数据结构与算法
（九）
树状数组和线段树基础

杜育根

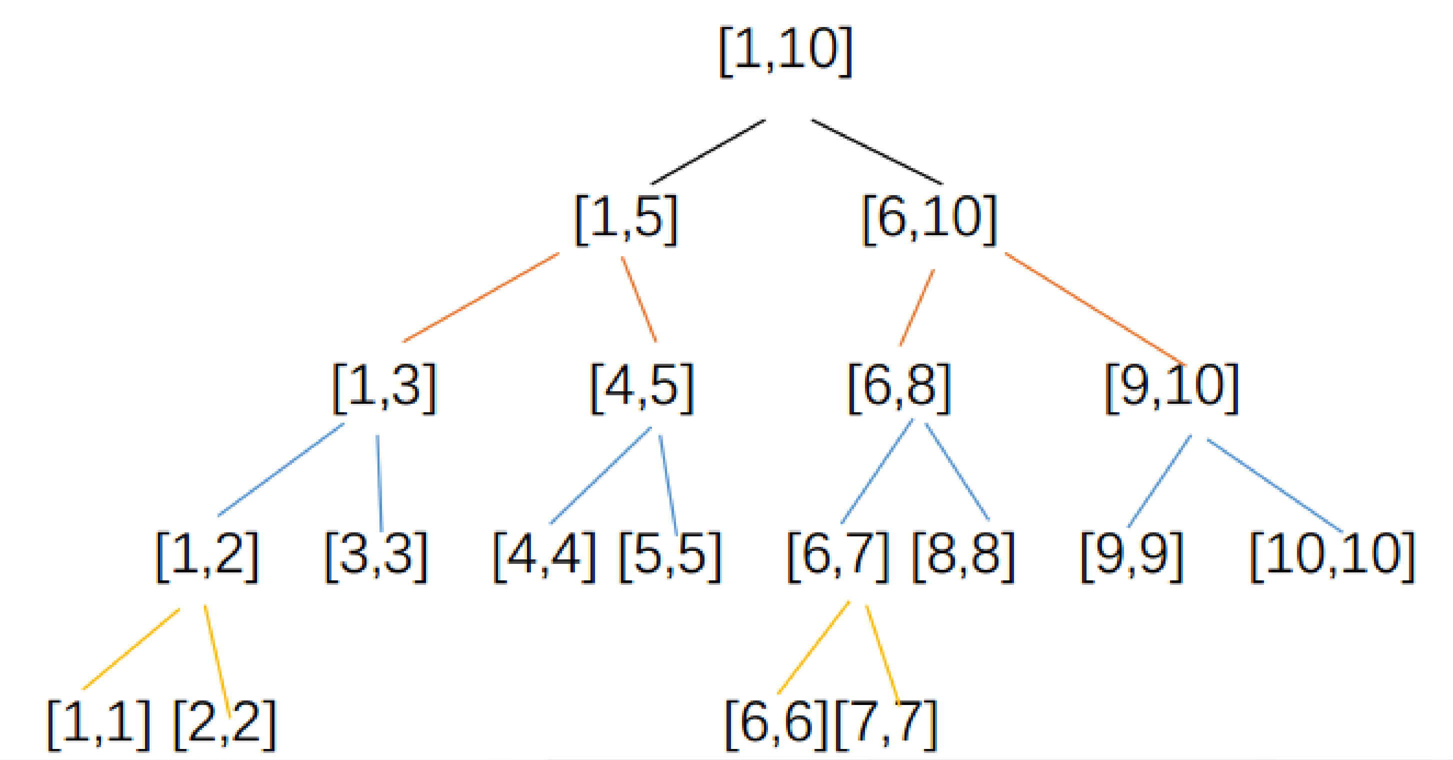
Ygdu@sei.ecnu.edu.cn

9. 树状数组和线段树基础

- 在节课程中，我们将学习两种高级数据结构——线段树和树状数组。线段树和数组都是可以维护区间上的信息，是解决这类问题的有用工具。我们将尝试单点修改、区间修改、区间查询等操作。
- 同时针对这两种高级数据结构，你将学习到两种高级用法——离散化和线扫描法。
- ACM题目中有不少连续区间的动态查询问题，例如求取某一区间上元素之和、求取某一区间上元素的最大值，此时如果使用一般的方法求解会使得时间超出要求。此时需要使用到线段树，主要用于高效解决连续区间的动态查询问题。

9.1. 线段树

- 就是一棵由线段构成的二叉树，每个结点都代表一条线段$[a,b]$（数组中的一段子数组）。非叶子的结点所对应的线段都有两个子结点，左儿子代表的线段为$[a,\frac{a+b}{2}]$，右儿子代表的线段为 $[\frac{a+b}{2}+1,b]$。使用线段树这一数据结构，可以查找一个连续区间中节点的信息，也可以修改一个连续区间中结点的信息。换句话说，它将优化区间操作的复杂度。



例

- 初始数组 $A_1, A_2, A_3 \dots A_n$ ，需要支持以下操作：

1. $x\ v$ 修改操作，对第x个元素加上v，即$A_x = A_x + v$。
2. $x\ y$ 查询操作， 询问区间和，即 $\sum_x^y A_i$。

修改一个节点x值+v

n=10,x=3
根结点 [1,10]，sum(1,10)+=v
左儿子是[1,5]，右儿子是[6,10]，选择左儿子。
当前结点 [1,5]，sum(1,5)+=v
左儿子是 [1,3]，右儿子是 [4,5]，选择左儿子。
当前结点 [1,3]，sum(1,3)+=v
左儿子是 [1,2]，右儿子是 [3,3]，选择右儿子。
当前结点 [3,3]，sum(3,3)+=v。
长度范围为 [1,n] 的一棵线段树的深度为 log(n)+1。
对照图示，我们若修改第3个元素，在线段树上，恰好修改 [1,10],[1,5],[1,3],[3,3] 这些区间。
由二叉树的性质可知，单次修改的区间数量是 log 级别的，单次修改的复杂度是 O(logn) 。

```
void modify(int p, int l, int r, int x, int v)
{
    s[p] += v;
    if (l == r) return; //叶结点则退出
    int mid = (l + r) / 2;
    if (x <= mid)
        //判断x在左儿子还是右儿子
        modify(p * 2, l, mid, x, v);
    else
        modify(p * 2 + 1, mid + 1, r, x, v);
}
```

- **也可以 push_up：把儿子结点的信息更新到父亲结点**

```
void up(int p)
{
    s[p] = s[p * 2] + s[p * 2 + 1];
}
```

```
void modify(int p, int l, int r, int x, int v)
{
    if (l == r)
    {
        s[p] += v;
        return;
    }
    int mid = (l + r) / 2;
    if (x <= mid)
        modify(p * 2, l, mid, x, v);
    else
        modify(p * 2 + 1, mid + 1, r, x, v);
    up(p);
}
```

查询

- 查询的区间［x,y］划分为线段树上的结点，然后将这些结点代表的区间合并起来得到所需信息。
- n=10,x=3,y=6，即我们需要求出 $A_3+A_4+A_5+A_6$。
- 而区间 [3,6] 的信息，刚好由线段树上区间 [3,3]，[4,5]，[6,6] 合并得到。
- 线段树上每层的结点最多会被选取2个，一共选取的结点数也是 O(logn) 的，因此查询的时间复杂度也是 O(logn)。

```
int query(int p, int l, int r, int x, int y)
{
    if (x <= l && r <= y) return s[p];//若该结点被查询区间包含
    int mid = (l + r) / 2, res = 0;
    if (x <= mid) res += query(p * 2, l, mid, x, y);
    if (y > mid) res += query(p * 2 + 1, mid + 1, r, x, y);
    return res;
}
```

习题：斑点蛇

- 有一种神奇斑点蛇，蛇如其名，全身都是斑点，斑点数量可以任意改变。有一天，小明十分的无聊，开始数蛇上的斑点。假设这条蛇的长度是N厘米，小明已经数完开始时蛇身的第i厘米上有a_i个斑点。现在小明想知道这条斑点蛇的任意区间的蛇身上一共有多少个斑点。这好像是一个很容易的事情，但是这条蛇好像是和小明过不去，总是刻意的改变蛇身上的斑点数量。于是，小明受不了了，加上小明有密集型恐惧症。聪明又能干的你能帮帮他吗?
- 输入格式：第一行一个正整数N（N≤50000）表示这条斑点蛇长度为 N 厘米，接下来有 N个正整数，第i个正整数 a_i代表斑点蛇第i 厘米上有 a_i个斑点（1≤a_i≤50）。接下来每行有一条命令，命令有 4 种形式：

(1) $Add\ i\ j$，i和 j为正整数，表示第 i 厘米增加 j 个斑点（j 不超过 30）；
(2) $Sub\ i\ j$，i和 j为正整数，表示第 i 厘米减少 j 个斑点（j 不超过 30）；
(3) $Query\ i\ j$，i和j为正整数，$i \le j$，表示询问第 i 到第 j厘米的斑点总数（包括第 i厘米和第 j厘米）；
(4) End 表示结束，这条命令在每组数据最后出现；

- 最多有 40000 条命令。
- 输出格式：对于每个 Query 询问，输出一个整数并回车，表示询问的段中的总斑点数，这个数保证在int范围内。

样例输入	样例输出
10 1 2 3 4 5 6 7 8 9 10 Query 1 3 Add 3 6 Query 2 7 Sub 10 2 Add 6 3 Query 3 10 End	6 33 59

习题：最甜的苹果

- 小明有很多苹果，每个苹果都有对应的甜度值。小明现在想快速知道从第i个苹果到第 j个苹果中，最甜的甜度值是多少。
- 因为存放时间久了，有的苹果会变甜，有的苹果会因为腐烂而变得不甜，所以小明有时候还需要修改第 i个苹果的甜度值。
- 输入格式：第一行输入两个正整数 N,M(0<N≤200000,0<M<5000)，分别代表苹果的个数和小明要进行的操作的数目。每个苹果从 1到 N 进行编号。
- 接下来一行共有 N个整数，分别代表这 N个苹果的初始甜度值。接下来 M 行。每一行有一个字符 C，和两个正整数 X，Y。当 C 为Q的时候，你需要输出从 X到 Y（包括 X, Y）的苹果当中，甜度值最高的苹果的甜度值。当 C 为U的时候，你需要把苹果 X 的甜度值更改为 Y。
- 输出格式：在一行里面输出每次询问的最高甜度值。

- 样例输入
- 5 6
- 1 2 3 4 5
- Q 1 5
- U 3 6
- Q 3 4
- Q 4 5
- U 2 9
- Q 1 5

- 样例输出
- 5
- 6
- 5
- 9

习题：公告板

- 工厂有一个 h×w 的矩形公告板，其中 h是高度，w 是宽度。
- 现在有若干张1×Wi 的公告， Wi 是宽度，公告只能横着放，即高度为 1 的边垂直于水平面，且不能互相有重叠，每张公告都要求尽可能的放在最上面的合法的位置上。
- 若可以放置，输出每块可放置的位置的行号；若不存在，输出−1。行号由上至下分别为 1,2,…, h。
- 输入格式：第一行三个整数 h，w，n（$1\le h,w\le 10^9;1\le n\le 200{,}000$）。接下来 n 行，每行一个整数 Wi($1\le Wi\le 10^9$)。
- 输出格式：输出n 行，一行一个整数。

- 样例输入
- 3 5 5
- 2
- 4
- 3
- 3
- 3

- 样例输出
- 1
- 2
- 1
- 3
- -1

9.2. 区间更新

- 本课程第一节讲过线段树区间查询的做法，同理，区间更新也可以分割成若干个子区间，每层的结点至多选取2个，时间复杂度 O(logn)。

懒惰（Lazy）标记

- 懒惰标记，也可称为延迟标记。一个区间可以转化为若干个结点，每个结点设一个标记，记录这个结点被进行了某种修改操作（这种修改操作会影响其子结点）。
- 也就是说，仅修改到这些结点，暂不修改其子结点；而后决定访问其子节点时，再下传懒惰（Lazy）标记，并消除原来的标记。优点在于，不用将区间的所有值暴力更新，大大提高效率。
- 在区间修改的一类问题中，我们可以设一个delta域，表示该节点需要加上数值delta。由于该节点表示的是一个区间，向下访问时，子节点的delta需要加上该节点的delta，同时该节点的delta变为0。访问叶子节点时，再对该元素的数值加上delta即可。
- 同理，在区间更新（赋值）的一类问题，我们可以设一个color域，表示该节点（区间）都被数值color覆盖。向下访问时，子节点的color更新成该节点的color，同时该节点的color变为0。访问叶子节点时，再将该元素修改成color即可。

线段树区间更新的一个例子的代码

```
void up(int p)
{
    if (!p) return;
    s[p] = s[p * 2] + s[p * 2 + 1];
}

void down(int p, int l, int r)
{
    if (col[p])
    {
        int mid = (l + r) / 2;
        s[p * 2] = col[p] * (mid - l + 1);
        s[p * 2 + 1] = col[p] * (r - mid);
        col[p * 2] = col[p * 2 + 1] = col[p];
        col[p] = 0;
    }
}
void modify(int p, int l, int r, int x, int y, int c)
{
    if (x <= l && r <= y)
    {
        s[p] = (r - l + 1) * c;  //仅修改该结点
        col[p] = c;  //增加标记，子结点待修改
        return;
    }
    down(p, l, r);  //下传lazy标记
     int mid = (l + r) / 2;
    if (x <= mid) modify(p * 2, l, mid, x, y, c);
    if (y>mid) modify(p*2+1, mid+1, r, x, y, c);
    up(p);
}
```

注意到，push_down 一般在访问子节点前执行，起到下传懒惰（延迟）标记的作用。push_up 在访问完子节点后执行，将两个子区间的信息合并起来，得到该区间的信息。

线段树区间查询的一个例子的代码

```
int query(int p,int l,int r,int x,int y){
    if(x <= l && r <= y){
        return s[p];
    }
    down(p,l,r);//如果没有return,这时子节点必须更新了  就向下更新一层
    int mid = (l + r) / 2, res = 0;
    if(x <= mid){
        res += query(p * 2, l,mid,x,y);
    }
    if(y > mid){
        res += query(p * 2 + 1,mid + 1,r,x,y);
    }
    return res;
}
```

习题：帕吉的肉钩

- 在 DotA 游戏中，帕吉的肉钩是很多英雄最害怕的东西。钩子由连续若干段的等长金属棒制成。
- 现在帕吉对钩子由一些操作：
- 我们将金属棒 1~n依次编号，帕吉可以把编号 x~y 的金属棒变成铜棒、银棒、金棒。
- 每段铜棒的价值是 1；每段银棒的价值是 2；每段金棒的价值是3。
- 肉钩的总价值是n 段金属棒价值之和。
- 帕吉想知道若干操作以后钩子的总价值。
- 输入格式：第一行一个整数N（$1\le N\le 10^5$），表示肉钩金属棒的数量，初始状态为 铜棒。
- 第二行一个整数 Q（$1\le Q\le 10^5$），表示操作数。
- 接下来Q行，一行三个整数，X,Y,Z($1\le X\le Y\le N, 1\le Z\le 3$)。
- 输出格式：一行一个整数，表示钩子的总价值，用样例中的格式。
- 样例输入
- 10
- 2
- 1 5 2
- 5 9 3
- 样例输出
- The total value of the hook is 24.

习题：区间整数操作

- 给出 N 个整数 $A1, A2, \ldots, AN$，你需要处理区间加，区间求和。
- 输入格式：第一行两个整数 N 和 Q （$1 \le N,Q \le 10^5$）。第二行 N 个整数，表示 A1,A2...AN($|Ai| \le 10^9$) 的初始值。接下来 Q行，每行一个操作：
- C a b c，表示 $A_a, Aa_{+1} \ldots Ab$ 每个数加 c ($|c| \le 10000$)。
- Q a b，表示询问$A_a, Aa_{+1} \ldots Ab$ 的和，答案可能超过 32位整数。
- 输出格式：对于所有 Q 询问，一行输出一个答案。
- 样例输入
- 10 5
- 1 2 3 4 5 6 7 8 9 10
- Q 4 4
- Q 1 10
- Q 2 4
- C 3 6 3
- Q 2 4
- 样例输出
- 4
- 55
- 9
- 15

习题：黑白石头

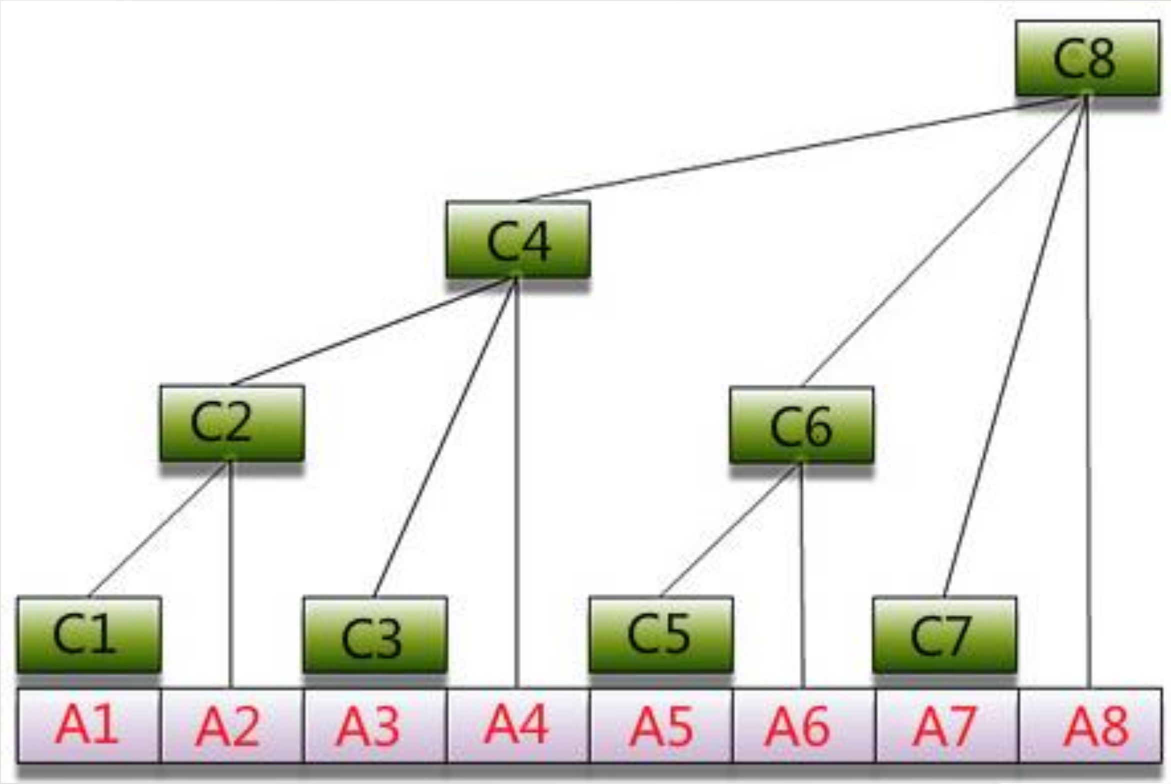
- 沙滩上有一些石头，石头的颜色是白色或者黑色。小羊会魔法，它能把连续一段的石头，黑色变成白色，白色变成黑色。小羊喜欢黑色，她想知道某些区间中最长的连续黑石头是多少个。
- 输入格式：第一行一个整数 n(1≤n≤100000)。第二行 n个整数，表示石头的颜色，0 是白色，1 是黑色。第三行一个整数 m(1≤m≤100000)。
- 接下来 m 行，每行三个整数 x,i,j。
- x=1 表示改变 [i,j] 石头的颜色。
- x=0 表示询问 [i,j] 最长连续黑色石头的长度。
- 输出格式：当 x=0 时，输出答案。

样例输入	样例输出
4	1
1 0 1 0	2
5	0
0 1 4	
1 2 3	
0 1 4	
1 3 3	
0 4 4	

提示：只需要维护区间内最大值，和区间两端的结果就可以了。
也许两种颜色的结果都保存会让计算过程更简单。

9.3. 树状数组

- 树状数组（Binary Indexed Tree(BIT), Fenwick Tree） 是一个查询和修改的复杂度都为 log(n) 的数据结构。
- 观察右图，令这棵树的结点编号为 C1,C2,..., Cn。令每个结点的值为这棵树的值的总和，那么容易发现：
- $C_1 = A_1$
- $C_2 = A_1 + A_2$
- $C_3 = A_3$
- $C_4 = A_1 + A_2 + A_3 + A_4$
- $C_5 = A_5$
- $C_6 = A_5 + A_6$
- $C_7 = A_7$
- $C_8 = A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8$
- ...
- $C_{16} = A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + A_8$
- $+ A_9 + A_{10} + A_{11} + A_{12} + A_{13} + A_{14} + A_{15} + A_{16}$



有一个有趣的性质：

- 设结点编号为 x，那么该结点区间为 2^k（其中 k为 x二进制末尾 0的个数）个元素。因为这个区间最后一个元素必然为A_x，
- 所以很明显，$C_n = A_{n-2^k+1} + \cdots + A_n$
- 计算这个 2^k，也就是最低位的 1，可以这样写：

1. int lowbit(int x) {
2. return x & (x ^ (x – 1));
3. }

- 利用机器补码特性，也可以写成：

1. int lowbit(int x) {
2. return x & -x;
3. }

查询

- 当想要查询一个$sum(1 \dots n)$ 即 $(a1 + a2 + \cdots + an)$，可以依据如下算法即可：

Step 1: 令 $sum = 0$，转第二步；
Step 2: 假如$n \le 0$，算法结束，返回sum值，否则 $sum = sum + Cn$，转第三步；
step 3: 令 $n = n - lowbit(n)$，转第二步。

- 可以看出，这个算法就是将这一个个区间的和全部加起来，为什么是效率是 log(n) 的呢?
- 证明：
- $n = n - lowbit(n)$ 等价于将 n的二进制的最后一个 1 减去。而 n 的二进制里最多有 log(n) 个 1，所以查询效率是 log(n) 的。

```
int getsum(int x) {
    int res = 0;
    for (; x; x -= lowbit(x)))
        res += C[x];
    return res;
}
```

修改

- step 1：当 i>n 时，算法结束，否则转第二步；
- step 2：$Ci = Ci + v, i = i + lowbit(i)$ 转第一步。
- $i = i + lowbit(i)$ 这个过程实际上也只是一个把末尾 1 补为 0的过程。
- 修改一个节点，必须修改其所有祖先，最坏情况下为修改第一个元素，最多有 log (n) 个祖先。

```
int change(int x, int v) {
    for (; x <= MAX_N; x +=lowbit(x))
        C[x]+=v;
}
```

习题：棋子等级

- 坐标系平面上有好多棋子，每个整点上至多有一个棋子。假定棋子的等级是左下方的棋子个数，现在给出若干棋子的位置，求不同等级的棋子各有多少个。
- 输入格式: 第一行一个整数 N (1≤N≤100000)
- 接下来 N 行，一行两个整数 X,Y (0≤X,Y<100000)，表示坐标。
- 数据保证坐标先按 Y 排序，再按 X 排序。
- 输出格式：N 行，每行一个整数，从 0 到 N–1 等级的棋子数量。
- 样例输入
- 5
- 1 1
- 5 1
- 7 1
- 3 3
- 5 5

- 样例输出
- 1
- 2
- 1
- 1
- 0

习题：木桩涂涂看

- n 个木桩排成一排，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），小明便骑上他的电动车从木桩 a 开始到木桩 b 依次给每个木桩涂一次颜色。但是 n次以后小明已经忘记了第 i 个木桩已经涂过几次颜色了，你能帮他算出每个木桩被涂过几次颜色吗?
- 输入格式
- 第一行是一个整数 n（n≤100000）。
- 接下来的 n行，每行包括两个整数 a, b (1≤a≤b≤n)。
- 输出格式
- n个整数，第 i个数代表第 i 个木桩总共被涂色的次数。
- 样例输入
- 3
- 1 1
- 1 2
- 1 3
- 样例输出
- 3 2 1

习题：校长的问题

- 学校中有 n 名学生，学号分别为 1...n。再一次考试过后，学校按照学生的分数排了一个名次（分数一样，按照名字的字典序排序）。你是一名老师，你明天要和校长汇报这次考试的考试情况，校长询问的方式很奇怪，比如说：“学号前 a的人中，排名前 b 的有多少人？”。
- 校长问了一堆这样的问题，你需要今天全部计算出来，明天好向他汇报。
- 输入格式：第一行俩个整数，n和m，分别表示学校学生的数量和校长问题的数量 ($1 \le n,m \le 10^5$)。
- 第二行有 n 个整数，表示按学号顺序这 n 个学生在这次考试中对应的排名。
- 接下来 m 行，每行俩个整数 a, b 表示校长的问题中的具体数字 ($1 \le a,b \le n$)。
- 输出格式：输出 m 行，每行一个整数代表校长问题的答案。

样例输入	样例输出
6 3	3
3 2 5 4 6 1	2
4 4	4
2 5	
6 4	

习题：奇怪的报数游戏

- 小明所在的单位在玩一个奇怪的报数游戏。一共有 n 个人参与游戏（小明不在其中），首先给每个人一个编号，分别为 1,2,3,..., n。之后，这些人按照某个顺序站成一队，并告诉小明，他们各自前面有多少人的编号比自己小。现在，你能帮小明计算出来队伍中每个人的编号依次是多少么?
- 输入格式：第一行一个整数 N(2≤N≤500000)。
- 第 2行到第 N 行，每行一个整数，其中第 i行表示第 i+1 个人前面有多少人的编号比他小；
- 第一个人的前面没有任何人，所以无需输入他报的结果。
- 输出格式：一共 N 行，每行一个整数，表示每个人的编号。

样例输入	样例输出
5	2
1	4
2	5
1	3
0	1

9.4. 二维树状数组

- 例: 由数字组成的矩阵，能进行两种操作：
- 1. 对矩阵里的某个数加上一个整数 Delta；
- 2. 查询某个子矩阵的和。
- 一维树状数组可以扩展到二维，数组 A 的树状数组定义为：
- $Cx,y = \sum A_{i,j} \quad ((x - lowbit(x) + 1) \le i \le x, (y - lowbit(y) + 1) \le j \le y))$
- 设原始二维数组为：

$$A = \{\{a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}, a_{17}, a_{18}, a_{19}\},$$
$$\{a_{21}, a_{22}, a_{23}, a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}\},$$
$$\{a_{31}, a_{32}, a_{33}, a_{34}, a_{35}, a_{36}, a_{37}, a_{38}, a_{39}\},$$
$$\{a_{41}, a_{42}, a_{43}, a_{44}, a_{45}, a_{46}, a_{47}, a_{48}, a_{49}\}\};$$

那么对应的二维树状数组 C 呢?

令:

$B[1]=\{a_{11},a_{11}+a_{12},a_{13},a_{11}+a_{12}+a_{13}+a_{14},a_{15},a_{15}+a_{16},...\}$ 这是第一行的一维树状数组;

$B[2]=\{a_{21},a_{21}+a_{22},a_{23},a_{21}+a_{22}+a_{23}+a_{24},a_{25},a_{25}+a_{26},...\}$ 这是第二行的一维树状数组;

$B[3]=\{a_{31},a_{31}+a_{32},a_{33},a_{31}+a_{32}+a_{33}+a_{34},a_{35},a_{35}+a_{36},...\}$ 这是第三行的一维树状数组;

$B[4]=\{a_{41},a_{41}+a_{42},a_{43},a_{41}+a_{42}+a_{43}+a_{44},a_{45},a_{45}+a_{46},...\}$ 这是第四行的一维树状数组。

那么: $C_{11}=a_{11},C_{12}=a_{11}+a_{12},C_{13}=a_{13},C_{14}=a_{11}+a_{12}+a_{13}+a_{14},C_{15}=a_{15},C_{16}=a_{15}+a_{16},...$

这是 A第一行的一维树状数组;

$$C21=a11+a21, C22=a11+a12+a21+a22, C23=a13+a23, C24=a11+a12+a13+a14+a21+a22+a23+a24, C25=a15+a25, C26=a15+a16+a25+a26, ...$$

这是 A数组第一行与第二行相加后的树状数组;

$$C31=a31, C32=a31+a32, C33=a33, C34=a31+a32+a33+a34, C35=a35, C36=a35+a36, ...$$

这是 A第三行的一维树状数组;

$$C41=a11+a21+a31+a41, C42=a11+a12+a21+a22+a31+a32+a41+a42, C43=a13+a23+a33+a43, ...$$

这是 A数组第一行 + 第二行 + 第三行 + 第四行后的树状数组。

修改

```
void change(int i, int j, int delta){
    A[i][j] += delta;
    for(int x = i; x < A.length; x += lowbit(x))
        for(int y = j; y < A[i].length; y += lowbit(y))
            C[x][y] += delta;
}
```

查询

```
int getsum(int i, int j){
   int res = 0;
   for(int x = i; x; x -= lowbit(x))
        for(int y = j; y; y -= lowbit(y))
            res += C[x][y];
   return res;
}
```

子矩阵 (x1,y1) ~ (x2,y2) 的和，我们只需四个矩阵容斥就可以解决：

$$Ans = sum(x2, y2) - sum(x2, y1-1) - sum(x1-1, y2) + sum(x1-1, y1-1)$$

习题：矩阵操作

给出 N×N 的矩阵 A，其中的元素是 0 或 1。初始时均为 0。我们可以修改矩阵，给定左上角（x1,y1），和右下角 (x2,y2)，对这个矩阵的所有元素执行取反操作，即 0 变成 1，1 变成0。

现在我们一共有两种操作：

1. $C\ x1\ y1\ x2\ y2(1 \le x1 \le x2 \le n, 1 \le y1 \le y2 \le n)$，修改矩形
2. $Q\ x\ y\ (1 \le x, y \le n)$ 查询$A_{x,y}$ （A的第x行第y列）的值

输入格式：第一行两个整数 N,T(2≤N≤1000,1≤T≤50000)。接下来 T 行，每行一个操作。

输出格式: 对于每个查询操作，一行一个整数表示$A_{x,y}$。

样例输入

```
2 10
C 2 1 2 2
Q 2 2
C 2 1 2 1
Q 1 1
C 1 1 2 1
C 1 2 1 2
C 1 1 2 2
Q 1 1
C 1 1 2 1
Q 2 1
```

样例输出

```
1
0
0
1
```

习题：矩阵查询

- 给出 N×N 的矩阵 A，初始时均为 0。我们需要支持两种操作：
- C x1 y1 c，表示 (x1,y1) 上的元素加上 c ($1 \le x1, y1 \le N, 1 \le c \le 100$) 。
- Q x1 y1 x2 y2，查询子矩阵元素之和 ($1 \le x1 \le x2 \le N, 1 \le y1 \le y2 \le N$) 。
- 输入格式：第一行两个整数 N,Q($1 \le N \le 1000, 1 \le Q \le 50000$)。接下来 Q 行，每行一个操作。
- 输出格式：对于每个询问操作，一行一个整数，表示矩阵的和。
- 样例输入
- 3 5
- C 1 1 3
- Q 1 1 2 3
- C 1 2 1
- C 2 1 1
- Q 1 1 2 3

- 样例输出
- 3
- 5

9.5. 树状数组维护区间最值

- 在区间求和时，我们只需求出 [1,r]，[1,l−1]，利用前缀和的可减性，得到区间 [l,r] 的和。
- 但区间最值不满足这个性质。
- 我们可以把区间 [l,r] 拆分成若干个子区间，再合并得到答案。
- 画图可知，max_i 需要的 max 只有 $max_{i-2^0}, max_{i-2^1}, max_{i-2^2}, \dots, max_{i-lowbit(i)+1}$。
- 修改

```
void change(int r) {
    c[r] = a[r];
    for(int i = 1; i < lowbit(r); i <<= 1)
        c[r] = max(c[r], c[r-i]);
}
```

查询

- 我们找 $[l,r]$ 的最值就是子区间最值的 max，即递减 r，在这里可以有个优化，即当找到一个 max_i，有 $i-lowbit(i)\geq l$ 时，更新后，$i=i-lowbit(i)$，然后继续递减。当 $l>r$ 就跳出循环。

```
int getmax(int l, int r) {
    int ret = a[r];
    while(l <= r) {
        ret = max(ret, a[r]);
        for(--r; r - l >= lowbit(r); r -= lowbit(r))
            ret = max(ret, c[r]);
    }
    return ret;
}
```

- 需要指出的是，它只支持末端插入，不支持单点修改操作。

习题：小明的任务

- 小明的上司给小明布置了一个任务，小明维护一个数列，要求提供以下两种操作：

1、 查询操作。语法：Q L 功能：查询当前数列中末尾 L 个数中的最大的数，并输出这个数的值。

2、 插入操作。语法：A n 功能：将 n 加上 t，其中 t 是最近一次查询操作的答案（如果还未执行过查询操作，则 t=0)，并将所得结果对一个固定的常数 D 取模，将所得答案插入到数列的末尾。

- 初始时数列是空的，没有一个数。
- 输入格式：第一行两个整数，M 和 D，其中 M 表示操作的个数(M≤200,000)，D 如上文中所述，满足 D 在 64 位整型范围内。接下来 M行，查询操作或者插入操作。
- 输出格式：对于每一个询问操作，输出一行。该行只有一个数，即序列中最后 L 个数的最大数。

- 样例输入
- 5 100
- A 96
- Q 1
- A 97
- Q 1
- Q 2

- 样例输出
- 96
- 93
- 96

9.6. 数据的离散化

- 什么是离散化
- 离散化 在程序设计竞赛中是一个非常实用的技巧，它可以通过降低数据的规模来降低算法的时间复杂度。
- 离散化可以在不改变数据相对大小的条件下，对数据进行相应的缩小。例如：

原数据：	1	999999	20	5555	100
离散化后：	1	5	2	4	3

- 可以发现，离散化之后数据之间的大小关系没有发生改变，但是数据范围从1~999999，变为了 1~5。

离散化的实现

离散化的实现方式很灵活，可以把原数据和离散后数据存在一个结构体里面进行俩次排序；可以对原数据排序后二分查找原数据所在位置；可以对原数据排序后通过映射的方式来确定离散化后的数据等等。可以发现，不管怎么做复杂度都是 O(nlgn)，所以任选一种方式就好。

来介绍一种代码实现起来很方便的方式，对原数据排序去重后，建立原数据与离散化后数据的映射。

- **对原数据排序**
- **去除排序后数组中的重复元素**
- **记录数组中的元素对应的离散化后的值，也就是此时元素的下标**

这样 ans 数组就是离散化之后的数据，我们也可以通过 id 数组来查询离散化后的数据对应的原数据。

```
int num[maxn]; //原数据数组
int tp[maxn]; // 中间数组
int ans[maxn]; //离散化后数组
int n; //数据数量
map<int, int> mp;
//原数组与离散化后数据的映射关系
int id[maxn]; //离散化后的数据对原数据的映射
for (int i = 0; i < n; ++i) {
  tp[i] = num[i];
}
sort(tp, tp + n);
int m = unique(tp, tp + n) - tp; // 对数组去重， m 为去
//重后数组中元素个数
for (int i = 0; i < m; ++i) {
   mp[tp[i]] = i + 1; //建立原数据与离散化后数据的映射
   id[i+1] = tp[i];
}
for (int i = 0; i < n; ++i) {
  ans[i] = mp[num[i]];
}
```

离散化扩展到点

- 我们可以从单个数字的离散化，扩展到点（俩个数字，横纵坐标）的离散化，只要对横纵坐标分别进行离散化处理即可。
- 离散化是一种缩小数据范围的方法，往往需要和其他算法的结合使用。例如：离散化 + 树状数组求逆序数。当数字的取值范围较小的时候，我们可以直接通过树状数组来求逆序数。如果数字取值范围较大，个数却很小时，我们可以通过离散化来缩小取值范围，再用树状数组来求逆序数。

习题：排序

- 你需要分析排序算法，将 n 个互不相同的整数，通过交换两个相邻的元素使得数列有序的 最少交换次数。
- 比如，原数列为：9,1,0,5,4。排序后的数列为：0,1,4,5,9
- 输入格式: 第一行一个整数 $n(n \le 500000)$。
- 接下来 n行，每行一个整数 $ai(ai \le 10^9)$。
- 输出格式: 输出一个整数，表示操作次数。
- 样例输入
- 5
- 9
- 1
- 0
- 5
- 4
- 样例输出
- 6

- 提示: 一次有效的交换意味着什么呢?
- 为了使序列有序，一次有效的交换应该是后一个较小的数与他前一个较大的数交换，那么单独一个数字的交换次数，应该是这个数字前面比它大的数字的个数。

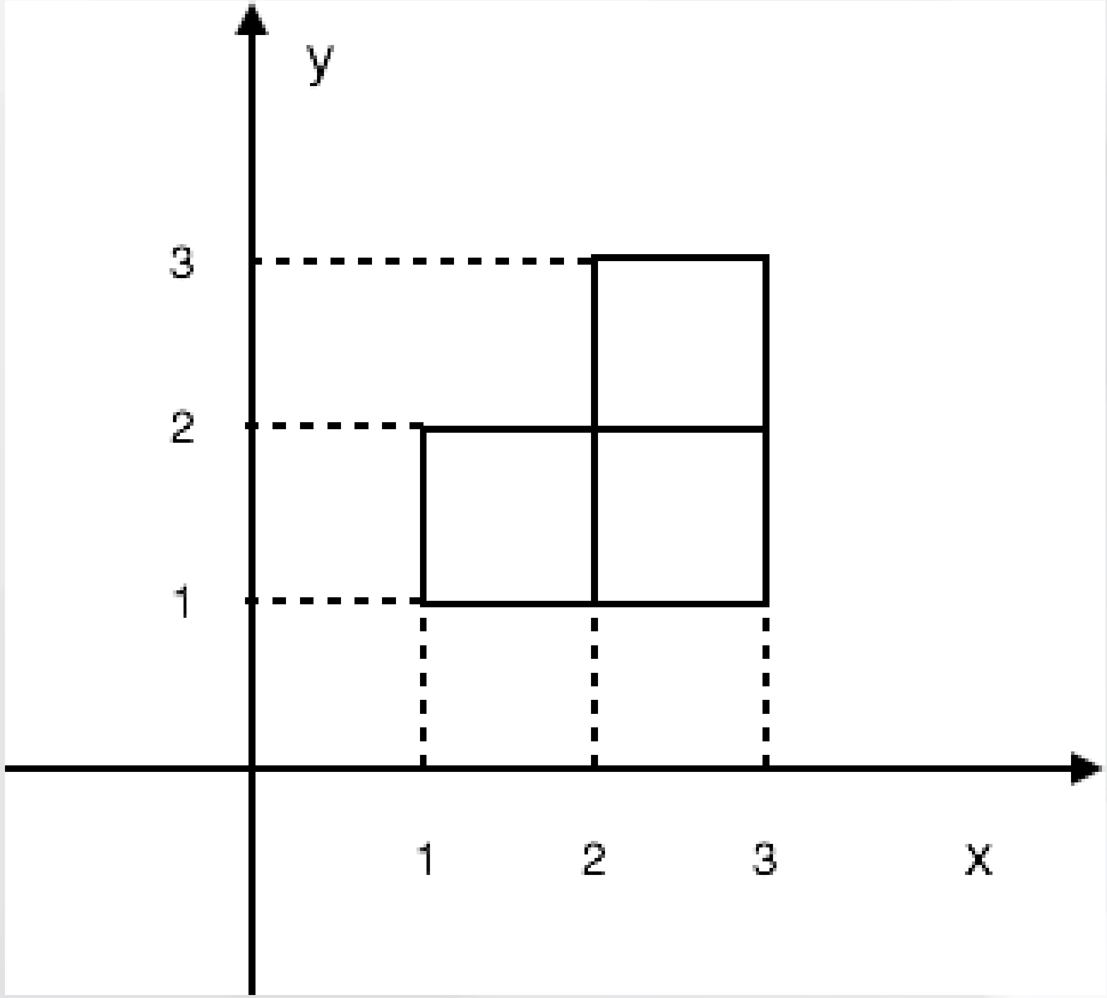
习题：学校的宣传板

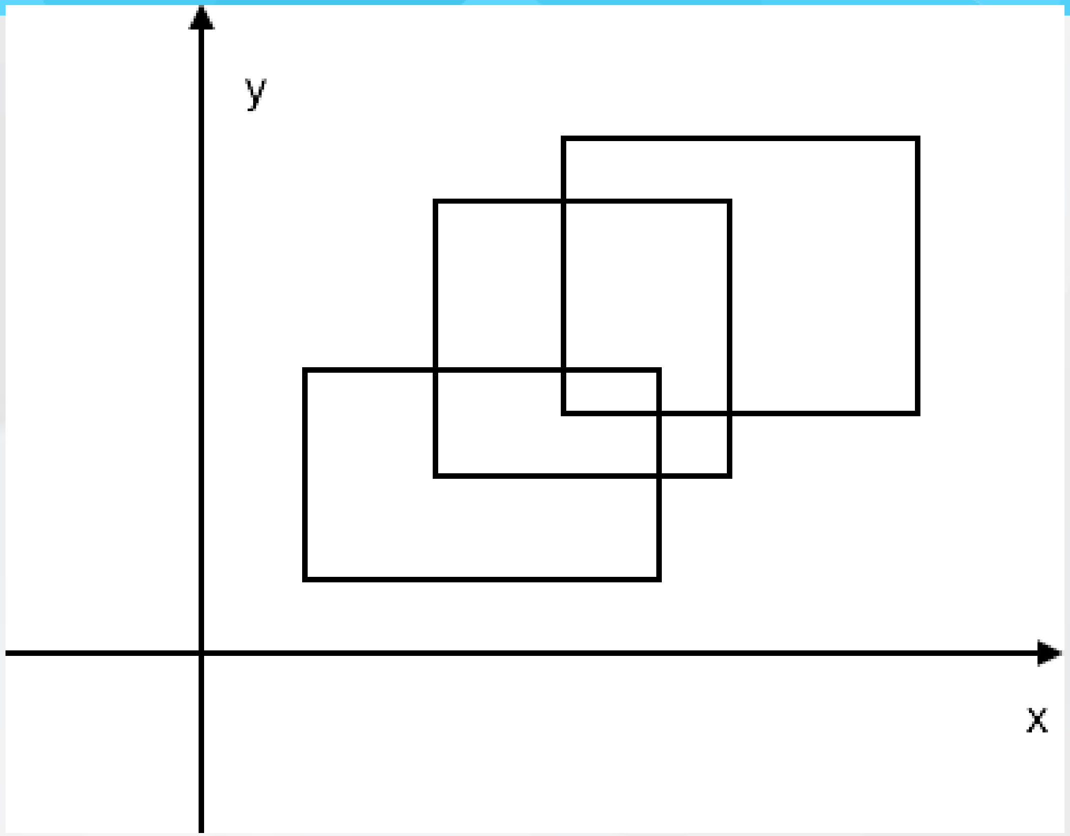
- 学校里有一个宣传板，大大小小事情的宣传都会往上面贴，上面布满了各种宣传海报。时间一长，上面的海报越来越多，一些陈旧的海报被新的完全覆盖掉了，一点都看不到了。
- 宣传板的高度和所有海报的高度都是一样的，但是宣传板很长。宣传板的长度为 m，每次贴完海报，都会做一个记录，这张海报贴在了宣传板的 a 到 b 处。希望你能通过这些数据，来计算出宣传板上有多少张海报是可以看到的。（只要能看到一个角，也算能看到）
- 输入格式：第一行俩个整数，n 和 m，分别表示海报的总个数和宣传板的长度 $(1 \le n \le 10^4, 2 \le m \le 10^9)$。接下来 n 行，每行俩个整数 a, b 表示海报的位置。输入的顺序代表贴海报的顺序，后面的海报会盖住前面的海报 $(1 \le a < b \le m)$。
- 输出格式：输出一个整数，代表最后能看到的海报个数。

样例输入	样例输出
5 100	3
20 35	
5 30	
50 80	
31 60	
55 85	

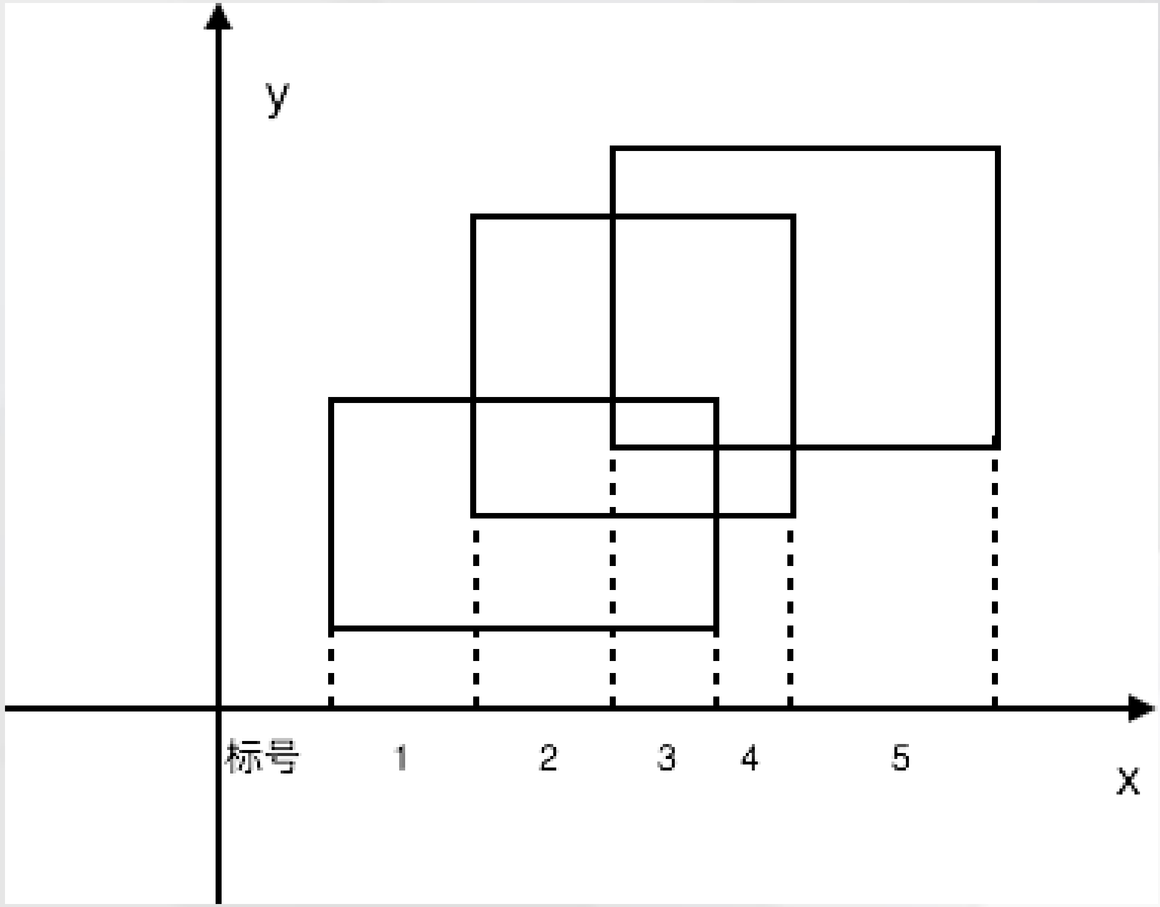
9.7. 线扫描法

- 离散化加线扫描法计算几何中常用的算法。我们通过求解矩形周长并来理解线段树和线扫描是如何结合起来的。
- 矩形周长并问题描述是这样：给定若干个二维平面坐标系上的矩形，矩形的顶点坐标都是整点。求出这些矩形的外轮廓的长度。如下图的这些矩形的周长并是8。
- 把矩形分成横线和竖线处理。处理横线和竖线的方法是一样的。所以我们只介绍处理横线的方法。

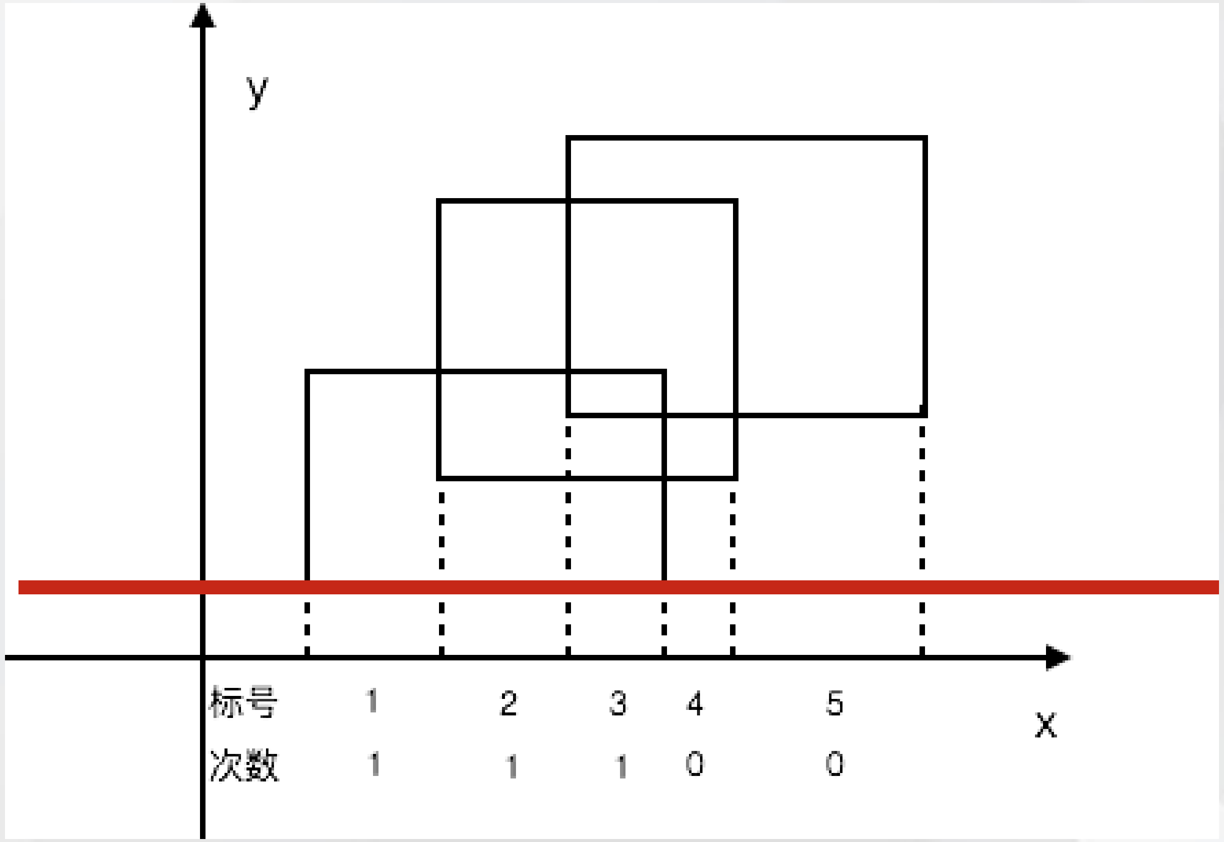




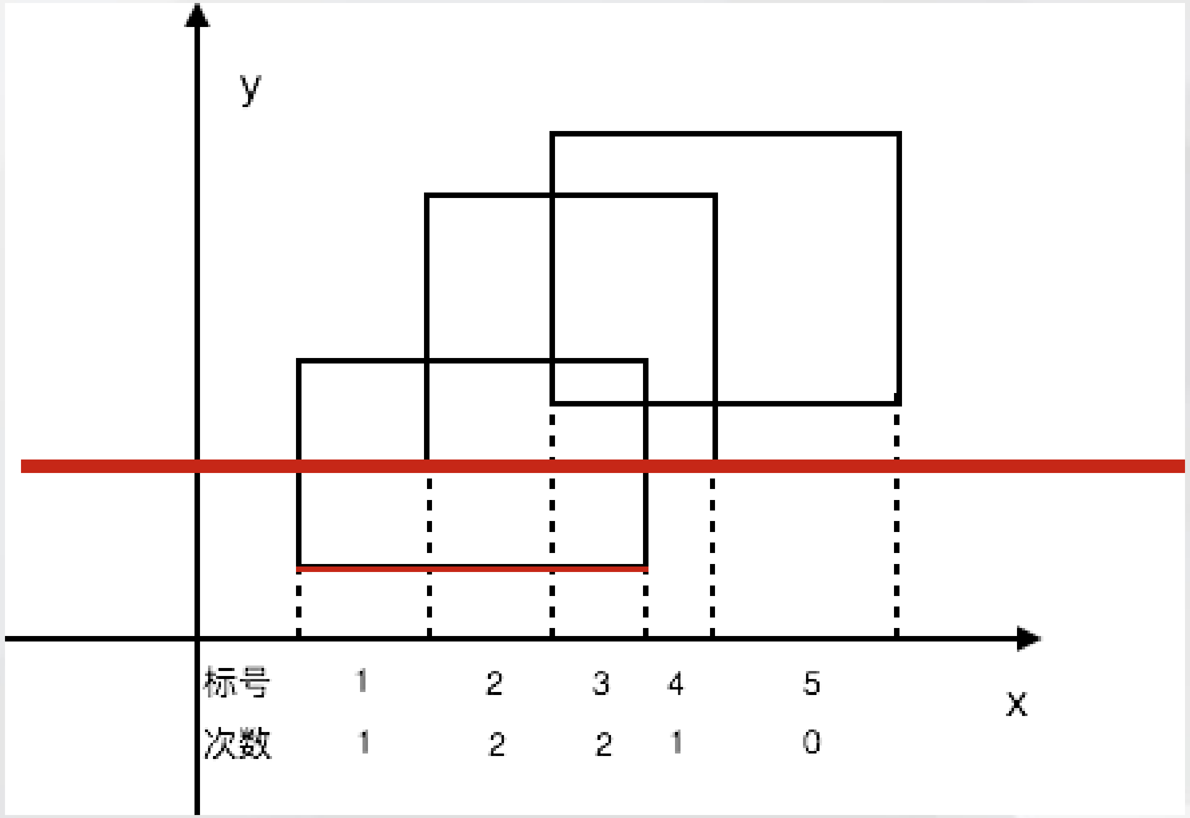
- 对于左面的图形我们先将横坐标离散化。这样每条横线都对应到成若干个区间的。我们区间从左到右标号。



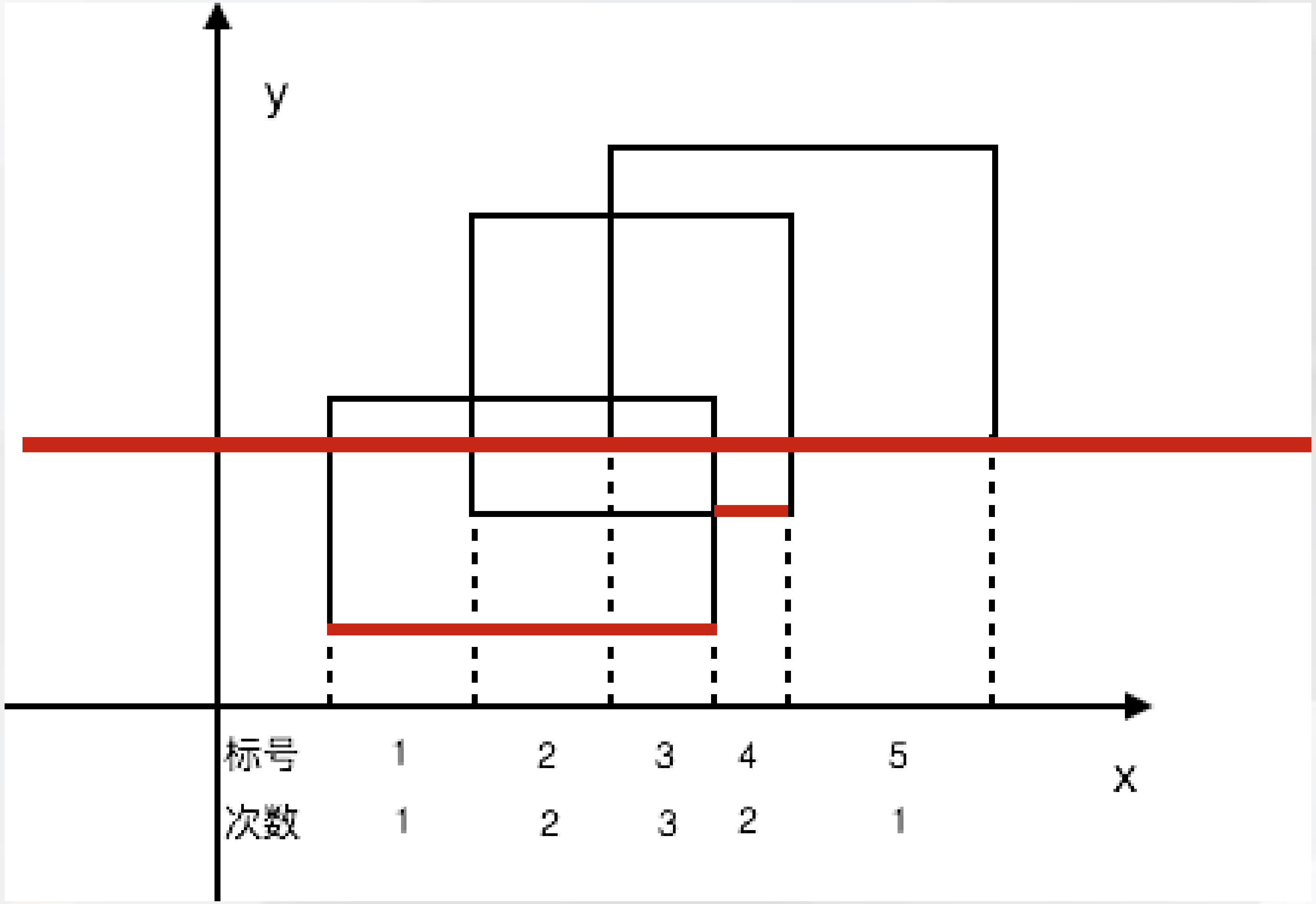
- 我们记录下面每个区间被覆盖的次数。我们画一条直线平行于 x 轴，让它从 y 轴的负无穷大的位置开始向 y 轴这正方向运动，直到遇到了第一条横线，这时候的状态就如下图。我们规定扫描线遇到了一个矩形的下边的时候，对应的区间覆盖次数加一，如果遇到上边，对应的区间的覆盖次数减一。因为在此之前都没有遇到横线，所以现在遇到的这条横线对应的区间覆盖次数都是一。这时候周长总长度加上这条横线的长度，也就是覆盖次数大于 0 的区间的长度。



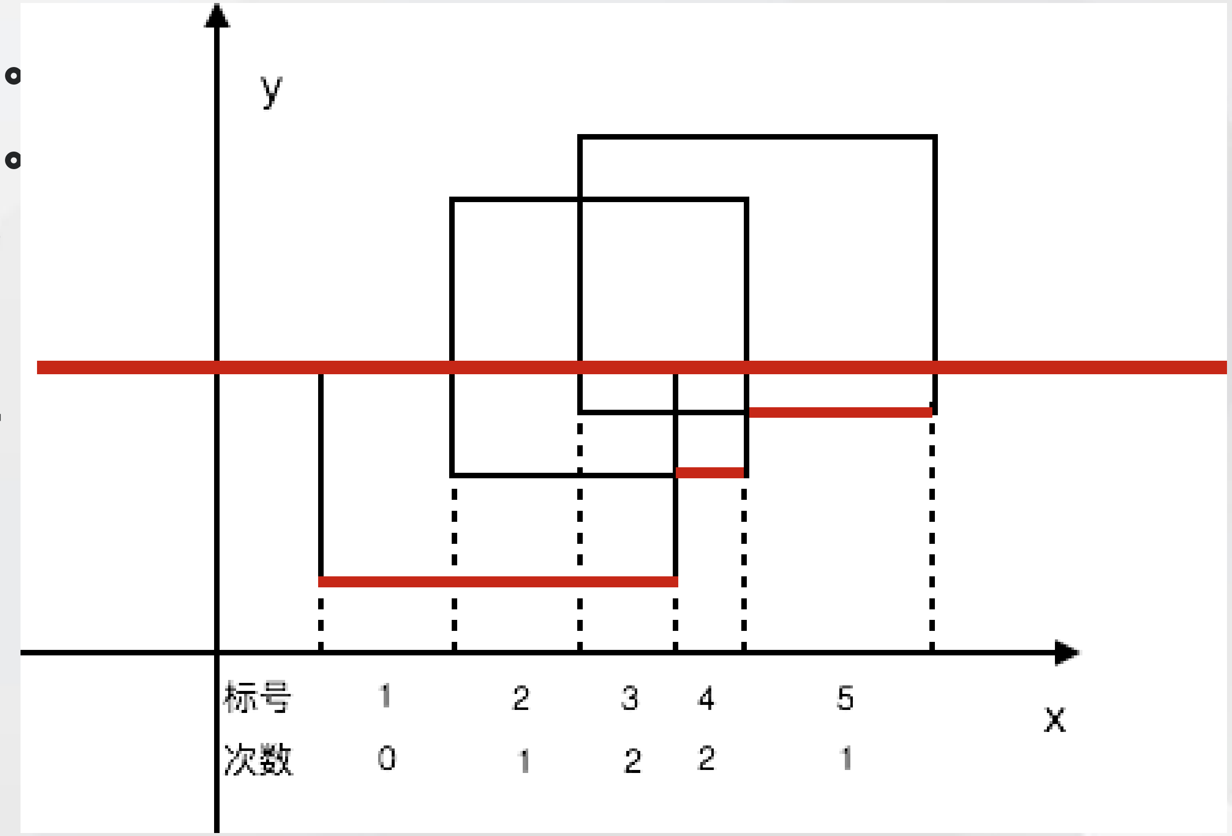
- 然后扫描线继续移动，在遇到下一条横线之前，状态都不会发生改变，知道遇到下一条横线的时候，如图，遇到的还是下边，这时候更新区间次数。这时候我们不能加上所有被覆盖次数大于 0 的区间的长度，因为有些地方是在其他矩形内部的。实际上，引起了被覆盖长度变化的部分才是真正的边界，也就是我们需要用覆盖的区间的长度减去上一次被覆盖的区间的长度才是真正增加的边界。这里我们就需要用线段树来维护，因为涉及到区间覆盖（加减一）和区间求和。对应图中实际上就是增加了区间 4 的长度。



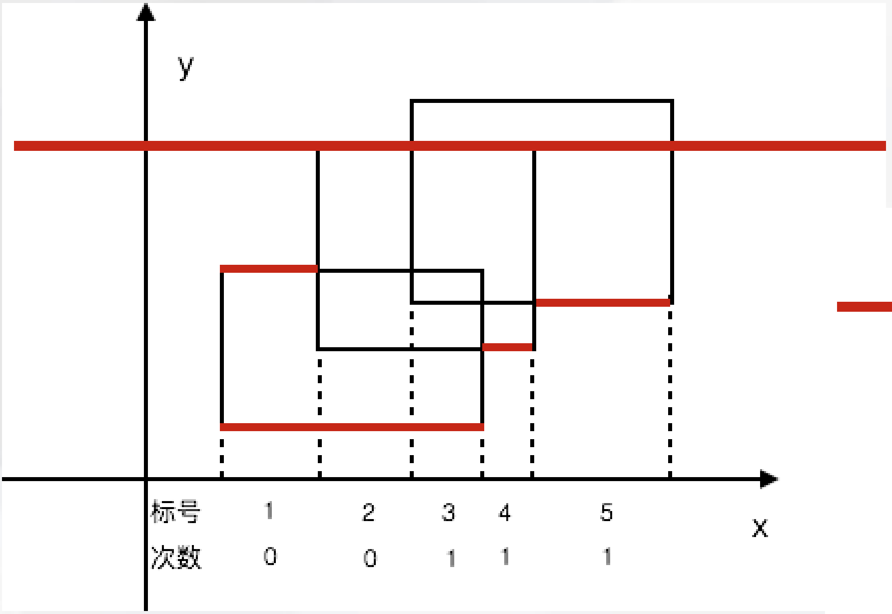
- 继续扫描，状态变成如下。

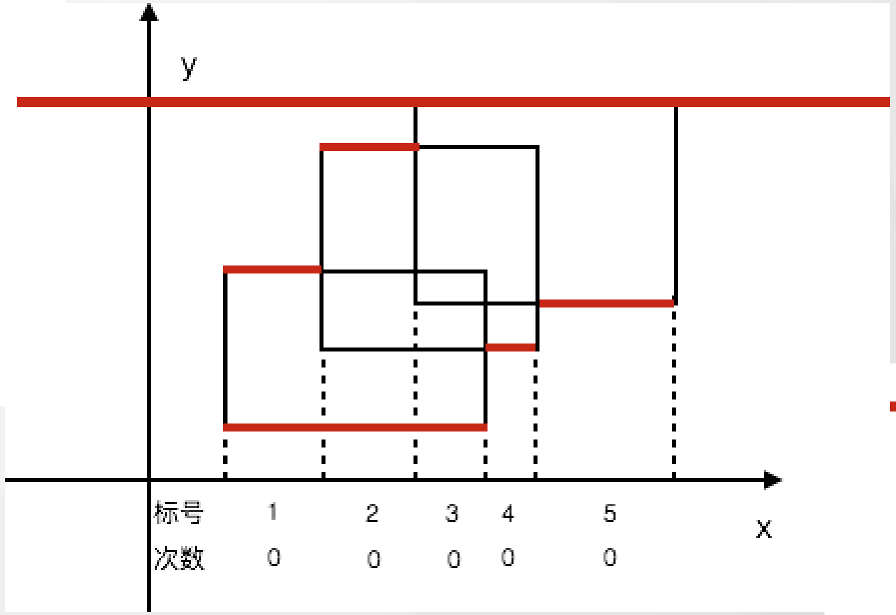


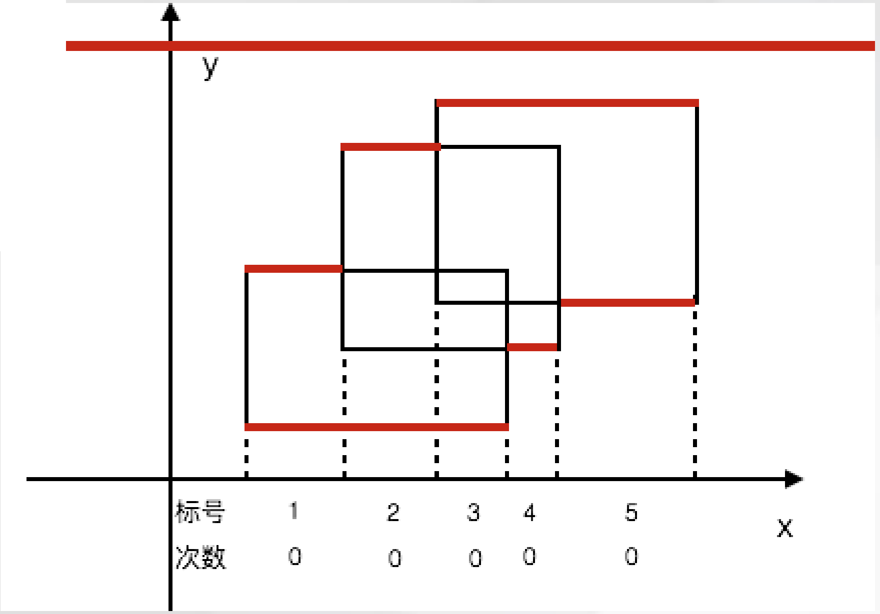
- 继续扫描，这时候遇到了一条上边。上面更新对应的区间覆盖次数建议。这时候总得被覆盖的区间可能会减少，而这减少的一部分实际上就是图形边界。我们还是加上当前覆盖总长度和上一次的差值。



扫描结束







流程总结

- 总结一下算法流程大致如下
- 1. 先把坐标离散化。
- 2. 然后把矩形的上边和下边单独拆出来，然后按照 y 排序。
- 3. 扫描横线，遇到下边区间加一，遇到上边区间减一。总周长加上更新后的总覆盖长度和更新前的总覆盖长度的差值的绝对值。
- 4. 按照用同样的方法统计竖线的长度。
- 对于求矩形的面积并的和交实际上也是一样的思想。

习题：矩形的周长并

- 在一个二维坐标平面中，有 n 个矩形，每个矩形的边都是平行于 x 轴或 y 轴的。给定这 n 个矩形的位置，这 n 个矩形之间可以相互覆盖。
- 求这 n个矩形所组成的图形的周长是多少。
- 输入格式
- 第一行输入一个整数 n，表示平面中矩形的数量。$(1 \le n \le 50000)$
- 接下来 n 行，每行四个整数 x1, y1, x2, y2 表示每个矩形左下角的坐标和右上角的坐标。$(0 \le x1 < x2 \le 10^7, 0 \le y1 < y2 \le 10^7)$
- 输出格式
- 输出一个数字，代表图形的周长。
- 样例输入
- 3
- 0 0 10 10
- 5 5 8 13
- 1 3 23 7
- 样例输出
- 72

习题：矩形的面积并

- 在一个二维坐标平面中，有 n 个矩形，每个矩形的边都是平行于 x 轴或 y 轴的。给定这 n 个矩形的位置，这 n 个矩形之间可以相互覆盖。
- 求这 n 个矩形所占用的平面总面积是多少。
- 输入格式：第一行输入一个整数 n，表示平面中矩形的数量。$(1 \le n \le 50000)$
- 接下来 n 行，每行四整数 x1, y1, x2, y2 表示每个矩形左下角的坐标和右上角的坐标。$(0 \le x1 < x2 \le 10^7, 0 \le y1 < y2 \le 10^7)$
- 输出格式：输出一个数字，代表矩形占用的总面积。
- 样例输入
- 3
- 0 0 10 10
- 5 5 8 13
- 1 3 23 7
- 样例输出
- 161