数据结构初步

Claris

Hangzhou Dianzi University

2018 年 2 月 2 日

Overview

- 本节课介绍几种数据结构以及一些技巧：

Overview

- 本节课介绍几种数据结构以及一些技巧：
- 全局最大值的维护。

Overview

- 本节课介绍几种数据结构以及一些技巧：
- 全局最大值的维护。
- 单调栈/单调队列。

Overview

- 本节课介绍几种数据结构以及一些技巧：
- 全局最大值的维护。
- 单调栈/单调队列。
- 链表。

Overview

- 本节课介绍几种数据结构以及一些技巧：
- 全局最大值的维护。
- 单调栈/单调队列。
- 链表。
- 寻找不变量与变化点。

Overview

- 本节课介绍几种数据结构以及一些技巧：
- 全局最大值的维护。
- 单调栈/单调队列。
- 链表。
- 寻找不变量与变化点。
- 并查集。

Overview

- 本节课介绍几种数据结构以及一些技巧：
- 全局最大值的维护。
- 单调栈/单调队列。
- 链表。
- 寻找不变量与变化点。
- 并查集。
- 时间倒流法。

Rage Minimum Query

维护一个序列 $a_1, a_2, \ldots, a_n$ ，m 次修改。

Rage Minimum Query

维护一个序列 $a_1, a_2, ..., a_n$ ，m 次修改。
每次指定一个 x ，将 a_x 修改为 y。

Rage Minimum Query

维护一个序列 $a_1, a_2, \ldots, a_n$ ，m 次修改。
每次指定一个 x ，将 a_x 修改为 y。
求每次修改之后的全局最小值。

Rage Minimum Query

维护一个序列 $a_1, a_2, \ldots, a_n$ ， m 次修改。
每次指定一个 x ，将 a_x 修改为 y。
求每次修改之后的全局最小值。
数据在指定范围内完全随机生成。

Rage Minimum Query

维护一个序列 $a_1, a_2, \dots, a_n$，m 次修改。
每次指定一个 x，将 a_x 修改为 y。
求每次修改之后的全局最小值。
数据在指定范围内完全随机生成。

- $n \leq 10^7$。

Rage Minimum Query

维护一个序列 $a_1, a_2, \dots, a_n$ ，m 次修改。
每次指定一个 x，将 a_x 修改为 y。
求每次修改之后的全局最小值。
数据在指定范围内完全随机生成。

- $n \le 10^7$。
- $m \le 5 \times 10^7$。

Rage Minimum Query

维护一个序列 $a_1, a_2, \ldots, a_n$ ，m 次修改。
每次指定一个 x ，将 a_x 修改为 y。
求每次修改之后的全局最小值。
数据在指定范围内完全随机生成。

- $n \leq 10^7$。
- $m \leq 5 \times 10^7$。
- $0 \leq a_i < 2^{32}$。

Rage Minimum Query

维护一个序列 $a_1, a_2, \ldots, a_n$ ，m 次修改。
每次指定一个 x，将 a_x 修改为 y。
求每次修改之后的全局最小值。
数据在指定范围内完全随机生成。

- $n \leq 10^7$。
- $m \leq 5 \times 10^7$。
- $0 \leq a_i < 2^{32}$。
- Source：XVII Open Cup named after E.V. Pankratiev. GP of Moscow Workshops

Solution

- 若 $y \leq a_x$ ，那么显然 $ans = \min(ans, y)$ 即可。

Solution

- 若 $y \le a_x$ ，那么显然 $ans = \min(ans, y)$ 即可。
- 否则当且仅当 a_x 之前是最小值时答案才可能会改变。

Solution

- 若 $y \le a_x$ ，那么显然 $ans = \min(ans, y)$ 即可。
- 否则当且仅当 a_x 之前是最小值时答案才可能会改变。
- a_x 是最小值的概率为 $\frac{1}{n}$ ，暴力重算最小值即可。

Solution

- 若 $y \le a_x$，那么显然 $ans = \min(ans, y)$ 即可。
- 否则当且仅当 a_x 之前是最小值时答案才可能会改变。
- a_x 是最小值的概率为 $\frac{1}{n}$，暴力重算最小值即可。
- 时间复杂度 $O(n+m)$。

+-1 Maximum Query

维护一个序列 $a_1, a_2, \ldots, a_n$ ，m 次修改。

+-1 Maximum Query

维护一个序列 $a_1, a_2, \ldots, a_n$ ， m 次修改。

每次指定一个 x ，将 a_x 增加 1 或者减少 1。

+-1 Maximum Query

维护一个序列 $a_1, a_2, \ldots, a_n$ ，m 次修改。

每次指定一个 x，将 a_x 增加 1 或者减少 1。

求每次修改之后的全局最大值。

+-1 Maximum Query

维护一个序列 $a_1, a_2, ..., a_n$ ，m 次修改。
每次指定一个 x，将 a_x 增加 1 或者减少 1。
求每次修改之后的全局最大值。

- $n \leq 10^7$。

+-1 Maximum Query

维护一个序列 $a_1, a_2, ..., a_n$ ，m 次修改。
每次指定一个 x ，将 a_x 增加 1 或者减少 1。
求每次修改之后的全局最大值。

- $n \leq 10^7$。
- $m \leq 10^7$。

+-1 Maximum Query

维护一个序列 $a_1, a_2, \ldots, a_n$ ，m 次修改。
每次指定一个 x，将 a_x 增加 1 或者减少 1。
求每次修改之后的全局最大值。

- $n \leq 10^7$。
- $m \leq 10^7$。
- $0 \leq a_i \leq n$。

Solution

- 若是增加，那么还是同理，$ans = \max(ans, a_x + 1)$。

Solution

- 若是增加，那么还是同理，$ans = \max(ans, a_x + 1)$。
- 否则最大值最多减少 1。

Solution

- 若是增加，那么还是同理，$ans = \max(ans, a_x + 1)$。
- 否则最大值最多减少 1。
- 记录每个数字出现次数，判断当前还剩几个最大值即可。

Solution

- 若是增加，那么还是同理，$ans = \max(ans, a_x + 1)$。
- 否则最大值最多减少 1。
- 记录每个数字出现次数，判断当前还剩几个最大值即可。
- 时间复杂度 $O(n + m)$。

+-1 Maximum Query 2

维护一个序列 $a_1, a_2, \ldots, a_n$ ， m 次修改。

+-1 Maximum Query 2

维护一个序列 $a_1, a_2, \ldots, a_n$ ， m 次修改。

每次指定一个 x ，将 a_x 增加 1 或者减少 1。

+-1 Maximum Query 2

维护一个序列 $a_1, a_2, \ldots, a_n$ ，m 次修改。

每次指定一个 x，将 a_x 增加 1 或者减少 1。

求每次修改之后的全局最大值。

+-1 Maximum Query 2

维护一个序列 $a_1, a_2, \ldots, a_n$ ， m 次修改。
每次指定一个 x ，将 a_x 增加 1 或者减少 1。
求每次修改之后的全局最大值。

- $n \leq 10^7$。

+-1 Maximum Query 2

维护一个序列 $a_1, a_2, \ldots, a_n$ ，m 次修改。
每次指定一个 x，将 a_x 增加 1 或者减少 1。
求每次修改之后的全局最大值。

- $n \leq 10^7$。
- $m \leq 10^7$。

+-1 Maximum Query 2

维护一个序列 $a_1, a_2, ..., a_n$ ，m 次修改。
每次指定一个 x ，将 a_x 增加 1 或者减少 1。
求每次修改之后的全局最大值。

- $n \leq 10^7$。
- $m \leq 10^7$。
- $0 \leq a_i \leq 10^9$。

Solution

- 数字范围太大，不能直接记录出现次数。

Solution

- 数字范围太大，不能直接记录出现次数。
- 注意到只有与初始最大值差值不超过 m 的数才有用。

Solution

- 数字范围太大，不能直接记录出现次数。
- 注意到只有与初始最大值差值不超过 m 的数才有用。
- 只记录这 $O(m)$ 个数的个数即可。

Solution

- 数字范围太大，不能直接记录出现次数。
- 注意到只有与初始最大值差值不超过 m 的数才有用。
- 只记录这 $O(m)$ 个数的个数即可。
- 时间复杂度 $O(n+m)$。

滑窗最大值

给定一个序列 $a_1, a_2, \ldots, a_n$ ，求每个长度为 k 的子区间的区间最大值。

滑窗最大值

给定一个序列 $a_1, a_2, ..., a_n$ ，求每个长度为 k 的子区间的区间最大值。

- $1 \le k \le n \le 10^7$。

Solution

- 从左往右考虑每个数。

Solution

- 从左往右考虑每个数。
- 若 $a_i \le a_j$ 且 $i < j$，那么 i 永远不会比 j 优。

Solution

- 从左往右考虑每个数。
- 若 $a_i \leq a_j$ 且 $i < j$，那么 i 永远不会比 j 优。
- 可以整理出一个单调递减的队列。

Solution

- 从左往右考虑每个数。
- 若 $a_i \le a_j$ 且 $i < j$，那么 i 永远不会比 j 优。
- 可以整理出一个单调递减的队列。
- 若队首不在区间内，则出队，否则队首就是最大值。

Solution

- 从左往右考虑每个数。
- 若 $a_i \leq a_j$ 且 $i < j$，那么 i 永远不会比 j 优。
- 可以整理出一个单调递减的队列。
- 若队首不在区间内，则出队，否则队首就是最大值。
- 不断将不满足递减性质的队尾弹出。

Solution

- 从左往右考虑每个数。
- 若 $a_i \le a_j$ 且 $i < j$，那么 i 永远不会比 j 优。
- 可以整理出一个单调递减的队列。
- 若队首不在区间内，则出队，否则队首就是最大值。
- 不断将不满足递减性质的队尾弹出。
- 时间复杂度 $O(n)$。

Parcel

$n \times n$ 的 01 方阵，从中找到一个面积最大的全 0 矩阵。

Parcel

$n \times n$ 的 01 方阵，从中找到一个面积最大的全 0 矩阵。

- $n \leq 2000$。

Parcel

$n \times n$ 的 01 方阵，从中找到一个面积最大的全 0 矩阵。

- $n \leq 2000$。
- Source： POI 2002

Solution

- 从上往下枚举矩阵的下底边。

Solution

- 从上往下枚举矩阵的下底边。
- 设 h_i 表示 i 往上最大延伸长度，可以在枚举的同时维护出来。

Solution

- 从上往下枚举矩阵的下底边。
- 设 h_i 表示 i 往上最大延伸长度，可以在枚举的同时维护出来。
- 一个可能成为答案的子矩阵必然是以某个 h_i 作为最小值。

Solution

- 从上往下枚举矩阵的下底边。
- 设 h_i 表示 i 往上最大延伸长度，可以在枚举的同时维护出来。
- 一个可能成为答案的子矩阵必然是以某个 h_i 作为最小值。
- 用单调栈求出每个 h_i 往左往右第一个比它大的位置 l_i, r_i。

Solution

- 从上往下枚举矩阵的下底边。
- 设 h_i 表示 i 往上最大延伸长度，可以在枚举的同时维护出来。
- 一个可能成为答案的子矩阵必然是以某个 h_i 作为最小值。
- 用单调栈求出每个 h_i 往左往右第一个比它大的位置 l_i, r_i。
- 则 h_i 作为最小值的范围是 $[l_i + 1, r_i - 1]$，子矩阵面积为 $h_i \times (r_i - l_i - 1)$。

Solution

- 从上往下枚举矩阵的下底边。
- 设 h_i 表示 i 往上最大延伸长度，可以在枚举的同时维护出来。
- 一个可能成为答案的子矩阵必然是以某个 h_i 作为最小值。
- 用单调栈求出每个 h_i 往左往右第一个比它大的位置 l_i, r_i。
- 则 h_i 作为最小值的范围是 $[l_i+1, r_i-1]$，子矩阵面积为 $h_i \times (r_i - l_i - 1)$。
- 时间复杂度 $O(n^2)$。

Kanade's sum

给定一个 1 到 n 的全排列 $A_1, A_2, \ldots, A_n$ 以及一个参数 k。

Kanade's sum

给定一个 1 到 n 的全排列 $A_1, A_2, \ldots, A_n$ 以及一个参数 k。对于一个区间 $[l, r]$，设 $f(l, r)$ 为 $[l, r]$ 区间内第 k 大的数。

Kanade's sum

给定一个 1 到 n 的全排列 $A_1, A_2, ..., A_n$ 以及一个参数 k。对于一个区间 $[l, r]$，设 $f(l, r)$ 为 $[l, r]$ 区间内第 k 大的数。求所有区间的 f 的和。

Kanade's sum

给定一个 1 到 n 的全排列 $A_1, A_2, \ldots, A_n$ 以及一个参数 k。对于一个区间 $[l, r]$，设 $f(l, r)$ 为 $[l, r]$ 区间内第 k 大的数。求所有区间的 f 的和。

- $n \leq 500000$。

Kanade's sum

给定一个 1 到 n 的全排列 $A_1, A_2, ..., A_n$ 以及一个参数 k。对于一个区间 $[l, r]$，设 $f(l, r)$ 为 $[l, r]$ 区间内第 k 大的数。求所有区间的 f 的和。

- $n \leq 500000$。
- $k \leq 80$。

Kanade's sum

给定一个 1 到 n 的全排列 $A_1, A_2, ..., A_n$ 以及一个参数 k。对于一个区间 $[l, r]$，设 $f(l, r)$ 为 $[l, r]$ 区间内第 k 大的数。求所有区间的 f 的和。

- $n \leq 500000$。
- $k \leq 80$。
- Source：2017 Multi-University Training Contest 3

Solution

- 考虑统计每个位置作为多少个区间的第 k 大的数。

Solution

- 考虑统计每个位置作为多少个区间的第 k 大的数。
- 对于每个位置 i，往左往右找到 k 个比 A_i 大的数的位置。

Solution

- 考虑统计每个位置作为多少个区间的第 k 大的数。
- 对于每个位置 i，往左往右找到 k 个比 A_i 大的数的位置。
- 将这些位置组合可以在 $O(k)$ 时间内计算一个 i 的贡献。

Solution

- 考虑统计每个位置作为多少个区间的第 k 大的数。
- 对于每个位置 i，往左往右找到 k 个比 A_i 大的数的位置。
- 将这些位置组合可以在 $O(k)$ 时间内计算一个 i 的贡献。
- 建立 1 到 n 的链表，从小到大依次删去每个数，则每个数的 k 个前驱和后继就是对应的位置。

Solution

- 考虑统计每个位置作为多少个区间的第 k 大的数。
- 对于每个位置 i，往左往右找到 k 个比 A_i 大的数的位置。
- 将这些位置组合可以在 $O(k)$ 时间内计算一个 i 的贡献。
- 建立 1 到 n 的链表，从小到大依次删去每个数，则每个数的 k 个前驱和后继就是对应的位置。
- 时间复杂度 $O(nk)$。

寻找不变量与变化点

- 寻找问题中不变的部分，起到优化效果。

寻找不变量与变化点

- 寻找问题中不变的部分，起到优化效果。
- 分析清楚变化点，考虑如何维护变化。

Zapiekanki 2

n 个顾客按时间顺序依次点餐，每个顾客会恰好点一份砂锅。第 i 个顾客点餐时间为 t_i，而且只接受 $\geq t_i$ 时刻之后出锅的砂锅。

Zapiekanki 2

n 个顾客按时间顺序依次点餐，每个顾客会恰好点一份砂锅。第 i 个顾客点餐时间为 t_i，而且只接受 $\geq t_i$ 时刻之后出锅的砂锅。

你只有一个烤箱，做一份砂锅需要 d 时间，且中间不能打开烤箱，也不能同时做多份砂锅，你可以从 0 时刻开始做砂锅。

Zapiekanki 2

n 个顾客按时间顺序依次点餐，每个顾客会恰好点一份砂锅。第 i 个顾客点餐时间为 t_i，而且只接受 $\geq t_i$ 时刻之后出锅的砂锅。

你只有一个烤箱，做一份砂锅需要 d 时间，且中间不能打开烤箱，也不能同时做多份砂锅，你可以从 0 时刻开始做砂锅。

m 次询问，每次给定一个 d，求所有顾客等待时间之和的最小值。

Zapiekanki 2

n 个顾客按时间顺序依次点餐，每个顾客会恰好点一份砂锅。第 i 个顾客点餐时间为 t_i，而且只接受 $\geq t_i$ 时刻之后出锅的砂锅。

你只有一个烤箱，做一份砂锅需要 d 时间，且中间不能打开烤箱，也不能同时做多份砂锅，你可以从 0 时刻开始做砂锅。

m 次询问，每次给定一个 d，求所有顾客等待时间之和的最小值。

- $n, m \leq 200000$。

Zapiekanki 2

n 个顾客按时间顺序依次点餐，每个顾客会恰好点一份砂锅。第 i 个顾客点餐时间为 t_i，而且只接受 $\geq t_i$ 时刻之后出锅的砂锅。

你只有一个烤箱，做一份砂锅需要 d 时间，且中间不能打开烤箱，也不能同时做多份砂锅，你可以从 0 时刻开始做砂锅。

m 次询问，每次给定一个 d，求所有顾客等待时间之和的最小值。

- $n, m \leq 200000$。
- Source：PA 2017

Solution

- 分析问题的本质：最小化所有顾客拿到砂锅的时间之和。

Solution

- 分析问题的本质：最小化所有顾客拿到砂锅的时间之和。
- 设 f_i 表示顾客 i 拿到砂锅的时间。

Solution

- 分析问题的本质：最小化所有顾客拿到砂锅的时间之和。
- 设 f_i 表示顾客 i 拿到砂锅的时间。
- $f_0 = 0$。

Solution

- 分析问题的本质：最小化所有顾客拿到砂锅的时间之和。
- 设 f_i 表示顾客 i 拿到砂锅的时间。
- $f_0 = 0$。
- $f_i = \max(f_{i-1} + d, t_i)$。

Solution

- 分析问题的本质：最小化所有顾客拿到砂锅的时间之和。
- 设 f_i 表示顾客 i 拿到砂锅的时间。
- $f_0 = 0$。
- $f_i = \max(f_{i-1} + d, t_i)$。
- 时间复杂度 $O(nm)$，不能接受。

Solution

- 寻找不变量。

Solution

- 寻找不变量。
- 随着 d 从小到大逐渐增加，f 只会变化 $O(n)$ 次。

Solution

- 寻找不变量。
- 随着 d 从小到大逐渐增加，f 只会变化 $O(n)$ 次。
- 寻找变化点。

Solution

- 寻找不变量。
- 随着 d 从小到大逐渐增加，f 只会变化 $O(n)$ 次。
- 寻找变化点。
- 当 $t_i - f_{i-1} < d$ 时，f_i 会从 t_i 变为 $f_{i-1} + d$。

Solution

- 寻找不变量。
- 随着 d 从小到大逐渐增加，f 只会变化 $O(n)$ 次。
- 寻找变化点。
- 当 $t_i - f_{i-1} < d$ 时，f_i 会从 t_i 变为 $f_{i-1} + d$。
- f 可以表示成若干段公差为 d 的等差数列。

Solution

- 寻找不变量。
- 随着 d 从小到大逐渐增加，f 只会变化 $O(n)$ 次。
- 寻找变化点。
- 当 $t_i - f_{i-1} < d$ 时，f_i 会从 t_i 变为 $f_{i-1} + d$。
- f 可以表示成若干段公差为 d 的等差数列。
- 链表维护每一段的开头，堆维护变化点。

Solution

- 寻找不变量。
- 随着 d 从小到大逐渐增加，f 只会变化 $O(n)$ 次。
- 寻找变化点。
- 当 $t_i - f_{i-1} < d$ 时，f_i 会从 t_i 变为 $f_{i-1} + d$。
- f 可以表示成若干段公差为 d 的等差数列。
- 链表维护每一段的开头，堆维护变化点。
- 时间复杂度 $O(n \log n)$。

Greenhouse Growth

维护一个序列 $h_1, h_2, ..., h_n$ ，两种操作：

Greenhouse Growth

维护一个序列 $h_1, h_2, ..., h_n$ ，两种操作：

A：从 2 到 n 依次考虑每个位置 i，若 $h_i < h_{i-1}$ ，则将 h_i 增加 1。

Greenhouse Growth

维护一个序列 $h_1, h_2, \ldots, h_n$，两种操作：

A：从 2 到 n 依次考虑每个位置 i，若 $h_i < h_{i-1}$，则将 h_i 增加 1。

B：从 $n-1$ 到 1 依次考虑每个位置 i，若 $h_i < h_{i+1}$，则将 h_i 增加 1。

Greenhouse Growth

维护一个序列 $h_1, h_2, ..., h_n$ ，两种操作：

A：从 2 到 n 依次考虑每个位置 i，若 $h_i < h_{i-1}$ ，则将 h_i 增加 1。

B：从 $n-1$ 到 1 依次考虑每个位置 i，若 $h_i < h_{i+1}$ ，则将 h_i 增加 1。

求 m 次操作结束后序列每一项的值。

Greenhouse Growth

维护一个序列 $h_1, h_2, ..., h_n$，两种操作：

A：从 2 到 n 依次考虑每个位置 i，若 $h_i < h_{i-1}$，则将 h_i 增加 1。

B：从 $n-1$ 到 1 依次考虑每个位置 i，若 $h_i < h_{i+1}$，则将 h_i 增加 1。

求 m 次操作结束后序列每一项的值。

- $n, m \leq 300000$。

Greenhouse Growth

维护一个序列 $h_1, h_2, ..., h_n$ ，两种操作：

A：从 2 到 n 依次考虑每个位置 i，若 $h_i < h_{i-1}$ ，则将 h_i 增加 1。

B：从 $n-1$ 到 1 依次考虑每个位置 i，若 $h_i < h_{i+1}$ ，则将 h_i 增加 1。

求 m 次操作结束后序列每一项的值。

- $n, m \leq 300000$。
- Source：CERC 2015

Solution

- 寻找不变量。

Solution

- 寻找不变量。
- 如果连续一段值相同，那么操作后值仍然相同。

Solution

- 寻找不变量。
- 如果连续一段值相同，那么操作后值仍然相同。
- 基于段做处理。

Solution

- 寻找不变量。
- 如果连续一段值相同，那么操作后值仍然相同。
- 基于段做处理。
- 根据与左右两段的大小关系，可以得出 A 或者 B 操作是否会使这一段增加 1。

Solution

- 寻找不变量。
- 如果连续一段值相同，那么操作后值仍然相同。
- 基于段做处理。
- 根据与左右两段的大小关系，可以得出 A 或者 B 操作是否会使这一段增加 1。
- 若有 x 次 A 操作，y 次 B 操作，则这一段增量为 $x \times isA + y \times isB$。

Solution

- 寻找不变量。
- 如果连续一段值相同，那么操作后值仍然相同。
- 基于段做处理。
- 根据与左右两段的大小关系，可以得出 A 或者 B 操作是否会使这一段增加 1。
- 若有 x 次 A 操作，y 次 B 操作，则这一段增量为 $x \times isA + y \times isB$。
- 只关心左右两段，故链表维护每一段。

Solution

- 寻找变化点。

Solution

- 寻找变化点。
- 当两段高度相同时会发生合并。

Solution

- 寻找变化点。
- 当两段高度相同时会发生合并。
- 可以根据高度差直接算出在第几次操作时会合并。

Solution

- 寻找变化点。
- 当两段高度相同时会发生合并。
- 可以根据高度差直接算出在第几次操作时会合并。
- 合并时重新计算左右两段的情况，并产生新的合并事件。

Solution

- 寻找变化点。
- 当两段高度相同时会发生合并。
- 可以根据高度差直接算出在第几次操作时会合并。
- 合并时重新计算左右两段的情况，并产生新的合并事件。
- 开桶按时间维护所有可能的合并事件。

Solution

- 寻找变化点。
- 当两段高度相同时会发生合并。
- 可以根据高度差直接算出在第几次操作时会合并。
- 合并时重新计算左右两段的情况，并产生新的合并事件。
- 开桶按时间维护所有可能的合并事件。
- 时间复杂度 $O(n+m)$。

Range Maximum Query

给定一个序列 $a_1, a_2, ..., a_n$ ，m 次询问一个区间 $[l, r]$ 的区间最大值。

Range Maximum Query

给定一个序列 $a_1, a_2, ..., a_n$ ， m 次询问一个区间 $[l, r]$ 的区间最大值。

- $n, m \leq 10^7$。

Solution

- 从 1 到 n 考虑每个右端点为 r 的询问。

Solution

- 从 1 到 n 考虑每个右端点为 r 的询问。
- 初始时 n 个位置独立。

Solution

- 从 1 到 n 考虑每个右端点为 r 的询问。
- 初始时 n 个位置独立。
- 到 r 时将 r 的父亲指向 $r+1$，边权为 a_r。

Solution

- 从 1 到 n 考虑每个右端点为 r 的询问。
- 初始时 n 个位置独立。
- 到 r 时将 r 的父亲指向 $r+1$，边权为 a_r。
- 则询问 $[l, r]$ 的答案为 l 到根路径上的边权最大值。

Solution

- 从 1 到 n 考虑每个右端点为 r 的询问。
- 初始时 n 个位置独立。
- 到 r 时将 r 的父亲指向 $r+1$，边权为 a_r。
- 则询问 $[l, r]$ 的答案为 l 到根路径上的边权最大值。
- 并查集路径压缩即可。

Solution

- 从 1 到 n 考虑每个右端点为 r 的询问。
- 初始时 n 个位置独立。
- 到 r 时将 r 的父亲指向 $r+1$，边权为 a_r。
- 则询问 $[l,r]$ 的答案为 l 到根路径上的边权最大值。
- 并查集路径压缩即可。
- 时间复杂度 $O(n\alpha(n))$。

区间赋值

给定一个序列 $a_1, a_2, \ldots, a_n$ ，m 次操作。

区间赋值

给定一个序列 $a_1, a_2, ..., a_n$ ，m 次操作。

每次指定 l, r, k ，将 $[l, r]$ 每个数都赋值为 k。

区间赋值

给定一个序列 $a_1, a_2, \ldots, a_n$ ， m 次操作。

每次指定 l, r, k ，将 $[l, r]$ 每个数都赋值为 k。

输出最终的序列。

区间赋值

给定一个序列 $a_1, a_2, ..., a_n$ ，m 次操作。
每次指定 l, r, k ，将 $[l, r]$ 每个数都赋值为 k。
输出最终的序列。

- $n, m \leq 10^7$。

Solution

- 时间倒流法。

Solution

- 时间倒流法。
- 倒着处理每个操作，那么之前赋值过的位置不能再被赋值。

Solution

- 时间倒流法。
- 倒着处理每个操作，那么之前赋值过的位置不能再被赋值。
- 设 f_i 表示 i 右侧第一个未被赋值的位置。

Solution

- 时间倒流法。
- 倒着处理每个操作，那么之前赋值过的位置不能再被赋值。
- 设 f_i 表示 i 右侧第一个未被赋值的位置。
- 每次沿着 f 遍历 $[l, r]$ 中所有未赋值的位置，同时将 f_i 指向 f_{i+1}。

Solution

- 时间倒流法。
- 倒着处理每个操作，那么之前赋值过的位置不能再被赋值。
- 设 f_i 表示 i 右侧第一个未被赋值的位置。
- 每次沿着 f 遍历 $[l, r]$ 中所有未赋值的位置，同时将 f_i 指向 f_{i+1}。
- 并查集路径压缩即可。

Solution

- 时间倒流法。
- 倒着处理每个操作，那么之前赋值过的位置不能再被赋值。
- 设 f_i 表示 i 右侧第一个未被赋值的位置。
- 每次沿着 f 遍历 $[l, r]$ 中所有未赋值的位置，同时将 f_i 指向 f_{i+1}。
- 并查集路径压缩即可。
- 时间复杂度 $O(n\alpha(n))$。

Buffalo Barricades

第一象限中有 n 个点，第 i 个点位于 $(x_i - 0.5, y_i - 0.5)$。

Buffalo Barricades

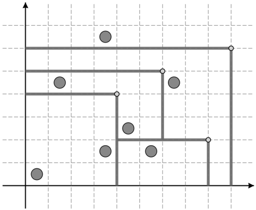
第一象限中有 n 个点，第 i 个点位于 $(x_i - 0.5, y_i - 0.5)$。

依次放入 m 个篱笆，每个篱笆是从 $P_i(x_i, y_i)$ 点往左往下发射两条射线，直到碰到另一个篱笆或者坐标轴。

Buffalo Barricades

第一象限中有 n 个点，第 i 个点位于 $(x_i - 0.5, y_i - 0.5)$。

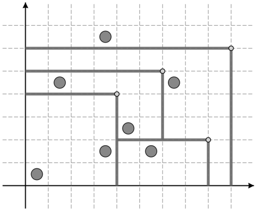
依次放入 m 个篱笆，每个篱笆是从 $P_i(x_i, y_i)$ 点往左往下发射两条射线，直到碰到另一个篱笆或者坐标轴。



Buffalo Barricades

第一象限中有 n 个点，第 i 个点位于 $(x_i - 0.5, y_i - 0.5)$。

依次放入 m 个篱笆，每个篱笆是从 $P_i(x_i, y_i)$ 点往左往下发射两条射线，直到碰到另一个篱笆或者坐标轴。

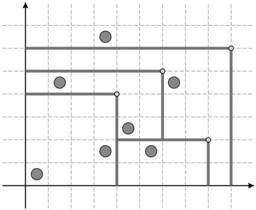


问每次新加入的篱笆划分出的区域内部有多少个点。

Buffalo Barricades

第一象限中有 n 个点，第 i 个点位于 $(x_i - 0.5, y_i - 0.5)$。

依次放入 m 个篱笆，每个篱笆是从 $P_i(x_i, y_i)$ 点往左往下发射两条射线，直到碰到另一个篱笆或者坐标轴。



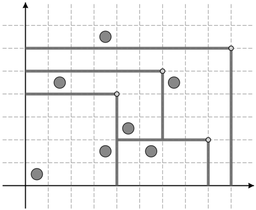
问每次新加入的篱笆划分出的区域内部有多少个点。

- $n, m \leq 300000$。

Buffalo Barricades

第一象限中有 n 个点，第 i 个点位于 $(x_i - 0.5, y_i - 0.5)$。

依次放入 m 个篱笆，每个篱笆是从 $P_i(x_i, y_i)$ 点往左往下发射两条射线，直到碰到另一个篱笆或者坐标轴。



问每次新加入的篱笆划分出的区域内部有多少个点。

- $n, m \leq 300000$。
- Source：CERC 2017

Solution

- 时间倒流法。

Solution

- 时间倒流法。
- 在最终图案中有若干个封闭区域，每个点只属于一个区域。

Solution

- 时间倒流法。
- 在最终图案中有若干个封闭区域，每个点只属于一个区域。
- 从后往前依次拿走每个篱笆，合并相邻的区域。

Solution

- 时间倒流法。
- 在最终图案中有若干个封闭区域，每个点只属于一个区域。
- 从后往前依次拿走每个篱笆，合并相邻的区域。
- 并查集维护每个连通块的点数。

Solution

- 如何求每个点属于哪个区域？

Solution

- 如何求每个点属于哪个区域？
- 从上到下考虑每个事件：点和篱笆。

Solution

- 如何求每个点属于哪个区域？
- 从上到下考虑每个事件：点和篱笆。
- 用 std::set 按从左往右的顺序维护还在往下继续延伸的所有篱笆。

Solution

- 如何求每个点属于哪个区域？
- 从上到下考虑每个事件：点和篱笆。
- 用 std::set 按从左往右的顺序维护还在往下继续延伸的所有篱笆。
- 对于一个点，在 std::set 中查找后继，它就属于后继篱笆所在的区域。

Solution

- 如何求每个点属于哪个区域？
- 从上到下考虑每个事件：点和篱笆。
- 用 std::set 按从左往右的顺序维护还在往下继续延伸的所有篱笆。
- 对于一个点，在 std::set 中查找后继，它就属于后继篱笆所在的区域。
- 对于一个篱笆，将其加入 std::set，不断终结比它晚出现的前驱篱笆。

Solution

- 如何求每个点属于哪个区域？
- 从上到下考虑每个事件：点和篱笆。
- 用 std::set 按从左往右的顺序维护还在往下继续延伸的所有篱笆。
- 对于一个点，在 std::set 中查找后继，它就属于后继篱笆所在的区域。
- 对于一个篱笆，将其加入 std::set，不断终结比它晚出现的前驱篱笆。
- 时间复杂度 $O(n \log n)$。

题目提交

课上例题：

http://acm.hdu.edu.cn/diy/contest_show.php?cid=33143

课后习题：

http://acm.hdu.edu.cn/diy/contest_show.php?cid=33144

密码：

G*&GSF&*t387tr

Thank you!