2020 年信息学奥林匹克
中国国家集训队论文

目 录

浅谈利用分散层叠算法对经典分块问题的优化

成都七中 蒋明润

摘 要

分散层叠是一种在多个按一定组织结构的数列中对一个数进行加速二分查找的技术。本文介绍了这个算法对经典分块题的优化并展示了它的一些应用。

前言

分散层叠算法已经产生了 20 多年，但是尚未在算法竞赛界得到广泛应用。本文试图利用该算法对经典分块问题进行优化，希望在算法竞赛中引入分散层叠算法。本文第一节介绍了分散层叠算法及其意义，第二节介绍经典分块题及其常用解法，第三节则利用分散层叠算法对经典分块题进行了算法优化，并与传统算法进行了比较，同时对经典题进行了扩展。

1 分散层叠

本章介绍了分散层叠（Fractional Cascading）算法。并将通过一个具体的例子来说明该算法的含义。

1.1 简介

在计算机科学中，分散层叠是一种在多个按一定组织结构的数列中对一个数进行加速二分查找的技术，其中第一次查找是普通的二分查找，时间复杂度为 $O(\log n)$，后面的二分查找会比第一次更快。分散层叠最早由 Chazelle 和 Guibas 在 1986 年发表的两篇文章中 [1, 2] 提出。顾名思义，Fractional 是指“零碎的、微小的”，“Cascade”则是一个比喻，指“像小瀑布一样层叠的”。Chazelle 和 Guibas 原文给出了一幅图画，如图 1 所示，它相当形象地揭示了这个名词背后的深刻含义：像瀑布一样从高到低逐渐分散成越来越细的支流，再层层叠叠地覆盖 [3, 4]。

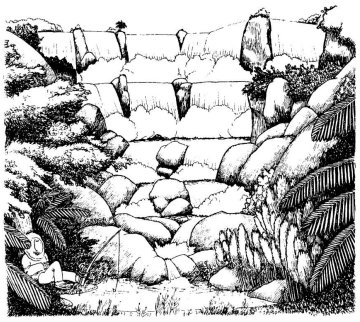


图 1: Chazelle 和 Guibas 原文给出的图画

1.2 举例说明

我们将用例题的形式来描述这个算法。

考虑下面这个问题：给出长度和为 n 的 k 个有序数列，第 i 个数列记作 L_i，要求建立一个数据结构，支持对同一个数 q 在 k 个数列中分别进行二分查找（查询第一个大于等于 q 的数的位置），例如下面这个例子中，$k=4$，$n=17$。

$$L_1 = 24, 64, 65, 80, 93$$
$$L_2 = 23, 25, 26$$
$$L_3 = 13, 44, 62, 66$$
$$L_4 = 11, 35, 46, 79, 81$$

最简单的方法是把每个序列分开存放。如果我们这么做，我们只需要 $O(n)$ 的空间复杂度。但是查询时需要对每个序列分别进行一次二分查找，在最坏情况下，k 个序列长度相等，需要 $O(k\log(\frac{n}{k}))$ 的时间复杂度。

第二种方法可以支持更快的查询，但是需要消耗更多的空间：我们可以把这 k 个序列合并成一个大的序列 L，并对 L 中的每个元素记录下它在原来的 k 个序列中进行二分查找的结果。如果我们把合并后的序列中每个元素记作 $x[a,b,c,d]$，x 是数值，a,b,c,d 为 x 在原来的序列中的后继所在的位置（从 0 开始标号），那么：

$$\begin{aligned} L =& 11[0,0,0,0], 13[0,0,0,1], 23[0,0,1,1], 24[0,1,1,1], 25[1,1,1,1], 26[1,2,1,1], 35[1,3,1,1], \\ & 44[1,3,1,2], 46[1,3,2,2], 62[1,3,2,3], 64[1,3,3,3]65[2,3,3,3], 66[3,3,3,3], \\ & 79[3,3,4,3], 80[3,3,4,4], 81[4,3,4,4], 93[4,3,4,5] \end{aligned}$$

这种方法可以做到查询的时间复杂度为 $O(k+\log n)$：直接在 L 中进行二分查找，然后返回在元素 x 中记录的信息。但是它需要 $O(nk)$ 的空间。

分散层叠算法支持对同样的问题同时做到最优的时间复杂度和空间复杂度：查询的时间复杂度为 $O(k+\log n)$，空间复杂度为 $O(n)$。建立 k 个新的有序序列，第 i 个记作 M_i。其

中最后一个序列 M_k 和 L_k 相同。前面的每一个序列 M_i 由 L_i 和 M_{i+1} 的从第二个元素开始隔一个采样一个得到的子序列归并得到，并对 M_i 中每个元素记录下它在 L_i 和 M_{i+1} 中二分查找的结果。例如，对于前面给出的 4 个序列，我们有：

$$M_1 = 24[0,1], 25[1,1], 35[1,3], 64[1,5], 65[2,5], 79[3,5], 80[3,6], 93[4,6]$$

$$M_2 = 23[0,1], 25[1,1], 26[2,1], 35[3,1], 62[3,3], 79[3,5]$$

$$M_3 = 13[0,1], 35[1,1], 44[1,2], 62[2,3], 66[3,3], 79[4,3]$$

$$M_4 = 11[0,0], 35[1,0], 46[2,0], 79[3,0], 81[4,0]$$

如果我们想要对 $q = 50$ 进行查询，我们先在 M_1 中进行二分查找，得到 $64[1,5]$。“1”表示在 L_1 中二分查找的结果是 $L_1[1] = 64$，“5”表示在 M_2 中进行二分查找的结果大致在下标 5。准确地说，是在下标 5 的 $79[3,5]$ 或者前面一个位置的 $62[3,3]$ 。将 q 和 62 进行比较，得知正确的查找结果为 $62[3,3]$。通过相同的方法，可以得到 q 在 L_2 ， M_3 ， L_3 ， M_4 中二分查找的结果。

更一般地说，对于任何一个这样的结构，我们可以先在 M_1 中进行二分查找，然后对于任何一个 i ，我们可以通过 q 在 M_i 中的位置定位出 q 在 L_i 中的位置和 q 在 M_{i+1} 中的位置。q 在 M_i 中查询的结果指向的 M_{i+1} 中的位置要么是 q 在 M_{i+1} 中查询的结果，要么只相差一个位置，所以可以通过一次比较确定。总时间复杂度为 $O(k + \log n)$ 。

在前面的例子中，四个新序列的长度总和是 25，不超过 $2n$。一般来说，M_i 的长度不超过 $|L_i| + \frac{1}{2}|L_{i+1}| + \frac{1}{4}|L_{i+2}| + \cdots$ 。所有 M_i 的长度总和不超过 $\sum |M_i| \le \sum |L_i|(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots) = 2n = O(n)$。

1.3 普遍情况

考虑给出一张有向无环图，每个节点的入度和出度都不超过一个常数 d，每个节点上存有一个有序序列 L_u。一次查询需要对一个数 q 在一条路径上的所有节点 u 上存有的序列 L_u 中进行二分查找。对于前面的例子，给出的有向图是一条长度为 4 的链。

对每一个节点 u 建立一个新序列 M_u ， M_u 由 L_u 和 u 的所有后继 v 上建立的序列 M_v 按一定比例均匀选取元素得到的序列归并得到（如果要做到 $O(n)$ 的空间复杂度，选取的比例必须小于 $\frac{1}{d}$ ），并对每个元素记录下它在 L_u 和每个 M_v 中二分查找的结果。对于前面的例子，$d = 1$ ，选取的比例为 $\frac{1}{2}$。

对于一次查询，我们先在路径的起点 s 上建立的序列 M_s 中进行一次 $O(\log n)$ 的二分查找。假设我们知道 q 在 M_u 中二分查找的结果，要对于 u 的一个后继 v 求出 q 在 M_v 中二分查找的结果。如果我们建立数据结构时选取的比例是 $\frac{1}{r}$ ，那么 q 在 M_u 中二分查找的结果指向的 M_v 中的位置和 q 在 M_v 中二分查找的正确结果的距离不会超过 r。因为 r 是一个常数，所以可以在 $O(1)$ 的时间复杂度内求出 q 在 M_v 中二分查找的正确结果。

设查询的路径长度为 k，则单次查询的时间复杂度为 $O(k + \log n)$。

另外分散层叠可以支持 $O(\log n)$ 在一个序列中加入一个元素，但查询复杂度会变为 $O(\log n + k \log\log n)$ ，不过与本文关系不大，这里略去。

2 经典分块题

本章介绍一道 OI 经典分块题及其传统解法。下一章我们将讨论利用分散层叠算法对这道题进行优化。

2.1 题目描述

给出一个长度为 n 的序列，有 n 次操作，操作分为两种：

1 修改操作：给定 l, r, x，对于所有 $l \le i \le r$，将 a_i 增加 x。

2 查询操作：给定 l, r, x，求出在所有 $l \le i \le r$ 中，有多少个 i 满足 $a_i \le x$。

2.2 常用解法（算法 1）

将序列中每相邻的 B 个数分成一块，总共 $O(\frac{n}{B})$ 块。每个块内存下这个块中所有元素排序后的结果。这样对于一个区间 $[l, r]$ ，最多只有 2 个块被不完全覆盖，其它块要么是完全覆盖，要么没有被覆盖。

对于修改操作，不完全覆盖的块需要暴力重构。此时需要重新求出排序后的结果，如果暴力排序，时间复杂度为 $O(B \log B)$, 不过注意到对于两个都没有被修改的数或者对于两个都被修改的数，它们的大小关系是不会改变的，所以如果我们排序时记录下元素在排序前的位置，就可以 $O(B)$ 将没有被修改的数和被修改的数分别排序，然后归并即可，时间复杂度为 $O(B)$ 。完全覆盖的块可以记录一个懒标记，表示将块内所有元素全部加上一个数。每个块的时间复杂度为 $O(1)$, 总共 $O(\frac{n}{B})$ 个块，总时间复杂度为 $O(\frac{n}{B})$ 。

对于查询操作，不完全覆盖的块可以直接枚举所有元素进行查询，时间复杂度为 $O(B)$，完全覆盖的块需要在排序后的数组上二分，每个块的时间复杂度为 $O(\log B)$, 总时间复杂度为 $O(\frac{n \log B}{B})$。

综上，单次操作的时间复杂度为 $O(B + \frac{n \log B}{B})$, 总时间复杂度为 $O(nB + \frac{n^2 \log B}{B})$, 取 B 为 $\sqrt{n \log n}$ 时时间复杂度最优，为 $O(n \sqrt{n \log n})$。

2.3 离线算法（算法 2）

本节讨论离线算法。

在本题中只有以下输入范围能使用本节算法：输入的所有数均为绝对值不超过 2^ω 的整数，且 $\omega = O(\log n)$。

注意到上一节中算法的瓶颈在二分查找，本节考虑对这部分进行优化。

然而优化单次二分查找是很困难的（可以使用 y-fast tree 等数据结构，但不实用）。因此，要优化算法，需要从优化多次二分查找的总时间复杂度入手。其中有两种方法，第一种方法是优化多次询问在一个块中进行的二分查找，第二种方法是优化单次询问在多个块中进行的二分查找。本节讨论第一种方法，第二种方法将在下一章中讨论。

考虑预先存下所有询问，逐块处理。在对一个块进行重构时求出之前所有在这个块中进行的二分查找的结果。这样问题转化成在一个长度为 B 的序列中进行 Q 次二分查找。

注意到如果每次二分查找的数单调递增，可以利用单调性做到 $O(B+Q)$ 的时间复杂度。所以如果可以快速地将 Q 个数排序，就可以快速地求出 Q 次二分查找的结果。

在 $\omega = O(\log B)$ 时，可以以 B 为底进行基数排序，时间复杂度为 $O(\frac{(B+Q)\omega}{\log B}) = O(B+Q)$。这样，原分块题的时间复杂度可以优化至 $O(nB + \frac{n^2}{B})$，取 $B = \sqrt{n}$ 时最优，为 $O(n\sqrt{n})$。

3 基于分散层叠的算法

上一章介绍了 OI 经典分块题及其传统解法。本章我们将讨论利用分散层叠算法对这道题进行优化。

基数排序的做法可以做到 $O(n\sqrt{n})$ 的时间复杂度，然而如果值域不是整数，或者强制在线（读入一个操作之前必须求出之前所有询问的结果，所以不能预先存下所有询问），就不适用了。

单独的一次二分查找难以优化，不过注意到我们需要的不是单独的二分查找，而是对同一个数在 $O(\frac{n}{B})$ 个序列中进行二分查找，所以可以考虑使用分散层叠算法。然而分散层叠算法不支持修改，所以需要一些处理。

3.1 $O(n\sqrt{n\log\log n})$ 做法（算法 3）

清华大学蔡承泽在笔者之前想出了时间复杂度为 $O(n\sqrt{n\log\log n})$ 的算法，但他并没有公开详细内容，本节中所述做法是笔者根据一些相关的讨论推断出的一种可能的做法，不知道是否与蔡学长做法一致。无论如何，在此对蔡承泽学长表示感谢。

在本算法中，建立分散层叠的方法为：在每一个块中均匀选取 $\frac{1}{D}$ 的元素建立分散层叠。这样在二分查找的时候可以根据分散层叠定位出一个距离相差不超过 D 的位置，从而对每个块分别做到 $O(\log D)$ 的时间复杂度。用此方法每连续 D 个块建立一组分散层叠，共 $O(\frac{n}{BD})$ 组。

执行修改操作的时候，至多只有 2 组分散层叠的结构会被破坏（别的分散层叠只可能对所有数加上同一个数，这不会改变分散层叠的结构）。由于每一个块只选取了 $O(\frac{B}{D})$ 个元

素建立分散层叠，每组分散层叠只在 D 个块之间建立，所以重构一组分散层叠的时间复杂度为 $O(B)$ 。

对于查询操作，需要在每组分散层叠中进行一次查询，在一组分散层叠中查询的时间复杂度为 $O(D+\log B)$, 总共 $O(\frac{n}{BD})$ 组，总时间复杂度为 $O(\frac{n}{B}+\frac{n\log B}{BD})$, 另外对每一个块还要进行一次 $O(\log D)$ 的二分查找，总时间复杂度为 $O(\frac{n\log D}{B})$, 另外还要对不完全覆盖的块进行 $O(B)$ 的枚举。

综上，单次操作的时间复杂度可以优化至 $O(B+\frac{n\log D}{B}+\frac{n\log B}{BD})$，总时间复杂度为 $O(nB+\frac{n^2\log D}{B}+\frac{n^2\log B}{BD})$, 取 $B=\sqrt{n\log\log n}$,$D=\log n$ ，时间复杂度为 $O(n\sqrt{n\log\log n})$。

3.2 $O(n\sqrt{n})$ 做法（算法 4）

如果我们采用 1.2 节中的合并方式，已经难以继续优化，接下来本小节将从另一种合并方式入手，对问题进行进一步的优化。

建立一棵有 $\frac{n}{B}$ 个叶节点的线段树，线段树的叶节点依次存放每个块中维护的排序后的序列，非叶节点存放的序列由两个子节点所存放的序列中分别按一定比例均匀选取元素得到的序列归并得到，并记录下每个元素在两个子节点所存的序列中二分查找的结果。（即在 1.3 节中，让线段树中每个非叶节点连向它的两个子节点形成有向无环图，叶节点的 L_u 为每个块中维护的序列，非叶节点的 L_u 为空序列得到的分散层叠结构）。

执行修改操作时，根据线段树的性质，只有 $O(\log\frac{n}{B})$ 个节点所对应的区间会被不完全覆盖。而如果一个节点对应的区间被完全覆盖，那么这个节点的子树中所有元素的数值会被加上一个数，但是结构不会改变。所以只有 $O(\log\frac{n}{B})$ 个节点需要重新计算存放的序列。当选取元素的比例为 $\frac{1}{2}$ 时，每个节点存放的序列长度都是 $O(B)$ ，总时间复杂度为 $O(B\log\frac{n}{B})$ ，不够优秀。但是，如果我们用更小的比例，比如 $\frac{1}{3}$ 时，线段树的一层中每个节点存放的序列长度都是下一层每个节点的 $\frac{2}{3}$, 而根据线段树的性质，每一层中只有 $O(1)$ 个节点会被不完全覆盖。所以总时间复杂度不超过 $O(B(1+\frac{2}{3}+\frac{4}{9}+\frac{8}{27}+\cdots))=O(B)$。

对于查询操作，与 1.3 节中类似，我们可以先在根节点中进行二分查找，然后根据在一个节点中二分查找的结果 $O(1)$ 确定出在两个子节点中二分查找的结果，总时间复杂度为 $O(\frac{n}{B}+\log n)$ ，另外还要对不完全覆盖的块进行 $O(B)$ 的枚举。

综上，单次操作的时间复杂度为 $O(B+\frac{n}{B}+\log n)$，总时间复杂度为 $O(nB+\frac{n^2}{B}+n\log n)$，取 $B=\sqrt{n}$ 时，时间复杂度最优，为 $O(n\sqrt{n})$ 。

3.3 算法小结

综上所述，各种算法的时间复杂度如表 1 所示，其中算法 1 为传统算法。算法 2 虽然也达到了优化的目的，但是只适用于部分输入情况。算法 3 和算法 4 都是利用分散层叠优化的算法，使用都不受限制，其中算法 4 在已知算法中最优。

算法	时间复杂度	备注
算法 1	$O(n\sqrt{n\log n})$	
算法 2	$O(n\sqrt{n})$	只适用于部分输入情况
算法 3	$O(n\sqrt{n\log\log n})$	
算法 4	$O(n\sqrt{n})$	

表 1: 各种算法的比较

3.4 算法应用扩展

分散层叠算法不仅可以应用于本道经典题，还可以扩展应用到一些其它题目中。

3.4.1 题目 1

给出一个长度为 n 的序列，有 n 次操作，操作分为两种：

1 修改操作：给定 l, r, x，对于所有 $l \le i \le r$，将 a_i 增加 x。

2 查询操作：给定 l, r, x，求出区间 $[l, r]$ 有多少个子区间 $[l', r']$，满足 $[l', r']$ 中所有数均不超过 x。[1]

注意到对于两个区间 $[l, mid]$ 和 $[mid+1, r]$，如果我们知道它们的答案（这里的“答案”包括查询操作的答案，区间中第一个大于 x 的数的位置，区间中最后一个大于 x 的数的位置，下同），我们可以求出区间 $[l, r]$ 的答案。

将每相邻的 B 个数分成一块，共 $O(\frac{n}{B})$ 块。对于一次查询，可以将区间 $[l, r]$ 拆分成 $O(\frac{n}{B})$ 个长度不超过 B 的区间，其中只有 $O(1)$ 个区间不是完整的一块。对于不是完整的一块的区间，可以拆分成 $O(B)$ 个长度为 1 的区间处理，对于完整的一块，求出块中所有元素排序后的序列，并求出块中每一个元素作为 x 时这个块的答案。后者可以通过以下方法得到：

考虑按递增的顺序处理，将所有大于 x 的数按出现的位置顺序加入链表中（为方便处理，可以在链表两端加入特殊节点）。随着 x 的增大，链表中的元素将逐渐被删除。求这个块的答案时，求出区间中第一个大于 x 的数的位置和区间中最后一个大于 x 的数的位置可以直接通过特殊节点的指针得到，求查询操作的答案时需要考虑删除一个节点对答案的影响，显然所有包含这个节点而不包含这个节点的前驱和后继的所有区间将被加入答案，而不会有其它影响。（这一步的时间复杂度为 $O(B)$，所以不会增加对块进行重构的时间复杂度）

求出这个之后，剩下的部分和第二章中的经典分块题做法基本相同，可以用 3.2 节中的方法进行优化。

[1] 本题改编自 Comet OJ Contest #7 F https://cometoj.com/contest/52/problem/F?problem_id=2426

3.4.2 题目 2

给出一个长度为 n 的序列，有 n 次操作，操作分为两种：

1 修改操作：给定 l, r, x，对于所有 $l \le i \le r$，将 a_i 增加 x。

2 查询操作：给定 l, r, x，求出区间 $[l, r]$ 的所有子区间中，区间中所有元素的和最大是多少。

下面先讲述官方题解的做法。

注意到与上一题类似，对于两个区间 $[l, mid]$ 和 $[mid + 1, r]$, 如果我们知道它们的答案（此处“答案”包括区间最大子段和（即查询操作的答案），区间最大前缀和，区间最大后缀和），我们可以求出区间 $[l, r]$ 的答案。

仍然可以考虑每相邻的 B 个数分成一块，共 $O(\frac{n}{B})$ 块。关键在于如何处理完整的一块的情况。

对完整的一块的处理需要在整块中所有数全部加上同一个数的情况下维护答案，首先对这种情况进行处理。

考虑使用线段树进行维护。对于一个节点，需要维护这个节点所代表的区间中所有数加上同一个数 x 时这个区间的答案。不难证明这是一个关于 x 的分段一次函数，且段数与区间的长度同阶。一个节点上维护的信息可以由它的两个子节点合并得到，合并只需要将分段函数相加或者取 max ，可以在 $O(len)$（len 是分段函数的段数，与区间长度同阶）的时间复杂度内完成。这样建立线段树的时间复杂度为线段树中所有节点对应的区间长度之和，为 $O(B \log B)$。

但是对于原问题，我们这么做会有一些问题：第一个问题是执行修改操作时，需要对不完全覆盖的块进行重构。如果暴力重建整个线段树，则时间复杂度为 $O(B \log B)$ ，不够优秀。

注意到对一个块进行重构时，对这个块进行的修改仅仅是区间加，根据线段树的性质，在每一层中分别只有 $O(1)$ 个节点会被不完全覆盖，而被完全覆盖的节点所进行的修改仅仅是对分段函数进行平移，可以用懒标记维护。这样每一层中分别只有 $O(1)$ 个节点的信息需要重新计算，时间复杂度为 $O(B + \frac{B}{2} + \frac{B}{4} + \frac{B}{8} + \cdots) = O(B)$ 。

另外一个问题是查询一个块的答案时，需要通过二分查找求出答案在分段函数的哪一段。如果直接进行二分查找，则单次操作的复杂度为 $O(B + \frac{n \log B}{B})$，不够优秀。官方题解给出的做法是类似 2.3 节中的方法进行优化，即基数排序然后利用单调性。时间复杂度为 $O(n\sqrt{n})$ 。由于进行了逐块处理，可以只在处理该块时建立该块的线段树，所以空间复杂度为 $O(B \log B + n) = O(n)$，而不是将每一块的线段树全部建立出来时的 $O(n \log n)$ 。

事实上，我们可以使用 3.2 节中算法 4 分散层叠法，其中叶节点所存储的序列为分段函数的分界点构成的序列。对于被完全覆盖的块，修改操作进行的修改仅仅是对分段函数进行平移，即将分界点全部加上同一个数。所以如果线段树中的节点对应的区间被完全覆盖，

这个节点的子树中所有节点维护的序列也只会被全部加上一个数，不会改变结构。所以 3.2 节中做法仍然适用，时间复杂度为 $O(n\sqrt{n})$，空间复杂度为 $O(n\log n)$。

4 总结

分散层叠算法虽然已经有 20 多年的历史，是一种常见的算法，但目前在信息学竞赛中应用还不广泛。本文介绍了分散层叠算法，并将该算法和其他算法进行了对比，应用于信息学竞赛中的一些题目，特别是在解决这类经典分块问题中进行了算法优化。希望对大家有所帮助。

但这一领域研究者不多，限于笔者水平有限，很多问题尚未解决，希望本文能抛砖引玉，吸引更多的同仁来研究这个算法。

致谢

感谢中国计算机学会提供学习和交流的平台。
感谢国家集训队高闻远教练的指导。
感谢蔺洋老师、张君亮老师、胡凡老师、李植武老师、林鸿老师对我的培养与教导。
感谢蔡承泽学长、李欣隆学长等在讨论中给予我的灵感。
感谢王修涵、王思齐、袁方舟等学长的指导。
感谢成都七中高 18 级信息组全体同学两年来的陪伴。
感谢父母对我的理解与支持。

参考文献

[1] Chazelle, Bernard;Guibas, Leonidas J. Fractional cascading: I. A data structuring technique. 1986.

[2] Chazelle, Bernard;Guibas, Leonidas J. Fractional cascading: II. Applications. 1986.

[3] Wikipedia contributors. Fractional cascading. Wikipedia. (https://en.wikipedia.org/wiki/Fractional_cascading)

[4] 白马负金羁 CSDN 博客 (https://blog.csdn.net/baimafujinji/article/details/52956605).

浅谈支配树及其应用

南京外国语学校 陈孙立

摘 要

本文主要介绍了支配树的求解算法以及它的应用。
本文第二节介绍了一些记号和定义。
本文第三节着重讲解了支配树的求解算法。
本文第四节介绍了支配树的一些应用。
本文第五节是总结。

1 引言

支配树这个概念在图论和计算机设计理论中经常出现，作为一个算法也时有出现在各类算法竞赛中，但是系统介绍它的中文资料却相对较少。笔者希望通过本文，能让更多的同学了解支配树，以及它的一些应用。

2 定义

本文中我们考察一个 n 个点 m 条边的**有向图** $G=(V,E)$，其中 V 是节点集，E 是边集，且 $|V|=n,|E|=m$。本文假设读者已经熟悉图论中路径、环、深度优先搜索 (DFS)、拓扑序、DFS 序等基础概念。

在给定一个源 $s\in V$ 的前提下，如果对于两点 x,y 满足任意以 s 为起点，y 为终点的路径都经过 x，则称 x 支配 (dominates) y。由于当不存在 s 到 x 的路径时，和 x 有关的支配关系都没有意义，所以**在本文中，我们始终假设从 s 能到达所有其他点**，对于一般的图只需先以 s 为起点进行深度优先遍历，并只保留被访问的点即可。

首先观察到，我们可以仅考虑简单路径，这是因为如果路径出现了重复节点，可以把重复部分之间的节点删去，这样的修改对支配条件不会产生影响。一个显然的性质是 s 支配所有点，除此之外，支配关系还满足以下性质，从而可以用一个树结构完全描述它们。

性质 1：如果 x 支配 y 且 y 支配 z 则 x 支配 z，也即支配关系满足**传递性**。

这个性质比较显然。

性质 2：如果 x 支配 y 且 y 支配 x，则 $x = y$。

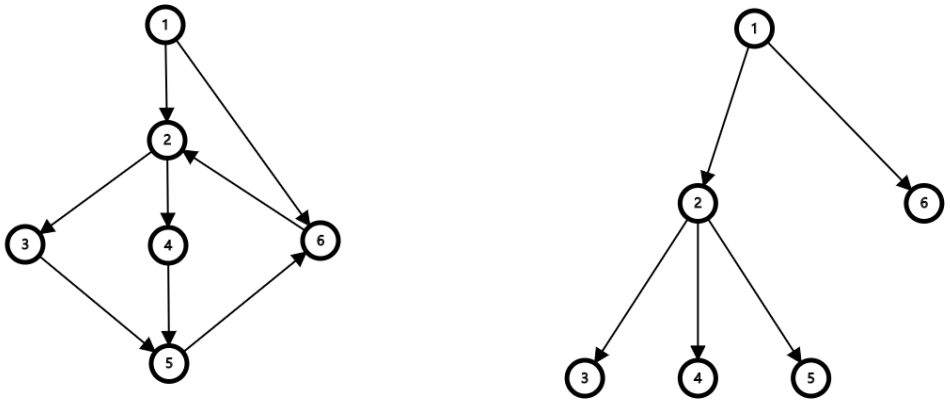
证明：假设 $x \neq y$，可知任意一条从 s 到 x 的简单路径都经过 y，且在经过 y 之前必须经过 x，于是这条路径形如 $s, \ldots, x, \ldots, y, \ldots, x$，和简单路径矛盾。由此导出必有 $x = y$。 □

性质 3：如果 x, y, z 互不相等，且 x 和 y 均支配 z，则必有 x 支配 y 或 y 支配 x。

证明：考察任意一条从 s 到 z 的简单路径，这条路径上必有 x 和 y，不失一般性可以假设路径形如 $s, \ldots, x, \ldots, y, \ldots, z$。如果 x 不支配 y，则存在一条不经过 x 的 s 到 y 的路径，此时若把前述路径从 s 到 y 的部分替换成这条不经过 x 的路径，就得到了一条从 s 到 z 且不经过 x 的路径了，而这和 x 支配 z 矛盾。 □

上述性质表明，对于某点 $x \neq s$，考察点集 $S = \{y \mid y \text{ 支配 } x\}$，则 S 中的点依支配关系构成全序集，且 S 中有至少两个元素。由此，一定存在一个点 z 满足：如果 $y \neq x$ 支配 x，则 y 也支配 z。我们称 z 是 x 的直接支配点 (immediate dominator)，记为 $z = idom(x)$。

现对于所有除 s 之外的点 x 连一条 $idom(x) \to x$ 的边，这样得到的图每个点的入度均不超过 1；结合**性质 1** 和**性质 2** 可知，这个图是无环的，于是这样的图一定是一个以 s 为根的有向树，称为支配树 (Dominator Tree)，记为 $DT(G, s)$。它满足：对于两点 x, y，x 支配 y 当且仅当在支配树上 x 是 y 的祖先（本文中的祖先等于自身），所以只要得到了支配树，我们就得到了所有的支配关系。



上面右图是左图在 $s = 1$ 时的支配树。

3 支配树的求解算法

3.1 有向无环图的特殊情况

对于有向无环图 $G = (V, E)$，由于拓扑序在后的点不会影响前面点的支配关系，可以按照拓扑序依次求解每个点 x 的直接支配点。现假设所有拓扑序在 x 之前的点的直接支配

点都已求出，此时可以得到一个不完整的支配树，它一定是最终得到的支配树的一部分，这里姑且叫做“当前支配树”。设以 x 为终点的边有 $(c_1,x),\ldots,(c_k,x)$ ，则 $idom(c_i)$ 均已求出。

引理 1：点 y 支配 x 当且仅当 $y=x$ 或 y 支配所有 $c_1,\ldots,c_k$ 。

证明：$y=x$ 时结论显然，下面假设 $y\neq x$ 。先证当 y 支配所有 $c_1,\ldots,c_k$ 时，y 必支配 x ：考察任意一条从 s 到 x 的路径，则在 x 之前的点一定是 $c_1,\ldots,c_k$ 之一，于是这条路径必须经过点 y 。

再证上面的否命题：不失一般性假设 y 不支配 c_1 ，则存在从 s 到 c_1 的路径不经过 y ，此时可以沿这条路径走到 x ，就得到了一条从 s 到 x 的、不经过 y 的路径，于是 y 不支配 x 。 □

满足上面性质的点就是 $c_1,\ldots,c_k$ 在“当前支配树”上的公共祖先，于是 $idom(x)=LCA(idom(c_1),\ldots,idom(c_k))$ （LCA 指最近公共祖先），这相当于在“当前支配树”上添加了一个叶子和一条边。如果使用倍增求解 LCA 的算法，只需稍加修改就能支持每次添加一个叶子了，这个做法的时间和空间复杂度都是 $O((n+m)\log n)$ 。

3.2 半支配点的定义和性质

在之后的算法中我们要考虑任意一个图 G 的从 s 出发的 DFS 树，即进行深度优先遍历时经过的边形成的树结构，同时按照遍历顺序（DFS 序）为点赋予大小。具体来说对于两点 x,y ，如果 x 在遍历时比 y 访问的时间更早，则称 $x<y$ ，类似地也可定义 $x>y$ ，以及点集的最大值和最小值等相关概念。有向图的 DFS 树不存在边 (x,y) 满足 $x<y$ 且 x 不是 y 的祖先。一个更有用的表述是：

定理 1：如果 $x\le y$ 则任意 x 到 y 的路径必须经过 x 和 y 在 DFS 树上的某个公共祖先。

证明：x 是 y 的祖先时结论显然，因此可认为 x 不是 y 的祖先。假设存在一条路径 $x=v_0,v_1,\ldots,v_k=y$ ，其中不存在 x 和 y 在 DFS 树上的公共祖先。令 z 是 x 和 y 在 DFS 树上的最近公共祖先，z 的孩子中必有唯一一点 y' 是 y 的祖先。

考察任意一条路径 $x=v_0,v_1,\ldots,v_k=y$ ，令 $p=\max\{i\mid v_i<y'\}$ ，并假设 v_p 不是 z 的祖先，那么 v_p 也一定不是 v_{p+1} 的祖先，这是由 DFS 序的性质得到的：若 x 是 y 的祖先，则 x 也是 $z(x\le z\le y)$ 的祖先。由此可得存在边 (v_p,v_{p+1}) 且 $v_p<v_{p+1}$ ，v_p 不是 v_{p+1} 的祖先。然而，这条边不可能在 DFS 树中存在，否则在遍历到 v_p 时会直接沿着这条边遍历 v_{p+1} 。这个矛盾表明 v_p 一定是 z 的祖先。 □

下面给出半支配点 (semi-dominator) 的定义。

定义：点 x 的半支配点 $sdom(x)=\min\{y\mid$ 存在一条路径 $y=v_0,v_1,\ldots,v_k=x$ 满足对于任意 $1\le i\le k-1$ 都有 $v_i>x\}$ 。

称定义中路径的存在性为“半支配点条件”。半支配点的求解是直接支配点求解的中间步骤。以下是一些关于直接支配点和半支配点的重要性质。

引理 2：$idom(x)$ 和 $sdom(x)$ 都是 DFS 树上 x 的祖先且不等于 x。更进一步地，在 DFS 树上 $idom(x)$ 是 $sdom(x)$ 的祖先。

证明：$idom(x)$ 在 DFS 树上是 x 的祖先是显然的，因为 DFS 树上有一条 s 到 x 的路径，$idom(x)$ 必定在这条路径上。

由于 x 在 DFS 树上的父亲已经满足了半支配点的要求，所以有 $sdom(x) < x$ 。另一方面，如果 $sdom(x)$ 不是 x 的祖先，则不难发现 $sdom(x)$ 在 DFS 树上的父亲也满足半支配点的要求，且比 $sdom(x)$ 更小，这和半支配点的定义矛盾。

根据半支配点的定义，可以沿 DFS 树从 s 走到 $sdom(x)$ 再沿定义所述的路径走到 x 。因此 DFS 树中，$sdom(x)$ 到 x 的路径上（这里不包括 $sdom(x)$）的点均不支配 x ，故 $idom(x)$ 一定是 $sdom(x)$ 的祖先。 □

引理 3：令 v, w 满足 DFS 树上 v 是 w 的祖先，则 v 是 $idom(w)$ 的祖先或 $idom(w)$ 是 $idom(v)$ 的祖先。

证明：若 $v = w$ 结论显然，因此假设 $v \neq w$ 。使用反证法，由于 $v, w, idom(v), idom(w)$ 都是 w 的祖先，如果引理中的结论不成立，则它们按深度排序依次是 $idom(v), idom(w), v, w$ 且它们互不相等。这意味着存在一条不经过 $idom(w)$ 的从 s 到 v 的路径，继续沿 DFS 树走到 w 就得到一条不经过 $idom(w)$ 的从 s 到 w 的路径，矛盾。 □

有了上述性质，我们将展示半支配点和直接支配点的紧密联系，从而可以通过前者计算后者。

定理 2：任取点 $w \neq s$ 。考虑 DFS 树上从 $sdom(w)$ 到 w 的路径上（不包括 $sdom(w)$）的任意点 x，如果均满足 $sdom(x) \geq sdom(w)$ ，则 $idom(w) = sdom(w)$ 。

证明：根据**引理 2**，有 $idom(w)$ 是 $sdom(w)$ 的祖先，因此只需证 $sdom(w)$ 支配 w 。考虑任意一条 s 到 w 的路径 P ，我们将要证明 $sdom(w)$ 一定在 P 中。令 x 是 P 中**最后一个**满足 $x < sdom(w)$ 的点，如果不存在则必有 $sdom(w) = idom(w) = s$ ，结论仍然成立。同时，令 y 是 P 中 x 之后的第一个在 DFS 树中从 $sdom(w)$ 到 w 的路径上的点。在 P 中截取 x 到 y 之间的部分得 $v_0 = x, v_1, \ldots, v_k = y$。

现在考虑路径 $v_0, \ldots, v_k$,我们声称对于 $1 \leq i < k$ 都有 $v_i > y$,也即 $sdom(y) \leq x < sdom(x)$ 。若不满足这一点，则有某个 $v_i < y$ ，此时根据**定理 1** 可得存在某个 $i \leq j \leq k-1$ 满足 v_j 是 y 的祖先。由 x 的取值可知 $sdom(w) \leq v_j$ ，于是 v_j 也在 DFS 树中从 $sdom(w)$ 到 w 的路径上，这和 y 的取值矛盾。上述矛盾可推得 $sdom(y) \leq x < sdom(x)$ ，结合定理的条件必有 $y = sdom(w)$ ，即路径 P 包含 $sdom(w)$。 □

定理 3：任取点 $w \neq s$ 。考虑 DFS 树上从 $sdom(w)$ 到 w 的路径上（不包括 $sdom(w)$）的所有点，令 u 是其中 $sdom$ 最小的点，则 $sdom(u) \leq sdom(w)$ 且 $idom(u) = idom(w)$ 。

证明：由于 w 也是 u 的一个候选，$sdom(u) \leq sdom(w)$ 是显然的。注意到在 DFS 树上 $idom(w)$ 是 u 的祖先，于是根据**引理 3** 可知 $idom(w)$ 也是 $idom(u)$ 的祖先，因此只需证 $idom(u)$ 支配 w 。考虑任意一条 s 到 w 的路径 P ，我们将要证明 $idom(u)$ 一定在 P 中。令 x 是 P 中**最后一个**满足 $x < idom(u)$ 的点，如果不存在则必有 $idom(u) = idom(w) = s$ ，结论仍然成立。同时，令 y 是 P 中 x 之后的第一个在 DFS 树中从 $idom(u)$ 到 w 的路径上的点。在 P 中截取 x 到 y 之间的部分得 $v_0 = x, v_1, \ldots, v_k = y$。

和**定理 2** 的证明类似，我们可以得到 $sdom(y) \le x$ 。根据**引理 2** 有 $idom(u) \le sdom(u)$ ，综合起来可得 $sdom(y) \le x < idom(u) \le sdom(u)$ 。至此，由 u 的选择可知 y 不能是 $sdom(w)$ 的除自身外的后代；另一方面，y 不能既是 $idom(u)$ 的除自身外后代也是 u 的祖先，否则沿 DFS 树从 s 走到 $sdom(y)$ 再沿 P 走到 y，最后沿 DFS 树走到 u 的这条路径就不经过 $idom(u)$ 了，这和支配点的定义矛盾。

根据 $idom(u) \le sdom(w) < u \le w$ 和 $idom(u) < y$ ，唯一的情况就只剩下 $y = idom(u)$ 了，于是路径 P 包含 $idom(u)$ 。 □

上面两个定理说明了半支配点和直接支配点的关系，也给出了由前者求出后者的方法。对于点 $w \ne s$ 来说，考虑 DFS 树上从 $sdom(w)$ 到 w 的路径上（不包括 $sdom(w)$）的所有点，令 u 是其中 $sdom$ 最小的点，若 $sdom(u) = sdom(w)$ 则可利用**定理 2** 得到 $idom(w) = sdom(w)$ ；否则 $u \ne w$ ，可利用**定理 3** 得到 $idom(w) = idom(u)$ 。

3.3 计算半支配点和直接支配点

3.3.1 半支配点

为了求出半支配点，我们需要下面的定理。

定理 4：$sdom(w)$ 由以下两种情况产生的候选取最小值得到。

- 1. 有边 (v, w) ，此时 v 是 $sdom(w)$ 的候选。
- 2. u 是 v 在 DFS 树上的祖先，$u > w$ 且有边 (v, w) ，此时 $sdom(u)$ 是 $sdom(w)$ 的候选。

证明：令 x 是所有候选的最小值。我们首先证明每个候选都满足半支配点的条件，从而得到 $sdom(w) \le x$ 。若 x 由情况 1 得到，则显然 (x, w) 就是满足半支配点条件的路径；否则 $x = sdom(u)$ ，此时取出 $sdom(u)$ 到 u 的半支配点条件中所述的路径，再接上 DFS 树中 u 到 v 的路径，最后从 v 到 w 就是满足 w 的半支配点条件的路径。

还需证明 $sdom(w) \ge x$ 。为此考虑 $sdom(w)$ 到 w 的半支配点条件对应的路径 $v_0 = sdom(w), v_1, \ldots, v_k = w$ ，其中对于任意 $v_i (1 \le i < k)$ 都有 $v_i \ge w$。若 $k = 1$ 则它在情况 1 中被考虑到，下设 $k > 1$ 。取 j 是最小的满足 $j \le 1$ 且 v_j 是 v_{k-1} 在 DFS 树上的祖先的数，这样的 j 一定存在，因为 k 就是一个合法的数。

现在考虑路径 $v_0, \ldots, v_j$ ，我们声称它是满足 v_j 的半支配点条件的路径，即对于任意 $1 \le i < j$ 都有 $v_i > v_j$ 。若不是，则考虑所有 $v_i \le v_j$ 的 i ，取其中满足 v_i 最小的 i ，可由**定理 1** 得到 v_i 一定是 v_j 的祖先，这和 j 的选取矛盾。于是 $sdom(v_j) \le sdom(w)$ ，而 $sdom(v_j)$ 必定在情况 2 中被考虑到，因此 $x \le sdom(w)$ 。 □

有了**定理 4**，可以设计如下求出半支配点的算法：

- 1) 求出图 G 的一个 DFS 树和 DFS 序。

- 2) 若当前所有点的 *sdom* 都已经求出，则结束算法；否则找到 $sdom(w)$ 未求出的点中最大的 w 。
- 3) 考虑情况 1，可以直接计算所有的候选。
- 4) 考虑情况 2，枚举边 (v,w) 且 $v > w$ ，此时 v 一定被标记。从 v 开始不断地在 DFS 树上向被标记的父亲走形成一条路径，并求出这条路径上 *sdom* 的最小值作为候选。
- 5) 合并情况 1 和情况 2 求出的所有候选，得到 $sdom(w)$ 的值，标记 w 为已求出 *sdom* ，并回到 2)。

可以看到，从大到小考虑所有节点正是为了步骤 4) 中所用到的 *sdom* 值均已求出。容易发现除步骤 4) 外的其他部分时间复杂度均不超过 $O(n+m)$ 。为了优化复杂度，在步骤 4) 中，选取带权并查集这个数据结构维护路径的 *sdom* 最小值和取到最小值的位置，在标记 x 时，把 DFS 树上 x 的所有孩子向 x 连边并重新维护最小值即可。朴素地使用路径压缩的并查集实现，时间复杂度可以做到 $O((n+m)\log n)$ ，空间复杂度为 $O(n+m)$ 。针对此问题还存在更快的并查集实现，复杂度可以做到更低的 $O((n+m)\alpha(n))$ 甚至线性，有兴趣的读者可以参考相关资料。

3.3.2 直接支配点

为求出所有点的 *idom* ，定义 $pivot(w)$ 是 DFS 树中 $sdom(w)$ 到 w 的路径上（不包括 $sdom(w)$）*sdom* 值最小的点。若求出了所有的 $sdom(w)$ 和 $pivot(w)$ ，就可以**从小到大**根据**引理 2** 和**引理 3** 依次确定每个点的 *idom* ，具体来说：若 $sdom(pivot(w)) = sdom(w)$ 则 $idom(w) = sdom(w)$ ；否则 $idom(w) = idom(pivot(w))$ ，这里由于 $pivot(w) \le w$ 在考虑到点 w 之前一定已经求出 $idom(pivot(w))$ 。

最后，可以在之前的求半支配点的算法上略加修改得到每个点的 *pivot* 。我们为每个点 x 开辟一个通 $bucket(x)$ 储存 $sdom(w) = x$ 的所有点 w 。若当前正在求解点 w 的 *sdom* ，则在进行步骤 5) 的标记，即并查集的 link 操作之前，考虑所有 $bucket(w)$ 中的点 x ，并直接在并查集中查询得到 $pivot(x)$ 的值。在求出 $sdom(w)$ 之后，也别忘记还要把 w 加入 $bucket(sdom(w))$ 中。

综合之前所述，我们已经得到了完整的支配树算法。

3.4 算法总结

下面把之前的算法汇总并简要概括如下：

- 1) 求出图 G 的一个 DFS 树和 DFS 序。

- 2) 若当前所有点的 $sdom$ 都已经求出，则进行 8)；否则找到 $sdom(w)$ 未求出的点中最大的 w 。
- 3) 考虑情况 1，可以直接计算所有的候选。
- 4) 考虑情况 2，枚举边 (v,w) 且 $v > w$，此时 v 一定被标记。从 v 开始不断地在 DFS 树上向被标记的父亲走形成一条路径，并求出这条路径上 $sdom$ 的最小值作为候选，这条路径就是当前并查集中从点 v 到点 v 的根的路径，可以直接在并查集中查询。
- 5) 合并情况 1 和情况 2 求出的所有候选，得到 $sdom(w)$ 的值，并把 w 加入 $bucket(sdom(w))$。
- 6) 对每个 $bucket(w)$ 中的数 x 求出 $pivot(x)$，它也可以直接在并查集中查询从 x 到 x 的根的路径最小值得到。
- 7) 对 DFS 树上 w 的每个孩子 c 执行：在并查集中设置 c 的父亲为 w，并维护路径最小值信息。
- 8) 从小到大依次考虑每个点 w，若 $sdom(pivot(w)) = sdom(w)$ 则 $idom(w) = sdom(w)$；否则 $idom(w) = idom(pivot(w))$ 。

这个算法空间复杂度为 $O(n+m)$，时间复杂度的瓶颈在维护并查集上，如果使用朴素的路径压缩，复杂度为 $O((n+m)\log n)$，在信息学竞赛范围内这个复杂度已经足够优秀。

4 支配树的应用

4.1 某个点集的支配点

点集 S 的支配点定义为 s 到任意一个 $x \in S$ 都必须经过的点。不难发现这样的点就是 S 中所有支配点的交集，而点 x 的支配点就是支配树中 x 的所有祖先，因此点集 S 的支配点就是所有 $x \in S$ 在支配树上的最近公共祖先（LCA）。

4.2 支配边

和支配点的定义类似，若任意从 s 到 x 的路径都经过边 (u,v) 则称 (u,v) 支配 x 。同时也可以类似地定义直接支配边：若 (u,v) 支配 x 且 x 的所有支配边也支配 u 则称 (u,v) 是 x 的直接支配边。

为了求出每个点的支配边，一个简单的做法是对原图的每条边 (u,v) 建立辅助点 w，并连边 (u,w) 和 (w,v)，并求出新图的支配树。对于一个原图的节点 x，它的支配边就是新图

的支配树中 x 的是辅助点的祖先，直接支配边就是其中深度最大的一个，可以简单地使用 DFS 求出，不过新图的点数是 $n+m$ 边数是 $2m$，这可能会带来较大的常数。仔细审视这个做法可以发现，辅助点在运行算法时会有诸多性质，从而不一定要在新图运行完整的算法就可能得出支配边。

引理 4：边 (u,v) 是点 x 的支配边当且仅当 u 和 v 都是 x 的支配点且 u 到 v 仅有一条简单路径（即边 (u,v) 是 v 的支配边）。

证明：若边 (u,v) 是点 x 的支配边，则任意 s 到 x 的路径都经过 u 和 v，即 u 和 v 支配 x；更进一步地，如果 u 到 v 有边 (u,v) 之外的路径，可沿任意一条从 s 到 u 的路径再到 v 再到 x 且不经过 (u,v)，因此 u 到 v 必须有且仅有一条简单路径。

反过来，若引理的条件成立，则显然边 (u,v) 将支配点 x。 □

有了**引理 4**，对于一条 DFS 树上的边 (u,v)，若 u 支配 v 且 u 到 v 仅有一条简单路径，则 (u,v) 支配所有 v 支配的点。回顾支配树算法，若边 (u,v) 在 DFS 树上，且 $idom(v)=u$，则必有 $sdom(v)=u$。

现在把注意力放在 u 到 v 仅有一条简单路径这个条件上，可以通过之前建立辅助点的方法理解。假设仅对这条边新建了辅助点 w，则 $sdom(w)=u$ 且 $sdom(v)\geq u$，此时 $idom(v)=w$ 当且仅当 $sdom(v)=w$。回到原图则是：在计算 $sdom(v)$ 时不考虑 (u,v) 这条边，即只考虑情形 2 时没有任何候选。这样，只要在求 $sdom$ 时加入一个判断，就能确定 u 到 v 是否只有一条简单路径了。

最后，我们求出了所有边 (u,v) 满足它支配 v，再在 DFS 树上进行一遍遍历即可确定每个点的直接支配边。这个做法不需要建立辅助点，常数更小且代码难度没有明显增加。

4.3 有向图的割点和桥

有向图 G 中，u 和 v 强连通指 u 能到达 v 且 v 也能到达 u。强连通关系是等价关系，它形成的等价类称为强连通分量，这和无向图中的连通分量对应。对图 G 来说，记 $G\backslash\{u\}$ 为删去点 u 和相关的边后得到的新图；$G\backslash\{(u,v)\}$ 为删去边 (u,v) 后得到的新图。

无向图的割点和桥指删去之后连通分量个数增加的点和边。与之对应地，有向图的强割点和强桥指删去之后强连通分量个数增加的点和边。一个自然的问题是如何求出某个图 G 中所有的强割点和强桥。和无向图一样，求强割点和强桥时可以对每个强连通分量分开考虑，因此**以下讨论均对强连通图进行**。

引理 5.1：点 v 是强割点当且仅当对于任意 $x\neq v$ 都存在一个点 $y\neq v$，使得以下之一成立：

- 1. 任意 x 到 y 的路径都经过 v
- 2. 任意 y 到 x 的路径都经过 v

引理 5.2：边 (u,v) 是强桥当且仅当对于任意 x 都存在一个点 y，使得以下之一成立：

- 1. 任意 x 到 y 的路径都经过边 (u,v)
- 2. 任意 y 到 x 的路径都经过边 (u,v)

证明：这两个引理的证明基本一致，这里仅证明**引理 5.1**。若 v 是强割点，则图 $G\backslash\{u\}$ 不是强连通图，那么考察与 x 不在同一强连通分量里的某点 y，要么不存在 x 到 y 的路径，要么不存在 y 到 x 的路径。由于 G 是强连通的，可得引理的条件成立；另一方面，若引理的条件成立则 x 和 y 不再强连通。 □

若使用**引理 5.1**，任选一点 v 即可求出所有除 v 之外的强割点。对于条件 1，只要求出 $DT(G,v)$，则支配树中所有除 v 之外的非叶子节点都是强割点；对于条件 2，只要构建出 G 的反图 G^R 并求出 $DT(G^R,v)$ 即可。最终再使用普通的求强连通分量算法即可判断点 v 是否是强割点。

对于强桥有类似的结论，可套用上一节所述的支配边算法解决。上述求强割点和强桥的算法时间复杂度都和求支配树的复杂度相同。

4.4 工业方面的应用

支配树在许多需要基于有向图的结构上有着很多用处。

神经网络可以抽象成一个有向图，每个节点的计算任务依赖于有边指向它的节点的结果，为了加速神经网络的运行，通常采用并行计算的方式优化节点的计算顺序，此时支配树可以部分地求出那些被高频使用的节点并优先计算它们。

除此之外，支配树还能在编译原理、内存泄漏检测中发挥作用。总的来说，“支配”的概念在很多领域都有类似的说法，因此支配树作为求解支配关系的工具应用广泛。

5 总结

本文介绍了支配树的求解算法和一些应用。虽然支配树问题已经被解决，但从它的应用来看，支配树的变种很多，也有许多值得研究之处。

希望能通过本文让更多的同学了解支配树这一算法和工具，并更深入地探究相关问题。

感谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练高闻远的指导。

感谢南京外国语学校李曙老师的指导和支持。

感谢所有曾经给予我帮助的老师和同学。

感谢我的父母一直以来的全力支持和鼓励。

参考文献

[1] Thomas Lengauer, Robert Endre Tarjan, A Fast Algorithm for Finding Dominators in a Flowgraph.

[2] Adam L. Buchsbaum, Haim Kaplan, Anne Rogers, Jeffery R. Westbrook, A new, simpler linear-time dominators algorithm.

[3] Giuseppe F. Italianoa, Luigi Laura b, Federico Santaroni, Finding strong bridges and strong articulation points in linear time.

[4] Robert E. Tarjan, Depth-first search and linear graph algorithms.

《拼数》命题报告

杭州学军中学 姜迅驰

摘 要

本文介绍了作者给校内训练命制的一道题《拼数》。解题过程从一个简单的算法开始，不断观察性质，优化复杂度，最后得到了一个巧妙的解法。

1 题目描述

1.1 题目大意

定义 $f(i)$ 将 i 的二进制形式写下所构成的字符串，例如 $f(3)=$“11”、$f(6)=$“110”。

你需要恰当地选择一个 1 到 n 的排列 P，使得字符串 $S=f(P_1)+f(P_2)+\cdots+f(P_n)$ 的字典序尽可能小。

输出 S 的前 k 位中的 1 的数量。

由于一些原因，输入和输出采用二进制形式。

1.2 数据规模

对于所有数据，$1\le n<2^{2000}$，$1\le k\le |S|$。

- 子任务 1（7 分）：$1\le n\le 8$；
- 子任务 2（16 分）：$1\le n\le 10^5$；
- 子任务 3（26 分）：$1\le n\le 2^{60}$；
- 子任务 4（23 分）：$1\le n\le 2^{300}$；
- 子任务 5（28 分）：无特殊限制。

时间限制：1s

空间限制：512MB

1.3 输入格式

从标准输入读入数据。

第一行一个不存在前导零的 01 串，以二进制形式给出了题目中的 n。

第二行一个不存在前导零的 01 串，以二进制形式给出了题目中的 k。

1.4 输入格式

输出到标准输出中。

共一行，一个不存在前导零的 01 串，以二进制形式给出题目的答案。

1.5 样例输入

1011

10010

1.6 样例输出

1000

1.7 样例说明

根据题意，一种可能的排列为（以下所有数均以二进制形式给出）：

1000,100,1001,10,1010,101,1011,110,1,11,111

将这些二进制首尾相接得到字符串 S，容易发现答案为 8。

2 记号与约定

对于一个字符串 S，定义 $S[i]$ 为 S 的第 i 个字符。

定义 $S[l\ldots r]$ 为 S 的第 l 个字符到第 r 个字符所构成的字符串，特殊地，若 $l > r$，则 $S[l\ldots r] = \emptyset$。

定义 $|S|$ 为 S 的长度。

定义 $h(S)$ 为将一个 01 字符串 S 从左到右写下后构成的二进制数。

定义 $|f(n)| = N$。

定义 $x \times S$ 为将一个字符串 S 重复 x 遍之后得到的字符串。

定义 S^{∞} 为将 S 无限重复之后得到的一个无限长的字符串。

定义字典序为：若 $|X| < |Y|$ 且 $X = Y[1 \dots |X|]$，或存在一个 $i \le |X|$ 满足 $X[1 \dots i-1] = Y[1 \dots i-1]$ 且 $X[i] < Y[i]$，则 X 的字典序小于 Y。

3 初步分析

3.1 算法一

枚举所有的合法排列 P，求出其生成的字符串中字典序最小的。

复杂度 $\mathrm{O}(n! \times n \log n)$。

期望得分：7 分。

3.2 算法二

引理 1. *对于两个字符串 X 和 Y，若 $X^\infty = Y^\infty$，则 $X + Y = Y + X$。*

证明. 找到 X^∞ 的最小循环节 s，则可得

$$X = \frac{|X|}{|s|} \times s,$$

$$Y = \frac{|Y|}{|s|} \times s.$$

则

$$X + Y = Y + X = \frac{|X| + |Y|}{|s|} \times s.$$

□

引理 2. *对于两个字符串 X 和 Y，若选择一个 $1 \le k \le |X| + |Y|$，满足*

$$X^\infty[1 \dots k-1] = Y^\infty[1 \dots k-1],$$

则

$$(X + Y)[k] = X^\infty[k],$$

$$(Y + X)[k] = Y^\infty[k].$$

证明. 不妨设 $|X| < |Y|$。

若 $k \le |X|$，则

$$(X + Y)[k] = X[k] = (X^\infty)[k],$$

$$(Y + X)[k] = Y[k] = (Y^\infty)[k],$$

若 $|X| < k \le |Y|$，则

$$(X+Y)[k] = Y[k-|X|] = (X^\infty)[k-|X|] = (X^\infty)[k],$$

$$(Y+X)[k] = Y[k] = (Y^\infty)[k],$$

若 $|Y| < k$，则

$$(X+Y)[k] = Y[k-|X|] = (X^\infty)[k-|X|] = (X^\infty)[k],$$

$$(Y+X)[k] = X[k-|Y|] = (Y^\infty)[k-|Y|] = (Y^\infty)[k].$$

□

引理 3. 对于两个字符串 X 和 Y，若 $X^\infty < Y^\infty$，则 $X+Y < Y+X$。

证明. 设 X^∞ 和 Y^∞ 的第一个不同的位置为 k，由引理二可得，

$$(X+Y)[1\ldots k] = (X^\infty)[1\ldots k],$$

$$(Y+X)[1\ldots k] = (Y^\infty)[1\ldots k].$$

可得

$$(X+Y)[1\ldots k] < (Y+X)[1\ldots k],$$

即 $X+Y < Y+X$。

□

定理 1. 给出 n 个字符串 $A_1, A_2, \ldots, A_n$，所有使得 $A_{P_1} + A_{P_2} + \cdots + A_{P_n}$ 的字典序最小的排列 P 均满足以下条件：对于任意 $1 \le i < n$，$(A_{P_i})^\infty \le (A_{P_{i+1}})^\infty$。

证明. 若存在一个 i 满足 $(A_{P_i})^\infty > (A_{P_{i+1}})^\infty$，根据引理三可得

$$A_{P_i} + A_{P_{i+1}} > A_{P_{i+1}} + A_{P_i}.$$

则交换 P_i 与 P_{i+1} 更优。

而对于一对 $(A_{P_i})^\infty = (A_{P_{i+1}})^\infty$，根据引理一可得

$$A_{P_i} + A_{P_{i+1}} = A_{P_{i+1}} + A_{P_i}.$$

则交换 $(A_{P_i})^\infty$ 和 $(A_{P_{i+1}})^\infty$ 后并不会对得到的字符串造成影响，即 $(A_{P_i})^\infty$ 相等的字符串以任意顺序排列答案均为最优。

□

对于 $f(1), f(2), \ldots, f(n)$，将它们以 $f(i)^\infty$ 从小到大排序，再将它们拼接起来即可得到字符串 S。

复杂度 $\mathrm{O}(n \log^2 n)$。

期望得分：23 分。

4 解法分析

将 1 到 n 的所有数以 $f(i)^\infty$ 从小到大排序，且在 $f(i)^\infty$ 相等的情况按照 i 从小到大排序，这样可以唯一确定一个排列 P。

定义 $Q = P^{-1}$，即一个排列满足 $Q_{P_i} = i$。

定义 c 为满足

$$\sum_{i=1}^{Q_c-1} |f(P_i)| \le k,$$

$$\sum_{i=1}^{Q_c} |f(P_i)| > k$$

的唯一整数。可以发现 $f(c)$ 在 $S[1\dots k]$ 中只会保留一个前缀，而 $f(P_1), f(P_2), \dots, f(P_{Q_c-1})$ 在 $S[1\dots k]$ 中完整出现。

参考算法分为两个部分，第一部分为获得 c 的值，第二部分为通过 c 的值得到求出答案。

由于第二部分较为简单，此处会优先介绍第二部分。

4.1 第二部分

4.1.1 预处理

引理 4. *若 $i < j$，且 $|f(i)| = |f(j)|$，则 $Q_i < Q_j$。*

证明. 若 $i < j$、$|f(i)| = |f(j)|$，则 $f(i)^\infty < f(j)^\infty$，即 i 的位置在 j 的位置之前。可得 $Q_i < Q_j$。 □

由引理四可得，同一个长度的所有数在 P 中是从小到大排列的。

定义

$$g(i, j) = \max\{k \mid |f(k)| = i, Q_k < Q_j\},$$

即排在 j 之前的最靠后的长度为 i 的数，之后会给出求出 $g(1, c), g(2, c), \dots, g(N, c)$ 的算法。

定理 2. *对于两个字符串 X 和 Y，若 $X^\infty \neq Y^\infty$，设 X^∞ 与 Y^∞ 的第一个不同的位置为 k，则 $k \le 2\max(|X|, |Y|)$。*

证明. 以下会证明若 X^∞ 与 Y^∞ 的前 $2\max(|X|, |Y|)$ 位相等，则 $X^\infty = Y^\infty$。

若 $\gcd(|X|, |Y|) = 1$，则对于 X^∞ 与 Y^∞ 的前 $|Y|$ 位，有 $X[(i-1) \mod |X|+1] = Y[i]$；对于 X^∞ 与 Y^∞ 的第 $|Y|+1$ 位到 $2|Y|$ 位，有 $X[(i+|Y|-1) \mod |X|+1] = Y[i]$。可得 $X[i] = X[(i+|Y|-1) \mod |X|+1]$。由于 $\gcd(|X|, |Y|) = 1$，可得 X 的所有字符均相等。由 $X[(i-1) \mod |X|+1] = Y[i]$ 可推出 Y 的所有字符同样全相等，可得 $X^\infty = Y^\infty$。

若 $\gcd(|X|,|Y|)=d$，将 X 和 Y 中所有 $\bmod d$ 相等的位单独取出考虑，可以采用类似于上述的方式证明它们均相等。这样同样可以说明 $X^\infty=Y^\infty$。

□

定义 $T=f(c)^\infty[1\dots 2N]$，则 $g(i,c)$ 可能为 $h(T[1\dots i])$ 或 $h(T[1\dots i])-1$。

将 $T[1\dots i]^\infty[1\dots 2N]$ 或 $f(h(T[1\dots i])-1)^\infty[1\dots 2N]$ 与 T 比较字典序后即可在 $\mathrm{O}(N)$ 的复杂度内得出答案。

注意对于 $i=N$，若求出的 $g(i,c)>n$，则需要将 $g(i,c)$ 赋值为 n。

求出所有 $g(i,c)$ 的复杂度为 $\mathrm{O}(N^2)$。

4.1.2 统计 $f(c)$ 的贡献

$f(c)$ 被截取的前缀长度为

$$k-\sum_{i=1}^{N} i\times(g(i,c)-2^{i-1}+1).$$

使用高精度计算以上式子即可得到 $f(c)$ 被截取的长度 l，求出 $f(c)[1\dots l]$ 中有几个 1 即可。

4.1.3 统计 $f(P_1),f(P_2),\dots,f(P_{Q_c-1})$ 的贡献

对于每个 $1\le i\le N$，需要将所有满足条件的长为 i 的数，即 $2^{i-1},2^{i-1}+1,\dots,g(i,c)$ 中 1 的个数统计入答案。

假设 $g(i,c)=\overline{a_1a_2\dots a_i}$，枚举所有的 $2\le j\le i$，若 $a_j=1$，统计

$$\left[\overline{a_1a_2\dots a_{j-1}0}\times 2^{i-j},\overline{a_1a_2\dots a_{j-1}1}\times 2^{i-j}\right)$$

中的数的贡献。这样共统计了 $\left[2^{i-1},g(i,c)\right)$ 中的数的贡献。

对于

$$\left[\overline{a_1a_2\dots a_{j-1}0}\times 2^{i-j},\overline{a_1a_2\dots a_{j-1}1}\times 2^{i-j}\right)$$

中所有数的贡献可以分成两部分进行处理：

对于前最高 j 位的贡献统计，由于这个区间中最高 j 位都是相同的，设最高 j 位共有 x 个 0，则答案需要加上 $x\times 2^{i-j}$。

对于剩下的 $i-j$ 位的贡献统计，可以看作 $[0,2^{i-j})$ 中 1 的个数。对于任意 $1\le k\le i-j$，第 k 位上有 1 的数的个数为 2^{i-j-1}。则答案需要加上 $(i-j)\times 2^{i-j-1}$。

使用以上步骤进行一个 i 的贡献统计需要 $\mathrm{O}(N)$ 次将答案加上 $y\times 2^i$，其中 y 是一个较小的正整数。这可以使用以二的幂次为进制数的高精度以近似 $\mathrm{O}(N)$ 的复杂度实现。

由于需要对 $1\le i\le N$ 全部统计贡献，这部分可以在近似 $\mathrm{O}(N^2)$ 的时间复杂度内解决。

以上过程给出了求出了 c 之后高效解决此题的方法。

4.2 第一部分

以下描述的算法仅会提供第一部分的解法，第二部分的解法已在上述过程中被解决。

4.2.1 算法三

枚举 c 的长度，之后二分 c 的大小。对于二分时的一个值 mid，采用以上方法求出 $g(1,mid),g(2,mid),\ldots,g(N,mid)$，并计算总长

$$len=\sum_{i=1}^{N} i\times(g(i,mid)-2^{i-1}+1).$$

通过比较 len 与 k 的大小来决定二分区间的取舍。最后若满足 $0\le k-len<|f(c)|$，则说明此时枚举的长度正好为 c 的长度。

复杂度 $O(N^4)$。

期望得分：49 分。

4.2.2 算法四

引理 5. *对于任意 $2^{N-2}\le i<2^{N-1}-1$，有 $Q_{i+1}-Q_i\le 2N+2$。*

证明. 对于所有 $1\le j<N$，满足长为 j、且排在 i 与 $i+1$ 之间的数只可能有两个：$f(i)[1\ldots j]$ 与 $f(i+1)[1\ldots j]$。

长为 N 且排在 i 与 $i+1$ 之间的数只可能有四个，$f(i)$+“0”、$f(i)$+“1”、$f(i+1)$+“0”、$f(i+1)$+“1”。

则排在 i 与 $i+1$ 之间的数只可能有 $2(N-1)+4=2N+2$ 个。

□

引理 6. *$n-Q_{2^{N-1}-1}\le 1$。*

证明. 排在 $2^{N-1}-1$ 后面的数必定只由 1 构成，且长度大于 $N-1$。

若 $n=2^N-1$，则 $n-Q_{2^{N-1}-1}=1$，否则 $n-Q_{2^{N-1}-1}=0$。

□

引理 7. *若 $g(N-1,c)$ 存在，则 $Q_c-Q_{g(N-1,c)}\le 2N+2$。*

证明. 根据引理五和引理六可得，所有长为 $N-1$ 的数的间距不超过 $2N+2$，且最后一个长为 $N-1$ 的数之后至多只会有一个数。则对于任意一个数，若找到在其之前最大的长为 $N-1$ 的数，这个数和它的距离不会超过 $2N+2$。

□

引理 8. $Q_{2^{N-2}} = 1$。

证明. 容易发现，排在 2^{N-2} 之前的数只有 2^{N-1}。

□

根据引理七和引理八的结论，可以发现若 c 之前（不包括 c）没有长为 $N-1$ 的数，则 c 只有可能是 2^{N-1} 或 2^{N-2}。

否则，可以通过算法三的二分方法来求出 $g(N-1,c)$。设 $g(N-1,c)=x$，之后需要求出排在 x 之后的 $2N+2$ 个数，同时维护 $len=\sum_{i=1}^{N} i\times(g(i,x)-2^{i-1}+1)$。

有关找出排在 x 之后的 $2N+2$ 个数，可以通过对于所有 $1\le i\le N$，维护 $g(i,x)$ 并将 $f(g(i,x)+1)^{\infty}[1\ldots 2N]$ 插入字典树中。每次操作可以在字典树中 $\mathrm{O}(N)$ 找到字典序最小的数并将其视作下一个数，同时需要将 len 加上这个数的长度。如果一次操作结束后发现 $len>k$，则当前的数就是 c。

这个部分的复杂度为 $\mathrm{O}(N^2)$。

由于减少了枚举长度的步骤，二分的部分复杂度降为 $\mathrm{O}(N^3)$。

期望得分：72 分。

4.2.3 算法五

发现算法四的瓶颈在于二分，而二分之后的步骤的复杂度似乎不好优化。则可以尝试将二分改为逐位确定，维护一个数

$$x=\overline{b_1b_2\ldots b_{N-1}}$$

和

$$len=\sum_{i=1}^{N} i\times(g(i,x)-2^{i-1}+1).$$

初始时

$$b_1=b_2=\cdots=b_{N-1}=0.$$

之后以 $b_1,b_2,\ldots,b_{N-1}$ 的顺序从高位到低位枚举，每次尝试将将 b_i 设为 1。如果设为 1 之后发现 $len>k$，则将 b_i 重新改为 0。最后得到的 x 即为 $g(N-1,c)$。

此处就需要支持修改 x 的一位，并快速维护 len 的值。

对于 $g(N,x)$ 的贡献，可以使用 $\mathrm{O}(N)$ 的时间单独处理。

对于 $1\le i<N$，$g(i,x)=h(f(x)[1\ldots i])$ 或 $h(f(x)[1\ldots i])-1$。定义数列

$$d_1,d_2,\ldots,d_{N-1}(0\le d_i\le 1),$$

满足

$$g(i,x)=h(f(x)[1\ldots i])-d_i.$$

那么每次修改会造成的影响为：$h(f(x)[1\dots i])$ 可能被修改了一位、d_i 被修改了。而这个修改对应到 *len* 上即为 *len* 变动了 $i\times(y\times 2^j+z)$（$-1\le y,z\le 1$）。

如果可以使用一些方法快速地维护的 $h(f(x)[1\dots i])$ 与 d_i 的变动，那么对于所有的 $1\le i<N$，可以使用以二的幂次为进制数的高精度以近似 $\mathrm{O}(N)$ 的复杂度将所有的 $i\times(y\times 2^j+z)$ 累加。

对于 $h(f(x)[1\dots i])$ 的修改，可以观察该次的修改操作是否影响到了 x 的前 i 位。如果是，执行对应的修改即可。

对于 d_i 的修改，定义

$$r_j=\Big[f(x)^\infty[j]=f(x)^\infty[j+i]\Big],$$

则若 r_j 为全 1 串则

$$d_i=\Big[i=N-1\Big],$$

否则设 r_j 第一个为 0 的位为 l，则

$$d_i=\Big[f(x)^\infty[j]=f(x)^\infty[j+i]\Big].$$

根据定理二，只需维护 r 的前 $2(N-1)$ 项即可。而当修改 x 的一位时，对应到 $f(x)^\infty$ 上只需修改 2 项，此时对应到 r 上只需修改 4 项。而之后问题就变成了修改 r 的一项，并快速给出 r 的第一个 0 的位置。

此处可以采用分块算法，由于在本题的数据范围下，$2(N-1)\le 4000$，此时将块大小设为 64，并使用一个 unsigned long long 维护每块的信息。而块数同样不会超过 64，可以使用 unsigned long long 维护每块是否存在 0。在查询时，可以先使用位运算找到第一个存在 0 的块，并将在这个块中找到第一个 0 的位置。

这样就可以在近似 $\mathrm{O}(1)$ 的时间复杂度内维护单个 d_i，而修改 x 的一位也达到了近似 $\mathrm{O}(N)$ 的复杂度。

在逐位确定的过程中需要进行 $\mathrm{O}(N)$ 次修改，这样便可以在近似 $\mathrm{O}(N^2)$ 的复杂度找到 $g(N-1,c)$，并解决此题。

期望得分：100 分。

5 总结

这道题最终的算法没有用的任何复杂的知识点，看上去是一道简单的高精度题，却考察了选手的思维能力、问题分析能力、代码能力。本题最终的算法将二分查找替换成逐位确定，并与本题的内容相结合，进行了优化，可以很好的锻炼选手思维的扩展性与创新性。本题的代码难度也较高，可以考验选手的代码基本功。

出题人希望使用该题启发大家，希望大家可以创造一些使用简单的知识点，却能够得到比较高的难度和比较优秀的区分度的好题。

参考文献

[1] 刘汝佳. 算法竞赛入门经典 [M]. 清华大学出版社, 2009.

[2] Cormen T H, Leiserson C E, Rivest R L, et al. Introduction to algorithms[M]. MIT press, 2009.

《最小连通块》命题报告

浙江省杭州第二中学 潘骏跃

摘 要

本文介绍了作者在一次校内联测中命制的一道交互题。该题解法多样，需要选手对树的结构有较为深刻的认识并能灵活地运用树的性质，是一道考察选手对树这种数据结构的了解程度的好题。

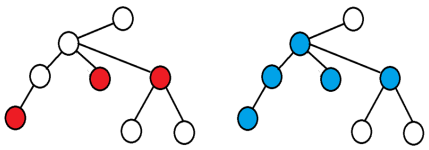
1 题目大意

1.1 题目描述

这是一道交互题。

对于一棵树 T，我们定义这棵树上的某个点集的最小连通块为包含这个点集中所有点的最小的树上连通块。

例如在下图中，蓝点所构成的点集就是红点所构成点集的最小连通块。



已知某一棵树的大小 n，你可以进行若干次询问，每次询问你可以给出一个点集 S 和这棵树上的一个点 x，交互库会返回一个布尔值表示 x 是否在点集 S 的最小连通块上。

你需要确定这棵树的形态。

本题保证所使用的树在交互开始之前已经完全确定，不会根据和你的程序的交互过程动态构造。

1.2 实现细节

你不需要，也不应该实现主函数，你只需要实现函数 $work(n)$，其中 n 表示所求树的点数。你可以调用如下四个函数来与交互库进行交互：

- *clear*()，表示清空当前的点集 S。
- *add*(x)，表示从点集 S 中加入 x 号点。
- *query*(x)，表示询问点 x 是否在点集 S 的最小连通块上。
- *report*(x,y)，表示确定所求树中存在一条边 (x,y)。

评测时，交互库会恰好调用 *work*(n) 一次。

我们只会对 *query* 操作的次数进行限制。

1.3 评分方式

本题共有一个测试包，内含若干个测试点。

对于每个测试点，若你的程序有不合法的询问或返回，或返回的树的形态不正确，则你在该测试点获得 0 分。否则令 *step* 表示你的程序的询问次数，则你在该测试点获得的分数将评定为 $min(\lfloor\frac{2.2\times10^6}{step}\rfloor,100)$。

你所获得的这道题的分数即为所有数据点的分数的最小值。（可以发现，若要获得满分，则询问次数不能超过 22000 次）

1.4 限制与约定

对于所有数据，均满足 $n=1000$。

时间限制：5s

空间限制: 1024MB

2 解法分析

2.1 算法一

在下文中，我们称一次“询问 x 是否在点集 S 的最小连通块上”的操作为 *query*(S,x)。

首先我们将给出的询问方式转化，询问 x 是否在点集 S 的最小连通块上等价于询问 S 中是否存在两个点 u,v，满足 x 在 u 到 v 的链上。

那么如果我们询问的点集 S 大小为 2，我们就可以直接询问出询问的点 x 是否在 S 中两个点所对应的那条链上。

我们可以直接得到一个多项式复杂度的做法，即对于任意三个点 u,v,w，通过 *query*($\{u,v\}$,w) 得到它们之间的关系，并由此确定整棵树的形态。

也就是说，我们得到了：

算法一：对于任意一对点 (u,v)，我们对剩下的任意一个点 w 进行一次 $query(\{u,v\},w)$，若这 $n-2$ 次询问的答案均为 false，则 u 与 v 直接相连。借此可以得出所有直接相连的点对，确定出整棵树的形态。

算法复杂度： $O(n^3)$

2.2 算法二

可以注意到我们并不需要对任意三个点 u,v,w 得到 $query(\{u,v\}.w)$ 的结果。实际上，将这棵树视为一棵有根树之后，如果我们能得出任意一对点之间的“祖先-后代”关系，那么也能确定出整棵树的形态。而这种做法也是解决这类问题的一种常用做法。

在下文中，我们都将该树视作一棵有根树，同时将该树的 1 号点视为根。

而要得出一对点 u,v 之间是否存在“祖先-后代”关系，只需要 $query(\{1,u\},v)$ 即可。

也就是说，我们得到了：

算法二：对于任意一对点 (u,v)，我们通过 $query(\{1,u\},v)$ 判断出 v 是否为 u 的祖先。我们可以借此得到任意一个点的祖先集合，设 i 号点的祖先集合为 Anc_i（特殊地，我们认为 i 号点本身也是 i 号点的祖先）。那么对于一个点 x 来说，它的父亲 f 就是满足 $Anc_f \subset Anc_x$ 的 $|Anc_f|$ 最大的点。确定每个点的父亲，我们确定了这棵树的形态。

算法复杂度： $O(n^2)$

2.3 算法三

注意到我们仍然只使用了点集大小为 2 时的询问，接下来我们将挖掘所给询问方式更多的用法。

既然可以得到两个点之间的“祖先-后代”关系，那么我们也可以用类似的方法只通过一次询问得出多个点之间是否存在“祖先-后代”关系。

对于一个点集 S 与一个点 u 我们进行一次 $query(S\cup\{1\},u)$，若结果为 true 则说明 S 中存在 u 的后代，否则不存在。这是因为如果 S 中同时存在两个点使得这两个点一个在 u 的子树内一个在 u 的子树外，那么它们所构成的那条链就会经过 u，而此时我们已经提供了一个子树外的点 1 号点；同样地，要想存在两个点使得它们对应的链经过 u，则这两个点必然有至少一个在 u 的子树内。故“S 中存在 u 的后代”与“该询问返回值为 true”是充要的。

进一步挖掘，若确认 S 中存在 u 的后代后，我们可以利用“二分”的方式直接找到 S 中 u 的某一个后代，具体操作方式如下：

- 将点集 S 分为两个点集 S_L,S_R，使得 $S_L\cup S_R=S$ 且 $S_L\cap S_R=\emptyset$ 且 $-1\le|S_L|-|S_R|\le 1$。
- 进行一次询问 $query(S_L\cup\{1\},u)$，若返回值为 true 则将 S 修改为 S_L，否则将 S 修改为 S_R。

- 重复上述两个步骤直至 $|S|=1$，此时 S 内的点即为所求的后代。

由于每次询问后 S 的大小至多变为 $\lceil\frac{|S|}{2}\rceil$，所以一次“找到某一个后代”的操作的复杂度为 $O(\log|S|)$。

更进一步挖掘，我们可以通过如下步骤找出 S 中 u 所有的后代：

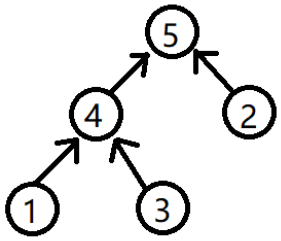
- 判断 S 中是否有 u 的后代，若无则直接退出。
- 找到 S 中 u 的一个后代。
- 从 S 中将找出的后代去掉，回到第一个步骤。

设 S 中 u 的后代个数为 m，则这样做的复杂度为 $O(m\log|S|)$。

得到上述工具后，我们发现可以将问题进行转化。

若我们将有根树的每一条边视为从儿子指向父亲的有向边，那么我们可以将有根树视作一张有向无环图，定义出一棵有根树的拓扑序。

例如下图中，对于这棵有根树的某种拓扑序，每个点上的数字展现了它在该拓扑序上的位置：



我们发现，如果我们能得出所求树的拓扑序，我们就能直接确定这棵树。具体的操作步骤如下：

- 设 $a_{1\ldots n}$ 为该树拓扑序上第 i 个点的编号，集合 V 的初始值为该树的点集。
- 令 i 从 1 扫到 n，每一步在 V 中找到 a_i 所有的后代，然后将这些后代从 V 中去掉。

由于一个点非儿子的后代已经在这些后代对应的父亲处从 V 中被去掉，所以一个点找到的那些后代就是它在原树中所有的儿子。

可以证明这部分的算法复杂度为 $O(n\log n)$，因为每个点被作为后代找到只有一次，而每次找到一个后代的复杂度都是 $O(\log n)$。

从这里开始，我们把问题转化为了求出原树的一种拓扑序。接下来的所有做法都是以“找出拓扑序”为目的。

考虑拓扑排序的常用算法，我们每次找到一个入度为 0 的点，然后把它加入当前拓扑序的末尾并在原图中把它删去。转化到树上，就是每一次找到原树的一个叶子，再把它从树中去掉。

我们称呼这种做法为“剥叶子”。

而判断一个点 u 是否为叶子很简单，设该树的点集为 V，则 $query(V-\{u\},u)$ 即可。

也就是说，我们得到了：

算法三：每一轮我们扫一遍原树的每一个点并判断它是否为叶子，若是则将其加入拓扑序的末尾，然后在该轮结束时从原树的点集中把所有找到的叶子删掉。判断一个点是否为叶子的方法和找到拓扑序之后的做法在此以及之后不再赘述。

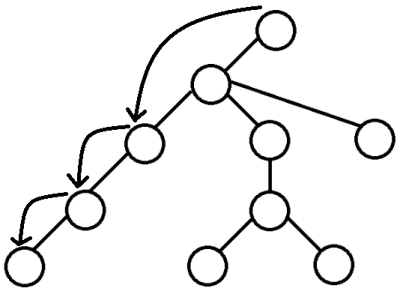
算法复杂度： $O(n^2)$

2.4 算法四

事实上我们每一次试图找到一个叶子的算法复杂度太高了，可以尝试在这里优化。

我们可以再一次利用之前的工具，每次不断从该树的点集中随机找到当前点的一个后代然后“跳”过去，直到找到一个叶子为止。

下图就是该算法“跳”的过程的一个例子：



令 siz_x 为 x 号点的子树大小，则对于每一个点 u 来说，它的后代里子树大小超过 $\lceil\frac{siz_u}{2}\rceil$ 的最多占到一半，所以最多期望 2 次就可以使当前所在点的子树大小变为原来的一半。由此可以证明，按照这种做法往下“跳”，期望“跳”的次数为 $O(\log n)$。而每次找到一个后代的复杂度也为 $O(\log n)$，故找到一个叶子的复杂度为 $O(\log^2 n)$。

也就是说，我们得到了：

算法四：这是一个随机算法。我们通过剥 n 次叶子来确定拓扑序。每次初始时我们令 $u=1$，再找到 u 号点的随机一个后代 v 号点，并将 v 赋值给 u，直到 u 成为一个叶子。

算法复杂度： $O(n\log^2 n)$

2.5 算法五

算法三还有另一种优化角度，那就是优化“剥叶子”的轮数。

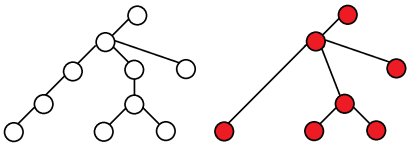
可以注意到若每次每一轮把原树的所有叶子剥掉，最劣情况下（也就是当原树为一条链时）需要剥 n 轮。

我们发现树有一个性质，那就是如果这棵树中没有二度点，则这棵树的叶子个数至少为总点数的一半。由此可以自然地想到如下的定义。

定义一棵树的虚树为对该树执行如下操作之后所形成的树：

- 找到当前树除根外的一个二度点 w，设它连接的两个点分别为 u, v。若找不到则退出。
- 将 w 以及以 w 为端点的边从树中删去，并加入一条边 (u, v)。
- 回到第一个步骤。

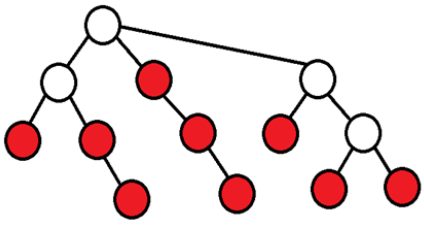
例如在下图中，右边的树即为左边的树的虚树。



可以注意到，如果我们能让每一轮所求树虚树的叶子都被剥掉，那么我们就可以用 $O(\log n)$ 轮剥掉所有的点。

而如果要使某一轮虚树的叶子都被剥掉，那么我们将要在原树中剥掉的点就是，原树中的叶子以及叶子之上那条由二度点构成的链，换句话说，后代中只有一个是叶子的那些点（特殊地，认为一个点也是它自己的后代）。

如下图，红点就是我们要在树中剥掉的点：



同时，找到这些点也是容易的，只需要先找出叶子集合 L，然后再对于每个点，判断它是否有大于等于 2 个后代在 L 中。判断方式是，先找出该点在 L 中的一个后代 v，然后看 $L - \{v\}$ 中是否存在该点的后代。

找出这些点之后按照它们对应的那个叶子后代分组，每组之内还要进行一次“祖先-后代”的排序。

这样我们就完成了一轮的工作，复杂度为 $O(n\log n)$。而我们一共要进行 $O(\log n)$ 轮，故总复杂度为 $O(n\log^2 n)$。需要特别注意的是，我们每一轮并没有让原树的点数变为原来的一半，所以不能认为复杂度是 $O(n\log n)$。

也就是说，我们得到了：

算法五：每一轮找出当前树中所有的叶子，再对于树上的其它点找出这些点的一个叶子后代并判断它们是否有大于等于 2 个叶子后代。将只有一个叶子后代的那些点按照叶子后代分组，每组内部进行排序，其中 u,v 之间通过 $query(\{1,u\},v)$ 来进行比较。最后将每一轮找到的那些点全部剥掉。

算法复杂度： $O(n\log^2 n)$

2.6 算法六

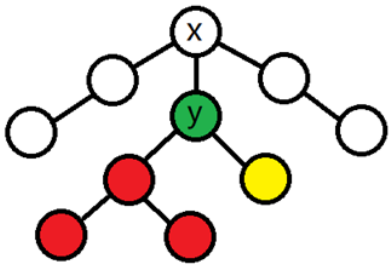
不再从“剥叶子”的角度入手，我们考虑这类问题的常用做法：分治。

我们令函数 $solve(x,S)$ 表示确定了 x 的子树的点集为 S，现在要将 x 子树内的点按照拓扑序排好。

随机找到 x 在 $S-\{x\}$ 中的一个后代 y，我们试图确定 y 的子树内的点所构成的点集为 S'，使得 x 的子树能被分为两个部分，以便进行分治。

实际上这与传统的点分治不太类似，但是我们依然能通过复杂度不错的确定性算法解决这个问题。

为了以较优的复杂度实现这个思路，我们需要先假设原树中这些点根据删掉点 y 之后所在的连通块被染成不同的颜色，如下图：



可以发现，我们的目的就是判断每一个点是否为白色。我们先把除了 y 以外的这些点按照结点标号排成一排，形如下图的样子：

称连续的极长的一串颜色相同的点为“一段”，现在我们尝试求出这排点段与段之间的分割线的位置。

我们知道当且仅当一个点集 A 中存在两个点颜色不同，$query(A,y)$ 会返回 true。那么我们就可以用一次询问判断连续的一串点是否全部颜色相同。当我们确定某一段的左端点时，就可以用这种操作二分出这一段的右端点，从左往右扫一遍就可以确定段与段之间的划分了。

最后可以使用 $query(\{1,u\},y)$ 判断点 u 是否为白色，借此判断出每一段是不是白色，就可以分治下去了。

考虑证明这样做的复杂度是对的。

设有 m 个点被染上了点数最多的那种颜色，试图确定段与段之间划分的复杂度是 $O((|S|-m)\log|S|)$ 的。这是因为每确定一段都需要付出 $O(\log|S|)$ 的复杂度，而段数是 $O(|S|-m)$ 的（出现次数最多的那种颜色最多被划分成 $|S|-m+1$ 段，而剩下的点也只有 $|S|-m$ 个）。

也就是说，我们可以视作每一次分治时，所染颜色不是点数最多的那种颜色的点会贡献 $O(\log|S|)$ 的复杂度。那么考虑每个点对复杂度的贡献：

- 若该点是一个白色点，由于白色不是出现次数最多的，说明被分开的另一个连通块比白色连通块大，白色连通块的大小小于等于原本连通块大小的一半，故只会发生不超过 $O(\log n)$ 次。
- 若该点是一个有色点，则若白色连通块是最大的就与第一种情况相同，否则该点所在连通块的根是 y 的一个轻儿子。又因为每个点被作为 y 选中至多只有一次，故这种情况也只会发生不超过 $O(\log n)$ 次。

由此，我们证明了总复杂度是 $O(n\log^2 n)$ 的。

也就是说，我们得到了：

算法六：处理 x 的子树时，设子树内的点分别为 $a_{1\ldots cnt}$（此处 $cnt=|S|$）。随机找到 x 的一个后代 y，假定树被因此染色。初始时令 $l=1$，每次二分找到最大的 r 满足 $l\le r\le cnt$ 且 $a_{l\ldots r}$ 同色，再令 $l=r+1$，直至 $l>cnt$。确定段与段之间的划分之后对于每个之前出现过的 l 询问 $query(\{1,a_l\},y)$ 判断该段颜色。最后根据颜色分治。

算法复杂度：$O(n\log^2 n)$

2.7 算法七

事实上采用类似分治的思想可以获得更优的复杂度。

令函数 $solve(x,S)$ 表示要求出 S 中有哪些点在 x 子树内并将它们按照拓扑序排好。也就是说，在 $solve(x,S)$ 之前我们并不确定 x 的子树内有哪些点。

算法流程如下：

- 找到 $S-\{x\}$ 中 x 的一个后代 y。若不存在这样的后代则退出。
- 执行函数 $solve(y,S)$，然后将 S 中 y 子树内的点删去，并将 y 子树内的点按照拓扑序加入当前拓扑序末尾。

- 回到第一个步骤。

初始时只要 $solve(1,\{1...n\})$ 即可。由于在该算法中每个点被作为 y 找到只有一次，所以总复杂度为 $O(n\log n)$。

这个算法的核心思想是，抛弃掉那些实际上无用的信息，将 x 的子树一块块剥掉，直到只剩 x。虽然流程十分简单，但是想到它的思维难度却并不低。

也就是说，我们得到了：

算法七：处理 x 的子树时，不断找到 x 的一个后代 y 并对 y 分治下去，最后把 x 子树内 y 的部分剥掉。

算法复杂度： $O(n\log n)$

2.8 算法八

上述的任意一种做法其实都是从树的结构出发，但实际上我们也可以直接在拓扑序上考虑。

我们知道对于这样一棵有根树的拓扑序有一个充要条件，那就是每个点的祖先都必须在这个点之后出现。如果我们能一个一个把点加入拓扑序并时刻维护这个性质，就可以解决整个问题。

我们已经有了判断点集 S 内是否存在 x 的后代的方法。那么只要二分出当前拓扑序的一个位置，使得这个位置之后不存在 x 的后代且这个位置尽量靠前，我们就可以保证加入这个点之后它所有的祖先仍然在它后面，它所有的后代仍然在它前面。

由于每加入一个点时我们只需要二分，所以总复杂度为 $O(n\log n)$。

也就是说，我们得到了：

算法八：枚举 i 从 1 到 n，设将 i 号点加入拓扑序之前，拓扑序为 $a_{1...i-1}$。二分一个位置 p 使得 $query(\{1\}\cup\{a_{p...i-1}\},i)$ 的返回值为 false 且 p 尽量小，将 i 插到第 p 个位置之前，也就是将拓扑序改为 $a_1,...,a_{p-1},i,a_p,...,a_{i-1}$。最后沿用得到拓扑序之后的做法即可。

算法复杂度： $O(n\log n)$

3 总结

本题题面简单精巧，所需要的知识仅是树的一些性质，却综合考察了选手制造工具、转化问题的能力，且解法中大量运用到了分治的技巧。该题对选手的代码能力要求不高，但对选手的思维有一定的要求。

解决本题需要将问题转化为“寻找原树的一种拓扑序”，在转化的过程中强调了“将给出的询问封装为一种种工具”的想法。将问题转化为“寻找拓扑序”之后，本题可以从“点分治”、“剥叶子”、“直接维护拓扑序”等多种角度切入，且都可以得到不错且相当有趣的算法。例如，我们巧妙地运用了“将点染色并划分为段”的思想得出算法六并大量使用树的

性质证明了算法六的复杂度；再例如，我们利用“没有二度点的树至少有一半的点时叶子”的思想得出了算法五。

出题人认为这样一道不需要高深的知识但能深入考察选手对树的性质的了解的题称得上是好题，也希望通过本题起到抛砖引玉的作用，看到更多更有趣的题目。

4 感谢

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练高闻远的指导。

感谢杭州第二中学的李建老师对我的关心与指导。

感谢周欣同学、方尤乐同学、方汤骐同学等与我讨论该题算法，帮忙验题。

感谢父母对我的关心与支持。

参考文献

[1] Codeforces 1129E, Legendary Tree.

转置原理的简单介绍

陈宇 安徽师范大学附属中学

摘 要

本文介绍了转置原理（transposition principle），也被称为特勒根（Tellegen）原理，是一组对线性算法（linear algorithm）改写的规则。文中会介绍一些常用线性算法的转置，指出转置方法对一些问题可起到方便解决和优化效率的作用，并会介绍用转置原理优化的一类特殊的矩阵乘向量及其应用。

1 概述

我们假设以下所有对数的操作在一个数域 F 上进行。

定义 1. 输入一个 $n\times 1$ 向量 a，用一个 $n\times n$ 的常数矩阵 A 左乘该向量后，输出其结果 $b=Aa$（b 便为一个 $n\times 1$ 向量）的算法，被称为**线性算法**。我们认为输入的向量是变量，矩阵为常量。为了方便这里只考虑 A 是方阵，否则适当补零即可。

定义 2. 对于形如 $b=Aa$ 的线性算法，我们称形如 $b'=A^Ta'$ 的线性算法是该**线性算法的转置**。

而转置原理断言，我们可以将一个线性算法改写成其转置后的算法，同时保持时空复杂度不变。这样我们就可以将算法进行转置后，优化该转置后的算法，再将该转置后的算法改写成解决原问题的算法。这在许多场合可以简化问题。

2 改写

这一节我们将指出，将线性算法改写为其转置，是一个较为机械的过程。

我们首先探讨矩阵转置后对算法过程的影响。

为了方便，我们将程序用到的与变量有关的额外空间，均算在输入向量中，这样我们适当改变输入输出即可得到相同效果。

定义 3. 我们在单位矩阵 I 上略作修改，得到以下两种矩阵：

1. 将 I 的第 i 行和第 j 行交换。该矩阵左乘一个向量的效果便为，将其第 i 位与第 j 位交换；
2. 将 I 的第 i 行第 i 个元素由 1 改为数 a。该矩阵左乘一个向量的效果便为，将其第 i 位乘上 a；
3. 将 I 的第 i 行第 $j(j \neq i)$ 个元素由 0 改为常数 a。该矩阵左乘一个向量的效果便为，将其第 j 位乘上 a 加到第 i 位上。

我们将这三种矩阵称为**初等矩阵**。

矩阵乘向量一般要使用 $O(n)$ 的额外空间。经过处理后，矩阵 A 就可以直接拆成一系列初等矩阵的乘积

$$A = E_1 E_2 \dots E_k$$

从而有：

$$A^T = E_k^T E_{k-1}^T \dots E_1^T$$

考虑初等矩阵的转置：第一种和第二种转置后不变，第三种是把第 i 位加到第 j 位改为第 j 位加到第 i 位。于是转置后的算法，就是将原算法的所有语句倒序执行，并将原来算法中形如 $x \leftarrow x + ay$ 的语句调整为 $y \leftarrow y + ax$。对于加常数操作，可在矩阵中补 1 实现。

实际应用时，我们可能要输入一个矩阵，然后根据该矩阵计算出算法使用的所有常量，得到一个线性算法。

对于函数参数。其中涉及到常量的部分原样不动，变量则是传入改为返回或修改，返回或修改改为传入。

函数调用只需要正常直接倒序，依次将里面的调用的其他函数转置执行即可；递归也可看做是对其他函数的调用。需要注意的是，由于我们要在反向执行的每个时刻，算出正向执行到这里的所有常量，这可能需要先预处理比较方便。

3 例子

3.1 离散傅里叶变换

考虑 DFT 的矩阵：

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix}$$

由于该矩阵是对称矩阵，该算法的转置即为其自身。IDFT 也类似。

3.2 快速傅里叶变换

广为使用的 FFT 是 DIT（decimation in time，按时域抽取）-FFT 形式的，即将多项式系数分为偶数项和奇数项，容易根据单位根性质递归为子问题，不再赘述；考虑改写成非递归，常用的方法是将数组下标的二进制位反转，这样每次取最低位变为最高位，偶数项和奇数项变成左半右半，就容易处理了。

另一方面，我们考虑直接分成前一半后一半：

$$\begin{aligned} DFT(a,n)_k &= \sum_{i=0}^{n-1} a_i \omega_n^{ik} \\ &= \sum_{i=0}^{n/2-1} \left(a_i^{(left)} + a_i^{(right)} \omega_n^{n/2k}\right) \omega_n^{ik} \end{aligned}$$

$\omega_n^{n/2k} = (-1)^k$，通过讨论 k 的奇偶性，可以得到一个将偶数项答案放在左边，奇数项答案放在右边的算法。这样改成非递归后，只需最后将数组下标的二进制位反转。

这个算法被称为 DIF（decimation in frequency，按频域抽取）-FFT。观察算法过程，两个功能相同的算法互为转置，也印证了 DFT 转置是其自身。

3.3 多项式乘法

我们用 $M(n)$ 表示求两个多项式循环卷积的前 $2n$ 位的时间，并假定 $M(n) = O(n \log n)$。

对最高次数分别为 $n-1$ 和 $m-1$ 的多项式 a 和 b，暴力多项式乘法得到次数为 $n+m-1$ 的多项式 $c = ab$ 的程序如下：

```
1  c ← 0
2  for i ← 0 to n + m − 2
3      for j ← max(0, i − n + 1) to min(i, m − 1)
4          c_i ← c_i + a_{i−j} b_j
```

我们将 b 视为常数，直接按上面规则转写，得到乘法的转置：给定次数为 $n-1$ 和 $m-1$ 的多项式 a 和 b，结果为 $n-m$ 次的多项式 $c = mul^T(a,b)$：

```
1  c ← 0
2  for i ← 0 to n + m − 2
3      for j ← min(i, m − 1) downto max(0, i − (n − m))
4          c_{i−j} ← c_{i−j} + a_i b_j
```

从而可看出，$mul^T(a,b)$ 是 a 与翻转系数后的 b 卷积结果的 $m-1$ 至 $n-1$ 位。

多项式乘法的时间自然是 $\frac{1}{2}M(n+m)$，转置乘法的时间则可做到 $\frac{1}{2}M(n)$，这是因为循环卷积溢出部分与我们需要的部分并不相交。同样的做法利用 [4.1] 中的结论也容易导出。

4 范德蒙德矩阵的转置

定义 4. 范德蒙德矩阵是一个 $n\times n$ 方阵，形如：

$$V_\alpha=\begin{bmatrix}1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^{n-1}\\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{n-1}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & \alpha_{n-1} & \alpha_{n-1}^2 & \cdots & \alpha_{n-1}^{n-1}\end{bmatrix}$$

定义 5. 对于最高次数为 $n-1$ 的多项式 b（其系数的向量也写成 b），

$$V_\alpha b^T=(b(\alpha_0),b(\alpha_1),\ldots,b(\alpha_{n-1}))^T$$

这个计算一般被称为**多点求值**。

下面我们将说明，转置原理的应用可以大幅度减小多点求值的时间常数。先介绍几个与该问题有关的问题。

4.1 多项式整除和取模

给定 $n-1$ 次多项式 f 和 $m-1(m\le n)$ 次多项式 g，求 h,r 使得 $f=gh+r$ 使得 $|r|<n$.（假定 m 大约是 $0.5n$）

对于该问题，Sieveking-Kung's algorithm 给出了计算公式：我们令 $rev(a,n)$ 表示 $n-1$ 次多项式 a 翻转系数的结果，有：

$$rev(n-m+1,h)=rev(n,f)\frac{1}{rev(m,g)}\pmod{x^{n-m+1}}$$

形式幂级数求商可以使用 $\frac{7}{3}M(m)=\frac{7}{6}M(n)$ 的时间完成 [1]。之后算出 $r=f-gh$ 还需要使用 $M(m)=\frac{1}{2}M(n)$ 的时间，总时间 $\frac{5}{3}M(n)$。

4.2 多个单项式求乘积

给定 $a_1,a_2,\ldots,a_n$，输出多项式 $(1+a_1x)(1+a_2x)\ldots(1+a_nx)$ 的系数。

我们可以直接使用分治解决。具体我们令

$$F(l,r)=\prod_{i=l}^{r}(1+a_ix)$$

用

$$F(l,r)=F\left(l,\left\lfloor\frac{l+r}{2}\right\rfloor\right)\times F\left(\left\lfloor\frac{l+r}{2}\right\rfloor+1,r\right)$$

计算。

这样的时间为

$$T(n)=2T(n/2)+M(n/2)=\frac{1}{2}M(n)\log_2 n+O(M(n))$$

4.3 幂和

给定 $a_1, a_2, \ldots, a_n$，令 $f(k) = \sum_{i=1}^{n} a_i^k$，求出 $f(0), f(1), \ldots, f(n-1)$。

考虑形式幂级数

$$F(z) = \sum_{k\geq 0} f(k) z^k = \sum_{k\geq 0} \sum_{i=1}^{n} a_i^k z^k = \sum_{i=1}^{n} \frac{1}{1 - a_i z}$$

即求 $F(z) \mod z^n$。

易知

$$\prod_{i=1}^{n} (1 - a_i z) F(z) = \sum_{i=1}^{n} \prod_{j\neq i} (1 - a_j z)$$

令其为 $A(z)F(z) = B(z)$，只需分别求出 $A, B \mod z^n$ 后用一次形式幂级数求商即可。

我们令 $A(l,r), B(l,r)$ 定义与 [5.1] 中的 $F(l,r)$ 类似，那么 A 类似求出，而

$$B(l,r) = A\left(l, \left\lfloor \frac{l+r}{2} \right\rfloor\right) B\left(\left\lfloor \frac{l+r}{2} \right\rfloor + 1, r\right) + B\left(l, \left\lfloor \frac{l+r}{2} \right\rfloor\right) A\left(\left\lfloor \frac{l+r}{2} \right\rfloor + 1, r\right)$$

这样分治计算，容易看出是 $\frac{3}{2} M(n) \log_2 n + O(M(n))$ 的时间。

4.4 多点求值与插值

多点求值传统上是使用一个分治算法。考虑 b 求 α 中的点值，当 $|b| \geq |\alpha|$ 时，可以将 b 对 $(x - \alpha_0)(x - \alpha_1) \ldots (x - \alpha_{|\alpha|-1})$ 取模，从而令 $|b|$ 总是小于 $|\alpha|$。这样，我们就可以每次将点值分为两半递归下去。

这样的时间为 [5.3] 的时间加上每次分治调用 [5.2] 的时间，经过计算得为 $\frac{11}{3} M(n) \log_2 n + O(M(n))$。对于流行的牛顿法求逆未优化的实现，这个速度还会慢一倍以上。

下面，我们考虑范德蒙德矩阵的转置：

$$V_\alpha^T = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_0 & \alpha_1 & \cdots & \alpha_{n-1} \\ \alpha_0^2 & \alpha_1^2 & \cdots & \alpha_{n-1}^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{n-1} & \alpha_1^{n-1} & \cdots & \alpha_{n-1}^{n-1} \end{bmatrix}$$

考虑转置后的问题 $V_\alpha^T b = c$，那么 $c_i = \sum_{j=0}^{n-1} \alpha_j^i b_j$，这与 [5.3] 中的问题基本类似，进行微小的修改后即得一个 $\frac{3}{2} M(n) \log_2 n + O(M(n))$ 时间的分治算法。

将该算法改写为转置前版本，会将原来一层分治的一次乘法变为两次，这会导致时间翻倍。改写后的算法时间为 $\frac{5}{2} M(n) \log_2 n + O(M(n))$。

插值的算法主要调用一次多点求值，与多点求值的时间也相同。

4.5 实践

我们对多点求值的实现提交在 https://www.luogu.com.cn/record/29053749，该题中 n 的量级为 2^{16}。我们的程序于目前（2020-1-08 之前）最快，并比未经底层优化的最快程序快一倍以上，符合以上理论计算。

之前多点求值在比赛中并不常见，并不一定是因为用处不多，而很大一部分原因是多点求值需要套用多种算法，记忆和编写较繁琐；同时算法本身常数很大，出题人不一定能在合理的时限内卡掉暴力，选手辛辛苦苦写完后又可能因为常数原因无法通过。而上述改进算法只需要使用分治和多项式乘法，常数也减小很多，一定程度上扩展了多点求值的实用性。

5 一类特殊矩阵乘向量的快速算法

5.1 描述

我们考虑一个线性变换 $A\alpha = \beta$。现在，考虑 $n \times n$ 矩阵 A 的系数的二元生成函数：$A(x,y) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} A_{i,j}x^i y^j$。如果 $A(x,y)$ 可以表示成 $u(x)v(y)f(g(x)h(y))$，其中 g 和 h 由一系列（常数个）我们将给出的简单函数复合而成，那么我们可以在 $\tilde{O}(n)$ 时间内完成该矩阵和其逆矩阵左乘向量；具体地，若 g,h 中含有 $\exp,\log$，则复杂度为 $O(M(n)\log n)$，否则为 $O(M(n))$（该复杂度优于之前所有已知方法）。为保证下面算法的运算有意义，还要求 gh 的常数项为 0，g 和 h 的一次项都非 0；u 和 v 常数项非 0，f 任意项非 0（如完全不需要逆可以省去）。

5.2 复合

定义 6. *考虑 $n-1$ 次多项式 F 和一个线性变换：$F(G(x)) \mod x^n$（$G(x)$ 是常数），称其为*右复合

这个计算并没有好的方法，但如果 $G = g_1 \circ g_2 \circ \ldots \circ g_k$，其中 g_i 是一些简单函数，那么我们可以运用复合的结合律：$A(B(C(x))) = A \circ B(C(x))$，之后将 F 进行 k 次对简单函数的复合就可算出。少部分函数由于不封闭会用到 $C(x)$ 的信息，需特别注意。如果符合上面提到的条件，除此之外的计算可以每一步都截断到 n 位而不丢失精度。

我们认为简单函数是以下几类：

1. 加法 $x+k$

   有

$$F(x+k) = \sum_{i=0}^{n-1} F_i(x+k)^i = \sum_{i=0}^{n-1} F_i \sum_{j=0}^{i} \binom{i}{j} x^j k^{i-j} = \sum_{j=0}^{n-1} \frac{x^j}{j!} \sum_{i=j}^{n-1} i! \frac{k^{i-j}}{(i-j)!}$$

显然可用 $M(n)$ 时间完成。

2. 乘法 kx，只需将 k 的若干次幂乘入系数，时间 $O(n)$。

3. 幂 $x^k(k \in \mathbb{Z}^+)$，将下标变换即可，没有进行任何算术运算。

4. 逆 x^{-1}，注意幂级数不能有负指数，考虑 $F(x^{-1}) = (rev(n,F))(x)x^{1-n}$，其中 x 是另一个被复合的幂级数，翻转系数没有进行算术运算，计算后面幂级数逆的系数需要 $O(M(n))$ 时间。

5. 根 $x^{1/k}(k \in \mathbb{Z}^+)$，这里需要保证 x 的 k 次方根唯一，推之前的 $g(x)h(y)$ 形式时就应当通过验证一些项去舍掉不合法的解。

   设 x 对应后面的幂级数是 g，且 $g = h^k$（这里要保证 h 的唯一性）。我们将 F 的下标做带余除法 F_{ik+j}，然后按模 k 分类：

   $$F_i(x) = \sum_j F_{jk+i}h^j$$

   那么有

   $$F(h) = \sum_i F_i(g)h^i$$

   递归为 $\lfloor n/k \rfloor$ 个规模为 k 的子问题后拼起来即可。求出 h 需要 $O(M(n))$ 时间 [1]，之后需用 $O(kM(n))$ 时间求出 h 的 $0 \ldots k-1$ 次幂，因此需保证 k 是常数。

6. 指数 $\exp(x) - 1$，减一是为了和对数互为逆，不妨不考虑减一，那么有：

   $$F(e^x) = \sum_{i=0}^{n-1} F_i e^{ix} = \sum_{i=0}^{n-1} F_i \sum_{j\geq 0} \frac{i^j}{j!}x^j = \sum_{j\geq 0} \frac{x^j}{j!} \sum_{i=0}^{n-1} F_i i^j$$

   我们考虑求出

   $$\sum_{j\geq 0} x^j \sum_{i=0}^{n-1} F_i i^j$$

   当然要截断到 n 位；之后我们将每个位置除以 $j!$ 即可。

   上述问题与之前提过的多点求值的转置问题是相同的，用分治和多项式乘法即可解决。复杂度 $O(M(n)\log n)$。

7. 对数 $\log(x+1)$，直接做并不好分析，我们不妨考虑复合指数算法的逆。

   复合指数算法可概述为，将 F 运行多点求值的转置问题 $0, 1, \ldots, n-1$，然后将每个位置乘 $\frac{1}{j!}$。

   将该算法转置可得，先将每个位置乘 $\frac{1}{j!}$，再对 $0, 1, \ldots, n-1$ 运行多点求值。其逆显然为，运行多点插值后将每个位置乘 $j!$。再转置回去复杂度不变，故为 $O(M(n)\log n)$。

更好的一点是，上面的函数都存在复合逆。

5.3 算法

我们回到开始的问题。先假设 $u = v = 1$，有：

$$F_{i,j} = \sum_k g_i^k f_k h_j^k$$

那么，该线性变换是三个线性变换的复合：g 右复合，点乘 f，h 右复合的转置，按上一节方法顺次计算即可。

加上 u, v 后，原线性变换变为三个线性变换的复合：乘 u，原线性变换，乘 v 的转置，同样可以轻松计算。

在限制条件下，上面这些操作都存在逆且可以直接取逆，故逆矩阵也可以相应计算。

综上，我们以很低的复杂度解决了该类问题。

6 应用

6.1 基变换

定义 7. *考虑一系列多项式 $\varphi_0(x), \varphi_1(x), \ldots, \varphi_n(x)$，若任何 n 次多项式都可以被它们线性表出，则这是模 x^n 多项式空间的一组基。OI 中常见的基有单项式基和下降幂基。*

多项式可以在不同的基下被表示。在线性代数中我们知道，从一个基的表示转换到另一个基，对应着一个线性变换。多项式基的变换往往对应 [6] 中提到的问题。我们有以下单项式基到其他基互相变换的结果：

- $O(M(n))$ 时间可计算：拉盖尔（Laguerre）多项式、厄米（Hermite）多项式、雅可比（Jacobi）多项式、斐波那契（Fibonacci）多项式、欧拉（Euler）多项式、伯努利（Bernouli）多项式、Mott 多项式、Spread 多项式、贝塞尔（Bessel）多项式
- $O(M(n) \log n)$ 时间可计算：下降幂、贝尔（Bell）多项式、第二类伯努利多项式、Poisson-Charlier 多项式、Actuarial 多项式、Narumi 多项式、彼得斯（Peters）多项式、Meixner-Pollaczek 多项式、Meixner 多项式、Krawtchouk 多项式

上面的许多基在 OI 中的优美应用尚待挖掘。

6.2 例题

6.2.1 真实无妄她们的人生之路

题目来源 Comet OJ #2 F

题目大意 有 $n(n \le 10^5)$ 件物品，第 $i(1 \le i \le n)$ 件物品有属性 $p_i(0 < p_i \le 1)$；

主人公等级初始为 0，使用第 i 件物品会有 p_i 的概率让等级加一，$1-p_i$ 概率不变；

若最后等级为 j，则会产生 a_j 的攻击力。令 f_i 表示使用除 i 外的 $n-1$ 个物品后，主人公产生的期望攻击力，求出 $f_1, f_2, \ldots, f_n$。在 $F_{998244353}$ 下计算。

解法 设 $C_i(x) = 1 - p_i + p_i x, C(x) = \prod_i C_i(x)$，则 C/C_i 与 a 点积结果的系数之和便为 f_i。

将 a 看做输入向量，这是一个线性变换问题 $Aa = f$，其中 $A_{i,j} = C/C_i[x^j]$。考虑转置问题 $A^T a = g$，则有 $g = \sum_i a_i C/C_i$。这可以用类似幂和问题的分治与多项式乘法求出。

6.2.2 随机游走

题目大意 给定 $n, a, b(n \le 10^5)$，在一个无穷大的二维平面上，一个人出发点是 $(0,0)$，每步可以沿向量 $\overrightarrow{(1,1)}$，$\overrightarrow{(1,0)}$ 和 $\overrightarrow{(2,0)}$ 走，分别带有 $1, a, b$ 的权值；

此人随时可以停止。若停在 (x,y)，则会产生 w_y 的权值；

定义一条路径的权是这些权值的乘积。对于 $1 \le a \le n$，求出结束在 x 坐标为 a 的所有点的所有路径权值之和 g_a。在 $F_{998244353}$ 下计算。

解法 不考虑 w_y，令 $f_{x,y}$ 表示到达 x, y 的路径权值和，$f_{0,0} = 1$，此外 $f_{i,j} = f_{i-1,j-1} + a f_{i-1,j} + b f_{i-2,j} (i, j \ge 0)$，此外 $f_{i,j} = 0$，那么 $g_a = \sum_y f_{a,y} w_y$，这是一个线性变换问题。

我们令 f 的二元生成函数为 $F(x,y) = \sum_{i,j} f_{i,j} x^i y^j$，那么方程 $F = 1 + x(a+y)F + x^2 F$ 成立，解得：

$$F = \frac{1}{1 - (a+y)x - x^2} = \frac{1}{(1-x^2)\left(1 - \frac{a+y}{1-x^2}\right)}$$

这可以转为 [6] 中的形式，其中

$$u(x) = \frac{1}{1-x^2}, v(y) = 1, f(x) = \frac{1}{1-x}, g(x) = \frac{1}{1-x^2}, h(y) = a + y$$

套用算法即得 $O(M(n))$；实际上这就是到一类正交多项式的基变换问题。

6.2.3 二叉树计数

题目大意 给定 $n(n \le 10^5)$，定义一个有 x 个叶子的二叉树权值为 w_x，设 $f(t)$ 表示所有 t 个点的无标号二叉树的权值和，请求出 $f(1), f(2), \ldots, f(n)$。在 $F_{998244353}$ 下计算。

解法 与上面类似的，令 $f_{i,j}$ 为 i 个点 j 个叶子的二叉树个数，F 为其二元生成函数，那么可列出 $F = x(y + 2F + F^2)$，解得：

$$F = \frac{1 - 2x - \sqrt{(2x-1)^2 - 4x^2y}}{2x} = \frac{1}{2x}(1-2x)\left(1 - \sqrt{1 - \left(\frac{1}{2x-1} + 1\right)^2 y}\right)$$

我们令

$$u(x) = 1 - 2x, v(y) = 1, f(x) = 1 - \sqrt{1-x}, g(x) = \left(\frac{1}{2x-1} + 1\right)^2, h(y) = y$$

套用算法即可算出里层，外面乘上 $\frac{1}{2x}$ 简单处理即可，复杂度 $O(M(n))$。

7 总结

本文探究了转置原理及其应用。转置原理给出了一种处理线性变换问题的崭新思路，从算法变化的角度考察了矩阵的变化。利用转置原理大幅优化了多点求值的常数和编程难度，揭示了许多看似不同经典问题蕴含的相同本质。在最后文章解决了一类范围很广的矩阵乘向量问题，让许多原来困难的计数问题被轻松解决，这些方面都可以看到转置原理广泛的应用、强大的威力和无穷的潜力。除本文提到的外，其他转置原理的巧妙运用还待读者发掘。希望本文可以对读者起到启发思想、拓宽出题与解题思路的作用。

8 致谢

感谢中国计算机学会提供交流和学习的平台。
感谢国家集训队高闻远教练的指导。
感谢叶国平老师在学习生活方面的帮助。
感谢朱震霆、周雨扬等学长同学的指导。
感谢罗恺、李白天、张好风等同学为本文审稿。

参考文献

[1] 关于优化形式幂级数计算的 Newton 法的常数（http://negiizhao.blog.uoj.ac/blog/4671）.

[2] Tellegen’s Principle into Practice, A. Bostan, G. Lecerf, ?. Schost, 2005.

[3] Power Series Composition and Change of Basis, Alin Bostan, Bruno Salvy, Eric Schost, 2008.

浅谈函数最值的动态维护

北京大学附属中学 李白天

摘 要

通过引入关于多个函数上包络线结构的讨论，本文主要对 OI 中的一些著名问题作出了如下改进：

- 将一个序列区间增加公差为正的等差数列，区间查询最值。本问题原本广为人知的做法为 $\Theta(\sqrt{n})$，而本文的做法目前有上界 $O(\log^2 n)$。
- 李超线段树可以在 $\Theta(\log^2 n)$ 内在平面中添加一条线段，并 $\Theta(\log n)$ 查询某一 x 位置上最大的 y 值。而本文通过完全不同的方法在保留询问复杂度的情况下将添加一条线段的复杂度改进为均摊 $\Theta(\alpha(n)\log n)$ 的理论复杂度，且该做法可在高次函数得到一般性的拓展。

1 概述

对于函数列 $f_i:\mathbb{R}\to\mathbb{R}$，我们希望支持对于函数列上进行一些修改，以及至少支持对于整体的最值查询，即给出 x，询问

$$\max_{j=1}^{n} f_j(x)$$

在接下来的分析中，我们需要引入 Davenport-Schinzel 序列的概念。

1.1 Davenport-Schinzel 序列

定义 1 ((n,s) Davenport-Schinzel 序列)**.** 记一个长度为 m 的序列 $\sigma_1,\sigma_2,\ldots,\sigma_m$ 是一个 (n,s) *Davenport-Schinzel* 序列（简记作 $DS(n,s)$ 序列），当且仅当 σ_i 为 1 至 n 中的整数，且满足：

- σ 中相邻两项值不同。
- 对于任意 $x\neq y$，任何 x,y 交替构成的序列如果是 σ 的子序列，则长度不超过 $s+1$。

接下来的一个定理尤为有效地刻画了 DS 序列：

定理 1. 记 $\lambda_s(n)$ 为 $DS(n,s)$ 序列可能的最长长度，有[1]

$$\lambda_s(n)=\begin{cases} n, & s=1 \\ 2n-1, & s=2 \\ 2n\alpha(n)+O(n), & s=3 \\ \Theta(n2^{\alpha(n)}), & s=4 \\ \Theta(n\alpha(n)2^{\alpha(n)}), & s=5 \\ n2^{\alpha(n)^t/t!+O(\alpha(n)^{t-1})}, & s\geq 6, t=\left\lfloor\dfrac{s-2}{2}\right\rfloor \end{cases}$$

其中 $\alpha(n)$ 是反 Ackermann 函数。由其增长速度我们可知，$\lambda_s(n)$ 对于任意常数 s 都是近乎线性的。

接下来给出 s 较小的两个情况的证明。

1.1.1 $s=1$ 情况的证明

假设序列中存在相同元素 $\sigma_i=\sigma_j(i<j)$，则说明 $\sigma_{i+1}\neq\sigma_i$，而 $\sigma_i,\sigma_{i+1},\sigma_j$ 构成了一个长为 3 的交替子序列，矛盾。

因此我们得到，序列中不存在重复元素，又因为 σ 的元素均为 $1\sim n$ 的正整数，所以 $m\leq n$，任何一个 $1\sim n$ 的排列均取到等号，因此 $\lambda_1(n)=n$。

1.1.2 $s=2$ 情况的证明

我们考虑施第二数学归纳法证明。对于 $n=1$ 的情况，$\lambda_2(1)=1$ 显然成立。

考虑对于 $n>1$，此时对于 $<n$ 的 n_0 均有 $\lambda_2(n_0)=2n_0-1$，考虑一个 $DS(n,2)$ 序列，不妨设 $\sigma_1=1$。

- 若 1 仅出现了一次，则 $m\leq 1+\lambda_2(n-1)=2n-2\leq 2n-1$。
- 设 1 出现的下一个位置为 i，则对于 $1<j<i$ 的所有元素 σ_j，因为不存在长为 4 的交替子序列，其在 $>i$ 的位置均不会再次出现。设 $2\sim i-1$ 中总共出现了 k 种元素，则说明 $2\sim i-1$ 构成一个 $DS(k,2)$ 序列，$\geq i$ 的部分构成一个 $DS(n-k,2)$ 序列，因此 $m\leq 1+\lambda_2(k)+\lambda_2(n-k)=1+2k-1+2(n-k)-1=2n-1$。

而 $m=2n-1$ 容易在形如 $1,2,\ldots,n-1,n,n-1,\ldots,2,1$ 的序列取到。综上所属，$\lambda_2(n)=2n-1$ 归纳成立。

[1] Seth Pettie, *Sharp Bounds on Davenport-Schinzel Sequences of Every Order*, 2015

1.2 DS 序列与问题的联系

我们认为 s 可以某种程度上衡量我们维护的一族函数的“复杂程度”。我们注意到，对于我们维护的一族函数，如果任何两个函数 f,g，$\max(f(x),g(x))$ 的分段数量不超过 $s+1$，那么 n 个该族函数的上包络线的分段情况便等价于某个 $DS(n,s)$ 序列，我们称这族函数是 s 阶交替的。因此上包络线分段长度最长只有 $\lambda_s(n)$。

其中最为常见的便是最高 s 次多项式。由于两个 s 次多项式的交点最多只有 s 个，因此最值的分段数量不超过 $s+1$ 个。自然而然地，我们知道对于 n 个最高 s 次多项式，上包络线分段长度不超过 $\lambda_s(n)$。

另一个值得注意的是在一段区间上定义的 s 次多项式。我们不妨将分段函数定义为：在原本定义域外的部分，函数值都为 $-\infty$。这样可以得到任意两个分段 s 次函数是 $s+2$ 阶交替的，因此 n 个分段 s 次函数的最值的上包络线分段长度不超过 $\lambda_{s+2}(n)$。

在之前 OI 研究的问题中，多为一次函数和分段一次函数，这两个问题分别有 $s=1$ 和 $s=3$。

在接下来的讨论中，我们将问题加以不同的特殊限制，并且在其上能够基于 $\lambda_s(n)$ 的上界设计出较为高效的算法。

1. 不对函数进行修改，仅作为一个集合来维护。这一限制下能够通过二进制分组的思路进行设计。

2. 询问的 x 保证是单调递增的。这一限制下能够通过类似 Segment Tree Beats 的思路进行设计。

2 函数集合维护

设有 n 个函数 $f_i:\mathbb{R}\to\mathbb{R}$，对于 $x_1,x_2,\ldots,x_m$ 中的每个 x_k，求

$$\max_{j=1}^{n} f_j(x_k)$$

当所有函数初始就被确定，我们可以进行分治，每次将当前函数分成两部分进行处理，然后给得到的两组分段函数合并。复杂度即为

$$T(n)=2T\left(\frac{n}{2}\right)+\Theta(\lambda_s(n))$$

根据主定理，解得 $T(n)=\Theta(\lambda_s(n)\log n)$。用于回答的时间为 $\Theta(m\log n)$。

对于函数被逐一确定的情况，我们可以使用二进制分组进行处理。

若插入 n 个最高 s 次函数，则可以做到 $\Theta(\lambda_s(n)\log n)$ 的总时间以及 $\Theta(\lambda_s(n))$ 的总空间进行维护，以及朴素实现可做到 $\Theta(\log^2 n)$ 时间处理每次询问。事实上，我们甚至可以做到 $\Theta(\log n)$ 时间处理每次询问。

对比 $s=3$ 的情况，本算法支持插入线段，均摊时间为 $\Theta(\alpha(n)\log n)$，而询问可以做到 $\Theta(\log n)$，相较于李超线段树的复杂度优秀，并且还有一个额外优点：本算法并不需要提前知道询问的 x 的值，它是完全在线的。

2.1 朴素实现

记 $k=\lfloor\log_2 n\rfloor$，n 的二进制分解为 $\sum_{j=0}^{k} n_j\cdot 2^j(n_j\in\{0,1\})$，对于 $n_j=1$ 的 j，我们维护块 L_j 为其中包含的 2^j 个函数的上包络线所组成的分段。每当加入一个函数的时候，如果存在两个包含同样多个函数的块，我们就将其合并，通过原来两块各自的分段计算出合并后的分段。当合并结束之后，可以得到剩下的各块大小即 n 的二进制分解。

整个过程可以看做一个 2^{k+1} 大小的分治的不完全执行，因此维护的复杂度为 $\Theta(\lambda_s(2^k)k)=\Theta(\lambda_s(n)\log n)$。

而对于查询，我们朴素地在每个块上二分查找到查询坐标所在的分段，时间复杂度为 $\Theta(\log^2 n)$。

2.2 分散层叠

通过 Fractional Cascading 的技术，我们可以优化询问的复杂度。

我们维护辅助数组 $T_k=L_k$，对于 $0\le j<k$，将 T_{j+1} 中所有 3 的倍数位置取出，与 L_j 归并得到 T_j。同时在 T_j 中记录每个元素的来源以及不同来源的前驱后继，便可以在 $\Theta(1)$ 时间内通过在 T_j 中的 lowerbound 找到在 L_j 和在 T_{j+1} 中的 lowerbound。因此查询的复杂度为 $\Theta(\log n)$。

由于 $|L_j|\le\lambda_s(2^j)$，且 $\lambda_s(n)=o(n^{1+\epsilon})$，我们可以得到 T_j 的长度不超过

$$\begin{aligned}|T_j|&\le\sum_{t=j}^{k}\frac{\lambda_s(2^t)}{3^{t-j}}\\&=\sum_{t=0}^{k-j}\Theta\left(\lambda_s\left(\frac{2^{t+j}}{3^t}\right)\right)\\&=\Theta\left(\lambda_s\left(\sum_{t=0}^{k-j}\frac{2^{t+j}}{3^t}\right)\right)\\&=\Theta(\lambda_s(3\cdot 2^j))\\&=\Theta(\lambda_s(2^j))\end{aligned}$$

当一次进位至第 j 位时，重构的复杂度为 $\sum_{t=0}^{j}\Theta(\lambda_s(2^t))=\Theta(\lambda_s(2^j))$，因此可以得到维护的总体复杂度仍然为 $\Theta(\lambda_s(n)\log n)$。

2.3 应用

2.3.1 维护分段一次函数最值

当该数据结构用于维护分段一次函数的时候，我们带入 $s=1$，可得维护的总复杂度为 $\Theta(\lambda_{1+2}(n)\log n)=\Theta(n\alpha(n)\log n)$。这在理论上优于以往李超线段树的 $\Theta(\log^2 n)$ 加入一条线段的复杂度。值得一提的是，存在使 n 条线段的上包络线分段数为 $\Theta(n\alpha(n))$ 级别的构造。[2]因此上述复杂度的分析是紧的。

2.3.2 动态规划问题

对于形如下式的动态规划问题：

$$a_i=\max_{0\le j<i} a_j+w_{j,i}$$

若 $a_j+w_{j,i}$ 能够改写为 $f_j(x_i)$ 的形式，且函数族 f_i 交替的阶数较低，则我们可以通过上述结构进行优化转移。

以往的这类问题中，通常需要将问题改写为一次函数的形式，或者依赖决策单调性解决。

事实上可以说明，这两种情况均是函数为 $s=1$ 阶交替情况下的特例。对于能够改写为一次函数的情况显然，对于决策单调性，通常我们要求 w 满足四边形不等式，从而可以得到

$$\begin{gathered}\forall i<j, w_{i,j+1}+w_{i+1,j}\le w_{i,j}+w_{i+1,j+1}\\ a_i+w_{i,j+1}+a_{i+1}+w_{i+1,j}\le a_i+w_{i,j}+a_{i+1}+w_{i+1,j+1}\\ f_i(j+1)+f_{i+1}(j)\le f_i(j)+f_{i+1}(j+1)\\ \Delta f_i\le\Delta f_{i+1}\\ 0\le\Delta(f_{i+1}-f_i)\end{gathered}$$

这说明对于可以通过四边形不等式进行决策单调性优化的问题，对于 $i\le i_0$ 的函数族 f 在 $j\ge i_0$ 的定义域上是 $s=1$ 阶交替的。

因此我们可以看到，上述数据结构在优化动态规划时能很好的兼容一些以往常见的条件。虽然在决策单调性问题上复杂度并不能达到最优，但上述数据结构却能处理性质略为复杂的转移代价函数。

[2]Ady Wiernik, Micha Sharir, *Planar realizations of nonlinear Davenport Schinzel sequences by segments*, 1988

3 询问点单调递增

设有 n 个函数 $f_i:\mathbb{R}\to\mathbb{R}$，对于 $x_1<x_2<\cdots<x_m$ 中的每个 x_k，求

$$\max_{j=1}^{n} f_j(x_k)$$

接下来我们讨论中假设处理的函数族是 s 阶交替的，且可以在常数时间内求出交替位置。

3.1 Kinetic Tournament 树

我们考虑一颗线段树，每个叶子节点表示了一个函数，线段树的每个节点存储了当前时间下子树中叶子节点的函数在当前 x 下取到最大值的那个函数。在 x 增加的过程中，我们需要不断修改某些节点的取值。我们维护 x 变大后子树里使得某个 max 所取函数第一次发生改变的 x 的值，当下一次询问超过这一 x 时，我们递归下去将发生替换的部分重置。维护这种信息的线段树我们称为 Kinetic Tournament 树。之后我们简称为 KTT。

考虑线段树中每个节点被修改次数，这等价于子树内所有函数包络线的分段数，因此所有节点修改次数总和与分治同理为 $\Theta(\lambda_s(n)\log n)$。因此直接在线段树上进行维护，由于每次修改需要走到该叶子，本算法的复杂度为 $\Theta(\lambda_s(n)\log^2 n)$。这比直接分治要显得逊色，但是我们将要看到这一方法在更强问题上的潜力。

3.2 带修改函数序列最值问题

本问题的带修改版本即允许在过程中在线将某个函数重新赋值。总共修改 m 次，询问 q 次。

我们考虑通过 KTT 来对此进行维护。在将一个函数重新赋值的时候，考虑将线段树中对应节点修改，并且更新到根节点的一串节点。

这样维护的复杂度是什么呢？我们考虑将维护过程进行如下等价转化：每个叶子节点假设被修改了 k_i 次，我们将其转化为这个节点下面放置 k_i+1 个节点，第一个节点是初始函数，后面接着是剩下的 k_i 个函数，它们都假设是在其存在时间上的分段函数。那么一个管辖区间为 $[l,r]$ 的节点子树里的上包络线分段数不会超过 $\lambda_{s+2}(r-l+1+\sum_{i=l}^{r}k_i)$，因此同一层的包络线分段总和不会超过

$$\begin{aligned}\sum_{[l,r]}\lambda_{s+2}\left(r-l+1+\sum_{i=l}^{r}k_i\right)&\le\lambda_{s+2}\left(\sum_{[l,r]}\left(r-l+1+\sum_{i=l}^{r}k_i\right)\right)\\&=\lambda_{s+2}(n+m)\end{aligned}$$

经过以上分析，可知对于在线修改的最值查询中，KTT 可以做到 $\Theta(\lambda_{s+2}(n+m)\log^2 n+q)$ 的时间内完成计算。

我们将看到这一结构如何可以用于优化一个经典的有关一次函数的问题。

4 线性情况的扩展

由于线性函数的性质良好，我们可以在其上研究一些扩展问题，遗憾的是笔者暂时未能确认在更高阶情况下的类似问题维护是否有相近的复杂度。

4.1 包含两类区间修改的序列最值问题

对于数列 k_i, b_i，我们有如下两种修改操作：

- 给定 l, r, x，对于 $l \le i \le r$，使 $b_i \leftarrow k_i x + b_i$。
- 给定 l, r, c, k', b'，对于 $l \le i \le r$，使 $k_i \leftarrow ck_i + k', b_i \leftarrow cb_i + b'$。

以及进行询问一段区间上 b_i 的最值。

本问题的做法改进自[3]。

这一问题在一部分 OI 题中有所部分出现，而这些问题在之前基本都只给出了 $\Theta(n + (m+q)\sqrt{n})$ 复杂度的基于分块的做法，而我们将看到通过 KTT，我们可以在 $O(n \log^2 n + m \log^3 n + q \log n)$ 时间内完成操作和询问，需增加限制：**第一种修改操作中保证 $\mathbf{x} > \mathbf{0}$**，且为了叙述主要思想，在这里仅考虑 $c > 0$ 的情况。

具体的做法思路是简单的：考虑在线段树上记录惰性标记，表示操作 2 的累计变化效果以及操作 1 的 x 的累计。我们注意到操作 1 本质上就是对 KTT 上的某个节点的子树开始进行“x 增大”的操作，而不是像原始的 KTT 上只从根节点进行。

4.1.1 复杂度分析

我们考虑通过势能来分析这一算法的复杂度。

我们定义一个非叶子节点组成的集合 $\mathcal{P}$，对于一个非叶子节点 v，如果 v 在当前保留的取值较大的孩子所对应的直线斜率是严格小于另一个孩子的，那么 $v \in \mathcal{P}$。

我们定义势函数 $\Phi = \sum_{v \in \mathcal{P}} d(v)$，其中 $d(v)$ 是节点 v 在线段树上的深度，根节点的深度为 1。

我们考虑 1 操作的最坏情况，即每个节点更新操作是单次进行的，那么我们定义一次更新操作的代价为 1，一次更新操作的均摊代价是 $1 + \Delta\Phi$，其中对于被修改的节点 v，可以

[3] Daniel Zhang, https://codeforces.com/blog/entry/68534?#comment-530381, 2019

得知 v 必然从在 $\mathcal{P}$ 中变为不在，而 v 的父节点 p 有可能从不在 $\mathcal{P}$ 中变成在 $\mathcal{P}$ 中。故

$$\begin{aligned}\widehat{c} &= 1 + \Delta\Phi \\ &\le 1 - d(v) + d(p) \\ &= 1 - d(v) + (d(v) - 1) \\ &= 0\end{aligned}$$

我们这一定义使得原本无法确认进行次数的更新操作在均摊代价中不需要被考虑。接下来考虑操作 1 和操作 2，它们其实本质上是类似的。注意到当一个节点的惰性标记被修改或以其为子树的数被更新时，这一节点的父节点可能由不在 $\mathcal{P}$ 中变为在 $\mathcal{P}$ 中，这最多导致势能增加 $d(p) = \Theta(\log n)$。而一次操作会影响 $\Theta(\log n)$ 个节点，因此一次操作导致势能增加最多 $\Theta(\log^2 n)$。

还有一点需要注意的是由于 $\sum \widehat{c_i} = \sum c_i + \Phi_t - \Phi_s$，因此我们给总共的代价加上 $\Phi_s - \Phi_t = \Theta(n \log n)$，这是因为最坏情况是最初所有节点均在 $\mathcal{P}$ 中。

综上，操作过程中最多会引发 $\Theta(n \log n + m \log^2 n)$ 次更新操作，因此复杂度有上界 $O(n \log^2 n + m \log^3 n + q \log n)$。

4.1.2 特殊情况

在诸如“区间增加公差为正的等差数列”中，我们提炼出一个隐含条件：序列的 k 是单调递增的。因此引发的更新操作必然是一个节点所取到的 max 由在左子切换为在右子。

此时我们定义势能 Φ 为“max 位于左子”的节点数。由于操作过程最多引发 $\Theta(n + m \log n)$ 次更新操作。我们有复杂度上界 $O(n \log n + m \log^2 n + q \log n)$。

4.2 例题

例题 1. *Bear and Bowling*[4]

这里仅讨论与本文相关的一种解法，在该解法中我们需要按照一个顺序在一个序列 a_n 中每次拿走一个数，设该数左边已经被拿走的数有 $k_i - 1$ 个，右边已经被拿走的数的总和为 s_i，那么每次拿的数必须是 $k_i \cdot a_i + s_i$ 中最大的。[5]

如果我们使用上述的数据结构，那么只需最初令 $b_i = a_i$，每次拿走一个数后我们将 b_i 加上 $-\infty$，给 $[1, i-1]$ 这一段加上 a_i，给 $[i+1, n]$ 这一段执行 $b_j \leftarrow b_j + a_j$ 即可。

例题 2. *Innophone*[6]

[4]Codeforces 573E，https://codeforces.com/problemset/problem/573/E

[5]徐翊轩, *IOI2020 中国国家集训队第一阶段作业试题准备*, 2019

[6]ROI 2018，https://loj.ac/problem/2845

给若干个 x_i, y_i，选取 a, b 最大化

$$\sum_{i=1}^{n}[a \le x_i]a + [a > x_i][b \le y_i]b$$

我们按照 y 排序，扫描过程中在 KTT 上区间增加斜率，按照对应的 y 值查询就可以了。

4.2.1 最大连续子段和

题目大意 对于一个数列 a_n，支持将区间上整体加一个正数，询问一个区间上的最大子段和。

解法 这是一个较为复杂的问题，我们首先回顾没有修改操作的做法。我们在线段树的每个节点上可以维护四个值 $sum, lmax, rmax, totmax$，即可得到维护方法

$$\begin{aligned} sum &= ls.sum + rs.sum \\ lmax &= \max(ls.lmax, ls.sum + rs.lmax) \\ rmax &= \max(rs.rmax, rs.sum + ls.rmax) \\ totmax &= \max(ls.totmax, rs.totmax, ls.rmax + rs.lmax) \end{aligned}$$

我们考虑将现在维护的这 4 个信息均表示为一个一次函数，那么每个节点维护的击败时的 x 即上式中所有 max 所取函数发生变更时间的最小值。

我们记一个包含 max 比较的变量的 rank 值 $r(v, para)$：

- 对于 $r(v, lmax)$ 或 $r(v, rmax)$：对于该变量的决定式 $f(x) = \max(a(x), b(x))$，rank 值表示当前 a, b 中斜率大于 f 的斜率的直线的数量乘以 $d(v)^2$。
- 对于 $r(v, totmax)$：对于该变量的决定式 $f(x) = \max(a(x), b(x), c(x))$，rank 值表示当前 a, b, c 中斜率大于 f 的斜率的直线的数量乘以 $d(v)$。

我们直接定义势能 $\Phi = \sum_v \sum_{para} r(v, para)$。

- 当发生一次 $lmax$ 或 $rmax$ 的击败，有：

$$\Delta\Phi \le 1 - d^2 + (d-1)^2 + d - 1 = 1 - d \le 0$$

- 当发生一次 $totmax$ 的击败，有：

$$\Delta\Phi \le 1 - d + d - 1 = 0$$

可以得到本算法的一个复杂度上界 $\Theta(n\log^3 n + m\log^4 n + q\log n)$，其中 m 为修改次数，q 为询问次数。

但类比之前对于斜率单调的特殊情况所进行的分析，我们考虑每个节点的 *lmax* 和 *rmax* 的决策发生变化的次数。注意到 *lmax* 的决策点只会递增，而在一个节点发生斜率切换的必要条件是决策位置从 $[l, mid]$ 切换到了 $[mid+1, r]$。每个节点最多触发一次，故 $lmax, rmax$ 的总共额外复杂度是 $\Theta(n\log n)$。因此我们现在知道在这个问题上 KTT 维护 *lmax* 和 *rmax* 的总共 dfs 到的节点数是 $\Theta((n+q)\log n)$ 的。我们沿用 *totmax* 的势能分析，复杂度是 $O((n+m)\log^3 n + q\log n)$。

但这其实并没有说明之前的势能分析完全冗余，刚刚所作的 $\log^4 n$ 的分析可以认为是在考虑一个更加一般性的操作，即加法是令 $a_i \leftarrow a_i + k_i x$，而原问题中隐含了 $k_i = 1$，即 $k_i > 0$。

4.2.2 henry_y 的数列

题目大意 给一个长为 n 的数列 A_i，支持修改操作：

给定 l, r, a, b, c，对于 $l \le i \le r$，使 $A_i \leftarrow A_i + ai^2 + bi + c$，保证 $a, b \ge 0$，以及询问一段区间上 A_i 的最小值。[7]

解法 这一问题中看似有二次项，但二次项中的 i^2 是只与下标相关的，因此实际上更接近于维护若干个二元一次函数的最值。

首先还是类似 KTT 的思想，考虑用线段树先维护当前最值，以及加了 (ai^2+bi) 到什么时候会使得子树的某个比较关系发生变更。由于 $a, b \ge 0$ 且 $i \ge 1$，所以最小值只可能从右子树切换到左子树。

假设左子树的最值在 A_i 处，右子树的最值在 A_j 处，若 $A_i > A_j$，则有可能未来 A_j 反超 A_i，这需要

$$\begin{aligned} A_i + ai^2 + bi &\le A_j + aj^2 + bj \\ A_i - A_j &\le a(j^2 - i^2) + b(j - i) \\ \frac{A_i - A_j}{j - i} &\le a(i + j) + b \end{aligned}$$

由此可知，当 (a, b) 进入一个半平面时，子树的最值发生切换。反过来说，若在该半平面的补集中，则还未发生最值切换。因此若当前的 (a, b) 修改不发生子树的切换，则需要该点落在子树的所有限制之内，即所有半平面的交集之内。由于半平面的直线斜率为 $i+j$，所以左子树中的所有斜率均小于右子树中的所有斜率，我们可以使用可持久化平衡树来维护半平面交，并且在平衡树上进行二分从而在 $\Theta(\log n)$ 的时间得到两个半平面交合并的结果。

[7]改自「hyOI2020」，https://loj.ac/problem/6727

由于本问题具有与前文问题相似的单调性，我们可以设势能 Φ 为“min 位于右子”的节点数。由于操作过程最多引发 $\Theta(n + m\log n)$ 次更新操作，每次操作消耗 $\Theta(\log^2 n)$ 复杂度用于更新信息。我们有复杂度上界 $O((n + m\log n)\log^2 n + q\log n)$。

5 总结

本文主要围绕 $DS(n, s)$ 序列本身长度的一个优秀上界，通过“减少维护的必要交点数”的方法进行设计，得到了一些复杂度比较优秀的方法用于维护函数的最值。希望本文上述的一些思路可以对 OI 中维护最值有关的问题产生更多的启发。

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。
感谢戴江齐同学和刘承奥学长与我讨论以及给予的启发。
感谢房励行同学为本文刊误。

参考文献

[1] Micha Sharir, Pankaj K. Agarwal, *Davenport-Schinzel Sequences and their Geometric Applications*, 1995.

[2] Seth Pettie, *Sharp Bounds on Davenport-Schinzel Sequences of Every Order*, 2015.

[3] Ady Wiernik, Micha Sharir, *Planar realizations of nonlinear Davenport Schinzel sequences by segments*, 1988.

[4] 徐翊轩, *IOI2020* 中国国家集训队第一阶段作业试题准备, 2019.

[5] Daniel Zhang, https://codeforces.com/blog/entry/68534?#comment-530381, 2019.

《图上的游戏》命题报告

华东师范大学第二附属中学 左骏驰

摘 要

本文将介绍作者在某次省选训练时为几个学校命制的一道交互题。题目深度挖掘了有关图的连通性、图的生成树、有根树的 DFS 序的知识，并有机地结合了二分查找、分治的基本算法，以及随机化的思想。本题所用的知识点并不高级，但需要选手对基本算法蕴含的重要思想有深刻的理解。

此外，本题的部分分很多，能引导选手走向正解，且区分出拥有不同思考程度的选手。正解大概分为三个部分，而想到每一部分都会有相应的分数。

本文提出一种命题上的目标：部分分的设计与正解高度契合。

1 试题大意及数据范围

1.1 题目描述

小 Z 有一张 n 个点 m 条边的可能有重边，但没有自环的无向图。小 U 想要知道这张图的样子，但小 Z 不肯告诉它。

小 Z：我可以告诉你这张图是连通的，点从 0 标号至 $n-1$，边从 0 标号至 $m-1$。

小 U：好那删去一条边后，这张图的连通状况如何呢？

小 Z：反正你也猜不出来，就这样吧。你每次告诉我一个边的编号的集合 S，再给定一个点 u，然后我可以帮你计算把编号在 S 中的边连接后，0 号点与点 u 是否连通。

听到小 Z 的这句话后，小 U 苦思冥想，仍然不知道如何才能问出这张图的每条边到底连接哪两个端点。

因此他找到了你，想让你帮他问出这张图的信息。我们保证这张图是预先生成的，也就是说不会随小 U 的询问而改变。并且你不能询问太多次，你需要在 30000 次内得到这张图的信息！

1.2 交互方式

你不需要实现主函数，你只需实现一个函数：bool query(int u, std::vector<int> s);

这个函数应该返回一个长度为 m 的数组 e，其中 $e[i]$ 表示第 i 条边连接的两个端点（顺序无关）。

你所实现的这个函数只能调用交互库中的函数：bool query(int u, std::vector<int> s);

这个函数中 u 是一个 $[0, n-1]$ 中的整数，s 是一个长度为 n 的 int 数组，且值只能为 0 或 1。若 $s[i]=1$，则表示 i 是否在小 U 所询问的集合 S 中, 否则表示不在集合 S 中。这个函数的意义是给定 u 和 S，询问 0 号点能否通过编号在 S 中的边到达 u。

1.3 题目数据范围及特殊限制

对于所有数据，保证 $1 \le n, m \le 600$，图是连通图。

Subtask 1(6 pts): $n, m \le 25$，$m = n-1$。

Subtask 2(10 pts): $n, m \le 25$。

Subtask 3(17 pts): $m = n-1$，且图中每个顶点的度数 ≤ 2，0 号点的度数为 1。

Subtask 4(24 pts): $m = n-1$。

Subtask 5(19 pts): 保证将编号 $< n-1$ 的边提出后，编号为 i 的边所连接的两个顶点为 i 和 $i+1$。

Subtask 6(13 pts): 保证将编号 $< n-1$ 的边提出后，图构成一棵树。

Subtask 7(11 pts): 无特殊限制。

2 前置技能

2.1 图的基本概念及术语

对于一个无向图 G，如果任意两个顶点之间均存在一条路径连接这两个顶点，就称该图为**连通图**。

若无向图 G 满足任意两个顶点之间有唯一的一条路径与它相连，则称 G 是一棵**树**（或称无根树）。

对于一般的无向连通图 G，若树 T 的点集与 G 相同且边集是它的子集，则称 T 是 G 的**生成树**。

给一棵无根树规定了一个根后，就形成了一棵有根树。每个非根顶点 u 到根的路径上第一个（不含起点）的顶点称为 u 的**父亲**。如果 v 为 u 的父亲，则称 u 是 v 的**儿子**。如果某个顶点不存在儿子，则称这个顶点为**叶子**。我们再把所有根到顶点 u 路径上的点成为 u 的**祖先**。对于一个特定的顶点 u，我们把去掉 u 到它父亲的边（如果存在的话），u 能到达的点的集合称为 u 的**子树**。

此外，如果某棵有根树 T 满足 T 的每个顶点的儿子个数 ≤ 1，则称 T 为**链**。

图 1 是一个有根树的例子，其中 0 有两个儿子 1, 2，1 有两个儿子 3, 4，2 有 5 这一个儿子，且 3, 4, 5 没有儿子，它们是叶子。

图 2 表示的是链。其中 0, 1, 2, 3, 4 恰好有 1 个儿子，5 是叶子。

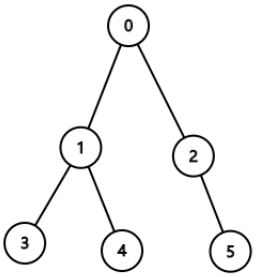


图 1: 以 0 为根的有根树的一个例子

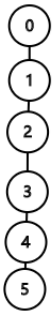


图 2: 以 0 为根的链的一个例子

2.2 树的 dfs 序

对于一棵从 0 开始标号的有根树 T，我们有一个常见的 dfs 算法如下：

```
DFS(u)
1   PUSH(Ans, u)
2   for v in child(u)
3       DFS(v)
```

伪代码中 $child(u)$ 表示 u 的儿子的某个序列。

运行该 dfs 算法后，Ans 记录了 dfs 过程中构成的顶点序列。我们记 u 在 dfs 序中的位置（从 0 开始标号）为 id_u，那么我们有如下性质：

性质 1: 若 v 是 u 的祖先，则 $id_u \geq id_v$。

证明: 在 dfs 访问过程中，若访问了某个顶点，则其父亲也必定被访问，故 $id_u \geq id_v$。

2.3 一个基本方法

我们来考虑这样一个简单的问题：$1, 2, \cdots, N$ 中有 M 个数是好的（这这里我们认为 M 比较小，N 比较大）。现在你可以指定一些数，询问在这些数里面是否有好数，你需要用 $O(M \log N)$ 次操作来找到所有好数。

这一思想在这一交互题中用到了很多次，也在 OI 比赛很多的交互题上都有用。

我们采用分治的思想。伪代码如下：（这里 $\text{Ask}(S)$ 的返回值表示 S 中是否存在好数，Ans 最终会得到所有好数的集合。）

SOLVE($left, right$)

```
1  mid ← (left + right)/2
2  if left == right
3      Ans ← Ans ∪ {mid}
4  else
5      S_1 ← {left, left + 1, ··· , mid}
6      S_2 ← {mid + 1, mid + 2, ··· , right}
7      if Ask(S_1)
8          SOLVE(left, mid)
9      if Ask(S_2)
10         SOLVE(mid + 1, right)
```

考虑这个算法所构成的分治树的结构。它的叶子的区间长度为 1，且这个区间内的数必定为好数。每个叶子的深度为 $O(\log N)$，故分治树总节点数是 $O(M \log N)$ 的。这样询问的次数也是 $O(M \log N)$ 的。

3 算法介绍

我们将所要介绍的算法将顺着部分分的提示逐步深入，因此我们将按照链、树、连通图的顺序介绍。

3.1 链

这个部分分有很多种 $O(n \log n)$ 做法，基本都借鉴了随机化的思想。

这里我们给出一种做法。

首先将点 $\{1, 2, \cdots, n-1\}$ 随机排列为 $p_1, p_2, \cdots, p_{n-1}$。然后依次加入每个点 p_i，我们需要维护：

1. 已经加入的点之间的相对顺序

2. 对于每条边，在它的左侧（就是靠近 0 号点的一侧）的所有点的集合。

（初始时我们认为 0 号点在最左侧，且我们加入一个在最右侧的虚拟节点 n 号点）

那么我们对于新加入的每个点 p_i，可以先在原序列上二分出来它要插入到已加入顶点的哪个位置。这一步具体可以保留在某个点左侧的边，然后判断 0 能否到达新加入的点，来缩小插入到的顶点的范围。

接下来我们在插入的两个点 u_l, u_r 之间的边中，暴力地对每一条边 e 判断它在 p_i 的左边还是右边。这一步我们可以通过删去这条边保留其它所有边，判断 0 是否与 p_i 连通来实现。

接下来我们来大致说明其复杂度为 $O(n \log n)$。

由于每次二分的复杂度已经是 $O(n\log n)$ 了，只需研究暴力判断每一条边的位置的复杂度。假设已经加了 i 个点，那么在随机意义下需要花费的代价是 $O(\frac{n}{i})$ 的！而 $\sum_{i=1}^{n+2}\frac{1}{i}=O(\ln n)$，故 $\sum_{i=1}^{n+2}\frac{n}{i}=O(n\ln n)$。得到总复杂度为 $O(n\log n)$。

于是我们在 $O(n\log n)$ 的询问次数内解决了这个子问题。

3.2 树

以下均把树看成是以 0 为根的有根树。

3.2.1 $O(n^2)$ 做法

对于每条边 e_i 我们把这条边删去，保留其它所有边。然后逐次询问 0 号点能够到哪些点。那些不能到达的点就构成了某个点的子树。

反过来，我们也可以直到每个点到它的根上的所有边，这样也就知道了每个点的深度。

然后对每条边，我们找删去它，根不能到达的点中深度最小的点，这就是**子树的根**, 也是**这条边上的较深**的端点。

之后对每个点 u，我们来找包含它的子树大小最小的子树（不包括 u 本身的子树），就可以得到它的父亲了。

最后，我们就可以从每条边的一个较深端点推到两个端点，就解决了这一问题。

而我们一共要删去 $n-1$ 条边，每条边询问 $n-1$ 次，因此总复杂度为 $O(n^2)$。

3.2.2 将边分割为链的做法

我们考虑把每条点标一个数 $0,1,\cdots,c-1$，使得标同一种数的点构成了一条链，且每条链的顶端的父亲所在的链的标号比它小。一开始所有树上的边都是未标记的。

图 3: 划分方法

图 3 是正确的划分方法的一个例子。其中红所在链的每条边的较深端点和根标号为 0，橙色链标号为 1，黄色链标号为 2，绿色链标号为 3，蓝色链标号为 4，紫色链标号为 5。对这种分割方法的直观理解是：后面的链都“接在”前面某条链之下。这使得我们之后只需要对每一条链分别求解，再找到每条链和之前部分的“拼接点”!

固定一个顶点 u，定义所有 0 到 u 路径上的**未标记的边为好边**。如果一次题目中询问的集合的补集包含了**好边**, 则 0 到 u 就不连通了。

依次考虑 $1, 2, \cdots, n-1$ 这些点。

情形一：只保留已经标记的边，0 就与 u 连通。这说明 u 本身在原先的某条链上，那么我们可以保留两个端点所在链的编号 $\leq mid$ 的边，询问 0 是否与 u 连通，通过二分可以知道 u 所在的链的编号了。

情形二：只保留已经标记的边，0 不与 u 连通，这个时候我们找到**所有未标记的好边**。这一步可以用 2.3 中的那个分治的方法来实现。最后我们让这些边所在的较深的点变为一条编号为 c 的新的链，然后令新的 c 为 $c+1$。

注意到碰到情形二的时候，你可以知道新开的一条链的点连向它父亲的边的编号的集合，但是你不知道这条链上有哪些点。然而你可以在之后通过情形一的做法来确定这条链上的所有点。

因此在最后，我们可以知道每条链上有哪些点，还知道这些点连向它父亲的编号的集合。

3.2.3 分割为链之后的方法

如果知道了这些，我们可以按照标号从小到大的顺序依次确定每条链的顺序，每条链上的每个点连到它父亲的边的编号，这一步直接使用链的部分分做法即可。所以我们只需要知道每条链的顶端的父亲的点的编号。

这一步可以使用在 dfs 序上二分的做法。具体来说就是假设你当前加入的这条链标号为 i，所有标号 $< i$ 的点构成的树 T 是已知的，只是从 $i-1$ 到 i 的这条边未知。然后你对 T 保留端点的 dfs 序 $\leq mid$ 的边，再保留这条链顶端到它父亲的边的编号，询问保留这些边能否到达链顶端的点，就可以知道这条边的父亲的 dfs 序是否 $\leq mid$。

注意到这里我们用到了 2.2 的性质一，这一性质保证了把两个端点的 dfs 序都 $\leq mid$ 得边保留后，得到的子图连通了所有 dfs 序 $\leq mid$ 的顶点。

3.3 连通图

3.3.1 找生成树的 $O(nm)$ 算法

我们只需找出一个边的下标的集合，使得这些下标的边构成了一棵生成树。这样就可以套用找树的所有方法了。

一开始，所有边都是未标记状态。然后依次加入 $1,2,\cdots,n-1$ 号顶点。假设加入到了顶点 u，且之前标记的边连通了 0 和 $1,2,\cdots,u-1$。

性质一: 依次扫描所有未标记的边，若这条边删去后 0 就不能到达 u，就把它变为标记的边，否则就跳过这条边。则我们这样得到的新的标记的边能够连通 0 和 u，且被标记的边仍然构成森林。

证明: 由于一开始图是连通的，显然 0 能够到达 u，且我们考虑依次扫描所有未标记的边，跳过一条边并不影响 0 到 u 之前的连通性。

再证明得到的新的被标记的边构成的子图无圈。反证法，假设存在一个圈 C，它由 $e_{i_1},e_{i_2},\cdots,e_{i_l}$ 构成，其中 $i_1<i_2<\cdots<i_k$ 为它的标记时间。则在标记 i_1 的时候，i_1 的两个端点由 $e_{i_2},e_{i_3},\cdots,e_{i_k}$ 这些边连通了。因此删去 e_{i_1} 不影响连通性，也就是说 e_{i_1} 不是桥，故 e_{i_1} 应该跳过，而不是被标记，矛盾！

通过性质一，我们就可以每一轮 $O(m)$ 地把连接 $0,1,\cdots,u-1$ 的森林扩充为 $0,1,\cdots,u$ 的生成森林，进而在 $O(nm)$ 的询问次数解决了找生成树的方法。

3.3.2 找生成树的 $O(n\log m)$ 算法

注意到最后我们只需要标记 $n-1$ 条边，我们不妨把前一个算法的依次扫描的过程改成**每次找一个最大的前缀删去，使得 0 仍然能够到达 u**。这一过程可以用简单的二分查找来实现。

这样依次二分下一条需要标记而不是被删除的边，一共进行 n 轮二分操作，每一轮二分操作除了最后一次二分之外，其余的二分操作都会使得我们多标记了一条边，所以最多 $n+n-1=O(n)$ 次二分操作。总时间复杂度 $O(n\log m)$。

3.3.3 找非树边的方法的 $O(nm)$ 算法

首先套用树的算法，可以找出其生成树 T 的每一条边连接的两个点了。

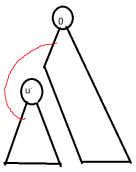


图 4: 其中红边代表我们未知的一条非树边

对于每条树边 (u,v), 设 u 是其中较深的顶点。假设我们要询问的边是 $e_i=(w_1,w_2)$，其

中 w_1, w_2 未知。我们把 (u, v) 删去，保留其余所有边，再保留 e_i，询问 0 是否和 u 连通。这样若答案为连通，就说明 w_1, w_2 中恰有一个在 u 的子树内（这种情况如图 4 所示），也就是说 $w_1 \to w_2$ 在 T 上的路径经过了 (u, v)。

对每条非树边，我们可以依次得到每条树边是否在它“跨过”的路径上。这样就可以从路径的端点推出 w_1, w_2 了！

3.3.4 找非树边的 $O(m \log m)$ 算法一

这个部分可以自顶向下做，也可以自底向上做。由于自底向上常数稍小，易于理解，我们就先写这种做法了。

考虑删去一个生成树的叶子 u，在删去的同时你需要知道与 u 相连的所有边，在把所有点删去后所有非树边就可以得到了。

第一步：我们称和 u 相邻的边为好边。把 T 上和 u 相邻的唯一一条边删去，如果某次询问包含了所有 T 的其它 $n-2$ 条边和一条好边，0 就与 u 连通。如果只包含 T 的其它 $n-2$ 条边，而没有包含好边，就不与 u 连通。我们再次套用之前找好数的方法，即可找到所有好边的**下标**。

第二步：对每条非树边找到另一个端点。这一步可以用之前说过的在 dfs 序上二分的思路去实现。具体来说，保留端点 dfs 序 $\le mid$ 的顶点，和你要考虑的那条非树边 e_i，如果 0 此时能够到达 u，说明 e_i 的另外一个端点 dfs 序 $\le mid$，否则就 $> mid$。这样我们就可以二分出这条连接叶子的非树边的另外一个端点了。

最终，我们用 O(m log m) 的询问次数解决了此题。

3.4 找非树边的 $O(m \log m)$ 算法二

我们按照任意的顺序考虑每条树边。如果我们考虑到了一条树边 (u, v)（设 v 是 u 的父亲），删除后形成了两个连通块 C_1, C_2，则我们要找到所有跨过 C_1, C_2 的非树边。

初始时将所有非树边标记为“未知的”。每次删去**恰好** (u, v) **这一条边**，保留其它的 $n-2$ 条树边，若我们再加上的非树边中存在好边，0 就和 u 连通，否则 0 就和 u 不连通。因此我们又可以用前文所述的分治的方法找到所有**未知的好边**。把这些好边标记为**已知的**，再之后不必考虑它们。

接下来对于某条找到的非树边 e，我们再把 C_1 这棵子树内的顶点按 dfs 序标号，然后保留端点 dfs 序 $\le mid$ 的边，保留 C_2 内所有树边，保留 e，询问 0 是否可以到达 u，若可以，则说明在 C_1 的端点的其 dfs 序 $\le mid$，否则就 $> mid$。同样地，我们对连通块 C_2 也做一样的事情，就可以知道这条非树边 e 的两个端点了。

在考虑完 $n-1$ 条边后，我们即可得到所有的树边了（因为每条非树边都必定“跨过”至少一条树边）。

注意到其实算法一是算法二在特殊的考虑边的顺序下变化出的算法。如果我们把算法二中考虑边的顺序变成：每次逐渐剥去一个叶子与它父亲相连的边，就相当于算法一了。然而剥叶子的时候，我们每次“好边”的一个端点是固定的，可以把两次 dfs 序上二分改成一次 dfs 序上二分，因此减少了常数。

4 命题过程

这一问题是笔者在试图命制一道树上的交互题的时候得到的。一开始笔者思考树上子图连通块的性质，得到了该题仅有树的部分分的版本。之后笔者通过思考，得到了化树为链的巧妙做法，再解决“链”这一问题。

但我认为这个优美的问题应该还可以推广，于是我开始思考在一般的连通图上问题还能不能高效地解决。我采取了同样的化一般为特殊的方法，找到了一个找到一棵生成树的方法。然后笔者再思考能否把树边作为工具来找到所有的非树边，就得到了一个完整的做法。

在设计部分分的时候，为了提高区分度，笔者尽量保留了我在思考此题的时候中间走过的路线。假设正解需要有 A, B, C 三个难点，我认为设计部分分的更好的模式是例如：打出暴力给 20 分，想到 A,B,C 中的一个多给 20 分，想到两个多给 40 分，想到三个多给 60 分，能够把 A, B, C 结合起来得到正解，给 100 分。

而现有的很多给部分分的方法是：想到 B 之前必须要先想到 A，想到 C 之前必须要先想到 A, B，才能得到更多的分数。这种模式使得想到 B 而未想到 A 的选手和暴力选手得同一个分数，可能增加了比赛的随机性，减少了比赛的区分度。所以，我认为在 OI 题目命制的部分分设计方面，我们有很大的改善空间。

5 得分情况

5.1 预计得分情况

前面两个 subtask 需要选手理解题目意思，对题目做出简单思考，以得到一个多项式的算法，预计有 70% 的选手能够得到。Subtask 3 做法繁多，且并没有很难想到的地方，预计 60% 的选手能够得到这一分数。Subtask 4,5,6 是正解在不同方面的特殊情况，预计 50% 的选手能够拿到恰好一个子任务的分数，且 30% 的选手能够拿到 2 个子任务的分数。而拿到 70 – 100 分的分数需要将不同的做法有机地结合起来，有很大的思维难度，因此估计不超过 5% 的选手能够 AC 此题。

5.2 实际得分情况

在所有有分的选手中，1 人获得 71 分，1 人获得 52 分，1 人获得 47 分，4 人获得 33 分，2 人获得 17 分，4 人获得 16 分，3 人获得 6 分。可见实际得分情况不好于我的预计得分情况。

从选手角度，这表明选手需要加强对于交互题的训练，掌握一些交互题的基本套路，例如我在 2.3 中所提到的思想方法，同时提升面对复杂、多步骤问题的处理能力。从出题人角度，我觉得我在今后的比赛中应该更加加强部分分的提示作用和引领作用，以提升区分度。

6 总结

从 WC2016,WC2018,WC2019,NOI2019 这四场比赛看，我们可以发现出题人们再试图提高 OI 中交互题的比重。因此，我选了交互题的命题报告这，作为我的集训队论文的主题。此外，同时兼任选手和出题人的我，不免对出题的过程有自己的理解，因此我在本文还融入了对命题过程的独特的思考，希望读者能够有所启发。

总之，本文介绍了本人命制的一道交互题。在介绍这道交互题的过程中，本文先介绍了图论的基本知识和交互题的基本思想方法，再深入分析了此题的各种算法。

最后，本文从这道题目引申开来，介绍了我们命题里面设计部分分环节的自己的想法，也展望了未来“交互题”这一领域在 OI 中的发展。

希望本文能让大家不仅学会了这一道题的解法，也领悟了交互题的基本思想。同时希望在本文发出之后，OI 赛场上能够出现更多的交互题，能够有更加科学的部分分的设计。

7 致谢

感谢 CCF 给了我宝贵的分享交流的平台机会。

感谢父母对我的养育之恩。

感谢华东师范大学第二附属中学的教练金靖对我的指导。

感谢国家集训队教练的辛勤付出。

感谢张博为同学和我讨论过此题的多种解法，特别是确定非树边的 $O(n \log n)$ 的第一个算法，也给对写论文给予了很大的帮助。感谢吕时清同学、陈思远同学教我 latex 的使用方法，帮我做论文的排版。

感谢所有对我有所帮助的同学。

参考文献

[1] https://csacademy.com/app/graph_editor/.

[2] https://zh.wikipedia.wikimirror.org/wiki/%E5%9B%BE_(%E6%95%B0%E5%AD%A6).

[3] https://zh.wikipedia.wikimirror.org/wiki/%E6%A0%91%E7%8A%B6%E5%9B%BE.

[4] https://zh.wikipedia.wikimirror.org/wiki/%E6%A0%91_(%E5%9B%BE%E8%AE%BA).

[5] https://max.book118.com/html/2019/0810/6243233210002053.shtm（《论偏题的危害》，王天懿）.

[6] http://vfleaking.blog.uoj.ac/blog/909.

不一般的 DFT

绍兴市第一中学 周雨扬

摘 要

快速傅里叶变换是信息学奥赛中的一种非常有用的算法。本文介绍了快速傅里叶变换的一些比较巧妙的应用。本文介绍了 bluestein 算法，并且在模数较小时将其应用到多项式多点求值和多项式多点插值上，同时也尝试利用 bluestein 算法解决一些比较有代表性的关于多项式的问题。

1 引言

快速傅里叶变换是一种非常有用的算法。利用快速傅里叶变换我们可以在 $O(n\log n)$ 的时间复杂度内计算出两个多项式的乘积。同时可以利用多项式乘法解决很多种关于多项式的问题，例如多项式求逆，多项式 exp 等等。

但是存在一些特殊的问题，他们虽然可以使用快速傅里叶变换解决，但是需要一些巧妙的转化。本文集中对某一类特殊的问题的做法进行了归纳与总结，并且尝试将其应用到更加常见的问题上。

本文主要分为三个部分，第一部分介绍了一些关于傅里叶变换的概念。第二部分从几个例题入手，介绍了 bluestein 算法的推导思路以及其较为基础的应用。第三部分围绕多项式多点求值和多项式多点插值，通过两个经典算法问题展示了 bluestein 算法在模数较小时的用途，并且尝试将这种做法推广到更加一般的问题上。

2 相关定义和说明

2.1 相关约定

在本文中，我们采用符号 $|f(x)|$ 表示多项式 $f(x)$ 的次数。这里我们约定任意非零次多项式的最高项系数非 0。

在本文中，我们定义多项式 $f(x)$ 在 p 处的点值为将 p 代入 $f(x)$ 中得到的结果。

在本文中，我们采用符号 ω_n 表示 n 次单位根，也就是 $\cos\left(\frac{2\pi}{n}\right)+\sin\left(\frac{2\pi}{n}\right)\mathrm{i}$。

2.2 离散傅里叶变换 (DFT)

离散傅里叶变换是将 $\omega_n^0, \omega_n^1, \cdots, \omega_n^{n-1}$ 依次代入多项式 $f(x)$ 中得到的长度为 n 的点值序列 a。在本文中，我们称离散傅里叶变换中的参数 n 为模长。在信息竞赛中离散傅里叶变换一般需要保证 $n > |f(x)|$。

在已知长度为 n 点值序列 a 的情况下，我们构造多项式 $g(x) = \sum_{i=0}^{n-1} a_i x^i$，并且求出将 $\omega_n^0, \omega_n^{-1}, \cdots, \omega_n^{-(n-1)}$ 依次代入 $g(x)$ 中得到的长度为 n 的点值序列 b，则存在 $f(x) = \frac{1}{n}\sum_{i=0}^{n-1} b_i x^i$。证明可以见该博客[1]，此处略去。因此我们可以利用两个多项式在 $\omega_n^0, \omega_n^1, \cdots, \omega_n^{n-1}$ 的点值来快速计算两个多项式相乘的结果。

快速傅里叶变换 (FFT) 是一个基于分治的离散傅里叶变换的优化。单次运行时间复杂度为 $O(n \log n)$。在下文中会讲述其做法。

2.3 一般模数快速傅里叶变换 (MTT)

假设我们需要求解 $A(x) \times B(x)$ 的各项系数对 $p = 10^9 + 7$ 取模的结果。如果直接采用离散傅里叶变换求解，运行结果的绝对值会达到 $p^2 \times |A(x)|$ 级别，在精度上无法接受。

考虑将多项式的各项系数拆成 $xC + y(0 \le y < C)$ 的形式，其中 C 是一个正整数。据此我们可以将多项式 $A(x)$ 拆成 $A_1(x)C + A_2(x)$，其中 $A_2(x)$ 各项系数均为 $[0, c-1)$ 内的正整数。类似的对于 $B(x)$ 也拆成 $B_1(x)C + B_2(x)$。此时两个多项式乘积为 $A_1(x)B_1(x)C^2 + (A_1(x)B_2(x) + A_2(x)B_1(x))C + A_2(x)B_2(x)$。对于 C 次数不同的那些项，我们采用上面的方法直接求解。取 $C = \sqrt{p}$ 时, 运行多项式乘积产生的结果的绝对值只有 $p \times |A(x)|$，在精度上可以接受。

或者找到 3 个不同的质数，使得其满足 $p \times 2^k + 1(k \ge 22)$ 的形式，在求出原根后使用快速傅里叶变换分别求解，最后使用中国剩余定理合并答案。该算法在常数上较劣。

在毛啸学长 2016 年的集训队论文中有提到对于第一种算法的优化，有兴趣的读者可以自行查阅

3 一般模长 DFT

3.1 引入

例题一 [2]

题意

[1] 博文链接: https://www.cnblogs.com/RabbitHu/p/FFT.html

[2] Source:【北大集训 2019】

有一个无限循环的序列 $x_1, x_2, \cdots$, 循环节为 n。对这个序列进行 1 次操作后，操作后的序列 x' 循环节仍为 n，且满足 $x'_i = x_i + x_{i+1}$。

有 q 次询问，每次给定 n, m, k, i, p，求当序列的循环节为 n，初始只有 $x_i = v$，其他 $x_j = 0(j \neq i)$,，操作 k 次以后，$x_m(m \leq n)$ 对 p 取模的值。保证 n 次单位根在模 p 意义下存在。

$k \leq 2 \times 10^9$，$p \leq 2 \times 10^9, \Sigma n \leq 10^6$

做法

考虑答案的组合意义后对其进行简单的单位根反演。

$$\begin{aligned}\sum_{i=0}^{\infty}\binom{K}{in+m} &= \sum_{i=0}^{K}\binom{K}{i}\frac{\sum_{j=0}^{j\leq n-1}\omega_n^{(i-m)j}}{n}\\ &= \frac{1}{n}\sum_{j=0}^{j\leq n-1}\omega_n^{-mj}\sum_{i=0}^{K}\binom{K}{i}\omega_n^{ij}\\ &= \frac{1}{n}\sum_{j=0}^{j\leq n-1}\omega_n^{-mj}(1+\omega_n^j)^K\end{aligned}$$

求出单位根之后暴力进行快速次幂求和即可。复杂度 $O(\sum n \log K)$。

例题二

题意

有 q 次询问，每次给定最高项次数小于 n 的多项式 $A(x)$，询问 $A(x)^K$ 对 $(x^n - 1)$ 取模后，结果的每一项系数对质数 p 取模的结果。保证在模 p 意义下存在 n 次单位根。

$k \leq 2 \times 10^9$，$p \leq 2 \times 10^9$，$\Sigma n \leq 10^6$

思考

直接推式子计算答案非常麻烦。采用 $O(n \log n \log K)$ 的多项式快速次幂复杂故过高。同时由于存在对 $x^n - 1$ 取模的缘故，无法通过多项式 exp 和多项式 ln 来解决该问题。

由于模长为 n 的离散傅里叶变换本质上是进行了一次长度为 n 的循环卷积，因此如果能够快速进行模长恰好为 n 的离散傅里叶变换，就可以以一个较小的常数在 $O(n(\log n + \log K))$ 的复杂度内解决这个问题。

但是快速傅里叶变换仅能处理模长为 2^k 的特殊情况，而离散傅里叶变换时间复杂度过大，因此需要更加优秀且通用的解决一般模长快速傅里叶变换的方法。

3.2 快速傅里叶变换

我们先来回顾一下快速傅里叶变换的做法。假设模长 $n = 2^k$，且我们要求出将 $\omega_n^0, \omega_n^1, \cdots, \omega_n^{n-1}$ 依次代入多项式 $f(x)$ 中得到的长度为 n 的点值序列 a。设 $f(x) = \sum_{i=0}^{n-1} f_i x^i$。

设 $f_1(x)=\sum_{i=0}^{n/2-1} f_{2i}x^i, f_2(x)=\sum_{i=0}^{n/2-1} f_{2i+1}x^i$, 则有 $f(x)=f_1(x^2)+xf_2(x^2)$。

考虑将 ω_n^k 和 $\omega_n^{k+n/2}$ 代入多项式 $f(x)$ 的值，分别可以得到:

$$\begin{aligned} f(\omega_n^k) &= f_1(\omega_n^{2k}) + \omega_n^k f_2(\omega_n^{2k}) \\ &= f_1(\omega_{n/2}^k) + \omega_n^k f_2(\omega_{n/2}^k) \\ f(\omega_n^{k+n/2}) &= f_1(\omega_n^{2k+n}) + \omega_n^{k+n/2} f_2(\omega_n^{2k+n}) \\ &= f_1(\omega_{n/2}^k) - \omega_n^k f_2(\omega_{n/2}^k) \end{aligned}$$

不难发现运行结果仅仅和 $f_1(\omega_{n/2}^k)$ 与 $f_2(\omega_{n/2}^k)$ 有关。因此可以使用 $O(n)$ 的复杂度将其转化为两个大小为 $n/2$ 的子问题。运用主定理不难得到时间复杂度为 $O(n\log n)$。这就是一般所说的快速傅里叶变换 (FFT)。

3.3 基于分治的快速傅里叶变换

类比于快速傅里叶变换，我们考虑将其扩展到更加一般的模长。一般的，现在假设我们需要计算 $V(j)=\sum_{i=0}^{i<n} a_i\omega_n^{ij}$ 的值。

类似于上面的思路，设 n 的最小质因子为 d, 设 $m=n/d, j=pd+q(0\le p<m, 0\le q<d), i=xm+y(0\le x<d, 0\le y<m)$, 将上述定义代入 DFT 式子，得到:

$$\begin{aligned} V(pd+q) &= \sum_{i=0}^{i<n} \omega_n^{i(pd+q)} a_i \\ &= \sum_{i=0}^{i<n} \omega_n^{ipd}\omega_n^{iq} a_i \\ &= \sum_{i=0}^{i<n} \omega_m^{ip}\omega_n^{iq} a_i \\ &= \sum_{y=0}^{y<m}\sum_{x=0}^{x<d} \omega_m^{(xm+y)p}\omega_n^{(xm+y)q} a_{xm+y} \\ &= \sum_{y=0}^{y<m}\sum_{x=0}^{x<d} \omega_m^{yp}\omega_d^{xq}\omega_n^{yq} a_{xm+y} \end{aligned}$$

依次枚举 q 的取值，则 $\Sigma_{x=0}^{x<d}\omega_d^{xq}\omega_n^{yq}a_{xm+y}$ 是一个仅和 y 有关的常数，我们将其设为 $B(y)$，并且代入原式，此时有:

$$\begin{aligned} V(pd+q) &= \sum_{y=0}^{y<m}\sum_{x=0}^{x<d} \omega_m^{yp}\omega_d^{xq}\omega_n^{yq} a_{xm+y} \\ &= \sum_{y=0}^{y<m} \omega_m^{yp} B(y) \end{aligned}$$

对照最初的 DFT 式定义，不难发现在 q 确定时我们将其转化为了恰好 1 个大小为 m 的子问题，这启发我们可以通过递归的方法解决这个子问题。

考虑时间复杂度。设解决模长为 n 的 DFT 问题的时间复杂度为 $F(n)$, 则根据算法定义我们有 $F(n) = dF(\frac{n}{d}) + O(dn)$, 且 $F(1) = O(1)$。

设 n 的质因子分解形式为 $\prod a_i^{p_i}$，同时根据 n 质因子分解形式设函数 $\Omega_0(n) = \sum a_i p_i$。则对于 $F(1)$ 满足 $F(n) = O(n\Omega_0(n))$, 且若对于所有 $< n$ 的整整数均满足 $F(n) = O(n\Omega_0(n))$, 则有 $F(n)$=$O(\Omega_0(n/d)n) + O(dn)$, 不难发现其仍然满足 $F(n) = O(n\Omega_0(n))$。据此我们即可计算其复杂度。

但是 $\Omega_0(n)$ 在最坏情况下仍然可以取到 $O(n)$ 级别，因此我们需要一个复杂度更加优秀的算法。

3.4 基于卷积的快速傅里叶变换

让我们再次回到 DFT 和 IDFT 的式子上来考虑这个问题。

$$V(j) = \sum_{i=0}^{i<n} \omega_n^{ij} a_i$$

考虑 $i \times j$ 的实际含义，可以看成有两堆物品，第一堆大小为 i，第二堆大小为 j，从每堆中选出一个物品的方案数。

在这个方案数上考虑进行容斥。用两堆物品合并，并在其中选择两个不记顺序的互异元素的方案数，减去第一堆中选两个不记顺序的互异元素的方案数和第二堆中选两个不记顺序的互异元素的方案数的和，即为从每堆中各自选出一个物品的方案数。转化成数学公式即为:$\binom{i+j}{2} - \binom{i}{2} - \binom{j}{2} = i \times j$。将上面的容斥式子代入 DFT 式，可以得到:

$$V(j)\omega_n^{\binom{j}{2}} = \sum_{i=0}^{i<n} a_i \omega_n^{-\binom{i}{2}} \omega_n^{\binom{i+j}{2}}$$

这个式子可以看成是 $C(x) = \sum \omega_n^{-\binom{i}{2}} x^i$ 对 $B(x) = \sum a_i \omega_n^{\binom{i}{2}} x^i$ 做一次减法卷积的结果，然后对每一项系数乘上一个对应常数的值。此时我们并不关心卷积的模长带来的影响，只关心卷积的结果，而不关心卷积的模长问题，因此可以直接采用 MTT 求解。时间复杂度 $O(n \log n)$。

类似的，对于 IDFT 式也代入上面的容斥式子，可以得到:

$$V(j)\omega_n^{-\binom{j}{2}} = \sum_{i=0}^{i<n} a_i \omega_n^{\binom{i}{2}} \omega_n^{-\binom{i+j}{2}}$$

类似的，这个式子仍可以被看成是一次减法卷积，然后对每一项系数乘上一个对应常数的值，因此仍然可以直接一次 MTT 求解。时间复杂度 $O(n \log n)$。这个算法也被称为 bluestein 算法或者 Z 变换。

事实上，对于任意的非零数字 a，我们都可以通过这种思路来计算 $V(j)=\Sigma_{i=0}^{i<n}x_j a^{ij}$ 的值。我们只需要保证存在 a^{ij} 以及其逆元即可。也就是说，对于复数该算法仍然适用。

- 小优化技巧：原本减法卷积部分的模长需要开到 $3*n$ 级别，但是根据 FFT 加速多项式乘法本质是进行了一次循环卷积的特性，由于只关心中间 n 个元素的正确性，因此将最后 n 个元素和前面的 n 个元素在结果中重合对答案的正确性不会产生任何的影响。因此只需要将模长开到 $2*n$ 级别即可。

3.5 例题

例题三 [3]

题意

定义两个简单无向图 $G_1=(V_1,E_1),G_2=(V_2,E_2)$ 的乘积为一个新的图 $G_1\times G_2=(V^*,E^*)$。其中新的点集 V^* 满足 $V^*=\{(a,b)|a\in V_1,b\in V_2\}$, 其中新的边集 E^* 满足 $E^*=\{((u_1,v_1),(u_2,v_2))\mid(u_1,u_2)\in E_1,(v_1,v_2)\in E_2\}$。

给定正整数 n，以及 n 个正整数 $m_1,m_2,\cdots,m_n$。你需要求出新图 $H=(((G_1\times G_2)\times G_3)\times\cdots)\times G_n$ 的期望连通块数量对 998244353 取模的结果。其中 G_i 所有包含 m_i 个节点的图中等概率随机生成。

$n,m_i\le 100000$

做法

我们考虑给我们一组 G_k 的序列之后如何计算连通块数量。

首先考虑在每个图中选一个点，如果某个选的点数的度数为 0（我们称其为“孤立点”），则这个点序列在 H 上对应的点是孤立点。否则我们考虑每个图中选择一个大小 >1 的连通块。考虑这些连通块的乘积得到的连通块有多少。注意到现在每个点都有邻边，且是无向图，因此我们可以在一条边上反复走。两个点序列之间的可达性可以简化为路径长度的奇偶性。

如果两个点在一个存在奇环的图中，那么显然奇数长度和偶数长度的路径都有。如果两个点在一个二分图中，那么这和他们是否在同一部中有关。因此我们可以得到：如果选的这 n 个连通块中有 k 个不存在奇环，那么这些连通块的乘积将会给答案贡献 $2^{\max(k-1,0)}$ 个连通块。因此我们只需要知道全体大小为 m_k 的图可以有多少个孤立点，多少个无奇环的连通块，多少个连通块，则可以由此算出答案。

记无向简单图的 EGF 为 $G(x)=\sum_{n\ge 0}\frac{2^{\binom{n}{2}}x^n}{n!}$，则联通图的 EGF 为 $C=\ln G$。不难得到连通块数量的 EGF 由枚举连通块如何插入一个图得到，即为 $G\ln G$。

我们考虑染色二分图的 EGF$B=\sum_{n\ge 0}\sum_{m\ge 0}\frac{\binom{n+m}{n}2^{nm}x^{n+m}}{(n+m)!}$，则无奇环的连通块显然恰有 2 种方

[3] Source:【UOJ498】新年的追逐

法染色，可以得到 EGF 为 $\frac{\ln B}{2}$，无奇环连通块数量可以通过 $\frac{G\ln B}{2}$ 表示。这里我们可以采用 bluestein 算法来优化求染色二分图的 EGF 的过程。

时间复杂度 $O(n\log n)$。

例题四 [4]

题意

给定长度为 n 的非负整数序列 b_i 和 c_i，下标均从 0 开始。

已知非负整数序列 a 满足 $c_i=\sum_{k=0}^{n-1}(b_{k-i \bmod n}-a_k)^2$, 求 a_i 的所有合法解。保证 b 的所有长度为 n 的循环移位线性无关。

$n\le 10^5$，$b_i\le 10^3$，$c_i\le 5\times 10^6$

做法

为了方便描述，接下来所有的序列都是循环节为 n 的无线序列。

由于 $(a-b)^2=a^2+b^2-2ab$，则 $c_k-c_{k-1}=\sum_{i=0}^{n-1}b_i(a_{i+k}-a_{i-k+1})$。设 $c'_k=c_k-c_{k-1}, a'_k=a_k-a_k-1$，则有 $c'_k=\sum\limits_{y-x \bmod n=k} b_x a'_y$。因此 c'_k 可以被看成是 $\sum b_i x^i$ 对 $\sum a'_i x^{n-i}$ 做模长为 n 的卷积的结果。利用模意义下的 bluestein 算法我们可以快速计算出唯一一组合法的 a'_i。

设 $a_0=x$，则其余数字可以使用 x 表示，此时即可列出一个一元二次方程，就可以解出不超过两个合法解，直接代入检验合法性即可。

时间复杂度 $O(n\log n)$。

4 DFT 在多点求值上的应用

4.1 一般思路

多项式多点求值问题是多次询问多项式 $f(x)$ 在值 y 处的点值的问题。

由于离散傅里叶变换是将 $\omega_n^0,\omega_n^1,\cdots,\omega_n^{n-1}$ 依次代入多项式 $f(x)$ 中得到的长度为 n 的点值序列 a。因此在解决多项式多点求值问题时，如果询问的点值满足某些特殊性质，我们就可以将其重排列之后，采用 bluestein 算法加速多项式多点求值的过程。

例题五 [5]

题意

给定 n 次多项式 $f(x)=\sum\limits_{i=0}^{i\le n} f_i x^i$。$Q$ 次询问，第 i 次询问 $f(q_i)$ 对 $p=998244353$ 取模后的结果。q_i 按照如下方式生成:

[4]Source:CF 901E Cyclic Cipher

[5]Source:【UOJ500】任意基 DFT

$\forall 1 \le i \le Q$，$q_i = (q_{i-1} \times a + b) \ mod \ 998244353$

$1 \le n \le 2.5 \times 10^5$，$1 \le Q \le 10^6$，$2 \le a < 998244353$，$0 \le q_0, b < 998244353$

做法

假设我们已知任意 n 次多项式 $f(x)$, 则我们可以:

- 在 $O(n)$ 的复杂度内找到唯一一个次数不超过 n 次多项式 $g(x)$ 满足 $g(x) = f(x * k)$
- 在 $O(n \log n)$ 的复杂度内找到唯一一个次数不超过 n 次多项式 $h(x)$ 满足 $h(x) = f(x + k)$。

观察到询问的值比较的特殊，我们尝试利用一些上述变换，通过加减乘除操作将询问的值转化成比较优美的形式。

首先归纳得到 $q_i = q_0 a^i + \Sigma_{j=0}^{j<i} b a^j$。接下来进行如下变换:

(1) 将右式乘以 $a-1$，得到 $q_i = q_0(a-1)a^i + ba^i - b$。

(2) 将右式加上 b，得到 $q_i = q_0(a-1)a^i + ba^i$。

(3) 将其除以 $q_0(a-1)+b$，得到 $q_i = a^i$

注意在 $q_0(a-1)+b$ 为 0 时，此时满足 $q_i = q_0$，因此只需要求出 q_0 处的单个点值即可。

否则在给定数据范围下，上述所有运算均合法，因此我们可以对 $f(x)$ 做类似的变换，即求出 n 次多项式 $g(x)$ 满足 $g(x) = f(\frac{x(q_0(qx-1)+qy)-qy}{qx-1})$。根据上述变换的性质，有且仅有一个满足条件的次数不超过 n 次的多项式 $g(x)$。

现在我们需要求出多项式 $g(x)$ 在 $a^1, a^2, \cdots, a^Q$ 处的点值。即需要求出 $V(i) = \Sigma_{j=0}^{j\le n} g_j a^{ij}$。不难发现求解的式子满足 bluestein 算法的各项性质，因此可以使用该算法求解。

时间复杂度 $O((n+Q)\log(n+Q))$。

例题六 [6]

题意

给定 n 次多项式 $f(x) = \sum_{i=0}^{i\le n} a_i x^i$。$Q$ 次询问, 每一次给定正整数 y, 询问 $f(y)$ 对 $p = 786433(3 * 2^{18} + 1)$ 取模的值。

$n, Q \le 250000$，$0 \le y < p$

做法

考虑离散傅里叶变换的实际含义，不难发现模长为 n 的 DFT 可以看成是将 $\omega_n^0, \omega_n^1, \cdots, \omega_n^{n-1}$ 代入多项式 $A(x)$ 产生的点值。

[6]Source:CODECHEF POLYEVAL

由于 p 是质数，因此 p 存在原根，这说明存在 g 满足 $\omega_{p-1} \equiv g(\bmod p)$。同时根据原根的定义，$g^0, g^1, \cdots, g^{p-2}$ 在模意义下互不相同且恰能取到所有与 p 互质的小于 p 的正整数。也就是形成了一个 $[1, p-1]$ 的排列。

因此如果我们对 $A(x)$ 跑一轮模长为 $p-1$ 的 DFT，并且将得到的点值重排列之后就可以得到 $1, 2, \cdots, p-1$ 的点值。同时，0 处的点值显然为 a_0。因此所有点值都已经被求出。

时间复杂度 $O(p \log p)$。

4.2 推广

我们尝试将例题六的模数拓展到更加一般的情况。现在假设我们需要求出点值对 $p = q^c (c \geq 1)$ 取模的结果, 其中 q 为大于 2 的任意质数。

由于 q^c 存在原根，这说明存在 g 满足 $\omega_{\phi(q^c)} \equiv g(\bmod q^c)$。同时根据原根的定义，$g^0, g^1, \cdots, g^{\phi(q^c)-1}$ 在模意义下互不相同且恰能取到所有与 q^c 互质的小于 q^c 的正整数。这些点的点值我们仍然可以使用一次 DFT 计算得出。

对于与 q^c 不互质的整数，设其为 x，则显然有 $x^c \equiv 0(\bmod q^c)$, 这说明我们仅需要考虑小于 c 次的项产生的贡献即可，因此直接暴力计算前 $c-1$ 项的值就可以算出点值。由于 $c = O(\log p)$，该部分时间复杂度仍为 $O(p \log p)$。

特殊的，对于 $p = 2^c (c \geq 3)$ 的特殊模数，虽然其不存在原根，但是仍然可以证明 $\pm 3^0, \pm 3^1, \cdots, \pm 3^{2^{c-2}}$ 在模意义下互不相同且均与 2 互质，证明过程较为复杂此处略去。这些点的点值我们仍然可以使用两次 DFT 计算得出。类似的，与 2 不互质的仍然只需要考虑小于 c 次的项。

但是由于快速傅里叶变换的模长与 2 不互质，在 IDFT 时候会出现问题。因此我们需要通过将快速傅里叶变换部分的模数乘以模长，同时将最后的结果暴力除以模长解决。或者通过将 FFT 部分的模长设置为 3^k，然后利用基于分治的快速傅里叶变换解决。因此仍然可以在 $O(p \log p)$ 的时间复杂度内解决该问题。

若使用中国剩余定理合并不同 q^c 的答案，则对于任意模数均可做到 $O(p \log p)$ 的多项式多点求值。

5 DFT 在多点插值上的应用

多项式多点插值问题是给定多项式 $f(x)$ 在某些位置 y 上的点值，询问 $f(x)$ 的各项系数的问题。

由于 DFT 可以在多项式多点求值上得到良好的应用，因此我们考虑将其拓展到其逆操作，即多项式多点插值上。

5.1 连续点值的多点插值

题意

给定 n 次多项式 $f(x)$ 在 $0 \sim n$ 处的点值序列 a，询问 $f(x)$ 每一项系数对 p 取模的结果。

$0 < n < p \leq 500000$，p 为质数。

算法

因为多点求值的逆操作为多点插值。DFT 的逆操作为 IDFT。如果多点求值看成是一次 DFT 操作，那么这提示我们可以尝试使用一次 IDFT 来解决多项式多点插值问题。

具体的，首先我们根据 $0 \sim n$ 的点值计算出 $0 \sim p-1$ 的点值。这里可以通过点值与多项式的下降幂性质之间的互相转换实现，这一部分不再展开。

由于我们已知 $1 \sim p-1$ 的点值，我们可以通过一次 IDFT 确定出一个不超过 $p-1$ 次多项式 $f(x)$, 使得 $f(x)$ 在 $1 \sim p-1$ 处的点值恰好为 $a_1, a_2, \cdots, a_{p-1}$。这里部分正确性可以由 DFT 与 IDFT 的正确性得到。因此现在只需要构造多项式 $h(x)$ 满足 $h(x) \equiv [x = 0] (\mathrm{mod} p)$。不难发现此时 $f(x) + h(x)(a_0 - f(0))$ 即为答案。

我们可以构造以下多项式 $h(x) = \sum_{j=0}^{j<p-1} x^j$。在 $x = 0$ 时其值为 1, 在 $x = 1$ 是其值为 p，在模意义下值为 0。在 x 大于 1 时根据等比数列求和公式，答案为 $\frac{x^{p-1}-1}{x-1}$。根据费马小定理 $x^{p-1} \equiv 1 (\mathrm{mod}\ p)$, 因此在模意义下值仍为 0。直接将其代入上面的式子中即可得到答案。

时间复杂度 $O(p \log p)$。

5.2 非连续点值的多点插值

题意

给定 n 次多项式 $f(x)$ 在 $a_1, a_2, \cdots, a_{n+1}$ 处的点值 $g_1, g_2, \cdots, g_{n+1}$，询问 $f(x)$ 每一项系数对 p 取模的结果。

$0 < n < p \leq 300000$，p 为质数，$n < p - 2$。保证 $a_i > 0$，且 a_i 互不相同。

算法

根据拉格朗日插值，我们可以很方便的计算出答案为 $\sum_i (g_i \sum_{j \neq i} \frac{x-a_j}{a_i-a_j})$。

设 $F(x) = \Pi_i (x - a_i)$, 并对式子稍作转化，答案可以化简为 $(\sum_i \frac{g_i}{(x-a_i)F'(a_i)}) F(x)$。

对于求解 $F'(a_i)$ 的那部分，我们可以通过多项式多点求值来解决，最后乘以 $F(x)$ 的部分可以放到最后处理。因此现在我们将其转化为两个比较有价值的子问题：

- 求解 $F(x) = \Pi_i (x - a_i)$。
- 求解 $G(x) = \sum_i \frac{v_i}{x-a_i}$。

子问题 1

$$F(x) = \prod_i (x - a_i)$$
$$\frac{F(x)}{\prod_i -a_i} = \prod_i (-\frac{1}{a_i}x + 1)$$
$$\ln\left(\frac{F(x)}{\prod_i -a_i}\right) = \sum_i \ln(-\frac{1}{a_i}x + 1)$$
$$\ln'\left(\frac{F(x)}{\prod_i -a_i}\right) = \sum_i \sum_j x^j \frac{1}{(a_i)^{j+1}}$$

根据上式子问题 1 可以通过一次多项式 exp 和一次积分，在 $O(p \log p)$ 的复杂度内转化为子问题 2。接下来我们仅讨论子问题 2 的做法。

子问题 2

$$\begin{aligned} G(x) &= \sum_i \frac{v_i}{x - a_i} \\ &= \sum_i v_i \sum_j x^j \left(\frac{1}{a_i}\right)^{j+1} \\ &= \sum_j x^j \sum_i v_i \left(\frac{1}{a_i}\right)^{j+1} \\ &= \sum_j x^j \sum_k \sum_i [\omega_{p-1}^k = a_i] v_i \omega_{p-1}^{-k(j+1)} \end{aligned}$$

设 $H(k) = \sum_i [\omega_{p-1}^k = a_i] v_i$, 则有

$$G(x) = \sum_j x^j \sum_k H(k) \omega_{p-1}^{-k(j+1)}$$
$$G(x)x = \sum_j x^j \sum_k H(k) \omega_{p-1}^{-kj}$$

观察最后式子，不难发现其满足 bluestein 算法的性质。因此仍然可以使用 bluestain 算法加速计算 $G(x)x$ 的值。

因此两个子问题我们都可以在 $O(p \log p)$ 的时间复杂度内解决，总复杂度为 $O(p \log p)$。

特殊情况

对于 $a_i = 0$ 的情况，由于在解决子问题 1,2 时，我们的推导过程使用到了 $\frac{v_i}{x}$ 以及其导数, 因此上面的推导会产生严重的错误。

其中一种解决方案是利用之前在例题三提到的构造 $g(x) = f(x + k)$ 的方式，将 a_i 整体平移之后使得对于所有 a_i 其值全部非 0, 据此我们可以计算出唯一的 g_i。最后将计算出来的 $g(x)$ 重新平移回来即可。

另外一种解决方案是在插值时忽略 $a_i = 0$ 处的点值进行插值，最后加上若干倍的 $\Pi_{a_i \neq 0} x - a_i$ 使得 $f(0)$ 处的点值正确。这种方法也被称为部分拉格朗日插值法。

5.3 拓展

子问题 2 可以看成是给定列向量之后将其右乘一个行列式非 0 的 Vandermonde 矩阵的结果。事实上我们也可以利用上述算法在模数较小时解 Vandermonde 矩阵。

设我们需要解方程组:$G(j) = \sum_i a_i x_i^j$, 类似于问题 2, 设 $H(k) = \sum_i [\omega_{p-1}^k = x_i] v_i$, 则方程组可以转化为:$G(j) = \sum_k H(k) \omega_{p-1}^{kj}$。

类比于之前探讨过的问题，不难发现，我们将问题转化到了连续点值的多项式多点插值上，因此可以直接利用该算法，可以在 $O(p \log p)$ 的时间复杂度内解决该问题。

6 后记

上述大部分算法都是在模域下的 bluestein 算法的应用。事实上在 2003 年，学界提出了一种时间复杂度为 $O(n \log n)$ 的 bluestein 算法的逆运算。有兴趣的读者可以自行查阅相关论文。

7 展望

本文提到了一些解决快速傅里叶变换问题的方法和技巧，但在浩瀚的算法海洋中仍只是冰山一角。希望未来能有更多的 bluestein 算法的应用被发现和发明，也希望有更多有意思的快速傅里叶变换题出现在信息学奥赛中。

8 致谢

感谢中国计算机学会提供学习和交流的平台。

感谢绍兴一中的陈合力老师和董烨华老师的关心和指导。

感谢国家集训队教练高闻远的指导和帮助。

感谢戴江齐同学对于非连续点值多点插值的算法方面做出的启发。

感谢邓明扬同学，孔朝哲学长为本文验稿。

感谢其他对我有过帮助和启发的老师和同学。

感谢父母对我的关心、支持与无微不至的照顾。

参考文献

[1] 张家琳, 《多项式乘法》, 2002 年集训队论文.

[2] 毛啸, 《再探快速傅里叶变换》, 2016 年集训队论文.

[3] Bluestein, L. (1970). ”A linear filtering approach to the computation of discrete Fourier transform”.

最小内向森林问题

浙江省杭州学军中学 张哲宇

摘 要

最小内向森林问题是指，对于一张带权的有向图，求固定边数的最小内向森林。本文会围绕最小内向森林问题展开，并给出时间复杂度为 $O(E\log E)$ 的算法。

1 引言

对于一张带权的有向图，并给定一个自然数 k，求最小的共包含 k 条边的子图，满足这个子图中每一个弱连通块都是一棵内向树。这样的问题就是最小内向森林问题，该问题在近几年才首次出现在算法竞赛中，是一个较为新颖的问题。

早在十几年前，国内的评测网站上就出现了最小树形图问题。但是，直到现在，树形图相关的问题依旧是一类冷门问题。最小内向森林问题拥有一定的潜力，可能成为一类用途广泛的图论模型，具有研究的意义。

如今，最小内向森林问题的解决方法通常是基于凸优化和最小树形图的算法，该算法效率低下、功能弱且扩展性差。本文提出了一个时间复杂度为 $O(E\log E)$ 的算法，该算法效率高，功能强且扩展性好。

本文共有七节，每节的内容大致如下。

第二节中，精确地定义了最小内向森林问题。

第三节中，简单介绍了拟阵。拟阵作为组合数学的一个分支，现在常常被用于解决各种图论中的最优化问题。本节中，将会用拟阵来描述最小内向森林问题，并借助拟阵发现性质。

第四节中，给出了基于凸优化和最小树形图的算法。最小树形图问题早在上个世纪六十年代就已经被解决，解决的算法被称为“朱刘算法”或“Edmonds' algorithm”。最小树形图问题与最小内向森林问题同为树形图相关问题，有些许相似之处。本节中，会详细介绍如何使用凸优化将最小内向森林问题转化成最小树形图问题，该算法需要使用二分，二分的次数决定了该算法的精度，当二分次数为 $\log_{\frac{1}{2}}\varepsilon$ 的时候，该算法的时间复杂度为 $O\left(E\log E\log_{\frac{1}{2}}\varepsilon\right)$。

第五节中，分析前几节的内容并得到优先内向树扩张算法。基于凸优化和最小树形图的算法拥有很多冗余操作，剖析这些操作的意义可以使算法更加精简。本节中，会详细介绍如何

得到优先内向树扩张算法，然后形式化地将其描述出来，最后简单说明如何得到 $O(E\log E)$ 的时间复杂度。

第六节中，举了一例最小内向森林问题的应用。

第七节中，对本文内容进行总结。

2 定义

给定一张有向图 $G=(V,E)$，其中 V 是有向图的点集，E 是有向图的边集。有边权 $w:E\mapsto\mathbb{R}$。

定义**内向树** 为一个弱连通块，没有环，其中有一个根节点，所有的非根节点恰有一条出边。

定义**内向森林** 为一个有向图，其中的每一个弱连通块都是一棵内向树。

定义**边子集** 为边集 E 的一个子集。

定义一个边子集 E^* 的**边导出子图** 为由原图中所有点和 E^* 中的所有边构成的有向图。

规定一个边子集 E^* 的权值等于其中每条边的权值和，即 $w(E^*)=\sum_{e\in E^*}w(e)$。

那么，最小内向森林问题可以被如下描述：

给定一张有向图和一个整数 k，求权值最小的大小为 k 的边子集，满足其边导出子图是内向森林。

对于同一张有向图，记最小内向森林问题的答案关于 k 的函数为 $\mathrm{MDF}(k)$。

3 拟阵

3.1 定义

有一个子集系统 $M=(S,\mathcal{I})$。如果 M 满足以下三个公理，则 M 是一个拟阵。

公理 1. $\emptyset\in\mathcal{I}$。

公理 2. $\forall A\subset B$，*若* $B\in\mathcal{I}$，则 $A\in\mathcal{I}$。

公理 3. $\forall|A|<|B|$，*若* $A\in\mathcal{I}$ *且* $B\in\mathcal{I}$，则 $\exists x\in B-A$，$A\cup\{x\}\in\mathcal{I}$。

对于拟阵 $M=(S,\mathcal{I})$，称 $\mathcal{I}$ 中的元素为独立集。

3.2 可截取性质

引理 3.1. *给定拟阵* $M=(E,\mathcal{I})$ *和任意自然数* s，*那么保留大小不超过* s *的独立集后依旧是拟阵，即* $M^*=(E,\mathcal{I}^*)$ *是拟阵，其中* $\mathcal{I}^*=\{S\in\mathcal{I}:|S|\le s\}$。

证明. 因为 $s \geq 0$，所以空集是独立集，即 $\emptyset \in \mathcal{I}^*$。

$\forall A \subset B$，若 $B \in \mathcal{I}^*$，则 $B \in \mathcal{I}$，所以 $A \in \mathcal{I}$。又因为 $|A| \leq |B| \leq s$，所以 $A \in \mathcal{I}^*$。

$\forall |A| < |B|$，若 $A \in \mathcal{I}^*$ 且 $B \in \mathcal{I}^*$，则 $A \in \mathcal{I}$ 且 $B \in \mathcal{I}$，所以 $\exists x \in B - A$，$A \cup \{x\} \in \mathcal{I}$。又因为 $|A| < |B| \leq s$，则 $|A| + 1 \leq s$，所以 $\exists x \in B - A$，$A \cup \{x\} \in \mathcal{I}^*$。 □

3.3 拟阵与最小内向森林问题

接下来我们使用拟阵表示最小内向森林问题。

令 $M_1 = (E, \mathcal{I}_1)$，其中 E 为边集，$\mathcal{I}_1$ 为边导出子图中把边视为无向边后没有环的边子集的集合。

令 $M_2 = (E, \mathcal{I}_2)$，其中 E 为边集，$\mathcal{I}_2$ 为边导出子图中每个点出度不超过 1 的边子集的集合。

接下来我们分别阐明 M_1 和 M_2 是拟阵，也就是说它们都满足三个公理。

对于 M_1，显然满足**公理 1** 和**公理 2**。接下来分析**公理 3**，$\forall |A| < |B|$，若 $A \in \mathcal{I}_1$ 且 $B \in \mathcal{I}_1$，那么 A 的连通块数量大于 B 的连通块数量，必然存在一条边 $e \in B$，连接 A 中的两个连通块，这便是 $B - A$ 中可以加入到 A 中的元素。所以**公理 3** 也满足，M_1 是一个拟阵。

对于 M_2，显然满足**公理 1** 和**公理 2**。接下来分析**公理 3**，$\forall |A| < |B|$，若 $A \in \mathcal{I}_2$ 且 $B \in \mathcal{I}_2$，令 A 中有出边的点集为 $U(A)$，B 中有出边的点集为 $U(B)$，则 $|U(A)| \leq |U(B)|$，只需要选择任一以 $U(B) - U(A)$ 中结点为起点的在 B 中的出边，加入到 A 中即可。所以**公理 3** 也满足，M_2 是一个拟阵。

引理 3.2. 对于任意边集 E^*，$E^* \in \mathcal{I}_1 \cap \mathcal{I}_2$ 当且仅当 E^* 的边导出子图是内向森林。

证明. 先证明，若 $E^* \in \mathcal{I}_1 \cap \mathcal{I}_2$ 可得 E^* 的边导出子图是内向森林。对于 E^* 的边导出子图中的一个弱连通块，由 $\mathcal{I}_1$ 的定义可得，在该弱连通块内，边数比点数少一。由 $\mathcal{I}_2$ 的定义可得，每个点出边数量不超过 1，所以其中恰有一个点没有出边，不妨设这个点为 t。那么，每一条和 t 相连的边都指向 t，以此类推即可得到以 t 为根的内向树。既然 E^* 的边导出子图中的每一个弱连通块都是内向树，所以 E^* 的边导出子图是内向森林。

另一个方向的证明通过内向森林的定义可得。 □

综上，现已使用拟阵来描述最小内向森林问题，即求 $\min\limits_{S \in \mathcal{I}_1 \cap \mathcal{I}_2, |S| = k} w(S)$。

因为最小内向森林问题固定了边的数量，所以所有的边的边权可以同时加减。为了方便下文的讨论，我们不妨设 $\max\limits_{e \in E} w(e) \leq 0$。因此，我们可以将问题变成 $\min\limits_{S \in \mathcal{I}_1 \cap \mathcal{I}_2, |S| \leq k} w(S)$。

3.4 拟阵交和线性规划

3.4.1 拟阵交问题

拟阵交问题的定义：给定两个定义在相同基础集上的拟阵 $M_1 = (E, \mathcal{I}_1), M_2 = (E, \mathcal{I}_2)$，求 $\min_{S \in \mathcal{I}_1 \cap \mathcal{I}_2} w(S)$。

最小内向森林问题可以被描述成一个标准的拟阵交问题。令 $M_2^* = (E, \mathcal{I}_2^*)$，其中 $\mathcal{I}_2^* = \{S \in \mathcal{I}_2 : |S| \le k\}$，$M_2^*$ 是拟阵。那么 $\min_{S \in \mathcal{I}_1 \cap \mathcal{I}_2, |S| \le k} w(S) = \min_{S \in \mathcal{I}_1 \cap \mathcal{I}_2^*} w(S)$。

3.4.2 线性规划

线性规划问题可以这样描述：给定一个矩阵 A 和两个向量 b, c。

$$\begin{aligned} \min \quad & c^T x. \\ \text{s.t.} \quad & Ax \le b, \\ & x \ge 0. \end{aligned}$$

其中 x 是变量，第一行表示最优化的目标，其余表示对 x 的约束。

如果只关注限制，可以发现 x 的可行范围是一个 $|c|$ 维的凸多胞形。而向量 c 的作用只是规定了最终要求哪一个方向上的极点，即某一个方向上的最远点。

如果，对于任意的 c，最优解中 x 都是整点，即凸多胞形上的任意一个极点都是整点。那么，这样的线性规划称为幺模的。当线性规划是幺模的时候，通常可以无视特定问题中的整数限制。

定理 3.1. *对于一个线性规划和一个向量 d，令 $F(k)$ 表示增加任意一条限制 $d^T x \le k$ 后线性规划的值。则，$F(k)$ 的导数 $F'(k)$ 单调。*

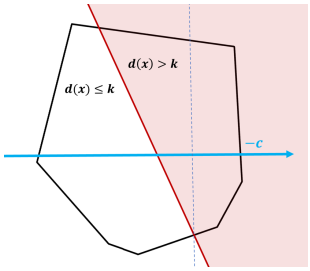


图 2.1

证明. *考虑加入限制之前，x 可行范围是一个凸多胞形，将其映射到 c 与 d 所在的平面上后一定是一个二维凸包，如图 2.1 所示。在这个平面上，若以 $-c$ 为基准，则 d 与凸包的切点在凸包上的斜率单调，故 $F'(k)$ 单调。* □

3.4.3 两者联系

令 $\mathcal{X}(S)$ 表示一个每一维坐标都是 0 或 1 的 $|E|$ 维空间内的点，其第 i 维坐标等于 1 当且仅当第 i 个元素属于 S。

令 $X(M_1, M_2) = \{\mathcal{X}(S) : S \in \mathcal{I}_1 \cap \mathcal{I}_2\}$。

令 $conv(X(M_1, M_2))$ 为包含点集 $X(M_1, M_2)$ 的最小的凸多胞形。

令

$$P(M_1, M_2) = \{x \in \mathbb{R}^{|E|} : \begin{aligned} x(S) &\le \max_{T \in \mathcal{I}_1, T \subset S} |T| && \forall S \subset E \\ x(S) &\le \max_{T \in \mathcal{I}_2, T \subset S} |T| && \forall S \subset E \\ x_e &\ge 0 && \forall e \in E\}. \end{aligned}$$

定理 3.2. $conv(X(M_1, M_2)) = P(M_1, M_2)$。

该定理的证明详见 [2]。

定义 $|E|$ 维空间内的点 $\mathcal{X}(w)$ 满足 $\forall e \in E, \mathcal{X}(w)_e = w(e)$。那么拟阵交问题可以视为 $X(M_1, M_2)$ 中在 $-\mathcal{X}(w)$ 方向上的极点，同时也等于 $conv(X(M_1, M_2))$ 在 $-\mathcal{X}(w)$ 方向上的极点。

如果将 $P(M_1, M_2)$ 的限制视为线性规划的限制，权值函数 w 视为线性规划的目标函数。那么，我们可以将拟阵交问题转化为线性规划问题。

3.4.4 凸的性质

回归本题，令边集大小为 k 的最小内向森林问题的答案为 $\mathrm{MDF}(k)$。我们有两个基于边集的拟阵，$\mathrm{MDF}(k) = \min_{S \in \mathcal{I}_1 \cap \mathcal{I}_2, |S| \le k} w(S)$。

我们在 $P(M_1, M_2)$ 中加入关于 k 的限制得到 $P(M_1, M_2, k)$。

令

$$P(M_1, M_2, k) = \{x \in \mathbb{R}^{|E|} : \begin{aligned} x(S) &\le \max_{T \in \mathcal{I}_1, T \subset S} |T| && \forall S \subset E \\ x(S) &\le \max_{T \in \mathcal{I}_2, T \subset S} |T| && \forall S \subset E \\ x_e &\ge 0 && \forall e \in E \\ \textstyle\sum_{e \in E} x_e &\le k && \}. \end{aligned}$$

引理 3.3. $P(M_1, M_2, k)$ 的极点全都是整点。

证明. 构造 $M_2^* = (E, \mathcal{I}_2^*)$，其中 $\mathcal{I}_2^* = \{S \in \mathcal{I}_2 : |S| \le k\}$。$M_2^*$ 是拟阵，所以 $conv(X(M_1, M_2^*)) = P(M_1, M_2^*)$。又因为 $P(M_1, M_2, k) = P(M_1, M_2^*)$，所以 $conv(X(M_1, M_2^*)) = P(M_1, M_2, k)$。 □

因此，加入大小不超过 k 的限制之后，依旧能忽略整数的限制，将问题转化为线性规划问题。令 $\mathrm{MDF}^*(i)$ 为以 $P(M_1, M_2, i)$ 为限制，以 w 为最优化函数的线性规划的值。那么，$\forall i \in \mathbb{Z}, \mathrm{MDF}^*(i) = \mathrm{MDF}(i)$。

将 $\sum_{e\in E} x_e \le k$ 置于线性规划之外，我们可得 $\mathrm{MDF}^{*\prime}(i)$ 是不减函数，所以 $\mathrm{MDF}^*(i)$ 是凸函数。因此，$\mathrm{MDF}(i)$ 也为凸函数，即 $\forall i \in \mathbb{Z}, \mathrm{MDF}(i+1) - \mathrm{MDF}(i) \le \mathrm{MDF}(i+2) - \mathrm{MDF}(i+1)$。

3.5 小结

本小节介绍了一些拟阵的知识，并将最小内向森林问题描述成拟阵交的形式。最后通过线性规划的知识得到了一个非常重要的结论：最小内向森林问题关于边数的函数是凸的。这个结论对后文有重要的帮助。

实际上，利用本节的内容，已经可以在多项式时间复杂度内解决最小内向森林问题。因为我们能将最小内向森林问题表示成拟阵交问题，并且可以在多项式时间复杂度内进行独立集的判断，所以可以直接套用带权拟阵交算法。但是复杂度过高，不推荐这种方法。

4 基于凸优化和最小树形图的算法

4.1 最小树形图问题

最小树形图问题可以被如下描述。

给定一张有向图 $G = (V, E)$，有边权 $w : E \mapsto \mathbb{R}$。对于给定的根 t，求权值最小的边子集，满足其边导出子图是一棵以 t 为根的内向树。

4.2 朱刘算法

“朱刘算法”的核心思想即为以下两个引理。

引理 4.1. 在最小树形图问题中，将一个结点的所有出边的权值同时加减，不影响最优解的树的形态。

证明. 任意一棵内向生成树的每个非根结点的出度都为 1，所以影响最优解的树的形态的因素只有每个结点出边的相对边权。 □

引理 4.2. 在最小树形图问题中，如果存在一个边权全为 0 的环，环上所有点的出边都非负，那么这个环可以视为一个结点。

证明. 这个引理非常抽象，具体来讲，该引理的意思是对于边权全为 0 的环，一定存在一种最优解使得恰有一条环上的边不被选。

因为内向树没有环，所以这个环上至少有一条边不被选。

如果存在一个最优解方案 H，使得这个环上有大于等于两条边不被选，令 S 为这些边的起点的点集，则在图 H 中，S 中至少有一个点到根的路径上不存在环上的点，令该点为 S_1。对于 H，将 $S-\{S_1\}$ 中每一个点的出边改成环上的边，得到 H^*，那么 H^* 一定是一棵内向生成树并且不劣于 H。如图 *3.1* 所示，左边是 H，右边是 H^*。 □

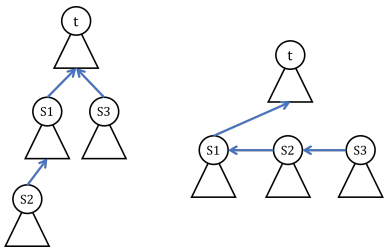


图 3.1

利用以上两个引理，就可以得到朱刘算法，以下是对其的形式化的描述。

步骤 (1) 任选一个待确定出边的结点 u。

步骤 (2) 利用同时加减 u 的出边边权，将 u 的最小的出边边权变成 0，然后将这条边作为这个结点的出边。如果出边形成了环，则将这个环缩成一个结点。

步骤 (3) 如果还有待确定出边的结点则返回**步骤 (1)**。

实现的时候，需要对每个结点建可并堆，支持全局加减并维护最小的出边，当多个结点缩成一个结点的时候将它们的堆合并，时间复杂度为 $O(E\log E)$。

4.3 最小内向森林问题的特殊情况

在 $k=|V|-1$ 时，最小内向森林问题可以很方便地转化成最小树形图问题。

当 $k=|V|-1$ 的时候，最小内向森林问题等价于要求一个不确定根的最小树形图问题。那么我们新建一个点 t 作为内向树的根，将 V 中的每一个点向其连一条权值一样且足够大的有向边。这样的话，任意一棵以 t 为根的最小内向树中，t 的入度都恰为 1，最后将这条边的权值减去即可。

4.4 凸优化

4.4.1 简介

凸优化是算法竞赛中常用的技巧，用来求解某个凸函数的某一项。

使用凸优化的条件是：对于任意的斜率，都能求出该斜率在凸函数上的切点。在算法竞赛中，求切点的方式往往是给凸函数增加一个一次函数后求全局的极值。

通过二分斜率，就能较为精确地求出凸函数上的某一点。

需要注意的是，凸优化只有在某些特定时候，才能求出凸函数上某一点的精确值。在凸优化只能求出近似值的时候，其精度可以达到指数级小。

4.4.2 在最小内向森林问题中的应用

由上述特殊情况受到启发，令 $T(\alpha)$ 表示，新建一个点 t 作为内向树的根，将 V 中的每一个点向其连一条权值为 α 的有向边后，最小树形图问题的答案。

考虑如何求解 $T(\alpha)$。让我们先枚举除了 t 结点的入边之外的边数，令其为 x，这一部分的最小代价为 $\mathrm{MDF}(x)$。此外，这 x 条边会形成 $|V|-x$ 棵彼此不连通的内向树，它们的根通过权值为 α 的边指向 t，这一部分的代价为 $(|V|-x)\alpha$。所以，$T(\alpha)$ 的值即为这两部分代价之和的最小值。

即：$T(\alpha)=\min\limits_{x\in[0,|V|)}\mathrm{MDF}(x)+(|V|-x)\alpha$。

由上一节的内容，我们可知，MDF 是一个凸函数，开口向上。我们观察这个式子，将括号展开，可以把 $T(\alpha)$ 看作是加上了斜率为 $-\alpha$ 后的最小值，也就是说可以求出 MDF 加上一个任意斜率的一次函数后的最小值。

加上一个斜率为 $-\alpha$ 的一次函数后的最小值，等同于凸函数在斜率为 α 上的切点。至此，我们就可以使用凸优化来求出 MDF 的某一项。

4.5 算法描述

我们得到了基于凸优化和最小树形图的算法，接下来对其形式化地描述。

步骤 (1) 二分斜率为 α。

步骤 (2) 新建结点 t，所有点向其连边权为 α 的有向边。

步骤 (3) 从此时起，在**步骤 (4)** 与**步骤 (5)** 中，利用同时加减一个点的出边边权，使得每个点的最小出边边权始终为 0。

步骤 (4) 任选一个待确定出边的结点 u。

步骤 (5) 选择 u 的最小的出边作为这个结点的出边。如果出边形成了环，则将这个环缩成一个结点。

步骤 (6) 如果还有待确定出边的结点则返回**步骤 (4)**。

步骤 (7) 判断根节点入度与 k 的关系，回到**步骤 (1)** 进行下一次二分。或达到目标精度退出。

4.6 小结

最小树形图算法的时间复杂度为 $O(E\log E)$，凸优化算法的时间复杂度为精度的对数。那么利用凸优化转化成最小树形图问题的时间复杂度为 $O(E\log E\log_{\frac{1}{2}}\varepsilon)$，其中 ε 为绝对精度误差除以边权的绝对值之和，即凸优化中的二分次数为 $\log_{\frac{1}{2}}\varepsilon$。

5 优先内向树扩张算法

5.1 避免新建结点

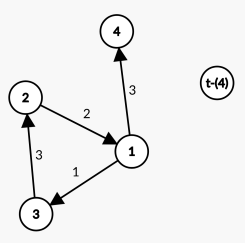
新建结点会失去很多直观的性质，所以考虑如何不新建结点。

令 3.5 中给出的算法的**步骤 (5)** 中，如果 u 选择出边后形成环并缩环，那么将这个操作称为内缩，否则称为扩张。令一组**步骤 (4)** 和**步骤 (5)** 称为一轮基础缩张操作。

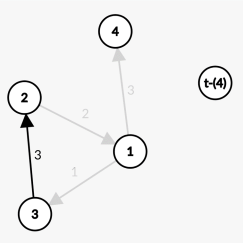
观察到，任意结点 u 进行内缩操作之后，缩成的点依旧是一个待选择出边的结点。如果对于当前的任意一棵未选择过根的内向树，不停地选择这棵内向树的根，直到其扩张为止，那么可以得到从这棵内向树引出一条出边的最小代价，将其称为这棵内向树的扩张代价。具体来说，扩张代价为，从一棵内向树的根结点未进行过基础缩张操作，到内向树引出边的所有代价，包括最初根节点的最小出边变成 0 的代价。

既然新建的结点 t 拥有每个点到其的权值相同且为 α 的边，但是新建结点的意义，并不仅限于当一棵内向树的扩张代价不小于 α 时，这棵内向树直接指向 t。

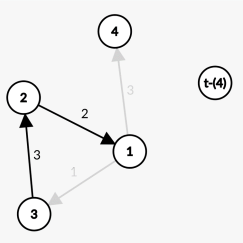
接下来给出一个例子来更好地说明这一点。



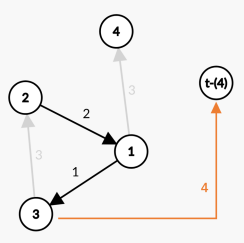
(a) 图 4.1.1



(b) 图 4.1.2



(c) 图 4.1.3



(d) 图 4.1.4

由图 4.1.1 所示，这是个四个点的图，现在新建了结点 t，且 $\alpha = 4$，即所有点到 t 都有一条权值为 4 的有向边（图上未表示出来）。

先对 3 号点进行基础缩张操作，如图 4.1.2 所示。再对 2 号点进行基础缩张操作，如图 4.1.3 所示。此时，以 1 号点为根的内向树的扩张代价为 3，比 α 要小，但对其进行基础缩张操作后的结果却是指向 t 的。造成这种现象的原因在于，这棵内向树内，存在一个结点，它在内向树上的出边非常的大，导致存在先将其内缩至根再由该点指向 t 的可能。

这使得我们需要引入一个概念：结点的预减值，记为 dec。

注意到，无论图上的边权如何变化，由同一个初始结点所引出的出边的变化值是一样的。被缩的原始结点并不是无端地消失了，而是对所有出边进行同时变化后，把出边和其他被缩点合并。那么，定义结点的预减值，为在当前状态下，将最小出边变成 0 后，每个原始结点的出边的减小值。

观察预减值在基础缩张操作中的实际表现，是将若干个集合合并，然后把合并完的集合内的所有预减值同时增加。对于缩完后的新点，我们定义这个新点的预减值为其所有结点的最大预减值。那么，在任意时候，一个结点 u 都有一条指向 t 的权值为 $\alpha - dec(u)$ 的边。

引理 5.1. *在最小内向森林问题中，给定一张带权有向图 $G = (V, E, w)$，在只进行基础缩张操作的情况下，形成了一棵内向树 H，且 H 的根的预减值是这棵内向树中最大的。那么，H 是其点导出子图的最小不定根内向生成树。*

证明. *只需要证明，当新建结点 t，所有点向 t 连一条边权足够大且为 α 的边的时候，对 H 的根不停进行基础缩张操作，那么最终会选择根直接指向 t。假设根为 r，缩点后的根的点集为 R，那么内缩操作的实质只是：不停往 R 中加入预减值更小的元素并全局加。如此反复，R 中预减值最大的始终为 r。* □

引理 5.2. *在最小树形图问题中，如果根节点 t 有且仅有其他点指向它的边权为 α 的边。在只进行基础缩张操作的情况下，对于一棵内向树 H，它的根是树上 dec 最大的结点，那么对它进行基础缩张操作后会直接指向 t 当且仅当无视 t 后其扩张代价大于 α。*

证明. *因为这样的 H 是其点导出子图中的最小不定根内向生成树。所以 H 要么对非 t 结点扩张，要么 H 的根直接指向 t。* □

那么，如果强制在根节点的预减值最大，那么 t 的意义就是对于扩张代价的限制。

得到 t 的作用后，如果我们能使得任意时刻每棵内向树的根都是预减值最大的点，那么就可以将 t 的限制显性地表示出来。要做到这一点，首先内缩操作不影响根节点的最大地位，所以只需要合理地改变扩张操作的顺序。对于两棵内向树 A, B，它们的根为 t_A, t_B，且 $dec(t_A) \le dec(t_B)$，将 A 指向 B，可以保持根的预减值最大。

对基础缩张操作稍加修改，增加对 dec 的优先级，即优先选择 dec 最小的结点。将这样的操作称为优先缩张操作。

引理 5.3. *在最小树形图问题中，如果根节点 t 有且仅有其他点指向它的边权为 α 的边。在只进行优先缩张操作的情况下，一棵内向树在最优解中会直接指向 t 当且仅当无视 t 后其扩张代价大于 α。*

证明. *只进行优先缩张操作的前提下，每一棵内向树的根的 dec 都是树中最大的。* □

现在可以形式化地描述不新建结点 t 的算法。

步骤 (1) 二分斜率为 α。

步骤 (2) 从此时起，在**步骤 (3)** 与**步骤 (4)** 中，利用同时加减一个点的出边边权，使得每个点的最小出边边权始终保持为 0，并维护预减值。

步骤 (3) 选择预减值最小的待选择出边的结点 u，判断其扩张代价若大于 α，那么直接将答案增加 $\alpha - dec(u)$，去掉这棵内向树并跳过**步骤 (4)**。

步骤 (4) 选择 u 的最小的出边作为这个结点的出边。如果出边形成了环，则将这个环缩成一个结点。

步骤 (5) 如果还有待确定出边的结点则返回**步骤 (3)**。

5.2 避免凸优化

思考扩张代价与预减值的关系，发现，扩张代价即为扩张时的根的预减值。另一件容易发现的结论是，把内向树 A 接到内向树 B 上，内向树 B 的内缩操作不变。所以对于当前扩张代价最小的内向树，可以先进行该内向树的操作直至其扩张。于是，优先缩张的选择方式求得的内向森林等同于以下的选择方式。

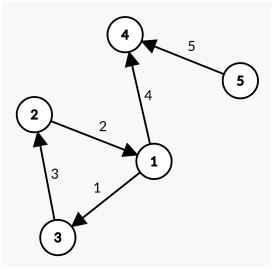
- 选择扩张代价最小的内向树，选择它的根 u。

也正因为把内向树 A 接到内向树 B 上，内向树 B 的内缩操作不变，所以内向树 B 的扩张代价不减。那么，在若干优先缩张操作中，所选的内向树的扩张代价只增不减。这时候，对于一个递增的操作序列，二分 α 就失去了意义。

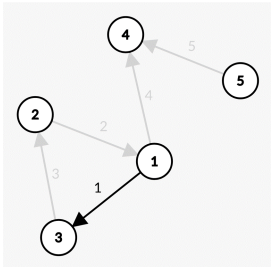
在去掉二分的同时，也不难发现，既然先前二分的位置是当前边集大小的最优解，那么这个过程中的任意时刻，都是当前边集的最优解。既然这样，那么不如顺便将任意的边集大小的答案同时求出。

5.3 具体算法与复杂度分析

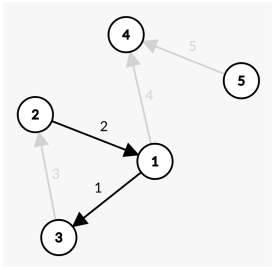
在形式化地描述之前，让我们看一个样例，回忆一下这个算法是如何运作的。



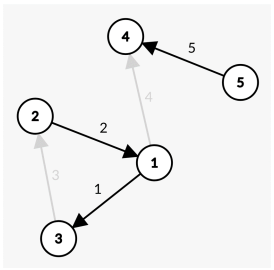
(e) 图 4.2.1



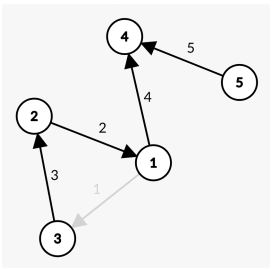
(f) 图 4.2.2



(g) 图 4.2.3



(h) 图 4.2.4



(i) 图 4.2.5

这个样例如图 4.2.1 所示，拥有 5 个点和 5 条边。起初每个点都是一棵独立的内向树，其中 1 号结点是扩张代价最小的结点，扩张后得到图 4.2.2。同理，2 号点扩张后得到图 4.2.3。

此时，拥有内向树 $\{1,2,3\}$、$\{4\}$ 和 $\{5\}$，他们的扩张代价分别为 6、$+\infty$ 和 5，故选择 5 号结点进行扩张，得到图 4.2.4。最后，选择内向树 $\{1,2,3\}$ 扩张，得到图 4.2.5。

5.3.1 算法描述

步骤 (1) 起初有 $|V|$ 个内向树，每个内向树是一个点。

步骤 (2) 令现在的边集为 E_{now}，记录 $\text{MDF}(|E_{\text{now}}|) = w(E_{\text{now}})$。

步骤 (3) 求出现在每个内向树的扩张代价。

步骤 (4) 如果没有内向树可以引出一条边则退出。

步骤 (5) 选择扩张代价最小的内向树进行若干缩张操作直至扩张，返回**步骤 (2)**。

5.3.2 复杂度分析

暴力实现的瓶颈在于求每个内向树的扩张代价。因为把内向树 A 接到内向树 B 上，内向树 B 的内缩操作不变。所以可以提前处理每一棵内向树的所有内缩操作，这样对于一棵内向树，就能 $O(1)$ 的时间内求得其扩张代价。

可以使用可并堆来实现内缩操作并维护出边，这一部分时间复杂度为 $O(E\log E)$。为了方便地寻找扩张代价最小的内向树，还需要一个堆来维护内向树的扩张代价，这一部分时间复杂度为 $O(V\log V)$。

所以总时间复杂度为 $O(E\log E)$。

5.4 正确性的另一种证明

在第 2 节中，我们可以把最小内向森林问题描述成带权拟阵交问题。

定理 5.1. *对于任意一个带权拟阵交问题，该问题在大小为 k 时有解，那么对于任意一个大小为 $k-1$ 的最优解 B，都至少存在一个大小为 k 的最优解 A，使得:*

1. 如果对于所有 (u,v) 满足 $u\in B-A, v\in A-B$ 且 $B-\{u\}+\{v\}\in\mathcal{I}_1$ 但 $B+\{v\}\notin\mathcal{I}_1$，就将 u 与 v 之间连一条边。那么，$\exists x\in A-B$，满足 $B+\{x\}\in\mathcal{I}_1$，使得 $A-B-\{x\}$ 到 $B-A$ 有完美匹配。

2. 如果对于所有 (u,v) 满足 $u\in B-A, v\in A-B$ 且 $B-\{u\}+\{v\}\in\mathcal{I}_2$ 但 $B+\{v\}\notin\mathcal{I}_2$，就将 u 与 v 之间连一条边。那么，$\exists y\in A-B$，满足 $B+\{y\}\in\mathcal{I}_2$，使得 $A-B-\{y\}$ 到 $B-A$ 有完美匹配。

上述定理就是带权拟阵交算法的一部分，所以在此不作证明。

让我们回忆一下用来描述最小内向森林问题的两个拟阵，$\mathcal{I}_1$ 是视为无向边后无环的图，$\mathcal{I}_2$ 是每个点出度为 0 或 1 的图。

观察第 2 个拟阵。如果现在已知一个大小为 $k-1$ 的最优解 B，我们利用定理知道存在一个大小为 k 的最优解 A。对于一条 u 到 v 的边，由 $B+v\notin\mathcal{I}_2$ 可知，v 的起点是 B 中一条边的起点。同理，$A-B-y$ 中的所有边的起点都是 B 中某条边的起点。所以 A 相较于 B，在出边的数量上，恰有一个点从 0 变成了 1。

引理 5.4. *在最小内向森林问题中，如果已知最优解中，拥有出边的结点的集合为 S，那么对 S 中的每个元素依次执行基础缩张操作直至扩张即可得到最优解。*

证明. *既然 $V-S$ 中的每一个点都没有出边，那么将这个集合合并成一个结点 t，原问题就转化成了最小树形图问题。* □

那么，如果已知 B，想要求解 A。可以枚举是哪一个结点的出度从 0 变成了 1，求出对其执行基础缩张操作直至扩张的代价，最后取代价最小的结点即可。这样的代价等价于一棵内向树的扩张代价。因此，该过程与优先内向树扩张算法等价。

5.5 小结

至此，我们较为完美地解决了最小内向森林问题，进一步地，该算法可以对于所有的 k 同时求出答案。

并且，这个算法或许还有继续优化的空间，例如使用软堆等算法竞赛中不常用的数据结构。

6 应用

2020 Multi-University Training Contest 4 的 J 题是该算法的优秀应用之一。我作为此题的命题人，希望通过此题来普及最小内向森林问题的解决方法。

6.1 题目大意

给定 n 个谜题，标号从 1 到 n。要求将这些谜题分成 k 个非空序列。每个谜题都有一个权值 A，初始 $\forall i, A_i = 0$。

接下来进行 m 次判定，每次判定形如 (x, l, r, c)，表示如果 x 所在的序列中，存在 $y \in [l, r]$ 出现在 x 前，那么 $A_x = \max(A_x, c)$。

制定这 k 个非空序列的排列划分方案，最大化 $\sum_{i\in[1,n]} A_i$。

6.2 建图

建图，每个谜题建立一个点，在任意两点之间连接一条权值为 0 的边。对于每次判定，将 x 向所有 $[l, r]$ 中的点连一条边权为 c 的有向边。答案即为弱连通块数量为 k 的最大内向森林。

我们接下来证明这个建图的正确性，其分为两个部分。

第一个部分是：对于任意一个弱连通块数量为 k 的内向森林，都能找到对应的 k 个序列的划分方案。实际上只需要对于每个弱连通块，将任一逆拓扑序作为序列即可。

第二个部分是：对于任意一个 k 个序列的划分方案，都能找到对应的弱连通块数量为 k 的内向森林。那么只需要对于每个谜题 i，保留任一使 A_i 最大的一条边即可。

将边权取反后，我们就将原问题转化成了最小内向森林问题。

6.3 分析

观察到，在优先内向树扩张算法中，如果不遍历所有的自环，那么在遍历至多 $O(V)$ 条边后，该算法一定会停止。因为对于每一条非自环的边，如果内缩，那么点数减少；否则如

果扩张，弱连通块数减少。所以我们的优化方向就是，在有限的时间内避免枚举过多的自环。

注意到在这张图上，虽然边数很多，如果使用 (x,l,r,c) 来描述一组边，图上的边的组数非常的少。于是对于每一个缩成的结点，用数据结构维护其初始点集，那么在询问这个点的出边时，可以非常快地找到 $[l,r]$ 中最小的不在点集里的点，以免枚举到自环。那么现在，除了每组边都会被至少被枚举一次以检验是否全是自环外，只会遍历 $O(V)$ 条边。

该算法的时间复杂度为 $O\left((n+m)\log(n+m)\right)$。

7 总结

本文围绕最小内向森林问题，先后介绍了拟阵、最小树形图问题和凸优化相关知识，并提出了优先内向树扩张算法，较好地解决了最小内向森林问题。并且，该算法代码简洁，实现细节较少，可以用来解决算法竞赛中的此类问题。在图论领域中，最小内向森林问题作为最小树形图问题的一个拓展，也有重要的理论价值。

参考文献

[1] 杨乾澜. 浅谈拟阵的一些拓展及其应用. IOI2018 中国国家候选队论文集, 2018.

[2] Michel X. Goemans. Massachusetts Institute of Technology 18.433, Combinatorial Optimization: Lecture notes on matroid intersection. MIT Mathematics, http://math.mit.edu/˜goemans/18433S11/matroid-intersect-notes.pdf.pp. 7.

[3] Wikipedia contributors. Matroid. Wikipedia, https://en.wikipedia.org/wiki/Matroid.

[4] Wikipedia contributors. Edmonds' Algorithm. Wikipedia, https://en.wikipedia.org/wiki/Edmonds%27_algorithm.

浅谈压缩后缀自动机

江苏省常州高级中学 徐翊轩

摘 要

本文从一系列传统的信息学竞赛中的后缀数据结构出发，自然地引出了一种新型后缀数据结构：压缩后缀自动机，以及它的一个变体：对称压缩后缀自动机。对于两者，本文均详细地分析了它们的性质，并且给出了一种，在允许离线的情况下，时空复杂度均为线性的构造算法，同时，也介绍了它们的实际应用。

在当今的信息学竞赛界，同学们对后缀自动机的研究热情依旧很高，希望本文引入的压缩后缀自动机，及其背后的思考方式，能够起到抛砖引玉的作用，激发更多同学对后缀数据结构，乃至更广泛的问题的思考与探究。

1 前言

后缀数据结构是当前在算法竞赛中被广泛应用的一个体系。它包含后缀数组、后缀字典树、后缀自动机、后缀树等广为人知的内容，这些后缀数据结构能高效地处理许多关于字符串的问题。

如果从自动机理论的角度来理解这些后缀数据结构的关系，便能发现：后缀字典树最小化后，可以得到后缀自动机，而将其收缩后，将会得到后缀树。而如果同时对后缀字典树进行最小化和收缩，便得到了本文将要介绍的压缩后缀自动机。

由此，笔者对压缩后缀自动机展开了研究，并写作了本文。

本文的结构如下：

第 2 节介绍了一些有关字符串和自动机的基本定义。

第 3 节回顾了 OI 中的传统后缀数据结构，并介绍了它们之间的关系。

第 4 节详细介绍了压缩后缀自动机的定义、构造方式，及应用。

第 5 节介绍了其变体，对称压缩后缀自动机的定义、构造方式，及应用。

第 6 节对文章所涉及的内容进行了分析和总结。

2 基础定义

2.1 字符串基础

令 S 为一个字符串，用 $|S|$ 表示 S 中的字符个数，即 S 的**长度**。

若 $|S|=0$，则称 S 为**空串**，记作 $S=\varnothing$。

记 $S[i]$ 表示 S 中的第 i 个字符，其中 $1\le i\le |S|$。

记 $S[l,r]$ 表示 S 中的第 l 到第 r 个字符组成的字符串，称为 S 的一个**子串**。

若 $1\le l\le r\le |S|$，则 $S[l,r]$ 是一个非空子串，否则，$S[l,r]=\varnothing$。

记 $Pre[i]=S[1,i]$，即 S 中第 i 个字符及其之前的部分，称为 S 的一个**前缀**。

记 $Suf[i]=S[i,|S|]$，即 S 中第 i 个字符及其之后的部分，称为 S 的一个**后缀**。

2.2 上下文

在本文中，我们所讨论的问题都是针对单一母串 S 的问题，因此，我们在此所规定的记号通常会略去字符串 S，而仅考虑将 S 的某个子串写入记号。

定义 2.2.1：对于字符串 S 的子串 $S[l,r]$，

定义其**上文** $LeftContext(S[l,r])$ 为：

$$S[\min\{x \mid \forall S[l,r]=S[y-(r-l),y], S[x,r]=S[y-(r-x),y]\}, r]$$

类似地，定义其**下文** $RightContext(S[l,r])$ 为：

$$S[l, \max\{x \mid \forall S[l,r]=S[y,y+(r-l)], S[l,x]=S[y,y+(x-l)]\}]$$

通俗地来讲，一个子串 $S[l,r]$ 的上文即为最长的，每当 $S[l,r]$ 在 S 中出现，其出现位置前方一定会出现的字符串与 $S[l,r]$ 拼接的结果；而 $S[l,r]$ 的下文即为最长的，$S[l,r]$ 与每当 $S[l,r]$ 在 S 中出现，其出现位置后方一定会出现的字符串拼接的结果。

在这样的表述下，不难发现

$$LeftContext(RightContext(S[l,r]))=RightContext(LeftContext(S[l,r]))$$

定义 2.2.2：对于子串 $S[l,r]$，定义其**上下文** $Context(S[l,r])$ 为：

$$LeftContext(RightContext(S[l,r]))$$

这样，一个子串的上下文即为最长的每当其在 S 中出现，便一定会出现的字符串。

定义 2.2.3：对于字符串 S 的子串 $S[l,r]$，

定义其**左集合** $Left(S[l,r])$ 为：

$$\{x \mid S[x,x+(r-l)]=S[l,r]\}$$

类似地，定义其**右集合** $Right(S[l,r])$ 为：

$$\{x \mid S[x-(r-l),x] = S[l,r]\}$$

可以发现，一个子串的左集合就是其在 S 中所有出现位置的左端点集合，而一个子串的右集合就是其在 S 中所有出现位置的右端点集合。

例 2.2：考虑字符串 $S = aabaaba$

则有

$$LeftContext(a) = a, LeftContext(aa) = aa, LeftContext(b) = aab$$

$$RightContext(a) = a, RightContext(aa) = aaba, RightContext(b) = ba$$

$$Context(a) = a, Context(aa) = aaba, Context(b) = aaba$$

对于 S 的子串 ab，其出现位置为 $\{[2,3],[5,6]\}$，因此

$$Left(ab) = \{2,5\}, Right(ab) = \{3,6\}$$

2.3 自动机

定义 2.3[1]：**确定有限状态自动机**（DFA）$\mathcal{A}$ 是由

* 一个非空有限的**状态**集合 Q

* 一个输入**字母表** Σ（非空有限的字符集合）

* 一个转移**函数** $\delta : Q \times \Sigma \to Q$（例如：$\delta(q,\sigma) = p, (p,q \in Q, \sigma \in \Sigma)$）

* 一个**开始状态** $s \in Q$

* 一个**接受状态**的集合 $F \subseteq Q$

所组成的多元组。因此一个 DFA 可以写成这样的形式：$\mathcal{A} = (Q, \Sigma, \delta, s, F)$。

简单来说，一个 DFA 可以看做是一张有限的有向图，点集为 Q，图中的每一条边上写有一个 Σ 中的字符。存在一个特殊的节点 s，称为开始状态，另外还有一系列特殊的节点 F，称为接受状态。在本文中，我们所研究的是能够接受一个字符串 S 的所有子串的自动机，因此，我们所谈论到的 DFA 均满足 $F = Q$。

3 OI 中的传统后缀数据结构

3.1 后缀字典树

为了得到一个能够接受所有 S 的子串，并且不会接受其它字符串的 DFA，不难想到对于 S 的所有后缀，建立一棵**字典树**[2]。由于字典树能够接受的字符串是集合中某一字符串的

[1]确定有限状态自动机的定义引用自参考文献 [1]

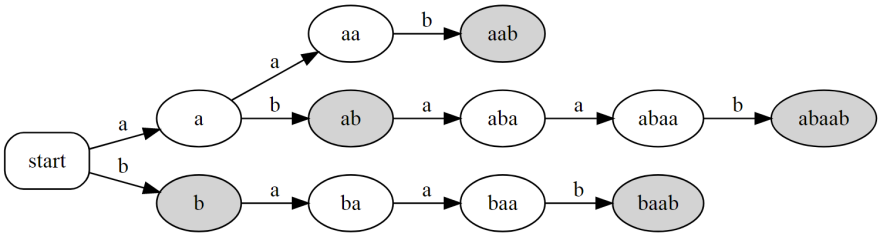
[2]字典树是一种用节点代表字符串，用边代表字符的数据结构，具体可见参考文献 [8]

前缀，而 S 所有后缀的前缀恰好构成了 S 子串的集合，因此，这样的字典树能够不重不漏地接受所有 S 的子串。

定义 3.1：

对 $\{Suf[i] \mid i \in \{1,2,\ldots,|S|\}\}$ 建立字典树，称得到的 DFA 为**后缀字典树**。

例 3.1：字符串 $S = abaab$ 的后缀字典树如下图所示：



图中标记为灰色的节点代表的字符串对应了 S 的一个后缀。

可以发现，后缀字典树能够不重不漏地接受 S 的所有子串。

3.2 后缀自动机

在最坏情况下，后缀字典树的节点数和边数均可能达到 $O(|S|^2)$ 级别，在实际应用中，往往是难以接受的。为此，我们需要寻找更为高效的后缀数据结构。

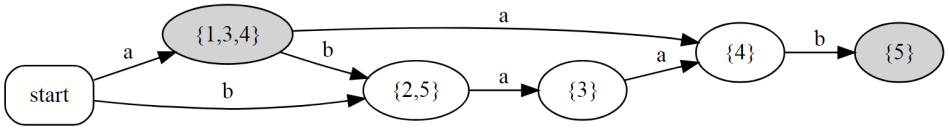
前面提到，后缀字典树是一种 DFA ，而 DFA 存在着一种**最小化**的概念[3]。

例如，在例 3.1 中，可以发现，节点 aab,baab,abaab 均没有出边。因此，可以认为，它们是等价的，从而将它们合并为同一个节点。类似地，节点 baa 和 abaa 均只有一条指向等价节点的，对应字符相同的边。因此，在这两个节点处的后续转移都是相同的，同样可以认为它们是等价的，而将它们合并为一个节点。

定义 3.2：

将后缀字典树上 *Right* 集合相同的节点合并，称得到的 DFA 为**后缀自动机**。

例 3.2.1：字符串 $S = abaab$ 的后缀自动机如下图所示：



在上面的例子中，节点上标注的集合即为该节点所代表的 *Right* 集合，除了开始状态以外，其余的出度不是 1 的节点被标记为了灰色。

可以发现，沿着后缀自动机的边进行 DFS ，能够还原出原本的后缀字典树。

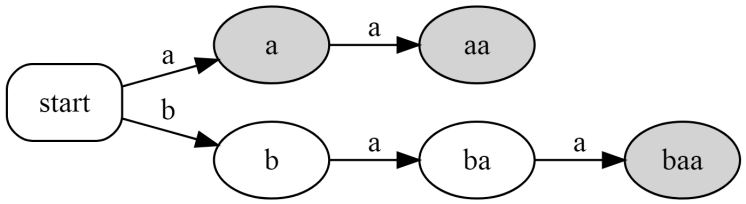
[3]有关内容可以参见参考文献 [2]

在定义 3.2 中，我们并没有直接提及对后缀字典树的最小化。但事实上，考虑对后缀字典树最小化的含义，我们希望在不影响每个节点可以接受的字符串集合的同时，尽量减少 DFA 的节点数。在匹配子串的问题中，该过程几乎就是将对应字符串的 *Right* 集合相同的节点合并为了一个。这和定义 3.2 中对后缀自动机的定义是类似的。

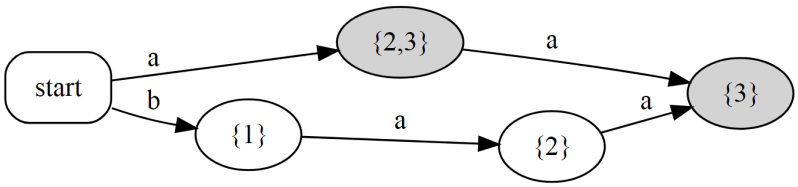
尽管如此，应当指出的是，后缀自动机并不等同于最小化的后缀字典树。

为了区分这两个概念，我们来看一个具体的例子。

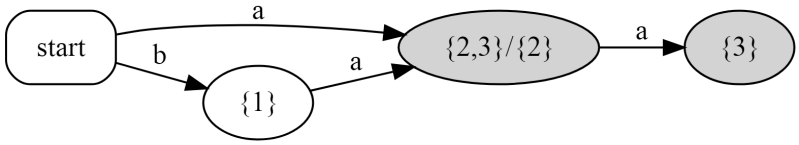
例 3.2.2：字符串 $S = baa$ 的后缀字典树如下图所示：



字符串 $S = baa$ 的后缀自动机如下图所示：



字符串 $S = baa$ 的最小化后缀字典树如下图所示：



可以发现，*Right* 集合分别为 $\{2,3\},\{2\}$ 的两个节点在最小化后缀字典树中被合并为了一个。事实上，由于一个以 $|S|$ 为右端点的出现位置右侧不再具有后续字符，如果两个 *Right* 集合仅仅相差 $|S|$ 这个元素，它们的后续转移是一样的。相反，如果两个 *Right* 集合的对称差包含除了 $|S|$ 以外的元素，它们的后续转移一定是不一样的，因此，在最小化后缀字典树中，这两个集合也会是不同的节点。

由此，我们可以看到，后缀自动机与最小化后缀字典树的唯一差别在于是否将满足对应 *Right* 集合的对称差为 $\{|S|\}$ 的节点合并起来。而如果我们在最小化时，将后缀字典树上代表一个后缀的节点，也即在图中标注为灰色的节点，看做与一般节点不同的节点，后缀自动机就和最小化后缀字典树的定义一致了。

在下文中，我们提及“后缀自动机是后缀字典树最小化的结果”时，其中“最小化”的含义均为在将后缀字典树上代表一个后缀的节点看做与一般节点不同的节点的情况下进行的最小化操作。

引理 3.2：对于 S 的任意两个子串 x, y，以下两者至少有一者成立：

$$Right(x) \cap Right(y) = \varnothing$$

$$Right(x) \subseteq Right(y) \ or \ Right(y) \subseteq Right(x)$$

证明：对于 $Right(x) \cap Right(y) \neq \varnothing$，一定存在 $s \in Right(x) \cap Right(y)$。

由此可知，x, y 均为 $Pre[s]$ 的一个后缀，从而 x, y 中有至少一者是另一者的后缀。

不失一般性地，令 x 为 y 的后缀，则有 $Right(y) \subseteq Right(x)$。

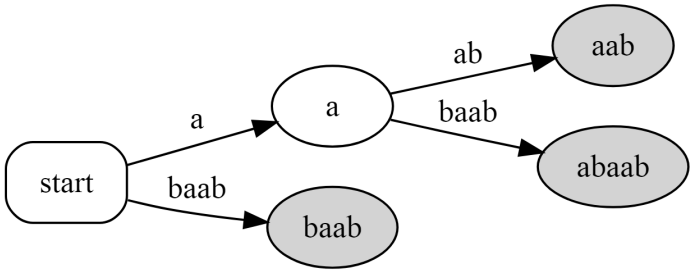
引理 3.2 的存在保证了后缀自动机的节点个数在 $O(|S|)$ 级别。事实上，一个后缀自动机至多具有 $2|S|-1$ 个节点和 $3|S|-4$ 条转移边[4]。正是因为后缀自动机优秀的时空复杂度，其在 OI 竞赛中的应用才能够如此广泛。

3.3 后缀树

想要降低后缀字典树的时空复杂度，还有着另外一种方式。注意到后缀字典树实际上是由至多 $|S|$ 条从根节点出发的路径并起来的，因此，其叶子节点的个数应当不超过 $|S|$。那么，这些叶子节点形成的**虚树**[5]节点数应当在 $2|S|-1$ 以内。

定义 3.3.1：对后缀字典树的叶子节点建立**虚树**，称得到的结果为**后缀树**。

例 3.3.1：字符串 $S = abaab$ 的后缀树如下图所示：



可以发现，建立后缀树的过程实际上是将后缀字典树上所有出度为 1 的节点收缩起来，从而达到减少后缀树节点数的效果。我们称该过程为对后缀字典树的**收缩**。

在例 3.3.1 中，一些原本代表着一个 S 的后缀的灰色节点同样被收缩简化了。而在实际处理问题的时候，我们经常会需要特别地考虑这些代表着一个后缀的节点，为此，我们可以用另一种方式定义后缀树。

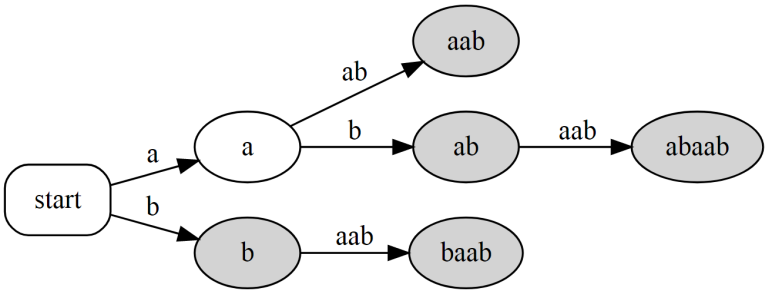
定义 3.3.2：

对后缀字典树中代表原串后缀的节点建立**虚树**，称得到的结果为**完整后缀树**。

例 3.3.2：字符串 $S = abaab$ 的完整后缀树如下图所示：

[4]有关后缀自动机的更多性质，以及线性构造方式，可见参考文献 [3],[4]

[5]一种压缩节点之间链的数据结构，详见参考文献 [9]



与后缀树相比，完整后缀树将代表原串后缀的节点全部保留了下来。

需要注意的是，由于将一些路径压缩了起来，导致一条边上可能存在多个字符，后缀树实际上不能够算作是一种 DFA 。尽管如此，后缀树依然具有着优秀的线性时空复杂度[6]，以及相比于后缀自动机更加直观的形式。正因如此，后缀树同样是 OI 竞赛中，处理字符串的有力工具。

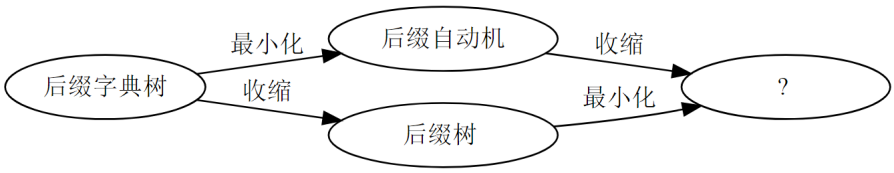
4 压缩后缀自动机

4.1 定义

对于后缀字典树进行**收缩**操作，将会得到后缀树。

对于后缀字典树进行**最小化**操作，将会得到后缀自动机。

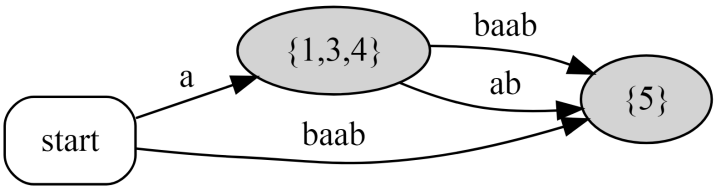
那么，如果同时对后缀字典树进行**收缩**操作和**最小化**操作[7]，将会得到什么呢？



定义 4.1.1：

同时对后缀字典树进行**收缩**操作和**最小化**操作，称得到的结果为**压缩后缀自动机**。

例 4.1.1：字符串 $S = abaab$ 的压缩后缀自动机如下图所示：



[6]有关后缀树的更多性质，以及线性构造方式，可见参考文献 [7]

[7]对于一个 DFA ，收缩和最小化操作的先后顺序是不影响结果的

对比例 3.2.1 和例 3.3.1 ，可以发现，字符串 S 的压缩后缀自动机恰好是收缩了后缀自动机上所有出度为 1 的节点的结果，同时，如果在压缩后缀自动机上 DFS ，将能够还原出字符串 S 的后缀树。

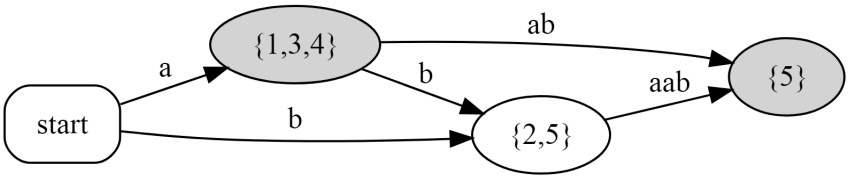
这也恰好符合了压缩后缀自动机同时对后缀字典树进行收缩和最小化操作的定义。

并且，我们也可以定义一种与完整后缀树对应的压缩后缀自动机。

定义 4.1.2：

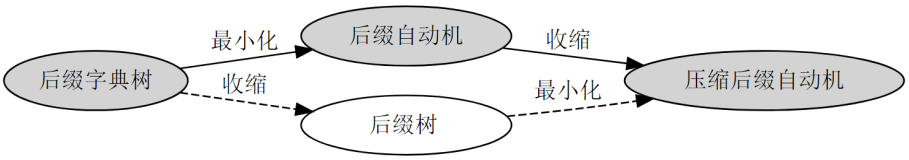
对完整后缀树进行**最小化**操作，称得到的结果为**完整压缩后缀自动机**。

例 4.1.2：字符串 $S = abaab$ 的完整压缩后缀自动机如下图所示：



4.2 构造方式

对于没有特殊性质的 DFA ，最小化的时间复杂度往往是难以接受的，但是，收缩却很容易。因此，可以对已知的后缀自动机进行收缩操作，从而得到压缩后缀自动机。



例 4.2.1：考虑如何构造字符串 $S = abaab$ 的压缩后缀自动机。

如例 3.2.1 所示，首先构造 S 的后缀自动机，并将出度不为 1 的点标记为灰色。

按照后缀自动机的拓扑序处理未被标记的，出度为 1 的节点，计算出沿着这些节点的出边将会达到的第一个灰色节点。此后，依次处理开始状态以及灰色节点的每条出边，将其简化为连向沿着该出边将会达到的第一个灰色节点的边即可。

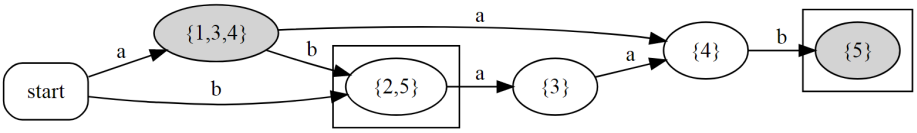
最终得到的结果如例 4.1.1 所示。

由于构造后缀自动机的时空复杂度均为 $O(|S|)$，并且收缩的过程时空复杂度同样为 $O(|S|)$ ，因此，**构造压缩后缀自动机的时空复杂度均为** $O(|S|)$ 。

但是，在完整压缩后缀自动机的定义中，并没有直接与之对应的后缀自动机用来收缩。那么，如何在避免最小化操作的情况下构造字符串的完整压缩后缀自动机呢？

例 4.2.2：考虑如何构造字符串 $S = abaab$ 的完整压缩后缀自动机。

按照定义，在完整压缩后缀自动机上 DFS 应当能够还原出完整后缀树。也就是说，除了后缀自动机上出度不是 1 的节点，我们还应当将一些额外的节点标记为灰色，使得后缀树上代表原串后缀的节点能够被保留下来。



如上图，考虑将后缀自动机上对应 *Right* 集合包含 $|S|$ 的节点同样标记为灰色。

这样的节点在后缀字典树上必定代表了一个原串的后缀，因此，将它们标记为灰色后，再进行例 4.2.1 中的收缩操作，就可以得到完整后缀树最小化的结果，即完整压缩后缀自动机。最终得到的结果如例 4.1.2 所示。

可以发现，**构造完整压缩后缀自动机的时空复杂度同样均为** $O(|S|)$ 。

4.3 性质与应用

考虑压缩后缀自动机的定义和构造方式，我们可以得到其下列基本性质：

性质 4.3.1：

压缩后缀自动机是后缀自动机**收缩**的结果。

各个节点在压缩后缀自动机上的所有入边上的字符串是其中最长者的后缀。

性质 4.3.2：

压缩后缀自动机是后缀树**最小化**的结果。

在压缩后缀自动机上 DFS ，可以还原出后缀树。

为了更好地发挥压缩后缀自动机的优势，我们还需要证明一个压缩后缀自动机的重要性质。在实际应用压缩后缀自动机解决问题时，这一性质的存在往往尤为关键。

定理 4.3：

令 L_i 表示压缩后缀自动机第 i 个节点的所有入边中最长边的长度，则

$$\sum L_i = O(|S|)$$

证明：考虑对后缀自动机收缩的过程。

对于后缀自动机上原有的一条边，有如下两种可能：

(1) 、该边出现在了灰色节点的最长入边上，此时，其对 $\sum L_i$ 的贡献为 1 。

(2) 、该边没有出现在灰色节点的最长入边上，此时，其对 $\sum L_i$ 的贡献为 0 。

因此，后缀自动机上原有的一条边至多对 $\sum L_i$ 产生 $O(1)$ 的贡献。

由引理 3.2 ，后缀自动机的边数为 $O(|S|)$ 级别，因此，$\sum L_i = O(|S|)$ 。

接下来，让我们来看一道具体的问题。

例 4.3 (Sasha and Swag Strings[8])：给定一个仅由小写字母组成的字符串 S $(1 \le |S| \le 10^5)$ ，求出其后缀树每条边上的字符串本质不同的子串的个数之和。

[8]题目来源：Petrozavodsk Summer 2015. Moscow IPT Contest

考虑这个问题的一种传统解法，注意到后缀树每条边上的字符串都是 S 的一个子串，问题可以被转化为对于 $O(|S|)$ 个 S 的子串，分别求出其本质不同的子串的个数。

对于转化后的问题，在参考文献 [5] 中，介绍了一种用 LinkCutTree 维护后缀树，并用线段树维护答案的离线解法。其时间复杂度为 $O(|S|Log^2|S|)$ 。

但是，这个解法需要应用大量的高级数据结构，实际实现的代码难度较大。同时，其时间复杂度也很高，只是刚好可以通过 10^5 级别的数据范围。接下来，我们将介绍本题的一种应用压缩后缀自动机的，更为优秀的解法。

由性质 4.3.2 ，我们知道，压缩后缀自动机是后缀树最小化的结果。因此，只需要求出压缩后缀自动机每条边上的字符串本质不同的子串的个数，就可以得到后缀树每条边上的字符串本质不同的子串的个数。

考虑对于各个节点 i ，分别求出其所有入边上的字符串本质不同的子串的个数。

由性质 4.3.1 ，这些字符串都是其中最长的字符串的后缀。

传统的求解字符串本质不同的子串的个数的方式一般是借助后缀树，或是后缀自动机，并且，是能够支持在字符串的一个方向上添加字符，同时动态维护其本质不同的子串的个数的。因此，我们可以在 $O(L_i)$ 的时间内求出点 i 的所有入边上的字符串的本质不同子串的个数。

对于所有节点，求解其入边上的字符串的本质不同子串的个数，总时间复杂度为

$$O(\sum L_i)$$

由定理 4.3 ，有

$$O(\sum L_i) = O(|S|)$$

由此，我们得到了本题的一个时空复杂度均为 $O(|S|)$ 的解法。同时，相比于前文提到的算法，运用压缩后缀自动机的算法更加易于实现。

可见，压缩后缀自动机不仅容易实现，其功能也十分强大。

5 对称压缩后缀自动机

5.1 定义

在例 3.2.1 中，我们使用输入字符串 T 的 *Right* 集合来描述后缀自动机上的节点。

考虑可以被同一节点接受的字符串 T 的集合 $\{T_i\}$ ，令其中最长的字符串为 T_{Max} ，则对于任意 $T \in \{T_i\}$ ， T 应当为 T_{Max} 的后缀[9]。

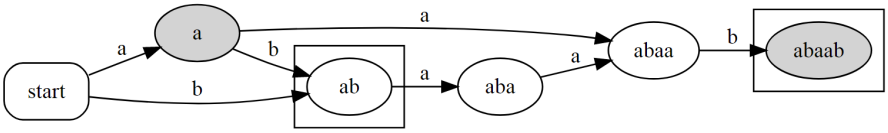
并且，考虑 *Right* 集合的含义，我们可以进一步得到：

$$\forall T \in \{T_i\}, LeftContext(T) = T_{Max}$$

[9]这同样是性质 4.3.1 成立的原因

这意味着，我们可以用一个具体的字符串 T_{Max} 来描述后缀自动机上的节点。

例 5.1.1：字符串 $S = abaab$ 的后缀自动机如下图所示：



由此，我们也可以看到，相比于后缀字典树，后缀自动机保留了满足

$$LeftContext(T) = T$$

的字符串 T 对应的节点。

后缀自动机上出度为 1 的，且对应 $Right$ 集合不包含 $|S|$ 的节点具有唯一的出边，并且不代表原串的一个后缀。这意味着其对应的字符串 T 满足

$$RightContext(T) \neq T$$

而在完整压缩后缀自动机的构造过程中，这些节点均被收缩了。

这意味着完整压缩后缀自动机上的每一个节点对应的字符串 T 应当同时满足

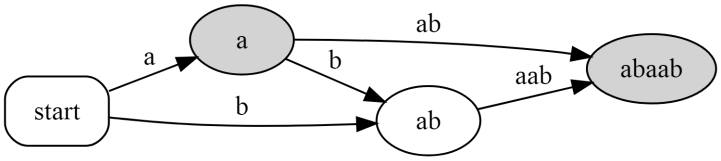
$$LeftContext(T) = T, RightContext(T) = T$$

也即

$$Context(T) = T$$

因此，我们也可以用满足 $Context(T) = T$ 的字符串 T 来描述一个节点。

例 5.1.2：字符串 $S = abaab$ 的完整压缩后缀自动机如下图所示：

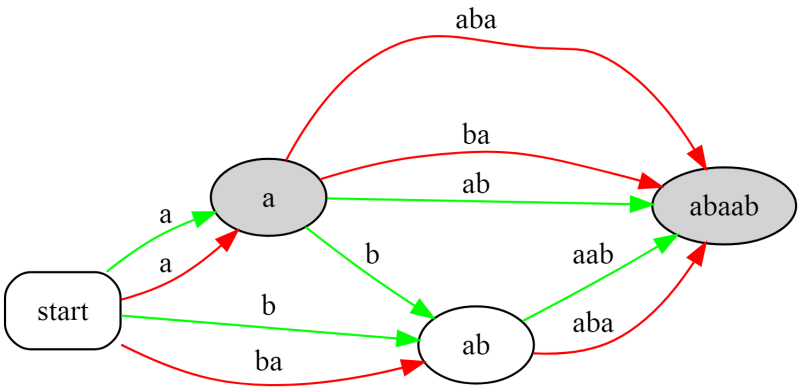


注意到满足 $Context(T) = T$ 的字符串 T 和在 S 的反串 S' 中满足 $Context(T') = T'$ 的字符串 T' 是一一对应的，我们可以同时对 S 的正反串建立完整压缩后缀自动机，并将对应的节点合并。

定义 5.1：

同时对字符串的正反串建立完整压缩后缀自动机，将对应的节点合并，并同时保留两组转移边，称得到的结果为**对称压缩后缀自动机**。

例 5.1.3：字符串 $S = abaab$ 的对称压缩后缀自动机如下图所示：



上图中，绿色的转移边是原串的转移边，代表向当前串的后侧添加字符，而红色的转移边是反串的转移边，代表向当前串的前侧添加字符。

5.2 构造方式

按照定义 5.1，我们可以通过对字符串的正反串建立完整压缩后缀自动机，再将对应的节点合并的方式来构造对称压缩后缀自动机。

那么，我们需要一个能够快速找到互为反串的子串的结构。

可以考虑使用字符串哈希[10]或是后缀数组[11]作为所需要的子串查询结构。

依照选择的子串查询结构的不同，构造对称压缩后缀自动机的时空复杂度为 $O(|S|)$ 或 $O(|S|Log|S|)$ 。

5.3 性质与应用

除了在 4.3 中提到的，压缩后缀自动机具有的性质以外，对称压缩后缀自动机还具有着**可以同时向两侧添加字符**的强大功能。接下来，让我们通过具体的例子来领略一下对称压缩后缀自动机的力量。

例 5.3.1 (子串查找[12])： 给定长度为 N $(1 \le N \le 10^6)$ 的仅由小写字母组成的字符串 S ，字符串 T 初始为空串。要求在线地支持如下四种操作共 Q $(1 \le Q \le 10^6)$ 次：

操作 (1)：给定字符 c，令 $T = T + c$

操作 (2)：给定字符 c，令 $T = c + T$

操作 (3)：给定整数 i，将 T 赋值为第 i 次操作后得到的字符串

操作 (4)：判断 T 是否为 S 的子串，如果是，求出其任意一个出现位置

对于没有操作 (2) 的问题，我们显然可以利用传统的后缀自动机来解决。

[10]一种基于哈希思想判断字符串是否相等的算法，详见参考文献 [10]

[11]一种广泛应用的处理字符串的数据结构，详见参考文献 [6]

[12]题目来源：原创

本题同时出现了向输入串的两端添加字符的操作，是传统的后缀自动机不能解决的。因此，考虑对字符串 S 建立对称压缩后缀自动机。考虑用 $Context(T)$ 对应的节点来表示字符串 T 。同时，维护 T 在 $Context(T)$ 中的出现位置，注意由于上下文的性质，T 在 $Context(T)$ 中的出现位置是唯一的。

对于操作 (1)：

若 $RightContext(T) \neq T$ ，则 T 在 S 中出现时，

其后方一定会出现一个特定的字符，判断 c 是否为该字符。

若是，则 $Context(T+c) = Context(T)$ ，否则，$T+c$ 不是 S 的子串；

若 $RightContext(T) = T$ ，则 $Context(T+c) \neq Context(T)$ 。

找到在正向压缩后缀自动机中以字符 c 开头的出边。

若存在，则 $Context(T+c)$ 即为到达的节点，否则，$T+c$ 不是 S 的子串。

类似地，对于操作 (2)：

若 $LeftContext(T) \neq T$ ，则 T 在 S 中出现时，

其前方一定会出现一个特定的字符，判断 c 是否为该字符。

若是，则 $Context(c+T) = Context(T)$ ，否则，$c+T$ 不是 S 的子串；

若 $LeftContext(T) = T$ ，则 $Context(c+T) \neq Context(T)$ 。

找到在反向压缩后缀自动机中以字符 c 开头的出边。

若存在，则 $Context(c+T)$ 即为到达的节点，否则，$c+T$ 不是 S 的子串。

考虑操作 (3)，由于对于一个状态，我们仅仅需要维护 $Context(T)$ 对应的节点，以及 T 在 $Context(T)$ 中的出现位置，因此，只需要用数组记录下每次操作后的这些信息，就可以支持操作 (3) 。

最后，对于操作 (4)，我们已经维护出了 $Context(T)$ ，以及 T 在 $Context(T)$ 中的出现位置。因此，判断 T 是否为 S 的子串即为判断是否存在 $Context(T)$ 对应的节点，而求出 $Context(T)$ 的任意一个出现位置，即可求出 T 的一个出现位置。

对于对称压缩后缀自动机上的每一个节点，我们可以在构造的时候预处理出其在 S 中的出现位置，从而支持操作 (4) 。该解法的时间复杂度为 $O(N+Q)$ ，空间复杂度为 $O(N)$ 。可见对称压缩后缀自动机在处理双向添加字符类型的问题时的优势所在。

例 5.3.2 (Alice and Bob and A String[13])：

给定一个仅由小写字母组成的字符串 S $(1 \leq |S| \leq 10^5)$ 。

Alice 和 Bob 正在玩一个游戏，初始时，他们有一个 S 的子串 T 。由 Alice 先手，两人轮流进行如下操作：在 T 的前后各添加一个字符，保证得到的结果仍然是 S 的一个子串。无法操作的玩家负。

假设 Alice 和 Bob 均采取使自己获胜的最优策略，则对于所有 S 的子串，请帮助 Alice 计算出，其中使她必胜的，字典序第 k $(1 \leq k \leq 10^{10})$ 小的 T ，或者判断不存在 k 个使得 Alice 必胜的 T 。

[13]题目来源：Petrozavodsk Summer 2018. Moscow IPT Contest

首先，考虑如何对一个给定的字符串 T ，判断 Alice 是否能够取胜。

对于这个问题，最直观的想法应当是从长到短考虑每一个 S 的子串 T ，记 dp_T 表示子串 T 是否为先手必胜态，如果 T 可以达到一个先手必败态，则 $dp_T = true$ ，否则，即 T 可以达到的状态均为先手必胜态，$dp_T = false$ 。

然而，该做法需要考虑 S 的所有子串，在 $|S|$ 达到 10^5 级别时是无法接受的。

考虑一个同时满足 $LeftContext(T) \neq T, RightContext(T) \neq T$ 的字符串 T 。每当 T 在 S 中出现时，其前后必然会出现固定的字符，因此，对于当前串为 T 的状态，先手玩家只有唯一的选择。那么，我们可以按照唯一的选择推进游戏，直到 $LeftContext(T) = T, RightContext(T) = T$ 中的至少一者成立。

由定理 4.3 ，满足以上至少一者的字符串数量在 $O(|S|)$ 级别。

由此，建立对称压缩后缀自动机后，只需要在这样的字符串上进行上述 DP 即可。

接下来，考虑如何求出字典序第 k 小的先手必胜态。

由性质 4.3.2 ，若在压缩后缀自动机上 DFS ，我们可以还原出原串的后缀树。而后缀树恰好将 S 的所有子串按照字典序排好了序。因此，如果我们能够快速求出一条边上的必胜态总数，便只需要在压缩后缀自动机上 DFS ，就可以确定所求的答案在哪一条边上，从而确定答案。

由转移规则，可以发现，一条边上的必胜态总数可以通过对于节点 X ，维护满足 $Context(T) = LeftContext(T) = X$ ，或者 $Context(T) = RightContext(T) = X$ 的字符串 T 胜负情况的，奇偶长度分别的前缀和而快速计算。

由此，我们得到了问题的一个时间复杂度为 $O(\sigma \times |S|)$ 的解法，其中 $\sigma = 26$ 。

6 总结

本文从一系列传统的信息学竞赛中的后缀数据结构出发，自然地引出了一种新型后缀数据结构：压缩后缀自动机，以及它的一个变体：对称压缩后缀自动机。对于两者，本文均详细地分析了它们的性质，并且给出了一种，在允许离线的情况下，时空复杂度均为线性的构造算法，同时，也介绍了它们的实际应用。

在本文中，笔者利用大量的配图进行辅助讲解，语言也通俗易懂。笔者认为，即使是字符串基础较差的选手，阅读本文后，也能够有一定的收获。

值得一提的是，本文所介绍的压缩后缀自动机，实际上就是对大家熟知的后缀树、后缀自动机进一步考虑后的自然结果。这充分说明了，许多时候，解决新问题的方式往往藏于已有的工具中，只要我们勇于创新，积极思考，便能够将它们发现。

笔者注意到，在当今的信息学竞赛界，同学们对后缀自动机的研究热情依旧很高，希望本文引入的压缩后缀自动机，及其背后的思考方式，能够起到抛砖引玉的作用，激发更多同学对后缀数据结构，乃至更广泛的问题的思考与探究。

感谢

感谢中国计算机学会提供学习和交流的平台。
感谢江苏省常州高级中学的曹文老师，吴涛老师多年来给予我的关心和指导。
感谢郭晓旭前辈对本文做出的帮助与指导。
感谢集训队教练高闻远对我的帮助与指导。
感谢王修涵同学、杨俊昭同学和我校的杜伟桦同学为本文审稿。
感谢家人、朋友对我的支持与鼓励。
感谢帮助过我的老师、同学们。

参考文献

[1] 维基百科：确定有限状态自动机：https://zh.wikipedia.org/zh-hans/确定有限状态自动机.

[2] 维基百科：确定有限状态自动机最小化：https://zh.wikipedia.org/zh-hans/确定有限状态自动机最小化.

[3] 陈立杰《后缀自动机》2012 年 NOI WC 讲课课件.

[4] 刘研绎《后缀自动机在字典树上的拓展》2015 国家集训队论文集.

[5] 陈江伦《后缀树结点数 命题报告及一类区间问题的优化》2018 国家集训队论文集.

[6] 罗穗骞《后缀数组 —— 处理字符串的有力工具》2009 国家集训队论文集.

[7] 东南大学出版社《高级数据结构》.

[8] 李煜东《算法竞赛进阶指南》.

[9] OI Wiki《虚树》https://oi-wiki.org/graph/virtual-tree/.

[10] OI Wiki《字符串哈希》https://oi-wiki.org/string/hash/.

浅谈 Nimber 和多项式算法

宁波市镇海中学 罗煜翔

摘 要

本文从 Nimber 的定义出发，介绍了 Nimber 的快速运算，Nimber 多项式的乘法、分治算法、复合、牛顿迭代、除法、多点求值、快速插值、欧几里得算法、中国剩余算法、Nimber 二项卷积、ln 和 exp、三角函数这些问题的定义和快速算法。

引言

组合游戏或者说博弈论是 OI 中的一个重要考点，在各大比赛中常常出现。目前的考察重点以 SG 定理为主，但相关的扩展主要以数据结构优化为主。

多项式问题也是 OI 中的一个重要考点，但笔者发现一些多项式方面的算法[1]虽然也已经被引入但并不普及，导致出现了使用其中一些算法的特例出题的情况。

于是，笔者希望通过本文，综合组合游戏和多项式相关的内容，介绍一些相关的算法和问题。

本文的第一节会给出 Nimber 的定义和相关问题，第二节讨论 Nimber 生成函数及其各种运算，第三节讨论 Nimber 指数生成函数及其各种运算。

1 Nimber

1.1 Nimber 和 Sprague-Grundy 定理

1.1.1 组合游戏

首先我们定义一般规则下的公平组合游戏为满足如下条件的组合游戏：

1. 游戏由两个玩家轮流进行，直到无法进行操作时的操作者输。

[1]如有限域上的多项式因式分解

2. 游戏允许进行的操作只取决于当前的状态，不取决于操作者是谁。游戏的所有信息对双方公开。

3. 游戏的总状态和总操作有限，所有的操作对状态的影响是确定性的。

我们可以用后继状态的集合表示游戏中的一个状态。例如无法操作为 $\varnothing$，只能操作到无法操作为 $\{\varnothing\}$。在不引起混淆的情况下，我们也直接用这个状态表示一个组合游戏。

定义单堆的 Nim 游戏为：有一堆石子，每次的操作可以从中取任意正整数数量的石子的一般规则公平组合游戏。

我们定义 Nimber $*n$ 为大小为 n 的石子的单堆 Nim 游戏。形式化的说，Nimber 由以下规则归纳定理 $*0 = \varnothing$ 且对 $n \geq 0$ 有 $*(n+1) = *n \cup \{*n\}$。在不会导致混淆的情况下，$*$ 可以省略。

对于两个组合游戏 A, B，我们可以定义他们的和：

$$A \oplus B = \{a \oplus B | a \in A\} \cup \{A \oplus b | b \in B\}$$

即每次只能在一个游戏中进行操作。显然其满足交换律和结合律。

容易发现，单堆 Nim 游戏的胜负情况只和 n 是否为 0 有关，但不难发现这些状态并不等价。这表明两个组合游戏的等价不能只看胜负情况。

于是我们定义两个组合游戏 A, B 等价，当且仅当对于任意的组合游戏 C，均有 $A \oplus C$ 和 $B \oplus C$ 的胜负情况相同。记作 $A \approx B$。

1.1.2 Sprague-Grundy 定理

Sprague-Grundy 定理指出，任意的公平组合游戏和一个 Nimber 等价。所以公平组合游戏的研究可以转化为 Nimber 的研究。由于和本文内容关系不大，Sprague-Grundy 定理的证明这里省略[2]。

我们将这个组合游戏等价的 Nimber 称为该组合游戏的 SG 函数。计算方法为所有后继状态的 SG 函数中，最小没出现的非负整数[3]。

定义两个 Nimber 的 Nim 和为这两个 Nimber 的游戏的和所等价的 Nimber，不难证明这就是将 Nimber 的数值的按位异或的结果。

1.2 Nim 积

Nim 积，可以看做是 Nim 和的扩展，其定义式如下：

$$x \otimes y = \text{mex}\{(a \otimes b) \oplus (a \otimes y) \oplus (x \otimes b) \,|\, 0 \leq a < x \wedge 0 \leq b < y\}$$

[2]证明可以参见参考文献 [3]

[3]也称为 mex 运算

这相当于是每次操作把 $(a,b)-(x,y)$ 这个矩形的四个顶点翻转颜色的组合游戏的 SG 函数。

可以证明，所有 $[0,2^{2^m})$ 中的 Nimber 构成了一个有限域，全体的 Nimber 构成代数闭域。证明较为复杂，这里略去。[4]

对于计算两个数的 Nim 积，需要利用如下结论，证明这里同样略去。

$$2^{2^n}\otimes 2^{2^m}=\begin{cases}2^{2^n}\cdot 2^{2^m} & \text{if } n\neq m\\ 3\cdot 2^{2^n-1} & \text{if } n=m\end{cases}$$

考虑利用这一结论求两个数的 Nim 积。

设 $x=a\cdot 2^{2^{m-1}}+b, y=c\cdot 2^{2^{m-1}}+d$，其中 $a,b,c,d\in[0,2^{2^{m-1}}), x,y\in[0,2^{2^m})$。

设 $n=2^{m-1}$，则不难证明有 $2^n\otimes a=2^n\cdot a$，于是：

$$\begin{aligned}x\otimes y=(a\otimes 2^n\oplus b)\otimes(c\otimes 2^n\oplus d)&=a\otimes c\otimes 3\cdot 2^{n-1}\oplus(a\otimes d\oplus b\otimes c)\otimes 2^n\oplus b\otimes d\\&=a\otimes c\otimes 2^{n-1}\oplus(a\otimes d\oplus b\otimes c\oplus a\otimes c)\cdot 2^n\oplus b\otimes d\end{aligned}$$

注意到 $a\otimes d\oplus b\otimes c\oplus a\otimes c=(a\oplus b)\otimes(c\oplus d)\oplus b\otimes d$。

于是进行 4 次 $m-1$ 的递归即可，时间复杂度为 $O(4^m)$。事实上，注意到其中一次递归为 2 的幂和另一个数的 Nim 积。在 x 是 2 的幂的情况下，可以发现上面的 a,b 中有一个是 0，另一个也是 2 的幂。于是进行的 4 次递归中，一次 $x=0$ 从而结果也为 0，另外三次 x 是 2 的幂，则每次只会进行三个递归，时间复杂度为 $O(3^m)$。

于是考虑总的过程，则有 $T(m)=3T(m-1)+O(3^m)$，解得 $T(m)=O(m\cdot 3^m)$。事实上，由于 $O(3^m)$ 实际上是 $m-1$ 的大小的复杂度，所以常数其实只有 $\frac{1}{3}$。

由于这一算法十分的重要，所以这里给出伪代码。事实上，$m\cdot 3^m$ 和 4^m 的差距并没有多大，不过由于代码上只差一个 if，所以这里给出的是 $O(m\cdot 3^m)$ 的版本。

Algorithm 1 NimMul(x,y,m)

```
if x = 0 ∨ y = 0 then
    return 0
end if
if m = 0 then
    return 1
end if
n = 2^{m-1}, a = ⌊x/2^n⌋, b = x mod 2^n, c = ⌊y/2^n⌋, d = y mod 2^n.
ac = NimMul(a, c, m − 1), bd = NimMul(b, d, m − 1).
return (NimMul(a ⊕ b, c ⊕ d, m − 1) ⊕ bd) · 2^n + (NimMul(2^{n-1}, ac, m − 1) ⊕ bd).
```

注意到 $5\cdot 3^4=405$ 仍然比较大，考虑对这个算法进行进一步的优化。

[4]相关的证明可见参考文献 [2]

一个直接的想法是进行记忆化，由于 x, y 共有 $2^{2^{m+1}}$ 种，在 $m \le 3$ 的时候记忆化是可以接受的。

更进一步的想法是注意到有限域中原根[5]的存在，可以预处理 $[0, 2^{2^m})$ 范围内的指数/对数表。于是可以处理 $m \le 4$ 的情况。使用了上述优化后，在 UOJ 上进行 10^7 次 Nim 积大约需要 400ms。

于是 $m = 5$ 的情况只需要一次递归就能得到答案，可以认为时间复杂度是 $O(1)$ 的。由于 $[0, 2^{32})$ 在 C++[6] 中就是 unsigned int 的表示范围，所以下文如无特殊说明只考虑这个范围内的 Nimber。

接下来的部分，如无特别说明，则 a^b 在 a 是具体数字或时间复杂度中时表示数字的乘方，在 a 是变量或者表达式时表示 Nim 幂，即做 b 次 Nim 积后的结果。

1.3 Nimber 开方与求逆

考虑解方程 $x^2 = a$ 和 $a \otimes x = 1$。

由有限域的基本性质，$x^{2^{32}} = x$。

从而一组可行解是 $a^{2^{31}}$ 和 $a^{2^{32}-2}$（$a \neq 0$），利用快速幂计算即可。

解的唯一性是不难证明的。

1.4 Nimber 积和式

给定一个 $n \times n$ 的二维数组 $a_{i,j}$，求 $\bigoplus_p \bigotimes_{i=1}^{n} a_{i,p_i}$，这里的 p 取遍所有 $1 \sim n$ 的排列。

由于 $x \oplus x = 0$，所以积和式就是行列式。

使用高斯消元法计算即可。

1.5 Nimber 多项式方程

现在考虑解一个一般的多项式方程 $\bigoplus_{i=0}^{n} a_i \otimes x^i = 0$。这个方程可能有 $x \ge 2^{32}$ 的根，但这里只考虑求 $[0, 2^{32})$ 中的根。

首先将原多项式和 $x^{2^{32}} \oplus x$ 求 gcd，则求完 gcd 的结果是若干个互不相同的一次式的乘积。

设 $F(x) = \bigoplus_{i=1}^{31} x^{2^i}$，则 $F(x)\,(F(x) \oplus 1) = x^{2^{32}} \oplus x$。

于是随机一个多项式 l，则 $\gcd\,(F(l(x)), G(x))$ 至少有 $\frac{1}{2}$ 的概率找到 G 的一个非平凡因式。

而 $F(l(x))$ 的计算可以类似快速幂的方法计算。

[5]对于 $m = 4$ 的情况，最小的原根为 258

[6]这里指 NOI Linux 上的 C++

时间复杂度不超过 $O(32n^2 \log n)$。

1.5.1 例 1. 小 Q 和小 Y 做游戏（一）

小 Q 和小 Y 是组合带师。

这天，祂们发现了二维平面第一象限中[7]的一个点 (a,b)，于是祂们决定来进行一次游戏。

祂们首先把这个点染成了黑色，并把其他点染成了白色。并从小 Q 开始轮流操作：

选择一个顶点均为非负整数、边平行于坐标轴且右上角是黑点的矩形[8]，将它的四个顶点翻转颜色。如果没有这样的矩形，则操作者输。

由于小 Y 非常希望胜利，所以祂准备作弊。祂可以在小 Q 开始操作前，选择一条第一象限中平行于 x 轴的直线，将它与直线 $x=a$ 与 $y=x$ 的两个交点依次翻转颜色[9]。

于是小 Y 准备寻求你的帮助，希望你求出应该取哪条直线。当然，小 Y 考虑到了你的计算能力，所以如果 $(0,2^{32})$ 中无解则只需要输出无解。

如果有多解求出任意一个均可。

$1 \le a,b < 2^{32}$。

1.5.2 解法

不难发现，问题相当于解方程 $x^2 \oplus a \otimes x \oplus a \otimes b = 0$。

普通的解二次方程方法是基于配方的，但对 Nimber 来说配方消不掉一次项。

事实上，这里直接套用 Nimber 多项式方程的解法就行了。

时间复杂度 $O(32)$。

2 Nimber 生成函数

现在，我们考虑 Nimber 多项式和 Nimber 生成函数。

首先要解决的问题就是多项式的乘法。

2.1 Nimber 多项式乘法

主要有三种方法可以解决这个问题。

[7]这里认为坐标轴上的点不在任何象限

[8]这里认为矩形的面积需要为正

[9]如果这两个交点是同一个则什么也不做

2.1.1 Karatsuba 算法

很容易想到可以利用 Karatsuba 算法解决这个问题。

由于 Karatsuba 算法在 OI 中已经比较普及，这里只简单的叙述一下过程。

每次将两个多项式拆成前一半和后一半，全部展开合并同类型后只需要计算最高项，最低项和三项的和，于是进行三次一半大小的递归即可。

时间复杂度为 $O\left(n^{\log_2 3}\right) \approx O\left(n^{1.585}\right)$。

2.1.2 FFT

有限域 $\mathbb{F}_{p^k}$ 的一种构造是：先选择一个模 p 意义下的 k 次既约多项式 $f(x)$，则对于所有模 p 的 p^k 个 $k-1$ 次多项式，乘法定义为模 $p, f(x)$ 的乘法。

在将 Nimber 转换成这种形式后我们可以用 FFT 做二元多项式乘法解决这个问题。

一种转换的方法是，找到 $\mathbb{F}_{2^{32}}$ 的一个原根，设其为 x，利用 $x^0, x^1, \cdots, x^{32}$ 进行高斯消元法解出这个 32 次多项式。

由于二元多项式乘法需要长度扩大 4 倍，且最后还需要做 $O(n)$ 次多项式取模，所以常数比较大，甚至不一定能跑过 Karatsuba 算法。

2.1.3 Cantor 算法

对于一般的多项式乘法问题，有时间复杂度为 $O(n \log n \log\log n)$ 的算法，下面就介绍其中的一种。

简单来说，这个算法直接在 FFT 的过程中维护单位根的和。

由于 $x \oplus x = 0$，这导致无法进行二进制的 IDFT，所以考虑使用三进制。[10]

考虑计算两个多项式乘积模 $x^{2\times 3^m} \oplus x^{3^m} \oplus 1$ 的结果。事实上，取模的多项式是分圆多项式，下面的计算过程的正确性需要用它来保证。

设 $v = \left\lfloor \frac{m}{2} \right\rfloor, u = \left\lceil \frac{m}{2} \right\rceil, n = 3^m, p = 3^u, q = 3^v$，则有：

$$\bigoplus_{i=0}^{2n-1} a_i x^i = \bigoplus_{j=0}^{q-1} \left(\bigoplus_{i=0}^{2p-1} a_{iq+j} x^{iq} \right) \otimes x^j$$

设 $x^q = y$，则可以看做是 x 次数为 $q-1$ 的二元多项式，且 y 的部分需要对 $y^{2p} \oplus y^p \oplus 1$ 取模。

对两个多项式做乘法，由于 $y^{2p} \oplus y^p \oplus 1$ 是 $y^{3p} \oplus 1$ 的因式，可以认为 y 就是 $3p$ 次单位根。

现在有两个长度为 $q-1$ 的多项式，做乘法的直接想法是做长度至少为 $2q-1$ 的循环卷积。由于需要做三进制 FFT，所以需要长度为 $3q$ 的。由于 $p \geq q$，所以 $3q$ 次单位根可以用 y 表示。

[10] 对于一般问题，需要做二、三进制并用扩展欧几里得算法合并

考虑做 FFT 需要的操作，交换元素和加减法可以用多项式加减法处理，而乘上单位根只需要做一次平移然后取模。由于取模的多项式非常简单所以都是 $O(p)$ 的。

在做完 DFT 之后，需要对应项做点积，这其实就是 u 大小的子问题，递归处理。

IDFT 和 DFT 是类似的，事实上 IDFT 就是 DFT 后翻转除了首项的一段区间。

时间复杂度 $T(n)=3\sqrt{n}T(\sqrt{n})+O(n\log n)$，$T(n)=O\left(n\log^{1+\log_2 3}n\right)$。

发现复杂度不优的原因是明明两边相乘长度只有 $2q$ 却要补到 $3q$，考虑直接做长度为 q 的循环卷积。

设 $C(x)=A(x)\otimes B(x)$，即原来两个多项式的乘积。

考虑计算 $D(x)=A(x)\otimes B(x)$ 和 $E(x)=A(x\otimes\omega_{3q})\otimes B(x\otimes\omega_{3q})$，这里的卷积是循环卷积。

则 $d_i=c_i\oplus c_{i+q},e_i\otimes\omega_{3q}^{-i}=c_i\oplus(c_{i+q}\otimes\omega_3)$。

显然只需要求出 $\omega_3\oplus 1$ 的逆即可，注意到 $\omega_3\oplus 1=\omega_3^2$，所以逆就是 ω_3。

于是，时间复杂度为 $T(n)=2\sqrt{n}T(\sqrt{n})+O(n\log n)$，得 $T(n)=O(n\log n\log\log n)$。

使用这个算法进行模素数的多项式乘法，在 UOJ 上对 $n=100000$ 的数据大约需要 70ms，大约为常数极好的拆系数 FFT/三模数 NTT [11]的 1.5 ~ 2 倍。

2.2 离散 Fourier 变换

虽然 Nimber 数不能做 2 的幂的 DFT，但由于原根存在，依然可以做一些长度的 DFT。

确切地说，对于 $n|2^{32}-1$，长度为 n 的 DFT 都是可以定义的。而且对于 $0\le i\le 31$，$[2^i,2^{i+1})$ 中恰好存在一个 n 可以做 DFT。

这一点可以枚举验证，也可以利用 $2^{32}-1$ 的质因子都是费马素数证明。

现在考虑如何求一个一般长度的 DFT：

$$b_i=\bigoplus_{j=0}^{n-1}a_j\otimes\omega_n^{i\cdot j}$$

利用等式 $\binom{i+j}{2}-\binom{i}{2}-\binom{j}{2}=i\cdot j$，得：

$$b_i\otimes\omega_n^{\binom{i}{2}}=\bigoplus_{j=0}^{n-1}a_j\otimes\omega_n^{-\binom{j}{2}}\otimes\omega_n^{\binom{i+j}{2}}$$

于是做一次减法卷积即可，这可以通过翻转 a 变成加法卷积实现。

而 IDFT 也没什么区别，事实上在 DFT 的结果上直接反转 $1\sim n-1$ 的值即可。

2.3 Nimber 半在线卷积

定义半在线卷积问题为：

[11]这方面的快速算法可以见参考文献 [7]

计算 $f=g\otimes h$，其中 g 已经给出且常数项为 0，而 h_i 将在计算出 f_i 后给出[12]。

容易想到用分治算法解决这个问题：

首先递归计算左边一半的 f_i，然后用左边的 h_i 更新右半边的 f_i，最后递归处理右边即可。

时间复杂度为 $O(n\log^2 n\log\log n)$。

考虑每次不是分成两段，而是分成 K 段，则每两段之间都需要进行转移，共有 $O(K^2)$ 次转移。

但是注意到多项式的个数只有 $O(K)$ 种，于是进行 DFT 之后转移就是线性的。

于是每层的时间复杂度就变成了 $O\left(n\log\frac{n}{K}\log\log\frac{n}{K}+nK\right)$，取 $K=O\left(\log n\log\log n\right)$ 时，每一层所花的时间还是 $O(n\log n\log\log n)$。但这样每次就可以分为 K 个子问题。

于是总的时间复杂度变为了 $O\left(\frac{n\log^2 n\log\log n}{\log\log n}\right)=O\left(n\log^2 n\right)$。

注意到转移的过程依然是若干次半在线卷积，可以做到 $O\left(n\log^{1+\varepsilon}n\right)$，但由于一些常数原因，实践中和 $O\left(n\log^2 n\right)$ 区别不大。

2.4 Nimber 多项式复合

计算 $(f\circ g)(x)\bmod x^{2^k}$，其中 $f\circ g$ 表示 f 和 g 的复合，即 $f(g)$。

设 $f(x)=f_0\left(x^2\right)\oplus\left(x\otimes f_1\left(x^2\right)\right)$，于是只需要计算 $f_0\circ g^2, f_1\circ g^2$。

由于 g^2 的奇数项均为 0，所以可设 $g^2(x)=h\left(x^2\right)$，于是只需要计算 $f_0(h), f_1(h)\bmod x^{2^{k-1}}$。

时间复杂度为 $O\left(n\log^2 n\log\log n\right)$。

2.5 牛顿迭代

给定一个多项式 $F(t)$，你需要求一个多项式 $X(x)$ 使得 $(F\circ X)(x)\bmod x^n=0$ 且 X 的常数项为 0。要求 X 常数项为 0 才能对形式幂级数比较准确的定义复合，但这可能导致公式中大量出现 $X(x)\oplus 1$。

事实上，传统的牛顿迭代法在这里还是适用的。

首先是牛顿迭代公式 $X_{t+1}=X_t\oplus\frac{F\circ X_t}{F'\circ X_t}\bmod x^{2^{t+1}}$，其中 X_t 表示 $X\bmod x^{2^t}$，F' 为导数，实现上就是去掉偶数项后除以 x。

于是第一个问题做多项式求逆，只需要解方程：$G(x)=\frac{1}{1\oplus X(x)}=1\oplus X(x)\oplus X(x)^2\oplus\cdots$。

带入上面的公式并化简得：$X_{t+1}=1\oplus\left(X_t\oplus 1\right)^2\otimes G\bmod x^{2^{t+1}}$。

时间复杂度为 $O\left(n\log n\log\log n\right)$。

于是对于一般的问题，时间复杂度为 $O\left(n\log^2 n\log\log n\right)$。如果函数比较特殊，复合计算较快则为 $O\left(n\log n\log\log n\right)$。

[12] 可以理解为 h_i 的计算依赖 f_i

2.6 Nimber 多项式带余除法

给定 n 次多项式 $A(x)$ 和 m 次多项式 $B(x)$，要求 $n-m$ 次多项式 $C(x)$ 和次数不超过 $m-1$ 次多项式 $D(x)$，使得

$$A(x) = B(x) \otimes C(x) \oplus D(x)$$

考虑将 x 用 $\frac{1}{x}$ 来代，并在两边乘 x^n，得：

$$x^n \otimes A\left(\frac{1}{x}\right) = \left(x^m \otimes B\left(\frac{1}{x}\right)\right) \otimes \left(x^{n-m} \otimes C\left(\frac{1}{x}\right)\right) \oplus x^{n-m+1} \otimes \left(x^{m-1} \otimes D\left(\frac{1}{x}\right)\right)$$

两遍模 x^{n-m+1}，则只需要进行多项式求逆和乘法就能求出 C，之后求 D 也是容易的。

2.7 Nimber 线性递推

对于递推数列：$f_m = \bigoplus_{i=1}^{d} a_i \otimes f_{m-i}$，给定 $\forall 1 \le i \le d, a_i, f_{i-1}$，求 f_n。

考虑构造矩阵：

$$\mathbf{M} = \begin{pmatrix} a_1 & a_2 & \cdots & a_{d-1} & a_d \\ 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \end{pmatrix}$$

我们可以得到 $\mathbf{F}_n = \mathbf{M}^n \otimes \mathbf{F}_0$，其中 $\mathbf{F}_i = (f_{i+d-1}, f_{i+d-2}, \cdots, f_i)^T$

对一个矩阵 $\mathbf{M}$，我们定义特征多项式 $f_{\mathbf{M}}(\lambda) = \det(\lambda \otimes \mathbf{I} \oplus \mathbf{M})$，则 Cayley-Hamilton 定理表明：$f_{\mathbf{M}}(\mathbf{M}) = \mathbf{0}$。

对于这个问题，$f_{\mathbf{M}}(\lambda) = \lambda^d \oplus \bigoplus_{i=1}^{d} a_i \otimes \lambda^{d-i}$，所以 $\mathbf{M}^d \oplus \bigoplus_{i=1}^{d} a_i \otimes \mathbf{M}^{d-i} = \mathbf{0}$。

当 $x^d \oplus \bigoplus_{i=1}^{d} a_i \otimes x^{d-i} = 0$ 时，我们有 $x^n = x^n \bmod \left(x^d \oplus \bigoplus_{i=1}^{d} a_i \otimes x^{d-i}\right) = \bigoplus_{i=0}^{d-1} c_i \otimes x^i$。

于是可以用快速幂计算出 $\mathbf{M}^n = \bigoplus_{i=0}^{d-1} c_i \mathbf{M}^i$，则 $\mathbf{F}_n = \mathbf{M}^n \otimes \mathbf{F}_0 = \bigoplus_{i=0}^{d-1} c_i \otimes \mathbf{M}^i \otimes \mathbf{F}_0$，从而 $f_n = \bigoplus_{i=0}^{d-1} c_i \otimes f_i$。

2.8 Nimber 多项式欧几里得算法

给定两个 Nimber 多项式，求出它们的最大公因式和 Bézout 系数[13]。

首先考虑下面这个问题：

给定两个次数不超过 $2 \cdot n$ 的多项式，求其辗转相除过程中最后两个度数不小于 n 次的多项式。

首先有如下引理：给定两个次数不超过 n 的多项式，若只考虑次数不小于 k 的部分的辗转相除，则辗转相除的过程[14]是不会受到多项式最低 $2k-n$ 位的影响的。

[13] 即利用给定的两个多项式线性组合出最大公因式

[14] 即每次的商式或者说结果的 Bézout 系数

证明：可以证明 Bézout 系数不超过 $n-k$ 次，于是最后 $2k-n$ 位不会影响到高位的值，从而不影响 Bézout 系数。

于是，先求次数不小于 $1.5n$ 的最后两个，需要递归的长度为 n。然后再求不小于 n 次的最后两个，需要递归的长度还是 n。

总的时间复杂度就是 $T(n)=T\left(\frac{n}{2}\right)+O(n\log n\log\log n)$，即 $T(n)=O\left(n\log^2 n\log\log n\right)$。

在解决了这个问题之后，每次减少一半长度，总时间复杂度为 $O\left(n\log^2 n\log\log n\right)$。

2.9 Nimber 多项式求值

给定 $F(x),x_1,x_2,\cdots,x_n$，计算 $F(x_1),F(x_2),\cdots,F(x_n)$。

注意到，$F(c)=F(x) \bmod (x\oplus c)$，于是可以考虑分治。

预处理出每个分治区间的 $x\oplus x_i$ 的乘积，每次将 F 关于两个子区间的乘积取模后分别递归处理即可。

时间复杂度为 $O\left(n\log^2 n\log\log n\right)$，使用转置原理可以得到一个常数更小的实现。

2.10 Nimber 多项式线性组合

给定 $2m$ 个多项式 $a_1(x),a_2(x),\cdots,a_m(x),b_1(x),b_2(x),\cdots,b_m(x)$，满足 $\deg b_i\le\deg a_i$。记 $n=\sum_{i=1}^{m}\deg b_i$，$A=\bigotimes_{i=1}^{m}a_i$

计算 $\bigoplus_{i=1}^{m}\frac{A}{a_i}\otimes b_i$。

实际上就是计算 $a_i(x)\oplus b_i(x)\otimes y$ 乘积中，y 一次项的系数。

在分治时保留 y^0,y^1 项系数即可。时间复杂度为 $O(n\log n\log m\log\log n)$。

2.11 Nimber 多项式插值

给定 $x_1,x_2,\cdots,x_n$ 和 $F(x_1),F(x_2),\cdots,F(x_n)$，求 $F(x)$。

考虑使用 Lagrange 插值公式：

$$F(x)=\bigoplus_{i=1}^{n}\frac{\bigotimes_{j\neq i}(x\oplus x_j)}{\bigotimes_{j\neq i}(x_i\oplus x_j)}F(x_i)$$

注意到如果计算出了所有的分母 Q_i，就是一个线性组合问题了。

设 $M(x)=\bigotimes_{i=1}^{n}(x\oplus x_i)$，则第 i 个分子为 $P_i(x)=\frac{M(x)}{x\oplus x_i}$。

注意到 $M'(x)=\bigoplus_{i=1}^{n}P_i(x)$，于是 $Q_i=M'(x_i)$，做多点求值即可。

时间复杂度为 $O\left(n\log^2 n\log\log n\right)$。

2.12 Nimber 多项式中国剩余算法

给定 m 个互素的多项式 $a_1(x), a_2(x), \cdots, a_m(x)$，和 m 的多项式 $b_1(x), b_2(x), \cdots, b_m(x)$，求多项式 $F(x)$，使得 $F \equiv b_i \pmod{a_i}$。

记 $A = \bigotimes_{i=1}^{m} a_i$，$n = \deg M$。

由中国剩余定理，$F = \bigoplus_{i=1}^{m} b_i \left[\left(\frac{A}{a_i}\right)^{-1}\right]_{a_i} \frac{A(x)}{a_i}$

于是只需要算出所有的 $\left[\left(\frac{A}{a_i}\right)^{-1}\right]_{a_i}$，问题就变为了线性组合。

首先需要计算 $\frac{A}{a_i} \bmod a_i = \frac{A \bmod a_i^2}{a_i}$，类似多项式求值那样分治多项式取模即可。

然后计算 $\left[\left(\frac{A}{a_i}\right)^{-1}\right]_{a_i}$，即模多项式的求逆，使用欧几里得算法即可。

于是总的时间复杂度为 $O\left(n \log^2 n \log\log n\right)$。

2.13 例 2. 小 Q 和小 Y 做游戏（二）

小 Q 和小 Y 是几何带师，这一次池们找到了 n 个互不相同的非负整数，用来进行接下来 n 天的研究。

在第 i 天池们准备研究 i 维的几何问题，于是池们决定先来做一个游戏。

选出所有这 n 个数的所有 $\binom{n}{i}$ 个无序 i 元组，按照从小到大的顺序排列。则一个 i 元组可以看成 i 维空间中的一个点。

首先将这些 i 元组的点都染上黑色，其他点染上白色。小 Q 先手，轮流进行如下操作：

选择一个顶点都是非负整点、棱平行于坐标轴且坐标最大的点是黑点的 i 维长方体[15]，将它的所有顶点翻转颜色。如果选不出这样的长方体则当前操作者输。

由于小 Y 非常希望胜利，所以池准备作弊。池可以在小 Q 开始操作前，选择一个前 $i-1$ 维坐标都是 1 的非负整点并翻转这个点的颜色。

于是小 Y 准备寻求你的帮助，希望你求出，对于 $i = 1, 2, \cdots, n$，分别应该染哪个点。你只需要求最后一维的坐标。

$1 \le n \le 10^5$，$0 \le a_i < 2^{32}$。

2.13.1 解法

不难发现，对于一个黑点，它的 SG 值就是每一维坐标的 Nim 积。

于是问题相当于计算 $\bigotimes_{i=1}^{n} (a_i \otimes x \oplus 1)$。

使用分治计算这些多项式的乘法即可。

时间复杂度 $O\left(n \log^2 n \log\log n\right)$。

[15]这里要求长方体的 i 维体积为正

3 Nimber 指数生成函数

3.1 Nimber 二项卷积

对 Nimber 多项式 a,b，其二项卷积定义为:

$$a \circledast b = \bigoplus_{i=0}^{n} \bigoplus_{j=0}^{m} \binom{i+j}{i} \cdot \left(a_i \otimes b_j \otimes x^{i+j}\right)$$

首先考虑 $\binom{i+j}{i}$ 模 2 的值，不难证明当且仅当 i,j 进行按位与运算为 0 时有 $\binom{i+j}{i}$ 模 2 为 1。

于是问题相当于计算 $\bigoplus_{i+j=k}[i \text{ and } j=0]\,(a_i \otimes b_j)$。

注意到，在这个条件相当于 i,j 是 k 不相交的两个子集，所以也称为子集卷积。

下面我们定义集合幂级数，并给出子集卷积的计算方法。

3.1.1 Nimber 集合幂级数

对 $[0,2^m)$ 的整数，我们可以将它理解为一个 $\{0,1,\cdots,m-1\}$ 的子集，其 2^i 项系数决定了 i 是否在集合中。

于是一个多项式可以理解为一个集合幂级数。

首先定义集合幂级数的莫比乌斯变换：

$$\hat{f_S} = \bigoplus_{T \subseteq S} f_T$$

事实上，如果将集合看做一个 m 维向量，莫比乌斯变换就是在做 m 维的前缀和。

于是容易发现其逆变换就是 m 维的差分。由于在 Nimber 域中加减法相同，并且每维的长度就是 2，所以莫比乌斯变换的逆变换就是其本身。

对于高维前缀和，可以依次对每一维做前缀和。具体来说，设当前考虑第 i 维，则从小到大枚举每个位置，如果它在第 i 维有前驱就加上前驱的值。时间复杂度为 $O\left(m2^m\right) = O\left(n \log n\right)$。

莫比乌斯变换能解决集合并卷积问题：

$$f_S = \bigoplus_{L \cup R = S} g_L \otimes h_R$$

注意到，$\hat{f_S} = \bigoplus_{(L \cup R) \subseteq S} g_L \otimes h_R = \bigoplus_{L \subseteq S \wedge R \subseteq S} g_L \otimes h_R = \left(\bigoplus_{L \subseteq S} g_L\right) \otimes \left(\bigoplus_{R \subseteq S} h_R\right) = \hat{g}_S \otimes \hat{h}_S$，所以先对 g,h 做莫比乌斯变换，对应项相乘之后再做莫比乌斯变换即可。

从另一个角度考虑，集合并卷积是高维的 max 卷积，所以用高维前缀和处理是很合理的。

但我们现在需要做的并不是并卷积，而是子集卷积：

$$f_S = \bigoplus_{L \cup R = S \wedge L \cap R = \varnothing} g_L \otimes h_R$$

一个朴素的做法是，每次枚举所有的 $L \subseteq S$，则 $R = S \setminus L$。由于每个元素只可能是不在 S 中，在 L 中或在 R 中这三种情况，所以时间复杂度为 $O(3^m) = O\left(n^{\log_2 3}\right)$。

注意到 $[L \cup R = S \wedge L \cap R = \varnothing] = [L \cup R = S \wedge |L| + |R| = |S|]$。

于是给每个位置添加上一个多项式 $p(z)$，保证 $z^{|S|}$ 项系数为 f_S 且更低项的值为 0。这个多项式称为占位多项式。

则子集卷积就可以先求莫比乌斯变换，然后对应项的占位多项式做乘法，然后再做莫比乌斯变换。时间复杂度为 $O\left(m^2 2^m\right) = O\left(n \log^2 n\right)$

3.2 Nimber 半半在线二项卷积

定义 Nimber 半半在线二项卷积为：$f = g \circledast h$。其中 g 已经给定且常数项为 0，而 h_i 需要等 f_i 计算完成后给出。

与下一个问题不同，这个问题是一个比较纯粹的集合幂级数问题。

定义 $\hat{f_{i,S}} = \bigoplus_{T \subseteq S \wedge |T| = i} f_T$。首先计算出所有的 $\hat{g}_i$。

考虑按照 $|S|$ 从 $0 \sim m$ 的顺序计算 f_S。

由于 $\hat{f_{i,S}} = \bigoplus_{j=1}^{i} \hat{g_{j,S}} \otimes \hat{h_{i-j,S}}$，计算的时间复杂度为 $O\left(m 2^m\right) = O\left(n \log n\right)$。

然后进行一次莫比乌斯变换就能得到这一轮的 f_S，并得到相应的 h_S。再做莫比乌斯变换就能得到 $\hat{h_{i,S}}$。

总的时间复杂度为 $O\left(m^2 2^m\right) = O\left(n \log^2 n\right)$。

3.3 Nimber 全半在线二项卷积

定义 Nimber 全半在线二项卷积为：

$f = g \circledast h$，其中 g 已经给定，而 h_{i+1} 需要等 $f_0, f_1, \cdots, f_i$ 都计算完成后给出。

对于单纯的集合幂级数问题，不会出现这样的情况。但由于我们做的实际上是二项卷积，如果要解微分方程就可能会出现这样的情况。

首先考虑使用分治法，设当前处理的是 $[0, 2^n)$ 这段区间。

首先递归处理 $[0, 2^{n-1})$，然后计算 $[0, 2^{n-1})$ 的 h_i 对 $[2^{n-1}, 2^n)$ 的 f_i 进行的转移。

考虑递归右半部分，注意到此时 f, h 的最高位都是 1，所以可以直接转换成都是 $[0, 2^{n-1})$ 的情况。

时间复杂度为 $O\left(n \log^3 n\right)$。

事实上对于这个问题，直接使用 $O\left(n^{\log_2 3}\right)$ 的暴力法和分治效率差不多，甚至可能更快。

3.4 Nimber 指数生成函数复合的定义

在将指数生成函数的乘法解决后，下一个问题就是如何定义复合，首先我们要考虑的就是，如何不用除法定义两个指数生成函数的复合。

考虑计算 $\frac{f^k}{k!}$。

如果 f 不是单项式，就能拆成 $g+h$，然后做二项式展开就变成了 $\sum_{i=0}^{k}\frac{g^i\cdot h^{k-i}}{i!\cdot(k-i)!}$。

对于单项式，设 $f=\frac{ax^n}{n!}$，则结果就是 $\frac{x^{kn}}{(kn)!}\cdot\frac{(kn)!}{(n!)^k\cdot k!}\cdot a^k$。

不难证明 $\frac{(kn)!}{(n!)^k\cdot k!}$ 是一个整数，所以我们可以不进行除法定义两个指数生成函数的复合了。

仿照上面可以定义 Nimber 指数生成函数的复合，并给出了一个复杂度比较高的算法，实际上重要的是如何快速计算。

事实上，由于定义基本相同，所以照搬普通的指数生成函数的运算方法即可，即在积分的时候右移一位，求导的时候左移一位，多项式乘法用二项卷积。

需要注意的是，子集卷积只是求二项卷积的一种方法，这种方法省略了二项卷积中的一些项，而这些项会在复合中重新出现。所以直接对莫比乌斯变换后的占位多项式做复合的方法是错误的。

3.5 Nimber 二项卷积逆

不妨设常数项为 1，则二项卷积逆是多项式本身。

简单计算可以发现，交叉项都会出现两遍，而平方项只有常数项，所以平方后的结果就是 1。

而唯一性十分明显。

3.6 Nimber ln 与 Nimber exp

利用 ln 和 exp 的基本公式 $g'=g\circledast f'$，由于二项卷积逆是本身所以有 $g'\circledast g=f'$。

于是给定 g 求 f 只需要一次二项卷积。

考虑给定 f 求 g。由于这是一个微分方程，所以无法使用半半在线法解决，而全半在线的效率比较低。

事实上，这里可以使用牛顿迭代法。注意到 $\ln(1\oplus x)$ 才是形式幂级数，所以方程变为了 $\ln(1\oplus f)=g$。

带入牛顿迭代公式，得 $f_{t+1}=1\oplus(1\oplus f_t)\circledast(1\oplus\ln(1\oplus f_t)\oplus g)\bmod x^{2^{t+1}}$。

时间复杂度同样和二项卷积相同。

3.7 Nimber 三角函数

通常三角函数计算方法需要对 2 做除法，无法在这里使用。

设 $g=\cos f$，则有 $g'=(e^f\oplus g)\circledast f'$。

两边乘上 e^f，得 $e^f\circledast g'=f'\oplus f'\circledast g\circledast e^f$，即 $(e^f\circledast g)'=f'$。

所以 $e^f\circledast g=f\oplus c$，令 $f=0$ 得 $c=1$。

于是 $\cos f=f\circledast e^f\oplus e^f$，$\sin f=f\circledast e^f$。

于是 $\tan f=\frac{\sin f}{\cos f}=\frac{f}{1+f}=f\circledast(1+f)=f\oplus f^2=f$。

3.8 例 3. 小 Q 和小 Y 做游戏（三）

小 Q 和小 Y 都是图论带师，他们**特别**喜欢**完全图**。

对于一张图，如果它的每个连通块都是**完全图**，那么这张图是小 Q 和小 Y 所喜欢的，否则不喜欢。

现在，小 Q 和小 Y 在一些喜欢的图上做游戏。

1. 最开始时，每张图上大小为 i 的连通块上写着一个整数 f_i。
2. 在一个人操作时，首先选择一张图，满足这张图的每个连通块上写的数都是正整数。如果没有这样的图则当前操作者失败。
3. 对于每个连通块，选择一个比原来的数更小的非负整数。
4. 对于每个连通块集合的非空子集，在图的集合中添加一张选择图的拷贝，并把这张图在子集中的连通块上数改为新选择的数。
5. 将选择的图从图的集合中删去，并由另一个人从步骤 2 开始操作。

不难证明游戏的过程是有限的。

现在，小 Q 和小 Y 在所有喜欢的顶点数为 m 的带标号图进行上面的游戏，小 Q 先手。

由于小 Y 非常希望胜利，所以祂准备作弊。由于一些原因，祂只能修改 f_m 的值。

于是小 Y 准备寻求你的帮助，希望你求出，对于 $m=1,2,\cdots,n$，分别应该把 f_m 改成多少才能使得祂必胜。

$1\le n\le 10^5$，$0\le f_i<2^{32}$。

3.8.1 解法

对于一个图 G，容易发现它的 SG 值就是每个连通块上的数的 Nim 积。

注意到问题相当于在带标号的情况下把连通图变成多个连通块的组合，于是所有图的 SG 值就是 $\exp f$。

现在问要改成什么使得后手必胜，于是相当于要 Nim 和为 0，于是就是求 $f\oplus\exp f$。

使用上面介绍的 Nimber exp 算法即可。

时间复杂度为 $O\left(n\log^2 n\right)$。

总结

本文总结了 Nimber 上的多项式问题，给出了一些问题的解决方案。但对于全半在线二项卷积和指数生成函数复合没有得到好的做法。

同时，本文的一些算法[16]的实际应用，还有待开发。因此希望本文能够起到一个抛砖引玉的作用，希望感兴趣的读者，能够进行进一步研究，扩展笔者做法，从而得到更多更有趣的做法与应用。

感谢

感谢中国计算机学会提供学习和交流的平台。
感谢国家集训队高闻远教练的指导。
感谢父母对我的培养和教育。
感谢学校的栽培，符水波老师和应平安老师的教导和同学们的帮助。
感谢虞皓翔同学，胡家齐同学，钱易同学与我交流讨论、给我启发。
感谢翁伟捷同学，潘佳奇同学，宣毅鸣同学为本文审稿。

参考文献

[1] John H. Conway. ”On numbers and games.” (2001).

[2] H.W. Lenstra. ”Nim multiplication.” Séminaire De Théorie Des Nombres 11(1978).

[3] Wikipedia, the free encyclopedia, “Sprague-Grundy theorem”,https://en.wikipedia.org/wiki/Sprague-Grundy_theorem.

[4] Wikipedia, the free encyclopedia, “Nimber”,https://en.wikipedia.org/wiki/Nimber.

[5] 彭雨翔，《多项式导论》，2016 年冬令营.

[6] 吕凯风，《集合幂级数的性质及其快速算法》，IOI2015 中国国家队候选队员论文集.

[7] 毛啸, 《再探快速傅里叶变换》, IOI2016 中国国家队候选队员论文集.

[16] 如三角函数

信息学竞赛中行列式的相关问题

绍兴市第一中学 王展鹏

摘 要

求解矩阵行列式是信息学竞赛中的一类经典问题，本文从今年的一道统一省选题出发，提出了两种思路，转化成了两种问题。对于求解任意矩阵的伴随矩阵的问题，本文详细介绍了基于分治以及基于递推的两种算法在不同模数情况下的应用。对于求解特殊代数结构下的矩阵行列式问题，本文讨论了元素是一次多项式时不同模数下的解法、元素是低次多项式时矩阵行列式在特殊模数下的解法、元素是低次多项式时矩阵行列式的解法。在最后，本文介绍了一个求解在任意交换环上的矩阵行列式的多项式算法及其优化。

1 引言

行列式是线性代数中的经典概念，有着优秀的性质。无论是在线性代数、多项式理论，还是在微积分学中，行列式作为基本的数学工具，都有着重要的应用。行列式在信息学竞赛中也时有考察。但在现有大多数问题中，题目往往仅仅计算要求对一个大素数取模的结果，而不涉及更深的知识与技巧。因此本文对求解行列式相关问题进行了探究，对于不同的代数系统，讨论了多种不同的思路与解法，希望能帮助填补这一方面的空白，同时也期待选手对这方面进行更深入的研究。

2 预备知识

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

2.1 二元运算与代数系统

定义 2.1.1 (二元运算)**.** 设 X 为一给定集合，则称映射 $f : X^2 \to X$ 为集合 X 上的一个二元运算。即对于任意有序对 $(a, b) \in X^2$，存在一个确定的 $h \in X$ 与之对应，记作

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

或

$$h = a \circ b$$

定义 2.1.2 (交换律与结合律)**.** 对于定义在集合 X 上的二元运算 $\circ$，如果对任意的 $a, b \in X$ 都有

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

则称这个二元运算是交换的。若对任意的 $a, b, c \in X$ 都有

$$(a \circ b) \circ c = a \circ (b \circ c) \tag{1}$$

则称这个二元运算是结合的。

定义 2.1.3. 设集合 $S \subset X$, $\circ$ 是 X 上的二元运算，且对任意的 $a, b \in S$ 都有

$$a \circ b \in S$$

则称集合 S 对运算 $\circ$ 封闭。

定义 2.1.4 (代数系统)**.** 在其上定义了若干二元运算 $\circ, \star, \cdots$ 的集合 X 称为一个代数系统。例如上述代数系统可记作 $(X, \circ, \star, \cdots)$。

2.2 半群与群

定义 2.2.1 (半群与交换半群)**.** 称代数系统 $(X, \circ)$ 是一个半群，当且仅当二元运算 $\circ$ 是结合的。特别地，如果半群中的二元运算 $\circ$ 是交换的，则称 $(X, \circ)$ 为交换半群。

定义 2.2.2 (半群的幺元素、幺半群)**.** 半群 $(X, \circ)$ 中的元素 e 如果满足：对任意的 $a \in X$，必有

$$a \circ e = e \circ a = a \tag{2}$$

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

定理 2.2.1. 半群中的幺元素要么不存在，要么有且仅有一个。

证明. 不妨设 $e \neq e'$ 均为半群 $(X, \circ)$ 的幺元素，则由式 (2) 可知

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

这与 $e \neq e'$ 矛盾。因此幺元素如果存在必然唯一。 □

定义 2.2.3 (可逆元素)**.** 设幺半群 $(X, \circ)$ 的幺元素为 e，对于元素 $a \in X$，若存在 $b \in X$ 使得

$$a \circ b = b \circ a = e \tag{3}$$

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

定理 2.2.2. 对于半群中的每一个可逆元素，其逆元素唯一。

证明. 不妨设 $b \neq b'$ 同时为半群 $(X,\circ)$ 中元素 a 的逆元素，则由式 (1) (2) (3) 可知

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

这与 $b \neq b'$ 矛盾。因此逆元素如果存在必然唯一。 □

定义 2.2.4 (群与交换群)**.** 设 $(X,\circ)$ 是一个幺半群。如果它的每个元素都有逆元，则称 $(X,\circ)$ 是群。特别地，如果二元运算 $\circ$ 是交换的，则称之为交换群。

2.3 环

定义 2.3.1 (环与交换环)**.** 设 X 是一个给定的集合，在其上定义了两种二元运算 $\oplus$ 和 $\odot$。代数系统 $(X,\oplus,\odot)$ 称为环，如果它满足以下条件:

(1) $(X,\oplus)$ 是一个交换群，通常称作环的加法群;

(2) $(X,\odot)$ 是一个半群，通常称作环的乘法半群;

(3) 对于任意的 $a,b,c \in X$，有

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

特别地，如果运算 $\odot$ 也是交换的，则称这个代数系统为交换环。

定义 2.3.2 (零元素)**.** 环 $(X,\oplus,\odot)$ 的加法群 $(X,\oplus)$ 含有幺元素 o，称 o 为环 X 的零元素，本文中用 0 表示。

定义 2.3.3 (幺元素)**.** 如果环 $(X,\oplus,\odot)$ 的乘法半群 $(X,\odot)$ 含有幺元素 e，则称 e 为环 X 的幺元素，本文中用 1 表示。

定义 2.3.4 (域)**.** 如果交换环 $(X,\oplus,\odot)$ 的乘法半群 $(X,\odot)$ 中所有非零元素都有逆元素，则称 A 是域。

2.4 矩阵与行列式

定义 2.4.1 (矩阵)**.** 由 $n\times m$ 个数 $a_{i,j}$ 排成的 n 行 m 列的数表称为 n 行 m 列的矩阵，简称 $n\times m$ 矩阵。这 $n\times m$ 个数称为矩阵的元素。

对于一个 n 行 m 列的矩阵 A，设它的行号集合为 $\{1,...,n\}$，列号集合为 $\{1,...,m\}$。矩阵 A 第 i 行第 j 列的元素记作 $A_{i,j}$。

定义 2.4.2 (上海森堡矩阵)**.** 对于 $n\times n$ 的矩阵 A，如果对于 $\forall i>j+1$ 都有

$$A_{i,j}=0$$

则称 A 是上海森堡矩阵。

定义 2.4.3 (行列式)**.** 对于一个 $n\times n$ 的方阵 A，将它的行列式定义为

$$\det(A)=\sum_{\sigma}(-1)^{sgn(\sigma)}\prod A_{i,\sigma_i}$$

σ 是任意一个 $1,...,n$ 的排列，$sgn(\sigma)$ 表示 σ 的逆序对数。

行列式有如下性质：

1. 在行列式中，一行（列）元素全为 0，则此行列式的值为 0；

2. 行列式中的两行（列）互换，改变行列式正负符号；

3. 在行列式中，某一行（列）有公因子 k，则可以提出 k；

4. 将一行（列）的 k 倍加进另一行（列）里，行列式的值不变。

证明直接考虑行列式的展开式即可，这里略去。

利用这些性质，可以用经典的高斯消元法在 $O(n^3)$ 级别的运算次数来计算一个定义在域上的矩阵的行列式。事实上，后三条性质对应的变换等价于左乘一个矩阵，称为初等行变换矩阵，这个变换称为初等行变换。

行列式还有另外一条经典性质：

定理 2.4.1 (行列式的乘法定理)**.** 对于任意两个 $n\times n$ 矩阵 A,B，有

$$\det(AB)=\det(A)\det(B)$$

证明. 按照高斯消元法，将矩阵 A 分解为一个上三角矩阵 A' 左乘一些初等行变换矩阵的乘积，即 $A=P_1P_2...P_kA'$，矩阵 B 分解为一个上三角矩阵 B' 右乘一些初等列变换矩阵的乘积，即 $B=B'Q_1Q_2...Q_k$。

由于两个上三角矩阵相乘，对角线上的元素也相乘，因此 $\det(A'B')=\det(A')\det(B')$，之后分类讨论证明左乘初等行变换矩阵和右乘初等列变换矩阵满足上述定理即证毕。 □

定义 2.4.4 (余子式和代数余子式)**.** 将方阵 A 去掉 $i_1,i_2,\ldots,i_k$ 行 $j_1,j_2,\ldots,j_k$ 列的矩阵的行列式称为这 k 行 k 列的余子式，记为 $A\begin{pmatrix}i_1,i_2,\ldots,i_k\\ j_1,j_2,\ldots,j_k\end{pmatrix}$，如果不特别强调，默认为去掉一行一列。将 $(-1)^{i_1+i_2+...+i_k+j_1+j_2+...+j_k}A\begin{pmatrix}i_1,i_2,\ldots,i_k\\ j_1,j_2,\ldots,j_k\end{pmatrix}$ 称为代数余子式。

定义 2.4.5 (伴随矩阵)**.** 定义 A 的伴随矩阵是一个 $n \times n$ 的矩阵，使得其第 i 行第 j 列的元素是 A 关于第 j 行第 i 列的代数余子式，记作 A^*。

定义 2.4.6 (分块矩阵与乘法)**.** 对于一个矩阵，以水平线和垂直线将其划分为若干更小的矩阵称为分块矩阵。分块矩阵中，位在同一行（列）的每一个子矩阵，都拥有相同的列数（行数）。左矩阵的列分块完全和右矩阵的行分块相同时，可以做分块乘法。规则类似于普通的矩阵乘法，相当于将每一块矩阵看作一个元素，元素乘法对应矩阵乘法。

2.5 图论

定义 2.5.1 (基尔霍夫矩阵)**.** 令 A 是图的邻接矩阵，D 是图的度数矩阵，则基尔霍夫矩阵 $L = D - A$。

定理 2.5.1 (矩阵树定理)**.** 一个图的生成树个数等于它的基尔霍夫矩阵的任意一行一列的代数余子式。

3 问题引入

3.1 题目描述

给定一个 n 个顶点 m 条边（点和边都从 1 开始编号）的无向图 G，保证图中无重边和无自环。每一条边有一个正整数边权 w_i，对于一棵 G 的生成树 T，定义 T 的价值为：T 所包含的边的边权的最大公约数乘以边权之和，即：

$$val(T) = \left(\sum_{i=1}^{n-1} w_{e_i}\right) \times \gcd(w_{e_1}, w_{e_2}, \ldots, w_{e_{n-1}})$$

其中 $e_1, e_2, \ldots, e_{n-1}$ 为 T 包含的边的编号。

询问 G 的所有生成树 T 的价值之和对 $P = 998244353$ 取模后的结果。

$2 \le n \le 30$，$1 \le m \le \frac{n(n-1)}{2}$，$1 \le w_i \le 152501$

3.2 约定

在下文中，我们认为 $O(\log P)$ 是远小于 $O(n)$ 的复杂度，并且令 $W = \max(w_i)$。

3.3 题目讨论

该题来自于 CCF 统一省选二试 A 卷第三题。

定义 S 表示所有生成树构成的集合。先做一些简单的转化：

$$\begin{aligned}&\sum_{T\in S}\left(\sum_{i=1}^{n-1}w_{e_i}\right)\times\gcd(w_{e_1},w_{e_2},\ldots,w_{e_{n-1}})\\=&\sum_{T\in S}\left(\sum_{i=1}^{n-1}w_{e_i}\right)\times\left(\sum_{d|\gcd(w_{e_1},w_{e_2},\ldots,w_{e_{n-1}})}\varphi(d)\right)\\=&\sum_{d=1}^{W}\varphi(d)\times\left(\sum_{T\in S,d|w_{e_1},\ldots,d|w_{e_{n-1}}}\sum_{i=1}^{n-1}w_{e_i}\right)\end{aligned}$$

也就是说，对于所有 d，只要计算当只考虑权值是 d 倍数的边时，所有生成树边权和的和。

自然的想法是，枚举所有边 (a,b)，计算它的出现次数，显然等于总生成树个数减去去掉这条边以后的个数。生成树计数可以利用矩阵树定理转化为求行列式，时间复杂度 $O(mn^3)$，难以接受。

注意到减去一条边会导致基尔霍夫矩阵 $O(1)$ 个元素值变化。如下图所示矩阵所示。

$$\begin{bmatrix}\ddots&\vdots&\cdots&\vdots&\iddots\\\cdots&A_{u,u}\to A_{u,u}-1&\cdots&A_{u,v}\to A_{u,v}+1&\cdots\\\cdots&\vdots&\ddots&\vdots&\cdots\\\cdots&A_{v,u}\to A_{v,u}+1&\cdots&A_{v,v}\to A_{v,v}-1&\cdots\\\iddots&\cdots&\cdots&\vdots&\ddots\end{bmatrix}$$

因此可以简单讨论一下这四个位置的权值修改对行列式产生的影响。

考虑依次修改 $A_{u,u},A_{u,v},A_{v,u},A_{v,v}$ 位置的值，实时计算新矩阵的行列式，容易发现行列式变化值依次是

$$-(-1)^{u+u}A\begin{pmatrix}u\\u\end{pmatrix},\ (-1)^{u+v}A\begin{pmatrix}u\\v\end{pmatrix},\ (-1)^{v+u}A\begin{pmatrix}v\\u\end{pmatrix}-A\begin{pmatrix}u,v\\u,v\end{pmatrix},\ -\left((-1)^{v+v}A\begin{pmatrix}v\\v\end{pmatrix}-A\begin{pmatrix}u,v\\u,v\end{pmatrix}\right)$$

注意到 $A\begin{pmatrix}u,v\\u,v\end{pmatrix}$ 项正负抵消，因此总的变化量恰好就是四个位置的代数余子式乘上对应的变化量的和。至此，问题转化为计算出所有位置的代数余子式。

另一个想法考虑问题的组合意义，最终的生成树恰好由一条固定计算权值的边和其余 $n-2$ 条边构成，那么可以将一条边的权值看作 w_ix+1，计算基尔霍夫矩阵任意代数余子式的一次项系数就是问题的答案。也就是说，问题变为求元素是一次多项式的矩阵的行列式的一次项系数。

因此，我们将原问题转化成了上述两个问题，接下来，本文将详细讨论这两个问题并加以推广。

4 取模意义下求解伴随矩阵

4.1 拆分问题

考虑如何计算删除一行一列的余子式，我们将算法分成两部分：

1. 枚举删除哪一行，计算剩余矩阵消元后的形式，类似下图。

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-2} & a_{1,n-1} & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & \cdots & a_{2,n-2} & a_{2,n-1} & a_{2,n} \\ 0 & 0 & a_{3,3} & \cdots & a_{3,n-2} & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-2,n-2} & a_{n-2,n-1} & a_{n-2,n} \\ 0 & 0 & 0 & \cdots & 0 & a_{n-1,n-1} & a_{n-1,n} \end{bmatrix}$$

2. 在枚举行的基础上，枚举删除哪一列，可以发现剩余矩阵的行列式等于一个上三角矩阵的行列式乘上一个上海森堡矩阵的行列式，也就是说，需要计算对于任意右下角 $i \times i$ 子矩阵的行列式。

由于本算法扩展性比较强，因此下文认为模数可以是任意数。

4.2 第一部分

考虑使用分治做法来解决这个问题。使用 $solve(l,r)$ 表示计算当删除的行处于区间 $[l,r]$ 内时，每行消元后的矩阵。

考虑每次递归到 $solve(l,mid)$ 和 $solve(mid+1,r)$，如果递归到 $solve(mid+1,r)$，那么可以将 $l \sim mid$ 这些行消成类似于上三角矩阵的形式，递归另一部分；如果递归到 $solve(l,mid)$，可以将 $l \sim mid$ 和 $mid+1 \sim r$ 整体交换后采用类似于上面的做法解决。

实现的时候，可以将下面的行换到上面，乘上对应的系数就可以了。

同时，由于模数可能是合数，因此消元过程需要使用辗转相除法，即将消元的那一列辗转相减。

4.3 第二部分

下图就是一个上海森堡矩阵：

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n-1} & a_{2,n} \\ 0 & a_{3,2} & a_{3,3} & \cdots & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

对于第二部分，相当于要求一个上海森堡矩阵任意右下角 $i\times i$ 子矩阵的行列式，这是一个经典问题。

令 f_i 表示右下角 $i\times i$ 子矩阵的行列式，考虑将行列式展开，可以发现一旦选择了第一行，就会变成一个子问题，因此

$$f_i = a_{i,i}f_{i+1} - a_{i+1,i}a_{i,i+1}f_{i+2} + a_{i+1,i}a_{i+2,i+1}a_{i,i+2}f_{i+3} + \dots$$

4.4 复杂度分析

对于第一部分，要消元的次数是递归到的区间长度和加上辗转相除时交换行的次数，容易发现区间长度和是 $O(n\log n)$。辗转相除部分每当一行确定不删以后（即当 $solve(l,r)$ 时，不在这个区间内的行都确定不删），每次用辗转相除法消元时，类似于求最大公约数，除了第一次交换以外，每被交换一次，第一个非零位置至少除以二，因此额外的消元次数 $O(n\log n\log P)$。

每次消元复杂度是 $O(n)$，因此第一部分复杂度是 $O(n^3\log n + n^2\log n\log P)$。

对于第二部分，需要执行 n 次，每次是一个 $O(n^2)$ 的 DP，因此复杂度为 $O(n^3)$ 。

忽略不是瓶颈的部分，总的复杂度 $O(n^3\log n)$。

4.5 从整体来看

注意到，我们是要计算 $n\times n$ 个答案，能不能用一些位置的答案来推出另一些位置呢？有定理：

定理 4.5.1. 令 I 表示单位矩阵，那么对于任意方阵 A，满足：$AA^* = \det(A)I$。

证明. 将上式展开，得到

$$\forall 1\le i,j\le n, \sum_{k=1}^{n} A_{i,k}\times a_{j,k}(-1)^{j+k} = [i=j]\times\det(A)$$

考虑将行列式按第 j 行展开的形式：$\sum_{k=1}^{n} A_{j,k}\times a_{j,k}(-1)^{j+k}$。

比较两个式子，左式结果实际相当于用第 i 行覆盖第 j 行后的行列式，因此显然等于右式。 □

4.6 回到质数

不妨重新思考模数是质数的情况，分类讨论矩阵的秩（令 n 为矩阵阶数）：

$rank(A) = n$：矩阵有逆，由 $AA^* = \det(A)I$ 可以得到 $A^* = \det(A)A^{-1}$，直接使用高斯消元算法计算逆矩阵和行列式即可；

$rank(A) \le n-2$：容易发现，无论如何去掉一行一列，剩余矩阵的秩都不会变大，因此任意位置的余子式一定是 0；

$rank(A) = n-1$：仍然考虑 $AA^* = \det(A)I$ 这一等式，因此可以对 A 进行初等行变换以实现高斯消元。我们知道，对矩阵进行初等行变换等价于左乘初等行变换矩阵。也就是说，左乘一个行变换矩阵 C，使得最后等式形式变为：$A'A^* = \det(A)C$，其中 A' 是消元后的矩阵。

具体实现时，假设处理到第 k 行第 k 列时，如果 $\forall i \ge k, A_{i,k} = 0$，那么最终必然 $A_{k,k} = 0$。由等式可以得出对于 $\forall i < k, j$，满足

$$A^*_{i,j} = \frac{0 - \sum_{i<l\le n} A'_{i,l}A^*_{l,j}}{A'_{i,i}}$$

由于秩是 $n-1$，因此之后还没消过的大小为 $(n-k+1)\times(n-k)$ 矩阵上必然列满秩，容易发现 $\forall i > k, j$，满足 $A^*_{i,j} = 0$。所以最后只需暴力计算第 k 行的余子式，就可以直接递推出其余所有行了。

朴素地计算一行余子式是 $O(n^4)$ 的，令人难以接受。不过容易发现，由于是同一行的余子式，可以先删掉这一行，之后利用之前的做法，先消元以后删掉任意一列就又满足上海森堡矩阵的性质了，复杂度仍然是 $O(n^3)$。

但事实上可以用 $A^*A = \det(A)I$ 类似地通过一列来推每一列，由于本题矩阵的对称性，并不需要再进行一次高斯消元，因为一行推其他行和一列推其他列的系数显然是相同的。因此，只需要暴力计算一个位置的余子式就可以了，时间复杂度同样是 $O(n^3)$。

4.7 递推方法

可以发现，不管模数是素数还是合数，我们的目标都是得到 $A'A^* = \det(A)C$，其中 A' 是上三角矩阵，对于 $\forall i, j$，可以得到

$$\sum_{k\ge i} A'_{i,k}A^*_{k,j} = \det(A)C_{i,j}$$

容易发现，能不能推出 $A^*_{i,j}$，取决于 $A'_{i,i}$ 是否存在逆元。如果存在，就可以通过已经算过的位置来递推此位置，同样地，如果用递推列的方法得到 $A'_{j,j}$ 存在逆元，也可以推出此位置。因此，只要算出所有 i, j 满足 $A'_{i,i}, A'_{j,j}$ 都没有逆元的位置的余子式，就可以推出所有的余子式了。用 s 表示这样的列个数，自然，我们希望这样的 s 不太大。

4.8 最终算法

考虑能不能将之前的算法扩展到合数的情况。

由于模合数意义下的运算不是整环，因此不能用传统的方法讨论矩阵的秩。

但我们仍然可以用高斯消元来处理 $AA^* = \det(A)I$，用辗转相除法来消元，如果用辗转相除法求得的 gcd 与模数不互质，因为没有逆元，我们就不能用之前的递推方法来推这一行的伴随矩阵了。为了使得结果是上三角矩阵，可以将没有逆元的列换到最后，同时这也保证了那些没有逆元的列都在最后。为了方便，之后就假设 A' 满足这样的性质。

定理 4.8.1. *对 P 质因数分解后，对于它的因子 p^k，如果按上述求得的 $\gcd \bmod p = 0$ 列的数量大于 k，那么所有要求的余子式模 $p^k = 0$。*

证明. 考虑把模数当成 p，如果满足上述条件，求得矩阵的秩显然 $< n-k$，同样地，去掉一行一列不会使矩阵的秩变大，因此最后消元以后的主对角线上至少有 k 个 p 的倍数，即这个位置的余子式 $\bmod p^k = 0$。 □

由上述定理，可以发现如果所有余子式模 $p^k = 0$，那么我们就可以去掉这个因子。设简化过后的模数 $P = \prod p_i^{k_i}$，那么 $s \le \sum k_i \le \log_2 P$。

朴素地求 s^2 个位置的余子式，时间复杂度 $O(n^3 s^2)$。还是不太能令人满意。为了方便讨论与实现，可以将这些位置交换到矩阵的右下角，只要求出这些位置的余子式就可以了。我们可以先将左上角消成上三角形式，如下图（为了方便，令 $t = n - s$）：

$$\left[\begin{array}{cccc|ccc} a_{1,1} & a_{1,2} & \cdots & a_{1,t} & a_{1,t+1} & \cdots & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,t} & a_{2,t+1} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{t,t} & a_{t,t+1} & \cdots & a_{t,n} \\ \hline a_{t+1,1} & a_{t+1,2} & \cdots & a_{t+1,t} & a_{t+1,t+1} & \cdots & a_{t+1,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,t} & a_{n,t+1} & \cdots & a_{n,n} \end{array}\right]$$

之后枚举删除哪一行，可以将其余行消掉

$$\left[\begin{array}{cccc|ccc} a_{1,1} & a_{1,2} & \cdots & a_{1,t} & a_{1,t+1} & \cdots & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,t} & a_{2,t+1} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{t,t} & a_{t,t+1} & \cdots & a_{t,n} \\ \hline 0 & 0 & \cdots & 0 & a_{t+1,t+1} & \cdots & a_{t+1,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & a_{i-1,t+1} & \cdots & a_{i-1,n} \\ 0 & 0 & \cdots & 0 & a_{i+1,t+1} & \cdots & a_{i+1,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & a_{n,t+1} & \cdots & a_{n,n} \end{array}\right]$$

最后再枚举删除哪一列，朴素使用 s^3 的高斯消元计算右下角的行列式再乘上 $\prod_{1\le i\le t} a_{i,i}$ 就行了。总的复杂度 $O(n^3+n^2\log^2 P+\log^5 P)$。

当然，我们还能继续优化。仍然采用之前 $O(n^3\log n)$ 算法的思想，把左上角消完以后，用分治算法消最后的 s 行，最后会剩下 $s\times s$ 的类似上海森堡矩阵的矩阵，依然可以使用消上海森堡矩阵的方法求出每个位置的余子式，总的复杂度是 $O(n^3+n^2\log P\log\log P)$。

这样的复杂度已经几乎不比朴素求行列式的 $O(n^3)$ 复杂度劣了。至此，在任意模数下，对于求解任意矩阵的伴随矩阵的问题，我们得到了一个复杂度令人满意的做法。

4.9 其他思路

可以发现，到最后一步以后，问题转化为求右下角一个方阵所有位置的代数余子式。考虑给矩阵加一行，加一列，那么对于 (x,y) 位置的代数余子式，等于形如下面矩阵的行列式的相反数。

$$\begin{bmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,y} & \cdots & a_{1,n-1} & a_{1,n} & 0\\
a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,y} & \cdots & a_{2,n-1} & a_{2,n} & 0\\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots\\
a_{x,1} & a_{x,2} & a_{x,3} & \cdots & a_{x,y} & \cdots & a_{x,n-1} & a_{x,n} & 1\\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots\\
a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,y} & \cdots & a_{n,n-1} & a_{n,n} & 0\\
0 & 0 & 0 & \cdots & 1 & \cdots & 0 & 0 & 0
\end{bmatrix}$$

这是因为根据定义计算行列式时，每行每列需要恰好选择一个位置，那么为了最后乘积非零，显然必须得选这两个 1，剩余自然就是 (x,y) 的余子式了。

由于左上角 $n\times n$ 的矩阵不随着 x,y 的变化而变化，因此可以对左上角进行高斯消元，并记录消元的结果，当固定 (x,y) 以后矩阵除了 $a_{n+1,v}=1$ 以外，一定是一个上三角矩阵。因此只需利用高斯消元使得最后一行也满足上三角矩阵性质就可以计算行列式了。由于 $v>n-s$，考虑到极端情况辗转相除每次可能会进行 $\log P$ 次，时间复杂度 $O(n^3+s^4\log P)=O(n^3+log^5P)$，代码实现比起之前的算法更简洁。如果不结合之前的递推算法，直接计算每个位置的余子式，通过一些技巧也容易做到 $O(n^3+n^2\log^3 P)$。

5 取模意义下计算元素是多项式的矩阵行列式的一次项

5.1 朴素实现

当模数是质数时，可以使用经典的高斯消元算法计算行列式。

在高斯消元过程中，可以用最低非零次项最小的多项式所在的行来消其余行，将矩阵消成上三角形式就可以直接计算行列式了。

由于最后只需求一次项，全过程可以在 mod x^2 过程下进行，总的复杂度 $O(n^3)$。

5.2 扩展到合数

如果模数是合数，可以发现很难将之前的算法扩展。

例如在 mod 8 意义下，如下图所示矩阵。

$$\begin{bmatrix} 4 & a \\ x+2 & b \end{bmatrix}$$

可以注意到，我们很难使用 4 消掉 $x+2$。

不过根据上一个问题 $O(n^3 \log n)$ 复杂度算法给我们的启示：如果开始将矩阵消成较简的形式，最后的求行列式部分可能就会简单得多。

可以发现，在模合数意义下，虽然不能使用高斯消元将以一次函数为元素的矩阵消成上三角，但可以仅仅将常数项消成上三角。

如下图

$$\begin{bmatrix} a_{1,1}x+b_{1,1} & a_{1,2}x+b_{1,2} & a_{1,3}x+b_{1,3} & \cdots & a_{1,n}x+b_{1,n} \\ a_{2,1}x & a_{2,2}x+b_{2,2} & a_{2,3}x+b_{2,3} & \cdots & a_{2,n}x+b_{2,n} \\ a_{3,1}x & a_{3,2}x & a_{3,3}x+b_{3,3} & \cdots & a_{3,n}x+b_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1}x & a_{n,2}x & a_{n,3}x & \cdots & a_{n,n}x+b_{n,n} \end{bmatrix}$$

仍然可以枚举选择哪个 x，贡献就是该位置的代数余子式。容易注意到，对于主对角线上的位置，余子式就是主对角线上其余常数项的乘积，如果是没有常数项的位置，那么余子式就是两条对角线常数项和一个只考虑常数项的上海森堡矩阵行列式的乘积，对于右上角的位置，余子式则显然是 0。

对于一个上海森堡矩阵，对于任意 s,t，需要求左上角为 (s,s) 右下角为 (t,t) 的子矩阵行列式。那么可以枚举任意左上角 $t \times t$ 的子矩阵，这显然仍然是一个上海森堡矩阵，然后计算所有右下角 $(t-s+1)\times(t-s+1)$ 子矩阵行列式，这与要求的所有行列式一一对应。综上，我们仍然在 $O(n^3)$ 的复杂度内解决了这个问题。

6 取模意义下计算元素是多项式的矩阵的行列式

6.1 问题介绍

回顾之前做法，可以发现：我们虽然得到了较为优秀的复杂度，但仍然使用了很取巧的思路：枚举选择的一次项位置。实际问题中，更常见的是需要求出整个行列式，而上述算法就难以再次扩展。考虑尝试其它的思路。

为了方便，本节认为矩阵元素的多项式次数是 $O(1)$ 的。事实上，如果设多项式最高次数为 d，复杂度往往只要再乘一个 d 就好了。

6.2 基于拉格朗日插值的做法

考虑最终行列式的形式，显然是一个次数 $O(n)$ 的多项式。因此考虑使用经典的拉格朗日插值算法还原出这个行列式。

将模数质因数分解，设 $P=\prod p_i^{k_i}$，考虑分别计算出对 $p_i^{k_i}$ 取模后的结果后再用中国剩余定理合并。

考虑拉格朗日插值的形式：

$$f(x)=\sum y_i\prod_{i\neq j}\frac{(x-x_i)}{x_j-x_i}$$

容易注意到，当 $\forall j<k, x_j\not\equiv x_k(\mathrm{mod}\, p_i)$ 时，由于除法有逆元，因此只要点值比次数大，就能插出多项式。

沿用之前的证明思路，假设模数是素数 p_i，如果存在 $x_j\equiv x_k(\bmod p_i)$，这说明在模 p_i 意义下，有一个点值是没有意义的。也就是说，想要解出系数对 p_i 取模的结果，也需要这么多点值，这也就说明了这是充分条件。

令答案多项式次数为 $k-1$，显然当 $k\leq\min(p_i)$ 时，只需选取 $0,...,k-1$ 这些点值暴力计算行列式就可以算出答案；但当 $k>\min(p_i)$ 时，即使选取了所有 $0,...,P-1$ 的点值也不足以解出行列式。

要解决这个问题，必须要扩大模数，在 $P=2^k$ 的最坏情况下，模数的位数会达到 $O(n)$ 级别，需要维护高精度数，总的复杂度高达 $O(n^4\times f(n))$，$f(n)$ 表示 n 位高精度数运算的复杂度。

综上，拉格朗日插值在模数最小质因子较大时比较优秀，而在其余情况下不那么令人满意。

6.3 基于中国剩余定理的做法

上一节中，在模数比较小的情况下，由于没有逆元除法很难进行。

那么不妨考虑一个更暴力的想法，去掉取模，直接计算答案。

由于现在没有取模，我们可以方便地进行插值，因此只需要计算元素是整数的矩阵行列式就可以了。

既然模数是素数的情况能够方便的计算，要将不取模的情况转化为素数，让人联想到中国剩余定理。

令最终行列式位数级别为 $g(n)$，根据定义，容易发现 $g(n)\leq n\log n$，实际上这个上界并不紧，更紧的上界可以参见参考文献 [2]，由于篇幅有限，这里略去。

我们需要用 $O(g(n))$ 个大质数来还原答案，每次复杂度 $O(n^3)$，再结合插值，总的复杂度 $O(n^4 g(n))$。

7 计算定义在任意交换环上的矩阵的行列式

7.1 算法思路

总结以上做法，发现上述算法都要用到高精度数，在信息学竞赛中难以在考场上实现。并且，两个算法都需要用到多项式运算的性质，在最后，我们更想要一个能在任意交换环上运行的算法。

定义新的矩阵 $B(z) = I + z(A - I)$，容易注意到 $B(0) = I, B(1) = A$。

也就是说，如果计算出 $B(z)$ 的行列式，直接计算系数和就是答案了。根据定义，矩阵 $B(z)$ 的形式如下图。

$$\begin{bmatrix} (a_{1,1}-1)z+1 & a_{1,2}z & a_{1,3}z & \cdots & a_{1,n}z \\ a_{2,1}z & (a_{2,2}-1)z+1 & a_{2,3}z & \cdots & a_{2,n}z \\ a_{3,1}z & a_{3,2}z & (a_{3,3}-1)z+1 & \cdots & a_{3,n}z \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1}z & a_{n,2}z & a_{n,3}z & \cdots & (a_{n,n}-1)z+1 \end{bmatrix}$$

考虑直接实现高斯消元法，由于矩阵只有主对角线上元素有常数项 1，在算法执行过程中，我们可以保证矩阵一直满足这个性质。因此这些元素总是有逆元，且求逆只需用到幺元、乘法和加法，符合交换环的性质，困扰我们的问题就被解决了。

观察答案形式容易注意到最终次数不超过 n，因此全过程可以对 z^{n+1} 取模，时间复杂度瓶颈在于 $O(n^3)$ 次两个交换环上的多项式卷积，朴素实现需要 $O(n^5)$ 次交换环上的运算。

事实上，交换环卷积算法可以在 $O(n \log n \log\log n)$ 内完成，具体算法可以见参考文献 [1]，由于该算法跟本文关系不大，这里略去。

综上，我们得到了一个 $O(n^4 \log n \log\log n)$ 的优秀算法。

7.2 矩阵乘法

众所周知，矩阵乘法可以做到 $O(n^\omega)$ 的复杂度，其中 $\omega \approx 2.373$，这个算法可以见参考文献 [4]，由于其过于复杂且和本文关系不大，这里不作展开。在实际应用中，往往使用相对好写且常数优秀的 $O(n^{2.807})$ 算法作为代替，下文对此做简单介绍。

为了方便，将矩阵扩充到 $2^k \times 2^k$ 大小，多的元素用零填充。设求 $C = A \times B$。

将每个矩阵分成规模为 $2^{k-1} \times 2^{k-1}$ 的四块：

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, \quad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, \quad C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

我们有：

$$C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$$
$$C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$$
$$C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$$
$$C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$$

引入新矩阵：

$$M_1 := (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$$
$$M_2 := (A_{2,1} + A_{2,2})B_{1,1}$$
$$M_3 := A_{1,1}(B_{1,2} - B_{2,2})$$
$$M_4 := A_{2,2}(B_{2,1} - B_{1,1})$$
$$M_5 := (A_{1,1} + A_{1,2})B_{2,2}$$
$$M_6 := (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$$
$$M_7 := (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$$

可得：

$$C_{1,1} = M_1 + M_4 - M_5 + M_7$$
$$C_{1,2} = M_3 + M_5$$
$$C_{2,1} = M_2 + M_4$$
$$C_{2,2} = M_1 - M_2 + M_3 + M_6$$

根据主定理，总共需要 $T(n) = 7T(n/2) + O(n^2) \approx O(n^{2.807})$ 元素运算。

7.3 矩阵求逆

类似地，将矩阵扩充到 $2^k \times 2^k$ 大小，多出来的右下角部分可以用单位矩阵填充，显然新矩阵逆的左上角部分就是原矩阵的逆。

考虑仍然将矩阵分块，即将矩阵表示为 $M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$ 的形式，定义其转换矩阵：

$$L = \begin{bmatrix} I_p & 0 \\ -D^{-1}C & D^{-1} \end{bmatrix}$$

其中 I_p 表示一个 $p \times p$ 的单位矩阵。矩阵 M 右乘转换矩阵 L，具体的形式是：

$$M \cdot L = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} I_p & 0 \\ -D^{-1}C & D^{-1} \end{bmatrix} = \begin{bmatrix} A - BD^{-1}C & BD^{-1} \\ 0 & I_q \end{bmatrix}$$

事实上，左上角的 $A-BD^{-1}C$ 称为原矩阵的舒尔补。因此，矩阵 M 的逆，如果存在的话，可以用 D^{-1} 以及其舒尔补（如果存在的话）来表示：

$$\begin{aligned}\begin{bmatrix}A & B\\ C & D\end{bmatrix}^{-1} &= \begin{bmatrix}I & 0\\ -D^{-1}C & D^{-1}\end{bmatrix}\begin{bmatrix}(A-BD^{-1}C)^{-1} & 0\\ 0 & I\end{bmatrix}\begin{bmatrix}I & -BD^{-1}\\ 0 & I\end{bmatrix}\\ &= \begin{bmatrix}A^{-1}+A^{-1}B(D-CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D-CA^{-1}B)^{-1}\\ -(D-CA^{-1}B)^{-1}CA^{-1} & (D-CA^{-1}B)^{-1}\end{bmatrix}\end{aligned}$$

由本算法的性质，容易注意到 D 和 $D-CA^{-1}B$ 永远可逆，这是因为无论何时，矩阵都只有在主对角线上有常数项 1。也就是说，我们将问题转化为了规模更小的矩阵求逆和矩阵乘法。一个 n 阶矩阵的逆，可以转化为两个 $n/2$ 阶矩阵求逆和多次矩阵乘法和加法，总计需要 $T(n)=2T(n/2)+O(n^{\omega})=O(n^{\omega})$ 元素运算。

7.4 矩阵的行列式

同样地，将矩阵扩充到 $2^k\times 2^k$ 大小，多出来的右下角部分可以用单位矩阵填充，显然行列式不变。对于矩阵 $\begin{bmatrix}A & B\\ C & D\end{bmatrix}$，我们还有：

$$\begin{bmatrix}A & B\\ C & D\end{bmatrix}\begin{bmatrix}I & -A^{-1}B\\ 0 & I\end{bmatrix}=\begin{bmatrix}A & 0\\ C & D-C\times A^{-1}B\end{bmatrix}$$

所以

$$\begin{aligned}\det\left(\begin{bmatrix}A & B\\ C & D\end{bmatrix}\right) &= \det\left(\begin{bmatrix}A & 0\\ C & D-C\times A^{-1}B\end{bmatrix}\right)\\ &= \det(A)\det(D-C\times A^{-1}B)\end{aligned}$$

同样由于本算法的性质，A 和 $D-CA^{-1}B$ 都一定可逆，我们将行列式转化为了规模更小的行列式和求逆以及乘法问题，总计需要 $T(n)=2T(n/2)+O(n^{\omega})=O(n^{\omega})$ 次元素运算。

7.5 优化

综上，矩阵的行列式可以用 $O(n^{\omega})$ 次元素的运算解决，再结合矩阵元素乘法的 $O(n\log n\log\log n)$ 算法，总的复杂度 $O(n^{\omega+1}\log n\log\log n)$。

7.6 上述算法的扩展

事实上，对于任意定义在实数域上的矩阵都可以用类似的方法求逆，我们有

$$A^{-1}=A^{T}(AA^{T})^{-1}$$

当 A 可逆时，可以证明 AA^T 由于有着优秀的性质，可以套用之前的算法。

对于行列式，如果用相同的方法，可以求出原行列式的平方，而开方会导致有两个答案，这使得问题变得十分复杂，有兴趣的读者可以自行阅读参考文献 [6]，本文不做展开。

不过即使如此，仍然可以用简单的随机化方法使得矩阵 A 可逆，如随机一个矩阵 F 对原矩阵做矩阵乘法来代替初等列变换，使得 $A' = A + BF, C' = C + DF$，可以取得不错的效果。

8 展望

事实上，本文提到的很多算法都可以进一步优化，例如使用大步小步递推矩阵数列、HALF-GCD 等方法。但由于本人数学水平实在有限，因此没有做更深的探讨。总而言之，希望本文可以起到抛砖引玉的作用，也希望有兴趣的读者，能够进一步研究。

9 致谢

感谢中国计算机学会提供学习和交流的平台。
感谢绍兴一中的陈合力老师和董烨华老师的关心和指导。
感谢国家集训队教练高闻远的指导和帮助。
感谢李佳衡、孟煜皓同学为本文审稿。
感谢其他对我有过帮助和启发的老师和同学。
感谢父母对我的关心、支持与无微不至的照顾。

10 参考文献

[1] David G. Cantor and Erich Kaltofen, ON FAST MULTIPLICATION OF POLYNOMIALS OVER ARBITRARY ALGEBRAS. Acta Informatica vol. 28, nr. 7, pp. 693–701 (1991).

[2] Erich Kaltofen and Gilles Villard. ON THE COMPLEXITY OF COMPUTING DETERMINANTS, comput. complex. 13 (2004), 91 –130.

[3] Gilles Villard. Exact computation of the determinant and of the inverse of a matrix.Workshop on Complexity —FoCM Minneapolis, Aug. 12, 2002.

[4] Williams, Virginia Vassilevska (2012-05-19). Multiplying matrices faster than coppersmith-winograd. Proceedings of the 44th symposium on Theory of Computing - STOC '12. ACM. pp. 887–898.

[5] Gilles Villard. Kaltofen’s division-free determinant algorithm differentiated for matrix adjoint computation. Journal of Symbolic Computation, Elsevier, 2011, 46 (7), pp.773-790.

[6] James R. Bunch and John E. Hopcroft. Triangular Factorization and Inversion by Fast Matrix Multiplication. mathematics of computation, volume 28, number 125, January, 1974.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. Introduction to Algorithms, 3rd ed. MIT Press, Cambridge, MA, 2009, §28.2.

[8] VOLKER STRASSEN. Gaussian Elimination is not Optimal. Numer. Math. t3, 354–356 (t969).

[9] Gilles Villard. Computation of the Inverse and Determinant of a Matrix. Algorithms Seminar 2001−2002, F. Chyzak (ed.), INRIA, (2003), pp. 29−32.

[10] Wikipedia, Schur complement.

计算球面复合区域面积的高效算法

杭州学军中学 周任飞

摘 要

在球面上，对球冠形区域反复进行交、并、差等布尔运算，可以表示出许多复杂的区域，本文中称为“复合区域”。计算球面复合区域的面积是一个重要的问题，在现实中有广泛的应用。本文提出了以布尔表达式形式给出的复合区域面积问题的定义，并提出了一个优秀的算法，以 $O(nm\log m)$ 的最坏情况运行时间复杂度求出球面复合区域面积的精确值，其中 n 是球冠形区域的数量，m 是布尔表达式的长度。

1 引言

给定球面上的 n 个球冠形区域 $D_1, D_2, \ldots, D_n$，以及一个以它们为参数的、长度为 m 的布尔表达式，以此定义出一个复合区域 D^*。其中，这些球冠两两不同；布尔表达式显式给出，每个区域在表达式中可以出现多次。如何计算这个复合区域 D^* 的面积，就是本文主要讨论的问题 —— 球面复合区域面积问题。

这类问题在现实中有广泛的应用，例如计算球形工艺品的原材料用量，计算特定地理区域的面积，计算通讯基站的覆盖范围，计算“北斗”卫星定位系统在地球表面上的有效服务范围，等等。这些应用在计算机辅助设计与地理信息系统中尤为重要。

这类问题在平面上的特例，在算法竞赛中出现已久。早在 NOI 2004，就出现了《降雨量》一题，要求计算平面上若干个平行四边形的并的面积。后来几年里，NOI 中又出现过若干次平面上的“面积并”问题，例如比较著名的 NOI 2005《月下柠檬树》[2]。至于球面复合区域面积问题，则在 ICPC 赛事中出现过一次 [6]。

此前，解决这类问题并没有通用的方法。有一种传统的数值积分方法，能计算出复合区域面积的一个近似值。此外，还有一种扫描线方法，通过切割复合区域，在 $O(n^2 m\log m)$ 的最坏时间复杂度内计算出平面复合区域面积的精确值。本文提出了一种优秀的算法，在 $O(nm\log m)$ 的最坏时间复杂度内计算出平面或球面的复合区域面积的精确值，具有效率高、精度高、数值稳定性好、拓展性强等优点。

本文由六个章节组成。第二章从一个较简单的问题 —— 平面面积并问题 —— 引入，并提出了格林公式方法来得到一个运行时间上界为 $O(n^2\log n)$ 的精确算法。第三章将格林公式方法应用到球面上，以相同的运行时间上界解决球面面积并问题。第四章利用布尔运算

符和布尔表达式，提出球面复合区域面积问题的精确定义，并介绍解决该问题的优秀算法。第五章讨论了这个算法的一些应用场景，并展示了如何对算法的各个阶段作出适当调整，以适应具体问题。第六章作简要的总结。

2 平面面积并问题

问题 1（平面面积并问题） 给定平面上的 n 个圆形区域 $D_1, \ldots, D_n$，定义

$$D^* = D_1 \cup D_2 \cup \ldots \cup D_n,$$

其中每个圆形区域 D_i 指的是一个圆的内部。计算 D^* 的面积。

本章提出一个基于格林公式的算法，解决上述问题。

格林公式 设 $\Omega \subset \mathbb{R}^2$ 是由有限条分段光滑的曲线围成的闭区域。如果函数 $P(x,y)$ 和 $Q(x,y)$ 在 Ω 上连续，并有连续的偏导数 $\partial Q/\partial x$ 和 $\partial P/\partial y$，那么有

$$\oint_{\partial\Omega} P\mathrm{d}x + Q\mathrm{d}y = \iint_{\Omega} \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) \mathrm{d}x\,\mathrm{d}y,$$

其中 $\partial\Omega$ 是区域 Ω 的边界。它的定向是这样被确定的：一个人沿着 $\partial\Omega$ 的正方向行进时，区域 Ω 总在这个人的左边。

该定理的证明可以在数学分析教材中找到，这里不再叙述其证明，而是叙述如何利用该定理计算 D^* 的面积。

记 S_{D^*} 表示 D^* 的面积，则

$$S_{D^*} = \iint_{D^*} \mathrm{d}x\,\mathrm{d}y.$$

如果令

$$\begin{cases} \Omega = D^*, \\ P(x,y) = -y, \\ Q(x,y) = 0, \end{cases}$$

则 Ω, P, Q 满足格林公式的应用条件。代入格林公式，得到

$$\begin{aligned} \oint_{\partial D^*} (-y)\,\mathrm{d}x &= \iint_{D^*} \mathrm{d}x\,\mathrm{d}y \\ &= S_{D^*}, \end{aligned}$$

从而将计算面积转化为计算边界 ∂D^* 的曲线积分。

图 1 是问题 1 的一个示例，其中 D^* 被涂成淡紫色，圆用虚线标出，∂D^* 用实线标出。如图所示，∂D^* 是若干段圆弧组成的，设这些圆弧分别为 $A_1, \ldots, A_m$，则根据积分可加性，有

$$\oint_{\partial D^*} (-y)\,\mathrm{d}x = \sum_{i=1}^{m} \int_{A_i} (-y)\,\mathrm{d}x,$$

其中每个 A_i 的方向都是逆时针。我们利用上式的右侧来计算上式的左侧。于是，问题 1 被转化为两个子问题：

问题 1.1 求出 $A_1, A_2, \ldots, A_m$。

问题 1.2 计算 $\int_{A_i} (-y)\,\mathrm{d}x$。

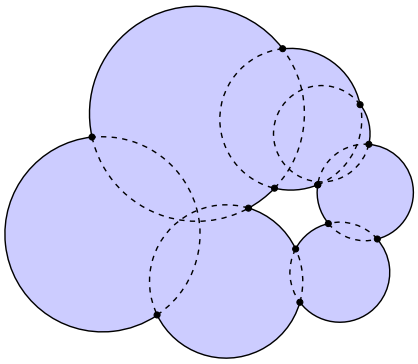


图 1

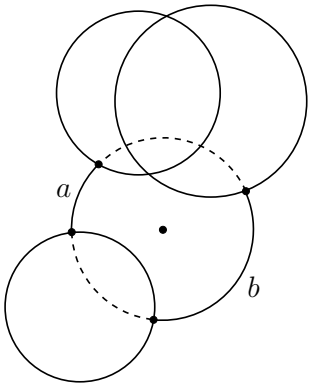


图 2

先讨论问题 1.1。不失一般性地，假设圆 $D_1, D_2, \ldots, D_n$ 两两不同；否则，对于出现多次的圆，我们可以保留一个并删去其它，而不改变问题的答案。如图 2 所示，每个圆周上未被其它圆覆盖的部分（以中央的圆为例：圆弧 a, b）构成 ∂D^*；被其它圆覆盖的部分（虚线部分）则不是 ∂D^* 的组成部分。求出构成 ∂D^* 的圆弧，就要求我们分别对于每一个圆，计算圆周上未被其它圆覆盖的部分。

定义“极角”运算。假设我们现在正在处理的圆的圆心是 O；定义圆周上的点 P 的极角 $\mathrm{Angle}(P)$ 为：从 x 轴正方向到向量 $\overrightarrow{OP}$ 的方向所转过的角度。这是一个有向角度，逆时针转动取正号，取值范围规定为 $[-\pi, \pi)$。极角运算是圆周到 $[-\pi, \pi)$ 的一一映射。我们转而计算 $[-\pi, \pi)$ 上未被覆盖的部分，每个圆在圆 O 上覆盖的部分（如果有）是一段圆弧，其对应的极角是 $[-\pi, \pi)$ 上至多两个区间的并。问题进一步转化为：

问题 1.1a 给定 $[-\pi, \pi)$ 上的 n' 个区间，求出没有被任何区间覆盖的部分，用若干个不交的区间表示。其中 $n' \le 2n$。

这是一个经典问题，这里简单介绍一下运行时间上界为 $O(n' \log n')$ 的算法。

利用扫描线的思想，令动点 P 从 $-\pi$ 逐渐移动到 π，在这个过程中维护有几个区间包含 P。这个值从 1 变成 0，以及从 0 变成 1 的位置，就是未覆盖区间的左右端点。在该过程之前，需要对所有区间端点进行排序，运行时间上界为 $O(n' \log n')$。

求出 $[-\pi, \pi)$ 上未被覆盖的若干个区间之后，可以在线性的时间内求出它们对应的圆弧。对于 n 个圆分别进行上述计算，就在 $O(n^2 \log n)$ 的最坏时间复杂度内求出了 $A_1, \ldots, A_m$，解决了问题 1.1。

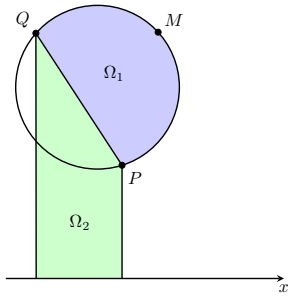


图 3

最后，讨论问题 1.2。如图 3 所示，假设我们要对于一段圆弧 $A_i = \overset{\frown}{PMQ}$ 计算 $\int_{A_i} (-y)\,\mathrm{d}x$（曲线方向为逆时针，$P \to Q$）。区域 Ω_1 是由 A_i 和 $\overrightarrow{QP}$ 围成的弓形区域，图中用淡蓝色标出；Ω_2 是由 $\overrightarrow{QP}$、x 轴和两条竖线围成的区域。对区域 Ω_1 应用格林公式，得

$$\begin{aligned} S_{\Omega_1} &= \oint_{\partial\Omega_1} (-y)\,\mathrm{d}x \\ &= \int_{A_i} (-y)\,\mathrm{d}x + \int_{\overrightarrow{QP}} (-y)\,\mathrm{d}x \\ \Rightarrow \quad \int_{A_i} (-y)\,\mathrm{d}x &= S_{\Omega_1} + \int_{\overrightarrow{QP}} y\,\mathrm{d}x \\ &= S_{\Omega_1} + S_{\Omega_2}, \end{aligned}$$

其中 S_{Ω_2} 是区域 Ω_2 的有向面积：如果 Q 的横坐标大于 P，则 $S_{\Omega_2} < 0$。最后一步的推导用到了 $\overrightarrow{QP}$ 的积分的几何意义。

Ω_1 是弓形，Ω_2 是梯形，它们的面积易于计算。使用上式可以在常数时间内计算出一段圆弧 A_i 的积分，从而解决了问题 1.2。

最后，我们再重新审视问题 1。我们在 $O(n^2 \log n)$ 的最坏时间复杂度内，求出组成 ∂D^* 的所有圆弧 $A_1, \ldots, A_m$，其中 m 有一个上界 n^2。接着我们对每一个 A_i 计算其曲线积分，每一个花费常数时间，并将它们相加得到 S_{D^*}。于是，我们得到一个运行时间上界为 $O(n^2 \log n)$ 的优秀算法，解决了问题 1。

3 球面面积并问题

在上一章中，我们利用格林公式将 D^* 上的二重积分转化为 ∂D^* 上的曲线积分，再对组成 ∂D^* 的各段圆弧分别求曲线积分。本章我们用同样的思路解决球面面积并问题。

问题 2（球面面积并问题） 给定球面上的 n 个球冠形区域 $D_1, \ldots, D_n$，球冠不能大于半球。定义

$$D^* = D_1 \cup D_2 \cup \ldots \cup D_n,$$

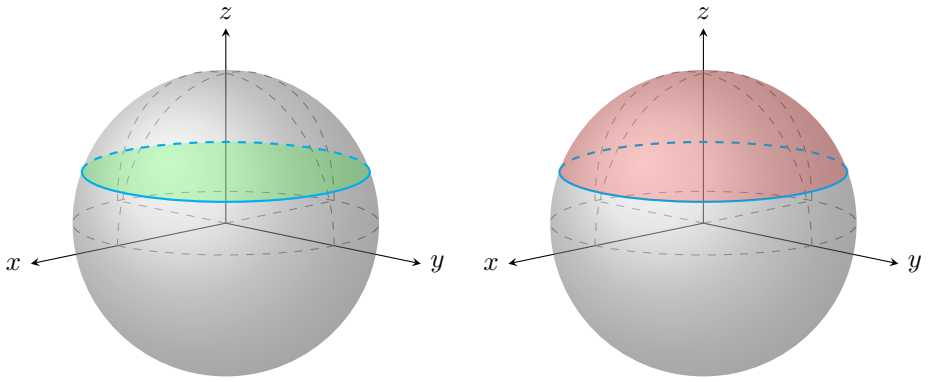
计算 D^* 的面积。

此处我们规定球冠不能大于半球，并不是说大的球冠无法被处理，而是因为在提出布尔运算后，我们可以使用较小球冠的补集来表示超过半球的球冠，从而无需讨论这种情况。作此规定可以为几何上的处理提供方便。

在介绍上述问题的解法之前，我们先简单介绍一下球面几何的基本知识，并约定一些术语的含义。

(1) 我们假定问题中的球面是单位球面，即球心在原点 $O = (0, 0, 0)$，半径为 1。能够这么做是因为球的平移、缩放对问题答案的影响容易处理。

(2) 用一个平面去截球面，截得的部分是一个**圆**（如图 4a 所示）。球面在该平面某一侧的部分称为**球冠**（如图 4b 所示）。这二者之间有对应关系，圆是球冠的边界，球冠是圆的内部。在不需要加以区分的情形下，它们的名称可以混用。



(a) 圆 (b) 球冠

图 4

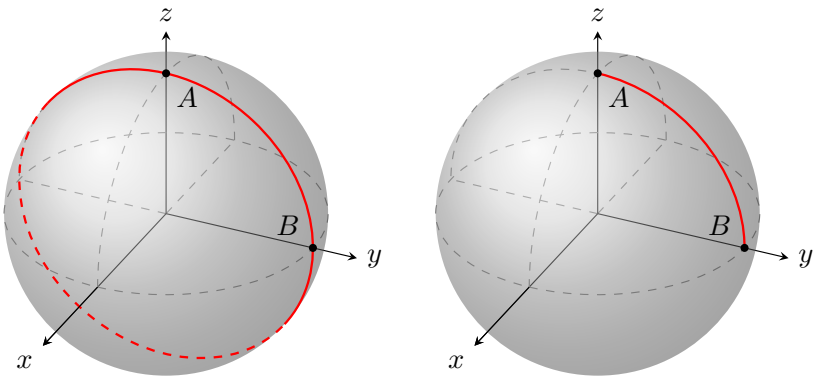
对于一个球冠，我们称其**顶点**为它在球面上的中心点。它在球冠上，且到球冠边界各点的距离相等。称其**底面**是对应的圆所在平面。

(3) 我们称球面上的两个点是**对径**的，当且仅当它们的连线穿过球心。这等价于它们关于球心对称。

(4) **大圆**是最大的圆，即过球心的平面去截球面时截得的圆。除大圆之外，其它的圆都称为**小圆**。大圆将整个球面分成两个大小相等的部分，即两个半球。大圆在球面几何中的地位好比平面几何中的直线。对于球面上的两个不同的点 A, B，如果它们不互相对径，则有唯一的大圆同时经过它们（如图 5a 所示），称为**球面直线** AB 或大圆 AB。此时 A, B 把这个大圆分成了长度不相等的两段，其中较短的一段称为**球面线段** AB（如图 5b 所示），它是球面上从 A 到 B 的最短路线。

每个大圆 AB 唯一确定一个平面 OAB。

如果 A, B 对径，则球面上从 A 到 B 的最短路线有无限条。任取一个过 A, B 的大圆，A, B 将其分为长度相等的两段，每一段都是从 A 到 B 的最短路线。我们将任意一条最短路线称为它们之间的广义球面线段。但为了指定一条广义球面线段，我们需要给出除 A, B 之外，它上面的另一个点 M。



(a) 球面直线 (b) 球面线段

图 5

(5) 如果 AB, AC 都是球面线段，那么可以定义**球面角** $\angle BAC$。如图 6 所示，过 A 点作 AB 的切线 AF，作 AC 的切线 AE，定义 $\angle BAC = \angle FAE$。这等价于平面 OAB 与 OAC 之间的二面角。

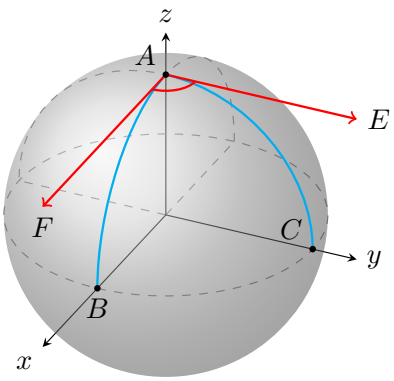


图 6

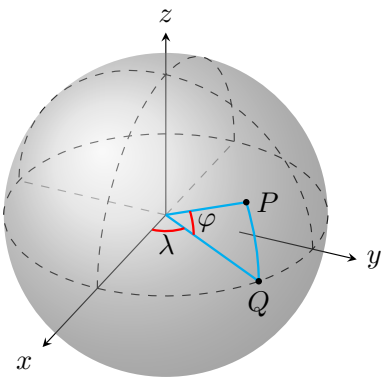


图 7

(6) 如果球面上三个点 A, B, C 两两不同、两两不对径、不在同一个大圆上，则可以定义**球面三角形** ΔABC。其三条边分别是球面线段 AB, BC, CA。这三条边把整个球面分成两个部分，其中较小那个部分是 ΔABC 的内部。球面三角形的三个内角均小于平角。

(7) 与地球上的情形一样，定义**南极点** $P_S=(0,0,-1)$、**北极点** $P_N=(0,0,1)$，并定义经线、纬线。具体地，连接 P_N 与 P_S 的所有球面线段被称为**经线**；平面 $z=z_0$ 割球面所得的圆被称为**纬线**。

我们可以用经纬度表示球面上的点，如图 7 所示。对于球面上的每个点 $P=(x,y,z)$，设实数 $\lambda\in[0,2\pi)$ 和 $\varphi\in[-\pi/2,\pi/2]$ 满足

$$\begin{cases} x=\cos\varphi\cdot\cos\lambda,\\ y=\cos\varphi\cdot\sin\lambda,\\ z=\sin\varphi, \end{cases}$$

就称 (λ,φ) 是点 P 的经纬表示。其中 λ 称为经度，φ 称为纬度。

两个极点有多种经纬表示方法。我们将两个极点从球面中删除，形成**去极球面**，这并不会改变我们希望计算的面积。去极球面的每一个点 P 有唯一的经纬表示

$$(\lambda,\varphi)\ \in\ [0,2\pi)\times\left(-\frac{\pi}{2},\frac{\pi}{2}\right).$$

上式右侧是平面上的一个矩形区域，称其为**经纬平面**。在经纬平面上，规定 λ 是横坐标、φ 是纵坐标。

点的经纬表示法建立了从去极球面到经纬平面的一一映射。我们可以利用这个映射，将球面上区域的面积转化为经纬平面上的二重积分，再通过格林公式与曲线积分联系起来。

定理 1 假设 Ω 是球面上由有限条分段光滑的曲线围成的区域，且北极点 P_N 不在 Ω 的边界上，那么有

$$S_\Omega=\oint_{\partial\Omega}(-\sin\varphi-1)\,\mathrm{d}\lambda+[P_N\in\Omega]\,4\pi.$$

证明

$$S_\Omega=\iint_\Omega \mathrm{d}a, \tag{1}$$

其中上式右侧是对 Ω 的曲面积分，$\mathrm{d}a$ 是面积微元。如图 8 所示，有

$$\mathrm{d}a=\cos\varphi\cdot\mathrm{d}\lambda\cdot\mathrm{d}\varphi.$$

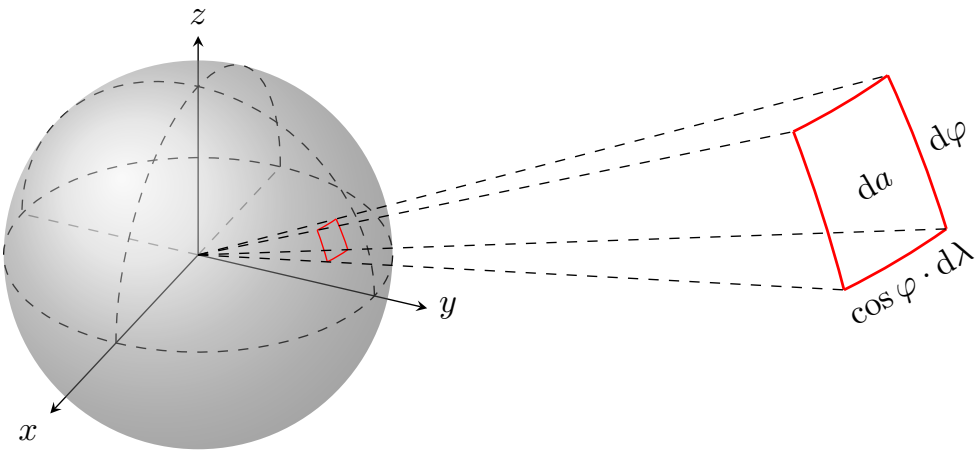


图 8

假设 $\hat{\Omega}$ 是 Ω 在经纬平面上的对应区域。(1) 式可以改写为

$$S_\Omega = \iint_{\hat{\Omega}} \cos\varphi \,\mathrm{d}\lambda \,\mathrm{d}\varphi,$$

右侧是经纬平面上的二重积分。请注意上式右侧的积分是不含两极的，在这一步中两极被去掉了，这不会改变区域的面积。

根据格林公式，经纬平面上有

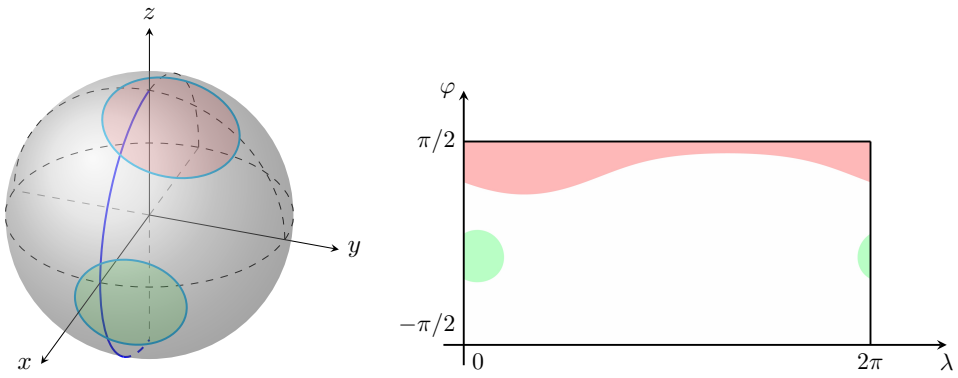
$$\oint_{\partial\hat{\Omega}} P\mathrm{d}\lambda + Q\mathrm{d}\varphi = \iint_{\hat{\Omega}} \left(\frac{\partial Q}{\partial \lambda} - \frac{\partial P}{\partial \varphi}\right) \mathrm{d}\lambda \,\mathrm{d}\varphi \tag{2}$$

成立。令

$$\begin{cases} P(\lambda,\varphi) = -\sin\varphi - 1, \\ Q(\lambda,\varphi) = 0, \end{cases}$$

并代入 (2)，得到

$$\begin{aligned} \oint_{\partial\hat{\Omega}} (-\sin\varphi - 1)\,\mathrm{d}\lambda &= \iint_{\hat{\Omega}} \cos\varphi \,\mathrm{d}\lambda \,\mathrm{d}\varphi \\ &= S_\Omega. \end{aligned}$$



(a) 去极球面 (b) 经纬平面

图 9

$\hat{\Omega}$ 的边界可以分为两部分：第一是经纬平面边界处的，第二是经纬平面内部的。如图 9 所示，其中第二部分与 $\partial\Omega$ 相同。第一部分中，对于左右边界有 $\mathrm{d}\lambda = 0$，对于下边界有 $-\sin\varphi - 1 = 0$，所以沿着这三条边界的曲线积分都为 0，从而我们也无需关心具体哪些部分是 $\hat{\Omega}$ 的边界。如果北极点在 Ω 内，则经纬平面的整个上边界都是 $\hat{\Omega}$ 的边界，方向为从右向左，其积分值为

$$\begin{aligned}&\int_{2\pi}^{0}\left(-\sin\frac{\pi}{2}-1\right)\mathrm{d}\lambda\\=&\int_{2\pi}^{0}(-2)\,\mathrm{d}\lambda\\=&\,4\pi.\end{aligned}$$

反之，如果北极点不在 Ω 内，则整个上边界都不是 $\hat{\Omega}$ 的边界。综上所述，对于 $\partial\hat{\Omega}$ 的第一部分，有

$$\int_{\text{Part-1}}(-\sin\varphi-1)\,\mathrm{d}\lambda = [P_N\in\Omega]\,4\pi.$$

结合 Part-2 $= \partial\Omega$，得到

$$\begin{aligned}\oint_{\partial\hat{\Omega}}(-\sin\varphi-1)\,\mathrm{d}\lambda &= \int_{\text{Part-1}}(-\sin\varphi-1)\,\mathrm{d}\lambda+\int_{\text{Part-2}}(-\sin\varphi-1)\,\mathrm{d}\lambda\\&= [P_N\in\Omega]\,4\pi+\oint_{\partial\Omega}(-\sin\varphi-1)\,\mathrm{d}\lambda.\end{aligned}$$

该等式左侧等于 S_Ω，右侧与定理 1 是一样的。 □

请注意在上述证明过程中去极球面与经纬平面的关系。我们在球面上进行的积分仅含变量 λ,φ，这和经纬平面是一样的。对于点、曲线、区域，它们在球面上与在经纬平面上没有区别，两种形式下它们的积分值也相同。唯一的区别是在经纬平面上应用格林公式的过程中，用区域定义其边界时 $\partial\Omega$ 与 $\partial\hat{\Omega}$ 的区别 —— 后者比前者多出一部分，即上文所述 $\partial\hat{\Omega}$ 的第一部分。

另一点需要补充的是，当曲线穿过子午线的时候，λ 有一个间断点。这时我们可以在该间断点的位置上把曲线切分为两段，这两段的 λ 都是连续的，从而其积分有良好定义；再利用积分的可加性，定义整条曲线的积分为这两段积分之和。为了叙述方便，文中仍然使用曲线积分的一般符号来表示它。

推论 S_Ω 与 $\oint_{\partial\Omega}(-\sin\varphi-1)\,\mathrm{d}\lambda$ 之间可以互相推出，即只要知道其中一个，就可以在常数的时间内计算出另一个的值。

回到问题 2。我们假定 $D_1,\ldots,D_n$ 两两不同。

在解决这个问题之前，以随机的角度旋转整个球面。具体来说，在球面上均匀随机选择一个点 P'_N，并通过旋转使该点到达 $(0,0,1)$。旋转完成后，再按前文的方法定义南北极、经纬度。这使得 $D_1,\ldots,D_n$ 中的某个圆穿过北极的概率是 0，我们假定它不会发生。另外，对于一条圆弧，如果其位置由 $D_1,\ldots,D_n$ 确定，而与旋转点的选取无关（例如某两个球冠顶点的连线），那么我们也假定它不穿过北极点。

根据推论，问题 2 转化为计算

$$\oint_{\partial D^*}(-\sin\varphi-1)\,\mathrm{d}\lambda.$$

与上一章同理，∂D^* 是由一些圆弧 —— 即每个圆未被其它圆覆盖的部分 —— 构成的。于是该问题转化为两个子问题：

问题 2.1 求出每个圆未被其它圆覆盖的若干圆弧。

问题 2.2 对于某一段有向圆弧 C，计算其曲线积分 $\int_C(-\sin\varphi-1)\,\mathrm{d}\lambda$。

先讨论问题 2.1。假设我们要对于圆 O' 求出圆周上未被覆盖的部分。该圆在平面 α 上，且点 O' 是平面 α 上该圆的圆心。在平面 α 上建立直角坐标系 $xO'y$，并以 O' 为中心来定义极角运算，在圆周上的点与 $[-\pi,\pi)$ 之间建立一一映射。使用上一章中解决问题 1.1 的方法，即可求出圆周上未被覆盖的部分，它们作为 $\partial\Omega$ 的定向也容易确定，这里不再赘述。

再讨论问题 2.2。如图 10a 所示，假设我们要计算圆弧 $C_1=PQ$ 的曲线积分

$$\int_{C_1}(-\sin\varphi-1)\,\mathrm{d}\lambda,$$

其中曲线的方向是 $P\to Q$。

第一种情况：C_1 是小圆的一部分。作有向球面线段 $C_2=\overrightarrow{QP}$，则 C_1 与 C_2 围成了一个弓形区域 A。根据推论，可以由 S_A 推出

$$\int_{C_1}(-\sin\varphi-1)\,\mathrm{d}\lambda+\int_{C_2}(-\sin\varphi-1)\,\mathrm{d}\lambda,$$

再结合 C_2 的积分来推出 C_1 的积分。问题转化为两部分：

问题 2.2a 计算球面上弓形的面积。

问题 2.2b 给定有向球面线段 $C_2=\overrightarrow{QP}$，计算

$$\int_{C_2}(-\sin\varphi-1)\,\mathrm{d}\lambda.$$

第二种情况：C_1 是大圆的一部分。如果 C_1 大于或等于半个大圆，则取 C_1 的中点 M，将其分为两个球面线段 $\overrightarrow{QM}$ 和 $\overrightarrow{MP}$；否则，C_1 本身就是球面线段。从而转化为问题 2.2b。

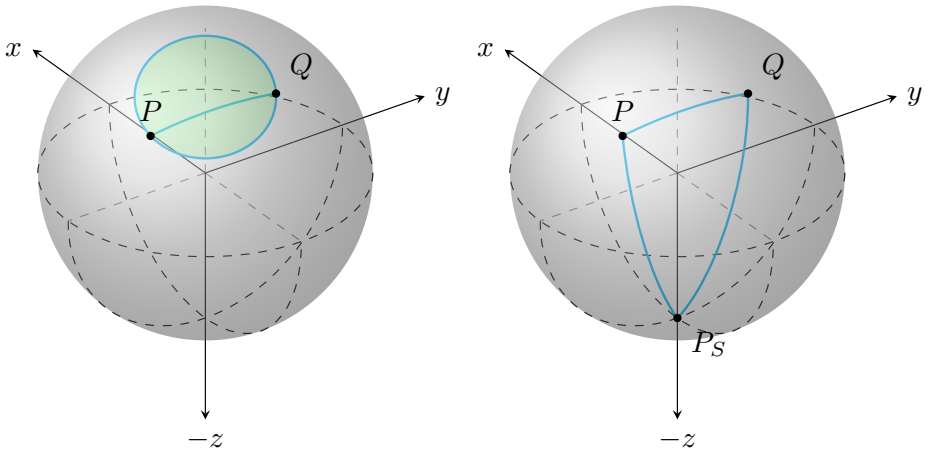


图 10

先讨论问题 2.2a。通过连接弓形的两个端点和它所在的球冠的顶点，我们将弓形表示为一个扇形与一个三角形的和或差。该问题被转化为下面两个：

问题 2.2c 计算球面扇形的面积。

问题 2.2d 计算球面三角形的面积。

接着讨论问题 2.2b。连接 P_SP 和 P_SQ，并判断 ΔP_SPQ 在 $\overrightarrow{QP}$ 的哪一侧。如果它在右侧，如图 10b 所示，则计算 $\overrightarrow{PQ}$ 的积分，再取其相反数。根据推论，有

$$\begin{aligned} S_{\Delta P_SPQ} &= \oint_{\partial \Delta P_SPQ} (-\sin\varphi - 1)\,\mathrm{d}\lambda \\ &= \int_{\overrightarrow{P_SP}} (-\sin\varphi - 1)\,\mathrm{d}\lambda + \int_{\overrightarrow{PQ}} (-\sin\varphi - 1)\,\mathrm{d}\lambda + \int_{\overrightarrow{QP_S}} (-\sin\varphi - 1)\,\mathrm{d}\lambda \\ &= \int_{\overrightarrow{PQ}} (-\sin\varphi - 1)\,\mathrm{d}\lambda. \end{aligned}$$

最后一步是因为对于 $\overrightarrow{P_SP}$ 和 $\overrightarrow{QP_S}$ 而言，$\mathrm{d}\lambda = 0$，所以其积分值为 0。

至此将问题 2.2b 转化为 2.2d。

通过简单的计算得到球冠的面积公式 $S = 2\pi h$，其中 h 是球冠顶点到底面的距离。利用球冠的对称性，我们容易导出扇形的面积公式

$$S = 2\pi h\theta,$$

其中 θ 是扇形的角度。

关于球面三角形的面积，[4] 指出

$$S_{\triangle ABC} = \angle A + \angle B + \angle C - \pi.$$

其中 $\angle A, \angle B, \angle C$ 分别表示以 A, B, C 为顶点的球面角。[7] 给出了一个数值稳定性更好的公式

$$\tan\frac{S_{\triangle ABC}}{2} = \frac{\tan\frac{1}{2}a\ \tan\frac{1}{2}b\ \sin\angle C}{1+\tan\frac{1}{2}a\ \tan\frac{1}{2}b\ \cos\angle C},$$

其中 a、b 分别指 $\angle A$、$\angle B$ 的对边 BC 与 CA 的长度。

至此，问题 2 已被完整解决。该算法的大体结构与上一章解决问题 1 的方法类似，且具有相同的运行时间上界 $O(n^2 \log n)$。

4 布尔运算

现实应用中，我们要计算的通常不是 n 个区域的并。比如，地球周围的空间中有 n 颗定位卫星，希望求出地表有多大的区域能与至少三颗卫星直连。再比如，在辅助设计工艺品时，用户反复使用交、并、补等运算，人为地规定一个复杂区域。即使只考虑面积并问题，也可以用三个半球面的交来表示球面三角形，进而用三角形的并表示球面多边形。这就提出了计算更一般的复合区域面积的要求。

k 元布尔函数 $F(x_1, \ldots, x_k)$ 是一个 $\{0,1\}^k \mapsto \{0,1\}$ 的映射。$x_1, \ldots, x_k$ 称为其参数。

假设 $D_1, \ldots, D_n$ 是给定的球冠。可以用一个 n 元布尔函数 $F(\lambda_1, \ldots, \lambda_n)$ 来定义复合区域 D^*：

$$D^* = \{P \mid F(\lambda_1, \lambda_2, \ldots, \lambda_n) = 1\},$$

即

$$[P \in D^*] = F(\lambda_1, \lambda_2, \ldots, \lambda_n) \qquad (\forall P),$$

其中 λ_i 表示点 P 是否在 D_i 中，即 $\lambda_i = [P \in D_i]$。严格地说，λ_i 是一个关于 P 的函数 $\lambda_i(P)$，但为了清晰，在我们考察的点 P 十分明确的前提下，省略了其参数 P，下文将沿用这种省略。这两个式子指出布尔函数 F 定义了 $D_1, \ldots, D_n$ 的哪些组合在复合区域 D^* 中。

有一类基本的、简单的布尔函数，称为布尔运算符。对于一个布尔运算符 $f(x_1, \ldots, x_k)$，必须存在一种数据结构，支持：

(1) INITIALIZE。初始化数据结构，令 $x_i = 0$ $(\forall i \in [1, k])$。

(2) MODIFY(i, b)。修改参数的值，令 $x_i = b$。

(3) QUERY。查询函数的值，即 $f(x_1, \ldots, x_k)$。

其中 INITIALIZE 操作需要在 $O(k)$ 的时间复杂度内完成，而 MODIFY 和 QUERY 必须在常数运行时间内完成。

我们将布尔运算符的内部逻辑视为一个黑盒，而仅通过它的数据结构来获得信息。所有 $\Theta(1)$ 元的、可计算的布尔函数，都属于布尔运算符。

我们用递归的方式来定义布尔表达式。一个布尔表达式 α 要么是某个 λ_i $(i \in [1,n])$，要么有

$$\alpha = f(\beta_1, \beta_2, \ldots, \beta_k),$$

其中 f 是一个布尔运算符，$\beta_1 \ldots, \beta_k$ 分别是一个布尔表达式。

定义表达式 α 的长度

$$|\alpha| = \begin{cases} 1, & \text{if } \alpha = \lambda_i, \\ 1 + \sum_{j=1}^{k} |\beta_j|, & \text{if } \alpha = f(\beta_1, \ldots, \beta_k). \end{cases}$$

我们以布尔表达式的形式来给出布尔函数 F，正式提出以下问题：

问题 3（球面复合区域面积问题） 给定球面上 n 个球冠形区域 $D_1, \ldots, D_n$。再给定一个长度为 m 的布尔表达式 α_F，定义

$$D^* = \{P \mid F(\lambda_1, \lambda_2, \ldots, \lambda_n) = 1\},$$

其中 $F(\lambda_1, \ldots, \lambda_n) = \alpha_F$ 是由布尔表达式 α_F 定义的 n 元布尔函数。希望计算 D^* 的面积。

仍然假定 $D_1, \ldots, D_n$ 两两不同，且没有球冠大于半球。能够如此假设是因为大于半球的球冠能够用一个较小球冠的补集来表示。进一步地，我们假设它们的边界也互不重合，即不存在两个球冠 D_i 和 D_j，它们的并等于整个球面。

利用上一章的方法，我们只需求出 ∂D^* 就可以算出 D^* 的面积，为此分别考察每一个圆 ∂D_i。

在上一章中，

$$F = \lambda_1 \vee \lambda_2 \vee \ldots \vee \lambda_n,$$

我们判断出，一段圆弧在 ∂D^* 中，当且仅当它没有被其它球冠覆盖，且方向相对于球冠内部是逆时针。这是因为，对于一段满足条件的有向圆弧，它的左侧（即相对于球冠而言的内侧）在 D^* 内，而右侧在 D^* 外，于是它是 ∂D^* 的成分；对于被其它球冠覆盖的圆弧，它的左右两侧均在 D^* 内，于是它就不是 ∂D^* 的成分。沿用此思路，如果 A 是 ∂D_i 的一段有向圆弧，A 的左侧在 D^* 内，右侧在 D^* 外，那么 $A \subset \partial D^*$；如果 A 的左侧在 D^* 外，右侧在 D^* 内，那么 A 的反向是 ∂D^* 的成分。为此，我们计算以下问题：

问题 3a 计算 ∂D_i 的所有部分，满足其左侧在 D^* 内。

问题 3b 计算 ∂D_i 的所有部分，满足其右侧在 D^* 内。

上述两个问题的答案可以快速合并，成为我们想要的信息。这两个问题唯一的区别是，对于 ∂D_i 左侧有 $\lambda_i = 1$；对于其右侧有 $\lambda_i = 0$。对于另外的 j $(j \neq i)$，∂D_i 上被 D_j 覆盖的部

分有 $\lambda_j = 1$，未被 D_j 覆盖的部分有 $\lambda_j = 0$。这两个问题的解决方法也是一致的，下面只讨论问题 3a 的解法。

利用上一章定义的极角运算，将 ∂D_i 上的点映射到 $[-\pi,\pi)$ 上，问题 3a 转化为：

问题 3.1 $\forall i \in [1,n]$，区间 $[-\pi,\pi)$ 上至多两个给定区间使 $\lambda_i = 1$，对于其它位置有 $\lambda_i = 0$。求出使 $F(\lambda_1,\ldots,\lambda_n) = 1$ 的所有位置，用 $[-\pi,\pi)$ 上若干个不交的区间来表示。

利用扫描线的思想，令动点 P 从 $-\pi$ 向 π 移动，并在移动的过程中维护 $F(\lambda_1,\ldots,\lambda_n)$ 的值。移动过程中会有 $O(n)$ 个“事件”——某个 λ_i 的取值发生了变化，F 值也随之更新。F 值从 1 变化为 0，以及从 0 变化为 1 的位置，就是某个所求区间的左右端点。

用表达式树来显示布尔表达式的结构。对于一个布尔表达式 α，定义其表达式树

$$T(\alpha) = \begin{cases} \lambda_i & \text{if } \alpha = \lambda_i, \\ f \text{ with children } T(\beta_1), T(\beta_2), \ldots, T(\beta_k) & \text{if } \alpha = f(\beta_1,\ldots,\beta_k). \end{cases}$$

在第一种情况中，$T(\alpha)$ 唯一的结点称为“输入结点”；第二种情况中的根节点称为“运算结点”，f 称为该结点的运算符。表达式树是有根树，输入结点都是叶结点。运算结点的孩子是一个有序的列表，对应运算符的各个参数，顺序不可调换。

建出 α_F 的表达式树，下面所有讨论都在这棵树上进行。容易验证，一个表达式的长度和其表达式树的结点数相等，所以 $T(\alpha_F)$ 有 m 个结点。对于每一个结点 u，称子树 u 表示 u 和它的全体后代形成的子树，子树 u 规定了一个布尔表达式，有时也用结点的名称 u 来指代该表达式，或是该表达式的值。

当 $\lambda_1,\ldots,\lambda_n$ 取值确定后，定义结点 u 的值 $W(u)$ 为子树 u 规定的表达式的值，即

$$W(u) = \begin{cases} \lambda_i, & \text{if } u \text{ is an input node,} \\ f_u(W(v_1), W(v_2), \ldots, W(v_k)), & \text{otherwise,} \end{cases}$$

其中 $v_1,\ldots,v_k$ 是 u 的子结点。在第二种情况中，f_u 指结点 u 的运算符。自底向上反复应用上式，就可以计算出根结点的值，即 $F(\lambda_1,\ldots,\lambda_n)$。

当 $\lambda_1,\ldots,\lambda_n$ 取值确定后，输入结点的值也被确定。这时，包含变量 λ_i 的输入结点等价于包含 0 元运算符的运算结点，该节点的值要么恒为 0，要么恒为 1，被 λ_i 的取值确定。我们在表达式树上将它们表示为运算结点，这样表达式树就与 $\lambda_1,\ldots,\lambda_n$ 没有直接关系了。但

为了行文方便，我们仍然将这些结点称为“包含 λ_i 的输入结点”。当某个 λ_i 被修改时，包含 λ_i 的若干输入结点的运算符需要被修改。每个结点至多被修改 $O(1)$ 次。我们需要解决如下问题：

问题 3.2 设计一个数据结构，支持三种操作：

(1) INITIALIZE-TREE。初始化数据结构。建出表达式树，令所有输入结点的值为 0。

(2) MODIFY-TREE(u, b)。修改输入结点 u 的值，令 $W(u) = b$。

(3) QUERY-TREE。查询根节点的值 $W(\text{root})$。

当一个叶子被修改的时候，它所有祖先的值也需要重新计算。在有些特殊的问题中，树的深度不超过 $O(\log m)$，则可以使用朴素方法，自底向上依次更新这些结点的值。

算法 1 数据结构的朴素实现

```
 1: procedure INITIALIZE-TREE-SIMPLE
 2:     for each node u bottom-up do
 3:         assume u = f_u(v_1, ..., v_k)              ▷ 此处用结点名称指代其值
 4:         let u.processor be a new data structure of operator f
 5:         u.processor.INITIALIZE
 6:         for each v_i do
 7:             u.processor.MODIFY(i, W(v_i))
 8:         end for
 9:         W(u) ← u.processor.QUERY
10:     end for
11: end procedure
12:
13: procedure MODIFY-TREE-SIMPLE(u, b)
14:     W(u) ← b
15:     if u = root then exit
16:     p ← u.parent
17:     assume p = f_p(v_1, ..., v_k) and v_i = u
18:     p.processor.MODIFY(i, b)
19:     MODIFY-TREE-SIMPLE(p, p.processor.QUERY)
20: end procedure
21:
22: function QUERY-TREE-SIMPLE
23:     return W(root)
24: end function
```

上面的伪代码展示了数据结构的朴素实现。INITIALIZE-TREE-SIMPLE 为每个结点创建其运算符的数据结构黑盒 *u.processor*，并将黑盒的参数设置为正确的值。子过程 MODIFY-TREE-SIMPLE 递归地依次访问需要重新计算的结点。对于每个结点而言，其参数中至多有一个改变了。第 18、19 行利用黑盒，在常数运行时间内将参数调整为正确的值，并取得新

的运算结果。INITIALIZE-TREE-SIMPLE 的运行时间是 $\Theta(m)$，MODIFY-TREE-SIMPLE 的运行时间以树的最大深度为上界，QUERY-TREE-SIMPLE 的运行时间是常数。如果树的最大深度不超过 $O(\log m)$，则该算法已经足够高效。

在树深较大的情况下，使用树链剖分。对每个结点 u，记 s_u 为子树 u 的结点数。对于非叶结点 u，若它的孩子中 s 最大的结点是 v，则称 (u,v) 是重边，v 是 u 的重孩子；u 到其它孩子的边是轻边，它们是 u 的轻孩子。以下性质成立：

- 重边形成若干条链，每个结点恰好属于一条链（链可以只有一个结点）。这些链称为重链。
- 从任意结点到根的路径至多经过 $O(\log m)$ 条轻边，至多经过 $O(\log m)$ 条重链。

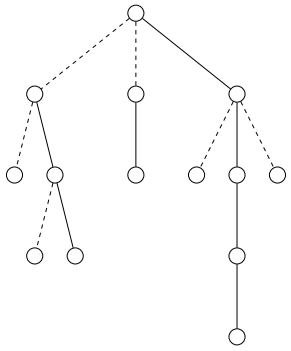


图 11

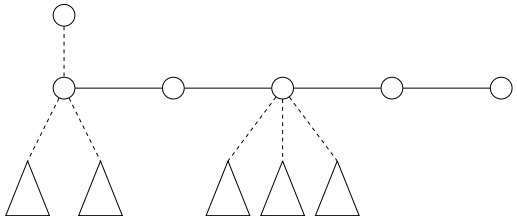


图 12

图 11 展示了一个树链剖分的结构。图中用实线表示重边，用虚线表示轻边。图 12 展示了一条重链的结构，图中为了清晰，将轻孩子画在链的下方，而实际上轻孩子与重孩子之间有固定的顺序。

对于任意结点 u，假设 $u = f_u(v_1, \ldots, v_k)$，其中 v_i 是其重儿子（即重链上的下一个结点）。那么，我们可以将除 v_i 外的 $k-1$ 个参数视作常量，定义

$$g_u(v_i) = f_u(v_1, \ldots, v_i, \ldots, v_k),$$

那么 $g_u(x)$ 便是一个一元布尔函数。特别地，如果 f_u 是 0 元的，没有参数，那么我们添加一个虚拟参数 x，并令

$$g_u(x) = f_u(\,),$$

此时 $g_u(x)$ 虽然形式上是一元函数，但有 $g_u(0) = g_u(1)$ 成立，即函数值与参数值无关。此外，定义两个一元函数 f、g 的复合

$$(f \circ g)(x) = f(g(x)),$$

它满足结合律。

对于一条重链，按照顺序复合链上的 g_u 将得到一个一元函数 g^*，它满足

$$g^*(0) = g^*(1) = W(r),$$

其中 r 是链顶，即这条链最顶部的结点。g_u 的定义直接说明了上式的正确性。

对每一条重链，用线段树维护一元函数 g_u 的序列，并维护链顶的值 $W(r)$。当叶结点的 0 元运算符被修改时，它在重链上的一元函数也被修改。在这棵线段树上作出这个修改，并重新查询全体一元函数的复合，来计算 $W(r)$ 的值。$W(r)$ 更新后，r 的父亲（如果存在）有一个轻孩子的值发生了变化，从而引起它的一元函数发生变化，进一步重复这个过程。算法 2 中的伪代码展示了一种实现。

在子过程 MODIFY-CHILD-VALUE 中，我们仍然利用运算符的黑盒来计算一元函数 g_u，这个子过程的运行时间是常数。*segment-tree*.QUERY 查询的总是全体一元函数的复合，该信息已经存储在线段树的根结点中，所以运行时间也是常数。在一次 MODIFY-TREE 中，*segment-tree*.MODIFY 的调用次数不超过 $O(\log m)$，每次运行时间不超过 $O(\log m)$，故 MODIFY-TREE 的运行时间有上界 $O(\log^2 m)$。

在此基础上，使用“全局平衡二叉树”代替线段树，可以使 MODIFY-TREE 的运行时间上界降至 $O(\log m)$，见 [1]。另外，INITIALIZE-TREE 的运行时间是 $\Theta(m)$，QUERY-TREE 的运行时间是 $\Theta(1)$。我们已经得到问题 3.2 的一个高效解法。

接着，应用上述数据结构解决问题 3.1。在“扫描”开始之前，对数据结构进行初始化需要 $\Theta(m)$ 的时间。在“扫描”的过程中，表达式树上的每个叶子会被修改常数次，这一部分的运行时间上界是 $O(m \log m)$。于是，问题 3.1 以 $O(m \log m)$ 的运行时间上界被解决。

在本章的开头，我们把问题 3 转化为 $\Theta(n)$ 个互相独立的问题 3.1。从而利用上述方法得到问题 3 的一个 $O(nm \log m)$ 运行时间上界的算法。

算法 2 数据结构的高效实现

```
1: procedure INITIALIZE-TREE
2:     INITIALIZE-TREE-SIMPLE
3:     build heavy-light decomposition
4:     calculate g_u  (∀u)
5:     for each chain c do
6:         let c.segment-tree be a new segment tree of g_u
7:         initialize c.segment-tree
8:     end for
9: end procedure
10:
11: procedure MODIFY-CHILD-VALUE(u, v, b)
12:     assume v_1, ..., v_k are children of u
13:     assume v = v_i, and v_j is the heavy-child of u (i ≠ j)
14:     u.processor.MODIFY(i, b)
15:     u.processor.MODIFY(j, 0)
16:     g_u(0) ← u.processor.QUERY
17:     u.processor.MODIFY(j, 1)
18:     g_u(1) ← u.processor.QUERY
19: end procedure
20:
21: procedure RECALCULATE(u)
22:     assume u is the i^th node of chain c
23:     r ← c.top
24:     c.segment-tree.MODIFY(i, g_u)
25:     g* ← c.segment-tree.QUERY
26:     W(r) ← g*(0)                          ▷ 此时 g*(0) = g*(1)
27:     if r ≠ root then
28:         p ← r.parent
29:         MODIFY-CHILD-VALUE(p, r, W(r))
30:         RECALCULATE(p)
31:     end if
32: end procedure
33:
34: procedure MODIFY-TREE(u, b)
35:     g_u(0) ← b,  g_u(1) ← b
36:     RECALCULATE(u)
37: end procedure
38:
39: function QUERY-TREE
40:     return W(root)
41: end function
```

5 应用与讨论

在本章，我们讨论球面复合区域面积问题的几种变形、推广，以及它们的应用。

球面多边形

球面多边形是由若干条球面线段首尾相连而围成的区域，现实中常用来表示地表的某个区域。

我们的算法能够处理球面多边形。我们可以用三个“半球面”的交表示三角形，其中半球面是指大圆某一侧的区域。进一步地，使用球面三角形的并来表示球面多边形，在这里我们并不反复使用常见的二元“逻辑或”运算符，而是直接使用 k 元逻辑或运算符。上述方法中，球面多边形是由某个表达式规定的，而这个表达式树的深度是常数。表示出球面多边形之后，它们可以继续参与布尔运算，来形成更复杂的复合区域。

平面复合区域

与实际应用不同的是，在算法竞赛中，常常将问题抽象为平面上的面积问题。这是因为球面几何代码编写繁琐，且通常与思维考察点关系不大，故通过抽象为平面问题将其省去了。

通过将第二章与第四章的内容进行结合，我们能够直接得到一个算法，它能够计算平面上若干个圆进行布尔运算规定的复合区域的面积。但是这还不够，因为球面上的“直线”是一种特殊的圆，而平面上不是。为此，除了圆，我们再引入另一种区域 —— 半平面。

半平面是指一条有向直线左侧的区域。这条有向直线就是半平面的边界，且拥有正确的定向。

模仿极角运算，我们可以定义关于有向直线的**点积**运算。假设一条有向直线上的点集是

$$\vec{a} + \vec{b}\,t \qquad (t \in \mathbb{R}),$$

其中 t 沿直线方向增大。定义直线上一个点 P 的点积

$$\mathrm{Dot}(P) = (P - \vec{a}) \cdot \vec{b},$$

右侧是两个向量的点积。点积运算建立了有向直线上的所有点到区间 $(-\infty, +\infty)$ 的一一映射。容易验证，对于任意两个区域 D_i 和 D_j，它们都是圆或者半平面，D_j 覆盖 ∂D_i 的部分映射到数轴上是至多两个区间的并。这样，我们就可以继续沿用第四章的方法，以扫描线的思想来求出复合区域的边界 ∂D^*。对 ∂D^* 的积分是简单的。

如此，我们就解决了含圆和半平面的“平面复合区域面积”问题，运行时间有同样的上界 $O(nm \log m)$。用半平面表示平面上的多边形是容易的，这里不再赘述。该方法可以直接通过三道 NOI 中出现过的此类问题：

- NOI 2004《降雨量》；
- NOI 2005《月下柠檬树》；
- NOI 2009《描边》。

容易计算边界的情形

在某些题目中，因为题目具有特殊性质，我们可以采用针对题目特殊性的方法，以快速计算 ∂D^*。分段计算 ∂D^* 的曲线积分不会成为运行效率的瓶颈，因为这一部分的运行时间正比于 ∂D^* 中圆弧的数量，这是前一个阶段（即快速计算 ∂D^*）的输出。

例 1《月下柠檬树》[2] 给定平面上的 n 个圆 $A_1,\ldots,A_n$，它们的圆心都在 x 轴上，且圆心的 x 坐标递增。对于相邻两个圆 A_i 与 A_{i+1}，如果它们互不包含，则作出它们的两条外公切线，并将四个切点连成梯形，记这个梯形区域为 B_i。定义

$$D^* = \bigcup_i A_i \cup \bigcup_i B_i,$$

计算 D^* 的面积。

在本题中，∂D^* 关于 x 轴对称，只需要计算 x 轴上方的部分，记为 C 并忽略方向。将每一个圆 A_i 与它左侧的梯形 B_{i-1}（如果存在）的并记为 A'_i。初始状态令 C 为空，从左到右依次加入 A'_i，并在每一个图形加入后计算新的边界 C'。容易证明，每个 A'_i 加入时，原先的 C 恰有一个“后缀”（最右侧连续的一段）被 A'_i 覆盖，从而被删除，代之以 $\partial A'_i$ 的某个部分。用栈维护这一过程，就可以在 $\Theta(n)$ 的运行时间内计算出 C，从而得到 ∂D^*。

综上，我们得到原问题的一个 $\Theta(n)$ 运行时间的算法。该算法达到了本题的时间复杂度理论下界，远快于出题者参考算法的 $O(n^3)$。

DAG 形式的布尔函数

在第四章中，我们用树形结构来表示从各个 λ_i 到最终布尔函数的计算过程。我们用这个结构解决实际问题，很大程度上依靠了布尔运算符的灵活性。例如，我们知道

$$a \oplus b = (a \vee b) \wedge \neg(a \wedge b),$$

即我们可以用基本的逻辑运算符 $\neg$、$\wedge$、$\vee$ 去定义“异或”运算。但右侧的展开形式在我们的算法中是行不通的，因为 a、b 各出现了两次，而树形结构没办法将一个结点的值作为多个运算符的输入。如果将整棵子树复制一份，时间复杂度将上升为指数级，无法接受。我们采取的方法是直接定义一个“异或”运算符，在这个运算符内处理中间变量的复用。

上述方法只能处理“局部的”变量复用。在计算机辅助设计中，常常出现以下情况：整个运算过程都由用户给定，每一次运算结果都会被存储为中间变量，它们在将来能够被任

意访问。对于这样的高度自由的运算过程，我们只能用 DAG（有向无环图）来表示它。在图中，每一个结点表示一个中间变量，它是一个运算符的直接结果，运算符的各个参数对应结点的各条入边。此处，一个结点的入边是有序的列表，对应运算符的各个参数的顺序，不能随意调换。

通过牺牲一些运行效率，我们的算法能够处理这种情况。假定 DAG 的边数是 m。那么当 $\lambda_1,\ldots,\lambda_n$ 的值确定后，最终的布尔函数 F 的值可以在 $O(m)$ 的时间内计算得到。我们不再使用数据结构去维护 F 值，而是在某个变量发生变化后，以 $O(m)$ 的时间完全重新计算 F 值。整个算法的时间复杂度为 $O(n^2m)$。

例 2《**计算体积**》[3] 输入 n 条指令，每条指令为以下几种之一：

(1) 创建一个球；

(2) 创建一个每条边都和坐标轴平行的长方体；

(3) 输入在这条指令前面的两条指令编号 i、j，创建一个立体图形为第 i 条指令与第 j 条指令的交/并/差。

对于每一条指令，输出它创建的立体图形的体积。

离线后对所有图形一起处理。本题的数据规模、值域、精度要求都比较低，可以对一个维度使用数值积分，将问题转化为二维问题。这个二维问题的输入只包含圆和长方形，运算由 DAG 给出，DAG 共有 $O(n)$ 条边。特别的是，我们需要求出 DAG 上每一个结点 u 的值规定的复合区域 D_u^* 的面积，即所有结点都是输出结点。当某个变量 λ_i 改变时，重新计算 DAG 上所有结点的值总共需要 $O(n)$ 的时间，故解决二维子问题需要 $O(n^3)$ 的时间。解决原问题的时间复杂度是 $O(n^3I)$，其中 I 是数值积分方法调用求值函数的次数。

球并的表面积

例 3 给定空间中 n 个球，计算它们的并的表面积。

对于一个球面而言，其中未被其它球覆盖的部分，就成为球并的表面。分别对于每一个球面，计算未被覆盖部分的面积，即球面上

$$D^* = U - \bigcup_i D_i$$

的面积，其中 U 是整个球面，D_i 是该球面被第 i 个球覆盖的区域，是一个球冠。这就将问题转化为球面复合区域面积问题，得到一个 $O(n^3\log n)$ 的算法。

另一个有趣的问题是计算球并的体积。虽然上述方法可以求出球并的边界，但笔者尚未找到计算球并的体积的精确算法。这个问题尚待继续研究。

6 总结

本文介绍了球面复合区域面积问题，并提出格林公式方法来解决它。该问题在现实中有广泛的应用，主要是在地球表面上的面积问题。球面几何因其代码冗长、细节繁多，不适合用于算法竞赛的命题，但我们的算法能够作一些修改以适应平面情形，从而在算法竞赛中体现其优势。

然而，对这个问题的研究还没有结束。在实际应用中，如果我们需要一块区域更精确的面积，就应该将地球表面抽象成一个椭球面，从而需要解决椭球面复合区域面积问题。此外，第五章中对于 DAG 形式的布尔函数，我们仅仅得到一个运行时间上界为 $O(n^2m)$ 的算法，它的运行效率或许有进步空间。上述两个相关问题尚待进一步研究。

致谢

感谢周航锐同学提供《月下柠檬树》[2] 的线性算法及其程序实现，并协助了部分图片的绘制工作。

感谢徐哲安同学与我讨论格林公式方法的应用。

参考文献

[1] 杨哲. SPOJ375 QTREE 解法的一些研究. 中国信息学奥林匹克国家集训队作业, 2007: 9-11.

[2] 胡伟栋. 月下柠檬树. 全国青少年信息学奥林匹克竞赛, 2005. https://www.luogu.com.cn/problem/P4207.

[3] 徐明宽. 二维计算几何相关算法与实战应用. 全国青少年信息学奥林匹克冬令营, 2018: 70-72.

[4] 人民教育出版社, 课程教材研究所, 中学数学教材实验研究组. 普通高中课程标准实验教科书数学选修 3-3（B 版）: 球面上的几何 [M]. 人民教育出版社, 2008: 27-28.

[5] 常庚哲, 史济怀. 数学分析教程 [M]. 第 3 版. 中国科学技术大学出版社, 2012.

[6] Area. ACM/ICPC Asia Regional Nanjing Online, 2013. http://acm.hdu.edu.cn/showproblem.php?pid=4748.

[7] Todhunter I. *Spherical Trigonometry, for the use of colleges and schools: with numerous examples*[M]. Macmillan, 1863: 67-71.

浅谈群论在信息学竞赛中的简单应用

宁波市镇海中学 虞皓翔

摘 要

本文介绍了有关群论中 Polya 定理，群的判定和表示以及 Schreier-Sims 等算法，以及它们在 OI 中的应用，并对计算群论及其算法进行了初步的研究。

引 言

群论是抽象代数中研究群的理论。群在抽象代数中具有重要的地位。群的概念在数学的许多分支中都有出现，而且群论的研究方法对抽象代数的其他分支也有重要影响。

1 置换

1.1 定义

1.1.1 置换

定义 1.1.1 (置换)**.** 一个置换可以看成一个一一映射 *(*双射*)* $g : \{1, 2, \cdots, n\} \to \{1, 2, \cdots, n\}$，满足 $g(i) = p_i$。

为了强调置换是一种变换/映射，我们通常使用 $2 \times n$ 的矩阵来表示一个置换：

$$g = \begin{pmatrix} 1 & 2 & \cdots & n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$$

通常，我们将 n 称为该置换的**大小**或**长度**，大小为 n 的置换又称为 n 元置换。

1.1.2 逆序数和奇偶性

定义 1.1.2 (逆序数和奇偶性)**.** 对于一个 n 元置换 g，如果 $i, j \in \{1, 2, \cdots, n\}$ 满足 $i < j$ 且 $g(i) > g(j)$，则称 (i, j) 是一个**逆序对**。

一个置换 g 中所有逆序对的总数叫做这个排列的**逆序数**，记作 $N(g)$。

若一个置换的逆序数为奇数，则称它为**奇置换**，否则称它为**偶置换**。

一个置换 g 的符号定义为 $\operatorname{sgn}(g)=(-1)^{N(g)}$，即奇置换的符号为 -1，偶置换的符号为 1。

1.1.3 置换的合成和逆

定义 1.1.3 (置换的合成)**.** 设 $f=\begin{pmatrix}1 & 2 & \cdots & n\\ i_1 & i_2 & \cdots & i_n\end{pmatrix}, g=\begin{pmatrix}1 & 2 & \cdots & n\\ j_1 & j_2 & \cdots & j_n\end{pmatrix}$。则它们的**合成**就是按照先 f 后 g 的顺序做两次变换，记作 $g\circ f$，简写为 gf。具体地，就是将 f 的下行和 g 的上行对应，则新的置换就是以 f 的上行为上行，以 g 的下行为下行。

用公式表达就是，$(g\circ f)(k)=g(f(k))=j_{i_k}$。

定义 1.1.4 (逆置换)**.** 对于置换 $g=\begin{pmatrix}1 & 2 & \cdots & n\\ p_1 & p_2 & \cdots & p_n\end{pmatrix}$，定义它的逆 $g^{-1}=\begin{pmatrix}p_1 & p_2 & \cdots & p_n\\ 1 & 2 & \cdots & n\end{pmatrix}$。即 g^{-1} 满足 $g^{-1}(g(i))=i$。

1.2 置换的循环表示

1.2.1 循环

定义 1.2.1 (循环)**.** 对 L 个元素的一个排列 $a_1,a_2,\cdots,a_L$，如果置换 g 满足 $g(a_1)=a_2, g(a_2)=a_3,\cdots,g(a_{L-1})=a_L, g(a_L)=a_1$，则称置换 g 为一个**循环** *(*或轮换*)*[1]，简记为 $(a_1\ a_2\ a_3\ \cdots\ a_{L-1}\ a_L)$，$L$ 称为该循环的**长度**。

1.2.2 循环分解和循环表示

定义 1.2.2 (循环表示)**.** 对 n 元置换 g，称下式为 g 的循环表示:

$$g=(a_{11}a_{12}\cdots a_{1L_1})(a_{21}a_{22}\cdots a_{2L_2})\cdots\left(a_{y1}a_{y2}\cdots a_{yL_y}\right)$$

其中 $g(a_{ij})=a_{i,j+1}$，$g(a_{iL_i})=a_{i1}$，$L_1+L_2+\cdots+L_y=n\,(1\le j<L_i)$，且所有 a_{ij} 互不相同。

在上式中，把 $(a_{i1}a_{i2}\cdots a_{iL_i})$ 为一个**循环**，L_i 为该循环的**长度**，y 为置换 g 的循环表示中的**循环数**，记作 $\#(g)$，在上下文已知的情况下可记为 $\#$。

如果有一个循环的**长度**为 1，则可以省略不写。

设 $g=(a_{11}a_{12}\cdots a_{1L_1})(a_{21}a_{22}\cdots a_{2L_2})\cdots\left(a_{y1}a_{y2}\cdots a_{yL_y}\right)$，则 g 等于组成它的所有循环的合成。

[1] Cycle

1.3 置换的循环指标

刻画置换性质的另一大工具是循环指标[2]。它在置换中的地位如同组合数学的生成函数。

循环指标是这样一个东西，它关注的是置换的骨架结构 —— 即各个循环的长度。

1.3.1 循环指标

定义 1.3.1 (循环指标)**.** 对 n 元置换 g，设 g 的循环表示为

$$g = (a_{11}a_{12}\cdots a_{1L_1})(a_{21}a_{22}\cdots a_{2L_2})\cdots\left(a_{y1}a_{y2}\cdots a_{yL_y}\right)$$

设这些循环中有 $\#_i$ 个循环大小为 i，则定义 g 的**循环指标**为 $t_1^{\#_1}t_2^{\#_2}\cdots t_n^{\#_n}$，其中 t_i 为形式变元，就像生成函数中的 x 一样。

它也有另一种理解方式：对于 g 的每一个循环 c_i，设它的长度为 L_i。则它会对“循环指标”贡献 t_{L_i}，最后将每个循环对“循环指标”的贡献相乘，即得最终的循环指标。

如：置换 $(1\ 4\ 2\ 5\ 3)$ 的循环指标为 t_5；而置换 $(1\ 4)(2\ 5)(3\ 6\ 7)$ 的循环指标为 $t_2^2t_3$。

2 群

2.1 定义

2.1.1 群

定义 2.1.1 (群)**.** 若一个非空集合 G 和其上的二元运算 $\circ$ 满足以下四个条件，则称二元组 $(G,\circ)$ 构成**群**，或称 G 在 $\circ$ 下构成**群**。在不混淆的情况下，也可称 G 是群。

1. *(封闭性)* $\forall f,g\in G, f\circ g\in G$。
2. *(结合律)* $\forall f,g,h\in G,(f\circ g)\circ h=f\circ(g\circ h)$。
3. *(单位元存在性)* $\exists g\in G$，使得 $\forall g\in G, e\circ g=g\circ e=g$。
4. *(逆存在性)* $\forall f\in G,\exists g\in G$，使得 $f\circ g=g\circ f=e$。

其中 e 叫做 g 的单位元 *(幺元)*，对满足 $f\circ g=g\circ f=e$ 的 g，称为 f 的逆元，记作 f^{-1}。

需要注意的是，群的定义中**并没有说明**群中元素的运算需要满足交换律。因为事实上，在这类问题中，结合律显得比交换律更基本些，更重要些。一个经典的例子是满足结合律的代数系统中可以定义幂 (Power)，而只满足交换律不满足结合律的代数系统中无法定义幂。

[2]Cycle index

定义 2.1.2 (交换群)**.** 当群 G 的运算满足交换律时，我们称 G 是一个交换群或阿贝尔群[3]。

2.1.2 一些特殊的群

1. 整数集合 $\mathbb{Z}$ 关于加法 $+$ 构成群 $(\mathbb{Z},+)$。
2. 对于任何正整数 m，在模 m 的意义下的加法也构成群。称为 m **阶循环群** (Cyclic group)，记作 Z_m。
3. 所有 $n!$ 个 n 元置换构成一个群，该群被称为 n **元对称群** (Symmetric group)，记作 S_n。
4. 所有 $\left\lceil \frac{n!}{2} \right\rceil$ 个偶置换构成一个群，该群被称为 n **元交错群** (Alternating group)，记作 A_n。
5. 对于一个正 n 边形，它的**旋转群**和 Z_n 是同构，故没必要取一个新的名称；而它的**旋转/翻转群**共有 $2n$ 个元素，被称为 $2n$ **阶二面体群** (Dihedral group)，记作 D_{2n}。

2.1.3 阶和子群

定义 2.1.3 (阶)**.** 群 G 的元素个数称为 G 的阶，简记为 $|G|$。

若 G 有无穷多个元素，称 G 为**无限群**，若 G 的元素个数有限，则称 G 是**有限群**。

定义 2.1.4 (子群)**.** 设 $(G,\circ)$ 是群，若 G 的子集 H 对于**同一种运算** $\circ$ 也构成群，则称 $(H,\circ)$ 是 $(G,\circ)$ 的子群。记作 $(H,\circ) \leq (G,\circ)$。

注意这里强调的是同一种运算，H 不能另辟一个新的运算。

2.1.4 生成子群

定义 2.1.5 (生成子群)**.** 设 $(G,\circ)$ 是群，S 为 G 的一个非空**子集**，则称包含 S 的所有子群的交称为 S 在 G 中生成的子群，记作 $\langle S\rangle$。

在这里，$\langle S\rangle$ 可以看作是 S 在 $\circ$ 运算下的闭包。如果 $\langle S\rangle = G$，则称 S 是 G 的一组生成集。

定义 2.1.6 (一个元素的阶)**.** 对于一个元素，我们同样可以定义它的阶 —— 对于元素 $g \in G$，如果存在 $n \in \mathbb{N}^*$，使得 $g^n = e$，则称满足 $g^n = e$ 的最小者为它的阶，记作 $|g| = n$。

若这样的 n 不存在，则称它是**无限阶的**。

[3] Abelian group

对于有限群，由抽屉原理和群的逆元素存在性可知，任何一个元素的阶都是有限的。

由阶的定义和生成子群的定义，容易验证：

对于任何群 G 和任何元素 $a \in G$，有 $|\langle a\rangle| = |a|$。即 a 在 G 中生成的子群大小等于 a 的阶数。

2.1.5 陪集，Lagrange 定理

定义 2.1.7 (陪集)**.** *对于群 G 和它的子群 $H \leq G$，对于一个元素 $g \in G$，记集合 $gH = \{g \circ h | h \in H\}$ 为 H 在 G 中导出的一个左陪集，同理可以定义右陪集。*

容易证明，对于确定的子群 H，它导出的所有陪集大小都是相等的，就等于 $|H|$。

陪集有一个比较好的性质：

定理 2.1.1. *对于子群 $H \leq G$，它导出的任意两个陪集，要么完全相同，要么交集为空。*

事实上，若存在 $a, b \in G$ 与 $h_1, h_2 \in H$ 满足 $a \circ h_1 = b \circ h_2$，则 $a = b \circ (h_2 \circ h_1^{-1}) \in bH$，即 $a \in bH \Rightarrow aH \subseteq bH$，同理 $bH \subseteq aH$，因此有 $aH = bH$。

从而，可以直接导出 Lagrange 定理：

定理 2.1.2 (Lagrange)**.** *对于有限群 G 及其子群 $H \leq G$，有 $|G| = |H| \cdot [G : H]$，其中 $[G : H]$ 表示 H 可以导出的陪集个数。*

2.2 置换群、Burnside 引理和 Pólya 计数定理

置换群是 OI 中最常见的一类群了，本节将介绍与置换群相关的基础理论以及 Pólya 计数定理。

定义 2.2.1 (置换群)**.** *由大小相同的置换作为元素构成的群称为置换群。如果置换的大小为 n，则称对应的群是一个 n 元置换群。*

2.2.1 染色

接下来引入群论中的一个重要概念 —— 染色。

为了方便，以下我们约定问题均在 n 元的情况下。即置换群中的置换大小均为 n。

定义 2.2.2 (染色)**.** *一个 n 元染色，指的是对集合 $\{1, 2, \cdots, n\}$ 的每个元素分配一个物品 (可以是颜色、数，等等) 的分配方案。*

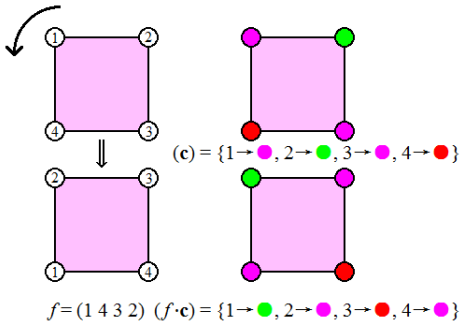
通常，用 $\mathbf{c}$ 表示一个染色，$\mathbf{c}[i]$ 表示该染色中 i 位置的物品。记所有染色的集合为 C。

2.2.2 置换的作用

置换是可以**作用**于染色，得到其它的染色的。

定义 2.2.3 (作用)**.** 对于置换 $f \in S_n$ 和染色 $\mathbf{c} \in C$，定义满足 $f(i)$ 的颜色是 $\mathbf{c}[i]$ 的染色 $\mathbf{c}'$，为 f **作用**于 $\mathbf{c}$ 的结果，记为 $f \cdot \mathbf{c}$，简记为 $f\,\mathbf{c}$。

即 $(f \cdot \mathbf{c})[i] = \mathbf{c}[f^{-1}(i)]$。



下面是一个具体的例子：

2.2.3 作用的性质，广义染色

不难验证，置换对染色的作用满足如下两个性质：

1. $e \cdot \mathbf{c} = \mathbf{c}$。
2. $(f \circ g) \cdot \mathbf{c} = f \cdot (g \cdot \mathbf{c})$。

而且，关于之后的所有结论和证明，也只需要用到这两个性质。因此，可以通过这两个性质，定义出抽象的染色概念：

定义 2.2.4 (广义染色)**.** 对于群 G *(无须是置换群)* 和一个全集 C，对于 G 中任意一个元素和 C 中任意一个元素 $\mathbf{c}$，定义运算 $\cdot$ 满足 $g \cdot \mathbf{c} \in C$，且满足如下两个性质：

1. $e \cdot \mathbf{c} = \mathbf{c}$。
2. $(f \circ g) \cdot \mathbf{c} = f \cdot (g \cdot \mathbf{c})$。

则称 C 是广义染色集合，C 中的元素 $\mathbf{c}$ 是**广义染色**。

2.2.4 轨道和稳定子群

定义 2.2.5 (轨道)**.** 对于一个置换群 G 和一个染色 $\mathbf{c}$，对于群中的所有元素，我们都对 $\mathbf{c}$ “作用”一下，可以得到一个 C 的子集，记作 $G \cdot \mathbf{c}$，即

$$G \cdot \mathbf{c} = \{g \cdot \mathbf{c} | g \in G\}$$

这个集合 $G \cdot \mathbf{c}$，被称作 $\mathbf{c}$ 在 G 中的**轨道**。

和轨道相对立的一个概念，称为**稳定子群**，它的定义如下：

定义 2.2.6 (稳定子群)**.** *对于一个置换群 G 和一个染色 $\mathbf{c}$，群中满足 $g\cdot\mathbf{c}=\mathbf{c}$ 的置换 g 构成一个群，称为染色 $\mathbf{c}$ 的稳定子群，记为 $G_{\mathbf{c}}$。*

此外，对一个染色集合 $X\subseteq C$，定义 $G\cdot X=\{g\cdot\mathbf{c}|g\in G,\mathbf{c}\in X\}$。若 $G\cdot X=X$，则称 X 在 G 下**固定**。

2.2.5 轨道 —— 稳定子群定理

有了这样的两个概念后，就可以得到群作用中奠基的定理：轨道 —— 稳定子群定理。

定理 2.2.1 (轨道 —— 稳定子群定理)**.** *对于置换群 G 和染色 $\mathbf{c}$，有 $|G\cdot\mathbf{c}|\cdot|G_{\mathbf{c}}|=|G|$。*

证明. *考虑置换群 G 以及对应的染色 $\mathbf{c}$，由定义，$G_{\mathbf{c}}$ 是一个子群.*

任取 $g\in G$，对于左陪集 $gG_{\mathbf{c}}=\{g\circ h|h\in G_{\mathbf{c}}\}$ 中的元素 $f=g\circ h_0$，有 $f\cdot\mathbf{c}=(g\circ h_0)\cdot\mathbf{c}=g\cdot(h_0\cdot\mathbf{c})=g\cdot\mathbf{c}$，因此左陪集 $gG_{\mathbf{c}}$ 中的所有置换作用于 $\mathbf{c}$ 产生相同的染色。

另一方面，对于两个不同的左陪集 $g_1G_{\mathbf{c}},g_2G_{\mathbf{c}}$，它们作用于 $\mathbf{c}$ 不能产生相同的染色。否则 $g_1\cdot\mathbf{c}=g_2\cdot\mathbf{c}$，有 $(g_1^{-1}\circ g_2)\cdot\mathbf{c}=g_1^{-1}\cdot(g_2\cdot\mathbf{c})=g_1^{-1}\cdot(g_1\cdot\mathbf{c})=\mathbf{c}$，从而由定义，$g_1^{-1}\circ g_2\in G_{\mathbf{c}}$，于是 $g_2\in g_1G_{\mathbf{c}}$，矛盾。

所以，设子群 $G_{\mathbf{c}}$ 导出的陪集数量为 K，每个陪集作用于 $\mathbf{c}$ 可以得到一个独一无二的染色，因此 K 就等于整个置换群中所有元素作用于 $\mathbf{c}$ 所得到的染色数量，即 $|G\cdot\mathbf{c}|$。

由 Lagrange 定理，$|G|=|G_{\mathbf{c}}|\cdot[G:G_{\mathbf{c}}]=|G\cdot\mathbf{c}|\cdot|G_{\mathbf{c}}|$。

2.2.6 Burnside 引理

轨道 —— 稳定子群定理的一个直接推论就是 **Burnside 引理**。

为了方便阐述这个定理，首先需要一些定义：

定义 2.2.7 (置换的不动点)**.** *对于一个置换 g 和一个染色集合 $X\subseteq C$，X 中满足 $g\cdot\mathbf{c}=\mathbf{c}$ 的染色 $\mathbf{c}$ 的集合记为 X^g。*

定义 2.2.8 (染色的等价)**.** *对于置换群 G 和两个染色 $\mathbf{c}_1,\mathbf{c}_2$，称两个染色**等价** (或**本质相同**) 当且仅当 $\exists g\in G$，使得 $g\cdot\mathbf{c}_1=\mathbf{c}_2$，记作 $\mathbf{c}_1\sim\mathbf{c}_2$。*

不难证明，两个染色 c_1,c_2 等价，当且仅当下列条件之一成立：

1. $\mathbf{c}_2\sim\mathbf{c}_1$。
2. $\exists g\in G$，使得 $g\cdot\mathbf{c}_1=\mathbf{c}_2$。
3. $\mathbf{c}_2$ 在“$\mathbf{c}_1$ 在 G 中的轨道”中。

4. $\mathbf{c}_1$ 在 G 中的轨道与 $\mathbf{c}_2$ 在 G 中的轨道相同。

这说明我们可以将 X 中不等价的染色数量看成 X 中元素在 G 中形成的不同轨道数目。我们用 X/G 表示 X 中元素形成的互异轨道的集合，$|X/G|$ 表示不同轨道数目。则有：

定理 2.2.2 (Burnside)**.** 对于置换群 G 和它固定的染色集合 X，有

$$|G|\,|X/G| = \sum_{g\in G} |X^g|$$

即在 G 的作用下，X 中元素形成的不同轨道数目，等于 G 中所有置换的不动点个数的平均值。

证明. 考虑计算集合 $\{(g,\mathbf{c})\big|g\cdot\mathbf{c}=\mathbf{c}, g\in G, \mathbf{c}\in X\}$ 的大小。

一方面，我们枚举每个置换，它就等于每个置换的不动点个数的和，即 $\sum_{g\in G}|X^g|$。

另一方面，枚举每个染色，它就等于该染色的稳定子群大小 $\sum_{\mathbf{c}\in X}|G_{\mathbf{c}}|$。

由轨道 —— 稳定子群定理，

$$\sum_{\mathbf{c}\in X}|G_{\mathbf{c}}| = \sum_{\mathbf{c}\in X}\frac{|G|}{|G\cdot\mathbf{c}|} = |G|\cdot\sum_{\mathbf{c}\in X}\frac{1}{|G\cdot\mathbf{c}|}$$

对于等式右端，考虑每一个完整的轨道 $G\cdot\mathbf{c}$，其中每个染色都会产生 $\dfrac{1}{|G\cdot\mathbf{c}|}$ 的贡献，因此每个轨道恰对右端的和式贡献 1，于是 $\sum\limits_{\mathbf{c}\in X}\dfrac{1}{|G\cdot\mathbf{c}|} = |X/G|$，*Burnside* 引理成立。

2.2.7 Pólya 计数定理 —— 简单版

注意到我们在推导 Burnside 引理的整个过程中，对于置换的“作用”，都只用到了两个性质 (单位性和结合性)，因此 Burnside 引理其实是对一般的群和广义染色成立的。

而当我们限制群为**置换群**，染色为**一般的染色**时，那么或许可以到更进一步的结果。

在 Burnside 引理中，考虑置换 $g\in G$。我们要计数在置换 g 下，X 中使得在 g 作用下不变的染色数量。

由之前置换的理论，设 g 的循环表示是 $g = c_1\circ c_2\circ\cdots\circ c_y$，则考虑一个循环 $c = (a_1\ a_2\cdots)$，将这个循环作用于染色 $\mathbf{c}$，由定义，有 $\mathbf{c}[a_i] = \mathbf{c}[a_{i+1}]$。因此，我们可以得到：

引理 2.2.1. 对于置换 g 和染色 $\mathbf{c}$，设 g 的循环表示是 $g = c_1\circ c_2\circ\cdots\circ c_y$，则 $g\cdot\mathbf{c}=\mathbf{c}$ 的充要条件是：对于 g 的每个循环 $(a_1\ a_2\cdots)$，$\mathbf{c}$ 中对 $a_1, a_2, \cdots$ 分配的颜色是相同的。

于是，假设颜色一共有 m 种，且每个位置可分配的颜色集合都是相同的，那么，如果 g 有 $\#(g)$ 个循环，那么，在 g 的作用下不变的染色数量就应该是 $m^{\#(g)}$，从而可以得到

定理 2.2.3 (Pólya)**.** 对于置换群 G 和它固定的染色集合 X，如果这 n 个位置可分配的颜色集合都是相同的，一共 m 种，那么对于 $g \in G$，有

$$|X^g| = m^{\#(g)}$$

从而代入 *Burnside* 引理，有

$$|G|\,|X/G| = \sum_{g \in G} m^{\#(g)}$$

其中 n 为置换的大小，$\#(g)$ 表示置换 g 的**循环数**。

2.3 Pólya 定理的完整版及其扩展

在上文我们定义了一个置换的循环指标，它可以较为方便地描述**生成函数版的 Pólya 定理**。

定义 2.3.1 (置换群的循环指标)**.** 定义一个置换群 G 的循环指标，为**群中所有置换的循环指标的平均值**，记作 $Z_G(t_1, t_2, \cdots, t_n)$。

定理 2.3.1 (Pólya)**.** 假设普通生成函数 $f(t) = f_0 + f_1 t + f_2 t^2 + \cdots$，其中 f_w 为权值为 w 的**颜色**数量。

定义一个染色的权值为 n 个位置所分配的颜色的权值之和。

用生成函数 $F(t)$ 表示在 G 的作用下不同轨道数的普通生成函数，则 *Pólya* 定理表明：

将 $t_i = f(t^i)$ 代入 G 的循环指标中，所得到的结果就是 $F(t)$，即：

$$F(t) = Z_G\left(f(t), f\left(t^2\right), \cdots, f\left(t^n\right)\right)$$

此定理可以推广到多元生成函数的情形中。特别地，当 $f(t)$ 为常数时该定理就是之前所述的简单版 *Pólya* 定理。

2.3.1 广义 Burnside 引理/Pólya 定理

接下来考虑对一般的 Burnside 引理进行推广。

在一般的 Burnside 引理中，我们是对下式进行算两次：

$$\sum_{g \in G} \sum_{\mathbf{c} \in X} [g \cdot \mathbf{c} = \mathbf{c}]$$

现在考虑对每个置换 g 赋予一个权值 $\omega(g)$，那么，对于一个子群 $H \le G$，定义它的权值 $\omega(H) = \sum\limits_{g \in H} \omega(g)$，然后对

$$\sum_{g \in G} \sum_{\mathbf{c} \in X} [g \cdot \mathbf{c} = \mathbf{c}]\, \omega(g)$$

算两次，可以得到：

定理 2.3.2 (广义 Burnside 引理)**.** 对于置换群 G 和它固定的染色集合 X，记群中的置换 g 的权值为 $\omega(g)$，子群 H 的权值为 $\omega(H)$，则：

$$\sum_{O\in X/G}\omega(G_O)|O|=\sum_{g\in G}\omega(g)|X^g|$$

即在 G 的作用下，X 中元素形成的所有轨道的大小与对应稳定子群的权值的乘积之和，等于 G 中所有置换的不动点个数与对应置换权值乘积之和。

同理，当 $\omega(g)$ 仅和置换的循环指标相关时，就能导出广义 Pólya 定理。

广义 Burnside 定理一般有两个用途：其一是通过合理给置换赋权，来解决带权轨道的计数问题，其二是推出接下来所要介绍的 **Pólya 容斥**。

2.3.2 Pólya 容斥

特别地，在广义 Burnside 定理的式子中，取 $\omega(g)=\operatorname{sgn}(g)$ (即符号，奇置换为 -1，偶置换为 1)。

考虑染色集合在 S_n 的作用下形成的不同轨道，可知一种染色 $\mathbf{c}$ 的稳定子群 $G_{\mathbf{c}}$ 同构于若干个对称群的直积。

而这些小的对称群中，一旦有 ≥ 2 元的对称群，那么其中所有置换的符号之和等于 0，从而稳定子群 $G_{\mathbf{c}}$ 的权值 $\omega(G_{\mathbf{c}})=0$。

也就是说，最终一个轨道的权值非零，当且仅当它的稳定子群是平凡群，也就是说 n 个位置的“颜色”互不相同，这就是 Pólya 容斥。

推论 2.3.1 (Pólya 容斥)**.** 对于置换群 $G=S_n$ 和它固定的染色集合 X，有

$$|G|\sum_{O\in X/G}\left[O\text{ 是颜色互异的轨道}\right]=\sum_{g\in G}\operatorname{sgn}(g)|X^g|$$

即在 G 的作用下，X 中元素形成的各颜色互不相同的轨道数，等于 G 中所有置换的不动点个数乘以其符号的平均值。

2.4 例题

例题 1. 树 [4]

定义 n 阶带标号有根树的集合 $\mathcal{T}$，满足以 1 为根，i 的父节点标号 p_i 满足 $1\leq p_i<i$。易知 $|\mathcal{T}|=(n-1)!$。

现在按顺序等概率随机取 k 个 $\mathcal{T}$ 中的元素 $T_1,T_2,\cdots,T_k$ *(可以相同)*，求 $T_1,T_2,\cdots,T_k$ *(作为有根树)* 两两同构的概率。

$n\leq 2000; k\leq 10^9$，对大素数取模。

[4]来源：ZJOI2018，有改动

先转化题意，显然两棵树同构是一个等价关系，因此我们可以将 $\mathcal{T}$ 划分为若干个等价类 $\mathcal{T}=E_1\cup E_2\cup\cdots\cup E_\lambda$，于是我们就要求 $\sum\limits_{i=1}^{\lambda}|E_i|^k$。

考虑 DP —— 记 f_i 表示 i 个点时的答案，那么它也就等于将其去掉后森林的答案，因此转而考虑有根森林的等价类。森林中树的大小参差不齐，故按照大小为 $1,2,\cdots,n$ 的顺序依次在森林中加入对应大小的树。记 $g_{i,s}$ 表示所有树大小不超过 i 的大小为 s 的森林所对应的答案 (各等价类大小的 k 次方和)，则有 $f_i=g_{i-1,i-1},g_{1,s}=1$。考虑转移，那么就是在森林中加入若干个大小为 i 的树。枚举加入了 d 个，那么有

$$g_{i,s}=\sum_{0\le d\le\lfloor\frac{s}{i}\rfloor}g_{i-1,s-d\cdot i}\cdot I_{i,d}\cdot\binom{s}{d\cdot i}^k$$

其中 $I_{i,d}$ 表示对于所有 d 棵大小为 i 的树构成的带标号有根森林，各等价类大小的 k 次方和。

下面考虑求解 $I_{i,d}$，我们换一个字母，用 $I_{n,m}$ 表示，即 m 棵大小为 n 的树。

首先，对于带标号的问题，外面的标号已经由一个二项式系数解决，实际 DP 时可以统统除以阶乘然后直接乘，因此里面的标号设为 $1\sim n$ 也无妨。因此，在一般情况下，这个标号分配方案就等于 $\binom{m\cdot n}{n,n,\cdots,n}$，但是这里不行，因为同构的两棵树之间换一下标号，得到的还是同一棵树。

考虑一个 m 维的，元素为 n 阶树等价类的向量的全集 X，两个向量“本质相同”当且仅当 n 阶树构成的等价类集合相同。设 n 阶树的等价类划分为 $\mathcal{T}_n=E_1\cup E_2\cup\cdots\cup E_\lambda$，然后记 $P_n(\delta)=\sum\limits_{i=1}^{\lambda}|E_i|^\delta$。那么，可以发现 X 中每个元素其实就是一个**染色** —— 对于每个位置分配一个 E_i。于是问题就转化为了等价类计数的问题，考虑使用 Pólya 定理来处理。

使用多元生成函数的 Pólya 定理，定义 $f(t_1,t_2,\cdots,t_\lambda)=t_1^k+t_2^k+\cdots+t_\lambda^k$。则最后我们要求的即为 $F(|E_1|,|E_2|,\cdots,|E_\lambda|)$。在 X 上作用的变换群显然是 S_m，因此我们将其代入 S_m 的循环指标，就得到一个关于 $f(|E_1|,|E_2|,\cdots,|E_\lambda|),f\left(|E_1|^2,|E_2|^2,\cdots,|E_\lambda|^2\right),\cdots,f(|E_1|^m,|E_2|^m,\cdots,|E_\lambda|^m)$，亦即 $P_n(k),P_n(2k),\cdots,P_n(mk)$ 的表达式。

假设我们已经知道了这些 $P_n(ik)$，那么考虑 Pólya 定理，其实质就是对于一个大小为 c 的循环给出 $P_n(ck)$ 的贡献。而熟知置换可以由循环之间的**带标号无序组**来刻画，也就是多项式 exp。

但是这里有个致命的问题 —— 每个轨道的贡献不一定是 1：具体地，考虑一个轨道 (“本质相同”的向量组)，设其中包含了 κ_i 个 E_i 中的元素，那么在**最终分配标号**的时候，这些树它们之间的标号是可以任意互换的，因此最后还需要除以 $\kappa_i!$。

也就是说，对于一个轨道，它会产生 $\prod\limits_{i=1}^{\lambda}\frac{1}{(\kappa_i!)^k}$ 倍的贡献。

可以发现这里我们需要统计带权轨道的权值和，因此我们需要使用**广义 Pólya 定理**：

$$\sum_{O\in X/G}\omega(G_O)|O|=\sum_{g\in G}\omega(g)|X^g|$$

我们希望对于满足 $(\kappa_1,\kappa_2,\cdots,\kappa_\lambda)$ 的轨道 O，有 $\omega(G_O)|O|=\prod\limits_{i=1}^{\lambda}\frac{1}{(\kappa_i!)^k}$。

考察 G_O 的结构，可得 $G_O = S_{\kappa_1} \times S_{\kappa_2} \times \cdots \times S_{\kappa_\lambda}$。

注意到在 Pólya 定理中，一个置换的权值仅仅和它的循环指标相关，以及两个群的直积的循环指标等于两个群的循环指标的乘积 (卷积)。因此有 $Z_{G_O} = Z_{S_{\kappa_1}} \cdot Z_{S_{\kappa_2}} \cdot \cdots \cdot Z_{S_{\kappa_\lambda}}$，从而有 $\omega(G_O) = \omega(S_{\kappa_1})\,\omega(S_{\kappa_2}) \cdots \omega(S_{\kappa_\lambda})$。

结合轨道 —— 稳定子群定理，得 $|O| = \dfrac{|G|}{|G_O|} = \dfrac{m!}{\kappa_1!\kappa_2! \cdots \kappa_\lambda!}$，而 $m!$ 可以看成常数，因此对比前式可知要

$$\omega(G_O) = \prod_{i=1}^{\lambda} \frac{1}{(\kappa_i!)^{k-1}} \Rightarrow \omega(S_\kappa) = \frac{1}{(\kappa!)^{k-1}}$$

而现在我们需要知道每个循环大小所产生的贡献，记大小为 c 的循环产生的贡献为 χ_c，于是通过对称群 S_κ 的权值我们可以通过 $\chi_1, \chi_2, \cdots$ 来推导出 χ_κ。其实，这里的置换还是可以由循环之间的带标号无序组来刻画，因此之前的多项式 exp 仍是可行的。

设 $f(x) = \sum\limits_{i \geq 1} \chi_i \dfrac{x^i}{i!}$，则 $\exp f(x) = 1 + \sum\limits_{i \geq 1} \omega(S_i) \dfrac{x^i}{i!} = \sum\limits_{i \geq 0} \dfrac{x^i}{(i!)^k}$，于是做一次多项式 ln 就能得到诸 χ_i 了。

当然，这里还剩最后一个问题：当时是假设已经知道 $P_n(k), P_n(2k), \cdots, P_n(mk)$，但事实上除了 $P_n(k)$，其余的值都是不知道的。这说明我们不仅仅要 DP 所有等价类的 k 次方和，还有 $2k$ 次方和，$3k$ 次方和……

不过注意到 $d \cdot i \leq n$，也就是说，对于 i 阶树的等价类，我们只需要知道其 k 次方和，$2k$ 次方和，……，$\left\lfloor \dfrac{n}{i} \right\rfloor \cdot k$ 次方和，这是一个调和级数。于是我们对多组 $(i, d \cdot k)$ $(i \cdot d \leq n)$ 分别计算即可，考虑平方实现的 exp/ln，则这部分的时间复杂度为

$$\sum_{i \cdot d \leq n} \left(\frac{n}{i \cdot d} \right)^2 = O\left(n^2\right)$$

其余部分转移的时间复杂度为

$$\sum_{i=1}^{n} \sum_{e=1}^{\lfloor \frac{n}{i} \rfloor} \sum_{s=1}^{\lfloor \frac{n}{e} \rfloor} \left\lfloor \frac{s}{i} \right\rfloor = O\left(n^2\right)$$

即总时间复杂度 $O(n^2)$。

3 群的判定和表示

3.1 基础知识

3.1.1 不变子群、商群

定义 3.1.1 (不变子群)**.** *设群 $(G, \circ)$ 的子群 $H \leq G$ 满足：对于是 $\forall g \in G, h \in H$，有*

$$g \circ h \circ g^{-1} \in H$$

则称 H 是 G 的**不变子群**或**正规子群**，记作 $H \trianglelefteq G$。

若 $H \trianglelefteq G$ 且 $H \neq G$，则记 $H \triangleleft G$ *(真不变子群)*。

定义 3.1.2 (商群)**.** 对于群 $(G, \circ)$ 和它的不变子群 $N \trianglelefteq G$，在 N 的所有陪集 *(左右都一样)* G/N 上定义运算 $\cdot$ 满足:

$$(aN) \cdot (bN) = (a \circ b) N$$

则称 $(G/N, \cdot)$ 为 G 对 N 的**商群**。

3.1.2 同态和核

定义 3.1.3 (同态)**.** 设有群 $(G, \circ), (H, \cdot)$，若映射 $f : G \to H$ 满足，对于 $\forall a, b \in G$ 均有

$$f(a \circ b) = f(a) \cdot f(b)$$

则称 f 是 G 到 H 的**同态**映射，简称同态。

根据 f 是否是单射、满射、双射，可以定义同态映射是否是单同态、满同态和同构。

定义 3.1.4 (核)**.** 设 f 是 G 到 H 的同态，e_H 为 H 的单位元，则集合

$$f^{-1}(e_H) = \{g \mid g \in G, f(g) = e_H\}$$

被称为 f 的**核**，记为 $\ker f$。

3.1.3 同构定理

同态和同构有着密切的联系，比如下面的群同态基本定理 (群同构第一定理) 和群同构第三定理：

定理 3.1.1 (群同态基本定理)**.** 设 f 是 $(G, \circ)$ 到 $(H, \cdot)$ 的**满同态**，那么 $G/\ker f$ 和 H 同构。

这是群同态中奠基的一个定理，在同态与熟悉的同构之间搭建了一座桥梁。

证明. 设 $K = \ker f$。定义映射 $\phi : G/K \to H$，满足:

$$\phi(gK) = f(g)$$

首先需要证明这个定义的合理性，即它没有歧义。事实上，设 $g \circ h_1, g \circ h_2 \in gK$，则

$$f(g \circ h_1) = f(g) \cdot f(h_1) = f(g) \cdot f(h_2) = f(g \circ h_2)$$

因此这个定理是合理的。

现在我们欲证明 ϕ 是同构 *(双同态)*，因此我们就需要分别证明 ϕ 是同态、单射和满射。

- 取 $gK, hK \in G/\ker f$，有

$$\phi((g \circ h)K) = f(g \circ h) = f(g) \cdot f(h) = \phi(gK) \cdot \phi(hK)$$

即 ϕ 是同态。

- 若 $\phi(gK) = \phi(hK)$，则 $f(g) = f(h) \Rightarrow$

$$f\left(g \circ h^{-1}\right) = f(g) \cdot f\left(h^{-1}\right) = f(g) \cdot f(h)^{-1} = e_G$$

即 $g \circ h^{-1} \in K \Rightarrow gK = hK$，即 ϕ 是单射。

- 任取 $h \in H$，由于 f 是满射，故存在 $g \in G$ 使得 $f(g) = h$，于是 $\phi(gK) = f(g) = h$，从而 ϕ 是满射。

综上，ϕ 是双同态，即 G/K 和 H 同构。

下面的群同构第三定理，可以用来简化一些代码的实现，限于篇幅，这里只给出定理，略去证明。

定理 3.1.2 (群同构第三定理)**.** 设 N 是 G 的不变子群，则:

- G 的子群 H 满足 $N \leq H \leq G$，当且仅当 $H/N \leq G/N$。
- G 的子群 H 满足 $N \leq H \trianglelefteq G$，当且仅当 $H/N \trianglelefteq G/N$，如果两者成立，则商群 $\dfrac{G/N}{H/N} \cong G/H$。

3.2 群的判定

3.2.1 根据定义判定群

考虑一个经典的问题，就是给定一张乘法表，如何检验其中的元素是否构成一个群？

尝试通过群的定义 —— 封闭性、结合律、单位元和逆元来检验。为了方便起见，以下约定该代数结构中的运算用 $\circ$ 表示。

- 封闭性。

  直接检验即可。

- 单位元。

  设 e 是单位元，则 $e \circ e = e$。同时，若 g 满足 $g \circ g = g$，则两边同乘 g^{-1} 得 $g = e$。也就是说，e 是满足 $g \circ g = g$ 的唯一元素。

  通过这一点，我们可以得到群的单位元，设为 e (如果不存在或不唯一说明不是群)。接下来根据定义对每个 g 检验是否有 $e \circ g = g \circ e = e$。

- 逆元。

  对于 $\forall g \in G$，我们寻找满足 $g \circ h = h \circ g = e$ 的元素 h，如果不存在或不唯一说明不是群。否则通过检验。

接下来就是最后一步 —— 结合律的检验。

如果直接按照定义检验，我们需要枚举元素 f, g, h，而这样做的时间复杂度是 $O(n^3)$。

而前面三种性质的检验都可以在输入复杂度 ($O(n^2)$) 内完成，那结合律的检验是否有更优秀的方法呢？

对于一个一般的代数结构结合律的检验，到笔者写本文时，学术界尚未有复杂度低于 $O(n^3)$ 的**确定性**算法 (但存在复杂度较为优秀的随机算法)。

不过，如果我们检验的对象是群，则可以利用群的性质，可以得到一个在 $O(n^2 \log n)$ 时间内的算法。

3.2.2 检验结合律的 Light 算法

引理 3.2.1. *对于任意 n 阶有限群 G，存在一个大小不超过 $\lfloor \log_2 n \rfloor$ 的子集 $S \subseteq G$，它生成 G (即 $G = \langle S \rangle$)。*[5]

证明. 定义子群链 $\{e\} = G_0 \leq G_1 \leq \cdots \leq G_k = G$，其中 $G_i = \langle \{g_1, g_2, \cdots, g_i\} \rangle$，具体构造方法如下:

- 设我们已经知道 $G_0, G_1, \cdots, G_{i-1}$，现在要确定 G_i。
- 若 $G_{i-1} = G$，则构造结束。否则，有 $G_{i-1} < G$。
- 任取 $g_i \in G \setminus G_{i-1}$，令 $G_i = \langle G_{i-1} \cup \{g_i\} \rangle$。
- 则 $G_{i-1} \leq G_i \leq G$ 且 $G_{i-1} \neq G_i$ ($g_i \in G_i \wedge g_i \notin G_{i-1}$)。
- 于是 $|G_i| \geq 2|G_{i-1}|$。
- 因此 $n = |G| = |G_k| \geq 2^k |G_0| = 2^k$，即 $k \leq \lfloor \log_2 n \rfloor$，证毕。

引理 3.2.2. *设 $(G, \circ)$ 是一个满足封闭性、单位元、逆元的代数结构，设 $G = \langle S \rangle$，则 G 满足结合律当且仅当:*

- *对 $\forall s \in S, g, h \in G, (g \circ s) \circ h = g \circ (s \circ h)$。*

证明. 必要性显然。下证充分性:

设 $A = \{s \mid \forall g, h \in G, (g \circ s) \circ h = g \circ (s \circ h)\}$，即所有满足结合律的**中间元素**。

我们证明：若 $a, b \in A$，则 $a \circ b \in A$。

事实上，有

[5] 下界可以取到，如 $G = Z_2^k$

$$(g\circ(a\circ b))\circ h=((g\circ a)\circ b)\circ h=(g\circ a)\circ(b\circ h)$$
$$=g\circ(a\circ(b\circ h))=g\circ((a\circ b)\circ h)$$

其中 g,h 为任意元素，四个等号分别运用了 a,b,a,b 作为中间元素的结合律。

故 $a\circ b\in A$。由条件知 $S\subseteq A$，由上述结论并结合生成子群的性质知 $\langle S\rangle\subseteq A$，即 $G\subseteq A\Rightarrow G=A$，即代数结构 G 满足结合律。

结合上述两个引理，我们就得到了 Light 算法，流程如下：

1. 按照 Lemma 2.1 所述方法找到大小不超过 $\lfloor\log_2 n\rfloor$ 的集合 S，满足 $G=\langle S\rangle$。[6]
2. 对于 S 中的每个元素 s，检验 s 作为中间元素时是否满足结合律，即是否对于 $\forall g,h\in G$，有 $(g\circ s)\circ h=g\circ(s\circ h)$。

   如果成立，则 G 是群，否则 G 不是群。

分析一下算法的时间复杂度：对于第一步，容易在 $O(n^2)$ 或 $O(n^2\log n)$ 时间内找到一组生成集 S；而对于第二步，检验时间等于 $O(n^2|S|)=O(n^2\log n)$。

故总时间复杂度为 $O(n^2\log n)$。

3.3 群的表示

那么，对于一个一般的群，除了使用乘法表外，还有哪些方法能表示它呢？

在 OI 中比较常见的群就是置换群，因此我们希望用置换群来表示一个一般群。

定义映射 $\lambda_g(x)=g\circ x$，则 λ_g 是一个双射。

考虑两个映射 λ_g,λ_h 的复合，有

$$\lambda_g(\lambda_h(x))=g\circ(h\circ x)=(g\circ h)\circ x=\lambda_{g\circ h}(x)$$

同理，可以证明 λ_g 和 $\lambda_{g^{-1}}$ 互为逆映射。

而且，变换 λ_g 将 G 中的所有元素变换为了 G 中的所有元素，只是其中的对应关系发生了改变，即 λ_g 实质上可以看成是 G 上的一个**置换**。而置换 $\left\{\lambda_g\middle|g\in G\right\}$ 就构成了一个置换群，即 $|G|$ 元对称群的子群。

于是，我们得到了 **Cayley 定理**：

定理 3.3.1 (Cayley). *每个 n 阶有限群都同构于一个不超过 n 元的置换群 (不超过 n 元的对称群的子群)*。

换句话说，对于 n 阶有限群 G，至少存在一个 G 到 S_n 的**单同态**。

[6]因为我们假设 G 是群，因此这个过程一定可以进行。但是如果 G 不是群，这个过程也可能成功进行。但是这个过程一旦不能成功进行，就能说明 G 已经不是群了，那么后面也没必要再去检验结合律了。

3.4 例题

例题 2. 列队[7]

给定群 G，求 G 到 n 元对称群 S_n 的单同态个数。

$|G| \le 30; n \le 1000$，对 998244353 取模。

对于群 G, H，记 G 到 H 的**同态**数量为 $\text{homo}(G,H)$[8]，**单同态**数量为 $\text{mono}(G,H)$[9]。

考虑一个同态 $f: G \to H$，记 $K = \ker f$，由群同态基本定理知 G/K 和 $\text{im}\, f$ 之间存在同构 ϕ，那么将同构 ϕ 的陪域扩展到 H 即得 G/K 到 H 的一个**单同态**。也就是说，$G \to H$ 的每一个同态都对应到 G/N 到 H 的一个单同态，其中 N 是 H 的一个不变子群。于是，有

$$\text{homo}(G,H) = \sum_{N \trianglelefteq G} \text{mono}(G/N, H)$$

根据上式，我们就可以将计算 $\text{mono}(G,H)$ 的问题通过类似于反应的手段转化为了若干个计算 $\text{homo}(G,H)$ 的子问题。

现在考虑给定群 G，计算它到 S_n 的**同态**个数。

设 f 是 G 到 S_n 的一个同态，设置换群 $H = \text{im}\, f \le S_n$。定义 i 的特征染色 χ_i 为：i 位置为黑色，其余位置为白色。那么诸轨道 $H \cdot \chi_i$ 中黑色出现的所有位置构成的集合，构成了集合 $\{1, 2, \cdots, n\}$ 的一个划分。

考虑其中一个集合 A ($|A| = k$)，不妨设 $1 \in A$，则 $|H \cdot \chi_1| = k$。由轨道 —— 稳定子群定理，$\left|H_{\chi_1}\right| = \dfrac{|H|}{|H \cdot \chi_1|} = \dfrac{|H|}{k}$。由同态的性质知，稳定子群 H_{χ_1} 的原像是 G 的一个 $\dfrac{|G|}{k}$ 阶子群。

之前讨论的是给定 f 后 G 的结构，接下来尝试从 G 的结构去构造 f。

对于 $\{1, 2, \cdots, n\}$ 的任意一个划分，作为诸元素的轨道。考虑其中一个集合 A ($|A| = k$)，仍然不妨设 $1 \in A$。在 G 中任意寻找一个大小为 $\dfrac{|G|}{k}$ 的子群 S，令它的像为特征染色 χ_1 的稳定子群。那么 S 导出的 k 个左陪集，作用于 χ_1 后将黑色分别移到 $1, 2, \cdots, k$。

记这 k 个左陪集分别为 $S, g_2S, g_3S, \cdots, g_kS$，由于单位元在 S 中，因此陪集 S 中元素的像会将黑色移到 1。对于剩下的 $2 \le i \le n$，合理调整 g_i 的顺序，不妨设陪集 g_iS 中元素的像会将黑色移到 i。

于是，对于这样一种 g_i 的顺序，考虑其中任意一个置换 g，我们有 $(\forall s \in S)$

$$g(j) = g((g_j \circ s)(1)) = (g \circ g_j \circ s)(1) = (g \circ g_j)(1)$$

即 $g(j)$ 由 $g \circ g_j$ 唯一确定，和 s 无关，于是这个定义没有歧义 (合理)，因而也就得到一个所有元素在 A 中唯一的变换。但是我们还能调整 g_i 的顺序，这里一共有 $(k-1)!$ 种调整的方式，每种方式都能对应到一个 A_1 上独一无二的变换。综上，对于一个大小为 k 的集

[7] 来源：UOJ Round #10，Problem C (uoj154)，有改动

[8] homomorphism

[9] monomorphism

合，我们需要一个大小为 $\frac{|G|}{k}$ 的子群作为其稳定子群的原像。且对于每个这样的子群，都能得到 $(k-1)!$ 种该等价类中变换的方式。

接下来就可以直接计数了，只需要作出 $\{1,2,\cdots,n\}$ 的一个划分，然后对划分中的每个集合找到一个对应大小的子群与之对应即可。

由于划分可以看成**带标号无序组**，因此设

$$f(x)=\sum_k \frac{x^k}{k}\sum_{H\le G,|H|=\frac{|G|}{k}} 1=\sum_{H\le G}\frac{x^{|G|/|H|}}{|G|/|H|}$$

则 $\text{homo}(G,S_n)=n![x^n]\exp f(x)$。

此外，对于这题的实现，其实是不需要递归的求解的，我们可以通过**群同构第三定理**来简化过程。考虑我们递归解决规模为 G/N 的子问题，我们需要枚举 G/N 的子群和不变子群。由群同构第三定理，G/N 的子群和不变子群对应于 G 的子群和不变子群 (中满足 N 是其子群者)，如果继续递归，所得到的商群 $\frac{G/N}{H/N}$ 其实是同构于 G/H 的。因此在整个过程中所涉及到的群，其实都是 G 的商群。

也就是说，我们只需要一次 bfs 得到 G 的所有子群和不变子群，然后按照阶数从大到小的顺序枚举不变子群 N，解决规模为 G/N 的问题。于是扫描到不变子群 N 时，这些商群的子群所对应的答案都是已知的，像 Möbius 反演一样操作即可。

如果使用 $O(n^2)$ 的多项式 exp，则总时间复杂度为 $O\left(M|G|^2+M_N\cdot M+M_N\cdot n^2\right)$ (M (M_N) 分别表示 30 阶以内的群的 (不变) 子群数量的最大值，其值等于 67，在 Z_2^4 处取到)。

4 计算群论初步

下面介绍的内容是一些计算群论的基础算法。

计算群论是研究某一类问题的利器：对于一些大小不大的置换构成的集合 S，它们可能生成一个很大的置换群 $\langle S\rangle$，而计算群论可以对形如 $\langle S\rangle$ 的置换群维护出一个有很多功能的“群论结构”。其中 Schreier-Sims 算法是计算群论中最基础的算法。

4.1 引入

考虑一个最基本的问题：给定若干个 n 元置换构成的集合 S，求 S 生成的子群大小 $|\langle S\rangle|$，这里 $n\le 50$。

怎么求一个巨大的群的大小呢？一个比较直观的思路是：

如果我们有个子群链 $\langle S\rangle=G_0\ge G_1\ge G_2\ge\cdots\ge G_k=\{e\}$，那么由 Lagrange 定理，有

$$|G|=\prod_{i=0}^{k-1}\frac{|G_i|}{|G_{i+1}|}=\prod_{i=0}^{k-1}[G_i:G_{i+1}]$$

于是我们就尝试去构造这样一个子群链。

根据之前的经验，对于一个置换群 G，对 $\forall 1 \le i \le n$，诸轨道 $G \cdot \chi_i$ 构成了 $\{1, 2, \cdots, n\}$ 的一个划分。

考虑特征染色 χ_1 的稳定子群 G_{χ_1}，由轨道 —— 稳定子群定理，有 $[G : G_{\chi_1}] = |G \cdot \chi_1|$，而轨道的大小显然不超过 n，因此这个数值是可接受的。

而且，这个稳定子群它**固定了元素** 1，也就是说它可以被嵌入到 $n-1$ 元置换群中，这就是原问题的一个子问题。

如果我们能对其进行递归求解，那就得到了我们所要的子群链了。

这就是 Schreier-Sims 算法的主要思想。

4.2 元素判定和截面

知道了算法的思想后，接下来考虑如何对这样的群进行操作。

由于 G 是置换群，那么可以比较容易知道 $G \cdot \chi_1$ 的大小，以及这些元素的轨道。但是现在我们无法方便地表示 G_{χ_1}。事实上，G_{χ_1} 也是一个庞大的置换群，也只能用生成集来表示。

因此，Schreier 和 Sims 选择了**增量构造法**，即逐渐向 S 中添加元素，然后对子群链中的每个子群进行维护。

初始时，$S = \varnothing$，那么 $G = \{e\}$，这些都是平凡的。

考虑向 S 中添加一个新元素 g。首先，我们需要检验 g 是否已经在 $\langle S \rangle$ 中。

也就是说，我们维护的群论结构需要支持查询一个置换是否在 $\langle S \rangle$ 中。

我们仍然考虑递归求解，那么此时不能显然只存储轨道划分了，我们需要存储一下 G_{χ_1} 导出的陪集的有关信息。

为了统一起见，本文接下来一律使用右陪集。

其实，虽然 G_{χ_1} 很大，但它导出的陪集并不多，我们可以在每个陪集中取一个代表元，构成一个集合。这个集合在计算群论被称为**截面** [10]。

定义 4.2.1 (截面)**.** *对于群 G 和它的子群 $H \le G$，设 H 导出的左陪集集合为 $C_1, C_2, \cdots, C_k$，则**包含单位元的**集合 $R = \{r_1, r_2, \cdots, r_k\}$ (其中 $r_i \in C_i$) 称为 H 的一个左截面，同理可以定义右截面。*

由于现在统一了使用右陪集，因此只需要考虑右截面。

考虑 G_{χ_1} 的一个右截面 $R = \{r_1 = e, r_2, r_3, \cdots, r_k\}$，它满足如下性质：

- 由定义知对于 $i \ne j$ 有 $Hr_i \ne Hr_j$，即 $r_i \circ r_j^{-1} \notin H$。
- 考虑陪集 Hr_i，任取其中元素 $h_0 \circ r_i$，那么有 $(h_0 \circ r_i)^{-1}(1) = \left(r_i^{-1} \circ h_0^{-1}\right)(1) = r_i^{-1}\left(h_0^{-1}(1)\right) = r_i^{-1}(1)$，也就是说，同一个陪集中的置换，具有相同的 1 的**原像**；而不同陪集中的置换则有不相同的原像。

[10] Transversal

- 那么，对于 $\forall g \in G$，我们根据 g 中 1 的原像 $g^{-1}(1)$ 就可以**唯一确定**它所在的陪集 Hr_i，也就是说，$Hg \cap R$ 包含唯一元素，我们称其为 g 的**标准置换**，记作 $\operatorname{norm} g$。

截面在 Schreier-Sims 算法中扮演着非常重要的角色，在后面的 Schreier 引理中会得到充分体现。

现在先回到元素判定，此时我们要判断 g 是否在 $\langle S \rangle$ 中。

我们希望找到一个 r_i 使得 $(r_i \circ g)(1) = 1$，也就是说 $r_i \circ g \in H \Leftrightarrow r_i \in Hg^{-1}$，也就是说求 $\operatorname{norm}(g^{-1})$。

注意到置换群的特殊性，一个置换的**标准置换**可以比较方便地求出：假设我们要求 $\operatorname{norm} g$，则可以先求出 $g^{-1}(1)$，找到 1 的原像和它相同的 r_i 即可。当然，如果不存在显然可以说明 $g \notin \langle S \rangle$。

找到了对应的 r_i 后，我们就得到了一个固定元素 1 的置换 $r_i \circ g$。那么，易知 $g \in G \Leftrightarrow r_i \circ g \in G_{\chi_1}$，于是我们成功转化为了子问题。

4.3 增量构造的过程

4.3.1 增量构造 —— S 影响 R

现在继续考虑增量构造，首先可以假设欲添加元素 $g \notin \langle S \rangle$，否则问题已经解决。

那么，改变了 S 后，考虑截面 R 会发生哪些变化。

回到置换群，R 中每个元素记录的是 1 的不同的**原像**。从这一点考虑，我们只需要知道 1 多了哪些原像即可。

设原先 1 的原像集合为 A_1，那么，当新增置换 g 后，考虑置换 $r_i \circ g$ $(r_i \in R)$，也就是说对于 $\forall p \in A_1$，假设 $r_i(p) = 1$，现在 $(r_i \circ g)^{-1}(1) = \left(g^{-1} \circ r_i^{-1}\right)(1) = g^{-1}\left(r_i^{-1}(1)\right) = g^{-1}(p)$ 也成了 1 的原像。

然后我们只需要枚举 S 中元素继续搜索即可。

从图论的角度来看，就是：把原先的轨道划分看成连通块，作出 g 对应的循环图 G_g，将这些边对应的连通块“连通”起来，就得到了新的轨道划分。于是我们先去找这些连接两个不同连通块的边 (即 g)，然后再将其它连通块中的值包含起来。

4.3.2 增量构造 —— R 影响 S'

我们现在已经成功处理了生成集 S 的变化对截面 R 的影响，现在就需要处理截面 R 的变化对稳定子群 G_{χ_1} 生成集，记为 S' 的影响。

看起来 S' 中添加了很多的置换，但是我们所维护的 $\langle S' \rangle$ 的增量必须是有限的，而且最好是可接受的。下面的 **Schreier 引理**就可以说明。

引理 4.3.1 (Schreier)**.** *设群 H 是群 $G=\langle S\rangle$ 的子群，R 为 H 的一个右截面，定义集合*

$$S'=\left\{(r\circ s)\circ(\operatorname{norm}(r\circ s))^{-1}\middle| r\in R,s\in S\right\}$$

则 $H=\langle S'\rangle$。

证明. *显然，$\langle S'\rangle\subseteq H$。下证 $H\subseteq\langle S'\rangle$。*

注意到 $e\in R$，因此 H 中任意一个元素 h 可以表示成:

$$h=r\circ s_1\circ s_2\circ\cdots\circ s_k$$

其中 $r\in R;s_1,s_2,\cdots,s_k\in S$。特别地，这里可以取 $r=e$。

接下来对 k 归纳证明：形如上式表示的元素一定在 $\langle S'\rangle$ 中。

当 $k=0$ 时，$h=r\in H\cap R=\{e\}$，故 $h\in\langle S'\rangle$。

设结论对 $k-1$ 成立，考虑 k，有

$$\begin{aligned}h&=r\circ s_1\circ s_2\circ\cdots\circ s_k\\&=(r\circ s_1)\circ(\operatorname{norm}(r\circ s_1))^{-1}\circ\operatorname{norm}(r\circ s_1)\circ s_2\circ\cdots\circ s_k\end{aligned}$$

注意到 $(r\circ s_1)\circ(\operatorname{norm}(r\circ s_1))^{-1}\in\langle S'\rangle,\operatorname{norm}(r\circ s_1)\in R$，故 $h\in\langle S'\rangle\Leftrightarrow\operatorname{norm}(r\circ s_1)\circ s_2\circ\cdots\circ s_k\in\langle S'\rangle$，即 $k-1$ 的子问题，由归纳假设知结论成立。

4.3.3 主要流程

有了 Schreier 引理后，我们就对 $\langle S'\rangle$ 有一个有限的刻画了。

由 Schreier 引理，我们可以通过 Cartesian 积 $R\times S$ 来构造 S'。因此，在增量构造中，R 对 S' 的影响就可以如下处理：

设 R 中新增了元素 r，我们枚举 S 中所有的元素 s，向 G_{χ_1} 中尝试添加 $(r\circ s)\circ(\operatorname{norm}(r\circ s))^{-1}$。

当然，由于之前是先在 S 中增加 g，因此我们也需要枚举 $r\in R$ 并加入 $(r\circ g)\circ(\operatorname{norm}(r\circ g))^{-1}$。

事实上，这两个搜索可以并到一起进行：

- 在“S 影响 R”的过程中，我们枚举 $\pi=r\circ g$，进入第二步；
- 如果 $G_{\chi_1}\pi\cap R=\varnothing$ (即 $\operatorname{norm}\pi$ 不存在)，则将其加入 R，并继续搜索 $\pi'=\pi\circ s$，回到第二步。

  否则，由于 $\operatorname{norm}\pi=\operatorname{norm}(r\circ g)$ 存在，故直接向 G_{χ_1} 中尝试添加 $\pi\circ(\operatorname{norm}\pi)^{-1}$ 即可。

4.3.4 Schreier-Sims 算法的伪代码

这个过程就可以用算法来描述了，它就被称为 Schreier-Sims 算法，它的伪代码如下：

Algorithm 1 test

Require: A permutation g

Ensure: Report whether $g \in \langle S \rangle$

```
pos = g(1)
if R[pos] = nil then
    return false
else
    if next = nil then
        return true
    else
        return next.test(R[pos] ∘ g)
    end if
end if
```

Algorithm 2 update_transversal

Require: A permutation g

Ensure: Update g as a transversal

```
pos = g^{-1}(1)
if R[pos] = nil then
    R[pos] = g
    for s ∈ S do
        update_transversal(g ∘ s)
    end for
else
    if next ≠ nil then
        next.update_generator(g ∘ R[pos]^{-1})
    end if
end if
```

Algorithm 3 update_generator

Require: A permutation g

Ensure: Update g as a generator

```
if test(g) then
    return
else
    S = S ∪ {g}
    for r ∈ R do
        update_transversal(r ∘ g)
    end for
end if
```

4.3.5 Schreier-Sims 算法的时间复杂度

上述就是 Schreier-Sims 代码的核心框架，下面分析它的时间复杂度。

首先还是考虑固定元素 1 的群论结构 (维护 $[G : G_{\chi_1}]$ 的)。

由上文 Light 算法的过程可知，按照上述算法产生的 n 阶群的生成集大小不超过 $\lfloor \log_2 n \rfloor$，因为每次添加元素后子群大小至少翻倍。

于是，对于 n 元对称群，这个生成集的大小不超过 $\log_2 |n!| = O(n \log n)$。

事实上，这其实是对称群的真子群链问题，由参考文献 [5] 可知生成集的大小不超过 $\frac{3}{2}n$，即 $|S| = O(n)$。

考虑计算 test 函数的时间复杂度，易知它的时间复杂度为 $O(n^2)$。对于每个群论结构，它调用 update_generator 的次数 (仅考虑通过 test 的) 为 $O(n)$，调用的 update_transversal 中使得截面 R 大小改变的次数也为 $O(n)$。于是调用的 update_transversal 中使得截面 R 大小不变的次数就不超过 $O(n^2)$，这也可以通过结论“S' 中每个元素可以通过 $R \times S$ 来构造”看出。

故该群论结构调用子结构的 test 函数至多 $O(n^2)$ 次，摊到该结构上的时间复杂度为 $O(n^4)$。因此 Schreier-Sims 算法的总时间复杂度为 $O(n^5)$。

不过由上述分析过程知，该算法的常数本身就非常小而且通常卡不满，因此实用价值比较高。不过，$O(n^5)$ 的确是该算法的上确界性，已经被 Knuth 证明，见参考文献 [4]。不过，在随机数据下的期望仍然是 $O(n^4)$。

5 总结

本文介绍了群论在 OI 中一些常见的运用，从置换、群、群的判定和表示以及计算群论等方面多角度介绍了群论。群论作为组合数学和抽象代数中极其重要的一个分支，但在信息学竞赛中目前并不普及，出现的题目也不是很多。

同时，计算群论中，除了 Schreier-Sims 作为其基础算法外，还有如 Todd-Coxeter, Product-replacement 等其它算法，以及很多很多东西等待我们去探索，可以说这里的水很深。

希望本文能起到抛砖引玉的作用，吸引更多读者来研究群论以及抽象代数类的问题。

感谢

感谢中国计算机学会提供学习和交流的平台。
感谢国家集训队高闻远教练的指导。
感谢符水波老师、应平安老师对我的关心与教导。
感谢罗煜翔、钱易等同学为本文验稿。
感谢父母对我的照顾与支持。

参考文献

[1] Wikipedia contributors. Group action. *Wikipedia*, https://en.wikipedia.org/wiki/Group_action.

[2] Rajagopalan, Sridhar; Schulman, Leonard J. (2000), *Verification of Identities*.

[3] Seress, A. (2002), *Permutation Group Algorithms*, Cambridge U Press.

[4] Knuth, Donald E. (1991), *Efficient representation of perm groups*, Combinatorica.

[5] Peter J. Cameron, Ron Solomon, Alexandre Turull (1988), *Chains of Subgroups in Symmetric Groups*.

[6] 罗雨屏 (2014)，抽象代数入门.