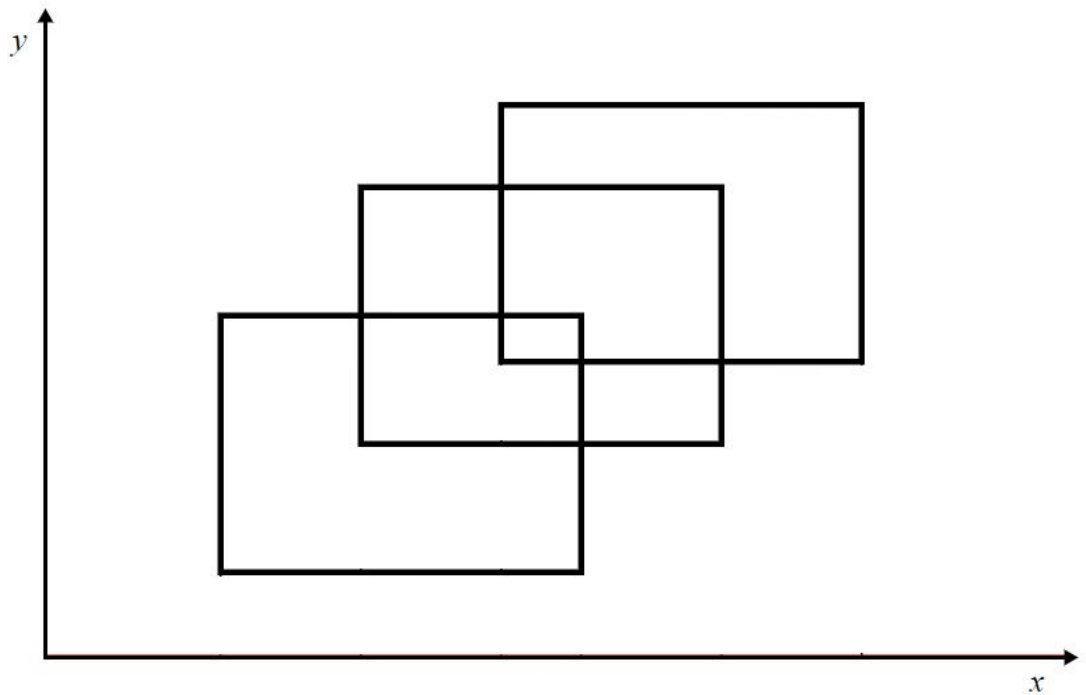
扫描线

扫描线

- 扫描线一般运用在图形上面，它和它的字面意思十分相似，就是一条线在整个图上扫来扫去，它一般被用来解决图形面积，周长，以及二维数点等问题。

Atlantis 问题

- 题意
- 在二维坐标系上，给出多个矩形的左下以及右上坐标，求出所有矩形构成的图形的面积。



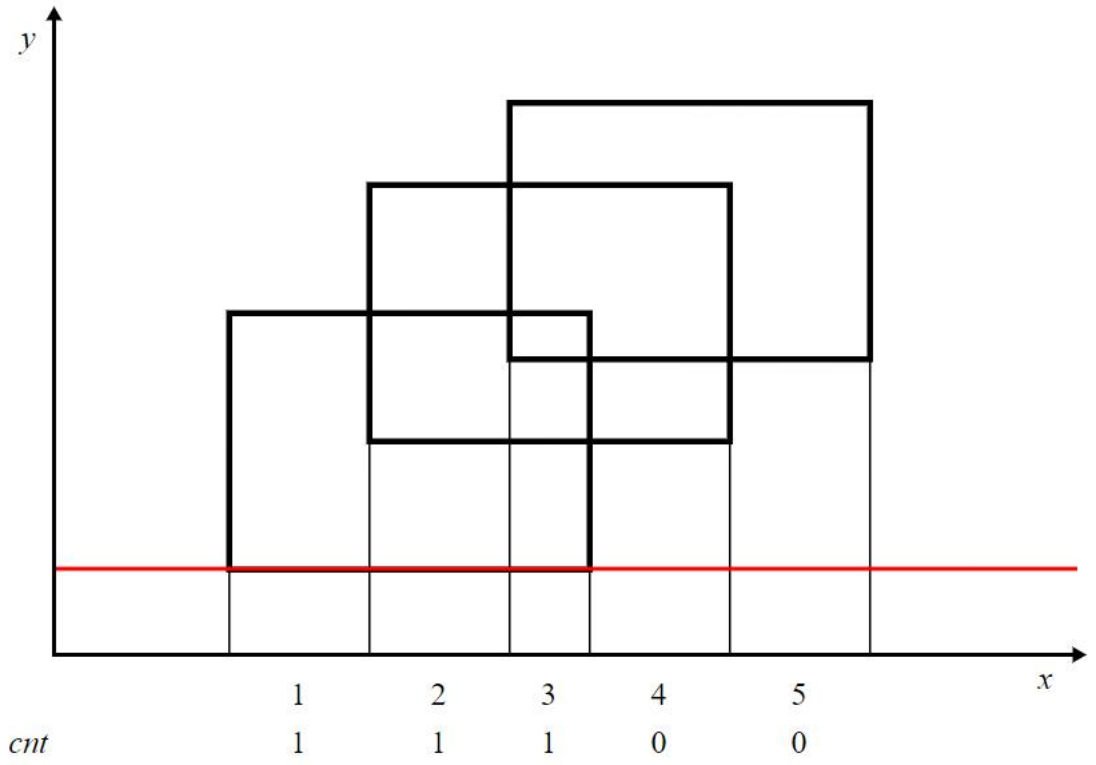
Atlantis 问题

暴力方法1：利用容斥的方法，先求出每个矩形的面积，再减去任意两个矩形的交集。求矩形的交集很花时间，需要两两配对。

暴力方法2：将平面划分为单位长度为1的方格，每读入一个矩形，就把它覆盖的方格标注为已覆盖。最后统计被覆盖的方格的个数。

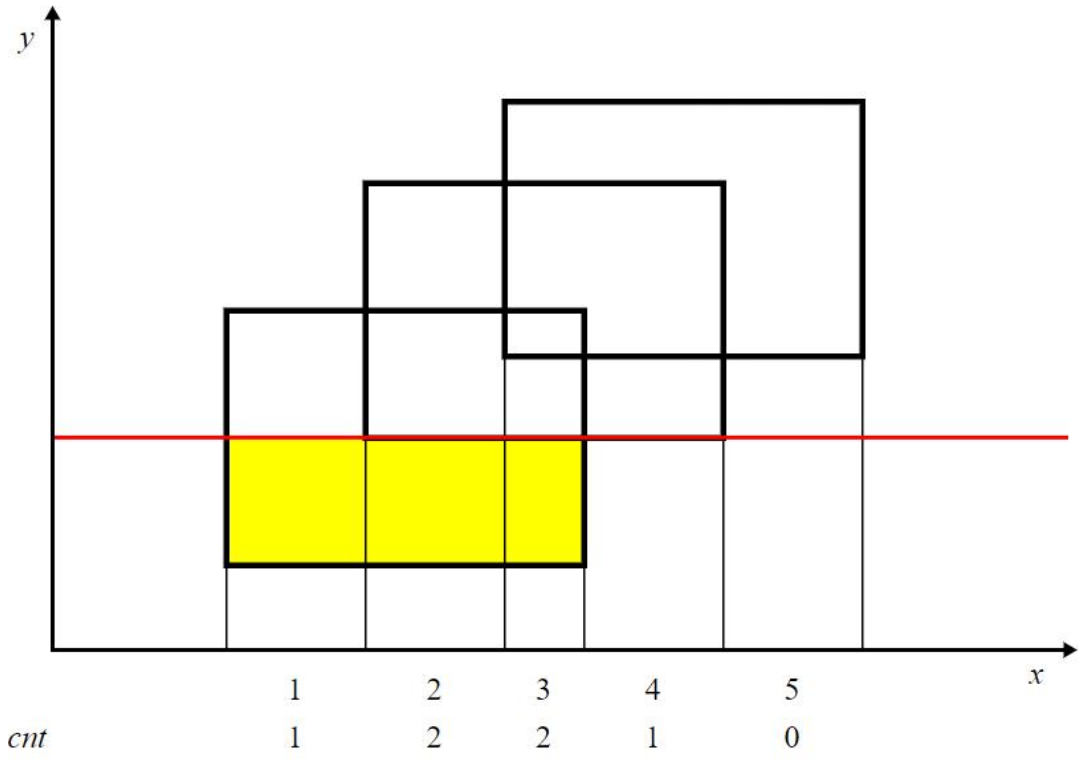
Atlantis 问题

现在假设一根线从下往上扫



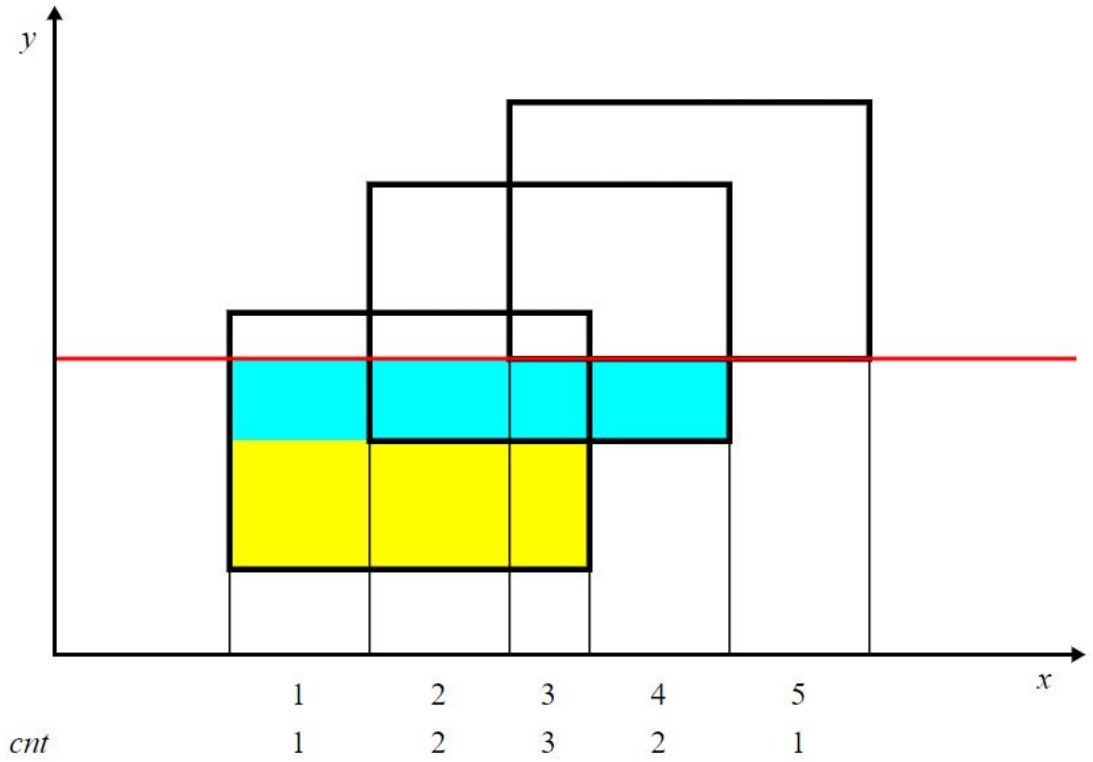
Atlantis 问题

现在假设一根线从下往上扫



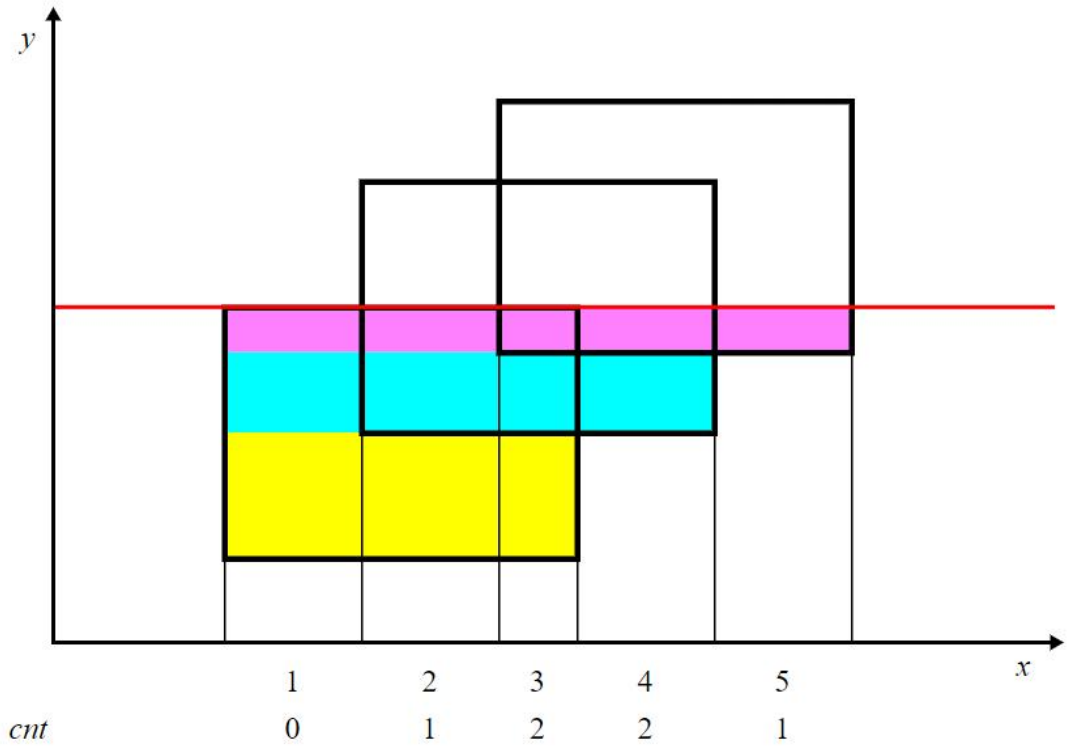
Atlantis 问题

现在假设一根线从下往上扫



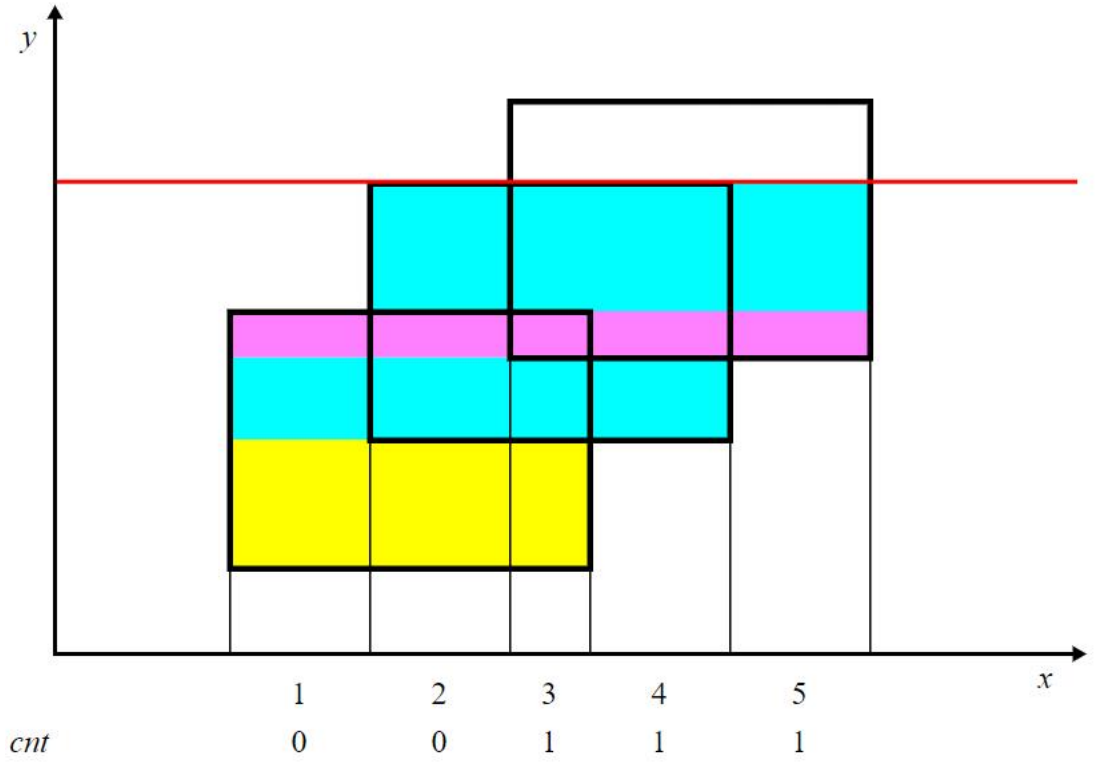
Atlantis 问题

现在假设一根线从下往上扫



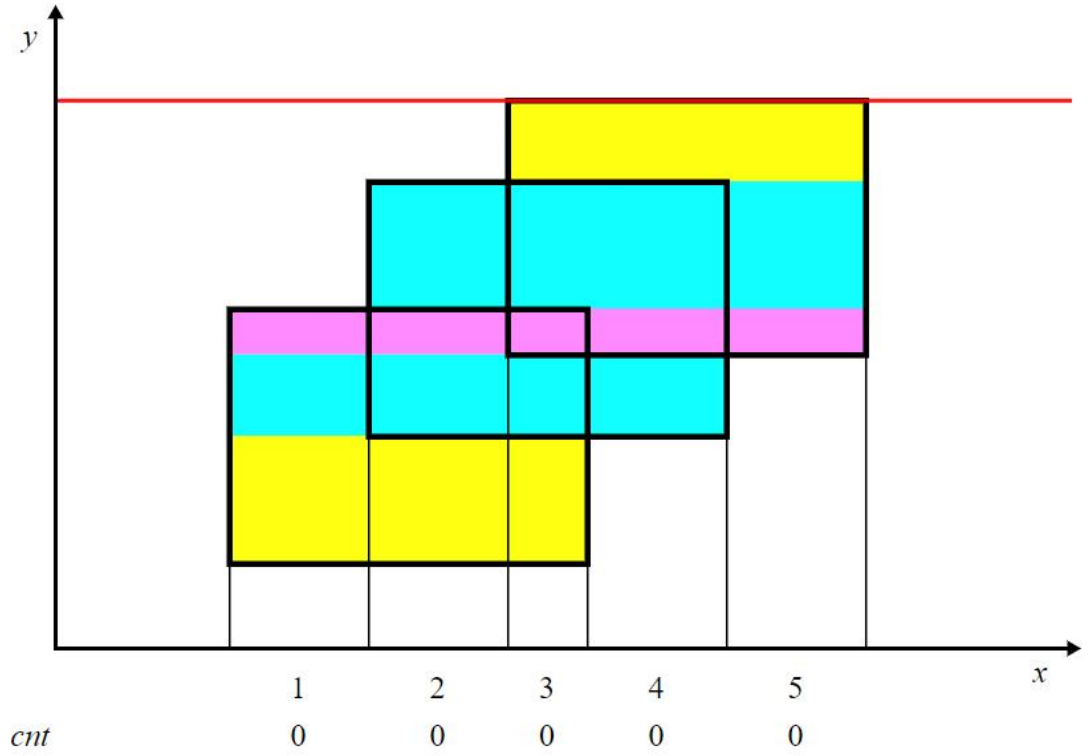
Atlantis 问题

现在假设一根线从下往上扫



Atlantis 问题

现在假设一根线从下往上扫



Atlantis 问题

我们可以把整个矩形分成5个颜色不同的小矩形。

每个矩形的面积该如何求呢？我们按照y坐标从下往上遍历每个矩形，高也就是相邻的矩形的y坐标的差值，矩形的长度是若干条线段的交集，并且在不断发生变化。

我们使用线段树维护矩形的长度。对于每个矩形上面的边为“出边”，下面的边为“入边”。

按照y坐标从下往上遍历每个矩形，入边先被扫描到，将入边加入到线段树，出边后被扫描到，将出边从线段树中删除。对于每一条入边和出边，可以标记为1和-1，代表加入和删除。

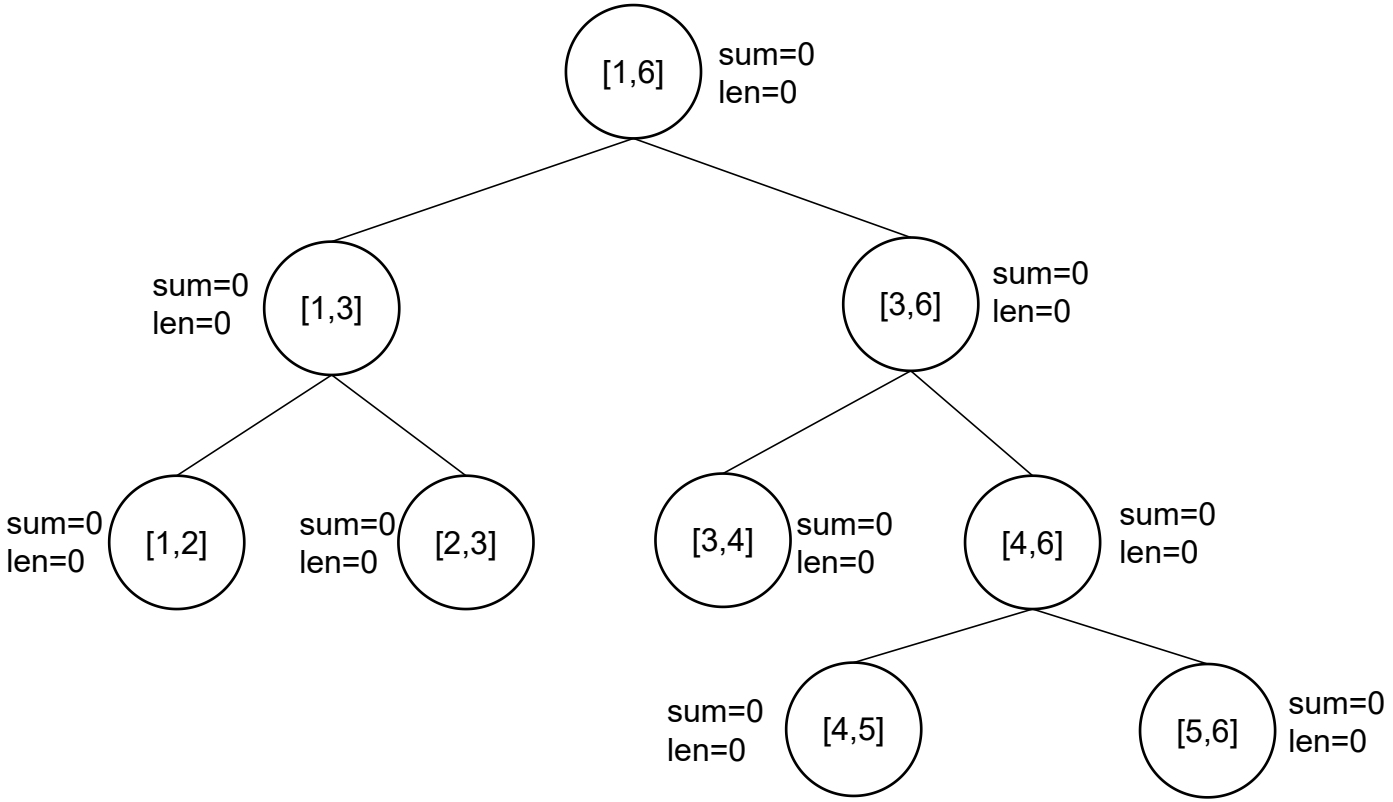
Atlantis 问题

```
for(int i=1;i<=n;i++){
        scanf("%d %d %d %d",&x1,&y1,&x2,&y2);
        a[i]=x1;a[n+i]=x2;
        line[i]=(node1){x1,x2,y1,1};
        line[n+i]=(node1){x1,x2,y2,-1};
}
n<<=1;
sort(a+1,a+n+1);//记录所有扫描线的端点信息x1,x2
sort(line+1,line+n+1,cmp);//扫描线按y坐标升序排列
int tot=unique(a+1,a+n+1)-a-1;//去除重复的x坐标
build(1,1,tot);//线段树维护x坐标
ll ans=0;
for(int i=1;i<n;i++){
        change(1,line[i].l,line[i].r,line[i].mark);
        ans+=(ll)tree[1].len*(ll)(line[i+1].h-line[i].h);
}
```

Atlantis 问题

线段树维护的东西都是点，但是我们需要维护的是区间，那么我们可以把区间下放到点上，也就是每一个叶子节点维护的是一个线段。

维护一个 sum 为当前区间被几个矩形覆盖，以及一个 len 表示当前区间被覆盖的区间长度



Atlantis 问题

建树

```
void build(int k,int l,int r){
      if(r-l==1){
            tree[k].l=a[l];tree[k].r=a[r];
            tree[k].sum=tree[k].len=0;
            return;
      }
      tree[k].l=a[l];tree[k].r=a[r];
      tree[k].sum=tree[k].len=0;
      int mid=(l+r)/2;
      build(k*2,l,mid);//左子树
      build(k*2+1,mid,r);//右子树
}
```

Atlantis 问题

我们扫描到一条线，将该条线段加入到线段树维护，修改对应区间的sum

```
void change(int k,int x,int y,int mark){//区间修改
    if(tree[k].r<=x||tree[k].l>=y)return;
     if(x<=tree[k].l&&tree[k].r<=y){//[x,y]包含当前区间[l,r],
整个区间更新
          tree[k].sum+=mark;
          pushup(k);
          return;
     }
     change(k*2,x,y,mark);//修改左区间
     change(k*2+1,x,y,mark);//修改右区间
     pushup(k);
}
```

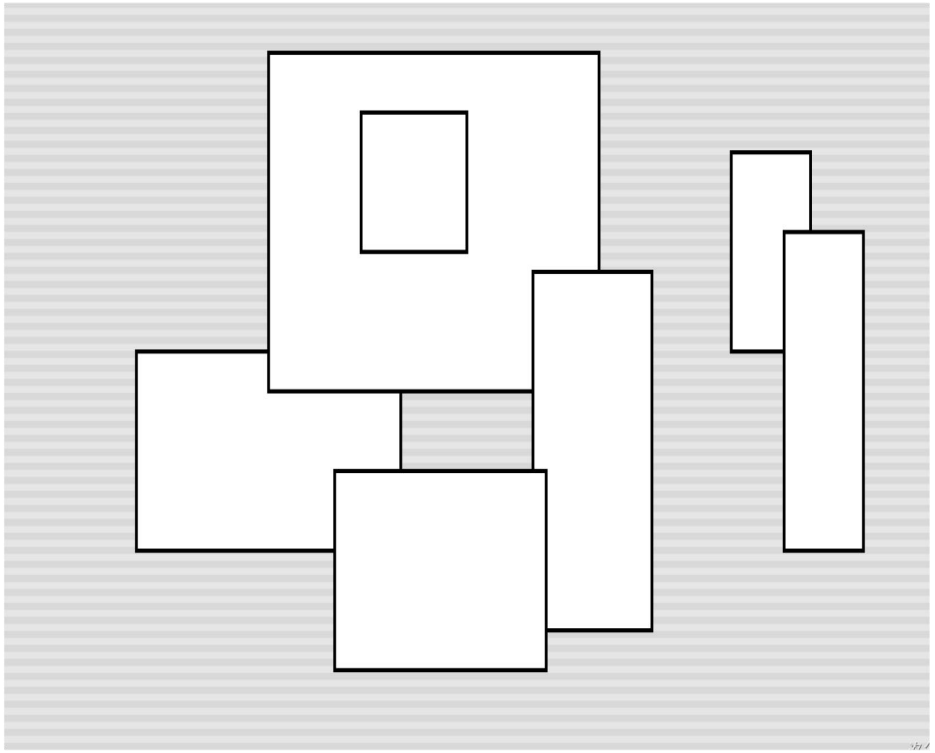
Atlantis 问题

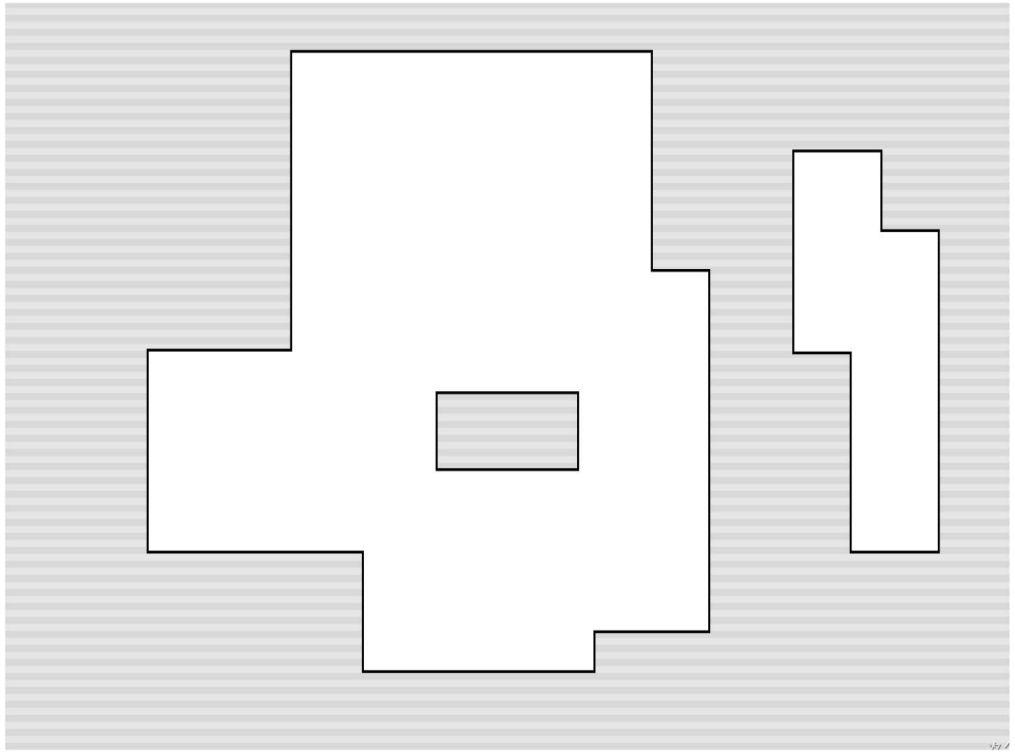
向上更新表示当前区间被覆盖的区间长度，当sum非0,则代表整个区间被覆盖，sum为0，则统计子区间被覆盖的长度

```
void pushup(int x){
	if(tree[x].sum){
		tree[x].len=tree[x].r-tree[x].l;
	}else{
		tree[x].len=tree[x<<1].len+tree[x<<1|1].len;
	}
}
```

矩形周长Picture

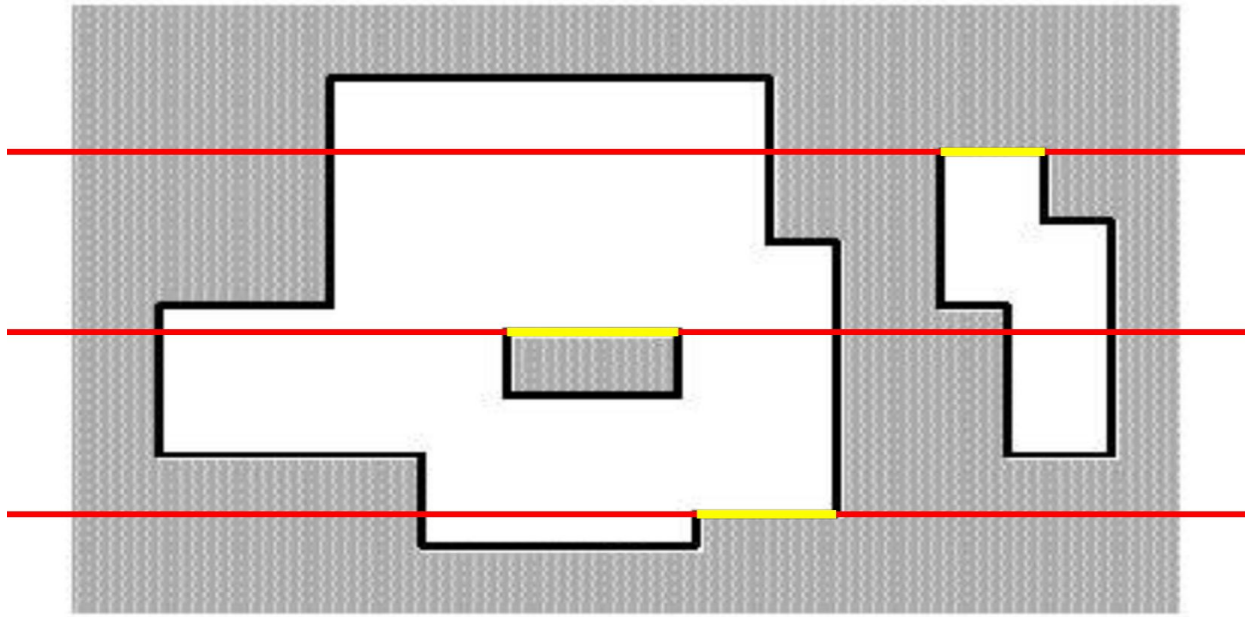
- 题意
- 在二维坐标系上，给出多个矩形的左下以及右上坐标，求出所有矩形构成的图形的周长。





矩形周长Picture

- 首先来计算横线，不难发现，我们在往上不断平移线段的时候，我们发现，当前的长，也就是和线段重合的长度，是上一次的长度与这次长度做差的绝对值！
- 为什么是绝对值？如果是一开始的时候，长会越来越大，就是这一次减去上一次的值，而到了后面，就是上一次长度减去这一次长度的值了。
- 所以我们需要加个绝对值。



矩形周长Picture

竖线的长度有两种方法：

1、可以从左到右再扫描一遍。

2、在扫描横线的过程中，考虑同时计算竖边的长度。当加入“入边”，接下来会有竖边，而当遇到“出边”，加下来不会该矩形的竖边。再考虑两个矩形有重合的部分，则竖边的共有的，而不是4条竖边。线段树除了统计横边的长度，再统一不相交区间的个数。

二维数点

- 给一个长为 n 的序列，有 m 次查询，每次查区间 [l,r] 中值在 [x,y] 内的元素个数。

二维数点

- 这个问题就叫做二维数点。我们可以发现等价于我们要查询一个二维平面上矩形内的点的数量和。
- 最简单的方法就是扫描线 + 树状数组。
- 很显然，这个问题是一个静态的二维问题，我们通过扫描线可以将静态的二维问题转换为动态的一维问题。维护动态的一维问题就可以使用数据结构，这里可以使用树状数组。
- 先将所有的询问离散化，用树状数组来维护权值，对于每次询问的 l 和 r ，我们在枚举到 l-1 的时候统计当前位于区间 [x,y] 内的数的数量 a ，继续向后枚举，枚举到 r 时统计当前位于区间 [x,y] 内的数的数量 b，b-a 即为该次询问的答案。

P2163[SHOI2007]园丁的烦恼

- 题意：
- 二维平面有n个坐标，每个坐标都表示该点处有一颗树苗，进行m次询问，每次询问给出两个坐标，表示一个矩形的。左下角和右上角，输出这个矩形中的树苗的数量，包括矩形的边界。

P2163[SHOI2007]园丁的烦恼

- 具体就是设 sum[i][j]表示以 (i,j)为右上角，以 (0,0)为左下角的矩阵的点数
- 那么对于询问以 (xa,ya)为左下角，以(xb,yb)为右上角的矩形点数
- 注意到询问区间是闭的，显然答案就是
- sum[xb][yb]-sum[xb][ya-1]-sum[xa-1][yb]+sum[xa-1][ya-1]
- 但是此题坐标系太大，所以要离散化。

P2163[SHOI2007]园丁的烦恼

- 把所有点和询问离散化（一个询问变成 4 个点 (xa−1,ya−1),(xa−1,yb),(xb,ya−1),(xb,yb)），按 x,y 为一二关键字排序，x,y相同时实点先，询问点后。
- 然后树状数组维护当前 y坐标小于等于某个数的点数，然后就可以用树状数组求 sum 了
- 扫描的时候更新答案就好了

P2163[SHOI2007]园丁的烦恼

```
struct dat{
    int x,y,id,p;//位置x,y,询问编号id,对询问的贡献系数
    inline bool operator < (const dat &tmp) const {
        if(x!=tmp.x) return x<tmp.x;
        return y!=tmp.y ? y<tmp.y : id<tmp.id;//注意x,y相同时,实点优先
    }
}d[M];//注意离散的点数比较大
```

P2163[SHOI2007]园丁的烦恼

```
for(int i=1;i<=m;i++){
    a1=read(),b1=read(),a2=read()+1,b2=read()+1;
    d[++cnt]=(dat){a1,b1,i,1};
    d[++cnt]=(dat){a1,b2,i,-1};
    d[++cnt]=(dat){a2,b1,i,-1};
    d[++cnt]=(dat){a2,b2,i,1};
}
sort(d+1,d+cnt+1);
for(int i=1;i<=cnt;i++){
    if(!d[i].id) { add(d[i].y); continue; }
    ans[d[i].id]+=d[i].p*ask(d[i].y);
}
```

P1972 [SDOI2009] HH 的项链

HH 有一串由各种漂亮的贝壳组成的项链。HH 相信不同的贝壳会带来好运，所以每次散步完后，他都会随意取出一段贝壳，思考它们所表达的含义。HH 不断地收集新的贝壳，因此，他的项链变得越来越长。

有一天，他突然提出了一个问题：某一段贝壳中，包含了多少种不同的贝壳？这个问题很难回答…… 因为项链实在是太长了。于是，他只好求助睿智的你，来解决这个问题。

P1972 [SDOI2009] HH 的项链

- 我们考虑将所有询问按照右端点归类。
- 然后从左往右扫描每个位置，如果前面有位置和它重复，就把前面的位置删掉（这样做是对的，因为右端点只可能在之后了，那么要访问到前面的位置，就必须要到达这个位置，相当于把重复的贡献减掉）。
- 初始时假设所有位置都不重复，都是1 。
- 每次操作只可能将某个位置减少1 ，或者区间查询。
- 可以使用树状数组。