FOI2023算法夏令营B班
第三讲

南方科技大学 匡亮

今天的目标

- 二叉堆、树状数组、线段树
- 其他相关的数据结构或算法或技巧等等

二叉堆

- 堆和优先队列一般都指二叉堆，多叉堆在竞赛中没什么意义
- 现在我们想解决的问题是：有一个集合，支持往里面增删数字，并可以随时回答它的最大值
- 不难想到，我们还是要用二叉树来维护这些数字
- 那么为了方便，我们的目标就是要保证最大值一直落在树根上
- 这其实并不难做到：我们只要保证每个节点都比自己的俩孩子大就行了，这样根节点就比所有节点大了（到每个点都可以走出一条由大于关系构成的路）

二叉堆

- 现在来想想如何加入一个元素
- 不妨加在最后一行最靠前的位置，这样我们的树一直是一个完全二叉树（前面层全满，最后一层居左），可以保证效率
- 那么加入的这个元素可能带来哪些矛盾呢？
  - 1，比父亲小，那么没有任何矛盾
  - 2，比父亲大，比兄弟小，这种情况不可能发生，因为父亲比兄弟大
  - 3，比父亲和兄弟都大，那么和父亲交换即可
- 交换后，自己来到父亲的位置，和上面可能还有矛盾，继续调整即可。这种调整被称为向上调整(update)，复杂度为O(logn)

二叉堆

- 而要删除一个元素则没有那么简单
- 首先，二叉堆不是二叉搜索树，我们没法定位一个值对应节点的位置
- 这个问题先放一放，假设我们知道要删除哪个节点
- 如果我们指定一个节点来删除，把它的两个孩子中较大的一个作为新的父亲，继续进入孩子的子树中删除，看上去没什么问题，其实会破坏堆的完全二叉树的性质

二叉堆

- 不过我们可以想一个好办法：把要删除的节点和最后一个节点交换一下，这样删除就容易了
- 如果交换以后，被换上来的节点比父亲大，就向上调整；如果比较大的孩子小，就向下调整(rootfix)
- 这种做法分了两种情况讨论，我不是很喜欢，想想办法：其实也不是它叫我删我就必须删
- 反正只能访问根，我只要保证根是对的就行了
- 我们可以采用惰性删除：给要删除的节点打上废除标签，等它们变成根的时候再删除，这样做的一定是rootfix

二叉堆

- 然而我们现在还是在骗自己：如果指定删除一个值，我们是找不到这个要删除的节点的
- 怎么让它变得好找一点呢？想办法让它是树根！
- 沿用我们刚才惰性删除的思想，我们另外建一个堆，用来存需要删掉的元素
- 当两个堆的根节点相同时，把两个堆的根节点都删掉即可
- 这种做法还有种天然的好处：可以直接使用stl！

二叉堆

- 总结一下：
- 二叉堆有两种核心操作：用于插入的向上调整(update)和用于删除的向下调整(rootfix)
- 插入元素时，插入在最底层最后一个节点，然后向上调整
- 删除根时，和最底层最后一个节点交换，然后根向下调整
- 删除元素时，存在另一个堆里，两个堆根相同时同时删除根
- 由于这种做法只需要插入和访问根，可以非常方便地用stl
- stl提供的是大根堆，如果需要小根堆可以（如果要用来比较结构体）自定义**小于号**或（如果只是用来存数字）打个负号（推荐）

二叉堆

- 来讲一个题外话——如何初始化一个堆？
- 新建一个空堆，然后一个个插入元素，这样做的时间复杂度是O(nlogn)
- 更好的做法是：建一个完全二叉树，然后从下往上对每个点进行rootfix，这样做的时间复杂度是O(n)

对顶堆

- 如果我要固定维护第k大元素，且k偶尔会+1或-1怎么做呢？
- 用一个小根堆维护前k大，另一个大根堆维护其他元素即可
- 事已至此还是用平衡树吧……

左偏树

- 如果要合并两个堆，如何操作?
- 最简单的做法是启发式合并：把小的堆暴力拆开，一个个插入大的堆里面
- 这样看似暴力，其实对于每个元素，它每被暴力拆开一次，自身所在的集合（从小变成小+大）至少变大一倍，因此每个元素只会被暴力拆开log次，总时间复杂度是$O(n\log^2 n)$的，不过还不够快

左偏树

- 左偏树在现在的竞赛环境中用到的比较少，但是本身是一种很有意思的数据结构，推荐大家学习
- 左偏树，又称可并堆，除了和堆一样支持插入、删除、最值查询之外，还支持快速合并两个堆
- 如果一个节点没有左孩子或右孩子，我们就说它的左孩子或右孩子是空节点(Null)
- 左偏树上的每个节点除了自己的值、左右儿子指针以外，额外维护了一个值：npl(Null-Path Length)，表示自己距离最近空节点后代的距离。规定：空节点npl=-1，叶子节点npl=0

左偏树

- 除了堆的性质以外，左偏树有一个额外的性质：每个节点左儿子的npl不小于右儿子的npl
- 这个性质非常好维护：一旦你发现一个节点更新后打破了这个性质，交换它的左右儿子就好了，交换左右儿子一定不会打破堆的性质
- 维护了这个性质后，每个节点的npl值就是右儿子的npl值+1

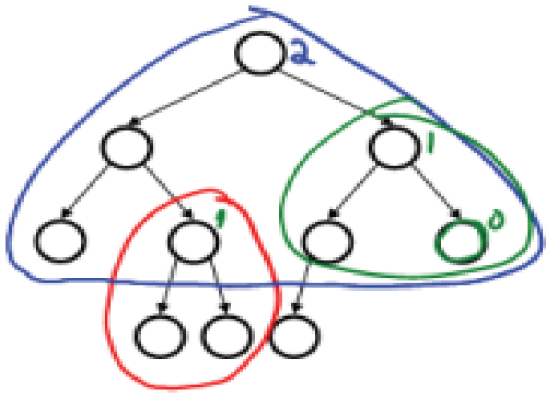
左偏树

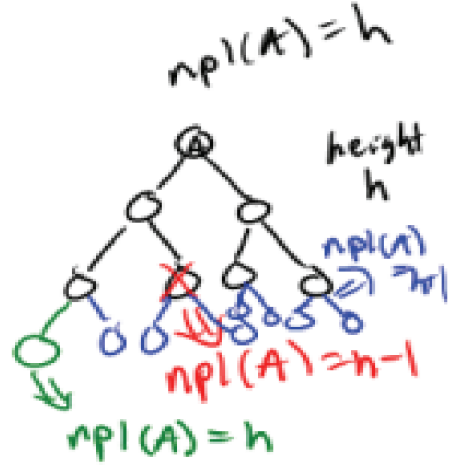
- npl值有一个非常好的性质（对所有二叉树成立）：根节点的npl值为n，则二叉树至少有2^{n+1}-1个节点，换句话说npl=O(logn)
- 这个性质非常好证明，只要考虑npl的另一个含义即可：从根开始填满的层数

Definition: Null Path Length

Another useful definition:

npl(*x*) is the height of the largest perfect binary tree that is both itself rooted at *x* and contained within the subtree rooted at x.





左偏树

- 有了以上条件以后，我们开始合并两个（大根）左偏树
- 假设我们正在合并x和y为根的两个左偏树，如果有一个是空的，直接返回另一个
- 如果两个都不是空的，那么如果x的根的值比较小就交换x和y
- 现在x的根的值比较大，它会成为合并后的根
- 递归合并x的右子树和y，结果作为x的新的右子树即可
- 递归完成后，若右子树npl更大，交换左右子树，然后更新根节点的npl值为右子树的npl值+1

左偏树

- 这个做法看上去简单又暴力，真的有那么快？
- 观察我们递归的对象会如何变化：(x,y)→(x的右子树,y)或(x,y的右子树)，取决于第一步是否交换x和y
- 递归函数本身的复杂度为O(1)
- 我们知道x的npl=x的右子树的npl+1，说明递归的两个对象的npl之和每次一定会-1
- 到递归底层的时候，两个对象的npl之和≥-1
- 我们又知道npl=O(logn)，因此只会有O(logn)次递归
- 因此合并函数的复杂度是O(logn)

左偏树

- 最后简单用合并(merge)操作代替update和rootfix来实现一下堆的基本功能：
- 查询最大值：依旧是直接看根
- 加入一个元素：新建一个堆，和目标堆merge起来
- 删除一个元素：惰性删除
- 删除根：把根的左右儿子merge起来作为新的根

树状数组

- 有些同学说：切，不如线段树
- 这样想就错了。如果唯复杂度论，那无旋treap也可以完美替代线段树
- 不论编码难度还是常数，树状数组都能爆杀线段树，能用树状数组的时候当然要毫不犹豫地用树状数组
- 当然，如果用树状数组进行操作的步骤非常复杂，还是用线段树比较好，硬上树状数组就违背我们降低编码难度的初衷了

树状数组

- 之前我们学习的大部分数据结构都是把数据看成一个个元素进行维护，比如并查集、堆；只有ST表（和猫树）是对一个序列进行维护，针对它的一个区间进行提问的
- 这种针对一个区间的问题往往比较困难，我们需要更强力的数据结构

树状数组

- 我们先从最简单的问题入手：有一个整数序列，操作是修改一个位置的元素，或者询问某个位置的前缀和
- 我们有两种暴力：要么在修改的时候O(n)算出所有位置的前缀和会变成什么，询问直接回答；要么修改直接修改，在询问的时候O(n)算出一个位置的前缀和
- 那么，有没有一种折中的方法，能让我们修改的时候修改少量的数据，回答的时候根据少量的数据拼起来呢？
- 我们很自然地想到：根据二进制位来管理元素的和

树状数组

- 例如我要求[1,13]的和，由于13=8+4+1，我就拆成[1,8], [9,12], [13,13] 来求和
- 也就是说，我的树状数组节点8管理的是[1,8]的和，12管理的是[9,12]的和，13管理的是[13,13]的和
- 这个管理的长度是如何定出来的呢？不难发现，其实是每个节点编号的最低二进制位(lowbit)：8=8，管理的长度就是8；12=8+4，管理的长度就是4；13=8+4+1，管理的长度就是1
- 查询已经搞定了，可是更新要怎么办呢？

树状数组

- 以41=32+8+1举例，它会被哪些节点管理呢？
- 首先1-40一定不会管理它，41一定会管理它
- 之后，42=32+8+2会管理它，44=32+8+4会管理它，48=32+16会管理它，64=64会管理它，128=128会管理它……
- 同样不难发现，下一个会管理自己的位置，就是自己加上自己的最低二进制位(lowbit)

树状数组

- 怎么有这样神奇的事情？道理其实非常简单：
  - 如果加上的数小于自己的lowbit，那么自己的新的lowbit就是加上的数的lowbit，小于等于加上的数，一定管不到自己
  - 如果加上的数等于自己的lowbit，那么自己的新的lowbit至少是原来的lowbit的两倍，一定可以管到自己
- 借此我们就得到了树状数组的具体实现：
  - 修改a[x]+=b时，不断执行t[x]+=b; x+=lowbit(x)，直到x超过n
  - 查询sum[x]时，ans=0，不断执行ans+=t[x]; x-=lowbit(x)，直到x=0，返回ans
- 两个操作的时间复杂度都是O(logn)，且常数极小

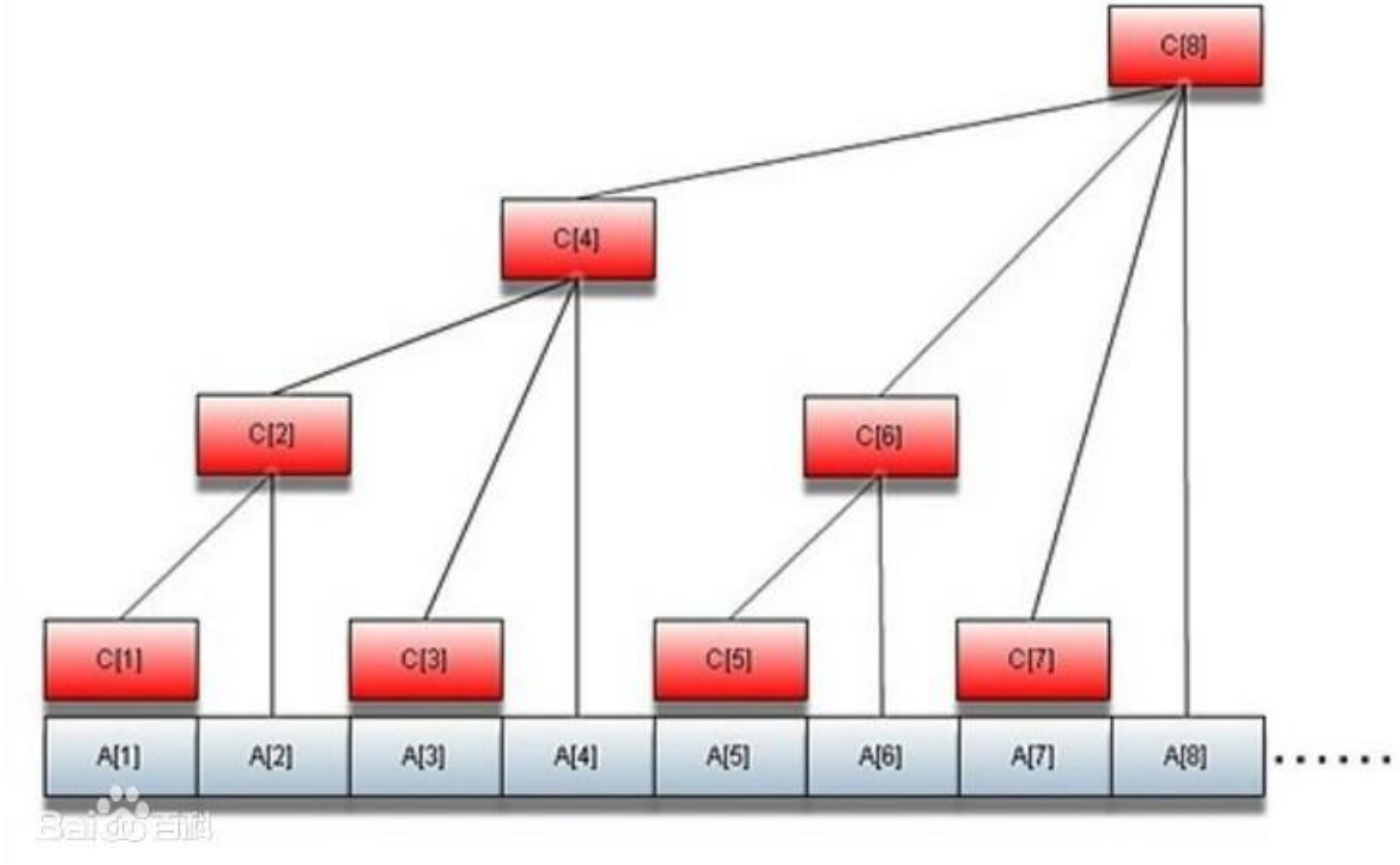
树状数组

- 那么，怎么求lowbit(x)呢？
- 在计算机中，负数是用补码的形式存储的，其中补码=反码+1
- 有符号数的最高位为符号位，非负数为0，负数为1
- 假设我们用8位二进制码表示有符号数（如果拓展到32位或64位，前面全部补符号位即可）
- 例如+0是00000000，反码是11111111，补码是00000000，因此-0也是00000000
- 例如+12是00001100，反码是11110011，补码是11110100，因此-12是11110100

树状数组

- 例如+12是00001100，反码是11110011，补码是11110100，因此-12是11110100
- 可以发现，反码刚好把第一个1改成了0，把低位的0改成了1，那么补码在反码的基础上+1，就把第一个1改回了1，低位的0又变回了0，高位则和反码相同
- 于是lowbit(x)=x&(-x)

树状数组



树状数组

- 我们用树状数组解决了单点修改，求前缀和的问题
- 那么求区间和是否也能做呢？很简单，两个前缀和相减即可
- 然而，并不是所有问题的答案都能用前缀和相减得出区间答案
- 例如我们用ST表解决的RMQ问题，就不是一个可减的问题：
  - 如果我告诉你[1,9]的最小值为9，[1,4]的最小值为9，你能告诉我[5,9]的最小值是多少吗？不可能
- 因此，树状数组的限制除了单点修改以外，还有问题可减（除非你只问前缀答案不问区间答案）

树状数组

- 树状数组还有很多高超的技巧，但我认为那些都已经超出了我们使用树状数组的初衷，就不再分享了
- 建议大家只使用树状数组解决单点修改+区间询问（答案可减）/前缀询问
- 只补充一个小技巧：区间修改+单点询问。这种情况下，我们用树状数组维护原数组的差分数组：s[i]=a[i]-a[i-1]。区间修改变成对s[l]和s[r+1]的单点修改，单点询问变成对s[1~x]求前缀和，刚好可以用最基本的树状数组解决。这个比写线段树的性价比高不少

线段树

- （线段树有很多种大同小异的写法，这里推荐的是我自己的写法，我认为非常简单而且不容易写错；但如果大家已经有习惯的写法，不需要改成我这种）
- 线段树是一种更强力的数据结构
- 之前我们在用二叉树（堆、并查集）的时候，每个节点自己本身也是一个元素，树结构本质上是元素之间的关系
- 但现在我们关心的不是元素之间的关系，而是序列上每个区间的信息，因此可以想到：把区间作为线段树的节点

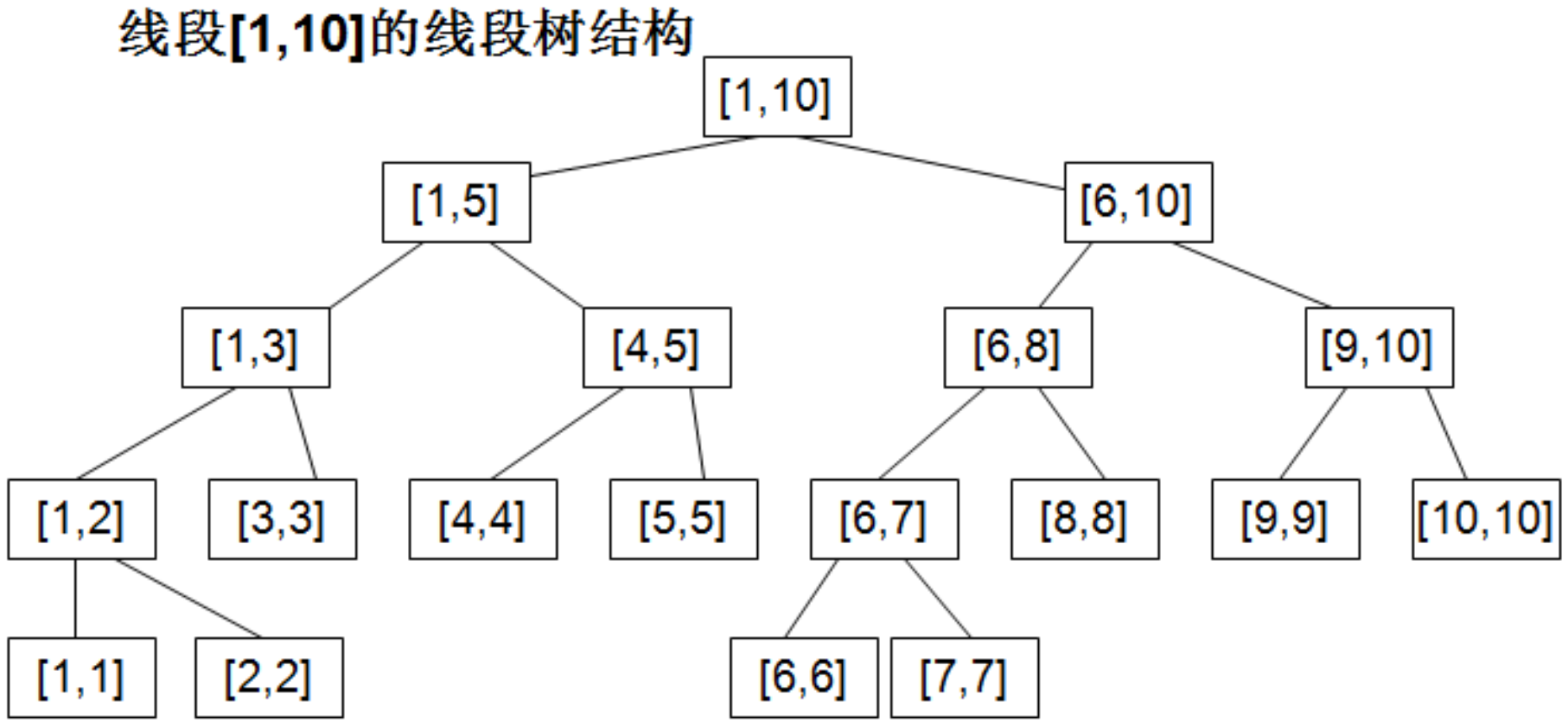
线段树

- 根节点就设为1号节点，用来存区间[1,n]
- 左右孩子当然越平均越好，那么设mid=(l+r)>>1，左孩子的区间就是[l,mid]，右孩子的区间就是[mid+1,r]；l=r的区间就作为叶子
- 和树状数组类似，我要管理一个区间的信息，这个信息就直接记录在节点上
- 不过，记录左右儿子的编号有点太麻烦了，我们不妨把它扩充成满二叉树：nod的左儿子是(nod<<1)，右儿子是(nod<<1|1)

线段树

- 我们知道，树状数组的数组大小就是n，而线段树呢？
- 正确答案是2-4倍之间——**不是说2-4倍都可以**，而是4倍一定够，而如果你会算的话也可以适当开小一点：
- 我们使用满二叉树，本质上是把n扩大到了下一个2的次幂，这个数一定是小于2n的；然后我们知道满二叉树节点数=2倍底层节点数-1，所以总节点数一定是小于4n的
- 例如n=100,000，下一个2的次幂是2^{17}=131072，两倍是26万多，我会开27万长度的数组（省一点就够了，别开的太紧）；当然也可以无脑开40万长度

线段树


线段[1,10]的线段树结构
[1,10]
[1,5]
[6,10]
[1,3]
[4,5]
[6,8]
[9,10]
[1,2]
[3,3]
[4,4]
[5,5]
[6,7]
[8,8]
[9,9]
[10,10]
[1,1]
[2,2]
[6,6]
[7,7]

线段树

- 现在我们来做单点修改，区间求和问题
- 这次修改变得简单多了：我们从根节点开始往下走，如果要修改的点在左子树，就进入左子树，否则进入右子树
- 一路走到叶子节点，修改它的值
- 然后在回到父亲的时候，更新一下父亲记录的答案（sum[nod]=sum[ls]+sum[rs]），直到回到根节点，结束

线段树

- 那么怎么做区间求和呢？假设我们询问的是[ql,qr]
- 如果l=ql,r=qr，问的就是这个区间，直接返回t[nod]
- 如果qr≤mid，整个区间都在左子树，直接进入左子树回答
- 如果ql>mid，整个区间都在右子树，直接进入右子树回答
- 否则，左子树回答[ql,mid]，右子树回答[mid+1,qr]，然后加起来返回
- https://www.luogu.com.cn/problem/P3374

线段树

- 修改的复杂度很显然：树高是O(logn)的，每一步是O(1)的，总复杂度就是O(logn)
- 询问的复杂度其实也很显然：
- 在询问的区间裂开以前，每个节点只会访问一个儿子，总共不超过树高次，复杂度是O(logn)的
- 在询问的区间裂开以后，它会在左子树询问一个后缀和，右子树询问一个前缀和
- 询问前缀和是O(logn)的，因为每次要么进入左子树，要么完全包含左子树然后进入右子树；同理询问后缀和也是O(logn)的

线段树

- 太棒了，我们现在掌握了一种O(logn)单点修改+区间查询的数据结构
- 但是……这玩意也不比树状数组厉害啊

线段树

- 太棒了，我们现在掌握了一种O(logn)单点修改+区间查询的数据结构
- 但是……这玩意也不比树状数组厉害啊
- 错错错！就算没有lazytag，线段树也比树状数组厉害：我们不再需要答案可减这个条件了（比如单点修改区间最值就可以做了）
- 不过为了让我们的线段树更厉害，lazytag是必不可少的一个DLC
- 不过进入这个话题之前，先来做一道例题巩固一下普通线段树

线段树

- 一个序列（整数，有正有负）的最大子段和是指从中选出一个子区间求和可以得到的最大值
- 给出一个序列，要求支持：单点修改，区间查询最大子段和

线段树

- 问题不过在于：如果我们的询问裂开了，如何把一个左子树的后缀答案和右子树的前缀答案合并起来
- 如果我们目标的区间完全处于左子树或者右子树，那么没有问题，答案的最大值就是答案
- 如果横跨了两个区间，那么我们需要左子树的最大后缀和和右子树的最大前缀和，加起来得到横跨区间的答案
- 最终合并后的答案=max(左子树答案，右子树答案，左子树的最大后缀和+右子树的最大前缀和)

线段树

- 于是我们现在需要在节点额外维护最大前缀和和最大后缀和
- 由于更新会进行到叶子，更新非常简单，主要是合并
- 思考合并后的最大前缀和出现在左子树还是右子树
- 合并时，最大前缀和=max(左子树的最大前缀和，左子树总和+右子树最大前缀和)，最大后缀和同理
- 因此我们还要还要再维护一个子树的总和，这个简单
- 时间复杂度O(nlogn)

线段树

- 现在我们来开启lazytag篇章
- 先来解决区间赋值，区间求和问题
- 区间求和变化其实不大，依旧是O(logn)个区间的答案加起来
- 但是区间赋值，我们肯定不能暴力访问每个节点，只能和区间求和一样拆成若干个区间进行修改
- 和求和一样，拆到最后，有O(logn)个节点是完全处于我们的复制操作范围内的，我们希望对它们O(1)操作

线段树

- 先往好处想：假设从此以后我再也不会访问它的子节点
- 那么我的操作可以到此为止，把这个区间的答案更新为k*len，其中k是我们要赋的值，len是区间的长度(r-l+1)
- 然而现实不会这么美好：我们之后可能会访问它的子节点
- 所以我们在节点上打一个tag=k，表示我上次有一个赋值操作进行到这里停止了，还没有修改孩子们
- 等我下一次访问到这个节点的时候，如果还要访问这个节点的孩子，那么就得先进行一下标记上的操作

线段树

- 由于自己的两个孩子是被自己完全包含的，对它们进行操作一定是直接更新答案、打tag，这一步是O(1)的
- 下放tag后，删掉自己的tag，然后就可以放心访问孩子了
- 只剩下一个问题没有思考：孩子接收到标签时如果自己有标签怎么办？
- 其实问题也不大，这题里我们的标签指的是“发生了一次区间赋值还没对孩子执行”，因此直接覆盖即可

线段树

- 就这样，我们通过名为lazytag的魔法，把修改和询问都控制在了O(logn)的复杂度
- 当初教我线段树的学长打了一个这样的比方：
- Lazytag就好比拖欠作业：
  - 1，你得让老师看上去你好像做完了作业，即当你用打tag代替完成每份作业时，你得快速更新出区间答案，就好像你做完了每份作业一样
  - 2，当老师细究起每份作业的时候，你也能快速完成，即当老师试图访问你这个作业区间的孩子的时候，你能够快速下传标记（更新出要访问的孩子的答案）
- 用向量来理解线段树：https://www.luogu.com.cn/blog/wangrx/ji-seg-tree

线段树

- 至此，总结一下我们需要做的：
  - 1，用少量信息记录下一个区间的答案和辅助信息
  - 2，用在节点上打标记代替修改区间内的每个元素
- 也就是，快速合并答案、快速打标记、快速下传标记
- 快速合并答案总是不难的
- 快速下传标记=快速给两个孩子打上标记，所以难度等于快速打标记
- 因此最难的问题就是快速打标记，需要注意的是，**被打标记的节点可能已经有标记了**，因此困难的其实是快速合并标记

线段树例题1

- 区间加法、区间求和。
- 快速合并答案？sum[nod]=sum[ls]+sum[rs]
- 快速打标记？sum[nod]+=len[nod]*val, tag[nod]+=val

线段树例题2

- 区间加法、区间乘法、区间求和
- 快速合并答案？sum[nod]=sum[ls]+sum[rs]
- 快速打标记？我们相当于在考虑：如何合并乘法操作和加法操作变成乘加操作、如何合并乘加操作。
- **由于乘法对加法满足分配律**，我们想把所有操作变成先乘后加。
- *a+b然后*c+d等于*ac+(bc+d)

线段树例题2

- 区间加法、区间乘法、区间求和
- 快速合并答案？sum[nod]=sum[ls]+sum[rs]
- 快速打乘法标记？tag2[nod]*=val, tag1[nod]*=val, sum[nod]*=val
- 快速打加法标记？tag1[nod]+=val, sum[nod]+=len[nod]*val
- 如果下传标记到一个有标记的节点，先下传乘法标记，后下传加法标记

线段树例题3

- 单点修改、区间取模、区间求和
- 快速合并答案？sum[nod]=sum[ls]+sum[rs]
- 快速打取模标记？……
- 我们遇到了问题：对区间内的每个元素取模，他们加起来的结果不知道会变化多少，不满足我们“对整个区间快速打标记代替操作”的要求
- 我们需要发掘更多细节

线段树例题3

- 如果区间内所有数都小于模数，什么事都不会发生
- 如果一个数被有效地取模，它至少会减小一半
- 我们只有单点修改
- 因此，有效取模次数=O(nlogx)
- 于是我们可以记录每个区间的最大值，每次访问到一个区间，如果最大值小于模数就返回，否则左右子区间都访问

线段树例题3

- 最后每次修改都是单点修改，且一定是有效取模，一次有效取模带来的代价是O(logn)（从根节点访问它）
- 时间复杂度 O(nlognlogx)，其中n是数组长度和操作次数，x是值域
- 线段树相当于在辅助我们进行快速精准地单点修改

线段树例题4

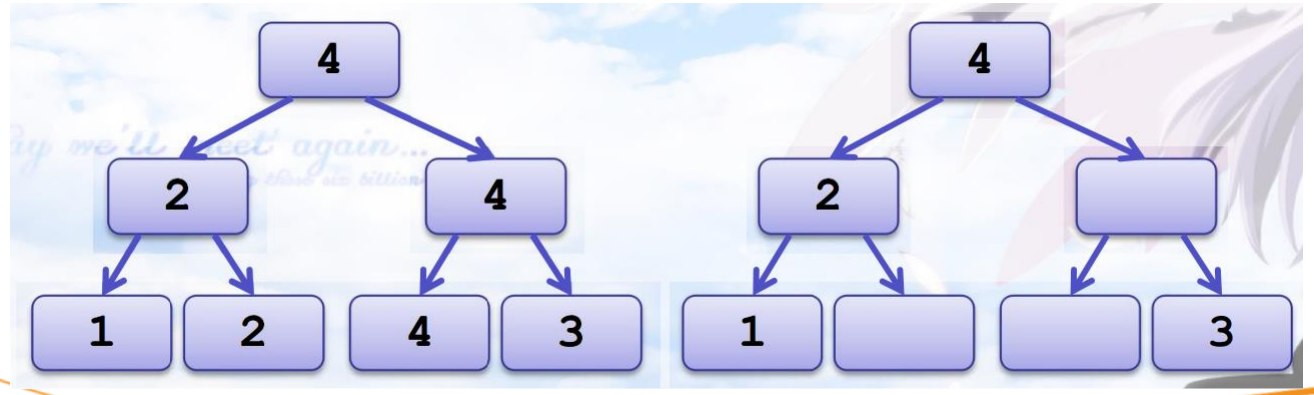
- 给定一个长度为n的数列A，接下来有m次操作：
- 区间[l,r]中的所有数变成min(Ai, x)
- 区间[l,r]中的所有数加上x（x可能是负数）
- 询问区间[l,r]中所有数的和
- 快速合并答案
- 快速打标记……

线段树例题4

- 受上一题启发，我们想在节点上维护一个区间最大值mx，如果最大值都小等于取min的数x，就可以跳过这次操作
- 很可惜，每次取min最坏情况下只会导致当前值-1，更别提我们还有区间加法的操作，所以我们还是要想办法区间操作一些东西的
- 于是我们想到，再记录一个区间次大值se，再记录一下最大值出现的次数t，那么如果se<x<mx，修改mx和区间答案即可
- 如果小于等于se，进入左右子树暴力
- 复杂度如何计算呢？

线段树例题4

- 我们需要换一个角度看这个问题：
- 把我们维护的最大值mx看成一个全局取min的标记
- 如果某个节点的标记和父亲相同，无视它
- 那么我们维护的次大值se就是子树内（不含自己）的标记的最大值
- 每个点的真实值是头上最近的标记的值



线段树例题4

- 考虑每次暴力会终止在什么节点上
- 当我们访问到一个节点，它的子树（不含自己）内没有一个标记大于等于x，那么se<x，暴力一定会立刻结束
- 否则，暴力将找到所有大于等于x的标记，终止在所有存放这些标记的节点上（如果走入了不含大于等于x的标记的子树会立刻结束）
- 暴力结束以后，这些节点的标记会被修改为x，暴力开始位置的标记也会被修改为x，因此这些节点的标记会消失（被无视）
- 因此，暴力过程本质上是在**回收子树内大于x的标记**

线段树例题4

- 区间加减法时，我们打标记（和基本线段树无差别）的复杂度一共是O(nlogn)
- 因此回收标记的复杂度不会超过O(nlogn)
- 总时间复杂度 O(nlogn)
- 来自：（补充阅读）Segment Tree Beats! - 罗哲正、吉如一

线段树例题5

P6242 【模板】线段树 3

提交答案 加入题单

题目背景

复制Markdown 展开

本题是线段树维护区间最值操作与区间历史最值的模板。

题目描述

给出一个长度为 n 的数列 A，同时定义一个辅助数组 B，B 开始与 A 完全相同。接下来进行了 m 次操作，操作有五种类型，按以下格式给出：

- `1 l r k`：对于所有的 $i \in [l, r]$，将 A_i 加上 k（k 可以为负数）。
- `2 l r v`：对于所有的 $i \in [l, r]$，将 A_i 变成 $\min(A_i, v)$。
- `3 l r`：求 $\sum_{i=l}^{r} A_i$。
- `4 l r`：对于所有的 $i \in [l, r]$，求 A_i 的最大值。
- `5 l r`：对于所有的 $i \in [l, r]$，求 B_i 的最大值。

在每一次操作后，我们都进行一次更新，让 $B_i \leftarrow \max(B_i, A_i)$。

线段树例题5

- 维护“历史最大值”，我们需要维护一个“历史最大标记”，表示从上一次修改至今，标记的最大值是多少。
- 假设标记为 add，历史最大标记为 add’，那么下传标记时，有 add’[son] = max(add’[son], add[son]+add’[fa]) 。

总结

- 线段树部分，我们先一起学习了最简单的单点修改区间求和线段树，初步了解了线段树的工作机制。
- 然后通过区间修改区间求和线段树了解了lazy tag这一强大的工具
- 然后通过区间取模线段树了解了暴力 dfs 一个节点后如何自圆其说地算复杂度
- 最后简单了解了维护历史信息的 lazy tag

主席树

- 我们知道，对线段树进行单点操作时，只会修改从根到叶子的log个节点。
- 如果我们不选择修改，而是在原基础上新建，我们就可以通过访问不同的根回到任意一个历史版本。
- 如果是简单的区间操作，我们思考如何将“下传标记”改成“永久化标记对询问的贡献”。

P3919 【模板】可持久化线段树 1（可持久化数组）

提交答案 加入题单

题目描述

如题，你需要维护这样的一个长度为 N 的数组，支持如下几种操作

1. 在某个历史版本上修改某一个位置上的值

2. 访问某个历史版本上的某一位置的值

此外，每进行一次操作（**对于操作2，即为生成一个完全一样的版本，不作任何改动**），就会生成一个新的版本。版本编号即为当前操作的编号（从1开始编号，版本0表示初始状态数组）

对于100%的数据：$1 \leq N, M \leq 10^6, 1 \leq loc_i \leq N, 0 \leq v_i < i, -10^9 \leq a_i, value_i \leq 10^9$

Elegia 更新时间：2018-01-10 14:13:51 在 Ta 的博客查看

虽然这违反了模板的本意，但是本人认为这种做法也会对应于一类特别的题目。即考虑离线做法。

考虑到每一个版本都被一些版本所依赖，这种依赖关系可以被当做一棵树。在读入时建树，最后进行操作时就是将树从原点0处进行dfs。

对于询问直接存进答案数组，对于修改操作，进行修改后再dfs，结束后再撤回修改。

这种思路可以过掉部分题目，但前提是操作可逆，所以不能过掉诸如可持久化并查集的题目。

```
void dfs(int u) {
    if (op[u] == 2) {
        ans[u] = a[k[u]];
        for (edge* p = g[u]; p; p = p->next)
            dfs(p->v);
    } else {
        int ori = a[k[u]];
        a[k[u]] = x[u];
        for (edge* p = g[u]; p; p = p->next)
            dfs(p->v);
        a[k[u]] = ori;
    }
}
```

正经主席树做法

- 用主席树维护整个序列即可，非叶节点不需要记任何孩子的信息，只需要知道孩子是谁即可。
- 修改时调用原来的根，生成新的根，然后一路访问到叶子，总是只有一个孩子被修改；复制并新建被修改的孩子，继承不被修改的孩子即可。
- 时空复杂度都是一个log。

P3834 【模板】可持久化线段树 2

提交答案 加入题单

题目背景

这是个非常经典的可持久化权值线段树入门题——静态区间第 k 小。

数据已经过加强，请使用可持久化权值线段树。同时请注意常数优化。

题目描述

如题，给定 n 个整数构成的序列 a，将对于指定的闭区间 $[l,r]$ 查询其区间内的第 k 小值。

输入格式

第一行包含两个整数，分别表示序列的长度 n 和查询的个数 m。
第二行包含 n 个整数，第 i 个整数表示序列的第 i 个元素 a_i。
接下来 m 行每行包含三个整数 l,r,k , 表示查询区间 $[l,r]$ 内的第 k 小值。

正经主席树做法

- 如果没有区间询问，我们可以把所有数插入一个权值线段树，每个节点记录自己管辖的区域内有多少个数，之后**在线段树上二分**即可。
- 注意在线段树上二分是直接利用线段树节点做的二分，复杂度是一个log，如果二分再上树查找就会变成两个log。
- 现在有了区间询问，由于“个数”是一个非常简单的“可减”信息，我们可以把原序列加入一个数看成生成一个新的版本。
- 用主席树维护从1到N加入数的过程，查询时同时访问主席树的R和L-1版本，根据两个版本某区间的个数差是否大等于k在主席树上二分即可。

P7424 [THUPC2017] 天天爱射击

提交答案 加入题单

题目描述

展开

小 C 爱上了一款名字叫做《天天爱射击》的游戏。如图所示，这个游戏有一些平行于 x 轴的木板。现在有一些子弹，按顺序沿着 x 轴方向向这些木板射去。第 i 块木板被 S_i 个子弹贯穿以后，就会碎掉消失。一个子弹可以贯穿其弹道上的全部木板，特别的，如果一个子弹触碰到木板的边缘，也视为贯穿木板。

小 C 现在知道了游戏中 n 块木板位置，以及知道了 m 个子弹射击位置。现在问你每个子弹射出去以后，有多少木板会碎掉?

正经主席树做法

- 与其考虑每个子弹击碎哪些木板，不如反过来枚举每块木板被哪个子弹击碎
- 我们将子弹的位置作为下标、时间作为值插入主席树，木板i会被[Li,Ri]上的第Si个子弹击碎，问题就变成求[Li,Ri]内的第Si小值

小结

- 今天只介绍了最简单的主席树用法
- 主席树很多时候会和整体二分这个离线算法大牛一起使用，但因为我们今天没有涉及这个算法，只能请大家课后有兴趣了解了
  - 整体二分入门+与主席树的辨析：**带修**区间第k大
  - 整体二分+主席树例题：[CTSC2018]混合果汁
- 另外，主席树的大常数和大空间还是不太讨喜，有些时候我们会用CDQ分治代替主席树（和其他高维数据结构）

线段树分治

- 线段树分治又称按时间分治
- 按时间分治是对于一系列“加入或删除”的操作，当“加入”比“删除”实现起来容易非常多时，我们可以将所有操作离线，将操作顺序看成“时间”后，用线段树来维护“时间”。
- 我们把加入和删除配对，即可得到某个元素的“存在时间”，在线段树的对应节点插入这个元素，访问结束后“撤销加入”即可。时间复杂度在原基础上乘一个log。

#121. 「离线可过」动态图连通性

传统 | 800 ms | 512 MiB | 显示标签

题目描述

这是一道被离线爆$^{++}$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。

离线做法

- 将操作离线，得到一条边的存在时间。
- 建立时间线段树即可。注意此时不能路径压缩，必须启发式合并（按秩合并），复杂度$O(n\log^2 n)$。

注意以下这题强制在线（回答一个问题后才知道下个操作）

#534. 「LibreOJ Round #6」花团

传统 | 3000 ms | 256 MiB | 显示标签

题目描述

> 「Alice —— !」 「Karen —— !」
>
> Alice 和 Karen 家边的大花坛给了她们无尽的欢乐。
>
> 这天 Karen 想重新规划一下花坛在一年里的外观。但是由于花朵各有其花期，而且花市上的选择实在太多了，所以她把问题进行了一些抽象，希望擅长程序设计的你可以为她解决。

物品集合 S 初始为空，按时间递增顺序依次给出 q 次操作，操作如下：

- `1 v w e` 表示在 S 中加入一个体积为 v 价值为 w 的物品，第 e 次操作结束之后移除该物品。
- `2 v` 表示询问。你需要回答：
  1. 当前 S 是否存在一个子集使得子集中物品体积和为 v。
  2. 当前 S 的所有物品体积和为 v 的子集中，价值和最大是多少（空集的价值和为 0）。

对于所有数据，$1 \le q \le 15000, 1 \le v_i \le \text{maxv} \le 15000, 0 \le w_i \le 15000, i \le e_i \le q$。

假在线做法

- 虽然强制在线看起来很可怕，但其实每个物品什么时候会消失是在加入时就知道的，已经满足我们使用线段树按时间分治的条件了，直接套用即可
- 当我们在叶子获得一个加入操作的真实信息时，把它补到线段树分治上即可，受影响的区间一定都在自己后面
- 不过dp数组的撤销太难了，还不如建一个新的。所以我们一层用一个dp数组，进入孩子节点的时候拷贝，退出孩子节点的时候不用管
- 时间复杂度O(qvlogq)

注意以下这题强制在线（回答一个问题后才知道下个操作）

时雨正在你的房间——提督室里面观雨。她将一个盛着少许水的纸杯放在了屋檐下，雨水顺着屋檐滴进纸杯内。时雨对盛着雨水的纸杯起了兴趣。她将下雨的过程划分成 n 个时段，每一个时段会有两种可能：

1. 更多的雨水滴进了纸杯里，纸杯中的水量变为原来的 c 倍。

2. 时雨决定倒掉一部分水。如果将到目前为止操作 1 中所有的 c 组成的集合记为 C，那么时雨会将将纸杯中的水量除以 x 且保证 $x \in C$

时雨想仔细记录下每个时刻水杯的水量，但她仅仅是执行操作就力不从心了，于是在一旁处理完事务的你决定帮助时雨。同时，为了简化统计量，你和时雨决定将水量对某个数 p 取模。

注意：在时刻 1 之前，纸杯中的水量为 1

真在线做法

- 如果离线，就是按时间分治的模板题
- 因此我们可以思考这题比动态连通性简单在哪里，以致于它可以强制在线：贡献可以快速合并，且贡献撤销不需要按顺序
- 因此我们可以用线段树维护时间，在第i时刻插入数就在线段树上插入，删除数就找到它插入的时间，单点修改成1；询问则输出[1,i]的区间积

题意翻译

给出一个连通带权无向图,边有边权,要求支持q 个操作:

$1\ x\ y\ d$ 在原图中加入一条x 到y 权值为b 的边

$2\ x\ y$ 把图中x 到y 的边删掉

$3\ x\ y$ 表示询问x 到y 的异或最短路

保证任意操作后原图连通无重边自环且操作均合法

$n, m, q \leq 200000$

离线做法

- 和动态图连通性差不多，多套了一个最小（最大也能做）异或和路径的板子。
- 维护一棵生成树，加入的如果是非树边则将环的异或和加入线性基。（参考[WC2011]最大XOR和路径）
- 虽然从线性基中撤销一个数不可能做到，但线性基本身很小，直接备份整个版本即可。

luogu.com.cn/problem/CF813F

题意翻译

给你一个由n个顶点组成的无向图，最初在图中没有边。同时给你q次查询，每次查询时会向图中添加一个无向边或者删除一个无向边。 在每次查询之后，您必须检查结果图是否为二分图（在保证没有连接相同颜色的两个顶点的边的条件下，您可以将图的所有顶点绘制为两种颜色）

离线做法

- 按老套路离线，判断是否是二分图可以参考“NOI2001食物链”，用一个扩展域并查集维护“颜色不同”这个信息

题意翻译

给定一张 n 个点 m 条边的无向图。

一共有 k 种颜色，一开始，每条边都没有颜色。

定义**合法状态**为仅保留**染成 k 种颜色中的任何一种颜色**的边，图都是一张二分图。

有 q 次操作，第 i 次操作将第 e_i 条边的颜色染成 c_i。

但并不是每次操作都会被**执行**，只有当执行后仍然合法，才会执行本次操作。

你需要判断每次操作是否会被执行。

$n, m, q \leq 5 \times 10^5$，$k \leq 50$。

半离线做法

- 每条边不一定会被染色→每条边可能被染色或被保持，这样的话我们仍旧可以将操作按时间分治。
- 发生在x时刻的操作决定了一条边在[x+1,y]的颜色（y是下一次操作同一条边的时刻），于是我们可以在处理过x（发生在某个叶子）后确定颜色，此时一定还未发生[x+1,y]的加入操作，其他就和上一题一样了。

小结

- 操作为“加入”、“删除”时，可考虑线段树分治
- 操作为“修改”时，可转化为“删除后加入”后考虑线段树分治
- 有些情况下，不需要完全离线
- 有些情况下，直接用在线线段树维护时间

总结

- 今天主要讲了两块内容（带*号下午一定不考）：
- 1，堆：普通堆（update，rootfix），左偏树（merge，npl）
- 2，树状数组：用法、证明、lowbit
- 3，线段树：lazytag，历史标签*，复杂度分析*，在线段树上二分
- 4，主席树
- 5，线段树分治

SUSTech Southern University of Science and Technology

谢谢大家！