电子科技大学

数据结构专题训练

解题报告

电子工程学院 张逸伟

2017.5

**A - An easy problem A**

**Time Limit: 2000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)**

Submit Status

N个数排成一列，Q个询问，每次询问一段区间内的数的极差是多少。

**Input**

第一行两个整数N(1≤N≤50000),Q(1≤Q≤200000)。接下来一行N个整数a1 a2 a3 ....an,(1≤ai≤1000000000)。接下来Q行，每行两个整数L,R(1≤L≤R≤N)。

**Output**

对于每个询问输出一行，一个整数表示区间内的极差。

**Sample input and output**

| **Sample Input** | **Sample Output** |
| --- | --- |
| 5 3  3 2 7 9 10  1 5  2 3  3 5 | 8  5  3 |

题意：

在给定的数列中静态查找区间最值。

题解：

RMQ模板题。

因为数列是静态的，所以可以直接利用RMQ算法高效解决，对于每一次查询分别在区间内查找最大值和最小值，最后输出他们的差值。预处理时间复杂度O(NlogN)，查询时间复杂度O(Q).(Q为查询次数)

RMQ算法的具体实现，详见代码及注释。

AC代码：

//RMQ

#include <cstdio>

#include <iostream>

using namespace std;

const int maxn=50005;

int a[maxn],amax[maxn][20],amin[maxn][20];

int b[20];

int main() {

int i,n,q,k,h,j;

scanf("%d%d",&n,&q);

for (i=1;i<=n;i++) {

scanf("%d",&a[i]);

amax[i][0]=amin[i][0]=a[i];

}

//预处理，amax[i][j]和amin[i][j]分别表示从a[i]开始连续2^j个数的最大值/最小值

for (h=1,i=2;i<=n;h++,i\*=2) {

for (j=1;j+i-1<=n;j++){

amax[j][h]=max(amax[j][h-1],amax[j+i/2][h-1]);

amin[j][h]=min(amin[j][h-1],amin[j+i/2][h-1]);

}

}

int x,y,m,l;

for (i=1;i<=q;i++) {

h=0;k=1;

scanf("%d%d",&x,&y);

while (k\*2<=(y-x+1)) {

k\*=2;h++;

}

//对于每次询问，将区间分为两段，每段长度为2^h,一段以区间头开始，一段以区间尾结束

m=max(amax[x][h],amax[y-k+1][h]);

l=min(amin[x][h],amin[y-k+1][h]);

printf("%d\n",m-l);

}

return 0;

}

/\*

5 3

3 2 7 9 10

1 5

2 3

3 5

\*/

# B - An easy problem B

##### Time Limit: 2000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)

Submit Status

N个数排成一列，每个数的大小为1或者0。有两种操作，第一种操作是把一段区间内的每个数异或1，第二种操作是询问区间内最长连续1的长度。

## Input

第一行一个整数N（1≤N≤100000），表示N个数。第二行N个数。接下来一行一个整数M（1≤M≤100000）,表示M个操作，接下来M行每行三个整数K,L,R。K=1表示把L到R这段区间的数全部异或上1，K=0表示询问L到R这段区间内最长连续1的长度。

## Output

对于每个询问，输出对应的答案，每个询问占一行。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 5  0 1 0 0 1  5  0 1 4  1 1 1  0 1 4  1 3 4  0 1 4 | 1  2  4 |

**题意：  
给定一个仅由0和1构成的数列，要求完成两项操作：**

1. **对任意一段区间内所有元素异或1。**
2. **查询任意区间内元素都为1的子序列的最长长度。**

**题解：**

**看到区间更新、区间查询，很自然的想到用线段树求解。**

**怎样维护区间内连续1的长度呢？本题中，更新操作只涉及一种运算，即异或1。异或的一些性质或许能成为本题的突破口。异或1的运算法则如下：**

**0 xor 1 = 1, 1 xor 1 =0。**

**我们发现，对于题中所给序列，异或1的操作实际上相当于取反。那么，我们可以在初始化线段树时同时记录0和1的连续长度及相关量，当一段区间需要更新时，直接交换此区间内0和1的性质，并随着线段树一层一层向上更新。**

**还有一个问题尚未解决：如何求区间内连续1的最长长度？受到SPOJ上一道求指定区间内连续元素最大和的题目的启发，我们可以用到一个小技巧：对于树上每个节点，记录区间内从区间开头元素开始的最长连续1长度、区间内以区间末尾元素结束的最长连续1长度、区间内最长连续1长度，然后从下至上递归求出最长长度。对于0，同理。**

**具体而言，**

**Root.len = max(lchild.len , rchild.len, lchild.rlen + rchild.llen)**

**Root.llen = max(lchild.llen , lchild.num + rchild.llen(左孩子的区间全为1时))**

**Root.rlen = max(rchild.rlen , lchild.rlen + rchild.num(右孩子的区间全为1时))**

**其中Root为根，lchild为左子，rchild为右子，Num为区间元素个数，llen为区间内从区间开头元素开始的最长连续1长度，rlen为区间内以区间末尾元素结束的最长连续1长度，len为区间内最长连续1长度。**

**此外，为了优化时间，还使用到了isxor标记。与线段树中的加法标记等一样，记录区间是否异或。异或两次的效果相当于没有异或，且标记有延迟作用，没有使用到更小的区间时可以先延时，不进行更多操作，利用这两点节省时间。**

**AC代码：**

#include <cstdio>

#include <iostream>

using namespace std;

const int maxn=100005;

struct Tree {

int l,r,lc,rc;

int m0,lm0,rm0,m1,lm1,rm1; //分别记录区间内0和1的性质

bool isxor,flag0,flag1; //flag记录区间内是否全为0或是否全为1

};

Tree tree[3\*maxn];

int num;

int max (int a,int b) {

if (a>b) return a; else return b;

}

int min(int a,int b) {

if (a>b) return b; else return a;

}

//向上递推函数

void pushup(int now) {

int l=tree[now].lc,r=tree[now].rc;

//递推flag0,flag1

tree[now].flag0=tree[l].flag0&&tree[r].flag0;

tree[now].flag1=tree[l].flag1&&tree[r].flag1;

//递推0的性质

tree[now].m0=max(tree[l].m0,tree[r].m0);

tree[now].m0=max(tree[now].m0,tree[l].rm0+tree[r].lm0);

if (tree[l].flag0)

tree[now].lm0=tree[l].r-tree[l].l+1+tree[r].lm0;

else tree[now].lm0=tree[l].lm0;

if (tree[r].flag0)

tree[now].rm0=tree[r].r-tree[r].l+1+tree[l].rm0;

else tree[now].rm0=tree[r].rm0;

//递推1的性质

tree[now].m1=max(tree[l].m1,tree[r].m1);

tree[now].m1=max(tree[now].m1,tree[l].rm1+tree[r].lm1);

if (tree[l].flag1)

tree[now].lm1=tree[l].r-tree[l].l+1+tree[r].lm1;

else tree[now].lm1=tree[l].lm1;

if (tree[r].flag1)

tree[now].rm1=tree[r].r-tree[r].l+1+tree[l].rm1;

else tree[now].rm1=tree[r].rm1;

}

//当区间异或时，调用ToDoXor函数交换0和1的性质

void ToDoXor(int now) {

Tree t=tree[now];

tree[now].flag0=t.flag1;tree[now].flag1=t.flag0;

tree[now].m0=t.m1; tree[now].m1=t.m0;

tree[now].lm0=t.lm1; tree[now].lm1=t.lm0;

tree[now].rm0=t.rm1; tree[now].rm1=t.rm0;

}

//向下下放isxor标记

void pushdown(int now) {

if (tree[now].isxor==0) return;

int l=tree[now].lc,r=tree[now].rc;

tree[l].isxor=tree[l].isxor^tree[now].isxor;

tree[r].isxor=tree[r].isxor^tree[now].isxor;

tree[now].isxor=0;

ToDoXor(l);

ToDoXor(r);

}

//递归建立线段树

void build(int now,int l,int r) {

tree[now].l=l;

tree[now].r=r;

tree[now].isxor=0;

if (l!=r) {

num++;

tree[now].lc=num;

build(num,l,(l+r)/2);

num++;

tree[now].rc=num;

build(num,(l+r)/2+1,r);

pushup(now);

} else {

int x;

scanf("%d",&x);

if (x) {

tree[now].flag1=1;

tree[now].lm1=tree[now].rm1=tree[now].m1=1;

tree[now].lm0=tree[now].rm0=tree[now].m0=0;

tree[now].flag0=0;

} else {

tree[now].flag1=0;

tree[now].lm1=tree[now].rm1=tree[now].m1=0;

tree[now].lm0=tree[now].rm0=tree[now].m0=1;

tree[now].flag0=1;

}

}

}

//更新异或情况

void update (int now,int l,int r) {

//cout << now << ' ' << tree[now].l << ' ' << tree[now].r << ' ' << tree[now].lm1 << ' ' << tree[now].rm1 << ' ' << tree[now].m1 << endl;

if (tree[now].l>=l&&tree[now].r<=r) {

tree[now].isxor=tree[now].isxor^1;

ToDoXor(now);

} else {

pushdown(now);

if (l<=(tree[now].l+tree[now].r)/2)

update(tree[now].lc,l,r);

if (r>(tree[now].l+tree[now].r)/2)

update(tree[now].rc,l,r);

pushup(now);

}

//cout << now << ' ' << tree[now].l << ' ' << tree[now].r << ' ' << tree[now].lm1 << ' ' << tree[now].rm1 << ' ' << tree[now].m1 << endl;

}

//查找指定区间最长长度

Tree findm1(int now,int l,int r) {

// cout << now << ' ' << tree[now].l << ' ' << tree[now].r << ' ' << tree[now].lm1 << ' ' << tree[now].rm1 << ' ' << tree[now].m1 << endl;

if (tree[now].l>=l&&tree[now].r<=r) {

return tree[now];

} else {

pushdown(now);

if (r<=tree[tree[now].lc].r) return findm1(tree[now].lc,l,r);

else if (l>=tree[tree[now].rc].l) return findm1(tree[now].rc,l,r);

else

{

Tree t,t1,t2;

t1=findm1(tree[now].lc,l,r);

t2=findm1(tree[now].rc,l,r);

t=tree[now];

t.l=t1.l;

t.r=t2.r;

t.flag1=t1.flag1&&t2.flag1;

t.m1=max(t1.m1,t2.m1);

t.m1=max(t.m1,t1.rm1+t2.lm1);

if (t1.flag1)

t.lm1=t1.r-t1.l+1+t2.lm1;

else t.lm1=t1.lm1;

if (t2.flag1)

t.rm1=t2.r-t2.l+1+t1.rm1;

else t.rm1=t2.rm1;

// cout << ' ' << t.l << ' ' << t.r << ' ' << t.lm1 << ' ' << t.rm1 << ' ' << t.m1 << endl;

return t;

}

}

}

int main() {

int n,m,i,k,l,r;

scanf("%d",&n);

num=1;

build(1,1,n);

scanf("%d",&m);

for (i=1;i<=m;i++) {

scanf("%d%d%d",&k,&l,&r);

if (k)

update(1,l,r);

else {

Tree t=findm1(1,l,r);

printf("%d\n",t.m1);

}

}

return 0;

}

/\*

8

1 0 1 1 0 1 0 1

100000

1 3 4

1 2 5

0 1 5

\*/

**C - An easy problem C**

**Time Limit: 4000/2000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)**

Submit Status

N个数排成一列，有三种操作。1.给一段区间内的每个数乘上一个非负整数。2.给一段区间内的每个数加上一个非负整数.3.询问一段区间的和模上P的值。

**Input**

第一行两个整数N(1≤N≤100000)表示数的个数,P（1≤P≤1000000000）表示模的值。接下来一行N个整数ai(0≤ai≤1000000000),接下来一行一个整数M(1≤M≤100000)表示操作数量，接下来M行每行描述一个操作。第一种操作描述：1 L R C（0≤C≤1000000000）,表示把L到R这段区间每个数乘上一个C。第二种操作描述：2 L R C（0≤C≤1000000000）,表示把L到R这段区间每个数加上一个C。第三种操作3 L R 表示询问L到R这段区间内的数的和模上P的值。

**Output**

对面每个询问，输出对应的答案，每个询问占一行。

**Sample input and output**

| **Sample Input** | **Sample Output** |
| --- | --- |
| 7 43  1 2 3 4 5 6 7  5  1 2 5 5  3 2 4  2 3 7 9  3 1 3  3 4 7 | 2  35  8 |

题意：

有一个长度为N的数列，要求进行三项操作：区间加法，区间乘法，区间和查询。

题解：

一开始想要用数组分块做，后来发现这样做的话对每个区间两侧的不组成完整的块的元素操作很麻烦，遂放弃，改用线段树。这里就涉及到了线段树的加法、乘法标记，处理的时候要特别小心，每次更新、查询要记得一同更新标记。可以说是线段树的模板题了。

具体实现见代码与代码旁的注释。

AC代码：

//线段树，加法、乘法标记，区间求和

#include <cstdio>

#include <iostream>

#include <string.h>

using namespace std;

typedef long long ll;

const int maxn=100500;

int num;

struct Tree {

int lc,rc,l,r;

ll tag,sum,mul;

};

Tree tree[4\*maxn];

ll p;

//递归建立线段树

//tag为加法标记，mul为乘法标记，sum为区间和

void build(int now,int l,int r) {

tree[now].l=l;

tree[now].r=r;

tree[now].tag=0;

tree[now].mul=1;

if (l!=r) {

num++;

tree[now].lc=num;

build(num,l,(l+r)/2);

num++;

tree[now].rc=num;

build(num,(l+r)/2+1,r);

tree[now].sum=tree[tree[now].lc].sum+tree[tree[now].rc].sum;

tree[now].sum%=p;

} else scanf("%lld",&tree[now].sum);

}

//下放加法标记、乘法标记，向下更新sum

void pushdown(int now) {

int l=tree[now].lc;

int r=tree[now].rc;

tree[l].tag=(tree[l].tag\*tree[now].mul+tree[now].tag)%p;

tree[r].tag=(tree[r].tag\*tree[now].mul+tree[now].tag)%p;

tree[l].mul\*=tree[now].mul;

tree[l].mul%=p;

tree[r].mul\*=tree[now].mul;

tree[r].mul%=p;

tree[l].sum\*=tree[now].mul;

tree[l].sum+=tree[now].tag\*(tree[l].r-tree[l].l+1);

tree[l].sum%=p;

tree[r].sum\*=tree[now].mul;

tree[r].sum+=tree[now].tag\*(tree[r].r-tree[r].l+1);

tree[r].sum%=p;

tree[now].tag=0;tree[now].mul=1;

}

//区间加法操作

void add(int now,int l,int r,ll c) {

if (tree[now].l>=l&&tree[now].r<=r) {

tree[now].tag+=c;

tree[now].tag%=p;

tree[now].sum+=c\*(tree[now].r-tree[now].l+1);

tree[now].sum%=p;

} else {

pushdown(now);

if (l<=(tree[now].l+tree[now].r)/2)

add(tree[now].lc,l,r,c);

if (r>(tree[now].l+tree[now].r)/2)

add(tree[now].rc,l,r,c);

tree[now].sum=(tree[tree[now].lc].sum+tree[tree[now].rc].sum)%p;

}

}

//区间乘法操作

void multiply(int now,int l,int r,ll c) {

if (tree[now].l>=l&&tree[now].r<=r) {

tree[now].tag\*=c;

tree[now].tag%=p;

tree[now].mul\*=c;

tree[now].mul%=p;

tree[now].sum\*=c;

tree[now].sum%=p;

} else {

pushdown(now);

if (l<=(tree[now].l+tree[now].r)/2)

multiply(tree[now].lc,l,r,c);

if (r>(tree[now].l+tree[now].r)/2)

multiply(tree[now].rc,l,r,c);

tree[now].sum=(tree[tree[now].lc].sum+tree[tree[now].rc].sum)%p;

}

}

//区间查询操作

ll findsum(int now,int l,int r) {

// cout << now << ' ' << tree[now].l << ' ' << tree[now].r << ' ' << tree[now].sum << ' ' << tree[now].tag << ' ' << tree[now].mul << endl;

if (tree[now].l>=l&&tree[now].r<=r) {

return tree[now].sum;

} else {

pushdown(now);

ll f=0;

if (l<=(tree[now].l+tree[now].r)/2)

f=findsum(tree[now].lc,l,r);

if (r>(tree[now].l+tree[now].r)/2)

f+=findsum(tree[now].rc,l,r);

return f%p;

}

}

int main() {

int n,m,l,op,r,k;

ll c;

scanf("%d%lld",&n,&p);

num=1;

build(1,1,n);

scanf("%d",&m);

for (k=1;k<=m;k++) {

scanf("%d",&op);

if (op==1) {

scanf("%d%d%lld",&l,&r,&c);

multiply(1,l,r,c);

} else if (op==2) {

scanf("%d%d%lld",&l,&r,&c);

add(1,l,r,c);

} else {

scanf("%d%d",&l,&r);

ll ans=findsum(1,l,r);

printf("%lld\n",ans);

}

// if (op!=3)

// for (int now=1;now<=num;now++)

// cout << now << ' ' << tree[now].l << ' ' << tree[now].r << ' ' << tree[now].sum << endl;

}

return 0;

}

# D - Washi与Sonochi的约定

##### Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 131072/131072KB (Java/Others)

Submit Status

#### SonochiSonochi，明年再一起看烟花。——WashioSumiWashioSumi



为了实现和SonochiSonochi的约定，WashiWashi必须要打败眼前强大的怪物。  
怪物分布在二维平面上，  
某个怪物的rankrank被定义为xx坐标不大于其xx坐标，且yy坐标不大于其yy坐标的怪物的数量。（不含其自身）  
WashiWashi要你输出nn行，每行一个整数，分别代表rankrank为00~n−1n−1的怪物数量。

## Input

输入第一行为一个正整数nn，  
接下来nn行，第ii行两个整数xixi、yiyi，表示一个怪物的坐标。  
保证输入的坐标两两不同。

## Output

包含nn行，每行一个整数，第ii行的数代表rankrank为i−1i−1的怪物数量。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 5  1 1  5 1  7 1  3 3  5 5 | 1  2  1  1  0 |

## Hint

1≤n≤1000001≤n≤100000  
1≤xi≤1000001≤xi≤100000  
1≤yi≤100000

题意：

坐标系上有N个点，计算每个点包含多少个点，输出包含I (0<=I<=N-1) 个点的点的个数。

题解：

何为包含？题目中的定义是X和Y的坐标均比自己小或等于自己。注意到此时有两个变量需要比较，不是很好操作，我们可以以其中一个为第一关键字，另一个为第二关键字从小到大排序，减少变量的数量。此外，排序还有一个好处，那就是对于任意一个点，它包含的点只可能排在它前面，这使得我们处理一个点时不用考虑还没有处理到的点。

好了，现在排序排好了，下一步怎么办呢？对于任意一个点，一个X或Y坐标显然只能被比它大或相等的坐标包含。现在，假设我们把X坐标排序好了，只要考虑Y坐标。怎么查询在此前访问过的坐标中多少个Y坐标比现在的Y坐标小或相等呢？注意到坐标数值最大只有100000，可以构造线段树解决问题。

开始时，令区间长度为100000的线段树中每个元素都为0.接着，对按X为第一关键字、Y为第二关键字排序后的坐标一个一个操作。对于一个Y坐标，它所能让后面的Y坐标被包含的范围是[Y,100000],不妨每次令区间[Y,100000]中每个元素加1，每个元素Ai代表Y坐标为i时之前操作过的坐标中有多少个可以被包含。这样，查询某点包含多少点只要在线段树中查询Ai就可以了。

AC代码：

#include <cstdio>

#include <iostream>

#include <string.h>

#include <algorithm>

using namespace std;

const int maxn=100500;

const int inf=0x3f3f3f3f;

struct Point {

int x,y;

};

Point point[maxn];

int r[maxn];

int num;

struct Tree {

int lc,rc,l,r,sum,tag;

};

Tree tree[4\*maxn];

//按X为第一关键字，Y为第二关键字的cmp函数

bool cmp(Point a,Point b) {

return (a.x<b.x)||(a.x==b.x&&a.y<b.y);

}

//建立线段树

void build(int now,int l,int r) {

tree[now].l=l;

tree[now].r=r;

tree[now].sum=tree[now].tag=0;

if (l!=r) {

num++;

tree[now].lc=num;

build(num,l,(l+r)/2);

num++;

tree[now].rc=num;

build(num,(l+r)/2+1,r);

}

// cout << now << ' ' << tree[now].l << ' ' << tree[now].r << endl;

}

//下放加法标记

void pushdown(int now) {

if (tree[now].tag==0) return;

int l=tree[now].lc,r=tree[now].rc;

tree[l].tag+=tree[now].tag;

tree[r].tag+=tree[now].tag;

tree[l].sum+=tree[now].tag\*(tree[l].r-tree[l].l+1);

tree[r].sum+=tree[now].tag\*(tree[r].r-tree[r].l+1);

tree[now].tag=0;

}

//更新区间元素

void update (int now,int l,int r,int c) {

if (tree[now].l>=l&&tree[now].r<=r) {

tree[now].tag+=c;

tree[now].sum+=c\*(tree[now].r-tree[now].l+1);

} else {

pushdown(now);

if (l<=(tree[now].l+tree[now].r)/2)

update(tree[now].lc,l,r,c);

if (r>(tree[now].l+tree[now].r)/2)

update(tree[now].rc,l,r,c);

tree[now].sum=tree[tree[now].lc].sum+tree[tree[now].rc].sum;

}

}

//查询区间和

int findsum(int now,int l,int r) {

// cout << now << ' ' << tree[now].l << ' ' << tree[now].r << ' ' << tree[now].tag << endl;

if (tree[now].l>=l&&tree[now].r<=r) {

return tree[now].sum;

} else {

pushdown(now);

int f=0;

if (l<=(tree[now].l+tree[now].r)/2)

f=findsum(tree[now].lc,l,r);

if (r>(tree[now].l+tree[now].r)/2)

f+=findsum(tree[now].rc,l,r);

return f;

}

}

int main() {

int n,i,maxy;

maxy=-1;

scanf("%d",&n);

for (i=1;i<=n;i++) {

scanf("%d%d",&point[i].x,&point[i].y);

maxy=max(maxy,point[i].y);

r[i]=0;

}

num=1;

sort(point+1,point+n+1,cmp);

build(1,1,maxy);

for (i=1;i<=n;i++) {

int f=findsum(1,point[i].y,point[i].y);

r[f]++;

update(1,point[i].y,maxy,1);

}

for (i=0;i<n;i++) printf("%d\n",r[i]);

return 0;

}

/\*

8

100000 100000

999 999

900 4200

100 86

80 45

40 80

40 20

1 4

\*/

# E - 曜酱的心意

##### Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 131072/131072KB (Java/Others)

Submit Status

#### ChikaChika说希望和我一起做学园偶像的时候，我真的很开心。——WatanabeYouWatanabeYou



曜是千歌的青梅竹马，但是AqoursAqours成立以后，千歌似乎总是与梨子在一起，而把曜冷落了。  
为了让千歌知晓自己的心意，曜酱决定做一件大事！  
她决定把一个给定的11~nn的排列{a1,a2,…,an}{a1,a2,…,an}（1≤ai≤n1≤ai≤n，且aiai各不相同），  
用最少的交换次数，变换成另一个11~nn的排列{b1,b2,…,bn}{b1,b2,…,bn}。并且，每次只交换**相邻**的两个元素。  
也许这样做了以后，千歌能更多地注意自己吧。曜这样想。

## Input

第一行是一个整数nn，  
第二行是一个长度为nn的11~nn的排列aa，  
第三行是另一个长度为nn的11~nn的排列bb。

## Output

输出一行，一个整数，表示最少的交换次数。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 4  2 3 1 4  3 2 1 4 | 1 |
| 3  3 2 1  1 2 3 | 3 |

## Hint

1≤n≤100000

题意：

将一个序列元素不变顺序改变变为另外一个序列，问只移动相邻元素至少需要多少步恢复原状。

题解：

首先是顺序的问题。两个序列元素相同，且元素保证各不相同，我们就可以将它们离散化。具体的过程是：将两个序列中的数按排序后的顺序编号[1,2,3…n]，再将原来的序列用[1,2,3…n]表示。这样，我们的问题就转换成了把无序序列变成有序，需要交换相邻元素至少多少次。

将无序变有序，只能移动相邻元素，这是冒泡排序的思想。如此，我们只要计算冒泡排序需要多少次。由于小的数字必然在前面，所以小的数字必然要和它前面比它大的数字交换，只要统计有多少对i<j,a[i]>a[j]的情况出现就可以了。

这样一来问题就变成了典型的逆序对计算问题，可以用树状数组轻松解决。具体求解过程如下：

开始时树状数组元素值全为0.离散化之后，对于a[i]=j，将树状数组原数列中的位置j加一，并在树状数组中更新区间和。这时，a[i]构成的逆序对的总数即之前的数中值比j大的数的个数，即树状数组原数列中区间[i+1,n]上的和。此时已经在树状数组更新了i个数，原数列所有数的和为i(每次加1)，所以区间[i+1,n]上的和就是i-sum(1,i),这也就是a[i]所构成的逆序数总数。Sum(1,i)利用树状数组可以求出。最后全部相加即可。

//逆序对

#include <cstdio>

#include <string.h>

#include <iostream>

#include <algorithm>

typedef long long ll;

using namespace std;

const int maxn=100005;

struct JCole {

int a,b;

};

JCole x[maxn],y[maxn];

int order[maxn];

ll f[maxn];

int n;

ll ans;

bool cmp(JCole q,JCole w) {

return q.a<w.a;

}

//树状数组专用的lowbit函数

ll lowbit(ll a) {

return (a&(-a));

}

//树状数组求和函数

ll getsum(int tt) {

ll sum=0;

int t=tt;

while (t) {

sum+=f[t];

t-=lowbit(t);

}

return sum;

}

//树状数组更新函数

void update(int tt,ll c) {

int t=tt;

while (t<=n) {

f[t]+=c;

t+=lowbit(t);

}

}

int main() {

int i;

scanf("%d",&n);

for (i=1;i<=n;i++) {

scanf("%d",&x[i].a);

x[i].b=i;

}

sort(x+1,x+n+1,cmp);

for (i=1;i<=n;i++) {

scanf("%d",&y[i].a);

y[i].b=i;

}

sort(y+1,y+n+1,cmp);

//离散化

for (i=1;i<=n;i++) {

order[x[i].b]=y[i].b;

f[i]=0;

}

//求解逆序对个数

ans=0;

for (i=1;i<=n;i++) {

update(order[i],1);

ans+=(ll)((ll)i-getsum(order[i]));

}

printf("%lld",ans);

return 0;

}

# F - 奇迹的魔法啊，再度出现！

##### Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 131072/131072KB (Java/Others)

Submit Status

#### 想对你说的话，如山似海。——AsahinaMiraiAsahinaMirai



决战之后，魔法界和无魔法界分离，为了和RicoRico再次相见，MiraiMirai必须用经受考验，唤醒奇迹的魔法。  
那考验就是——出现了nn个非负整数a1,a2,…,ana1,a2,…,an，  
对于mm次询问，第jj次询问给定一个正整数xjxj，  
输出max{a1XORxj,a2XORxj,…,anXORxj}max{a1XORxj,a2XORxj,…,anXORxj}。

XORXOR运算：0XOR1=10XOR1=1，1XOR0=11XOR0=1，0XOR0=00XOR0=0，1XOR1=01XOR1=0。  
按位XORXOR运算：对两个数的二进制位依次进行XORXOR运算。  
还对XORXOR运算不懂的请去百度或谷歌一下：异或。

## Input

输入的第一行为一个正整数nn，  
接下来一行是nn个非负整数a1,a2,…,ana1,a2,…,an。  
接下来为一个正整数mm，  
接下来一行，为mm个非负整数x1,x2,...,xmx1,x2,...,xm。

## Output

输出mm行，每行11个值，表示所求答案。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 5  1 8 0 5 14  5  7 14 10 4 2 | 15  15  15  12  12 |

## Hint

1≤n≤1000001≤n≤100000，  
1≤m≤1000001≤m≤100000，  
0≤ai≤21474836470≤ai≤2147483647，  
0≤xi≤2147483647

题意：

有N个数，M次询问，每次询问输入一个数，输出这个数与N个数分别异或结果的最大值。

题解：

首先要想到的是，异或有什么性质？

0 xor 0 = 0, 1 xor 0 = 1, 0 xor 1 = 1, 1 xor 1 =0

既然是求最大值，肯定要让结果当中的1尽可能多。

求异或的最大值，我们对于读入的每一个数，可以使用贪心。即从读入的数的二进制最高位开始，逐位选择0或1尽量使结果为1。如果某位为0，尽量选择1；某位为1，尽量选择0。

怎样判断N个数中有没有可以贪心的数呢？考虑到数字最大到2^31-1，即二进制最多有30位，可以使用Trie树。构造一棵深度31的Trie树，从根节点开始，每个节点的左儿子表示0，右儿子表示1，从根节点到叶节点的路径连起来便构成了一个数的二进制。然而，这样需要2^31-1个节点，内存不足。

但是实际上，我们注意到最多有100000个数字，每个数字二进制最多有30位，从而最多产生60个节点，这样一来节点最多有100000\*60=6,000,000，满足空间要求。

对于每个节点，增加一个Num标记，当这个节点可以继续往下走时Num标记为1，否则为0.查询时，沿着Trie树自上而下尽量凑出1即可。

#include <cstdio>

#include <iostream>

using namespace std;

const int maxn=100005;

int num;

struct Tree {

bool num;

int lc,rc;

};

Tree tree[maxn\*60];

int a[33];

//在字典树中插入一个数，递归实现

void insert(int now,int v,int depth) {

if (tree[now].num==0) {

tree[now].num=1;

num++;

tree[now].lc=num;

tree[num].num=0;

num++;

tree[now].rc=num;

tree[num].num=0;

}

if (depth==-1) return;

if ((v>>depth)%2==1)

insert(tree[now].rc,v,depth-1);

else

insert(tree[now].lc,v,depth-1);

}

//在字典树中根据读入的数尽量凑1，递归实现

int find(int now,int v,int depth) {

if (depth<0) return 0;

if ((v>>depth)%2==1) {

if (tree[tree[now].lc].num)

return find(tree[now].lc,v,depth-1)+a[depth];

else

return find(tree[now].rc,v,depth-1);

} else {

if (tree[tree[now].rc].num)

return find(tree[now].rc,v,depth-1)+a[depth];

else

return find(tree[now].lc,v,depth-1);

}

}

int main() {

int n,i,m,x;

a[0]=1;

for (i=1;i<=30;i++) a[i]=a[i-1]\*2;

scanf("%d",&n);

num=1;

tree[1].num=0;

//插入，查询操作都从数字二进制的第30位开始

for (i=1;i<=n;i++) {

scanf("%d",&x);

insert(1,x,30);

}

scanf("%d",&m);

for (i=1;i<=m;i++) {

int f;

scanf("%d",&x);

f=find(1,x,30);

printf("%d\n",f);

}

return 0;

}

/\*

5

1 8 0 5 14

5

7 14 10 4 2

\*/

# G - 加帕里公园的friends

##### Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 131072/131072KB (Java/Others)

Submit Status

#### 我还有很多话想和她说，还有很多地方想和她去，把KabanKaban酱还给我！——SabaruSabaru



薮猫酱为了从天蓝怪手里拯救小包，必须发现天蓝怪们的弱点所在。  
具体来说，nn只天蓝怪组成了一个序列AA，每一只有一个战斗力数值AiAi，  
之后会发生mm个事件，事件共有两种类型，有可能是  
11、薮猫酱给你一个区间[a,b][a,b]，要你输出max{Ap+Ap+1+…+Aq}(a≤p≤q≤b)max{Ap+Ap+1+…+Aq}(a≤p≤q≤b)  
22、第pospos只天蓝怪的战斗力变成了XX

## Input

第一行是两个整数nn、mm，  
第二行包含nn个整数A1,A2,…,AnA1,A2,…,An，  
接下来mm行，每行三个整数，可能是  
1ab1ab，代表薮猫酱的一次询问；  
2posX2posX，代表某只天蓝怪战斗力的变化。

## Output

对于每次询问，单独输出一行，代表答案。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 5 3  1 2 -3 4 5  1 2 3  2 2 -1  1 2 3 | 2  -1 |

## Hint

1≤n≤5000001≤n≤500000，  
1≤m≤1000001≤m≤100000，  
−1000≤Ai≤1000−1000≤Ai≤1000，  
1≤a≤b≤n1≤a≤b≤n，  
1≤pos≤n1≤pos≤n，  
−1000≤X≤1000−1000≤X≤1000

题意：

对于长度为N的数列，实现区间内数字最大连续和查询、单点修改两个操作。

题解：

首先，对于这类区间查询问题，线段树是首选。

此题中，要求查询区间内最大的连续和，应该怎么办呢？

与B题类似，我们可以在线段树上定义几个标记lms, rms, ms,sum, 分别表示从区间开头开始的最大区间和、以区间末尾结束的最大区间和、区间内最大区间和、区间内所有数字之和，然后从下至上递推出结果。递推公式如下：

Root.lms = max(lchild.lms, lchild.sum+rchild.lms)

Root.rms = max(rchild.rms, rchild.sum+lchild.rms)

Root.ms = max(lchild.ms, rchild.ms, lchild.rms + rchild.lms)

其中root为根节点，lchild为左孩子，rchild为右孩子。

至于单点修改的问题，只要按照普通的线段树区间更新来做即可，只是此时更新区间的l与r相等。

AC代码：

#include <cstdio>

#include <iostream>

using namespace std;

const int maxn=500005;

const int inf=0x3f3f3f3f;

struct Tree {

int l,r,lc,rc,lms,rms,ms,sum;

};

Tree tree[maxn\*4];

int num;

int max(int a,int b) {

if (a>b) return a; else return b;

}

//线段树中向上递推更新

void pushup(int now) {

int l=tree[now].lc,r=tree[now].rc;

tree[now].ms=max(tree[l].ms,tree[r].ms);

tree[now].ms=max(tree[now].ms,tree[l].rms+tree[r].lms);

tree[now].sum=tree[l].sum+tree[r].sum;

tree[now].lms=max(tree[l].lms,tree[l].sum+tree[r].lms);

tree[now].rms=max(tree[r].rms,tree[r].sum+tree[l].rms);

}

//建立线段树

void build(int now,int l,int r) {

tree[now].l=l;

tree[now].r=r;

if (l==r) {

scanf("%d",&tree[now].ms);

tree[now].sum=tree[now].rms=tree[now].lms=tree[now].ms;

} else {

num++;

tree[now].lc=num;

build(num,l,(l+r)/2);

num++;

tree[now].rc=num;

build(num,(l+r)/2+1,r);

pushup(now);

}

}

//线段树单点更新

void update(int now,int l,int r,int c) {

if (l<=tree[now].l&&r>=tree[now].r) {

tree[now].sum=tree[now].lms=tree[now].rms=tree[now].ms=c;

} else {

if (l<=(tree[now].l+tree[now].r)/2)

update(tree[now].lc,l,r,c);

if (r>(tree[now].l+tree[now].r)/2)

update(tree[now].rc,l,r,c);

pushup(now);

}

}

//查找区间最大和

Tree findms(int now,int l,int r) {

if (l<=tree[now].l&&r>=tree[now].r) {

return tree[now];

} else if (r<=tree[tree[now].lc].r) return findms(tree[now].lc,l,r); else

if (l>=tree[tree[now].rc].l) return findms(tree[now].rc,l,r); else {

Tree fl=findms(tree[now].lc,l,r),fr=findms(tree[now].rc,l,r);

Tree f=tree[now];

f.ms=max(fl.ms,fr.ms);

f.ms=max(f.ms,fl.rms+fr.lms);

f.sum=fl.sum+fr.sum;

f.lms=max(fl.lms,fl.sum+fr.lms);

f.rms=max(fr.rms,fr.sum+fl.rms);

return f;

}

}

int main() {

int i,n,m,a,b,x;

scanf("%d%d",&n,&m);

num=1;

build(1,1,n);

for (i=1;i<=m;i++) {

scanf("%d%d%d",&x,&a,&b);

if (x==1) printf("%d\n",findms(1,a,b).ms);

else update(1,a,a,b);

}

return 0;

}

/\*

5

1 2 -3 4 5

3

1 2 3

0 2 -1

1 2 5

\*/

# H - Rikka的烦恼

##### Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 262144/262144KB (Java/Others)

Submit Status

#### 你这快乐的王子，我难道不可以做你的小燕子吗？——HishikawaRikkaHishikawaRikka



RikkaRikka想要和ManaMana在一起，然而ManaMana依然到处收着她的后宫。即使RikkaRikka是CureDiamondCureDiamond，钻石之心也是会受伤的！RikkaRikka只得玩卡片游戏暂时缓解心中的烦恼。  
然而今天RikkaRikka在玩卡片游戏时遇到了麻烦，需要你的帮助。  
所有卡片构成了一个序列，每张卡片上最开始有一个整数。  
RikkaRikka有时候会修改某张卡片上的数字。  
还会询问某一段**下标是等差数列**的子序列的最大值。

## Input

第一行是一个整数nn，  
第二行是一个长度为nn的整数序列aa，第ii个数代表第ii张卡片上初始的数字，  
第三行是一个整数mm，  
接下来mm行，每行首先有一个整数opop，  
然后，若op=0op=0，则之后有两个整数pp，vv，代表将apap的值加上vv，  
若op=1op=1，则之后有两个整数x0x0，dd，代表询问max{ax0,ax0+d,ax0+2d,…,ax0+kd}(x0+(k+1)d>n)max{ax0,ax0+d,ax0+2d,…,ax0+kd}(x0+(k+1)d>n)。

## Output

对每个op=1op=1，单独输出一行，代表该**下标是等差数列**的子序列的最大值。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 10  1 6 1 4 9 4 8 2 8 5  10  1 3 3  0 5 4  0 3 8  1 2 5  1 4 8  1 7 5  1 3 6  0 1 2  1 5 3  1 4 9 | 8  8  4  8  9  13  4 |
| 10  -9 -6 2 -10 -2 -6 10 6 -4 -2  10  1 2 3  1 6 3  0 7 8  0 4 -6  0 10 -5  1 10 4  0 3 -8  1 2 4  0 10 -5  1 1 2 | 6  -4  -7  -6  18 |

## Hint

1≤n≤700001≤n≤70000，  
1≤m≤700001≤m≤70000，  
保证任何时刻abs(ai)≤2147483647(1≤i≤n)abs(ai)≤2147483647(1≤i≤n)，  
0≤op≤10≤op≤1，  
1≤p≤n1≤p≤n，  
abs(v)≤2147483647abs(v)≤2147483647，  
1≤x0≤n1≤x0≤n，  
1≤d≤n1≤d≤n，  
保证涉及的所有数在C++C++的intint内。

题意：

对长度为N的数列实现两个操作：单点修改和区间最大值查询，查询区间是下标为等差数列的不连续区间。

题解：

区间操作，线段树求解。

构造一个值K，当区间下标的公差大于K时暴力求解，公差小于等于K时在线段树上求解。开始时构造1+2+3…+k棵线段树，分别存储下标满足%1=0,%2=0,%2=1,%3=0….%k=k-1时的数据。单点更新时更新满足条件的k棵线段树，查询时查询满足条件的一棵线段树即可。

K应该取多少呢？这是一个玄学问题。

以下是我用同一套代码，仅仅修改程序中常量maxk的值得到的结果：

Maxk 空间 时间

4 14668kb 236ms

5 20128kb 187ms

6 26700kb 219ms

7 34348kb 243ms

8 43096kb 276ms

10 63884kb 329ms

由此可见，K=5时时间复杂度最优，空间占用也相对较少。

AC代码：

#include <cstdio>

#include <iostream>

#include <string.h>

using namespace std;

typedef long long ll;

const int maxn=70005;

const int maxk=7;

const int treenum=maxk\*(maxk+1)/2;

struct Tree {

int l,r,lc,rc;

ll max[treenum];

};

Tree tree[maxn\*3];

int num;

ll a[maxn];

int b[maxk+1][maxk+1];

ll max(ll a,ll b) {

if (a>b) return a; else return b;

}

void build(int now,int l,int r) {

tree[now].l=l;

tree[now].r=r;

if (l==r) {

memset(tree[now].max,-0x2,sizeof(tree[now].max));

scanf("%lld",&a[l]);

for (int i=1;i<=maxk;i++)

tree[now].max[b[i][l%i]]=a[l];

} else {

num++;

tree[now].lc=num;

build(num,l,(l+r)/2);

num++;

tree[now].rc=num;

build(num,(l+r)/2+1,r);

for (int i=0;i<treenum;i++)

tree[now].max[i]=max(tree[tree[now].lc].max[i],tree[tree[now].rc].max[i]);

}

// cout << now << ' ' << tree[now].l << ' ' << tree[now].r << ' ' << tree[now].max[3] << endl;

}

void update(int now,int pos,int c) {

if (pos==tree[now].l&&pos==tree[now].r) {

for (int i=1;i<=maxk;i++)

tree[now].max[b[i][pos%i]]+=c;

} else {

if (pos<=(tree[now].l+tree[now].r)/2)

update(tree[now].lc,pos,c);

if (pos>(tree[now].l+tree[now].r)/2)

update(tree[now].rc,pos,c);

for (int i=1;i<=maxk;i++)

tree[now].max[b[i][pos%i]]=max(tree[tree[now].lc].max[b[i][pos%i]],tree[tree[now].rc].max[b[i][pos%i]]);

}

}

ll findmax(int now,int l,int r,int sn) {

// cout << now << ' ' << tree[now].l << ' ' << tree[now].r << ' ' << tree[now].max[sn] << endl;

if (l<=tree[now].l&&r>=tree[now].r) {

return tree[now].max[sn];

} else if (r<=tree[tree[now].lc].r) return findmax(tree[now].lc,l,r,sn); else

if (l>=tree[tree[now].rc].l) return findmax(tree[now].rc,l,r,sn); else

return max(findmax(tree[now].lc,l,r,sn),findmax(tree[now].rc,l,r,sn));

}

int main() {

int i,j,n,m,c,x,y;

scanf("%d",&n);

num=0;

for (i=1;i<=maxk;i++)

for (j=0;j<i;j++)

b[i][j]=num++;

num=1;

build(1,1,n);

scanf("%d",&m);

for (i=1;i<=m;i++) {

scanf("%d%d%d",&x,&y,&c);

if (x==1) {

if (c<=maxk)

printf("%lld\n",findmax(1,y,n,b[c][y%c]));

else {

ll ans=-0x3f3f3f3f3f;

for (j=y;j<=n;j+=c)

ans=max(ans,a[j]);

printf("%lld\n",ans);

}

}

else {

a[y]+=c;

update(1,y,c);

}

}

return 0;

}

/\*

10

1 6 1 4 9 4 8 2 8 5

10

1 3 3

0 5 4

0 3 8

1 2 5

1 4 8

1 7 5

1 3 6

0 1 2

1 5 3

1 4 9

\*/

**J - wtmsb**

**Time Limit: 1000/100MS (Java/Others)     Memory Limit: 131072/131072KB (Java/Others)**

Submit Status

这天，AutSky\_JadeKAutSky\_JadeK看到了nn张图片,他忍不住说道：“我TMTM社保！”。

  
每张图片有一个社保值，他可以合并两张图片，合并所得的图片的社保值是原来两张图片的社保值之和。  
每次合并需要消耗的体力也是原来两张图片的社保值之和。 显然，n−1次合并之后，只剩下一张图片。  
他想知道，在这个过程中，他消耗的总体力最少是多少。

**Input**

第一行包含一个整数nn，  
第二行包含nn个整数A1,A2,…,AnA1,A2,…,An，代表nn张图片的社保值。

**Output**

输出一行，包含一个整数，代表他消耗的总体力最少是多少。

**Sample input and output**

| **Sample Input** | **Sample Output** |
| --- | --- |
| 3  1 2 9 | 15 |

**Hint**

1≤n≤200001≤n≤20000，  
1≤Ai≤20001≤Ai≤2000

题意：

一串数列有N个数，每次操作可以合并任意两个数，代价是两个数的和，询问合并N-1次后的最小代价。

题解：

贪心，每次将最小的两个数合并。

如何快速找到最小的两个数呢？可以用小根堆。因为C++内部已经存在优先队列的模板，所以可以直接利用标准模板库找到两个最小的元素。

AC代码：

#include <cstdio>

#include <iostream>

#include <queue>

using namespace std;

//优先队列

priority\_queue<int, vector<int>, greater<int> > q;

int main() {

int a,b,n,x,ans,i;

scanf("%d",&n);

for (i=1;i<=n;i++) {

scanf("%d",&x);

q.push(x);

}

ans=0;

for (i=1;i<=n-1;i++) {

ans+=a=q.top();

q.pop();

ans+=b=q.top();

q.pop();

q.push(a+b);

}

printf("%d",ans);

return 0;

}

# K - 老司机破阵

##### Time Limit: 4500/1500MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)

Submit Status

老司机的女朋友被坏人抓起来了，护女票心切的老司机心急火燎的赶到坏人藏匿女票的地点想要救出她，但是老司机的神通广大坏人们早有耳闻，等老司机赶到时已经有一个法阵摆在他的面前阻挡着他。  
法阵是一条直线上的**n**个法力元件构成的，老司机每次可以将一个法力元件击碎，法阵的能量就是所有 **连贯的元件能量和的最大值** 。  
老司机非常的自信，他有一套自己的破除法阵的方案  
**老司机希望能实时的关注着法阵的能量**，一旦能量允许，他就破阵而入，救出女票。  
忙着破阵的老司机自然没有功夫去计算他每步操作之后法阵的能量，他只能将此重任交与在座的各位大侠，请大家助他一臂之力。

## Input

第一行n (1 ≤ n ≤ 100,000)，为法力元件数量  
第二行有n个数，为每个法力元件所含有的能量ei(0 ≤ ei ≤ 1e9)  
第三行有n个数，为老司机击破法力元件的顺序

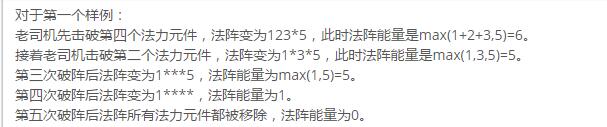
## Output

输出n行，为老司机每次击破法力元件后法阵的能量。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 5  1 2 3 4 5  4 2 3 5 1 | 6  5  5  1  0 |
| 8  5 5 4 4 6 6 5 5  5 2 8 7 1 3 4 6 | 18  16  11  8  8  6  6  0 |

## Hint



题意：

对于一个长度为N的数列，进行单点删除、最大区间和查询两种操作。

题解：

这题同样涉及区间和，应想到 用线段树求解。

最大区间和查询同G题类似，可以在线段树中通过递归解决。

怎样处理单点删除呢？这里我使用了一个技巧：不直接删除该点，而是把该点的值修改为负无穷，这样从下至上递归最优结果时就不可能取到该点。这题其他操作都与G题相似，不再赘述。

AC代码：

#include <cstdio>

#include <iostream>

#include <string.h>

#include <algorithm>

using namespace std;

const int maxn=100500;

const long long inf=1e13;

typedef long long ll;

int num;

struct Tree {

int lc,rc,l,r;

ll sum,ms,lms,rms;

};

Tree tree[4\*maxn];

ll max (ll a,ll b) {

if (a>b) return a; else return b;

}

//向上递推结果

void pushup(int now) {

int l=tree[now].lc,r=tree[now].rc;

tree[now].sum=tree[l].sum+tree[r].sum;

tree[now].ms=max(tree[l].ms,tree[r].ms);

tree[now].ms=max(tree[now].ms,tree[l].rms+tree[r].lms);

tree[now].lms=max(tree[l].lms,tree[l].sum+tree[r].lms);

tree[now].rms=max(tree[r].rms,tree[r].sum+tree[l].rms);

}

//建立线段树

void build(int now,int l,int r) {

tree[now].l=l;

tree[now].r=r;

if (l!=r) {

num++;

tree[now].lc=num;

build(num,l,(l+r)/2);

num++;

tree[now].rc=num;

build(num,(l+r)/2+1,r);

pushup(now);

} else {

scanf("%lld",&tree[now].sum);

tree[now].ms=tree[now].lms=tree[now].rms=tree[now].sum;

}

}

//区间更新（删除），置更新的点为负无穷

void update (int now,int l,int r) {

// if (l==21723)cout << tree[now].sum << ' ' << tree[now].l << ' ' << tree[now].r << ' ' << tree[now].lms << ' ' << tree[now].rms << ' ' << tree[now].ms << endl;

if (tree[now].l>=l&&tree[now].r<=r) {

tree[now].ms=tree[now].lms=tree[now].rms=tree[now].sum=-inf;

} else {

if (l<=(tree[now].l+tree[now].r)/2)

update(tree[now].lc,l,r);

if (r>(tree[now].l+tree[now].r)/2)

update(tree[now].rc,l,r);

pushup(now);

}

}

int main() {

int n,i,x;

scanf("%d",&n);

num=1;

build(1,1,n);

for (i=1;i<n;i++) {

scanf("%d",&x);

update(1,x,x);

printf("%lld\n",tree[1].ms);

}

printf("0");

return 0;

}

**L - 老司机的奇幻漂流**

**Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)**

Submit Status

老司机在救出了女票之后，就和她在全世界旅游，有一天，他们来到了一个神奇的小岛上。  
这个小岛上有三种动物，他们互相克制，就像石头剪刀布一样，他们形成了一个食物环。  
热爱小动物的老司机女票已经对这个岛上的N只动物进行了细致的研究，对他们所属的族群也了如指掌，并对他们的行为做了记录。  
第一种记录方式是1 X Y，表示X和Y关系密切（是同类）。  
第二种记录方式是2 X Y，表示X对Y有攻击性行为（X克制Y）。  
不过很不凑巧，这些记录被毛手毛脚的JJ给打乱了，使得其中一些记录出现了错误。  
为了安慰着急的女票，老司机急忙来帮忙查看是哪些记录出现了错误，当一条记录满足以下三种可能***之一***时，这就是一条错误的记录。  
1）这条记录中的X或者Y比N大  
2）攻击性记录中的X攻击了X  
3）这条记录与前面的记录相悖  
老司机说是来帮忙，但是还要忙着跑去和wyy踢fifa，所以就让你来完成这个任务，找出这些记录中哪些错了。

**Input**

第一行两个整数N和M，为记录的动物数量和记录条数 接下来M行，每行代表一条记录。

**Output**

一行n个数字，表示哪几条记录有误，用空格隔开  
按编号升序输出

**Sample input and output**

| **Sample Input** | **Sample Output** |
| --- | --- |
| 100 7  1 101 1  2 1 2  2 2 3  2 3 3  1 1 3  2 3 1  1 5 5 | 1 4 5 |

**Hint**

数据范围：  
1<= N <= 50,000  
1<= M <=100,000

题意：有N个动物，编号1-N，他们属于A、B、C三类，A吃B，B吃C，C吃A，现在给出这些动物的一些同类关系或互吃关系，判断这些关系当中哪些不合理。如果某一关系满足如下三个标准之一：

1. 动物编号大于N
2. 这一关系在逻辑上与之前的关系矛盾
3. 某一动物自己吃自己

则它是不合理的。

题解：

很经典的一道关系并查集题目。

首先，某两个动物可能有关系，也可能关系不确定。怎么表示这种关系呢？我们可以用0表示同类，1表示吃，2表示被吃，这样表示两个动物的相对关系时，可以根据已有的关系推理得到他们的关系，只要将它们的值相加对3求余即可。

AC代码：

#include <cstdio>

#include <iostream>

#include <string.h>

using namespace std;

const int maxn=50005;

int f[maxn],r[maxn];

//并查集的查找，在同一棵树上则代表两个节点直接已经有了关系，否则没有关系

int find(int now) {

if (f[now]==now) return now; else {

int fa=f[now];

f[now]=find(f[now]);

r[now]=r[now]+r[fa];

r[now]%=3;

return f[now];

}

}

int main() {

int n,k,i,j,d,x,y,ans=0;

scanf("%d%d",&n,&k);

memset(r,0,sizeof(r));

for (i=1;i<=n;i++) f[i]=i;

for (i=1;i<=k;i++) {

scanf("%d%d%d",&d,&x,&y);

if ((x==y&&d==2)||(x>n)||(y>n)) cout << i << ' '; else{

int fa=find(x);

int fb=find(y);

//判断读入的动物是否已有关系，如果有直接根据现有关系判断，如果没有则利用并查集合并

if (fa==fb) {

if (d==1) {

if (r[x]!=r[y]) cout << i << ' ';

}

else {

if ((r[y]+3-r[x])%3!=1) cout << i << ' ';

}

} else {

f[fb]=fa;

r[fb]=3-r[y]+r[x]+d-1;

r[fb]%=3;

}

}

}

return 0;

}

**N - 可以说是非常豹笑了**

**Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)**

Submit Status



你总是说：“可以说是非常豹笑了”

可是非常豹已经很累了，

非常豹不想笑，

它想念自己的家人，

而不是被你逼着笑。

你关心过这些吗?

没有！你只关心你自己。

**Input**

第一行输入一个正整数N（2<=N<=1000000）表示总天数，一个正整数M（1<=M<=1000000）表示给出的关系数。

接下来M行，每行三个整数T（T=1 or 0）、A、B（1<=A,B<=N）。T=1表示在第A天和第B天非常豹的表情一样、T=0表示在第A天和第B天非常豹的表情不一样。

非常豹在一天里要么笑了，要么没笑。

**Output**

输出 'YES’ 如果输入是可能的话；

输出'NO' 如果无论怎样都不可能满足输入给出的关系。

**Sample input and output**

| **Sample Input** | **Sample Output** |
| --- | --- |
| 2 2  1 1 2  0 2 1 | NO |
| 2 2  1 1 1  0 2 1 | YES |

**Hint**

没有多组数据

题意：

有N天，每天可以并且只能选择给定的两种状态之一。给出这些天数之间状态的相同或不同的关系，判断它们是否矛盾。

题解：

与L题类似，使用关系并查集求解。本题只有两种状态，因此推理过程相比之下简单不少。

#include <cstdio>

#include <iostream>

#include <string.h>

using namespace std;

const int maxn=1000005;

int f[2\*maxn],r[maxn];

//并查集查找操作

int find(int now) {

if (f[now]==now) return now; else {

int t=f[now];

f[now]=find(f[now]);

r[now]=(r[now]+r[t])%2;

return f[now];

}

}

int main() {

int i,j,a,b,n,m,num=0;

scanf("%d%d",&n,&m);

memset(r,0,sizeof(r));

for (i=1;i<=n;i++) f[i]=i;

for (i=1;i<=m;i++) {

scanf("%d%d%d",&j,&a,&b);

if (j==0) {

int fa=find(a);

int fb=find(b);

//如果已有关系则判断，没有关系则建立关系，将两棵树合并为一棵树

if (fa==fb) {

if ((r[a]+r[b])%2==0) {

cout << "NO";

return 0;

}

} else {

f[fb]=fa;

r[fb]=r[b]+r[a]+1;

r[fb]%=2;

}

} else {

int fa=find(a);

int fb=find(b);

if (fa==fb) {

if ((r[a]+r[b])%2) {

cout << "NO";

return 0;

}

} else {

f[fb]=fa;

r[fb]=r[a]+r[b];

r[fb]%=2;

}

}

}

cout << "YES" << endl;

return 0;

}

**P - 艾尔大停电**

**Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)**

Submit Status

由于星灵的所有建筑都必须放置在水晶塔的能量场中，并由水晶塔提供能量才能工作，所以一旦水晶塔出了问题（比如被打爆），艾尔就会大停电。



大主教阿塔尼斯认为水晶塔不合理的圆形能量场是效率低下的关键，因为建筑都是方的，圆形很浪费而且无法镶嵌。所以他开发了一种矩形能量场的水晶塔以解决停电问题。

现在他要测试一下成果，在平面上分布了一些矩形的能量场，他想要知道可以建造建筑的总面积是多少（不考虑水晶塔占据的面积）。

**Input**

本题有多组数据

每组数据第一行给出一个N（1<=N<=100）表示能量场的数量。

接下来N行描述这些矩形能量场，每行四个浮点数 x1;y1;x2;y2 (0<=x1<x2<=100000;0<=y1<y2<=100000) 表示矩形的在平面直角坐标系上的坐标(x1,y1)为左下角，(x2,y2)右上角。所有矩形都平行于坐标系。

输入以单独的一行，一个0结束。

**Output**

对于每组数据输出一行一个实数，表示能量场总面积。答案保留两位小数。

**Sample input and output**

| **Sample Input** | **Sample Output** |
| --- | --- |
| 2  10 10 20 20  15 15 25 25.5  0 | 180.00 |

题意：

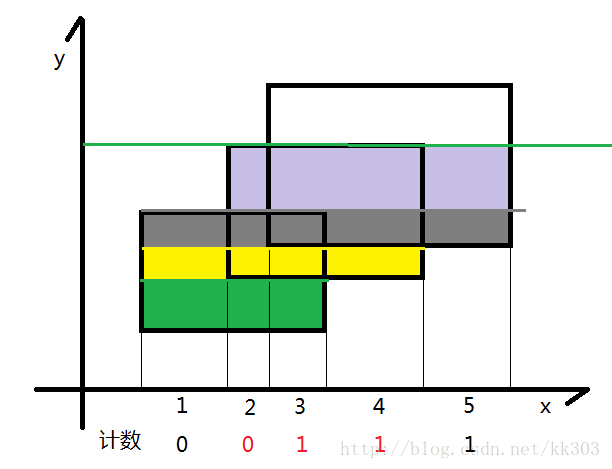
在坐标系中有一些长方形，给定它们的位置，求它们的总面积。重叠的部分只算一次。

题解：

这道题与POJ上的一道题目极为相似。

Poj1151 Atlantis <http://poj.org/problem?id=1151>

对于这一类问题，我们可以利用扫描线法来解决。



如图，我们将整个区域分为一个个的小矩形，扫描线从下至上上升，最后把每次计算得到的面积加起来就是答案。

遇到的一个问题就是：本题中所给坐标不一定为整数，该如何处理呢？

我们可以将这些值离散化，先对他们排序，再从小到大编号为1,2,3…n.在线段树中，l,r用来存储离散化之后的整数区间，而ly,ry用来存储这些整数所代表的真实的Y坐标值。

参考资料：

<http://www.2cto.com/kf/201307/231714.html>

POJ 1151 - Atlantis 线段树+扫描线.. - c++语言程序开发技术文章\_c++编程 - 红黑联盟

//线段树求矩形面积

//扫描线法

#include <cstdio>

#include <iostream>

#include <algorithm>

using namespace std;

struct Tree {

int l,r,lc,rc;

double ly,ry,len;

int cover;

};

struct Line {

double x,y1,y2;

int flag;

};

Tree tree[605];

Line line[205];

double y[205];

int co;

bool cmp(Line a,Line b) {

return a.x<b.x;

}

//建立线段树

void build(int now,int l,int r) {

tree[now].l=l;tree[now].r=r;

tree[now].ly=y[l];tree[now].ry=y[r];

tree[now].len=tree[now].cover=0;

if (l+1!=r) {

co++;

tree[now].lc=co;

build(co,l,(l+r)/2);

co++;

tree[now].rc=co;

build(co,(l+r)/2,r);

}

}

//计算有效线段长度

void cal(int now) {

if (tree[now].cover>0) {

tree[now].len=tree[now].ry-tree[now].ly;

} else {

if (tree[now].l+1==tree[now].r)

tree[now].len=0;

else tree[now].len=tree[tree[now].lc].len+tree[tree[now].rc].len;

}

}

//更新线段覆盖情况，cover为正数线段有效，否则无效

void update(int now,Line l) {

// cout << now << ' ' << tree[now].cover << ' ' << tree[now].ly << ' ' << tree[now].ry << endl;

// cout << l.x << ' ' << l.y1 << ' ' << l.y2 << endl;

if (tree[now].ly==l.y1&&tree[now].ry==l.y2) {

tree[now].cover+=l.flag;

cal(now);

return;

} else if (tree[tree[now].lc].ry>=l.y2) update(tree[now].lc,l);

else if (tree[tree[now].rc].ly<=l.y1) update(tree[now].rc,l); else{

Line l1=l;

l1.y2=tree[tree[now].lc].ry;

update(tree[now].lc,l1);

l1=l;

l1.y1=tree[tree[now].rc].ly;

update(tree[now].rc,l1);

}

cal(now);

}

int main() {

int n,i;

double x1,x2,y1,y2;

scanf("%d",&n);

int time=0;

while (n) {

time++;

int num=0;

for (i=1;i<=n;i++) {

scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);

num++;

line[num].x=x1;

line[num].y1=y1;

line[num].y2=y2;

line[num].flag=1;

num++;

line[num].x=x2;

line[num].y1=y1;

line[num].y2=y2;

line[num].flag=-1;

y[2\*i-1]=y1;y[2\*i]=y2;

}

sort(line+1,line+2\*n+1,cmp);

sort(y+1,y+2\*n+1);

co=1;

build(1,1,2\*n);

update(1,line[1]);

double ans=0;

for (i=2;i<=2\*n;i++) {

ans+=tree[1].len\*(line[i].x-line[i-1].x);

update(1,line[i]);

}

printf("%.2f\n",ans);

scanf("%d",&n);

}

return 0;

}

# Q - 艾尔大停电2

##### Time Limit: 3000/1000MS (Java/Others)     Memory Limit: 65535/65535KB (Java/Others)

Submit Status

（剧情提要：请看上题）

阿塔尼斯的新式矩形能量场成功的研制出来了。但是实验过程中却发现了一个巨大的问题，一股神秘的东方力量影响了新式能量场，两个能量场互相覆盖的区域将会互相抵消，变回无能量场区。

有趣的是，如果这个区域被第三个能量场覆盖，它将仍是有效的能量场区，当然如果有第四个能量场，那么就会和第三个能量场抵消，又变成无能量场区……也就是说，一个能量场将会把它范围内的能量场区变为非能量场区，把非能量场区变为能量场区。

现在，阿塔尼斯正在建造一些能量场，你需要知道某些时刻某个坐标点上是否是有效的能量场区。

## Input

（注意能量场形式与上题的不同）

本题有多组数据。第一行一个正整数T（T<=10）表示数据的组数。

每组数据第一行一个正整数N(N<=50000)，表示操作的次数。

接下来N行形容按时间顺序进行的N次操作，每行可能有两种形式：

1、一个大写字母C和四个正整数 C x1 y1 x2 y2 (1 <= x1 <= x2 <= 1000, 1 <= y1 <= y2 <= 1000) 表示以(x1,y1)为左下角(x2,y2)为右上角建造一个平行坐标轴的矩形能量场（包括边界）。

2、一个大写字母Q和两个正整数 Q x y (1 <= x, y <= 1000) 表示询问坐标(x,y)是否是有效的能量场区。

## Output

对于每个询问输出一行一个整数。如果是有效的能量场区输出1，否则输出0。

## Sample input and output

| **Sample Input** | **Sample Output** |
| --- | --- |
| 1  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 | 1  0  0  1 |

题意：

二维空间内，有一个由0和1组成的矩阵，开始时矩阵全为0，每次操作选择一个子矩阵将里面所有数字取反，动态查询矩阵中某一个元素的值。

题解：

这是一道很经典的二维树状数组题目，我居然没做过，看来我还是太菜了。解法很简单，将树状数组扩展到二维即可。在矩阵左下、右上角+1，左上、右下角-1，查询时对某元素的坐标求和即可。

AC代码：

#include <cstdio>

#include <iostream>

#include <algorithm>

#include <string.h>

using namespace std;

typedef long long ll;

const int maxn=2000;

ll f[maxn][maxn];

int N,M;

char s[10];

int lowbit(int v) {

return (v&(-v));

}

//二维树状数组更新

void update (int n,int m,ll c) {

int i=n,j=m;

for (i=n;i<=N;i+=lowbit(i))

for (j=m;j<=N;j+=lowbit(j))

f[i][j]+=c;

}

//二维树状数组求和

ll findsum(int n,int m) {

int i=n,j=m;

ll sum=0;

for (i=n;i>0;i-=lowbit(i))

for (j=m;j>0;j-=lowbit(j))

sum+=f[i][j];

return sum;

}

int main() {

int n,i,t,q;

int x1,x2,y1,y2;

char c;

scanf("%d",&t);

N=2000;

for(q=1;q<=t;q++) {

memset(f,0,sizeof(f));

scanf("%d",&M);

for (i=1;i<=M;i++) {

scanf("%s",s);

if (s[0]=='C') {

scanf("%d%d%d%d",&x1,&y1,&x2,&y2);

//更新子矩阵的四个角

update(x1,y1,1);

update(x2+1,y2+1,1);

update(x1,y2+1,-1);

update(x2+1,y1,-1);

} else {

scanf("%d%d",&x1,&y1);

//查询即求和

ll s=findsum(x1,y1);

if (s%2) printf("1\n"); else printf("0\n");

}

}

}

return 0;

}