inline char getc(){  
static const int BUFLEN = 1 << 15;  
static char B[BUFLEN], \*S = B, \*T = B;  
if(S==T) S=B, T=S+fread(B, 1, BUFLEN, stdin);  
return (S==T) ? 0 : \*(S ++);  
}  
int ReadInt(){  
char ch; int aa=0;  
do ch = getc(); while(!isdigit(ch));  
do aa = aa\*10+ch-'0', ch = getc();  
while(isdigit(ch));  
return aa;  
}

//cin加速(不可与scanf,printf混用)

std::ios::sync\_with\_stdio(0);

std::cin.tie(0);

二分

double binarysearch(double l,double r){

if(r-l<0.0000001)

return r;

double mid = (l+r)/2;

if(mid< )

return binarysearch(mid,r);

return binarysearch(l,mid);

}

三分

double trisectsearch(double l,double r){

if(r-l<0.000001)

return r;

double m = (l+r)/2,mm=(m+r)/2;

if(f(m)>=f(mm))

return trisectsearch(l,mm);

return trisectsearch(m,r);

}

二分求等比数列

using namespace std;

const int M = 1000000007;

typedef long long LL;

LL power(LL a,LL b)

{

LL ans = 1;

a %= M;

while(b)

{

if(b & 1)

{

ans = ans \* a % M;

b--;

}

b >>= 1;

a = a \* a % M;

}

return ans;

}

LL sum(LL a,LL n)

{

if(n == 1) return a;

LL t = sum(a,n/2);

if(n & 1)

{

LL cur = power(a,n/2+1);

t = (t + t \* cur % M) % M;

t = (t + cur) % M;

}

else

{

LL cur = power(a,n/2);

t = (t + t \* cur % M) % M;

}

return t;

}

int main()

{

LL a,n;

while(cin>>a>>n)

cout<<sum(a,n)<<endl;

return 0;

}

01背包

初始值 dp[0] = 0;

memset(dp,0,sizeof(dp));

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

{

for(k=V;k>=w[i];k--)

dp[k] = maxn(dp[k],dp[k-w[i]]+c[i]);

}

完全背包

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

{

for(k=w[i];k<=v[k++])

dp[k] = maxn(dp[k],dp[k-w[i]]+c[i]);

}

二维费用背包

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

{

for(j=A;j>=a[i];j--)

{

for(k=B;k>=b[i];k--)

dp[j][k] = maxn(dp[j][k],dp[j-a[i]][k-b[i]]+c[i]);

}

}

分组背包

for 所有的组k

for v=V..0

for 所有的i属于组k

f[v]=maxn(f[v],f[v-c[i]]+w[i])

LCS

for(i=1;i<=len1;i++)

for(j=1;j<=len2;j++)

if(s1[i-1]==s2[j-1])

{

dp[i][j]=dp[i-1][j-1]+1;

}

else

{

dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

}

//线段树

#define lson l,m,x<<1

#define rson m+1,r,x<<1|1

//建树

void BuildTree(int l,int r,int x)

{

if(l==r)

{

t[x]=a[l];

return;

}

int m=(l+r)/2;

BuildTree(lson);

BuildTree(rson);

t[x]=t[x<<1]+t[x<<1|1];

}

//区间值增加

void PushDown(int l,int r,int x)

{

int m=(l+r)/2;

if(lazy[x]!=0)

{

t[x<<1]+=(lazy[x]\*(m-l+1));

t[x<<1|1]+=(lazy[x]\*(r-m));

lazy[x<<1|1]+=lazy[x];

lazy[x<<1]+=lazy[x];

lazy[x]=0;

}

}

void SegModify(int L,int R,int val,int l,int r,int x){

if(l==L&&r==R)

{

t[x] += ((R-L+1)\*val);

lazy[x] += val;

return;

}

PushDown(l,r,x);

int m=(l+r)/2;

if(R<=m) SegModify(L,R,val,lson);

else if(L>m) SegModify(L,R,val,rson);

else{

SegModify(L,m,val,lson);

SegModify(m+1,R,val,rson);

}

t[x]=t[x<<1]+t[x<<1|1];

}

//区间更改

void PushDown(int l,int r,int x)

{

int m=(l+r)/2;

if(lazy[x]!=0)

{

t[x<<1]=(lazy[x]\*(m-l+1));

t[x<<1|1]=(lazy[x]\*(r-m));

lazy[x<<1|1]=lazy[x];

lazy[x<<1]=lazy[x];

lazy[x]=0;

}

}

void SegModify(int L,int R,int val,int l,int r,int x){

if(l==L&&r==R)

{

t[x] = ((R-L+1)\*val);

lazy[x] = val;

return;

}

PushDown(l,r,x);//要用到下面的了，先用lazy数组更新

int m=(l+r)/2;

if(R<=m) SegModify(L,R,val,lson);

else if(L>m) SegModify(L,R,val,rson);

else{

SegModify(L,m,val,lson);

SegModify(m+1,R,val,rson);

}

t[x]=t[x<<1]+t[x<<1|1];

}

//查询

int Query(int L,int R,int l,int r,int x)

{

if(L==l&&R==r)

{

return t[x];

}

PushDown(l,r,x);

int m=(l+r)/2;

if(R<=m) return Query(L,R,lson);

if(L>m) return Query(L,R,rson);

return Query(L,m,lson)+Query(m+1,R,rson);

}

//二维线段树（四分法）

//求最值

const int maxn = 1000\*1000\*1.5;

//2D line tree

struct node

{

short x1,y1,x2,y2; //(a,b)--(c,d)

float max;

int \*ch;

};

node tree[maxn]; //空间:O(N\*M \* 2) //实际上,1.4\*N\*M就足够了

int tol;

void maketree(int x1,int y1,int x2,int y2) //时间:O(2\*N\*M) 空间:O(2\*N\*M)

{

tol++;

int k = tol;

tree[k].x1 = x1;

tree[k].y1 = y1;

tree[k].x2 = x2;

tree[k].y2 = y2;

tree[k].max = 0.0;

if(x1==x2 && y1 == y2) //c==d

{

tree[k].ch = 0;

return;

}

tree[k].ch = new int[4];

int midx = (x1+x2)/2;

int midy = (y1+y2)/2;

if(x1<=midx && y1 <= midy) //左下

{

tree[k].ch[0] = tol+1;

maketree(x1,y1,midx,midy);

}

else

tree[k].ch[0] = 0;

if(midx+1 <= x2 && midy+1 <= y2) //右上

{

tree[k].ch[1] = tol+1;

maketree(midx+1,midy+1,x2,y2);

}

else

tree[k].ch[1] = 0;

if(x1 <= midx && midy+1 <= y2) //左上

{

tree[k].ch[2] = tol+1;

maketree(x1,midy+1,midx,y2);

}

else

tree[k].ch[2] = 0;

if(midx+1 <= x2 && y1 <= midy) //右下

{

tree[k].ch[3] = tol+1;

maketree(midx+1,y1,x2,midy);

}

else

tree[k].ch[3] = 0;

}

//时间:O(logN)//N\*N的矩形

void insert(int x,int y,int L,int k) //这里,针对插入的是一个矩形[x,y]--[x,y]..一个单位正方形

{

if(tree[k].x1 == tree[k].x2 && tree[k].y1 == tree[k].y2) //叶子结点 //

{

if(L > tree[k].max)

tree[k].max = L;

return;

}

int midx = (tree[k].x1+tree[k].x2)/2;

int midy = (tree[k].y1+tree[k].y2)/2;

if(x<=midx)

{

if(y<=midy)

insert(x,y,L,tree[k].ch[0]);

else

insert(x,y,L,tree[k].ch[2]);

}

else //x>midx

{

if(y<=midy)

insert(x,y,L,tree[k].ch[3]);

else

insert(x,y,L,tree[k].ch[1]);

}

float m = tree[tree[k].ch[0]].max;

for(int i=1; i<4; i++)

m = max(m,tree[tree[k].ch[i]].max);

tree[k].max = m;

}

inline bool corss(int x1,int y1,int x2,int y2,int k) //判断两矩形是否相交

{

int x3 = tree[k].x1;

int y3 = tree[k].y1;

int x4 = tree[k].x2;

int y4 = tree[k].y2;

if(y2 < y3 || y4 < y1 || x4 < x1 || x2 < x3)

return false;

else

return true;

}

//时间:O(logN)//N\*N的矩形

float Query(int x1,int y1,int x2,int y2,int k)

{

if(corss(x1,y1,x2,y2,k) == false || tree[k].max == 0) //矩形不相交或 矩形内max==0 则直接返回0

return 0;

if(x1<=tree[k].x1 && y1<=tree[k].y1 && tree[k].x2 <= x2 && tree[k].y2 <= y2) //如果要查询的矩形覆盖了当前矩形..则返回当前矩形的max值

return tree[k].max;

int midx = (tree[k].x1+tree[k].x2)/2;

int midy = (tree[k].y1+tree[k].y2)/2;

float m[4] = {0,0,0,0};

for(int i=0; i<4; i++)

m[i] = Query(x1,y1,x2,y2,tree[k].ch[i]);

float mm = m[0];

for(int i=1; i<4; i++)

mm = max(mm,m[i]);

return mm;

}

//求和

const int maxn = 1000\*1000\*1.5;

//2D line tree

struct node

{

short x1,y1,x2,y2; //(a,b)--(c,d)

int max;

int ch[4];

};

node tree[maxn]; //空间:O(N\*M \* 2) //实际上,1.4\*N\*M就足够了

int tol;

void maketree(int x1,int y1,int x2,int y2) //时间:O(2\*N\*M) 空间:O(2\*N\*M)

{

tol++;

int k = tol;

tree[k].x1 = x1;

tree[k].y1 = y1;

tree[k].x2 = x2;

tree[k].y2 = y2;

tree[k].max = 0;

if(x1==x2 && y1 == y2) //c==d

{

return;

}

int midx = (x1+x2)/2;

int midy = (y1+y2)/2;

//printf("%d %d %d %d %d %d %d\n",x1,y1,x2,y2,tol,midx,midy);

if(x1<=midx && y1 <= midy) //左下

{

tree[k].ch[0] = tol+1;

maketree(x1,y1,midx,midy);

}

else

tree[k].ch[0] = 0;

if(midx+1 <= x2 && midy+1 <= y2) //右上

{

tree[k].ch[1] = tol+1;

maketree(midx+1,midy+1,x2,y2);

}

else

tree[k].ch[1] = 0;

if(x1 <= midx && midy+1 <= y2) //左上

{

tree[k].ch[2] = tol+1;

maketree(x1,midy+1,midx,y2);

}

else

tree[k].ch[2] = 0;

if(midx+1 <= x2 && y1 <= midy) //右下

{

tree[k].ch[3] = tol+1;

maketree(midx+1,y1,x2,midy);

}

else

tree[k].ch[3] = 0;

}

//时间:O(logN)//N\*N的矩形

void insert(int x,int y,int L,int k) //这里,针对插入的是一个矩形[x,y]--[x,y]..一个单位正方形

{

//printf("%d %d %d %d %d %d %d %d\n",tree[k].x1,tree[k].y1,tree[k].x2,tree[k].y2,x,y,L,k);

if(tree[k].x1 == tree[k].x2 && tree[k].y1 == tree[k].y2) //叶子结点 //

{

tree[k].max = L;

return;

}

int midx = (tree[k].x1+tree[k].x2)/2;

int midy = (tree[k].y1+tree[k].y2)/2;

if(x<=midx)

{

if(y<=midy)

insert(x,y,L,tree[k].ch[0]);

else

insert(x,y,L,tree[k].ch[2]);

}

else //x>midx

{

if(y<=midy)

insert(x,y,L,tree[k].ch[3]);

else

insert(x,y,L,tree[k].ch[1]);

}

int m = tree[tree[k].ch[0]].max;

for(int i=1; i<4; i++)

m += tree[tree[k].ch[i]].max;

tree[k].max = m;

}

inline bool corss(int x1,int y1,int x2,int y2,int k) //判断两矩形是否相交

{

int x3 = tree[k].x1;

int y3 = tree[k].y1;

int x4 = tree[k].x2;

int y4 = tree[k].y2;

if(y2 < y3 || y4 < y1 || x4 < x1 || x2 < x3)

return false;

else

return true;

}

//时间:O(logN)//N\*N的矩形

int Query(int x1,int y1,int x2,int y2,int k)

{

if(corss(x1,y1,x2,y2,k) == false || tree[k].max == 0) //矩形不相交或 矩形内max==0 则直接返回0

return 0;

if(x1==tree[k].x1 && y1==tree[k].y1 && tree[k].x2 == x2 && tree[k].y2 == y2){ //如果要查询的矩形为当前矩形..则返回当前矩形的max值

//printf("%d %d %d %d %d %d\n",tree[k].max,x1,x2,y1,y2,k);

return tree[k].max;

}

int midx = (tree[k].x1+tree[k].x2)/2;

int midy = (tree[k].y1+tree[k].y2)/2;

int mm = 0;

if(x1<=midx && y1 <= midy) //左下

{

mm += Query(x1,y1,min(midx,x2),min(midy,y2),tree[k].ch[0]);

}

if(midx+1 <= x2 && midy+1 <= y2) //右上

{

mm += Query(max(midx+1,x1),max(midy+1,y1),x2,y2,tree[k].ch[1]);

}

if(x1 <= midx && midy+1 <= y2) //左上

{

mm += Query(x1,max(midy+1,y1),min(midx,x2),y2,tree[k].ch[2]);

}

if(midx+1 <= x2 && y1 <= midy) //右下

{

mm += Query(max(midx+1,x1),y1,x2,min(midy,y2),tree[k].ch[3]);

}

return mm;

}

并查集与最小生成树

int father[N];

struct graph

{int u,v,cost;

void set(int a,int b,int w)

{u=a;v=b;cost=w;}};

graph d[N\*(N-1)/2];

int find(int x)

{

if(father[x]==-1) return x;

return father[x]=find(father[x]);

}

bool Union(int x,int y){

x=find(x);

y=find(y);

if(x==y) return false;

if(x>y) father[x]=y;

if(x<y) father[y]=x;

return true;

}

Kruskal

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

{

if(Union(d[i].u,d[i].v))

{

++ct;

sum+=d[i].cost;

}

if(ct==n-1) break;

}

食物链

int find(int x)

{

if(father[x][0]==-1) return x;

int fa=father[x][0];

father[x][0]=find(father[x][0]);

father[x][1]=(father[x][1]+father[fa][1])%3;

return father[x][0];

}

bool Union(int x,int y){

int xx,yy;

xx=find(x);

yy=find(y);

if(xx==yy) return false;

father[xx][0]=y;

father[xx][1]=(father[xx][1]+1-father[x][1]+2)%2;

return true;

}

最短路

Floyd

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

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

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

if(floyd[i][j]>floyd[i][k]+floyd[k][j])

floyd[i][j]=floyd[i][k]+floyd[k][j];}

dijkstra

void djs(int a)

{

int j,k,u;u=0;

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

{

dis[j]=map[a][j];

ss[j]=s[a][j];

}

dis[a]=0;

ss[a]=0;

vis[a]=1;

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

{

int tmp;tmp=168430090;

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

if(!vis[k]&&dis[k]<tmp)

{

u=k;

tmp=dis[u];

}

vis[u]=1;

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

{

if(!vis[k]&&dis[k]>map[u][k]+dis[u])

{

dis[k]=dis[u]+map[u][k];

ss[k]=ss[u]+s[u][k];

}

if(!vis[k]&&dis[k]==map[u][k]+dis[u]&&ss[k]>ss[u]+s[u][k])

ss[k]=ss[u]+s[u][k];

}

}

}

/\*

Dijkstra的算法思想：

在所有没有访问过的结点中选出dis(s,x)值最小的x

对从x出发的所有边(x,y),更新

dis(s,y)=min(dis(s,y),dis(s,x)+dis(x,y))

\*/

#include <iostream>

#include <cstdio>

#include <queue>

#include <vector>

using namespace std;

const int Ni = 10000;

const int INF = 1<<27;

struct node{

int x,d;

node(){}

node(int a,int b){x=a;d=b;}

bool operator < (const node & a) const

{

if(d==a.d) return x<a.x;

else return d > a.d;

}

};

vector<node> eg[Ni];

int dis[Ni],n;

void Dijkstra(int s)

{

int i;

for(i=0;i<=n;i++) dis[i]=INF;

dis[s]=0;

//用优先队列优化

priority\_queue<node> q;

q.push(node(s,dis[s]));

while(!q.empty())

{

node x=q.top();q.pop();

for(i=0;i<eg[x.x].size();i++)

{

node y=eg[x.x][i];

if(dis[y.x]>x.d+y.d)

{

dis[y.x]=x.d+y.d;

q.push(node(y.x,dis[y.x]));

}

}

}

}

int main()

{

int a,b,d,m;

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

{

for(int i=0;i<=n;i++) eg[i].clear();

while(m--)

{

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

eg[a].push\_back(node(b,d));

eg[b].push\_back(node(a,d));

}

Dijkstra(1);

printf("%d\n",dis[n]);

}

return 0;

}

spfa

void spfa(){

int i,now;

memset(visit,false,sizeof(visit));

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

dist[i]=INF;

dist[1] = 0;

queue<int>Q;

Q.push(1);

visit[1] = true;

while(!Q.empty()){

now = Q.front();

Q.pop();

visit[now] = false;

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

if(dist[i] > dist[now] + map[now][i]){

dist[i] = dist[now] + map[now][i];

if(visit[i] == 0){

Q.push(i);

visit[i] = true;

}

}

}

}

BFS

void bfs()

{

queue<int>q;

int now,next;

now=a;

q.push(now);

while(!q.empty())

{

now=q.front();q.pop();//printf("%d %d\n",now,s[now]);

if(){

q.push();

}

}

}

sort

#include <algorithm>

using namespace std;

sort(a,a+n,cmp1);

数排序

bool cmp1(int x,int y)

{

return x<y; //升序排列，如果改为return a>b，则为降序

}

结构体排序

bool cmp2(int x,int y)

{

return x.v<y.v; //升序排列，如果改为return a>b，则为降序

}

结构体字符串排序//#include <cstring>

bool compare\_struct\_str(const In &a,const In &b){

return string(a.str)<string(b.str);

}

数组字符串排序

bool cmp(const char \*a, const char \*b)

{

if (strcmp(a, b) < 0)

return true;

return false;

}

拓展欧几里得

int gcd(int a,int b,int &x,int &y){

if (b==0){

x=1,y=0;

return a;

}

int q=gcd(b,a%b,y,x);

y-=a/b\*x;

return q;

}

逆元

1到n的逆元

inv[1] = 1;

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

{

if(i >= mod) break;

inv[i] = (mod - mod / i) \* inv[mod % i] % mod;

}

1到n！的逆元(组合数模质数)

int \_inv(int x) {

if(x == 1) return 1;

return LL(MOD - MOD / x) \* \_inv(MOD % x) % MOD;

}

void init(int n = 100000) {

fact[0] = 1;

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

fact[i] = fact[i - 1] \* LL(i) % MOD;

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

inv[i] = \_inv(fact[i]);

}

LL comb(int a, int b) {

if(a < b) return 0;

return LL(fact[a]) \* inv[b] % MOD \* LL(inv[a - b]) % MOD;

}

LL Lucas(int n, int m,) {

LL ans = 1;

while(n&&m&&ans) {

ans = (ans\*comb(n%p, m%p, p)) % p;

n /= p;

m /= p;

}

return ans;

}

孙子定理

while(scanf("%I64d",&n)!=EOF){

scanf("%I64d%I64d",&a,&b);

flag = 0;

while(--n){

scanf("%I64d%I64d",&aa,&bb);

c = (bb-b);

d = gcd(a,aa,x,y);

if(c%d)

flag = 1;

if(flag)

continue;

x\*=c/d;

aa/=d;

x = (x%aa+aa)%aa;

b = b+x\*a;

a\*=aa;

//printf("%I64d %I64d\n",a,b);

}

if(flag)

printf("-1\n");

else

printf("%I64d\n",b);

}

//直接求解欧拉函数

int euler(int n){ //返回euler(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;

}

//筛选法打欧拉函数表

#define Max 1000001

int euler[Max];

void Init(){

euler[1]=1;

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

euler[i]=i;

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

if(euler[i]==i)

for(int j=i;j<Max;j+=i)

euler[j]=euler[j]/i\*(i-1);//先进行除法是为了防止中间数据的溢出

}

TSP

for (int i=0 ; i<(2<<n) ; ++i)

{

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

{

dp[1<<j][j]=map[0][j];

//if(i==1<<j)dp[i][j]=map[0][j];这个和上面的语句都能AC

if(i&(1<<j))

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

{

if((i-(1<<j)) & (1<<k))//第k位为1，第j位不为1的状态

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

}

//printf("%d %d %d\n",i,j,dp[i][j]);

}

}

int ans=dp[(2<<n)-1][0];

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

{

ans=min(dp[(2<<n)-1][i]+map[i][0],ans);

}

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

//矩阵快速幂呜呜呜呜~~~

网络流

const int N = 1000;

const int INF = 100000000;

struct node {

int u, v, next, cap;

} edge[N\*N];

int next[N], head[N], layer[N], Q[N \* 2], mark[N];

int ecnt;

void init(){

ecnt= 0;

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

}

void add(int u, int v, int c) {

edge[ecnt].u = u;

edge[ecnt].v = v;

edge[ecnt].cap = c;

edge[ecnt].next = head[u];

head[u] = ecnt++;

edge[ecnt].u = v;

edge[ecnt].v = u;

edge[ecnt].cap = 0;

edge[ecnt].next = head[v];

head[v] = ecnt++;

}

bool BFS(int begin, int end) {

int i, l, h, k, y;

for (i = 0; i <= end; i++) layer[i] = -1;

layer[begin] = 0;

l = h = 0;

Q[l++] = begin;

while (h < l) {

k = Q[h++];

for (i = head[k]; i != -1; i = edge[i].next) {

y = edge[i].v;

if (edge[i].cap > 0 && layer[y] == -1) {

layer[y] = layer[k] + 1;

if (y == end)

return true;

Q[l++] = y;

}

}

}

return false;

}

int DFS(int x, int exp, int end) {

mark[x] = 1;

if (x == end)return exp;

int y, temp, i;

for (i = next[x]; i != -1; i = edge[i].next, next[x] = i) {

y = edge[i].v;

if (edge[i].cap > 0 && layer[y] == layer[x] + 1 && !mark[y]) {

if ((temp = (DFS(y, min(exp, edge[i].cap), end))) > 0) {

edge[i].cap -= temp;//流完后正向流表示剩余流量

edge[i^1].cap += temp;//流完后反向流表示正向流的流量

return temp;

}

}

}

return 0;

}

int Dinic\_flow(int begin, int end) {

int i, ans = 0, flow;

while (BFS(begin, end)) {

for (i = 0; i <= end; i++)next[i] = head[i];

while (true) {

for (i = 0; i <= end; i++) mark[i] = 0;

flow = DFS(begin, INF, end);

if (flow == 0)break;

ans += flow;

}

}

return ans;

}

//高精度

#define MAX\_L 2005 //最大长度，可以修改

class bign

{

public:

int len, s[MAX\_L];//数的长度，记录数组

//构造函数

bign();

bign(const char\*);

bign(int);

bool sign;//符号 1正数 0负数

string toStr() const;//转化为字符串，主要是便于输出

friend istream& operator>>(istream &,bign &);//重载输入流

friend ostream& operator<<(ostream &,bign &);//重载输出流

//重载复制

bign operator=(const char\*);

bign operator=(int);

bign operator=(const string);

//重载各种比较

bool operator>(const bign &) const;

bool operator>=(const bign &) const;

bool operator<(const bign &) const;

bool operator<=(const bign &) const;

bool operator==(const bign &) const;

bool operator!=(const bign &) const;

//重载四则运算

bign operator+(const bign &) const;

bign operator++();

bign operator++(int);

bign operator+=(const bign&);

bign operator-(const bign &) const;

bign operator--();

bign operator--(int);

bign operator-=(const bign&);

bign operator\*(const bign &)const;

bign operator\*(const int num)const;

bign operator\*=(const bign&);

bign operator/(const bign&)const;

bign operator/=(const bign&);

//四则运算的衍生运算

bign operator%(const bign&)const;//取模（余数）

bign factorial()const;//阶乘

bign Sqrt()const;//整数开根（向下取整）

bign pow(const bign&)const;//次方

//一些乱乱的函数

void clean();

~bign();

};

#define max(a,b) a>b ? a : b

#define min(a,b) a<b ? a : b

bign::bign()

{

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

len = 1;

sign = 1;

}

bign::bign(const char \*num)

{

\*this = num;

}

bign::bign(int num)

{

\*this = num;

}

string bign::toStr() const

{

string res;

res = "";

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

res = (char)(s[i] + '0') + res;

if (res == "")

res = "0";

if (!sign&&res != "0")

res = "-" + res;

return res;

}

istream &operator>>(istream &in, bign &num)

{

string str;

in>>str;

num=str;

return in;

}

ostream &operator<<(ostream &out, bign &num)

{

out<<num.toStr();

return out;

}

bign bign::operator=(const char \*num)

{

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

char a[MAX\_L] = "";

if (num[0] != '-')

strcpy(a, num);

else

for (int i = 1; i < strlen(num); i++)

a[i - 1] = num[i];

sign = !(num[0] == '-');

len = strlen(a);

for (int i = 0; i < strlen(a); i++)

s[i] = a[len - i - 1] - 48;

return \*this;

}

bign bign::operator=(int num)

{

if (num < 0)

sign = 0, num = -num;

else

sign = 1;

char temp[MAX\_L];

sprintf(temp, "%d", num);

\*this = temp;

return \*this;

}

bign bign::operator=(const string num)

{

const char \*tmp;

tmp = num.c\_str();

\*this = tmp;

return \*this;

}

bool bign::operator<(const bign &num) const

{

if (sign^num.sign)

return num.sign;

if (len != num.len)

return len < num.len;

for (int i = len - 1; i >= 0; i--)

if (s[i] != num.s[i])

return sign ? (s[i] < num.s[i]) : (!(s[i] < num.s[i]));

return !sign;

}

bool bign::operator>(const bign&num)const

{

return num < \*this;

}

bool bign::operator<=(const bign&num)const

{

return !(\*this>num);

}

bool bign::operator>=(const bign&num)const

{

return !(\*this<num);

}

bool bign::operator!=(const bign&num)const

{

return \*this > num || \*this < num;

}

bool bign::operator==(const bign&num)const

{

return !(num != \*this);

}

bign bign::operator+(const bign &num) const

{

if (sign^num.sign)

{

bign tmp = sign ? num : \*this;

tmp.sign = 1;

return sign ? \*this - tmp : num - tmp;

}

bign result;

result.len = 0;

int temp = 0;

for (int i = 0; temp || i < (max(len, num.len)); i++)

{

int t = s[i] + num.s[i] + temp;

result.s[result.len++] = t % 10;

temp = t / 10;

}

result.sign = sign;

return result;

}

bign bign::operator++()

{

\*this = \*this + 1;

return \*this;

}

bign bign::operator++(int)

{

bign old = \*this;

++(\*this);

return old;

}

bign bign::operator+=(const bign &num)

{

\*this = \*this + num;

return \*this;

}

bign bign::operator-(const bign &num) const

{

bign b=num,a=\*this;

if (!num.sign && !sign)

{

b.sign=1;

a.sign=1;

return b-a;

}

if (!b.sign)

{

b.sign=1;

return a+b;

}

if (!a.sign)

{

a.sign=1;

b=bign(0)-(a+b);

return b;

}

if (a<b)

{

bign c=(b-a);

c.sign=false;

return c;

}

bign result;

result.len = 0;

for (int i = 0, g = 0; i < a.len; i++)

{

int x = a.s[i] - g;

if (i < b.len) x -= b.s[i];

if (x >= 0) g = 0;

else

{

g = 1;

x += 10;

}

result.s[result.len++] = x;

}

result.clean();

return result;

}

bign bign::operator \* (const bign &num)const

{

bign result;

result.len = len + num.len;

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

for (int j = 0; j < num.len; j++)

result.s[i + j] += s[i] \* num.s[j];

for (int i = 0; i < result.len; i++)

{

result.s[i + 1] += result.s[i] / 10;

result.s[i] %= 10;

}

result.clean();

result.sign = !(sign^num.sign);

return result;

}

bign bign::operator\*(const int num)const

{

bign x = num;

bign z = \*this;

return x\*z;

}

bign bign::operator\*=(const bign&num)

{

\*this = \*this \* num;

return \*this;

}

bign bign::operator /(const bign&num)const

{

bign ans;

ans.len = len - num.len + 1;

if (ans.len < 0)

{

ans.len = 1;

return ans;

}

bign divisor = \*this, divid = num;

divisor.sign = divid.sign = 1;

int k = ans.len - 1;

int j = len - 1;

while (k >= 0)

{

while (divisor.s[j] == 0) j--;

if (k > j) k = j;

char z[MAX\_L];

memset(z, 0, sizeof(z));

for (int i = j; i >= k; i--)

z[j - i] = divisor.s[i] + '0';

bign dividend = z;

if (dividend < divid) { k--; continue; }

int key = 0;

while (divid\*key <= dividend) key++;

key--;

ans.s[k] = key;

bign temp = divid\*key;

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

temp = temp \* 10;

divisor = divisor - temp;

k--;

}

ans.clean();

ans.sign = !(sign^num.sign);

return ans;

}

bign bign::operator/=(const bign&num)

{

\*this = \*this / num;

return \*this;

}

bign bign::operator%(const bign& num)const

{

bign a = \*this, b = num;

a.sign = b.sign = 1;

bign result, temp = a / b\*b;

result = a - temp;

result.sign = sign;

return result;

}

bign bign::pow(const bign& num)const

{

bign result = 1;

for (bign i = 0; i < num; i++)

result = result\*(\*this);

return result;

}

bign bign::factorial()const

{

bign result = 1;

for (bign i = 1; i <= \*this; i++)

result \*= i;

return result;

}

void bign::clean()

{

if (len == 0) len++;

while (len > 1 && s[len - 1] == '\0')

len--;

}

bign bign::Sqrt()const

{

if(\*this<0)return -1;

if(\*this<=1)return \*this;

bign l=0,r=\*this,mid;

while(r-l>1)

{

mid=(l+r)/2;

if(mid\*mid>\*this)

r=mid;

else

l=mid;

}

return l;

}

bign::~bign()

{

}

bign num0,num1,res;

void prin(bign numm){

for(int i = numm.len-1;i >=0;i--)

putchar(numm.s[i]+'0');

printf("\n");

}

int main()

{

cin>>num0>>num1;

res=num0+num1;

cout<<res<<endl;

num0=5;

num1="71";

res=num0-num1;

cout<<res<<endl;

res=num0.Sqrt();

cout<<res<<endl;

res=num0.pow(5);

cout<<res<<endl;

return 0;

}

//KMP

void getfill(string s)

{

memset(f,0,sizeof(f)); //根据其前一个字母得到

for(int i=1;i<s.size();i++)

{

int j=f[i];

while(j && s[i]!=s[j])

j=f[j];

f[i+1]=(s[i]==s[j])?j+1:0;

}

}

int find(string a,string s)

{

getfill(s);int j=0;

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

{

while(j && a[i]!=s[j])

j=f[j];

if(a[i]==s[j])

j++;

if(j==s.size()){

return i-s.size()+1;

}

}

}

//平面最优点模拟退火(三圆夹角相等)

struct pt {

double x;

double y;

double r;

};

pt mkp(double x, double y) {

pt ret;

ret.x = x;

ret.y = y;

return ret;

}

double dis(pt a, pt b) {

return sqrt((a.x-b.x)\*(a.x-b.x) + (a.y-b.y)\*(a.y-b.y));

}

double cost(pt \*p, double x, double y) {

double ang[3];

for (int i = 0; i < 3; i++)

ang[i] = dis(p[i], mkp(x, y)) / p[i].r;

double diff[3];

for (int i = 0; i < 3; i++)

diff[i] = ang[i] - ang[(i+1)%3];

double ret = 0;

for (int i = 0; i < 3; i++)

ret += diff[i] \* diff[i];

return ret;

}

const int dx[] = {0, 1, -1, 0};

const int dy[] = {1, 0, 0, -1};

const double err = 1e-6;

int main() {

pt p[3];

for (int i = 0; i < 3; i++)

scanf("%lf %lf %lf", &(p[i].x), &(p[i].y), &(p[i].r));

pt ans;

ans.x = (p[0].x + p[1].x + p[2].x) / 3.0;

ans.y = (p[0].y + p[1].y + p[2].y) / 3.0;

double ncost = cost(p, ans.x, ans.y);

pt tmp;

double step = 1.0;

int flag = -1;

for (int i = 0; i < 300000 && ncost > 1e-8; i++) {

flag = -1;

for (int k = 0; k < 4; k++) {

tmp.x = ans.x + step \* ((double)dx[k]);

tmp.y = ans.y + step \* ((double)dy[k]);

if (ncost > cost(p, tmp.x, tmp.y)) {

ncost = cost(p, tmp.x, tmp.y);

flag = k;

}

}

if (flag == -1) step \*= 0.5;

else{

tmp.x = ans.x + step \* ((double)dx[flag]);

tmp.y = ans.y + step \* ((double)dy[flag]);

ans = tmp;

}

}

if (cost(p, ans.x, ans.y) <= err) printf("%.5lf %.5lf\n", ans.x, ans.y);

return 0;

}

//最小覆盖圆

const double pi = acos(-1.0);

int n;

double X,Y;

struct pt {

double x;

double y;

double r;

};

pt p[1010];

double dis(pt a, pt b) {

return sqrt((a.x-b.x)\*(a.x-b.x) + (a.y-b.y)\*(a.y-b.y));

}

double cost(pt ans) {

double ret = 0;

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

ret = max(ret,dis(ans,p[i]));

return ret;

}

const double err = 1e-6;

double Rand(double L, double R) {//Çø¼äÄÚËæ»úÊýÉú³Éº¯Êý

return (rand() % 10000) / 10000.0 \* (R - L) + L;

}

int main() {

srand(time(0));

while(scanf("%lf%lf%d",&X,&Y,&n)!=EOF){

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

scanf("%lf %lf", &(p[i].x), &(p[i].y));

double minx =1e5,miny = 1e5,maxx = 0,maxy = 0;

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

minx = min(minx,p[i].x);

miny = min(miny,p[i].y);

maxx = max(maxx,p[i].x);

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

}

int t = 30;

pt cnt;

double acost = 100000;

while(t--){

pt ans;

ans.x = Rand(minx,maxx);

ans.y = Rand(miny,maxy);

double ncost = cost(ans);

pt tmp;

double step = max(maxx-minx,maxy-miny);

int flag = -1;

for (int i = 0;step > 1e-3; i++) {

flag = -1;

for (int k = 0; k < 40; k++) {

double ang = (rand()%1000+1)/1000.0\*10\*pi;

tmp.x = ans.x + step \* cos(ang);

tmp.y = ans.y + step \* sin(ang);

if (ncost > cost(tmp)) {

ncost = cost(tmp);

flag = 1;

ans = tmp;

}

}

if (flag == -1) step \*= 0.8;

}

if(ncost<acost){

acost = ncost;

cnt = ans;

}

}

printf("(%.1lf,%.1lf).\n", cnt.x, cnt.y);

printf("%.1lf\n", acost);

}

return 0;

}

//rand随机

srand(time(NULL));

double Rand(double L, double R) {//区间内随机数生成函数

return (rand() % 10000) / 10000.0 \* (R - L) + L;

}

母函数

//初始化a，因为有last，所以这里无需初始化其他位

a[0]=1;

int last=0;

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

{

int last2=min(last+n[i]\*v[i],P);//计算下一次的last

memset(b,0,sizeof(int)\*(last2+1));//只清空b[0..last2]

for (int j=n1[i];j<=n2[i]&&j\*v[i]<=last2;j++)//这里是last2

for (int k=0;k<=last&&k+j\*v[i]<=last2;k++)//这里一个是last，一个是last2

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

memcpy(a,b,sizeof(int)\*(last2+1));//b赋值给a，只赋值0..last2

last=last2;//更新last

}