|  |
| --- |
| ACM-ICPC模板 |
| F:\ACM\download.png |

目录

[**数据结构：** 4](#_Toc434361898)

[树链剖分 4](#_Toc434361899)

[LCT动态树 8](#_Toc434361900)

[LCT\_Kuangbin 12](#_Toc434361901)

[RMQ(区间最值，区间GCD) 17](#_Toc434361902)

[可持久化字典树\_HDU\_4757 18](#_Toc434361903)

[树状数组 21](#_Toc434361904)

[线段树终极版 24](#_Toc434361905)

[主席树-区间K大-POJ-2104 27](#_Toc434361906)

[主席树-区间修改hihocoder couple tree 29](#_Toc434361907)

[**数论** 32](#_Toc434361908)

[朴素筛素数 32](#_Toc434361909)

[欧拉函数：求小于n的有多少个数与n互质 33](#_Toc434361910)

[快速幂运算（模） 34](#_Toc434361911)

[康拓展开hash阶乘 34](#_Toc434361912)

[高斯消元(异或) 35](#_Toc434361913)

[分数类 36](#_Toc434361914)

[大素数判断(无random) 37](#_Toc434361915)

[高精度(字符串) 38](#_Toc434361916)

[大数(精简版) 50](#_Toc434361917)

[Pollard\_rho进行质因数分解 58](#_Toc434361918)

[NTT快速数论变换 59](#_Toc434361919)

[Miller\_Rabin算法进行素数测试(random) 62](#_Toc434361920)

[Lucas定理-求组合数取模 64](#_Toc434361921)

[Hiho1230 FWT快速沃尔什变换 65](#_Toc434361922)

[**图论及树** 67](#_Toc434361923)

[最小生成树Prim 67](#_Toc434361924)

[次小生成树 68](#_Toc434361925)

[HDU 4453 Splay 70](#_Toc434361926)

[Floyd最短路 75](#_Toc434361927)

[Dijkstra最短路 77](#_Toc434361928)

[Astar求K短路 77](#_Toc434361929)

[最小费用最大流(SPFA\_rujia) 79](#_Toc434361930)

[ZKW费用流 80](#_Toc434361931)

[最小顶点覆盖/二分图最大匹配 83](#_Toc434361932)

[最近公共祖先 86](#_Toc434361933)

[最大流(ISAP-rujia) 88](#_Toc434361934)

[一般匹配-带花树开花算法 93](#_Toc434361935)

[匈牙利算法 96](#_Toc434361936)

[全局最小割-HDU 3691 97](#_Toc434361937)

[判断是否为二分图(是否有奇圈) 98](#_Toc434361938)

[欧拉回路 99](#_Toc434361939)

[计算双联通分量 100](#_Toc434361940)

[二分图最佳完美匹配(KM匈牙利算法) 102](#_Toc434361941)

[点双联通缩点 104](#_Toc434361942)

[并查集 105](#_Toc434361943)

[SPOJ QTREE5 树分治+优先队列 106](#_Toc434361944)

[SPFA 109](#_Toc434361945)

[2-SAT 111](#_Toc434361946)

[**字符串** 114](#_Toc434361947)

[AC自动机 114](#_Toc434361948)

[字符串的最小表示法(环串中的最小字典序) 116](#_Toc434361949)

[扩展KMP 117](#_Toc434361950)

[快速hash值计算 118](#_Toc434361951)

[可持久化字典树HDU4757 118](#_Toc434361952)

[回文树 121](#_Toc434361953)

[后缀数组 122](#_Toc434361954)

[Trie树 124](#_Toc434361955)

[KMP算法 125](#_Toc434361956)

[**杂板子** 128](#_Toc434361957)

[三分 128](#_Toc434361958)

[手写递归栈 128](#_Toc434361959)

[数位DP 129](#_Toc434361960)

[五子棋判断某个位置是否连成5个 131](#_Toc434361961)

[一些大质数 132](#_Toc434361962)

[C++各个类型输入符控制 132](#_Toc434361963)

[C++扩栈 132](#_Toc434361964)

[Cin取消同步 133](#_Toc434361965)

[G++扩栈 133](#_Toc434361966)

[Headfile 134](#_Toc434361967)

[Unordered\_map 135](#_Toc434361968)

[Unordermap简化版 135](#_Toc434361969)

[读入挂 136](#_Toc434361970)

[多重背包二进制 138](#_Toc434361971)

[矩阵快速幂 138](#_Toc434361972)

[离散化 140](#_Toc434361973)

[**计算几何** 140](#_Toc434361974)

[最近点对问题 140](#_Toc434361975)

[凸包 142](#_Toc434361976)

[求多边形重心 142](#_Toc434361977)

[计算几何常规模板 143](#_Toc434361978)

[多边形圆交面积 146](#_Toc434361979)

[点集的直径 147](#_Toc434361980)

[半平面交 148](#_Toc434361981)

## 数据结构：

### 树链剖分

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Author : kuangbin

Created Time: 2013/8/11 22:00:02

File Name : F:\2013ACM练习\专题学习\数链剖分\SPOJ\_QTREE.cpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

#include <stdio.h>

#include <string.h>

#include <iostream>

#include <algorithm>

#include <vector>

#include <queue>

#include <set>

#include <map>

#include <string>

#include <math.h>

#include <stdlib.h>

using namespace std;

**const** **int** MAXN = 10010;

**int** top[MAXN];//top[v]表示v所在的重链的顶端节点

**int** fa[MAXN]; //父亲节点

**int** deep[MAXN];//深度

**int** num[MAXN];//num[v]表示以v为根的子树的节点数

**int** p[MAXN];//p[v]表示v与其父亲节点的连边在线段树中的位置

**int** fp[MAXN];//和p数组相反

**int** son[MAXN];//重儿子

**int** pos;

struct Edge

{

**int** to,next;

}edge[MAXN\*2];

**int** head[MAXN],tot;

**void** init()

{

tot = 0;

memset(head,-1,sizeof(head));

pos = 0;

memset(son,-1,sizeof(son));

}

**void** addedge(**int** u,**int** v)

{

edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;

}

////////////////////线段树////////////////////

#define lson(x) (x)<<1

#define rson(x) ((x)<<1)|1

struct Node

{

**int** l,r;

**int** num;

Node(**int** ll=0,**int** rr=0,**int** n=0){

l = ll,r = rr,num = n;

}

//bool flag;

}st[N\*6];

struct Seg{

**void** bulid(**int** idx,**int** l,**int** r)

{

st[idx] = Node(l,r,0);

**if**(l == r)

{

st[idx].num = w[fp[l]];

**return** ;

}

**int** mid = (l + r) / 2;

bulid(lson(idx),l,mid);

bulid(rson(idx),mid+1,r);

}

**void** pushdown(**int** idx)

{

**int** tmp = st[idx].num;

**if**(tmp)

{

st[lson(idx)].num += tmp;

st[rson(idx)].num += tmp;

}

st[idx].num = 0;

}

**void** update(**int** idx, **int** l, **int** r, **int** value)

{

**if**(st[idx].l >= l && st[idx].r <= r)

{

st[idx].num += value;

**return** ;

}

pushdown(idx);

**int** mid = (st[idx].l+st[idx].r)/2;

**if**(r <= mid) update(lson(idx),l,r,value);

**else** **if**(l > mid) update(rson(idx),l,r,value);

**else**{

update(lson(idx),l,mid,value);

update(rson(idx),mid+1,r,value);

}

}

**int** query(**int** idx, **int** k)

{

**if**(st[idx].l == st[idx].r && st[idx].l == k)

{

//if(st[idx].num)

**return** st[idx].num;

}

pushdown(idx);

**int** mid = (st[idx].l + st[idx].r)/2;

**if**(k <= mid) **return** query(lson(idx),k);

**else** **return** query(rson(idx),k);

}

}seg;

////////////////////树链剖分////////////////////

struct TreeLink{

**void** build(){

dfs1(1,0,0);

getpos(1,1);

}

**int** find(**int** u,**int** v)//查询u->v边的最大值

{

**int** f1 = top[u], f2 = top[v];

**int** tmp = 0;

**while**(f1 != f2)

{

**if**(deep[f1] < deep[f2])

{

swap(f1,f2);

swap(u,v);

}

tmp = max(tmp,seg.query(1,p[f1],p[u]));

u = fa[f1]; f1 = top[u];

}

**if**(u == v)**return** tmp;

**if**(deep[u] > deep[v]) swap(u,v);

**return** max(tmp,seg.query(1,p[son[u]],p[v]));//如果属性在点上，那这里的p[son[u]]需要修改为p[u].

}

/\*Useless\*/

**void** dfs1(**int** u,**int** pre,**int** d) //第一遍dfs求出fa,deep,num,son

{

deep[u] = d;

fa[u] = pre;

num[u] = 1;

**for**(**int** i = head[u];i != -1; i = edge[i].next)

{

**int** v = edge[i].to;

**if**(v != pre)

{

dfs1(v,u,d+1);

num[u] += num[v];

**if**(son[u] == -1 || num[v] > num[son[u]])

son[u] = v;

}

}

}

**void** getpos(**int** u,**int** sp) //第二遍dfs求出top和p

{

top[u] = sp;

**if**(son[u] != -1)

{

p[u] = pos++;

fp[p[u]] = u;

getpos(son[u],sp);

}

**else**

{

p[u] = pos++;

fp[p[u]] = u;

**return**;

}

**for**(**int** i = head[u] ; i != -1; i = edge[i].next)

{

**int** v = edge[i].to;

**if**(v != son[u] && v != fa[u])

getpos(v,v);

}

}

}trlnk;

**int** e[MAXN][3];

**int** main()

{

//freopen("in.txt","r",stdin);

//freopen("out.txt","w",stdout);

**int** T;

**int** n;

scanf("%d",&T);

**while**(T--)

{

init();

scanf("%d",&n);

**for**(**int** i = 0;i < n-1;i++)

{

scanf("%d%d%d",&e[i][0],&e[i][1],&e[i][2]);

addedge(e[i][0],e[i][1]);

addedge(e[i][1],e[i][0]);

}

trlnk.build();

seg.build(1,0,pos-1);

**for**(**int** i = 0;i < n-1; i++)

{

**if**(deep[e[i][0]] > deep[e[i][1]])

swap(e[i][0],e[i][1]);

seg.update(1,p[e[i][1]],e[i][2]);

}

**char** op[10];

**int** u,v;

**while**(scanf("%s",op) == 1)

{

**if**(op[0] == 'D')**break**;

scanf("%d%d",&u,&v);

**if**(op[0] == 'Q')

printf("%d\n",trlnk.find(u,v));

**else** seg.update(1,p[e[u-1][1]],v);

}

}

**return** 0;

}

### LCT动态树

#include <cstdio>

#include <algorithm>

#include <iostream>

#include <string.h>

#include <stdio.h>

**const** **int** maxn=100011;

**const** **int** INF=0x7fffffff;

using namespace std;

struct SplayTree

{

**int** val,mn,lazy;

bool remark;

**int** ch[2],pre;

};

SplayTree \*tree;

**int** N;

**int** val[maxn];

**void** Init\_Splay(**int** x)

{

tree[x].ch[0]=tree[x].ch[1]=tree[x].pre=0;

tree[x].remark=0;

tree[x].val=val[x];

tree[x].mn=val[x];

}

bool IsRoot(**int** x)

{

**return** !tree[x].pre || (tree[tree[x].pre].ch[0]!=x && tree[tree[x].pre].ch[1]!=x);

}

**void** DynamicTree(**int** n)

{

N=n;

tree=**new** SplayTree[n+1];

**for**(**int** i=0; i<=n; i++) Init\_Splay(i);

tree[0].val=-INF;

tree[0].mn=-INF;

}

**void** Inc(**int** x,**int** d)

{

tree[x].val+=d;

tree[x].mn+=d;

tree[x].lazy+=d;

}

**void** Rev(**int** x)

{

swap(tree[x].ch[0],tree[x].ch[1]);

tree[x].remark^=1;

}

**void** PushDown(**int** x)

{

**if**(!x) **return**;

**if**(tree[x].lazy)

{

**if**(tree[x].ch[0]) Inc(tree[x].ch[0],tree[x].lazy);

**if**(tree[x].ch[1]) Inc(tree[x].ch[1],tree[x].lazy);

tree[x].lazy=0;

}

**if**(tree[x].remark)

{

**if**(tree[x].ch[0]) Rev(tree[x].ch[0]);

**if**(tree[x].ch[1]) Rev(tree[x].ch[1]);

tree[x].remark=0;

}

}

**void** Update(**int** x)

{

**if**(!x) **return**;

tree[x].mn=tree[x].val;

**if**(tree[x].ch[0]) tree[x].mn=max(tree[tree[x].ch[0]].mn,tree[x].mn);

**if**(tree[x].ch[1]) tree[x].mn=max(tree[tree[x].ch[1]].mn,tree[x].mn);

}

**void** Rotate(**int** p,**int** c)

{

**int** x=tree[p].pre,y=tree[x].pre;

tree[p].pre=y;

tree[x].pre=p;

**if**(y) **if**(x==tree[y].ch[0]) tree[y].ch[0]=p;

**else** **if**(x==tree[y].ch[1]) tree[y].ch[1]=p;

tree[x].ch[!c]=tree[p].ch[c];

**if**(tree[x].ch[!c]) tree[tree[x].ch[!c]].pre=x;

tree[p].ch[c]=x;

Update(x);

}

**int** stack[maxn];

**void** Splay(**int** x)

{

**int** top=1;

stack[0]=x;

**for**(**int** q=x; !IsRoot(q);) stack[top++]=(q=tree[q].pre);

**while**(top) PushDown(stack[--top]);

**while**(!IsRoot(x))

{

**int** q=tree[x].pre;

**if**(IsRoot(q)) **if**(tree[q].ch[0]==x) Rotate(x,1);

**else** Rotate(x,0);

**else**

{

**if**(q==tree[tree[q].pre].ch[0])

**if**(tree[q].ch[0]==x) Rotate(q,1),Rotate(x,1);

**else** Rotate(x,0),Rotate(x,1);

**else** **if**(x==tree[q].ch[1]) Rotate(q,0),Rotate(x,0);

**else** Rotate(x,1),Rotate(x,0);

}

}

Update(x);

}

**int** Head(**int** x)

{

Splay(x);

**for**(PushDown(x); tree[x].ch[0]; x=tree[x].ch[0]) PushDown(x);

Splay(x);

**return** x;

}

**int** Expose(**int** x)

{

**int** y;

**for**(y=0; x; x=tree[x].pre) Splay(x),PushDown(x),tree[x].ch[1]=y,Update(y=x);

**return** y;

}

**void** ChangeRoot(**int** x)

{

Rev(Expose(x));

}

**void** Change(**int** x,**int** y,**int** val)

{

ChangeRoot(y);

Expose(x);

Splay(x);

tree[x].val+=val;

tree[x].lazy+=val;

tree[x].mn+=val;

// PushDown(x);

// Update(x);

}

**int** AskMax(**int** x,**int** y)

{

ChangeRoot(x);

Expose(y);

Splay(y);

**return** tree[y].mn;

}

**void** Link(**int** x,**int** y)//link操作即为链接两个树，那么要先进性expose操作，把到路径上的边都变为实边，这样才能进行把x调整到根部，进而通过更改祖先来进行链接

{

ChangeRoot(x);

Splay(x);

tree[x].pre=y;

}

**void** Cut(**int** x,**int** y)

{

ChangeRoot(y);

Splay(x);

**if**(tree[x].ch[0])

{

tree[tree[x].ch[0]].pre=tree[x].pre;

tree[x].pre=tree[x].ch[0]=0;

}

**else** tree[x].pre=0;

}

**int** LCA(**int** x,**int** y)

{

**int** p=Head(Expose(x));

**int** q=Expose(y),w=Head(q);

**if**(p==w) **return** q;

**return** 0;

}

struct data

{

**int** x,y;

} a[maxn];

**int** main()

{

**int** n,m;

**while**(scanf("%d",&n)==1)

{

**for**(**int** i=1; i<n; i++) scanf("%d%d",&a[i].x,&a[i].y);

val[0]=val[n+1]=-INF;

**for**(**int** i=1; i<=n; i++) scanf("%d",&val[i]);

DynamicTree(n+1);

**for**(**int** i=1; i<n; i++) Link(a[i].x,a[i].y);

scanf("%d",&m);

**for**(**int** i=1; i<=m; i++)

{

**int** c;

**int** x,y,val;

scanf("%d",&c);

**if**(c==1)

{

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

**if**(!LCA(x,y)) Link(x,y);

**else** printf("-1\n");

}

**else** **if**(c==2)

{

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

**if**(LCA(x,y) && x!=y) Cut(y,x);

**else** printf("-1\n");

}

**else** **if**(c==3)

{

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

**if**(LCA(x,y)) Change(x,y,val);

**else** printf("-1\n");

}

**else** **if**(c==4)

{

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

**int** tmp=LCA(x,y);

**if**(tmp) printf("%d\n",AskMax(x,y));

**else** printf("-1\n");

}

}

printf("\n");

}

**return** 0;

}

### LCT\_Kuangbin

//BZOJ\_3669\_维护最小生成树

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 200010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

struct edge{

**int** u, v, a, b;

edge(){}

edge(**int** \_u, **int** \_v, **int** \_a, **int** \_b){

u = \_u, v = \_v, a = \_a, b = \_b;

}

bool operator<(**const** edge& o) **const**{

**return** a < o.a;

}

}e[N];

**int** n, m;

**int** pa[N];

**int** find(**int** x)

{

**return** pa[x] == x ? x : pa[x] = find(pa[x]);

}

**void** unite(**int** u, **int** v)

{

u = find(u), v = find(v);

**if**(u == v) **return** ;

pa[u] = v;

}

struct LCT{

**int** ch[N][2],pre[N], rev[N];//rev这个数组是不能去掉的

**int** ma[N], maid[N];

bool rt[N];

**void** Treaval(**int** x) {

**if**(x) {

Treaval(ch[x][0]);

printf("结点%2d:左儿子 %2d 右儿子 %2d 父结点 %2d size = %2d ,key = %2d \n",x, ch[x][0], ch[x][1], pre[x], sz[x], sum[x]);

Treaval(ch[x][1]);

}

}

**void** Update\_Add(**int** r,**int** d)

{

// if(!r)return;

// key[r] += d;

// add[r] += d;

// Max[r] += d;

}

**void** Update\_Rev(**int** r)

{

**if**(!r)**return**;

swap(ch[r][0],ch[r][1]);

rev[r] ^= 1;

}

**void** push\_down(**int** r)

{

// if(add[r])

// {

// Update\_Add(ch[r][0],add[r]);

// Update\_Add(ch[r][1],add[r]);

// add[r] = 0;

// }

**if**(rev[r])

{

Update\_Rev(ch[r][0]);

Update\_Rev(ch[r][1]);

rev[r] = 0;

}

}

**void** push\_up(**int** r)

{

maid[r] = r;

**if**(ma[maid[r]] < ma[maid[ch[r][0]]]) maid[r] = maid[ch[r][0]];

**if**(ma[maid[r]] < ma[maid[ch[r][1]]]) maid[r] = maid[ch[r][1]];

// Max[r] = max(max(Max[ch[r][0]],Max[ch[r][1]]),key[r]);

}

**void** init(**int** n, **int** m)

{

**for**(**int** i = 1; i <= n; i++)

rev[i] = pre[i] = ch[i][0] = ch[i][1] = 0, rt[i] = **true**, ma[i] = 0, maid[i] = i;

**for**(**int** i = 1; i <= m; i++)

{

**int** id = i + n;

rev[id] = pre[id] = ch[id][0] = ch[id][1] = 0;

rt[id] = **true**;

ma[id] = e[i].b;

maid[id] = id;

}

}

**void** Rotate(**int** x)

{

**int** y = pre[x], kind = ch[y][1]==x;

ch[y][kind] = ch[x][!kind];

pre[ch[y][kind]] = y;

pre[x] = pre[y];

pre[y] = x;

ch[x][!kind] = y;

**if**(rt[y])

rt[y] = **false**, rt[x] = **true**;

**else**

ch[pre[x]][ch[pre[x]][1]==y] = x;

push\_up(y);

}

//P函数先将根结点到r的路径上所有的结点的标记逐级下放

**void** P(**int** r)

{

**if**(!rt[r])P(pre[r]);

push\_down(r);

}

**void** Splay(**int** r)

{

P(r);

**while**( !rt[r] )

{

**int** f = pre[r], ff = pre[f];

**if**(rt[f])

Rotate(r);

**else** **if**( (ch[ff][1]==f)==(ch[f][1]==r) )

Rotate(f), Rotate(r);

**else**

Rotate(r), Rotate(r);

}

push\_up(r);

}

**int** Access(**int** x)

{

**int** y = 0;

**for**( ; x ; x = pre[y=x])

{

Splay(x);

rt[ch[x][1]] = **true**, rt[ch[x][1]=y] = **false**;

push\_up(x);

}

**return** y;

}

//判断是否是同根(真实的树，非splay)

bool judge(**int** u,**int** v)

{

**while**(pre[u]) u = pre[u];

**while**(pre[v]) v = pre[v];

**return** u == v;

}

//先Access(r),形成一条路径，再使r成为它所在的树的根

**void** mroot(**int** r)

{

Access(r);

Splay(r);

Update\_Rev(r);

}

//调用后u是原来u和v的lca,v和ch[u][1]分别存着lca的2个儿子

//(原来u和v所在的2颗子树)

**void** lca(**int** &u,**int** &v)

{

Access(v), v = 0;

**while**(u)

{

Splay(u);

**if**(!pre[u])**return**;

rt[ch[u][1]] = **true**;

rt[ch[u][1]=v] = **false**;

push\_up(u);

u = pre[v = u];

}

}

**int** kth(**int** k, **int** root)

{

**int** x = root;

**while**(x){

push\_down(x);

push\_up(x);

**if**(sz[ch[x][0]] >= k) x = ch[x][0];

**else** {

k -= sz[ch[x][0]] + 1;//

**if**(k == 0) **return** x;

x = ch[x][1];

}

}

**return** x;

}

**void** link(**int** u,**int** v)

{

**if**(judge(u,v))

{

// puts("-1");

**return**;

}

mroot(u);//这里的换根操作需要特别注意，有的题目不能换根

pre[u] = v;

}

//先将u变为将v与他的父亲连边切断

**void** cut(**int** x, **int** v)

{

mroot(x);//这里的换根操作需要特别注意，有的题目不能换根

Splay(v);

pre[ch[v][0]] = pre[v];

pre[v] = 0;

rt[ch[v][0]] = **true**;

ch[v][0] = 0;

push\_up(v);

}

//------------End---------------

**int** queryid(**int** x, **int** y)

{

mroot(x);

Access(y);

Splay(y);

**return** maid[y];

}

**int** query(**int** x, **int** y)

{

**return** ma[queryid(x, y)];

}

}lct;

**int** main()

{

// Open();

**while**(~scanf("%d%d", &n, &m))

{

**for**(**int** i = 0 ; i <= n; i++) pa[i] = i;

**for**(**int** i = 1; i <= m; i++)

{

**int** u, v, a, b;scanf("%d%d%d%d", &u, &v, &a, &b);

e[i] = edge(u, v, a, b);

}

sort(e+1, e+1+m);

lct.init(n, m);

**int** ans = INF;

**for**(**int** i = 1; i <= m; i++)

{

**int** u = e[i].u, v = e[i].v, w = e[i].b;

**if**(find(u) != find(v)){

unite(u, v);

lct.link(u, n+i);

lct.link(v, n+i);

}**else** {

**int** id = lct.queryid(u, v);

**if**(w < lct.ma[id]){

lct.cut(e[id - n].u, id);

lct.cut(e[id - n].v, id);

lct.link(u, n+i);

lct.link(v, n+i);

}

}

**if**(find(1) == find(n)){

ans = min(ans, e[i].a + lct.query(1, n));

}

}

**if**(ans == INF) ans = -1;

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

}

**return** 0;

}

### RMQ(区间最值，区间GCD)

#include<iostream>

#include<cmath>

#include<algorithm>

using namespace std;

#define M 100010

#define MAXN 500

#define MAXM 500

**int** dp[M][18];

/\*

\*一维RMQ ST算法 左闭右闭区间

\*构造RMQ数组 makermq(int n,int b[]) O(nlog(n))的算法复杂度

\*dp[i][j] 表示从i到i+2^j -1中最小的一个值(从i开始持续2^j个数)

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

\*查询RMQ rmq(int s,int v)

\*将s-v 分成两个2^k的区间

\*即 k=(int)log2(s-v+1)

\*查询结果应该为 min(dp[s][k],dp[v-2^k+1][k])

\*/

**void** makermq(**int** n,**int** b[],**int** dp[][18])

{

**int** i,j;

**for**(i=0;i<n;i++)

dp[i][0]=b[i];

**for**(j=1;(1<<j)<=n;j++)

**for**(i=0;i+(1<<j)-1<n;i++)

dp[i][j]=min(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);

}

**int** rmq(**int** s,**int** v,**int** dp[][18])

{

**int** k=(**int**)(log((v-s+1)\*1.0)/log(2.0));

**return** min(dp[s][k],dp[v-(1<<k)+1][k]);

}

**void** makeRmqIndex(**int** n,**int** b[],**int** dp[][18]) //返回最小值对应的下标

{

**int** i,j;

**for**(i=0;i<n;i++)

dp[i][0]=i;

**for**(j=1;(1<<j)<=n;j++)

**for**(i=0;i+(1<<j)-1<n;i++)

dp[i][j]=b[dp[i][j-1]] < b[dp[i+(1<<(j-1))][j-1]]? dp[i][j-1]:dp[i+(1<<(j-1))][j-1];

}

**int** rmqIndex(**int** s,**int** v,**int** b[],**int** dp[][18])

{

**int** k=(**int**)(log((v-s+1)\*1.0)/log(2.0));

**return** b[dp[s][k]]<b[dp[v-(1<<k)+1][k]]? dp[s][k]:dp[v-(1<<k)+1][k];

}

**int** main()

{

**int** a[]={3,4,5,7,8,9,0,3,4,5};

//返回下标

makeRmqIndex(sizeof(a)/sizeof(a[0]),a);

cout<<rmqIndex(0,9,a)<<endl;

cout<<rmqIndex(4,9,a)<<endl;

//返回最小值

makermq(sizeof(a)/sizeof(a[0]),a);

cout<<rmq(0,9)<<endl;

cout<<rmq(4,9)<<endl;

**return** 0;

}

**void** initRMQ(**int** n)

{

mm[0] = -1;

**for**(**int** i = 1;i <= n;i++)

{

mm[i] = ((i&(i-1)) == 0)?mm[i-1]+1:mm[i-1];

dp1[i][0] = a[i];

dp2[i][0] = a[i];

}

**for**(**int** j = 1;j <= mm[n];j++)

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

{

dp1[i][j] = max(dp1[i][j-1],dp1[i + (1<<(j-1))][j-1]);

dp2[i][j] = min(dp2[i][j-1],dp2[i + (1<<(j-1))][j-1]);

}

}

**int** rmq(**int** x,**int** y)

{

**int** k = mm[y-x+1];

**return** max(dp1[x][k],dp1[y-(1<<k)+1][k]) - min(dp2[x][k],dp2[y-(1<<k)+1][k]);

}

### 可持久化字典树\_HDU\_4757

/\*

\* 题意：给出一棵树，给出询问u, v, z; 返回uv路径中与z异或的最大值

\*

\* 解法：利用可持久化字典树，维护每个节点到根的信息，每个节点的信息从父亲节点继承，记录这个节点的cnt

\* 那么对于一个u，v来说，如果u到LCA(u,v)+v到LCA(u,v)间的cnt大于1的话，表明当前节点可走。具体可看代码。

\*

\*

\*/

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 100010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("F:/in.txt","r",stdin);

//freopen("F:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

struct node{

**int** go[2];

**int** cnt;

}pool[N\*20];

**int** tot;//

vector<**int**> G[N];//

**int** pa[20][N];//

**int** w[N];//

**int** dep[N];//

**int** n,m;

**int** root[N];//

**int** insert(**int** pre, **int** val)

{

**int** p = ++tot, ret = p;

pool[p] = pool[pre];

**for**(**int** i = 15; i >= 0; i--)

{

**int** tmp = (val>>i)&1;

**int** cur = ++tot;

pool[cur] = pool[pool[p].go[tmp]];

pool[cur].cnt++;

pool[p].go[tmp] = cur;

p = cur;

}

**return** ret;

}

**void** dfs(**int** v, **int** p, **int** d)

{

root[v] = insert(root[max(0, p)], w[v]);

pa[0][v] = p;

dep[v] = d;

**for**(**int** i = 0; i < G[v].size(); i ++)

**if**(G[v][i] != p) dfs(G[v][i], v, d+1);

}

**void** init()

{

tot = 0;

root[0] = 0;

memset(pool, 0, sizeof(pool));

dfs(1, -1, 0);

**for**(**int** k = 0; k + 1 < 20; k++)

**for**(**int** v = 1; v <= n; v++)

**if**(pa[k][v] < 0) pa[k+1][v] = -1;

**else** pa[k+1][v] = pa[k][pa[k][v]];

}

**int** lca(**int** u, **int** v)

{

**if**(dep[u] > dep[v]) swap(u, v);

**for**(**int** k = 0; k < 20; k++)

**if**((dep[v] - dep[u]) >> k & 1)

v = pa[k][v];

**if**(u == v) **return** u;

**for**(**int** k = 19; k >= 0; k--)

**if**(pa[k][u] != pa[k][v])

u = pa[k][u], v = pa[k][v];

**return** pa[0][u];

}

**int** getans(**int** u, **int** v, **int** val)

{

**int** LCA = lca(u, v);

**int** pu = root[u], pv = root[v], pl = root[LCA];

**int** ans = 0;

**for**(**int** i = 15; i >= 0; i--)

{

**int** tmp = (val >> i)&1;

**int** sum = pool[pool[pu].go[!tmp]].cnt + pool[pool[pv].go[!tmp]].cnt - 2 \* pool[pool[pl].go[!tmp]].cnt;

**if**(sum > 0){

pu = pool[pu].go[!tmp];

pv = pool[pv].go[!tmp];

pl = pool[pl].go[!tmp];

ans += 1<<i;

}**else**{

pu = pool[pu].go[tmp];

pv = pool[pv].go[tmp];

pl = pool[pl].go[tmp];

}

}

**return** max(val ^ w[LCA], ans);

}

**int** main()

{

// Open();

**while**(~scanf("%d%d", &n, &m))

{

**for**(**int** i = 1; i <= n; i++)

scanf("%d", &w[i]), G[i].clear();

**for**(**int** i = 1; i < n; i++)

{

**int** u, v;scanf("%d%d", &u, &v);

G[u].push\_back(v);

G[v].push\_back(u);

}

init();

**while**(m--)

{

**int** u, v, z;

scanf("%d%d%d", &u, &v, &z);

printf("%d\n", getans(u, v, z));

}

}

**return** 0;

}

### 树状数组

//树状数组

#define N 100001

**int** n;

**int** c[N];// 每个C数组代表v[i-lowbit(i)+1]到v[i]之间的和

**void** add(**int** i,**int** x)

{

**while**(i<=n)

{

c[i]+=x;

c[i]%=mod;

i+= i & -i;

}

}

**int** getsum(**int** i)

{

**int** cnt=0;

**while**(i>0)

{

cnt+=c[i];

cnt%=mod;

i-=i & -i;

}

**return** cnt%mod;

}

**int** getK(**int** K) //已知前n项和为K，返回n值！

{

**int** ans = 0,cnt=0;

**for**(**int** i=18;i>=0;i--)//i>=0

{

ans+=(1<<i);

**if**(ans>=N||cnt+c[ans]>=K)ans-=(1<<i);

**else** cnt+=c[ans];

}

**return** ans+1;

}

//树状数组

//求前缀最大, 更新值只能增大。

**int** c[N];

**void** update(**int** x, **int** val)

{

**if**(x == 0) **return** ;

**for**(**int** i=x;i<=N;i+=i&-i) c[i] = max(c[i], val);

}

**int** getmax(**int** x)

{

**int** rnt = -INF;

**for**(**int** i=x;i>0;i-=i&-i) rnt = max(rnt, c[i]);

**return** rnt;

}

//求前缀最大

//二维树状数组

**int** Lowbit(**int** x)

{

**return** x & (-x);

}

**void** Updata(**int** x,**int** y,**int** a)

{

**int** i,k;

**for**(i=x; i<=n; i+=Lowbit(i))

**for**(k=y; k<=n; k+=Lowbit(k))

c[i][k]+=a;

}

**int** Getsum(**int** x,**int** y)

{

**int** i,k,sum = 0;

**for**(i=x; i>0; i-=Lowbit(i))

**for**(k=y; k>0; k-=Lowbit(k))

sum += c[i][k];

**return** sum;

}

//二维树状数组

#include <cstdlib>

#include <cstring>

#include <cstdio>

#include <algorithm>

#include <iostream>

#include <map>

using namespace std;

#define Lowbit(x) ((x)&(-x))

/\*

index from 1,like 1,2,3,4,5,,,,

array c is the 2-dimensional BIT array

\*/

**int** c[4][4];

**int** m[4][4];

**int** n=3;

**void** add(**int** x, **int** y,**int** delta){

**int** i=y;

**while**(x<=n){

y=i;

**while**(y<=n){

c[x][y]+=delta;

y+=Lowbit(y);

}

x+=Lowbit(x);

}

}

**int** Sum(**int** x, **int** y){

**int** i=y, sum=0;

**while**(x>0){

y=i;

**while**(y>0){

sum+=c[x][y];

y-=Lowbit(y);

}

x-=Lowbit(x);

}

**return** sum;

}

**int** main() {

//freopen("G:/in.txt","r",stdin);

/\* the input is

1 2 2

2 1 3

3 4 5

\*/

**for**(**int** i=1;i<=3;i++)

**for**(**int** j=1;j<=3;j++){

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

add(i,j,m[i][j]);

}

**for**(**int** i=1;i<=3;i++){

**for**(**int** j=1;j<=3;j++){

cout<<Sum(i,j)<<' ';

}

cout<<endl;

}

/\* the output should be

1 3 5

3 6 11

6 13 23

\*/

**return** 0;

}

//区间修改，点查询

**long** **long** c[N];

**long** **long** sum[N];

**long** **long** n;

**long** **long** getnum(**long** **long** x)

{

**long** **long** rnt=0;

**for**(**long** **long** i=x;i<=n;i+=(i&(-i)))

{

rnt+=c[i];

}

**return** rnt;

}

**void** add(**long** **long** i,**long** **long** a)

{

**while**(i>=1)

{

c[i]+=a;

i-=(i&(-i));

}

}

//区间修改，区间查询, cnt 相当于cas清空用

LL query(LL a[][2], **int** x)

{

LL res = 0;

**for**(; x > 0; x -= (x&(-x)))

{

**if**(a[x][0] == cnt) res += a[x][1];

}

**return** res;

}

LL query(**int** l, **int** r)

{

**return** query(x1, l)\*(r-l+1)+ (r+1)\*(query(x1, r)-query(x1, l)) - (query(x2, r)-query(x2, l));

}

**void** add(LL a[][2], **int** x, LL c)

{

**for**(; x <= n; x += ((-x)&x))

{

**if**(a[x][0] == cnt) a[x][1] += c;

**else** a[x][0] = cnt, a[x][1] = c;

}

}

**void** add(**int** l, **int** r, **int** c)

{

add(x1, l, c);

add(x2, l, (LL)l\*c);

add(x1, r+1, -c);

add(x2, r+1, (LL)(r+1)\*(-c));

}

//区间修改，区间查询

//推导: http://www.cnblogs.com/lazycal/archive/2013/08/05/3239304.html

### 线段树终极版

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 201000

#define lson x<<1

#define rson x<<1|1

#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

typedef pair<PII, **int**> PIII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

**int** a[N];

struct node

{

**int** l, r;

**int** lazy;

**int** mi;

node(){}

node(**int** ll, **int** rr, **int** lazyy, **int** mii){

l = ll, r = rr, lazy = lazyy, mi = mii;

}

}lt[N\*10];

**void** push\_up(**int** x)

{

lt[x].mi = min(lt[lson].mi, lt[rson].mi);

}

**void** push\_down(**int** x)

{

**if**(lt[x].lazy){

lt[lson].mi += lt[x].lazy;

lt[lson].lazy += lt[x].lazy;

lt[rson].mi += lt[x].lazy;

lt[rson].lazy += lt[x].lazy;

lt[x].lazy = 0;

}

}

**void** build(**int** l, **int** r, **int** x)

{

lt[x] = node(l, r, 0, 0);

**if**(l == r){

lt[x].mi = a[l];

**return** ;

}

build(l, mid, lson);

build(mid+1, r, rson);

push\_up(x);

}

**void** update(**int** l, **int** r, **int** val, **int** x){

**if**(l > r) **return** ;

**if**(lt[x].l >= l && lt[x].r <= r){

lt[x].mi += val;

lt[x].lazy += val;

**return** ;

}

push\_down(x);

**if**(r <= mid) update(l, r, val, lson);

**else** **if**(l > mid) update(l, r, val, rson);

**else** update(l, mid, val, lson), update(mid+1, r, val, rson);

push\_up(x);

}

**int** query(**int** l, **int** r, **int** x)

{

**if**(lt[x].l == lt[x].r){

**if**(lt[x].mi == 0) **return** lt[x].l;

**else** **return** -1;

}

**if**(lt[x].l >= l && lt[x].r <= r && lt[x].mi > 0){

**return** -1;

}

push\_down(x);

**int** res;

**if**(r <= mid) res = query(l, r, lson);

**else** **if**(l > mid) res = query(l, r, rson);

**else** {

**int** lres = query(l, mid, lson);

**int** rres = query(mid+1, r, rson);

**if**(lres != -1) res = lres;

**else** **if**(rres != -1) res = rres;

**else** res = -1;

}

push\_up(x);

**return** res;

}

**int** ans[N];

**int** main()

{

Open();

**int** n;

**while**(scanf("%d",&n)==1)

{

**if**(!n)**break**;

**for**(**int** i = 1; i <= n; i++) scanf("%d", &a[i]);

build(1, n, 1);

**int** cnt = 2 \* n;

queue<**int**> q;

bool flag = **true**;

**int** tail = 0;

**while**(cnt-- && flag)

{

**int** idx = query(1, n, 1);

**if**(idx == -1){

**if**(q.empty()) {

flag = **false**;

**continue**;

}**else**{

**int** res = q.front();q.pop();

ans[tail++] = -res;

// printf("%d", -res);

update(res+1, n, -1, 1);

}

}**else**{

ans[tail++] = idx;

q.push(idx);

update(1, idx - 1, -1, 1);

update(idx, idx, INF, 1);

}

}

**if**(!flag){

printf("Impossible\n");

}**else**{

**for**(**int** i = tail - 1; i >= 0; i--)

{

printf("%d", ans[i]);

**if**(i) printf(" ");

**else** printf("\n");

}

}

}

**return** 0;

}

### 主席树-区间K大-POJ-2104

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Author :kuangbin

Created Time :2013-9-4 20:13:20

File Name :POJ2104.cpp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

#include <stdio.h>

#include <string.h>

#include <iostream>

#include <algorithm>

#include <vector>

#include <queue>

#include <set>

#include <map>

#include <string>

#include <math.h>

#include <stdlib.h>

#include <time.h>

using namespace std;

**const** **int** MAXN = 100010;

**const** **int** M = MAXN \* 30;

**int** n,q,m,tot;

**int** a[MAXN], t[MAXN];

**int** T[M], lson[M], rson[M], c[M];

**void** Init\_hash()

{

**for**(**int** i = 1; i <= n;i++)

t[i] = a[i];

sort(t+1,t+1+n);

m = unique(t+1,t+1+n)-t-1;

}

**int** build(**int** l,**int** r)

{

**int** root = tot++;

c[root] = 0;

**if**(l != r)

{

**int** mid = (l+r)>>1;

lson[root] = build(l,mid);

rson[root] = build(mid+1,r);

}

**return** root;

}

**int** hash(**int** x)

{

**return** lower\_bound(t+1,t+1+m,x) - t;

}

**int** update(**int** root,**int** pos,**int** val)

{

**int** newroot = tot++, tmp = newroot;

c[newroot] = c[root] + val;

**int** l = 1, r = m;

**while**(l < r)

{

**int** mid = (l+r)>>1;

**if**(pos <= mid)

{

lson[newroot] = tot++; rson[newroot] = rson[root];

newroot = lson[newroot]; root = lson[root];

r = mid;

}

**else**

{

rson[newroot] = tot++; lson[newroot] = lson[root];

newroot = rson[newroot]; root = rson[root];

l = mid+1;

}

c[newroot] = c[root] + val;

}

**return** tmp;

}

**int** query(**int** left\_root,**int** right\_root,**int** k)

{

**int** l = 1, r = m;

**while**( l < r)

{

**int** mid = (l+r)>>1;

**if**(c[lson[left\_root]]-c[lson[right\_root]] >= k )

{

r = mid;

left\_root = lson[left\_root];

right\_root = lson[right\_root];

}

**else**

{

l = mid + 1;

k -= c[lson[left\_root]] - c[lson[right\_root]];

left\_root = rson[left\_root];

right\_root = rson[right\_root];

}

}

**return** l;

}

**int** main()

{

//freopen("in.txt","r",stdin);

//freopen("out.txt","w",stdout);

**while**(scanf("%d%d",&n,&q) == 2)

{

tot = 0;

**for**(**int** i = 1;i <= n;i++)

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

Init\_hash();

T[n+1] = build(1,m);

**for**(**int** i = n;i ;i--)

{

**int** pos = hash(a[i]);

T[i] = update(T[i+1],pos,1);

}

**while**(q--)

{

**int** l,r,k;

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

printf("%d\n",t[query(T[l],T[r+1],k)]);

}

}

**return** 0;

}

### 主席树-区间修改hihocoder couple tree

/\*

\* 题意：给出两棵树，有若干个询问，每次询问u,v,需要返回这两个节点分别在各自的

\* 树上的最近公共祖先，保证父亲节点的编号一定小于儿子。强制在线。

\*

\* 做法： 首先考虑离线的做法，对于一个第一棵树上特定的u点，我们在线段树中保存根

\* 节点到u节点的信息，对于u节点到根上的节点x对于子树上的所有点来说都是父亲。如果

\* 将子树上面的所有点对应的第二棵树上的点都标记上父亲的标号，并且取最大标号留下

\* 的话，那么做完之后第二棵树中每个节点的标号就是这个节点与u节点的最近公共祖先(不

\* 同树上的)。那么也就是说这里对于第一棵树中每一个节点来说都得有一棵对应的线段

\* 树，那么很容易就想到了主席树的做法。每个节点的信息从父亲处继承下来。那么对于

\* 一个询问就在相应的线段树上面查询即可。

\*/

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 100010

#define M 2500010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

**int** lson[M], rson[M], ma[M], tot, lazy[M], mi[M];//维护的信息

**int** Tn;

**int** dep[2][N];

**int** fa[2][N];

**int** st[N], ed[N];

vector<**int**> G[2][N];

**int** n, m;

**int** T[N];//存储每个节点的线段树的根节点编号。

**void** dfsseq(**int** u, vector<**int**> G[], **int** d)

{

dep[1][u] = d;

st[u] = ++Tn;

**for**(**int** i = 0; i < G[u].size(); i ++)

dfsseq(G[u][i], G, d + 1);

ed[u] = Tn;

}

**void** dfsdep(**int** u, **int** d)

{

dep[0][u] = d;

**for**(**int** i = 0; i < G[0][u].size(); i ++)

dfsdep(G[0][u][i], d+1);

}

**int** build(**int** l, **int** r)

{

**int** root = tot++;

ma[root] = 0, mi[root] = 0, lazy[root] = 0;

**if**(l == r) **return** root;

**int** mid = l + r >> 1;

lson[root] = build(l, mid);

rson[root] = build(mid+1, r);

**return** root;

}

// 有的模板下放懒标记的时候，用一个数组标记了当前点的左右儿子是否是历史版本， 如果是的话再重新申请节点，如果不是就直接更新。

// 这么做的意义是为了节省空间，我个人测试之后发现这样做不得不再开一个与线段树大小相同的标记数组，其实大多数情况下内存是会更

// 大的...我一开始的做法就是，不管是不是历史版本， 只要懒标记有效，我就申请两个新节点当做儿子。

// 但是也会有恶心的数据会让这种不加标记的MLE的，所以注意取舍。

**void** push\_down(**int** root){

**if**(lazy[root]){

lson[tot] = lson[lson[root]];

rson[tot] = rson[lson[root]];

ma[tot] = max(ma[lson[root]], lazy[root]);

mi[tot] = max(mi[lson[root]], lazy[root]);

lazy[tot] = max(lazy[lson[root]], lazy[root]);

lson[root] = tot++;

lson[tot] = lson[rson[root]];

rson[tot] = rson[rson[root]];

ma[tot] = max(ma[rson[root]], lazy[root]);

mi[tot] = max(mi[rson[root]], lazy[root]);

lazy[tot] = max(lazy[rson[root]], lazy[root]);

rson[root] = tot++;

lazy[root] = 0;

}

}

**void** push\_up(**int** root){

ma[root] = max(ma[lson[root]], ma[lson[root]]);

mi[root] = min(mi[lson[root]], mi[lson[root]]);

}

**int** update(**int** root, **int** L, **int** R, **int** l, **int** r, **int** val)// [l, r]修改区间，[L, R]当前区间

{

**if**(mi[root] >= val) **return** root;

**int** newroot = tot ++;

**if**(L >= l && R <= r){

lazy[newroot] = max(lazy[root], val);

ma[newroot] = max(ma[root], val);

mi[newroot] = max(mi[root], val);

lson[newroot] = lson[root];

rson[newroot] = rson[root];

**return** newroot;

}

lazy[newroot] = ma[newroot] = 0, mi[newroot] = 0;

push\_down(root);

**int** mid = L + R >> 1;

**if**(r <= mid)

{

lson[newroot] = update(lson[root], L, mid, l, r, val);

rson[newroot] = rson[root];

}**else** **if**(l > mid){

lson[newroot] = lson[root];

rson[newroot] = update(rson[root], mid+1, R, l, r, val);

}**else**{

lson[newroot] = update(lson[root], L, mid, l, mid, val);

rson[newroot] = update(rson[root], mid+1, R, mid+1, r, val);

}

push\_up(newroot);

**return** newroot;

}

**int** query(**int** root, **int** L, **int** R, **int** idx){

**if**(L == R && L == idx) **return** ma[root];

push\_down(root);

**int** mid = L + R >> 1;

**if**(idx <= mid) **return** query(lson[root], L, mid, idx);

**else** **return** query(rson[root], mid+1, R, idx);

}

**void** dfs(**int** u, vector<**int**> G[])

{

T[u] = update(T[fa[0][u]], 1, Tn, st[u], ed[u], u);

**for**(**int** i = 0; i < G[u].size(); i ++)

{

dfs(G[u][i], G);

}

}

**int** main()

{

Open();

**while**(~scanf("%d%d", &n, &m)){

Tn = 0;

tot = 0;

**for**(**int** i = 0; i <= n; i++) G[1][i].clear(), G[0][i].clear();

**for**(**int** i = 2; i <= n; i++)

{

scanf("%d", &fa[0][i]);

G[0][fa[0][i]].push\_back(i);

}

**for**(**int** i = 2; i <= n; i++)

{

scanf("%d", &fa[1][i]);

G[1][fa[1][i]].push\_back(i);

}

dfsseq(1, G[1], 1);

dfsdep(1, 1);

fa[0][1] = n + 1;

T[n+1] = build(1, Tn);

dfs(1, G[0]);

**int** preans = 0;

**while**(m--)

{

**int** u, v;

scanf("%d%d", &u, &v);

u += preans; u %= n; u++;

v += preans; v %= n; v++;

preans = query(T[u], 1, Tn, st[v]);

printf("%d %d %d\n", preans, dep[0][u] - dep[0][preans] + 1, dep[1][v] - dep[1][preans] + 1);

}

}

**return** 0;

}

## 数论

### 朴素筛素数

//

**int** prime[N];

**int** pn;

**void** get\_prime()

{

**for**(**int** i=2;i<N;i++)

{

**if**(vis[i]) **continue**;

prime[pn++]=i;

**for**(**int** j=i;j<N;j+=i)

{

vis[j]=1;

}

}

}

//朴素筛素数

//--------------------O(n)筛法------------------------//

bool pvis[N];

**int** pri[N];

**int** prin;

**void** getpri(){

prin = 0;

memset(pvis, 0, sizeof(pvis));

**for**(**int** i = 2; i < N; i++) {

**if**(pvis[i] == 0) pri[prin++] = i;

**for**(**int** j = 0; j < prin && (**long** **long**)pri[j]\*i < N; j++) {

pvis[pri[j]\*i] = 1;

**if**(i % pri[j] == 0) **break**;

}

}

}

### 欧拉函数：求小于n的有多少个数与n互质

//欧拉函数：求小于n的有多少个数与n互质O(log(n))，

//这里根据代码逻辑，似乎可以先打出质数表，利用质数表的话效率会更高才对

**int** phi(**int** n)

{

**int** res=n,a=n;

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

**if**(a%i==0){

res=res/i\*(i-1);//先进行除法是为了防止中间数据的溢出

**while**(a%i==0) a/=i;

}

}

**if**(a>1) res=res/a\*(a-1);

**return** res;

}

//欧拉函数：求小于n的有多少个数与n互质

//筛法欧拉函数，复杂度与素数表一样

**int** euler[N];

**void** getEuler(**int** UP)

{

memset(euler, 0, sizeof(euler));

euler[1] = 1;

**for**(**int** i = 2; i <= UP; i++)

{

**if**(!euler[i])

{

**for**(**int** j = ; j <= UP; j += i)

{

**if**(!euler[j]) euler[j] = j;

euler[j] = euler[j] / i \* (i-1);

}

}

}

}

//筛法欧拉函数

### 快速幂运算（模）

//快速幂运算（模）

**int** mod\_pow(**int** x,**int** n)

{

**int** res=1;

**while**(n>0)

{

**if**(n&1) res=(**long** **long**)res\*x%mod;

x=x\*x%mod;

n>>=1;

}

**return** res;

}

//快速幂运算（模）

//如果乘法可能爆long long的话， 使用下面的板子

ll mul(ll a,ll b){

**return** ((a\*b-ll(((**long** **double**)a)/mod\*b+1e-3)\*mod)%mod+mod)%mod;

}

ll pow\_mod(ll a, ll n) {

a%=mod;

ll r = 1;

**while**(n) {

**if**(n&1) r = mul(r,a);

a=mul(a,a);

n>>=1;

}

**return** r;

}

### 康拓展开hash阶乘

**int** fac[]={1,1,2,6,24,120,720,5040,40320,362880}; //康拖展开判重

// 0!1!2!3! 4! 5! 6! 7! 8! 9!

**int** Cantor(**int** s[],n){ //康拖展开求该序列的hash值

**int** sum=0;

**for**(**int** i=0;i<n;i++){

**int** cnt=0;

**for**(**int** j=i+1;j<n;j++)

**if**(s[i]>s[j])

cnt++;

sum+=(cnt\*fac[n-i-1]);

}

**return** sum;

}

**void** invCantor(**int** ans[], **int** n, **int** k)

{

**int** vis[20] = {0};

**int** i, j, t;

**for** (i = 0; i < n; ++i)

{

t = k / fac[n - i - 1];

**for** (j = 1; j <= n; j++)

**if** (!vis[j])

{

**if** (t == 0) **break**;

--t;

}

ans[i] = j, vis[j] = **true**;

k %= fac[n - i - 1];///余数

}

}

### 高斯消元(异或)

**int** cal(**int** row) //高斯消元中，有num-row个自由元，枚举自由元之后计算结果，回代过程

{

**for**(**int** i=row-1;i>=0;i--)

{

ans[i] = mat[i][num];

**for**(**int** j=i+1;j<num;j++)

ans[i]^=mat[i][j]\*ans[j];

}

**int** rnt = 0;

//枚举自由元之后的操作

**for**(**int** i=0;i<num;i++)

rnt += ans[i];

**return** rnt;

}

**int** gauss(**int** ty)

{

init(ty);//初始化系数矩阵

**int** row = 0, col = 0;

**while**(row < num && col < num)

{

**int** i;

**for**(i = row; i<num; i++)

**if**(mat[i][col]) **break**;

**if**(i == num){

col++;**continue**;

}

**if**(i != row)// 这里的行交换和普通高斯消元相同，将矩阵变为上三角矩阵

{

**for**(**int** j = col; j <= num;j++)

swap(mat[i][j], mat[row][j]);

}

**if**(row != col)// 这里的列交换是为了将自由元移动到后面，方便处理

{

**for**(**int** j=0;j<num;j++)

swap(mat[j][row], mat[j][col]);

}

**for**(**int** i = row + 1; i < num; i++)

{

**if**(mat[i][row] == 0) **continue**;

**for**(**int** j = row; j <= num; j++)

mat[i][j] ^= mat[row][j];

}

row++, col++;

}

**for**(**int** i = row; i < num; i++) // 检测所有全0行的结果是否为1，如果为1，则无解

**if**(mat[i][num]) **return** -1;

// 枚举自由元

**int** free = num - row;

**int** len = 1<<free;

**int** rnt = INF;

**for**(**int** i = 0; i < len; i++)

{

**for**(**int** j = 0; j < free; j++)

ans[num - j - 1] = ((i >> j) & 1);

rnt = min(rnt , cal(row));

}

**return** rnt;

}

### 分数类

struct Fraction{

LL num, den;

Fraction(LL num = 0, LL den = 1)

{

**if**(den < 0){

num = -num;

den = -den;

}

LL g = \_\_gcd(abs(num), den);

**this** -> num = num / g;

**this** -> den = den / g;

}

Fraction operator + (**const** Fraction& o)**const**{

**return** Fraction(num \* o.den + den \* o.num, den \* o.den);

}

Fraction operator + (**const** LL& o)**const**{

**return** Fraction(num + den \* o, den);

}

Fraction operator - (**const** Fraction& o)**const**{

**return** Fraction(num \* o.den - den \* o.num, den \* o.den);

}

Fraction operator \* (**const** Fraction& o)**const**{

**return** Fraction(num \* o.num, den \* o.den);

}

Fraction operator \* (**const** LL& o)**const**{

**return** Fraction(num \* o, den);

}

Fraction operator / (**const** Fraction& o)**const**{

**return** Fraction(num \* o.den, den \* o.num);

}

Fraction operator / (**const** LL& o)**const**{

**return** Fraction(num, den \* o);

}

bool operator < (**const** Fraction& o)**const**{

**return** num \* o.den < den \* o.num;

}

bool operator == (**const** Fraction& o)**const**{

**return** num \* o.den == den \* o.num;

}

};

### 大素数判断(无random)

LL mul(LL a,LL b, LL c)

{

LL ans=0;

**while**(b){

**if**(b&1)ans=(ans+a)%c;

a=(a\*2)%c;

b>>=1;

}

**return** ans;

}

LL pow(LL x,LL n,LL mod)

{

LL res=1;

**while**(n>0){

**if**(n&1)res=mul(res,x,mod);

x=mul(x,x,mod);

n>>=1;

}

**return** res;

}

bool test(LL nn,LL a,LL d)

{

**if**(nn==1) **return** **false**;

**if**(nn==2) **return** **true**;

**if**(nn==a) **return** **true**;

**if**((nn&1)==0) **return** **false**;

**while**(!(d&1))

d=d>>1;

LL t=pow(a,d,nn);

**while**((d!=nn-1)&&(t!=1)&&(t!=nn-1))

{

t=mul(t,t,nn);

d=d<<1;

}

**return** (t==nn-1||(d&1)==1);

}

bool isPrime(LL N)

{

**if**(N<2)

**return** **false**;

LL a[]={2,3,61};//一些质数

**for**(**int** i=0;i<=2;i++)//利用每个质数进行测试

{

**if**(!test(N,a[i],N-1))

**return** **false**;

}

**return** **true**;

}

### 高精度(字符串)

#include <cstdio>

#include <cstring>

#include <cstdlib>

//允许生成1120位（二进制）的中间结果

#define BI\_MAXLEN 105

#define DEC 10

#define HEX 16

**class** CBigInt

{

**public**:

//大数在0x100000000进制下的长度

unsigned m\_nLength;

//用数组记录大数在0x100000000进制下每一位的值

unsigned **long** m\_ulValue[BI\_MAXLEN];

CBigInt();

~CBigInt();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

基本操作与运算

Mov，赋值运算，可赋值为大数或普通整数，可重载为运算符“=”

Cmp，比较运算，可重载为运算符“==”、“!=”、“>=”、“<=”等

Add，加，求大数与大数或大数与普通整数的和，可重载为运算符“+”

Sub，减，求大数与大数或大数与普通整数的差，可重载为运算符“-”

Mul，乘，求大数与大数或大数与普通整数的积，可重载为运算符“\*”

Div，除，求大数与大数或大数与普通整数的商，可重载为运算符“/”

Mod，模，求大数与大数或大数与普通整数的模，可重载为运算符“%”

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** Mov(unsigned \_\_int64 A);

**void** Mov(CBigInt& A);

CBigInt Add(CBigInt& A);

CBigInt Sub(CBigInt& A);

CBigInt Mul(CBigInt& A);

CBigInt Div(CBigInt& A);

CBigInt Mod(CBigInt& A);

CBigInt Add(unsigned **long** A);

CBigInt Sub(unsigned **long** A);

CBigInt Mul(unsigned **long** A);

CBigInt Div(unsigned **long** A);

unsigned **long** Mod(unsigned **long** A);

**int** Cmp(CBigInt& A);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

输入输出

Get，从字符串按10进制或16进制格式输入到大数

Put，将大数按10进制或16进制格式输出到字符串

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** Get(**char** str[], unsigned **int** system=DEC);

**void** Put(**char** str[], unsigned **int** system=DEC);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RSA相关运算

Rab，拉宾米勒算法进行素数测试

Euc，欧几里德算法求解同余方程

RsaTrans，反复平方算法进行幂模运算

GetPrime，产生指定长度的随机大素数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**int** Rab();

CBigInt Euc(CBigInt& A);

CBigInt RsaTrans(CBigInt& A, CBigInt& B);

**void** GetPrime(**int** bits);

};

//小素数表

**const** **static** **int** PrimeTable[550]=

{ 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,

37, 41, 43, 47, 53, 59, 61, 67, 71, 73,

79, 83, 89, 97, 101, 103, 107, 109, 113, 127,

131, 137, 139, 149, 151, 157, 163, 167, 173, 179,

181, 191, 193, 197, 199, 211, 223, 227, 229, 233,

239, 241, 251, 257, 263, 269, 271, 277, 281, 283,

293, 307, 311, 313, 317, 331, 337, 347, 349, 353,

359, 367, 373, 379, 383, 389, 397, 401, 409, 419,

421, 431, 433, 439, 443, 449, 457, 461, 463, 467,

479, 487, 491, 499, 503, 509, 521, 523, 541, 547,

557, 563, 569, 571, 577, 587, 593, 599, 601, 607,

613, 617, 619, 631, 641, 643, 647, 653, 659, 661,

673, 677, 683, 691, 701, 709, 719, 727, 733, 739,

743, 751, 757, 761, 769, 773, 787, 797, 809, 811,

821, 823, 827, 829, 839, 853, 857, 859, 863, 877,

881, 883, 887, 907, 911, 919, 929, 937, 941, 947,

953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019,

1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087,

1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153,

1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229,

1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297,

1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381,

1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453,

1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523,

1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597,

1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663,

1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741,

1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811, 1823,

1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901,

1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993,

1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063,

2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131,

2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221,

2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293,

2297, 2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371,

2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437,

2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539,

2543, 2549, 2551, 2557, 2579, 2591, 2593, 2609, 2617, 2621,

2633, 2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687, 2689,

2693, 2699, 2707, 2711, 2713, 2719, 2729, 2731, 2741, 2749,

2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819, 2833,

2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909,

2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001,

3011, 3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083,

3089, 3109, 3119, 3121, 3137, 3163, 3167, 3169, 3181, 3187,

3191, 3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259,

3271, 3299, 3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343,

3347, 3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413, 3433,

3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517,

3527, 3529, 3533, 3539, 3541, 3547, 3557, 3559, 3571, 3581,

3583, 3593, 3607, 3613, 3617, 3623, 3631, 3637, 3643, 3659,

3671, 3673, 3677, 3691, 3697, 3701, 3709, 3719, 3727, 3733,

3739, 3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823,

3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911,

3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001

};

//构造大数对象并初始化为零

CBigInt::CBigInt()

{

m\_nLength=1;

**for**(**int** i=0;i<BI\_MAXLEN;i++)m\_ulValue[i]=0;

}

//解构大数对象

CBigInt::~CBigInt()

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数比较

调用方式：N.Cmp(A)

返回值：若N<A返回-1；若N=A返回0；若N>A返回1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**int** CBigInt::Cmp(CBigInt& A)

{

**if**(m\_nLength>A.m\_nLength)**return** 1;

**if**(m\_nLength<A.m\_nLength)**return** -1;

**for**(**int** i=m\_nLength-1;i>=0;i--)

{

**if**(m\_ulValue[i]>A.m\_ulValue[i])**return** 1;

**if**(m\_ulValue[i]<A.m\_ulValue[i])**return** -1;

}

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数赋值

调用方式：N.Mov(A)

返回值：无，N被赋值为A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** CBigInt::Mov(CBigInt& A)

{

m\_nLength=A.m\_nLength;

**for**(**int** i=0;i<BI\_MAXLEN;i++)m\_ulValue[i]=A.m\_ulValue[i];

}

**void** CBigInt::Mov(unsigned \_\_int64 A)

{

**if**(A>0xffffffff)

{

m\_nLength=2;

m\_ulValue[1]=(unsigned **long**)(A>>32);

m\_ulValue[0]=(unsigned **long**)A;

}

**else**

{

m\_nLength=1;

m\_ulValue[0]=(unsigned **long**)A;

}

**for**(**int** i=m\_nLength;i<BI\_MAXLEN;i++)m\_ulValue[i]=0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数相加

调用形式：N.Add(A)

返回值：N+A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::Add(CBigInt& A)

{

CBigInt X;

X.Mov(\***this**);

unsigned carry=0;

unsigned \_\_int64 sum=0;

**if**(X.m\_nLength<A.m\_nLength)X.m\_nLength=A.m\_nLength;

**for**(unsigned i=0;i<X.m\_nLength;i++)

{

sum=A.m\_ulValue[i];

sum=sum+X.m\_ulValue[i]+carry;

X.m\_ulValue[i]=(unsigned **long**)sum;

carry=(unsigned)(sum>>32);

}

X.m\_ulValue[X.m\_nLength]=carry;

X.m\_nLength+=carry;

**return** X;

}

CBigInt CBigInt::Add(unsigned **long** A)

{

CBigInt X;

X.Mov(\***this**);

unsigned \_\_int64 sum;

sum=X.m\_ulValue[0];

sum+=A;

X.m\_ulValue[0]=(unsigned **long**)sum;

**if**(sum>0xffffffff)

{

unsigned i=1;

**while**(X.m\_ulValue[i]==0xffffffff){X.m\_ulValue[i]=0;i++;}

X.m\_ulValue[i]++;

**if**(m\_nLength==i)m\_nLength++;

}

**return** X;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数相减

调用形式：N.Sub(A)

返回值：N-A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::Sub(CBigInt& A)

{

CBigInt X;

X.Mov(\***this**);

**if**(X.Cmp(A)<=0){X.Mov(0);**return** X;}

unsigned carry=0;

unsigned \_\_int64 num;

unsigned i;

**for**(i=0;i<m\_nLength;i++)

{

**if**((m\_ulValue[i]>A.m\_ulValue[i])||((m\_ulValue[i]==A.m\_ulValue[i])&&(carry==0)))

{

X.m\_ulValue[i]=m\_ulValue[i]-carry-A.m\_ulValue[i];

carry=0;

}

**else**

{

num=0x100000000+m\_ulValue[i];

X.m\_ulValue[i]=(unsigned **long**)(num-carry-A.m\_ulValue[i]);

carry=1;

}

}

**while**(X.m\_ulValue[X.m\_nLength-1]==0)X.m\_nLength--;

**return** X;

}

CBigInt CBigInt::Sub(unsigned **long** A)

{

CBigInt X;

X.Mov(\***this**);

**if**(X.m\_ulValue[0]>=A){X.m\_ulValue[0]-=A;**return** X;}

**if**(X.m\_nLength==1){X.Mov(0);**return** X;}

unsigned \_\_int64 num=0x100000000+X.m\_ulValue[0];

X.m\_ulValue[0]=(unsigned **long**)(num-A);

**int** i=1;

**while**(X.m\_ulValue[i]==0){X.m\_ulValue[i]=0xffffffff;i++;}

X.m\_ulValue[i]--;

**if**(X.m\_ulValue[i]==0)X.m\_nLength--;

**return** X;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数相乘

调用形式：N.Mul(A)

返回值：N\*A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::Mul(CBigInt& A)

{

**if**(A.m\_nLength==1)**return** Mul(A.m\_ulValue[0]);

CBigInt X;

unsigned \_\_int64 sum,mul=0,carry=0;

unsigned i,j;

X.m\_nLength=m\_nLength+A.m\_nLength-1;

**for**(i=0;i<X.m\_nLength;i++)

{

sum=carry;

carry=0;

**for**(j=0;j<A.m\_nLength;j++)

{

**if**(((i-j)>=0)&&((i-j)<m\_nLength))

{

mul=m\_ulValue[i-j];

mul\*=A.m\_ulValue[j];

carry+=mul>>32;

mul=mul&0xffffffff;

sum+=mul;

}

}

carry+=sum>>32;

X.m\_ulValue[i]=(unsigned **long**)sum;

}

**if**(carry){X.m\_nLength++;X.m\_ulValue[X.m\_nLength-1]=(unsigned **long**)carry;}

**return** X;

}

CBigInt CBigInt::Mul(unsigned **long** A)

{

CBigInt X;

unsigned \_\_int64 mul;

unsigned **long** carry=0;

X.Mov(\***this**);

**for**(unsigned i=0;i<m\_nLength;i++)

{

mul=m\_ulValue[i];

mul=mul\*A+carry;

X.m\_ulValue[i]=(unsigned **long**)mul;

carry=(unsigned **long**)(mul>>32);

}

**if**(carry){X.m\_nLength++;X.m\_ulValue[X.m\_nLength-1]=carry;}

**return** X;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数相除

调用形式：N.Div(A)

返回值：N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::Div(CBigInt& A)

{

**if**(A.m\_nLength==1)**return** Div(A.m\_ulValue[0]);

CBigInt X,Y,Z;

unsigned i,len;

unsigned \_\_int64 num,div;

Y.Mov(\***this**);

**while**(Y.Cmp(A)>=0)

{

div=Y.m\_ulValue[Y.m\_nLength-1];

num=A.m\_ulValue[A.m\_nLength-1];

len=Y.m\_nLength-A.m\_nLength;

**if**((div==num)&&(len==0)){X.Mov(X.Add(1));**break**;}

**if**((div<=num)&&len){len--;div=(div<<32)+Y.m\_ulValue[Y.m\_nLength-2];}

div=div/(num+1);

Z.Mov(div);

**if**(len)

{

Z.m\_nLength+=len;

**for**(i=Z.m\_nLength-1;i>=len;i--)Z.m\_ulValue[i]=Z.m\_ulValue[i-len];

**for**(i=0;i<len;i++)Z.m\_ulValue[i]=0;

}

X.Mov(X.Add(Z));

Y.Mov(Y.Sub(A.Mul(Z)));

}

**return** X;

}

CBigInt CBigInt::Div(unsigned **long** A)

{

CBigInt X;

X.Mov(\***this**);

**if**(X.m\_nLength==1){X.m\_ulValue[0]=X.m\_ulValue[0]/A;**return** X;}

unsigned \_\_int64 div,mul;

unsigned **long** carry=0;

**for**(**int** i=X.m\_nLength-1;i>=0;i--)

{

div=carry;

div=(div<<32)+X.m\_ulValue[i];

X.m\_ulValue[i]=(unsigned **long**)(div/A);

mul=(div/A)\*A;

carry=(unsigned **long**)(div-mul);

}

**if**(X.m\_ulValue[X.m\_nLength-1]==0)X.m\_nLength--;

**return** X;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数求模

调用形式：N.Mod(A)

返回值：N%A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::Mod(CBigInt& A)

{

CBigInt X,Y;

unsigned \_\_int64 div,num;

unsigned **long** carry=0;

unsigned i,len;

X.Mov(\***this**);

**while**(X.Cmp(A)>=0)

{

div=X.m\_ulValue[X.m\_nLength-1];

num=A.m\_ulValue[A.m\_nLength-1];

len=X.m\_nLength-A.m\_nLength;

**if**((div==num)&&(len==0)){X.Mov(X.Sub(A));**break**;}

**if**((div<=num)&&len){len--;div=(div<<32)+X.m\_ulValue[X.m\_nLength-2];}

div=div/(num+1);

Y.Mov(div);

Y.Mov(A.Mul(Y));

**if**(len)

{

Y.m\_nLength+=len;

**for**(i=Y.m\_nLength-1;i>=len;i--)Y.m\_ulValue[i]=Y.m\_ulValue[i-len];

**for**(i=0;i<len;i++)Y.m\_ulValue[i]=0;

}

X.Mov(X.Sub(Y));

}

**return** X;

}

unsigned **long** CBigInt::Mod(unsigned **long** A)

{

**if**(m\_nLength==1)**return**(m\_ulValue[0]%A);

unsigned \_\_int64 div;

unsigned **long** carry=0;

**for**(**int** i=m\_nLength-1;i>=0;i--)

{

div=m\_ulValue[i];

div+=carry\*0x100000000;

carry=(unsigned **long**)(div%A);

}

**return** carry;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

从字符串按10进制或16进制格式输入到大数

调用格式：N.Get(str,sys)

返回值：N被赋值为相应大数

sys暂时只能为10或16

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** CBigInt::Get(**char** str[], unsigned **int** system)

{

**int** len=strlen(str),k;

Mov(0);

**for**(**int** i=0;i<len;i++)

{

Mov(Mul(system));

**if**((str[i]>='0')&&(str[i]<='9'))k=str[i]-48;

**else** **if**((str[i]>='A')&&(str[i]<='F'))k=str[i]-55;

**else** **if**((str[i]>='a')&&(str[i]<='f'))k=str[i]-87;

**else** k=0;

Mov(Add(k));

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

将大数按10进制或16进制格式输出为字符串

调用格式：N.Put(str,sys)

返回值：无，参数str被赋值为N的sys进制字符串

sys暂时只能为10或16

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** CBigInt::Put(**char** str[], unsigned **int** system)

{

**if**((m\_nLength==1)&&(m\_ulValue[0]==0)){str="0";**return**;}

**char** t[]="0123456789ABCDEF";

**int** a;

**char** ch;

CBigInt X;

X.Mov(\***this**);

**int** i = 0;

**while**(X.m\_ulValue[X.m\_nLength-1]>0)

{

a=X.Mod(system);

ch=t[a];

str[i++] = ch;

X.Mov(X.Div(system));

}

str[i] = 0x00;

**int** len = strlen(str) - 1;

**int** half\_len = strlen(str) / 2;

**char** tmp;

**for** (i = 0; i<half\_len; i++)

{

tmp = str[i];

str[i] = str[len-i];

str[len-i] = tmp;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

求不定方程ax-by=1的最小整数解

调用方式：N.Euc(A)

返回值：X,满足：NX mod A=1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::Euc(CBigInt& A)

{

CBigInt M,E,X,Y,I,J;

**int** x,y;

M.Mov(A);

E.Mov(\***this**);

X.Mov(0);

Y.Mov(1);

x=y=1;

**while**((E.m\_nLength!=1)||(E.m\_ulValue[0]!=0))

{

I.Mov(M.Div(E));

J.Mov(M.Mod(E));

M.Mov(E);

E.Mov(J);

J.Mov(Y);

Y.Mov(Y.Mul(I));

**if**(x==y)

{

**if**(X.Cmp(Y)>=0)Y.Mov(X.Sub(Y));

**else**{Y.Mov(Y.Sub(X));y=0;}

}

**else**{Y.Mov(X.Add(Y));x=1-x;y=1-y;}

X.Mov(J);

}

**if**(x==0)X.Mov(A.Sub(X));

**return** X;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

求乘方的模

调用方式：N.RsaTrans(A,B)

返回值：X=N^A MOD B

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::RsaTrans(CBigInt& A, CBigInt& B)

{

CBigInt X,Y;

**int** i,j,k;

unsigned n;

unsigned **long** num;

k=A.m\_nLength\*32-32;

num=A.m\_ulValue[A.m\_nLength-1];

**while**(num){num=num>>1;k++;}

X.Mov(\***this**);

**for**(i=k-2;i>=0;i--)

{

Y.Mov(X.Mul(X.m\_ulValue[X.m\_nLength-1]));

Y.Mov(Y.Mod(B));

**for**(n=1;n<X.m\_nLength;n++)

{

**for**(j=Y.m\_nLength;j>0;j--)Y.m\_ulValue[j]=Y.m\_ulValue[j-1];

Y.m\_ulValue[0]=0;

Y.m\_nLength++;

Y.Mov(Y.Add(X.Mul(X.m\_ulValue[X.m\_nLength-n-1])));

Y.Mov(Y.Mod(B));

}

X.Mov(Y);

**if**((A.m\_ulValue[i>>5]>>(i&31))&1)

{

Y.Mov(Mul(X.m\_ulValue[X.m\_nLength-1]));

Y.Mov(Y.Mod(B));

**for**(n=1;n<X.m\_nLength;n++)

{

**for**(j=Y.m\_nLength;j>0;j--)Y.m\_ulValue[j]=Y.m\_ulValue[j-1];

Y.m\_ulValue[0]=0;

Y.m\_nLength++;

Y.Mov(Y.Add(Mul(X.m\_ulValue[X.m\_nLength-n-1])));

Y.Mov(Y.Mod(B));

}

X.Mov(Y);

}

}

**return** X;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

拉宾米勒算法测试素数

调用方式：N.Rab()

返回值：若N为素数，返回1，否则返回0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**int** CBigInt::Rab()

{

unsigned i,j,pass;

**for**(i=0;i<550;i++){**if**(Mod(PrimeTable[i])==0)**return** 0;}

CBigInt S,A,I,K;

K.Mov(\***this**);

K.m\_ulValue[0]--;

**for**(i=0;i<5;i++)

{

pass=0;

A.Mov(rand()\*rand());

S.Mov(K);

**while**((S.m\_ulValue[0]&1)==0)

{

**for**(j=0;j<S.m\_nLength;j++)

{

S.m\_ulValue[j]=S.m\_ulValue[j]>>1;

**if**(S.m\_ulValue[j+1]&1)S.m\_ulValue[j]=S.m\_ulValue[j]|0x80000000;

}

**if**(S.m\_ulValue[S.m\_nLength-1]==0)S.m\_nLength--;

I.Mov(A.RsaTrans(S,\***this**));

**if**(I.Cmp(K)==0){pass=1;**break**;}

}

**if**((I.m\_nLength==1)&&(I.m\_ulValue[0]==1))pass=1;

**if**(pass==0)**return** 0;

}

**return** 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

产生随机素数

调用方法：N.GetPrime(bits)

返回值：N被赋值为一个bits位（0x100000000进制长度）的素数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** CBigInt::GetPrime(**int** bits)

{

unsigned i;

m\_nLength=bits;

begin:

**for**(i=0;i<m\_nLength;i++)m\_ulValue[i]=rand()\*0x10000+rand();

m\_ulValue[0]=m\_ulValue[0]|1;

**for**(i=m\_nLength-1;i>0;i--)

{

m\_ulValue[i]=m\_ulValue[i]<<1;

**if**(m\_ulValue[i-1]&0x80000000)m\_ulValue[i]++;

}

m\_ulValue[0]=m\_ulValue[0]<<1;

m\_ulValue[0]++;

**for**(i=0;i<550;i++){**if**(Mod(PrimeTable[i])==0)**goto** begin;}

CBigInt S,A,I,K;

K.Mov(\***this**);

K.m\_ulValue[0]--;

**for**(i=0;i<5;i++)

{

A.Mov(rand()\*rand());

S.Mov(K.Div(2));

I.Mov(A.RsaTrans(S,\***this**));

**if**(((I.m\_nLength!=1)||(I.m\_ulValue[0]!=1))&&(I.Cmp(K)!=0))**goto** begin;

}

}

**int** main()

{

**int** t;

**int** i, j;

CBigInt big\_a, big\_b, big\_ans;

**char** ans[2005], a[1005], b[1005];

**while** (scanf("%d", &t) != EOF)

{

**for** (i = 0; i<t; i++)

{

**if** (i != 0)

printf("/n");

scanf("%s%s", a, b);

big\_a.Get(a);

big\_b.Get(b);

big\_ans = big\_a.Add(big\_b);

big\_ans.Put(ans);

printf("Case %d:/n%s + %s = %s/n", i+1, a, b, ans);

}

}

**return** 0;

}

### 大数(精简版)

#include <cstdio>

#include <cstring>

#include <cstdlib>

//允许生成1120位（二进制）的中间结果

#define BI\_MAXLEN 105

#define DEC 10

#define HEX 16

**class** CBigInt

{

**public**:

//大数在0x100000000进制下的长度

unsigned m\_nLength;

//用数组记录大数在0x100000000进制下每一位的值

unsigned **long** m\_ulValue[BI\_MAXLEN];

CBigInt();

~CBigInt();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

基本操作与运算

Mov，赋值运算，可赋值为大数或普通整数，可重载为运算符“=”

Cmp，比较运算，可重载为运算符“==”、“!=”、“>=”、“<=”等

Add，加，求大数与大数或大数与普通整数的和，可重载为运算符“+”

Sub，减，求大数与大数或大数与普通整数的差，可重载为运算符“-”

Mul，乘，求大数与大数或大数与普通整数的积，可重载为运算符“\*”

Div，除，求大数与大数或大数与普通整数的商，可重载为运算符“/”

Mod，模，求大数与大数或大数与普通整数的模，可重载为运算符“%”

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** Mov(unsigned \_\_int64 A);

**void** Mov(CBigInt A);

CBigInt Add(CBigInt& A);

CBigInt Sub(CBigInt A);

CBigInt Mul(CBigInt& A);

CBigInt Div(CBigInt& A);

CBigInt Mod(CBigInt& A);

CBigInt Add(unsigned **long** A);

CBigInt Sub(unsigned **long** A);

CBigInt Mul(unsigned **long** A);

CBigInt Div(unsigned **long** A);

unsigned **long** Mod(unsigned **long** A);

**int** Cmp(CBigInt& A);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

输入输出

Get，从字符串按10进制或16进制格式输入到大数

Put，将大数按10进制或16进制格式输出到字符串

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** Get(**char** str[], unsigned **int** system=DEC);

**void** Put(**char** str[], unsigned **int** system=DEC);

};

//构造大数对象并初始化为零

CBigInt::CBigInt()

{

m\_nLength=1;

**for**(**int** i=0;i<BI\_MAXLEN;i++)m\_ulValue[i]=0;

}

//解构大数对象

CBigInt::~CBigInt()

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数比较

调用方式：N.Cmp(A)

返回值：若N<A返回-1；若N=A返回0；若N>A返回1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**int** CBigInt::Cmp(CBigInt& A)

{

**if**(m\_nLength>A.m\_nLength)**return** 1;

**if**(m\_nLength<A.m\_nLength)**return** -1;

**for**(**int** i=m\_nLength-1;i>=0;i--)

{

**if**(m\_ulValue[i]>A.m\_ulValue[i])**return** 1;

**if**(m\_ulValue[i]<A.m\_ulValue[i])**return** -1;

}

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数赋值

调用方式：N.Mov(A)

返回值：无，N被赋值为A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** CBigInt::Mov(CBigInt A)

{

m\_nLength=A.m\_nLength;

**for**(**int** i=0;i<BI\_MAXLEN;i++)m\_ulValue[i]=A.m\_ulValue[i];

}

**void** CBigInt::Mov(unsigned \_\_int64 A)

{

**if**(A>0xffffffff)

{

m\_nLength=2;

m\_ulValue[1]=(unsigned **long**)(A>>32);

m\_ulValue[0]=(unsigned **long**)A;

}

**else**

{

m\_nLength=1;

m\_ulValue[0]=(unsigned **long**)A;

}

**for**(**int** i=m\_nLength;i<BI\_MAXLEN;i++)m\_ulValue[i]=0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数相加

调用形式：N.Add(A)

返回值：N+A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::Add(CBigInt& A)

{

CBigInt X;

X.Mov(\***this**);

unsigned carry=0;

unsigned \_\_int64 sum=0;

**if**(X.m\_nLength<A.m\_nLength)X.m\_nLength=A.m\_nLength;

**for**(unsigned i=0;i<X.m\_nLength;i++)

{

sum=A.m\_ulValue[i];

sum=sum+X.m\_ulValue[i]+carry;

X.m\_ulValue[i]=(unsigned **long**)sum;

carry=(unsigned)(sum>>32);

}

X.m\_ulValue[X.m\_nLength]=carry;

X.m\_nLength+=carry;

**return** X;

}

CBigInt CBigInt::Add(unsigned **long** A)

{

CBigInt X;

X.Mov(\***this**);

unsigned \_\_int64 sum;

sum=X.m\_ulValue[0];

sum+=A;

X.m\_ulValue[0]=(unsigned **long**)sum;

**if**(sum>0xffffffff)

{

unsigned i=1;

**while**(X.m\_ulValue[i]==0xffffffff){X.m\_ulValue[i]=0;i++;}

X.m\_ulValue[i]++;

**if**(m\_nLength==i)m\_nLength++;

}

**return** X;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数相减

调用形式：N.Sub(A)

返回值：N-A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::Sub(CBigInt A)

{

CBigInt X;

X.Mov(\***this**);

**if**(X.Cmp(A)<=0){X.Mov(0);**return** X;}

unsigned carry=0;

unsigned \_\_int64 num;

unsigned i;

**for**(i=0;i<m\_nLength;i++)

{

**if**((m\_ulValue[i]>A.m\_ulValue[i])||((m\_ulValue[i]==A.m\_ulValue[i])&&(carry==0)))

{

X.m\_ulValue[i]=m\_ulValue[i]-carry-A.m\_ulValue[i];

carry=0;

}

**else**

{

num=0x100000000+m\_ulValue[i];

X.m\_ulValue[i]=(unsigned **long**)(num-carry-A.m\_ulValue[i]);

carry=1;

}

}

**while**(X.m\_ulValue[X.m\_nLength-1]==0)X.m\_nLength--;

**return** X;

}

CBigInt CBigInt::Sub(unsigned **long** A)

{

CBigInt X;

X.Mov(\***this**);

**if**(X.m\_ulValue[0]>=A){X.m\_ulValue[0]-=A;**return** X;}

**if**(X.m\_nLength==1){X.Mov(0);**return** X;}

unsigned \_\_int64 num=0x100000000+X.m\_ulValue[0];

X.m\_ulValue[0]=(unsigned **long**)(num-A);

**int** i=1;

**while**(X.m\_ulValue[i]==0){X.m\_ulValue[i]=0xffffffff;i++;}

X.m\_ulValue[i]--;

**if**(X.m\_ulValue[i]==0)X.m\_nLength--;

**return** X;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数相乘

调用形式：N.Mul(A)

返回值：N\*A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::Mul(CBigInt& A)

{

**if**(A.m\_nLength==1)**return** Mul(A.m\_ulValue[0]);

CBigInt X;

unsigned \_\_int64 sum,mul=0,carry=0;

unsigned i,j;

X.m\_nLength=m\_nLength+A.m\_nLength-1;

**for**(i=0;i<X.m\_nLength;i++)

{

sum=carry;

carry=0;

**for**(j=0;j<A.m\_nLength;j++)

{

**if**(((i-j)>=0)&&((i-j)<m\_nLength))

{

mul=m\_ulValue[i-j];

mul\*=A.m\_ulValue[j];

carry+=mul>>32;

mul=mul&0xffffffff;

sum+=mul;

}

}

carry+=sum>>32;

X.m\_ulValue[i]=(unsigned **long**)sum;

}

**if**(carry){X.m\_nLength++;X.m\_ulValue[X.m\_nLength-1]=(unsigned **long**)carry;}

**return** X;

}

CBigInt CBigInt::Mul(unsigned **long** A)

{

CBigInt X;

unsigned \_\_int64 mul;

unsigned **long** carry=0;

X.Mov(\***this**);

**for**(unsigned i=0;i<m\_nLength;i++)

{

mul=m\_ulValue[i];

mul=mul\*A+carry;

X.m\_ulValue[i]=(unsigned **long**)mul;

carry=(unsigned **long**)(mul>>32);

}

**if**(carry){X.m\_nLength++;X.m\_ulValue[X.m\_nLength-1]=carry;}

**return** X;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数相除

调用形式：N.Div(A)

返回值：N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::Div(CBigInt& A)

{

**if**(A.m\_nLength==1)**return** Div(A.m\_ulValue[0]);

CBigInt X,Y,Z;

unsigned i,len;

unsigned \_\_int64 num,div;

Y.Mov(\***this**);

**while**(Y.Cmp(A)>=0)

{

div=Y.m\_ulValue[Y.m\_nLength-1];

num=A.m\_ulValue[A.m\_nLength-1];

len=Y.m\_nLength-A.m\_nLength;

**if**((div==num)&&(len==0)){X.Mov(X.Add(1));**break**;}

**if**((div<=num)&&len){len--;div=(div<<32)+Y.m\_ulValue[Y.m\_nLength-2];}

div=div/(num+1);

Z.Mov(div);

**if**(len)

{

Z.m\_nLength+=len;

**for**(i=Z.m\_nLength-1;i>=len;i--)Z.m\_ulValue[i]=Z.m\_ulValue[i-len];

**for**(i=0;i<len;i++)Z.m\_ulValue[i]=0;

}

X.Mov(X.Add(Z));

Y.Mov(Y.Sub(A.Mul(Z)));

}

**return** X;

}

CBigInt CBigInt::Div(unsigned **long** A)

{

CBigInt X;

X.Mov(\***this**);

**if**(X.m\_nLength==1){X.m\_ulValue[0]=X.m\_ulValue[0]/A;**return** X;}

unsigned \_\_int64 div,mul;

unsigned **long** carry=0;

**for**(**int** i=X.m\_nLength-1;i>=0;i--)

{

div=carry;

div=(div<<32)+X.m\_ulValue[i];

X.m\_ulValue[i]=(unsigned **long**)(div/A);

mul=(div/A)\*A;

carry=(unsigned **long**)(div-mul);

}

**if**(X.m\_ulValue[X.m\_nLength-1]==0)X.m\_nLength--;

**return** X;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

大数求模

调用形式：N.Mod(A)

返回值：N%A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CBigInt CBigInt::Mod(CBigInt& A)

{

CBigInt X,Y;

unsigned \_\_int64 div,num;

unsigned **long** carry=0;

unsigned i,len;

X.Mov(\***this**);

**while**(X.Cmp(A)>=0)

{

div=X.m\_ulValue[X.m\_nLength-1];

num=A.m\_ulValue[A.m\_nLength-1];

len=X.m\_nLength-A.m\_nLength;

**if**((div==num)&&(len==0)){X.Mov(X.Sub(A));**break**;}

**if**((div<=num)&&len){len--;div=(div<<32)+X.m\_ulValue[X.m\_nLength-2];}

div=div/(num+1);

Y.Mov(div);

Y.Mov(A.Mul(Y));

**if**(len)

{

Y.m\_nLength+=len;

**for**(i=Y.m\_nLength-1;i>=len;i--)Y.m\_ulValue[i]=Y.m\_ulValue[i-len];

**for**(i=0;i<len;i++)Y.m\_ulValue[i]=0;

}

X.Mov(X.Sub(Y));

}

**return** X;

}

unsigned **long** CBigInt::Mod(unsigned **long** A)

{

**if**(m\_nLength==1)**return**(m\_ulValue[0]%A);

unsigned \_\_int64 div;

unsigned **long** carry=0;

**for**(**int** i=m\_nLength-1;i>=0;i--)

{

div=m\_ulValue[i];

div+=carry\*0x100000000;

carry=(unsigned **long**)(div%A);

}

**return** carry;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

从字符串按10进制或16进制格式输入到大数

调用格式：N.Get(str,sys)

返回值：N被赋值为相应大数

sys暂时只能为10或16

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** CBigInt::Get(**char** str[], unsigned **int** system)

{

**int** len=strlen(str),k;

Mov(0);

**for**(**int** i=0;i<len;i++)

{

Mov(Mul(system));

**if**((str[i]>='0')&&(str[i]<='9'))k=str[i]-48;

**else** **if**((str[i]>='A')&&(str[i]<='F'))k=str[i]-55;

**else** **if**((str[i]>='a')&&(str[i]<='f'))k=str[i]-87;

**else** k=0;

Mov(Add(k));

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

将大数按10进制或16进制格式输出为字符串

调用格式：N.Put(str,sys)

返回值：无，参数str被赋值为N的sys进制字符串

sys暂时只能为10或16

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** CBigInt::Put(**char** str[], unsigned **int** system)

{

**if**((m\_nLength==1)&&(m\_ulValue[0]==0)){str="0";**return**;}

**char** t[]="0123456789ABCDEF";

**int** a;

**char** ch;

CBigInt X;

X.Mov(\***this**);

**int** i = 0;

**while**(X.m\_ulValue[X.m\_nLength-1]>0)

{

a=X.Mod(system);

ch=t[a];

str[i++] = ch;

X.Mov(X.Div(system));

}

str[i] = 0x00;

**int** len = strlen(str) - 1;

**int** half\_len = strlen(str) / 2;

**char** tmp;

**for** (i = 0; i<half\_len; i++)

{

tmp = str[i];

str[i] = str[len-i];

str[len-i] = tmp;

}

}

**int** main()

{

**int** t;

**int** i, j;

CBigInt big\_a, big\_b, big\_ans;

**char** ans[2005], a[1005], b[1005];

**while** (scanf("%d", &t) != EOF)

{

**for** (i = 0; i<t; i++)

{

**if** (i != 0)

printf("\n");

scanf("%s%s", a, b);

big\_a.Get(a);

big\_b.Get(b);

big\_a.Put(a);

big\_b.Put(b);

big\_ans = big\_a.Add(big\_b);

big\_ans.Put(ans);

printf("Case %d:\n%s + %s = %s\n", i+1, a, b, ans);

}

}

**return** 0;

}

### Pollard\_rho进行质因数分解

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//pollard\_rho 算法进行质因数分解

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**long** **long** factor[100];//质因数分解结果（刚返回时是无序的）

**int** tol;//质因数的个数。数组小标从0开始

**long** **long** gcd(**long** **long** a,**long** **long** b)

{

**if**(a==0)**return** 1;//???????

**if**(a<0) **return** gcd(-a,b);

**while**(b)

{

**long** **long** t=a%b;

a=b;

b=t;

}

**return** a;

}

**long** **long** Pollard\_rho(**long** **long** x,**long** **long** c)

{

**long** **long** i=1,k=2;

**long** **long** x0=rand()%x;

**long** **long** y=x0;

**while**(1)

{

i++;

x0=(mult\_mod(x0,x0,x)+c)%x;

**long** **long** d=gcd(y-x0,x);

**if**(d!=1&&d!=x) **return** d;

**if**(y==x0) **return** x;

**if**(i==k){y=x0;k+=k;}

}

}

//对n进行素因子分解

**void** findfac(**long** **long** n)

{

**if**(Miller\_Rabin(n))//素数

{

factor[tol++]=n;

**return**;

}

**long** **long** p=n;

**while**(p>=n)p=Pollard\_rho(p,rand()%(n-1)+1);

findfac(p);

findfac(n/p);

}

### NTT快速数论变换

//第二类斯特林数的快速求解。ZOJ 3899

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 100010

#define lson x<<1

#define rson x<<1|1

#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<LL,LL> PII;

**const** LL INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

**const** LL P = 880803841LL;//119\*2^23 + 1

**const** LL G = 26;

**const** LL NUM = 26; //2^NUM > P

LL wn[NUM];

LL a[N\*5], b[N\*5];//这里要小心数组的大小要开大，一般原数组\*4

LL qpow\_mod(LL x, LL k, LL m)

{

LL res = 1;

**while**(k)

{

**if**(k & 1) res = x \* res % m;

x = x\*x%m;

k >>= 1;

}

**return** res;

}

**void** getwn()

{

**for**(LL i = 0; i < NUM; i++)

{

LL t = 1LL << i;

wn[i] = qpow\_mod(G, (P-1)/t, P);

}

}

**void** Rader(LL a[], LL len)

{

LL j = len >> 1;

**for**(LL i = 1; i < len - 1; i ++)

{

**if**(i < j) swap(a[i], a[j]);

LL k = len >> 1;

**while**(j >= k)

{

j -= k;

k >>= 1;

}

**if**(j < k) j += k;

}

}

**void** NTT(LL a[], LL len, LL on)

{

Rader(a, len);

LL id = 0;

**for**(LL h = 2; h <= len; h <<= 1)

{

id ++;

**for**(LL j = 0; j < len; j += h){

LL w = 1;

**for**(LL k = j; k < j + h / 2; k ++)

{

LL u = a[k] % P;

LL t = w \* (a[k + h / 2] % P) % P;

a[k] = (u + t) % P;

a[k + h / 2] = ((u - t) % P + P) % P;

w = w \* wn[id] % P;

}

}

}

**if**(on == -1){

**for**(LL i = 1; i < len / 2; i++)

swap(a[i], a[len - i]);

LL inv = qpow\_mod(len, P - 2, P);

**for**(LL i = 0; i < len; i++)

a[i] = a[i] % P \* inv % P;

}

}

//求卷积

**void** Conv(LL a[], LL b[], LL n)

{

NTT(a, n, 1);

NTT(b, n, 1);

**for**(LL i = 0; i < n; i++)

a[i] = a[i] \* b[i] % P;

NTT(a, n, -1);

}

LL f[N], invf[N];

//这边是线段树部分，无关的地方

struct node

{

LL l, r, lazy, cnt;

node(){}

node(LL ll, LL rr){l = ll, r = rr, lazy = 0, cnt = r - l + 1;}

**void** UPDATE()

{

cnt = r - l + 1 - cnt;

}

}lt[N \* 8];

**void** build(LL l, LL r, LL x)

{

lt[x] = node(l, r);

**if**(l == r) **return** ;

build(l, mid, lson);

build(mid+1, r, rson);

}

**void** push\_up(LL x)

{

lt[x].cnt = lt[lson].cnt + lt[rson].cnt;

}

**void** push\_down(LL x)

{

**if**(lt[x].lazy){

lt[lson].lazy ^= 1;

lt[rson].lazy ^= 1;

lt[x].lazy = 0;

lt[lson].UPDATE();

lt[rson].UPDATE();

}

}

**void** update(LL l, LL r, LL x)

{

**if**(lt[x].l >= l && lt[x].r <= r)

{

lt[x].UPDATE();

lt[x].lazy ^= 1;

**return** ;

}

push\_down(x);

**if**(r <= mid) update(l, r, lson);

**else** **if**(l > mid) update(l, r, rson);

**else** update(l, mid, lson), update(mid+1, r, rson);

push\_up(x);

}

**int** main()

{

// Open();

f[0] = invf[0] = 1;

**for**(LL i = 1; i < N; i ++)

{

f[i] = f[i - 1] \* i % P;

invf[i] = qpow\_mod(f[i], P - 2, P);

}

getwn();

LL T;scanf("%lld", &T);

**while**(T--)

{

LL n, m, d;

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

LL len = 1;

**while**(len <= m \* 2) len <<= 1;

memset(a, 0, sizeof(a));

memset(b, 0, sizeof(b));

**for**(LL i = 0; i <= n; i++){

a[i] = (((i & 1) ? -1 : 1) \* invf[i] + P) % P;

b[i] = qpow\_mod(i, n, P) \* invf[i] % P;

}

Conv(a, b, len);

//预处理数组的过程。

build(1, m, 1);

**while**(d--)

{

LL x, y;

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

update(x, y, 1);

printf("%lld\n", a[lt[1].cnt]);

}

}

**return** 0;

}

### Miller\_Rabin算法进行素数测试(random)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Miller\_Rabin 算法进行素数测试

//速度快，而且可以判断 <2^63的数

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**const** **int** S=20;//随机算法判定次数，S越大，判错概率越小

//计算 (a\*b)%c. a,b都是long long的数，直接相乘可能溢出的

// a,b,c <2^63

**long** **long** mult\_mod(**long** **long** a,**long** **long** b,**long** **long** c)

{

a%=c;

b%=c;

**long** **long** ret=0;

**while**(b)

{

**if**(b&1){ret+=a;ret%=c;}

a<<=1;

**if**(a>=c)a%=c;

b>>=1;

}

**return** ret;

}

//计算 x^n %c

**long** **long** pow\_mod(**long** **long** x,**long** **long** n,**long** **long** mod)//x^n%c

{

**if**(n==1)**return** x%mod;

x%=mod;

**long** **long** tmp=x;

**long** **long** ret=1;

**while**(n)

{

**if**(n&1) ret=mult\_mod(ret,tmp,mod);

tmp=mult\_mod(tmp,tmp,mod);

n>>=1;

}

**return** ret;

}

//以a为基,n-1=x\*2^t a^(n-1)=1(mod n) 验证n是不是合数

//一定是合数返回true,不一定返回false

bool check(**long** **long** a,**long** **long** n,**long** **long** x,**long** **long** t)

{

**long** **long** ret=pow\_mod(a,x,n);

**long** **long** last=ret;

**for**(**int** i=1;i<=t;i++)

{

ret=mult\_mod(ret,ret,n);

**if**(ret==1&&last!=1&&last!=n-1) **return** **true**;//合数

last=ret;

}

**if**(ret!=1) **return** **true**;

**return** **false**;

}

// Miller\_Rabin()算法素数判定

//是素数返回true.(可能是伪素数，但概率极小)

//合数返回false;

bool Miller\_Rabin(**long** **long** n)

{

**if**(n<2)**return** **false**;

**if**(n==2)**return** **true**;

**if**((n&1)==0) **return** **false**;//偶数

**long** **long** x=n-1;

**long** **long** t=0;

**while**((x&1)==0){x>>=1;t++;}

**for**(**int** i=0;i<S;i++)

{

**long** **long** a=rand()%(n-1)+1;//rand()需要stdlib.h头文件

**if**(check(a,n,x,t))

**return** **false**;//合数

}

**return** **true**;

}

### Lucas定理-求组合数取模

//C(n,m)%p=C(n/p,m/p)C(n%p,m%p), p为素数

ll qPow (ll a, ll k, ll p) {

ll ans = 1;

**while** (k) {

**if** (k&1)

ans = (ans \* a) % p;

a = (a \* a) % p;

k /= 2;

}

**return** ans;

}

ll C (ll a, ll b, ll p) {

**if** (a < b)

**return** 0;

**if** (b > a - b)

b = a - b;

ll up = 1, down = 1;

**for** (ll i = 0; i < b; i++) {

up = up \* (a-i) % p;

down = down \* (i+1) % p;

}

**return** up \* qPow(down, p-2, p) % p; // 逆元

}

ll lucas (ll a, ll b, ll p) {

**if** (b == 0)

**return** 1;

**return** C(a%p, b%p, p) \* lucas(a/p, b/p, p) % p;

}

/\*

const int maxn = 25;

const ll mod = 1e9+7;

int n;

ll s, f[maxn];

ll solve () {

ll ans = 0;

for (int i = 0; i < (1<<n); i++) {

ll sign = 1, sum = s;

for (int j = 0; j < n; j++) {

if (i&(1<<j)) {

sum -= (f[j]+1);

sign \*= -1;

}

}

if (sum < 0)

continue;

ans += sign \* lucas(sum + n - 1, n - 1, mod);

ans %= mod;

}

return (ans + mod) % mod;

}

int main () {

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

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

scanf("%lld", &f[i]);

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

return 0;

}

\*/

### Hiho1230 FWT快速沃尔什变换

/\*

\* dp[i][j]=sigma(dp[i-1][k]\*b[f(k^j)]

\*

\* 这个吊模板就是用来解决上述式子的，更为标准的是CF#259div1D那个题中的表达式。

\* 可以很快速的求解dp[n]的每一项。复杂度为m\*log(n)，m为第二维大小

\*

\* 这个题，我的想法是这样的：首先枚举X，那么也就是需要在[x, m+x]中找出n个数，使得这n个数异或和为0，

\* 然后方案数加起来即可。那么对于每个枚举的X来说，我们记dp[i][j]表示前i个数，异或和为j的方案数，

\* 那么dp[i][j]=sigma(dp[i-1][k]\*b[k^j])。其中b[i]=1当且仅当x <= i <=x+m，否则为0；那么上述式子其实

\* 和上一题CF的题就没多大区别了。然后就直接套用fwt，当然这里需要注意的是模数必须乘上做fwt的那个数组

\* 的大小len(这里的len也和NTT,FFT一样必须是大于等于N的最小二次幂数)。最后答案也需要除以这个len。

\* 下面的解法中，直接用a[i]数组pow\_mod,我认为是因为dp[0][j]的初始值都与b[i]数组相同；

\* 验证发现的确两种方法都可以AC。

\*/

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 100010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**const** LL basemod = 1000000007;

LL mod = basemod;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

//防爆long long mul\_mod

LL mul(LL a, LL b, LL p) {

**return** (a \* b - (LL)((**long** **double**)a / p \* b + 1e-3) \* p + p) % p;

}

**void** arrMul(LL n, LL \*c, LL \*a, LL \*b) {

**for** (**int** i = 0; i < n; ++i)

c[i] = mul(a[i], b[i], mod);

}

LL pow\_mod(LL x, LL k, LL mod)

{

LL res = 1;

**while**(k)

{

**if**( k & 1) res = mul(res, x, mod);

x = mul(x, x, mod);

k >>= 1;

}

**return** res;

}

**void** FWT(LL \*a, **int** n) {

**if** (n == 1) **return**;

**int** m = n >> 1;

FWT(a, m);

FWT(a + m, m);

**for** (**int** i = 0; i < m; ++i) {

LL u = a[i], v = a[i + m];

a[i] = (u + v) % mod;

a[i + m] = (u - v + mod) % mod;

}

}

**void** dFWT(LL \*a, **int** n) {

**if** (n == 1) **return**;

**int** m = n >> 1;

**for** (**int** i = 0; i < m; ++i) {

LL u = a[i], v = a[i + m];

a[i] = (u + v) % mod;

a[i + m] = (u - v + mod) % mod;

}

dFWT(a, m);

dFWT(a + m, m);

}

**void** Conv(LL \*A, LL \*B, **int** n, **int** t)//t -> 运算次数，n -> 运算长度, B -> 为转换数组, A -> 结果/初值数组。

{

FWT(A, n);

FWT(B, n);

**for** (; t; t >>= 1, arrMul(n, B, B, B)) //类似快速幂

**if** (t & 1) arrMul(n, A, A, B);

dFWT(A, n);

}

LL a[1<<12], b[1<<12];

LL solve(**int** n, **int** m, **int** L, **int** R)

{

LL len = 1;

**while**(len <= R) len \*= 2;

mod = basemod \* len;

**for**(**int** i = 0; i < len ;i++)

a[i] = b[i] = (i >= L && i <= R);

// FWT(a, len);

// for(int i = 0; i < len; i++)

// a[i] = pow\_mod(a[i], 2 \* n + 1, mod);

// dFWT(a, len);

Conv(a, b, len, 2\*n);

**return** (a[0] + mod)%mod/len;

}

**int** main()

{

Open();

**int** n, m, L, R;

**while**(~scanf("%d%d%d%d", &n, &m, &L, &R))

{

LL ans = 0;

**for**(**int** i = L; i <= R; i++)

ans = (ans + solve(n, m, i, i+m))%basemod;

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

}

**return** 0;

}

## 图论及树

### 最小生成树Prim

//最小生成树

#include <cstdio>

#include <algorithm>

#include <cstring>

#include <cmath>

#include <queue>

#define N 111

using namespace std;

**const** **int** INF=0x3f3f3f3f;

**int** last[N];//

struct edge

{

**int** u,v,w,nxt;

edge(){}

edge(**int** uu,**int** vv,**int** ww,**int** n):u(uu),v(vv),w(ww),nxt(n){}

}e[N\*N];

typedef pair<**int**,**int**> PII;

**int** ans,n,m;//

bool vis[N];//

**int** prim(**int** s)

{

priority\_queue<PII,vector<PII>,greater<PII> > que;

ans=0;

que.push(PII(0,s));

**int** prev=0;

**int** num=0;

**while**(!que.empty())

{

**int** u=que.top().second;

**int** w=que.top().first;

que.pop();

**if**(vis[u]) **continue**;

ans+=w;

vis[u]=**true**;

num++;

**if**(num==n) **return** ans;

**for**(**int** i=last[u];~i;i=e[i].nxt)

{

**int** v=e[i].v;

w=e[i].w;

**if**(vis[v]) **continue**;

que.push(PII(w,v));

}

}

**if**(num!=n) **return** -1;

}

//////////

### 次小生成树

//次小生成树

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <cstring>

#include <cmath>

#include <queue>

#define N 111

using namespace std;

**const** **int** INF=0x3f3f3f3f;

**int** last[N];//

struct edge

{

**int** u,v,w,nxt;

edge(){}

edge(**int** uu,**int** vv,**int** ww,**int** n):u(uu),v(vv),w(ww),nxt(n){}

}e[N\*N\*2];

typedef pair<**int**,pair<**int**,**int**> > PII;//三元组

**int** ans,n,m;//

bool vis[N];//

**int** pre[N];//

**int** eMax[N][N];//最小生成树中u,v路径上的最大边

**int** prim(**int** s)

{

priority\_queue<PII,vector<PII>,greater<PII> > que;

ans=0;

que.push(PII(0,pair<**int**,**int**>(0,s)));

**int** num=0;

**while**(!que.empty())

{

**int** prev=que.top().second.first;

**int** u=que.top().second.second;

**int** w=que.top().first;

que.pop();

**if**(vis[u]) **continue**;

ans+=w;

vis[u]=**true**;

num++;

pre[u]=prev;

**for**(**int** i=1;i<=n;i++)

{

**if**(vis[i] && i!=u )

{

eMax[i][u]=eMax[u][i]=max(eMax[i][pre[u]],w);

}

}

**if**(num==n) **return** ans;

**for**(**int** i=last[u];i!=-1;i=e[i].nxt)

{

**int** v=e[i].v;

w=e[i].w;

**if**(vis[v]) **continue**;

que.push(PII(w,pair<**int**,**int**>(u,v)));

}

}

**if**(num!=n) **return** -1;

}

**int** main()

{

**int** t;

scanf("%d",&t);

**while**(t--){

memset(last,-1,sizeof(last));

memset(vis,0,sizeof(vis));

memset(eMax,0,sizeof(eMax));

memset(pre,0,sizeof(pre));

ans=0;

**int** edgeNum=0;

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

**for**(**int** i=0;i<m;i++)

{

**int** u,v,w;

scanf("%d%d%d",&u,&v,&w);

e[edgeNum]=edge(u,v,w,last[u]),last[u]=edgeNum++;

e[edgeNum]=edge(v,u,w,last[v]),last[v]=edgeNum++;

}

ans=prim(1);

**if**(ans==-1)

{

printf("0\n");

**continue**;

}

bool flag=**true**;

**for**(**int** i=0;i<edgeNum && flag;i+=2)

{

**int** u=e[i].u,v=e[i].v,w=e[i].w;

**if**(pre[u]==v || pre[v]==u)

**continue**;

**if**(w==eMax[u][v]) flag=**false**;

}

**if**(!flag)

printf("Not Unique!\n");

**else**

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

}

**return** 0;

}

### HDU 4453 Splay

/\*

\* HDU 4453 Looploop

\* 。。。这个题太恶心了，写得精疲力尽，但是无疑是对splay又更加理解了

\* 六种操作：

\* 1. add x: 找到树中的第k2+1个点，旋转为根，然后给左儿子打标记即可

\* 2. reverse: 找到树中的第k1+1个点，旋转为根，然后给左儿子打标记即可

\* 3. insert x: 啊。。我这里是比较丑的写法，直接找到最左边的儿子，

\* 把新节点变成这个节点的父亲，然后暴力往上push\_up。。但是

\* 其实利用splay旋转的性质完全可以不这么做，比如说将最左边

\* 儿子旋转到根，然后再插入，这样更新会方便一些吧。

\* 4. delete: 就只是删除最左边一个点而已，旋转到根，然后删。

\* 5. move x: 将第一个点放到最后去，也就是将第一个点删除，然后给尾巴添加

\* 一个新节点。挺简单的一个操作。不过我这里写的太挫了。。

\* 6. query: 这个不用说啦，找到最左一个点，然后输出即可。

\*

\* 需要注意的就是push\_down, push\_up需要随时考虑用不用

\*/

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 300010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

**int** a[N];

**int** val[N];

**int** pre[N], ch[N][2], add[N], flip[N], sz[N];

**int** tot, root;

struct Splay

{

**void** Treaval(**int** x) {

**if**(x) {

Treaval(ch[x][0]);

printf("结点%2d:左儿子 %2d 右儿子 %2d 父结点 %2d size = %2d ,key = %2d \n",x, ch[x][0], ch[x][1], pre[x], sz[x], val[x]);

Treaval(ch[x][1]);

}

}

**void** init()

{

tot = root = 0;

}

**int** newnode(**int** fa, **int** v)

{

**int** k = ++tot;

ch[k][0] = ch[k][1] = 0;

pre[k] = fa;

val[k] = v;

add[k] = 0, flip[k] = 0;

sz[k] = 1;

**return** k;

}

**void** push\_down(**int** x)

{

**if**(add[x]){

**int** lc = ch[x][0];

**int** rc = ch[x][1];

**if**(lc != 0) val[lc] += add[x], add[lc] += add[x];

**if**(rc != 0) val[rc] += add[x], add[rc] += add[x];

add[x] = 0;

}

**if**(flip[x]){

flip[x] = 0;

swap(ch[x][0], ch[x][1]);

**if**(ch[x][0] != 0) flip[ch[x][0]] ^= 1;

**if**(ch[x][1] != 0) flip[ch[x][1]] ^= 1;

}

}

**void** push\_up(**int** x)

{

sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + 1;

}

**void** rotate(**int** x)

{

**int** y = pre[x], d = (ch[y][1] == x);

push\_down(y);push\_down(x);

ch[y][d] = ch[x][!d];

**if**(ch[x][!d]) pre[ch[x][!d]] = y;

ch[x][!d] = y;

pre[x] = pre[y];

pre[y] = x;

**if**(pre[x]) ch[pre[x]][ch[pre[x]][1] == y] = x;

push\_up(y);

push\_up(x);

}

**void** splay(**int** x, **int** goal)

{

**while**(pre[x] != goal){

**int** f = pre[x], ff = pre[f];

**if**(ff == goal) rotate(x);

**else** **if**((ch[ff][1] == f) == (ch[f][1] == x))

rotate(f), rotate(x);

**else**

rotate(x), rotate(x);

}

push\_up(x);

**if**(goal == 0) root = x;

}

**int** kth(**int** k)

{

**int** x = root;

**while**(x)

{

push\_down(x);

**if**(sz[ch[x][0]] >= k) x = ch[x][0];

**else** {

k -= sz[ch[x][0]] + 1;

**if**(k == 0) **return** x;

x = ch[x][1];

}

}

**return** x;

}

**int** build(**int** l, **int** r, **int** fa){

**if**(l > r) **return** 0;

**int** mid = l + r >> 1;

**int** k = newnode(fa, a[mid]);

ch[k][0] = build(l, mid-1, k);

ch[k][1] = build(mid+1, r, k);

push\_up(k);

**return** k;

}

**void** increase(**int** k, **int** x)

{

**int** idx = kth(k + 1);

**if**(idx == 0){

val[root] += x;

add[root] += x;

**return** ;

}

splay(idx, 0);

val[ch[idx][0]] += x;

add[ch[idx][0]] += x;

}

**void** reverse(**int** k)

{

**int** idx = kth(k + 1);

**if**(idx == 0){

flip[root] ^= 1;

**return** ;

}

splay(idx, 0);

// Treaval(root);

flip[ch[idx][0]] ^= 1;

}

**void** remove()

{

**int** x = root;

push\_down(x);

**while**(ch[x][0]) x = ch[x][0], push\_down(x);

splay(x, 0);

**if**(ch[x][0] == 0){

root = ch[x][1];

pre[ch[x][1]] = 0;

**return** ;

}

**int** y = ch[x][0];

push\_down(y);

**while**(ch[y][1]) y = ch[y][1], push\_down(y);

splay(y, x);

ch[y][1] = ch[x][1];

**if**(ch[x][1]) pre[ch[x][1]] = y;

pre[y] = 0;

root = y;

push\_up(y);

}

**void** insert(**int** v){

**if**(ch[root][0] == 0) splay(ch[root][1], 0);

**int** x = root;

push\_down(x);

**while**(ch[x][0]) x = ch[x][0], push\_down(x);

**int** tmpk = newnode(pre[x], v);

ch[tmpk][0] = x, ch[tmpk][1] = ch[x][1];

ch[pre[x]][0] = tmpk;

**if**(ch[x][1] != 0) pre[ch[x][1]] = tmpk;

ch[x][1] = 0;

pre[x] = tmpk;

**while**(x) push\_up(x), x = pre[x];

}

**void** move(**int** kind){

**if**(kind == 1){

**int** x = root;push\_down(x);

**while**(ch[x][1]) x = ch[x][1], push\_down(x);

splay(x, 0);

**int** y = ch[x][0];push\_down(y);

**while**(ch[y][1]) y = ch[y][1], push\_down(y);

splay(y, x);

pre[y] = 0;

root = y;

push\_up(y);

ch[x][1] = ch[x][0] = 0;

y = root;push\_down(y);

**while**(ch[y][0]) y = ch[y][0], push\_down(y);

splay(y, 0);

ch[y][0] = x; pre[x] = y;

sz[x] = 1;

push\_up(y);

}**else**{

**int** x = root; push\_down(x);

**while**(ch[x][0]) x = ch[x][0], push\_down(x);

splay(x, 0);

**int** y = ch[x][1]; push\_down(y);

**while**(ch[y][0]) y = ch[y][0], push\_down(y);

splay(y, x);

pre[y] = 0; root = y;

push\_up(y);

ch[x][1] = ch[x][0] = 0;

y = root; push\_down(y);

**while**(ch[y][1]) y = ch[y][1], push\_down(y);

splay(y, 0);

ch[y][1] = x, pre[x] = y;

sz[x] = 1; push\_up(y);

}

}

**int** query()

{

**int** x = root;push\_down(x);

**while**(ch[x][0]) x = ch[x][0], push\_down(x);

**return** val[x];

}

}spl;

**char** tmp[22];

**int** main()

{

Open();

**int** n, m, k1, k2;

**int** cas = 1;

**while**(scanf("%d%d%d%d", &n, &m, &k1, &k2), n||m||k1||k2)

{

**for**(**int** i = 1; i <= n; i++) scanf("%d", &a[i]);

spl.init();

root = spl.build(1, n, 0);

// spl.Treaval(root);

printf("Case #%d:\n", cas++);

**while**(m --)

{

scanf("%s", tmp);

**if**(tmp[0] == 'm')

{

**int** kind;

scanf("%d", &kind);

spl.move(kind);

}

**if**(tmp[0] == 'q')

{

printf("%d\n", spl.query());

}

**if**(tmp[0] == 'i')

{

**int** v;scanf("%d", &v);

spl.insert(v);

}

**if**(tmp[0] == 'r')

{

spl.reverse(k1);

}

**if**(tmp[0] == 'a')

{

**int** v;scanf("%d", &v);

spl.increase(k2, v);

}

**if**(tmp[0] == 'd')

{

spl.remove();

}

// printf("--------------------------%s------------------------\n", tmp);

// spl.Treaval(root);

**int** asdfasd = 11212;

}

}

**return** 0;

}

### Floyd最短路

**int** d[N][N];

**int** n;

**void** floyd(**int** d[][N])

{

**for**(**int** k=0;k<n;k++)

**for**(**int** i=0;i<n;i++)

**for**(**int** j=0;j<n;j++)

d[i][j]=min(d[i][j],d[i][k]+d[k][j]);

}

////floyd 快速幂求解恰好有N条边的最短路

#include <iostream>

#include <cstring>

#include <cstdio>

#include <algorithm>

using namespace std;

#define N 222

**int** v[1111];

**int** tmp[N][N];

**int** n,t,s,e;

**int** d[N][N];

**int** ans[N][N];

**int** V;

**void** floyd(**int** target[][N],**int** a[][N],**int** b[][N])

{

memset(tmp,0x3f,sizeof(tmp));

**for**(**int** k=0;k<V;k++)

**for**(**int** i=0;i<V;i++)

**for**(**int** j=0;j<V;j++)

tmp[i][j]=min((**long** **long**)tmp[i][j],(**long** **long**)a[i][k]+b[k][j]);

**for**(**int** i=0;i<V;i++)

**for**(**int** j=0;j<V;j++)

target[i][j]=tmp[i][j];

}

**void** quickfloyd(**int** n)

{

**while**(n)

{

**if**(n&1)

{

floyd(ans,ans,d);

}

floyd(d,d,d);

n>>=1;

}

}

**int** main()

{

memset(v,-1,sizeof(v));

memset(d,0x3f,sizeof(d));

scanf("%d%d%d%d",&n,&t,&s,&e);

V=0;

**for**(**int** i=0;i<t;i++)

{

**int** a,b,l;

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

**if**(v[a]==-1) v[a]=V++;

**if**(v[b]==-1) v[b]=V++;

d[v[a]][v[b]]=l;

d[v[b]][v[a]]=l;

}

memset(ans,0x3f,sizeof(ans));

**for**(**int** i=0;i<V;i++)

ans[i][i]=0;

quickfloyd(n);

printf("%d\n",ans[v[s]][v[e]]);

**return** 0;

}

/////

### Dijkstra最短路

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <cstring>

#include <cmath>

#include <queue>

#define N 50050

using namespace std;

**const** **int** INF=0x3f3f3f3f3f3f3f3fL;

**int** c[N];

**int** last[N];

struct edge

{

**int** v,w,nxt;

edge(){}

edge(**int** vv,**int** ww,**int** n):v(vv),w(ww),nxt(n){}

}e[N\*2];

typedef pair<**int**,**int**> PII;

**int** ans,n,m;

**int** d[N];

priority\_queue<PII,vector<PII>,greater<PII> > que;

**void** dijkstra(**int** s)

{

memset(d,0x3f,sizeof(d));

//fill(d+1,d+V+1,INF);

d[s]=0;

que.push(PII(0,s));

**while**(!que.empty())

{

PII pp=que.top();que.pop();

**int** u=pp.second;

**if**(d[u]<pp.first) **continue**;

**for**(**int** i=last[u];i!=-1;i=e[i].nxt){

**int** v=e[i].v,w=e[i].w;

**if**(d[v]>d[u]+w)

{

d[v]=d[u]+w;

que.push(PII(d[v],v));

}

}

}

}

### Astar求K短路

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <queue>

using namespace std;

//const long long INF=0x7fffffffffffffffL;

#define N 100010

struct edge

{

**int** v,w,nxt;

}e[N],reve[N];

**int** last[1010];

**int** revlast[1010];

typedef pair<**int**,**int**> PII;

**int** V;

**int** d[1010];

priority\_queue<PII,vector<PII>,greater<PII> > que;

**void** dijkstra(**int** s)

{

memset(d,0x3f,sizeof(d));

//fill(d+1,d+V+1,INF);

d[s]=0;

que.push(PII(0,s));

**while**(!que.empty())

{

PII pp=que.top();que.pop();

**int** u=pp.second;

**if**(d[u]<pp.first) **continue**;

**for**(**int** i=revlast[u];i!=-1;i=reve[i].nxt){

**int** v=reve[i].v,w=reve[i].w;

**if**(d[v]>d[u]+w)

{

d[v]=d[u]+w;

que.push(PII(d[v],v));

}

}

}

}

struct A

{

**int** u,g,f;

bool operator< (**const** A& a) **const**

{

**return** f > a.f;

}

A(**int** uu=0,**int** gg=0,**int** hh=0){u=uu,g=gg,f=hh;}

};

**int** cnt[1010];

**int** Astar(**int** s,**int** t,**int** k)

{

**if**(s==t) k++;

priority\_queue<A> Aque;

Aque.push(A(s,0,0));

**while**(!Aque.empty())

{

A tmp=Aque.top();Aque.pop();

**int** u=tmp.u,g=tmp.g,f=tmp.f;

cnt[u]++;

**if**(cnt[u]==k && u==t) **return** f;

**for**(**int** i=last[u];i!=-1;i=e[i].nxt)

{

**int** v=e[i].v,w=e[i].w;

**if**(cnt[v]<k)

Aque.push(A(v,g+w,g+w+d[v]));

}

}

**return** -1;

}

**int** main()

{

**int** n,m;

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

memset(last,-1,sizeof(last));

memset(revlast,-1,sizeof(revlast));

**for**(**int** i=0;i<m;i++)

{

**int** u,v,w;

scanf("%d%d%d",&u,&v,&w);

e[i].v=v,e[i].w=w,e[i].nxt=last[u],last[u]=i;

reve[i].v=u,reve[i].w=w,reve[i].nxt=revlast[v],revlast[v]=i;

}

**int** s,t,k;

scanf("%d%d%d",&s,&t,&k);

dijkstra(t);

printf("%d\n",Astar(s,t,k));

**return** 0;

}

### 最小费用最大流(SPFA\_rujia)

struct Edge{

**int** u, v, cap, flow, cost;

};

struct MCMF{

**int** n, m, s, t;

vector<Edge> edges;

vector<**int**> G[N];

**int** inq[N];

**int** d[N];

**int** p[N];

**int** a[N];

**void** init(**int** n, **int** s, **int** t) {

**this** -> n = n;

**this** -> s = s;

**this** -> t = t;

**for**(**int** i = 0; i<n; i++) G[i].clear();

edges.clear();

}

**void** addedge(**int** u, **int** v, **int** cap, **int** cost){

edges.push\_back((Edge){u, v, cap, 0, cost});

edges.push\_back((Edge){v, u, 0, 0, -cost});

m = edges.size();

G[u].push\_back(m-2);

G[v].push\_back(m-1);

// cerr<<u<<" -> "<<v<<" : "<<cost<<endl;

}

bool SPFA(**int**& flow, **int**& cost){

**for**(**int** i=0;i<n;i++) d[i] = INF;

memset(inq, 0, sizeof(inq));

d[s] = 0; inq[s] = 1; p[s] = 0; a[s] = INF;

queue<**int**> Q;

Q.push(s);

**while**(!Q.empty()){

**int** u = Q.front(); Q.pop();

inq[u] = 0;

**for**(**int** i = 0; i < G[u].size(); i++) {

Edge& e = edges[G[u][i]];

**if**(e.cap > e.flow && d[e.v] > d[u] + e.cost){

d[e.v] = d[u] + e.cost;

p[e.v] = G[u][i];

a[e.v] = min(a[u], e.cap - e.flow);

**if**(!inq[e.v]) {Q.push(e.v); inq[e.v] = 1;}

}

}

}

**if**(d[t] >= 0) **return** **false**;

flow += a[t];

cost += d[t];

**int** u = t;

**while**(u != s){

edges[p[u]].flow += a[t];

edges[p[u]^1].flow -= a[t];

u = edges[p[u]].u;

}

**return** **true**;

}

**int** Mincost() {

**int** flow = 0, cost = 0;

**while**(SPFA(flow, cost));

**return** cost;

}

}mcmf;

### ZKW费用流

/\*

\* 比一般费用流快很多，适用于边非常密集的地方

\* 本模板是不能直接用于任何有负权的图，更不能用于有负圈的情况

\*/

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 100010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("F:/in.txt","r",stdin);

//freopen("F:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

struct ZKW\_flow{

**const** **static** **int** MAXN = 210;

**const** **static** **int** MAXE = 210\*210;

**int** st, ed, ecnt, n;

**int** head[MAXN];

**int** cap[MAXE], cost[MAXE], to[MAXE], next[MAXE];

**void** init(){

memset(head, 0, sizeof(head));

ecnt = 2;

}

//cc容量，ww花费

**void** addEdge(**int** u, **int** v, **int** cc, **int** ww){

cap[ecnt] = cc; cost[ecnt] = ww; to[ecnt] = v;

next[ecnt] = head[u]; head[u] = ecnt++;

cap[ecnt] = 0; cost[ecnt] = -ww; to[ecnt] = u;

next[ecnt] = head[v]; head[v] = ecnt++;

}

**int** dis[MAXN];

**void** SPFA(){

**for**(**int** i = 1; i <= n; ++i) dis[i] = INF;

priority\_queue<pair<**int**, **int**> > Q;

dis[st] = 0;

Q.push(make\_pair(0, st));

**while**(!Q.empty()){

**int** u = Q.top().second, d = -Q.top().first;

Q.pop();

**if**(dis[u] != d) **continue**;

**for**(**int** p = head[u]; p; p = next[p]){

**int** &v = to[p];

**if**(cap[p] && dis[v] > d + cost[p]){

dis[v] = d + cost[p];

Q.push(make\_pair(-dis[v], v));

}

}

}

**for**(**int** i = 1; i <= n; ++i) dis[i] = dis[ed] - dis[i];

}

**int** minCost, maxFlow;

bool use[MAXN];

**int** add\_flow(**int** u, **int** flow){

**if**(u == ed){

maxFlow += flow;

minCost += dis[st] \* flow;

**return** flow;

}

use[u] = **true**;

**int** now = flow;

**for**(**int** p = head[u]; p; p = next[p]){

**int** &v = to[p];

**if**(cap[p] && !use[v] && dis[u] == dis[v] + cost[p]){

**int** tmp = add\_flow(v, min(now, cap[p]));

cap[p] -= tmp;

cap[p^1] += tmp;

now -= tmp;

**if**(!now) **break**;

}

}

**return** flow - now;

}

bool modify\_label(){

**int** d = INF;

**for**(**int** u = 1; u <= n; ++u) **if**(use[u])

**for**(**int** p = head[u]; p; p = next[p]){

**int** &v = to[p];

**if**(cap[p] && !use[v]) d = min(d, dis[v] + cost[p] - dis[u]);

}

**if**(d == INF) **return** **false**;

**for**(**int** i = 1; i <= n; ++i) **if**(use[i]) dis[i] += d;

**return** **true**;

}

//nn表示图中点的数量，图中的点编号从1开始。

PII min\_cost\_flow(**int** ss, **int** tt, **int** nn){

st = ss, ed = tt, n = nn;

minCost = maxFlow = 0;

SPFA();

**while**(**true**){

**while**(**true**){

**for**(**int** i = 1; i <= n; ++i) use[i] = 0;

**if**(!add\_flow(st, INF)) **break**;

}

**if**(!modify\_label()) **break**;

}

**return** PII(minCost, maxFlow);

}

}G;

**int** n;

struct Point

{

**int** x, y, z, w;

**void** read()

{

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

}

inline **int** D(**int** x)

{

**return** x\*x;

}

inline **int** dist(Point& o)

{

**return** floor(sqrt((**double**)D(o.x - x) + D(o.y - y) + D(o.z - z)))+0.5;

}

}p[111];

**int** main()

{

// Open();

**while**(~scanf("%d", &n), n)

{

**int** allw = 0;

**for**(**int** i = 1; i <= n; i++)

{

p[i].read();

allw += p[i].w;

}

G.init();

**int** s = 2 \* n + 1, t = s + 1;

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

G.addEdge(s, i, p[i].w, 0);

G.addEdge(i+n, t, p[i].w, 0);

**for**(**int** j = 1; j <= n; j++){

**if**(i == j) **continue**;

G.addEdge(i, j+n, INF, p[i].dist(p[j]));

}

}

PII res = G.min\_cost\_flow(s, t, t);

**if**(res.second != allw) res.first = -1;

printf("%d\n", res.first);

}

**return** 0;

}

### 最小顶点覆盖/二分图最大匹配

**const** **int** MAXN=;//$

////////////////最小顶点覆盖/二分图最大匹配///////////////////

//下标从0开始

struct BPM{

**int** n,m;

vector<**int** > G[MAXN];

**int** left[MAXN];

bool T[MAXN];

**int** right[MAXN];

bool S[MAXN];

**void** init(**int** n,**int** m){

**this** -> n = n;

**this** -> m = m;

**for**(**int** i = 0;i < MAXN;i++) G[i].clear();

}

**void** add\_edge(**int** u,**int** v){

G[u].push\_back(v);

}

bool match(**int** u){

S[u] = **true**;

**for**(**int** i = 0;i < G[u].size();i++){

**int** v = G[u][i];

**if**(!T[v]){

T[v] = **true**;

**if**(left[v] == -1 || match(left[v])){

left[v] = u;

right[u] = v;

**return** **true**;

}

}

}

**return** **false**;

}

//最大匹配数

**int** solve(){

memset(left,-1,sizeof(left));

memset(right,-1,sizeof(right));

**int** ans = 0;

**for**(**int** u = 0;u < n;u++){

memset(S,0,sizeof(S));

memset(T,0,sizeof(T));

**if**(match(u)) ans++;

}

**return** ans;

}

//计算最小顶点覆盖，返回点数，覆盖哪些点分别在X，Y中

**int** mincover(vector<**int**>& X,vector<**int**>& Y){

**int** ans = solve();

memset(S,0,sizeof(S));

memset(T,0,sizeof(T));

**for**(**int** u = 0;u < n;u++)

**if**(right[u] == -1) match(u);

**for**(**int** u = 0;u < n;u++)

**if**(!S[u]) X.push\_back(u);

**for**(**int** v = 0;v < m;v++)

**if**(T[v]) Y.push\_back(v);

**return** ans;

}

}solver;

/\*how to use

solver.init(m,n);//左边m个点，右边n个点

...solver.AddEdge(u,v) //下标从0开始

vector<int > X,Y;

int point\_num=solver.mincover(X,Y);//最小覆盖点数，也是最大匹配数

//输出覆盖了哪些点

for(int i = 0;i < X.size();i++)

printf(" r%d",X[i]);

for(int j = 0;j < Y.size();j++)

printf(" c%d",Y[j]);

\*/

////////////////最小顶点覆盖///////////////////

**const** **int** maxn = 1000 + 5;

struct BPM{

**int** n,m;

vector<**int** > G[maxn];

**int** left[maxn];

bool T[maxn];

**int** right[maxn];

bool S[maxn];

**void** init(**int** n,**int** m){

**this** -> n = n;

**this** -> m = m;

**for**(**int** i = 0;i < maxn;i++) G[i].clear();

}

**void** AddEdge(**int** u,**int** v){

G[u].push\_back(v);

}

bool match(**int** u){

S[u] = **true**;

**for**(**int** i = 0;i < G[u].size();i++){

**int** v = G[u][i];

**if**(!T[v]){

T[v] = **true**;

**if**(left[v] == -1 || match(left[v])){

left[v] = u;

right[u] = v;

**return** **true**;

}

}

}

**return** **false**;

}

**int** solve(){

memset(left,-1,sizeof(left));

memset(right,-1,sizeof(right));

**int** ans = 0;

**for**(**int** u = 0;u < n;u++){

memset(S,0,sizeof(S));

memset(T,0,sizeof(T));

**if**(match(u)) ans++;

}

**return** ans;

}

**int** mincover(vector<**int**>& X,vector<**int**>& Y){

**int** ans = solve();

memset(S,0,sizeof(S));

memset(T,0,sizeof(T));

**for**(**int** u = 0;u < n;u++)

**if**(right[u] == -1) match(u);

**for**(**int** u = 0;u < n;u++)

**if**(!S[u]) X.push\_back(u);

**for**(**int** v = 0;v < n;v++)

**if**(T[v]) Y.push\_back(v);

**return** ans;

}

};

BPM solver;

**int** main(){

**int** n,r,c;

**while**(scanf("%d%d%d",&r,&c,&n)){

**if**(r == 0 && c == 0 && n == 0) **break**;

solver.init(r,c);

**while**(n--){

**int** x,y;

scanf("%d%d",&x,&y);x--;y--;//模版中编号是从0开始的

solver.AddEdge(x,y);

}

vector<**int**> X,Y;

printf("%d",solver.mincover(X,Y));

**for**(**int** i = 0;i < X.size();i++)

printf(" r%d",X[i]+1);//最后别忘了加回来

**for**(**int** j = 0;j < Y.size();j++)

printf(" c%d",Y[j]+1);

printf("\n");

}

**return** 0;

}

### 最近公共祖先

//最近公共祖先（LCA）

#include <iostream>

#include <cmath>

#include <cstdio>

#include <cstring>

using namespace std;

#include <vector>

vector<**int**> g[N];//

**int** root;

**int** parent[LOG\_N][N];

**int** depth[N];

**void** dfs(**int** v,**int** p,**int** d)

{

parent[0][v]=p;

depth[v]=d;

**for**(**int** i=0;i<g[v].size();i++)

{

**if**(g[v][i]!=p) dfs(g[v][i],v,d+1);

}

}

//预处理

**void** init(**int** V)//预处理出parent

{

root = 0; // or 1

dfs(root,-1,0);

**for**(**int** k=0;k+1<LOG\_N;k++)

{

**for**(**int** v=1;v<=n;v++)//这里根据顶点的起始下标改变

{

**if**(parent[k][v]<0) parent[k+1][v]=-1;

**else** parent[k+1][v]=parent[k][parent[k][v]];

}

}

}

//计算u和v的LCA

**int** lca(**int** u,**int** v)

{

//让u和v向上走到同一深度

**if**(depth[u]>depth[v]) swap(u,v);

**for**(**int** k=0;k<LOG\_N;k++)

**if**((depth[v]-depth[u])>>k&1)

v=parent[k][v];

**if**(u==v) **return** u;

//利用二分搜索计算LCA

**for**(**int** k=LOG\_N-1;k>=0;k--)

**if**(parent[k][u]!=parent[k][v])

u=parent[k][u], v=parent[k][v];

**return** parent[0][u];

}

//最近公共祖先（LCA）

//tarjan离线LCA，先将询问全部存下来，然后一次DFS求出所有询问的答案

struct info{

**int** u, v, id;

info(**int** u = 0, **int** v = 0, **int** id = 0):u(u), v(v), id(id){}

};

**int** n, m;

vector<**int**> g[N];

vector<PII> query[N];

vector<info> rt[N];

**int** Uto[N], toU[N], ma[N], mi[N], sn[N], ans[N], fa[N];

bool vis[N];

**void** init()

{

**for**(**int** i=0;i<=n;i++)

{

vis[i] = 0;

g[i].clear();

query[i].clear();

rt[i].clear();

fa[i] = i;

Uto[i] = toU[i] = 0;

ma[i] = mi[i] = sn[i];

}

}

**int** Find(**int** u)

{

**if**(fa[u] == u) **return** u;

**int** root = Find(fa[u]);

Uto[u] = max(Uto[u], max(Uto[fa[u]], ma[fa[u]] - mi[u]));

toU[u] = max(toU[u], max(toU[fa[u]], ma[u] - mi[fa[u]]));

ma[u] = max(ma[u], ma[fa[u]]);

mi[u] = min(mi[u], mi[fa[u]]);

**return** fa[u] = root;

}

**void** lca(**int** u)

{

vis[u] = 1;

**for**(**int** i=0;i<query[u].size();i++)

{

**int** v = query[u][i].first, id = query[u][i].second;

**if**(vis[v])

{

**int** root = Find(v);

**if**(id < 0) rt[root].push\_back(info(v, u, -id));

**else** rt[root].push\_back(info(u, v, id));

}

}

**for**(**int** i=0;i<g[u].size();i++)

{

**int** v = g[u][i];

**if**(vis[v]) **continue**;

lca(v);

fa[v] = u;

}

**for**(**int** i=0;i<rt[u].size(); i++)

{

**int** x = rt[u][i].u, y = rt[u][i].v, id = rt[u][i].id;

//cout<<"lca: "<<u<<"("<<x<<","<<y<<") ans:";

Find(x), Find(y);

ans[id] = max(Uto[x], max(toU[y], ma[y] - mi[x]));

//cout<<id<<":"<<ans[id]<<endl;

}

}

//离线LCA

### 最大流(ISAP-rujia)

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 222

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**const** **double** eps = 1e-9;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

**int** dcmp(**double** x)

{

**if**(fabs(x) < eps) **return** 0;

**return** x > eps;

}

struct Edge

{

**int** from, to;

**double** cap, flow;

};

struct ISAP

{

**int** n, m, s, t;

vector<Edge> edges;

vector<**int**> G[N]; // 邻接表，G[i][j]表示结点i的第j条边在e数组中的序号

bool vis[N]; // BFS使用

**int** d[N]; // 从起点到i的距离

**int** cur[N]; // 当前弧指针

**int** p[N]; // 可增广路上的上一条弧

**int** num[N]; // 距离标号计数

**void** AddEdge(**int** from, **int** to, **double** cap)

{

edges.push\_back((Edge)

{

from, to, cap, 0

});

edges.push\_back((Edge)

{

to, from, 0, 0

});

m = edges.size();

G[from].push\_back(m-2);

G[to].push\_back(m-1);

}

bool BFS()

{

memset(vis, 0, sizeof(vis));

queue<**int**> Q;

Q.push(t);

vis[t] = 1;

d[t] = 0;

**while**(!Q.empty())

{

**int** x = Q.front();

Q.pop();

**for**(**int** i = 0; i < G[x].size(); i++)

{

Edge& e = edges[G[x][i]^1];

**if**(!vis[e.from] && dcmp(e.cap - e.flow) > 0)

{

vis[e.from] = 1;

d[e.from] = d[x] + 1;

Q.push(e.from);

}

}

}

**return** vis[s];

}

**void** ClearAll(**int** n)

{

**this**->n = n;

**for**(**int** i = 0; i < n; i++) G[i].clear();

edges.clear();

}

**void** ClearFlow()

{

**for**(**int** i = 0; i < edges.size(); i++) edges[i].flow = 0;

}

**double** Augment()

{

**int** x = t;

**double** a = INF;

**while**(x != s)

{

Edge& e = edges[p[x]];

a = min(a, e.cap-e.flow);

x = edges[p[x]].from;

}

x = t;

**while**(x != s)

{

edges[p[x]].flow += a;

edges[p[x]^1].flow -= a;

x = edges[p[x]].from;

}

**return** a;

}

**double** Maxflow(**int** s, **int** t)

{

**this**->s = s;

**this**->t = t;

**double** flow = 0;

BFS();

memset(num, 0, sizeof(num));

**for**(**int** i = 0; i < n; i++) num[d[i]]++;

**int** x = s;

memset(cur, 0, sizeof(cur));

**while**(d[s] < n)

{

**if**(x == t)

{

flow += Augment();

x = s;

}

**int** ok = 0;

**for**(**int** i = cur[x]; i < G[x].size(); i++)

{

Edge& e = edges[G[x][i]];

**if**(dcmp(e.cap - e.flow) > 0 && d[x] == d[e.to] + 1) // Advance

{

ok = 1;

p[e.to] = G[x][i];

cur[x] = i; // 注意

x = e.to;

**break**;

}

}

**if**(!ok) // Retreat

{

**int** m = n-1; // 初值注意

**for**(**int** i = 0; i < G[x].size(); i++)

{

Edge& e = edges[G[x][i]];

**if**(dcmp(e.cap - e.flow) > 0) m = min(m, d[e.to]);

}

**if**(--num[d[x]] == 0) **break**;

num[d[x] = m+1]++;

cur[x] = 0; // 注意

**if**(x != s) x = edges[p[x]].from;

}

}

**return** flow;

}

vector<**int**> Mincut() // call this after maxflow

{

BFS();

vector<**int**> ans;

**for**(**int** i = 0; i < edges.size(); i++)

{

Edge& e = edges[i];

**if**(!vis[e.from] && vis[e.to] && e.cap > 0) ans.push\_back(i);

}

**return** ans;

}

**void** Reduce()

{

**for**(**int** i = 0; i < edges.size(); i++) edges[i].cap -= edges[i].flow;

}

**void** print()

{

printf("Graph:\n");

**for**(**int** i = 0; i < edges.size(); i++)

printf("%d->%d, %d, %d\n", edges[i].from, edges[i].to , edges[i].cap, edges[i].flow);

}

};

ISAP mf;

**int** U;

**int** n, a[N];

**int** deg[N];

**int** source, sink;

**int** check(**double** g)

{

mf.ClearAll(n+2);

memset(deg, 0, sizeof(deg));

**for**(**int** i=1; i<=n; i++)

**for**(**int** j = i+1; j <= n; j++)

**if**(a[j] < a[i]) mf.AddEdge(i, j, 1), mf.AddEdge(j, i, 1), deg[i]++, deg[j]++;

**for**(**int** i=1; i<=n; i++)

mf.AddEdge(source, i, U);

**for**(**int** i=1; i<=n; i++)

mf.AddEdge(i, sink, g\*2.0 + 1.0\*U - 1.0\*deg[i]);

**double** curf = mf.Maxflow(source, sink);

**return** dcmp((U\*n - curf)/2.0);

}

**int** main()

{

Open();

**int** T;

scanf("%d", &T);

**int** cas = 1;

**while**(T--)

{

scanf("%d", &n);

**for**(**int** i=1; i<=n; i++)

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

sink = n+1, source = 0;

U = n+3;

**double** lb = 0, ub = n\*n;

**while**(lb + eps < ub)

{

**double** mid = (lb + ub) / 2;

**int** tmp = check(mid);

**if**(tmp > 0) lb = mid;

**else** ub = mid;

}

printf("Case #%d: %.12f\n", cas++, lb+eps);

}

**return** 0;

}

### 一般匹配-带花树开花算法

#include <cstdio>

#include <cstring>

#include <iostream>

#include <queue>

using namespace std;

**const** **int** N = 250;

// 并查集维护

**int** belong[N];

**int** findb(**int** x) {

**return** belong[x] == x ? x : belong[x] = findb(belong[x]);

}

**void** unit(**int** a, **int** b) {

a = findb(a);

b = findb(b);

**if** (a != b) belong[a] = b;

}

**int** n, match[N];

vector<**int**> e[N];

**int** Q[N], rear;

**int** next\_[N], mark[N], vis[N];

// 朴素算法求某阶段中搜索树上两点x, y的最近公共祖先r

**int** LCA(**int** x, **int** y) {

**static** **int** t = 0; t++;

**while** (**true**) {

**if** (x != -1) {

x = findb(x); // 点要对应到对应的花上去

**if** (vis[x] == t) **return** x;

vis[x] = t;

**if** (match[x] != -1) x = next\_[match[x]];

**else** x = -1;

}

swap(x, y);

}

}

**void** group(**int** a, **int** p) {

**while** (a != p) {

**int** b = match[a], c = next\_[b];

// next数组是用来标记花朵中的路径的，综合match数组来用，实际上形成了

// 双向链表，如(x, y)是匹配的，next[x]和next[y]就可以指两个方向了。

**if** (findb(c) != p) next\_[c] = b;

// 奇环中的点都有机会向环外找到匹配，所以都要标记成S型点加到队列中去，

// 因环内的匹配数已饱和，因此这些点最多只允许匹配成功一个点，在aug中

// 每次匹配到一个点就break终止了当前阶段的搜索，并且下阶段的标记是重

// 新来过的，这样做就是为了保证这一点。

**if** (mark[b] == 2) mark[Q[rear++] = b] = 1;

**if** (mark[c] == 2) mark[Q[rear++] = c] = 1;

unit(a, b); unit(b, c);

a = c;

}

}

// 增广

**void** aug(**int** s) {

**for** (**int** i = 0; i < n; i++) // 每个阶段都要重新标记

next\_[i] = -1, belong[i] = i, mark[i] = 0, vis[i] = -1;

mark[s] = 1;

Q[0] = s; rear = 1;

**for** (**int** front = 0; match[s] == -1 && front < rear; front++) {

**int** x = Q[front]; // 队列Q中的点都是S型的

**for** (**int** i = 0; i < (**int**)e[x].size(); i++) {

**int** y = e[x][i];

**if** (match[x] == y) **continue**; // x与y已匹配，忽略

**if** (findb(x) == findb(y)) **continue**; // x与y同在一朵花，忽略

**if** (mark[y] == 2) **continue**; // y是T型点，忽略

**if** (mark[y] == 1) { // y是S型点，奇环缩点

**int** r = LCA(x, y); // r为从i和j到s的路径上的第一个公共节点

**if** (findb(x) != r) next\_[x] = y; // r和x不在同一个花朵，next标记花朵内路径

**if** (findb(y) != r) next\_[y] = x; // r和y不在同一个花朵，next标记花朵内路径

// 将整个r -- x - y --- r的奇环缩成点，r作为这个环的标记节点，相当于论文中的超级节点

group(x, r); // 缩路径r --- x为点

group(y, r); // 缩路径r --- y为点

}

**else** **if** (match[y] == -1) { // y自由，可以增广，R12规则处理

next\_[y] = x;

**for** (**int** u = y; u != -1; ) { // 交叉链取反

**int** v = next\_[u];

**int** mv = match[v];

match[v] = u, match[u] = v;

u = mv;

}

**break**; // 搜索成功，退出循环将进入下一阶段

}

**else** { // 当前搜索的交叉链+y+match[y]形成新的交叉链，将match[y]加入队列作为待搜节点

next\_[y] = x;

mark[Q[rear++] = match[y]] = 1; // match[y]也是S型的

mark[y] = 2; // y标记成T型

}

}

}

}

bool g[N][N];

**int** main() {

freopen("perfect.in","r",stdin);

freopen("perfect.out","w",stdout);

scanf("%d", &n);

**for** (**int** i = 0; i < n; i++)

**for** (**int** j = 0; j < n; j++) g[i][j] = **false**;

// 建图，双向边

**for**(**int** i=0;i<n;i++){

**int** num;scanf("%d",&num);

**for**(**int** j=0;j<num;j++){

**int** x;scanf("%d",&x);

g[i][x-1]=**true**;

}

}

**for**(**int** i=0;i<n;i++){

**for**(**int** j=0;j<n;j++){

**if**(g[i][j]==**true** && g[j][i]==**true**){

e[i].push\_back(j), e[j].push\_back(i);

}**else**{

g[i][j]=g[j][i]=**false**;

}

}

}

// int x, y; while (scanf("%d%d", &x, &y) != EOF) {

// x--, y--;

// if (x != y && !g[x][y])

// e[x].push\_back(y), e[y].push\_back(x);

// g[x][y] = g[y][x] = true;

// }

// 增广匹配

**for** (**int** i = 0; i < n; i++) match[i] = -1;

**for** (**int** i = 0; i < n; i++) **if** (match[i] == -1) aug(i);

// 输出答案

**int** tot = 0;

**for** (**int** i = 0; i < n; i++) **if** (match[i] != -1) tot++;

// printf("%d\n", tot);

// for (int i = 0; i < n; i++) if (match[i] > i)

// printf("%d %d\n", i + 1, match[i] + 1);

**if**(tot==n){

puts("YES");

}**else**{

puts("NO");

}

**return** 0;

}

### 匈牙利算法

//////////////最佳完美匹配KM算法//////////////////

//下标从1开始。设置w,lx,ly,slack的类型。所有变量无需清零，仅w需要重新赋值

//有slack数组的O(n^3) 版本

**int** n;//点数

**double** w[MAXN][MAXN];//边权

**double** lx[MAXN],ly[MAXN],slack[MAXN];//顶标

**int** s[MAXN],t[MAXN],left\_[MAXN];

**int** match(**int** i)

{

s[i] = 1;

**for**(**int** j = 1;j <= n;j++)

**if**(!t[j])

{

**double** tmp=lx[i]+ly[j]-w[i][j];

**if**( fabs(tmp) < EPS)

{

t[j] = 1;

**if**(!left\_[j] || match(left\_[j]))

{

left\_[j] = i;

**return** 1;

}

}

**else** slack[j] = min(tmp,slack[j]);

}

**return** 0;

}

**void** update()

{

**double** a = INF;

**for**(**int** i = 1;i <= n;i++)

**if**(!t[i]) a = min(a,slack[i]);

**for**(**int** i = 1;i <= n;i++)

{

**if**(s[i]) lx[i] -= a;

**if**(t[i]) ly[i] += a;

}

}

**void** km()

{

**for**(**int** i = 1;i <= n;i++)

{

left\_[i] = ly[i] = 0;

lx[i] = -INF;

**for**(**int** j = 1;j <= n;j++)

lx[i] = max(lx[i],w[i][j]);

}

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

**for**(**int** j = 1;j <= n;j++)

slack[j] = INF;

**while**(1)

{

**for**(**int** j = 1;j <= n;j++) s[j] = t[j] = 0;

**if**(match(i)) **break**;

**else** update();

}

}

}

/\*how to use

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

for(int j = 1;j <= n;j++)

w[i][j] = -cal\_dis(i,j);//赋权

km();

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

printf("%d\n",left\_[i]);

\*/

//////////////最佳完美匹配KM算法//////////////////

### 全局最小割-HDU 3691

#include <cstdio>

#include <algorithm>

#include <cstring>

#include <iostream>

using namespace std;

typedef **long** **long** LL;

**const** **int** MAXN = 310;

LL mat[MAXN][MAXN];

LL weight[MAXN];

bool del[MAXN], vis[MAXN];;

**int** n, m, st;

**void** init() {

memset(mat, 0, sizeof(mat));

memset(del, 0, sizeof(del));

}

LL StoerWagner(**int** &s, **int** &t, **int** cnt) {

memset(weight, 0, sizeof(weight));

memset(vis, 0, sizeof(vis));

**for**(**int** i = 1; i <= n; ++i)

**if**(!del[i]) {t = i; **break**; }

**while**(--cnt) {

vis[s = t] = **true**;

**for**(**int** i = 1; i <= n; ++i) **if**(!del[i] && !vis[i]) {

weight[i] += mat[s][i];

}

t = 0;

**for**(**int** i = 1; i <= n; ++i) **if**(!del[i] && !vis[i]) {

**if**(weight[i] >= weight[t]) t = i;

}

}

**return** weight[t];

}

**void** merge(**int** s, **int** t) {

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

mat[s][i] += mat[t][i];

mat[i][s] += mat[i][t];

}

del[t] = **true**;

}

LL solve() {

LL ret = -1;

**int** s, t;

**for**(**int** i = n; i > 1; --i) {

**if**(ret == -1) ret = StoerWagner(s, t, i);

**else** ret = min(ret, StoerWagner(s, t, i));

merge(s, t);

}

**return** ret;

}

**int** main() {

**while**(scanf("%d%d%d", &n, &m, &st) != EOF) {

**if**(n == 0 && m == 0 && st == 0) **break**;

init();

**while**(m--) {

**int** x, y, z;

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

mat[x][y] += z;

mat[y][x] += z;

}

cout<<solve()<<endl;

}

}

### 判断是否为二分图(是否有奇圈)

**int** color[N];

//判断编号为b的双联通分量是否为二分图

bool bipartite(**int** u)

{

**for**(**int** i=0;i<G[u].size();i++)

{

**int** v = G[u][i];

**if**(color[v]== color[u]) **return** **false**;

**if**(!color[v]){

color[v]=3-color[u];

**if**(!bipartite(v,b)) **return** **false**;

}

}

**return** **true**;

}

### 欧拉回路

#include <iostream>

#include <algorithm>

#include <cstdio>

#include <cstring>

#define N 55

using namespace std;

**int** du[N];

**int** par[N];

**int** eNum[N][N];

**int** Find(**int** x)

{

**return** par[x]==x?x:(par[x]=Find(par[x]));

}

bool connectable(**int** u)

{

**for**(**int** i=1;i<N;i++)

**if**(du[i] && (Find(i)!=Find(u) || (du[i] & 1)))

**return** **false**;

**return** **true**;

}

**int** dfs(**int** u)

{

**for**(**int** i=1;i<N;i++)

{

**if**(eNum[u][i])

{

eNum[i][u]--;

eNum[u][i]--;

dfs(i);

printf("%d %d\n",i,u);

}

}

}

**int** main()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

#endif // ONLINE\_JUDGE

**int** t,cas=1;

scanf("%d",&t);

**while**(t--)

{

printf("Case #%d\n",cas++);

memset(du,0,sizeof(du));

memset(eNum,0,sizeof(eNum));

**for**(**int** i=1;i<N;i++)

par[i]=i;

**int** n;

**int** V=0;

scanf("%d",&n);

**int** pre;

**for**(**int** i=0;i<n;i++)

{

**int** u,v;

scanf("%d%d",&u,&v);

pre=u;

du[u]++,du[v]++;

eNum[u][v]++;

eNum[v][u]++;

**if**(Find(u)!=Find(v)) //判断联通性

par[Find(u)]=Find(v);

}

**if**(!connectable(pre))//判断联通性！

{

printf("some beads may be lost\n");

**if**(t) printf("\n");

**continue**;

}

dfs(pre);

**if**(t) printf("\n");

}

**return** 0;

}

### 计算双联通分量

//如今的版本：

/\*

此版本可处理重边。

st: 某节点的dfs序

bccno: 某条边属于的分量编号

Tn: time\_stamp

b\_cnt: 分量编号标记

G: 图

iscut：标记某个点是否割点

\*/

**int** st[N], bccno[N\*10], Tn, b\_cnt;

vector<PII > G[N];

stack<**int**> S;

**int** iscut[N];

**int** dfs(**int** u, **int** fa)

{

**int** lowu = st[u] = ++Tn;

**int** child = 0;

bool flag = **true**;

**for**(**int** i = 0; i < G[u].size(); i++)

{

**int** v = G[u][i].first;

**int** No = G[u][i].second;

**if**(!st[v]){

S.push(No);

child++;

**int** lowv = dfs(v, u);

lowu = min(lowu, lowv);

**if**(lowv >= st[u]){

iscut[u] = 1;

b\_cnt++;

**while**(**true**)

{

**int** curNo = S.top();S.pop();

bccno[curNo] = b\_cnt;

**if**(curNo == No) **break**;

}

}

}**else** **if**(st[v] < st[u]){

**if**(v == fa && flag) {flag = **false**; **continue**;}

S.push(No);

lowu = min(lowu, st[v]);

}

}

**if**(fa < 0 && child == 1) iscut[u] = 0;

**return** lowu;

}

**void** find\_bcc(**int** n)

{

memset(st, 0, sizeof(st));

memset(iscut, 0, sizeof(iscut));

memset(bccno, 0, sizeof(bccno));

Tn = b\_cnt = 0;

**for**(**int** i = 1; i <= n; i++)

**if**(!st[i]) dfs(i, -1);

}

typedef pair<**int**,**int**> edge;

**int** pre[N],iscut[N],bccno[N],dfs\_clock,bcc\_cnt;

vector<**int**> G[N],bcc[N];

stack<edge> S;

/\*

点-双联通

G[],上述数组中唯一需要初始化的数组

dfs\_clock 时钟标记，记录访问路径中这个点的访问下标

bcc\_cnt 双联通分量的数量

bccno[] 数组表示每个点属于的双联通分量，有的点可能属于多个分量

iscut[] 标记该点是否为割顶

pre[] 每个点的时钟标记

bcc[] 每个双联通分量中有哪些点

\*/

**int** dfs(**int** u,**int** fa)

{

**int** lowu=pre[u] = ++dfs\_clock;

**int** child=0;

**for**(**int** i=0;i<G[u].size();i++)

{

**int** v=G[u][i];

edge e=edge(u,v);

**if**(!pre[v])

{

S.push(e);

child++;

**int** lowv=dfs(v,u);

lowu=min(lowu,lowv); //用后代的low函数更新u的low函数

**if**(lowv>=pre[u])

{

iscut[u]=**true**;

bcc\_cnt++;bcc[bcc\_cnt].clear(); // bcc从1开始编号

**for**(;;)

{

edge x = S.top();S.pop();

**if**(bccno[x.first]!=bcc\_cnt){

bcc[bcc\_cnt].push\_back(x.first);

bccno[x.first]=bcc\_cnt;

}

**if**(bccno[x.second]!=bcc\_cnt){

bcc[bcc\_cnt].push\_back(x.second);

bccno[x.second]=bcc\_cnt;

}

**if**(x.first == u && x.second == v) **break**;

}

}

}**else** **if**(pre[v] < pre[u] && v != fa){

//用反向边更新u的low函数

S.push(e);

lowu=min(lowu,pre[v]);

}

}

**if**(fa < 0 && child ==1) iscut[u]=0;

**return** lowu;

}

//函数顶点编号从0~n-1

**void** find\_bcc(**int** n)

{

//调用结束之后S都为空，不需要清空

memset(pre,0,sizeof(pre));

memset(iscut,0,sizeof(iscut));

memset(bccno,0,sizeof(bccno));

dfs\_clock=bcc\_cnt=0;

**for**(**int** i=0;i<n;i++)//遍历每个点

**if**(!pre[i]) dfs(i,-1);

}

### 二分图最佳完美匹配(KM匈牙利算法)

//二分最佳完美匹配//

/\*下标从1开始，需要对nx，ny赋值，并不需要初始化其中的任何数组\*/

/\*如果求最大匹配，那么权值不变，如果求最小匹配，那么权值变为负数。\*/

//const int N = 101;

**const** **int** INF = 0x3f3f3f3f;

//const double EPS = 1e-9;

typedef **int** w\_type; /////权值的类型，int or double

w\_type w[N][N],lx[N],ly[N],slack[N]; //顶标,权值矩阵

**int** linky[N],linkx[N]; //二分匹配的对象

bool visx[N],visy[N]; //访问数组

**int** nx,ny; //x集合的大小nx，y集合的大小ny

bool find(**int** x)

{

visx[x] = **true**;

**for**(**int** y = 1; y <= ny; y++)

{

**if**(visy[y]) **continue**;

w\_type t = lx[x] + ly[y] - w[x][y];

//if(fabs(t) < EPS) //(double)t==0

**if**( t==0 ) //(int)t==0

{

visy[y] = **true**;

**if**(linky[y]==-1 || find(linky[y]))

{

linky[y] = x;

linkx[x] = y;

**return** **true**; //找到增广轨

}

}

**else** slack[y] = min(t,slack[y]);

}

**return** **false**; //没有找到增广轨（说明顶点x没有对应的匹配，与完备匹配(相等子图的完备匹配)不符）

}

w\_type KM() //返回最优匹配的值

{

**int** i,j;

memset(linky,-1,sizeof(linky));

memset(linkx,-1,sizeof(linkx));

memset(ly,0,sizeof(ly));

**for**(i = 1; i <= nx; i++)

**for**(j = 1,lx[i] = -INF; j <= ny; j++)

lx[i]=max(w[i][j],lx[i]);

/\* if(w[i][j] > lx[i])

lx[i] = w[i][j]; \*/

**for**(**int** x = 1; x <= nx; x++)

{

**for**(i = 1; i <= ny; i++) slack[i] = INF;

**while**(**true**)

{

memset(visx,0,sizeof(visx));

memset(visy,0,sizeof(visy));

**if**(find(x)) **break**; //找到增广轨，退出

w\_type d = INF;

**for**(i = 1; i <= ny; i++) //没找到，对l做调整(这会增加相等子图的边)，重新找

**if**(!visy[i]) d = min(d,slack[i]);

**for**(i = 1; i <= nx; i++)

**if**(visx[i]) lx[i] -= d;

**for**(i = 1; i <= ny; i++)

**if**(visy[i]) ly[i] += d;

}

}

//以下为判断是否完全匹配的代码。

//如下，-INF为权值矩阵的初始值。返回值可根据情况修改如下图，-INF为权值矩阵的初始值。

//返回值可根据情况修改，一般情况可删除。

// for(int i = 1; i <= ny; i++)

// if(w[linky[i]][i] == -INF)

// return 1;

w\_type result = 0;

**for**(i = 1; i <= ny; i++)

**if**(linky[i]>-1)

result += w[linky[i]][i];

**return** result;

}

/\*

\* //如果想要检查是否完全匹配，则需要遍历linkx或linky数组。

\* for(int i=1;i<=nx;i++)

\* for(int j=1;j<=ny;j++)

\* {

\* w[i][j] = -x[i].Distance(y[j]);

\* }

\* KM();

\* for(int i=1;i<=nx;i++)

\* {

\* printf("%d\n",linkx[i]);

\* }

\*/

### 点双联通缩点

**const** **int** MAXN = 100005;

**const** **int** MAXE = 200300;

struct Edge{

**int** from,to,next;

bool cut;

}edge[2\*MAXE];

**int** head[MAXN],edgenum;

**int** Low[MAXN],DFN[MAXN],Stack[MAXN];//Belong数组的值是1~block

**int** dfn,top;

**int** Belong[MAXN],block;//新图的连通块标号（1~block）

bool Instack[MAXN];

**int** bridge; //割桥数量

**void** addedge(**int** u,**int** v){

Edge E={u,v,head[u],0}; edge[edgenum]=E; head[u] = edgenum++;

Edge E2={v,u,head[v],0};edge[edgenum]=E2;head[v] = edgenum++;

}

**void** Tarjan(**int** u,**int** pre){

**int** v;

Low[u] = DFN[u] = ++dfn;

Stack[top++] = u;

Instack[u] = **true**;

**for**(**int** i = head[u]; ~i ;i = edge[i].next){

v = edge[i].to;

// 如果重边有效的话下面这句改成: if(v == pre && pre\_num == 0){pre\_num++;continue;} pre\_num在for上面定义 int pre\_num=0;

**if**( v == pre )**continue**;

**if**( !DFN[v] ){

Tarjan(v,u);

Low[u] = min(Low[u], Low[v]);

**if**(Low[v] > DFN[u]){

bridge++;

edge[i].cut = **true**;

edge[i^1].cut = **true**;

}

}

**else** **if**(Instack[v])Low[u] = min(Low[u], DFN[v]);

}

**if**(Low[u] == DFN[u]){

block++;

**do**{

v = Stack[--top];

Instack[v] = **false**;

Belong[v] = block;

}**while**( v != u );

}

}

**void** work(**int** l, **int** r){

memset(DFN,0,sizeof(DFN));

memset(Instack,**false**,sizeof(Instack));

dfn = top = block = bridge = 0;

**for**(**int** i = l; i <= r; i++)**if**(!DFN[i])Tarjan(i,i);

}

vector<**int**>G[MAXN];//点标从1-block

**void** suodian(){

**for**(**int** i = 1; i <= block; i++)G[i].clear();

**for**(**int** i = 0; i < edgenum; i+=2){

**int** u = Belong[edge[i].from], v = Belong[edge[i].to];

**if**(u==v)**continue**;

G[u].push\_back(v), G[v].push\_back(u);

}

}

**void** init(){edgenum = 0; memset(head,-1,sizeof(head));}

### 并查集

//以下是并查集的函数

**void** init(**int** n)

{

**for**(**int** i=0;i<n;i++)

{

par[i]=i;

rank[i]=0;

}

}

**int** find(**int** x)

{

**if**(par[x]==x)

**return** x;

**else**

**return** par[x]=find(par[x]);

}

**void** unite(**int** x,**int** y)

{

x=find(x);

y=find(y);

**if**(x==y) **return** ;

**if**(rank[x]<rank[y])

par[x]=y;

**else**

{

par[y]=x;

**if**(rank[x]==rank[y]) rank[x]++;

}

}

bool same(**int** x,**int** y)

{

**return** find(x)==find(y);

}

//并查集

### SPOJ QTREE5 树分治+优先队列

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 200010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int** > PII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

struct Reader{

**static** **const** **int** MSIZE = 1000 \* 8 \* 1024;

**char** buf[MSIZE], \*pt = buf, \*o = buf;

**void** init(){

fread(buf, 1, MSIZE, stdin);

}

**char** getch()

{

**char** ch;

**while**((\*pt < 'A' || \*pt > 'Z') && (\*pt < 'a' || \*pt > 'z')) pt++;

ch = \*pt;pt++;

**return** ch;

}

**int** getint()

{

**int** f = 1, x = 0;

**while**(\*pt != '-' && !isdigit(\*pt)) pt++;

**if**(\*pt == '-') f = -1, pt++;

**else** x = \*pt++ - 48;

**while**(isdigit(\*pt)) x = x \* 10 + \*pt++ - 48;

**return** x \* f;

}

}frd;

struct edge{

**int** to, nxt;

edge(){}

edge(**int** \_t, **int** \_n)

{

to = \_t, nxt = \_n;

}

}e[N\*2];

**int** head[N], eN = 0;

**void** addedge(**int** u, **int** v)

{

e[eN] = edge(v, head[u]);

head[u] = eN++;

}

priority\_queue<PII, vector<PII>, greater<PII > > que[N];

vector<PII > wV[N];

**int** root, s[N], f[N];

**int** col[N];

bool vis[N];

**int** n;

**int** maxn;

**int** getroot(**int** now, **int** fa, **int** sz)

{

**int** cnt=1;

**int** mx=0;

**for**(**int** i=head[now]; i!=-1; i=e[i].nxt)

{

**int** to=e[i].to;

**if**(to==fa || vis[to]) **continue**;

f[to]=getroot(to,now,sz);

mx = max(mx,f[to]);

cnt+=f[to];

}

mx = max(mx,sz-cnt);

**if**(mx<maxn)

{

maxn=mx, root=now;

**for**(**int** i = head[now]; i != -1; i = e[i].nxt)

{

**int** to = e[i].to;

**if**(vis[to]) **continue**;

**if**(to == fa)

{

s[to] = sz - cnt;

**continue**;

}

s[to] = f[to];

}

}

**return** cnt;

}

**void** dfsgao(**int** u, **int** fa, **int** rt, **int** dis)

{

wV[u].push\_back(PII(dis, rt));

**for**(**int** i = head[u]; ~i ; i = e[i].nxt)

{

**int** v = e[i].to;

**if**(vis[v] || v == fa) **continue**;

dfsgao(v, u, rt, dis+1);

}

}

**void** dfs(**int** u)

{

maxn = INF;

getroot(u, 0, s[u]);

**int** trt = root;

vis[trt] = 1;

dfsgao(trt, 0, trt, 0);

**for**(**int** i = head[trt]; ~i ; i = e[i].nxt)

{

**int** v = e[i].to;

**if**(vis[v]) **continue**;

dfs(v);

}

}

bool pvvis[N];

**int** main()

{

// Open();

frd.init();

memset(head, -1, sizeof(head));

n = frd.getint();

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

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

**int** x, y;

x = frd.getint();

y = frd.getint();

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

addedge(x, y);

addedge(y, x);

}

s[1] = n;

dfs(1);

**int** q;

q = frd.getint();

**int** ndnum = 0;

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

**while**(q--){

**int** op, u;

op = frd.getint();

u = frd.getint();

// scanf("%d%d", &op, &u);

**if**(op == 0){

col[u] ^= 1;

**if**(col[u]) ndnum++;

**else** ndnum--;

**if**(col[u]){

//pvvis[u] = 1;

**for**(**int** i = 0; i < wV[u].size(); i ++)

{

**int** v = wV[u][i].second;

**int** dis = wV[u][i].first;

que[v].push(PII(dis, u));

}

}

}**else**{

**if**(ndnum == 0){

puts("-1");

**continue**;

}

**int** ans = INF;

**for**(**int** i = 0; i < wV[u].size(); i ++){

**int** v = wV[u][i].second;

**int** dis = wV[u][i].first;

**while**(!que[v].empty()){

PII pp = que[v].top();

**int** curv = pp.second, dist = pp.first;

**if**(!col[curv]) {que[v].pop(); **continue**;}

ans = min(ans, dist+dis);

**break**;

}

}

**if**(ans == INF) ans = -1;

**if**(ans == -1) **while**(1);

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

}

}

**return** 0;

}

### SPFA

#include <iostream>

#include <cstdio>

#include <cmath>

#include <cstring>

#include <queue>

#define N 400100

using namespace std;

struct edge

{

**int** from,to,c,nxt;

}e[N];

**int** head[N];

**int** d[N];

**int** s;

bool vis[N];

**int** n,m;

**void** spfa(**int** s)

{

queue<**int**> q;

memset(d,0x3f,sizeof(d));

d[s]=0;

memset(vis,0,sizeof(vis));

q.push(s);

vis[s]=1;

**while**(!q.empty())

{

**int** x=q.front();

q.pop();

vis[x]=0;

**for**(**int** k=head[x];k!=-1;k=e[k].nxt)

{

**if**(d[e[k].to]>d[e[k].from]+e[k].c)

{

d[e[k].to]=d[e[k].from]+e[k].c;

**if**(!vis[e[k].to])

{

vis[e[k].to]=1;

q.push(e[k].to);

}

}

}

}

}

//DFS版SPFA(判断正/负环非常快)伪代码

Void SPFA(Node) {

Instack[Node]=**true**;

For (Node,v) ∈E

If dis[Node]+edge(Node,v)<dis[v] then {

dis[v]=dis[Node]+edge(Node,v);

If not Instack[v] then

SPFA(v);

Else{

Contain an negative cycle.

Halt;

}

}

Instack [Node] =**false**;

}

### 2-SAT

/\*模型一：两者（A，B）不能同时取

　　那么选择了A就只能选择B’，选择了B就只能选择A’

　　连边A→B’，B→A’

模型二：两者（A，B）不能同时不取

　　那么选择了A’就只能选择B，选择了B’就只能选择A

　　连边A’→B，B’→A

模型三：两者（A，B）要么都取，要么都不取

　　那么选择了A，就只能选择B，选择了B就只能选择A，选择了A’就只能选择B’，选择了B’就只能选择A’

　　连边A→B，B→A，A’→B’，B’→A’

模型四：两者（A，A’）必取A

　　那么，那么，该怎么说呢？先说连边吧。

　　连边A’→A

AND 结果为1：建边 ~x->x,~y->y （两个数必须全为1）

AND 结果为0：建边 y->~x,x->~y （两个数至少有一个为0）

OR 结果为1：建边 ~x->y,~y->x （两个数至少有一个为1）

OR 结果为0：建边 x->~x,y->~y （两个数必须全为0）

XOR 结果为1：建边 x->~y,y->~x,~y->x,~x->y （两个数必须不同）

XOR 结果为0：建边 x->y,y->x,~x->~y,~y->~x （两个数必须相同）\*/

////vector邻接表

/\*

hints:

·构造一组由‘且’链接起来的二元布尔式子。对每一个式子建边。

·需要初始化的有g[],rg[],

·每次需要对V重新赋值

·建边过程中只需要利用冲突的条件来建边，而满足的情况不需要在意。（有待确定）

·（如：x与y冲突，则构造出(x且y=0)来建边）

\*/

**int** n;

**int** V;

vector<**int**> g[N];

vector<**int**> rg[N];

vector<**int**> vs;

bool used[N];

**int** cmp[N];

//0~n-1 true,n~2n-1 false

**void** init(**int** n)

{

**for**(**int** i=0;i<n\*2;i++)

g[i].clear(),rg[i].clear();

}

**void** add\_edge(**int** from,**int** to)

{

g[from].push\_back(to);

rg[to].push\_back(from);

}

**void** dfs(**int** v)

{

used[v]=1;

**for**(**int** i=0;i<g[v].size();i++)

**if**(!used[g[v][i]]) dfs(g[v][i]);

vs.push\_back(v);

}

**void** rdfs(**int** v,**int** k)

{

used[v]=1;

cmp[v]=k;

**for**(**int** i=0;i<rg[v].size();i++)

**if**(!used[rg[v][i]]) rdfs(rg[v][i],k);

}

**int** scc()

{

memset(used,0,sizeof(used));

vs.clear();

**for**(**int** v=0;v<V;v++)

**if**(!used[v]) dfs(v);

memset(used,0,sizeof(used));

**int** k=0;

**for**(**int** i=vs.size()-1;i>=0;i--)

**if**(!used[vs[i]])

rdfs(vs[i],k++);

**return** k;

}

**int** main()

{

/\*add\_edge\*/

V=n\*2;

scc();

bool flag=**true**;

**for**(**int** i=0;i<n && flag;i++)

**if**(cmp[i] == cmp[i+n])

flag=**false**;

**else**

ans.push\_back(Output[age[i]>=x][cmp[i]>cmp[i+n]]);//cmp[i]>cmp[i+n]说明i项应该为true；

}

//vector邻接表

////////////

//用之前需要用init函数

//////////

struct Twosat{

**int** n;

vector<**int**> G[N\*2];

bool mark[N\*2];

**int** S[N\*2],c;

**void** init(**int** n)

{

**this**->n=n;

**for**(**int** i=0;i<n\*2;i++)

G[i].clear();

memset(mark,0,sizeof(mark));

}

bool dfs(**int** x)

{

**if**(mark[x^1]) **return** **false**;

**if**(mark[x]) **return** **true**;

mark[x]=**true**;

S[c++]=x;

**for**(**int** i=0;i<G[x].size();i++)

**if**(!dfs(G[x][i])) **return** **false**;

**return** **true**;

}

**void** add\_edge(**int** x,**int** xval,**int** y,**int** yval)

{

x=x\*2+xval;

y=y\*2+yval;

G[x^1].push\_back(y);

G[y^1].push\_back(x);

}

bool solve()

{

**for**(**int** i=0;i<n\*2;i+=2)

{

**if**(!mark[i] && !mark[i+1])

{

c=0;

**if**(!dfs(i))

{

**while**(c>0) mark[S[--c]]=**false**;

**if**(!dfs(i+1)) **return** **false**;

}

}

}

**return** **true**;

}

};

/////

//邻接矩阵

**int** m[MAXN][MAXN];

**int** id[MAXN];

**int** find\_components(**int** n,**int** mat[][MAXN],**int**\* id){

**int** ret=0,a[MAXN],b[MAXN],c[MAXN],d[MAXN],i,j,k,t;

**for** (k=0;k<n;id[k++]=0);

**for** (k=0;k<n;k++)

**if** (!id[k]){

**for** (i=0;i<n;i++)

a[i]=b[i]=c[i]=d[i]=0;

a[k]=b[k]=1;

**for** (t=1;t;)

**for** (t=i=0;i<n;i++){

**if** (a[i]&&!c[i])

**for** (c[i]=t=1,j=0;j<n;j++)

**if** (mat[i][j]&&!a[j])

a[j]=1;

**if** (b[i]&&!d[i])

**for** (d[i]=t=1,j=0;j<n;j++)

**if** (mat[j][i]&&!b[j])

b[j]=1;

}

**for** (ret++,i=0;i<n;i++)

**if** (a[i]&b[i])

id[i]=ret;

}

**return** ret;

}

## 字符串

### AC自动机

//HDU 5442

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 100010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

**int** smallestRepresation(**char** s[], **int** n, bool kind)

{

**for**(**int** tt = 0; tt < n; tt++) s[n + tt] = s[tt];

s[n \* 2] = '\0';

**int** i, j, k ,l;

**for**(i = 0, j = 1; j < n;){

**for**(k = 0; k < n && s[i+k] == s[j+k]; k++);

**if**(k >= n) {

**if**(kind) //求相同字典序的出现的最小坐标

**break**;

i = j;j ++; //求相同字典序的出现的最大坐标..

//不过这里效率并不高，效率比较高的方法还是找出符合串之后用字符串匹配算法求出最大坐标比较好

**continue**;

}

**if**(s[i+k] > s[j+k]) j += k + 1;//这里是最大表示法(即字典序最大， 最小只需要将大于改为小于即可)

**else** {

l = i+k;

i = j;

j = max(l, j) + 1;

}

}

**return** i;

}

**char** s[42222], s1[42222], s2[42222];

**int** main()

{

Open();

**int** T;scanf("%d", &T);

**while**(T--){

**int** n;

scanf("%d", &n);

scanf("%s", s);

strcpy(s1, s);

strcpy(s2, s);

**int** shunidx = smallestRepresation(s1, n, 1);

reverse(s2, s2+n);

**int** liidx = smallestRepresation(s2, n, 0);

liidx = n - liidx - 1;

**int** kind = -1;

**int** cnt = 0;

**for**(**int** i = shunidx, j = liidx; cnt < n; cnt++){

**if**(s[i] != s[j]){

kind = (s[i] < s[j]);

**break**;

}

i++, j--;

i = i % n;

j = (j + n) % n;

}

**if**(kind == -1){

**if**(shunidx <= liidx){

printf("%d 0\n", shunidx+1);

}**else** **if**(shunidx > liidx){

printf("%d 1\n", liidx+1);

}

}**else**{

**if**(kind == 0) printf("%d %d\n", shunidx+1, kind);

**else** printf("%d %d\n", liidx+1, kind);

}

}

**return** 0;

}

### 字符串的最小表示法(环串中的最小字典序)

//HDU 5442

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 100010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

**int** smallestRepresation(**char** s[], **int** n, bool kind)

{

**for**(**int** tt = 0; tt < n; tt++) s[n + tt] = s[tt];

s[n \* 2] = '\0';

**int** i, j, k ,l;

**for**(i = 0, j = 1; j < n;){

**for**(k = 0; k < n && s[i+k] == s[j+k]; k++);

**if**(k >= n) {

**if**(kind) //求相同字典序的出现的最小坐标

**break**;

i = j;j ++; //求相同字典序的出现的最大坐标..

//不过这里效率并不高，效率比较高的方法还是找出符合串之后用字符串匹配算法求出最大坐标比较好

**continue**;

}

**if**(s[i+k] > s[j+k]) j += k + 1;//这里是最大表示法(即字典序最大， 最小只需要将大于改为小于即可)

**else** {

l = i+k;

i = j;

j = max(l, j) + 1;

}

}

**return** i;

}

**char** s[42222], s1[42222], s2[42222];

**int** main()

{

Open();

**int** T;scanf("%d", &T);

**while**(T--){

**int** n;

scanf("%d", &n);

scanf("%s", s);

strcpy(s1, s);

strcpy(s2, s);

**int** shunidx = smallestRepresation(s1, n, 1);

reverse(s2, s2+n);

**int** liidx = smallestRepresation(s2, n, 0);

liidx = n - liidx - 1;

**int** kind = -1;

**int** cnt = 0;

**for**(**int** i = shunidx, j = liidx; cnt < n; cnt++){

**if**(s[i] != s[j]){

kind = (s[i] < s[j]);

**break**;

}

i++, j--;

i = i % n;

j = (j + n) % n;

}

**if**(kind == -1){

**if**(shunidx <= liidx){

printf("%d 0\n", shunidx+1);

}**else** **if**(shunidx > liidx){

printf("%d 1\n", liidx+1);

}

}**else**{

**if**(kind == 0) printf("%d %d\n", shunidx+1, kind);

**else** printf("%d %d\n", liidx+1, kind);

}

}

**return** 0;

}

### 扩展KMP

//求S的所有后缀与T的最长公共前缀（存储在extand中）

**void** GetNext(**const** **char**\* T, **int**\* nxt)

{

**int** len = strlen(T), a = 0;

nxt[0] = len;

**while**(a<len-1 && T[a] == T[a+1]) a++;

nxt[1] = a;

a = 1;

**for**(**int** k=2;k<len;k++){

**int** p = a + nxt[a] - 1, L = nxt[k - a];

**if**(k - 1 + L >= p){

**int** j = max(0, p-k+1);

**while**(k+j<len && T[k+j] == T[j]) j++;

nxt[k] = j;a = k;

}**else** nxt[k] = L;

}

}

**void** GetExtand(**const** **char**\* S, **const** **char**\* T, **int**\* nxt, **int**\* extand)

{

GetNext(T, nxt);

**int** slen = strlen(S), tlen = strlen(T), a = 0;

**int** Minlen = min(slen, tlen);

**while**(a < Minlen && S[a] == T[a]) a++;

extand[0] = a;

a = 0;

**for**(**int** k=1;k<slen;k++){

**int** p = a + extand[a] - 1, L = nxt[k-a];

**if**(k-1+L >= p){

**int** j = max(0, p-k+1);

**while**(k+j < slen && j < tlen && S[k+j] == T[j]) j++;

extand[k] = j; a = k;

}**else** extand[k] = L;

}

}

### 快速hash值计算

//快速hash值计算

//一般修改为unsigned long long会比较准确

inline **void** init\_hash(**char** \*s, unsigned **int** \*h, **int** l)

{

h[0]= 0;

**for**(**int** i = 1; i <= l;++i)

h[i] = h[i-1] \* MAGIC + s[i-1];

base[0] = 1;

**for**(**int** i = 1; i <= l; ++i)

base[i] = base[i-1] \* MAGIC;

}

inline unsigned **int** string\_hash(unsigned \*h, **int** l, **int** r) //[0-base)

{

**return** h[r] - h[l]\* base[r-l];

}

//快速hash值计算

### 可持久化字典树HDU4757

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 100010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("F:/in.txt","r",stdin);

//freopen("F:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

struct node{

**int** go[2];

**int** cnt;

}pool[N\*20];

**int** tot;//

vector<**int**> G[N];//

**int** pa[20][N];//

**int** w[N];//

**int** dep[N];//

**int** n,m;

**int** root[N];//

**int** insert(**int** pre, **int** val)

{

**int** p = ++tot, ret = p;

pool[p] = pool[pre];

**for**(**int** i = 15; i >= 0; i--)

{

**int** tmp = (val>>i)&1;

**int** cur = ++tot;

pool[cur] = pool[pool[p].go[tmp]];

pool[cur].cnt++;

pool[p].go[tmp] = cur;

p = cur;

}

**return** ret;

}

**void** dfs(**int** v, **int** p, **int** d)

{

root[v] = insert(root[max(0, p)], w[v]);

pa[0][v] = p;

dep[v] = d;

**for**(**int** i = 0; i < G[v].size(); i ++)

**if**(G[v][i] != p) dfs(G[v][i], v, d+1);

}

**void** init()

{

tot = 0;

root[0] = 0;

memset(pool, 0, sizeof(pool));

dfs(1, -1, 0);

**for**(**int** k = 0; k + 1 < 20; k++)

**for**(**int** v = 1; v <= n; v++)

**if**(pa[k][v] < 0) pa[k+1][v] = -1;

**else** pa[k+1][v] = pa[k][pa[k][v]];

}

**int** lca(**int** u, **int** v)

{

**if**(dep[u] > dep[v]) swap(u, v);

**for**(**int** k = 0; k < 20; k++)

**if**((dep[v] - dep[u]) >> k & 1)

v = pa[k][v];

**if**(u == v) **return** u;

**for**(**int** k = 19; k >= 0; k--)

**if**(pa[k][u] != pa[k][v])

u = pa[k][u], v = pa[k][v];

**return** pa[0][u];

}

**int** getans(**int** u, **int** v, **int** val)

{

**int** LCA = lca(u, v);

**int** pu = root[u], pv = root[v], pl = root[LCA];

**int** ans = 0;

**for**(**int** i = 15; i >= 0; i--)

{

**int** tmp = (val >> i)&1;

**int** sum = pool[pool[pu].go[!tmp]].cnt + pool[pool[pv].go[!tmp]].cnt - 2 \* pool[pool[pl].go[!tmp]].cnt;

**if**(sum > 0){

pu = pool[pu].go[!tmp];

pv = pool[pv].go[!tmp];

pl = pool[pl].go[!tmp];

ans += 1<<i;

}**else**{

pu = pool[pu].go[tmp];

pv = pool[pv].go[tmp];

pl = pool[pl].go[tmp];

}

}

**return** max(val ^ w[LCA], ans);

}

**int** main()

{

// Open();

**while**(~scanf("%d%d", &n, &m))

{

**for**(**int** i = 1; i <= n; i++)

scanf("%d", &w[i]), G[i].clear();

**for**(**int** i = 1; i < n; i++)

{

**int** u, v;scanf("%d%d", &u, &v);

G[u].push\_back(v);

G[v].push\_back(u);

}

init();

**while**(m--)

{

**int** u, v, z;

scanf("%d%d%d", &u, &v, &z);

printf("%d\n", getans(u, v, z));

}

}

**return** 0;

}

### 回文树

**const** **int** MAXN = 100005 ;

**const** **int** N = 26 ;

struct Palindromic\_Tree {

**int** next[MAXN][N] ;//next指针，next指针和字典树类似，指向的串为当前串两端加上同一个字符构成

**int** fail[MAXN] ;//fail指针，失配后跳转到fail指针指向的节点

**int** cnt[MAXN] ;

**int** num[MAXN] ;

**int** len[MAXN] ;//len[i]表示节点i表示的回文串的长度

**int** S[MAXN] ;//存放添加的字符

**int** last ;//指向上一个字符所在的节点，方便下一次add

**int** n ;//字符数组指针

**int** p ;//节点指针

**int** newnode ( **int** l ) {//新建节点

**for** ( **int** i = 0 ; i < N ; ++ i ) next[p][i] = 0 ;

cnt[p] = 0 ;

num[p] = 0 ;

len[p] = l ;

**return** p ++ ;

}

**void** init () {//初始化

p = 0 ;

newnode ( 0 ) ;

newnode ( -1 ) ;

last = 0 ;

n = 0 ;

S[n] = -1 ;//开头放一个字符集中没有的字符，减少特判

fail[0] = 1 ;

}

**int** get\_fail ( **int** x ) {//和KMP一样，失配后找一个尽量最长的

**while** ( S[n - len[x] - 1] != S[n] ) x = fail[x] ;

**return** x ;

}

**void** add ( **int** c ) {

c -= 'a' ;

S[++ n] = c ;

**int** cur = get\_fail ( last ) ;//通过上一个回文串找这个回文串的匹配位置

**if** ( !next[cur][c] ) {//如果这个回文串没有出现过，说明出现了一个新的本质不同的回文串

**int** now = newnode ( len[cur] + 2 ) ;//新建节点

fail[now] = next[get\_fail ( fail[cur] )][c] ;//和AC自动机一样建立fail指针，以便失配后跳转

next[cur][c] = now ;

num[now] = num[fail[now]] + 1 ;

}

last = next[cur][c] ;

cnt[last] ++ ;

}

**void** count () {

**for** ( **int** i = p - 1 ; i >= 0 ; -- i ) cnt[fail[i]] += cnt[i] ;

//父亲累加儿子的cnt，因为如果fail[v]=u，则u一定是v的子回文串！

}

} ;

### 后缀数组

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

#include <map>

//#include <unordered\_map>

#define N 200010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("F:/in.txt","r",stdin);

//freopen("F:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

**char** s[N], T[N];

**int** sa[N], t[N], t2[N], c[N], n;

**int** rank[N], height[N];

**void** getHeight(**char** \*s, **int** n)

{

**int** i, k = 0;

**for**(i = 0; i < n; i++) rank[sa[i]] = i;

**for**(i = 0; i < n; i++)

{

**if**(k)k--;

**if**(rank[i] == 0) **continue**;

**int** j = sa[rank[i]-1];

**while**(i+k < n && j+k < n && s[i+k] == s[j+k]) k++;

height[rank[i]] = k;

}

}

**void** build\_sa(**char** \*s, **int** n, **int** m)

{

**int** i, \*x = t, \*y = t2;

**for**(i = 0; i < m; i++) c[i] = 0;

**for**(i = 0; i < n; i++) c[x[i] = s[i]]++;

**for**(i = 1; i < m; i++) c[i] += c[i-1];

**for**(i = n-1; i>=0; i--)sa[--c[x[i]]] = i;

**for**(**int** k = 1; k <= n; k <<= 1){

**int** p = 0;

**for**(i = n-k; i < n; i++) y[p++] = i;

**for**(i = 0; i < n; i++) **if**(sa[i] >= k) y[p++] = sa[i]-k;

**for**(i = 0; i < m; i++) c[i] = 0;

**for**(i = 0; i < n; i++) c[x[y[i]]]++;

**for**(i = 1; i < m; i++) c[i] += c[i-1];

**for**(i = n-1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];

swap(x, y);

p = 1; x[sa[0]] = 0;

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

x[sa[i]] = y[sa[i-1]]== y[sa[i]] && y[sa[i-1]+k] == y[sa[i]+k] ? p-1 : p++;

**if**(p >= n) **break**;

m = p;

}

}

**int** main()

{

// Open();

**while**(~scanf("%s%s", s, T))

{

**int** slen = strlen(s), tlen = strlen(T);

s[slen] = 30;

**for**(**int** i = 0; i < tlen; i++)

s[slen + i + 1] = T[i];

**int** sumlen = slen + tlen+1;

s[sumlen++] = 31;

build\_sa(s, sumlen, 128);

getHeight(s, sumlen);

**int** ans = 0;

**for**(**int** i = 1; i < sumlen;i++)

{

**int** x = sa[i], y = sa[i-1];

**if**(x > y) swap(x, y);

**if**(x < slen && y > slen)

ans = max(ans, height[i]);

}

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

}

**return** 0;

}

### Trie树

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <cstring>

#define N 100010

using namespace std;

struct node

{

**long** **long** nxt[2];//存储nxt节点在数组中的下标

}trie[N\*40];

**long** **long** all\_idx,bit[42],num[N],n;

**long** **long** createNode()

{

memset(trie[all\_idx].nxt,-1,sizeof(trie[all\_idx].nxt));//没有访问过的置为-1

**return** all\_idx++;

}

**void** insert\_Node(**long** **long** root,**long** **long** cur)

{

memset(bit,0,sizeof(bit));//存储cur的二进制数

**long** **long** len=0;

**while**(cur)

{

bit[len++]=cur&1;

cur>>=1;

}

**long** **long** idx=root;

//将40位全部存进去

**for**(len=40;len>=0;len--)

{

**long** **long** k=bit[len];

**if**(trie[idx].nxt[k]==-1)

trie[idx].nxt[k]=createNode();

idx=trie[idx].nxt[k];

}

}

**long** **long** running(**long** **long** root,**long** **long** cur)

{

**long** **long** sum=0;

memset(bit,0,sizeof(bit));

**long** **long** len=0;

**while**(cur)

{

bit[len++]=cur&1;

cur>>=1;

}

**long** **long** idx=root;

**for**(len=40;len>=0;len--)

{

**long** **long** k=!bit[len];/////两个数不相同，异或结果为1

**if**(trie[idx].nxt[k]!=-1) {

sum+=(1LL<<len);///////一定要加LL，被坑了好久。。。也是无语了

idx=trie[idx].nxt[k];

}

**else** idx=trie[idx].nxt[!k];//如果不存在这样的数，只能取0，并且从这边走下去。

}

**return** sum;

}

**int** main()

{

scanf("%I64d",&n);

**long** **long** pre=0,post=0;

**long** **long** ans=0;

**for**(**long** **long** i=0;i<n;i++)

scanf("%I64d",num+i),pre^=num[i];

**long** **long** root=createNode();

**for**(**long** **long** i=n-1;i>=0;pre^=num[i],post^=num[i],i--)

{

insert\_Node(root,post);

ans=max(ans,running(root,pre));

}

ans=max(ans,running(root,pre));//判断取0个前缀的时候的值

cout<<ans<<endl;

**return** 0;

}

### KMP算法

//KMP算法

**void** GetNextval(**char**\* p, **int** next[])

{

nxt[0] = -1;

**int** tlen = strlen(T), j=0, k=-1;

**while**(j<tlen)

**if**(k == -1 || T[j] == T[k]) nxt[++j] = ++k;

**else** k = nxt[k];

}

**int** KmpSearch(**char**\* s, **char**\* p)

{

**int** i = 0;

**int** j = 0;

**int** sLen = strlen(s);

**int** pLen = strlen(p);

**while** (i < sLen && j < pLen)

{

//①如果j = -1，或者当前字符匹配成功（即S[i] == P[j]），都令i++，j++

**if** (j == -1 || s[i] == p[j])

{

i++;

j++;

}

**else**

j = next[j];

}

**if** (j == pLen)

**return** i - j;

**else**

**return** -1;

}

//KMP算法

//统计次数：

**void** GetNextval(**int**\* p, **int** n, **int** nxt[])

{

**int** pLen = n;

nxt[0] = -1;

**int** k = -1;

**int** j = 0;

**while** (j < pLen )

{

//p[k]表示前缀，p[j]表示后缀

**if** (k == -1 || p[j] == p[k])

{

++j;

++k;

//较之前next数组求法，改动在下面4行

**if** (p[j] != p[k] || j==pLen)

nxt[j] = k; //之前只有这一行

**else**

//因为不能出现p[j] = p[nxt[j]]，所以当出现时需要继续递归，k = nxt[k] = nxt[nxt[k]]

nxt[j] = nxt[k];

}

**else**

{

k = nxt[k];

}

}

}

**int** KmpSearch(**int**\* s, **int** ns, **int**\* p, **int** ps)

{

**if**(ns < ps) **return** 0;

**int** i = 0;

**int** j = 0;

**int** sLen = ns;

**int** pLen = ps;

**int** cnt=0;

**int** tmp=0;

**while** (i < sLen)

{

//①如果j = -1，或者当前字符匹配成功（即S[i] == P[j]），都令i++，j++

**if** (j == -1 || s[i] == p[j])

{

i++;

j++;

}

**else**

{

//②如果j != -1，且当前字符匹配失败（即S[i] != P[j]），则令 i 不变，j = nxt[j]

//nxt[j]即为j所对应的next值

j = nxt[j];

}

**if** (j == pLen)

{

cnt++;

j = nxt[j];

}

}

**return** cnt;

}

//统计次数(int)

**int** nxt[N];

**char** s[N];

**char** t[N];

**int** vis[N];

**int** sum[N];

**void** GetNextval(**char**\* p, **int** nxt[])

{

**int** pLen = strlen(p);

nxt[0] = -1;

**int** k = -1;

**int** j = 0;

**while** (j < pLen )

{

//p[k]表示前缀，p[j]表示后缀

**if** (k == -1 || p[j] == p[k])

{

++j;

++k;

//较之前next数组求法，改动在下面4行

**if** (p[j] != p[k])

nxt[j] = k; //之前只有这一行

**else**

//因为不能出现p[j] = p[nxt[j]]，所以当出现时需要继续递归，k = nxt[k] = nxt[nxt[k]]

nxt[j] = nxt[k];

}

**else**

{

k = nxt[k];

}

}

}

**void** KmpSearch(**char**\* s, **char**\* p)

{

**int** i = 0;

**int** j = 0;

**int** sLen = strlen(s);

**int** pLen = strlen(p);

**while** (i < sLen )

{

//①如果j = -1，或者当前字符匹配成功（即S[i] == P[j]），都令i++，j++

**if** (j == -1 || s[i] == p[j])

{

i++;

j++;

}

**else**

{

//②如果j != -1，且当前字符匹配失败（即S[i] != P[j]），则令 i 不变，j = nxt[j]

//nxt[j]即为j所对应的next值

j = nxt[j];

}

**if**(j==pLen)

{

vis[i-pLen]=**true**;

j=nxt[j];

}

}

}

//统计数目（char）

## 杂板子

### 三分

**double** Calc(Type a)

{

/\* 根据题目的意思计算 \*/

}

**void** Solve(**void**)

{

**double** Left, Right;

**double** mid, midmid;

**double** mid\_value, midmid\_value;

Left = MIN; Right = MAX;

**while** (Left + EPS < Right)

{

mid = (Left + Right) / 2;

midmid = (mid + Right) / 2;

mid\_area = Calc(mid);

midmid\_area = Calc(midmid);

// 假设求解最大极值.

**if** (mid\_area >= midmid\_area) Right = midmid;

**else** Left = mid;

}

}

### 手写递归栈

**void** dfs(){

stack<pair<**int**,pair<**int**,**int**> > >s;

s.push(mp(1,mp(0,0)));

**while**(!s.empty()){

pair<**int**,pair<**int**,**int**> >now=s.top();s.pop();

**int** u=now.first,pre=now.second.first,i=now.second.second;

**if**(i==0){

**int** t=++depth;

b[++tot]=t;

f[t]=u;

p[u]=tot;

}

**if**(i<edge[u].size()){ //重点是这里

**int** v=edge[u][i].first,w=edge[u][i].second;

s.push(mp(u,mp(pre,i+1)));

**if**(v==pre) **continue**;

dist[v]=dist[u]+w;

s.push(mp(v,mp(u,0)));

}

**else**

b[++tot]=b[p[pre]];

}

}

//大概是一个用bfs求图中各个点的dfs序的代码。

### 数位DP

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 22

using namespace std;

typedef pair<**long** **long**,**long** **long**> PII;

**const** **long** **long** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

**long** **long** dit[N];

**long** **long** dp[N][3][3][3];

**long** **long** dfs(**long** **long** idx,bool ppre,**long** **long** pre,bool limit,**long** **long** edidx)

{

**if**(dp[idx][ppre][pre][limit]!=-1)

**return** dp[idx][ppre][pre][limit];

**if**(idx==edidx+1)

{

**if**(ppre && pre == 1)

{

**return** dp[idx][ppre][pre][limit]=1;

}

**return** dp[idx][ppre][pre][limit]=0;

}

**if**(ppre && pre == 1)

{

**long** **long** cur=10;

**if**(limit){

**int** tmpidx=idx;

cur=dit[edidx-idx];

**while**(++tmpidx<=edidx)

{

cur\*=10;

cur+=dit[edidx-tmpidx];

}

}**else**{

**int** tmpidx=idx;

**while**(++tmpidx<=edidx)

{

cur\*=10;

}

}

**return** dp[idx][ppre][pre][limit]=cur;

}

**long** **long** ret=0;

**for**(**long** **long** i=0;i <= (limit?dit[edidx-idx]:9);i++)

{

**long** **long** flag=0;

**if**(i==4) flag=2;

**if**(i==9) flag=1;

ret+=dfs(idx+1,pre==2,flag,limit && i==dit[edidx-idx],edidx);

//ret+=dfs(idx+1,pre,i,limit && i==dit[edidx-idx],edidx);

}

**return** dp[idx][ppre][pre][limit]=ret;

}

**long** **long** getval(**long** **long** x)

{

**long** **long** ditnum=0;

memset(dp,-1,sizeof dp);

**while**(x)

{

dit[ditnum++]=x%10;

x/=10;

}

**return** dfs(1,0,0,1,ditnum);

}

**int** main()

{

Open();

**long** **long** T;

scanf("%I64d",&T);

**while**(T--)

{

**long** **long** a;

scanf("%I64d",&a);

printf("%I64d\n",getval(a));

}

**return** 0;

}

### 五子棋判断某个位置是否连成5个

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 100010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

#define ID(x, y) ((x)\*m+(y))

#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

typedef pair<PII, **int**> PIII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

**int** g[22][22];

**int** n = 15, m = 15;

bool judge(**int** play , **int** xp, **int** yp){

**int** n0=0 , n1=0 ,n2=0,n3=0,n4=0,n5=0,n6=0,n7=0;

**for**(**int** x=xp , y=yp+1 ; CHECK(x,y) && g[x][y]==play ; n0++ , y++);

**for**(**int** x=xp+1 , y=yp+1 ; CHECK(x,y) && g[x][y]==play ; n1++ ,x++, y++);

**for**(**int** x=xp+1 , y=yp ; CHECK(x,y) && g[x][y]==play ; n2++ , x++);

**for**(**int** x=xp+1 , y=yp-1 ; CHECK(x,y) && g[x][y]==play ; n3++ ,x++, y--);

**for**(**int** x=xp , y=yp-1 ; CHECK(x,y) && g[x][y]==play ; n4++ , y--);

**for**(**int** x=xp-1 , y=yp-1 ; CHECK(x,y) && g[x][y]==play ; n5++ ,x--, y--);

**for**(**int** x=xp-1 , y=yp ; CHECK(x,y) && g[x][y]==play ; n6++ , x--);

**for**(**int** x=xp-1 , y=yp+1 ; CHECK(x,y) && g[x][y]==play ; n7++ , x--,y++);

**int** a0 = n0+n4+1 , a1 = n1+n5+1 , a2=n2+n6+1, a3=n3+n7+1;

**return** a0>=5 || a1>=5 || a2>=5 || a3>=5;

}

### 一些大质数

100000007

200000033

300000007

400000009

500000009

600000007

699999953

799999999

900000053

1000000007

### C++各个类型输入符控制

符号属性  长度属性 基本型 所占位数    取值范围           输入符举例   输出符举例

--          --      **char**     8      -2^7 ~ 2^7-1            %c         %c %d %u

signed      --      **char**     8      -2^7 ~ 2^7-1            %c         %c %d %u

unsigned    --      **char**     8      0 ~ 2^8-1               %c         %c %d %u

signed     **short**    **int**     16     -2^15 ~ 2^15-1          %hd

unsigned  **short**    **int**      16     0 ~ 2^16-1          %hu %ho  %hx

signed      --      **int**      32     -2^31 ~ 2^31-1          %d

unsigned    --      **int**      32     0 ~ 2^32-1           %u %o %x

signed     **long**     **int**      32     -2^31 ~ 2^31-1         %ld

unsigned   **long**     **int**      32     0 ~ 2^32-1         %lu %lo %lx

signed     **long**     **int**      64     -2^63 ~ 2^63-1        %I64d

unsigned   **long**     **int**      64     0 ~ 2^64-1      %I64u %I64o %I64x

 --         --     **float**     32    +/- 3.40282e+038    %f %e %g

--         --     **double**  64    +/- 1.79769e+308   %lf %le %lg     %f %e %g

--        **long**    **double**  96    +/- 1.79769e+308 %Lf、%Le、%Lg

### C++扩栈

#pragma comment(linker, "/STACK:102400000,102400000")

### Cin取消同步

std::ios::sync\_with\_stdio(**false**);

//注意取消同步之后cin,cout不能和scanf,printf混用

### G++扩栈

//亲测HDU可用

//zoj扩栈内存会算到总内存中，需要小心一些

**int** main2() {

//your code

**return** 0;//博客上说这里换成exit(0),不过我在hdu上面提交这个也没有问题//ZOJ上也可用，不过的确得改为exit(0)

}

extern **int** main2(**void**) \_\_asm\_\_ ("\_main2");

**int** main()

{

**int** size = 256 << 20; // 256Mb

**char** \*p = (**char** \*)malloc(size) + size;

\_\_asm\_\_ \_\_volatile\_\_(

"mov %0, %%rsp\n" //这里很多时候会报错“bad register name '%rsp'”此时只需要将rsp换成esp就行了(原理就是两个不同的寄存器，在某些平台上名字不同)

"push $\_exit\n"

"jmp \_main2\n"

:: "r"(p));

**return** 0;

}

//博客中的扩栈代码：

//Win 32位MinGW 4.7.2环境

extern **int** main2(**void**) \_\_asm\_\_ ("\_main2");

**int** main2() {

**char** test[255 << 20];

memset(test, 42, sizeof(test));

printf(":)\n");

exit(0);

}

**int** main() {

**int** size = 256 << 20; // 256Mb

**char** \*p = (**char** \*)malloc(size) + size;

\_\_asm\_\_ \_\_volatile\_\_(

"movl %0, %%esp\n"

"pushl $\_exit\n"

"jmp \_main2\n"

:: "r"(p));

}

//Linux 64位gcc 4.8.1环境

extern **int** main2(**void**) \_\_asm\_\_ ("main2");

**int** main2() {

**char** test[255 << 20];

memset(test, 42, sizeof(test));

printf(":)\n");

exit(0);

}

**int** main() {

**int** size = 256 << 20; // 256Mb

**char** \*p = (**char** \*)malloc(size) + size;

\_\_asm\_\_ \_\_volatile\_\_(

"movq %0, %%rsp\n"

"pushq $exit\n"

"jmp main2\n"

:: "r"(p));

}

### Headfile

#include <iostream>

#include <cstdio>

#include <stack>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

//#include <unordered\_map>

#define N 100010

//#define lson x<<1

//#define rson x<<1|1

//#define mid ((lt[x].l+lt[x].r)/2)

//#define ID(x, y) ((x)\*m+(y))

//#define CHECK(x, y) ((x)>=0 && (x)<n && (y)>=0 && (y)<m)

using namespace std;

typedef **long** **long** LL;

typedef pair<**int**,**int**> PII;

**const** **int** INF=0x3f3f3f3f;

**void** Open()

{

#ifndef ONLINE\_JUDGE

freopen("D:/in.txt","r",stdin);

//freopen("D:/my.txt","w",stdout);

#endif // ONLINE\_JUDGE

}

**int** main()

{

//Open();

**return** 0;

}

### Unordered\_map

struct Node

{

string str;

**int** idx;

bool operator==(**const** Node& o)**const**

{

**return** str==o.str && idx == o.idx;

}

};

struct Node\_hash

{

size\_t operator()(**const** Node& o)**const**

{

**return** hash<string>()(o.str) ^ (hash<**int**>()(o.idx) >> 1);

}

};

unordered\_map<Node,**int**,Node\_hash> vis;

//我们需要重载==符号，以及另外在另一个结构体里面写hash函数，格式如上：

### Unordermap简化版

struct Item{

**int** key,val,nxt;

};

struct UMP{

Item item[555555];

**int** itnum;

**int** head[999997];

**int** MOD;

UMP(){

MOD=999997;

clear();

}

**void** clear(){

memset(head,-1,sizeof(head));

itnum=0;

}

bool find(**int** x)

{

**int** idx=x%MOD;

**for**(**int** i=head[idx]; i!=-1; i=item[i].nxt)

{

**if**(item[i].key==x) **return** item[i].val;

}

**return** -1;//没找到

}

**int**& operator[](**int** x){

**int** idx=x%MOD;

//cerr<<idx<<endl;

**for**(**int** i=head[idx];i!=-1;i=item[i].nxt){

**if**(item[i].key==x) **return** item[i].val;

}

item[itnum]=(Item){x,0,head[idx]};

head[idx]=itnum;

**return** item[itnum++].val;

}

};

### 读入挂

inline **long** **long** Scan() //输入外挂

{

**long** **long** res=0,ch,flag=0;

**if**((ch=getchar())=='-')

flag=1;

**else** **if**(ch>='0'&&ch<='9')

res=ch-'0';

**while**((ch=getchar())>='0'&&ch<='9')

res=res\*10+ch-'0';

**return** flag?-res:res;

}

inline **void** Out(**long** **long** a) //输出外挂

{

**if**(a>9)

Out(a/10);

putchar(a%10+'0');

}

template<**class** T>

inline bool read(T &n){

T x = 0, tmp = 1; **char** c = getchar();

**while** ((c < '0' || c > '9') && c != '-' && c != EOF) c = getchar();

**if** (c == EOF) **return** **false**;

**if** (c == '-') c = getchar(), tmp = -1;

**while** (c >= '0' && c <= '9') x \*= 10, x += (c - '0'), c = getchar();

n = x\*tmp;

**return** **true**;

}

template <**class** T>

inline **void** write(T n) {

**if** (n < 0) {

putchar('-');

n = -n;

}

**int** len = 0, data[20];

**while** (n) {

data[len++] = n % 10;

n /= 10;

}

**if** (!len) data[len++] = 0;

**while** (len--) putchar(data[len] + 48);

}

///////////////////////////////fread加速!!!!!!!!///////////////

**char** \*ch1, buf1[40\*1024000+5];

**char** \*ch, buf[40\*1024000+5];

template <**class** T>

**void** read(T &x) {

**for** (++ch; \*ch <= 32; ++ch);

**for** (x = 0; '0' <= \*ch; ch++) x = x \* 10 + \*ch - '0';

}

**void** out(**int** x) {

**if** (!x) \*(++ch1) = '0';

**else** {

**char** \*ch0 = ch1, \*ch = ch1 + 1;

**while** (x) {

\*(++ch0) = x % 10 + '0';

x /= 10;

}

ch1 = ch0;

**while** (ch <= ch0) swap(\*(ch++), \*(ch0--));

}

\*(++ch1) = '\n';

}

**void** out(**long** **long** x) {

**if** (!x) \*(++ch1) = '0';

**else** {

**char** \*ch0 = ch1, \*ch = ch1 + 1;

**while** (x) {

\*(++ch0) = x % 10 + '0';

x /= 10;

}

ch1 = ch0;

**while** (ch <= ch0) swap(\*(ch++), \*(ch0--));

}

\*(++ch1) = '\n';

}

**int** main(){

Open();//freopen

ch = buf - 1;

ch1 = buf1 - 1;

fread(buf, 1, 1000 \* 35 \* 1024, stdin);

// fwrite(buf1, 1, ch1 - buf1 + 1, stdout);//输出，放在main函数的最后一行

}

//----------------------比较规整的版本，注意内存消耗比较大-------------

struct Reader{

**static** **const** **int** MSIZE = 1000 \* 8 \* 1024;

**char** buf[MSIZE], \*pt = buf, \*o = buf;

**void** init(){

fread(buf, 1, MSIZE, stdin);

}

**char** getch()

{

**char** ch;

**while**((\*pt < 'A' || \*pt > 'Z') && (\*pt < 'a' || \*pt > 'z')) pt++;

ch = \*pt;pt++;

**return** ch;

}

**int** getint()

{

**int** f = 1, x = 0;

**while**(\*pt != '-' && !isdigit(\*pt)) pt++;

**if**(\*pt == '-') f = -1, pt++;

**else** x = \*pt++ - 48;

**while**(isdigit(\*pt)) x = x \* 10 + \*pt++ - 48;

**return** x \* f;

}

}frd;

### 多重背包二进制

**for**(**int** i = 0; i < K; ++i){

**int** num = ev[i].c;

**for**(**int** k = 1; num; k <<= 1){

**int** mul = min(k,num);

**for**(**int** j = ev[i].a ; j >= mul \* ev[i].h; --j)

**if**(dp[j- ev[i].h\*mul]) dp[j] = 1,ans = max(j,ans);

num -= mul;

}

}

**for**(**int** d = 0; d < w[i]; d++) { // 对于所有余数 d [0, w[i])

// 窗口大小为 c[i]

**int** sum = 0, st = 0, ed = -1; //st,ed 单调队列的开始和结尾, sum 队列中是否有一个 true

**for**(**int** v = d; v <= m; v+= w[i]) { // 完全背包 model, 但步长是 w[i]

**if**(ed - st == c[i]) { // 窗口大小为0, 移除队首元素, 队首后移一位

sum -= queue[st++];

}

queue[++ed] = dp[v];

sum += dp[v];

**if**(!dp[v] && sum)

dp[v] = 1;

}

}

### 矩阵快速幂

#include <iostream>

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <cmath>

#include <algorithm>

#define N 6

using namespace std;

**const** **int** mod=1e9+7;

**void** printfm(**int** A[N][N],**int** n,**int** m)

{

**for**(**int** i=0;i<n;i++)

**for**(**int** j=0;j<m;j++)

{

printf("%d%c",A[i][j],(j==m-1)?'\n':' ');

}

printf("\n");

}

**void** mul(**int** A[N][N],**int** B[N][N],**int** t[N][N],**int** n,**int** m,**int** l)//A 为n\*m的矩阵，B为m\*l的矩阵,t为结果矩阵

{

**int** tmp[N][N];//为了防止冲突

**for**(**int** i=0;i<n;i++)

**for**(**int** j=0;j<l;j++){

tmp[i][j]=0;

**for**(**int** k=0;k<m;k++)

tmp[i][j]=(tmp[i][j]+A[i][k]\*B[k][j])%mod;

}

**for**(**int** i=0;i<n;i++) **for**(**int** j=0;j<l;j++) t[i][j]=tmp[i][j];

}

**void** expo(**int** p[N][N],**int** e[N][N],**int** k,**int** n)//P为n\*n的矩阵，k为计算k次幂，e为结果矩阵

{

**for**(**int** i = 0; i < n; ++i) **for**(**int** j = 0; j < n; ++j) e[i][j] = (i == j);

**while**(k) {

**if**(k&1) mul(e,p,e,n,n,n);

mul(p,p,p,n,n,n);

k>>=1;

}

}

**int** a[N][N];

**int** b[N][N];

**int** c[N][N];

**int** main()

{

#ifndef ONLINE\_JUDGE

//freopen("E:/in.txt","r",stdin);

//freopen("E:/my.txt","w",stdout);

#endif

**int** n,a0,ax,ay,b0,bx,by;

**while**(~scanf("%d%d%d%d%d%d%d",&n,&a0,&ax,&ay,&b0,&bx,&by))

{

memset(a,0,sizeof(a));

memset(b,0,sizeof(b));

memset(c,0,sizeof(c));

a[0][0]=0,a[0][1]=a0\*b0%mod,a[0][2]=a0,a[0][3]=b0,a[0][4]=1;

b[0][0]=1,b[1][0]=1,b[1][1]=ax\*bx%mod,b[2][1]=ax\*by%mod,b[2][2]=ax,b[3][1]=bx\*ay%mod;

b[3][3]=bx,b[4][1]=ay\*by%mod,b[4][2]=ay,b[4][3]=by,b[4][4]=1;

//printfm(a,1,5);

//printfm(b,5,5);

expo(b,c,n,5);

mul(a,c,c,1,5,5);

printf("%d\n",c[0][0]);

}

**return** 0;

}

### 离散化

cnt = 0;

**int** len = 1;

**for**(**int** i=1;i<=n;i++)

{

**int** a, b;

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

// read(a);read(b);

**if**(a == 0){

q[i] = node(a, b, b+len);

mp[cnt++] = b, mp[cnt++] = b+len;

idx[len] = i;

len++;

}**else**{

q[i] = node(a, q[idx[b]].l, q[idx[b]].r);

}

}

sort(mp, mp+cnt);

cnt = unique(mp, mp+cnt) - mp;

**for**(**int** i=1;i<=n;i++)

{

q[i].l = lower\_bound(mp, mp+cnt, q[i].l) - mp + 1;

q[i].r = lower\_bound(mp, mp+cnt, q[i].r) - mp + 1;

}

## 计算几何

### 最近点对问题

//最近点对问题

#include <iostream>

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

**const** **double** INF = 1e20;

**const** **int** N = 100005;

struct Point

{

**double** x;

**double** y;

}point[N];

**int** n;

**int** tmpt[N];

bool cmpxy(**const** Point& a, **const** Point& b)

{

**if**(a.x != b.x)

**return** a.x < b.x;

**return** a.y < b.y;

}

bool cmpy(**const** **int**& a, **const** **int**& b)

{

**return** point[a].y < point[b].y;

}

**double** min(**double** a, **double** b)

{

**return** a < b ? a : b;

}

**double** dis(**int** i, **int** j)

{

**return** sqrt((point[i].x-point[j].x)\*(point[i].x-point[j].x)

+ (point[i].y-point[j].y)\*(point[i].y-point[j].y));

}

**double** Closest\_Pair(**int** left, **int** right)

{

**double** d = INF;

**if**(left==right)

**return** d;

**if**(left + 1 == right)

**return** dis(left, right);

**int** mid = (left+right)>>1;

**double** d1 = Closest\_Pair(left,mid);

**double** d2 = Closest\_Pair(mid+1,right);

d = min(d1,d2);

**int** i,j,k=0;

//分离出宽度为d的区间

**for**(i = left; i <= right; i++)

{

**if**(fabs(point[mid].x-point[i].x) <= d)

tmpt[k++] = i;

}

sort(tmpt,tmpt+k,cmpy);

//线性扫描

**for**(i = 0; i < k; i++)

{

**for**(j = i+1; j < k && point[tmpt[j]].y-point[tmpt[i]].y<d; j++)

{

**double** d3 = dis(tmpt[i],tmpt[j]);

**if**(d > d3)

d = d3;

}

}

**return** d;

}

//最近点对问题

### 凸包

/\*

\* 计算凸包，输入点数组为p，个数为n，输出点数组为ch，返回凸包顶点数

\* 输入不能有重复点。函数执行完之后，sort，会破坏顺序

\* 如果不希望在凸包的边上有输入点，把两个 <= 改为 <

\*/

**int** ConvexHull(Point\* p, **int** n, Point\* ch)

{

sort(p, p+n);

**int** m = 0;

**for**(**int** i=0;i<n;i++){

**while**(m > 1 && Cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;

ch[m++] = p[i];

}

**int** k=m;

**for**(**int** i=n-2;i>=0;i--){

**while**(m>k && Cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;

ch[m++] = p[i];

}

**if**(n>1) m--;

**return** m;

}

/\*

\* 用法：

\* for(int i=0;i<n;i++)

\* {

\* double x,y;

\* scanf("%lf%lf",&x,&y);

\* p[i] = (Point){x,y};

\* }

\* int bagNum=ConvexHull(p,n,tubag);

\*/

### 求多边形重心

Point MassCenter(Point a[] , **int** n){

Point ans = Point(0,0);

**double** area = AREA(a, n);

**if**(dcmp(area) == 0) **return** ans;

a[n] = a[0];

**for**(**int** i=0;i<n;i++) ans = ans+(a[i] + a[i+1])\*Cross(a[i+1] , a[i]);

**return** ans /area /6.0;

}

### 计算几何常规模板

struct Point

{

**double** x,y;

Point(**double** x = 0, **double** y = 0):x(x),y(y){}

**void** read()

{

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

}

}pa[N],pb[N];

**const** **double** eps = 1e-10;

**const** **double** PI = acos(-1.0);

typedef Point Vector;

**int** dcmp(**double** x){**if**(fabs(x) < eps) **return** 0;**else** **return** x<0?-1:1;}

Vector operator+(Vector A,Vector B){**return** Vector(A.x+B.x, A.y+B.y);}

Vector operator-(Point A,Point B){**return** Vector(A.x-B.x, A.y-B.y);}

Vector operator\*(Vector A, **double** p){**return** Vector(A.x\*p, A.y\*p);}

Vector operator/(Vector A, **double** p){**return** Vector(A.x/p, A.y/p);}

bool operator<(**const** Point& a, **const** Point& b){**return** a.x<b.x || (a.x == b.x && a.y < b.y);}

bool operator==(**const** Point& a, **const** Point& b){**return** dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0;}

**double** angle(Vector A){**return** atan2(A.y,A.x);}//返回A向量的极角atan2(y,x)所表达的意思是坐标原点为起点，指向(x,y)的射线在坐标平面上与x轴正方向之间的角的角度。

**double** Dot(Vector A, Vector B){**return** A.x\*B.x+A.y\*B.y;}

**double** Length(Vector A){**return** sqrt(Dot(A,A));}

**double** Angle(Vector A,Vector B){**return** acos(Dot(A,B)/Length(A)/Length(B));}//A到B的逆时针转的角

**double** Cross(Vector A, Vector B){**return** A.x\*B.y-A.y\*B.x;}

Vector Rotata(Vector A,**double** rad){**return** Vector(A.x\*cos(rad)-A.y\*sin(rad), A.x\*sin(rad)+A.y\*cos(rad));}//A逆时针转ang弧度

**double** torad(**double** ang){**return** ang / 180 \* PI;}

Vector Normal(Vector A){**double** L = Length(A); **return** Vector(-A.y/L, A.x/L);}//需要确保A不是0向量，左转90度

//非规范相交，端点上视为在线段上

bool OnSegment(Point p, Point a, Point b) { **return** dcmp(Length(p - a) + Length(p - b) - Length(a - b)) == 0; }//精度也很高的！

**double** NormalAng(**double** x)//将弧度x通过+-2\*PI的方式约束到[-PI,PI];

{

**if**(x > 0){

**while**(x > PI) x -= 2.0 \* PI;

}**else**{

**while**(x < -PI) x += 2.0 \* PI;

}

**return** x;

}

struct Line{

Point p,v;

**double** a,b,c;//得到一般式的参数

**double** ang;

Line(){}

Line(Point p = Point(0,0), Vector v = Vector(0,0)):p(p),v(v){a = v.y - p.y; b = p.x - v.x; c = p.y\*v.x - v.y\*p.x;ang = angle(v);}

Point point(**double** t){**return** p + v\*t;}//只能在点斜式中用

bool operator < (**const** Line& L)**const**{

**return** ang < L.ang;

}

}L[N];

struct Circle

{

Point c;

**double** r;

Circle(){}

Circle(Point c, **double** r):c(c),r(r){}

Point point(**double** a)

{

**return** Point(c.x + cos(a)\*r, c.y+sin(a)\*r);

}

};

bool OnSegment(Point p, Point a1, Point a2){**return** dcmp(Cross(a1-p, a2-p)) == 0 && dcmp(Dot(a1-p, a2-p)) < 0;}

bool OnSegment(Point p, Point a, Point b)//精度较高的判断

{

Point v1,v2;

v1=p-a;

v2=p-b;

**if**( dcmp(Cross(v1,v2))!=0) //叉积不为0 就不在直线上

**return** 0;

**else**

{

**if**(dcmp(min(a.x,b.x)- p.x)<=0 && dcmp(p.x-max(a.x,b.x))<=0 && dcmp(min(a.y,b.y)- p.y)<=0 && dcmp( p.y-max(a.y,b.y))<=0)

**return** 1;

**else**

**return** 0;

}

}

//两直线相交

Point GetLineIntersection(Point P,Vector v, Point Q, Vector w)

{

Vector u = P-Q;

**double** t = Cross(w,u)/ Cross(v,w);

**return** P+v\*t;

}

//点到线段距离

**double** DistanceToSegment(Point P, Point A, Point B)

{

**if**(A==B) **return** Length(P-A);

Vector v1 = B - A, v2 = P - A, v3 = P - B;

**if**(dcmp(Dot(v1, v2)) < 0) **return** Length(v2);

**else** **if**(dcmp(Dot(v1, v3)) > 0) **return** Length(Length(v3));

**else** **return** fabs(Cross(v1, v2)) / Length(v1);

}

//点到直线距离

**double** DistanceToLine(Point P, Point A, Point B)

{

Vector v1 = B-A,v2 = P-A;

**return** fabs(Cross(v1,v2)) / Length(v1);

}

//线段是否规范相交

bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2)

{

**double** c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);

**double** c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);

**return** dcmp(c1)\*dcmp(c2)<0 && dcmp(c3)\*dcmp(c4)<0;

}

//过点P的圆C的切线

**int** getTangents(Point P, Circle C, Vector\* v)

{

Vector u = C.c - P;

**double** dist = Length(u);

**if**(dist < C.r) **return** 0;

**if**(dcmp(dist - C.r) == 0) {

v[0] = Rotata(u, PI/2);

**return** 1;

}

**double** ang = asin(C.r/dist);

v[0] = Rotata(u, -ang);

v[1] = Rotata(u, ang);

**return** 2;

}

//圆圆相交

**int** getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol)

{

**double** d = Length(C1.c-C2.c);

**if**(dcmp(d) == 0)

{

**if**(dcmp(C1.r-C2.r) == 0) **return** -1;

**return** 0;

}

**if**(dcmp(C1.r + C2.r - d ) < 0) **return** 0;

**if**(dcmp(fabs(C1.r-C2.r) - d) > 0) **return** 0;

**double** a = angle(C2.c-C1.c);

**double** da = acos((C1.r\*C1.r + d\*d - C2.r\*C2.r) / (2\*C1.r\*d));

Point P1 = C1.point(a - da), P2 = C1.point(a + da);

sol.push\_back(P1);

**if**(P1 == P2) **return** 1;

sol.push\_back(P2);

**return** 2;

}

//直线与圆相交,直线必须是点斜式,如果是线段的话，需要检查t1,t2是否在[0,1]之间。

**int** getLineCircleIntersection(Line L, Circle C, vector<Point >& sol)

{

**double** a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;

**double** e = a\*a + c\*c, f = 2\*(a \* b + c \* d), g = b\*b+d\*d-C.r\*C.r;

**double** delta = f\*f - 4\*e\*g;

**double** dist = DistanceToLine(C.c, L.p, L.p + L.v);

**double** t1,t2;

**if**(dcmp(dist - C.r) > 0) **return** 0;

**if**(dcmp(dist - C.r) == 0) {

t1 = t2 = -f / (2\*e); sol.push\_back(L.point(t1));

**return** 1;

}

t1 = (-f - sqrt(delta)) / (2\*e); sol.push\_back(L.point(t1));

t2 = (-f + sqrt(delta)) / (2\*e); sol.push\_back(L.point(t2));

**return** 2;

}

//计算多边形的有向面积

**double** PolygonArea(Point\* p, **int** n)

{

**double** area = 0;

**for**(**int** i=1;i<n-1;i++) area += Cross(p[i]-p[0], p[i+1]-p[0]);

**return** area/2;

}

//判断点是否在多边形（可以是凹多边形）的内部

**int** isPointInPolygon(Point p, Point\* poly, **int** n)

{

**int** wn = 0;

**for**(**int** i = 0; i<n; i++)

{

**if**(OnSegment(p, poly[i], poly[(i+1)%n])) **return** -1;//边界

**int** k = dcmp(Cross(poly[(i+1)%n]-poly[i], p-poly[i]));

**int** d1 = dcmp(poly[i].y - p.y);

**int** d2 = dcmp(poly[(i+1)%n].y - p.y);

**if**(k>0 && d1 <= 0 && d2 > 0) wn++;

**if**(k<0 && d2 <= 0 && d1 > 0) wn--;

}

**if**(wn != 0) **return** 1;//内部

**return** 0;//外部

}

//点是否在凸多边形内

bool isPointInConvexPolygon(Point p, Point\* poly, **int** n)

{

**for**(**int** i=0;i<n;i++)

**if**(dcmp(Cross(poly[(i+1)%n] - poly[i], p - poly[i])) <= 0) **return** 0;

**return** 1;

}

### 多边形圆交面积

struct CPIArea

{

Circle cir;

**double** Scir;

Point p[MAXN];

**int** tail;

CPIArea()

{

tail=0;

}

CPIArea(Circle cir):cir(cir)

{

Scir = PI\*cir.r\*cir.r;

tail=0;

}

//tp[]是多边形的点集，n是点的个数。tp[]必须满足点是按顺时针或者逆时针排序的

**double** solve(Point tp[],**int** n)

{

tail = 0;

**for**(**int** i=0; i<n; i++)

{

p[tail++]=tp[i];//p[]是囊括了圆和多边形交点的点集，也是按顺时针或者逆时针排序的

Line line = Line(tp[i],tp[(i+1)%n] - tp[i]);

**double** t1,t2;

vector<Point > sol;

sol.clear();

getLineCircleIntersection(line , cir , t1,t2,sol);

**for**(**int** j=0; j<sol.size(); j++)

{

p[tail++]=sol[j];

}

}

**double** res=0;

**for**(**int** i=0; i<tail; i++)

{

Point O = cir.c;

**double** ang = Angle(p[(i+1)%tail]-O , p[i]-O);

**if**( dcmp( Cross( p[i]-O , p[(i+1)%tail]-O)) > 0 ) ang\*=1;

**else** ang\*=-1;

**double** Sshan = ang/(2\*PI)\*Scir;

**double** Strian = Area(O , p[i] ,p[(i+1)%tail] );

**if**(dcmp( abs(Sshan) - abs(Strian))<=0 )

{

res += Sshan;

}

**else** res += Strian;

}

**return** abs(res);

}

};

### 点集的直径

**int** diameter2(Point\* points, **int** n, Point\* p)

{

n = ConvexHull(points, n, p);

**if**(n == 1) **return** 0;

**if**(n == 2) **return** Dot(p[1] - p[0], p[1] - p[0]);

p[n] = p[0];

**int** ans = 0;

**for**(**int** u = 0, v = 1; u<n; u++)

// 一条直线贴住边p[u]-p[u+1]

**while**(**true**)

{

// 当Area(p[u], p[u+1], p[v+1]) <= Area(p[u], p[u+1], p[v])时停止旋转

// 即Cross(p[u+1]-p[u], p[v+1]-p[u]) - Cross(p[u+1]-p[u], p[v]-p[u]) <= 0

// 根据Cross(A,B) - Cross(A,C) = Cross(A,B-C)

// 化简得Cross(p[u+1]-p[u], p[v+1]-p[v]) <= 0

**int** diff = Cross(p[u+1]-p[u], p[v+1]-p[v]);

**if**(diff <= 0)

{

ans = max(ans, (**int**)(Dot(p[u]-p[v], p[u]-p[v])+eps));//u和v是对踵点

**if**(diff == 0) ans = max(ans, (**int**)(Dot(p[u]-p[v+1], p[u]-p[v+1])+eps));// diff == 0时u和v+1也是对踵点

**break**;

}

v = (v+1)%n;

}

**return** ans;

}

### 半平面交

struct line

{

Point P;

Vector v;

**double** ang;

line(){}

line(Point P, Vector v):P(P), v(v){ang = atan2(v.y, v.x);}//点和向量

line(**double** a, **double** b, **double** c)//从一般式转化过来

{

v = Vector(b, -a);

**if**(b != 0) P = Point(0, -c/b);

**else** P = Point(-c/a, 0);

Vector nor = Normal(v);

Point tmp = nor + P;

**if**(dcmp(a \* tmp.x + b \* tmp.y + c) > 0){//这里保证是ax+by+c <= 0的半平面

v = v\*-1;

}

ang = atan2(v.y, v.x);

}

bool operator<(**const** line &o)**const**{

**return** ang < o.ang;

}

}L[111];

bool OnLeft(line L, Point P)

{

**return** Cross(L.v, P - L.P) > 0;

}

Point GetIntersection(line a, line b)

{

Vector u = a.P - b.P;

**double** t = Cross(b.v, u) / Cross(a.v, b.v);

**return** a.P + a.v\*t;

}

//函数过后， L的顺序会改变。返回半平面交的凸包的节点数。

LL HalfPlaneIntersection(line\* L, LL n, Point\* poly)

{

sort(L, L+n);

LL first, last;

Point \*p = **new** Point[n];

line \*q = **new** line[n];

q[first = last = 0] = L[0];

**for**(LL i = 1; i < n ;i++)

{

**while**(first < last && !OnLeft(L[i], p[last - 1])) last--;

**while**(first < last && !OnLeft(L[i], p[first])) first++;

q[++last] = L[i];

**if**(fabs(Cross(q[last].v, q[last-1].v)) < eps){

last--;

**if**(OnLeft(q[last], L[i].P)) q[last] = L[i];

}

**if**(first < last) p[last - 1] = GetIntersection(q[last - 1], q[last]);

}

**while**(first < last && !OnLeft(q[first], p[last - 1])) last -- ;

**if**(last - first <= 1) **return** 0;

p[last] = GetIntersection(q[last], q[first]);

LL m = 0;

**for**(LL i = first; i <= last; i++) poly[m++] = p[i];

**return** m;

}