



浅谈一类蒙日矩阵

北京大学附属中学 张景行

摘要

本文将介绍一类特殊的蒙日矩阵相关的算法以及其在信息学竞赛中的应用。

1 引言

蒙日矩阵 (Monge Matrix 或 Monge Array), 即其中元素满足四边形不等式的矩阵, 是信息学竞赛中常见的一种矩阵, 在动态规划等方面均有着广泛的应用。而其中的简单单位蒙日矩阵有着更多的性质。对于一般的 $n \times n$ 矩阵, 其 $\min +$ 乘法的计算目前没有 $O(n^{3-\epsilon})$ 的算法, 但是对于蒙日矩阵, 以及简单单位蒙日矩阵, 其分别存在 $O(n^2)$ 以及 $O(n \log n)$ 的乘法算法。

本文将主要介绍简单单位蒙日矩阵的乘法算法, 其是如何优化一些经典问题的复杂度的, 以及介绍其在信息学竞赛中的应用。

作者是从 Library Checker^{[1][2]} 中两题了解到了一篇介绍该算法的论文^[3], 本文中涉及单位蒙日矩阵的算法均来自该论文。

据作者的了解, 在本文撰写之时, 单位蒙日矩阵在信息学竞赛中并没有被广泛应用。希望通过本文的介绍, 读者能够对简单单位蒙日矩阵这一简洁但强大的结构有所理解, 并希望可以启发读者利用它解决更多问题, 以及启发读者命制利用该算法的新颖题目。

2 基本概念

此处给出本文中使用的符号的定义, 并对一些定义加以说明。以及指出本文研究的对象的一些基本的性质。

2.1 基本定义

定义 2.1 (艾佛森括号). 我们令 $[X] = \begin{cases} 1 & X \text{ 为真} \\ 0 & X \text{ 为假} \end{cases}$ 。

定义 2.2. 我们用 i^+ 表示 $i + 1/2$, i^- 表示 $i - 1/2$ 。

定义 2.3. 我们用 $[i : j]$ 表示集合 $\{i, i + 1, \dots, j\}$, $\langle i : j \rangle$ 表示集合 $\{i^+, i^+ + 1, \dots, j^-\}$ 。

定义 2.4 (矩阵). 若一个矩阵 A 的下标 $A_{i,j}$ 中 i, j 的范围分别是 X, Y , 则我们称 A 是 X, Y 上的。本文中, 我们规定下标范围一定是 $[i : j]$ 或 $\langle i : j \rangle$ 。与惯例一致, i 对应着从上到下的行数, j 对应着从左到右的列数。本文中, 若不加说明, 矩阵元素均为实数。

定义 2.5 (列向量). 若一个矩阵 a 是在 X, Y 上的, 其中 $|Y| = 1$, 我们称 a 为一个列向量, 并用 a_i 来表示 $a_{i,j}$, 其中 j 是 Y 中唯一一个元素。

2.2 蒙日矩阵

定义 2.6. 对于一个 $[i_1 : i_2], [j_1 : j_2]$ 上的矩阵 A , 定义其密度矩阵 A^M 为一个 $\langle i_1, i_2 \rangle, \langle j_1 : j_2 \rangle$ 上的矩阵, 其中 $A_{i,j}^M = A_{i^-,j^+} + A_{i^+,j^-} - A_{i^-,j^-} - A_{i^+,j^+}$ 。

定义 2.7. 对于一个 $\langle i_1 : i_2 \rangle, \langle j_1 : j_2 \rangle$ 上的矩阵 A , 定义其分布矩阵 A^D 为一个 $[i_1, i_2], [j_1 : j_2]$ 上的矩阵, 其中 $A_{i,j}^D = \sum_{i' \in \langle i_1 : i_2 \rangle, j' \in \langle j_1 : j_2 \rangle} A_{i',j'}$ 。

不难看出, 上述的取密度矩阵和分布矩阵的操作分别为对矩阵取二维差分和前缀和。对于 i 是从大到小, 对于 j 是从小到大, 若将矩阵写出来, 可以理解为从左下角到右上角的前缀和和差分。

定义 2.8 (蒙日矩阵). 我们称一个矩阵 A 是蒙日矩阵, 当且仅当 A^M 的每个元素都是非负的。

读者可能对动态规划中的四边形不等式有所了解, 蒙日矩阵的等价定义就是满足四边形不等式的矩阵。

引理 2.1 (四边形不等式). 一个矩阵 A 是蒙日矩阵, 当且仅当对于所有 $i_1 \leq i_2, j_1 \leq j_2$, $A_{i_1,j_1} + A_{i_2,j_2} \leq A_{i_2,j_1} + A_{i_1,j_2}$ 。

证明. 首先这显然是一个充分条件, 取 $i_1 = i^-, i_2 = i^+, j_1 = j^-, j_2 = j^+$ 即得密度矩阵的定义。

必要性则是由于 $A_{i_2,j_1} + A_{i_1,j_2} - A_{i_1,j_1} - A_{i_2,j_2} = \sum_{i \in \langle i_1 : i_2 \rangle, j \in \langle j_1 : j_2 \rangle} A_{i,j}^M \geq 0$ 。□

2.3 简单、单位、亚单位蒙日矩阵

定义 2.9 (简单蒙日矩阵). 若一个蒙日矩阵满足第一列和最后一行的元素都为 0, 则我们称其是简单蒙日矩阵。

引理 2.2. 一个蒙日矩阵 A 是简单蒙日矩阵, 当且仅当 $A^{MD} = A$ 。

证明. 回顾定义 2.6 2.7, 取密度矩阵是二维差分, 取分布矩阵是二维前缀和, 若第一列和最后一行为 0, 两者显然互为逆运算, 故 $A^{MD} = A$. 若 $A^{MD} = A$, 注意到对于任意矩阵 A , A^D 的第一列和最后一行都为 0, 故 A 是简单蒙日矩阵. $\square$

定义 2.10 (置换矩阵). 我们称一个矩阵是一个置换矩阵, 当且仅当其是一个方阵, 且每行和每列除了一个元素为 1 以外其它元素均为 0.

定义 2.11 (亚置换矩阵). 我们称一个矩阵是一个亚置换矩阵, 当且仅当其是全零的, 或去掉所有全零行和全零列后是个置换矩阵.

定义 2.12 (单位蒙日矩阵). 若一个蒙日矩阵 A 满足其密度矩阵 A^M 是置换矩阵, 则我们称其为单位蒙日矩阵.

定义 2.13 (亚单位蒙日矩阵). 若一个蒙日矩阵 A 满足其密度矩阵 A^M 是亚置换矩阵, 则我们称其为亚单位蒙日矩阵.

显然, 单位蒙日矩阵一定是亚单位蒙日矩阵.

注意对于简单单位蒙日矩阵和简单亚单位蒙日矩阵, 其密度矩阵可以使用一个置换或者一个点集来表示, 并不需要存储整个矩阵, 所以这类矩阵通常使用其密度矩阵对应的置换或点集来表示.

定义 2.14. 若一个蒙日矩阵 A 是单位的, 也是简单的, 那么我们称其为简单单位蒙日矩阵. 类似的, 我们称同时是亚单位的和简单的蒙日矩阵为简单亚单位蒙日矩阵.

引理 2.3. 一个蒙日矩阵 A 是简单单位蒙日矩阵, 当且仅当 $A_{i,j}$ 为非负整数, 且其第一列和最后一行都为 0, 第一行从左到右为 $0, 1, 2, \dots, n$, 最后一列从下到上为 $0, 1, 2, \dots, n$, 其中 $n+1$ 是矩阵的长和宽.

证明. 不妨设 A 是 $[i_1 : i_2], [j_1 : j_2]$ 上的, 依然根据定义, 先证必要性, 若其为简单单位蒙日矩阵, 注意其密度矩阵的每一行和每一列恰有一个 1, 故 $A_{i_1,j} = A_{i_1,j-1} + 1$, $A_{i,j_2} = A_{i+1,j_2} + 1$, 即得. 再证充分性, 由于 $A_{i_1,j} = A_{i_1,j-1} + 1$, $A_{i,j_2} = A_{i+1,j_2} + 1$, 且元素均为整数, 故其密度矩阵必定每行每列恰有一个 1. $\square$

引理 2.4. 一个矩阵 A 是简单亚单位蒙日矩阵, 当且仅当其是简单蒙日矩阵, 且 $A_{i,j}$ 为非负整数, 且第一行和最后一列的相邻两个元素差不超过 1.

证明. 与引理 2.3 同理. $\square$

推论 2.1. 一个矩阵 A 是简单亚单位蒙日矩阵, 当且仅当其是简单蒙日矩阵, 且 $A_{i,j}$ 为非负整数, 且 $A_{i,j+1} \leq A_{i,j} + 1$, $A_{i-1,j} \leq A_{i,j} + 1$.

证明. 不妨设 A 是在 $[i_1 : i_2], [j_1 : j_2]$ 上的, 充分性显然, 必要性则是因为引理 2.1, $A_{i,j+1} - A_{i,j} \leq A_{i_1,j+1} - A_{i_1,j} \leq 1$, $A_{i,j} - A_{i+1,j} \leq A_{i,j_2} - A_{i+1,j_2} \leq 1$. $\square$

该推论说明了, 如果一个矩阵是简单蒙日矩阵, 每个元素都是整数, 且相邻两个位置的差不超过 1, 那么它就是简单亚单位蒙日矩阵, 我们在下文会给出若干在信息学竞赛中常见的, 满足这个条件的矩阵的例子。

推论 2.2. 对于简单亚单位蒙日矩阵 A , 若 A 在 $[i_1 : i_2], [j_1 : j_2]$ 上, 那么一定有 $0 \leq A_{i,j} \leq \min(j - j_1, i_2 - i)$ 。

证明. 对于每个元素到矩阵的左边和下边连续使用推论 2.1 即得。 $\square$

2.4 距离乘法

定义 2.15 (距离乘法). 对于在 X, Y 上的矩阵 A , Y, Z 上的矩阵 B , 我们定义 A, B 的距离乘法的结果是一个在 X, Z 上的矩阵 C , 其中 $C_{i,j} = \min_{k \in Y} A_{i,k} + B_{k,j}$, 记为 $AB = C$, 若不多加说明, 本文中所有矩阵乘法均为距离乘法。

可以发现, 本文中研究的几种矩阵均对于距离乘法封闭。

定理 2.1. 蒙日矩阵对距离乘法封闭。换言之, 对于蒙日矩阵 A, B , AB 是蒙日矩阵。

证明. 令 $C = AB$, $C_{i,j} = \min_k A_{i,k} + B_{k,j}$ 。

考虑由于引理 2.1, 只需证明对于所有 $i_1 \leq i_2, j_1 \leq j_2$, $C_{i_1,j_2} + C_{i_2,j_1} \geq C_{i_1,j_1} + C_{i_2,j_2}$ 即可

$C_{i_1,j_2} = \min_k A_{i_1,k} + B_{k,j_2}$, 令 k_1 为任意一个这个式子取到最小值的 k ; $C_{i_2,j_1} = \min_k A_{i_2,k} + B_{k,j_1}$, 令 k_2 为任意一个这个式子取到最小值的 k 。

若 $k_2 \leq k_1$, 那么由于 A 是蒙日矩阵, 以及引理 2.1, $A_{i_1,k_2} + A_{i_2,k_1} \leq A_{i_1,k_1} + A_{i_2,k_2}$, 那么,

$$\begin{aligned} C_{i_1,j_1} + C_{i_2,j_2} &= \left(\min_k A_{i_1,k} + B_{k,j_1} \right) + \left(\min_k A_{i_2,k} + B_{k,j_2} \right) \\ &\leq A_{i_1,k_2} + B_{k_2,j_1} + A_{i_2,k_1} + B_{k_1,j_2} \\ &= (A_{i_1,k_2} + A_{i_2,k_1}) + (B_{k_2,j_1} + B_{k_1,j_2}) \\ &\leq (A_{i_1,k_1} + A_{i_2,k_2}) + (B_{k_2,j_1} + B_{k_1,j_2}) \\ &= (A_{i_1,k_1} + B_{k_1,j_2}) + (A_{i_2,k_2} + B_{k_2,j_1}) \\ &= C_{i_1,j_2} + C_{i_2,j_1} \end{aligned}$$

若 $k_1 \leq k_2$, 则由于 B 是蒙日矩阵, $B_{k_1,j_1} + B_{k_2,j_2} \leq B_{k_2,j_1} + B_{k_1,j_2}$, 那么,

$$\begin{aligned}
 C_{i_1, j_1} + C_{i_2, j_2} &= \left(\min_k A_{i_1, k} + B_{k, j_1} \right) + \left(\min_k A_{i_2, k} + B_{k, j_2} \right) \\
 &\leq A_{i_1, k_1} + B_{k_1, j_1} + A_{i_2, k_2} + B_{k_2, j_2} \\
 &= (A_{i_1, k_1} + A_{i_2, k_2}) + (B_{k_1, j_1} + B_{k_2, j_2}) \\
 &\leq (A_{i_1, k_1} + A_{i_2, k_2}) + (B_{k_2, j_1} + B_{k_1, j_2}) \\
 &= (A_{i_1, k_1} + B_{k_1, j_2}) + (A_{i_2, k_2} + B_{k_2, j_1}) \\
 &= C_{i_1, j_2} + C_{i_2, j_1}
 \end{aligned}$$

故不论 k_1, k_2 大小关系, 均有 $C_{i_1, j_1} + C_{i_2, j_2} \leq C_{i_1, j_2} + C_{i_2, j_1}$, 所以 C 是蒙日矩阵。 $\square$

定理 2.2. 简单单位蒙日矩阵对距离乘法封闭。换言之, 对于简单单位蒙日矩阵 A, B , AB 是简单单位蒙日矩阵。

证明. 由于定理 2.1, 以及引理 2.3, 只需证明 C 满足引理 2.3 的条件即可。

不妨设 A, B, C 都是在 $[0 : n], [0 : n]$ 上的。

对于第一列, $C_{i, 0} = \min_k A_{i, k} + B_{k, 0}$, 由于 $A_{i, k} + B_{k, 0} \geq 0$, $A_{i, 0} + B_{0, 0} = 0$, 故 $C_{i, 0} = 0$ 。

对于最后一行, $C_{n, j} = \min_k A_{n, k} + B_{k, j}$, 由于 $A_{n, k} + B_{k, j} \geq 0$, $A_{n, n} + B_{n, j} = 0$, 故 $C_{n, j} = 0$ 。

对于最后一列, $C_{i, n} = \min_k A_{i, k} + B_{k, n} = n + \min_k A_{i, k} - k$, 由于 $A_{i, n} - n = n - i - n = -i$, 考虑根据推论 2.1 $A_{i, k} \leq A_{i, k-1} + 1$, 故 $A_{i, k} - k$ 关于 k 单调不增, 故 $C_{i, n} = n - i$ 。

对于第一行, $C_{0, j} = \min_k A_{0, k} + B_{k, j} = \min_k k + B_{k, j}$, 由于 $0 + B_{0, j} = j$, 且 $B_{k-1, j} \leq B_{k, j} + 1$, 故 $k + B_{k, j}$ 关于 k 单调不降, 故 $C_{0, j} = j$ 。 $\square$

定理 2.3. 简单亚单位蒙日矩阵对距离乘法封闭。换言之, 对于简单亚单位蒙日矩阵 A, B , AB 是简单亚单位蒙日矩阵。

证明. 与定理 2.2 相似, 只需证明 C 满足引理 2.4 的条件即可。

不妨设 A, B 分别是在 $[0 : a], [0 : b], [0 : b], [0 : c]$ 上的。

对于第一列, $C_{i, 0} = \min_k A_{i, k} + B_{k, 0}$, 由于 $A_{i, k} + B_{k, 0} \geq 0$, $A_{i, 0} + B_{0, 0} = 0$, 故 $C_{i, 0} = 0$ 。

对于最后一行, $C_{a, j} = \min_k A_{a, k} + B_{k, j}$, 由于 $A_{a, k} + B_{k, j} \geq 0$, $A_{a, b} + B_{b, j} = 0$, 故 $C_{a, j} = 0$ 。

对于最后一列, 根据引理 2.4, $C_{i, c} = \min_k A_{i, k} + B_{k, c} \leq \min_k A_{i+1, k} + 1 + B_{k, c} = 1 + C_{i+1, c}$ 。

对于第一行, $C_{a, i} = \min_k A_{a, k} + B_{k, i} \leq \min_k A_{a, k} + B_{k, i-1} + 1 = 1 + C_{a, i-1}$ 。 $\square$

3 快速距离乘法

对于一般的矩阵, 其距离乘法的计算目前没有 $O(n^{3-\epsilon})$ 的算法。但是对于蒙日矩阵, 以及简单 (亚) 单位蒙日矩阵, 其分别存在 $O(n^2)$ 以及 $O(n \log n)$ 的乘法算法。本节中将重点介绍简单 (亚) 单位蒙日矩阵的 $O(n \log n)$ 的乘法。

$d_1 \backslash d_2$	< 0	$= 0$	> 0
< 0	✓		
$= 0$	✓	✓	
> 0	✓	✓	✓

表 1: 完全单调矩阵的条件

3.1 一般蒙日矩阵与向量的乘法

定义 3.1 (完全单调矩阵). 我们称一个矩阵 A 是完全单调的, 当且仅当对于所有 $i_1 < i_2, j_1 < j_2$, 有 $\neg(A_{i_1, j_2} \leq A_{i_1, j_1} \wedge A_{i_2, j_2} > A_{i_2, j_1})$ 且 $\neg(A_{i_1, j_2} < A_{i_1, j_1} \wedge A_{i_2, j_2} = A_{i_2, j_1})$, 换言之, 若令 $d_1 = A_{i_1, j_2} - A_{i_1, j_1}, d_2 = A_{i_2, j_2} - A_{i_2, j_1}$, 那么只能出现表 1 中是 ✓ 的情况。

完全单调矩阵满足决策单调性, 严谨的说, 就是如下的引理。

引理 3.1 (决策单调性). 对于一个完全单调矩阵 A , 对于 $i_1 < i_2$, 若 $A_{i_1, k_1} = \min_k A_{i_1, k}$, 那么一定存在 $k_2 \geq k_1$, 使得 $A_{i_2, k_2} = \min_k A_{i_2, k}$; 对称的, 若 $A_{i_2, k_2} = \min_k A_{i_2, k}$, 那么一定存在 $k_1 \leq k_2$, 使得 $A_{i_1, k_1} = \min_k A_{i_1, k}$ 。

证明. 先证前半部分, 若不成立, 则考虑一定存在 $i_1 < i_2, k_1, \min_k A_{i_1, k} = A_{i_1, k_1}$, 存在 $k_3 < k_1$, 使得 $\forall k_2 \geq k_1, A_{i_2, k_2} > A_{i_2, k_3}$ 。

故一定有 $A_{i_2, k_1} > A_{i_2, k_3}$, 但是由于 $\min_k A_{i_1, k} = A_{i_1, k_1}, A_{i_1, k_3} \geq A_{i_1, k_1}$, 与定义 3.1 矛盾。

考虑再证对称的情况, 若不成立, 则一定存在 $i_1 < i_2, k_2, \min_k A_{i_2, k} = A_{i_2, k_2}$, 存在 $k_3 > k_2$, 使得 $\forall k_1 \leq k_2, A_{i_1, k_1} > A_{i_1, k_3}$ 。

故一定有 $A_{i_1, k_2} > A_{i_1, k_3}$, 由于 $\min_k A_{i_2, k} = A_{i_2, k_2}, A_{i_2, k_2} \leq A_{i_2, k_3}$, 与定义 3.1 矛盾。□

这告诉了我们, 对于某一行, 若这行的某一个最小值在第 k 列, 则其上面的行的一定存在一个 $\leq k$ 列的最小值, 下面的行一定存在一个 $\geq k$ 列的最小值。这也就是我们常说的决策单调性。

推论 3.1. 对于一个完全单调矩阵 A , 则 A 每行最靠左 (右) 的最小值的所在列数单调不降。

证明. 考虑根据引理 3.1, 若第 i 行最靠左的最小值的位置在第 k 列, 则第 $i-1$ 行一定存在 $\leq k$ 列的最小值, 故最靠左的最小值的位置一定 $\leq k$ 。靠右同理。□

引理 3.2. 若一个矩阵是完全单调的, 那么其选择任意某些行列组成的子矩阵也是完全单调的。

证明. 观察完全单调矩阵的定义, 显然对于子矩阵也成立。□

引理 3.3. 若一个矩阵是蒙日矩阵, 那么它一定是完全单调的。

证明. 对于 $i_1 < i_2, j_1 < j_2$, 若不满足完全单调的定义, 则考虑根据定义 3.1, 一定有 $(A_{i_1, j_2} - A_{i_1, j_1}) + (A_{i_2, j_1} - A_{i_2, j_2}) < 0$, 但根据引理 2.1, $A_{i_1, j_1} + A_{i_2, j_2} \leq A_{i_1, j_2} + A_{i_2, j_1}$, 矛盾。 $\square$

回到一般蒙日矩阵与向量的乘法, 我们使用 SMAWK^[4] 算法来解决这个问题。

定理 3.1 (SMAWK 算法). 对于一个长宽的最大值为 n 的蒙日矩阵 A , 以及一个列向量 b , 可以只得知 $O(n)$ 个矩阵中的元素的情况下, 在 $O(n)$ 的时间内计算出 Ab 。

证明. 先考虑 A 是一个方阵的情况。

我们考虑令 $C_{i,j} = A_{i,j} + b_j$, 我们的目标就是在只得知 $O(n)$ 个 C 中的元素的情况下求出 C 每一行的最靠左的最小值的所在位置。显然 $C^M = A^M$, 故 C 也是蒙日矩阵, 根据引理 3.3, C 是完全单调的。

我们考虑每次首先求出 C 中所有下标为奇数的行构成的矩阵, 称为 D , 根据引理 3.2, D 也是完全单调的, 假设 D 有 m 行, 其中必定只有最多 m 列包含一个最靠左的最小值, 此时我们试图消除多余的列。

不妨设 D 是在 $[1:m], [1:n]$ 上的。

考虑从左到右依次加入 D 的每一列, 并维护当前没有被消除的列所构成的一个序列 P , 设 P 的长度为 t , P_i 为 P 从右往左第 $i+1$ 个列的编号。

我们将维护 P , 使得任意时刻, 对于 P_i, P_{i+1} , 有 $D_{t-i-1, P_{i+1}} \leq D_{t-i-1, P_i}$ 。

试图加入第 j 列时, 若 $t = 0$, 则直接将第 j 列加入末尾;

若 $D_{t, P_0} \leq D_{t, j}, t < m$, 则也直接将第 j 列加入 P 的末尾。

若 $D_{t, P_0} \leq D_{t, j}, t = m$, 现证明不可能有任何一行的最靠左的最小值位于第 j 列。

考虑对于 i , 若第 i 行的最靠左的最小值在第 j 列, 考虑由第 i, m 行, P_0, j 列构成的 2×2 子矩阵, 根据引理 3.2 其必定是完全单调的, 但注意到其第 i 行的最靠左最小值位于第 j 列, 第 m 行的最靠左的最小值位于第 P_0 列, 与推论 3.1 矛盾。

故若 $D_{t, P_0} \leq D_{t, j}, t = m$, 只需跳过该列即可。

若 $D_{t, P_0} > D_{t, j}$, 现证明不可能有任何一行的最靠左的最小值位于第 P_0 列。

考虑对于 $i > t$, 若第 i 行的最靠左的最小值在第 P_0 列, 考虑由第 t, i 行, P_0, j 列构成的 2×2 子矩阵, 根据引理 3.2 其必定是完全单调的, 但注意到其第 t 行的最靠左最小值位于第 j 列, 第 i 行的最靠左的最小值位于第 P_0 列, 与推论 3.1 矛盾。对于 $i = t$, 显然最靠左的最小值不可能位于第 P_0 列。对于 $i < t$, 只需考虑 $t \geq 2$ 的情况, 此时, 若第 i 行的最靠左的最小值位于第 P_0 列, 由于 $D_{t-1, P_1} \leq D_{t-1, P_0}$, 考虑由第 $i, t-1$ 行, P_1, P_0 列构成的子矩阵, 显然, 第 $t-1$ 行的最靠左的最小值位于第 P_1 列, 第 i 行的位于第 P_0 列, 同理, 矛盾。

故若 $D_{t, P_0} > D_{t, j}$, 此时从 P 中删去 P_0 , 并继续检查是否有 $D_{t, P_0} > D_{t, j}$, 一直删除直到 P 为空, 或者 $D_{t, P_0} \leq D_{t, j}$, 此时加入第 j 列。



容易发现,这实际上就是维护了一个类似单调栈的结构,由于每一列最多被加入一次,删除一次,每次判断只要求一个位置的值,故只需要花费 $O(n)$ 的时间,得知 $O(n)$ 个矩阵元素,而且最后 P 中最多剩余 m 个元素。

消除了多余的列后,变成了行数列数均减半的一个子问题,我们求出这个子问题的答案后,由于推论 3.1,偶数行的最靠左的最小值一定在前后两行最靠左最小值位置之间,而相邻两个奇数行之间的最左最小值位置的个数的和是 $O(n)$ 的,所以只要求出最多 $O(n)$ 个位置的值即可。

所以时间和得知的矩阵元素个数的复杂度均为 $T(n) = T(n/2) + O(n) = O(n)$,而若 A 不是方阵,若其列数大于行数,则先用 $O(n)$ 消除多余的列即可,显然复杂度依然不超过 $O(n)$ 。□

由于一般蒙日矩阵相关算法并不是本文的重点,故不展开讨论。

3.2 一般蒙日矩阵的乘法

定理 3.2. 对于两个蒙日矩阵 A, B , 可以在 $O(n^2)$ 的时间内计算 $C = AB$, 其中 n 为 A, B 的长度、宽度的最大值。

显然可以直接对于 B 的每一列均套用一次 SMAWK 算法得到一个 $O(n^2)$ 的算法,但是存在更加简洁的算法。

证明. $C_{i,j} = \min_k A_{i,k} + B_{k,j}$, 不妨设 $p_{i,j}$ 为 $C_{i,j}$ 取到最小值的最小的 k , 对于每一列,考虑 B 在该列对应的列向量与 A 的乘法,显然根据推论 3.1, 有 $p_{i,j} \leq p_{i+1,j}$ 。

根据蒙日矩阵关于副对角线的对称性,可以发现 $p_{i,j} \leq p_{i,j+1}$, 具体证明不加赘述。

那么我们考虑从下到上,从左到右依次求出每一个 $C_{i,j}, p_{i,j}$, 由于 $p_{i,j-1} \leq p_{i,j} \leq p_{i+1,j}$, 最小值点只需要 $p_{i,j-1}, p_{i+1,j}$ 这两个位置之间枚举即可(若这两个元素不存在则视为 $0, y$), 也就是 $p_{i+1,j} - p_{i,j-1} + 1$ 个位置,考虑把 $p_{i+1,j} - p_{i,j-1} + 1$ 对于某一条副对角线上所有 i, j 求和,其等于 y 与这条对角线的元素个数的和,是 $O(n)$ 的,而一共有 $O(n)$ 条副对角线,所以总共枚举的最小值点不超过 $O(n^2)$ 个,时间复杂度也不超过 $O(n^2)$ 。

□

由于一般蒙日矩阵相关算法并不是本文的重点,故不展开讨论。

3.3 简单亚单位蒙日矩阵与向量的乘法

注意如果直接套用定理 3.1 的算法,注意到每次询问是一个二维数点问题,如果使用树套树等数据结构是 $O(n \log^2 n)$ 的。原论文中给出了一个 $O(n)$ 的算法。我们这里将介绍一个简洁的 $O(n \log n)$ 算法。

定理 3.3. 对于一个长宽的最大值为 n 的简单亚单位蒙日矩阵 A , 以及一个列向量 b , 可以在 $O(n \log n)$ 的时间内计算 Ab 。

证明. 我们考虑令 $C_{i,j} = A_{i,j} + b_j$ 。显然 $C^M = A^M$, 故 C 是蒙日矩阵, 根据引理 3.3, C 是完全单调的。

同时, 注意到 $A_{i,j}$ 的某一个元素的求值是在 A^M 对应的点集上的一个二维前缀数点, 注意到 A^M 是一个亚置换矩阵, 故 $A_{i,j}$ 可以在 $O(1)$ 的时间内推出其相邻四个点的值, 即 $A_{i\pm 1, j\pm 1}$ 的值。

考虑分治算法, 如果对于第 m 行, 我们知道 $C_{m,k} = \min_{k'} C_{m,k'}$, 那么根据引理 3.1, 对于 $i < m$ 行, 一定存在 $\leq k$ 列的最小值, 对于 $i > m$ 行, 一定存在 $\geq k$ 列的最小值。那么我们可以截取 $\leq m-1$ 行, $\leq k$ 列构成的子矩阵, 以及 $\geq m+1$ 行 $\geq k$ 列构成的子矩阵分治计算。

考虑 $A_{i,j}$ 的计算, 考虑使用类似莫队算法的方式来维护, 也就是维护一个坐标 x, y , 以及对应的 $A_{x,y}$, 若需要计算 $A_{i,j}$, 则用 $|i-x| + |j-y|$ 次操作从 $A_{x,y}$ 得到 $A_{i,j}$, 然后把 x, y 设为 i, j 。

如果我们每次取 m 为正中间一行, 从左到右枚举所有的列, 计算出 k , 然后先递归计算上面的矩阵, 再递归下面的矩阵, 考虑坐标初始在第 m 行的最左边, 设当前矩阵长宽为 a, b , 则计算第 m 行所有列的值需要不超过 b 次运算, 然后将坐标移动到上面的矩阵, 以及在递归完成后再次移动到下面的矩阵都需要不超过 $a+b$ 次操作。故在此处最多需要 $O(a+b)$ 次操作就可以完成 x, y 坐标的移动。

那么注意到递归最多 $O(\log n)$ 层, 每一层所有矩阵的行数与列数和不超过 $O(n)$, 那么进行 $O(n \log n)$ 次操作, 复杂度即 $O(n \log n)$ 。□

3.4 简单单位蒙日矩阵的乘法

由于我们使用密度矩阵对应的置换来表示简单单位蒙日矩阵, 不妨引入一个置换的乘法。

定义 3.2. 我们称一个 $[1:n]$ 的置换是一个长度为 n 的数组, 其中 $[1:n]$ 中每个数恰好出现一次。

定义 3.3. 对于一个 $[1:n]$ 的置换 a , 定义 $\text{Mat}(a)$ 为一个 $\langle 0, n \rangle, \langle 0, n \rangle$ 上的矩阵, 满足 $\text{Mat}(a)_{i-j} = [a_i = j]$, 显然 $\text{Mat}(a)$ 存在反函数, 定义为 $\text{Mat}^{-1}(A)$ 。

定义 3.4. 对于两个 $[1:n]$ 的置换 a, b , 我们令 $a \otimes b = \text{Mat}^{-1} \left(\left(\text{Mat}(a)^D \text{Mat}(b)^D \right)^M \right)$ 。

由于简单单位蒙日矩阵的乘法是封闭的, 故这样的定义是正确的。

现在, 我们研究两个置换上的 $\otimes$ 运算。

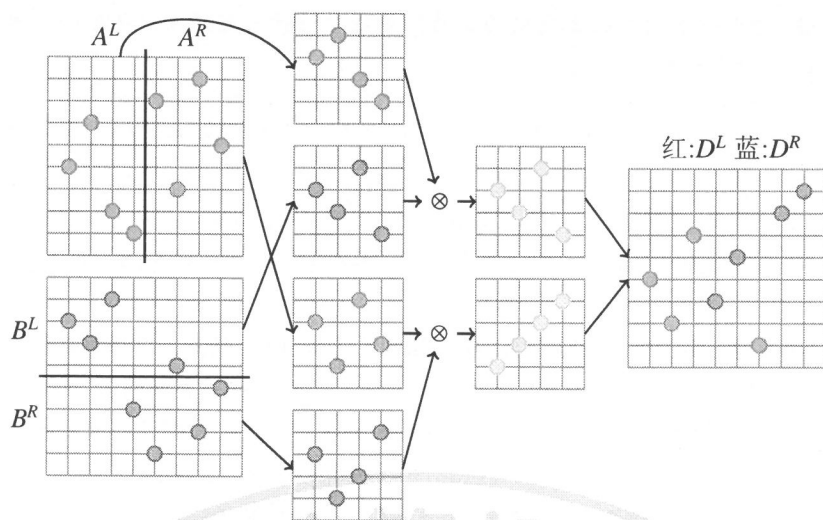


图 1: 分治得到 D^L, D^R 的过程

3.4.1 快速乘法

引理 3.4. 对于亚置换矩阵 $A, B, C = (A^D B^D)^M$, 若 A 的第 i 行全零, 则 C 中第 i 行也全零; 若 B 的第 j 列全零, 则 C 的第 j 列也全零。

证明. 考虑一个全零行对应在前缀和上是两个相同的行, 计算距离乘法时, A 的两个相同的行显然结果相同, B 的相同的列同理。□

引理 3.4 的意义是, 计算距离乘法时, 可以先删去全零行列, 得到结果后再插入。

定理 3.4. 对于两个长度为 n 的置换 a, b , 存在 $O(n \log n)$ 时间内计算 $a \otimes b$ 的算法。

证明. 考虑使用分治算法, 输入两个置换 a, b , 长度均为 n 。若置换长度 $n = 1$, 直接返回唯一的长度为 1 的置换即可。

否则, 令 $A = \text{Mat}(a), B = \text{Mat}(b)$, 考虑计算 $C = A^D B^D$ 时, $C_{i,j} = \min_{k \in [0:n]} A_{i,k}^D + B_{k,j}^D$ 。若我们将这个 k 的范围分成 $[0:m], [m:n]$ 两部分, 即考虑 C^L, C^R 两个矩阵, 其中 $C_{i,j}^L = \min_{k \in [0:m]} A_{i,k}^D + B_{k,j}^D, C_{i,j}^R = \min_{k \in [m:n]} A_{i,k}^D + B_{k,j}^D$ 。

令 A^L, A^R 为 A 截取 $\langle 0:m \rangle, \langle m:n \rangle$ 列的矩阵, B^L, B^R 为 B 截取 $\langle 0:m \rangle, \langle m:n \rangle$ 行的矩阵。

现考虑分治, 计算 $D^L = (A^L B^L)^M, D^R = (A^R B^R)^M$ 。由于引理 3.4, 并且在删去全零行列后, A^L, B^L, A^R, B^R 均是置换矩阵, 且大小分别为 $m, n-m$, 故转化为了两个规模为 $m, n-m$ 的子问题。过程的图示见 1, 除了子问题的求解都可以通过对应的置换上的操作在 $O(n)$ 时间内解决。

不难发现, D^L, D^R 的并是一个置换矩阵, 但其不一定是答案矩阵。我们接下来将说明如何从 D^L, D^R 得到答案矩阵。

令 A^{DL}, A^{DR} 为 A^D 截取 $[0:m], [m:n]$ 列的矩阵, B^{DL}, B^{DR} 为 B^D 截取 $[0:m], [m:n]$ 行的矩阵, 那么 $C^L = A^{DL} B^{DL}, C^R = A^{DR} B^{DR}$ 。

我们有 $A_{i,j}^{DL} = A_{i,j}^{LD}, B_{i,j}^{DR} = B_{i,j}^{RD}, A_{i,j}^{DR} = A_{i,j}^{RD} + A_{i,m}^{LD}, B_{i,j}^{DL} = B_{i,j}^{LD} + B_{m,j}^{RD}$ 。

$$\begin{aligned} C_{i,j}^L &= \min_{k \in [0:m]} A_{i,k}^{DL} + B_{k,j}^{DL} \\ &= \min_{k \in [0:m]} A_{i,k}^{LD} + B_{k,j}^{LD} + B_{m,j}^{RD} \\ &= \left(\min_{k \in [0:m]} A_{i,k}^{LD} + B_{k,j}^{LD} \right) + B_{m,j}^{RD} \\ &= D_{i,j}^{LD} + B_{m,j}^{RD} \\ &= D_{i,j}^{LD} + D_{0,j}^{RD} \end{aligned}$$

其中最后一步,是由于 A^R 是一个置换矩阵,且每列恰好有一个 1, $D_{0,j}^{RD} = \min_{k \in [m:n]} A_{0,k}^{RD} + B_{k,j}^{RD} = \min_{k \in [m:n]} k - m + B_{k,j}^{RD}$, 由于推论 2.1, $B_{k,j}^{RD} + k$ 关于 k 单调不降, 故最小值在 $k = m$ 时取到, $D_{0,j}^{RD} = B_{m,j}^{RD}$ 。

对称地, 易得 $C_{i,j}^R$ 的表达式。

$$\begin{aligned} C_{i,j}^R &= \min_{k \in [m:n]} A_{i,k}^{DR} + B_{k,j}^{DR} \\ &= \min_{k \in [m:n]} A_{i,k}^{RD} + A_{i,m}^{LD} + B_{k,j}^{RD} \\ &= \left(\min_{k \in [m:n]} A_{i,k}^{RD} + B_{k,j}^{RD} \right) + A_{i,m}^{LD} \\ &= D_{i,j}^{RD} + A_{i,m}^{LD} \\ &= D_{i,j}^{RD} + D_{i,n}^{LD} \end{aligned}$$

这样, 我们可以把 C 用 D^L, D^R 表示了。 $C_{i,j} = \min(C_{i,j}^L, C_{i,j}^R)$, 由于 $\min$ 的取值位置与两项的差有关, 我们考察 $\delta_{i,j} = C_{i,j}^L - C_{i,j}^R = D_{i,j}^{LD} + D_{0,j}^{RD} - D_{i,j}^{RD} - D_{i,n}^{LD} = (D_{0,j}^{RD} - D_{i,j}^{RD}) - (D_{i,n}^{LD} - D_{i,j}^{LD})$ 。

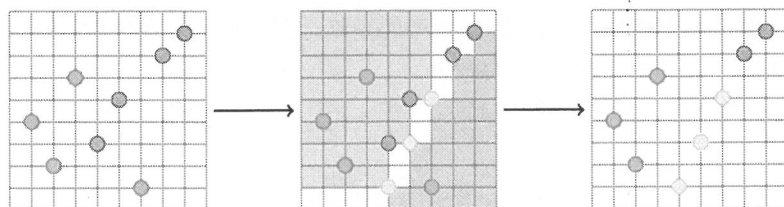
不难发现, $\delta_{i,j}$ 实际上是 D^R 从左到右下的二维前缀和与 D^L 从右下到左上的二维前缀和之差。也就是说 $\delta_{i,j} = \sum_{i' \in (0:i), j' \in (0:j)} D_{i',j'}^R - \sum_{i' \in (i:n), j' \in (j:n)} D_{i',j'}^L$ 。

注意到 D^L, D^R 的并是一个置换矩阵, 所以 $\delta_{i,j}$ 相邻两项的差最多为 1, 且观察到其关于行和列都是单调不降的, 也就是越向右下角越大。

令答案矩阵为 $D = C^M$ 。考虑 $D_{i,j} = C_{i^-,j^+}^R + C_{i^+,j^-}^R - C_{i^-,j^-}^R - C_{i^+,j^+}^R$, 也就是需要考察 (i,j) 周围这四个点的值。

若 $\delta_{i^-,j^-} \geq 0$, 那么这四个点的 δ 均 ≥ 0 , 最小值均在 C^R 取到。

$$\begin{aligned} D_{i,j} &= C_{i^-,j^+}^R + C_{i^+,j^-}^R - C_{i^-,j^-}^R - C_{i^+,j^+}^R \\ &= D_{i^-,j^+}^{RD} + D_{i^+,j^-}^{LD} + D_{i^-,j^-}^{RD} + D_{i^+,j^+}^{LD} - D_{i^-,j^-}^{RD} - D_{i^-,j^+}^{LD} - D_{i^+,j^-}^{RD} - D_{i^+,j^+}^{LD} \\ &= D_{i^-,j^+}^{RD} + D_{i^+,j^-}^{RD} - D_{i^-,j^-}^{RD} - D_{i^+,j^+}^{RD} \\ &= D_{i,j}^R \end{aligned}$$


 图 2: 从 D^L, D^R 得到 D 的过程

同理, 若 $\delta_{i^+, j^+} \leq 0$, 那么这四个点的 δ 均 ≤ 0 , 最小值均在 C^L 取到。

$$\begin{aligned} D_{i,j} &= C_{i^-, j^+}^L + C_{i^+, j^-}^L - C_{i^-, j^-}^L - C_{i^+, j^+}^L \\ &= D_{i^-, j^+}^{LD} + D_{0, j^+}^{RD} + D_{i^+, j^-}^{LD} + D_{0, j^-}^{RD} - D_{i^-, j^-}^{LD} - D_{0, j^-}^{RD} - D_{i^+, j^+}^{LD} - D_{0, j^+}^{RD} \\ &= D_{i^-, j^+}^{LD} + D_{i^+, j^-}^{LD} - D_{i^-, j^-}^{LD} - D_{i^+, j^+}^{LD} \\ &= D_{i,j}^L \end{aligned}$$

唯一剩下的情况是 $\delta_{i^-, j^-} < 0, \delta_{i^+, j^+} > 0$ 的情况, 由于相邻两项最多差 1, 所以这种情况一定有 $\delta_{i^-, j^-} = -1, \delta_{i^+, j^+} = 1, \delta_{i^+, j^-} = \delta_{i^-, j^+} = 0$, 考虑用 $C_{i^+, j^+}^L - 1$ 来表示 C_{i^+, j^+}^R 。

$$\begin{aligned} D_{i,j} &= C_{i^-, j^+}^L + C_{i^+, j^-}^L - C_{i^-, j^-}^L - (C_{i^+, j^+}^L - 1) \\ &= D_{i,j}^L + 1 \end{aligned}$$

由于 $D_{i,j}^L \geq 0, 1 \leq D_{i,j} \leq 1$, 这种情况下 $D_{i,j}$ 一定为 1。

那么, 在获得了 D^L 和 D^R 后, 我们只需要双指针求出 $\delta_{i,j} = 0$ 的区域的上下边界的轮廓线, 求出这些多出的 1, 其余部分直接填充 D^L 与 D^R 的并即可。依然只需要在对应置换上进行操作即可完成, 复杂度依然为 $O(n)$ 。这个过程的图示见图 2。

分治时, 显然取 $m = \lfloor n/2 \rfloor$ 最优, 故该算法的总复杂度 $T(n) = O(n) + 2T(n/2) = O(n \log n)$ 。

□

3.4.2 乘法的形式

虽然上一小节中给出了 $O(n \log n)$ 对两个置换进行 $\otimes$ 运算的算法, 但是我们依然对这个乘法具体的作用没有一个完整的了解。接下来我们将说明 $\otimes$ 在置换上的具体形式。

定理 3.5. 置换 $1, 2, \dots, n$ 是 $\otimes$ 运算上的单位元。

证明. 置换 $1, 2, \dots, n$ 对应的分布矩阵 B 有 $B_{i,j} = \max(0, j - i)$, 只需证对于所有简单单位蒙日矩阵 A , 都有 $AB = BA = A$ 即可。

令 $AB = C$, $C_{i,j} = \min_{k \in [0:n]} A_{i,k} + \max(0, j - k)$ 。在 $k = j$ 时, 取到 $A_{i,j}$; 在 $k > j$ 时, $A_{i,k} \geq A_{i,j}$; 在 $k < j$ 时, 根据推论 2.1, $A_{i,k} + j - k \geq A_{i,j}$ 。故 $C_{i,j} = A_{i,j}$ 。

$BA = A$ 的证明同理。

□

定理 3.6. 对于两个 $[1:n]$ 的置换 a, b , 且 b 中恰好有一个相邻的逆序对 $(i, i+1)$, 即 b 满足 $b_i = i+1, b_{i+1} = i$, 且对于 $j \notin \{i, i+1\}, b_j = j$. 令 u, v 为 a 中的下标, 使得 $a_u = i, a_v = i+1$.

若 $u > v$, 则 $a \otimes b = a$. 否则, $a \otimes b$ 为置换 a 交换 a_u, a_v 后的置换。

证明. 若 b 中恰好有一个相邻的逆序对 $(p, p+1)$, 考虑 $B = \text{Mat}(b)^D, B_{i,j} = \max(j-i, 0) + [i = p][j = p]$.

令 $A = \text{Mat}(a)^D, C = AB, C_{i,j} = \min_{k \in [0:n]} A_{i,k} + \max(0, j-k) + [k = p][j = p]$. 对于 $j \neq p$, 同定理 3.5 的证明, $C_{i,j} = A_{i,j}$.

若 $j = p$, 则 $C_{i,p} = \min_{k \in [0:n]} A_{i,k} + \max(0, p-k) + [k = p]$. 此时, 在 $k = p$ 时是 $A_{i,p} + 1$; 在 $k > p$ 时是 $A_{i,k}$, 最小值在 $A_{i,p+1}$ 时取到; 在 $k < p$ 时是 $A_{i,k} + j - k$, 根据推论 2.1, 最小值在 $A_{i,p-1} + 1$ 时取到. $C_{i,p} = \min(A_{i,p-1} + 1, A_{i,p} + 1, A_{i,p+1}) = \min(A_{i,p-1} + 1, A_{i,p+1})$.

同时, 由于式子相对于 $j \neq p$ 的情况只多了 $[k = p]$ 一个非负项, 我们有 $C_{i,p} \geq A_{i,p}$.

若 $a_u = p, a_v = p+1$, 若 $i < u$, 则 $A_{i,p-1} + 1 = A_{i,p}, C_{i,p} = A_{i,p}$; 若 $i \geq v$, 则 $A_{i,p+1} = A_{i,p}, C_{i,p} = A_{i,p}$.

故若 $u > v$, 则对于所有 i , 均有 $C_{i,p} = A_{i,p}$, 故 $C = A, a \otimes b = a$.

若 $u < v$, 则对于 $i \in [u: v-1]$, $A_{i,p-1} = A_{i,p} = A_{i,p+1} - 1, C_{i,p} = A_{i,p} + 1$. C 相对 A 的变化是在第 $[u: v-1]$ 行, 第 k 列加上 1, 在密度矩阵 A^M 上, 则是在 $A_{u^-, p^-}^M, A_{v^-, p^+}^M$ 处减一, 在 $A_{u^-, p^+}^M, A_{v^-, p^-}^M$ 处加一, 在 a 上就是交换了 a_u, a_v 的值. $\square$

定理 3.7. 对于任意一个置换 a , 一定可以将其拆成若干个恰好有一个相邻逆序对的置换的 $\otimes$, 即存在若干恰有一个相邻逆序对的置换 $b_1, b_2, \dots, b_t$, 使得 $a = b_1 \otimes b_2 \otimes \dots \otimes b_t$.

证明. 不妨考虑按照 b 的逆序对数量进行归纳, 没有逆序对显然是单位置换, 由 0 个这样的置换可得。

否则, 考虑最小的 i , 使得 $b_u = i, b_v = i+1, u > v$, 考虑令 b' 为 b 交换 b_u, b_v 得到的置换, b' 比 b 恰好少一个逆序对, 故考虑先得到 $c_1 \otimes c_2 \otimes \dots \otimes c_t = b'$, 令 t 为交换 $i, i+1$ 的置换, 由于 b' 中对应 $b_u = i, b_v = i+1$ 的位置有 $u < v$, 故 $b' \otimes t = b$. $\square$

有了上面的定理, 我们可以直观的描述这个运算的过程. 对于 b , 将其按照顺序拆分成一个个相邻的交换的操作, 若 a 中两个元素已经是逆序的了, 则不操作, 否则进行交换. 如果我们使用两列点之间从第一列第 i 个点到第二列第 a_i 个点连边表示置换 a 的话, 如图 3, 可以形象的画出乘法的过程. 将 b 拆成若干个相邻交换后, 若这两条线之前在 a 中已经相交, 则不进行对换, 否则对换。

实际上, 根据这个过程, 很容易猜到这个运算的零元。

引理 3.5. 对于一个 $[1:n]$ 的置换 a , a 是置换 $n, n-1, \dots, 1$ 当且仅当 $\text{Mat}(a)_{n-i,i}^D = i$.

证明. 容易验证这是必要条件, 充分性则考虑 $\text{Mat}(a)_{n-1,1}^D = 1$, 故一定有 $a_n = 1$, 然后 $\text{Mat}(a)_{n-2,2}^D = 2$, 故一定有 $a_{n-1} = 2$, 以此类推. $\square$

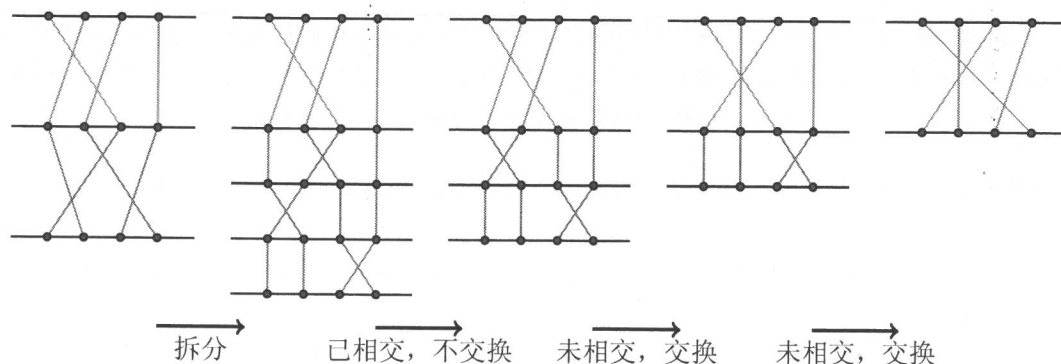


图 3: 对两个置换进行 $\otimes$ 运算

定理 3.8. 置换 $n, n-1, \dots, 1$ 是 $\otimes$ 运算上的零元, 即令 a 为置换 $n, n-1, \dots, 1$, b 为任意置换, $a \otimes b = b \otimes a = a$.

证明. 置换 $n, n-1, \dots, 1$ 对应的分布矩阵 A 有 $A_{i,j} = \min(n-i, j)$, 只需证对于所有简单单位蒙日矩阵 B , 都有 $AB = BA = A$ 即可。

令 $AB = C$, $C_{n-i,i} = \min_{k \in [0:n]} \min(i, k) + B_{k,i}$. 在 $k = n$ 时, 取到 i ; 在 $i \leq k < n$ 时, $i + B_{k,i} \geq i$; 在 $k < i$ 时, 根据推论 2.1, $i - B_{k,i} = B_{0,i} - B_{k,i} \leq k$. 故 $k + B_{k,i} \geq i$, 故 $C_{n-i,i} = i$, 根据引理 3.5, $C = A$.

$BA = A$ 的证明同理。 $\square$

不严谨的说, $\otimes$ 运算上, 每一个置换还对应着类似它的左 (右同理, 不赘述) 零化子的另一个置换。

定义 3.5. 对于一个 $[1:n]$ 的置换 a , 定义其旋转置换 a^R 为另一个 $[1:n]$ 的置换, 使得若 $a_i = j, a_{n-j}^R = i$ 。

定理 3.9. 对于任意置换 a , $a^R \otimes a$ 是置换 $n, n-1, \dots, 1$ 。

证明. 考虑令 $A = \text{Mat}(a), B = \text{Mat}(a^R)$, 容易发现 B 实际上是 A 逆时针旋转四分之一圈得到的, 那么容易得到 $j - B_{n-j,i}^D = A_{i,j}^D$ 。

令 $C = B^D A^D$, $C_{n-i,i} = \min_k B_{n-i,k}^D + A_{k,i}^D = \min_k i - A_{k,i}^D + A_{k,i}^D = i$, 根据引理 3.5, 得证。 $\square$

3.5 简单亚单位蒙日矩阵的乘法

我们假设亚排列矩阵是由其 1 的位置的点集表示的。

定理 3.10. 对于亚排列矩阵 A, B , 可以在 $O(n \log n)$ 时间内计算 $(A^D B^D)^M$, 其中 n 为 A, B 的长度、宽度的最大值。

证明. 首先, 应用引理 3.4. 计算 $C = (A^D B^D)^M$ 时, 可以先删除 A 的全零行以及 B 的全零列, 得到 A^S, B^S , 计算 $C^S = (A^{SD} B^{SD})^M$ 后再进行补全.

删除全零行列后, A^S, B^S 仍然有可能并不是置换矩阵, 考虑向 A^S 上方补上适当的矩阵, B^S 右方补上适当的矩阵, 使得 $\begin{bmatrix} * \\ A^S \end{bmatrix}, \begin{bmatrix} B^S & * \end{bmatrix}$ 是置换矩阵. 注意在上面和右面添加并不会改

变其分布矩阵的原有位置的值, 故 $\begin{bmatrix} * \\ A^S \end{bmatrix}^D \begin{bmatrix} B^S & * \end{bmatrix}^D = \begin{bmatrix} * & * \\ C^S & * \end{bmatrix}^D$.

故直接套用定理 3.4 的算法计算补齐的矩阵乘法即可得到 C^S . $\square$

通过定理 3.10 的算法中补上任意排列的过程, 容易得到类似简单单位蒙日矩阵的乘法的形式.

计算 $C = (A^D B^D)^M$ 时, 我们如果将 A 的全零列看成有一个在 $-\infty$ 列的 1, B 的全零行看成有一个 $+\infty$ 列的 1, 第 i 行第 j 列的 1 对应一个第一排第 i 个位置向第二排第 j 个位置的边, 进行与图 3 相同的操作, 然后对于结果的上下位置 i, j 均不是 $\pm\infty$ 的边, $C_{i,j} = 1$.

4 应用

4.1 特殊网格图距离矩阵维护

有一类题目, 其本质是在要求求一类特殊的网格图上的距离的相关信息, 经典的题目如下.

例题 1 (静态区间 LIS).¹ 给定一个长度为 n 的排列 a , 以及 q 次询问 l, r , 每次输出该置换在 $[l, r]$ 内的最长上升子序列长度. $n, q \leq 10^5$.

解法 我们设排列的元素下标和值域都是 $[0, n)$.

我们考虑这样的—个网格图, 一共有 $n+1$ 行, $n+1$ 列, 我们用 (i, j) 来表示第 i 行第 j 列的点. (i, j) 到 $(i, j+1)$ 连边权为 1 的有向边, (i, j) 到 $(i+1, j)$ 连边权为 0 的有向边, 而对于给定的排列 a , 若 $a_x = y$, 则 (y, x) 到 $(y+1, x+1)$ 连边权为 0 的有向边.

令 $A_{i,j}$ 为 $(0, i)$ 到 (n, j) 的最短路长度, 显然, $[l, r]$ 内的最长上升子序列的长度就是 $r - l - A_{l,r}$. 若规定, 对于 $r \leq l, A_{l,r} = 0$, 可以发现, 对于 $A_{l-1,r+1} + A_{l,r}$ 对应的两条最短路, 由于这两条路径一定相交, 所以从某一条交点把剩下的部分交换, 得到了总长度相同的, $(0, l)$ 到 $(n, r+1)$ 与 $(0, l-1)$ 到 (n, r) 的路径, 故 $A_{l-1,r} + A_{l,r+1} \leq A_{l-1,r+1} + A_{l,r}$. 然后再考虑 $r \leq l$ 的一些平凡的情况, 容易得到 A 是一个简单蒙日矩阵. 由于可以通过扩展 $A_{l,r}$ 的最优路径开头和结尾得到 $A_{l,r+1}, A_{l-1,r}$, 所以 $A_{l,r}$ 相邻两项的差不超过 1, 根据推论 2.1, A 是一个简单亚单位蒙日矩阵.

¹来源^[1]

如果我们能快速计算给定置换的 A 对应的 A^M , 由于 A^M 中最多只有 n 个为 1 的点, 那么就可以通过简单的二维数点得到每个子区间的最长上升子序列的长度。

我们考虑分治计算, 将网格图分成上下两部分, 如果令 B, C 为另两个矩阵, $B_{i,j}$ 为 $(0, i)$ 到 (m, j) 的最短路, $C_{i,j}$ 为 (m, i) 到 (n, j) 的最短路长度。也规定 $j \leq i$ 时 $B_{i,j} = C_{i,j} = 0$, 那么, 由于向后走一定不优于不动, 所以这些 $j \leq i$ 的部分一定不改变最小值, 故 $A = BC$ 。

而注意到, 此时, 上下两行中有些列之间的间隔是空的, 即有两列之间只有 (i, j) 到 $(i, j+1)$ 边权为 1 的边, 那么我们删去这些间隔, 分治计算得到对应的两个缩小的网格的 B', C' 。而注意到向一个网格中插入一个 i 列到 $i+1$ 列的边权为 0 的间隔对 B' 的影响是将第 i 行, i 列依次复制一份, 而边权加 1, 则是在复制后的右上角的部分全部加一。对 B'^M 的影响就是在第 i^+ 行 i^+ 列的位置插入一行一列, 除 $B'_{i^+, i^+} = 1$ 外全为 0。

那么, 对于很多的间隔, 实际上, 我们就是先在其中插入所有间隔位置对应的行和列, 然后插入的行列的位置的对角线上填上 1, 显然可以在对应的排列上 $O(n)$ 解决。

所以, 我们可以在 $O(n)$ 的时间内, 从删去空间隔的 B^M, C^M 中得到删去之前的 B^M, C^M , 从而使用定理 3.10, 以 $O(n \log n)$ 的复杂度计算出 A^M , 若分治时将网格的分界点选在 $\lfloor n/2 \rfloor$ 处, 则总复杂度为 $T(n) = 2T(n/2) + O(n \log n) = O(n \log^2 n)$ 。

将询问离线后二维数点即可做到 $O(\log n)$ 回答一组询问。 ■

对于上一题中这种稀疏的图, 似乎没有什么显然的优于分治的算法, 但是对于稠密的图, 是还有着优化空间的。我们来考虑下一例题。

例题 2 (子串-前缀 LCS).² 给定两个长度为 n, m 的字符串 a, b , q 次询问 x, y, z , 回答 a 的长度为 x 的前缀与 b 从第 y 个字符到第 $z-1$ 个字符的子串的最长公共子序列长度。 $n, m, q \leq 10^4$ 。

解法 我们设 a, b 的下标范围分别是 $[0, n), [0, m)$ 。

我们考虑与上一题类似的网格图, 若 $a_i = b_j$, 则在 (i, j) 至 $(i+1, j+1)$ 加边权为 0 的边。令 $A_{l,r}^{(i)}$ 代表 $(0, l)$ 到 (i, r) 的最短路长度 ($r \leq l$ 时为 0), 与上一题相同, $A^{(i)}$ 均是简单亚单位蒙日矩阵。显然, 一组询问要求的就是 $z - y - A_{y,z}^{(x)}$ 。

注意到, 若令 $B_{l,r}^{(i)}$ 代表 (i, l) 到 $(i+1, r)$ 的最短路长度 ($r \leq l$ 时为 0), B 也是简单亚单位蒙日矩阵, 且有 $A^{(i)} = A^{(i-1)} B^{(i-1)}$ 。显然对于 $l < r$, $B_{l,r}^{(i)} = r - l - [\exists p \in [l, r), a_i = b_p]$, 现考虑 $B^{(i)M}$ 的形式。

对于 $l, l+1$, 考虑 $B_{l,r}^{(i)} - B_{l+1,r}^{(i)} = 1$ 的最小的 r , $B^{(i)M}$ 第 l^+ 行 1 所处的位置即是 r^- 。

对于 $r \leq l$, 这显然不可能成立; 对于 $r = l+1$, $B_{l,r}^{(i)} - B_{l+1,r}^{(i)} = 1 \iff a_i \neq b_l$; 对于 $r > l+1$, $B_{l,r}^{(i)} - B_{l+1,r}^{(i)} = 1 \iff [\exists p \in [l, r), a_i = b_p] = [\exists p \in [l+1, r), a_i = b_p]$ 。

若 $a_i \neq b_l$, 显然最小的就是 $l+1$; 否则, 考虑最小的 $l_1 > l$, 使得 $a_i = b_{l_1}$, 对于 $r \leq l_1$, 显然均不成立, 且在 $r = l_1 + 1$ 时等式成立, 所以最小的是 $l_1 + 1$ 。

²来源^[2], 数据范围有修改

那么 $B^{(i)M}$ 可以通过倒着扫描一遍直接求出, 直接套用定理 3.10 即可得到 $O(mn \log m)$ 求出所有 $A^{(i)}$ 的算法。

考察定理 3.10 的过程, 注意到将 $B^{(i)M}$ 补其成置换矩阵之后, 其对应排列最多有 m 个逆序对, 因为观察上面构造 $B^{(i)M}$ 的方式, 对于每一个 $a_i = b_l$, 第 l 行的 1 与第 $l+1$ 至 l_1-1 行的 1 构成逆序对, 其中 l_1 为最小的 $l_1 > l$, 使得 $a_i = b_{l_1}$, 所有相邻两个相等位置的距离和不超过 m 。

同时, 很容易得到这个对应排列的一个拆成相邻交换的 $\otimes$ 的方法, 而根据定理 3.6, 与一个只有一个相邻逆序对的排列的 $\otimes$ 显然可以 $O(1)$ 计算, 所以和这种特殊的简单亚单位蒙日矩阵的乘法可以做到 $O(m)$ 。

那么, 可以 $O(nm)$ 得到所有 $A^{(i)}$, 对于每个询问可以 $O(m)$ 计算, 可以在 $O(nm + mq)$ 的复杂度内解决。■

值得一提的是, 本题也可以使用二维数点来计算答案, 可以让一次询问的复杂度降低, 但会增加预处理所需的复杂度。

4.2 1D-1D 动态规划

有一些 1D-1D 的动态规划的代价矩阵是一个简单亚单位蒙日矩阵。其一些相关问题的求解可以得到比信息竞赛已经大范围引入的动态规划技巧更优的复杂度。

例题 3.³ 给定一个长度为 n 的序列, 以及常数 k , 有 q 次询问 l, r , 求若将这个序列下标在 $[l:r]$ 内这个子段分成至多 k 个子段, 每一段的不同的元素个数的和最大是多少。 $n, q \leq 10^5$ 。

解法 不妨设序列下标从 0 开始, 那么与之前的例题相同, 设一个 $[0:n], [0:n]$ 上的矩阵 A , 对于 $r > l$, $r-l-A_{l,r}$ 为 $[l,r]$ 这个子段内颜色个数, 否则 $A_{l,r} = 0$ 。

依然, 我们说明 A 是简单亚单位蒙日矩阵。首先先说明其是蒙日矩阵。由于对于三个有限集合, A, B, C , 有 $|A \setminus B| + |C \setminus B| \geq |(A \setminus B) \cup (C \setminus B)| = |(A \cup C) \setminus B|$, 不等式两边加上 $2|B|$, 得到 $|A \cup B| + |B \cup C| \geq |B| + |A \cup B \cup C|$ 。颜色个数显然满足反向的四边形不等式, 但注意 $A_{l,r}$ 定义为了区间长度减去颜色个数, 所以 $A_{l,r}$ 在 $l < r$ 的时候满足 2.1 的条件。其他情况是平凡的, 故 $A_{l,r}$ 是蒙日矩阵。而显然其也满足推论 2.1 的条件, 故是简单亚单位蒙日矩阵。

我们现在考虑求 A^k , 即 A 的距离乘法的 k 次幂。同样的, $r < l$ 的转移一定不优, 故可以只考虑 $r \geq l$ 的情况, 根据距离乘法的定义, 显然这就等价于原问题。

现在, 我们只要求出 A^M , 类似上一题, 对每一个 l , 考虑使 $A_{l,r} - A_{l+1,r} = 1$ 的最小的 r 。不难发现 $A_{l,r} - A_{l+1,r} = 1$ 当且仅当 $[l,r]$ 这个子段的颜色个数与 $[l+1,r]$ 的相同, 也就是 l 的颜色在 $[l+1,r]$ 内出现过。显然最小的 r 就是 l 处颜色下一个出现位置加一。若序列的值域是 $O(n)$ 的, 则可以 $O(n)$ 求出 A^M 。

³ 加强自^[5]

接下来, 使用 $O(n \log n \log k)$ 的快速幂算法求出 A^{k^M} , 然后询问离线后使用二维数点 $O(q \log n)$ 求出答案即可。 ■

借助快速的简单亚单位蒙日矩阵之间的乘法, 以及其与向量的乘法算法, 可以解决一系列类似的问题。只要代价矩阵是一个简单亚单位蒙日矩阵就可以。由于其做法的相似性故不再列举其它例题。

4.3 Bruhat 序判定

实际上, 简单单位蒙日矩阵的乘法的结构也有很多值得探索的性质。例如计算一种对于排列的名为 Bruhat 序的偏序关系。

定义 4.1. 对于两个置换 a, b 我们称 $a \leq b$, 当且仅当可以进行若干次交换 a 中一个顺序对中两个元素的位置的操作后使 $a = b$ 。

我们发现, 这个操作与简单单位蒙日矩阵有很大的联系, 有如下的引理。

引理 4.1. 对于两个 $[1 : n]$ 的置换 a, b , $a \leq b$ 当且仅当 $\text{Mat}(a)^D$ 中每一个元素均小于等于 $\text{Mat}(b)^D$ 对应位置的元素。

证明. 考虑这是一个必要条件。交换顺序对后 $\text{Mat}(a)^D$ 中的每一个元素都不会变小。现证明其充分性。

若 $\text{Mat}(a)^D$ 中每一个元素均小于等于 $\text{Mat}(b)^D$ 对应位置的元素, 令 $A = \text{Mat}(a), B = \text{Mat}(b)$, 考虑关于 A^D, B^D 对应位置元素的差值的总和进行归纳, 显然, 若这个总和为 0, $a = b$ 。否则, 我们将说明一定可以交换一个 a 的顺序对或者 b 的一个逆序对, 使得条件依然成立, 但是总和变小。在归纳得到的操作序列前面或后面加入这次交换, 即可完成归纳。

否则, 从下到上考虑 A, B 的每一行, 找到最靠下的不同的行, 设这一行 A 的 1 在第 i 列, B 的 1 在第 j 列, 一定有 $j < i$, 否则中间一段 A^D 的元素就大于 B^D 了。设这行为第 k 行。

现在继续向上考虑每一行, 设这一行 A 的 1 在第 i_1 列, B 的 1 在第 j_1 列, 找到最靠下的一行, 使得 i_1, j_1 的其中至少一个 $\in \langle j^- : i^+ \rangle$ 。设这行为第 k_1 行。注意这一行是一定能找到的, 由于考虑 A 第 j 列的 1 的行数, 显然需要 $< k$, 否则由于 A, B 超过 k 的行相同, 那么 B 这里也是 1, 那么第 j 列 B 就有两个 1, 矛盾。故至少能找到这一行。

那么对于所有在第 k 和 k_1 行之间的行 k_2 , 设这一行 A 的 1 在第 i_2 列, B 的 1 在第 j_2 列, 那么 i_2, j_2 一定在 $\langle j^- : i^+ \rangle$ 外, 现在考虑, 注意到 $B^D - A^D$ 的每一行相对于下面一行的修改都是一个区间 $+1$ 或者 -1 , 而由于 i_2, j_2 在 $\langle j^- : i^+ \rangle$ 外, 所以 $B^D - A^D$ 的第 $[j^- : i^+]$ 列一定变化情况相同, 所以他们的差值和第 k^- 行是相同的, 我们知道 $B_{k^-, j^-}^D - A_{k^-, j^-}^D = B_{k^-, i^+}^D - A_{k^-, i^+}^D = 0$, 且对于 $p \in [j^+ : i^-]$, $B_{k^-, p}^D - A_{k^-, p}^D = 1$, 故对于 $p \in [j^+ : i^-]$, $B_{k_2^-, p}^D - A_{k_2^-, p}^D = B_{k_2^-, j^-}^D - A_{k_2^-, j^-}^D + 1 > 0$ 。

那么, $B^D - A^D$ 第 $[k_1^+ : k^-]$ 行, $[j^+ : i^-]$ 列均 > 0 。现在考虑第 k_1 行, 若 $j_1 \in \langle j^-, i^+ \rangle$, 那么注意到 B 第 k_1 行第 j_1 列的 1 与第 k 行第 j 列的 1 构成一个逆序对, 且其为顶点的矩形内

$B^D - A^D$ 的元素 > 0 , 故交换后矩阵所有元素依然 ≥ 0 , 且差值的总和变小, 是合法的归纳。
若 $i_1 \in \langle j^-, i^+ \rangle$ 同理。

所以归纳成立, 得证。 $\square$

定理 4.1. 对于两个 $[1:n]$ 的置换 a, b , $a \leq b$ 当且仅当 $a^R \otimes b$ 是置换 $n, n-1, \dots, 1$ 。

证明. 我们记置换 $n, n-1, \dots, 1$ 为 z 。 $A = \text{Mat}(a), B = \text{Mat}(b), C = \text{Mat}(a^R)$ 。由于 C 是 A 逆时针旋转四分之一圈得到的, 那么 $j - C_{n-j,i}^D = A_{i,j}^D$ 。

首先这是一个必要条件, 根据引理 4.1, A^D 中每一个元素均小于等于 B^D 对应位置的元素。注意根据距离乘法的定义, 对于一个矩阵 M , MA^D 中每一个元素也均小于等于 MB^D 对应位置的元素。则 $C^D A^D$ 中每一个元素均小于等于 $C^D B^D$ 对应位置的元素。根据定理 3.9, $C^D A^D = \text{Mat}(a^R \otimes a)^D = \text{Mat}(z)^D$ 。故 $(C^D B^D)_{n-i,i} \geq i$, 注意到根据推论 2.2, $(C^D B^D)_{n-i,i} \leq i$, 所以 $(C^D B^D)_{n-i,i} = i$, 根据引理 3.5, $a^R \otimes b = z$ 。

而考虑充分性。若 $a^R \otimes b = z$, 根据引理 3.5, $\min_k C_{n-i,k}^D + B_{k,i}^D = i$, 故对于所有 k , $C_{n-i,k}^D + B_{k,i}^D \geq i$, 那么 $i - A_{k,i}^D + B_{k,i}^D \geq i$, $B_{k,i}^D \geq A_{k,i}^D$, 根据引理 4.1, $a \leq b$ 。 $\square$

例题 4. 对于两个 $[1:n]$ 的置换 a, b , 求是否有 $a \leq b$ 。 $n \leq 10^5$ 。

解法 根据定理 4.1, 与定理 3.4, 立刻得到一个 $O(n \log n)$ 的算法。

当然, 也可以使用引理 4.1, 使用一个支持区间加, 查询区间最小值的线段树, 依次维护出两个分布矩阵的差的每一列, 并检查是否有负的元素。也可以做到 $O(n \log n)$ 。 ■

虽然本题并没有利用简单单位蒙日矩阵的算法得到更加优秀的复杂度, 但是它说明了置换上的 $\otimes$ 运算也有很多性质和值得研究的问题。但受限于作者的水平, 本文并没有对简单单位蒙日矩阵更加深刻的结构和性质以及这些性质在信息学竞赛中的应用进行探讨, 感兴趣的读者可以研究原论文^[3]中给出的相关的资料以及证明。

5 总结

本文中主要介绍了简单单位蒙日矩阵这一强大的结构, 可以使用 $O(n)$ 的空间表示一个满足特殊性质的 $n \times n$ 的距离矩阵, 并可以在 $O(n \log n)$ 的时间内进行乘法, 以及进行与向量的乘法。这种结构在信息学竞赛中广泛出现, 例如 LIS、LCS、不同元素个数等等。并且其相关算法易于实现, 常数合理, 可以将其与暴力算法进行区分, 在信息学竞赛中的潜力无限。

但是, 本文仅对于简单单位蒙日矩阵相关结构进行了简单的介绍, 其更深刻的性质, 以及如何将其有机地融入到题目中, 利用它命制优秀的信息学竞赛题目等等问题仍是未知的。希望本文能够启发读者继续深入研究相关理论, 使其更加成熟、完整。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢北大附中肖然老师的关心和指导。

感谢家人、朋友对我的支持与鼓励。

感谢李白天学长与我讨论以及给予的启发。

参考文献

- [1] Static Range LIS Query[EB/OL]. https://judge.yosupo.jp/problem/static_range_lis_query.
- [2] Prefix-substring LCS[EB/OL]. https://judge.yosupo.jp/problem/prefix_substring_lcs.
- [3] TISKIN A. Fast Distance Multiplication of Unit-Monge Matrices[J/OL]. *Algorithmica*, 2015, 71(4): 859-888. <https://doi.org/10.1007/s00453-013-9830-z>. DOI: 10.1007/s00453-013-9830-z.
- [4] AGGARWAL A, KLAWE MM, MORAN S, et al. Geometric applications of a matrix-searching algorithm[J/OL]. *Algorithmica*, 1987, 2(1): 195-208. <https://doi.org/10.1007/BF01840359>. DOI: 10.1007/BF01840359.
- [5] The Bakery[EB/OL]. <https://codeforces.com/problemset/problem/834/D>.



浅谈一类哈希表的复杂度分析

中国人民大学附属中学 陈于思

摘要

本文对哈希表进行了简要的介绍,介绍了几种解决哈希冲突的方式,分析了它们在对应的哈希函数下的复杂度并给出了证明。

1 引言

哈希表是一种广为人知的词典数据结构,由于其相比平衡树等其它词典结构实现更为简单且运行效率更高的特点,在信息学竞赛以及实际工程中均有广泛的应用。然而,目前国内鲜有对于哈希表复杂度的严格分析。本文将介绍几种较常见的哈希表实现方式,并对它们的时空复杂度进行分析。

2 哈希表

对正整数 s , 记 $[s] = \{0, 1, \dots, s-1\}$ 。

哈希表是一类可以通过键值 (key) 查找或修改对应的值 (value) 的数据结构, 支持以下操作:

1. $\text{Insert}(x, v)$ 插入键值 x 及对应的值 v 。
2. $\text{Modify}(x, v)$ 将键值 x 对应的值修改为 v 。
3. $\text{Delete}(x)$ 删除键值 x 及对应的值。
4. $\text{Query}(x)$ 查询键值 x 对应的值。

记 n 为哈希表中的元素个数, 那么 n 不超过操作的总次数。

设键值均是 $\mathcal{U} \subseteq [u]$ 中的元素, 其中 $u \gg n$ 。

哈希表通过哈希函数将键值集合 $\mathcal{U}$ 映射到规模较小的哈希值集合 $[m]$ 上, 通常我们选取的 m 满足 $m = \Theta(n)$ 且 $m > n$ 。对于一次涉及键值 x 的操作, 哈希表首先通过哈希函数 $h: \mathcal{U} \rightarrow [m]$ 得到 $y = h(x)$, 然后对存储位置 y 进行操作。

然而, 当存在两个不同的键值 $x_1, x_2 \in \mathcal{U}$ 满足 $h(x_1) = h(x_2)$ 时, 上述做法会出现错误。这种现象被称作哈希冲突。下文将介绍部分能够尽可能减少冲突的哈希函数, 并介绍几种解决冲突的方法。

3 哈希函数

定义 3.1. 称一个键值到哈希值的随机映射 $h: \mathcal{U} \rightarrow [m]$ 是一个哈希函数。

定义 3.2. 称从 $|\mathcal{U}|^m$ 个 $\mathcal{U}$ 到 $[m]$ 的映射中等概率随机选取的哈希函数 h 为全真随机哈希函数 (truly random hash function)。

定义 3.3. 称一组随机变量 $X_0, X_1, \dots, X_{n-1} \in S$ 是 k -独立 (k -wise independent) 的, 如果对任意互不相同的 k 个下标 $i_0, i_1, \dots, i_{k-1} \in [n]$ 与 k 个定值 $y_0, y_1, \dots, y_{k-1} \in S$, 都有

$$\Pr \left[\bigwedge_{j \in [k]} X_{i_j} = y_j \right] = \frac{1}{m^k}$$

定义 3.4. 设 $\mathcal{U} = \{x_0, x_1, \dots, x_{|\mathcal{U}|-1}\}$ 。称一个哈希函数 $h: \mathcal{U} \rightarrow [m]$ 是 k -独立的, 如果随机变量 $h(x_0), h(x_1), \dots, h(x_{|\mathcal{U}|-1})$ 是 k -独立的。

一个直观的理解是, 一个哈希函数 h 是 k -独立的, 当且仅当它满足对任意 k 个互不相同的键值 $x_0, x_1, \dots, x_{k-1}$, 对应的哈希值 $h(x_0), h(x_1), \dots, h(x_{k-1})$ 均可以被看作相互独立的 k 个随机变量。

性质 3.4.1. 如果哈希函数 h 是 $(k+1)$ -独立的, 那么 h 是 k -独立的。

证明。

$$\begin{aligned} & \Pr[h(x_0) = y_0 \wedge \dots \wedge h(x_{k-1}) = y_{k-1}] \\ &= \sum_{y_k=0}^{m-1} \Pr[h(x_0) = y_0 \wedge \dots \wedge h(x_k) = y_k] \\ &= m \times \frac{1}{m^{k+1}} \\ &= \frac{1}{m^k} \end{aligned}$$

□

性质 3.4.2. 对任意 k , 全真随机哈希函数都是 k -独立的。

给定 k , 我们给出一个 k -独立的哈希函数。

性质 3.4.3. 如果哈希函数 $h: \mathcal{U} \rightarrow [p]$ 是 k -独立的, 那么 $h'(x) = h(x) \bmod m$ 在 $p \gg m$ 时是 k -独立的。

引理 3.5. 取质数 $p > u$ 。等概率随机选择 $a_0, a_1, \dots, a_{k-1} \in [p]$ ，令 $h: \mathcal{U} \rightarrow [p]$ 为：

$$h(x) = (a_{k-1}x^{k-1} + \dots + a_1x + a_0) \bmod p$$

则 $h'(x) = h(x) \bmod m$ 是 k -独立的。

证明. 对于互不相同的 $x_0, x_1, \dots, x_{k-1} \in \mathcal{U}$ 与 $y_0, y_1, \dots, y_{k-1} \in [p]$ ，关于 $a_0, a_1, \dots, a_{k-1}$ 的模线性方程组 $\forall i \in [k], h(x_i) = y_i$ 对应的齐次线性方程组的系数矩阵的行列式非零，于是有唯一解。从而

$$\Pr[h(x_0) = y_0 \wedge \dots \wedge h(x_{k-1}) = y_{k-1}] = \frac{1}{p^k}$$

由性质 3.4.3， h' 是 k -独立的。 $\square$

定义 3.6. 称 $h: \mathcal{U} \rightarrow [m]$ 是一个全域哈希函数 (universal hash function)，如果 $\forall x_1 \neq x_2$ ，都有 $\Pr[h(x_1) = h(x_2)] = \frac{1}{m}$ 。

性质 3.6.1. 2-独立 哈希函数都是全域哈希函数。

性质 3.4.2，性质 3.4.1 和性质 3.6.1 从某种意义上给出了上述哈希函数独立性的强弱关系。可以认为，选择的哈希函数独立性越强，哈希冲突就越不容易发生。

4 独立链法

独立链法 (separate chaining) 是一种解决哈希冲突的方法。

独立链法对每个哈希值建立一个初始为空的链表。执行 $\text{Insert}(x, v)$ 时，找到 x 对应的哈希值 $y = h(x)$ ，在 y 对应的链表的链首直接插入 (x, v) ；执行 $\text{Modify}(x, v), \text{Delete}(x)$ 与 $\text{Query}(x)$ 时，在 $y = h(x)$ 对应的链表找到键值 x 对应的二元组 (x, v) ，再进行相应的操作。记 y 对应的链表的大小为 c_y ，那么一次操作的访问的链表元素个数不会超过 c_y 。我们考察一次操作访问的访问的链表元素个数的最大值，即 $\max_{t=0}^{m-1} c_t$ 。

定理 4.1. 对全真随机哈希函数 h ，独立链法满足 $\mathbb{E} \left[\max_{t=0}^{m-1} c_t \right] = \Theta \left(\frac{\log n}{\log \log n} \right)$ 。

证明. 记 $llps = \max_{t=0}^{m-1} c_t$ 。

对 $y \in [m]$ ，

$$\Pr[c_y = k] = \binom{n}{k} m^{-n} (m-1)^{n-k} \sim \frac{e^{-\alpha} \alpha^k}{k!} \quad (1)$$

其中 $\alpha = \frac{n}{m}$ 。

记 $e_i(\alpha) = \sum_{k=0}^i \frac{e^{-\alpha} \alpha^k}{k!}$ ， $d_i(\alpha) = 1 - e_i(\alpha) = \sum_{k=i+1}^{\infty} \frac{e^{-\alpha} \alpha^k}{k!}$ 。当 n 足够大时， c_y 可以被看做相互独立的变量，从而有

$$\Pr[llps \leq i] = e_i(\alpha)^m = (1 - d_i(\alpha))^m$$

于是

$$\begin{aligned} -\ln \Pr[llps \leq k] &= -m \ln(1 - d_j(\alpha)) \\ &= m \left(d_j(\alpha) + \frac{d_j(\alpha)^2}{2} + \dots \right) \end{aligned}$$

于是

$$\begin{aligned} -\frac{\ln \Pr[llps \leq k]}{m} &= d_k(\alpha) + O(m^{-2}) \\ &= \frac{e^{-\alpha} \alpha^{k+1}}{(k+1)!} \left(1 + \frac{\alpha}{k+2} + \frac{\alpha^2}{(k+2)(k+3)} + \dots \right) + O(m^{-2}) \end{aligned}$$

即

$$-\ln \Pr[llps \leq k] = m \frac{e^{-\alpha} \alpha^{k+1}}{(k+1)!} + O(k^{-1}) + O(m^{-1})$$

设 $llps$ 的分布的中位数为 j , 则 $\Pr[llps < j] < \frac{1}{2} \leq \Pr[llps \leq j]$ 。

从而

$$\ln 2 = m \frac{e^{-\alpha} \alpha^{j+1}}{(j+1)!} + O(j^{-1}) + O(m^{-1})$$

于是当 $n, m \rightarrow \infty$ 时, 由于 $\alpha < 1$, 有

$$(j+1)! < \frac{m}{e^\alpha \ln 2}$$

$$j < \Gamma^{-1} \left(\frac{m}{e^\alpha \ln 2} \right) - 2 = \Gamma^{-1}(m) + O(1)$$

从而 $j = \Theta(\Gamma^{-1}(m)) = \Theta\left(\frac{\log n}{\log \log n}\right)$ 。

下面我们证明 $\alpha < 1$ 时 $\frac{1}{2} \leq \lim_{m \rightarrow \infty} \frac{E[llps]}{j} \leq 1$ 。当 n, m 充分大时, 一方面,

$$\begin{aligned} E[llps] &= \sum_{k=0}^{\infty} (1 - \Pr[llps \leq k]) \\ &> \sum_{k=0}^j (1 - \Pr[llps \leq k]) \\ &> (j+1)(1 - \Pr[llps \leq j]) \\ &> \frac{j}{2} \end{aligned}$$

另一方面,

$$\begin{aligned}
 E[llps] &< j + \sum_{k=j}^{\infty} (1 - \Pr[llps \leq k]) \\
 &< j - \sum_{k=j}^{\infty} \ln \Pr[llps \leq k] \\
 &= j + \sum_{k=j}^{\infty} \frac{me^{-\alpha} \alpha^{k+1}}{(k+1)!} + O(1) \\
 &< j + \frac{me^{-\alpha} \alpha^{j+1}}{(1-\alpha)(j+1)!} + O(1) \\
 &= j + \frac{\ln 2}{1-\alpha} + O(1)
 \end{aligned}$$

于是 $\frac{1}{2} \leq \lim_{m \rightarrow \infty} \frac{E[llps]}{j} \leq 1$,

$$E[llps] = \Theta\left(\frac{\log n}{\log \log n}\right)$$

□

定理 4.1 表明, 独立链法单次操作的最坏复杂度 (即执行 n 次操作, 其中最慢的一次的复杂度) 期望与中位数均为 $\Theta\left(\frac{\log n}{\log \log n}\right)$, 而并非我们通常认为的 $O(1)$ 。

事实上, 达到这一复杂度并不需要 h 是全真随机的。对于全域哈希函数 h , 式 1 仍然成立, 从而复杂度仍是 $O\left(\frac{\log n}{\log \log n}\right)$ 。

定理 4.2. 对全域哈希函数 h , 独立链法单次操作复杂度为 $O\left(\frac{\log n}{\log \log n}\right)$ 的概率足够高。事实上, 如果 $\alpha = \frac{n}{m}$ 是一个定值, 则对任意 $b > 0$ 均存在 a 使得下式恒成立:

$$\Pr\left[\max_{y \in [m]} c_y > \frac{a \ln n}{\ln \ln n}\right] < n^{-b}$$

证明. 考察 $t \in [m]$ 对应的链表长度 c_t 。

记 $\mu = E[c_t] = \frac{n}{m}$, 则由 Chernoff bound 有

$$\Pr[c_t > (1 + \delta)\mu] < \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}}\right)^\mu$$

取 $\delta = \frac{\lambda \ln n}{\ln \ln n} - 1$, 则

$$\begin{aligned}
 \Pr[c_t > (1 + \delta)\mu] &< \left(\frac{e^{\frac{\lambda \ln n}{\ln \ln n} - 1}}{e^{\frac{\lambda \ln n}{\ln \ln n} \ln\left(\frac{\lambda \ln n}{\ln \ln n}\right)}}\right)^\mu \\
 &< \left(\frac{n^{\frac{\lambda}{\ln \ln n}}}{e^{\lambda \ln n \left(1 + \frac{\ln \lambda - \ln \ln \ln n}{\ln \ln n}\right)}}\right)^\mu \\
 &= n^{\lambda \mu \left(\frac{1 + \ln \ln \ln n - \ln \lambda}{\ln \ln n} - 1\right)}
 \end{aligned}$$

取 $\lambda = \frac{b+1+\epsilon}{\mu}$, 则 $\Pr[c_t > (1 + \delta)\mu] = O(n^{-b-1})$ 。

于是

$$\Pr \left[\max_{y \in [m]} c_y > (1 + \delta)\mu \right] < \sum_{y \in [m]} \Pr[c_y > (1 + \delta)\mu] = O(n^{-b})$$

□

尽管独立链法的单次操作最坏时间复杂度期望并不是 $O(1)$ 的,但是它对于 h 独立性的要求较低,常数较小,且实际应用中询问操作往往可以被看成是随机的,操作次数很少能达到 $\Theta\left(\frac{\log n}{\log \log n}\right)$ 的上界,因此在实际应用中表现较好。C++ 的 STL 容器 `unordered_map` 利用的底层实现也是独立链法。

5 线性探查法

线性探查法 (linear probing) 是一种解决哈希冲突的方法。

线性探查法首先建立一个长为 m 的数组 T , 每个位置存储一对 (x, v) 。执行 $\text{Insert}(x, v)$ 时, 找到 x 对应的哈希值 $y = h(x)$, 如果 T_y 是空的, 就将 (x, v) 存储在 T_y 位置; 如果 T_y 已经被占用了, 扫描 $T_{y+1}, T_{y+2}, \dots, T_{m-1}, T_0, T_1, \dots$, 找到第一个未被占用的位置, 将 (x, v) 存储在这个位置。执行 $\text{Modify}(x, v), \text{Delete}(x)$ 与 $\text{Query}(x)$ 时, 从 T_y 开始向后扫描, 直到找到一个位置的键值等于 x , 对这个位置进行操作。

显然, 对键值 $x \in [u]$, x 将会被存储在 $h(x)$ 及以后的第一个空位置。执行一次关于 x 的操作所用的时间不会超过 $h(x)$ 到下一个的空位置的距离。

为了方便起见, 下文中我们认为 $m \geq \frac{3}{2}n$ 且 m 是 2 的幂。

如果 h 是 k -独立 的, 那么有下表^[1]:

k	2	3	4	5
查询复杂度	$O(\sqrt{n})$	$O(\log n)$	$O(\log n)$	$O(1)$
构造复杂度	$O(n \log n)$	$O(n \log n)$	$O(n)$	$O(n)$

表中, 查询复杂度代表单次查询的最坏复杂度的期望, 即执行 n 次操作, 其中最慢的一次查询操作的时间复杂度的期望; 显然, 单次修改、删除和插入也均不会超过这一复杂度。构造复杂度代表执行 n 次插入操作的总时间复杂度的期望。

下面我们说明对于 5-独立 的哈希函数, 单次操作的最坏复杂度期望是 $O(1)$ 的。

设目前哈希表中已经存储了一个大小为 n 的键值集合 $S \subseteq \mathcal{U}$, 现在要执行一次 $\text{Query}(q)$ 。

定义 5.1. 称一个区间 $R = [l, r]$ 是一个极大连续段, 如果 R 中每个位置都被占用了, 且 $l-1$ 与 $r+1$ 均是空的。

性质 5.1.1. 包含 $h(q)$ 的极大连续段 R 唯一, 且 $\text{Query}(q)$ 访问的位置数不会超过 $|R| + 1$ 。

定义 5.2. 称一个形如 $[i2^l, (i+1)2^l)$ 的区间是一个 l -区间, 其中 $i \in \left[\frac{m}{2^l}\right]$.

定义 5.3. 称一个 l -区间 I 是危险的, 如果 S 中至少有 $\frac{3}{4}2^l$ 个元素对应的哈希值属于 I .

引理 5.4. 对于一个长为 $r \geq 2^{l+2}$ 的极大连续段 R , 与 R 有交的前 4 个 l -区间 中至少有一个是危险的.

证明. 设与 R 有交的前 4 个 l -区间 为 I_0, I_1, I_2, I_3 , 则 I_0 的最后一个元素属于 R , 且 $I_1, I_2, I_3 \subset R$, 于是 $L = R \cap \left(\bigcup_{i=0}^3 I_i\right)$ 满足 $|L| \geq 3 \times 2^l + 1$.

而 L 是 R 的一个前缀, 从而如果 x 被存储在 L 中, 那么 $h(x) \in L$. 从而 $|\{x \in S \mid h(x) \in L\}| \geq |L| \geq 3 \times 2^l + 1$, 因此

$$\sum_{i=0}^3 |\{x \in S \mid h(x) \in I_i\}| \geq 3 \times 2^l + 1$$

从而存在 $i \in [4]$ 使 $|\{x \in S \mid h(x) \in I_i\}| \geq \frac{3}{4}2^l$, 即 I_i 是危险的. $\square$

引理 5.5. 如果包含 q 的极大连续段 R 的长度 r 满足 $2^{l+2} \leq r < 2^{l+3}$, 则以下 12 个 l -区间 中至少有一个区间是危险的: 包含 $h(q)$ 的 l -区间, 它左侧的 8 个 l -区间 以及它右侧的 3 个 l -区间.

证明. 考虑引理 5.4 中的四个区间 I_0, I_1, I_2, I_3 . 由于 $r < 8 \times 2^l$, 故 I_0 一定是包含 $h(q)$ 的 l -区间 和它左侧的 8 个 l -区间 之一. 于是这四个区间包含于引理所述的 12 个区间, 由引理 5.4 这 12 个区间中至少有一个是危险的. $\square$

显然, 每个 l -区间 危险的概率是相等的. 记 P_l 为一个 l -区间 危险的概率.

定理 5.6. $Query(q)$ 的期望复杂度为

$$O\left(1 + \sum_{l=0}^{\log_2 m} 2^l P_l\right)$$

证明. 考虑包含 q 的极大连续段 R , 记 $r = |R|$.

若 $r \geq 4$, 设引理 5.5 中的 12 个区间分别为 $J_0, J_1, \dots, J_{11}$, 那么

$$\begin{aligned} & \Pr[r \in [2^{l+2}, 2^{l+3})] \\ & \leq \Pr[J_0, J_1, \dots, J_{11} \text{ 中至少有一个是危险的}] \\ & \leq \sum_{i=0}^{11} \Pr[J_i \text{ 是危险的}] \\ & = 12P_l \end{aligned}$$

于是

$$\begin{aligned} E[r] &< \sum_{l=0}^{\log_2 m} 2^{l+1} \Pr[r \in [2^l, 2^{l+1})] \\ &= 3 + \sum_{l=0}^{\log_2 m} 2^{l+3} \Pr[r \in [2^{l+2}, 2^{l+3})] \\ &\leq 3 + \sum_{l=0}^{\log_2 m} 2^{l+3} \times 12P_l \end{aligned}$$

从而

$$E[r] = O\left(1 + \sum_{l=0}^{\log_2 m} 2^l P_l\right) \quad (2)$$

□

引理 5.7. 如果随机变量 $X_0, X_1, \dots, X_{n-1} \in \{0, 1\}$ 是 4-独立的, $X = \sum_{i=0}^{n-1} X_i$, $\mu = E[X] \geq 1$, 则

$$\Pr[|X - \mu| \geq d\sqrt{\mu}] \leq \frac{4}{d^4}$$

证明. 记 $p_i = E[X_i] = \Pr[X_i = 1]$, 则 $\mu = \sum_{i=0}^{n-1} p_i$, 于是 $X - \mu = \sum_{i=0}^{n-1} (X_i - p_i)$.

记 $\sigma_i^2 = \text{Var}[X_i] = E[(X_i - p_i)^2] = p_i(1 - p_i)^2 + (1 - p_i)p_i^2 = p_i - p_i^2$.

记 $\sigma^2 = \text{Var}[X]$. 由于 $X_0, X_1, \dots, X_{n-1}$ 是 4-独立的, 进而是 2-独立的, 从而有 $\sigma^2 = \text{Var}[X] = \sum_{i \in [n]} \text{Var}[X_i] = \mu - \sum_{i \in [n]} p_i^2 \leq \mu$.

下面我们估计 $E[(X - \mu)^4]$.

$$E[(X - \mu)^4] = E\left[\left(\sum_{i=0}^{n-1} (X_i - p_i)\right)^4\right] = \sum_{i, j, k, l \in [n]} E[(X_i - p_i)(X_j - p_j)(X_k - p_k)(X_l - p_l)] \quad (3)$$

考虑其中一项 $E[(X_i - p_i)(X_j - p_j)(X_k - p_k)(X_l - p_l)]$. 由于 $X_0, X_1, \dots, X_n$ 是 4-独立的, 于是如果 i, j, k, l 存在一项与另外三项均不同 (不妨设 i 与 j, k, l 均不同), 就有

$$E[(X_i - p_i)(X_j - p_j)(X_k - p_k)(X_l - p_l)] = E[X_i - p_i] E[(X_j - p_j)(X_k - p_k)(X_l - p_l)]$$

而 $E[X_i - p_i] = 0$. 从而计算式 3 时, 只需要计算 i, j, k, l 四个下标都至少出现两次的项, 即 i, j, k, l 全部相同或者可以分成两组相同下标的项.

于是

$$\begin{aligned}
 E[(X - \mu)^4] &= \sum_{i \in [n]} E[(X_i - p_i)^4] + \binom{4}{2} \sum_{0 \leq i < j < n} E[(X_i - p_i)^2 (X_j - p_j)^2] \\
 &\leq \sum_{i \in [n]} E[(X_i - p_i)^2] + 6 \sum_{0 \leq i < j < n} E[(X_i - p_i)^2] E[(X_j - p_j)^2] \\
 &= \sum_{i \in [n]} \sigma_i^2 + 6 \sum_{0 \leq i < j < n} \sigma_i^2 \sigma_j^2 \\
 &\leq \sigma^2 + 3 \left(\sum_{i \in [n]} \sigma_i^2 \right)^2 \\
 &= \sigma^2 + 3\sigma^4 \\
 &\leq \mu + 3\mu^2 \\
 &\leq 4\mu^2
 \end{aligned}$$

由 Markov inequality,

$$\Pr[|X - \mu| \geq d\sqrt{\mu}] = \Pr[(X - \mu)^4 \geq d^4 \mu^2] \leq \frac{E[(X - \mu)^4]}{d^4 \mu^2} \leq \frac{4}{d^4}$$

□

定理 5.8. $m \geq \frac{3}{2}n$ 且 m 是 2 的幂时, 如果选取一个 5-独立的哈希函数 $h: \mathcal{U} \rightarrow [m]$, 那么线性探查法单次操作的最坏复杂度期望是 $O(1)$ 。

证明. 考虑固定 q 对应的哈希值 $h(q) = y_q$, 那么对于不同的四个键值 $x_0, x_1, x_2, x_3 \in \mathcal{U} \setminus \{q\}$ 与四个哈希值 $y_0, y_1, y_2, y_3 \in [m]$, 有

$$\Pr\left[\bigwedge_{i=0}^3 h(x_i) = y_i \mid h(q) = y_q\right] = \frac{\Pr\left[h(q) = y_q \wedge \left(\bigwedge_{i=0}^3 h(x_i) = y_i\right)\right]}{\Pr[h(q) = y_q]} = m^{-4}$$

从而 S 中所有键值对应的哈希值是 4-独立的。

由定理 5.6, 为了估计一次操作的复杂度, 我们尝试给出 P_l 的一个上界。

对于一个 l -区间 I , 考虑它危险的概率。对 $x \in S$, 记 $X_x = [h(x) \in I]$ 。于是, $X = \sum_{x \in S} X_x$ 即为 S 中哈希值落在 I 中的元素个数, I 危险当且仅当 $X \geq \frac{3}{4}2^l$ 。

记 $\mu = E[X]$, 则 $\mu = \frac{n2^l}{m} \leq \frac{2}{3}2^l$, 于是

$$X \geq \frac{3}{4}2^l \implies X - \mu > \frac{1}{12}2^l > \frac{1}{10}\sqrt{2^l\mu}$$

由引理 5.7,

$$\Pr\left[X \geq \frac{3}{4}2^l\right] \leq \Pr\left[X - \mu \geq \frac{1}{10}\sqrt{2^l\mu}\right] \leq \frac{40000}{2^{2l}}$$

从而由定理 5.6, 单次操作的最坏复杂度期望是

$$O\left(1 + \sum_{l=0}^{\log_2 m} 2^l P_l\right) = O\left(1 + \sum_{l=0}^{\log_2 m} 2^l \times 2^{-2l}\right) = O(1)$$

□

推论 5.9. $m \geq 3n$ 时, 选取一个 5-独立 的哈希函数 $h: \mathcal{U} \rightarrow [m]$, 线性探查法单次操作的最坏复杂度期望是 $O(1)$ 。

事实上, 对于 $m = (1 + \epsilon)n$ 和 5-独立 的哈希函数 h , 可以证明线性探查法单次操作的最坏复杂度期望是 $O(\epsilon^{-2})^{[2]}$ 。

6 总结

本文对哈希表的两种解决哈希冲突的实现方式——独立链法与线性探查法做了简要的介绍, 并证明了独立链法对于全域哈希函数的单次操作最坏复杂度期望为 $\Theta\left(\frac{\log n}{\log \log n}\right)$, 线性探查法对 5-独立 哈希函数的单次操作最坏复杂度期望为 $O(1)$ 。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢父母对我多年的培养与教导。

感谢叶金毅老师、谷多玉老师对我的培养与教导。

感谢邓明扬学长、许庭强同学、向昱帆同学为本文验稿。

参考文献

- [1] Mikkel Thorup, (2017). Linear Probing with 5-Independent Hashing.
- [2] Mihai Pătraşcu and Mikkel Thorup, (2012). The power of simple tabulation-based hashing. Journal of the ACM.
- [3] G. H. Gönnet, (1981). Expected length of the longest probe sequence in hash code searching, Journal of the ACM.



线性代数在OI中的应用

重庆市巴蜀中学 常瑞年

摘要

本文比较全面地概括了OI中的线性代数方法，并延伸了一些线性代数的用途。

1 前言

近年来，线性代数在OI中出现次数逐渐增多，考察方面以行列式，线性基等基础方面为主，但是线性代数远不止这些，通过使用线性代数方法我们可以得出许多有趣的结论并解决一些崭新的问题。本文将从这些方面阐述线性代数的作用。

2 符号与约定

假定读者已经掌握基础的线性代数知识，多项式的基本运算与基本的组合计数，随机过程知识。

如不特殊说明，本文均在一个域下展开讨论，大多数时候假设域中矩阵存在相应的特征值。

对于矩阵 A ，记 $A[S, T]$ 为一个矩阵，且满足 $A[S, T]_{i,j} = A_{S_i, T_j} \ 1 \leq i \leq |S|, 1 \leq j \leq |T|$ ，其中 S, T 是两个序列，大多数情况下 S, T 为集合，其中元素升序排列。

对于矩阵 A ，记 $A[S] = A[S, S]$ 。

记 A^* 为 A 的伴随矩阵。

记 $|A|$ 为 A 的行列式，若 A 是矩阵。记 $|X|$ 为 X 的模，若 X 为向量。

记 R_A, C_A 为矩阵 A 的行，列的集合，在不产生混淆的情况下，下标 A 可以被忽略，并且可以用 $*$ 代替对应的全集。

定义 O, I 分别为零矩阵和单位矩阵，如果限制大小则会在右下角标出。

约定 $[A]$ 为艾弗森括号，其中命题 A 为真则 $[A] = 1$ 否则 $[A] = 0$ 。



3 基础

3.1 行列式相关

我们先来介绍一些有关矩阵的行列式的恒等式,以便使用。

定理 3.1 (广义拉普拉斯展开). 对于方阵 A , 以及 $I \subseteq R$, 我们有

$$|A| = \sum_{J \subseteq C, |J|=|I|} \text{sgn}(I, J) |A[I, J]| \cdot |A[R \setminus I, C \setminus J]|$$

这里 $\text{sgn}(I, J) = (-1)^{i_1+i_2+\dots+i_{|I|}+j_1+j_2+\dots+j_{|J|}}$, i_t, j_t 分别为 I, J 的所有元素。

证明. 按照行列式的定义可证。 □

定理 3.2 (格拉斯曼恒等式). 令 A 是一个满足 $|R| \leq |C|$ 的矩阵, 对于 $J, J' \subseteq C$ 且 $|J| = |J'| = |R|$ 以及一个 $i \in J \setminus J'$, 则有

$$|A[R, J]| \cdot |A[R, J']| = \sum_{j \in J' \setminus J} |A[R, J - i + j]| \cdot |A[R, J' + i - j]|$$

这里 $J - i + j$ 代表 $J \setminus \{i\} \cup \{j\}$, 并附有位置关系, 表示把第 j 列替换第 i 列的位置, 类似的我们有 $J' + i - j$ 的意义, 之后我们会沿用这种记号。

证明. 考虑分块矩阵

$$\tilde{A} = \begin{bmatrix} A[R, J] & A[R, J'] \\ A' & A[R, J'] \end{bmatrix}$$

其中 A' 为仅第 k 列为 $A[R, \{i\}]$ 其余列全为 0 的矩阵, 这里 k 为 i 在 J 中的位置。

那么考虑 $\tilde{A}$ 的行列式, 其显然为 0, 我们对后 $|R|$ 行展开即可证得。 □

定理 3.3 (Cauchy-Binet). 对于 $n \times m$, $m \times n$ 的矩阵 A, B , 其中 $n \leq m$, 有

$$|AB| = \sum_{|S|=n, S \subseteq C_A=R_B} |A[R_A, S]| \cdot |B[S, C_B]|$$

证明. 考虑分块矩阵

$$\begin{bmatrix} O_n & A \\ B & I_m \end{bmatrix}$$

对前 $|R_A|$ 行展开得到右式带一些符号, 对这个矩阵初等行变换即可得左式, 消去符号即证得 □

定理 3.4 (舒尔补). 如果 A 是方阵, D 非奇异, 则

$$\det \begin{bmatrix} A & B \\ C & D \end{bmatrix} = |D| \cdot |A - BD^{-1}C|$$

证明. 对矩阵初等行变换即得。 □

3.2 秩相关

矩阵的秩也是重要的变量, 我们来考察一些有关矩阵的秩的恒(不)等式。

定理 3.5.

$$\text{rank } AB \leq \min(\text{rank } A, \text{rank } B)$$

$$\text{rank } AB \geq \text{rank } A + \text{rank } B - |C_A|$$

证明. 考虑有关线性映射的一些定理会非常简单。 □

推论 3.1. 对于满秩方阵 A , 矩阵 B , A 左乘 B 不改变 B 的秩

推论 3.2. 如果 A 是方阵, D 非奇异, 则

$$\text{rank} \begin{bmatrix} A & B \\ C & D \end{bmatrix} = |R_D| + \text{rank}(A - BD^{-1}C)$$

定理 3.6.

$$\text{rank } A = \max\{|I| \mid |A[I, J]| \neq 0, I \subseteq R, J \subseteq C\}$$

其中 $|A[I, J]| \neq 0$ 表示 $A[I, J]$ 非奇异

证明. 先找行极大无关组, 在再这个子矩阵上找列极大无关组即可。 □

这建立了秩到非奇异子阵的关系, 让我们可以使用行列式相关的恒等式去证明秩相关的不等式。

另外, 我们还有以下三者互相等价:

- A 是方阵, A 的行极大无关组是 R
- A 可逆
- A 非奇异

以及计算以下四者的时间复杂度复杂度一致:

- A 的秩

- 若 A 可逆, A 的逆
- 若 A 非奇异, A 的行列式
- 方阵 A, B 的乘法

我们将这四者的最优复杂度记为 $O(n^\omega)^1$, 而我们一般使用 $O(n^3)$ 的算法计算。

3.3 广义矩阵

接下来我们引入广义矩阵和广义秩的基础知识以方便我们介绍带未定元的矩阵 (尤其是图特矩阵)。

定义 3.1 (代数独立). 令数域 $\mathbf{K}$, $\mathbf{F}$ 是 $\mathbf{K}$ 的扩域, 令 $\mathcal{Y} = \{y_i\}$ 是一簇 q 个 $\mathbf{F}$ 上的元素, 若不存在数域 $\mathbf{K}$ 上的有 q 个未定元的多项式 p , 使得 $p(y_1, y_2, \dots, y_q) = 0$, 则称 $\mathcal{Y}$ 在 $\mathbf{K}$ 上代数独立

定义 3.2 (广义秩). 当矩阵 A 的元素中有未定元时, 我们依然定义最大的非奇异子式的大小为 A 的广义秩, 其还等价于选择一个扩域 $\mathbf{F}$, 在其上选择一簇合适的 a 并将其代入 A 求得矩阵的秩的最大值。这里合适的 a 是指在 A 的域 $\mathbf{K}$ 下代数独立。

3.4 Jordan 标准型

设 n 阶 Jordan 块 $J_n(a)$ 为:

$$\begin{pmatrix} a & 1 & 0 & \cdots & 0 & 0 \\ 0 & a & 1 & \cdots & 0 & 0 \\ 0 & 0 & a & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a & 1 \\ 0 & 0 & 0 & \cdots & 0 & a \end{pmatrix}$$

定义 3.3. 矩阵的 Jordan 标准型 对于 n 阶矩阵 A , 不妨设其特征值分别为 λ_i , 则 A 相似于 $\text{diag}\{J_{k_i}(\lambda_i)\}$, 对于一根 λ_i , 其总共出现其在特征多项式中的重数次。

其中 Jordan 块的确定以及其证明比较复杂, 此处不证, 请读者参考任意一本线性代数教材获得证明。

通过这种形式, 简单计算可得:

推论 3.3.

¹目前 $\omega \approx 2.3727$

- $\text{trace}(A^k) = \sum_{i=1}^{n-1} \lambda_i^k$
- $|\exp A| = \exp \text{trace}(A)$
- 设多项式 f , 则若 A 的特征根为 $\{\lambda_i\}$, 则 $f(A)$ 的特征根为 $\{f(\lambda_i)\}$ 。

4 带修改的矩阵

4.1 伴随矩阵

定义 4.1 (代数余子式). 对于 n 阶方阵 A , 定义余子式 $M_{i,j}$ 为 $|A[R \setminus \{i\}, C \setminus \{j\}]|$, 定义代数余子式 $C_{i,j} = (-1)^{i+j} M_{i,j}$ 。

性质 4.1.

$$\sum_{j=1}^n a_{i,j} C_{k,j} = |A| [i = k]$$

定义 4.2 (伴随矩阵). 对于 n 阶方阵 A , 我们定义其伴随矩阵 C 为:

$$A_{i,j}^* = C_{j,i} (1 \leq i, j \leq n)$$

性质 4.2. $AA^* = A^*A = |A|I$

由于代数余子式是行列式对某元素的导数, 伴随矩阵可以用于处理大量对矩阵暂时修改动态求逆的问题。

接下来我们考虑求伴随矩阵, 分三类讨论:

如果矩阵满秩, $A^* = |A|A^{-1}$ 。

如果 $\text{rank } A \leq n-2$, 那么任意余子式都为 0, 故 A^* 为零矩阵。

对于 $\text{rank } A = n-1$ 的情形比较复杂, 考虑引理:

引理 4.1. 对于 $m \times n$ 的矩阵 A , $n \times k$ 的矩阵 B , 若 $AB = 0$, 则 $\text{rank } A + \text{rank } B \leq n$ 。

证明依旧考虑将 A 看做线性映射。

这样, 由于 $A^* \neq 0$, $AA^* = 0$, 则 $\text{rank } A^* = 1$, 我们知道存在向量 u, v 使得 $A^* = uv^T$ 。

由代数余子式的性质, 我们有 $Auv^T = (Au)v^T = 0$, 可得 $Au = 0$ (因为 $v^T \neq 0$)。

同理可得 $v^T A = 0$, 不妨随便解得一组 u', v' 使 $u', v' \neq 0$ 。

由于 $u' = k_1 u, v' = k_2 v$, 则存在一个 k 使得 $u'v'^T = kuv^T$ 。

这时我们只要求出任意一个 $u'v'^T$ 不为 0 的位置上的代数余子式进行比较求出 k 即可。

4.2 舍曼-莫里斯公式

舍曼-莫里斯公式(以下简称SMW公式),常用于矩阵的部分修改,如低秩修改,子矩阵修改,动态求逆等。

定理 4.1. SMW公式对于 n 阶可逆矩阵 A , $n \times m$ 矩阵 U , $m \times n$ 矩阵 V , $m \times m$ 可逆矩阵 C , 有:

$$(A - UC^{-1}V)^{-1} = A^{-1} + A^{-1}U(C - VA^{-1}U)^{-1}VA^{-1}$$

并且 $A - UC^{-1}V$ 可逆的充要条件为 $C - VA^{-1}U$ 可逆。

证明. 注意到

$$\begin{aligned} \begin{bmatrix} A & U \\ V & C \end{bmatrix} &= \begin{bmatrix} I_n & UC^{-1} \\ O & I_m \end{bmatrix} \begin{bmatrix} A - UC^{-1}V & O \\ O & C \end{bmatrix} \begin{bmatrix} I_n & O \\ C^{-1}V & I_m \end{bmatrix} \\ &= \begin{bmatrix} I_n & O \\ -VA^{-1} & I_m \end{bmatrix}^{-1} \begin{bmatrix} A & O \\ O & C - VA^{-1}U \end{bmatrix} \begin{bmatrix} I_n & -A^{-1}U \\ O & I_m \end{bmatrix}^{-1} \end{aligned}$$

两端取逆即证得(这里是指第二个等号连接的两端,注意到我们已经转化出了 $A - UC^{-1}V$)。 $\square$

将SMW公式进行一些变换我们就能得到以下常用的结果:

推论 4.1 (删行列求逆). 对 $n+1$ 阶可逆矩阵 M , 设其 n 阶顺序主子矩阵 A , 则 A 有逆当且仅当 $M_{n,n} \neq 0$, 且 A 的逆为

$$M^{-1} - M^{-1}[* , \{n+1\}]M^{-1}[\{n+1\}, *](M_{n,n}^{-1})^{-1}$$

的 n 阶顺序主子矩阵。

证明. 不妨设 $M[* , \{n+1\}] = (u^T \ t)^T$, 其中 t 为数域 $\mathbf{K}$ 的元素, u 为 n 阶列向量, 由于空间原因这里使用行向量的转置来表示。考虑令 $U = (u^T \ t - 1)^T$, $C = 1$, $m = 1$, $V = (0 \ 1)$, 代入SMW公式, 并注意到 $M^{-1}M[* , \{n+1\}]$ 的关系即可证得。 $\square$

推论 4.2 (添行列求逆). 对 n 阶可逆矩阵 A , 设其在末尾增加一行一列得到矩阵 M , 由于直接操作太过复杂, 我们不妨先为其添加一行, 变成形如:

$$\begin{bmatrix} A & O \\ v^T & 1 \end{bmatrix}$$

再为其添加一列, 变为:

$$\begin{bmatrix} A & u \\ v^T & t \end{bmatrix}$$

用SMW公式更新即可

推论 4.3 (小区域更新). 不妨设可逆矩阵 M 的逆矩阵为 N , $\tilde{M}$ 相较于 M 仅有主子矩阵 $M[S]$ 发生了变化, 不妨设变化为 $\Delta_{S,S}$ 是一个大小为 S 的方阵, 其中行列的集合均为 S .

则 $\tilde{M}$ 可逆当且仅当 $I + \Delta_{S,S}N[S]$ 可逆, 且其逆为:

$$\tilde{M}^{-1} = N - N[*, S](I + \Delta_{S,S}N[S])^{-1}\Delta_{S,S}N[S, *]$$

证明. 考虑在 SMW 中取 $m = |S|$, $C = I_m$, $U_{i,j} = [i \in S]$, $V_{i,j} = [j \in S]\Delta_{i,j}$ 即证得. $\square$

特别地, 考虑在小区域更新公式中截取 S , 则:

$$\tilde{M}^{-1}[S] = N[S](I + \Delta_{S,S}N[S])^{-1}$$

这样的更新可以在 $O(|S|^{\omega})$ 时间内被算出。

5 图论相关

5.1 图上的两个重要线性空间

不妨设图 $G(V, E)$, $|V_G| = n$, $|E_G| = m$ 。

定义 5.1.

- 环空间 (余树空间), 将边集的子集视为 m 维的 01 向量, 加法是对称差, 由所有欧拉子集 (即使得每个点的度数都为偶数的边集) 组成。
- 割集空间 (树空间), 将边集的子集视为 m 维的 01 向量, 加法是对称差, 由所有割集 (即选出一个点子集 S , S 与 $V \setminus S$ 之间的边集) 组成。

容易发现以下重要的性质

性质 5.1.

- 环空间的某一组基为, 随便找一丛生成森林, 每一条非树边与其上树边组成的基础环。
- 割集空间的某一组基为, 随便找一丛生成森林, 每一条树边与跨过这条树边的非树边组成的基础割集。

由此, 有推论:

推论 5.1.



- 图 $G(V, E)$ 上的欧拉子集数量是 $2^{|E|-|V|+C}$, 割集数量是 $2^{|V|-C}$, 其中 C 是连通块数
- 环空间是割集空间的正交补 (任一环交任一割集大小为偶数)

接下来我们举一个经典的例子:

例题 1 (DZY loves chinese).

给一张图 $G(V, E)$, 每次询问一个大小不超过 15 的边集, 需要判断是否有一个子集是割。

解法 1.

把非树边看做很多个 F_2 上的代数独立的未定元, 令树边的权值为跨过这条树边的不定元的和, 那么这些边里有割当且仅当线性相关。

这是因为若把在这样的权值定义下子集的和看做是 $2^E \rightarrow F_2^k$ 的映射, 首先零化度是 $n-1$, 其次割空间是其零空间的子空间, 则割空间恰为其零空间。

考虑给未定元在 F_2^n 上随机赋权, k 个本不线性相关的元素随机赋值之后线性相关的概率至多为 $\binom{2^{k-1}}{k} \binom{n}{k-1}_2 / \binom{2^n}{k}$ 。

而这在 $n = 64, k = 15$ 的情形下仅有 3×10^{-15} 左右, 可以接受 (使用子空间反演可以更准确地估计)。

5.2 匹配相关

线性代数在图论相关的部分中最广泛的应用当属解决匹配问题, 究其原因是因为代数方法可以解决线性拟阵奇偶问题, 我们后面会讲到。

定义 5.2 (图特矩阵). 首先有一张图 $G(V, E)$, 我们定义一个矩阵 $T(G)$, 其行列集合 $R = C = V$, 且满足:

$$T_{i,j} = \begin{cases} x_{i,j} & i < j \wedge (i, j) \in E \\ 0 & i = j \vee (i, j) \notin E \\ -T_{j,i} & i > j \end{cases}$$

则称 $T(G)$ 为 G 的图特矩阵 (又称 Tutte 矩阵), G 为 $T(G)$ 的基图, 在不发生混淆的情形下我们把 $T(G)$ 简记为 T , 并且后文的 T 总是指某一个图特矩阵

图特矩阵建立了代数到图的联系:

定理 5.1. $\text{rank } T = 2\nu(G)$, 其中 $\nu(G)$ 为 G 的最大匹配

证明可以参考 2017 年的候选队论文集。

注意到图特矩阵是斜对称矩阵, 为了更加深入的研究图特矩阵的性质, 我们介绍一些与斜对称矩阵有关的东西:

定理 5.2.

- 斜对称矩阵的秩为偶数
- 令斜对称矩阵 K , $\text{rank } K = \max\{|S| \mid |K[S]| \neq 0\}$

以上两条的证明比较简单, 此处略去。

定义 5.3. Pfaffian 定义一个 $2n \times 2n$ 的斜对称矩阵 K 的 Pfaffian 为:

$$\text{Pf } K = \frac{1}{2^n n!} \epsilon_{i_1, j_1, i_2, j_2, \dots, i_n, j_n} K_{i_1, j_1} K_{i_2, j_2} \dots K_{i_n, j_n} = \sum_P \text{sgn}(P) \prod_{i=1}^n K_{P_{2i-1}, P_{2i}}$$

其中 ϵ 为勒维符号, 可以视作是排列的符号 sgn 并省略 $\sum$, 最后一个等式中的 P 取遍 $1 \sim 2n$ 的排列, 并需要满足 $P_{2i-1} < P_{2i}$, $P_{2i-1} < P_{2i+1}$, 可以视作取遍所有匹配。

Pfaffian 也有一些重要性质:

性质 5.2.

- $\text{Pf}(BMB^T) = \text{Pf}(M)|B|$
- $|M| = \text{Pf}(M)^2$

我们先来证明第一条性质:

证明.

$$\text{Pf } M = \frac{1}{2^n n!} \epsilon_{i_1, j_1, i_2, j_2, \dots, i_n, j_n} (B_{i_1, k_1} B_{j_1, l_1}) (B_{i_2, k_2} B_{j_2, l_2}) \dots (B_{i_n, k_n} B_{j_n, l_n}) M_{k_1, l_1} M_{k_2, l_2} \dots M_{k_n, l_n}$$

这里省略枚举符号, i, j 枚举每一个排列, k, l 枚举所有 $1 \sim 2n$ 的元素。

注意到

$$\det B \epsilon_{k_1, l_1, k_2, l_2, \dots, k_n, l_n} = \epsilon_{i_1, j_1, i_2, j_2, \dots, i_n, j_n} B_{i_1, k_1} B_{j_1, l_1} B_{i_2, k_2} B_{j_2, l_2} \dots B_{i_n, k_n} B_{j_n, l_n}$$

这里 $\{k\} \cup \{l\}$ 构成排列, 这是由于 $\{k\}, \{l\}$ 若有相重, 我们可以在右式将第一次相重的两个 i 或者 j 对应的数交换, 其不改变 B 乘积的值却改变符号, 又这是一个双射, 故这样的和为 0。

所以我们其实证得了 k, l 都是排列, 将 (2) 代入 (1) 并整理即得证。□

对于第二条性质, 类似的方法可以证明, 我们将细节留给读者自证。

由以上性质我们可以轻松地求出一个数值矩阵的 Pfaffian, 但若我们先求出行列式再开方, 其是在大多数情形下都不被允许的。

注意到第二条性质类似于一种合同变换, 我们不妨结合斜对称矩阵的性质进行消元。

消元的过程最终会得到由很多个二阶斜对称矩阵组成的对角形, 因此我们只需要叙述如何消出第一个二阶斜对称性, 然后归纳即可。

消元过程 1.

不妨设矩阵为 A 。

如果第一列无非 0 元素，则由斜对称性，第一行全为 0，因此矩阵不满秩， $\text{Pf}(A) = 0$ 。

否则找到一个第一列元素非 0 的行 i ，将其交换到第二行，视作初等矩阵 B ，由于 B^T 的存在，对应地将第 i 列交换到第二列。

此时对于剩下的第一列元素非 0 的行 j ，我们用第二行将其首位元素消去，同时由于需要做对称的变换，我们用第二列将第 j 列第一个元素消去，由斜对称性这可以做到。

此时还会有第二行和第二列的某些元素存在，对于第二列的非 0 元素（除第一二行外），我们使用第一行的第二个元素将其消去，对称地，我们消去第二行的元素。

以上操作均不改变斜对称性，由第二条性质，我们作用的 B 均初等，故不改变 Pf 值，接着我们继续在右下角阶减少二的矩阵上消元直到消成对角形可以简单求值。

类似使用 $\det$ 替代 permutant 的作用，我们也可以使用图特矩阵上的 Pf 代替枚举匹配，经典地我们对图特矩阵随机赋值，由施瓦兹-锡伯引理我们可以以极大的正确率检查匹配的某种相关的存在性。

如 CodeChef 中 PRECPAIR 一题，首先将权值转化到多项式的次数，再对矩阵随机赋值求 Pf ，最后插值即可。

同 $\det$ 一样， Pf 也有展开式：

公式 1.

$$\text{Pf}(A) = \sum_{i \neq 1} (-1)^i A_{1,i} \text{Pf}(A \setminus \{1, i\})$$

其中 $A \setminus \{1, i\}$ 表示 A 去掉第 1, i 行列

证明类似 $\det$ 的展开，留给读者自己思考。

接下来我们将介绍本文的一个重点：如何用图特矩阵导出图特-伯格定理。

定理 5.3 (图特-伯格定理的矩阵形式加强). 对于图特矩阵 $A(G)$ ，不妨设其行列集合一致，以及一对行和列的子集 I, J ，则有

$$\text{rank } A[I, J] = \min\{|I \setminus I'| + |J \setminus J'| + |I' \cap J'| - \text{odd}(G[I' \cap J'])\} \mid (I', J') \in D(I, J)$$

这里 $D(I, J) = \{(I', J') \mid A[I' \setminus J', J'] = O, A[I', J' \setminus I'] = O, I' \subseteq I, J' \subseteq J\}$ ， $\text{odd}(G)$ 表示图 G 中的奇连通块的数目， $G[S]$ 表示子集 S 的导出子图。

由这条定理我们可以得到图论中的图特-伯格定理，于是我们需要证明这条定理。

在证明定理之前，我们还需要准备一些引理和定义。

引理 5.1.

对于矩阵 A ，行集合的子集 I, I' ，有：

$$\text{rank } A[I, *] + \text{rank } A[I', *] \geq \text{rank } A[I \cap I', *] + \text{rank } A[I \cup I', *]$$

引理 5.1 的证明.

设 X 是 $A[I \cap I', *]$ 的一组行基, 设 Y 是由 X 在 I 中扩充得到的一组基, 同理设 Z 是由 X 在 I' 中扩充得到的基, 则:

$$\text{rank } A[I, *] + \text{rank } A[I', *] = |Y| + |Z| = |X| + (|Y| + |Z| - |X|) \geq \text{rank } A[I \cap I'] + \text{rank } A[I \cup I']$$

□

引理 5.2.

对于矩阵 A , 行集合的子集 I, I' , 列集合的子集 J, J' , 有:

$$\text{rank } A[I, J] + \text{rank } A[I', J'] \geq \text{rank } A[I \cap I', J \cup J'] + \text{rank } A[I \cup I', J \cap J']$$

引理 5.2 的证明.

首先不妨设 A 的行列集合 S, T 不交, 令 $T' = T \cup S$, 令 $A'[S, S] = I, A'[S, T] = A$, 则 $\text{rank } A[I, J] = \text{rank } A'[*], (S \setminus I) \cup J - |S| + |I|$, 则有:

$$\text{rank } A[I, J] + \text{rank } A[I', J'] = \text{rank } A'[*], (S \setminus I) \cup J + \text{rank } A'[*], (S \setminus I') \cup J' - |S| + |I \cap I'| - |S| + |I \cup I'|$$

由引理 5.1, $\text{rank } A'[*], (S \setminus I) \cup J + \text{rank } A'[*], (S \setminus I') \cup J' \geq \text{rank } A'[*], (S \setminus (I \cup I')) \cup (J \cap J') + \text{rank } A'[*], (S \setminus (I \cap I')) \cup (J \cup J')$

将其代入上面的等式即证

□

引理 5.3.

对于行列集合分别为 S, T 的不满秩矩阵 A , 存在非空子集 $I \subseteq S, J \subseteq T$ 使得:

$$|S| + |T| - \text{rank } A[S, T] = |I| + |J| - \text{rank } A[I, J]$$

$$\forall i \in I, j \in J, \text{rank } A[I, J] = \text{rank } A[I \setminus i, J \setminus j]$$

引理 5.3 的证明.

首先满足第一条的 I, J 是存在的, 因为取 $(I, J) = (S, T)$ 即可。

不妨设 (I, J) 是满足这样的条件中 $|I| + |J|$ 最小的某一对, 设 $\text{rank } A[I, J] = K$, 下面我们反证 (I, J) 满足第二条。

不妨设 (I, J) 不满足第二条, 则由引理 5.2 我们有:

$$2K > \text{rank } A[I \setminus i, J] + \text{rank } A[I, J \setminus j] \geq \text{rank } A[I, J] + \text{rank } A[I \setminus i, J \setminus j] \geq 2K - 2$$

故 $\text{rank } A[I \setminus i, J], \text{rank } A[I, J \setminus j]$ 中必有一个为 $K - 1$, 若对应的矩阵不为空, 则得到一组更小的 (I, J) , 矛盾。

否则对应的矩阵为空, 也就是 $|I|$ 或者 $|J|$ 为 1, 且 $K = 1$, 那么我们有 $|S| + |T| - \text{rank } A[S, T] = |I| + |J| - 1 \leq \max(|S|, |T|)$

可是 $|S| + |T| - \text{rank } A[S, T] > |S| + |T| - \min(|S|, |T|) = \max(|S|, |T|)$, 矛盾, 故命题得证。

□

引理 5.4.

若连通图 $G(V, E)$ 中删去任意一个点后最大匹配不变, 则这张图由奇数个点组成, 并且删去任意一个点都有完美匹配。

引理 5.4 的证明.

首先我们把删去后最大匹配不变的点称为这张图的质环, 也就是说题设的图中的点全为质环, 另外我们把删去后最大匹配变小的一对点称作是串联的。

那么串联显然是可迁的, 即所有包含 $(x, y), (y, z)$ 的最大匹配一定包含 (x, z) 。

由于图上相邻的质环串联, 则 G 上任意两点串联, 则最大匹配一定恰不能覆盖一个点 (否则多于一个, 其中两点不串联), 因此证毕。

□

接下来我们开始证明原定理

图特-伯格定理矩阵形式加强的证明.

首先我们有 $\text{rank } A[I, J] \leq |I \setminus I'| + \text{rank } A[I', J] \leq |I \setminus I'| + |J \setminus J'| + \text{rank } A[I' \cap J']$

又 $A[I' \cap J']$ 斜对称, 所以 $\text{rank } A[I' \cap J'] \leq |I' \cap J'| - \text{odd}(G[I' \cap J'])$, 即左边不大于右边。

接下来, 若 $A[I, J]$ 满秩, 取 $I' = \emptyset$ 或者 $J' = \emptyset$ 可得左右两边相等, 否则由引理 5.3 可以取 (I', J') 满足引理的条件。

我们证明 (I', J') 满足条件且取到最小值, 那么首先需要说明 $(I', J') \subseteq D(I, J)$ 。

若不然, 不妨设存在非 0 元素 (i, j) 使得 $A[i, j] \neq 0, i \in I', j \in J' \setminus I'$

先将 i 从 I' 中删去, 并取极大无关组, 由于 $A[i, j]$ 与剩下元素代数独立, 则重新加入第 i 行后秩增大, 与引理的条件矛盾。

由引理 5.3 的条件及引理 5.4 可得 $\text{rank } A[I' \cap J'] = |I' \cap J'| - \text{odd}(G[I' \cap J'])$, 那么由以上可得:

$$\text{rank } A[I, J] = |I| + |J| - |I'| - |J'| + \text{rank } A[I' \cap J'] = |I \setminus I'| + |J \setminus J'| + |I' \cap J'| - \text{odd}(G[I' \cap J'])$$

, 得证。

□

因此我们得到图特-伯格定理:

定理 5.4 (图特-伯格定理).

对于图 $G(V, E)$, 我们有:

$$2\nu(G) = \min_{S \subseteq V} \{2|V| - |S| - \text{odd}(G[S])\}$$

6 随机过程及其衍生

随机过程是OI中经常出现的考察点,但是目前并没有特别规范的引入,类似彭博同学的鞅与停时就是随机过程的一部分,而与线性代数比较相关的随机过程就有马尔科夫链。

6.1 马尔可夫链与图上随机游走

定义 6.1 (随机矩阵 (行)). $\forall i \in [1, n], \sum_{j=1}^n P_{i,j} = 1$

定义 6.2 (其他). 对于随机矩阵 P , 定义 $f_{i,j}^{(n)}$ 表示从状态 i 转移 n 步第一次到达 j 的概率。

定义 $f_{i,j} = \sum_{n=1}^{\infty} f_{i,j}^{(n)}$ 。

定义 $\mu_{i,j} = \sum_{n=1}^{\infty} n f_{i,j}^{(n)}$, 特别的 $\mu_i = \mu_{i,i}$ 叫做 i 的均回时间。

定义 $S_{i,j} = \{n \mid P_{i,j}^n > 0\}$ 。

定义 $p_i = \gcd(S_{i,i}) = \gcd(x_1, x_2, \dots) (x_k \in S_{i,i})$ 叫做 i 的周期。

对于状态 (点) i , 我们将其分类:

若 $f_{i,i} = 1$, 我们将其叫做常返态, 否则叫做非常返态。

若 i, j 处于同一强连通块 (由 $P_{x,y} > 0$ 的边连接), 则 i, j 同为常返或非常返, 且周期相同 (后面有证明)。

且有 $f_{i,i} = 1 \Leftrightarrow \sum_{n=0}^{\infty} P_{i,i}^n = \infty$ 。

这是由于 $\sum_{n=0}^{\infty} P_{i,i}^n = \frac{1}{1 - f_{i,i}}$ 。

而这又是因为 $P_{i,j}^n = \sum_{l=1}^n f_{i,j}^{(l)} P_{j,j}^{n-l}$ (至于推导留给读者作为练习)。

并且我们还有加强, $\sum_{n=0}^{\infty} P_{i,j}^n = \frac{f_{i,j}}{1 - f_{j,j}}$, 组合意义是 i 到达 j 的次数 (这也可以简单证明这个式子)。

对于马尔科夫链有一个非常重要的定理:

定理 6.1. 对于周期 d 且常返的状态 i , 有 $\lim_{n \rightarrow \infty} P_{i,i}^{nd} = \frac{d}{\mu_i}$ 。

对于周期 1 的常返态 j , 有 $\lim_{k \rightarrow \infty} P_{i,j}^k = \sum_{k=1}^{\infty} f_{i,j}^{(k)}$ 。

一个吸收态 (永远回到自己的状态) 就是一个周期 1 的常返态, 而这种情形下上面这条性质从组合意义方面是显然的, 其他情形比较复杂。

接下来我们介绍极限与平稳分布:

定义分布 d_i 为一 n 阶行向量, 满足 $\sum_i d_i = 1$, 其中平稳分布 π_i 满足 $\pi P = \pi$ 。

若图为强联通 (即马尔科夫链不可约) 且每个状态周期都为 1, 我们将其称作不可约遍历链。

对于不可约遍历链, 其极限 $\lim_{n \rightarrow \infty} P_{i,j}^n = w_j, \forall j$, 并且由 $w_j = \pi_j$ 为一平稳分布, 且唯一。

证明考虑对 $P_{i,j}^{n+1} = \sum_k P_{i,k}^n P_{k,j}$ 两端取极限证得极限分布平稳, 再取平稳分布 π' , 对 $\pi' = \pi' P^n$ 右端取极限证得平稳分布唯一。

实际上单从平稳分布的角度描述, 由于随机矩阵 P 有一特征值 1, 并且 $I - P$ 秩为 $n - 1$, 因此其特征子空间维度为 1, 归一化后的特征向量即平稳分布唯一确定。

也就是说对于强连通图 (不可约链) 平稳分布总是唯一的, 只是极限分布可能不存在了, 并且我们还有 $\pi_i \mu_i = 1$, 从线性代数角度比较容易推出 (2019 年王修涵论文)。

接下来我们阐述可约矩阵的极限计算方式 (前提是常返态有极限)。

首先我们将状态分类, 即常返态 (汇强连通块) 和非常返态 (其他的强连通块), 我们知道非常返态的极限分布为 0。

而对于常返态, 我们考虑设 $F_{i,j} = \sum_{k=1}^{\infty} f_{i,j}^{(k)}$, 对于同处一汇连通块的 i, j 必有 $F_{i,j} = 1$, 不可达的 i, j 有 $F_{i,j} = 0$ 。

对于某一汇连通块, 我们知道从某个非常返态 i 到这个块的 f 仅会经过一条进入块的边, 也就是说我们可以分开计算, 先在非常返的状态的导出子图中算出 $\sum_{k=0}^{\infty} P_{i,j}^k$ 。

对于这一部分的计算, 我们发现此时转移矩阵 P' 谱半径 < 1 (即所有特征值绝对值 < 1 , 证明需要取次可乘的行和范数, 并考虑 $Ax = \lambda x$ 的范数)。

于是 $I - P'$ 可逆, 求其逆即可, 可逆的证明就考虑 $I - P'$ 的特征值为 P' 的特征值减一, 全不为零, 则可逆。

接着我们转移过那些进入汇连通块的边, 即将 f 分为两段计算, 接着在块中计算剩下的 f 时, 继续从矩阵极限的角度考虑。

我们认为求得一半的 f 近似一个初始分布, 将其归一化后, 我们便得到了一个分布, 我们有以下定理:

定理 6.2. 对于分布 X , 不可约遍历链 P , $\lim_{k \rightarrow \infty} X \cdot P^k = \pi$, 即平稳分布

由于证明大部分来自于应用随机过程与分析, 与本文无关, 因此这里不证。

于是我们对汇连通块只需求出平稳分布, 最终乘上一个归一化时带来的常数即可, 而对强连通块求平稳分布相当于对特征值 1 求归一化的特征向量, 解方程即可。

我们还有一些关于平稳分布的结论:

$$\forall j, \sum_{i \neq j} \pi_i P_{i,j} = \sum_{i \neq j} \pi_j P_{j,i}, \text{ 即出入一致。}$$

比这个结论更强的一个条件被称作细致平衡条件: $\pi_x P_{x,y} = \pi_y P_{y,x}$ 。

对于无向图随机走路的平稳分布 (即带边权, 转移满足按边权分布等概率转移), 其满足细致平衡条件, 所以 $\pi_x = \frac{d_x}{\sum d_i}$, 其中 $d_x = \sum_y e_{x,y}$ 。

这样对于无向图我们可以快速求解其均回时间。

由于在OI中出现不少,这里仅列举经典的几道:HNCPC2019H有向图,[Wf2014]Pachinko。

6.2 有向图幂敛指数与周期

对有向图的邻接矩阵 A , 我们定义其乘法为与, 加法为或, 则存在 $k, k_0 \in \mathbb{Z}, k > 0$ 满足 $\forall c \geq k_0, A^{c+k} = A^c$ 。

其中 k 叫做图的周期, k_0 叫做幂敛指数, 我们将从求 k, k_0 的角度探索其存在性。

首先我们只需考虑强连通块的周期, 因为整张图的周期是所有强连通块的周期的 lcm , 而 k_0 则是所有块幂敛指数的最大值。

观察到此处周期与马尔科夫链中定义的周期一致, 只需将 $A_{i,j} = 1$ 的定义视作 $P_{i,j} > 0$ 。

那么我们有显然性质: 若 $x, y \in S_{i,i}$ 则 $x + y \in S_{i,i}$ 。

观察到对于 i 存在 c_0 使得 $\forall k \geq \lceil \frac{c_0}{p_i} \rceil, kp_i \in S_{i,i}$ 。

证明.

我们考虑 $p_i = 1$ 的情形即可, 首先找到一组 $\gcd = 1$ 的子集 $\{t_i\}$, 再由裴蜀定理找到一组 $\sum w_k t_k = 1$, 那么我们将 w 正负分离, 得到 $\sum w_k^+ t_k - \sum (-w_k^-) t_k = 1$ 。

则我们可取出 $x, y \in S_{i,i}$ 且 $x = y + 1$, 接下来我们取 $y + y = 2y, y + x = 2y + 1, x + x = 2y + 2$, 这样我们获得三个连续的数, 一直这样取直到形成连续 y 个数, 我们就证得了。□

我们上面说到, 对于强连通块, 其中每个点周期相同, 这只需要考虑从 x 到 y 长 l 的路径以及从 y 到 x 长 l' 的路径, 并取一非常大的 k 有 $kp_y + l + l' \in S_{x,x}$ 。

那么 $p_x | kp_y + l + l'$, 由 k 的任意性, $p_x | p_y, p_x | l + l'$, 即证得, 并且我们还有周期整除图中所有的环长。

那么周期就是所有环长(可能非简单)的 $\gcd$ (因为周期的定义 $P_{i,i}^n > 0$ 表明存在有一条环路为周期长)。

不过如果环非简单, 那么分解为每部分简单环的环长的 $\gcd$ 一定整除整个环长。

考虑快速的求法, 我们先找一棵 dfs 树, 然后假设当前有一个简单环。

不妨设这环的根为 x , 定义为环上 dfs 序最小的点, 从 x 出发不断沿着环直到走到第一条非树边 $y \rightarrow z$ 。

若 $y \rightarrow z$ 是返祖边, 那么 $x = z$, 否则有比 x 的 dfs 序还要小的点, 或者不是一个简单环, 矛盾。

若 $y \rightarrow z$ 是横叉边(注意横叉边只会从 dfs 序大的连向小的), 那么 $x \rightsquigarrow y$ 的这条路上一定经过 $w = LCA(y, z)$ 。

这时我们观察到简单环与另一个环有交, 并且差为 $x \rightsquigarrow z$ (只走树边) 的长度和 $x \rightsquigarrow y$ (先走树边再跨过那条横叉边) 的长度的差。

这意味着我们可以将两个环互相转化, 即我们将 $x \rightsquigarrow y$ 的路径改为 $x \rightsquigarrow z$ 的路径, 并将答案与那条横叉边两端的差取 $\gcd$ 。

如果改变后的环不简单了,我们就将其分为很多个简单环。

若 $y \rightarrow z$ 是前向边,同横叉边,我们依旧可以将这个环做同样转换。

经过这些过程,环上的非树边必定减少,按照这种减少所说明的,我们只要求得所有非树边两端路径长度的差(其为 dfs 树上深度加一的差的绝对值)的 gcd 即可。

至于幂级指数,我们估计它的界。

由于环可以被分解为简单环,因此我们在找周期的生成组(即我们取的那个 gcd 为 1 的子集)的时候只需要找 $\leq n$ 的部分。

而 n 个 gcd = 1 的数 a_i 正系数不可表示的最大的元素的一个上界是 $(\min\{a_i\}-1)(\max\{a_i\}-1)-1$ (由同余最短路),再乘上周期,我们可以粗略估计这是 $O(n^3)$ 的(实际上似乎是不超过 $(n-1)^2-1$ 的)。

7 计数相关

7.1 对角化方法

通常时候,将一个线性变换做 k 次即求一个矩阵 A^k ,若 A 相似于对角形 $C = \text{diag}\{\lambda_1, \lambda_2, \dots, \lambda_n\}$, 则:

$$A^k = (Q^{-1}BQ)^k = Q^{-1}B^kQ$$

其中使用对角化方法非常经典的例子是循环矩阵行列式,不妨设多项式 $f(x) = \sum_{i=0}^{n-1} a_i x^i$, 矩阵 $A_{i,j} = a_{(i-j) \bmod n}$ 。

不妨设范德蒙矩阵 $W_{ij} = \omega_n^{ij}$ 。

那么 $AW = W \cdot \text{diag}\{f(\omega_n^0), f(\omega_n^1), \dots, f(\omega_n^{n-1})\}$, 也就是说 A 的行列式就等于 $|A| = \prod_{i=0}^{n-1} f(\omega_n^i)$ 。

7.2 迹计数方法

这种方法是笔者在做一道 CF 题目时偶然想到,为此笔者命制了以下题目:

例题 2 (uoj699). 你需要给 n 个数分配 $[1, m]$ 的整数权值,使得 $a_i + K \geq a_{(i \bmod n)+1}$, n, m, K 给定,问方案数 $\bmod 998244353$ 。

笔者给出的标算是 $O(\min(m, nK) + \text{len} \log n \log \text{len})$ 的,其中 $\text{len} = \min(n, \frac{m}{K})$ 。

首先我们考虑暴力 DP,等价的,我们考虑一个矩阵 $M_{i,j} = [i + K \geq j]$,答案就是 $\text{tr}(M^n)$ 。

然后由我们上面所说的, $\text{tr}(M^n) = \sum \lambda_i^n$ 。

由一些推导,不妨设矩阵的特征多项式为 F ,则 $\text{tr}(M^n) = [x^n] \frac{F^R}{F}$,其中 F^R 表示将 F 翻转。

由特征多项式与行列式的关系我们可知, F 的第 $\partial_0 F - k$ 项 (倒数第 k 项, 倒数第 0 项即最高项) 即矩阵的 k 阶主子式之和 (若 $k = 0$, 则为 1) 乘上 $(-1)^k$ 。

那么问题转化为求 k 阶主子式之和, 即从主对角线上选出来 k 个位置, 这 k 行 k 列形成的矩阵的行列式的和。

首先这个矩阵的下三角总是满的, 并且若新的矩阵中格子 (i, j) 为 1, 则 $(i+1, j), (i, j-1)$ 都为 1 (左边和下面)。

那么这个矩阵的行列式非 1 即 0, 当且仅当这个矩阵为下三角矩阵时行列式值为 1。

我们可以将矩阵从左到右下沿主对角线分为 $\lceil \frac{n}{K} \rceil$ 份, 每份是一个 $K \times K$ 的矩阵, 最后一个矩阵可能不足 K 阶。

那么我们发现同一个矩阵中只会选取一行一列, 若相邻两个矩阵都选取了行列, 则后一个元素在其矩阵中的相对位置要靠后。

其实就是选择恰 k 个 $[1, m]$ 的数使得 $a_{i+1} > a_i + K$, 那么插空法即可, 多项式的系数就是一个组合数。

那么后面的 $\frac{(F')^R}{FR}$ 即线性递推。

对于笔者当时所做的 CF1580F², 只需要将后半部分改为对矩阵分块并观察即可。

8 线性基

在线性代数中, 我们确定线性空间 V 的一组基 $v_1, v_2, \dots, v_k$, 并解方程 $c_1 v_1 + c_2 v_2 + \dots + c_k v_k = t$ 来表出 t 。

但是在实际计算机表示中, 我们不可能通过一句简单的解方程获得元素之间的关系, 为此我们引入了线性基这一数据结构。

为了解方程, 我们将寻找的基消成上三角形, 我们对某一元素的消元, 其实只是在寻找另一个等价的元素作为新的基。

后文我们称线性基里的元素为消元元素, 任意一组基里的元素为基元素 (注意这个基的意义不同于线性基)。

8.1 线性基交

假设我们以基的方式获取了两个线性空间 A, B , 现在要求他们的交的一组基。

那么方法非常简单:

第一步, 首先获取 $v \in A \cap B$, 无法获取则退出。

第二步, 将 B 商掉 v , 重复这两步操作。

进一步的, 我们将数学语言转换为可以实现的步骤。

我们先来实现第一步:

²<https://codeforces.com/contest/1580/problem/F>

考虑将 A, B 的基元素放在一起以获取一组线性相关表示 $c_1a_1 + c_2a_2 + \cdots + c_na_n + c_{n+1}b_1 + c_{n+2}b_2 + \cdots + c_{n+m}b_m = 0$ 。

如果只有全 0 的平凡解, 说明 A, B 线性无关, 程序可以立即退出。

否则前一半是 A 的元素, 后一半是 B 的元素, 取后一半即可获得 v 。

继续实现第二步:

不妨设 $c_{n+1} \neq 0$, 则由于 $v = c_{n+1}b_1 + c_{n+2}b_2 + \cdots + c_{n+m}b_m$ 。

用 v 把 b_1 替换了, 取剩下的元素, 就相当于直接删去 b_1 。

这里, 笔者也命制了一道线性基交的简单题:

例题 3 (uoj698). 给定 n 组大小 w_i 的线性空间, 总的空间维度不超过 w , Q 次询问一个向量第一次不在哪个线性空间中出现 (从左到右)。

解法 2.

考虑前 i 个线性空间的交, 这样会得到 w 个本质不同的线性空间, 并且呈包含关系。

那么我们将这 w 个线性空间按照时间顺序插入一组新的基, 对于询问在这组新基中找到表出当前向量的最早在什么时候不出现。

这样就可以做到 $(\sum w_i + q)w$ 了

8.2 带删除线性基

首先考虑离线删除:

仍然考虑按照元素的删除时间维护最大删除时间线性基, 即基元素 $v_1, v_2, \dots, v_k$ 按照删除时间从大到小构成基。

在新加入一个元素进行地位上的交换。

首先我们在正常线性基的每一个被消元的元素上面附加上一个删除时间, 表示这个元素属于删除时间在记录时间及以前的基元素组成的线性空间中。

麻烦点的做法是先将 v 用线性基中记录时间比 v 删除时间早的元素消去得到 v' 。

若 $v' = 0$, 则退出, 否则 v' 有主元 t 。

若 t 对应的那一位被占用 (则那一位删除时间一定比 v 晚), 不妨考虑以前那一位上基元素并没有被加入, 先加入 v' , 再加入原来基元素及以后的元素。

那么就相当于再次插入那一位上被替换的基元素, 注意到我们之前就用所有比 v 删除时间早的元素消去了它, 那么现在只需要让它消去一次 v 即可, 这样是一种实现。

注意到我们只需要上三角基, 于是我们按照从高到低, 能换就换即可 (但是此时由于消元不充分, 换下去的元素还可以消很多次, 而上面的实现中只需要最开始 v 消很多次)。

然后是在线删除:

首先我们找到每个元素 (包括线性基中的消元元素和剩下的能被线性基表出的元素) 在消元元素对应的基元素上的线性表示。

如果删去一个不是基元素直接删就行了。



否则我们找所有的非基元素,若其表示含有当前删去的基元素 v ,就用那个元素(不妨设为 z)替代 v ,具体来说就像线性规划里的转轴一样。

否则删去 v 后非基元素依然能被剩下基元素表出,则基大小减一,非基元素线性表示不变,由于我们还维护了一个线性基,所以还要关心线性基里的元素的表示情况。

按主元顺序找到最低位上线性表示含有 v 的消元元素 x ,再从这一位向上访问元素 y ,若其表示也含 v ,则用 x 消去 y 中的 v 。

这样操作之后,线性基仍是上三角基,主元不变,此时除 x 之外的元素张成除 v 之外基元素的线性空间,则我们可以直接删掉 x 。

时间复杂度单次 $O(nm+m^2)$,其中 n 是总元素个数, m 是基的大小,视情况可以再除一个 w ,几乎只在 n, m 同阶时好用。

一道经典的例题是 UOJ453³。

9 线性拟阵奇偶问题

线性拟阵奇偶问题是组合优化中一类重要的问题,在近年得到充分研究,可以看作非常多问题的推广,如最大流,最大匹配都可以被线性拟阵奇偶问题包含(但相应地时间复杂度上升)。

我们先来描述线性拟阵奇偶问题:

定义 9.1 (线性拟阵奇偶问题). 给定一线性拟阵 A , 将其中的元素划分为很多两两不交的对, 求最多能选出多少对使得选出来的对包含的所有元素独立。

我们定义 $\nu(A)$ 为最多能选出来的对数。

对于线性拟阵奇偶问题的代数算法有两种构造:

方法 1 (稀疏型). 设线性拟阵 A 的一种线性表示为矩阵 A 的列空间, 不妨设 A 的行列集合分别为 n, m 不交(我们还约定 n, m 表示对应的集合大小)。

先构造一个图特矩阵 T , 其基图为点集为 m 的, 边集为线性拟阵奇偶问题中所划分出来的对之间的连边的图。

则我们有:

$$2\nu(A) = \text{rank} \begin{bmatrix} O & A \\ -A^T & T \end{bmatrix} - m$$

方法 2 (紧凑型). 设线性拟阵 A 的一种线性表示所划分出的每一对列向量为 (b_i, c_i) , 共有 $\frac{m}{2}$ 对, 不妨设 $\frac{m}{2}$ 个代数独立的未定元 x_i , 则:

³<https://uoj.ac/problem/453>



$$2\nu(A) = \text{rank} \sum_i x_i (b_i \wedge c_i)$$

其中 $\wedge$ 是楔积: $a \wedge b = ab^T - ba^T$

接下来我们说明稀疏型构造的正确性:

引理 9.1.

对于混合斜对称矩阵 $K+T$, 其中 K 是数值斜对称矩阵, T 是图特矩阵, 有:

$K+T$ 非奇异当且仅当存在其行列 (不妨设两集合一致, 为 U) 的子集 S 使得 $K[S]$ 和 $T[U \setminus S]$ 均非奇异

引理的证明.

考虑对 $K+T$ 的 Pf 展开的等式:

$$\text{Pf}(K+T) = \sum_{I \subseteq U} \text{sgn} \cdot \text{Pf}(K[I]) \text{Pf}(T[U \setminus I])$$

此式的证明只需考虑不断展开, 其中 sgn 为 1 或者 -1, 这里只需展现一种形式。

若左边不为 0, 则存在右边的一项不为 0, 也就是说存在一个 $K[S]$ 和 $T[U \setminus S]$ 非奇异。

反之, 若 $K[S]$ 和 $T[U \setminus S]$ 非奇异, 由于 T 是未定元矩阵, 对于任意的 $T \neq S$, $\text{Pf } T$ 都不可能消去, 由此证毕。

□

那么此时就容易证明稀疏形式的正确性了。首先我们发现 T 非奇异 (由定义), 不妨设使主子式不为 0 的主子矩阵的行列集合为 S 。

故设 $m \subseteq S$, 并设 $X = S \setminus m$, 则由引理存在一个 S 的子集 V 使得 $T[V]$ 和 $K[S \setminus V]$ 均非奇异。

由这个图特矩阵的基图的特殊性, 我们发现 V 和 $m \setminus V$ 都是由一对一对的元素对拼成的。

又由于左上角为全 0 矩阵, 所以 $|X| = |m \setminus V|$ 并且 $m \setminus V$ 对应的 A 中的列线性无关 (否则 $K[S \setminus V]$ 奇异)。

反之选择一组线性拟阵对偶问题的极大解也可以得到对应的无关组, 那么就能说明稀疏型的构造成立了。

考虑对紧凑型的证明, 后者显然具有计算复杂性上的优势 (如同后者的名字)。

紧凑型构造的证明.

考虑对稀疏型构造得到的矩阵应用舒尔补得到的秩恒等式:

$$\text{rank} \begin{bmatrix} O & A \\ -A^T & T \end{bmatrix} = m + \text{rank } AT^{-1}A^T$$

整理式子, 并将 x_i 换为 $\frac{1}{x_i}$ 可证。

□



那么我们考虑使用紧凑型的构造获取方案。

首先假设我们构造得到的矩阵为 W ，我们获取主子矩阵 S 使得其非奇异，则我们可以把所有向量均在 S 上截取，这样 W 变为 $W[S]$ 。

因此我们总可以在这样的转化下假设 W 满秩，因此可逆。

于是我们需要依次移除向量对 b_i, c_i ，然后检查秩是否发生变化，亦即是否仍然可逆。

由于 $b_i \wedge c_i$ 的秩至多为 2，因此使用 SMW 公式加速运算。

总共的时间复杂度是 $O(n^\omega + mr^2)$ ，其中 r 是答案大小。

我们简述一道例题：

例题 4 (极大 3-子仙人掌). 选取图中最多的边不交三元环，使选出来的三元环点集在原图中的导出子图除了选出来的三元环之外无其他环。

这是一道线性拟阵奇偶问题的模板题。

解法 3.

即选出来最多的三元环使得形成一棵边仙人掌，那么其生成森林一定恰经过每个三元环两条边。

那么选取拟阵为图拟阵，每个图中的三元环任意选取两条边作为一对，则选出来的所有对构成对应仙人掌的生成森林。

那么最多能选取的对数就是答案。

此外，线性拟阵奇偶问题还有更多的应用，甚至包括物理方面，笔者也为此命制了一些题目，不过没有公开。

10 杂项

10.1 广义串并联图方法

通常我们会被要求求特殊的与图论有关的行列式，其通常由许多连续主对角方阵相加得到。

在通过变换使得矩阵是某张广义串并联图的基尔霍夫矩阵的时候，就可以使用广义串并联图的方法获取其行列式。

这方面的例题是集训队互测的 D1T1，可以反复差分得到一张图的基尔霍夫矩阵，对其求行列式就是带权生成树计数。

应用去年所介绍的广义串并联图方法即可求得。



10.2 实二次型的极值

这方面的应用将由一道例题总结

例题 5 (BZOJ4181⁴). 给定 n 阶实对称矩阵 A , 有 $\mathbf{x}^T A \mathbf{x} = 1$, 试问 $\mathbf{x}^T \mathbf{x} = |\mathbf{x}|^2$ 的最小值。

熟知, 实对称矩阵 A 正交相似于 $W = \text{diag}\{\lambda_1, \lambda_2, \dots, \lambda_n\}$, 其中 λ_i 为 A 的特征值且为实数。

不妨设 $A = C^T W C$ 其中 $C^T C = I$ 。

则令 $\mathbf{x}' = C \mathbf{x}$ 有 $\mathbf{x}'^T W \mathbf{x}' = 1$ 。

由于 C 是正交阵, $|\mathbf{x}'| = |\mathbf{x}|$ 。

而 $\min(|\mathbf{x}'|^2) = \frac{1}{\max(\lambda_i)}$ 。

那么现在变为求特征值绝对值的最大值, 即矩阵的谱半径。

考虑迭代方法, 首先假定我们有一个合适的向量 X_0 , 作迭代 $X_{k+1} = \text{normalize}(A X_k)$ 。

其中 normalize 表示把向量单位化。

则迭代多次后 $A X = \lambda_{\max} X$, 取 X 不等于 0 的分量即解得 $\lambda_{\max}$ 。

考虑证明, 取 λ_i 的特征子空间 V_i , 不妨将 X 在所有 V_i 的直和上分解, 设为 $X = \sum c_i v_i$ 。

则 $X_k = \text{normalize}(\sum \lambda_i^k c_i v_i)$, 迭代多次后 X 在 $V_i, \lambda_i \neq \lambda_{\max}$ 上的分量与在 $V_{\max}$ 上的分量的比值指数级衰减。

其中 $V_{\max}$ 是 $\lambda_{\max}$ 对应的特征子空间。

由证明, 其实存在一些特殊情形, 例如随机取 X_0 时仅取到 $V/V_{\max}$ 中的向量, 这时相应的概率为 0。

还有一种情况, 特征值中既有 $\lambda_{\max}$ 出现也有 $-\lambda_{\max}$, 也会干扰证明。

这时我们注意到实对称矩阵谱半径 $\rho(A) = \sqrt{\rho(A^2)}$ (谱映射定理), 而 $A^2 = A^T A$ 是半正定的, 所以我们求 A^2 的谱半径再开方即可。

10.3 转置原理

这也是近年来发展的一种幂级数方法, 其借用了线性代数来描述问题, 并借用一些定理作转化。

转置原理将线性算法表示为 $A \mathbf{x}$ 的形式, 考察 $A^T \mathbf{x}$ 的作用, 并将 A^T 分解, 依次转置获得 $A \mathbf{x}$ 的描述。

类似的, 我们可以推出“求逆”原理 (例如 DFT 和 $IDFT$ 的关系) 等等。

由于近年皆有论文描述转置原理, 这里不再赘述。

⁴<https://darkbzoj.tk/problem/4181>



11 总结

本文介绍了非常多的应用，但是总的来说不算深入，这是由于作者的水平依然不足或者线性代数太过基础。对图特定理的证明是作者对线性代数算法寄予的厚望，希望对代数感兴趣的同学可以找到一些更加有趣深入的应用，在幂级数推导之余也思考线性代数的算法，或者将二者结合发挥出更大的作用。

致谢

感谢中国计算机学会提供学习和交流的平台。感谢重庆巴蜀中学校的黄新军老师给予的关心和指导。感谢家人对我的关心，支持与鼓励，生活上的照顾。感谢唐绍轩同学为本文验稿。

参考文献

- [1] Kazuo Murota, Matrices and Matroids for Systems Analysis
- [2] CHEUNG & LAU & LEUNG Algebraic Algorithms for Linear Matroid Parity Problems
- [3] 孙耀峰。(2017)。动态传递闭包问题的探究。IOI2017 中国国家集训队论文集。
- [4] 杨家齐。(2017)。基于线性代数的一般图匹配。IOI2017 中国国家候选队论文集。
- [5] 陈宇。(2020)。转置原理的简单介绍。IOI2020 中国国家候选队论文集。
- [6] 王展鹏。(2020)。信息学竞赛中行列式的相关问题。IOI2020 中国国家候选队论文集。
- [7] 王修涵。(2019)。浅谈图模型上的随机游走问题。IOI2019 中国国家候选队论文集。
- [8] 张波，应用随机过程。
- [9] JAMES GEELLEN, SATORU IWATA, MATROID MATCHING VIA MIXED SKEW-SYMMETRIC MATRICES
- [10] JOSEPH P. S. KUNG, Bimatroids and Invariants
- [11] R. B. Bapat, Konig' s Theorem and Bimatroids
- [12] 詹兴致，矩阵论



浅谈与 Lyndon 理论有关的字符串组合问题

华东师范大学第二附属中学 万成章

摘要

Lyndon 串在字符串的最小表示等问题上有较好的性质,能揭示一些字符串中的深刻结构,本文在 Lyndon 串的基本性质的基础上介绍了由此可以得到的一些特殊结构和算法,用于解决一些有关字符串的组合问题。

1 引言

在信息学竞赛中, Lyndon 理论是字符串论重要的一部分,它目前在信息学竞赛中的普及程度相对较低,但其实 Lyndon 串的应用场合以及其本身的结构其实都很丰富和深刻。本文回顾了 Lyndon 串的基本知识,然后介绍了它在一些字符串问题中的应用。

本文第 2,3 节介绍了在本文中用到的关于字符串的记号和前置知识。

第 4 节介绍了 Lyndon 分解及其基本应用。

第 5 节介绍了 Lyndon 数组以及它在与平方串有关的问题中的应用。

第 6 节介绍了对 Lyndon 串计数和排名的算法。

2 定义与约定

对于字符串 S (本文中一般惯用 S, T, w, u, v 等表示字符串):

记 $|S|$ 表示它的长度 (包含的字符数量);

记 S_i 表示它的第 i ($i \in [1, |S|]$) 个字符;

记 $S[l, r]$ 表示其第 l 到第 r ($1 \leq l, r \leq |S|$) 个字符组成的子串,特别地当 $l > r$ 时这表示空串;

记 $Pre(S, i) = S[1, i]$ 表示 S 的长度为 i ($i \in [0, |S|]$) 的前缀, $i \neq |S|$ 时称为真前缀;

记 $Suf(S, i) = S[|S| - i + 1, |S|]$ 表示 S 的长度为 i ($i \in [0, |S|]$) 的后缀, $i \neq |S|$ 时称为真后缀。

长度为零的串称为空串,用 ϵ 表示。

字符串 S 的字符集是它的所有字符的取值范围,本文中默认记为 Σ 。

对于字符串 S, T , 可以按字典序进行比较,称 $S < T$ 当且仅当 S 是 T 的真前缀,或存在 i 使得 $Pre(S, i) = Pre(T, i)$ 且 $S_{i+1} < T_{i+1}$ ($i < \min(|S|, |T|)$)。

对于字符串 S, T , 记 ST 或 $S + T$ 表示顺次拼接 S 和 T 得到的字符串。

记 S^n ($n \in \mathbb{Z}^+$) 表示 n 个 S 顺次拼接得到的字符串。

3 基础知识

3.1 Border 与周期

对于字符串 S :

如果整数 $p \in [1, |S|]$ 满足 $\forall i \in [1, |S| - p], S_i = S_{i+p}$, 则称 p 是 S 的周期, 如果 p 是 $|S|$ 的因子则称其为整周期。 S 的最小周期记为 $\text{minper}(S)$ 。

如果整数 $p \in [0, |S|)$ 满足 $\text{Pre}(S, p) = \text{Suf}(S, p)$, 则称 p 是 S 的 Border。

如果周期 $p \neq |S|$ 或 Border $p \neq 0$, 则称它是非平凡的。

那么我们有:

引理 3.1 p 是 S 的周期 $\iff q = |S| - p$ 是 S 的 Border。

引理 3.2 (弱周期引理) p, q 是 S 的周期且 $p + q \leq |S| \implies \text{gcd}(p, q)$ 是 S 的周期。

证明思路是通过说明 $|p - q|$ 是周期, 从而进行辗转相除, 详细证明略。

引理 3.3 若 p, q ($p < q$) 都是 S 的 Border, 那么 p 是 $\text{Pre}(S, q)$ 的 Border。

3.2 后缀数据结构

描述字符串的所有子串或后缀的结构包括后缀数组、后缀自动机、后缀树等。本文只需要用到较简单的后缀数组。

对给定串 S 的所有非空后缀按字典序排序, 排名第 i 的后缀的起始位置记为 $\text{SA}(i)$, 显然这是一个排列, 其逆排列记为 $\text{ISA}(i)$ 。

i	1	2	3	4	5
S_i	a	b	a	b	a
$\text{SA}(i)$	5	3	1	4	2
$\text{ISA}(i)$	3	5	2	4	1

实际应用中通常使用比较简单的倍增法在 $O(|S| \log |S|)$ 的时间复杂度内构造后缀数组, 也有 $O(|S|)$ 构造的方法, 本文不做介绍。

通过对后缀数组进行扩展, 定义 $\text{H}(i)$ 表示排名分别为 i 和 $i - 1$ 的后缀的最长公共前缀 (Longest Common Prefix, LCP) 长度, 我们通过求它的区间最小值即可求得任意两个后缀的 LCP, 这理论上可以在 $O(|S|)$ 的预处理后每次 $O(1)$ 回答。

4 Lyndon 分解

4.1 基本介绍

定义 4.1 (Lyndon 串) 若字符串 w 小于它的每一个真后缀, 则称 w 是 Lyndon 串。

定义 4.2 (近似 Lyndon 串) 若字符串 w 是 Lyndon 串, w' 是其前缀, 形如 $w^k w'$ ($k \in \mathbb{Z}^*$) 的串称为近似 Lyndon 串。

对于任意字符串 w , 如果我们可以把它分解成若干个 Lyndon 串 $w_1, \dots, w_k$ 拼接的形式, 并且 $w_i \geq w_{i+1}$ ($1 \leq i < k$), 就把这种分解 $w = w_1 + \dots + w_k$ 称为 w 的 Lyndon 分解, 每个 w_i 都称为 w 的 Lyndon 因子。例如 $abbabbaba = abb + abb + ab + a$ 就是 $abbabbaba$ 的 Lyndon 分解。

由于相对于之后的内容, Lyndon 分解比较为人所熟知, 所以这里仅简略地介绍一些必要的性质, 如果需要对 Lyndon 分解更详细的说明, 读者可以参考 2021 年集训队论文 (参考文献 [1])。

引理 4.1 若 $w = w_1 + \dots + w_k$ 是 w 的 Lyndon 分解, 则 w_k 是 w 的最小真后缀。

Proof. 设 w 的最小真后缀为 $w' = uw_i w_{i+1} \dots w_k$ ($|u| \leq |w_{i-1}|$), 则根据 Lyndon 串的定义有 $w' \geq u \geq w_{i-1} \geq w_k$, 所以 w' 只能为 w_k 。□

引理 4.2 若 u, v 是 Lyndon 串, 且 $u < v$, 则 uv 也是 Lyndon 串。

Proof. 由 $u < v$ 和 v 是 Lyndon 串知 $uv < v$ 。

对于 v 的后缀 v' 有 $v' > v > uv$, 而对于 u 的真后缀 u' 有 $u' > u$, u 不是 u' 的前缀, 则 $u'v > uv$ 。所以 uv 的任何真后缀都大于 uv , 即 uv 是 Lyndon 串。□

由引理 4.2 可以得到 w 的 Lyndon 分解存在 (一开始每个 Lyndon 因子设为单个字符, 然后每次合并相邻的满足 $u < v$ 的 Lyndon 因子 u, v), 结合引理 4.1 立刻得:

定理 4.1 任何字符串 w 的 Lyndon 分解存在且唯一。

Duval 算法可以 $O(|w|)$ 求出 w 的 Lyndon 分解。从前到后加入字符, 维护当前前缀的一个还未分解的近似 Lyndon 后缀 $w^k w'$, 令 $x = w_{|w'|+1}$, 若下一个加入的字符是 y , 那么: 当 $x = y$ 时直接延长当前的近似 Lyndon 串; 当 $x < y$ 时说明 $w^k w'y$ 是 Lyndon 串, 将其整体作为新的 w ; 当 $x > y$ 时就分解出 k 个 w , 然后从 $w'y$ 这一段开头继续尝试寻找近似 Lyndon 串。有关其正确性、复杂度证明及简单应用 (例如求出每个前缀的最小后缀), 可见 [1]。

Duval 算法的过程引导我们发现下述性质:

引理 4.3 若 w 是近似 Lyndon 串, 则 w 的任意前缀 w' 是近似 Lyndon 串。

4.2 最小后缀族

这一小节我们将介绍 Lyndon 分解在有关最小化字典序的问题上的应用。

在上一小节的引理 4.1 中已经知道 w 的最小真后缀是其最后一个 Lyndon 因子, 那么自然引出与之类似的一些问题, 比如 w 的最大后缀, w 的最小循环表示等等, 我们需要提取这些问题的共同点进行抽象, 并得到有用的结构。

定义 4.3 (最小后缀族) 对于字符串 w 的后缀 w' , 如果存在字符串 u 使得对于任意 w 的后缀 w'' 有 $w'u < w''u$, 则称 w' 是 w 的一个有效后缀 (Significant Suffix), 所有有效后缀构成的集合称为 w 的最小后缀族, 记为 $SS(w)$ 。

设 $w = w_1^{c_1} w_2^{c_2} \dots w_k^{c_k}$ 是 w 的 Lyndon 分解, 其中 w_i 两两不同, 我们先证明最小后缀族的基本性质。

引理 4.4 若 u^2v, uv, v 都是 w 的后缀, 则 uv 不是 w 的有效后缀。

Proof. 只需注意到 $uvy < vy \iff u^2vy < uvy$ 即可, uvy 一定不是 u^2vy, uvy, vy 中最小的。 $\square$

引理 4.5 若 $u, v \in SS(w)$, 且 $|u| < |v|$, 则 u 是 v 的 border, 且 $2|u| \leq |v|$ 。

Proof. 由定义知, 存在 y 使得 $uy < vy$, 也存在 y 使得 $vy < uy$, 所以 uy 和 vy 的比较不可能在前 $|u|$ 位就结束, 所以 u 是 v 的前缀, 进而 u 是 v 的 border。

若 $2|u| > |v|$, 则令 $v' = Suf(u, 2|u| - |v|)$, $u' = Pre(u, |v| - |u|)$, 于是 $v = u'u'v'$, $u = u'v'$, v' 都是 w 的后缀, 由引理 4.4 知 u 不是有效后缀, 矛盾。 $\square$

引理 4.6 有效后缀一定形如 $w_i^{c_i} \dots w_k^{c_k}$ 。

Proof. 引理的证明实际分为两个部分。

我们首先说明有效后缀一定形如 $w_i^x w_{i+1}^{c_{i+1}} \dots w_k^{c_k}$ ($0 < x \leq c_i$)。假设不然, 则某个有效后缀形如 $uw_i^x w_{i+1}^{c_{i+1}} \dots w_k^{c_k}$, 其中 u 是 w_i 的真后缀, 由于 $u > w_i$ 且 w_i 不是 u 的前缀, 所以对于任意的 v 有 $uv > w_i v$, 对于任意给定的 v' 可以令 $v = w_i^x w_{i+1}^{c_{i+1}} \dots w_k^{c_k} v'$, 于是这个以 uw_i^x 开头的后缀一定没有以 w_i^{x+1} 开头的后缀优。

然后, 我们说明上一段的 x 必定为 c_i 。假设 $x \neq c_i$, 令 $u = w_i$, $v = w_i^{x-1} w_{i+1}^{c_{i+1}} \dots w_k^{c_k}$, 由引理 4.4 知 $w_i^x w_{i+1}^{c_{i+1}} \dots w_k^{c_k} = uv$ 不是有效后缀, 矛盾。 $\square$

令 $s_i = w_i^{c_i} \dots w_k^{c_k}$, 整理一下目前已经得到的性质: w 的所有有效后缀出自 s_i 中, 并且任何一个更长的有效后缀长度至少为更短有效后缀的 2 倍, 因此有效后缀的数量为 $O(\log |w|)$ 。

4.2.1 SS(w) 的求法

从上面的分析不难看出要从 Lyndon 分解的角度求解 w 的有效后缀, 先给出一个关于 Lyndon 分解结构的引理:

引理 4.7 s_{i+1} 是 s_i 的前缀的充要条件是 $i \geq \lambda$, 其中 s_λ 是对 w 进行 Duval 算法时第一次比较完字符串末尾时的近似 Lyndon 后缀。

Proof. 根据 Duval 算法的比较规则可知, $s_\lambda = w_\lambda^x s_{\lambda+1}$ 是近似 Lyndon 串, 那么 $s_{\lambda+1}$ 就是 w_λ 的前缀, 从而是 s_λ 的前缀。由引理 4.3 知 $s_{\lambda+1}$ 也是近似 Lyndon 串, 故可以继续考虑 $\lambda' = \lambda + 1$, 以此类推可证明充分性。

如果 Duval 算法在截取出 w_i 时没有比较到末尾, 说明 s_{i+1} 有一位比 w_i 的对应位小, 显然不可能是 s_i 的前缀, 于是必要性得证。□

现在, 我们事实上已经有了一个 $O(|w|)$ 计算 $SS(w)$ 的方法: 对 w 进行一次 Duval 算法得到 λ , 再取所有的 s_i ($i \geq \lambda$), 不过我们还没有根据定义说明这其中的所有 s_i 确实是 $SS(w)$ 的元素, 这会在稍后进行说明。

4.2.2 后接 v 的最小后缀

我们要解决的问题是: 给定字符串 w 和 v , 求 uv 的最小值, 要求 u 是 w 的后缀, 记这样的 uv 为 $\text{Minsuf}(w, v)$ 。

前面的讨论说明了 u 一定是 $s_\lambda, \dots, s_k$ 之一。令 $s_i = s_{i+1} + y_i$ ($s_{m+1} = \epsilon$), 并令 $x_i = y_i + s_{i+1}$, 那么有 $s_i = s_{i+1} + x_i^{c_i}$ 。

引理 4.8 $\forall i \in [\lambda, k-1], y_i > x_{i+1}^\infty$ 。

Proof. 我们在不等式两边都前加一个 s_{i+1} , 也就是要证 $s_{i+1}y_i > s_{i+1}x_{i+1}^\infty$ 。

而 $s_{i+1}y_i = w_i$, $s_{i+1}x_{i+1}^\infty = s_{i+1}(y_{i+1}s_{i+2})^\infty = (s_{i+2}y_{i+1})^{c_{i+1}}s_{i+2}(y_{i+1}s_{i+2})^\infty = (s_{i+2}y_{i+1})^\infty s_{i+2} = w_{i+1}^\infty s_{i+2}$, 也就是要证 $w_i > w_{i+1}^\infty$ 。

根据定义有 $w_i > w_{i+1}$, 则设 $w_i = w_{i+1}^k + w'$ ($k = 0, 1, 2, \dots$), 其中 w_{i+1} 不是 w' 的前缀, 那么根据 w_i 是 Lyndon 串知 $w' > w_{i+1}$, 所以 $w_i > w_{i+1}^\infty$, 故原命题得证。□

引理 4.9 $\forall i \in [\lambda, k-1], x_i^\infty > x_{i+1}^\infty$ 。

Proof. $x_i^\infty \geq y_i > x_{i+1}^\infty$ 。□

现在考虑比较 $s_i v$ 和 $s_{i+1} v$ 的大小, 由于 $s_i = s_{i+1}x_i^{c_i}$, 所以还可以继续展开成 $s_i v = x_k^{c_k} x_{k-1}^{c_{k-1}} \dots x_i^{c_i} v$, 同时 $s_{i+1} v = x_k^{c_k} x_{k-1}^{c_{k-1}} \dots x_{i+1}^{c_{i+1}} v$, 删去公共前缀后等价于比较 $x_i^{c_i} v$ 和 v , 这又等价于比较 x_i^∞ 和 v 。

根据引理 4.9, x_i^∞ 是单调的, 因此我们可以二分找到一个 i 使得 $x_i^\infty > v > x_{i+1}^\infty$, 那么此时 $\text{Minsuf}(w, v) = s_{i+1}v$ 。当然取 s_{k+1} (即空串) 和 s_λ 分别对应二分中的两种极端情况, 此处不再赘述。

从这里我们也可以看出每个 s_i ($i \in [\lambda, k+1]$) 确实都是 $\text{SS}(w)$ 的元素。

定理 4.2 按照上文记号, $\text{SS}(w) = \{s_\lambda, \dots, s_{k+1}\}$, 并且:

$$\text{Minsuf}(w, v) = \begin{cases} s_\lambda v & v > x_\lambda^\infty \\ s_i v & x_{i-1}^\infty > v > x_i^\infty \\ v & x_k^\infty > v \end{cases}$$

4.2.3 相关例题

例题 4.1 给定字符串 S , 求 S 的最大后缀。

将字母表反转, 就大致是要求出最小后缀。但这是不准确的, 因为此时如果某个后缀是另一个后缀的前缀, 那么我们希望取更长的那个。

对此解法稍作修改, 反转字母表后设一个极大字符 z , 求 $\text{Minsuf}(S, z)$ 就是答案, 而由于 z 大于任何其他字符, 所以 $\text{Minsuf}(S, z) = s_\lambda z$, 于是进行一次 Duval 算法即可求出答案, 时间复杂度为 $O(|S|)$ 。

例题 4.2 (JSOI 2019 节日庆典) 给定字符串 S , 求每个前缀 $\text{Pre}(S, i)$ 的最小表示, 即 $S[j, i] + S[1, j-1]$ ($j \in [1, i]$) 的最小值。

虽然该问题不能直接写成 $\text{Minsuf}(w, v)$ 的形式, 但是依然可以得到类似引理 4.6 的结论: S 的最小表示的起始部分一定是 $s_\lambda, \dots, s_{k+1}$ 之一。

所以只要维护前缀的 Lyndon 分解, 再暴力比较每个 s_i 对应的循环表示的大小即可, 根据 $|\text{SS}(w)| = O(\log |w|)$, 事件复杂度为 $O(|S| \log |S|)$ 。

事实上本题还存在线性做法: 进行 Duval 算法时若当前前缀为 $uw^k w'$, 并且它第一次被遍历到, 其近似 Lyndon 后缀为 $w^k w'$, 那么 u 中的部分不可能作为最小表示的开头 (类似引理 4.7), 所以只需考虑以 $w^k w'$ 开头和以 w' 中的某个位置开头这两种情况。如果是后者, 我们删除最小表示的最后 $|w|$ 个字符, 不难发现这恰好得到了 $uw^{k-1} w'$ 的一个循环表示, 而这是已经求得答案的一个前缀, 我们从这个最小表示和以 $w^k w'$ 开头的循环表示二者中挑选更优的, 就得到了当前前缀 $uw^k w'$ 的答案, 利用 Z 函数可以 $O(|S|)$ 完成。

有关最小表示的一些性质在第 6 节有所讨论。

5 Lyndon 数组与 Runs

5.1 平方串与 Runs

定义 5.1 (本原串) 如果字符串 S 没有整周期, 那么称 S 是一个本原串。

定义 5.2 (k 次方串) 形如 w^k ($k \in \mathbb{Z}, k \geq 2$) 的字符串称为 k 次方串, 当 $k = 2$ 时也称为平方串。特别地, 当 w 是本原串时, 称 w^k 为本原 k 次方串, 所有 $k \geq 2$ 次方串统称幂串。

在本节中我们期望探究有关一个字符串的平方子串和 k 次方子串的一些性质。

考虑一个最小周期为 p 的字符串 S , 由于对于任意的 $i \leq |S| - p$ 都有 $S_i = S_{i+p}$, 所以对于任意的 $i \leq |S| - 2p + 1$ 都有 $S[i, i + p - 1] = S[i + p, i + 2p - 1]$, 也就是说 $S[i, i + 2p - 1]$ 是一个平方串。同时, 假设 $S[i, i + 2p - 1]$ 不是本原平方串, 这表示它还存在整周期 $p' < p$, 这导出 p' 也是 S 的周期, 从而与 p 是 S 的最小周期矛盾。于是, $S[i, i + 2p - 1]$ 是本原平方串。

与此类似地, 可以说明形如 $S[i, i + kp - 1]$ 的子串是本原 k 次方串。

这启发我们: 一个字符串中的平方子串 (或 k 次方子串) 与上述的这种 S 的结构关系密切。具体地, 我们可以考虑任意字符串 S 的一个子串 $S[l, r]$, 满足它有一个较小的周期 p , 从而 $S[l, r]$ 的任意长度为 $2p$ 的子串都将成为一个平方串。

定义 5.3 (Runs) 对于字符串 S , 如果它的一个子串 $S[i, j]$ 具有最小周期 p , 满足 $2p \leq j - i + 1$, 且 $S_{i-1} \neq S_{i+p-1}$ (或 $i = 1$), $S_{j+1} \neq S_{j+p+1}$ (或 $j = |S|$) (即在满足周期 p 的条件下它是极长的), 那么称三元组 $r = (i, j, p)$ 构成串 S 的一个 run, 记 $e_r = \frac{j - i + 1}{p}$ 为它的指数。 S 的所有 runs 的集合记为 $\text{Runs}(S)$ 。

引理 5.1 周期相同的两个 runs $r_1 = (i_1, j_1, p)$ 和 $r_2 = (i_2, j_2, p)$ 的交长度小于 p 。

Proof. 不妨设 $i_1 < i_2$, 那么显然 $j_1 < j_2$ 。假设它们的交的长度不小于 p , 则 $i_2 \leq j_1 + 1 - p$, 所以在 (i_2, j_2, p) 中有 $S_{j_1+1} = S_{j_1+1-p}$, 这与 (i_1, j_1, p) 是 run 矛盾。 $\square$

于是, 可以得到本原平方串和 run 的如下关系:

引理 5.2 任何一个 run $r = (i, j, p)$ 可以导出 $j - i + 2 - 2p$ 个本原平方串, 即 $S[i, j]$ 的所有长度为 $2p$ 的子串; 同时每个本原平方子串都按照上述规则由唯一的一个 run 导出。

Proof. 引理的前半部分实际上已在前面的分析中说明。

对于一个周期为 p 的本原平方串 $S[u, u + 2p - 1]$, 将其向两侧扩展, 即如果 $S_{u-1} = S_{u+p-1}$ 就将其左端点扩展到 $u - 1$, 以此类推, 最后我们就得到了一个周期为 p 的 run。假设导出 $S[u, u + 2p - 1]$ 的 run 不唯一, 那么这些 runs 的周期都为 p , 且它们的交长度至少为 $2p$, 这与引理 5.1 矛盾, 故命题的后半部分也成立。 $\square$

这条引理是很有用的,它告诉我们要研究本原平方串的结构就只需研究 run 的结构,于是接下来开始讨论 $\text{Runs}(S)$ 的结构和求法。

5.2 Lyndon 根与 Runs 定理

在这一小节中,我们的主要任务是得出一个关于 $\text{Runs}(S)$ 的一个重要命题:

定理 5.1 (Runs 定理) 记 $\rho(n)$ 和 $\sigma(n)$ 分别表示长度为 n 的字符串的 runs 数量最大值和 runs 指数和最大值,有 $\rho(n) < n$, $\sigma(n) < 3n$ 。

我们将引入 Lyndon 数组作为工具进行证明,这一证明来自参考文献 [3]。

记 $<_0$ 和 $<_1$ 为定义在字符集 Σ 上的两种相反的序,即对于任意不同字符 a, b 有 $a <_0 b \iff b <_1 a$ 。据此可以定义出两种不同的字典序,也记为 $<_0$ 和 $<_1$ 。一般情况下 $S <_0 T \iff T <_1 S$,但是当 S, T 中某一个为另一个的前缀时则不是如此。

定义 5.4 (Lyndon 根) 对于 S 的一个 run $r = (i, j, p)$,如果在字典序 $<$ 意义下其长度为 p 的子串 $S[u, u+p-1]$ 为 Lyndon 串,则称 $S[u, u+p-1]$ 是 r 关于 $<$ 的一个 Lyndon 根。

由于 Lyndon 根的长度恰好是 r 的周期大小,所以任何一个 run 的所有 Lyndon 根就是其循环节的最小表示,由于循环节是本原串(否则与 p 是最小周期矛盾),故 r 的 Lyndon 根是存在的,且其各个 Lyndon 根是相等的。

为了在 runs 和 Lyndon 理论之间建立联系,再定义:

定义 5.5 (Lyndon 数组) 对于字符串 S 和 $i \in [1, |S|]$,定义 $l_t(i)$ ($t = 0, 1$) 表示在 $<_t$ 意义下左端点为 i 的最长 Lyndon 子串的右端点,将 $l_t(i)$ 称为 S 的 Lyndon 数组。

下面考虑 Lyndon 数组与 Lyndon 根的联系。令 $\bar{S} = S\$$, 其中 $\$$ 是满足 $\$ <_0 a (\forall a)$ 的一个字符,容易发现 $\text{Runs}(\bar{S}) = \text{Runs}(S)$, 下面讨论的 Lyndon 数组都是 $\bar{S}$ 的 Lyndon 数组。

引理 5.3 $\forall i \in [1, |S|]$, 在 $l_0(i)$ 和 $l_1(i)$ 中恰有一个为 i 。

Proof. 对于 $i \in [1, |S|]$, 设 $j = \min\{x \mid x > i, \bar{S}_j \neq \bar{S}_i\}$, 由 $\bar{S}$ 定义知这样的 j 一定存在。

若 $\bar{S}_j <_t \bar{S}_i$, 首先考虑 $l_t(i)$, 由于 $\bar{S}_j$ 更小所以不可能 $l_t(i) \geq j$, 又 $\bar{S}[i, k]$ ($i < k < j$) 所有字符都相等,一定不是 Lyndon 串,故 $l_t(i) = i$ 。

再考虑 $l_{1-t}(i)$, 显然 $\bar{S}[i, j]$ 在 $<_{1-t}$ 意义下是 Lyndon 串,所以 $l_{1-t}(i) \geq j > i$ 。 $\square$

引理 5.4 对于 S 的任意 run $r = (i, j, p)$, 设 $\bar{S}_{j+1} <_t \bar{S}_{j+1-p}$, 那么对于其任何 Lyndon 根 $S[u, u+p-1]$ 有 $l_t(u) = u + p - 1$ 。

Proof. 根据 Duval 算法的相关知识, 我们已经知道形如 $w^k w'$ ($k \in \mathbb{N}^*$) 的串不是 Lyndon 串, 其中 w' 是 w 的非空前缀。而所有的 $S[u, v]$ ($u + p - 1 < v \leq j$) 都满足这一形式, 所以它们不是 Lyndon 串。另外, 对于 $v > j$, 由于 $\bar{S}_{j+1} <_l \bar{S}_{j+p-1}$, 所以 $S[u + p, v] <_l S[u, v - p]$, 因此 $S[u, v]$ 不是 $<_l$ 意义下的 Lyndon 串。

所以在 $<_l$ 意义下以 u 为左端点的最长 Lyndon 串就是 $S[u, u + p - 1]$ 。 $\square$

如果 $\text{run } r = (i, j, p)$ 的一个 Lyndon 根 $S[u, u + p - 1]$ 满足 $u > i$, 那么称其为真 Lyndon 根, 易证 r 的真 Lyndon 根存在。

引理 5.5 对于 $u \in [1, |S|]$, $\bar{S}[u, l_0(u)]$ 和 $\bar{S}[u, l_1(u)]$ 不可能都作为某个 run 的真 Lyndon 根。

Proof. 假设 $\bar{S}[u, l_0(u)]$ 是 $r_0 = (i_0, j_0, p_0)$ 的真 Lyndon 根, $\bar{S}[u, l_1(u)]$ 是 $r_1 = (i_1, j_1, p_1)$ 的真 Lyndon 根。不妨设 $l_0(u) = u$, $l_1(u) > u$, 那么根据 $\bar{S}[u, l_1(u)]$ 为 Lyndon 串知 $\bar{S}_{l_1(u)} \neq \bar{S}_u$, 但同时有 $\bar{S}_u = \bar{S}_{u-1} = \bar{S}_{u+p-1} = \bar{S}_{l_1(u)}$ (这是因为 r_1 和 r_2 都是 run), 矛盾。 $\square$

引理 5.6 任何两个不同的 runs $r_1 = (i_1, j_1, p_1)$ 和 $r_2 = (i_2, j_2, p_2)$ 的全部真 Lyndon 根的左端点集合不变。

Proof. 假设 r_1, r_2 的真 Lyndon 根左端点集合有交, 即存在 u 使得 $S[u, v_1]$ 和 $S[u, v_2]$ 分别是 r_1, r_2 的 Lyndon 根, 由引理 5.4 知 v_1, v_2 一定是 $l_0(u)$ 或 $l_1(u)$ 之一, 由引理 5.5 知 v_1, v_2 不能分别取 $l_0(u), l_1(u)$, 所以 $v_1 = v_2$, 这导出 $r_1 = r_2$, 矛盾。 $\square$

有了上面三条引理, Runs 定理的证明已经呼之欲出:

设 S 是任意长度为 n 的字符串, 对于 $\text{run } r = (l, r, p)$, 记 $B(r)$ 表示其所有真 Lyndon 根的左端点集合, 我们已经知道对于不同的 r_1, r_2 有 $B(r_1) \cap B(r_2) = \emptyset$, 同时由定义知 $B(r) \subseteq \{2, 3, \dots, |S|\}$, 所以:

$$\sum_{r \in \text{Runs}(S)} |B(r)| \leq n - 1$$

同时 $|B(r)| \geq 1$, 所以:

$$|\text{Runs}(S)| \leq n - 1 < n$$

这正是定理 5.1 的第一部分。

同时, 如果 $r = (i, j, p)$ 的循环节完整出现了 x 次, 那么其真 Lyndon 根至少有 $x - 1$ 个 (注意以 i 开头的 Lyndon 根会被排除), 所以 $|B(r)| \geq \lfloor e_r - 1 \rfloor > e_r - 2$, 故:

$$\sum_{r \in \text{Runs}(S)} (e_r - 2) < \sum_{r \in \text{Runs}(S)} |B(r)| \leq n - 1$$

又由于 $|\text{Runs}(S)| \leq n - 1$, 所以:

$$\sum_{r \in \text{Runs}(S)} e_r = \sum_{r \in \text{Runs}(S)} (e_r - 2) + 2|\text{Runs}(S)| < 3n - 3 < 3n$$

这就证明了定理 5.1。

5.3 Runs 的求解

这里提供两种求解长度为 n 的字符串 S 的全部 runs 的方法。

5.3.1 算法一

枚举周期 p ，任何一个周期为 p 的 run 长度至少为 $2p$ 。

定义下标为 p 的倍数的位置为关键点，则周期为 p 的 run 一定经过了至少 2 个相邻的关键点。于是我们枚举每一对相邻的关键点 $i, i+p$ ，求出 $L_1 = \text{LCP}(S[i, n], S[i+p, n])$ ， $L_2 = \text{LCS}(S[1, i-1], S[1, i+p-1])$ ，如果这两个公共部分在 $[i, i+p-1]$ 这一段上能连起来，说明这就是某个 run 的一个循环节。因此我们检查 $L_1 + L_2$ 与 p 的大小，如果 $L_1 + L_2 \geq p$ 那么就找到了一个周期为 p 的 run $r = (i - L_2, i + p + L_1 - 1, p)$ 。

考虑算法的时间复杂度，利用后缀数组可以实现 $O(1)$ 查询每个 L_1, L_2 ，所以时间复杂度为 $O(n) + O(\sum_{i=1}^n \frac{n}{i}) = O(n \log n)$ 。

这种算法在 [NOI 2016] 优秀的拆分 一题中得到了一定的普及。

5.3.2 算法二

这里重点介绍一种由 5.2 中理论而来的求解 runs 的算法，它相比算法一更加简便。

由于 runs 的 Lyndon 根一定形如 $S[i, l_i(i)]$ ，所以可以首先求出 Lyndon 数组，然后再从所有形如 $S[i, l_i(i)]$ 的子串开始进行扩展，得到所有的 runs。而主要的问题就是如何求解 Lyndon 数组，即每个后缀的最长 Lyndon 前缀。

考虑一种从后往前构造 Lyndon 分解的算法：若当前字符串为后缀 $\text{Suf}(S, i)$ ，我们维护了它的 Lyndon 分解，现在向最前面加一个字符 $c = S_{|S|-i}$ ，得到更长的后缀 $\text{Suf}(S, i+1)$ 的 Lyndon 分解。

以栈的形式存储 $\text{Suf}(S, i)$ 的每个 Lyndon 因子，设从前往后依次为 $L_1, \dots, L_k$ ，加入字符 c 时，先考虑 cL_1 是否是 Lyndon 串，如果是则将它们合并，再考虑 cL_1L_2 ，以此类推，若最后合并到了 $cL_1L_2 \dots L_q$ ，则将这个串加入栈顶，同时这就是 $\text{Suf}(S, i+1)$ 的最长 Lyndon 前缀。我们实际上就是在从后往前模拟不断合并相邻 Lyndon 串的过程。

求出 Lyndon 数组后，与算法一类似，我们枚举所有可能的 $S[i, l_i(i)]$ ，然后分别向前求 $\text{LCS}(S[1, i-1], S[1, l_i(i)-1])$ 和向后求 $\text{LCP}(S[i, n], S[l_i(i), n])$ ，然后将它们拼合，如果长度不小于 $2(l_i(i) - i + 1)$ ，就找到了一个 run。

如果 $O(n)$ 构造后缀数组，则该算法的时间复杂度就是 $O(n)$ 。



在用 runs 解决的实际问题中，几乎都需要考虑平方串（或幂串）的结构。

引理 5.7 (Three Square Lemma) 如果 u^2, v^2, w^2 是本原平方串, 并且 u^2 是 v^2 的前缀, v^2 是 w^2 的前缀, 则 $|w| \geq |u| + |v|$ 。

当 $i + |w| - |v| \leq |v|$ 时有 $v_i = w_i = w_i^2 = w_{i+|w|}^2 = v_{i+|w|-|v|}$, 所以 $|w| - |v|$ 是 v 的周期。

故 $|u|$ 和 $|w|-|v|$ 都是 v 的周期, 假设 $|u|+(|w|-|v|) \leq |v|$ 则根据弱周期引理得 $\gcd(|u|, |w|-|v|)$ 是 v 的周期, 从而是 u 的周期, 且是整周期, 与 u 本原矛盾, 所以 $|u|+(|w|-|v|) > |v|$, 即 $|u|+|w| > 2|v|$.

由 $|u| + |w| > 2|v|$ 可知 $|v| - |u| < |w| - |v|$, 也即 $|s_3| < |s_1|$.

上图(来自参考文献[6])展示了 u^2, v^2, w^2 的形式, 根据红色部分可知 s_3 是 s_2s_3 的前缀, 从而 $|s_2|$ 是 s_2s_3 的周期, 又由右侧红色部分知 s_2s_3 是 $u = s_1s_2$ 的前缀, 于是 $|v| - |u| = |s_3|$ 就是 s_2s_3 的周期, 由弱周期引理知 $\gcd(|s_2|, |s_3|)$ 是 s_2s_3 的周期, 同时 $|s_2|$ 是 u 的周期, $Pre(u, s_2)$ 有整周期 $\gcd(|s_2|, |s_3|)$, 因此 $\gcd(|s_2|, |s_3|)$ 是 u 的周期。

推论 5.4.1 字符串 w 的本原平方前缀为 $O(\log |w|)$ 个。

推论 5.4.2 字符串 w 的本原平方子串为 $O(|w| \log |w|)$ 个。

每个幂串也同样由唯一的 run 导出, 即对于 $\text{run } r = (i, j, p)$, 所有形如 $w[x, x + kp - 1]$ ($i \leq x < x + kp - 1 \leq j, k \geq 2$) 的最小周期为 p 的串都是由 r 导出的幂串。不过幂串的总数量显然没有本原平方串那么好的估计, 因为一个所有字符均一样的字符串的所有长度大于等于 2 的子串都是幂串。



对于 w 的本质不同本原平方子串数量, 也有一个比较理想的结论。

引理 5.8 字符串 w 的本质不同本原平方子串数为 $O(|w|)$ 。

Proof. 考虑最后一次(最靠右的)出现的左端点为 i 的本原平方子串, 假设有三个这样的串 u^2, v^2, w^2 , 不妨设 $|u| \leq |v| \leq |w|$, 根据 Three Square Lemma 有 $|u| + |v| \leq |w|$, 因此 $2|u| \leq |w|$, 也就是说 u^2 是 w 的前缀, 从而在 w^2 的后一个 w 处也有一次出现, 与 i 是其最后一次出现矛盾, 所以至多有两个这样的串。

从 1 到 $|w|$ 枚举 i 并考虑这样的本原平方子串数量, 就不重不漏地计数了所有本质不同本原平方子串, 其数量不超过 $2|w|$, 从而是 $O(|w|)$ 的。□

5.5 相关例题

例题 5.1 (ZJOI 2020 字符串) 给定长度为 n 的字符串 T , 有 q 次询问, 每个询问给定 l, r ($1 \leq l \leq r \leq n$), 求 $T[l, r]$ 有多少个本质不同的平方子串。($1 \leq n, q \leq 2 \times 10^5, |\Sigma| = 2$)

我们首先不考虑“本质不同”这一条件, 只算位置不同的平方子串数量。

将子串 $T[l, r]$ 看成二维平面上的点 (l, r) , 那么对于一个 run $r = (i, j, p)$, 枚举所有 $k \in \mathbb{Z}^+$, 则形如 $T[l, l + 2kp - 1]$ ($i \leq l < l + 2kp - 1 \leq r$) 的子串都是平方子串, 并且每个位置不同的平方子串在这种方式下恰好被统计一次。注意到在 k, p 相同时, 所有符合条件的 $(l, l + 2kp - 1)$ 就是一条斜 45 度线段上的所有整点, 这样的线段总共有至多 $\sum_{r=(i,j,p) \in \text{Runs}(T)} \frac{j-i+1}{p} = \sum_{r=(i,j,p) \in \text{Runs}(T)} e_r$ 条, 根据 Runs 定理, 其为 $O(n)$ 。

再考虑如何去掉本质相同的平方串。我们发现询问时实际上是矩形求和, 所以考虑差分: 如果 $T[i, i + 2p - 1], T[j, j + 2p - 1]$ ($i < j$) 是同一个平方串的相邻两次出现, 我们就增加一个新点 $(i, j + 2p - 1)$, 权值为 -1 , 这样求和时就会自动去重。

同一个 run 中的新点显然关于每个 k 也是一条斜线段, 而对于不同的两个 run 中相同平方串的情况, 我们从左到右考虑所有 runs, 用哈希维护之前已经出现过的所有本质不同平方串集合, 遇到了一个相同的就加入一个新点, 容易发现新点的个数为 $O(|T| \log |T|)$ 。

接下来就是一个斜线段加、单点加, 最后进行若干矩形求和的数据结构问题, 这一部分不再详述, 总复杂度为 $O(|T| \log^2 |T|)$ 。

例题 5.2 (集训队作业 2018 串串划分) 给定长度为 n 的字符串 S , 需要将其划分成一些子串拼接的形式 $S = S_1 + S_2 + \dots + S_k$, 满足 S_i ($i \in [1, k]$) 为本原串, 且 $S_i \neq S_{i+1}$ ($i \in [1, k-1]$), 问划分的方案数, 对 998244353 取模。($1 \leq n \leq 2 \times 10^5$)

首先转化问题: 若子串 $S[l, r]$ 的最小整周期为 p , 循环节出现次数为 $c(l, r) = \frac{r-l+1}{p}$, 则令 $S[l, r]$ 的权值为 $(-1)^{c(l,r)+1}$, 再求所有 $S[1, n]$ 的划分的权值之积之和。

这样转化的好处在于, 若划分中出现了 $S_i = S_{i+1}$, 则将它们合并为 $S_i + S_{i+1}$ 将会得到权值为相反数的一种方案, 而后者不符合 S_i 是本原串的条件, 就这样我们用一个容斥同时满足了两

个条件。

现在就可以直接动态规划了：令 $f(i)$ 表示 $Pre(S, i)$ 的答案，那么：

$$f(i) = \sum_{j=1}^i f(j-1) \times (-1)^{c(j,i)+1}$$

注意到使得 $c(j, i) \neq 1$ 的只有 $S[j, i]$ 是幂串的情况，所以我们首先假设所有 $c(j, i) = 1$ ，进行一次前缀和就能完成的简单求和，然后再重新考虑幂串的贡献。具体地，对于一个 $\text{run } r = (i, j, p)$ ，显然其导出的幂串右端点在 $[i + 2p - 1, j]$ 中，所以只有这些位置的 f 值需要额外考虑，这部分的复杂度也就是 $\sum_{r=(i,j,p) \in \text{Runs}(S)} (j - i + 1 - 2p)$ ，根据上节推论 5.4.2 知其为 $O(|S| \log |S|)$ 。

5.6 Lyndon 树

我们知道每个长度大于 1 的 Lyndon 串 w 可以拆分成两个 Lyndon 串的拼接 $w = u + v$ ，其中 v 是 w 的最小真后缀，将这种拆分称为 w 的标准分解。

所以，我们可以由一个 Lyndon 串 S 构造一棵二叉树：每个结点对应一个 Lyndon 子串，根结点就对应 S 自身，而串 w 对应结点的左右儿子就分别是其标准分解中的 u, v 。

这棵二叉树称为 S 的 Lyndon 树。

如果 S 不是 Lyndon 串，则可以构造其所有 Lyndon 因子的 Lyndon 树，并在一起就得到了 S 的 Lyndon 森林。另一种方法是在其开头加上一个极小字符，得到一个 Lyndon 串，这相当于将前述森林中的各个根串联起来。

5.6.1 Lyndon 树的构建

考虑如何求出给定 Lyndon 串 S 的 Lyndon 树。

令 $n = |S|$ ，取 $S[2, n]$ 的 Lyndon 分解 $S[2, n] = w_1 + \dots + w_k$ ，则容易发现 $S_1 + w_1 + \dots + w_i$ 的最小真后缀（即 $w_1 + \dots + w_i$ 的最小后缀）是 w_i ，因此 S 的 Lyndon 树中，代表 $S_1 + w_1 + \dots + w_i$ 的结点的左儿子为 $S_1 + w_1 + \dots + w_{i-1}$ ，右儿子为 w_i 。那么 S 的 Lyndon 树就是以一条左偏的链将 $w_1, \dots, w_k$ 的 Lyndon 树依次串联。

将这种想法推广，对 S 进行从后往前构造 Lyndon 分解的算法，用一个栈维护当前后缀 $S[i+1, n]$ 的所有 Lyndon 因子，同时维护当前后缀 $S[i+1, n]$ 的 Lyndon 森林，在前加入字符 S_i 时会合并栈顶的一些 Lyndon 因子，我们在 Lyndon 森林中将这因子代表的树根用一条左偏的链串联，最终就完成了 S 的 Lyndon 树构造。

5.6.2 一些性质

为了方便，我们将 S 的 Lyndon 树的叶子从左到右（即按照它们代表的字符）标号为 $1, \dots, |S|$ ，并称一个子树的左右端点为其中叶子标号的最小值和最大值。



引理 5.9 对于字符串 S , 以 i 为左端点的最长 Lyndon 子串对应 Lyndon 树中以 i 为左端点的最长子树。

Proof. 根据 5.6.1 所述的 Lyndon 树构造算法, 结合 5.3.2 中 Lyndon 数组的构造算法, 不难发现它们都代表算法运行到后缀 $S[i, |S|]$ 时栈顶的 Lyndon 串。□

推论 5.6.1 若 S 的子串 $S[l, r]$ 是 Lyndon 串, 则 Lyndon 树上 $l, \dots, r$ 的 LCA 的子树左端点为 l 。

根据引理 5.9 可知 Lyndon 数组完全蕴含在了 Lyndon 树中, 同时根据 Lyndon 数组也可以确定 Lyndon 树的结构。

现在让我们回顾 5.6.1 中的算法, 在加入新字符时我们需要将它和栈顶的一些 Lyndon 因子合并, 假设当前合并出的包含新加入字符的 Lyndon 串是 w_1 , 而 w_1 之后的后缀是 $w_2 + \dots + w_k$, 现在考虑 w_1 和 w_2 能否合并。 w_1 和 w_2 能合并当且仅当 $w_1 w_2$ 是 Lyndon 串, 则 $w_1 w_2 < w_2$, 于是 $w_1 w_2 \dots w_k < w_2 \dots w_k$, 这是两个后缀的比较, 也就是: w_1 与 w_2 能合并当且仅当以 w_1 开头的后缀排名更小。

现在我们忽视字符串 S 本身, 仅仅保留其后缀数组, 重新考虑一遍求 Lyndon 树的算法:

维护一个栈 s , 从后往前枚举 $i = |S|, \dots, 1$, 若栈非空则取出栈顶 x , 如果 $\text{ISA}(i) < \text{ISA}(x)$ 就将 x 弹出并检查新的栈顶, 最后将 i 加入栈中。

这个算法看起来十分眼熟, 事实上其中的栈就是 ISA 数组的单调栈, 根据单调栈与笛卡尔树的相关性质我们直接得到下列性质:

引理 5.10 字符串 S 中, $l(i)$ 表示以 i 为左端点的最长 Lyndon 子串右端点, 则若 j 是满足 $j > i$, $\text{ISA}(j) < \text{ISA}(i)$ 的最小整数, 则 $l(i) = j - 1$; 若这样的 j 不存在, 则 $l(i) = |S|$ 。

推论 5.6.2 字符串 S 的 Lyndon 因子与其 ISA 数组的前缀最小值一一对应。

5.6.3 区间短周期查询

这里介绍一个 Lyndon 树的最典型应用: 区间短周期查询问题, 即给定字符串 S , 若干次询问子串 $S[l, r]$ 的最小周期是否不超过其长度的一半, 如果是则求出这个最小周期。

注意如果周期不超过长度一半, 那么循环节至少完整出现两次, 也就有了一个 run, 所以我们实际上要求一个包含 $[l, r]$ 且周期不超过 $\frac{r-l+1}{2}$ 的 run。

我们首先给出以下算法: 对 S 以 $<_0$ 和 $<_1$ 两种字典序分别建立 Lyndon 树, 并分别求出树上编号为 $l, \dots, \lceil \frac{l+r}{2} \rceil$ 的叶子的 LCA, 检查是否存在以 LCA 的右子树为 Lyndon 根的 run, 如果存在则找到了答案。

下面证明上述算法的正确性:

令 $m = \lceil \frac{l+r}{2} \rceil$, 考虑要求 run $r = (i, j, p)$ 在相应字典序下的包含 m 的 Lyndon 根, 它是某个 Lyndon 树上的一个子树, 设其根为 x , 由引理 5.9 知 x 不可能是父亲的左儿子, 所以 x 是父亲的右儿子 (它显然不是根)。

再考虑求出的 LCA (设为 y), 它和 x 都是叶子 m 的祖先, 并且长度更长 (至少超过了 $S[l, r]$ 的一半长度), 所以它是 x 的祖先。

考虑 y 的右儿子 z , 假设 $z \neq x$, 由于 z 的子树代表 Lyndon 串, 所以 z 的子树右端点一定大于 j (否则就存在周期 p), 而 $S_{j+1} < S_{j+1-p}$ (这是当前字典序的定义), 所以删除 z 子树代表的串中前 p 个字符就得到了一个更小后缀, 与 z 的子树代表 Lyndon 串矛盾。

所以只能 $z = x$, 也就是所求 run 的 Lyndon 根为 LCA 的右儿子。

6 Lyndon 串有关的组合计数

6.1 Necklace 与最小表示

定义 6.1 对于字符串 w , 形如 $Suf(w, i) + Pre(w, |w| - i)$ ($i \in [0, |w| - 1]$) 的串称为 w 的一个循环表示, 其中字典序最小的称为 w 的最小表示, 记为 $\langle w \rangle$ 。若 $\langle w \rangle = w$, 则称 w 是一个 necklace。

与 Lyndon 串有关的组合计数经常要转化为研究 necklace 的问题上来, 所以我们在此首先介绍 Lyndon 串与 necklace 间的一些简单关系。

引理 6.1 每个 Lyndon 串都是 necklace; 每个 necklace 可以唯一地分解为 w^k ($k \in \mathbb{Z}^+$) 的形式, 其中 w 是 Lyndon 串。

Proof. 假设 w 是 Lyndon 串, 且 w 不是 necklace, 则存在 $i \in [1, n-1]$ 使得 $Suf(w, i) + Pre(w, |w| - i) < w$, 这导出 $Suf(w, i) < w$, 与 w 是 Lyndon 串矛盾。

若 w 是 necklace, 设 p 为其最小整周期, 则不同的最小表示共有 p 种 (一种最小表示可以由它的起始位置在循环节中的位置确定)。令 $u = Pre(w, p)$, 易证 $\langle w \rangle = \langle u \rangle^{|w|/p}$, 故 $\langle u \rangle = u$, 则对于每个 $i \in [1, p-1]$ 有 $Suf(u, i) + Pre(u, p-i) < u$, 所以 $Suf(u, i) \leq Pre(u, i)$, 且假设 $Suf(u, i) = Pre(u, i)$ 则 $Pre(u, p-i) < Suf(u, p-i)$, 所以 $Suf(u, p-i) + Pre(i) < u$, 矛盾, 故 $Suf(u, i) < Pre(u, i)$, 故 $Suf(u, i) < u$, 所以 u 是 Lyndon 串。 $\square$

推论 6.1.1 本原的 necklace 是 Lyndon 串; 每个本原串的所有循环表示中恰有一个 Lyndon 串。

6.1.1 $\langle w \rangle$ 的求法

一个稍简单的问题是: 判定 w 是不是 necklace。我们可以直接求 w 的 Lyndon 分解, 只要它形如 u^k , 那就表明 w 是 necklace, 否则 w 不是 necklace。

接下来考虑求 $\langle w \rangle$, 我们有如下结论: 令 ww 的 Lyndon 分解中左端点在 $[1, |w|]$ 中的最后一个 Lyndon 因子左端点为 x , 则 $\langle w \rangle = Suf(w, |w| - x + 1) + Pre(w, x - 1)$ 。于是我们首先求出 w 的最小整周期 p , 令 $u = Pre(w, p)$, 求 uu 的 Lyndon 分解即可得到 $\langle u \rangle$, 而 $\langle w \rangle = \langle u \rangle^{|w|/p}$ (此外, 此问题还有利用后缀自动机的 $O(|w| \log |\Sigma|)$ 算法, 这里不再赘述)。所以:

引理 6.2 可以在 $O(|w|)$ 的时间复杂度内求 $\langle w \rangle$, 以及判定 w 是否是 necklace.

由于 Duval 算法是从前到后的, 所以事实上我们可以判定 w 的每个前缀是否是 necklace.

6.1.2 前驱 necklace 的求法

定义字符串 w 的前驱 necklace (记为 $PL(w)$) 为: 长度等于 $|w|$ 且字典序小于等于 w 的最大 necklace. 由于在后面的问题中常常需要首先对输入字符串 w 进行变为 necklace 的标准化处理, 所以值得在此首先讨论这个问题.

若 w 本身是 necklace, 则情况平凡; 对于另外的情况, 问题的主要部分是要证明下述命题:

引理 6.3 若 w 不是 necklace, 则 $PL(w)$ 一定形如 $Pre(w, i-1) + P(w_i) + z^{|w|-i}$ ($i \in [1, |w|]$), 其中 $P(x)$ 表示字符集中比 x 小的最大字符, z 表示字符集中的最大字符.

Proof. 令 $PL(w) = w'$, 设 w' 与 w 的 LCP 长度为 $t-1$, 则 $Pre(w', t-1) = Pre(w, t-1)$ 且 $w'_t < w_t$. 令 $\bar{w} = Pre(w, t-1) + P(w_t) + z^{|w|-t}$, 显然 $\bar{w} \geq w'$, 因此只要证 $\bar{w}$ 是 necklace 即可说明 $w' = \bar{w}$, 也就是证明了引理.

假设 $\bar{w}$ 不是 necklace, 则存在 $i \in [1, |w|-1]$ 使得 $Suf(\bar{w}, i) + Pre(\bar{w}, |w|-i) < \bar{w}$. 那么 $\bar{w}_1 \geq \bar{w}_{|w|-i+1}$, 又 $\bar{w}_1 \neq z$ (因为 $\bar{w}_1 = P(w_1)$ 或 $\bar{w}_1 = w'_1 \leq w'_t < w_t$ 中有一成立), 所以 $\bar{w}_{|w|-i+1} \neq z$, 也就是 $i > |w|-t$. 再设 $r-1$ 为 $\bar{w}$ 和 w' 的 LCP 长度, 则 $r \geq t$, 因此在进行 $Suf(\bar{w}, i)$ 与 $Pre(\bar{w}, i)$ 的比较时会比较 $\bar{w}_{i+r-|w|}$ 和 $\bar{w}_r$ (此前部分都与 w' 中相同). 由于 w' 是 necklace, 这推出 $\bar{w}_{i+r-|w|} = w'_{i+r-|w|} \leq w'_r < \bar{w}_r$, 也就是 $Pre(\bar{w}, i) < Suf(\bar{w}, i)$, 与假设矛盾. 所以 $\bar{w}$ 是 necklace, 进而它是 $PL(w)$. $\square$

于是, 我们立刻得到一种 $O(|w|^2)$ 算法计算 $PL(w)$: 首先判断 w 是否是 necklace, 若是则 $PL(w) = w$; 否则依次枚举 $i = |w|, \dots, 1$, 判断 $Pre(w, i-1) + P(w_i) + z^{|w|-i}$ 是不是 necklace, 若是则该串为 $PL(w)$.

当然, 判定部分是可以优化的, 注意到我们需要判定是否是 necklace 的串有一个长度为 $i-1$ 的与 w 的公共前缀, 同时第 $i+1$ 个字符及之后全为 z . 如果 $Pre(w, i-1)$ 不是一个近似 Lyndon 串, 那么不管后面的情况如何都不可能是 necklace 了; 如果 $Pre(w, i-1)$ 是近似 Lyndon 串, 我们在后面加上字符 $P(w_i)$, 若还是近似 Lyndon 串, 再向后加入 $|w|-i$ 个 z , 显然这还是近似 Lyndon 串, 因此我们只要求其周期是不是 $|w|$ 的因数. 考虑 Duval 算法过程, 设加入 $P(w_i)$ 后当前的近似 Lyndon 串为 $u^k u'$ (其中 u 为 Lyndon 串, u' 为其真前缀), 那么接下来只需要看后面的 $|w|-i$ 个 z 是不是与 u 中 u' 得到的后缀能完全匹配上 (即这个后缀中长度为 $|w|-i$ 的前缀都是 z) 即可. 若能完全匹配上, 则最终周期为 $|u|$; 否则从不能匹配的位置开始, 整个串就成为一个 Lyndon 串, 最后周期为 $|w|$.

另一种思路是直接向字符集中加入一个新字符, 钦定它为最大字符 z , 但实际上这已经不是原本字符集上的字符串了, 并且在计数问题中这样做不利于此后的处理, 所以这里所说的是多一些讨论的不容易出错的方案.

上述过程只需对 w 进行一遍 Duval 算法即可完成, 所以有:

引理 6.4 可以在 $O(|w|)$ 时间复杂度内计算 $PL(w)$ 。

6.2 Lyndon 串计数的基本方法

在不加说明的情况下, 以下讨论的字符集都预设为 Σ 。

为了计数长度固定为 n 的 Lyndon 串, 我们此前已经建立了一个映射 $w \rightarrow \langle w \rangle$, 对于每个 Lyndon 串, 它恰好是 n 个本原串的像, 所以求 Lyndon 串的数量可转化为求本原串的数量。

我们首先从最简单的情形开始考虑, 即没有任何特殊限制的情形。设 $L(n)$ 表示长度为 n 的 Lyndon 串数量, $P(n)$ 表示长度为 n 的本原串数量。根据推论 6.1.1 知:

$$L(n) = \frac{1}{n} P(n)$$

我们知道长度为 n 的字符串数量为 $|\Sigma|^n$, 然而另一种算法是枚举字符串的最小整周期, 据此:

$$|\Sigma|^n = \sum_{d|n} P(d)$$

对此式运用莫比乌斯反演得到 $L(n)$ 的计算公式:

$$L(n) = \frac{1}{n} P(n) = \frac{1}{n} \sum_{d|n} \mu\left(\frac{n}{d}\right) \times |\Sigma|^d$$

现在已经知道如何计算长度为 n 的 Lyndon 串数量, 但这只是最简单的结果之一。接下来我们要对这个式子进行简单的扩展, 使得它可以支持字典序限制下的 Lyndon 串计数问题。

6.3 Lyndon 串排名

这一节的剩余部分, 我们主要讨论的是如下问题及其扩展: 给定长度为 n 的字符串 w , 求有多少长度为 n 的 Lyndon 串字典序小于等于 w , 答案记为 $\text{Rank}(w)$ 。由于当 w 为 Lyndon 串时其实这就是在求 w 在全长为 n 的 Lyndon 串中的字典序排名, 所以称为 Lyndon 串排名问题。

在 OI 中, 组合计数往往在 $\mathbb{F}_p$ 下计算, 所以这里仅考虑模意义下的算法, 此算法综合参考文献 [4] [7] 得来。

6.3.1 算法框架

由于 Lyndon 串是 necklace, 所以我们可以令 $w \leftarrow PL(w)$, 而不影响答案, 因此下面我们假定 w 是 necklace。

类似 6.2 中的方法, 我们定义 $P(d), P'(d)$ ($d|n$) 分别表示满足 $\langle u \rangle^{n/d} \leq w$ 的长为 d 的任意串 u 的数量及本原串 u 的数量, 那么有:

$$\begin{aligned}\text{Rank}(w) &= \frac{1}{n} P'(n) \\ P(x) &= \sum_{d|x} P'(d) \quad (x|n) \\ \Rightarrow \text{Rank}(w) &= \frac{1}{n} \sum_{d|n} \mu\left(\frac{n}{d}\right) \times P(d)\end{aligned}$$

$P(d)$ 的性质不好, 但可以对其运用下述性质进行转化。

引理 6.5 若 w 是长为 n 的 necklace, u 是长为 d ($d|n$) 的字符串, 则 $u^{n/d} \leq w \iff u \leq \text{Pre}(w, d)$ 。

Proof. 先证左推右。假设 $u > \text{Pre}(w, d)$ 则 u 的某一位比 w 的对应位大, 则显然 $u^{n/d} > w$, 矛盾。

再证右推左。由 $u \leq \text{Pre}(w, d)$ 知 $u^{n/d} \leq \text{Pre}(w, d)^{n/d}$, 又因 w 是 necklace, 故 w 的任意长度为 d 的子串不大于 $\text{Pre}(w, d)$, 所以 $\text{Pre}(w, d)^{n/d} \leq w$, 所以 $u^{n/d} \leq w$ 。□

因此 $P(d)$ 的定义可以改为: 长为 d 且最小表示的字典序小于等于 $\text{Pre}(w, d)$ 的字符串数量。

6.3.2 识别自动机

现在的目标是对满足一定限制的字符串计数, 不难想到用自动机这一工具进行辅助。

定义 6.2 (匹配自动机) 对于长度为 n 的字符串 w , 其匹配自动机 (或 KMP 自动机) 是一个包含 $n+1$ 个状态的 DFA, 将状态分别编号为 $0, 1, \dots, n$, 起点为 0 。满足对于任意字符串 u , 其在自动机上转移到的状态编号 i 恰好等于 u 的后缀与 w 的前缀的最大匹配长度, 即最大的自然数使得 $\text{Suf}(u, i) = \text{Pre}(w, i)$ 。

熟悉 KMP 算法的读者应当对此十分了解, 我们在此只简单介绍其结构: 令 fail_i 表示 $\text{Pre}(w, i)$ 的最长 Border 长度, 则状态 i 的字符为 x 的转移为:

$$\text{tr}(i, x) = \begin{cases} i+1 & (i < |w| \wedge w_{i+1} = x) \\ 0 & (i = 0 \wedge w_{i+1} \neq x) \\ \text{tr}(\text{fail}_i, x) & (i = |w| \vee (i \neq 0 \wedge w_{i+1} \neq x)) \end{cases}$$

下面我们希望对某个 necklace w 的匹配自动机进行改造, 使得它能够用于识别最小表示小于等于 w 的串。在此之前我们需要一些理论基础。

引理 6.6 若 w 是长度为 n 的 necklace, u 是长度同样为 n 的串, 则 $\langle u \rangle \leq w$ 当且仅当 u^∞ 中存在一个不为 w 真前缀的子串 v 满足 $v \leq w$ 。

Proof. 显然 $\langle u \rangle = w \iff \exists v = w$, 所以下面只考虑 $\langle u \rangle < w$ 与 $v < w$ 这种情况。

由于 $\langle u \rangle$ 是 u^∞ 的子串且小于 w , 所以左推右是显然的: 令 $v = \langle u \rangle$ 即可; 而右推左只需注意由于 v 不是 w 的前缀, 所以 v 后接任何字符串都小于 w , 若 $|v| \leq |w|$ 则以 v 开头的 u 的循环表示就小于 w , 而若 $|v| > |w|$ 则由 $v < w$ 知 w 不可能是 v 的前缀, 故 $Pre(v, n) < w$. $\square$

推论 6.3.1 引理 6.6 中的 u^∞ 可以改成 u^k , 其中 k 是大于等于 2 的整数。

那么现在的限制就是: 在 u^∞ 中存在着小于等于 w 的子串。设 v 是其中最短的一个, 根据上面的证明其实可以看出 $|v| \leq |w|$, 并且进一步地, v 还必定是下述集合 $R(w)$ 的元素:

定义 6.3 对于长为 n 的 necklace w , 定义所有形如 $Pre(w, i) + x$ ($i < n, x \in \Sigma, x < w_{i+1}$) 的串和 w 本身构成的字符串集合为 $R(w)$ 。

于是现在我们的目的就是设计一个 DFA, 识别至少包含 $R(w)$ 中的一个元素为子串的字符串, 随后的问题就是设法计数使得 u^∞ 能被识别的 u 的数量。

此处先给出结论: 令 $n = |w|$, 这个 DFA 包含 $n + 1$ 个结点, 编号为 $0, 1, \dots, n$, 转移如下:

$$tr(i, x) = \begin{cases} i + 1 & (i < n \wedge w_{i+1} = x) \\ 0 & (i < n \wedge w_{i+1} < x) \\ n & (i = n \vee (i < n \wedge w_{i+1} > x)) \end{cases}$$

起点为 0, 最终转移到 n 的字符串就是包含 $R(w)$ 中至少一个元素为子串的字符串。将这个 DFA 称为 w 的识别自动机, 下面要证明它能正确识别所有包含 $R(w)$ 中元素的字符串。

不难看出识别自动机和匹配自动机非常类似, 我们首先就要揭示它们的联系。

引理 6.7 设 w 是长度为 n 的 necklace, $f(i, x)$ 和 $g(i, x)$ 分别是其匹配自动机和识别自动机的转移, 则 $g(i, x) \neq n \Rightarrow g(i, x) = f(i, x)$ 。

Proof. 若 $x = w_{i+1}$, 则显然 $f(i, x) = g(i, x) = i + 1$; 而当 $x > w_{i+1}$ 时 $g(i, x) = 0$, 只需证明在这种情况下 $f(i, x)$ 也为 0。

假设 $f(i, x) = j + 1 \neq 0$, 则说明 $x = w_{j+1}$, 并且 $Pre(w, j)$ 是 $Pre(w, i)$ 的 Border, 也即 $w[1, j] = w[i - j + 1, i]$, 又因为 $x > w_{i+1} \Rightarrow w_{j+1} > w_{i+1}$, 故 $w[1, j + 1] > w[i - j + 1, i + 1]$, 所以从 w_{i-j+1} 开始的循环表示小于 w , 这与 w 是 necklace 矛盾。 $\square$

推论 6.3.2 若字符串 u 在 w 的识别自动机上转移到点 $x \neq n$, 则 x 就是 u 的后缀与 w 的前缀匹配的最大长度。

现在我们可以证明识别自动机的正确性了: 显然被识别的 (转移到 n 的) 字符串都包含了 $R(w)$ 中的元素为子串 (例如, 从它最后一次到状态 0 直到第一次到状态 n 之间的这段子串); 而在到状态 n 之前识别自动机和匹配自动机的行为完全相同, 所以某个字符串中只要包含了 $R(w)$ 中的元素 u 为子串, 那么类似引理 6.7 的证明就可以说明在 u 的最后一位处一定能转移到 n (具

体地, 若 $u \neq w$, 设转移完了 u 的最后一位之前的部分, 到达状态 i , 那么根据 necklace 的性质有 $u_{|u|} < w_{|u|} \leq w_{i+1}$; $u = w$ 的情况类似)。

6.3.3 $P(d)$ 的计算

为了方便, 假设现在计算的是 $P(n)$, 即给定长为 n 的 necklace w , 求有多少长度为 n 的字符串 u 满足 $\langle u \rangle \leq w$ 。在 $d \neq n$ 时我们令 $w \leftarrow \text{PL}(\text{Pre}(w, d))$ 即可变成同样形式的大小为 d 的问题。

整理一下现在的问题被转化成了什么: 我们要计算满足 u^∞ 能在 w 的识别自动机上转移到状态 n 的 u 的数量。做补集转化, 求出不能转移到状态 n 的 u 的数量, 再用 $|\Sigma|^n$ 去减即可。

引理 6.8 对于充分大的 j , $\text{Pre}(u^\infty, j)$ 和 $\text{Pre}(u^\infty, j + |u|)$ 在 w 的识别自动机上转移到的状态相同。

Proof. 若 u^∞ 最终能转移到 n , 则对于充分大的 j 都有 $\text{Pre}(u^\infty, j)$ 转移到 n , 此时结论平凡。

否则, 识别自动机和匹配自动机等价, 我们知道 $\text{Pre}(u^\infty, j)$ 转移到的状态就是 $\text{Pre}(u^\infty, j)$ 的后缀和 w 的前缀的最大匹配长度, 而 $\text{Pre}(u^\infty, j + |u|)$ 可以看成是在 $\text{Pre}(u^\infty, j)$ 的最前面加了个 u , 当 j 充分大时这对其后缀与 w 的匹配显然没有影响, 所以转移到的状态是相同的。□

下面考虑的都是 u^∞ 无法转移到 n 的 u 。我们定义当 j 充分大时 $\text{Pre}(u^\infty, j|u|)$ 转移到的状态为 u 的特征状态。根据引理 6.8 知 u^∞ 在识别自动机上的匹配最终是以 $|u|$ 为周期的, 每个周期相当于是从 u 的特征状态出发的一个环。

所以, 满足条件的 u 与一个由特征状态 t 和以 t 为起点的一个自动机上长度为 $|u|$ 的环 C 构成的二元组 (t, C) 一一对应 (注意 C 中的起点 t 有特殊标记, 即使 t 和不考虑起点的环 C 都相同, 只要这个 t 在 C 上的位置不同, 还是对应不同的二元组)。

可以更加严谨地说明这一对应是一一对应: 由 u 得到 (t, C) 的过程就是上面说的 u^∞ 匹配的循环节; 而由 (t, C) 得到 u 的过程是首先由 C 得到 u 的一个循环表示, 再由 t 得到 u 的起始位置。

将识别自动机上形如 $i \rightarrow i+1$ 的边称为正边, 形如 $i \rightarrow 0$ 的边称为反边, 则从每个 $i < n-1$ 出发恰有一条正边, 从 $i < n$ 出发的反边数量设为 b_{i+1} (实际上就是比 w_{i+1} 大的字符数量)。我们考虑环 C 的结构, 它一定由若干个经过 0 的子环拼成, 每个子环形如 $0 \rightarrow 1 \rightarrow 2 \rightarrow \dots \rightarrow a_i \rightarrow 0$, 经过 a_i 条正边和 1 条反边。

先考虑比较简单的情形: 此时设 f_i 表示长度为 i 的环 C 的数量, 那么我们枚举其中第一个子环大小, 就有:

$$f_0 = 1, \quad f_i = \sum_{j=1}^{\min(n,i)} b_j f_{i-j}$$

设 F, B 分别为序列 f_i, b_i 的一般型生成函数, 则:

$$F = 1 + BF \implies F = \frac{1}{1-B}$$

从而可以利用形式幂级数求逆在 $O(m \log m)$ 的时间复杂度内算出每个 f_i ($i \leq m$)。

但是 t 不一定为 0, 不能直接用 f_n 作为答案。我们可以枚举包含 t 的子环长度 i , 则 t 就有 i 种选法 (子环为 $0 \rightarrow 1 \rightarrow \dots \rightarrow i-1 \rightarrow 0$, 则 t 可以等于其中的 $0, 1, \dots, i-1$), 因此答案是:

$$Ans = \sum_{i=1}^n i \times b_i \times f_{n-i}$$

最后 $P(n) = |\Sigma|^n - Ans$, 所以求出所有 $f_i (i \leq n)$ 后就可以 $O(n)$ 计算答案, 于是我们得到了 $O(n \log n)$ 计算 $P(n)$ 的算法。

6.3.4 算法总结

上面已经介绍了在模意义下计算 $\text{Rank}(w)$ 的算法的全过程。

首先利用莫比乌斯反演转化为求每个 $P(d)$, 而求 $P(d)$ 时先 $O(d)$ 求出 $\text{PL}(\text{Pre}(w, d))$, 再利用识别自动机的结构列式, 使用 FFT 和幂级数科技加速半在线卷积 $F = 1 + BF$, 即在 $O(d \log d)$ 的复杂度内完成了 $P(d)$ 的计算, 所以算法的总复杂度为:

$$O\left(\sum_{d|n} d \log d\right) = O(\sigma_1(n) \times \log n) = O(n \log^2 n)$$

可以对该算法进行一些简单的扩展, 例如考虑 6.3.3 中的递推式 $f_i = \sum_{j=1}^{\min(n,i)} b_j f_{i-j}$, 这显然是一个常系数齐次线性递推, 利用多项式取模可以在 $O(n \log n \log m)$ 的复杂度内算出 $f_{m-n}, f_{m-n+1}, \dots, f_{m-1}$, 从而可以算出 $P(m)$ 。因此在计算 $P(m)$ 时只要 necklace w 的有效位数 (即除去末尾的所有 z) 只有 n 位, 则复杂度仅为 $O(n \log n \log m)$, 这可以用来处理目标字符串长度很长而给定的 w 较短的情况。

6.4 相关例题

例题 6.1 (CTS 2019 重复) 给定长度为 n 的字符串 w , 求有多少长度为 m 的字符串 u 满足 u^∞ 中存在一个长度为 n 且小于 w 的子串。 ($1 \leq n, m \leq 2000$)

设 u 的最小表示的起始位置为 u_x , 不难证明: 如果 u^∞ 中存在长度为 n 且小于 (或小于等于) w 的子串, 则以 u_x 为起始位置的长度为 n 的子串一定是其中之一。

因此我们已经会解决 $n = m$ 的情形: 只需要令 $w \leftarrow \text{PL}(w^-)$, 再求 $P(n)$ 即可, 这里 w^- 表示 w 的前驱 (即长度为 n 且小于 w 的最大字符串)。

对于 $n < m$ 的情形, 只需在 w^- 后加一些 z 使其长度为 m , 再应用 $n = m$ 的方法, 这是因为 $\langle u \rangle$ 小于等于新的 w^- 当且仅当其长度为 n 的前缀小于等于原来的 w^- 。

对于 $n > m$ 的情形, 我们分别考虑 $\langle u \rangle < \text{Pre}(w, m)$ 和 $\langle u \rangle = \text{Pre}(w, m)$, 前者只需截取 $\text{Pre}(w, m)$ 并应用 $n = m$ 的方法; 而后者由于已经知道 $\langle u \rangle$, 所以只需对这种给定的最小表示判断是否合法以及计数对应 u 的数量, 这是容易的。

总时间复杂度为 $O((m+n) \log(m+n))$ 。



7 总结

本文从 Lyndon 分解, Runs 定理, Lyndon 串计数等不同角度揭示了 Lyndon 串的重要组合性质。某些性质和应用实际上是有趣且深刻的, 比如本文大致介绍的 Lyndon 树解区间周期, 还有本文并未介绍的最小 DeBruijn 序列的 Lyndon 分解, 以及 Lyndon 串与不可约多项式的对应等。希望本文能够让更多人了解到 Lyndon 理论的奇妙之处, 也希望能起到抛砖引玉的作用, 让更多关于 Lyndon 理论的研究进入 OI 视野, 在 OI 比赛中出现相关的好题。

8 致谢

- 感谢中国计算机学会提供学习和交流的平台。
- 感谢华东师范大学第二附属中学的金靖老师的关心与指导。
- 感谢与我交流相关内容的同学们的帮助。
- 感谢父母的关心与支持。

参考文献

- [1] 胡昊, 《浅谈 Lyndon 分解》, IOI 2021 集训队论文集
- [2] Tomasz Kociumaka (2016). Minimal Suffix and Rotation of a Substring in Optimal Time. arXiv:1601.08051
- [3] Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, Kazuya Tsuruta (2015). The "Runs" Theorem. arXiv:1406.0263
- [4] Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter (2015). Efficient Ranking of Lyndon Words and Decoding Lexicographically Minimal de Bruijn Sequence. arXiv:1510.02637
- [5] zghtyarecrenj. 浅谈从 Lyndon Words 到 Three Square Lemma, <https://www.luogu.com.cn/blog/zght/lyndon>
- [6] command_block. Lyndon & Runs. <https://www.luogu.com.cn/blog/command-block/lyndon-runs>
- [7] rqy. LOJ3123 [CTS2019] 重复, <https://rqy.moe/Solutions/loj3123/>



《可爱多的字符串》命题报告

成都石室中学 冯施源

摘要

本文将主要介绍本人在 UOJ 举办的候选队互测¹中命制的一道字符串题目²的解法和命题背景。并另外给出了三道例题以总结并介绍后缀树配合 *dsu on tree* 在一些字符串问题上的应用。

1 题目

1.1 一些约定

对于给定长度为 n 的字符串 S , 令:

$S[i]$ 表示 S 的第 i 个字符, 其中 $1 \leq i \leq n$;

$S[l, r]$ 表示 S 中第 l 个字符到第 r 个字符组成的串;

$lcp(i, j)$ 表示串 $S[i, n]$ 与 $S[j, n]$ 的最长公共前缀;

$lcs(i, j)$ 表示串 $S[1, i]$ 与 $S[1, j]$ 的最长公共后缀;

若 S 满足 $S[1, i] = S[n - i + 1, n]$, $1 \leq i \leq n$, 则称 $S[1, i]$ 是 S 的一个 *border*。

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

若无特殊说明, 给定的字符串 S 长度默认为 n 。

1.2 题目描述

对于一个长为 n 的串, *kmp* 算法会对 $i \in [1, n]$ 求出 $next_i$ 表示 $\max\{x \mid x < i, S_{1 \dots x} = S_{i-x+1 \dots i}\}$ 。定义一个串的 *next* 树为每个 i 向 $next_i$ 连边后形成的以 0 为根的树, 定义 0 的深度为 0。给一个长为 n 的串 S , 以及一个数组 W , m 次询问, 每次给出 l, r , 问当 $S' = S[l, r]$, $W' = \{w_l, w_{l+1}, \dots, w_r\}$ 时, *next* 树的 $\sum_{i=1}^{n'} dep_i \times w_i$ 。字符集为 $\{0, 1\}$ 。

¹<https://uoj.ac/contest/73>

²<https://uoj.ac/problem/700>



1.3 数据范围

子任务 1 (5 分): $n, m \leq 10^4$ 。

子任务 2 (10 分): $n, m \leq 10^5, r = n, w_i = 1$ 。

子任务 3 (20 分): $n, m \leq 10^5, r = n$ 。

子任务 4 (10 分): $n, m \leq 10^5$, 字符串的每个字符在 $\{0, 1\}$ 中随机, $w_i = 1$ 。

子任务 5 (10 分): $n, m \leq 10^5$, 字符串的每个字符在 $\{0, 1\}$ 中随机。

子任务 6 (5 分): $n, m \leq 10^5, w_i = 1$ 。

子任务 7 (20 分): $n, m \leq 10^5$ 。

子任务 8 (20 分): $n, m \leq 2 \times 10^5$ 。

时空限制: 6s, 1G。

1.4 得分分布

有 2 名选手获得 80 分, 1 名选手获得 55 分, 1 名选手获得 40 分, 1 名选手获得 35 分, 8 名选手获得 25 分, 1 名选手获得 15 分, 11 名选手获得 5 分。

2 算法介绍

2.1 算法 1

暴力使用 *kmp* 算法求出每个询问的答案, 复杂度 $O(nq)$, 期望得分 5 分。

2.2 算法 2

2.2.1 后缀树

我们把一个字符串的倒序建出的后缀自动机的树结构称为后缀树。令:

fa_i 表示节点 i 在树上的父亲。

len_i 表示节点 i 代表的串的最长长度;

$Right_i$ 表示节点 i 代表的串在倒序串中出现位置的右端点集合 (原串中的左端点集合);

pt_i 表示后缀 $S[i, n]$ 在后缀树上对应的节点。

我们知道有 $lcp(i, j) = len_{lca(pt_i, pt_j)}$ ³。

³关于后缀自动机和后缀树可以参见参考文献 [1] 和 [2]。

2.2.2 算法 2

这个算法可以解决 $r = n, w_i = 1$ 的情况。

我们考虑将计算 $next$ 树上每个点的深度之和变成每个点的子树大小之和。考查每个结点被计算的次数，这显然是等价的。考察 $next$ 树的性质，若 $S[l, x]$ 是 $S[l, y] (y \geq x)$ 的 $border$ ，则 y 将出现在 x 的子树中， y 也唯一对应了 $S[l, x]$ 在 $S[l, n]$ 中的一个出现位置（即 $S[y - x + l, y]$ ），故一个节点 x 的子树大小即为串 $S[l, x]$ 在 $S[l, n]$ 中的出现次数。

现在要统计 $S[l, n]$ 的每个前缀在 $S[l, n]$ 中的出现次数，我们从后向前建立后缀自动机，那么答案即为后缀树上 pt_l 到根的每一个点 i 的 $(len_i - len_{fa_i}) \times |Right_i|$ 。

使用 LCT 维护，时间复杂度 $O(n \log n)$ ，期望得分 15 分。

2.3 算法 3

这个算法可以解决数据随机， $w_i = 1$ 的情况。

考虑延续算法 2，我们要求出 $S[l, r]$ 的每个前缀在 $S[l, r]$ 中的出现次数，找到 $S[l, r]$ 在后缀树上对应的点 x 。对于 x 到根路径上的每一个点 y ，枚举一个长度 $len' \in [len_{fa_y} + 1, len_y]$ ，那么就在其子树查询 $Right_y$ 中 $[l, r - len' + 1]$ 的个数。可以使用线段树合并进行查询。

取 $B > 2 \log n + \log \frac{1}{\epsilon}$ ，可以证明任意两个长为 B 的子串相同的概率小于 $1 - \epsilon$ 。

预处理长度在 $[1, B]$ 的串的 $hash$ 值，枚举 len' 时，查询与 $S[l, l + len' - 1]$ $hash$ 值相同的子串左端点在 $[l, r - len' + 1]$ 中的个数即可。

对于数据随机的情况，由于 len' 的范围为 $O(\log n)$ ，故时间复杂度为 $O(n \log n + q \log^2 n)$ 。结合算法 2，期望得分 25 分。

2.4 算法 4

这个算法可以解决数据随机的情况。

同样使用算法 3 中 $hash$ 的做法，枚举 len' 时，查询与 $S[l, l + len' - 1]$ $hash$ 值相同的子串左端点在 $[l, r - len' + 1]$ 中的 $\sum w_i$ 即可。

若使用后缀树查询，设当前节点为 x ，枚举的长度为 len' ，则要查询 $\sum_{i \in [l, r - len' + 1]} w_{i + len' - 1} [i \in Right_x]$ 。我们任选一个 $t \in Right_x$ ，正序建立后缀自动机，在其树结构上使用倍增找到 $S[t, t + len' - 1]$ 对应的点 y 。

我们知道， $Right_x$ 和 $Right_y$ 是一一对应的，即每个 $p \in Right_x$ 对应一个 $q \in Right_y$ 使得 $p + len' - 1 = q$ ，故我们在 y 的子树中查询 $\sum_{i \in [l + len' - 1, r]} w_i [i \in Right_y]$ 即可。

我们将这种方法称为对称后缀树。

时间复杂度 $O(n \log n + q \log^2 n)$ 。结合算法 2 期望得分 35 分。

2.5 算法 5

这个算法可以解决 $r = n$ 的情况。

考虑修改算法 2, 对于 $S[j, k] = S[l, l + k - j]$, 有 w_k 的贡献。那么我们可以求出 w 的前缀和 S 并枚举 j , 我们知道其应有 $S_{j+lcp(l,j)-1} - S_{j-1}$ 的贡献。

下面考虑求 $\sum_{j=l+1}^n S_{j+lcp(l,j)-1}$ 。

考虑树链剖分, 对于一条特定的重链, 设从 pt_j 开始向上走到根的路径上第一次到这个重链上时遇到的点为 x_j , 定义 len'_j 以表示 len_{x_j} 。

对于 x 的重儿子 son_x 的子树, 同样使用对称后缀树的方法, 但是要做一定的修改。因为当前求和的式子为 $\sum w_{j+len_x-1} (j > l, j \in Right_{son_x})$ 。

我们考虑一个 $t \in Right_{son_x}$, 与 $S[t, t + len_{son_x} - 1]$ 出现的左端点相同的串有 $S[t, t + len_x], S[t, t + len_x + 1], \dots, S[t, t + len_{son_x} - 1]$ 。注意到我们只需要对任意 $t \in Right_{son_x}$ 找到 $S[t, t + len_x]$ 在对称后缀树上对应的节点 y , 并查询 $\sum w_{p-1} (p \in Right_y, p > l + len_x)$ 即可。

剩下的情况, 我们要查询一个重链的前缀的所有轻子树中的 $w_{j+len'_j-1} (j > l)$ 。

使用 *dsu on tree*⁴, 按重链从上到下的顺序扫描, 到一个点时就将其轻子树中的所有 j 按 j 为下标, $w_{j+len'_j-1}$ 为权值加入树状数组。由于每次只会暴力递归轻子树, 所以一个点被算到的次数为其到根链上的轻边个数, 故总的加入次数为 $O(\log n)$ 。

最后, 要减去 pt_l 自己的那个轻子树的贡献, 仍然使用对称后缀树的方法做子树查询。

复杂度 $O((n + q) \log^2 n)$ 。结合算法 4 期望得分 55 分。

2.6 算法 6

这个算法可以解决 $w_i = 1$ 的情况。

继续算法 5, 我们发现要求的式子就是 $\sum_{j=l}^r \min(lcp(l, j), r - j + 1)$ 。

此时问题变成了 CF1098F⁵。因此, 在这里我们只大致阐述官方题解⁶的做法。

延续算法 3, 枚举一个点 x 以及 len' , 查询 $Right_x$ 中 $[l, r - len' + 1]$ 的个数, 我们分别计算 $[1, r - len' + 1]$ 以及 $[1, l]$ 的个数。

对于 $[1, r - len' + 1]$ 的部分。考察一条重链, 设链顶为 top , 那么在这条重链上, 我们需要对应枚举 $len' \in [len_{fa_{top}} + 1, len'_l]$ 。设 $L = len_{fa_{top}} + 1$, 考虑一个 x 会在枚举哪些 len' 的时候会被计算, 我们看作平面上有若干个点: $(x + L, L), (x + L + 1, L + 1), \dots, (x + len'_x, len'_x)$, 那么 x 贡献的次数就是这些点中满足 $x_i \leq r + 1, y_i \leq len'_l$ 的点 (x_i, y_i) 的个数。注意到这若干个点构成一条线段, 我们将其差分为两条射线, 用 (x_i, y_i) 这个二元组表示射线 $(x_i, y_i), (x_i + 1, y_i + 1), \dots$ 。

对于查询, 讨论是 x_i 先撞上 $r + 1$ 还是 y_i 先撞上 len'_l 。即, 当 $r + 1 - x_i \leq len'_l - y_i$ 时, 查询的值即为 $\sum_i (r + 2 - x_i) [x_i \leq r + 1]$, 反之, 查询的值为 $\sum_i (len'_l - y_i + 1) [y_i \leq len'_l]$ 。按 $x_i - y_i$

⁴不熟悉 *dsu on tree* 的同学可以参见参考文献 [3]

⁵<https://codeforces.com/problemset/problem/1098/F>

⁶见参考文献 [4]

排序进行扫描线并使用一个树状数组就可以解决。

对于 $[1, l]$ 的部分, 在考虑 x 贡献几次时, 看成 $(x, L), (x, L+1), \dots, (x, len'_x)$ 这些点, 其贡献的次数即为这些点中满足 $x_i < l, y_i \leq len'_l$ 的个数, 同样使用扫描线以及树状数组解决。

复杂度 $O((n+q)\log^2 n)$, 结合算法 5 期望得分 60 分。

2.7 算法 7

这个算法可以解决 $n, m \leq 10^5$ 的情况。

改进算法 5, 设 w 的前缀和为 S , 将要求的式子写为 $\sum_{j \in (l, r]} S_{\min(lcp(l, j) + j - 1, r)} - S_{j-1}$ 。

同样使用 *dsu on tree*, 设 pt_l 到根的链与重链的交为 x , 那么我们需要求出所有 pt_j 到根的链与重链的交在 x 以及其上方的 j 的 $S_r \times \sum_{j \in (l, r]} [len'_j + j - 1 \geq r] + \sum_{j \in (l, r]} [len'_j + j - 1 < r] S_{len'_j + j - 1}$ 。

使用从上至下的顺序扫描线, 对于 x 处的查询, 是一个矩阵数点问题。由于插入次数以及查询次数分别为 $O(n \log n), O(q \log n)$, 故我们使用树套树, 复杂度 $O((n+q)\log^3 n)$ 。

对于 $O(\log n)$ 次询问子树 x 的 $\sum_{j \in (l, r]} S_{\min(j + len_{fa_x} - 1, r)}$, 同样使用算法 5 中的对称后缀树进行子树查询。期望得分 80 分。

2.8 算法 8

考虑优化算法 7。我们先暂不要求 j 与重链的交在 x 的上方, 对于交在 x 下方的点, 由于其贡献为 $\sum_{j \in (l, r]} S_{\min(j + len_x - 1, r)}$, 我们采取的是子树查询。这反而增加了一个维度, 即要求 j 与重链的交在 x 的上方。

注意到我们可以将式子写成 $\sum_{j \in (l, r]} S_{\min(r, j + len'_j - 1, j + len'_j - 1)}$, 要求 j 在重链的轻子树中。唯一需要去掉的子树为 pt_l 所在的轻子树, 通过对称后缀树查询就可以减去这个子树的贡献。这里 len'_l 的定义和 len'_j 相同。

当 $r \leq i + len'_l - 1$ 时, 即计算 $\sum_i S_{\min(r, i + len'_l - 1)} (i \in [\max(l+1, r - len'_l + 1), r])$, 使用扫描线以及树状数组解决。

否则, $i \in (l, r - len'_l]$, 此时求 $\sum_i S_{\min(i + len'_l - 1, i + len'_l - 1)}$ 。

对于 $len'_i \leq len'_l$, 我们只需要继续按重链从上至下的顺序扫描, 并使用树状数组查询即可。对于 $len'_i > len'_l$ 使用对称后缀树的方法查询。

复杂度 $O((n+q)\log^2 n)$, 期望得分 100 分。

2.9 算法 9

考虑从另一个角度观察算法 7。

算法 7 中需要求和的式子为 $S_r \times \sum_j [len'_j + j \geq r + 1] [j \in (l, r)] [len'_j \leq len'_l]$ 以及 $\sum_j [len'_j + j \leq r] [j \in (l, r)] S_{len'_j + j - 1}$ 。我们若将 (j, len'_j) 看作平面上的点 (x_j, y_j) , 那么限制形如 $x_j \in (l, r], y_j \leq len'_l, x_j + y_j \leq r$ 。



不妨考虑以下这种情况: $x_j \leq a, y_j \leq b, x_j + y_j \leq c$, 若 $c > a + b$ 则查询就为一个矩阵查询, 否则对于 $c \leq a + b$, 分别查询以下 3 个集合的信息: $A = \{j \mid x_j \leq a, x_j + y_j \leq c\}$, $B = \{j \mid y_j \leq b, x_j + y_j \leq c\}$, $C = \{j \mid x_j + y_j \leq c\}$, 那么合法的点可以通过 $A + B - C$ 得到。由于每个集合都对应只有两个限制或一个限制, 使用扫描线解决。

对于其他的情况, 也可以使用同样的方法求得答案, 故通过直接观察算法 7 的性质, 也可以将其优化为 $O((n+q)\log^2 n)$ 。

3 命题契机与题目总结

笔者是在 Simple KMP⁷ 这道题获得的灵感, 此题要求对一个串的所有子串求出其 *next* 树的深度和, 使用 *LCT* 以及后缀自动机就可以解决。笔者对其进行了修改, 变为多次询问某个子串的 *next* 树深度和。但十分不幸的是多次询问一个子串的 *next* 树深度和这个想法写出的式子与 CF1098F 重叠。基于这个原因, 笔者对求 *next* 树深度和这一问题进行了修改和加强, 改为了多次求一个字串的带权 *next* 深度和。

在尝试这个问题时, 笔者除了 *dsu on tree* 外, 还使用了对称后缀树的思想解决这个问题, 以此给出了 $O((n+q)\log^3 n)$ 的做法, 即算法 7。而后马耀华学长给出了更为优秀的 $O((n+q)\log^2 n)$ 做法, 即算法 8。

回到题目, 此题区分度良好, 使用简单的 *hash* 即可通过随机的数据。对于能力更高, 能发现更多性质的选手, 想出 $r = n$ 或是 $w_i = 1$ 的部分也能获得相应的不错的分数。

另外, 对带权 *next* 深度和的剖析方式也很多样, 除了基于后缀树和 *dsu on tree* 的做法, 彭思进学长还给出了基于分块的, 复杂度为 $O((n+q)\sqrt{n})$ 的简洁做法, 可以参见参考文献 [5]。

4 *dsu on tree* 在后缀树上的应用

以下将介绍 3 个例题, 它们都可以通过上文中的后缀树以及 *dsu on tree* 的方法来解决, 笔者借此来总结 *dsu on tree* 配合后缀树的应用。

4.1 例题 1⁸

4.1.1 题目描述

给定字符串 S 以及正整数 K , 问 S 有多少子串可以被划分成相同的 K 段。

⁷<http://www.51nod.com/Challenge/Problem.html#problemId=1600>

⁸来源: 校内模拟题

4.1.2 问题分析

这里给出的做法，可以求出以每个 i 开头的串有多少个可以被划分成相同的 K 段。我们直接将限制条件变成 $\text{lcp}(i, j) \geq (j - i) \times (K - 1)$ 。使用 *dsu on tree*，将限制条件变成 $j > i, \text{len}'_j \geq (j - i) \times (K - 1)$ ，即 $j - \lfloor \frac{\text{len}'_j}{K-1} \rfloor \leq i < j$ 。

对于每条重链从上至下处理的时，暴力 *dfs* 轻子树，对走到的点 j 进行 $[j - \lfloor \frac{\text{len}'_j}{K-1} \rfloor, j)$ 的区间加操作，在 i 处单点询问即可。

对于 $O(\log n)$ 次重儿子的询问，查询子树中 $(i, i + \lfloor \frac{\text{len}'_i}{K-1} \rfloor]$ 的个数即可。

时间复杂度 $O(n \log^2 n)$ 。此题存在更优的做法，但与本文关系不大，故这里不再讨论。

4.2 例题 2 Border 的四种求法⁹

4.2.1 题目描述

给定字符串 S 以及 m 次询问，每次给出 l, r ，问 $S[l, r]$ 的除本身以外的最长 *border*。

4.2.2 问题分析

我们将答案写成 $r + 1 - \min\{x \mid x \in (l, r], \text{lcp}(l, x) \geq r - x + 1\}$ 。

使用同样的方法，将要求式子写为 $\min\{x \mid x \in (l, r], \text{len}'_x + x - 1 \geq r\}$ ，并要求 $\text{len}'_x \leq \text{len}'_l$ 。

我们将 x 放入线段树，其位置上的值设为 $\text{len}'_x + x - 1$ ，我们维护这个值的最大值，在线段树上二分就可以得到答案。对于 $O(\log n)$ 次重儿子的查询，类似使用线段树合并得到 *Right* 集合并查询即可。对于 pt_l 自己所在的轻子树，设求出了 x 作为答案，由于 $\text{len}'_x < \text{lca}(pt_x, pt_l)$ ，故这个答案仍然是合法的，于是不需要除去轻子树。时间复杂度 $O((n + m) \log^2 n)$ 。

使用上文算法 8 的思路，将式子写为 $\min\{x \mid x \in (l, r], \min(\text{len}'_x, \text{len}'_l) + x - 1 \geq r\}$ 。只需考虑 $x + \text{len}'_l - 1 \geq r$ 即 $x \geq r - \text{len}'_l + 1$ 的 x ，故问题变为查询 $\min\{x \mid x \in [\max(l + 1, r - \text{len}'_l + 1), r], x + \text{len}'_x \geq r + 1\}$ 。使用基数排序，按 r 从小到大查询，当前保留所有 $x + \text{len}'_x \geq r + 1$ 的点，用并查集查询某个位置的下一个合法位置，复杂度 $O((n + q)\alpha(n) \log n)$ 。

本题利用 *border* 的性质可以得到一个更优的做法，这里不再赘述。

⁹来源：<https://www.luogu.com.cn/problem/P4482>

4.3 例题 3¹⁰

4.3.1 题目描述

给定字符串 S 和 m ，定义两个长为 m 的串相似，当且仅当两个串相同或只有一个位置不一样。对字符串的每个长为 m 的子串求出有多少其它长为 m 的子串与它相似。

4.3.2 问题分析

将答案写成对每个 i 求 $\text{lcp}(i, j) + \text{lcs}(i + m - 1, j + m - 1) \geq m - 1$ 的 j 的个数。

我们正序建立一个后缀自动机，那么在其树结构上，有 $\text{lcs}(i, j) = \text{len}_{\text{lca}(pt_i, pt_j)}$ 。

使用 *dsu on tree* 的方法，类似地，我们就可以将限制写成 $\text{lcp}(i - m + 1, j - m + 1) + \text{len}'_j \geq m - 1$ 。对于这个限制，我们可以求出后缀数组¹¹，设后缀 i 的排名为 rank_i ，那么通过在 rank_{j-m+1} 左右二分，可以将限制变为 $\text{rank}_{i-m+1} \in [L_j, R_j]$ 。我们每遇到一个 j 就对 $[L_j, R_j]$ 这个区间进行区间加 1 操作，最后查询 rank_{i-m+1} 处的值就可以了。

对于 $O(\log n)$ 次子树的询问，要求的式子变为 $\text{lcp}(i - m + 1, j - m + 1) \geq m - \text{len}'_i - 1$ 。我们在后缀数组的 rank_{i-m+1} 处二分，对应的就是查询子树中 $\text{rank}_{j-m+1} \in [L, R]$ 的个数。

时间复杂度 $O(n \log^2 n)$ 。

5 总结与展望

从本题以及例题中可以看到，*dsu on tree* 在处理 *lcp* 相关问题时的优越性。究其原因，是因为复杂的 $\text{lcp}(i, j)$ 在重链上可以写成 len'_j 这样简单的形式，配合其他的数据结构进行数点就可以得到我们想要的答案。而特殊处理的 $O(\log n)$ 个子树形式较为简单，往往可以通过简单的子树查询就可以求出答案。

总的来说，*dsu on tree* 在这类问题上写出的式子形式简单，并不需要做太多的推导，通过简短的分类讨论就可以变成常规的数据结构题，无疑是一个“小清新”的数据结构。

笔者在 *dsu on tree* 配合后缀树这种方法上做出了总结，并给出了 3 道例题来阐释其在后缀树上的应用，希望能起到抛砖引玉的作用，启发读者扩展出 *dsu on tree* 更多的应用，并引出更加新颖有趣的题目。

6 致谢

感谢父母的养育之恩。

¹⁰来源：校内模拟题

¹¹后缀数组可以参见参考文献 [6]



感谢中国计算机学会给予的交流机会。
感谢成都石室中学曾贵胜老师、梁德伟老师多年来的指导与帮助。
感谢涂宵箫同学，张家瑞同学，曹越同学为本文审稿。
感谢 UOJ 提供的比赛平台以及各位 UOJ 管理员的帮助。
感谢所有 OI 路上支持和帮助我的人。

参考文献

- [1] 陈立杰，后缀自动机，2012 年冬令营营员交流。
- [2] 张天扬，后缀自动机及其应用，2015 年中国国家候选队论文集。
- [3] Codeforces Blog, <http://codeforces.com/blog/entry/44351>。
- [4] Codeforces Blog, <https://codeforces.com/blog/entry/64331>。
- [5] Itst 的 UOJ 博客, <https://itst.blog.uoj.ac/blog/7514>
- [6] Wikipedia, The free encyclopedia, Suffix Array,
https://en.wikipedia.org/wiki/Suffix_array。



浅谈OI中的模拟费用流问题

浙江省余姚中学 陈扩文

摘要

本文围绕模拟费用流展开,对一部分现有的模型进行总结并提出了一些新的想法。

1 引言

费用流是信息学竞赛中一类重要的问题,有着广泛的应用,但朴素费用流算法的运行效率往往不尽人意。在一些特殊图中往往能通过一些效率更高的方法来解决费用流问题,一般称之为模拟费用流。本文重点讨论模拟费用流问题,希望对读者有所启发。

在第2节中,笔者形式化了一类基本的费用流模型并展开分析了若干性质。

在第3节中,笔者从解题方法出发介绍了若干种模拟费用流的思路 and 技巧。

在第4节中,笔者从特殊图的结构出发讨论了一些模拟费用流的方法。

2 费用流简述

2.1 定义与约定

对于图 $G = (V, E)$, 每条边 (u, v) 有容量 $c(u, v)$ 与代价 $a(u, v)$ (这里不妨令两者均为整数)。下文为了方便描述我们用加边 $(u, v, c(u, v), a(u, v))$ 表示向图中加入一条对应边。

定义 2.1 (有源汇最小费用可行流问题)。对于源点 s 与汇点 t , 定义为如下线性规划:

$$\begin{aligned} \min & \sum_{(u,v) \in E} x_{u,v} a(u,v) \\ \left(\sum_{(v,u) \in E} x_{v,u} \right) - \left(\sum_{(u,v) \in E} x_{u,v} \right) &= 0 & (u \in V - \{s, t\}) \\ x_{u,v} &\leq c(u,v) & ((u,v) \in E) \\ x_{u,v} &\geq 0 & ((u,v) \in E) \end{aligned} \tag{1}$$

定义 2.2 (可行流)。 (1) 式的一组可行解。

定义 2.3 (可行流费用). 对于可行流 x^* 定义其费用 $cost_G(x^*) = \sum_{(u,v) \in E} x_{u,v}^* a(u,v)$

定义 2.4 (可行流流量). 对于可行流 x^* 定义其流量 $flow_G(x^*) = \sum_{(s,u) \in E} x_{s,u}^* - \sum_{(u,s) \in E} x_{u,s}^*$

2.2 残存网络

为了方便描述, 这里不展开讨论图中有重边、自环、反向边的情况。

对于一组可行流 x^* , 定义其残存网络 $G(x^*) = (V, E(x^*))$

其中 $E(x^*) = \{(u,v) | (u,v) \in E \vee (v,u) \in E\}$

定义新的流量与费用:

$$c^*(u,v) = \begin{cases} c(u,v) - x_{u,v}^* & (u,v) \in E \\ x_{v,u}^* & (v,u) \in E \end{cases}$$

$$a^*(u,v) = \begin{cases} a(u,v) & (u,v) \in E \\ -a(v,u) & (v,u) \in E \end{cases}$$

定义 2.5 (可行流递增). 对于 $G(x^*)$ 的一组可行流 x^o , 我们定义 $x^* \uparrow x^o$ 为 x^o 对 x^* 的递增, 其中 $(x^* \uparrow x^o)_{u,v} = x_{u,v}^* + x_{u,v}^o - x_{v,u}^o$ ($(u,v) \in E$)

引理 2.1. 对于 G 的可行流 x^* 与 $G(x^*)$ 的可行流 x^o , $x^* \uparrow x^o$ 是 G 的一组可行流且 $cost_G(x^* \uparrow x^o) = cost_G(x^*) + cost_{G(x^*)}(x^o)$

证明. 将 $x^* \uparrow x^o$ 的定义式带入(1) 易证。 □

2.3 增广路与负环

对于残存网络 $G(x^*)$ 和边集 P ,

定义其容量 $capacity(P) = \min\{c^*(u,v) | (u,v) \in P\}$ 。

定义 2.6 (增广路). 对于残存网络 $G(x^*)$, 称 s 到 t 的增广路是一条从 s 到 t 的简单路径, 对于路径上的每一条边 (u,v) , 有 $c^*(u,v) > 0$ 。

定义 2.7 (负环). 对于残存网络 $G(x^*)$, 负环是一个简单环, 对于环上每一条边 (u,v) , 有 $c^*(u,v) > 0$, 且环上边权总和 < 0 。

我们称 x^o 为边集 P 的一组对应流当且仅当 $x_{u,v}^o = [(u,v) \in P]e$, 其中 e 为任意 $(0, capacity(P)]$ 中的实数。

引理 2.2. 对于 $G(x^*)$, 负环与增广路的对应流均为可行流。

证明. 显然 □

这里我们考虑以下两个线性规划:

$$\begin{aligned} \max & 0 \\ d_v & \leq d_u + a^*(u, v) (c^*(u, v) > 0) \\ d_t & = d_s \end{aligned} \quad (2)$$

$G(x^*)$ 的最小费用可行流问题的对偶问题:

$$\begin{aligned} \max & - \sum_{(u,v) \in E(x^*)} c^*(u, v) \max(0, p_v - p_u - a^*(u, v)) \\ p_t & = p_s \end{aligned} \quad (3)$$

引理 2.3. 对于残存网络 $G(x^*)$, (2) 式无解当且仅当如下三个条件中至少一个成立:

1. 存在负环
2. 存在 s 到 t 的费用小于 0 的增广路。
3. 存在 t 到 s 的费用小于 0 的增广路。

证明. 充分性显然, 对于必要性可以转化为最短路模型用 *Bellman - Ford* 算法构造一组解. □

定理 2.1. 一个残存网络 $G(x^*)$ 的最小费用可行流问题找不到负环或负的增广路当且仅当其最优解权值为 0。

证明. 必要性显然, 下面证明充分性:

结合**引理 2.3.**, 此时(2)式有可行解, 我们取其任意一组解 d^* , 将其带入(3)式。

此时有: $d_t^* \geq d_s^*$

对于每条边 (u, v) , 一定符合下列两个条件中的至少一个:

1. $c^*(u, v) = 0$
2. $d_v^* \leq d_u^* + a^*(u, v)$

即 $c^*(u, v) \max(0, d_u^* - d_v^* - a^*(u, v)) = 0$ 。此时(3)式的解权值为 0。

根据线性规划弱对偶性定理, 原可行流问题的最优解权值 ≥ 0 , 而我们直接取 $x_{u,v} = 0$ 即可得到一组权值为 0 的解, 因此得证. □

定理 2.2. 若残存网络 $G(x^*)$ 的最优解为 0, 则原问题的最优解为 $cost_G(x^*)$ 。

证明. 假设原问题的最优解不是 $cost_G(x^*)$, 我们设其中一组最优解为 x^M .

根据假设有: $cost_G(x^M) < cost_G(x^*)$.

我们构造一组 $G(x^*)$ 的可行流:

$$x'_{u,v} = \begin{cases} \max(0, x^M_{u,v} - x^*_{u,v}) & (u, v) \in E \\ \max(0, x^*_{v,u} - x^M_{v,u}) & (v, u) \in E \end{cases}$$

因为 $G(x^*)$ 的最优解为 0, 所以 $cost_{G(x^*)}(x') \geq 0$.

容易发现 $x^* \uparrow x'_{u,v} = x^M$, 根据引理 2.1., $cost_G(x^M) = cost_G(x^*) + cost_{G(x^*)}(x') \geq cost_G(x^*)$.

与假设矛盾, 因此得证.

□

2.4 SSP 算法

SSP 算法 (Successive Shortest Path Algorithm) 是目前 OI 中常用的求解费用流问题的算法。

使用该算法求解有源汇最小费用可行流问题需要符合两个条件:

1. 图中无负环
2. 图中无 t 到 s 的费用为负的增广路

若条件 2 不满足则可交换源汇。

算法流程如下:

Algorithm 1 SECESSIVE SHORTEST PATH ALGORITHM

```

1:  $x^* \leftarrow 0$ 
2: while  $G(x^*)$  的最优解  $< 0$  do
3:   使用 Bellman-Ford 算法在  $G(x^*)$  找一条  $s$  到  $t$  的最短路径  $P$ 
4:   构造  $P$  的容量为 1 的对应可行流  $x^o$ 
5:    $x^* \leftarrow x^* \uparrow x^o$ 
6: end while
7: return  $x^*$ 
    
```

引理 2.4. SSP 算法运行过程中不会出现负环或 t 到 s 的负费用增广路, 且每次得到的 $cost_{G(x^*)}(x^o)$ 单调不降。

证明. 考虑如下线性规划:

$$\begin{aligned} \max & d_t - d_s \\ & d_v \leq d_u + a^*(u, v) \quad (c^*(u, v) > 0) \end{aligned} \quad (4)$$

我们维护一组解 d , 考虑归纳证明。

初始时没有负环, 因此可以得到一组可行解。

每轮循环我们考虑将 d 更新为新残存网络上的解 d' , 考虑图的变化, 依次删去若干边并加入一些反平行边。

我们考虑先依次加入反平行边, 对于最短路径上的点 (u, v) 有 $d_u + a^*(u, v) = d_v$, 因此加入限制 $d'_u \leq d'_v - a^*(u, v)$ 并不会产生松弛, 即经过该过程后 d 依然是一组最优解。

对于删边, 显然不会产生负环, 且一定存在一组新图上的最优解 d' 使得 $d'_u \geq d_u$ 。

考虑将边权改为 $a'(u, v) = d'_u - d'_v + a^*(u, v)$, 根据(4)式有 $a'(u, v) \geq 0$ 。则 t 到 s 的任意一条增广路的费用 $\geq d'_s - d'_t \geq d_s - d_t \geq 0$ 。

综上, 该引理得证。 $\square$

定理 2.3. 对于符合条件的流网络 G , SSP 算法能够在有限时间内终止并返回一组最优解。

证明. 根据引理 2.4. 该算法总能找到最短路并构造出新的可行解。

容易发现外层的 while 循环运行次数不可能超过流量的上限, 因此 SSP 算法能在有限时间内终止。

根据定理 2.1. 和定理 2.2., 该算法终止时必定返回了最优解。

综上, 该算法正确性得证。 $\square$

对于非负整数 k , 我们用 $cost_G(k)$ 表示流量恰好为 k 时的最优解权值。

定理 2.4. 对于符合条件的流网络 G 和任意非负整数 k , SSP 算法第 3 至 5 行执行 k 次得到的解 x 有 $cost_G(x) = cost_G(k)$ 。

证明. 考虑加入条件 $flow_G(x) = k$ 重新构图, 可以建一个超级源点 SS , 加边 $(SS, s, k, -\infty)$ 。然后求解源点为 SS 、汇点为 t 的最小费用可行流。

对上述问题使用 SSP 算法, 与原问题比较运行步骤, 容易发现其等价于直接在原问题上执行 k 轮循环, 因此得证。 $\square$

3 技巧与思路

本节从费用流的定义与性质出发, 介绍一些可能的模型转化方法以及常见的模拟费用流思考方向。

3.1 模拟增广路

增广路的过程即每次在残存网络上找一条从源点到汇点的最短路, 然后将路径上的边流量减少, 并增加反平行边的流量。

对于一些特殊图可以采用线段树等数据结构来快速模拟增广路的过程。

模拟增广路的方法往往可以高效地动态增减边的流量维护最小费用。

部分具体例子可以参考第 4 节。

3.1.1 寻找关键点

在一些图增广路长度较小且存在少量结点决定了图的主要性质, 我们可以尝试取这些结点作为关键点。

例题 1. (AGC018C) Coins

题目大意: 有 n 个人, 第 i 个有 A_i 枚金币, B_i 枚银币, C_i 枚铜币, 要将人分成三组, 人数分别是 X, Y, Z ($n = X + Y + Z$), 要最大化第一组人的金币个数之和 + 第二组人的银币个数 + 第三组人的铜币个数之和。 $X + Y + Z \leq 10^5, 1 \leq A_i, B_i, C_i \leq 10^9$

解法

建图:

首先假设所有人分到第一组, 然后再分配到二、三组。

1. 对每个人建一个点 P_i , 加边 $(s, P_i, 1, A_i)$
2. 建两个点表示 F_2, F_3 表示第二、三组, 对每个人 i 加边 $(P_i, F_2, 1, -B_i), (P_i, F_3, 1, -C_i)$
3. 加边 $(F_2, t, Y, -\infty), (F_3, t, Z, -\infty)$, 表示强制两边分配到对应人数。

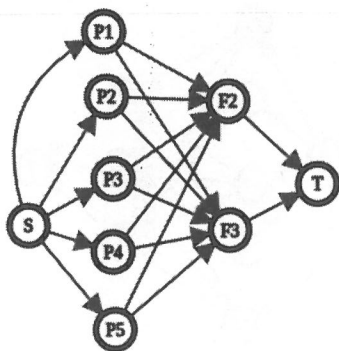
我们选取关键点 s, F_2, F_3, t , 对它们两两之间建一个堆。

初始时将每条能从一个关键点到另一个关键点且路上不经过其它关键点的路径按权值压入对应堆中。

此时关键点间的堆顶即为两点间的一条最短路。

接下来考虑实现寻找增广路和增广的操作:

1. 寻找增广路即在四个关键点中跑 s 到 t 的最短路, 然后将对应堆顶对应的路径连起来即可。可以用 spfa 等算法, 也可以分类讨论降低复杂度。



(a) 示例图

2. 进行增广操作, 每次操作后在影响到的堆中将原先的路径删去再加入新的可能路径。本题中可以发现一条增广路长度是 $O(1)$, 每次修改路径只需要进行 $O(1)$ 次堆的删除和插入操作。

如果将关键点个数视为常数, 我们就得到了一个 $O(n \lg n)$ 的算法。 ■

例题 2. (NOI2019) 序列

题面大意: 给定非负整数 n, k, L 和两个长度为 n 的序列 $\{a_i\}, \{b_i\}$ 。

要确定两个长度为 k 的序列 $\{c_i\}, \{d_i\}$, 其中 $1 \leq c_1 < c_2 < \dots < c_k \leq n, 1 \leq d_1 < d_2 < \dots < d_k \leq n$ 。

要求 $\{c_1, c_2, \dots, c_k\} \cap \{d_1, d_2, \dots, d_k\} \geq L$

最大化 $\sum_{i=1}^k a_{c_i} + b_{d_i}$

$n \leq 10^6, 1 \leq a_i, b_i \leq 10^9$

解法

可以发现一个位置 i 有四种情况:

1. 不贡献进答案
2. 取入 c 序列, 即对答案贡献 a_i
3. 取入 d 序列, 即对答案贡献 b_i
4. 同时取入 c 和 d 序列, 即对答案贡献 $a_i + b_i$ 。

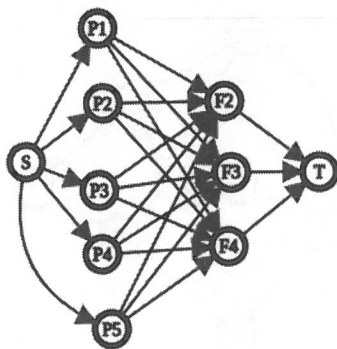
解法一:

我们考虑枚举第四种情况的个数 $d(L \leq d \leq n)$, 这样四种情况的个数都确定了。

建图: 1. 建 n 个点 P_i , 加边 $(S, P_i, 1, 0)$ 。

2. 建三个点 F_2, F_3, F_4 分别表示情况 2, 3, 4。对每个 i 加边 $(S, P_i, F_2, -a_i), (S, P_i, F_3, -b_i), (S, P_i, F_4, -a_i - b_i)$ 。

3. 加边 $(F_2, T, k - d, -\infty), (F_3, T, k - d, -\infty), (F_4, T, d, -\infty)$ 。



(b) 示例图

图的结构和**例题 1** 几乎没有区别，可以取 s, F_2, F_3, F_4, t 五个点作为关键点模拟。

但这道题外面还需要枚举 d ，直接枚举的复杂度无法接受。我们考虑从小到大枚举 d ，考虑上一次 $d-1$ 的费用流图和现在 d 的费用流图，我们可以在上一张图的基础上修改边容量，退掉不合法的流（相当于从 T 往 s 经过对应边跑增广路），然后更新新出现的增广路（本题因为图性质特殊所以不会产生负环）。

因为每次修改的边容量是 $O(1)$ 的，所以我们需要找的增广路条数也是 $O(1)$ 的。这样我们的总复杂度就变为了 $O(n \lg n)$ 。

解法二：

由于第四种情况只有下界限制，我们可以放宽一点限制，在满足第四种情况至少 k 个的情况下允许同时选择第二、三种情况。我们设 x_i 为取第二种情况， y_i 为第三种情况， z_i 为强制取的第四种情况，写出线性规划：

$$\begin{aligned} \max \quad & \sum_{1 \leq i \leq n} a_i x_i + b_i y_i + (a_i + b_i) z_i \\ & x_i + z_i \leq 1 (1 \leq i \leq n) \\ & y_i + z_i \leq 1 (1 \leq i \leq n) \\ & \sum_{1 \leq i \leq n} z_i \geq L \\ & \sum_{1 \leq i \leq n} x_i + z_i = k \\ & \sum_{1 \leq i \leq n} y_i + z_i = k \\ & 0 \leq x_i, y_i, z_i \leq 1 \end{aligned}$$

可以改写成：

$$\min - \sum_{1 \leq i \leq n} a_i x_i + b_i y_i + (a_i + b_i) z_i$$

$$\hat{x}_i = x_i + z_i$$

$$\hat{y}_i = y_i + z_i$$

$$w = \sum_{1 \leq i \leq n} x_i$$

$$w = \sum_{1 \leq i \leq n} y_i$$

$$\sum_{1 \leq i \leq n} \hat{x}_i = k$$

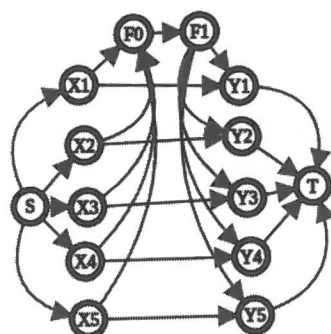
$$\sum_{1 \leq i \leq n} \hat{y}_i = k$$

$$0 \leq \hat{x}_i, \hat{y}_i \leq 1, 0 \leq x_i, y_i, z_i \leq 1, 0 \leq w \leq k - L$$

上式容易转化为费用流的模型。

建图：

1. 对于每个 i 建两个点 X_i, Y_i ，加边 $(S, X_i, 1, 0), (Y_i, T, 1, 0), (X_i, Y_i, 1, -a_i - b_i)$
 2. 建点 F_0, F_1 ，加边 $(F_0, F_1, k - L, 0)$ ，对于每个 i 加边 $(X_i, F_0, 1, -a_i), (F_1, Y_i, 1, -b_i)$
- 要求流量正好为 k 时的最小费用。



(c) 示例图

同样的方法，我们可以取 S, F_0, F_1, T 作为关键点。但某些关键点之间的路径需要分类讨论一下（下文用 $\rightarrow$ 连接路径上依次经过的点）：

从 S 到 F_0 只有一种情况， $S \rightarrow X_i \rightarrow F_0$ 。

从 S 到 F_1 只有一种情况， $S \rightarrow X_i \rightarrow Y_i \rightarrow F_1$ 。

从 F_0 到 F_1 有两种情况， $F_0 \rightarrow F_1$ ， $F_0 \rightarrow X_i \rightarrow Y_i \rightarrow F_1$ 。



从 F_1 到 F_0 有两种情况, $F_1 \rightarrow F_0$, $F_1 \rightarrow Y_i \rightarrow X_i \rightarrow F_0$ 。

从 F_0 到 T 只有一种情况, $F_0 \rightarrow X_i \rightarrow Y_i \rightarrow T$ 。

从 F_1 到 T 只有一种情况, $F_1 \rightarrow Y_i \rightarrow T$ 。

用堆维护即可。时间复杂度 $O(n \lg n)$ 。 ■

3.2 凸性

定理 3.1. 对于非负整数 k , $cost_G(k)$ 在定义域内为下凸函数。

证明. 由引理 2.4. 结合 SSP 算法易得。 □

因此对于一些特殊问题, 我们可以根据凸包的性质分析问题。

常见的有如下几种方法:

3.2.1 二分流量

利用凸性对流量进行二分, 三分, wqs 二分等。

例题 3. (网络流 24 题) 餐巾计划问题

题目大意: 一个餐厅在 n 天里, 第 i 天需要用 r_i 块餐巾, 用完后可以送到快洗部或慢洗部, 前者洗一块需要 t_0 天, 费用为 f_0 , 后者需要 t_1 天, 费用为 f_1 , 购买新餐巾的代价是 p 。求度过 n 天的最小花费。

解法 建图:

1. 对于每个点 i 建立两个点 A_i 和 B_i , 加边 $(A_i, B_i, r_i, -\infty), (B_i, T, \infty, 0)$
2. 对于 $i < n$, 加边 $(A_i, A_{i+1}, \infty, 0)$
3. 对于 $i + t_0 \leq n$, 加边 $(B_i, A_{i+t_0}, \infty, f_0)$
4. 对于 $i + t_1 \leq n$, 加边 $(B_i, A_{i+t_1}, \infty, f_1)$
5. 加边 $(S, A_1, \infty, p), (A_n, T, \infty, 0)$

跑最小费用可行流即可求出答案。由于 $cost$ 是凸的, 所以可以直接三分流量, 然后贪心。 ■

时间复杂度 $O(n \lg(\sum r_i))$

3.2.2 维护折线

若将 $V - \{s, t\}$ 划分成两个集合 V_0, V_1 , 若两点集之间只有一条边, 固定这条边的流量跑最小费用可行流, 则该边流量与其中一侧产生的最小费用成凸函数关系 (容易通过构图转化为定理 3.1. 的情形)

因此在一些特殊图中，我们可以通过折线维护子图的凸函数来解决一些问题。
具体例子参考第 4 节。

3.3 线性规划对偶

对偶后的形式参考第 2 节，

若原先容量和费用均为整数，可以证明对偶后一定存在一组整数解是最优解。¹

例题 4. (ZJOI2020) 序列

题目大意： 给一个长度为 n 的非负整数序列 a_i ，每次可以选择一个区间 $[l, r]$ ，然后选择如下三种操作中的一种：

1. 将下标在这个区间里的所有数都减 1。
2. 将下标在这个区间里且下标为奇数的所有数都减 1。
3. 将下标在这个区间里且下标为偶数的所有数都减 1。

求能将序列变成全 0 的最少操作次数。

解法

可以将题目转化为费用流的形式。

然后对该费用流的线性规划形式进行对偶可以转化为一个动态规划模型。具体步骤在参考文献 [5] 中已有较详细的介绍，这里不再赘述。 ■

3.3.1 保序回归模型

对于如下最小费用可行流问题可以转化为保序回归模型²：

1. 存在一个非负整数序列 h_i ，使得对于每条边 $(u, v) (u, v \in V - \{s, t\})$ ，其容量为 ∞ 且费用为 $h_v - h_u$ 。
2. 每个 s 和 t 与其它节点的连边边权 ≥ 0 且容量为正整数。

不妨写为如下线性规划：

$$\begin{aligned} \max \quad & \sum_i \left(\sum_j \alpha_{i,j} - \sum_j \beta_{i,j} \right) h_i - \sum_i \sum_j a_{i,j} \alpha_{i,j} - \sum_i \sum_j b_{i,j} \beta_{i,j} \\ & \sum_j \alpha_{u,j} - \sum_j \beta_{u,j} + \sum_{(v,u) \in E} f_{v,u} - \sum_{(u,v) \in E} f_{u,v} = 0 \quad (u \in V) \\ & \alpha_{i,j} \leq 1, \beta_{i,j} \leq 1 \\ & \alpha_{i,j}, \beta_{i,j}, f_{u,v} \geq 0 \end{aligned}$$

¹详见参考文献 [5]

²该部分内容参考自本人的互测命题报告

我们进行一些转化：

对于每个 u 构造出无穷多个常量 $E_{i,j}, \hat{E}_{i,j} \leq 0$

$$\text{并令 } t_i = \sum_j \alpha_{i,j} - \sum_j \beta_{i,j}$$

可将线性规划改写为：

$$\min - \sum_i t_i h_i + \sum_{i,j} \max(E_{i,j} + t_i, 0) + \sum_{i,j} \max(\hat{E}_{i,j} - t_i, 0)$$

$$t_u + \sum_{(v,u) \in E} f_{v,u} - \sum_{(u,v) \in E} f_{u,v} = 0 (u \in V)$$

$$f_{u,v} \geq 0$$

用约束代掉 max：

$$\min - \sum_i t_i h_i + \sum_{i,j} s_{i,j} + \sum_{i,j} \hat{s}_{i,j}$$

$$s_{i,j} - t_i \geq E_{i,j}$$

$$\hat{s}_{i,j} + t_i \geq \hat{E}_{i,j}$$

$$t_u + \sum_{(v,u) \in E} f_{v,u} - \sum_{(u,v) \in E} f_{u,v} = 0 (u \in V)$$

$$f_{u,v}, s_{i,j}, \hat{s}_{i,j} \geq 0$$

然后进行线性规划对偶：

$$\max \sum_{i,j} r_{i,j} E_{i,j} + \hat{r}_{i,j} \hat{E}_{i,j}$$

$$r_{i,j} \leq 1$$

$$\hat{r}_{i,j} \leq 1$$

$$- \sum_{i,j} r_{i,j} + \sum_{i,j} \hat{r}_{i,j} + e_i = -h_i$$

$$e_u - e_v \geq 0 ((u,v) \in E)$$

$$r_{i,j}, \hat{r}_{i,j} \geq 0$$

令 $z_i = -e_i$ 再整理一下:

$$\begin{aligned} \min & - \sum_{i,j} r_{i,j} E_{i,j} - \hat{r}_{i,j} \hat{E}_{i,j} \\ & 0 \leq r_{i,j} \leq 1 \\ & 0 \leq \hat{r}_{i,j} \leq 1 \\ & \sum_{i,j} \hat{r}_{i,j} - \sum_{i,j} r_{i,j} = z_i - h_i \\ & z_v \geq z_u ((u, v) \in E) \end{aligned}$$

然后用类似之前的方法, 对于每个 i , 当 $\sum_j \hat{r}_{i,j} - \sum_j r_{i,j}$ 确定的时候构造一个函数把它的最优值算出来, 设这个函数为 G_i 。那么改写之前的线性规划:

$$\begin{aligned} \min & \sum_i G_i(z_i - h_i) \\ & z_u \leq z_v ((u, v) \in E) \end{aligned}$$

即转化为了一个类似保序回归的模型。

该保序回归同样可以采用整体二分的方法来解决。证明如下:

引理 3.1. G_i 是一个 0 点为最小值的凸函数

引理 3.2. 在对于任意整数 t , G_i 在区间 $(t, t+1)$ 中均为一次函数。

以上两者均根据前面的推导过程易得。

引理 3.3. 对于任意集合 U , 满足 $\sum_{i \in U} G_i(k - h_i)$ 最小的 k 的取值一定是一段区间。

证明. 由引理 3.2. 易得。 □

定义 3.1. z 向 $\{a, b\} (a \leq b)$ 取整: 将 z 中的元素大于 b 的变成 b , 小于 a 的变成 a

定义 3.2. 集合 U 的均值区间: 满足 $\sum_{i \in U} G_i(k - h_i)$ 最小的 k 的取值区间

定理 3.2. 对于 $a < b$, 任意 $h_i \notin (a, b)$, 若 z^s 为原问题加入条件 $z_i^s \in \{a, b\}$ 后的一组最优解, 则一定存在一组最优解 z 使得其向 $\{a, b\}$ 取整后为 z^s

证明. 反证法, 不妨假设原问题的最优解 z' 必有 $z'_i \leq a$ 且 $z^S = b$

设 $x = \{z'_i | z'_i \leq a, z_i^S = b\}, U = \{i | z'_i = x, z_i^S = b\}$ 。

若 $l > a$, 将 U 中 z'_i 改成 $\min(l, b)$ 后不劣。

若 $l \leq a, r \geq b$, 将 U 中 z'_i 改成 b 后不劣。

若 $l \leq a, r < a$, 将 U 中 z_i^S 改成 a 不劣。

若 $l \leq a, r < b$:

若 $U = \{i | z'_i \leq a, z_i^S = b\}$, 全部改成 a 不劣。

否则设 $y = \max\{z'_i | z'_i < x, z_i^S = b\}, U' = \{i | z'_i = y, z_i^S = b\}$, 其均值区间为 $[l', r']$

若 $r' \leq x$, 将 $f_i^S (i \in U \cup U')$ 改成 a 不劣。

若 $r' > x$, 则将 $z'_i (i \in U')$ 改成 x 不劣。

重复上述过程即可找到矛盾。

□

剩余部分在参考文献 [4] 中已有较详细的论述, 这里不再赘述。

4 特殊图举例

本节针对一些特殊的图总结一些可能的利用图性质的方法并结合例题来说明。

这里我们主要通过第3节中的使用数据结构模拟增广路的方法和维护凸包的方法来解决。

为了方便描述问题, 本节我们通常采用“老鼠进洞”模型, 对于 $V = \{s, t\}$, 每条 S 的出边我们视为一群老鼠, 每条 T 的入边我们视为一个洞。每个老鼠和洞都可能代价 (即对应边的费用), 洞有容量、老鼠有数量 (对应边的容量)。我们的任务是选择若干老鼠, 每个老鼠进入一个洞, 每个洞里老鼠的数量不能超过洞的容量, 最小化总代价³。

若无特殊说明, 我们通常认为老鼠群数、洞的数量和图上节点数量三者同阶, 均为 $\Theta(n)$ 级别。

³借鉴自参考文献 [1]

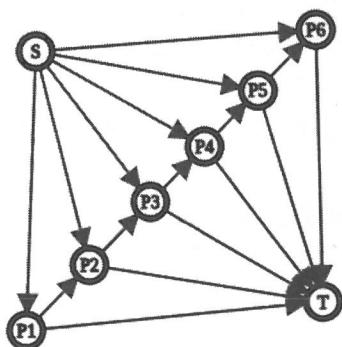
4.1 单向链

4.1.1 定义与约定

单向链一般指形如如下的图：

n 个点，第 i 个点向第 $i+1$ 个点连一条单向边。 S 和 T 与图中每个点连有若干条边。

单向链问题通常可以采用前缀和等方式将链上的边的费用转化到老鼠或洞上，所以我们这里不妨认为老鼠和洞均有代价，链上的边权均为 0。



(d) 示例图

4.1.2 例题

例题 5. 跳伞求生⁴

题目大意：单向链，链上边容量为 ∞ ，每群老鼠只有一只，每个洞只能进一只老鼠。

解法

解法一：

考虑模拟增广路，用线段树维护每条反平行边的流量。

按任意顺序加入老鼠或洞。

考虑新加入边后产生的增广路，发现这里可能会出现负环。结合 SSP 算法，我们发现已经满流的老鼠对于后续的退流操作相当于是个洞，因此每当我们取了一个代价为 x 的老鼠，我们就删去这个老鼠然后加入一个代价为 $-x$ 的洞。对于满流的洞同理。

容易证明这样操作后图中不会产生负环。

考虑加入老鼠的情况，问题变成了：对于新出现在点 i 的老鼠，找一个能到达的点 j 的洞，最小化双方的代价之和。

⁴Lydsy1708 月赛

那么我们只需要在线段树上找出能到达的区间并求最小值即可,如果代价和 < 0 就增广这条路。

对于增广操作,我们取的老鼠修改为洞,取的洞修改为老鼠,然后剩余部分相当于反平行边的流量区间加一或者区间减一。

复杂度为 $\Theta(n \lg n)$ 。

该算法还可用于处理动态增加边容量的问题。

解法二:

考虑维护折线。

不妨按节点编号从小到大加入老鼠或洞。

若当前遍历到第 i 个点,考虑维护多余的流量与前面部分最小费用的关系。

可以尝试写成动态规划的形式:

$f(i, j)$ 表示加入了前 i 个老鼠或洞,剩余的流量为 j 的最小代价。若老鼠和洞一共有 m 个,答案即为 $f(m, 0)$ 。

我们分开讨论加入的是老鼠还是洞。

1. 新加入的边代表一个代价为 x 的老鼠。

有转移方程:

$$f(i, j) = \begin{cases} \min(f(i-1, j-1) + x, f(i-1, j)) & j > 0 \\ f(i-1, 0) & j = 0 \end{cases}$$

考虑对于每个 $j(j > 0)$ 维护 $g(i, j) = f(i, j) - f(i, j-1)$, 该函数关于 j 单调递增。

那么之前的转移方程等价于:

$$f(i, j) = \begin{cases} f(i-1, j-1) + x & j > 0 \wedge x < g(i-1, j) \\ f(i-1, j) & j > 0 \wedge x \geq g(i-1, j) \\ f(i-1, j) & j = 0 \end{cases}$$

即:

$$f(i, 0) = f(i-1, 0)$$

$$g(i, j) = \begin{cases} g(i-1, j-1) & j > 1 \wedge x < g(i-1, j-1) \\ x & (j = 1 \vee g(i-1, j-1) \leq x) \wedge x < g(i-1, j) \\ g(i-1, j) & x \geq g(i-1, j) \end{cases}$$

相当于在 $g(i)$ 是在 $g(i-1)$ 的基础上加入 x 再排序。

2. 加入一个代价为 x 的洞。

有转移方程:

$$f(i, j) = \min(f(i-1, j), f(i-1, j+1) + x)$$

即：

$$f(i, j) = \begin{cases} f(i-1, j) & -x < g(i-1, j+1) \\ f(i-1, j+1) + x & -x \geq g(i-1, j+1) \end{cases}$$

由于 $g(i-1, 1)$ 是最小的，所以我们分类讨论 $-x$ 与 $g(i-1, 1)$ 的大小。

case 1: $-x < g(i-1, 1)$

则 $f(i) = f(i-1)$

case 2: $-x \geq g(i-1, 1)$

$$f(i, 0) = f(i-1, 0) + x + g(i-1, 1)$$

$$g(i, j) = \begin{cases} g(i-1, j+1) & -x \geq g(i-1, j+1) \\ -x & g(i-1, j) \leq -x < g(i-1, j+1) \\ g(i-1, j) & -x < g(i-1, j) \end{cases}$$

和之前的形式类似， $g(i-1)$ 到 $g(i)$ 相当于删除 $g(i-1, 1)$ 再插入 $-x$ 然后排序。

综上，我们只需要用一个数据结构维护插入、查询最小值、删除最小值。即我们可以用一个堆维护。与解法一复杂度相同。

解法二的另一种理解方法：

考虑对前 i 个点的图跑最小费用可行流，产生一个残存网络，然后采用类似解法一的方法，将满流的老鼠改成洞、满流的洞改成老鼠。我们考虑维护修改后的残存网络。

在后面加入新的点时，注意到后面的老鼠不可能进入前面的洞，后面的洞一定能被前面的老鼠进入，这意味着前面部分链上的流量对后面的部分毫无影响且前面部分修改后残存网络上的洞对后面部分毫无影响。

那么我们只需要记录前面部分残存网络上每个老鼠的代价即可。对于后续操作，加入一个老鼠相当于残存网络上加一个老鼠；加入一个洞的时候我们可能会产生增广路，即匹配之前的一个老鼠，我们只需要拿出代价最小的老鼠来判断即可，如果匹配成功对应的老鼠会变成洞、该洞变成老鼠。综合一下只需要用一个数据结构维护插入、找最小值、删除最小值，即我们可以用一个堆来维护。

例题 6. 维护单向链⁵

题目大意：单向链，链上边容量为 ∞ ，每群老鼠只有一只，每个洞只能进一只老鼠。要实现以下操作：

1. 在链首的前面加一个节点，该节点上有一只老鼠或一个容量为 1 的洞，然后将链首改为新的节点。

⁵笔者命制的练习题

2. 在链尾的后面加一个节点, 该节点上有一只老鼠或一个容量为 1 的洞, 然后将链尾改为新的节点。

操作次数为 $\Theta(n)$ 级别, 强制在线, 每次操作后输出当前状态下的最优答案。

解法

参考上一题解法二的另一种理解方法, 我们考虑维护一个修改后的残存网络。

由于老鼠和洞等价, 我们不妨只考虑加入老鼠。在前面加入的老鼠一定可以进入后面的任意一个洞, 在后面加入的老鼠一定不能进入前面的洞, 所以中间部分链上的流量对后续操作毫无影响, 我们只需要维护当前状态下残存网络上的每个老鼠和洞即可, 可以使用两个小根堆, 分别表示残存网络上的老鼠和洞。

复杂度 $\Theta(n \lg n)$ 。

例题 7. 贸易⁶

题目大意: 单向链, 链上边容量为 ∞ , 每群老鼠只有一只, 每个洞只能进一只老鼠。 q 次询问, 每次询问只保留一个区间 $[l, r]$ 中的点时的最小费用可行流权值。

解法

考虑对序列分治, 对于包含在区间 $[L, R]$ 中的询问, 我们取中点 $M = \lfloor \frac{L+R}{2} \rfloor$, 我们将询问分成三类:

1. $l \leq M < r$
2. $r \leq M$
3. $l > M$ 。

递归处理 2、3 两种情况, 分治到只有一个点时直接求出答案。

接下来我们只讨论第一种情况, 考虑上一题的方法, 我们可以用可持久化数据结构对 $[L, M]$ 的每个后缀和 $[M+1, R]$ 的每个前缀维护对应残存网络。这样我们可以将每个询问拆成两半, 然后考虑中间那条边的流量。

容易发现中间这条边的流量关于最优解是凸函数, 所以我们可以三分流量然后在两边的数据结构上查询对应的值。

我们要支持插入、查询最小值、删除最小值、查询 k 小值、查询前 k 小的和, 可以采用可持久化平衡树。

时间复杂度为 $\Theta((n+q) \lg^2 n)$, 空间复杂度为 $\Theta(n \lg n)$

笔者目前不清楚是否有时间复杂度或空间复杂度更优的解法, 欢迎读者来讨论相关问题。

本题可以扩展到树上, 将序列上的分治改为点分治即可。

⁶笔者命制的练习题

例题 8. 任务调度问题⁷

题目大意：单向链，链上边容量为 ∞ ，每群老鼠只有一只，老鼠代价均为 0，求恰好有 k 只老鼠进洞的最小代价。

解法

解法一：

考虑线段树解法，由于本题的增广路代价等价于对应的洞的代价，所以可以按洞代价从小到大尝试增广对应的边。容易得到 $\Theta(n \lg n)$ 的解法。

但我们发现事实上可以每次贪心地选择可行的靠后的老鼠，这样就变成了删点求后继问题，可以用并查集解决。复杂度 $\Theta(n\alpha(n))$ 。

解法二：

考虑 wqs 二分流量然后维护折线。

直接做的复杂度为 $\Theta(n \lg n \lg(\sum v_i))$

观察例题 5 的解法二，容易发现此时不需要考察 $-x$ 和 $g(i-1, 1)$ 的大小关系，即堆中每次插入和删除操作和查询的结果没有关系，所以可以用并查集把堆离线掉。

容易发现使用并查集时可以直接控制流量大小，因此不需要 wqs 二分。复杂度 $\Theta(n\alpha(n))$

小结：

上面两种解法本质相同，该算法还有其它的解释方法，这里不展开。

例题 9. (NOI2017) 蔬菜

题目大意：要卖若干天菜，每天最多只能卖 m 单位的菜。有 n 种蔬菜，第 i 种有四个指标： a_i, s_i, c_i, x_i ，表示其每单位蔬菜价格为 a_i ，第一次卖一单位这种蔬菜额外获得 s_i 的补贴，初始时库存为 c_i ，每天有 x_i 单位的蔬菜变质（每单位变质时间确定，对于所有 $d \times x_i \leq c_i$ 的 d ，有 x_i 单位蔬菜在第 d 天结束时变质，若 $(d-1) \times x_i \leq c_i < d \times x_i$ 则有 $c_i - (d-1) \times x_i$ 单位蔬菜在第 d 天结束时变质）。

q 次询问，第 j 次询问如果只有前 p_j 天卖菜最多能赚多少钱。

$m \leq 10, n, q, p_j \leq 10^5$

解法

可以看作第 i 种菜有一棵菜在第 $\lceil \frac{c_i}{x_i} \rceil$ 天变质，且代价为 $s_i + a_i$ ，其余若干棵菜的代价均为 a_i 。

解法一：

暴力建图的话就是对每一天建一个点 B_i ，加边 $(B_i, B_{i+1}, \infty, 0)$ ，然后前 p_j 天加边 $(S, B_i, m, 0)$ ，再对于每棵菜让对应的点向 T 连边。

⁷改编自参考文献 [8] 中的例题

由于 $m \leq 10$, 我们可以让每天连向 T 的边只保留前 m 小的, 然后依次加边用线段树模拟即可。另外本题可以让新加的边强制满流, 所以不需要考虑负环。

复杂度 $\Theta(nm \lg n)$

解法二:

考虑把问题转化成上一题的形式。

把每棵蔬菜看作是一个任务, 菜变质的日期看作是截止日期。“卖 p_j 天菜, 每天最多卖 m 棵”可以转化为“每天最多卖 m 棵菜, 总共卖出去的菜不能超过 $p_j \times m$ 棵”。这样天数的限制变成了流量限制, 直接套用并查集做法即可。

复杂度 $\Theta(n(m\alpha(n) + \lg n))$ ■

例题 10. 区间⁸

题目大意: 单向链, 每群老鼠只有一只, 每个洞只能进一只老鼠, 链上边容量均为 1, 给定 k, q 次修改, 每次修改一条链上边的边权, 修改后输出新图上正好 k 个老鼠进洞的最小代价。

解法

我们先考虑没有修改的情形。我们需要实现一个数据结构维护最短的增广路。我们大致可以在线段树上的每个结点维护如下信息: 该区间中能到达区间最左侧的代价最小的老鼠、能到达区间最右侧的代价最小的老鼠、能被区间外左侧的老鼠到达的代价最小的洞、能被区间外右侧的老鼠到达的代价最小的洞、整段区间正向边容量是否均大于 0、整段区间反向边是否均大于 0、该区间中的最优增广路。

这样我们可以实现合并操作。增广过程还需要区间对容量 +1 或 -1, 本题中每条边只有两种情况, 区间修改相当于修改区间里每条边的状态, 那我们在线段树维护时还需要记整段状态被反转后的若干信息。

这样我们可以设计了一个 $\Theta(n + k \lg n)$ 的算法来解决没有修改的情形。

对于有修改的情形, 由于边上容量只有 1, 所以我们只会产生 $O(1)$ 个负环或所需的增广路。我们只需要考虑如下四种情形:

1. 只经过 S 不经过 T 的负环
2. 只经过 T 不经过 S 的负环
3. S 到 T 的增广路
4. T 到 S 的增广路

用四棵线段树维护即可。复杂度 $\Theta(n + (k + q) \lg n)$, 实测运行速度不是很理想。

笔者目前不清楚本题是否有更为简洁或效率更高的方法, 欢迎前来讨论。 ■

⁸笔者命制的练习题

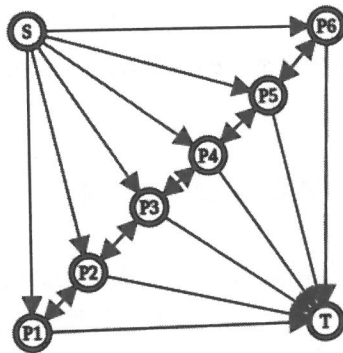
小结

本节对一些单向链问题的特例及相关拓展问题进行了讨论。主要从线段树维护流量和维护折线两个角度来优化一些费用流问题。

对于有更复杂容量限制的问题，一部分可以从本节的基础模型拓展而来，也有一些由于缺乏某些关键性质导致难以设计合适的算法。本节篇幅有限，不再展开讨论。

4.2 双向链

4.2.1 定义与约定



(e) 示例图

双向链即在单向链的基础上对于每个 $1 \leq i < n$ 加入了 $(i+1, i)$ 的边。

在该问题上，我们无法忽略链上的边权。我们不妨设 $(i, i+1)$ 的边权为 x_i ， $(i+1, i)$ 的边权为 y_i ，满足 $x_i + y_i \geq 0$ 。

4.2.2 例题

例题 11. 双向链⁹

题目大意：双向链，链上边容量均为 ∞ ，每群老鼠只有一只，每个洞只能进一只老鼠。

解法

解法一：

考虑线段树做法。参考例题 5. 的解法一，这里我们链上的边权可能会在做的过程中改变，为了保证复杂度我们考虑先加入所有的洞，然后按编号从小到大加入老鼠。

⁹笔者命制的练习题

可以发现负环的问题依然可以通过将老鼠修改为洞、将洞修改为老鼠实现。因此只需要考虑增广路。

我们每加入一只老鼠寻找增广路，考虑对于每个 j 维护 i 到 j 的最短路。我们分两种情况讨论：

case 1: $j \geq i$

最短路即为 $\sum_{i \leq k < j} x_k$ 。

case 2: $j < i$

对于路径上的一条边 $(k, k+1)$ ，若其反平行边容量 > 0 ，则代价为 $-x_k$ ，否则为 y_k 。

容易发现对于每个 k ，当 $i > k$ 时 $(k, k+1)$ 的反平行边流量在做的过程中单调不降，所以至多一次从有反平行边的状态变为没有反平行边的状态，所以我们可以每次暴力找出这样的边修改代价。

综上，对于最短路为维护我们只需要建一棵线段树实现单点修改、区间加和全局查询，然后可以再建一棵线段树用于维护每条边的流量以及找出 **case 2** 中代价变化的边。

时间复杂度 $\Theta(n \lg n)$ 。

解法二：

考虑维护折线，按编号从小到大遍历链上节点加入对应的老鼠或洞。这里老鼠与洞等价，不妨只讨论洞。

设此时 x_i 的前缀和为 x ， y_i 的前缀和为 y ， T 入边的边权为 c 。

我们将向右走的边代价定为 $c - y$ ，向左走的代价定为 $c + x$ 。

定义 $f(j)$ ： $j \geq 0$ 时表示当前剩余流量为 j 的最小代价， $j < 0$ 时表示需要从后面补充恰好 $-j$ 的流量的最小代价。

$f(j)$ 可能不是凸的，但我们根据 **定理 3.1**。分别构造图容易证明以下函数都是凸的：

$$1. f^+(j) = f(j) (j \geq 0)$$

$$2. f^-(j) = f(j) (j \leq 0)$$

$$3. h(j) = \begin{cases} |j| * y & j < 0 \\ 0 & j = 0 \\ |j| * x & j > 0 \end{cases}$$

$$4. f(j) + h(j)$$

令更新后的 dp 状态为 $f'(j)$ ，我们可以写出转移方程：

$$f'(j) = \begin{cases} \min(f(j), f(j+1) + c + x) & j \geq 0 \\ \min(f(j), f(j+1) + c - y) & j < 0 \end{cases}$$

定义

$$g(j) = \begin{cases} f(j) - f(j-1) & j > 0 \\ 0 & j = 0 \\ f(j) - f(j+1) & j < 0 \end{cases}$$

$g'(j)$ 同理。我们不妨将 $g(j)j < 0$ 的部分表示为可重集 G_0 , $j > 0$ 的部分表示为可重集 G_1 。

我们分情况讨论：

case 1: $f(0) \leq f(1) + c + x$

条件等价于 $g(1) \geq -c - x$, 此时 $f'(0) = f(0)$ 。

对于 $j > 0$ 的部分, 有 $g(j) \geq g(1) \geq -c - x$, 所以该部分不会改变, 即 $G'_1 \leftarrow G_1$

对于 $j < 0$ 的部分, 问题转化为类似单向链的形式, 相当于 $G'_0 \leftarrow G_0 \cup \{c - y\}$

case 2: $f(0) > f(1) + c + x$

条件等价于 $g(1) < -c - x$, 此时 $f'(0) = f(1) + c + x = f(0) + g(1) + c + x$ 。

对于 $j > 0$ 的部分, 类似于单向链的形式, 相当于 $G'_1 \leftarrow (G_1 - \{\min(G_1)\}) \cup \{-c - x\}$ 。

根据 $f + h$ 的凸性可以发现: $f(1) - f(0) + x \geq f(0) - f(-1) - y$ 。

即 $f(-1) \geq f(0) - g(1) - x - y > f(0) + c - y$

此时 $f'(-1) = f(0) + c - y$, $g'(-1) = f'(-1) - f'(0) = -x - y - g(1)$

对于 $j < -1$ 有: $g(j) \geq g(-1) = f(-1) - f(0) \geq c - y$, 所以剩余部分不会改变。

这部分相当于: $G'_0 \leftarrow G_0 \cup \{-x - y - g(1)\}$

综上, 我们可以建小根堆 Q_0, Q_1 来维护可重集 G_0, G_1 , 时间复杂度 $\Theta(n \lg n)$ 。

伪代码:

Algorithm 2 Extend(Q_0, Q_1, f_0, x, y, c)

if $Q_1.empty() \vee Q_1.top() \geq -c - x$ **then**

$Q_0.push(c - y)$

else

$val \leftarrow Q_1.top() + c + x$

$f_0 \leftarrow f_0 + val$

$Q_1.pop()$

$Q_1.push(-c - x)$

$Q_0.push(c - y - val)$

end if

解法三

我们如果将解法二中的 $f + h$ 作为 dp 状态进行维护。

现在要支持两种操作: 在后面加一条链上的边, 加入一个新的老鼠或洞。

对于前者的 dp 转移, 可以写成 $(f+h')(x) = (f+h)(x) + h'(x) - h(x)$, 相当于整体和两段一次函数相加。

对于后者, 不妨只考虑加入洞, 该操作形如: $(f'+h)(x) = \min\{(f+h)(x+1)+c, (f+h)(x)\}$, 相当于凸函数 $\min$ 卷积。

这两部分均可以用平衡树实现, 复杂度 $\Theta(n \lg n)$ 。

该算法可以拓展到费用与流量的关系更复杂的问题上。

■

例题 12. (UER # 8) 雪灾与外卖¹⁰

题目大意: 双向链, 链上边容量均为 ∞ , 每群老鼠只有一只, 每个洞有容量、可以容纳多只老鼠。每只老鼠强制进洞。

解法

解法一:

可以采用类似上一题的线段树的解法, 这里不再赘述。

解法二:

考虑维护折线, 强制进洞的老鼠代价视为 $-\infty$, 考虑将权值相同的放在一起维护。

在堆中存放一个 `pair` 类型, 其中 `first` 表示权值, `second` 表示这种权值的个数。

根据上一题的解法二可以写一份伪代码:

Algorithm 3 $\text{Extend}(Q_0, Q_1, f_0, x, y, c, cnt)$

```

1: //cnt 表示洞的容量
2:  $cnt_s \leftarrow cnt$ 
3: while  $cnt > 0$  do
4:   if  $Q_1.empty() \vee Q_1.top().first \geq -c-x$  then
5:      $Q_0.push(\text{make\_pair}(c-y, cnt))$ 
6:     break
7:   else
8:      $s \leftarrow Q_1.top()$ 
9:      $Q_1.pop()$ 
10:     $val \leftarrow s.first + c + x$ 
11:     $cnt_0 \leftarrow \min(s.second, cnt)$ 
12:     $f_0 \leftarrow f_0 + val \times cnt_0$ 
13:     $Q_0.push(\text{make\_pair}(c-y-val, cnt_0))$ 
14:     $s.second \leftarrow s.second - cnt_0$ 
15:     $cnt \leftarrow cnt - cnt_0$ 
    
```

¹⁰<https://uoj.ac/problem/455>

```

16:   if s.second>0 then
17:       Q1.push(s)
18:   end if
19: end if
20: end while
21: if cnts - cnt > 0 then
22:     Q1.push(make_pair(-c-x, cnts-cnt))
23: end if
    
```

发现加入洞的时候这里的 while 循环可能会运行多次，我们来分析一下复杂度。

我们不妨设其执行一轮循环消耗一单位时间。

以加入洞时的 Q_0, Q_1 为准：

使用势能法做摊还分析，定义势能 $\Phi(D) = Q_1.size()$ 。初始时势能为 0。

容易发现 Q_1 中只会存在 $second = 1$ 的 pair。

当加入老鼠时，只会循环一次，实际代价和势能变化量均为 $O(1)$ 单位时间。

当加入洞时，最后一次循环实际代价和势能变化量均为 $O(1)$ 单位时间。对于其它若干轮循环，每轮必定使 Q_1 中的元素减少一个，所以摊还代价为 $1 - 1 = 0$ 。

做完后的势能一定不小于 0，所以该摊还分析给出了一个上界，即总循环轮数为 $O(n+m)$ 级别，总复杂度为 $\Theta((n+m) \lg(n+m))$

例题 13. (IOI2017) Wiring

题目大意：通讯塔上有 n 连接点，第 i 个的坐标为 $x_i (x_i \geq 0)$ ，其中一些是红色的，另一些是蓝色的。一条电线可以链接一个红色连接点和一个蓝色连接点，若两个点的坐标分别为 x_i 和 x_j ，则代价为 $|x_i - x_j|$ 。要求每个连接点至少连一根电线，最小化总代价。

解法

本题有更优秀的非模拟费用流做法，这里暂不展开。

本题可以将至少连一根电线的条件看作连一条 $(\infty, 0)$ 和一条 $(1, -\infty)$ 的边，然后进行费用流建图。

解法一：

线段树解法，直接照搬之前的做法即可。

解法二：

考虑维护折线，直接照搬上一题的解法二。

设当前的坐标为 x ，当容量为 ∞ 的边进行多次匹配时必定会减少两个堆中 $first > -x$ 的 pair，而这样的 pair 不会再次被容量为 ∞ 的边匹配。因此可以保证复杂度。具体证明只需将上一题复杂度分析的势能改为 $first > -x$ 的 pair 个数再做分析即可，这里不再赘述。 ■

**例题 14.** (CTSC2010) 产品销售

题目大意：双向链，链上边容量均为 ∞ ，每群老鼠有多只，每个洞有容量、可以容纳多只老鼠。每只老鼠强制进洞。

解法

解法一：

线段树做法可以直接套过来。

本题中由于老鼠强制进洞，所以不需要考虑负环的问题。

如果没有这个要求，我们在做的过程中会将老鼠改为洞或将洞改为老鼠，对于一只老鼠或洞，容易证明这样的次数不会超过两次。而每找到一条增广路至少会使一个 pair 的状态被改变。复杂度依然可以保证。

复杂度 $\Theta(n \lg n)$

解法二：

考虑折线做法，我们直接照搬上面的代码复杂度无法保证，但我们可以参考**例题 11**的**解法三**用平衡树维护。时间复杂度依然可以做到 $\Theta(n \lg n)$ 。

例题 15. 工厂¹¹

题目大意：双向链，链上边有容量，每群老鼠有多只，每个洞有容量，求最小费用可行流的权值。另有 q 次独立的询问，每次询问形如钦定某对链上双向边的流量求最小费用可行流的权值。

解法

参考上一题的**解法二**，对于链上边的容量限制，我们在操作时相当于删去平衡树上前若干小的元素和前若干大的元素。

对于询问，可以对每个前缀和每个后缀分别处理出对应的结构在平衡树上进行询问。

4.2.3 小结

对于双向链的模拟最小费用可行流问题，我们通常还是可以考虑用数据结构维护流量和维护折线两种方法。

对于双向链动态加边问题，笔者目前没有想到合适的方法，还需进一步探讨。

¹¹改编自笔者命制的练习题

4.3 树

4.3.1 例题

例题 16. (LibreOJ NOI Round# 2) 黄金矿工

题目大意： n 个藏宝点构成了一棵树，每条边从父亲指向儿子。有若干矿工和黄金，一个矿工最多只能挖一个黄金，让矿工 x 挖黄金 y 的收益是 $P_x + Q_x + \sum_{e \in \text{path}(x,y)} c_e$ ，其中 $\text{path}(x,y)$ 是 x 到 y 的最短路径上的边集。 q 次操作，每次加入一个新的黄金或者矿工，求最大收益。

解法

本题要动态增加边容量，用折线难以维护，所以可以考虑用数据结构模拟增广路。

考虑拓展单向链的线段树做法，只需要把线段树改成动态树分治维护每条边流量和询问可达到的最小值即可，若 n 和 q 同阶时间复杂度即为 $\Theta(n \lg^2 n)$ 或 $\Theta(n \lg n)$ 。 ■

例题 17. (ICPC World Finals 2018) 征服世界¹²

题目大意：一棵 n 个点由无向边连接的树，初始时第 i 个点有 x_i 个军队，一个军队沿着一条代价为 $c(c > 0)$ 的边移动到另一个点花费的代价为 c 。最终要对于每个点 i ，其军队个数不小于 y_i 。求最小代价。

$\sum x_i$ 与 n 同阶。

解法

容易建出有源汇最小费用可行流的模型。

解法一：

将双向链的线段树解法扩展到树上。按 dfs 序加一侧边可以保证树上每条边代价变化次数为 $O(1)$ ，用树链剖分等方法维护即可。

时间复杂度 $\Theta(n \lg^2 n)$ 或 $\Theta(n \lg n)$

解法二：

考虑转化为最小费用可行流维护折线。对于点 x 的子树，对于所有 j 维护其向父亲的边流量为 j 的最小代价。

可以采用类似**例题 11 解法三**的方法用平衡树进行维护，合并子树的 dp 转移形如 $f_0(x) = \min_{y+z=x} \{f_1(y) + f_2(z)\}$ ，为凸函数 $\min$ 卷积，可以用平衡树启发式合并实现，其它部分与之前类似。复杂度 $\Theta(n \lg^2 n)$

也可以对链上边权和进行树上差分然后折线的正负部分拆开分别用可并堆实现，复杂度 $\Theta(n \lg n)$ ¹³ ■

¹²<https://loj.ac/p/6405>

¹³详见 <https://www.csc.kth.se/austrian/icpc/finals2018solutions.pdf>



4.3.2 小结

对于树上模拟费用流问题，我们通常可以先考虑对应的链上模拟费用流问题，再将其拓展到树上。

5 总结

本文从费用流的定义与性质出发，对一部分OI中出现的模拟费用流问题进行了整理和拓展。

模拟费用流问题还有诸多方面可以进一步探讨，由于笔者能力有限，这里没有展开。

希望本文可以起到抛砖引玉的作用，让读者更好地了解这类问题，期待有更多的模拟费用流问题在信息学竞赛中出现。

6 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练杨耀良的指导。

感谢浙江省余姚中学诸一行、史习艺、刘弘老师的关心和指导。

感谢父母的养育之恩。

感谢彭思进学长、杨琰同学为本文审稿并提出建议。

感谢彭思进学长、杨琰同学、戴江齐同学、赵雨扬学长、方科晨学长、李欣隆学长与我交流、给我启发。

感谢陈江伦、高睿泉等学长的论文或课件给我启发。

参考文献

- [1] 陈江伦,《模拟费用流问题》,WC2019 课件。
- [2] Ahuja, Ravindra K.; Magnanti, Thomas L.; Orlin, James B. (1993). "Network Flows. Theory, Algorithms, and Applications" Prentice Hall.
- [3] 姜志豪,《网络流的一些建模方法》,2016 年集训队论文。
- [4] 高睿泉,《浅谈保序回归问题》,2018 年集训队论文。
- [5] 丁晓漫,《再探线性规划对偶在信息学竞赛中的应用》,2021 年集训队论文。
- [6] 董克凡,《浅谈线性规划与对偶问题》,2016 年集训队论文。



- [7] 杨宇辰,《模拟费用流小记》, <https://www.luogu.com.cn/blog/command-block/mu-ni-fei-yong-liu-xiao-ji>
- [8] Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein,《Introduction to Algorithms》, 3rd Edition.

