# 计算几何模板

#include <iostream>

#include <algorithm>

#include <cmath>

using namespace std;

const double eps=1e-10;//精度控制

const double maxl=1e10;//全局最大值

const double pi=3.1415926535897932384626433832795;

const int maxv=10100;//多边形顶点最大值

template <class T>

T Max(T a,T b){return a>b?a:b;}

template <class T>

T Min(T a,T b){return a<b?a:b;}

double sqr(double a){return a\*a;}

int sgn(double a){if (a>eps) return 1;if (a<-eps) return -1;return 0;}

struct Point

{

double x,y;

Point(){};

Point(double dx,double dy){x=dx;y=dy;}

inline int init(){ return scanf("%lf%lf",&x,&y); }

};

inline double dis(Point a,Point b)

{return sqrt(sqr(a.x-b.x)+sqr(a.y-b.y));}

inline Point operator -(Point a,Point b)

{Point c;c.x=a.x-b.x;c.y=a.y-b.y;return c;}

inline Point operator +(Point a,Point b)

{Point c;c.x=a.x+b.x;c.y=a.y+b.y;return c;}

inline Point operator \*(Point a,double d)

{Point c;c.x=a.x\*d;c.y=a.y\*d;return c;}

inline Point operator \*(double d,Point a)

{Point c;c.x=a.x\*d;c.y=a.y\*d;return c;}

inline Point operator /(Point a,double d)

{Point c;c.x=a.x/d;c.y=a.y/d;return c;}

inline double operator \*(Point a,Point b)

{return a.x\*b.x+a.y\*b.y;}

inline int operator ==(Point a,Point b){return dis(a,b)<eps;}

inline int operator !=(Point a,Point b){return dis(a,b)>=eps;}

inline bool operator < (const Point &l, const Point &r)

{return sgn(l.y-r.y)<0 ||( sgn(l.y-r.y)==0 && sgn(l.x-r.x)<0 );}

inline double crossmuti(Point a,Point b){return a.x\*b.y-a.y\*b.x;}

inline bool crossmuti(Point sp, Point ep, Point op)

{

return sgn((sp.x - op.x) \* (ep.y - op.y)

-(ep.x - op.x) \* (sp.y - op.y))>=0;

}

struct Polygon

{

Point vertex[maxv];

int numv;

Polygon(){numv=0;}

Polygon(Point \*s,int num)

{numv=num;memcpy(vertex,s,num\*sizeof(Point));}

Point &operator [](int k){return vertex[k];}

};

int findtubao(Point\* pnt, int n, Polygon &res)//凸包部分 返回凸包顶点个数

{

int i, len, k = 0, top = 1;

sort(pnt, pnt + n);

res.numv=n;//防止后面三行return的时候没赋顶点数

for (i=0;i<3;i++) res[i]=pnt[i];

if (n<3) return n;

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

{

while (top && crossmuti(pnt[i], res[top], res[top-1]))

top--;

res[++top] = pnt[i];

}

len = top; res[++top] = pnt[n - 2];

for (i = n - 3; i >= 0; i--)

{

while (top!=len && crossmuti(pnt[i], res[top], res[top-1]))

top--;

res[++top] = pnt[i];

}

res.numv=top;

return top; // 返回凸包中点的个数

}

int compagu(const void \*e,const void \*b)//极角排序 要求极点必须是(0,0) 没用函数

{

Point\* p=(Point\*) e,\*q=(Point\*) b;

if (\*p==\*q) return 0;

if (sgn(atan2(p->y,p->x)-atan2(q->y,q->x))!=0)

return sgn(atan2(p->y,p->x)-atan2(q->y,q->x));

else return sgn(dis(\*p,Point(0,0))-dis(\*q,Point(0,0)));

}

struct LineSegment

{

Point pt1,pt2;

LineSegment(){};

LineSegment(Point p1,Point p2){pt1=p1;pt2=p2;}

};

struct Line

{

double a,b,c; //ax+by+c==0

double f; //正向法向量atan2值 注意到ax+by+c>=0的正向法向量恰好是(a,b)

Line(){};

Line(double p,double q,double r)

{

a=p;b=q;c=r;

f=atan2(b,a);

double one=sqrt(a\*a+b\*b);

a/=one;b/=one;c/=one;

}

Line(Point p,Point q) //将点q-〉p转成半平面 直线和半平面正向法向量成右手螺旋

{

a=q.y-p.y;

b=p.x-q.x;

c=p.y\*q.x-p.x\*q.y;

f=atan2(b,a);

double one=sqrt(a\*a+b\*b);

a/=one;b/=one;c/=one;

}

Line(LineSegment l)

{

Line ls(l.pt1,l.pt2);

a=ls.a;b=ls.b;c=ls.c;

f=atan2(b,a);

double one=sqrt(a\*a+b\*b);

a/=one;b/=one;c/=one;

}

double func(const Point &p){return a\*p.x+b\*p.y+c;}

};

double countarea(Polygon &p) //计算面积

{

int i;

double tot=crossmuti(p[p.numv-1],p[0]);

for (i=0;i+1<p.numv;i++) tot+=crossmuti(p[i],p[i+1]);

return fabs(tot/2.0);

}

double disl(Line l,Point p){return fabs(l.a\*p.x+l.b\*p.y+l.c)/sqrt(sqr(l.a)+sqr(l.b));}

//点到直线距离

double disls(LineSegment b,Point a)

{//点到线段距离

if (sgn((a-b.pt1)\*(b.pt2-b.pt1))>=0&&sgn((a-b.pt2)\*(b.pt1-b.pt2))>=0)

return disl(b,a);

else return Min(dis(a,b.pt1),dis(a,b.pt2));

}

int ondiffside(Line l,Point a,Point b)

{//判断ab是否在l两侧

return sgn(l.func(a))\*sgn(l.func(b))<0;

}

int PntonSeg(LineSegment ls,Point p)

{//判断点是否在线段上

Line l(ls.pt1,ls.pt2);

if (ls.pt1==p||ls.pt2==p) return 1;

if (sgn(l.func(p))==0&&((ls.pt1<p&&p<ls.pt2)||(ls.pt2<p&&p<ls.pt1))) return 1;

return 0;

}

int Intersect(LineSegment L1, LineSegment L2)

{//判断两条线段是否相交 若规范相交 返回1，不规范,返回-1，不相交返回0

if (ondiffside(L1,L2.pt1,L2.pt2)&&ondiffside(L2,L1.pt1,L1.pt2))

return 1;

if (PntonSeg(L1,L2.pt1)||PntonSeg(L1,L2.pt2)||PntonSeg(L2,L1.pt1)||PntonSeg(L2,L1.pt2))

return -1;

return 0;

}

inline int intersection(const Line &p,const Line &q,Point &s)

{//求两直线的交点存入s 返回是否成功

// if (sgn(p.a\*q.b-p.b\*q.a)==0) return 0;

s.y=(p.a\*q.c-p.c\*q.a)/(q.a\*p.b-p.a\*q.b);

s.x=(p.b\*q.c-p.c\*q.b)/(p.a\*q.b-p.b\*q.a);

return 1;

}

Point projection(Point p,const Line &l)

{

if (sgn(disl(l,p))==0) return p;

Line r;

r.a=l.b;r.b=-l.a;r.c=-r.func(p);

intersection(l,r,p);

return p;

}

struct Circle

{

Point c;

double r;

Circle(){}

Circle(Point \_h,double \_r){c=\_h;r=\_r;}

};

void intersection\_line\_circle(const Circle &o, const Point &l1, const Point &l2,

Point &p1, Point &p2) {

Point p = o.c;

p.x += l1.y - l2.y;

p.y += l2.x - l1.x;

intersection(Line(p, o.c), Line(l1, l2),p);

double t = sqrt(o.r \* o.r - dis(p, o.c) \* dis(p, o.c)) / dis(l1, l2);

p1 = p + (l2 - l1) \* t;

p2 = p - (l2 - l1) \* t;

}

void intersection\_circle\_circle(const Circle &c1,const Circle &c2,Point& p1,Point& p2) {

Point u, v;

double t = (1+(sqr(c1.r)-sqr(c2.r))/dis(c1.c,c2.c)/dis(c1.c,c2.c))/2;

u=c1.c+(c2.c-c1.c)\*t;

v.x = u.x+c1.c.y-c2.c.y;

v.y = u.y-c1.c.x+c2.c.x;

intersection\_line\_circle(c1,u,v,p1,p2);

}

double area\_circle\_halfsegment(const Circle &c,const Point &h,const Point &p)

{

double d1=dis(c.c,h),d2=dis(c.c,p),l1;

if (sgn(dis(p,c.c)-c.r)<=0) return d1\*dis(h,p)/2;

if (sgn(d1-c.r)>=0) l1=0;else

l1=sqrt(sqr(c.r)-sqr(d1));

return d1\*l1/2+(acos(d1/d2)-asin(l1/c.r))\*sqr(c.r)/2;

}

double area\_circle\_segment(const Circle &c,const Point &p1,const Point &p2)

{

if (p1==p2) return 0;

Point h=projection(c.c,Line(p1,p2));

return fabs(area\_circle\_halfsegment(c,h,p1)\*sgn((p1-h)\*(h-p2))+area\_circle\_halfsegment(c,h,p2));

}

double area\_circle\_Polygon(Circle c,Polygon &p)

{

double tot=0;

p[p.numv]=p[0];

for (int i=0;i<p.numv;i++)

tot+=area\_circle\_segment(c,p[i],p[i+1])\*sgn(crossmuti(p[i]-c.c,p[i+1]-c.c));

return fabs(tot);

}

double countangular(Point a,Point b,Point c) //countangu a->b&&a->c return pi

{//计算向量a->b和a->c的夹角,返回pi进制

double l1,l2,l3;

l1=dis(a,b);l2=dis(a,c);l3=dis(b,c);

if (sgn(l1-l2)==0&&sgn(l3)==0) return 0;

if (sgn(l1+l2-l3)==0) return pi;

return acos((sqr(l1)+sqr(l2)-sqr(l3))/2.0/l1/l2);

}

Point circumcenter(Point A,Point B,Point C)//外心

{

Point a=C-B,b=A-C,c=B-A;

double af,bf,cf,s;

s=2\*sqr(fabs(crossmuti(c,a)));

af=-(a\*a)\*(c\*b)/s;

bf=-(b\*b)\*(c\*a)/s;

cf=-(c\*c)\*(b\*a)/s;

Point res=af\*A+bf\*B+cf\*C;

return res;

}

Point orthocenter(Point A,Point B,Point C)//垂心

{

Point a=C-B,b=A-C,c=B-A;

double af,bf,cf;

af=(a\*b)\*(a\*c);

bf=(b\*c)\*(b\*a);

cf=(c\*a)\*(c\*b);

Point res=(af\*A+bf\*B+cf\*C)/(af+bf+cf);

return res;

}

Point incenter(Point A,Point B,Point C)//内心

{

double a=dis(B,C),b=dis(C,A),c=dis(A,B);

return (a\*A+b\*B+c\*C)/(a+b+c);

}

int RotatingCalipers(Polygon &p1,int k1,Polygon &p2,int k2,Line &l)//逆时针转 返回贴着的那条线 如果第一条贴 返回1 如果第二条 返回2 如果两条都贴 返回3

{ //注意 当求两个凸多边形的最近距离时 小心相交情况判多边形交 如果两个多边形套一起 起始射线方向同向

int k11=(k1+1)%p1.numv,k22=(k2+1)%p2.numv;

Line l1(p1[k1],p1[k11]),l2(p2[k2],p2[k22]);

double aug1=atan2(l1.a,-l1.b)-atan2(l.a,-l.b),aug2=atan2(l2.a,-l2.b)-atan2(-l.a,l.b);

if (aug1<0) aug1+=2\*pi;

if (aug2<0) aug2+=2\*pi;

if (sgn(aug1-aug2)==0) {k1=k11;k2=k22;l=l1;return 3;}

if (aug1<aug2){l=l1;return 1;}

if (aug1>aug2){l=l2;return 2;}

}

//////////////////以下为半平面交部分

Line linemove(Line s,double d)//半平面正向平移d

{

s.c-=d\*sqrt(sqr(s.a)+sqr(s.b));

return s;

}

int needtopop(Line s,Line a,Line b)//是否需要退栈 看a,b的交点是否在s外

{

Point cro;

intersection(a,b,cro);

return sgn(s.func(cro))<=0;

}

int compf(const void \*a,const void \*b)//正法向排序 如果相同 按c排序

{

return sgn(((Line\*)a)->f-((Line\*)b)->f)\*10+sgn(((Line\*)a)->c-((Line\*)b)->c);

}

int banpingmianjiao(Line\* c,int n,Line\* que,int& head,int& tail)

{//半平面交 注意传入的c必须有外围边框保证封闭 否则手动加 最后返回在head和tail之间那段

int i,j;

qsort(c,n,sizeof(Line),compf);//极角排序

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

if (sgn(c[i-1].f-c[i].f)==0) j++;

else c[i-j]=c[i];//干掉极角相同的

n-=j;

head=0;tail=1;

que[0]=c[0];que[1]=c[1];

for (i=2;i<n;i++) //加线进去

{

while (head<tail&&needtopop(c[i],que[tail],que[tail-1])) tail--;//注意这两句不能写反!

while (head<tail&&needtopop(c[i],que[head],que[head+1])) head++;//注意这两句不能写反!

que[++tail]=c[i];

}

while (head<tail&&needtopop(que[head],que[tail],que[tail-1])) tail--;//删掉尾部多余线

while (head<tail&&needtopop(que[tail],que[head],que[head+1])) head++;//删掉头部多余线

return tail-head>=2;// 必须有3条边 否则挂

}

//////////////////////////////////////////////////////////////////////////

//旋转卡壳样例:求俩凸包距离

/\*

int n,m;

Polygon p1,p2,pp1,pp2;

void work()

{

int i,j,k1,k2;

double mint;

Line l(Point(0,0),Point(1,0));

for (i=0;i<n;i++) p1.vertex[i].init();

for (i=0;i<m;i++) p2.vertex[i].init();

p1.numv=n;p2.numv=m;

pp1=p1;pp2=p2;

findtubao(pp1.vertex,pp1.numv,p1);

findtubao(pp2.vertex,pp2.numv,p2);

n=p1.numv;m=p2.numv;

k1=k2=0;

for (i=0;i<n;i++) if (sgn(p1.vertex[i].y-p1.vertex[k1].y)<0) k1=i;

for (i=0;i<m;i++) if (sgn(p2.vertex[i].y-p2.vertex[k2].y)>0) k2=i;

mint=dis(p1.vertex[k1],p2.vertex[k2]);

int tot1=0,tot2=0;

do

{

int k=RotatingCalipers(p1,k1,p2,k2,l);

int kp1=(k1+1)%n,kp2=(k2+1)%m;

mint=Min(mint,disls(LineSegment(p1.vertex[k1],p1.vertex[kp1]),p2.vertex[k2]));

mint=Min(mint,disls(LineSegment(p2.vertex[k2],p2.vertex[kp2]),p1.vertex[k1]));

mint=Min(mint,disls(LineSegment(p1.vertex[k1],p1.vertex[kp1]),p2.vertex[kp2]));

mint=Min(mint,disls(LineSegment(p2.vertex[k2],p2.vertex[kp2]),p1.vertex[kp1]));

if (k&1) {k1=kp1;tot1++;}

if (k&2) {k2=kp2;tot2++;}

} while (tot1<n||tot2<m);//这里是或!!!!!以前的代码错了

printf("%.5lf\n",mint);

}

\*/

char ot[100];

char\* o;

void getf(double &a)

{

int f=1,dot=0;

if (\*o=='-') {f=-1;o++;}

for (a=0;\*o&&\*o!=' ';o++)

if (\*o=='.') dot++;

else

{

if (dot) dot++;

a=a\*10+\*o-48;

}

o++;

if (dot)

while (--dot) a/=10;

a\*=f;

}

# 矩阵乘法模板

#include <iostream>

using namespace std;

const int maxn=100;

typedef \_\_int64 lld;

inline lld Max(lld a,lld b){return a>b?a:b;}

lld minl;

struct Matrix

{

lld mat[maxn][maxn];

int n,m;

Matrix(){n=m=maxn;memset(mat,0,sizeof(mat));}

lld\* operator [] (int k) {return mat[k];}

void printM(){for (int i=0;i<n;i++){ for (int j=0;j<m;j++) cout<<(int)mat[i][j]<<' ';cout<<endl;}}

};

lld mod;

Matrix operator\*(Matrix &a,Matrix &b)

{

Matrix c;

c.n=a.n;c.m=b.m;

for (int i=0;i<a.n;i++)

{

for (int j=0;j<a.m;j++)

{

if (a.mat[i][j]==0)continue;

for (int k=0;k<b.m;k++)

{

c[i][k]+=a[i][j]\*b[j][k];

c[i][k]%=mod;

}

}

}

return c;

}

Matrix operator ^(Matrix &a,int m)

{

int j;

Matrix b,c = a;

b.n=a.n;b.m=a.m;

for (j=0;j<b.n;j++) b[j][j]=1;

//for (i=0;i<b.n;i++) for (j=0;j<b.m;j++) b[i][j]=minl;//如果是floyed形式的用这两行初始化b

//for (j=0;j<b.n;j++) b[j][j]=0;

while (m)

{

if (m & 1) b = b\*c;

if (m >>= 1) c = c\*c;

}

return b;

}

# 双连通分量

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <string>

#include <cmath>

#include <vector>

using namespace std;

const int maxn=110000,maxm=210000;

inline int Min(int a,int b){return a<b?a:b;}

struct Edge

{

int y,ne;

};

struct Tarjan\_Cut\_Point

{

int d[maxn],low[maxn],index;

int \*st,ee;

Edge \*e;

int n;

int s[maxn],ss;

int color[maxn],cc;//双连通分量染色，0 .. cc-1，cc表示连通分量数

int expt[maxn];

//一个割点可能属于很多分块，但是每个分块只有唯一一个“借来”的点，就是最高点。该数组保存这个特殊的最高点

//注意由于是借来的，在遍历相同颜色分块的时候，要特判这个点，以及遍历完后该点的所有信息要还原！

bool isCut[maxn];//一个点是否是割点

void init(Edge\* \_e,int\* \_st,int \_ee,int \_n)

{//传进双向边集，以及点数n,点编号0..n-1

e=\_e;st=\_st;ee=\_ee;n=\_n;

for (int i=0;i<n;i++) d[i]=low[i]=isCut[i]=0,color[i]=-1;

index=0;ss=-1;

cc=0;

}

void dfs(int u,int fa)//Tarjan部分

{

int i,son=0;

d[u]=low[u]=++index;

s[++ss]=u;

for (i=st[u];i!=-1;i=e[i].ne) if (e[i].y!=fa)

if (d[e[i].y]==0)

{

son++;//统计儿子数，树根要用

dfs(e[i].y,u);

low[u]=Min(low[u],low[e[i].y]);

if (low[e[i].y]>=d[u])

{

isCut[u]=1;//如果非树根，该点是割点

while (s[ss]!=e[i].y)

color[s[ss--]]=cc;

color[s[ss--]]=cc;

//将该儿子所在一下点全部出栈，这些点，加上u，是一个分块

color[expt[cc]=u]=cc;//u作为特殊点保存

cc++;

}

}

else

low[u]=Min(low[u],d[e[i].y]);

if (u==fa)

isCut[u]=(son>1);

//树根是割点的条件：有两个以上儿子

}

void work()

{

for (int i=0;i<n;i++) if (d[i]==0)

{

ss=-1;

dfs(i,i);

if (color[i]==-1) color[i]=cc++;

//防止该连通分量只有一个点的情况 这种情况算不算一个分块根据题目要求而定

}

}

};

Tarjan\_Cut\_Point T;

Edge e[maxm];

int st[maxn],ee;

void addedge(int x,int y)

{

e[ee].y=y;e[ee].ne=st[x];st[x]=ee++;

e[ee].y=x;e[ee].ne=st[y];st[y]=ee++;

}

int color[maxn];

int h[maxn];

int que[maxn],head,tail;

int bfs(int k,int c)

{

int i;

que[0]=k;head=0;tail=1;

while (head<tail)

{

k=que[head++];

for (i=st[k];i!=-1;i=e[i].ne) if (T.color[e[i].y]==c||e[i].y==T.expt[c])

if (color[e[i].y]==-1)

{

color[e[i].y]=1-color[k];

que[tail++]=e[i].y;

}

else if (color[e[i].y]==color[k]) return 1;

}

return 0;

}

char buf[1048576],\*o;

int getint() {

while (!isdigit(\*o)) o++;

int f = 0;

while (isdigit(\*o)) f=f\*10+\*o-'0',o++;

return f;

}

int can[maxn];

int main()

{

#ifdef \_DEBUG

freopen("D.in", "r", stdin);

#endif

int n,m,i,j,k;

int cass;

for (scanf("%d\n",&cass);cass--;)

{

o = gets(buf);

n = getint(); m = getint();

for (i=0;i<n;i++) st[i]=color[i]=h[i]=-1,can[i]=0;

ee=0;

while (m--)

{

o = gets(buf);

i = getint()-1; j = getint()-1;

addedge(i,j);

}

T.init(e,st,ee,n);

T.work();

k=0;

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

{

if (h[T.color[i]]==-1)

{

color[T.expt[T.color[i]]]=-1;

color[i]=0;

h[T.color[i]]=bfs(i,T.color[i]);

can[T.expt[T.color[i]]]|=h[T.color[i]];

color[T.expt[T.color[i]]]=-1;

}

can[i]|=h[T.color[i]];

}

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

k+=can[i];

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

}

return 0;

}

# 2-sat

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <string>

#include <cmath>

#include <vector>

using namespace std;

const int maxn=101000,maxm=1000000;

inline int Min(int a,int b){return a<b?a:b;}

struct Edge

{

int y,ne;

};

struct Tarjan\_Strong\_Connected

{

int d[maxn],low[maxn],index;

int \*st,ee;

Edge \*e;

int n;

int s[maxn],ss;

bool cs[maxn];

int color[maxn],cc;//环染色，0 .. cc-1，cc表示环数

void init(Edge\* \_e,int\* \_st,int \_ee,int \_n)

{

e=\_e;st=\_st;ee=\_ee;n=\_n;

for (int i=0;i<n;i++) d[i]=low[i]=cs[i]=0;

index=0;ss=-1;

cc=0;

}

void dfs(int u)

{

d[u]=low[u]=++index;

s[++ss]=u;

cs[u]=1;

for (int i=st[u];i!=-1;i=e[i].ne)

if (d[e[i].y]==0)

{

dfs(e[i].y);

low[u]=Min(low[u],low[e[i].y]);

}

else if (cs[e[i].y])

low[u]=Min(low[u],d[e[i].y]);

if (d[u]==low[u])

{

while (cs[u])

{

color[s[ss]]=cc;

cs[s[ss--]]=0;

}

cc++;

}

}

void work()

{

for (int i=0;i<n;i++) if (d[i]==0) dfs(i);

}

};

struct Two\_Sat //点0..2n-1,一个点i的对立点为i^1

{

Edge e[maxm],l[maxm];

int st[maxn],ee,lst[maxn],ll;

int rudu[maxn];

//e,st,ee保存原图，l,lst,ll保存缩环后的反向拓扑图

int n; //顶点数，即对数\*2

Tarjan\_Strong\_Connected T;

int color\_t[maxn];

//缩环后的点的选取与否，-1尚未处理，1选，2不选

int s[maxn],ss; //拓扑排序中保存入度为0的栈

int color[maxn]; //真正的返回每个点或其对立是否被选取，1选，2不选。保证与对立的值不同

vector<int> cir[maxn]; //缩环时，保存每个环上对应哪些点（速度较慢，当数据小时直接for）

void addedge(int x,int y) //手动添加边

{

e[ee].y=y;e[ee].ne=st[x];st[x]=ee++;

}

void T\_addedge(int x,int y) //内部函数 缩环拓扑加边

{

if (x==y) return ;

l[ll].y=y;l[ll].ne=lst[x];lst[x]=ll++;

rudu[y]++;

}

void init(int \_n) //初始化 传进点对数\*2 编号从0开始

{

ee=ll=0;

n=\_n;

int i;

for (i=0;i<n;i++) lst[i]=color\_t[i]=-1,rudu[i]=0,st[i]=-1,cir[i].clear();

ss=-1;

}

void dele(int k) //从拓扑图中删除一个点

{

for (int i=lst[k];i!=-1;i=l[i].ne)

if (--rudu[l[i].y]==0&&color\_t[l[i].y]==-1)

s[++ss]=l[i].y; //如果有点入度变成0，入栈

}

void dfs\_dele(int k) //从拓扑图中将k以下点全部移除

{

color\_t[k]=2; //该点标记不选

for (int i=lst[k];i!=-1;i=l[i].ne) if (color\_t[l[i].y]!=2)

dfs\_dele(l[i].y);

//这里直接删，不需要考虑入栈情况，因为子点也被删了

}

int work()

{

T.init(e,st,ee,n);

T.work(); //Tarjan 将环染色0 .. T.cc-1

int i,j;

for (i=0;i<n;i++) if (T.color[i]==T.color[i^1]) return 0; //判无解

else cir[T.color[i]].push\_back(i);

//记入每个环有哪些点

//以下构造一组解。注意如果要求解中必须包含某个点k，只需在初始化时添加边addedge(k^1,k)即可

for (i=0;i<n;i++) for (j=st[i];j!=-1;j=e[j].ne)

T\_addedge(T.color[e[j].y],T.color[i]);

//构建反向拓扑图，注意这里加边是加反向的

for (i=0;i<T.cc;i++) if (rudu[i]==0) s[++ss]=i;//栈初始化

while (ss>=0)

{

if (color\_t[i=s[ss--]]!=-1) continue; //出栈

dele(i); //将点从拓扑图中删除

color\_t[i]=1; //该点标记选

for (j=cir[i].size()-1;j>=0;j--)

dfs\_dele(T.color[cir[i][j]^1]);

//将环的对立点及其以下点全部标记不选删除

}

for (i=0;i<n;i++) color[i]=color\_t[T.color[i]];

//返回原图选取情况

return 1;

}

};

Two\_Sat TS;

int main()

{

int n,m;

#ifdef \_DEBUG

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

#endif

while (scanf("%d%d",&n,&m),n||m)

{

int i,j,k;

char s[3],t[3];

TS.init(n\*2);

while (m--)

{

scanf("%d%s%d%s",&i,s,&j,t);

i=i\*2+(s[0]=='w');j=j\*2+(t[0]=='w');

if (i==1||j==1) continue;

TS.addedge(i,j^1);

TS.addedge(j,i^1);

}

TS.addedge(1,0);

if (TS.work()==0) puts("bad luck");

else

for (i=2;i<2\*n;i++) if (TS.color[i]==2)

{

if (i&1) printf("%dw",i/2);

else printf("%dh",i/2);

if (i+2>=2\*n) puts("");else putchar(' ');

}

}

return 0;

}

# 线性筛素数

#include <iostream>

using namespace std;

const int maxn=10000;

int n=maxn;

bool IsPrime[maxn];

int Prime[maxn];

int num;

void Eratosthenes ()

{

num = 1;

memset(Prime,0,sizeof(Prime));

memset(IsPrime,1,sizeof(IsPrime));

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

{

if (IsPrime[i])

Prime[num++] = i;

for (int j=1;j<num && i\*Prime[j]<=n;j++)

{

IsPrime[i\*Prime[j]] = 0;

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

}

}

}

# DFA

////////////////////////////////////dfa模板

const int maxch=52,maxnode=maxn;

//maxch=最大字符集大小,maxnode=最大节点数(无用)

struct DFA

{

DFA\* pt[maxch];//指针

DFA\* pre;//前向指针

int term,id;

//是否是终结(这个term是0，表示非终结,否则是终结点的个数,id=这个点的标号

DFA(){memset(pt,NULL,sizeof(pt));pre=NULL;term=0;}

};

void addDFA(int\* s,int ss,DFA \*root,int& nDFA)

//建trie树 nDFA是已有节点数,\*s是要插入的串,ss是串长

{

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

{

if (root->pt[s[i]]==NULL)

root->pt[s[i]]=new DFA,root->pt[s[i]]->id=nDFA++;

root=root->pt[s[i]];

}

root->term++;//尾巴的终结个数++

}

DFA\* bfs[maxnode];//bfs建立DFA

int head,tail;//bfs的两个指针

void makeDFA(DFA \*root,int mch)//DFA建立前向指针,然后完成所有虚节点的跳转

{

int i,j,k;

DFA\* proot=new DFA;//增加一个临时的超父亲,其所有指针指向root

for (i=0;i<mch;i++) proot->pt[i]=root;

root->pre=proot;

head=0,tail=1;

bfs[0]=root;//开始bfs

while (head<tail)

{

for (i=0;i<mch;i++) if (bfs[head]->pt[i]!=NULL)//扩展bfs

{

DFA\* p=bfs[head]->pt[i];

bfs[tail++]=p;//增加新节点

for(p->pre=bfs[head]->pre;(p->pre)->pt[i]==NULL;p->pre=(p->pre)->pre);

//构建该节点的前向指针

p->pre=(p->pre)->pt[i];

}

else bfs[head]->pt[i]=bfs[head]->pre->pt[i];

//否则 这个指针是虚指针,完成跳转

bfs[head]->term+=bfs[head]->pre->term;//注意!终结标记传递

head++;

}

}

///////////////////////////////////

# 只支持+,-,\*的高精度

///////////////////////////////////

#include <cmath>

const int maxBt=30,maxUt=4,maxMo=pow(10.0,maxUt);

//maxBt是高精度数的长度,maxUt是进制,maxMo是10^maxUt

struct BigInt

{

int s[maxBt];

int ss;

BigInt()

{

ss=0;

memset(s,0,sizeof(s));

}

BigInt(int k)//用整数k初始化

{

ss=0;

memset(s,0,sizeof(s));

while (k) s[ss++]=k%maxMo,k/=maxMo;

}

int& operator [](int k){return s[k];}

void printBt()//输出

{

if (ss==0) printf("0\n");

else

{

printf("%d",s[ss-1]);

for (int i=ss-2;i>=0;i--) printf("%04d",s[i]);

//注意这里输出的进制格式补0，当进制调整时一起调整

puts("");

}

}

};

BigInt operator +(BigInt a,BigInt b)

{

int i,t=0,code;

for (i=0;i<a.ss||i<b.ss||t!=0;i++)

{

code=a[i]+b[i]+t;

a[i]=code%maxMo;

t=code/maxMo;

}

a.ss=i;return a;

}

BigInt operator \*(BigInt a,BigInt b)

{

int i,j,t=0,code;

BigInt c;

for (i=0;i<a.ss;i++)

{

for (j=t=0;j<b.ss||t>0;j++)

{

code=a[i]\*b[j]+t+c[i+j];

c[i+j]=code%10000;

t=code/10000;

}

if (c.ss<i+j) c.ss=i+j;

}

return c;

}

# PMK算法

#include <iostream>

using namespace std;

const int maxn= 410000,maxm=1100;

void count(char\* p,int\* f,int m)

{//模板串p,前缀最长循环匹配输出到f,模板串长度m

int i,k;

for (f[k=1]=1;f[1]<m&&p[f[1]]==p[0];f[1]++);

for (i=2;i<=m;i++)

if (f[i-k]+k<f[k]) f[i]=f[i-k]+k;

else

for (f[i]=f[k],k=i;f[i]<m&&p[f[i]]==p[f[i]%i];f[i]++);

}

void pipei(char\* s,int n,char\* p,int m,int\* f,int\* g)

{//匹配串s,长度n,匹配串p,长度m,循环最长前缀f,输出s-p前缀匹配到g

int i,k;

for (g[k=0]=0;g[0]<m&&g[0]<n&&p[g[0]]==s[g[0]];g[0]++);

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

{

g[i]=f[i-k]-(i-k);

if (g[i]+i>=g[k]+k||i>=g[k]+k)

{

g[i]=g[k]-i+k;

if (g[i]<0) g[i]=0;

while (g[i]+i<n&&g[i]<n&&s[g[i]+i]==p[g[i]]) g[i]++;

k=i;

}

}

}

# KMP算法（浙大模板）

/模式匹配,kmp算法,复杂度O(m+n)

//返回匹配位置,-1表示匹配失败,传入匹配串和模式串和长度

//可更改元素类型,更换匹配函数

#define MAXN 10000

#define \_match(a,b) ((a)==(b))

typedef char elem\_t;

int pat\_match(int ls,elem\_t\* str,int lp,elem\_t\* pat){

int fail[MAXN]={-1},i=0,j;

for (j=1;j<lp;j++){

for (i=fail[j-1];i>=0&&!\_match(pat[i+1],pat[j]);i=fail[i]);

fail[j]=(\_match(pat[i+1],pat[j])?i+1:-1);

}

for (i=j=0;i<ls&&j<lp;i++)

if (\_match(str[i],pat[j]))

j++;

else if (j)

j=fail[j-1]+1,i--;

return j==lp?(i-lp):-1;

}

# 最大匹配模板

#include <iostream>

using namespace std;

const int maxn=1100000,maxm=2\*maxn;

struct Edge

{

int y;

int ne;

} e[maxm];

int st[maxn],ee;

void addedge(int x,int y)

{

e[ee].y=y;e[ee].ne=st[x];st[x]=ee++;

}

int p[maxn],c[maxn];

int aug(int k,int m,int hash)

{

int i;

if (c[k]==hash) return 0;

c[k]=hash;

for (i=st[k];i!=-1;i=e[i].ne) if (e[i].y<m&&p[e[i].y]==-1)

{

p[e[i].y]=k;

return 1;

}

for (i=st[k];i!=-1;i=e[i].ne)

if (e[i].y<m&&aug(p[e[i].y],m,hash))

{

p[e[i].y]=k;

return 1;

}

return 0;

}

int maxmatch(int n,int m)

{

memset(p,-1,sizeof(int)\*m);

memset(c,0,sizeof(int)\*n);

int i,tot=0;

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

{

tot+=aug(i,m,i+1);

if (tot>=m) return 1;

}

return 0;

}

# 最大流SAP模板

#include <iostream>

using namespace std;

const int maxn=20100,maxm=200100\*4;

typedef int lld;

struct Edge

{

int y,f,bro,ne;

} e[maxm];

int ee;

int st[maxn];

void addedge(int x,int y,int f,int ff)

{//x->y增加一条容量f的正向边和容量ff的反向边

e[ee].y=y;e[ee].f=f;e[ee].ne=st[x];st[x]=ee++;

e[ee].y=x;e[ee].f=ff;e[ee].ne=st[y];st[y]=ee++;

e[ee-2].bro=ee-1;e[ee-1].bro=ee-2;

}

#define INF (lld)1<<30

int h[maxn],vh[maxn],n;

//注意 这里n是全局变量 表示节点个数

lld flow;

lld aug(const int m,const int ex,const lld t)

{//目标ex t是流量 m是当前所在节点

int i,minh;

lld l,d;

if(m== ex) return t;

l = t;

minh = n - 1;

for(i=st[m];i!=-1;i=e[i].ne) if(e[i].f>0){

if(h[e[i].y]+1==h[m]){

if(l<e[i].f) d=l; else d=e[i].f;

d = aug(e[i].y,ex,d);

e[i].f-=d;

e[e[i].bro].f+=d;

l-=d;

if(h[0]>=n) return t-l;

if (l==0) break;

}

if(minh>h[e[i].y]) minh=h[e[i].y];

}

if(l==t)

{

vh[h[m]]--;

if(vh[h[m]]==0) h[0]=n;

h[m] = minh + 1;

vh[h[m]]++;

}

return t-l;

}

lld maxflow(int sx,int ex)

{

memset(h,0,sizeof(h));

memset(vh,0,sizeof(vh));

vh[sx] = n;

flow=0;

while( h[sx]<n )

flow += aug(sx,ex,INF);

return flow;

}

# 最小费用流模板

#include <iostream>

using namespace std;

const int maxn=1000,maxm=maxn\*8\*4;//最大顶点数和边数

const int maxl=999999999;

inline int Min(int a,int b){return a<b?a:b;}

inline int Max(int a,int b){return a>b?a:b;}

struct st

{

int y,d;

int ne;

int bro;

int f;

} e[maxm];

int ee;

int st[maxn];

int n,m;

void addedge(int x,int y,int d,int f)

{//给顶点x和y间添加一条费用d,流量f的边

e[ee].y=y;e[ee].d=d;e[ee].ne=st[x];e[ee].f=f;st[x]=ee++;

e[ee].y=x;e[ee].d=-1\*d;e[ee].ne=st[y];e[ee].f=0;st[y]=ee++;

e[ee-2].bro=ee-1;e[ee-1].bro=ee-2;

}

int d[maxn],p[maxn];

//spfa所用到起点的最短距离(这里距离相当于cost)和路径记录之前的一个节点

int c[maxn];//spfa所用数组:是否在队列中

int que[maxn],head,tail;//spfa专用队列

int spfa(int sx,int ex)//求sx到ex的一次费用增广

{//如果没有增广路就返回maxl 否则返回费用

int i,j,k;

for (i=0;i<maxn;i++) d[i]=maxl;//初始化d为maxl

memset(c,0,sizeof(c));//初始化都没进

d[sx]=0;

que[head=0]=sx;tail=1;

c[sx]=1;

while (head!=tail)//spfa开始

{

k=que[head++];head%=maxn;

c[k]=0;

for (i=st[k];i!=-1;i=e[i].ne) if (e[i].f)

if (d[k]+e[i].d<d[e[i].y])

{

d[e[i].y]=d[k]+e[i].d;

p[e[i].y]=i;

if (c[e[i].y]==0)

{

c[e[i].y]=1;

if (e[i].d<0){head=(head-1+maxn)%maxn;que[head]=e[i].y;}

else

{que[tail++]=e[i].y;tail%=maxn;}

}

}

}

if (d[ex]==maxl) return maxl;//如果无法到达终点返回maxl

k=maxl;

for (i=ex;i!=sx;i=e[e[p[i]].bro].y) k=Min(k,e[p[i]].f);//计算流

for (i=ex;i!=sx;i=e[e[p[i]].bro].y)//增加反向边

{

e[p[i]].f-=k;

e[e[p[i]].bro].f+=k;

}

return d[ex]\*k;//返回费用为流大小\*路径长度(cost累加)

}

# 带花树模板

#include<iostream>

using namespace std;

//ZOJ 3316 一般图匹配 武大的模板

const int maxn=361;

struct Graph

{

int n,match[maxn];

bool adj[maxn][maxn];

void clear(){memset(adj,0,sizeof(adj));n=0;}

void insert(const int &u,const int &v)

{n=max(n,max(u,v)+1);adj[u][v]=adj[v][u]=1;}

int max\_match()

{

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

int ans=0;

for(int i=0;i<n;i++)if(match[i]==-1)ans+=bfs(i);

return ans;

}

int Q[maxn],pre[maxn],base[maxn];

bool hash[maxn],in\_blossom[maxn];

int bfs(int p)

{

memset(pre,-1,sizeof(pre));memset(hash,0,sizeof(hash));

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

Q[0]=p;hash[p]=1;

for(int s=0,t=1;s<t;s++)

{

int u=Q[s];

for (int v=0;v<n;v++)

{

if(adj[u][v]&&base[u]!=base[v]&&v!=match[u])

{

if(v==p||(match[v]!=-1&&pre[match[v]]!=-1))

{

int b=contract(u,v);

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

{

if(in\_blossom[base[i]])

{

base[i]=b;

if(hash[i]==0){hash[i]=1;Q[t++]=i;

}

}

}

}

else if(pre[v]==-1)

{

pre[v]=u;

if(match[v]==-1){argument(v);return 1;}

else {Q[t++]=match[v];hash[match[v]]=1;}

}

}

}

}

return 0;

}

void argument(int u)

{

while(u!=-1)

{

int v=pre[u],k=match[v];

match[u]=v;match[v]=u;u=k;

}

}

void change\_blossom(int b,int u)

{

while(base[u]!=b)

{

int v=match[u];

in\_blossom[base[v]]=in\_blossom[base[u]]=true;

u=pre[v];

if(base[u]!=b)pre[u]=v;

}

}

int contract(int u,int v)

{

memset(in\_blossom,0,sizeof(in\_blossom));

int b=find\_base(base[u],base[v]);

change\_blossom(b,u);change\_blossom(b,v);

if(base[u]!=b)pre[u]=v;

if(base[v]!=b)pre[v]=u;

return b;

}

int find\_base(int u,int v)

{

bool in\_path[maxn]={};

while(1)

{

in\_path[u]=true;

if(match[u]==-1)break;

u=base[pre[match[u]]];

}

while(!in\_path[v])v=base[pre[match[v]]];

return v;

}

};

Graph ans;

int x[maxn],y[maxn];

inline int fab(int x){return x>0?x:-x;}

int main()

{

int n,m,L,i,j,k,cnt;

while(scanf("%d",&n)!=EOF)

{

ans.clear();

for(i=0;i<n;i++)scanf("%d%d",&x[i],&y[i]);

scanf("%d",&L);

for(i=0;i<n;i++)for(j=i+1;j<n;j++)if(fab(x[i]-x[j])+fab(y[i]-y[j])<=L)

ans.insert(i,j);

cnt=ans.max\_match();

if(cnt\*2==n)puts("YES");

else puts("NO");

}

return 0;

}

# 后缀数组模板

#include <iostream>

using namespace std;

#define maxn 6100

#define F(x) ((x)/3+((x)%3==1?0:tb))

#define G(x) ((x)<tb?(x)\*3+1:((x)-tb)\*3+2)

int wa[maxn],wb[maxn],wv[maxn],wss[maxn];

int c0(int \*r,int a,int b)

{return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];}

int c12(int k,int \*r,int a,int b)

{if(k==2) return r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);

else return r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];}

void sort(int \*r,int \*a,int \*b,int n,int m)

{

int i;

for(i=0;i<n;i++) wv[i]=r[a[i]];

for(i=0;i<m;i++) wss[i]=0;

for(i=0;i<n;i++) wss[wv[i]]++;

for(i=1;i<m;i++) wss[i]+=wss[i-1];

for(i=n-1;i>=0;i--) b[--wss[wv[i]]]=a[i];

return;

}

void dc3(int \*r,int \*sa,int n,int m)

{

int i,j,\*rn=r+n,\*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;

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

for(i=0;i<n;i++) if(i%3!=0) wa[tbc++]=i;

sort(r+2,wa,wb,tbc,m);

sort(r+1,wb,wa,tbc,m);

sort(r,wa,wb,tbc,m);

for(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)

rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;

if(p<tbc) dc3(rn,san,tbc,p);

else for(i=0;i<tbc;i++) san[rn[i]]=i;

for(i=0;i<tbc;i++) if(san[i]<tb) wb[ta++]=san[i]\*3;

if(n%3==1) wb[ta++]=n-1;

sort(r,wb,wa,ta,m);

for(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;

for(i=0,j=0,p=0;i<ta && j<tbc;p++)

sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];

for(;i<ta;p++) sa[p]=wa[i++];

for(;j<tbc;p++) sa[p]=wb[j++];

return;

}

int rank[maxn],height[maxn];

void calheight(int \*r,int \*sa,int n)

{

int i,j,k=0;

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

for(i=0;i<n;height[rank[i++]]=k)

for(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++);

return;

}

/////////用法

// dc3(r,sa,n+1,128);//转int后的字符串,注意手动r[n]=0 sa数组存放 长度+1(因为人工加个0) 字符集大小

// calheight(r,sa,n);//r同上 sa为上面返回的 n就是长度

// 所有数组都要开到3倍以上 如果是拼接串 要开到总长3倍

// height数组大小为n+1 其中height[0]无意义 height[1]==0 因为sa[0]=n=r[n]是附加小字符结尾 height[2..n]有意义

// r 和sa是要自己开的 rank和height在上面自带已声明

int mm[maxn];

int best[20][maxn];

void initRMQ(int n,int\* RMQ)

{//这个RMQ是从1..n的 但是height碰巧也是从1开始

int i,j,a,b;

for(mm[0]=-1,i=1;i<=n;i++)

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

for(i=1;i<=n;i++) best[0][i]=i;

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

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

{

a=best[i-1][j];

b=best[i-1][j+(1<<(i-1))];

if(RMQ[a]<RMQ[b]) best[i][j]=a;//最小RMQ

else best[i][j]=b;

}

return;

}

int askRMQ(int a,int b,int\* RMQ)

{

int t;

t=mm[b-a+1];b-=(1<<t)-1;

a=best[t][a];b=best[t][b];

return RMQ[a]<RMQ[b]?a:b;//最小RMQ

}

int lcp(int a,int b)

{

int t;

a=rank[a];b=rank[b];

if(a>b) {t=a;a=b;b=t;}

return height[askRMQ(a+1,b,height)];

}

# 高效回文字串模板

#include<iostream>

#include<string>

#include<algorithm>

using namespace std;

inline int Min(int a,int b) {return a<b?a:b;}

inline int Max(int a,int b) {return a>b?a:b;}

void cal(int pos,char\* s,int\* len,int slen)//所有数组长度加倍 中间和两端插空格的 len存每个点为对称轴的回文一半长度 slen存字符串长

{

int i,ans;

bool flag;

flag = true;

ans=0;

for(i=1;flag;i++)

if(pos-i>=0 && pos+i<slen)

if(s[pos-i]==s[pos+i])

{

i+=Min(len[pos-i],len[pos+i]);

ans=i;

}

else flag=false;

else flag=false;

len[pos]=ans;

}

void dfs(int l,int r,char\* s,int\* len,int slen)

{

if(l>r) return ;

int mid = (l+r)/2;

cal(mid,s,len,slen);

dfs(l,mid-1,s,len,slen);

dfs(mid+1,r,s,len,slen);

}

void work(char\* s,int\* len)

{//传入未加工的s(注意s里面不得有空格 否则修改间隔符) 输出到len 所有数组长度开到2倍+3以上

int j;

int l=strlen(s);

memset(len,0,sizeof(int)\*(l\*2+2));

for (j=l-1;j>=0;j--)

{

s[j\*2+1]=s[j];

s[j\*2]=' ';

}

s[l\*2]=' ';

s[l\*2+1]=0;

dfs(0,l\*2,s,len,l\*2+1);

}

# 离散化模板

struct Lisan

{

int x;

int\* idx;

} lisan[maxn];

int ll;

int compx(const void \*a,const void \*b)

{

return ((Lisan\*)a)->x-((Lisan\*)b)->x;

}

void lis()

{

int i,j,k;

ll=0;

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

{

lisan[ll].x=obj[i].l;

lisan[ll++].idx=&obj[i].l;

lisan[ll].x=obj[i].r;

lisan[ll++].idx=&obj[i].r;

}

qsort(lisan,ll,sizeof(Lisan),compx);

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

{

if (i>0&&lisan[i].x==lisan[i-1].x) j++;

\*lisan[i].idx=i-j;

lisan[i-j]=lisan[i];

}

ll-=j;

}

# 数论

/\*

\*功能: 扩展欧几里得

\*传入: a,b,x,y

\*返回: x,y满足ax+by=gcd(a,b)的一组解

\*/

template <class T>T EXT\_GCD(T a,T b,T& x,T& y)

{

T t,ret;

if (!b){

x=1,y=0;

return a;

}

ret=EXT\_GCD(b,a%b,x,y);

t=x,x=y,y=t-a/b\*y;

return ret;

}

inline int Inval(int a,int b,int n){

int x,y,e;

EXT\_GCD(a,n,x,y);

e=(LL)x\*b%n;

return e<0?e+n:e;

}

/\*

\*功能: 二分计算a^b mod c

\*注意: 无法处理c\*c溢出可是c+c不溢出的情况

\*传入: a,b,c(a,b可以小于,可是c不能小于等于,否则返回无解)

\*返回: a^b mod c的值,-1表示不存在

\*/

template<class T,class Tb> inline T PowMod(Tb a,T b,T c)

{

if(c==1)return 0;

if(c<=0)return -1;

a=MOD(a,(Tb)c);

Tb ret(1L%c);

while(b){if(b&0x1)ret=ret\*a%c;a=a\*a%c;b>>=1;}

return (T)ret;

}

int BabyStep(int A,int B,int C)

{

map<int,int> Hash;

LL buf=1%C,D=buf,K;

int i,d=0,tmp;

for(i=0;i<=100;buf=buf\*A%C,++i)if(buf==B)return i;

while((tmp=GCD(A,C))!=1)

{

if(B%tmp)return -1;

++d;

C/=tmp;

B/=tmp;

D=D\*A/tmp%C;

}

Hash.clear();

int M=(int)sqrt(C+0.0);

for(buf=1%C,i=0;i<=M;buf=buf\*A%C,++i)if(Hash.find((int)buf)==Hash.end())Hash[(int)buf]=i;

for(i=0,K=PowMod((LL)A,M,C);i<=M;D=D\*K%C,++i)

{

tmp=Inval((int)D,B,C);

if(tmp>0&&Hash.find(tmp)!=Hash.end())return i\*M+Hash[tmp]+d;

}

return -1;

}

/\*

\*功能: 返回一个数的欧拉函数

\*注意: 不能处理Long Long

\*传入: n(n<=Plimit\*Plimit)

\*返回: n的欧拉函数

\*/

int Phi(int n)

{

int ret(n),i;

if(!IsPlistCreated)CreatPlist(Plimit);

for(i=0;i<Plen&&n>=Plist[i];++i)

if(n%Plist[i]==0)

{

ret-=ret/Plist[i];

while(n%Plist[i]==0)n/=Plist[i];

}

if(n!=1)ret-=ret/n;

return ret;

}

/\*

\*功能: 将n转化为b进制

\*传入: n,b (|b|>=2,b可以为负)

\*返回: Ans[] -答案,Ans[0]保存最高位,以此类推. len -长

\*/

template<class T>inline void SwapVec(T a[],int len)

{int i;for(i=0;i<len/2;++i)SWAP(a[i],a[len-1-i]);}

template<class T> inline void ToBase(T n,int b,T Ans[],int &len)

{

if(n==0){Ans[0]=0;len=1;return;}

len=0;

if(b>0)

while(n!=0)

{

Ans[len++]=n%b;

n/=b;

}

else

while(n!=0)

{

Ans[len]=n-DIV(n,b)\*b;

n=DIV(n,b);

++len;

}

SwapVec(Ans,len);

return;

}

/\*

\*功能: 求解x^2 - n\* y^2 = 1的最小整数解

\*注意: 如果n为完全平方数,则返回一组平凡解(x=0,y=1)

\*否则返回一组最小正解x,y(x,y>0)

\*注意如果n比较大(几百)那么x,y可能需要高精度

\*返回: 最小整数解x,y

\*/

template<class T>inline int Pell(T n,T& x,T& y)

{

T aa=(T)sqrt((double)n),a=aa;

x=1;

y=0;

if(aa\*aa==n)return 0;

T p1=1,p2=0,q1=0,q2=1,g=0,h=1;

while(true)

{

g=-g+a\*h;

h=(n-g\*g)/h;

x=a\*p1+p2;

y=a\*q1+q2;

if(x\*x-n\*y\*y==1)

return 1;

p2=p1;

q2=q1;

p1=x;

q1=y;

a=(g+aa)/h;

}

return 1;

}

# Splay 树模板

#include <iostream>

using namespace std;

typedef \_\_int64 lld;

class Splaytree

{

public:

struct Node

{

lld key;

int data;

Node \*left,\*right,\*father;

Node(Node\* fa){key=0;father=fa;left=right=NULL;}

Node(){key=0;father=left=right=NULL;}

};

Node\* root;

Splaytree(){root=NULL;} //////////////////////////////////////////////////注意以下&root必须全部是实树指针 不能是临时指针

void splay(Node\* &root,Node\* x);//将x提到根

Node\* splayfind(lld keyf);

//查找第一个key不小于keyf的节点

void splayzig(Node\* &root,Node\* x);//右旋

void splayzag(Node\* &root,Node\* x);//左旋

void splaydelete(Node\* &root,Node\* x);

//将x转到root 干掉

Node\* splayMax(Node\* &root);//将最大的转到根

Node\* splayinsert(lld key);

//插入key 返回那个点 已经存在就直接返回

};

void Splaytree::splayzig(Node\* &root,Node\* x)//对x做左旋 将x旋下去 x的左儿子旋上来

{

if (x==NULL) return ;

Node\* fa=x->father;//保存x的父亲

Node\* y=x->left; //x的左儿子

if (y==NULL) return ;//如果y是空 没法旋

x->left=y->right; //y的右给x的左

if (y->right!=NULL) y->right->father=x;//修改y右的父亲

y->right=x; //x接到y右

x->father=y;//修改x父亲

if (x==root) root=y; else//判断根是否改变

if (fa->left==x) fa->left=y;else fa->right=y;//修改父亲指针

y->father=fa;//修改y父亲

}

void Splaytree::splayzag(Node\* &root,Node\* x)

{

if (x==NULL) return;

Node\* fa=x->father;

Node\* y=x->right;

if (y==NULL) return ;

x->right=y->left;

if (y->left!=NULL) y->left->father=x;

y->left=x;

x->father=y;

if (x==root) root=y; else

if (fa->left==x) fa->left=y;else fa->right=y;

y->father=fa;

}

void Splaytree::splay(Node\* &root,Node\* x)//将x旋到根

{

if (x==NULL) return ;

while (1)

{

if (x==root) return ;//如果x就是根 停了

if (x->father==root) //如果x的父亲是根

{

if (root->left==x) splayzig(root,root);

else splayzag(root,root);

return ;

}

Node \*fa=x->father;//父亲

Node\* fa2=fa->father;//父亲的父亲

if (fa2->left==fa&&fa->left==x)//一条线 先转fa2

{

splayzig(root,fa2);

splayzig(root,fa);

}

else if (fa2->right==fa&&fa->right==x)

{

splayzag(root,fa2);

splayzag(root,fa);

}

else if (fa2->left==fa&&fa->right==x)//叉 先转fa

{

splayzag(root,fa);

splayzig(root,fa2);

}

else

{

splayzig(root,fa);

splayzag(root,fa2);

}

}

}

Splaytree::Node\* Splaytree::splayfind(lld keyf)//查找不小于keyf的第一个

{

Node \*x=root,\*last=NULL;//保存最靠近x且不小于x的

while (x!=NULL)

{

if (keyf==x->key) { last=x;break;}//刚好找到

if (keyf<x->key) //如果在左子树 那么x可能成为结果 用last保留

{

last=x;

x=x->left;

}

else

x=x->right; //在右子树 x没用 不用保留

}

x=last;

splay(root,x); //将x转到根

return x;

}

Splaytree::Node\* Splaytree::splayinsert(lld key)//插入key的节点

{

if (root==NULL) //空树直接建

{

root=new Node;

root->key=key;

return root;

}

Node\* cut=splayfind(key); //先找到比key大的

if (cut==NULL) //如果全小于key

{

Node\* p=new Node;

p->key=key;

p->left=root; //root接p左

root->father=p; //修改父指针

root=p;

return root;

}else //cut有东西 这时候cut相当于root

if (cut->key==key) return cut;

//已经存在的话 不插入返回指针

else //插 cut的左=p左 cut=p右

{

Node\* p=new Node;

p->key=key;

p->left=cut->left; //p左=cut左

if (cut->left!=NULL) cut->left->father=p; //修改cut左父指针

p->right=cut; //p右=cut

cut->father=p; //修改cut父指针

cut->left=NULL; //断开cut左(这个已经是p左了)

root=p; //p成为新根

return p;

}

}

Splaytree::Node\* Splaytree::splayMax(Node\* &root)//将root子树中最大的旋到root

{

Node\* p=root;

if (p==NULL) return NULL;

while (p->right!=NULL) p=p->right;//向右儿子找到底

splay(root,p);//旋之

return p;

}

void Splaytree::splaydelete(Node\* &root,Node\* x)//删除x

{

if (x==NULL) return ;

splay(root,x); //先旋到根

if (x->left==NULL) //如果x左子树为空 直接干

{

root=x->right; //x的右儿子变成根

if (root!=NULL) //断开

root->father=x->father;//接上rootfather

}

else //有左儿子 那么将左儿子的最大的提到根 这时候这个点必然没有右子树 将root的右接上去

{

Node\* p=splayMax(x->left); //提左儿子的最大

p->right=x->right; //接右子树

if (x->right!=NULL) x->right->father=p; //修改父指针

p->father=x->father; //p成为新根

root=p;

}

delete x;

}

////////////////////////////////////////////////////////////////////////////////////////

inline int Min(int a,int b){return a<b?a:b;}

inline int Abs(int a){return a>0?a:-a;}

const int maxn=60000,maxl=99999999;

struct Qujian

{

int l,r;

} a[maxn];

Splaytree tree;

int n,start;

void iin()

{

int i;

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

for (i=n-1;i>=0;i--) scanf("%d%d",&a[i].l,&a[i].r);

}

void dp()

{

int i,j,k;

// a[n].l=-maxl;a[n].r=0;

// a[n+1].l=0;a[n+1].r=maxl;

tree.splayinsert(start);

tree.root->data=0;

// Splaytree::Node \*q=tree.root;

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

{

lld lmin=maxl,rmin=maxl;

Splaytree::Node \*p;

while (1)

{

p=tree.splayfind(a[i].l);

if (p==NULL||p->key>a[i].r) break;

lmin=Min(lmin,Abs(p->key-a[i].l)+p->data);

rmin=Min(rmin,Abs(a[i].r-p->key)+p->data);

tree.splaydelete(tree.root,p);

}

p=tree.splayinsert(a[i].l);

p->data=lmin;

p=tree.splayinsert(a[i].r);

p->data=rmin;

}

//tree.splayfind(0);

//printf("%d\n",tree.root->data);

lld res=maxl;

int tot=0;

while (tree.root !=NULL)

{

tot++;

res=Min(res,tree.root->data+Abs(tree.root->key));

tree.splaydelete(tree.root,tree.root);

}

//cout<<tot<<endl;

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

}