

# NCTU\_TaNoShiI

## Contents

<b>1 Basic</b>	<b>1</b>
1.1 Vimrc	1
1.2 Default	1
1.3 Print	1
<b>2 DataStructure</b>	<b>1</b>
2.1 PersistentTreap	1
2.2 Pbds Kth	2
2.3 PbdsHeap	2
2.4 Heavy-LightDecomposition	2
2.5 KDtree	3
2.6 LCT	4
<b>3 Flow</b>	<b>4</b>
3.1 Minmumwiegthmatchclique	4
3.2 CostFlow	5
3.3 MincutTree	5
3.4 Dinic	6
3.5 GeneralGraphmatch	6
3.6 KM	7
3.7 SWmincut	7
<b>4 Geometry</b>	<b>7</b>
4.1 Circleintersection	7
4.2 Fermat's Point	7
4.3 Pointoperators	8
4.4 3DConvexHull	8
4.5 Halfplaneintersection	9
4.6 ConvexHull	9
4.7 Triangulation	9
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	17
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>19</b>
7.1 Annealing	19
7.2 DLX	19
7.3 MahattanMST	20
7.4 MoOnTree	20
7.5 Det	21
<b>8 String</b>	<b>21</b>
8.1 AC	21
8.2 SuffixAutomata	21

8.3 Palindromic Tree	22
8.4 MinLexicographicalRotate	23
8.5 ZvaluePalindromes	23
8.6 SuffixArray	23
8.7 Zvalue	23

## 9 Math 23

9.1 MillerRabin	23
9.2 Simplex	24
9.3 Theorem	24
9.4 Prime	24
9.5 FFT	24
9.6 FWT	25
9.7 Extgcd	25
9.8 Pollard'sRho	25

## 10 無權邊的生成樹個數 Kirchhoff's Theorem 25

## 11 monge 25

## 12 四心 25

## 13 Runge-Kutta 25

## 14 Householder Matrix 25

## 1 Basic

### 1.1 Vimrc

```
1 set ts=4
2 set sw=4
3 set et
4 set ai
5 set nu
6
7 map <F9> :w<LF>:!g++ -O2 -g -std=c++11 -o %.out % &&
   echo "----Start----" && ./%.out<LF>
8 imap <F9> <ESC><F9>
```

### 1.2 Default

```
1 #include<bits/stdc++.h>
2 #define mp(a,b) make_pair((a),(b))
3 #define pli pair<int,int>
4 #define pdd pair<double,double>
5 #define pll pair<LL,LL>
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<int,int>
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;
```

### 1.3 Print

```
1 cat -n "%s" > tmp.print
2 lpr tmp.print
```

## 2 DataStructure

### 2.1 PersistentTreap

```

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(&nil), r(&nil), size(0) {}
8     Treap (char _val) :
9         l(&nil), r(&nil), val(_val), size(1) {}
10 } Treap::nil, Treap::mem[MEM], *Treap::pmem = Treap::
11 mem;
12 int size(const Treap *t) { return t->size; }
13 void pull(Treap *t) {
14     if (!size(t)) return;
15     t->size = size(t->l) + size(t->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;
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

```

```

78         Treap *tl, *tm, *tr;
79         cin >> v >> p >> c;
80         split(rt[v], p-1, tl, tm);
81         split(tm, c, tm, tr);
82         print(tm);
83         cout << "n";
84     }
85 }
86 return 0;
87 }

```

## 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){

```

```

23 int fr,bk;
24 fr=bk=0; que[bk++] = 1; fa[st]=st; dep[st]=0;
25 while (fr < bk){
26     int u=que[fr++];
27     for (vector<int>::iterator it=v[u].begin();
28         it!=v[u].end();it++){
29         if (*it == fa[u]) continue;
30         que[bk++] = *it;
31         dep[*it] = dep[u]+1;
32         fa[*it] = u;
33     }
34     for (int i=bk-1,u,pos; i>=0; i--){
35         u = que[i]; sz[u] = 1; pos = -1;
36         for (vector<int>::iterator it=v[u].begin();
37             it!=v[u].end();it++){
38             if (*it == fa[u]) continue;
39             sz[u] += sz[*it];
40             if (pos==-1 || sz[*it]>sz[pos]) pos=*it;
41             if (pos == -1) belong[u] = u;
42             else belong[u] = belong[pos];
43             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         }
88     }
89 }

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
15 .x; }
16 static bool cmpy(Node& a, Node& b){ return a.y<b
17 .y; }
18 void init(vector<pair<int,int>> ip) {
19     n = ip.size();
20     for (int i=0; i<n; i++) {
21         tree[i].id = i;
22         tree[i].x = ip[i].first;
23         tree[i].y = ip[i].second;
24     }
25     root = build_tree(0, n-1, 0);
26 }
27 Node* build_tree(int L, int R, int dep) {
28     if (L>R) return nullptr;
29     int M = (L+R)/2;
30     tree[M].f = dep%2;
31     nth_element(tree+L, tree+M, tree+R+1, tree[M
32 ].f ?
33     cmpy : cmpx);
34     tree[M].x1 = tree[M].x2 = tree[M].x;
35     tree[M].y1 = tree[M].y2 = tree[M].y;
36     tree[M].L = build_tree(L, M-1, dep+1);
37     if (tree[M].L) {
38         tree[M].x1 = min(tree[M].x1, tree[M].L->
39 x1);
40         tree[M].x2 = max(tree[M].x2, tree[M].L->
41 x2);
42         tree[M].y1 = min(tree[M].y1, tree[M].L->
43 y1);
44         tree[M].y2 = max(tree[M].y2, tree[M].L->
45 y2);
46     }
47     tree[M].R = build_tree(M+1, R, dep+1);
48     if (tree[M].R) {
49         tree[M].x1 = min(tree[M].x1, tree[M].R->
50 x1);
51         tree[M].x2 = max(tree[M].x2, tree[M].R->
52 x2);
53         tree[M].y1 = min(tree[M].y1, tree[M].R->
54 y1);
55         tree[M].y2 = max(tree[M].y2, tree[M].R->
56 y2);
57     }
58     return tree+M;
59 }
60 int touch(Node* r, int x, int y, long long d2){
61     long long dis = sqrt(d2)+1;
62     if (x<r->x1-dis || x>r->x2+dis || y<r->y1-
63 dis || y>
64 r->y2+dis)
65     return 0;
66     return 1;
67 }
68 void nearest(Node* r, int x, int y, int &mID,
69 long
70 long &md2) {
71     if (!r || !touch(r, x, y, md2)) return;
72     long long d2 = dis2(r->x, r->y, x, y);
73     if (d2 < md2 || (d2 == md2 && mID < r->id))
74     {
75         mID = r->id;
76         md2 = d2;
77     }
78     // search order depends on split dim
79     if ((r->f == 0 && x < r->x) ||
80         (r->f == 1 && y < r->y)) {
81         nearest(r->L, x, y, mID, md2);
82         nearest(r->R, x, y, mID, md2);
83     } else {
84         nearest(r->R, x, y, mID, md2);
85         nearest(r->L, x, y, mID, md2);
86     }
87 }
88 int query(int x, int y) {
89     int id = 1029384756;
90     long long d2 = 102938475612345678LL;
91     nearest(root, x, y, id, d2);
92     return id;
93 }

```

## 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;

```

```
79 }
80 }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 Minimunwieghtmatchclique

```
1 struct Graph {
2     // Minimum General Weighted Matching (Perfect Match) clique
3     static const int MXN = 105;
4     int n, edge[MXN][MXN];
5     int match[MXN], dis[MXN], onstk[MXN];
6     vector<int> stk;
7     void init(int _n) {
8         n = _n;
9         MEM(edge);
10    }
11    void add_edge(int u, int v, int w) {
12        edge[u][v] = edge[v][u] = w;
13    }
14    bool SPFA(int u){
15        if (onstk[u]) return true;
16        stk.pb(u);
17        onstk[u] = 1;
18        for (int v=0; v<n; v++){
```

```

19         if (u != v && match[u] != v && !onstk[v
20     ]){
21         int m = match[v];
22         if (dis[m] > dis[u] - edge[v][m] +
23     edge[u][v]){
24             dis[m] = dis[u] - edge[v][m] +
25     edge[u][v];
26             onstk[v] = 1;
27             stk.pb(v);
28             if (SPFA(m)) return true;
29             stk.pop_back();
30             onstk[v] = 0;
31         }
32     }
33     onstk[u] = 0;
34     stk.pop_back();
35     return false;
36 }
37 int solve() {
38     // find a match
39     for (int i=0; i<n; i+=2){
40         match[i] = i+1;
41         match[i+1] = i;
42     }
43     while (true){
44         int found = 0;
45         MEM(dis); MEM(onstk);
46         for (int i=0; i<n; i++){
47             stk.clear();
48             if (!onstk[i] && SPFA(i)){
49                 found = 1;
50                 while (stk.size()>=2){
51                     int u = stk.back(); stk.
52     pop_back();
53                     int v = stk.back(); stk.
54     pop_back();
55                     match[u] = v;
56                     match[v] = u;
57                 }
58             }
59             if (!found) break;
60         }
61         int ret = 0;
62         for (int i=0; i<n; i++){
63             ret += edge[i][match[i]];
64         }
65         ret /= 2;
66         return ret;
67     }
68 }
69 }graph;

```

### 3.2 CostFlow

```

1 struct CostFlow {
2     static const int MXN = 205;
3     static const long long INF = 102938475610293847
4     LL;
5     struct Edge {
6         int v, r;
7         long long f, c;
8         Edge(int a, int b, int _c, int d):v(a),r(b),f(
9     _c),c(d){
10     }
11     };
12     int n, s, t, prv[MXN], prvl[MXN], inq[MXN];
13     long long dis[MXN], fl, cost;
14     vector<Edge> E[MXN];
15     void init(int _n, int _s, int _t) {
16         n = _n; s = _s; t = _t;
17         for (int i=0; i<n; i++) E[i].clear();
18         fl = cost = 0;
19     }
20     void add_edge(int u, int v, long long f, long
21     long c)
22     {
23         E[u].pb(Edge(v, E[v].size(), f, c));
24         E[v].pb(Edge(u, E[u].size()-1, 0, -c));
25     }
26     pll flow() {
27         while (true) {

```

```

28         for (int i=0; i<n; i++) {
29             dis[i] = INF;
30             inq[i] = 0;
31         }
32         dis[s] = 0;
33         queue<int> que;
34         que.push(s);
35         while (!que.empty()) {
36             int u = que.front(); que.pop();
37             inq[u] = 0;
38             for (int i=0; i<E[u].size(); i++) {
39                 int v = E[u][i].v;
40                 long long w = E[u][i].c;
41                 if (E[u][i].f > 0 && dis[v] >
42     dis[u] + w) {
43                     prv[v] = u; prvL[v] = i;
44                     dis[v] = dis[u] + w;
45                     if (!inq[v]) {
46                         inq[v] = 1;
47                         que.push(v);
48                     }
49                 }
50             }
51         }
52         if (dis[t] == INF) break;
53         long long tf = INF;
54         for (int v=t, u, l; v!=s; v=u) {
55             u=prv[v]; l=prvL[v];
56             tf = min(tf, E[u][l].f);
57         }
58         for (int v=t, u, l; v!=s; v=u) {
59             u=prv[v]; l=prvL[v];
60             E[u][l].f -= tf;
61             E[v][E[u][l].r].f += tf;
62         }
63         cost += tf * dis[t];
64         fl += tf;
65     }
66     return {fl, cost};
67 }
68 }
69 }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         }
38         swap(s2,s);
39         int x=dc(s1,0);

```



```

38 int y=dc(s2,1);
39 vt[x].pb(mp(y,res));
40 vt[y].pb(mp(x,res));
41 if(flag==0)
42 return x;
43 else
44 return y;
45 }

```

### 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)
4         :v(a),f(b),re(c){}};
5     int n,s,t,level[MXN];
6     vector<Edge> E[MXN];
7     void init(int _n, int _s, int _t){
8         n = _n; s = _s; t = _t;
9         for (int i=0; i<=n; i++) E[i].clear();
10    }
11    void add_edge(int u, int v, int f){
12        E[u].pb(Edge(v,f,E[u].size()));
13        E[v].pb(Edge(u,0,E[v].size()-1)); //direct
14    }
15    bool BFS(){
16        MEMS(level);
17        queue<int> que;
18        que.push(s);
19        level[s] = 0;
20        while (!que.empty()){
21            int u = que.front(); que.pop();
22            for (auto it : E[u]){
23                if (it.f > 0 && level[it.v] == -1){
24                    level[it.v] = level[u]+1;
25                    que.push(it.v);
26                }
27            }
28        }
29        return level[t] != -1;
30    }
31    int DFS(int u, int nf){
32        if (u == t) return nf;
33        int res = 0;
34        for (auto &it : E[u]){
35            if (it.f > 0 && level[it.v] == level[u]
36                ]+1){
37                int tf = DFS(it.v, min(nf,it.f));
38                res += tf; nf -= tf; it.f -= tf;
39                E[it.v][it.re].f += tf;
40                if (nf == 0) return res;
41            }
42        }
43        if (!res) level[u] = -1;
44        return res;
45    }
46    int flow(int res=0){
47        while ( BFS() )
48            res += DFS(s,2147483647);
49        return res;
50    }
51 }

```

### 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        void aug() {
92            int u,v,w;
93            u = ed;
94            while(u > 0) {
95                v = bk[u];
96                w = pr[v];

```

```

96         pr[v] = u;
97         pr[u] = v;
98         u = w;
99     }
100 }
101 int solve() {
102     memset(pr,0,sizeof(pr));
103     for(int u = 1; u <= V; u++)
104         if(pr[u] == 0) {
105             st = u;
106             flow();
107             if(ed > 0) {
108                 aug();
109                 ans ++;
110             }
111         }
112     return ans;
113 }
114 }gp;

```

### 3.6 KM

```

1 typedef pair<long long, long long> pll;
2 struct KM{
3     // Maximum Bipartite Weighted Matching (Perfect Match)
4     static const int MXN = 650;
5     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     void add_edge(int x, int y, int w){ // long long
16         edge[x][y] = w;
17     }
18     bool DFS(int x){
19         vx[x] = 1;
20         for (int y=0; y<n; y++){
21             if (vy[y]) continue;
22             if (lx[x]+ly[y] > edge[x][y]){
23                 slack[y] = min(slack[y], lx[x]+ly[y]
24 ]-edge[x][y]);
25             } else {
26                 vy[y] = 1;
27                 if (match[y] == -1 || DFS(match[y]))
28                     match[y] = x;
29                 return true;
30             }
31         }
32     }
33     return false;
34 }
35 int solve(){
36     fill(match, match+n, -1);
37     fill(lx, lx+n, -INF);
38     fill(ly, ly+n, 0);
39     for (int i=0; i<n; i++)
40         for (int j=0; j<n; j++)
41             lx[i] = max(lx[i], edge[i][j]);
42     for (int i=0; i<n; i++){
43         fill(slack, slack+n, INF);
44         while (true){
45             fill(vx, vx+n, 0);
46             fill(vy, vy+n, 0);
47             if (DFS(i)) break;
48             int d = INF; // long long
49             for (int j=0; j<n; j++)
50                 if (!vy[j]) d = min(d, slack[j]);
51             for (int j=0; j<n; j++){
52                 if (vx[j]) lx[j] -= d;
53                 if (vy[j]) ly[j] += d;
54                 else slack[j] -= d;
55             }
56         }
57     }

```

```

57     }
58     int res=0;
59     for (int i=0; i<n; i++){
60         res += edge[match[i]][i];
61         return res;
62     }
63 }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;
36 }
37 inline double dist2(point a, point b) {
38     return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
39 }
40 inline point rot(point o, point p) {
41     p.x -= o.x, p.y -= o.y;
42     return point(0.x+p.x*ct-p.y*st, 0.y+p.x*st+p.y*ct);
43 }
44 inline line cln(point a, point b) {
45     return line(a.y-b.y, b.x-a.x, calc(a.y-b.y, b.x-a.x, a));
46 }
47 inline point ntse(line f, line g) {
48     double det = f.a*g.b-g.a*f.b, dx = f.c*g.b-g.c*f.b,
49     dy = f.a*g.c-g.a*f.c;
50     return point(dx/det, dy/det);
51 }
52 inline point fema(point a, point b, point c) {
53     double la = dist2(b, c), lb = dist2(a, c), lc =
54     dist2(a, b);
55     double sa = sqrt(la), sb = sqrt(lb), sc = sqrt(lc);
56     if ((lb+lc-la)/(2.0*sb*sc) < -0.5 + eps)
57         return a;
58     if ((la+lc-lb)/(2.0*sa*sc) < -0.5 + eps)
59         return b;
60     if ((la+lb-lc)/(2.0*sa*sb) < -0.5 + eps)
61         return c;
62     point t1 = rot(a, b), t2 = rot(b, a);
63     if (dist2(c, t1) < dist2(c, t2)) swap(t1, t2);
64     point s1 = rot(b, c), s2 = rot(c, b);
65     if (dist2(a, s1) < dist2(a, s2)) swap(s1, s2);
66     return ntse(cln(c, t1), cln(a, s1));
67 }
68 int main() {
69     ios_base::sync_with_stdio(false);
70     cin.tie(NULL);
71     point a, b, c;
72     cin >> a >> b >> c;
73     cout << setprecision(10) << fixed << fema(a, b, c)
74     << '\n';
75 }

```

### 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(
26     p-o, q-o); }
27 pdd operator * (double f, cpdd &p) { return p*f; }
28 //!! 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 {
23     int n = SZ(pt);
24     REP(i,n) REP(j,n)
25         flag[i][j] = 0;
26     vector<Face> now;
27     now.push_back((Face){0,1,2});
28     now.push_back((Face){2,1,0});
29     int ftop = 0;
30     for (int i=3; i<n; i++){
31         ftop++;
32         vector<Face> next;
33         REP(j, SZ(now)) {
34             Face& f=now[j];
35             ld d=(pt[i]-pt[f.a]).dot(ver(pt[f.a], pt[f.b], pt[f.c]));
36             if (d <= 0) next.push_back(f);
37             int ff = 0;
38             if (d > 0) ff=ftop;
39             else if (d < 0) ff=-ftop;
40             flag[f.a][f.b] = flag[f.b][f.c] = flag[f.c][f.a] = ff;
41         }
42         REP(j, SZ(now)) {
43             Face& f=now[j];
44             if (flag[f.a][f.b] > 0 and flag[f.a][f.b] != flag[f.b][f.a])

```



```

48     next.push_back((Face){f.a,f.b,i});
49     if (flag[f.b][f.c] > 0 and flag[f.b][f.c]
] != flag
50     [f.c][f.b])
51     next.push_back((Face){f.b,f.c,i});
52     if (flag[f.c][f.a] > 0 and flag[f.c][f.a]
] != flag
53     [f.a][f.c])
54     next.push_back((Face){f.c,f.a,i});
55     }
56     now=next;
57 }
58 return now;
59 }

```

## 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         if (L < R) p[R-1] = intersection(l[R], l[R
-1]);
54     }
55     while (L < R && cross(l[L], p[R-1]) < 0) R--;
56     if (R-L <= 1) return Polygon();
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&&(p[i].y-ans[now-1].y)*(ans[now].x-
ans[now-1].x)<=(p[i].x-ans[now-1].x)*(ans[now].
y-ans[now-1].y)){
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
cross(d, a, c) * cross(d, b, c) < 0;
18 }
19 const double EPS = 1e-12;
20 struct Triangulation {
21     static const int MXN = 1e5+5;
22     int N;
23     vector<int> ord;
24     vector<pdd> pts;
25     set<int> E[MXN];
26     vector<vector<int>> solve(vector<pdd> p) {
27         N = SZ(p);
28         ord.resize(N);
29         for (int i=0; i<N; i++) {
30             E[i].clear();
31             ord[i] = i;
32         }
33         sort(ALL(ord), [&p](int i, int j) {

```

```

38         return p[i] < p[j];
39     });
40     pts.resize(N);
41     for (int i=0; i<N; i++) pts[i] = p[ord[i]];
42     go(0, N);
43     vector<vector<int>> res(N);
44     for (int i=0; i<N; i++) {
45         int o = ord[i];
46         for (auto x: E[i]) {
47             res[o].PB(ord[x]);
48         }
49     }
50     return res;
51 }
52 void add_edge(int u, int v) {
53     E[u].insert(v);
54     E[v].insert(u);
55 }
56 void remove_edge(int u, int v) {
57     E[u].erase(v);
58     E[v].erase(u);
59 }
60 void go(int l, int r) {
61     int n = r - l;
62     if (n <= 3) {
63         for (int i=l; i<r; i++)
64             for (int j=i+1; j<r; j++) add_edge(i, j);
65         return;
66     }
67     int md = (l+r)/2;
68     go(l, md);
69     go(md, r);
70     int il = l, ir = r-1;
71     while (1) {
72         int nx = -1;
73         for (auto i: E[il]) {
74             double cs = cross(pts[il], pts[i],
75 pts[
76         ir]);
77             if (cs > EPS ||
78 (abs(cs) < EPS and abs(pts[i]-pts[
79         ir]) < abs(pts[il]-pts[ir]))) {
80                 nx = i;
81                 break;
82             }
83         }
84         if (nx != -1) {
85             il = nx;
86             continue;
87         }
88         for (auto i: E[ir]) {
89 pts[
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
AcThPaUNpPuAmCmBkCfEsFmMdNoLr 19
109             for (int i: E[il]) {
110                 if (cross(pts[il], pts[i], pts[ir])
111 < -
112                     EPS and
113 (nx == -1 or inCircle(pts[il], pts[
114             ir], pts[nx], pts[i]))) nx = i;
115             }
116             for (int i: E[ir]) {
117                 if (cross(pts[ir], pts[i], pts[il])

```

```

115         EPS and
116         (nx == -1 or inCircle(pts[il], pts[
117             ir], pts[nx], pts[i]))) nx = i,
118             is2 = 1;
119     }
120     if (nx == -1) break;
121     int a = il, b = ir;
122     if (is2) swap(a, b);
123     for (auto i: E[a]) {
124         if (intersect(pts[a], pts[i], pts[b
125             ],
126                 pts[nx])) {
127                 remove_edge(a, i);
128             }
129     }
130     if (is2) {
131         add_edge(il, nx);
132         ir = nx;
133     } else {
134         add_edge(ir, nx);
135         il = nx;
136     }
137 }
138 } tri;

```

#### 4.8 K-closest 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         ) {

```

```

49     Q.pop();
50 }
51 while (!Q.empty()) {
52     delta2 = min(delta2, dist2(p[i], Q.front()));
53     Q.pop();
54 }
55 }
56 return delta2;
57 }
58 ll find_distance(int l, int r) {
59     if (r - l <= 3000) {
60         ll ans = 0x3f3f3f3f3f3f3f3f;
61         for (int i = l; i < r; ++i)
62             for (int j = i+1; j < r; ++j)
63                 ans = min(ans, dist2(p[i], p[j]));
64         return ans;
65     }
66     int m = (l+r)/2;
67     ll delta2 = min(find_distance(l, m), find_distance(m, r));
68     return min(delta2, closet_point(l, m, r, delta2));
69 }
70 int main() {
71     ios_base::sync_with_stdio(false);
72     cin.tie(NULL);
73     int n;
74     cin >> n >> k;
75     F(n) cin >> p[i];
76     sort(p, p+n);
77     find_distance(0, n);
78     cout << PQ.top() << '\n';
79 }

```

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

```

## 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));
18 }

```

# 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     int visited[N];
64     void addtree(int u,int t1,int v,int t2)
65     {
66         nxt[u][t1]=v; nxt[v][t2]=u;
67     }
68     void findnxt(int u,int v,int& u1,int& v1)
69     {
70         u1=nxt[u][v^1];
71         if(nxt[u1][0]==u) v1=0;
72         else v1=1;
73     }
74     void walkup(int u,int v)
75     {
76         back[v]=u;
77         int v1=v,v2=v,u1=1,u2=0,z;
78         for (;;)
79         {
80             if(hash[v1]==u || hash[v2]==u) break;
81             hash[v1]=u;hash[v2]=u; z=max(v1,v2);
82             if(z>n)
83             {
84                 int p=fa[z-n];
85                 if(p!=u)
86                 {
87                     proots[p].insert(mp(-low[z-n], z
88                     ));
89                     v1=p,v2=p,u1=0,u2=1;
90                 }
91                 else break;
92             }
93             else
94             {
95                 findnxt(v1,u1,v1,u1);
96                 findnxt(v2,u2,v2,u2);
97             }
98         }
99     }
100     int topstack;
101     pair<int,int> stack[N];
102     int outer(int u,int v)
103     {
104         return ecp[v]<deep[u] || (SDlist[v].size()
105         && SDlist[v].begin()->first<deep[u]);
106     }
107     int inside(int u,int v)
108     {
109         return proots[v].size()>0 || back[v]==u;
110     }
111     int active(int u,int v)
112     {
113         return inside(u,v) || outer(u,v);
114     }

```

```

120     }
121     void push(int a,int b)
122     {
123         stack[++topstack]=mp(a,b);
124     }
125     void mergestack()
126     {
127         int v1,t1,v2,t2,s,s1;
128         v1=stack[topstack].first;t1=stack[topstack].
129         second;
130         topstack--;
131         v2=stack[topstack].first;t2=stack[topstack].
132         second;
133         topstack--;
134         s=nxt[v1][t1^1];
135         s1=(nxt[s][1]==v1);
136         nxt[s][s1]=v2;
137         nxt[v2][t2]=s;
138         SDlist[v2].erase( make_pair(low[v1-n],v1-n)
139         );
140         proots[v2].erase( make_pair(-low[v1-n],v1) )
141         ;
142     }
143     void findnxtActive(int u,int t,int& v,int& w1,
144     int S)
145     {
146         findnxt(u,t,v,w1);
147         while(u!=v && !active(S,v))
148             findnxt(v,w1,v,w1);
149     }
150     void walkdown(int S,int u)
151     {
152         topstack=0;
153         int t1,v=S,w1,x2,y2,x1,y1,p;
154         for(t1=0;t1<2;++t1)
155         {
156             findnxt(S,t1^1,v,w1);
157             while(v!=S)
158             {
159                 if(back[v]==u)
160                 {
161                     while(topstack>0) mergestack();
162                     addtree(S,t1,v,w1); back[v]=0;
163                 }
164                 if(proots[v].size())
165                 {
166                     push(v,w1);
167                     p=proots[v].begin()->second;
168                     findnxtActive(p,1,x1,y1,u);
169                     findnxtActive(p,0,x2,y2,u);
170                     if(active(u,x1) && !outer(u,x1))
171                         v=x1,w1=y1;
172                     else if(active(u,x2) && !outer(u
173                     ,x2))
174                         v=x2,w1=y2;
175                     else if(inside(u,x1) || back[x1
176                     ]==u)
177                         v=x1,w1=y1;
178                     else v=x2,w1=y2;
179                     push(p,v==x2);
180                 }
181                 else if(v>n || ( ecp[v]>=deep[u] &&
182                 !outer(u,v) ))
183                     findnxt(v,w1,v,w1);
184                 else if(v<=n && outer(u,v) && !
185                 topstack)
186                 {
187                     addtree(S,t1,v,w1); break;
188                 }
189                 else break;
190             }
191         }
192     }
193     int work(int u)
194     {

```

```

193     int v;
194     for (int i = 0; i < g[u].size(); ++i)
195         if (fa[v=g[u][i]]==u)
196         {
197             son[u].push_back(n+v);
198             proots[n+v].clear();
199             addtree(n+v, 1, v, 0);
200             addtree(n+v, 0, v, 1);
201         }
202     for (int i = 0; i < g[u].size(); ++i)
203         if (deep[v=g[u][i]]>deep[u]+1)
204             walkup(u, v);
205     topstack=0;
206     for (int i = 0; i < son[u].size(); ++i)
207         walkdown(son[u][i], u);
208     for (int i = 0; i < g[u].size(); ++i)
209         if (deep[v=g[u][i]]>deep[u]+1 && back[v])
210             return 0;
211     return 1;
212 }
213 void init(int _n)
214 {
215     n = _n;
216     m = 0;
217     for(int i=1; i<=2*n; ++i)
218     {
219         g[i].clear();
220         SDlist[i].clear();
221         son[i].clear();
222         proots[i].clear();
223         nxt[i][0]=nxt[i][1]=0;
224         fa[i]=0;
225         hash[i]=0; low[i]=ecp[i]=deep[i]=back[i]
226     ]=0;
227     q.clear();
228     }
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.
281             g[cur].end(), path[0]);
282         if (it != base.g[cur].end() && *it == path
283             [0])
284         {
285             //a cycle
286             int x = n+1;
287             for (int i = 0; i < 5; ++i) edges.pb(mp(
288                 x, path[i]));
289             build(x, _new);
290             if (_new.check_planar())
291             {
292                 for (int i = 0; i < maxp; ++i)
293                     printf("%d%c", path[i], i==maxp-1?'\n':' ');
294                 exit(0);
295             }
296             for (int i = 0; i < 5; ++i) edges.
297                 pop_back();
298         }
299     }
300     for (auto e : base.g[cur]) if (!vis[e]) dfs(
301         e);
302     vis[cur] = false;
303     --tp;
304 }
305 int main()
306 {
307     scanf("%d %d", &n, &m);
308     if (n <= 4)
309     {
310         assert(false);
311         puts("0"); return 0;
312     }
313     for (int i = 0; i < m; ++i)
314     {
315         int u, v; scanf("%d %d", &u, &v);
316         edges.pb(mp(u, v));
317     }
318     build(n, base);
319     if (!base.check_planar()) end();
320     for (int i = 1; i <= n; ++i)
321         sort(base.g[i].begin(), base.g[i].end());
322     for (int i = 1; i <= n; ++i)
323         dfs(i);
324     end();
325 }

```

## 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]
12 ];
13 Edge e[MAXE];
14 vector<int> edgeID, cycle, rho;
15 double d[MAXN][MAXN];
16 inline void bellman_ford() {
17     for(int i=0; i<n; i++) d[0][i]=0;
18     for(int i=0; i<n; i++) {
19         fill(d[i+1], d[i+1]+n, inf);
20         for(int j=0; j<m; j++) {
21             int v = e[j].v, u = e[j].u;

```



```

21         if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j]
22         ].c) {
23             d[i+1][u] = d[i][v]+e[j].c;
24             prv[i+1][u] = v;
25             prve[i+1][u] = j;
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]
38             ]-d[k][i])
39             /(n-k));
40             else avg=max(avg,inf);
41         }
42         if (avg < mmc) tie(mmc, st) = tie(avg, i);
43     }
44     MEM(vst); edgeID.clear(); cycle.clear(); rho.
45     clear();
46     for (int i=n; !vst[st]; st=prv[i--][st]) {
47         vst[st]++;
48         edgeID.pb(prve[i][st]);
49         rho.pb(st);
50     }
51     while (vst[st] != 2) {
52         int v = rho.back(); rho.pop_back();
53         cycle.pb(v);
54         vst[v]++;
55     }
56     reverse(edgeID.begin(),edgeID.end());
57     edgeID.resize(cycle.size());
58     return mmc;
59 }

```

### 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|
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
12     [v]; }
13
14     int eval(int u) {
15         if(mom[u] == u) return u;
16         int res = eval(mom[u]);
17         if(cmp(sdom[mn[mom[u]]],sdom[mn[u]]))
18             mn[u] = mn[mom[u]];
19         return mom[u] = res;
20     }
21 }

```

```

20 void init(int _n, int _s) {
21     n = _n;
22     s = _s;
23     REP1(i,1,n) {
24         g[i].clear();
25         pred[i].clear();
26         idom[i] = 0;
27     }
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[
57             mn[v]];
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     }
68     REP1(i,2,ts) {
69         int u = nfd[i];
70         if(u == 0) continue;
71         if(idom[u] != sdom[u]) idom[u] = idom[idom[u]
72         ];
73     }
74 }
75 }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
8         (0),l(0),r(0){}
9         void down(){
10             w+=tag;
11             if(l)l->tag+=tag;
12             if(r)r->tag+=tag;
13             tag=0;
14         }
15     }mem[MAXN];
16     node *pq[MAXN*2],*E[MAXN*2];
17     int st[MAXN*2],id[MAXN*2],m,from[MAXN*2];
18     void init(int n){
19         for(int i=1;i<=n;++i){
20             pq[i]=E[i]=0;
21             st[i]=id[i]=i;
22             from[i]=0;
23         }
24     }
25 }

```

```

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 }
37 int find(int x,int *st){
38     return st[x]==x?x:st[x]=find(st[x],st);
39 }
40 LL build(int root,int n){
41     LL ans=0;int N=n,all=n;
42     for(int i=1;i<=N;++i){
43         if(i==root||!pq[i])continue;
44         while(pq[i]){
45             pq[i]->down(),E[i]=pq[i];
46             pq[i]=merge(pq[i]->l,pq[i]->r);
47             if(find(E[i]->u,id)!=find(i,id))break;
48         }
49         if(find(E[i]->u,id)==find(i,id))continue;
50         from[E[i]->v]=E[i]->u;
51         ans+=E[i]->w;
52         if(find(E[i]->u,st)==find(i,st)){
53             if(pq[i])pq[i]->tag-=E[i]->w;
54             pq[++N]=pq[i],id[N]=N;
55             for(int u=find(E[i]->u,id);u!=i;u=find(E[u]
56 ]->u,id)){
57                 if(pq[u])pq[u]->tag-=E[u]->w;
58                 id[find(u,id)]=N;
59                 pq[N]=merge(pq[N],pq[u]);
60             }
61             st[N]=find(i,st);
62             id[find(i,id)]=N;
63             }else st[find(i,st)]=find(E[i]->u,st),--all;
64         }
65     }return all==1?ans:-1;//圖不連通就無解
66 }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[
16     MAXN*2+1];
17 int flower_from[MAXN*2+1][MAXN+1],S[MAXN*2+1],vis[
18     MAXN*2+1];
19 vector<int> flower[MAXN*2+1];
20 queue<int> q;
21 inline int e_delta(const edge &e){ // does not work
22     inside blossoms
23     return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
24 }
25 inline void update_slack(int u,int x){
26     if(!slack[x]||e_delta(g[u][x])<e_delta(g[slack[x]
27 ][x]))slack[x]=u;
28 }
29 inline void set_slack(int x){
30     slack[x]=0;
31     for(int u=1;u<=n;++u)
32         if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)
33             update_slack(u,x);
34 }
35 void q_push(int x){
36     if(x<=n)q.push(x);
37     else for(size_t i=0;i<flower[x].size();i++)q.push(
38         flower[x][i]);
39 }
40 inline void set_st(int x,int b){
41     st[x]=b;
42     if(x>n)for(size_t i=0;i<flower[x].size();++i)
43         set_st(flower[x][i],b);
44 }
45 inline int get_pr(int b,int xr){
46     int pr=find(flower[b].begin(),flower[b].end(),xr)-
47         flower[b].begin();
48     if(pr%2==1){//檢查他在前一層圖是奇點還是偶點
49         reverse(flower[b].begin()+1,flower[b].end());
50         return (int)flower[b].size()-pr;
51     }else return pr;
52 }
53 inline void set_match(int u,int v){
54     match[u]=g[u][v].v;
55     if(u>n){
56         edge e=g[u][v];
57         int xr=flower_from[u][e.u],pr=get_pr(u,xr);
58         for(int i=0;i<pr;++i)set_match(flower[u][i],
59             flower[u][i+1]);
60         set_match(xr,v);
61         rotate(flower[u].begin(),flower[u].begin()+pr,
62             flower[u].end());
63     }
64 }
65 inline void augment(int u,int v){
66     for(;;){
67         int xnv=st[match[u]];
68         set_match(u,v);

```

```

60     if(!xnv)return;
61     set_match(xnv,st[pa[xnv]]);
62     u=st[pa[xnv]],v=xnv;
63 }
64 }
65 inline int get_lca(int u,int v){
66     static int t=0;
67     for(++t;u!=v;swap(u,v)){
68         if(u==0)continue;
69         if(vis[u]==t)return u;
70         vis[u]=t;//這種方法可以不用清空v陣列
71         u=st[match[u]];
72         if(u)u=st[pa[u]];
73     }
74     return 0;
75 }
76 inline void add_blossom(int u,int lca,int v){
77     int b=n+1;
78     while(b<=n_x&&st[b])+b;
79     if(b>n_x)++n_x;
80     lab[b]=0,S[b]=0;
81     match[b]=match[lca];
82     flower[b].clear();
83     flower[b].push_back(lca);
84     for(int x=u,y;x!=lca;x=st[pa[y]])
85         flower[b].push_back(x),flower[b].push_back(y=st[
86             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[
90             match[x]]),q_push(y);
91     set_st(b,b);
92     for(int x=1;x<=n_x;++x)g[b][x].w=g[x][b].w=0;
93     for(int x=1;x<=n_x;++x)flower_from[b][x]=0;
94     for(size_t i=0;i<flower[b].size();++i){
95         int xs=flower[b][i];
96         for(int x=1;x<=n_x;++x)
97             if(g[b][x].w==0||e_delta(g[xs][x])<e_delta(g[b][x]))
98                 g[b][x]=g[xs][x],g[x][b]=g[x][xs];
99         for(int x=1;x<=n_x;++x)
100             if(flower_from[xs][x])flower_from[b][x]=xs;
101     }
102     set_slack(b);
103 }
104 inline void expand_blossom(int b){ // S[b] == 1
105     for(size_t i=0;i<flower[b].size();++i)
106         set_st(flower[b][i],flower[b][i]);
107     int xr=flower_from[b][g[b][pa[b]].u],pr=get_pr(b,xr);
108     for(int i=0;i<pr;i+=2){
109         int xs=flower[b][i],xns=flower[b][i+1];
110         pa[xs]=g[xns][xs].u;
111         S[xs]=1,S[xns]=0;
112         slack[xs]=0,set_slack(xns);
113         q_push(xns);
114     }
115     S[xr]=1,pa[xr]=pa[b];
116     for(size_t i=pr+1;i<flower[b].size();++i){
117         int xs=flower[b][i];
118         S[xs]=-1,set_slack(xs);
119     }
120     st[b]=0;
121 }
122 inline bool on_found_edge(const edge &e){
123     int u=st[e.u],v=st[e.v];
124     if(S[v]==-1){
125         pa[v]=e.u,S[v]=1;
126         int nu=st[match[v]];
127         slack[v]=slack[nu]=0;
128         S[nu]=0,q_push(nu);
129     }else if(S[v]==0){
130         int lca=get_lca(u,v);
131         if(!lca)return augment(u,v),augment(v,u),true;
132         else add_blossom(u,lca,v);
133     }
134     return false;
135 }
136 inline bool matching(){
137     memset(S+1,-1,sizeof(int)*n_x);
138     memset(slack+1,0,sizeof(int)*n_x);
139     q=queue<int>();

```

```

138     for(int x=1;x<=n_x;++x)
139         if(st[x]==x&&!match[x])pa[x]=0,S[x]=0,q_push(x);
140     if(q.empty())return false;
141     for(;;){
142         while(q.size()){
143             int u=q.front();q.pop();
144             if(S[st[u]]==1)continue;
145             for(int v=1;v<=n_x;++v)
146                 if(g[u][v].w>0&&st[u]!=st[v]){
147                     if(e_delta(g[u][v])==0){
148                         if(on_found_edge(g[u][v]))return true;
149                     }else update_slack(u,st[v]);
150                 }
151         }
152         int d=INF;
153         for(int b=n+1;b<=n_x;++b)
154             if(st[b]==b&&S[b]==1)d=min(d,lab[b]/2);
155         for(int x=1;x<=n_x;++x)
156             if(st[x]==x&&slack[x]){
157                 if(S[x]==-1)d=min(d,e_delta(g[slack[x]][x]));
158                 else if(S[x]==0)d=min(d,e_delta(g[slack[x]][
159                     x])/2);
160             }
161         for(int u=1;u<=n_x;++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         for(int b=n+1;b<=n_x;++b)
178             if(st[b]==b&&S[b]==1&&lab[b]==0)expand_blossom(b);
179     }
180     return false;
181 }
182 inline pair<long long,int> weight_blossom(){
183     memset(match+1,0,sizeof(int)*n);
184     n_x=n;
185     int n_matches=0;
186     long long tot_weight=0;
187     for(int u=0;u<=n;++u)st[u]=u,flower[u].clear();
188     int w_max=0;
189     for(int u=1;u<=n;++u)
190         for(int v=1;v<=n;++v){
191             flower_from[u][v]=(u==v?u:0);
192             w_max=max(w_max,g[u][v].w);
193         }
194     for(int u=1;u<=n;++u)lab[u]=w_max;
195     while(matching())++n_matches;
196     for(int u=1;u<=n;++u)
197         if(match[u]&&match[u]<u)
198             tot_weight+=g[u][match[u]].w;
199     return make_pair(tot_weight,n_matches);
200 }
201 inline void init_weight_graph(){
202     for(int u=1;u<=n;++u)
203         for(int v=1;v<=n;++v)
204             g[u][v]=edge(u,v,0);
205 }
206 int main(){
207     int m;
208     scanf("%d",&n,&m);
209     init_weight_graph();
210     for(int i=0;i<m;++i){
211         int u,v,w;
212         scanf("%d%d%d",&u,&v,&w);
213         g[u][v].w=g[v][u].w=w;
214     }
215     printf("%lld\n",weight_blossom().first);
216     for(int u=1;u<=n;++u)printf("%d ",match[u]);puts("

```

```

    ");
215 return 0;
216 }

```

## 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>
    >> QQQ;
13 QQQ pri[205];
14 void check(int d) {
15     PII t = pri[d].top();
16     int v;
17     if (pri[d].size() - samescore[d][t.first] + 1 <=
        quota[d]) return;
18     while (pri[d].top().first == t.first) {
19         v = pri[d].top().second;
20         ans[v] = -1;
21         --samescore[d][t.first];
22         pri[d].pop();
23     }
24 }
25 void push(int s, int d) {
26     if (pri[d].size() < quota[d]) {
27         pri[d].push(PII(scoretoDep[s][d], s));
28         ans[s] = d;
29         ++samescore[s][scoretoDep[s][d]];
30     } else if (scoretoDep[s][d] >= pri[d].top().first) {
31         pri[d].push(PII(scoretoDep[s][d], s));
32         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
        format or judge may troll us.
        sadmit = stof = dexceed = dfew = 0;
        memset(iter, 0, sizeof(iter));
        memset(ans, 0, sizeof(ans));
        Fi (q, 205) {
            pri[q] = QQQ();
            samescore[q].clear();
        }
60     }
61     cin >> S >> P;
62     Fi (q, D) {
63         cin >> quota[q];
64         Fi (w, 5) cin >> weight[q][w];
65     }
66     Fi (q, S) {
67         Fi (w, 5) cin >> score[w];
68         Fi (w, D) {
69             scoretoDep[q][w] = 0;
70             F (5) scoretoDep[q][w] += weight[w][i] *
                score[i];

```

```

71     }
72 }
73 Fi (q, S) Fi (w, P) {
74     cin >> prefer[q][w];
75     --prefer[q][w];
76 }
77 f();
78 Fi (q, D) sadmit += pri[q].size();
79 Fi (q, S) if (ans[q] == prefer[q][0]) ++stof;
80 Fi (q, D) if (pri[q].size() > quota[q]) ++
    dexceed;
81 Fi (q, D) if (pri[q].size() < quota[q]) ++dfew;
82 cout << sadmit << ' ' << stof << ' ' << dexceed
    << ' ' << dfew << '\n';
83 }
84 }

```

## 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                }
33            } else {
34                low[u] = min(low[u], dfn[v]);
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 ; d++) {
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]++;
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++){
11                 c[i][j]=c[i-1][j].
12                 add(c[i-1][j-1]);
13             }
14             Scanner scanner = new Scanner(System.in);
15             int T = scanner.nextInt();
16             BigInteger x;
17             BigInteger ans;
18             while(T-- > 0){
19                 ans = BigInteger.ZERO;
20                 int n = scanner.nextInt();
21                 for(int i=0; i<n; i++){
22                     x = new BigInteger(scanner.next());
23                     if(i%2 == 1) ans=ans.subtract(c[n-1][i].multiply(x));
24                     else ans=ans.add(c[n-1][i].multiply(x));
25                 }
26                 if(n%2 == 0) ans=ans.subtract(c[n-1][0].multiply(x));
27                 System.out.println(ans);
28             }
29         }
30     }
31 }

```

### 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                 if (x.isProbablePrime(5) == true) {
15                     System.out.println(x);
16                 }
17             }
18         }
19     }
20 }

```



## 7 Other

### 7.1 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 }

```

### 7.2 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];

```

```

6     int ansd,ans[maxn];
7
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[r]] < 0)
30            H[R[r]] = L[len] = R[len] = len;
31        else{
32            R[len] = R[H[R[r]]];
33            L[R[H[R[r]]]] = len;
34            L[len] = H[R[r]];
35            R[H[R[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 }

```

### 7.3 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=j){
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 }

```

```

72     return r;
73 }
74 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%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 }

```

### 7.4 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);

```

```

57     E[u].push_back({v,x});
58     E[v].push_back({u,x});
59 }
60 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,qr=qry[i].r;
77     while (curL > ql) poke(--curL);
78     while (curR < qr) poke(++curR);
79     while (curL < ql) poke(curL++);
80     while (curR > qr) poke(curR--);
81     ans[qry[i].qid] = bit.qry();
82 }
83 for (int i=1; i<=Q; i++) cout << ans[i] << "\n";
84 }

```

## 7.5 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][j]/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 }
21 }

```

## 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 }

```

```

23 n->num=t++;
24 }
25 void ac(){
26     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             {
58                 ans[p->num]++;
59             }
60             p=p->fail;
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]) {

```

```

30     p->ch[c] = np;
31     np->deg++;
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%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 ;
10 #define SZ(x) ((int)((x).size()))
11 #define ALL(x) begin(x),end(x)
12 #define REP(i,x) for (int i=0; i<(x); i++)
13 #define REP1(i,a,b) for (int i=(a); i<=(b); i++)
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(){

```

```

55     return state.size()-2;
56 }
57 }pt;
58
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++;
60             heigh[rank[i]] = lcp;
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 }

```



```

25     return true;
26 }

```

## 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                     d[i][s] *= d[r][s];
42                 }
43             r = -1; s = -1;
44             for (int j = 0; j < m; ++j)
45                 if (s < 0 || ix[s] > ix[j]) {
46                     if (d[n + 1][j] > eps || (d[n + 1][j] > -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                 if (r < 0 || (dd = d[r][m] / d[r][s] - d[i][m] / d[i][s]) < -eps || (dd < eps && ix[r + m] > ix[i + m])) r = i;
51             }
52             if (r < 0) return -1; // not bounded
53         }
54         if (d[n + 1][m] < -eps) return -1; // not executable
55         double ans = 0;
56         for (int i = 0; i < m; ++i) x[i] = 0;
57         for (int i = m; i < n + m; ++i) { // the missing enumerated x[i] = 0
58             if (ix[i] < m - 1) {
59                 ans += d[i - m][m] * c[ix[i]];
60                 x[ix[i]] = d[i - m][m];
61             }
62         }
63     }
64     return ans;
65 }

```

## 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 * 1000000000000037
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 FWT

```

1
2 const int mod=1e9+7, rev=(mod+1)>>1;
3 void FWT(int *a, int n, int inv)
4 {
5     for(int d=1; d<n; d<=1)
6         for(int m=d<1, i=0; i<n; i+=m)
7             for(int j=0; j<d; j++)
8                 {
9                     int x=a[i+j], y=a[i+j+d];
10                    if(inv)
11                        a[i+j]=1LL*(x+y)*rev%mod, a[i+j+d]=(1LL*(x-
12                        y)*rev%mod+mod)%mod;
13                    else
14                        a[i+j]=(x+y)%mod, a[i+j+d]=(x-y+mod)%mod;
15                }
16 }W

```

## 9.7 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 }

```

## 9.8 Pollard's Rho

```

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 無權邊的生成樹個數 Kirchhoff's Theorem

1. 定義  $n \times m$  矩陣  $E = (a_{i,j})$ ,  $n$  為點數,  $m$  為邊數, 若  $i$  點在  $j$  邊上,  $i$  為小點  $a_{i,j} = 1$ ,  $i$  為大點  $a_{i,j} = -1$ , 否則  $a_{i,j} = 0$ . (證明省略)
  4. 令  $E(E^T) = Q$ , 他是一種有負號的 kirchhoff 的矩陣, 取  $Q$  的子矩陣即為  $F(F^T)$
- 結論：做  $Q$  取子矩陣算  $\det$  即為所求。(除去第一行第一列 by mz)

## 11 monge

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

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

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

## 12 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  $1 : 1 : 1$

## 13 Runge-Kutta

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

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

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

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

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

## 14 Householder Matrix

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