Team notebook

November 15, 2019

Contents

1 Graph-Aero

```
/*
Given an undirected graph with N vertices's and M edges. You are to
    process K queries of adding new edges into the graph. After each
    query you should output the only number - amount of bridges in graph.
    Graph may contain loops and multiple edges.
*/
```

```
#include<bits/stdc++.h>
using namespace std;

const int N = 1e5+7;
int bridges = 0;

struct DSU {
    int par[N], sz[N];

    DSU() {
        for (int i=0; i<N; i++) {
            par[i] = i;
            sz[i] = 1;
        }
    }

    int find(int u) {
        if (par[u] == u)  return u;
        return par[u] = find(par[u]);
    }

    void merge(int u, int v) {
        u = find(u);
        v = find(v);
        if (u==v)  return;
        par[v] = u;
        sz[u] += sz[v];
    }
};

DSU rep, root;
int par[N];
```

```
list<int>::iterator pos[N];
list<int> adj[N];

void merge(int u) {
    assert(par[u]);
    adj[u].erase(adj[u].begin());
    adj[par[u]].erase(pos[u]);

    for (int v: adj[u]) {
        par[v] = par[u];
        *(adj[v].begin()) = par[u];
    }
    adj[par[u]].splice(adj[par[u]].end(), adj[u]);
    par[u] = 0;
}

void dfs(int u, int p) {
    for (auto it = adj[u].begin(); it != adj[u].end();) {
        if (*it == p)  it = adj[u].erase(it);
        else {
            int v = *it;
            pos[v] = it;
            dfs(v, u);
            it++;
        }
    }
    par[u] = p;
    if (p) {
        adj[u].insert(adj[u].begin(), p);
    }
}

bool takenu[N], takenv[N];

void AddEdge(int u, int v) {
    u = rep.find(u);
    v = rep.find(v);
    if (u == v)    return;

    int ru = root.find(u);
    int rv = root.find(v);

    if (ru == rv) {
        vector<int> au, av;
        int cu = u, cv = v, lca = -1;

        while (true) {
            if (cu) {
                au.push_back(cu);
                takenu[cu] = 1;
            }

            if (cv) {
                av.push_back(cv);
                takenv[cv] = 1;
            }

            if (cu && takenu[cu] && takenv[cu]) {
                lca = cu;
                break;
            }
            if (cv && takenu[cv] && takenv[cv]) {
                lca = cv;
                break;
            }
            cu = par[cu];
            cv = par[cv];
        }

        assert(lca > 0);

        for (int x: au) {
            if (x == lca)  break;
            bridges--;
            merge(x);
            rep.merge(lca, x);
        }

        for (int x: av) {
            if (x == lca)  break;
            bridges--;
            merge(x);
            rep.merge(lca, x);
        }

        for (int x: au)   takenu[x] = false;
        for (int x: av)   takenv[x] = false;

    }
    else {
        if (root.sz[ru] < root.sz[rv]) {
            swap(ru, rv);
            swap(u, v);
        }

        dfs(v, 0);
        assert(par[v] == 0);

        pos[v] = adj[u].insert(adj[u].end(), v);
        par[v] = u;
        adj[v].insert(adj[v].begin(), u);
        root.merge(u, v);
        bridges++;
    }
//
//    for (int i=1; i<=4; i++) {
//        cout<<i<<"----> par: "<<par[i]<<", rep: "<<rep.find(i)<<", root:
    "<<root.find(i);
//        if (par[i]) cout<<", pos"<<": "<<distance(adj[par[i]].begin(),
    pos[i]);
//        cout<<endl;
//
//        for (auto it = adj[i].begin(); it != adj[i].end(); it++) {
//            cout << "::: "<< distance(adj[i].begin(), it) << ": " <<
    (*it) << endl;
//        }
//    }
}

int main() {
    freopen("bridges.in", "r", stdin);
    freopen("bridges.out", "w", stdout);

    int n, m;
    cin>>n>>m;

    for (int i=1; i<=m; i++) {
        int u, v;
        cin>>u>>v;
        AddEdge(u, v);
    }

    int q;
    cin>>q;
    for (int i=1; i<=q; i++) {
        int u, v;
        cin>>u>>v;
        AddEdge(u, v);
        cout<<bridges<<"\n";
    }
}
```

2 Stoer-Wagner

```
///////////////////////////////////////////////////////////////////////
// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting path
// algorithm for finding min cost perfect matchings in dense
// graphs. In practice, it solves 1000x1000 problems in around 1
// second.
//
//   cost[i][j] = cost for pairing left node i with right node j
//   Lmate[i] = index of right node that left node i pairs with
//   Rmate[j] = index of left node that right node j pairs with
//
// The values in cost[i][j] may be positive or negative. To perform
// maximization, simply negate the cost[][] matrix.
///////////////////////////////////////////////////////////////////////

#include <algorithm>
#include <cstdio>
#include <cmath>
#include <vector>

using namespace std;

typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
  int n = int(cost.size());

  // construct dual feasible solution
  VD u(n);
  VD v(n);
```

```
for (int i = 0; i < n; i++) {
  u[i] = cost[i][0];
  for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
}
for (int j = 0; j < n; j++) {
  v[j] = cost[0][j] - u[0];
  for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
}

// construct primal solution satisfying complementary slackness
Lmate = VI(n, -1);
Rmate = VI(n, -1);
int mated = 0;
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    if (Rmate[j] != -1) continue;
    if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
      Lmate[i] = j;
      Rmate[j] = i;
      mated++;
      break;
    }
  }
}

VD dist(n);
VI dad(n);
VI seen(n);

// repeat until primal solution is feasible
while (mated < n) {

  // find an unmatched left node
  int s = 0;
  while (Lmate[s] != -1) s++;

  // initialize Dijkstra
  fill(dad.begin(), dad.end(), -1);
  fill(seen.begin(), seen.end(), 0);
  for (int k = 0; k < n; k++)
    dist[k] = cost[s][k] - u[s] - v[k];

  int j = 0;
  while (true) {

    // find closest
    j = -1;
    for (int k = 0; k < n; k++) {
      if (seen[k]) continue;
      if (j == -1 || dist[k] < dist[j]) j = k;
    }
    seen[j] = 1;

    // termination condition
    if (Rmate[j] == -1) break;

    // relax neighbors
    const int i = Rmate[j];
    for (int k = 0; k < n; k++) {
      if (seen[k]) continue;
      const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
      if (dist[k] > new_dist) {
        dist[k] = new_dist;
        dad[k] = j;
      }
    }
  }

  // update dual variables
  for (int k = 0; k < n; k++) {
    if (k == j || !seen[k]) continue;
    const int i = Rmate[k];
    v[k] += dist[k] - dist[j];
    u[i] -= dist[k] - dist[j];
  }
  u[s] += dist[j];

  // augment along path
  while (dad[j] >= 0) {
    const int d = dad[j];
    Rmate[j] = Rmate[d];
    Lmate[Rmate[j]] = j;
    j = d;
  }
  Rmate[j] = s;
  Lmate[s] = j;

  mated++;
}
```

```
  double value = 0;
  for (int i = 0; i < n; i++)
    value += cost[i][Lmate[i]];

  return value;
}
```

3 all$_{t}riangle_{w}ith_{n}o_{p}oint_{i}nside$

```
#include <bits/stdc++.h>
using namespace std ;

struct Point {
    int x, y ;
    Point() {}
    Point(int x_,int y_) {x=x_,y=y_;}
    Point operator + (const Point &p) {
        return Point(x+p.x,y+p.y);
    }
    Point operator - (const Point &p) {
        return Point(x-p.x,y-p.y);
    }
    Point operator * (int t) {
        return Point(x*t,y*t);
    }
    int operator * (const Point &p) {
        return x*p.x + y*p.y;
    }
    int operator ^ (const Point &p) {
        return x*p.y - y*p.x;
    }
    bool operator < (const Point &p) const {
        return make_pair(y,x) < make_pair(p.y,p.x);
    }
};

bool comp(Point p, Point q) {
    return ( (p^q) > 0 );
}


int main() {
    freopen("farmer.in","r",stdin);
    freopen("farmer.out","w",stdout);
    int n ; cin >> n ;
    vector <Point> P(n) ;
    for(int i = 0 ; i < n ; i++) {
        cin >> P[i].x >> P[i].y;
    }
    sort(P.begin(),P.end());

    int ans = 0;
    for(int i = 0 ; i < n ; i++) {
        Point p = P[i];
        vector<Point> upper;
        for(int j = i+1 ; j < n ; j++) {
            upper.push_back(P[j]-p) ;
        }
        sort(upper.begin(),upper.end(),comp);
        for(int j = 0 ; j + 1 < upper.size() ; j++) {
            Point p = upper[j];
            Point q = upper[j+1];
            for(int k = j + 1 ; k < upper.size() ; k++) {
                Point r = upper[k];
                if ( ((q-p)^(r-p)) >= 0 ) {
                    q = r;
                    ans++;
                }
            }
        }
    }
    printf ("%d\n" , ans);
}
```

4 circle$_{c}ut_{l}ine$

```
#include <bits/stdc++.h>
using namespace std ;

#define double long double

const double eps = 1e-9 ;

int dcmp(double x) {
    if (fabs(x) < eps) return 0;
    if (x < 0.0) return -1;
    return 1;
}

struct Point {
    double x , y ;
    Point() {}
    Point(double x_,double y_) {x=x_,y=y_;}
    Point operator + (const Point &p) {
        return {x+p.x,y+p.y} ;
    }
    Point operator - (const Point &p) {
        return {x-p.x,y-p.y} ;
    }
    Point operator * (const double &t) {
        return {x*t,y*t} ;
    }
    double operator * (const Point &p) {
        return x*p.x + y*p.y ;
    }
    double operator ^ (const Point &p) {
        return x*p.y - y*p.x ;
    }
};


struct Line {
    double a , b , c ; // ax + by + c = 0
    Point p , d ; // p + dt
    Line() {}
    Line(double a_,double b_,double c_) {
        a = a_ , b = b_ , c = c_ ;
        d = Point(-b,a) ;
        if (dcmp(a) == 0) {
            p = Point(0,-c/b) ;
        }
        else {
            p = Point(-c/a,0) ;
        }
    }
    Line(Point p_,Point q_) {
        p = p_ , d = q_ - p_;
    }
    double val(Point p) {
        return a*p.x + b*p.y + c ;
    }
}line[4];

int n ;
double R ;

const int N = 50005 ;
const double pi = acos(-1.0) ;


void intersect(vector<Point> &V,Point a,Point b,Line l) {
    Point p = l.p , q = l.p + l.d ;
    double na = (a-p)^(q-p) , nb = (b-p)^(q-p) ;
   // double na = l.val(a) , nb = l.val(b) ;
//    if (dcmp(na*nb) >= 0) return ;
    if (na*nb < 0.0) {
        V.push_back(a + (b-a)*(na/(na-nb))) ;
    }
}

void cut(vector<Point> &polygon, Line l, int sign) {
    vector<Point> np ;
    int sz = polygon.size();
    for(int i = 0 ; i < sz ; i++) {
        Point p = polygon[i] , q = polygon[(i+1)%sz];
//        cout << "val: " << l.val(p) << endl ;
        if (dcmp(l.val(p))*sign >= 0) {
            np.push_back(p);
        }
        intersect(np,p,q,l);
    }
    polygon = np ;
}

vector <Point> Circle ;

double f(int mask) {
    vector <Point> circle = Circle ;
    for(int i = 0 ; i < n ; i++) {
        if ( (mask>>i)&1 ) {
            cut(circle,line[i],+1);
        }
        else {
            cut(circle,line[i],-1);
        }
    }
    if (circle.size() == 0.0) return 0.0 ;
    int sz = circle.size() ;
    double area = 0 ;
    for(int i = 0 ; i < sz ; i++) {
        area += (circle[i]^circle[(i+1)%sz]) ;
    }
    return area/2.0 ;
}


void init() {
    for(int i = 0 ; i < N ; i++) {
        double angle = 2.0*pi/N*i ;
        Circle.push_back(Point(R*cos(angle),R*sin(angle))) ;
    }
}


bool used[20] ;
vector < double > v ;
map < pair<double,double> , pair<double,double> > Map ;

void solve(double &t1,double &t2) {
    if (Map.find({t1,t2}) != Map.end()) {
        pair<double,double> t = Map[{t1,t2}] ;
        t1 = t.first , t2 = t.second ;
        return ;
    }
    int who = 2 ;
    if (t1 < t2 + eps) {
        who = 1 ;
    }
    double best1 = t1 , best2 = t2 ;
    if (who == 1) {
        for(int i = 0 ; i < v.size() ; i++) {
            if (!used[i]) {
                double cur = v[i] ;
                used[i] = true ;
                double tt1 = t1 + cur , tt2 = t2 ;
                solve(tt1,tt2) ;
                if (best1 < tt1) {
                    best1 = tt1 ;
                    best2 = tt2 ;
                }
                used[i] = false ;
            }
        }
    }
    else {
        for(int i = 0 ; i < v.size() ; i++) {
            if (!used[i]) {
                double cur = v[i] ;
                used[i] = true ;
                double tt1 = t1 , tt2 = t2 + cur ;
                solve(tt1,tt2) ;
                if (best2 < tt2) {
                    best1 = tt1 ;
                    best2 = tt2 ;
                }
                used[i] = false ;
            }
        }
    }
    Map[{t1,t2}] = {best1,best2} ;
    t1 = best1 , t2 = best2 ;
}


int main() {

    freopen("vs.in" , "r" , stdin) ;
    freopen("vs.out" , "w" , stdout) ;



    cout << setprecision(12) << fixed ;
    cin >> n >> R ;
    init() ;
    for(int i = 0 ; i < n ; i++) {
        double a,b,c ;
        cin >> a >> b >> c ;
        line[i] = {a,b,c} ;
    }
    for(int i = 0 ; i < (1<<n) ; i++) {
        double slice = f(i) ;
        if (slice > eps) {
            v.push_back(slice) ;
//          cout << "slice: " << slice << endl ;
```

```
        }
    }
    double best1 = 0 , best2 = 0 ;
    solve(best1,best2) ;
    cout << best1 << " " << best2 << endl ;

}
```

5 expected number of points in 3d convex hull

```
#include <bits/stdc++.h>
using namespace std ;

struct Point {
    int x , y , z;
    Point() {}
    Point(int x_,int y_,int z_) {x=x_,y=y_,z=z_;}
    Point operator + (const Point &p) {
        return {x+p.x,y+p.y,z+p.z} ;
    }
    Point operator - (const Point &p) {
        return {x-p.x,y-p.y,z-p.z} ;
    }
    Point operator * (const int &t) {
        return {x*t,y*t,z*t} ;
    }
    int operator * (const Point &p) {
        return x*p.x + y*p.y + z*p.z;
    }
    Point operator ^ (const Point &p) {
        return Point(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y - y*p.x);
    }
};

int main() {
    //freopen ("in.txt","r",stdin);
    int n; cin >> n;
    vector <Point> P(n);
    vector <double> prob(n);
    double cc = 1.0;
    for(int i = 0 ; i < n ; i++) {
        cin >> prob[i] ;
        cc *= (1.0-prob[i]);
        cin >> P[i].x >> P[i].y >> P[i].z ;
    }
    double f = 0;
    double ans = 0;
    double chance = 1.0;
    for(int i = 0; i < n; i++) {
        for(int j = i+1 ; j < n; j++) {
            for(int k = j+1; k < n; k++) {
                Point norm = (P[j]-P[i])^(P[k]-P[i]);
                double probUp = 1.0 , probDown = 1.0;
                double just = prob[i]*prob[j]*prob[k];
                for(int l = 0; l < n; l++) {
                    if (l == i or l == j or l == k) continue;
                    just *= (1.0-prob[l]);
                    if (norm*(P[l]-P[i]) > 0) {
                        probUp *= (1.0-prob[l]);
                    }
                    else {
                        probDown *= (1.0-prob[l]);
                    }
                }
                double cur = prob[i]*prob[j]*prob[k]*(probUp + probDown -
                    2.0*probUp*probDown);
                f += cur;
                ans += 3.0*just;
                chance -= just;
            }
        }
    }
    for(int i = 0; i < n; i++) {
        for(int j = i+1; j < n; j++) {
            double just = prob[i]*prob[j];
            for(int k = 0; k < n; k++) {
                if (k == i or k == j) continue;
                just *= (1.0-prob[k]);
            }
            ans += 2.0*just;
            chance -= just;
        }
    }
    for(int i = 0; i < n; i++) {
        double just = prob[i];
        for(int j = 0; j < n; j++) {
            if (j == i) continue;
            just *= (1.0-prob[j]);
        }
        ans += just;
        chance -= just;
    }
    chance -= cc;

    double e = 1.5*f ;
    ans += e - f + 2.0*chance ;
    cout << setprecision(12) << ans << endl;
}
```

6 find-intersecting-pair

```
#include <algorithm>
#include <vector>
#include <set>
using namespace std;

typedef pair<int, int> pii;

int cross(int ax, int ay, int bx, int by, int cx, int cy) {
    return (bx - ax) * (cy - ay) - (by - ay) * (cx - ax);
}

int cross(pii a, pii b, pii c) {
    return cross(a.first, a.second, b.first, b.second, c.first, c.second);
}

class segment {
    public:
    pii a, b;
    int id;
    segment(pii a, pii b, int id) :
        a(a), b(b), id(id) {
    }
    bool operator<(const segment &o) const {
        if (a.first < o.a.first) {
            int s = cross(a, b, o.a);
            return (s > 0 || s == 0 && a.second < o.a.second);
        } else if (a.first > o.a.first) {
            int s = cross(o.a, o.b, a);
            return (s < 0 || s == 0 && a.second < o.a.second);
        }
        return a.second < o.a.second;
    }
};

class event {
    public:
    pii p;
    int id;
    int type;
    event(pii p, int id, int type) :
        p(p), id(id), type(type) {
    }
    bool operator<(const event &o) const {
        return p.first < o.p.first || p.first == o.p.first && (type >
            o.type || type == o.type && p.second < o.p.second);
    }
};

bool intersect(segment s1, segment s2) {
    int x1 = s1.a.first, y1 = s1.a.second, x2 = s1.b.first, y2 =
        s1.b.second;
    int x3 = s2.a.first, y3 = s2.a.second, x4 = s2.b.first, y4 =
        s2.b.second;
    if (max(x1, x2) < min(x3, x4) || max(x3, x4) < min(x1, x2) || max(y1,
        y2) < min(y3, y4) || max(y3, y4) < min(y1, y2)) {
        return false;
    }
    int z1 = (x3 - x1) * (y2 - y1) - (y3 - y1) * (x2 - x1);
    int z2 = (x4 - x1) * (y2 - y1) - (y4 - y1) * (x2 - x1);
    if (z1 < 0 && z2 < 0 || z1 > 0 && z2 > 0) {
        return false;
    }
    int z3 = (x1 - x3) * (y4 - y3) - (y1 - y3) * (x4 - x3);
    int z4 = (x2 - x3) * (y4 - y3) - (y2 - y3) * (x4 - x3);
    if (z3 < 0 && z4 < 0 || z3 > 0 && z4 > 0) {
        return false;
    }
    return true;
}

pii findIntersection(vector<segment> s) {
    int n = s.size();
    vector<event> e;
    for (int i = 0; i < n; ++i) {
        if (s[i].a > s[i].b)
            swap(s[i].a, s[i].b);
        e.push_back(event(s[i].a, i, 1));
        e.push_back(event(s[i].b, i, -1));
    }
    sort(e.begin(), e.end());

    set<segment> q;

    for (int i = 0; i < n * 2; ++i) {
        int id = e[i].id;
        if (e[i].type == 1) {
            set<segment>::iterator it = q.lower_bound(s[id]);
            if (it != q.end() && intersect(*it, s[id]))
                return make_pair(it->id, s[id].id);
            if (it != q.begin() && intersect(*--it, s[id]))
                return make_pair(it->id, s[id].id);
            q.insert(s[id]);
        } else {
            set<segment>::iterator it = q.lower_bound(s[id]), next = it,
                 prev = it;
            if (it != q.begin() && it != --q.end()) {
                ++next, --prev;
                if (intersect(*next, *prev))
                    return make_pair(next->id, prev->id);
            }
            q.erase(it);
        }
    }
    return make_pair(-1, -1);
}

int main() {
}
```

7 gomory-hu-rafid

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

#define INF 2000000000
const int MAX_E=60003;
const int MAX_V=5003;
int
    ver[MAX_E],cap[MAX_E],nx[MAX_E],last[MAX_V],ds[MAX_V],st[MAX_V],now[MAX_V],e

inline void reset()
{
        memset(nx,-1,sizeof(nx));
        memset(last,-1,sizeof(last));
        edge_count=0;
}
inline void addedge(const int v,const int w,const int capacity,const int
    reverse_capacity)
{
        ver[edge_count]=w; cap[edge_count]=capacity;
            nx[edge_count]=last[v]; last[v]=edge_count++;
        ver[edge_count]=v; cap[edge_count]=reverse_capacity;
            nx[edge_count]=last[w]; last[w]=edge_count++;
}
inline bool bfs()
{
        memset(ds,-1,sizeof(ds));
        int a,b;
        a=b=0;
        st[0]=T;
        ds[T]=0;
        while (a<=b)
        {
                int v=st[a++];
                for (int w=last[v];w>=0;w=nx[w])
                {
                        if (cap[w^1]>0 && ds[ver[w]]==-1)
                        {
                                st[++b]=ver[w];
                                ds[ver[w]]=ds[v]+1;
                        }
                }
        }
        return ds[S]>=0;
}
int dfs(int v,int cur)
{
        if (v==T) return cur;
        for (int &w=now[v];w>=0;w=nx[w])
        {
                if (cap[w]>0 && ds[ver[w]]==ds[v]-1)
                {
                        int d=dfs(ver[w],min(cur,cap[w]));
                        if (d)
                        {
                                cap[w]-=d;
                                cap[w^1]+=d;
                                return d;
                        }
                }
        }
        return 0;
}
inline long long flow()
{
        long long res=0;
        while (bfs())
        {
                for (int i=0;i<MAX_V;i++) now[i]=last[i];
                while (1)
                {
                        int tf=dfs(S,INF);
                        res+=tf;
                        if (!tf) break;
                }
        }
        return res;
}

typedef struct { int u, v, w; } edge;

// returns edges of gomory hu tree
// nodes are labeled 1..n here and also in dinic
vector<edge> gomory_hu(int n, vector<edge> &e) {
        vector<edge> edg;
        vector<int> par(n+1, 1);

        for(int i=2; i<=n; ++i) {
                reset();
                for(auto &qq : e) addedge(qq.u, qq.v, qq.w, qq.w);
                S = par[i], T = i;
                edg.push_back( { par[i], i, (int) flow() } );
```

```
                for(int j=i+1; j<=n; ++j) {
                        if(ds[j] >= 0 and par[j] == par[i]) {
                                par[j] = i;
                        }
                }
        }
        return edg;
}

// ------------------ BEGIN CUT ------------------
// The following solves CF 343E. Pumping Stations

const int N = 207;
int par[N];
vector<int> seq[N];   // node sequences

inline int Find(int r) {
        return par[r] == r ? r : par[r] = Find(par[r]);
}

int main() {
        ios::sync_with_stdio(false);
        cin.tie(0); cout.tie(0);

        int n, m;
        cin >> n >> m;

        vector< edge > e;
        while(m--) {
                int u, v, c;
                cin >> u >> v >> c;
                e.push_back( { u, v, c } );
        }

        auto edg = gomory_hu(n, e);
        // for(auto &qq : edg) {
        //      cerr << qq.u << " - " << qq.v << ": " << qq.w << "\n";
        // }
        sort(edg.begin(), edg.end(), [](edge &p, edge &q) { return p.w >
            q.w; });
        for(int i=1; i<=n; ++i) {
                par[i] = i;
                seq[i].push_back(i);
        }
```

```
	int res = 0;
	for(auto &qq : edg) {
		int pu = Find(qq.u), pv = Find(qq.v);
		if(pu != pv) {
			par[pv] = pu;
			seq[pu].insert(seq[pu].end(), seq[pv].begin(),
				seq[pv].end());
		}
		res += qq.w;
	}

	cout << res << "\n";
	int root = Find(1);
	for(auto &qq : seq[root]) {
		cout << qq << " ";
	} cout << "\n";

	return 0;
}
```

8 lin-rec

```
//
// Linear Recurrence Solver
//
// Description:
//   Consider
//     x[i+n] = a[0] x[i] + a[1] x[i+1] + ... + a[n-1] x[i+n-1].
//   with initial solution x[0], x[1], ..., x[n-1].
//   We compute k-th term of x in O(n^2 log k) time.
//
// Algorithm:
//   Since x[k] is linear in x[0], ..., x[n-1],
//   there exists function f: Z -> R^n such that
//     x[k] = f(k)[0] x[0] + ... + f(k)[n-1] x[n-1].
//   Here, f satisfies the following identities:
//     x[2k] = f(k)[0] x[k] + ... + f(k)[n-1] x[k+n-1]
//           = f(k)[0] (f(k)[0] x[0] + ... + f(k)[n-1] x[n-1])
//           + f(k)[1] (f(k)[0] x[1] + ... + f(k)[n-1] x[n-1+1])
//           + ...
//           = sum{0<=i<n, 0<0j<n} f(k)[i] f(k)[j] x[i+j].
//           = t[0] x[0] + ... + t[2*n-1] x[2*n-1].
//   Also, we have
//     x[n+k] = a[0] x[n] + ... + a[n+k-1] x[n+k-1],
//   thus
//     t[0] x[0] + ... + t[2*n-1] x[2*n-1]
//     = t[0] x[0] + ... + t[2*n-1] (a[0] x[n] + ... + a[n+k-1] x[n+k-1])
//     = t'[0] x[0] + ... + t[2*n-2]' x[2*n-2].
//     ...
//     = t''[0] x[0] + ... + t''[n-1] x[n-1].
//   This means, we can compute f(2*k) from f(k) in O(n^2) time.
//
// Complexity:
//   O(n^2 log k) time, O(n log k) space.
//
#include <iostream>
#include <vector>
#include <cstdio>
#include <algorithm>
#include <functional>

using namespace std;

#define fst first
#define snd second
#define all(c) ((c).begin()), ((c).end())

int linear_recurrence(vector<int> a, vector<int> x, int k) {
  int n = a.size();
  vector<int> t(2*n+1);

  function<vector<int> (int)> rec = [&](int k) {
    vector<int> c(n);
    if (k < n) c[k] = 1;
    else {
      vector<int> b = rec(k / 2);
      fill(all(t), 0);
      for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
          t[i+j+(k&1)] += b[i]*b[j];
      for (int i = 2*n-1; i >= n; --i)
        for (int j = 0; j < n; j++)
          t[i-n+j] += a[j]*t[i];
      for (int i = 0; i < n; ++i)
        c[i] = t[i];
    }
    return c;
  };
  vector<int> c = rec(k);
  int ans = 0;
  for (int i = 0; i < x.size(); ++i)
    ans += c[i]*x[i];
  return ans;
}


int main() {
  // x[n+k] = x[n] + 2*x[n+1] + 3*x[n+2];
  // x[0] = 6, x[1] = 5, x[2] = 4.
  // 10-th term = 220696
  cout << linear_recurrence({1,2,3}, {6,5,4}, 10) << endl;
}
```

9 maximal circle clique

```
/*
You are given a disk of radius R with its center at the origin and N
    integer points outside that disk. Let us consider a graph on these
    points as vertices, where points A and B are connected by an edge if
    and only if the line AB does not intersect the disk.

Find the maximal clique in this graph
*/
#include <bits/stdc++.h>
using namespace std ;

const int N = 5005 ;
const double pi = acos(-1.0) ;

struct Data {
    double a , b ;
    int id ;
    bool operator < (const Data &p) {
        return a < p.a ;
    }
}P[N];

bool comp(Data p, Data q) {
    p.a < q.a ;
}

int main() {
    int n , r ;
    cin >> n >> r ;
    for(int i = 1 ; i <= n ; i++) {
        int x , y ;
        cin >> x >> y ;
        double u = atan2((double)y,(double)x) ;
        double range = acos(r/sqrt(x*x+y*y)) ;
        double a = u-range , b = u+range ;
        if (a <= -pi) {
            a += 2.0*pi ;
        }
        if (b > pi) {
            b -= 2.0*pi ;
        }
        if (a > b) swap(a,b);
        P[i] = {a,b,i} ;
    }
    sort(P+1,P+n+1) ;
    int ans = 0 ;
    vector <int> res ;
    for(int i = 1; i <= n; i++) {
        vector<int> par(n+1,-1) ;
        vector < pair<double,int> > cur ;
        cur.push_back({P[i].b,i}) ;
        for(int j = i+1 ; j <= n ; j++) {
            if (P[j].a > P[i].b) break ;
            if (P[j].b < P[i].b) continue ;
            int sz =
                upper_bound(cur.begin(),cur.end(),make_pair(P[j].b,j))-cur.begin(
                ;
            if (sz == cur.size()) {
                par[j] = cur.back().second ;
                cur.push_back({P[j].b,j}) ;
            }
            else {
                cur[sz] = make_pair(P[j].b,j) ;
                par[j] = cur[sz-1].second;
            }
        }
        int cur_ans = cur.size() ;
        if (cur_ans > ans) {
            ans = cur_ans ;
            res.clear() ;
            int p = cur.back().second ;
            while(p!=-1) {
                res.push_back(P[p].id) ;
                p = par[p] ;
            }
        }
    }
    cout << ans << endl ;
    for(int x : res) {
        cout << x << " " ;
    }
}
```

10 median slope for all pair points

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree < pair<int,int> , null_type , less< pair<int,int> >
    ,rb_tree_tag,tree_order_statistics_node_update > ordered_set;

typedef long long db;

double dx, dy;

struct Point{
    db x, y;
    Point() {}
    Point(db x_,db y_) {x=x_,y=y_;}
    Point operator + (Point &p) {
        return Point(x+p.x,y+p.y);
    }
    Point operator - (Point &p) {
        return Point(x-p.x,y-p.y);
    }
    Point operator * (double t) {
        return Point(x*t,y*t);
    }
    db operator * (Point &p) {
        return x*p.x + y*p.y;
    }
    db operator ^ (Point &p) {
        return x*p.y - y*p.x;
    }
    bool operator < (const Point &p) const {
        return dx*y - dy*x < dx*p.y - dy*p.x;
    }
};

const double pi = acos(-1.0);
const int INF = 1e8;

double solve(vector<Point> &P,long long kth) {
    double lo = 0 , hi = pi;
    int n = P.size();
    for(int it = 1 ; it <= 35 ; it++) {
        double mid = (hi+lo)/2.0;
        long long cnt = 0;
        dx = cos(mid), dy = sin(mid);
        sort(P.begin(),P.end());
        ordered_set S ;
        for(int i = 0 ; i < n ; i++) {
            int c = S.order_of_key(make_pair(-P[i].y,INF));
            cnt += c;
            S.insert(make_pair(-P[i].y,i));
        }
        if(cnt >= kth) {
            hi = mid;
        } else {
            lo = mid;
        }
    }
    return lo;
}


int main() {

   // freopen ("in.txt","r",stdin);
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n; cin >> n;
    vector<Point> P(n);
    for(int i = 0; i < n; i++) {
        cin >> P[i].x >> P[i].y;
    }
    double ans;
    long long m = (1ll*n*(n-1))/2;
    if(m&1) {
        ans = solve(P,m/2 + 1);
    }
    else {
        ans = 0.5*(solve(P,m/2)+solve(P,m/2+1));
    }
    cout << setprecision(12) << fixed << ans << endl;
}
```

11 prime-sig

```
#include<bits/stdc++.h>
using namespace std;

typedef long long LL;
const int MX = 100;
vector<int> primes;

void sieve() {
    vector<bool> isp(MX, 1);
    for (int i=2; i<MX; i++)
        if (isp[i]) {
            primes.push_back(i);
            for (int j=2*i; j<MX; j+=i)
                isp[j] = 0;
        }
}


LL LIM;
vector<pair<vector<int>, LL>> ans;
vector<int> ps;

void go(int idx, LL val, int mx) {
    assert(ans.size() < 100000);
    assert(idx < primes.size());

    ans.push_back({ps, val});
    int p = primes[idx];
    ps.push_back(0);

    for (int i=1; i<=mx; i++) {
        if (val > LIM/p)  break;
        ps.back()++;
        val *= p;
        go(idx+1, val, i);
    }
    ps.pop_back();
}


///first = signature, second = min value with that signature
vector<pair<vector<int>, LL>> getAllSignature(LL lim) {
    LIM = lim;
    ans.clear();
    ps.clear();
    go(0, 1, 100);
    return ans;
}

int main() {
    sieve();

    map<int, LL> mp;

    for (auto pr: getAllSignature(LLONG_MAX)) {
        int ans = 1;
        for (int x: pr.first) ans *= (x+1);
        if (mp.count(ans) == 0) mp[ans] = pr.second;
        else                    mp[ans] = min(mp[ans], pr.second);
    }

    int n;
    cin>>n;
    cout<<mp[n]<<endl;
}
```