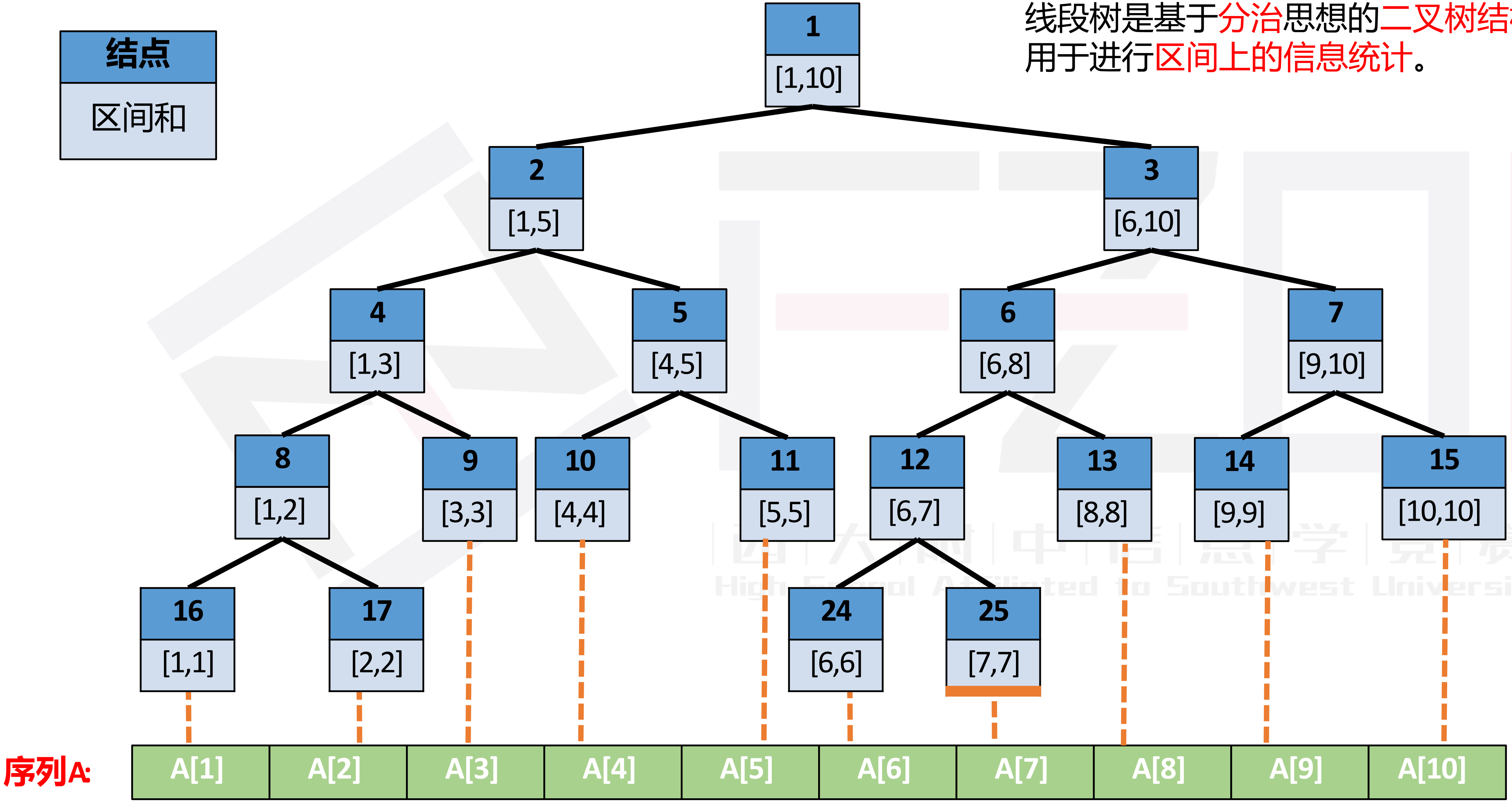
信息学

线段树

一点复习
线段树

线段树是基于分治思想的二叉树结构，用于进行区间上的信息统计。



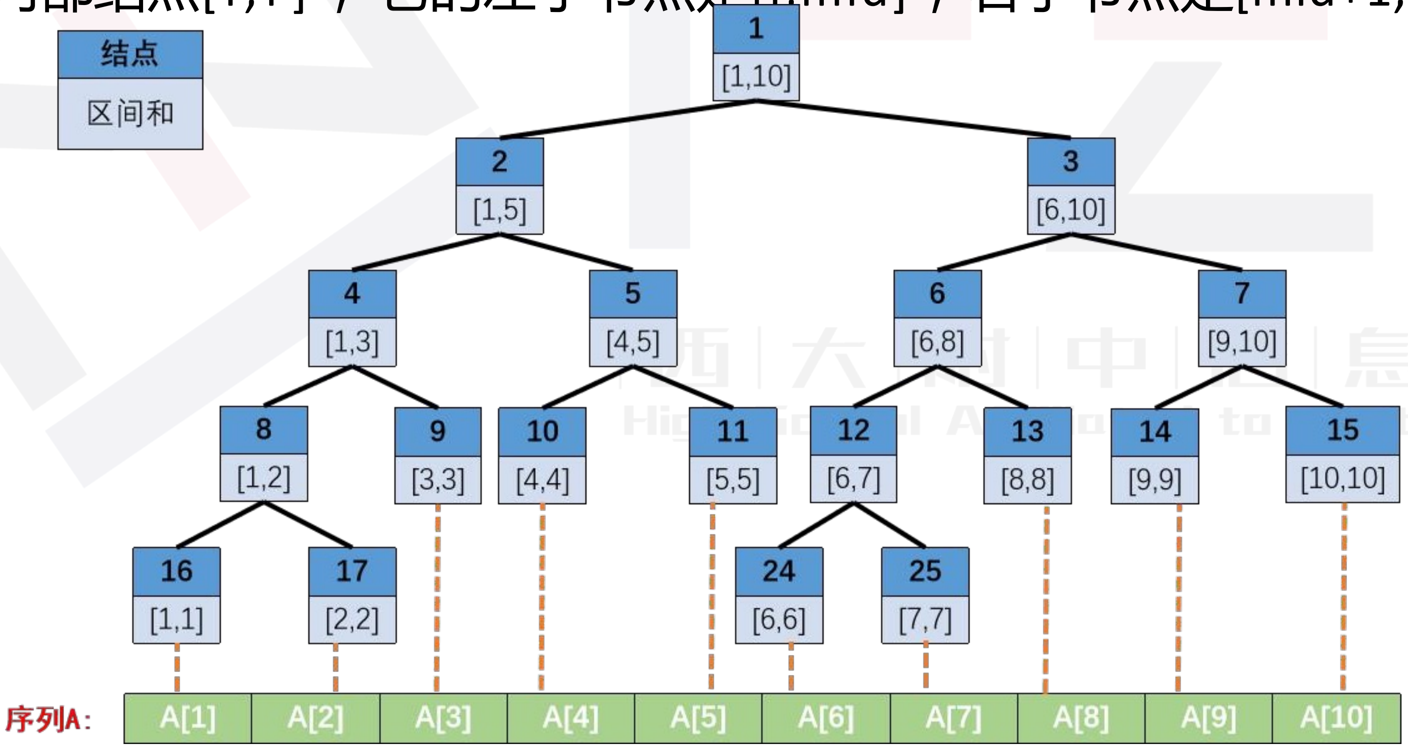
① 线段树的每个节点都代表一个区间。

② 线段树具有唯一的根节点，代表整个区间的统计范围：[1,N]。

③ **线段树的每个叶结点都代表一个长度为1的区间[x,x]，对应序列的A[x]**。

④ 根节点编号为1，其他编号节点为x的左子节点为x*2，右子节点为x*2+1。

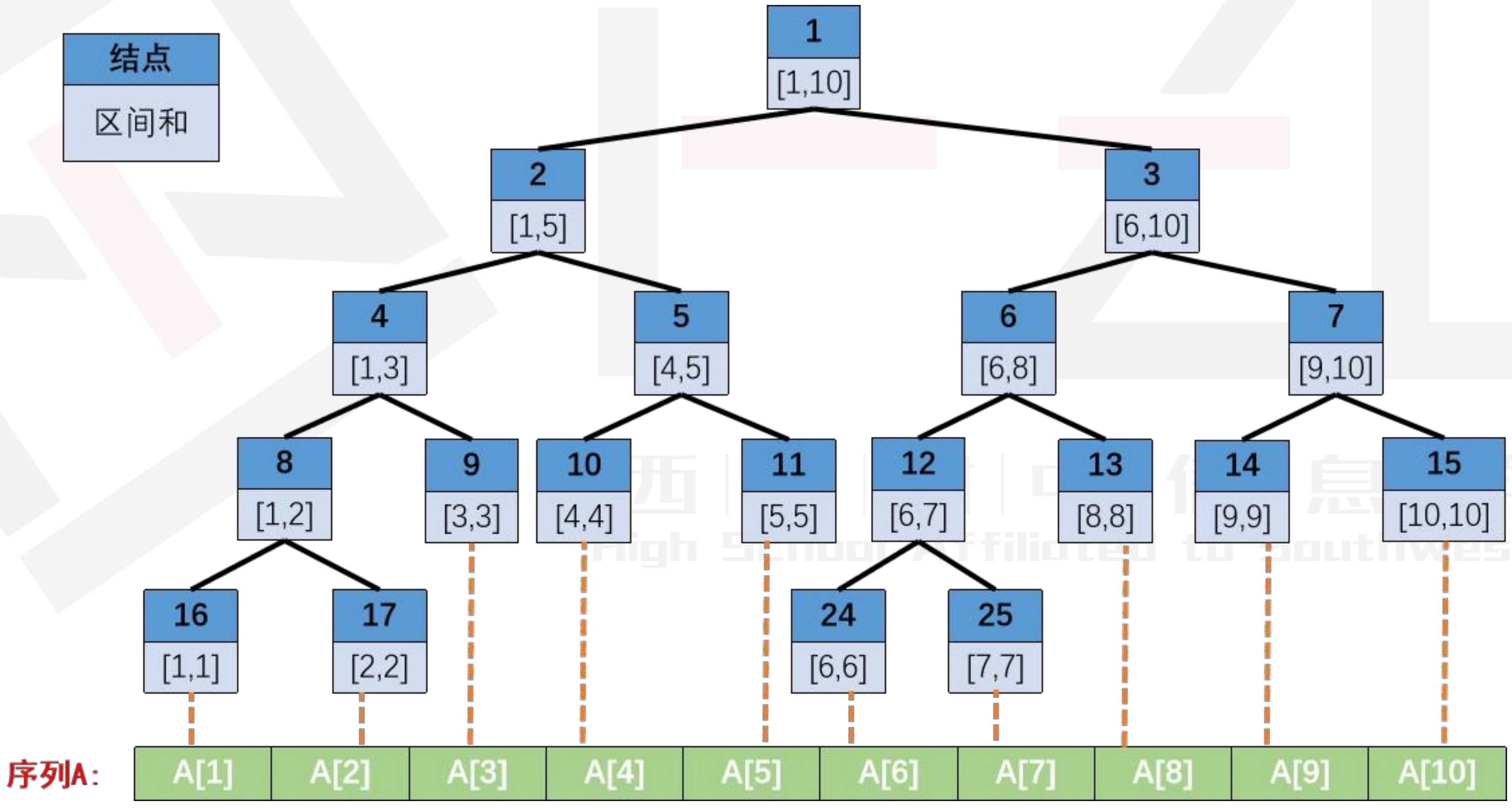
⑤ 对于每个内部结点[l,r]，它的左子节点是[l,mid]，右子节点是[mid+1,r]，其中mid=(l+r)/2。



N个叶结点的满二叉树共有结点： N+N/2+N/4+...+1=2N-1

有时候最后一层产生了空余，2N+N+N/2+N/4+...+1=4N-1

所以**保存线段树的数组长度要不小于4N，才能保证不会越界**。

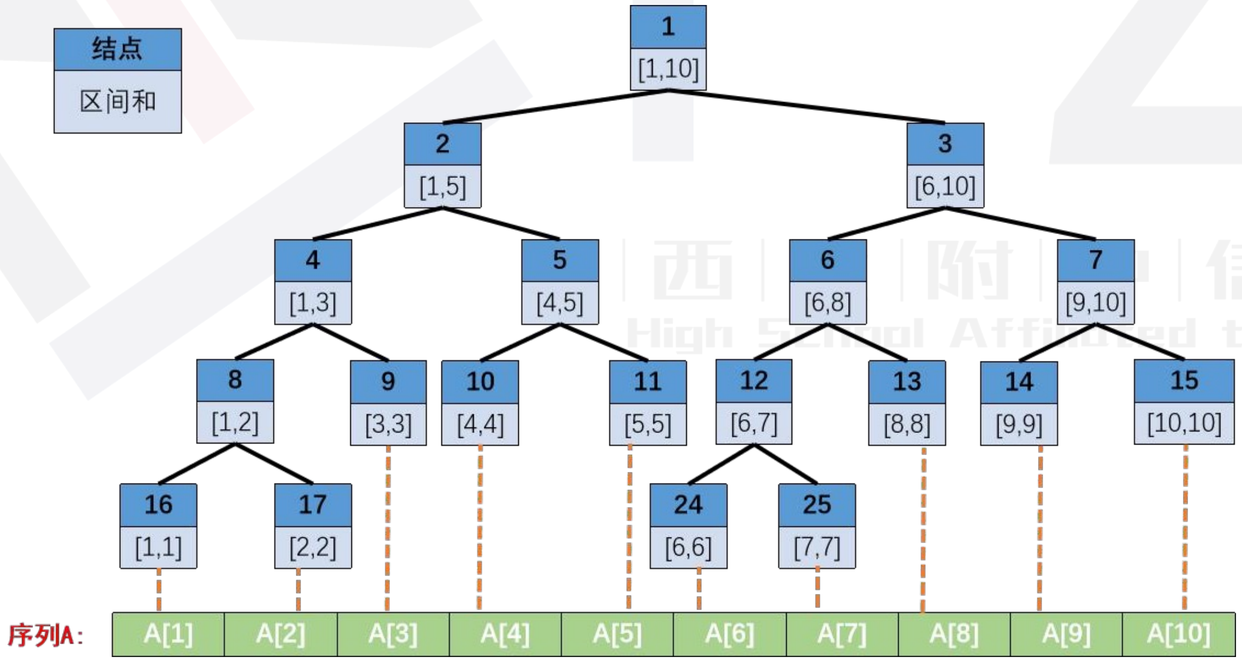


1.线段树的建树

给定一个长度为N的序列A，我们可以在区间[1,N]上建立一棵线段树，每个叶节点[i,i]保 存A[i]的值。

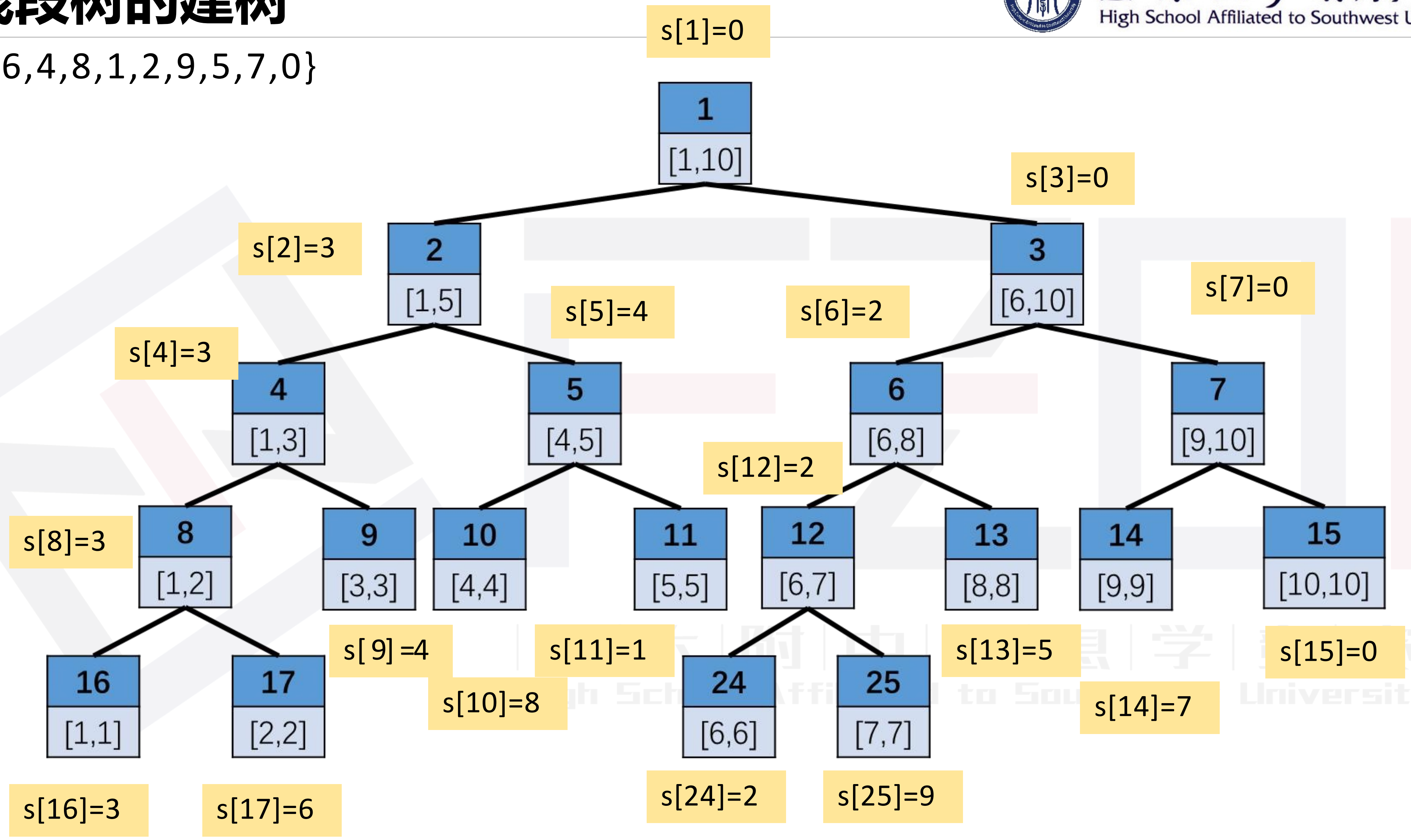
线段树的二叉结构可以很方便的从下往上传递信息，可以用**结点维护区间最大值，区间最小值，区间和等信息**。后面都以节点维护区间最小值为例。

用数组s[i]表示节点i 对应区间的最小值。



1.线段树的建树

序列A={3,6,4,8,1,2,9,5,7,0}



序列A:

3	6	4	8	1	2	9	5	7	0

1. 线段树的建树

```
void built(int k,int l,int r)//第k个节点代表的区间为[l,r]
{
        if(l==r)                //叶节点
        {
                s[k]=a[l];
                return;
        }
        int mid=(l+r)/2;
        built(k*2,l,mid);
        built(k*2+1,mid+1,r);
        s[k]=min(s[k*2],s[k*2+1]);//维护最小值
}
//主函数main()里
for(int i=1;i<=n;i++)
        cin>>a[i];
built(1,1,n);        //调用
```

```
s[k]=max(s[k*2],s[k*2+1]); //维护最大值
s[k]=s[k*2]+s[k*2+1]; //维护区间和
```

时间复杂度: O(N)

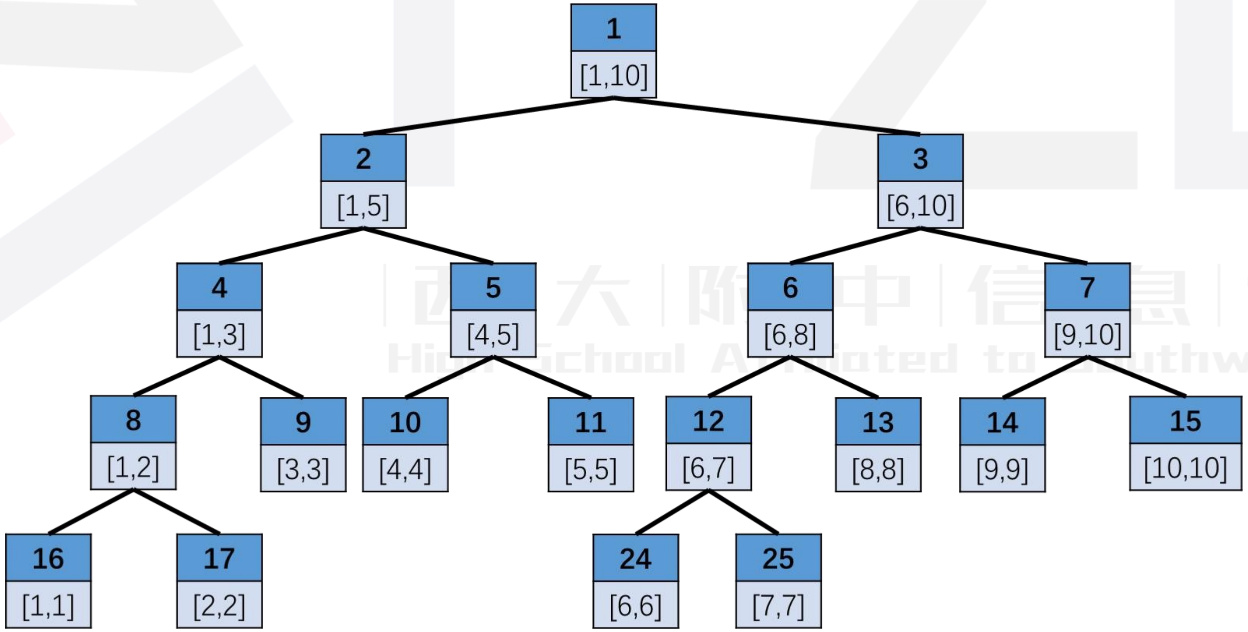
线段树的单点修改

将A[x]的值修改为v。

在线段树中，**根节点(编号1的节点)是执行各种命令的入口。**

我们从根节点出发，**递归**找到区间[x, x]的叶节点进行修改，同时**回溯**时从下往上更新节点维护的最小值(或最大值，区间和等)。

时间复杂度O(logN)



2. 线段树的单点修改


```
void change(int k,int l,int r,int x,int v)    //将a[x]修改为v
{
        if(l==r&&x==l)                  //到达对应的叶结点
        {
                s[k]=v;
                return;
        }
        int mid=(l+r)/2;
        if(x<=mid)  change(k*2,l,mid,x,v);              //修改左区间
        else  change(k*2+1,mid+1,r,x,v);     //修改右区间
        s[k]=min(s[k*2],s[k*2+1]);          //回溯，维护最小值
}
//主函数main()里
change(1,1,n,x,v);          //调用
```

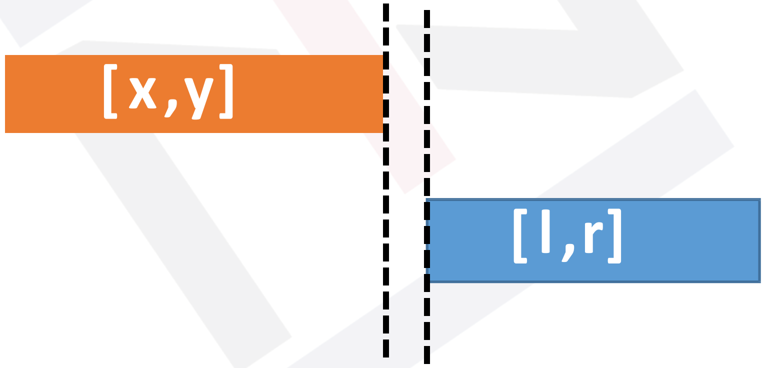
3.线段树的区间询问

查询区间[x,y]的最小值(或最大值，和等)。

int ask(int k,int l,int r,int x,int y)　　//询问区间[x,y]

从根节点出发，**查询区间[x,y]**和**节点k表示区间[l,r]**存在三种关系：

（1）查询区间[x,y]与节点k表示区间[l,r]无交集，



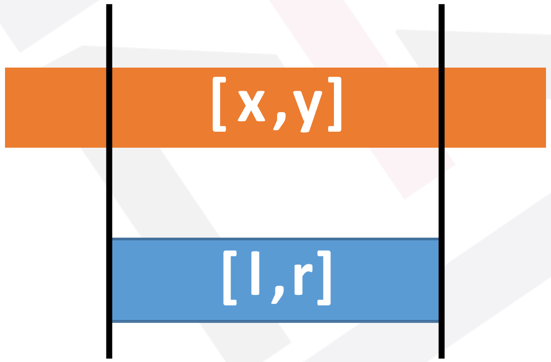
3.线段树的区间询问

查询区间[x,y]的最小值(或最大值，和等)。

int ask(int k,int l,int r,int x,int y)　　//询问区间[x,y]

从根节点出发，**查询区间[x,y]**和**节点k表示区间[l,r]**存在三种关系：

（2）查询区间[x,y]包含节点k表示区间[l,r]，



直接返回当前节点k维护的最小值s[k]，作为**候选答案**。

3.线段树的区间询问

查询区间[x,y]的最小值(或最大值，和等)。

int ask(int k,int l,int r,int x,int y)　　//询问区间[x,y]

从根节点出发，**查询区间[x,y]**和**节点k表示区间[l,r]**存在三种关系：

（3）查询区间[x,y]与节点k表示区间[l,r]，存在交集

继续在[l,mid]和[mid+1,r]寻找，直到出现包含或者不相交的情况。mid=(l+r)/2

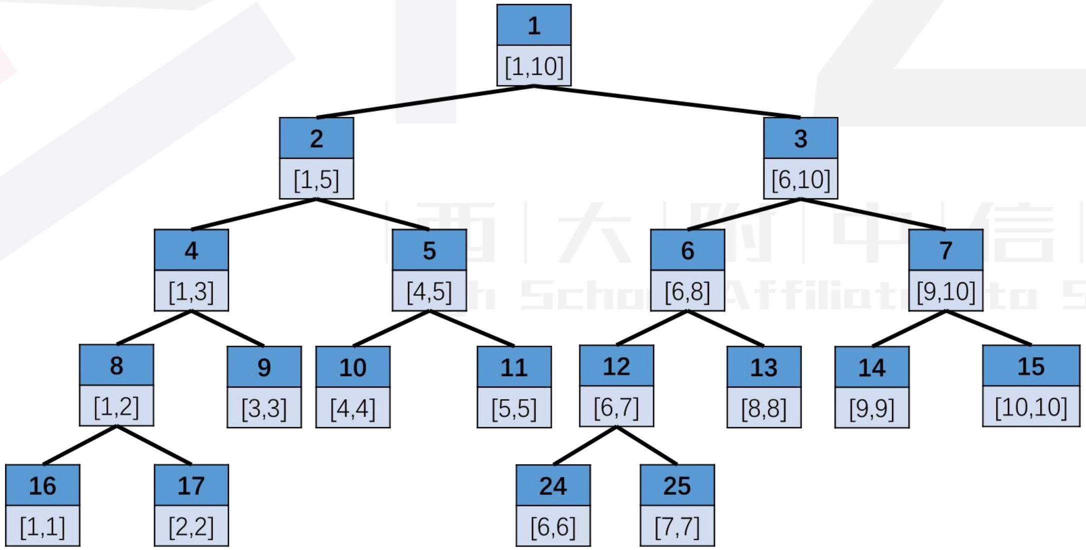
3. 线段树的区间询问

- 查询区间[x,y]的最小值(或最大值，和等)。
- 从根节点出发，**查询区间[x,y]**和**节点表示区间[l,r]**存在三种关系：

- ①查询区间[x,y]与表示区间[l,r]无交集。
- ②查询区间[x,y]包含表示区间[l,r]，直接返回当前节点维护的最小值，作为候选答案。
- ③除了上面两种情况，继续在[l,mid]和[mid+1,r]寻找，直到出现包含或者不相交的情况, 返回区间的最小值

。

- 时间复杂度:O(logN)



3. 线段树的区间询问


```
int ask(int k,int l,int r,int x,int y)      //询问区间[x,y]最小值
{
            if(x<=l&&r<=y)  return s[k];          //查询区间包含当前区间，返回维护好的最小值
            int mid=(l+r)/2;
            int val=1<<30;                        //初始化为最大值
            if(x<=mid) val=min(val,ask(k*2,l,mid,x,y));
            if(y>mid)   val=min(val,ask(k*2+1,mid+1,r,x,y));
            return val;
}
//主函数main()里
ask(1,1,n,x,y);      //调用
```

线段树有两个数组：

数组a：存储序列。建好线段树就不需要了，叶节点的值就是数组a的值。数组s，存储节点表示的信息。（最大值，最小值，和等）

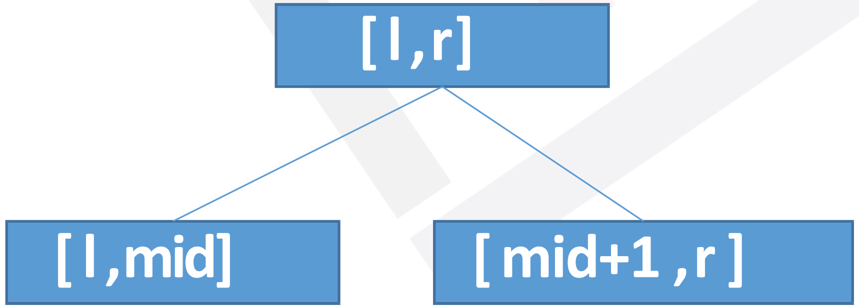
4.线段树的区间修改

将区间[x,y]的所有数加v

将区间[x,y]看作一个一个点进行单点修改，时间复杂度为O(NlogN)。

这里我们改为维护区间和，s[k]表示节点k表示的区间[l,r]的和。

<u>考虑在每个节点维护一个数组**lazy**，**表示这个节点所对应区间内的所有数据都加上lazy[]**</u>。



区间修改原理：

现在对区间[l , r] 所以数增加值v。

如果询问区间和时，不会询问到区间[l , r] ，那么区间[l , r] 的修改操作就是无用的。

因此添加一个标记，表示要修改区间[l , r] ，但是不进行修改操作，当要访问这个区间时，再进行修改操作。这就是**lazy标记**。

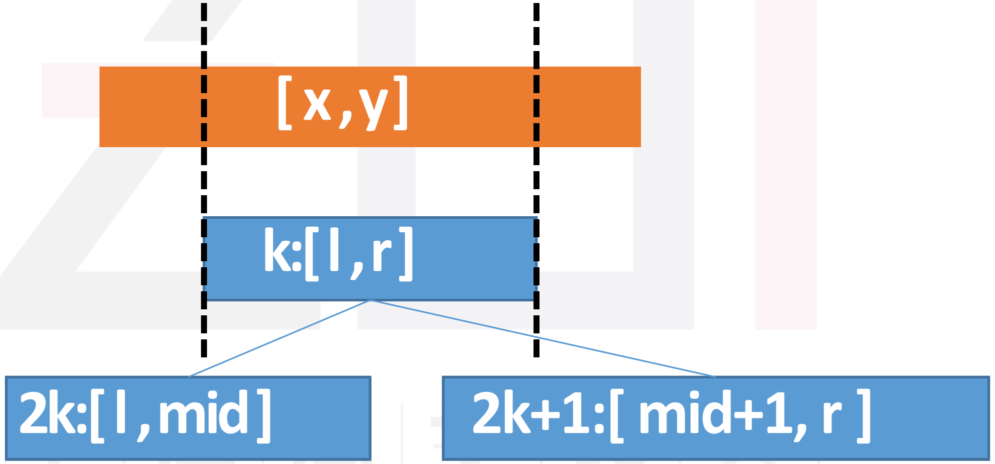
4.线段树的区间修改

如果修改区间[x,y]包含第k个节点的区间[l,r]，说明当前节点(设为k)表示的区间[l,r]以及子节点都会被修改，此时，我们将s[k]进行修改：s[k]+=(r-l+1)*v。同时给k节点打上标记lazy[k]+=v，

表示**当前节点已经被更新，但其子节点未更新。**

当需要这些子节点的信息时，才会去更新。

这个标记我们叫延迟标记，或**Lazy-Tag(懒标记)**。

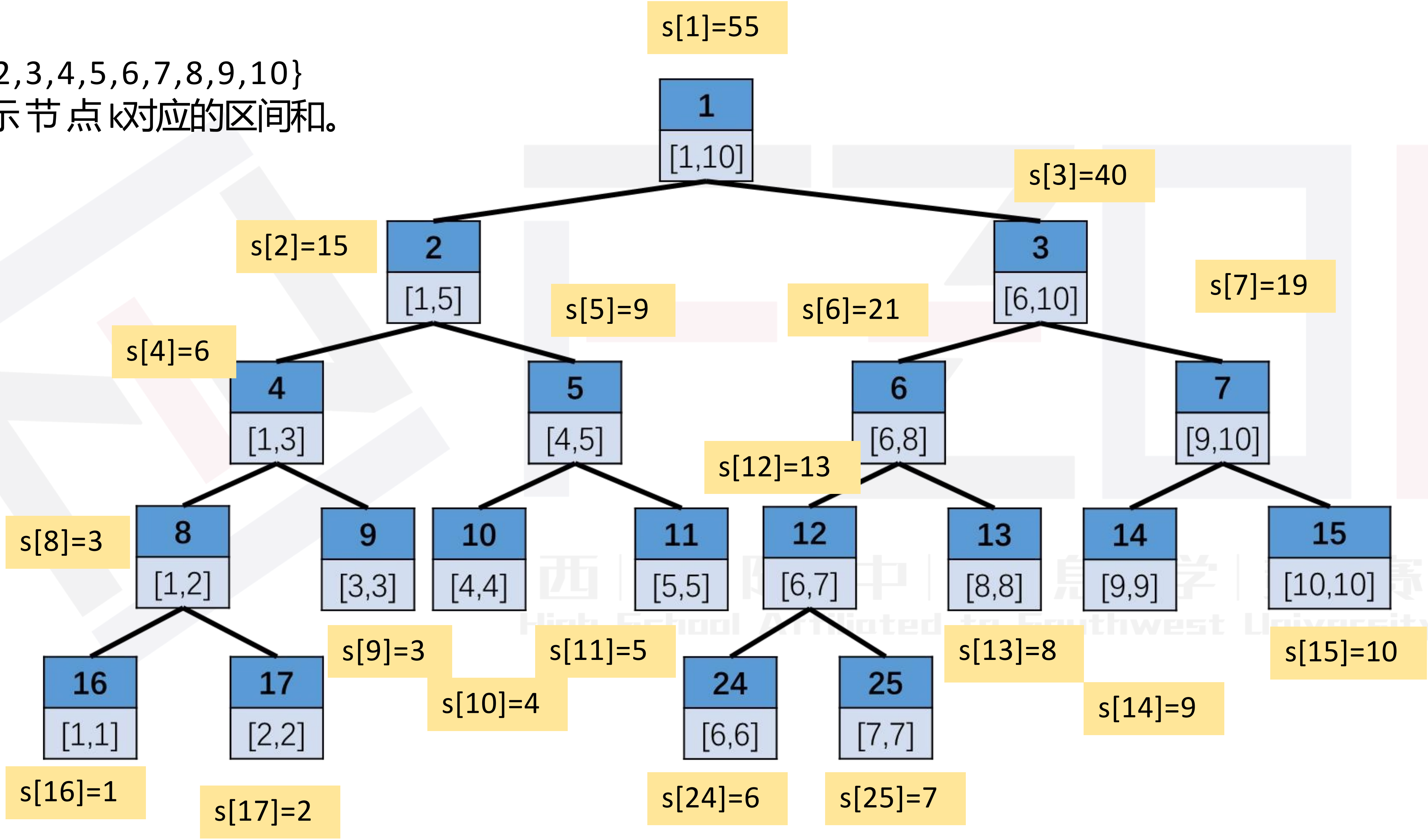


标记下传：

当访问某个节点，有标记时，将当前节点的lazy下传，更新两个子节点的s值和lazy值，并将当前节点的lazy值清零。

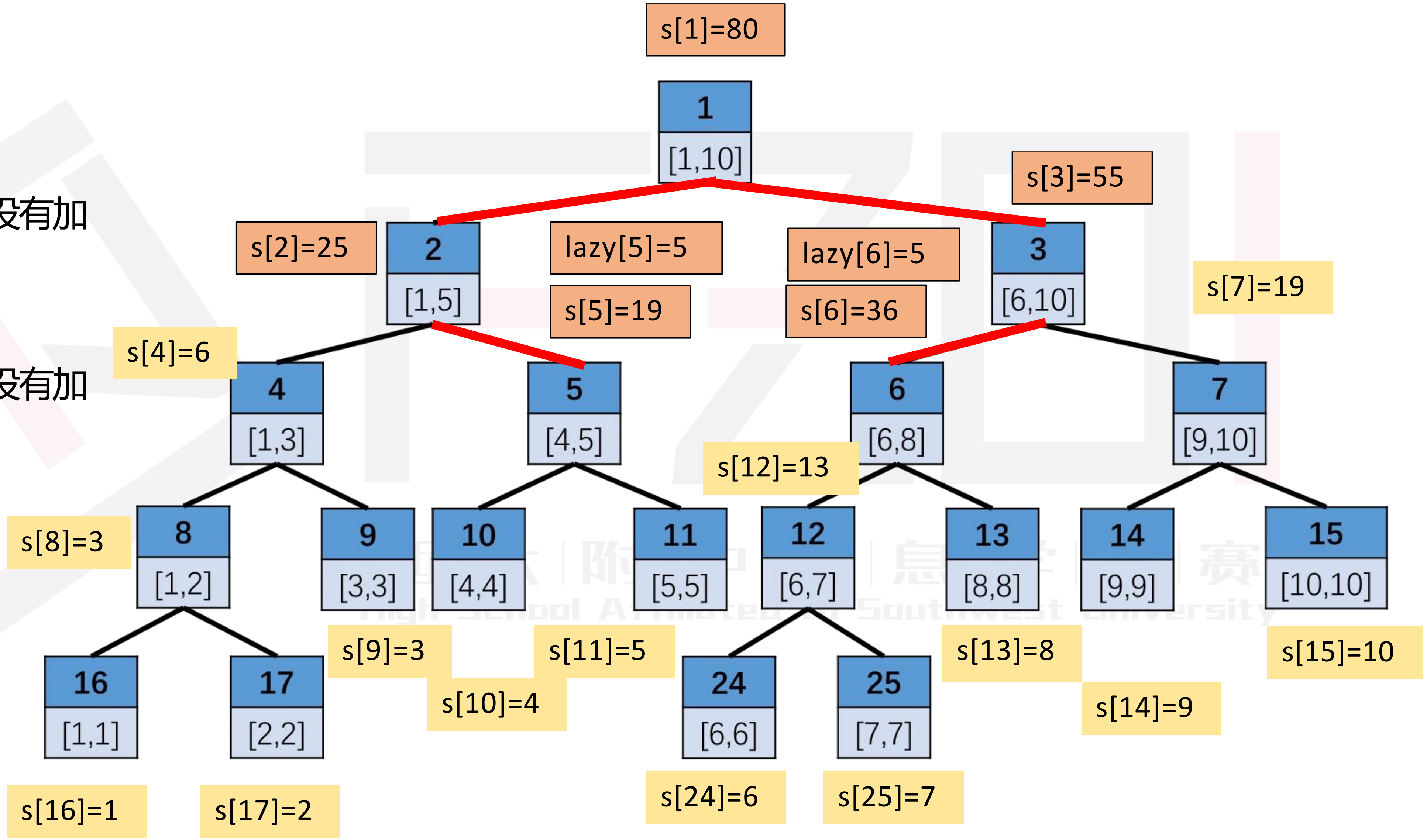
4. 线段树的区间修改

a[]={1,2,3,4,5,6,7,8,9,10}
s[k]表示节点k对应的区间和。



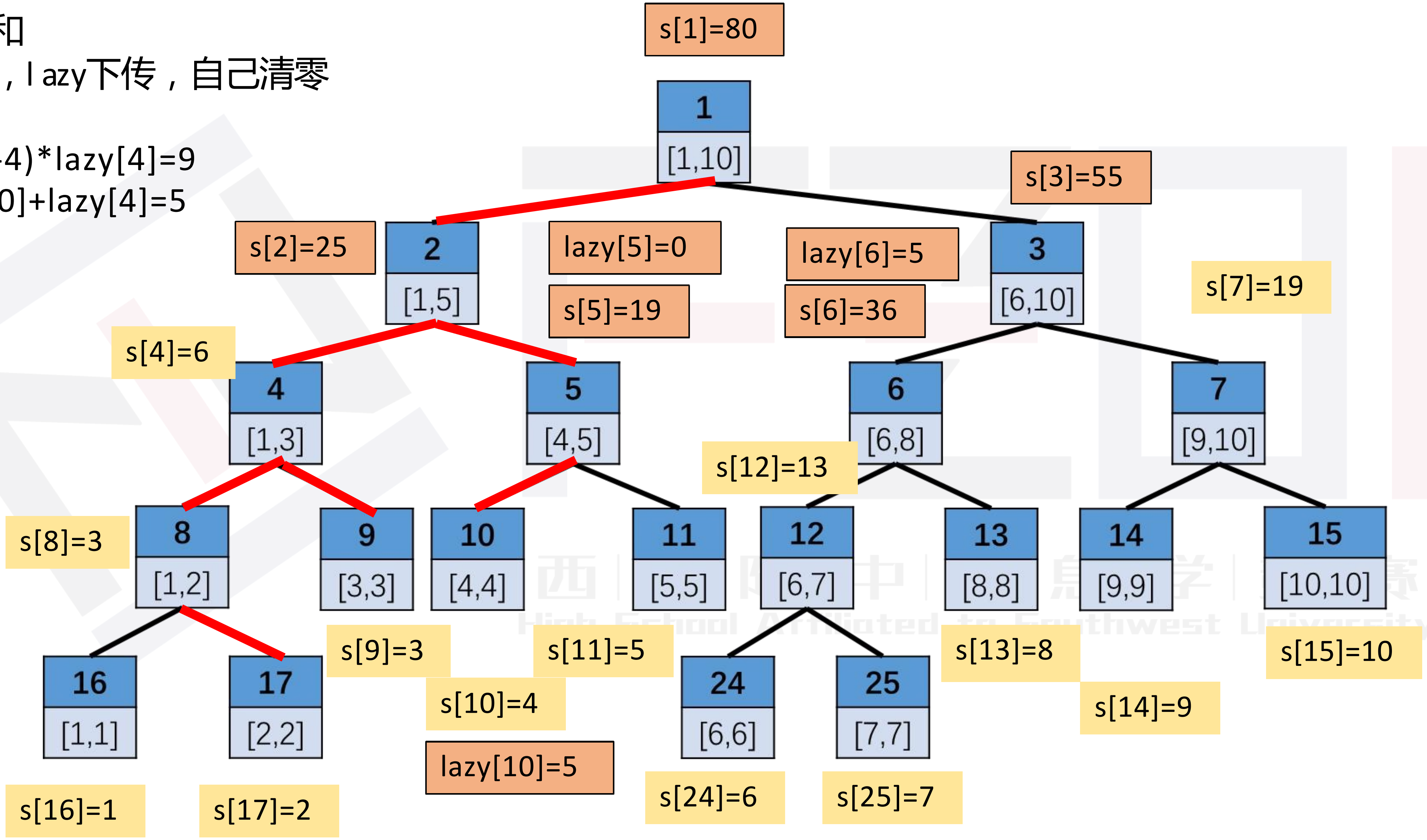
4. 线段树的区间修改

将区间[4, 8] 每个元素增加5 s[5]=s[5]+(5-4+1)*5=19
lazy[5]=lazy[5]+5=5 ，
表示节点10和11的区间还没有加
s[2]=s[4]+s[5]=25
s[6]=s[6]+(8-6+1)*5=36
lazy[6]=lazy[6]+5=5
表示节点12和13的区间还没有加
s[3]=s[6]+s[7]=55
s[1]=s[2]+s[3]=80



4. 线段树的区间修改

求区间[2,4]的和
访问到节点5时，l azy下传，自己清零
lazy[5]=0
s[10]=s[10]+(4-4)*lazy[4]=9
lazy[10]=lazy[10]+lazy[4]=5



4. 线段树的区间修改

```
void pushdown(int k,int l,int r) //将节点k的标记下传给子节点
{
        if(lazy[k]==0) return;
        int mid=(l+r)/2;
        //注意，当前节点k，不用增加，已经增加过了
        //下传左子树
        s[2*k]+=(mid-l+1)*lazy[k];
        lazy[2*k]+=lazy[k];
        //下传右子树
        s[2*k+1]+=(r-mid)*lazy[k];
        lazy[2*k+1]+=lazy[k];
        lazy[k]=0;                    //下传自身清零
}
```

4. 线段树的区间修改

```
void modify(int k,int l,int r,int x,int y,int v) //给区间[x,y]所有数加上v
{
        if(x<=l&&r<=y)                    //完全包含直接处理
        {
                s[k]+=(r-l+1)*v;
                lazy[k]+=v;
                return;
        }
        pushdown(k,l,r);          //不存在完全包含就将k节点之前可能存在的标记下传，继续找
        int mid=(l+r)/2;
        if(x<=mid) modify(k*2,l,mid,x,y,v);          //找左子树
        if(y>mid) modify(k*2+1,mid+1,r,x,y,v);       //找右子树
        s[k]=s[k*2]+s[k*2+1];                    //下传后更新区间和s
}
```

4. 线段树的区间修改


```
int ask(int k,int l,int r,int x,int y)      //询问区间[x,y]的和
{
        if(x<=l&&r<=y) return s[k];                //包含直接返回
        int mid=(l+r)/2;
        pushdown(k,l,r);              //下传标记，将未更新的节点先更新
        int val=0;
        if(x<=mid) val+=ask(2*k,l,mid,x,y);
        if(y>mid)  val+=ask(2*k+1,mid+1,r,x,y);
        return val;
}
```

线段树是基于**分治**思想的**二叉树结构**，用于进行**区间上的信息统计**。

线段树和树状数组的基本功能都是在某一满足结合律的操作(比如加法，乘法，最大值，最小值)下，O(logn)的时间复杂度内修改单个元素并且维护区间信息。

不同的是，树状数组只能维护前缀“操作和”(前缀和，前缀积，前缀最大最小)，而线段树可以维护区间操作和。线段树的适用范围更广。

树状数组能做的，线段树都能做。线段树能做的，树状数组不一定会做。

区间合并

「USACO2008FEB」Hotel

有一个旅馆，有N(<=50000)间连续的房间，从左往右依次编号1~N，初始时都没有住人，有两种操作：

1　　需要连续的D间房间，如果有多个满足，起点房间编号最小的入住，然后输出起点房间编号；

2　　将编号X起连续的D间房间退房；

看作N个数的0、1序列，0表示无人入住，1表示入住。

入住：寻找连续的D个0的区间，然后都修改为1；

退房：将区间[X,X+D-1]全部变为0。

需要解决的问题：

修改区间[x, y] 为1或者0。　　lazy标记

求连续的D个为0的区间 ？

线段树——区间合并

对于结点k，表示区间[l , r] 维护三个数组：

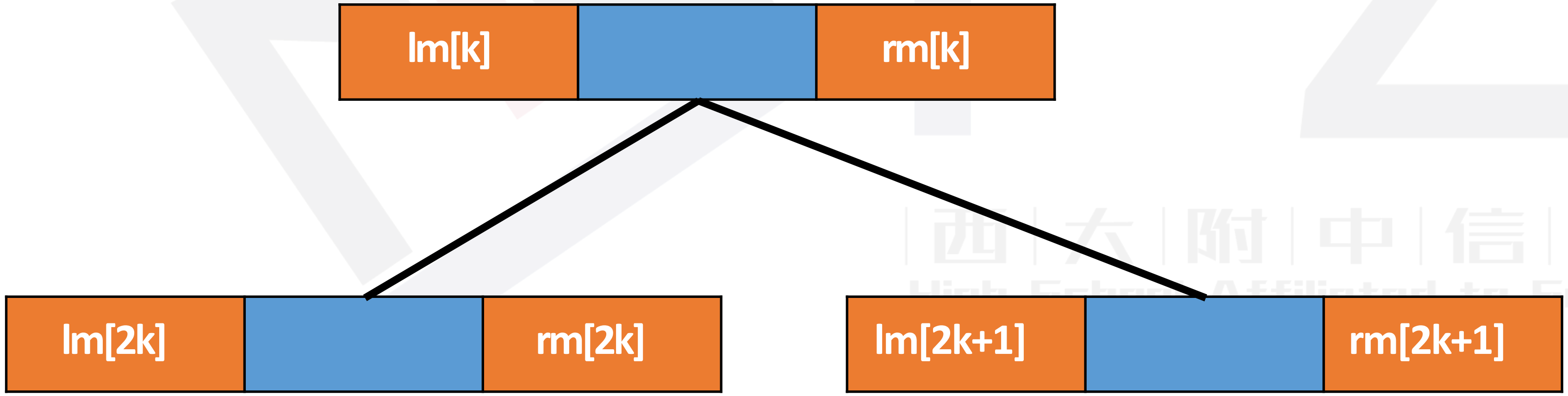
lm[]——为左端点l 开始最大连续0的长度

rm[]——为右端点r开始最大连续0的长度

m[]——为区间[l , r] 最大连续0的长度

初始化：

lm[k]=rm[k]=m[k]=r-l+1



线段树——区间合并

对于结点k，表示区间[l , r] 维护三个数组：

lm[]——为左端点l 开始最大连续0的长度

rm[]——为右端点r开始最大连续0的长度

m[]——为区间[l,r]最大连续0的长度

lm[k]=?

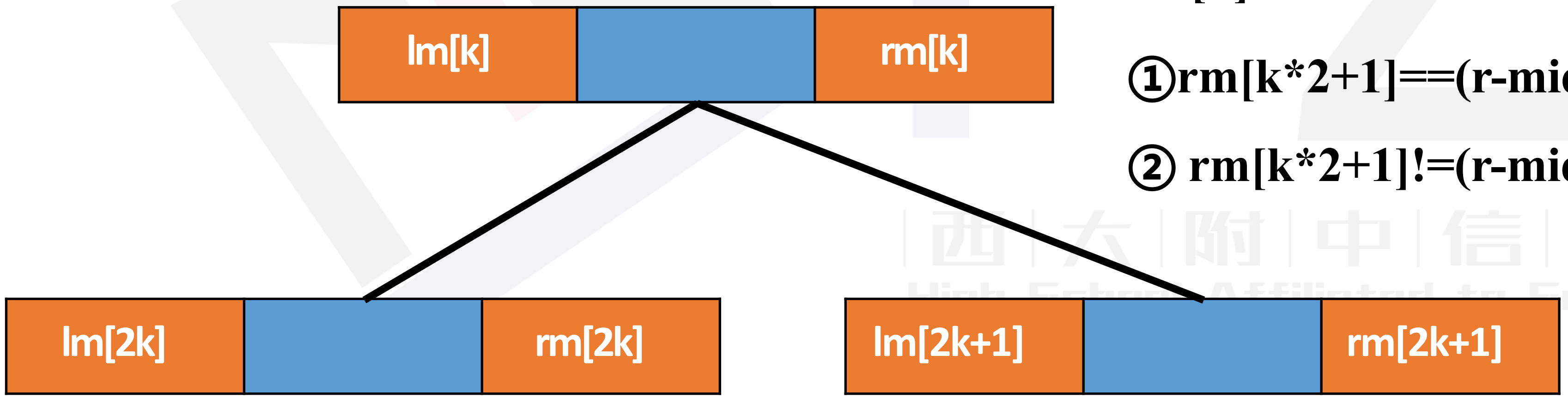
①lm[k*2]==(mid-l+1)　　**lm[k]=lm[k*2]+lm[k*2+1];**

②lm[k*2]!=(mid-l+1)　　**lm[k]=lm[k*2];**

rm[k]=?

①rm[k*2+1]==(r-mid)　　**rm[k]=rm[k*2+1]+rm[k*2];**

② rm[k*2+1]!=(r-mid)　　**rm[k]=rm[k*2+1];**



线段树——区间合并

对于结点k，表示区间[l , r] 维护三个数组：

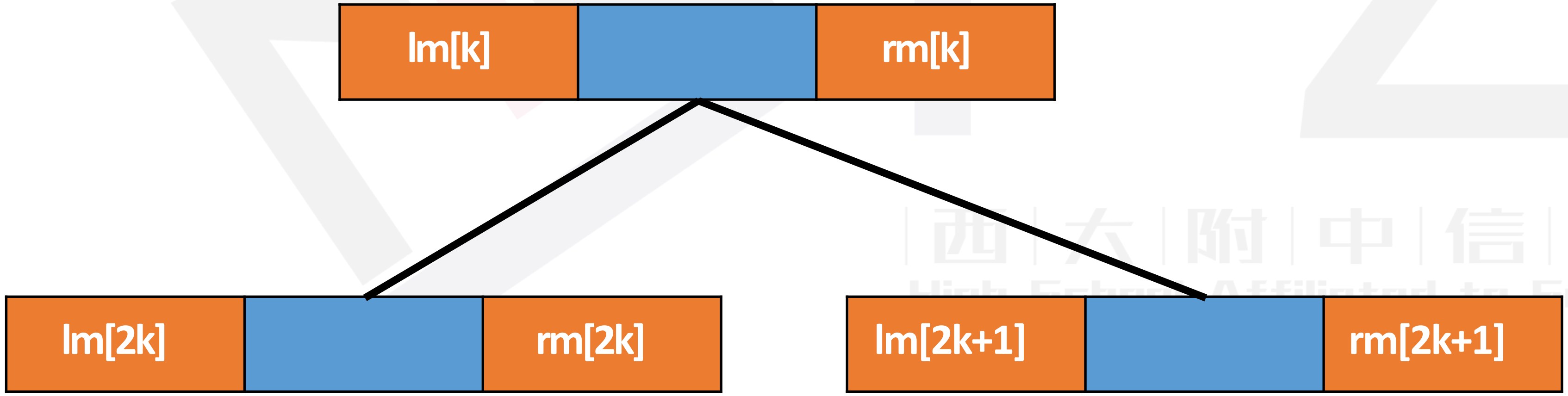
lm[]——为左端点l 开始最大连续0的长度

rm[]——为右端点r开始最大连续0的长度

m[]——为区间[l , r] 最大连续0的长度

m[k]=?

max{lm[k] , rm[k] , rm[2k]+lm[2k+1],m[2k],m[2k+1]}



线段树——区间合并

对于结点k，表示区间[l , r] 维护三个数组：

lm[]——为左端点l 开始最大连续0的长度

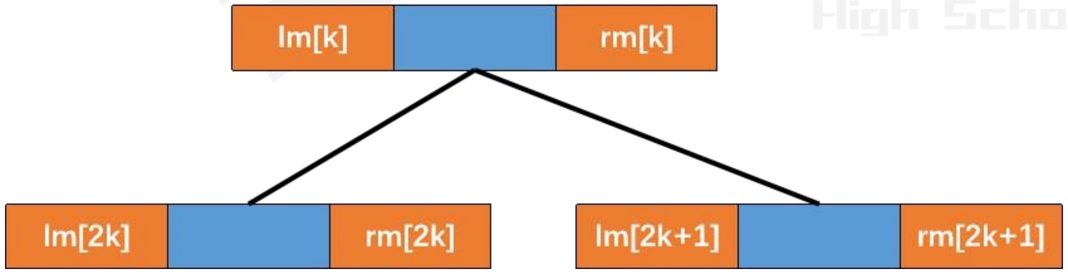
rm[]——为右端点r开始最大连续0的长度

m[]——为区间[l , r] 最大连续0的长度

询问连续为0的长度D的最小起点编号：

m[]可以判断区间是否存在，但不能确定起点

当出现rm[2k]+lm[2k+1]>=D时，能够确定起点为：mid-rm[2k]+1 ，mid=(l+r)/2。



对于结点k，表示区间[l , r] 维护三个数组：

lm[]——为左端点l 开始最大连续0的长度

rm[]——为右端点r开始最大连续0的长度

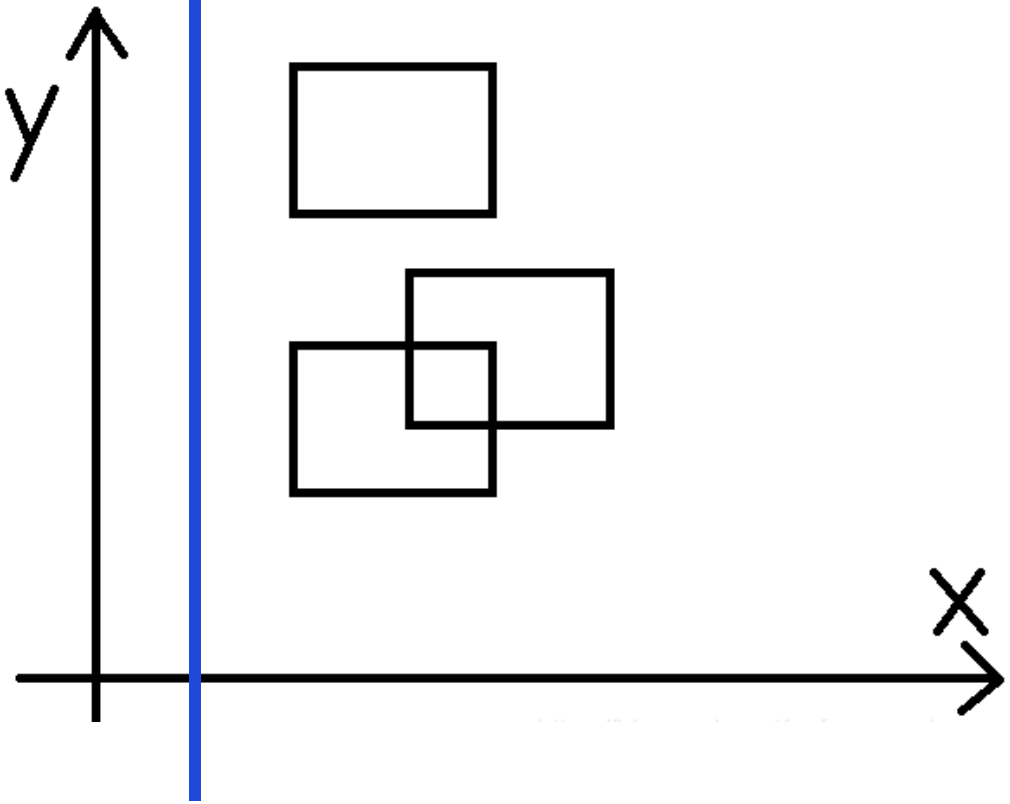
m[]——为区间[l , r] 最大连续0的长度

```
int ask(int k,int l,int r,int D)
{
    if(l==r)
    {
        if(D==1) return l;
        else return 0;
    }
    if(lazy[k]) pushdown(k,l,r);
    int mid=(l+r)/2;
    if(m[k*2]>=D) return ask(k*2,l,mid,D);//全部在左孩子里取
    if(rm[k*2]+lm[k*2+1]>=D) return mid-rm[k*2]+1;//左右孩子各取一部分
    else return ask(k*2+1,mid+1,r,D);//全部在右孩子里取
}
```

新的一些问题1

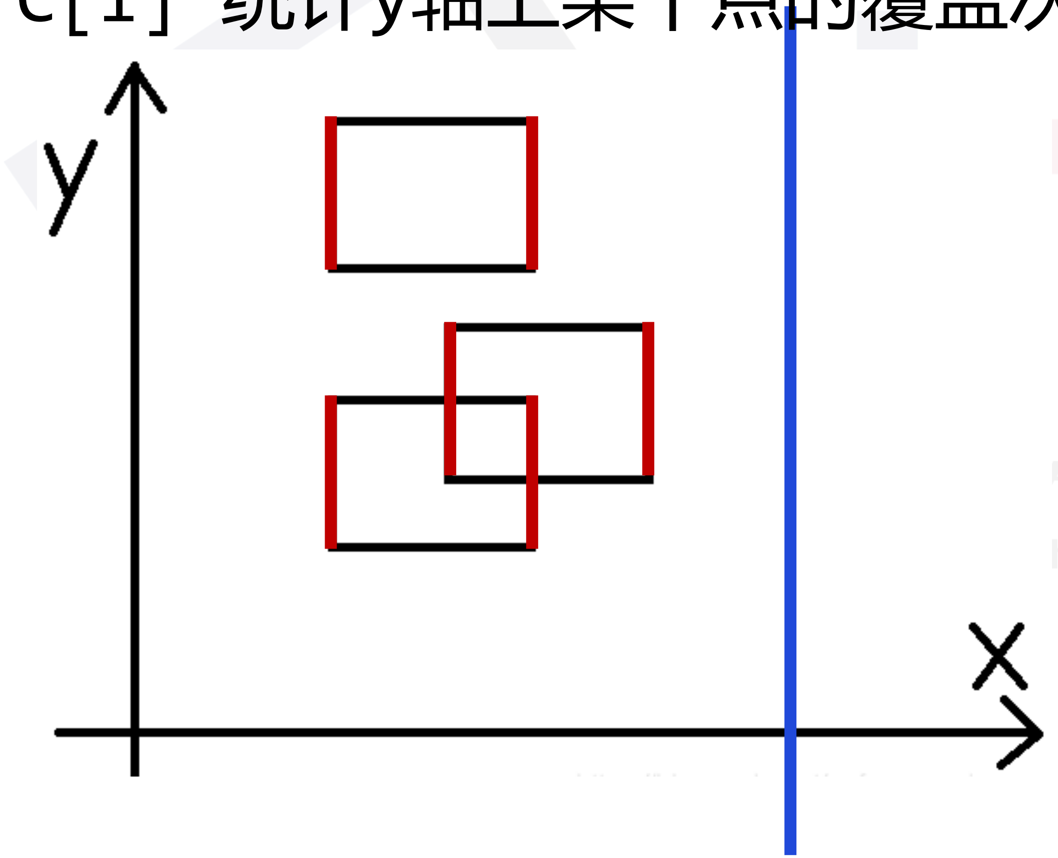
问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数



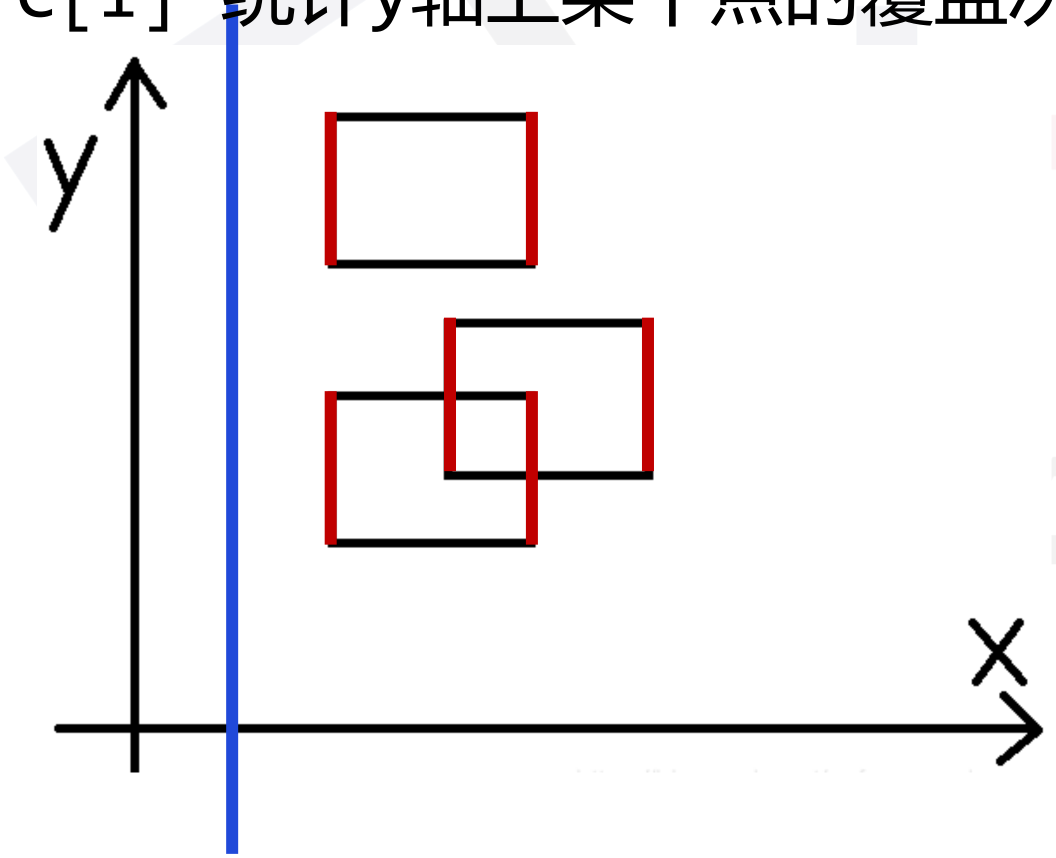
问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数



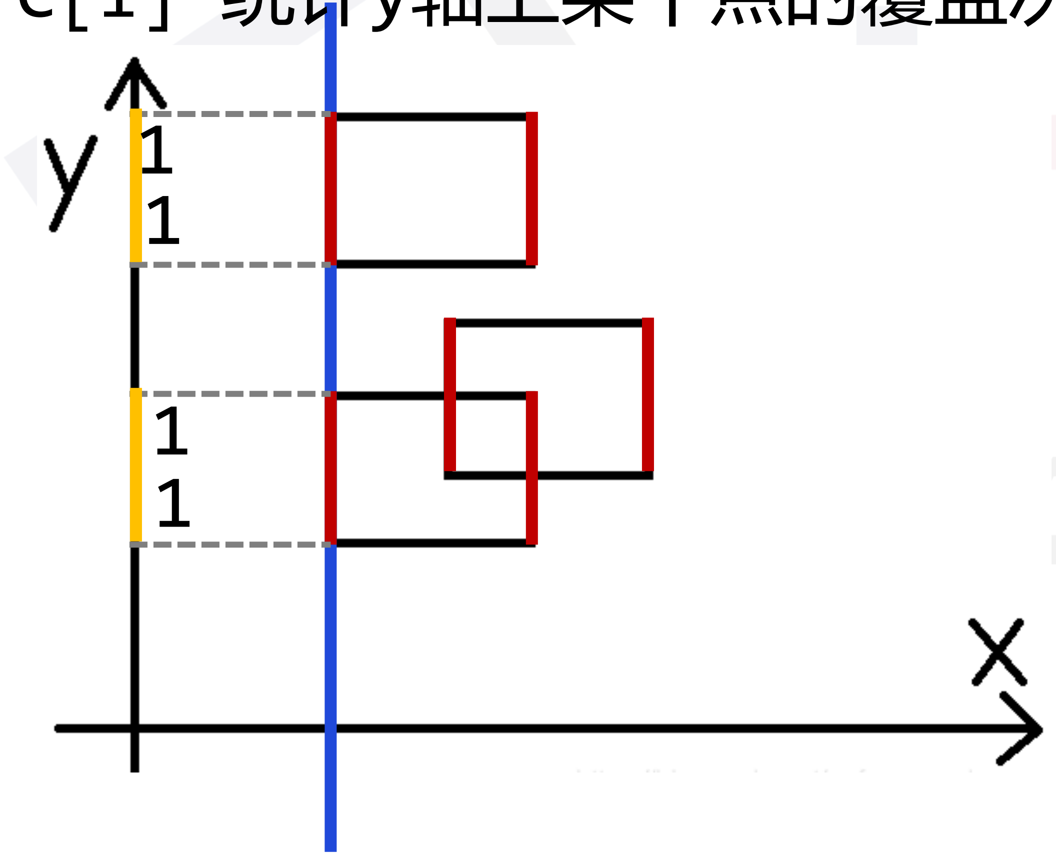
问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数



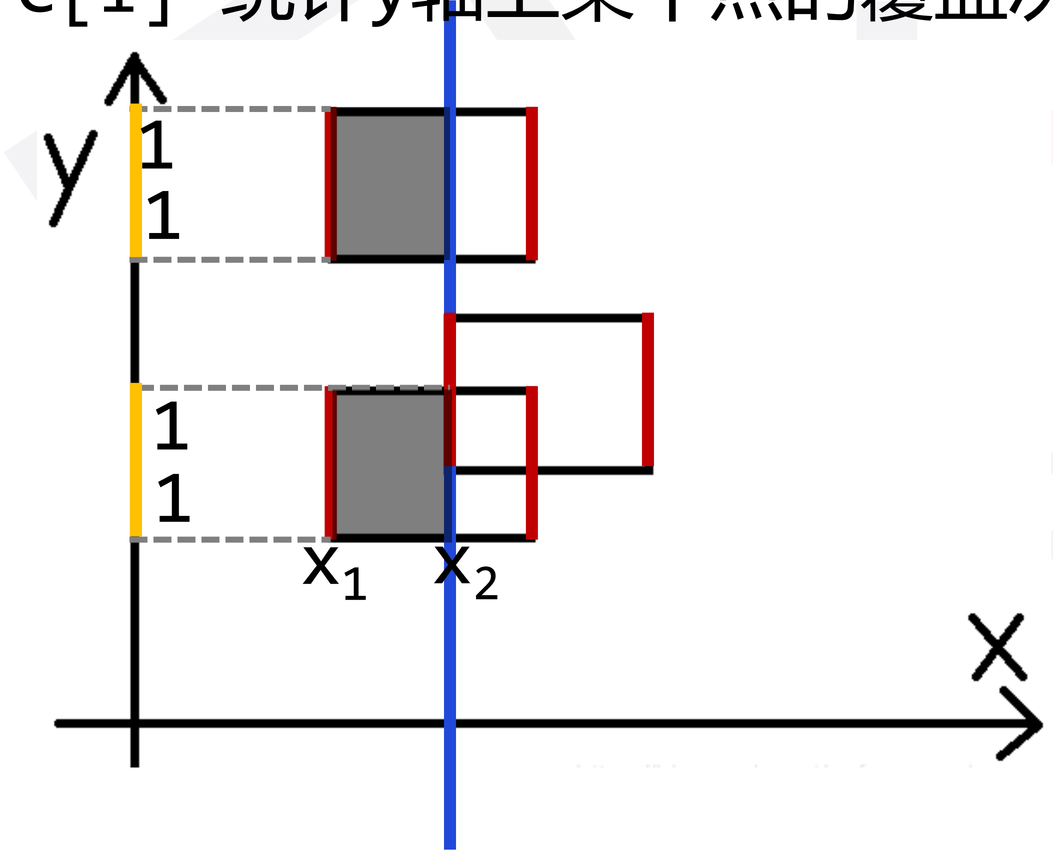
问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

c[i] 统计y轴上某个点的覆盖次数



问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数



C[i] 中>0的个数？

4个

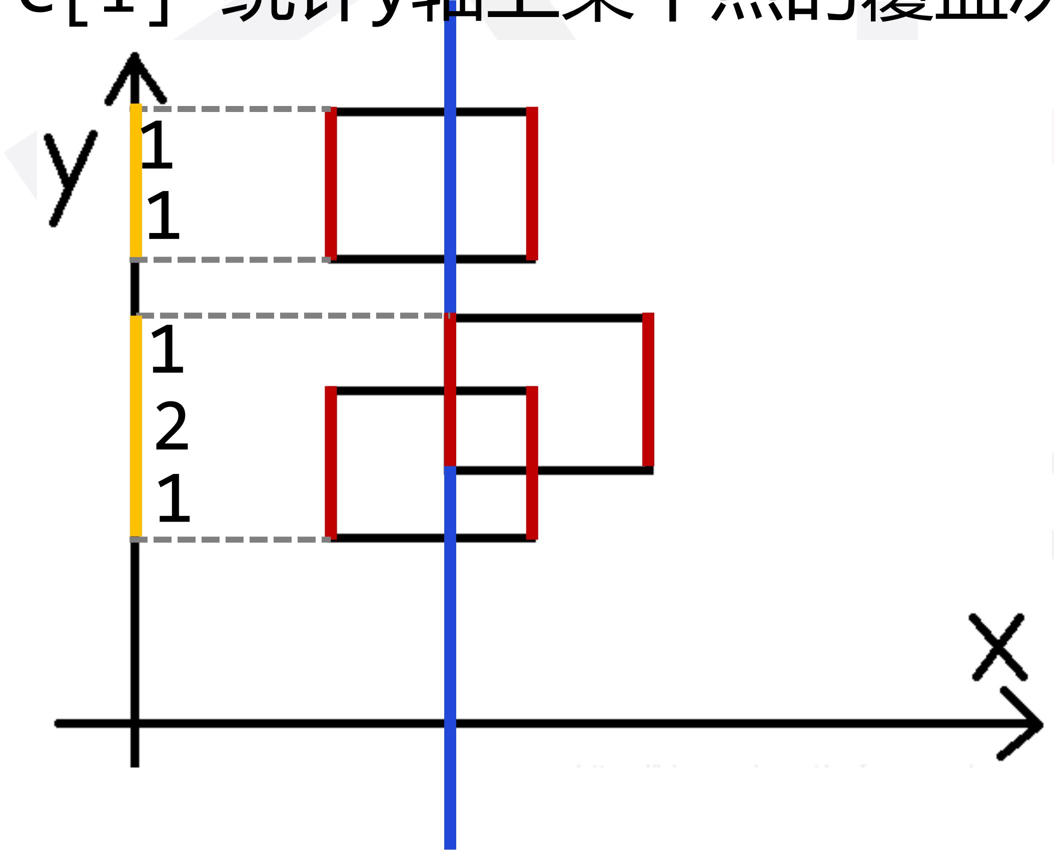
表示4个点被覆盖

覆盖的面积？

$4*(x_2-x_1)$

问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

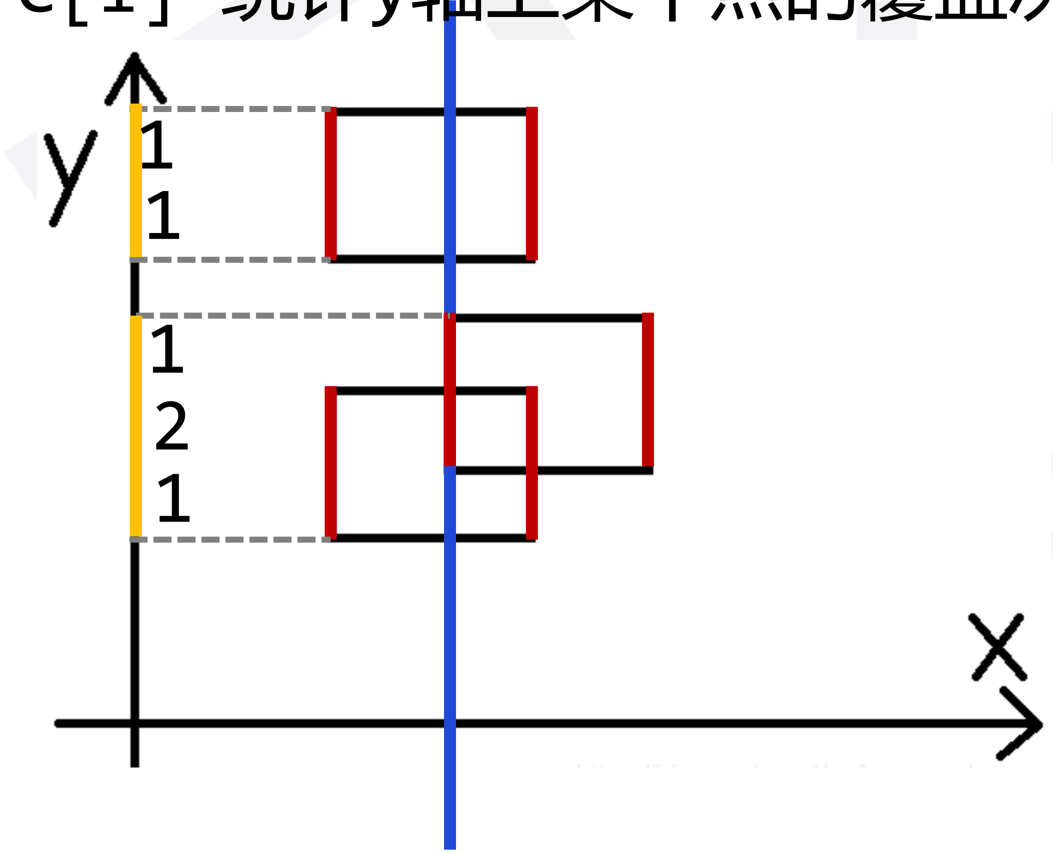
c[i] 统计y轴上某个点的覆盖次数



遇到左边界
就更新c[i]

问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

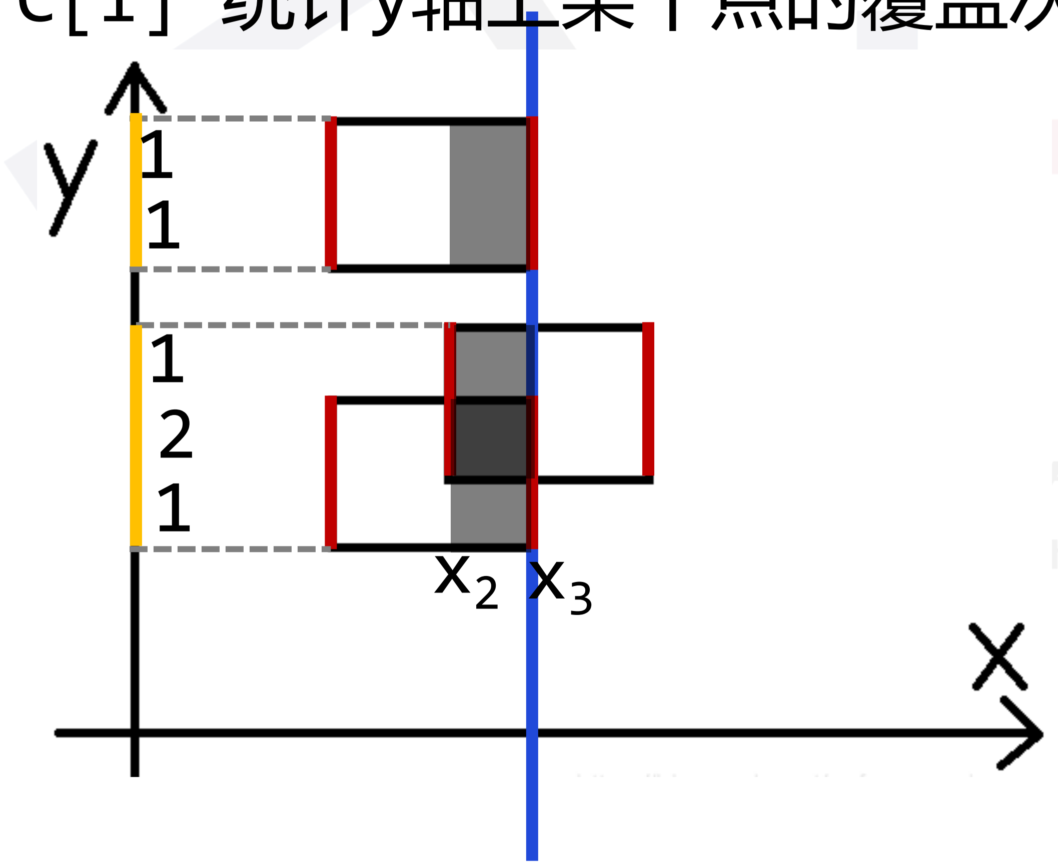
C[i] 统计y轴上某个点的覆盖次数



继续向后移动

问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数



C[i] 中>0的个数？

5个

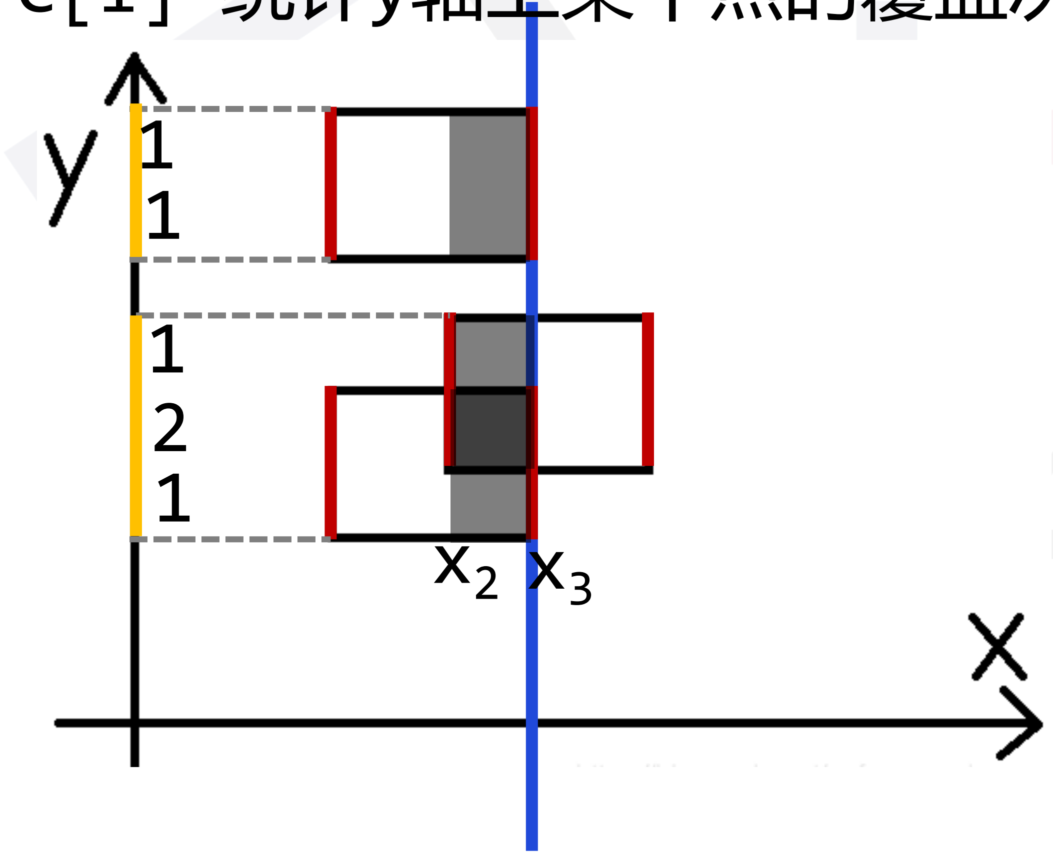
表示5个点被覆盖

覆盖的面积？

$5*(x_3-x_2)$

问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数

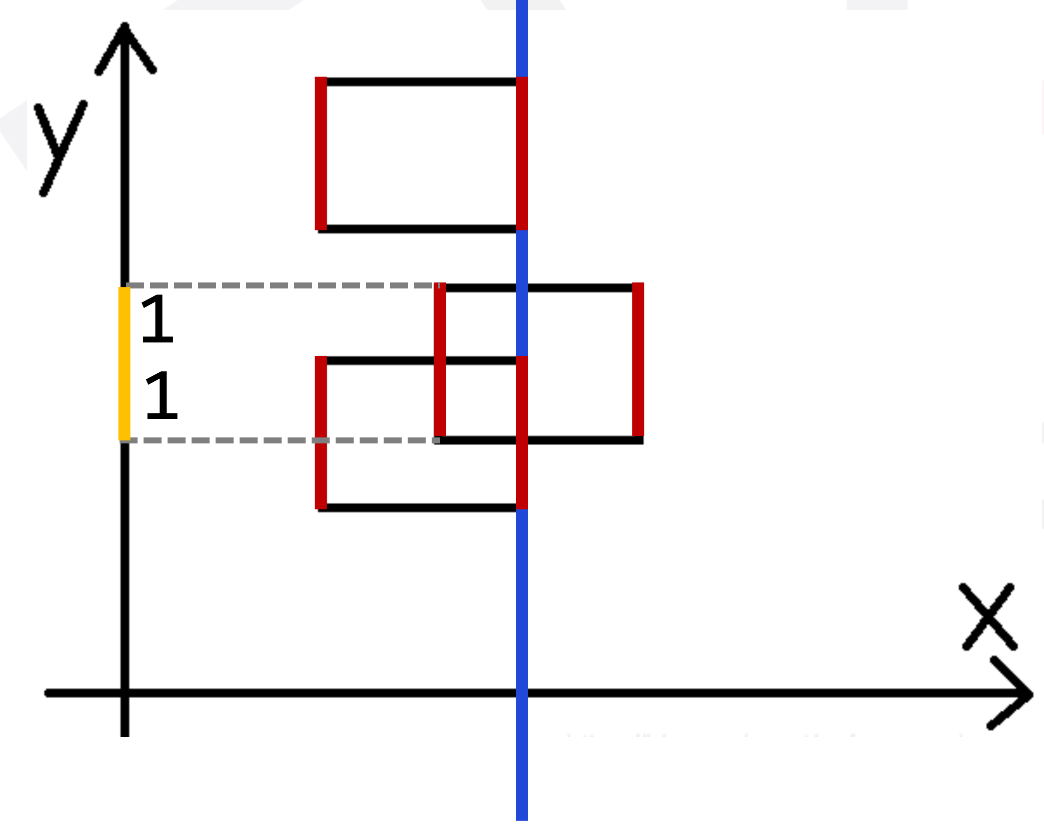


遇到了右边界

做区间修改

问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数

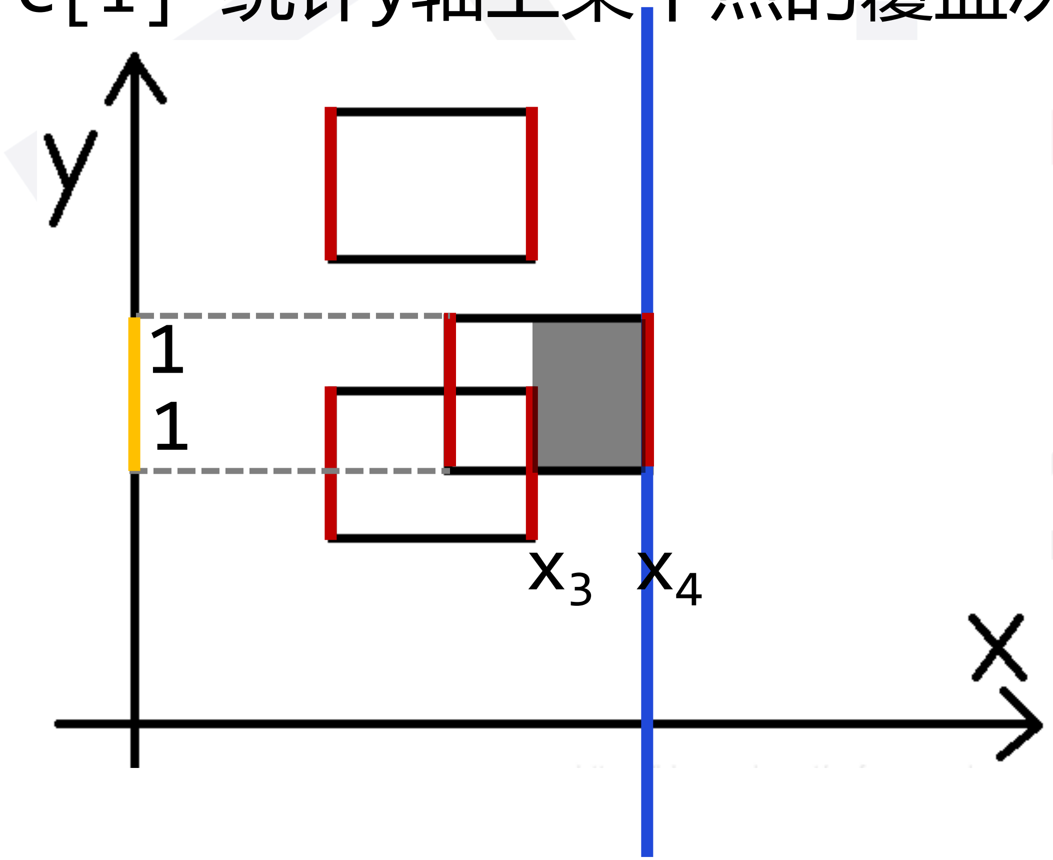


遇到了右边界

做区间修改

问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

c[i] 统计y轴上某个点的覆盖次数



c[i] 中>2的个数？

2个

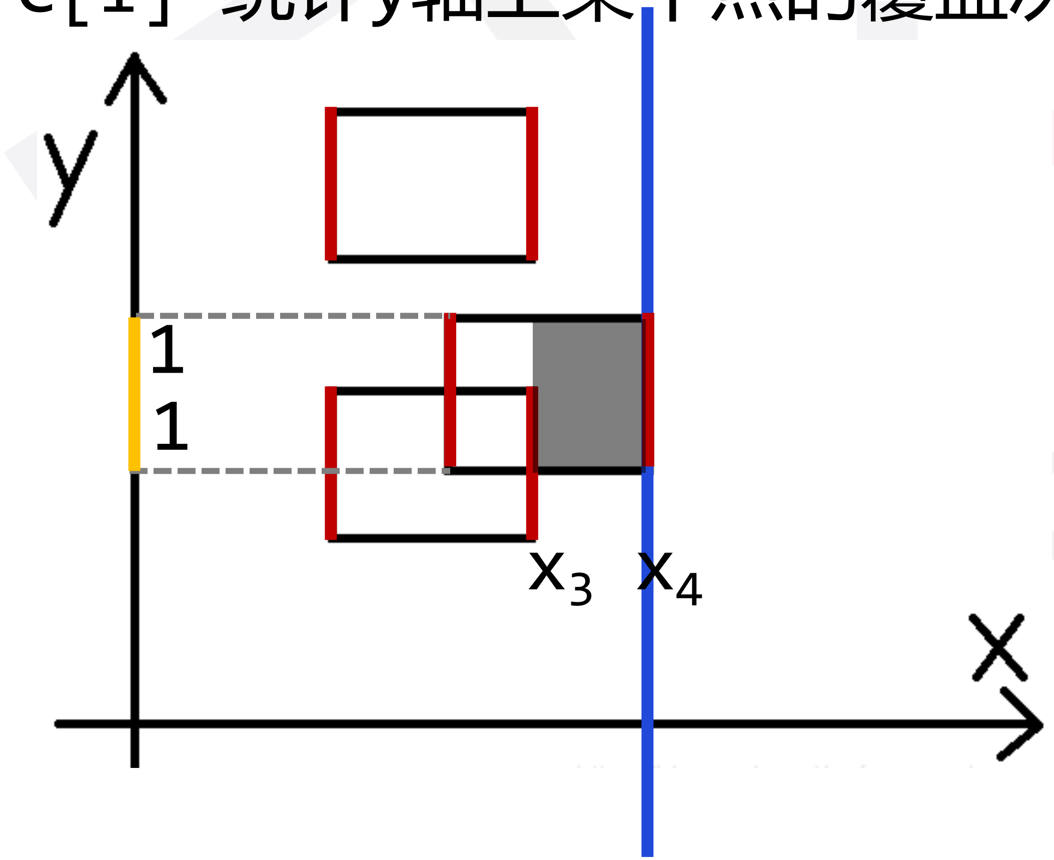
表示2个点被覆盖

覆盖的面积？

$2*(x_4-x_3)$

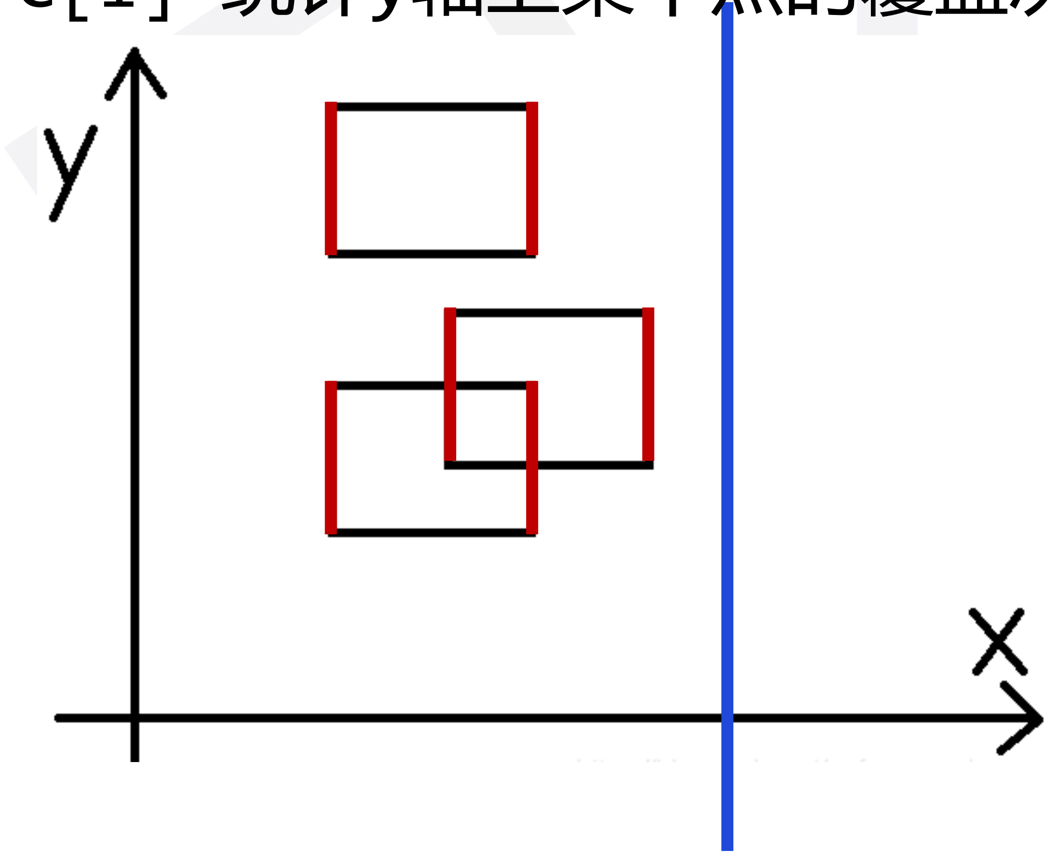
问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数



问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数

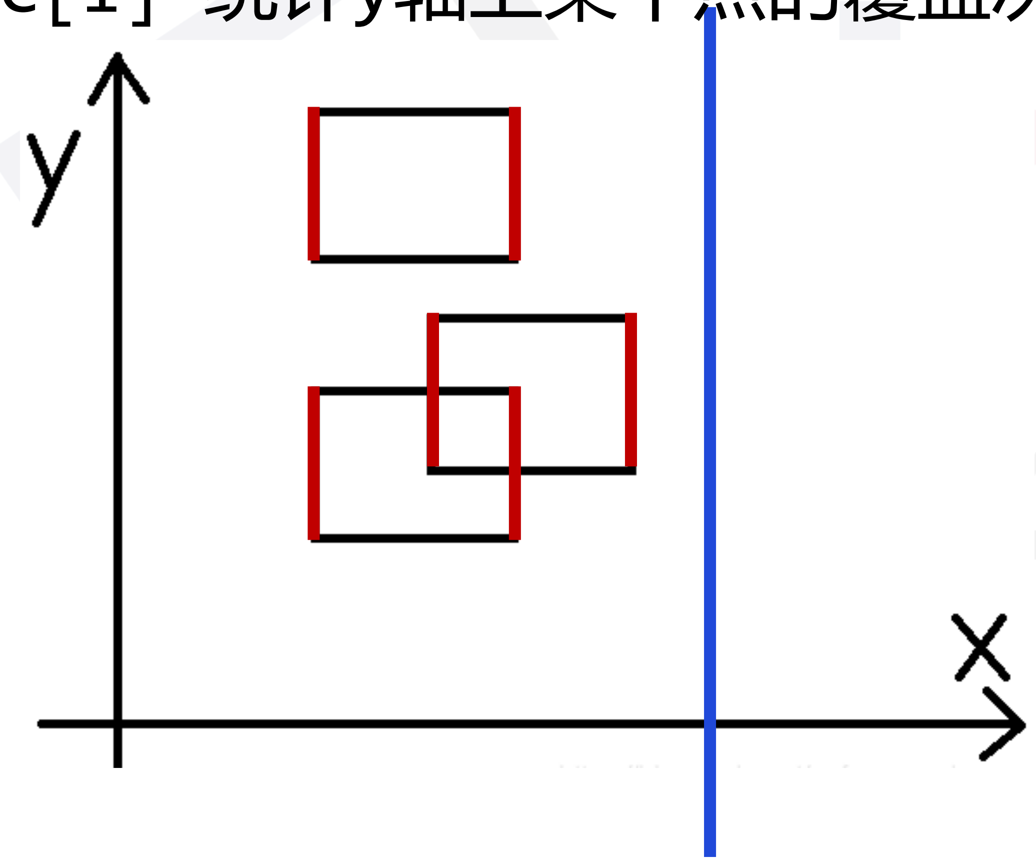


右边界，做区间修改

直到扫描完成为止。

问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数



遇到边界，则意味着改变

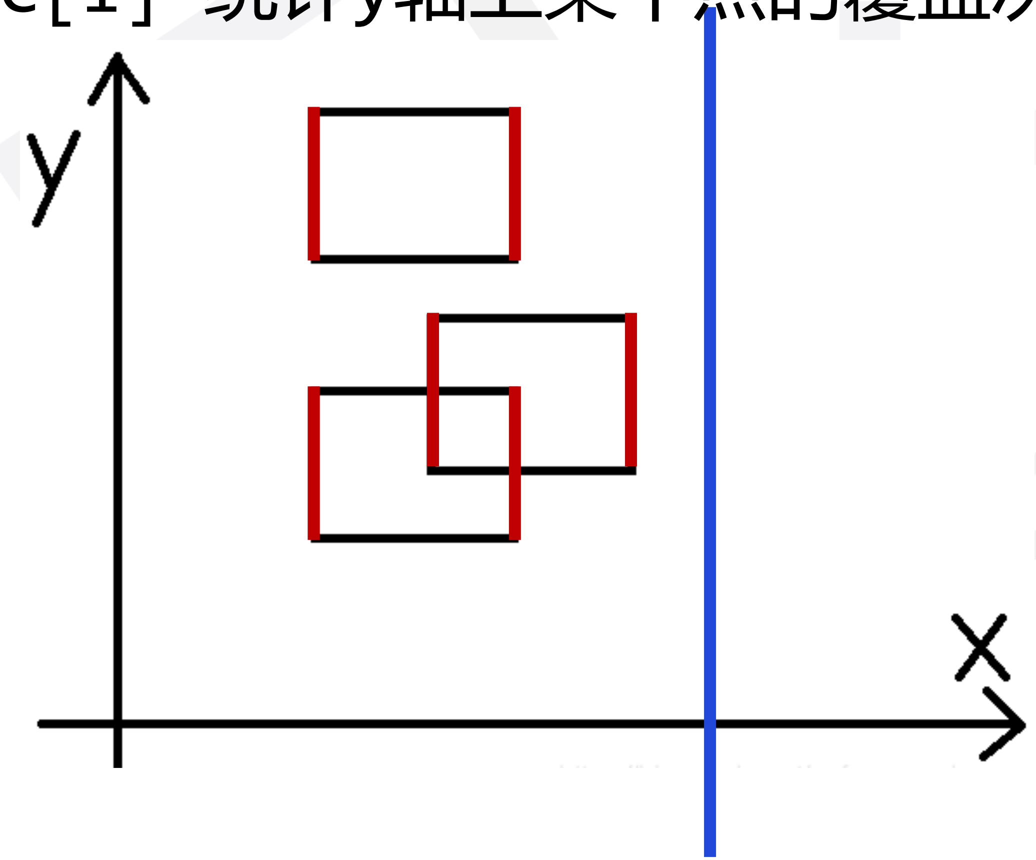
1. 对之前扫过的面积求和
2. 通过C数组，统计当前扫过的点数
3. 扫过的面积=∑（点数*(x2-x1)）

我们可以暴力处理C数组

复杂度N^2

问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数



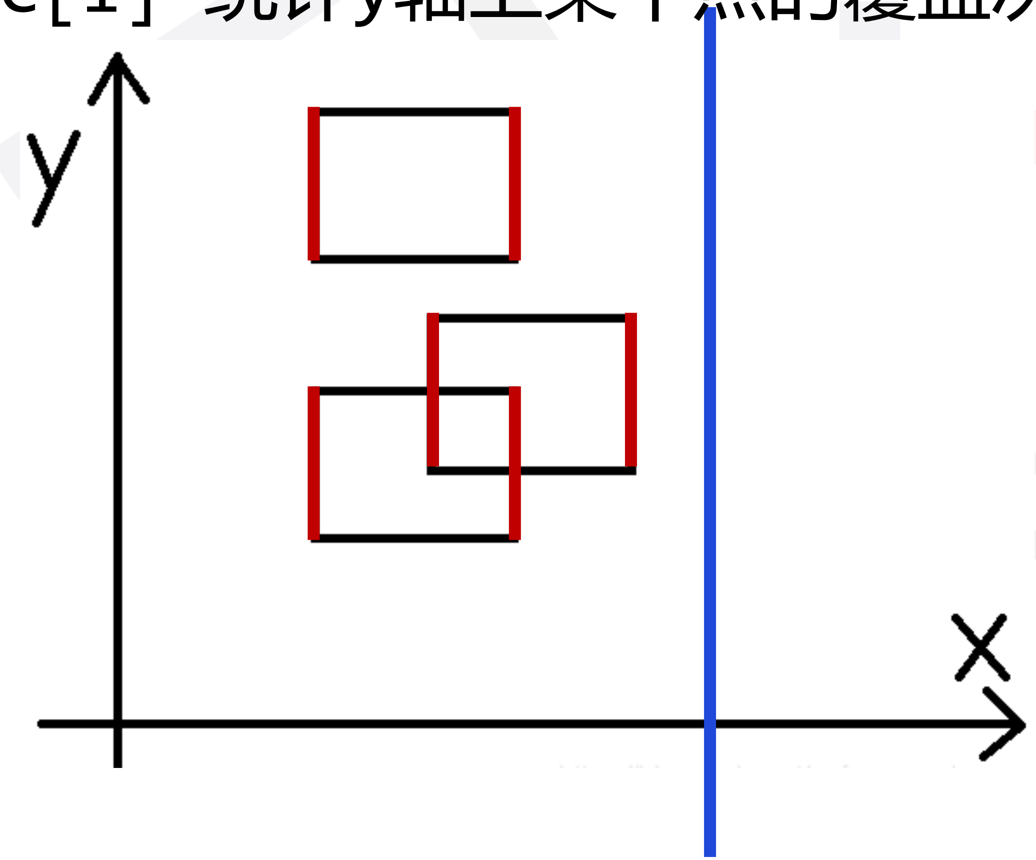
遇到边界，则意味着改变

1. 对之前扫过的面积求和
2. 通过C数组，统计当前扫过的点数
3. 扫过的面积=∑（点数*(x2-x1)）

当然也可以优雅线段树
处理什么操作？

问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数



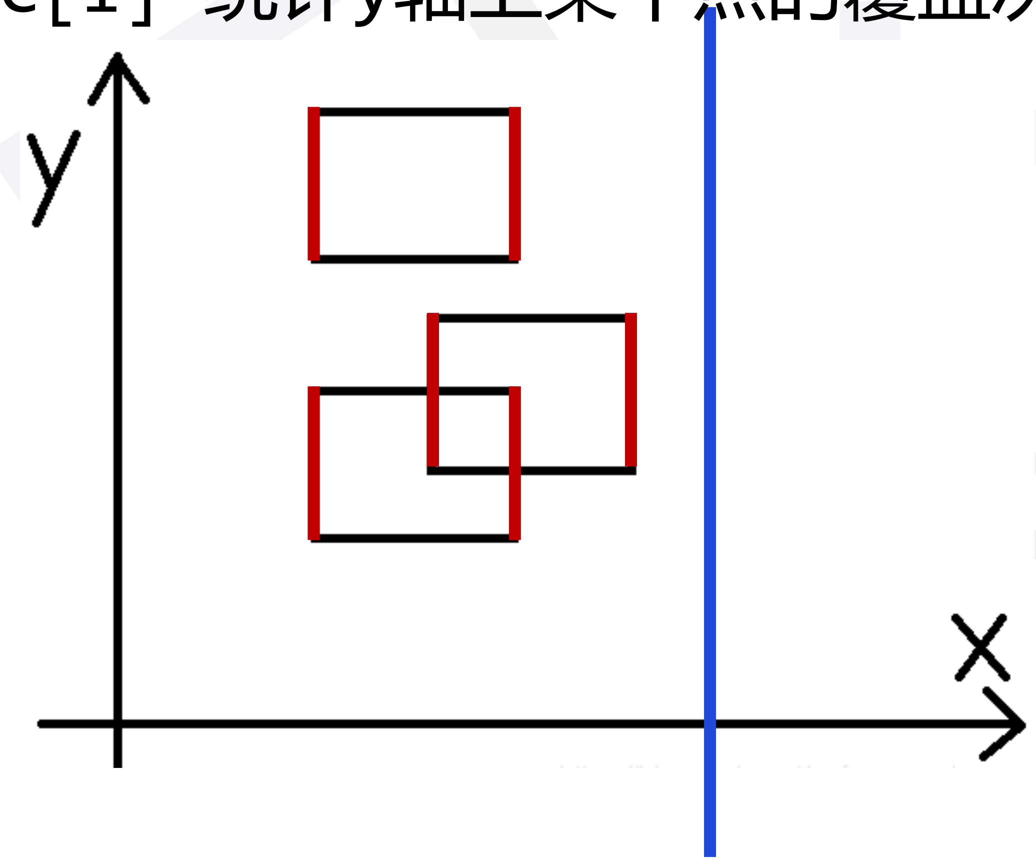
遇到边界，则意味着改变

1. 对之前扫过的面积求和
2. 通过C数组，统计当前扫过的点数
3. 扫过的面积=∑（点数*(x2-x1)）

1. 区间修改（+1，-1）
2. 区间查询（C中非0个数）
3. 复杂度NlogN

问题：在坐标轴上有若干个矩形，问他们覆盖的面积总和。

C[i] 统计y轴上某个点的覆盖次数



这条蓝色的线
俗称

扫描线

例Atlantis POJ 1151

给出矩形的左下和右上两个端点的横纵坐标，求出所有矩形的面积，覆盖部分算一次。

第一行输入一个n
接下来n行，n个矩形的左下和右上端点的坐标

```
2
10 10 20 20
15 15 25 25.5
```

注意y不超过int，有小数

例 Atlantis POJ 1151

- Y值过大的问题
- 1. C数组过大
- 2. double 精度

- 解决：离散化

Atlantis

```
#define lson (rt<<1)
#define rson (rt<<1|1)
#define MAXN 200
struct NODE {
    int l, r;
    double left, right, len;
    int cover; //记录覆盖情况
} sgtr[MAXN<<2];


double y[MAXN<<1]; //对应的序号装对应的边
struct E {
    double x,y1, y2;
    int flag; //标记入边还是出边
} e[MAXN<<1];
bool cmp(E s, E t){return s.x < t.x;}
void build(int rt, int left, int
right){//建树
    sgtr[rt].l = left;
    sgtr[rt].r = right;
    sgtr[rt].left = y[left];
    sgtr[rt].right = y[right];
    sgtr[rt].len = 0;
    sgtr[rt].cover = 0;
    if (sgtr[rt].l + 1 == sgtr[rt].r)
return;
    int m = (left + right) / 2;
    build(lson, left, m);
    build(rson, m, right);
}
```

```
void updata(int rt, E b) {
    if (b.y1 == sgtr[rt].left && b.y2 == sgtr[rt].right){ //命中
        sgtr[rt].cover += b.flag;
    } else if (b.y2 <= sgtr[lson].right) {//在左子树中找（包含完整区间）
        updata(lson, b);
    } else if (b.y1 >= sgtr[rson].left){ //在右子树中找（包含完整区间）。
        updata(rson, b);
    } else{ //分成两段，左右各一部分
        E temp;
        temp = b; temp.y2 = sgtr[lson].right; updata(lson, temp);
        temp = b; temp.y1 = sgtr[rson].left; updata(rson, temp);
    }
    //更新区间长度
    if (sgtr[rt].cover > 0) sgtr[rt].len = sgtr[rt].right - sgtr[rt].left;
    else if (sgtr[rt].cover == 0 && sgtr[rt].r - sgtr[rt].l == 1)  sgtr[rt].len = 0;
    else sgtr[rt].len = sgtr[lson].len + sgtr[rson].len; //长度递归到线段树的根
}
int main(){
    int n, cases = 1;
    while (~scanf("%d", &n) && n) {
        int t = 1;
        for (int i = 1; i <= n; i++,t += 2) { //一边读入一边离散化
            scanf("%lf%lf", &e[t].x, &y[t]);
            scanf("%lf%lf", &e[t + 1].x, &y[t + 1]);
            e[t].y1 = y[t];      e[t].y2 = y[t + 1];      e[t].flag = 1;
            e[t + 1].y1 = y[t];  e[t + 1].y2 = y[t + 1];  e[t + 1].flag = -1;
        }
        t--;
        sort(e + 1, e + t+1, cmp); //对边信息排序
        sort(y + 1, y + t+1);
        build(1, 1, t);//以1为根节点，建立区间在1~t的树。
        double sum = 0;
        for (int i = 1; i <= t; i++) {
            sum += sgtr[1].len * (e[i].x - e[i-1].x);//阶段性求面积
            updata(1, e[i]); //新加边
        }
        printf("Test case #%d\n", cases++);
        printf("Total explored area: %.2f\n\n", sum);
    }
}
```

例 Atlantis POJ 1151

西南大学附属中学
High School Affiliated to Southwest University

- 一个优化

在本题中，我们只关心整个扫描线（线段树根节点）上被矩形覆盖的长度。而且，因为四元组 $(x,y_1,y_2,1)$ 和 $(x,y_1,y_2,-1)$ 成对出现，所以线段树区间修改也是成对出现的。在这种特殊情形下，我们没有必要下传延迟标记，可以采用更为简单的做法。

除左右端点 l,r 之外，在线段树的每个节点上维护两个值：该节点代表的区间被矩形覆盖的长度 len，该节点自身被覆盖的次数 cnt。最初，二者均为 0。

对于一个四元组 (x,y_1,y_2,k)，我们在 $[val(y_1),val(y_2)-1]$ 上执行区间修改。该区间被线段树划分成 $O(\log N)$ 个节点，我们把这些节点的 cnt 都加 k。

对于线段树中任意一个节点 $[l,r]$，若 $cnt>0$，则 len 等于 $raw(r+1)-raw(l)$。否则，该点 len 等于两个子节点的 len 之和。在一个节点的 cnt 值被修改，以及线段树从下往上传递信息时，我们都按照该方法更新 len 值。根节点的 len 值就是整个扫描线上被覆盖的长度。

Code Atlantis POJ 1151


```cpp
#include <bits/stdc++.h>
using namespace std;
const int N=120;
struct line{
   double x,d,u;int flag;
}a[N<<1];
struct node{
   int l,r,cnt;
   double len;
}v[N<<3];
int n,t,val[N<<1][2],T;
double raw[N<<2],ans;
bool compare(line p1,line p2){return p1.x<p2.x;};
void input(){
   for (int i=1;i<=n;i++)
   {
       double x1,y1,x2,y2;
       scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
       a[2*i-1] = (line){x1,y1,y2,1};
       a[2*i] = (line){x2,y1,y2,-1};
       raw[++t] = y1 , raw[++t] = y2;
   }
   sort(a+1,a+2*n+1,compare);
}
void discrete(){  //离散化
   sort(raw+1,raw+t+1);
   t = unique(raw+1,raw+t+1) - (raw+1);
   for (int i=1;i<=2*n;i++){
       val[i][0] = lower_bound(raw+1,raw+t+1,a[i].d) - raw;
       val[i][1] = lower_bound(raw+1,raw+t+1,a[i].u) - raw;
   }
}
void updata(int p){
   if (v[p].cnt>0) v[p].len = raw[v[p].r+1] - raw[v[p].l];
   else if (v[p].l==v[p].r) v[p].len = 0;
   else v[p].len = v[ p<<1 ].len + v[ p<<1|1 ].len;
}
void build(int p,int l,int r){
   v[p].l = l , v[p].r = r;
   if (l==r){v[l].cnt = v[l].len = 0; return;}
   int mid = l+r >> 1;
   build( p<<1 , l , mid );
   build( p<<1|1 , mid+1 , r );
}
void modify(int p,int l,int r,int delta){
   if (l<=v[p].l&&r>=v[p].r){
       v[p].cnt += delta;
       updata(p); return;
   }
   int mid = v[p].l+v[p].r >> 1;
   if (l<=mid) modify( p<<1 , l , r , delta );
   if (r>mid) modify( p<<1|1 , l , r , delta );
   updata(p);
}
double query(){return v[1].len;}
void solve(){
   for (int i=1;i<=2*n;i++){
       modify(1,val[i][0],val[i][1]-1,a[i].flag);
       ans += (a[i+1].x-a[i].x) * query();
   }
}
int main(){
    while ( scanf("%d",&n) && n ){
        ans = t = 0;
        input();
        discrete();
        build(1,1,t);
        solve();
        printf("Test case #%d\nTotal 
explored area: %.2lf\n\n",++T,ans);
    }
    return 0;
}
```

扫描线应用

- 矩形面积问题
- 矩形周长问题
- 多边形面积问题

- 属于计算几何的范畴。
- 线段树多用于处理一维空间上的几何统计

递归的大常数

- 递归实现的线段树
- 常数太大?
- T的飞起?

- 非递归版本线段树？
- zkw线段树 （我们后面讲）

众所周知 zkw =张昆玮

zkw PPT在学习资料里，大家可以先看看。

新的一些问题2

一些问题2

- 之前维护了区间的一些信息。
- 现在我们尝试维护节点“表示一个区间的数**出现的次数**”

- 为了统计“出现次数” 之前是怎么做的？
- 开个桶

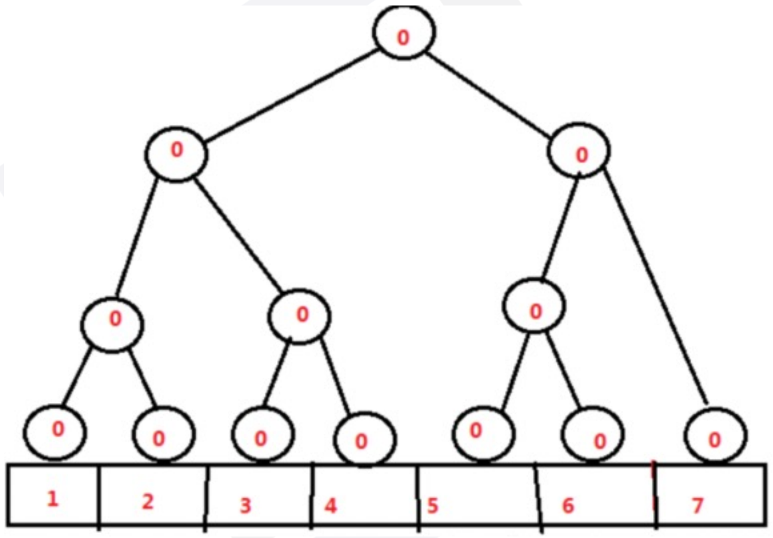
- 现在呢？
- 也开一个。。。。
- 然后在桶上建一个线段树

- 这种将权值作为统计对象，将值域作为线性空间分治的数据结构叫

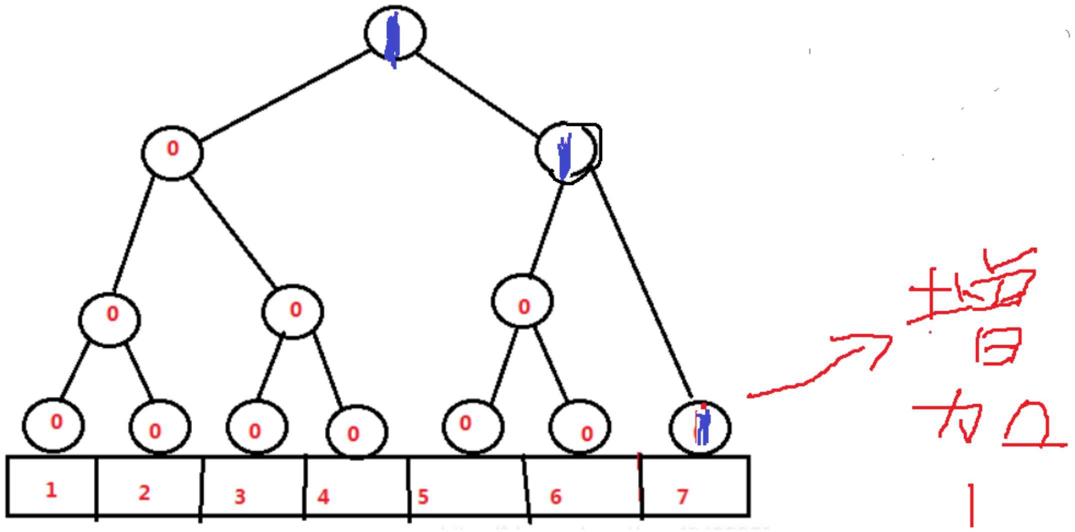
·权值线段树

权值线段树-图例

最初有一个序列 7 2 3 5 6 1 4



依次先插入7，线段树变为下图



权值线段树-定义

- 每个节点用来表示一个值域区间内的数**出现的次数。**
- 实现了桶的功能，并支持按区间统计。

权值线段树-应用

- 一段区间的数的出现次数
- 整个数列第k大/小的问题（及其应用）

权值线段树-实现

添加值

```
void add(int l,int r,int v,int x){
    if(l==r) f[v]++;
    else{
        int mid=(l+r)/2;
        if(x<=mid) add(l,mid,v*2,x);
        else add(mid+1,r,v*2+1,x);
        f[v]=f[v*2]+f[v*2+1];
    }
}
```

单点查询值x出现次数

```
int find(int l,int r,int v,int x){
    if(l==r) return f[v];
    else{
        int mid=(l+r)/2;
        if(x<=mid) return find(l,mid,v*2,x);
        else return find(mid+1,r,v*2+1,x);
    }
}
```

权值线段树-实现

区间[x,y]查询值v出现次数

```
int find(int l,int r,int v,int x,int y){
    if(l==x&&r==y) return f[v];
    else{
        int mid=(l+r)/2;
        if(y<=mid) return find(l,mid,v*2,x,y);
        else if(x>mid) return find(mid+1,r,v*2+1,x,y);
        else return find(l,mid,v*2,x,mid)+find(mid+1,r,v*2+1,mid+1,y);
    }
}
```

查询整个区间的第k大值？

1. 到每个节点时，如果右子树的总和大于等于 k 递归进右子树；
2. 否则说明此时的第 k大值在左子树中，则递归进左子树
3. 注意：此时要将 k的值减去右子树的总和。(why)
4. 最后一直递归到只有一个数时，那个数就是答案。

```
int kth(int l,int r,int v,int k){
    if(l==r) return l;
    else{
        int mid=(l+r)/2,s1=f[v*2],s2=f[v*2+1];
        if(k<=s2) return kth(mid+1,r,v*2+1,k);
        else return kth(l,mid,v*2,k-s2);
    }
}
```

权值线段树-实现

查询部分区间的第k大值？

需要用到主席树（挖个坑，后面再讲）
同样也提供自学资料。
你现在自学完，马上就可以A掉NOI Online TG T1

求一个数在全局的排名？

排名数值上等于全局比那个数小的数的个数+1，那么放在权值线段树里就等价于前缀和查询

因为节点维护的是区间内出现的数的次数，例如我们求x的排名我们只需要查询[1,x-1]这个区间内的前缀和，然后加1就是x的排名了。

```
ll query(int L,int R,int l,int r,int rt){
    //普通的区间查询
    if(L<=l&&r<=R){
        return a[rt];
    }
    int m=l+r>>1;
    ll ans=0;
    if(L<=m)
        ans+=query(L,R,l,m,rt<<1);
    if(R>m)
        ans+=query(L,R,m+1,r,rt<<1|1);
    return ans;
}
ll frank(ll x){
    if(x==1) return 1;
    return query(1,x-1,1,n,1)+1;
}
```