

1 Data_Structure

1.1 hull_dynamic

```

1 const ll is_query = -(1LL<<62);
2 struct Line {
3     ll m, b;
4     mutable function<const Line*> succ;
5     bool operator<(const Line& rhs) const {
6         if (rhs.b != is_query) return m < rhs.m;
7         const Line* s = succ();
8         if (!s) return 0;
9         ll x = rhs.m;
10        return b - s->b < (s->m - m) * x;
11    }
12 };
13 // Upper envelope, erase cannot be done.
14 // Even if you do erase, the popped lines
15 // are gone, it won't be a correct hull.
16 struct HullDynamic : public multiset<Line> {
17     bool bad(iterator y) {
18         auto z = next(y);
19         if (y == begin()) {
20             if (z == end()) return 0;
21             return y->m == z->m && y->b <= z->b;
22         }
23         auto x = prev(y);
24         if (z == end()) return y->m == x->m && y
25             ->b <= x->b;
26         return 1.0 * (x->b - y->b)*(z->m - y->m)
27             >= 1.0 * (y->b - z->b)*(y->m - x->m)
28             );
29     }
30     void insert_line(ll m, ll b) {
31         auto y = insert({ m, b });
32         y->succ = [=] { return next(y) == end()
33             ? 0 : &*next(y); };
34         if (bad(y)) { erase(y); return; }
35         while (next(y) != end() && bad(next(y)))
36             erase(next(y));
37         while (y != begin() && bad(prev(y)))
38             erase(prev(y));
39     }
40     ll eval(ll x) {
41         auto l = *lower_bound((Line) { x,
42             is_query });
43         return l.m * x + l.b;
44     }
45 };

```

1.2 persistent_treap

```

1 // Once persistent, every op must not change
2 // tree struct.
3 // Also don't free them if any version may
4 // be referenced later.
5 Treap* merge(Treap *a, Treap *b) { // When
6     new, also copy priority
7     if (!a || !b) return a ? (new Treap(a)) :
8         (new Treap(b));

```

```

5     Treap *t;
6     if (a->pri > b->pri) {
7         t = new Treap(a);
8         t->r = merge(t->r, b);
9         pull(t);
10        return t;
11    }
12    else {
13        t = new Treap(b);
14        t->l = merge(a, t->l);
15        pull(t);
16        return t;
17    }
18 }
19 void split(Treap *t, int k, Treap *&a, Treap
20     *&b) {
21     // First k numbers <-> in *a
22     if (!t) { a = b = NULL; return; }
23     t = new Treap(t);
24     if (Size(t->l) < k) {
25         a = t;
26         split(t->r, k - Size(t->l) - 1, a->r, b)
27         ;
28         pull(a);
29     }
30     else {
31         b = t;
32         split(t->l, k, a, b->l);
33         pull(b);
34     }
35 }

```

1.3 Treap

```

1 struct Treap{
2     int sz, val, pri, tag;
3     Treap *l, *r;
4     Treap( int _val ){
5         val = _val; sz = 1;
6         pri = rand(); l = r = NULL; tag = 0;
7     }
8 };
9 void push( Treap * a ){
10    if( a->tag ){
11        Treap *swp = a->l; a->l = a->r; a->r =
12            swp;
13        int swp2;
14        if( a->l ) a->l->tag ^= 1;
15        if( a->r ) a->r->tag ^= 1;
16        a->tag = 0;
17    }
18 }
19 int Size( Treap * a ){ return a ? a->sz : 0;
20 }
21 void pull( Treap * a ){
22     a->sz = Size( a->l ) + Size( a->r ) + 1;
23 }
24 Treap* merge( Treap *a, Treap *b ){
25     if( !a || !b ) return a ? a : b;
26     if( a->pri > b->pri ){
27         push( a );
28         a->r = merge( a->r, b );
29         pull( a );

```

```

28     return a;
29 }else{
30     push( b );
31     b->l = merge( a, b->l );
32     pull( b );
33     return b;
34 }
35 }
36 void split( Treap *t, int k, Treap*&a,
37     Treap*&b ){
38     // First k elements <-> in *a
39     if( !t ){ a = b = NULL; return; }
40     push( t );
41     if( Size( t->l ) + 1 <= k ){
42         a = t;
43         split( t->r, k - Size( t->l ) - 1, a->
44             r, b );
45         pull( a );
46     }else{
47         b = t;
48         split( t->l, k, a, b->l );
49         pull( b );
50     }
51 void split2(Treap *t, int k, Treap *&a,
52     Treap *&b ) {
53     // key<k <-> in *a, when used as a BST
54     if (!t) { a = b = NULL; return; }
55     push(t);
56     if (Key(t) < k) {
57         a = t;
58         split2(t->r, k, a->r, b);
59         pull(a);
60     }
61     else {
62         b = t;
63         split2(t->l, k, a, b->l);
64         pull(b);
65     }
66 }

```

1.4 undo_disjoint_set

```

1 struct DisjointSet {
2     // save() is like recursive
3     // undo() is like return
4     int n, fa[MXN], sz[MXN];
5     vector<pair<int*,int>> h;
6     vector<int> sp;
7     void init(int tn) {
8         n=tn;
9         for (int i=0; i<n; i++) sz[fa[i]=i]=1;
10        sp.clear(); h.clear();
11    }
12    void assign(int *k, int v) {
13        h.PB({k, *k});
14        *k=v;
15    }
16    void save() { sp.PB(SZ(h)); }
17    void undo() {
18        assert(!sp.empty());
19        int last=sp.back(); sp.pop_back();
20        while (SZ(h)!=last) {

```

```

21        auto x=h.back(); h.pop_back();
22        *x.F=x.S;
23    }
24 }
25 int f(int x) {
26     while (fa[x]!=x) x=fa[x];
27     return x;
28 }
29 void uni(int x, int y) {
30     x=f(x); y=f(y);
31     if (x==y) return;
32     if (sz[x]<sz[y]) swap(x, y);
33     assign(&sz[x], sz[x]+sz[y]);
34     assign(&fa[y], x);
35 }
36 }djs;

```

1.5 整體二分

```

1 void totBS(int L, int R, vector<Item> M){
2     if(Q.empty()) return; //維護全域B陣列
3     if(L==R) 整個M的答案=r, return;
4     int mid = (L+R)/2;
5     vector<Item> mL, mR;
6     do_modify_B_with_divide(mid,M);
7     //讓B陣列在遞迴的時候只會保留[L~mid]的資訊
8     undo_modify_B(mid,M);
9     totBS(L,mid,mL);
10    totBS(mid+1,R,mR);
11 }

```

2 Flow

2.1 DFSflow

```

1 struct Edge{
2     int to, cap, rev;
3     Edge(int a,int b,int c) {
4         to = a; cap = b; rev = c;
5     }
6 };
7 // IMPORANT, MAXV != MAXN
8 vector<Edge> G[MAXV];
9 int V, flow[MAXV];
10 void init(int _V){
11     V = _V;
12     for(int i=0; i<=V; i++) G[i].clear();
13 }
14 void add_edge(int f,int t,int c, bool
15     directed){
16     int s1 = G[f].size(), s2 = G[t].size();
17     G[f].push_back(Edge(t,c,s2));
18     G[t].push_back(Edge(f,c,!directed,s1));
19 }
20 int dfs(int v, int t) {
21     if(v == t) return flow[t];
22     for(Edge &e : G[v]){

```

```

22     if(e.cap==0||flow[e.to]!=-1)
23         continue;
24     flow[e.to] = min(flow[v], e.cap);
25     int f = dfs(e.to, t);
26     if (f!=0) {
27         e.cap -= f;
28         G[e.to][e.rev].cap += f;
29         return f;
30     }
31     return 0;
32 }
33 int max_flow(int s,int t){
34     int ans = 0, add = 0;
35     do {
36         fill(flow,flow+V+1,-1);
37         flow[s] = INF;
38         add = dfs(s, t);
39         ans += add;
40     } while (add != 0);
41     return ans;
42 }

```

2.2 Dinic

```

1 struct Edge{
2     int f,to,rev;
3     T c;
4     Edge(int _to,int _r,T _c):to(_to),rev(_r
5     ),c(_c){}
6 };
7 // IMPOREANT
8 // maxn is the number of vertices in the
9 // graph
10 // Not the N in the problem statement!!
11 vector<Edge> G[maxn];
12 int level[maxn],st, end, n;
13 int cur[maxn];
14 void init(int _n){
15     n = _n;
16     for(int i=0; i<n; i++) G[i].clear();
17 }
18 void add_edge(int f,int t,T c, bool directed
19 ){
20     int r1 = G[f].size(), r2 = G[t].size();
21     G[f].push_back(Edge(t,r2,c));
22     G[t].push_back(Edge(f,r1,directed?0:c));
23 }
24 bool BFS(int s,int t){
25     queue<int> Q;
26     for(int i=0; i<n; i++) level[i] = 0;
27     level[s] = 1;
28     Q.push(s);
29     while(!Q.empty()){
30         int x = Q.front(); Q.pop();
31         for(int i=0; i<G[x].size(); i++){
32             Edge e = G[x][i];
33             if(e.c==0 || level[e.to])
34                 continue;

```

```

35         level[e.to] = level[x] + 1;
36         Q.push(e.to);
37     }
38     return level[t]!=0;
39 }
40 T DFS(int s,T cur_flow){ // can't exceed c
41     if(s==end) return cur_flow;
42     T ans = 0, temp, total = 0;
43     for(int& i=cur[s]; i<G[s].size(); i++){
44         Edge &e = G[s][i];
45         if(e.c==0 || level[e.to]!=level[s
46             ]+1) continue;
47         temp = DFS(e.to, min(e.c, cur_flow))
48             ;
49         if(temp!=0){
50             e.c -= temp;
51             G[e.to][e.rev].c += temp;
52             cur_flow -= temp;
53             total += temp;
54             if(cur_flow==0) break;
55         }
56     }
57     return total;
58 }
59 T max_flow(int s,int t){
60     /* If you want to incrementally doing
61     maxFlow,
62     you need to add the result manually.
63     This function returns difference in
64     that case. */
65     T ans = 0;
66     st = s, end = t;
67     while(BFS(st,t)){
68         while(true) {
69             memset(cur, 0, sizeof(cur));
70             T temp = DFS(st,INF);
71             if(temp==0) break;
72             ans += temp;
73         }
74     }
75     return ans;

```

2.3 min_cost_flow

```

1 // 0-based
2 #define fi first
3 #define se second
4 struct Edge {
5     int to,cap;
6     int cost,rev;
7 };
8 static const int MAXV = 605;
9 int V,E;
10 vector<Edge> G[MAXV];
11 void init(int _V) {
12     V=_V;
13     for (int i=0;i<V;i++) G[i].clear();

```

```

16 }
17 void add_edge(int fr, int to, int cap, int
18 cost) {
19     int a = G[fr].size(), b = G[to].size();
20     G[fr].push_back({to,cap,cost,b});
21     G[to].push_back({fr,0,-cost,a});
22 }
23 bool SPFA(int s, int t, int &ans_flow, int &
24 ans_cost) {
25     queue<int> que;
26     PII pre[MAXV];
27     int flow[MAXV], dist[MAXV];
28     bool inque[MAXV];
29     for (int i=0;i<V;i++) {
30         dist[i]=INF;
31         inque[i]=false;
32     }
33     dist[s]=0;
34     flow[s]=INF;
35     inque[s]=true;
36     que.push(s);
37     while (!que.empty()) {
38         int v=que.front(); que.pop();
39         inque[v]=false;
40         for (int i=0;i<G[v].size();i++) {
41             const Edge &e = G[v][i];
42             if (e.cap>0 && dist[v]+e.cost<
43                 dist[e.to]) {
44                 flow[e.to]=min(flow[v],e.cap
45                     );
46                 dist[e.to]=dist[v]+e.cost;
47                 pre[e.to]={v,i};
48                 if (!inque[e.to]) que.push(e
49                     .to),inque[e.to]=true;
50             }
51         }
52     }
53     if (dist[t]==INF) return false;
54     //if (dist[t]>=0) return false;
55     // Add above line -> min cost > max flow
56     // (priority)
57     // Without -> max flow > min cost
58     int v=t,f=flow[t];
59     ans_flow+=flow[t];
60     ans_cost+=(dist[t]*flow[t]);
61     while (v!=s) {
62         Edge &e = G[pre[v].fi][pre[v].se];
63         e.cap-=f;
64         G[v][e.rev].cap+=f;
65         v=pre[v].fi;
66     }
67     return true;
68 }
69 pair<int,int> min_cost_flow(int s, int t) {
70     int ans_flow=0, ans_cost=0;
71     while (SPFA(s,t,ans_flow,ans_cost));
72     return make_pair(ans_flow,ans_cost);
73 }

```

3 Geometry

3.1 circle

```

1 /* Common tangent, circle is a point c and
2 radius r */
3 void get_tangent(Point c, double r1, double
4 r2, vector<Line> &ans) {
5     double r = r2 - r1;
6     double z = c.x*c.x + c.y*c.y;
7     double d = z - r*r;
8     if (d < -EPS) return;
9     d = sqrt(abs(d));
10     Line l;
11     l.a = (c.x * r + c.y * d) / z;
12     l.b = (c.y * r - c.x * d) / z;
13     l.c = r1;
14     ans.push_back(l);
15 }
16 vector<Line> tangents(Circle a, Circle b) {
17     // Tangent line of two circles, may have
18     // 0, 1, 2, 3, 4, inf solutions
19     // In case 0 or inf (a = b), no line will
20     // be reported. Otherwise,
21     // this program always find 4 lines, even
22     // if some of them are the same.
23     vector<Line> ans;
24     for (int i=-1; i<=1; i+=2)
25         for (int j=-1; j<=1; j+=2)
26             get_tangent(b.c-a.c, a.r*i, b.r*
27                 j, ans);
28     for (size_t i=0; i<ans.size(); ++i)
29         ans[i].c -= ans[i].a * a.c.x + ans[i
30             ].b * a.c.y;
31     return ans;
32 }
33 // Circle-Line intersection, Line:ax+by+c=0
34 vector<Point> CL_intersection(Circle cir,
35     Line li) {
36     // Li.pton(); // To Ax+By+C=0
37     Point o = cir.c;
38     li.c += li.a*o.x + li.b*o.y; // Shift w.r.
39     // t. cir.c
40     vector<Point> res;
41     double r = cir.r, a = li.a, b = li.b, c =
42         li.c;
43     double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a
44         +b*b);
45     if (c*c > r*r*(a*a+b*b)+EPS) {
46         return res; // No point
47     }
48     else if (abs(c*c - r*r*(a*a+b*b)) < EPS) {
49         res.push_back({x0 + o.x, y0 + o.y}); //
50         // 1 point
51     }
52     else {
53         double d = r*r - c*c/(a*a+b*b);
54         double mult = sqrt (d / (a*a+b*b));
55         double ax, ay, bx, by;
56         ax = x0 + b * mult;
57         bx = x0 - b * mult;
58         ay = y0 + a * mult;
59         by = y0 - a * mult;

```

3.3 geometry

```

48     by = y0 + a * mult;
49     res.push_back({ax + o.x, ay + o.y});
50     // 2 points
51     res.push_back({bx + o.x, by + o.y});
52 }
53 return res;
54 }
55 // Circle-circle intersection
56 vector<Point> CC_intersection(Circle a,
57     Circle b) {
58     if (a.c.x == b.c.x && a.c.y == b.c.y && a.
59         r == b.r) {
60         return vector<Point>(); // coincide, inf
61             points
62     }
63     Point o = a.c;
64     b.c = b.c - o; // Shift
65     a.c = {0.0, 0.0};
66     double x2 = b.c.x, y2 = b.c.y, r1 = a.r,
67         r2 = b.r;
68     Line li = {-2*x2, -2*y2, x2*x2 + y2*y2 +
69         r1*r1 - r2*r2}; // Ax+By+C = 0
70     vector<Point> res = CL_intersection(a, li)
71         ;
72     for (Point &p : res) {
73         p.x += o.x;
74         p.y += o.y;
75     }
76     return res;
77 }

```

3.2 convex_hull

```

1 void convex_hull(vector<Point> &ps, vector<
    Point> &hull) {
2     // Find convex hull of ps, store in hull
3     vector<Point> &stk=hull;
4     stk.resize(ps.size()+1);
5     sort(ps.begin(),ps.end()); // Using x to
6         cmp, y secondary.
7     int t=-1; // top
8     for (int i=0;i<ps.size();i++) {
9         // cross<-EPS -> count collinear, cross<
10             EPS -> not
11         while (t>=1&&(stk[t]-stk[t-1]).cross(ps[
12             i]-stk[t])<EPS) t--;
13         stk[++t]=ps[i];
14     }
15     int low=t;
16     for (int i=ps.size()-2;i>=0;i--) {
17         // cross<-EPS -> count collinear, cross<
18             EPS -> not
19         while (t>low&&(stk[t]-stk[t-1]).cross(ps[
20             i]-stk[t])<EPS) t--;
21         stk[++t]=ps[i];
22     }
23     stk.resize(t); // pop_back contain in this
24         instruction
25 }

```

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const double PI=acos(-1);
4
5 struct Point {
6     double x,y;
7     double cross(const Point &v) const {
8         return x*v.y-y*v.x;
9     }
10    double dot(const Point &v) const {
11        return x*v.x+y*v.y;
12    }
13    Point normal() { // Normal vector to the
14        left
15        return {-y,x};
16    }
17    double angle(const Point &v) const {
18        // Angle from *this to v in [-pi,pi].
19        double ang = atan2(cross(v),dot(v));
20        return ang < 0 ? ang + PI * 2 : ang;
21    }
22    double getA()const{//angle to x-axis
23        T A=atan2(y,x);//<0 when exceed PI
24        if(A<=-PI/2)A+=PI*2;
25        return A;
26    }
27    Point rotate_about(double theta, const
28        Point &p) const {
29        // Rotate this point conterClockwise by
30            theta about p
31        double nx=x-p.x,ny=y-p.y;
32        return {nx*cos(theta)-ny*sin(theta)+p.x,
33            nx*sin(theta)+ny*cos(theta)+p.y};
34    }
35 };
36 struct Line {
37     // IMPORTANT, remember to transform
38     // between two-point form
39     // and normal form by yourself, some
40     // methods may need them.
41     Point p1,p2;
42     double a,b,c; // ax+by+c=0
43     Line(){}
44     void pton() {
45         a=p1.y-p2.y;
46         b=p2.x-p1.x;
47         c=-a*p1.x-b*p1.y;
48     }
49     int relation(const Point &p) {
50         // For line, 0 if point on line
51         // -1 if left, 1 if right
52         Point dir=p2-p1;
53         double crs=dir.cross(p-p1);
54         return crs==0?0:crs<0?-1:1;
55     }
56     Point normal() { // normal vector to the
57         left.
58         Point dir=p2-p1;
59         return {-dir.y,dir.x};
60     }
61     bool on_segment(const Point &p) {
62         // Point on segment

```

```

57     return relation(p)==0&&(p2-p).dot(p1-p)
58         <=0;
59 }
60 bool parallel(const Line &l) {
61     // Two line parallel
62     return (p2-p1).cross(l.p2-l.p1)==0;
63 }
64 bool equal(const Line &l) {
65     // Two line equal
66     return relation(l.p1)==0&&relation(l.p2)
67         ==0;
68 }
69 bool cross_seg(const Line &seg) {
70     // Line intersect segment
71     Point dir=p2-p1;
72     return dir.cross(seg.p1-p1)*dir.cross(
73         seg.p2-p1)<=0;
74 }
75 int seg_intersect(const Line &s) const{
76     // Two segment intersect
77     // 0 -> no, 1 -> one point, -1 ->
78         infinity
79     Point dir=p2-p1, dir2=s.p2-s.p1;
80     double c1=dir.cross(s.p2-p1);
81     double c2=dir.cross(s.p1-p1);
82     double c3=dir2.cross(p2-s.p1);
83     double c4=dir2.cross(p1-s.p1);
84     if (c1==0&&c2==0) {
85         if((s.p2-p1).dot(s.p1-p1)>0&&(s.p2-p2)
86             .dot(s.p1-p1)>0&&
87             (p1-s.p1).dot(p2-s.p1)>0&&(p1-s.p2)
88             .dot(p2-s.p1)>0) return 0;
89         if(p1==s.p1&&(p2-p1).dot(s.p2-p1)<=0)
90             return 1;
91         if(p1==s.p2&&(p2-p1).dot(s.p1-p1)<=0)
92             return 1;
93         if(p2==s.p1&&(p1-p2).dot(s.p2-p2)<=0)
94             return 1;
95         if(p2==s.p2&&(p1-p2).dot(s.p1-p2)<=0)
96             return 1;
97         return -1;
98     }else if(c1*c2<=0&&c3*c4<=0)return 1;
99     return 0;
100 }
101 Point intersection(Line l) {
102     // RE if d1.cross(d2) == 0 (parallel /
103         coincide)
104     Point d1 = p2 - p1, d2 = l.p2 - l.p1;
105     return p1 + d1 * ((l.p1 - p1).cross(d2)
106         / d1.cross(d2));
107 }
108 Point seg_intersection(Line &s) const {
109     Point dir=p2-p1, dir2=s.p2-s.p1;
110     // pton(); L.pton();
111     double c1=dir.cross(s.p2-p1);
112     double c2=dir.cross(s.p1-p1);
113     double c3=dir2.cross(p2-s.p1);
114     double c4=dir2.cross(p1-s.p1);
115     if (c1==0&&c2==0) {
116         if(p1==s.p1&&(p2-p1).dot(s.p2-p1)<=0)
117             return p1;
118         if(p1==s.p2&&(p2-p1).dot(s.p1-p1)<=0)
119             return p1;
120         if(p2==s.p1&&(p1-p2).dot(s.p2-p2)<=0)
121             return p2;
122     }

```

```

107     if(p2==s.p2&&(p1-p2).dot(s.p1-p2)<=0)
108         return p2;
109     }else if(c1*c2<=0&&c3*c4<=0)return
110         line_intersection(s);
111     // Reaches here means either INF or NOT
112     ANY
113     // Use seg_intersect to check 0u0
114     return {1234,4321};
115 }
116 double dist(const Point &p, bool
117     is_segment) const {
118     // Point to Line/segment
119     Point dir=p2-p1,v=p-p1;
120     if (is_segment) {
121         if (dir.dot(v)<0) return v.len();
122         if ((p1-p2).dot(p-p2)<0) return (p-p2)
123             .len();
124     }
125     double d=abs(dir.cross(v))/dir.len();
126     return d;
127 }
128 template<typename T>
129 struct polygon{
130     polygon(){}
131     vector<point<T>> p;//counterclockwise
132     T area()const{
133         T ans=0;
134         for(int i=p.size()-1,j=0;j<(int)p.size()
135             ;i=j++){
136             ans+=p[i].cross(p[j]);
137         }
138         return ans/2;
139     }
140     point<T> center_of_mass()const{
141         T cx=0,cy=0,w=0;
142         for(int i=p.size()-1,j=0;j<(int)p.size()
143             ;i=j++){
144             T a=p[i].cross(p[j]);
145             cx+=(p[i].x+p[j].x)*a;
146             cy+=(p[i].y+p[j].y)*a;
147             w+=a;
148         }
149         return point<T>(cx/3/w,cy/3/w);
150     }
151     char ahas(const point<T>& t)const{//return
152         1 if in simple polygon, -1 if on, 0
153         if no.
154     bool c=0;
155     for(int i=0,j=p.size()-1;i<p.size();j=i
156         ++){
157         if(line<T>(p[i],p[j]).point_on_segment
158             (t))return -1;
159         else if((p[i].y>t.y)!=p[j].y>t.y)&&
160             t.x<(p[j].x+p[i].x)*(t.y-p[i].y)/(p[j]
161             .y-p[i].y)+p[i].x)
162             c=!c;
163         return c;
164     }
165     char point_in_convex(const point<T>&x)
166         const{
167         int l=1,r=(int)p.size()-2;
168         while(l<=r){//return 1 if in convex
169             polygon, -1 if on, 0 if no.
170             int mid=(l+r)/2;
171             T a1=(p[mid]-p[0]).cross(x-p[0]);

```

```

159 T a2=(p[mid+1]-p[0]).cross(x-p[0]);
160 if(a1>=0&&a2<=0){
161     T res=(p[mid+1]-p[mid]).cross(x-p[
162         mid]);
163     return res>0?1:(res>0?-1:0);
164 }else if(a1<0)r=mid-1;
165 else l=mid+1;
166 }
167 return 0;
168 }
169 vector<T> getA()const{//angle of each edge
170     to x-axis
171     vector<T>res;//must be increasing
172     for(size_t i=0;i<p.size();++i)
173         res.push_back((p[(i+1)%p.size()]-p[i])
174             .getA());
175     return res;
176 }
177 bool line_intersect(const vector<T>&A,
178     const line<T> &l)const{//O(LogN)
179     int f1=upper_bound(A.begin(),A.end(),(l.
180         p1-l.p2).getA())-A.begin();
181     int f2=upper_bound(A.begin(),A.end(),(l.
182         p2-l.p1).getA())-A.begin();
183     return l.cross_seg(line<T>(p[f1],p[f2]))
184         ;
185 }
186 polygon cut(const line<T> &l)const{
187     polygon ans;//convex polygon cut by a
188     line, left side of the line is
189     remained.
190     for(int n=p.size(),i=n-1,j=0;j<n;i=j++){
191         if(l.ori(p[i])>=0){
192             ans.p.push_back(p[i]);
193             if(l.ori(p[j])<0)
194                 ans.p.push_back(l.
195                     line_intersection(line<T>(p[i]
196                         ],p[j])));
197         }else if(l.ori(p[j])>0)
198             ans.p.push_back(l.line_intersection(
199                 line<T>(p[i],p[j])));
200     }
201     return ans;
202 }
203 static bool graham_cmp(const point<T>& a,
204     const point<T>& b){//CMP for finding
205     hull
206     return (a.x<b.x)||((a.x==b.x&&a.y<b.y));
207 }
208 void graham(vector<point<T> > &s){//convex
209     hull
210     sort(s.begin(),s.end(),graham_cmp);
211     p.resize(s.size()+1);
212     int m=0;
213     for(size_t i=0;i<s.size();++i){
214         while(m>=2&&(p[m-1]-p[m-2]).cross(s[i]
215             ]-p[m-2])<=0)--m;
216         p[m++]=s[i];
217     }
218     for(int i=s.size()-2,t=m+1;i>0;--i){
219         while(m>=t&&(p[m-1]-p[m-2]).cross(s[i]
220             ]-p[m-2])<=0)--m;
221         p[m++]=s[i];
222     }
223     if(s.size()>1)--m;
224     p.resize(m);
225 }
226 }
227 T diameter(){
228     int n=p.size(),t=1;
229     T ans=0;p.push_back(p[0]);
230     for(int i=0;i<n;i++){
231         point<T> now=p[i+1]-p[i];
232         while(now.cross(p[t+1]-p[i])>now.cross
233             (p[t]-p[i]))t=(t+1)%n;
234         ans=max(ans,(p[i]-p[t]).abs2());
235     }
236     return p.pop_back(),ans;
237 }
238 T min_cover_rectangle(){// find convex
239     hull before call this
240     int n=p.size(),t=1,r=1,l;
241     if(n<3)return 0;
242     T ans=1e99;p.push_back(p[0]);
243     for(int i=0;i<n;i++){
244         point<T> now=p[i+1]-p[i];
245         while(now.cross(p[t+1]-p[i])>now.cross
246             (p[t]-p[i]))t=(t+1)%n;
247         while(now.dot(p[r+1]-p[i])>now.dot(p[r]
248             ]-p[i]))r=(r+1)%n;
249         if(!l)l=r;
250         while(now.dot(p[l+1]-p[i])<now.dot(p[
251             l]-p[i]))l=(l+1)%n;
252         T d=now.abs2();
253         T tmp=now.cross(p[t]-p[i])*(now.dot(p[
254             r]-p[i])-now.dot(p[l]-p[i]))/d;
255         ans=min(ans,tmp);
256     }
257     return p.pop_back(),ans;
258 }
259 T dis2(polygon &p1){//square of distance
260     of two convex polygon
261     vector<point<T> > &P=p,&Q=p1.p;
262     int n=P.size(),m=Q.size(),l=0,r=0;
263     for(int i=0;i<n;++i)if(P[i].y<P[l].y)l=i;
264     for(int i=0;i<m;++i)if(Q[i].y<Q[r].y)r=i;
265     P.push_back(P[0]),Q.push_back(Q[0]);
266     T ans=1e99;
267     for(int i=0;i<n;++i){
268         while((P[l]-P[l+1]).cross(Q[r+1]-Q[r])
269             <0)r=(r+1)%m;
270         ans=min(ans,line<T>(P[l],P[l+1]).
271             seg_dis2(line<T>(Q[r],Q[r+1])));
272         l=(l+1)%n;
273     }
274     return P.pop_back(),Q.pop_back(),ans;
275 }
276 static char sign(const point<T>&t){
277     return (t.y==0?t.x:t.y)<0;
278 }
279 static bool angle_cmp(const line<T>& A,
280     const line<T>& B){
281     point<T> a=A.p2-A.p1,b=B.p2-B.p1;
282     return sign(a)<sign(b)||((sign(a)==sign(b)
283         )&&a.cross(b)>0);
284 }
285 int halfplane_intersection(vector<line<T>
286     > &s){
287     sort(s.begin(),s.end(),angle_cmp);//half
288     plane is left side of the line
289     int L,R,n=s.size();
290     vector<point<T> > px(n);
291     vector<line<T> > q(n);
292     q[L=R=0]=s[0];
293     for(int i=1;i<n;++i){
294         while(L<R&&s[i].ori(px[R-1])<=0)--R;
295         while(L<R&&s[i].ori(px[L])<=0)++L;
296         q[++R]=s[i];
297         if(q[R].parallel(q[R-1])){
298             --R;
299             if(q[R].ori(s[i].p1)>0)q[R]=s[i];
300         }
301         if(L<R)px[R-1]=q[R-1].
302             line_intersection(q[R]);
303     }
304     while(L<R&&q[L].ori(px[R-1])<=0)--R;
305     p.clear();
306     if(R-L<=1)return 0;
307     px[R]=q[R].line_intersection(q[L]);
308     for(int i=L;i<R;++i)p.push_back(px[i]);
309     return R-L+1;
310 }
311 template<typename T>
312 struct triangle{
313     point<T> a,b,c;
314     triangle(){
315         triangle(const point<T> &a,const point<T>
316             &b,const point<T> &c):a(a),b(b),c(c){}
317     }
318     T area()const{
319         T t=(b-a).cross(c-a)/2;
320         return t>0?t:-t;
321     }
322     point<T> barycenter()const{//center of
323         mass
324         return (a+b+c)/3;
325     }
326     point<T> circumcenter()const{//outer
327         center
328         static line<T> u,v;
329         u.p1=(a+b)/2;
330         u.p2=point<T>(u.p1.x-a.y+b.y,u.p1.y+a.x-
331             b.x);
332         v.p1=(a+c)/2;
333         v.p2=point<T>(v.p1.x-a.y+c.y,v.p1.y+a.x-
334             c.x);
335         return u.line_intersection(v);
336     }
337     point<T> incenter()const{//inner center
338         T A=sqrt((b-c).abs2()),B=sqrt((a-c).abs2
339             ()),C=sqrt((a-b).abs2());
340         return point<T>(A*a.x+B*b.x+C*c.x,A*a.y+
341             B*b.y+C*c.y)/(A+B+C);
342     }
343     point<T> perpencenter()const{//
344         perpendicular(?) center
345         return barycenter()*3-circumcenter()*2;
346     }
347 };
348 }
349 int touch(Node* r, int x, int y, LL d2){
350     LL dis = sqrt(d2)+1;
351     if (x<->x1-dis || x>->x2+dis ||
352         y<->y1-dis || y>->y2+dis)
353         return 0;
354     return 1;
355 }
356 void nearest(Node* r, int x, int y,
357     int &mdID, LL &md2){
358     if (!r || !touch(r, x, y, md2)) return;
359     int id,f;
360     Node *l, *r;
361 }tree[MXN];
362 int n;
363 Node *root;
364 LL dis2(int x1, int y1, int x2, int y2) {
365     LL dx = x1-x2;
366     LL dy = y1-y2;
367     return dx*dx+dy*dy;
368 }
369 static bool cmpx(Node& a, Node& b){ return
370     a.x<b.x; }
371 static bool cmpy(Node& a, Node& b){ return
372     a.y<b.y; }
373 void init(vector<pair<int,int>> ip) {
374     n = ip.size();
375     for (int i=0; i<n; i++) {
376         tree[i].id = i;
377         tree[i].x = ip[i].first;
378         tree[i].y = ip[i].second;
379     }
380     root = build_tree(0, n-1, 0);
381 }
382 Node* build_tree(int L, int R, int dep) {
383     if (L>R) return nullptr;
384     int M = (L+R)/2;
385     tree[M].f = dep%2;
386     nth_element(tree+L, tree+M, tree+R+1,
387         tree[M].f ? cmpx : cmpy);
388     tree[M].x1 = tree[M].x2 = tree[M].x;
389     tree[M].y1 = tree[M].y2 = tree[M].y;
390     tree[M].L = build_tree(L, M-1, dep+1);
391     if (tree[M].L) {
392         tree[M].x1 = min(tree[M].x1, tree[M].L
393             ->x1);
394         tree[M].x2 = max(tree[M].x2, tree[M].L
395             ->x2);
396         tree[M].y1 = min(tree[M].y1, tree[M].L
397             ->y1);
398         tree[M].y2 = max(tree[M].y2, tree[M].L
399             ->y2);
400     }
401     tree[M].R = build_tree(M+1, R, dep+1);
402     if (tree[M].R) {
403         tree[M].x1 = min(tree[M].x1, tree[M].R
404             ->x1);
405         tree[M].x2 = max(tree[M].x2, tree[M].R
406             ->x2);
407         tree[M].y1 = min(tree[M].y1, tree[M].R
408             ->y1);
409         tree[M].y2 = max(tree[M].y2, tree[M].R
410             ->y2);
411     }
412     return tree[M];
413 }
414 int touch(Node* r, int x, int y, LL d2){
415     LL dis = sqrt(d2)+1;
416     if (x<->x1-dis || x>->x2+dis ||
417         y<->y1-dis || y>->y2+dis)
418         return 0;
419     return 1;
420 }
421 void nearest(Node* r, int x, int y,
422     int &mdID, LL &md2){
423     if (!r || !touch(r, x, y, md2)) return;
424     int id,f;
425     Node *l, *r;
426 }tree[MXN];
427 int n;
428 Node *root;
429 LL dis2(int x1, int y1, int x2, int y2) {
430     LL dx = x1-x2;
431     LL dy = y1-y2;
432     return dx*dx+dy*dy;
433 }
434 static bool cmpx(Node& a, Node& b){ return
435     a.x<b.x; }
436 static bool cmpy(Node& a, Node& b){ return
437     a.y<b.y; }
438 void init(vector<pair<int,int>> ip) {
439     n = ip.size();
440     for (int i=0; i<n; i++) {
441         tree[i].id = i;
442         tree[i].x = ip[i].first;
443         tree[i].y = ip[i].second;
444     }
445     root = build_tree(0, n-1, 0);
446 }
447 Node* build_tree(int L, int R, int dep) {
448     if (L>R) return nullptr;
449     int M = (L+R)/2;
450     tree[M].f = dep%2;
451     nth_element(tree+L, tree+M, tree+R+1,
452         tree[M].f ? cmpx : cmpy);
453     tree[M].x1 = tree[M].x2 = tree[M].x;
454     tree[M].y1 = tree[M].y2 = tree[M].y;
455     tree[M].L = build_tree(L, M-1, dep+1);
456     if (tree[M].L) {
457         tree[M].x1 = min(tree[M].x1, tree[M].L
458             ->x1);
459         tree[M].x2 = max(tree[M].x2, tree[M].L
460             ->x2);
461         tree[M].y1 = min(tree[M].y1, tree[M].L
462             ->y1);
463         tree[M].y2 = max(tree[M].y2, tree[M].L
464             ->y2);
465     }
466     tree[M].R = build_tree(M+1, R, dep+1);
467     if (tree[M].R) {
468         tree[M].x1 = min(tree[M].x1, tree[M].R
469             ->x1);
470         tree[M].x2 = max(tree[M].x2, tree[M].R
471             ->x2);
472         tree[M].y1 = min(tree[M].y1, tree[M].R
473             ->y1);
474         tree[M].y2 = max(tree[M].y2, tree[M].R
475             ->y2);
476     }
477     return tree[M];
478 }
479 int touch(Node* r, int x, int y, LL d2){
480     LL dis = sqrt(d2)+1;
481     if (x<->x1-dis || x>->x2+dis ||
482         y<->y1-dis || y>->y2+dis)
483         return 0;
484     return 1;
485 }
486 void nearest(Node* r, int x, int y,
487     int &mdID, LL &md2){
488     if (!r || !touch(r, x, y, md2)) return;
489     int id,f;
490     Node *l, *r;
491 }tree[MXN];
492 int n;
493 Node *root;
494 LL dis2(int x1, int y1, int x2, int y2) {
495     LL dx = x1-x2;
496     LL dy = y1-y2;
497     return dx*dx+dy*dy;
498 }
499 static bool cmpx(Node& a, Node& b){ return
500     a.x<b.x; }
501 static bool cmpy(Node& a, Node& b){ return
502     a.y<b.y; }
503 void init(vector<pair<int,int>> ip) {
504     n = ip.size();
505     for (int i=0; i<n; i++) {
506         tree[i].id = i;
507         tree[i].x = ip[i].first;
508         tree[i].y = ip[i].second;
509     }
510     root = build_tree(0, n-1, 0);
511 }
512 Node* build_tree(int L, int R, int dep) {
513     if (L>R) return nullptr;
514     int M = (L+R)/2;
515     tree[M].f = dep%2;
516     nth_element(tree+L, tree+M, tree+R+1,
517         tree[M].f ? cmpx : cmpy);
518     tree[M].x1 = tree[M].x2 = tree[M].x;
519     tree[M].y1 = tree[M].y2 = tree[M].y;
520     tree[M].L = build_tree(L, M-1, dep+1);
521     if (tree[M].L) {
522         tree[M].x1 = min(tree[M].x1, tree[M].L
523             ->x1);
524         tree[M].x2 = max(tree[M].x2, tree[M].L
525             ->x2);
526         tree[M].y1 = min(tree[M].y1, tree[M].L
527             ->y1);
528         tree[M].y2 = max(tree[M].y2, tree[M].L
529             ->y2);
530     }
531     tree[M].R = build_tree(M+1, R, dep+1);
532     if (tree[M].R) {
533         tree[M].x1 = min(tree[M].x1, tree[M].R
534             ->x1);
535         tree[M].x2 = max(tree[M].x2, tree[M].R
536             ->x2);
537         tree[M].y1 = min(tree[M].y1, tree[M].R
538             ->y1);
539         tree[M].y2 = max(tree[M].y2, tree[M].R
540             ->y2);
541     }
542     return tree[M];
543 }
544 int touch(Node* r, int x, int y, LL d2){
545     LL dis = sqrt(d2)+1;
546     if (x<->x1-dis || x>->x2+dis ||
547         y<->y1-dis || y>->y2+dis)
548         return 0;
549     return 1;
550 }
551 void nearest(Node* r, int x, int y,
552     int &mdID, LL &md2){
553     if (!r || !touch(r, x, y, md2)) return;
554     int id,f;
555     Node *l, *r;
556 }tree[MXN];
557 int n;
558 Node *root;
559 LL dis2(int x1, int y1, int x2, int y2) {
560     LL dx = x1-x2;
561     LL dy = y1-y2;
562     return dx*dx+dy*dy;
563 }
564 static bool cmpx(Node& a, Node& b){ return
565     a.x<b.x; }
566 static bool cmpy(Node& a, Node& b){ return
567     a.y<b.y; }
568 void init(vector<pair<int,int>> ip) {
569     n = ip.size();
570     for (int i=0; i<n; i++) {
571         tree[i].id = i;
572         tree[i].x = ip[i].first;
573         tree[i].y = ip[i].second;
574     }
575     root = build_tree(0, n-1, 0);
576 }
577 Node* build_tree(int L, int R, int dep) {
578     if (L>R) return nullptr;
579     int M = (L+R)/2;
580     tree[M].f = dep%2;
581     nth_element(tree+L, tree+M, tree+R+1,
582         tree[M].f ? cmpx : cmpy);
583     tree[M].x1 = tree[M].x2 = tree[M].x;
584     tree[M].y1 = tree[M].y2 = tree[M].y;
585     tree[M].L = build_tree(L, M-1, dep+1);
586     if (tree[M].L) {
587         tree[M].x1 = min(tree[M].x1, tree[M].L
588             ->x1);
589         tree[M].x2 = max(tree[M].x2, tree[M].L
590             ->x2);
591         tree[M].y1 = min(tree[M].y1, tree[M].L
592             ->y1);
593         tree[M].y2 = max(tree[M].y2, tree[M].L
594             ->y2);
595     }
596     tree[M].R = build_tree(M+1, R, dep+1);
597     if (tree[M].R) {
598         tree[M].x1 = min(tree[M].x1, tree[M].R
599             ->x1);
600         tree[M].x2 = max(tree[M].x2, tree[M].R
601             ->x2);
602         tree[M].y1 = min(tree[M].y1, tree[M].R
603             ->y1);
604         tree[M].y2 = max(tree[M].y2, tree[M].R
605             ->y2);
606     }
607     return tree[M];
608 }
609 int touch(Node* r, int x, int y, LL d2){
610     LL dis = sqrt(d2)+1;
611     if (x<->x1-dis || x>->x2+dis ||
612         y<->y1-dis || y>->y2+dis)
613         return 0;
614     return 1;
615 }
616 void nearest(Node* r, int x, int y,
617     int &mdID, LL &md2){
618     if (!r || !touch(r, x, y, md2)) return;
619     int id,f;
620     Node *l, *r;
621 }tree[MXN];
622 int n;
623 Node *root;
624 LL dis2(int x1, int y1, int x2, int y2) {
625     LL dx = x1-x2;
626     LL dy = y1-y2;
627     return dx*dx+dy*dy;
628 }
629 static bool cmpx(Node& a, Node& b){ return
630     a.x<b.x; }
631 static bool cmpy(Node& a, Node& b){ return
632     a.y<b.y; }
633 void init(vector<pair<int,int>> ip) {
634     n = ip.size();
635     for (int i=0; i<n; i++) {
636         tree[i].id = i;
637         tree[i].x = ip[i].first;
638         tree[i].y = ip[i].second;
639     }
640     root = build_tree(0, n-1, 0);
641 }
642 Node* build_tree(int L, int R, int dep) {
643     if (L>R) return nullptr;
644     int M = (L+R)/2;
645     tree[M].f = dep%2;
646     nth_element(tree+L, tree+M, tree+R+1,
647         tree[M].f ? cmpx : cmpy);
648     tree[M].x1 = tree[M].x2 = tree[M].x;
649     tree[M].y1 = tree[M].y2 = tree[M].y;
650     tree[M].L = build_tree(L, M-1, dep+1);
651     if (tree[M].L) {
652         tree[M].x1 = min(tree[M].x1, tree[M].L
653             ->x1);
654         tree[M].x2 = max(tree[M].x2, tree[M].L
655             ->x2);
656         tree[M].y1 = min(tree[M].y1, tree[M].L
657             ->y1);
658         tree[M].y2 = max(tree[M].y2, tree[M].L
659             ->y2);
660     }
661     tree[M].R = build_tree(M+1, R, dep+1);
662     if (tree[M].R) {
663         tree[M].x1 = min(tree[M].x1, tree[M].R
664             ->x1);
665         tree[M].x2 = max(tree[M].x2, tree[M].R
666             ->x2);
667         tree[M].y1 = min(tree[M].y1, tree[M].R
668             ->y1);
669         tree[M].y2 = max(tree[M].y2, tree[M].R
670             ->y2);
671     }
672     return tree[M];
673 }
674 int touch(Node* r, int x, int y, LL d2){
675     LL dis = sqrt(d2)+1;
676     if (x<->x1-dis || x>->x2+dis ||
677         y<->y1-dis || y>->y2+dis)
678         return 0;
679     return 1;
680 }
681 void nearest(Node* r, int x, int y,
682     int &mdID, LL &md2){
683     if (!r || !touch(r, x, y, md2)) return;
684     int id,f;
685     Node *l, *r;
686 }tree[MXN];
687 int n;
688 Node *root;
689 LL dis2(int x1, int y1, int x2, int y2) {
690     LL dx = x1-x2;
691     LL dy = y1-y2;
692     return dx*dx+dy*dy;
693 }
694 static bool cmpx(Node& a, Node& b){ return
695     a.x<b.x; }
696 static bool cmpy(Node& a, Node& b){ return
697     a.y<b.y; }
698 void init(vector<pair<int,int>> ip) {
699     n = ip.size();
700     for (int i=0; i<n; i++) {
701         tree[i].id = i;
702         tree[i].x = ip[i].first;
703         tree[i].y = ip[i].second;
704     }
705     root = build_tree(0, n-1, 0);
706 }
707 Node* build_tree(int L, int R, int dep) {
708     if (L>R) return nullptr;
709     int M = (L+R)/2;
710     tree[M].f = dep%2;
711     nth_element(tree+L, tree+M, tree+R+1,
712         tree[M].f ? cmpx : cmpy);
713     tree[M].x1 = tree[M].x2 = tree[M].x;
714     tree[M].y1 = tree[M].y2 = tree[M].y;
715     tree[M].L = build_tree(L, M-1, dep+1);
716     if (tree[M].L) {
717         tree[M].x1 = min(tree[M].x1, tree[M].L
718             ->x1);
719         tree[M].x2 = max(tree[M].x2, tree[M].L
720             ->x2);
721         tree[M].y1 = min(tree[M].y1, tree[M].L
722             ->y1);
723         tree[M].y2 = max(tree[M].y2, tree[M].L
724             ->y2);
725     }
726     tree[M].R = build_tree(M+1, R, dep+1);
727     if (tree[M].R) {
728         tree[M].x1 = min(tree[M].x1, tree[M].R
729             ->x1);
730         tree[M].x2 = max(tree[M].x2, tree[M].R
731             ->x2);
732         tree[M].y1 = min(tree[M].y1, tree[M].R
733             ->y1);
734         tree[M].y2 = max(tree[M].y2, tree[M].R
735             ->y2);
736     }
737     return tree[M];
738 }
739 int touch(Node* r, int x, int y, LL d2){
740     LL dis = sqrt(d2)+1;
741     if (x<->x1-dis || x>->x2+dis ||
742         y<->y1-dis || y>->y2+dis)
743         return 0;
744     return 1;
745 }
746 void nearest(Node* r, int x, int y,
747     int &mdID, LL &md2){
748     if (!r || !touch(r, x, y, md2)) return;
749     int id,f;
750     Node *l, *r;
751 }tree[MXN];
752 int n;
753 Node *root;
754 LL dis2(int x1, int y1, int x2, int y2) {
755     LL dx = x1-x2;
756     LL dy = y1-y2;
757     return dx*dx+dy*dy;
758 }
759 static bool cmpx(Node& a, Node& b){ return
760     a.x<b.x; }
761 static bool cmpy(Node& a, Node& b){ return
762     a.y<b.y; }
763 void init(vector<pair<int,int>> ip) {
764     n = ip.size();
765     for (int i=0; i<n; i++) {
766         tree[i].id = i;
767         tree[i].x = ip[i].first;
768         tree[i].y = ip[i].second;
769     }
770     root = build_tree(0, n-1, 0);
771 }
772 Node* build_tree(int L, int R, int dep) {
773     if (L>R) return nullptr;
774     int M = (L+R)/2;
775     tree[M].f = dep%2;
776     nth_element(tree+L, tree+M, tree+R+1,
777         tree[M].f ? cmpx : cmpy);
778     tree[M].x1 = tree[M].x2 = tree[M].x;
779     tree[M].y1 = tree[M].y2 = tree[M].y;
780     tree[M].L = build_tree(L, M-1, dep+1);
781     if (tree[M].L) {
782         tree[M].x1 = min(tree[M].x1, tree[M].L
783             ->x1);
784         tree[M].x2 = max(tree[M].x2, tree[M].L
785             ->x2);
786         tree[M].y1 = min(tree[M].y1, tree[M].L
787             ->y1);
788         tree[M].y2 = max(tree[M].y2, tree[M].L
789             ->y2);
790     }
791     tree[M].R = build_tree(M+1, R, dep+1);
792     if (tree[M].R) {
793         tree[M].x1 = min(tree[M].x1, tree[M].R
794             ->x1);
795         tree[M].x2 = max(tree[M].x2, tree[M].R
796             ->x2);
797         tree[M].y1 = min(tree[M].y1, tree[M].R
798             ->y1);
799         tree[M].y2 = max(tree[M].y2, tree[M].R
800             ->y2);
801     }
802     return tree[M];
803 }
804 int touch(Node* r, int x, int y, LL d2){
805     LL dis = sqrt(d2)+1;
806     if (x<->x1-dis || x>->x2+dis ||
807         y<->y1-dis || y>->y2+dis)
808         return 0;
809     return 1;
810 }
811 void nearest(Node* r, int x, int y,
812     int &mdID, LL &md2){
813     if (!r || !touch(r, x, y, md2)) return;
814     int id,f;
815     Node *l, *r;
816 }tree[MXN];
817 int n;
818 Node *root;
819 LL dis2(int x1, int y1, int x2, int y2) {
820     LL dx = x1-x2;
821     LL dy = y1-y2;
822     return dx*dx+dy*dy;
823 }
824 static bool cmpx(Node& a, Node& b){ return
825     a.x<b.x; }
826 static bool cmpy(Node& a, Node& b){ return
827     a.y<b.y; }
828 void init(vector<pair<int,int>> ip) {
829     n = ip.size();
830     for (int i=0; i<n; i++) {
831         tree[i].id = i;
832         tree[i].x = ip[i].first;
833         tree[i].y = ip[i].second;
834     }
835     root = build_tree(0, n-1, 0);
836 }
837 Node* build_tree(int L, int R, int dep) {
838     if (L>R) return nullptr;
839     int M = (L+R)/2;
840     tree[M].f = dep%2;
841     nth_element(tree+L, tree+M, tree+R+1,
842         tree[M].f ? cmpx : cmpy);
843     tree[M].x1 = tree[M].x2 = tree[M].x;
844     tree[M].y1 = tree[M].y2 = tree[M].y;
845     tree[M].L = build_tree(L, M-1, dep+1);
846     if (tree[M].L) {
847         tree[M].x1 = min(tree[M].x1, tree[M].L
848             ->x1);
849         tree[M].x2 = max(tree[M].x2, tree[M].L
850             ->x2);
851         tree[M].y1 = min(tree[M].y1, tree[M].L
852             ->y1);
853         tree[M].y2 = max(tree[M].y2, tree[M].L
854             ->y2);
855     }
856     tree[M].R = build_tree(M+1, R, dep+1);
857     if (tree[M].R) {
858         tree[M].x1 = min(tree[M].x1, tree[M].R
859             ->x1);
860         tree[M].x2 = max(tree[M].x2, tree[M].R
861             ->x2);
862         tree[M].y1 = min(tree[M].y1, tree[M].R
863             ->y1);
864         tree[M].y2 = max(tree[M].y2, tree[M].R
865             ->y2);
866     }
867     return tree[M];
868 }
869 int touch(Node* r, int x, int y, LL d2){
870     LL dis = sqrt(d2)+1;
871     if (x<->x1-dis || x>->x2+dis ||
872         y<->y1-dis || y>->y2+dis)
873         return 0;
874     return 1;
875 }
876 void nearest(Node* r, int x, int y,
877     int &mdID, LL &md2){
878     if (!r || !touch(r, x, y, md2)) return;
879     int id,f;
880     Node *l, *r;
881 }tree[MXN];
882 int n;
883 Node *root;
884 LL dis2(int x1, int y1, int x2, int y2) {
885     LL dx = x1-x2;
886     LL dy = y1-y2;
887     return dx*dx+dy*dy;
888 }
889 static bool cmpx(Node& a, Node& b){ return
890     a.x<b.x; }
891 static bool cmpy(Node& a, Node& b){ return
892     a.y<b.y; }
893 void init(vector<pair<int,int>> ip) {
894     n = ip.size();
895     for (int i=0; i<n; i++) {
896         tree[i].id = i;
897         tree[i].x = ip[i].first;
898         tree[i].y = ip[i].second;
899     }
900     root = build_tree(0, n-1, 0);
901 }
902 Node* build_tree(int L, int R, int dep) {
903     if (L>R) return nullptr;
904     int M = (L+R)/2;
905     tree[M].f = dep%2;
906     nth_element(tree+L, tree+M, tree+R+1,
907         tree[M].f ? cmpx : cmpy);
908     tree[M].x1 = tree[M].x2 = tree[M].x;
909     tree[M].y1 = tree[M].y2 = tree[M].y;
910     tree[M].L = build_tree(L, M-1, dep+1);
911     if (tree[M].L) {
912         tree[M].x1 = min(tree[M].x1, tree[M].L
913             ->x1);
914         tree[M].x2 = max(tree[M].x2, tree[M].L
915             ->x2);
916         tree[M].y1 = min(tree[M].y1, tree[M].L
917             ->y1);
918         tree[M].y2 = max(tree[M].y2, tree[M].L
919             ->y2);
920     }
921     tree[M].R = build_tree(M+1, R, dep+1);
922     if (tree[M].R) {
923         tree[M].x1 = min(tree[M].x1, tree[M].R
924             ->x1);
925         tree[M].x2 = max(tree[M].x2, tree[M].R
926             ->x2);
927         tree[M].y1 = min(tree[M].y1, tree[M].R
928             ->y1);
929         tree[M].y2 = max(tree[M].y2, tree[M].R
930             ->y2);
931     }
932     return tree[M];
933 }
934 int touch(Node* r, int x, int y, LL d2){
935     LL dis = sqrt(d2)+1;
936     if (x<->x1-dis || x>->x2+dis ||
937         y<->y1-dis || y>->y2+dis)
938         return 0;
939     return 1;
940 }
941 void nearest(Node* r, int x, int y,
942     int &mdID, LL &md2){
943     if (!r || !touch(r, x, y, md2)) return;
944     int id,f;
945     Node *l, *r;
946 }tree[MXN];
947 int n;
948 Node *root;
949 LL dis2(int x1, int y1, int x2, int y2) {
950     LL dx = x1-x2;
951     LL dy = y1-y2;
952     return dx*dx+dy*dy;
953 }
954 static bool cmpx(Node& a, Node& b){ return
955     a.x<b.x; }
956 static bool cmpy(Node& a, Node& b){ return
957     a.y<b.y; }
958 void init(vector<pair<int,int>> ip) {
959     n = ip.size();
960     for (int i=0; i<n; i++) {
961         tree[i].id = i;
962         tree[i].x = ip[i].first;
963         tree[i].y = ip[i].second;
964     }
965     root = build_tree(0, n-1, 0);
966 }
967 Node* build_tree(int L, int R, int dep) {
968     if (L>R) return nullptr;
969     int M = (L+R)/2;
970     tree[M].f = dep%2;
971     nth_element(tree+L, tree+M, tree+R+1,
972         tree[M].f ? cmpx : cmpy);
973     tree[M].x1 = tree[M].x2 = tree[M].x;
974     tree[M].y1 = tree[M].y2 = tree[M].y;
975     tree[M].L = build_tree(L, M-1, dep+1);
976     if (tree[M].L) {
977         tree[M].x1 = min(tree[M].x1, tree[M].L
978             ->x1);
979         tree[M].x2 = max(tree[M].x2, tree[M].L
980             ->x2);
981         tree[M].y1 = min(tree[M].y1, tree[M].L
982             ->y1);
983         tree[M].y2 = max(tree[M].y2
```



```

60 LL d2 = dis2(r->x, r->y, x, y);
61 if (d2 < md2 || (d2 == md2 && mID < r->
    id)) {
62     mID = r->id;
63     md2 = d2;
64 }
65 // search order depends on split dim
66 if ((r->f == 0 && x < r->x) ||
67     (r->f == 1 && y < r->y)) {
68     nearest(r->L, x, y, mID, md2);
69     nearest(r->R, x, y, mID, md2);
70 } else {
71     nearest(r->R, x, y, mID, md2);
72     nearest(r->L, x, y, mID, md2);
73 }
74 }
75 int query(int x, int y) {
76     int id = 1029384756;
77     LL d2 = 102938475612345678LL;
78     nearest(root, x, y, id, d2);
79     return id;
80 }
81 }tree;

```

3.5 smallest_circle

```

1 using PT=point<T>; using CPT=const PT;
2 PT circumcenter(CPT &a,CPT &b,CPT &c){
3     PT u=b-a, v=c-a;
4     T c1=u.abs2()/2, c2=v.abs2()/2;
5     T d=u.cross(v);
6     return PT(a.x+(v.y*c1-u.y*c2)/d, a.y+(u.x*
    c2-v.x*c1)/d);
7 }
8 void solve(PT p[],int n,PT &c,T &r2){
9     random_shuffle(p,p+n);
10    c=p[0]; r2=0; // c,r2 = center,radius
    square
11 for(int i=1;i<n;i++){if((p[i]-c).abs2())>r2){
12     c=p[i]; r2=0;
13 for(int j=0;j<i;j++){if((p[j]-c).abs2())>r2){
14     c.x=(p[i].x+p[j].x)/2;
15     c.y=(p[i].y+p[j].y)/2;
16     r2=(p[j]-c).abs2();
17 for(int k=0;k<j;k++){if((p[k]-c).abs2())>r2){
18     c=circumcenter(p[i],p[j],p[k]);
19     r2=(p[i]-c).abs2();
20 }
21 }
22 }
23 }

```

3.6 最近點對

```

1 template<typename _IT=point<T>*>
2 T closest_pair(_IT L, _IT R){
3     if(R-L <= 1) return INF;
4     _IT mid = L+(R-L)/2;
5     T x = mid->x;

```

```

6     T d = min(closest_pair(L,mid),closest_pair(
    mid,R));
7     inplace_merge(L, mid, R, ycmp);
8     static vector<point> b; b.clear();
9     for(auto u=L;u<R;++u){
10        if((u->x-x)*(u->x-x)>=d) continue;
11        for(auto v=b.rbegin();v!=b.rend();++v){
12            T dx=u->x-v->x, dy=u->y-v->y;
13            if(dy*dy>=d) break;
14            d=min(d,dx*dx+dy*dy);
15        }
16        b.push_back(*u);
17    }
18    return d;
19 }
20 T closest_pair(vector<point<T>> &v){
21     sort(v.begin(),v.end(),xcmp);
22     return closest_pair(v.begin(),v.end());
23 }

```

4 Graph

4.1 3989_ 穩定婚姻

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int maxn = 1100;
6
7 int manWant[maxn][maxn], nextW[maxn];
8 int women[maxn][maxn], order[maxn][maxn];
9 int wife[maxn], husband[maxn];
10 queue<int> singleDog;
11
12 void engage(int m, int w){
13     if(husband[w]!=0){
14         wife[ husband[w] ] = 0;
15         singleDog.push( husband[w] );
16         husband[w] = 0;
17     }
18     husband[w] = m;
19     wife[m] = w;
20     // cout << m << " --> " << w << endl;
21 }
22 int main()
23 {
24     int Time, n, cas = 0;
25     scanf("%d",&Time);
26
27     while(Time-- && scanf("%d",&n)==1){
28         for(int i=1; i<=n; i++){
29             for(int j=1; j<=n; j++) scanf("%d",
                &manWant[i][j]);
30             nextW[i] = 1;
31             wife[i] = 0;
32             singleDog.push(i);
33         }
34
35         for(int i=1; i<=n; i++){
36             for(int j=1; j<=n; j++){

```

```

31         scanf("%d",&women[i][j]);
32         order[i][ women[i][j] ] = j;
33     }
34     husband[i] = 0;
35 }
36
37 while(!singleDog.empty()){
38     int x = singleDog.front();
39     singleDog.pop();
40     // cout << x << endl;
41     int to = manWant[x][nextW[x]++];
42     if(husband[to]==0) engage(x, to);
43     ;
44     else if(order[to][husband[to]] >
        order[to][x]) engage(x, to);
45     ;
46     else singleDog.push(x);
47 }
48 if(cas++) printf("\n");
49 for(int i=1; i<=n; i++) printf("%d\n",
    wife[i]);
50
51 }
52 return 0;
53 }
54 }
55 }
56 }

```

4.2 blossom

```

1 struct Blossom {
2     #define MAXN 505 // Max solvable problem,
    DON'T CHANGE
3     // 1-based, IMPORTANT
4     vector<int> g[MAXN];
5     int parent[MAXN], match[MAXN], belong[MAXN];
6     int n;
7     int lca(int u, int v) {
8         static int cases = 0, used[MAXN] = {};
9         for (++cases; ; swap(u, v)) {
10             if (u == 0)
11                 continue;
12             if (used[u] == cases)
13                 return u;
14             used[u] = cases;
15             u = belong[parent[match[u]]];
16         }
17     }
18 void flower(int u, int v, int l, queue<int>
    &q) {
19     while (belong[u] != 1) {
20         parent[u] = v, v = match[u];
21         if (state[v] == 1)
22             q.push(v), state[v] = 0;
23         belong[u] = belong[v] = 1, u = parent[
            v];
24     }
25 }
26 bool bfs(int u) {
27     for (int i = 0; i <= n; i++)
28         belong[i] = i;
29     memset(state, -1, sizeof(state[0]))*(n+1);
30     queue<int> q;

```

```

31     q.push(u), state[u] = 0;
32     while (!q.empty()) {
33         u = q.front(), q.pop();
34         for (int i = 0; i < g[u].size(); i++)
35             {
36                 int v = g[u][i];
37                 if (state[v] == -1) {
38                     parent[v] = u, state[v] = 1;
39                     if (match[v] == 0) {
40                         for (int prev; u; v = prev, u =
                            parent[v]) {
41                             prev = match[u];
42                             match[u] = v;
43                             match[v] = u;
44                         }
45                         return 1;
46                     }
47                     q.push(match[v]), state[match[v]]
                        = 0;
48                 } else if (state[v] == 0 && belong[v]
                    != belong[u]) {
49                     int l = lca(u, v);
50                     flower(v, u, l, q);
51                     flower(u, v, l, q);
52                 }
53             }
54     }
55     return 0;
56 }
57 int blossom() {
58     memset(parent, 0, sizeof(parent[0]))*(n
        +1);
59     memset(match, 0, sizeof(match[0]))*(n+1);
60     ;
61     int ret = 0;
62     for (int i = 1; i <= n; i++) {
63         if (match[i] == 0 && bfs(i))
64             ret++;
65     }
66     return ret;
67 }
68 void addEdge(int x, int y) {
69     g[x].push_back(y), g[y].push_back(x);
70 }
71 void init(int _n) {
72     n = _n;
73     for (int i = 0; i <= n; i++)
74         g[i].clear();
75 }
76 } algo;

```

4.3 Eulerian_cycle

```

1 // The cycle will be output in reverse order
2 // if you want eulerian "path",
3 // Add one edge, find cycle, transform to
    path
4 void dfs(int v) {
5     while(!g[v].empty()) {
6         int u = g[v].back();
7         g[v].pop_back();
8         dfs(u);
9         output(Edge(v, u)); // v to u

```

```
10 }
11 }
```

4.4 KM

```
1 // Maximum Bipartite Weighted Matching (
  Perfect Match)
2 static const int MXN = 650;
3 static const int INF = 2147483647; // LL
4 int n, match[MXN], vx[MXN], vy[MXN];
5 int edge[MXN][MXN], lx[MXN], ly[MXN], slack[MXN];
6 // ^^^^ LL
7 void init(int _n){
8     n = _n;
9     for(int i=0; i<n; i++) for(int j=0; j<n; j++)
10         edge[i][j] = 0;
11 }
12 void addEdge(int x, int y, int w) // LL
13 { edge[x][y] = w; }
14 bool DFS(int x){
15     vx[x] = 1;
16     for(int y=0; y<n; y++){
17         if (vy[y] continue;
18         if (lx[x]+ly[y] > edge[x][y]){
19             slack[y]=min(slack[y], lx[x]+ly[y]-
20                 edge[x][y]);
21         } else {
22             vy[y] = 1;
23             if (match[y] == -1 || DFS(match[y]))
24                 { match[y] = x; return true; }
25         }
26     }
27     return false;
28 }
29 int solve(){
30     fill(match, match+n, -1);
31     fill(lx, lx+n, -INF); fill(ly, ly+n, 0);
32     for(int i=0; i<n; i++){
33         for(int j=0; j<n; j++){
34             lx[i] = max(lx[i], edge[i][j]);
35         }
36         for(int i=0; i<n; i++){
37             fill(slack, slack+n, INF);
38             while (true){
39                 fill(vx, vx+n, 0); fill(vy, vy+n, 0);
40                 if ( DFS(i) ) break;
41                 int d = INF; // Long Long
42                 for(int j=0; j<n; j++){
43                     if (!vy[j]) d = min(d, slack[j]);
44                 }
45                 for(int j=0; j<n; j++){
46                     if (vx[j]) lx[j] -= d;
47                     if (vy[j]) ly[j] += d;
48                     else slack[j] -= d;
49                 }
50             }
51             int res=0;
52             for(int i=0; i<n; i++){
53                 res += edge[match[i]][i];
54             }
55             return res;
56 }
```

4.5 MaximumClique

```
1 struct MaxClique{
2     static const int MAXN=105;
3     int N, ans;
4     int g[MAXN][MAXN], dp[MAXN], stk[MAXN][MAXN];
5     int sol[MAXN], tmp[MAXN]; //sol[0~ans-1] 為答案
6     void init(int n){
7         N=n; //0-base
8         memset(g, 0, sizeof(g));
9     }
10    void add_edge(int u, int v){
11        g[u][v]=g[v][u]=1;
12    }
13    int dfs(int ns, int dep){
14        if(!ns){
15            if(dep>ans){
16                ans=dep;
17                memcpy(sol, tmp, sizeof tmp);
18                return 1;
19            } else return 0;
20        }
21        for(int i=0; i<ns; ++i){
22            if(dep+ns-i<=ans) return 0;
23            int u=stk[dep][i], cnt=0;
24            if(dep+dp[u]<=ans) return 0;
25            for(int j=i+1; j<ns; ++j){
26                int v=stk[dep][j];
27                if(g[u][v]) stk[dep+1][cnt++]=v;
28            }
29            tmp[dep]=u;
30            if(dfs(cnt, dep+1)) return 1;
31        }
32        return 0;
33    }
34    int clique(){
35        int u, v, ns;
36        for(ans=0, u=N-1; u>=0; --u){
37            for(ns=0, tmp[0]=u, v=u+1; v<N; ++v){
38                if(g[u][v]) stk[1][ns++]=v;
39            }
40            dfs(ns, 1), dp[u]=ans;
41        }
42        return ans;
43    }
44 };
45
```

4.6 MinimumMeanCycle

```
1 #include<cstdio> //for DBL_MAX
2 int dp[MAXN][MAXN]; // 1-base, 0(NM)
3 vector<tuple<int, int, int>> edge;
4 double mmc(int n){ //allow negative weight
5     const int INF=0x3f3f3f3f;
6     for(int t=0; t<n; ++t){
7         memset(dp[t+1], 0x3f, sizeof(dp[t+1]));
8         for(const auto &e: edge){
9             int u, v, w;
10            tie(u, v, w) = e;
11            dp[t+1][v]=min(dp[t+1][v], dp[t][u]+w);
12        }
13    }
14 }
```

```
12 }
13 }
14 double res = DBL_MAX;
15 for(int u=1; u<=n; ++u){
16     if(dp[n][u]==INF) continue;
17     double val = -DBL_MAX;
18     for(int t=0; t<n; ++t){
19         val=max(val, (dp[n][u]-dp[t][u])*1.0/(n-t));
20     }
21     res=min(res, val);
22 }
23 return res;
24 }
```

4.7 SAT2

```
1 int N, sid[MAXV*2]; // all 1-based
2 bool vis[MAXV*2], sol[MAXV]; // 1 if i is true
3 vector<int> stk, G[MAXV*2], Gr[MAXV*2];
4 void init(int _N) {
5     N = _N;
6 }
7 int get_not(int x) {
8     return x <= N ? x + N : x - N;
9 }
10 void add_edge(int x, int y) {
11     G[x].push_back(y);
12     Gr[y].push_back(x);
13 }
14 void add_or(int x, int y) {
15     add_edge(get_not(x), y);
16     add_edge(get_not(y), x);
17 }
18 void dfs(int v) {
19     vis[v] = 1;
20     for(int to : G[v]) {
21         if (!vis[to]) {
22             dfs(to);
23         }
24     }
25     stk.push_back(v);
26 }
27 void rdfs(int v, int root) {
28     sid[v] = root;
29     for(int to : Gr[v]) {
30         if (sid[to] == 0) {
31             rdfs(to, root);
32         }
33     }
34 }
35 bool solve() {
36     int V = 2 * N;
37     fill(vis, vis + V + 1, 0);
38     fill(sid, sid + V + 1, 0);
39     for(int i = 1; i <= V; i++) {
40         if (!vis[i]) {
41             dfs(i);
42         }
43     }
44     int cnt = 0;
45 }
```

```
46 for(int i = (int) stk.size() - 1; i >= 0; i--) {
47     if (sid[stk[i]] == 0) {
48         rdfs(stk[i], ++cnt);
49     }
50 }
51 for(int i = 1; i <= N; i++) {
52     if (sid[i] == sid[i + N]) return false;
53     sol[i] = (sid[i + N] < sid[i]);
54 }
55 return true;
56 }
```

4.8 一般圖最小權完美匹配

```
1 struct Graph {
2     // Minimum General Weighted Matching (
  Perfect Match) 0-base
3     static const int MXN = 105;
4     int n, edge[MXN][MXN];
5     int match[MXN], dis[MXN], onstk[MXN];
6     vector<int> stk;
7     void init(int _n) {
8         n = _n;
9         for(int i=0; i<n; i++){
10             for(int j=0; j<n; j++){
11                 edge[i][j] = 0;
12             }
13         }
14         void add_edge(int u, int v, int w) {
15             edge[u][v] = edge[v][u] = w;
16         }
17         bool SPFA(int u){
18             if (onstk[u]) return true;
19             stk.push_back(u);
20             onstk[u] = 1;
21             for(int v=0; v<n; v++){
22                 if (u != v && match[u] != v && !onstk[v]){
23                     int m = match[v];
24                     if (dis[m] > dis[u] - edge[v][m] + edge[u][v]){
25                         dis[m] = dis[u] - edge[v][m] + edge[u][v];
26                         onstk[v] = 1;
27                         stk.push_back(v);
28                         if (SPFA(m)) return true;
29                         stk.pop_back();
30                         onstk[v] = 0;
31                     }
32                 }
33             }
34             onstk[u] = 0;
35             stk.pop_back();
36             return false;
37         }
38         int solve() {
39             // find a match
40             for(int i=0; i<n; i++){
41                 match[i] = i+1, match[i+1] = i;
42             }
43             for(;;){
44                 int found = 0;
45             }
46         }
47     }
48 }
```

```

44 for (int i=0; i<n; i++) dis[i] = onstk 34];
45 [i] = 0;
46 for (int i=0; i<n; i++){
47     stk.clear();
48     if (!onstk[i] && SPFA(i)){
49         found = 1;
50         while (stk.size()>=2){
51             int u = stk.back(); stk.pop_back
52             ();
53             int v = stk.back(); stk.pop_back
54             ();
55             match[u] = v;
56             match[v] = u;
57         }
58         if (!found) break;
59     }
60     int ret = 0;
61     for (int i=0; i<n; i++){
62         ret += edge[i][match[i]];
63         ret /= 2;
64         return ret;
65     }
}graph;

```

4.9 全局最小割

```

1 const int INF=0x3f3f3f3f;
2 template<typename T>
3 struct stoer_wagner{// 0-base
4     static const int MAXN=150;
5     T g[MAXN][MAXN],dis[MAXN];
6     int nd[MAXN],n,s,t;
7     void init(int _n){
8         n=_n;
9         for(int i=0;i<n;++i)
10             for(int j=0;j<n;++j)g[i][j]=0;
11     }
12     void add_edge(int u,int v,T w){
13         g[u][v]=g[v][u]+=w;
14     }
15     T min_cut(){
16         T ans=INF;
17         for(int i=0;i<n;++i)nd[i]=i;
18         for(int ind,tn=n;tn>1;--tn){
19             for(int i=1;i<tn;++i)dis[ind[i]]=0;
20             for(int i=1;i<tn;++i){
21                 ind=i;
22                 for(int j=i;j<tn;++j){
23                     dis[ind[j]]+=g[ind[i-1]][nd[j]];
24                     if(dis[ind[ind]]<dis[ind[j]])ind=j;
25                 }
26                 swap(nd[ind],nd[i]);
27             }
28             if(ans>dis[ind])ans=dis[t=nd[ind]
29             ],s=nd[ind-1];
30             for(int i=0;i<tn;++i)
31                 g[ind[ind-1]][nd[i]]=g[ind[i]][nd[ind
32                 -1]]+=g[ind[i]][nd[ind]];
33         }
34     }
35     return ans;
36 }

```

4.10 平面圖判定

```

1 static const int MAXN = 20;
2 struct Edge{
3     int u, v;
4     Edge(int s, int d) : u(s), v(d) {}
5 };
6 bool isK33(int n, int degree[]){
7     int t = 0, z = 0;
8     for(int i=0;i<n;++i){
9         if(degree[i] == 3)++t;
10        else if(degree[i] == 0)++z;
11        else return false;
12    }
13    return t == 6 && t + z == n;
14 }
15 bool isK5(int n, int degree[]){
16     int f = 0, z = 0;
17     for(int i=0;i<n;++i){
18         if(degree[i] == 4)++f;
19         else if(degree[i] == 0)++z;
20         else return false;
21     }
22     return f == 5 && f + z == n;
23 }
24 // it judge a given graph is Homeomorphic
25 // with K33 or K5
26 bool isHomeomorphic(bool G[MAXN][MAXN],
27                     const int n){
28     for(;;){
29         int cnt = 0;
30         for(int i=0;i<n;++i){
31             vector<Edge> E;
32             for(int j=0;j<n&&E.size()<3;++j)
33                 if(G[i][j] && i != j)
34                     E.push_back(Edge(i, j));
35             if(E.size() == 1){
36                 G[i][E[0].v] = G[E[0].v][i] = false;
37             }else if(E.size() == 2){
38                 G[i][E[0].v] = G[E[0].v][i] = false;
39                 G[i][E[1].v] = G[E[1].v][i] = false;
40                 G[E[0].v][E[1].v] = G[E[1].v][E[0].v]
41                 ] = true;
42                 ++cnt;
43             }
44             if(cnt == 0)break;
45         }
46         static int degree[MAXN];
47         fill(degree, degree + n, 0);
48         for(int i=0;i<n;++i){
49             for(int j=i+1; j<n; ++j){
50                 if(!G[i][j])continue;
51                 ++degree[i];
52                 ++degree[j];
53             }
54         }
55         return !(isK33(n, degree) || isK5(n,
56         degree));
57     }
58 }

```

4.11 最小斯坦納樹 DP

```

1 //n個點，其中r個要構成斯坦納樹
2 //答案在max(dp[(1<r)-1][k]) k=0~n-1
3 //p表示要構成斯坦納樹的點集
4 //O( n^3 + n^3*r + n^2*2^r )
5 #define REP(i,n) for(int i=0;i<(int)n;++i)
6 const int MAXN=30,MAXM=8;// 0-base
7 const int INF=0x3f3f3f3f;
8 int dp[1<MAXN][MAXN];
9 int g[MAXN][MAXN];
10 void init(){memset(g,0x3f,sizeof(g));}
11 void add_edge(int u,int v,int w){
12     g[u][v]=g[v][u]=min(g[v][u],w);
13 }
14 void steiner(int n,int r,int *p){
15     REP(k,n)REP(i,n)REP(j,n)
16         g[i][j]=min(g[i][j],g[i][k]+g[k][j]);
17     REP(i,n)g[i][i]=0;
18     REP(i,r)REP(j,n)dp[1<i][j]=g[p[i]][j];
19     for(int i=1;i<(1<r);++i){
20         if(!i&(i-1))continue;
21         REP(j,n)dp[i][j]=INF;
22         REP(j,n){
23             int tmp=INF;
24             for(int s=i&(i-1);s;s=s-1)
25                 tmp=min(tmp,dp[s][j]+dp[i^s][j]);
26             REP(k,n)dp[i][k]=min(dp[i][k],g[j][k]+
27             tmp);
28         }
29     }
30 }

```

```

26 if(b->w<a->w)return merge(b,a);
27 swap(a->l,a->r);
28 a->l=merge(b,a->l);
29 return a;
30 }
31 void add_edge(int u,int v,T w){
32     if(u!=v)pq[v]=merge(pq[v],&(mem[m++]=
33     node(u,v,w)));
34 }
35 int find(int x,int *st){
36     return st[x]==x?st[x]=find(st[x],st);
37 }
38 T build(int root,int n){
39     T ans=0;int N=n,all=n;
40     for(int i=1;i<N;++i){
41         if(i==root||!pq[i])continue;
42         while(pq[i]){
43             pq[i]->down(),E[i]=pq[i];
44             pq[i]=merge(pq[i]->l,pq[i]->r);
45             if(find(E[i]->u,id)!=find(i,id))
46                 break;
47         }
48         if(find(E[i]->u,id)==find(i,id))
49             continue;
50         ans+=E[i]->w;
51         if(find(E[i]->u,st)==find(i,st)){
52             if(pq[i])pq[i]->tag-=E[i]->w;
53             pq[i]=pq[i];id[N]=N;
54             for(int u=find(E[i]->u,id);u!=i;u=
55             find(E[u]->u,id)){
56                 if(pq[u])pq[u]->tag-=E[u]->w;
57                 id[find(u,id)]=N;
58                 pq[N]=merge(pq[N],pq[u]);
59             }
60             st[N]=find(i,st);
61             id[find(i,id)]=N;
62             }else st[find(i,st)]=find(E[i]->u,st)
63             ,--all;
64     }
65     return all==1?ans:-INT_MAX;//圖不連通就
66     無解
67 }
68 }
69 }

```

4.12 最小樹形圖 — 朱劉

```

1 template<typename T>
2 struct zhu_liu{
3     static const int MAXN=110,MAXM=10005;
4     struct node{
5         int u,v;
6         T w,tag;
7         node *l,*r;
8         node(int u=0,int v=0,T w=0):u(u),v(v),w(
9         w),tag(0),l(0),r(0){}
10     void down(){
11         w+=tag;
12         if(l)l->tag+=tag;
13         if(r)r->tag+=tag;
14         tag=0;
15     }
16     }mem[MAXN];
17     //靜態記憶體
18     node *pq[MAXN*2],*E[MAXN*2];
19     int st[MAXN*2],id[MAXN*2],m;
20     void init(int n){
21         for(int i=1;i<=n;++i){
22             pq[i]=E[i]=0, st[i]=id[i]=i;
23             m+=0;
24         }
25     }
26     node *merge(node *a,node *b){//skew heap
27         if(!a||!b)return a?a:b;
28         a->down(),b->down();
29     }
30 }

```

4.13 穩定婚姻模板

```

1 queue<int> Q;
2 for ( i : 所有考生 ) {
3     設定在第0志願;
4     Q.push( 考生i );
5 }
6 while(Q.size()){
7     當前考生=Q.front();Q.pop();
8     while ( 此考生未分發 ) {
9         指標移到下一志願;
10        if ( 已經沒有志願 or 超出志願總數 )
11            break;
12        計算該考生在該科系加權後的總分;
13        if ( 不符合科系需求 ) continue;
14        if ( 目前科系有餘額 ) {

```

```

14 依加權後分數高低順序將考生id加入科系錄
    取名單中;
15  break;
16 }
17 if ( 目前科系已額滿 ) {
18     if ( 此考生成績比最低分數還高 ) {
19         依加權後分數高低順序將考生id加入科系
            錄取名單;
20         Q.push(被踢出的考生);
21     }
22 }
23 }
24 }

```

5 Linear_Programming

5.1 simplex

```

1  /*target:
2   max \sum_{j=1}^n A_{0,j} * x_j
3  condition:
4   \sum_{j=1}^n A_{i,j} * x_j <= A_{i,0} | i=1~m
5   x_j >= 0 | j=1~n
6  VDB = vector<double>*/
7  template<class VDB>
8  VDB simplex(int m,int n,vector<VDB> a){
9      vector<int> left(m+1), up(n+1);
10     iota(left.begin(), left.end(), n);
11     iota(up.begin(), up.end(), 0);
12     auto pivot = [&](int x, int y){
13         swap(left[x], up[y]);
14         auto k = a[x][y]; a[x][y] = 1;
15         vector<int> pos;
16         for(int j = 0; j <= n; ++j){
17             a[x][j] /= k;
18             if(a[x][j] != 0) pos.push_back(j);
19         }
20         for(int i = 0; i <= m; ++i){
21             if(a[i][y]==0 || i == x) continue;
22             k = a[i][y], a[i][y] = 0;
23             for(int j : pos) a[i][j] -= k*a[x][j];
24         }
25     };
26     for(int x,y;){
27         for(int i=x+1; i <= m; ++i)
28             if(a[i][0]<a[x][0]) x = i;
29         if(a[x][0]>=0) break;
30         for(int j=y+1; j <= n; ++j)
31             if(a[x][j]<a[x][y]) y = j;
32         if(a[x][y]>=0) return VDB();//infeasible
33         pivot(x, y);
34     }
35     for(int x,y;){
36         for(int j=y+1; j <= n; ++j)
37             if(a[0][j] > a[0][y]) y = j;
38         if(a[0][y]<=0) break;
39         x = -1;
40         for(int i=1; i<=m; ++i) if(a[i][y] > 0)
41             if(x == -1 || a[i][0]/a[i][y]
42                 < a[x][0]/a[x][y]) x = i;

```

```

43     if(x == -1) return VDB();//unbounded
44     pivot(x, y);
45 }
46 VDB ans(n + 1);
47 for(int i = 1; i <= m; ++i)
48     if(left[i] <= n) ans[left[i]] = a[i][0];
49 ans[0] = -a[0][0];
50 return ans;
51 }

```

6 Number_Theory

6.1 basic

```

1  template<typename T>
2  void gcd(const T &a,const T &b,T &d,T &x,T &y){
3      if(!b) d=a,x=1,y=0;
4      else gcd(b,a%b,d,y,x), y-=x*(a/b);
5  }
6  long long int phi[N+1];
7  void phiTable(){
8      for(int i=1;i<=N;i++)phi[i]=i;
9      for(int i=1;i<=N;i++)for(x=i*2;x<=N;x+=i)
10         phi[x]-=phi[i];
11 }
12 void all_divdown(const LL &n) { // all n/x
13     for(LL a=1;a<=n;a=n/(n/(a+1))) {
14         // dosomething;
15     }
16 }
17 const int MAXPRIME = 1000000;
18 int iscom[MAXPRIME], prime[MAXPRIME],
    primecnt;
19 int phi[MAXPRIME], mu[MAXPRIME];
20 void sieve(void){
21     memset(iscom,0,sizeof(iscom));
22     primecnt = 0;
23     phi[1] = mu[1] = 1;
24     for(int i=2;i<MAXPRIME;++i) {
25         if(!iscom[i]) {
26             prime[primecnt++] = i;
27             mu[i] = -1;
28             phi[i] = i-1;
29         }
30         for(int j=0;j<primecnt;++j) {
31             int k = i * prime[j];
32             if(k>=MAXPRIME) break;
33             iscom[k] = prime[j];
34             if(i%prime[j]==0) {
35                 mu[k] = 0;
36                 phi[k] = phi[i] * prime[j];
37                 break;
38             } else {
39                 mu[k] = -mu[i];
40                 phi[k] = phi[i] * (prime[j]-1);
41             }
42         }
43     }
44 }

```

```

45 bool g_test(const LL &g, const LL &p, const
    vector<LL> &v) {
46     for(int i=0;i<v.size();++i)
47         if(modexp(g,(p-1)/v[i],p)==1)
48             return false;
49     return true;
50 }
51 LL primitive_root(const LL &p) {
52     if(p==2) return 1;
53     vector<LL> v;
54     Factor(p-1,v);
55     v.erase(unique(v.begin(), v.end()), v.end
        ());
56     for(LL g=2;g<p;++g)
57         if(g_test(g,p,v))
58             return g;
59     puts("primitive_root NOT FOUND");
60     return -1;
61 }
62 int Legendre(const LL &a, const LL &p) {
63     return modexp(a%p,(p-1)/2,p); }
64 LL inv(const LL &a, const LL &n) {
65     LL d,x,y;
66     gcd(a,n,d,x,y);
67     return d==1 ? (x+n)%n : -1;
68 }
69 int inv[maxn];
70 LL invtable(int n,LL P){
71     inv[1]=1;
72     for(int i=2;i<n;++i)
73         inv[i]=(P-(P/i))*inv[P%i]%P;
74 }
75 }
76 LL Tonelli_Shanks(const LL &n, const LL &p)
    {
77     // x^2 = n ( mod p )
78     if(n==0) return 0;
79     if(Legendre(n,p)!=1) while(1) { puts("SQRT
        ROOT does not exist"); }
80     int S = 0;
81     LL Q = p-1;
82     while( !(Q&1) ) { Q>=1; ++S; }
83     if(S==1) return modexp(n%p,(p+1)/4,p);
84     LL z = 2;
85     for(;Legendre(z,p)!=-1;++z)
86         LL c = modexp(z,Q,p);
87         LL R = modexp(n%p,(Q+1)/2,p), t = modexp(n
            %p,Q,p);
88     int M = S;
89     while(1) {
90         if(t==1) return R;
91         LL b = modexp(c,1L<<(M-i-1),p);
92         R = LLmul(R,b,p);
93         t = LLmul(LLmul(b,b,p), t, p);
94         c = LLmul(b,b,p);
95         M = i;
96     }
97     return -1;
98 }
99 }
100 template<typename T>
101 T Euler(T n){
102     T ans=n;
103     for(T i=2;i*i<=n;++i){

```

```

105     if(n%i==0){
106         ans=ans/i*(i-1);
107         while(n%i==0)n/=i;
108     }
109 }
110 if(n>1)ans=ans/n*(n-1);
111 return ans;
112 }
113 //Chinese_remainder_theorem
114 template<typename T>
115 T pow_mod(T n,T k,T m){
116     T ans=1;
117     for(n=(n>=m?n%m:n);k;k>>=1){
118         if(k&1)ans=ans*n%m;
119         n=n*n%m;
120     }
121     return ans;
122 }
123 template<typename T>
124 T crt(vector<T> &m,vector<T> &a){
125     T M=1,tM,ans=0;
126     for(int i=0;i<(int)m.size();++i)M*=m[i];
127     for(int i=0;i<(int)a.size();++i){
128         tM=M/m[i];
129         ans=(ans+a[i]*tM%M)*pow_mod(tM,Euler(m[
            i])-1,m[i])%M;
130         /*如果m[i]是質數 · Euler(m[i])-1=m[i]-2 ·
            就不用算Euler了*/
131     }
132     return ans;
133 }
134 //java code
135 //求sqrt(N)的連分數
136 public static void Pell(int n){
137     BigInteger N,p1,p2,q1,q2,a0,a1,a2,g1,g2,h1
        ,h2,p,q;
138     g1=q2=p1=BigInteger.ZERO;
139     h1=q1=p2=BigInteger.ONE;
140     a0=a1=BigInteger.valueOf((int)Math.sqrt
        (1.0*n));
141     BigInteger ans=a0.multiply(a0);
142     if(ans.equals(BigInteger.valueOf(n))){
143         System.out.println("No solution!");
144         return ;
145     }
146     while(true){
147         g2=a1.multiply(h1).subtract(g1);
148         h2=N.subtract(g2.pow(2)).divide(h1);
149         a2=g2.add(a0).divide(h2);
150         p=a1.multiply(p2).add(p1);
151         q=a1.multiply(q2).add(q1);
152         if(p.pow(2).subtract(N.multiply(q.pow
            (2))).compareTo(BigInteger.ONE)==0)
153             break;
154         g1=g2;h1=h2;a1=a2;
155         p1=p2;p2=p;
156         q1=q2;q2=q;
157     }
158     System.out.println(p+" "+q);
159 }
160 }

```


6.2 bit_set

```

1 void sub_set(int S){
2     int sub=S;
3     do{
4         //對某集合的子集合的處理
5         sub=(sub-1)&S;
6     }while(sub!=S);
7 }
8 void k_sub_set(int k,int n){
9     int comb=(1<<k)-1,S=1<<n;
10    while(comb<S){
11        //對大小為k的子集合的處理
12        int x=comb&-comb,y=comb+x;
13        comb=((comb&~y)/x>>1)|y;
14    }
15 }

```

6.3 EXT_GCD

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long LL;
4 typedef pair < LL, LL> ii;
5
6 ii exd_gcd( LL a, LL b) {
7     if (a % b == 0) return ii(0, 1);
8     ii T = exd_gcd(b, a % b);
9     return ii( T.second, T.first - a / b * T
10     .second);
11 }
12 LL mod_inv(LL x) { // P is mod number, gcd(x
13     ,P) must be 1
14     return (exd_gcd(x,P).first%P)%P;
15 }

```

6.4 FFT

```

1 const double PI = acos(-1);
2 using cd = complex<double>;
3 // Do FFT. invert=true to do iFFT.
4 // n MUST be power of 2.
5 void fft(cd a[], int n, bool invert) {
6     for (int i = 1, j = 0; i < n; i++) {
7         int bit = n >> 1;
8         for (; j & bit; bit >>= 1)
9             j ^= bit;
10        if (i < j)
11            swap(a[i], a[j]);
12    }
13
14    for (int len = 2; len <= n; len <= 1) {
15        double ang = 2 * PI / len * (invert
16        ? -1 : 1);
17        cd wlen(cos(ang), sin(ang));
18        for (int i = 0; i < n; i += len) {
19

```

```

20            cd w(1);
21            for (int j = 0; j < len / 2; j
22            ++){
23                cd u = a[i+j], v = a[i+j+len
24                /2] * w;
25                a[i+j] = u + v;
26                a[i+j+len/2] = u - v;
27                w *= wlen;
28            }
29        }
30        if (invert) {
31            for (int i = 0; i < n; i++)
32                a[i] /= n;
33        }
34    }

```

6.5 find_real_root

```

1 // an*x^n + ... + a1x + a0 = 0;
2 int sign(double x){
3     return x < -eps ? -1 : x > eps;
4 }
5
6 double get(const vector<double>&coef, double
7     x){
8     double e = 1, s = 0;
9     for(auto i : coef) s += i*e, e *= x;
10    return s;
11 }
12 double find(const vector<double>&coef, int n
13     , double lo, double hi){
14     double sign_lo, sign_hi;
15     if( !(sign_lo = sign(get(coef,lo))) )
16         return lo;
17     if( !(sign_hi = sign(get(coef,hi))) )
18         return hi;
19     if(sign_lo * sign_hi > 0) return INF;
20     for(int stp = 0; stp < 100 && hi - lo >
21     eps; ++stp){
22         double m = (lo+hi)/2.0;
23         int sign_mid = sign(get(coef,m));
24         if(!sign_mid) return m;
25         if(sign_lo*sign_mid < 0) hi = m;
26         else lo = m;
27     }
28     return (lo+hi)/2.0;
29 }
30 vector<double> cal(vector<double>coef, int n
31 ) {
32     vector<double>res;
33     if(n == 1){
34         if(sign(coef[1])) res.pb(-coef[0]/coef
35         [1]);
36         return res;
37     }
38     vector<double>dcoef(n);
39     for(int i = 0; i < n; ++i) dcoef[i] = coef
40     [i+1]*(i+1);
41     vector<double>droot = cal(dcoef, n-1);

```

```

36     droot.insert(droot.begin(), -INF);
37     droot.pb(INF);
38     for(int i = 0; i+1 < droot.size(); ++i){
39         double tmp = find(coef, n, droot[i],
40         droot[i+1]);
41         if(tmp < INF) res.pb(tmp);
42     }
43     return res;
44 }
45 int main () {
46     vector<double>ve;
47     vector<double>ans = cal(ve, n);
48     // 視情況把答案 +eps · 避免 -0
49 }

```

6.6 FWT

```

1 vector<int> F_OR_T(vector<int> f, bool
2     inverse){
3     for(int i=0; (2<<i)<=f.size(); ++i)
4         for(int j=0; j<f.size(); j+=2<<i)
5             for(int k=0; k<(1<<i); ++k)
6                 f[j+k+(1<<i)] += f[j+k]*(inverse
7                 ?-1:1);
8     return f;
9 }
10 vector<int> rev(vector<int> A) {
11     for(int i=0; i<A.size(); i+=2)
12         swap(A[i],A[i^(A.size()-1)]);
13     return A;
14 }
15 vector<int> F_AND_T(vector<int> f, bool
16     inverse){
17     return rev(F_OR_T(rev(f), inverse));
18 }
19 vector<int> F_XOR_T(vector<int> f, bool
20     inverse){
21     for(int i=0; (2<<i)<=f.size(); ++i)
22         for(int j=0; j<f.size(); j+=2<<i)
23             for(int k=0; k<(1<<i); ++k){
24                 int u=f[j+k], v=f[j+k+(1<<i)];
25                 f[j+k+(1<<i)] = u-v, f[j+k] = u+v;
26             }
27     if(inverse) for(auto &a:f) a/=f.size();
28     return f;
29 }

```

6.7 gauss_elimination

```

1 typedef double Matrix[maxn][maxn];
2 void gauss_elimination(Matrix A, int n){
3     int r;
4     for(int i=0; i<n; i++){
5         r = i;
6         for(int j=i+1; j<n; j++)
7             if(fabs(A[j][i]) > fabs(A[r][i]))
8                 r = j;
9         if(r!=i) for(int j=0; j<n; j++)
10            swap(A[r][j], A[i][j]);

```

```

9         for(int k=i+1; k<n; k++){
10             double f = A[k][i]/A[i][i];
11             for(int j=i; j<n; j++) A[k][j]
12             -= f*A[i][j];
13         }
14     }
15     for(int i=n-1; i>=0; i--){
16         for(int j=i+1; j<n; j++)
17             A[i][n] -= A[j][n] * A[i][j];
18         A[i][n] /= A[i][i];
19     }
20 }
21 }

```

6.8 LL_mul

```

1 long long mul(long long a, long long b) {
2     long long ans = 0, step = a % MOD;
3     while (b) {
4         if (b & 1L) ans += step;
5         if (ans >= MOD) ans %= MOD;
6         step <<= 1L;
7         if (step >= MOD) step %= MOD;
8         b >>= 1L;
9     }
10    return ans % MOD;
11 }

```

6.9 Lucas

```

1 int mod_fact(int n,int &e){
2     e=0;
3     if(n==0)return 1;
4     int res=mod_fact(n/P,e);
5     e += n/P;
6     if((n/P)%2==0)return res*fact[n%P]%P;
7     return res*(P-fact[n%P])%P;
8 }
9 int Cmod(int n,int m){
10    int a1,a2,a3,e1,e2,e3;
11    a1=mod_fact(n,e1);
12    a2=mod_fact(m,e2);
13    a3=mod_fact(n-m,e3);
14    if(e1>e2+e3)return 0;
15    return a1*inv(a2*a3%P)%P;
16 }

```

6.10 Matrix

```

1 template<typename T>
2 struct Matrix{
3     using rt = std::vector<T>;
4     using mt = std::vector<rt>;
5     using matrix = Matrix<T>;
6     int r,c;

```

```

7 mt m;
8 Matrix(int r,int c):r(r),c(c),m(r,rt(c)){
9 rt& operator[](int i){return m[i];}
10 matrix operator+(const matrix &a){
11 matrix rev(r,c);
12 for(int i=0;i<r;++i)
13 for(int j=0;j<c;++j)
14 rev[i][j]=m[i][j]+a.m[i][j];
15 return rev;
16 }
17 matrix operator-(const matrix &a){
18 matrix rev(r,c);
19 for(int i=0;i<r;++i)
20 for(int j=0;j<c;++j)
21 rev[i][j]=m[i][j]-a.m[i][j];
22 return rev;
23 }
24 matrix operator*(const matrix &a){
25 matrix rev(r,a.c);
26 matrix tmp(a.c,a.r);
27 for(int i=0;i<a.r;++i)
28 for(int j=0;j<a.c;++j)
29 tmp[j][i]=a.m[i][j];
30 for(int i=0;i<r;++i)
31 for(int j=0;j<a.c;++j)
32 for(int k=0;k<c;++k)
33 rev.m[i][j]+=m[i][k]*tmp[j][k];
34 return rev;
35 }
36 bool inverse(){
37 Matrix t(r,r+c);
38 for(int y=0;y<r;y++){
39 t.m[y][c+y] = 1;
40 for(int x=0;x<c;++x)
41 t.m[y][x]=m[y][x];
42 }
43 if( !t.gas() )
44 return false;
45 for(int y=0;y<r;y++){
46 for(int x=0;x<c;++x)
47 m[y][x]=t.m[y][c+x]/t.m[y][y];
48 return true;
49 }
50 T gas(){
51 vector<T> lazy(r,1);
52 bool sign=false;
53 for(int i=0;i<r;++i){
54 if( m[i][i]==0 ){
55 int j=i+1;
56 while(j<r&&!m[j][i])j++;
57 if(j==r)continue;
58 m[i].swap(m[j]);
59 sign=!sign;
60 }
61 for(int j=0;j<r;++j){
62 if(i==j)continue;
63 lazy[j]=lazy[j]*m[i][i];
64 T mx=m[j][i];
65 for(int k=0;k<c;++k)
66 m[j][k]=m[j][k]*m[i][i]-m[i][k]*mx;
67 }
68 }
69 T det=sign?-1;
70 for(int i=0;i<r;++i){
71 det = det*m[i][i];

```

```

72 det = det/lazy[i];
73 for(auto &j:m[i])j/=lazy[i];
74 }
75 return det;
76 }
77 };

```

6.11 Miller_Rabin

```

1 LL mod_mul(LL a, LL b, LL mod) {
2 // return (__int128)a*b%mod;
3 /* In case __int128 doesn't work(32* multi
4 to avoid ovf) */
5 LL x=0,y=a%mod;
6 while(b > 0){
7 if (b&1) x = (x+y)%mod;
8 y = (y*2)%mod;
9 b >>= 1;
10 }
11 return x%mod;
12 }
13 LL qpow(LL a, LL p, LL mod) {
14 if (p<=0) return 1;
15 LL temp = qpow(a,p/2,mod);
16 temp = mod_mul(temp,temp,mod);
17 if (p&1) return mod_mul(temp,a,mod);
18 return temp;
19 }
20 bool MRtest(LL a, LL d, LL n) {
21 LL x = qpow(a,d,n);
22 if (x==1 || x==n-1) return true;
23 while (d != n-1) {
24 x = mod_mul(x,x,n);
25 d *= 2;
26 if (x==n-1) return true;
27 if (x==1) return false;
28 }
29 return false;
30 }
31 bool is_prime(LL n) {
32 if (n==2) return true;
33 if (n<2 || n%2==0) return false;
34 LL table[7] = {2, 325, 9375, 28178,
35 450775, 9780504, 1795265022}, d=n-1;
36 while (d%2 != 0) d>>=1; // n-1 = d * 2^r,
37 d is odd.
38 for (int i=0; i<7; i++) {
39 LL a = table[i] % n;
40 if (a==0 || a==1 || a==n-1) continue;
41 if (!MRtest(a,d,n)) {
42 return false;
43 }
44 }
45 return true;

```

6.12 mod_log

```

1 const LL SQRT = 10005;
2 pair<LL, LL> bs[SQRT];

```

```

3 // O(sqrt(n)log(n))
4 LL baby_giant(LL a, LL b, LL m) {
5 // Solve a^x = b (mod m) for x, gcd(a, m)
6 = 1
7 bs[0] = {1, 0};
8 for (int i = 1; i < SQRT; i++) {
9 bs[i] = {bs[i-1].first * a % m, i};
10 }
11 LL cur = b, inv = mod_inv(bs[SQRT-1].
12 first * a % m, m); // inv of G.S.
13 sort(bs, bs + SQRT);
14 for (int i = 0; i < m; i += SQRT) {
15 auto it = upper_bound(bs, bs + SQRT,
16 make_pair(cur, (LL)-1));
17 if (it != bs + SQRT && it->first == cur)
18 {
19 return i + it->second;
20 }
21 cur = cur * inv % m;
22 }
23 return -1; // no solution

```

6.13 NTT

```

1 const LL mod = 998244353;
2 const LL p_root = 3;
3 const LL root_pw = 1LL << 23;
4
5 // Do NTT under mod. invert=true to do iNTT.
6 // mod MUST be a prime, if mod=c*2^k+1, then
7 // p_root is any primitive root of mod
8 // root_pw=2^k, and n(size) MUST <= 2^k
9 // n MUST be power of 2.
10 // mod=2013265921, root_pw=1LL<<27, p_root
11 =31
12 void ntt(LL a[], int n, bool invert) {
13 LL root = qpow(p_root, (mod-1)/root_pw,
14 mod);
15 LL root_1 = mod_inv(root, mod);
16 for (int i = 1, j = 0; i < n; i++) {
17 LL bit = n >> 1;
18 for ( ; j & bit; bit >>= 1)
19 j ^= bit;
20 j ^= bit;
21 if (i < j)
22 swap(a[i], a[j]);
23 }
24 }
25 for (int len = 2; len <= n; len <= 1) {
26 LL wlen = invert ? root_1 : root;
27 for (int i = len; i < root_pw; i <= 1)
28 wlen = wlen * wlen % mod;
29
30 for (int i = 0; i < n; i += len) {
31 LL w = 1;
32 for (int j = 0; j < len / 2; j++) {
33 LL u = a[i+j], v = a[i+j+len/2] * w
34 % mod;

```

```

35 a[i+j] = u + v < mod ? u + v : u + v
36 - mod;
37 a[i+j+len/2] = u - v >= 0 ? u - v :
38 u - v + mod;
39 w = w * wlen % mod;
40 }
41 }
42 if (invert) {
43 LL n_1 = mod_inv(n, mod);
44 for (int i = 0; i < n; i++) {
45 a[i] = a[i] * n_1 % mod;
46 }
47 }
48 }

```

6.14 pollard

```

1 LL pollard_rho(LL n, int c = 1) {
2 // c is seed, rand can be replaced by 2,
3 much faster
4 LL x = rand() % n, y = x, d = 1;
5 while (d == 1) {
6 x = mod_mul(x, x, n) + c;
7 y = mod_mul(y, y, n) + c;
8 y = mod_mul(y, y, n) + c;
9 d = gcd(x - y >= 0 ? x - y : y - x, n);
10 if (d == n) return pollard_rho(n, c + 1);
11 return d;
12 }
13 void factorize(LL n, vector<LL> &pf) {
14 // N^(1/3) + LogN*(N^(1/4))
15 // For all primes <= N^(1/3)
16 for (LL p = 2; p <= (LL)1e6+5; p++) {
17 while (n % p == 0) {
18 pf.push_back(p);
19 n /= p;
20 }
21 }
22 // Use Miller-Rabin pls
23 if (n == 1) return;
24 else if (is_prime(n)) pf.push_back(n);
25 else {
26 LL d = pollard_rho(n);
27 pf.push_back(d);
28 pf.push_back(n / d);
29 }
30 }
31 }

```

7 String

7.1 ACA

```

1 static const int MAXL=200005,SIGMA=26; //
2 MAXL: sum of length in dictionary

```

```

2 // Link: suffix link, next: DFA link, n: #
  // of nodes, tag: ID of str ends here
3 // next and link always exist, others exist
  // iff values != -1.
4 // nocc: next occurrence, first node with
  // tag != -1 along suffix link
5 int n, dep[MAXL], link[MAXL], next[MAXL][
  SIGMA];
6 int trie[MAXL][SIGMA], tag[MAXL], nocc[MAXL
  ];
7
8 int new_node(int p) {
9     // Add you init if recording more values.
10    dep[n] = n == 0 ? 0 : dep[p] + 1;
11    link[n] = tag[n] = nocc[n] = -1;
12    for (int i = 0; i < SIGMA; i++) {
13        next[n][i] = 0;
14        trie[n][i] = -1;
15    }
16    return n++;
17 }
18 void build(vector<string> &dict) {
19     // Some init should be written in new_node
    , 0(N*SIGMA).
20    n = 0;
21    new_node(0);
22    for (int i = 0; i < dict.size(); i++) {
23        int v = 0;
24        for (char ch : dict[i]) {
25            int to = ch - 'a'; // CHANGE THIS !!
26            if (trie[v][to] == -1) {
27                trie[v][to] = next[v][to] = new_node
                    (v);
28            }
29            v = trie[v][to];
30        }
31        tag[v] = i;
32    }
33
34    queue<int> Q;
35    link[0] = 0;
36    Q.push(0);
37    while (!Q.empty()) {
38        int v = Q.front(); Q.pop();
39        for (int to = 0; to < SIGMA; to++) {
40            if (trie[v][to] != -1) {
41                int u = trie[v][to];
42                link[u] = v == 0 ? 0 : next[link[v]
                    ][to];
43                nocc[u] = tag[link[u]] != -1 ? link[
                    u] : nocc[link[u]];
44                for (int j = 0; j < SIGMA; j++) {
45                    if (trie[u][j] == -1) {
46                        next[u][j] = next[link[u]]
                            [j];
47                    }
48                }
49                Q.push(u);
50            }
51        }
52    }
53 }

```

7.2 hash

```

1 #define MAXN 1000000
2 #define mod 1073676287
3 /*mod 必須要是質數*/
4 typedef long long T;
5 char s[MAXN+5];
6 T h[MAXN+5]; /*hash陣列*/
7 T h_base[MAXN+5]; /*h_base[n]=(prime^n)%mod*/
8 void hash_init(int len, T prime){
9     h_base[0]=1;
10    for (int i=1; i<=len; ++i){
11        h[i]=(h[i-1]*prime+s[i-1])%mod;
12        h_base[i]=(h_base[i-1]*prime)%mod;
13    }
14 }
15 T get_hash(int l, int r){ /*閉區間寫法·設編號
    為0 ~ Len-1*/
16    return (h[r+1]-(h[l]*h_base[r-l+1])%mod+
17        mod)%mod;

```

7.3 KMP

```

1 vector<int> lps; // Longest prefix suffix,
  0-based
2 int match(const string &text, const string &
  pat) {
3     /* Init is included */
4     lps.resize(pat.size());
5     /* DP */
6     lps[0]=0;
7     for (int i=1; i<pat.size(); i++) {
8         int len=lps[i-1];
9         while (len>0 && pat[len]!=pat[i]) len=lps
            [len-1];
10        lps[i] = pat[len]==pat[i] ? len+1 : 0;
11    }
12    /* Match */
13    int i = 0, j = 0;
14    while (i < text.size() && j < pat.size()) {
15        if (text[i] == pat[j]) i++, j++;
16        else if (j == 0) i++;
17        else j = lps[j-1];
18    }
19    if (j == pat.size()) return i - j;
20    return -1;
21 }

```

7.4 manacher

```

1 vector<int> d1(n); // Max len of palindrome
  centered at s[i]
2 for (int i = 0, l = 0, r = -1; i < n; i++) {
3     int k = (i > r) ? 1 : min(d1[l + r - i],
        r - i + 1);

```

```

4     while (0 <= i - k && i + k < n && s[i -
        k] == s[i + k]) {
5         k++;
6     }
7     d1[i] = k--;
8     if (i + k > r) {
9         l = i - k;
10        r = i + k;
11    }
12 }
13 vector<int> d2(n); // Max len of centered
  at "gap" before s[i]
14 for (int i = 0, l = 0, r = -1; i < n; i++) {
15     int k = (i > r) ? 0 : min(d2[l + r - i +
        1], r - i + 1);
16     while (0 <= i - k - 1 && i + k < n && s[
        i - k - 1] == s[i + k]) {
17         k++;
18     }
19     d2[i] = k--;
20     if (i + k > r) {
21         l = i - k - 1;
22         r = i + k;
23     }
24 }

```

7.5 minimal_string_rotation

```

1 int min_string_rotation(const string &s){
2     int n=s.size(), i=0, j=1, k=0;
3     while(i<n&&j<n&&k<n){
4         int t=s[(i+k)%n]-s[(j+k)%n];
5         ++k;
6         if(t){
7             if(t>0)i+=k;
8             else j+=k;
9             if(i==j)++j;
10            k=0;
11        }
12    }
13    return min(i,j); //最小循環表示法起始位置
14 }

```

7.6 reverseBWT

```

1 const int MAXN = 305, MAXC = 'Z';
2 int ranks[MAXN], tots[MAXC], first[MAXC];
3 void rankBWT(const string &bw){
4     memset(ranks, 0, sizeof(int)*bw.size());
5     memset(tots, 0, sizeof(tots));
6     for (size_t i=0; i<bw.size(); ++i)
7         ranks[i] = tots[ bw[i] ]++;
8 }
9 void firstCol(){
10    memset(first, 0, sizeof(first));
11    int totc = 0;
12    for (int c='A'; c<='Z'; ++c){
13        if (!tots[c]) continue;
14        first[c] = totc;

```

```

15        totc += tots[c];
16    }
17 }
18 string reverseBwt(string bw, int begin){
19     rankBWT(bw), firstCol();
20     int i = begin; //原字串最後一個元素的位置
21     string res;
22     do{
23         char c = bw[i];
24         res = c + res;
25         i = first[ bw[i] ] + ranks[i];
26     } while (i != begin);
27     return res;
28 }

```

7.7 SA

```

1 /* rank: inverse sa */
2 /* MAXL: Maximum length of string, lcp[i]:
  LCP(sa[i], sa[i-1]) */
3 string text;
4 int sa[MAXL], isa[MAXL], lcp[MAXL], cnt[MAXL
  +ALPHA];
5 void build(const vector<int> &text) {
6     text = _text + '\0'; // Must add this,
    must >= 0
7     int sz = text.size(), lim = ALPHA; //
    Takes ALPHA time, note when #TC is
    large
8     for (int i = 0; i < lim; i++) cnt[i] = 0;
9     for (int i = 0; i < sz; i++) cnt[ isa[i] =
        text[i] ]++;
10    for (int i = 1; i < lim; i++) cnt[i] +=
        cnt[i-1];
11    for (int i = sz-1; i >= 0; i--) sa[ --
        cnt[text[i]] ] = i;
12
13    lim = max(sz, ALPHA);
14    int *rk = isa, *nsa = lcp, *nrk = lcp;
15    for (int len = 1; len < sz; len <= 1) {
16        int num = 0;
17        for (int i = sz - len; i < sz; i++) nsa[
            num++] = i;
18        for (int i = 0; i < sz; i++) if (sa[i]
            >= len) nsa[num++] = sa[i] - len;
19
20        for (int i = 0; i < lim; i++) cnt[i] =
            0;
21        for (int i = 0; i < sz; i++) cnt[ rk[i]
            ]++;
22        for (int i = 1; i < lim; i++) cnt[i] +=
            cnt[i-1];
23        for (int i = sz-1; i >= 0; i--) sa[ --
            cnt[rk[nsa[i]]] ] = nsa[i];
24
25        num = 0;
26        nrk[sa[0]] = num++;
27        for (int i = 1; i < sz; i++) {
28            bool cond = rk[sa[i]] == rk[sa[i-1]]
                && sa[i] + len < sz;
29            cond = cond && sa[i-1] + len < sz &&
                rk[sa[i]+len] == rk[sa[i-1]+len];
30            if (cond) nrk[sa[i]] = num - 1;

```

```

31     else nrk[sa[i]] = num++;
32 }
33
34 if (num >= sz) break;
35 lim = num;
36 swap(rk, nrk);
37 nsa = nrk;
38 }
39 for (int i=0; i<sz; i++) isa[sa[i]] = i;
40
41 /* LCP */
42 int len = 0;
43 lcp[0] = 0; // Undefined
44 for (int i=0; i<sz; i++) {
45     if (isa[i] == 0) continue;
46     len = max(0, len-1);
47     int j = sa[isa[i]-1];
48     while (text[i+len] == text[j+len]) len
49         ++;
50     lcp[isa[i]] = len;
51 }

```

7.8 Z

```

1 void z_alg(char *s, int len, int *z){
2     int l=0, r=0;
3     z[0]=len;
4     for(int i=1; i<len; ++i){
5         z[i]=i>r?0:(i-l+z[i-l]<z[l]?z[i-l]:r-i
6             +1);
7         while(i+z[i]<len&&s[i+z[i]]==s[z[i]])++z
8             [i];
9         if(i+z[i]-1>r)r=i+z[i]-1, l=i;
10    }
11 }

```

8 Tarjan

8.1 dominator_tree

```

1 struct dominator_tree{
2     static const int MAXN=5005;
3     int n; // 1-base
4     vector<int> suc[MAXN], pre[MAXN];
5     int fa[MAXN], dfn[MAXN], id[MAXN], Time;
6     int semi[MAXN], idom[MAXN];
7     int anc[MAXN], best[MAXN]; // disjoint set
8     vector<int> dom[MAXN]; // dominator_tree
9     void init(int _n){
10         n=_n;
11         for(int i=1; i<=n; ++i) suc[i].clear(), pre[
12             i].clear();
13     }
14     void add_edge(int u, int v){
15         suc[u].push_back(v);
16         pre[v].push_back(u);

```

```

17 void dfs(int u){
18     dfn[u]=++Time, id[Time]=u;
19     for(auto v:suc[u]){
20         if(dfn[v]) continue;
21         dfs(v), fa[dfn[v]]=dfn[u];
22     }
23 }
24 int find(int x){
25     if(x==anc[x]) return x;
26     int y=find(anc[x]);
27     int semi[best[x]]>semi[best[anc[x]]]best
28         [x]=best[anc[x]];
29     return anc[x]=y;
30 }
31 void tarjan(int r){
32     Time=0;
33     for(int t=1; t<=n; ++t){
34         dfn[t]=idom[t]=0; // u=r 或是 u無法到達r時
35         idom[id[u]]=0
36         dom[t].clear();
37         anc[t]=best[t]=semi[t]=t;
38     }
39     dfs(r);
40     for(int y=Time; y>=2; --y){
41         int x=fa[y], idy=id[y];
42         for(auto z:pre[idy]){
43             if(!(z=dfn[z])) continue;
44             find(z);
45             semi[y]=min(semi[y], semi[best[z]]);
46         }
47         dom[semi[y]].push_back(y);
48         anc[y]=x;
49         for(auto z:dom[x]){
50             find(z);
51             idom[z]=semi[best[z]]<x?best[z]:x;
52         }
53         dom[x].clear();
54     }
55     for(int u=2; u<=Time; ++u){
56         if(idom[u]!=semi[u]) idom[u]=idom[idom[
57             u]];
58         dom[id[idom[u]]].push_back(id[u]);
59     }
60 }
61 } dom;

```

8.2 橋連通分量

```

1 #define N 1005
2 struct edge{
3     int u, v;
4     bool is_bridge;
5     edge(int u=0, int v=0):u(u), v(v), is_bridge
6         (0){}
7 };
8 vector<edge> E;
9 vector<int> G[N]; // 1-base
10 int low[N], vis[N], Time;
11 int bcc_id[N], bridge_cnt, bcc_cnt; // 1-base
12 int st[N], top; // BCC用
13 void add_edge(int u, int v){
14     G[u].push_back(E.size());

```

```

14     E.emplace_back(u, v);
15     G[v].push_back(E.size());
16     E.emplace_back(v, u);
17 }
18 void dfs(int u, int re=-1) { // u當前點, re為u連
19     接前一個點的邊
20     int v;
21     low[u]=vis[u]=++Time;
22     st[top++]=u;
23     for(int e:G[u]){
24         v=E[e].v;
25         if(!vis[v]){
26             dfs(v, e^1); // e^1 反向邊
27             low[u]=min(low[u], low[v]);
28             if(vis[u]<low[v]){
29                 E[e].is_bridge=E[e^1].is_bridge=1;
30                 ++bridge_cnt;
31             }
32         } else if(vis[v]<vis[u]&&e!=re){
33             low[u]=min(low[u], vis[v]);
34         }
35     }
36     if(vis[u]==low[u]){ // 處理BCC
37         ++bcc_cnt; // 1-base
38         do bcc_id[v=st[--top]]=bcc_cnt; // 每個點
39             所在的BCC
40         while(v!=u);
41     }
42 }
43 void bcc_init(int n){
44     Time=bcc_cnt=bridge_cnt=top=0;
45     E.clear();
46     for(int i=1; i<=n; ++i){
47         G[i].clear();
48         vis[i]=bcc_id[i]=0;
49     }

```

8.3 雙連通分量 & 割點

```

1 #define N 1005
2 vector<int> G[N]; // 1-base
3 vector<int> bcc[N]; // 存每塊雙連通分量的點
4 int low[N], vis[N], Time;
5 int bcc_id[N], bcc_cnt; // 1-base
6 bool is_cut[N]; // 是否為割點
7 int st[N], top;
8 void dfs(int u, int pa=-1) { // u當前點, pa父親
9     int t, child=0;
10     low[u]=vis[u]=++Time;
11     st[top++]=u;
12     for(int v:G[u]){
13         if(!vis[v]){
14             dfs(v, u), ++child;
15             low[u]=min(low[u], low[v]);
16             if(vis[u]<=low[v]){
17                 is_cut[u]=1;
18                 bcc[++bcc_cnt].clear();
19                 do{
20                     bcc_id[t=st[--top]]=bcc_cnt;
21                     bcc[bcc_cnt].push_back(t);
22                 } while(t!=v);

```

```

23         bcc_id[u]=bcc_cnt;
24         bcc[bcc_cnt].push_back(u);
25     }
26     } else if(vis[v]<vis[u]&&v!=pa) // 反向邊
27         low[u] = min(low[u], vis[v]);
28     } // u是dfs樹的根要特判
29     if(pa==-1&&child<2) is_cut[u]=0;
30 }
31 void bcc_init(int n){
32     Time=bcc_cnt=top=0;
33     for(int i=1; i<=n; ++i){
34         G[i].clear();
35         is_cut[i]=vis[i]=bcc_id[i]=0;
36     }
37 }

```

9 Tree

9.1 HLD

```

1 // In this template value is on the edge,
2 // everything is 1-based
3 int N;
4 vector<Edge> G[MAXN+5];
5
6 // Preprocess info, setup in dfs1
7 int heavy[MAXN+5], pa_w[MAXN+5], sz[MAXN+5];
8 int pa[MAXN+5], dep[MAXN+5], recorder[MAXN
9     +5]; // Which node record edge i.
10
11 // HLD info, setup in build, 1-based
12 // pos: position of node i in seg tree.
13 // head: For NODE i, points to head of the
14 // chain.
15 int chain_no, border, pos[MAXN+5], head[MAXN
16     +5];
17
18 void dfs1(int v, int p) {
19     pa[v] = p;
20     sz[v] = 1;
21     dep[v] = dep[p] + 1;
22     heavy[v] = -1;
23
24     for (const Edge &e : G[v]) {
25         if (e.to == p) continue;
26         dfs1(e.to, v);
27         pa_w[e.to] = e.w;
28         recorder[e.id] = e.to;
29         sz[v] += sz[e.to];
30         if (heavy[v] == -1 || sz[e.to] > sz[
31             heavy[v]]) {
32             heavy[v] = e.to;
33         }
34     }
35 }
36
37 void build(int v, int chain_head) {
38     pos[v] = ++border;
39     head[v] = chain_head;
40     tree.update(pos[v], pa_w[v], 1, N, 1);

```



```

36 if (heavy[v] != -1) build(heavy[v],
37   chain_head);
38 for (const Edge &e : G[v]) {
39   if (e.to == pa[v] || e.to == heavy[v]
40     ]) continue;
41   build(e.to, e.to);
42 }
43
44 void init_HLD() {
45   /* Only init used data, be careful. */
46   /* Does not init G!!!! */
47   border = dep[1] = pa_w[1] = 0;
48   dfs1(1, 1);
49   build(1, 1);
50 }
51
52 int query_up(int a, int b) {
53   int ans = 0;
54   while (head[a] != head[b]) {
55     if (dep[head[a]] < dep[head[b]]) swap(
56       a, b);
57     ans = max(ans, tree.query(pos[head[a]
58       ], pos[a], 1, N, 1));
59     a = pa[head[a]];
60   }
61   if (a == b) return ans;
62   if (dep[a] < dep[b]) swap(a, b);
63   // Query range is pos[b] if value is on
64   // node.
65   ans = max(ans, tree.query(pos[b] + 1,
66     pos[a], 1, N, 1));
67   return ans;
68 }

```

9.2 treeDC

```

1 int get_size(int v, int p) {
2   sz[v] = 1;
3   for (int to : G[v]) {
4     if (to != p && !vis[to]) {
5       get_size(to, v);
6       sz[v] += sz[to];
7     }
8   }
9   return sz[v];
10 }
11
12 void find_cent(int v, int p, int &cent, int
13   S) {
14   int big = S - sz[v];
15   for (int to : G[v]) {
16     if (!vis[to] && to != p) {
17       big = max(big, sz[to]);
18       find_cent(to, v, cent, S);
19     }
20   }
21   maxs[v] = big;
22   if (cent == -1 || big < maxs[cent]) {
23     cent = v;
24   }
25 }
26
27 void dfs(int v, int p, int d, vector<int> &
28   sub) {
29   dep[v] = d;
30   sub.push_back(v);
31   for (int to : G[v]) {
32     if (!vis[to] && to != p) {
33       dfs(to, v, d + 1, sub);
34     }
35   }
36 }
37
38 LL solve(int v, int l, int r) {
39   // # unordered (x, y), l <= dist(x, y) <=
40   // r, in tree of v.
41   int S = get_size(v, v), root = -1;
42   find_cent(v, v, root, S);
43   vis[root] = 1;
44
45   LL res = 0;
46   tree.add(0, 1); // ***** tree MUST be 0-
47   // based RSQ
48   vector<int> all;
49   for (int to : G[root]) {
50     if (!vis[to]) {
51       vector<int> sub;
52       dfs(to, root, 1, sub);
53       for (int u : sub) {
54         all.push_back(u);
55         if (r - dep[u] >= 0) {
56           res += tree.get(r - dep[u]);
57         }
58         if (l - 1 - dep[u] >= 0) {
59           res -= tree.get(l - 1 - dep[u]);
60         }
61       }
62       for (int u : sub) {
63         tree.add(dep[u], 1);
64       }
65     }
66   }
67   tree.add(0, -1);
68   for (int u : all) {
69     tree.add(dep[u], -1);
70   }
71   all.clear();
72   all.shrink_to_fit();
73
74   for (int to : G[root]) {
75     if (!vis[to]) {
76       res += solve(to, l, r);
77     }
78   }
79   return res;
80 }

```

10 others

10.1 vimrc

```

1 se ai nu ru cul mouse=a
2 se cin et ts=2 sw=2 sts=2
3 colo desert
4 se gfn=Monospace\ 14

```

11 zformula

11.1 formula

11.1.1 formula.txt

- 若多項式 $f(x)$ 有有理根 P/Q (P, Q 互質), 則 P 必為常數項 a_0 之因數, Q 必為領導係數 a_n 之因數
- 滿足 $\text{ceil}(n/i)=k$ 之最大 i :

- INF, if $k=1$
- $n/(k-1)-1$, else if $k-1$ 整除 n
- $n/(k-1)$, else

- 滿足 $\text{floor}(n/i)=k$ 之最大 i : $\text{floor}(n/k)$
- 尤拉函數: $\phi(n)=n$ 乘上所有 $(1-1/p)$ · 對 n 之所有質因數 p
- 尤拉定理: $a^{\phi(n)} \equiv 1 \pmod{n}$, a, n 互質
- 尤拉降冪: $a^b = a^{b \bmod \phi(n) + \phi(n)} \pmod{n}$, $b > \phi(n)$, 不必互質
- 次方同餘定理: $a^k \bmod p = (a \bmod p)^{(k \bmod p-1) \cdot p}$ 是質數
- Modulo inverse: $\text{inv}[i] = -\text{floor}(p/i) * \text{inv}[p \bmod i] \pmod{p}$
- 中國剩餘定理: $x = A_i \pmod{m_i}$, m_i 互質, $M_i =$ 所有 m 的乘積/ m_i , $T_i = M_i^{-1} \pmod{m_i}$, 則 $x = \text{sigma}(M_i * T_i * A_i) \pmod{M}$
- 枚舉擴展歐幾里得之解: 若 x_0, y_0 為 $a*x + b*y = k$ 之一組解, 則 $x = x_0 + t*b/\text{gcd}(a, b)$, $y = y_0 + t*a/\text{gcd}(a, b)$ 亦為解, t 為整數
- $\text{Sigma}\{i : \text{gcd}(i, n) = 1 \text{ and } i \text{ in } [1, n]\} = n * \phi(n)/2$ for $n > 1$
- $\text{Sigma}\{i * r^i : i \text{ in } [1, n]\} = (n * r * (n+1) - r * (r^n - 1)/(r-1))/(r-1)$
- 投擲正面機率 p 之硬幣 n 次 · 正面偶數次機率: $0.5 + 0.5 * (1 - 2p)^n$
- 分式拆分: $(a-b)/(ab) = 1/b - 1/a$
- 最大獨立集: 點的集合 · 其內點不相鄰
- 最小點覆蓋: 點的集合 · 所有邊都被覆蓋
- 最大匹配: 邊的集合 · 其內邊不共用點
- 最小邊覆蓋: 邊的集合 · 所有點都被覆蓋
- 最大獨立集 + 最小點覆蓋 = V (數值)
- 最大匹配 + 最小邊覆蓋 = V (數值)
- 最大匹配 = 最大流 (directed, 二分圖)
- 最大匹配 = 最小點覆蓋 (二分圖)
- 最小點覆蓋 + 最小邊覆蓋 = V (數值, 二分圖)
- 二分圖帶權最小點覆蓋 = 對左邊的點 v 連 $\text{cap}(\text{src}, v) = w(v)$ 之邊 · 右邊每個 v 連 $\text{cap}(v, \text{tgt}) = w(v)$ 之邊 · 每條邊 (u, v) 連 $\text{cap}(u, v) = \text{INF}$ · 皆有向 · 最大流即為所求。
- 一般圖帶權最小邊覆蓋 = (將原圖每個 $w(u, v)$ 改為 $w'(u, v) = c(u) + c(v) - w(u, v)$) · 所求為新圖之最大權匹配 + $\text{sigma}\{c(v)\}$ · $c(v)$ 為點 v 連到的最小 edge 權重。

- 一矩陣 A 所有 eigen value 之合 = 對角線合
- 一矩陣 A 所有 eigen value 之積 = $\det(A)$
- 三角形 ABC , 對邊長 abc :
- $\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$, $s = \text{周長}/2$
- $a/\sin A = b/\sin B = c/\sin C = 2R$, R 為外接圓半徑
- 內接圓半徑 = $2 * \text{area} / (a+b+c)$
- 外接圓半徑 = $abc / 4 * \text{area}$
- 球缺體積, h 為高, 且 $h \leq R$: $PI * h^2 * (R - h/3)$
- 枚舉 submask: for (int s=m; s; s=(s-1)&m) // Take care of ZERO after loop
- 某些質數: 54018521, 370248451, 6643838879, 119218851371, 5600748293801, 39916801, 479001599, 87178291199, 8589935681, 433494437, 2971215073

11.1.2 Pick 公式

給定頂點坐標均是整點的簡單多邊形 · 面積 = 內部格點數 + 邊上格點數/2-1

11.1.3 圖論

- 對於平面圖 · $F = E - V + C + 1$ · C 是連通分量數
- 對於平面圖 · $E \leq 3V - 6$
- 對於連通圖 G · 最大獨立點集的大小設為 $I(G)$ · 最大匹配大小設為 $M(G)$ · 最小點覆蓋設為 $C_v(G)$ · 最小邊覆蓋設為 $C_e(G)$ · 對於任意連通圖:

- $I(G) + C_v(G) = |V|$
- $M(G) + C_e(G) = |V|$

- 對於連通二分圖:

- $I(G) = C_v(G)$
- $M(G) = C_e(G)$

- 不相交環覆蓋: 每個 v 拆 v_{in}, v_{out} , 存在 iff. 二分完美匹配存在, 最小邊權環覆蓋 = 最小完美匹配
- vertex disjoint DAG path cover (蓋住所有點): 每個 v 拆 v_{in}, v_{out} , 原圖 $|V|-1$ 最大二分匹配 | 即為所求
- 可相交 DAG path cover: 每個 v 對他走到的所有點 u 連一條邊, 轉為 disjoint. (轉換後所有中途點母須存在)
- max anti-chain over partial order (最大 subset 任兩人不可比較): 建出 partial order 的 transitive closure, disjoint DAG path cover 即為所求。
- 最大權閉合圖:

- $C(u, v) = \infty, (u, v) \in E$
- $C(S, v) = W_v, W_v > 0$
- $C(v, T) = -W_v, W_v < 0$
- $\text{ans} = \sum_{W_v > 0} W_v - \text{flow}(S, T)$

- 最大密度子圖:

- 求 $\max \left(\frac{w_e + w_v}{|V'|} \right), e \in E', v \in V'$
- $U = \sum_{v \in V} 2W_v + \sum_{e \in E} W_e$
- $C(u, v) = W_{(u, v)}, (u, v) \in E$ · 雙向邊
- $C(S, v) = U, v \in V$
- $D_u = \sum_{(u, v) \in E} W_{(u, v)}$
- $C(v, T) = U + 2g - D_v - 2W_v, v \in V$

- (g) 二分搜 g :
 $l = 0, r = U, eps = 1/n^2$
 if $((U \times |V| - flow(S, T))/2 > 0)$ $l = mid$
 else $r = mid$
 (h) $ans = \min_cut(S, T)$
 (i) $|E| = 0$ 要特殊判斷

11. 弦圖：

- (a) 點數大於 3 的環都要有一條弦
 (b) 完美消除序列從後往前依次給每個點染色，給每個點染上可以染的最小顏色
 (c) 最大團大小 = 色數
 (d) 最大獨立集：完美消除序列從前往後能選就選
 (e) 最小圖覆蓋：最大獨立集的點和他延伸的邊構成
 (f) 區間圖是弦圖
 (g) 區間圖的完美消除序列：將區間按造又端點由小到大排序
 (h) 區間圖染色：用線段樹做

11.1.4 dinic 特殊圖複雜度

- 單位流： $O\left(\min\left(V^{3/2}, E^{1/2}\right)E\right)$
- 二分圖： $O\left(V^{1/2}E\right)$

11.1.5 0-1 分數規劃

$x_i \in \{0, 1\}$ · x_i 可能會有其他限制 · 求 $\max\left(\frac{\sum B_i x_i}{\sum C_i x_i}\right)$

- $D(i, g) = B_i - g \times C_i$
- $f(g) = \sum D(i, g)x_i$
- $f(g) = 0$ 時 g 為最佳解 · $f(g) < 0$ 沒有意義
- 因為 $f(g)$ 單調可以二分搜 g
- 或用 Dinkelbach 通常比較快

```

1 binary_search(){
2   while(r-l>eps){
3     g=(l+r)/2;
4     for(i:所有元素)D[i]=B[i]-g*C[i];//D(i,g)
5     找出一組合法x[i]使f(g)最大;
6     if(f(g)>0) l=g;
7     else r=g;
8   }
9   Ans = r;
10 }
11 Dinkelbach(){
12   g=任意狀態(通常設為0);
13   do{
14     Ans=g;
15     for(i:所有元素)D[i]=B[i]-g*C[i];//D(i,g)
16     找出一組合法x[i]使f(g)最大;
17     p=0,q=0;
18     for(i:所有元素)
19       if(x[i]p+=B[i],q+=C[i];
20     g=p/q;//更新解 · 注意q=0的情況
21   }while(abs(Ans-g)>EPS);
22   return Ans;
23 }
```

11.1.6 學長公式

- $\sum_{d|n} \phi(n) = n$
- $g(n) = \sum_{d|n} f(d) \Rightarrow f(n) = \sum_{d|n} \mu(d) \times g(n/d)$
- Harmonic series $H_n = \ln(n) + \gamma + 1/(2n) - 1/(12n^2) + 1/(120n^4)$
- $\gamma = 0.57721566490153286060651209008240243104215$
- 格雷碼 $= n \oplus (n >> 1)$
- $SG(A + B) = SG(A) \oplus SG(B)$
- 選轉矩陣 $M(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$

11.1.7 基本數論

- $\sum_{d|n} \mu(n) = [n == 1]$
- $g(m) = \sum_{d|m} f(d) \Leftrightarrow f(m) = \sum_{d|m} \mu(d) \times g(m/d)$
- $\sum_{i=1}^n \sum_{j=1}^m$ 互質數量 $= \sum \mu(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$
- $\sum_{i=1}^n \sum_{j=1}^m lcm(i, j) = n \sum_{d|n} d \times \phi(d)$

11.1.8 排組公式

- k 卡特蘭 $\frac{C_{kn}^{kn}}{n(k-1)+1} \cdot C_m^n = \frac{n!}{m!(n-m)!}$
- $H(n, m) \cong x_1 + x_2 \dots + x_n = k, num = C_{n+k-1}^n$
- Stirling number of 2^{nd} , n 人分 k 組方法數目
 - $S(0, 0) = S(n, n) = 1$
 - $S(n, 0) = 0$
 - $S(n, k) = kS(n-1, k) + S(n-1, k-1)$
- Bell number, n 人分任意多組方法數目
 - $B_0 = 1$
 - $B_n = \sum_{i=0}^n S(n, i)$
 - $B_{n+1} = \sum_{k=0}^n C_k^n B_k$
 - $B_{p+n} \equiv B_n + B_{n+1} \pmod{p}$, p is prime
 - $B_{p+m+n} \equiv mB_n + B_{n+1} \pmod{p}$, p is prime
 - From $B_0 : 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975$
- Derangement, 錯排, 沒有人在自己位置上
 - $D_n = n!(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} \dots + (-1)^n \frac{1}{n!})$
 - $D_n = (n-1)(D_{n-1} + D_{n-2}), D_0 = 1, D_1 = 0$
 - From $D_0 : 1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496$

6. Binomial Equality

- $\sum_k \binom{r}{m+k} \binom{s}{n-k} = \binom{r+s}{m+n}$
- $\sum_k \binom{r}{m+k} \binom{s}{n+k} = \binom{l+s}{l+m+n}$
- $\sum_k \binom{r}{m+k} \binom{s+k}{n} (-1)^k = (-1)^{l+m} \binom{s-m}{n-l}$
- $\sum_{k \leq l} \binom{l-k}{m} \binom{s}{k-n} (-1)^k = \frac{(-1)^{l+m} \binom{s-m-1}{l-n-m}}{(-1)^{l+m} \binom{s-m-1}{l-n-m}}$
- $\sum_{0 \leq k \leq l} \binom{l-k}{m} \binom{q+k}{n} = \binom{l+q+1}{m+n+1}$
- $\binom{r}{k} = (-1)^k \binom{k-r-1}{k}$

- $\binom{r}{m} \binom{m}{k} = \binom{r}{k} \binom{r-k}{m-k}$
- $\sum_{k \leq n} \binom{r+k}{k} = \binom{r+n+1}{n}$
- $\sum_{0 \leq k \leq n} \binom{k}{m} = \binom{n+1}{m+1}$
- $\sum_{k \leq m} \binom{m+r}{k} x^k y^{m-k} = \sum_{k \leq m} \binom{m+r}{k} (-x)^k (x+y)^{m-k}$

11.1.9 幕次, 幕次和

- $a^b \% P = a^{b \% \varphi(P) + \varphi(P)}, b \geq \varphi(P)$
- $1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$
- $1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$
- $1^5 + 2^5 + 3^5 + \dots + n^5 = \frac{n^6}{6} + \frac{n^5}{2} + \frac{5n^4}{12} - \frac{n^2}{12}$
- $0^k + 1^k + 2^k + \dots + n^k = P(k), P(k) = \frac{(n+1)^{k+1} - \sum_{i=0}^{k-1} C_i^{k+1} P(i)}{k+1}, P(0) = n+1$
- $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1} \sum_{k=0}^n C_k^{n+1} B_k m^{n+1-k}$
- $\sum_{j=0}^m C_j^{m+1} B_j = 0, B_0 = 1$
- 除了 $B_1 = -1/2$ · 剩下的奇數項都是 0
- $B_2 = 1/6, B_4 = -1/30, B_6 = 1/42, B_8 = -1/30, B_{10} = 5/66, B_{12} = -691/2730, B_{14} = 7/6, B_{16} = -3617/510, B_{18} = 43867/798, B_{20} = -174611/330,$

11.1.10 Burnside's lemma

- $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$
- $X^g = t^{c(g)}$
- G 表示有幾種轉法 · X^g 表示在那種轉法下 · 有幾種是會保持對稱的 · t 是顏色數 · $c(g)$ 是循環節不動的面數。
- 正立方體塗三顏色 · 轉 0 有 3^6 個元素不變 · 轉 90 有 6 種 · 每種有 3^3 不變 · 180 有 3×3^4 · 120(角) 有 8×3^2 · 180(邊) 有 6×3^3 · 全部 $\frac{1}{24} (3^6 + 6 \times 3^3 + 3 \times 3^4 + 8 \times 3^2 + 6 \times 3^3) = \frac{24}{57}$

11.1.11 Count on a tree

- Rooted tree: $s_{n+1} = \frac{1}{n} \sum_{i=1}^n (i \times a_i \times \sum_{j=1}^{\lfloor n/i \rfloor} a_{n+1-i \times j})$
- Unrooted tree:
 - Odd: $a_n - \sum_{i=1}^{n/2} a_i a_{n-i}$
 - Even: $Odd + \frac{1}{2} a_{n/2} (a_{n/2} + 1)$
- Spanning Tree (for n labeled vertices)
 - 完全圖 $n^n - 2$
 - 完全二分圖 $K_{n,m}: m^{n-1} \times n^{m-1}$
 - 一般圖 (Kirchhoff's theorem) $M[i][i] = degree(V_i), M[i][j] = -1, \text{if have } E(i, j), 0 \text{ if no edge. delete any one row and col in } A, ans = det(A)$

11.1.12 Horrible bugs

- int 開成 bool 導致計算出錯或其他型別開錯導致 cin 出錯
- cmp 寫成非嚴格偏序
- 該開 multiset 不小心開成 set
- 你以為 sort 只要排一維, 其實兩維都要排
- 分成多個地方 output, 忘記設定 precision 或沒 return
- 把 N 向上補成 2 的倍數或改動常數, 但是 N 會用在別的地方
- l, r, 題目沒有說 $l \leq r$ 之類的
- 填入無限大或負數之類的湊成整數倍, 結果被拿來當 array id
- 感覺都沒錯, 生一些有相同物的 case 或邊界條件

11.2 java

11.2.1 文件操作

```

1 import java.io.*;
2 import java.util.*;
3 import java.math.*;
4 import java.text.*;
5
6 public class Main{
7
8   public static void main(String args[]){
9     throws FileNotFoundException,
10    IOException
11    Scanner sc = new Scanner(new FileReader(
12      "a.in"));
13    PrintWriter pw = new PrintWriter(new
14      FileWriter("a.out"));
15    int n,m;
16    n=sc.nextInt();//读入下一个INT
17    m=sc.nextInt();
18
19    for(ci=1; ci<=c; ++ci){
20      pw.println("Case #"+ci+": easy for
21        output");
22    }
23
24    pw.close();//关闭流并释放 · 这个很重要 ·
25    否则是没有输出的
26    sc.close();//关闭流并释放
27  }
```

11.2.2 优先队列

```

1 PriorityQueue queue = new PriorityQueue( 1,
2   new Comparator(){
3     public int compare( Point a, Point b ){
4       if( a.x < b.x || a.x == b.x && a.y < b.y )
5         return -1;
6       else if( a.x == b.x && a.y == b.y )
7         return 0;
```

```
7 | else return 1;
8 | }
9 | });
```

11.2.3 Map

```
1 | Map map = new HashMap();
2 | map.put("sa","dd");
3 | String str = map.get("sa").toString;
4 |
5 | for(Object obj : map.keySet()){
6 |     Object value = map.get(obj );
7 | }
```

11.2.4 sort

```
1 | static class cmp implements Comparator{
2 |     public int compare(Object o1,Object o2){
3 |         BigInteger b1=(BigInteger)o1;
4 |         BigInteger b2=(BigInteger)o2;
5 |         return b1.compareTo(b2);
6 |     }
7 | }
8 | public static void main(String[] args)
9 |     throws IOException{
10 |     Scanner cin = new Scanner(System.in);
11 |     int n;
12 |     n=cin.nextInt();
13 |     BigInteger[] seg = new BigInteger[n];
14 |     for (int i=0;i<n;i++)
15 |         seg[i]=cin.nextBigInteger();
16 |     Arrays.sort(seg,new cmp());
17 | }
```

11.2.5 utility

```
1 | BigInteger x,y,z; z=x.divide(y); // multiply
2 |     , subtract, add, mod, z=x.negate()
3 | Arrays.sort(arr, 0, size);
4 | BigInteger dp[][] = new BigInteger[n][n];
5 | Math.min(x, y) // Math.max
6 | Integer.toString(5);
7 | x=BigInteger.valueOf(5);
8 | while (fin.hasNext()) x = fin.nextBigInteger
9 |     ();
```

ACM ICPC TEAM REFERENCE - POLARSHEEP

Contents

1 Data_Structure	1	3.1 circle	2	6.3 EXT_GCD	9	9 Tree	12
1.1 hull_dynamic	1	3.2 convex_hull	3	6.4 FFT	9	9.1 HLD	12
1.2 persistent_treap	1	3.3 geometry	3	6.5 find_real_root	9	9.2 treeDC	13
1.3 Treap	1	3.4 KD_TREE	4	6.6 FWT	9	10 others	13
1.4 undo_disjoint_set	1	3.5 smallest_circle	5	6.7 gauss_elimination	9	10.1 vimrc	13
1.5 整體二分	1	3.6 最近點對	5	6.8 LL_mul	9	11 zformula	13
2 Flow	1	4 Graph	5	6.9 Lucas	9	11.1 formula	13
2.1 DFSflow	1	4.1 3989_ 穩定婚姻	5	6.10 Matrix	9	11.1.1 formula.txt	13
2.2 Dinic	2	4.2 blossom	5	6.11 Miller_Rabin	10	11.1.2 Pick 公式	13
2.3 min_cost_flow	2	4.3 Eulerian_cycle	5	6.12 mod_log	10	11.1.3 圖論	13
3 Geometry	2	4.4 KM	6	6.13 NTT	10	11.1.4 dinic 特殊圖複雜度	14
		4.5 MaximumClique	6	6.14 pollard	10	11.1.5 0-1 分數規劃	14
		4.6 MinimumMeanCycle	6	7 String	10	11.1.6 學長公式	14
		4.7 SAT2	6	7.1 ACA	10	11.1.7 基本數論	14
		4.8 一般圖最小權完美匹配	6	7.2 hash	11	11.1.8 排組公式	14
		4.9 全局最小割	7	7.3 KMP	11	11.1.9 冪次, 冪次和	14
		4.10 平面圖判定	7	7.4 manacher	11	11.1.10 Burnside's lemma	14
		4.11 最小斯坦納樹 DP	7	7.5 minimal_string_rotation	11	11.1.11 Count on a tree	14
		4.12 最小樹形圖_朱劉	7	7.6 reverseBWT	11	11.1.12 Horrible bugs	14
		4.13 穩定婚姻模板	7	7.7 SA	11	11.2 java	14
		5 Linear_Programming	8	7.8 Z	12	11.2.1 文件操作	14
		5.1 simplex	8	8 Tarjan	12	11.2.2 優先队列	14
		6 Number_Theory	8	8.1 dominator_tree	12	11.2.3 Map	15
		6.1 basic	8	8.2 橋連通分量	12	11.2.4 sort	15
		6.2 bit_set	9	8.3 雙連通分量 & 割點	12	11.2.5 utility	15