FOI2022算法夏令营基础班DAY4 数据结构-2

王文铎

优先队列

可以认为是一种特殊的队列，每次取出时会取出队列中的最大（最小）值（而不是队头）。

与队列不同的是，插入和删除的复杂度都是O(logn)。

STL中的优先队列

```
#include <queue>          //头文件
priority_queue <T> q;     //每次取出最大值
//priority_queue <T, vector <T>, greater <T> > q;
q.push(a);                //将a加入队列
q.top();                  //返回最大元素
q.pop();                  //将最大元素出队列
q.empty();                //返回队列是否为空
q.size();                 //返回队列内元素个数
```

堆

优先队列一般使用堆来实现。

完全二叉堆在形态上为一颗完全二叉树。

以大根堆为例，根是所有节点中最大的。

每个节点上的值都比它的儿子更大。

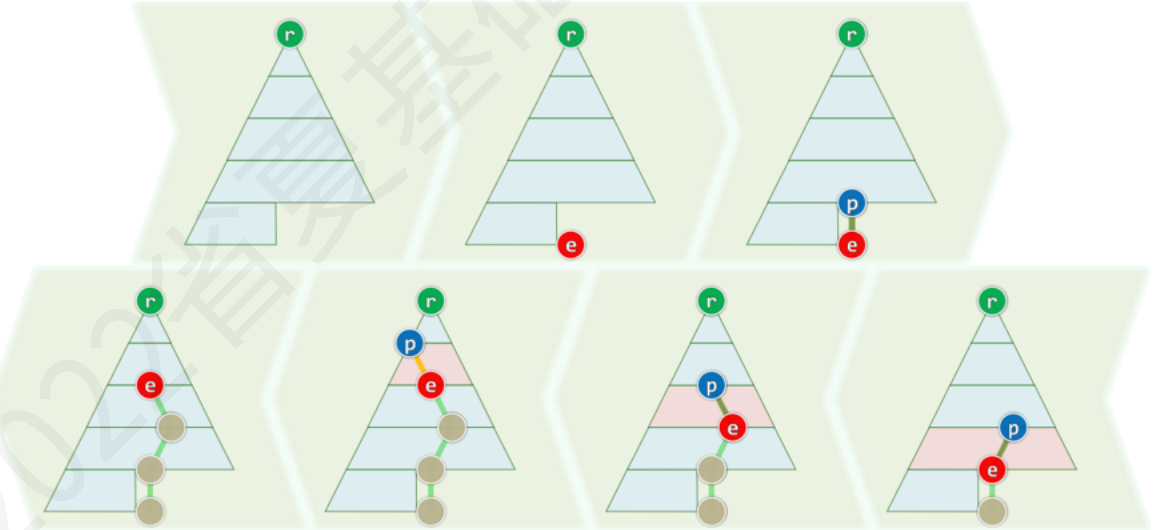
堆：插入

堆的插入为逐层上滤。

首先插入最后保持完全二叉树的形态。

不断与父节点比较，如果更大则交换。

复杂度O(logn)。



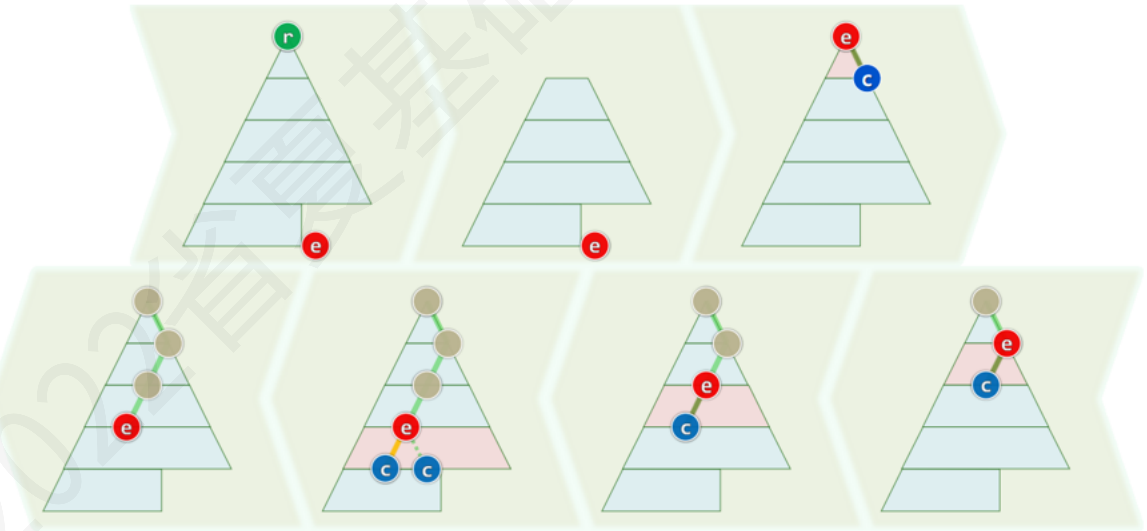
堆：删除

堆的删除为逐层下滤。

首先用最后节点替换根节点以保持完全二叉树形态。

不断与儿子中较大的比较，如果更小则交换。

复杂度O(logn)。



一道例题

有一个集合，初始为空。三种操作：

1. 向集合中添加一个数。
2. 删除集合中一个数（保证合法）。
3. 查询最大值。

一道例题

没有删除就是堆的模板题。

如果删除堆中一个确定位置的值，可以用最后一个节点替换，然后上下交换来调整（就像插入删除一样）。

本题需要删除一个值而不知道位置。

一般做法是另外建一个堆记录需要删除的值。每当两个堆顶相同时就删除。

幼儿园

有n个小朋友被分配在不同的幼儿园，每个小朋友有一个权值。

有q次转校事件，每一次有一个小朋友转到另一个幼儿园。

称有小朋友在的幼儿园的等级为拥有小朋友的权值最大值。

每次转校后询问所有有小朋友在的幼儿园的等级最小值。

n, q<=2e5

幼儿园

每个幼儿园建堆记录在这所幼儿园的小朋友的权值。

转校时将对应权值删除（利用额外的删除堆）。

再对所有幼儿园的堆顶（也就是这个幼儿园的等级）建一个堆。

每次转校只会更改其中两个值，相当于两次删除和两次增加。

灯泡

有 n 个房间和 n 盏灯，你需要在每个房间里放入一盏灯。每盏灯都有一定功率，每间房间都需要不少于一定功率的灯泡才可以完全照亮。

你可以去附近的商店换新灯泡，商店里所有正整数功率的灯泡都有售。但由于背包空间有限，你至多只能换 k 个灯泡。

你需要找到一个合理的方案使得每个房间都被完全照亮，并在这个前提下使得总功率尽可能小。

$n \leq 5\times 10^5$

灯泡

需要有**n-k**个房间要换上已有的灯泡，我们需要让这部分总功率-要求功率之和尽量小。显然可以贪心每次找差值最小的一组灯泡和房间。

将房间要求值和灯泡功率放在一起排序，显然只有相邻的房间和灯泡才有可能成为答案。

类似括号序列，将房间当作左括号，灯泡当作右括号。用栈找出配队在一起的一组，将差值加入堆中，再取堆的前**n-k**个最小值即可。

并查集

并查集可以维护若干个元素的集合。

初始时每个元素各自属于一个集合。

支持两种操作：

1. Union 合并两个集合

2. Find 查询一个元素所在集合的一个代表（同一个集合查询结果相同）。

并查集

定义数组fa，表示每个集合所指向的元素。

初始时fa[i]=0或fa[i]=i，这意味着i所在集合代表元是自己。

合并u, v所在集合可以设两个集合代表元为fu, fv，令fa[fu]=fv。

查询时沿着fa方向走，找到满足fa[i]=0或fa[i]=i的点即可。

路径压缩

时间复杂度？最差情况形成一条链！

考虑查询时将路上经过的点直接连到代表元处，之后查询可以快速结束。

这是最常用的优化方法！

```
int getf(int u) { return fa[u] ? fa[u] = getf(fa[u]) : u; }
```

按子树大小合并

有一些题目需要统计每个集合的大小，可以在合并时统计。

初始令siz[i]=1， 当fa[fu]=fv时令siz[fv]+=siz[fu]即可。

- 如果记录了siz，则合并fu与fv时将siz小的连向siz大的。
- 这样查询时每向上一层子树大小至少*2，故复杂度为O(logn)。
- 这种优化允许再撤回原先某次连接同时保证复杂度。

按秩合并

类似地，我们维护rank数组代表集合的深度（秩）。

合并时将rank小的连接到rank大的。

将按秩合并与路径压缩同时使用，可能会使rank数组不准确，这时复杂度为O(log*n)。可以认为是O(1)。

SWAP AND SORT

给出一个1-n的排列以及m种操作，每个操作是一个二元组(a_i, b_i)。每次操作可以选择一个二元组，交换(P_{a_i}, P_{b_i})。

问是否能通过交换将排列变为升序。

$n<=1000$, $m<=2e5$

AtCoder Beginner Contest 233 F

SWAP AND SORT

将一对二元组(ai, bi)用并查集连接，同一个集合内可以任意交换。

所以每个集合按升序排序好判断整体是否升序即可。

FRIEND SUGGESTIONS

有n个人，他们之间有m对朋友关系和k对敌对关系。

我们称a和b为朋友候选人当且仅当：

- $a \neq b$。
- a和b没有朋友关系和敌对关系。
- 存在序列$c_0=a, c_1, c_2, \ldots, c_k=b$满足$c_i$和$c_{i+1}$是朋友关系。

求有多少对朋友候选人。

$n, m, k \le 1e5$

AtCoder Beginner Contest 157 D

FRIEND SUGGESTIONS

对朋友关系建立并查集。

一个人的朋友候选人=同一个并查集中的人数-好友人数-同一个并查集中的敌对人数。

看一道题

有一个长度为n的数组，初始全为0。

有q次操作，操作有两种：

1. 增加数组中某个位置的数值
2. 询问a[1]+...+a[k]的值。

n, q<=100000

想想暴力

直接做每次修改为O(1)，询问是O(n)。

由于询问的是前缀和，所以考虑能否维护前缀和。

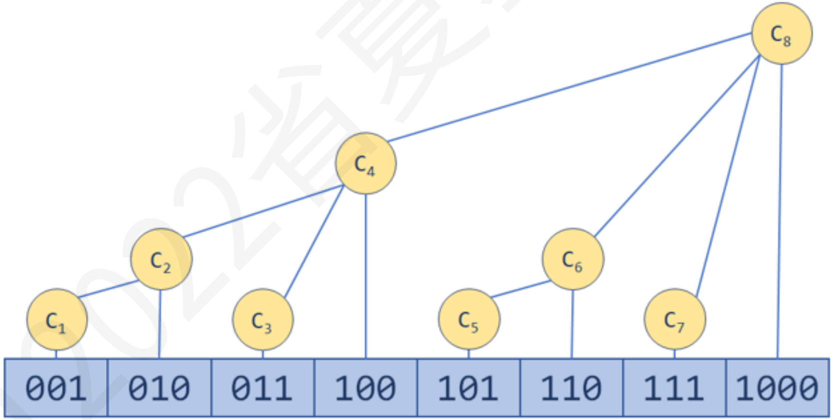
这样做每次修改为O(n)，询问是O(1)。

所以我们需要一种方式来平衡修改和询问的复杂度。

树状数组

树状数组是如下的结构。

每个位置记录的是自己以及相连的位置数值之和。



树状数组

观察上图可以发现，每个位置所在层数等于对应二进制数最低位的1的位数。

如何求一个数最低位的1?

```
int lowbit(int x) { return x & -x; }
```

树状数组中$t[x]=a[x]+a[x-1]+...+a[x-lowbit(x)+1]$。

修改

假设对a[x]增加w，如何修改t?

设x=10011010。

需要修改t[10011010], t[10011100], t[10100000], t[11000000]。

代码：

```
void add(int x, int w) { while(x <= n) { t[x] += w; x +=
lowbit(x); } }
```

询问

假设求前x个数之和。

设x=10011010。

答案为t[10011010] + t[10011000] + t[10010000] + t[10000000]。

代码：

```
int sum(int x, int w) { int r = 0; while(x) { r += t[x]; x -= lowbit(x); } return r; }
```

另一个问题

有一个长度为n的数组，初始全为0。

有q次操作，操作有两种：

1. 数组中区间[l, r]中每个位置数值+x

2. 询问a[x]的值。

n, q<=100000

另一个问题

考虑原数组的差分：$b[1]=a[1]$, $b[i]=a[i]-a[i-1]$ $(i>=2)$。

区间$[l, r]$元素都$+x$等价于$b[l]+=x$, $b[r+1]-=x$。

询问$a[x]$等价于询问$b[1]+b[2]+...+b[x]$。

原问题又转化为单点修改求前缀和的问题，使用树状数组即可。

再修改一下问题

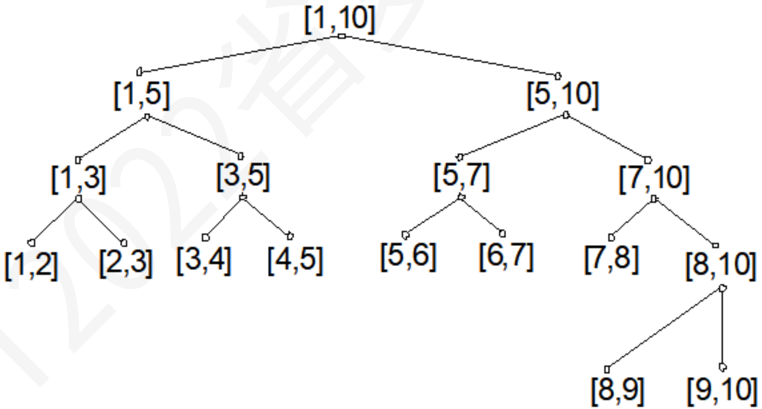
如果我们同时要区间修改且区间查询（不只是查询前缀和）呢？

树状数组难以完成。

我们需要更强的数据结构。

线段树

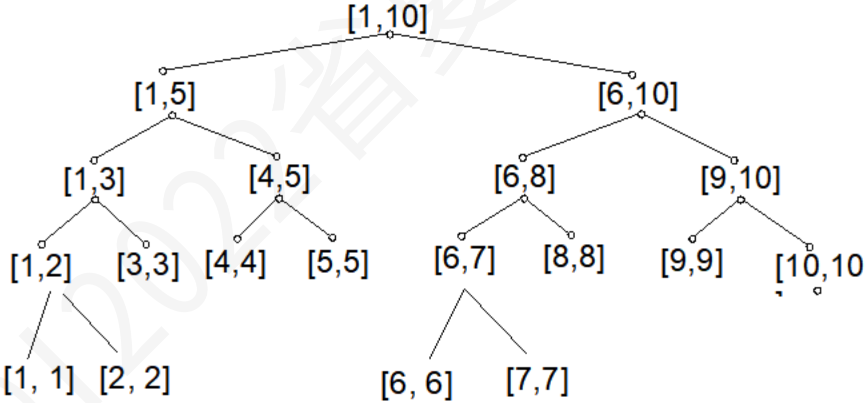
线段树是一棵二叉树，树中的每一个结点表示了一个区间[a,b]。每一个叶子节点表示了一个单位区间。对于每一个非叶结点所表示的结点[a,b]，其左儿子表示的区间为[a,(a+b)/2]，右儿子表示的区间为[(a+b)/2,b]。



线段树

大部分情况下（例如上面那道题），我们使用的是点树而不是线段树。

也即左右儿子区间分别是[a, (a+b)/2]和[(a+b)/2+1, b]



线段树的特点

总共有O(logn)层。

对任意x，共有O(logn)个节点覆盖它。

对任意区间[l, r]，可以被线段树上互不相交的O(logn)个节点恰好覆盖住（因为每一层最多只需要两个节点）。

回到最先的问题

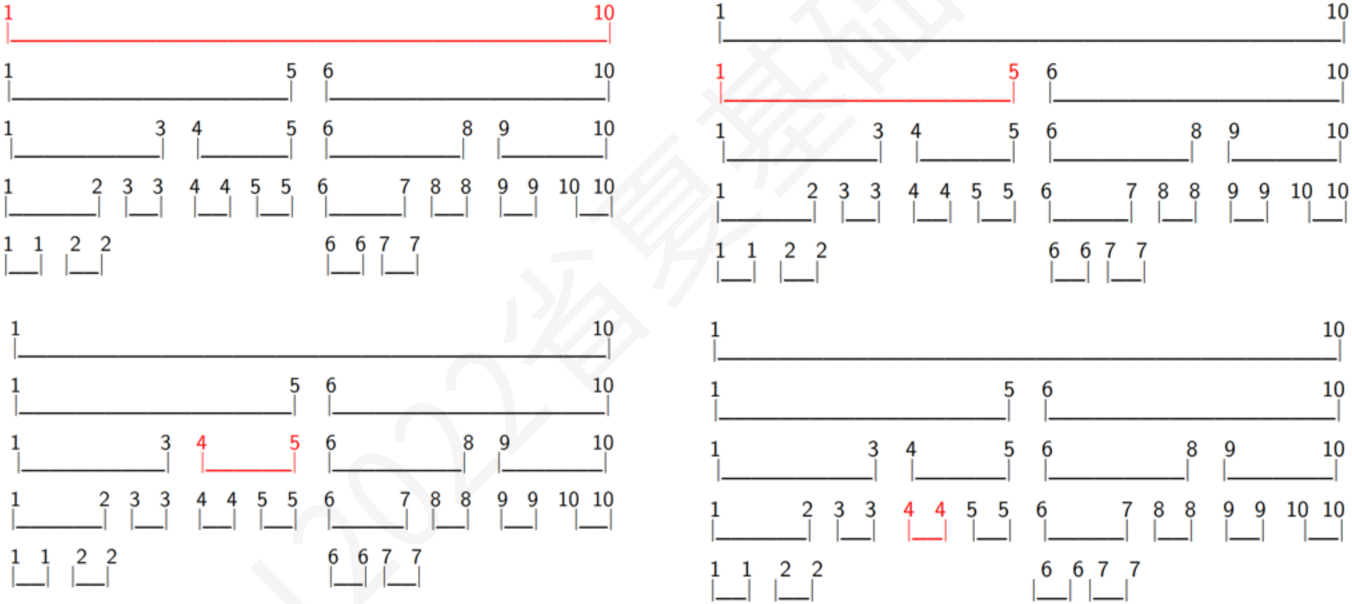
首先只考虑单点修改和区间查询的情况。

为保证查询时复杂度为O(logn)，故只能遍历到那O(logn)个覆盖区间的节点。

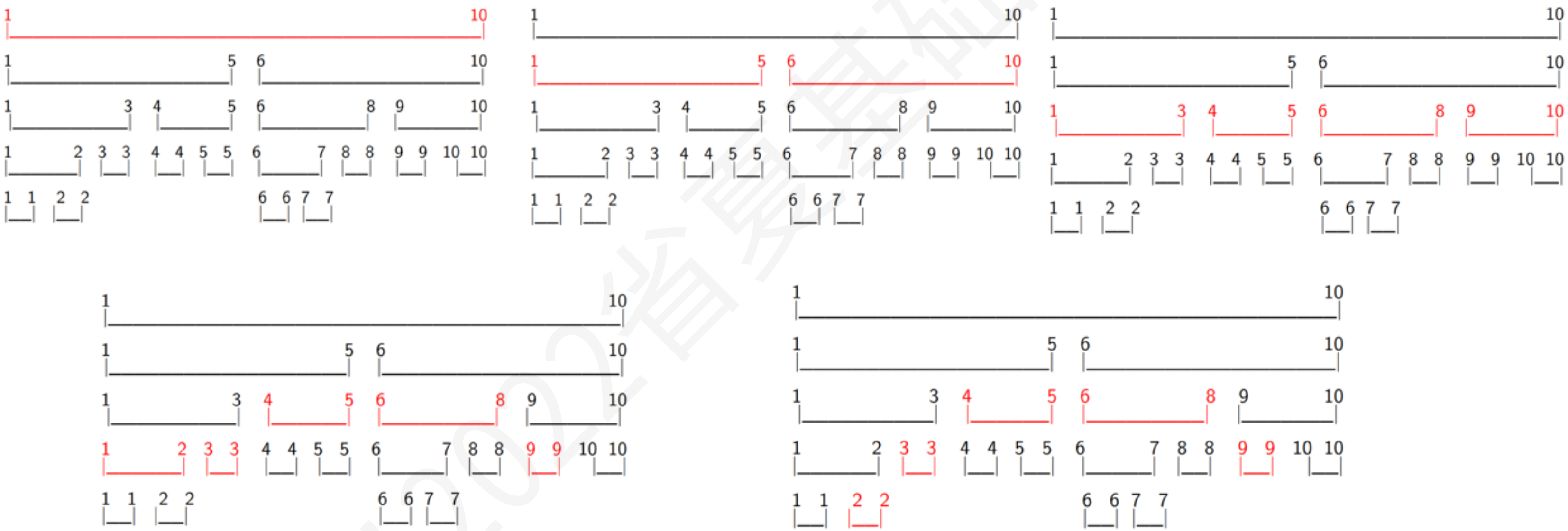
所以我们可以令每个节点上记录它所代表的区间的和，将它们加起来就是答案。

修改操作也很显然了，从根找到到对应叶子节点的路径，修改它们上面记录的值。

修改A[4]


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
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
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
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

查询A[2..9]



修改操作代码

```
void Modify(int p, int l, int r, int x, int w) { //节点p区间为[l, r]，操作为a[x]加w
    if(l == r) { t[p] += w; return; } //叶子节点
    int mid = (l + r) >> 1;
    if(x <= mid) Modify(p << 1, l, mid, x, w); //x在左子树
    else Modify(p << 1 | 1, mid + 1, r, x, w); //x在右子树
    t[p] = t[p << 1] + t[p << 1 | 1]; //更新当前节点
}
```

查询操作代码

```
int Query(int p, int l, int r, int ll, int rr) { //节点p区间为[l, r]，查询[ll, rr]之和
	if(l >= ll && r <= rr) return t[p]; //当前节点被完全覆盖
	int mid = (l + r) >> 1, res = 0;
	if(ll <= mid) res += Query(p << 1, l, mid, ll, rr); //查询区间与左区间有交
	if(rr > mid) res += Query(p << 1 | 1, mid + 1, r, ll, rr); //查询区间与右区间有交
	return res;
}
```

再看更新后的题目

区别在于单点修改改为了区间修改。

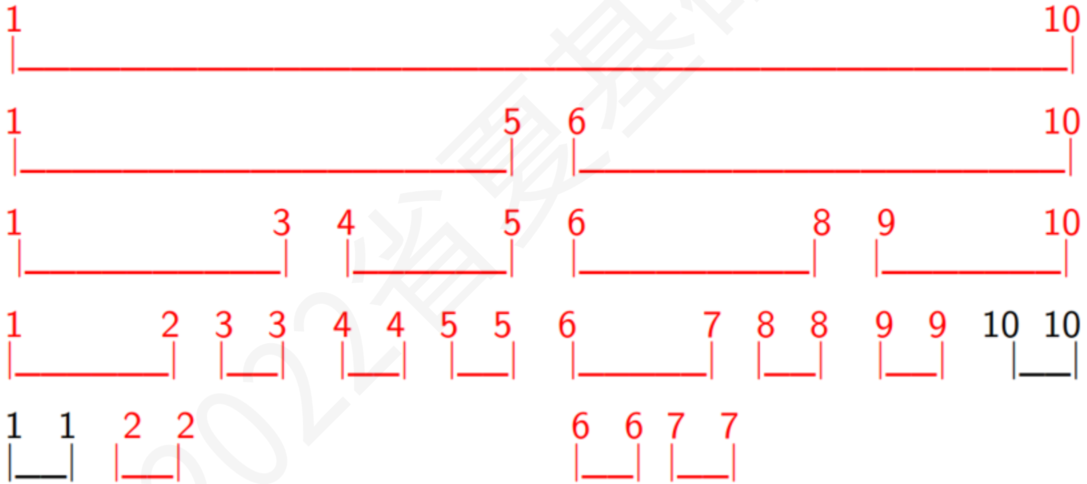
为保证复杂度，依旧只能遍历到那O(logn)个区间。

可以直接对它们加上w*区间长度，但它们子树内其他节点值没有更改，如何处理?

引入懒标记（lazy tag），记录之前更新停止在当前节点未继续向下更新的部分，之后的所有操作（更新与查询）需要遍历到下面节点前需要将懒标记下传。

本题中懒标记可以记录当前区间加的总值。

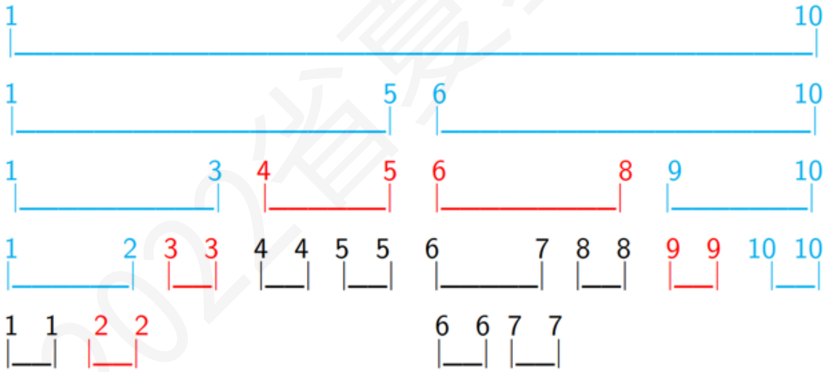
没有LAZY-TAG时修改A[2..9]...



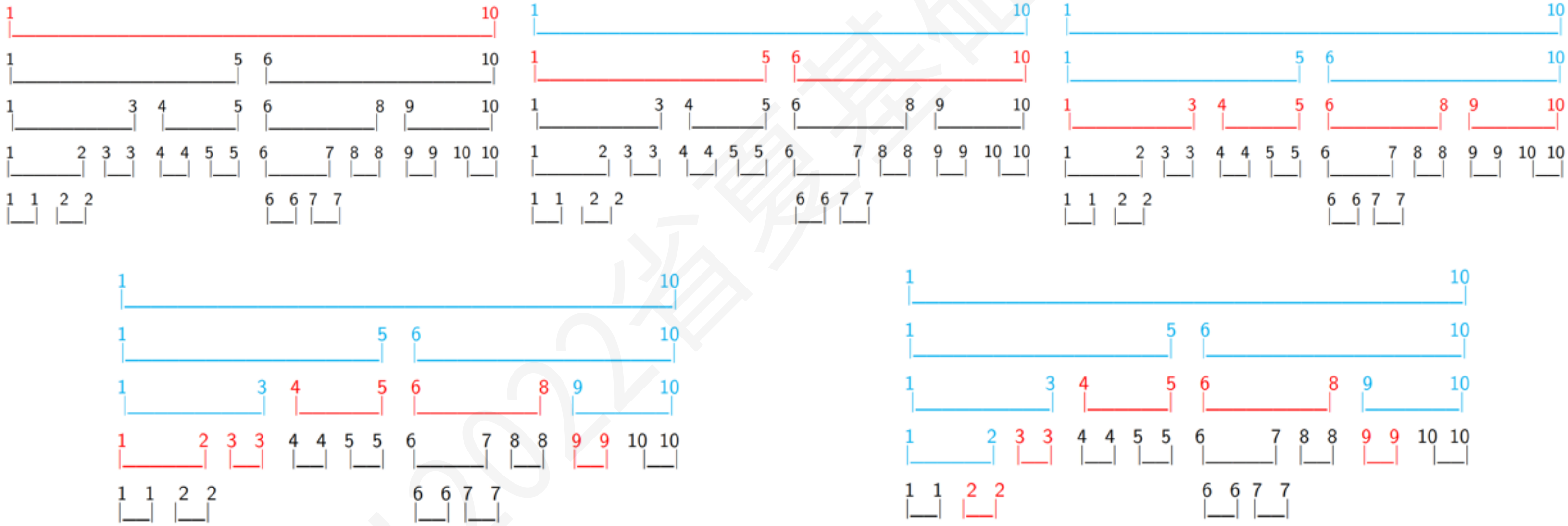
使用LAZY-TAG

蓝色是需要及时维护的节点，红色是lazy-tag下传停止的位置。

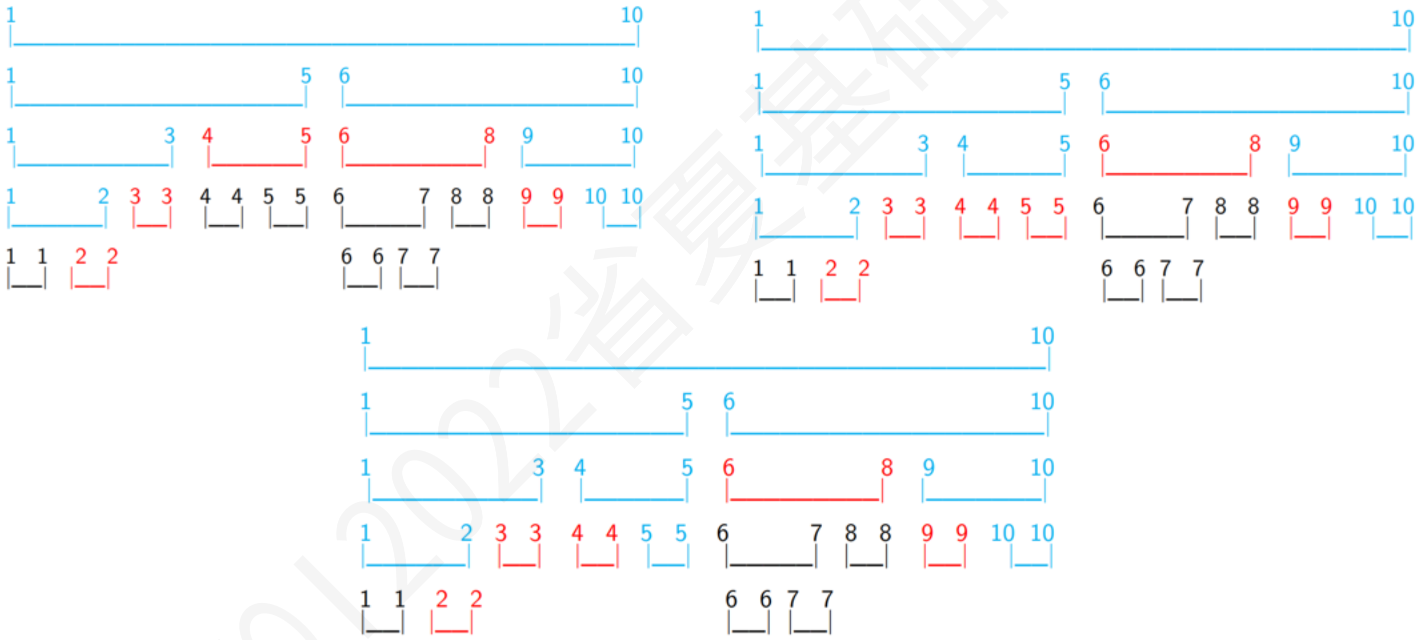
每一层最多两个红色区间，两个蓝色区间。



修改A[2..9]



查询A[5]



修改操作代码

```
void Modify(int p, int l, int r, int ll, int rr, int w) {
        if(l >= ll && r <= rr) {
                t[p] += w * (r - l + 1), tag[p] += w; //更新节点值以及懒标记
                return;
        }
        PushDown(p, l, r); //下传标记
        int mid = (l + r) >> 1;
        if(ll <= mid) Modify(p << 1, l, mid, ll, rr, w);
        if(rr > mid) Modify(p << 1 | 1, mid + 1, r, ll, rr, w);
        t[p] = t[p << 1] + t[p << 1 | 1];
}
```

查询操作代码

```
int Query(int p, int l, int r, int ll, int rr) {
        if(l >= ll && r <= rr) return t[p];
        PushDown(p, l, r); //下传标记
        int mid = (l + r) >> 1, res = 0;
        if(ll <= mid) res += Query(p << 1, l, mid, ll, rr);
        if(rr > mid) res += Query(p << 1 | 1, mid + 1, r, ll, rr);
        return res;
}
```

下传标记代码

```
void PushDown(int p, int l, int r) {
    int mid = (l + r) >> 1;
    //往下更新一层
    t[p << 1] += tag[p] * (mid - l + 1);
    t[p << 1 | 1] += tag[p] * (r - mid);
    tag[p << 1] += tag[p], tag[p << 1 | 1] += tag[p]; //也要更新儿子的tag
    tag[p] = 0; //清空当前tag
}
```

其他操作

修改不限于区间+C，还需要支持区间*C，区间=C……

查询不限于区间求和，还需要支持查询区间最小值，区间最大值……

例题

对于一个序列维护以下操作：

1. 修改某个a[i]的值。
2. 询问区间[l, r]中的数两两差之和以及两两差的平方和是多少。

答案对1e9+7取模。

n, q <= 100000

例题

两两差之和是0。

将差的平方和括号拆开，可以发现需要维护区间和以及区间平方和。

例如：

(a[1]-a[2])^2+(a[1]-a[3])^2+(a[2]-a[3])^2

=2a[1]^2+2a[2]^2+2a[3]^2-2a[1]a[2]-2a[1]a[3]-2a[2]a[3]

=3(a[1]^2+a[2]^2+a[3]^2)-(a[1]+a[2]+a[3])^2

矩形面积并

给出二维平面上若干个矩形，求它们覆盖的总面积。

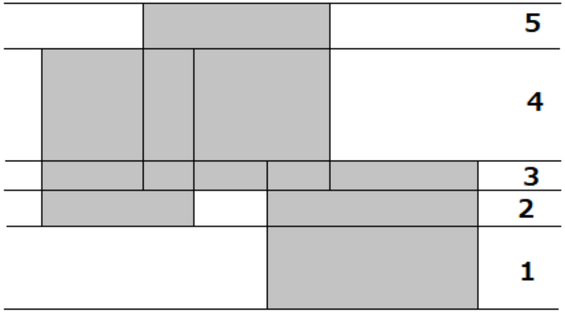
0<=n<=100000

矩形面积并

先对坐标离散。

扫描线，从下之上，遇到矩形下边为区间+1，上边为区间-1。

答案增加非零的区间总长度*这一段高度。



最大子段和

给定数组a，要求支持：

1. 单点修改。
2. 询问一个区间的最大子段和。

n, q<=100000

最大子段和

首先当然每个节点要记录所代表区间的最大子段和。

问题在于如何合并两个节点的信息求出大区间的区间的最大子段和。

存在方式有三种：左区间最大子段和、右区间最大子段和和跨越中间的。

跨越中间最大字段和=左区间右起最大子段和+右区间左起最大子段和。

又有左起最大子段和=max{左区间左起最大子段和，左区间和+右区间左起最大子段和}。

所以每个节点需要维护四个信息：区间和，区间最大子段和，左起/右起最大子段和。

TRAFFIC

有一个2*n的点阵，初始时互相之间没有边。需要维护三种操作：

1. 在一对相邻点之间连边。
2. 删除一条边。
3. 询问两点是否连通。

n<=100000

TRAFFIC

线段树上区间[l, r]的节点记录只经过[l, r]中的节点(l, 0/1)和(r, 0/1)四个节点互相之间能否联通（6个值）。

合并子节点方式显然（细节较多需要注意）。

查询[l, r]是否连通，需要注意可能掉头。

所以询问时要查询[1, l]、[l, r]、[r, n]一共三次。