数据结构

mcfx

2019 年 7 月 4 日

CF 793 F

Julia the snail

有一个长为 n 的杆，上面有 m 条绳子，每条绳子可以让你从 l_i 爬到 r_i，保证 r_i 各不相同。可以自然下落。
每次询问 x 出发，途中高度不能低于 x 或高于 y，问最高能到多高。
$n, m, q \leq 10^5$。

Julia the snail

有一个长为 n 的杆，上面有 m 条绳子，每条绳子可以让你从 l_i 爬到 r_i，保证 r_i 各不相同。可以自然下落。
每次询问 x 出发，途中高度不能低于 x 或高于 y，问最高能到多高。
$n, m, q \leq 10^5$。

一个做法是参考 CF 题解分块。

Julia the snail

有一个长为 n 的杆，上面有 m 条绳子，每条绳子可以让你从 l_i 爬到 r_i，保证 r_i 各不相同。可以自然下落。
每次询问 x 出发，途中高度不能低于 x 或高于 y，问最高能到多高。
$n, m, q \leq 10^5$。

一个做法是参考 CF 题解分块。
一个做法是，扫描线。把绳子和询问挂到 l 或 x。

CF 793 F

Julia the snail

有一个长为 n 的杆，上面有 m 条绳子，每条绳子可以让你从 l_i 爬到 r_i，保证 r_i 各不相同。可以自然下落。
每次询问 x 出发，途中高度不能低于 x 或高于 y，问最高能到多高。
$n, m, q \leq 10^5$。

一个做法是参考 CF 题解分块。
一个做法是，扫描线。把绳子和询问挂到 l 或 x。
从高到低枚举绳子，然后线段树维护要到 i 至少需要多少 y。

CF 793 F

Julia the snail

有一个长为 n 的杆，上面有 m 条绳子，每条绳子可以让你从 l_i 爬到 r_i，保证 r_i 各不相同。可以自然下落。
每次询问 x 出发，途中高度不能低于 x 或高于 y，问最高能到多高。
$n, m, q \leq 10^5$。

一个做法是参考 CF 题解分块。
一个做法是，扫描线。把绳子和询问挂到 l 或 x。
从高到低枚举绳子，然后线段树维护要到 i 至少需要多少 y。
绳子会在左端点被加入。加入一条绳子 l, r 时，可以把 $[l, r]$ 的上述值和 r 取 min。

Julia the snail

有一个长为 n 的杆，上面有 m 条绳子，每条绳子可以让你从 l_i 爬到 r_i，保证 r_i 各不相同。可以自然下落。
每次询问 x 出发，途中高度不能低于 x 或高于 y，问最高能到多高。
$n, m, q \le 10^5$。

一个做法是参考 CF 题解分块。
一个做法是，扫描线。把绳子和询问挂到 l 或 x。
从高到低枚举绳子，然后线段树维护要到 i 至少需要多少 y。
绳子会在左端点被加入。加入一条绳子 l, r 时，可以把 $[l, r]$ 的上述值和 r 取 min。
查询时找到第一个上述值超过 y 的点。

还有一个我似乎不会证复杂度的做法:

CF 793 F

还有一个我似乎不会证复杂度的做法：
离线，枚举当前最高位置，对每个位置维护能到的最高位置 f_i。

还有一个我似乎不会证复杂度的做法：
离线，枚举当前最高位置，对每个位置维护能到的最高位置 f_i。
每次要把 $f_i \geq l_i$ 的 f_i 全部改成 r_i。

CF 793 F

还有一个我似乎不会证复杂度的做法：
离线，枚举当前最高位置，对每个位置维护能到的最高位置 f_i。
每次要把 $f_i \geq l_i$ 的 f_i 全部改成 r_i。
显然是 $O(n^2)$ 的。

还有一个我似乎不会证复杂度的做法：
离线，枚举当前最高位置，对每个位置维护能到的最高位置 f_i。
每次要把 $f_i \geq l_i$ 的 f_i 全部改成 r_i。
显然是 $O(n^2)$ 的。
考虑用线段树优化，如果当前区间的 max 小于 l_i，那么可以退出。

还有一个我似乎不会证复杂度的做法：
离线，枚举当前最高位置，对每个位置维护能到的最高位置 f_i。
每次要把 $f_i \geq l_i$ 的 f_i 全部改成 r_i。
显然是 $O(n^2)$ 的。
考虑用线段树优化，如果当前区间的 max 小于 l_i，那么可以退出。
现在复杂度是 $O(n^2 \log n)$。

考虑给线段树加一个次大值。

考虑给线段树加一个次大值。
如果当前区间只有最大值需要改，那么可以直接打标记。

考虑给线段树加一个次大值。
如果当前区间只有最大值需要改，那么可以直接打标记。
否则递归下去。

CF 793 F

考虑给线段树加一个次大值。
如果当前区间只有最大值需要改，那么可以直接打标记。
否则递归下去。
现在复杂度是 $O(n\log n)$ 了。

考虑给线段树加一个次大值。
如果当前区间只有最大值需要改，那么可以直接打标记。
否则递归下去。
现在复杂度是 $O(n\log n)$ 了。
可能需要奇怪的方法证明。

考虑给线段树加一个次大值。
如果当前区间只有最大值需要改，那么可以直接打标记。
否则递归下去。
现在复杂度是 $O(n \log n)$ 了。
可能需要奇怪的方法证明。
你们可以感性理解一下。

圣诞树

给一棵树，每次在一条链 (u_i, v_i) 上的每个点上挂上 a_i 个种类为 b_i 的物品。
一个点的 k-美观度这样计算：把这个点上的所有种类的礼物按照个数从小到大排序，如果个数一样就按照种类从小到大排。它的 k-美观度就是排好序后前 k 种礼物种类的 xor 值（如果礼物种类不足 k 种，就把这个点上所有礼物的种类 xor 起来）。
给 Q 个询问，给定 w_i, k_i，求点 w_i 的 k_i-美观度。

圣诞树

给一棵树，每次在一条链 (u_i, v_i) 上的每个点上挂上 a_i 个种类为 b_i 的物品。
一个点的 k-美观度这样计算：把这个点上的所有种类的礼物按照个数从小到大排序，如果个数一样就按照种类从小到大排。它的 k-美观度就是排好序后前 k 种礼物种类的 xor 值（如果礼物种类不足 k 种，就把这个点上所有礼物的种类 xor 起来）。
给 Q 个询问，给定 w_i, k_i，求点 w_i 的 k_i-美观度。

原问题等价于在两个端点处挂上 a_i 个 b_i，在端点 lca 及 lca 的父亲处挂上 $-a_i$ 个 b_i，询问子树的 k-美观度。

圣诞树

给一棵树，每次在一条链 (u_i, v_i) 上的每个点上挂上 a_i 个种类为 b_i 的物品。
一个点的 k-美观度这样计算：把这个点上的所有种类的礼物按照个数从小到大排序，如果个数一样就按照种类从小到大排。它的 k-美观度就是排好序后前 k 种礼物种类的 xor 值（如果礼物种类不足 k 种，就把这个点上所有礼物的种类 xor 起来）。
给 Q 个询问，给定 w_i, k_i，求点 w_i 的 k_i-美观度。

原问题等价于在两个端点处挂上 a_i 个 b_i，在端点 lca 及 lca 的父亲处挂上 $-a_i$ 个 b_i，询问子树的 k-美观度。
做法一：每个点开一棵平衡树对每种物品排序，处理完儿子后进行启发式合并得到这个点的平衡树，合并时需要记一个索引，表示这种物品对应平衡树中的哪个点。这个索引用平衡树或者线段树维护，更新索引时进行平衡树启发式合并或者线段树合并，复杂度 $O(n\log^2 n)$，常数较大，可能需要卡常数才能通过。

51nod 1782

做法二：把前一个做法的启发式合并改成 dsu on tree，复杂度相同，常数小一些。

51nod 1782

做法二：把前一个做法的启发式合并改成 dsu on tree，复杂度相同，常数小一些。
做法三：对 dfs 序分治，处理跨过中间那条线的区间。因为 dfs 序的区间都是相互包含的，只要从中间往外面做就行了。同样用平衡树维护，复杂度 $O(n\log^2 n)$。

Powerful Pair in Tree

给一棵树，每次给两个点 x, y，问在 x 子树里选一个点，y 子树里选一个点，有多少种方案使得他们的异或为 2 的若干次方。
$n \leq 10^5$，时限 8s。

Powerful Pair in Tree

给一棵树，每次给两个点 x, y，问在 x 子树里选一个点，y 子树里选一个点，有多少种方案使得他们的异或为 2 的若干次方。
$n \le 10^5$，时限 8s。

先转化成 dfs 序上的区间。

Powerful Pair in Tree

给一棵树，每次给两个点 x, y，问在 x 子树里选一个点，y 子树里选一个点，有多少种方案使得他们的异或为 2 的若干次方。
$n \leq 10^5$，时限 8s。

先转化成 dfs 序上的区间。
假设询问是 $[a, b]$ 和 $[c, d]$，$a \leq c$。

Powerful Pair in Tree

给一棵树，每次给两个点 x, y，问在 x 子树里选一个点，y 子树里选一个点，有多少种方案使得他们的异或为 2 的若干次方。
$n \leq 10^5$，时限 8s。

先转化成 dfs 序上的区间。
假设询问是 $[a, b]$ 和 $[c, d]$，$a \leq c$。
设 $f(l, r)$ 表示在区间 $[l, r]$ 中选两个点，他们的异或为 2 的若干次方的方案数。

Powerful Pair in Tree

给一棵树，每次给两个点 x, y，问在 x 子树里选一个点，y 子树里选一个点，有多少种方案使得他们的异或为 2 的若干次方。
$n \leq 10^5$，时限 8s。

先转化成 dfs 序上的区间。
假设询问是 $[a, b]$ 和 $[c, d]$，$a \leq c$。
设 $f(l, r)$ 表示在区间 $[l, r]$ 中选两个点，他们的异或为 2 的若干次方的方案数。
如果 $a \leq b < c \leq d$，那么答案是
$f(a, d) + f(b+1, c-1) - f(a, c-1) - f(b+1, d)$。

Powerful Pair in Tree

给一棵树，每次给两个点 x, y，问在 x 子树里选一个点，y 子树里选一个点，有多少种方案使得他们的异或为 2 的若干次方。
$n \leq 10^5$，时限 8s。

先转化成 dfs 序上的区间。
假设询问是 $[a, b]$ 和 $[c, d]$，$a \leq c$。
设 $f(l, r)$ 表示在区间 $[l, r]$ 中选两个点，他们的异或为 2 的若干次方的方案数。
如果 $a \leq b < c \leq d$，那么答案是
$f(a, d) + f(b+1, c-1) - f(a, c-1) - f(b+1, d)$。
如果 $a \leq c \leq b \leq d$，那么答案是 $f(a, d) - f(a, c-1) - f(b+1, d)$。

Powerful Pair in Tree

给一棵树，每次给两个点 x, y，问在 x 子树里选一个点，y 子树里选一个点，有多少种方案使得他们的异或为 2 的若干次方。
$n \leq 10^5$，时限 8s。

先转化成 dfs 序上的区间。
假设询问是 $[a, b]$ 和 $[c, d]$，$a \leq c$。
设 $f(l, r)$ 表示在区间 $[l, r]$ 中选两个点，他们的异或为 2 的若干次方的方案数。
如果 $a \leq b < c \leq d$，那么答案是
$f(a, d) + f(b + 1, c - 1) - f(a, c - 1) - f(b + 1, d)$。
如果 $a \leq c \leq b \leq d$，那么答案是 $f(a, d) - f(a, c - 1) - f(b + 1, d)$。
然后使用莫队就能求出这些新询问的答案。

Alex and a TV Show

维护 n 个长为 7000 的 bitset，支持把某个 bitset 设为只有一个 1，xor 两个 bitset，询问某个 *bitset* 的某一位，以及

```
x.clear();for(i)for(j)x[gcd(i,j)]^=y[i]&z[j];
```

$n \leq 10^5, m \leq 10^6$

CF 1097 F

Alex and a TV Show

维护 n 个长为 7000 的 bitset，支持把某个 bitset 设为只有一个 1，xor 两个 bitset，询问某个 $bitset$ 的某一位，以及

```
x.clear();for(i)for(j)x[gcd(i,j)]^=y[i]&z[j];
```

$n \le 10^5, m \le 10^6$

如果把一个 bitset 里位置 i 变成 i 的倍数的异或，那么 xor 操作仍然是 xor，而这个奇怪的操作会变成直接 and。
对于设为 1 的操作，可以预处理出一堆 $bitset$。
对于求某一位的操作，可以莫比乌斯反演，发现只需要把 $j \bmod i = 0$ 且 $\mu(\frac{j}{i}) \neq 0$ 的 j 的值异或起来就是答案。同样预处理 bitset 即可。

Paired Parentheses

给两个长为 $2n$ 的数组 a 和 b，每次修改某个位置的两个值，并询问，如果构造两个长为 $2n$ 的括号串，对于位置 i，如果对应括号相同，$ans+=a_i$，否则 $ans+=b_i$，求最大 ans。

$n,m \leq 1e5, |a_i|,|b_i| \leq 10^9$

ATC CF17 Tournament Round 1 A

Paired Parentheses

给两个长为 $2n$ 的数组 a 和 b，每次修改某个位置的两个值，并询问，如果构造两个长为 $2n$ 的括号串，对于位置 i，如果对应括号相同，$ans+=a_i$，否则 $ans+=b_i$，求最大 ans。
$n, m \leq 1e5, |a_i|, |b_i| \leq 10^9$

显然，首尾只能取 a_i。

Paired Parentheses

给两个长为 $2n$ 的数组 a 和 b，每次修改某个位置的两个值，并询问，如果构造两个长为 $2n$ 的括号串，对于位置 i，如果对应括号相同，$ans+=a_i$，否则 $ans+=b_i$，求最大 ans。
$n, m \leq 1e5, |a_i|, |b_i| \leq 10^9$

显然，首尾只能取 a_i。
中间必须选偶数个 a_i。必要性显然。

Paired Parentheses

给两个长为 $2n$ 的数组 a 和 b，每次修改某个位置的两个值，并询问，如果构造两个长为 $2n$ 的括号串，对于位置 i，如果对应括号相同，$ans+=a_i$，否则 $ans+=b_i$，求最大 ans。
$n,m \le 1e5, |a_i|, |b_i| \le 10^9$

显然，首尾只能取 a_i。
中间必须选偶数个 a_i。必要性显然。
充分性的话，可以考虑前半部分 a_i 均为 (，后半部分均为)，两个括号序列的 b_i 均为左右交替，显然满足条件。

Paired Parentheses

给两个长为 $2n$ 的数组 a 和 b，每次修改某个位置的两个值，并询问，如果构造两个长为 $2n$ 的括号串，对于位置 i，如果对应括号相同，$ans+=a_i$，否则 $ans+=b_i$，求最大 ans。
$n, m \le 1e5, |a_i|, |b_i| \le 10^9$

显然，首尾只能取 a_i。
中间必须选偶数个 a_i。必要性显然。
充分性的话，可以考虑前半部分 a_i 均为 (，后半部分均为)，两个括号序列的 b_i 均为左右交替，显然满足条件。
线段树维护即可。

Chef at the River

大厨和 N 只动物（编号为 $1 \sim N$）需要过河。他们都在河的左侧，需要去往河的右侧。一些动物相处不来，因此这些动物不能在无人看管的情况下处于同一侧河岸。
我们用一棵 N 个节点的树描述动物之间的关系。如果节点 u 和 v 之间有边，则意味着动物 u 和 v 无法和平相处。
大厨和动物们只有一艘船能过河。大厨可以划船过河载着动物过河。当大厨在船上时，除了船上之外的所有动物都是无人看管的。船的载重量代表船在渡河时可以承载的动物与人的数量。假如大厨带一只动物过河，那么船的载重量应当至少为 2。
初始时，树仅包含 1 号节点。之后，节点 $2, 3, \ldots, N$ 依次被加入这棵树。加入节点时，该节点会与一个已加入的节点（其父节点）相连。
每次加入节点后，你需要计算：要使得大厨和所有动物顺利渡河，船的载重量最少需要是多少。

下面答案都是指能载的动物数量。

下面答案都是指能载的动物数量。
答案显然不低于最小点覆盖，而也不高于最小点覆盖 $+1$。

下面答案都是指能载的动物数量。
答案显然不低于最小点覆盖，而也不高于最小点覆盖 $+1$。
最小点覆盖 $+1$ 的做法是把最小点覆盖留在船上，然后每次运一个其他动物过去。

下面答案都是指能载的动物数量。
答案显然不低于最小点覆盖，而也不高于最小点覆盖 $+1$。
最小点覆盖 $+1$ 的做法是把最小点覆盖留在船上，然后每次运一个其他动物过去。
对于 $N \geq 4$ 且不是菊花图，可以做到最小点覆盖。

下面答案都是指能载的动物数量。
答案显然不低于最小点覆盖，而也不高于最小点覆盖 $+1$。
最小点覆盖 $+1$ 的做法是把最小点覆盖留在船上，然后每次运一个其他动物过去。
对于 $N \geq 4$ 且不是菊花图，可以做到最小点覆盖。
可以在最小点覆盖中选出两个点 x, y，先把 x 运过去，然后用上面的办法把不和 x 相连的点运过去。

下面答案都是指能载的动物数量。
答案显然不低于最小点覆盖，而也不高于最小点覆盖 $+1$。
最小点覆盖 $+1$ 的做法是把最小点覆盖留在船上，然后每次运一个其他动物过去。
对于 $N \geq 4$ 且不是菊花图，可以做到最小点覆盖。
可以在最小点覆盖中选出两个点 x, y，先把 x 运过去，然后用上面的办法把不和 x 相连的点运过去。
接下来把 x 放回船上，然后把 y 放到原来的河岸。x 和 y 最多只有一个公共点，所以可以先把这个点运过去，然后再依次把其他和 x 相连的点运过去。

下面答案都是指能载的动物数量。
答案显然不低于最小点覆盖，而也不高于最小点覆盖 $+1$。
最小点覆盖 $+1$ 的做法是把最小点覆盖留在船上，然后每次运一个其他动物过去。
对于 $N \geq 4$ 且不是菊花图，可以做到最小点覆盖。
可以在最小点覆盖中选出两个点 x, y，先把 x 运过去，然后用上面的办法把不和 x 相连的点运过去。
接下来把 x 放回船上，然后把 y 放到原来的河岸。x 和 y 最多只有一个公共点，所以可以先把这个点运过去，然后再依次把其他和 x 相连的点运过去。
菊花图和 $N \leq 3$ 的做法比较显然。

下面答案都是指能载的动物数量。
答案显然不低于最小点覆盖，而也不高于最小点覆盖 $+1$。
最小点覆盖 $+1$ 的做法是把最小点覆盖留在船上，然后每次运一个其他动物过去。
对于 $N \geq 4$ 且不是菊花图，可以做到最小点覆盖。
可以在最小点覆盖中选出两个点 x, y，先把 x 运过去，然后用上面的办法把不和 x 相连的点运过去。
接下来把 x 放回船上，然后把 y 放到原来的河岸。x 和 y 最多只有一个公共点，所以可以先把这个点运过去，然后再依次把其他和 x 相连的点运过去。
菊花图和 $N \leq 3$ 的做法比较显然。
那么只需要动态维护最小点覆盖，可以动态 dp 维护。

UOJ 309

排兵布阵

平面上 n 个点，支持：横坐标等于 x 的点横坐标 $+= d$，值 $+= p$ 或求最大值，纵坐标等于 y 的点纵坐标 $+= d$，值 $+= p$ 或求最大值。

UOJ 309

排兵布阵

平面上 n 个点，支持：横坐标等于 x 的点横坐标 $+=d$，值 $+=p$ 或求最大值，纵坐标等于 y 的点纵坐标 $+=d$，值 $+=p$ 或求最大值。

考虑这么一个数据结构：我们对平面里的每一行统计这一行的点数 nx_i，每一列统计这一列的点数 ny_i。对于一个点 (x,y)，如果 $nx_x \geq ny_y$，那么它就作为 y 列的关键点，否则就作为 x 行的关键点。

排兵布阵

平面上 n 个点，支持：横坐标等于 x 的点横坐标 $+= d$，值 $+= p$ 或求最大值，纵坐标等于 y 的点纵坐标 $+= d$，值 $+= p$ 或求最大值。

考虑这么一个数据结构：我们对平面里的每一行统计这一行的点数 nx_i，每一列统计这一列的点数 ny_i。对于一个点 (x, y)，如果 $nx_x \geq ny_y$，那么它就作为 y 列的关键点，否则就作为 x 行的关键点。

不难发现每一行每一列的关键点个数都是严格 $O(\sqrt{n})$ 的。

排兵布阵

平面上 n 个点，支持：横坐标等于 x 的点横坐标 $+= d$，值 $+= p$ 或求最大值，纵坐标等于 y 的点纵坐标 $+= d$，值 $+= p$ 或求最大值。

考虑这么一个数据结构：我们对平面里的每一行统计这一行的点数 nx_i，每一列统计这一列的点数 ny_i。对于一个点 (x, y)，如果 $nx_x \geq ny_y$，那么它就作为 y 列的关键点，否则就作为 x 行的关键点。
不难发现每一行每一列的关键点个数都是严格 $O(\sqrt{n})$ 的。
于是思路就是每一个操作，都是关键点上暴力，非关键点打标记，同时对每一行每一列维护非关键点的权值最大值。这样每一次操作都是严格 $O(\sqrt{n})$ 的。

这个问题的核心在于把横纵的修改和询问联系起来。在刚才那个结构中，可以定义一个行关键点 (x, y) 的权值为它的点权加上列 y 上的非关键点加标记。

这个问题的核心在于把横纵的修改和询问联系起来。在刚才那个结构中，可以定义一个行关键点 (x, y) 的权值为它的点权加上列 y 上的非关键点加标记。
这样在行操作的时候，我们把每一个行关键点的点权修改一下，同时更新这个关键点所在列的非关键点最大值。列操作类似。

在合并两个有标记的行的时候，因为标记是不能下传的（下传要遍历每一个非关键点并修改，复杂度爆炸了）。所以可以对行和列分别维护一个带权并查集，在并查集的每一条边上维护儿子和父亲的标记差。

在合并两个有标记的行的时候，因为标记是不能下传的（下传要遍历每一个非关键点并修改，复杂度爆炸了）。所以可以对行和列分别维护一个带权并查集，在并查集的每一条边上维护儿子和父亲的标记差。
这样对于一个原来坐标是 (x, y) 的点，我们要看它列上的标记，只要在列并查集里求它到根的边权和就可以了。

在合并两个有标记的行的时候，因为标记是不能下传的（下传要遍历每一个非关键点并修改，复杂度爆炸了）。所以可以对行和列分别维护一个带权并查集，在并查集的每一条边上维护儿子和父亲的标记差。

这样对于一个原来坐标是 (x, y) 的点，我们要看它列上的标记，只要在列并查集里求它到根的边权和就可以了。

接着，因为在合并了两行之后，行的点数变多了，所以一些行关键点会变成列关键点。这个时候我们要遍历所有行关键点，然后看关键点是否会跑到另一边去。因为一次合并之后关键点也是 $O(\sqrt{n})$ 的，所以这一步也可以暴力。但是要注意的是行关键点变成列关键点后，这个点的点权要根据行列的标记修改一下以满足我们刚才的定义。

行列的关键点个数是 $O(\sqrt{n})$ 的分析建立在没有重点的基础上，但是合并的过程中会出现重点，这个时候我们需要对关键点去重。

行列的关键点个数是 $O(\sqrt{n})$ 的分析建立在没有重点的基础上，但是合并的过程中会出现重点，这个时候我们需要对关键点去重。
去重的过程相当于把所有相同位置的关键点变成一个点，且权值为这个位置所有关键点的最大值。

行列的关键点个数是 $O(\sqrt{n})$ 的分析建立在没有重点的基础上，但是合并的过程中会出现重点，这个时候我们需要对关键点去重。

去重的过程相当于把所有相同位置的关键点变成一个点，且权值为这个位置所有关键点的最大值。

如果每一次合并都去重的话，去重的复杂度在瓶颈上，必须要写 hash 之类的方法，常数较大。

行列的关键点个数是 $O(\sqrt{n})$ 的分析建立在没有重点的基础上，但是合并的过程中会出现重点，这个时候我们需要对关键点去重。
去重的过程相当于把所有相同位置的关键点变成一个点，且权值为这个位置所有关键点的最大值。
如果每一次合并都去重的话，去重的复杂度在瓶颈上，必须要写 hash 之类的方法，常数较大。
实际上可以设一个阈值 S，例如 $1.5\sqrt{n}$，然后在每一次我们要暴力扫关键点集合的时候，如果我们发现关键点数量超过了 S，那么就进行一次去重。

行列的关键点个数是 $O(\sqrt{n})$ 的分析建立在没有重点的基础上，但是合并的过程中会出现重点，这个时候我们需要对关键点去重。
去重的过程相当于把所有相同位置的关键点变成一个点，且权值为这个位置所有关键点的最大值。
如果每一次合并都去重的话，去重的复杂度在瓶颈上，必须要写 hash 之类的方法，常数较大。
实际上可以设一个阈值 S，例如 $1.5\sqrt{n}$，然后在每一次我们要暴力扫关键点集合的时候，如果我们发现关键点数量超过了 S，那么就进行一次去重。
因为每一次去重都至少删掉了 $O(\sqrt{n})$ 个点，而总点数只有 n，所以去重的复杂度不在瓶颈上，可以用 map 暴力。

行列的关键点个数是 $O(\sqrt{n})$ 的分析建立在没有重点的基础上，但是合并的过程中会出现重点，这个时候我们需要对关键点去重。
去重的过程相当于把所有相同位置的关键点变成一个点，且权值为这个位置所有关键点的最大值。
如果每一次合并都去重的话，去重的复杂度在瓶颈上，必须要写 hash 之类的方法，常数较大。
实际上可以设一个阈值 S，例如 $1.5\sqrt{n}$，然后在每一次我们要暴力扫关键点集合的时候，如果我们发现关键点数量超过了 S，那么就进行一次去重。
因为每一次去重都至少删掉了 $O(\sqrt{n})$ 个点，而总点数只有 n，所以去重的复杂度不在瓶颈上，可以用 map 暴力。
综上，这个算法的时间复杂度是 $O(\sqrt{n})$ 的，并查集、去重的时间复杂度都不是瓶颈。可以通过后两个数据集。

CC SAFPAR

Safe Partition

给一个长为 $n(n \le 5 \cdot 10^5)$ 的序列 A。对于一个区间 $[l, r]$，如果 $min(A_l, \dots, A_r) \le r - l + 1 \le max(A_l, \dots, A_r)$，那么这个区间是好的。

你需要求出有多少种方法把 A 划分为若干个区间，使得每个区间都是好的。

CC SAFPAR

Safe Partition

给一个长为 $n(n \le 5 \cdot 10^5)$ 的序列 A。对于一个区间 $[l, r]$，如果 $min(A_l, \ldots, A_r) \le r - l + 1 \le max(A_l, \ldots, A_r)$，那么这个区间是好的。

你需要求出有多少种方法把 A 划分为若干个区间，使得每个区间都是好的。

朴素的 $O(n^2)$ dp 是显然的。

Safe Partition

给一个长为 $n(n \leq 5 \cdot 10^5)$ 的序列 A。对于一个区间 $[l, r]$，如果 $min(A_l, \ldots, A_r) \leq r - l + 1 \leq max(A_l, \ldots, A_r)$，那么这个区间是好的。
你需要求出有多少种方法把 A 划分为若干个区间，使得每个区间都是好的。

朴素的 $O(n^2)$ dp 是显然的。
假设要计算 $dp_i = \sum dp_j$。

CC SAFPAR

Safe Partition

给一个长为 $n(n \leq 5 \cdot 10^5)$ 的序列 A。对于一个区间 $[l, r]$，如果 $min(A_l, \ldots, A_r) \leq r - l + 1 \leq max(A_l, \ldots, A_r)$，那么这个区间是好的。
你需要求出有多少种方法把 A 划分为若干个区间，使得每个区间都是好的。

朴素的 $O(n^2)$ dp 是显然的。
假设要计算 $dp_i = \sum dp_j$。
$min \leq len$ 显然在某个 j 之前一定满足。

Safe Partition

给一个长为 $n(n \le 5 \cdot 10^5)$ 的序列 A。对于一个区间 $[l, r]$，如果 $min(A_l, \dots, A_r) \le r - l + 1 \le max(A_l, \dots, A_r)$，那么这个区间是好的。
你需要求出有多少种方法把 A 划分为若干个区间，使得每个区间都是好的。

朴素的 $O(n^2)$ dp 是显然的。
假设要计算 $dp_i = \sum dp_j$。
$min \le len$ 显然在某个 j 之前一定满足。
那么只需要考虑 $len \le max$ 的限制。

Safe Partition

给一个长为 $n(n \leq 5 \cdot 10^5)$ 的序列 A。对于一个区间 $[l, r]$，如果 $min(A_l, \ldots, A_r) \leq r - l + 1 \leq max(A_l, \ldots, A_r)$，那么这个区间是好的。
你需要求出有多少种方法把 A 划分为若干个区间，使得每个区间都是好的。

朴素的 $O(n^2)$ dp 是显然的。
假设要计算 $dp_i = \sum dp_j$。
$min \leq len$ 显然在某个 j 之前一定满足。
那么只需要考虑 $len \leq max$ 的限制。
设 $v_j = j$ 到 i 的 $max - len$。

Safe Partition

给一个长为 $n(n \leq 5 \cdot 10^5)$ 的序列 A。对于一个区间 $[l, r]$，如果 $min(A_l, \ldots, A_r) \leq r - l + 1 \leq max(A_l, \ldots, A_r)$，那么这个区间是好的。
你需要求出有多少种方法把 A 划分为若干个区间，使得每个区间都是好的。

朴素的 $O(n^2)$ dp 是显然的。
假设要计算 $dp_i = \sum dp_j$。
$min \leq len$ 显然在某个 j 之前一定满足。
那么只需要考虑 $len \leq max$ 的限制。
设 $v_j = j$ 到 i 的 $max - len$。
那么要统计前面 $v_j \geq 0$ 的 dp 值之和，而每次 i 右移会使所有 v_j 减 1。

Safe Partition

给一个长为 $n(n \leq 5 \cdot 10^5)$ 的序列 A。对于一个区间 $[l, r]$，如果 $min(A_l, \ldots, A_r) \leq r - l + 1 \leq max(A_l, \ldots, A_r)$，那么这个区间是好的。

你需要求出有多少种方法把 A 划分为若干个区间，使得每个区间都是好的。

朴素的 $O(n^2)$ dp 是显然的。
假设要计算 $dp_i = \sum dp_j$。
$min \leq len$ 显然在某个 j 之前一定满足。
那么只需要考虑 $len \leq max$ 的限制。
设 $v_j = j$ 到 i 的 $max - len$。
那么要统计前面 $v_j \geq 0$ 的 dp 值之和，而每次 i 右移会使所有 v_j 减 1。
用一个单调栈维护前面 max，那么每次还会进行一些区间加。

Safe Partition

给一个长为 $n(n \le 5 \cdot 10^5)$ 的序列 A。对于一个区间 $[l, r]$，如果 $min(A_l, \dots, A_r) \le r - l + 1 \le max(A_l, \dots, A_r)$，那么这个区间是好的。
你需要求出有多少种方法把 A 划分为若干个区间，使得每个区间都是好的。

朴素的 $O(n^2)$ dp 是显然的。
假设要计算 $dp_i = \sum dp_j$。
$min \le len$ 显然在某个 j 之前一定满足。
那么只需要考虑 $len \le max$ 的限制。
设 $v_j = j$ 到 i 的 $max - len$。
那么要统计前面 $v_j \ge 0$ 的 dp 值之和，而每次 i 右移会使所有 v_j 减 1。
用一个单调栈维护前面 max，那么每次还会进行一些区间加。

总的区间加次数是 $O(n)$ 的。每个数由负变正的次数可以如下考虑：
每次 i 变化时，实际是把 v_j 和 $max - i + j - 1$ 取 max。
那么一个数要变正，只可能在红色区域内。(图在下一页)
假设有 x 个数由负变正，红色区域的面积显然不低于 $1 + 2 + \cdots + x = O(x^2)$。总的变号次数不超过 $O(n^{1.5})$，复杂度不超过 $O(n^{1.5} \log n)$。
这个做法虽然特别菜，但是跑的还挺快的（可能实际根本到不了 $sqrt$ 吧，说不定是 $\log$ 或者 $\log^2$ 的)。

总的区间加次数是 $O(n)$ 的。每个数由负变正的次数可以如下考虑：
每次 i 变化时，实际是把 v_j 和 $max - i + j - 1$ 取 max。
那么一个数要变正，只可能在红色区域内。(图在下一页)
假设有 x 个数由负变正，红色区域的面积显然不低于 $1 + 2 + \cdots + x = O(x^2)$。总的变号次数不超过 $O(n^{1.5})$，复杂度不超过 $O(n^{1.5} \log n)$。
这个做法虽然特别菜，但是跑的还挺快的（可能实际根本到不了 $sqrt$ 吧，说不定是 $\log$ 或者 $\log^2$ 的)。
几个题解：
`https://www.cnblogs.com/Enceladus/p/9494066.html`
`https://smijake3.hatenablog.com/entry/2018/08/18/181044`
`https://codeforces.com/blog/entry/60982?#comment-454544`

CC SAFPAR