1. Data Structure
   1. Binary Indexed Tree 树状数组

int tree[MAXX];

inline int lowbit(const int &a)

{

return a&-a;

}

inline void update(int pos,const int &val)

{

while(pos<MAXX)

{

tree[pos]+=val;

pos+=lowbit(pos);

}

}

inline int read(int pos)

{

int re(0);

while(pos>0)

{

re+=tree[pos];

pos-=lowbit(pos);

}

return re;

}

int find\_Kth(int k)

{

int now=0;

for (char i=20;i>=0;--i)

{

now|=(1<<i);

if (now>MAXX || tree[now]>=k)

now^=(1<<i);

else k-=tree[now];

}

return now+1;

}

* 1. Dived Tree 划分树

//数组保存在val[0][1],val[0][n]的双闭区间中

template<class Tp>class DT

{

public:

int n;

Tp val[20][MAXX],sorted[MAXX];

inline void make()

{

std::sort(sorted+1,sorted+1+n);

make(1,1,n,0);

}

inline int query(const int &l,const int &r,const int &k)

{

return query(1,1,n,l,r,k,0);

}

private:

int toleft[20][MAXX],mid[MAXX<<2];

// toleft: 有多少过数分到左边去了

void make(const int &id,const int &l,const int &r,const int &d)

{

if(l!=r)

{

mid[id]=(l+r)>>1;

int lsame(mid[id]-l+1),i;

for(i=l;i<=r;++i)

if(val[d][i]<sorted[mid[id]])

--lsame;

int lpos(l),rpos(mid[id]+1),same(0);

for(i=l;i<=r;++i)

{

if(i==l)

toleft[d][i]=0;

else

toleft[d][i]=toleft[d][i-1];

if(val[d][i]<sorted[mid[id]])

{

++toleft[d][i];

val[d+1][lpos++]=val[d][i];

}

else

if(val[d][i]>sorted[mid[id]])

val[d+1][rpos++]=val[d][i];

else

if(same<lsame)

{

++same;

++toleft[d][i];

val[d+1][lpos++]=val[d][i];

}

else

val[d+1][rpos++]=val[d][i];

}

make(id<<1,l,mid[id],d+1);

make(id<<1|1,mid[id]+1,r,d+1);

}

}

int query(const int &id,const int &ll,const int &rr,const int &l,const int &r,const int &k,const int &d)

{

if(l==r)

return val[d][l];

int s,ss;

if(l==ll)

{

s=toleft[d][r];

ss=0;

}

else

{

s=toleft[d][r]-toleft[d][l-1];

ss=toleft[d][l-1];

}

if(s>=k)

{

int newl(ll+ss),newr(ll+ss+s-1);

return query(id<<1,ll,mid[id],newl,newr,k,d+1);

}

int bb(l-ll-ss),b(r-l+1-s),newl(mid[id]+bb+1),newr(mid[id]+bb+b);

return query(id<<1|1,mid[id]+1,rr,newl,newr,k-s,d+1);

}

};

* 1. LCA - tarjan – minmax tarjan离线LCA，带树上区间最值

#include<cstdio>

#include<list>

#include<algorithm>

#include<cstring>

#define MAXX 100111

#define inf 0x5fffffff

short T,t;

int set[MAXX],min[MAXX],max[MAXX],ans[2][MAXX];

bool done[MAXX];

std::list<std::pair<int,int> >edge[MAXX];

std::list<std::pair<int,int> >q[MAXX];

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

struct node

{

int a,b,id;

node() {}

node(const int &aa,const int &bb,const int &idd): a(aa),b(bb),id(idd){}

};

std::list<node>to[MAXX];

int find(const int &a)

{

if(set[a]==a)

return a;

int b(set[a]);

set[a]=find(set[a]);

max[a]=std::max(max[a],max[b]);

min[a]=std::min(min[a],min[b]);

return set[a];

}

void tarjan(const int &now)

{

done[now]=true;

for(std::list<std::pair<int,int> >::const\_iterator it(q[now].begin());it!=q[now].end();++it)

if(done[it->first])

if(it->second>0)

to[find(it->first)].push\_back(node(now,it->first,it->second));

else

to[find(it->first)].push\_back(node(it->first,now,-it->second));

for(std::list<std::pair<int,int> >::const\_iterator it(edge[now].begin());it!=edge[now].end();++it)

if(!done[it->first])

{

tarjan(it->first);

set[it->first]=now;

min[it->first]=it->second;

max[it->first]=it->second;

}

for(std::list<node>::const\_iterator it(to[now].begin());it!=to[now].end();++it)

{

find(it->a);

find(it->b);

ans[0][it->id]=std::min(min[it->b],min[it->a]);

ans[1][it->id]=std::max(max[it->a],max[it->b]);

}

}

int main()

{

scanf("%hd",&T);

for(t=1;t<=T;++t)

{

scanf("%d",&n);

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

{

edge[i].clear();

q[i].clear();

to[i].clear();

done[i]=false;

set[i]=i;

min[i]=inf;

max[i]=0;

}

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

{

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

edge[j].push\_back(std::make\_pair(k,l));

edge[k].push\_back(std::make\_pair(j,l));

}

scanf("%d",&m);

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

{

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

q[j].push\_back(std::make\_pair(k,i));

q[k].push\_back(std::make\_pair(j,-i));

}

tarjan(1);

printf("Case %hd:\n",t);

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

printf("%d %d\n",ans[0][i],ans[1][i]);

}

return 0;

}

* 1. segment tree – discretization 离散化扫描线线段树

std::map<double,short>map; //正向hash

std::map<double,short>::iterator it;

double rmap[inf]; //逆向hash

short mid[MAX],cnt[MAX];

double len[MAX];

void make(const short &id,const short &l,const short &r)

{

mid[id]=(l+r)>>1;

if(l!=r)

{

make(id<<1,l,mid[id]);

make(id<<1|1,mid[id]+1,r);

}

}

void update(const short &id,const short &ll,const short &rr,const short &l,const short &r,const char &val)

{

if(ll==l && rr==r)

{

cnt[id]+=val;

if(cnt[id])

len[id]=rmap[r]-rmap[l-1];

else

if(l!=r)

len[id]=len[id<<1]+len[id<<1|1];

else

len[id]=0;

return;

}

if(mid[id]>=r)

update(id<<1,ll,mid[id],l,r,val);

else

if(mid[id]<l)

update(id<<1|1,mid[id]+1,rr,l,r,val);

else

{

update(id<<1,ll,mid[id],l,mid[id],val);

update(id<<1|1,mid[id]+1,rr,mid[id]+1,r,val);

}

if(!cnt[id])

len[id]=len[id<<1]+len[id<<1|1];

}

int main()

{

n<<=1;

map.clear();

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

{

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

if(x1>x2)

std::swap(x1,x2);

if(y1>y2)

std::swap(y1,y2);

sum+=(x2-x1)\*(y2-y1)\*d;

ln[i].l=x1;

ln[i].r=x2;

ln[i].h=y1;

ln[i].up=1;

ln[++i].l=x1;

ln[i].r=x2;

ln[i].h=y2;

ln[i].up=-1;

map[x1]=1;

map[x2]=1;

}

k=1;

for(it=map.begin();it!=map.end();++it,++k) //离散化

{

it->second=k;

rmap[k]=it->first;

}

std::sort(ln,ln+n);

update(1,1,inf,map[ln[0].l]+1,map[ln[0].r],ln[0].up);

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

{

//依次铺线，len[1]为当前状态总长

//ln[i].h-ln[i-1].h 为两条线直接的高度差

update(1,1,inf,map[ln[i].l]+1,map[ln[i].r],ln[i].up);

}

}

* 1. size-blanced binary search tree

template<class Tp>class sbt

{

public:

inline void init()

{

rt=cnt=l[0]=r[0]=sz[0]=0;

}

inline void ins(const Tp &a)

{

ins(rt,a);

}

inline void del(const Tp &a)

{

del(rt,a);

}

inline bool find(const Tp &a)

{

return find(rt,a);

}

inline Tp pred(const Tp &a)

{

return pred(rt,a);

}

inline Tp succ(const Tp &a)

{

return succ(rt,a);

}

inline bool empty()

{

return !sz[rt];

}

inline Tp min()

{

return min(rt);

}

inline Tp max()

{

return max(rt);

}

inline void delsmall(const Tp &a)

{

dels(rt,a);

}

inline int rank(const Tp &a)

{

return rank(rt,a);

}

inline Tp sel(const int &a)

{

return sel(rt,a);

}

inline Tp delsel(int a)

{

return delsel(rt,a);

}

private:

int cnt,rt,l[MAXX],r[MAXX],sz[MAXX];

Tp val[MAXX];

inline void rro(int &pos)

{

int k(l[pos]);

l[pos]=r[k];

r[k]=pos;

sz[k]=sz[pos];

sz[pos]=sz[l[pos]]+sz[r[pos]]+1;

pos=k;

}

inline void lro(int &pos)

{

int k(r[pos]);

r[pos]=l[k];

l[k]=pos;

sz[k]=sz[pos];

sz[pos]=sz[l[pos]]+sz[r[pos]]+1;

pos=k;

}

inline void mt(int &pos,bool flag)

{

if(!pos)

return;

if(flag)

if(sz[r[r[pos]]]>sz[l[pos]])

lro(pos);

else

if(sz[l[r[pos]]]>sz[l[pos]])

{

rro(r[pos]);

lro(pos);

}

else

return;

else

if(sz[l[l[pos]]]>sz[r[pos]])

rro(pos);

else

if(sz[r[l[pos]]]>sz[r[pos]])

{

lro(l[pos]);

rro(pos);

}

else

return;

mt(l[pos],false);

mt(r[pos],true);

mt(pos,false);

mt(pos,true);

}

void ins(int &pos,const Tp &a)

{

if(pos)

{

++sz[pos];

if(a<val[pos])

ins(l[pos],a);

else

ins(r[pos],a);

mt(pos,a>=val[pos]);

return;

}

pos=++cnt;

l[pos]=r[pos]=0;

val[pos]=a;

sz[pos]=1;

}

Tp del(int &pos,const Tp &a)

{

--sz[pos];

if(val[pos]==a || (a<val[pos] && !l[pos]) || (a>val[pos] && !r[pos]))

{

Tp ret(val[pos]);

if(!l[pos] || !r[pos])

pos=l[pos]+r[pos];

else

val[pos]=del(l[pos],val[pos]+1);

return ret;

}

else

if(a<val[pos])

return del(l[pos],a);

else

return del(r[pos],a);

}

bool find(int &pos,const Tp &a)

{

if(!pos)

return false;

if(a<val[pos])

return find(l[pos],a);

else

return (val[pos]==a || find(r[pos],a));

}

Tp pred(int &pos,const Tp &a)

{

if(!pos)

return a;

if(a>val[pos])

{

Tp ret(pred(r[pos],a));

if(ret==a)

return val[pos];

else

return ret;

}

return pred(l[pos],a);

}

Tp succ(int &pos,const Tp &a)

{

if(!pos)

return a;

if(a<val[pos])

{

Tp ret(succ(l[pos],a));

if(ret==a)

return val[pos];

else

return ret;

}

return succ(r[pos],a);

}

Tp min(int &pos)

{

if(l[pos])

return min(l[pos]);

else

return val[pos];

}

Tp max(int &pos)

{

if(r[pos])

return max(r[pos]);

else

return val[pos];

}

void dels(int &pos,const Tp &v)

{

if(!pos)

return;

if(val[pos]<v)

{

pos=r[pos];

dels(pos,v);

return;

}

dels(l[pos],v);

sz[pos]=1+sz[l[pos]]+sz[r[pos]];

}

int rank(const int &pos,const Tp &v)

{

if(val[pos]==v)

return sz[l[pos]]+1;

if(v<val[pos])

return rank(l[pos],v);

return rank(r[pos],v)+sz[l[pos]]+1;

}

Tp sel(const int &pos,const int &v)

{

if(sz[l[pos]]+1==v)

return val[pos];

if(v>sz[l[pos]])

return sel(r[pos],v-sz[l[pos]]-1);

return sel(l[pos],v);

}

Tp delsel(int &pos,int k)

{

--sz[pos];

if(sz[l[pos]]+1==k)

{

Tp re(val[pos]);

if(!l[pos] || !r[pos])

pos=l[pos]+r[pos];

else

val[pos]=del(l[pos],val[pos]+1);

return re;

}

if(k>sz[l[pos]])

return delsel(r[pos],k-1-sz[l[pos]]);

return delsel(l[pos],k);

}

};

* 1. sparse table 稀疏表RMQ

int num[MAXX],min[MAXX][20];

int lg[MAXX];

int main()

{

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

lg[i]=lg[i>>1]+1;

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

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

{

scanf("%d",num+i);

min[i][0]=num[i];

}

for(j=1;j<=lg[n];++j)

{

l=n+1-(1<<j);

j\_=j-1;

j\_\_=(1<<j\_);

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

min[i][j]=std::min(min[i][j\_],min[i+j\_\_][j\_]);

}

printf("Case %hd:\n",t);

while(q--)

{

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

k=lg[j-i+1];

printf("%d\n",std::min(min[i][k],min[j-(1<<k)+1][k]));

}

}

* 1. sparse table – square

int num[MAXX][MAXX],max[MAXX][MAXX][10];

short lg[MAXX];

int main()

{

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

lg[i]=lg[i>>1]+1;

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

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

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

{

scanf("%d",num[i]+j);

max[i][j][0]=num[i][j];

}

for(k=1;k<=lg[n];++k)

{

l=n+1-(1<<k);

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

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

max[i][j][k]=std::max(std::max(max[i][j][k-1],max[i+(1<<(k-1))][j][k-1]),std::max(max[i][j+(1<<(k-1))][k-1],max[i+(1<<(k-1))][j+(1<<(k-1))][k-1]));

}

printf("Case %hd:\n",t);

while(q--)

{

scanf("%hd %hd %hd",&i,&j,&l);

--i;

--j;

k=lg[l];

printf("%d\n",std::max(std::max(max[i][j][k],max[i][j+l-(1<<k)][k]),std::max(max[i+l-(1<<k)][j][k],max[i+l-(1<<k)][j+l-(1<<k)][k])));

}

}

* 1. sparse table – rectangle

#include<iostream>

#include<cstdio>

#include<algorithm>

#define MAXX 310

int mat[MAXX][MAXX];

int table[9][9][MAXX][MAXX];

int n;

short lg[MAXX];

int main()

{

for(int i(2);i<MAXX;++i)

lg[i]=lg[i>>1]+1;

int T;

std::cin >> T;

while (T--)

{

std::cin >> n;

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

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

{

std::cin >> mat[i][j];

table[0][0][i][j] = mat[i][j];

}

// 从小到大计算，保证后来用到的都已经计算过

for(int i=0;i<=lg[n];++i) // width

{

for(int j=0;j<=lg[n];++j) //height

{

if(i==0 && j==0)

continue;

for(int ii=0;ii+(1<<j)<=n;++ii)

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

if(i==0)

table[i][j][ii][jj]=std::min(table[i][j-1][ii][jj],table[i][j-1][ii+(1<<(j-1))][jj]);

else

table[i][j][ii][jj]=std::min(table[i-1][j][ii][jj],table[i-1][j][ii][jj+(1<<(i-1))]);

}

}

long long N;

std::cin >> N;

int r1, c1, r2, c2;

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

{

scanf("%d%d%d%d",&r1,&c1,&r2,&c2);

--r1;

--c1;

--r2;

--c2;

int w=lg[c2-c1+1];

int h=lg[r2-r1+1];

printf("%d\n",std::min(table[w][h][r1][c1],std::min(table[w][h][r1][c2-(1<<w)+1],std::min(table[w][h][r2-(1<<h)+1][c1],table[w][h][r2-(1<<h)+1][c2-(1<<w)+1]))));

}

}

return 0;

}

1.9 K-D tree k维最近点对

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <algorithm>

#include <stack>

#include <algorithm>

using namespace std;

#define MAXN 100010

typedef long long ll;

struct Point{

ll x,y;

void operator =(const Point &p){

x=p.x; y=p.y;

}

ll dis(const Point &a){

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

}

}point[MAXN],pp[MAXN];

struct Node{

int split;//{0,1} 0表示垂直于x轴的超平面，1表示垂直于y轴的超平面

Point p;//点

}tree[MAXN\*4];

bool cmpx(const Point &a,const Point &b)

{

return a.x<b.x;

}

bool cmpy(const Point &a,const Point &b)

{

return a.y<b.y;

}

void initTree(int x,int y,int split,int pos)

{

if(y<x) return ;

int mid=(x+y)>>1;

random\_shuffle(point+x,point+y);

if(split==0) nth\_element(point+x,point+mid,point+y+1,cmpx);

else nth\_element(point+x,point+mid,point+y+1,cmpy);

tree[pos].split=split;

tree[pos].p=point[mid];

initTree(x,mid-1,(split^1),2\*pos);

initTree(mid+1,y,(split^1),2\*pos+1);

}

ll ans;

void insert(int x,int y,Point &p,int pos)

{

if(y<x) return ;

int mid=(x+y)>>1;

ll temp=p.dis(tree[pos].p);

if(temp!=0) ans=min(ans,temp);

if(tree[pos].split==0){

if(p.x<=tree[pos].p.x){

insert(x,mid-1,p,2\*pos);

if(ans>=(p.x-tree[pos].p.x)\*(p.x-tree[pos].p.x))

insert(mid+1,y,p,2\*pos+1);

}

else{

insert(mid+1,y,p,2\*pos+1);

if(ans>=(p.x-tree[pos].p.x)\*(p.x-tree[pos].p.x))

insert(x,mid-1,p,2\*pos);

}

}

else

{

if(p.y<=tree[pos].p.y){

insert(x,mid-1,p,2\*pos);

if(ans>=(p.y-tree[pos].p.y)\*(p.y-tree[pos].p.y))

insert(mid+1,y,p,2\*pos+1);

}

else{

insert(mid+1,y,p,2\*pos+1);

if(ans>=(p.y-tree[pos].p.y)\*(p.y-tree[pos].p.y))

insert(x,mid-1,p,2\*pos);

}

}

}

int main()

{

int cases,n;

scanf("%d",&cases);

while(cases--)

{

scanf("%d",&n);

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

scanf("%I64d%I64d",&pp[i].x,&pp[i].y);

point[i]=pp[i];

}

initTree(1,n,0,1);

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

ans=1LL<<62;

insert(1,n,pp[i],1);

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

}

}

return 0;

}

1. geometry

2.1 3D

struct pv

{

double x,y,z;

pv() {}

pv(double xx,double yy,double zz):x(xx),y(yy),z(zz) {}

pv operator -(const pv& b)const

{

return pv(x-b.x,y-b.y,z-b.z);

}

pv operator \*(const pv& b)const

{

return pv(y\*b.z-z\*b.y,z\*b.x-x\*b.z,x\*b.y-y\*b.x);

}

double operator &(const pv& b)const

{

return x\*b.x+y\*b.y+z\*b.z;

}

};

//模

double Norm(pv p)

{

return sqrt(p&p);

}

//`绕单位向量V旋转theta角度`

pv Trans(pv pa,pv V,double theta)

{

double s = sin(theta);

double c = cos(theta);

double x,y,z;

x = V.x;

y = V.y;

z = V.z;

pv pp =

pv(

(x\*x\*(1-c)+c)\*pa.x+(x\*y\*(1-c)-z\*s)\*pa.y+(x\*z\*(1-c)+y\*s)\*pa.z,

(y\*x\*(1-c)+z\*s)\*pa.x+(y\*y\*(1-c)+c)\*pa.y+(y\*z\*(1-c)-x\*s)\*pa.z,

(x\*z\*(1-c)-y\*s)\*pa.x+(y\*z\*(1-c)+x\*s)\*pa.y+(z\*z\*(1-c)+c)\*pa.z);

return pp;

}

//经纬度转换

x=r\*sin(θ)\*cos(α);

y=r\*sin(θ)\*sin(α);

z=r\*cos(θ);

r=sqrt(x\*2+y\*2+z\*2);//??

r=sqrt(x^2+y^2+z^2);//??

α=atan(y/x);

θ=acos(z/r);

r∈[0,∞]

α∈[0,2π]

θ∈[0,π]

lat1∈[-π/2,π/2]

lng1∈[-π,π]

pv getpv(double lat,double lng,double r)

{

lat += pi/2;

lng += pi;

return

pv(r\*sin(lat)\*cos(lng),r\*sin(lat)\*sin(lng),r\*cos(lat));

}

//经纬度球面距离

#include<cstdio>

#include<cmath>

#define MAXX 1111

char buf[MAXX];

const double r=6875.0/2,pi=acos(-1.0);

double a,b,c,x1,x2,y2,ans;

int main()

{

double y1;

while(gets(buf)!=NULL)

{

gets(buf);

gets(buf);

scanf("%lf^%lf'%lf\" %s\n",&a,&b,&c,buf);

x1=a+b/60+c/3600;

x1=x1\*pi/180;

if(buf[0]=='S')

x1=-x1;

scanf("%s",buf);

scanf("%lf^%lf'%lf\" %s\n",&a,&b,&c,buf);

y1=a+b/60+c/3600;

y1=y1\*pi/180;

if(buf[0]=='W')

y1=-y1;

gets(buf);

scanf("%lf^%lf'%lf\" %s\n",&a,&b,&c,buf);

x2=a+b/60+c/3600;

x2=x2\*pi/180;

if(buf[0]=='S')

x2=-x2;

scanf("%s",buf);

scanf("%lf^%lf'%lf\" %s\n",&a,&b,&c,buf);

y2=a+b/60+c/3600;

y2=y2\*pi/180;

if(buf[0]=='W')

y2=-y2;

ans=acos(cos(x1)\*cos(x2)\*cos(y1-y2)+sin(x1)\*sin(x2))\*r;

printf("The distance to the iceberg: %.2lf miles.\n",ans);

if(ans+0.005<100)

puts("DANGER!");

gets(buf);

}

return 0;

}

inline bool ZERO(const double &a)

{

return fabs(a)<eps;

}

//三维向量是否为零

inline bool ZERO(pv p)

{

return (ZERO(p.x) && ZERO(p.y) && ZERO(p.z));

}

//直线相交

bool LineIntersect(Line3D L1, Line3D L2)

{

pv s = L1.s-L1.e;

pv e = L2.s-L2.e;

pv p = s\*e;

if (ZERO(p))

return false; //是否平行

p = (L2.s-L1.e)\*(L1.s-L1.e);

return ZERO(p&L2.e); //是否共面

}

//线段相交

bool inter(pv a,pv b,pv c,pv d)

{

pv ret = (a-b)\*(c-d);

pv t1 = (b-a)\*(c-a);

pv t2 = (b-a)\*(d-a);

pv t3 = (d-c)\*(a-c);

pv t4 = (d-c)\*(b-c);

return sgn(t1&ret)\*sgn(t2&ret) < 0 && sgn(t3&ret)\*sgn(t4&ret) < 0;

}

//点在直线上

bool OnLine(pv p, Line3D L)

{

return ZERO((p-L.s)\*(L.e-L.s));

}

//点在线段上

bool OnSeg(pv p, Line3D L)

{

return (ZERO((L.s-p)\*(L.e-p)) && EQ(Norm(p-L.s)+Norm(p-L.e),Norm(L.e-L.s)));

}

//点到直线距离

double Distance(pv p, Line3D L)

{

return (Norm((p-L.s)\*(L.e-L.s))/Norm(L.e-L.s));

}

//线段夹角

//范围值为[0,π]之间的弧度

double Inclination(Line3D L1, Line3D L2)

{

pv u = L1.e - L1.s;

pv v = L2.e - L2.s;

return acos( (u & v) / (Norm(u)\*Norm(v)) );

}

2.2 三维凸包

#include<cstdio>

#include<cmath>

#include<vector>

#include<algorithm>

#define MAXX 1111

#define eps 1e-8

#define inf 1e20

struct pv

{

double x,y,z;

pv(){}

pv(const double &xx,const double &yy,const double &zz):x(xx),y(yy),z(zz){}

inline pv operator-(const pv &i)const

{

return pv(x-i.x,y-i.y,z-i.z);

}

inline pv operator\*(const pv &i)const //叉积

{

return pv(y\*i.z-z\*i.y,z\*i.x-x\*i.z,x\*i.y-y\*i.x);

}

inline double operator^(const pv &i)const //点积

{

return x\*i.x+y\*i.y+z\*i.z;

}

inline double len()

{

return sqrt(x\*x+y\*y+z\*z);

}

};

struct pla

{

short a,b,c;

bool ok;

pla(){}

pla(const short &aa,const short &bb,const short &cc):a(aa),b(bb),c(cc),ok(true){}

inline void set();

inline void print()

{

printf("%hd %hd %hd\n",a,b,c);

}

};

pv pnt[MAXX];

std::vector<pla>fac;

short to[MAXX][MAXX];

inline void pla::set()

{

to[a][b]=to[b][c]=to[c][a]=fac.size();

}

inline double ptof(const pv &p,const pla &f) //点面距离?

{

return (pnt[f.b]-pnt[f.a])\*(pnt[f.c]-pnt[f.a])^(p-pnt[f.a]);

}

inline double vol(const pv &a,const pv &b,const pv &c,const pv &d)//有向体积\*6，即六面体体积

{

return (b-a)\*(c-a)^(d-a);

}

inline double ptof(const pv &p,const short &f) //p点到f号面的距离

{

return fabs(vol(pnt[fac[f].a],pnt[fac[f].b],pnt[fac[f].c],p)/((pnt[fac[f].b]-pnt[fac[f].a])\*(pnt[fac[f].c]-pnt[fac[f].a])).len());

}

void dfs(const short&,const short&);

void deal(const short &p,const short &a,const short &b)

{

if(fac[to[a][b]].ok)

if(ptof(pnt[p],fac[to[a][b]])>eps)

dfs(p,to[a][b]);

else

{

pla add(b,a,p);

add.set();

fac.push\_back(add);

}

}

void dfs(const short &p,const short &now)

{

fac[now].ok=false;

deal(p,fac[now].b,fac[now].a);

deal(p,fac[now].c,fac[now].b);

deal(p,fac[now].a,fac[now].c);

}

inline void make()

{

fac.resize(0);

if(n<4)

return;

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

if((pnt[0]-pnt[i]).len()>eps)

{

std::swap(pnt[i],pnt[1]);

break;

}

if(i==n)

return;

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

if(((pnt[0]-pnt[1])\*(pnt[1]-pnt[i])).len()>eps)

{

std::swap(pnt[i],pnt[2]);

break;

}

if(i==n)

return;

for(i=3;i<n;++i)

if(fabs((pnt[0]-pnt[1])\*(pnt[1]-pnt[2])^(pnt[2]-pnt[i]))>eps)

{

std::swap(pnt[3],pnt[i]);

break;

}

if(i==n)

return;

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

{

pla add((i+1)%4,(i+2)%4,(i+3)%4);

if(ptof(pnt[i],add)>0)

std::swap(add.c,add.b);

add.set();

fac.push\_back(add);

}

for(;i<n;++i)

for(j=0;j<fac.size();++j)

if(fac[j].ok && ptof(pnt[i],fac[j])>eps)

{

dfs(i,j);

break;

}

short tmp(fac.size());

fac.resize(0);

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

if(fac[i].ok)

fac.push\_back(fac[i]);

}

inline pv gc() //重心

{

pv re(0,0,0),o(0,0,0);

double all(0),v;

for(i=0;i<fac.size();++i)

{

v=vol(o,pnt[fac[i].a],pnt[fac[i].b],pnt[fac[i].c]);

re+=(pnt[fac[i].a]+pnt[fac[i].b]+pnt[fac[i].c])\*0.25\*v;

all+=v;

}

return re\*(1/all);

}

inline bool same(const short &s,const short &t) //两面是否相等

{

pv &a=pnt[fac[s].a],&b=pnt[fac[s].b],&c=pnt[fac[s].c];

return fabs(vol(a,b,c,pnt[fac[t].a]))<eps && fabs(vol(a,b,c,pnt[fac[t].b]))<eps && fabs(vol(a,b,c,pnt[fac[t].c]))<eps;

}

//表面多边形数目

inline short facetcnt()

{

short ans=0;

for(short i=0;i<fac.size();++i)

{

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

if(same(i,j))

break;

if(j==i)

++ans;

}

return ans;

}

//表面三角形数目

inline short trianglecnt()

{

return fac.size();

}

//三点构成的三角形面积\*2

inline double area(const pv &a,const pv &b,const pv &c)

{

return (b-a)\*(c-a).len();

}

//表面积

inline double area()

{

double ret(0);

for(i=0;i<fac.size();++i)

ret+=area(pnt[fac[i].a],pnt[fac[i].b],pnt[fac[i].c]);

return ret/2;

}

//体积

inline double volume()

{

pv o(0,0,0);

double ret(0);

for(short i(0);i<fac.size();++i)

ret+=vol(o,pnt[fac[i].a],pnt[fac[i].b],pnt[fac[i].c]);

return fabs(ret/6);

}

2.3 圆

//单位圆覆盖

#include<cstdio>

#include<cmath>

#include<vector>

#include<algorithm>

#define MAXX 333

#define eps 1e-8

struct pv

{

double x,y;

pv(){}

pv(const double &xx,const double &yy):x(xx),y(yy){}

inline pv operator-(const pv &i)const

{

return pv(x-i.x,y-i.y);

}

inline double cross(const pv &i)const

{

return x\*i.y-y\*i.x;

}

inline void print()

{

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

}

inline double len()

{

return sqrt(x\*x+y\*y);

}

}pnt[MAXX];

struct node

{

double k;

bool flag;

node(){}

node(const double &kk,const bool &ff):k(kk),flag(ff){}

inline bool operator<(const node &i)const

{

return k<i.k;

}

};

std::vector<node>alpha;

short n,i,j,k,l;

short ans,sum;

double R=2;

double theta,phi,d;

const double pi(acos(-1.0));

int main()

{

alpha.reserve(MAXX<<1);

while(scanf("%hd",&n),n)

{

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

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

ans=0;

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

{

alpha.resize(0);

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

if(i!=j)

{

if((d=(pnt[i]-pnt[j]).len())>R)

continue;

if((theta=atan2(pnt[j].y-pnt[i].y,pnt[j].x-pnt[i].x))<0)

theta+=2\*pi;

phi=acos(d/R);

alpha.push\_back(node(theta-phi,true));

alpha.push\_back(node(theta+phi,false));

}

std::sort(alpha.begin(),alpha.end());

for(j=0;j<alpha.size();++j)

{

if(alpha[j].flag)

++sum;

else

--sum;

ans=std::max(ans,sum);

}

}

printf("%hd\n",ans+1);

}

return 0;

}

//最小覆盖圆

#include<cstdio>

#include<cmath>

#define MAXX 511

#define eps 1e-8

struct pv

{

double x,y;

pv(){}

pv(const double &xx,const double &yy):x(xx),y(yy){}

inline pv operator-(const pv &i)const

{

return pv(x-i.x,y-i.y);

}

inline pv operator+(const pv &i)const

{

return pv(x+i.x,y+i.y);

}

inline double cross(const pv &i)const

{

return x\*i.y-y\*i.x;

}

inline double len()

{

return sqrt(x\*x+y\*y);

}

inline pv operator/(const double &a)const

{

return pv(x/a,y/a);

}

inline pv operator\*(const double &a)const

{

return pv(x\*a,y\*a);

}

}pnt[MAXX],o,tl,lt,aa,bb,cc,dd;

short n,i,j,k,l;

double r,u;

inline pv ins(const pv &a1,const pv &a2,const pv &b1,const pv &b2)

{

tl=a2-a1;

lt=b2-b1;

u=(b1-a1).cross(lt)/(tl).cross(lt);

return a1+tl\*u;

}

inline pv get(const pv &a,const pv &b,const pv &c)

{

aa=(a+b)/2;

bb.x=aa.x-a.y+b.y;

bb.y=aa.y+a.x-b.x;

cc=(a+c)/2;

dd.x=cc.x-a.y+c.y;

dd.y=cc.y+a.x-c.x;

return ins(aa,bb,cc,dd);

}

int main()

{

while(scanf("%hd",&n),n)

{

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

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

o=pnt[0];

r=0;

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

if((pnt[i]-o).len()>r+eps)

{

o=pnt[i];

r=0;

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

if((pnt[j]-o).len()>r+eps)

{

o=(pnt[i]+pnt[j])/2;

r=(o-pnt[j]).len();

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

if((o-pnt[k]).len()>r+eps)

{

o=get(pnt[i],pnt[j],pnt[k]);

r=(o-pnt[i]).len();

}

}

}

printf("%.2lf %.2lf %.2lf\n",o.x,o.y,r);

}

return 0;

}

//两原面积交

double dis(int x,int y)

{

return sqrt((double)(x\*x+y\*y));

}

double area(int x1,int y1,int x2,int y2,double r1,double r2)

{

double s=dis(x2-x1,y2-y1);

if(r1+r2<s) return 0;

else if(r2-r1>s) return PI\*r1\*r1;

else if(r1-r2>s) return PI\*r2\*r2;

double q1=acos((r1\*r1+s\*s-r2\*r2)/(2\*r1\*s));

double q2=acos((r2\*r2+s\*s-r1\*r1)/(2\*r2\*s));

return (r1\*r1\*q1+r2\*r2\*q2-r1\*s\*sin(q1));

}

//三角形外接圆

{

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

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

tp = pv((p[0].x+p[1].x)/2,(p[0].y+p[1].y)/2);

l[0] = Line(tp,pv(tp.x-(p[1].y-p[0].y),tp.y+(p[1].x-p[0].x)));

tp = pv((p[0].x+p[2].x)/2,(p[0].y+p[2].y)/2);

l[1] = Line(tp,pv(tp.x-(p[2].y-p[0].y),tp.y+(p[2].x-p[0].x)));

tp = LineToLine(l[0],l[1]);

r = pv(tp,p[0]).Length();

printf("(%.6f,%.6f,%.6f)\n",tp.x,tp.y,r);

}

//三角形内切圆

{

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

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

if (xmult(pv(p[0],p[1]),pv(p[0],p[2])) < 0)

swap(p[1],p[2]);

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

len[i] = pv(p[i],p[(i+1)%3]).Length();

tr = (len[0]+len[1]+len[2])/2;

r = sqrt((tr-len[0])\*(tr-len[1])\*(tr-len[2])/tr);

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

{

v = pv(p[i],p[i+1]);

tv = pv(-v.y,v.x);

tr = tv.Length();

tv = pv(tv.x\*r/tr,tv.y\*r/tr);

tp = pv(p[i].x+tv.x,p[i].y+tv.y);

l[i].s = tp;

tp = pv(p[i+1].x+tv.x,p[i+1].y+tv.y);

l[i].e = tp;

}

tp = LineToLine(l[0],l[1]);

printf("(%.6f,%.6f,%.6f)\n",tp.x,tp.y,r);

}

2.4 昀昀的多圆面积并

//去重

{

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

{

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

del[i] = false;

}

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

if (del[i] == false)

{

if (c[i].r == 0.0)

del[i] = true;

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

if (i != j)

if (del[j] == false)

if (cmp(Point(c[i].c,c[j].c).Len()+c[i].r,c[j].r) <= 0)

del[i] = true;

}

tn = n;

n = 0;

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

if (del[i] == false)

c[n++] = c[i];

}

//ans[i]表示被覆盖i次的面积

const double pi = acos(-1.0);

const double eps = 1e-8;

struct Point

{

double x,y;

Point(){}

Point(double \_x,double \_y)

{

x = \_x;

y = \_y;

}

double Length()

{

return sqrt(x\*x+y\*y);

}

};

struct Circle

{

Point c;

double r;

};

struct Event

{

double tim;

int typ;

Event(){}

Event(double \_tim,int \_typ)

{

tim = \_tim;

typ = \_typ;

}

};

int cmp(const double& a,const double& b)

{

if (fabs(a-b) < eps) return 0;

if (a < b) return -1;

return 1;

}

bool Eventcmp(const Event& a,const Event& b)

{

return cmp(a.tim,b.tim) < 0;

}

double Area(double theta,double r)

{

return 0.5\*r\*r\*(theta-sin(theta));

}

double xmult(Point a,Point b)

{

return a.x\*b.y-a.y\*b.x;

}

int n,cur,tote;

Circle c[1000];

double ans[1001],pre[1001],AB,AC,BC,theta,fai,a0,a1;

Event e[4000];

Point lab;

int main()

{

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

{

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

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

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

ans[i] = 0.0;

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

{

tote = 0;

e[tote++] = Event(-pi,1);

e[tote++] = Event(pi,-1);

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

if (j != i)

{

lab = Point(c[j].c.x-c[i].c.x,c[j].c.y-c[i].c.y);

AB = lab.Length();

AC = c[i].r;

BC = c[j].r;

if (cmp(AB+AC,BC) <= 0)

{

e[tote++] = Event(-pi,1);

e[tote++] = Event(pi,-1);

continue;

}

if (cmp(AB+BC,AC) <= 0) continue;

if (cmp(AB,AC+BC) > 0) continue;

theta = atan2(lab.y,lab.x);

fai = acos((AC\*AC+AB\*AB-BC\*BC)/(2.0\*AC\*AB));

a0 = theta-fai;

if (cmp(a0,-pi) < 0) a0 += 2\*pi;

a1 = theta+fai;

if (cmp(a1,pi) > 0) a1 -= 2\*pi;

if (cmp(a0,a1) > 0)

{

e[tote++] = Event(a0,1);

e[tote++] = Event(pi,-1);

e[tote++] = Event(-pi,1);

e[tote++] = Event(a1,-1);

}

else

{

e[tote++] = Event(a0,1);

e[tote++] = Event(a1,-1);

}

}

sort(e,e+tote,Eventcmp);

cur = 0;

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

{

if (cur != 0 && cmp(e[j].tim,pre[cur]) != 0)

{

ans[cur] += Area(e[j].tim-pre[cur],c[i].r);

ans[cur] += xmult(Point(c[i].c.x+c[i].r\*cos(pre[cur]),c[i].c.y+c[i].r\*sin(pre[cur])),

Point(c[i].c.x+c[i].r\*cos(e[j].tim),c[i].c.y+c[i].r\*sin(e[j].tim)))/2.0;

}

cur += e[j].typ;

pre[cur] = e[j].tim;

}

}

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

ans[i] -= ans[i+1];

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

printf("[%d] = %.3f\n",i,ans[i]);

}

return 0;

}

2.5 圆与多边形的交

bool InCircle(Point a,double r)

{

return cmp(a.x\*a.x+a.y\*a.y,r\*r) <= 0;

//`这里判断的时候EPS一定不要太小！！`

}

double CalcArea(Point a,Point b,double r)

{

Point p[4];

int tot = 0;

p[tot++] = a;

Point tv = Point(a,b);

Line tmp = Line(Point(0,0),Point(tv.y,-tv.x));

Point near = LineToLine(Line(a,b),tmp);

if (cmp(near.x\*near.x+near.y\*near.y,r\*r) <= 0)

{

double A,B,C;

A = near.x\*near.x+near.y\*near.y;

C = r;

B = C\*C-A;

double tvl = tv.x\*tv.x+tv.y\*tv.y;

double tmp = sqrt(B/tvl); //这样做只用一次开根

p[tot] = Point(near.x+tmp\*tv.x,near.y+tmp\*tv.y);

if (OnSeg(Line(a,b),p[tot]) == true) tot++;

p[tot] = Point(near.x-tmp\*tv.x,near.y-tmp\*tv.y);

if (OnSeg(Line(a,b),p[tot]) == true) tot++;

}

if (tot == 3)

{

if (cmp(Point(p[0],p[1]).Length(),Point(p[0],p[2]).Length()) > 0)

swap(p[1],p[2]);

}

p[tot++] = b;

double res = 0.0,theta,a0,a1,sgn;

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

{

if (InCircle(p[i],r) == true && InCircle(p[i+1],r) == true)

{

res += 0.5\*xmult(p[i],p[i+1]);

}

else

{

a0 = atan2(p[i+1].y,p[i+1].x);

a1 = atan2(p[i].y,p[i].x);

if (a0 < a1) a0 += 2\*pi;

theta = a0-a1;

if (cmp(theta,pi) >= 0) theta = 2\*pi-theta;

sgn = xmult(p[i],p[i+1])/2.0;

if (cmp(sgn,0) < 0) theta = -theta;

res += 0.5\*r\*r\*theta;

}

}

return res;

}

//调用

area2 = 0.0;

for (int i = 0;i < resn;i++) //遍历每条边，按照逆时针

area2 += CalcArea(p[i],p[(i+1)%resn],r);

2.6 closest point pair 最近点对

//演算法笔记1

struct Point {double x, y;} p[10], t[10];

bool cmpx(const Point& i, const Point& j) {return i.x < j.x;}

bool cmpy(const Point& i, const Point& j) {return i.y < j.y;}

double DnC(int L, int R)

{

if (L >= R) return 1e9; // 沒有點、只有一個點。

/\* Divide：把所有點分成左右兩側，點數盡量一樣多。 \*/

int M = (L + R) / 2;

/\* Conquer：左側、右側分別遞迴求解。 \*/

double d = min(DnC(L,M), DnC(M+1,R));

// if (d == 0.0) return d; // 提早結束

/\* Merge：尋找靠近中線的點，並依Y座標排序。O(NlogN)。 \*/

int N = 0; // 靠近中線的點數目

for (int i=M; i>=L && p[M].x - p[i].x < d; --i) t[N++] = p[i];

for (int i=M+1; i<=R && p[i].x - p[M].x < d; ++i) t[N++] = p[i];

sort(t, t+N, cmpy); // Quicksort O(NlogN)

/\* Merge：尋找橫跨兩側的最近點對。O(N)。 \*/

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

for (int j=1; j<=2 && i+j<N; ++j)

d = min(d, distance(t[i], t[i+j]));

return d;

}

double closest\_pair()

{

sort(p, p+10, cmpx);

return DnC(0, N-1);

}

//演算法笔记2

struct Point {double x, y;} p[10], t[10];

bool cmpx(const Point& i, const Point& j) {return i.x < j.x;}

bool cmpy(const Point& i, const Point& j) {return i.y < j.y;}

double DnC(int L, int R)

{

if (L >= R) return 1e9; // 沒有點、只有一個點。

/\* Divide：把所有點分成左右兩側，點數盡量一樣多。 \*/

int M = (L + R) / 2;

// 先把中線的X座標記起來，因為待會重新排序之後會跑掉。

double x = p[M].x;

/\* Conquer：左側、右側分別遞迴求解。 \*/

// 遞迴求解，並且依照Y座標重新排序。

double d = min(DnC(L,M), DnC(M+1,R));

// if (d == 0.0) return d; // 提早結束

/\* Merge：尋找靠近中線的點，並依Y座標排序。O(N)。 \*/

// 尋找靠近中線的點，先找左側。各點已照Y座標排序了。

int N = 0; // 靠近中線的點數目

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

if (x - p[i].x < d)

t[N++] = p[i];

// 尋找靠近中線的點，再找右側。各點已照Y座標排序了。

int P = N; // P為分隔位置

for (int i=M+1; i<=R; ++i)

if (p[i].x - x < d)

t[N++] = p[i];

// 以Y座標排序。使用Merge Sort方式，合併已排序的兩陣列。

inplace\_merge(t, t+P, t+N, cmpy);

/\* Merge：尋找橫跨兩側的最近點對。O(N)。 \*/

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

for (int j=1; j<=2 && i+j<N; ++j)

d = min(d, distance(t[i], t[i+j]));

/\* Merge：重新以Y座標排序所有點。O(N)。 \*/

// 如此一來，更大的子問題就可以直接使用Merge Sort。

inplace\_merge(p+L, p+M+1, p+R+1, cmpy);

return d;

}

double closest\_pair()

{

sort(p, p+10, cmpx);

return DnC(0, N-1);

}

//mzry

//分治

double calc\_dis(Point &a ,Point &b) {

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

}

//别忘了排序

bool operator<(const Point &a ,const Point &b) {

if(a.y != b.y) return a.x < b.x;

return a.x < b.x;

}

double Gao(int l ,int r ,Point pnts[]) {

double ret = inf;

if(l == r) return ret;

if(l+1 ==r) {

ret = min(calc\_dis(pnts[l],pnts[l+1]) ,ret);

return ret;

}

if(l+2 ==r) {

ret = min(calc\_dis(pnts[l],pnts[l+1]) ,ret);

ret = min(calc\_dis(pnts[l],pnts[l+2]) ,ret);

ret = min(calc\_dis(pnts[l+1],pnts[l+2]) ,ret);

return ret;

}

int mid = l+r>>1;

ret = min (ret ,Gao(l ,mid,pnts));

ret = min (ret , Gao(mid+1, r,pnts));

for(int c = l ; c<=r; c++)

for(int d = c+1; d <=c+7 && d<=r; d++) {

ret = min(ret , calc\_dis(pnts[c],pnts[d]));

}

return ret;

}

//增量

#include <iostream>

#include <cstdio>

#include <cstring>

#include <map>

#include <vector>

#include <cmath>

#include <algorithm>

#define Point pair<double,double>

using namespace std;

const int step[9][2] = {{-1,-1},{-1,0},{-1,1},{0,-1},{0,0},{0,1},{1,-1},{1,0},{1,1}};

int n,x,y,nx,ny;

map<pair<int,int>,vector<Point > > g;

vector<Point > tmp;

Point p[20000];

double tx,ty,ans,nowans;

vector<Point >::iterator it,op,ed;

pair<int,int> gird;

bool flag;

double Dis(Point p0,Point p1)

{

return sqrt((p0.first-p1.first)\*(p0.first-p1.first)+

(p0.second-p1.second)\*(p0.second-p1.second));

}

double CalcDis(Point p0,Point p1,Point p2)

{

return Dis(p0,p1)+Dis(p0,p2)+Dis(p1,p2);

}

void build(int n,double w)

{

g.clear();

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

g[make\_pair((int)floor(p[i].first/w),(int)floor(p[i].second/w))].push\_back(p[i]);

}

int main()

{

int t;

scanf("%d",&t);

for (int ft = 1;ft <= t;ft++)

{

scanf("%d",&n);

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

{

scanf("%lf%lf",&tx,&ty);

p[i] = make\_pair(tx,ty);

}

random\_shuffle(p,p+n);

ans = CalcDis(p[0],p[1],p[2]);

build(3,ans/2.0);

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

{

x = (int)floor(2.0\*p[i].first/ans);

y = (int)floor(2.0\*p[i].second/ans);

tmp.clear();

for (int k = 0;k < 9;k++)

{

nx = x+step[k][0];

ny = y+step[k][1];

gird = make\_pair(nx,ny);

if (g.find(gird) != g.end())

{

op = g[gird].begin();

ed = g[gird].end();

for (it = op;it != ed;it++)

tmp.push\_back(\*it);

}

}

flag = false;

for (int j = 0;j < tmp.size();j++)

for (int k = j+1;k < tmp.size();k++)

{

nowans = CalcDis(p[i],tmp[j],tmp[k]);

if (nowans < ans)

{

ans = nowans;

flag = true;

}

}

if (flag == true)

build(i+1,ans/2.0);

else

g[make\_pair((int)floor(2.0\*p[i].first/ans),(int)floor(2.0\*p[i].second/ans))].push\_back(p[i]);

}

printf("%.3f\n",ans);

}

}

2.7 half-plane intersection 半平面交

//abc解析几何方式

inline pv ins(const pv &p1,const pv &p2)

{

u=fabs(a\*p1.x+b\*p1.y+c);

v=fabs(a\*p2.x+b\*p2.y+c);

return pv((p1.x\*v+p2.x\*u)/(u+v),(p1.y\*v+p2.y\*u)/(u+v));

}

inline void get(const pv& p1,const pv& p2,double & a,double & b,double & c)

{

a=p2.y-p1.y;

b=p1.x-p2.x;

c=p2.x\*p1.y-p2.y\*p1.x;

}

inline pv ins(const pv &x,const pv &y)

{

get(x,y,d,e,f);

return pv((b\*f-c\*e)/(a\*e-b\*d),(a\*f-c\*d)/(b\*d-a\*e));

}

std::vector<pv>p[2];

int main()

{

k=0;

p[k].resize(0);

p[k].push\_back(pv(-inf,inf));

p[k].push\_back(pv(-inf,-inf));

p[k].push\_back(pv(inf,-inf));

p[k].push\_back(pv(inf,inf));

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

{

get(pnt[i],pnt[(i+1)%n],a,b,c);

c+=the\*sqrt(a\*a+b\*b);

p[!k].resize(0);

for(l=0;l<p[k].size();++l)

if(a\*p[k][l].x+b\*p[k][l].y+c<eps)

p[!k].push\_back(p[k][l]);

else

{

m=(l+p[k].size()-1)%p[k].size();

if(a\*p[k][m].x+b\*p[k][m].y+c<-eps)

p[!k].push\_back(ins(p[k][m],p[k][l]));

m=(l+1)%p[k].size();

if(a\*p[k][m].x+b\*p[k][m].y+c<-eps)

p[!k].push\_back(ins(p[k][m],p[k][l]));

}

k=!k;

if(p[k].empty())

break;

}

//结果在p[k]中

return p[k].empty();

}

//计算几何方式

//本例求多边形核

inline pv ins(const pv &a,const pv &b)

{

u=fabs(ln.cross(a-pnt[i]));

v=fabs(ln.cross(b-pnt[i]))+u;

tl=b-a;

return pv(u\*tl.x/v+a.x,u\*tl.y/v+a.y);

}

int main()

{

j=0;

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

{

ln=pnt[(i+1)%n]-pnt[i];

p[!j].resize(0);

for(k=0;k<p[j].size();++k)

if(ln.cross(p[j][k]-pnt[i])<=0)

p[!j].push\_back(p[j][k]);

else

{

l=(k-1+p[j].size())%p[j].size();

if(ln.cross(p[j][l]-pnt[i])<0)

p[!j].push\_back(ins(p[j][k],p[j][l]));

l=(k+1)%p[j].size();

if(ln.cross(p[j][l]-pnt[i])<0)

p[!j].push\_back(ins(p[j][k],p[j][l]));

}

j=!j;

}

//结果在p[j]中

}

//mrzy

bool HPIcmp(Line a, Line b)

{

if (fabs(a.k - b.k) > eps)

return a.k < b.k;

return ((a.s - b.s) \* (b.e-b.s)) < 0;

}

Line Q[100];

void HPI(Line line[], int n, Point res[], int &resn)

{

int tot = n;

std::sort(line, line + n, HPIcmp);

tot = 1;

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

if (fabs(line[i].k - line[i - 1].k) > eps)

line[tot++] = line[i];

int head = 0, tail = 1;

Q[0] = line[0];

Q[1] = line[1];

resn = 0;

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

{

if (fabs((Q[tail].e-Q[tail].s)\*(Q[tail - 1].e-Q[tail - 1].s)) < eps || fabs((Q[head].e-Q[head].s)\*(Q[head + 1].e-Q[head + 1].s)) < eps)

return;

while (head < tail && (((Q[tail]&Q[tail - 1]) - line[i].s) \* (line[i].e-line[i].s)) > eps)

--tail;

while (head < tail && (((Q[head]&Q[head + 1]) - line[i].s) \* (line[i].e-line[i].s)) > eps)

++head;

Q[++tail] = line[i];

}

while (head < tail && (((Q[tail]&Q[tail - 1]) - Q[head].s) \* (Q[head].e-Q[head].s)) > eps)

tail--;

while (head < tail && (((Q[head]&Q[head + 1]) - Q[tail].s) \* (Q[tail].e-Q[tail].s)) > eps)

head++;

if (tail <= head + 1)

return;

for (int i = head; i < tail; i++)

res[resn++] = Q[i] & Q[i + 1];

if (head < tail + 1)

res[resn++] = Q[head] & Q[tail];

}

* 1. 点在多边形内

/\*

射线法, 多边形可以是凸的或凹的

poly的顶点数目要大于等于3

返回值为：

0 -- 点在poly内

1 -- 点在poly边界上

2 -- 点在poly外

\*/

int inPoly(pv p,pv poly[], int n)

{

int i, count;

Line ray, side;

count = 0;

ray.s = p;

ray.e.y = p.y;

ray.e.x = -1; //-INF，注意取值防止越界！

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

{

side.s = poly[i];

side.e = poly[(i+1)%n];

if(OnSeg(p, side))

return 1;

// 如果side平行x轴则不作考虑

if (side.s.y == side.e.y)

continue;

if (OnSeg(side.s, ray))

{

if (side.s.y > side.e.y)

count++;

}

else

if (OnSeg(side.e, ray))

{

if (side.e.y > side.s.y)

count++;

}

else

if (inter(ray, side))

count++;

}

return ((count % 2 == 1) ? 0 : 2);

}

* 1. rotating caliper 旋转卡壳

//最远点对

l=ans=0;

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

{

tl=pnt[(i+1)%n]-pnt[i];

while(abs(tl.cross(pnt[(l+1)%n]-pnt[i]))>abs(tl.cross(pnt[l]-pnt[i])))

l=(l+1)%n;

ans=std::max(ans,std::max(dist(pnt[l],pnt[i]),dist(pnt[l],pnt[(i+1)%n])));

}

return ans;

//两凸包最近距离

int main()

{

sq=sp=0;

for(i=1;i<ch[1].size();++i)

if(ch[1][sq]<ch[1][i])

sq=i;

tp=sp;

tq=sq;

ans=(ch[0][sp]-ch[1][sq]).len();

do

{

a1=ch[0][sp];

a2=ch[0][(sp+1)%ch[0].size()];

b1=ch[1][sq];

b2=ch[1][(sq+1)%ch[1].size()];

tpv=b1-(b2-a1);

tpv.x = b1.x - (b2.x - a1.x);

tpv.y = b1.y - (b2.y - a1.y);

len=(tpv-a1).cross(a2-a1);

if(fabs(len)<eps)

{

ans=std::min(ans,p2l(a1,b1,b2));

ans=std::min(ans,p2l(a2,b1,b2));

ans=std::min(ans,p2l(b1,a1,a2));

ans=std::min(ans,p2l(b2,a1,a2));

sp=(sp+1)%ch[0].size();

sq=(sq+1)%ch[1].size();

}

else

if(len<-eps)

{

ans=std::min(ans,p2l(b1,a1,a2));

sp=(sp+1)%ch[0].size();

}

else

{

ans=std::min(ans,p2l(a1,b1,b2));

sq=(sq+1)%ch[1].size();

}

}while(tp!=sp || tq!=sq);

return ans;

}

//外接矩形 by mzry

inline void solve()

{

resa = resb = 1e100;

double dis1,dis2;

Point xp[4];

Line l[4];

int a,b,c,d;

int sa,sb,sc,sd;

a = b = c = d = 0;

sa = sb = sc = sd = 0;

Point va,vb,vc,vd;

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

{

va = Point(p[a],p[(a+1)%n]);

vc = Point(-va.x,-va.y);

vb = Point(-va.y,va.x);

vd = Point(-vb.x,-vb.y);

if (sb < sa)

{

b = a;

sb = sa;

}

while (xmult(vb,Point(p[b],p[(b+1)%n])) < 0)

{

b = (b+1)%n;

sb++;

}

if (sc < sb)

{

c = b;

sc = sb;

}

while (xmult(vc,Point(p[c],p[(c+1)%n])) < 0)

{

c = (c+1)%n;

sc++;

}

if (sd < sc)

{

d = c;

sd = sc;

}

while (xmult(vd,Point(p[d],p[(d+1)%n])) < 0)

{

d = (d+1)%n;

sd++;

}

//`卡在p[a],p[b],p[c],p[d]上`

sa++;

}

}

//合并凸包

给定凸多边形 P = { p(1) , ... , p(m) } 和 Q = { q(1) , ... , q(n) }，一个点对 (p(i), q(j)) 形成 P 和 Q 之间的桥当且仅当：

(p(i), q(j)) 形成一个并踵点对。

p(i-1), p(i+1), q(j-1), q(j+1) 都位于由 (p(i), q(j)) 组成的线的同一侧。

假设多边形以标准形式给出并且顶点是以顺时针序排列， 算法如下：

1、分别计算 P 和 Q 拥有最大 y 坐标的顶点。 如果存在不止一个这样的点， 取 x 坐标最大的。

2、构造这些点的遂平切线， 以多边形处于其右侧为正方向（因此他们指向 x 轴正方向）。

3、同时顺时针旋转两条切线直到其中一条与边相交。 得到一个新的并踵点对 (p(i), q(j)) 。 对于平行边的情况， 得到三个并踵点对。

4、对于所有有效的并踵点对 (p(i), q(j))： 判定 p(i-1), p(i+1), q(j-1), q(j+1) 是否都位于连接点 (p(i), q(j)) 形成的线的同一侧。 如果是， 这个并踵点对就形成了一个桥， 并标记他。

5、重复执行步骤3和步骤4直到切线回到他们原来的位置。

6、所有可能的桥此时都已经确定了。 通过连续连接桥间对应的凸包链来构造合并凸包。

上述的结论确定了算法的正确性。 运行时间受步骤1，5，6约束。 他们都为 O(N) 运行时间（N 是顶点总数）。 因此算法拥有现行的时间复杂度。

一个凸多边形间的桥实际上确定了另一个有用的概念：多边形间公切线。 同时， 桥也是计算凸多边形交的算法核心。

//临界切线

1、计算 P 上 y 坐标值最小的顶点（称为 yminP ） 和 Q 上 y 坐标值最大的顶点（称为 ymaxQ）。

2、为多边形在 yminP 和 ymaxQ 处构造两条切线 LP 和 LQ 使得他们对应的多边形位于他们的右侧。 此时 LP 和 LQ 拥有不同的方向， 并且 yminP 和 ymaxQ 成为了多边形间的一个对踵点对。

3、令 p(i)= yminP， q(j)= ymaxQ。 (p(i), q(j)) 构成了多边形间的一个对踵点对。 检测是否有 p(i-1),p(i+1) 在线 (p(i), q(j)) 的一侧， 并且 q(j-1),q(j+1) 在另一侧。 如果成立， (p(i), q(j)) 确定了一条CS线。

4、旋转这两条线， 直到其中一条和其对应的多边形的边重合。

5、一个新的对踵点对确定了。 如果两条线都与边重合， 总共三对对踵点对（原先的顶点和新的顶点的组合）需要考虑。 对于所有的对踵点对， 执行上面的测试。

6、重复执行步骤4和步骤5， 直到新的点对为(yminP,ymaxQ)。

7、输出CS线。

//最小/最大周长/面积外接矩形

1、计算全部四个多边形的端点， 称之为 xminP， xmaxP， yminP， ymaxP。

2、通过四个点构造 P 的四条切线。 他们确定了两个“卡壳”集合。

3、如果一条（或两条）线与一条边重合， 那么计算由四条线决定的矩形的面积， 并且保存为当前最小值。 否则将当前最小值定义为无穷大。

4、顺时针旋转线直到其中一条和多边形的一条边重合。

5、计算新矩形的周长/面积， 并且和当前最小值比较。 如果小于当前最小值则更新， 并保存确定最小值的矩形信息。

6、重复步骤4和步骤5， 直到线旋转过的角度大于90度。

7、输出外接矩形的最小周长。

2.10 点极角排序

inline bool cmp(const pv& a,const pv& b)

{

if (a.y\*b.y <= 0)

{

if (a.y > 0 || b.y > 0)

return a.y < b.y;

if (a.y == 0 && b.y == 0)

return a.x < b.x;

}

return a.cross(b) > 0;

}

* 1. triangle's fermat point 三角形费马点

当有一个内角不小于120°时，费马点为此角对应顶点。

当三角形的内角都小于120°时

以三角形的每一边为底边，向外做三个正三角形△ABC'，△BCA'，△CAB'。

连接CC'、BB'、AA'，则三条线段的交点就是所求的点。

* 1. Pick's theorem 皮克定理

A:面积

i:内部格点数目

b:边上格点数目

A = i + b/2 - 1。

给定顶点座标均是整点（或正方形格点）的简单多边形

取格点的组成图形的面积为一单位。在平行四边形格点，皮克定理依然成立。套用于任意三角形格点，皮克定理则是

A = 2i + b - 2。

* 1. others

eps

如果sqrt(a), asin(a), acos(a) 中的a是你自己算出来并传进来的，那就得小心了。如果a本来应该是0的，由于浮点误差，可能实际是一个绝对值很小的负数（比如-1e-12），这样sqrt(a)应得0的，直接因a不在定义域而出错。类似地，如果a本来应该是±1,则asin(a)、acos(a)也有可能出错。因此，对于此种函数，必需事先对a进行校正。

现在考虑一种情况，题目要求输出保留两位小数。有个case的正确答案的精确值是0:005,按理应该输出0:01，但你的结果可能是0:005000000001(恭喜)，也有可能是0:004999999999(悲剧)，如果按照printf("%.2lf", a)输出，那你的遭遇将和括号里的字相同。

如果a为正，则输出a + eps, 否则输出a - eps。

不要输出-0.000

注意double的数据范围

a==b fabs(a-b)<eps

a!=b fabs(a-b)>eps

a<b a+eps<b

a<=b a<b+eps

a>b a>b+eps

a>=b a+eps>b

三角函数

cos/sin/tan 输入弧度

acos 输入[-1,+1]，输出[0,π]

asin 输入[-1,+1]，输出[-π/2,+π/2]

atan 输出[-π/2,+π/2]

atan2 输入(y,x)(注意顺序),返回tan(y/x),[-π,+π]。xy都是零的时候会发生除零错误

other

log 自然对数(ln)

log10 你猜……

ceil 向上

floor 向下

round

cpp: 四舍六入五留双

java: add 0.5,then floor

cpp:

（一）当尾数小于或等于4时，直接将尾数舍去。

（二）当尾数大于或等于6时，将尾数舍去并向前一位进位。

（三）当尾数为5，而尾数后面的数字均为0时，应看尾数“5”的前一位：若前一位数字此时为奇数，就应向前进一位；若前一位数字此时为偶数，则应将尾数舍去。数字“0”在此时应被视为偶数。

（四）当尾数为5，而尾数“5”的后面还有任何不是0的数字时，无论前一位在此时为奇数还是偶数，也无论“5”后面不为0的数字在哪一位上，都应向前进一位。

1. graph

3.1 Two-sat

#define maxn 2008

struct Twosat

{

int n;

std::vector<int>G[maxn\*2];

bool mark[maxn\*2];

int s[maxn\*2],c;

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 init(int n)

{

this->n=n;

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

G[i].clear();

memset(mark,0,sizeof(mark));

}

void add\_clause(int x,int xval,int y,int yval)//增加一条子句

{

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;

}

};

3.2 Articulation 割点

void dfs(const short &now,const short &fa) // now从1开始

{

short p(0);

dfn[now]=low[now]=cnt++;

for(std::list<short>::const\_iterator it(edge[now].begin());it!=edge[now].end();++it)

if(dfn[\*it]==-1)

{

dfs(\*it,now);

++p;

low[now]=std::min(low[now],low[\*it]);

if((now==1 && p>1) || (now!=1 && low[\*it]>=dfn[now])) // 如果从出发点出发的子节点不能由兄弟节点到达，那么出发点为割点。 如果现节点不是出发点，但是其子孙节点不能达到祖先节点，那么该节点为割点

ans.insert(now);

}

else

if(\*it!=fa)

low[now]=std::min(low[now],dfn[\*it]);

}

3.4 Brige 割边

void dfs(const short &now,const short &fa)

{

dfn[now]=low[now]=cnt++;

for(int i(0);i<edge[now].size();++i)

if(dfn[edge[now][i]]==-1)

{

dfs(edge[now][i],now);

low[now]=std::min(low[now],low[edge[now][i]]);

if(low[edge[now][i]]>dfn[now]) //如果子节点不能够走到父节点之前去,那么该边为桥

{

if(edge[now][i]<now)

{

j=edge[now][i];

k=now;

}

else

{

j=now;

k=edge[now][i];

}

ans.push\_back(node(j,k));

}

}

else

if(edge[now][i]!=fa)

low[now]=std::min(low[now],low[edge[now][i]]);

}

3.3 Biconnected Component 双连通分量

#include<cstdio>

#include<cstring>

#include<stack>

#include<queue>

#include<algorithm>

const int MAXN=100000\*2;

const int MAXM=200000;

//0-based

struct edges

{

int to,next;

bool cut,visit;

} edge[MAXM<<1];

int head[MAXN],low[MAXN],dpt[MAXN],L;

bool visit[MAXN],cut[MAXN];

int idx;

std::stack<int> st;

int bcc[MAXM];

void init(int n)

{

L=0;

memset(head,-1,4\*n);

memset(visit,0,n);

}

void add\_edge(int u,int v)

{

edge[L].cut=edge[L].visit=false;

edge[L].to=v;

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

head[u]=L++;

}

void dfs(int u,int fu,int deg)

{

cut[u]=false;

visit[u]=true;

low[u]=dpt[u]=deg;

int tot=0;

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

{

int v=edge[i].to;

if (edge[i].visit)

continue;

st.push(i/2);

edge[i].visit=edge[i^1].visit=true;

if (visit[v])

{

low[u]=dpt[v]>low[u]?low[u]:dpt[v];

continue;

}

dfs(v,u,deg+1);

edge[i].cut=edge[i^1].cut=(low[v]>dpt[u] || edge[i].cut);

if (u!=fu) cut[u]=low[v]>=dpt[u]?1:cut[u];

if (low[v]>=dpt[u] || u==fu)

{

while (st.top()!=i/2)

{

int x=st.top()\*2,y=st.top()\*2+1;

bcc[st.top()]=idx;

st.pop();

}

bcc[i/2]=idx++;

st.pop();

}

low[u]=low[v]>low[u]?low[u]:low[v];

tot++;

}

if (u==fu && tot>1)

cut[u]=true;

}

int main()

{

int n,m;

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

{

init(n);

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

{

int u,v;

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

add\_edge(u,v);

add\_edge(v,u);

}

idx=0;

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

if (!visit[i])

dfs(i,i,0);

}

return 0;

}

3.5 Strongly Connected Component 强连通分量

void dfs(const short &now)

{

dfn[now]=low[now]=cnt++;

st.push(now);

for(std::list<short>::const\_iterator it(edge[now].begin());it!=edge[now].end();++it)

if(dfn[\*it]==-1)

{

dfs(\*it);

low[now]=std::min(low[now],low[\*it]);

}

else

if(sc[\*it]==-1)

low[now]=std::min(low[now],dfn[\*it]);

if(dfn[now]==low[now])

{

while(sc[now]==-1)

{

sc[st.top()]=p;

st.pop();

}

++p;

}

3.6 best spanning tree 最优比率生成树

#include<cstdio>

#include<cstring>

#include<cmath>

#define MAXX 1111

struct

{

int x,y;

double z;

} node[MAXX];

struct

{

double l,c;

} map[MAXX][MAXX];

int n,l,f[MAXX],pre[MAXX];

double dis[MAXX];

double mst(double x)

{

int i,j,tmp;

double min,s=0,t=0;

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

f[1]=1;

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

{

dis[i]=map[1][i].c-map[1][i].l\*x;

pre[i]=1;

}

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

{

min=1e10;

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

if (!f[j] && min>dis[j])

{

min=dis[j];

tmp=j;

}

f[tmp]=1;

t+=map[pre[tmp]][tmp].l;

s+=map[pre[tmp]][tmp].c;

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

if (!f[j] && map[tmp][j].c-map[tmp][j].l\*x<dis[j])

{

dis[j]=map[tmp][j].c-map[tmp][j].l\*x;

pre[j]=tmp;

}

}

return s/t;

}

int main()

{

int i,j;

double a,b;

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

{

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

scanf("%d%d%lf",&node[i].x,&node[i].y,&node[i].z);

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

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

{

map[j][i].l=map[i][j].l=sqrt(1.0\*(node[i].x-node[j].x)\*(node[i].x-node[j].x)+(node[i].y-node[j].y)\*(node[i].y-node[j].y));

map[j][i].c=map[i][j].c=fabs(node[i].z-node[j].z);

}

a=0,b=mst(a);

while (fabs(b-a)>1e-8)

{

a=b;

b=mst(a);

}

printf("%.3lf\n",b);

}

return 0;

}

**3.7 zhu-liu algorithm 最小树形图**

#include<cstdio>

#include<cstring>

#include<algorithm>

const int inf = 0x5fffffff;

int n,m,u,v,cost,dis[1001][1001],L;

int pre[1001],id[1001],visit[1001],in[1001];

void init(int n)

{

L = 0;

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

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

dis[i][j] = inf;

}

struct Edge

{

int u,v,cost;

};

Edge e[1001\*1001];

int zhuliu(int root,int n,int m,Edge e[])

{

int res = 0,u,v;

while (true)

{

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

in[i] = inf;

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

if (e[i].u != e[i].v && e[i].cost < in[e[i].v])

{

pre[e[i].v] = e[i].u;

in[e[i].v] = e[i].cost;

}

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

if (i != root)

if (in[i] == inf)

return -1;

int tn = 0;

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

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

in[root] = 0;

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

{

res += in[i];

v = i;

while (visit[v] != i && id[v] == -1 && v != root)

{

visit[v] = i;

v = pre[v];

}

if(v != root && id[v] == -1)

{

for(int u = pre[v] ; u != v ; u = pre[u])

id[u] = tn;

id[v] = tn++;

}

}

if(tn == 0) break;

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

if (id[i] == -1)

id[i] = tn++;

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

{

int v = e[i].v;

e[i].u = id[e[i].u];

e[i].v = id[e[i].v];

if (e[i].u != e[i].v)

e[i++].cost -= in[v];

else

std::swap(e[i],e[--m]);

}

n = tn;

root = id[root];

}

return res;

}

int main()

{

freopen("asdf","r",stdin);

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

{

init(n);

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

{

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

if (u == v) continue;

dis[u][v] = std::min(dis[u][v],cost);

}

L = 0;

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

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

if (dis[i][j] != inf)

{

e[L].u = i;

e[L].v = j;

e[L++].cost = dis[i][j];

}

printf("%d\n",zhuliu(0,n,L,e));

}

return 0;

}

**3.8 k-th shortest path K短路**

#include<cstdio>

#include<cstring>

#include<queue>

#include<vector>

int K;

class states

{

public:

int cost,id;

};

int dist[1000];

class cmp

{

public:

bool operator ()(const states &i,const states &j)

{

return i.cost>j.cost;

}

};

class cmp2

{

public:

bool operator ()(const states &i,const states &j)

{

return i.cost+dist[i.id]>j.cost+dist[j.id];

}

};

struct edges

{

int to,next,cost;

} edger[100000],edge[100000];

int headr[1000],head[1000],Lr,L;

void dijkstra(int s)

{

states u;

u.id=s;

u.cost=0;

dist[s]=0;

std::priority\_queue<states,std::vector<states>,cmp> q;

q.push(u);

while (!q.empty())

{

u=q.top();

q.pop();

if (u.cost!=dist[u.id])

continue;

for (int i=headr[u.id]; i!=-1; i=edger[i].next)

{

states v=u;

v.id=edger[i].to;

if (dist[v.id]>dist[u.id]+edger[i].cost)

{

v.cost=dist[v.id]=dist[u.id]+edger[i].cost;

q.push(v);

}

}

}

}

int num[1000];

inline void init(int n)

{

Lr=L=0;

memset(head,-1,4\*n);

memset(headr,-1,4\*n);

memset(dist,63,4\*n);

memset(num,0,4\*n);

}

void add\_edge(int u,int v,int x)

{

edge[L].to=v;

edge[L].cost=x;

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

head[u]=L++;

edger[Lr].to=u;

edger[Lr].cost=x;

edger[Lr].next=headr[v];

headr[v]=Lr++;

}

inline int a\_star(int s,int t)

{

if (dist[s]==0x3f3f3f3f)

return -1;

std::priority\_queue<states,std::vector<states>,cmp2> q;

states tmp;

tmp.id=s;

tmp.cost=0;

q.push(tmp);

while (!q.empty())

{

states u=q.top();

q.pop();

num[u.id]++;

if (num[t]==K)

return u.cost;

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

{

int v=edge[i].to;

tmp.id=v;

tmp.cost=u.cost+edge[i].cost;

q.push(tmp);

}

}

return -1;

}

int main()

{

int n,m;

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

init(n);

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

{

int u,v,x;

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

add\_edge(u-1,v-1,x);

}

int s,t;

scanf("%d%d%d",&s,&t,&K);

if (s==t)

++K;

dijkstra(t-1);

printf("%d\n",a\_star(s-1,t-1));

}

**3.9 LCA – DA LCA倍增**

#include<cstdio>

#include<cstring>

#include<queue>

const int NSIZE = 50000;

const int DEG = 20;

struct trees

{

int fa[DEG];

int head,deg;

} tree[NSIZE];

struct edges

{

int to , next;

} edge[NSIZE];

struct states

{

int u,fu,deg;

};

int L;

void add\_edge(int x, int y)

{

edge[L].to = y;

edge[L].next = tree[x].head;

tree[x].head = L++;

}

int Root;

inline void BFS(int s)

{

std::queue<states> que;

states st;

st.deg=0;

st.fu=st.u=s;

que.push(st);

while(!que.empty())

{

states st=que.front();

que.pop();

tree[st.u].deg = st.deg;

tree[st.u].fa[0] = st.fu;

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

tree[st.u].fa[i]=s;

for (int tmp=st.fu,num=1;tree[tmp].deg;tmp=tree[st.u].fa[num++])

tree[st.u].fa[num]=tree[tmp].fa[num-1];

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

{

int v = edge[i].to;

if (v == st.fu)

continue;

states nst;

nst.u=v;

nst.fu=st.u;

nst.deg=st.deg+1;

que.push(nst);

}

}

}

inline int LCA(int x, int y)

{

if(tree[x].deg > tree[y].deg)

std::swap(x,y);

int hx=tree[x].deg,hy=tree[y].deg;

int tx=x,ty=y;

for (int det=hy-hx,i=0; det; det>>=1,i++)

if (det&1)

ty=tree[ty].fa[i];

if(tx == ty)

return tx;

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

{

if(tree[tx].fa[i] == tree[ty].fa[i])

continue;

tx = tree[tx].fa[i];

ty = tree[ty].fa[i];

}

return tree[tx].fa[0];

}

int main()

{

int t;

scanf("%d",&t);

while(t--)

{

int n;

scanf("%d",&n);

L = 0;

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

tree[i].head = -1;

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

{

int a,b;

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

add\_edge(a-1,b-1);

add\_edge(b-1,a-1);

}

Root=0;

BFS(Root);

int a,b;

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

int lca=LCA(a-1,b-1)+1;

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

}

return 0;

}

**3.10 hungarian algorithm 匈牙利算法**

#include<cstdio>

#include<cstring>

#define MAXX 111

bool Map[MAXX][MAXX],visit[MAXX];

int link[MAXX],n,m;

bool dfs(int t)

{

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

if (!visit[i] && Map[t][i]){

visit[i] = true;

if (link[i]==-1 || dfs(link[i])){

link[i] = t;

return true;

}

}

return false;

}

int main()

{

int k,a,b,c;

while (scanf("%d",&n),n){

memset(Map,false,sizeof(Map));

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

while (k--){

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

if (b && c)

Map[b][c] = true;

}

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

int ans = 0;

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

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

if (dfs(i))

ans++;

}

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

}

}

**3.11 Kuhn-Munkres algorithm KM算法**

bool match(int u)//匈牙利

{

vx[u]=true;

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

if(lx[u]+ly[i]==g[u][i]&&!vy[i])

{

vy[i]=true;

if(!d[i]||match(d[i]))

{

d[i]=u;

return true;

}

}

return false;

}

inline void update()//

{

int i,j;

int a=1<<30;

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

for(j=1;j<=n;++j)if(!vy[j])

a=min(a,lx[i]+ly[j]-g[i][j]);

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

{

if(vx[i])lx[i]-=a;

if(vy[i])ly[i]+=a;

}

}

void km()

{

int i,j;

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

{

lx[i]=ly[i]=d[i]=0;

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

lx[i]=max(lx[i],g[i][j]);

}

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

{

while(true)

{

memset(vx,0,sizeof(vx));

memset(vy,0,sizeof(vy));

if(match(i))

break;

update();

}

}

int ans=0;

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

if(d[i]!=0)

ans+=g[d[i]][i];

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

}

int main()

{

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

{

for(int i=1;i<=n;++i)gets(s[i]);

memset(g,0,sizeof(g));

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

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

if(i!=j) g[i][j]=cal(s[i],s[j]);

km();

}

return 0;

}

//bupt

//km算法：求二分图最佳匹配 n^3复杂度

int dfs(int u)//匈牙利求增广路

{

int v;

sx[u]=1;

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

if (!sy[v] && lx[u]+ly[v]==map[u][v])

{

sy[v]=1;

if (match[v]==-1 || dfs(match[v]))

{

match[v]=u;

return 1;

}

}

return 0;

}

int bestmatch(void)//km求最佳匹配

{

int i,j,u;

for (i=1; i<=n; i++)//初始化顶标

{

lx[i]=-1;

ly[i]=0;

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

if (lx[i]<map[i][j])

lx[i]=map[i][j];

}

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

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

{

while (true)

{

memset(sx,0,sizeof(sx));

memset(sy,0,sizeof(sy));

if (dfs(u))

break;

int dx=Inf;//若找不到增广路，则修改顶标~~

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

{

if (sx[i])

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

if(!sy[j] && dx>lx[i]+ly[j]-map[i][j])

dx=lx[i]+ly[j]-map[i][j];

}

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

{

if (sx[i])

lx[i]-=dx;

if (sy[i])

ly[i]+=dx;

}

}

}

int sum=0;

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

sum+=map[match[i]][i];

return sum;

}

**3.12 Minimum Cost Maximum Flow 最小费用最大流**

struct mcmf

{

struct Edge

{

int from,to,cap,flow,cost;

};

int n,m,s,t;

std::vector<Edge>edges;

std::vector<int>G[maxn];

int inq[maxn],d[maxn],p[maxn],a[maxn];

void init(int n)

{

this->n=n;

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

G[i].clear();

edges.clear();

}

void addedge(int from,int to,int cap,int cost)

{

Edge x={from,to,cap,0,cost};

edges.push\_back(x);

Edge y={to,from,0,0,-cost};

edges.push\_back(y);

m=edges.size();

G[from].push\_back(m-2);

G[to].push\_back(m-1);

}

int mincost(int s,int t)

{

int flow=0,cost=0;

while(BellmanFord(s,t,flow,cost));

if(flow!=(n-1)/2)return -1;

return cost;

}

private:

bool BellmanFord(int s,int t,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;

std::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.to]>d[u]+e.cost)

{

d[e.to]=d[u]+e.cost;

p[e.to]=G[u][i];

a[e.to]=min(a[u],e.cap-e.flow);

if(!inq[e.to])

{

Q.push(e.to);

inq[e.to]=1;

}

}

}

}

if(d[t]==INF)

return false;

flow+=a[t];

cost+=d[t]\*a[t];

int u=t;

while(u!=s)

{

edges[p[u]].flow+=a[t];

edges[p[u]^1].flow-=a[t];

u=edges[p[u]].from;

}

return true;

}

}G;

**3.13 minimum cut 全局最小割**

#include <iostream>

using namespace std;

const int maxn=510;

int map[maxn][maxn];

int n;

void contract(int x,int y)//合并两个点

{

int i,j;

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

if (i!=x) map[x][i]+=map[y][i],map[i][x]+=map[i][y];

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

{

map[i-1][j]=map[i][j];

map[j][i-1]=map[j][i];

}

n--;

}

int w[maxn],c[maxn];

int sx,tx;

int mincut()

//求最大生成树，计算最后一个点的割，并保存最后一条边的两个顶点

{

int i,j,k,t;

memset(c,0,sizeof(c));

c[0]=1;

for (i=0; i<n; i++) w[i]=map[0][i];

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

{

t=k=-1;

for (j=0; j<n; j++) if (c[j]==0&&w[j]>k)

k=w[t=j];

c[sx=t]=1;

for (j=0; j<n; j++) w[j]+=map[t][j];

}

for (i=0; i<n; i++) if (c[i]==0) return w[tx=i];

}

int main()

{

int i,j,k,m;

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

{

memset(map,0,sizeof(map));

while (m--)

{

scanf("%d%d%d",&i,&j,&k);

map[i][j]+=k;

map[j][i]+=k;

}

int mint=999999999;

while (n>1)

{

k=mincut();

if (k<mint) mint=k;

contract(sx,tx);

}

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

}

return 0;

}

**3.14 Shortest Augmenting Path algorithm 网络流SAP**

#include <cstring>

#include <cstdio>

#include <vector>

#include <queue>

#define maxn 1005

#define INF 1<<30

using namespace std;

struct Edge

{

int from,to,cap,flow;

};

vector<Edge>edges;

vector<int>G[maxn];

int num[maxn],p[maxn],n,m;

int st[maxn],et[maxn],nt[maxn];

int d[maxn],s,t,cur[maxn];

void addedge(int from,int to,int cap)

{

struct Edge x={from,to,cap,0};

edges.push\_back(x);

struct Edge y={to,from,0,0};

edges.push\_back(y);

int m=edges.size();

G[from].push\_back(m-2);

G[to].push\_back(m-1);

}

void bfs()//一次BFS求出每个点到汇点的“距离”

{

queue<int>q;

memset(d,0,sizeof(d));

d[t]=1;

q.push(t);

while(!q.empty())

{

int u=q.front(); q.pop();

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

if(G[u][i]&1)

{

Edge& e=edges[G[u][i]];

if(!d[e.to])

{

d[e.to]=d[u]+1;

q.push(e.to);

}

}

}

}

int augment()//找到一条增广路，增广

{

int u=t, a=INF;

while(u!=s)

{

Edge& e=edges[p[u]];

a=min(a,e.cap-e.flow);//增广流量等于增广路径上残余流量最小的一条边

u=e.from;

}

u=t;

while(u!=s)

{

edges[p[u]].flow+=a;

edges[p[u]^1].flow-=a;

u=edges[p[u]].from;

}

return a;

}

int sap()

{

int flow=0;

bfs();

memset(num,0,sizeof(num));

for(int i=0;i<=t;++i)num[d[i]]++;

int u=s;

memset(cur,0,sizeof(cur));

while(d[s]<t)

{

if(u==t)

{

flow+=augment();

u=s;

}

int ok=0;

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

{

Edge& e=edges[G[u][i]];

if(e.cap>e.flow && d[u]==d[e.to]+1)

{

ok=1;

p[e.to]=G[u][i];

cur[u]=i;

u=e.to;

break;

}

}

if(!ok)//当前图没有可增广的路，改变距离标记

{

int m=t-1;

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

{

Edge& e=edges[G[u][i]];

if(e.cap>e.flow) m=min(m,d[e.to]);

}

if(--num[d[u]]==0)break;//gap优化

num[d[u]=m+1]++;

cur[u]=0;//重新构图以后当前弧指向第一条边

if(u!=s)u=edges[p[u]].from;

}

}

return flow;

}

void init()

{

edges.clear();

for(int i=0;i<maxn;++i)G[i].clear();

}

int main()

{

int T, i;

// freopen("1.txt","r",stdin);

scanf("%d", &T);

while (T--)

{

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

s=0; t=-1;

int res=0;

init();

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

{

scanf("%d%d%d",&st[i],&et[i],&nt[i]);

res+=nt[i];

if(et[i]+n+1>t)t=et[i]+n+1;

}

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

{

int j;

addedge(s,i,nt[i]);

for(j=st[i];j<=et[i];++j) addedge(i,j+n,1);

}

for(i=n+1;i<t;++i) addedge(i,t,m);

int ans = sap();

if(ans==res)printf("Yaha, Garfield, You Finish it!\n");

else printf("Oh~~Garfield, You are just as before!\n");

}

return 0;

}

**3.15 Stable Marriage 稳定婚姻**

//对于每个预备队列中的对象，及被匹配对象，先按照喜好程度排列匹配对象

while(!g.empty()) // 预备匹配队列

{

if(dfn[edge[g.front()].front()]==-1)

dfn[edge[g.front()].front()]=g.front(); // 如果目前还没尝试匹配过的对象没有被任何别的对象占据

else

{

for(it=edge[edge[g.front()].front()].begin();it!=edge[edge[g.front()].front()].end();++it)

if(\*it==dfn[edge[g.front()].front()] || \*it==g.front()) //如果被匹配对象更喜欢正在被匹配的人或现在准备匹配的对象

break;

if(\*it==g.front()) //如果更喜欢新的

{

g.push\_back(dfn[edge[g.front()].front()]);

dfn[edge[g.front()].front()]=g.front();

}

else

g.push\_back(g.front()); //否则放到队尾，重新等待匹配

}

edge[g.front()].pop\_front(); //每组匹配最多只考虑一次

g.pop\_front();

}

}

**四、math**

**4.1 cantor 康托展开**

const int PermSize = 12;

int fac[PermSize] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800};

inline int Cantor(int a[])

{

int i, j, cnt;

int res = 0;

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

{

cnt = 0;

for (j = i + 1; j < PermSize; ++j)

if (a[i] > a[j])

++cnt;

res = res + cnt \* fac[PermSize - i - 1];

}

return res;

}

bool h[13];

inline void UnCantor(int x, int res[])

{

int i,j,l,t;

for (i = 1;i <= 12;i++)

h[i] = false;

for (i = 1; i <= 12; i++)

{

t = x / fac[12 - i];

x -= t \* fac[12 - i];

for (j = 1, l = 0; l <= t; j++)

if (!h[j])

l++;

j--;

h[j] = true;

res[i - 1] = j;

}

}

**4.2 combinations 组合数**

#include<cstdio>

#include<cstring>

#include<iostream>

int mod;

long long num[100000];

int ni[100],mi[100];

int len;

void init(int p)

{

mod=p;

num[0]=1;

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

num[i]=i\*num[i-1]%p;

}

void get(int n,int ni[],int p)

{

for (int i = 0; i < 100; i++)

ni[i] = 0;

int tlen = 0;

while (n != 0)

{

ni[tlen++] = n%p;

n /= p;

}

len = tlen;

}

long long power(long long x,long long y)

{

long long ret=1;

for (long long a=x%mod; y; y>>=1,a=a\*a%mod)

if (y&1)

ret=ret\*a%mod;

return ret;

}

long long getInv(long long x)//`mod为素数`

{

return power(x,mod-2);

}

long long calc(int n,int m,int p)//C(n,m)%p

{

init(p);

long long ans=1;

for (; n && m && ans; n/=p,m/=p)

{

if (n%p>=m%p)

ans = ans\*num[n%p]%p \*getInv(num[m%p]%p)%p \*getInv(num[n%p-m%p])%p;

else

ans=0;

}

return ans;

}

int main()

{

int t;

scanf("%d",&t);

while (t--)

{

int n,m,p;

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

printf("%lld\n",calc(n+m,m,p));

}

return 0;

}

**4.3 euler's totient function 欧拉函数**

对正整数n，欧拉函数φ是少于或等于n的数中与n互质的数的数目。φ函数的值通式：φ(x)=x(1-1/p1)(1-1/p2)(1-1/p3)(1-1/p4)…..(1-1/pn),其中p1, p2……pn为x的所有质因数，x是不为0的整数。φ(1)=1（唯一和1互质的数就是1本身）。若m,n互质，φ(mn)=φ(m)φ(n)。

int Euler(int n)

{

int ans = n;

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

{

if (n%i==0)

{

ans = ans-ans/i;

while (n%i==0)

n /= i;

}

}

if (n>1)

ans = ans-ans/n;

return ans;

}

//递推

inline void Euler2()

{

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

euler[1] = 1;

for (int i = 2; i <= 3000000; i++)

{

if (!euler[i])å

{

for (int j = i; j <= 3000000; j += i)

{

if (!euler[j])

euler[j] = j;

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

}

}

}

}

**4.4 extended euclidean algorithm 扩展欧几里得**

//返回ax+by=gcd(a,b)的一组解

long long ex\_gcd(long long a,long long b,long long &x,long long &y)

{

if (b)

{

long long ret = ex\_gcd(b,a%b,x,y),tmp = x;

x = y;

y = tmp-(a/b)\*y;

return ret;

}

else

{

x = 1;

y = 0;

return a;

}

}

**4.5 inverse element 逆元**

inline void getInv2(int x,int mod)

{

inv[1]=1;

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

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

}

long long power(long long x,long long y,int mod)

{

long long ret=1;

for (long long a=x%mod; y; y>>=1,a=a\*a%mod)

if (y&1)

ret=ret\*a%mod;

return ret;

}

inline int getInv(int x,int mod)//mod为素数

{

return power(x,mod-2);

}

**4.6 Lucas' theorem 组合数求模**

#include <cstdio>

/\*

Lucas 快速求解C(n,m)%p

\*/

void gcd(int n,int k,int &x,int &y)

{

if(k==0){x=1;y=0;return ;}

else

{

gcd(k,n%k,x,y);

int t=x;x=y;

y=t-(n/k)\*y;

return;

}

}

int CmodP(int n,int k,int p)

{

if(k>n) return 0;

int a,b,flag=0,x,y;

a=b=1;

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

{

x=n-i+1,y=i;

while(x%p==0) x/=p,flag++;

while(y%p==0) y/=p,flag--;

x%=p,y%=p,a\*=x,b\*=y;

b%=p,a%=p;

/\* while(x%p==0)

{

x/=p;

flag++;

}

while(y%p==0)

{

y/=p;

flag--;

}

x=x%p; y=y%p; a=a\*x;

b=b\*y; b=b%p; a=a%p;\*/

}

if(flag) return 0;

gcd(b,p,x,y);

if(x<0) x+=p;

a\*=x,a%=p;

return a;

}

//用Lucas 定理求解 C(n,m) % p ,p 是素数

long long Lucas(long long n, long long m, long long p)

{

long long ans=1;

long long a,b;

while(m&&n&&ans)

{

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

ans=ans%p;

n=n/p;

m=m/p;

}

return ans;

}

int main()

{

long long n,k,p,ans;

int cas=0;

// freopen("1.txt","r",stdin);

// freopen("out2.txt","w",stdout);

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

{

++cas;

if(k>n-k)k=n-k;

ans=Lucas(n+1,k,p)+n-k;

printf("Case #%d: %I64d\n",cas,ans%p);

}

return 0;

}

**4.7 matrix 矩阵**

//矩阵快速幂

struct Matrix

{

int a[52][52];

Matrix operator \* (const Matrix &b)const

{

Matrix res;

for (int i = 0; i < 52; i++)

for (int j = 0; j < 52; j++)

{

res.a[i][j] = 0;

for (int k = 0; k < 52; k++)

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

}

return res;

}

Matrix operator ^ (int y)const

{

Matrix res, x;

for (int i = 0; i < 52; i++)

{

for (int j = 0; j < 52; j++)

res.a[i][j] = 0, x.a[i][j] = a[i][j];

res.a[i][i] = 1;

}

for (; y; y >>= 1, x = x \* x)

if (y & 1)

res = res \* x;

return res;

}

};

**4.8 miller rabin 素数判断**

inline unsigned long long multi\_mod(const unsigned long long &a,unsigned long long b,const unsigned long long &n)

{

unsigned long long exp(a%n),tmp(0);

while(b)

{

if(b&1)

{

tmp+=exp;

if(tmp>n)

tmp-=n;

}

exp<<=1;

if(exp>n)

exp-=n;

b>>=1;

}

return tmp;

}

inline unsigned long long exp\_mod(unsigned long long a,unsigned long long b,const unsigned long long &c)

{

unsigned long long tmp(1);

while(b)

{

if(b&1)

tmp=multi\_mod(tmp,a,c);

a=multi\_mod(a,a,c);

b>>=1;

}

return tmp;

}

inline bool miller\_rabbin(const unsigned long long &n,short T)

{

if(n==2)

return true;

if(n<2 || !(n&1))

return false;

unsigned long long a,u(n-1),x,y;

short t(0),i;

while(!(u&1))

{

++t;

u>>=1;

}

while(T--)

{

a=rand()%(n-1)+1;

x=exp\_mod(a,u,n);

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

{

y=multi\_mod(x,x,n);

if(y==1 && x!=1 && x!=n-1)

return false;

x=y;

}

if(y!=1)

return false;

}

return true;

}

**4.9 pollard rho 大数分解质因数**

#include<cstdio>

#include<cstdlib>

#include<list>

short T;

unsigned long long a;

std::list<unsigned long long>fac;

inline unsigned long long multi\_mod(const unsigned long long &a,unsigned long long b,const unsigned long long &n)

{

unsigned long long exp(a%n),tmp(0);

while(b)

{

if(b&1)

{

tmp+=exp;

if(tmp>n)

tmp-=n;

}

exp<<=1;

if(exp>n)

exp-=n;

b>>=1;

}

return tmp;

}

inline unsigned long long exp\_mod(unsigned long long a,unsigned long long b,const unsigned long long &c)

{

unsigned long long tmp(1);

while(b)

{

if(b&1)

tmp=multi\_mod(tmp,a,c);

a=multi\_mod(a,a,c);

b>>=1;

}

return tmp;

}

inline bool miller\_rabbin(const unsigned long long &n,short T)

{

if(n==2)

return true;

if(n<2 || !(n&1))

return false;

unsigned long long a,u(n-1),x,y;

short t(0),i;

while(!(u&1))

{

++t;

u>>=1;

}

while(T--)

{

a=rand()%(n-1)+1;

x=exp\_mod(a,u,n);

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

{

y=multi\_mod(x,x,n);

if(y==1 && x!=1 && x!=n-1)

return false;

x=y;

}

if(y!=1)

return false;

}

return true;

}

unsigned long long gcd(const unsigned long long &a,const unsigned long long &b)

{

return b?gcd(b,a%b):a;

}

inline unsigned long long pollar\_rho(const unsigned long long n,const unsigned long long &c)

{

unsigned long long x(rand()%(n-1)+1),y,d,i(1),k(2);

y=x;

while(true)

{

++i;

x=(multi\_mod(x,x,n)+c)%n;

d=gcd((x-y+n)%n,n);

if(d>1 && d<n)

return d;

if(x==y)

return n;

if(i==k)

{

k<<=1;

y=x;

}

}

}

void find(const unsigned long long &n,short c)

{

if(n==1)

return;

if(miller\_rabbin(n,6))

{

fac.push\_back(n);

return;

}

unsigned long long p(n);

short k(c);

while(p>=n)

p=pollar\_rho(p,c--);

find(p,k);

find(n/p,k);

}

int main()

{

scanf("%hd",&T);

while(T--)

{

scanf("%llu",&a);

fac.clear();

find(a,120);

if(fac.size()==1)

puts("Prime");

else

{

fac.sort();

printf("%llu\n",fac.front());

}

}

return 0;

}

**4.10 prime 素数**

#include<vector>

std::vector<int>prm;

bool flag[MAXX];

int main()

{

prm.reserve(MAXX); // pi(x)=x/ln(x);

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

{

if(!flag[i])

prm.push\_back(i);

for(j=0;j<prm.size() && i\*prm[j]<MAXX;++j)

{

flag[i\*prm[j]]=true;

if(i%pmr[j]==0)

break;

}

}

return 0;

}

**4.11 mod 同余：**

(a+b)%m = (a%m+b%m)%m

(a-b)%m = ((a%m-b%m)%m+m)%m (a>b)

(a\*b)%m = (a%m\*b%m)%m

(a/b)%m = (a%(b\*m))/b //连乘时不可用

1. string

**5.1 aho corasick**

#include<queue>

#define MAX 26

struct node

{

node \*fal,\*nxt[MAX];

short idx;

node()

{

memset(this,0,sizeof(node));

}

}\*rt(new node());

std::queue<node\*>Q;

void free(node \*p)

{

for(short i(0);i<MAX;++i)

if(p->nxt[i])

free(p->nxt[i]);

delete p;

}

inline void add(const std::vector<short> &s,const short &idx)

{

node \*p=rt;

for(int i(0);i<s.size();++i)

{

if(!p->nxt[s[i]])

p->nxt[s[i]]=new node();

p=p->nxt[s[i]];

}

p->idx=idx;

}

inline void make()

{

Q.push(rt);

node \*p,\*q;

short i;

while(!Q.empty())

{

p=Q.front();

Q.pop();

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

if(p->nxt[i])

{

q=p->fal;

while(q)

{

if(q->nxt[i])

{

p->nxt[i]->fal=q->nxt[i];

break;

}

q=q->fal;

}

if(!q)

p->nxt[i]->fal=rt;

Q.push(p->nxt[i]);

}

}

}

inline void match(const std::vector<short> &s)

{

node \*p(rt),\*q;

for(int i(0);i<s.size();++i)

{

while(p!=rt && !p->nxt[s[i]])

p=p->fal;

if(p)

p=p->nxt[s[i]];

else

p=rt;

for(q=p;q!=rt;q=q->fal)

done[q->idx]=true;

}

}

**5.2 manacher 最长回文子串**

#include<cstdio>

#include<vector>

#define MAXX 1111

std::vector<char>str;

char buf[MAXX];

int z[MAXX<<1];

int i,j,l,r;

int ii,n,c;

inline int match(const int &a,const int &b)

{

int i(0);

while(a-i>=0 && b+i<str.size() && str[a-i]==str[b+i])

++i;

return i;

}

int main()

{

gets(buf);

str.reserve(MAXX<<1);

for(i=0;buf[i];++i)

{

str.push\_back('$');

str.push\_back(buf[i]);

}

str.push\_back('$');

z[0]=1;

c=l=r=0;

for(i=1;i<str.size();++i)

{

ii=(l<<1)-i;

n=r+1-i;

if(i>r)

{

z[i]=match(i,i);

l=i;

r=i+z[i]-1;

}

else

if(z[ii]==n)

{

z[i]=n+match(i-n,i+n);

l=i;

r=i+z[i]-1;

}

else

z[i]=std::min(z[ii],n);

if(z[i]>z[c])

c=i;

}

for(i=c-z[c]+2,n=c+z[c];i<n;i+=2)

putchar(str[i]);

puts("");

return 0;

}

**5.3 MP 字符串匹配**

int i,j;

inline void make(char \*buf,int \*fal)

{

fal[0]=-1;

for(i=1,j=-1;buf[i];++i)

{

while(j>=0 && buf[j+1]!=buf[j])

j=fal[j];

if(buf[j+1]==buf[j])

++j;

fal[i]=j;

}

}

inline int void match(char \*p,char \*t,int\* fal)

{

for(i=0,j=-1;t[i];++i)

{

while(j>=0 && p[j+1]!=t[i])

j=fal[j];

if(p[j+1]==t[i])

++j;

if(!p[j])

{

//匹配成功

j=fal[j];

}

}

**5.4 smallest representation 最小表示法**

int min(char a[],int len)

{

int i = 0,j = 1,k = 0;

while (i < len && j < len && k < len)

{

int cmp = a[(j+k)%len]-a[(i+k)%len];

if (cmp == 0)

k++;

else

{

if (cmp > 0)

j += k+1;

else

i += k+1;

if (i == j) j++;

k = 0;

}

}

return std::min(i,j);

}

**5.5 suffix array – da 后缀数组倍增**

int wx[maxn],wy[maxn],\*x,\*y,wss[maxn],wv[maxn];

bool cmp(int \*r,int n,int a,int b,int l)

{

return a+l<n && b+l<n && r[a]==r[b]&&r[a+l]==r[b+l];

}

void da(int str[],int sa[],int rank[],int height[],int n,int m)

{

int \*s = str;

int \*x=wx,\*y=wy,\*t,p;

int i,j;

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

wss[i]=0;

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

wss[x[i]=s[i]]++;

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

wss[i]+=wss[i-1];

for(i=n-1; i>=0; i--)

sa[--wss[x[i]]]=i;

for(j=1,p=1; p<n && j<n; j\*=2,m=p)

{

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

y[p++]=i;

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

if(sa[i]-j>=0)

y[p++]=sa[i]-j;

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

wv[i]=x[y[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--)

sa[--wss[wv[i]]]=y[i];

for(t=x,x=y,y=t,p=1,i=1,x[sa[0]]=0; i<n; i++)

x[sa[i]]=cmp(y,n,sa[i-1],sa[i],j)?p-1:p++;

}

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

rank[sa[i]]=i;

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

if(rank[i]>0)

for(k?k--:0,j=sa[rank[i]-1]; i+k < n && j+k < n && str[i+k]==str[j+k]; ++k);

}

**5.6 suffix array 后缀数组**

#include<cstdio>

#include<cstring>

#include<algorithm>

#define MAXX 1111

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

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

int wa[MAXX],wb[MAXX],wv[MAXX],ws[MAXX];

inline bool c0(const int \*str,const int &a,const int &b)

{

return str[a]==str[b] && str[a+1]==str[b+1] && str[a+2]==str[b+2];

}

inline bool c12(const int \*str,const int &k,const int &a,const int &b)

{

if(k==2)

return str[a]<str[b] || str[a]==str[b] && c12(str,1,a+1,b+1);

else

return str[a]<str[b] || str[a]==str[b] && wv[a+1]<wv[b+1];

}

inline void sort(int \*str,int \*a,int \*b,const int &n,const int &m)

{

memset(ws,0,sizeof(ws));

int i;

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

++ws[wv[i]=str[a[i]]];

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

ws[i]+=ws[i-1];

for(i=n-1;i>=0;--i)

b[--ws[wv[i]]]=a[i];

}

inline void dc3(int \*str,int \*sa,const int &n,const int &m)

{

int \*strn(str+n);

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

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

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

if(i%3)

wa[tbc++]=i;

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

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

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

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

strn[F(wb[i])]=c0(str,wb[i-1],wb[i])?j-1:j++;

if(j<tbc)

dc3(strn,san,tbc,j);

else

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

san[strn[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(str,wb,wa,ta,m);

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

wv[wb[i]=G(san[i])]=i;

for(i=j=k=0;i<ta && j<tbc;)

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

while(i<ta)

sa[k++]=wa[i++];

while(j<tbc)

sa[k++]=wb[j++];

}

int rk[MAXX],lcpa[MAXX],sa[MAXX\*3];

int str[MAXX\*3]; //必须int

int main()

{

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

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

{

scanf("%d",&k);

num[i]=k-j+100;

j=k;

}

num[n]=0;

dc3(num,sa,n+1,191); //191: str中取值范围，桶排序

for(i=1;i<=n;++i) // rank数组

rk[sa[i]]=i;

for(i=k=0;i<n;++i) // lcp数组

if(!rk[i])

lcpa[0]=0;

else

{

j=sa[rk[i]-1];

if(k>0)

--k;

while(num[i+k]==num[j+k])

++k;

lcpa[rk[i]]=k;

}

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

sptb[0][i]=i;

for(i=1;i<=lg[n];++i) //sparse table RMQ

{

k=n+1-(1<<i);

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

{

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

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

sptb[i][j]=lcpa[a]<lcpa[b]?a:b;

}

}

}

inline int ask(int l,int r)

{

a=lg[r-l+1];

r-=(1<<a)-1;

l=sptb[a][l];

r=sptb[a][r];

return lcpa[l]<lcpa[r]?l:r;

}

inline int lcp(int l,int r) // 字符串上[l,r]区间的rmq

{

l=rk[l];

r=rk[r];

if(l>r)

std::swap(l,r);

return lcpa[ask(l+1,r)];

}

**5.7 z algorithm**

inline void make(int \*z,char \*buf)

{

int i,j,l,r;

l=0;

r=1;

z[0]=strlen(buf);

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

if(r<=i || z[i-l]>=r-i)

{

j=std::max(i,r);

while(j<z[0] && buf[j]==buf[j-i])

++j;

z[i]=j-i;

if(i<j)

{

l=i;

r=j;

}

}

else

z[i]=z[i-l];

}

for(i=1;i<len && i+z[i]<len;++i); //i=可能最小循环节长度

六、 search

**6.1 dlx - precise cover**

#include<cstdio>

#define INF 0x7FFFFFFF

#define MAXN 1000010

int n, m, size;

int L[MAXN], R[MAXN], U[MAXN], D[MAXN], H[MAXN];

int S[MAXN], C[MAXN], X[MAXN], Q[MAXN];

void Init()

{

int i;

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

{

S[i] = 0;

L[i + 1] = i;

R[i] = i + 1;

U[i] = D[i] = i;

}

R[m] = 0;

size = m + 1;

}

void Remove(int c)

{

int i, j;

R[L[c]] = R[c];

L[R[c]] = L[c];

for (i = D[c]; i != c; i = D[i])

{

for (j = R[i]; j != i; j = R[j])

{

D[U[j]] = D[j];

U[D[j]] = U[j];

S[C[j]]--;

}

}

}

void Resume(int c)

{

int i, j;

R[L[c]] = c;

L[R[c]] = c;

for (i = D[c]; i != c; i = D[i])

{

for (j = R[i]; j != i; j = R[j])

{

U[D[j]] = j;

D[U[j]] = j;

S[C[j]]++;

}

}

}

void Link(int r, int c)

{

D[size] = D[c];

U[size] = c;

U[D[c]] = size;

D[c] = size;

if (H[r] < 0)

H[r] = L[size] = R[size] = size;

else

{

L[size] = H[r];

R[size] = R[H[r]];

L[R[H[r]]] = size;

R[H[r]] = size;

}

S[c]++;

C[size] = c;

X[size++] = r;

}

bool Dance(int now)

{

int i, j, c, temp;

if (R[0] == 0)

return true;

for (temp = INF, i = R[0]; i; i = R[i])

{

if (S[i] < temp)

{

c = i;

temp = S[i];

}

}

Remove(c);

for (i = D[c]; i != c; i = D[i])

{

for (j = R[i]; j != i; j = R[j])

Remove(C[j]);

if (Dance(now + 1))

return true;

for (j = L[i]; j != i; j = L[j])

Resume(C[j]);

}

Resume(c);

return false;

}

**6.2 dlx - repeat cover**

#include<cstdio>

#include<cstring>

#include<algorithm>

#define MAXN 110

#define MAXM 1000000

#define INF 0x7FFFFFFF

using namespace std;

int G[MAXN][MAXN];

int L[MAXM], R[MAXM], U[MAXM], D[MAXM];

int size, ans, S[MAXM], H[MAXM], C[MAXM];

bool vis[MAXN \* 100];

void Link(int r, int c)

{

U[size] = c;

D[size] = D[c];

U[D[c]] = size;

D[c] = size;

if (H[r] < 0)

H[r] = L[size] = R[size] = size;

else

{

L[size] = H[r];

R[size] = R[H[r]];

L[R[H[r]]] = size;

R[H[r]] = size;

}

S[c]++;

C[size++] = c;

}

void Remove(int c)

{

int i;

for (i = D[c]; i != c; i = D[i])

{

L[R[i]] = L[i];

R[L[i]] = R[i];

}

}

void Resume(int c)

{

int i;

for (i = D[c]; i != c; i = D[i])

L[R[i]] = R[L[i]] = i;

}

int A()

{

int i, j, k, res;

memset(vis, false, sizeof(vis));

for (res = 0, i = R[0]; i; i = R[i])

{

if (!vis[i])

{

res++;

for (j = D[i]; j != i; j = D[j])

{

for (k = R[j]; k != j; k = R[k])

vis[C[k]] = true;

}

}

}

return res;

}

void Dance(int now)

{

if (R[0] == 0)

ans = min(ans, now);

else if (now + A() < ans)

{

int i, j, temp, c;

for (temp = INF,i = R[0]; i; i = R[i])

{

if (temp > S[i])

{

temp = S[i];

c = i;

}

}

for (i = D[c]; i != c; i = D[i])

{

Remove(i);

for (j = R[i]; j != i; j = R[j])

Remove(j);

Dance(now + 1);

for (j = L[i]; j != i; j = L[j])

Resume(j);

Resume(i);

}

}

}

void Init(int m)

{

int i;

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

{

R[i] = i + 1;

L[i + 1] = i;

U[i] = D[i] = i;

S[i] = 0;

}

R[m] = 0;

size = m + 1;

}

**6.3 fibonacci knapsack 搜索背包**

#include<stdio.h>

#include<stdlib.h>

#include<algorithm>

#define MAXX 71

struct mono

{

long long weig,cost;

}goods[MAXX];

short n,T,t,i;

long long carry,sumw,sumc;

long long ans,las[MAXX];

int com(const void \*n,const void \*m)

{

struct mono \*a=(struct mono \*)n,\*b=(struct mono \*)m;

if(a->weig!=b->weig)

return a->weig-b->weig;

else

return b->cost-a->cost;

}

bool comp(const struct mono a,const struct mono b)

{

if(a.weig!=b.weig)

return a.weig<b.weig;

else

return b.cost<a.cost;

}

void dfs(short i,long long cost\_n,long long carry\_n,short last)

{

if(ans<cost\_n)

ans=cost\_n;

if(i==n || goods[i].weig>carry\_n || cost\_n+las[i]<=ans)

return;

if(last || (goods[i].weig!=goods[i-1].weig && goods[i].cost>goods[i-1].cost))

dfs(i+1,cost\_n+goods[i].cost,carry\_n-goods[i].weig,1);

dfs(i+1,cost\_n,carry\_n,0);

}

int main()

{

// freopen("asdf","r",stdin);

scanf("%hd",&T);

for(t=1;t<=T;++t)

{

scanf("%hd%lld",&n,&carry);

sumw=0;

sumc=0;

ans=0;

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

{

scanf("%lld%lld",&goods[i].weig,&goods[i].cost);

sumw+=goods[i].weig;

sumc+=goods[i].cost;

}

if(sumw<=carry)

{

printf("Case %hd: %lld\n",t,sumc);

continue;

}

// qsort(goods,n,sizeof(struct mono),com);

std::sort(goods,goods+n,comp);

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

{

// printf("%lld %lld\n",goods[i].weig,goods[i].cost);

las[i]=sumc;

sumc-=goods[i].cost;

}

dfs(0,0,carry,1);

printf("Case %hd: %lld\n",t,ans);

}

return 0;

}

**七、大数模板**

// header files

#include <cstdio>

#include <string>

#include <algorithm>

#include <iostream>

struct Bigint

{

// representations and structures

std::string a; // to store the digits

int sign; // sign = -1 for negative numbers, sign = 1 otherwise

// constructors

Bigint() {} // default constructor

Bigint( std::string b ) { (\*this) = b; } // constructor for std::string

// some helpful methods

int size() // returns number of digits

{

return a.size();

}

Bigint inverseSign() // changes the sign

{

sign \*= -1;

return (\*this);

}

Bigint normalize( int newSign ) // removes leading 0, fixes sign

{

for( int i = a.size() - 1; i > 0 && a[i] == '0'; i-- )

a.erase(a.begin() + i);

sign = ( a.size() == 1 && a[0] == '0' ) ? 1 : newSign;

return (\*this);

}

// assignment operator

void operator = ( std::string b ) // assigns a std::string to Bigint

{

a = b[0] == '-' ? b.substr(1) : b;

reverse( a.begin(), a.end() );

this->normalize( b[0] == '-' ? -1 : 1 );

}

// conditional operators

bool operator < ( const Bigint &b ) const // less than operator

{

if( sign != b.sign )

return sign < b.sign;

if( a.size() != b.a.size() )

return sign == 1 ? a.size() < b.a.size() : a.size() > b.a.size();

for( int i = a.size() - 1; i >= 0; i-- )

if( a[i] != b.a[i] )

return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];

return false;

}

bool operator == ( const Bigint &b ) const // operator for equality

{

return a == b.a && sign == b.sign;

}

// mathematical operators

Bigint operator + ( Bigint b ) // addition operator overloading

{

if( sign != b.sign )

return (\*this) - b.inverseSign();

Bigint c;

for(int i = 0, carry = 0; i<a.size() || i<b.size() || carry; i++ )

{

carry+=(i<a.size() ? a[i]-48 : 0)+(i<b.a.size() ? b.a[i]-48 : 0);

c.a += (carry % 10 + 48);

carry /= 10;

}

return c.normalize(sign);

}

Bigint operator - ( Bigint b ) // subtraction operator overloading

{

if( sign != b.sign )

return (\*this) + b.inverseSign();

int s = sign; sign = b.sign = 1;

if( (\*this) < b )

return ((b - (\*this)).inverseSign()).normalize(-s);

Bigint c;

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

{

borrow = a[i] - borrow - (i < b.size() ? b.a[i] : 48);

c.a += borrow >= 0 ? borrow + 48 : borrow + 58;

borrow = borrow >= 0 ? 0 : 1;

}

return c.normalize(s);

}

Bigint operator \* ( Bigint b ) // multiplication operator overloading

{

Bigint c("0");

for( int i = 0, k = a[i] - 48; i < a.size(); i++, k = a[i] - 48 )

{

while(k--)

c = c + b; // ith digit is k, so, we add k times

b.a.insert(b.a.begin(), '0'); // multiplied by 10

}

return c.normalize(sign \* b.sign);

}

Bigint operator / ( Bigint b ) // division operator overloading

{

if( b.size() == 1 && b.a[0] == '0' )

b.a[0] /= ( b.a[0] - 48 );

Bigint c("0"), d;

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

d.a += "0";

int dSign = sign \* b.sign;

b.sign = 1;

for( int i = a.size() - 1; i >= 0; i-- )

{

c.a.insert( c.a.begin(), '0');

c = c + a.substr( i, 1 );

while( !( c < b ) )

{

c = c - b;

d.a[i]++;

}

}

return d.normalize(dSign);

}

Bigint operator % ( Bigint b ) // modulo operator overloading

{

if( b.size() == 1 && b.a[0] == '0' )

b.a[0] /= ( b.a[0] - 48 );

Bigint c("0");

b.sign = 1;

for( int i = a.size() - 1; i >= 0; i-- )

{

c.a.insert( c.a.begin(), '0');

c = c + a.substr( i, 1 );

while( !( c < b ) )

c = c - b;

}

return c.normalize(sign);

}

// output method

void print()

{

if( sign == -1 )

putchar('-');

for( int i = a.size() - 1; i >= 0; i-- )

putchar(a[i]);

}

};

int main()

{

Bigint a, b, c; // declared some Bigint variables

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

// taking Bigint input //

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

std::string input; // std::string to take input

std::cin >> input; // take the Big integer as std::string

a = input; // assign the std::string to Bigint a

std::cin >> input; // take the Big integer as std::string

b = input; // assign the std::string to Bigint b

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

// Using mathematical operators //

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

c = a + b; // adding a and b

c.print(); // printing the Bigint

puts(""); // newline

c = a - b; // subtracting b from a

c.print(); // printing the Bigint

puts(""); // newline

c = a \* b; // multiplying a and b

c.print(); // printing the Bigint

puts(""); // newline

c = a / b; // dividing a by b

c.print(); // printing the Bigint

puts(""); // newline

c = a % b; // a modulo b

c.print(); // printing the Bigint

puts(""); // newline

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

// Using conditional operators //

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

if( a == b )

puts("equal"); // checking equality

else

puts("not equal");

if( a < b )

puts("a is smaller than b"); // checking less than operator

return 0;

}

**八、java**

//Scanner

Scanner in=new Scanner(new FileReader("asdf"));

PrintWriter pw=new PrintWriter(new Filewriter("out"));

boolean in.hasNext();

String in.next();

BigDecimal in.nextBigDecimal();

BigInteger in.nextBigInteger();

BigInteger in.nextBigInteger(int radix);

double in.nextDouble();

int in.nextInt();

int in.nextInt(int radix);

String in.nextLine();

long in.nextLong();

long in.nextLong(int radix);

short in.nextShort();

short in.nextShort(int radix);

int in.radix(); //Returns this scanner's default radix.

Scanner in.useRadix(int radix);// Sets this scanner's default radix to the specified radix.

void in.close();//Closes this scanner.

//String

char str.charAt(int index);

int str.compareTo(String anotherString); // <0 if less. ==0 if equal. >0 if greater.

int str.compareToIgnoreCase(String str);

String str.concat(String str);

boolean str.contains(CharSequence s);

boolean str.endsWith(String suffix);

boolean str.startsWith(String preffix);

boolean str.startsWith(String preffix,int toffset);

int str.hashCode();

int str.indexOf(int ch);

int str.indexOf(int ch,int fromIndex);

int str.indexOf(String str);

int str.indexOf(String str,int fromIndex);

int str.lastIndexOf(int ch);

int str.lastIndexOf(int ch,int fromIndex);

//(ry

int str.length();

String str.substring(int beginIndex);

String str.substring(int beginIndex,int endIndex);

String str.toLowerCase();

String str.toUpperCase();

String str.trim();// Returns a copy of the string, with leading and trailing whitespace omitted.

//StringBuilder

StringBuilder str.insert(int offset,...);

StringBuilder str.reverse();

void str.setCharAt(int index,int ch);

//BigInteger

compareTo(); equals(); doubleValue(); longValue(); hashCode(); toString(); toString(int radix); max(); min(); mod(); modPow(BigInteger exp,BigInteger m); nextProbablePrime(); pow();

andNot(); and(); xor(); not(); or(); getLowestSetBit(); bitCount(); bitLength(); setBig(int n); shiftLeft(int n); shiftRight(int n);

add(); divide(); divideAndRemainder(); remainder(); multiply(); subtract(); gcd(); abs(); signum(); negate();

//BigDecimal

movePointLeft(); movePointRight(); precision(); stripTrailingZeros(); toBigInteger(); toPlainString();

**九、Others**

chmod +x [filename]

while true; do

./gen > input

./sol < input > output.sol

./bf < input > output.bf

diff output.sol output.bf

if[ $? -ne 0];then break fi

done