分治算法选讲

Part II

wYYSZL

2026.5.5

目录

序贡献分治

所谓“序贡献分治”，其实常说为“CDQ 分治”。这种分治与一般的分治不同的是，它的不同元素之间有一个“序”，在合并的时候，一般只有左边的元素对右边的元素产生贡献。

序贡献分治最常解决的问题是三维偏序问题，如果你把序贡献分治看作“分治解决高维偏序”的算法，也没有很大的问题。

“序贡献分治”这个词是我造的，至于为什么要起一个新名字，以及为什么要起这个名字，可能看这一部分最后的“闲话”。

序贡献分治

我们先来看如何利用序贡献分治解决二维偏序问题。

二维偏序

给定 n 个二元组，对每一组 (x_i, y_i)，求满足 $x_j < x_i, y_j < y_i$ 的 j 的个数（为了方便，我们假定 x 两两不同）。

序贡献分治

我们先来看如何利用序贡献分治解决二维偏序问题。

二维偏序

给定 n 个二元组，对每一组 (x_i, y_i)，求满足 $x_j < x_i, y_j < y_i$ 的 j 的个数（为了方便，我们假定 x 两两不同）。

二维偏序有各种做法：排序 + 树状数组，树套树等，我们考虑用排序 + 分治的做法。

序贡献分治

首先去重，序贡献分治一般解决不了有重复元素的问题，除非重复元素之间不算贡献。

我们按 x 从小到大为这些二元组排序。接下来我们考虑分治。

具体来说，对于 $[l, r]$ 这个子问题，考虑合并时的贡献。

我们可以假定左右区间分别按 y 从小到大排序。这样，我们用双指针 i, j 分别指向 l 和 $mid + 1$，即两个区间的开头。

序贡献分治

首先去重，序贡献分治一般解决不了有重复元素的问题，除非重复元素之间不算贡献。

我们按 x 从小到大为这些二元组排序。接下来我们考虑分治。

具体来说，对于 $[l, r]$ 这个子问题，考虑合并时的贡献。

我们可以假定左右区间分别按 y 从小到大排序。这样，我们用双指针 i, j 分别指向 l 和 $mid + 1$，即两个区间的开头。

尽管这样破坏了 x 的顺序，但由于是在右区间内部的排序，所以对所有 $i \in [l, mid], j \in [mid + 1, r]$，总有 $x_i < x_j$。第一维的限制已经被开始时的排序解决了，所以只有左边会对右边做出贡献，而且这样做就可以保证保证第一维的限制。

所以，我们只需统计 $y_i < y_j$ 的个数。这个用双指针扫一遍即可。

时间复杂度 $O(n \log n)$

例题 G

G - 【模板】三维偏序 / 陌上花开

有 n 个元素，第 i 个元素有 a_i, b_i, c_i 三个属性，设 $f(i)$ 表示满足 $a_j \le a_i$ 且 $b_j \le b_i$ 且 $c_j \le c_i$ 且 $j \ne i$ 的 j 的数量。

对于所有 $d \in [0, n)$，求 $f(i) = d$ 的数量。

$n \le 10^5$

例题 G 题解

回顾二维偏序的各种做法，我们的各种策略有一个共同的思想——降维。

实现降维，我们有很多方式，最常见的是：排序、数据结构、分治。而这些东西可以结合起来使用，每使用一个就会降维一维（当然排序只能使用一次）。

例题 G 题解

回顾二维偏序的各种做法，我们的各种策略有一个共同的思想——降维。

实现降维，我们有很多方式，最常见的是：排序、数据结构、分治。而这些东西可以结合起来使用，每使用一个就会降维一维（当然排序只能使用一次）。

因此，我们先考虑排序，剩下的就是不能通过排序降维的二维偏序。容易想到两种做法：树套树、分治 + 树状数组。

第一种很容易，考虑序贡献分治的做法。

例题 G 题解

回顾二维偏序的各种做法，我们的各种策略有一个共同的思想——降维。

实现降维，我们有很多方式，最常见的是：排序、数据结构、分治。而这些东西可以结合起来使用，每使用一个就会降维一维（当然排序只能使用一次）。

因此，我们先考虑排序，剩下的就是不能通过排序降维的二维偏序。容易想到两种做法：树套树、分治 + 树状数组。

第一种很容易，考虑序贡献分治的做法。我们不妨对 a 分治，按 b 合并。这样，在 j 指针统计贡献时，左区间内被记入的贡献已经满足了 $a_i \leq a_j, b_i \leq b_j$。依然考虑双指针。左指针 i 记入贡献时，在其上修改 c_i 处的值；右指针 j 统计贡献时，在其上查询 $\leq c_j$ 的总和即可。

时间复杂度：$O(n \log^2 n)$

数点问题

注意到 k 维偏序本质上是在处理 k 维的前缀查询。因此序贡献分治可以用于静态处理高维数点问题。

具体地，偏序问题每个元素，既是一个修改，也是一个查询。意味着一个元素既可以计入贡献，又可以统计贡献，取决于其所在的区间。而数点问题将查询独立出来了，也就是说只有左区间的修改操作能计入贡献，右区间的查询操作能统计贡献。

例题 H

H - 「DTOI-2」星之河

令 T 为一棵以 1 为根的有根树，节点集 $V = \{1, 2, \ldots, n\}$。每个节点 $v \in V$ 有两个权值 r_v, b_v。对于每个节点 x，计算 $\text{ans}(x) = |\{y \in S(x) \mid r_y \le r_x \text{ 且 } b_y \le b_x\}|$. 其中 $S(x) = \{y \in V \mid y \ne x \text{ 且 } y \text{ 在 } x \text{ 的子树中}\}$.

$n \le 2 \times 10^5$，$r_v, b_v \in [-10^9, 10^9]$。

例题 H

H - 「DTOI-2」星之河

令 T 为一棵以 1 为根的有根树，节点集 $V = \{1, 2, \ldots, n\}$。每个节点 $v \in V$ 有两个权值 r_v, b_v。对于每个节点 x，计算 $\text{ans}(x) = |\{y \in S(x) \mid r_y \le r_x \text{ 且 } b_y \le b_x\}|$. 其中 $S(x) = \{y \in V \mid y \ne x \text{ 且 } y \text{ 在 } x \text{ 的子树中}\}$.

$n \le 2 \times 10^5$，$r_v, b_v \in [-10^9, 10^9]$。

将点用 DFS 序重新编号，这样“i 在 j 子树中”的条件就转化为 $\text{dfn}_i \in [\text{dfn}_j, \text{dfn}_j + \text{size}_j - 1]$。再加上 r, b 两个属性，就是三维数点。

时间复杂度 $O(n \log^2 n)$。

例题 I

I - [USACO17FEB] Why Did the Cow Cross the Road III P

道路两侧各有一排田地，左侧田地编号 $1 \dots N$，右侧田地编号 $1 \dots N$（均从左到右）。两侧的奶牛品种编号均为 $1 \dots N$ 的排列（每个品种恰好出现一次）。将左侧每头奶牛与右侧同品种的奶牛连线，共 N 条线。若两条线相交，则称它们为一个交叉对。给定整数 K，称品种 a 与 b 友好当且仅当 $|a-b| \leq K$，否则不友好。求所有不友好的交叉对的数量。

$N \leq 10^5$

例题 I 题解

令 a_i 表示 i 在排列 A 中的位置，b_i 同理。

容易看出，两个数 i, j 的连线相交当且仅当满足下列一个条件：

- $a_i > a_j$ 且 $b_i < b_j$。
- $a_i < a_j$ 且 $b_i > b_j$。

不难看出 (i, j) 和 (j, i) 其实是同一种情况。接下来我们就默认使用第二种。到这里，我们很容易看出使用三维偏序来处理，前两维存入 x 在两个排列中的位置即可。

再来考虑 $|a - b| > k$ 的情况。我们可以将 x 的值存为第三维，将满足前两维条件的数用树状数组在 x 的位置插入 1。因为 $|a - b| > k$，所以对于 x 来说，$[x - k, x + k]$ 之间的数是没有贡献的。简单容斥可得，其真正有贡献的区间为 $[1, x - k) \cup (x + k, m]$。对于这个区间前缀和处理。

时间复杂度 $O(n \log^2 n)$。

例题 J

J - [HEOI2016/TJOI2016] 序列

有一个长度为 n 的整数序列。每个位置 i 有一个初始值 a_i，并给出若干种可能的变化：第 x 项可以变成 y。同一时刻最多只有一个位置的值发生变化（即修改独立）。请选出一个子序列，使得在原始序列以及所有可能的变化后的序列中，这个子序列都是不降的。求该子序列的最长长度。

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

例题 J 题解

考虑怎样两个数在最后的序列里面可以相邻。

对每个位置 i，记 $maxv[i]$ 为所有可能取值的最大值，$minv[i]$ 为最小值。合法子序列中相邻两项 $j < i$ 必须满足：$maxv[j] \le a[i]$ 且 $a[j] \le minv[i]$。考虑 DP，有：

$$dp[i] = \max_{j<i,\ maxv[j]\le a[i],\ a[j]\le minv[i]} \{dp[j]\} + 1$$

可以看作三维偏序问题：第一维位置顺序，第二维 $maxv[j]$ 与 $a[i]$，第三维 $a[j]$ 与 $minv[i]$。具体处理分治的时候，考虑在树状数组中存 DP 值，询问的时候取 max 即可。

时间复杂度 $O(n \log n)$。

闲话

“序贡献分治”这个词是我自己造的，主要是原来的名字没有对算法具体是什么做出限制，人们对其本质的讨论似乎也没有什么结果，这导致多数博客在介绍“CDQ 分治”时，经常将明明是分治的定义给予“CDQ 分治”。虽然讲述的内容大都是这个思想的常见应用，但不免造成误会。很多博客甚至以为“将序列通过递归的方式分给左右两个区间，每一个子问题只处理跨左右区间的贡献。”就是“CDQ 分治”，那分治思想又是什么呢？

当然，“序贡献分治”这个名词也并不太好，而且我们实际上我们就没有一个关于“CDQ 分治”和普通分治的界定，但它毕竟符合“在时间维度上做分治”的原始宽泛定义，符合“只有一边对另一边产生贡献”的分治，而且竞赛中用到的“CDQ 分治”多数解决的是高位偏序问题及解决这种问题的范式能优化的其他问题。这个名词也可以有效区分 Part I 所讲的“序列分治”。总之堪堪够用。

目录

例题 K

K - 天天爱射击

n 个木板，平行于 x 轴，第 i 块木板区间为 $[x_{1,i}, x_{2,i}]$，耐久 s_i。子弹按时间顺序沿 y 轴方向射击，坐标 p_j。子弹贯穿所有包含其 x 坐标且未破碎的木板，命中 s_i 次后木板消失。需要输出每颗子弹射出后有多少木板首次破碎。

$n \leq 2 \times 10^5$

例题 K 题解

考虑只有一块木板，想知道它在第几颗子弹时破碎。它可以二分答案 mid：检查前 mid 颗子弹中落在 $[x_1, x_2]$ 的数量是否达到 s。实现时将前 mid 颗子弹坐标插入树状数组，查询区间和。若数量 $\geq s$，则答案 $\leq mid$；否则 $> mid$。最终求出的时刻 t 就是破碎时间。

例题 K 题解

考虑只有一块木板，想知道它在第几颗子弹时破碎。它可以二分答案 mid：检查前 mid 颗子弹中落在 $[x_1, x_2]$ 的数量是否达到 s。实现时将前 mid 颗子弹坐标插入树状数组，查询区间和。若数量 $\geq s$，则答案 $\leq mid$；否则 $> mid$。最终求出的时刻 t 就是破碎时间。

很多木板都需要二分，而且二分判定时都要查询前若干子弹的区间计数。单独处理导致子弹被反复插入树状数组。整体二分的思路是：一次性对所有木板进行二分，共享二分过程和数据结构。类似于分治，每次处理一个子弹时间区间 $[L, R]$ 以及一批答案落在此区间的木板，用中点的子弹统一判定并分治下去。

例题 K 题解

具体的，考虑分治实现，假设已知集合 $boards$ 是答案在 $[L, R]$ 的木板集合。将时间在 $[L, mid]$ 的子弹插入树状数组。对每块木板查询其区间内的子弹数 cnt。若 $cnt \geq s$，则该木板答案在 $[L, mid]$，归入左集合；否则答案在右集合，归入右集合。之后分治处理。

一个细节是，如果一个木板归入右集合，要减去左边子弹的贡献，即 $s \leftarrow s - cnt$。这样以来，递归右集合的时候，我们只需考虑这段区间内的子弹即可。所以当开始分治的时候，需要清空树状数组。

考虑时间复杂度，每个木板每层只会询问一次，所以树状数组询问次数为 $O(n \log n)$；每个子弹每层只会插入 $O(1)$ 次，所以插入次数为 $O(n \log n)$（假设 n 和值域同阶）。总的复杂度 $O(n \log^2 n)$。

单 log 的整体二分

考虑优化掉树状数组的 log。

我们将求解答案 $sum_{l,r}$ 拆成前缀和，问题转化为如何快速地求所有需要的前缀和。

注意到，如果我们能实现每一层的时候，得到一个下标（x 值）有序的询问（前缀和的）序列和下标有序的子弹序列，我们可以通过双指针，一边对子弹求前缀和，一边记录下来询问的答案。我们无需在整个下标下跑前缀和，只需在有意义的点（有询问/有子弹）计算即可。（类似离散化）这样，一个区间的复杂度是 $O(\quad + \quad)$。整个复杂度 $O(n \log n)$。

做到这个也不难。考虑一开始按照下标排序，然后分治的时候，遍历所有在这个时间区间内的询问/修改，看它们的时间，分到两个子问题中，同时保证下标的顺序。

例题 L

L - 可持久化线段树 2

如题，给定 n 个整数构成的序列 a，m 次询问，查询闭区间 $[l, r]$ 内的第 k 小值。

$1 \leq n, m \leq 2 \times 10^5$

例题 L

L - 可持久化线段树 2

如题，给定 n 个整数构成的序列 a，m 次询问，查询闭区间 $[l, r]$ 内的第 k 小值。
$1 \leq n, m \leq 2 \times 10^5$

考虑可持久化线段树。考虑整体二分。对值域二分，假设二分的值为 mid，树状数组维护 $\leq mid$ 的数的个数，之后就容易了。
时间复杂度 $O((n+m)\log^2 n)$。当然也可以一个 $\log$。

* 例题 M

M - 网络

给定一棵 n 个节点的树，节点编号 $1, \ldots, n$。按时间顺序给出 m 个操作，操作类型如下：

- 添加一个请求，其路径为树上 a 到 b 的简单路径，重要度为 v
- 删除第 t 个操作所添加的请求
- 询问当前所有未被删除的请求中，路径不包含节点 x 的那些请求的重要度的最大值

$1 \leq n, m \leq 2 \times 10^5$。

* 例题 M 题解

先考虑如何简单二分答案，如果某个询问点被所有大于当前答案的路径所经过，那么答案小于等于当前答案，否则大于等于当前答案。

如何统计被经过的路径数量？考虑树上差分，设一条路径 $x \to y$，则 x, y 的差分数组 $+1$，lca, fa_{lca} 的差分数组 -1，数量就是其子树差分数组的和。

之后整体二分即可。

时间复杂度 $O(n \log^2 n)$。

* 例题 N

N - WD 与地图

给定一张 n 个点 m 条边的有向图，每个点有一个点权，要求支持以下操作：

- 删掉一条之前存在的边。
- 更改一个点的点权。
- 询问某个点所在强连通分量内部前 k 大的点权之和。

* 例题 N 题解

反转时间轴，转化为加边。问题化为维护强连通分量（SCC）。

若知每条边两端点同属一个 SCC 的时间，则按该时间排序，依次合并两端 SCC（并查集维护）。每个 SCC 维护权值线段树存内部点权，加边时合并线段树，修改只需单点更新。

求每条边合并时间：二分答案。设当前答案为 x，加入出现时间 $\leq x$ 的边，跑 Tarjan 判断两端是否在同一 SCC。改为整体二分：设当前答案区间 $[l, r]$，$mid = \lfloor \frac{l+r}{2} \rfloor$，区间内边集 $[a, b]$。先加入出现时间 $[0, l-1]$ 的边，再遍历 $[a, b]$，加入出现时间 $\leq mid$ 的边，得出现时间 $\leq mid$ 的图，跑 Tarjan。用可撤销并查集维护 Tarjan 缩点结果，上一层已缩点形成“虚图”。处理完 $[l, r]$ 后递归 $[mid+1, r]$，撤销后递归 $[l, mid]$。每条边处理 $O(\log n)$ 次，总复杂度 $O(m \log^2 n)$。

复杂度 $O(m \log^2 n)$

目录

笛卡尔树

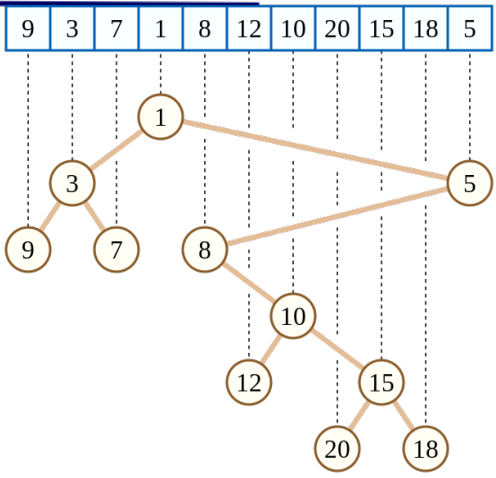
笛卡尔树是一种特殊的二叉树，其包含至少两个维度，一个维度满足二叉搜索树的性质，另一个维度满足堆的性质。

常见的笛卡尔树（例如在序列上）下标满足二叉搜索树的性质，权值满足堆的性质，也就是说：

- 对笛卡尔树进行中序遍历，得到的序列就是原数列。
- 每个节点的子节点都比该节点要大/小

例如，下面就是一个笛卡尔树：

笛卡尔树



笛卡尔树的构造

考虑如何构造笛卡尔树。最直接的想法就是每次找出区间最值所在的位置。随便找一个 RMQ 的算法可以做到 $O(n \log n)$。

笛卡尔树的构造

考虑如何构造笛卡尔树。最直接的想法就是每次找出区间最值所在的位置。随便找一个 RMQ 的算法可以做到 $O(n \log n)$。

有一种线性建树的方法（以堆为小根堆为例）：我们考虑将元素按满足 BST 性质的一维（下称 x 维）（一般是下标）升序依次插入到当前的笛卡尔树中。

对于一棵笛卡尔树，定义「右链」为从根节点开始一直走右儿子，走到叶节点形成的链。则插入节点后，这个节点一定在右链上。因为是按照满足 BST 性质的 x 升序插入，那么这个新插入的节点必然在树的最右端。这个节点不可能是一个左儿子，也没有右儿子。

于是我们执行这样一个过程，从下往上比较右链节点与当前节点 u 的 v（第二维，满足堆性质），如果找到了一个右链上的节点 y 满足 $w_y < w_u$，就把 u 接到 y 的右儿子上，而 y 原本的右子树就变成 u 的左子树。

笛卡尔树的构造

显然每个数最多进出右链一次（或者说每个点在右链中存在的是一段连续的时间）。这个过程可以用一个栈维护，栈中维护当前笛卡尔树的右链上的节点。一个点不在右链上了就把它弹掉。这样每个点最多进出一次，复杂度 $O(n)$。

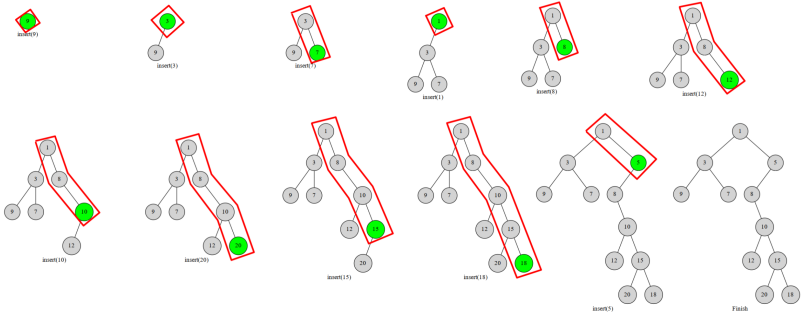
Hint

插入的过程可以看作 BST 左旋使之满足堆的性质，然后加入一个新的节点。所以最后建出来的树“同时满足”BST 和堆的性质。

以及很多时候用不着 $O(n)$ 建树。

笛卡尔树的构造

OI-wiki 上的图片非常清晰明了，因此借用过来便于理解：（图中红框部分就是我们始终维护的右链）



例题 O

O - 笛卡尔树

给定一个 $1 \sim n$ 的排列 p，构建其笛卡尔树。

即构建一棵二叉树，满足：

1. 每个节点的编号满足二叉搜索树的性质。
2. 节点 i 的权值为 p_i，每个节点的权值满足小根堆的性质。

$1 \leq n \leq 10^7$。

例题 O

O - 笛卡尔树

给定一个 $1 \sim n$ 的排列 p，构建其笛卡尔树。

即构建一棵二叉树，满足：

1. 每个节点的编号满足二叉搜索树的性质。
2. 节点 i 的权值为 p_i，每个节点的权值满足小根堆的性质。

$1 \leq n \leq 10^7$。

模板题，没什么说的。

例题 P

P - 树的序

二叉查找树的形态和键值的插入顺序密切相关。准确的讲：

1. 空树中加入一个键值 k，则变为只有一个结点的二叉查找树，此结点的键值即为 k。
2. 在非空树中插入一个键值 k，若 k 小于其根的键值，则在其左子树中插入 k，否则在其右子树中插入 k。

我们将一棵二叉查找树的键值插入序列称为树的生成序列，现给出一个生成序列，求与其生成同样二叉查找树的所有生成序列中字典序最小的那个，其中，字典序关系是指对两个长度同为 n 的生成序列，先比较第一个插入键值，再比较第二个，依此类推。$1 \leq n\, 10^5$。

例题 P 题解

考虑建出这个搜索树。然后从根节点开始重新建树。注意到接下来可以加进来的点就是与已经重建好的点相邻的所有点，找到最小的即可。朴素建树的复杂度是 $O(n^2)$，可以用 ST 表或者其他什么技巧做到 $O(n \log n)$。

例题 P 题解

考虑建出这个搜索树。然后从根节点开始重新建树。注意到接下来可以加进来的点就是与已经重建好的点相邻的所有点，找到最小的即可。朴素建树的复杂度是 $O(n^2)$，可以用 ST 表或者其他什么技巧做到 $O(n \log n)$。

另解：考虑从笛卡尔树的角度思考。注意到插入顺序在后的一定是插入顺序在前的点的子节点，所以插入顺序满足堆性质。BST 本身满足 BST 的性质，所以可以看出是笛卡尔树。键出这样的笛卡尔树，实际上就键出了原来的 BST，剩下的就同上了。排序瓶颈，时间复杂度 $O(n \log n)$。

例题 Q

Q - Recursive Queries

有一个长度为 n 的排列 $p_1, p_2, \ldots, p_n$，还有 q 次询问，每次询问一段区间 $[l_i, r_i]$，你需要计算出 $f(l_i, r_i)$。

定义 $m_{l,r}$ 表示 $p_l, p_{l+1}, \ldots, p_r$ 这一段数列的最大值的出现位置，则

$$f(l,r) = \begin{cases} (r-l+1) + f(l, m_{l,r}-1) + f(m_{l,r}+1, r) & \text{if } l <= r \\ 0 & \text{else} \end{cases}$$

$n, q \leq 10^6$

例题 Q 题解

注意到把区间的贡献算到单点上，就是求对 $[l,r]$ 的元素建出笛卡尔树，求每个节点的 $\mathrm{dep}/\mathrm{size}$ 之和：

对于一个区间 $[l,r]$，它的笛卡尔树可以通过前缀 $[1,r]$ 的笛卡尔树得到。求出 $[l,r]$ 最大值位置 p 后，p 显然是 $[l,r]$ 笛卡尔树的根。由于一个子树对应的下标连续，因此 $[p+1,r]$ 对答案的贡献可以通过求它们在 $[1,r]$ 的笛卡尔树上的 dep 之和减去 $\mathrm{dep}_p \times (r-p)$ 得到，即 p 的右子树一定是 $[p+1,r]$ 的笛卡尔树。

求区间最大值可以在单调栈维护的笛卡尔树的右键上二分。

整理一下，在加入节点时我们需要支持区间修改：不断弹出小于当前值的栈顶直到大于当前值，设其下标为 p，那么 p 的右子树对应的下标 $[p+1,r]$ 的深度加上 1。查询时需要区间查询。BIT 即可，时间复杂度 $\mathcal{O}((n+q)\log n)$。

* 例题 Q

Q - Removing Blocks

有 N 个方块从左到右排成一排，编号为 1 到 N。每个方块都有一个权重，第 i 个方块的权重为 A_i。Snuke 将对这些方块执行 N 次如下操作：

每次选择一个尚未被移除的方块并将其移除。该操作的代价是与被移除方块相连的所有方块的权重之和（包括其本身）。这里，若两个方块 x 和 y（$x \leq y$）之间的所有方块 z（$x \leq z \leq y$）都尚未被移除，则称 x 和 y 是相连的。

Snuke 可以以 $N!$ 种不同的顺序移除这些方块。请你对所有这 $N!$ 种移除顺序，计算每种顺序下所有操作的总代价，并求出这些总代价之和。由于答案可能非常大，请对 $10^9 + 7$ 取模后输出。

$1 \leq N \leq 10^5$

* 例题 Q 题解

考虑每个点对答案的贡献：如果按照删除时间为值建出笛卡尔树，那么一个点的贡献应为其在笛卡尔树上的深度。

这类统计 sum 的计数题可以转化成方案数 $\times$ 答案的期望，前者是 $n!$，而后者根据期望的线性性，可以被拆为 $)\sum a_i \times E(d_i)$ 再进一步拆为 $\sum a_i \times \sum_{u \neq i} E\big([u \text{ is the ancestor of } i]\big)$. 考虑 u 成为 i 的祖先的概率（不妨设 $u < i$，反之同理），这意味着如果只看下标 $u \sim i$，那么 u 是第一个被删去的，因此其余所有数被删去的顺序对答案没有影响，而在所有 $(i-u+1)!$ 种删去 $u \sim i$ 的排列中，只有 $(i-u)!$ 种是合法的（即 u 排在第一个），因此期望加上 $\frac{1}{i-u+1}$. 考虑枚举每个成为 i 的祖先的下标 u，那么 $E = \sum a_i \times \sum_u \frac{1}{|i-u+1|}$.

预处理 $\frac{1}{i}$ 关于 i 的前缀和即可做到 $\mathcal{O}(n)$。

例题 R

R - Subsequence

Alice 有一个长度为 n 的整数序列 a，所有元素都是不同的。她将选择一个长度为 m 的 a 的子序列，并将一个子序列 $a_{b1}, a_{b2}, ..., a_{bm}$ 的价值定义为

$$\sum_{i=1}^{m}(m \cdot a_{b_i}) - \sum_{i=1}^{m}\sum_{j=1}^{m} f(\min(b_i, b_j), \max(b_i, b_j))$$

其中 $f(i, j)$ 表示 $\min(a_i, a_{i+1}, ..., a_j)$。

Alice 希望你能帮助她将她所选择的子序列的价值最大化。

$m \leq n \leq 4000$。

例题 R 题解

我们建出笛卡尔树，然后是裸的树形背包。设 $f_{i,j}$ 表示在 i 的子树内选了 j 个点的最大值，若不选 i 则有：

$$f_{i,j} = \max_{k=0}^{j} f_{i,k} + f_{r,j-k} - j \times k \times v_i$$

选 i 则有：

$$f_{i,j} = \max_{k=0}^{j-1} f_{i,k} + f_{r,j-k-1} - (j \times k + 2j - 1 - m) \times v_i$$

时间复杂度 $O(n^2)$。

“四毛子算法”

笛卡尔树还有一个应用，即 $O(n)-O(1)$ RMQ（区间最值）。

大概的思想是首先对序列建出笛卡尔树，则问题转化为求两个点的树上 LCA。不难看出求得笛卡尔树的欧拉序后，这是一个 ± 1 RMQ 问题，可以使用分块的思想在 $O(n)-O(1)$ 的时间内解决：

具体地，取块长 $B=\left\lfloor\frac{\log_2 n}{2}\right\rfloor$，将整个序列分成 $\lceil\frac{n}{B}\rceil$ 个块，对于整块之间的 RMQ 可以 ST 表 $O\left(\frac{n}{B}\log\frac{n}{B}\right)=O(n)$ 预处理。

而对于块内任意子区间的 RMQ，注意到本质不同的序列只有 $2^B=\sqrt{n}$ 个（差分数组只有 $+1$ 或 -1），因此可以对于这 $\sqrt{n}$ 个序列分别 $O(B)$ 预处理：为什么不是 $O(B^2)$ 预处理？实际上如果你要求块 $[l_i,r_i]$ 内区间 $[l,r]$ 最大值，那么左边没有值的位置 $(p\in[l_i,l])$ 可视作 $+1$，右边没有值的位置 $(p\in(r,r_i])$ 可视作 -1。