线段树

线段树

问题模型

给定一个n（n <= 100000）个元素的数组A，待解决的问题有：

- 1、区间最值：有m(m <= 100000)个操作，共两种操作，

  （1）Q a b　询问：表示询问区间[a，b]的最大值；

  （2）C a c　更新：表示将第a个元素变成c。

- 2、区间求和：有m(m <= 100000)个操作，共两种操作，

  （1）Q a b　询问：表示询问区间[a，b]的元素和；

  （2）A a b c　更新：表示将区间[a，b]的每个元素加上一个值c。

线段树

- **定义：**线段树是一种二叉树，即每个结点最多有两棵子树，每个结点存储了一个区间，是一种维护区间信息的数据结构。

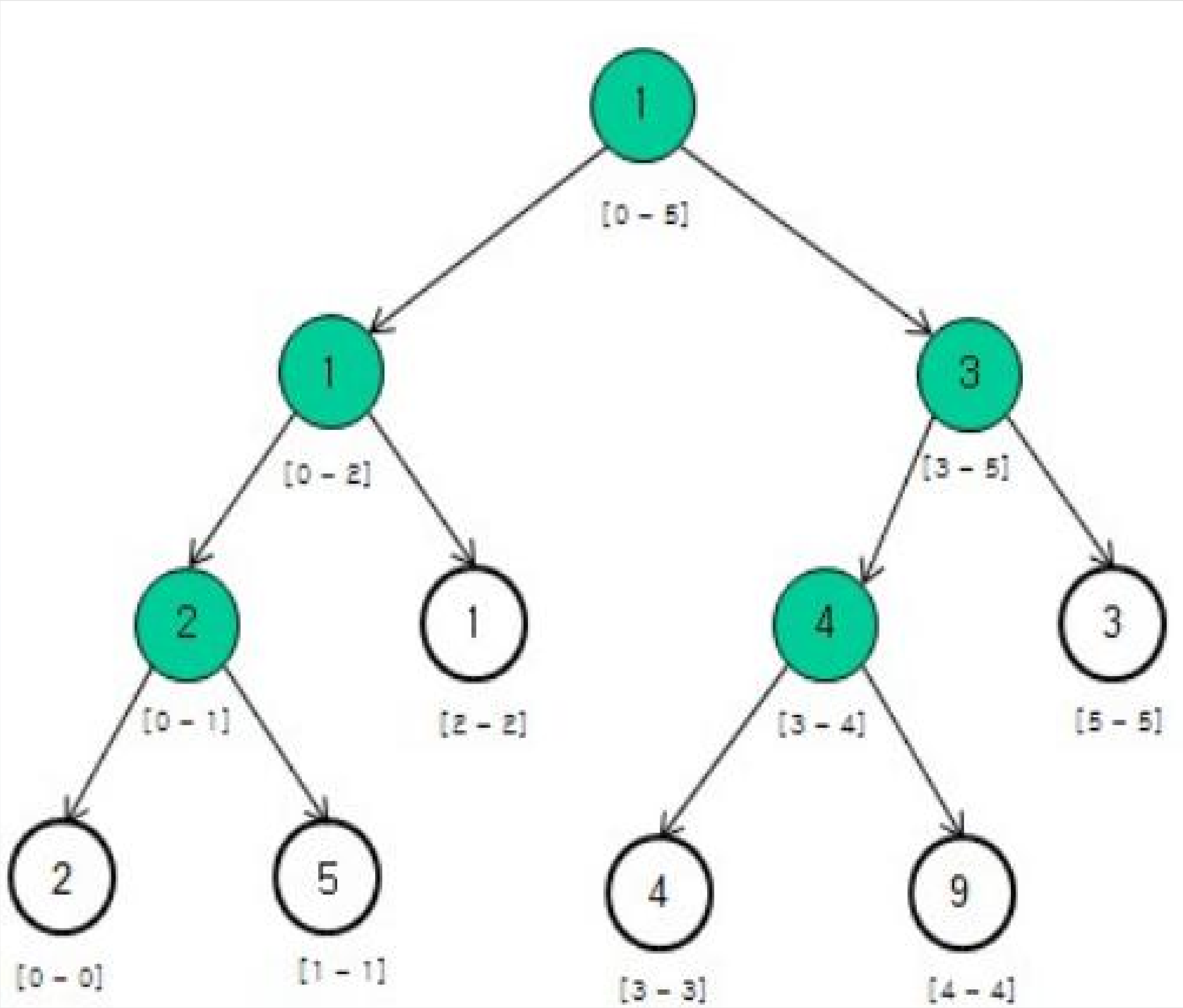
- **特点：** 线段树利用了区间拆分的思想，将一个区间划分成一些子区间，每个结点存储一个区间（用左、右端点表示，相当于一条线段）。
  - 每个子结点分别表示父结点的左、右各半区间，如果父结点的区间是[a, b]，那么左儿子的区间是[a, c]，右儿子的区间是[c+1, b]，其中c=└（a+b）/2┘。
  - 每个叶子结点对应一个单元区间，此时a=b。

线段树

线段树示例

数组[2, 5, 1, 4, 9, 3]，构造线段树。白色表示叶子结点，分支结点对应数组区间内的最小值。

线段树不考虑排序，不必遵循左小右大。



线段树

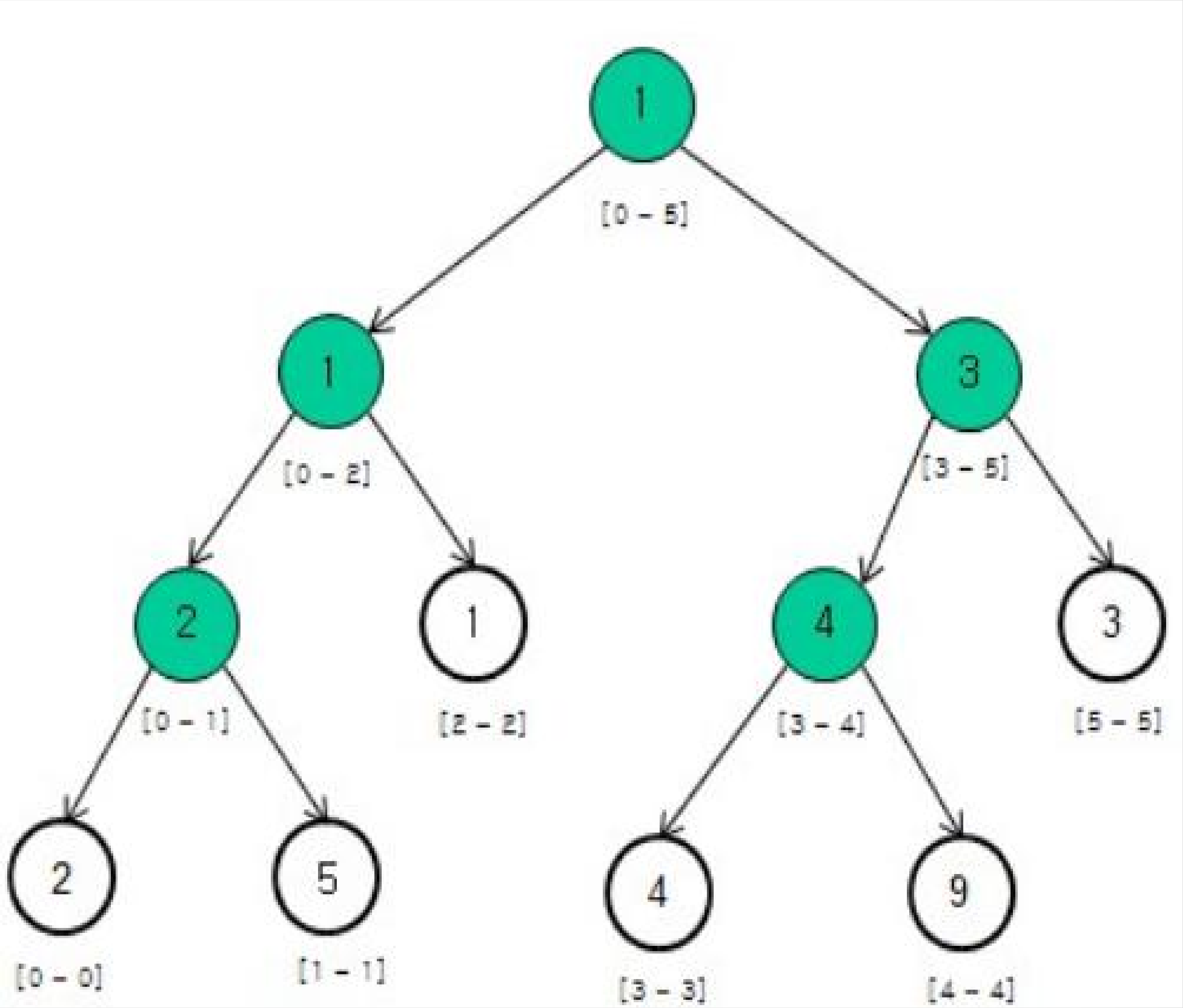
线段树空间分析

线段树的父结点区间是平均分割到左、右子树。

对于n个叶子结点一定有n-1个非叶子结点，总共有2n-1个结点。

一般的，在实际应用中，线段树按照完全二叉树来保存，采用数组形式（注意双亲结点下标与孩子结点下标的关系）。

往往使用4n空间避免越界。



线段树

- 基本操作
  - 创建
  - 查询
  - 更新

线段树

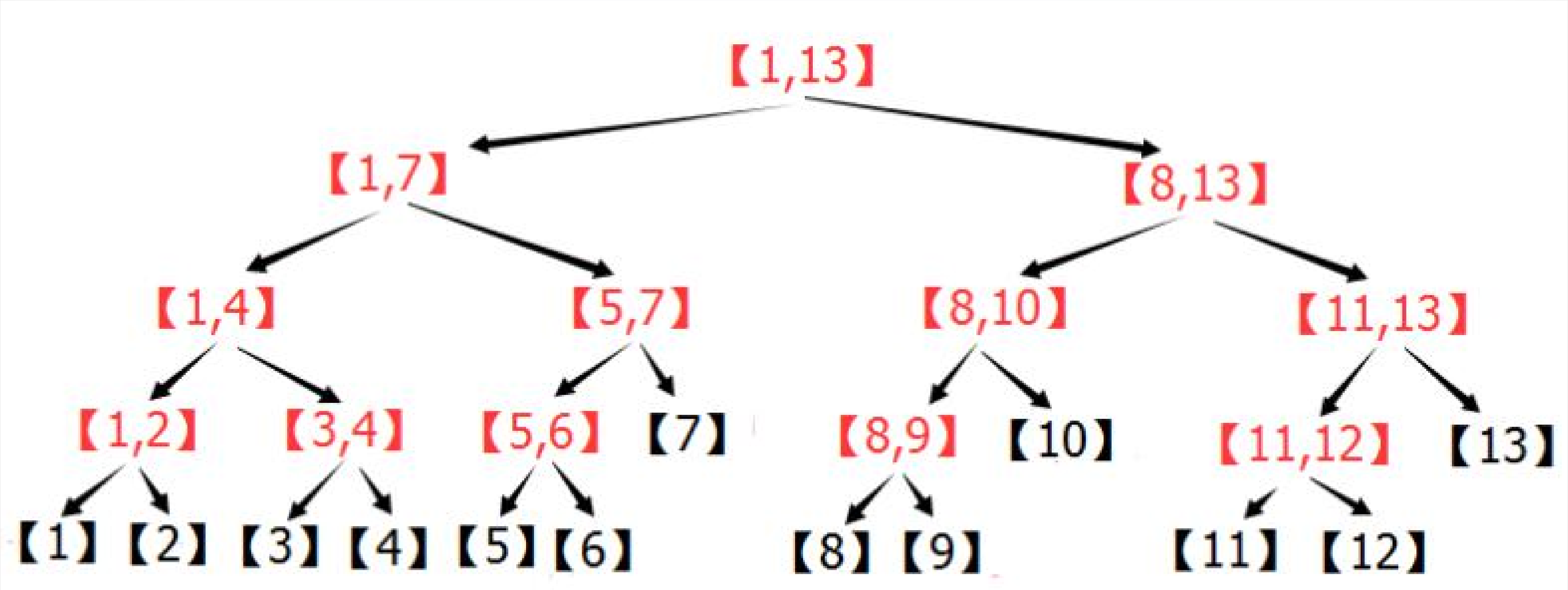
基本操作

线段树结点定义：

```
struct treeNode {
        Data val;            // 数据域
        int lStart,rEnd;     // 区间起始编号
        int sum;             // 区间和
        int vMin;            // 区间最小值
        int lazy;            // 延迟标记
        .......
}SegTree[ MAXNODES ];
```

线段树

- **线段树的创建：**与数组下标相关，与数值无关，树的最大高度为$\lfloor log_2 n \rfloor+1$。每个区间[L,R]分解成[L,M]和[M+1,R]（其中M=$\lfloor$(L+R)/2$\rfloor$）直到 L=R 为止。



线段树

- 线段树的创建示例代码：区间最小值

按完全二叉树保存（二叉树性质5）

```
void build(int root, int arr[], int istart, int iend)
{     if(istart == iend)//叶子节点
          segTree[root].val = arr[istart];
      else
      {   int mid = (istart + iend) / 2;
          build(root*2+1, arr, istart, mid);//递归构造左子树
          build(root*2+2, arr, mid+1, iend);//递归构造右子树
          //根据左右子树根节点的值，更新当前根节点的值
          segTree[root].val = min(segTree[root*2+1].val, segTree[root*2+2].val);
      }
 }
```

线段树

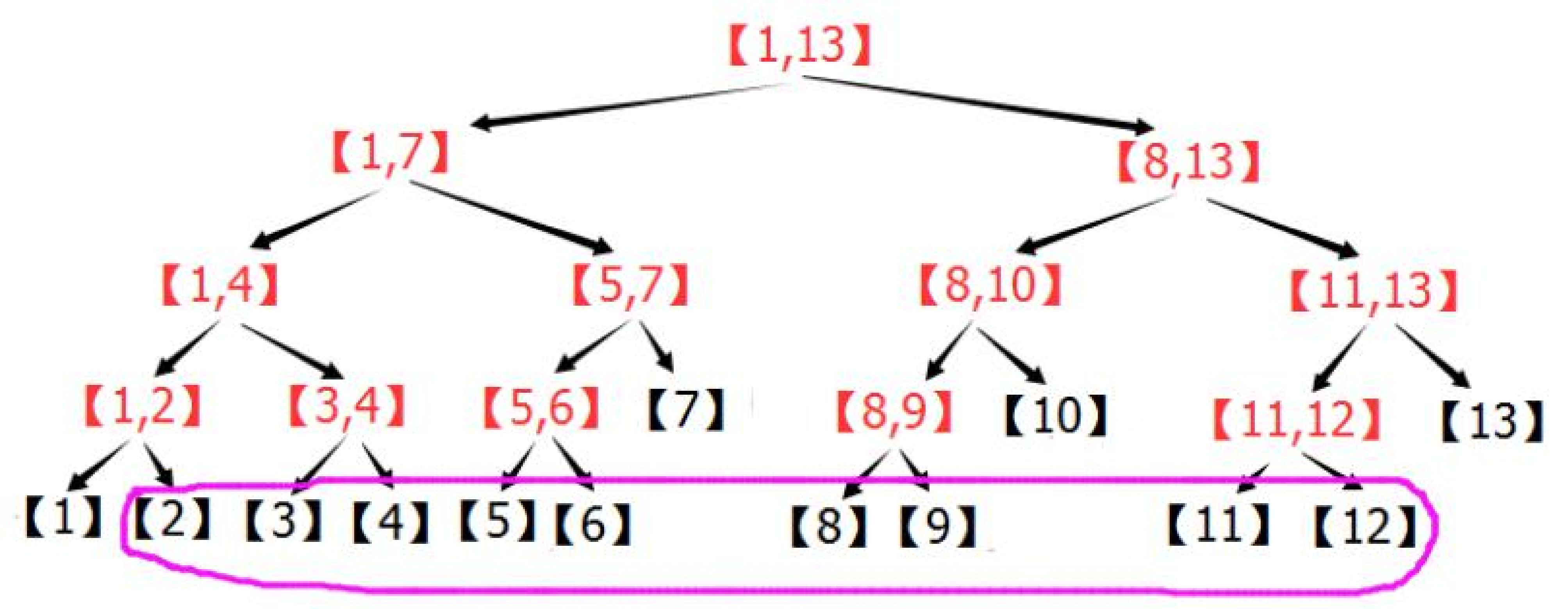
区间查询

- **定理：** n>=3时，一个[1, n]的线段树可以将[1, n]的任意子区间[L, R]分解为不超过$2\,log_2(n-1)$个子区间。

在查询[L, R]的统计值的时候，只需要访问不超过$2\log_2(n-1)$个结点，就可以获得[L, R]的统计信息，实现了区间查询效率为$O(log_2 n)$。

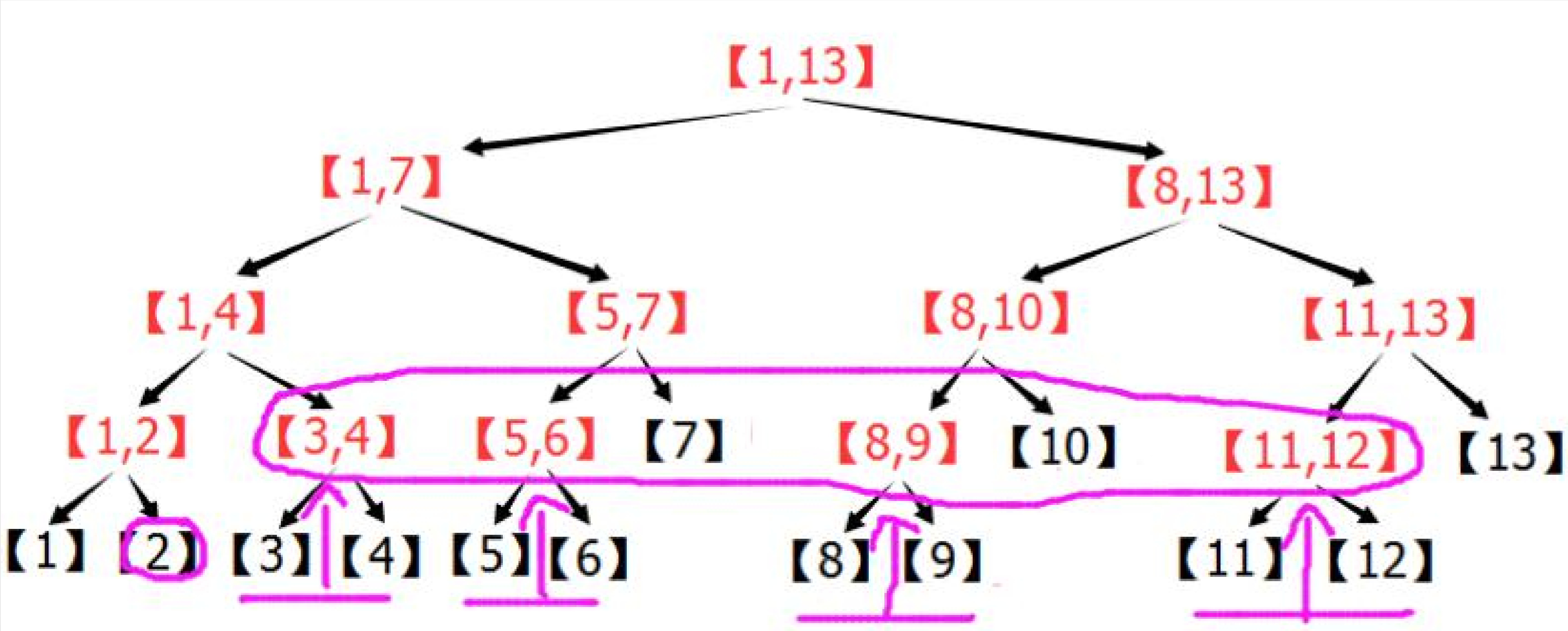
线段树

- 区间查询举例
  - n=13的线段树，查询区间[2, 12]



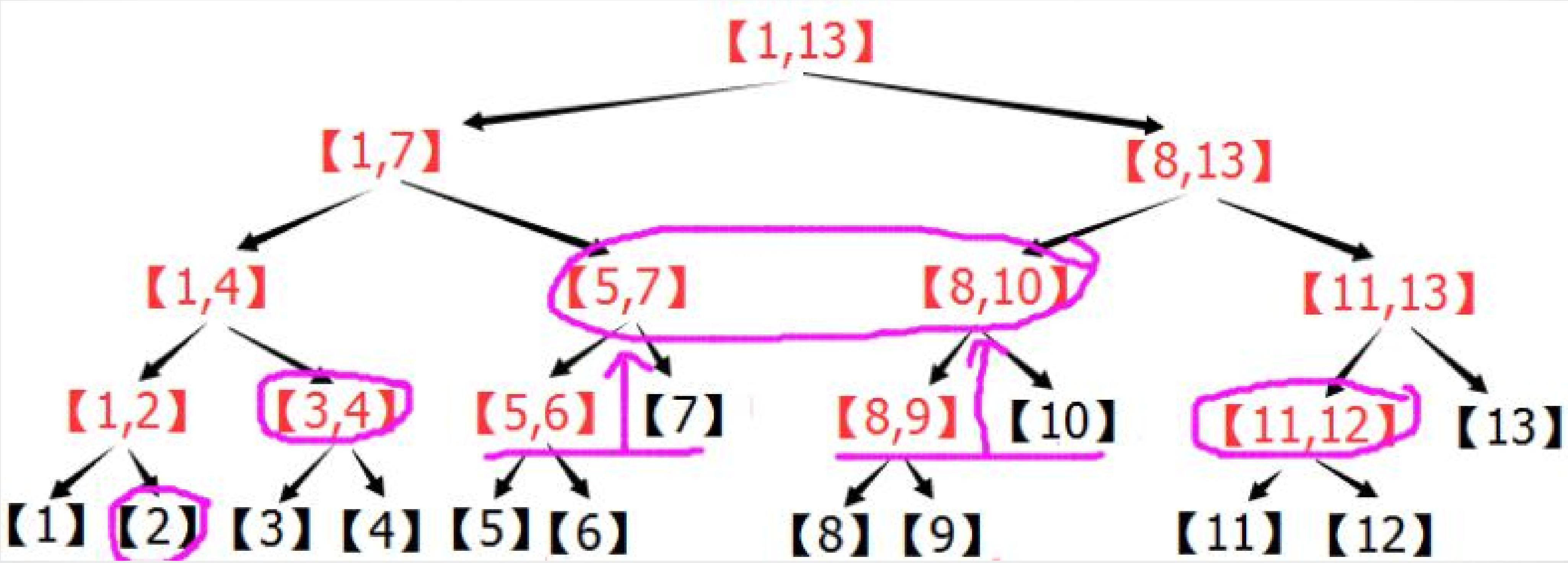
线段树

- 区间查询举例
  - n=13的线段树，查询区间[2, 12]



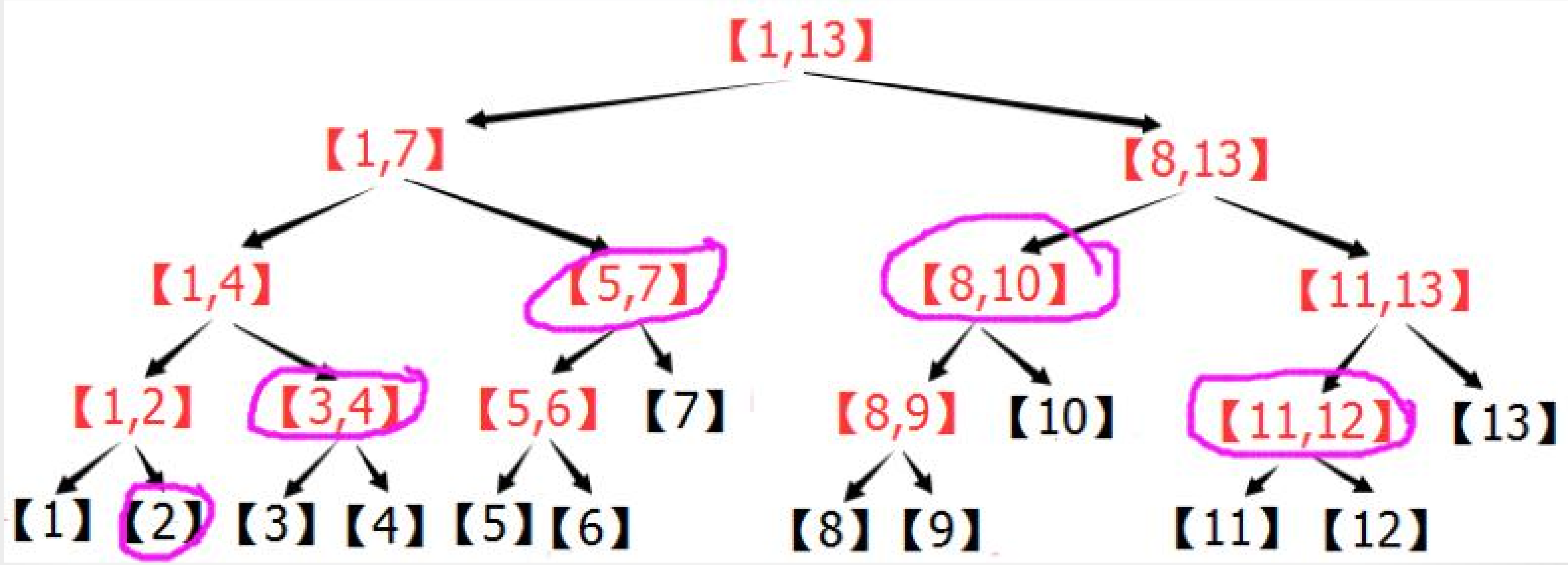
线段树

- 区间查询举例
  - n=13的线段树，查询区间[2, 12]



线段树

- 区间查询举例
  - n=13的线段树，查询区间[2, 12]



结果：由图可以看出，查询的有［2］、［3, 4］、［5, 7］、［8, 10］和［11, 12］共计5个区间。

线段树

- 线段树的区间查询类似与二分查找，通过递归把查询分解到左、右子区间，最终把结果合并处理。

```
//查询区间和的代码
int query(int k,int l,int r)
//当前到了编号为k的节点，查询[l..r]的和
{
    if(a[k].lazy)
    //如果当前节点被打上了懒惰标记，那么就把这个标记下传
        pushdown(k);
    if(a[k].l==l&&a[k].r==r)
    //如果当前区间就是询问区间，完全重合，那么显然可以直接返回
        return a[k].sum;
    int mid=(a[k].l+a[k].r)/2;
    if(r<=mid)  //如果询问区间包含在左子区间中
        return query(k*2,l,r);
    if(l>mid)   //如果询问区间包含在右子区间中
        return query(k*2+1,l,r);
    //如果询问区间跨越两个子区间
    return query(k*2,l,mid)+query(k*2+1,mid+1,r);
}
```

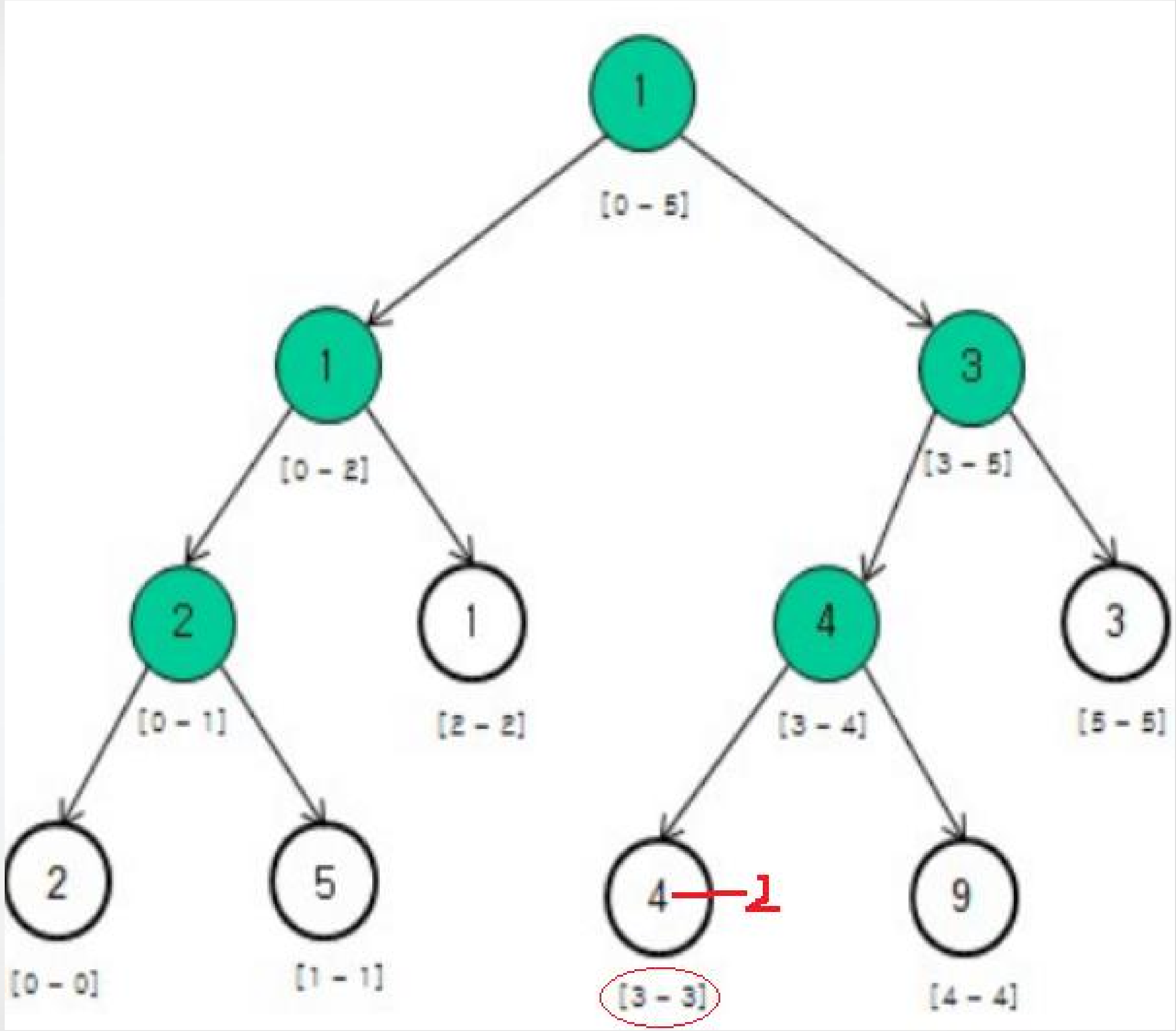
线段树

单结点更新

- 单结点更新是指更新某个叶子结点的值。

　　更新叶子结点会对其父结点的值产生影响，因此更新子结点后，要回溯更新其父结点的值。

- 时间复杂度O（$log_2 n$）



线段树

线段树的区间更新

- 区间更新是指更新某个区间内的所有叶子结点的值。

因为涉及到的叶子结点不止一个，而叶子结点会影响其相应的父结点，那么回溯需要更新的非叶子结点也会有很多，如果一次性更新完，效率可能不只是log_2n。

- **为此引入了线段树中的延迟标记（Lazy标记）**

线段树

线段树的区间更新

- 对于任意区间的修改，首先按照区间查询的方式将其对应成线段树中的结点，然后修改这些结点的信息，并给这些结点标记上代表这种修改操作的标记。
- 在修改和查询的时候，如果找到了一个结点p，并且决定考虑其子结点，那么就要看结点p是否被标记，如果有，就要按照标记修改其子结点的信息，并且给子结点都标上相同的标记，同时消掉结点p的标记。
- 每个结点增加一个标记—Lazy标记，用Lazy标记记录这个结点是否进行了某种修改（这种修改操作会影响其子结点）。

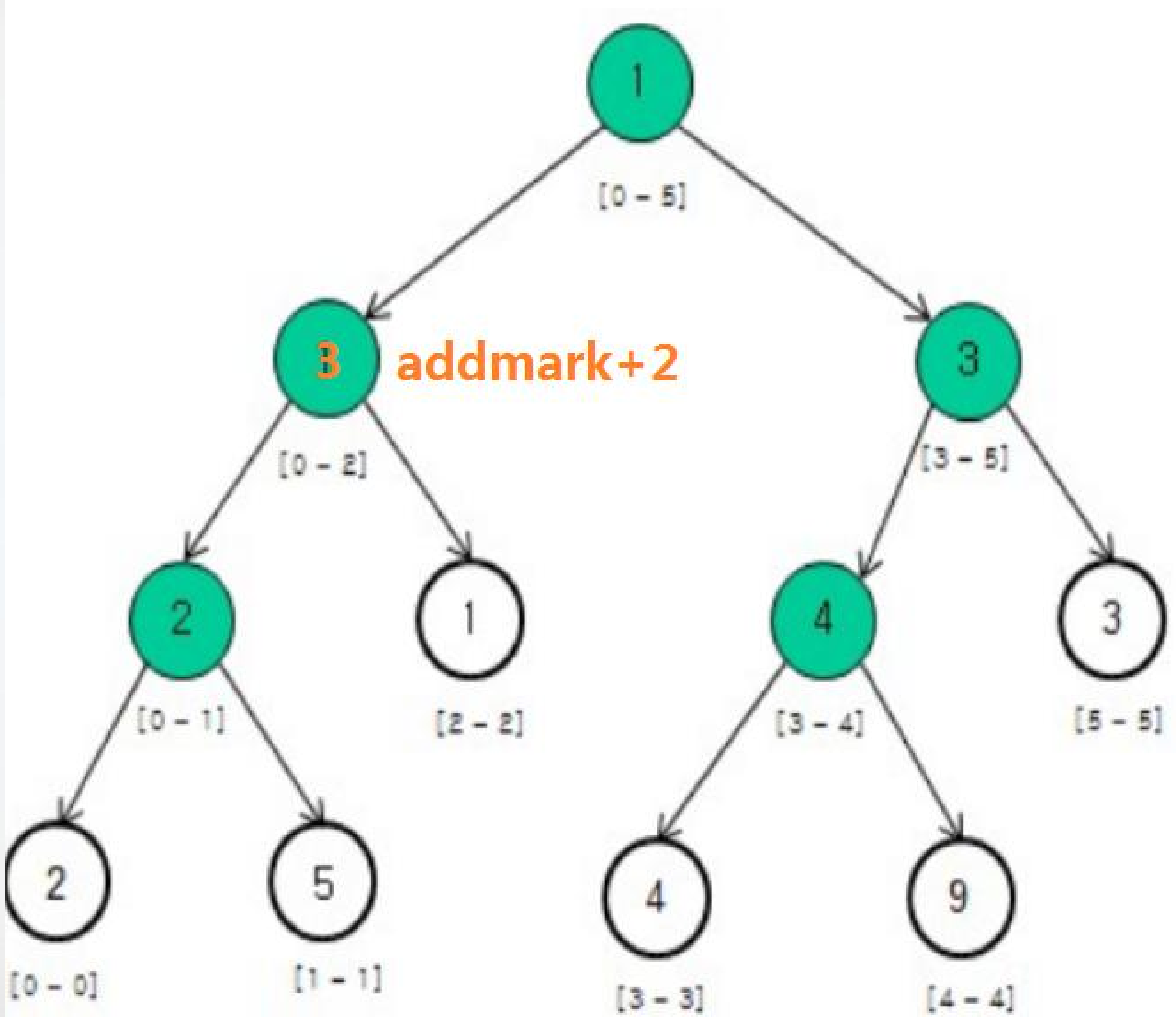
※ 核心思想：“自上而下”，先改上层的分支结点，下层直到叶子结点可以推迟到下次再有类似操作时再改。

线段树

线段树区间更新示例

要对区间[0,2]的叶子结点增加2

- 用区间查询的方法找到了分支结点[0-2]
- 把它的值设置为1+2 = 3
- 把它的addmark（即Lazy标记）设置为+2
- 更新完毕
- 它的以下子结点不必实时更新

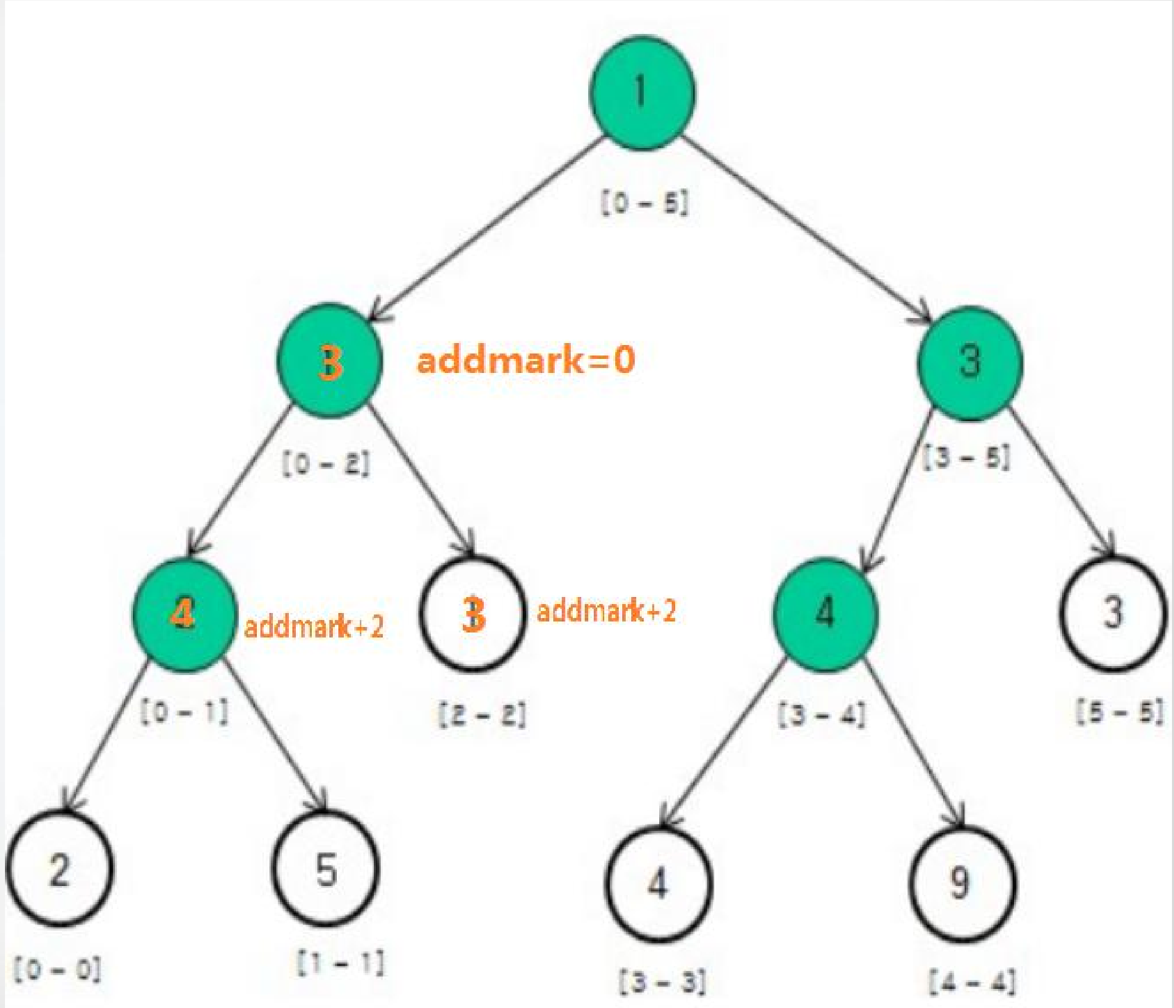


线段树

线段树区间更新示例

现在要查询[0, 1]

- 用区间查询的方法找到了分支结点[0-2]
- 发现它的Lazy标记不为0，并且还要继续向下搜索
- 把Lazy标记向下传递，自己的Lazy标记置0
- 结点[0-1]的值设置为2+2=4，Lazy标记设置为+2
- 结点[2-2]的值设置为1+2=3，Lazy标记设置为+2
- 然后返回查询结果
- 结点[0-1]和[2-2]的以下结点不必实时更新
- 叶子结点的Lazy标记是无意义的

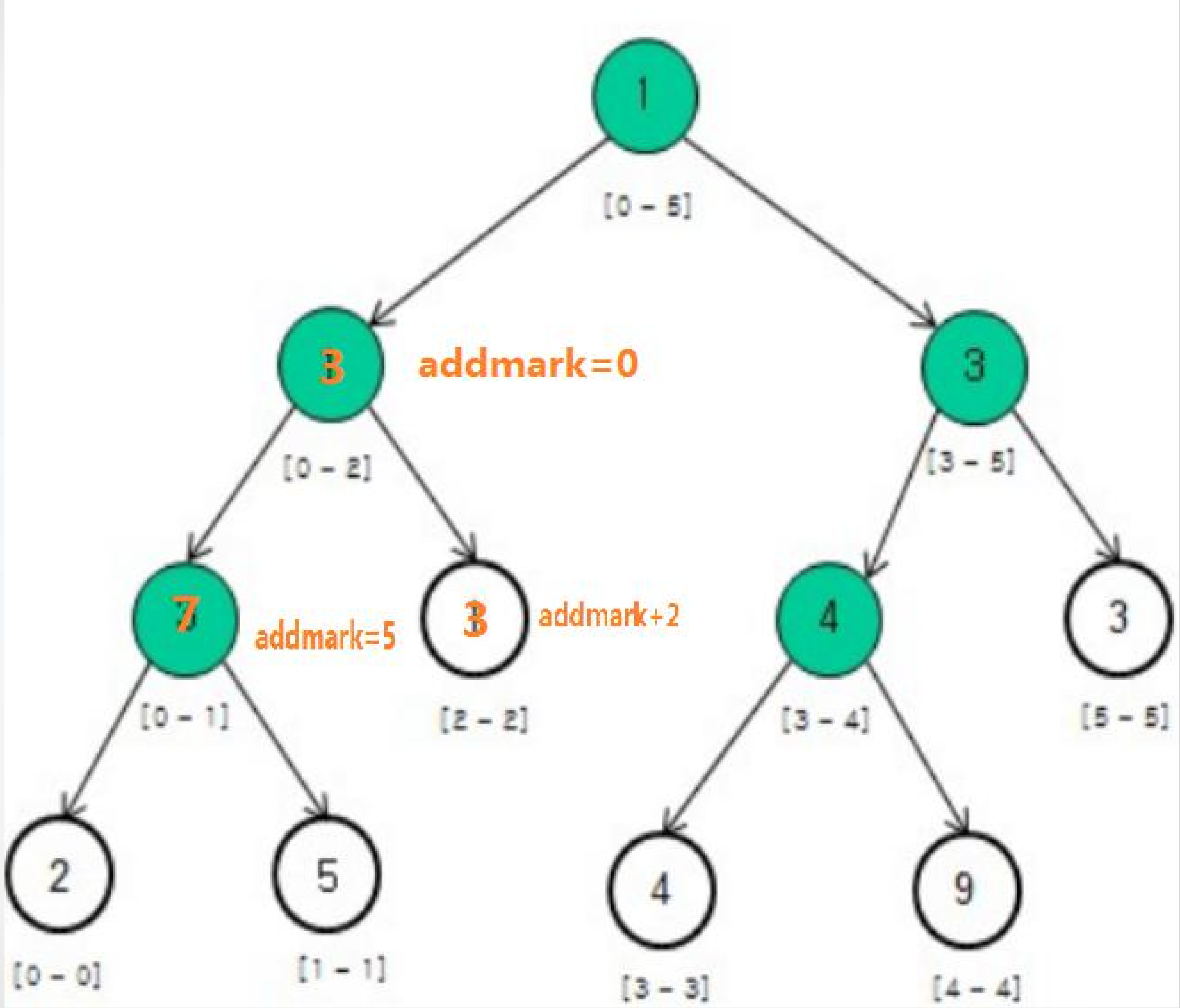


线段树

线段树区间更新示例

现在更新[0, 1]增加3

- 查询到结点[0–1],发现它的Lazy标记值为2
- 把Lazy标记值设置为2+3=5
- 结点的值设置为4+3=7
- 更新完毕
- 它的以下结点不必实时更新



线段树

- 线段树的Lazy标记程序，对应下推标记函数PushDown

```
void pushDown(int root)
{
    if(segTree[root].addMark != 0)
    {
        //设置左右孩子节点的标志域，因为孩子节点可能被多次延迟标记又没有向下传递
        //所以是 “+=”
        segTree[root*2+1].addMark += segTree[root].addMark;
        segTree[root*2+2].addMark += segTree[root].addMark;
        //根据标志域设置孩子节点的值。因为我们是求区间最小值，因此当区间内每个元
        //素加上一个值时，区间的最小值也加上这个值
        segTree[root*2+1].val += segTree[root].addMark;
        segTree[root*2+2].val += segTree[root].addMark;
        //传递后，当前节点标记域清空
        segTree[root].addMark = 0;
    }
}
```

线段树

- 线段树的Lazy标记程序——在做完Lazy标记后，做区间查询，基于最小值

```
int query(int root, int nstart, int nend, int qstart, int qend)
{
    //查询区间和当前节点区间没有交集
    if(qstart > nend || qend < nstart)
        return INFINITE;
    //当前节点区间包含在查询区间内
    if(qstart <= nstart && qend >= nend)
        return segTree[root].val;
    //分别从左右子树查询，返回两者查询结果的较小值
    pushDown(root); //----延迟标志域向下传递
    int mid = (nstart + nend) / 2;
    return min(query(root*2+1, nstart, mid, qstart, qend),
               query(root*2+2, mid + 1, nend, qstart, qend));

}
```

线段树

- 在做完Lazy标记后，做区间更新，基于最小值

```
void update(int root, int nstart, int nend, int ustart, int uend, int addVal)
{
    //更新区间和当前节点区间没有交集
    if(ustart > nend || uend < nstart)
        return ;
    //当前节点区间包含在更新区间内
    if(ustart <= nstart && uend >= nend)
    {
        segTree[root].addMark += addVal;
        segTree[root].val += addVal;
        return ;
    }
    pushDown(root); //延迟标记向下传递
    //更新左右孩子节点
    int mid = (nstart + nend) / 2;
    update(root*2+1, nstart, mid, ustart, uend, addVal);
    update(root*2+2, mid+1, nend, ustart, uend, addVal);
    //根据左右子树的值回溯更新当前节点的值
    segTree[root].val = min(segTree[root*2+1].val, segTree[root*2+2].val);
}
```

线段树

线段树与树状数组的区别：

1. 两者在复杂度上同级， 但是树状数组的编程简洁，而线段树还要维护其它数据域，编码复杂度较高。
2. 线段树可以完全涵盖树状数组的作用。凡是可以使用树状数组解决的问题， 使用线段树一定可以解决，但是线段树能够解决的问题树状数组未必能够解决。

线段树

- 线段树的经典案例

1、区间最值

2、区间求和

3、区间染色

4、区间K大数

5、矩形面积并

◆ 线段树的经典案例

3、区间染色

【例】给定一个长度为n(n <= 100000)的木板，支持两种操作：保证染色的颜色数少于30种。

1、P a b c　　将[a, b]区间段染色成c；

2、Q a b　　询问[a, b]区间内有多少种颜色；

这类染色问题是对区间的值进行替换（或者叫覆盖）。

线段树

◆ 线段树的经典案例

3、区间染色

提示：线段树的结点上要存储30种颜色的有无，如果每个结点都要存储30个bool值，太浪费空间，而且在计算合并操作的时候必须遍历30个元素，大大降低效率。考虑将30个bool值压缩在一个int32数据中，利用二进制压缩可以用一个32位的整型数完美的存储30种颜色的有无情况。

基本操作的几个函数和区间求和非常相似，不同的是回溯统计的时候，对于两个子结点的数据域不再是加和，而是位或和。

线段树

◆ 4、区间K大数

【例题】给定n(n <= 100000)个数的数组，然后m(m <= 100000)条询问，询问格式如下：

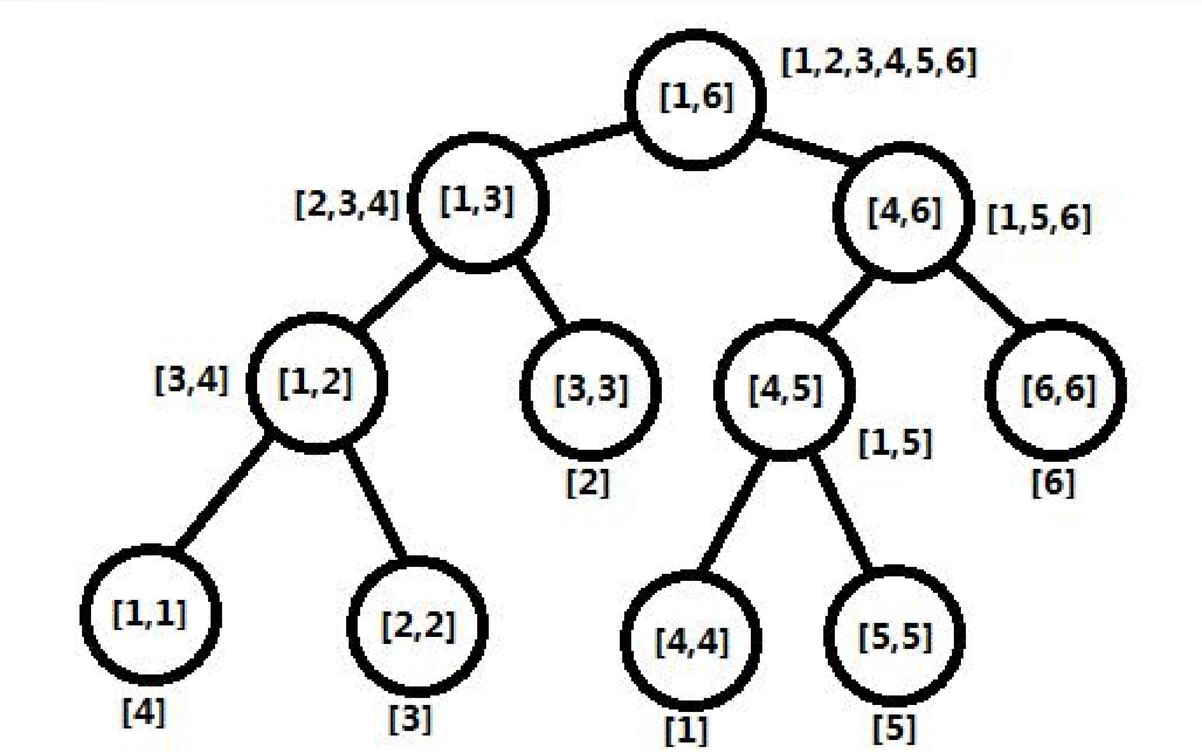
l r k 询问[l, r]的第K大的数的值

一个经典的面试题，利用了线段树划分区间的思想。

提示：线段树的每个结点存储的是这个区间内所有数组成的递增有序序列。建树过程是一个归并排序的过程，从叶子结点自底向上进行归并。

线段树

- 例如，对于数组[4, 3, 2, 1, 5, 6]，建立线段树。



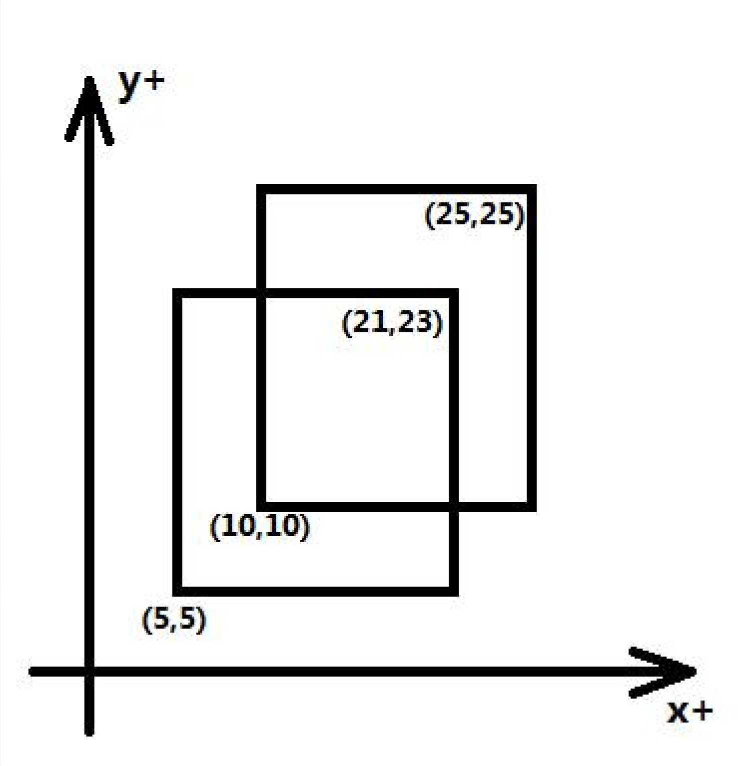
提示：在每次询问区间[a,b]时，将[a,b]划分成每一段都有序的小区间，之后就可以通过二分枚举答案T，通过二分查找在每一段小区间内统计小于等于当前答案T的数的个数count，和K进行比较，从而确定T是否是第K大的，最后得到答案。

线段树

◆ 线段树的经典案例

5、矩形面积并

【例】给定n(n <= 100000)个平行于X、Y轴的矩形，求它们的面积并。



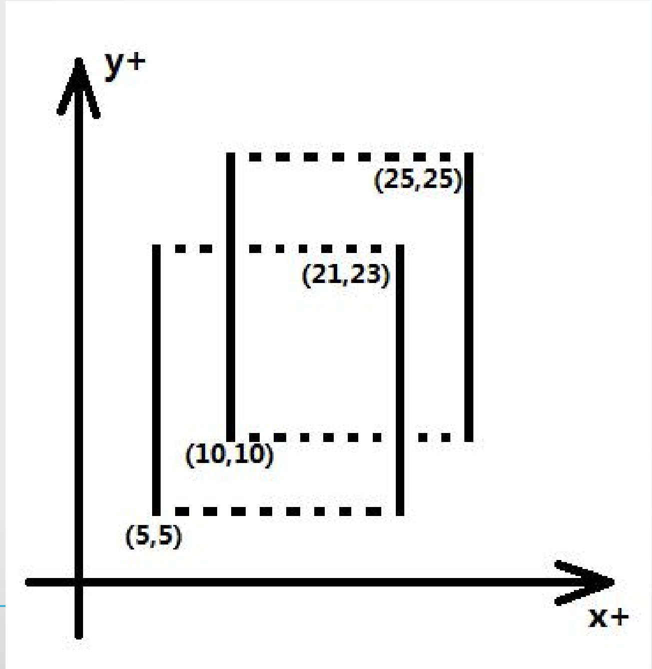
提示：用线段树求解，核心思想是降维，将某一维套用线段树，另外一维则用来枚举。

首先将每个矩形的纵向边投影到Y轴上，这样可以把矩形的纵向边看成一个闭区间，用线段树来维护这些矩形边的并。现在求的是矩形的面积并，于是可以枚举矩形的x坐标，然后检测当前相邻x坐标上y方向的合法长度，两者相乘就是其中一块面积，枚举完毕后就求得了所有矩形的面积并。

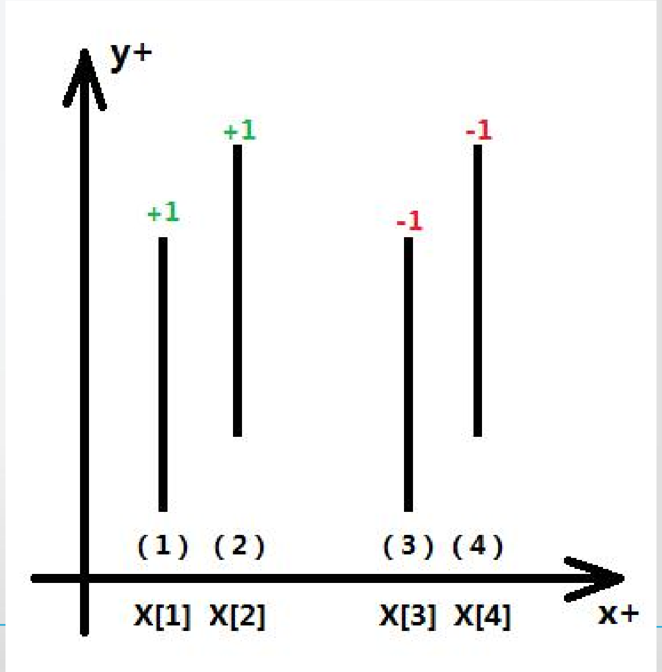
5、矩形面积并

提示：

第一步：将一个矩形分成两条垂直于x轴的线段来存储。



第二步：x坐标较小的为入边，x坐标较大的为出边。入边权值为+1，出边权值为-1，并将所有的线段按照x坐标递增排序，第i条线段的x坐标记为X[i]。

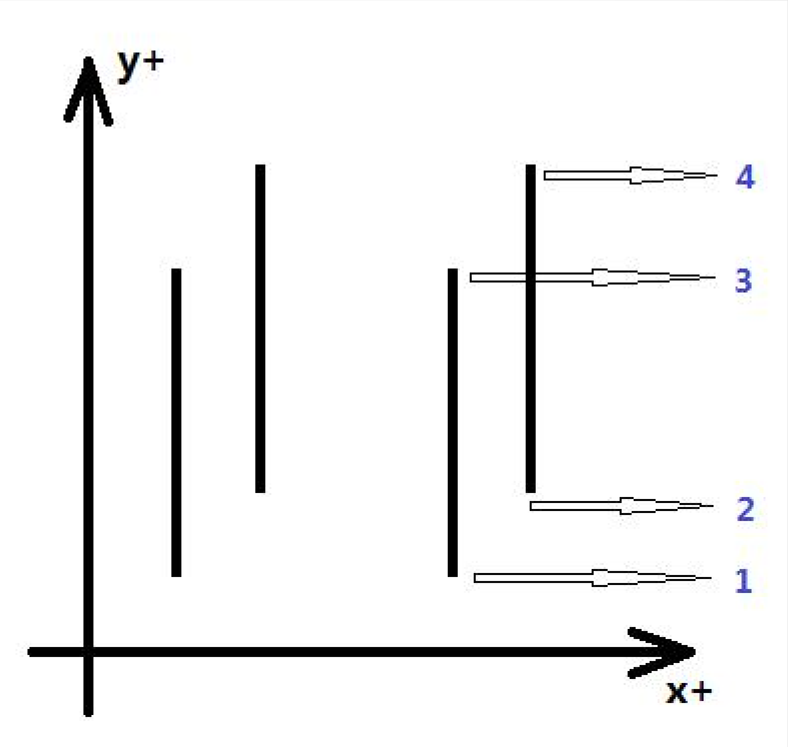


线段树

5、矩形面积并

第三步：由于矩形端点的y坐标有可能很大而且不一定是整数，所以先将所有y坐标进行重映射(也叫离散化)，将原坐标映射成小范围的整数可以作为数组下标，更方便计算，映射可以将所有y坐标进行排序去重，然后二分查找确定映射后的值。

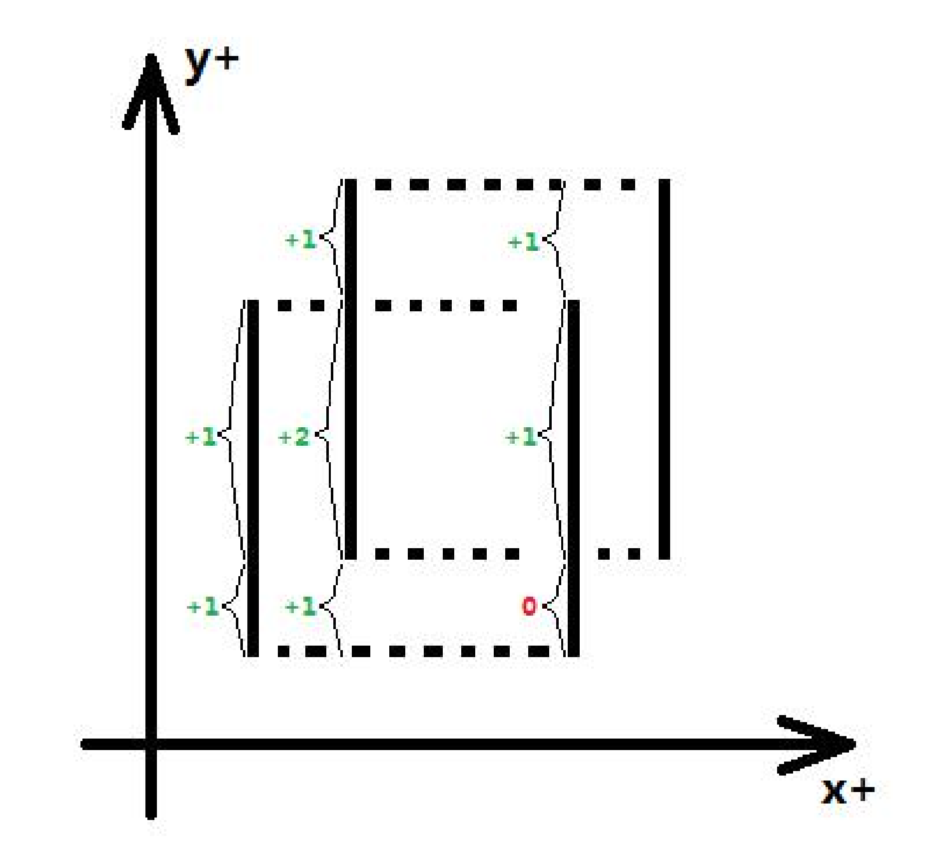
离散化的具体步骤：蓝色数字表示的是离散后的坐标，即1、2、3、4分别对应原先的5、10、23、25。假设离散后的y方向的坐标个数为m，则y方向被分割成m-1个独立单元，即“单位线段”，分别记为<1-2>、<2-3>、<3-4>。



5、矩形面积并

第四步：以x坐标递增的方式枚举每条垂直线段，y方向用一个长度为m-1的数组来维护“单位线段”的权值。右图展示了每条线段按x递增方式插入之后每个“单位线段”的权值。

当枚举到第i条线段时，检查所有“单位线段”的权值，所有权值大于零的“单位线段”的实际长度之和(离散化前的长度)被称为“合法长度”，记为L，那么(X[i] - X[i-1]) * L，就是第i条线段和第i-1条线段之间的矩形面积和，计算完第i条垂直线段后将它插入。所谓"插入"就是利用该线段的权值更新（即累加）该线段对应的“单位线段”的权值和。



线段树

5、矩形面积并

右图中，红色、黄色、蓝色三个矩形分别是3对相邻线段间的矩形面积和，其中红色部分的y方向由<1-2>、<2-3>两个“单位线段”组成，黄色部分的y方向由<1-2>、<2-3>、<3-4>三个“单位线段”组成，蓝色部分的y方向由<2-3>、<3-4>两个“单位线段”组成。

特殊的，在计算蓝色部分的时候，<1-2>部分的权值由于第3条线段的插入(第3条线段权值为-1)而变为零，所以不能计入“合法长度”。

以上所有相邻线段之间的面积和就是最后要求的矩形面积并。