

数据结构的联合

长沙市雅礼中学 黄刚

问题 1 顽皮的猴子

问题描述:

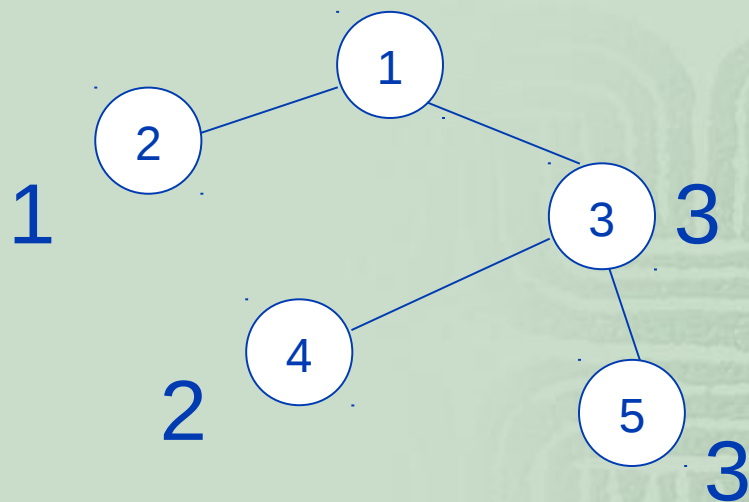
有 N ($1 \leq N \leq 30000$) 只顽皮的猴子挂在树上。每只猴子都有两只手，编号为 **1** 的猴子的尾巴挂在树枝上，其他的猴子的尾巴都被别的猴子的某只手抓着。每一时刻，都有且只有一只猴子的某只手松开，从而可能会有一些猴子掉落至地面。输入一开始猴子们的情况和每一时刻松开手的情况，输出每只猴子落地的时间。



问题 1 顽皮的猴子

分析：

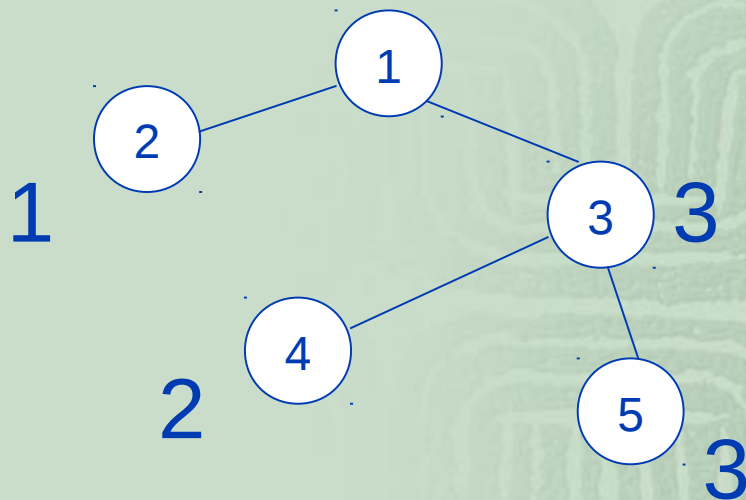
我们把猴子抽象成点，猴子的手抽象成边，题目就转化为一个连通图不断去边，求每个点离开编号为 **1** 的点所在的连通分量的时间。



问题 1 顽皮的猴子

分析:

把删边的顺序倒过来，问题转化为从一个无边的图不断添边，求每个点进入编号为 **1** 的点所在的连通分量的时间。我们自然的想到用并查集维护。



问题 1 顽皮的猴子

一个问题:

上面的算法中，当并查集中某个集合加入编号为 **1** 的点所在的集合时，需要把这个集合中的所有元素的时间记录一下。但是并查集并不支持“枚举集合元素”的功能，怎么办？



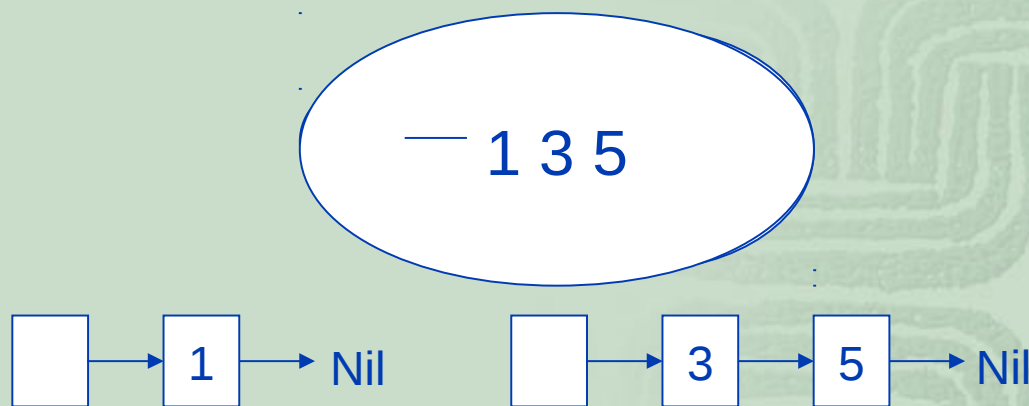
Time[3]:=2
Time[5]:=2



问题 1 顽皮的猴子

一个问题:

给每个并查集分配一个链表，记录这个集合中所有的元素。这样，能够方便的枚举集合中的元素。而在并查集合并的时候，链表也能够很快的合并。问题解决。



数据结构的并联

问题 1 中，在并查集不支持枚举元素操作的时候，引入一个新的数据结构链表来辅助，是数据结构的一种联合方式：并联，这种联合成功的解决了问题。

定义：

我们将用多个数据结构共同对同一数据集合统计的方法叫做数据结构的并联。



数据结构的并联

为了方便起见，先把一个数据结构支持的操作分为两类：

- 询问操作：顾名思义，就是获取该数据结构记录的某些信息的操作，而并没有改动数据结构里的元素及其相互关系。
- 维护操作：跟询问操作相反，改动数据结构中元素或元素的相互关系的操作，称为维护操作。



数据结构的并联

- 并联的优点：

并联而成的新数据结构，支持组成它的数据结构的所有操作。

- 并联的缺点：

维护操作的时间复杂度存在瓶颈，是所有组成它的数据结构的维护操作复杂度的和。



问题 2 DynamicRankings

问题描述:

给定一个长度为 N ($1 \leq N \leq 50000$) 的正整数序列 A ($1 \leq A[i] \leq \text{MaxLongint}$)，模拟以下操作：

- 1、改值操作 $C(i,j)$ ：将 $A[i]$ 的值改为 j
- 2、询问操作 $Q(i,j,k)$ ：程序输出 $A[i], A[i+1], \dots, A[j]$ 这 $j-i+1$ 个数中第 k 大的数。



问题 2 DynamicRankings

例如:

$N=6$

$A=[3,1,6,5,2,4]$

$C(5,4)$ $[3,1,6,5,2,4$

$Q(2,5,3)$ $[3,1,6,5,4,4$

$C(6,2)$ $[3,1,6,5,4,2$

$Q(3,6,1)$ $[3,1,6,5,4,2$

$]$

输出: 4

输出: 6



问题 2 DynamicRankings

分析:

- 单纯的算法必定会超时，我们需要设计一个对整数序列进行统计的高效数据结构，并且支持改值和查找指定区间中第 k 大的数。
- 问题过于复杂，先考虑简单一点的情况。



问题 2 DynamicRankings

简化情况 1 :

每次询问 $Q(i,j)$, 要求程序输出 $A[i], A[i]+1, \dots, A[j]$ 这 $j-i+1$ 个数中最大的数。

解决方式:

只需用一颗线段树维护 A 序列, 改值, 询问操作都是 $O(\log N)$ 的时间复杂度, 完全满足题意。

问题 2 DynamicRankings

简化情况 2 :

每次询问 $Q(k)$, 要求程序输出 $A[1], A[2], \dots, A[N]$ 这 N 个数中, 第 k 大的数。

解决方式:

用一颗平衡二叉树维护 A 序列即可, 同样的, 改值, 询问操作也只需要 $O(\log N)$ 的时间。

问题 2 DynamicRankings

问题的嵌套:

- 现在，两个很简单的问题“嵌套”在了一起，成了一个棘手的问题，使得原先的方法都失效了。
- 既然问题能够“嵌套”，那我们原先解决问题的数据结构是不是也能够“嵌套”呢？



问题 2 DynamicRankings

数据结构的嵌套:

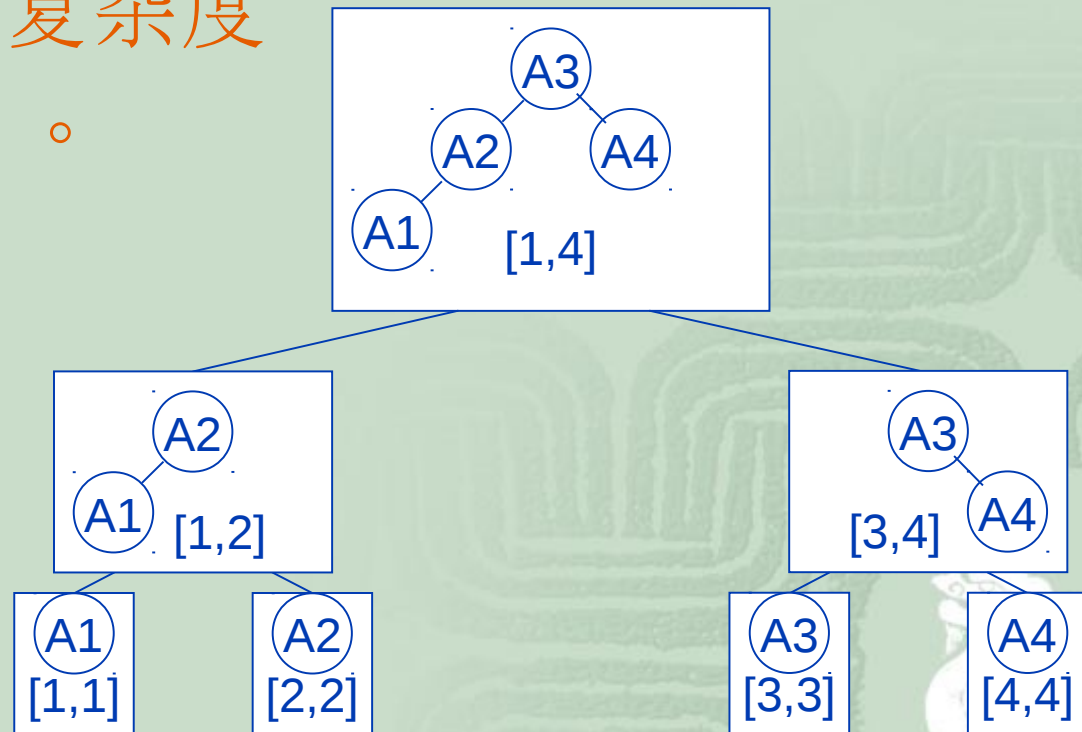
考虑一颗这样的线段树: 它维护整个 A 序列, 而它的每个节点 V , 设表示的区间为 $[i,j]$, 拥有一颗平衡二叉树, 维护 $A[i], A[i+1], \dots, A[j]$ 。

为下文方便起见, 暂时称之为“嵌套树”。



问题 2 DynamicRankings

N=4 的一颗嵌套树：
容易证明空间复杂度
是 $O(N\log N)$ 。



问题 2 DynamicRankings

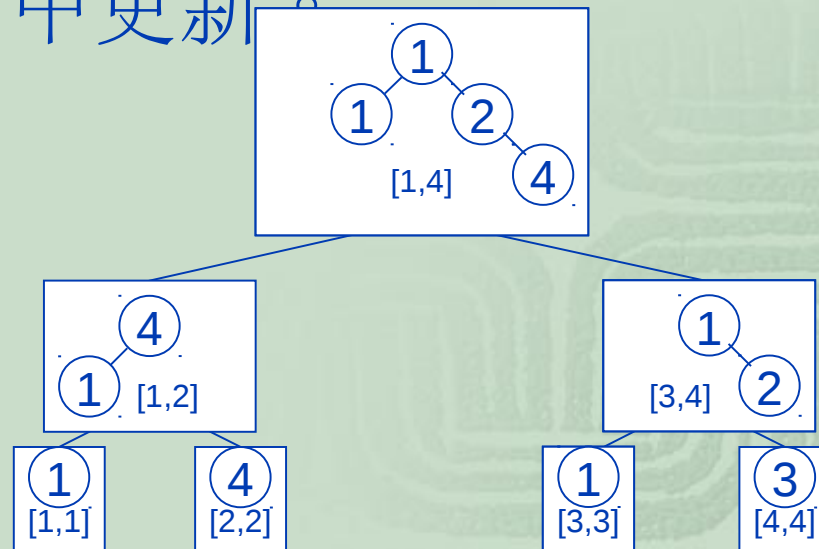
改值操作 $C(i,j)$:

在线段树中将包含 $A[i]$ 的节点找出来，再在相应的平衡树中更新。

例如 $N=4$,

$A=[1,4,2,3]$

改值 $C(3,1)$



时间复杂度 $O(\text{Log}^2 N)$ 。



问题 2 DynamicRankings

询问操作 $Q(i,j,k)$:

- 如果按照线段树求区间最大值的方法：将 $[i,j]$ 分解为若干个嵌套树中的区间并，分别求出每个子区间中第 k 大的数，再合并，这样是行不通的！
- 我们无法根据几个区间的第 k 大的数求出这些区间并的第 k 大的数。



问题 2 DynamicRankings

另辟蹊径：

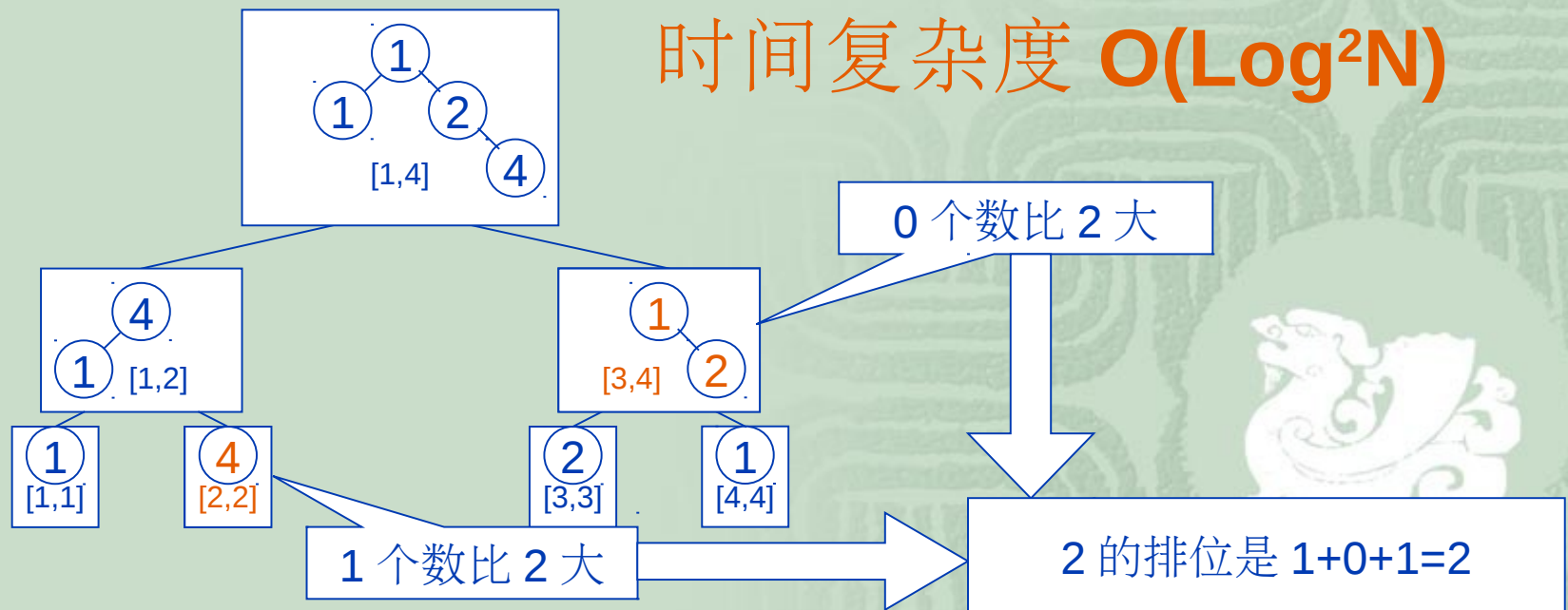
将问题逆转过来，设询问 $Q'(i,j,w)$ ，表示求正整数 w 在 $A[i], A[i+1], \dots, A[j]$ 中排第几，应该怎么做？

将 $[i,j]$ 区间在嵌套树中分解为若干子区间的并，分别求出每个区间中，有多少个比 w 大的数，最后把结果合并，就能够求出 w 在 $A[i], A[i+1], \dots, A[j]$ 的排位。

问题 2 DynamicRankings

另辟蹊径:

例如, $N=4$, $A=[1,4,2,1]$, 询问 $Q'(2,4,2)$,
即求 2 在 $A[2], A[3], A[4]$ 中的排位。



问题 2 DynamicRankings

另辟蹊径：

- w 越大，它的排位肯定越靠前。
- 我们不妨二分枚举 w ，使得 w 在 $A[i], A[i+1], \dots, A[j]$ 中排位为 k ，再求出 $A[i], A[i+1], \dots, A[j]$ 中不大于 w 的最大的数，设为 $T(i, j, w)$ 。则 $Q(i, j, k)$ 的答案就是 $T(i, j, w)$ 。



问题 2 DynamicRankings

$T(i,j,w)$ 的求法:

此时，我们可以用线段树求区间最大值的类似方法求出 $T(i,j,w)$ 。

将 $[i,j]$ 区间在嵌套树中分解为若干子区间的并。对每个子区间，找出不大于 w 的最大的数，若找不到，记答案为 $-\infty$ 。所有的子区间的答案中的最大值，就是 $T(i,j,w)$ 。

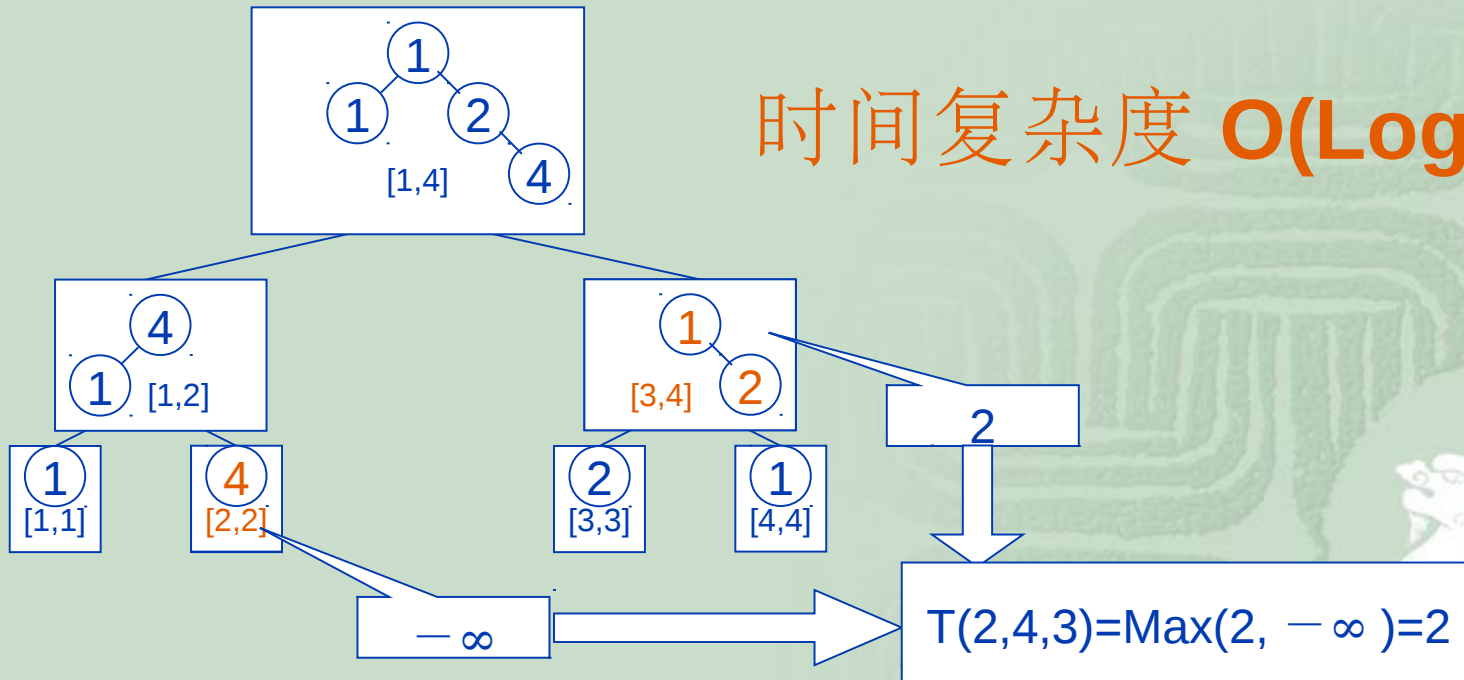


问题 2 DynamicRankings

$T(i,j,w)$ 的求法:

例如, $N=4$, $A=[1,4,2,1]$, 求 $T(2,4,3)$

时间复杂度 $O(\text{Log}^2 N)$



问题 2 DynamicRankings

回顾一下整个算法：

- 使用嵌套树作为数据结构，空间复杂度是 $O(N\log N)$ 。
- 改值操作的时间复杂度 $O(\log^2 N)$ 。
- 询问操作 $Q(i,j,k)$ ，先二分枚举求出 w ，再求出 $T(i,j,w)$ ，时间复杂度是 $O(\log \text{MaxLongint} * \log^2 N)$

整个问题的复杂度是
 $O(M \log \text{MaxLongint} \log^2 N)$ 。



数据结构的嵌套

总结:

- 上例中成功的运用了数据结构的嵌套解决了问题。
- 数据结构的嵌套，无疑是一种本质的变化，形成新的更强力的数据结构，但也有明显的缺点：空间复杂度大幅增高。
- 能不能真正发挥出数据结构嵌套的威力，无疑还是要靠我们的运用方式了。

总结

- 数据结构 + 数据结构 = 数据结构。
- 数据结构的联合，肯定不止上面提到的两种方式。
- 敏捷的思维，灵活的运用，才能将数据结构的联合应用于信息学竞赛当中。



