Sharif University of Technology - Kolompeh - Notebook

Contents

1 Geometry

1.1 Geometry Primitives

```
const int ON = 0, LEFT = 1, RIGHT = -1, BACK = -2, FRONT = 2, IN = 3,
    OUT = -3;
bool byX(const pt &a, const pt &b){
        if (Equ(a.x, b.x)) return Lss(a.y, b.y);
        return Lss(a.x, b.x);
}
bool byY(const pt &a, const pt &b){
        if (Equ(a.y, b.y)) return Lss(a.x, b.x);
        return Lss(a.y, b.y);
}
struct cmpXY{ bool operator ()(const pt &a, const pt &b){ return byX(a
    , b); } };
struct cmpYX{ bool operator ()(const pt &a, const pt &b){ return byY(a
    , b); } };
istream& operator >> (istream &in, pt &p){ in >> p.x >> p.y; return in
    ; }
ostream& operator << (ostream &out, pt p){ out << p.x << ' ' << p.y;
    return out; }
pt rot(pt a){ return pt(-a.y, a.x); }
pt proj(pt a, pt b, pt c){
        b = b-a, c = c-a;
        return a + (b*c)/(b*b)*b;
}
pt reflect(pt a, pt b, pt c){
        pt d = c;
        b = b-a, c = c-a;
        return d + (c^b)/abs(b)*rot(unit(b))*2;
}
bool intersect(pt a, pt b, pt c, pt d){
        int as = dir(c, d, a), bs = dir(c, d, b),
                cs = dir(a, b, c), ds = dir(a, b, d);
        if (as && as == bs || cs && cs == ds)
                return false;
        else if (as || bs || cs || ds)
                return true;
        for (int j = 0; j < 2; j++){
                cord mX = min(a.x, b.x), MX = max(a.x, b.x),
                         mY = min(a.y, b.y), MY = max(a.y, b.y);
                for (int k = 0; k < 2; k++){
                        if (c.x + EPS > mX && c.x < MX + EPS && c.y +
                            EPS > mY && c.y < MY + EPS)
                                return true;
                        swap(c, d);
```

```
			}
			swap(a, c); swap(b, d);
		}
		return false;
}
pt intersection(pt a, pt b, pt c, pt d){
		cord c1 = (b-a)^(c-a), c2 = (b-a)^(d-a);
		return (c1*d - c2*c)/(c1 - c2);
}
ld signedArea(vector<pt> &p){
		int n = p.size();
		cord res = 0;
		for (int i = 0; i < n; i++)
			res += (p[i]^p[(i+1)%n]);
		return (ld)res/2;
}
//relative position of c toward ab
int relpos(pt a, pt b, pt c){
		b = b-a; c = c-a;
		if (Grt(b^c, 0)) return LEFT;
		if (Lss(b^c, 0)) return RIGHT;
		if (Lss(b*c, 0)) return BACK;
		if (Grt(b*c, abs(b))) return FRONT;
		return ON;
}
//distance of a point from a line segment
ld distLSP(pt a, pt b, pt c){
		int rpos = relpos(a, b, proj(a, b, c));
		if (rpos == BACK) return abs(c-a);
		if (rpos == FRONT) return abs(c-b);
		b = b-a, c = c-a;
		return abs(b^c/abs(b));
}
//distance between two line segments
ld distLS(pt a, pt b, pt c, pt d){
		if (intersect(a, b, c, d)) return 0;
		return min(min(distLSP(a, b, c), distLSP(a, b, d)), min(
		    distLSP(c, d, a), distLSP(c, d, b)));
}
//angles less than or equal to 180
bool isConvex(vector<pt> &p){
		int n = p.size();
		bool neg = false, pos = false;
		for (int i = 0; i < n; i++){
			int rpos = relpos(p[i], p[(i+1)%n], p[(i+2)%n]);
			if (rpos == LEFT) pos = true;
			if (rpos == RIGHT) neg = true;
		}
		return (neg&pos) == false;
}
int crossingN(vector<pt> &p, pt a){
		int n = p.size();
		pt b = a;
		for (pt q : p)
			b.x = max(b.x, q.x);
		int cn = 0;
		for (int i = 0; i < n; i++){
			pt q1 = p[i], q2 = p[(i+1)%n];
			if (intersect(a, b, q1, q2) && (dir(a, b, q1) == 1 ||
			    dir(a, b, q2) == 1))
				cn++;
		}
		return cn;
}
int windingN(vector<pt> &p, pt a){
		int n = p.size();
		pt b = a;
		for (pt q : p)
			b.x = max(b.x, q.x);
		int wn = 0;
		for (int i = 0; i < n; i++){
			pt q1 = p[i], q2 = p[(i+1)%n];
			if (intersect(a, b, q1, q2)){
				int ps = dir(a, b, q1), qs = dir(a, b, q2);
				if (qs >= 0) wn++;
				if (ps >= 0) wn--;
			}
		}
		return wn;
}
//returns IN, ON or OUT
int pointInPoly(vector<pt> &p, pt a){
		int n = p.size();
		for (int i = 0; i < n; i++)
			if (relpos(p[i], p[(i+1)%n], a) == ON)
				return ON;
		return (crossingN(p, a)%2 ? IN : OUT);
		//return (windingN(po, a) ? IN : OUT);
}
pair <pt, pt> nearestPair(vector<pt> &po){
		int n = po.size();
		sort(po.begin(), po.end(), cmpXY());
		multiset <pt, cmpYX> s;
		ld rad = abs(po[1]-po[0]);
		pair <pt, pt> res = {po[0], po[1]};
		int l = 0, r = 0;
		for (int i = 0; i < n; i++){
			while (l < r && Geq(po[i].x - po[l].x, rad))
				s.erase(po[l++]);
			while (r < i && Leq(po[r].x, po[i].x))
				s.insert(po[r++]);
			for (auto it = s.lower_bound(pt(po[i].x, po[i].y-rad))
			    ; it != s.end(); it++){
				if (Grt(it->y, po[i].y+rad))
					break;
				ld cur = abs(po[i] - (*it));
				if (Lss(cur, rad)){
					rad = cur;
					res = {*it, po[i]};
				}
			}
		}
		return res;
}
//Cuts polygon with line ab and returns the left cut polygon
vector<pt> convexCut(vector<pt> &po, pt a, pt b){
		int n = po.size();
		vector<pt> res;
		for (int i = 0; i < n; i++){
			if (dir(a, b, po[i]) >= 0)
				res.push_back(po[i]);
			if (abs(dir(a, b, po[i]) - dir(a, b, po[(i+1)%n])) ==
```

```
                2)
                    res.push_back(intersection(a, b, po[i], po[(i
                        +1)%n]));
        }
        return res;
}
//slightly line
pair <pt, pt> get_segment(pt a, pt b){
    const int deltax = b.x - a.x;
    const int deltay = b.y - a.y;
    const int k = 100001;
    pt aa(a.x - k * deltax, a.y - k * deltay);
    pt bb(b.x + deltax, b.y + deltay);

    static const int dx[4] = { -1, 0, 1, 0 };
    static const int dy[4] = { 0, 1, 0, -1 };
    for (int d = 0; d < 4; ++d) {
        pt aaa(aa.x + dx[d], aa.y + dy[d]);
        pt bbb(bb.x, bb.y);
        if (dir(aaa, bbb, a) >= 0) continue;
        if (dir(aaa, bbb, b) == 0) continue;
        return {aaa, bbb};
    }
}
pair<int, int> tangent(vector <pt> &A0, vector <pt> &B0){
    vector <pair<pt, int>> A, B;
    for (int i = 0; i < sz(A0); i++)
        A.pb({A0[i], i});
    for (int i = 0; i < sz(B0); i++)
        B.pb({B0[i], i});
    sort(A.begin(), A.end());
    sort(B.begin(), B.end());
    A = convex_hull(A);
    B = convex_hull(B);
    int ia = 0, ib = 0;
    //direction must be considered
    while (1){
        bool fin = true;
        for (; dir(A[ia].F, B[ib].F, B[(ib+sz(B)-1)%sz(B)].F) < 0 ||
            dir(A[ia].F, B[ib].F, B[(ib+1)%sz(B)].F) < 0; ib = (ib+1)%
            sz(B))
            fin = false;
        for (; dir(B[ib].F, A[ia].F, A[(ia+sz(A)-1)%sz(A)].F) > 0 ||
            dir(B[ib].F, A[ia].F, A[(ia+1)%sz(A)].F) > 0; ia = (ia+1)%
            sz(A))
            fin = false;
        if (fin) break;
    }
    return {A[ia].S, B[ib].S};
}

//Sweep Line Example
int main(){
        vector <pt> adds, rems;
        vector <pair<pt, pt>> query;
        int it;
        cin >> it;
        for (int i = it; i; i--){
                pt p, q;
                cin >> p >> q;
                if (Equ(p.y, q.y)){
                        if (Lss(q.x, p.x))
                                swap(p, q);
                        adds.push_back(p);
                        rems.push_back(q);
                }
                else{
                        if (Lss(q.y, p.y))
                                swap(p, q);
                        query.push_back({p, q});
                }
        }
        sort(adds.begin(), adds.end());
        sort(rems.begin(), rems.end());
        sort(query.begin(), query.end());
        multiset <ld> ys;
        int iadd = 0, irem = 0;
        int ans = 0;
        for (auto p : query){
                while (iadd < adds.size() && Leq(adds[iadd].x, p.F.x))
                        ys.insert(adds[iadd++].y);
                while (irem < rems.size() && Lss(rems[irem].x, p.F.x)
                    && ys.find(rems[irem].y) != ys.end())
                        ys.erase(ys.find(rems[irem++].y));
                int cur = distance(ys.lower_bound(p.F.y), ys.
                    upper_bound(p.S.y));
                ans += cur;
        }
        cout << ans << endl;
        return 0;
}

//rotate b with center=a theta radian
PT rotate(PT a, PT b, ld theta){
        return (b-a)*polar<ld>(1, theta) + a;
}
```

1.2 Line intersection

```
point intersection(point a, point b, point c, point d){
        point ab = b - a;
        point cd = d - c;
        point ac = c - a;
        double alpha = cross(ac, cd) / cross(ab, cd);
        return a + alpha * ab;
}
```

1.3 Line and circle intersection

```
// return pair<point, point> which is intersections point
// for each point if it's not exist, return (INF,INF)
typedef pair<point,point> ppp;
const ld INF = 1e18;
const ld eps = 1e-15;
ppp line_circle_intersection(point p1,point p2,point o,ld r){
  point q = dot(o-p1,p2-p1)/dist(p1,p2)*(p2-p1) + p1;
  ld d = r*r - dist(o,q);
  if(d<eps && d>-eps) return ppp(q, point(INF,INF));
  if(d<0) return ppp(point(INF,INF), point(INF,INF));
```

```
  point dif = sqrt(d/dist(p1,p2))*(p1-p2);
  return ppp(q-dif,q+dif);
}
```

1.4 Intersection of two circles

```
#define _USE_MATH_DEFINES
const int MAX_N = 2e5+10;
const int INF = 1e9;
const ld eps = 1e-8;
struct circle {
public:
  ld r;
  point o;
  circle(ld rr, ld x, ld y) {
    r = rr;
    o.x = x;
    o.y = y;
  }
  ld S() {
    return M_PI*r*r;
  }
  ld distance(point p1, point p2) { return hypot(p2.x-p1.x,p2.y-p1.y);
      }
  /*
    0 = other is inside this, zero point
    1 = other is tangent inisde of this, one point
    2 = other is intersect with this, two point
    3 = other is tangent outside of this, one point
    4 = other is outside of this, zero point
  */
  pair<int, vector<point> > intersect(circle other) {
    vector<point> v;
    ld sumr = other.r + r;
    ld rr = r - other.r;
    ld dis = distance(o, other.o);
    ld a = (r*r - other.r*other.r + dis*dis)/(2*dis);
    ld h = sqrt(r*r-a*a);
    point p2(o.x, o.y);
    p2.x = a*(other.o.x - o.x)/dis;
    p2.y = a*(other.o.y - o.y)/dis;
    if(is_zero(sumr-dis)) {
      v.push_back(p2);
      return make_pair(3, v);
    }
    if(is_zero(rr - dis)) {
      v.push_back(p2);
      return make_pair(1, v);
    }
    if(dis <= rr)
      return make_pair(0, v);
    if(dis >= sumr)
      return make_pair(4, v);
    point p3(p2.x + h*(other.o.y - o.y)/dis, p2.y - h*(other.o.x - o.x
        )/dis);
    point p4(p2.x - h*(other.o.y - o.y)/dis, p2.y + h*(other.o.x - o.x
        )/dis);
    v.push_back(p3);
    v.push_back(p4);
    return make_pair(2, v);
  }
  ld f(ld l, ld r, ld R){
    ld cosa = (l*l + r*r - R*R)/(2.0*r*l);
    ld a = acos(cosa);
    return r*r*(a - sin(2*a)/2);
  }
  ld intersection_area(circle c2) {
    ld l = distance(o, c2.o);
    if(l >= r + c2.r) return 0;
    else if(c2.r >= l + r) return S();
    else if(r >= l + c2.r) return c2.S();
    return f(l, r, c2.r) + f(l, c2.r, r);
  }
};
```

1.5 Rotating Calipers

```
vector <pair<pt, pt>> get_antipodals(vector <pt> &p){
        int n = sz(p);
        sort(p.begin(), p.end());
        vector <pt> U, L;
        for (int i = 0; i < n; i++){
                while (sz(U) > 1 && side(U[sz(U)-2], U[sz(U)-1], p[i])
                     >= 0)
                        U.pop_back();
                while (sz(L) > 1 && side(L[sz(L)-2], L[sz(L)-1], p[i])
                     <= 0)
                        L.pop_back();
                U.pb(p[i]);
                L.pb(p[i]);
        }
        vector <pair<pt, pt>> res;
        int i = 0, j = sz(L)-1;
        while (i+1 < sz(U) || j > 0){
                res.pb({U[i], L[j]});
                if (i+1 == sz(U))   j--;
                else if (j == 0)    i++;
                else if (cross(L[j]-L[j-1], U[i+1]-U[i]) >= 0)  i++;
                else    j--;
        }
        return res;
}
```

1.6 Delaunay Triangulation $O(n^2)$

```
struct Delaunay{
    vector <pt> p;
    vector <int> to, nxt, perm;
    int add_edge(int q, int bef=-1){
        int cnt = sz(to);
        to.pb(q);
        nxt.pb(-1);
        if (bef != -1){
            nxt[bef] = cnt;
            to.pb(to[bef]);
            nxt.pb(-1);
        }
        return cnt;
```

```cpp
}
bool onconvex(int e){
    if (nxt[nxt[nxt[e]]] != e) return true;
    if (dir(p[to[e^1]], p[to[e]], p[to[nxt[e]]]) < 0) return true;
    return false;
}
int before(int e){
    int cur = e, last = -1;
    do{
        last = cur;
        cur = nxt[cur^1];
    }while (cur != e);
    return last^1;
}
void easy_triangulate(){
    to.clear();
    nxt.clear();
    perm = vector<int>(sz(p));
    for (int i = 0; i < sz(p); i++)
        perm[i] = i;
    sort(perm.begin(), perm.end(), [&] (int i, int j){
        return p[i] < p[j]; });
    sort(p.begin(), p.end());
    if (dir(p[0], p[1], p[2]) > 0){
        swap(p[1], p[2]);
        swap(perm[1], perm[2]);
    }
    int to0 = add_edge(0), to0c = add_edge(2),
    to1 = add_edge(1), to1c = add_edge(0),
    to2 = add_edge(2), to2c = add_edge(1);
    nxt[to1] = to2; nxt[to2] = to0;
    nxt[to0] = to1; nxt[to0c] = to2c;
    nxt[to2c] = to1c; nxt[to1c] = to0c;
    int e = to0;
    bool D2 = true;
    for (int i = 3; i < sz(p); i++){
        pt q = p[i];
        if (D2){
            int edge = e;
            do{
                if (dir(q, p[to[edge^1]], p[to[edge]])){
                    D2 = false;
                    break;
                }
                edge = nxt[edge];
            } while (edge != e);
        }
        vector <int> vis;
        if (D2){
            while (p[to[e^1]] < p[to[e]])
                e = nxt[e];
            vis.pb(e);
            e = nxt[e];
        }
        else{
            while (dir(q, p[to[e^1]], p[to[e]]) <= 0 || dir(q, p[
                to[e^1]], p[to[before(e)^1]]) < 0)
                e = nxt[e];
            while (dir(q, p[to[e^1]], p[to[e]]) > 0){
                vis.pb(e);
                e = nxt[e];
            }
        }
        int b = before(vis[0]);
        int ex = add_edge(i, b);
        int last = ex^1;
        for (int edge : vis){
            nxt[last] = edge;
            int eq = add_edge(i, edge);
            nxt[edge] = eq;
            nxt[eq] = last;
            last = eq^1;
        }
        nxt[ex] = last;
        nxt[last] = e;
    }
}
bool incircle(pt a, pt b, pt c, pt d){
    if (dir(a, b, c) < 0)
        swap(b, c);
    return a.z() * (b.x * (c.y - d.y) - c.x * (b.y - d.y)
                    + d.x * (b.y - c.y))
    - b.z() * (a.x * (c.y - d.y) - c.x  * (a.y - d.y) + d.x * (a.y
         - c.y))
    + c.z() * (a.x * (b.y - d.y) - b.x * (a.y - d.y) + d.x * (a.y
        - b.y))
    - d.z() * (a.x * (b.y - c.y) - b.x * (a.y - c.y) + c.x * (a.y
        - b.y)) > 0;
}
bool locally(int e){
    pt a = p[to[e^1]], b = p[to[e]], c = p[to[nxt[e]]], d = p[to[
        nxt[e^1]]];
    if (onconvex(e)) return true;
    if (onconvex(e^1)) return true;
    if (incircle(a, b, c, d)) return false;
    if (incircle(b, a, d, c)) return false;
    return true;
}
void flip(int e){
    int a = nxt[e], b = nxt[a],
    c = nxt[e^1], d = nxt[c];
    nxt[d] = a;
    nxt[b] = c;
    to[e] = to[c];
    nxt[a] = e;
    nxt[e] = d;
    to[e^1] = to[a];
    nxt[c] = e^1;
    nxt[e^1] = b;
}
void delaunay_triangulate(){
    if (sz(to) == 0)
        easy_triangulate();
    bool *mark = new bool[sz(to)];
    fill(mark, mark + sz(to), false);
    vector <int> bad;
    for (int e = 0; e < sz(to); e++){
        if (!mark[e/2] && !locally(e)){
            bad.pb(e);
            mark[e/2] = true;
        }
    }
```

```
        while (sz(bad)){
            int e = bad.back();
            bad.pop_back();
            mark[e/2] = false;
            if (!locally(e)){
                int to_check[4] = {nxt[e], nxt[nxt[e]], nxt[e^1], nxt[
                    nxt[e^1]]};
                flip(e);
                for (int i = 0; i < 4; i++)
                    if (!mark[to_check[i]/2] && !locally(to_check[i]))
                        {
                        bad.pb(to_check[i]);
                        mark[to_check[i]/2] = true;
                    }
            }
        }
        for (int e = 0; e < sz(to); e++)
            assert(locally(e));
    }
    vector <tri> get_triangles(){
        vector <tri> res;
        bool *mark = new bool[sz(to)];
        fill(mark, mark + sz(to), false);
        for (int e = 0; e < sz(to); e++){
            if (mark[e]) continue;
            if (onconvex(e)) continue;
            pt a = p[to[e^1]], b = p[to[e]], c = p[to[nxt[e]]];
            mark[e] = mark[nxt[e]] = mark[nxt[nxt[e]]] = true;
            res.pb(tri(perm[to[e^1]], perm[to[e]], perm[to[nxt[e]]]));
        }
        return res;
    }
    vector <pair<ls, pt>> get_voronoi_edges(){
        vector <pair<ls, pt>> res;
        for (int e = 0; e < sz(to); e++){
            pt a = p[to[e^1]], b = p[to[e]], c = p[to[nxt[e]]], d = p[
                to[nxt[e^1]]];
            if (onconvex(e^1)){
                pt o1 = center(a, b, c),
                o2 = (a+b)/2;
                pt ab = (b-a);
                pt per(ab.y, -ab.x);
                o2 = o2 + per*100000; //infinity
                res.pb({{o1, o2}, a});
                continue;
            }
            if (onconvex(e)) continue;
            if (e&1) continue;
            res.pb({{center(a, b, c), center(b, a, d)}, a});
        }
        return res;
    }
    Delaunay(vector <pt> &p):p(p){}
};
```

1.7 Delaunay $O(n \log^2 n)$

```
const int MAXN = 100 * 1000 + 10;
const int MAXLG = 20;
const int INF = 100 * 1000 * 1000 + 10;
const int MAXPOINTS = MAXN * MAXLG;
typedef pair<int ,int> point;
struct tria{
        int a, b, c;
        tria(int _a, int _b, int _c){
                a = _a; b = _b; c = _c;
        }
        tria(){a = b = c = 0;}
};
struct Delaunay {
        typedef pair<point, int> ppi;
        typedef pair<int, int> pii;
        typedef pair<pii, int> pip;
        tria t[MAXPOINTS];
        bool mrk[MAXPOINTS];
        int last[MAXPOINTS];
        int childs[MAXPOINTS][3];
        int cnt;
        vector<ppi> points;
        set<pip> edges;
        vector<tria> res;
        int n;
        inline void add_edge(int a, int b, int c){
                edges.insert(pip(pii(min(a, b), max(a, b)), c));
        }
        inline void remove_edge(int a, int b, int c){
                edges.erase(pip(pii(min(a, b), max(a, b)), c));
        }
        int add_triangle(int a, int b, int c){
                if (cross(points[b].first - points[a].first, points[c
                    ].first - points[a].first) == 0)
                        return -1;
                if (cross(points[b].first - points[a].first, points[c
                    ].first - points[a].first) < 0)
                        swap(b, c);
                add_edge(a, b, cnt);
                add_edge(b, c, cnt);
                add_edge(c, a, cnt);
                t[cnt] = tria(a, b, c);
                childs[cnt][0] = childs[cnt][1] = childs[cnt][2] = -1;
                mrk[cnt] = false;
                last[cnt] = -1;
                cnt++;
                return cnt - 1;
        }
        inline void remove_triangle(int v){
                childs[v][0] = childs[v][1] = childs[v][2] = -1;
                remove_edge(t[v].a, t[v].b, v);
                remove_edge(t[v].b, t[v].c, v);
                remove_edge(t[v].c, t[v].a, v);
        }
        inline void relax_edge(const int &a, const int &b){
                pii key(min(a, b), max(a, b));
                set<pip>::iterator it = edges.lower_bound(pip(key, -1)
                    );
                if (it == edges.end() || it->first != key)
                        return;
                set<pip>::iterator it2 = it;
                it2++;
                if (it2 == edges.end() || it2->first != key)
                        return;
```

```cpp
		int c1 = t[it->second].a + t[it->second].b + t[it->
		    second].c - a - b;
		int c2 = t[it2->second].a + t[it2->second].b + t[it2->
		    second].c - a - b;
		if (c1 > n || c2 > n)
			return;
		if (inCircle(points[a].first, points[b].first, points[
		    c1].first, points[c2].first) < 0 ||
				inCircle(points[a].first, points[b].
				    first, points[c2].first, points[c1
				    ].first) < 0)
		{
			int v1 = it->second;
			int v2 = it2->second;
			remove_triangle(v1);
			remove_triangle(v2);
			mrk[v1] = mrk[v2] = true;
			childs[v1][0] = childs[v2][0] = add_triangle(a
			    , c1, c2);
			childs[v1][1] = childs[v2][1] = add_triangle(b
			    , c1, c2);
			relax(childs[v1][0]);
			relax(childs[v1][1]);
		}
	}
	inline void relax(int v){
		relax_edge(t[v].a, t[v].b);
		relax_edge(t[v].b, t[v].c);
		relax_edge(t[v].c, t[v].a);
	}
	inline bool inLine(int a, int b, int c){
		return cross(points[b].first - points[a].first, points
		    [c].first - points[a].first) >= 0;
	}
	inline bool inTriangle(int a, int b, int c, int d){
		return inLine(a, b, d) && inLine(b, c, d) && inLine(c,
		     a, d);
	}
	void find(int v, int p, int cl){
		if (last[v] == cl)
			return;
		bool reached = false;
		last[v] = cl;
		for (int i = 0; i < 3; i++)
		{
			int u = childs[v][i];
			if (u == -1)
				continue;
			reached = true;
			if (mrk[u] || inTriangle(t[u].a, t[u].b, t[u].
			    c, p))
				find(u, p, cl);
		}
		if (reached)
			return ;
		remove_triangle(v);
		childs[v][0] = add_triangle(p, t[v].a, t[v].b);
		childs[v][1] = add_triangle(p, t[v].b, t[v].c);
		childs[v][2] = add_triangle(p, t[v].c, t[v].a);
		relax(childs[v][0]);
		relax(childs[v][1]);
		relax(childs[v][2]);
	}
	void getRes(int v, int cl){
		if (last[v] == cl)
			return;
		last[v] = cl;
		bool reached = false;
		for (int i = 0; i < 3; i++)
		{
			int u = childs[v][i];
			if (u == -1)
				continue;
			reached = true;
			getRes(u, cl);
		}
		if (!reached && t[v].a < n && t[v].b < n && t[v].c < n
		    )
			res.push_back(t[v]);
	}
	vector<tria> delaunay(vector<point> v){
		cnt = 0;
		int cl = 0;
		points.clear();
		for (int i = 0; i < v.size(); i++)
			points.push_back(ppi(v[i], i));
		random_shuffle(points.begin(), points.end());
		n = points.size();
		points.push_back(ppi(point(INF, INF), n));
		points.push_back(ppi(point(-INF * 3, INF), n + 1));
		points.push_back(ppi(point(INF, -INF * 3), n + 2));
		int root = add_triangle(n, n + 1, n + 2);
		for (int i = 0; i < n; i++){
			//	cout << "" << inTriangle(n,n+1, n+2, i
			    ) << endl;
			find(root, i, cl++);
		}
		res.clear();
		getRes(root, cl++);
		for (int i = 0; i < res.size(); i++){
			res[i].a = points[res[i].a].second;
			res[i].b = points[res[i].b].second;
			res[i].c = points[res[i].c].second;
		}
		return res;
	}
};
long double getRadius(pointD a, pointD b, pointD c){
	pointD v1 = norm(b - a) + ((a + b) / 2);
	pointD v2 = norm(c - b) + ((b + c) / 2);
	pointD center =  intersect((a + b) / 2, v1, (b + c) / 2, v2);
	pointD ret = a - center;
	return sqrt(dot(ret, ret));
}
Delaunay d;
int main(){
	srand(2019);
	ios::sync_with_stdio(false);
	cin.tie(0);
	int n;
	cin >> n;
	vector<point> v;
```

```
	for (int i = 0; i < n; i++)
	{
		int x, y;
		cin >> x >> y;
		v.push_back(point(x, y));
	}
	vector<tria> ans = d.delaunay(v);
	long double res = 0;
	for (int i = 0; i < ans.size(); i++)
		res = max(res, getRadius(v[ans[i].a], v[ans[i].b], v[
			ans[i].c]));
	cout.precision(6);
	cout << fixed << res << endl;
}
```

1.8 Convex Hull 3D

```
struct point{
  int X,Y,Z;
  point(int x=0,int y=0,int z=0){
    X=x; Y=y; Z=z;
  }
  bool operator==(const point& rhs) const {
    return (rhs.X==this->X && rhs.Y==this->Y && rhs.Z==this->Z);
  }
  bool operator<(const point& rhs) const {
    return rhs.X > this->X || (rhs.X == this->X && rhs.Y > this->Y) ||
         (rhs.X==this->X && rhs.Y==this->Y && rhs.Z>this->Z);
  }
};

const int maxn=1000;
int n;
point P[maxn];
vector<point>ans;
queue<pii>Q;
set<pii>mark;
int cross2d(point p,point q){ return p.X*q.Y-p.Y*q.X;}
point operator -(point p,point q){ return point(p.X-q.X,p.Y-q.Y,p.Z-q.
    Z); }
point _cross(point u,point v){ return point(u.Y*v.Z-u.Z*v.Y,u.Z*v.X-u.
    X*v.Z,u.X*v.Y-u.Y*v.X); }
point cross(point o,point p,point q){ return _cross(p-o,q-o);}
point shift(point p) { return point(p.Y,p.Z,p.X);}
point norm(point p)
{
  if(p.Y<p.X || p.Z<p.X) p=shift(p);
  if(p.Y<p.X) p=shift(p);
  return p;
}

int main()
{
  cin>>n;
  int mn=0;
  for(int i=0;i<n;i++){
    cin>>P[i].X>>P[i].Y>>P[i].Z;
    if(P[i]<P[mn]) mn=i;
  }
  int nx=(mn==0);
  for(int i=0;i<n;i++)
    if(i!=mn && i!=nx && cross2d(P[nx]-P[mn],P[i]-P[mn])>0)
      nx=i;
  Q.push(pii(mn,nx));
  while(!Q.empty())
    {
      int v=Q.front().first,u=Q.front().second;
      Q.pop();
      if(mark.find(pii(v,u))!=mark.end()) continue;
      mark.insert(pii(v,u));
      int p=-1;
      for(int q=0;q<n;q++)
        if(q!=v && q!=u)
          if(p==-1 || dot(cross(P[v],P[u],P[p]),P[q]-P[v])<0)
            p=q;
      ans.push_back(norm(point(v,u,p)));
      Q.push(pii(p,u));
      Q.push(pii(v,p));
    }
  sort(ans.begin(),ans.end());
  ans.resize(unique(ans.begin(),ans.end())-ans.begin());
  for(int i=0;i<ans.size();i++)
    cout<<ans[i].X<<" "<<ans[i].Y<<" "<<ans[i].Z<<endl;
}
```

1.9 Half Plane Intersection

```
typedef int T;
typedef long long T2;
typedef long long T4; // maybe int128_t

const int MAXLINES = 100 * 1000 + 10;
const int INF = 20 * 1000 * 1000;

typedef pair<T, T> point;
typedef pair<point, point> line;

#define X first
#define Y second
#define A first
#define B second

// REPLACE ZERO WITH EPS FOR DOUBLE

point operator - (const point &a, const point &b) {
	return point(a.X - b.X, a.Y - b.Y);
}


T2 cross(point a, point b) {
	return ((T2)a.X * b.Y - (T2)a.Y * b.X);
}

bool cmp(line a, line b) {
	bool aa = a.A < a.B;
	bool bb = b.A < b.B;
	if (aa == bb) {
		point v1 = a.B - a.A;
		point v2 = b.B - b.A;
		if (cross(v1, v2) == 0)
```

```
				return cross(b.B - b.A, a.A - b.A) > 0;
			else
				return cross(v1, v2) > 0;
		}
		else
			return aa;
}

bool parallel(line a, line b) {
	return cross(a.B - a.A, b.B - b.A) == 0;
}

pair<T2, T2> alpha(line a, line b) {
	return pair<T2, T2>(cross(b.A - a.A, b.B - b.A),
							cross(a.B - a.A, b.B -
								b.A));
}

bool fcmp(T4 f1t, T4 f1b, T4 f2t, T4 f2b) {
	if (f1b < 0) {
		f1t *= -1;
		f1b *= -1;
	}
	if (f2b < 0) {
		f2t *= -1;
		f2b *= -1;
	}
	return f1t * f2b < f2t * f1b; // check with eps
}

bool check(line a, line b, line c) {
	bool crs = cross(c.B - c.A, a.B - a.A) > 0;
	pair<T2, T2> a1 = alpha(a, b);
	pair<T2, T2> a2 = alpha(a, c);
	bool alp = fcmp(a1.A, a1.B, a2.A, a2.B);
	return (crs ^ alp);
}

bool notin(line a, line b, line c) { // is intersection of a and b in
    ccw direction of c?
	if (parallel(a, b))
		return false;
	if (parallel(a, c))
		return cross(c.B - c.A, a.A - c.A) < 0;
	if (parallel(b, c))
		return cross(c.B - c.A, b.A - c.A) < 0;
	return !(check(a, b, c) && check(b, a, c));
}

void print(vector<line> lines) {
	cerr << "    " << endl;for (int i = 0; i < lines.size();
	    i++)cerr << lines[i].A.X << " " << lines[i].A.Y << " -> "
	    << lines[i].B.X << " " << lines[i].B.Y << endl;cerr << "
	     "


 << endl<< endl;
}

line dq[MAXLINES];
vector<line> half_plane(vector<line> lines) {
	lines.push_back(line(point(INF, -INF), point(INF, INF)));
	lines.push_back(line(point(-INF, INF), point(-INF, -INF)));
	lines.push_back(line(point(-INF, -INF), point(INF, -INF)));
	lines.push_back(line(point(INF, INF), point(-INF, INF)));
	sort(lines.begin(), lines.end(), cmp);
	int ptr = 0;
	for (int i = 0; i < lines.size(); i++)
		if (i > 0 &&
				(lines[i - 1].A < lines[i - 1].B) == (lines[i
				    ].A < lines[i].B) &&
				parallel(lines[i - 1], lines[i]))
				continue;
		else
				lines[ptr++] = lines[i];
	lines.resize(ptr);
	if (lines.size() < 2)
		return lines;
	//print(lines);
	int f = 0, e = 0;
	dq[e++] = lines[0];
	dq[e++] = lines[1];
	for (int i = 2; i < lines.size(); i++) {
		while (f < e - 1 && notin(dq[e - 2], dq[e - 1], lines[
		    i]))
				e--;
		//print(vector<line>(dq + f, dq + e));
		if (e == f + 1) {
			T2 crs = cross(dq[f].B - dq[f].A, lines[i].B -
			    lines[i].A) ;
			if (crs < 0)
				return vector<line>();
			else if (crs == 0 && cross(lines[i].B - lines[
			    i].A, dq[f].B - lines[i].A) < 0)
				return vector<line>();
		}
		while (f < e - 1 && notin(dq[f], dq[f + 1], lines[i]))
			f++;
		dq[e++] = lines[i];
	}
	while (f < e - 1 && notin(dq[e - 2], dq[e - 1], dq[f]))
		e--;
	while (f < e - 1 && notin(dq[f], dq[f + 1], dq[e - 1]))
		f++;
	vector<line> res;
	res.resize(e - f);
	for (int i = f; i < e; i++)
		res[i - f] = dq[i];
	return res;
}


int main() {
	int n;
	cin >> n;
	vector<line> lines;
	for (int i = 0; i < n; i++) {
		int x1, y1, x2, y2;
		cin >> x1 >> y1 >> x2 >> y2;
		lines.push_back(line(point(x1, y1), point(x2, y2)));
	}
```

```
        lines = half_plane(lines);
        cout << lines.size() << endl;
        for (int i = 0; i < lines.size(); i++)
                cout << lines[i].A.X << " " << lines[i].A.Y << " " <<
                    lines[i].B.X << " " << lines[i].B.Y << endl;
}
```

1.10 Minimum Enclosing Circle

```
const int N = 1000*100 + 10;
struct point {
    ll x, y, z;
};
typedef vector<point> circle;
bool ccw(point a, point b, point c) {
    return (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y) >= 0;
}
bool incircle( circle a , point p ) {
    if( sz(a) == 0 ) return false;
    if( sz(a) == 1 )
        return a[0].x == p.x && a[0].y == p.y;
    if( sz(a) == 2 ) {
        point mid = {a[0].x+a[1].x, a[0].y+a[1].y};
        return sq(2*p.x-mid.x) + sq( 2*p.y-mid.y) <= sq(2*a[0].x-mid.x
            ) + sq(2*a[0].y-mid.y);
    }
    if( !ccw(a[0], a[1], a[2]) )
        swap(a[0], a[2]);
    return incircle(a[0],a[1],a[2], p) >= 0;
}
point a[N];
circle solve(int i, circle curr) {
    assert(curr.size() <= 3);
    if( i == 0 )
        return curr;
    circle ret = solve(i-1, curr);
    if( incircle(ret, a[i-1]) )
        return ret;
    curr.pb(a[i-1]);
    return solve(i-1, curr);
}
int n;
void gg(circle c) {
    if( sz(c) == 1 ) {
        cout << ld(a[0].x) << " " << ld(a[0].y) << endl;
        cout << 0.l << endl;
        return;
    }
    if( sz(c) == 2 ) {
        point mid = {c[0].x+c[1].x, c[0].y+c[1].y};
        ld ret = sqrt(sq(2*c[0].x-mid.x) + sq(2*c[0].y-mid.y))/2;
        cout << ld(mid.x) / 2 << " " << ld(mid.y) /2 << endl;
        cout << ret << endl;
    } else {
        lpt a[3];
        for(int i = 0; i < 3; i++)
            a[i] = lpt(c[i].x, c[i].y);
        lpt A = ld(0.5) * (a[0] + a[1]), C = ld(0.5) * (a[1] + a[2]);
        lpt B = A + (a[1] - a[0]) * lpt(0, 1), D = C + (a[2] - a[1]) *
             lpt(0, 1);
        lpt center = intersection( A , B , C , D );
        ld ret = abs(a[0] - center);
        cout << center.real() << " " << center.imag() << endl;
        cout << ret << endl;
    }
}
int main(){
    cin >> n;
    for(int i = 0; i < n; i++) {
        cin >> a[i].x >> a[i].y;
        a[i].z = sq(a[i].x) + sq(a[i].y);
    }
    srand(time(NULL));
    for(int i = 1; i < n; i++)
        swap(a[i], a[rand()%(i+1)]);
    circle ans = solve(n, circle());
    cout << fixed << setprecision(3) ;
    gg(ans);
    return 0;
}
```

1.11 Number of integer points inside polygon

```
S = I + B / 2 - 1
```

1.12 Useful Geo Facts

```
Area of triangle with sides a, b, c: sqrt(S *(S-a)*(S-b)*(S-c)) where
    S = (a+b+c)/2
Area of equilateral triangle: s^2 * sqrt(3) / 4 where is side lenght
Pyramid and cones volume: 1/3 area(base) * height

if p1=(x1, x2), p2=(x2, y2), p3=(x3, y3) are points on circle, the
    center is
x = -((x2^2 - x1^2 + y2^2 - y1^2)*(y3 - y2) - (x2^2 - x3^2 + y2^2 - y3
    ^2)*(y1 - y2)) / (2*(x1 - x2)*(y3 - y2) - 2*(x3 - x2)*(y1 - y2))
y = -((y2^2 - y1^2 + x2^2 - x1^2)*(x3 - x2) - (y2^2 - y3^3 + x2^2 - x3
    ^2)*(x1 - x2)) / (2*(y1 - y2)*(x3 - x2) - 2*(y3 - y2)*(x1 - x2))
```

1.13 Duality and properties

duality of point (a, b) is y = ax - b and duality of line y = ax + b is (a, -b)
Properties:

1. p is on l iff l* is in p*
2. p is in intersection of l1 and l2 iff l1* and l2* lie on p*
3. Duality preserve vertical distance
4. Translating a line in primal to moving vertically in dual
5. Rotating a line in primal to moving a point along a non-vertical line

6. $li \cap lj$ is a vertex of lower envelope $\iff$ (li*, lj*) is an edge of upper hull in dual

2 String

2.1 Suffix Automata

```
const int maxn = 2 e5 + 42; // Maximum amount of states
map < char , int > to [ maxn ]; // Transitions
int link [ maxn ]; // Suffix links
int len [ maxn ]; // Lengthes of largest strings in states
int last = 0; // State corresponding to the whole string
int sz = 1; // Current amount of states
void add_letter ( char c ) { // Adding character to the end
    int p = last ; // State of string s
    last = sz ++; // Create state for string sc
    len [ last ] = len [ p ] + 1;
    for (; to [ p ][ c ] == 0; p = link [ p ]) // (1)
        to [ p ][ c ] = last ; // Jumps which add new suffixes
    if ( to [ p ][ c ] == last ) { // This is the first occurrence of
        c if we are here
        link [ last ] = 0;
        return ;
    }
    int q = to [ p ][ c ];
    if ( len [ q ] == len [ p ] + 1) {
        link [ last ] = q ;
        return ;
    }
    // We split off cl from q here
    int cl = sz ++;
    to [ cl ] = to [ q ]; // (2)
    link [ cl ] = link [ q ];
    len [ cl ] = len [ p ] + 1;
    link [ last ] = link [ q ] = cl ;
    for (; to [ p ][ c ] == q ; p = link [ p ]) // (3)
        to [ p ][ c ] = cl ; // Redirect transitions where needed
}
```

2.2 Suffix Tree

```
#define fpos adla
const int inf = 1e9;
const int maxn = 1e4;
char s[maxn];
map<int, int> to[maxn];
int len[maxn], fpos[maxn], link[maxn];
int node, pos;
int sz = 1, n = 0;
int make_node(int _pos, int _len) {
    fpos[sz] = _pos;
    len [sz] = _len;
    return sz++;
}
void go_edge() {
    while(pos > len[to[node][s[n - pos]]]) {
        node = to[node][s[n - pos]];
        pos -= len[node];
    }
}
void add_letter(int c) {
    s[n++] = c;
    pos++;
    int last = 0;
    while(pos > 0) {
        go_edge();
        int edge = s[n - pos];
        int &v = to[node][edge];
        int t = s[fpos[v] + pos - 1];
        if(v == 0) {
            v = make_node(n - pos, inf);
            link[last] = node;
            last = 0;
        } else if(t == c) {
            link[last] = node;
            return;
        } else {
            int u = make_node(fpos[v], pos - 1);
            to[u][c] = make_node(n - 1, inf);
            to[u][t] = v;
            fpos[v] += pos - 1;
            len [v] -= pos - 1;
            v = u;
            link[last] = u;
            last = u;
        }
        if(node == 0)
            pos--;
        else
            node = link[node];
    }
}
int main() {
    len[0] = inf;
    string s;
    cin >> s;
    int ans = 0;
    for(int i = 0; i < s.size(); i++)
        add_letter(s[i]);
    for(int i = 1; i < sz; i++)
        ans += min((int)s.size() - fpos[i], len[i]);
    cout << ans << "\n";
}
```

2.3 Palindromic Tree

```
int n, last, sz;
void init() {
    s[n++] = -1;
    link[0] = 1;
    len[1] = -1;
    sz = 2;
}
int get_link(int v) {
```

```
    while(s[n - len[v] - 2] != s[n - 1]) v = link[v];
    return v;
}
void add_letter(int c) {
    s[n++] = c;
    last = get_link(last);
    if(!to[last][c]) {
        len [sz] = len[last] + 2;
        link[sz] = to[get_link(link[last])][c];
        to[last][c] = sz++;
    }
    last = to[last][c];
}
```

3 Data structure

3.1 Treap

```
struct item {
    int key, prior;
    item * l, * r;
    item() { }
    item (int key, int prior) : key(key), prior(prior), l(NULL), r(
        NULL) { }
};
typedef item * pitem;
void split (pitem t, int key, pitem & l, pitem & r) {
    if (!t)
        l = r = NULL;
    else if (key < t->key)
        split (t->l, key, l, t->l), r = t;
    else
        split (t->r, key, t->r, r), l = t;
}
void insert (pitem & t, pitem it) {
    if (!t)
        t = it;
    else if (it->prior > t->prior)
        split (t, it->key, it->l, it->r), t = it;
    else
        insert (it->key < t->key ? t->l : t->r, it);
}
void merge (pitem & t, pitem l, pitem r) {
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r), t = l;
    else
        merge (r->l, l, r->l), t = r;
}
void erase (pitem & t, int key) {
    if (t->key == key)
        merge (t, t->l, t->r);
    else
        erase (key < t->key ? t->l : t->r, key);
}
pitem unite (pitem l, pitem r) {
    if (!l || !r) return l ? l : r;
    if (l->prior < r->prior) swap (l, r);
    pitem lt, rt;
    split (r, l->key, lt, rt);
    l->l = unite (l->l, lt);
    l->r = unite (l->r, rt);
    return l;
}
```

3.2 Treap Full

```
typedef struct item * pitem;
struct item {
    int prior, value, cnt;
    bool rev;
    pitem l, r;
};

int cnt (pitem it) {
    return it ? it->cnt : 0;
}

void upd_cnt (pitem it) {
    if (it)
        it->cnt = cnt(it->l) + cnt(it->r) + 1;
}

void push (pitem it) {
    if (it && it->rev) {
        it->rev = false;
        swap (it->l, it->r);
        if (it->l) it->l->rev ^= true;
        if (it->r) it->r->rev ^= true;
    }
}

void merge (pitem & t, pitem l, pitem r) {
    push (l);
    push (r);
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r), t = l;
    else
        merge (r->l, l, r->l), t = r;
    upd_cnt (t);
}

void split (pitem t, pitem & l, pitem & r, int key, int add = 0) {
    if (!t)
        return void( l = r = 0 );
    push (t);
    int cur_key = add + cnt(t->l);
    if (key <= cur_key)
        split (t->l, l, t->l, key, add), r = t;
    else
        split (t->r, t->r, r, key, add + 1 + cnt(t->l)), l = t;
    upd_cnt (t);
}

void reverse (pitem t, int l, int r) {
```

```
    pitem t1, t2, t3;
    split (t, t1, t2, l);
    split (t2, t2, t3, r-l+1);
    t2->rev ^= true;
    merge (t, t1, t2);
    merge (t, t, t3);
}

void output (pitem t) {
    if (!t)  return;
    push (t);
    output (t->l);
    printf ("%d ", t->value);
    output (t->r);
}
```

3.3 Link-cut Tree

```
Node x[N];
struct Node {
    int sz, label; /* size, label */
    Node *p, *pp, *l, *r; /* parent, path-parent, left, right pointers
         */
    Node() { p = pp = l = r = 0; }
};
void update(Node *x) {
    x->sz = 1;
    if(x->l) x->sz += x->l->sz;
    if(x->r) x->sz += x->r->sz;
}
void rotr(Node *x) {
    Node *y, *z;
    y = x->p, z = y->p;
    if((y->l = x->r)) y->l->p = y;
    x->r = y, y->p = x;
    if((x->p = z)) {
        if(y == z->l) z->l = x;
        else z->r = x;
    }
    x->pp = y->pp;
    y->pp = 0;
    update(y);
}
void rotl(Node *x) {
    Node *y, *z;
    y = x->p, z = y->p;
    if((y->r = x->l)) y->r->p = y;
    x->l = y, y->p = x;
    if((x->p = z)) {
        if(y == z->l) z->l = x;
        else z->r = x;
    }
    x->pp = y->pp;
    y->pp = 0;
    update(y);
}
void splay(Node *x) {
    Node *y, *z;
    while(x->p) {
        y = x->p;
        if(y->p == 0) {
            if(x == y->l) rotr(x);
            else rotl(x);
        }
        else {
            z = y->p;
            if(y == z->l) {
                if(x == y->l) rotr(y), rotr(x);
                else rotl(x), rotr(x);
            }
            else {   if(x == y->r) rotl(y), rotl(x);
                else rotr(x), rotl(x);
            }
        }
    }
    update(x);
}
Node *access(Node *x) {
    splay(x);
    if(x->r) {
        x->r->pp = x;
        x->r->p = 0;
        x->r = 0;
        update(x);
    }
    Node *last = x;
    while(x->pp) {
        Node *y = x->pp;
        last = y;
        splay(y);
        if(y->r) {
            y->r->pp = y;
            y->r->p = 0;
        }
        y->r = x;
        x->p = y;
        x->pp = 0;
        update(y);
        splay(x);
    }
    return last;
}
Node *root(Node *x) {
    access(x);
    while(x->l) x = x->l;
    splay(x);
    return x;
}
void cut(Node *x) {
    access(x);
    x->l->p = 0;
    x->l = 0;
    update(x);
}
void link(Node *x, Node *y) {
    access(x);
    access(y);
    x->l = y;
    y->p = x;
    update(x);
}
```

```
Node *lca(Node *x, Node *y) {
    access(x);
    return access(y);
}
int depth(Node *x) {
    access(x);
    return x->sz - 1;
}
void init(int n) {
    for(int i = 0; i < n; i++) {
        x[i].label = i;
        update(&x[i]);
    }
}
```

3.4 Dynimic convex hull

```
const ld is_query = -(1LL << 62);
struct Line {
    ld m, b;
    mutable std::function<const Line *()> succ;
    bool operator<(const Line &rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s = succ();
        if (!s) return 0;
        ld x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // dynamic upper hull +
    max value query
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b) * (z->m - y->m) >= (y->b - z->b) * (y->m
            - x->m);
    }
    void insert_line(ld m, ld b) {
        auto y = insert({m, b});
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) {
            erase(y);
            return;
        }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ld best(ld x) {
        auto l = *lower_bound((Line) {x, is_query});
        return l.m * x + l.b;
    }
};
```

3.5 Heavy-Light Decomposition

```
const int N = 2000*100 + 10;
const int L = 20;
int par[N][L], h[N], fath[N], st[N], en[N], sz[N];
vector<int> c[N]; //Adjacency List
int dsz(int s, int p) {
    sz[s] = 1;
    for(int xt = 0; xt < sz(c[s]); xt++) {
        int x = c[s][xt];
        if( x != p ) {
            sz[s] += dsz( x , s );
            if( sz[x] > sz[c[s][0]] )
                swap( c[s][0], c[s][xt] );
        }
    }
    return sz[s];
}
void dfs(int s, int p) {
    static int ind = 0;
    st[s] = ind++;
    for(int k = 1; k < L; k++)
        par[s][k] = par[par[s][k-1]][k-1];
    for(int xt = 0; xt < sz(c[s]); xt++) {
        int x = c[s][xt];
        if( x == p ) continue;
        fath[x] = x;
        if( xt == 0 ) fath[x] = fath[s];
        h[x] = h[s] + 1;
        par[x][0] = s;
        dfs(x, s);
    }
    en[s] = ind;
}
int n, q;
void upset(int u, int w, int qv) {
    int stL = max( st[w] , st[fath[u]] );
    set( stL, st[u] + 1 , qv , 0, n , 1 );
    if( stL == st[w] ) return;
    upset( par[fath[u]][0] , w , qv );
}
```

4 Graph

4.1 Maximum matching - Edmond's blossom

```
/*
GETS:
n->number of vertices
you should use add_edge(u,v) and
add pair of vertices as edges (vertices are 0..n-1)
(note: please don't add multiple edge)
GIVES:
output of edmonds() is the maximum matching in general graph
match[i] is matched pair of i (-1 if there isn't a matched pair)
O(mn^2)
*/
struct struct_edge{int v;struct_edge* nxt;};
```

```
typedef struct_edge* edge;
const int MAXN=500;
struct Edmonds{
        struct_edge pool[MAXN*MAXN*2];
        edge top=pool,adj[MAXN];
        int n,match[MAXN],qh,qt,q[MAXN],father[MAXN],base[MAXN];
        bool inq[MAXN],inb[MAXN];
        void add_edge(int u,int v){
                top->v=v,top->nxt=adj[u],adj[u]=top++;
                top->v=u,top->nxt=adj[v],adj[v]=top++;
        }
        int LCA(int root,int u,int v){
                static bool inp[MAXN];
                memset(inp,0,sizeof(inp));
                while(1){
                        inp[u=base[u]]=true;
                        if (u==root) break;
                        u=father[match[u]];
                }
                while(1){
                        if (inp[v=base[v]]) return v;
                        else v=father[match[v]];
                }
        }
        void mark_blossom(int lca,int u){
                while (base[u]!=lca){
                        int v=match[u];
                        inb[base[u]]=inb[base[v]]=true;
                        u=father[v];
                        if (base[u]!=lca) father[u]=v;
                }
        }
        void blossom_contraction(int s,int u,int v){
                int lca=LCA(s,u,v);
                memset(inb,0,sizeof(inb));
                mark_blossom(lca,u);
                mark_blossom(lca,v);
                if (base[u]!=lca)
                        father[u]=v;
                if (base[v]!=lca)
                        father[v]=u;
                for (int u=0;u<n;u++)
                        if (inb[base[u]]){
                                base[u]=lca;
                                if (!inq[u])
                                        inq[q[++qt]=u]=true;
                        }
        }
        int find_augmenting_path(int s){
                memset(inq,0,sizeof(inq));
                memset(father,-1,sizeof(father));
                for (int i=0;i<n;i++) base[i]=i;
                inq[q[qh=qt=0]=s]=true;
                while (qh<=qt){
                        int u=q[qh++];
                        for (edge e=adj[u];e;e=e->nxt){
                                int v=e->v;
                                if (base[u]!=base[v] && match[u]!=v){
                                        if (v==s || (match[v]!=-1 &&
                                            father[match[v]]!=-1))
                                                blossom_contraction(s,
                                                    u,v);
                                        else if (father[v]==-1){
                                                father[v]=u;
                                                if (match[v]==-1)
                                                        return v;
                                                else if (!inq[match[v
                                                    ]])
                                                        inq[q[++qt]=
                                                            match[v]]=
                                                            true;
                                        }
                                }
                        }
                }
                return -1;
        }
        int augment_path(int s,int t){
                int u=t,v,w;
                while (u!=-1){
                        v=father[u];
                        w=match[v];
                        match[v]=u;
                        match[u]=v;
                        u=w;
                }
                return t!=-1;
        }
        int edmonds(){
                int matchc=0;
                memset(match,-1,sizeof(match));
                for (int u=0;u<n;u++)
                        if (match[u]==-1)
                                matchc+=augment_path(u,
                                    find_augmenting_path(u));
                return matchc;
        }
};
```

4.2 Biconnected components

```
vector<int> adj[maxn];
bool vis[maxn];
int dep[maxn], par[maxn], lowlink[maxn];
vector<vector<int> > comp;
stack<int> st;
void dfs(int u, int depth = 0, int parent = -1){
        vis[u] = true;
        dep[u] = depth;
        par[u] = parent;
        lowlink[u] = depth;
        st.push(u);
        for (int i = 0; i < adj[u].size(); i++){
                int v = adj[u][i];
                if (!vis[v])
                {
                        dfs(v, depth + 1, u);
                        lowlink[u] = min(lowlink[u], lowlink[v]);
                }
                else
                        lowlink[u] = min(lowlink[u], dep[v]);
```

```
		}
		if (lowlink[u] == dep[u] - 1){
			comp.push_back(vector<int>());
			while (st.top() != u)
			{
				comp.back().push_back(st.top());
				st.pop();
			}
			comp.back().push_back(u);
			st.pop();
			comp.back().push_back(par[u]);
		}
}
void bicon(int n){
		for (int i = 0; i < n; i++)
			if (!vis[i])
				dfs(i);
}
```

4.3 Flow - Dinic

```
int from[MAXE], to[MAXE], cap[MAXE], prv[MAXE], head[MAXN], pt[MAXN], ec;
void addEdge(int u, int v, int uv, int vu = 0){
		from[ec] = u, to[ec] = v, cap[ec] = uv, prv[ec] = head[u], head[u] = ec++;
		from[ec] = v, to[ec] = u, cap[ec] = vu, prv[ec] = head[v], head[v] = ec++;
}
int lv[MAXN], q[MAXN];
bool bfs(int source, int sink){
		memset(lv, 63, sizeof(lv));
		int h = 0, t = 0;
		lv[source] = 0;
		q[t++] = source;
		while (t-h){
			int v = q[h++];
			for (int e = head[v]; ~e; e = prv[e])
				if (cap[e] && lv[v] + 1 < lv[to[e]]){
					lv[to[e]] = lv[v] + 1;
					q[t++] = to[e];
				}
		}
		return lv[sink] < 1e8;
}
int dfs(int v, int sink, int f = 1e9){
		if (v == sink || f == 0)
			return f;
		int ret = 0;
		for (int &e = pt[v]; ~e; e = prv[e])
			if (lv[v]+1 == lv[to[e]]){
				int x = dfs(to[e], sink, min(f, cap[e]));
				cap[e] -= x;
				cap[e^1] += x;
				ret += x;
				f -= x;
				if (!f)
					break;
			}
		return ret;
}
int dinic(int source, int sink){
		int ret = 0;
		while (bfs(source, sink)){
			memcpy(pt, head, sizeof(head));
			ret += dfs(source, sink);
		}
		return ret;
}
```

4.4 Maximum weighted matching - Hungarian

```
const int N = 2002;
const int INF = 1e9;
int hn, weight[N][N];
int x[N], y[N];
int hungarian()  // maximum weighted perfect matching{
		int n = hn;
		int p, q;
		vector<int> fx(n, -INF), fy(n, 0);
		fill(x, x + n, -1);
		fill(y, y + n, -1);
		for (int i = 0; i < n; ++i)
			for (int j = 0; j < n; ++j)
				fx[i] = max(fx[i], weight[i][j]);
		for (int i = 0; i < n; ) {
			vector<int> t(n, -1), s(n+1, i);
			for (p = 0, q = 1; p < q && x[i] < 0; ++p) {
				int k = s[p];
				for (int j = 0; j < n && x[i] < 0; ++j)
					if (fx[k] + fy[j] == weight[k][j] && t[j] < 0) {
						s[q++] = y[j], t[j] = k;
						if (y[j] < 0) // match found!
							for (int p = j; p >= 0; j = p)
								y[j] = k = t[j], p = x[k], x[k] = j;
					}
			}
			if (x[i] < 0) {
				int d = INF;
				for (int k = 0; k < q; ++k)
					for (int j = 0; j < n; ++j)
						if (t[j] < 0) d = min(d, fx [s[k]] + fy[j] - weight[s[k]][j]);
				for (int j = 0; j < n; ++j) fy[j] += (t[j] < 0? 0: d);
				for (int k = 0; k < q; ++k) fx[s[k]] -= d;
			} else ++i;
		}
		int ret = 0;
		for (int i = 0; i < n; ++i) ret += weight[i][x[i]];
		return ret;
}
```

4.5 Ear decomposition

```
1- Find a spanning tree of the given graph and choose a root for the
   tree.
2- Determine, for each edge uv that is not part of the tree, the
   distance between the root and the lowest common ancestor of u and
   v.
3- For each edge uv that is part of the tree, find the corresponding "
   master edge", a non-tree edge wx such that the cycle formed by
   adding wx to the tree passes through uv and such that, among such
   edges, w and x have a lowest common ancestor that is as close to
   the root as possible (with ties broken by edge identifiers).
4- Form an ear for each non-tree edge, consisting of it and the tree
   edges for which it is the master, and order the ears by their
   master edges' distance from the root (with the same tie-breaking
   rule).
```

4.6 Stoer-Wagner min cut $O(n^3)$

```
const int N = -1, MAXW = -1;
int g[N][N], v[N], w[N], na[N];
bool a[N];
int minCut( int n ) // initialize g[][] before calling!{
    for( int i = 0; i < n; i++ ) v[i] = i;
    int best = MAXW * n * n;
    while( n > 1 ){
        // initialize the set A and vertex weights
        a[v[0]] = true;
        for( int i = 1; i < n; i++ ){
            a[v[i]] = false;
            na[i - 1] = i;
            w[i] = g[v[0]][v[i]];
        }
        // add the other vertices
        int prev = v[0];
        for( int i = 1; i < n; i++ ){
            // find the most tightly connected non-A vertex
            int zj = -1;
            for( int j = 1; j < n; j++ )
                if( !a[v[j]] && ( zj < 0 || w[j] > w[zj] ) )
                    zj = j;
            // add it to A
            a[v[zj]] = true;
            // last vertex?
            if( i == n - 1 ){
                // remember the cut weight
                best = min(best, w[zj]);

                // merge prev and v[zj]
                for( int j = 0; j < n; j++ )
                    g[v[j]][prev] = g[prev][v[j]] += g[v[zj]][v[j]];
                v[zj] = v[--n];
                break;
            }
            prev = v[zj];
            // update the weights of its neighbors
            for( int j = 1; j < n; j++ ) if( !a[v[j]] )
                w[j] += g[v[zj]][v[j]];
        }
    }
    return best;
}
```

4.7 Directed minimum spanning tree $O(m \log n)$

```
/*
	GETS:
		call make_graph(n) at first
		you should use add_edge(u,v,w) and
		add pair of vertices as edges (vertices are 0..n-1)
	GIVES:
		output of dmst(v) is the minimum arborescence with
		    root v in directed graph
		(INF if it hasn't a spanning arborescence with root v)
	O(mlogn)
*/
const int INF = 2e7;
struct MinimumAborescense{
	struct edge {
		int src, dst, weight;
	};
	struct union_find {
		vector<int> p;
		union_find(int n) : p(n, -1) { };
		bool unite(int u, int v) {
			if ((u = root(u)) == (v = root(v))) return
			    false;
			if (p[u] > p[v]) swap(u, v);
			p[u] += p[v]; p[v] = u;
			return true;
		}
		bool find(int u, int v) { return root(u) == root(v); }
		int root(int u) { return p[u] < 0 ? u : p[u] = root(p[
		    u]); }
		int size(int u) { return -p[root(u)]; }
	};
	struct skew_heap {
		struct node {
			node *ch[2];
			edge key;
			int delta;
		} *root;
		skew_heap() : root(0) { }
		void propagate(node *a) {
			a->key.weight += a->delta;
			if (a->ch[0]) a->ch[0]->delta += a->delta;
			if (a->ch[1]) a->ch[1]->delta += a->delta;
			a->delta = 0;
		}
		node *merge(node *a, node *b) {
			if (!a || !b) return a ? a : b;
			propagate(a); propagate(b);
			if (a->key.weight > b->key.weight) swap(a, b);
			a->ch[1] = merge(b, a->ch[1]);
			swap(a->ch[0], a->ch[1]);
			return a;
		}
		void push(edge key) {
			node *n = new node();
```

```
            n->ch[0] = n->ch[1] = 0;
            n->key = key; n->delta = 0;
            root = merge(root, n);
        }
        void pop() {
            propagate(root);
            node *temp = root;
            root = merge(root->ch[0], root->ch[1]);
        }
        edge top() {
            propagate(root);
            return root->key;
        }
        bool empty() {
            return !root;
        }
        void add(int delta) {
            root->delta += delta;
        }
        void merge(skew_heap x) {
            root = merge(root, x.root);
        }
};
vector<edge> edges;
void add_edge(int src, int dst, int weight) {
        edges.push_back({src, dst, weight});
}
int n;
void make_graph(int _n) {
        n = _n;
        edges.clear();
}
int dmst(int r) {
        union_find uf(n);
        vector<skew_heap> heap(n);
        for (auto e: edges)
                heap[e.dst].push(e);
        double score = 0;
        vector<int> seen(n, -1);
        seen[r] = r;
        for (int s = 0; s < n; ++s) {
                vector<int> path;
                for (int u = s; seen[u] < 0;) {
                        path.push_back(u);
                        seen[u] = s;
                        if (heap[u].empty()) return INF;
                        edge min_e = heap[u].top();
                        score += min_e.weight;
                        heap[u].add(-min_e.weight);
                        heap[u].pop();
                        int v = uf.root(min_e.src);
                        if (seen[v] == s) {
                                skew_heap new_heap;
                                while (1) {
                                        int w = path.back();
                                        path.pop_back();
                                        new_heap.merge(heap[w
                                            ]);
                                        if (!uf.unite(v, w))
                                            break;
                                }
                                heap[uf.root(v)] = new_heap;
                                seen[uf.root(v)] = -1;
                        }
                        u = uf.root(v);
                }
        }
        return score;
    }
};
```

4.8 Directed minimum spanning tree $O(nm)$

```
/*
        GETS:
                call make_graph(n) at first
                you should use add_edge(u,v,w) and
                add pair of vertices as edges (vertices are 0..n-1)
        GIVES:
                output of dmst(v) is the minimum arborescence with
                    root v in directed graph
                (-1 if it hasn't a spanning arborescence with root v)
        O(mn)
*/
const int INF = 2e7;
struct MinimumAborescense{
        int n;
        struct edge {
                int src, dst;
                int weight;
        };
        vector<edge> edges;
        void make_graph(int _n) {
                n=_n;
                edges.clear();
        }
        void add_edge(int u, int v, int w) {
                edges.push_back({u, v, w});
        }
        int dmst(int r) {
                int N = n;
                for (int res = 0; ;) {
                        vector<edge> in(N, {-1,-1,(int)INF});
                        vector<int> C(N, -1);
                        for (auto e: edges)
                                if (in[e.dst].weight > e.weight)
                                        in[e.dst] = e;
                        in[r] = {r, r, 0};

                        for (int u = 0; u < N; ++u) { // no comming
                            edge ==> no aborescense
                                if (in[u].src < 0) return -1;
                                res += in[u].weight;
                        }
                        vector<int> mark(N, -1); // contract cycles
                        int index = 0;
                        for (int i = 0; i < N; ++i) {
                                if (mark[i] != -1) continue;
                                int u = i;
                                while (mark[u] == -1) {
                                        mark[u] = i;
```

```
                        u = in[u].src;
                    }
                    if (mark[u] != i || u == r) continue;
                    for (int v = in[u].src; u != v; v = in
                        [v].src) C[v] = index;
                    C[u] = index++;
                }
                if (index == 0) return res; // found
                    arborescence
                for (int i = 0; i < N; ++i) // contract
                    if (C[i] == -1) C[i] = index++;

                vector<edge> next;
                for (auto &e: edges)
                    if (C[e.src] != C[e.dst] && C[e.dst]
                        != C[r])
                        next.push_back({C[e.src], C[e.
                            dst], e.weight - in[e.dst
                            ].weight});
                edges.swap(next);
                N = index; r = C[r];
            }
        }
};
```

4.9 Dominator tree

```
struct DominatorTree{
    vector<int> adj[MAXN], radj[MAXN], tree[MAXN], bucket[MAXN];
        // SET MAXIMUM NUMBER OF NODES
    int sdom[MAXN], par[MAXN], idom[MAXN], dsu[MAXN], label[MAXN];
    int arr[MAXN], rev[MAXN], cnt;
    void clear(){
        for (int i = 0; i < MAXN; i++){
            adj[i].clear();
            radj[i].clear();
            tree[i].clear();
            sdom[i] = idom[i] = dsu[i] = label[i] = i;
            arr[i] = -1;
        }
        cnt = 0;
    }
    void add_edge(int u, int v){
        adj[u].push_back(v);
    }
    void dfs(int v){
        arr[v] = cnt;
        rev[cnt] = v;
        cnt++;
        for (int i = 0; i < adj[v].size(); i++){
            int u = adj[v][i];
            if (arr[u] == -1){
                dfs(u);
                par[arr[u]] = arr[v];
            }
            radj[arr[u]].push_back(arr[v]);
        }
    }
    int find(int v, int x = 0){
        if (dsu[v] == v)
            return (x ? -1 : v);
        int u = find(dsu[v], x + 1);
        if (u < 0)
            return v;
        if (sdom[label[dsu[v]]] < sdom[label[v]])
            label[v] = label[dsu[v]];
        dsu[v] = u;
        return (x ? u : label[v]);
    }
    void merge(int u, int v){
        dsu[v] = u;
    }
    void build(int root){
        dfs(root);
        int n = cnt;
        for (int v = n - 1; v >= 0; v--){
            for (int i = 0; i < radj[v].size(); i++){
                int u = radj[v][i];
                sdom[v] = min(sdom[v], sdom[find(u)]);
            }
            if (v > 0)
                bucket[sdom[v]].push_back(v);
            for (int i = 0; i < bucket[v].size(); i++){
                int u = bucket[v][i];
                int w = find(u);
                if (sdom[u] == sdom[w])
                    idom[u] = sdom[u];
                else
                    idom[u] = w;
            }
            if (v > 0)
                merge(par[v], v);
        }
        for (int v = 1; v < n; v++){
            if (idom[v] != sdom[v])
                idom[v] = idom[idom[v]];
            tree[rev[v]].push_back(rev[idom[v]]);
            tree[rev[idom[v]]].push_back(rev[v]);
        }
    }
    DominatorTree(){
        clear();
    }
};
```

5 Combinatorics

5.1 LP simplex

```
// Two-phase simplex algorithm for solving linear programs of the form
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
// OUTPUT: value of the optimal solution (infinity if unbounded
```

```
//          above, nan if infeasible)
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).
typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;
const DOUBLE EPS = 1e-9;
struct LPSolver {
  int m, n;
  VI B, N;
  VVD D;
  LPSolver(const VVD &A, const VD &b, const VD &c) :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] =
        A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n +
         1] = b[i]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m + 1][n] = 1;
  }
  void Pivot(int r, int s) {
    double inv = 1.0 / D[r][s];
    for (int i = 0; i < m + 2; i++) if (i != r)
      for (int j = 0; j < n + 2; j++) if (j != s)
        D[i][j] -= D[r][j] * D[i][s] * inv;
    for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
    for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }
  bool Simplex(int phase) {
    int x = phase == 1 ? m + 1 : m;
    while (true) {
      int s = -1;
      for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j]
             < N[s]) s = j;
      }
      if (D[x][s] > -EPS) return true;
      int r = -1;
      for (int i = 0; i < m; i++) {
        if (D[i][s] < EPS) continue;
        if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s]
             ||
          (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] <
               B[r]) r = i;
      }
      if (r == -1) return false;
      Pivot(r, s);
    }
  }
  DOUBLE Solve(VD &x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
    if (D[r][n + 1] < -EPS) {
      Pivot(r, n);
      if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -
          numeric_limits<DOUBLE>::infinity();
      for (int i = 0; i < m; i++) if (B[i] == -1) {
        int s = -1;
        for (int j = 0; j <= n; j++)
          if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[
              j] < N[s]) s = j;
        Pivot(i, s);
      }
    }
    if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
    return D[m][n + 1];
  }
};

int main() {
  const int m = 4;
  const int n = 3;
  DOUBLE _A[m][n] = {
    { 6, -1, 0 },
    { -1, -5, 0 },
    { 1, 5, 1 },
    { -1, -5, -1 }
  };
  DOUBLE _b[m] = { 10, -4, 5, -5 };
  DOUBLE _c[n] = { 1, -1, 0 };
  VVD A(m);
  VD b(_b, _b + m);
  VD c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);
  LPSolver solver(A, b, c);
  VD x;
  DOUBLE value = solver.Solve(x);
  cerr << "VALUE: " << value << endl; // VALUE: 1.29032
  cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
  for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
  cerr << endl;
  return 0;
}
```

5.2 LP for game theory

```
ld solve(int n, int m){
    VVD A;
    VD B;
    VD C;
    for (int j = 0; j < m; j++){
        VD v;
        for (int i = 0; i < n; i++)
            v.pb(-mat[i][j]);
        v.pb(1);
        A.pb(v);
    }
    VD v;
    for (int i = 0; i < n; i++)
        v.pb(1);
    v.pb(0);
    A.pb(v);
    v.clear();
    for (int i = 0; i < n; i++)
        v.pb(-1);
```

```
    v.pb(0);
    A.pb(v);
    for (int i = 0; i < m; i++)
        B.pb(0);
    B.pb(1);
    B.pb(-1);
    for (int i = 0; i < n; i++)
        C.pb(0);
    C.pb(1);
    LPSolver solver(A, B, C);
    VD x;
    ld res = solver.Solve(x);
    return res;
}
```

5.3 FFT

```
const int LG = 20; // IF YOU WANT TO CONVOLVE TWO ARRAYS OF LENGTH N
    AND M CHOOSE LG IN SUCH A WAY THAT 2^LG > n + m
const int MAX = 1 << LG;
struct point{
        double real, imag;
        point(double _real = 0.0, double _imag = 0.0){
                real = _real;
                imag = _imag;
        }
};
point operator + (point a, point b){
        return point(a.real + b.real, a.imag + b.imag);
}
point operator - (point a, point b){
        return point(a.real - b.real, a.imag - b.imag);
}
point operator * (point a, point b){
        return point(a.real * b.real - a.imag * b.imag, a.real * b.
            imag + a.imag * b.real);
}

void fft(point *a, bool inv){
        for (int mask = 0; mask < MAX; mask++){
                int rev = 0;
                for (int i = 0; i < LG; i++)
                        if ((1 << i) & mask)
                                rev |= (1 << (LG - 1 - i));
                if (mask < rev)
                        swap(a[mask], a[rev]);
        }
        for (int len = 2; len <= MAX; len *= 2){
                double ang = 2.0 * M_PI / len;
                if (inv)
                        ang *= -1.0;
                point wn(cos(ang), sin(ang));
                for (int i = 0; i < MAX; i += len){
                        point w(1.0, 0.0);
                        for (int j = 0; j < len / 2; j++){
                                point t1 = a[i + j] + w * a[i + j +
                                    len / 2];
                                point t2 = a[i + j] - w * a[i + j +
                                    len / 2];
                                a[i + j] = t1;
                                a[i + j + len / 2] = t2;
                                w = w * wn;
                        }
                }
        }
        if (inv)
                for (int i = 0; i < MAX; i++){
                        a[i].real /= MAX;
                        a[i].imag /= MAX;
                }
}
```

5.4 NTT

```
const int MOD = 998244353;
const int LG = 16; // IF YOU WANT TO CONVOLVE TWO ARRAYS OF LENGTH N
    AND M CHOOSE LG IN SUCH A WAY THAT 2^LG > n + m
const int MAX = (1 << LG);
const int ROOT = 44759; // ENSURE THAT ROOT^2^(LG - 1) = MOD - 1
int bpow(int a, int b){
        int ans = 1;
        while (b){
                if (b & 1)
                        ans = 1LL * ans * a % MOD;
                b >>= 1;
                a = 1LL * a * a % MOD;
        }
        return ans;
}
void ntt(int *a, bool inv){
        for (int mask = 0; mask < MAX; mask++){
                int rev = 0;
                for (int i = 0; i < LG; i++)
                        if ((1 << i) & mask)
                                rev |= (1 << (LG - 1 - i));
                if (mask < rev)
                        swap(a[mask], a[rev]);
        }
        for (int len = 2; len <= MAX; len *= 2){
                int wn = bpow(ROOT, MAX / len);
                if (inv)
                        wn = bpow(wn, MOD - 2);
                for (int i = 0; i < MAX; i += len){
                        int w = 1;
                        for (int j = 0; j < len / 2; j++){
                                int l = a[i + j];
                                int r = 1LL * w * a[i + j + len / 2] %
                                    MOD;
                                a[i + j] = (l + r);
                                a[i + j + len / 2] = l - r + MOD;
                                if (a[i + j] >= MOD)
                                        a[i + j] -= MOD;
                                if (a[i + j + len / 2] >= MOD)
                                        a[i + j + len / 2] -= MOD;
                                w = 1LL * w * wn % MOD;
                        }
                }
        }
        if (inv){
                int x = bpow(MAX, MOD - 2);
```

```
        for (int i = 0; i < MAX; i++)
            a[i] = 1LL * a[i] * x % MOD;
    }
}
```

5.5 Base Vectors in Z2

```
struct Base{
    ll a[B] = {};
    ll eliminate(ll x){
        for(int i=B-1; i>=0; --i) if(x >> i & 1) x ^= a[i];
        return x;
    }
    void add(ll x){
        x = eliminate(x);
        for(int i=B-1; i>=0; --i) if(x >> i & 1)
        {
            a[i] = x;
                for(int j = i - 1; j >= 0; j--) if(a[j] >> i &
                    1) a[j] ^= x;
            return;
        }
    }
    int size(){
        int cnt = 0;
        for(int i=0; i<B; ++i) if(a[i]) ++cnt;
        return cnt;
    }
};
```

5.6 Gaussian Elimination

```
const int N = 505, MOD = 1e9 + 7;
typedef vector <ll> vec;
ll pw(ll a, ll b) {
    if(!b)
        return 1;
    ll x = pw(a, b/2);
    return x * x % MOD * (b % 2 ? a : 1) % MOD;
}
ll inv(ll x) { return pw(x, MOD - 2); }
bool solve() {
    int n = in();
    vector <vec> matrix(n);
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++) {
            matrix[i].push_back((in() % MOD + MOD) % MOD);
        }
    ll res = 1;
    for(int i = 0; i < n; i++) {
        int ind = -1;
        for(int row = i; row < n; row++)
            if(matrix[row][i])
                ind = row;
        if(ind == -1) {
            res = 0;
            break;
        }
        if(i != ind)
            res = (MOD - res)%MOD;
        matrix[i].swap(matrix[ind]);
        res = res * matrix[i][i] % MOD;
        ll inverse = inv(matrix[i][i]);
        for(int row = i + 1; row < n; row++) {
            ll z = matrix[row][i] * inverse % MOD;
            for(int j = 0; j < n; j++)
                matrix[row][j] = (matrix[row][j] % MOD
                    - matrix[i][j]*z % MOD + MOD) %
                    MOD;
        }
    }
    cout << res << endl;
}
```

5.7 Stirling 1

```
const int mod = 998244353;
const int root = 15311432;
const int root_1 = 469870224;
const int root_pw = 1 << 23;
const int N = 400004;
vector<int> v[N];
ll modInv(ll a, ll mod = mod){
    ll x0 = 0, x1 = 1, r0 = mod, r1 = a;
    while(r1){
        ll q = r0 / r1;
        x0 -= q * x1; swap(x0, x1);
        r0 -= q * r1; swap(r0, r1);
    }
    return x0 < 0 ? x0 + mod : x0;
}
void fft (vector<int> &a, bool inv) {
    int n = (int) a.size();
    for (int i=1, j=0; i<n; ++i) {
        int bit = n >> 1;
        for (; j>=bit; bit>>=1)
            j -= bit;
        j += bit;
        if (i < j)
            swap (a[i], a[j]);
    }
    for (int len=2; len<=n; len<<=1) {
        int wlen = inv ? root_1 : root;
        for (int i=len; i<root_pw; i<<=1)
            wlen = int (wlen * 1ll * wlen % mod);
        for (int i=0; i<n; i+=len) {
            int w = 1;
            for (int j=0; j<len/2; ++j) {
                int u = a[i+j],  v = int (a[i+j+len/2]
                    * 1ll * w % mod);
                a[i+j] = u+v < mod ? u+v : u+v-mod;
                a[i+j+len/2] = u-v >= 0 ? u-v : u-v+
                    mod;
                w = int (w * 1ll * wlen % mod);
            }
        }
    }
    if(inv) {
```

```
                int nrev = modInv(n, mod);
                for (int i=0; i<n; ++i)
                        a[i] = int (a[i] * 1ll * nrev % mod);
        }
}
void pro(const vector<int> &a, const vector<int> &b, vector<int> &res)
     {
        vector<int> fa(a.begin(), a.end()),  fb(b.begin(), b.end());
        int n = 1;
        while (n < (int) max(a.size(), b.size()))  n <<= 1;
        n <<= 1;
        fa.resize (n),  fb.resize (n);

        fft(fa, false), fft (fb, false);
        for (int i = 0; i < n; ++i)
                fa[i] = 1LL * fa[i] * fb[i] % mod;
        fft (fa, true);
        res = fa;
}
int S(int n, int r) {
        int nn = 1;
        while(nn < n) nn <<= 1;
        for(int i = 0; i < n; ++i) {
                v[i].push_back(i);
                v[i].push_back(1);
        }
        for(int i = n; i < nn; ++i) {
                v[i].push_back(1);
        }
        for(int j = nn; j > 1; j >>= 1){
                int hn = j >> 1;
                for(int i = 0; i < hn; ++i){
                        pro(v[i], v[i + hn], v[i]);
                }
        }
        return v[0][r];
}

int fac[N], ifac[N], inv[N];

void prencr(){
        fac[0] = ifac[0] = inv[1] = 1;
        for(int i = 2; i < N; ++i)
                inv[i] = mod - 1LL * (mod / i) * inv[mod % i] % mod;
        for(int i = 1; i < N; ++i){
                fac[i] = 1LL * i * fac[i - 1] % mod;
                ifac[i] = 1LL * inv[i] * ifac[i - 1] % mod;
        }
}

int C(int n, int r){
        return (r >= 0 && n >= r) ? (1LL * fac[n] * ifac[n - r] % mod
            * ifac[r] % mod) : 0;
}

int main(){
        prencr();
        int n, p, q;
        cin >> n >> p >> q;
        ll ans = C(p + q - 2, p - 1);
        ans *= S(n - 1, p + q - 2);
        ans %= mod;
        cout << ans;
}
```

5.8 Chinese remainder

```
long long GCD(long long a, long long b) { return (b == 0) ? a : GCD(b,
     a % b); }
inline long long LCM(long long a, long long b) { return a / GCD(a, b)
     * b; }
inline long long normalize(long long x, long long mod) { x %= mod; if
     (x < 0) x += mod; return x; }
struct GCD_type { long long x, y, d; };
GCD_type ex_GCD(long long a, long long b){
    if (b == 0) return {1, 0, a};
    GCD_type pom = ex_GCD(b, a % b);
    return {pom.y, pom.x - a / b * pom.y, pom.d};
}
int testCases;
int t;
long long r[N], n[N], ans, lcm;
int main(){
    cin >> t;
    for(int i = 1; i <= t; i++) cin >> r[i] >> n[i], normalize(r[i], n
        [i]);
    ans = r[1];
    lcm = n[1];
    for(int i = 2; i <= t; i++){
        auto pom = ex_GCD(lcm, n[i]);
        int x1 = pom.x;
        int d = pom.d;
        if((r[i] - ans) % d != 0) return cerr << "No solutions" <<
            endl, 0;
        ans = normalize(ans + x1 * (r[i] - ans) / d % (n[i] / d) * lcm
            , lcm * n[i] / d);
        lcm = LCM(lcm, n[i]); // you can save time by replacing above
            lcm * n[i] /d by lcm = lcm * n[i] / d
    }
    cout << ans << " " << lcm << endl;
    return 0;
}
```

5.9 Stirling 2

$$\left\{ \begin{matrix} \text{n} \\ \text{k} \end{matrix} \right\} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

5.10 Popular LP

BellmanFord:

maximize X_n

$X_1 = 0$

and for eache edge ($v- > u$ and weight w):

$X_u - X_v \leq w$

Flow:

maximize Σf_{out} (where *out* is output edges of vertex 1)
for each vertex (except 1 and n):
$\Sigma f_{in} - \Sigma f_{out} = 0$ (where *in* is input edges of v and *out* is output edges of v)

Dijkstra(IP):

minimize $\Sigma z_i * w_i$
for eache edge ($v- > u$ and weight w):
$0 \leq z_i \leq 1$
and for each ST-cut which vertex 1 is in S and vertex n is in T:
$\Sigma z_e \geq 1$ (for each edge e from S to T)

5.11 Duality of LP

primal: Maximize $c^T x$ subject to $Ax \leq b, x \geq 0$
dual: Minimize $b^T y$ subject to $A^T y \geq c, y \geq 0$

5.12 Extended catalan

number of ways for going from 0 to A with k moves without going to -B:

$$\binom{k}{\frac{A+k}{2}} - \binom{k}{\frac{2B+A+k}{2}}$$

5.13 Find polynomial from it's points

$P(x) = \sum_{i=1}^{n} y_i \prod_{j=1,j\neq i}^{n} \frac{x-x_j}{x_i-x_j}$

6 Constants

6.1 Number of primes

```
30: 10
60: 17
100: 25
1000: 168
10000: 1229
100000: 9592
1000000: 78498
10000000: 664579
```

6.2 Factorials

```
1: 1
2: 2
3: 6
4: 24
5: 120
6: 720
7: 5040
8: 40320
9: 362880
10: 3628800
11: 39916800
12: 479001600
13: 6227020800
14: 87178291200
15: 1307674368000
```

6.3 Powers of 3

```
1: 3
2: 9
3: 27
4: 81
5: 243
6: 729
7: 2187
8: 6561
9: 19683
10: 59049
11: 177147
12: 531441
13: 1594323
14: 4782969
15: 14348907
16: 43046721
17: 129140163
18: 387420489
19: 1162261467
20: 3486784401
```

6.4 C(2n,n)

```
1: 2
2: 6
3: 20
4: 70
5: 252
6: 924
7: 3432
8: 12870
9: 48620
10: 184756
11: 705432
12: 2704156
13: 10400600
```

```
14: 40116600
15: 155117520
```

6.5 Most divisor

```
<= 1e2: 60 with 12 divisors
<= 1e3: 840 with 32 divisors
<= 1e4: 7560 with 64 divisors
<= 1e5: 83160 with 128 divisors
<= 1e6: 720720 with 240 divisors
<= 1e7: 8648640 with 448 divisors
<= 1e8: 73513440 with 768 divisors
<= 1e9: 735134400 with 1344 divisors
<= 1e10: 6983776800 with 2304 divisors
<= 1e11: 97772875200 with 4032 divisors
<= 1e12: 963761198400 with 6720 divisors
<= 1e13: 9316358251200 with 10752 divisors
<= 1e14: 97821761637600 with 17280 divisors
<= 1e15: 866421317361600 with 26880 divisors
<= 1e16: 8086598962041600 with 41472 divisors
<= 1e17: 74801040398884800 with 64512 divisors
<= 1e18: 897612484786617600 with 103680 divisors
```

Useful formulas

$\binom{n}{k} = \frac{n!}{k!(n-k)!}$ — number of ways to choose k objects out of n

$\binom{n+k-1}{k-1}$ — number of ways to choose k objects out of n with repetitions

$\left[{n \atop m}\right]$ — Stirling numbers of the first kind; number of permutations of n elements with k cycles

$\left[{n+1 \atop m}\right] = n\left[{n \atop m}\right] + \left[{n \atop m-1}\right]$

$(x)_n = x(x-1)\dots x-n+1 = \sum\limits_{k=0}^{n}(-1)^{n-k}\left[{n \atop k}\right]x^k$

$\left\{{n \atop m}\right\}$ — Stirling numbers of the second kind; number of partitions of set $1,\dots,n$ into k disjoint subsets.

$\left\{{n+1 \atop m}\right\} = k\left\{{n \atop k}\right\} + \left\{{n \atop k-1}\right\}$

$\sum\limits_{k=0}^{n}\left\{{n \atop k}\right\}(x)_k = x^n$

$C_n = \frac{1}{n+1}\binom{2n}{n}$ — Catalan numbers

$C(x) = \frac{1-\sqrt{1-4x}}{2x}$

Binomial transform

If $a_n = \sum\limits_{k=0}^{n}\binom{n}{k}b_k$, then $b_n = \sum\limits_{k=0}^{n}(-1)^{n-k}\binom{n}{k}a_k$

- $a = (1, x, x^2, \dots), b = (1, (x+1), (x+1)^2, \dots)$
- $a_i = i^k, b_i = \left\{{n \atop i}\right\}i!$

Burnside's lemma

Let G be a group of *action* on set X (Ex.: cyclic shifts of array, rotations and symmetries of $n \times n$ matrix, ...)

Call two objects x and y *equivalent* if there is an action f that transforms x to y: $f(x) = y$.

The number of equivalence classes then can be calculated as follows: $C = \frac{1}{|G|}\sum\limits_{f \in G}|X^f|$, where X^f is the set of *fixed points* of f: $X^f = \{x | f(x) = x\}$

Generating functions

Ordinary generating function (o.g.f.) for sequence $a_0, a_1, \dots, a_n, \dots$ is $A(x) = \sum\limits_{n=0}^{\infty} a_i x^i$

Exponential generating function (e.g.f.) for sequence $a_0, a_1, \dots, a_n, \dots$ is $A(x) = \sum\limits_{n=0}^{\infty} a_i x^i$

$B(x) = A'(x), b_{n-1} = n \cdot a_n$

$c_n = \sum\limits_{k=0}^{n} a_k b_{n-k}$ (o.g.f. convolution)

$c_n = \sum\limits_{k=0}^{n}\binom{n}{k}a_k b_{n-k}$ (e.g.f. convolution, compute with FFT using $\widetilde{a_n} = \frac{a_n}{n!}$)

General linear recurrences

If $a_n = \sum\limits_{k=1}^{n} b_k a_{n-k}$, then $A(x) = \frac{a_0}{1-B(x)}$. We also can compute all a_n with Divide-and-Conquer algorithm in $O(n \log^2 n)$.

Inverse polynomial modulo x^l

Given $A(x)$, find $B(x)$ such that $A(x)B(x) = 1 + x^l \cdot Q(x)$ for some $Q(x)$

1. Start with $B_0(x) = \frac{1}{a_0}$
2. Double the length of $B(x)$: $B_{k+1}(x) = (-B_k(x)^2 A(x) + 2B_k(x)) \mod x^{2^{k+1}}$

Fast subset convolution

Given array a_i of size 2^k, calculate $b_i = \sum\limits_{j \& i = i} b_j$

```
for b = 0..k-1
  for i = 0..2^k-1
    if (i & (1 << b)) != 0:
      a[i + (1 << b)] += a[i]
```

Hadamard transform

Treat array a of size 2^k as k-dimentional array of size $2 \times 2 \times \dots \times 2$, calculate FFT of that array:

```
for b = 0..k-1
  for i = 0..2^k-1
    if (i & (1 << b)) != 0:
      u = a[i], v = a[i + (1 << b)]
      a[i] = u + v
      a[i + (1 << b)] = u - v
```