

# NCTU\_TaNoShiI

## Contents

<b>1 Basic</b>	<b>1</b>	<b>8.4 MinLexicographicalRotate</b>	<b>23</b>
1.1 Default	1	<b>8.5 ZvaluePalindromes</b>	<b>23</b>
<b>2 DataStructure</b>	<b>1</b>	<b>8.6 SuffixArray</b>	<b>23</b>
2.1 PersistentTreap	1	<b>8.7 Zvalue</b>	<b>23</b>
2.2 Pbds Kth	2	<b>9 Math</b>	<b>23</b>
2.3 PbdsHeap	2	9.1 MillerRabin	23
2.4 Heavy-LightDecomposition	2	9.2 Simplex	23
2.5 KDtree	3	9.3 Theorem	24
2.6 LCT	3	9.4 Prime	24
<b>3 Flow</b>	<b>4</b>	9.5 FFT	24
3.1 Minmunwieghtmatchclique	4	9.6 Crt Solve2	25
3.2 CostFlow	5	9.7 FWT	25
3.3 MincutTree	5	9.8 Extgcd	25
3.4 Dinic	5	9.9 Pollard'sRho	25
3.5 GeneralGraphmatch	6	<b>10 monge</b>	<b>25</b>
3.6 KM	6	<b>11 四心</b>	<b>25</b>
3.7 SWmincut	7	<b>12 Runge-Kutta</b>	<b>25</b>
<b>4 Geometry</b>	<b>7</b>	<b>13 Householder Matrix</b>	<b>25</b>
4.1 Circleintersection	7	<b>14 Simpson's-rule</b>	<b>25</b>
4.2 Fermat's Point	7	<b>1 Basic</b>	
4.3 Pointoperators	8	<b>1.1 Default</b>	
4.4 3DConvexHull	8	<pre>1 #include&lt;bits/stdc++.h&gt; 2 #define mp(a,b) make_pair((a),(b)) 3 #define pii pair&lt;int,int&gt; 4 #define pdd pair&lt;double,double&gt; 5 #define pll pair&lt;LL,LL&gt; 6 #define pb(x) push_back(x) 7 #define x first 8 #define y second 9 #define sqr(x) ((x)*(x)) 10 #define EPS 1e-6 11 #define mii map&lt;int,int&gt; 12 #define MEM(x) memset(x,0,sizeof(x)) 13 #define MEMS(x) memset(x,-1,sizeof(x)) 14 #define pi 3.14159265359 15 //#define INF 0x7fffffff 16 #define IOS ios_base::sync_with_stdio(0); cin.tie(0) 17 #define N 300005 18 using namespace std; 19 typedef long long LL;</pre>	
4.5 Halfplaneintersection	8	<b>2 DataStructure</b>	
4.6 ConvexHull	9	<b>2.1 PersistentTreap</b>	
4.7 Triangulation	9	<pre>1 const int MEM = 16000004; 2 struct Treap { 3     static Treap nil, mem[MEM], *pmem; 4     Treap *l, *r; 5     char val; 6     int size; 7     Treap () : l(&amp;nil), r(&amp;nil), size(0) {} 8     Treap (char _val) : 9         l(&amp;nil), r(&amp;nil), val(_val), size(1) {} 10 } Treap::nil, Treap::mem[MEM], *Treap::pmem = Treap:: 11 nil; 12 int size(const Treap *t) { return t-&gt;size; } 13 void pull(Treap *t) { 14     if (!size(t)) return; 15     t-&gt;size = size(t-&gt;l) + size(t-&gt;r) + 1; 16 } 17 Treap* merge(Treap *a, Treap *b) { 18     if (!size(a)) return b; 19     if (!size(b)) return a; 20     Treap *t;</pre>	
4.8 K-closet Pair	10		
4.9 MCC	11		
4.10 LineIntersection	11		
4.11 PointToLine	11		
<b>5 Graph</b>	<b>11</b>		
5.1 Planar	11		
5.2 MMC	13		
5.3 SomeTheroem	14		
5.4 Dominator	14		
5.5 DMST	14		
5.6 SCC	15		
5.7 GeneralGraphMaximunValueMatch	15		
5.8 Stable Marriage	16		
5.9 BCCvertex	17		
5.10 MaxClique	17		
5.11 BCCedge	18		
<b>6 JAVA</b>	<b>18</b>		
6.1 Big Integer	18		
6.2 Prime	18		
<b>7 Other</b>	<b>18</b>		
7.1 Dp Optimizer	18		
7.2 Annealing	19		
7.3 DLX	19		
7.4 MahattanMST	19		
7.5 MoOnTree	20		
7.6 Det	21		
<b>8 String</b>	<b>21</b>		
8.1 AC	21		
8.2 SuffixAutomata	21		
8.3 Palindromic Tree	22		

```

21     if (rand() % (size(a) + size(b)) < size(a)) {
22         t = new (Treap::pmem++) Treap(*a);
23         t->r = merge(a->r, b);
24     } else {
25         t = new (Treap::pmem++) Treap(*b);
26         t->l = merge(a, b->l);
27     }
28     pull(t);
29     return t;
30 }
31 void split(Treap *t, int k, Treap *&a, Treap *&b) {
32     if (!size(t)) a = b = &Treap::nil;
33     else if (size(t->l) + 1 <= k) {
34         a = new (Treap::pmem++) Treap(*t);
35         split(t->r, k - size(t->l) - 1, a->r, b);
36         pull(a);
37     } else {
38         b = new (Treap::pmem++) Treap(*t);
39         split(t->l, k, a, b->l);
40         pull(b);
41     }
42 }
43 int nv;
44 Treap *rt[50005];
45 void print(const Treap *t) {
46     if (!size(t)) return;
47     print(t->l);
48     cout << t->val;
49     print(t->r);
50 }
51 int main(int argc, char** argv) {
52     IOS;
53     rt[nv=0] = &Treap::nil;
54     Treap::pmem = Treap::mem;
55     int Q, cmd, p, c, v;
56     string s;
57     cin >> Q;
58     while (Q--) {
59         cin >> cmd;
60         if (cmd == 1) {
61             // insert string s after position p
62             cin >> p >> s;
63             Treap *tl, *tr;
64             split(rt[nv], p, tl, tr);
65             for (int i=0; i<s.size(); i++)
66                 tl = merge(tl, new (Treap::pmem++) Treap
67                     (s[i]));
68             rt[++nv] = merge(tl, tr);
69         } else if (cmd == 2) {
70             // remove c characters starting at
71             // position
72             Treap *tl, *tm, *tr;
73             cin >> p >> c;
74             split(rt[nv], p-1, tl, tm);
75             split(tm, c, tm, tr);
76             rt[++nv] = merge(tl, tr);
77         } else if (cmd == 3) {
78             // print c characters starting at
79             // position p, in version v
80             Treap *tl, *tm, *tr;
81             cin >> v >> p >> c;
82             split(rt[v], p-1, tl, tm);
83             split(tm, c, tm, tr);
84             print(tm);
85             cout << "n";
86         }
87     }
88     return 0;

```

## 2.2 Pbds Kth

```

1 #include <bits/extc++.h>
2 using namespace __gnu_pbds;
3 typedef tree<int, null_type, less<int>, rb_tree_tag,
4 tree_order_statistics_node_update> set_t;
5 int main()
6 {
7     // Insert some entries into s.
8     set_t s;
9     s.insert(12); s.insert(505);

```

```

10 // The order of the keys should be: 12, 505.
11 assert(*s.find_by_order(0) == 12);
12 assert(*s.find_by_order(3) == 505);
13 // The order of the keys should be: 12, 505.
14 assert(s.order_of_key(12) == 0);
15 assert(s.order_of_key(505) == 1);
16 // Erase an entry.
17 s.erase(12);
18 // The order of the keys should be: 505.
19 assert(*s.find_by_order(0) == 505);
20 // The order of the keys should be: 505.
21 assert(s.order_of_key(505) == 0);
22 }

```

## 2.3 PbdsHeap

```

1 #include <bits/extc++.h>
2 typedef __gnu_pbds::priority_queue<int> heap_t;
3 heap_t a, b;
4 int main() {
5     a.clear(); b.clear();
6     a.push(1); a.push(3);
7     b.push(2); b.push(4);
8     assert(a.top() == 3);
9     assert(b.top() == 4);
10    // merge two heap
11    a.join(b);
12    assert(a.top() == 4);
13    assert(b.empty());
14    return 0;
15 }

```

## 2.4 Heavy-LightDecomposition

```

1 #define N
2 void init(); // implement
3 int n, fa[N], belong[N], dep[N], sz[N], que[N];
4 int step, line[N], stPt[N], edPt[N];
5 vector<int> v[N], chain[N];
6 void DFS(int u) {
7     vector<int> &c = chain[belong[u]];
8     for (int i=c.size()-1; i>=0; i--) {
9         int v = c[i];
10        stPt[v] = step;
11        line[step++] = v;
12    }
13    for (int i=0; i<(int)c.size(); i++) {
14        u = c[i];
15        for (vector<int>::iterator it=v[u].begin();
16            it!=v[u].end(); it++) {
17            if (fa[u] == *it || (i && *it == c[i-1]))
18                continue;
19            DFS(*it);
20        }
21        edPt[u] = step-1;
22    }
23 }
24 void build_chain(int st) {
25     int fr, bk;
26     fr=bk=0; que[bk++] = 1; fa[st]=st; dep[st]=0;
27     while (fr < bk) {
28         int u=que[fr++];
29         for (vector<int>::iterator it=v[u].begin();
30             it!=v[u].end(); it++) {
31             if (*it == fa[u]) continue;
32             que[bk++] = *it;
33             dep[*it] = dep[u]+1;
34             fa[*it] = u;
35         }
36     }
37 }
38 for (int i=bk-1, u=pos; i>=0; i--) {
39     u = que[i]; sz[u] = 1; pos = -1;
40     for (vector<int>::iterator it=v[u].begin();
41         it!=v[u].end(); it++) {
42         if (*it == fa[u]) continue;
43         sz[u] += sz[*it];
44         if (pos==-1 || sz[*it]>sz[pos]) pos=*it;
45     }
46     if (pos == -1) belong[u] = u;
47     else belong[u] = belong[pos];
48     chain[belong[u]].pb(u);

```

```

44     }
45     step = 0;
46     DFS(st);
47 }
48 int getLCA(int u, int v){
49     while (belong[u] != belong[v]){
50         int a = chain[belong[u]].back();
51         int b = chain[belong[v]].back();
52         if (dep[a] > dep[b]) u = fa[a];
53         else v = fa[b];
54     }
55     return sz[u] >= sz[v] ? u : v;
56 }
57 vector<pii> getPathSeg(int u, int v){
58     vector<pii> ret1, ret2;
59     while (belong[u] != belong[v]){
60         int a = chain[belong[u]].back();
61         int b = chain[belong[v]].back();
62         if (dep[a] > dep[b]){
63             ret1.pb(mp(stPt[a], stPt[u]));
64             u = fa[a];
65         } else {
66             ret2.pb(mp(stPt[b], stPt[v]));
67             v = fa[b];
68         }
69     }
70     if (dep[u] > dep[v]) swap(u, v);
71     ret1.pb(mp(stPt[u], stPt[v]));
72     reverse(ret2.begin(), ret2.end());
73     ret1.insert(ret1.end(), ret2.begin(), ret2.end());
74     return ret1;
75 }
76 // Usage
77 void build(){
78     build_chain(1); //change root
79     init();
80 }
81 int get_answer(int u, int v){
82     int ret = -2147483647;
83     vector<pii> vec = getPathSeg(u, v);
84     for (vector<pii>::iterator it = vec.begin(); it !=
85         vec.end(); it++){
86         // check answer with segment [it.F, it.S]
87         return ret;
88     }

```

## 2.5 KDtree

```

1 struct KDTree {
2     struct Node {
3         int x, y, x1, y1, x2, y2;
4         int id, f;
5         Node *L, *R;
6     } tree[MXN];
7     int n;
8     Node *root;
9     long long dis2(int x1, int y1, int x2, int y2) {
10         long long dx = x1 - x2;
11         long long dy = y1 - y2;
12         return dx * dx + dy * dy;
13     }
14     static bool cmpx(Node& a, Node& b) { return a.x < b.x; }
15     static bool cmpy(Node& a, Node& b) { return a.y < b.y; }
16     void init(vector<pair<int, int>> ip) {
17         n = ip.size();
18         for (int i = 0; i < n; i++) {
19             tree[i].id = i;
20             tree[i].x = ip[i].first;
21             tree[i].y = ip[i].second;
22         }
23         root = build_tree(0, n - 1, 0);
24     }
25     Node* build_tree(int L, int R, int dep) {
26         if (L > R) return nullptr;
27         int M = (L + R) / 2;
28         tree[M].f = dep % 2;
29         nth_element(tree + L, tree + M, tree + R + 1, tree[M].f ?
30             cmpx : cmpy);
31         tree[M].x1 = tree[M].x2 = tree[M].x;

```

```

32         tree[M].y1 = tree[M].y2 = tree[M].y;
33         tree[M].L = build_tree(L, M - 1, dep + 1);
34         if (tree[M].L) {
35             tree[M].x1 = min(tree[M].x1, tree[M].L->
36                 x1);
37             tree[M].x2 = max(tree[M].x2, tree[M].L->
38                 x2);
39             tree[M].y1 = min(tree[M].y1, tree[M].L->
40                 y1);
41             tree[M].y2 = max(tree[M].y2, tree[M].L->
42                 y2);
43         }
44         tree[M].R = build_tree(M + 1, R, dep + 1);
45         if (tree[M].R) {
46             tree[M].x1 = min(tree[M].x1, tree[M].R->
47                 x1);
48             tree[M].x2 = max(tree[M].x2, tree[M].R->
49                 x2);
50             tree[M].y1 = min(tree[M].y1, tree[M].R->
51                 y1);
52             tree[M].y2 = max(tree[M].y2, tree[M].R->
53                 y2);
54         }
55         return tree + M;
56     }
57     int touch(Node* r, int x, int y, long long d2) {
58         long long dis = sqrt(d2) + 1;
59         if (x < r->x1 - dis || x > r->x2 + dis || y < r->y1 -
60             dis || y > r->y2 + dis)
61             return 0;
62         return 1;
63     }
64     void nearest(Node* r, int x, int y, int &mID,
65         long
66         long &md2) {
67         if (!r || !touch(r, x, y, md2)) return;
68         long long d2 = dis2(r->x, r->y, x, y);
69         if (d2 < md2 || (d2 == md2 && mID < r->id))
70             {
71                 mID = r->id;
72                 md2 = d2;
73             }
74         // search order depends on split dim
75         if ((r->f == 0 && x < r->x) ||
76             (r->f == 1 && y < r->y)) {
77             nearest(r->L, x, y, mID, md2);
78             nearest(r->R, x, y, mID, md2);
79         } else {
80             nearest(r->R, x, y, mID, md2);
81             nearest(r->L, x, y, mID, md2);
82         }
83     }
84     int query(int x, int y) {
85         int id = 1029384756;
86         long long d2 = 102938475612345678LL;
87         nearest(root, x, y, id, d2);
88         return id;
89     }
90 } tree;

```

## 2.6 LCT

```

1 const int MXN = 100005;
2 const int MEM = 100005;
3
4 struct Splay {
5     static Splay nil, mem[MEM], *pmem;
6     Splay *ch[2], *f;
7     int val, rev, size;
8     Splay() : val(-1), rev(0), size(0) {
9         f = ch[0] = ch[1] = &nil;
10     }
11     Splay(int _val) : val(_val), rev(0), size(1) {
12         f = ch[0] = ch[1] = &nil;
13     }
14     bool isr() {
15         return f->ch[0] != this && f->ch[1] != this;
16     }
17     int dir() {
18         return f->ch[0] == this ? 0 : 1;
19     }

```

```

20 void setCh(Splay *c, int d) {
21     ch[d] = c;
22     if (c != &nil) c->f = this;
23     pull();
24 }
25 void push() {
26     if (rev) {
27         swap(ch[0], ch[1]);
28         if (ch[0] != &nil) ch[0]->rev ^= 1;
29         if (ch[1] != &nil) ch[1]->rev ^= 1;
30         rev=0;
31     }
32 }
33 void pull() {
34     size = ch[0]->size + ch[1]->size + 1;
35     if (ch[0] != &nil) ch[0]->f = this;
36     if (ch[1] != &nil) ch[1]->f = this;
37 }
38 } Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay
    ::mem;
39 Splay *nil = &Splay::nil;
40
41 void rotate(Splay *x) {
42     Splay *p = x->f;
43     int d = x->dir();
44     if (!p->isr()) p->f->setCh(x, p->dir());
45     else x->f = p->f;
46     p->setCh(x->ch[!d], d);
47     x->setCh(p, !d);
48     p->pull(); x->pull();
49 }
50
51 vector<Splay*> splayVec;
52 void splay(Splay *x) {
53     splayVec.clear();
54     for (Splay *q=x; q=q->f) {
55         splayVec.push_back(q);
56         if (q->isr()) break;
57     }
58     reverse(begin(splayVec), end(splayVec));
59     for (auto it : splayVec) it->push();
60     while (!x->isr()) {
61         if (x->f->isr()) rotate(x);
62         else if (x->dir()==x->f->dir()) rotate(x->f),
            rotate(x);
63         else rotate(x), rotate(x);
64     }
65 }
66
67 Splay* access(Splay *x) {
68     Splay *q = nil;
69     for (; x!=nil; x=x->f) {
70         splay(x);
71         x->setCh(q, 1);
72         q = x;
73     }
74     return q;
75 }
76 void evert(Splay *x) {
77     access(x);
78     splay(x);
79     x->rev ^= 1;
80     x->push(); x->pull();
81 }
82 void link(Splay *x, Splay *y) {
83     // evert(x);
84     access(x);
85     splay(x);
86     evert(y);
87     x->setCh(y, 1);
88 }
89 void cut(Splay *x, Splay *y) {
90     // evert(x);
91     access(y);
92     splay(y);
93     y->push();
94     y->ch[0] = y->ch[0]->f = nil;
95 }
96
97 int N, Q;
98 Splay *vt[MXN];
99

```

```

100 int ask(Splay *x, Splay *y) {
101     access(x);
102     access(y);
103     splay(x);
104     int res = x->f->val;
105     if (res == -1) res=x->val;
106     return res;
107 }
108 int main(int argc, char** argv) {
109     scanf("%d%d", &N, &Q);
110     for (int i=1; i<=N; i++)
111         vt[i] = new (Splay::pmem++) Splay(i);
112     while (Q--) {
113         char cmd[105];
114         int u, v;
115         scanf("%s", cmd);
116         if (cmd[1] == 'i') {
117             scanf("%d%d", &u, &v);
118             link(vt[u], vt[v]);
119         } else if (cmd[0] == 'c') {
120             scanf("%d", &v);
121             cut(vt[1], vt[v]);
122         } else {
123             scanf("%d%d", &u, &v);
124             int res=ask(vt[u], vt[v]);
125             printf("%d\n", res);
126         }
127     }
128
129     return 0;
130 }

```

## 3 Flow

### 3.1 Minunwieghtmatchclique

```

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

```

```

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

```

### 3.2 CostFlow

```

1 struct CostFlow {
2     static const int MXN = 205;
3     static const long long INF = 102938475610293847
LL;
4     struct Edge {
5         int v, r;
6         long long f, c;
7         Edge(int a, int b, int _c, int d):v(a), r(b), f(
_c), c(d){
8         }
9     };
10    int n, s, t, prv[MXN], prvL[MXN], inq[MXN];
11    long long dis[MXN], fl, cost;
12    vector<Edge> E[MXN];
13    void init(int _n, int _s, int _t) {
14        n = _n; s = _s; t = _t;
15        for (int i=0; i<n; i++) E[i].clear();
16        fl = cost = 0;
17    }
18    void add_edge(int u, int v, long long f, long
long c)
19    {
20        E[u].pb(Edge(v, E[v].size(), f, c));
21        E[v].pb(Edge(u, E[u].size()-1, 0, -c));
22    }
23    pll flow() {
24        while (true) {
25            for (int i=0; i<n; i++) {
26                dis[i] = INF;
27                inq[i] = 0;
28            }
29            dis[s] = 0;
30            queue<int> que;
31            que.push(s);
32            while (!que.empty()) {
33                int u = que.front(); que.pop();
34                inq[u] = 0;
35                for (int i=0; i<E[u].size(); i++) {
36                    int v = E[u][i].v;
37                    long long w = E[u][i].c;
38                    if (E[u][i].f > 0 && dis[v] >
dis[u] + w) {
39                        prv[v] = u; prvL[v] = i;
40                        dis[v] = dis[u] + w;
41                        if (!inq[v]) {
42                            inq[v] = 1;
43                            que.push(v);
44                        }
45                    }
46                }
47            }
48            if (dis[t] == INF) break;
49            long long tf = INF;
50            for (int v=t, u, l; v!=s; v=u) {

```

```

51                u=prv[v]; l=prvL[v];
52                tf = min(tf, E[u][l].f);
53            }
54            for (int v=t, u, l; v!=s; v=u) {
55                u=prv[v]; l=prvL[v];
56                E[u][l].f -= tf;
57                E[v][E[u][l].r].f += tf;
58            }
59            cost += tf * dis[t];
60            fl += tf;
61        }
62        return {fl, cost};
63    }
64 }flow;

```

### 3.3 MincutTree

```

1 set<int> temp;
2 int Vis[3005];
3 int cvis[3005];
4 void dfs(int n){
5     Vis[n]=1;
6     for(auto it=v[n].begin(); it!=v[n].end(); it++){
7         if(val[n][*it]>flow[n][*it]&&!Vis[*it]){
8             dfs(*it);
9             if(cvis[*it])
10                 temp.insert(*it);
11         }
12     }
13 }
14 int n;
15 int dc(set<int> s, int flag){
16     if(s.size()==1)
17         return *s.begin();
18     for(int i=0; i<n; i++){
19         for(auto it=v[i].begin(); it!=v[i].end(); it++){
20             flow[i][*it]=0;
21         }
22         for(auto it=s.begin(); it!=s.end(); it++){
23             cvis[*it]=1;
24         }
25         int res=Flow(*s.begin(), *s.rbegin());
26         MEM(Vis);
27         dfs(*s.begin());
28         temp.insert(*s.begin());
29         for(auto it=s.begin(); it!=s.end(); it++){
30             cvis[*it]=0;
31         }
32         set<int> s1, s2;
33         swap(s1, temp);
34         temp.clear();
35         for(auto it=s1.begin(); it!=s1.end(); it++){
36             s.erase(*it);
37             swap(s2, s);
38             int x=dc(s1, 0);
39             int y=dc(s2, 1);
40             vt[x].pb(mp(y, res));
41             vt[y].pb(mp(x, res));
42             if(flag==0)
43                 return x;
44             else
45                 return y;
46         }
47     }

```

### 3.4 Dinic

```

1 struct Dinic{
2     static const int MXN = 10000;
3     struct Edge{ int v, f, re; Edge(int a, int b, int c)
:v(a), f(b), re(c){}};
4     int n, s, t, level[MXN];
5     vector<Edge> E[MXN];
6     void init(int _n, int _s, int _t){
7         n = _n; s = _s; t = _t;
8         for (int i=0; i<n; i++) E[i].clear();
9     }
10    void add_edge(int u, int v, int f){
11        E[u].pb(Edge(v, f, E[v].size()));
12        E[v].pb(Edge(u, 0, E[u].size()-1)); //direct
13    }
14    bool BFS(){
15        MEMS(level);

```



```

16     queue<int> que;
17     que.push(s);
18     level[s] = 0;
19     while (!que.empty()){
20         int u = que.front(); que.pop();
21         for (auto it : E[u]){
22             if (it.f > 0 && level[it.v] == -1){
23                 level[it.v] = level[u]+1;
24                 que.push(it.v);
25             }
26         }
27     }
28     return level[t] != -1;
29 }
30 int DFS(int u, int nf){
31     if (u == t) return nf;
32     int res = 0;
33     for (auto &it : E[u]){
34         if (it.f > 0 && level[it.v] == level[u
35 ]+1){
36             int tf = DFS(it.v, min(nf,it.f));
37             res += tf; nf -= tf; it.f -= tf;
38             E[it.v][it.re].f += tf;
39             if (nf == 0) return res;
40         }
41     }
42     if (!res) level[u] = -1;
43     return res;
44 }
45 int flow(int res=0){
46     while ( BFS() )
47         res += DFS(s,2147483647);
48     return res;
49 }flow;

```

### 3.5 GeneralGraphmatch

```

1 struct GenMatch { // 1-base
2     static const int MAXN = 505;
3     int V;
4     bool el[MAXN][MAXN];
5     int pr[MAXN];
6     bool inq[MAXN],inp[MAXN],inb[MAXN];
7     queue<int> qe;
8     int st,ed;
9     int nb;
10    int bk[MAXN],djs[MAXN];
11    int ans;
12    void init(int _V) {
13        V = _V;
14        MEM(el); MEM(pr);
15        MEM(inq); MEM(inp); MEM(inb);
16        MEM(bk); MEM(djs);
17        ans = 0;
18    }
19    void add_edge(int u, int v) {
20        el[u][v] = el[v][u] = 1;
21    }
22    int lca(int u,int v) {
23        memset(inp,0,sizeof(inp));
24        while(1) {
25            u = djs[u];
26            inp[u] = true;
27            if(u == st) break;
28            u = bk[pr[u]];
29        }
30        while(1) {
31            v = djs[v];
32            if(inp[v]) return v;
33            v = bk[pr[v]];
34        }
35        return v;
36    }
37    void upd(int u) {
38        int v;
39        while(djs[u] != nb) {
40            v = pr[u];
41            inb[djs[u]] = inb[djs[v]] = true;
42            u = bk[v];
43            if(djs[u] != nb) bk[u] = v;
44        }

```

```

45    }
46    void blo(int u,int v) {
47        nb = lca(u,v);
48        memset(inb,0,sizeof(inb));
49        upd(u); upd(v);
50        if(djs[u] != nb) bk[u] = v;
51        if(djs[v] != nb) bk[v] = u;
52        for(int tu = 1; tu <= V; tu++)
53            if(inb[djs[tu]]) {
54                djs[tu] = nb;
55                if(!inq[tu]){
56                    qe.push(tu);
57                    inq[tu] = 1;
58                }
59            }
60    }
61    void flow() {
62        memset(inq,false,sizeof(inq));
63        memset(bk,0,sizeof(bk));
64        for(int i = 1; i <= V;i++)
65            djs[i] = i;
66        while(qe.size()) qe.pop();
67        qe.push(st);
68        inq[st] = 1;
69        ed = 0;
70        while(qe.size()) {
71            int u = qe.front(); qe.pop();
72            for(int v = 1; v <= V; v++)
73                if(el[u][v] && (djs[u] != djs[v]) && (pr
74 [u] !=
75 v)) {
76                 if((v == st) || ((pr[v] > 0) && bk[
77 pr[v]] >
78 0))
79                     blo(u,v);
80                 else if(bk[v] == 0) {
81                     bk[v] = u;
82                     if(pr[v] > 0) {
83                         if(!inq[pr[v]]) qe.push(pr[v
84 ]));
85                     } else {
86                         ed = v;
87                         return;
88                     }
89                 }
90             }
91         }
92     void aug() {
93         int u,v,w;
94         u = ed;
95         while(u > 0) {
96             v = bk[u];
97             w = pr[v];
98             pr[v] = u;
99             pr[u] = v;
100            u = w;
101        }
102    }
103    int solve() {
104        memset(pr,0,sizeof(pr));
105        for(int u = 1; u <= V; u++)
106            if(pr[u] == 0) {
107                st = u;
108                flow();
109                if(ed > 0) {
110                    aug();
111                    ans ++;
112                }
113            }
114        return ans;
115    }
116 }gp;

```

### 3.6 KM

```

1 typedef pair<long long, long long> pll;
2 struct KM{
3     // Maximum Bipartite Weighted Matching (Perfect
4     Match)
5     static const int MXN = 650;
6     static const int INF = 2147483647; // long long

```

```

6   int n, match[MXN], vx[MXN], vy[MXN];
7   int edge[MXN][MXN], lx[MXN], ly[MXN], slack[MXN];
8   // ^^^^ long long
9   void init(int _n){
10      n = _n;
11      for (int i=0; i<n; i++){
12         for (int j=0; j<n; j++){
13            edge[i][j] = 0;
14        }
15    }
16    void add_edge(int x, int y, int w){ // long long
17        edge[x][y] = w;
18    }
19    bool DFS(int x){
20        vx[x] = 1;
21        for (int y=0; y<n; y++){
22            if (vy[y]) continue;
23            if (lx[x]+ly[y] > edge[x][y]){
24                slack[y] = min(slack[y], lx[x]+ly[y]
25                               ]-edge[x][y]);
26            } else {
27                vy[y] = 1;
28                if (match[y] == -1 || DFS(match[y])){
29                    match[y] = x;
30                    return true;
31                }
32            }
33        }
34        return false;
35    }
36    int solve(){
37        fill(match, match+n, -1);
38        fill(lx, lx+n, -INF);
39        fill(ly, ly+n, 0);
40        for (int i=0; i<n; i++){
41            for (int j=0; j<n; j++){
42                lx[i] = max(lx[i], edge[i][j]);
43            }
44            for (int i=0; i<n; i++){
45                fill(slack, slack+n, INF);
46                while (true){
47                    fill(vx, vx+n, 0);
48                    fill(vy, vy+n, 0);
49                    if (DFS(i)) break;
50                    int d = INF; // long long
51                    for (int j=0; j<n; j++){
52                        if (!vy[j]) d = min(d, slack[j]);
53                    }
54                    for (int j=0; j<n; j++){
55                        if (vx[j]) lx[j] -= d;
56                        if (vy[j]) ly[j] += d;
57                        else slack[j] -= d;
58                    }
59                }
60            }
61            int res=0;
62            for (int i=0; i<n; i++){
63                res += edge[match[i]][i];
64            }
65            return res;
66        }
67    }graph;

```

### 3.7 SWmincut

```

1   struct SW{ // 0(V^3)
2       static const int MXN = 514;
3       int n, vst[MXN], del[MXN];
4       int edge[MXN][MXN], wei[MXN];
5       void init(int _n){
6           n = _n;
7           MEM(edge);
8           MEM(del);
9       }
10      void add_edge(int u, int v, int w){
11          edge[u][v] += w;
12          edge[v][u] += w;
13      }
14      void search(int &s, int &t){
15          MEM(vst); MEM(wei);
16          s = t = -1;
17          while (true){
18              int mx=-1, cur=0;
19              for (int i=0; i<n; i++){

```

```

20              if (!del[i] && !vst[i] && mx<wei[i])
21                  cur = i, mx = wei[i];
22              if (mx == -1) break;
23              vst[cur] = 1;
24              s = t;
25              t = cur;
26              for (int i=0; i<n; i++){
27                  if (!vst[i] && !del[i]) wei[i] += edge[
28                      cur][i];
29              }
30          }
31          int solve(){
32              int res = 2147483647;
33              for (int i=0, x=y; i<n-1; i++){
34                  search(x, y);
35                  res = min(res, wei[y]);
36                  del[y] = 1;
37                  for (int j=0; j<n; j++){
38                      edge[x][j] = (edge[j][x] += edge[y][j]);
39                  }
40              }
41          }graph;

```

## 4 Geometry

### 4.1 Circleintersection

```

1   using ld = double;
2   vector<pdd> interCircle(pdd o1, double r1, pdd o2,
3   double r2) {
4       ld d2 = (o1 - o2) * (o1 - o2);
5       ld d = sqrt(d2);
6       if (d > r1+r2) return {};
7       pdd u = 0.5*(o1+o2) + ((r2*r2-r1*r1)/(2*d2))*(o1
8       -o2);
9       double A = sqrt((r1+r2+d) * (r1-r2+d) * (r1+r2-d)
10       * (-r1+r2+d));
11       pdd v = A / (2*d2) * pdd(o1.S-o2.S, -o1.F+o2.F);
12       return {u+v, u-v};

```

### 4.2 Fermat's Point

```

1   #define F(n) Fi(i,n)
2   #define Fi(i,n) Fl(i,0,n)
3   #define Fl(i,l,n) for(int i=(l);i<(int)(n);++i)
4   #include <bits/stdc++.h>
5   // #include <ext/pb_ds/assoc_container.hpp>
6   // #include <ext/pb_ds/priority_queue.hpp>
7   using namespace std;
8   // using namespace __gnu_pbds;
9   const double pi = acos(-1), eps = 1e-9;
10  const double st = sin(pi/3), ct = cos(pi/3);
11  struct point {
12      point(double x_ = 0, double y_ = 0): x(x_), y(y_)
13      {}
14      double x, y;
15      inline friend istream& operator>>(istream &is,
16      point &p) {
17          is >> p.x >> p.y;
18          return is;
19      }
20      inline friend ostream& operator<<(ostream &os,
21      const point &p) {
22          os << p.x << ' ' << p.y;
23          return os;
24      }
25  };
26  struct line {
27      line(double a_ = 0, double b_ = 0, double c_ = 0):
28      a(a_), b(b_), c(c_) {}
29      double a, b, c;
30      inline double calc(point p) {
31          return a*p.x+b*p.y;
32      }
33  };
34  inline double calc(double a, double b, point p) {
35      return a*p.x+b*p.y;

```

```

32 }
33 inline double dist2(point a, point b) {
34     return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
35 }
36 inline point rot(point o, point p) {
37     p.x -= o.x, p.y -= o.y;
38     return point(o.x+p.x*ct-p.y*st, o.y+p.x*st+p.y*ct);
39 }
40 inline line cln(point a, point b) {
41     return line(a.y-b.y, b.x-a.x, calc(a.y-b.y, b.x-a.x, a));
42 }
43 inline point ntse(line f, line g) {
44     double det = f.a*g.b-g.a*f.b, dx = f.c*g.b-g.c*f.b,
45             dy = f.a*g.c-g.a*f.c;
46     return point(dx/det, dy/det);
47 }
48 inline point fema(point a, point b, point c) {
49     double la = dist2(b, c), lb = dist2(a, c), lc = dist2(a, b);
50     double sa = sqrt(la), sb = sqrt(lb), sc = sqrt(lc);
51     if ((lb+lc-la)/(2.0*sb*sc) < -0.5 + eps) return a;
52     if ((la+lc-lb)/(2.0*sa*sc) < -0.5 + eps) return b;
53     if ((la+lb-lc)/(2.0*sa*sb) < -0.5 + eps) return c;
54     point t1 = rot(a, b), t2 = rot(b, a);
55     if (dist2(c, t1) < dist2(c, t2)) swap(t1, t2);
56     point s1 = rot(b, c), s2 = rot(c, b);
57     if (dist2(a, s1) < dist2(a, s2)) swap(s1, s2);
58     return ntse(cln(c, t1), cln(a, s1));
59 }
60 int main() {
61     ios_base::sync_with_stdio(false);
62     cin.tie(NULL);
63     point a, b, c;
64     cin >> a >> b >> c;
65     cout << setprecision(10) << fixed << fema(a, b, c) << '\n';
66 }

```

### 4.3 Pointoperators

```

1 #define x first
2 #define y second
3 #define cpdd const pdd
4 struct pdd : pair<double, double> {
5     using pair<double, double>::pair;
6     pdd operator + (cpdd &p) const {
7         return {x+p.x, y+p.y};
8     }
9     pdd operator - () const {
10         return {-x, -y};
11     }
12     pdd operator - (cpdd &p) const {
13         return (*this) + (-p);
14     }
15     pdd operator * (double f) const {
16         return {f*x, f*y};
17     }
18     double operator * (cpdd &p) const {
19         return x*p.x + y*p.y;
20     }
21 };
22 double abs(cpdd &p) { return hypot(p.x, p.y); }
23 double arg(cpdd &p) { return atan2(p.y, p.x); }
24 double cross(cpdd &p, cpdd &q) { return p.x*q.y - p.y*q.x; }
25 double cross(cpdd &p, cpdd &q, cpdd &o) { return cross(p-o, q-o); }
26 pdd operator * (double f, cpdd &p) { return p*f; }
27 //!! Not f*p !!

```

### 4.4 3DConvexHull

```

1 int flag[MXN][MXN];
2 struct Point {
3     ld x,y,z;
4     Point operator - (const Point &b) const {
5         return (Point){x-b.x,y-b.y,z-b.z};
6     }
7     Point operator * (const ld &b) const {
8         return (Point){x*b,y*b,z*b};
9     }
10     ld len() const { return sqrtl(x*x+y*y+z*z); }
11     ld dot(const Point &a) const {
12         return x*a.x+y*a.y+z*a.z;
13     }
14     Point operator * (const Point &b) const {
15         return (Point){y*b.z-b.y*z,z*b.x-b.z*x,x*b.y-b.x*y};
16     };
17 };
18 Point ver(Point a, Point b, Point c) {
19     return (b - a) * (c - a);
20 }
21 vector<Face> convex_hull_3D(const vector<Point> pt) {
22     int n = SZ(pt);
23     REP(i,n) REP(j,n) flag[i][j] = 0;
24     vector<Face> now;
25     now.push_back((Face){0,1,2});
26     now.push_back((Face){2,1,0});
27     int ftop = 0;
28     for (int i=3; i<n; i++){
29         ftop++;
30         vector<Face> next;
31         REP(j, SZ(now)) {
32             Face& f=now[j];
33             ld d=(pt[i]-pt[f.a]).dot(ver(pt[f.a], pt[f.b], pt[f.c]));
34             if (d <= 0) next.push_back(f);
35             if (d > 0) ff=ftop;
36             else if (d < 0) ff=-ftop;
37             flag[f.a][f.b] = flag[f.b][f.c] = flag[f.c][f.a] = ff;
38         }
39         REP(j, SZ(now)) {
40             Face& f=now[j];
41             if (flag[f.a][f.b] > 0 and flag[f.a][f.b] != flag[f.b][f.a])
42                 next.push_back((Face){f.a,f.b,i});
43             if (flag[f.b][f.c] > 0 and flag[f.b][f.c] != flag[f.c][f.b])
44                 next.push_back((Face){f.b,f.c,i});
45             if (flag[f.c][f.a] > 0 and flag[f.c][f.a] != flag[f.a][f.c])
46                 next.push_back((Face){f.c,f.a,i});
47             now=next;
48         }
49     }
50     return now;
51 }

```

### 4.5 Halfplaneintersection

```

1 typedef pdd Point;
2 typedef vector<Point> Polygon;
3 typedef pair<Point,Point> Line;
4 #define N 10
5 #define p1 first
6 #define p2 second
7 pdd operator-(const pdd &a,const pdd &b){
8     return mp(a.x-b.x,a.y-b.y);
9 }
10 pdd operator+(const pdd &a,const pdd &b){
11     return mp(a.x+b.x,a.y+b.y);
12 }

```



```

13 pdd operator*(const pdd &a, const double &b){
14     return mp(b*a.x, b*a.y);
15 }
16 double cross(Point a, Point b){
17     return a.x * b.y - a.y * b.x;
18 }
19 double cross(Point o, Point a, Point b){
20     return cross(a-o, b-o);
21 }
22 double cross(Line l, Point p){
23     return cross(l.p1, l.p2, p);
24 }
25 double arg(const pdd &a){
26     return atan2(a.y, a.x);
27 }
28 bool parallel(Line l1, Line l2){
29     return cross(l1.p2 - l1.p1, l2.p2 - l2.p1) < 1e
-8 && cross(l1.p2 - l1.p1, l2.p2 - l2.p1) > -1e
-8;
30 }
31 Point intersection(Line l1, Line l2){
32     Point& a1 = l1.p1, &a2 = l1.p2;
33     Point& b1 = l2.p1, &b2 = l2.p2;
34     Point a = a2 - a1, b = b2 - b1, s = b1 - a1;
35     return a1 + a * (cross(b, s) / cross(b, a));
36 }
37 bool cmp(Line l1, Line l2){
38     return arg(l1.p2 - l1.p1) < arg(l2.p2 - l2.p1);
39 }
40 Polygon halfplane_intersection(vector<Line> hp){
41     sort(hp.begin(), hp.end(), cmp);
42     int L = 0, R = 0;
43     vector<Line> l(N);
44     vector<Point> p(N);
45     l[R] = hp[0];
46     for (int i=1; i<hp.size(); i++)
47     {
48         while (L < R && cross(hp[i], p[R-1]) < 0) R
--;
49         while (L < R && cross(hp[i], p[L]) < 0) L
++;
50         l[++R] = hp[i];
51         if (parallel(l[R-1], hp[i]) &&
52             cross(l[R-1], hp[i].p1) > 0) l[R] = hp[i];
53     };
54     if (L < R) p[R-1] = intersection(l[R], l[R
-1]);
55     while (L < R && cross(l[L], p[R-1]) < 0) R--;
56     if (R-L <= 1) return Polygon(); //printf("?");
57     if (L < R) p[R] = intersection(l[L], l[R]);
58     Polygon ch;
59     for (int i=L; i<=R; i++) ch.push_back(p[i]);
60     ch.resize(unique(ch.begin(), ch.end()) - ch.
begin());
61     if (ch.size() > 1 && ch.front() == ch.back())
62         ch.pop_back();
63     return ch;
64 }
65 double cal(Polygon p){
66     if(p.empty())
67         return 0;
68     p.pb(*p.begin());
69     double ans=0;
70     for(int i=0; i<p.size()-1; i++){
71         ans+=p[i].x*p[i+1].y;
72         ans-=p[i].y*p[i+1].x;
73     }
74     ans/=2;
75     ans=abs(ans);
76     return ans;
77 }

```

## 4.6 ConvexHull

```

1 sort(p, p+n);
2 pii ans[N];
3 ans[0]=p[0];
4 int k=0;
5 int now=0;
6 for(int yy=0; yy<2; yy++){
7     for(int i=1; i<n; i++){

```

```

8         while(now!=k&&cross(ans[now].x, p[j].x, ans[now
-1].x)<0){
9             now--;
10        }
11        ans[++now]=p[i];
12    }
13    k=now;
14    reverse(p, p+n);
15 }

```

## 4.7 Triangulation

```

1 bool inCircle(pdd a, pdd b, pdd c, pdd d) {
2     b = b - a;
3     c = c - a;
4     d = d - a;
5     if (cross(b, c) < 0) swap(b, c);
6     double m[3][3] = {
7         {b.x, b.y, b*b},
8         {c.x, c.y, c*c},
9         {d.x, d.y, d*d}
10    };
11    double det = m[0][0] * (m[1][1]*m[2][2] - m
[1][2]*m
[2][1])
12    + m[0][1] * (m[1][2]*m[2][0] - m[1][0]*m
[2][2])
13    + m[0][2] * (m[1][0]*m[2][1] - m[1][1]*m
[2][0]);
14    return det < 0;
15 }
16 bool intersect(pdd a, pdd b, pdd c, pdd d) {
17     return cross(b, c, a) * cross(b, d, a) < 0 and
18     cross(d, a, c) * cross(d, b, c) < 0;
19 }
20 const double EPS = 1e-12;
21 struct Triangulation {
22     static const int MXN = 1e5+5;
23     int N;
24     vector<int> ord;
25     vector<pdd> pts;
26     set<int> E[MXN];
27     vector<vector<int>> solve(vector<pdd> p) {
28         N = SZ(p);
29         ord.resize(N);
30         for (int i=0; i<N; i++) {
31             E[i].clear();
32             ord[i] = i;
33         }
34         sort(ALL(ord), [&p](int i, int j) {
35             return p[i] < p[j];
36         });
37         pts.resize(N);
38         for (int i=0; i<N; i++) pts[i] = p[ord[i]];
39         go(0, N);
40         vector<vector<int>> res(N);
41         for (int i=0; i<N; i++) {
42             int o = ord[i];
43             for (auto x: E[i]) {
44                 res[o].PB(ord[x]);
45             }
46         }
47         return res;
48     }
49     void add_edge(int u, int v) {
50         E[u].insert(v);
51         E[v].insert(u);
52     }
53     void remove_edge(int u, int v) {
54         E[u].erase(v);
55         E[v].erase(u);
56     }
57     void go(int l, int r) {
58         int n = r - l;
59         if (n <= 3) {
60             for (int i=l; i<r; i++)
61                 for (int j=i+1; j<r; j++) add_edge(i, j);
62             return;
63         }
64         int md = (l+r)/2;
65         go(l, md);
66     }

```

```

70     go(md, r);
71     int il = l, ir = r-1;
72     while (1) {
73         int nx = -1;
74         for (auto i: E[il]) {
75             double cs = cross(pts[il], pts[i],
76 pts[
77                 ir]);
78             if (cs > EPS ||
79                 (abs(cs) < EPS and abs(pts[i]-pts[
80                 ir]) < abs(pts[il]-pts[ir]))) {
81                 nx = i;
82                 break;
83             }
84         }
85         if (nx != -1) {
86             il = nx;
87             continue;
88         }
89         for (auto i: E[ir]) {
90             double cs = cross(pts[ir], pts[i],
91 pts[
92                 il]);
93             if (cs < -EPS ||
94                 (abs(cs) < EPS and abs(pts[i]-pts[
95                 il]) < abs(pts[ir]-pts[il]))) {
96                 nx = i;
97                 break;
98             }
99         }
100         if (nx != -1) {
101             ir = nx;
102             } else break;
103     }
104     add_edge(il, ir);
105     while (1) {
106         int nx = -1;
107         bool is2 = false;
108         National Taiwan University
109         AcThPaUNpPuAmCmBkCfEsFmMdNoLr 19
110         for (int i: E[il]) {
111             if (cross(pts[il], pts[i], pts[ir])
112 < -
113             EPS and
114             (nx == -1 or inCircle(pts[il], pts[
115             ir], pts[nx], pts[i]))) nx = i;
116         }
117         for (int i: E[ir]) {
118             if (cross(pts[ir], pts[i], pts[il])
119 >
120             EPS and
121             (nx == -1 or inCircle(pts[il], pts[
122             ir], pts[nx], pts[i]))) nx = i,
123             is2 = 1;
124         }
125         if (nx == -1) break;
126         int a = il, b = ir;
127         if (is2) swap(a, b);
128         for (auto i: E[a]) {
129             if (intersect(pts[a], pts[i], pts[b]
130             ],
131             pts[nx])) {
132                 remove_edge(a, i);
133             }
134         }
135         if (is2) {
136             add_edge(il, nx);
137             ir = nx;
138         } else {
139             add_edge(ir, nx);
140             il = nx;
141         }
142     }
143 }
144 } tri;

```

#### 4.8 K-closet Pair

```

1 #define F(n) Fi(i,n)
2 #define Fi(i,n) Fl(i,0,n)
3 #define Fl(i,l,n) for(int i=(l);i<(int)(n);++i)
4 #include <bits/stdc++.h>

```

```

5 // #include <ext/pb_ds/assoc_container.hpp>
6 // #include <ext/pb_ds/priority_queue.hpp>
7 using namespace std;
8 // using namespace __gnu_pbds;
9 typedef long long ll;
10 struct point {
11     point(ll x_ = 0, ll y_ = 0): x(x_), y(y_) {} ll x
12     , y;
13     inline bool operator<(const point &e_) const {
14         return (x != e_.x ? x < e_.x : y < e_.y);
15     }
16     inline friend istream& operator>>(istream &is_,
17     point& e_) {
18         is_ >> e_.x >> e_.y;
19         return is_;
20     }
21 }
22 int k;
23 priority_queue<ll> PQ;
24 inline ll dist2(const point &e1, const point &e2) {
25     ll res = (e1.x-e2.x)*(e1.x-e2.x)+(e1.y-e2.y)*(e1.y
26     -e2.y);
27     PQ.push(res);
28     if (PQ.size() > k) {
29         PQ.pop();
30     }
31     return res;
32 }
33 #define N 500005
34 point p[N];
35 queue<point> Q;
36 ll closet_point(int l, int m, int r, ll delta2) {
37     ll xmid = p[m-1].x;
38     while (!Q.empty()) {
39         Q.pop();
40     }
41     for (int i = l, j = m ; i < m ; ++i) {
42         if ((p[i].x-xmid)*(p[i].x-xmid) >= delta2) {
43             continue;
44         }
45         while (j < r && p[j].y < p[i].y && (p[j].y-p[i].
46         y)*(p[j].y-p[i].y) < delta2) {
47             if ((p[j].x-xmid)*(p[j].x-xmid) < delta2) {
48                 Q.push(p[j]);
49             }
50             ++j;
51         }
52         while (!Q.empty() && Q.front().y < p[i].y && (Q.
53         front().y-p[i].y)*(Q.front().y-p[i].y) > delta2
54         ) {
55             Q.pop();
56         }
57         while (!Q.empty()) {
58             delta2 = min(delta2, dist2(p[i], Q.front()));
59             Q.pop();
60         }
61     }
62     return delta2;
63 }
64 ll find_distance(int l, int r) {
65     if (r - l <= 3000) {
66         ll ans = 0x3f3f3f3f3f3f3f3f;
67         for (int i = l ; i < r ; ++i)
68             for (int j = i+1 ; j < r ; ++j)
69                 ans = min(ans, dist2(p[i], p[j]));
70         return ans;
71     }
72     int m = (l+r)/2;
73     ll delta2 = min(find_distance(l, m), find_distance
74     (m, r));
75     return min(delta2, closet_point(l, m, r, delta2));
76 }
77 int main() {
78     ios_base::sync_with_stdio(false);
79     cin.tie(NULL);
80     int n;
81     cin >> n >> k;
82     F(n) cin >> p[i];
83     sort(p, p+n);
84     find_distance(0, n);
85     cout << PQ.top() << '\n';
86 }

```

## 4.9 MCC

```

1 struct Mcc{
2     // return pair of center and r^2
3     static const int MAXN = 1000100;
4     int n;
5     pdd p[MAXN],cen;
6     double r2;
7     void init(int _n, pdd _p[]){
8         n = _n;
9         memcpy(p,_p,sizeof(pdd)*n);
10    }
11    double sqr(double a){ return a*a; }
12    double abs2(pdd a){ return a*a; }
13    pdd center(pdd p0, pdd p1, pdd p2) {
14        pdd a = p1-p0;
15        pdd b = p2-p0;
16        double c1=abs2(a)*0.5;
17        double c2=abs2(b)*0.5;
18        double d = a.x*b.y-b.x*a.y;
19        double x = p0.x + (c1 * b.y - c2 * a.y) / d;
20        double y = p0.y + (a.x * c2 - b.x * c1) / d;
21        return pdd(x,y);
22    }
23    pair<pdd,double> solve(){
24        random_shuffle(p,p+n);
25        r2=0;
26        for (int i=0; i<n; i++){
27            if (abs2(cen-p[i]) <= r2) continue;
28            cen = p[i];
29            r2 = 0;
30            for (int j=0; j<i; j++){
31                if (abs2(cen-p[j]) <= r2) continue;
32                cen = 0.5 * (p[i]+p[j]);
33                r2 = abs2(cen-p[j]);
34                for (int k=0; k<j; k++){
35                    if (abs2(cen-p[k]) <= r2)
36                        continue;
37                    cen = center(p[i],p[j],p[k]);
38                    r2 = abs2(cen-p[k]);
39                }
40            }
41            return {cen,r2};
42        }
43    }mcc;

```

## 4.10 LineIntersection

```

1 pdd interPnt(pdd p1, pdd p2, pdd q1, pdd q2, bool &
2 res)
3 {
4     double f1 = cross(p2, q1, p1);
5     double f2 = -cross(p2, q2, p1);
6     double f = (f1 + f2);
7     if(fabs(f) < EPS) {
8         res = false;
9         return {};
10    }
11    res = true;
12    return (f2 / f) * q1 + (f1 / f) * q2;

```

## 4.11 PointToLine

```

1 double cal(const pii &a,const pii &b,const pii &c){
2     int hi=dot(mp(a.x-b.x,a.y-b.y),mp(c.x-b.x,c.y-b.y))
3     );
4     if(hi<=0){
5         return dis(a,b);
6     }
7     hi=dot(mp(a.x-c.x,a.y-c.y),mp(b.x-c.x,b.y-c.y));
8     if(hi<=0){
9         return dis(c,a);
10    }
11    if(b.x==c.x)
12        return abs(a.x-b.x);
13    if(b.y==c.y)
14        return abs(a.y-b.y);
15    double B=(double)(b.x-c.x)/(b.y-c.y);
16    double C=(double)(b.y*c.x-b.x*c.y)/(b.y-c.y);
17    return abs(-a.x+B*a.y+C)/sqrt(1+sqr(B));

```

## 5 Graph

### 5.1 Planar

```

1 //skydog
2 #include <iostream>
3 #include <cstdio>
4 #include <cstdlib>
5 #include <iomanip>
6
7 #include <vector>
8 #include <cstring>
9 #include <string>
10 #include <queue>
11 #include <deque>
12 #include <stack>
13 #include <map>
14 #include <set>
15
16 #include <utility>
17 #include <list>
18
19 #include <cmath>
20 #include <algorithm>
21 #include <cassert>
22 #include <bitset>
23 #include <complex>
24 #include <climits>
25 #include <functional>
26 using namespace std;
27
28 typedef long long ll;
29 typedef pair<int, int> ii;
30 typedef pair<ll, ll> ll;
31
32 #define mp make_pair
33 #define pb push_back
34
35 #define debug(x) cerr << #x << " = " << x << " "
36
37 const int N=400+1;
38
39 struct Planar
40 {
41     int n,m,hash[N],fa[N],deep[N],low[N],ecp[N];
42     vector<int> g[N],son[N];
43     set< pair<int,int> > SDlist[N],proots[N];
44     int nxt[N][2],back[N],rev[N];
45     deque<int> q;
46     void dfs(int u)
47     {
48         hash[u]=1; q.pb(u);
49         ecp[u]=low[u]=deep[u];
50         int v;
51         for (int i = 0; i < g[u].size(); ++i)
52             if(!hash[v=g[u][i]])
53             {
54                 fa[v]=u;
55                 deep[v]=deep[u]+1;
56                 dfs(v);
57                 low[u]=min(low[u],low[v]);
58                 SDlist[u].insert(mp(low[v],v));
59             }
60         else ecp[u]=min(ecp[u],deep[v]);
61         low[u]=min(low[u],ecp[u]);
62     }
63
64     int visited[N];
65
66     void addtree(int u,int t1,int v,int t2)
67     {
68         nxt[u][t1]=v; nxt[v][t2]=u;
69     }
70
71     void findnxt(int u,int v,int& u1,int& v1)
72     {
73         u1=nxt[u][v^1];
74         if(nxt[u1][0]==u) v1=0;
75         else v1=1;
76     }
77

```

```

78 void walkup(int u,int v)
79 {
80     back[v]=u;
81     int v1=v,v2=v,u1=1,u2=0,z;
82     for (;;)
83     {
84         if(hash[v1]==u || hash[v2]==u) break;
85         hash[v1]=u;hash[v2]=u; z=max(v1,v2);
86         if(z>n)
87         {
88             int p=fa[z-n];
89             if(p!=u)
90             {
91                 roots[p].insert(mp(-low[z-n], z
92 ));
93                 v1=p,v2=p,u1=0,u2=1;
94             }
95             else break;
96         }
97         else
98         {
99             findnxt(v1,u1,v1,u1);
100             findnxt(v2,u2,v2,u2);
101         }
102     }
103 }
104 int topstack;
105 pair<int,int> stack[N];
106 int outer(int u,int v)
107 {
108     return ecp[v]<deep[u] || (SDlist[v].size()
109 && SDlist[v].begin()->first<deep[u]);
110 }
111 int inside(int u,int v)
112 {
113     return roots[v].size()>0 || back[v]==u;
114 }
115 int active(int u,int v)
116 {
117     return inside(u,v) || outer(u,v);
118 }
119 void push(int a,int b)
120 {
121     stack[++topstack]=mp(a,b);
122 }
123 void mergestack()
124 {
125     int v1,t1,v2,t2,s,s1;
126     v1=stack[topstack].first;t1=stack[topstack].
127 second;
128     topstack--;
129     v2=stack[topstack].first;t2=stack[topstack].
130 second;
131     topstack--;
132     s=nxt[v1][t1^1];
133     s1=(nxt[s][1]==v1);
134     nxt[s][s1]=v2;
135     nxt[v2][t2]=s;
136     SDlist[v2].erase( make_pair(low[v1-n],v1-n)
137 );
138     roots[v2].erase( make_pair(-low[v1-n],v1) )
139 ;
140 }
141 void findnxtActive(int u,int t,int& v,int& w1,
142 int S)
143 {
144     findnxt(u,t,v,w1);
145     while(u!=v && !active(S,v))
146         findnxt(v,w1,v,w1);
147 }
148 void walkdown(int S,int u)
149 {

```

```

153     topstack=0;
154     int t1,v=S,w1,x2,y2,x1,y1,p;
155     for(t1=0;t1<2;++t1)
156     {
157         findnxt(S,t1^1,v,w1);
158         while(v!=S)
159         {
160             if(back[v]==u)
161             {
162                 while(topstack>0) mergestack();
163                 addtree(S,t1,v,w1); back[v]=0;
164             }
165             if(roots[v].size())
166             {
167                 push(v,w1);
168                 p=roots[v].begin()->second;
169                 findnxtActive(p,1,x1,y1,u);
170                 findnxtActive(p,0,x2,y2,u);
171                 if(active(u,x1) && !outer(u,x1))
172                     v=x1,w1=y1;
173                 else if(active(u,x2) && !outer(u
174 ,x2))
175                     v=x2,w1=y2;
176                 else if(inside(u,x1) || back[x1
177 ]==u)
178                     v=x1,w1=y1;
179                 else v=x2,w1=y2;
180                 push(p,v==x2);
181             }
182             else if(v>n || (ecp[v]>=deep[u] &&
183 !outer(u,v) ))
184                 findnxt(v,w1,v,w1);
185             else if(v<=n && outer(u,v) && !
186 topstack)
187             {
188                 addtree(S,t1,v,w1); break;
189             }
190             else break;
191         }
192     }
193 }
194 int work(int u)
195 {
196     int v;
197     for (int i = 0; i < g[u].size(); ++i)
198         if(fa[v=g[u][i]]==u)
199         {
200             son[u].push_back(n+v);
201             roots[n+v].clear();
202             addtree(n+v,1,v,0);
203             addtree(n+v,0,v,1);
204         }
205     for (int i = 0; i < g[u].size(); ++i)
206         if(deep[v=g[u][i]]>deep[u]+1)
207             walkup(u,v);
208     topstack=0;
209     for (int i = 0; i < son[u].size(); ++i)
210         walkdown(son[u][i], u);
211     for (int i = 0; i < g[u].size(); ++i)
212         if(deep[v=g[u][i]]>deep[u]+1 && back[v])
213             return 0;
214     return 1;
215 }
216 void init(int _n)
217 {
218     n = _n;
219     m = 0;
220     for(int i=1;i<=2*n;++i)
221     {
222         g[i].clear();
223         SDlist[i].clear();
224         son[i].clear();
225         roots[i].clear();
226         nxt[i][0]=nxt[i][1]=0;
227         fa[i]=0;
228         hash[i]=0;low[i]=ecp[i]=deep[i]=back[i
229 ]=0;
230         q.clear();
231     }

```

```

229 void add(int u, int v)
230 {
231     ++m;
232     g[u].pb(v); g[v].pb(u);
233 }
234 bool check_planar()
235 {
236     if(m>3*n-5)
237         return false;
238     // memset(hash,0,sizeof hash);
239     for(int i=1;i<=n;++i)
240         if(!hash[i])
241             {
242                 deep[i]=1;
243                 dfs(i);
244             }
245     memset(hash,0,sizeof hash);
246     //memset(hash, 0, (2*n+1)*sizeof(hash[0]));
247     // originally only looks at last n element
248     assert(q.size() == n);
249     while (!q.empty())
250     {
251         if (!work(q.back()))
252             return false;
253         q.pop_back();
254     }
255     return true;
256 }
257 } base, _new;
258 vector<ii> edges;
259 int n, m;
260 inline void build(int n, Planar &_new)
261 {
262     _new.init(n);
263     for (auto e : edges)
264         _new.add(e.first, e.second);
265 }
266 void end()
267 {
268     puts("-1");
269     exit(0);
270 }
271 bool vis[N];
272 const int maxp = 5;
273 int path[maxp], tp=0;
274 void dfs(int cur)
275 {
276     vis[cur] = true;
277     path[tp++] = cur;
278     if (tp == maxp)
279     {
280         auto it = lower_bound(base.g[cur].begin(), base.g[cur].end(), path[0]);
281         if (it != base.g[cur].end() && *it == path[0])
282         {
283             //a cycle
284             int x = n+1;
285             for (int i = 0; i < 5; ++i) edges.pb(mp(x, path[i]));
286             build(x, _new);
287             if (_new.check_planar())
288             {
289                 for (int i = 0; i < maxp; ++i)
290                     printf("%d%c", path[i], i==maxp-1?'\\n':' ');
291                 exit(0);
292             }
293             for (int i = 0; i < 5; ++i) edges.pb(mp(x, path[i]));
294             pop_back();
295         }
296         else
297         {
298             for (auto e : base.g[cur]) if (!vis[e]) dfs(e);
299             vis[cur] = false;
300             --tp;
301         }
302     }
303 }
304 int main()
305 {
306     scanf("%d %d", &n, &m);

```

```

305     if (n <= 4)
306     {
307         assert(false);
308         puts("0"); return 0;
309     }
310     for (int i = 0; i < m; ++i)
311     {
312         int u, v; scanf("%d %d", &u, &v);
313         edges.pb(mp(u, v));
314     }
315     build(n, base);
316     if (!base.check_planar()) end();
317     for (int i = 1; i <= n; ++i)
318         sort(base.g[i].begin(), base.g[i].end());
319     for (int i = 1; i <= n; ++i)
320         dfs(i);
321     end();
322 }
323 }

```

## 5.2 MMC

```

1 /* minimum mean cycle 最小平均值環*/
2 const int MXN = 16004;
3 const int MAXE = 1805;
4 const int MAXN = 35;
5 const double inf = 1029384756;
6 const double eps = 1e-6;
7 struct Edge {
8     int v,u;
9     double c;
10 };
11 int n,m,prv[MAXN][MAXN], prve[MAXN][MAXN], vst[MAXN][MAXN];
12 Edge e[MAXE];
13 vector<int> edgeID, cycle, rho;
14 double d[MAXN][MAXN];
15 inline void bellman_ford() {
16     for(int i=0; i<n; i++) d[0][i]=0;
17     for(int i=0; i<n; i++) {
18         fill(d[i+1], d[i+1]+n, inf);
19         for(int j=0; j<m; j++) {
20             int v = e[j].v, u = e[j].u;
21             if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {
22                 d[i+1][u] = d[i][v]+e[j].c;
23                 prv[i+1][u] = v;
24                 prve[i+1][u] = j;
25             }
26         }
27     }
28 }
29 double karp_mmc() {
30     // returns inf if no cycle, mmc otherwise
31     double mmc=inf;
32     int st = -1;
33     bellman_ford();
34     for(int i=0; i<n; i++) {
35         double avg=-inf;
36         for(int k=0; k<n; k++) {
37             if(d[n][i]<inf-eps) avg=max(avg,d[n][i]-d[k][i]);
38             else avg=max(avg,inf);
39         }
40         if (avg < mmc) tie(mmc, st) = tie(avg, i);
41     }
42     MEM(vst); edgeID.clear(); cycle.clear(); rho.clear();
43     for (int i=n; !vst[st]; st=prv[i--][st]) {
44         vst[st]++;
45         edgeID.pb(prve[i][st]);
46         rho.pb(st);
47     }
48     while (vst[st] != 2) {
49         int v = rho.back(); rho.pop_back();
50         cycle.pb(v);
51         vst[v]++;
52     }
53     reverse(edgeID.begin(),edgeID.end());
54     edgeID.resize(cycle.size());
55     return mmc;
56 }

```



```
57 }
```

### 5.3 SomeTheroem

```
1 /*
2 General graph
3 |maximum independent set|+|minimum vertex cover|=|V|
4 |maximum independent edge|+|minimum edge cover|=|V|
5 ||
6 Max_match
7 Bipartite graph
8 |Maximun independent set|=|Minimun edge cover|
9 |Maximun independent edge|=|Minimun vertex cover|
10 |Maximun Independent set|+|Minimun vertex cover|=|V|
11 + +
12 |Maximun Independent edge|+|Minimun edge cover|=|V|
13 || ||
14 |V| |V|
15 */
```

### 5.4 Dominator

```
1 struct DominatorTree{
2 static const int MAXN = 200010;
3 int n,s;
4 vector<int> g[MAXN],pred[MAXN];
5 vector<int> cov[MAXN];
6 int dfn[MAXN],nfd[MAXN],ts;
7 int par[MAXN];
8 int sdom[MAXN],idom[MAXN];
9 int mom[MAXN],mn[MAXN];
10
11 inline bool cmp(int u,int v) { return dfn[u] < dfn[v]; }
12
13 int eval(int u) {
14 if(mom[u] == u) return u;
15 int res = eval(mom[u]);
16 if(cmp(sdom[mn[mom[u]]],sdom[mn[u]]))
17 mn[u] = mn[mom[u]];
18 return mom[u] = res;
19 }
20
21 void init(int _n, int _s) {
22 n = _n;
23 s = _s;
24 REP1(i,1,n) {
25 g[i].clear();
26 pred[i].clear();
27 idom[i] = 0;
28 }
29 }
30 void add_edge(int u, int v) {
31 g[u].push_back(v);
32 pred[v].push_back(u);
33 }
34 void DFS(int u) {
35 ts++;
36 dfn[u] = ts;
37 nfd[ts] = u;
38 for(int v:g[u]) if(dfn[v] == 0) {
39 par[v] = u;
40 DFS(v);
41 }
42 }
43 void build() {
44 ts = 0;
45 REP1(i,1,n) {
46 dfn[i] = nfd[i] = 0;
47 cov[i].clear();
48 mom[i] = mn[i] = sdom[i] = i;
49 }
50 DFS(s);
51 for (int i=ts; i>=2; i--) {
52 int u = nfd[i];
53 if(u == 0) continue;
54 for(int v:pred[u]) if(dfn[v]) {
55 eval(v);
56 if(cmp(sdom[mn[v]],sdom[u])) sdom[u] = sdom[mn[v]];
57 }
58 }
```

```
58 cov[sdom[u]].push_back(u);
59 mom[u] = par[u];
60 for(int w:cov[par[u]]) {
61 eval(w);
62 if(cmp(sdom[mn[w]],par[u])) idom[w] = mn[w];
63 else idom[w] = par[u];
64 }
65 cov[par[u]].clear();
66 }
67 REP1(i,2,ts) {
68 int u = nfd[i];
69 if(u == 0) continue;
70 if(idom[u] != sdom[u]) idom[u] = idom[idom[u]];
71 }
72 }
73 }dom;
```

### 5.5 DMST

```
1 struct zhu_liu{
2 static const int MAXN=1100,MAXM=1005005;
3 struct node{
4 int u,v;
5 LL w,tag;
6 node *l,*r;
7 node(int u=0,int v=0,LL w=0):u(u),v(v),w(w),tag(0),l(0),r(0){}
8 void down(){
9 w+=tag;
10 if(l)l->tag+=tag;
11 if(r)r->tag+=tag;
12 tag=0;
13 }
14 }mem[MAXN];
15 node *pq[MAXN*2],*E[MAXN*2];
16 int st[MAXN*2],id[MAXN*2],m,from[MAXN*2];
17 void init(int n){
18 for(int i=1;i<=n;++i){
19 pq[i]=E[i]=0;
20 st[i]=id[i]=i;
21 from[i]=0;
22 }m=0;
23 }
24 node *merge(node *a,node *b){//skew heap
25 if(!a||!b)return a?a:b;
26 a->down(),b->down();
27 if(b->w<a->w)return merge(b,a);
28 if(b->w==a->w&&b->v<a->v)return merge(b,a);//
29 swap(a->l,a->r);
30 a->l=merge(b,a->l);
31 return a;
32 }
33 void add_edge(int u,int v,LL w){
34 if(u!=v)pq[v]=merge(pq[v],&(mem[m++]=node(u,v,w)));
35 }
36 int find(int x,int *st){
37 return st[x]==x?x:st[x]=find(st[x],st);
38 }
39 LL build(int root,int n){
40 LL ans=0;int N=n,all=n;
41 for(int i=1;i<=N;++i){
42 if(i==root||!pq[i])continue;
43 while(pq[i]){
44 pq[i]->down(),E[i]=pq[i];
45 pq[i]=merge(pq[i]->l,pq[i]->r);
46 if(find(E[i]->u,id)!=find(i,id))break;
47 }
48 if(find(E[i]->u,id)==find(i,id))continue;
49 from[E[i]->v]=E[i]->u;
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[++N]=pq[i],id[N]=N;
54 for(int u=find(E[i]->u,id);u!=i;u=find(E[u]->u,id)){
55 if(pq[u])pq[u]->tag-=E[u]->w;
56 id[find(u,id)]=N;
57 pq[N]=merge(pq[N],pq[u]);
58 }
59 st[N]=find(i,st);
```

```

60     id[find(i,id)]=N;
61 }else st[find(i,st)]=find(E[i]->u,st),--all;
62 }
63 return all==1?ans:-1;//圖不連通就無解
64 }
65 }MST;

```

## 5.6 SCC

```

1 struct Scc{
2     int n, nScc, vst[MXN], bln[MXN];
3     vector<int> E[MXN], rE[MXN], vec;
4     void init(int _n){
5         n = _n;
6         for (int i=0; i<MXN; i++){
7             E[i].clear();
8             rE[i].clear();
9         }
10    }
11    void add_edge(int u, int v){
12        E[u].pb(v);
13        rE[v].pb(u);
14    }
15    void DFS(int u){
16        vst[u]=1;
17        for (auto v : E[u])
18            if (!vst[v]) DFS(v);
19        vec.pb(u);
20    }
21    void rDFS(int u){
22        vst[u] = 1;
23        bln[u] = nScc;
24        for (auto v : rE[u])
25            if (!vst[v]) rDFS(v);
26    }
27    void solve(){
28        nScc = 0;
29        vec.clear();
30        MEM(vst);
31        for (int i=0; i<n; i++)
32            if (!vst[i]) DFS(i);
33        reverse(vec.begin(),vec.end());
34        FZ(vst);
35        for (auto v : vec){
36            if (!vst[v]){
37                rDFS(v);
38                nScc++;
39            }
40        }
41    }
42 };

```

## 5.7 GeneralGraphMaximunValueMatch

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 //from vfleaking
4 //自己進行一些進行一些小修改
5 #define INF INT_MAX
6 #define MAXN 400
7 struct edge{
8     int u,v,w;
9     edge(){}
10    edge(int u,int v,int w):u(u),v(v),w(w){}
11 };
12 int n,n_x;
13 edge g[MAXN*2+1][MAXN*2+1];
14 int lab[MAXN*2+1];
15 int match[MAXN*2+1],slack[MAXN*2+1],st[MAXN*2+1],pa[
    MAXN*2+1];
16 int flower_from[MAXN*2+1][MAXN+1],S[MAXN*2+1],vis[
    MAXN*2+1];
17 vector<int> flower[MAXN*2+1];
18 queue<int> q;
19 inline int e_delta(const edge &e){ // does not work
    inside blossoms
20     return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
21 }
22 inline void update_slack(int u,int x){
23     if(!slack[x]||e_delta(g[u][x])<e_delta(g[slack[x]
    ][x]))slack[x]=u;

```

```

24 }
25 inline void set_slack(int x){
26     slack[x]=0;
27     for(int u=1;u<=n;++u)
28         if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)
29             update_slack(u,x);
30 }
31 void q_push(int x){
32     if(x<=n)q.push(x);
33     else for(size_t i=0;i<flower[x].size();i++)q.push(
    flower[x][i]);
34 }
35 inline void set_st(int x,int b){
36     st[x]=b;
37     if(x>n)for(size_t i=0;i<flower[x].size();++i)
38         set_st(flower[x][i],b);
39 }
40 inline int get_pr(int b,int xr){
41     int pr=find(flower[b].begin(),flower[b].end(),xr)-
    flower[b].begin();
42     if(pr%2==1){//檢查他在前一層圖是奇點還是偶點
43         reverse(flower[b].begin()+1,flower[b].end());
44         return (int)flower[b].size()-pr;
45     }else return pr;
46 }
47 inline void set_match(int u,int v){
48     match[u]=g[u][v].v;
49     if(u>n){
50         edge e=g[u][v];
51         int xr=flower_from[u][e.u],pr=get_pr(u,xr);
52         for(int i=0;i<pr;++i)set_match(flower[u][i],
    flower[u][i^1]);
53         set_match(xr,v);
54         rotate(flower[u].begin(),flower[u].begin()+pr,
    flower[u].end());
55     }
56 }
57 inline void augment(int u,int v){
58     for(;;){
59         int xnv=st[match[u]];
60         set_match(u,v);
61         if(!xnv)return;
62         set_match(xnv,st[pa[xnv]]);
63         u=st[pa[xnv]],v=xnv;
64     }
65 }
66 inline int get_lca(int u,int v){
67     static int t=0;
68     for(++t;u||v;swap(u,v)){
69         if(u==0)continue;
70         if(vis[u]==t)return u;
71         vis[u]=t;//這種方法可以不用清空v陣列
72         u=st[match[u]];
73         if(u)u=st[pa[u]];
74     }
75     return 0;
76 }
77 inline void add_blossom(int u,int lca,int v){
78     int b=n+1;
79     while(b<=n_x&&st[b])++b;
80     if(b>n_x)++n_x;
81     lab[b]=0,S[b]=0;
82     match[b]=match[lca];
83     flower[b].clear();
84     flower[b].push_back(lca);
85     for(int x=u,y;x!=lca;x=st[pa[y]])
86         flower[b].push_back(x),flower[b].push_back(y=st[
    match[x]]),q_push(y);
87     reverse(flower[b].begin()+1,flower[b].end());
88     for(int x=v,y;x!=lca;x=st[pa[y]])
89         flower[b].push_back(x),flower[b].push_back(y=st[
    match[x]]),q_push(y);
90     set_st(b,b);
91     for(int x=1;x<=n_x;++x)g[b][x].w=g[x][b].w=0;
92     for(int x=1;x<=n;++x)flower_from[b][x]=0;
93     for(size_t i=0;i<flower[b].size();++i){
94         int xs=flower[b][i];
95         for(int x=1;x<=n_x;++x)
96             if(g[b][x].w==0||e_delta(g[xs][x])<e_delta(g[b]
    ][x]))
97                 g[b][x]=g[xs][x],g[x][b]=g[x][xs];
98     }
99 }

```

```

98     if(flower_from[xs][x])flower_from[b][x]=xs;
99 }
100 set_slack(b);
101 }
102 inline void expand_blossom(int b){ // S[b] == 1
103     for(size_t i=0;i<flower[b].size();++i)
104         set_st(flower[b][i],flower[b][i]);
105     int xr=flower_from[b][g[b][pa[b]].u],pr=get_pr(b,
106         xr);
107     for(int i=0;i<pr;i+=2){
108         int xs=flower[b][i],xns=flower[b][i+1];
109         pa[xs]=g[xns][xs].u;
110         S[xs]=1,S[xns]=0;
111         slack[xs]=0,set_slack(xns);
112         q_push(xns);
113     }
114     S[xr]=1,pa[xr]=pa[b];
115     for(size_t i=pr+1;i<flower[b].size();++i){
116         int xs=flower[b][i];
117         S[xs]=-1,set_slack(xs);
118     }
119     st[b]=0;
120 }
121 inline bool on_found_edge(const edge &e){
122     int u=st[e.u],v=st[e.v];
123     if(S[v]==-1){
124         pa[v]=e.u,S[v]=1;
125         int nu=st[match[v]];
126         slack[v]=slack[nu]=0;
127         S[nu]=0,q_push(nu);
128     }else if(S[v]==0){
129         int lca=get_lca(u,v);
130         if(!lca)return augment(u,v),augment(v,u),true;
131         else add_blossom(u,lca,v);
132     }
133     return false;
134 }
135 inline bool matching(){
136     memset(S+1,-1,sizeof(int)*n_x);
137     memset(slack+1,0,sizeof(int)*n_x);
138     q=queue<int>();
139     for(int x=1;x<=n_x;++x)
140         if(st[x]==x&&!match[x])pa[x]=0,S[x]=0,q_push(x);
141     if(q.empty())return false;
142     for(;;){
143         while(q.size()){
144             int u=q.front();q.pop();
145             if(S[st[u]]==1)continue;
146             for(int v=1;v<=n;v++){
147                 if(g[u][v].w>0&&st[u]!=st[v]){
148                     if(e_delta(g[u][v])==0){
149                         if(on_found_edge(g[u][v]))return true;
150                     }else update_slack(u,st[v]);
151                 }
152             }
153             int d=INF;
154             for(int b=n+1;b<=n_x;++b)
155                 if(st[b]==b&&S[b]==1)d=min(d,lab[b]/2);
156             for(int x=1;x<=n_x;++x)
157                 if(st[x]==x&&slack[x]){
158                     if(S[x]==-1)d=min(d,e_delta(g[slack[x]][x]));
159                     else if(S[x]==0)d=min(d,e_delta(g[slack[x]][x])/2);
160                 }
161             for(int u=1;u<=n;u++){
162                 if(S[st[u]]==0){
163                     if(lab[u]<=d)return 0;
164                     lab[u]-=d;
165                 }else if(S[st[u]]==1)lab[u]+=d;
166             }
167             for(int b=n+1;b<=n_x;++b)
168                 if(st[b]==b){
169                     if(S[st[b]]==0)lab[b]+=d*2;
170                     else if(S[st[b]]==1)lab[b]-=d*2;
171                 }
172             q=queue<int>();
173             for(int x=1;x<=n_x;++x)
174                 if(st[x]==x&&slack[x]&&st[slack[x]]!=x&&
175                     e_delta(g[slack[x]][x])>0)
176                     if(on_found_edge(g[slack[x]][x]))return true;
177         }
178     }

```

```

175     for(int b=n+1;b<=n_x;++b)
176         if(st[b]==b&&S[b]==1&&lab[b]==0)expand_blossom(b);
177 }
178 return false;
179 }
180 inline pair<long long,int> weight_blossom(){
181     memset(match+1,0,sizeof(int)*n);
182     n_x=n;
183     int n_matches=0;
184     long long tot_weight=0;
185     for(int u=0;u<=n;u++)st[u]=u,flower[u].clear();
186     int w_max=0;
187     for(int u=1;u<=n;u++){
188         for(int v=1;v<=n;v++){
189             flower_from[u][v]=(u==v?u:0);
190             w_max=max(w_max,g[u][v].w);
191         }
192     }
193     for(int u=1;u<=n;u++)lab[u]=w_max;
194     while(matching())n_matches++;
195     for(int u=1;u<=n;u++)
196         if(match[u]&&match[u]<u)
197             tot_weight+=g[u][match[u]].w;
198     return make_pair(tot_weight,n_matches);
199 }
200 inline void init_weight_graph(){
201     for(int u=1;u<=n;u++)
202         for(int v=1;v<=n;v++)
203             g[u][v]=edge(u,v,0);
204 }
205 int main(){
206     int m;
207     scanf("%d",&n,&m);
208     init_weight_graph();
209     for(int i=0;i<m;i++){
210         int u,v,w;
211         scanf("%d%d%d",&u,&v,&w);
212         g[u][v].w=g[v][u].w=w;
213     }
214     printf("%lld\n",weight_blossom().first);
215     for(int u=1;u<=n;u++)printf("%d ",match[u]);puts("
");
216     return 0;

```

## 5.8 Stable Marriage

```

1 #define F(n) Fi(i, n)
2 #define Fi(i, n) Fl(i, 0, n)
3 #define Fl(i, l, n) for(int i = l ; i < n ; ++i)
4 #include <bits/stdc++.h>
5 using namespace std;
6 int D, quota[205], weight[205][5];
7 int S, scoretodep[12005][205], score[5];
8 int P, prefer[12005][85], iter[12005];
9 int ans[12005];
10 typedef pair<int, int> PII;
11 map<int, int> samescore[205];
12 typedef priority_queue<PII, vector<PII>, greater<PII>
13 >> QQQ;
14 void check(int d) {
15     PII t = pri[d].top();
16     int v;
17     if (pri[d].size() - samescore[d][t.first] + 1 <=
18         quota[d]) return;
19     while (pri[d].top().first == t.first) {
20         v = pri[d].top().second;
21         ans[v] = -1;
22         --samescore[d][t.first];
23         pri[d].pop();
24     }
25 }
26 void push(int s, int d) {
27     if (pri[d].size() < quota[d]) {
28         pri[d].push(PII(scoretodep[s][d], s));
29         ans[s] = d;
30         ++samescore[s][scoretodep[s][d]];
31     } else if (scoretodep[s][d] >= pri[d].top().first) {
32         pri[d].push(PII(scoretodep[s][d], s));
33         ans[s] = d;

```

```

33     ++samescore[s][scoretodep[s][d]];
34     check(d);
35 }
36 }
37 void f() {
38     int over;
39     while (true) {
40         over = 1;
41         Fi (q, S) {
42             if (ans[q] != -1 || iter[q] >= P) continue;
43             push(q, prefer[q][iter[q]++]);
44             over = 0;
45         }
46         if (over) break;
47     }
48 }
49 main() {
50     ios::sync_with_stdio(false);
51     cin.tie(NULL);
52     int sadmit, stof, dexceed, dfew;
53     while (cin >> D, D) { // Beware of the input
54         format or judge may troll us.
55         sadmit = stof = dexceed = dfew = 0;
56         memset(iter, 0, sizeof(iter));
57         memset(ans, 0, sizeof(ans));
58         Fi (q, 205) {
59             pri[q] = QQQ();
60             samescore[q].clear();
61         }
62         cin >> S >> P;
63         Fi (q, D) {
64             cin >> quota[q];
65             Fi (w, 5) cin >> weight[q][w];
66         }
67         Fi (q, S) {
68             Fi (w, 5) cin >> score[w];
69             Fi (w, D) {
70                 scoretodep[q][w] = 0;
71                 F (5) scoretodep[q][w] += weight[w][i] *
72                 score[i];
73             }
74             Fi (q, S) Fi (w, P) {
75                 cin >> prefer[q][w];
76                 --prefer[q][w];
77             }
78             f();
79             Fi (q, D) sadmit += pri[q].size();
80             Fi (q, S) if (ans[q] == prefer[q][0]) ++stof;
81             Fi (q, D) if (pri[q].size() > quota[q]) ++
82             dexceed;
83             Fi (q, D) if (pri[q].size() < quota[q]) ++dfew;
84             cout << sadmit << ' ' << stof << ' ' << dexceed
85             << ' ' << dfew << '\n';
86         }
87     }
88 }

```

## 5.9 BCCvertex

```

1 const int MXN = 16004;
2 struct BccVertex {
3     int n, nScc, step, dfn[MXN], low[MXN];
4     vector<int> E[MXN], sccv[MXN];
5     int top, stk[MXN];
6     void init(int _n) {
7         n = _n;
8         nScc = step = 0;
9         for (int i=0; i<n; i++) E[i].clear();
10    }
11    void add_edge(int u, int v) {
12        E[u].pb(v);
13        E[v].pb(u);
14    }
15    void DFS(int u, int f) {
16        dfn[u] = low[u] = step++;
17        stk[top++] = u;
18        for (auto v:E[u]) {
19            if (v == f) continue;
20            if (dfn[v] == -1) {
21                DFS(v, u);
22                low[u] = min(low[u], low[v]);
23                if (low[v] >= dfn[u]) {

```

```

24                int z;
25                sccv[nScc].clear();
26                do {
27                    z = stk[--top];
28                    sccv[nScc].pb(z);
29                } while (z != v);
30                sccv[nScc].pb(u);
31                nScc++;
32            } else {
33                low[u] = min(low[u], dfn[v]);
34            }
35        }
36    }
37 }
38 vector<vector<int>> solve() {
39     vector<vector<int>> res;
40     for (int i=0; i<n; i++) {
41         dfn[i] = low[i] = -1;
42     }
43     for (int i=0; i<n; i++) {
44         if (dfn[i] == -1) {
45             top = 0;
46             DFS(i, i);
47         }
48     }
49     for(int i=0; i<nScc; i++) res.pb(sccv[i]);
50     return res;
51 }
52 }graph;

```

## 5.10 MaxClique

```

1 class MaxClique {
2 public:
3     static const int MV = 210;
4     int V;
5     int el[MV][MV/30+1];
6     int dp[MV];
7     int ans;
8     int s[MV][MV/30+1];
9     vector<int> sol;
10    void init(int v) {
11        V = v; ans = 0;
12        MEMS(el); MEMS(dp);
13    }
14    /* Zero Base */
15    void addEdge(int u, int v) {
16        if(u > v) swap(u, v);
17        if(u == v) return;
18        el[u][v/32] |= (1<<(v%32));
19    }
20    bool dfs(int v, int k) {
21        int c = 0, d = 0;
22        for(int i=0; i<(V+31)/32; i++) {
23            s[k][i] = el[v][i];
24            if(k != 1) s[k][i] &= s[k-1][i];
25            c += __builtin_popcount(s[k][i]);
26        }
27        if(c == 0) {
28            if(k > ans) {
29                ans = k;
30                sol.clear();
31                sol.push_back(v);
32                return 1;
33            }
34            return 0;
35        }
36        for(int i=0; i<(V+31)/32; i++) {
37            for(int a = s[k][i]; a; a >> 1) {
38                if(k + (c-d) <= ans) return 0;
39                int lb = a&(-a), lg = 0;
40                a ^= lb;
41                while(lb!=1) {
42                    lb = (unsigned int)(lb) >> 1;
43                    lg++;
44                }
45                int u = i*32 + lg;
46                if(k + dp[u] <= ans) return 0;
47                if(dfs(u, k+1)) {
48                    sol.push_back(v);
49                    return 1;
50                }

```

```

51     }
52     }
53     return 0;
54 }
55 int solve() {
56     for(int i=V-1; i>=0; i--) {
57         dfs(i, 1);
58         dp[i] = ans;
59     }
60     return ans;
61 }
62 };

```

## 5.11 BCCedge

```

1 vector<vector<int>> > v;
2 int vis[100005], lwn[100005];
3 vector<int> stk;
4 int f[100005];
5 int bln[100005];
6 int Find(int a){
7     if(bln[a]==a) return a;
8     return bln[a]=Find(bln[a]);
9 }
10 int t;
11 void dfs(int a, int p){
12     stk.pb(a);
13     bln[a]=a;
14     vis[a]=lwn[a]=++t;
15     int cnt=0;
16     for(int i=0; i<v[a].size(); i++){
17         int x=v[a][i];
18         if(x!=p||cnt==1){
19             if(vis[x]==0){
20                 dfs(x, a);
21                 if(lwn[x]>vis[a]){
22                     int fa=Find(x);
23                     f[x]=Find(a);
24                     while(stk.back()!=x){
25                         bln[stk.back()]=fa;
26                         stk.pop_back();
27                     }
28                     bln[stk.back()]=fa;
29                     stk.pop_back();
30                 }
31                 lwn[a]=min(lwn[a], lwn[x]);
32             }
33             else{
34                 lwn[a]=min(lwn[a], vis[x]);
35             }
36         }
37         else{
38             cnt++;
39         }
40     }
41 }

```

## 6 JAVA

### 6.1 Big Integer

```

1 import java.math.*;
2 import java.io.*;
3 import java.util.*;
4 public class Main{
5     public static void main(String []argv){
6         c[0][0]=BigInteger.ONE;
7         for(int i=1; i<3001; i++){
8             c[i][0]=BigInteger.ONE;
9             c[i][i]=BigInteger.ONE;
10            for(int j=1; j<i; j++) c[i][j]=c[i-1][j].
add(c[i-1][j-1]);
11        }
12        Scanner scanner = new Scanner(System.in);
13        int T = scanner.nextInt();
14        BigInteger x;
15        BigInteger ans;
16        while(T-- > 0){
17            ans = BigInteger.ZERO;
18            int n = scanner.nextInt();

```

```

19            for(int i=0; i<n; i++){
20                x = new BigInteger(scanner.next());
21                if(i%2 == 1) ans=ans.subtract(c[n-1][i].multiply(x));
22                else ans=ans.add(c[n-1][i].multiply(x));
23            }
24            if(n%2 == 0) ans=BigInteger.ZERO.subtract(ans);
25            System.out.println(ans);
26        }
27    }
28 }

```

## 6.2 Prime

```

1 import java.math.*;
2 import java.io.*;
3 import java.util.*;
4 public class Main{
5     public static void main(String []argv){
6         Scanner scanner = new Scanner(System.in);
7         int T = scanner.nextInt();
8         for (int cs = 0 ; cs < T ; cs++){
9             if (cs != 0) { System.out.println(""); }
10            int a = scanner.nextInt();
11            int b = scanner.nextInt();
12            for (int i = a ; i <= b ; i++) {
13                BigInteger x = BigInteger.valueOf(i)
14                ;
15                if (x.isProbablePrime(5) == true) {
16                    System.out.println(x);
17                }
18            }
19        }
20 }

```

## 7 Other

### 7.1 Dp Optimizer

```

1 int search(list<int>::iterator x, int i, int n){
2     int Max=n+1, Min=*x;
3     list<int>::iterator last=x;
4     last--;
5     while(Max>Min+1){
6         int mid=(Max+Min)/2;
7         int a=*last, b=*x;
8         int val1=dp[a][i-1]+cost[a+1][mid], val2=dp[b
][i-1]+cost[b+1][mid];
9         if(val1>val2) Max=mid;
10        else Min=mid;
11    }
12    return Max;
13 }
14 void solve(){
15     for(int i=2; i<=m; i++){
16         mylist.clear(); mylist.pb(i-1);
17         v.clear(); v.resize(n);
18         vis.clear(); vis.resize(n+1, 0);
19         for(int j=i; j<=n; j++){
20             while(!pq.empty() && pq.top().x<=j){
21                 pii p=pq.top();
22                 pq.pop();
23                 if(vis[p.y]) continue;
24                 auto it=v[p.y]; it--;
25                 vis[*it]=1;
26                 mylist.erase(it);
27                 if(v[p.y]!=mylist.begin())
28                     pq.push(mp(search(v[p.y], i, n), p.
y));
29             }
30             int opt=mylist.front();
31             dp[j][i]=dp[opt][i-1]+cost[opt+1][j];
32             mylist.push_back(j);
33             v[j]=mylist.end();
34             v[j]--;
35             pq.push(mp(search(v[j], i, n), j));
36         }
37         while(!pq.empty()) pq.pop();
38     }
39 }

```



## 7.2 Annealing

```

1 double distForAllPoints(double x, double y,
2     vector< pair<int, int> > &D) {
3     double sum = 0;
4     for(int i = D.size()-1; i >= 0; i--) {
5         sum += hypot(D[i].first - x, D[i].second - y);
6     }
7     return sum;
8 }
9 double randDouble() {
10    return (rand() % 32767) / 32767.0;
11 }
12 double annealing(vector< pair<int, int> > &D) {
13     #define S_MUL 0.6f
14     #define S_LEN 1000
15     #define T_CNT 10
16     #define E_CNT 10
17     double step = S_LEN;
18     double x[E_CNT], y[E_CNT], val[E_CNT];
19     double Lx, Ly, Rx, Ry, tx, ty, tcost;
20     Lx = Rx = D[0].first;
21     Ly = Ry = D[0].second;
22     for(int i = 0; i < D.size(); i++) {
23         Lx = min(Lx, (double)D[i].first);
24         Rx = max(Rx, (double)D[i].first);
25         Ly = min(Ly, (double)D[i].second);
26         Ry = max(Ry, (double)D[i].second);
27     }
28     for(int i = 0; i < E_CNT; i++) {
29         x[i] = randDouble() * (Rx - Lx) + Lx;
30         y[i] = randDouble() * (Ry - Ly) + Ly;
31         val[i] = distForAllPoints(x[i], y[i], D);
32     }
33     while(step > 0.1) {
34         for(int i = 0; i < E_CNT; i++) {
35             for(int j = 0; j < T_CNT; j++) {
36                 tx = x[i] + randDouble() * 2 * step - step;
37                 ty = y[i] + randDouble() * 2 * step - step;
38                 tcost = distForAllPoints(tx, ty, D);
39                 if(tcost < val[i]) {
40                     val[i] = tcost, x[i] = tx, y[i] = ty;
41                 }
42             }
43         }
44         step *= S_MUL;
45     }
46     double ret = val[0];
47     for(int i = 0; i < E_CNT; i++) {
48         ret = min(ret, val[i]);
49     }
50     printf("%.0lf\n", ret);
51 }
52 int main() {
53     int testcase, N;
54     scanf("%d", &testcase);
55     while(testcase--) {
56         scanf("%d", &N);
57         vector< pair<int, int> > D;
58         int x, y;
59         for(int i = 0; i < N; i++) {
60             scanf("%d %d", &x, &y);
61             D.push_back(make_pair(x, y));
62         }
63         annealing(D);
64         if(testcase)
65             puts("");
66     }
67     return 0;
68 }

```

## 7.3 DLX

```

1 struct DLX{
2     int n,m,len;
3     int U[maxnode], D[maxnode], R[maxnode], L[maxnode],
4     Row[maxnode], Col[maxnode];
5     int H[maxn];
6     int S[maxm];
7     int ansd,ans[maxn];
8     void init(int _n,int _m){

```

```

9         n = _n; m = _m;
10        for(int i = 0; i <= m; i++){
11            S[i] = 0;
12            U[i] = D[i] = i;
13            L[i] = i-1;
14            R[i] = i+1;
15        }
16        R[m] = 0, L[0] = m;
17        len = m;
18        for(int i = 1; i <= n; i++){
19            H[i] = -1;
20        }
21
22        void link(int r, int c){
23            ++S[Col[++len]=c];
24            Row[len] = r;
25            D[len] = D[c];
26            U[D[c]] = len;
27            U[len] = c;
28            D[c] = len;
29            if(H[r] < 0)
30                H[r] = L[len] = R[len] = len;
31            else{
32                R[len] = R[H[r]];
33                L[R[H[r]]] = len;
34                L[len] = H[r];
35                R[H[r]] = len;
36            }
37        }
38
39        void del(int c){
40            L[R[c]] = L[c];
41            R[L[c]] = R[c];
42            for(int i = D[c]; i != c; i = D[i]){
43                for(int j = R[i]; j != i; j = R[j]){
44                    U[D[j]] = U[j];
45                    D[U[j]] = D[j];
46                    --S[Col[j]];
47                }
48            }
49        }
50
51        void resume(int c){
52            for(int i = U[c]; i != c; i = U[i]){
53                for(int j = L[i]; j != i; j = L[j]){
54                    ++S[Col[U[D[j]]=D[U[j]]=j]];
55                }
56            }
57            L[R[c]] = R[L[c]] = c;
58        }
59
60        void dance(int d){
61            //剪枝
62            if(ansd != -1 && ansd <= d)
63                return;
64            if(R[0] == 0){
65                if(ansd == -1)
66                    ansd = d;
67                else if(d < ansd)
68                    ansd = d;
69                return;
70            }
71            int c = R[0];
72            for(int i = R[0]; i != 0; i = R[i]){
73                if(S[i] < S[c])
74                    c = i;
75            }
76            del(c);
77            for(int i = D[c]; i != c; i = D[i]){
78                ans[d] = Row[i];
79                for(int j = R[i]; j != i; j = R[j])
80                    del(Col[j]);
81                dance(d+1);
82                for(int j = L[i]; j != i; j = L[j])
83                    resume(Col[j]);
84            }
85            resume(c);
86        }
87    };

```

## 7.4 MahattanMST

```

1 #include<bits/stdc++.h>
2 #define REP(i,n) for(int i=0;i<n;i++)
3 using namespace std;
4 typedef long long LL;
5 const int N=200100;
6 int n,m;
7 struct PT {int x,y,z,w,id;}p[N];
8 inline int dis(const PT &a,const PT &b){return abs(a
    .xb.x)+abs(a.y-b.y);}
9 inline bool cpx(const PT &a,const PT &b){return a.x
    !=b.
10 x? a.x>b.x:a.y>b.y;}
11 inline bool cpz(const PT &a,const PT &b){return a.z<
    b.z
12 ;}
13 struct E{int a,b,c;}e[8*N];
14 bool operator<(const E&a,const E&b){return a.c<b.c;}
15 struct Node{
16     int L,R,key;
17 }node[4*N];
18 int s[N];
19 int F(int x){return s[x]==x?s[x]=F(s[x]);}
20 void U(int a,int b){s[F(b)]=F(a);}
21 void init(int id,int L,int R) {
22     node[id]=(Node){L,R,-1};
23     if(L==R)return
24     ;
25     init(id*2,L,(L+R)/2);
26     init(id*2+1,(L+R)/2+1,R);
27 }
28 void ins(int id,int x) {
29     if(node[id].key==-1 || p[node[id].key].w>p[x].w)
30     node[
31     id].key=x;
32     if(node[id].L==node[id].R)return
33     ;
34     if(p[x].z<=(node[id].L+node[id].R)/2)ins(id*2,x)
35     ;
36     else ins(id*2+1,x);
37 }
38 int Q(int id,int L,int R){
39     if(R<node[id].L || L>node[id].R)return -1;
40     if(L<=node[id].L && node[id].R<=R)return node[id
41     ].key ;
42     int a=Q(id*2,L,R),b=Q(id*2+1,L,R);
43     if(b==-1 || (a!=-1 && p[a].w<p[b].w)) return a;
44     else return b;
45 }
46 void calc() {
47     REP(i,n) {
48         p[i].z=p[i].y-p[i].x;
49         p[i].w=p[i].x+p[i].y;
50     }
51     sort(p,p+n,cpz);
52     int cnt=0,j,k;
53     for
54     (int i=0;i<n;i++){
55         for(j=i+1;p[j].z==p[i].z && j<n;j++);
56         for(k=i,cnt++;k<j;k++)p[k].z=cnt;
57     }
58     init(1,1,cnt);
59     sort(p,p+n,cpx);
60     REP(i,n) {
61         j=Q(1,p[i].z,cnt);
62         if(j!=-1)e[m++]=(E){p[i].id,p[j].id,dis(p[i
63     ],p[j])
64     };
65     ins(1,i);
66 }
67 LL MST() {
68     LL r=0;
69     sort(e,e+m);
70     REP(i,m) {
71         if(F(e[i].a)==F(e[i].b))continue;
72         U(e[i].a,e[i].b);
73         r+=e[i].c;
74     }
75     return r;
76 }
77 int main(){

```

```

75     int ts;
76     scanf("%d",&ts);
77     while (ts--) {
78         m = 0;
79         scanf("%d",&n);
80         REP(i,n) {scanf("%d",&p[i].x,&p[i].y);p[i
81     ].id=s[i]=i;}
82         calc();
83         REP(i,n)p[i].y= -p[i].y;
84         calc();
85         REP(i,n)swap(p[i].x,p[i].y);
86         calc();
87         REP(i,n)p[i].x=-p[i].x;
88         calc();
89         printf("%lld\n",MST()*2);
90     }
91     return 0;
92 }

```

## 7.5 MoOnTree

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define IOS ios_base::sync_with_stdio(0); cin.tie(0)
4 ;
5 #define SZ(x) ((int)((x).size()))
6 const int MX = 500005;
7 const int SQ = 1400;
8 const int LOG = 17;
9 struct BIT {
10     int bit[MX];
11     int lb(int x) { return x & -x; }
12     void add(int p, int v) {
13         p++;
14         for (int i=p; i<MX; i+=lb(i)) bit[i] += v;
15     }
16     int qry() {
17         int v = 0;
18         for (int i=1<<LOG; i>0; i>=1) {
19             if ((v|i) < MX and bit[v|i]==i) v |= i;
20         }
21         return v;
22     }
23 }bit;
24 struct Query {
25     int l,r,qid;
26 }qry[MX];
27 struct Edge {
28     int v,x;
29 };
30 int N,Q,timestamp[MX],ans[MX];
31 int in[MX],cnt[MX];
32 vector<Edge> E[MX];
33 vector<Edge> seq;
34 void DFS(int u, int f) {
35     timestamp[u] = SZ(seq);
36     for (auto it:E[u]) {
37         if (it.v == f) continue;
38         seq.push_back(it);
39         DFS(it.v,u);
40         seq.push_back(it);
41     }
42 }
43 void poke(int id) {
44     int v = seq[id].v;
45     int x = seq[id].x;
46     in[v] ^= 1;
47     cnt[x] += in[v] ? 1 : -1;
48     if (in[v] and cnt[x] == 1) bit.add(x, 1);
49     if (!in[v] and cnt[x] == 0) bit.add(x, -1);
50 }
51 int main() {
52     IOS;
53     cin >> N >> Q;
54     for (int i=0; i<N-1; i++) {
55         int u,v,x;
56         cin >> u >> v >> x;
57         x = min(x,N);
58         E[u].push_back({v,x});
59         E[v].push_back({u,x});
60     }
61     DFS(1,1);

```

```

61 for (int i=1; i<=Q; i++) {
62     int u,v;
63     cin >> u >> v;
64     int l = timestamp[u], r = timestamp[v];
65     if (l > r) swap(l,r);
66     r--;
67     qry[i] = {l,r,i};
68 }
69 sort(qry+1,qry+1+Q, [](Query a, Query b) {
70     return make_pair(a.l/SQ,a.r) < make_pair(b.l
71     /SQ,b
72     .r);
73 });
74 int curL = 1, curR = 0;
75 for (int i=1; i<=Q; i++) {
76     int ql=qry[i].l,qv=qry[i].r;
77     while (curL > ql) poke(--curL);
78     while (curR < qv) poke(++curR);
79     while (curL < ql) poke(curL++);
80     while (curR > qv) poke(curR--);
81     ans[qry[i].qid] = bit.qry();
82 }
83 for (int i=1; i<=Q; i++) cout << ans[i] << "\n";
84 return 0;
85 }

```

## 7.6 Det

```

1 LL det(LL a[][20],int n)
2 {
3     LL ret=1;
4     for(int i=1;i<n;i++)
5     {
6         for(int j=i+1;j<n;j++)
7             while(a[j][i])
8             {
9                 LL t=a[i][i]/a[j][i];
10                for(int k=i;k<n;k++)
11                    a[i][k]=a[i][k]-a[j][k]*t;
12                for(int k=i;k<n;k++)
13                    swap(a[i][k],a[j][k]);
14                ret=-ret;
15            }
16        if(a[i][i]==0)return 0;
17        ret=ret*a[i][i];
18    }
19    return ret;
20 }

```

## 8 String

### 8.1 AC

```

1 struct Node{
2     Node *index[30];
3     Node *fail;
4     int word;
5     int num;
6     Node(){
7         for(int i=0;i<30;i++)
8             index[i]=NULL;
9         fail=NULL;
10        word=0;
11        num=-1;
12    }
13 }*root=new Node();
14 void add(char c[]){
15     Node *n=root;
16     for(int i=0;c[i]!='\0';i++){
17         if(!n->index[c[i]-'a'])
18             n->index[c[i]-'a']=new Node();
19         n=n->index[c[i]-'a'];
20     }
21     n->word=1;
22     n->num++;
23 }
24 void ac(){
25     queue<Node*> q;

```

```

27 q.push(root);
28 root->fail=NULL;
29 while(!q.empty()){
30     Node *n=q.front();
31     q.pop();
32     for(int i=0;i<30;i++){
33         if(n->index[i]){
34             q.push(n->index[i]);
35             Node* p=n->fail;
36             while(p!=NULL&&!p->index[i])
37                 p=p->fail;
38             if(p)
39                 n->index[i]->fail=p->index[i];
40             else
41                 n->index[i]->fail=root;
42         }
43     }
44 }
45 }
46 void search(char c[]){
47     Node *n=root;
48     for(int i=0;c[i]!='\0';i++){
49         while(!n->index[c[i]-'a']&&n!=root){
50             n=n->fail;
51         }
52         if(n->index[c[i]-'a'])
53             n=n->index[c[i]-'a'];
54         Node *p=n;
55         while(p){
56             if(p->num!=-1)
57                 ans[p->num]++;
58             p=p->fail;
59         }
60     }
61 }
62 }
63 }
64 }
65 void del(Node *n=root){
66     for(int i=0;i<30;i++){
67         if(n->index[i])
68             del(n->index[i]);
69         free(n);
70     }

```

## 8.2 SuffixAutomata

```

1 // BZOJ 3998
2 const int MAX_N = 500000 + 10;
3 struct Node {
4     static Node mem[MAX_N<<1], *pmem;
5     Node *ch[26], *fail;
6     int mx, val;
7     ll dp;
8     int tag, deg;
9     Node():mx(0),fail(0),dp(0),val(0),tag(0),deg(0){
10         MS(ch, 0);
11     }
12 }
13 Node::mem[MAX_N<<1], *Node::pmem = Node::mem, *
14     root
15     , *last;
16 int T, N;
17 char s[MAX_N];
18 inline void init() {
19     last = root = new (Node::pmem++)Node();
20 }
21 inline int idx(char c) {
22     return c - 'a';
23 }
24 inline void insert(char c) {
25     c = idx(c);
26     Node *p = last;
27     Node *np = new (Node::pmem++)Node();
28     np->mx = p->mx + 1;
29     np->val = 1;
30     while(p && !p->ch[c]) {
31         p->ch[c] = np;
32         p = p->fail;
33     }
34     if(!p) np->fail = root;

```

```

35 else
36 {
37     Node *q = p->ch[c];
38     if(q->mx == p->mx + 1) np->fail = q;
39     else
40     {
41         Node *nq = new (Node::pmem++)Node();
42         nq->mx = p->mx + 1;
43         nq->val = 0;
44         memcpy(nq->ch, q->ch, sizeof(q->ch));
45         REP(i, 26) {
46             if(nq->ch[i]) nq->ch[i]->deg++;
47         }
48         nq->fail = q->fail;
49         q->fail = np->fail = nq;
50         while(p && p->ch[c] == q) {
51             p->ch[c] = nq;
52             q->deg--;
53             nq->deg++;
54             p = p->fail;
55         }
56     }
57 }
58 last = np;
59 }
60 inline void bfs() {
61     static Node* que[MAX_N<<1];
62     int l = 0, r = 0;
63     que[r++] = root;
64     root->tag = 2;
65     vector<Node*> vec;
66     while(l < r) {
67         Node *u = que[l++];
68         REP(i, 26) {
69             if(u->ch[i]) {
70                 if(--u->ch[i]->deg == 0 && u->ch[i]
71 ]->
72                 tag != 1) {
73                     u->ch[i]->tag = 1;
74                     que[r++] = u->ch[i];
75                     vec.PB(u->ch[i]);
76                 }
77             }
78         }
79     }
80     for(int i = SZ(vec) - 1; i >= 0; i--) {
81         Node *u = vec[i];
82         if(T) {
83             if(u->fail) u->fail->val += u->val;
84         }
85         else u->val = 1;
86     }
87     root->val = 0;
88     for(int i = SZ(vec) - 1; i >= 0; i--) {
89         Node *u = vec[i];
90         u->dp = u->val;
91         REP(j, 26) {
92             if(u->ch[j]) u->dp += u->ch[j]->dp;
93         }
94     }
95     REP(i, 26) {
96         if(root->ch[i]) root->dp += root->ch[i]->dp;
97     }
98     inline void solve(int k) {
99         Node *p = root;
100         if(k > p->dp || k <= 0) {
101             puts("-1");
102             return;
103         }
104         while(k > 0) {
105             int flag = 0;
106             REP(i, 26) {
107                 if(!p->ch[i]) continue;
108                 if(k <= p->ch[i]->dp) {
109                     putchar('a' + i);
110                     k -= p->ch[i]->val;
111                     p = p->ch[i];
112                     flag = 1;
113                     break;
114                 }
115             }

```

```

116         else k -= p->ch[i]->dp;
117     }
118     if(!flag) break;
119 }
120 }
121 int main() {
122     scanf("%s", s);
123     int n = strlen(s);
124     N = n;
125     init();
126     REP(i, n) insert(s[i]);
127     int K;
128     scanf("%d", &T, &K);
129     bfs();
130     solve(K);
131     return 0;
132 }

```

### 8.3 Palindromic Tree

```

1 #include<bits/stdc++.h>
2 #include<unistd.h>
3 using namespace std;
4 #define F first
5 #define S second
6 #define MP make_pair
7 #define PB push_back
8 #define IOS ios_base::sync_with_stdio(0); cin.tie(0)
9 #define SZ(x) ((int)((x).size()))
10 #define ALL(x) begin(x),end(x)
11 #define REP(i,x) for (int i=0; i<(x); i++)
12 #define REP1(i,a,b) for (int i=(a); i<=(b); i++)
13
14 struct palindromic_tree{
15     struct node{
16         int next[26], fail, len;
17         int cnt, num, st, ed;
18         node(int l=0): fail(0), len(l), cnt(0), num(0){
19             for(int i=0; i<26; ++i) next[i]=0;
20         }
21     };
22     vector<node> state;
23     vector<char> s;
24     int last, n;
25
26     void init(){
27         state.clear();
28         s.clear();
29         last=1;
30         n=0;
31         state.push_back(0);
32         state.push_back(-1);
33         state[0].fail=1;
34         s.push_back(-1);
35     }
36     int get_fail(int x){
37         while(s[n-state[x].len-1]!=s[n])x=state[x].fail;
38         return x;
39     }
40     void add(int c){
41         s.push_back(c-'a');
42         ++n;
43         int cur=get_fail(last);
44         if(!state[cur].next[c]){
45             int now=state.size();
46             state.push_back(state[cur].len+2);
47             state[now].fail=state[get_fail(state[cur].fail
48 )].next[c];
49             state[cur].next[c]=now;
50             state[now].num=state[state[now].fail].num+1;
51         }
52         last=state[cur].next[c];
53         ++state[last].cnt;
54     }
55     int size(){
56         return state.size()-2;
57     }
58 }pt;
59 int main() {
60     string s;

```

```

61 cin >> s;
62 pt.init();
63 for (int i=0; i<SZ(s); i++) {
64     int prvsz = pt.size();
65     pt.add(s[i]);
66     if (prvsz != pt.size()) {
67         int r = i;
68         int l = r - pt.state[pt.last].len + 1;
69         cout << "Find pal @ [" << l << " " << r << "]"
70         : " << s.substr(l,r-l+1) << endl;
71     }
72 }
73 return 0;
74 }

```

## 8.4 MinLexicographicalRotate

```

1 string mcp(string s){
2     int n = s.length();
3     s += s;
4     int i=0, j=1;
5     while (i<n && j<n){
6         int k = 0;
7         while (k < n && s[i+k] == s[j+k]) k++;
8         if (s[i+k] <= s[j+k]) j += k+1;
9         else i += k+1;
10        if (i == j) j++;
11    }
12    int ans = i < n ? i : j;
13    return s.substr(ans, n);
14 }

```

## 8.5 ZvaluePalindromes

```

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

```

## 8.6 SuffixArray

```

1 int ss[N];
2 int heigh[N];
3 int sa[N];
4 int rank[N];
5 int length;
6 int val[30];
7 int c[N]; // counting sort array
8 int temp[2][N];
9 void suffix_array()
10 {
11     int A = 250;
12     int* rank = temp[0];
13     int* new_rank = temp[1];
14     for (int i=0; i<A; ++i) c[i] = 0;
15     for (int i=0; i<length; ++i) c[rank[i] = ss[i]]++;
16     for (int i=1; i<A; ++i) c[i] += c[i-1];
17     for (int i=length-1; i>=0; --i) sa[--c[ss[i]]] = i;
18     for (int n=1; n<length; n*=2)
19     {
20         for (int i=0; i<A; ++i) c[i] = 0;
21         for (int i=0; i<length; ++i) c[rank[i]]++;
22         for (int i=1; i<A; ++i) c[i] += c[i-1];
23         int* sa2 = new_rank;
24         int r = 0;
25         for (int i=length-n; i<length; ++i)
26             sa2[r++] = i;
27         for (int i=0; i<length; ++i)
28             if (sa[i] >= n)
29                 sa2[r++] = sa[i] - n;
30         for (int i=length-1; i>=0; --i)
31             sa[--c[rank[sa2[i]]]] = sa2[i];
32         new_rank[sa[0]] = r = 0;

```

```

33         for (int i=1; i<length; ++i)
34         {
35             if (!(rank[sa[i-1]] == rank[sa[i]] &&
36                 sa[i-1]+n < length && // stable
37                 rank[sa[i-1]+n] == rank[sa[i]+n]))
38                 r++;
39             new_rank[sa[i]] = r;
40         }
41         swap(rank, new_rank);
42         if (r == length-1) break;
43         A = r + 1;
44     }
45 }
46 void lcp_array()
47 {
48     for (int i=0; i<length; ++i)
49         rank[sa[i]] = i;
50
51     for (int i=0, lcp=0, h=0; i<length; i++)
52         if (rank[i] == 0)
53             heigh[0] = 0;
54         else
55         {
56             int j = sa[rank[i]-1];
57             if (lcp > 0) lcp=val[ss[i-1]-'a'],h--;
58             while (ss[i+h] == ss[j+h]) lcp+=val[ss[i
59             +h]-'a'],h++;
59             heigh[rank[i]] = lcp;
60         }
61 }

```

## 8.7 Zvalue

```

1 inline void z_alg1(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]=r>i?min(r-i+1,z[z[l]-(r-i+1)]):0;
6         while(i+z[i]<len&&s[z[i]]==s[i+z[i]])++z[i];
7         if(i+z[i]-1>r)r=i+z[i]-1,l=i;
8     }
9 }

```

## 9 Math

### 9.1 MillerRabin

```

1 // 4759123141 2, 7, 61
2 //2^64 2, 325, 9375, 28178, 450775, 9780504,
3 //1795265022
4 bool Isprime(LL n)
5 {
6     if (n == 2) return true;
7     if (n < 2 || n % 2 == 0) return false;
8     LL u = n - 1, t = 0;
9     while (u % 2 == 0) {u >>= 1; t++;}
10    LL sprp[7] = {2, 325, 9375, 28178, 450775,
11    9780504, 1795265022};
12    for (int k=0; k<7; ++k)
13    {
14        LL a = sprp[k] % n;
15        if (a == 0 || a == 1 || a == n-1) continue;
16        long long x = f_pow(a, u, n);
17        if (x == 1 || x == n-1) continue;
18        for (int i = 0; i < t-1; i++)
19        {
20            x = f_pow(x, 2, n);
21            if (x == 1) return false;
22            if (x == n-1) break;
23        }
24        if (x == n-1) continue;
25        return false;
26    }
27    return true;
28 }

```

### 9.2 Simplex



```

1 const int maxn = 111;
2 const int maxm = 111;
3 const double eps = 1E-10;
4
5 double a[maxn][maxm], b[maxn], c[maxm], d[maxn][maxm];
6 double x[maxm];
7 int ix[maxn + maxm]; // !!! array all indexed from 0
8 // max{cx} subject to {Ax<=b, x>=0}
9 // n: constraints, m: vars !!!
10 // x[] is the optimal solution vector
11 //
12 // usage :
13 // value = simplex(a, b, c, N, M);
14 double simplex(double a[maxn][maxm], double b[maxn],
15     double c[maxm], int n, int m) {
16     ++m;
17     int r = n, s = m - 1;
18     memset(d, 0, sizeof(d));
19     for (int i = 0; i < n + m; ++i) ix[i] = i;
20     for (int i = 0; i < n; ++i) {
21         for (int j = 0; j < m - 1; ++j)
22             d[i][j] = -a[i][j];
23         d[i][m - 1] = 1;
24         d[i][m] = b[i];
25         if (d[r][m] > d[i][m]) r = i;
26     }
27     for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
28     d[n + 1][m - 1] = -1;
29     for (double dd;; ) {
30         if (r < n) {
31             int t = ix[s];
32             ix[s] = ix[r + m]; ix[r + m] = t;
33             d[r][s] = 1.0 / d[r][s];
34             for (int j = 0; j <= m; ++j)
35                 if (j != s) d[r][j] *= -d[r][s];
36             for (int i = 0; i <= n + 1; ++i)
37                 if (i != r) {
38                     for (int j = 0; j <= m; ++j)
39                         if (j != s)
40                             d[i][j] += d[r][j] * d[i][s];
41                 }
42             r = -1; s = -1;
43             for (int j = 0; j < m; ++j)
44                 if (s < 0 || ix[s] > ix[j]) {
45                     if (d[n + 1][j] > eps || (d[n + 1][j]
46 ] > -eps && d[n][j] > eps)) s = j;
47                 }
48             if (s < 0) break;
49             for (int i = 0; i < n; ++i) if (d[i][s] < -eps)
50             {
51                 if (r < 0 || (dd = d[r][m] / d[r][s] - d
52 [i][m] / d[i][s]) < -eps || (dd < eps && ix[r +
53 m] > ix[i + m])) r = i;
54             }
55             if (r < 0) return -1; // not bounded
56         }
57         if (d[n + 1][m] < -eps) return -1; // not
58 executable
59 double ans = 0;
60 for (int i = 0; i < m; i++) x[i] = 0;
61 for (int i = m; i < n + m; ++i) { // the missing
62     enumerated x[i] = 0
63     if (ix[i] < m - 1)
64     {
65         ans += d[i - m][m] * c[ix[i]];
66         x[ix[i]] = d[i - m][m];
67     }
68 }
69 return ans;
70 }

```

### 9.3 Theorem

```

1 /*
2 Lucas's Theorem:
3 For non-negative integer n,m and prime P,
4 C(m,n) mod P = C(m/P,n/P) * C(m%P,n%P) mod P

```

```

5 -----
6 Pick's Theorem
7 A = i + b/2 - 1
8 */

```

### 9.4 Prime

```

1 /*
2 * 12721
3 * 13331
4 * 14341
5 * 75577
6 * 123457
7 * 222557
8 * 556679
9 * 999983
10 * 1097774749
11 * 1076767633
12 * 100102021
13 * 999997771
14 * 1001010013
15 * 1000512343
16 * 987654361
17 * 999991231
18 * 999888733
19 * 98789101
20 * 987777733
21 * 999991921
22 * 1010101333
23 * 1010102101
24 * 1000000000039
25 * 100000000000037
26 * 2305843009213693951
27 * 4611686018427387847
28 * 9223372036854775783
29 * 18446744073709551557
30 */

```

### 9.5 FFT

```

1 #define N 524288
2 #define pi acos(-1)
3 typedef complex<double> C;
4 int n,m,i,t,g[N];
5 C a[N],b[N];
6 void FFTinit(){
7     for (i=1;i<N;i++) g[i]=g[i>>1]>>1|((i&1)<<18);
8 }
9 void FFT(C *a,int f)
10 {
11     int i,j,k,p;
12     for (i=0;i<N;i++)
13         if (g[i]>i) swap(a[i],a[g[i]]);
14     for (i=1;i<N;i<=1)
15     {
16         C e(cos(pi/i),f*sin(pi/i));
17         for (j=0;j<N;j+=i<1)
18         {
19             C w(1,0);for (k=0;k<i;k++,w*=e)
20             {
21                 C x=a[j+k],y=w*a[j+k+i];
22                 a[j+k]=x+y;a[j+k+i]=x-y;
23             }
24         }
25     }
26 }
27 int res[400005];
28 int main()
29 {
30     FFTinit();
31     FFT(a,1);
32     FFT(b,1);
33     for(i=0;i<N;i++) a[i]=a[i]*b[i];
34     FFT(a,-1);
35     for (i=0;i<n+m;i++)
36         (int)a[i].real()/N+0.5)
37 }

```

## 9.6 Crt Solve2

```

1 ll a[10],n[10],k,k2;
2 int cs;
3 vector<PLL> v[100];
4 int srt(PLL a,PLL b){
5     return a.Y>b.Y;
6 }
7 PLL extgcd(ll a,ll b){
8     if(b==0) return mp(1,0);
9     ll p;
10    PLL q;
11    p = a/b;
12    q = extgcd(b,a%b);
13    return mp(q.Y,q.X-q.Y*p);
14 }
15 ll crt (){
16     ll i,alln,mf,ans,mi,ci;
17     PII f;
18     alln = 1;
19     ans = 0;
20     for(i=0;i<k;i++) alln *= n[i];
21     for(i=0;i<k;i++){
22         mi = alln/n[i];
23         mf = extgcd(mi,n[i]).X; // m[i]*mf % n[i] =
24         1
25         ci = mi*(mf % n[i]); // m[i] * (mf % n[i])
26         ans = (ans + (a[i]*ci))%alln + alln)%alln;
27     }
28     return (ans==0?alln:ans);
29 }
30 int chg(){
31     ll f,mi,xa,xm,c;
32     REP(i,k){
33         f = n[i];
34         REP1(j,2,f+1){
35             c = 0;
36             mi = 1;
37             while(f%j == 0){
38                 f/=j;
39                 c++;
40                 mi*=j;
41             }
42             if(c)v[j].pb(mp(a[i]%mi,mi));
43         }
44     }
45     k = 0;
46     REP(i,100){
47         if(LE(v[i])){
48             sort(ALL(v[i]),srt);
49             REP(j,LE(v[i])){
50                 xa = v[i][j].X;
51                 xm = v[i][j].Y;
52                 if(v[i][0].X % xm != xa % xm)
53                     return 0;
54             }
55             a[k] = v[i][0].X;
56             n[k] = v[i][0].Y;
57             k++;
58         }
59     }
60     return 1;
61 }

```

## 9.7 FWT

```

1 void FWT(int *x,int inv) {
2     for(int i=1;i<lim;i<=1){
3         for(int j=0;j<lim;++j)/* or */
4             if(j&i)
5                 x[j]= inv ? x[j]-x[j^i] : x[j]+x[j^i];
6     }
7     for(int j=0;j<lim;j+=(i<1))/* and */
8         for(int k=0;k<i;++k)
9             x[j+k]=inv ? x[j+k]-x[j+k+i] : x[j+k]
10            ]+x[j+k+i];
11     for(int j=0;j<lim;j+=(i<1))/* xor */
12         for(int k=0;k<i;++k) {
13             int y=x[j+k],z=x[j+k+i];
14             x[j+k]=inv ? (y+z)/2 : y+z;
15             x[j+k+i]=inv ? (y-z)/2 : y-z;
16         }
17 }

```

```

15     }
16 }

```

## 9.8 Extgcd

```

1 typedef pair<int, int> pii;
2 pii gcd(int a, int b){
3     if(b == 0) return mp(1, 0);
4     else{
5         int p = a / b;
6         pii q = gcd(b, a % b);
7         return make_pair(q.y, q.x - q.y * p);
8     }
9 }

```

## 9.9 Pollard'sRho

```

1 // does not work when n is prime
2 inline LL f(LL x, LL mod) {
3     return (x * x % mod + 1) % mod;
4 }
5 inline LL pollard_rho(LL n) {
6     if(!(n&1)) return 2;
7     while(true) {
8         LL y = 2, x = rand() % (n - 1) + 1, res = 1;
9         for(int sz = 2; res == 1; sz *= 2) {
10             for(int i = 0; i < sz && res <= 1; i++) {
11                 x = f(x, n);
12                 res = __gcd(abs(x - y), n);
13             }
14             y = x;
15         }
16         if (res != 0 && res != n) return res;
17     }
18 }

```

## 10 monge

$$i \leq i' < j \leq j'$$

$$m(i, j) + m(i', j') \leq m(i', j) + m(i, j')$$

$$k(i, j - 1) \leq k(i, j) \leq k(i + 1, j)$$

## 11 四心

$$\frac{sa \cdot A + sb \cdot B + sc \cdot C}{sa + sb + sc}$$

外心  $\sin 2A : \sin 2B : \sin 2C$

內心  $\sin A : \sin B : \sin C$

垂心  $\tan A : \tan B : \tan C$

重心  $1 : 1 : 1$

## 12 Runge-Kutta

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$$

$$k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$$

$$k_4 = f(t_n + h, y_n + hk_3)$$

## 13 Householder Matrix

$$I - 2 \frac{vv^T}{v^T v}$$

## 14 Simpson's-rule

$$\int_a^b f(x)dx = \frac{b-a}{6}(f(a) + 4f(\frac{a+b}{2}) + f(b))$$