



数据结构和算法实习

郭 炜

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄



北京大学
PEKING UNIVERSITY

信息科学技术学院

线段树



北京大学
PEKING UNIVERSITY

信息科学技术学院

线段树的概念



云岗石窟

我们是谁？



线段树!



我们任务是什么？



区间更新，区间查询!



我们有多快？



$\text{Log}(N)$!



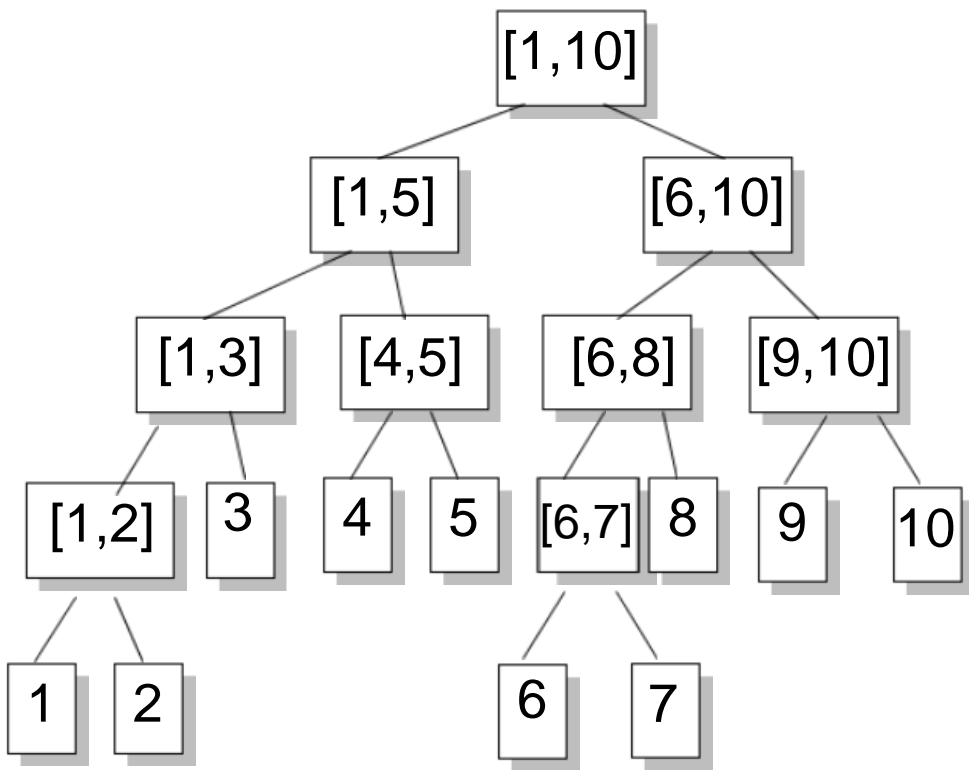
线段树(Interval Tree)的定义

- 实际上还是称为区间树更准确和易于理解
- 是一棵二叉树
- 树上的每个节点对应于一个区间 $[a,b]$ (也称线段) , a,b 通常为整数

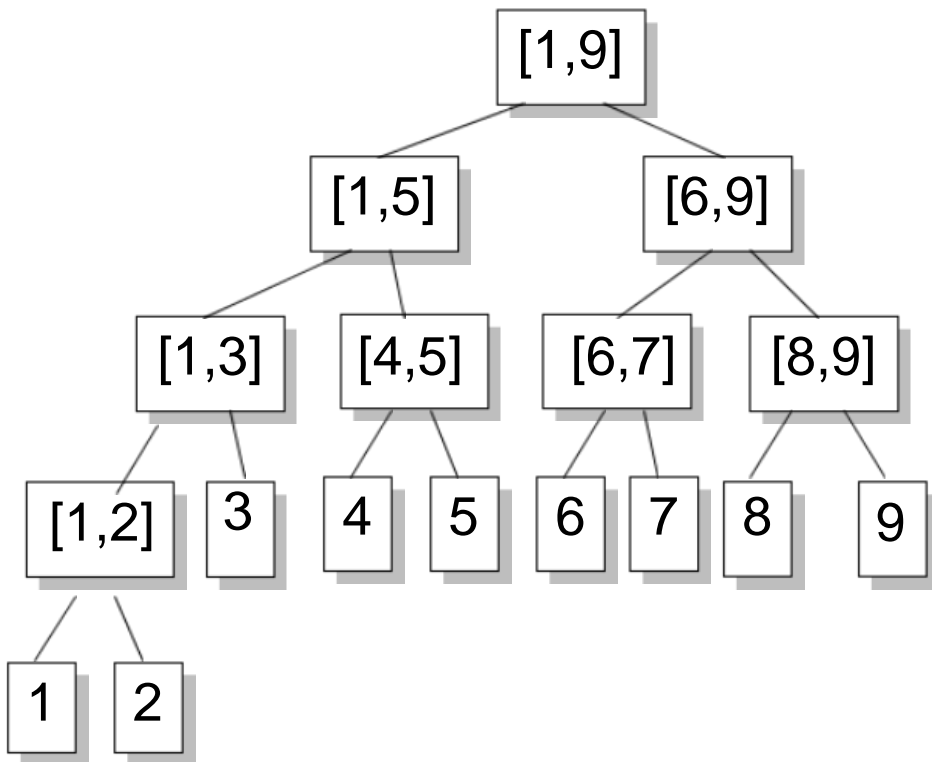
线段树(Interval Tree)的定义

- 同一层的节点所代表的区间，相互不会重叠
同一层节点所代表的区间，加起来是个连续的区间
- 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 $[(a+b)/2+1, b]$ (除法去尾取整)
- 叶子节点表示的区间长度为 1

区间[1, 10]对应的线段树



区间[1, 9]对应的线段树



线段树的性质

- 用二分的方法构造线段树。若根节点对应的区间是 $[a,b]$ ，则最多二分 $\log_2(b-a+1)$ 次(向上取整)就会分到叶子节点，不能再分。因此线段树深度为： $\log_2(b-a+1) + 1$ (向上取整)
- 叶子节点的数目和根节点表示区间的长度相同
- 线段树节点要么0度，要么2度。因此若叶子节点数目为 N ，则线段树总结点数目为 $2N-1$ 。
- 线段树除了最底下一层，其它层节点都是满的。



北京大学
PEKING UNIVERSITY

信息科学技术学院

线段树的操作



敦煌鸣沙山月牙泉

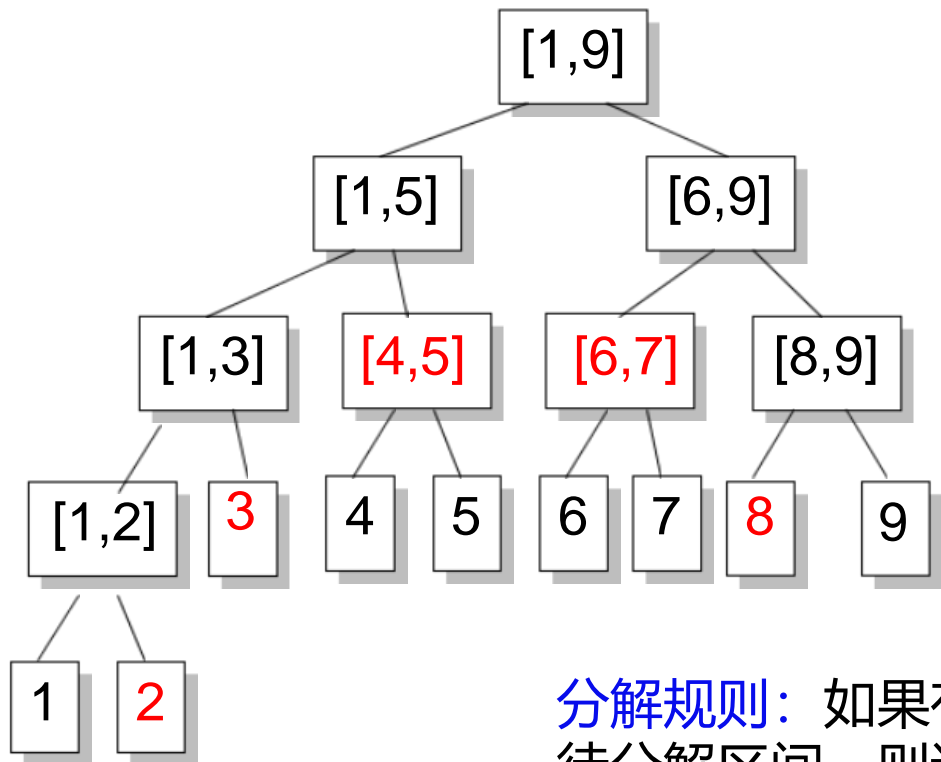
线段树的核心操作：区间分解

- 若线段树根节点对应区间为 $[a,b]$ ，给定区间 $[c,d]$ ($a \leq c, d \leq b$)，区间分解，就是找出一些节点，这些节点对应的区间互不重叠，且加起来正好是 $[c,d]$
- 这些节点称为“终止节点”

线段树的核心操作：区间分解

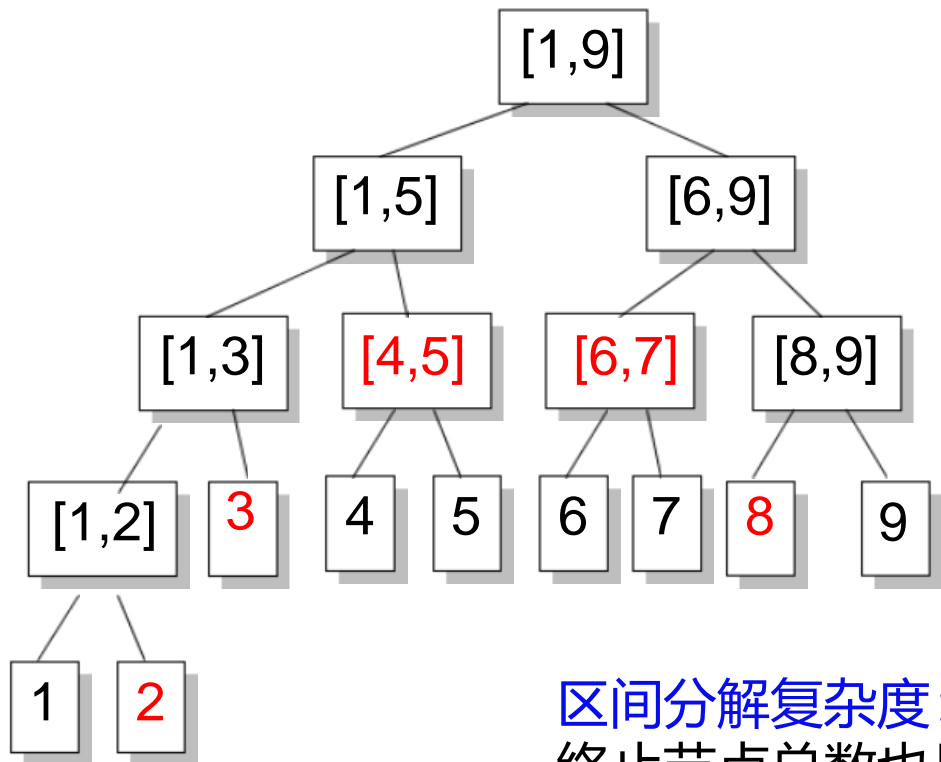
- 从根节点开始，递归进行区间分解
- 走到节点 $[L,R]$ 时，如果要分解的的区间就是 $[L,R]$ ，则找到一个终止节点
如果不是，则：取 $\text{mid} = (L+R) / 2$ ；
看要分解的区间与 $[L,\text{mid}]$ 或 $[\text{mid}+1,R]$ 哪个有交集，就进入哪个区间进行进一步分解。有可能两个区间都进入

[1, 9]的线段树上，区间[2,8]的分解



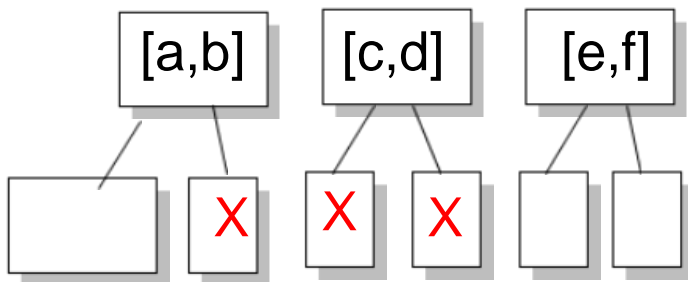
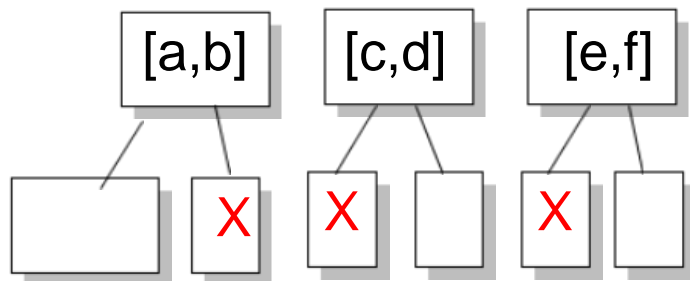
分解规则：如果有某个节点代表的区间，完全属于待分解区间，则该节点为“终止”节点，不再继续往下分解

[1, 9]的线段树上，区间[2,8]的分解



区间分解复杂度：每层最多2个“终止节点”，所以终止节点总数也是 $\log_2(n)$ 量级的，即区间分解复杂度是 $\log(n)$ n 为根区间长度

证明每层最多2个“终止节点”：



红X代表终止节点。上述情况不可能发生

线段树的特性总结

- 线段树的深度为 $\log_2(n)+1$ （向上取整， n 是根节点对应区间的长度）
- 线段树上，任意一个区间被分解后得到的“终止节点”数目都是 $\log(n)$ 量级
- 线段树上更新叶子节点和进行区间分解时间复杂度都是 $O(\log(n))$ 的
- 线段树能在 $O(\log(n))$ 的时间内完成以区间为单位的数据更新和查询，能用树状数组做的题，都能用线段树做

线段树的构建

function 以节点 v 为根建树、 v 对应区间为 $[L,R]$

```
{  
    对节点 $v$ 初始化  
    if ( $L \neq R$ )  
    {  
        以 $v$ 的左孩子为根建树、区间为 $[L, (L+R)/2]$   
        以 $v$ 的右孩子为根建树、区间为 $[(L+R)/2+1, R]$   
    }  
}
```

建树的时间复杂度是 $O(n)$ n 为根节点对应的区间长度

线段树的区间分解

function 从节点V开始，分解区间 [L,R]

```
{ //V代表的区间是 [V.L,V.R]
```

```
  if(!( L == V.L && R == V.R)) { //V非终止节点
```

```
    if( R <= mid ) //mid是[V.L,V.R]中点
```

```
      从节点V左孩子开始，分解区间 [L,R]    //路径1
```

```
    else if( L > mid )
```

```
      从节点V右孩子开始，分解区间 [L,R]    //路径2
```

```
    else { //路径3
```

```
      从节点V左孩子开始，分解区间 [L,mid]    //分支3-1
```

```
      从节点V右孩子开始，分解区间 [mid+1,R]  //分支3-2
```

```
  }
```

```
}
```

```
}
```

区间分解从根节点开始。递归时，若走一次路径3，则以后每次都只走路径1或路径2，或即便走路径3，其中的一个分支也是往下一层立即碰到终止节点而返回，即等价于走路径1或路径2，故 $O(\log(n))$ 次递归即到底，区间分解复杂度为 $O(\log(n))$



北京大学
PEKING UNIVERSITY

信息科学技术学院

线段树解题关键



敦煌鸣沙山

线段树解题

- 对区间所对应的一些数据进行查询，操作是区间分解
- 对区间所对应的一些数据进行更新，操作也是区间分解
- 关键是要想清楚每个节点要存哪些信息，以及这些信息如何高效更新，维护，查询。不要一更新就更新到叶子节点，那样更新效率最坏就可能变成 $O(n)$ 的了

线段树解题

- 确定根节点对应的区间，建树
- 插入数据（或建树同时就初始化数据）
- 更新，查询
- 树节点结构：

```
struct CNode
{
    int L,R; //区间起点和终点
    XXXX ;   //与区间[L,R]相关的数据
    CNode * pLeft, * pRight; //左右孩子指针
};
```

用一维数组存放线段树

➤ 如果用一维数组存放线段树，且根节点区间 $[1, n]$

- 使用左右节点指针，则数组需要有 $2n-1$ 个元素
- 不要左右节点指针。令根节点下标为0，对下标为 i 的节点：

左子节点下标为 $i*2+1$,
右子节点下标为 $i*2+2$

则数组需要有： $2*2^{\lceil \log_2 n \rceil} - 1$ 个元素 ($\lceil \log_2 n \rceil$ 向上取整)

$2*2^{\lceil \log_2 n \rceil} - 1 \leq 4n - 1$ ，数组开 $4n$ 可以确保不越界



北京大学
PEKING UNIVERSITY

信息科学技术学院

线段树例题1



安徽高铁随拍

例题1：单点更新，区间求和

- 给一个数的序列 $A_1A_2\ldots A_n$ 。并且可能多次进行下列两个操作：
 - 1、对序列里面的某个数进行加减
 - 2、询问这个序列里面任意一个连续的子序列 $A_iA_{i+1}\ldots A_j$ 的和是多少。
- 希望第2个操作每次能在 $\log(n)$ 时间内完成

例题1：单点更新，区间求和

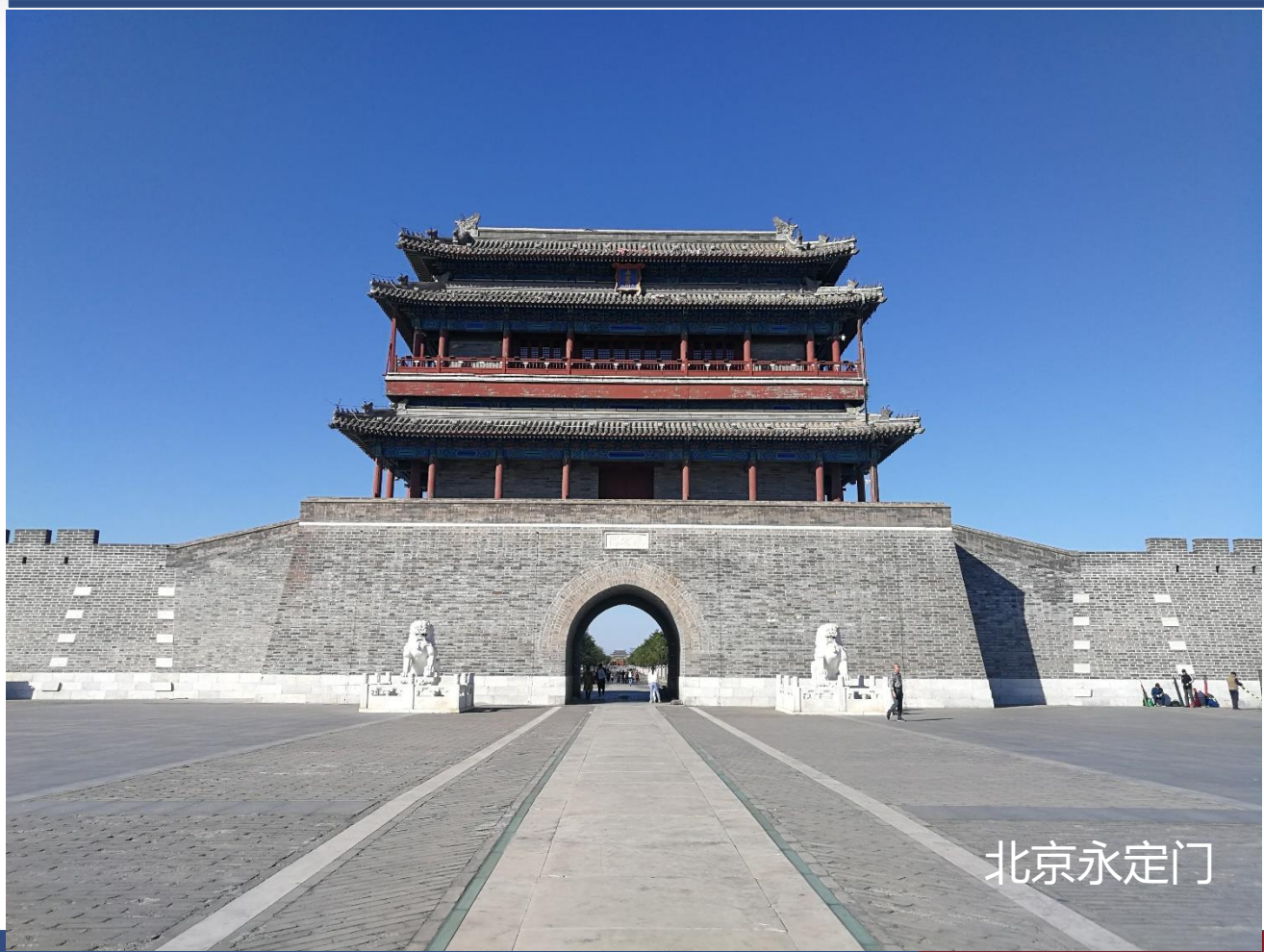
- 构建线段树，根节点对应区间 $[1, n]$
- 在每个节点记录该节点对应的区间的数的和Sum
- 操作1：相当于对区间 $[i, i]$ 进行区间分解，更新因 A_i 只会被一个叶子节点和该叶子节点的所有祖先覆盖， A_i 更新，则只要对覆盖 A_i 的节点的Sum进行修改，因此复杂度是 $\log(n)$ （线段树为 $\log(n)$ 层）
- 操作2：将被查询区间，分解成若干个“终止”节点，然后把这些“终止”节点的Sum累加起来。区间分解复杂度 $\log(n)$ ，故这一步的复杂度也是 $\log(n)$



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题2 Balanced Lineup



北京永定门

例题2: POJ 3264 Balanced Lineup

给定 Q ($1 \leq Q \leq 50,000$)个数 $A_1, A_2 \dots A_Q$, 多次求任一区间 $A_i - A_j$ 中最大数和最小数的差。

本题树节点结构是什么?

例题2: POJ 3264 Balanced Lineup

给定 Q ($1 \leq Q \leq 50,000$)个数 $A_1, A_2 \dots A_Q$, 多次求任一区间 $A_i - A_j$ 中最大数和最小数的差。

本题树节点结构是什么?

```
struct CNode
{
    int L,R; //区间起点和终点
    int minV,maxV; //本区间里的最大最小值
    CNode * pLeft, * pRight;
};
```

例题2: POJ 3264 Balanced Lineup

Sample Input

6 3 //6个数, 3个查询

1

7

3

4

2

5

1 5

4 6

2 2

Sample Output

6

3

0

例题2: POJ 3264 Balanced Lineup

```
#include <iostream>
using namespace std;
const int INF = 0xffffffff0;
int minV = INF;
int maxV = -INF;
struct Node //不要左右子节点指针的做法
{
    int L, R;
    int minV, maxV;
    int Mid() {
        return (L+R)/2;
    }
};
Node tree[200010]; //4倍叶子节点的数量就够
```

```
void BuildTree(int root , int L, int R)
{
    tree[root].L = L;
    tree[root].R = R;
    tree[root].minV = INF;
    tree[root].maxV = - INF;
    if( L != R ) {
        BuildTree(2*root+1,L, (L+R)/2) ;
        BuildTree(2*root+2, (L+R)/2 + 1, R) ;
    }
}
```

```
void Insert(int root, int i,int v)
//将第i个数, 其值为v, 插入线段树
{
    if( tree[root].L == tree[root].R ) {
        //成立则亦有 tree[root].R == i
        tree[root].minV = tree[root].maxV = v;
        return;
    }
    tree[root].minV = min(tree[root].minV,v);
    tree[root].maxV = max(tree[root].maxV,v);
    if( i <= tree[root].Mid() )
        Insert(2*root+1,i,v);
    else
        Insert(2*root+2,i,v);
}
```



```
void Query(int root,int s,int e)  {  
    //查询区间[s,e]中的最小值和最大值, 如果更优就记在全局变量里  
    //minV和maxV里  
    if( tree[root].minV >= minV && tree[root].maxV <= maxV )  
        return;  
    if( tree[root].L == s && tree[root].R == e ) {  
        minV = min(minV,tree[root].minV);  
        maxV = max(maxV,tree[root].maxV);  
        return ;  
    }  
    if( e <= tree[root].Mid())  
        Query(2*root+1,s,e);  
    else if( s > tree[root].Mid() )  
        Query(2*root+2,s,e);  
    else {  
        Query(2*root+1,s,tree[root].Mid());  
        Query(2*root+2,tree[root].Mid()+1,e);  
    }  
}
```

```
int main()
{
    int n,q,h;
    int i,j,k;
    scanf("%d%d",&n,&q);
    BuildTree(0,1,n);
    for( i = 1;i <= n;i ++ ) {
        scanf("%d",&h);
        Insert(0,i,h);
    }
    for( i = 0;i < q;i ++ ) {
        int s,e;
        scanf("%d%d", &s,&e);
        minV = INF;
        maxV = -INF;
        Query(0,s,e);
        printf("%d\n",maxV - minV);
    }
    return 0;
}
```



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题3

A Simple Problem with Integers



挪威冰川

例题3: POJ 3468 A Simple Problem with Integers

给定 Q ($1 \leq Q \leq 100,000$)个数 $A_1, A_2 \dots A_Q$,
以及可能多次进行的两个操作:

- 1) 对某个区间 $A_i \dots A_j$ 的每个数都加 n (n 可变)
- 2) 求某个区间 $A_i \dots A_j$ 的数的和

本题树节点要存哪些信息? 只存该区间的数的和, 行不行?

例题3: POJ 3468 A Simple Problem with Integers

只存和, 会导致每次加数的时候都要更新到叶子节点, 速度太慢($O(n\log n)$), 这是必须要避免的。

本题树节点结构:

```
struct CNode
{
    int L ,R;
    CNode * pLeft, * pRight;
    long long sum; //原来的和
    long long inc; //增量c的累加
}; //本节点区间的和实际上是nSum+Inc* (R-L+1)
```

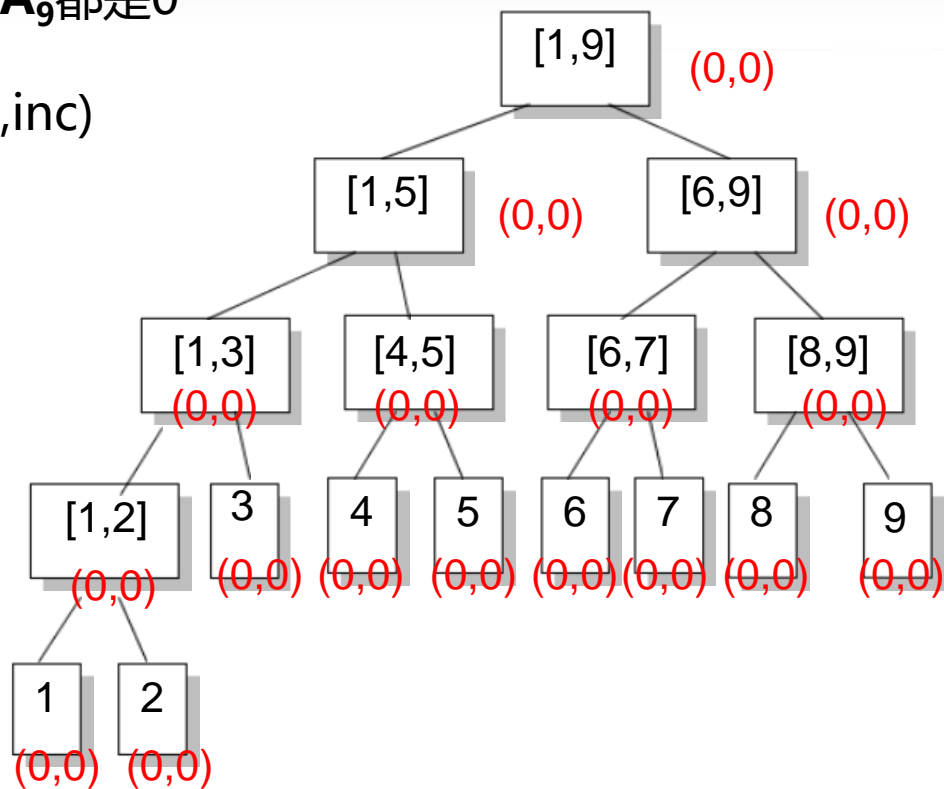
例题3: POJ 3468 A Simple Problem with Integers

在区间增加时, 如果要加的区间正好覆盖一个节点, 则增加其节点的inc值, 不再往下走, 否则要更新sum(加上本次增量), 再将增量往下传。这样区间增加的复杂度就是 $O(\log(n))$

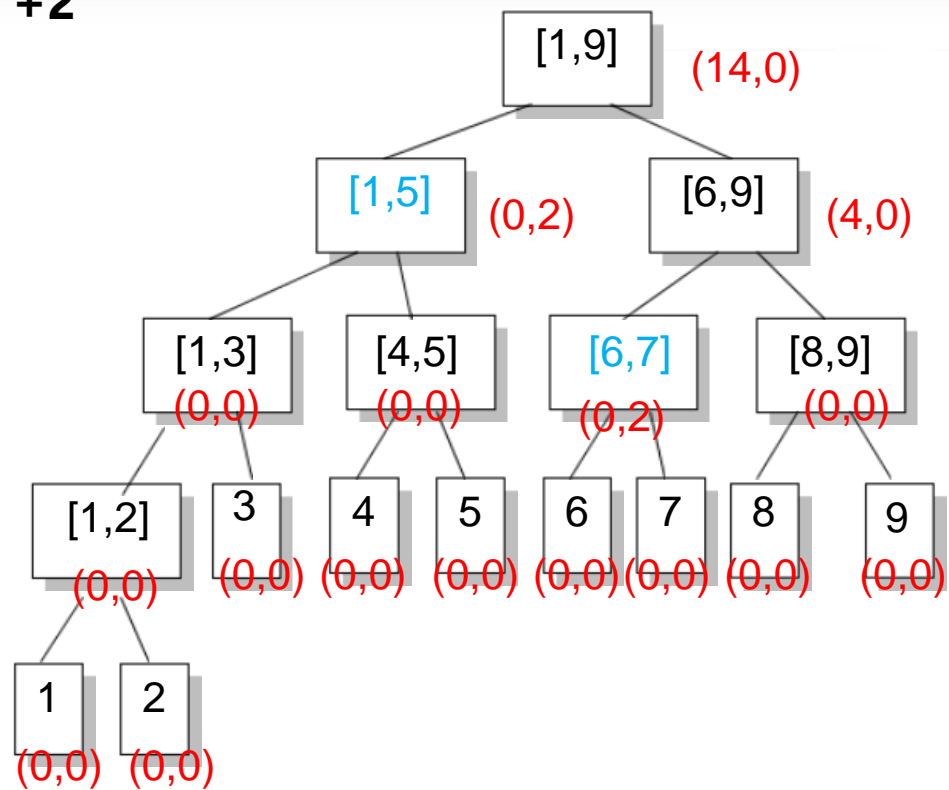
在查询时, 如果待查区间不是正好覆盖一个节点, 就将节点的inc往下带到下一层, 然后将inc代表的所有增量累加到sum上后将inc清0, 接下来再往下查询。

假设开始 $A_1 \dots A_9$ 都是0

括号内是 (sum,inc)

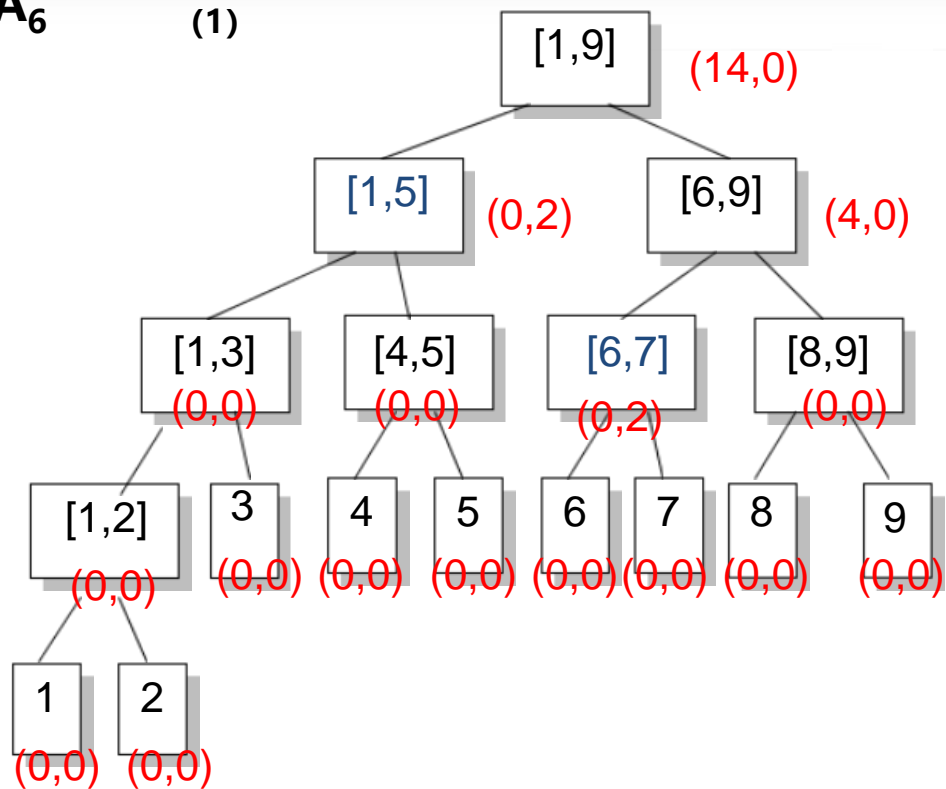


$A_1 \dots A_7$ +2



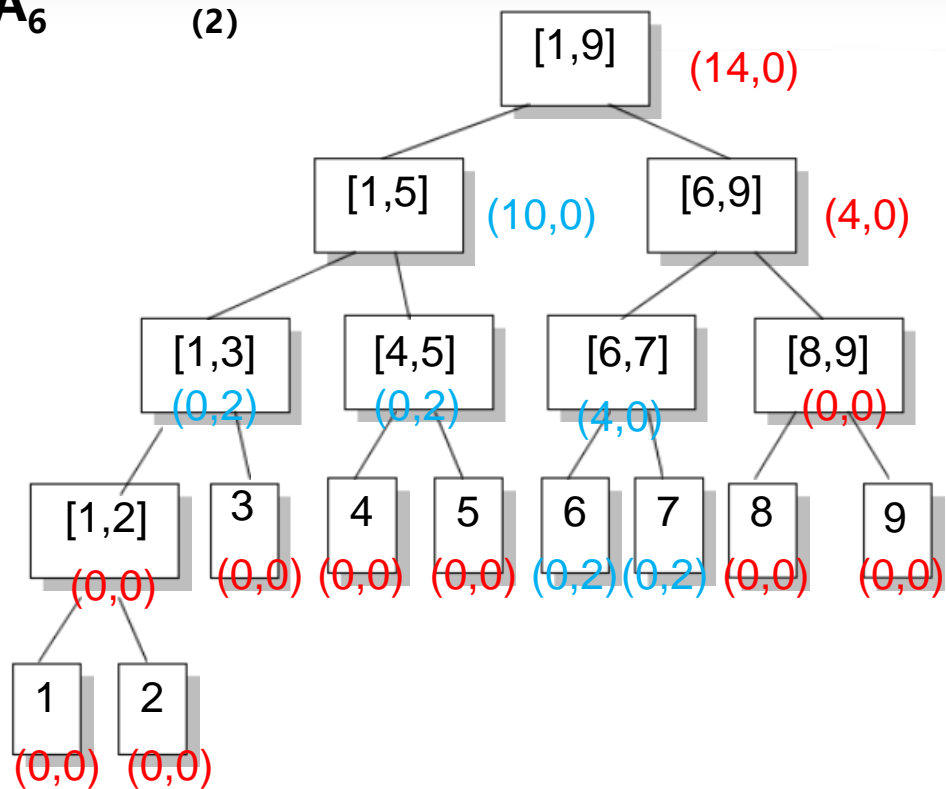
查 $A_2 + \dots + A_6$

(1)



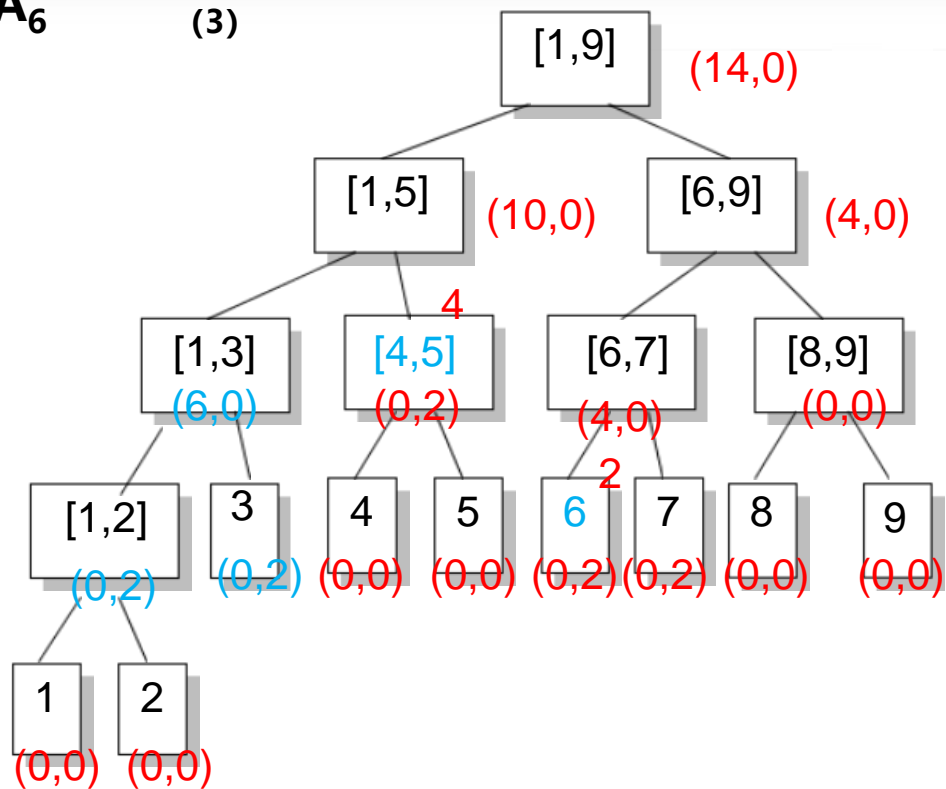
查 $A_2 + \dots + A_6$

(2)



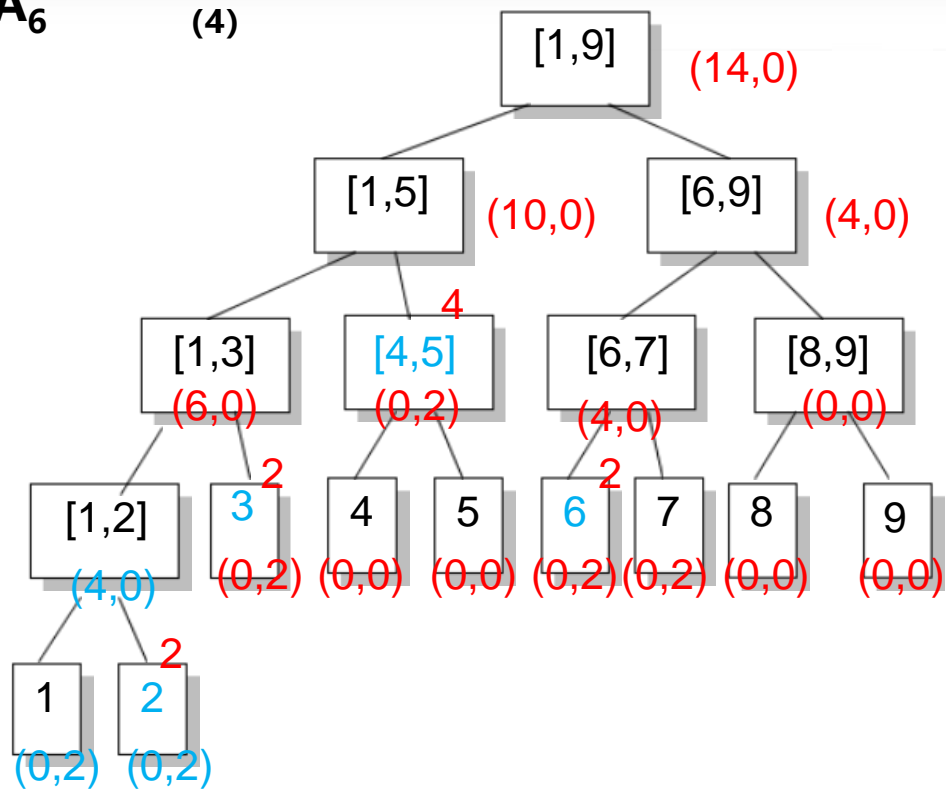
查 $A_2 + \dots + A_6$

(3)



查 $A_2 + \dots + A_6$

(4)



例题3: POJ 3468 A Simple Problem with Integers

延迟更新: 信息更新时, 未必要真的做彻底的更新, 可以只是将应该如何更新记录下来, 等到真正需要查询准确信息时, 才正真去更新足以应付查询的部分。

例题3: POJ 3468 A Simple Problem with Integers

```
#include <iostream>
using namespace std;
struct CNode
{
    int L ,R;
    CNode * pLeft, * pRight;
    long long sum; //原来的和
    long long inc; //增量c的累加
    int Mid(){
        return (L + R)/2;
    }
};
CNode Tree[200010]; // 2倍叶子节点数目就够
int count = 0;
```

```
void BuildTree(CNode * pRoot,int L, int R)
{
    pRoot->L = L;
    pRoot->R = R;
    pRoot->sum = 0;
    pRoot->inc = 0;
    if( L == R)
        return;
    count ++;
    pRoot->pLeft = Tree + count;
    count ++;
    pRoot->pRight = Tree + count;
    BuildTree(pRoot->pLeft,L, (L+R)/2) ;
    BuildTree(pRoot->pRight, (L+R)/2+1,R) ;
}
```

```
void Insert( CNode * pRoot,int i, int v)
{
    if( pRoot->L == i && pRoot->R == i) {
        pRoot->sum = v;
        return ;
    }
    pRoot->sum += v;
    if( i <= pRoot->Mid())
        Insert(pRoot->pLeft,i,v) ;
    else
        Insert(pRoot->pRight,i,v) ;
}
```



```
void Add( CNode * pRoot, int a, int b, long long c)
{
    if( pRoot->L == a && pRoot->R == b) {
        pRoot->inc += c;
        return ;
    }
    pRoot->sum += c * ( b - a + 1) ;
    if( b <= (pRoot->L + pRoot->R)/2)
        Add(pRoot->pLeft,a,b,c);
    else if( a >= (pRoot->L + pRoot->R)/2 +1)
        Add(pRoot->pRight,a,b,c);
    else {

        Add(pRoot->pLeft,a,
            (pRoot->L + pRoot->R)/2 ,c);
        Add(pRoot->pRight,
            (pRoot->L + pRoot->R)/2 + 1,b,c);
    }
}
```

```
long long Querysum( CNode * pRoot, int a, int b)
{
    if( pRoot->L == a && pRoot->R == b)
        return pRoot->sum +
            (pRoot->R - pRoot->L + 1) * pRoot->inc ;

    pRoot->sum += (pRoot->R - pRoot->L + 1) * pRoot->inc ;
    pRoot->pLeft->inc += pRoot->inc;
    pRoot->pRight->inc += pRoot->inc;
    pRoot->inc = 0;

    if( b <= pRoot->Mid())
        return Querysum(pRoot->pLeft,a,b) ;
    else if( a >= pRoot->Mid() + 1)
        return Querysum(pRoot->pRight,a,b) ;
    else {
        return Querysum(pRoot->pLeft,a,pRoot->Mid()) +
            Querysum(pRoot->pRight,pRoot->Mid() + 1,b) ;
    }
}
```

```
int main()
{
    int n,q,a,b,c;
    char cmd[10];
    scanf("%d%d",&n,&q);
    int i,j,k;
    count = 0;
    BuildTree(Tree,1,n);
    for( i = 1;i <= n;i ++ ) {
        scanf("%d",&a);
        Insert(Tree,i,a);
    }
}
```

```
for( i = 0;i < q;i ++ ) {  
    scanf("%s",cmd) ;  
    if ( cmd[0] == 'C' ) {  
        scanf("%d%d%d",&a,&b,&c) ;  
        Add( Tree,a,b,c) ;  
    }  
    else {  
        scanf("%d%d",&a,&b) ;  
        printf("%lld\n",Querysum(Tree,a,b)) ;  
    }  
}  
return 0;  
}
```



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题4

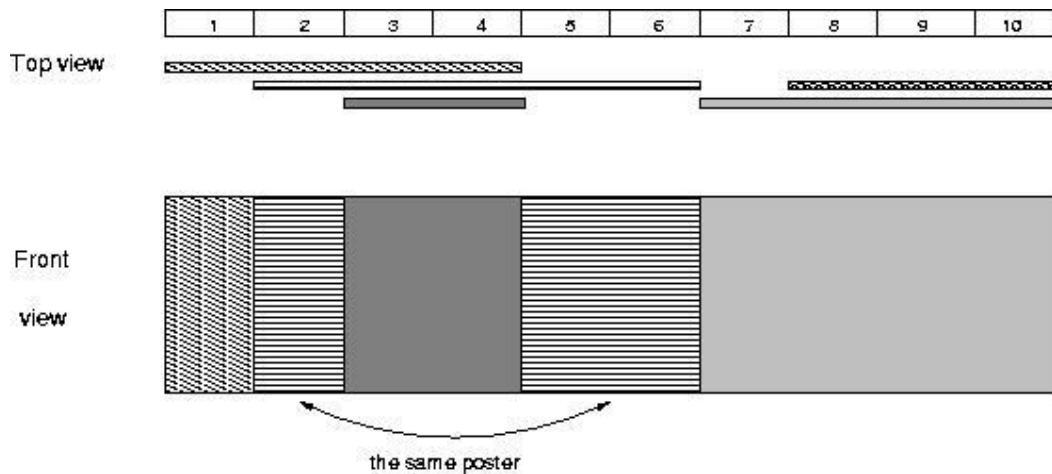
Mayor's posters



挪威盖朗厄尔峡湾

例题4： POJ 2528 Mayor's posters

给定一些海报，可能互相重叠，告诉你每个海报宽度（高度都一样）和先后叠放次序，问**没有被完全盖住**的海报有多少张。



海报最多10,000张，但是墙有10,000,000块瓷砖长。海报端点不会落在瓷砖中间。

例题4： POJ 2528 Mayor's posters

给瓷砖编号，一个海报就相当于一个整数区间。贴海报就是区间操作，查询海报是否可见也是区间操作。

因此可以用线段树来解决

例题4： POJ 2528 Mayor's posters

思路：依次贴上一张张海报，每贴一张海报，就询问这张海报有没有被全部遮住（假设贴海报的过程和现实可以不同，也可以先贴上上面的，再贴下面的）

贴的顺序如何选择？

例题4: POJ 2528 Mayor's posters

思路：依次贴上一张张海报，每贴一张海报，就询问这张海报有没有被全部遮住（假设贴海报的过程和现实可以不同，也可以先贴上面的，再贴下面的）

贴的顺序如何选择？

关键：插入数据的顺序 ----- 从上往下依次插入每张海报，这样后插入的海报不可能覆盖先插入的海报，因此插入一张海报时，如果发现海报对应的瓷砖有一块露出来，就说明该海报部分可见。

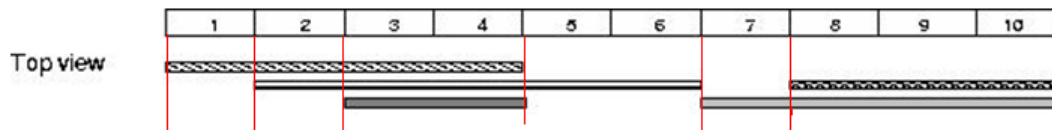
时间复杂度: $n \log n$ (n 为海报数目)

例题4： POJ 2528 Mayor's posters

如果每个叶子节点都代表一块瓷砖，那么线段树会导致MLE，即单位区间的数目太多。而且时间复杂度为 $n \log m$ (n 是海报数目 m 是瓷砖数目)

实际上，由于最多10,000个海报，共计20,000个端点，这些端点把墙最多分成19,999个单位区间（题意为整个墙都会被盖到）。每个单位区间的瓷砖数目可以不同

我们只要对这19,999个区间编号，然后建树即可。这就是**离散化**

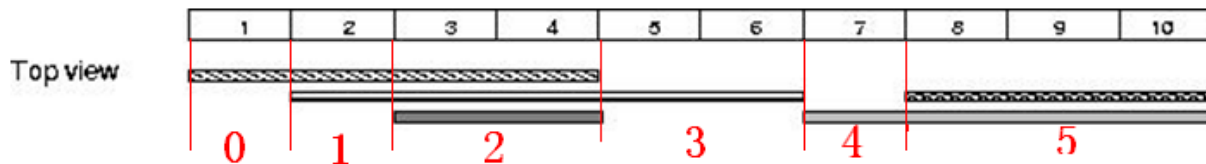


离散化

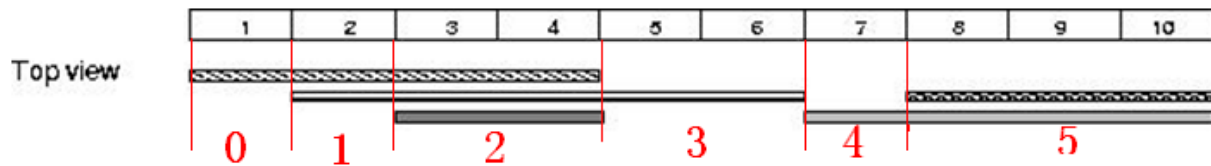
有时，区间的端点不是整数，或者区间太大导致建树内存开销过大MLE，那么就需要进行“离散化”后再建树。

例题4： POJ 2528 Mayor's posters

- 这些单位区间在线段树上是叶子节点
- 每个单位区间要么全被覆盖，要么全部露出
- 没有海报的端点会落在一个单位区间内部
- 每张海报一定完整覆盖若干个连续的单位区间
- 要算出一共有多少个单位区间，并且算出每张海报覆盖的单位区间[a,b] (海报覆盖了从a号单位区间到b号单位区间)

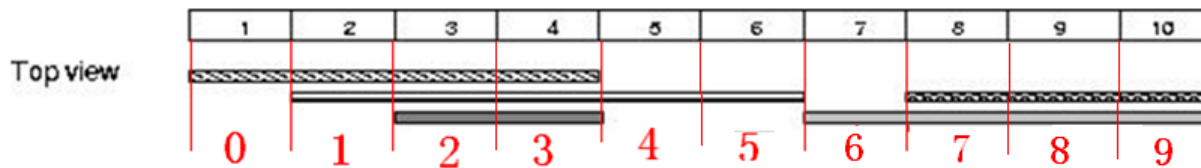


例题4: POJ 2528 Mayor's posters



按上图的离散化方法，求每张海报覆盖了哪些单位区间，写起来稍麻烦

例题4: POJ 2528 Mayor's posters



更方便的离散化方法，是将所有海报的端点瓷砖排序，把每个海报的端点瓷砖都看做一个单位区间，两个相邻的端点瓷砖之间的部分是一个单位区间

这样最多会有 $20000 + 19999$ 个单位区间

时间复杂度变为 $n \log n$ n 是海报数目

例题4： POJ 2528 Mayor's posters

如果海报端点坐标是浮点数，其实也一样处理。

树节点要保存哪些信息，而且这些信息该如何动态更新呢？

例题4： POJ 2528 Mayor's posters

```
struct CNode
{
    int L,R;
    bool covered;
    CNode * pLeft, * pRight;
};
```

covered表示本区间是否已经完全被海报盖住

例题4： POJ 2528 Mayor's posters

从上到下依次往线段树中插入海报

```
bool Post( CNode *pRoot, int L, int R);
```

插入一张海报覆盖了区间 $[L,R]$ 的那一部分，返回该部分是否可见（未被完全覆盖即为可见）

例题4： POJ 2528 Mayor's posters

从上到下依次往线段树中插入海报

```
bool Post( CNode *pRoot, int L, int R);
```

插入一张海报覆盖了区间[L,R]的那一部分，返回该部分是否可见（未被完全覆盖即为可见）

若 [L,R]与 pRoot节点重合，则根据 pRoot->covered 做判断，并修改pRoot->covered, 返回。

例题4： POJ 2528 Mayor's posters

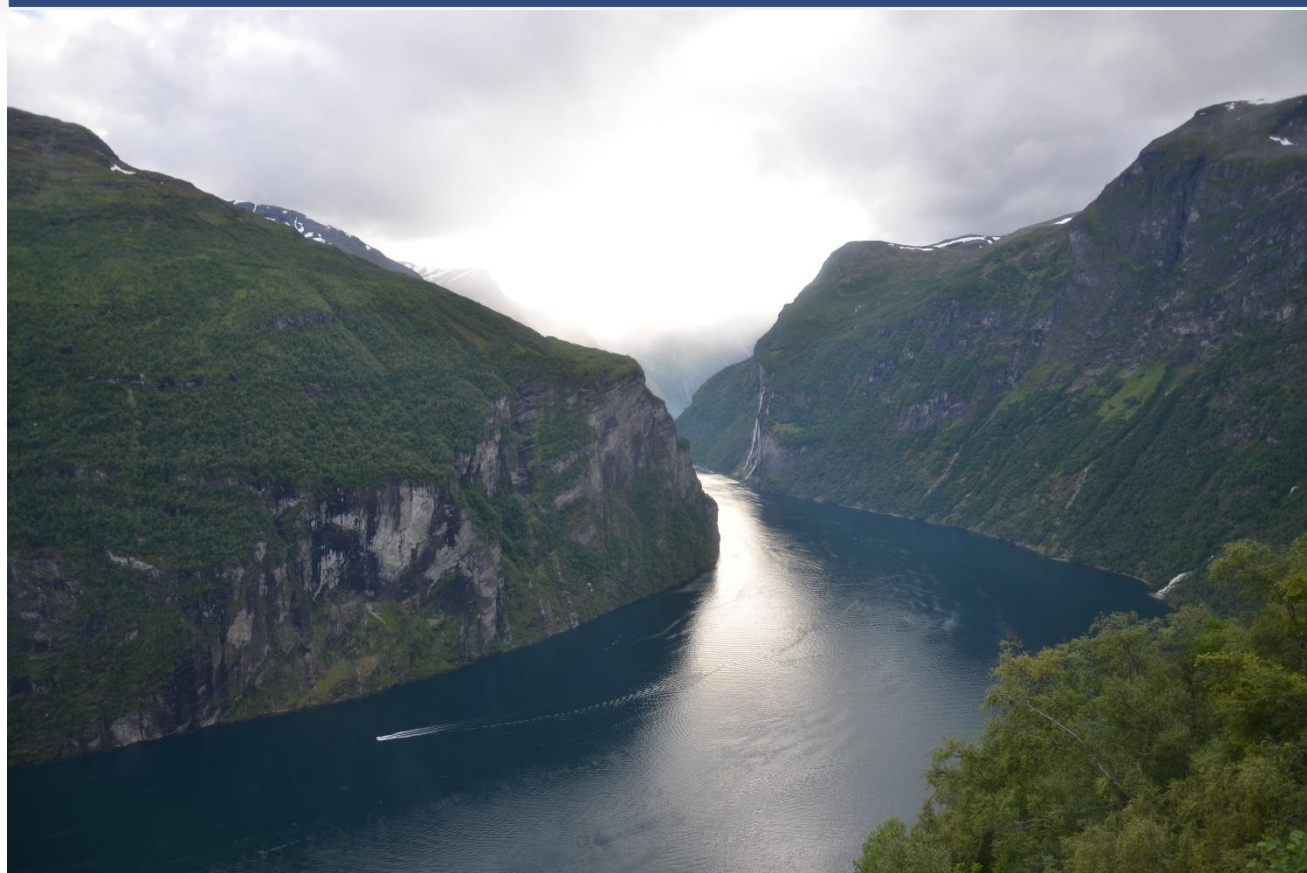
注意：若 $[L,R]$ 与 pRoot节点不重合，则继续往下插入。往下插入完成后，还要看看 pRoot是否被完全覆盖，有可能要更新 pRoot->covered。因pRoot对应区间可能已经被部分覆盖，加上新海报后，就可能被全部覆盖



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题5 Atlantis



挪威盖朗厄尔峡湾

例题5: POJ 1151 Atlantis

给定 n 个矩形 ($n \leq 100$), 其顶点坐标是浮点数, 可能互相重叠, 问这些矩形覆盖到的面积是多大。

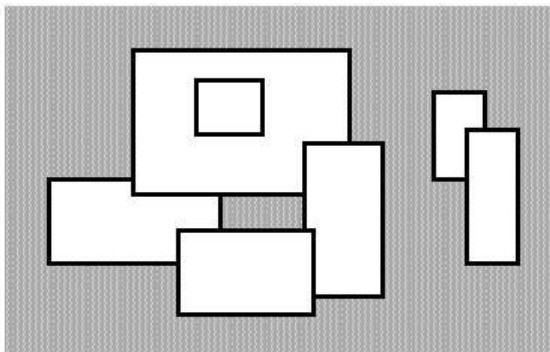
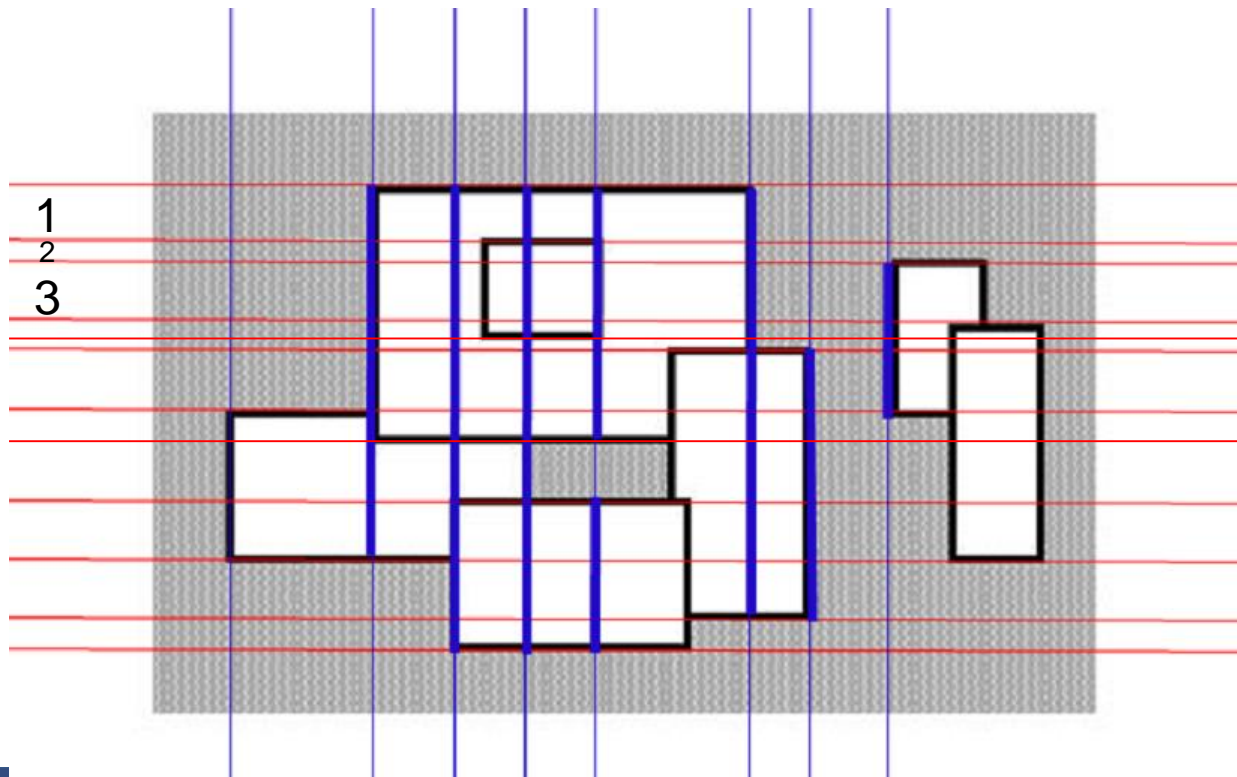


Figure 1. A set of 7 rectangles

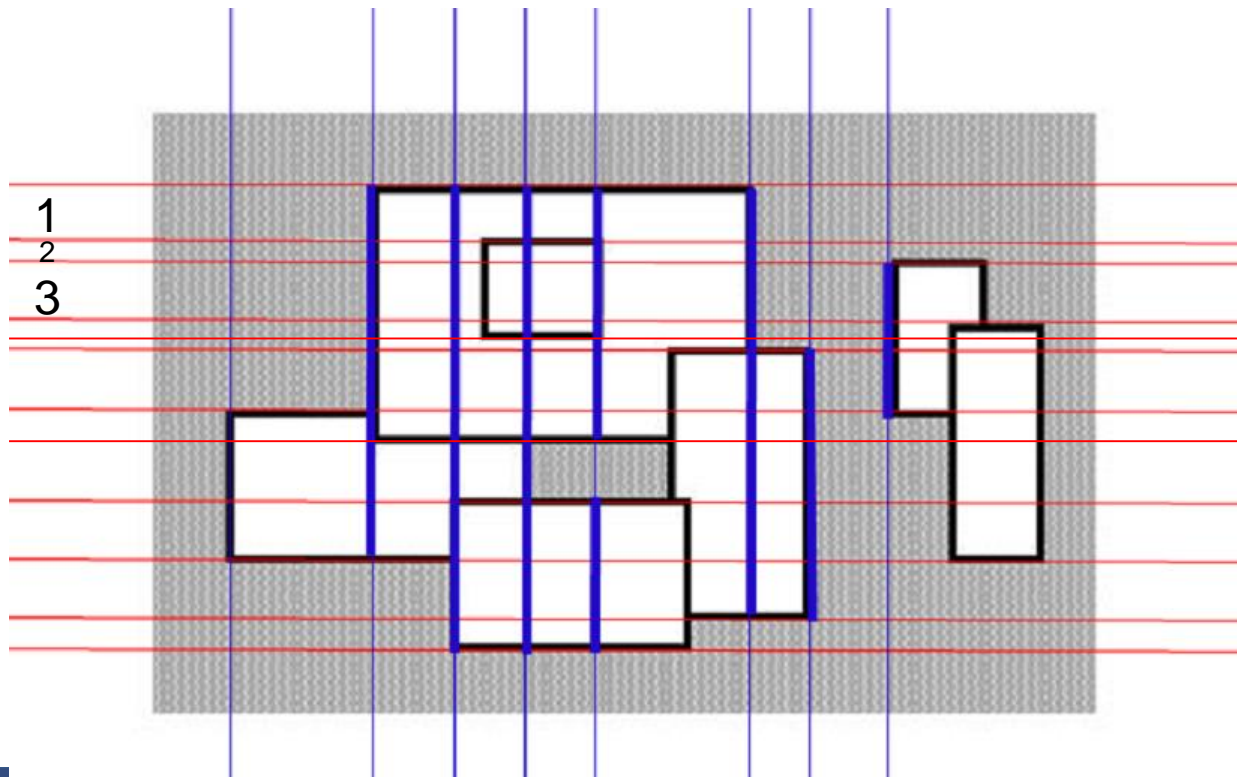
例题5: POJ 1151 Atlantis

在Y轴进行离散化。
 n 个矩形的 $2n$ 个横
边纵坐标共构成最
多 $2n-1$ 个区间的边
界，对这些区间编
号，建立起线段树



例题5: POJ 1151 Atlantis

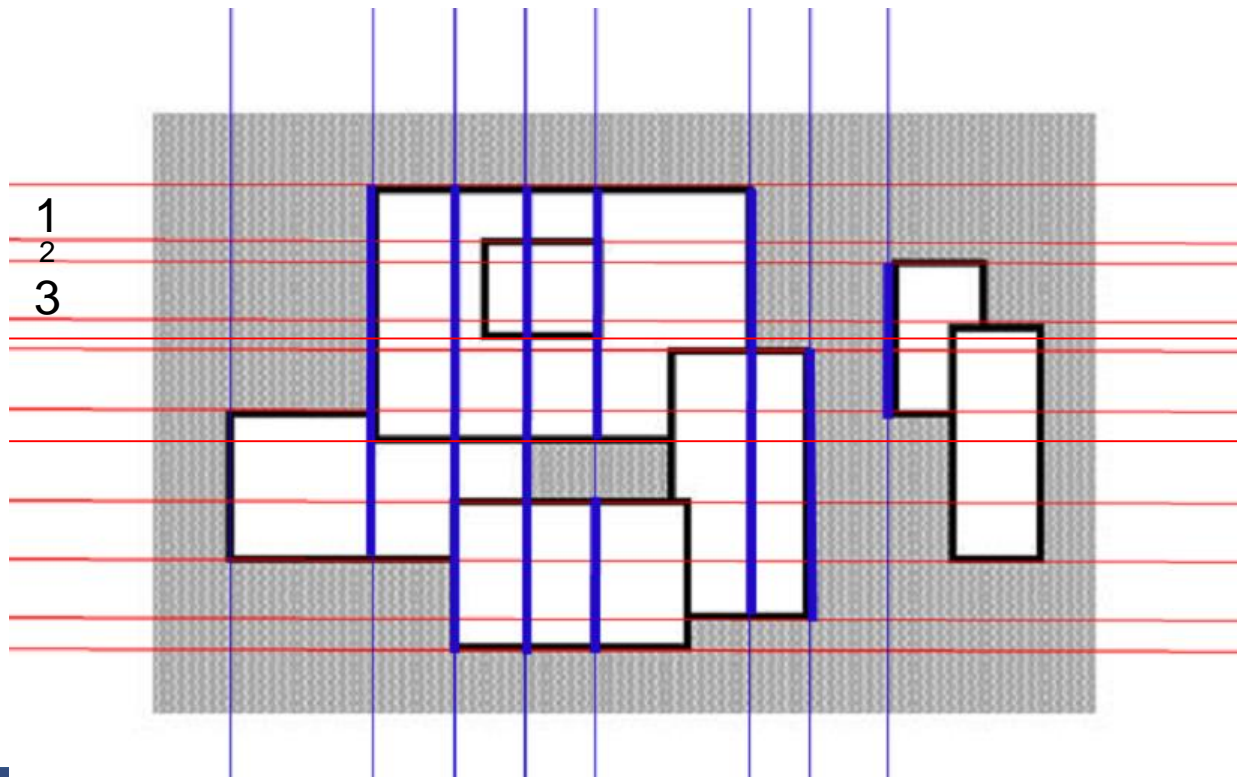
用一条直线从左到右扫描，碰到一条矩形竖边的时候，就计算该直线有多长被矩形覆盖。碰到矩形左边，要增加被覆盖的长度，碰到右边，要减少被覆盖的长度



例题5: POJ 1151 Atlantis

随着扫描线的右移动,
覆盖面积不断增加。

每碰到一条矩形的纵边,
覆盖面积就增加 Len 乘
以该纵边到下一条纵边
的距离。 Len 是此时扫
描线被矩形覆盖的长度



例题5: POJ 1151 Atlantis

线段树的节点要保存哪些信息？如何将一个个矩形插入线段树？插入过程中这些信息如何更新？怎样查询？

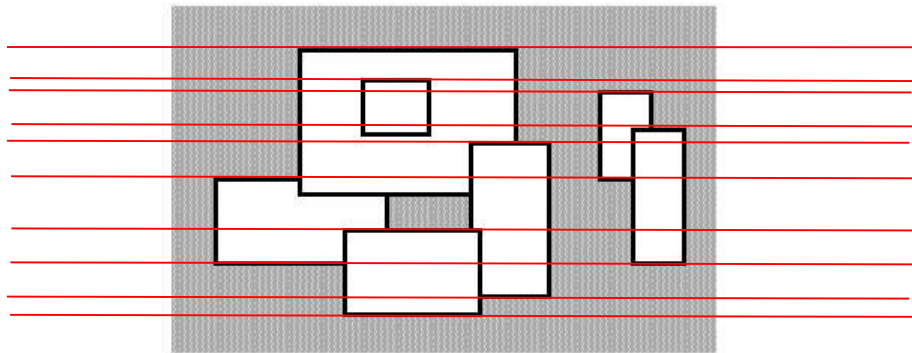


Figure 1. A set of 7 rectangles

例题5: POJ 1151 Atlantis

```
struct CNode
{
    int L,R;
    CNode * pLeft, * pRight;
    double Len; //当前,本区间上有多长的部分被矩形覆盖
    int Covers; //本区间当前被多少个矩形完全包含
};
```

一开始, 所有区间 $Len = 0$ $Covers = 0$

例题5: POJ 1151 Atlantis

插入数据的顺序:

将矩形的纵边从左到右排序, 然后依次将这些纵边插入线段树。要记住哪些纵边是一个矩形的左边(开始边), 哪些纵边是一个矩形的右边(结束边), 以便插入时, 对Len和Covers做不同的修改。

插入一条边后, 就根据根节点的Len 值增加总覆盖面积的值。
增量是 $\text{Len} * \text{本边到下一条边的距离}$

例题5: POJ 1151 Atlantis

```
#include <iostream>
#include <algorithm>
#include <math.h>
#include <set>
using namespace std;
double y[210];
struct CNode
{
    int L,R;
    CNode * pLeft, * pRight;
    double Len; //当前,本区间上有多长的部分是落在那些矩形中的
    int Covers; //本区间当前被多少个矩形完全包含
};
CNode Tree[1000];
```

```
struct CLine
{
    double x,y1,y2;
    bool bLeft; //是否是矩形的左边
} lines[210];

int nNodeCount = 0;

bool operator< ( const CLine & l1,const CLine & l2)
{
    return l1.x < l2.x;
}
```

```
template <class F,class T>
F bin_search(F s, F e, T val)
{    //在区间[s,e)中查找 val,找不到就返回 e
        F L = s;
        F R = e-1;
    while(L <= R )    {
        F mid = L + (R-L)/2;
        if( !( * mid < val  || val < * mid ))
            return mid;
        else if(val < * mid)
            R = mid - 1;
        else
            L = mid + 1;
    }
    return e;
}

int Mid(CNode * pRoot)
{
    return (pRoot->L + pRoot->R ) >>1;
}
```

```
void Insert(CNode * pRoot,int L, int R) {  
    //在区间pRoot 插入矩形左边的一部分或全部, 该左边的一部分或全部覆盖了区间[L,R]  
    if( pRoot->L == L && pRoot->R == R){  
        pRoot->Len = y[R+1] - y[L];  
        pRoot->Covers ++;  
        return;  
    }  
    if( R <= Mid(pRoot))  
        Insert(pRoot->pLeft,L,R);  
    else if( L >= Mid(pRoot)+1)  
        Insert(pRoot->pRight,L,R);  
    else {  
        Insert(pRoot->pLeft,L,Mid(pRoot));  
        Insert(pRoot->pRight,Mid(pRoot)+1,R);  
    }  
    if( pRoot->Covers == 0) //如果不为0, 则说明本区间当前仍然被某个矩形完全  
    包含, 则不能更新 Len  
        pRoot->Len = pRoot->pLeft ->Len +  
            pRoot->pRight ->Len;  
}
```

```
void Delete(CNode * pRoot,int L, int R) {
```

/在区间pRoot 删除矩形右边的一部分或全部, 该矩形右边的一部分或全部覆盖了区间[L,R]

```
    if( pRoot->L == L && pRoot->R == R) {
```

```
        pRoot->Covers --;
```

```
        if( pRoot->Covers == 0 )
```

```
            if( pRoot->L == pRoot->R ) pRoot->Len = 0;
```

```
            else pRoot->Len = pRoot->pLeft ->Len +  
                            pRoot->pRight ->Len;
```

```
        return ;
```

```
    }
```

```
    if( R <= Mid(pRoot))    Delete(pRoot->pLeft,L,R);
```

```
    else if( L >= Mid(pRoot)+1)    Delete(pRoot->pRight,L,R);
```

```
    else {
```

```
        Delete(pRoot->pLeft,L,Mid(pRoot));
```

```
        Delete(pRoot->pRight,Mid(pRoot)+1,R);
```

```
    }
```

if(pRoot->Covers == 0) //如果不为0, 则说明本区间当前仍然被某个矩形完全包含, 则不能更新 Len

```
        pRoot->Len = pRoot->pLeft ->Len + pRoot->pRight ->Len;
```

```
}
```



```
void BuildTree( CNode * pRoot, int L,int R)
{
    pRoot->L = L;
    pRoot->R = R;
    pRoot->Covers = 0;
    pRoot->Len = 0;
    if( L == R)
        return;
    nNodeCount ++;
    pRoot->pLeft = Tree + nNodeCount;
    nNodeCount ++;
    pRoot->pRight = Tree + nNodeCount;
    BuildTree( pRoot->pLeft,L, (L+R)/2);
    BuildTree( pRoot->pRight, (L+R)/2+1,R);
}
```

```
int main()
{
    int n;  int i,j,k;    double x1,y1,x2,y2;  int yc,lc;
    int nCount = 0;
    int t = 0;
    while(true) {
        scanf("%d",&n);
        if( n == 0)  break;
        t ++;    yc = lc = 0;
        for( i = 0;i < n;i ++ ) {
            scanf("%lf%lf%lf%lf", &x1, &y1,&x2,&y2);
            y[yc++] = y1;    y[yc++] = y2;
            lines[lc].x = x1;        lines[lc].y1 = y1;
            lines[lc].y2 = y2;
            lines[lc].bLeft = true;
            lc ++;
            lines[lc].x = x2;        lines[lc].y1 = y1;
            lines[lc].y2 = y2;
            lines[lc].bLeft = false;
            lc ++;
        }
    }
}
```

```
sort(y,y + yc);  
yc = unique(y,y+yc) - y;  
nNodeCount = 0;
```

//yc是横线的条数, yc- 1是纵向区间的个数, 这些区间从0开始编号, 那么最后一个区间编号就是yc-1-1

```
BuildTree(Tree, 0, yc - 1 - 1);  
sort(lines,lines + lc);  
double Area = 0;  
for( i = 0;i < lc - 1 ; i ++ ) {  
    int L = bin_search( y,y+yc,lines[i].y1) - y;  
    int R = bin_search( y,y+yc,lines[i].y2) - y;  
    if( lines[i].bLeft ) Insert(Tree,L,R-1);  
    else Delete(Tree,L,R-1);  
    Area += Tree[0].Len * (lines[i+1].x - lines[i].x);  
}  
printf("Test case #%d\n",t);  
printf("Total explored area: %.21f\n",Area);  
printf("\n",Area);  
}  
return 0;
```

```
}
```



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题6 K-th Number



瑞士马特洪峰

例题6: POJ 2104 K-th Number

给定数列 $a_1, a_2 \dots a_n$, 各不相同。要 m 次查询 $a[L \dots R]$ 中第 k 小的数是什么 ($1 \leq n \leq 100\,000$, $1 \leq m \leq 5\,000$) $\text{abs}(a_i) \leq 10^9$

样例输入

7 3 //7个数, 3次查询

1 5 2 6 3 7 4

2 5 3

4 4 1

1 7 3

样例输出

5

6

3

例题6: POJ 2104 K-th Number

- 构造线段树，用区间分解硬求 $a[L...R]$ 中第 k 小的数，会陷入死胡同。将 $[L,R]$ 分成两半后，总是需要两边每个元素都看才能知道第 k 小的是哪个

例题6: POJ 2104 K-th Number

- 构造线段树，用区间分解硬求 $a[L...R]$ 中第 k 小的数，会陷入死胡同。将 $[L,R]$ 分成两半后，总是需要两边每个元素都看才能知道第 k 小的是哪个
- 转变思路：

如果硬求不行，可以硬试。枚举答案，进行验证

例题6: POJ 2104 K-th Number

- 枚举第k小的数 X 。枚举范围是 a_i 的取值范围。对假定的 X ，用线段树查询 $a[L...R]$ 中小于等于 X 的数的个数 num ，看是否正好是 k 。

例题6: POJ 2104 K-th Number

- 枚举第k小的数 X 。枚举范围是 a_i 的取值范围。对假定的 X ，用线段树查询 $a[L...R]$ 中小于等于 X 的数的个数 num ，看是否正好是 k 。
- 用二分法加快枚举。若 $num \geq k$ ，则减小 X ，否则加大 X 。最小的使得 $num \geq k$ 的 X ，就是问题的答案(此时 $num = k$)

例题6: POJ 2104 K-th Number

- 枚举第k小的数 X 。枚举范围是 a_i 的取值范围。对假定的 X ，用线段树查询 $a[L...R]$ 中小于等于 X 的数的个数 num ，看是否正好是 k 。
- 用二分法加快枚举。若 $num \geq k$ ，则减小 X ，否则加大 X 。最小的使得 $num \geq k$ 的 X ，就是问题的答案(此时 $num = k$)
- 用离散化减少 X 的枚举范围。构建数组 b ， $b[i]$ 为 $a[i]$ 在 a 数组中从小到大排序后的序号。则所有查询等价于在 b 数组上查询。枚举 X 的范围变成 $[1, n]$ 。再由 b 上的答案 $b[i]$ 找回对应的 $a[i]$

例题6: POJ 2104 K-th Number

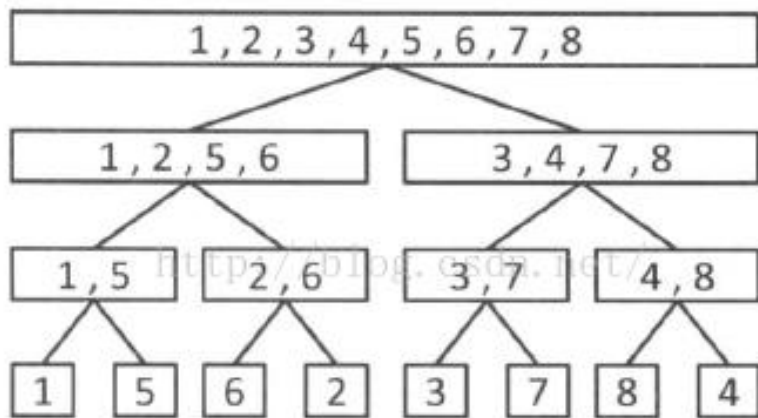
- 线段树节点存放什么信息?

例题6: POJ 2104 K-th Number

- 线段树节点存放什么信息?

存放对应区间 $[L,R]$ 中所有数的排序的结果 `vect<int> va`

数列: 1 5 6 2 3 7 8 4



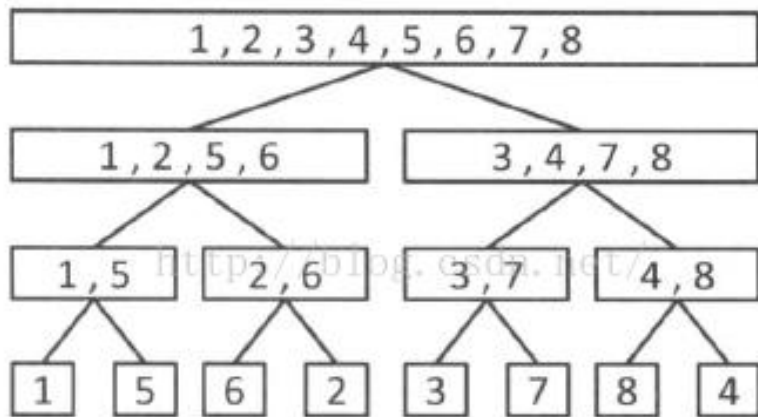
如何实现?

例题6: POJ 2104 K-th Number

- 线段树节点存放什么信息?

存放对应区间 $[L,R]$ 中所有数的排序的结果 `vect<int> va`

数列: 1 5 6 2 3 7 8 4



如何实现?

归并排序!

空间 $O(n\log(n))$

例题6: POJ 2104 K-th Number

- 若待查询区间 $[L, R]$ 和树节点区间重合, 如何查询 $[L, R]$ 中有几个数比 X 小?

例题6: POJ 2104 K-th Number

- 若待查询区间 $[L, R]$ 和树节点区间重合, 如何查询 $[L, R]$ 中有几个数比 X 小?

在 va 中用 `upper_bound` 即可, 实际上是二分

例题6: POJ 2104 K-th Number

- 总的复杂度?

例题6: POJ 2104 K-th Number

- 总的复杂度?

建树就是归并排序, $O(n\log(n))$

一次查询: $O(\log^3(n))$

总复杂度 $m\log^3(n)$



北京大学
PEKING UNIVERSITY

信息科学技术学院

二维线段树



挪威大洋路

二维线段树

- 一维线段树的每个节点代表一个一维区间，那么二维线段树的每个节点就代表一个二维区间（一个矩阵）
- 二维线段树是一棵4叉树。如果一个节点代表的区间是矩阵 $[x1, y1] - [x2, y2]$ ，那么它的四个子节点代表的区间分别为

$$[x1, y1] - [(x1+x2)/2, (y1+y2)/2]$$

$$[x1, (y1+y2)/2+1] - [(x1+x2)/2, y2]$$

$$[(x1+x2)/2+1, y1] - [x2, (y1+y2)/2]$$

$$[(x1+x2)/2+1, (y1+y2)/2+1] - [x2, y2]$$

参考：<http://www.cppblog.com/menjitianya/archive/2011/04/03/143374.html>

二维线段树

- 还可以用树套树方式实现，即每个外层线段树的节点对应于一棵内层线段树。如果外层线段树根对应的区间是y方向的 $[1,n]$ ，内层线段树根节点对应的区间是x方向的 $[1,m]$ ，那么整个线段树可以存在一个二维数组里。
- 树套树方式插入、修改、查找等时间复杂度为 $O(\log(n)*\log(m))$

二维线段树

```
Node Tree[MAX_N * 4][MAX_M * 4];
```

二维线段树，每一行都是一棵二叉树，用于存放一棵x方向线段树（列线段树），树节点只存放对应的区间（矩形）的信息

9 * 9 矩阵的行线段树(列线段树结构一样) :



问: Tree[2][4]
对应于哪个矩阵?

Tree[0][0] 是二维线段树的根节点, 对应于整个矩阵

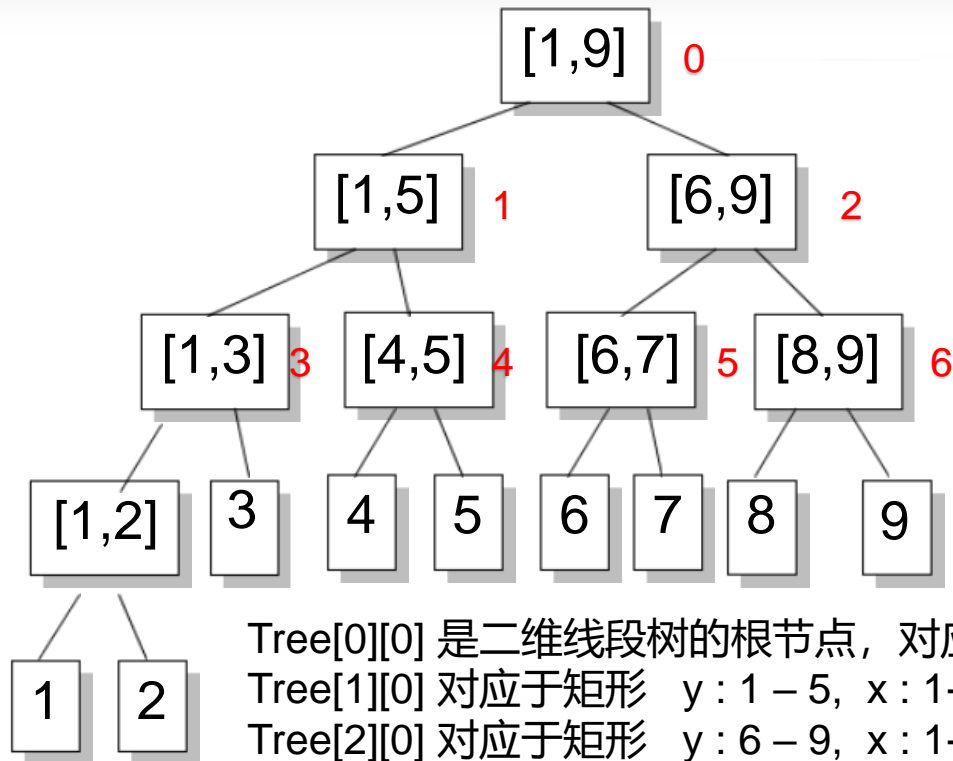
Tree[1][0] 对应于矩形 $y : 1 - 5, x : 1-9$

Tree[2][0] 对应于矩形 $y : 6 - 9, x : 1-9$

Tree[3][0] 对应于矩形 $y : 1 - 3, x : 1-9$

Tree[4][0] 对应于矩形 $y : 4 - 5, x : 1-9$

9 * 9 矩阵的行线段树(列线段树结构一样) :



问: Tree[2][4]
对应于哪个矩阵?

答: $y : 6 - 9$, $x : 4-5$

Tree[0][0] 是二维线段树的根节点, 对应于整个矩阵

Tree[1][0] 对应于矩形 $y : 1 - 5$, $x : 1-9$

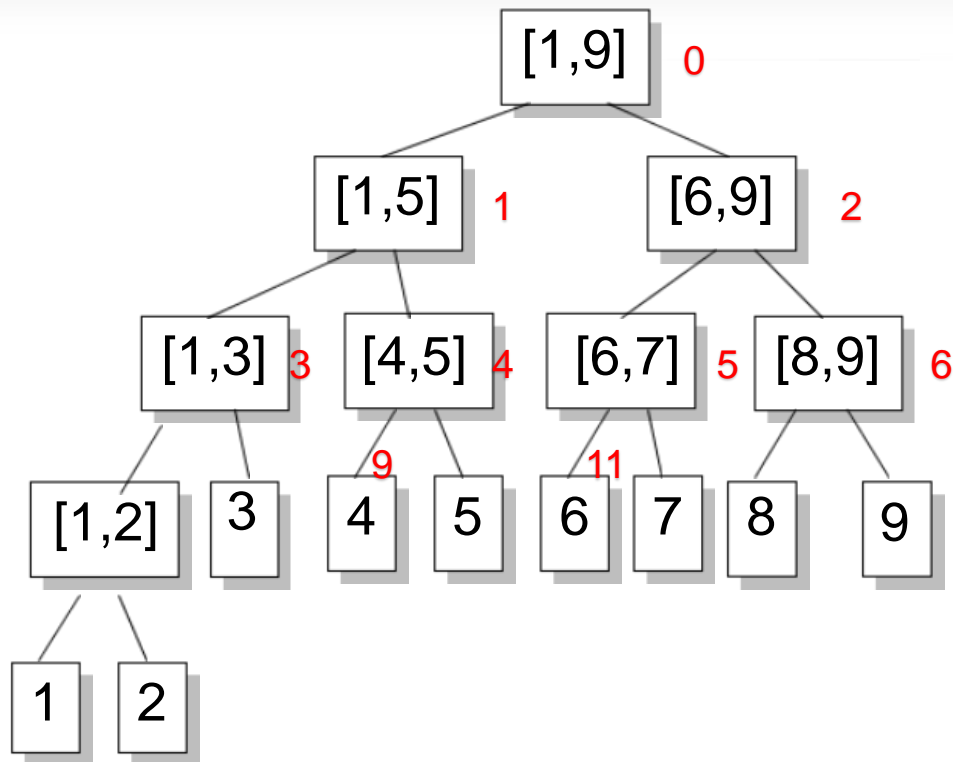
Tree[2][0] 对应于矩形 $y : 6 - 9$, $x : 1-9$

Tree[3][0] 对应于矩形 $y : 1 - 3$, $x : 1-9$

Tree[4][0] 对应于矩形 $y : 4 - 5$, $x : 1-9$

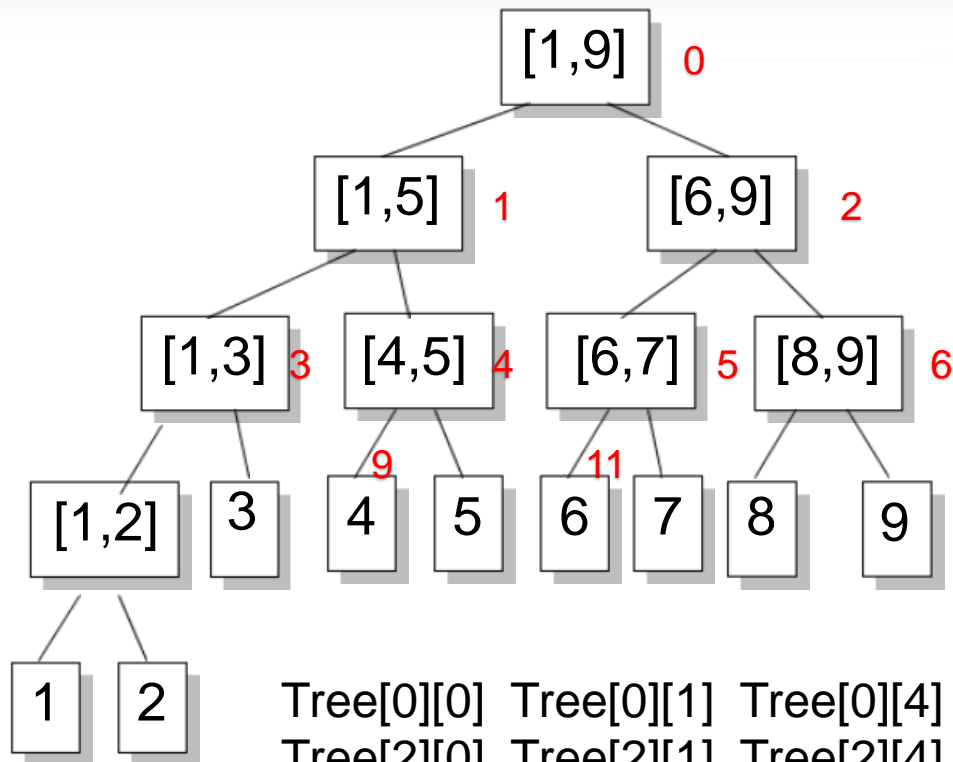
Tree[1][2] 对应于矩形 $y : 1 - 5$, $x : 6-9$

9 * 9 矩阵的行线段树(列线段树结构一样) :



问：对矩阵元素
(6,4) 加a,那么
Tree数组里面哪
些元素要修改？

9 * 9 矩阵的行线段树(列线段树结构一样) :



问: 对矩阵元素
(6,4) 加a,那么
Tree数组里面哪
些元素要修改?

Tree[0][0] Tree[0][1] Tree[0][4] Tree[0][9]
Tree[2][0] Tree[2][1] Tree[2][4] Tree[2][9]
Tree[5][0] Tree[5][1] Tree[5][4] Tree[5][9]
Tree[11][0] Tree[11][1] Tree[11][4] Tree[11][9]



北京大学
PEKING UNIVERSITY

信息科学技术学院

二维线段树例题 Mobile phones



冰岛黄金瀑布

用二维线段树做POJ 1195 Mobile phones

给定一个矩阵 a ，不断进行两种操作

- 1) 给 $a[i][j]$ 加上 x
- 2) 询问从 $a[x1][y1]$ 到 $a[x2][y2]$ 的子矩阵中所有数的和

用二维线段树做POJ 1195 Mobile phones

```
#include <iostream>
using namespace std;
#define MY_MAX 1100
int Tree[MY_MAX * 4][MY_MAX * 4]; //二维线段树, 每一行都是一棵二叉树, 用于存放一棵x
//方向线段树, 树节点只存放 对应的区间 (矩阵) 的数字之和
int S; //矩阵宽度
void Add_x( int rooty, int rootx, int L, int R, int x ,int a)
//tree[rooty][rootx]对应的矩阵x方向上范围是[L,R]
//操作rooty那一行的行线段树
{
    Tree[rooty][rootx] += a;
    if( L == R )
        return;
    int mid = (L + R )/2;
    if( x <= mid )
        Add_x(rooty, ( rootx << 1) + 1, L ,mid, x, a);
    else
        Add_x(rooty, ( rootx << 1) + 2, mid + 1,R, x, a);
}
```

```
void Add_y(int rooty, int L,int R, int y, int x,int a)
//tree[rooty][xxx]对应的矩阵y方向上范围是[L,R]
//操作根为rooty的行线段树。行线段树上的每个节点
//对应于一棵列线段树，即tree数组的一行)
{
    Add_x( rooty,0, 1, S, x,a); //更新第rooty行的列线段树
    if( L == R)
        return;
    int mid = (L + R )/2;
    if( y <= mid )
        Add_y( ( rooty << 1) + 1, L, mid,y, x, a);
    else
        Add_y( ( rooty << 1) + 2, mid+1, R, y, x, a);
}
```

```
int QuerySum_x(int rooty, int rootx, int L, int R ,int x1,int x2)
{ //查询第rooty行的列线段树.
    if( L == x1 && R == x2)
        return Tree[rooty][rootx];
    int mid = ( L + R ) /2 ;
    if( x2 <= mid )
        return QuerySum_x( rooty, (rootx << 1) + 1,
                            L, mid,x1,x2);
    else if( x1 > mid )
        return QuerySum_x( rooty, (rootx << 1) + 2,
                            mid+1,R, x1,x2);
    else
        return QuerySum_x( rooty,(rootx << 1) + 1,
                            L, mid ,x1,mid) +
            QuerySum_x( rooty,(rootx << 1) + 2,
                            mid + 1, R, mid + 1,x2);
}
```



```
int main() {
    int cmd,x,y,a,l,b,r,t,Sum = 0;
    while( true) {
        scanf("%d",&cmd);
        switch( cmd) {
            case 0:
                scanf("%d",& S);
                memset( Tree,0,sizeof(Tree));
                break;
            case 1:
                scanf("%d%d%d",&x ,&y,&a);
                Add_y(0, 1,S, y + 1, x + 1, a);
                break;
            case 2:
                scanf("%d%d%d%d",&l , &b, &r,&t);
                l ++; b++; r ++; t ++;
                printf("%d\n",QuerySum_y(0,1,S,b,t,l,r));
                break;
            case 3:
                return 0;
        }
    }
}
```


二维线段树好题：POJ 2155 Matrix

$N * N$ 的矩阵，每个元素要么是0，要么是1，开始全0

不断进行两种操作：

- 1) C x1 y1 x2 y2 表示要将左上角为 $(x1, y1)$, 右下角为 $(x2, y2)$ 的子矩阵里的全部元素都取反 (0变1, 1变0)
($1 \leq x1 \leq x2 \leq n, 1 \leq y1 \leq y2 \leq n$)
- 2) Q x y 查询 (x, y) 处元素的值

线段树POJ题目推荐:

2182, 2352, 1177, 3667, 3067