```
-std=c++20 -Wall -Wl,--stack=1234567890 -O2 -fsanitize=undefined -fsanitize-
undefined-trap-on-error -D_GLIBCXX_DEBUG
```

计算几何

```cpp
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cassert>
#include<chrono>
#include<random>
#include<vector>
#include<functional>
#include<iomanip>
#include<algorithm>
using namespace std;
namespace Geometry
{
    const double eps=1e-12;
    const double PI=acos(-1);
    const double INF=1e18;
    bool equal(double a,double b)
    {
        return abs(a-b)<eps;
    }
    bool less(double a,double b)
    {
        return b-a>=eps;
    }
    bool greater(double a,double b)
    {
        return a-b>=eps;
    }
    bool less_equal(double a,double b)
    {
        return b-a>-eps;
    }
    bool greater_equal(double a,double b)
    {
        return a-b>-eps;
    }
    class Point
    {
    public:
        double x,y;
        Point(){x=0,y=0;}
        Point(const double &_x,const double &_y):x(_x),y(_y) {}
        friend Point operator * (const Point &a,const double &b)
        {
            return Point(a.x*b,a.y*b);
        }
        friend Point operator * (const double &a,const Point &b)
        {
            return Point(a*b.x,a*b.y);
        }
        friend Point operator / (const Point &a,const double &b)
        {
            return Point(a.x/b,a.y/b);
        }
        friend Point operator + (const Point &a,const Point &b)
        {
            return Point(a.x+b.x,a.y+b.y);
        }
        Point operator += (const Point &b)
        {
            x+=b.x,y+=b.y;
            return *this;
        }
        friend Point operator - (const Point &a,const Point &b)
        {
            return Point(a.x-b.x,a.y-b.y);
        }
        Point operator -= (const Point &b)
        {
            x-=b.x,y-=b.y;
            return *this;
        }
        friend double cross(const Point &a,const Point &b)
        {
            return a.x*b.y-a.y*b.x;
        }
        friend double dot(const Point &a,const Point &b)
        {
            return a.x*b.x+a.y*b.y;
        }
        friend bool operator == (const Point &a,const Point &b)
        {
            return equal(a.x,b.x)&&equal(a.y,b.y);
        }
        friend bool operator != (const Point &a,const Point &b)
        {
            return (!equal(a.x,b.x))||(!equal(a.y,b.y));
        }
        friend bool operator < (const Point &a,const Point &b)
        {
            if(equal(a.x,b.x)) return less(a.y,b.y);
            else return less(a.x,b.x);
        }
        friend bool operator > (const Point &a,const Point &b)
        {
            if(equal(a.x,b.x)) return greater(a.y,b.y);
            else return greater(a.x,b.x);
        }
        friend bool operator <= (const Point &a,const Point &b)
        {
            if(equal(a.x,b.x)) return less_equal(a.y,b.y);
            else return less_equal(a.x,b.x);
        }
        friend bool operator >= (const Point &a,const Point &b)
        {
            if(equal(a.x,b.x)) return greater_equal(a.y,b.y);
            else return greater_equal(a.x,b.x);
        }
        Point operator - ()const
        {
            return Point(-x,-y);
        }
        double length()const
        {
            return sqrt(x*x+y*y);
        }
        Point unit()const
        {
            return *this/length();
        }
        double angle()const
        {
            return atan2(y,x);
        }
        int quadrant()const
        {
            if(x>0&&y>=0) return 1;
            else if(x<=0&&y>0) return 2;
            else if(x<0&&y<=0) return 3;
            else if(x>=0&&y<0) return 4;
            else return 0;
        }
        friend double angle(const Point &a,const Point &b)
        {
            return atan2(cross(a,b),dot(a,b));
        }
        Point rotate(const double &theta)const
        {
            return Point(x*cos(theta)-y*sin(theta),x*sin(theta)+y*cos(theta));
        }
        friend istream &operator>>(istream &in,Point &obj)
        {
            in>>obj.x>>obj.y;
            return in;
        }
        friend ostream &operator<<(ostream &out,const Point &obj)
        {
            out<<obj.x<<" "<<obj.y;
            return out;
        }
    };
    double distance(const Point &a,const Point &b)
    {
        return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
    }
    enum Direction
    {
        COUNTER_CLOCKWISE,
        CLOCKWISE,
        ONLINE_BACK,
        ONLINE_FRONT,
        ON_SEGMENT
    };
    istream& operator>>(istream& in,Direction& direction)
    {
        string value;
        in>>value;
        if(value=="COUNTER_CLOCKWISE") direction=COUNTER_CLOCKWISE;
        else if(value=="CLOCKWISE") direction=CLOCKWISE;
        else if(value=="ONLINE_BACK") direction=ONLINE_BACK;
        else if(value=="ONLINE_FRONT") direction=ONLINE_FRONT;
        else if(value=="ON_SEGMENT") direction=ON_SEGMENT;
        else in.setstate(ios::failbit);
        return in;
    }
    ostream& operator<<(ostream& out,const Direction& direction)
    {
        if(direction==COUNTER_CLOCKWISE) out<<"COUNTER_CLOCKWISE";
        else if(direction==CLOCKWISE) out<<"CLOCKWISE";
        else if(direction==ONLINE_BACK) out<<"ONLINE_BACK";
        else if(direction==ONLINE_FRONT) out<<"ONLINE_FRONT";
        else if(direction==ON_SEGMENT) out<<"ON_SEGMENT";
        return out;
    }
    class Line
    {
    public:
        Point a,b;
        Line(){}
        Line(const Point &_a,const Point &_b):a(_a),b(_b){}
        Point projection(const Point &p)const
        {
            return a+(b-a).unit()*(dot(p-a,b-a)/(b-a).length());
        }
        Point reflection(const Point &p)const
        {
            return projection(p)*2-p;
        }
        Direction direction(const Point &p)const
        {
            double t=cross(b-a,p-a);
            if(greater(t,0)) return COUNTER_CLOCKWISE;
            if(less(t,0)) return CLOCKWISE;
            double l1=dot(p-a,b-a);
            if(less(l1,0)) return ONLINE_BACK;
            double l2=dot(b-a,b-a);
            if(l1>l2) return ONLINE_FRONT;
            else return ON_SEGMENT;
        }
        double distance(const Point &p)const
        {
            Point u=projection(p);
            if(direction(u)==ON_SEGMENT) return Geometry::distance(u,p);
            else return min(Geometry::distance(a,p),Geometry::distance(b,p));
        }
        Point middle_point()const
        {
            return (a+b)/2;
        }
        Line perpendicular_bisector()const
        {
            Point p=middle_point();
            return Line(p,p+(b-a).rotate(PI/2));
        }
        double length()const
        {
            return Geometry::distance(a,b);
        }
        friend istream &operator>>(istream &in,Line &obj)
        {
            in>>obj.a>>obj.b;
            return in;
        }
        friend ostream &operator<<(ostream &out,const Line &obj)
        {
            out<<obj.a<<" "<<obj.b;
            return out;
        }
    };
    bool parallel(const Line &x,const Line &y)
    {
        return equal(cross(x.b-x.a,y.b-y.a),0);
    }
    bool orthogonal(const Line &x,const Line &y)
    {
        return equal(dot(x.b-x.a,y.b-y.a),0);
    }
    vector<Point> cross_point(const Line &x,const Line &y)
    {
        if(parallel(x,y)) return {};
        Point u=x.a-y.a,v=x.b-x.a,w=y.b-y.a;
        double t=cross(w,u)/cross(v,w);
        return {x.a+t*v};
    }
    int sign(double x)
    {
        return greater(x,0)-less(x,0);
    }
    bool intersection(const Line &x,const Line &y)
    {
```

```
if(x.direction(y.a)==ON_SEGMENT||x.direction(y.b)==ON_SEGMENT||y.direction(x.a)==O
N_SEGMENT||y.direction(x.b)==ON_SEGMENT) return true;
        return sign(cross(x.b-x.a,y.a-x.a))*sign(cross(x.b-x.a,y.b-x.a))
<0&&sign(cross(y.b-y.a,x.a-y.a))*sign(cross(y.b-y.a,x.b-y.a))<0;
    }
    double distance(const Line &x,const Line &y)
    {
        if(intersection(x,y)) return 0;
        else return
min({x.distance(y.a),x.distance(y.b),y.distance(x.a),y.distance(x.b)});
    }
    const int IN=2,ON=1,OUT=0;
    mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());
    class Polygon
    {
    private:
        vector<Point>g;
    public:
        Polygon(){}
        Polygon(const int &n){g.resize(n);}
        Polygon(const vector<Point> &f):g(f){}
        void clear()
        {
            g.clear();
        }
        void resize(int n)
        {
            g.resize(n);
        }
        void push_back(const Point &x)
        {
            return g.push_back(x);
        }
        void push_back(const vector<Point> &x)
        {
            for(const Point &p:x)
                g.push_back(p);
            return;
        }
        void pop_back()
        {
            return g.pop_back();
        }
        Point& front()
        {
            return g.front();
        }
        const Point& front()const
        {
            return g.front();
        }
        Point& back()
        {
            return g.back();
        }
        const Point& back()const
        {
            return g.back();
        }
        size_t size()const
        {
            return g.size();
        }
        Point& operator [](const int &i)
        {
            return g[i];
        }
        const Point& operator [](const int &i)const
        {
            return g[i];
        }
        vector<Point>::iterator begin()
        {
            return g.begin();
        }
        vector<Point>::iterator end()
        {
            return g.end();
        }
        vector<Point>::const_iterator begin()const
        {
            return g.begin();
        }
        vector<Point>::const_iterator end()const
        {
            return g.end();
        }
        vector<Point>::reverse_iterator rbegin()
        {
            return g.rbegin();
        }
        vector<Point>::reverse_iterator rend()
        {
            return g.rend();
        }
        vector<Point>::const_reverse_iterator rbegin()const
        {
            return g.rbegin();
        }
        vector<Point>::const_reverse_iterator rend()const
        {
            return g.rend();
        }
        double area()const
        {
            int n=g.size();
            double res=0;
            for(int i=0;i<n;i++)
                res+=cross(g[i],g[(i+1)%n]);
            res/=2;
            return abs(res);
        }
        double perimeter()const
        {
            int n=g.size();
            double sum=0;
            for(int i=0;i<n;i++)
                sum+=distance(g[i],g[(i+1)%n]);
            return sum;
        }
        bool is_convex()const
        {
            int n=g.size();
            for(int i=0;i<n;i++)
                if(less(cross(g[(i+1)%n]-g[i],g[(i-1+n)%n]-g[i]),0)) return false;
            return true;
        }
        int point_containment(const Point &a)const
        {
            int n=g.size();
            for(int i=0;i<n;i++)
                if(Line(g[i],g[(i+1)%n]).direction(a)==ON_SEGMENT) return ON;
            function<bool(const Line &)> check=[=](const Line &l)
            {
                for(int i=0;i<n;i++)
                    if(parallel(l,Line(g[i],g[(i+1)%n]))) return false;
                for(int i=0;i<n;i++)

if(l.direction(g[i])==ON_SEGMENT||l.direction(g[i])==ONLINE_FRONT||l.direction(g[i
])==ONLINE_BACK) return false;
                return true;
            };
            Line l=Line(a,Point(rnd(),rnd()));
            while(!check(l))
                l=Line(a,Point(rnd(),rnd()));
            int s=0;
            for(int i=0;i<n;i++)
                if(intersection(l,Line(g[i],g[(i+1)%n]))) s++;
            if(s&1) return IN;
            else return OUT;
        }
        double convex_diamater()const
        {
            int n=g.size();
            double ans=0;
            for(int i=0,j=0;i<n;i++)
            {
                while(less(cross(g[i]-g[j],g[(i+1)%n]-g[j]),cross(g[i]-
g[(j+1)%n],g[(i+1)%n]-g[(j+1)%n]))) j=(j+1)%n;
                ans=max(ans,max(distance(g[j],g[i]),distance(g[j],g[(i+1)%n])));
            }
            return ans;
        }
        pair<Polygon,Polygon> convex_cut(const Line &l)const
        {
            Polygon res1,res2;
            int n=g.size();
            for(int i=0;i<(int)g.size();i++)
            {
                Point u=g[i],v=g[(i+1)%n];
                if(greater_equal(cross(l.b-l.a,u-l.a),0))
                {
                    res1.push_back(u);
                    if(less(cross(l.b-l.a,v-l.a),0))
res1.push_back(cross_point(Line(u,v),l));
                }
                else if(greater(cross(l.b-l.a,v-l.a),0))
res1.push_back(cross_point(Line(u,v),l));
            }
            for(int i=0;i<(int)g.size();i++)
            {
                Point u=g[i],v=g[(i+1)%n];
                if(greater_equal(cross(l.a-l.b,u-l.b),0))
                {
                    res2.push_back(u);
                    if(less(cross(l.a-l.b,v-l.b),0))
res2.push_back(cross_point(Line(u,v),l));
                }
                else if(greater(cross(l.a-l.b,v-l.b),0))
res2.push_back(cross_point(Line(u,v),l));
            }
            return make_pair(res1,res2);
        }
        Polygon kernel()const;
    };
    Polygon convex_hull(const vector<Point> &_p)
    {
        vector<Point> p=_p;
        int n=p.size();
        if(n<=2)
        {
            sort(p.begin(),p.end(),[](const Point &a,const Point &b){return
a.y==b.y?a.x<b.x:a.y<b.y;});
            Polygon res;
            for(const Point &q:p)
                res.push_back(q);
            return res;
        }
        sort(p.begin(),p.end(),[](const Point &a,const Point &b){return a.x==b.x?
a.y<b.y:a.x<b.x;});
        vector<int>stk;
        int top=0;
        for(int i=0;i<n;i++)
        {
            while(top>=2&&less_equal(cross(p[stk[top-1]]-p[stk[top-2]],p[i]-
p[stk[top-1]]),0)) stk.pop_back(),top--;
            stk.emplace_back(i),top++;
        }
        int tmp=top;
        for(int i=n-2;i>=0;i--)
        {
            while(top>tmp&&less_equal(cross(p[stk[top-1]]-p[stk[top-2]],p[i]-
p[stk[top-1]]),0)) stk.pop_back(),top--;
            stk.emplace_back(i),top++;
        }
        stk.pop_back(),top--;
        vector<Point>hull;
        for(int i=0;i<top;i++)
            hull.emplace_back(p[stk[i]]);
        int t=min_element(hull.begin(),hull.end(),[](const Point &a,const Point
&b){return a.y==b.y?a.x<b.x:a.y<b.y;})-hull.begin();
        Polygon res;
        for(int i=t;i<top;i++)
            res.push_back(hull[i]);
        for(int i=0;i<t;i++)
            res.push_back(hull[i]);
        return res;
    }
    Polygon non_strictly_convex_hull(const vector<Point> &_p)
    {
        vector<Point> p=_p;
        int n=p.size();
        if(n<=2)
        {
            sort(p.begin(),p.end(),[](const Point &a,const Point &b){return
a.y==b.y?a.x<b.x:a.y<b.y;});
            Polygon res;
            for(const Point &q:p)
                res.push_back(q);
            return res;
        }
        sort(p.begin(),p.end(),[](const Point &a,const Point &b){return a.x==b.x?
a.y<b.y:a.x<b.x;});
        vector<int>stk;
        int top=0;
        for(int i=0;i<n;i++)
        {
            while(top>=2&&less(cross(p[stk[top-1]]-p[stk[top-2]],p[i]-p[stk[top-
1]]),0)) stk.pop_back(),top--;
            stk.emplace_back(i),top++;
        }
        int tmp=top;
        for(int i=n-2;i>=0;i--)
        {
            while(top>tmp&&less(cross(p[stk[top-1]]-p[stk[top-2]],p[i]-p[stk[top-
1]]),0)) stk.pop_back(),top--;
            stk.emplace_back(i),top++;
        }
        stk.pop_back(),top--;
        vector<Point>hull;
        for(int i=0;i<top;i++)
            hull.emplace_back(p[stk[i]]);
        int t=min_element(hull.begin(),hull.end(),[](const Point &a,const Point
&b){return a.y==b.y?a.x<b.x:a.y<b.y;})-hull.begin();
        Polygon res;
        for(int i=t;i<top;i++)
            res.push_back(hull[i]);
        for(int i=0;i<t;i++)
            res.push_back(hull[i]);
        return res;
    }
    Polygon minkowski_sum(const vector<Point> &a,const vector<Point> &b)
    {
        assert(a.size()!=0&&b.size()!=0);
        Polygon ca=convex_hull(a),cb=convex_hull(b);
        int na=ca.size(),nb=cb.size();
        vector<Point>la,lb;
        for(int i=0;i<na;i++)
            la.emplace_back(ca[(i+1)%na]-ca[i]);
        for(int i=0;i<nb;i++)
            lb.emplace_back(cb[(i+1)%nb]-cb[i]);
        int pa=0,pb=0;
        vector<Point> l;
        l.emplace_back(ca[0]+cb[0]);
        while(pa<(int)la.size()&&pb<(int)lb.size())
        {
            double val=cross(la[pa],lb[pb]);
            if(greater(val,0)) l.emplace_back(l.back()+la[pa]),pa++;
            else if(less(val,0)) l.emplace_back(l.back()+lb[pb]),pb++;
            else l.emplace_back(l.back()+la[pa]+lb[pb]),pa++,pb++;
        }
        while(pa<(int)la.size())
            l.emplace_back(l.back()+la[pa]),pa++;
        while(pb<(int)lb.size())
            l.emplace_back(l.back()+lb[pb]),pb++;
        Polygon res=convex_hull(l);
        return res;
    }
    Polygon half_plane_intersection(const vector<Line> &l,double x1=-INF,double
y1=-INF,double x2=INF,double y2=INF)
    {
        vector<pair<double,Line>>f;
        for(int i=0;i<(int)l.size();i++)
            f.emplace_back((l[i].b-l[i].a).angle(),l[i]);
        f.emplace_back(0,Line(Point(x1,y1),Point(x2,y1)));
        f.emplace_back(PI/2,Line(Point(x2,y1),Point(x2,y2)));
        f.emplace_back(PI,Line(Point(x2,y2),Point(x1,y2)));
        f.emplace_back(-PI/2,Line(Point(x1,y2),Point(x1,y1)));
        int n=f.size();
        sort(f.begin(),f.end(),[](const pair<double,Line> &a,const
pair<double,Line> &b){return !equal(a.first,b.first)?
a.first<b.first:a.second.direction(b.second.a)==CLOCKWISE;});
        vector<Line>Ql(n);
        vector<Point>Qp(n);
        Polygon res;
        int head=0,tail=-1;
        Ql[++tail]=f[0].second;
        for(int i=1;i<n;i++)
            if(!equal(f[i].first,f[i-1].first))
            {
                while(head<tail&&f[i].second.direction(Qp[tail-1])==CLOCKWISE)
tail--;
                while(head<tail&&f[i].second.direction(Qp[head])==CLOCKWISE)
head++;
                Ql[++tail]=f[i].second;
                if(head<tail)
                {
                    vector<Point> tmp=cross_point(Ql[tail],Ql[tail-1]);
                    if(!tmp.empty()) Qp[tail-1]=tmp[0];
                    else return res;
                }
            }
        while(head<tail&&Ql[head].direction(Qp[tail-1])==CLOCKWISE) tail--;
        while(head<tail&&Ql[tail].direction(Qp[head])==CLOCKWISE) head++;
        vector<Point> tmp=cross_point(Ql[tail],Ql[head]);
        if(tmp.empty()||tail-head+1<=2) return res;
        for(int i=head;i<tail;i++)
            res.push_back(Qp[i]);
        res.push_back(tmp[0]);
        return res;
    }
    Polygon Polygon::kernel()const
    {
        int n=g.size();
        vector<Line>l;
        for(int i=0;i<n;i++)
            l.emplace_back(Line(g[i],g[(i+1)%n]));
        return half_plane_intersection(l);
    }
    double closest_pair(const vector<Point> &_p)
    {
        vector<Point>p=_p;
        sort(p.begin(),p.end(),[](const Point &a,const Point &b){return
a.x<b.x;});
        function<double(const int &,const int &)> solve=[&](const int &l,const int
&r)
        {
            if(r-l+1<=1) return INF;
            if(r-l+1<=7)
            {
                double ans=INF;
                sort(p.begin()+l,p.begin()+r+1,[](const Point &a,const Point &b)
{return a.y<b.y;});
                for(int i=l;i<=r;i++)
                    for(int j=i+1;j<=r;j++)
                        ans=min(ans,distance(p[i],p[j]));
                return ans;
            }
            int mid=(l+r)/2;
            double w=p[mid].x;
            double d=min(solve(l,mid),solve(mid+1,r));
            inplace_merge(p.begin()+l,p.begin()+mid+1,p.begin()+r+1,[](const Point
&a,const Point &b){return a.y<b.y;});
            vector<Point>q;
            for(int i=l;i<=r;i++)
                if(abs(w-p[i].x)<=d) q.emplace_back(p[i]);
            for(int i=0,j=0;i<(int)q.size();i++)
            {
                while(j<(int)q.size()&&q[j].y<=q[i].y+d) j++;
                for(int k=i+1;k<j;k++)
                    d=min(d,distance(q[i],q[k]));
            }
            return d;
        };
        return solve(0,p.size()-1);
    }
    class Circle
    {
    public:
        Point o;
        double r;
        Circle(){}
        Circle(const Point &_o,const double &_r):o(_o),r(_r){}
        friend istream &operator>>(istream &in,Circle &obj)
        {
            in>>obj.o>>obj.r;
            return in;
        }
        friend ostream &operator<<(ostream &out,const Circle &obj)
        {
            out<<obj.o<<" "<<obj.r;
            return out;
        }
        friend bool operator==(const Circle &a,const Circle &b)
        {
            return a.o==b.o&&equal(a.r,b.r);
        }
        friend bool operator!=(const Circle &a,const Circle &b)
        {
            return a.o!=b.o||(!equal(a.r,b.r));
        }
        double area()const
        {
            return PI*r*r;
        }
        bool is_tangent(const Line &l)const
        {
            return equal(Geometry::distance(l.projection(o),o),r);
        }
        int point_containment(const Point &p)const
        {
```

```
        double d=distance(o,p);
        if(equal(d,r)) return ON;
        else if(less(d,r)) return IN;
        else return OUT;
    }
    vector<Point>cross_point(const Line &l)const
    {
        Point pr=l.projection(o),e=(l.b-l.a).unit();
        double d=distance(pr,o);
        if(greater(d,r)) return {};
        double t=sqrt(r*r-distance(pr,o)*distance(pr,o));
        if(equal(t,0)) return {pr};
        else return {pr-e*t,pr+e*t};
    }
    vector<Point>cross_point(const Circle &c)const
    {
        double d=distance(o,c.o);
        if(less(d,abs(r-c.r))||greater(d,r+c.r)) return {};
        double x=(r*r-c.r*c.r+d*d)/(d*2),h=sqrt(r*r-x*x);
        Point p=o+(c.o-o).unit()*x;
        if(equal(d,abs(r-c.r))||equal(d,r+c.r)) return {p};
        Point v=(c.o-o).unit().rotate(PI/2)*h;
        return {p-v,p+v};
    }
    vector<Point>tangent(const Point &p)const
    {
        double d=distance(o,p);
        if(greater(r,d)) return {};
        if(equal(d,r)) return {p};
        return cross_point(Circle(p,sqrt(d*d-r*r)));
    }
    vector<Line>common_tangent_out(const Circle &c)const
    {
        assert(*this!=c);
        if(equal(r,c.r))
        {
            Point p=(c.o-o).unit().rotate(PI/2)*r;
            return {Line(o-p,c.o-p),Line(o+p,c.o+p)};
        }
        double d=distance(o,c.o);
        if(less(d,abs(r-c.r))) return {};
        if(equal(d,abs(r-c.r)))
        {
            Point p;
            if(r>c.r) p=o+(c.o-o).unit()*r;
            else p=c.o+(o-c.o).unit()*c.r;
            return {Line(p,p)};
        }
        Point p((o.x*c.r-c.o.x*r)/(c.r-r),(o.y*c.r-c.o.y*r)/(c.r-r));
        vector<Point>p1=tangent(p),p2=c.tangent(p);
        assert((int)p1.size()==2&&(int)p2.size()==2);
        return {Line(p1[0],p2[0]),Line(p1[1],p2[1])};
    }
    vector<Line>common_tangent_in(const Circle &c)const
```

```
        {
            assert(*this!=c);
            double d=distance(o,c.o);
            if(less(d,abs(r+c.r))) return {};
            if(equal(d,abs(r+c.r)))
            {
                Point p=o+(c.o-o).unit()*r;
                return {Line(p,p)};
            }
            Point p((o.x*c.r+c.o.x*r)/(r+c.r),(o.y*c.r+c.o.y*r)/(r+c.r));
            vector<Point>p1=tangent(p),p2=c.tangent(p);
            assert((int)p1.size()==2&&(int)p2.size()==2);
            return {Line(p1[0],p2[0]),Line(p1[1],p2[1])};
        }
        vector<Line>common_tangent(const Circle &c)const
        {
            assert(*this!=c);
            vector<Line>f=common_tangent_out(c),g=common_tangent_in(c);
            for(const Line &l:g)
                f.emplace_back(l);
            g.clear();
            sort(f.begin(),f.end(),[](const Line &x,const Line &y){return
x.a.x<y.a.x||(x.a.x==x.a.x&&x.a.y<x.a.y);});
            return f;
        }
        double intersection_area(const Point &a,const Point &b)const
        {
            bool ta=less_equal(distance(o,a),r),tb=less_equal(distance(o,b),r);
            if(ta&&tb) return cross(a-o,b-o)/2;
            vector<Point>t=cross_point(Line(b,a));
            if(ta&&!tb) return angle(t.front()-o,b-o)*r*r/2+cross(a-o,t.front()-
o)/2;
            if(!ta&&tb) return angle(a-o,t.back()-o)*r*r/2+cross(t.back()-o,b-
o)/2;
            double s=angle(a-o,b-o)*r*r/2;
            if(greater_equal(Line(a,b).distance(o),r)) return s;
            return s+angle(t.front()-o,t.back()-o)*r*r/2-cross(t.front()-
o,t.back()-o)/2;
        }
        double intersection_area(const Polygon &g)const
        {
            int n=g.size();
            double s=0;
            for(int i=0;i<n;i++)
                s+=intersection_area(g[i],g[(i+1)%n]);
            return s;
        }
        double intersection_area(const Circle &c)const
        {
            double d=distance(o,c.o);
            if(greater(d,r+c.r)) return 0;
            if(less_equal(d,abs(r-c.r))) return min(area(),c.area());
            vector<Point>t=cross_point(c);
            double alpha=acos((d*d+r*r-c.r*c.r)/(2*d*r))*2,beta=acos((d*d+c.r*c.r-
r*r)/(2*d*c.r))*2;
            double
s1=alpha*r*r/2,s2=beta*c.r*c.r/2,s3=sin(alpha)*r*r/2+sin(beta)*c.r*c.r/2;
            return s1+s2-s3;
        }
    };
    const int SEPARATED=4,CIRCUMSCRIBED=3,INTERSECTED=2,INSCRIBED=1,INCLUDED=0;
    int intersection(const Circle &a,const Circle &b)
    {
        double d=distance(a.o,b.o);
        if(greater(d,a.r+b.r)) return SEPARATED;
        else if(equal(d,a.r+b.r)) return CIRCUMSCRIBED;
        else if(greater(d,abs(a.r-b.r))) return INTERSECTED;
        else if(equal(d,abs(a.r-b.r))) return INSCRIBED;
        else return INCLUDED;
    }
    class Triangle
    {
    public:
        Point A,B,C;
        Triangle(){}
        Triangle(const Point &_A,const Point &_B,const Point
&_C):A(_A),B(_B),C(_C){}
        friend istream &operator>>(istream &in,Triangle &obj)
        {
            in>>obj.A>>obj.B>>obj.C;
            return in;
        }
        friend ostream &operator<<(ostream &out,const Triangle &obj)
        {
            out<<obj.A<<" "<<obj.B<<" "<<obj.C;
            return out;
        }
        Circle inscribed_circle()const
        {
            double a=distance(B,C),b=distance(A,C),c=distance(A,B);
            double p=(a+b+c)/2;
            double s=abs(cross(B-A,C-A))/2;
            double r=s/p;
            Point o((a*A.x+b*B.x+c*C.x)/(a+b+c),(a*A.y+b*B.y+c*C.y)/(a+b+c));
            return Circle(o,r);
        }
        Circle circumscribed_circle()const
        {
            double t1=A.x*A.x+A.y*A.y;
            double t2=B.x*B.x+B.y*B.y;
            double t3=C.x*C.x+C.y*C.y;
            double t=A.x*B.y+B.x*C.y+C.x*A.y-A.x*C.y-B.x*A.y-C.x*B.y;
            Point o((t2*C.y+t1*B.y+t3*A.y-t2*A.y-t3*B.y-t1*C.y)/t/2,
(t3*B.x+t2*A.x+t1*C.x-t1*B.x-t2*C.x-t3*A.x)/t/2);
            double a=distance(B,C),b=distance(A,C),c=distance(A,B);
            double s=abs(cross(B-A,C-A))/2;
            double r=a*b*c/(4*s);
            return Circle(o,r);
        }
    };
    Circle smallest_enclosing_circle(const vector<Point> &_p)
    {
        vector<Point>p=_p;
        shuffle(p.begin(),p.end(),rnd);
        int n=p.size();
        Circle c=Circle(Point(0,0),0);
        for(int i=0;i<n;i++)
            if(c.point_containment(p[i])==OUT)
            {
                c=Circle(p[i],0);
                for(int j=0;j<i;j++)
                    if(c.point_containment(p[j])==OUT)
                    {
                        c=Circle((p[i]+p[j])/2,distance(p[i],p[j])/2);
                        for(int k=0;k<j;k++)
                            if(c.point_containment(p[k])==OUT)
                                c=Triangle(p[i],p[j],p[k]).circumscribed_circle();
                    }
            }
        return c;
    }
}
using namespace Geometry;
```

网络流

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<queue>
using namespace std;
struct Max_Flow
{
    static const int N=105,M=5005;
    static const long long INF=4557430888798830399;
    struct Edge
    {
        int to,next;
        long long flow;
    }edge[M*2];
    int cur[N],head[N],cnt;
    int tot;
    Max_Flow():cnt(1),tot(0)
    {
        memset(head,0,sizeof(head));
    }
    void add_edge(int u,int v,long long f)
    {
        cnt++;
        edge[cnt].to=v;
        edge[cnt].flow=f;
        edge[cnt].next=head[u];
        head[u]=cnt;
        return;
    }
    void add(int u,int v,long long f)
    {
        add_edge(u,v,f);
        add_edge(v,u,0);
        return;
    }
    int dep[N];
    bool bfs(int s,int t)
    {
        for(int i=1;i<=tot;i++)
            dep[i]=-1;
        queue<int>q;
        q.push(s);
        dep[s]=0;
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            for(int i=head[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(dep[v]!=-1||edge[i].flow<=0) continue;
                dep[v]=dep[u]+1;
                q.push(v);
            }
        }
        return dep[t]!=-1;
    }
    long long dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return flow;
        long long used=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(dep[v]!=dep[u]+1||edge[i].flow<=0) continue;
            long long now=dfs(v,t,min(flow,(long long)edge[i].flow));
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        return used;
    }
    long long dinic(int s,int t)
    {
        long long res=0;
        while(bfs(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            res+=dfs(s,t,INF);
        }
        return res;
    }
}max_flow;
struct Min_Cost_Max_Flow
{
    static const int N=405,M=15005;
    static const long long INF=4557430888798830399;
    struct Edge
    {
        int to,next;
        int cost;
        long long flow;
    }edge[M*2];
    int cur[N],head[N],cnt;
    int tot;
    Min_Cost_Max_Flow():cnt(1),tot(0)
    {
        memset(head,0,sizeof(head));
    }
    void add_edge(int u,int v,int c,long long f)
    {
        cnt++;
        edge[cnt].to=v;
        edge[cnt].cost=c;
        edge[cnt].flow=f;
        edge[cnt].next=head[u];
        head[u]=cnt;
        return;
    }
    void add(int u,int v,int c,long long f)
    {
        add_edge(u,v,c,f);
        add_edge(v,u,-c,0);
        return;
    }
    long long dis[N];
    bool spfa(int s,int t)
    {
        static bool vis[N];
        for(int i=1;i<=tot;i++)
            vis[i]=false;
        for(int i=1;i<=tot;i++)
            dis[i]=INF;
        queue<int>q;
        vis[s]=true;
        dis[s]=0;
        q.push(s);
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            vis[u]=false;
            for(int i=head[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(edge[i].flow<=0) continue;
                if(dis[v]>dis[u]+edge[i].cost)
                {
                    dis[v]=dis[u]+edge[i].cost;
                    if(!vis[v])
                    {
                        vis[v]=true;
                        q.push(v);
                    }
                }
            }
        }
        return dis[t]!=INF;
    }
    bool book[N];
    pair<long long,long long> dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return make_pair(flow,0);
        book[u]=true;
        long long used=0,res=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(book[v]||dis[v]!=dis[u]+edge[i].cost||edge[i].flow<=0) continue;
            pair<long long,long long>val=dfs(v,t,min(flow,edge[i].flow));
            long long now=val.first;
            res+=val.second+now*edge[i].cost;
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        book[u]=false;
        return make_pair(used,res);
    }
    pair<long long,long long> ssp(int s,int t)
    {
        long long ans=0,cost=0;
        for(int i=1;i<=tot;i++)
            book[i]=false;
        while(spfa(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            pair<long long,long long> res=dfs(s,t,INF);
```

```
                ans+=res.first,cost+=res.second;
            }
            return make_pair(ans,cost);
        }
    }min_cost_max_flow;
    struct Min_Cost_Feasible_Flow
    {
        static const int N=405,M=15005;
        static const long long INF=4557430888798830399;
        struct Edge
        {
            int to,next;
            int cost;
            long long flow;
        }edge[M*2];
        int cur[N],head[N],cnt;
        int tot;
        Min_Cost_Feasible_Flow():cnt(1),tot(0)
        {
            memset(head,0,sizeof(head));
        }
        void add_edge(int u,int v,int c,long long f)
        {
            cnt++;
            edge[cnt].to=v;
            edge[cnt].cost=c;
            edge[cnt].flow=f;
            edge[cnt].next=head[u];
            head[u]=cnt;
            return;
        }
        void add(int u,int v,int c,long long f)
        {
            add_edge(u,v,c,f);
            add_edge(v,u,-c,0);
            return;
        }
        long long dis[N];
        bool vis[N];
        bool spfa(int s,int t)
        {
            for(int i=1;i<=tot;i++)
                vis[i]=false;
            for(int i=1;i<=tot;i++)
                dis[i]=INF;
            queue<int>q;
            vis[s]=true;
            dis[s]=0;
            q.push(s);
            while(!q.empty())
            {
                int u=q.front();
                q.pop();
                vis[u]=false;
                for(int i=head[u];i;i=edge[i].next)
                {
                    int v=edge[i].to;
                    if(edge[i].flow<=0) continue;
                    if(dis[v]>dis[u]+edge[i].cost)
                    {
                        dis[v]=dis[u]+edge[i].cost;
                        if(!vis[v])
                        {
                            vis[v]=true;
                            q.push(v);
                        }
                    }
                }
            }
            return dis[t]!=INF;
        }
        bool book[N];
        pair<long long,long long> dfs(int u,int t,long long flow)
        {
            if(u==t||flow==0) return make_pair(flow,0);
            book[u]=true;
            long long used=0,res=0;
            for(int &i=cur[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(book[v]||dis[v]!=dis[u]+edge[i].cost||edge[i].flow<=0) continue;
                pair<long long,long long>val=dfs(v,t,min(flow,edge[i].flow));
                long long now=val.first;
                res+=val.second+now*edge[i].cost;
                flow-=now;
                edge[i].flow-=now;
                edge[i^1].flow+=now;
                used+=now;
                if(flow==0) break;
            }
            book[u]=false;
            return make_pair(used,res);
        }
        pair<long long,long long> ssp(int s,int t)
        {
            long long ans=0,cost=0;
            for(int i=1;i<=tot;i++)
                book[i]=false;
            while(spfa(s,t))
            {
                for(int i=1;i<=tot;i++)
                    cur[i]=head[i];
                pair<long long,long long> res=dfs(s,t,INF);
                if(cost<0) break;
                ans+=res.first,cost+=res.second;
            }
            return make_pair(ans,cost);
        }
```

```
}min_cost_feasible_flow;
struct Bounded_Feasible_Flow_Without_Source_Sink
{
    static const int N=205,M=10205;
    static const long long INF=4557430888798830399;
    struct Edge
    {
        int to,next;
        long long flow;
    }edge[M*2+N*2];
    int cur[N],head[N],cnt;
    long long extra[N];
    vector<long long>flow;
    int tot;
    Bounded_Feasible_Flow_Without_Source_Sink():cnt(1),tot(0)
    {
        memset(head,0,sizeof(head));
        memset(extra,0,sizeof(extra));
    }
    void add_edge(int u,int v,long long f)
    {
        cnt++;
        edge[cnt].to=v;
        edge[cnt].flow=f;
        edge[cnt].next=head[u];
        head[u]=cnt;
        return;
    }
    void add(int u,int v,long long f)
    {
        add_edge(u,v,f);
        add_edge(v,u,0);
        return;
    }
    void add(int u,int v,long long lower,long long upper)
    {
        add(u,v,upper-lower);
        extra[v]+=lower,extra[u]-=lower;
        flow.emplace_back(lower);
        return;
    }
    int dep[N];
    bool bfs(int s,int t)
    {
        for(int i=1;i<=tot;i++)
            dep[i]=-1;
        queue<int>q;
        q.push(s);
        dep[s]=0;
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            for(int i=head[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(dep[v]!=-1||edge[i].flow<=0) continue;
                dep[v]=dep[u]+1;
                q.push(v);
            }
        }
        return dep[t]!=-1;
    }
    long long dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return flow;
        long long used=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(dep[v]!=dep[u]+1||edge[i].flow<=0) continue;
            long long now=dfs(v,t,min(flow,(long long)edge[i].flow));
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        return used;
    }
    long long dinic(int s,int t)
    {
        long long res=0;
        while(bfs(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            res+=dfs(s,t,INF);
        }
        return res;
    }
    vector<long long> solve()
    {
        int s=++tot,t=++tot;
        long long sum=0;
        for(int i=1;i<=tot-2;i++)
            if(extra[i]>0)
            {
                sum+=extra[i];
                add(s,i,extra[i]);
            }
            else if(extra[i]<0)
            {
                add(i,t,-extra[i]);
            }
        if(dinic(s,t)!=sum) return {};
        for(int i=0;i<(int)flow.size();i++)
            flow[i]+=edge[i*2+3].flow;
        return flow;
    }
}bounded_feasible_flow_without_source_sink;
struct Bounded_Feasible_Flow_With_Source_Sink
{
    static const int N=205,M=10205;
    static const long long INF=4557430888798830399;
    struct Edge
    {
        int to,next;
        long long flow;
    }edge[M*2+N*2];
    int cur[N],head[N],cnt;
    long long extra[N];
    vector<long long>flow;
    int tot;
    Bounded_Feasible_Flow_With_Source_Sink():cnt(1),tot(0)
    {
        memset(head,0,sizeof(head));
        memset(extra,0,sizeof(extra));
    }
    void add_edge(int u,int v,long long f)
    {
        cnt++;
        edge[cnt].to=v;
        edge[cnt].flow=f;
        edge[cnt].next=head[u];
        head[u]=cnt;
        return;
    }
    void add(int u,int v,long long f)
    {
        add_edge(u,v,f);
        add_edge(v,u,0);
        return;
    }
    void add(int u,int v,long long lower,long long upper)
    {
        add(u,v,upper-lower);
        extra[v]+=lower,extra[u]-=lower;
        flow.emplace_back(lower);
        return;
    }
    int dep[N];
    bool bfs(int s,int t)
    {
        for(int i=1;i<=tot;i++)
            dep[i]=-1;
        queue<int>q;
        q.push(s);
        dep[s]=0;
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            for(int i=head[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(dep[v]!=-1||edge[i].flow<=0) continue;
                dep[v]=dep[u]+1;
                q.push(v);
            }
        }
        return dep[t]!=-1;
    }
    long long dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return flow;
        long long used=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(dep[v]!=dep[u]+1||edge[i].flow<=0) continue;
            long long now=dfs(v,t,min(flow,(long long)edge[i].flow));
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        return used;
    }
    long long dinic(int s,int t)
    {
        long long res=0;
        while(bfs(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            res+=dfs(s,t,INF);
        }
        return res;
    }
    vector<long long> solve(int S,int T)
    {
        int s=++tot,t=++tot;
        long long sum=0;
        for(int i=1;i<=tot-2;i++)
            if(extra[i]>0)
            {
                sum+=extra[i];
                add(s,i,extra[i]);
            }
            else if(extra[i]<0)
            {
                add(i,t,-extra[i]);
            }
        add(T,S,INF);
```

```
            if(dinic(s,t)!=sum) return {};
            for(int i=0;i<(int)flow.size();i++)
                flow[i]+=edge[i*2+3].flow;
            return flow;
        }
    }bounded_feasible_flow_with_source_sink;
    struct Bounded_Max_Flow_With_Source_Sink
    {
        static const int N=205,M=10005;
        static const long long INF=4557430888798830399;
        struct Edge
        {
            int to,next;
            long long flow;
        }edge[M*2+N*2];
        int cur[N],head[N],cnt;
        long long extra[N];
        int tot;
        Bounded_Max_Flow_With_Source_Sink():cnt(1),tot(0)
        {
            memset(head,0,sizeof(head));
            memset(extra,0,sizeof(extra));
        }
        void add_edge(int u,int v,long long f)
        {
            cnt++;
            edge[cnt].to=v;
            edge[cnt].flow=f;
            edge[cnt].next=head[u];
            head[u]=cnt;
            return;
        }
        void add(int u,int v,long long f)
        {
            add_edge(u,v,f);
            add_edge(v,u,0);
            return;
        }
        void add(int u,int v,long long lower,long long upper)
        {
            add(u,v,upper-lower);
            extra[v]+=lower,extra[u]-=lower;
            return;
        }
        int dep[N];
        bool bfs(int s,int t)
        {
            for(int i=1;i<=tot;i++)
                dep[i]=-1;
            queue<int>q;
            q.push(s);
            dep[s]=0;
            while(!q.empty())
            {
```

```
            int u=q.front();
            q.pop();
            for(int i=head[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(dep[v]!=-1||edge[i].flow<=0) continue;
                dep[v]=dep[u]+1;
                q.push(v);
            }
        }
        return dep[t]!=-1;
    }
    long long dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return flow;
        long long used=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(dep[v]!=dep[u]+1||edge[i].flow<=0) continue;
            long long now=dfs(v,t,min(flow,(long long)edge[i].flow));
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        return used;
    }
    long long dinic(int s,int t)
    {
        long long res=0;
        while(bfs(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            res+=dfs(s,t,INF);
        }
        return res;
    }
    long long solve(int S,int T)
    {
        int s=++tot,t=++tot;
        long long sum=0;
        for(int i=1;i<=tot-2;i++)
            if(extra[i]>0)
            {
                sum+=extra[i];
                add(s,i,extra[i]);
            }
            else if(extra[i]<0)
            {
                add(i,t,-extra[i]);
            }
        add(T,S,INF);
        if(dinic(s,t)!=sum) return -1;
        return dinic(S,T);
    }
}bounded_max_flow_with_source_sink;
struct Bounded_Min_Flow_With_Source_Sink
{
    static const int N=50010,M=125010;
    static const long long INF=4557430888798830399;
    struct Edge
    {
        int to,next;
        long long flow;
    }edge[M*2+N*2];
    int cur[N],head[N],cnt;
    long long extra[N];
    int tot;
    Bounded_Min_Flow_With_Source_Sink():cnt(1),tot(0)
    {
        memset(head,0,sizeof(head));
        memset(extra,0,sizeof(extra));
    }
    void add_edge(int u,int v,long long f)
    {
        cnt++;
        edge[cnt].to=v;
        edge[cnt].flow=f;
        edge[cnt].next=head[u];
        head[u]=cnt;
        return;
    }
    void add(int u,int v,long long f)
    {
        add_edge(u,v,f);
        add_edge(v,u,0);
        return;
    }
    void add(int u,int v,long long lower,long long upper)
    {
        add(u,v,upper-lower);
        extra[v]+=lower,extra[u]-=lower;
        return;
    }
    int dep[N];
    bool bfs(int s,int t)
    {
        for(int i=1;i<=tot;i++)
            dep[i]=-1;
        queue<int>q;
        q.push(s);
        dep[s]=0;
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            for(int i=head[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(dep[v]!=-1||edge[i].flow<=0) continue;
                dep[v]=dep[u]+1;
                q.push(v);
            }
        }
        return dep[t]!=-1;
    }
    long long dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return flow;
        long long used=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(dep[v]!=dep[u]+1||edge[i].flow<=0) continue;
            long long now=dfs(v,t,min(flow,(long long)edge[i].flow));
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        return used;
    }
    long long dinic(int s,int t)
    {
        long long res=0;
        while(bfs(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            res+=dfs(s,t,INF);
        }
        return res;
    }
    long long solve(int S,int T)
    {
        int s=++tot,t=++tot;
        long long sum=0;
        for(int i=1;i<=tot-2;i++)
            if(extra[i]>0)
            {
                sum+=extra[i];
                add(s,i,extra[i]);
            }
            else if(extra[i]<0)
            {
                add(i,t,-extra[i]);
            }
        add(T,S,INF);
        if(dinic(s,t)!=sum) return -1;
        long long res=edge[cnt].flow;
        edge[cnt].flow=edge[cnt^1].flow=0;
        return res-dinic(T,S);
    }
}bounded_min_flow_with_source_sink;
struct Bounded_Min_Cost_Feasible_Flow_With_Source_Sink
{
    static const int N=305,M=5305;
    static const long long INF=4557430888798830399;
    struct Edge
    {
        int to,next;
        int cost;
        long long flow;
    }edge[M*2+N*2];
    int cur[N],head[N],cnt;
    long long extra[N];
    int tot;
    long long totalcost;
    Bounded_Min_Cost_Feasible_Flow_With_Source_Sink():cnt(1),tot(0),totalcost(0)
    {
        memset(head,0,sizeof(head));
        memset(extra,0,sizeof(extra));
    }
    void add_edge(int u,int v,int c,long long f)
    {
        cnt++;
        edge[cnt].to=v;
        edge[cnt].cost=c;
        edge[cnt].flow=f;
        edge[cnt].next=head[u];
        head[u]=cnt;
        return;
    }
    void add(int u,int v,int c,long long f)
    {
        add_edge(u,v,c,f);
        add_edge(v,u,-c,0);
        return;
    }
    void add(int u,int v,int c,long long lower,long long upper)
    {
        totalcost+=lower*c;
        add(u,v,c,upper-lower);
        extra[v]+=lower,extra[u]-=lower;
        return;
    }
    long long dis[N];
    bool vis[N];
    bool spfa(int s,int t)
    {
        for(int i=1;i<=tot;i++)
            vis[i]=false;
        for(int i=1;i<=tot;i++)
            dis[i]=INF;
        queue<int>q;
        vis[s]=true;
        dis[s]=0;
        q.push(s);
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            vis[u]=false;
            for(int i=head[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(edge[i].flow<=0) continue;
                if(dis[v]>dis[u]+edge[i].cost)
                {
                    dis[v]=dis[u]+edge[i].cost;
                    if(!vis[v])
                    {
                        vis[v]=true;
                        q.push(v);
                    }
                }
            }
        }
        return dis[t]!=INF;
    }
    bool book[N];
    pair<long long,long long> dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return make_pair(flow,0);
        book[u]=true;
        long long used=0,res=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(book[v]||dis[v]!=dis[u]+edge[i].cost||edge[i].flow<=0) continue;
            pair<long long,long long>val=dfs(v,t,min(flow,edge[i].flow));
            long long now=val.first;
            res+=val.second+now*edge[i].cost;
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        book[u]=false;
        return make_pair(used,res);
    }
    pair<long long,long long> ssp(int s,int t)
    {
        long long ans=0,cost=0;
        for(int i=1;i<=tot;i++)
```

```
                book[i]=false;
            while(spfa(s,t))
            {
                for(int i=1;i<=tot;i++)
                    cur[i]=head[i];
                pair<long long,long long> res=dfs(s,t,INF);
                ans+=res.first,cost+=res.second;
            }
            return make_pair(ans,cost);
        }
        pair<long long,long long> solve(int S,int T)
        {
            int s=++tot,t=++tot;
            long long sum=0;
            for(int i=1;i<=tot-2;i++)
                if(extra[i]>0)
                {
                    sum+=extra[i];
                    add(s,i,0,extra[i]);
                }
                else if(extra[i]<0)
                {
                    add(i,t,0,-extra[i]);
                }
            add(T,S,0,INF);
            pair<long long,long long>res=ssp(s,t);
            if(res.first!=sum) return make_pair(-1,-1);
            totalcost+=res.second;
            return make_pair(res.first,totalcost);
        }
    }bounded_min_cost_feasible_flow_with_source_sink;
    const long long INF=Bounded_Min_Cost_Feasible_Flow_With_Source_Sink::INF;
    int main()
    {
        int n;
        scanf("%d",&n);
        bounded_min_cost_feasible_flow_with_source_sink.tot=n+1;
        int t=n+1;
        for(int i=1;i<=n;i++)
        {
            int k;
            scanf("%d",&k);
            for(int j=1;j<=k;j++)
            {
                int b,v;
                scanf("%d%d",&b,&v);
                bounded_min_cost_feasible_flow_with_source_sink.add(i,b,v,1,INF);
            }
            bounded_min_cost_feasible_flow_with_source_sink.add(i,t,0,0,INF);
        }
        long long
    ans=bounded_min_cost_feasible_flow_with_source_sink.solve(1,t).second;
        printf("%lld",ans);
        return 0;
    }
```

O(n)-O(1) LCA

```
#include<iostream>
#include<cstdio>
#include<vector>
using namespace std;
const int N=500005;
int n,q,s;
vector<int>G[N];
int dfn[N],Index;
int mn[N][20],lg2[N];
void dfs(int u,int father)
{
    dfn[u]=++Index;
    mn[Index][0]=father;
    for(int v:G[u])
    {
        if(v==father) continue;
        dfs(v,u);
    }
    return;
}
int lca(int u,int v)
{
    if(u==v) return u;
    u=dfn[u],v=dfn[v];
    if(u>v) swap(u,v);
    u++;
    int d=lg2[v-u+1];
    if(dfn[mn[u][d]]<dfn[mn[v-(1<<d)+1][d]]) return mn[u][d];
    else return mn[v-(1<<d)+1][d];
}
void init_lca(int n)
{
    lg2[0]=-1;
    for(int i=1;i<=n;i++)
        lg2[i]=lg2[i/2]+1;
    for(int j=1;(1<<j)<=n;j++)
        for(int i=1;i+(1<<j)-1<=n;i++)
            if(dfn[mn[i][j-1]]<dfn[mn[i+(1<<(j-1))][j-1]]) mn[i][j]=mn[i][j-1];
            else mn[i][j]=mn[i+(1<<(j-1))][j-1];
    return;
}
int main()
{
    scanf("%d%d%d",&n,&q,&s);
    for(int i=1;i<n;i++)
    {
        int x,y;
        scanf("%d%d",&x,&y);
        G[x].emplace_back(y);
        G[y].emplace_back(x);
    }
    dfs(s,0);
    init_lca(n);
    while(q--)
    {
        int u,v;
        scanf("%d%d",&u,&v);
        printf("%d\n",lca(u,v));
    }
    return 0;
}
```

虚树

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<algorithm>
using namespace std;
const int N=500005;
int n,q;
vector<int>G[N];
int dfn[N],Index;
int mn[N][20],lg2[N];
void dfs(int u,int father)
{
    dfn[u]=++Index;
    mn[Index][0]=father;
    for(int v:G[u])
    {
        if(v==father) continue;
        dfs(v,u);
    }
    return;
}
int lca(int u,int v)
{
    if(u==v) return u;
    u=dfn[u],v=dfn[v];
    if(u>v) swap(u,v);
    u++;
    int d=lg2[v-u+1];
    if(dfn[mn[u][d]]<dfn[mn[v-(1<<d)+1][d]]) return mn[u][d];
    else return mn[v-(1<<d)+1][d];
}
```

```
vector<int>E[N];
void build(vector<int>h)
{
    sort(h.begin(),h.end(),[=](const int &x,const int &y){return dfn[x]<dfn[y];});
    static int s[N],top;
    top=0;
    s[++top]=1;
    E[1].clear();
    for(int i=0;i<(int)h.size();i++)
        if(h[i]!=1)
        {
            int l=lca(h[i],s[top]);
            if(l!=s[top])
            {
                while(dfn[l]<dfn[s[top-1]])
                {
                    E[s[top]].emplace_back(s[top-1]),E[s[top-
1]].emplace_back(s[top]);
                    top--;
                }
                if(dfn[l]>dfn[s[top-1]])
                {
                    E[l].clear();
                    E[l].emplace_back(s[top]),E[s[top]].emplace_back(l);
                    top--;
                    s[++top]=l;
                }
                else
                {
                    E[s[top-1]].emplace_back(s[top]),E[s[top]].emplace_back(s[top-
1]);
                    top--;
                }
            }
            E[h[i]].clear();
            s[++top]=h[i];
        }
    for(int i=1;i<top;i++)
        E[s[i]].emplace_back(s[i+1]),E[s[i+1]].emplace_back(s[i]);
    return;
}
int main()
{
    scanf("%d%d",&n,&q);
    for(int i=1;i<n;i++)
    {
        int x,y;
        scanf("%d%d",&x,&y);
        G[x].emplace_back(y);
        G[y].emplace_back(x);
    }
    dfs(1,0);
    lg2[0]=-1;
    for(int i=1;i<=n;i++)
```

```
        lg2[i]=lg2[i/2]+1;
    for(int j=1;(1<<j)<=n;j++)
        for(int i=1;i+(1<<j)-1<=n;i++)
            if(dfn[mn[i][j-1]]<dfn[mn[i+(1<<(j-1))][j-1]]) mn[i][j]=mn[i][j-1];
            else mn[i][j]=mn[i+(1<<(j-1))][j-1];
    return 0;
}
```