回课：分治

https://frank3215.github.io/

分治

- 略。

题目讲解

- 坏消息：我们没有找到“纯分治”的题目。
- 因此，我们直接讲分治相关的知识点。
- 考虑到在场有神仙，我们会同时给出简单题和较为困难的题目。

根号分治

- 求 n 的所有**约数**。

根号分治

- 求 n 的所有**约数**。

- 按 $< \sqrt{n}$ 和 $> \sqrt{n}$ 讨论即可，复杂度 $O(\sqrt{n})$.

P3396 哈希冲突

- 给定 n 长序列，m 个操作：
  - A x y 询问在序列下标 mod x 时，余数为 y 的下标的对应的值的加和
  - C x y 把序列第 x 个数的值替换成 y

- n⩽150000,m⩽150000.

根号分治？！

- 按询问的 x 与 $\sqrt{n}$ 的大小关系分类讨论：

根号分治？！

- 按询问的 x 与 $\sqrt{n}$ 的大小关系分类讨论：
  - $x < \sqrt{n}$ 的询问：每次修改时实时更新。
  - $x > \sqrt{n}$ 的询问：每次询问时暴力查询。

根号分治？！

- 按询问的 x 与 $\sqrt{n}$ 的大小关系分类讨论：
  - $x < \sqrt{n}$ 的询问：每次修改时实时更新。修改复杂度 $O(\sqrt{n})$。
  - $x > \sqrt{n}$ 的询问：每次询问时暴力查询。询问复杂度 $O(\sqrt{n})$。
- 总复杂度 $O(m\sqrt{n})$

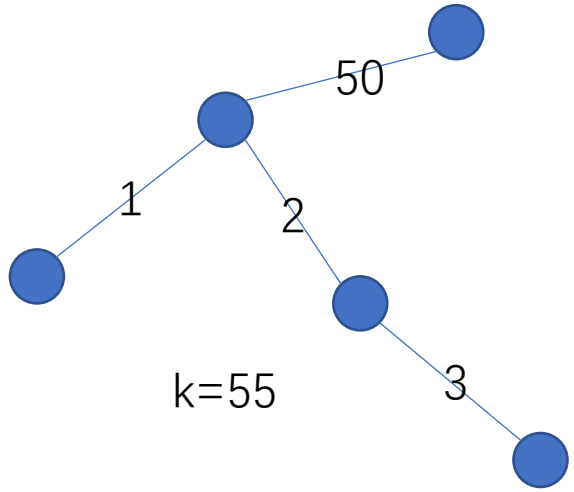
分块

- 略。可搜 “根号平衡” 了解。

P3806 【模板】点分治1（点分治）

题目描述

给定一棵有 n 个点的树，询问树上距离为 k 的点对是否存在。



P3806 【模板】点分治1（点分治）

题目描述

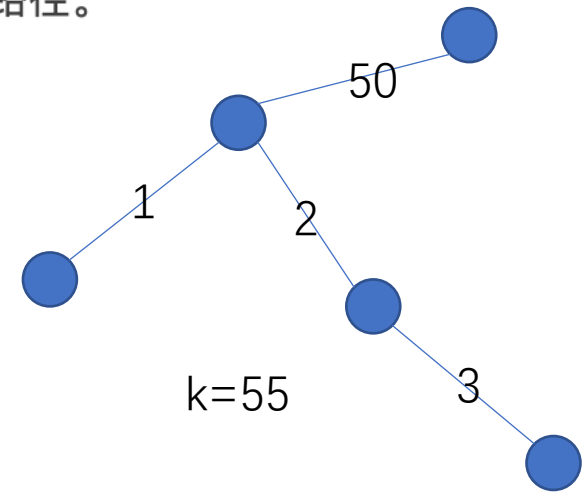
给定一棵有 n 个点的树，询问树上距离为 k 的点对是否存在。

输入格式

第一行两个数 n, m。

第 2 到第 n 行，每行三个整数 u, v, w，代表树上存在一条连接 u 和 v 边权为 w 的路径。

接下来 m 行，每行一个整数 k，代表一次询问。



P3806 【模板】点分治1（点分治）

题目描述

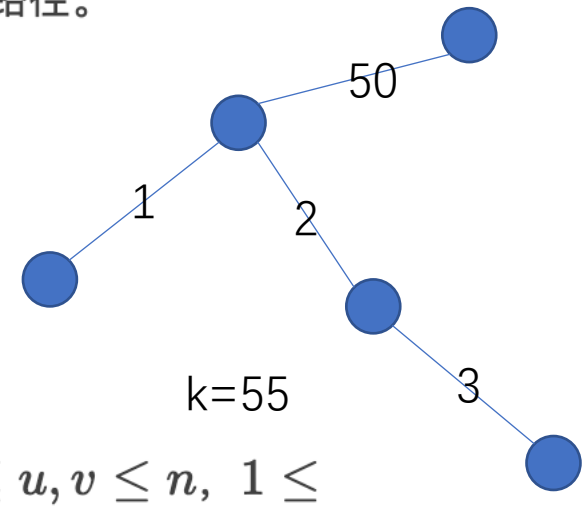
给定一棵有 n 个点的树，询问树上距离为 k 的点对是否存在。

输入格式

第一行两个数 n, m。

第 2 到第 n 行，每行三个整数 u, v, w，代表树上存在一条连接 u 和 v 边权为 w 的路径。

接下来 m 行，每行一个整数 k，代表一次询问。

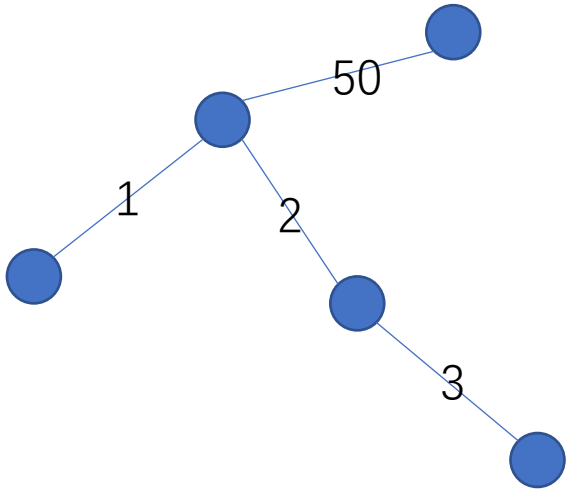


数据规模与约定

- 对于 30% 的数据，保证 $n \leq 100$。
- 对于 60% 的数据，保证 $n \leq 1000$，$m \leq 50$ 。
- 对于 100% 的数据，保证 $1 \leq n \leq 10^4$，$1 \leq m \leq 100$，$1 \leq k \leq 10^7$，$1 \leq u, v \leq n$，$1 \leq w \leq 10^4$。

P3806 【模板】点分治1

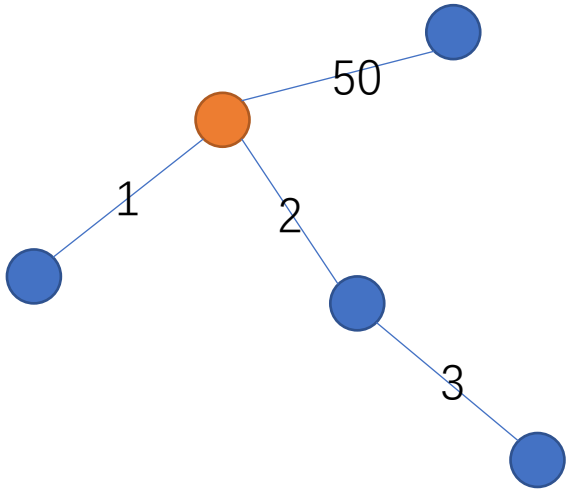
- https://oi-wiki.org/graph/tree-divide/



P3806 【模板】点分治1

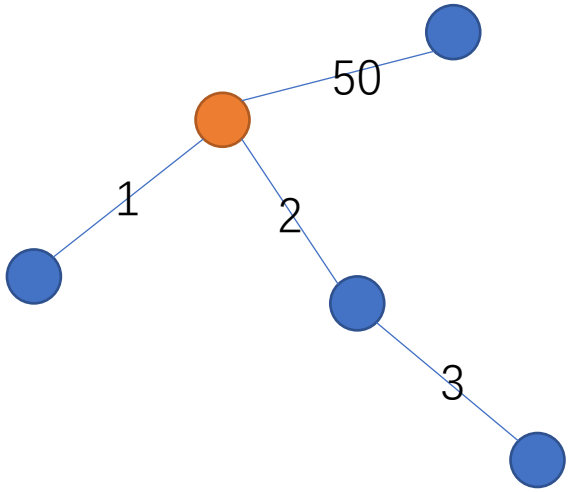
- https://oi-wiki.org/graph/tree-divide/

- 按点分治。



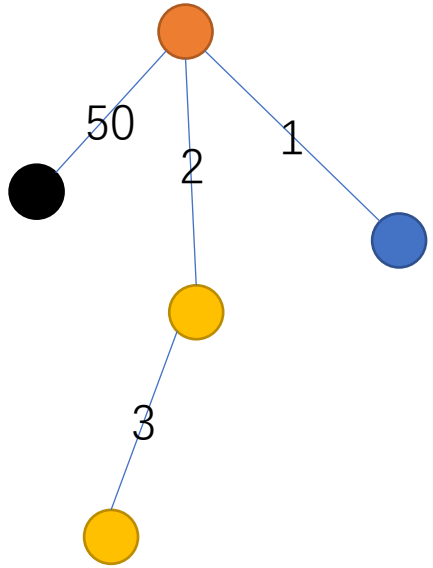
P3806 【模板】点分治1

- https://oi-wiki.org/graph/tree-divide/
- 按点分治。
- 树的重心的性质：所有子树节点数均 <= n/2。



伪代码

- Solve(T):
  - 找到重心 g
  - 遍历 g 的儿子 u:
    - 统计子树 u 的答案
    - 更新子树 u 的信息
  - 清除存储的信息
  - 删去 g 点
  - 遍历 g 曾经的儿子 u:
    - Solve(子树u)



复杂度

- 空间复杂度 O(n)

复杂度

- 空间复杂度 O(n)

- O(log n) 层（重心性质）
- 每层复杂度 O(n)
- 总时间复杂度 O(n log n)

复杂度

- 空间复杂度 O(n)

- O(log n) 层（重心性质）
- 每层复杂度 O(n)
- 总时间复杂度 O(n log n)

- 举例：菊花图、链。

BZOJ2152 聪聪可可

- 求树上长为 3 的倍数的路径数。

- n <= 20000

BZOJ2152 聪聪可可

- 求树上长为 3 的倍数的路径数。
- n <= 20000
- 简单。DFS时记录到根的路径模3的余数即可。

P2617 Dynamic Rankings（整体二分）

给定一个含有 n 个数的序列 $a_1, a_2 \dots a_n$，需要支持两种操作：

- `Q l r k` 表示查询下标在区间 $[l, r]$ 中的第 k 小的数
- `C x y` 表示将 a_x 改为 y

P2617 Dynamic Rankings（整体二分）

给定一个含有 n 个数的序列 $a_1, a_2 \dots a_n$，需要支持两种操作：

- `Q l r k` 表示查询下标在区间 $[l, r]$ 中的第 k 小的数
- `C x y` 表示将 a_x 改为 y

【数据范围】

对于 10% 的数据，$1 \leq n, m \leq 100$；
对于 20% 的数据，$1 \leq n, m \leq 1000$；
对于 50% 的数据，$1 \leq n, m \leq 10^4$；
对于 100% 的数据，$1 \leq n, m \leq 10^5$，$1 \leq l \leq r \leq n$，$1 \leq k \leq r - l + 1$，$1 \leq x \leq n$，$0 \leq a_i, y \leq 10^9$。

假设只询问一个区间[1,n]，没有修改

- 选择算法 / 排序。

假设只询问一个区间[1,n]，没有修改

- 选择算法 / 排序 => 难处理修改，也难扩展到多次询问。

假设只询问一个区间[1,n]，没有修改

- 选择算法 / 排序 => 难处理修改，也难扩展到多次询问。
- 二分答案？

伪代码

- 初始二分区间 [valL, valR] = [0, 10^9]，要求数组 A[1..n] 的第 k 小。
- 直到 valL == valR:
  - 令 valM = (valL + valR) / 2，下取整。
  - 计数 [1, n] 中小于等于 valM 的数的个数，记为 k`
  - 若 k` >= k:
    - 可知前 k 小的数都 <= valM。
    - 令 [valL, valR] = [valL, valM]，循环。
  - 否则，若 k` < k:
    - 可知第 k 小的数 > valM
    - 令 [valL, valR] = [valM+1, valR]，循环。

时间复杂度

- O(log n) 层
- 每层有 n 个数。
- 总时间复杂度 O(n log n)。

重复计算？

- 每次更新区间时，可以丢弃一半的值。
- [valL, valR] := [valL, valM]
  - > valM 的数可以不要（一定不会产生贡献）。
- [valL, valR] := [valM+1, valL]
  - <= valM 的数可以不要（一定会产生贡献）。

更新后的伪代码

- 初始区间 [valL, valR] = [1, n]，要求第 k 小。
- 直到 valL == valR:
  - 令 valM = (valL + valR) / 2，下取整。
  - 计数 [1, n] 中小于等于 valM 的数的个数，记为 k`
    - 将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。
  - 若 k` >= k:
    - 可知前 k 小的数都 <= valM。
    - 令 [valL, valR] = [valL, valM]，令 A[] = L[]，循环。
  - 否则，若 k` < k:
    - 可知第 k 小的数 > valM
    - 令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 k = k – k`，循环。

举个例子

- A = [1, 5, 2, 4, 3]，找第 k = 4 小：
- 最开始，[valL, valR] = [1,5]。
- valM = 4, A = [1, 5, 2, 4, 3], 共有 3 个小于等于 valM 的数。
  - 令 A = [5, 4], k = 4 – 3 = 1, [valL, valR] = [4, 5] 继续。
- valM = 4, A = [5, 4], 共有 1 个小于等于 valM 的数。
  - 令 A = [4], [valL, valR] = [4, 4], 达到结束条件 valL == valR, 答案为 4。

时间复杂度

- O(log n) 层
- 每层有 O(n) 个数。
- 总时间复杂度 O(n log n)。

- 如果数的分布均匀的话？
- 第一层 n 个，第二层约 n/2 个，第三层约 n/4 个……
- 时间复杂度约为 O(n)。

假设询问多个区间$[l_i, r_i]$，无修改

- 尝试扩展之前的二分答案的解法。

假设询问多个区间$[l_i, r_i]$，无修改

- 尝试扩展之前的二分答案的解法。

- 离线后，同时进行多个二分答案？——怎么做？

同时二分

- 考虑有若干个询问，$[l_1, r_1]$, $[l_2, r_2]$, $[l_3, r_3]$。初始的二分区间为 $[1, n]$

同时二分

- 考虑有若干个询问，$[l_1, r_1]$, $[l_2, r_2]$, $[l_3, r_3]$。初始的二分区间为 [1, n]
- 根据之前的伪代码，考虑需要更改哪些部分。

- 初始区间 [valL, valR] = [1, n]，要求第 k 小。
- 直到 valL == valR:
  - 令 valM = (valL + valR) / 2，下取整。
  - 计数 [1, n] 中小于等于 valM 的数的个数，记为 k`
    - 将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。
  - 若 k` >= k:
    - 令 [valL, valR] = [valL, valM]，令 A[] = L[]，循环。
  - 否则，若 k` < k:
    - 令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 k = k – k`，循环。

同时二分

- 考虑有若干个询问，$[l_1, r_1]$, $[l_2, r_2]$, $[l_3, r_3]$。初始的二分区间为 [1, n]
- 根据之前的伪代码，考虑需要更改哪些部分。
- 如何对这些区间分别计数大于等于 n/2 的数的个数？

- 初始区间 [valL, valR] = [1, n]，要求第 k 小。
- 直到 valL == valR:
  - 令 valM = (valL + valR) / 2，下取整。
  - 计数 [1, n] 中小于等于 valM 的数的个数，记为 k`
    - 将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。
  - 若 k` >= k:
    - 令 [valL, valR] = [valL, valM]，令 A[] = L[]，循环。
  - 否则，若 k` < k:
    - 令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 k = k – k`，循环。

同时二分

- 考虑有若干个询问，$[l_1, r_1], [l_2, r_2], [l_3, r_3]$。初始的二分区间为 $[1, n]$
- 如何对这些区间分别计数大于等于 $n/2$ 的数的个数？

- 将 $< n/2$ 的数视为 0，$>= n/2$ 的数视为 1。

同时二分

- 考虑有若干个询问，$[l_1, r_1], [l_2, r_2], [l_3, r_3]$。初始的二分区间为 $[1, n]$
- 如何对这些区间分别计数大于等于 $n/2$ 的数的个数？

- 将 $< n/2$ 的数视为 0，$>= n/2$ 的数视为 1。
- 离散化后，按下标进行前缀和即可。

同时二分

- 考虑有若干个询问，$[l_1, r_1]$, $[l_2, r_2]$, $[l_3, r_3]$。初始的二分区间为 [1, n]
- 继续根据之前的伪代码，考虑需要更改哪些部分。

- 初始区间 [valL, valR] = [1, n]，要求第 k 小。假设已离散化。
- 直到 valL == valR:
  - 令 valM = (valL + valR) / 2，下取整。
  - 用下标前缀和分别计数 $[l_i, r_i]$ 中小于等于 valM 的数的个数，记为 k_i`
    - 将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。
  - 若 k` >= k:
    - 令 [valL, valR] = [valL, valM]，令 A[] = L[]，循环。
  - 否则，若 k` < k:
    - 令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 k = k – k`，循环。

同时二分

- 考虑有若干个询问，$[l_1, r_1]$, $[l_2, r_2]$, $[l_3, r_3]$。初始的二分区间为 [1, n]
- 继续根据之前的伪代码，考虑需要更改哪些部分。
- 如何把分类讨论的循环改成分治？

  - 初始区间 [valL, valR] = [1, n]，要求第 k 小。假设已离散化。
  - 直到 valL == valR:
    - 令 valM = (valL + valR) / 2，下取整。
    - 用下标前缀和分别计数 $[l_i, r_i]$ 中小于等于 valM 的数的个数，记为 k_i`
      - 将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。
    - 若 k` >= k:
      - 令 [valL, valR] = [valL, valM]，令 A[] = L[]，循环。
    - 否则，若 k` < k:
      - 令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 k = k – k`，循环。

同时二分

- 考虑有若干个询问，$[l_1, r_1]$, $[l_2, r_2]$, $[l_3, r_3]$。初始的二分区间为 [1, n]
- 继续根据之前的伪代码，考虑需要更改哪些部分。
- 如何把分类讨论的循环改成分治？

- 很简单。

- 初始区间 [valL, valR] = [1, n]，要求第 k 小。假设已离散化。
- 直到 valL == valR:
  - 令 valM = (valL + valR) / 2，下取整。
  - 用下标前缀和分别计数 $[l_i, r_i]$ 中小于等于 valM 的数的个数，记为 k_i`
    - 将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。
  - 将所有 k_i` >= k_i 的询问：
    - 令 [valL, valR] = [valL, valM]，令 A[] = L[]，**递归**解决。
  - 将所有 k_i` < k_i 的询问：
    - 令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 $k_i = k_i - k_i$`，**递归**解决。

时间复杂度

初始区间 [valL, valR] = [1, n]，要求第 k 小。假设已离散化。

直到 valL == valR:

令 valM = (valL + valR) / 2，下取整。

用下标前缀和分别计数 $[l_i, r_i]$ 中小于等于 valM 的数的个数，记为 k_i`

将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。

将所有 k_i` >= k_i 的询问：

令 [valL, valR] = [valL, valM]，令 A[] = L[]，**递归**解决。

将所有 k_i` < k_i 的询问：

令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 $k_i = k_i - k_i$`，**递归**解决。

时间复杂度

- 一共有 O(log n) 层。

初始区间 [valL, valR] = [1, n]，要求第 k 小。假设已离散化。
直到 valL == valR:
令 valM = (valL + valR) / 2，下取整。
用下标前缀和分别计数 $[l_i, r_i]$ 中小于等于 valM 的数的个数，记为 k_i`
将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。
将所有 k_i` >= k_i 的询问：
令 [valL, valR] = [valL, valM]，令 A[] = L[]，**递归**解决。
将所有 k_i` < k_i 的询问：
令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 $k_i = k_i - k_i$`，**递归**解决。

时间复杂度

- 一共有 O(log n) 层。
- 每一层共处理 O(n) 个元素、O(m) 个询问。
  - 划分的复杂度为 O(n + m)。

初始区间 [valL, valR] = [1, n]，要求第 k 小。假设已离散化。
直到 valL == valR:
令 valM = (valL + valR) / 2，下取整。
用下标前缀和分别计数 $[l_i, r_i]$ 中小于等于 valM 的数的个数，记为 k_i`
将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。
将所有 k_i` >= k_i 的询问：
令 [valL, valR] = [valL, valM]，令 A[] = L[]，**递归**解决。
将所有 k_i` < k_i 的询问：
令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 $k_i = k_i - k_i$`，**递归**解决。

时间复杂度

- 一共有 O(log n) 层。
- 每一层共处理 O(n) 个元素、O(m) 个询问。
  - 划分的复杂度为 O(n + m)。
- 每一个子问题的 A[] 是原来 A[1..n] 的子序列，元素两两不交。故长度之和为 O(n)。
  - 下标前缀和的复杂度是 O(n) 的。
- 总复杂度为 O((n+m)log n)

初始区间 [valL, valR] = [1, n]，要求第 k 小。假设已离散化。
直到 valL == valR:
令 valM = (valL + valR) / 2，下取整。
用下标前缀和分别计数 $[l_i, r_i]$ 中小于等于 valM 的数的个数，记为 k_i`
将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。
将所有 k_i` >= k_i 的询问：
令 [valL, valR] = [valL, valM]，令 A[] = L[]，**递归**解决。
将所有 k_i` < k_i 的询问：
令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 $k_i = k_i - k_i$`，**递归**解决。

假设只询问[1,n]，有修改

- 很明显，答案与序列元素的顺序无关。

假设只询问[1,n]，有修改

- 很明显，答案与序列元素的顺序无关。
- 更进一步地，答案只与每个数出现了多少次有关。

假设只询问[1,n]，有修改

- 很明显，答案与序列元素的顺序无关。
- 更进一步地，答案只与每个数出现了多少次有关。
- ~~只需要用平衡树/线段树/树状数组维护每一个值出现多少次即可。~~

假设只询问[1,n]，有修改

- 很明显，答案与序列元素的顺序无关。
- 更进一步地，答案只与每个数出现了多少次有关。
- ~~只需要用平衡树/线段树/树状数组维护每一个值出现多少次即可。~~
  - 利用可离线处理的性质，可以不用数据结构进行解答。
- 尝试扩展之前的二分答案的解法。

假设只询问[1,n]，有修改

- 很明显，答案与序列元素的顺序无关。
- 更进一步地，答案只与每个数出现了多少次有关。
- ~~只需要用平衡树/线段树/树状数组维护每一个值出现多少次即可。~~
  - 利用可离线处理的性质，可以不用数据结构进行解答。
- 尝试扩展之前的二分答案的解法。——怎么处理修改？

假设只询问[1,n]，有修改

- 很明显，答案与序列元素的顺序无关。
- 更进一步地，答案只与每个数出现了多少次有关。
- ~~只需要用平衡树/线段树/树状数组维护每一个值出现多少次即可。~~
  - 利用可离线处理的性质，可以不用数据结构进行解答。
- 尝试扩展之前的二分答案的解法。——把修改看成删去一个值后、再加上一个值；按时间顺序处理所有询问与修改！

伪代码

- 如何修改之前的伪代码？

初始区间 [valL, valR] = [1, n]，要求第 k 小。假设已离散化。
直到 valL == valR:
    令 valM = (valL + valR) / 2，下取整。
    用下标前缀和分别计数 $[l_i, r_i]$ 中小于等于 valM 的数的个数，记为 k_i`
        将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。
    将所有 k_i` >= k_i 的询问：
        令 [valL, valR] = [valL, valM]，令 A[] = L[]，**递归**解决。
    将所有 k_i` < k_i 的询问：
        令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 $k_i = k_i - k_i$`，**递归**解决。

伪代码

- 如何修改之前的伪代码？
- 只需要修改核心语句。

初始区间 [valL, valR] = [1, n]，要求第 k 小。假设已离散化。
直到 valL == valR:
    令 valM = (valL + valR) / 2，下取整。
    用下标前缀和分别计数 $[l_i, r_i]$ 中小于等于 valM 的数的个数，记为 k_i`
        将 A[] 中小于等于 valM 的数放入 L[] 中，将大于 valM 的数放入 R[] 中。
    将所有 k_i` >= k_i 的询问：
        令 [valL, valR] = [valL, valM]，令 A[] = L[]，**递归**解决。
    将所有 k_i` < k_i 的询问：
        令 [valL, valR] = [valM+1, valR]，令 A[] = R[]，令 $k_i = k_i - k_i$`，**递归**解决。

伪代码

- 如何修改之前的伪代码？

- 很简单。甚至不需要前缀和。
  - 不如说前缀和根本不支持动态修改。
- 时间复杂度依然为 O((n+m)log n)

初始区间 [valL, valR] = [1, n]，要求第 k 小，A[] 按时间顺序记录所有值、修改、与询问。假设已离散化。
直到 valL == valR:
    令 valM = (valL + valR) / 2，下取整。
    按时序处理修改与询问。假设 $[l_i, r_i]$ 中小于等于 valM 的数的个数为 k_i`
        将 A[] 中小于等于 valM 的数与修改放入 L[] 中，将大于 valM 的数与修改放入 R[] 中。
    将所有 k_i` >= k_i 的询问：
        令 [valL, valR] = [valL, valM]，令 A[] = L[] + 这些询问，递归解决。
    将所有 k_i` < k_i 的询问：
        令 [valL, valR] = [valM+1, valR]，令 A[] = R[] + 这些询问，令 $k_i = k_i - k_i$`，递归解决。

假设询问多个区间$[l_i, r_i]$，有修改（原问题）

- 考虑之前的伪代码。

初始区间 [valL, valR] = [1, n]，要求第 k 小，A[] 按时间顺序记录所有值、修改、与询问。假设已离散化。
直到 valL == valR:
    令 valM = (valL + valR) / 2，下取整。
    按时序处理修改与询问。假设 $[l_i, r_i]$ 中小于等于 valM 的数的个数为 k_i`
        将 A[] 中小于等于 valM 的数与修改放入 L[] 中，将大于 valM 的数与修改放入 R[] 中。
    将所有 k_i` >= k_i 的询问：
        令 [valL, valR] = [valL, valM]，令 A[] = L[] + 这些询问，**递归**解决。
    将所有 k_i` < k_i 的询问：
        令 [valL, valR] = [valM+1, valR]，令 A[] = R[] + 这些询问，令 $k_i = k_i - k_i$`，**递归**解决。

假设询问多个区间$[l_i, r_i]$，有修改（原问题）

- 考虑之前的伪代码。
- 核心部分需要支持：单点修改，区间查询。

初始区间 [valL, valR] = [1, n]，要求第 k 小，A[] 按时间顺序记录所有值、修改、与询问。假设已离散化。
直到 valL == valR:
令 valM = (valL + valR) / 2，下取整。
按时序处理修改与询问。假设 $[l_i, r_i]$ 中小于等于 valM 的数的个数为 k_i`
将 A[] 中小于等于 valM 的数与修改放入 L[] 中，将大于 valM 的数与修改放入 R[] 中。
将所有 k_i` >= k_i 的询问：
令 [valL, valR] = [valL, valM]，令 A[] = L[] + 这些询问，**递归**解决。
将所有 k_i` < k_i 的询问：
令 [valL, valR] = [valM+1, valR]，令 A[] = R[] + 这些询问，令 $k_i = k_i - k_i$`，**递归**解决。

假设询问多个区间$[l_i, r_i]$，有修改（原问题）

- 考虑之前的伪代码。
- 核心部分需要支持：单点修改，区间查询。
- 用树状数组即可。
- 记得还原。

初始区间 [valL, valR] = [1, n]，要求第 k 小，A[] 按时间顺序记录所有值、修改、与询问。假设已离散化。
直到 valL == valR:
    令 valM = (valL + valR) / 2，下取整。
    用下标树状数组，按时序处理修改与询问。假设 $[l_i, r_i]$ 中小于等于 valM 的数的个数为 k_i`
        将 A[] 中小于等于 valM 的数与修改放入 L[] 中，将大于 valM 的数与修改放入 R[] 中。
        还原树状数组。
    将所有 k_i` >= k_i 的询问：
        令 [valL, valR] = [valL, valM]，令 A[] = L[] + 这些询问，**递归**解决。
    将所有 k_i` < k_i 的询问：
        令 [valL, valR] = [valM+1, valR]，令 A[] = R[] + 这些询问，令 $k_i = k_i - k_i$`，**递归**解决。

时间复杂度？

- 一共有 O(log n) 层。
- 每一层共处理 O(n) 个元素、O(m) 个询问。
  - 划分的复杂度为 O(n + m)。
- 每一个子问题的 A[] 是原来 A[1..n] 的子序列，元素两两不交。故长度之和为 O(n)。
  - 下标树状数组的总复杂度是 O((n+m) log n) 的。

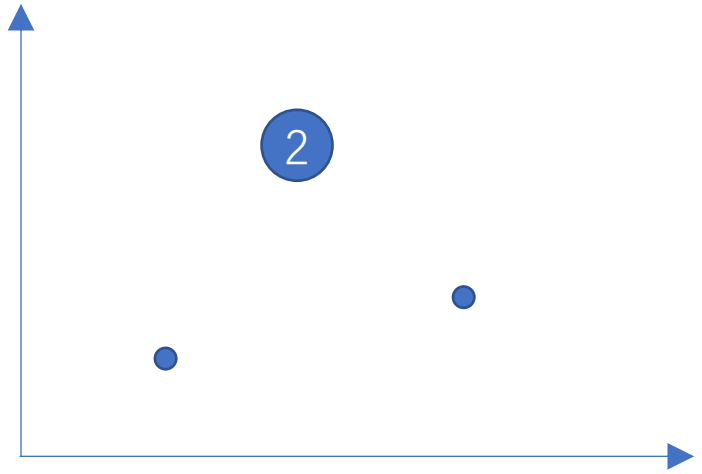
- 总复杂度为 $O((n+m)\log^2 n)$

P3810 【模板】三维偏序（陌上花开）

题目描述

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 $a_j \leq a_i$ 且 $b_j \leq b_i$ 且 $c_j \leq c_i$ 且 $j \neq i$ 的 j 的数量。

对于 $d \in [0, n)$，求 $f(i) = d$ 的数量。



P3810 【模板】三维偏序（陌上花开）

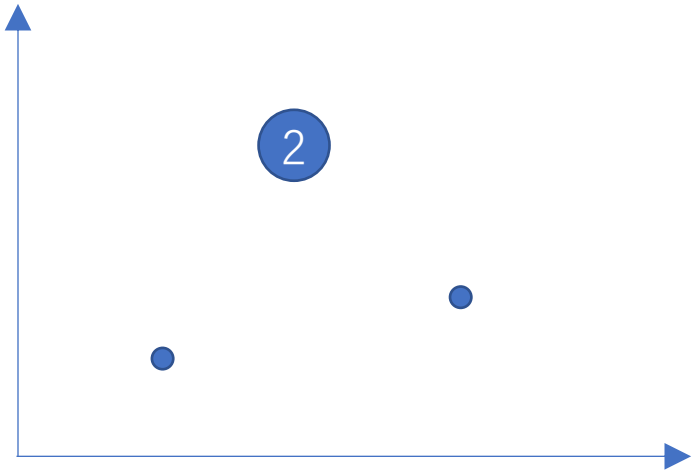
题目描述

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 $a_j \leq a_i$ 且 $b_j \leq b_i$ 且 $c_j \leq c_i$ 且 $j \neq i$ 的 j 的数量。

对于 $d \in [0, n)$，求 $f(i) = d$ 的数量。

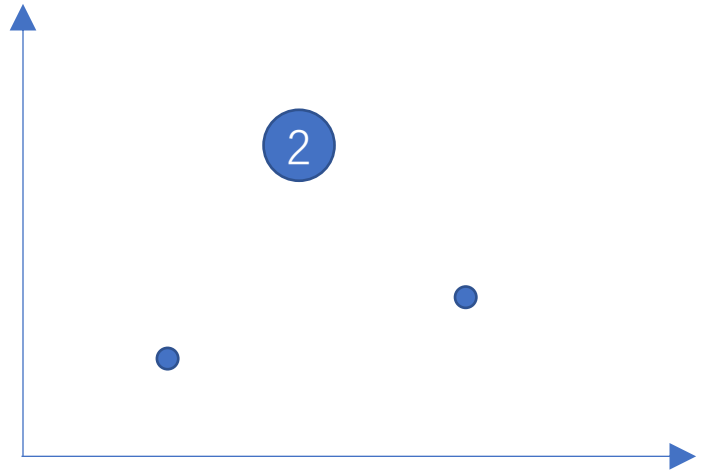
说明/提示

$1 \leq n \leq 10^5$，$1 \leq a_i, b_i, c_i \leq k \leq 2 \times 10^5$。



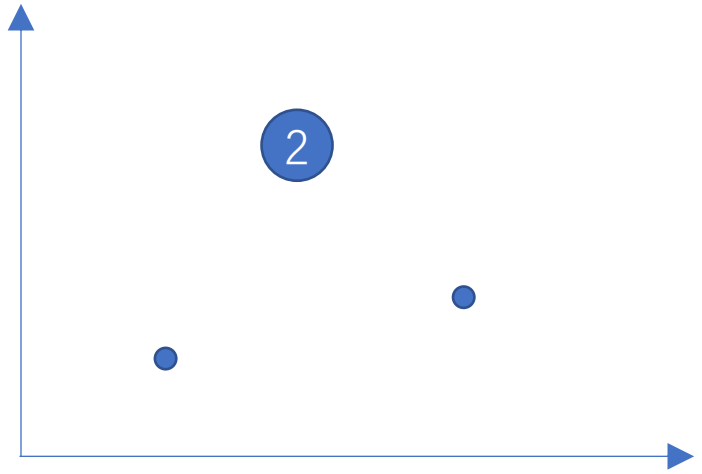
二维偏序怎么做？

- 按 ai 排序后，用数据结构维护有多少个点 j 的 bj 小于 bi。



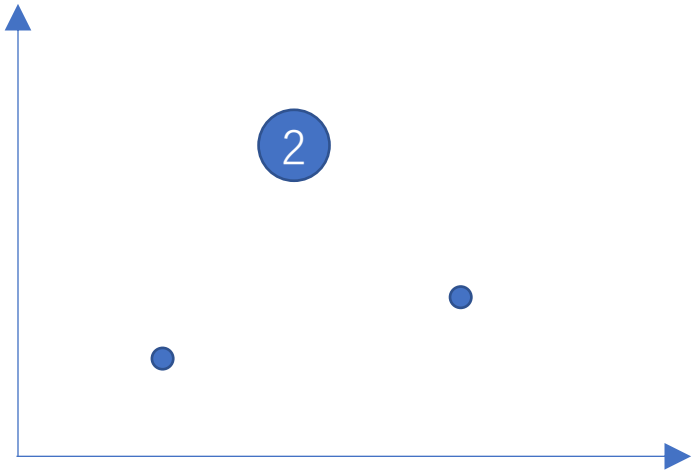
二维偏序怎么做？

- 按 ai 排序后，用数据结构维护有多少个点 j 的 bj 小于等于 bi。
- 可以用树状数组 / 线段树。



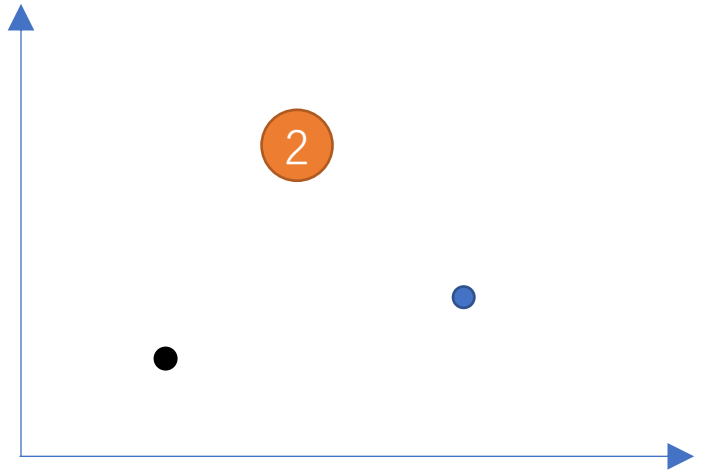
二维偏序怎么做？

- 按 ai 排序后，用数据结构维护有多少个点 j 的 bj 小于等于 bi。
- 可以用树状数组 / 线段树。
- 小细节：ai, bi 可能相同 => 要同时向数据结构中加入 ai 相同的 bi。



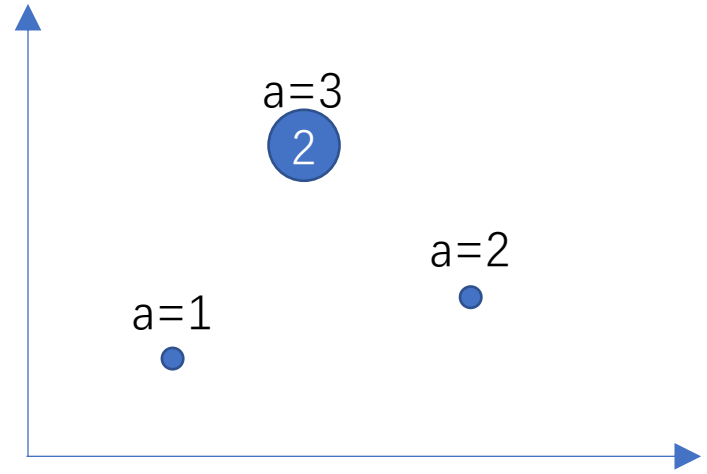
伪代码

- 假设 (ai, bi) 已按 ai 排序：
  - 枚举 a：
  - 将所有 ai = a 的 (ai, bi) 加入数据结构 S。
  - 对每一个 ai = a 的 (ai, bi)：
    - 询问数据结构 S 中有多少 bj <= bi。
    - 答案为询问结果。



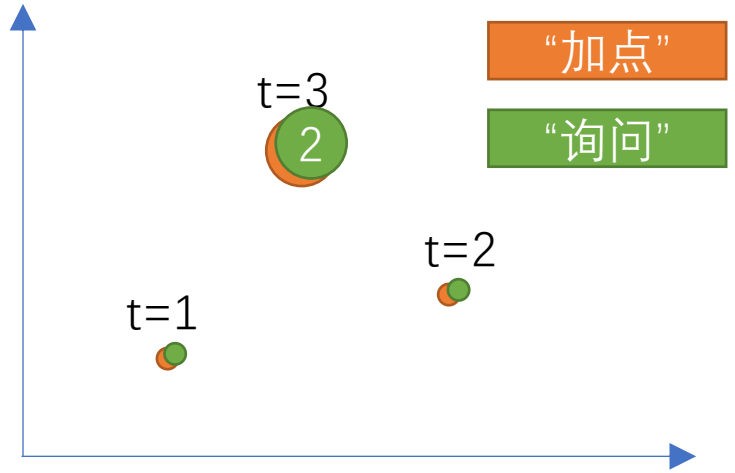
三维偏序怎么做？

- 按 ai 排序后，我们把每个 ai 相同的 (bi, ci) 看作这样的操作：
  - 首先，把所有 ai 相同的 (bi, ci) 加入点集。
  - 之后，对每个 (bi, ci)，询问点集有多少个点 (bj, cj) 与 (bi, ci) 形成偏序。



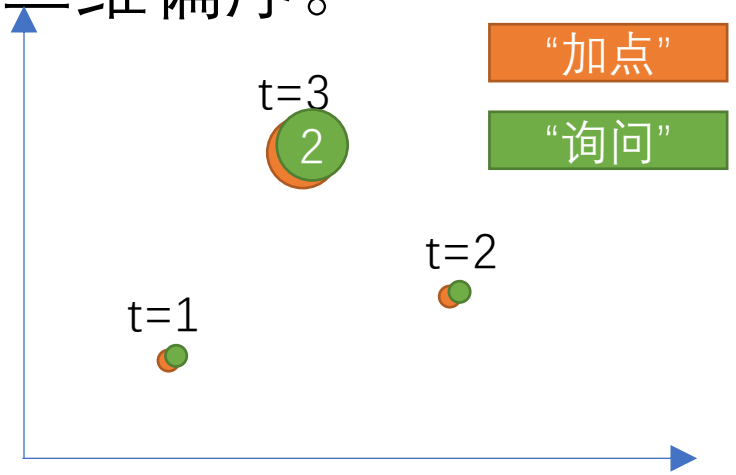
三维偏序怎么做？

- 按 ai 排序后，我们把每个 ai 相同的 (bi, ci) 看作这样的操作：
  - 首先，把所有 ai 相同的 (bi, ci) 加入点集。
  - 之后，对每个 (bi, ci)，询问点集有多少个点 (bj, cj) 与 (bi, ci) 形成偏序。
- 把 ai 看作时间的话，这些点可以看作有时间顺序的一次“加点”和一个“询问”。

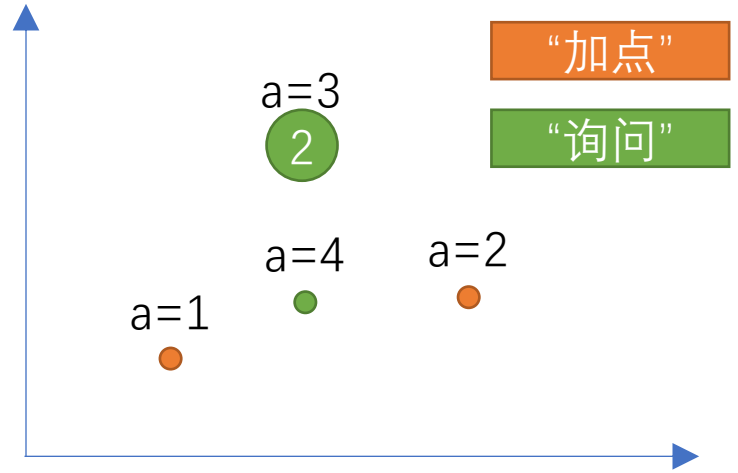


三维偏序怎么做？

- 按 ai 排序后，我们把每个 ai 相同的 (bi, ci) 看作这样的操作：
  - 首先，把所有 ai 相同的 (bi, ci) 加入点集。
  - 之后，对每个 (bi, ci)，询问点集有多少个点 (bj, cj) 与 (bi, ci) 形成偏序。
- 把 ai 看作时间的话，这些点可以看作有时间顺序的一次“加点”和一个“询问”。
- 也即，我们把问题转化为了支持加点与询问的二维偏序。

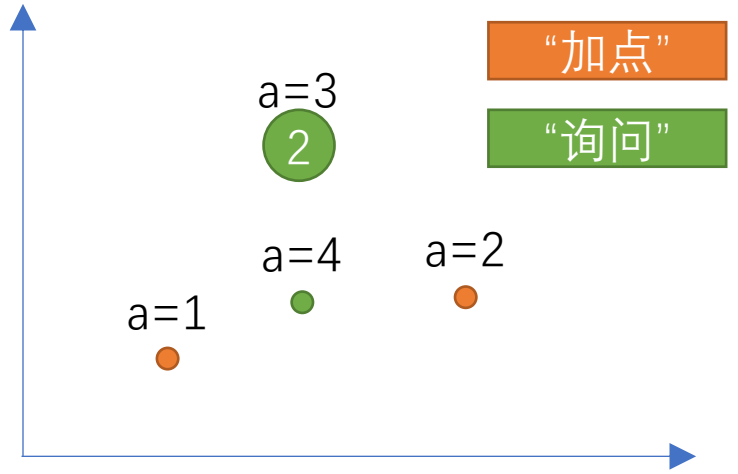


动态二维偏序：假设所有询问在修改后



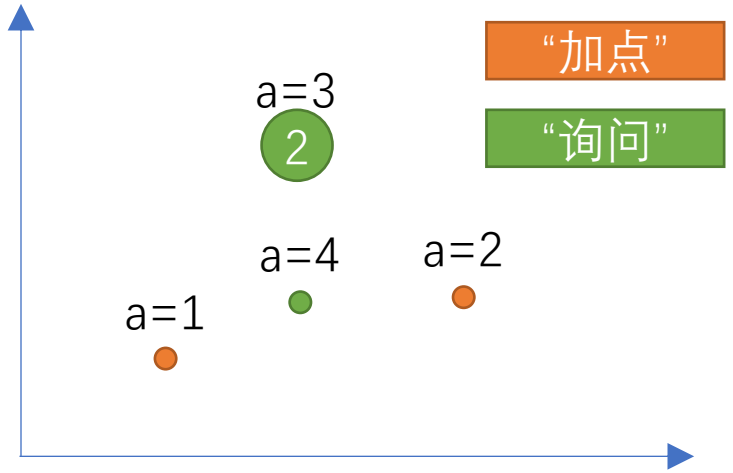
动态二维偏序：假设所有询问在修改后

- 按 bi 排序后，用之前的做二维偏序的方法即可。



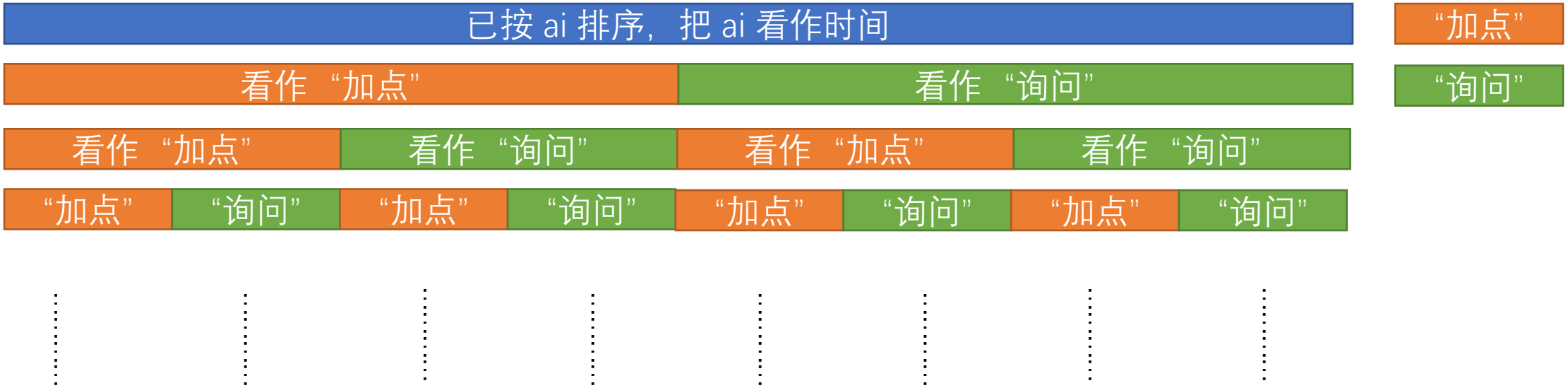
动态二维偏序：假设所有询问在修改后

- 按 bi 排序后，用之前的做二维偏序的方法即可。
- 伪代码：
  - 假设 (bi, ci) 已按 bi 排序：
    - 枚举 b：
    - 将所有 bi = b 的 (bi, ci) 的“加点”操作的 ci 加入数据结构 S。
    - 对每一个 bi = b 的 (bi, ci) 的“询问”操作：
      - 询问数据结构 S 中有多少 cj <= ci。
      - 答案为询问结果。



原问题：CDQ分治

- 可以通过分治把原问题拆成若干个“所有询问在修改后”的问题。



LOJ121 「离线可过」动态图连通性

题目描述

这是一道被离线爆艹的模板题。

你要维护一张无向简单图。你被要求加入删除一条边及查询两个点是否连通。

- `0`：加入一条边。保证它不存在。
- `1`：删除一条边。保证它存在。
- `2`：查询两个点是否连通。

输入格式

输入的第一行是两个数 N M。$N \leq 5000, M \leq 500000$。

接下来 M 行，每一行三个数 op, x, y。op 表示操作编号。

假设没有删边？

假设没有删边？

- 并查集模板题。

假设所有“删边”操作实质为“撤销”

- 也即，所有“删边”操作都删除的是最近加入的（还没被删去）的边。

假设所有“删边”操作实质为“撤销”

- 也即，所有“删边”操作都删除的是最近加入的（还没被删去）的边。

- 可撤销的并查集？

假设所有“删边”操作实质为“撤销”

- 也即，所有“删边”操作都删除的是最近加入的（还没被删去）的边。

- 可撤销的并查集？——可以！

假设所有“删边”操作实质为“撤销”

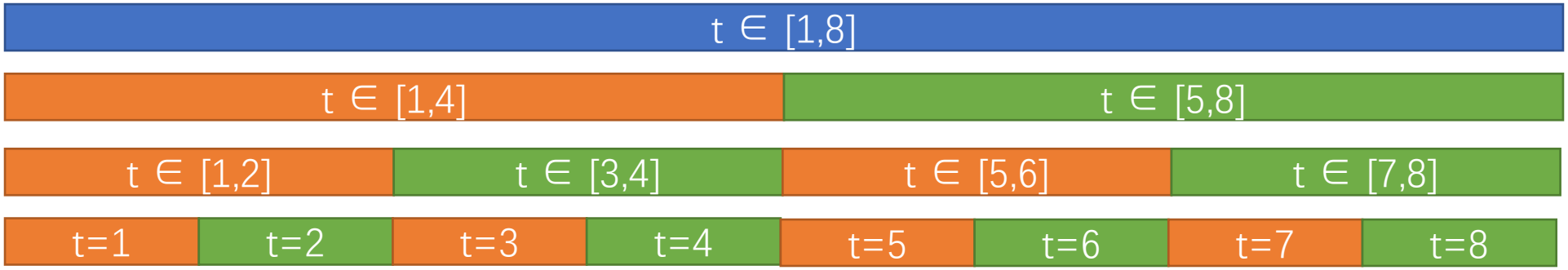
- 也即，所有“删边”操作都删除的是最近加入的（还没被删去）的边。

- 可撤销的并查集？——可以！

- 但不能路径压缩了。
  - “加边”操作复杂度为 O(log n)。
  - “撤销”操作复杂度为 O(1)。

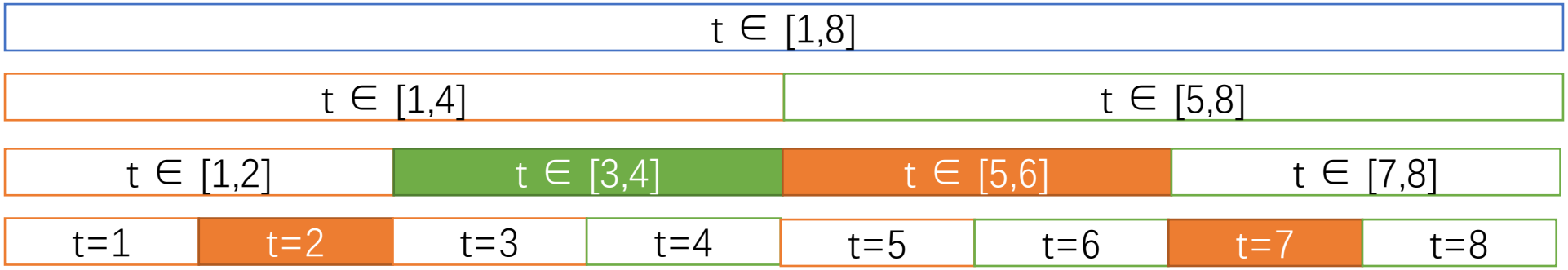
线段树分治

- 按时间进行分治。



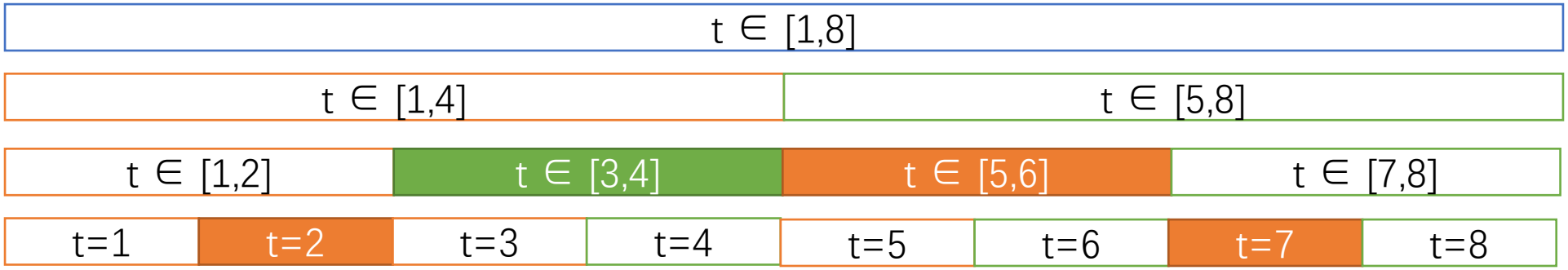
线段树分治

- 按时间进行分治。
- 假设一条边在 t ∈ [2,7] 存在。



线段树分治

- 按时间进行分治。
- 假设一条边在 $t \in [2,7]$ 存在。



- 看作在进入这些节点时“加边”，离开这些节点时“撤销”即可。

时间复杂度

- 由线段树的性质可以证明，一条边至多被拆成 O(log m) 条。
- 并查集的复杂度为 O(log n)
- 总复杂度为 O(m log n log m)。

- 但实际上跑的会很快！