 $\max$ 的标记）。
4. 查询序列末尾的标记。
5. 查询整个序列的最大值。

考虑用“栈”的结构来维护。对于栈中每个元素，维护栈底到该元素的“前缀”最大值以及懒标记。

对于插入操作，将栈顶的信息和插入的信息合并，得到新的全局最大值后插入元素。

对于删除操作，将栈顶的信息弹出，同时将懒标记下传给新的栈顶，以及以弹出元素为代表元的集合的栈顶。

对于修改和查询操作，直接对栈顶元素操作即可。

可以发现，每种操作都只需要 $O(1)$ 次的信息合并。总共有 $O(m \log q + q)$ 次操作，但是每次操作都需要调用并查集定位集合，因此时间复杂度为 $O(n + (m \log q + q) \log n)$ ，空间复杂度为 $O(n + m \log q + q)$ ，与算法 1 的时空复杂度是相同的，同时避免了逆元的问题。

该算法的时间复杂度瓶颈在于并查集，因此实际运行中常数较小，多数情况下效率与大常数的单 $\log$ 算法相当。

该算法实际上维护了一个类似“Kruskal 重构树”的二叉树结构。每个集合对应了一棵二叉树，合并时新建根节点合并两个子树的信息，撤销时将根节点的标记下传给两个子树并分裂，修改和查询直接对根节点操作。

同时可以发现，如果已经知道了每条边的存在时刻信息，那么线段树分治其实是可以“在线”回答询问的。

2.3 空间复杂度的优化

我们最初将 m 次的区间加边操作拆分为 $O(m \log q)$ 个操作，并且保存在了线段树的节点上。然而，实际上可以同时将所有加边操作传入分治过程，模拟线段树的区间修改过程，将需要的操作转给子节点，同时该节点不保留。通过一些精细的实现即可做到线性空间。

一种比较简单的实现方式是，在递归到子节点时，将所有加边操作进行排序，将需要下传的加边操作排到序列前面，这样就保证了每个节点的加边操作是一段前缀。这相当于对 01 序列排序，容易做到线性。

尽管这里有 $O(m \log q)$ 次合并操作，但在不撤销的情况下， n 个点最多合并 $n-1$ 次。因此，在任意时刻，栈中元素的总数是 $O(n)$ 的。通过进行空间回收和重复利用，可以做到线性空间。

因此，可以将算法 2 的空间复杂度优化至 $O(n + m + q)$ 。

2.4 序列区间信息的维护

例题 2.3 (【模板】线段树 2³). 已知一个长为 n 的数列，你需要进行 q 次下面三种操作：

1. 将某区间每一个数乘上 x ；
2. 将某区间每一个数加上 x ；
3. 求出某区间每一个数的和。

我们可以将序列视为一条链，进行区间操作时，断开区间两端的边，操作完成后重新连接。直接应用算法 2 的时间复杂度为 $O((n + q) \log q \log n)$ 。

由于在任意时刻，连通块均为“链”，连边操作仅涉及链的端点。因此，我们可以在并查集合并的过程中，更新链的端点的代表元，从而实现 $O(1)$ 定位集合。

因此，可以将时间复杂度优化至 $O((n + q) \log q)$ ，同时空间复杂度为 $O(n + q)$ 。

³来源：<https://www.luogu.com.cn/problem/P3373>

3 单点的修改与查询

例题 3.1. 给定一个包含 n 个点和 m 条边的无向图，其中第 i 个点的点权为 a_i 。接下来 q 个时刻，每个时刻有一个操作。

第 i 条边连接 x_i, y_i ，仅在时刻 $[l_i, r_i]$ 存在 ($1 \leq l_i \leq r_i \leq q$)。

每个时刻的操作是以下之一：

1. 给出 x ，询问当前时刻 x 所在连通块的所有点中点权的最大值。
2. 给出 x, w ，将当前时刻 x 所在连通块的所有点的点权对 w 取 $\max$ 。
3. 给出 x ，询问当前时刻 x 的点权。
4. 给出 x, w ，将 x 的点权修改为 w 。

3.1 单个集合的信息维护

类似**算法 2**，对于一个集合而言，需要支持以下操作：

1. 在序列末尾插入一个数。
2. 删除序列末尾的数。
3. 对序列中的所有数进行修改（打上取 $\max$ 的标记）。
4. 查询序列中某个数的标记。
5. 修改序列中的某个数。
6. 查询整个序列的最大值。

由于存在对序列中间数的修改和查询操作，所以不能单纯使用“栈”来维护。平衡树是一种容易想到的维护方式。但考虑到仅有末尾的插入和删除操作，因此可以使用“二进制分组⁴”加“线段树”的结构来维护，在每个节点维护 $O(\log n)$ 棵线段树。

插入或删除信息时，需要进行 $O(\log n)$ 次的线段树节点的合并或分裂，每次是 $O(1)$ 的，因此往序列末尾插入或删除信息的时间复杂度为 $O(\log n)$ 。

整体修改或查询时，对于 $O(\log n)$ 棵线段树分别操作即可，时间复杂度为 $O(\log n)$ 。

单点修改或查询时，找到对应的线段树进行修改，时间复杂度为 $O(\log n)$ 。

因此，我们可以在 $O(\log n)$ 的复杂度内，维护一个集合的 6 种操作。

3.2 连通块的信息维护

合并两个集合时，需要定位集合和在集合中插入一个数，时间复杂度为 $O(\log n)$ 。

分裂两个集合时，需要在集合中单点查询标记，删除末尾的数，并将标记下载到子集合（即进行集合的全局修改），时间复杂度为 $O(\log n)$ 。

对于连通块整体修改或查询，在定位集合后，对集合进行操作，时间复杂度为 $O(\log n)$ 。

⁴二进制分组：详见许昊然 2013 年国家集训队论文《浅谈数据结构题的几个非经典解法》

单点修改或查询时，需要对目标点到并查集根节点的路径上的所有 $O(\log n)$ 个集合进行操作，将路径上的标记从根节点开始按顺序下传，每次下传标记的时间复杂度与分裂集合时相同，共 $O(\log n)$ 次，时间复杂度为 $O(\log^2 n)$ 。

修改后按顺序更新每一个集合的信息。在求出子集合的全局答案后，对父集合进行单点更新，共 $O(\log n)$ 次，时间复杂度为 $O(\log^2 n)$ 。

综上所述，总时间复杂度为 $O(n + m \log q \log n + q \log^2 n)$ 。由于在任意时刻，集合序列中的节点总数是 $O(n)$ 的，因此构建的线段树节点数也是 $O(n)$ 的。类似于**算法 2**，可以将空间复杂度优化至 $O(n + m + q)$ 。

4 可持久化信息的维护

例题 4.1. 给定一个包含 n 个点和 m 条边的无向图，其中第 i 个点的点权为 a_i 。第 i 条边连接 x_i, y_i 。接下来 q 个时刻，每个时刻有一个操作。

初始时的图为版本 0， m 条边都存在。

每个时刻的操作是以下之一，设当前是时刻 i ：

1. 给出 u, x ，新建一个版本 i 和版本 u 相同，询问版本 i 中 x 所在连通块的所有点中点权的最大值。
2. 给出 u, x, w ，新建一个版本 i 和版本 u 相同，将版本 i 中 x 所在连通块的所有点的点权对 w 取 $\max$ 。
3. 给出 u, x ，新建一个版本 i 和版本 u 相同，将版本 i 中第 x 条边的状态反转（存在变为不存在，不存在变为存在）。

4.1 问题的初步分析

考虑构建出版本树，那么每个操作的作用范围是一个子树。我们需要对加边操作进行一些分解，考虑怎么样的分解是可行的。

在序列上线段树分治时，我们将区间加边操作分解为 $O(\log q)$ 个线段树节点上的操作，每个操作的作用范围是线段树上的一个子树，或者说，这些操作的作用范围只有包含关系，而没有相交关系。因此，可以实现后进先出的可撤销结构。

与**例题 2.2** 不同的是，在构建出版本树后，每个修改操作的作用范围是一个子树，并且需要能够被撤销。由于修改操作基于某一特定时刻的图结构，难以进一步分解，因此在对加边操作进行分解时，不能与子树产生相交关系。这在很大程度上限制了加边操作的分解方式。

下面介绍一种可行的算法。

4.2 整体结构的建立

对版本树进行**重链剖分**，对每条重链按照 DFS 序建立线段树，并对每个节点的轻儿子按照 DFS 序建立线段树。

将加边操作分为以下三类：

- **重区间边**：在 x 节点上的重区间边，其作用范围是在 x 的子树内，但不在 x 的重儿子的子树内的所有节点。
- **轻区间边**：在 x 节点上的轻区间边，其作用范围是在 x 的子树内的所有节点。
- **节点边**：在 x 节点上的节点边，其作用范围仅为 x 这一个节点。

这种分类方式有助于明确加边操作的作用范围，便于后续的处理和优化。

4.3 加边操作的分解

对于每一条边，将涉及该边修改操作的点提取出来，构建虚树⁵。这部分的时间复杂度为 $O((m+q)\log q)$ ，虚树上的节点总数是 $O(m+q)$ 的，边的总数是 $O(q)$ 的。

4.3.1 虚树上点的操作分解

对于虚树上的点 x ，如果是删边操作，则无需理会。

对于加边操作，首先在 x 上加入**节点边**。

对于 x 原树上的所有子树，如果子树内存在虚树上的节点（即虚树上 x 有连向该子树内的边），则暂时不予考虑。

否则，加边操作需要对该子树生效。

对于轻儿子，在上面加入**轻区间边**。在 x 节点的轻儿子线段树上进行区间覆盖。虚树上的每条边会使覆盖区间数增加 1，但因为边的总数是 $O(q)$ 的，所以总共会进行 $O(m+q)$ 次轻儿子线段树上的区间覆盖。

对于重儿子，我们对其到**重链底端**的所有节点上加入**重区间边**来实现子树覆盖的效果。总共会进行 $O(m+q)$ 次重链线段树上的区间覆盖。

4.3.2 虚树上边的操作分解

如果虚树上存在边 (x, y) ，其中 x 是 y 在原树上的祖先，并且 x 节点是加边操作。那么设 v 是 x 原树上的儿子，并且 y 在 v 的子树内，则完成**点的操作分解**后， x 节点操作的作用范围还应包括在 v 子树但不在 y 子树内的点。

⁵虚树及其构建方法详见：<https://oi-wiki.org/graph/virtual-tree/>

接下来进行分类讨论。

如果 y 和 v 在同一条重链上，直接对 y 的父亲到 v 链上的节点加入**重区间边**。

否则，如果 y 不是重链顶端，设重链顶端是 z ，那么对 y 的父亲到 z 链上的节点加入**重区间边**，并递归解决 z 到 v 的链。

对于 y 是重链顶端的情况，设 z 为 y 的父亲。那么我们对 z 做**点的操作分解**，然后递归解决 z 到 v 的链。

递归的次数是 $O(\log q)$ 的，每次递归会进行 $O(1)$ 次线段树上的区间覆盖。

因此， $O(q)$ 条边总共会产生 $O(q\log q)$ 次线段树上的区间覆盖。

综合点上的分解操作和边上的分解操作，总共有 $O(m+q\log q)$ 次线段树上的区间覆盖，并且不重不漏的覆盖了每条边的作用范围。

4.4 处理询问

在处理完加边操作的分解后，可以采用类似于**算法 2** 的方法来处理询问。

我们维护与**算法 2** 相同的并查集结构，不同之处在于我们需要撤销修改操作，因此需要用栈保存每次对集合的修改。

我们按照如下算法流程解决以 x 为根的子树，保证 x 是重链的链顶。

- 遍历 x 节点的重链的线段树，加入相应的**重区间边**。
- 遍历到重链线段树的叶子节点 y 时，加入 y 上的**节点边**，然后完成 y 节点上的查询或修改操作，并撤销 y 上的**节点边**。
- 遍历 y 节点的轻儿子线段树，加入相应的**轻区间边**。
- 遍历到轻儿子线段树的叶子节点 z 时，递归解决以 z 为根的子树，并撤销 z 子树内的所有操作，继续轻儿子线段树的遍历。
- 完成 y 的轻儿子线段树的遍历并撤销所有操作后，继续完成 x 的重链线段树的遍历，完成后撤销所有操作。

通过以上步骤，我们解决了以 x 为根的子树内的所有询问。

4.5 复杂度分析

总共进行了 $O(m+q\log q)$ 次线段树上的区间覆盖，区间拆分到线段树的节点上就是 $O((m+q\log q)\log q)$ 次加边操作，每次加边操作需要调用并查集，因此总的时间复杂度为 $O(n+(m+q\log q)\log q\log n)$ 。

每次的加边操作会造成 $O(1)$ 次的集合信息修改，而为了撤销修改操作，需要保存修改操作的信息，因此空间复杂度为 $O(n+(m+q\log q)\log q)$ 。

5 总结

本文简要介绍了通过线段树分治与可撤销并查集相结合，维护连通块信息的一类算法，以及通过与树链剖分相结合，实现版本树上的信息分解，从而解决可持久化问题的方法。

线段树分治和可撤销数据结构的应用远不止于此。希望本文能让读者对线段树分治在离线解决问题时的功能有更深入的了解，也希望感兴趣的读者能继续研究。

6 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢王可可老师、周成老师、黄丽华老师的关心和指导。

感谢父母对我的关心和支持。

感谢所有帮助过我的同学、老师。

参考文献

- [1] 罗哲正.《从 Unknown 谈一类支持末尾插入删除的区间信息维护方法》. 2016 年信息学奥林匹克中国国家队候选队论文集, 2016.
- [2] 许昊然.《浅谈数据结构题的几个非经典解法》. 2013 年信息学奥林匹克中国国家队候选队论文集, 2013.

浅谈连续型随机变量的相关问题

长沙市长郡中学 周桓毅

摘要

本文将介绍关于连续型随机变量问题的一些常见类型，并主要介绍其中的整数差分约束类型，阐述解决该种类型已经现有的两种方法和笔者提出的分段积分多项式方法，并将着重探讨分段积分多项式方法及其本质。

目录

1 前言	2
2 定义	2
3 仅存在偏序关系的类型	2
3.1 问题的引入	3
3.2 将连续型随机变量转化为排列	4
4 整数差分约束的类型	5
4.1 标准模型	5
4.2 特殊限制	7
5 一类多项式的引入	9
5.1 积分多项式的定义	9
5.2 非负域多项式函数的加减与数乘	10
5.3 非负域可导函数的类卷积	10
5.4 非负域多项式函数的类卷积	11
6 积分多项式的应用	13
6.1 分段积分多项式	13
6.2 具体应用	13
6.3 对三种连续型随机变量解决方法的对比	16

7 积分多项式的本质	16
7.1 指数加权积分变换的引入	16
7.2 实际应用	18
8 总结	19
9 致谢	19
10 参考文献	20

1 前言

本文源于笔者在解决 PKUWC2021 D2T3 招新一题过程中发现了一个比标算要更优的做法，对此，笔者对其进行了深入研究。本文将讲述由该题所引出的连续型随机变量问题，第三、四章将通过对仅存在偏序关系的类型与整数差分约束类型两种连续型随机变量问题的常见类型，介绍信息学竞赛主流的解决连续型随机变量问题的方法，第五章将介绍笔者在解决 PKUWC2021 D2T3 招新一题过程中发现的一个多项式，笔者将其命名为积分多项式，第六章将介绍该多项式在解决一些连续型随机变量问题的一种全新的方法，第七章则探讨了积分多项式的本质，第八章则进行总结。

2 定义

定义 2.1 (连续型随机变量). 对于随机变量 X ，如果存在一个非负的可积函数 $f(x)$ ，使得对于任意实数 x ，都有 $P(X \leq x) = \int_{-\infty}^x f(t)dt$ ，则称 X 为连续性随机变量，并定义 $f(x)$ 为 X 的概率分布密度函数。

定义 2.2 (连续型随机变量的数学期望). 设连续型随机变量 X 的概率分布密度函数为 $f(x)$ ，若 $\int_{-\infty}^{\infty} tf(t)dt$ 绝对收敛，则称 $\int_{-\infty}^{\infty} tf(t)dt$ 的值为连续型随机变量 X 的数学期望。

3 仅存在偏序关系的类型

对于连续型随机变量问题的研究，考虑先从最简单的问题，仅存在偏序关系的类型入手。

为了更好的描述问题，这里先作一些简单的定义：

定义 3.1 (仅存在偏序关系). 当一些对实数序列 a 的不等关系约束中仅存在形如 $a_i < a_j$ 或 $a_i \leq a_j$ 的形式时，称这些不等关系约束仅存在偏序关系。

定义 3.2 (偏序关系生成矩阵). 对于一个长为 n 的实数序列 x ，定义其偏序关系生成矩阵 $M(x)$ 为一个 $n \times n$ 的布尔矩阵满足 $M(x)_{i,j} = [x_i < x_j]$ 。

定义 3.3 (实数的取模运算). 对于一个实数 r ，定义其对于一个正整数 m 取模的结果为一个在 $[0, m)$ 间的实数 r' 满足 $\frac{r-r'}{m}$ 为整数，不难证明 r' 是存在且唯一的，并记作 $r \bmod m = r'$ 。

3.1 问题的引入

先看一道被称为欧拉数的经典问题：

例题 3.1 (欧拉数问题¹). 给定 n ，对于每一个整数 $k \in [0, n-1]$ ，求有多少个 n 阶排列 p 满足 $\sum_{i=1}^{n-1} [p_i < p_{i+1}] = k$ ，对 998244353 取模。
 $n \leq 10^6$ 。

该问题存在多种方法，如拓扑序容斥，连续型随机变量方法等，这里着重介绍第二种。

引理 3.1. 对于一个定义域为 $n \times n$ 的 01 矩阵的权值函数 W ，对于一个有 n 个元素的每个元素在 $(0, 1)$ 之间均匀随机的实数序列 x ，则 $W(M(x))$ 的期望恰好等于所有 n 阶排列中 $W(M(p))$ 的期望。

证明. 考虑对于每一个 n 阶排列 p ，如果 x 满足 $M(x) = M(p)$ ，令 $q = p^{-1}$ ，则根据偏序关系生成矩阵的定义可以得到 $i \in [1, n-1]$ 均有 $x_{q_i} < x_{q_{i+1}}$ ，通过积分 n 轮来求解其出现的概率，由于在第 i 轮积分时会乘上一个 $\frac{1}{i}$ ，所以可以得到该条件出现的概率为 $\frac{1}{n!}$ ，故对于每一个排列 p ，满足 $M(x) = M(p)$ 的概率恰好为 $\frac{1}{n!}$ 。

而对于序列 x ，有 $P(x_i = x_j) = 0$ ，所以添加条件 x 互不相同不会对结果产生影响，对于每一个满足 x 互不相同的序列 x 都存在一个排列 p 满足 $M(x) = M(p)$ (将 x 离散化即可得到)，那么就有：

$$\begin{aligned}
 E(W(M(x))) &= \int_{x_1=0}^1 \int_{x_2=0}^1 \dots \int_{x_n=0}^1 W(M(x)) dx_1 dx_2 \dots dx_n \\
 &= \sum_p \frac{W(M(p))}{n!} \\
 &= \frac{\sum_p W(M(p))}{n!} \\
 &= E(W(M(p)))
 \end{aligned} \tag{1}$$

□

根据引理 3.1，实际上可以将引入问题转化为对于一个 n 个数每一个数均为 $[0, 1]$ 之间均匀随机生成的实数序列 x ，求 $\sum_{i=1}^{n-1} [x_i < x_{i+1}] = k$ 的概率，最后将答案乘以 $n!$ 即可。

¹luogu P5825 改

考虑定义一个长度为 n 的序列 y 满足 $y_i = (x_i - x_{i-1}) \bmod 1$ (其中定义 $x_0 = 0$)，那么有

$$\begin{aligned} \sum_{i=1}^n y_i &= \sum_{i=1}^n (x_i - x_{i-1}) \bmod 1 \\ &= \sum_{i=1}^n x_i - x_{i-1} + [x_i < x_{i-1}] \\ &= x_n + \sum_{i=1}^n [x_i < x_{i-1}] \\ &= x_n + \sum_{i=2}^n [x_i < x_{i-1}] \end{aligned} \quad (2)$$

根据对称性则可以得到 $[\sum_{i=1}^n y_i] = \sum_{i=2}^n [x_i < x_{i-1}]$ ，而又有 $P([\sum_{i=1}^n y_i] = k) = P(\sum_{i=2}^n [x_i < x_{i-1}] = k) = P(\sum_{i=1}^{n-1} [x_i < x_{i+1}] = k)$ ，所以我们实际上仅需求出 $P([\sum_{i=1}^n y_i] = k)$ 即可。

不难证明 y 也是 n 个数的每个数均为 $(0, 1)$ 之间均匀随机生成的实数序列，且存在与 x 的双射，而考虑到实际上我们仅需求出 $\sum_{i=1}^n y_i$ 落在 $[k, k+1)$ 的概率，即 $\int_{y_1=0}^1 \int_{y_2=0}^1 \dots \int_{y_n=0}^1 [\sum_{i=1}^n y_i \in [k, k+1)] dy_1 dy_2 \dots dy_n$ 。

通过简单容斥可以得到

$$\begin{aligned} ans &= \sum_{S \subseteq \{1, 2, \dots, n\}} (-1)^{|S|} \int_{y_1=1 \in S}^{+\infty} \int_{y_2=2 \in S}^{+\infty} \dots \int_{y_n=n \in S}^{+\infty} [\sum_{i=1}^n y_i \in [k, k+1)] dy_1 dy_2 \dots dy_n \\ &= \sum_{i=0}^n \binom{n}{i} (-1)^i \int_{y_1=0}^{+\infty} \int_{y_2=0}^{+\infty} \dots \int_{y_n=0}^{+\infty} [\sum_{i=0}^n y_i \in [k-i, k+1-i)] dy_1 dy_2 \dots dy_n \\ &= \sum_{i=0}^k \binom{n}{i} (-1)^i \frac{(k+1-i)^n - (k-i)^n}{n!} \end{aligned} \quad (3)$$

最后乘上 $n!$ 即 $\sum_{i=0}^k \binom{n}{i} (-1)^i ((k+1-i)^n - (k-i)^n)$ ，可以使用一次卷积轻松做到 $O(n \log n)$ 。

3.2 将连续型随机变量转化为排列

通过对上述问题的探讨可以发现当约束仅有偏序关系的形式时，可以将排列与连续型随机变量作相互的转化，这样可能可以使得问题得到简化。

例如上述问题中将排列转化为了连续型随机变量从而得到了一个非常简单的推导，下面再来看一些将连续型随机变量转化为排列的简单例题，来看一下这种方法在仅存在偏序关系的连续型随机变量问题的实用效果：

例题 3.2 (树上的随机²)。有一个由 n 个变量组成的序列 x ，其中每一个变量在 $[0, 1]$ 之间均匀随机生成，现在有 $n-1$ 组限制，每组限制形如 $x_i \leq x_j$ ，保证将所有限制视为从 i 到 j 的无向边后原图为一个连通图，求满足所有限制的概率，对 $10^9 + 7$ 取模。

²原创

$n \leq 2000$ 。

发现可以将 x 视作一个排列 p ，这样问题就变为了有若干组形如 $p_i < p_j$ 的限制，且限制在视为无向边后构成了一棵树。这是一个经典问题，考虑取 1 为根，对每一条不是父亲向儿子朝向的有向边进行容斥，这样偏序关系的图就变为了若干个外向树森林，概率为 $\frac{1}{\prod_{i=1}^n sz_i}$ 。

容斥时需要通过树形 dp 对每一个点 x 记录当前 x 子树的子树大小，复杂度 $O(n^2)$ 。

然而一些问题看上去并不是仅存在偏序关系的类型，但通过非常简单的转化就可以变成仅存在偏序关系的类型：

例题 3.3 (Tenzing and Random Real Numbers³)。有 n 个在 $[0, 1]$ 之间均匀随机的变量 $x_1, x_2, \dots, x_n$ ，有 m 组限制，每组限制形如 $x_i + x_j \leq 1$ 或 $x_i + x_j \geq 1$ ，求满足所有限制的概率，对 998244353 取模。

$n \leq 20$ 。

首先原问题并不是一个偏序关系的形式，考虑进行一些变换使其变成具有偏序关系的形式，令 $y_i = |x_i - 0.5|$ ，那么限制实际上等价于若 $y_i < y_j$ ，那么如果存在限制 $x_i + x_j \leq 1$ 则 $x_j \leq 0.5$ ；如果存在限制 $x_i + x_j \geq 1$ ，则 $x_j \geq 0.5$ 。

将 y 视作一个排列 p ，将 $x_j \leq 0.5$ 视为 A 型点， $x_j \geq 0.5$ 视为 B 型点 (由于 $P(x_j = 0.5) = 0$ ，可以限制 $x_j \neq 0.5$ ，不会对结果产生影响)，那么相当于对于 $p_i < p_j$ 时，若存在 $x_i + x_j \leq 1$ 的限制时，则要求 j 为 A 型点；若存在 $x_i + x_j \geq 1$ 的限制时，则要求 j 为 B 型点， j 取 A, B 型点的概率各为 $\frac{1}{2}$ 。

通过对 j 的约束条件可以非常简单的算出 j 满足所有限制的概率，仅需作一个 $O(2^n n)$ 的一个状压 dp 即可。

4 整数差分约束的类型

在很多连续型随机变量问题的限制并不是单纯的仅存在偏序关系的限制，这里再探讨一种比该限制要更强的一种限制： $a_j - a_i \leq c$ (其中 c 为一个整数)，为了更好的描述问题，这里先作一些简单的定义：

定义 4.1 (整数差分约束)。当一些对于实数序列 a 的不等关系约束中仅存在形如 $a_i - a_j \leq c$ 或 $a_i - a_j \geq c$ (其中 c 为整数) 的形式时，称这些不等关系约束构成了整数差分约束。

4.1 标准模型

在整数差分约束问题中，此时单纯的将连续型随机变量和排列进行转化的方法将不再适用，下面看一个标准的例题，并探讨解决该类问题的两种通用型方法，后文中绝大部分的连续型随机变量问题都将围绕着整数差分约束问题展开。

³CF1842H

例题 4.1 (新年的多维几何⁴)。设 $x_1, x_2, \dots, x_n$ 是 n 个实数变量，其中第 i 个变量 x_i 在区间 $[l_i, r_i]$ 内均匀随机生成，所有 l_i 和 r_i 均为给定的整数且 $l_i \leq r_i$ (约定 $l_i = r_i$ 时， $[l_i, r_i]$ 表示单元素集合 $\{l_i\}$)。

给定 $n \times n$ 的整数矩阵，矩阵的每个元素代表一个约束，其中第 i 行第 j 列的元素 $a_{i,j}$ 代表约束

$$x_i - x_j \geq a_{i,j}$$

求这 $n \times n$ 个约束同时被满足的概率，当答案与参考答案的绝对误差不超过 10^{-7} 时被认为是正确的。

$$n \leq 5, 0 \leq l_i, r_i \leq 10, -10 \leq a_{i,j} \leq 10。$$

原问题的实数变量之间的约束显然是十分复杂的，直接得到一个简单的刻画是困难的。

不妨先考虑 $l_i < r_i$ 的情况，注意到其中 $a_{i,j}$ 是整数，而且 l_i 与 r_i 的范围很小，如果对于每一个变量 x_i 如果已经事先确定好了其整数部分 $y_i = \lfloor x_i \rfloor$ ，令其小数部分为 $z_i = x_i - y_i$ ，可以认为 z 是 $[0, 1)$ 之间的连续型随机变量。

考虑现在约束条件变成了什么，如果有 $y_i - y_j \leq a_{i,j} - 1$ ，由于有 $z_i - z_j < 1$ 此时一定有 $x_i - x_j \leq a_{i,j}$ ，所以当 $y_i - y_j \leq a_{i,j}$ 时概率一定为 0；而当 $y_i - y_j \geq a_{i,j} + 1$ 时，由于 $z_i - z_j > -1$ ，一定有 $x_i - x_j \geq a_{i,j}$ ，故此时该限制恒成立。

现在仅需考虑 $y_i - y_j = a_{i,j}$ 所产生的限制，不难发现等价于 $z_i - z_j \geq 0$ ，即 $z_i \geq z_j$ ，这相当于有 n 个 $[0, 1)$ 之间均匀分布的随机变量，有若干组限制，求这些限制均满足的概率，按照前一部分的方法跑一个 $O(2^n n)$ 的简单状压 dp 即可。

然而事情还没有结束，我们还忽略了 $l_i = r_i$ 的情况，但实际上由于此时 x_i 的取值是确定的，则对于 x_j 要求其一定要落在 $[x_i + a_{j,i}, x_i - a_{i,j}]$ 中，可以将这个与 $[l_j, r_j]$ ，如果出现了 $l_j = r_j$ 再继续增广，直到均满足 $l_i < r_i$ 即可。

于是复杂度可以做到 $O((r - l + 1)^n 2^n n)$ ，可以发现这样的做法是依赖于值域较小的性质的。

对于这种整数差分约束的情况，在上述做法中，我们将一个实数拆分成了整数部分和小数部分，其中当整数部分的取值确定时小数部分的限制仅存在偏序关系，这样可以将小数部分的问题转化为排列问题，从而得到一个较好的解决。

但这还不够，实际上可以在值域上可以做到更优：

例题 4.2 (新年的多维几何加强版⁵)。设 $x_1, x_2, \dots, x_n$ 是 n 个实数变量，其中第 i 个变量 x_i 在区间 $[l_i, r_i]$ 内均匀随机生成，所有 l_i 和 r_i 均为给定的整数且 $l_i \leq r_i$ (约定 $l_i = r_i$ 时， $[l_i, r_i]$ 表示单元素集合 $\{l_i\}$)。

⁴UOJ352

⁵UOJ352 加强版

给定 $n \times n$ 的整数矩阵，矩阵的每个元素代表一个约束，其中第 i 行第 j 列的元素 $a_{i,j}$ 代表约束

$$x_i - x_j \geq a_{i,j}$$

求这 $n \times n$ 个约束同时被满足的概率，对 998244353 取模。

$$n \leq 8, 0 \leq l_i, r_i \leq 10^{16}, -10^{16} \leq a_{i,j} \leq 10^{16}。$$

先用前面的方法将 $l_i = r_i$ 的变量去除，下面认为 $l_i < r_i$ ，考虑设一个元 $x_0 = 0$ ，则 $l_i \leq x_i \leq r_i$ 相当于 $x_i - x_0 \geq l_i, x_0 - x_i \geq -r_i$ ，令 $a_{i,0} = l_i, a_{0,i} = -r_i$ 。

考虑逐步的消除变量，考虑对变量 n 的限制条件是什么，即有对于 $0 \leq i < n$ 有 $x_i + a_{n,i} \leq x_n \leq x_i - a_{i,n}$ ，那么对于任意的 $0 \leq i, j < n$ 有 $x_i + a_{n,i} \leq x_j - a_{j,n}$ ，则可以对前 $n - 1$ 个变量添加约束条件 $x_j - x_i \geq a_{n,i} + a_{j,n}$ 。

考虑对于 x_n 的约束，由于有了对前 $n - 1$ 个变量的额外约束条件，所以左约束一定小于等于右约束，可以让右约束减去左约束即可得到答案，每次分别枚举取到右约束的最紧的约束和取到左约束的最紧的约束，共 $2(n - 1)$ 种方法，并对紧约束本身再添加新的约束。以左约束为例，如果枚举到 d 为最优的约束，则有对于 $i \neq d$ 有 $x_d - a_{d,n} \geq x_i - a_{i,n}$ 即 $x_i - x_d \leq a_{i,n} - a_{d,n}$ 。

然后 x_n 就可以不用考虑了，将其删除不断的搜索变为子问题，则可以得到 $O(n! 2^n)$ 个问题，对这些问题，令枚举最紧约束所对应变量的下标为 $f a_i$ ，则每一个约束都形如对于 $1 \leq i \leq n$ ，有 $0 \leq x_i \leq x_{f a_i} + c_i$ ，求 x 的测度之和，不难发现其构成了一个树形结构，考虑在上面进行自底而上的树形 dp，对于每一个节点 d 维护 $x_d = y$ 是所对应的测度的多项式 $f_d(y)$ ，那么有 $f_d(y) = \prod_{w \in \text{son}(d)} \int_{z=0}^{y+c_w} f_w(z) dz$ ，通过积分和卷积即可维护。

最终仅需求出 $f_0(0)$ 即可，但需要将其转变为普通的概率需要乘上 $\frac{1}{\prod_{i=1}^n (r_i - l_i)}$ ，每一次跑树形 dp 的复杂度为树形背包复杂度即 $O(n^2)$ ，故复杂度可以做到 $O(2^n n! n^2)$ 。

实际上所有整数差分约束问题就都可以用这种方法做到 $O(2^n n! n^2)$ 的复杂度了，实用性是非常强的。

上述做法实际上不难扩展到一般的差分约束的版本，即 c 为任意实数的情况，但在不取模的情况下带来的精度误差往往是比较大的，所以实际上在很多连续型随机变量问题中主要考虑的是整数差分约束的情况而不是一般的差分约束。

4.2 特殊限制

但不难发现对于标准整数差分约束问题的将约束关系逐步转化为树的方法往往只能做变量个数 n 很小的范围，复杂度带的 $n!$ 因子实际上是非常高的，然而在很多问题中限制是比较特殊的，从而可以做到更优的时间复杂度。

下面探讨几个特殊的例子，在值域较小时上述的第一种方法即将一个数拆分成整数部分和小数部分的做法是具有更好的拓展性的，下文将加以介绍。值得一提的是如果采用上述的第二种方法即将约束关系逐步转化为树的方法可以在下面两题中分别做到 $O(2^n n^2)$ 于

$O(3^n n! n^2)$ 而不依赖于值域，但实际上在第六章中会讲述这两个问题不依赖值域的低复杂度做法，所以这里不在此展开。

例题 4.3 (招新⁶). 在一条数轴上一次加入 n 条线段，每一条线段的长度均为 k ，每一条线段的左端点为一个 $[0, m]$ 间均匀随机的实数，求不存在一个位置同时被三个及其以上的位置覆盖的概率，对 p 取模。

$$n \leq 50, k \leq m \leq 150, 10^8 < p \leq 10^9 + 9.$$

不难发现如果将所有线段左端点所在位置对应的实数排成一个长度为 n 的序列 x ，如果令 x 从小到大的排序结果为 y ，则原问题的限制等价于 $y_{i-1} \leq y_i, y_i - y_{i-2} > k$ ，考虑先计算 y 的概率，再将结果乘以 $n!$ 。

令 $z_i = \lfloor y_i \rfloor, w_i = y_i - z_i$ ，则不难得到 $z_{i-1} \leq z_i, z_i - z_{i-2} \geq k$ ，且有 $z_{i-1} = z_i$ 时 $w_{i-1} \leq w_i$ ， $z_i - z_{i-2} = k$ 时 $w_{i-2} < w_i$ ，这样如果我们如果事先确定好了 z 序列，建立好对 w 的约束后，将其化为排列后就只需要求出随机一个合法的 w 其满足所有限制的概率。

先考虑 z 确定时如何计算合法的 w 的方案数，考虑令 $dp_{i,j,t}$ 表示考虑了 $[1, i]$ 的元素， w_{i-1} 在 $[1, i]$ 的排名为 j ， w_i 在 $[1, i]$ 的排名为 t 的概率，转移分类讨论 $i+1$ 插入的位置即可，在插入时要求与 $i-1, i$ 的偏序关系要合法，这部分容易使用前缀和优化至 $O(n^3)$ 。

考虑 **dp** 套 **dp** 的思想，在外层 **dp** 中再记录 z_i 的取值，将 **dp** 定义修改为 $dp_{i,j,t,l,m}$ 表示考虑了 $[1, i]$ 的元素， $z_{i-1} = j, z_i = t, w_{i-1}$ 在 $[1, i]$ 的排名为 l, w_i 在 $[1, i]$ 的排名为 m 的概率，在添加时对于 z_{i+1} 不为 t 与 $t+k$ 的部分，由于不会新增约束所有 z_{i+1} 对内层 **dp** 的转移等价，可以朴素的使用前缀和进行解决。

直接这样做复杂度为 $O(n^3 m^2)$ ，可能无法通过，但注意到在 **dp** 过程中如果当 $t-j > k$ 时将 j 设置为 $t-k-1$ 不会对状态产生影响，这样复杂度就降为了 $O(n^3 mk)$ 了。根据题意不难得到只有当 $nk \leq 2m$ 时概率才会非 0，这部分可以事先判掉，于是复杂度可以做到 $O(n^2 m^2)$ 。

例题 4.4 (Arcs on a Circle⁷). 有一个长度为 C 的圆，在上面画了 n 个弧，弧 i 有长度 l_i 。

每一条弧 i 随机均匀地放置在圆上：选择圆上的一个随机实点，然后出现一条以该点为中心的、长度为 l_i 的弧，弧是独立放置的，例如，它们可以相互交叉或包含。

现在问你圆的每一个实点被至少一个弧覆盖的概率是多少？(注意一条弧覆盖了它的两个端点)，答案合法当且仅当与标准答案的绝对误差 $\leq 10^{-11}$ 。

$$2 \leq n \leq 6, 2 \leq C \leq 50, 1 \leq l_i < C.$$

考虑如何判定合法性，考虑将 l_1 的左端点位置将环断开，令第 i 个弧向顺时针方向的左端点的位置为 x_i ，将所有弧按照 x 从小到大排序，那么则有对于 $[2, n]$ 内的整数 $d, \max_{i=1}^{d-1} (x_i + l_i) \geq x_d$ ，且 $\max_{i=1}^n (x_i + l_i) \geq C$ 。

按照前一题的方法套路性的转化为整数部分和小数部分后，枚举小数部分偏序关系全排列 p ，然后对所有弧进行状压 **dp**，令 $dp_{x,S,d,r}$ 表示处理坐标在 $[0, x]$ 的部分时，已经加入

⁶PKUWC2021 D2T3

⁷AGC020F

了集合 S 中的弧后 $x_i + l_i$ 的最大值的整数部分为 d ，小数部分对应排列中的排名为 r 的概率，在同一位置加入弧时一定要按照小数部分的排名从小到大的顺序加入。

复杂度于是可以做到 $O(2^{n-1} n^2 C^2 (n-1)!)$ 。

然而实际上这个做法有一个明显不优的地方，一开始枚举的 $(n-1)!$ 种全排列带来了较高的复杂度，实际上这部分是可以放在 **dp** 内部的，考虑令 $dp_{x,S,d,r}$ 表示处理坐标在 $[0, x]$ 的部分时，已经加入了集合 S 中的弧后 $x_i + l_i$ 的最大值的整数部分为 d ，小数部分在已经加入的集合 S 中排名为 r 的概率。

值得注意的是如果当前位置恰好为 d 时则要求对于整数部分在该位置的所有段的小数部分的排名中最小的一个要插入在 r 前面，此处需要多枚举在插入之前的 r 再在当前位置跑一个额外的 **dp**。

于是复杂度可以做到 $O(2^{n-1} n^3 C^2)$ ，可以做更大的数据范围，如 $n \leq 11$ 。

5 一类多项式的引入

在第二章标准模型介绍的两种方法中，第一种方法往往依赖于值域较小的性质，第二种方法的时间复杂度往往随着变量个数增长而高速增长，对于变量很多但值域范围也非常大的整数差分约束问题两种方法都将不再适用。实际上在很多具有特殊限制的整数差分约束问题中，需要更强的工具来帮助我们完成相应的任务，笔者将引入一类特殊的多项式，在本文中笔者将其命名为积分多项式 (Integral polynomials) 来帮助解决连续型随机变量的相关问题。

5.1 积分多项式的定义

定义 5.1 (非负域可导函数). 定义一个函数 $f(x)$ 为非负域可导函数，当且仅当 $f(x)$ 的定义域为非负实数且 $f(x)$ 可导。

定义 5.2 (积分多项式算子). 对于一个定义域为非负域的多项式 $f(x) = \sum_{i=0}^n a_i x^i$ ，定义积分多项式算子 $\mathfrak{I}$ 满足 $\mathfrak{I}f(x) = \sum_{i=0}^n a_i i! x^i$ ，同时定义积分多项式逆算子 $\mathfrak{I}^{-1}f(x) = \sum_{i=0}^n \frac{a_i}{i!} x^i$ 。

定义 5.3 (积分多项式). 如果对于一个非负域可导函数 $f(x)$ ，满足 $f(x)$ 为一个多项式，令 $f(x) = \sum_{i=0}^n a_i x^i$ ，那么定义其积分多项式为 $\mathfrak{I}f(x)$ ，通常情况下 $f(x)$ 可以看做描述长度为 x 的线段内部某种测度的函数，多项式的指数 i 可以看作是线段内部某种元的数目，积分多项式的作用在于使得 $f(x)$ 的一些操作得到更加简便的计算。

接下来分析一下一些非负域多项式函数的运算在积分多项式中所表达的意义：

5.2 非负域多项式函数的加減与数乘

引理 5.1. 若 $h(x) = f(x) + g(x)$, 则有 $\Im h(x) = \Im f(x) + \Im g(x)$ 。

引理 5.2. 若 $h(x) = f(x) - g(x)$, 则有 $\Im h(x) = \Im f(x) - \Im g(x)$ 。

引理 5.3. 若 $g(x) = kf(x)$, 则有 $\Im g(x) = k\Im f(x)$ 。

证明十分显然。

实际上上述引理告诉了我们积分多项式算子是线性算子, 当需要对某种描述测度的多项式进行线性运算时仅需对其积分多项式进行相同的运算即可。

5.3 非负域可导函数的类卷积

两个积分多项式 f 与 g 的乘法运算的意义看上去非常抽象, 但可以考虑在解决连续型随机变量问题时需要解决的一个非常重要的问题: 两个描述连续型变量或随机过程内部测度的函数的类卷积问题, 在这里我们认为这些描述内部测度的函数均为非负域可导函数。

定义 5.4 (非负域可导函数的类卷积). 对于两个非负域可导函数 $F(x)$ 与 $G(x)$, 定义 $F(x)$ 和 $G(x)$ 的类卷积为一个非负域可导函数 $H(x)$ 满足 $H(x) = (\int_{y=0}^x F'(y)G(x-y)dy) + F(0)G(x)$, 将其记作 $H(x) = F(x) * G(x)$, 非常要注意的是类卷积与数学上卷积的定义是相似但是不同的。

可以这样直观上来理解两个非负域可导函数的类卷积, 一个长度为 y 的线段内有若干个元, 一个长度为 z 的线段中有若干个元, 由于要求元的拆分的方式必须是唯一的, 所以要求 y 线段的末端必须要有一个元, 所以需要求一次导, 然后用积分将其合并贡献到 $x = y + z$ 的长度上。

但 $F(0)$ 的信息在求导过程中将会被丢失, 所以需要最后在加上一个 $F(0)G(x)$ 。

实际上这可以与普通型生成函数的无序拼接与指数型生成函数的有序拼接作类比, 这也是一种独特的关于连续型随机变量组合对象的拼接方式。

而且可以证明非负域可导函数的类卷积满足如下性质:

引理 5.4. $F(x) * G(x)$ 的结果还是一个非负域可导函数。

引理 5.5. $F(x) * G(x) = G(x) * F(x)$

引理 5.6. $F(x) * G(x) * H(x) = F(x) * (G(x) * H(x))$

引理 5.7. $F(x) * (G(x) + H(x)) = F(x) * G(x) + F(x) * H(x)$

引理 5.8. 令 $I(x) = 1(x \geq 0)$, 则 $F(x) * I(x) = I(x) * F(x) = F(x)$ 。

由于证明较为繁琐且与本文关系不大, 故在此略去, 实际上绝大部分与卷积的证明是相似的, 可以参考 [6] 中的证明。

同时为了让后文中的一些描述更加方便, 我们再定义一下非负域可导函数类卷积的幂次:

定义 5.5 (非负域可导函数类卷积的非负整数幂). 定义一个非负域可导函数 $F(x)$ 类卷积的 k (k 为一个非负整数) 次幂为 $G(x) = F(x)^{[k]}$, 其中当 $k = 0$ 时 $G(x) = [x = 0]$, 当 $k > 0$ 时有 $G(x) = F(x)^{[k-1]} * F(x)$ 。

5.4 非负域多项式函数的类卷积

实际上对于非负域可导函数的类卷积, 如果两个函数都是多项式, 那么实际上仅需将其积分多项式相乘即可, 有如下引理成立:

引理 5.9. 若 $h(x) = f(x) * g(x)$, 则有 $\Im h(x) = (\Im f(x))(\Im g(x))$ 。

证明. 令 $f(x) = \sum_{i=0}^n a_i x^i, g(x) = \sum_{i=0}^m b_i x^i, h(x) = \sum_{i=0}^t c_i x^i$, 有

$$\begin{aligned} f(x) * g(x) &= \left(\int_{y=0}^x f'(y)g(x-y)dy \right) + f(0)g(x) \\ &= \left(\int_{y=0}^x \sum_{i=0}^{n-1} a_{i+1}(i+1)y^i \sum_{j=0}^m b_j(x-y)^j dy \right) + a_0 \sum_{i=0}^m b_i x^i \\ &= \left(\sum_{i=0}^{n-1} a_{i+1}(i+1) \sum_{j=0}^m b_j \int_{y=0}^x y^i(x-y)^j dy \right) + a_0 \sum_{i=0}^m b_i x^i \\ &= \left(\sum_{i=0}^{n-1} a_{i+1}(i+1) \sum_{j=0}^m b_j \int_{y=0}^x y^i \sum_{t=0}^j \binom{j}{t} (-1)^t x^{j-t} y^t dy \right) + a_0 \sum_{i=0}^m b_i x^i \\ &= \left(\sum_{i=0}^{n-1} a_{i+1}(i+1) \sum_{j=0}^m b_j \sum_{t=0}^j \binom{j}{t} (-1)^t x^{j-t} \int_{y=0}^x y^{i+t} dy \right) + a_0 \sum_{i=0}^m b_i x^i \\ &= \left(\sum_{i=0}^{n-1} a_{i+1}(i+1) \sum_{j=0}^m b_j \sum_{t=0}^j \binom{j}{t} (-1)^t x^{j-t} \frac{x^{i+t+1}}{i+t+1} \right) + a_0 \sum_{i=0}^m b_i x^i \\ &= \left(\sum_{i=0}^{n-1} a_{i+1}(i+1) \sum_{j=0}^m b_j x^{j+1} \sum_{t=0}^j \binom{j}{t} \frac{(-1)^t}{i+t+1} \right) + a_0 \sum_{i=0}^m b_i x^i \end{aligned} \quad (4)$$

令 $L(i, j) = \sum_{t=0}^j \binom{j}{t} \frac{(-1)^t}{i+t+1}$ ，进行简单变形有：

$$\begin{aligned}
 L(i, j) &= \sum_{t=0}^j \binom{j}{t} \frac{(-1)^t}{i+t+1} \\
 &= \left(\sum_{t=1}^j \binom{j}{t} \frac{(-1)^t}{i+t+1} \right) + \frac{1}{i+1} \\
 &= \left(\sum_{t=1}^j \left(\binom{j-1}{t-1} + \binom{j-1}{t} \right) \frac{(-1)^t}{i+t+1} \right) + \frac{1}{i+1} \\
 &= \left(\sum_{t=1}^j \left(\binom{j-1}{t-1} + \binom{j-1}{t} \right) \frac{(-1)^t}{i+t+1} \right) + \frac{1}{i+1} \quad (5) \\
 &= \left(\sum_{t=0}^{j-1} \binom{j-1}{t} \frac{(-1)^{t+1}}{i+t+1} + \sum_{t=1}^j \binom{j}{t} \frac{(-1)^t}{i+t+1} \right) + \frac{1}{i+1} \\
 &= - \sum_{t=0}^{j-1} \binom{j-1}{t} \frac{(-1)^t}{i+t+1} + \sum_{t=0}^j \binom{j}{t} \frac{(-1)^t}{i+t+1} \\
 &= L(i, j-1) - L(i+1, j-1)
 \end{aligned}$$

对于 j 很小的情况有 $L(i, 0) = \frac{1}{i+1}$, $L(i, 1) = \frac{1}{(i+1)(i+2)}$, $L(i, 2) = \frac{2}{(i+1)(i+2)(i+3)}$, $L(i, 3) = \frac{6}{(i+1)(i+2)(i+3)(i+4)}$ ，可以猜想 $L(i, j) = \frac{j!}{(i+1)^j} = \frac{i!j!}{(i+j+1)!}$ 。

可以通过递推式归纳证明， $j=0$ 的情况显然成立，对于 $j>0$ ，此时已经有 $L(i, j-1) = \frac{i!(j-1)!}{(i+j)!}$, $L(i+1, j-1) = \frac{(i+1)!(j-1)!}{(i+j+1)!}$ ，那么 $L(i, j) = L(i, j-1) - L(i+1, j-1) = \frac{i!(j-1)!}{(i+j)!} - \frac{(i+1)!(j-1)!}{(i+j+1)!} = \frac{i!(j-1)(i+j+1) - i!(j-1)!(i+1)}{(i+j+1)!} = \frac{i!j!}{(i+j+1)!}$ 。

将 $L(i, j) = \frac{i!j!}{(i+j+1)!}$ 重新带入 (4) 得：

$$\begin{aligned}
 f(x) * g(x) &= \left(\sum_{i=0}^{n-1} a_{i+1}(i+1) \sum_{j=0}^m b_j x^{i+j+1} \sum_{t=0}^j \binom{j}{t} \frac{(-1)^t}{i+t+1} \right) + a_0 \sum_{i=0}^m b_i x^i \\
 &= \left(\sum_{i=0}^{n-1} a_{i+1}(i+1) \sum_{j=0}^m b_j x^{i+j+1} \frac{i!j!}{(i+j+1)!} \right) + a_0 \sum_{i=0}^m b_i x^i \\
 &= \left(\sum_{i=0}^{n-1} a_{i+1} \sum_{j=0}^m b_j x^{i+j+1} \frac{(i+1)!j!}{(i+j+1)!} \right) + a_0 \sum_{i=0}^m b_i x^i \quad (6) \\
 &= \left(\sum_{i=1}^n a_i \sum_{j=0}^m b_j x^{i+j} \frac{i!j!}{(i+j)!} \right) + a_0 \sum_{i=0}^m b_i x^i \\
 &= \sum_{i=0}^n \sum_{j=0}^m a_i b_j \frac{i!j!}{(i+j)!} x^{i+j}
 \end{aligned}$$

而有 $f(x) * g(x) = h(x)$ ，则可知 $c_k = \sum_{i=0}^n \sum_{j=0}^m [i+j=k] a_i b_j \frac{i!j!}{k!}$ ，则有 $k!c_k = \sum_{i=0}^n \sum_{j=0}^m i!j! a_i b_j = (\sum_{i=0}^n i!a_i)(\sum_{j=0}^m j!b_j)$ ，将其乘上 x^k 后作求和可以得到 $\sum_{i=0}^n i!c_i x^i = (\sum_{j=0}^m j!a_j x^j)(\sum_{j=0}^m j!b_j x^j)$ ，即有 $\Im h(x) = (\Im f(x))(\Im g(x))$ 。□

这告诉了我们实际上如果要将两个非负域多项式函数作类卷积，实际上就仅需将它们的积分多项式相乘即可。

6 积分多项式的应用

6.1 分段积分多项式

在上文中我们已经论述了积分多项式算子为线性算子，而且将两个描述内部测度的非负域多项式函数的进行类卷积运算就对应着将积分多项式相乘，所以自然而然的其他的复杂运算也有了它们的定义。

然而在实际的应用中，仅仅用一个多项式不足以描述一个连续型随机变量或一个随机过程内部测度的函数，在很多情况下它们都是一个分段的多项式，下面定义分段积分多项式：

定义 6.1 (分段多项式). 定义一个非负域可导函数 $F(x)$ 为分段多项式当且仅当其可以写为 $F(x) = \sum_{i=1}^n f_i(x - l_i)[x \geq l_i]$ 的形式，且满足所有 l_i 均为非负实数，每个 $f_i(x)$ 均为非负域多项式。

定义 6.2 (分段积分多项式算子). 对于分段多项式 $F(x) = \sum_{i=1}^n f_i(x - l_i)[x \geq l_i]$ ，定义分段积分多项式算子 $\Im$ 满足 $\Im F(x) = \sum_{i=1}^n \Im f_i(x - l_i)[x \geq l_i]$ 。

定义 6.3 (分段积分多项式). 如果分段积分多项式 $F(x)$ ，定义其分段积分多项式为 $\Im F(x)$ ，通常情况下 $F(x)$ 可以看做描述长度为 x 的线段内部某种测度的函数，指数 i 仍然可以看作是线段内部某种元的数目，一般的非负域多项式往往比较简单，不能够描述处理一些较为复杂的连续型随机变量或随机过程，而由于分段多项式可以描述更多复杂的连续型随机变量或随机过程内部的测度，分段积分多项式较普通的积分多项式更加强大。

由于分段积分多项式的段可能非常的繁多，很多时候都是二维，三维，甚至多维的多项式运算，这意味着它的复杂度很有可能会带若干个 n 因子，但通常情况下复杂度与值域无关。

6.2 具体应用

在第三章中，我们举例了两个在整数差分约束问题在特殊限制下的例题，接下来我们看一下积分多项式以及分段积分多项式在这两个例题中的具体表现：

例题 6.1 (招新加强版⁸). 在一条数轴上一次加入 n 条线段，每一条线段的长度均为 k ，每

⁸PKUWC2021 D2T3 加强版

一条线段的左端点为一个 $[0, m]$ 间均匀随机的实数，求不存在一个位置同时被三个及其以上的位置覆盖的概率，对 998244353 取模。

$$n \leq 250, k \leq m \leq 10^{18}.$$

考虑换一个角度思考这个问题，将其看作在一个线段中撒了 n 个点，如果两个点 x, y 满足 $|x - y| \leq k$ 则连一条边，则原问题的限制条件相当于这个图是一个二分图，考虑先假设点的位置序列 x 满足 $x_i < x_{i+1}$ ，求出其测度之后再修正为概率。

考虑使用二分图计数的技巧，先定义原二分图所有染色方案的内部测度的函数为 $F(x)$ ，不难发现这就相当于将其点集划分成了 A, B 两个独立的集合，满足 A 为一个独立集， B 也为一个独立集，不难发现实际上仅需枚举每组 $(|A|, |B|)$ 的取值求相应的伴随函数的平移之和即可得到。

考虑对每一组满足 $a + b > 0$ 的 (a, b) 计算 $|A| = a, |B| = b$ 所对应内部测度的函数，经过分析不难得到 $f_{a,b}(x) = \frac{(x-k(a-1))^a(x-k(b-1))^b}{a!b!} (x \geq k(\max(a, b) - 1))$ ，但此时 x 的定义域为 $x \geq k(\max(a, b) - 1)$ ，与非负域可导函数的定义域不同，所以要进行一些微调。

考虑将定义变为 $f_{a,b}(x) = \frac{(x+k(\max(a,b)-1)-k(a-1))^a(x+k(\max(a,b)-1)-k(b-1))^b}{a!b!} (x \geq 0)$ ，不妨令 $a \leq b$ ，则有 $f_{a,b}(x) = \frac{(x+k(b-a))^a x^b}{a!b!} (x \geq 0)$ ，并取相应的 $l_{a,b} = k(\max(a, b) - 1)$ ，则有 $F(x) = \sum_{a=0}^n \sum_{b=0}^{n-a} [a + b > 0][x \geq l_{a,b}] f_{a,b}(x - l_{a,b})$ ，同理有 $\Im F(x) = \sum_{a=0}^n \sum_{b=0}^{n-a} [a + b > 0][x \geq l_{a,b}] \Im f_{a,b}(x - l_{a,b})$ ，此时我们仅需让 $\Im F(x)$ 进行一系列运算再还原即可。

但这显然是不够的，由于要求恰好有 n 个点的方案数，所以要设置一个元 y ，最后求 $[y^n]F$ 处的取值。而且由于是分段积分多项式，但段点一定形如 $kd (0 \leq d \leq n - 1)$ 的形式，所以需要再设置一个元 z ，如果取到 $[z^d]F$ 且 $kd \leq m$ 则可以在 $m - kd$ 代入积分多项式中。

即实际上我们需要定义一个三元多项式 $\tilde{F}(x, y, z)$ ，有 $\tilde{F}(x, y, z) = \sum_{a=0}^n \sum_{b=0}^{n-a} [a + b > 0][x \geq l_{a,b}] \Im f_{a,b}(x - l_{a,b}) y^{a+b} z^{l_{a,b}}$ ，但这仍然是有问题的，在 z^A 与 z^B 相乘时指数实际上还会增加 1，因为要求两个段在拼接时还要求其中要有 k 的间隔。

所以可以修改为 kd 在 $[z^{d+1}] \tilde{F}(x, y, z)$ 处产生贡献，即可将 $\tilde{F}(x, y, z)$ 修改为满足 $\tilde{F}(x, y, z) = \sum_{a=0}^n \sum_{b=0}^{n-a} [a + b > 0][x \geq l_{a,b}] \Im f_{a,b}(x - l_{a,b}) y^{a+b} z^{l_{a,b}+1}$ 。

并定义 $\tilde{G}(x, y, z)$ 表示连通二分图的内部测度的多元函数， $\tilde{H}(x, y, z)$ 表示二分图的内部测度的多元函数，那么有 $\tilde{H}(x, y, z) = \frac{\tilde{G}(x, y, z)}{1 - \tilde{G}(x, y, z)}$ ， $\tilde{F}(x, y, z) = \frac{2\tilde{G}(x, y, z)}{1 - 2\tilde{G}(x, y, z)}$ (需要注意的是这里的运算不是 $\exp$ 因为段与段的拼接是无序的，这与 EGF 的有序拼接是不同的)，联立两式可以得到 $\tilde{H}(x, y, z) = \frac{\frac{\tilde{F}(x, y, z)-1}{2\tilde{F}(x, y, z)}}{1 - \frac{\tilde{F}(x, y, z)-1}{2\tilde{F}(x, y, z)}} = \frac{\tilde{F}(x, y, z)}{\tilde{F}(x, y, z)+2}$ ，最后仅要求出 $\sum_{a=0}^n \sum_{b=1}^n [x^a y^b z^b] \tilde{H}(x, y, z) \frac{[m \geq k(b-1)](m-k(b-1))^a}{a!}$ 即可，注意还需乘上 $\frac{n!}{m^n}$ 将测度转变为普通的概率。

直接进行三维多项式求逆可以做到 $O(n^3 \log n)$ ，但这个做法常数太大了，无法通过。更好的方法实际上是注意到如下事实：对于多项式的非 0 项 $x^a y^b z^c$ ，则有 $\lceil \frac{a}{2} \rceil \leq a \leq c \leq b$ ，将 a 变形为 $b - a$ ， c 变形为 $b - c$ ，这样有 $0 \leq a \leq c \leq \lfloor \frac{b}{2} \rfloor$ 。

对 x, z 使用拉格朗日插值法，对 y 这一维跑暴力求逆，由于 b, c 的值域 $\leq \lfloor \frac{n}{2} \rfloor$ ，对于每一维仅需代入 $\lfloor \frac{n}{2} \rfloor + 1$ 个点值，即仅需代入 $(\lfloor \frac{n}{2} \rfloor + 1)^2$ 个点值，虽然复杂度为 $O(n^4)$ ，但常数十分的小，可以通过 $n \leq 250$ 的数据范围。

例题 6.2 (Arcs on a Circle 加强版⁹)。有一个长度为 C 的圆，在上面画了 n 个弧，弧 i 有长度 l_i 。

每一条弧 i 随机均匀地放置在圆上：选择圆上的一个随机实点，然后出现一条以该点为中心的、长度为 l_i 的弧，弧是独立放置的，例如，它们可以相互交叉或包含。

现在问你圆的每一个实点被至少一个弧覆盖的概率是多少？(注意一条弧覆盖了它的两个端点)，你只需要求出这个概率对 $10^9 + 7$ 取模的结果即可，数据保证概率的分母不为 $10^9 + 7$ 的倍数。

$$2 \leq n \leq 12, 2 \leq C \leq 10^{18}, 1 \leq l_i \leq \min(C - 1, 10^{16}).$$

同样换一个角度思考这个问题，考虑不合法的情况，相当于存在一个点没有被覆盖，考虑先取 $x = 0$ 这个位置进行考虑，如果 $x = 0$ 没有被覆盖，那么一定不合法，该事件发生的概率为 $\prod_{i=1}^n \frac{C-l_i}{C}$ 。

对于剩下的不合法的情况就一定满足存在一个段跨过了 $x = 0$ 的部分，考虑将所有有交的弧之间连边，那么可以认为跨过 $x = 0$ 的极大连通块的最左部点 p 没有被覆盖，将这个位置断开，则变为为了一个链的问题。

考虑先确定好一个以 p 为最左端点的链，这个链中有若干个连通块，那么我们要求最左边的连通块的左端点一定要在最左侧，那么合法的 p 则有左端点长度的放置方案。

令一个分段多项式 $F_S(x)$ 表示集合 S 中的弧构成内部测度的函数，令 S 中所有弧中 l 最大的为 h_S ，则根据前一题的方法可以得到 $f_S(x) = \prod_{i \in S} (x + l_{h_S} - l_i) (x \geq 0)$ ， $F_S(x) = [x \geq l_{h_S}] f_S(x - l_{h_S})$ 。

同样将其扩展到多元多项式，考虑定义 $\tilde{F}(x, y, z) = \sum_{S \in \{1, 2, \dots, n\}} [x \geq l_d] \Im f_S(x - l_d) y^S z^{l_{h_S}}$ ，其中 x 表示积分多项式的次数，作普通的序列卷积， y, z 分别表示所有弧构成的集合，以及构成积分多项式的段的集合，作子集卷积。

令 $\tilde{G}(x, y, z)$ 表示要求集合内的弧构成连通块的内部测度的函数，由 $\tilde{F}(x, y, z) = \frac{\tilde{G}(x, y, z)}{1 - \tilde{G}(x, y, z)}$ 可得 $\tilde{G}(x, y, z) = \frac{\tilde{F}(x, y, z)}{1 + \tilde{F}(x, y, z)}$ ，由于只有当 $z = \{h_y\}$ 时才有值，则直接暴力枚举子集的复杂度为 $O(4^n n^2)$ 。

考虑如何求答案，注意我们需要将其乘上第一段长度并且强制第一段左端点取到首，可以先用两次导将其变成首尾都有点乘上长度后再积分回去使得右端点不一定取到末，即令 $\tilde{H}(x, y, z) = \frac{\partial^2 (\Im_x^{-1} \tilde{G}(x, y, z))}{\partial x^2}$ ，此时对 $\tilde{H}(x, y, z)$ 进行对于长度的点乘。

但是直接点乘 x 实际上是不对的，因为如果在分段多项式的一项为 $[x^i y^j z^k]$ 则已有的长度实际上是 $x + \sum_{d \in k} l_d$ ，令 $res = \sum_{d \in k} l_d$ ，则需要将该项点乘上 $x + res$ ，即对于多项式 $\tilde{W}(x, y, z)$ ，我们需要将其变为 $\tilde{R}(x, y, z)$ 满足 $\tilde{R}(x, y, z) = \sum_{i=0}^n \sum_{j \in \{1, 2, \dots, n\}} \sum_{k \subset j} ([x^i y^j z^k] W(x, y, z)) (x + \sum_{d \in k} l_d) x^i y^j z^k$ 。

则要求的答案即为 $\tilde{W}(x, y, z) = (\Im_x (\int \tilde{R}(x, y, z) dx)) (\tilde{F}(x, y, z) + 1)$ ，但这样仍然是不对的，对 $\Im_x^{-1} \tilde{G}(x, y, z)$ 进行求 x 的二次偏导所丢失的 $[x^1] G(x, y, z)$ 需要特殊修正，对于实际问题分析发

⁹AGC020F 加强版

现对于 $[x^l y^i z^j]G(x, y, z)$ 则其会乘上 $(\sum_{d \in j} l_d) x^i y^j z^k$ 再在 $\mathfrak{I}_x(\int \tilde{R}(x, y, z) dx)$ 加上一个额外的增量, 常数项无需修正因为根据上文的运算常数项恒为 0, 因此可以将 $\tilde{W}(x, y, z)$ 最终的式子整理为 $\tilde{W}(x, y, z) = ((\mathfrak{I}_x(\int \tilde{R}(x, y, z) dx)) + \sum_{i \in \{1, 2, \dots, n\}} \sum_{j \in i} ([x^l y^i z^j]G(x, y, z))(\sum_{d \in j} l_d) x^i y^j z^j)(\tilde{F}(x, y, z) + 1)$ 。

最终需要让 ans 额外减的值其实是 $\frac{\sum_{i=0}^n \sum_{j \in \{1, 2, \dots, n\}} [x^l y^i z^j] \tilde{W}(x, y, z) i! j! (C - \sum_{x \in j} l_x)(C - \sum_{x \in j} l_x)^j}{C^n}$, 直接对每一步暴力枚举子集实现 $O(4^n n^2)$ 的做法就可以通过了, 使用拉格朗日插值可以优化掉一个 n 因子做到 $O(4^n n)$, 但是只是快了一点, 对 z 这一维 dp , 对 y 这维子集卷积优化还可以优化到 $O(3^n n^3)$, 但是由于常数很大变得更慢了。

6.3 对三种连续型随机变量解决方法的对比

接下来, 我们将把将一个数转化为整数部分和小数部分的方法 (方法 A), 将约束关系转化为树的方法 (方法 B), 和分段积分多项式方法 (方法 C) 在整数差分约束问题上最优时间复杂度的表现进行一些比较:

方法	新年的多维几何	招新	Arcs on a Circle
A	$O(2^n V^n n)$	$O(n^2 V^2)$	$O(2^n n^3 V^2)$
B	$O(2^n n! n^2)$	$O(2^n n^2)$	$O(3^n n! n^2)$
C	-	$O(n^3 \log n)$	$O(3^n n^3)$

由此可以看出在一些具有特殊限制的整数差分约束问题中分段积分多项式方法的时间复杂度是十分优秀的, 但无法解决一般的整数差分约束问题。

7 积分多项式的本质

7.1 指数加权积分变换的引入

上述积分多项式的构造不禁让人联想, 对于一般的非负域可导函数是否也存在类似的构造, 在经过一些思考后笔者发现是存在的, 而积分多项式仅仅是它在特殊情况下的一种表现形式, 下面笔者将其命名为指数加权积分变换的积分变换:

定义 7.1 (指数加权积分变换). 对于一个定义域在 $[0, +\infty)$ 的函数 $f(x)$, 定义其指数加权积分变换的结果为函数 $F(x) = \int_{y=0}^{+\infty} f(xy) e^{-y} dy$, 记作 $F(x) = \mathcal{E}f(x)$, 值得注意的是该变换虽然与拉普拉斯变换虽然相似, 但是不是拉普拉斯变换, 同时将称 $\mathcal{E}$ 为指数加权积分变换算子。

在引出本章之前, 先介绍一个概念:

定义 7.2 (伽玛函数). 定义伽玛函数为 $\Gamma(n) = \int_{y=0}^{+\infty} y^{n-1} e^{-y} dy (n > 0)$, 有性质 $\Gamma(n+1) = \Gamma(n)n$, 当 n 为正整数时满足 $\Gamma(n) = (n-1)!$ 。

引理 7.1. 若 $H(x) = F(x) * G(x)$, 则 $\mathcal{E}H(x) = (\mathcal{E}F(x))(\mathcal{E}G(x))$ 。

证明.

$$\begin{aligned}
 \mathcal{E}H(x) &= \int_{y=0}^{+\infty} H(xy) e^{-y} dy \\
 &= \left(\int_{z=0^+}^{+\infty} \int_{w=0}^{+\infty} F'(z) G(w) dz e^{-\frac{z+w}{x}} d\frac{z+w}{x} \right) + F(0) \int_{y=0}^{+\infty} G(xy) e^{-y} dy \\
 &= \left(\int_{z=0^+}^{+\infty} \int_{w=0}^{+\infty} F'(z) G(w) dz e^{-\frac{z+w}{x}} \frac{1}{x} dw \right) + F(0) \mathcal{E}G(x) \\
 &= \frac{\left(\int_{z=0^+}^{+\infty} F'(z) e^{-\frac{z}{x}} dz \right) \left(\int_{w=0}^{+\infty} G(w) e^{-\frac{w}{x}} dw \right)}{x} + F(0) \mathcal{E}G(x) \\
 &= \left(\int_{z=0^+}^{+\infty} F'(z) e^{-\frac{z}{x}} dz \right) \left(\int_{w=0}^{+\infty} G(xw) e^{-w} dw \right) + F(0) \mathcal{E}G(x) \\
 &= \left(\int_{z=0^+}^{+\infty} F'(z) e^{-\frac{z}{x}} dz \mathcal{E}G(x) \right) + F(0) \mathcal{E}G(x) \\
 &= \left(\left(\int_{z=0^+}^{+\infty} F'(z) e^{-\frac{z}{x}} dz \right) + F(0) \right) \mathcal{E}G(x) \\
 &= \left(\left(\int_{z=0^+}^{+\infty} e^{-\frac{z}{x}} dF(z) \right) + F(0) \right) \mathcal{E}G(x) \\
 &= \left(\left((e^{-\frac{z}{x}} F(z)) \Big|_0^{+\infty} - \int_{z=0}^{+\infty} F(z) d e^{-\frac{z}{x}} \right) + F(0) \right) \mathcal{E}G(x) \\
 &= \left(\left(\int_{z=0}^{+\infty} F(z) \frac{e^{-\frac{z}{x}}}{x} dx \right) \mathcal{E}G(x) \right) \\
 &= \left(\int_{z=0}^{+\infty} F(xz) \frac{e^{-z}}{x} d(xz) \right) \mathcal{E}G(x) \\
 &= \left(\int_{z=0}^{+\infty} F(xz) e^{-z} dz \right) \mathcal{E}G(x) \\
 &= (\mathcal{E}F(x))(\mathcal{E}G(x))
 \end{aligned} \tag{7}$$

□

将推导过程倒过来可知其逆命题也成立, 而实际上有:

引理 7.2. $\mathfrak{I}f(x) = \mathcal{E}f(x)$ 。

证明.

$$\begin{aligned}
 \Im f(x) &= \sum_{i=0}^n a_i i! a^i \\
 &= \sum_{i=0}^n a_i x^i \Gamma(i+1) \\
 &= \sum_{i=0}^n a_i \int_{y=0}^{+\infty} (xy)^i e^{-y} dy \\
 &= \sum_{i=0}^n a_i (xy)^i \int_{y=0}^{+\infty} y^i e^{-y} dy \\
 &= \int_{y=0}^x \sum_{i=0}^n a_i (xy)^i e^{-y} dy \\
 &= \int_{y=0}^x f(xy) e^{-y} dy \\
 &= \mathcal{E}f(x)
 \end{aligned} \tag{8}$$

□

实际上这告诉了我们积分多项式算子实际上就是指数加权积分变换算子的一种特殊形式，它实际上对应着非负域多项式的指数加权积分变换，对于一般的非多项式的非负域可导函数实际上可以采用积分多项式同样的方法使用指数加权积分变换进行解决。

7.2 实际应用

由此可见指数加权积分变换可以做一些描述内部的测度的函数非多项式的连续型随机变量问题，在上文中我们探讨了连续型随机变量中整数差分约束的类型，它是线性的，这里看一个使用指数加权积分变换解决非线性的连续型随机变量问题的例子：

例题 7.1 (随机¹⁰). 有一个由 n 个随机变量组成的序列 x ，每一个变量在 $[0, 1]$ 之间均匀随机生成，求 $\sum_{i=1}^n x_i^2 \leq 1$ 的概率，当答案和参考答案的绝对误差不超过 10^{-11} 时被认为是正确的。

$$1 \leq n \leq 10^5.$$

和积分多项式的处理方法相似，定义 $f(x)$ 为描述一个随机变量的平方小于等于其的内部测度的函数，则有 $f(x) = x^{\frac{1}{2}}$ ，要求的结果即为 $f^{[n]}(1)$ 。

$$\text{有 } \mathcal{E}f(x) = \int_{y=0}^{+\infty} (xy)^{\frac{1}{2}} e^{-y} dy = x^{\frac{1}{2}} \int_{y=+\infty}^x y^{\frac{1}{2}} e^{-y} dy = x^{\frac{1}{2}} \Gamma(\frac{3}{2}).$$

$$\text{那么有 } \mathcal{E}f^{[n]}(x) = (\mathcal{E}f(x))^n = x^{\frac{n}{2}} \Gamma^n(\frac{3}{2}).$$

考虑令 $f^{[n]}(x) = ax^d$ ，则有 $\mathcal{E}f^{[n]}(x) = \int_{y=0}^{+\infty} a(xy)^d e^{-y} dy = ax^d \int_{y=+\infty}^x y^d e^{-y} dy = ax^d \Gamma(d+1)$ ，可以得到 $d = \frac{n}{2}, a = \frac{1}{\Gamma(\frac{3}{2}+1)}$ ，则 $f^{[n]}(x) = \frac{\Gamma^n(\frac{3}{2})}{\Gamma(\frac{n}{2}+1)} x^{\frac{n}{2}}$ 。

¹⁰ 原创

而有 $\Gamma(\frac{1}{2}) = \sqrt{\pi}$ ，根据 $\Gamma(x+1) = \Gamma(x)x$ 可得当 n 为奇数时 $\Gamma(\frac{n}{2}) = \sqrt{\pi} \frac{(n-2)!!}{2^{\frac{n-1}{2}}}$ 。

则当 n 为奇数时 $f^{[n]}(x) = \frac{(\frac{\sqrt{\pi}}{2})^n}{\sqrt{\pi} \frac{n!!}{2^{\frac{n+1}{2}}}} x^{\frac{n}{2}} = \frac{\pi^{\frac{n-1}{2}}}{2^{\frac{n-1}{2}} n!!} x^{\frac{n}{2}}$ ，当 n 为偶数时 $f^{[n]}(x) = \frac{(\frac{\sqrt{\pi}}{2})^n}{(\frac{\sqrt{\pi}}{2})!} x^{\frac{n}{2}} = \frac{\pi^{\frac{n}{2}}}{2^n (\frac{n}{2})!} x^{\frac{n}{2}}$ 。

将 $x = 1$ 代入即可得到答案为 $\begin{cases} \frac{\pi^{\frac{n-1}{2}}}{2^{\frac{n-1}{2}} n!!} & (n \text{ 是奇数}) \\ \frac{\pi^{\frac{n}{2}}}{2^n (\frac{n}{2})!} & (n \text{ 是偶数}) \end{cases}$ ，实际上将偶数的部分改写为双阶乘

的形式可以将其化简为 $\frac{(\frac{\pi}{2})^{\frac{n+1}{2}}}{n!!}$ 。

当 $n > 25$ 时可以发现结果都 $< 10^{-11}$ 了，所以当 $n > 25$ 时可以直接输出 0，对 $n \leq 25$ 的进行计算即可，实验表明使用 `double` 类型误差是就是可以接受的。

实际通过对该题目的推导不难得到在高维空间中“球体”的“体积”公式为 $V = \frac{\pi^{\frac{1}{2}} 2^{\frac{1}{2}} \Gamma^{\frac{1}{2}}}{n!!}$ 。

将 $n = 1, 2, 3$ 代入发现分别为 $2r, \pi r^2, \frac{4}{3}\pi r^3$ ，这与我们的认知是相符合的。

8 总结

本文对连续型随机变量的相关问题进行了简单的介绍，从最基础的仅存在偏序关系的类型引入，然后分析整数差分约束的类型，介绍了解决该种类型比较主流的转化为整数部分和小数部分的方法与将约束关系转化为树的方法，之后介绍了笔者在 PKUWC 2021 逛街一题中发现的分段积分多项式方法。

实际上能够牵扯到的问题还有很多：例如整数差分约束类型实际上是比较弱的，对于一般的线性规划类型能做到怎样的最优复杂度呢？不难发现 $d = 2$ 时等价于半平面交问题，实际上这是一个和计算几何相关的问题；又例如本文中连续型随机变量的个数是有限的，但在具有无限个连续型随机变量的随机过程中又会得到怎样的结果呢？当允许较大误差时原问题又难以精确解决时还可以采用蒙特卡诺方法进行估计等等。介于篇幅和笔者能力都有限，在此不一一展开，感兴趣的同学可以自行进行研究。

9 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练杨耀良，彭思进，张艺缤的指导。

感谢家人对我的陪伴与支持。

感谢长郡中学谭献龙老师的关心与指导。

感谢长郡中学 2025 届信息组的同学给予我的帮助与陪伴。

感谢杨鑫和同学、彭思进学长为本文验稿并提出了对本文宝贵的修改意见。

感谢其他给予我帮助的老师与同学。

10 参考文献

参考文献

- [1] Probability distribution, https://en.wikipedia.org/wiki/Probabilitydistribution#Continuous_probability_distribution
- [2] Random variable, https://en.wikipedia.org/wiki/Random_variable
- [3] 李白天《排列计数题解》, <https://www.luogu.com.cn/article/r4zu3vh1>
- [4] 戴江齐, 许庭强《反重: 求熵题解》, <https://qingyu.blog.uoj.ac/blog/8549>
- [5] Editorial of AGC020F Arcs on a Circle, <https://img.atcoder.jp/agc020/editorial.pdf>
- [6] Convolution, <https://en.wikipedia.org/wiki/Convolution>
- [7] 李白天《信息学竞赛中的生成函数计算理论框架》2021 年信息学奥林匹克中国国家队候选队论文集, 2021

《无限地狱》命题报告

福建省福州第一中学 陈旭磊

摘要

本文介绍了笔者为 IOI2025 国家集训队互测命制的题目《无限地狱》。这是一道拥有数论背景的组合计数问题, 同时考察了选手对动态规划的设计能力与优化能力, 具有一定的难度。

1 题目描述

将 $1 \sim n$ 的所有整数划分成三个集合, 要求任意两个属于不同集合的元素之和不在剩下的那个集合之内。集合之间是无序的。

求方案数, 对 998244353 取模。

2 数据范围

Subtask 1(4%): $n \leq 10$ 。

Subtask 2(13%): $n \leq 40$ 。

Subtask 3(17%): $n \leq 3000$ 。

Subtask 4(21%): $n \leq 10^6$ 。

Subtask 5(22%): $n \leq 10^9$ 。

Subtask 6(23%): $n \leq 2 \times 10^{10}$ 。

3 指数级复杂度的算法

3.1 暴力搜索

暴力枚举集合划分, 并枚举所有数对是否符合条件, 从而判断是否满足题意。

朴素实现的时间复杂度为 $O(3^n n^2)$, 可以通过子任务 1, 期望得分 4 分。

3.2 对暴力搜索的一个优化

进行剪枝，如果已经不满足题意就直接终止搜索过程。

此时代码的运行速度没有质的改变，原因是若允许含空集，答案相当大，而这部分答案为 2^{n-1} 。

对于三个集合都非空的情况，方案数则并不多。可以先钦定三个集合里的某个数，再结合剪枝进行搜索。

可以通过子任务 2，期望得分 17 分。

4 $O(n^2)$ 的算法

4.1 定义

不妨设三个集合均非空，且三者按照最小元素的值升序排序后依次为 A, B, C 。

设 $a_i (1 \leq i \leq |A|)$ 为 A 中元素升序排序后的第 i 个元素。同理于 B, C 。

设 $g = \gcd(c_1, c_2, \dots, c_{|C|})$ 。

4.2 A, B, C 所满足的性质

引理 1.

对于 $1 \leq i, j \leq |C|$ ，若 $x \not\equiv 0 \pmod{g}$ 且 $x \in A$ ，有 $x + (c_j - c_i) \in A$ （如果在值域内）。同理于 B 。

证明.

若 $x < c_i$ ，有 $c_i - x \in A$ ，又 $c_j - (c_i - x) \in A$ 。

若 $x > c_i$ ，有 $x - c_i \in A$ ，又 $x - c_i + c_j \in A$ 。

引理 2.

对于 $\forall a + b \leq n$ 且 g 整除 a 与 b ，若 $\forall x \not\equiv 0 \pmod{g}$ 且 $x \in A$ ，有 $\forall y \equiv x \pmod{a}$ 与 $y \equiv x \pmod{b}$ 均满足 $y \in A$ 。那么 $\forall y \equiv x \pmod{\gcd(a, b)}$ ，有 $y \in A$ 。同理于 B 。

证明.

进行如下的一个构造过程。若 $x > b$ ，将其变为 $x - b$ ，反之变为 $x + a$ 。由于 $a + b \leq n$ ，这是一定可以操作的。

这相当于在模 b 意义下，合并 x 与 $x + a$ 的两个等价类。由裴蜀定理 [1] 可知，在模 $\gcd(a, b)$ 的意义下，同一等价类中的两个数在 A 中的存在性是一致的。

定理 1.

对于 $\forall x \not\equiv 0 \pmod{g}$ 且 $x \in A$ ，则 $\forall y \equiv x \pmod{g}$ ，有 $y \in A$ 。同理于 B 。

证明.

使用数学归纳法，以下均认为 $x \not\equiv 0 \pmod{g}$ 。

对于 c_1 ， $\forall y \equiv x \pmod{c_1}$ 显然 $\in A$ 。

对于 $1 \leq i < |C|$ ，设 $g' = \gcd(c_1, c_2, \dots, c_i)$ 。假设若 $\forall y \equiv x \pmod{g'}$ ，有 $y \in A$ 成立，尝试证明对于 $i + 1$ 也成立。

设 $d = c_{i+1} - c_i$ ，由引理 1 可知，对于 $\forall y \equiv x \pmod{d}$ ，有 $y \in A$ 。

由于 $g' + d \leq n$ ，由引理 2 可知，对于 $\forall y \equiv x \pmod{\gcd(g', d)}$ ，有 $y \in A$ 。

而 $\gcd(g', d) = \gcd(c_1, c_2, \dots, c_{i+1})$ ，于是我们从 i 成立得到了 $i + 1$ 成立。

推论 1.

对于 $\forall 1 \leq i < |C|$ ，若 $x \not\equiv 0 \pmod{\gcd(c_1, c_2, \dots, c_i)}$ 且 $x \in A$ ，则对于 $\forall y \equiv x \pmod{\gcd(c_1, c_2, \dots, c_i)}$ 且 $1 \leq y < c_{i+1}$ ，有 $y \in A$ 。同理于 B 。

证明.

若一个集合没有出现矛盾，则他的子集也不会出现矛盾。

取出所有 $< c_{i+1}$ 的数，要求这些数之间没有矛盾，于是这便是一个可以应用定理 1 的子问题。

定理 2.

有 $g > 1$ 恒成立。

证明.

仍然考虑定理 1 证明中的归纳过程。

显然有 $c_1 > 1$ 。

现证若对于 $1 \leq i < |C|$ ，假设有 $g' > 1$ ，则 $\gcd(g', c_{i+1}) > 1$ 。

使用反证法，假设 $g' > 1$ 且 $\gcd(g', c_{i+1}) = 1$ 。下述过程若无特殊说明，均默认值域 $< c_{i+1}$ 。

若对于 $\forall x \in B$ ，有 $x \equiv 0 \pmod{g'}$ 成立。则 $\forall x \not\equiv 0 \pmod{g'}$ ，有 $x \in A$ 。

由推论 1 知 $d \pm kg' \in A$ ，同时若 $c_{i+1} - (d \pm kg') \notin C$ ，则其属于 A 。

又 $c_{i+1} - (d \pm kg') = c_i \pm kg'$ ，故 A, C 可以取遍所有满足 $x < c_{i+1}$ 且 $x \equiv 0 \pmod{g'}$ 的 x 。于是 $b_1 > c_{i+1}$ ，这与 B, C 的顺序矛盾。

另一方面，若 $\exists x \in B$ ，使得 $x \not\equiv 0 \pmod{g'}$ ，则对 $\forall y \equiv x \pmod{g'}$ 与 $y \equiv -x \pmod{g'}$ ，都有 $y \in B$ 。

又对 $\forall y \equiv -(c_{i+1} - x) \pmod{g'}$ 与 $y \equiv c_{i+1} - (-x) \pmod{g'}$ ，都有 $y \in B$ （若 $y \not\equiv 0 \pmod{g'}$ ）。

这相当于在模 g' 意义下，若 x 与 $x \pm c_{i+1}$ 均 $\not\equiv 0$ ，则他们都属于 B 。

因为 $\gcd(g', c_{i+1}) = 1$ ，于是这可以取完模 g' 意义下的除了 0 以外的所有等价类。
又由于 $1 \in A$ ，故出现矛盾。

定理 3.

有 $\gcd(b_1, b_2, \dots, b_{|B|})$ 整除 g 。

证明.

若对 $\forall x \in B$ ，都有 $x \equiv 0 \pmod{g}$ ，则由于 B, C 间的大小关系，有 $g \in B$ 。

反之，则一定存在 $x < g$ ，使得 $x \in B$ 。则有 $g - x \in B$ ，而 $\gcd(x, g - x) = \gcd(x, g)$ 。

4.3 动态规划的设计

通过以上结论，我们可以知道，对于 $\forall x \not\equiv 0 \pmod{g}$ ，唯一的限制是 $\forall y \equiv x \pmod{g}$ 与 $y \equiv -x \pmod{g}$ 需要和他在一个集合内。

对于剩余部分，限制是由他们内部带来的。由于 $g > 1$ ，我们可以先将其除以 g 再考虑。发现这类似一个规模更小的子问题，但限制会有所不同。如原先的 C 内要满足 $\gcd = 1$ ，可以为空集等。

现在，我们已经发现了足够多的这个结构所具有的性质，尝试设计动态规划来解决这个问题。

设 $f(n)$ 表示总方案数。考虑分类讨论计算贡献。

考虑保留所有 g 的倍数，并将他们除以 g ，得到 A', B', C' 。此时 C' 内的 $\gcd$ 应为 1。

当 B 中所有数都为 g 的倍数时，这时 A' 可为空集，或者满足顺序 $B'_1 < C'_1 < A'_1$ （由于定理 2 的限制）。

设 $h(n)$ 表示这种情况下（除以 g 之后，区分三个集合）的方案数，这一部分对 $f(n)$ 的贡献为 $\sum_{d=2}^n h(\lfloor \frac{n}{d} \rfloor)$ 。

考虑 $h(n)$ 的转移式，枚举 C' 的公约数 d ，使用莫比乌斯反演 [2] 消去 $\gcd = 1$ 的限制。

对于 A' 为空集的情况，唯一的要求是 $1 \in B$ 。贡献为 $-2^{n-1} + \sum_{d=1}^n \mu(d) \times (2^{\lfloor \frac{n}{d} \rfloor} - 1)$ 。

对于 A' 非空的情况，根据定理 3，此时 A' 中的所有数必然是 d 的倍数，即非 d 的倍数的数一定在 B' 中，于是可以将所有数先除以 d 再判断合法性。

若 B' 中不含 d 的倍数，则只需满足 C', A' 之间的大小关系，反之则是 $f(n)$ 所定义的结构。这一部分的贡献为 $-2f(n) - (2^{n-1} - 1) + \sum_{d=1}^n \mu(d) \times [3f(\lfloor \frac{n}{d} \rfloor) + (2^{\lfloor \frac{n}{d} \rfloor - 1} - 1)]$ 。

这两种情况求和号外面的部分均是来自于 $d = 1$ 时的特殊情况，这时 1 为 d 的倍数。

综上，有 $h(n) = -2f(n) - (2^n - 1) + \sum_{d=1}^n \mu(d) \times [3f(\lfloor \frac{n}{d} \rfloor) + 3 \times 2^{\lfloor \frac{n}{d} \rfloor - 1} - 2]$ 。

当 B 中存在不为 g 的倍数时，此时 A', B' 均可为空集，且 C' 不能作为最小值最大的元素（若均非空）。

设 $g(n)$ 为这部分的方案数。唯一的限制是 $1 \in A$ ，则对 $f(n)$ 的贡献为 $\sum_{d=2}^n (2^{\lfloor \frac{n}{d} \rfloor - 1} - 1)g(\lfloor \frac{n}{d} \rfloor)$ 。

转移推导和 $h(n)$ 是类似的，仍然通过莫比乌斯反演消除 $\gcd = 1$ 的限制。

对于 A', B' 均为空的情况，方案数为 1。

对于 A', B' 中存在恰好一个空集的情况，方案数为 $-2 + 2 \sum_{d=1}^n \mu(d) \times (2^{\lfloor \frac{n}{d} \rfloor} - 1)$ 。

对于 A', B' 均非空的情况，仍然将较大的两个集合除以 d 后再判断合法性，并讨论较小的集合是否存在 d 的倍数。

这时方案数为 $-2f(n) - (2^n - 2) + 2 \sum_{d=1}^n \mu(d) \times [3f(\lfloor \frac{n}{d} \rfloor) + (2^{\lfloor \frac{n}{d} \rfloor - 1} - 1)]$ 。

综上，有 $g(n) = -2f(n) - (2^n - 1) + 2 \sum_{d=1}^n \mu(d) \times [3f(\lfloor \frac{n}{d} \rfloor) + 3 \times 2^{\lfloor \frac{n}{d} \rfloor - 1} - 2]$ 。

如果我们暴力模拟 $f(n), g(n), h(n)$ 的递推式，则复杂度为 $O(n^2)$ 。可以通过子任务 3，期望得分 34 分。

5 对 $O(n^2)$ 算法的优化

5.1 一个快速求单点点值的算法

对于莫比乌斯函数的前缀和 $s(n) = \sum_{i=1}^n \mu(i)$ ，我们有经典等式 $s(n) = 1 - \sum_{d=2}^n s(\lfloor \frac{n}{d} \rfloor)$ （当 $n \geq 1$ 时）。

于是可以使用数论分块快速转移与找到有用状态，所有转移对的数量为：

$$\sum_{i=1}^{\sqrt{n}} O(\sqrt{i}) + O\left(\sqrt{\frac{n}{i}}\right) = O\left(\int_0^{\sqrt{n}} \left(\sqrt{x} + \sqrt{\frac{n}{x}}\right) dx\right) = O(n^{\frac{3}{4}})$$

如果使用记忆化实现则需要对所有状态预处理 2 的次幂等需要的系数，不然复杂度会多一个 $\log n$ 。

可以通过子任务 5，期望得分 77 分。

5.2 一个快速求前缀所有点值的算法

考虑 $f(n)$ 与 $f(n-1)$ 转移的差异。对于一个确定的 d ，若 $\lfloor \frac{n}{d} \rfloor$ 与 $\lfloor \frac{n-1}{d} \rfloor$ 相等，则 d 的贡献时相同的。即只有 d 整除 n 时才会产生不同的贡献。

于是可以对 $f(n)$ 的差分进行动态规划转移，对 $g(n)$ 以及 $h(n)$ 也做相同的处理。

此时所有转移对的数量为：

$$\sum_{i=1}^n O\left(\frac{n}{i}\right) = O\left(n \int_1^n \frac{1}{x} dx\right) = O(n \ln n)$$

可以通过子任务 4，期望得分 55 分。

5.3 更优秀的算法

使用杜教筛 [3] 的思想，若我们取 m 为阈值 ($m \geq \sqrt{n}$)，则时间复杂度为

$$O(m \ln m) + \sum_{i=1}^{\lfloor \frac{n}{m} \rfloor} O\left(\sqrt{\frac{n}{i}}\right) = O\left(m \ln m + \int_0^{\lfloor \frac{n}{m} \rfloor} \sqrt{\frac{n}{x}} dx\right) = O\left(m \ln m + \frac{n}{\sqrt{m}}\right)$$

取 $m = \frac{n^{\frac{2}{3}}}{\ln^{\frac{1}{3}} n}$ 时，得到时间复杂度的最小值为 $O(n^{\frac{2}{3}} \ln^{\frac{1}{3}} n)$ ，可以通过子任务 6，期望得分 100 分。

6 致谢

感谢中国计算机学会提供学习与交流的平台。

感谢吴俊书学长率先发现此问题，并在命题过程中给予我许多帮助。

感谢厦门双十中学李静榕同学为本题验题。

参考文献

- [1] [Wikipedia, Bézout's identity](#)
- [2] [Wikipedia, Möbius inversion formula](#)
- [3] [OI Wiki, 杜教筛](#)

《数据结构》解题报告

长沙市长郡中学 朱屹帆

摘要

本文结合笔者的集训队互测，针对一类树上邻域范围相关的问题，提出基于树链剖分传统算法的思考、改进和优化。

1 题目描述

给定一棵 n 个节点的树，钦定点 1 为根，每条边的边权均为 1，节点 x 有点权 a_x ，初始所有节点的点权均为 0。总共有 m 次操作，每次操作先给出操作类型 opt ，接下来给出该操作的所有参数。

1. 给定一条路径 (x, y) 和范围 k 以及权值 u ，对于所有点 i ，当点 i 满足其到树上路径 (x, y) 的最短距离 $\leq k$ 时，有 $a_i \leftarrow a_i + u$ 。
2. 给定点 x 以及权值 u ，对于所有点 x 子树内的节点 i ，有 $a_i \leftarrow a_i + u$ 。
3. 给定一条路径 (x, y) 和范围 k ，对于所有点 i ，当点 i 满足其到树上路径 (x, y) 的最短距离 $\leq k$ 时，询问所有这样的点 i 的点权和。
4. 给定点 x ，对于所有点 x 子树内的节点 i ，询问所有这样的点 i 的点权和。
5. 给定一条路径 (x, y) 和范围 k ，对于所有点 i ，当点 i 满足其到树上路径 (x, y) 的最短距离 $\leq k$ 时，询问所有这样的点 i 的点权 $\max$ 。
6. 给定点 x ，对于所有点 x 子树内的节点 i ，询问所有这样的点 i 的点权 $\max$ 。

其中编号对应操作类型，操作的所有参数将以上述描述的变量顺序给出。

2 数据范围

本题开启子任务评测，子任务之间将会设置合理的依赖关系。

子任务 1 和子任务 2 (10 分 + 10 分): 保证 $k = 0$ 。

子任务 3 和子任务 4 (10 分 + 10 分): 保证所有涉及路径 (x, y) 的操作满足 $x = y$ 。

子任务 5 和子任务 6 (10 分 + 10 分): 保证 $k = 1$ 。

子任务 7 和子任务 8 (20 分 + 20 分): 无特殊限制。

上述子任务中所有奇数编号子任务不包含操作五和操作六。

对于所有数据，保证 $1 \leq n, m \leq 2 \times 10^5$ ， $1 \leq k \leq 3$ ， $-10^5 \leq u \leq 10^5$ 。

时间限制 7 s，空间限制 512 MB。

3 算法一

当 $k = 0$ 时，使用树链剖分维护链和子树的修改与查询，时间复杂度 $O(n \log^2 n)$ ，期望得分 10 到 20 分。

4 算法二

当保证操作一、操作三和操作五的路径 (x, y) 均满足 $x = y$ ，且不包含操作五和操作六时，考虑对操作序列进行根号重构，并分析前面的所有修改操作对后续的查询操作的影响。通过分析可以发现修改操作对查询操作的贡献即为两者操作的范围交的点的数量乘以修改操作对应权值。

操作一对操作三的贡献即为两个单点的对应邻域交的大小乘以修改权值，树上任意两个单点的邻域交必然可以表示成一个点的邻域或一条边的中点的邻域，而找到这个中点是容易的，由于 k 的数据范围较小，因此两个单点只有在树上距离 $\leq 2k$ 时邻域交不为空，此时可以取出两个单点在树上的路径并找到对应中点。

操作一对操作四的贡献即为一个单点的邻域和一个单点的子树的交的大小乘以修改权值，通过类似 dp 的方式预处理单点子树内外邻域的一些信息，随后即可根据修改单点和查询单点的相对位置关系快速得到交的大小。

操作二对操作三的贡献即为一个单点的子树和一个单点的邻域的交的大小乘以修改权值，类似操作一对操作四的贡献分析。

操作二对操作四的贡献即为两个单点的子树交的大小乘以修改权值，这是简单的，根据两个单点的祖先后代关系确定子树交得到的新的子树。

可以发现上述贡献分析在根号重构时是相对容易维护的，使用类似 dp 的方式下放修改信息即可。

具体细节可以见参考资料中笔者的 2023 年集训队互测《数据结构》解题报告。由于查询 max 信息并不支持将修改操作拆分后类似的考虑，因此该做法似乎无法拓展到维护操作五和操作六。时间复杂度 $O(n \sqrt{n} \times \text{poly}(k))$ ，结合算法一，期望得分 30 分。

5 算法三

注意到算法二的核心思想在于分析修改对询问的贡献，不妨尝试换一种思考方向，当 $k = 1$ 且不存在操作五和操作六时，得到类似 $k = 0$ 的处理方法。

简单回顾算法一，当 $k = 0$ 时，根据轻重链剖分的性质，从任意一个点开始不断跳到父节点，直至根节点时停止，在这个过程中至多会跳 $\log n$ 条轻边，因此可以将一条路径拆分成 $\log n$ 条重链，并且由于在树链剖分的 dfs 序中，重链上的节点的标号具有连续性，因此可以使用数据结构维护。这启发我们可以尝试在保持轻重链剖分对 $\log n$ 条重链进行处理以代替对一条路径进行处理的性质的基础上，通过调整重链的标号方式，使得重链邻域节点的标号大致连续。

笔者将该思想称为“基于调整搜索顺序的重标号树链剖分”，具体细节可以见参考资料中笔者 2024 年的集训队论文。接下来将介绍一种能够处理上述模型的标号方式。

首先采用传统树链剖分的标号方式中的顺序考虑每一条重链。对于一条重链，首先将该重链中的所有点按照深度从小到大依次进行标号，接下来仍然以深度从小往大的顺序扫描该重链的所有节点，对于当前扫描的节点 x ，将点 x 的所有轻儿子进行标号。需要注意的是由于该标号方式会对一条重链及其一度邻域进行标号，因此可能会出现在标号过程中一个节点已经被标号的情况，此时不改变该节点的原有标号。接下来将会说明该标号方式的部分性质以及基于该标号方式的维护方式。

可以发现该标号方式仍然保持了树链剖分的许多优秀性质，例如可以将一条路径大致定位到 $\log n$ 条重链上进行处理，更为关键的是该标号方式使得树的一度邻域信息在标号中具有了一些连续性，且标号过程中出现节点已经被标号的情况仅有可能在一条重链的链顶发生，因为对点 x 的轻儿子进行标号的过程仅会影响到其他重链的链顶节点。

对于操作一和操作三，取出一条重链上点 x 到点 y 的路径一度邻域信息，不妨有点 x 为点 y 祖先节点，则一度邻域路径即为点 x 一度邻域最小标号到点 y 一度邻域最大标号构成的标号区间，此时由于标号的特殊的连续性，因此可以使用数据结构进行维护。对于操作二和操作四，注意到上述标号方式仍然保证了类似子树的标号性质，因此将子树对应标号区间取出后类似的数据结构维护即可。上述分析中由于存在重链链顶节点标号不与重链其他节点保持连续性，因此需要补充重链链顶以及重链区间末尾端点的重儿子的贡献，这是简单的细节部分。

具体细节可以见参考资料中笔者的 2023 年集训队互测《数据结构》解题报告。时间复杂度 $O(n \log^2 n)$ ，结合算法一，期望得分 30 分。

6 算法四

可以发现算法三似乎具有极大的拓展空间，不妨尝试将其拓展到更加经典的模型中，当不存在操作五和操作六时，得到足够优秀的处理方法。考虑进一步优化标号方式，不妨设 $a = 3$ 。

采用传统树链剖分的标号方式中的顺序考虑每一条重链。对于一条重链，首先将该重链中的所有点按照深度从小到大依次进行标号，接下来循环以下过程 a 轮：对于第 i 轮，以深度从小往大的顺序扫描该重链的所有节点，对于当前扫描的节点 x ，将点 x 的所有轻儿子子树内距离点 x 恰好为 i 的所有节点进行标号。值得注意的是该标号方式会对一条重链及其 a 度邻域内的所有点进行标号，因此可能会出现在标号过程中一个节点已经被标号的情况，此时不改变该节点的原有标号。接下来将会说明基于该标号方式的维护方式。

对于四种类型的操作，大致的重链邻域信息以及子树信息的维护是相对容易的。但可以发现，此时一条重链顶端的 a 层内的所有节点可能会和重链的点的标号不相关，而这 a 层涉及到的标号不相关的节点数量并没有理论限制，因此接下来将分析重链顶端这 a 层的点的信息所具备的优良性质。

定义 6.1. 对于树上的节点 x ，维护线段集合 $s_0\{x, k\}$ 表示与点 x 距离恰好等于 k 的、在点 x 子树内的所有节点构成的标号区间（当 $k > a$ 时描述与点 x 的所有距离 $> a$ 的点的信息）；

定义 6.2. 对于树上的节点 x ，维护线段集合 $s_1\{x, k\}$ 表示与点 x 距离恰好等于 k 的、在点 x 的轻儿子的子树内的所有节点构成的标号区间（当 $k > a$ 时描述与点 x 的所有距离 $> a$ 的点的信息）。

可以发现维护出具体的 $s_0\{x, k\}(0 \leq k \leq a+1)$ 和 $s_1\{x, k\}(0 \leq k \leq a+1)$ 是容易的，只需要在完成标号后类似 dp 的方式转移处理即可。接下来称标号区间为线段。

性质 6.1. 对于树上任意节点 x ，可以发现 $s_1\{x, k\}$ 均满足线段数量是 $O(1)$ 的，具体来说满足线段数量 ≤ 2 。

证明. 对于任意节点 x 而言，当 $0 \leq k \leq a$ 时，根据上述标号方式，与其距离恰好为 k 的轻儿子节点必然会在同一时刻被连续标号，而点 x 子树内重链上距离恰好为 k 的点至多只有一个，该节点可能会将连续标号划分成两段（且简单分析可以发现只有点 x 的重儿子可能发生这种情况），因此线段数量 ≤ 2 ；而当 $k = a+1$ 时，显然标号是完全连续的，因此线段数量 ≤ 1 。□

性质 6.2. 对于树上任意节点 x ，可以发现 $s_0\{x, k\}$ 均满足线段数量是 $O(a)$ 的，具体来说满足线段数量 $\leq a+1$ 。

证明. 对于任意节点 x 而言，当 $0 \leq k \leq a$ 时，节点 x 子树内的与其距离恰好为 k 的所有节点可以被表示成节点 x 及其重链对应位置的若干后继节点的 $s_1\{x, k'\}$ 的并集，则可以得到 $O(a)$ 的上界，且进一步分析可以发现对于重儿子来说，每一层均满足标号的连续性，因此即使每一层的标号之间都是独立的，也仍然满足线段数量 $\leq a+1$ 。而当 $k = a+1$ 时，重链的标号会将轻儿子子树的连续标号划分，因此线段数量 ≤ 2 。□

因此修改单点的轻儿子子树范围内的距离 $\leq k$ 的点时，线段数量为 $O(a)$ ；修改单点的子树范围内的距离 $\leq k$ 的点时，线段数量为 $O(a^2)$ 。

对于操作一和操作三，取出一条重链上点 x 到点 y 的路径 k 度邻域信息，不妨有点 x 为点 y 祖先节点，则 k 度邻域路径即为点 x 的 k 度邻域最小标号到点 y 的 k 度邻域最大标号构成的标号区间，此时由于标号的特殊的连续性，因此可以使用数据结构维护。对于操作二和操作四，注意到上述标号方式仍然保持了类似子树的标号性质，取出子树对应标号区间后类似的数据结构维护即可。由于一条重链的前 a 层的标号与重链本身的标号并不相关，因此需要对于前 a 层进行特殊处理，不妨根据预处理得到的 $s_0\{x, k\}$ 和 $s_1\{x, k\}$ 信息，即在处理一条重链时，对重链区间右端点的重儿子子树以及对重链顶端的 k 层邻域进行处理。需要注意的是当涉及到处理路径 (x, y) 时，对于路径的最近公共祖先之上的部分，需要使用类似容斥的技巧补充贡献。子树的前 a 层的点的标号与子树内的点的标号并不相关，因此处理子树时需要对于前 a 层单独处理，类似的利用 $s_0\{x, k\}$ 信息取出前 a 层的标号区间即可。

时间复杂度 $O(\text{poly}(a) \times n \log^2 n)$ ，上述做法大致可以做到 $O(a^2 \times n \log^2 n)$ ，精细实现可以做到 $O(a \times n \log^2 n)$ ，常数较大，结合算法一，期望得分 60 分。

7 算法五

可以发现算法四似乎仍然具备极大的拓展空间，在算法四中，特殊的标号方式保持着每个点对应唯一标号的优秀性质，即点权修改操作并没有进行拆分，而是以统一的整体形式作用到每一个点上。但算法四并不能直接查询路径 (x, y) 的 $\max$ 信息，因为在查询过程中采用了类似容斥的技巧补充贡献，而 $\max$ 信息不支持简单形式的容斥得到信息。

不妨将操作五类似操作的取出路径查询的所有区间及其贡献系数，加和信息可以直接将所有区间和乘以对应贡献系数加和得到，但查询 $\max$ 信息需要规避容斥的形式。考虑扫描线得到所有产生有效贡献的区间，将所有区间差分并在左端点加入其贡献系数，右端点后继减去其贡献系数，则扫描线过程中所有贡献系数为正数的位置即为产生有效贡献的所有位置，这是可以扫描线轻松维护的。

得到产生有效贡献的所有区间后，此时由于每个点对应唯一标号，因此查询所有区间 $\max$ 信息即可。操作六对于子树本身取出的区间并不需要容斥，直接查询即可。

时间复杂度 $O(\text{poly}(a) \times n \log^2 n)$ ，上述做法大致可以做到 $O(a^2 \log a \times n \log^2 n)$ ，精细实现可以做到 $O(a^2 \times n \log^2 n)$ ，常数较大，期望得分 100 分。

8 总结

本文介绍了基于传统树链剖分算法的、在树上邻域范围相关问题中的一类全新应用。笔者希望，本文能够激发读者的想象力和创造力，鼓励算法竞赛选手对传统算法的勇敢创新。

致谢

感谢中国计算机学会提供学习和交流的平台。
感谢父母对我的支持和鼓励。
感谢长沙市长郡中学谭献龙老师的关心和指导。
感谢国家集训队杨耀良教练、彭思进教练的指导。

参考文献

- [1] 朱屹帆. 《数据结构》. IOI2023 集训队互测, 2023.
- [2] 朱屹帆. 《数据结构》. IOI2024 集训队互测, 2024.
- [3] 朱屹帆. 浅谈一类树上范围相关问题. IOI2024 中国国家候选队论文集, 2024.

特殊等价关系下的字符串问题

杭州学军中学教育集团文渊中学 章弥炫

摘要

信息学竞赛中已经出现过一些研究非严格相等的等价关系的字符串题, 并已经出现了一些研究的比较深入的题目, 本文介绍了一些处理这种问题的方法, 并讲解了一些例题。

1 约定和记号

1.1 基础定义

令 $S[i]$ 表示 S 中的第 i 个字符, 串长用 n 表示。

令 $S[l, r]$ 表示 S 中第 l 个字符至第 r 个字符组成的字符串, 称为 S 的一个子串, 其中 $S[l, l-1]$ 表示空串。

令 $S[l, r]^R$ 表示 S 中第 r 个字符至第 l 个字符组成的字符串, 即若 $S[l, r] = s_l s_{l+1} \dots s_r$, 那么 $S[l, r]^R = s_r s_{r-1} \dots s_l$ 。

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

1.2 讨论的等价关系

定义两个字符串相等当且仅当 $S \sim T$ 。

其中 $\sim$ 表示一个等价关系, 接下来讨论的等价关系都满足如下性质:

- 对于字符串 S , 存在一个序列 $Fl(S)$ 和函数 fl , 满足 $Fl(S)$ 由从 1 到 n 遍历 S , 遍历到 i 时在末尾加上 $fl(S[1, i-1], S[i])$ 得到。并且满足 $S \sim T$ 当且仅当 $Fl(S) = Fl(T)$ 。
- 对于字符串 S , 存在一个序列 $Fr(S)$ 和函数 fr , 满足 $Fr(S)$ 由从 n 到 1 遍历 S , 遍历到 i 时在开头加上 $fr(S[i+1, n], S[i])$ 得到。并且满足 $S \sim T$ 当且仅当 $Fr(S) = Fr(T)$ 。

举例几个满足上述性质的等价关系:

1. $S \sim T$ 当且仅当, $\forall 1 \leq i \leq n, S[i] = T[i]$ 。

2. $S \sim T$ 当且仅当, $\forall 1 \leq i < j \leq n, [S[i] = S[j]] = [T[i] = T[j]]$ 。

3. $S \sim T$ 当且仅当, $\forall 1 \leq i < j \leq n, [S[i] \leq S[j]] = [T[i] \leq T[j]]$ 。

令 $G(S) = n + \sum_{2 \leq j \leq n} [fl(S[j-1, i-1], S[i]) \neq fl(S[j, i-1], S[i])]$, 之后在一些复杂度分析中会用到这个值。

2 一些常用的方法

我们先证明一个重要的引理。

引理 2.1. 对于两个字符串 S 和 T , 若 $S[1, n] \sim T[1, n]$, 则 $\forall 1 \leq i \leq j \leq n, S[i, j] \sim T[i, j]$ 。

证明. 因为 $S[1, n] \sim T[1, n]$, 所以 $fl(S[1, n]) = fl(T[1, n])$, 因此 $fl(S[1, j]) = fl(T[1, j])$, 所以 $S[1, j] \sim T[1, j]$ 。同理, 就可以得到 $S[i, j] \sim T[i, j]$ 。□

2.1 Hash

如果能够求出一个子串的 Hash 值, 那么许多问题都会简便许多。

我们扫描 i , 对于每个 $j < i$ 维护 $fl(j, i)$ 的 Hash 值, 那么我们只需要找到所有 $fl(S[j-1, i-1], S[i]) \neq fl(S[j, i-1], S[i])$ 的位置并进行区间修改即可, 用可持久化线段树维护复杂度 $O(G(S) \log n)$ 。

这样我们就可以花费 $O(\log n)$ 的时间查询一个子串的 Hash 值。

2.2 后缀排序

这里我们给出一个基于 Hash 的后缀排序方法。设 $n = 2^{k+1}$ 。

我们按照 k 从大往小扫描, 先将所有后缀的前 2^k 个字符的 Hash 值求出, 这一部分复杂度是 $O(n \log n)$, 然后把相同的合并为一组, 分成 m 组, 组内递归的排序, m 组间递归的排序。

那么我们写出复杂度, 应该是 $f(n, k) = n \log n + f(m, k-1) + \sum_{i=1}^m f(s_i, k-1)$, 其中 $\sum_{i=1}^m s_i = n$ 。

令 $f(n, k) = 2(n-1)k \log n$, 那么 $f(n, k) - (n \log n + f(m, k-1) + \sum_{i=1}^m f(s_i, k-1)) = (n-2) \log n > 0$, 因此 $f(n, k) = O(n \log^2 n)$, 我们在 $O(n \log^2 n + G(S) \log n)$ 的时间实现了后缀排序。

2.3 KMP 算法

扫描当前处理的前缀 $S[1, i]$, 维护指针 j 满足 $j < i$, 表示最长的同时作为 $S[1, i]$ 前缀和后缀的串是 $S[1, j]$, 设为 $next_i$ 。

在从 $i-1$ 扫到 i 的时候, 判断 $fl(S[1, j], S[j+1])$ 是否等于 $fl(S[i-j, i-1], S[i])$, 如果等于则 $j = j+1$, 否则令 $j = next_j$, 迭代下去。

能够发现在所述的等价关系下, 上述 KMP 算法还是成立的。

2.4 AC 自动机

给定一个 Trie 树要求建立其 AC 自动机。

记点 u 到根链上经过的字符串为 S_u , 记 $fail_u$ 表示 S_{fail_u} 是 S_u 不为 S_u 的最长后缀。

我们要先对 Trie 树进行压缩, 假设当前 u 的儿子为 $v_1 \dots v_k$, 则要将满足 $fl(S_u, S[v_i]) = fl(S_u, S[v_j])$ 的两个子树合并成一个。

为了说明这样压缩的正确性, 我们要证明一个定理。

引理 2.2. 对于字符串 S 和 T , 若 $S[1, n]c \sim T[1, n]c$, 则 $\forall 1 \leq i \leq n, fl(S[i, n], c) = fl(T[i, n], c)$ 。

证明. 因为 $S[1, n]c \sim T[1, n]c$, 所以 $S[i, n]c \sim T[i, n]c$, 所以可以得到 $fl(S[i, n], c) = fl(T[i, n], c)$ 。□

同理还可以得到如下引理。

引理 2.3. 对于字符串 S 和 T 满足 $S \sim T$, 若 $\exists 1 \leq i \leq n, fl(S[i, n], c) \neq fl(T[i, n], c)$, 则 $fl(S[1, n], c) \neq fl(T[1, n], c)$ 。

则我们只要能够快速求出函数 fl , 用可持久化线段树维护每个点的出边, 其余的部分就和普通的 AC 自动机构建相同。

例题 2.1 (回文匹配¹)。定义一个等价关系, $S \sim T$ 当且仅当, $\forall 1 \leq i < j \leq n, [S[i, j] = S[i, j]^R] = [T[i, j] = T[i, j]^R]$ 。

给定一个 n 个点的字典树, 设点 u 到根链上的字符串为 S_u , q 次询问, 每次询问 i, j , 回答 S_j 有多少个子串 $S_j[l, r]$ 满足 $S_j[l, r] \sim S_i$ 。

解法 我们能够设计函数 $fl(S[1, i-1], S[i])$ 表示最大的 j 满足 $S[i-j+1, i] = S[i-j+1, i]^R$, $fl(S[i+1, n], S[i])$ 表示最大的 j 满足 $S[i, i+j-1] = S[i, i+j-1]^R$ 。

接下来先说明这个条件是充要的。首先若 $S \sim T$, 则一定有 $fl(S) = fl(T)$ 。

若 $fl(S) = fl(T)$, 考虑以位置 i 为中心的最长回文串 $S[i-k, i+k]$, 若 $fl(S[1, i+k-1], S[i+k]) \neq 2k+1$, 则说明存在一个更长的回文后缀, 将 $S[i-k, i+k]$ 沿着这个回文后缀对称, 可以得到位置 $j < i$ 满足 $S[j-k, j+k]$ 回文, 不断递归下去, 一定存在位置满足 $fl(S[1, j+k-1], S[j+k]) = 2k+1$ 。

¹来源: 2022-2023 集训队互测 Round 12, <https://qoj.ac/problem/5166>

我们设点 i 往两侧的最长回文长度为 d_i ，初始利用 $Fl(S)$ 将每个串 $S[i - Fl(S)_i + 1, i]$ 中点上的 d 对 $Fl(S)_i$ 取 $\max$ ，然后假设已经求出了 $d_1 \dots d_{i-1}$ ，我们就可以通过枚举所有包含 i 并且中点小于 i 的回文串将 i 对称到 i' ，然后将 d_i 对 $d_{i'}$ 取 $\max$ 。

由于我们上面证明了若 $fl(S[1, i + d_i - 1], S[i + d_i]) \neq 2d_i + 1$ 则 d_i 一定能从 $d_1 \dots d_{i-1}$ 对称过来，因此这表明了我们可以从前往后计算出每个点 i 往两侧的最长回文长度。而每个点 i 往两侧的最长回文长度都相同即满足题目的条件，可以推出 $S \sim T$ 。

因此这个等价关系和题面的条件等价。

考虑对给定的字典树构建 AC 自动机，如果构建好了，那么询问相当于查询在 fail 树上，有多少个 j 到根链上的点，在 i 的子树内。那么只需要在 Trie 上进行 dfs，记点 i 在 fail 树上的 dfs 序为 dfn_i ，子树大小为 $size_i$ ，然后在 Trie 树上进入一个点的时候加入 dfn ，回溯时删去，遍历到 j 时查询 $[dfn_j, dfn_j + size_j - 1]$ 中的点数即可。

在建 AC 自动机的过程中，我们同时对 Trie 树构建回文自动机，维护 $f(u, c)$ 表示当前点 u 在回文自动机的祖先链上最深的满足存在这个结点前一个字符为 c 的串的结点，这样每次可以直接定位到当前新串的位置，复杂度非均摊，为 $O(n\Sigma)$ 。

我们对一个 AC 自动机结点，记录后面添加每个字符之后的转移，这里我们只关注添加后最长回文后缀长度不同的字符，考虑当前点 u 要添加字符 c 。我们设 $T = S_{fail_u}$ ，此时找到 $S_u c$ 最长的一个回文后缀满足长度 t 小于等于 T 的长度加 1，那么此时在 T 的转移中找，此时一定存在一个 T 的转移满足这个转移的最长回文后缀长度等于 t ，那么继承这个转移即可。为了说明一定存在这个转移，我们要证明如下引理。

引理 2.4. 对于两个串 $S[1, n]$ 和 $T[1, n]$ 满足 $S \sim T$ ，对于任意 S_{c_1} 一定存在 c_2 满足 $S_{c_1} \sim T_{c_2}$ 。

证明. 找到 S_{c_1} 的最长回文后缀，假设长度为 $k+1$ ，则 $S[n-k+1] = c_1$ ，那么令 $c_2 = T[n-k+1]$ ，此时 T_{c_2} 的最长回文后缀至少长度为 $k+1$ 。

而如果 T_{c_2} 的最长回文后缀大于 $k+1$ ，假设为 $j+1$ ，则 $T[n-j+1] = T[n-j+k] = c_2$ ，那么由于此时 $S[n-j+1, n-j+k]$ 和 $S[n-j+2, n]$ 都是回文串，可以导出 $S[n-j+1] = S[n-j+k] = S[n-k+1] = c_1$ ，那么 S_{c_1} 的最长回文后缀应该是 $j+1$ ，矛盾。 $\square$

那么我们只需要每次在回文树上二分找到长度小于等于 T 的长度的最长回文后缀，然后为当前点 u 的每个转移找到其在 $fail_u$ 中对应的转移即可，复杂度 $O(n\Sigma \log \Sigma + (n+q) \log n)$ 。 $\spadesuit$

2.5 border

我们称同时作为 S 前缀和 S 后缀的串为 S 的 border，可以证明如下定理。

定理 2.1. 对于字符串 S ，其所有 border 构成 $O(\log n)$ 个等差数列。

证明. 我们假设 x 是 border， y 是 border，并且 $x < y < 2x$ 。

由 $\forall 1 \leq i \leq j \leq x, S[i, j] \sim S[n-x+i, n-x+j]$ 和 $\forall 1 \leq i \leq j \leq y, S[i, j] \sim S[n-y+i, n-y+j]$ 可以得到 $\forall 1 \leq i \leq j \leq x, S[i, j] \sim S[i+y-x, j+y-x]$ 。

那么 $\forall 1 \leq i \leq j \leq 2x-y, S[i, j] \sim S[i+y-x, j+y-x] \sim S[n+y-2x+i, n+y-2x+j]$ ，因此 $2x-y$ 也是原串 border。

假设所有的 border 依次为 $a_1 \dots a_k$ ，满足 $a_1 < a_2 < \dots < a_k$ 。

记 $b_i = a_{i+1} - a_i$ ，那么我们有 $b_i \leq b_{i+1}$ ，否则 $a_i + b_i - b_{i+1}$ 一定是一个新的 border。

那么假设已经有了 $b_1 \dots b_k$ ，要在末尾添加 b_{k+1} ，那么若 $b_{k+1} \neq b_k$ ，则一定有 $b_{k+1} \geq b_k + b_{k-1}$ ，因此满足 $b_i \neq b_{i+1}$ 的位置 i 一定只有 $O(\log n)$ 个。

因此 $a_1 \dots a_k$ 可以划分为 $O(\log n)$ 个等差数列。 $\square$

3 后缀数据结构

3.1 一些定义

定义 3.1 (后缀 Trie). 定义将 S 的所有后缀插入至 Trie 树中得到的字典树为后缀 Trie。

定义 3.2 (后缀树). 称后缀 Trie 中有多于一个儿子的点和所有后缀结点为关键点，定义只保留关键点，将非关键点形成的链压缩成一条边形成的压缩 Trie 树为后缀树。

定义 3.3 (隐式后缀树). 称后缀 Trie 中有多于一个儿子的点和所有叶子结点为关键点，定义只保留关键点，将非关键点形成的链压缩成一条边形成的压缩 Trie 树为隐式后缀树。

在后缀 Trie 中，记 $trans_{u,c}$ 表示在结点 u 对应的字符串开头添加字符 c 后对应的结点，如果不存在设为空状态 $NULL$ 。

定义后缀树上 u 的父亲为 fa_u ，结点 u 对应的串长为 len_u 。

3.2 后缀自动机

我们修改传统的后缀自动机构建方法。

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

记录后缀 $i+1$ 对应的结点 $last$ ，新增后缀结点 np 表示 $S[i, n]$ 。

那么我们需要先找到 $S[i, n]$ 在原树中出现的最长前缀 $S[i, j]$ ，我们从 $last$ 往父亲跳，每次检查当前结点和父边对应的 $S[i+1, l] \sim S[i+1, r]$ 这些串中是否存在 $fr(S[i+1, j], c)$ 的转移，如果存在则说明找到了 j 满足条件，如果不存在，则需要为当前结点添加所有 $fr(S[i+1, j], c)$ 的转移。

假设最后我们找到了 j ，令 q 表示 $S[i, j]$ 对应的位置，假设 q 在 u 到 fa_u 这条边上（不包含 fa_u ）。如果其恰好是点 u ，那么将 np 的父亲设为 u 即可。否则，我们需要将这条树边分裂，新建一个结点 q ，将 u 和 np 的父亲设为 q ， q 的父亲设为 fa_u ，然后 q 需要保存这条边长度区间在 $len(fa_u) + 1 \sim len(q)$ 的转移， u 需要保存这条边长度区间在 $len(q) + 1 \sim len(u)$ 的转移。然后对于 $S[i + 1, j]$ 对应结点及其父亲要判断是否仍存在指向 u 的转移，同时要判断能否新加入指向 q 的转移，不断跳父亲同时按照顺序遍历转移，直到当前仅存在指向 fa_u 的转移。

定理 3.1. 后缀自动机的转移数是 $O(G(S))$ 的。

证明. 我们称一个后缀树结点及其父亲边上表示的所有边为一个等价类，其出现位置的左端点集合就是子树内的后缀结点集合。

将所有等价类画到二维平面上，二维平面的点 (x, y) 表示串 $S[n - y + 1, n - x + 1]$ 。

将每个等价类移到其最后一次出现的位置，转移边的个数相当于指向这个等价类下边界的等价类的个数求和。

注意到对于同一列，假设这一列的等价类从低到高依次为 $v_1 \dots v_k$ ，则 v_i 可以从 v_{i-1} 连接转移边。

那么我们让所有等价类连上在最后一次出现位置的所有转移边，然后剩下的下边界总长是 n ，而第一次出现位置的所有转移边总数小于等于状态数加上 $G(S)$ ，因此总转移边数是 $O(G(S))$ 的。□

我们如果用平衡树维护每条边的转移，那么修改就是新加入一个结点和分裂操作，如果忽略计算 fr 函数的复杂度，由于转移数是 $O(G(S))$ 的，状态数是 $O(n)$ 的，复杂度就是 $O(G(S) \log n)$ 。

3.3 Ukkonen 算法

Ukkonen 算法可以支持在线从前往后添加字符，维护当前字符串 S 的隐式后缀树。下面大致描述此算法的流程。

我们将每个串在隐式后缀树中的位置表示为二元组 $(active, remain)$ ，表示当前串最长的一个作为隐式后缀树结点的前缀对应的结点是 $active$ ，当前串和这个结点的距离是 $remain$ 。

令 $link_u$ 表示在隐式后缀树上的结点 u 去掉第一个字符后跳到的结点，但是这并不一定还是一个隐式后缀树结点，因此对于这种情况，我们要对每条隐式后缀树的边维护一棵平衡树表示这条边上作为别的结点 $link$ 的结点有哪些，对于这些边上的结点不维护 $link$ 。

假设当前已经加入了 $S[1, n]$ ，即将加入 $c = S[n + 1]$ 。我们维护指针 i 表示最长的一个后缀 $S[i, n]$ 并且在 S 中出现至少 2 次，记其位置为 $(active, remain)$ 。

加入 c 后，所有叶子对应的串长首先会加 1，这很好记录。

接着看 $(active, remain)$ 这个状态是否存在 $fl(S[i, n], c)$ 这样一条字典树上的边，如果存在，那么就沿着这条边走下去，并令 $remain \leftarrow remain + 1$ ，结束加点过程。

如果不存在，首先要将当前点作为一个新的后缀树上结点，然后给当前结点添加一个转移为 $fl(S[i, n], c)$ 的新儿子，那么令 $i \leftarrow i + 1$ ， $active \leftarrow link_{active}$ ，如果此时的 $active$ 不是隐式后缀树上结点，那么就找到这个点在平衡树上的根结点从而确定其上方第一个隐式后缀树结点，然后确定当前点的位置更新 $active$ 和 $remain$ ，然后如果发现当前沿着 $S[i + 1, n]$ 往下走的边，边长小于等于 $remain$ ，那么令 $remain$ 减去边长，继续往下走，直到定位到串 $S[i + 1, n]$ 对应的点，再接着判断是否存在一条树边可以走到 $S[i + 1, n + 1]$ 。

假设这过程中经过的结点依次是 $v_1 \dots v_k$ ，如果 v_k 不是当前维护的后缀树结点，我们就把这个点加入对应边上的平衡树，但是不维护这个点的 $link$ ，接着令 $link_{v_i} = v_{i+1}$ 。

后缀树结点数是线性的，要对每条边维护平衡树，在边进行分裂时要进行平衡树分裂。如果忽略计算 fl 函数的复杂度，用这种方式构建后缀树的复杂度是 $O(n \log n)$ 的。

3.4 在线维护后缀树

维护一个字符串 S ，支持双端插入，要求在线维护 S 的后缀树结构。

我们结合上述两种算法来实现这一点，但是过程中我们并不维护自动机上的转移边，只维护后缀树结构。

同时维护正串和反串的后缀树，接下来我们只考虑对正串后缀树的修改，反串后缀树的修改对称，并用 LCT 维护两棵后缀树的结构。

在末端添加字符，执行和 Ukkonen 算法进行一样的流程。

在前端添加字符，我们先修改反串后缀树，假设当前已经维护了 $S[2, n]$ 的后缀树，加入 $S[1]$ 。我们通过反串后缀树，得到了最大的一个 i 满足 $S[1, i]$ 在串 S 中至少出现 2 次。那么此时串 $S[1, n]$ 在 $S[2, n]$ 对应的正串后缀树中无法继续往下走的结点一定也是 $S[1, i]$ ，此时可能新增一个结点表示 $S[1, i]$ 然后接一个代表 $S[1, n]$ 的儿子。

我们在反串后缀树对每个点再记录一个 g_u 表示点 u 代表的串在 S 中的任意一次出现位置，即其子树内任意一个作为叶子结点的前缀，那么找到 $S[1, i]$ 除了 i 以外的另一个前缀结点的出现位置 j ，接下来要在正串后缀树上定位后缀 $S[j - i + 1, n]$ ，设 $l = j - i + 1$ 。

考虑此时正串后缀树维护的最长的在 $S[2, n]$ 中出现 2 次的串 $S[p, n]$ ，若 $l < p$ ，则 $S[l, n]$ 在正串后缀树一定作为一个叶子结点存在，那么可以通过维护每个叶子结点对应的后缀来定位。否则 $l \geq p$ ，那么找到 $S[p, n]$ 除了 p 的另一个出现位置 q ，不断将 l 减去 $p - q$ 直到 $l < p$ ，假设这时候的 l 变成 l' ，能够发现此时 $S[l', l' + n - l] \sim S[l, n]$ 。由于此时的 $S[l', n]$ 是一个作为叶子结点的后缀，我们从这个串对应的位置往上跳即可找到串 $S[l, n]$ 对应的结点，从而定位到 $S[1, i]$ 。这个过程可以通过 LCT 实现。

在前端添加字符还可能导致正串后缀树维护的最长的出现至少 2 次的后缀 $S[p, n]$ 的 p 减一，用类似的方式找到 $S[p, n]$ 在反串后缀树上的位置，然后通过判断这个结点是否存在

$fr(S[p, n], S[p - 1])$ 的转移来得到是否需要将 p 变成 $p - 1$ 。此时 $S[p - 1, n] \sim S[1, n - p + 2]$ ，因此从 $S[1, n]$ 对应的结点往上跳即可找到 $S[p - 1, n]$ 定位到这个结点。

如果忽略计算 fl 和 fr 函数的复杂度，这个算法的复杂度是 $O(n \log n)$ 的。

3.5 处理子串问题的方法

我们可以利用后缀树构建当前串 S 的基本子串结构。通过如下算法实现：

- 先建出串 S 的后缀树，对于后缀树上的一个结点，其对应的是其到其父亲这一条边上的所有字符串，我们令这些串为一个等价类，提取出这些串第一次在 S 中出现对应的位置，记为三元组 (k, l, r) ，表示这个等价类包含了串 $S[k, l] \dots S[k, r]$ 。
- 如果发现两个等价类满足出现次数相同，并且形如 (k, l_1, r_1) 和 $(k + 1, l_2, r_2)$ ，并且 $l_1 \leq l_2 \leq r_1 \leq r_2$ ，那么将其合并。
- 最后在二维平面上合并出若干个等价类，每个等价类形如阶梯状。

这样构建出的结构具有原先基本子串结构的大部分性质。

例题 3.1 (太阳神的宴会²)。给定 n 个字符串 $T_1, T_2, \dots, T_n$ ，字符集为 k ，为每个 T_i 选择一个子串 T'_i ，然后将这个子串中的字符作用于任意一个置换，即一个置换 f ，然后令 $T'_i[j] = f(T_i[j])$ ，最后令 $S = T'_1 T'_2 \dots T'_n$ ，问最后能形成多少种本质不同的串 S ，答案对 998244353 取模。

解法 我们先定义 $fl(S[1, i - 1], S[i])$ ，如果 $S[1, i - 1]$ 中不存在 $S[i]$ 则为 0，否则为 $S[i]$ 到上一个等于 $S[i]$ 的位置之间出现了几种不同的字符，同样定义 $fr(S[i + 1, n], S[i]) = fl(S[i + 1, n]^R, s[i])$ 。

接着证明这是充要的，首先若 $S \sim T$ 则一定有 $Fl(S) = Fl(T)$ 。若 $Fl(S) = Fl(T)$ ，能够发现这种方式描述了若干个相等的集合，那么找到 $Fl(S)$ 中为 0 的位置 $v_1 \dots v_k$ ，令 $f(S_{v_i}) = T_{v_i}$ 两个串就变得相等，因此 $S \sim T$ 。

我们不从建后缀自动机的角度入手，而是直接建出后缀树，并构建基本子串结构，然后处理出等价类之间的连边关系。

然后在每个等价类的边界处进行 dp 转移，令二维平面 (i, j) 位置表示串 $T[i, j]$ ，那么后缀添加一个字符相当于从 (x, y) 移到 $(x, y + 1)$ 。

设 $f(u, k)$ 表示当前在等价类边界上的点 u ，已经有 k 个不同的字符出现过，我们从 T_n 转移到 T_1 ，对于串 T_i 按照等价类之间的拓扑序转移，假设当前要从边界点 u_2 对应的 (x, y_2) 转移到边界点 u_1 对应的 (x, y_1) ，中间的每个 (x, y') 可能在末尾添加一个不等于 $fl(T_i[x, y'], T_i[y' + 1])$ 的转移，实际上就是添加一个不等于 $T_i[y' + 1]$ 的字符，从 T_{i+1} 的自动机转移过来，由于添

²来源：集训队互测 2021，<https://uoj.ac/problem/595>

加什么字符并不重要，我们只需要找到其中每个 (x, y') 满足 $fl(T_i[x, y'], T_i[y' + 1]) = 0$ 的位置即可，因为在这种位置出现过的不同字符数会增加，假设出现过的不同字符数为 k ，那么方案数就会乘上 $k!$ 。而找到这种位置只需要记录 $g(i, j, c)$ 表示 $T_i[j, n]$ 中第一个等于 c 的位置在哪即可。

如果中间没有从 T_{i+1} 转移过来，那么就相当于直接从 u_2 转移到 u_1 ，假设满足 $fl(T_i[x, y'], T_i[y' + 1]) = 0$ 的位置个数为 t ，就给 $f(u_1, k)$ 加上 $f(u_2, k + t)$ 即可，而对于边界间的转移方法一样，复杂度 $O(nk)$ 。♣

4 总结

本文主要介绍了一些解决特殊等价关系下字符串问题的方法，介绍了 Hash，KMP 算法，AC 自动机，后缀自动机，后缀树等常见字符串处理工具在特殊等价关系下的做法和应用。希望本文能够带给大家更多关于字符串等价类问题的思考。

5 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练彭思进、杨耀良、张艺缤的指导。

感谢学军中学徐先友等老师的关心与指导。

感谢家人对我的陪伴与支持。

感谢信友队的同学、学长给予我帮助和陪伴。

感谢周康阳同学与我讨论以及给予我启发。

感谢赵海鲲、朱鹏睿、柳易辰等同学为本文验稿并提出建议。

感谢其他给予我帮助的老师与同学。

6 参考文献

- [1] 2022-2023 集训队互测 Round 12《回文匹配》题解，可于 <https://uoj.ac/download.php?type=solution&id=5166> 获取。
- [2] 陈雨昕，《〈太阳神的宴会〉命题报告》，2021 年信息学奥林匹克中国国家队候选队员论文集。
- [3] 代晨昕，《后缀树的构建》，2021 年信息学奥林匹克中国国家队候选队员论文集。
- [4] 许庭强，《一类基础子串数据结构》，2023 年信息学奥林匹克中国国家队候选队员论文集。

浅谈无精度误差的计算几何

南京外国语学校 姚熙

摘要

计算几何相关的题目一般较少在国内的 OI 竞赛中出现,大部分选手也不擅长处理此类问题。这就导致选手遇到此类问题时,往往会使用赛场上临时自创的方法编写代码,从而疏漏一些边界情况或产生较大的精度误差。

本文首先描述了计算几何中的一些经典算法(如极角排序、二维凸包、半平面交以及最小圆覆盖),并对其中一些常见的实现方式指出如何构造出容易疏漏的边界情况以及能够产生较大精度误差的数据,最后对这些算法分别给出实现简单、无情况疏漏且无精度误差的实现方法。

1 前言

计算几何相关问题在国外的 OI 竞赛和 ACM 等比赛中以较高的频率出现,同时此类问题与选手日后可能的发展方向(如图形学、计算机视觉等领域)也有一定的联系,是许多实际问题的基础。尽管如此,计算几何的题目在国内 OI 中相对较少出现,因此,许多选手在遇到此类问题时,缺乏足够的准备,可能会使用临时自创的方法来应对,容易忽视边界情况,导致代码出现边界错误或精度误差。

在实际比赛中,由于题目往往是经过精心设计的,很多情况下边界情况和精度问题是决定算法正确性和效率的关键。尤其是在面对一些经典算法时,例如极角排序、二维凸包、半平面交、最小圆覆盖等,选手在处理时可能会忽略这些细节,从而影响最终的解决效果。本文将通过对这些算法的讨论,揭示常见的边界情况,并提出相应的改进方法,帮助选手在面对此类问题时更加得心应手。这些改进不仅关注正确性,还考虑了实现的简洁性和高效性,力求为读者提供一种更加可靠和高效的解决方案。

2 基本定义

2.1 无精度误差

在计算几何中,“无精度误差”是指在计算过程中,所有数值操作都完全精确,不受浮点运算或数值表示精度限制的影响。这种精确性是计算几何算法的核心基础,因为哪怕是微小的数值误差,都可能导致几何拓扑结构的变化或结果错误。

显然,要实现无精度误差,必须保证问题的输入本身也不存在精度误差。在本文中,我们则默认所有输入数据(如点的坐标和向量的对应分量)均为整数。

与 OI 相关的计算几何算法由于竞赛时间的限制,往往需要兼顾实现的简洁性与运行效率。因此,我们希望提供的算法既能够确保无精度误差,又具有较高的可实现性和执行效率。本文中提出的实现方式均避免手动实现有理数计算,以简化代码复杂度。

2.2 二维向量

定义 2.1 (二维向量). 二维向量是一个具有大小和方向的量,用于描述平面上的点或运动,记为 $v = (x, y)$, 其中 x 和 y 分别表示向量在 x 轴和 y 轴的分量。特别地,零向量记为 $0 = (0, 0)$ 。

二维向量的模长定义为:

$$|v| = \sqrt{x_v^2 + y_v^2}.$$

向量的加减法可以通过对应分量相加减得到, 公式为:

$$u \pm v = (x_u \pm x_v, y_u \pm y_v).$$

二维向量的数乘操作定义为:

$$kv = (kx_v, ky_v).$$

二维向量的内积(点乘)定义为(θ 表示向量 u 到 v 的有向夹角):

$$u \cdot v = |u||v| \cos \theta = x_u x_v + y_u y_v,$$

其结果等价于 u 在 v 上的投影长度与 $|v|$ 的乘积。向量内积具有交换律。

二维向量的外积(叉乘)定义为:

$$u \times v = |u||v| \sin \theta = x_u y_v - y_u x_v,$$

其结果等价于 u 和 v 构成的平行四边形的有向面积。向量外积不具有交换律,但满足 $u \times v = -(v \times u)$ 。

2.3 点和线

在计算几何中，点可以使用向量 v 来描述。直线则有多种表示方法，例如使用斜截式 $y = kx + b$ 或一般式 $Ax + By + C = 0$ 。在计算几何中，也往往会使用 $l = (p, d)$ 来描述，其中 p 为直线上的任意一点， d 为直线的方向。

线段可以使用 $s = (p, d)$ 来描述，其中 p 为线段的一个端点， d 为从 p 到线段另一端点的位移。尽管也可以记录线段的两个端点，但使用端点和位移来描述线段有助于与直线的统一。对于射线而言，则 p 记录射线的端点。

2.3.1 点和直线的关系

若点 u 在直线 (p, d) 上，则有：

$$d \times (u - p) = 0.$$

若点 u 和点 v 在直线 (p, d) 的同侧，则有：

$$(d \times (u - p))(d \times (v - p)) > 0.$$

2.3.2 直线和直线的关系

若 l_1 与 l_2 平行，则有：

$$d_1 \times d_2 = 0.$$

若 l_1 与 l_2 重合，则 l_1 与 l_2 平行且 p_2 在 l_1 上。

2.3.3 点和线段的关系

若点 u 在线段 (p, d) 上，首先 u 必须在直线 (p, d) 上，并且满足以下条件：

$$x_u \in [\min(x_p, x_p + x_d), \max(x_p, x_p + x_d)], \quad y_u \in [\min(y_p, y_p + y_d), \max(y_p, y_p + y_d)].$$

2.3.4 线段和线段的关系

要判断两条线段是否相交，可以求出其所在直线的交点，并根据其的位置关系来判定是否相交；也可以通过快速排斥实验和跨立实验来判断两线段是否相交。

快速排斥实验是指判断两个线段的 Bounding Box（最小边界矩形）是否相交。如果两个线段的 Bounding Box 不相交，则线段必定不相交；如果相交，则可能相交。

跨立实验是另一个必要但不充分的条件。对于线段 s_1 和 s_2 ，跨立实验要求：

- s_1 的两端点在 s_2 所在直线的两侧；

- s_2 的两端点在 s_1 所在直线的两侧。

需要注意的是，两条线段共线但不相交时，跨立实验也可能通过，因此该条件不充分。

线段有交的充要条件是同时满足快速排斥实验和跨立实验。经过这两步验证后，即可确定线段是否有交。

2.3.5 线段求交

对于求线段的交点，首先需要判断两线段是否相交。如果相交，则可以先求出其所在直线的交点。假设两条直线分别为 $l_1 = (p_1, d_1)$ 和 $l_2 = (p_2, d_2)$ ，假设交点的坐标为 $u = p_1 + kd_1$ 。根据交点 u 位于 l_2 上的条件，可以列出方程：

$$(p_1 + kd_1 - p_2) \times d_2 = 0.$$

移项后可以解出 k ：

$$k = \frac{(p_2 - p_1) \times d_2}{d_1 \times d_2}.$$

求出 k 后，即可得到交点的坐标 u 。

对于线段交点的求解，可以先求出对应直线的交点，然后再判断交点是否在线段上。

2.4 多边形

定义 2.2 (多边形). 多边形是由一系列点依次连接形成的封闭图形。这些点称为多边形的顶点，连接顶点的线段称为边。

在计算几何中，往往按照（逆时针）顺序依次记录多边形的顶点 $p_1, \dots, p_n$ 。

多边形的周长可以通过直接计算边长的总和来得到。多边形的面积可以利用向量外积进行计算：

$$S = \frac{1}{2} \left| \sum_{i=1}^n p_i \times p_{i \bmod n+1} \right|.$$

若保证顶点记录顺序为逆时针，则无需外层的绝对值。

定义 2.3 (简单多边形). 简单多边形是指没有自交的多边形，即多边形的边只在端点处相交，且每个顶点恰好连接两条边。

例题 2.1. 给定简单多边形 $P = \{p_1, \dots, p_n\}$ 以及点 u ，判断 u 是否在多边形 P 内部。

容易判断 u 是否在多边形 P 的某条边上。对于 u 不在多边形的边上的情况，可以从 u 向任意方向引出一条射线 l ，通过统计 l 与多边形边界的交点数量来判断 u 是否在多边形内部。如果交点数量为奇数，则 u 在多边形内部。

由于射线的方向可以任取，因此可以选择一条沿着 x 轴正方向的射线 l 以简化交点的计算。对于每条边 $p_i p_{i+1}$ ，我们认为 l 与 $p_i p_{i+1}$ 有交当且仅当下面二者之一成立：

- $y_i < y_u \leq y_{i+1}$ 且 $(u - p_i) \times (p_{i+1} - p_i) < 0$;
- $y_{i+1} < y_u \leq y_i$ 且 $(u - p_i) \times (p_{i+1} - p_i) > 0$ 。

我们实际上将 y 坐标划分为 $(-\infty, y_u)$ 和 $[y_u, +\infty)$ 两个部分，以避免讨论 l 恰好穿过某些端点或与某条边重合的边界情况。

3 极角排序

定义 3.1 (极角排序). 给定二维平面上的 n 个点 $p_1, \dots, p_n$ 。定义点 p 到原点的极角为从 x 轴正方向逆时针旋转到点 p 所经过的角度，值域为 $[0, 2\pi)$ 。极角排序即按照点到原点的极角为第一关键字、点到原点的距离为第二关键字从小到大排序。特别地，原点到原点的极角设为 0。

3.1 算法 1

C++ 标准库中的 “std::atan2” 函数（或 “std::atan2l”）可以直接计算点到原点的极角，因此我们可以直接使用该函数进行排序：

```
bool polar_cmp(const vector2d& u, const vector2d& v) {
    ld au = atan2l(-u.y, -u.x);
    ld av = atan2l(-v.y, -v.x);
    if (abs(au - av) > eps) return au < av;
    return norm2(u) < norm2(v);
}
```

由于 “std::atan2” 返回一个 $[-\pi, \pi]$ 的值，所以将点的坐标按照原点中心对称之后再比较极角大小。

上述写法容易构造出能够产生精度误差的数据，只需要构造两个极角非常相近的点即可。例如 $p_1 = (10^6 - 1, -(10^6 - 1) + 1)$, $p_2 = (10^6, -10^6 + 1)$ 则会在 $\text{eps} \geq 10^{-12}$ 时产生错误的排序结果。

3.2 算法 2

注意到根据向量外积的正负性可以直接判断两个向量的极角关系，我们以此为依据进行排序。

```
bool polar_cmp(const vector2d& u, const vector2d& v) {
    ll d = det(u, v);
```

```
if (d) return d > 0;
return norm2(u) < norm2(v);
}
```

这种方法显然不会产生精度误差，然而直接比较向量外积结果的正负性是有问题的。例如如图 1 中的三个向量 v_1, v_2, v_3 之间满足 $v_1 \times v_2 > 0$, $v_2 \times v_3 > 0$, $v_3 \times v_1 > 0$ ，其比较函数并不满足全序关系，可能在排序时产生 Runtime Error。

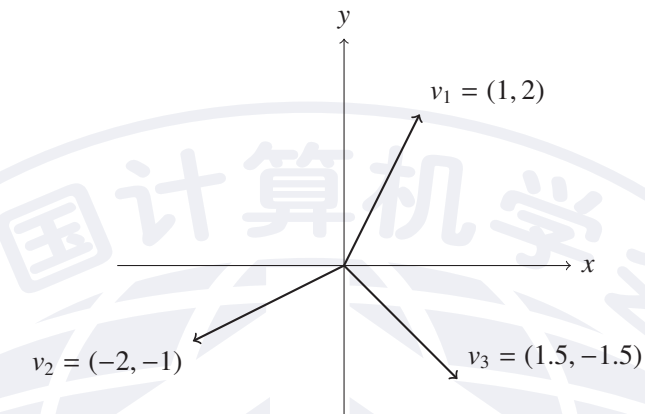


图 1: 直接比较外积正负性不满足全序关系的例子

当然，如果数据保证所有点都在沿原点切开的某个半平面内，这种写法是简洁而正确的（在不要求极角的值域必须是 $[0, 2\pi)$ 的情况下）。

3.3 算法 3

算法 2 是很有前途的，我们可以稍加修改，在使用向量外积的正负性判断之前先对点所在的象限进行判断。象限的判断有一种简单的写法：

```
int quadrant(const vector2d& u) {
    return (u.y < 0) << 1 | ((u.x < 0) ^ (u.y < 0));
}
```

在这种写法下，我们会将坐标轴上的点视为在正方向上的点，这样在向量外积的判断不产生问题的同时，可以将原点排在最前面，并保证极角的值域 $[0, 2\pi)$ 是左闭右开的。比较函数也进行相应的修改：

```
bool polar_cmp(const vector2d& u, const vector2d& v) {
    int qu = quadrant(u);
    int qv = quadrant(v);
```

```
if (qu != qv) return qu < qv;
ll d = det(u, v);
if (d) return d > 0;
return norm2(u) < norm2(v);
}
```

这样便在不产生精度误差的情况下完成了极角排序，并覆盖了所有边界情况。

4 二维凸包

定义 4.1 (二维凸包). 给定二维平面上的 n 个点 $p_1, \dots, p_n$ ，其构成的二维凸包为平面上能包含所有给定点的最小凸多边形。

二维凸包常见的求法有 Andrew [1] 和 Graham [3] 两种算法。由于普及度较高，本文不再描述有精度误差的方法。

4.1 算法 1

将所有点以 x 坐标为第一关键字、 y 坐标为第二关键字升序排序，容易知道排序后第一个点和最后一个点一定在凸包上。

首先从左往右遍历每个点，使用单调栈维护下凸壳。再从右往左遍历每个点，使用单调栈维护上凸壳。将上下两个凸壳合并即可得到最终的凸包。

维护凸壳时，可以使用向量外积的正负性判断是否需要弹出栈顶元素。这种方法在不产生精度误差的情况下，可以得到正确的凸包。

```
for (int i = 1; i <= n; ++i) {
    while (top > 1 && det(stk[top] - stk[top - 1], a[i] - stk[top]) <= 0)
        --top;
    stk[++top] = a[i];
}
for (int i = n - 1, rec = top; i >= 1; --i) {
    while (top > rec && det(stk[top] - stk[top - 1], a[i] - stk[top]) <= 0)
        --top;
    stk[++top] = a[i];
}
```

4.2 算法 2

选取最左下角的点作为凸包的起点，即 x 坐标最小点中 y 坐标最小的点，该点一定在凸包中。将所有点以该点为中心进行极角排序，依次加入每个点，使用单调栈维护凸包。

由于剩下的点都在该点的 x 轴正方向，所以极角排序时无需再判断象限。假设该点已经被移动至 a_1 ：

```
sort(a + 1, a + n + 1, [&](const vector2d& u, const vector2d& v) {
    ll d = det(u - a[1], v - a[1]);
    if (d) return d > 0;
    return norm2(u - a[1]) < norm2(v - a[1]);
});
stk[top = 1] = a[1];
for (int i = 2; i <= n; ++i) {
    while (top > 1 && det(stk[top] - stk[top - 1], a[i] - stk[top]) <= 0)
        --top;
    stk[++top] = a[i];
}
```

4.3 Convex Checker

学会写 Convex Checker 也是重要的，否则可能会造成一些严重的后果。

例题 4.1 (Convex Checker). 给定二维平面上的 n 个点 $p_1, \dots, p_n$ ，判断这些点按顺序依次相连后是否构成一个简单凸多边形。

这也许看起来非常简单，只需要判断所有相邻的三个点是否都构成凸角即可（即便如此还是有写错的可能性）。然而这种方法遇到图 2 的情况就会出现错误：



图 2: 只判断是否为凸角的错误例子

为了避免这种情况，我们还需要检查凸包有没有绕好几圈。一种简单的办法是直接将该点的坐标差分边所对应的向量，然后从极角序的最小值处断环为链，判断是否有序。

最后记得将多边形的点顺序反转再次判断，以防止点按照顺时针顺序给出。

5 半平面交

定义 5.1 (半平面). 给定一条直线 l ，它将二维平面划分为两个半平面，其中一个半平面被称为半平面 l^+ ，另一个半平面被称为半平面 l^- 。直线 l 上的点被称为半平面 l 的边界。

在计算几何中，往往使用 $l = (p, d)$ 来表示一条直线，其中 p 是直线上的某点， d 是直线的方向向量。我们认为其对应的半平面为满足 $d \times (v - p) > 0$ 的点 v 构成的集合，即直线 l 划分出的两个半平面中在 d 的“左侧”的一个。

定义 5.2 (半平面交). 给定 n 条直线 $l_1, \dots, l_n$ 所对应的半平面 $h_1, \dots, h_n$ ，半平面交的可行域为这 n 个半平面的交集。

5.1 算法 1

为了规避一些边界情况，我们加入 4 条直线作为整个半平面交的“外边框”，这样可以减少可行域面积无限时的讨论。

首先将这些直线以 d 的极角为关键字进行排序，若极角相同，则保留离可行域较近（即更靠左）的一个。

按照顺序依次加入每条直线，并维护当前的半平面交。因为半平面交是一个凸多边形，所以需要维护一个凸壳。因为后来加入的只可能会影响最开始加入的或最后加入的边（此时凸壳连通），只需要删除队首和队尾的元素，所以可以使用单调队列进行维护。

在使用单调队列维护凸壳的同时，可以使用另一个单调队列维护凸壳中相邻直线的交点坐标。如图 3 所示，依次加入直线 l_1, l_2, l_3 。对于直线 l_3 ，若 l_1 和 l_2 的交点 p 不在 l_3 左侧，则 l_2 不在可行域内，可以将其从凸壳中删除。

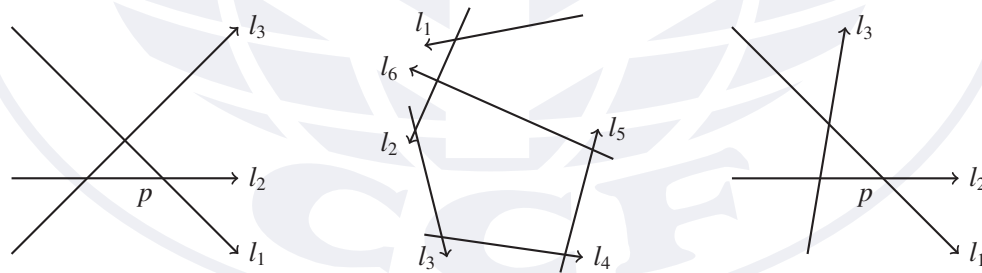


图 3: 队尾需要弹出的情况

另一种可能的情况如图 4，当半平面交即将闭合时，新加入到队尾的直线 l_6 可能会对队首的直线 l_1 造成影响。

因此在加入直线时，需要对当前凸壳队首和队尾的直线都判定是否弹出。上述过程结束之后，还需要用队首的直线排除队尾多余的直线，理由同图 4。

当出现一个可以把队列里的点全部弹出去的直线时，必须先处理队尾，再处理队首 [5]。原因如图 5 所示，当加入 l_3 时， l_1 和 l_2 的交点 p 在 l_3 右侧，所以应当弹出 l_2 和交点 p ，而非 l_1 和交点 p 。实际上，我们进行排除队首操作的原因是当前直线的限制比队首直线更大，其前提条件是队列里有不少于 2 条直线。

5.2 算法 2

我们尝试去除算法 1 中可能产生精度误差的部分。

首先还是加入 4 条直线作为“外边框”，并按照 d 的极角排序。

对于将直线极角排序并去重，可以在常规的极角排序的基础上稍加修改，将第三关键字由到原点的距离改为到可行域的距离：

```
a.push_back({{-inf, -inf}, {1, 0}});
a.push_back({{inf, -inf}, {0, 1}});
a.push_back({{inf, inf}, {-1, 0}});
a.push_back({{-inf, inf}, {0, -1}});
sort(a.begin(), a.end(), [](const line& u, const line& v) {
    int qu = quadrant(u.d);
    int qv = quadrant(v.d);
    if (qu != qv) return qu < qv;
    ll d = det(u.d, v.d);
    if (d) return d > 0;
    return det(v.d, u.p - v.p) > 0;
});
a.erase(unique(a.begin(), a.end(), [](const line& u, const line& v) {
    return quadrant(u.d) == quadrant(v.d) && det(u.d, v.d) == 0;
}), a.end());
```

接下来尝试去除求交点过程中的产生的精度误差。一种思路简单的方法是使用有理数储存交点信息，但实现起来较为复杂，本文并不推荐这种做法。

而我们实际上并不要求出交点的坐标，只需要判断交点是否在直线的左侧。考虑现在有直线 l_1, l_2, l_3 ，尝试判断 l_1 与 l_2 的交点是否在 l_3 的左侧。

我们先只考虑一般的情况，即半平面交实际存在。

首先表示出交点 u 。假设 $u = p_1 + kd_1$ ，由于 u 还在 l_2 上，所以

$$(u - p_2) \times d_2 = (p_1 + kd_1 - p_2) \times d_2 = 0 \iff k = \frac{(p_2 - p_1) \times d_2}{d_1 \times d_2}.$$

我们需要判断 u 是否在 l_3 左侧，即是否有

$$d_3 \times (u - p_3) = d_3 \times (p_1 + kd_1 - p_3) = d_3 \times (p_1 - p_3) + \frac{((p_2 - p_1) \times d_2)(d_3 \times d_1)}{d_1 \times d_2} > 0.$$

由于 l_1, l_2 为凸壳上按顺序相邻的两条直线，所以总有 $d_1 \times d_2 > 0$ 。将上式乘以 $d_1 \times d_2$ 并稍加整理，就能得到

$$(d_3 \times (p_1 - p_3))(d_1 \times d_2) > (d_2 \times (p_2 - p_1))(d_3 \times d_1).$$

上式等价于

$$(d_3 \times (p_1 - p_3))(d_1 \times d_2) > (d_2 \times (p_1 - p_2))(d_1 \times d_3).$$

这个形式非常优美，我们直接将封装成函数：

```
auto above = [](const line& u, const line& v, const line& w) {  
    return det(w.d, u.p - w.p) * det(u.d, v.d) >  
           det(v.d, u.p - v.p) * det(u.d, w.d);  
};
```

并修改原来的算法：

```
int hd = 0, tl = -1;  
vector<line> que(a.size());  
for (auto it : a) {  
    while (hd < tl && !above(que[tl - 1], que[tl], it)) --tl;  
    while (hd < tl && !above(que[hd], que[hd + 1], it)) ++hd;  
    que[++tl] = it;  
}  
while (hd < tl && !above(que[tl - 1], que[tl], que[hd])) --tl;
```

最后我们再对一些边界情况进行判断，例如半平面交不存在。如果最后的凸壳边数小于 3，则半平面交显然不存在。否则我们可以选择两条相邻的变，判断其交点是否在所有半平面内，这里可以复用上文的函数，但要注意判断 $d_1 \times d_2 > 0$ ：

```
if (tl - hd + 1 < 3) return {};  
if (det(que[hd].d, que[hd + 1].d) <= 0) return {};  
for (auto it : a) if (!above(que[hd], que[hd + 1], it)) return {};  
return {que.begin() + hd, que.begin() + tl + 1};
```

至此整个算法流程结束。

5.3 算法 3

算法 2 适用于求解开半平面交的情况，即定义直线上的点不在半平面内。对于闭半平面交，我们可以稍加修改。

考虑闭半平面与开半平面的区别，实际上只是多了可行域面积为 0 的情况需要讨论。这使我们不得不在 l_1 与 l_2 的交点恰好在 l_3 上时保留 l_2 ，从而维护一个不严格的凸壳。而我们在简单分析之后，发现只需要将 above 函数中的 $>$ 改为 $\geq$ 即可。

由此不难看出该方法的简洁性与优美性。

5.4 凸优化

在凸优化的过程中，初学者常用的方法是维护若干直线 $y = kx + b$ 所构成的凸壳，并查询在某个给定 x 处的最值。这种方法确实易于理解，因为求凸壳的过程可以看作是维护若干

函数的最值。但维护凸壳的过程要求出交点，这会导致一定的精度误差，同时 $y = kx + b$ 的形式并不能很好地描述垂直于 x 轴的直线。

对于这种情况，一种方法是将 $y = kx + b$ 转化为 $p = (0, b), d = (1, k)$ 的形式，并使用算法 2 中的 above 函数弹栈进行维护。另一种方法则是改为记录点 $(-k, b)$ ，并查询在斜率为 x 时截距的最值，此时便可以使用常规的二维凸壳进行维护。

由此也不难发现，二维凸包和半平面交在一定程度上是有对偶关系的。

5.5 算法 4

考虑将半平面交问题进行对偶 [2] [6]。对于给定的所有直线，其对应的半平面具有形式 $y < kx + b$ 或 $y > kx + b$ 。对于形如 $y < kx + b$ 的半平面，我们希望找到每个 x 处 $kx + b$ 的最小值；对于形如 $y > kx + b$ 的半平面，我们希望找到每个 x 处 $kx + b$ 的最大值。

类比凸优化的第二种方法，考虑对这两种类型的半平面分别处理。以 $y < kx + b$ 为例，我们将其转化为 $(-k, b)$ 位置的点，并尝试找到在每个斜率 x 处截距的最小值，这样便可以维护点构成的凸壳，即求出所有形如 $(-k, b)$ 的点构成的下凸壳，由对偶关系可知 $y < kx + b$ 在半平面交的边界当且仅当 $(-k, b)$ 在下凸壳上。类似地，对于形如 $y > kx + b$ 的半平面，求出对应的点 $(-k, b)$ 构成的上凸壳。

最后还需要将这两种半平面进行合并，并判断半平面交非空。不难发现交集为空当且仅当某个 x 处 $y < kx + b$ 的最小值小于等于 $y > kx + b$ 的最大值，也即对偶之后求出的上凸壳与下凸壳有交。

考虑处理垂直于 x 轴的直线，不难在求出不包含垂直直线的半平面交后进行特判。

5.6 点线对偶

定义 5.3 (点线对偶). 对于 $x-y$ 平面中的直线 $l: y = ax - b$ ，其到 $a-b$ 平面中的点的变换 $y = ax - b \rightarrow (a, b)$ 记作 l^* 。对于 $x-y$ 平面中的点 $p: (x, y)$ ，其到 $a-b$ 平面中的直线的变换 $(x, y) \rightarrow b = ax - y$ 记作 p^* 。我们将以上变换合称点线对偶。

点线对偶具有以下性质：

定理 5.1. 点 p 在直线 l 上，当且仅当点 l^* 在直线 p^* 上。

证明. 假设 $p: (x, y)$ 以及 $l: y = ax - b$ ，则 $p^*: b = ax - y$ 以及 $l^*: (a, b)$ ，则

$$y = ax - b \iff b = ax - y.$$

□

推论 5.1. 点 p_1, p_2, p_3 三点共线，当且仅当直线 p_1^*, p_2^*, p_3^* 三线共点。

定理 5.2. 点 p 在直线 l 下方，当且仅当点 l^* 在直线 p^* 下方。

证明.

$$y < ax - b \iff b < ax - y.$$

□

定理 5.3. $p^{**} = p$, $l^{**} = l$ 。

证明留作读者练习。

不难发现我们此处定义的点线对偶与此前的对偶关系有所不同。这是因为尽管 $y = ax - b$ 的形式不符合直觉，但其相比此前的对偶形式具有 $p^{**} = p$ 以及 $l^{**} = l$ 的性质。

6 最小圆覆盖

定义 6.1 (最小圆覆盖). 给定二维平面上的 n 个点，求一个半径最小的圆满足所有点都在圆内。

6.1 算法 1

考虑增量构造。假设圆 O_i 是前 i 个点的最小覆盖圆。考虑加入第 i 个点 p_i ，如果 p_i 在圆内则 $O_i = O_{i-1}$ 。否则 O_i 一定经过 p_i 。

以 p_i 为基础，强制圆经过 p_i 并重新依次加入 $p_1, \dots, p_{i-1}$ 。若加入 p_j 时圆 O'_{j-1} 无法覆盖 p_j ，则 O'_j 一定经过 p_i 和 p_j ，此时可以再次递归地重新加入。

由于最多需要 3 个点来确定一个圆，所以最多递归 3 层就会结束。可以证明 [4] 将点的顺序随机打乱后运行此算法的期望时间复杂度为 $O(n)$ ，但复杂度的证明不是本文的重点，因此略过。

6.2 算法 2

要使上述算法无精度误差，我们只需要精确地判断点 u 是否在 p_1, p_2 或 p_1, p_2, p_3 的最小覆盖圆内。

对于 p_1, p_2 的最小覆盖圆，其圆心位于 $\frac{1}{2}(p_1 + p_2)$ ，半径为 $|p_2 - p_1|$ ，我们只需判断

$$\left| u - \frac{1}{2}(p_1 + p_2) \right| \leq |p_2 - p_1| \iff |2u - p_1 - p_2|^2 \leq |2p_2 - 2p_1|^2.$$

对于 p_1, p_2, p_3 的最小覆盖圆，一种方法是求出圆心后判断距离，但是这需要复杂的公式进行大量的计算。我们可以通过计算圆心角来避免这种计算。

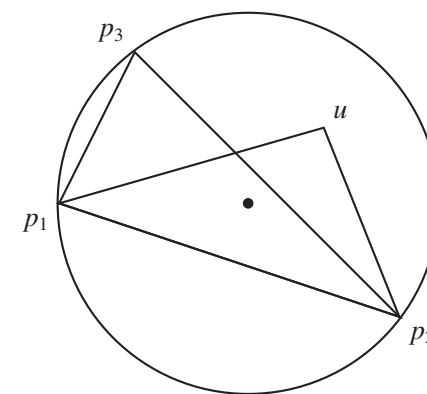


图 6: 使用圆心角判断点是否在圆内

如图 6 所示，假设 p_1, p_2, p_3 按照逆时针顺序排列，且 u 与 p_3 位于直线 p_1, p_2 的同侧。我们可以计算 $\angle p_1 u p_2$ 以及 $\angle p_1 p_3 p_2$ 的大小，如果前者大于（等于）后者，则 u 在圆内（圆上）。

考虑 $\angle p_1 u p_2$ 和 $\angle p_1 p_3 p_2$ 均在值域 $(0, \pi)$ 内，而 $\cos \theta$ 在值域 $(0, \pi)$ 上单调递减，所以我们只需使用向量内积计算角度的 $\cos$ 值并比较大小即可。所以 u 在圆内或圆上当且仅当：

$$\cos \angle p_1 u p_2 = \frac{(p_1 - u) \cdot (p_2 - u)}{|p_1 - u| |p_2 - u|} \leq \frac{(p_1 - p_3) \cdot (p_2 - p_3)}{|p_1 - p_3| |p_2 - p_3|} = \cos \angle p_1 p_3 p_2.$$

显然可以做到无精度误差进行比较。

对于 p_1, p_2, p_3 的排列顺序，可以根据 $\triangle p_1 p_2 p_3$ 的面积正负性进行判断。不难发现总可以通过调整 p_1, p_2, p_3 的顺序以使得上述判断的前提条件成立。

7 总结

本文简单介绍了一些计算几何中常用的算法，并在输入均为整数的特定情况下给出了无精度误差的实现方法。在实际应用中，无精度误差的极角排序、二维凸包、半平面交以及最小圆覆盖都在思维难度、代码复杂度以及实际表现中有一定的优势。

8 致谢

感谢父母和家人一直以来对我的支持和鼓励。

感谢中国计算机学会提供学习和交流的平台。

感谢南京外国语学校张超老师，李曙老师和史钊镭老师对我的指导。

感谢集训队教练彭思进和杨耀良的指导。

感谢赵英智同学为本文审稿。

参考文献

- [1] Alex M Andrew. Another efficient algorithm for convex hulls in two dimensions. Information Processing Letters, 9(5):216–219, 1979.
- [2] Kevin Q Brown. Fast intersection of half spaces. Carnegie-Mellon University. Department of Computer Science, 1978.
- [3] Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. Info. Proc. Lett., 1:132–133, 1972.
- [4] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In New Results and New Trends in Computer Science: Graz, Austria, June 20–21, 1991 Proceedings, pages 359–370. Springer, 2005.
- [5] OI Wiki. 半平面交. <https://oi-wiki.org/geometry/half-plane/>.
- [6] 常瑞年. Oi 中的几何. In IOI 2023 集训队论文集. 中国计算机学会, 2023.

浅谈 OI 中的大容量背包问题

浙江省衢州第二中学 方心童

摘要

本文主要讲述了将 OI 中的大容量背包问题转化成规模较小的问题，并在较小规模的问题上进一步利用分治、二进制拆分等技巧优化的方法，尤其指出了同余在优化背包问题方面的前景。

1 概述

背包问题可以描述为：有 n 种物品，第 i 种物品体积为 w_i ，价值为 v_i ，共有 d_i 个。你需要找到一个选取物品装入背包的方案，使得总体积不超过背包的体积上限 M ，且价值和最大。所有物品体积、价值均非负。

形式化的，你需要找到一组 x ，满足

$$\forall i, x_i \in [0, d_i] \cap \mathbb{N}$$

$$\sum_{i=1}^n w_i x_i \leq M$$

目标是最大化

$$\sum_{i=1}^n v_i x_i$$

对于 $d_i = 1$ 的情形，一般称其为 01 背包问题；对于 $d_i = \infty$ 的情形，称其为完全背包问题；对于 d_i 没有限制的情形，称为多重背包问题。此外，还有 $v_i = w_i$ 的情形，类似于子集和问题。

背包问题可以用一个朴素的 $O(nM)$ 动态规划解决，但是当 M 较大时，这个动态规划算法就力有不逮。我们希望复杂度只与 $\log M$ 有关或和 M 无关。由于 01 背包问题本身限制了 M 不会很大，因此我们主要讨论完全背包问题和多重背包问题。

为了方便叙述，避免一些不必要的讨论，将一些符号与约定列在下方：

- w_i, v_i, d_i, n, M 均与上述意义相同。

- 记 $W = \max_{i=1}^n w_i, V = \max_{i=1}^n v_i$ 。
- 记 $sum(S) = \sum_{i \in S} i$, 其中 S 为可重集合。
- 所有物品均已按照性价比 $p_i = \frac{v_i}{w_i}$ 从大到小排序, 且所有物品的体积价值均为正。
- $\sum_{i=1}^n w_i d_i > M$

2 计算复杂性

为了更好说明背包问题的计算复杂性, 我们引入 $(\min, +)$ 卷积。

定义 1 ($(\min, +)$ 卷积). 有长度为 n 的序列 A 与长度为 m 的序列 B , 定义其 $(\min, +)$ 卷积 $A \times B$ 的结果 C 为

$$C_i = \min_{j+k=i} A_j + B_k$$

其中 $0 \leq i \leq n + m - 2$ 。

不妨假设 $n = m$, 目前学术界不存在 $O(n^{2-\delta})$ 计算 $(\min, +)$ 卷积的算法, 因此在 OI 中, 我们认为 $(\min, +)$ 卷积的复杂度是 $O(n^2)$ 的。同时, 01 背包、完全背包、 $(\min, +)$ 卷积三者可以相互规约¹, 而显然多重背包不弱于 01 背包, 因此我们认为 OI 中的背包问题不存在低于平方的做法。且由于背包问题目前只存在伪多项式做法², 因此平方项中至少一项和值域有关。

3 一般性的分析

如果我们能将 M 缩减至关于 W 的多项式范围, 那么整体复杂度将与 M 无关。

背包问题有一个错误的贪心做法: 对于所有物品按照性价比从大到小排序, 贪心尽量选满。记 t_i 表示在这组贪心解中第 i 个物品选取了多少个。这个方案会存在一个位置 p , 满足 $\forall i < p, t_i = d_i$, 且 $\sum_{i=p+1}^n w_i t_i < W$ 。也即, 在 p 之前的所有物品全部选上, 而物品 p 由于剩余体积不够未能选完, 之后剩余体积就小于 W 。

我们先来考虑一个看似无关的引理:

引理 1. 对于一个大小为 n 的非空可重集合 S , 其存在一个非空可重子集 S' , 满足

$$sum(S') \bmod n = 0$$

¹Isakovsky, <https://www.cnblogs.com/isakovsky/p/17385641.html>, 2023

²Wikipedia, https://wikipedia.org/wiki/knapsack_problem

证明. 设 $S = \{P_1, P_2, \dots, P_n\}$, 计算 $sum_i = (\sum_{j=1}^i P_j) \bmod n$, 特别地, $sum_0 = 0$ 。根据抽屉原理, 一定存在一对 $l < r, l, r \in [0, n]$ 满足 $sum_l = sum_r$, 则区间 $[l + 1, r]$ 内的数即为要求的子集。□

基于这个引理, 有如下定理:

定理 1. 存在一个最优选取方案 x , 满足 $\forall i, x_i \geq t_i - \frac{W^2}{w_i}$ 。

证明. 反证, 考虑最后一个不满足这个条件的位置 q , 显然有 $q \leq p$ 。因为 $< p$ 的所有物品均顶到选取上界, 所以所有 $x_i > t_i$ 的位置均 $\geq p$, 也即均 $> q$, 说明所有相比于贪心方案多选取的物品性价比均不高于 q 。

记所有多选取的物品体积集合为 S 。根据 $|S|$ 分类讨论:

- 若 $|S| < W$, 则新选取的物品体积和 $\sum_{i \in S} i \leq W(W - 1) \leq (t_q - x_q - 1)w_q$, 多选取一个 q 物品会优于当前方案。
- 否则, 根据引理 1, S 存在一个大小不超过 w_q 的非空子集满足总和 sum 为 w_q 的倍数, 则在最终方案中去掉这个子集并多选取 $\frac{sum}{w_q}$ 个物品 q 不会改变总体积, 但是提升了总性价比。

两类情况均与 x 为最优方案矛盾。□

有了定理 1, 我们就可以预先运行一遍贪心算法, 并将每个物品选取的下界在总体积中移除。这样, 剩余的体积不会超过 $W + \sum_{i=1}^n w_i \times \frac{W^2}{w_i} = (nW + 1)W$, 也即 M 被缩减到 $O(nW^2)$ 的范围。然后直接运行朴素的动态规划算法可以得到一个 $O(n^2W^2)$ 的算法, 虽然不尽如人意, 但是起码和 M 无关了。

对于 V 较小的情况, 也可以进行类似的分析, 将每种物品需要决策的物品个数控制在 $O(V^2)$ 范围内。

4 完全背包问题

设一个最优选取方案为 x , 则最优选取物品方案的总体积 $sumW = \sum w_i x_i$ 满足 $sumW > M - W$, 这是因为若 $sumW \leq M - W$, 多选取一个物品更优。因此, 记 f_i 表示选取总体积恰好为 i 的物品的最大价值, 我们只要求出所有满足 $i \in (M - W, M]$ 的 f_i 。

为了快速计算一段区间的 f_i , 有如下引理:

引理 2. 设 S 为某一选取方案中选取物品体积的可重集合, 则存在一种划分 S 为 S_1, S_2 的方案, 满足 $|sum(S_1) - sum(S_2)| \leq W$ 。

证明. 反证, 假设不存在这样的划分方案, 则找到使 $|sum(S_1) - sum(S_2)|$ 最小的划分方案为 S_1, S_2 。不妨假设 $sum(S_1) < sum(S_2)$, 则任取 $x \in S_2$, 分类讨论:

- 若 $sum(S_2) - sum(S_1) \geq 2x$, 则将 x 从 S_2 中加入 S_1 会使两集合绝对值之差减小 $2x$ 。
- 否则, 将 x 从 S_2 中加入 S_1 后形成的新集合 S'_1, S'_2 的绝对值之差为 $sum(S_1) + 2x - sum(S_2) \leq 2x - W \leq W$ 。

两类情况均与 $|sum(S_1) - sum(S_2)|$ 最小矛盾。□

因此, f_i 处的值可以由 f 的 $[\frac{i}{2} - W, \frac{i}{2} + W]$ 区间自 $(\max, +)$ 卷积得到。容易归纳证明, 这样递归计算 k 层之后的范围不会超出 $[\frac{M}{2^k} - 2W, \frac{M}{2^k} + 2W]$, 对于 $\leq O(W)$ 的 i 运行朴素的 $O(W^2)$ DP 预处理 f_i 的值, 就只需要 $O(\log M)$ 次 $(\min, +)$ 卷积即可推出 f_i 在 $[M - W, M]$ 区间内的答案。于是可以得到一个 $O(W^2 \log W)$ 的做法。

这个问题在 OI 中一个可以参考的版本是 2016 USP Try-outs L³。

上述做法仅能求出一段长为 $O(W)$ 区间的 DP 值, 接下来我们将进行精细分析, 使得能在同样的时间复杂度内对于所有 $i \in [1, W^2]$ 求出相应的 DP 值。从前文的分析可以看出, 对于背包问题的优化与同余有比较大的联系。我们先来介绍一个关于线性丢番图方程⁴的定理:

定理 2. 给定 n 个数 $0 < a_1 < \dots < a_n$, 记 $g = \gcd(a_1, \dots, a_n)$, $G(a, n)$ 表示最大的不能表示为 $\sum_{i=1}^n k_i a_i (k_i \in \mathbb{N})$ 的 g 的倍数。则有

$$G(a, n) \leq 2a_{n-1} \left\lfloor \frac{a_n}{gn} \right\rfloor - a_n$$

受篇幅所限, 且证明过程⁵和本文关系不大, 略去。

不失一般性地, 以下分析假设没有两个物品体积相同, 并记 x 为一个选取方案。考虑一个位置 p , 记 S 为所有 $> p$ 的被选取物品体积的可重集合。若存在 $\emptyset \neq S' \subset S$ 使得存在一个长为 p 的自然数序列 k 满足 $\sum_{i=1}^p k_i w_i = sum(S')$, 则可以将 S' 替换成一个个性价比更高的选取方案而总体积不变, 那么 x 就不是最优选取方案。

记 $g = \gcd(w_1, \dots, w_p)$, 显然所有能被这段前缀表示出来的数一定是 g 的倍数。在一个集合中选取一个子集为 g 的倍数使我们想到引理 1, 对其进行扩展可以得到如下定理:

定理 3. 对于一个大小为 m 的可重集合 S 及一常数 $n(m \geq n)$, S 存在一个大小至少为 $m - n + 1$ 的子集 S' 使得

$$sum(S') \bmod n = 0$$

证明. 考虑归纳。对于 S 任一大小为 n 的子集运用引理 1 的方法得到一个 $\bmod n = 0$ 的集合为 S_1 , 剩下的数集合为 S_2 。对于 $|S_2|$ 分类讨论:

³<https://codeforces.com/gym/101064/problem/L>

⁴Wikipedia, https://en.wikipedia.org/wiki/Diophantine_equation

⁵P. Erdős and R. L. Graham, *On a linear diophantine problem of Frobenius*, 1972

- 当 $|S_2| < n$ 时, S_1 已满足条件。
- 当 $|S_2| \geq n$ 时, 记 S_2 的解为 S_3 , 则 $S_1 + S_3$ 即为一个满足条件的解。

□

最后就可以得出对于选取体积的限制:

定理 4. 记 x 为一个最优划分方案, 则 $\forall p \in [1, n]$, 满足

$$\sum_{i=p+1}^n x_i w_i \leq \left\lfloor \frac{3W^2}{p} \right\rfloor$$

证明. 反证, 假设存在一个 p 满足

$$\sum_{i=p+1}^n x_i w_i > \left\lfloor \frac{3W^2}{p} \right\rfloor$$

记 $g = \gcd(w_1, \dots, w_p)$ 。因为 $w_1, \dots, w_p$ 互不相同, 所以 $g \leq \lfloor \frac{W}{p} \rfloor$ 。

现考虑在 $> p$ 选取的物品体积的可重集合 S 中选出一个和最大且 $\bmod g = 0$ 的集合 S' , 这至多需要去除 $g - 1$ 个数, 因此, S' 内数之和 $> \lfloor \frac{2W^2}{p} \rfloor$ 。由定理 2, $w_1, \dots, w_p$ 最大不能表示出来的数 $< \lfloor \frac{2W^2}{pg} \rfloor \leq \lfloor \frac{2W^2}{p} \rfloor$, 因此, S' 内的数之和一定能被表示出来, 即存在一种方案将 S' 所对应的物品替换成前 p 种物品且体积不变, 与 x 为最优方案矛盾。□

因此, 从后往前加入物品, 加入物品 i 时只需要更新前 $\lfloor \frac{3W^2}{i} \rfloor$ 个 DP 状态, 这样的复杂度为 $O(\sum_{i=1}^W \frac{3W^2}{i}) = O(W^2 \log W)$, 最后得到的 DP 数组即为 $i = 1, \dots, W^2$ 的答案。相比于分治做法, 这个做法的代码难度和常数均显著降低。

但是分治也并非一无是处, 类似的分治思想可以解决不那么标准的完全背包问题, 如例题 1。

例题 1. Items⁶

给定 n 种物品, 每种物品有无限个, 第 i 种物品有体积 w_i 。给定 M , 试判断是否存在一种恰选择 n 个物品的方案使得体积和恰好为 M 。 $n \leq 10^5, M \leq n^2, 0 \leq w_i \leq n$ 。

记 $B = \lfloor \frac{M}{n} \rfloor$, 则将所有物品体积减去 B , M 减去 nB 显然不影响答案。此时所有物品的体积在 $[-B, n - B]$ 区间内。

接下来我们证明如下定理:

定理 5. 对于一个选择 n 个物品使得体积之和恰为 M 的方案, 存在一种这 n 个物品的排列方式 P , 使得 $\forall p, \sum_{i=1}^p P_i \in [0, n]$ 。

⁶The 2024 ICPC Asia Kunming Regional Contest Problem I, <https://qoj.ac/contest/1871/problem/9870?v=1>

证明. 考虑增量构造。物品数为 0 时显然成立。现在假设对于 $k \geq 1$, 已经选出了满足条件的 $k-1$ 个物品并排列, 体积总和为 W , 希望加入第 k 个物品。接下来对于 W 和 B 的大小关系分类讨论:

- 当 $W = B$ 时, 任意选取一个物品加入体积仍符合要求。
- 当 $W < B$ 时, 选取一个体积为正的物品加入。若体积为正的物品不存在, 则说明 $M \leq W$, 且剩余物品体积均 ≤ 0 , 体积总和恰好为 $M - W$, 任意选取物品加入即可。
- 当 $W > B$ 时, 选取一个体积为负的物品加入。若体积为负的物品不存在, 则说明 $M \geq W$, 且剩余物品体积均 ≥ 0 , 体积总和恰好为 $M - W$, 任意选取物品加入即可。

□

记 $f^k(i)$ 表示恰好选取了 k 个物品, 体积总和是否可以 i 。根据上述定理, f^k 只保留 $[-n, n]$ 区间内的答案不影响正确性, 则容易用 FFT $O(n \log n)$ 做到 $f^k \rightarrow f^{2k}, f^k \rightarrow f^{k+1}$, 进行 $O(\log n)$ 轮分治即可求出正确答案。

5 多重背包问题

在 W 较小的情形下, 优化多重背包问题常采用的思路有两种: 单调队列与二进制拆分。对于前者, 笔者暂未找到拓展的方法。对于后者, 考虑进行更为完全的拆分, 也即, 对于一个物品 (w, v, d) , 令 $k = \lfloor \log_2(d+1) \rfloor$, 先对于所有 $0 \leq i < k$, 拆出物品 $(2^i w, 2^i v, 1)$, 然后对于所有剩下的物品, 也根据二进制表示拆分成 $(2^i w, 2^i v, 1)$ 的形式。容易证明, 所有可能的选取次数均可以用这 $O(\log d)$ 个物品组合出来。

现在转化为了 01 背包问题。从大到小对于每个 k , 考虑所有形如 $(2^k w, 2^k v)$ 的物品, 维护 DP 数组 f_i 表示剩余体积还有 $i2^k$ 的最大价值。当 $k \rightarrow k-1$ 时, 先修改体积的变化, 然后用所有符合条件的物品更新 DP 状态。

直接这样做的时间复杂度还是 $O(nM)$ 的, 但是可以发现, 对于每个物品, 拆到每个 k 上不超过 2 个。因此对于每个 k , 所有 2 的幂次小于 k 的物品体积之和不超过 $2^k \times 2 \sum_{i=1}^n w_i$ 。

那么 dp 状态只需要保留前 $2 \sum_{i=1}^n w_i$ 个传入 $k-1$, 时间复杂度优化为 $O(n \sum w_i \log W)$ 。

对于 $\sum w_i$ 较大, 但是 $\sum v_i$ 较小的情形, 可以将 DP 状态中的值域与状态互换。具体的, 设 f_i 表示价值为 $i2^k$ 的最小体积, 但是此时如何保留状态并不确定。一个相对简单的办法是二分答案 ans , 这样在 $k \rightarrow k-1$ 时就只需要保留 $[\frac{ans}{2^k} - 2 \sum v_i, \frac{ans}{2^k}]$ 区间内的状态, 但是这样的复杂度是 $O(n \sum v_i \log^2 V)$ 的, 与 $\sum w_i$ 较小的情况并不统一。

记 $V = \max v_i$ 。为了去除二分带来的 $\log V$, 我们考虑利用贪心算法。有如下引理:

引理 3 (贪心算法的 V 最优性). 记贪心算法得到的最大权值和为 P , 实际答案为 Q , 则 $P > Q - V$ 。

证明. 仍然考虑贪心算法的分界点 p 。记 $\leq p$ 的物品选择了的总体积为 W , 则若 $M = W$, 贪心算法求得的就是最优解, 因为每一个体积的性价比均已达到最大。

对于多出来的 $M - W < w_p$ 个体积, 其性价比不可能高于 $\frac{v_p}{w_p}$, 则新增的价值 $\leq \frac{(M-W)v_p}{w_p} < v_p \leq V$, 也即 $P > Q - V$ 。□

则预先运行一遍贪心算法得到答案 P , 在 $k \rightarrow k-1$ 时只需要保留 $[\frac{P}{2^k} - 2 \sum v_i, \frac{P+V}{2^k}]$ 区间内的状态, 复杂度降至 $O(n \sum v_i \log V)$ 。

此外, 还有另一种不基于贪心的做法。在 $k \rightarrow k-1$ 时, 找到最大的 q 使得 $f_q \leq W$, 则所有 $i > q$ 的部分显然均可舍弃。对于 $i < q - 2 \sum v_i$ 的部分, 类似之前的分析也可以得出是不必要的, 因此只需要保留 $[q - 2 \sum v_i, q]$ 区间内的状态, 时间复杂度也为 $O(n \sum v_i \log V)$ 。

值得一提的是, 可以构造出需要保留 $O(\sum v_i)$ 级别状态的数据, 因此这个算法在保留状态的界上是紧的。

6 总结

本文先通过对一般情况的分析将 M 降低到 n 和 W 的多项式级别, 然后对于完全背包, 利用分治和同余分析做到 $O(W^2 \log W)$, 对于多重背包, 利用更为完全的二进制拆分和对于 DP 有效状态分析做到 $O(n \sum w_i \log W)$ 。

值得一提的是, 对于大容量多重背包, 以往的题目如 Sweet Alchemy⁷, Uplifting Excursion⁸ 等均只需要对于 M 进行必要的缩小, 然后仍然使用朴素的暴力 DP 即可通过, 而忽略了在 M 相对较大的部分仍有可以优化之处。笔者命制的 GDCPC2024 Warm Up G 出题⁹ 就需要进行更进一步的优化方能通过。

遗憾的是, 笔者才疏学浅, 对于多重背包没有找到能够将复杂度优化至低于 $O(n \sum w_i)$ 的办法, 希望本文能够激发读者的思考, 期待更好的、在 OI 中实用的做法出现。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢浙江省衢州第二中学叶敏红老师的关心和指导。

感谢家人、朋友对我的支持与鼓励。

感谢葛致远同学与我讨论第三节的证明。

感谢郑钧同学与我讨论, 给予我启发并为本文刊误。

⁷AtCoder Regular Contest 096 Problem F, https://atcoder.jp/contests/arc096/tasks/arc096_d

⁸BalticOI 2022 Day1 T3, <https://www.luogu.com.cn/problem/P8392>

⁹可在 <https://www.luogu.com.cn/problem/U517032> 查看题面及提交

参考文献

- [1] Isakovsky, <https://www.cnblogs.com/isakovsky/p/17385641.html>, 2023
- [2] Wikipedia, https://wikipedia.org/wiki/knapsack_problem
- [3] Yan Soares Couto, <https://linux.ime.usp.br/~yancouto/2016-WIP-Tutorial.pdf>, 2016
- [4] Wikipedia, https://en.wikipedia.org/wiki/Diophantine_equation
- [5] P. Erdős and R. L. Graham, On a linear diophantine problem of Frobenius, 1972

浅谈一类树上换根问题

江苏省常州高级中学 赵海鲲

摘要

树上换根问题是一类非常经典的结构。相较于维护子树内的信息，换根问题由于其子树补部分的结点数、高度都没有保证，所以难以求解。本文阐述了几种处理换根问题的结构与常用方法，非常具有可拓展性，并通过一些例题展现了这些方法在信息学竞赛中的广阔应用。

1 记号与约定

为了方便，本文一般默认研究对象为有根树，且结点数量为 n 。

定义 1.1 (深度). 定义深度表示结点到根结点路径上的结点数量，下文中我们一般记作 dep_x 表示结点 x 的深度。

定义 1.2 (高度). 定义高度表示结点子树内的最大深度，即结点子树内距离该结点最远的点到该结点路径上的结点数量，下文中我们一般记作 height_x 表示结点 x 的高度。

定义 1.3 (子树大小). 定义子树大小表示结点子树内的结点数量，下文中我们一般记作 size_x 表示结点 x 的子树大小。

定义 1.4 (子结点集合). 定义 child_x 表示结点 x 的子结点集合。

2 树链剖分

树链剖分是处理换根问题的利器，原理是通过选择合适的主儿子，将原树分成若干条链，从而能够保证某些性质。

2.1 重链剖分

定义 2.1 (重儿子). 定义重儿子表示一个结点所有子结点中子树大小最大的子结点。如果有多个子树大小最大的子结点，任取其一。如果没有子结点，就无重儿子。

定义 2.2 (轻儿子). 定义轻儿子表示一个结点所有子结点中不是重儿子的结点。

定义 2.3 (重结点). 一个结点是**重结点**当且仅当，这个结点是其父亲结点的重儿子。特别地，我们一般认为根结点不是重结点。

定义 2.4 (轻结点). 定义轻结点为不是重结点的结点。

定义 2.5 (重边). 定义**重边**表示一个结点和它重儿子之间的边。

定义 2.6 (重链). 定义**重链**表示极长的首尾衔接的重边构成的链。

定理 2.1. 我们将重链缩成点，构建一棵新树。对于重构树上高度为 i 的结点，在原树上其子树大小 $f(i)$ 至少为 $2^i - 1$ 。

证明. 考虑归纳证明，首先有 $f(1) = 1$ 。

对于 $i > 1$ ，则其存在一个轻儿子的子树大小至少为 $f(i-1)$ ，而因为它是轻儿子，所以重儿子子树大小也至少为 $f(i-1)$ ，所以 $f(i) \geq 2f(i-1) + 1 = 2 \times (2^{i-1} - 1) + 1 = 2^i - 1$ 。 □

定理 2.2. 对于任意结点，其到根路径上经过轻结点的次数不超过 $O(\log n)$ 次。

证明. 假设经过了 m 次，则有 $f(m) \leq n$ ，即 $2^m - 1 \leq n$ 。

说明 $m \leq \log_2(n+1)$ ，所以经过轻结点的次数不超过 $O(\log n)$ 次。 □

定理 2.3. 所有轻结点的子树大小和不超过 $O(n \log n)$ 。

证明. 每个结点会被计算到根路径上经过轻结点的个数次，所以每个点的贡献次数不超过 $O(\log n)$ ，所以轻儿子的子树大小不超过 $O(n \log n)$ 。 □

定理 2.4. [1] 所有轻结点，它们子树内结点跳到它最多需要经过重链的次数和是 $O(n)$ 的。

或者换一种表述，我们将重链缩成点，构建一棵新树，所有结点的高度和是 $O(n)$ 的。

证明. 由于重构树上高度为 i 的结点，一定互不为祖先后代，所以个数不超过 $\frac{n}{2^{i-1}}$ 。

那么，重构树上所有点的高度和就可以写成：

$$\begin{aligned} & \sum_{i=1}^{\lfloor \log_2(n) \rfloor} i \lfloor \frac{n}{2^{i-1}} \rfloor \\ & \leq \sum_{i=1}^{\lfloor \log_2(n) \rfloor} i \lfloor \frac{n}{2^{i-1}} \rfloor \\ & \leq n \sum_{i=1}^{\lfloor \log_2(n) \rfloor} \frac{1}{2^{i-1}} \\ & \leq n \sum_{i=0}^{\lfloor \log_2(n) \rfloor - 1} \sum_{j=i}^{\lfloor \log_2(n) \rfloor - 1} \frac{1}{2^j} \\ & = n \sum_{i=0}^{\lfloor \log_2(n) \rfloor - 1} \frac{1}{2^{i-1}} - \frac{1}{2^{\lfloor \log_2(n) \rfloor - 1}} \\ & \leq n \sum_{i=0}^{\lfloor \log_2(n) \rfloor - 1} \frac{1}{2^{i-1}} \\ & \leq 4n \end{aligned}$$

此外，事实上重构树上所有点的高度和是可以 $> n$ 的，我们有如下构造。

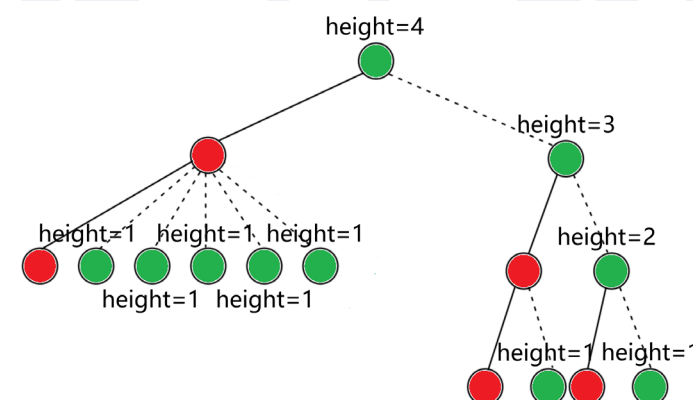


图 1: 构造

其中红色结点是重结点，绿色结点是轻结点。这棵树有 15 个结点，重构树上所有点的高度和为 16。

□

定理 2.5. 所有轻结点的高度和，即子树内最大深度和不超过 $O(n \log \log n)$ 的。

证明. 一个简单的观察是这个界应该在 2.4 和 2.3 之间。

我们先证明上界是 $O(n \log \log n)$ 的。

我们先证明所有二叉树都满足条件。

将重链缩成点，构建一棵新树。定义 $\deg_x$ 表示新树上，结点 x 子结点的数量。

由于是二叉树，所以这条重链的结点数量就是新树上的 $\deg$ 。

我们对新树进行带权长链剖分。具体来讲，我们记 s_x 表示从结点 x 子树内到 x 路径上权值和的最大值，然后我们每次选择 s 最大的子结点作为轻儿子。

我们考虑对一条长链，结点编号分别为 $v_1, v_2, \dots, v_k$ 计算贡献。注意到结点 v_i 的贡献不超过 $\sum_{j=i}^k \deg_{v_j}$ ，所以这条长链的贡献和不超过 $\sum_{i=1}^k \sum_{j=i}^k \deg_{v_j}$ 。

我们设定阈值 B ，进行分析。对于第 i 条长链，假设其 $\deg$ 总和为 m_i 。我们对前 B 个点分析，每个点的贡献不超过 m_i ，由于 $\sum_i m_i = n$ ，所以这部分不超过 nB 。

我们考虑每次向上跳一次重儿子，子树大小至少翻一倍，那么每次向下跳一次轻儿子，子树大小至少减一半。

由于所有长链中 v_1 的子树大小和是 $O(n \log n)$ 的。

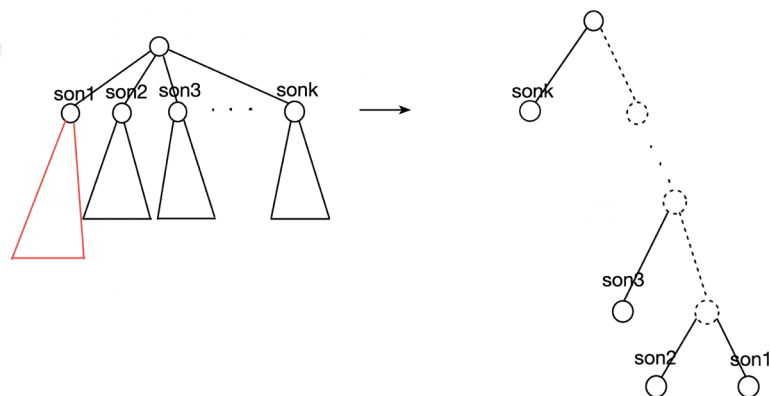
因此，所有长链中 v_j 的子树大小和是 $O(\frac{n \log n}{2^j})$ 的。

所以，对于第 $j > B$ 个点，我们按照子树大小分析，那么它的子树大小不超过 $\frac{n}{2^j}$ 。

所以我们分析得到的轻结点的高度和是不超过 $O(nB + \frac{n \log n}{2^B})$ ，取 $B = \log \log n$ ，就可以得到轻结点的高度和是不超过 $O(n \log \log n)$ 的。

这样我们对于一棵二叉树都证明了轻结点的高度和是不超过 $O(n \log \log n)$ 的，那么对于一般树，我们可以通过加入 $O(n)$ 个点把它变成二叉树，并且满足所有轻结点的高度和不会变小。

具体来讲，我们对于只有一个儿子的点，加一个点作为儿子；对于超过两个儿子的点，我们对重儿子向上建一个虚点，再依次连接其它轻儿子，形如下图这样的过程。

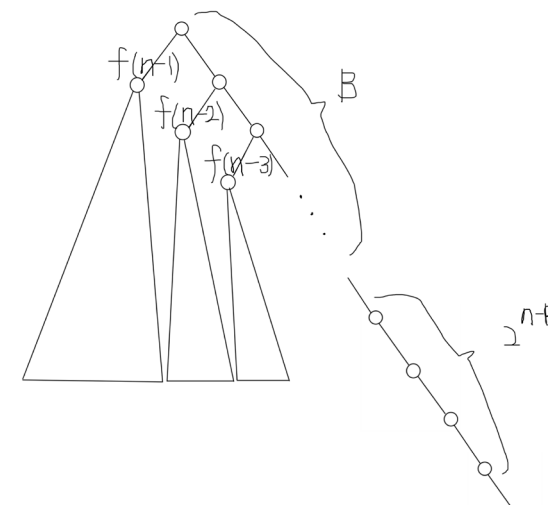


由于我们证明了 n 个点的二叉树内轻结点的高度和是不超过 $O(n \log \log n)$ 的，那么对于普通树，我们可以得到一棵 $O(n)$ 个点的二叉树，并且轻结点的高度和不会变小，所以我们可以说明原树的轻结点的高度和是 $O(n \log \log n)$ 的。

我们再来给出一个 $O(n \log \log n)$ 的构造。

考虑构造一颗满二叉树，令其有 $2^n - 1$ 个点，并设一个阈值 B 。不妨假设满二叉树左边是重的，右边是轻的。然后考虑把右链拿出来，找出来右链的第 B 个元素，将这个子树的所有点改掉，改成一条链，并将这条链旁边的子树，递归处理。

那么，这棵树有 n 层，就会形如下图的结构。



我们令 $f(n)$ 表示 n 层这样结构的树，轻儿子高度和，为了方便分析，我们放缩掉 $\frac{B(B+1)}{2}$ 的贡献。

$$f(n) = \begin{cases} 0 & n < B \\ 2^{n-B} B + \sum_{i=1}^B f(n-i) & n \geq B \end{cases}$$

我们令 $B = \log_2 n$ （注意这里的 B 对于子问题是一样的）， $g(n) = 2^n - \frac{f(n)}{B}$ ，考虑对 $g(n)$ 进行分析。

$$g(n) = \begin{cases} 2^n & n < B \\ \sum_{i=1}^B g(n-i) & n \geq B \end{cases}$$

我们证明， $2^n - g(n)$ 还是一个 $O(2^n)$ 级别的数。

当 $n < 4$ ，显然成立。

取一个 K 和 $x = 2 - \frac{1}{2^B}$ 使 $Kx^n \geq g(n)$ ，

由于

$$\begin{aligned}
 & x^n - x^{n-1} - \dots - x^{n-B} \\
 &= x^{n-B} \left(x^B - \frac{x^B - 1}{x - 1} \right) \\
 &= x^{n-B} \left(x^B - \frac{x^B - 1}{1 - \frac{1}{2^B}} \right) \\
 &= x^{n-B} \left(\frac{1 - \frac{x^B}{2^B}}{1 - \frac{1}{2^B}} \right) > 0
 \end{aligned}$$

这说明当第 $n-B, n-B+1, \dots, n-1$ 项都满足 $Kx^i \geq g(i)$ 时, 第 n 项一定满足 $Kx^n \geq g(n)$ 。

因此只需要满足 $\forall 0 \leq i \leq B, g(i) \leq Kx^i$ 即满足条件, 即 $\forall 0 \leq i \leq B, 2^i \leq Kx^i$ 。

我们分析一下, 由于 $2^B = n$:

$$\begin{aligned}
 & K \left(2 - \frac{1}{2^B} \right)^i - 2^i \\
 &= K \left(2 - \frac{1}{n} \right)^i - 2^i \\
 &= -2^i + K \sum_{j=0}^i \binom{i}{j} 2^{i-j} \left(-\frac{1}{n} \right)^j \\
 &= (K-1)2^i + K \sum_{j=1}^i \binom{i}{j} 2^{i-j} \left(-\frac{1}{n} \right)^j \\
 &\geq (K-1)2^i - K \sum_{j=0}^{\lfloor \frac{i-1}{2} \rfloor} \binom{i}{2j+1} 2^{i-(2j+1)} \left(\frac{1}{n} \right)^{2j+1} \\
 &\geq (K-1)2^i - K \sum_{j=0}^{\lfloor \frac{i-1}{2} \rfloor} i^{2j+1} 2^{i-(2j+1)} \left(\frac{1}{n} \right)^{2j+1} \\
 &= (K-1)2^i - K \sum_{j=0}^{\lfloor \frac{i-1}{2} \rfloor} \frac{2^{i-2j-1} i^{2j+1}}{n^{2j+1}} \\
 &= 2^{i-1} (2K-2-K \sum_{j=0}^{\lfloor \frac{i-1}{2} \rfloor} \frac{2^{-2j} i^{2j+1}}{n^{2j+1}})
 \end{aligned}$$

由于 $n \geq 4$, $\frac{i}{n} \leq \frac{1}{2}$ 。

则 $2K-2-\frac{K}{2}-\frac{K}{16} \geq 0$ 。

可以得到 $K \geq \frac{32}{23}$ 。

那么,

$$\begin{aligned}
 & 2^n - g(n) \geq 2^n - Kx^n \\
 &= 2^n - \sum_{i=0}^n \binom{n}{i} K \frac{(-1)^i 2^{n-i}}{n^i} \\
 &\geq -(K-1)2^n + K2^{n-1} - K \left(\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} 2^{n-2i} \frac{\binom{n}{2i}}{n^{2i}} \right) \\
 &\geq -(K-1)2^n + K2^{n-1} - K(2^{n-3} + \sum_{i=2}^{\lfloor \frac{n}{2} \rfloor} 2^{n-2i} \frac{n^{2i}}{n^{2i} 4!}) \\
 &\geq (2-K)2^{n-1} - K(2^{n-3} + 2^{n-7}) \\
 &= 2^n - K(2^{n-1} + 2^{n-3} + 2^{n-7})
 \end{aligned}$$

当 $K < \frac{128}{81}$ 即可说明满足条件, 取 $K = \frac{32}{23}$ 即可。

□

2.2 长链剖分

定义 2.7 (长儿子). 定义长儿子表示一个结点所有子结点中子树高度最大的子结点。如果有多个子树高度最大的子结点, 任取其一。如果没有子结点, 就无长儿子。

定义 2.8 (短儿子). 定义短儿子表示一个结点所有子结点中不是长儿子的结点。

定义 2.9 (长结点). 一个结点是长结点当且仅当, 这个结点是其父亲结点的长儿子。特别地, 我们一般认为根结点不是长结点。

定义 2.10 (短结点). 定义短结点为不是长结点的结点。

定义 2.11 (长边). 定义长边表示一个结点和它长儿子之间的边。

定义 2.12 (长链). 定义长链表示极长的首尾衔接的长边构成的链。

定理 2.6. 我们将长链缩成点, 构建一棵新树。对于重构树上高度为 i 的结点, 在原树上其子树大小 $f(i)$ 至少为 $\frac{i(i+1)}{2}$ 。

证明. 考虑归纳证明, 首先有 $f(1) = 1$ 。

对于 $i > 1$, 则其存在一个短儿子的子树大小至少为 $f(i-1)$, 而长儿子子树高度至少为 $i-1$, 所以长儿子子树大小至少为 $i-1$, 所以 $f(i) \geq f(i-1) + i-1 + 1 = \frac{i(i+1)}{2}$ 。 □

定理 2.7. 对于任意结点, 其到根路径上经过短结点的次数不超过 $O(\sqrt{n})$ 次。

证明. 假设经过了 m 次, 则有 $f(m) \leq n$, 即 $\frac{m(m+1)}{2} \leq n$ 。

说明 $m \leq \sqrt{2n}$, 所以经过轻结点的次数不超过 $O(\sqrt{n})$ 次。 □

定理 2.8. 所有短结点的子树大小和不超过 $O(n\sqrt{n})$ 。

证明. 每个结点会被计算到根路径上经过短结点的个数次, 所以每个结点的贡献不超过 $O(\sqrt{n})$, 所以短结点的子树大小和不超过 $O(n\sqrt{n})$. $\square$

定理 2.9. 所有短结点的高度和, 即子树内最大深度和是恰好为 n 的。

证明. 每个短结点的高度, 即为长链的结点数量。不难发现, 由于每个结点都恰好属于一条长链, 所以短结点高度和恰好为 n . $\square$

2.3 小结

根据上文整理, 可以得到下表。

表 1: 树链剖分总结 (注: 下面描述的都是其上界)

树链剖分	轻结点子树大小和	轻结点子树高度和	轻结点子树内最大跳链次数和
重链剖分	$O(n \log n)$	$O(n \log \log n)$	$O(n)$
长链剖分	$O(n\sqrt{n})$	n	n

在信息学竞赛中, 一般树上问题中, n 的规模不超过 10^6 。在这种情况下, $O(\log \log n)$ 几乎可以看做常数。

由此可见, 重链剖分性质极强, 可以在大多数时候取代长链剖分。而长链剖分则在处理子树高度相关问题上, 由于能保证长儿子高度最大, 所以代码上会更加容易实现。

3 多树二分

在换根问题中, 我们一般的做法是在换根的过程中使用重链剖分, 并对全局维护一个数据结构。

到一个结点 x 时, 将结点 x 的轻儿子的信息加入, 对于结点 x 的重儿子加入其数据结构编号, 得到一个大小不超过 $O(\log n)$ 的集合。递归进子树 y 时, 先将子树 y 的信息删除, 处理完子树 y 之后再子树 y 的信息加入回来。离开子树 x 时, 再将结点 x 的轻儿子子树信息删除。

所以如何处理多树二分就显得非常重要, 直接关系到最终的时间复杂度。

此时, 如果是结构相同的权值线段树, 我们可以同时在这些线段树上二分。但对于结构不同的线段树或者是平衡树, 我们还需要更加优秀的做法。

3.1 引入

例题 3.1 (元旦激光炮¹)。给定三个单调不降的数组 a, b, c , 长度分别为 n_a, n_b, n_c , 每次查询可以询问一个位置的值, 求三个数组归并后的第 k 小值。

不妨将问题泛化, 变成 n 个数组, 第 i 个数组 a_i 的长度为 l_i , 并令单个 l_i 的级别为 $O(m)$ 。这里我们默认数组下标从 1 开始。

3.2 数组上考虑

3.2.1 算法 1

解法. 朴素的做法是, 我们先在外层二分, 再在 n 个序列上分别二分。

时间复杂度 $O(n(\log n + \log m) \log m)$ 。

3.2.2 算法 2

解法. 我们令 $l = \lfloor \frac{k}{n} \rfloor$, 并求出第 i 个序列的第 l 个位置 p_i 。

假设其中的最小值为 p_i ($1 \leq i \leq n$), 那么由于比 p_i 小的数不超过 $n(l-1)$ 个, 所以 p_i 的排名最大为 $nl - n + 1$ 。

- 对于 $n = 1$, 我们可以特判;
- 否则就有 $nl - n + 1 < k$, 那么说明第 i 个数组前 l 个元素都不可能成为第 k 小值, 所以我们可以将 $k \leftarrow k - l$ 。

时间复杂度 $O(n(\log n + \log m))$ 。

3.2.3 算法 3

解法. 我们对于每个考虑取出所有序列的中点, 令第 i 个序列的中点为 mid_i 。

令 $s = \sum_{i=1}^n \text{mid}_i$ 。

- 假设 $s > k$, 那么说明 a_{i, mid_i} 最大的位置及其后缀不可能是第 k 大;
- 否则对于 a_{i, mid_i} 最小的位置比它小的元素数量不超过 $s - n$ 个, 所以它的排名最大为 $s - n + 1$ 。

¹来源: UOJ Round 4, 链接: <https://uoj.ac/problem/52>

- 对于 $n = 1$ ，我们可以特判；
- 否则就有 $s - n + 1 < k$ ，那么说明这个位置及其前缀都不可能成为第 k 小值。

所以我们每次可以将一个数组的长度缩小一半。

注意到由于每次需要查找 a_{i,mid_i} 的最大值、最小值，并弹出的操作，所以我们可以需要使用可删堆维护。

时间复杂度 $O(n \log n \log m)$ 。

♠

3.3 数据结构上考虑

我们这里，以平衡树为例，进行分析。

3.3.1 算法 1

解法. 查询一个数的排名时，数组和平衡树的时间复杂度都是 $O(\log m)$ 的。

时间复杂度 $O(n(\log n + \log m) \log m)$ 。

♠

3.3.2 算法 2

解法. 由于平衡树不能支持 $O(1)$ 查询第 k 大，所以还需要平衡树二分，单次时间复杂度 $O(\log m)$ 。

时间复杂度 $O(n(\log n + \log m) \log m)$ 。

♠

3.3.3 算法 3

解法. 注意到我们上面的做法中，并不需要保证 mid_i 恰好是中点。我们取中点的目的，仅仅是为了保证复杂度。

所以，我们可以依照平衡树的结构，将根结点设为 mid_i 。而将以 mid_i 开头的前缀删除和以 mid_i 开头的后缀删除，恰好分别对应平衡树上递归到右子树和左子树的操作。

所以，该算法的时间复杂度不变。

时间复杂度 $O(n \log n \log m)$ 。

♠

3.3.4 算法 4

解法. ² 但是实际上有更好的做法，这里我们使用 WBLT 平衡树。

²做法来自李欣隆 Ynoi 的技巧

我们先假设 $k \leq$ 总平衡树大小和的 50%。那么就可以对于第 i 棵平衡树，找到一个排名在 $[71\%, 80\%]$ 中间的元素 val_i ，并将第 i 棵平衡树按照 val_i ，分裂成两棵平衡树。WBLT 可以在 $O(1)$ 的时间复杂度下实现这个过程，但是诸如 FHQ Treap 就无法做到 $O(1)$ ，因为 FHQ Treap 的合并操作需要合并到底，只能做到 $O(\log m)$ 的时间复杂度。

接下来，我们 $O(n)$ 求出加权 nth_element 。具体来讲，我们将所有平衡树分成三组。第二组只有一棵平衡树，假设其编号为 pos ，不妨令 $\text{key} = \text{val}_{\text{pos}}$ 。我们要求：

- 第一组中平衡树所有的 $\text{val}_i \leq \text{key}$ ，第三组中平衡树所有的 $\text{val}_i \geq \text{key}$ 。
- 第一组平衡树和第二组平衡树的结点个数总和在总平衡树中的占比 $\geq 71\%$ ，第二组平衡树和第三组平衡树的大小总和在总平衡树中的占比 $\geq 29\%$ 。

注意到这时我们能确定 $\leq \text{key}$ 的元素个数在总平衡树中的占比 $\geq 71\% \times 71\% = 50.51\%$ ，又由于我们假设了 $k \leq$ 总平衡树大小和的 50%，所以我们可以说明 $> \text{key}$ 的元素都不可能成为答案。

注意到可以确定 $> \text{key}$ 的元素，至少在总平衡树中的占比 $\geq 29\% \times 20\% = 5.8\%$ 。

对于 $k >$ 总平衡树大小和的 50%，我们可以按照排名在 $[20\%, 29\%]$ 中间的元素 val_i 做类似的过程。

所以，总而言之，我们每次可以在 $O(n)$ 的时间复杂度下，将至少占平衡树大小 5.8% 的元素删除。

时间复杂度 $O(n(\log n + \log m))$ 。

♠

3.4 小结

在信息学竞赛中，算法 4 过于复杂，难以实现，并且对于平衡树的结构有比较严格的要求。

而实际问题中，往往数据结构数量的 n 是远远小于 m 的。

所以相比于朴素的算法 1，算法 3 的时间复杂度是有优势的。

并且，不难发现算法 3 可以轻松拓展到权值线段树上，适用性很广。具体来讲，如果左儿子是空的，可以直接进入右儿子；如果右儿子是空的，可以直接进入左儿子。然后将右儿子最左侧元素设为树根，并且将递归进右子树改成递归进右儿子相当于保留树根，在叶子结点的时候特判即可。

4 换根维护深度相关信息

例题 4.1 (希望³)。有一棵树，求选出 k 个连通块，满足这些连通块交集中存在一点让这些连通块中任意一点到这个点的距离不超过 L 的方案数，对 998244353 取模。

³来源：十二省联考 2019，链接：<https://www.luogu.com.cn/problem/P5291>

数据范围： $1 \leq n \leq 10^6$, $1 \leq k \leq 10$ 。

解法. 不难发现对于一个选连通块的方案，满足条件的点必然构成树上的一个连通块。于是我们可以进行点减边容斥，即我们对每个满足要求的点计算 1 的贡献，对每条边要求其两端的点都满足要求计算 -1 的贡献。这样就可以保证对于一个满足条件的树上连通块，整体计算 1 的贡献。

我们记录 $f_{x,i}$ 表示结点 x 子树内到结点 x 的距离 $\leq i$ 的非空连通块数量。转移：

$$f_{x,i} = \prod_{y \in \text{child}_x} (f_{y,i-1} + 1)$$

我们记录 $F_{x,i}$ 表示结点 x 子树外（不包含结点 x ）到结点 x 的距离 $\leq i$ 的非空连通块数量。转移：

$$F_{x,i} = (F_{\text{fa}_x, i-1} + 1) \prod_{y \in \text{child}_{\text{fa}_x} \setminus \{x\}} (f_{y,i-2} + 1)$$

- 点的贡献是 $(f_{x,L}(F_{x,L} + 1))^k$;
- 边的贡献是 $(f_{x,L-1}F_{x,L})^k$ 。

我们考虑**长链剖分**来维护，记结点 x 的长儿子为 hson_x 。

我们先考虑计算 f 。注意到对于 $i \geq \text{height}_x$ ，答案都等于 $f_{x, \text{height}_x - 1}$ ，所以我们只需要维护 $\min\{L + 1, \text{height}_x\}$ 个值。我们可以对每条长链维护 f 数组。具体地，我们对结点 x ，

1. 继承 f_{hson_x} 的状态；
2. 加入轻儿子 y 贡献。对于 $i \in [1, \min\{L + 1, \text{height}_y\} + 1]$ 的前缀，我们可以暴力单点修改修改 $f_{x,i}$ ，对于后缀，我们可以打后缀乘法的标记维护。

我们再考虑计算 F 。注意到由于我们只需要最后求出每个点 $F_{x,L}$ 的值，所以对于 $F_{x,i}$ ，若 $i + \text{height}_x - 1 < L$ ，我们并不需要维护该值。于是，我们对结点 x 只需要维护一个长度为 $\min\{\text{height}_x, L + 1\}$ 的数组。我们可以对每条长链维护 F 数组。具体地，我们对结点 x ，

1. 根据 f_x 回退出 f_{hson_x} ；
2. 先加入所有轻儿子 y 的贡献。依旧是形如前缀暴力单点修改，后缀打乘法标记来维护；
3. 对于长儿子，我们继承当前 F_x 的状态；
4. 对于轻儿子 y ，我们先删除其贡献，接着我们根据 F_x 和 f_{hson_x} 求出 F_y ，递归进 y 子树求解，回溯时再将轻儿子 y 的信息加入即可；

5. 将 f_{hson_x} 还原成 f_x 。

于是，我们整个算法的时间复杂度瓶颈在于求 k 次幂。

时间复杂度 $O(n \log k)$ ，空间复杂度 $O(n)$ 。▲

例题 4.2 (网络⁴)。给一棵 n 个结点的树和一个参数 d ，定义两个结点 x, y 之间的距离为 x 到 y 的简单路径上的边数。

定义一个树上连通块的权值为连通块中任意两点的距离之和。定义一个树上连通块的直径为连通块中任意两点距离的最大值。

给出 q 次询问，第 i 次询问给出一个点 u_i ，请求出所有包含 u_i 的直径不超过 d 的树上连通块中权值的最大值。

数据范围： $n \leq 5 \times 10^5$, $q \leq 10$ 。

解法. 一道更加复杂的题目。

我们考虑选出连通块的直径中点。

- 如果 d 是奇数，那么直径中心是一条边。所以选出的连通块形如选择一条边距离其中一个端点不超过 $\frac{d-1}{2}$ 的所有结点；
- 如果 d 是偶数，那么直径中心是一个点。所以选出的连通块形如选择一个点距离不超过 $\frac{d}{2}$ 的所有结点。

下文中，我们令 $L = \lfloor \frac{d}{2} \rfloor$ 。

我们记如下状态：

记 $f_{x,i}$ 表示 x 子树内到 x 的距离 $\leq i$ 的点的个数。

记 $g_{x,i}$ 表示 x 子树内到 x 的距离 $\leq i$ 的点到 x 的距离和。

记 $h_{x,i}$ 表示 x 子树内到 x 的距离 $\leq i$ 的所有点中任意两点距离和。

考虑加入一棵子树 y 的转移，记 f', g', h' 为加入子树 y 之前的 DP 值，转移：

$$f_{x,i} = f'_{x,i} + f_{y,i-1}$$

$$g_{x,i} = g'_{x,i} + g_{y,i-1} + f_{y,i-1}$$

$$h_{x,i} = h'_{x,i} + h_{y,i-1} + g'_{x,i}f_{y,i-1} + f'_{x,i}g_{y,i-1} + f'_{x,i}f_{y,i-1}$$

记 $F_{x,i}$ 表示 x 子树外到 fa_x 的距离 $\leq i$ 的点的个数。

记 $G_{x,i}$ 表示 x 子树外到 fa_x 的距离 $\leq i$ 的点到 fa_x 的距离和。

记 $H_{x,i}$ 表示 x 子树外到 fa_x 的距离 $\leq i$ 的所有点中任意两点距离和。

转移是，我们先继承 F, G, H_{fa_x} 的信息，再加入结点 fa_x 的贡献，转移：

⁴来源：SNOI2019，链接：<https://www.luogu.com.cn/problem/P5411>[3]

$$F_{x,i} = F_{fa_x,i-1}$$

$$G_{x,i} = G_{fa_x,i-1} + F_{fa_x,i-1}$$

$$H_{x,i} = H_{fa_x,i-1} + G_{fa_x,i-1} + F_{fa_x,i-1}$$

考虑加入一棵子树 y 的转移，记 F', G', H' 为加入子树 y 之前的 DP 值。

$$F_{x,i} = F'_{x,i} + F_{y,i-2}$$

$$G_{x,i} = G'_{x,i} + G_{y,i-2} + F_{y,i-2}$$

$$H_{x,i} = H'_{x,i} + H_{y,i-2} + G'_{x,i}F_{y,i-2} + F'_{x,i}G_{y,i-2} + F'_{x,i}F_{y,i-2}$$

考虑计算答案。

- 若 d 为偶数，则以点 x 为中心的极大合法连通块的权值为：

$$h_{x,L} + H_{x,L-1} + g_{x,L}F_{x,L-1} + f_{x,L}G_{x,L-1} + f_{x,L}F_{x,L-1}$$

- 若 d 为奇数，则以边 (x, fa_x) 为中心的极大合法连通块的权值为：

$$h_{x,L} + H_{x,L} + g_{x,L}F_{x,L} + f_{x,L}G_{x,L} + f_{x,L}F_{x,L}$$

我们考虑**长链剖分**来维护，记结点 x 的长儿子为 $hson_x$ 。

我们先考虑计算 f, g, h 。注意到对于 $i \geq \text{height}_x$ ， $f_{x,i}, g_{x,i}, h_{x,i}$ 的值不变，所以我们只需要维护 $\min\{L+1, \text{height}_x\}$ 个值。我们可以对每条长链维护 f, g, h 数组。具体地，我们对结点 x ，

- 继承 $f_{hson_x}, g_{hson_x}, h_{hson_x}$ 的状态；
- 加入轻儿子 y 贡献。对于 $i \in [1, \min\{L+1, \text{height}_y\} + 1]$ 的前缀，我们可以暴力单点修改修改 $f_{x,i}$ ，对于后缀，注意到转移的形式可以用矩阵乘法来表示，所以我们可以打后缀矩阵乘法的标记维护。

和 4.1 类似，我们对结点 x 只需要维护一个长度为 $\min\{\text{height}_x, L+1\}$ 的数组。我们可以对每条长链维护 F, G, H 数组。具体地，我们对结点 x ，

- 根据 f_x, g_x, h_x 回退出 $f_{hson_x}, g_{hson_x}, h_{hson_x}$ ；
- 先加入所有轻儿子 y 的贡献。依旧是形如前缀暴力单点修改，后缀打**矩阵乘法标记**来维护；
- 对于长儿子，我们继承当前 F_x 的状态；

- 对于轻儿子 y ，我们先删除其贡献，接着我们根据 F_x, G_x, H_x 和 $f_{hson_x}, g_{hson_x}, h_{hson_x}$ 求出 F_y, G_y, H_y ，递归进 y 子树求解，回溯时再将轻儿子 y 的信息加入即可；

- 将 $f_{hson_x}, g_{hson_x}, h_{hson_x}$ 还原成 f_x, g_x, h_x 。

接着，我们再考虑对于每个结点，求出要求包含它的答案。

我们同样可以用上文类似的方法，对于子树内设状态：

记 $t_{x,i}$ 表示 x 子树内到 x 的距离 $\leq i$ 的直径中心的答案最大值。

记 $T_{x,i}$ 表示 x 子树外到 fa_x 的距离 $\leq i$ 的直径中心的答案最大值。

转移的方法和上文大致相同，不过这里由于是维护最大值信息，所以我们需要对 $T_{x,i}$ ，维护来自不同子树的最大值、次大值，这样就能处理删除一棵轻子树信息了。

于是，我们就可以对所有结点都求出答案。

时间复杂度 $O(n+q)$ 。

例题 4.3 (运筹帷幄⁵)。给定一棵 n 个结点的树，第 i 个结点有 b_i 个棋子，且最多能放 a_i 个棋子。现在有一个结点 k 是根。每次操作你可以选择一个结点，将它的一个棋子，移到它的父亲上，需要满足它父亲的棋子数没有超过限制，然后需要最小化所有棋子到 k 的距离和。

对 $k = 1, 2, \dots, n$ 都求出答案。

数据范围： $1 \leq n \leq 5 \times 10^5$ 。

解法。 本题是笔者今年的集训队互测试题。

本题还有很多其它有意思的做法，可以详见参考文献 [4]。

由于 $a_i \geq 1$ ，所以我们可以把题目转化成每次选择一个棋子，移到其祖先。

首先，我们有一个暴力的贪心做法。我们 DFS 整棵树，假设当前在结点 x ，先对于结点 x 的子树进行贪心的过程，然后再将子树内最深的棋子移到结点 x 。

这个贪心做法的证明跟本文无关，可以在参考文献 [4] 中找到。

我们考虑维护这个过程。我们记 $f_{x,i}$ 表示 x 子树内操作完后到 x 的距离 $\geq i$ 的棋子数量。

记 $g_{x,i}$ 表示 x 子树内操作完后到结点 x 的距离 $\geq i$ 的棋子到 x 的距离和。

我们考虑**长链剖分**来维护，记结点 x 的长儿子为 $hson_x$ 。

我们先考虑计算 f, g 。注意到对于 $i \geq \text{height}_x$ ， $f_{x,i}, g_{x,i} = 0$ ，所以我们只需要维护 height_x 个值。我们可以对每条长链维护 f, g 数组，并且维护当前删除了前 del_x 大的结点，并维护这些结点到 x 的距离和 sum_x 。具体地，我们对结点 x ，

- 继承 f_{hson_x} 的状态；

⁵来源：2024-2025 集训队互测 Round 5，链接：<https://qoj.ac/contest/1822/problem/9534>

- 加入轻儿子 y 贡献。由于我们维护的是后缀和，而每次修改是一段前缀，因此我们可以维护；
- 将前 $a_x - b_x$ 大删除，我们在 f 数组上二分，即可更新 sum_x 。

我们考虑换根计算子树补的信息，这个时候跟先前两道例题的情况大不相同，我们需要新的处理方法。我们考虑把信息拆成轻重子树来处理。

我们考虑在换根的过程中**重链剖分**维护，记结点 x 的重儿子为 wson_x 。

我们在过程中维护：

- 到根路径经过的轻结点，它们的父亲的重儿子结点编号集合 S_x ；
- 用**数组**来维护 F_i 表示到 x 距离 $\geq i$ 的棋子数量。
 G_i 表示到 x 的距离 $\geq i$ 的棋子到 x 的距离和。

假设当前在结点 x ，

- 根据 f_x, g_x 回退出 $f_{\text{hson}_x}, g_{\text{hson}_x}$ ；
- 我们先将所有轻儿子都加入 F 里面；
- 对于重儿子：不需要做额外处理；
- 对于轻儿子：我们先将结点 y 在 f 内的贡献删除，然后再截取重儿子距离 $\leq \text{size}_y$ 的部分，对于重儿子 f_{wson_x} 中 $> \text{size}_y$ 的部分，我们将重儿子的结点编号 wson_x 加入 S_y 内，并记录 size_y ；
- 将 $f_{\text{hson}_x}, g_{\text{hson}_x}$ 还原成 f_x, g_x 。

此外，对于删除距离前 $a_x - b_x$ 大的，注意到我们肯定是依次删 S_x 最早加入的重儿子，我先判断这个数组能否删空，如果能删空，我们就直接在 S 中删除，否则再二分。这样，我们就只需要进行至多一次二分。

求答案时，我们如果删完 S 后，结点 x 仍有空余位置，我们只需要对数组 f_x 和 F_x 进行二分即可。

如果加入轻儿子时，在已经被删除的地方增加了棋子，则我们可以直接重构 f, g 。

时间复杂度 $O(n \log n)$ ，空间复杂度 $O(n)$ 。

♠

5 换根维护凸壳

5.1 引入

例题 5.1 (长野原龙势流星群 改⁶)。有一棵 n 个点的树，第 i 个点的点权为 a_i 。

现在你需要对每个点 i 找到一个**包含 i** 的连通块，并最大化这个连通块内所有点的点权的平均值。

对每个点 i 求出这个最大的平均值。

数据范围： $1 \leq n \leq 2 \times 10^5$ ， $1 \leq a_i \leq 10^9$ 。

5.2 基本思路

我们考虑树形 DP， $\text{dp}_{x,i}$ 表示在 x 子树中，大小为 i 的包含 x 的连通块中，和的最大值。
 $\text{dp}_{x,i}$ 本身并不是凸的，但我们有以下结论。

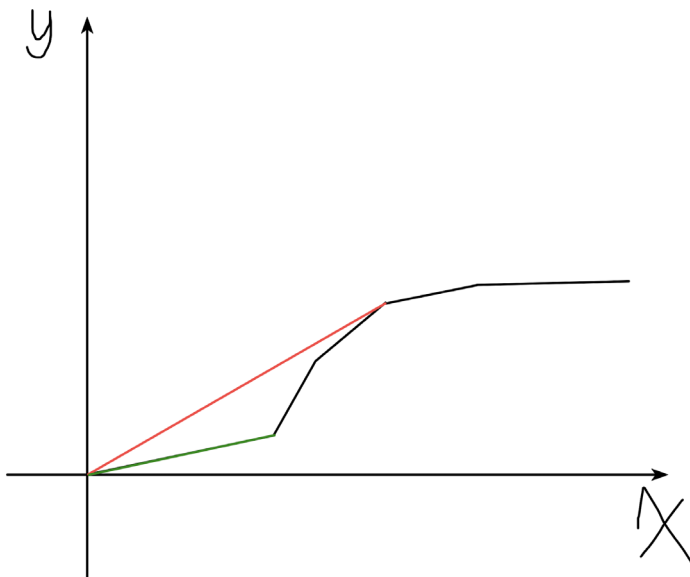
引理 5.1. 把一个 $\text{dp}_{x,i}$ 看作平面上的点 $(i, \text{dp}_{x,i})$ ，对平面上的若干个点 (x_i, y_i) ，这些点和原点连线的斜率最大值只会在它们的上凸壳中取到。

证明. 考虑三个点如果形成下凸，那么两边至少会有一个点比中间的那个点优。如此去除无用的点，最后只剩下上凸壳中的点。 $\square$

我们记 f_x 表示 x 子树内 DP 值构成的凸壳。

树上背包合并的过程可以看作是先将所有儿子的凸壳，做闵可夫斯基和。接着，在加入结点 x ，并合并一段前缀的边，来保证凸性。

⁶来源：2024-2025 集训队互测 Round 5 <https://qoj.ac/contest/1822/problem/9532>[2]



形如图片所示的过程，其中，绿色边是 $(1, a_x)$ ，后面黑色边是原凸壳上的边。

记 F_x 表示 x 子树外 DP 值构成的凸壳。

我们考虑在换根的过程中**重链剖分**维护，记结点 x 的重儿子为 $wson_x$ 。

我们在过程中维护：

- 到根路径经过的轻结点，它们的父亲的重儿子结点编号集合 S_x ；
- 到根路径经过的结点，它们的父亲不在路径上的结点子树内的凸壳构成的数据结构 G 。

假设当前在结点 x ，

1. 我们先将所有轻儿子都加入 F 里面；
2. 对于重儿子：不需要做额外处理；
3. 对于轻儿子：我们先将结点 y 在 F 内的贡献删除，再将重儿子的结点编号 $wson_x$ 加入 S_y 内。

我们考虑换根过程中如何处理 $O(\log n)$ 个数据结构中处理当前向量，维护凸壳的过程。

5.3 算法 1 基于值域的做法

我们考虑使用权值线段树来维护。

定理 5.1. 对于任意两个不同的分数 $\frac{b}{a}$ 和 $\frac{d}{c}$ ，满足 $1 \leq a, c \leq n$ 。

$|\frac{b}{a} - \frac{d}{c}|$ 的最小值为 $\frac{1}{n(n-1)}$ 。

证明. 通分之后的结果是 $\frac{|bc-ad|}{ac}$ ，由于两个是不同的分数，所以 $|bc-ad| \geq 1$ ，所以最小值为 $\frac{1}{n(n-1)}$ 。□

定理 5.2. 两个差 ≥ 1 的数下取整后不同。

由于我们维护的向量分母都 $\leq n$ ，所以我们可以对向量都乘上 n^2 ，并下取整，然后就可以使用权值线段树维护。

由于合并一定是将斜率 $\leq x$ 的全部合并，所以我们每次查询的时候，利用权值线段树的结构都相同，可以在过程中同时二分。

同时，注意到本质不同的向量只有 $O(n)$ 种，所以我们维护 G 时，区间赋零时用 tag 来维护，保留结点编号，这样就可以做到 $O(n(\log n + \log a))$ 的空间复杂度。

时间复杂度 $O(n \log n(\log n + \log a))$ ，空间复杂度 $O(n(\log n + \log a))$ 。

5.4 算法 2 不基于值域的做法

我们考虑魔改 3.3.3。

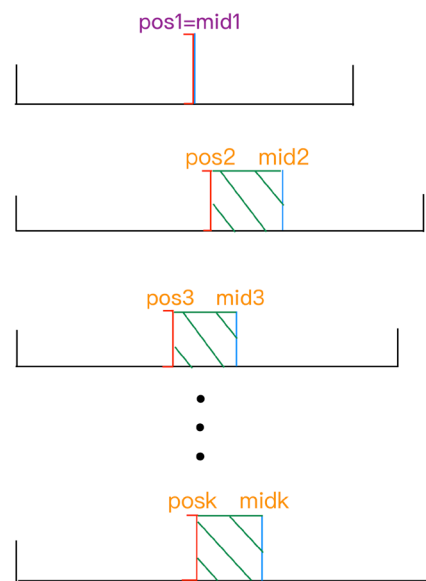
首先，假设当前向量和所有平衡树的左子树，以及树根都合并后，斜率比树根斜率最小的大。

那么，说明斜率最小的那棵平衡树的树根和右子树都不需要被合并，于是可以将其递归到左子树。

定理 5.3. 斜率最大的那棵平衡树的左子树和树根都必须要被合并。

证明. 考虑反证法，假设其不应被合并。

那么令每个数组斜率 $\geq$ 最大的位置叫 pos_i ，那么由于当前这棵平衡树树根的斜率最大，所以有 $pos_i \leq mid_i$ 。



那么说明次数将红色的前缀跟当前向量合并后的斜率 $\geq$ 平衡树斜率最大的树根的斜率，那么也一定 $\geq$ 平衡树斜率最小的树根的斜率。

此时，我们考虑加入绿色部分，由于绿色部分的斜率也都 $\geq$ 平衡树斜率最小的树根的斜率。

所以当前向量再跟绿色部分合并完后，斜率一定还 $\geq$ 平衡树斜率最小的树根的斜率。

这与我们的假设不符，说明原命题成立。

□

由于我们找斜率最大、最小的树根，需要用可删堆维护，所以这部分时间复杂度带个 $O(\log \log n)$ 。

时间复杂度 $O(n \log^2 n \log \log n)$ ，空间复杂度 $O(n \log n)$ 。

6 总结

本文由浅至深地阐述了利用树链剖分并配合多树二分维护换根信息的各种方法，并通过多道例题展现了这些方法的应用。其中例题希望和 network 是对前人研究的总结，而后续例题则是笔者的全新探索。

不过上文提到的树链剖分，还有许多性质没有能在题目中崭露锋芒；而在换根维护凸壳的部分中，笔者给出的做法仍然存在进一步优化的空间；笔者提出的方法也可能在维护

新的信息时，展现其独特价值，互相组合，创造出新的题目。

希望本文可以起到抛砖引玉的作用，让读者更好地认识换根模型，更好地感受换根问题仍然存在非常广阔的空间。期待有更多换根问题在信息学竞赛中出现。

7 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练彭思进、杨耀良、张艺缤的指导。

感谢江苏省常州高级中学曹文等老师的关心与指导。

感谢家人、朋友对我的支持与鼓励。

感谢周康阳同学、章弥炫、武林同学与我讨论以及给予我启发。

感谢章弥炫、焦思源、殷潇轩等同学为本文验稿并提出建议。

参考文献

- [1] 姚奕. 重链剖分的另一个性质 <https://www.cnblogs.com/kyecccccc/p/17829220.html>. 2023.
- [2] 孙培轩. 2024-2025 集训队互测《长野原龙势流星群》命题报告. 2024.
- [3] 赵运充. [SNOI2019] 网络题解 <https://www.luogu.com.cn/article/3jrfloyh>. 2023.
- [4] 赵海鲲. 2024-2025 集训队互测《运筹帷幄》命题报告. 2024.

一类由数据结构解决的树上字符串问题

福建省厦门双十中学 李静榕

1 前言

对于在序列上的字符串的各种问题，例如区间查询不同子串个数，已经有 SA 以及 SAM 能处理极为复杂的问题，字符串子区间的结构已经能被基本子串结构对称完整地刻画。但是对于树上字符串的问题，目前暂时没有较为通用的结构用于刻画树上字符串的结构。尽管没有通用的结构，我们依然可以借助数据结构的力量来解决很多树上字符串问题。

以下问题仅考虑 $O(n \times \text{poly}(\log(n)))$ 解法，不考虑根号算法。

2 前置知识：树上数据结构

2.1 重链剖分

树链剖分是把一棵树分解成若干条链的结构，其中重链剖分是指保留所有重边的树剖。其中的概念：

1. 一个点 u 的重儿子指的是 u 的一个儿子使得它的子树大小是所有兄弟中最大的（有多个相同则任选一个）。若 u 是叶子，则 u 没有重儿子。一个点 u 的轻儿子指的是除了重儿子的儿子。
2. 重边指的是所有非叶子节点和它的重儿子之间的边。轻边指的是除去重边的边。
3. 重链是首尾相接的重边形成的极大的链。特别的，保留终边后留下的单点也被看作重链。

每个点都属于且恰好属于一条重链。

重链剖分的核心性质：任意一个点到根的链上的轻边数量不超过 $\log(n)$ 。

证明. 考虑从根开始向下找到这个点，每经过一次轻边当前的子树大小至少会减少原先的一半，所以经过轻边的次数不超过 $\log(n)$ 。□

由此可见树上的每条路径都能被分解成不超过 $2\log(n)$ 条不包含轻边的链，在每条重链上建立线段树即可完成 $O(n\log^2(n))$ 的链修改链查询。

2.2 重链剖分的分治结构

因为每个点祖先的轻边数量之和不超过 $n\log(n)$ ，所以所有轻儿子的子树大小和也不超过 $n\log(n)$ 。

于是我们可以把与子树相关的问题转化到树剖上，并且对于每条重链分别计算，因为一条重链顶端的节点恰好是一个轻儿子（或是根）。在这条重链上可以遍历重链及重链上节点的轻儿子子树部分，只需要计算重链上的信息。

2.3 全局平衡二叉树

如果把重链看作一个节点，轻边看作重链之间的连边，那么整棵树会重新形成一个分治结构树。对于类似链加链求和的问题，重链剖分可以在每条重链上建立线段树以支持修改，但关注其分治的特点，对于一条重链应当建立一个以重链节点为叶子、重链节点的轻儿子子树大小和为权重的更加平衡的二叉树。二叉树的根为重链本身，其他节点为重链的一个区间，即一个更小的链。这棵二叉树形成的过程可以看作不断将链裂开成两条链。每次选取分裂位置时需要尽量让分裂出来的两条链的权重较为平衡，即权重和之差的绝对值最小。

下证明在一条重链上由上述方法形成的二叉树上任意存在二级祖先的非叶子节点 w 和它的二级祖先 u 满足 $2val_w \leq val_u$ 其中 val 代表权重。

证明. 设 w 的父节点为 v 。若 $2val_v \leq val_u$ ，则证明结束。不妨设 v 是 u 的左儿子，设 v 子树内最靠右的叶子权重为 y ，剩余权重和为 x 。能发现 $\max(val_w, val_v - val_w) \leq \max(x, y)$ ，其中一定有 $2x \leq val_u$ 。若 $2y > val_u$ 因为 w 不是叶子节点所以一定是 v 的左儿子，证明结束；否则也有 $2y \leq val_u$ ，则一定有 $2val_w \leq val_u$ 。□

将这些二叉树通过轻边连接得到一棵重构树，若将二叉树叶子节点下轻边建立哈夫曼树则可以形成一棵二叉树，即全局平衡二叉树。

注意到原树的一个节点对应的全局平衡二叉树上节点祖先中的边有一些是由重链分治形成，其他是由轻边的二叉树形成；向上连续经过两个重链的边当前权值会翻倍，轻边形成的二叉树边同理。轻重交替不超过 $\log(n)$ 次，所以这个节点祖先的边的数量不超过 $k\log(n)$ 其中 k 是一个小常数。

3 序列字符串的开端

3.1 后缀数组^[2]

后缀数组是将一个字符串的所有后缀按照字典序大小从小到大排序后的下标数组，求后缀数组的方法有很多，最简单的可以通过二分加哈希实现后缀的比较以实现后缀排序。

令 sa_i 为后缀数组的第 i 个元素, 即字典序第 i 小的后缀下标; 定义 $height_i = \text{lcp}(sa_i, sa_{i+1})$, 由字典序的性质可以发现 $\text{lcp}(sa_i, sa_j) = \min_{k=i}^{j-1} sa_k$ 其中 $i < j$ 。值得一提的是字符串不同子串的个数为 $\frac{n(n+1)}{2} - \sum_{i=1}^{n-1} height_i$ 。

3.2 后缀树^[3]

本文为了方便讨论, 将后缀树定义为后缀 trie, 而不是后缀 trie 压缩后的结果。

后缀树是由所有后缀形成的 trie 树。后缀树的节点个数和边数可能是 $O(n^2)$ 的, 但是由于后缀树的叶子个数不超过 n 个, 所以所有叶子形成的虚树的大小不会超过 $2n$ 个, 虚树中一条边即为原本后缀树上的一条链 (自上而下)。

定义字符串的后缀在后缀树上的端点为 end 节点, 则按照先序遍历遍历到的 end 节点的列表形成了后缀数组。

3.3 后缀自动机^[4]

把后缀树的 end 节点看作接受状态, 根看作初始状态, 后缀自动机是由后缀树形成的自动机的极小化, 即接受语言相同的最小的自动机。它最多包含了 $2n - 1$ 个状态和 $3n - 4$ 个转移。

4 例题 1 研心^[5]

4.1 题意简述

给定大小为 n 的字符串序列 S 和大小为 m 的字符串序列 T , 其中 S 的第 i 个字符串为 S_i , T 的第 j 个字符串为 T_j 。

定义一个字符串的权值 $f(s)$ 为 s 中最长奇回文子串的长度。

定义两个字符串的加法 $s + t$ 为把两个字符串拼接起来得到的新字符串。

求 $\sum_{i=1}^n \sum_{j=1}^m f(S_i + T_j)$ 。

4.2 题目分析

此题相较于一般序列上字符串问题的区别在于在 S, T (左右) 两棵 trie 树上统一处理问题, 而不是每个字符串分别解决。进行基本转化后题目变为近似于偏序的问题, 建立 trie 树后可以利用数据结构将问题集中处理。

4.3 具体做法

首先考虑如何统计一个被缝合的串的奇回文长度。分成三段, 一个是左边最长回文, 右边最长回文, 中间的回文。

中间的回文可以通过回文中心区分成左右两边, 以左边为例: 我们枚举左边串的回文后缀, 将这个串去掉这个后缀后倒着对右侧匹配, 匹配最长的是其中一个奇回文串, 这里用二分哈希实现。匹配过程可以看作左边的串在右边的 m 个串形成的正 trie 进行匹配。

设左串个数为 n , 右串个数为 m , 考虑求出一个 $n \times m$ 的矩阵表示答案的矩阵, 最终进行求和。

我们对于上述总共四种回文对矩阵进行覆盖 (即 $chkmax$), 例如左边最长回文是行覆盖, 右边最长回文是列覆盖, 回文中心在左侧是对一行进行一个 trie 树上的覆盖, 回文中心在右侧是对一列进行一个 trie 树上的覆盖。

行和列的交互是非常困难的, 首先考虑如何在只考虑行操作之后求出答案。

试图把一个可重的覆盖变成不重的覆盖, 即多个“连续段”的值直接覆盖。单个左侧回文中心的操作具体是在 trie 树上对一条路径 S 进行如下操作:

设左回文后缀为 l , 那么对右侧 S 路径上深度为 d 的点 u 的子树进行关于 $2d + l$ 的 $chkmax$ 。这里 d 的系数直接钦定为 1 (即原题求最长回文半径)。

提取出所有的 u 建立虚树, 考虑所有在虚树上 (包括长链上的) 的 trie 树节点得到的权值。这个权值的意义是子树内所有 trie 树结束节点 (例如固定左边那么 trie 树上就是 T 的 m 个结束节点) 对应的最长回文对这个权值取 $\max$ 。

考虑这么限制权值的值, 使得每个点 u 的权值掌控 (能真正更新 $\max$) 的部分是 u 的子树除去子树节点掌控的部分。进行两轮更新:

1. $chkmax(val_{fa_u}, val_u + 1)$

2. $chkmax(val_u, val_{fa_u})$

虚树上虚边对应的链的权值是形如 1, 1, 1, 2, 3, 4 的形式, 即一段相同权值接一段公差为 1 的等差数列。这样原 trie 树上每个点的权值都能被唯一覆盖, 覆盖形式要不然是直接赋值 (上面序列平的部分) 或者是一个嵌套的结构, 形式化地讲: 设这条祖先-儿子方向的链为 C , 第 i 个点是 C_i , 那第 i 个点掌控的范围是 $\text{subtree}(C_i) - \text{subtree}(C_{i+1})$, 权值是 $i + k$, k 是个关于这条链定值。

考虑把上述覆盖转到树剖上, 其中第二种会拆成较多标记, 忽略分讨过程之后, 总的来说可以总结成三种:

1. 单点以权值 k 覆盖

2. 重链的一个区间的所有点和他们的轻儿子以权值 k 被覆盖

3. 重链的一个区间的所有点和他们的轻儿子以重链上的点的深度 $+k$ 为权值被覆盖

注意这种标记会有 $1/-1$ 的系数。当一个位置被权值为 $\{a_i\}$ (如 $\{1, -1, 2, -2, 3\}$) 的标记覆盖, 那么真正的标记权值是 $\sum a_i$ (上个例子是 3)。实现时保证标记集合可以通过消掉相反数的对最终得到至多一个正数标记。

然后进入纯 ds 部分。把树剖的重链拆成全局平衡二叉树，上述标记最终被拆成 $O(\log)$ 个子标记，每个标记都在一个全局平衡二叉树节点上，因为此时标记的类型相同（只需要考虑不同的 k ）。考虑 s 串产生的在 t-trie 上的标记和 t 串产生的在 s-trie 上的标记。将前者挂在 s-end（即 s-trie 的结束节点）上，它有对应的在 t-trie 全局平衡二叉树上的一个标记；将后者的标记部分挂在 s-trie 的全局平衡二叉树上，它相应的有一个对应的 t-end。考虑特定的一对 $s-t$ ，我们想知道第一种标记和第二种标记的最大值。但是尽管上文提到这种标记是不重的覆盖，但实现中我们并不能做到真正的不重，是通过带系数的权值之间取 max 得到的。

具体的，这种标记可以看作 (val, k) 其中 k 表示系数。

两个标记取 max 结果为 $(\max(val_1, val_2), k_1 k_2)$ 。可以证明这种标记具有交换律和结合律，更重要的，具有分配律。

对于这个特定的 $s-t$ ，两侧的标记实际上是标记集合，但是因为标记具有分配律，所以可以求出两个集合两两标记的最大值之后再求和。关注 s 的标记覆盖到 t-end 且 t 的标记覆盖到 s-end 的条件，枚举 s-trie 全局平衡二叉树的节点 u ，枚举所有 t 的标记覆盖在 u 上的标记，得到一种暴力做法：先枚举 u 覆盖下的节点 v （这个节点是原 s-trie 树节点），并枚举 v 作为 s-end 时所有对 t-trie 全局平衡二叉树的标记挂到这个二叉树节点 x 上，然后枚举标记覆盖 u 的对应的所有 t-end 节点 w ，枚举 w 的所有祖先和之前挂的标记取 max 后求和。因为查询一个 t-trie 全局平衡二叉树节点 x 上所有比某个标记大的标记的和需要二分，所以到这里最终复杂度为 $O(s \log^3 s)$ 。

实际上可以优化掉这个二分。首先考虑 s-end 对 t-trie 全局平衡二叉树的标记，放到同一个 x 上的标记需要排序，所以可以换成在 u 时先将所有 v 上的标记放到一起整体排序后按顺序添加标记，这个排序可以通过从 u 的儿子继承排序好的标记进行归并替换。对于 u 上标记对应的 w 向上查询的过程，我们先把查询挂在 w 上，并遍历 t-trie 全局平衡二叉树有查询的部分，从下至上向上归并查询，这样同时查询一个 x 的查询是排好序的，可以把标记和查询进行归并，同样求出 max 之和。总复杂度为 $O(s \log^2 s)$ 。

4.4 启发

出题人在试图解决这道问题时长时间未想出 $O(s \times \text{poly}(\log(s)))$ 做法，但尽管最终做法极为复杂，问题的本质依然只是一个复杂偏序问题。这提示我们即使没有树上字符串的通用结构也能使用数据结构让问题变为可做问题。

5 深入树上字符串结构

实际上相较于真正的树上字符串问题，上述问题显得过于浅显，因为对于字符串的匹配仅停留在了根开始的匹配上。以下详细定义树上字符串：

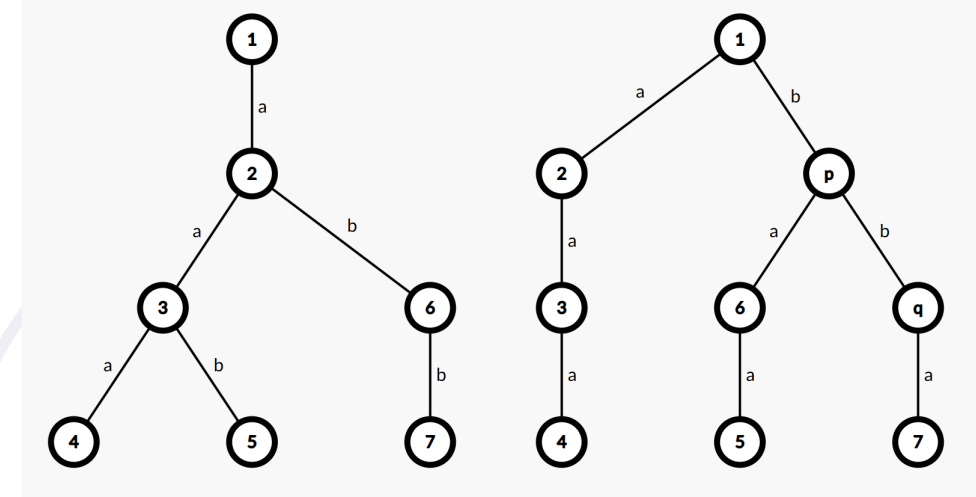
树上字符串是一棵边有边权的有根树，边权为字符。一个子串可以表示为二元组 (u, v) ，其中 v 是 u 的祖先。

(a, b) 是 (c, d) 的前缀当且仅当 $a = c$ 且 d 是 b 的祖先。 (a, b) 是 (c, d) 的后缀当且仅当 $b = d$ 且 a 是 c 的祖先。

一个节点 x 的子树的所有字符串定义为集合 $\{(u, v) | \forall u, v \in \text{subtree}(x), v \in \text{anc}(u)\}$ 。

一个节点 x 的子树的后缀集合定义为 $\{(u, x) | \forall u \in \text{subtree}(x)\}$ 。

一个树上字符串的后缀就是整棵树的后缀。



上图左侧树为树上字符串，右侧为树上字符串的后缀形成的后缀树，即根的子树对应的后缀树。

从一个问题入手：给定树上字符串，多次查询，每次查询两个子串的最长公共前缀。

除去直接哈希的做法，可以发现树上字符串具有和序列字符串相似的性质，都能求出“后缀数组”。具体地，考虑所有子串 (u, root) ，这些串能够形成和序列字符串的所有后缀一样的结构，都能求出后缀数组，都能形成一棵后缀树。

于是此问题就变成后缀数组上区间最小值问题或者是后缀树上的 lca 问题。能发现这个后缀树的所有前缀（前缀链）形成了这个树上字符串的所有子串。

能发现在此背景下树上字符串问题的强度并没有和序列字符串问题有本质区别，下面给出一个无法被树上字符串所有后缀形成的后缀树结构简单解决的问题。

6 例题 2 子树串计数

给定一个树上字符串，问每个子树中本质不同的串的个数。

6.1 题目分析

沿用对于整个树上字符串的后缀树结构，我们希望对每个子树都建立出后缀树结构，并且本题要求的就是每个子树的后缀树的大小（即边数）。

6.2 具体做法

先给整个树上字符串的后缀排序，方便之后的操作。

考虑如何维护一个子树的后缀树。因为整个字符串的后缀树包含了所有小的后缀树的信息，所以考虑从根开始向下不断分裂维护后缀树。

对于分裂过程，可以使用启发式分裂，也就是沿着树的重链进行分裂，仅保留重儿子的信息，对于轻儿子则递归下去重新计算。

设当前维护的是节点 u 的后缀树， u 的重儿子为 v 。则将 u 的后缀树变为 v 的后缀树的过程为

1. 删除 u 当中在轻儿子中的字符串后缀 end ，并仅保留后缀树的有用部分。
2. 将所有后缀树的 end 节点向父亲移动一次，即变为自己的父亲，并保留后缀树的有用部分。

向父亲移动一次的操作不好维护，但是考虑到答案只关心后缀树的大小，所以只需要维护后缀树的所有叶子。对于第一个操作，删除一个节点可能会导致原本不是叶子的 end 变为叶子，但是一次删除操作最多新加一个叶子，而删除操作次数是合理的，所以这里复杂度正确；对于所有叶子向上移动一次的操作，可能会导致一些叶子不再是叶子变为别的叶子的祖先，因为添加叶子的次数合理，所以删除任意次叶子的操作都是合理的。

具体而言，我们需要用 set 来维护所有叶子，按照字典序排序，即这些叶子在后缀树上的 dfs 序。这形成了整个后缀树的形态，后缀树的大小即节点深度和减去相邻叶子 lca 深度和。

对于删除叶子的操作，考虑如何寻找用于替代的节点。设删除的点是 u_2 ，设在 sa 上 u_2 左边是 u_1 ，右边是 u_3 。我们希望找到一个不是叶子的、并且还在当前原树子树内的、深度最大的、是 u_2 祖先的 end 节点来替代 u_2 。这里有一个细节条件，这个新节点 v 必须要是 u_2 的祖先。这个条件的限制方式为限制 rk_v 在 $[l, r]$ 中，其中 $[l, r]$ 区间是由限制一个节点与 u_2 的 lcp 长度至少超过了 $\max(lcp(u_1, u_2), lcp(u_2, u_3))$ 从 rk_{u_2} 开始向左右两边二分得到，区间查询用线段树维护即可。

对于向父亲移动的操作，我们需要维护所有未来叶子凋亡的事件并推入一个小根堆中用于提取事件。一个叶子凋亡事件是由一个叶子 u 变为和它相邻的叶子的祖先引发的，可以看出所有叶子凋亡的事件都是由相邻叶子引发的。每次修改维护叶子的 set 时一起维护新增的叶子相邻对即可维护出所有可能的凋亡事件（叶子被删除事件会有凋亡事件失效）。

叶子相邻 lca 深度和也是在相邻叶子上维护的信息，与上述事件一同维护即可。

最终总复杂度为原树树剖 $\log$ 和上述所有操作总共的一个 $\log$ 。二分的单 $\log$ 需要一个后缀数组上的 st 表。总复杂度为 $O(n \log^2 n)$ 。

6.3 启发

观察问题的结构，虽然该问题已经不能被树的后缀数组或后缀树简单解决，但是问题只保留了后缀树中的叶子节点。是否存在一种结构能维护原树所有子树的整个后缀树的信息？

目前我给出的答案是，存在一种结构能维护一些种类的信息，并且强度要强于单纯的树上字符串的后缀数组或后缀树结构。

7 不完整树上后缀树结构

考虑对于上述从上至下维护后缀树的思路，将 set 维护的后缀树叶子改为用树剖维护的后缀树。虽然后缀树的边数个数无保证，但叶子个数有保证，所以依然可以进行树剖。

树剖之后考虑删叶子和向上移动两种操作。先考虑向上移动的操作，在树剖的结构上我们需要暴力处理一个轻儿子向上移动到父亲的过程，即删一个点再加一个点，此处看出删叶子的操作是不重要的；对于重儿子向上移动的过程，则需要整体维护。考虑一条重链的信息，重链上有若干个点正在向上移动，重链的轻儿子的点集固定。也就是说这里的信息被分成了移动中的重链 end 节点和不动的轻儿子 end 节点两个部分。设这两部分为运动部分和静止部分。考虑这个结构的强度：

1. 是否能维护后缀树有效节点的个数（子树内 end 节点个数大于零的节点个数）？
2. 是否支持查询子树内为一个串前缀的子串的个数？
3. 是否能维护后缀树子树内 end 节点个数平方之和？

对于第一个问题，考虑树剖维护出所有叶子消失的事件，树边长度将简单维护。

对于第二个问题，若定义子树大小为子树中 end 节点的个数，则相当于查询后缀树的树链（根开始）的子树大小和。而子树大小可以拆成运动部分提供的子树大小和静止部分提供的子树大小，都可以用树剖处理。

对于第三个问题，能发现对于一条后缀树重链，重链上移动节点与重链的轻子树之间会产生贡献，这是一个差卷积过程，不能用此结构维护。从另一个角度看，若仅考虑后缀树的叶子节点，那么答案需要计算 $\sum_{i=1}^n \sum_{j=i+1}^n de_{lca(leaf_i, leaf_j)}$ ，因为叶子已经按照 $rank$ 排好序，所以两点的 de_{lca} 相当于区间最小值。

而到这里此问题的强度已经大于题目大海的深度^[6]，因为还未考虑非叶子节点产生的混乱的影响。

相比之下，普通的树上 sa 难以处理前两个问题，可以说明这个结构的强度严格强于树上 sa ；而对于第三个问题并不能给出答案，说明此结构不能处理运动部分信息和静止部分

信息之间会产生贡献的问题，因为会产生无法快速计算的动态差卷积；若无需维护此类全局信息（例如子树大小 k 次方和等）或运动部分信息与静止部分信息没有贡献（例如第二个问题），这个结构可以给出答案。

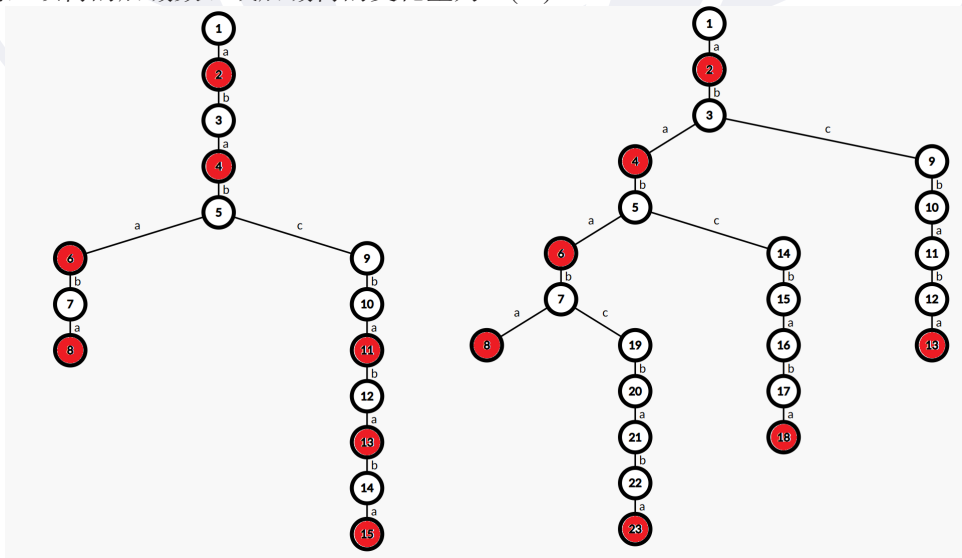
8 数据结构的尽头

通过以上两个例证和一个结构能看出数据结构的作用极为关键，我们仅仅借助最简单的 trie 树就解决了以上两个问题。但是以上所有问题和结构都很有可能隐藏着暂未开发的性质。例如对于研心，整道题并未使用过回文串的性质；对于子树串计数和不完整树上后缀树结构，不同串之间在不同子树的 trie 树（后缀树）上的重合很可能隐藏着更深层次的内在结构。而数据结构仅仅是一个强大的工具，在上述结构的基础上无法维护差卷积过程，也无法触及可能的更深层次的结构。

9 结构之上

尽管数据结构没能得到一个处理所有树上字符串问题的结构，我们依然可以在对序列的结构进行模仿后试图找出可能的最终结构。

对于后缀数组和后缀树，我们试图对其进行模仿以维护出原树每棵子树的后缀数组和后缀树。若最简单地考虑维护后缀数组的元素变化或后缀树的虚树的变化，以下给出一个反例，该树的后缀数组或后缀树的变化量为 $O(n^2)$ 。



上图为反例的缩小版本。左侧代表树上字符串，右侧代表树上字符串的后缀树的一部分，红点表示考虑的后缀以及在后缀树上对应的 end 节点。在沿着重链向下分裂的过程中，每次后缀树虚树的形态都会产生剧烈变化，每个原树后缀的排名也会发生剧烈变化。右侧

2, 4, 6, 8 号节点会向上运动，和 13, 18, 23 号节点的后缀数组排名大小关系会不断变化，可以看出变化次数为 $O(n^2)$ ；后缀树的形态也随着重链向下递归的过程不断变化，变化量同样是 $O(n^2)$ 。

笔者的研究也在此停止，但通过这个反例可以看出后缀数组以及后缀树的变化十分有规律，这是由后缀树上树链中密集的 end 节点造成的小周期形成的；但笔者暂未研究对于 end 节点较为稀疏乃至一般情况下是否还能在合理的时间复杂度下维护后缀树树链上运动节点与静止节点之间的深度关系。

除去维护后缀数组及后缀树的思路，还有至少另外两种可能思路：

1. 维护后缀自动机的变化量
2. 抛弃原本树上字符串所有后缀的结构，转而尝试所有树上节点向下的前缀树（trie 树）

之并

对于后缀自动机笔者暂不清楚是否可以维护，但是因为后缀树结构本身无法维护，维护后缀自动机的思路很可能行不通或无法维护所有信息；对于第二种思路，其结构过于复杂也很可能难以维护。

希望读者能够更进一步地探究以上问题及结构的内在本质，或找到全新的思路尝试彻底解决树上字符串子树信息维护的问题。

10 参考资料

- [1] OI-wiki 树链剖分 <https://oi-wiki.org/graph/hld>
- [2] OI-wiki 后缀数组 <https://oi-wiki.org/string/sa>
- [3] OI-wiki 后缀树 <https://oi-wiki.org/string/suffix-tree>
- [4] OI-wiki 后缀自动机 <https://oi-wiki.org/string/sam>
- [5] 2024-2025 集训队互测 Round 3 研心 <https://qoj.ac/problem/9514>
- [6] 【UNR #8】大海的深度 <https://uoj.ac/problem/892>

11 致谢

感谢中国计算机协会提供学习和交流的平台。

感谢施涵森同学提供了研心的 idea。

感谢陈昕阳、仲煦北、史家丞、陈旭磊同学与我的沟通交流。

感谢家人对我的关心和支持。

浅谈一类提取有理分式系数的方法

广东实验中学 黄建恒

摘要

生成函数作为信息学竞赛中的常见考点，近年来有较为深入的研究。使用生成函数时经常需要提取其中的某一项系数，对于不同类型的函数，有不同的方法解决这一问题。本文将介绍一种提取有理分式系数的做法，利用分母的根进行部分分式分解。在前两章介绍两种常见用途，并在第三章列举两个和拉格朗日反演一起使用进行解题的例子。

1 线性递推数列的通项公式

定理 1.1. 对于无限数列 $f = \{f_0, f_1, \dots\}$ ，令 $F(x) = \sum f_i x^i$ 。对于 $Q(x) = 1 + q_1 x + q_2 x^2 + \dots + q_k x^k$ ，以下四个对 f 的条件限制是等价的：

- (i) 存在多项式 $P(x)$ 满足 $\deg P < \deg Q = k, F(x) = \frac{P(x)}{Q(x)}$ 。称其为 $F(x)$ 的有理分式形式。
- (ii) $\forall n \geq 0, \sum_{i=0}^k f_{n+k-i} q_i = 0$ 。即 f 是线性递推数列， q 是其线性递推式。
- (iii) 设方程 $Q(x) = 0$ 的每个根去重后为 $\xi_1, \xi_2, \dots, \xi_l$ ，各出现 $d_1, d_2, \dots, d_l$ 次，则可以将 $F(x)$ 写为 $\sum_{i=1}^l \frac{P_i(x)}{(1-\xi_i^{-1}x)^{d_i}}$ 的形式，满足 $\deg P_i(x) < d_i$ 。称其为 $F(x)$ 的部分分式分解。
- (iv) 存在一组 $P_i(x)$ ，满足 $f_n = \sum_{i=1}^l P_i(n) \xi_i^{-n}$ ， $\deg P_i(x) < d_i$ 。

该定理的证明见 [1] 定理 4.1.1。

定理 1 说明，对于一个线性递推数列 f ，可以先写出其生成函数的有理分式形式，对其进行部分分式分解，进而得到数列的通项公式。当 $n \rightarrow \infty$ 时。若 $|\xi|$ 最小的一项为实数，则对 f 的渐进复杂度分析只需保留这一项。特别地，若其他 ξ_i 均满足 $|\xi_i| > 1$ ，则完全可以忽略其他项的影响。

例题 1.1. 计算递推关系式 $T(n) = T(n-1) + T(n-3) + 1$ 的渐进复杂度。¹

¹Potyczki Algorytmiczne 2022, runda trzecia, Mędrycy / 洛谷 P9257 的时间复杂度分析

构造数列 $f_0 = f_1 = f_2 = 1, \forall n \geq 3, f_n = f_{n-1} + f_{n-3}$ 。令 $F(x) = \sum_{i \geq 0} f_i x^i = \frac{1}{1-x-x^3}$ 。方程 $1-x-x^3=0$ 的根满足 $\xi_1 \approx 0.682, |\xi_2|, |\xi_3| > 1$ 。因此 ξ_2^{-n}, ξ_3^{-n} 对 f_n 的影响趋近于 0， $f_n = c_1 \xi_1^{-n} + c_2 \xi_2^{-n} + c_3 \xi_3^{-n} \approx c_1 \times 1.456^n$ 。

由于 $T(n) = \sum_{0 \leq i \leq n} f_i$ ， $1.456 > 1$ ，等比数列求和不影响复杂度，所以 $T(n) = \Theta(1.456^n)$ 。

推论 1.1. 对于递推数列 $f_n = \sum_{i=1}^k q_i f_{n-i}$ ，满足 q_i 非负， $q_k > 0$ ，设 q 非 0 的位置集合为 S ，满足 $\gcd(S) = 1, f_0 \sim f_{k-1}$ 非负且非全 0，设方程 $Q(x) = 1 - q_1 x - \dots - q_k x^k = 0$ 的唯一正实根为 ξ 。那么当 $n \rightarrow \infty$ 时， $f_n = \Theta(\xi^{-n})$ 。

考虑 $Q(0) = 1, x \rightarrow \infty$ 时 $Q(x) < 0$ ，且 $Q(x)$ 在 $[0, \infty)$ 上单调，因此有唯一正实根 ξ ，同时， $Q(x)$ 所有根的模长最小值也是 ξ ，因为 $x \in [0, \xi)$ 时 $Q(x) > 0$ 。且除 $\xi e^{\frac{2\pi ni}{\gcd(S)}}$ 外无法达到这一最小值。由于 $\gcd(S) = 1$ ，只有 ξ 可能为 $Q(x)$ 的根，因此其他 ξ_i 不影响渐进复杂度。

例题 1.2. 给定正整数 $n \leq 10^{18}$ ，计算 $\lfloor (1 + \sqrt{2})^n \rfloor \bmod 10^9 + 7$ 。

令 $a = 1 + \sqrt{2}$ ，满足 $(a-1)^2 = 2, a^2 - 2a - 1 = 0$ 。考虑逆用定理 1，构造函数 $F(x) = \frac{1}{1-(1+\sqrt{2})x} + \frac{1}{1-(1-\sqrt{2})x} = \frac{2-2x}{1-2x-x^2}$ ，则数列 f 满足 $f_0 = f_1 = 2, f_n = 2f_{n-1} + f_{n-2}$ ，通项公式 $f_n = (1 + \sqrt{2})^n + (1 - \sqrt{2})^n$ 。不难使用矩阵快速幂或 Boston-Mori 算法求出 f_n ，而 $-1 < 1 - \sqrt{2} < 0$ ，因此答案为 $f_n - \lfloor n \bmod 2 = 0 \rfloor$ ，时间复杂度 $O(\log n)$ 。

值得一提的是，由于 $\sum \lfloor n \bmod 2 = 0 \rfloor x^n = \frac{1}{1-x^2}$ ，因此答案的生成函数 $G(x) = \frac{2-2x}{1-2x-x^2} - \frac{1}{1-x^2}$ 同样可以线性递推，递推式长度为 4。因此只用求出 $g_0 \sim g_7$ ，即可使用 Berlekamp-Massey 算法求出最短递推式，时间复杂度 $O(\log n)$ 。

2 提取二元有理分式系数

定义 2.1. 设 K 为一个代数闭域（例如复数域 $\mathbb{C}$ ）。

$K[x]$ 为 K 上关于不定元 x 的多项式环，即 $F(x) = \sum_{0 \leq n \leq n_0} f_n x^n$ 。

$K(x)$ 为关于 x 的有理函数域（ $K[x]$ 的商域），即 $F(x) = \frac{P(x)}{Q(x)}$ ， $P(x), Q(x) \in K[x]$ 。

$K[[x]]$ 为关于 x 的形式幂级数环，即 $F(x) = \sum_{n \geq 0} f_n x^n$ 。

$K((x))$ 为关于 x 的形式洛朗级数域（ $K[[x]]$ 的商域），即 $F(x) = \sum_{n \geq n_0} f_n x^n, n_0 \in \mathbb{Z}$ 。

$K^{fra}[[x]]$ 为关于 x 的分式幂级数环，即 $F(x) = \sum_{n \geq 0} f_n x^{n/N}, N \in \mathbb{Z}^+$ 。

$K^{fra}((x))$ 为关于 x 的分式洛朗级数域（ $K[[x]]^{fra}$ 的商域），即 $F(x) = \sum_{n \geq n_0} f_n x^{n/N}, n_0 \in \mathbb{Z}, N \in \mathbb{Z}^+$ 。

对于二元有理分式 $F(x, y) = \frac{P(x, y)}{Q(x, y)}$ ，满足 $P(x, y), Q(x, y) \in K[x, y]$ ，设 $Q(x, y) = \prod_{i=1}^l (y - \xi_i)^{d_i}$ ，即 ξ_i 为关于 y 的方程 $Q(x, y) = 0$ 的根，我们首先有如下定理。

定理 2.1 (Puisseux's Theorem). $K^{fra}((x))$ 是代数闭域，同时是 $K((x))$ 的代数闭包。

这说明,对于每个形如 $P_0(x)+P_1(x)y+\dots+P_k(x)y^k=0$ 的关于 y 的方程,满足 $P_i(x) \in K((x))$, 方程的解 $\xi \in K^{fra}((x))$ 。

定义 2.2. 设 K 为一个代数闭域。

称 $F(x) \in K^{fra}((x))$ 是代数的², 当且仅当存在一组非全 0 的 $P_0(x), P_1(x), \dots, P_k(x) \in K[x]$, 使得 $P_0(x) + P_1(x)F(x) + \dots + P_k(x)F(x)^k = 0$ 。

定理 2.2. 设 f, g 为两个非 0 代数函数, 则 $f+g, fg, \frac{1}{f}$ 均为代数函数。

因此, $Q(x, y) = 0$ 的每个根 $\xi_i \in K^{fra}((x))$ 都是代数的, 因为 $Q(x, \xi_i) = 0$ 。

那么 $F(x, y) = \frac{P(x, y)}{Q(x, y)} = \sum_{i=1}^l \frac{N_i}{(y-\xi_i)^{d_i}}$, 其中 $N_i \in K(\xi_1, \xi_2, \dots, \xi_l)[x, y]$, 可以通过待定系数法求出。由于代数函数相加, 相乘, 取倒数仍然是代数函数, 因此取任意 $[y^b]N_i$ 均是代数的, $N_i \in K^{fra}((x))[y]$ 。例如, 当所有 $d_i = 1$ 时, 有特解 $N_i = P(x, y) \prod_{j \neq i} \frac{1}{\xi_i - \xi_j}$ 。这可以通过代入 $y \rightarrow \xi_i$ 的极限得到。

定义 2.3. 二元幂级数 $F(x, y) = \sum_{i,j \geq 0} f_{i,j} x^i y^j$ 的对角线 $\text{diag } F = \sum_{n \geq 0} f_{n,n} x^n$ 。

定理 2.3. 二元有理分式 $F(x, y) \in K[[x, y]] \cap K(x, y)$ 的对角线是代数的。

考虑 $G(x, y) = F(x/y, y)$, 那么 $\text{diag } F = [y^0]G(x, y)$ 。

如果 F 为有理分式, 那么 G 也为有理分式, 代入 $x' = x/y$ 后只需 P, Q 同乘充分多的 y 即可。

与之前的过程相同, 得到 $G(x, y) = \sum_{i=1}^l \frac{N_i}{(y-\xi_i)^{d_i}}$ 的形式。然后将 ξ 分为 α_i 和 β_j 两类。如果 ξ 的每一项的次数均为正数, 则分入 α , 否则分入 β 。形式化地, $\alpha \in x^{1/r} K[[x^{1/r}]], \beta \notin x^{1/r} K[[x^{1/r}]], r \in \mathbb{Z}^+$ 。那么有 $\alpha, \beta^{-1} \in K^{fra}[[x]]$ 是代数的。

$$\begin{aligned} G(x, y) &= \sum_{i=1}^{l_1} \frac{a_i}{(y-\alpha_i)^{c_i}} + \sum_{j=1}^{l_2} \frac{b_j}{(y-\beta_j)^{d_j}} \\ &= \sum_{i=1}^{l_1} \frac{A_i}{(1-\alpha_i y^{-1})^{c_i}} + \sum_{j=1}^{l_2} \frac{B_j}{(1-\beta_j^{-1} y)^{d_j}} \\ &= \sum_{i=1}^{l_1} A_i \left(\sum_{k \geq 0} \binom{c_i+k-1}{c_i-1} \alpha_i^k y^{-k} \right) + \sum_{j=1}^{l_2} B_j \left(\sum_{k \geq 0} \binom{d_j+k-1}{d_j-1} y^k \beta_j^{-k} \right) \end{aligned}$$

我们有 $A_i = \frac{a_i}{y^{c_i}}, B_j = \frac{b_j}{(-\beta_j)^{d_j}}$, 由于 $N_i \in K^{fra}((x))[y]$, 因此 $A_i, B_j \in K((x^{1/r}))[y, y^{-1}]$, 最后得到的 $G(x, y) \in K[[y^{1/r}, (x/y)^{1/r}]] [y^{-1/r}] \supset K[[x/y, y]]$ 。因此上述的操作都是合理的。

最后提取 $\text{diag } F = [y^0]G(x, y)$, 由于任意 $[y^b]N_i$ 都是代数的, 因此 $\text{diag } F$ 是代数的, 且 $F(x, y) \in K[[x, y]]$, 所以 $\text{diag } F$ 是代数幂级数。

²在 [1] 等文献中要求 $F(x) \in K((x))$ 。事实上, 在本文中影响不大。

推论 2.1. 对于二元有理分式 $F(x, y) \in K[[x, y]] \cap K(x, y)$, 非负整数 k, b , 令 $G(x) = \sum_{n \geq 0} ([x^n y^{kn+b}] F(x, y)) x^n$ 。则 $G(x)$ 是代数的。

即 $G(x) = [y^b] F(x/y^k, y)$, 由于 ξ 是代数的, 因此 $b > 0$ 结论仍然成立。

例题 2.1. 计算德兰诺依数 (Delannoy numbers) $D(x, y) = \frac{1}{1-x-y-xy}$ 的对角线。

令 $G(x, y) = D(x/y, y) = \frac{1}{1-x/y-y-xy} = \frac{y}{y-x-y^2-xy}$, 则只用求出 $[y^0]G(x, y)$ 。

设 $y^2 + (x-1)y + x = 0$ 的两根为 $\xi_1 = \frac{-x+1-\sqrt{\Delta}}{2}, \xi_2 = \frac{-x+1+\sqrt{\Delta}}{2}$, 其中 $\Delta = x^2 - 6x + 1$ 。由于 $[x^0]\Delta = 0$, 令 $\alpha = \xi_1, \beta = \xi_2$ 。

$$\begin{aligned} G(x, y) &= \frac{y}{y-x-y^2-xy} \\ &= \frac{y}{-(y-\alpha)(y-\beta)} \\ &= \frac{y}{\beta-\alpha} \left(\frac{1}{y-\alpha} - \frac{1}{y-\beta} \right) \\ &= \frac{1}{\Delta} \left(\frac{1}{1-\alpha y^{-1}} - \frac{y\beta^{-1}}{1-\beta^{-1}y} \right) \\ \text{diag } F &= [y^0]G(x, y) = \frac{1}{\sqrt{\Delta}} = \frac{1}{\sqrt{x^2 - 6x + 1}} \end{aligned}$$

同时可以求出递推式, 由于 $(\frac{1}{\sqrt{\Delta}})' = \frac{\Delta'}{2\Delta\sqrt{\Delta}}$, 即 $(\frac{1}{\sqrt{\Delta}})\Delta = \frac{\Delta'}{2\sqrt{\Delta}}$ 。对比等式两边系数, 设 $f_n = [x^n] \frac{1}{\sqrt{\Delta}}$, 得到

$$\begin{aligned} 2(nf_n - 6(n-1)f_{n-1} + (n-2)f_{n-2}) &= -6f_{n-1} + 2f_{n-2} \\ nf_n &= 3(2n-1)f_{n-1} - (n-1)f_{n-2} \end{aligned}$$

事实上, 对于分母上 y 的次数至多只有 2 的 $G(x, y)$, 还可以较快地递推 $[y^b]G(x, y)$ 。

例题 2.2. 冲刺³

有一个点初始在 $(0, 0)$, 以向右为 x 轴正方向, 向上为 y 轴正方向, 定义一次冲刺形如:

- 首先以 q 的概率向右移动一步, 以 $1-q$ 的概率向上移动一步。
- 然后以 $(1-p)p^i$ 的概率向右上移动 $i+1$ 步, $i \geq 0$ 。

有 n 个草莓, 第 i 个草莓位于 (u_i, v_i) , 如果在任意时刻经过了某个草莓, 则视为获得该草莓。问经过充分多次冲刺后期望获得的草莓数量。所有运算在 $\text{mod } P = 1004535809 = 479 \times 2^{21} + 1$ 意义下进行。

$$1 \leq n \leq 5000, 0 \leq u_i, v_i \leq 10^7, |u_i - v_i| \leq 5000, 0 \leq p, q < P, p \neq 1。$$

³笔者命制的集训队互测题目

记 $V = \max(\max_{i=1}^n u_i, \max_{i=1}^n v_i), b = \max_{i=1}^n \{|u_i - v_i|\}$ 。答案的生成函数为

$$\begin{aligned} F_0(x, y) &= \frac{1}{1 - xy(qx + (1 - q)y)^{\frac{1-p}{1-pxy}}} (1 + \frac{qx + (1 - q)y}{1 - pxy}) \\ &= \frac{1 - pxy + qx + (1 - q)y}{1 - pxy - (1 - p)xy(qx + (1 - q)y)} \end{aligned}$$

其中第一行的前者为某次冲刺结束后的位置的生成函数，第一行的后者补上了冲刺过程中的贡献。

为了方便，我们把分子变为 1，将一次查询 (u, v) 拆为四次对 $(u, v), (u - 1, v), (u, v - 1), (u - 1, v - 1)$ 的查询。

$$F(x, y) = \frac{1}{1 - pxy - (1 - p)xy(qx + (1 - q)y)} = \frac{F_0(x, y)}{1 - pxy + qx + (1 - q)y}$$

注意题目限制了 $|u_i - v_i|$ 不会很大，仍考虑提取对角线。令 $G(x, y) = F(\frac{x}{y}, y)$,

$$\begin{aligned} G(x, y) &= \frac{1}{1 - px - (1 - p)x(\frac{qx}{y} + (1 - q)y)} \\ &= \frac{y}{y - pxy - (1 - p)x(qx + (1 - q)y^2)} \\ &= \frac{y}{-(1 - p)(1 - q)xy^2 + (1 - px)y - (1 - p)qx^2} \\ &= \frac{y}{-(1 - p)(1 - q)x(y - \alpha)(y - \beta)} \\ &= \frac{1}{(1 - p)(1 - q)x(\beta - \alpha)} (\frac{\alpha y^{-1}}{1 - \alpha y^{-1}} + \frac{1}{1 - \beta^{-1}y}) \\ \alpha &= \frac{(1 - px) - \sqrt{\Delta}}{2(1 - p)(1 - q)x}, \beta = \frac{(1 - px) + \sqrt{\Delta}}{2(1 - p)(1 - q)x}, \Delta = (1 - px)^2 - 4(1 - p)^2(1 - q)qx^3 \end{aligned}$$

$$b \geq 0, [y^b]G(x, y) = \frac{1}{(1 - p)(1 - q)x(\beta - \alpha)\beta^b} = \frac{1}{\sqrt{\Delta}\beta^b}$$

$$b < 0, [y^b]G(x, y) = \frac{\alpha^{-b}}{(1 - p)(1 - q)x(\beta - \alpha)} = \frac{\alpha^{-b}}{\sqrt{\Delta}}$$

注意 Δ 中 x 最高只有三次项，可以直接 $O(V)$ 递推出 $[x^V] \frac{1}{\sqrt{\Delta}}$ 。

由于 b 很小，考虑维护分子为 $A + B\sqrt{\Delta}$ 的形式，对于 $b \geq 0$ ，有 $\beta^{-1} = \frac{1 - px - \sqrt{\Delta}}{2(1 - p)qx^2}$ ，因此 b 每加 1，新的 $A' = (1 - px)A - B\Delta$ ， $B' = (1 - px)B - A$ ， A, B 的次数都是 $O(b)$ 级别的，因此这一部分时间复杂度 $O(b^2)$ ， $b < 0$ 同理。而分母可以写为 $kx^t \sqrt{\Delta}$ 的形式，所以实际对 $\frac{A}{\sqrt{\Delta}} + B$ 进行查询，那么每次查询只需要遍历一遍 A 即可。总的时间复杂度 $O(V + b^2 + nb)$ 。■

顺带一提，本题数据范围可以改为 $n = 1, b = |u_1 - v_1| \leq 10^5, V = \max(u_1, v_1) \leq 10^9$ 。注意刚才对 A, B 的修改可以写为矩阵形式，因此可以使用矩阵快速幂和 NTT 在 $O(b \log b)$ 的

时间复杂度内求出 A, B 。对于 $\frac{1}{\sqrt{\Delta}}$ ，我们可以先使用整式递推⁴ 在 $O(\sqrt{V} \log V)$ 的时间复杂度内求出第 $V - 3b$ 到 $V - 3b + 2$ 项，然后再递推至 V 附近的点值。即可计算 $[x^u](\frac{A}{\sqrt{\Delta}} + B)$ 。

对于 y 上次数超过 2 的 $G(x, y)$ ，亦有方法可以求出递推式，并进行递推或远处求值，详见 [2] 章节 2.3 与章节 2.4 例题 4。但较难写出对角线的封闭形式，参与其他计算。

3 代数推导中的工具

下面是两个与拉格朗日反演一起使用的例子。

例题 3.1. q 次询问，给定正整数 n, m, k ，满足 $m \geq n + k$ ，计算从二维平面 $(0, 0)$ 走到 (n, m) ，在 (x, y) 每次走到 $(x + 1, y)$ 或者 $(x, y + 1)$ 的所有格路与直线 $y = x + k$ 的交点个数和，对 998244353 取模。 $n, m, q \leq 10^5$ 。

令 $F(x, y) = \frac{1}{1 - x - y}$ ， $G(x, y) = F(x/y, y)$ ， $H_b(x) = [y^b]G(x, y)$

那么 $G(x, y) = \frac{y}{y - x - y^2}$ ，令 $\alpha = \frac{1 - \sqrt{\Delta}}{2}$ ， $\beta = \frac{1 + \sqrt{\Delta}}{2}$ ，其中 $\Delta = 1 - 4x$ 。

同上一题，有

$$\begin{aligned} G(x, y) &= \frac{y}{y - x - y^2} \\ &= \frac{y}{-(y - \alpha)(y - \beta)} \\ &= \frac{1}{\Delta} (\frac{1}{1 - \alpha y^{-1}} - \frac{y\beta^{-1}}{1 - \beta^{-1}y}) \\ b \geq 0, [y^b]G(x, y) &= \frac{1}{\beta^b \sqrt{\Delta}} = \frac{C(x)^b}{\sqrt{1 - 4x}} \end{aligned}$$

其中 $C(x)$ 为卡特兰数的生成函数 $\frac{1 - \sqrt{1 - 4x}}{2x}$ 。

于是答案为 $[x^n]H_k(x)H_{m-n-k}(x) = [x^n]\frac{C(x)^{m-n}}{1 - 4x}$ 。与 k 无关。

注意到 $x C(x) = \alpha$ ，又 $\alpha - \alpha^2 = x$ ，令 $t(u) = u - u^2$ ，于是 $t(\alpha) = x$ ，使用拉格朗日反演：

$$\begin{aligned} [x^n]H(F^{<-1>}(x)) &= \frac{1}{n} [x^{n-1}]H'(x)(\frac{x}{F(x)})^n \\ [x^n]C(x)^m &= [x^{n+m}]\alpha^m = \frac{1}{n+m} [x^{n+m-1}]m x^{m-1} \frac{1}{(1-x)^{n+m}} = \frac{m}{n+m} \binom{2n+m-1}{n} \end{aligned}$$

由于分母为 $1 - 4x$ ，即做一个等比数列求和，答案为

⁴详见 [2] 章节 2.3.2

$$\sum_{i=0}^n 4^{n-i} \frac{m-n}{i+m-n} \binom{2i+m-n-1}{i} \\ = \sum_{i=0}^n 4^{n-i} \left(\binom{2i+m-n-1}{i} - \binom{2i+m-n-1}{i-1} \right)$$

考虑归纳证明 $\sum_{i=0}^t 4^{t-i} \binom{2i+c}{i} = \sum_{i=0}^t (t-i+1) \binom{2t+c+2}{i}$, $t=0$ 时两边均为 1。利用组合恒等式 $\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$, 将一行的组合数往下平移。在杨辉三角上形如

表 1:

行列	0	...	t-3	t-2	t-1	t
2t+c	4t	...	12	8	4	1
2t+c+1	2t+1	...	7	5	3	1
2t+c+2	t+1	...	4	3	2	1

其中第 i 行第 j 列的数表示 $\binom{i}{j}$ 的系数, 例如第 $2t+c$ 行第 $t-3$ 列的系数为 12, $2t+c$ 行的和与 $2t+c+1$ 行, $2t+c+2$ 行相等, 形式化地说:

$$\sum_{i=0}^t 4^{t-i} \binom{2i+c}{i} \\ = \binom{2t+c}{t} + \sum_{i=0}^{t-1} 4(t-i) \binom{2t+c}{i} \\ = \binom{2t+c+1}{t} + \sum_{i=0}^{t-1} (2t-2i+1) \binom{2t+c+1}{i} \\ = \binom{2t+c+2}{t} + \sum_{i=0}^{t-1} (t-i+1) \binom{2t+c+2}{i} \\ = \sum_{i=0}^t (t-i+1) \binom{2t+c+2}{i}$$

于是答案为:

$$\sum_{i=0}^n 4^{n-i} \left(\binom{2i+m-n-1}{i} - \binom{2i+m-n-1}{i-1} \right) \\ = \sum_{i=0}^n (n-i+1) \binom{m+n+1}{i} - \sum_{i=0}^{n-1} (n-i) \binom{m+n+1}{i} \\ = \sum_{i=0}^n \binom{m+n+1}{i}$$

是经典的组合数前缀和问题, 令 $S(n, m) = \sum_{i=0}^m \binom{n}{i}$, 有 $S(n, m) = S(n, m-1) + \binom{n}{m} = 2S(n-1, m) - \binom{n-1}{m}$ 。因此可以使用莫队在 $O(q\sqrt{n+m})$ 的时间复杂度内解决。■

本题可以使用容斥的方法得到相同的结论, 详见 [3] 定理 5.1。

例题 3.2. zeta/圣巢万神殿⁵

给定 n, k, r, s 。求从 $(0, 0)$ 出发, 在 (x, y) 每次走到 $(x+1, y)$ 或者 $(x, y+1)$, 分别到 $(r, n-r)$ 和 $(s, n-s)$, 恰有 k 个交点 (除去起点, 只计算整点) 的简单有序格路对的数量, 答案对 10^9+7 取模。 $1 \leq r \leq s \leq n \leq 10^7$, $1 \leq k \leq n$ 。

设 $M_{r,s}^{n,k}$ 表示答案, 终点处不计入交点, 设 $N_k^{n,r} = M_{r,r}^{n,k}$, $F(x, y) = \sum_{n \geq 1, r} N_0^{n,r} x^r y^{n-r}$, 即起点均为 $(0, 0)$, 终点均为 $(r, n-r) (n \geq 1)$, 中间不相交的有序格路对数的生成函数。令 $u = F(x, xy)$, 则有

$$\frac{1}{1-F} = \text{SEQ}(F) = \sum_{a,b} \binom{a+b}{a}^2 x^a y^b \\ \frac{1}{1-u} = \sum_{a,b} \binom{a}{b}^2 x^a y^b$$

为所有可能的等长有序格路对。欲求解 u 的封闭形式, 考虑函数

$$G(x, y, z, w) = \frac{1}{(1-x-xy)(1-z-zw)}$$

不难发现只需分别对 yw , xz 提取对角线即可。

$$[w^0]G(x, \frac{y}{w}, z, w) = [w^0] \frac{1}{(1-x-\frac{xy}{w})(1-z-zw)} \\ = \frac{1}{(1-x)(1-z)} [w^0] \frac{1}{(1-\frac{xy}{(1-x)w})(1-\frac{zw}{1-z})} \\ = \frac{1}{(1-x)(1-z)} \sum_{i \geq 0} \left(\frac{xy}{1-x} \right)^i \left(\frac{z}{1-z} \right)^i \\ = \frac{1}{(1-x)(1-z)} \frac{1}{1-\frac{xyz}{(1-x)(1-z)}} \\ = \frac{1}{(1-x)(1-z)-xyz}$$

这里我们利用了如下性质: $[x^0] \frac{1}{(1-ax)(1-\frac{b}{x})} = \frac{1}{1-ab}$, 其中 a, b 不含 x 。第二行到第三行的操作是合理的, 因为任取 $[x^i y^j z^k]$ 的系数都是可以计算的。

⁵一道模拟赛题目

$$\begin{aligned}[z^0]G\left(\frac{x}{z}, y, z\right) &= [z^0] \frac{1}{\left(1 - \frac{x}{z}\right)(1 - z) - xy} \\ &= [z^0] \frac{1}{1 + x - \frac{x}{z} - z - xy} \\ &= [z^0] \frac{z}{-z^2 + (1 + x - xy)z - x} \\ &= \frac{1}{\sqrt{\Delta}}\end{aligned}$$

$$\begin{aligned}\Delta &= (1 + x - xy)^2 - 4x = 1 - 2x(y + 1) + x^2(y - 1)^2 \\ u &= 1 - \sqrt{1 - 2x(y + 1) + x^2(y - 1)^2} \\ F &= 1 - \sqrt{1 - 2(x + y) + (x - y)^2}\end{aligned}$$

注意 F 形如一个二次方程的根，令 $F = x + y + 2f$ 。

$$\begin{aligned}f &= \frac{1 - x - y - \sqrt{1 - 2(x + y) + (x - y)^2}}{2} \\ f^2 - (1 - x - y)f + xy &= 0 \\ f &= (x + f)(y + f)\end{aligned}$$

值得一提，这里可以用组合意义解释。第一步两条格路若同向则贡献 $x + y$ ，异向贡献 f ，则 $F = x + y + 2f$ 。 $[x^a y^b]f$ 表示两条格路分别从 $(0, 1)$ ， $(1, 0)$ 出发，到达 (a, b) 且中间不相交的方案数。则下一步相向到达 $(1, 1)$ 贡献 xy ，同向贡献 $(x + y)f$ 。如果继续走到 $(0, 2)$ 和 $(2, 0)$ ，考虑这之后以首次两格路横坐标差为 1 的时刻分为两段，将两段对应的 f 接上初始的两次异向行走，即 f^2 。有 $f = (x + f)(y + f)$ 。

然后对于 $N_{k-1}^{n,r}$ ，可以拆为 k 段 $N_0^{n,r}$ ，即 F^k ，所以 $\sum_{n,r} N_{k-1}^{n,r} x^r y^{n-r} = (x + y + 2f)^k$ 。

按走到 $(r, n - r) (r \leq s)$ 的格路考虑，那么答案

$$\begin{aligned}M_{r,s}^{n,k} &= [x^r y^{n-r}] F^k \left(\sum_{n,r} M_{r,r+1}^{n,0} x^r y^{n-r} \right)^{s-r} \\ &= [x^r y^{n-r}] (x + y + 2f)^k \left(\frac{f}{x} \right)^{s-r} \\ &= [x^r y^{n-r}] \sum_{i=0}^k \binom{k}{i} (x + f)^i (y + f)^{k-i} \left(\frac{f}{x} \right)^{s-r}\end{aligned}$$

考虑以两条格路最后一次横坐标差为 $0, 1, \dots, s - r - 1$ 的时刻分段，那么所有的交点出现在第一段，贡献 F^k ，除第一段外，每段开头一定是异向走的，单独截取出来形如从 $(0, 0)$ 到

$(a - 1, b)$ 和 $(a, b - 1)$ ，我们暂且按 f 考虑，那么计算的是 $[x^a y^b]$ ，而实际上第一条格路的是 $[x^{a-1} y^b]$ ，因此贡献为 $\left(\frac{f}{x}\right)^{s-r}$ 。

令 $f = z(x + f)(y + f)$ ， $G(t) = (x + t)(y + t)$ ， $\phi(t) = (x + t)^a (y + t)^b t^c$ ，有 $z = f(z)/G(f(z))$ ， $f^{<-1>}(z) = z/G(z)$ ，使用拉格朗日反演（注意与上题略有不同）：

$$\begin{aligned}[x^n]H(F^{<-1>}(x)) &= [x^n]H(x)F'(x)\left(\frac{x}{F(x)}\right)^{n+1} \\ [z^n]\phi(f) &= \phi(z) \frac{G(z) - zG'(z)}{G^2(z)} G^{n+1}(z) \\ &= \phi(z)(G(z) - zG'(z))G^{n-1}(z) \\ \phi(f) &= \sum_n z^n [t^n]\phi(t)(G(t) - tG'(t))G^{n-1}(t)\end{aligned}$$

代入 $z = 1$ 和 G, ϕ ：

$$\begin{aligned}& (x + f)^a (y + f)^b f^c \\ &= \sum_n [t^n]\phi(t)(G(t) - tG'(t))G^{n-1}(t) \\ &= \sum_n [t^n](x + t)^{a+n-1} (y + t)^{b+n-1} t^c (xy + tx + ty + t^2 - 2t^2 - xt - yt) \\ &= \sum_n [t^n](x + t)^{a+n-1} (y + t)^{b+n-1} t^c (xy - t^2)\end{aligned}$$

假设取 $[x^l y^m]$ 系数，则满足 $l + m + n = a + n - 1 + b + n - 1 + c + 2 = a + b + c + 2n$ ， $l + m - a - b - c = n$ ，不难使用 a, b, c 计算 l, m 。一组 (a, b, c) 的贡献为： $\binom{a+n-1}{l-1} \binom{b+n-1}{m-1} - \binom{a+n-1}{l} \binom{b+n-1}{m}$ ，那么对 k 个 (a, b, c) 计算一遍即可，时间复杂度 $O(n)$ 。■

总的来说，对于更多问题，提取对角线往往只是推导过程中的工具。需要充分利用对角线是代数幂级数这一性质，配合其他的生成函数技巧进行解题。

致谢

感谢中国计算机学会提供学习和交流的平台。
感谢广东实验中学陈茂彬教练的关心和指导。
感谢家人、朋友对我的支持与鼓励。
感谢赵晟昊、丁相炜同学为本文刊误。

参考文献

- [1] Stanley RP. Enumerative combinatorics, Cambridge Studies in Advanced Mathematics, 1999
- [2] 钟子谦. 两类递推数列的性质和应用, IOI2019 中国国家候选队论文集
- [3] 吴畅. 再谈格路计数, IOI2024 中国国家候选队论文集
- [4] Jack Hsieh. A Technique for Bivariate Main Diagonal Extraction, 2021

浅谈张量及其在 OI 的运用

福建师范大学附属中学 魏忠浩

1 前言

张量作为一种数学工具，其在 20 世纪中后期在物理学、工程学、计算机科学等方面得到广泛运用。特别是在 21 世纪随着深度学习的兴起，成为其核心概念。对于张量的研究也优化了理论计算机科学的一些结果，但在 OI 里传播较少。

本文将引入张量，特别是三阶张量的秩和分解，并讲述若干例子来介绍其在一类分治题目的用处。

本文第 2,3 节介绍一些基本约定和符号定义；第 4 节讲述给出和高维“卷积”问题的联系；第 5 节给出张量分解的困难性和一些特殊张量的性质，第 6 节介绍分解张量的一些算法或方法；第 7 节介绍张量秩的推广——边界秩，并再次给出其在高维“卷积”的运用和对普通张量秩的优化；最后一节介绍其在更广的一类“分治乘法”的运用。

2 约定

默认所有下标从 1 开始。

在表示向量的时候，可以使用正常的列向量形式，也可以使用 $(a_1, a_2, \dots, a_n)$ 表示。

在表示三维数组的时候，一般采用 $[A^{(1)} \mid \dots \mid A^{(n)}]$ 的形式， (i) 指第一层下标，即 $A_{j,k}^{(i)} = A_{i,j,k}$ 。

为了表示方便，一些特殊的三维数组（排列张量等）还可以表示为“运算表”形式： $\begin{pmatrix} f_{1,1} & \cdots & f_{1,m} \\ \vdots & \ddots & \vdots \\ f_{n,1} & \cdots & f_{n,m} \end{pmatrix}$ ，此时三维数组 $X_{i,j,k} = [f_{i,j} + 1 = k]$ ，允许 $f_{i,j}$ 对应位置不存在元素，此时对于任意 k 均为 0。

通常给出的矩阵使用粗体 ($\mathbf{A}$)，张量（特别是三阶张量）使用花体 ($\mathcal{X}$)，向量和变量不做要求。

本文中提出的一些定理较为简单，此时不提供证明；如果涉及到较为困难的定理则会引用原论文。

3 定义

张量的定义有多种。这里可以简单定义：

定义 3.1 (张量). N 阶张量 X 是 N 维数组 $F^{I_1 \times I_2 \times \dots \times I_N}$, $(i_1, i_2, \dots, i_N)$ 对应元素记为 $X_{i_1 i_2 \dots i_N}$ ($i_j \leq I_j$), 或者使用小写字母, $x_{i_1 i_2 \dots i_N}$, $i_1, i_2, \dots, i_N$ 之间也可以使用逗号隔开。

张量的加法和减法可以类比向量, 是将对应的元素相加减。

注意到一阶张量是向量, 二阶张量是矩阵, 在之后视为等价概念。

定义 3.2. 对于 N 阶张量 $\mathcal{B} = \mathcal{A}^{(n)}$, 则 $b_{i_2 i_3 \dots i_N} = a_{ni_2 i_3 \dots i_N}$, 表示其包含的第 n 个 $N-1$ 阶张量。

上述表示也可以用来表达序列的第 i 个元素。并规定 $a_{i_1 i_2 \dots i_N}^{(j)} = (a^{(j)})_{i_1 i_2 \dots i_N}$ 来保证优先级。

定义 3.3 (范数). 张量范数 $\|X\| = \sqrt{\sum_{i=1}^{I_1} \sum_{i_2=1}^{I_2} \dots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \dots i_N}^2}$ 。

定义 3.4 (内积 / Handmard 积). 张量内积 $\langle X, Y \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \dots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \dots i_N} y_{i_1 i_2 \dots i_N}$ 。

定义 3.5 (直和). 记 $\mathcal{Z} = \mathcal{X} \oplus \mathcal{Y}$, 则 $z_{i_1 i_2 \dots i_N} = x_{i_1 i_2 \dots i_N}$ ($j_i \leq I_i$), $z_{j_1 j_2 \dots j_N} = y_{(j_1-I_1)(j_2-I_2) \dots (j_N-I_N)}$ ($j_i > I_i$)。

例如 $\begin{pmatrix} 0 & 1 \end{pmatrix} \oplus \begin{pmatrix} 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 2 \end{pmatrix}$ 。

定义 3.6 (外积 / 直积). 记 $\mathcal{Z} = \mathcal{X} \circ \mathcal{Y}$, 则 $z_{i_1 i_2 \dots i_N j_1 j_2 \dots j_M} = x_{i_1 i_2 \dots i_N} y_{j_1 j_2 \dots j_M}$ 。亦可记作 $\mathcal{Z} = \mathcal{X} \oplus \mathcal{Y}$ 。

定义 3.7. 记 $\mathcal{X}^n = \mathcal{X} \circ \dots \circ \mathcal{X}$, 共计 n 个。

定义 3.8 (Hadamard 积). 记 $\mathbf{Z} = \mathbf{X} * \mathbf{Y}$, 则 $z_{i,j} = x_{i,j} y_{i,j}$ 。

定义 3.9 (秩一张量). 称 $\mathcal{X}$ 为秩一张量, 当且仅当存在长为 N 的向量序列 y , y_i 长度为 I_i , 且 $\mathcal{X} = y^{(1)} y^{(2)} \circ \dots \circ y^{(N)}$, 也就是 $x_{i_1 i_2 \dots i_N} = y_{i_1}^{(1)} y_{i_2}^{(2)} \dots y_{i_N}^{(N)}$ 。

对于三阶张量, 可以简化成 $\mathcal{X} = a \circ b \circ c$ 。

定义 3.10 (CANDECOMP/PARAFAC(CP) 分解). CP 分解是将张量 $\mathcal{X}$ 分解成若干个秩一张量的和。

在三阶张量里, 可以写作 $\mathcal{X} = \sum_{i=1}^R \mathbf{A}^{(i)} \circ \mathbf{B}^{(i)} \circ \mathbf{C}^{(i)}$ 。

定义 3.11 (张量秩). 对于一个张量 $\mathcal{X}$, $\text{rank}(\mathcal{X})$ 表示最小的 R , 使得 $\mathcal{X}$ 能被分解成 R 个一阶张量的和。

在三阶张量中可以直接写作 $\text{rank}(\mathcal{X}) = \min\{R | \exists \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathcal{X} = \sum_{i=1}^R \mathbf{A}^{(i)} \circ \mathbf{B}^{(i)} \circ \mathbf{C}^{(i)}\}$ 。

定义 3.12 (Khatri-Rao 积). 定义 $\mathbf{C} = \mathbf{A} \odot \mathbf{B}$, $\mathbf{A} \in F^{I \times K}$, $\mathbf{B} \in F^{J \times K}$, 则 $\mathbf{C} \in F^{IJ \times K}$, 且 $\mathbf{C}_{i \times (n-1) + j, k} = \mathbf{A}_{i,k} \mathbf{B}_{j,k}$ 。

4 CP 分解在高维“卷积”的联系

该节会先讲述高维“卷积”问题的形式定义, 然后给出一类可能比暴力优的算法思路。

4.1 问题定义

这里给出的高维“卷积”是指这一类问题：

定义 4.1 (高维“卷积”). 给定维数 k . 对于有限集合 $U^{(1)}, U^{(2)}, \dots, U^{(k)}, V^{(1)}, V^{(2)}, \dots, V^{(k)}$, 和 $W^{(1)}, W^{(2)}, \dots, W^{(k)}$, 不妨假设这些集合均可以表示为 $\mathbb{Z} \cap [0, x)$, 且大小为常数。又有 $f^{(i)} : (U^{(i)}, V^{(i)}) \rightarrow W^{(i)}$, 记

$$f((u_1, u_2, \dots, u_k), (v_1, v_2, \dots, v_k)) = (f^{(1)}(u_1, v_1), f^{(2)}(u_2, v_2), \dots, f^{(k)}(u_k, v_k))$$

给定 a_i 和 b_j 的值, 求

$$c_k = \sum_{f(i,j)=k} a_i b_j$$

容易发现, 快速沃尔什变换 (FWT), 快速莫比乌斯变换 (FMT) 等“位运算卷积”均可以规约至此。

其也有 f_i 为多元运算的定义, 容易推导出类似的定义, 不做展开。

通过一些推广可以获得更加广泛的定义：

定义 4.2 (高维双线性计算). 在原定义基础上, 修改 $f^{(i)} : (U^{(i)}, V^{(i)}, W^{(i)}) \rightarrow F$, 记

$$f((u_1, u_2, \dots, u_k), (v_1, v_2, \dots, v_k), (w_1, w_2, \dots, w_k)) = \prod_{i=1}^k f^{(i)}(u^{(i)}, v^{(i)}, w^{(i)})$$

, 给定 a_i 和 b_j 的值, 求

$$c_k = \sum a_i b_j f(i, j, k)$$

此时子集卷积也可以被规约至此。

一般的双线性计算问题的定义是：

定义 4.3 (双线性计算问题¹). 给定 $x_1, \dots, x_N, y_1, \dots, y_M$, 和一个三阶张量 $\mathcal{T} \in F^{N \times M \times K}$, 计算

$$f_k = \sum_{i=1}^N \sum_{j=1}^M t_{i,j,k} x_i y_j$$

注意到“高维”双线性计算本质上是将一般的三阶张量分解成若干个小三阶张量的外积/直积。

这三者的联系比较直观。下一节给出一种通用的分治算法, 解决前两个问题。

下下节给出前两者和三阶张量的联系, 并给出对应的三阶张量 CP 分解问题。

¹这里的命名源自 EntropyInreaser 的博客 [5]

4.2 Karatsuba 算法及推广

Karatsuba 算法是用来计算大数乘法（多项式乘法）的算法，这里给出解决多项式乘法的算法流程：

对于解决 $P(x), Q(x)$ 的乘法，设其最大度数为 n ，令

$$\begin{aligned} P(x) &= p_1(x)x^{\lfloor \frac{n}{2} \rfloor} + p_2(x) \\ Q(x) &= q_1(x)x^{\lfloor \frac{n}{2} \rfloor} + q_2(x) \end{aligned}$$

递归计算

$$\begin{aligned} a_1(x) &= p_1(x)q_1(x) \\ a_2(x) &= p_2(x)q_2(x) \\ a_3(x) &= (p_1(x) + p_2(x))(q_1(x) + q_2(x)) \end{aligned}$$

则 $p_1(x)q_2(x) + p_2(x)q_1(x) = a_3(x) - a_1(x) - a_2(x)$ ，所以能够构造。

复杂度有 $T(n) = 3T(\frac{n}{2}) + O(n)$ ，根据主定理有 $T(n) = O(n^{\log_2 3})$ ，低于暴力。

其关键在于单次过程中减少了乘法次数，因为乘法运算需要递归，减少单层乘法次数就相当于减少复杂度。

对于普通的高维“卷积”问题，暴力算法是需要单层 $|U^{(i)}||V^{(i)}|$ 次乘法，所以最后复杂度仍然是 $\tilde{O}(\prod |U^{(i)}||V^{(i)}|)$ 。

如何减少乘法次数？仿照 Karatsuba 算法设计：

对于第 i 层，设 $n = |U^{(i)}|, m = |V^{(i)}|, k = |W^{(i)}|$ ，同时给出参数 $r, \mathbf{A}, \mathbf{B}, \mathbf{C}$ 。

这里该算法需要解决需要计算 $c_k = \sum_{f(i,j)=k} a_i b_j$ ，其中维数为 i 。

允许该算法调用维数为 $i-1$ 的乘法（删除 $U^{(i)}, V^{(i)}$ 和 $W^{(i)}$ 后的高维“卷积”问题）。

这里规定 $a^{(j)}(x_1, x_2, \dots, x_{i-1}) = a(x_1, x_2, \dots, x_{i-1}, j)$ ，也就是钦定最高位为 j 后的序列。

对于 $1 \leq i \leq r$ ，计算 $x_i = \sum_{j=1}^n \mathbf{A}_{i,j} a^{(j)}$ 和 $y_i = \sum_{j=1}^m \mathbf{B}_{i,j} b^{(j)}$ 。

随后计算 $z_i = x_i y_i$ ，此部分需要使用 r 次乘法。

最后计算 $c^{(k)} = \sum_{i=1}^r \mathbf{C}_{i,k} z_i$ 。

不妨假设 $r \geq \max\{|U^{(i)}|, |V^{(i)}|, |W^{(i)}|\}$ ，否则可以发现 $W^{(i)}$ 中至多有 r 个不同的数字， $U^{(i)}$ 和 $V^{(i)}$ 则是可以分别压缩成至多 r 个变量。在最后的复杂度里加上平凡的 $O(\prod |U^{(i)}| + \prod |V^{(i)}| + \prod |W^{(i)}|)$ 即可。

可以发现复杂度变为 $\tilde{O}(\prod r(i))$ ，其中 $r(i)$ 是对应层的 r 的大小。并且容易构造和暴力等价的算法。具体复杂度分析需要参考分治树。

此算法和一般的快速沃尔什变换差距较大。如果仿照快速沃尔什变换，则应该先对两个原数组做变换，然后相乘，再做逆变换。中间结果存储 x, y, z 对应的结果。但空间开销较大，而上述算法可以保证空间和输入输出量呈线性关系。

4.3 和 CP 分解的联系

现在关键是如何构造出对应的 r 及 $\mathbf{A}, \mathbf{B}, \mathbf{C}$ 。

将 z_i 展开得到：

$$c^{(k)} = \sum_{i=1}^r \mathbf{C}_{i,k} \left(\sum_{i=1}^n \mathbf{A}_{i,j} a^{(j)} \right) \left(\sum_{i=1}^m \mathbf{B}_{i,j} b^{(j)} \right)$$

可以发现最后 $c^{(k)}$ 是若干 $a^{(i)} b^{(j)}$ 的线性组合，这里只需要要求算法的结果满足本层的“卷积”。固定 i, j, k ，考察 $a^{(i)} b^{(j)}$ 对 $c^{(k)}$ 贡献的系数：

$$c^{(k)} \leftarrow \sum_{i=1}^r \mathbf{C}_{i,k} \mathbf{A}_{i,i} \mathbf{B}_{i,j} a^{(i)} b^{(j)}$$

于是可以构造张量 $\mathcal{X}$ ，使得 $x_{ijk} = [f(i, j) = k]$ ，则

$$\mathcal{X} = \sum_{i=1}^r \mathbf{A}^{(i)} \circ \mathbf{B}^{(i)} \circ \mathbf{C}^{(i)}$$

这是一个 CP 分解的形式。也就是说，求一组 $r, \mathbf{A}, \mathbf{B}, \mathbf{C}$ ，就是求一组 CP 分解；要求最小的 r 就是求该张量的秩。

对于高维双线性计算问题，直接令 $x_{ijk} = f(i, j, k)$ ，剩下的推导是类似的。

对于多元运算 f_i ，则是对高阶（大于三阶）张量做分解。

在先前的集训队论文里 [15]，其给出的形式较为特殊，但降低这类算法的复杂度的关键仍然在于 r ，并提供了一些例题，使用 Karatsuba 算法的方式讲述如何构造出特定算法。下一节将从张量角度分析这些问题，并给出一些基础或者特殊的性质。

在实际的题目里，通常情况 $U^{(i)}, V^{(i)}$ 和 $W^{(i)}$ 是分别相同的，此时有较为简单的复杂度分析结论：

定理 4.1. 假设加法乘法需要常数时间。设 $U^{(i)}, V^{(i)}$ 和 $W^{(i)}$ 分别相同，记 $n = |U^{(i)}|, m = |V^{(i)}|, k = |W^{(i)}|$ ，单层对应的张量为 $\mathcal{X}$ 。设维数为 K ，则一般情况下复杂度为 $O(\max\{\text{rank}(\mathcal{X}), n, m, k\}^K)$ ；当 $\text{rank}(\mathcal{X}) = \max\{n, m, k\}$ 时复杂度为 $O(\text{rank}(\mathcal{X})^K K)$ 。注意这里只是上界，一些情况可以更优。

设输入总量为 n ， $c = \frac{\log \text{rank}(\mathcal{X})}{\log \max\{n, m, k\}}$ 。

则当 $c \neq 1$ 时，复杂度可以做到 $O(n^{\max\{c, 1\}})$ 。否则复杂度为 $O(n \log_{\max\{n, m, k\}} n)$ 。

特别的，如果所有 U, V, W 相同，则复杂度为 $O(n^c)$ ，当 $c = 1$ 时复杂度为 $O(n \log_c n)$ 。

5 张量秩的性质

5.1 张量秩的基础性质

张量秩的性质较为怪异。例如在复数域 ($F = \mathbb{C}$) 和实数域 ($F = \mathbb{R}$) 上，同一个三阶张量的张量秩不一样 [10]。而二阶张量 (矩阵) 则一样。

同时，目前张量分解 (求张量秩) 问题在有限域 F 是 NPC，在一些常见的无限域 $F = \mathbb{R}, \mathbb{C}, \mathbb{Q}$ 上是 NP hard [6]，所以不太可能存在求出精确解的优秀做法，这也说明上述问题在最小化 r 的部分遇到了非常大的困难。

但其至少有一些比较基础的性质：

定理 5.1. 给定 $X = F^{I_1 \times I_2 \times \dots \times I_N}$ ，满足 $I_i = n$ ，给定排列 $\pi \in S_N$ 。定义 $\mathcal{Y}$ 满足 $y_{i_1 i_2 \dots i_n} = x_{i_{\pi(1)} i_{\pi(2)} \dots i_{\pi(N)}}$ ，则 $\text{rank}(\mathcal{Y}) = \text{rank}(X)$ 。

这里说明了张量维度的轮换性质，意味着两种输入和一种输出本质等价。

定理 5.2. $\text{rank}(X \oplus \mathcal{Y}) \leq \text{rank}(X) + \text{rank}(\mathcal{Y})$

其中如果取等，变成 $\text{rank}(X \oplus \mathcal{Y}) = \text{rank}(X) + \text{rank}(\mathcal{Y})$ ，是 Strassen's additivity conjecture。难以找到反例。目前有一个非构造性的反例证明：

定理 5.3 ([12][14])。存在 $N \geq 450$ 和 $p, p' \in \mathbb{C}^{N \times N \times N}$ ， $\text{rank}(p \oplus p') < \text{rank}(p') + \text{rank}(p'')$ 。

同时还有关于一些特殊情况的证明：

定理 5.4 ([12][7])。设 p 可以分解成 A_1, A_2, A_3 ， p' 分解成 B_1, B_2, B_3 ，若存在 $\dim(A_i) = 2$ 或者 $\dim(B_i) = 2$ 则猜想成立。

在大部分实际应用中，可以假设该猜想为真。

定理 5.5. $\text{rank}(X \circ \mathcal{Y}) \leq \text{rank}(X) \text{rank}(\mathcal{Y})$

相比上面的直和，直积的严格小于比较容易取，例如考虑子集卷积转化出来的张量：

$$\begin{pmatrix} 0 & 1 \\ 1 & \end{pmatrix} = (1, 0) \circ (1, 0) + (1, 0) \circ (0, 1) + (0, 1) \circ (1, 0)$$

而

$$\begin{pmatrix} 0 & 1 \\ 1 & \end{pmatrix} \circ \begin{pmatrix} 0 & 1 \\ 1 & \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & & 3 & \\ 2 & 3 & & \\ 3 & & & \end{pmatrix}$$

可以被证明 $\text{rank} \leq 8$ ：

$$\begin{aligned} & (0, 0, 0, 1) \circ (1, 0, 0, 0) \circ (0, 0, 0, 1) + \\ & (1, 0, 0, 0) \circ (0, 0, 0, 1) \circ (0, 0, 0, 1) + \\ & (1, 0, 1, 0) \circ (1, 0, 1, 0) \circ (0, 0, 1, 0) + \\ & (0, 1, 1, 0) \circ (0, 1, 1, 0) \circ (0, 0, 0, 1) + \\ & (1, 1, 0, 0) \circ (1, 1, 0, 0) \circ (0, 1, 0, 0) + \\ & (0, 0, 1, 0) \circ (0, 0, 1, 0) \circ (0, 0, -1, -1) + \\ & (0, 1, 0, 0) \circ (0, 1, 0, 0) \circ (0, -1, 0, -1) + \\ & (1, 0, 0, 0) \circ (1, 0, 0, 0) \circ (1, -1, -1, 0) \end{aligned}$$

结合复杂度分析，可以发现在一些情况下展开会减小复杂度。事实上在这里如果展开的层数足够多，可以做到 $\tilde{O}((2 + o(1))^n)$ 但并不实用。

5.2 典型秩和最大秩

对于任意 $F^{n \times m \times k}$ 的三阶张量，有最大秩、典型秩、通用秩这些概念。

最大秩是指给定张量的大小 (n, m, k) 和域 F ，可能出现的最大的秩：

定义 5.1 (典型秩 (Typical Rank))。若一个非零测度的张量集合上秩均为 x ，则 x 是典型秩。

一个简单的理解是出现概率非 0。

定义 5.2 (通用秩 (Generic Rank))。若几乎所有张量的秩为 x ，则称其为通用秩。

对此的相关研究也很多，但对于任意的三阶张量，最大秩只有一个很弱的上界 $\min\{nm, mk, nk\}$ 。

除此之外对于一些特殊张量有研究：

1 给出了一些特殊三阶张量的典型秩。2 给出了固定大小的张量的典型秩。

[8] 中的一些表格也提供了结果。

对于最大秩，则有 $3 \times 3 \times 3$ 三阶张量在实数域下的最大秩为 5 [10, 4]，在复数域也是一样的 [4]。

除此之外，对于 $I \times J \times 2$ 的张量也有最大秩 $\min\{I, J\} + \min\{I, J, \max\{I, J\}/2\}$ [10]。

下文也将介绍特定张量 $X = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$ 的一种分解方式 ($0 \leq a_{ij} < 3$)，此方式可以

证明 $\text{rank}(X) \leq 4$ ；若允许 $a_{i,j}$ 为空，对于随机出来的张量进行分解，几乎满足 $r \leq 5$ ，且大部分满足 $r \leq 4$ 。

Tensor Size	Typical Rank	Citation
$2 \times 2 \times 2$	$\{2, 3\}$	[120]
$3 \times 3 \times 2$	$\{3, 4\}$	[119, 173]
$5 \times 3 \times 3$	$\{5, 6\}$	[175]
$I \times J \times 2$ with $I \geq 2J$ (very tall)	$2J$	[177]
$I \times J \times 2$ with $J < I < 2J$ (tall)	I	[177]
$I \times I \times 2$ (compact)	$\{I, I + 1\}$	[173, 177]
$I \times J \times K$ with $I \geq JK$ (very tall)	JK	[174]
$I \times J \times K$ with $JK - J < I < JK$ (tall)	I	[174]
$I \times J \times K$ with $I = JK - J$ (compact)	$\{I, I + 1\}$	[174]

Table 3. Typical rank over $\mathbb{R}$ for three-way tensors.

图 1: 实数张量的典型秩 [8]

Table 1

Typical ranks for 2-slice, 3-slice, and 4-slice unconstrained arrays. Values reported in bold correspond to smallest typical ranks computed numerically; values in plain font were known before. Values separated by commas are known typical ranks. In the complex field, the smallest value in a cell is generic.

N_1	N_3 N_2	2	3	4	5	3	4	5	4	5
2		2,3	3	4	4	3,4	4	5	4,5	5
3		3	3,4	4	5	5	5	5,6	6	6
4		4	4	4,5	5	5	6	6	7	8
5		4	5	5	5,6	5,6	6	8	8	9
6		4	6	6	6	6	7	8	8	10
7		4	6	7	7	7	7	9	9	10
8		4	6	8	8	8	8,9	9	10	11
9		4	6	8	9	9	9	9	10	12
10		4	6	8	10	9	10	10	10	12
11		4	6	8	10	9	11	11	11	13
12		4	6	8	10	9	12	12	12,13	13

图 2: 实数张量的典型秩，若在复数域内，最小值为通用秩 [3]

5.3 CP 分解唯一性

三阶张量的分解在一定的条件下是唯一的，忽略一些简单的等价关系，例如置换等。

定义 5.3. 称两分解 $\sum_{i=1}^n \mathbf{v}_i^{(1)} \circ \mathbf{v}_i^{(2)} \circ \cdots \circ \mathbf{v}_i^{(k)}$ 和 $\sum_{i=1}^n \mathbf{w}_i^{(1)} \circ \mathbf{w}_i^{(2)} \circ \cdots \circ \mathbf{w}_i^{(k)}$ 等价当且仅当 $\{\mathbf{v}_i^{(1)} \circ \mathbf{v}_i^{(2)} \circ \cdots \circ \mathbf{v}_i^{(k)}\} = \{\mathbf{w}_i^{(1)} \circ \mathbf{w}_i^{(2)} \circ \cdots \circ \mathbf{w}_i^{(k)}\}$ 。

定义 5.4. 对于矩阵 A ，定义 $k(A)$ 为最大的 t ，使得任选 t 个 A 的列，都线性无关（须存在选择方案）。

定理 5.6 (Kruskal' Theorem[10, 9]). 对于一个三阶张量 $X = \sum_{i=1}^n \mathbf{a}_i \circ \mathbf{b}_i \circ \cdots \circ \mathbf{c}_i$ ，若满足 $k(A) + k(B) + k(C) \geq 2R - 2$ 则所有分解等价，即分解唯一。

但此条件较为苛刻。尽管如此，在一些情况下配合下一节的一种启发式近似算法有很好的效果。

6 CP 分解算法

6.1 暴力枚举和优化

对于有限域可以暴力枚举 $\mathbf{A}, \mathbf{B}, \mathbf{C}$ ，但复杂度较高。

一个发现是枚举 $\mathbf{A}, \mathbf{B}$ ，则 $x_{ijk} = \sum_{t=1}^r \mathbf{A}_t^{(i)} \mathbf{B}_t^{(j)} \mathbf{C}_t^{(k)}$ ，可以发现这符合高斯消元的形式。这样可以省去枚举 $\mathbf{C}$ 的复杂度。

同时，这种方法也可以用来人工构造：枚举 $\mathbf{A}^{(i)} \circ \mathbf{B}^{(i)}$ （秩一矩阵），对于每个 k ，判断 $\mathbf{D}_{i,j} = x_{ijk}$ 是否可以被线性组合出。

例题 6.1 (GYM102119A Tritwise Mex²). 定义两个数的 $\text{mex}_3(a, b)$ 为两个数的在三进制下每一位的 mex 所组成的数。求 $c_k = \sum_{\text{mex}_3(i,j)=k} a_i b_j$ 。保证 $0 \leq i, j < 3^n$ 且 $n \leq 12$ 。

考虑对应的张量 $\begin{pmatrix} 1 & 2 & 1 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{pmatrix}$ 。

可以发现右下角的 0 构成矩形，先放一个矩阵 $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$ 。

接下来可以使用四个角减去右下角刻画出 1: $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ 。

²题目链接: <https://codeforces.com/problemset/gymProblem/102129/A>

接下来既然 0, 1 均被刻画, 使用全 1 矩阵减去即可: $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ 。

此时构造出来 4 个秩一矩阵, 使用高斯消元或者根据上述观察得出 $\mathbf{C}$, 做到 $r = 4$ 的 CP 分解。

根据主定理和 4.3 节, 该题存在对应的算法, 时间复杂度为 $O(4^n)$ 。

可以发现相比直接采用初始设计来构造, 此方法会更为形象。

例题 6.2 (欧伊昔³)。给定任意的 3×3 运算表 f 。定义两个数的 $f_3(a, b)$ 为两个数的在三进制下每一位的 f 所组成的数。求 $c_k = \sum_{f_3(i, j)=k} a_i b_j$ 。保证 $0 \leq i, j < 3^n, 0 \leq a_i \leq 10$ 且 $n \leq 12$ 。

原题保证数据随机, 这里不保证。

对于任意的三阶张量, 可以使用下文所述的交替最小二乘法求近似解, 此启发式算法在这里可以做到 $r \leq 4$ 且效率优秀, 但这里介绍另外几种构造算法。

一种简单的做法是考虑先放一个全 1 矩阵, 然后任取 c 减去 $f(i, j) \neq c$ 的位置, 这样可以做到 $r \leq 7$ 。

如果存在出现次数小于 3 的数, 可以 $r = 6$; 否则如果存在 $f(i, j) = f(i, k)$ 或者 $f(i, j) = f(k, j)$, 可以缩成一个矩阵, 则可以做到 $r \leq 6$ 。否则是拉丁方, 在三进制下一定可以转化至三进制异或的求解, 可以直接得到 $r = 3$ 的 CP 分解。如果要求实数可以做到 $r \leq 4$ 。经过此优化可以得到 $r \leq 6$ 的结果。

接下来可以通过代码或者较复杂的证明, 对于任意一个不含空元素的运算表, 存在一个拉丁方使得至多两个元素不同。于是可以 $r \leq 5$ 。

同时发现直到目前的所有构造, 除了拉丁方, 其余的 $\mathbf{A}, \mathbf{B}$ 元素均为 $-1, 0, 1$ 。

通过使用暴力枚举加高斯消元, 可以穷举所有这样的 $a \circ b$ 发现 $r \leq 4$, 正常实现可以跑进 300ms, 能够满足一般题目的要求。所以复杂度可以做到 $O(4^n)$ 且不依赖数据随机和无精度问题 (因为 $\mathbf{C}_{i,j}$ 是分母较小的分数)。

6.2 一种启发式算法: 交替最小二乘法

这里需要一些线性代数的知识, 并假设解决实数域下的分解问题:

定义 6.1 (范数)。 $n \times m$ 的矩阵 $\mathbf{A}$ 的范数 $\|\mathbf{A}\| = \sum_{i=1}^n \sum_{j=1}^m |\mathbf{A}_{i,j}|^2$ 。向量视作 $n \times 1$ 的矩阵。

定义 6.2 (最小二乘法和伪逆)。 $x = \mathbf{M}^\dagger = (\mathbf{M}^T \mathbf{M})^{-1} \mathbf{M}^T$ 是 $\|\mathbf{M}x - y\|$ 达到最小值的一个解, x 称作 $\mathbf{M}$ 的伪逆。

³加强自笔者的集训队互测题目, 题目名称不变

特别地, 如果矩阵 $\mathbf{M}$ 不满秩, 需要对其进行 SVD 分解求出伪逆。

先考虑分解三阶张量。

为了符合 Khatri-Rao 积的要求, 在这里考虑所有分解矩阵的转置, 认为分解三阶张量 $\mathcal{X} \in F^{I \times J \times K}$ 时, $\mathbf{A} \in F^{I \times R}, \mathbf{B} \in F^{J \times R}, \mathbf{C} \in F^{K \times R}$ 。

根据上节的发现推出, 如果固定 $\mathbf{B}, \mathbf{C}$ 且存在解, 那么使用高斯消元法一定存在对应的 $\mathbf{A}$ 。

如果不存在, 那么可以尝试将 $\mathbf{A}$ 修改至最小值。

在这里需要最小化 $\|\mathbf{A}(\mathbf{B} \odot \mathbf{C})^T - \mathcal{X}_{I \times JK}\| = \|(\mathbf{B} \odot \mathbf{C})\mathbf{A}^T - \mathcal{X}_{JK \times I}\|$ 。

这里 $JK \times I$ 是指将 $\mathcal{X}$ 的 JK 二维拍成一维, 并按照顺序重新排列下标。

根据最小二乘法, 结果可以写作

$$\mathbf{A}'^T = (\mathbf{B} \odot \mathbf{C})^\dagger \mathcal{X}_{JK \times I}$$

同理, 轮换操作:

$$\mathbf{B}'^T = (\mathbf{C} \odot \mathbf{A})^\dagger \mathcal{X}_{KI \times J}$$

$$\mathbf{C}'^T = (\mathbf{A} \odot \mathbf{B})^\dagger \mathcal{X}_{IJ \times K}$$

交替进行下去即可。

为了方便和优化效率, 其可以转化成 $\mathbf{A}' = \mathcal{X}_{(1)}(\mathbf{C} \odot \mathbf{B})[\mathbf{C}^T \mathbf{C} * \mathbf{B}^T \mathbf{B}]^\dagger$, 这里 $\mathcal{X}_{(i)}$ 指保留第 i 维作为第一维, 其余拍平成为第二维, 这里对于伪逆的转化并使用了一些性质 [11, 8]。

之后可以轮换地修改 $\mathbf{A}, \mathbf{B}, \mathbf{C}$, 就是交替最小二乘法的主要流程。

对于分解高阶张量, 则固定除了 $\mathbf{A}^{(i)}$ 之外的其余矩阵, 修改 $\mathbf{A}^{(i)}$, 轮换进行。具体算法参考3。

虽然此算法没有比较严谨的正确性保证, 但实践中基本上都能获得较好的结果。一般能够分解的情况称为收敛 (随着次数增多向 0 贴近)

6.3 其他分解方法与实际应用

像梯度下降法等算法也可以运用在张量分解上。这里不讲述。

梯度下降法的一个优势是, 计算过程不需要考虑矩阵求逆 (伪逆), 实现起来会更加方便, 劣势则是一些情况不会像交替最小二乘法那样稳定。

虽然这些方法不是求精确解, 也无法保证正确性, 但在绝大部分测试中 (只发现极个别反例) 表现非常优秀。

根据 CP 分解在一些情况的唯一性, 在一些情况下这些算法能够为求出精确解提供非常大的帮助, 但这种情形较为严苛。

所以一般 CP 分解近似算法主要用来给出张量秩的估计。和提供一些提示。

```

procedure CP-ALS( $\mathbf{X}, R$ )
  initialize  $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$  for  $n = 1, \dots, N$ 
  repeat
    for  $n = 1, \dots, N$  do
       $\mathbf{V} \leftarrow \mathbf{A}^{(1)\top} \mathbf{A}^{(1)} * \dots * \mathbf{A}^{(n-1)\top} \mathbf{A}^{(n-1)} * \mathbf{A}^{(n+1)\top} \mathbf{A}^{(n+1)} * \dots * \mathbf{A}^{(N)\top} \mathbf{A}^{(N)}$ 
       $\mathbf{A}^{(n)} \leftarrow \mathbf{X}^{(n)} (\mathbf{A}^{(N)} \odot \dots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \dots \odot \mathbf{A}^{(1)}) \mathbf{V}^\dagger$ 
      normalize columns of  $\mathbf{A}^{(n)}$  (storing norms as  $\lambda$ )
    end for
  until fit ceases to improve or maximum iterations exhausted
  return  $\lambda, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)}$ 
end procedure

```

Figure 8. Alternating least squares algorithm to compute a CP decomposition with R components for an N th order tensor $\mathbf{X}$ of size $I_1 \times I_2 \times \dots \times I_N$.

图 3: 算法伪代码流程。在此算法里额外维护了 λ 向量来吸收 $\mathbf{A}^{(i)}$ 的范数 [8]

然后对于简单的张量，可以使用暴力枚举 0, 1, -1 元素的方式尝试，或者通过已有的经验手动尝试。

对于复杂的张量，可以选择直接使用近似算法的结果，但不使用高精度浮点数的时候容易导致精度误差。或者由于要求的结果精度过高跑得较慢。

例子 6.1 (三进制异或卷积). 分解张量 $\begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{pmatrix}$ 。

将其进行 CP 分解迭代 2×10^5 轮，当 $r = 4$ 时快速收敛，误差 10^{-12} 一下；当 $r = 3$ 时不收敛。

暴力枚举可能的秩一矩阵可以得到一组合法的秩一矩阵：

$$\mathbf{D}_1 = (1, 1, 1) \circ (1, 1, 1), \mathbf{D}_2 = (0, 1, -1) \circ (0, 1, -1), \mathbf{D}_3 = (1, 0, -1) \circ (1, 0, -1), \mathbf{D}_4 = (1, -1, 0) \circ (1, -1, 0)$$

$$\text{比如 } \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \frac{1}{3}(\mathbf{D}_1 - 2\mathbf{D}_2 + \mathbf{D}_3 + \mathbf{D}_4)$$

这是在实数域上（和一些 $\mathbb{Z}_n$ ）下“最优”的分解，如果到了复数域，使用暴力 DFT 的范德蒙德矩阵即可。

例题 6.3 (Jellyfish[15]⁴). 每个点是一个的向量，每维下标为范围在 $[0, d)$ 的整数， $d \in$

⁴2018 集训队第一轮作业的一道题目“海蜇？海蜇！”。

$\{2, 4\}$ ，保证对应的十进制数不超过 250000。现在随机从某个关键点出发，随机走 K 步至相邻向量。求期望经过的不同的关键点的个数。

转化到求距离为 k 的点对数。由于 $d = 2$ 是二进制异或。考虑 $d = 4$ 。

定义 $f(i, j) = |i - j|$ ，则转化至高维“卷积”问题，转化到求三阶张量的 CP 分解 $\begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{pmatrix}$ 。

对其使用上文所述的交替最小二乘法迭代大约 50000 次，在 $r = 6$ 时误差范数小于 10^{-10} ，在 $r = 5$ 时不收敛。

同时，根据 CP 分解的结果，可以发现存在 $\begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix}$ 和全一矩阵。

加加减减后发现，只需要分别划分 $\begin{pmatrix} 1 & 3 \\ 1 & 1 \\ 3 & 1 \end{pmatrix}$ 和 $\begin{pmatrix} 0 & 2 \\ 0 & 2 \\ 2 & 0 \\ 2 & 0 \end{pmatrix}$ 即可。

可以使用 $(1, 0, 0, 1) \circ (1, 0, 0, 1)$ 和 $(1, 0, 0, -1)$ 和 $(1, 0, 0, -1)$ 划分四个角的 0 和 3，中间的两个 0 直接使用两个秩一张量刻画。

从另一个角度看，这个运算表和 $\begin{pmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{pmatrix}$ （将 3 换成 1）差了两个元素，而此时可

以直接利用 $f(i, j) = i - j$ 对应的张量分解，也能给出 $r \leq 6$ 的结论（但将域提升至复数域）。此方法还可以适用于高维情况，可以得到 $r \leq n + \sum_{i=1}^n \min\{i, n - i\} - [2n]_2^{\frac{n}{2}}$ 。

例题 6.4.⁵ 给定 $f^{(i)}(x, y) = x^y \bmod 10$ ，同时记 $0^0 = 0$ ，对其进行高维“卷积”。维数 $k \leq 7$ 。

等价于分解张量：

⁵改编自集训队论文 [15] 里的一道例题

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1_7 & 1_7 & 1_7 & 1_7 & 1_7 & 1_7 & 1_7 & 1_7 & 1_7 \\ 1 & 2_1 & 4_5 & 8_1 & 6_6 & 2_1 & 4_5 & 8_1 & 6_6 & 2_1 \\ 1 & 3_2 & 9_3 & 7_2 & 1_4 & 3_2 & 9_3 & 7_2 & 1_4 & 3_2 \\ 1 & 4_8 & 6_8 & 4_8 & 6_8 & 4_8 & 6_8 & 4_8 & 6_8 & 4_8 \\ 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 6_8 & 6_8 & 6_8 & 6_8 & 6_8 & 6_8 & 6_8 & 6_8 & 6_8 \\ 1 & 7_2 & 9_3 & 3_2 & 1_4 & 7_2 & 9_3 & 3_2 & 1_4 & 7_2 \\ 1 & 8_1 & 4_5 & 2_1 & 6_6 & 8_1 & 4_5 & 2_1 & 6_6 & 8_1 \\ 1 & 9_7 & 1_7 & 9_7 & 1_7 & 9_7 & 1_7 & 9_7 & 1_7 & 9_7 \end{pmatrix}$$

其中除去第一行的全部 1 被一个矩阵表示外，剩下每一行未被标记角标的元素自成一个矩阵。此时花费 3 个矩阵。

对于角标为 1 和 2 的元素，排列方式形如黑白棋盘，所以可以使用二进制异或的矩阵形式刻画，分别需要 2 个。

对于角标为 3,4,5,6 的元素各需要 1 个。

对于角标为 7,8 的元素可以使用大减小的形式完成，各需要 2 个。

总计 15 个，复杂度 $O(n^{\ln 15 \ln 10}) = O(n^{1.176})$ 。

通过使用交替最小二乘法（梯度下降法此时不太稳定），发现在 $r = 11$ 时误差范数不能低于 1 并几乎停止减小，但 $r = 12$ 却能在 5×10^5 次迭代内达到 10^{-3} 。

这意味着上述做法并非“最优”，可以发现我们实际上是能用全一矩阵换掉最大的刻画元素的矩阵开销。于是有新的方案：

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2_1 & 4_5 & 8_1 & 6_6 & 2_1 & 4_5 & 8_1 & 6_6 & 2_1 \\ 1 & 3_2 & 9_3 & 7_2 & 1 & 3_2 & 9_3 & 7_2 & 1 & 3_2 \\ 1 & 4_8 & 6_8 & 4_8 & 6_8 & 4_8 & 6_8 & 4_8 & 6_8 & 4_8 \\ 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 6_8 & 6_8 & 6_8 & 6_8 & 6_8 & 6_8 & 6_8 & 6_8 & 6_8 \\ 1 & 7_2 & 9_3 & 3_2 & 1 & 7_2 & 9_3 & 3_2 & 1 & 7_2 \\ 1 & 8_1 & 4_5 & 2_1 & 6_6 & 8_1 & 4_5 & 2_1 & 6_6 & 8_1 \\ 1 & 9 & 1 & 9 & 1 & 9 & 1 & 9 & 1 & 9 \end{pmatrix}$$

其中先分别使用 1 个矩阵表示 0, 5 和 9。

接着分别使用 2 个矩阵表示角标为 1 和角标为 2 的元素。

然后直接分别使用 1 个矩阵刻画角标为 3,5,6 的元素。

之后使用 2 个矩阵刻画角标为 8 的元素。

最后加上全一矩阵，其减去上述所有矩阵就能表示 1。

总共花费 13 个矩阵，此时复杂度为 $O(n^{\frac{\ln 13}{\ln 10}}) = O(n^{1.114})$ 。

若要改进至 12 个，可以考虑套用上述的容斥方式至 6：

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2_1 & 4_5 & 8_1 & 6 & 2_1 & 4_5 & 8_1 & 6 & 2_1 \\ 1 & 3_2 & 9_3 & 7_2 & 1 & 3_2 & 9_3 & 7_2 & 1 & 3_2 \\ 1 & 4_8 & 6 & 4_8 & 6 & 4_8 & 6 & 4_8 & 6 & 4_8 \\ 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 1 & 7_2 & 9_3 & 3_2 & 1 & 7_2 & 9_3 & 3_2 & 1 & 7_2 \\ 1 & 8_1 & 4_5 & 2_1 & 6 & 8_1 & 4_5 & 2_1 & 6 & 8_1 \\ 1 & 9 & 1 & 9 & 1 & 9 & 1 & 9 & 1 & 9 \end{pmatrix}$$

先分别使用 1 个矩阵表示 0,5 和 9。

接着使用全一矩阵。并使用 $(0,0,1,0,1,0,1,0,1,0) \circ (0,1,1,1,1,1,1,1,1,1)$ ，此是包含 6 的“极小”秩一矩阵。

然后分别使用 2 个矩阵表示角标为 1 和 2 的元素。

之后分别使用 1 个矩阵表示角标 3,5,8 的元素。

此时 6 可以被那个“极小”秩一矩阵，减去表示角标 1,5,8 的元素矩阵得出，1 同理。

这时候数一数， $1 \times 3 + 1 + 1 + 2 \times 2 + 1 \times 3 = 12$ ，做到了估计的下界。复杂度为 $O(n^{\frac{\ln 12}{\ln 10}}) = O(n^{1.079})$ 。

当 n 取 10^7 的时候， $n^{1.176} \approx 170608238 \approx 1.7 \times 10^8$ ， $n^{1.114} \approx 62805835 \approx 6.3 \times 10^7$ ， $n^{1.079} \approx 35727283 \approx 3.6 \times 10^7$ ，虽然总共优化至原先的 $\frac{1}{6}$ ，但后两者的优化效果略少，所以如果达不到最优的估计，有些时候也可以适当放宽。

7 边界秩

在对子集卷积对应的张量进行分解时，会发现 $r = 3$ ；但考虑自身的外积，也就是 4×4 的二进制子集卷积，就会出现奇怪的情况：当 $r = 4$ 的时候持续下降，说明有可能存在 $r = 4$ 的构造，但实际情况仍然是不太可能存在精确构造。

观察近似算法的结果，分析其分解的形式，并套用至 2×2 的二进制子集卷积得：

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \approx (1, \epsilon) \circ (1, \epsilon) \circ (0, \frac{1}{\epsilon}) + (1, 0) \circ (1, 0) \circ (1, -1)$$

当 ϵ 趋于 0 时，结果趋近于子集卷积的对应张量。但并不能直接做到 $O(2^n n)$ ，需要使用高精度小数防止精度问题，此时复杂度为 $O(2^n n^2)$ ，需要 2^n 次高精度乘法和 $2^n n$ 次加法及低精度乘法。

所以有如下结论：

定理 7.1. 张量秩均为 R 的张量序列，其极限的张量至可以不是 R 。

对于这一类张量，允许小量的存在会使其的张量分解变得更小。因此引入边界秩的概念。

边界秩的一种定义是存在张量秩均为 R 的张量序列，极限为该张量 [8]。但没有较为直接的优秀性质。这里给出另一种定义：

定义 7.1. 记 $\text{rank}_h(\mathcal{X}) = \{R | \exists \mathbf{A}, \mathbf{B}, \mathbf{C}, \sum_{i=1}^R \mathbf{A}^{(i)} \mathbf{B}^{(i)} \mathbf{C}^{(i)} = x^h \mathcal{X} + o(x^h)\}$ ，这里 $\mathbf{A}, \mathbf{B}, \mathbf{C}$ 的元素是 h 次多项式。

引理 7.1. $\text{rank}_h(\mathcal{X})$ 单调不增，因此在 $h \rightarrow +\infty$ 收敛。

定义 7.2 (边界秩). 记 $\underline{\text{rank}}(\mathcal{X}) = \lim_{h \rightarrow +\infty} \text{rank}_h(\mathcal{X})$ 。

由于 rank_h 的结果是整数，存在 H 使得 $\underline{\text{rank}}(\mathcal{X}) = \text{rank}_H(\mathcal{X})$ ，通常 H 较小。

虽然边界秩在二阶张量（矩阵）下和张量秩是一致的，但在高阶张量下不一致（例如上面的子集卷积）。

容易发现其有一些张量秩的性质：

定理 7.2. 定义 $\mathcal{Y}$ 满足 $y_{i_1 i_2 \dots i_n} = x_{i_{n(1)} i_{n(2)} \dots i_{n(N)}}$ ，则 $\text{rank}_h(\mathcal{Y}) = \text{rank}_h(\mathcal{X})$ 。

定理 7.3. $\text{rank}_h(\mathcal{X} \oplus \mathcal{Y}) \leq \text{rank}_h(\mathcal{X}) + \text{rank}_h(\mathcal{Y})$

相比张量秩，边界秩的此条件容易简单粗暴的构造出一组不满足等式的例子，考虑下列形式：

$$\begin{pmatrix} 1 & 2 & 3 & & & \\ 4 & 5 & 6 & & & \\ 7 & 8 & 9 & & & \\ & & & 10 & & \\ & & & & 10 & \\ & & & & & 10 & \\ & & & & & & 10 \end{pmatrix}$$

考虑 $(1, 0, 0, x, 0, 0, 0, 0)^2$ 和类似的形式，将左上角的 $\begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$ 分别和下面的 10 对应。

将四个秩一矩阵相加可得：（根据选择顺序获得的一种情况，其余同理）

$$\begin{pmatrix} 1 & 1 & & x & x & & \\ & 1 & & & & x & x \\ & & 0 & & & & \\ x & & & x^2 & & & \\ & x & & & x^2 & & \\ x & & & & & x^2 & \\ & x & & & & & x^2 \end{pmatrix}$$

可以考虑利用剩余的 5 个元素，将对应位置加成 1 的同时减去多余的 x 。随后使用 $(1, 1, 1, 0, 0, 0, 0)^2$ 减去前面的一。

花费了 10 个秩一矩阵，因此 $\text{rank}_2 \leq 10$ ，比分开的 $3 \times 3 + 4 = 13$ 小。

定理 7.4. $\text{rank}_{h+l}(\mathcal{X} \circ \mathcal{Y}) \leq \text{rank}_h(\mathcal{X}) \text{rank}_l(\mathcal{Y})$

同时边界秩可以转化至普通张量秩：

定理 7.5. $\text{rank}(\mathcal{X}) \leq \binom{h+2}{2} \text{rank}_h(\mathcal{X})$ 。

证明考虑假设 $\mathbf{A} = \sum_{i=1}^h ([x^i] \mathbf{A}) x^i$ ， $\mathbf{B}, \mathbf{C}$ 同理。

则 $x^h \mathcal{X} = \sum_{i+j+k=h} \sum_{t=1}^R ([x^i] \mathbf{A}^{(t)}) ([x^j] \mathbf{B}^{(t)}) ([x^k] \mathbf{C}^{(t)})$ 。

这部分花费 $\binom{h+2}{2} \times R$ 个秩一张量。

值得注意的是， $\text{rank}(\mathcal{X}^k) \leq \binom{hk+2}{2} \text{rank}_{hk}(\mathcal{X}^k) \leq \binom{hk+2}{2} \text{rank}_h(\mathcal{X})^k$ ，所以取极限，和先前的复杂度分析，可以做到 $O(n^{w+\epsilon})$ ，其中 $w = \frac{\log \text{rank}_h(\mathcal{X})}{\log \max\{n, m, k\}}$ 。也就是说可以“当作”普通张量秩使用。

但注意到这里要求使用 $\mathcal{X}^k$ 作为分治对应的张量，隐含常数过于庞大，因此不实用。

尽管如此，仍然可以像子集卷积那样，假设维数为 k ，只需要维护 hk 次多项式即可。复杂度从维数角度看变为 $\tilde{O}((\prod r(i))(\sum h(i)))$ ，相比正常的张量秩只多了一个 $\sum h(i) = O(k)$ 。

此时维护的多项式在子集卷积中一般叫做“占位多项式”，也可以先对于占位多项式的 x^i 系数集体做变换，然后卷积，然后逆变换。

例题 7.1. 对于 B 进制数（ B 为常数），定义 i and j 为 i, j 按位最小值，给定 $a_i, b_i, i < B^n$ ，求：

$$c_k = \sum_{i+j=k, i \text{ and } j=0} a_i b_j$$

考虑 $B = 3$ 。

对应张量为 $\begin{pmatrix} 0 & 1 & 2 \\ 1 & & \\ 2 & & \end{pmatrix}$ 。

注意到左上角的 $\begin{pmatrix} 0 & 1 \\ 1 & \end{pmatrix}$ 可以用 2 个秩一矩阵表示，分别为 $(1, 0) \circ (1, 0), (1, \epsilon) \circ (1, \epsilon)$ 。

同理，选出子矩阵 $\begin{pmatrix} 0 & 2 \\ 2 & \end{pmatrix}$ ，也能用 2 个秩一矩阵表示。注意到其中 $(1,0,0) \circ (1,0,0)$ 出现了两次，所以总共使用 3 个，复杂度可以做到 $O(3^n n^2)$ 。

对于 $B \neq 3$ 的情况，可以发现 $(1,0,\dots,0)^2$ 每一处均出现一次，所以复杂度仍然是 $O(B^n n^2)$ 。若 B 不是常数，需要计入复杂度，由于这题的张量较为特殊，只需要多乘上一个 B 即可。

笔者猜想：对于能够用运算表形式表示的张量，通常边界秩比秩小的情况满足该张量在运算表形式下存在空元素。

使用通常修饰，是因为存在一个“反例”是 $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{pmatrix}$ ，目前没有比较显然的 $r = 4$ 构造（复数域下存在），但将其分成两份二进制子集卷积就能构造出 $\text{rank}(\mathcal{X}) = 4$ 。

8 一类“分治乘法”

张量在优化分治复杂度上的方法不仅能够适用于高维“卷积”和双线性计算问题，对于一些特定的分治算法也能优化。

在之前的高维“卷积”中，能利用的张量只有 $\mathcal{X}^k$ ，但在这类特殊的“分治乘法”里，张量的大小选取可以更加多样。

例题 8.1 (卷积). 求 $c_k = \sum_{i+j=k} a_i b_j$ 。

在假设精度足够 ($O(\log n + \log V)$ 大小)，已有的结论是：一个长为 n 循环卷积对应张量存在大小为 n 的 CP 分解，也就是暴力 DFT。

于是对于此题，设长为 l ，可以利用 $2l+1$ 的循环卷积构造出来 CP 分解。根据主定理，复杂度为 $O(n^{\frac{\log(2l+1)}{\log l} l^2})$ 。

所以当 n 充分大的时候可以做到 $O(n^{1+o(1)})$ 。

例题 8.2 (矩阵乘法). 求 $C_{i,j} = \sum_{k=1}^m A_{i,k} \times B_{k,j}$ 。

矩阵乘法在学术界研究深入，对于其复杂度的改进一种方案是给出更加优秀的张量秩（边界秩）的上界。

设对于 $N \times M \times K$ 的矩阵乘法，分治成 $\frac{N}{n} \times \frac{M}{m} \times \frac{K}{k}$ 的矩阵乘法解决，不妨假设结果为整数否则补全。

记其对应张量为 $\langle n, m, k \rangle$ ， $R = \text{rank}(\langle n, m, k \rangle)$ ，由于下标置换不影响张量秩，将其做三遍获得 $\frac{N}{nmk} \times \frac{M}{nmk} \times \frac{K}{nmk}$ 。使用主定理得到复杂度 $O(n^w)$ ，其中 $w = 3 \frac{\log R}{\log(nmk)}$ 。

则单层可以视作块状矩阵的 $n \times m \times k$ 矩阵乘法，根据先前的分析，接下来要给出对应的张量秩或者边界秩的上界。

对于 Strassen 算法，在 $\langle 2, 2, 2 \rangle$ 的时候给出了 7 的张量秩上界。（注意张量总元素个数为 $(nmk)^2$ ）

对于边界秩的运用，可以证明 $\langle 2, 2, 3 \rangle$ 的时候边界秩上界为 5[2]，进而根据前文的定理做到更优复杂度。

同时注意到由于边界秩在直和情况下不等情况较为常见（前面的反例实为 $3 \times 1 \times 3$ 向量外积和 $1 \times 4 \times 1$ 向量内积），复杂度可能更加优秀。

定义 8.1. 定义 $n \odot \mathcal{X} = \mathcal{X} \oplus \mathcal{X} \cdots \oplus \mathcal{X}$ ，重复 n 次。

引理 8.1. $\text{rank}(a \odot \mathcal{X}) \leq a \text{rank}(\mathcal{X})$

引理 8.2. 若 $\text{rank}(a \odot \mathcal{X}) = r$ ，则 $\text{rank}(a \odot \mathcal{X}^k) \leq (\frac{r}{a})^k a$ 。

证明考虑数学归纳法，其中第一步是运用类似“分配律”的形式，不难手动证明：

$$\text{rank}(a \odot \mathcal{X}^{k+1}) = \text{rank}((a \odot \mathcal{X}^k) \otimes \mathcal{X}) \leq (\frac{b}{a})^k \text{rank}(a \odot \mathcal{X}) = (\frac{b}{a})^k b = (\frac{b}{a})^{k+1} a$$

定理 8.1 (Schönhage's τ theorem[13]). 若有 $\mathcal{X} = \text{rank}(\oplus \langle n_i, m_i, k_i \rangle) \leq r$ ，且 $\sum_{i=1}^p (n_i m_i k_i)^\tau = r$ ，则矩阵乘法最优指数 $\omega \leq 3\tau$ 。

给出一种简要证明：考虑 $\mathcal{X}^s = \oplus_{s_i=s} \binom{s}{s_1, s_2, \dots, s_p} \langle \prod n_i^{s_i}, \prod m_i^{s_i}, \prod k_i^{s_i} \rangle$ 。

对于 $\sum_{i=1}^p (n_i m_i k_i)^\tau = r$ 同样 s 次方，得到 $\sum_{s_i=s} \binom{s}{s_1, s_2, \dots, s_p} (\prod n_i^{s_i}, \prod m_i^{s_i}, \prod k_i^{s_i})^\tau = \sum_{s_i=s} f(s) = r^s$ ，取求和中最大的项，记此时的序列为 t_i 。

取 h 使得对于所有张量 $\text{rank}_h = \text{rank}$ 。求和的个数为组合数，关于 h, s 是多项式（度数为 p ），所以 $\text{rank}(\mathcal{X}^k) \leq r^s \text{poly}(h, s) \leq f(t) \text{poly}(h, s)$ 。

记 $N = \prod n_i^{s_i}, M = \prod m_i^{s_i}, K = \prod k_i^{s_i}, a = \binom{s}{s_1, s_2, \dots, s_p}$ ，则 $\text{rank}(a \odot \langle N, M, K \rangle) \leq (NMK)^\tau \text{poly}(h, s)$ 。

根据引理 8.2，再次放大，带入复杂度公式内，发现 poly 和 a 项在取极限时全部消失，因此 $\omega \leq 3\tau$ 。

根据之前边界秩直和的反例，容易构造充分大的 $\langle n, 1, m \rangle \oplus \langle 1, (n-1)(m-1), 1 \rangle$ ，其边界秩有上界 $nm+1$ ，取 $n=4, m=3$ 就能得到 $12^\tau + 6^\tau = 13$ ，进而 $\omega \leq 2.57$ 。

更为优秀的构造可以得到 2.52。[13]。

在这之后也有更优的算法改进，受篇幅所限不介绍 [1, Lecture 21,22][5]。

但是上述算法基本无法实际运用，其中一个原因是边界秩的出现使得可以使用非常大的张量。

上述算法也启发可能可以通过类似的方法解决高维“卷积”或者双线性计算问题，这时候可以通过一些构造使得能够应用上直和。但尚未找到相应的例子。

例题 8.3 (jellyfish⁶)。

此题目虽然几乎难以对 4×4 的运算表进行优化，但是可以考虑 16×16 的运算表，此时的结果不是简单的两个 4×4 的外积（由于题目最后只需要求距离和），可以轻松使用近似算法（例如梯度下降法）获得 $r < 36$ 的答案。

⁶同例题 6.3

假设 $d = 3$ 也能获得一样的结果，对于 3×3 的运算表存在 $R = 4$ 的构造；但考虑 9×9 , $r = 15$ 在 3×10^4 的迭代内就达到 10^{-10} 的误差。说明能够做到更低的复杂度。但并没有继续深入研究，这是因为这题精度要求较低（只需要对数，答案值域不会很大），继续迭代可以选取一个足够精度的分解套用。

9 总结

本文介绍了张量、张量秩、边界秩，并给出了一些实践较优秀的分解算法和性质，以及与一些OI分治算法和题目的关联。还介绍了张量分解的困难性，从而直接说明这类算法优化复杂度的困难程度。本文给出的性质较为稀少且部分比较平凡，但这些性质在优化复杂度中仍然能发挥重要的作用。希望本文能够起到抛砖引玉的作用，也希望有兴趣的读者继续研究和寻找更多有用的性质。

10 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢父母对我的培养和教育。
感谢学校对我的栽培。
感谢福建师范大学附属中学的教练对我的指导。
感谢黄佳旭同学和我讨论交流。
特别感谢任舍予学长和林圣涵学长给予我的启发。

参考文献

- [1] Mit 6.890: Matrix multiplication and graph algorithms – fall 2021. [EB/OL].
- [2] Dario Bini, Milvio Capovani, Francesco Romani, and Grazia Lotti. $o(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication. Information Processing Letters, 8(5):234–235, 1979.
- [3] Paul Breiding, Sarah Eggleston, and Andrea Rosana. Typical ranks of random order-three tensors, 2024.
- [4] Murray R. Bremner and Jiaxiong Hu. On kruskal’s theorem that every $3 \times 3 \times 3$ array has rank at most 5. Linear Algebra and its Applications, 439(2):401–421, 2013.
- [5] EntropyInreater. 矩阵乘法: 从 strassen 到 coppersmith-winograd. [EB/OL], [2022-10-08].

- [6] Christopher J. Hillar and Lek-Heng Lim. Most tensor problems are np-hard. ArXiv, abs/0911.1393, 2009.
- [7] Joseph Ja’ Ja’ and Jean Takche. On the validity of the direct sum conjecture. SIAM Journal on Computing, 15(4):1004–1020, 1986.
- [8] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. SIAM Review, 51(3):455–500, 2009.
- [9] Joseph B. Kruskal. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. Linear Algebra and its Applications, 18(2):95–138, 1977.
- [10] Joseph B. Kruskal. Rank, decomposition, and uniqueness for 3-way and n-way arrays. 1989.
- [11] Charles F. Van Loan. The ubiquitous kronecker product. Journal of Computational and Applied Mathematics, 123(1):85–100, 2000. Numerical Analysis 2000. Vol. III: Linear Algebra.
- [12] Filip Rupniewski. Tensor rank of the direct sum of two copies of 2×2 matrix multiplication tensor is 14. 2022.
- [13] A. Schönhage. Partial and total matrix multiplication. SIAM Journal on Computing, 10(3):434–455, 1981.
- [14] Yaroslav Shitov. Counterexamples to strassen’s direct sum conjecture. Acta Mathematica, 2017.
- [15] 梁晏成. 《jellyfish》命题报告及拓展探究. 2018 集训队论文集.

浅谈一类线性同余不等式问题

长沙市长郡中学 杨鑫和

摘要

本文归纳了在信息学竞赛中出现过的一类线性同余不等式问题，着重探讨了如何处理单个线性同余不等式，以及研究线性同余不等式组解的结构的方法。

1 前言

信息学竞赛中，涉及到同余理论的部分经常的是考察一次的同余等式，也就是所谓的线性同余方程。而对于不限定同余式的某个具体取值，而是限制线性同余式的结果属于某个集合内的“线性同余不等式”考察极少，在OI中是一个新鲜的话题。

本文将通过先考察线性同余方程的特殊情况，从而推广到不等式的一般情况的方法研究线性同余不等式。

第四节对于求解单个线性同余不等式，着重研究了求出其最小解的方法，并且介绍了该方法在OI中的一些应用。

第五节对于线性同余不等式组问题，探讨了如何利用质数指数折半思想更快地求解其解的出现频率，并把该方法推广到了求解其解的最小周期上。

第六节对于第五节算法复杂度中难以分析的部分进行了更加细致的讨论。

2 一些约定

以下默认各位读者对于解决线性同余方程的工具扩展欧几里德算法、中国剩余定理有基本的了解。

若无特殊声明，认为之后所有出现的小写字母变量均定义在正整数集 $\mathbb{Z}^+$ 上。特别地，对于小写字母 p ，若未特殊声明，认为 p 一定是质数。

对于 $a \in \mathbb{Z}, b \in \mathbb{Z}^+$ ，记 $a \bmod b = r$ 当且仅当 $\exists k \in \mathbb{Z}, a = kb + r$ ，且 $0 \leq r < b$ ，容易验证合法的 r 存在且唯一。

记 $a|b$ 表示 a 整除 b ，即 $\exists k \in \mathbb{Z}^+, b = ka$ 。

记 $a \equiv b \pmod m$ 表示 a 与 b 在模 m 意义下同余，即 $m|(a - b)$ 。

记 $\mathbb{Z}_m$ 表示集合 $\{0, 1, 2, \dots, m-1\}$ 。

记 $\gcd(a, b)$ 表示 a, b 的最大公因数；记 $\text{lcm}(a, b)$ 表示 a, b 的最小公倍数。

记 $v_a(b)$ 表示最大的自然数 m ，满足 $a^m \leq b$ 。

3 线性同余不等式

定义 3.1 (线性同余不等式). 给定模数 m 、系数 $a \in \mathbb{Z}_m$ ，以及集合 $S \subseteq \mathbb{Z}_m$ ，对于某个未知的变量 $x \in \mathbb{Z}_m$ ，如果限定 $ax \bmod m \in S$ ，则称这个限制是一个线性同余不等式。

后文也会将这种形式的线性同余不等式简记为 $ax \equiv S \pmod m$ 。

特别地，诸如 $L \leq (ax + b) \bmod m \leq R$ 、 $ax \not\equiv b$ 等其它形式的限制均可整理成关于 x 的上述形式，我们以下同样将其称为线性同余不等式。

定义 3.2 (线性同余不等式的解). 称一个集合 $S' \subseteq \mathbb{Z}_m$ 为该线性同余不等式的解，当且仅当 $\forall x' \in \mathbb{Z}_m, x' \in S' \iff ax' \bmod m \in S$ 。依据此定义，一个线性同余不等式的解是存在且唯一的。

求解单个线性同余不等式，可以直接遍历 $\mathbb{Z}_m$ 中的元素检查其是否满足条件做到复杂度 $O(m)$ 。在 S 较小的时候，也可以遍历 S 中的元素利用扩展欧几里德算法求解单个线性同余方程，做到 $O(|S| \log_2 m)$ 。

通过上面的例子可以看到，线性同余不等式问题乃至后面的线性同余不等式组问题的整个解集 S' ，其大小往往与朴素算法的时间复杂度渐进意义上几乎等同，求出线性同余不等式（组）问题的完全解通常意义不大。因此后文中所有所谓的探讨同余不等式问题，主要是研究其解的某种结构特征，如求出解集中的最小值、解集大小等其它信息。

例题 3.1 (A Modular Arithmetic Challenge¹). 给定 a, m, L, R 满足 $a \in \mathbb{Z}_m, 0 \leq L \leq R < m$ 。对于一类特殊的线性同余不等式 $L \leq ax \bmod m \leq R$ ，求在 $\mathbb{Z}_m$ 中 x 的最小解。

这个问题相当于将经典的线性同余方程问题扩展到了区间上，我们需要同时解决一个区间的同余方程求解问题，整体性地维护解的最小值信息。

求解普通的线性同余方程的方式是扩展欧几里德算法，该算法的一个理解方式即注意到 $ax \equiv b \pmod m$ 等价于 $\exists y \in \mathbb{Z}, ax + my = b$ ，然后运用类似欧几里德算法的方式翻转 x, y 的地位，缩减规模，递归求解。

仿照上述算法的思想， $L \leq ax \bmod m \leq R$ 等价于存在某个自然数 y ，满足 $L \leq ax - my \leq R$ 。我们将所有符合 $y = \lfloor \frac{ax}{m} \rfloor$ 的整点 (x, y) 画在坐标系上，称一个坐标点的权值为 $ax - my$ ，将所有权值在 $[L, R]$ 间的点标成红色。题目所求的就是横坐标最小的红色点。

¹来源：<http://poj.org/problem?id=3530>

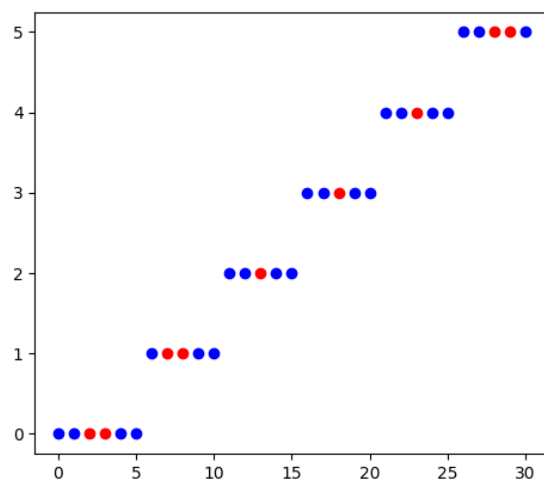


图 1: $a=6, m=31, L=11, R=19$ 的情形, 满足 $R-L+1 \geq m$

可以发现也容易说明以下事实: 点列从左向右排成若干个横坐标不交、纵坐标递增的行, 每一行符合限制的红色点恰为一个区间。那么我们其实只需要找到纵坐标最小的存在红点的行就找到了答案。

我们首先解决平凡的情况, 纵坐标为零的行对应的权值依次是: $0, a, 2a, \dots, \lfloor \frac{m}{a} \rfloor a$ 。判断第一行是否具有红色点只需要判断 $\lceil \frac{L}{a} \rceil a \leq R$ 是否成立即可。

接下来, 若第一行不存在红点, 一定有 $R-L+1 < a$ 。而注意到同一行的相邻点的权值恰好差 a 。也就是说若第一行不存在红点, 一定有每一行的红点个数不超过一。如何判定一行是否存在红点呢? 对于纵坐标为 y 的行, 这等价于要求存在一个 x 满足 $L \leq ax - my \leq R$ 。如果合法的 x 存在, 那么合法的 x 一定等于 $\frac{L+my+(-L-my) \bmod a}{a}$ 。将这个 x 带回原式整理, 得到存在合法的 x 等价于 $(-L-my) \bmod a \leq R-L$ 。可以发现这个问题的形式仍然是要求 $my \bmod a$ 的值在某个区间内, 求最小的 y 。递归求解出最小的 y , 反过来代入求出最小的 x 即可。

时间复杂度同欧几里德算法, 为 $O(\log_2 m)$ 。

例题 3.2 (A Modular Arithmetic Challenge 扩展). 求例题 3.1 中的线性同余不等式在 $\mathbb{Z}$ 中前 k 小的解。

例题 3.1 中的分析方式, 启发我们将 $R-L+1$ 的大小与 m 进行比较。

如果 $R-L+1 \geq m$, 那么每一行必定会有至少一个红点。暴力枚举前若干行的纵坐标 y , 利用 $ax - my \in [L, R]$ 解出红色点的横坐标区间, 如果红色点的总个数达到了 k 个就退出。

否则如果 $R-L+1 < m$, 我们知道每一行至多只有一个红点, 此时这些红点按横坐标排序之后, 纵坐标也是严格上升的。仿照求最小解的方式翻转坐标系递归下去, 每一层需要 $O(k)$ 扫描所有解还原横坐标, 时间复杂度为 $O(k \log_2 m)$ 。

例题 3.3 ($\pm AB^2$). 你拥有一个变量 x , 初始时 $x = V$ 。给定两个正整数 A, B 和数轴长度 M , 满足 $1 \leq A < B \leq M, \gcd(A, B) = 1$ 。你可以做任意次操作, 每次操作从如下四种操作中选择一种:

- 将 x 的值修改为 $x + A$ 。
- 将 x 的值修改为 $x - A$ 。
- 将 x 的值修改为 $x + B$ 。
- 将 x 的值修改为 $x - B$ 。

在操作过程中, 你需要时刻保证 $0 \leq x \leq M$ 。问最终的 x 有多少种可能的取值?

由于 $\gcd(A, B) = 1$, 根据裴蜀定理, 任意整数 X 都能被表示成 $s, t \in \mathbb{Z}, X = sA + tB + V$ 的形式。若 $M \geq A + B - 1$, 我们有结论: $0 \sim M$ 中所有数都可以被题述过程构造出来。

证明可以考虑, 若 s, t 正负性相同 ($st \geq 0$), 直接操作不会经过任何在 $[\min(X, V), \max(X, V)]$ 外的点; 若 s, t 正负性不同 ($st < 0$), 不妨设 $s < 0, t > 0$, 考虑先不断执行 $+B$ 操作直至不可执行。此时你所在的位置距离右边界 M 一定 $< B$, 距离左边界 0 的距离 $\geq M - B + 1 \geq A$, 一定可以先执行 $-A$ 操作再继续不断 $+B$ 。

若 $M < A + B - 1$, 我们可以先通过 $-A$ 操作将 x 调整至 $< A$ 的位置, 考虑从这个位置出发能经过的所有点。枚举下一次操作是 $+A$ 还是 $+B$, 后续所有操作都只剩下不超过一种选择。以 $+A$ 为例, 下一次操作不能是 $-A$, 因为已经访问过了其终点; 不能是 $+B$, 这样一定会超出右边界; 而对于 $+A$ 操作和 $-B$ 操作不可能同时能够操作, 否则数轴长度 M 超过了 $A + B - 1$ 。

将每一个点向它下一次操作的终点连一条无向边, 我们断言该操作不会形成环。若存在一个简单环, 提取这个环上的所有操作累加, 一定有 $s'A + t'B = 0$ 且 $s', t' > 0$ 。由于 $\gcd(A, B) = 1$, 而由 $B|(-s'A), A|(-t'B)$, 得到 $B|s', A|t'$, $|s| + |t| \geq A + B > m$ 与简单环的假设矛盾。

由上面的讨论我们知道, 能访问的所有点的形态形如两条不交链, 一条只包含 $+A, -B$ 操作, 另一条只包含 $+B, -A$ 操作。下文以 $+A, -B$ 这条链为例, 其从起始点开始、到第一次访问到 $[M - A + 1, B - 1]$ 中的某个点时结束。

假设这条链一共进行了 t 次 $+A$ 操作, 那么就一定进行了 $\lfloor \frac{V+tA}{B} \rfloor$ 次 $-B$ 操作, 当前位置就是 $(V + tA) \bmod B$ 。我们相当于要求出使得 $(V + tA) \bmod B$ 在 $[M - A + 1, B - 1]$ 区间内的最小的 t , 这就是例题 3.1 中的同余不等式问题。运用例题 3.1 中的方法, 时间复杂度同样为 $O(\log_2 M)$ 。

²来源: https://atcoder.jp/contests/arc127/tasks/arc127_f

4 线性同余不等式组

定义 4.1 (线性同余不等式组). 对于单个未知的变量 $x \in \mathbb{Z}$, 有 k 条模数不一定相同线性同余不等式的限制:

$$\begin{cases} a_1 x \equiv S_1 \pmod{m_1} \\ a_2 x \equiv S_2 \pmod{m_2} \\ \vdots \\ a_k x \equiv S_k \pmod{m_k} \end{cases}$$

则称这是一个线性同余不等式组。

由于可以通过解出单个线性同余不等式来将所有 x 前的系数化为 1, 所以之后讨论的线性同余不等式组通常可以认为 $a_1 = a_2 = \dots = a_k = 1$ 。

定义 4.2 (线性同余不等式组的解). 称一个模数 m' 和一个集合 $S' \subseteq \mathbb{Z}_{m'}$ 为该线性同余方程组的解, 当且仅当对于任意一个方程 t , 都有 $\forall x' \in S', a_t x' \bmod m_t \in S_t$; 且 $\forall x' \in \mathbb{Z}_{m'}/S'$ 都至少存在一个方程 t 满足 $a_t x' \bmod m_t \notin S_t$ 。

另一个关于线性同余不等式组的解的直观理解是, 将每一条线性同余不等式看作以一定周期不断重复的无限 01 序列, 将所有的这样的 01 序列按位取与, 得到的无限 01 序列的每一种周期都对应一组解。

4.1 解的频率问题

线性同余不等式组的解一定存在, 因为当 $m' = \text{lcm}_{i=1}^k m_i$ 至少存在一组解。根据无穷 01 串周期的性质我们可以得知: 所有可能的解的模数 m' 都是最小的可能的解的模数 $m'_{\min}$ 的倍数, 且对于所有的解, 合法解的出现频率 $\frac{|S'|}{m'}$ 一定相同。下面的例题就展示了如何刻画线性同余不等式组解的出现频率。

例题 4.1 (Traffic Blights³). 一条马路上有 n 个红绿灯, 第 i 个红绿灯距离道路左端 a_i 个单位。从第 0 秒开始, 红绿灯将会先持续 r 秒红灯, 再持续 g 秒绿灯, 然后继续持续 r 秒红灯……以此循环。一辆车将从前 2019! 秒中随机一个实数时刻作为出发时刻, 从这个时刻开始以一个单位每秒的速度匀速向右行驶。问这辆车有多少概率经过每一个红绿灯时都亮绿灯。 $n \leq 500, r + g \leq 100$ 。

原问题的出发时刻是均匀随机分布的实数, 称第 x 秒到第 $x + 1$ 秒之间的时间区间为时间段 x , 注意到如果车在同一个时间段出发, 遇到的红绿灯状态相同。问题实际上可以被转化为关于出发时间段 x 的一个线性同余不等式组。

³来源: ICPC 2019 World Finals K 题

由于 $r + g \leq 100$, 所以该同余不等式组的所有模数 $m_i \leq 100$ 。合并所有模数相同的方程之后, 有 $(\prod m_i) | 2019!$ 。因此我们只要求出任意一个解, 该解的 $\frac{|S'|}{m'}$ 就是题目要求的答案。

本题的形式虽然保证了同余不等式中所有的 S_i 在模意义下是一段区间, 但是本题的做法并不需要用到任何特殊性质, 我们可以将本题泛化成下述问题:

给定一个满足 $\max_{i=1}^k m_i \leq B$ 的线性同余不等式组, 求当解的模数 $m' = \text{lcm}_{i=1}^k m_i$ 时, 解集的大小 $|S'|$ 。

考虑一些特殊性质情况下的想法:

如果 m_i 满足性质两两互质, 由中国剩余定理可知从每一个模数对应的集合 S_i 中任选一个, 组成线性同余方程, 其必定有且唯一解。因此解集大小 $|S'| = \prod_{i=1}^k |S_i|$ 。

如果两条线性同余不等式的模数之间有倍数关系, 那么你可以通过将模数较小的同余不等式合并到大的同余不等式上去, 从而去除模数之间的倍数关系。

我们可以思考如何转化问题, 才可以将原问题转化到特殊性质的情况上去。原问题的朴素做法是枚举 x 在模 $\text{lcm}_{i=1}^k m_i$ 意义下的所有结果检验其是否满足同余方程。考虑假设不枚举模 $\text{lcm}_{i=1}^k m_i$ 的结果, 转而枚举 x 模一个更小的模数 T 之后的结果, 考察我们新得到了什么信息。

若我们枚举了 $x \equiv r \pmod{T}$, 那么有 $\exists y \in \mathbb{Z}, x = yT + r$, 对于同余不等式 $x \bmod m \in S$, 我们做代换 $(yT + r) \bmod m \in S$, 利用一次同余方程的分析方法化简, 可以得到一个关于 y 的同余不等式 $y \bmod \frac{m}{\gcd(m, T)} \in S'$ 。也就是说, 如果合理选取 T 的取值, 可以简化同余方程的形式从而可以应用特殊性质时的做法。

称满足 $p^2 > B$ 的质数为大质数, 否则为小质数。取 T 为 $\prod_{p^2 \leq B} p^{v_p(B)}$, 此时所有的 $\frac{m}{\gcd(m, T)}$ 要么为 1, 要么为一个大质数。将模数为 1 的方程特殊处理之后, 套用模数两两互质的做法即解决了枚举模 T 结果之后的问题。由于在 $\text{mod } \frac{\text{lcm}_{i=1}^k m_i}{T}$ 意义下 y 的每一种解唯一对应着一个 $\text{mod } \text{lcm}_{i=1}^k m_i$ 意义下 x 的解, 将各个关于 y 的问题结果相加就得到了原问题的答案, 时间复杂度为 $O(B^2 T)$, 在 $B = 100$ 时, T 取 $2^6 \times 3^4 \times 5^2 \times 7^2 = 6350400$ 并不能接受。

考虑更精细的 T 的取法。如果取 T 为 $\prod_{p \leq B} p^{\lfloor \frac{v_p(B)}{2} \rfloor}$, 则所有的 $\frac{m}{\gcd(m, T)}$ 要么为 1, 要么为某一个质数的若干次方。证明该事实可以考虑反证, 单个模数 m 的质因子分解 $\prod_i p_i^{\alpha_i}$, 如果存在两个质数的指数 $\alpha_i > \lfloor \frac{v_{p_i}(B)}{2} \rfloor$, 那么一定有对应的 $p_i^{\alpha_i} > \sqrt{B}$, 与 $m \leq B$ 矛盾。

特殊处理模数为 1 的方程后, 将所有模数为质数若干次幂的方程合并到对应的最大指数上, 然后就转化成了模数两两互质特殊性质的情况。当 $B = 100$ 时, $T = 2^3 \times 3^2 \times 5 \times 7 = 2520$ 可以接受。

综上所述, 我们在 $O(B^2 \prod_{p \leq B} p^{\lfloor \frac{v_p(B)}{2} \rfloor})$ 的时间复杂度内解决了泛化后的同余不等式组解的频率问题。

4.2 最小周期问题

笔者在针对例题 4.1 的做法进行仔细研究之后, 发现线性同余不等式组除了研究解的个数以外还有其它一些值得发掘的问题。

例题 4.2. (最小周期问题) 给定一个满足 $\max_{i=1}^k m_i \leq B$ 的线性同余不等式组，求最小的解的模数 $m'_{\min}$ 。

对于一个同余不等式 $x \equiv S \pmod{m}$ ，如果存在 $n|m$ 以及 $T \subseteq \mathbb{Z}_n$ 使得 $\forall x \in \mathbb{Z}_m, (x \bmod n) \in T \iff x \in S$ ，则 $x \equiv T \pmod{n}$ 和原同余不等式完全等价。从所有可能的 n 中选取最小的一个替代原同余不等式，我们称该操作为对于原同余不等式的“压缩”。

一个猜想是，将所有同余不等式压缩之后， $\text{lcm}_{i=1}^k m_i$ 就是最小的解的模数。

我们可以比较简单地举出这个猜想的反例：

$$\begin{cases} x \equiv \{0, 2, 3\} & (\bmod 4) \\ x \equiv \{0, 4\} & (\bmod 6) \end{cases}$$

上面同余不等式组均经过压缩，但存在合法解 $x \equiv \{0, 4\} \pmod{6}$ ，而 $\text{lcm}(4, 6) = 12$ 。

出现问题的一大原因在于，根据第二个方程， x 一定是 2 的倍数，从而 $\bmod 4$ 意义下其绝对不可能等于 3。所以说，3 这个元素对于整个同余方程是“无效的”。在去除 3 这个元素之后，第一个方程可以继续压缩。

可以发现出现这种情况的原因是同余不等式集合中存在许多实际上“无效”的元素，干扰了压缩操作的进行。我们将元素的“有效性”形式化定义：

定义 4.3 (有效性). 线性同余不等式组中的某条线性同余不等式 $x \equiv S \pmod{m}$ ，对于其中的某个 $r \in S$ ，如果 $\exists e \in \mathbb{Z}$ 使得其满足该不等式组中每一条不等式，且 $e \equiv r \pmod{m}$ ，则称 r 这个元素是“有效的”，反之是“无效的”。

根据定义，删去线性同余不等式组中所有“无效的”元素，不会影响解的结构。我们考虑利用有效性的想法修正上面给出的猜想。

定理 4.1. 删去所有不等式中所有“无效的”元素，然后压缩所有不等式，最后剩下来的所有不等式模数的最小公倍数就是原不等式解的最小模数。

证明. 定义 $f(e)$ 表示 e 是否满足不等式组中所有的同余不等式，我们需要说明 $W = \text{lcm}_{i=1}^k m_i$ 为 f 函数的最小正整数周期。由于上述的去除“无效”元素、压缩操作不改变解的结构，所以求出来的最小公倍数 W 一定至少对应了一个周期，我们只需要证明该周期的最小性即可。

假设 f 的最小周期 $w < W$ ，则有 $w|W$ 。由于 W 为 m_i 的最小公倍数，那么一定 $\exists i, m_i \nmid w$ 。对于满足 $m_i \nmid w$ 的某个方程 i ，由于所有元素都“有效”，除了存在方程 $x \equiv 0 \pmod{1}$ 的平凡情况，一定有 $S_i \neq \emptyset$ 。找出任意一个元素 $r \in S_i$ ，由有效性的定义可知 $\exists e, f(e) = 1$ 且 $e \equiv r \pmod{m_i}$ 。由于 w 是 f 的一个周期，所以 $\forall k \in \mathbb{Z}, f(e + kw) = f(e) = 1$ ，而这就说明了 $\forall k \in \mathbb{Z}, (e + kw) \bmod m_i \in S_i$ ，即 $\forall k \in \mathbb{Z}, [r + k \gcd(w, m_i)] \bmod m_i \in S_i$ 。由于 $m_i \nmid w$ ，有 $\gcd(w, m_i) < m_i$ ，原不等式可以被压缩成一个模数为 $\gcd(w, m_i)$ 的不等式，与原不等式经过压缩矛盾。

综上所述，该算法的正确性得证。 $\square$

暴力判断每一个元素是否有效，其时间复杂度并不优于朴素算法。我们需要更加高效的方法来判断一个元素的有效性。

考虑推广例题 4.1 中对于质因子的指数折半的思想，对于 $T = \prod_{p \leq B} p^{\lfloor \frac{v_p(B)}{2} \rfloor}$ ，枚举 $x \equiv r \pmod{T}$ ，将所有的模数为 m_i 的同余不等式转化为一个模数为 $\frac{m_i}{\gcd(m_i, T)}$ 的同余不等式，合并模数具有倍数关系的同余不等式之后，原问题转化成了模数两两互质的情况。此时由于中国剩余定理，所有还剩下来的元素一定都是“有效的”。对于原不等式中的元素，只需要在枚举 r 之后的任意一个问题中有效，就是有效的。因此，我们可以在与例题 4.1 相同的复杂度内解决最小周期问题。

例题 4.3 (生命的循环⁴). 给定一张 n 个点 m 条边的带权有向图。定义 $f(x)$ 表示是否存在一条从 1 到 n 的、长度为 x 的途径，求 $f(x)$ 去掉一个充分长的前缀之后的最小周期。保证该图中所有简单回路的长度 $\leq B$ 。途径即为一个任意长度边序列，满足前一条边的终点等于后一条边的起点。途径的起点定义为第一条边的起点，终点定义为最后一条边的终点，长度定义为序列中所有边的边权和。

注意到任意一条途径总是可以拆成一条简单路径和若干条简单回路，考虑延长现有的有向途径构造。设一条长度为 L 的途径与 c 条简单回路相交，设 c 条简单回路长度分别为 $l_1, l_2, \dots, l_c$ 。由于简单回路的部分可以分别重复任意多次，利用裴蜀定理容易证明，当 x 充分大时，途径总长度可以延长取到 $x \equiv L \pmod{\gcd(l_1, l_2, \dots, l_c)}$ 的所有 x 。以下我们称一条途径的 $\gcd(l_1, l_2, \dots, l_c)$ 为该途径的周期。

以下记边 e 的起点为 u_e ，终点为 v_e ，长度为 w_e 。

定理 4.2 (强连通图的周期). 对于一张强连通带权有向图来说，取以任意一点 rt 为根的外向生成树，记 dis_u 表示只经过树边 $rt \rightarrow u$ 的路径总长度。对于所有非树边 e ， $w_e - (dis_{v_e} - dis_{u_e})$ 的 $\gcd$ 就是所有回路长度的最大公因数。称该式为强连通图的周期。

证明. 我们可以从两方面说明上述定理。设答案为 g ，算法给出的结果为 d 。

一方面首先有 $d|g$ ，因为对于任何一条简单回路 C ，有 $d|\sum_{e \in C} w_e - (dis_{v_e} - dis_{u_e})$ ，可以推出 $d|\sum_{e \in C} w_e$ 。

另一方面由强联通性，存在 $v \rightarrow rt$ 的一条途径，这条途径和 $rt \rightarrow v$ 的长度为 dis_v 的途径组成了一条回路。由 g 的定义这条回路的长度是 g 的倍数。此时 $w_e - (dis_{v_e} - dis_{u_e})$ 在模 g 意义下同余于 $u \rightarrow v$ 、 $v \rightarrow rt$ 、 $rt \rightarrow u$ 拼接成的一条回路，长度同样是 g 的倍数，因此有 $g|d$ 。

综上所述有 $d = g$ ，定理 4.2 的正确性得证。 $\square$

由定理 4.2，对于任意一张有向图，如果一条途径经过了一个存在至少一条回路强连通块，我们可以取经过这个强连通块中所有点的一条回路，增广原途径，即将这个途径目前的周期与当前强连通块的周期取 $\gcd$ 。

⁴本题为笔者命题的一道原创集训队互测题，来源：<https://qoj.ac/contest/1813/problem/9491>

由于图中所有简单环环长均不超过 B ，一定所有强连通块的周期不超过 B 。我们考虑记忆化搜索出模每一种周期意义下，有哪些长度的途径。搜索过程中维护当前的周期 g' 、当前的途径长度对 g' 取模后的结果 r 、当前途径的终点 u ，然后枚举 u 出边指向的所有节点 v 转移。如果遇到了重复访问的状态 (g', r, u) 就返回。

不幸的是，这个搜索算法的复杂度仍然是指数级的。由于起始状态为 $(0, 0, 1)$ ，在进入第一个拥有回路的强连通块之前，其 g' 保持为 0， r 保持为当前途径的长度， r 的种类数上界并没有保证。

应该首先利用类似的记忆话搜索求出一个数组 $V_{g',r,u}$ ，表示是否存在一条 $1 \rightarrow u$ 的途径模 g' 余 r ，再从 $V_{g',r,u}$ 为真的状态开始运行上述记忆话搜索算法。由于所有的状态均有 $r < g' \leq B, u \leq n$ ，每个状态至多访问一次，访问时遍历其所有出边，所以该部分的复杂度为 $O(B^2m)$ 。

当 x 充分大时， $1 \rightarrow n$ 是否存在一条长度为 x 的途径，只与我们是否求出来的所有状态 (g', r, n) 中，是否存在至少一个状态满足 $x \equiv r \pmod{g'}$ 。考虑这个问题的反面， $f(x) = 0$ 当且仅当对于所有的 (g', r, n) 状态都有 $x \not\equiv r \pmod{g'}$ 。这就是一个线性同余不等式组的最小周期问题，套用例 4.2 中的算法即解决了这个问题。

5 进一步的复杂度分析

本节中的小写字母 e 指代的是自然对数的底而非一个正整数。

接下来，我们考虑对例题 4.1、例题 4.2 最后的复杂度进行一些更加深入的讨论。

这两道题复杂度的关键部分都是 $\prod_{p \leq B} p^{\lfloor \frac{v_p(B)}{2} \rfloor}$ 。一个明显的结论是，该式关于 B 的增长是 $O(\sqrt{B}!)$ 的。本节的一些讨论将给出一个更精确的估计，指出该式的增长率级别为 $e^{\sqrt{B} + o(\sqrt{B})}$ 。

一个不太精细的放缩是 $\prod_{p \leq B} p^{\lfloor \frac{v_p(B)}{2} \rfloor} \leq \sqrt{\prod_{p \leq B} p^{v_p(B)}}$ ，而 $\prod_{p \leq B} p^{v_p(B)}$ 就等于 $\text{lcm}_{i=1}^B i$ 。考虑首先运用对数法估计后者的增长率。

5.1 前若干个自然数最小公倍数的增长率

定义 $\psi(x) = \sum_{p \leq x} v_p(x) \ln(p)$ ，定义 $\theta(x) = \sum_{p \leq x} \ln(p)$ 。根据定义可以发现 $\psi(x) = \ln(\text{lcm}_{i=1}^x i)$ 。

考虑 $\psi(x)$ 的一个实际意义，对于 $1 \sim x$ 中的所有数，每有一个数的质因子分解形式形如 p^α ，就将 $\ln(p)$ 加入答案中一次。我们换一种枚举方式，枚举 α ，要求 $p \leq x^{\frac{1}{\alpha}}$ 。这就是 $\theta(x)$ 的实际意义，因此有：

$$\psi(x) = \sum_{\alpha \geq 1} \theta(\lfloor x^{\frac{1}{\alpha}} \rfloor)$$

这个和式只有 $O(\log_2 x)$ 项，因为当 $\alpha > \log_2(x)$ ， $\theta(\lfloor x^{\frac{1}{\alpha}} \rfloor)$ 一定为 0。而由于 $\theta(x) \leq x \ln x$ ，除去第一项之后，剩下的每一项都 $\leq \sqrt{x} \ln x$ ，我们就得到了：

$$\psi(x) = \theta(x) + O(\sqrt{x} \log_2 x \ln x)$$

。

对于素数个数函数 $\pi(x)$ ，有 $\theta(x) \geq \pi(x)$ ，而后者根据素数定理增长率为 $\frac{x}{\ln x}$ 。则 $\psi(x)$ 表达式中 $O(\sqrt{x} \log_2 x \ln x)$ 部分对于 $\theta(x)$ 一定是小量可以忽略。

定理 5.1. $\theta(x) < 2x \ln 2$ 。

证明. 设 $M = \binom{2m+1}{m}$ ，由二项式定理，其在 $(1+1)^{2m+1}$ 的展开式中出现两次。所以有 $2M < 2^{2m+1}$ ，即 $M < 2^{2m}$ 。

又由于 $M = \frac{(2m+1)!}{m!(m+1)!}$ ，所有质数 $p \in (m+1, 2m+1]$ 一定都整除 M ，即：

$$\prod_{p \in (m+1, 2m+1]} p \leq M < 2^{2m}$$

两边取对数得到 $\theta(2m+1) - \theta(m+1) < 2m \ln 2$ 。

考虑对 $\theta(x) < 2x \ln 2$ 进行数学归纳，对于 $x = 1, 2$ 的情况容易验证其成立，名设其对 $x < x_0$ 的所有自然数均成立。

当 x_0 为合数时一定有 $\theta(x_0) = \theta(x_0 - 1)$ ，归纳成立；否则 x_0 为奇质数，设 $m = \frac{x_0-1}{2}$ ，有：

$$\begin{aligned} \theta(x_0) &= \theta(2m+1) - \theta(m+1) + \theta(m+1) \\ &< 2m \ln 2 + 2(m+1) \ln 2 = 2x_0 \ln 2 \end{aligned}$$

归纳同样成立，从而定理得证。□

由定理 5.1 可以看出 $\psi(x), \theta(x)$ 都是线性增长的，有 $\psi(x) < 2x \ln 2 + o(x)$ 。

更进一步地，如果不局限在初等数学的范畴，参考文献 [3] 给出了 $\psi(x) = x + o(x)$ ；[2] 甚至给出了黎曼猜想等价于 $\psi(x) = x + O(\sqrt{x} \ln^2 x)$ 的结论。由于这些结论难以绕开解析数论知识，笔者在这方面一无所知，因此无法展开讨论，感兴趣的读者可以自行查阅文献。

由于 $\psi(x)$ 为 $\text{lcm}_{i=1}^x i$ 取对数后的函数，所以其被渐进复杂度记号忽略的部分也对原增长率有显著影响，难以使用常规的 O 记号直接表示 $\text{lcm}_{i=1}^x i$ 的渐进增长。根据定理 5.1 我们至少能得到 $\text{lcm}_{i=1}^x i$ 的一个上界 $4^{x+o(x)}$ ，该上界对于实际情况来说过于松了。根据 $\psi(x) = x + o(x)$ 的结论， $\text{lcm}_{i=1}^x i$ 的增长率是 $e^{x+o(x)}$ 级别的。换句话说即对于任意小的正数 ϵ 来说，当 x 充分大时总有 $(e - \epsilon)^x < \text{lcm}_{i=1}^x i < (e + \epsilon)^x$ 。

5.2 原式增长率

对于原式 $\prod_{p \leq B} p^{\lfloor \frac{v_p(B)}{2} \rfloor}$ 来说，如果将增长率估计为 $(\sqrt{e})^{x+o(x)}$ ，相对于实际情况仍然有极大的相对误差。原因是原式中下取整函数实际上极其关键，实际当 B 稍微大一点后 $\prod_{p \leq B} p^{\frac{v_p(B)}{2}}$ 就远大于 $\prod_{p \leq B} p^{\lfloor \frac{v_p(B)}{2} \rfloor}$ 。这也揭示了线性同余不等式组问题中一些基于将模数分成两组暴力

求解合并的算法为何远远劣于质数指数折半的算法。我们考虑分析 $\psi'(x) = \sum_{p \leq x} \lfloor \frac{v_p(x)}{2} \rfloor \ln p$ 的增长率。

注意到 $\lfloor \frac{1}{2} \rfloor = 0$ 且 $\lfloor \frac{v_p(x)}{2} \rfloor = v_p(\lfloor \sqrt{x} \rfloor)$ ，所以有：

$$\psi'(x) = \sum_{p^2 \leq x} \lfloor \frac{v_p(x)}{2} \rfloor \ln p = \sum_{p \leq \lfloor \sqrt{x} \rfloor} v_p(\lfloor \sqrt{x} \rfloor) \ln p = \psi(\lfloor \sqrt{x} \rfloor)$$

因此，由 5.1 节中的部分结论，原式的增长率可以估计为 $e^{\sqrt{x}+o(\sqrt{x})}$ 。当 $x = 100$ 时，该式给出估计 22026.46，与实际情况更加接近。

6 总结

本文归纳了一类同余问题，类比“线性同余方程”的名字，暂时地统称这类问题为“线性同余不等式”问题。

对于单个线性同余不等式，本文主要讨论了一种复杂度分析形如欧几里德算法的递归算法，这类算法一般被人统称为“类欧几里德”的算法，同样是信息学竞赛中一个十分有趣的话题。

对于线性同余不等式组问题则给出了一个较为巧妙且通用的基于质数指数折半的算法，其能够解决的频率、解的周期等多种形式的题目，体现了折半思想在指数级复杂度题目中的多种多样的应用。

质数指数折半的算法复杂度中包含了增长率形如 $e^{\sqrt{x}+o(\sqrt{x})}$ 的部分，这种算法复杂度通常难以使用普通的渐进记号估计。近年来，算法竞赛中也同样出现了一些对于非常规算法复杂度考察的题目，例如在最近的 UOJ Round #28 中便出现了一道复杂度为 $e^{\tilde{O}(\sqrt[3]{x})}$ 的问题，可见非常规复杂度的算法问题仍是 OI 中一大值得研究的板块。

笔者水平仍然十分有限，对于线性同余不等式类的问题发掘较浅，希望各位读者能更加深入研究这类问题，寻找这类问题更多的变体、解决算法和应用。

7 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢父母对我一直以来的关心、鼓励与支持。

感谢谭献龙老师在我 OI 学习生涯中的细心教导。

感谢长郡中学 2025 届信息组的同学给予我的长期陪伴与帮助。

感谢国家集训队杨耀良、彭思进、张艺缤教练的指导。

感谢周桓毅同学关于本文复杂度证明部分提供的重要建议。

感谢彭思进、周桓毅、朱屹帆、程楷轩为本文验稿并提出宝贵的改进意见。

参考文献

- [1] Walk (graph theory). [https://en.wikipedia.org/wiki/Walk_\(graph_theory\)](https://en.wikipedia.org/wiki/Walk_(graph_theory)), 2024.
- [2] Harold Davenport. Multiplicative Number Theory, page 104. Springer-Verlag, 2nd edition, 1980.
- [3] Pierre Dusart. Estimates of some functions over primes without RH. arXiv preprint arXiv:1002.0442, 2010.
- [4] E.M.Wright G.H. Hardy. 哈代数论, 第六版, pages 343–345. 人民邮电出版社, 2010.



全国青少年信息学奥林匹克竞赛 (CCF NOI)

E-mail: noi@ccf.org.cn

网址: <https://www.noi.cn/>

