线段树及其应用

长沙市雅礼中学 朱全民

中国计算机学会 China Computer Federation

问题1: RMQ

- RMQ(Range Minimum/Maximum Query)问题是指：对于长度为n的数列A，回答若干询问RMQ(A,i,j)(i,j<=n)，返回数列A中下标在[i,j]里的最小(大）值，也就是说，RMQ问题是指求区间最值的问题。

题目描述:

给定一个数的序列，查询任意给定区间内数的最小值。

输入:

一个整数n(1<=n<=100000)，代表数字序列的长度。接下去一行给出n个数，代表数字序列。数在int范围内。

下一行为一个整数t(1<=t<=10000),代表查询的次数。最后t行，每行给出一个查询，由两个整数表示l、r(1<=l<=r<=n)。

输出:

对于每个查询，输出区间[l,r]内的最小值。

样例 输入:

```
5
3 2 1 4 3
3
1 3
2 4
4 5
```

样例输出:

```
1
1
3
```

- 方法一： 直接搜索

读入数据后，对每一个询问，采用枚举方法查找最小值，设有T个询问，则时间复杂度为O(n*T)。

```
#include<bits/stdc++.h>
const int MAX=100010;
int main(){
        int a[MAX];
        int n,i,T,L,R,ans;
        scanf("%d",&n);
        for(i=1;i<=n;i++) scanf("%d",&a[i]);
        scanf("%d",&T);
        while(T--){            /* 对T个询问逐一求解 */
                scanf("%d%d",&L,&R);
                int ans=0x7FFFFFFF;
                for(i=L;i<=R;i++) if(a[i]<ans) ans=a[i];  /*枚举比较区间的每一个值*/
                printf("%d\n",ans);
        }
        return 0;
}
```

中国计算机学会 China Computer Federation

方法二：动态规划

设dp[i][j]表示从第i个数开始长度为j的区间最小值，则dp[i][j]=min{dp[i][j-1],a[j]},求出dp[i][j]后，询问可查表得出。

此算法时间复杂度与询问次数无关，但求dp[i][j]的时间复杂度显然为$O(n^2)$。

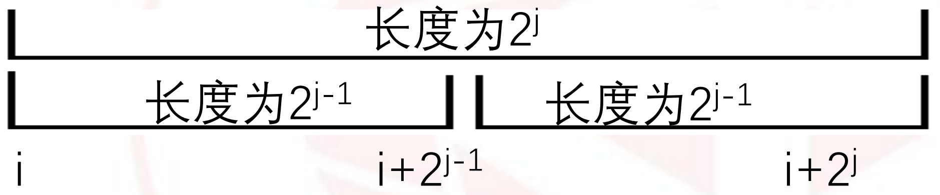
```
#include<bits/stdc++.h>
const int MAX=100010;
int main(){
        int a[MAX],f[MAX][MAX];
        int n,T,L,R,ans;
        scanf("%d",&n);for(int i=1;i<=n;i++) scanf("%d",&a[i]);
        for(int i=1;i<=n;i++) dp[i][1]=a[i];      //长度为1的值;
        for(int j=2;j<=n;i++) for(int i=1;i<=n-j+1; i++) dp[i][j]=min(dp[i][j-1],a[j]);
        scanf("%d",&T);
        while(T--){          /* 对T个询问逐一求解 */
                scanf("%d%d",&L,&R);
                printf("%d\n",dp[L][R-L+1]);
        }
        return 0;
}
```

方法三：ST算法——动态规划+倍增

设dp[i][j]表示从第i个数开始长度为2^j的区间最小值，则

$$dp[i][j]=\min\{dp[i][j-1], dp[\ i + 2^{j-1}][\ j - 1\ ]\}$$

计算dp[i][j]的时间复杂度显然为O(n*logn)



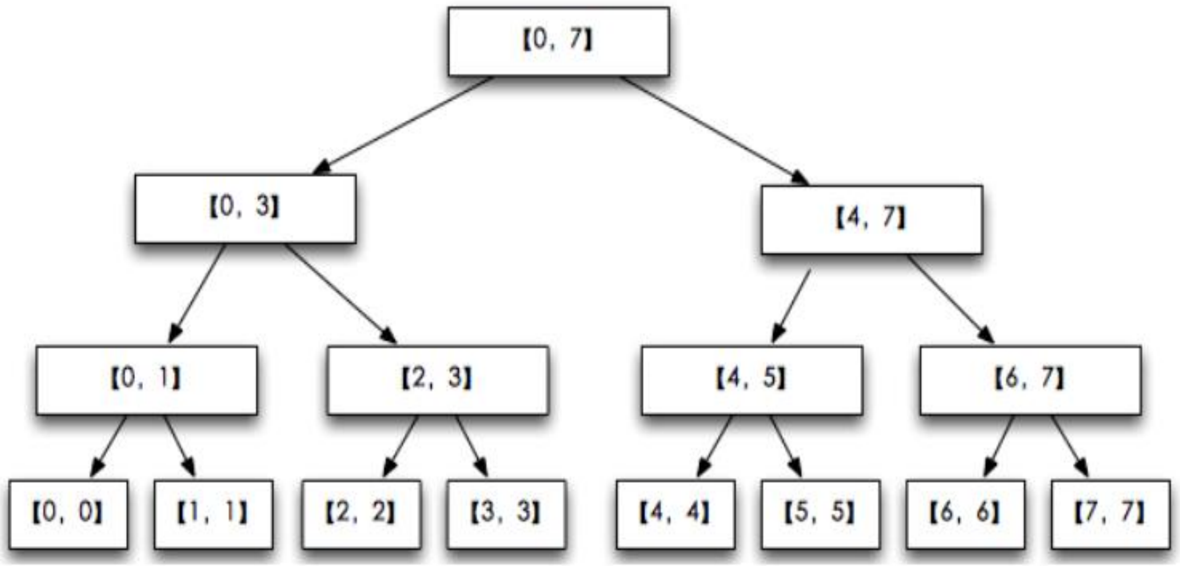
查找区间[L,R]的最小值，区间长度为R-L+1

令$k=\log_2(R-L+1)$，则 $ans=\min(dp[L][k],dp[R-2^{j-1}+1][k])$

```
//  ST算法
#include<bits/stdc++.h>
const int MAX=100010;
int main(){
        int a[MAX],f[MAX][MAX];
        int n,T,L,R,ans;
        scanf(“%d”,&n);for(int i=1;i<=n;i++) scanf("%d",&a[i]);
        for(int i=1;i<=n;i++) dp[i][0]=a[i];     //长度为1的值;
        for(int j=1;(1<<j)<=n;j++) for(int i=1;i+(1<<j)-1<=n;i++)
          dp[i][j]=min(dp[i][j-1],dp[i+(1<<j-1)][j-1]);
        scanf("%d",&T);
        while(T--){          /* 对T个询问逐一求解 */
                scanf("%d%d",&L,&R);
                int k=log2(R-L+1);
                ans=min(dp[L][k],dp[R-(1<<k)+1][k]);
                printf(“%d\n”,ans]);
        }
        return 0;
}
```

方法四：线段树

线段树是一种二叉搜索树，与区间树相似，**它将一个区间划分成一些单元区间，每个单元区间对应线段树中的一个叶结点。**对于线段树中的每一个非叶子节点[a,b]，它的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]。因此线段树是平衡二叉树，最后的子节点数目为N，即整个线段区间的长度。



N=8的线段树

线段树的特征

- 线段树的深度不超过logN。
- 线段树把区间上的任意一条线段都分成不超过2logN条线段。
- 这些结论为线段树能在O(logN)的时间内完成一条线段的插入、删除、查找等工作，提供了理论依据。

线段树的链式存储结构

线段树是一棵**平衡二叉树**，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。根节点表示的是所有的区间。

```
typedef struct node {
        int  l, r；  //左右边界
        int  key；  //key为[l,r]区间最小值
        struct  node  *lc,  *rc；  //指向线段节点的左右孩子指针
} node；
```

线段树的顺序存储结构

线段树可以**类似完全二叉树**，对于每一个非叶结点所表示的区间[a,b]，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]。

数据结构定义如下：

```
typedef struct node {
    int  l,r;    //l,r表示区间的左端点和右端点，闭区间。
    int  key; //min为[l,r]区间最小值
}tree[4*MAXN];
```

注意：线段树的大小需要要开4*MAXN，为什么，请大家思考？

线段树为什么要开4*MAXN

首先线段树是一棵二叉树，最底层有n个叶子节点（n为区间大小），那么由此可知，此二叉树的高度为$\log_2 n$ ，可证 $[\log_2 n] <= \log_2 n + 1$，然后通过等比数列求和公式，求得二叉树的节点个数，具体公式为：

$$\frac{1*(1-2^x)}{1-2}, x\text{为树的层数，即树的高度}+1$$

化简后得到，$2^{\log_2 n+1+1}-1$

整理后即为4n（近似计算，忽略-1）

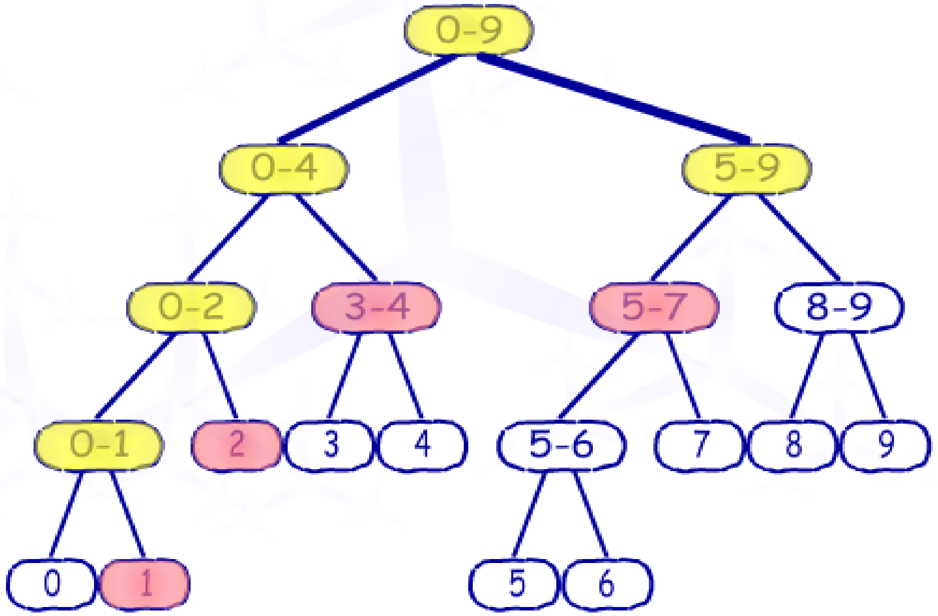
线段树的构造

```
void BuildTree(int left,int right,int u)   //构造[left,right]的线段树
{
    tree[u].l=left;  tree[u].r=right;
    if(left==right)
        tree [u].key=a[left];
    else {
        BuildTree(left, (left+right)>>1, 2*u);
        BuildTree(((left+right)>>1)+1, right, 2*u+1);
        tree[u].key=min(tree[2*u].key, tree[2*u+1]. key);
    }
}
```

线段树的查询

A:	2	5	6	9	1	5	6	1	7	4

- 要查询中区间[1,7]的最小值



线段树的查询，查询某条线段的最小值

```
int query(int left,int right,int u)
{
    if(tree[u].l==left && tree[u].r==right) return tree[u]. key;      //找到线段
    if(right<=tree[2*u].r)  return query(left,right,2*u);             //沿左儿子查找
    if(left>=tree[2*u+1].l)  return query(left,right,2*u+1);          //沿右儿子查找
    int mid=(tree[u].l+tree[u].r)>>1;
    return  min(query(left,mid,2*u),query(mid+1,right,2*u+1));  //左右儿子最小值
}
```

线段树解RMQ问题

```
int main
 {        int i,n,ans;  int T, L, R;
          scanf(“%d”,&n);
          for(i=1;i<=n;i++) scanf("%d",&a[i]);
          BuildTree(1,n,1);
          scanf("%d",&T);
          while(T--) {
                    scanf("%d%d",&L,&R); printf("%d\n", query(L,R,1));
           }
          return 0;
}
```

问题2:求一维序列的区间和

- 序列的区间和问题是指：对于长度为n的数列A，回答若干询问sum(A,i,j)(i,j<=n)，返回数列A中下标在[i,j]里的和。

区间染色：HDU-1556 Color the ball

N个气球排成一排，从左到右依次编号为1,2,3....N.每次给定2个整数a b(a <= b)，lele骑上电单车从气球a开始到气球b依次给每个气球涂一次颜色。但是N次以后lele已经忘记了第i个气球已经涂过几次颜色了，你能帮他算出每个气球被涂过几次颜色吗?

Input

- 每个测试实例第一行为一个整数N,(N <= 100000).接下来的N行，每行包括2个整数a b(1 <= a <= b <= N)。当N = 0，输入结束。

Output

- 每个测试实例输出一行，包括N个整数，第i个数代表第i个气球总共被涂色的次数。

Sample Input:

```
3
1 1
2 2
3 3
3
1 1
1 2
1 3
0
```

Sample Output:

```
1 1 1
3 2 1
```

- 方法一： 直接搜索

读入数据后，对每一个操作，采用枚举方法直接对每个点进行标记，最后统计每个点的标记次数，则时间复杂度为O(n^2)。

```
#include<bits/stdc++.h>
const int MAX=100010;
int a[MAX];
int main(){
        int n,L,R,ans;
        while(scanf(“%d”, &n)!=EOF && n){
           for(int i=1;i<=n;i++){
              scanf("%d%d",&L,&R);
              for(int i=L;i<=R;i++) a[i]++;       //对每一个区间进行标记
           }
         for(int i=1;i<=n; i++) printf("%d\n", a[i],’ ‘);
         }
        return 0;
}
```

方法二：差分数组（前缀和）

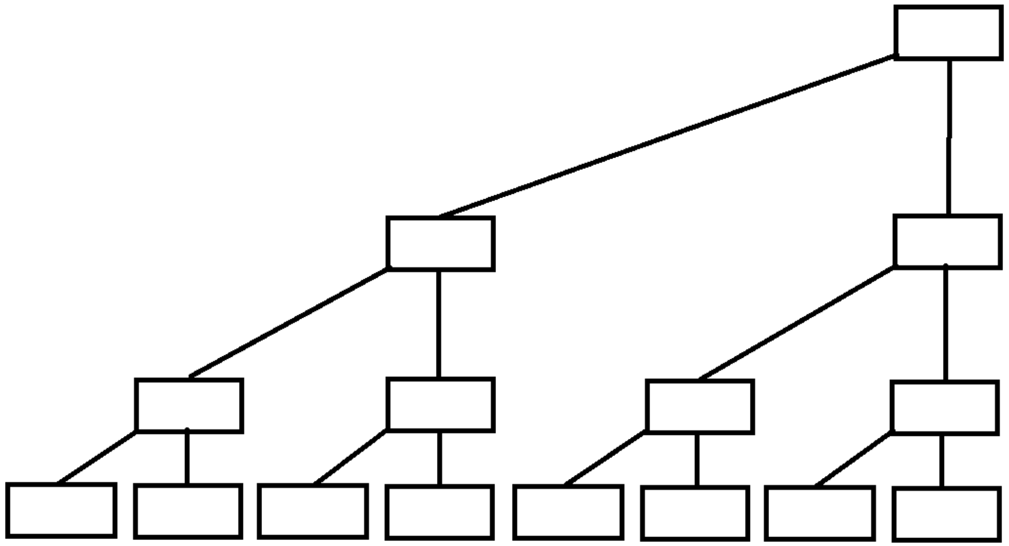
- 对于每次增加的区间，不是要区间里所有的都加一，仅仅左端点+1，右端点右边一个-1就够了，最后要求的就只是这个点的前缀和而已。时间复杂度O(n)
- 例如：对[x,y]区间的修改，表示[x,y]中的值都要加1.

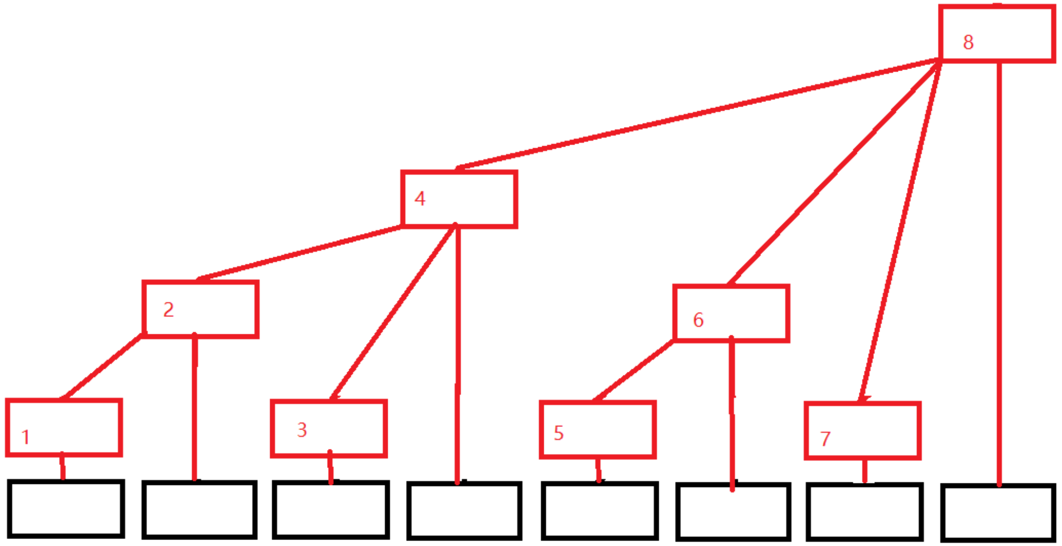
s[i]表示前i个数的和，则a[i]= s[i]-s[i-1]

如果我们只修改两个端点：x和y的值，比如给x端点加1，a[x]++，则[x,y]区间的前缀和都加了1，为了保证[y+1,n]端点的前缀和不加1，需要对y+1的端点减去1，即a[y+1]--

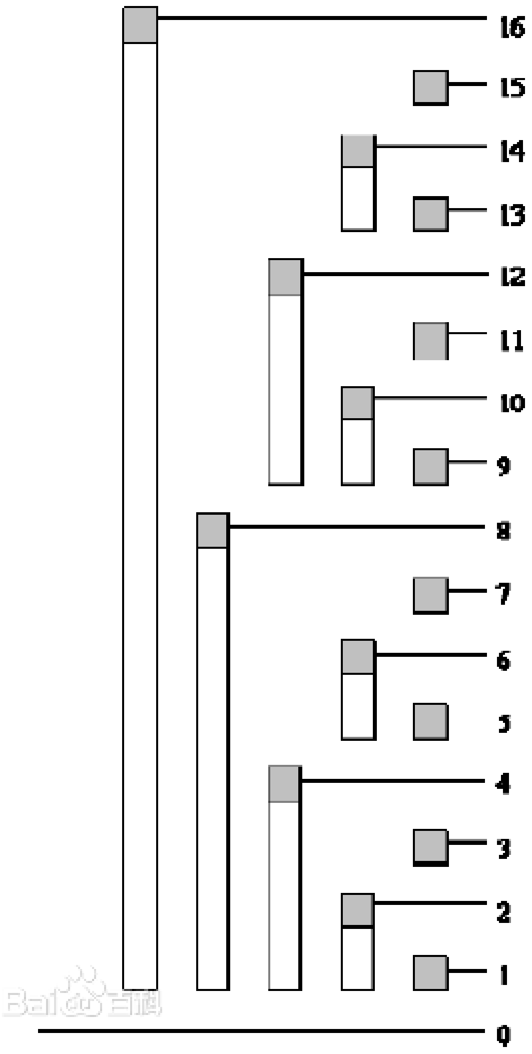
```
#include<bits/stdc++.h>
int a[100010];
int main(){
    int n, x, y;
    while(scanf("%d",&n)!=EOF && n)  {
        memset(a, 0, sizeof(a));
        for(int i = 1; i <= n; ++i)   {
            scanf("%d%d",&x,&y);
            ++a[x];
            --a[y+1];
        }
        for(int i = 1; i <= n; ++i)  {
            a[i] += a[i-1];
            printf("%d%c", a[i], i == n ? '\n' : ' ');
        }
    }
    return 0;
}
```

方法三：树状数组





- 线段树：每条线段分成两段
- 树状数组：底层数组a，中间数组C，对比线段树，少了些中间节点。



其中，a数组与C数组之间的关系，可以发现这些规律:

C[1]=a[1]

C[2]=a[1]+a[2]

C[3]=a[3]

C[4]=a[1]+a[2]+a[3]+a[4]

C[5]=a[5]

C[6]=a[5]+a[6]

C[7]=a[7]

C[8]=a[1]+a[2]+a[3]+a[4]+a[5]+a[6]+a[7]+a[8]

……

$C[2^n]=a[1]+a[2]+....+a[2^n]$

可以发现具有如下规律：

$C[i]=a[i-2^k+1]+\cdots+a[i]$，其中k为i在二进制下末尾0的个数

lowbit：计算一个数的最低位有多少个连续0

计算C[x]对应的**2^k**，lowbit公式如下：

lowbit(x)=x and (x xor (x-1)) = 2^k

例如, 100101011010000　k =4

```
     100101011010000          100101011010000
xor  100101011001111      and 000000000011111
---------------------------------------------
     000000000011111          000000000010000
```

```
int lowbit(int x) {   return x & (x^(x-1); }
```

中国计算机学会
China Computer Federation

另一个结论：$2^k = i\&(-i)$

证明：

- 负数是以补码存储的，正数的就是本身，负数的补码是对各位取反加1，0的补码为0
- 当x为0时，即 0 & 0，结果为0；
- 当x为奇数时，最后一个比特位为1，取反加1没有进位，故x和-x除最后一位外前面的位正好相反，按位与结果为0，最终结果为1。3&-3=011&101=1。
- 当x为偶数，可以写作$x= y*2^k$，y为奇数，y的最低位为1。实际上就是把x用一个奇数左移k位来表示。这时，x的二进制表示最右边有k个0，从右往左第k+1位为1。当对x取反加1，最右边的k位变成了0，第k+1位因为进位的关系变成了1，左边的位因为没有进位，正好和x原来对应的位上的值相反。二者按位与，得到：第k+1位上为1，左边右边都为0。结果为2^k。例如，$x=24=3*2^3$, $x\&-x=011000\&101000=1000=2^3$

  总结：x&(-x)，当x为0时结果为0；x为奇数时，结果为1；x为偶数时，结果为x中2的最大次方的因子。

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

查询：按公式修改每一段的数值

公式如下：

$C_1=x$

$C_{i+1}=C_i+lowbit(P_i)$

```
void update(int x,int e)
{
        while(x>0){
                c[x]+=e;
                x-=lowbit(x);
        }
}
```

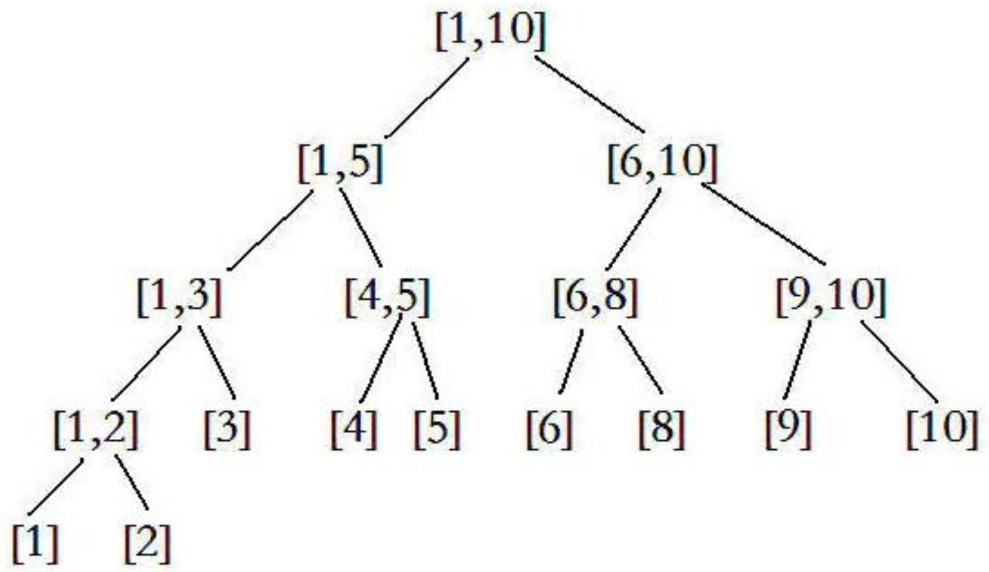
查询：按公式累加每一段的数值

```
int query(int x)
{
    int res = 0;
    while (x){
        res += c[x];
        x -= lowbit(x);
    }
    return res;
}
```

```
int main(){
    while (~scanf("%d", &n) && n) {
        memset(c, 0, sizeof(c));
        for (int i = 1; i <= n; i++) {
            int from, to;
            scanf("%d%d", &from, &to);
            update(from, 1);          //区间起点加1
            update(to + 1, -1);       //区间终点后一个位置减1
        }
        for (int i = 1; i <= n - 1; i++)
            printf("%d%c", query(i), i == n ? '\n' : ' ');
    }
    return 0;
}
```

线段树的修改——单点修改

- 修改某一个元素：对某个元素的修改，即修改某个叶子节点的值，那么只要从该节点开始，顺着子树一直找到根节点，然后更新每个节点的值即可。修改时间复杂度为O(logN)。

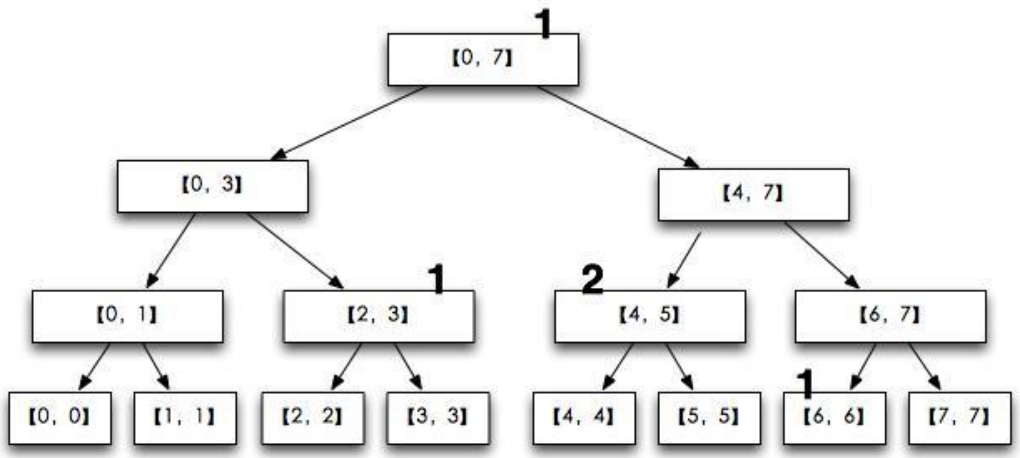


线段树的修改——单点修改

```
void update(int k,int l, int r, int x,int v)//x为原序列位置，v为要修改的值
{
    if(r<x||l>x)return; //当前区间与原序列位置完全无交集
    if(l==r&&l==x)//当前节点为叶子结点
    {
        tree[k].key=v;//修改叶子结点
        return;
    }
    int mid=(l+r)/2;
    update(k*2,l,mid,x,v);
    update(k*2+1,mid+1,r,x,v);
    tree[k].key=min(tree[k*2].key,tree[k*2+1].key);
}
```

线段树的修改——区间修改

- 如果我们想改变一个区间里的所有值怎么办：比如我想让编号为2-5的节点的值都加2，线段树维护的是区间和。我们难道要访问到所有的子节点然后修改？O(n)的复杂度显然是我们接受不了的，那么我们如何解决呢？
- 答案就是打上**懒标记**，类似于访问操作，我们在根节点不断向下访问，直到访问区间完全被要求区间所包含，然后打上标记，不改变本节点的值，然后递归***改变祖先节点的值。***
- ***注意：***因为该节点的值只要不被访问就不用去掉标记和将标记下移，但是祖先节点的值必须要改变！



区间和

给出一个序列，第一行输入两个数n,m分别代表数字个数和操作个数(n<=100000, m<=100000)

第二行输入n个数，编号从1到n，分别代表这n个数的值

接下来m行有m个操作，格式如下：

1 x y k 将区间[x,y]内每个数加上k

2 x y 询问区间[x,y]内每个数的和

输入样例

```
5 5
1 5 4 2 3
2 2 4
1 2 3 2
2 3 4
1 1 5 1
2 1 4
```

输出样例

```
11
8
20
```

中国计算机学会 China Computer Federation

建树

```
#define N 200100 //开两倍空间
struct node {
    int l, r, lc, rc; //左端点，右端点，左儿子，右儿子
    long long v, tag; //区间和，懒标记
} tr[N];
void pushup(int lc, int rc, int rt) { //区间和上浮
    tr[rt].v = tr[lc].v + tr[rc].v;
}
void build(int l, int r, int & rt) {
    rt = ++tot;
    tr[rt].l = l, tr[rt].r = r;          //记录该节点区间端点
    if (l == r) {                        //叶子结点
        scanf("%lld", &tr[rt].v);
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, tr[rt].lc);            //建左子树
    build(mid + 1, r, tr[rt].rc);        //建右子树
    pushup(tr[rt].lc, tr[rt].rc, rt); //上浮
}
```

修改

```
void pushdown(int lc, int rc, int rt) { //懒标记下沉
    tr[lc].tag += tr[rt].tag; //注意是加上父节点的懒标记
    tr[rc].tag += tr[rt].tag;
    tr[lc].v += (tr[lc].r - tr[lc].l + 1) * tr[rt].tag; //修改区间和
    tr[rc].v += (tr[rc].r - tr[rc].l + 1) * tr[rt].tag;
    tr[rt].tag = 0; //清零，防止干扰下次操作
}
void update(int l, int r, int rt) {
    if (x <= l && y >= r) { //区间覆盖，[x,y]表示修改区间
        tr[rt].tag += k; //修改懒标记
        tr[rt].v += (r - l + 1) * k; //区间和加上区间长度与k的乘积
        return;
    }
    pushdown(tr[rt].lc, tr[rt].rc, rt); //下沉
    int mid = (l + r) >> 1;
    if (x <= mid)   update(l, mid, tr[rt].lc); //修改左子树
    if (y > mid)   update(mid + 1, r, tr[rt].rc); //修改右子树
    pushup(tr[rt].lc, tr[rt].rc, rt); //上浮
}
```

查询

```
long long getans(int l, int r, int rt) {
    if (x <= l && y >= r)    //[x,y]为查询区间
        return tr[rt].v;
    pushdown(tr[rt].lc, tr[rt].rc, rt); //下沉
    long long ans = 0;
    int mid = (l + r) >> 1;
    if (x <= mid)
        ans += getans(l, mid, tr[rt].lc);
    if (y > mid)
        ans += getans(mid + 1, r, tr[rt].rc);
    return ans;
}
```

```
int main() {
    scanf("%d %d", &n, &m);
    build(1, n, root);
    while (m--) {
        scanf("%d %d %d", &p, &x, &y);
        if (p == 1) {
            scanf("%lld", &k);
            update(1, n, 1);
        } else   printf("%lld\n", getans(1, n, 1));
    }
    return 0;
}
```

线段树的维护

- 修改
  - 对元素进行修改
  - 对区间进行修改
    - 统一加上一个数
    - 设置为同一个数
- 查询
  - 区间的和
  - 区间的最大值
  - 区间的最小值
  - 其它

线段树的维护方法

线段树上的参数通常有两种维护方法：

（1）一类参数表达了结点的性质，通常具有树型的递推性质，可以从下**向上递推**计算；如求和sum,求最大最小值max/min）

（2）一类参数表达了子树的性质，维护的时候可以先打上懒标记，在需要进一步访问其子结点的时候从上**向下传递**；如修改为某个数delta, 增加某个值en）

应用1 票务系统

某次列车途经C个城市，城市编号依次为1到C，列车上共有S个座位，每一个售票申请包含三个参数，分别用O、D、N表示，O为起始站，D为目的地站，N为车票张数，售票系统对该售票申请作出受理或不受理的决定。只有在从O到D的区段内列车上都有N个或N个以上的空座位时该售票申请才被受理。1<=C<=60000，1<=S<=60000，1<=R<=60000，C为城市个数，S为列车上的座位数，R为所有售票申请总数。

输入输出样例

输入：

4 6 4

1 4 2

1 3 2

2 4 3

1 2 3

输出：

YES

YES

NO

NO

分析

可以把所有的车站顺次放在一个数轴上，在数轴上建立线段树，在线段树上维护区间的delta与max，每次判断一个售票申请是否可行就是查询区间上的最大值；每个插入一个售票请求，就是给一个区间上所有的元素加上购票数。

参考程序： http://www.mamicode.com/info-detail-1735589.html

应用2： Fast Matrix Operations

有一个矩阵，给你三个操作

1.给一个子矩阵加一个值

2.将一个子矩阵的每个元素赋成一个值

3.查询一个子矩阵的所有元素的和，最小值，最大值

- 矩阵元素不超过10^6，矩阵和不超过10^9
- 操作数m不超过20, 000,矩阵不超过20行

输入输出样例

输入样例

4 4 8
1 1 2 4 4 5
3 2 1 4 4
1 1 1 3 4 2
3 1 2 4 4
3 1 1 3 4
2 2 1 4 4 2
3 1 2 4 4
1 1 1 4 3 3

输出样例

45 0 5
78 5 7
69 2 7
39 2 7

0 0 0 0	0 5 5 5	2 7 7 7	2 7 7 7	5 10 10 7
0 0 0 0	0 5 5 5	2 7 7 7	2 2 2 2	5 5 5 2
0 0 0 0	0 5 5 5	2 7 7 7	2 2 2 2	5 5 5 2
0 0 0 0	0 5 5 5	0 5 5 5	2 2 2 2	5 5 5 2

方法1

- 由于行数不超过20，用不超过20个线段树存下每一行的数字，同时线段树维护一个mul乘数，add加数，sum总和，min最小值，max最大值。设置成v，可以看成是*0+v，加上v可以看成*1+v;
- **注意赋值的标记优先级比加标记的优先级要高。**
- **每次赋值的时候，要把加标记清空。**
- 相当于加标记只存在于赋值以后。
- 参考程序:
- https://www.cnblogs.com/chujian123/p/3840487.html

方法2

- 因为总元素个数不超过10^6，而且更新是对于连续的行进行更新，所以我们可以把矩阵转化为一个一元组，通过下一行拼接在上一行的末尾，那么在更新与查询的时候只要对相应的区间进行操作即可。

- 参考程序：https://blog.csdn.net/libin56842/article/details/46489841

中国计算机学会
China Computer Federation

应用3： Snake (SGU 128)

- 在平面上有N个点，现在要求一些线段，使其满足以下要求：
  a． 这些线段必须闭合
  b． 线段的端点只能是这N个点
  c． 交于一点的两条线段成90度角
  d． 线段都必须平行于坐标轴
  e． 所有线段除在这N个点外不自交
  f． 所有线段的长度之和必须最短

这些条件说明，我们所要构造的是一个边平行于坐标轴的多边形，且此多边形唯一。

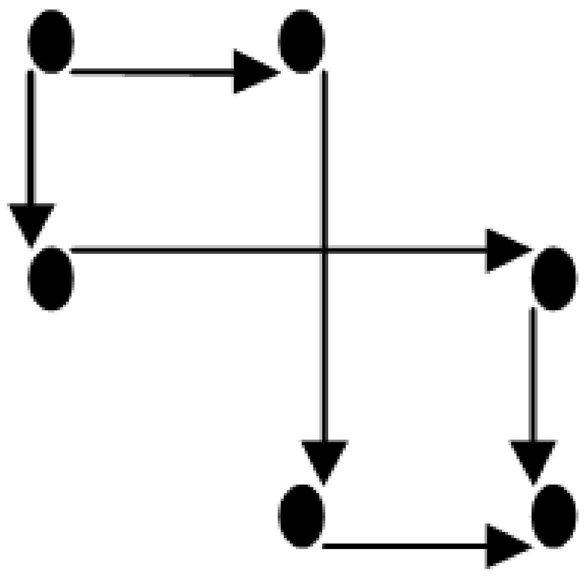
输入：

N (4 <= N <= 10000)和N个坐标 xi， yi (1 <= i <= N).

输出：

如果存在这样的线段，则输出最小长度，否则输出0。

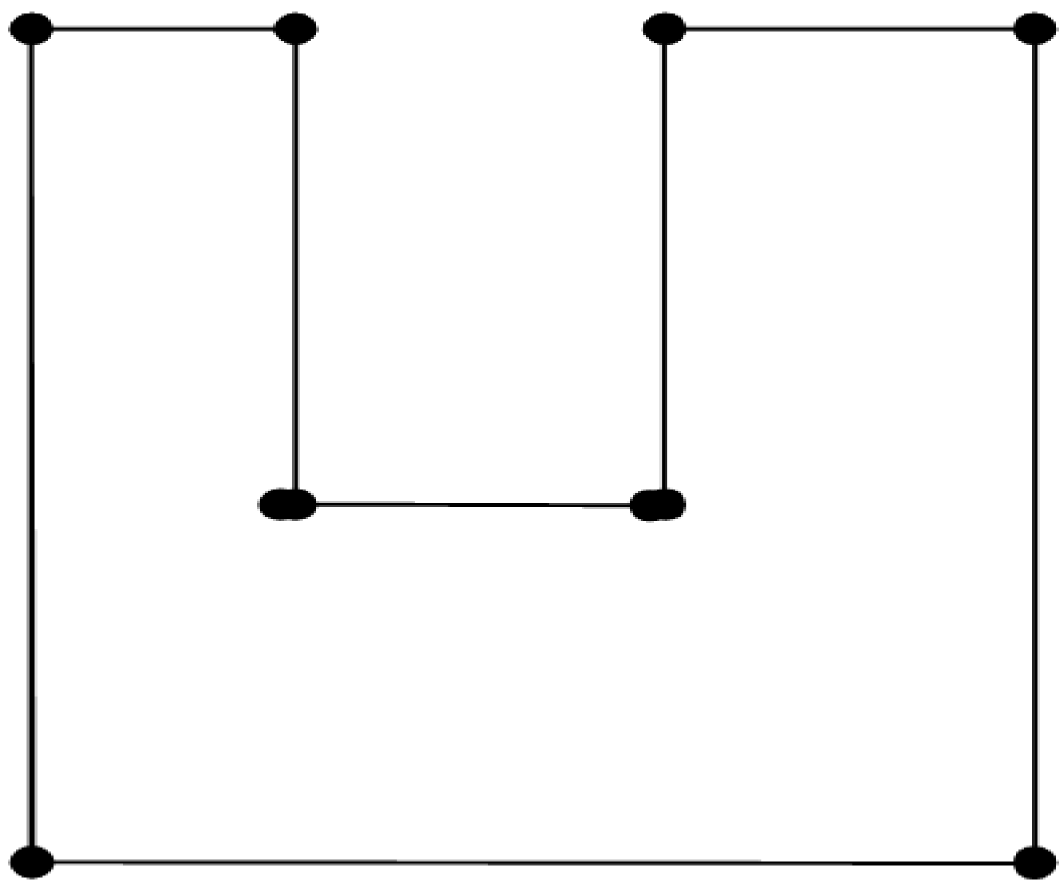
注意



自交—不合法

分析

- N个点都要用
- 所有线段都要和坐标轴平行,所以每个点只能与上下左右四个点相连。
- 一个点相连的两条线段成 90 度，因此，**每个顶点必须与一条平行于 X 轴和一条平行于 Y 轴的线段相连。**



分析

- 鉴于以上性质，假设在同一水平线(y相同)上有6个点，我们应该如何连接？
- 对于同一垂直线(x相同)上的点，我们的处理方法是一样的。
- 如果同一水平或者垂直线上的点的数量为奇数，那么此数据不合法，直接输出0即可。

深搜找出那个可行的连接方案

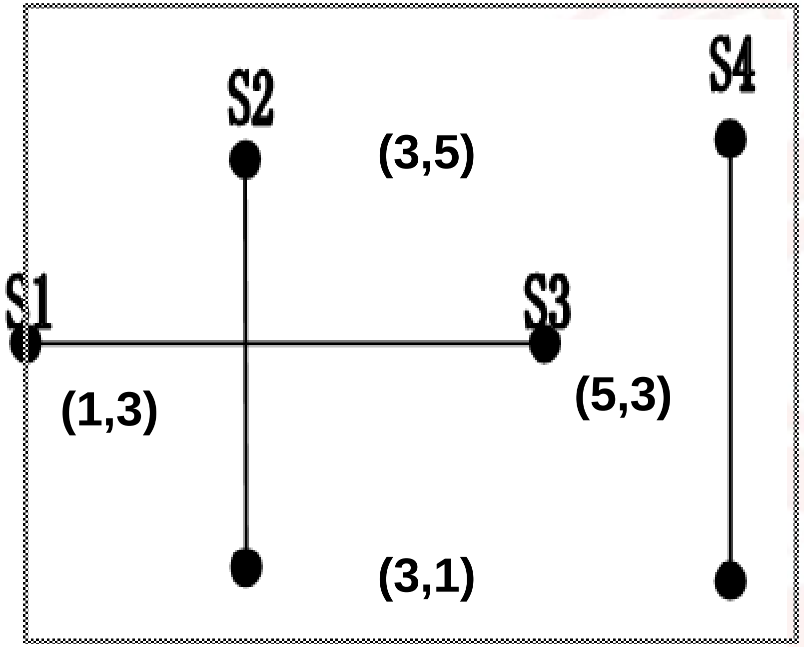
- step1：将所有的点从小到大进行排序，关键字1：x坐标，关键字2：y坐标。用快排，时间复杂度为O(nlogn)。
- step2：由题目条件知，每个点可连接的点最多有4个，就是离它最近的上下左右4个点。所以，第二步，找出每个点的上下左右四个可连接的点。因为排过顺序，所以，对于当前第i个点来说，上下连接的点如果存在，那就是i+1和i-1，左右连接的点，以关键字1为y，关键字2为x进行排序，再用O(1)的时间复杂度判断出。
- step3：深搜该图。从编号为1的点开始。dfs(当前编号，上次连接方向，已经用过多少点，长度)。

中国计算机学会
China Computer Federation

如何判断“自交”

- 首先将所有点按X坐标从小到大排序。
- 依次处理每一个“事件”。
- 何谓“事件”？
  - 当前点在一条平行于X轴的直线上，则当前点为一个事件。
  - 当前点在一条平行于Y轴的直线上，则当前点所在的线段为一个事件。

分析

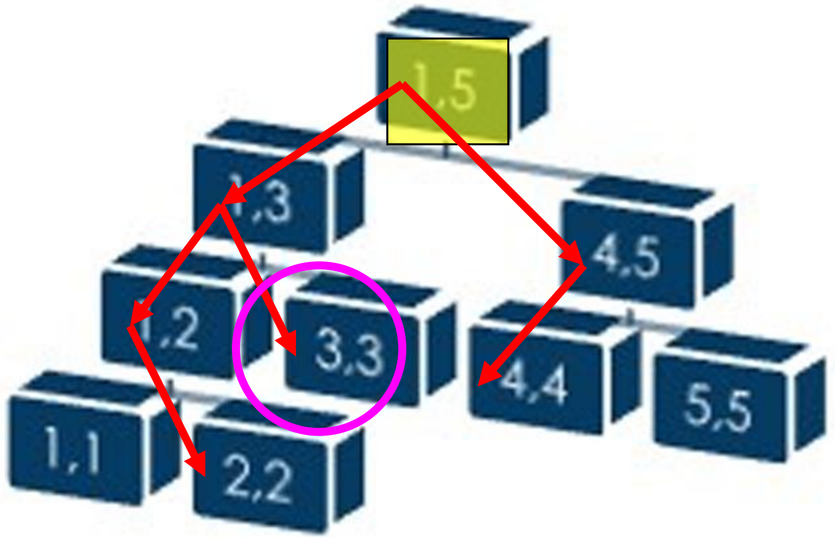


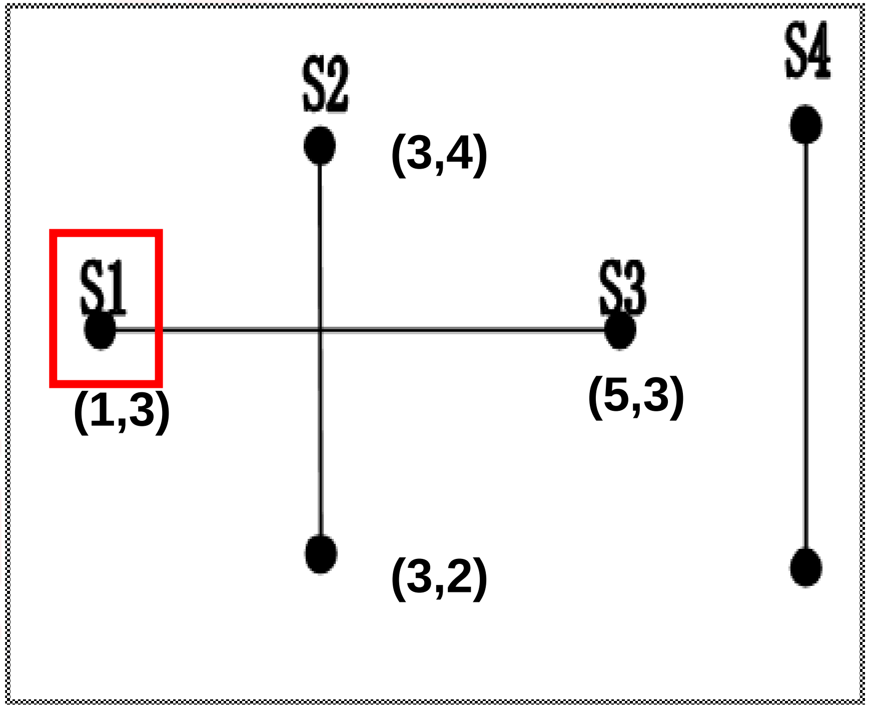
S1,S2,S3,S4分别为一个事件

- 依次扫描所有事件，如果遇到平行于 X 轴线段的左端点，则按它的 Y 坐标当成一个点，插入到表示 Y 轴区间的线段树中，如果遇到平行于 X 轴线段的右端点，则把它代表的点从线段树中删除。
- 如果遇到与 Y 轴平行的线段**L1(x,y1)-(x,y2)**，只需查询当前线段树中在区间**[y1,y2]**所包含的结点中是否有结点被覆盖，如果有，则图形不合法。

分析

此时检测到有“自交”情况，图形非法。





分析

- 此题解法基本介绍完成。至此，还有几个需要注意的地方。
- 1.连接出来的多边形有可能不相连，不相连的不合法。
- 2.线段树存储之前需要离散化，不然会爆空间。
- 参考程序:
- https://www.cnblogs.com/xuesu/p/4014989.html

应用4：矩形覆盖

在平面上有N个平行于坐标轴矩形

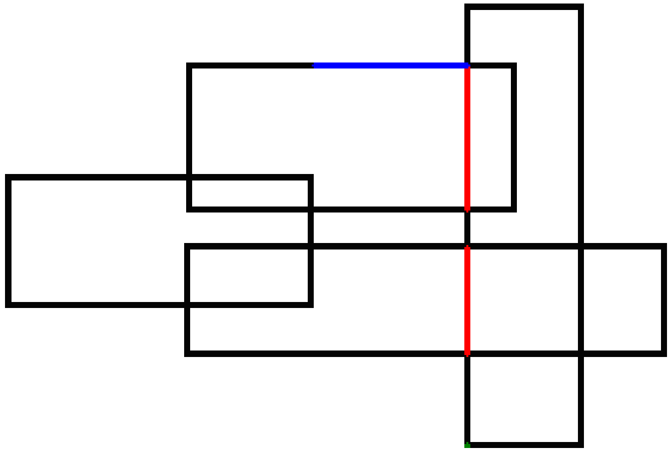
我们现在想知道，这些矩形所覆盖的面积是多少？

当然，重复覆盖只算一次

数据范围

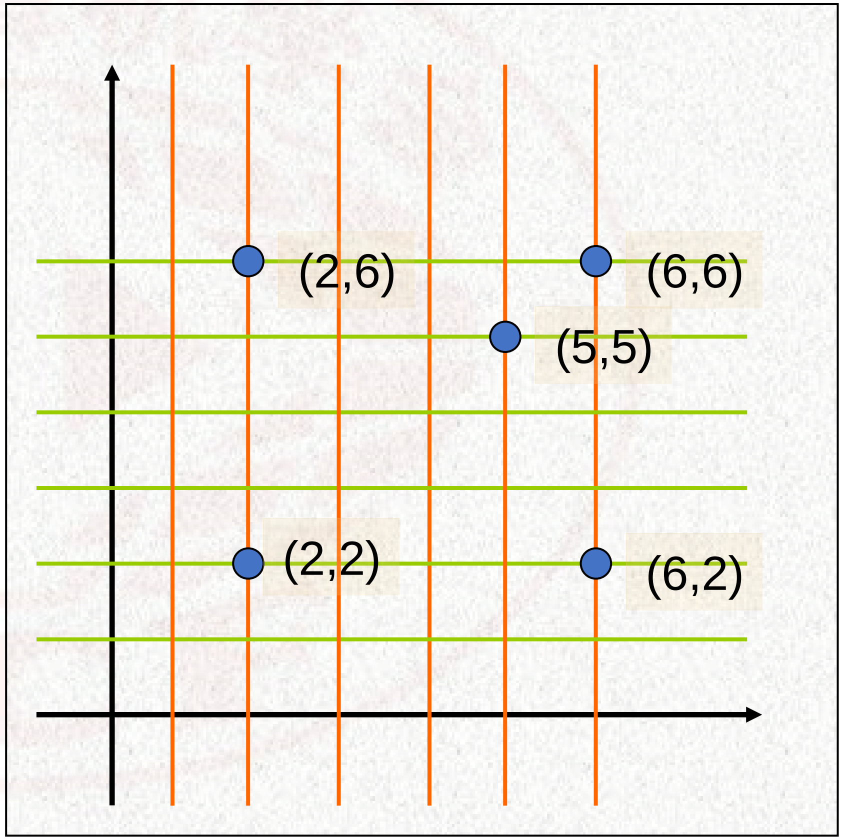
$N \leqslant 10^5$

$|坐标| \leqslant 10^9$



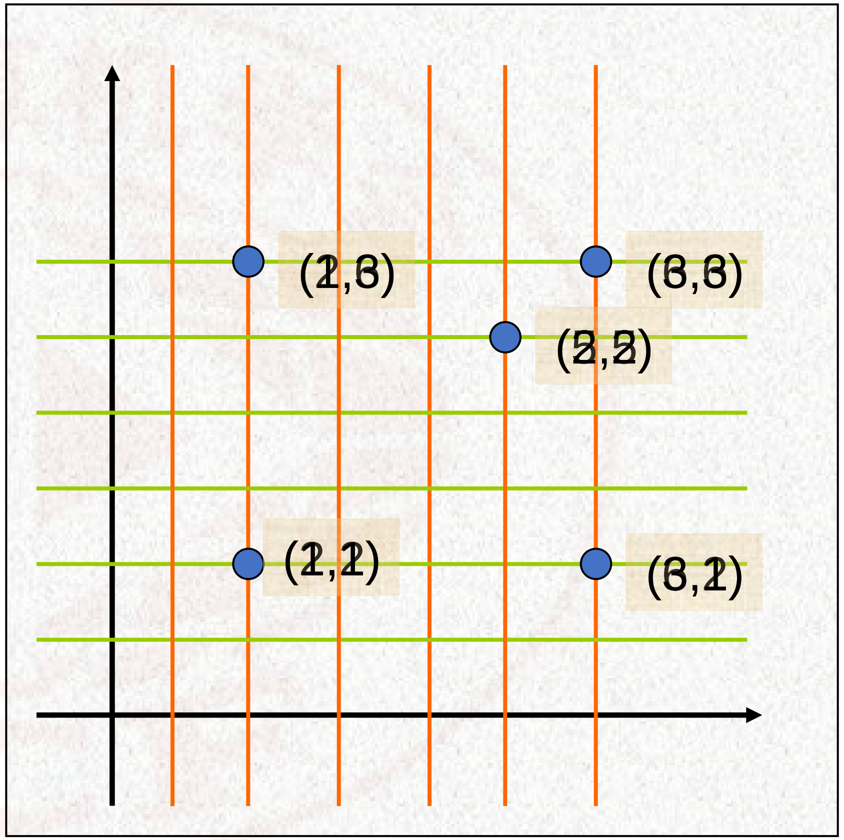
离散化

- 如右图所示，一个平面直角坐标系上有一些点，这些点的坐标已给出。
- 假设我们要对这些点或者点构成的线段进行线段树的相关操作。
- 如果坐标的跨度很大，空间可能承受不起；即使坐标跨度不大，如果这些坐标中包含小数该怎么办？



离散化

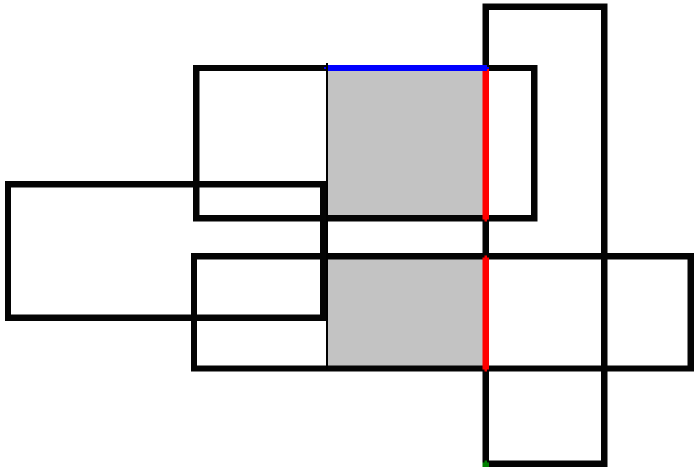
- 此时我们不妨删去一些无用的“坐标网格”。就像这样。
- 现在看来是不是清爽多了？我们还需要将这些点的坐标***重新编写***。
- 这就是离散化

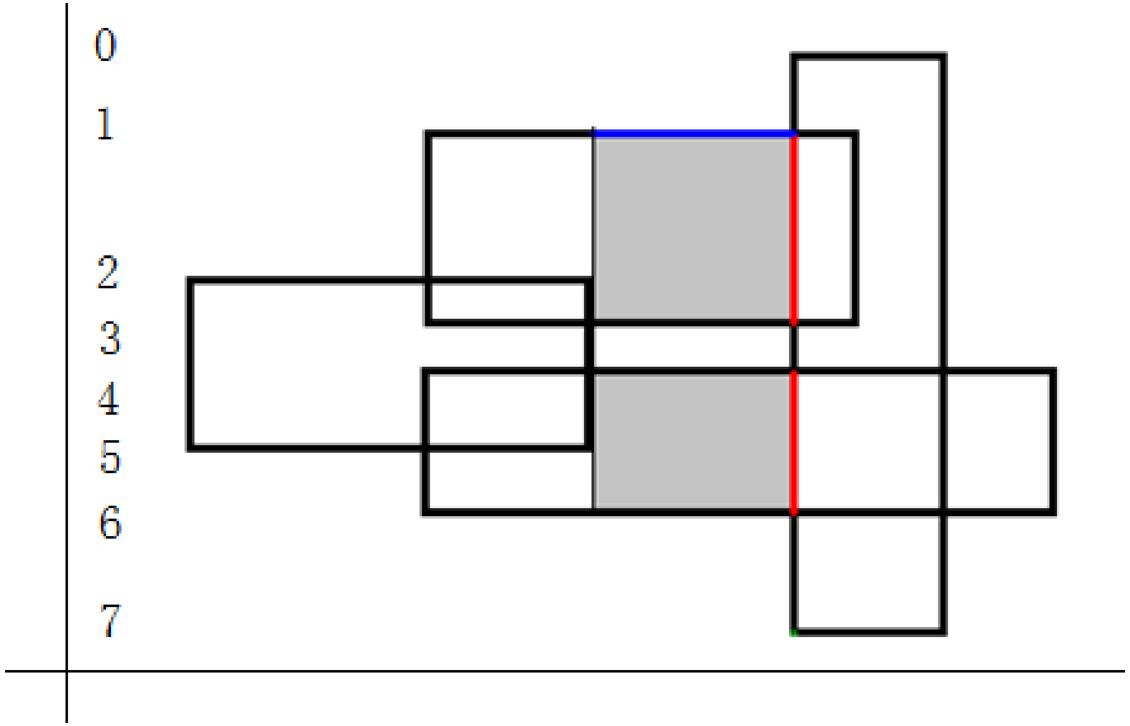


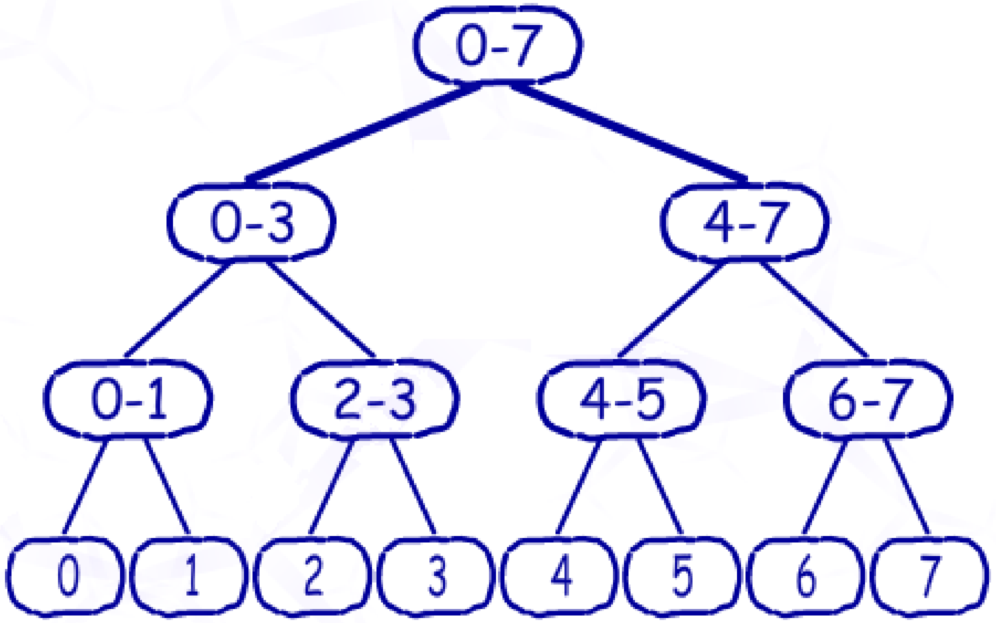
中国计算机学会
China Computer Federation

分析

- 由于坐标范围较大，我们可以采用离散化来将这些坐标离散。
- 将所有点按横坐标排序，那么，每两个相邻的横坐标之间的宽度和便是相同的
- 他们的面积可以用长*宽计算，如下图灰色部分，该面积只要知道红色部分的线段长度即可。
- 我们枚举横坐标，那么只要考虑纵坐标的线段长度了。




0
1
2
3
4
5
6
7


0-7
0-3
4-7
0-1
2-3
4-5
6-7
0
1
2
3
4
5
6
7

分析

这样一来，我们可以用线段树维护两个值A B

A表示该节点代表的纵坐标范围被线段被覆盖的层数

B表示该节点代表的纵坐标范围被线段被覆盖的长度

显然，若A>0，B为所代表范围的长度

若A=0，B为它两个孩子节点的B值之和，若为叶子节点则为0

那么，对于一个横坐标，若有矩形新加进来，则将它所覆盖的纵坐标在线段树中A值+1，若有矩形出去，则将它所覆盖的纵坐标在线段树中A值-1，再进行统计即可

参考程序：https://blog.csdn.net/u011056504/article/details/73824404

应用5 Mobiles (IOI2001)

- 在一个N*N的方格中，开始每个格子里的数都是0。现在动态地提出一些问题和修改：提问的形式是求某一个特定的子矩阵(x1,y1)-(x2,y2)中所有元素的和；修改的规则是指定某一个格子(x,y)，在(x,y)中的格子元素上加上或者减去一个特定的值A。现在要求你能对每个提问作出正确的回答。
- 1≤N≤1024,提问和修改的总数可能达到60000条。

Sample Input

```
0 4
1 1 2 3
2 0 0 2 2
1 1 1 2
1 1 2 -1
2 1 1 2 3
3
```

Sample Output

```
3
4
```

二维树状数组

- 一维树状数组维护的是区间和,最后统计的是1~x数的和,二维树状数组维护的是一个面的和,最后统计的是 (1,1)~(x,y)区域内数的和。
- add和query操作的可利用两层循环结合二分来实现。首先，对二维表格的X坐标进行二分，同时将分得的每个部分再按Y坐标进行二分，并记下最终分得的每个部分的移动电话总数。“多重二分”的结果，实际上类似于一维情况下二分的结果，也是形成一种“树”状的结构，只不过由于是多重二分，所以这棵“树”的每个节点仍然是一棵“树”。
- 修改操作：在“二分树”上进行“改动”操作，只要按照X坐标从“根”到“叶”，处理路径上每个节点，由于这些节点也是树，所以要按照Y坐标从“根”到“叶”，依次修改路径上的每个节点的数据。这样，处理每个节点的时间复杂度为$O(\log_2 n)$，每次修改需要处理的节点数为$\log_2 n$，所以每次“改动”操作的时间复杂度为$O(\log_2 n * \log_2 n)$ 。
- 查询操作：以x为标准的“二分树”上的每个节点同时也是一棵以Y坐标为标准的“二分树”，查询方法为：从“根”到“叶”查找Y，如果某步是向右子树走，则将相应左子树整个部分的移动电话总数加入Total，最后找到Y，将叶节点上的数值加入Total，时间复杂度为$O(\log_2 n * \log_2 n)$ 。

```
#include <cstdio>
#include <cstring>
using namespace std;
int n,cc[1030][1030];
void add(int x,int y,int value){
      for (int i=x;i<=n;i+=i&(-i))
        for (int j=y;j<=n;j+=j&(-j))
            cc[i][j]+=value;
}
int qurry(int x,int y){
       int ans=0,i,j;
       for (i=x;i>0;i-=i&(-i))
       for (j=y;j>0;j-=j&(-j))  ans+=cc[i][j];
      return ans;
}
```

```
int main(){
    int a,b,c,d,id;
    while (~scanf("%d",&id) && id!=3){
    switch (id){
        case 0:
            scanf("%d",&n); memset(cc,0,sizeof(cc));break;
        case 1:
             scanf("%d%d%d",&a,&b,&c); add(a+1,b+1,c);break;
        case 2:
          scanf("%d%d%d%d",&a,&b,&c,&d);
          int ans=qurry(c+1,d+1)-qurry(c+1,b)-qurry(a,d+1)+qurry(a,b);
          printf("%d\n"ans,); break;
      }
    }
}
```

参考程序：https://blog.csdn.net/blue_cuso4/article/details/70544908

应用6：HDU1823 Luck and Love

- 前段日子，枫冰叶子给Wiskey做了个征婚启事，聘礼达到500万，由于人数太多，就把统计的事情全交给了枫冰叶子，他要处理的有两类事情，一是得接受MM的报名，二是要查找符合要求的MM中缘分最高值。
- Input
- 有多组测试数据，第一个数字M，表示接下来有连续的M个操作，当M = 0时处理中止。 接下来是一个操作符C。 当操作符为‘I’时，表示有一个MM报名，后面接着一个整数，H表示身高，两个浮点数，A表示活泼度，L表示缘分值。 （100<=H<=200，0.0<=A，L<=100.0） 当操作符为‘Q’时，后面接着四个浮点数，H1，H2表示身高区间，A1，A2表示活泼度区间，输出符合身高和活泼度要求的MM中的缘分最高值。 （100<=H1，H2<=200，0.0<=A1，A2<=100.0）所有输入的浮点数，均只有一位小数。
- Output
- 对于每一次询问操作，在一行里输出缘分最高值，保留一位小数。 对查找不到的询问，输出-1。

输入输出样例

Sample Input

```
8
I 160 50.5 60.0
I 165 30.0 80.5
I 166 10.0 50.0
I 170 80.5 77.5
Q 150 166 10.0 60.0
Q 166 177 10.0 50.0
I 166 40.0 99.9
Q 166 177 10.0 50.0
0
```

Sample Output

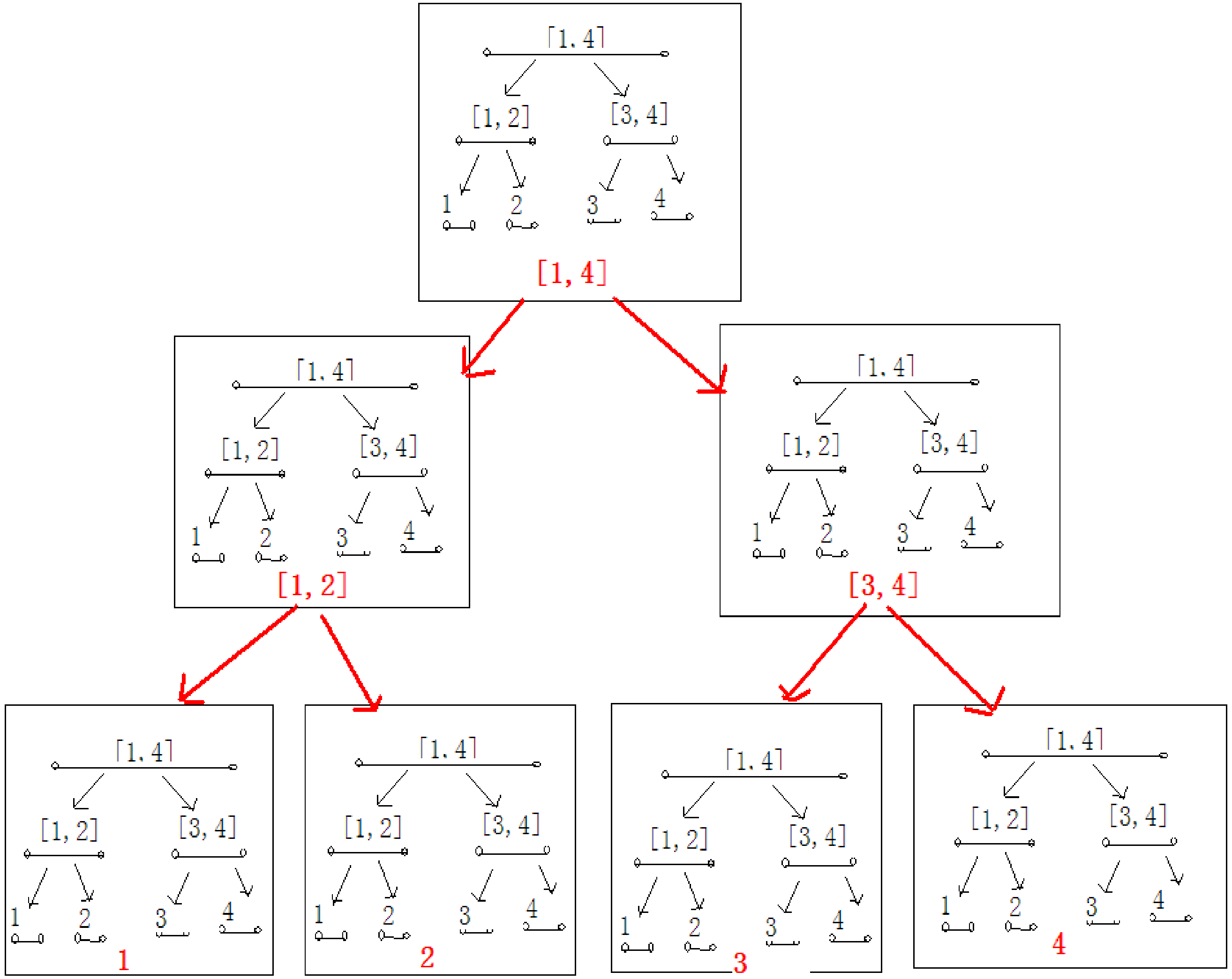
```
80.5
50.0
99.9
```

分析

- 本题与RMQ问题类似，接收报名就是序列的插入，查找符合要求的缘分最高值，就是对序列的查询。只不过这里的序列有两维，身高和活泼度。
- 因此，我们可以建立一个二维线段树，以身高为第一维，活泼度为第二维，需要对二维线段树进行单点修改和区间查询操作。
- 由于身高是100~200，缘分值为0.0~100.0，为了存储方便，我们都将身高减去100，缘分值乘以10，变成整数。
- 数据结构

```
int maxn[105*4][1005*4];
```

二维线段树



二维线段树的理解

母树保存x轴上面的信息；
子树保存当y轴上的信息；
所以，我们每当对母树进行更新或者建立的时候，都要对母树所对应的子树进行所有的建立；
相当于，先考虑x轴的范围，然后当x的范围一定时，在考虑对于x范围内的y值范围！

二维线段树

- 线段树的深度不超过logL。
- 线段树把区间上的任意一条线段都分成不超过2*logL条线段。
- 这些结论为线段树能在O(logL)的时间内完成一条线段的插入、删除、查找等工作，提供了理论依据，而如果仍用这种树结构表示二维空间，不能保证第2条，所以会退化。
- 因此，要实现平面操作，需要用二维线段树或二维树状数组，时间复杂度为$O(\log^2 n)$。

单点修改

```
/*更新第二维，即活泼度，fk为第一维的节点下标*/
void update1(int fk,int l,int r,int pos,int k,int v){
     if(l==r) {
        maxn[fk][k]=max(maxn[fk][k],v);
        return;
    }
    int mid=(l+r)<<1;
    if(pos<=mid) update1(fk,l,mid,pos,lson,v);
    else update1(fk,mid+1,r,pos,rson,v);
     /*更新父节点区间*/
    maxn[fk][k]=max(maxn[fk][lson],maxn[fk][rson]);
}
```

```
/*更新第一维，pos1为要插入的身高，pos2为活泼度，v为缘分值*/
void update2(int k,int l,int r,int pos1,int pos2,int )
{
    /*更新第二维*/
    update1(k,1,1000,pos2,1,v);
    if(l==r)  return;
    int mid=(l+r)<<1;
    if(pos1<=M)
           update2(lson,l,mid,pos1,pos2,v);
    else
        update2(rson,mid+1,r,pos1,pos2,v);
}
```

矩阵查询

```
/*查询第二维活泼度,L1,L2,R1,R2分别表示要查找的身高和活泼度范围，que存储查询结果*/
void query1(int l,int r,int k,int fk){
    if(L2<=l&&r<=R2) {      //查到对应子区间
        que=max(que,maxn[fk][k]);
        return;
    }
    int mid=(l+r)<<1;
    if(R2<=mid)  query1(l,mid,lson,fk);
    else
      if(L2>mid) query1(mid+1,r,rson,fk);
    else {
        query1(l,mid,lson,fk);
        query1(mid+1,r,rson,fk);
    }
}
```

```
/*查询第一维身高，L1,L2,R1,R2分别表示要查找的身高和活泼度范围*/
void query2(int l,int r,int k) {
    if(L1<=l&&r<=R1){     //第一维查找完毕
        query1(1,1000,1,k);   //查找第二维
        return;
    }
    int mid=(l+r)<<1;
    if(R1<=mid)  query2(l,mid,lson);
    else
        if(L1>mid) query2(mid+1,r,rson);
    else{
        query2(l,mid,lson);
        query2(mid+1,r,rson);
    }
}
```

```
int main()
{
    int i,j,k,n;
    int a,b,v,h,h1,h2;
    double e,o,e1,e2;
    char s[10];
    while(scanf("%d",&n)!=EOF&&n)  {
        memset(maxn,-1,sizeof(maxn));//初始化为-1
        while(n--) {
            scanf("%s",s);
            if(s[0]=='I') {
                scanf("%d%lf%lf",&h,&e,&o);
                a=h-99;  //第一维整体减去为了节约空间
                b=e*10;  //第二维变成整数
                v=o*10; //变成整数
                update2(1,1,100,a,b,v);  //线段树单节点更新
            }
```

```
            else if(s[0]=='Q') {
                scanf("%d%d%lf%lf",&h1,&h2,&e1,&e2);
                if(h1>h2) swap(h1,h2);   //防止数据挖坑
                if(e1>e2)  swap(e1,e2);
                L1=h1-99;
                R1=h2-99;
                L2=e1*10;
                R2=e2*10;
                que=-1;
                query2(1,100,1);    //区间查询
                if(que==-1)  printf("-1\n");
                else  printf("%d.%d\n",que/10, que%10);
            }
        }
    }
    return 0;
}
```

参考程序：https://blog.csdn.net/qq_37497322/article/details/77165175

推荐练习题

- **Picture (USACO 5.5.1)**
- **Snake (SGU 128)**
- **Inversions(sgu 180)**
- **Towers(sgu 263)**
- **Ice-cream Tycoon(sgu 311)**
- **Cashier (noi2004)**
- **Stars(ural 1028)**