

# 1 DataStructure

## 1.1 1d\_segTree

```

1 void buildst(int l, int r, int idx) //l, r是st的區間
2 {
3     if(l == r){
4         st[idx] = arr[l];
5         return;
6     }
7     int mid = (l+r)/2;
8     buildst(l, mid, idx*2);
9     buildst(mid+1, r, idx*2+1);
10    st[idx] = max(st[idx*2], st[idx*2+1]);
11 }
12
13 ll query(int l, int r, int idx, int L, int R) //L,R是操作的
    區間
14 {
15     if(r < L || R < l) return -INF;
16     if(L <= l && r <= R) return st[idx];
17     int mid = (l+r)/2;
18     return max(query(l, mid, idx*2, L, R), query(mid+1, r,
19         idx*2+1, L, R));
20 }
21
22 void modify(int l, int r, int idx, int x, int v)
23 {
24     if(r < x || x < l) return;
25     if(l == r){
26         st[idx] += v; return;
27     }
28     int mid = (l+r)/2;
29     modify(l, mid, idx*2, x, v);
30     modify(mid+1, r, idx*2+1, x, v);
31     st[idx] = max(st[idx*2], st[idx*2+1]);
32 }

```

## 1.2 2d\_st\_tag

```

1 //二維陣列單點查詢區間加值
2 class St1d
3 {
4 private:
5     ll st[4*N];
6
7 public:
8     void build();
9     void modify(int l, int r, int idx, int L, int R, ll v);
10    ll query(int l, int r, int idx, int x);
11    void down(int idx);
12 };
13
14 void St1d::build(){
15     memset(st, 0, sizeof(st));
16 }
17
18 void St1d::modify(int l, int r, int idx, int L, int R, ll v){
19     if(r < L || R < l) return;
20     if(L <= l && r <= R)

```

```

21     {
22         st[idx] += v;
23         return;
24     }
25     assert(l != r);
26     down(idx);
27     int mid = (l+r)/2;
28     modify(l, mid, idx*2, L, R, v);
29     modify(mid+1, r, idx*2+1, L, R, v);
30 }
31
32 ll St1d::query(int l, int r, int idx, int x){
33     if(x < l || r < x) return 0;
34     if(l == x && r == x) return st[idx];
35     down(idx);
36     int mid = (l+r)/2;
37     ll left = query(l, mid, idx*2, x);
38     ll right = query(mid+1, r, idx*2+1, x);
39     return left+right;
40 }
41
42 void St1d::down(int idx){
43     st[idx*2] += st[idx], st[idx*2+1] += st[idx];
44     st[idx] = 0;
45 }
46
47 ////////////////
48
49 class St2d
50 {
51 private:
52     St1d st[4*N];
53
54 public:
55     void build(int il, int ir, int idx);
56     void modify(int il, int ir, int jl, int jr, int idx, int
57         iL, int iR, int jL, int jR, ll v);
58     ll query(int il, int ir, int jl, int jr, int idx, int i,
59         int j);
60 };
61
62 void St2d::build(int il, int ir, int idx){
63     st[idx].build();
64     if(il == ir) return;
65     int mid = (il+ir)/2;
66     build(il, mid, idx*2);
67     build(mid+1, ir, idx*2+1);
68 }
69
70 void St2d::modify(int il, int ir, int jl, int jr, int idx,
71     int iL, int iR, int jL, int jR, ll v){
72     if(ir < iL || iR < iL) return;
73     if(iL <= il && ir <= iR){
74         st[idx].modify(jl, jr, 1, jL, jR, v); return;
75     }
76     int mid = (il+ir)/2;
77     modify(il, mid, jl, jr, idx*2, iL, iR, jL, jR, v);
78     modify(mid+1, ir, jl, jr, idx*2+1, iL, iR, jL, jR, v);
79 }
80
81 ll St2d::query(int il, int ir, int jl, int jr, int idx, int i
82     , int j){
83     ll tot = 0;
84     if(i < iL || iR < i) return 0;
85     if(iL <= i && i <= iR) tot += st[idx].query(jl, jr, 1, j)
86         ;
87 }

```

```

82     if(il == i && ir == i) return tot;
83     int mid = (il+ir)/2;
84     tot += query(il, mid, jl, jr, idx*2, i, j);
85     tot += query(mid+1, ir, jl, jr, idx*2+1, i, j);
86     return tot;
87 }

```

## 1.3 undo\_disjoint\_set

```

1 struct DisjointSet {
2     // save() is like recursive
3     // undo() is like return
4     int n, fa[MXN], sz[MXN];
5     vector<pair<int*,int>> h;
6     vector<int> sp;
7     void init(int tn) {
8         n=tn;
9         for (int i=0; i<n; i++) sz[fa[i]=i]=1;
10        sp.clear(); h.clear();
11    }
12    void assign(int *k, int v) {
13        h.PB({k, *k});
14        *k=v;
15    }
16    void save() { sp.PB(SZ(h)); }
17    void undo() {
18        assert(!sp.empty());
19        int last=sp.back(); sp.pop_back();
20        while (SZ(h)!=last) {
21            auto x=h.back(); h.pop_back();
22            *x.F=x.S;
23        }
24    }
25    int f(int x) {
26        while (fa[x]!=x) x=fa[x];
27        return x;
28    }
29    void uni(int x, int y) {
30        x=f(x); y=f(y);
31        if (x==y) return;
32        if (sz[x]<sz[y]) swap(x, y);
33        assign(&sz[x], sz[x]+sz[y]);
34        assign(&fa[y], x);
35    }
36 }djs;

```

## 1.4 treap

```

1 struct Treap {
2     int pri, sz;
3     int rev;
4     ll data, tag; // tag: make-same
5     Treap *l, *r;
6     Treap(ll d):pri(rand()), sz(1), rev(0), data(d), tag(INF)
7         , l(NULL), r(NULL) {}
8     inline void up();
9     inline void down();
10 };
11
12 int size(Treap *t) { return t? t->sz:0; }

```

```

12 ll get_data(Treap *t) { return t? t->data:0; }
13
14 void Treap::up() {
15     if(l) l->down();
16     if(r) r->down();
17     sz = 1+size(l)+size(r);
18 }
19 void Treap::down() {
20     if(tag != INF) {
21         data = tag;
22         if(l) l->tag = tag;
23         if(r) r->tag = tag;
24         tag = INF;
25     }
26     if(rev) {
27         swap(l, r);
28         if(l) l->rev ^= 1;
29         if(r) r->rev ^= 1;
30         rev ^= 1;
31     }
32 }
33 void freeTreap(Treap *t) {
34     if(!t) return;
35     if(t->l) freeTreap(t->l);
36     if(t->r) freeTreap(t->r);
37     delete t;
38 }
39 Treap *merge(Treap *a, Treap *b) {
40     if(!a || !b) return (a? a:b);
41     if(a->pri < b->pri) {
42         a->down();
43         a->r = merge(a->r, b);
44         a->up();
45         return a;
46     } else {
47         b->down();
48         b->l = merge(a, b->l);
49         b->up();
50         return b;
51     }
52 }
53 void split(Treap *o, Treap *&a, Treap *&b, int k) {
54     if(!o) a = b = NULL;
55     else {
56         o->down();
57         if(k >= size(o->l)+1) {
58             a = o;
59             split(o->r, a->r, b, k-size(o->l)-1);
60         } else {
61             b = o;
62             split(o->l, a, b->l, k);
63         }
64         o->up();
65     }
66 }
67
68 Treap* buildTreap(vector<int> &arr) {
69     srand(7122+time(NULL));
70     Treap *tp = NULL;
71     for(auto x : arr)
72         tp = merge(tp, new Treap(x));
73     return tp;
74 }
75 void ins(Treap *&tp, int pos, int x) {
76     Treap *a, *b;
77     split(tp, a, b, pos);

```

```

78     tp = merge(a, merge(new Treap(x), b));
79 }
80 void del(Treap *&tp, int pos, int k) {
81     Treap *a, *b, *c;
82     split(tp, a, b, pos-1);
83     split(b, b, c, k);
84     freeTreap(b);
85     tp = merge(a, c);
86 }
87 void makeSame(Treap *tp, int pos, int k, int val) {
88     Treap *a, *b, *c;
89     split(tp, a, b, pos-1);
90     split(b, b, c, k);
91     b->tag = val;
92     tp = merge(a, merge(b, c));
93 }
94 void rev(Treap *&tp, int pos, int k) {
95     Treap *a, *b, *c;
96     split(tp, a, b, pos-1);
97     split(b, b, c, k);
98     b->rev ^= 1;
99     tp = merge(a, merge(b, c));
100 }

```

## 1.5 disjoint\_set

```

1 // path compression
2 int f[N];
3
4 int findrt(int x)
5 {
6     if(f[x] == x) return x;
7     else return f[x] = findrt(f[x]);
8 }
9
10 int same(int x, int y)
11 {
12     return findrt(x) == findrt(y);
13 }
14
15 void uni(int x, int y)
16 {
17     f[findrt(y)] = findrt(x);
18 }
19
20 void init()
21 {
22     for(int i = 0; i < N; i++) f[i] = i;
23 }
24
25 //union by rank
26 int f[N]; //disjoint set
27 int rk[N]; //union by rank
28
29 int findrt(int x)
30 {
31     if(f[x] == x) return x;
32     else return f[x] = findrt(f[x]);
33 }
34
35 bool same(int x, int y)
36 {
37     return findrt(x) == findrt(y);

```

```

38 }
39
40 void uni(int x, int y)
41 {
42     x = findrt(x), y = findrt(y);
43     if(x == y) return;
44     if(rk[x] < rk[y]) f[x] = y;
45     else if(rk[x] == rk[y]) f[x] = y, rk[y]++;
46     else f[y] = x;
47 }
48
49 void init()
50 {
51     for(int i = 0; i < N; i++) f[i] = i, rk[i] = 0;
52 }

```

## 1.6 Matrix

```

1 ll SZ,MOD;
2 const int MAXSZ=105;
3
4 struct Mat{
5     ll m[MAXSZ][MAXSZ];
6     Mat(){memset(m, 0, sizeof(m));}
7 };
8
9 Mat matMul(const Mat &A, const Mat &B){
10     Mat rtn;
11     for(int i = 0; i < SZ; i++)
12         for(int k = 0; k < SZ; k++){
13             if(A.m[i][k])for(int j=0; j<SZ; j++){
14                 rtn.m[i][j]+=(A.m[i][k]*B.m[k][j]);
15             }
16         }
17     return rtn;
18 }
19 //B is of size SZ
20 vector<ll> matMul(const Mat &A, const vector<ll> &B)
21 {
22     vector<ll> rtn(SZ,0);
23     for(int i = 0; i < SZ; i++)
24         for(int j = 0; j < SZ; j++)
25             rtn[i]=(rtn[i]+A.m[i][j]*B[j]);
26     return rtn;
27 }
28
29 Mat matPow(Mat& M, ll p){
30     if(p == 0){
31         Mat iden;
32         for(int i=0;i<SZ;i++)iden.m[i][i]=1;
33         return iden;
34     }
35     if(p == 1)return M;
36     Mat rtn = matPow(M, p/2);
37     if(p&1)return matMul(matMul(rtn, rtn), M);
38     else return matMul(rtn, rtn);
39 }

```

## 1.7 1d\_segTree\_tag

```

1 //線段樹懶人標記：一維陣列區間加值區間乘值區間查詢總和
2 struct Node //data = data*mul+add;
3 {
4     ll data, mul, add;
5 };
6
7 ll getval(int l, int r, int idx){
8     return (st[idx].data*st[idx].mul%MD+(r-l+1)*st[idx].add%
9         MD)%MD;
10 }
11 void up(int l, int r, int idx){
12     int mid = l+(r-l)/2;
13     st[idx].data = (getval(l, mid, idx*2)+getval(mid+1, r,
14         idx*2+1))%MD;
15 }
16 void down(int l, int r, int idx){
17     st[idx].data = getval(l, r, idx);
18     int lson = idx*2, rson = idx*2+1;
19     if(l != r){
20         st[lson].mul = st[lson].mul*st[idx].mul%MD;
21         st[lson].add = (st[lson].add*st[idx].mul+st[idx].add)
22             %MD;
23         st[rson].mul = st[rson].mul*st[idx].mul%MD;
24         st[rson].add = (st[rson].add*st[idx].mul+st[idx].add)
25             %MD;
26     }
27     st[idx].mul = 1, st[idx].add = 0;
28 }
29 void buildst(int l, int r, int idx){
30     st[idx].mul = 1, st[idx].add = 0;
31     if(l == r){
32         st[idx].data = arr[l];
33         return;
34     }
35     int mid = l+(r-l)/2;
36     buildst(l, mid, idx*2);
37     buildst(mid+1, r, idx*2+1);
38     up(l, r, idx);
39 }
40 void add(int l, int r, int idx, int L, int R, int v){ //操作L
41     ,R
42     if(r < L || R < l) return;
43     if(L <= l && r <= R){
44         st[idx].add = (st[idx].add+v)%MD;
45         return;
46     }
47     down(l, r, idx);
48     int mid = l+(r-l)/2;
49     add(l, mid, idx*2, L, R, v);
50     add(mid+1, r, idx*2+1, L, R, v);
51     up(l, r, idx);
52 }
53 void mul(int l, int r, int idx, int L, int R, int v){
54     if(r < L || R < l) return;
55     if(L <= l && r <= R){
56         st[idx].add = st[idx].add*v%MD;
57         st[idx].mul = st[idx].mul*v%MD;
58         return;
59     }
60     down(l, r, idx);

```

```

61     int mid = l+(r-l)/2;
62     mul(l, mid, idx*2, L, R, v);
63     mul(mid+1, r, idx*2+1, L, R, v);
64     up(l, r, idx);
65 }
66
67 ll query(int l, int r, int idx, int L, int R){
68     if(r < L || R < l) return 0;
69     if(L <= l && r <= R){
70         return getval(l, r, idx);
71     }
72     down(l, r, idx);
73     int mid = l+(r-l)/2;
74     return (query(l, mid, idx*2, L, R)+query(mid+1, r, idx
75         *2+1, L, R))%MD;

```

## 1.8 BIT

```

1 #define lowbit(x) x&-x
2
3 int arr[N]; //紀錄前綴和
4 int bit[N];
5
6 void conv(int a[], int n) //離散化
7 {
8     vector<int> tmp;
9     for(int i=1; i<=n; i++) tmp.push_back(a[i]);
10    sort(tmp.begin(), tmp.end());
11    for(int i=1; i<=n; i++) a[i] = lower_bound(tmp.begin(),
12        tmp.end(), a[i]) - tmp.begin() + 1;
13 }
14 void buildbit() //每個bit[x]紀錄[x-lowbit(x)+1, x]的總和
15 {
16     for(int i = 0; i < n; i++) bit[i] = arr[i]-arr[i-lowbit(i
17         )];
18 }
19 int sum(int x) //查詢[1,x]的總和
20 {
21     int rtn = 0;
22     for(;x>=lowbit(x);) rtn += bit[x];
23     return rtn;
24 }
25
26 void modify(int x, int d) //把位置x的東西加上d
27 {
28     for(;x<=n;x+=lowbit(x)) bit[x] += d;
29 }

```

## 2 Flow

### 2.1 dinic

```

1 template<typename T>
2 struct DINIC{

```

```

3     static const int MAXN=105;
4     static const T INF=INT_MAX;
5     int n, level[MAXN], cur[MAXN];
6     struct edge{
7         int v,pre;
8         T cap,flow,r;
9         edge(int v,int pre,T cap):v(v),pre(pre),cap(cap),flow(0),
10             r(cap){}
11     };
12     int g[MAXN];
13     vector<edge> e;
14     void init(int _n){
15         memset(g,-1,sizeof(int)*((n=_n)+1));
16         e.clear();
17     }
18     void add_edge(int u,int v,T cap,bool directed=false){
19         e.push_back(edge(v,g[u],cap));
20         g[u]=e.size()-1;
21         e.push_back(edge(u,g[v],directed?0:cap));
22         g[v]=e.size()-1;
23     }
24     int bfs(int s,int t){
25         memset(level,0,sizeof(int)*(n+1));
26         memcpy(cur,g,sizeof(int)*(n+1));
27         queue<int> q;
28         q.push(s);
29         level[s]=1;
30         while(q.size()){
31             int u=q.front();q.pop();
32             for(int i=g[u];~i;i=e[i].pre){
33                 if(!level[e[i].v]&&e[i].r){
34                     level[e[i].v]=level[u]+1;
35                     q.push(e[i].v);
36                     if(e[i].v==t)return 1;
37                 }
38             }
39         }
40         return 0;
41     }
42     T dfs(int u,int t,T cur_flow=INF){
43         if(u==t)return cur_flow;
44         T df;
45         for(int &i=cur[u];~i;i=e[i].pre){
46             if(level[e[i].v]==level[u]+1&&e[i].r){
47                 if(df=dfs(e[i].v,t,min(cur_flow,e[i].r)))
48                     e[i].flow+=df;
49                     e[i^1].flow-=df;
50                     e[i].r-=df;
51                     e[i^1].r+=df;
52                     return df;
53             }
54         }
55         return level[u]=0;
56     }
57     T dinic(int s,int t,bool clean=true){
58         if(clean){
59             for(size_t i=0;i<e.size();++i){
60                 e[i].flow=0;
61                 e[i].r=e[i].cap;
62             }
63         }
64         T ans=0, mf=0;
65         while(bfs(s,t))while(mf=dfs(s,t))ans+=mf;
66         return ans;
67     }

```

68|};

## 2.2 dinic-Benq

```

1  /**
2   * Description: fast flow
3   * Time:  $O(N^2M)$  $ flow,  $O(M\sqrt{N})$  $ bipartite matching
4   * Source: GeeksForGeeks, Chilli
5   * Verification: RMI 2017 Day 1 Fashion
6   * https://pastebin.com/VJxTvEg1
7   */
8
9  template<int SZ> struct Dinic {
10     typedef ll F; // flow type
11     struct Edge { int to, rev; F flow, cap; };
12     int N,s,t;
13     vector<Edge> adj[SZ];
14     typename vector<Edge>::iterator cur[SZ];
15     void addEdge(int u, int v, F cap) {
16         assert(cap >= 0); // don't try smth dumb
17         Edge a{v, sz(adj[v]), 0, cap}, b{u, sz(adj[u]), 0, 0};
18         adj[u].pb(a), adj[v].pb(b);
19     }
20     int level[SZ];
21     bool bfs() { // level = shortest distance from source
22         // after computing flow, edges {u,v} such that level[u] !=
23         // level[v] are part of min cut
24         FOR(i,N) level[i] = -1, cur[i] = begin(adj[i]);
25         queue<int> q({s}); level[s] = 0;
26         while (sz(q)) {
27             int u = q.front(); q.pop();
28             trav(e,adj[u]) if (level[e.to] < 0 && e.flow < e.cap)
29                 q.push(e.to), level[e.to] = level[u]+1;
30         }
31         return level[t] >= 0;
32     }
33     F sendFlow(int v, F flow) {
34         if (v == t) return flow;
35         for (; cur[v] != end(adj[v]); cur[v]++) {
36             Edge& e = *cur[v];
37             if (level[e.to] != level[v]+1 || e.flow == e.cap)
38                 continue;
39             auto df = sendFlow(e.to,min(flow,e.cap-e.flow));
40             if (df) { // saturated at least one edge
41                 e.flow += df; adj[e.to][e.rev].flow -= df;
42                 return df;
43             }
44         }
45         return 0;
46     }
47     F maxFlow(int _N, int _s, int _t) {
48         N = _N, s = _s, t = _t; if (s == t) return -1;
49         F tot = 0;
50         while (bfs()) while (auto df = sendFlow(s,numeric_limits<
51             F>::max()) tot += df;
52         return tot;
53     }
54 };

```

## 2.3 MaxDensitySubgraph

```

1  #include<stdio.h>
2  #include<string.h>
3  const int N=1500;
4  const double inf=0x3fffffff;
5  const double eps=1e-8;
6  int gap[N],dis[N],start,end,ans,sum,head[N],num,dep[N],n,m;
7  bool vis[N];
8  struct edge
9  {
10     int st,ed,next;
11     double flow;
12 }e[80*N];
13 struct node
14 {
15     int x,y;
16 }P[1100];
17 void addedge(int x,int y,double w)
18 {
19     e[num].st=x;e[num].ed=y;e[num].flow=w;e[num].next=head[x];
20     head[x]=num++;
21     e[num].st=y;e[num].ed=x;e[num].flow=0;e[num].next=head[y];
22     head[y]=num++;
23 }
24 void makemap(double g)
25 {
26     int i;
27     memset(head,-1,sizeof(head));
28     num=0;
29     for(i=1;i<=n;i++)
30         addedge(i,end,g);
31     for(i=0;i<m;i++)
32     {
33         addedge(n+i+1,P[i].y,inf);
34         addedge(n+i+1,P[i].x,inf);
35         addedge(start,n+i+1,1.0);
36     }
37 }
38 double dfs(int u,double minflow)
39 {
40     if(u==end)return minflow;
41     int i,v;
42     double f,flow=0.0;
43     for(i=head[u];i!=-1;i=e[i].next)
44     {
45         v=e[i].ed;
46         if(e[i].flow>0)
47         {
48             if(dis[v]+1==dis[u])
49             {
50                 f=dfs(v,e[i].flow>minflow-flow?minflow-flow:e
51                     [i].flow);
52                 flow+=f;
53                 e[i].flow-=f;
54                 e[i^1].flow+=f;
55                 if(minflow-flow<=1e-8)return flow;
56                 if(dis[start]>=ans)return flow;
57             }
58         }
59     }
60     if(--gap[dis[u]]==0)
61         dis[start]=ans;
62     dis[u]++;
63     gap[dis[u]]++;
64     return flow;
65 }
66 double isap()

```

```

64 {
65     double maxflow=0.0;
66     memset(gap,0,sizeof(gap));
67     memset(dis,0,sizeof(dis));
68     gap[0]=ans;
69     while(dis[start]<ans)
70         maxflow+=dfs(start,inf);
71     return 1.0*m-maxflow;
72 }
73 void dfs1(int u)
74 {
75     vis[u]=true;
76     if(u>=1&&u<=n)
77         sum++;
78     for(int i=head[u];i!=-1;i=e[i].next)
79     {
80         int v=e[i].ed;
81         if(vis[v]==false&&e[i].flow>0)
82             dfs1(v);
83     }
84 }
85 int main()
86 {
87     int i;
88     double Left,Right,mid,flow;
89     while(scanf("%d",&n,&m)!=-1)
90     {
91         if(m==0){printf("1\n");continue;}
92         start=0,end=n+m+1,ans=end+1;
93         for(i=0;i<m;i++)
94         {
95             scanf("%d",&P[i].x,&P[i].y);
96         }
97         Left=0;Right=m;
98         while(Right-Left>=1.0/n/n)//胡伯涛的论文给出了证明,不同解
99             之间误差的精度不超过1/(n*n)
100         {
101             mid=(Left+Right)/2;
102             makemap(mid);
103             flow=isap();//求出最大权值闭合图
104             if(flow<eps)//如果小于0, g值太大
105                 Right=mid;
106             else Left=mid;
107         }
108         makemap(Left);//最大密度建图
109         isap();
110         memset(vis,false,sizeof(vis));
111         sum=0;
112         dfs1(start);
113         printf("%d\n",sum);
114         for(i=1;i<=n;i++)
115             if(vis[i]==true)//残留网络中源点能到达的点
116                 printf("%d\n",i);
117     }
118     return 0;
119 }

```

## 2.4 MinCostMaxFlow

```

1  template<typename TP>
2  struct MCMF{
3      static const int MAXN=440;

```

```

4 static const TP INF=999999999;
5 struct edge{
6     int v,pre;
7     TP r,cost;
8     edge(int v,int pre,TP r,TP cost):v(v),pre(pre),r(r),cost(cost){}
9 };
10 int n,S,T;
11 TP dis[MAXN],PIS,ans;
12 bool vis[MAXN];
13 vector<edge> e;
14 int g[MAXN];
15 void init(int _n){
16     memset(g,-1,sizeof(int)*(n=_n)+1));
17     e.clear();
18 }
19 void add_edge(int u,int v,TP r,TP cost,bool directed=false)
20 {
21     e.push_back(edge(v,g[u],r,cost));
22     g[u]=e.size()-1;
23     e.push_back(
24         edge(u,g[v],directed?0:r,-cost));
25     g[v]=e.size()-1;
26 }
27 TP augment(int u,TP CF){
28     if(u==T||!CF)return ans+=PIS*CF,CF;
29     vis[u]=1;
30     TP r=CF,d;
31     for(int i=g[u];~i;i=e[i].pre){
32         if(e[i].r&&!e[i].cost&&!vis[e[i].v]){
33             d=augment(e[i].v,min(r,e[i].r));
34             e[i].r-=d;
35             e[i+1].r+=d;
36             if(!(r-=d))break;
37         }
38     }
39     return CF-r;
40 }
41 bool modlabel(){
42     for(int u=0;u<=n;++u)dis[u]=INF;
43     static deque<int>q;
44     dis[T]=0,q.push_back(T);
45     while(q.size()){
46         int u=q.front();q.pop_front();
47         TP dt;
48         for(int i=g[u];~i;i=e[i].pre){
49             if(e[i+1].r&&(dt=dis[u]-e[i].cost)<dis[e[i].v]){
50                 if((dis[e[i].v]=dt)<=dis[q.size()-1]?q.front():S){
51                     q.push_front(e[i].v);
52                 }else q.push_back(e[i].v);
53             }
54         }
55     }
56     for(int u=0;u<=n;++u)
57         for(int i=g[u];~i;i=e[i].pre)
58             e[i].cost+=dis[e[i].v]-dis[u];
59     return PIS+=dis[S], dis[S]<INF;
60 }
61 TP mincost(int s,int t){
62     S=s,T=t;
63     PIS=ans=0;
64     while(modlabel()){
65         do memset(vis,0,sizeof(bool)*(n+1));
66         while(augment(S,INF));
67     }return ans;
68 }

```

68| };

## 3 Geometry

### 3.1 point

```

1 const double eps = 5e-8;
2 struct Point{
3     double x,y;
4     Point(){}
5     Point(double x,double y):x(x),y(y){}
6     Point operator+(Point b)const{
7         return Point(x+b.x,y+b.y);
8     }
9     Point operator-(Point b)const{
10        return Point(x-b.x,y-b.y);
11    }
12    Point operator*(double b)const{
13        return Point(x*b,y*b);
14    }
15    Point operator/(double b)const{
16        return Point(x/b,y/b);
17    }
18    bool operator==(Point b)const{
19        return (fabs(x-b.x)<=eps&&fabs(y-b.y)<=eps);
20    }
21    double dot(Point b)const{
22        return x*b.x+y*b.y;
23    }
24    double cross(Point b)const{
25        return x*b.y-y*b.x;
26    }
27    Point normal()const{
28        return Point(-y,x);
29    } // 求法向量
30    double abs2()const{
31        return dot(*this);
32    } // 向量长度的平方
33    double rad(const Point b)const{
34        return fabs(atan2(fabs(cross(b)),dot(b)));
35    } // 兩向量的弧度
36 };

```

### 3.2 intercircle

```

1 vector<Point> interCir(Point o1, double r1, Point o2, double
2     r2){
3     double d=sqrt((o1-o2).abs2());
4     double c=(r1*r1 + d*d - r2*r2)/2.0/r1/d;
5     double s=sqrt(1.0-c*c);
6     Point v=(o2-o1)*r1/d;
7     // case 0 intersections
8     if(d>r1+r2||d<fabs(r1-r2)) return{};
9     // case 1 intersection
10    if(d-eps<=r1+r2&&r1+r2<=d+eps) return{o1+v};
11    if(d-eps<=fabs(r1-r2)&&fabs(r1-r2)<=d+eps) return{o1-v};
12    // case 2 intersections

```

```

12 Point v_up=(Point){v.x*c-v.y*s,v.x*s+v.y*c};
13 Point v_down=(Point){v.x*c+v.y*s,-v.x*s+v.y*c};
14 return {o1+v_up,o1+v_down};
15 } // 求兩圓交點

```

### 3.3 SegmentGeometry

```

1 double EPS = 1e-10;
2
3 double add(double a, double b){
4     if(fabs(a+b)<EPS*(fabs(a)+fabs(b)))return 0;
5     else return a+b;
6 }
7
8 struct P//struct for 2d vector/point
9 {
10     double x,y;
11     P(){}
12     P(double x, double y):x(x),y(y){}
13     P operator+(P p){return P(add(x,p.x), add(y,p.y));}
14     P operator-(P p){return P(add(x,-p.x), add(y,-p.y));}
15     P operator*(double d){return P(x*d,y*d);}
16     double dot(P p){return add( x*p.x, y*p.y );}
17     double det(P p){return add( x*p.y, -y*p.x );}
18 };
19
20 //is point q on p1p2
21 bool on_seg(P p1, P p2, P q){return (p1-q).det(p2-q)==0&&(p1-
22     q).dot(p2-q)<=0;}
23
24 P intersection(P p1, P p2, P q1, P q2)//p and q Must not be
25     parallel
26 {return p1 + (p2-p1)*((q2-q1).det(q1-p1)/(q2-q1).det(p2-p1))
27     ;}
28
29 bool par(P p1, P p2, P p3, P p4){return (p2-p1).det(p4-p3)
30     ==0;}
31
32 bool operator<(const P& lhs, const P& rhs)
33 {return (lhs.x==rhs.x)?lhs.y<rhs.y:lhs.x<rhs.x;}
34
35 bool operator==(const P& lhs, const P& rhs)
36 {return lhs.x==rhs.x&&lhs.y==rhs.y;}
37
38 double len(P vec)
39 {return sqrt(add(vec.x*vec.x, vec.y*vec.y));}
40
41 double dis(P p1, P p2)
42 {return len(p2-p1);}
43
44 struct seg
45 {
46     seg(){}
47     seg(P _p1, P _p2)
48     {
49         p[0]=_p1;
50         p[1]=_p2;
51         if(p[1]<p[0])swap(p[0],p[1]);
52     }
53     P p[2];
54 };
55
56 bool par(seg& lhs, seg& rhs)

```

```

53 {return par(lhs.p[0],lhs.p[1],rhs.p[0],rhs.p[1]);}
54
55 P intersection(seg& lhs, seg& rhs)//p and q Must not be
    parallel
56 {return intersection(lhs.p[0],lhs.p[1],rhs.p[0],rhs.p[1]);}
57
58 bool on_seg(seg& sg, P q)
59 {return on_seg(sg.p[0],sg.p[1],q);}
60
61 bool overlap(seg s1, seg s2){
62     return par(s1,s2)&&
63     ( on_seg(s1,s2.p[0])||on_seg(s1,s2.p[1])||
64     on_seg(s2,s1.p[0])||on_seg(s2,s1.p[1]) );
65 }
66
67 bool is_intersect(seg s1, seg s2){
68     if(par(s1,s2))return false;
69     P p0 = intersection(s1,s2);
70     return on_seg(s1,p0)&&on_seg(s2,p0);
71 }
72
73 //make sure the vec is not vertical
74 double interpolate(seg& vec, double X){
75     double y0=vec.p[0].y,y1=vec.p[1].y,
76     x0=vec.p[0].x,x1=vec.p[1].x;
77     return y0+(y1-y0)*(X-x0)/(x1-x0);
78 }
79
80 //pts in clockwise order, p[N]=p[0]
81 bool in_poly(P* pol,int N,P pt){
82     double X = pt.x,Y=pt.y;
83     int pas=0;
84     for(int i=0;i<N;i++){
85         if(pol[i].x==pol[i+1].x)continue;
86         seg s0(pol[i],pol[i+1]);
87         //up or down?
88         double Y1 = interpolate(s0,X);
89         if(Y1<Y-EPS)continue;
90         double xl=min(pol[i].x,pol[i+1].x),xr=max(pol[i].x,
91         pol[i+1].x);
92         if(xl<X-EPS&&xr>=X-EPS)pas++;
93     }
94     return pas&1;
95 }
96
97 double dpseg(P p, P p1, P p2)//p to p1p2, p1!=p2
98 {
99     P v=p2-p1, v1=p-p1, v2=p-p2;
100     if( v.dot(v1) < EPS )return dis(p,p1);
101     if( v.dot(v2) > EPS )return dis(p,p2);
102     return fabs((p-p1).det(v))/len(v);
103 }
104
105 double dpseg(P p, seg s1){
106     return dpseg(p,s1.p[0],s1.p[1]);
107 }
108
109 double dsegseg(P p1, P p2, P p3, P p4){
110     if( is_intersect( seg(p1,p2), seg(p3,p4) ) )return 0;
111     return min( min( dpseg(p1,p3,p4),dpseg(p2,p3,p4) ), min(
112     dpseg(p3,p1,p2),dpseg(p4,p1,p2) ) );
113 }
114
115 double dsegseg(seg s1, seg s2)
116 {
117     return dsegseg( s1.p[0],s1.p[1],s2.p[0],s2.p[1] );
118 }

```

### 3.4 convexHullTrick

```

1 // usage ( for example )
2 // dp[i] = min(dp[j] - 2*a[i]*a[j] + a[j]^2) + m + a[i]^2
3 // insert into hull :
4 // a x + b
5 // (-2*a[j]) (a[i]) + (dp[j]+a[j]^2)
6 // get dp[i] :
7 // dp[i] = hull(a[i]) + m + a[i]^2
8
9 template<typename Ty = long long int>
10 class Linear{
11 private:
12     Ty a, b;
13 public:
14     Linear(Ty a, Ty b):a(a), b(b) {}
15     Ty operator()(Ty x) { return a*x+b; }
16     // get x of intersection of two lines (fraction)
17     tuple<Ty, Ty> inter(Linear &that){
18         ll up = that.b-this->b;
19         ll down = this->a-that.a;
20         if(down < 0) up *= -1, down *= -1;
21         return make_tuple(up, down);
22     }
23 };
24
25 template<typename Ty = long long int>
26 class ConvexHull{
27 private:
28     using L = Linear<Ty>;
29     vector<L> hull;
30 public:
31     void push_back(L h){
32         while(hull.size() >= 2){
33             auto &f = hull.end()[-2];
34             auto &g = hull.end()[-1]; // back
35             // x of inter(h,f) <= x of inter(f,g)
36             Ty a, b, c, d;
37             tie(a, b) = h.inter(f);
38             tie(c, d) = f.inter(g);
39             if(a*d <= b*c) hull.pop_back();
40             else break;
41         }
42         hull.push_back(h);
43     }
44     Ty operator() (Ty x){
45         static int idx = 0;
46         if(idx >= hull.size())
47             idx = max(0, (int)hull.size()-2);
48         while(idx+1 < hull.size())
49             {
50                 if(hull[idx+1](x) <= hull[idx](x)) idx++;
51                 else break;
52             }
53         return hull[idx](x);
54     }
55 };

```

### 3.5 Geometry

```

1 const double PI=atan2(0.0,-1.0);
2 template<typename T>
3 struct point{
4     T x,y;
5     point(){}
6     point(const T&x,const T&y):x(x),y(y){}
7     point operator+(const point &b)const{
8         return point(x+b.x,y+b.y); }
9     point operator-(const point &b)const{
10        return point(x-b.x,y-b.y); }
11     point operator*(const T &b)const{
12        return point(x*b,y*b); }
13     point operator/(const T &b)const{
14        return point(x/b,y/b); }
15     bool operator==(const point &b)const{
16        return x==b.x&&y==b.y; }
17     T dot(const point &b)const{
18        return x*b.x+y*b.y; }
19     T cross(const point &b)const{
20        return x*b.y-y*b.x; }
21     point normal()const{//求法向量
22        return point(-y,x); }
23     T abs2()const{//向量长度的平方
24        return dot(*this); }
25     T rad(const point &b)const{//兩向量的弧度
26     return fabs(atan2(fabs(cross(b)),dot(b))); }
27     T getA()const{//對x軸的弧度
28        T A=atan2(y,x); //超過180度會變負的
29        if(A<=-PI/2)A+=PI*2;
30        return A;
31     }
32 };
33 template<typename T>
34 struct line{
35     line(){}
36     point<T> p1,p2;
37     T a,b,c;//ax+by+c=0
38     line(const point<T>&x,const point<T>&y):p1(x),p2(y){}
39     void pton()const{//轉成一般式
40         a=p1.y-p2.y;
41         b=p2.x-p1.x;
42         c=-a*p1.x-b*p1.y;
43     }
44     T ori(const point<T> &p)const{//點和有向直線的關係, >0左
45         //邊、=0在線上<0右邊
46         return (p2-p1).cross(p-p1);
47     }
48     T btw(const point<T> &p)const{//點投影落在線段上<=0
49         return (p1-p).dot(p2-p);
50     }
51     bool point_on_segment(const point<T>&p)const{//點是否在線段
52         //上
53         return ori(p)==0&&btw(p)<=0;
54     }
55     T dis2(const point<T> &p,bool is_segment=0)const{//點跟直線
56         //線段的距離平方
57         point<T> v=p2-p1,v1=p-p1;
58         if(is_segment){
59             point<T> v2=p-p2;
60             if(v.dot(v1)<=0)return v1.abs2();
61             if(v.dot(v2)>=0)return v2.abs2();
62         }

```



```

59 }
60 T tmp=v.cross(v1);
61 return tmp*tmp/v.abs2();
62 }
63 T seg_dis2(const line<T> &l) const{//兩線段距離平方
64 return min({dis2(l.p1,1),dis2(l.p2,1),l.dis2(p1,1),l.dis2
65 (p2,1)});
66 }
67 point<T> projection(const point<T> &p) const{//點對直線的投
68 影
69 point<T> n=(p2-p1).normal();
70 return p-n*(p-p1).dot(n)/n.abs2();
71 }
72 point<T> mirror(const point<T> &p) const{
73 //點對直線的鏡射，要先呼叫pton轉成一般式
74 point<T> R;
75 T d=a*b+b*b;
76 R.x=(b*b*p.x-a*a*p.x-2*a*b*p.y-2*a*c)/d;
77 R.y=(a*a*p.y-b*b*p.y-2*a*b*p.x-2*b*c)/d;
78 return R;
79 }
80 bool equal(const line &l) const{//直線相等
81 return ori(l.p1)==0&&ori(l.p2)==0;
82 }
83 bool parallel(const line &l) const{
84 return (p1-p2).cross(l.p1-l.p2)==0;
85 }
86 bool cross_seg(const line &l) const{
87 return (p2-p1).cross(l.p1-p1)*(p2-p1).cross(l.p2-p1)<=0;
88 //直線是否交線段
89 }
90 int line_intersect(const line &l) const{//直線相交情況，-1無
91 限多點、1交於一點、0不相交
92 return parallel(l)?(ori(l.p1)==0?-1:0):1;
93 }
94 int seg_intersect(const line &l) const{
95 T c1=ori(l.p1), c2=ori(l.p2);
96 T c3=l.ori(p1), c4=l.ori(p2);
97 if(c1==0&&c2==0){//共線
98 bool b1=btw(l.p1)>=0,b2=btw(l.p2)>=0;
99 T a3=l.btw(p1),a4=l.btw(p2);
100 if(b1&&b2&&a3==0&&a4==0) return 2;
101 if(b1&&b2&&a3>=0&&a4==0) return 3;
102 if(b1&&b2&&a3>=0&&a4>=0) return 0;
103 return -1;//無限交點
104 }else if(c1*c2<=0&&c3*c4<=0) return 1;
105 return 0;//不相交
106 }
107 point<T> line_intersection(const line &l) const{/*直線交點*/
108 point<T> a=p2-p1,b=l.p2-l.p1,s=l.p1-p1;
109 //if(a.cross(b)==0) return INF;
110 return p1+a*(s.cross(b)/a.cross(b));
111 }
112 point<T> seg_intersection(const line &l) const{//線段交點
113 int res=seg_intersect(l);
114 if(res<=0) assert(0);
115 if(res==2) return p1;
116 if(res==3) return p2;
117 return line_intersection(l);
118 }
119 };
120 template<typename T>
121 struct polygon{
122     118 polygon(){}
123     119 vector<point<T> > p;//逆時針順序
124     120 T area() const{//面積
125     121 T ans=0;
126     122 for(int i=p.size()-1,j=0;j<(int)p.size();i=j++){
127     123 ans+=p[i].cross(p[j]);
128     124 return ans/2;
129     125 }
130     126 point<T> center_of_mass() const{//重心
131     127 T cx=0,cy=0,w=0;
132     128 for(int i=p.size()-1,j=0;j<(int)p.size();i=j++){
133     129 T a=p[i].cross(p[j]);
134     130 cx+=(p[i].x+p[j].x)*a;
135     131 cy+=(p[i].y+p[j].y)*a;
136     132 w+=a;
137     133 }
138     134 return point<T>(cx/3/w,cy/3/w);
139     135 }
140     136 char ahas(const point<T> &t) const{//點是否在簡單多邊形內，
141     137 是的話回傳1、在邊上回傳-1、否則回傳0
142     138 bool c=0;
143     139 for(int i=0,j=p.size()-1;i<p.size();j=i++){
144     140 if((line<T>(p[i],p[j]).point_on_segment(t)) return -1;
145     141 else if((p[i].y>t.y)!=p[j].y>t.y)&&
146     142 t.x<(p[j].x-p[i].x)*(t.y-p[i].y)/(p[j].y-p[i].y)+p[i].x
147     143 )
148     144 c=!c;
149     145 return c;
150     146 }
151     147 char point_in_convex(const point<T> &x) const{
152     148 int l=1,r=(int)p.size()-2;
153     149 while(l<r){//點是否在凸多邊形內，是的話回傳1、在邊上回傳
154     150 -1、否則回傳0
155     151 int mid=(l+r)/2;
156     152 T a1=(p[mid]-p[0]).cross(x-p[0]);
157     153 T a2=(p[mid+1]-p[0]).cross(x-p[0]);
158     154 if(a1>=0&&a2<=0){
159     155 T res=(p[mid+1]-p[mid]).cross(x-p[mid]);
160     156 return res>0?1:(res>=0?-1:0);
161     157 }else if(a1<0)r=mid-1;
162     158 else l=mid+1;
163     159 }
164     160 return 0;
165     161 }
166     162 vector<T> getA() const{//凸包邊對x軸的夾角
167     163 vector<T> res;//一定是遞增的
168     164 for(size_t i=0;i<p.size();i++){
169     165 res.push_back((p[(i+1)%p.size()]-p[i]).getA());
170     166 return res;
171     167 }
172     168 bool line_intersect(const vector<T> &A,const line<T> &l)
173     169 const{//O(logN)
174     170 int f1=upper_bound(A.begin(),A.end(),(l.p1-l.p2).getA())-
175     171 A.begin();
176     172 int f2=upper_bound(A.begin(),A.end(),(l.p2-l.p1).getA())-
177     173 A.begin();
178     174 return l.cross_seg(line<T>(p[f1],p[f2]));
179     175 }
180     176 polygon cut(const line<T> &l) const{//凸包對直線切割，得到直
181     177 線l左側的凸包
182     178 polygon ans;
183     179 for(int i=p.size(),i=n-1,j=0;j<n;i=j++){
184     180 if(l.ori(p[i])>=0){
185     181 ans.p.push_back(p[i]);
186     182 }
187     183 if(l.ori(p[j])<0)
188     184 ans.p.push_back(l.line_intersection(line<T>(p[i],p[
189     185 j])));
190     186 }else if(l.ori(p[j])>0)
191     187 ans.p.push_back(l.line_intersection(line<T>(p[i],p[
192     188 j])));
193     189 }
194     190 return ans;
195     191 }
196     192 static bool graham_cmp(const point<T> &a,const point<T> &b)
197     193 {
198     194 //凸包排序函數
199     195 return (a.x<b.x)||a.x==b.x&&a.y<b.y;
200     196 }
201     197 void graham(vector<point<T> > &s){//凸包
202     198 sort(s.begin(),s.end(),graham_cmp);
203     199 p.resize(s.size()+1);
204     200 int m=0;
205     201 for(size_t i=0;i<s.size();i++){
206     202 while(m>=2&&(p[m-1]-p[m-2]).cross(s[i]-p[m-2])<=0)--m;
207     203 p[m++]=s[i];
208     204 }
209     205 for(int i=s.size()-2,t=m+1;i>=0;--i){
210     206 while(m>=2&&(p[m-1]-p[m-2]).cross(s[i]-p[m-2])<=0)--m;
211     207 p[m++]=s[i];
212     208 }
213     209 if(s.size()>1)--m;
214     210 p.resize(m);
215     211 }
216     212 T diam(){//直徑
217     213 int n=p.size(),t=1;
218     214 T ans=0;p.push_back(p[0]);
219     215 for(int i=0;i<n;i++){
220     216 point<T> now=p[i+1]-p[i];
221     217 while(now.cross(p[t+1]-p[i])>now.cross(p[t]-p[i]))t=(t
222     218 +1)%n;
223     219 ans=max(ans,(p[i]-p[t]).abs2());
224     220 }
225     221 return p.pop_back(),ans;
226     222 }
227     223 T min_cover_rectangle(){//最小覆蓋矩形
228     224 int n=p.size(),t=1,r=1,l;
229     225 if(n<3) return 0;//也可以做最小周長矩形
230     226 T ans=1e99;p.push_back(p[0]);
231     227 for(int i=0;i<n;i++){
232     228 point<T> now=p[i+1]-p[i];
233     229 while(now.cross(p[t+1]-p[i])>now.cross(p[t]-p[i]))t=(t
234     230 +1)%n;
235     231 while(now.dot(p[r+1]-p[i])>now.dot(p[r]-p[i]))r=(r+1)%n
236     232 ;
237     233 if(!l)r=l;
238     234 while(now.dot(p[l+1]-p[i])<=now.dot(p[l]-p[i]))l=(l+1)%
239     235 n;
240     236 T d=now.abs2();
241     237 T tmp=now.cross(p[t]-p[i])*(now.dot(p[r]-p[i])-now.dot(
242     238 p[l]-p[i]))/d;
243     239 ans=min(ans,tmp);
244     240 }
245     241 return p.pop_back(),ans;
246     242 }
247     243 T dis2(polygon &pl){//凸包最近距離平方
248     244 vector<point<T> > &P=p,&Q=pl.p;
249     245 int n=P.size(),m=Q.size(),l=0,r=0;
250     246 for(int i=0;i<n;i++){
251     247 if(P[i].y<P[l].y)l=i;
252     248 for(int i=0;i<m;i++){
253     249 if(Q[i].y<Q[r].y)r=i;
254     250 P.push_back(P[0]),Q.push_back(Q[0]);
255     251 }
256     252 }
257     253 }
258     254 }
259     255 }
260     256 }
261     257 }
262     258 }
263     259 }
264     260 }
265     261 }
266     262 }
267     263 }
268     264 }
269     265 }
270     266 }
271     267 }
272     268 }
273     269 }
274     270 }
275     271 }
276     272 }
277     273 }
278     274 }
279     275 }
280     276 }
281     277 }
282     278 }
283     279 }
284     280 }
285     281 }
286     282 }
287     283 }
288     284 }
289     285 }
290     286 }
291     287 }
292     288 }
293     289 }
294     290 }
295     291 }
296     292 }
297     293 }
298     294 }
299     295 }
300     296 }
301     297 }
302     298 }
303     299 }
304     300 }
305     301 }
306     302 }
307     303 }
308     304 }
309     305 }
310     306 }
311     307 }
312     308 }
313     309 }
314     310 }
315     311 }
316     312 }
317     313 }
318     314 }
319     315 }
320     316 }
321     317 }
322     318 }
323     319 }
324     320 }
325     321 }
326     322 }
327     323 }
328     324 }
329     325 }
330     326 }
331     327 }
332     328 }
333     329 }
334     330 }
335     331 }
336     332 }
337     333 }
338     334 }
339     335 }
340     336 }
341     337 }
342     338 }
343     339 }
344     340 }
345     341 }
346     342 }
347     343 }
348     344 }
349     345 }
350     346 }
351     347 }
352     348 }
353     349 }
354     350 }
355     351 }
356     352 }
357     353 }
358     354 }
359     355 }
360     356 }
361     357 }
362     358 }
363     359 }
364     360 }
365     361 }
366     362 }
367     363 }
368     364 }
369     365 }
370     366 }
371     367 }
372     368 }
373     369 }
374     370 }
375     371 }
376     372 }
377     373 }
378     374 }
379     375 }
380     376 }
381     377 }
382     378 }
383     379 }
384     380 }
385     381 }
386     382 }
387     383 }
388     384 }
389     385 }
390     386 }
391     387 }
392     388 }
393     389 }
394     390 }
395     391 }
396     392 }
397     393 }
398     394 }
399     395 }
400     396 }
401     397 }
402     398 }
403     399 }
404     400 }
405     401 }
406     402 }
407     403 }
408     404 }
409     405 }
410     406 }
411     407 }
412     408 }
413     409 }
414     410 }
415     411 }
416     412 }
417     413 }
418     414 }
419     415 }
420     416 }
421     417 }
422     418 }
423     419 }
424     420 }
425     421 }
426     422 }
427     423 }
428     424 }
429     425 }
430     426 }
431     427 }
432     428 }
433     429 }
434     430 }
435     431 }
436     432 }
437     433 }
438     434 }
439     435 }
440     436 }
441     437 }
442     438 }
443     439 }
444     440 }
445     441 }
446     442 }
447     443 }
448     444 }
449     445 }
450     446 }
451     447 }
452     448 }
453     449 }
454     450 }
455     451 }
456     452 }
457     453 }
458     454 }
459     455 }
460     456 }
461     457 }
462     458 }
463     459 }
464     460 }
465     461 }
466     462 }
467     463 }
468     464 }
469     465 }
470     466 }
471     467 }
472     468 }
473     469 }
474     470 }
475     471 }
476     472 }
477     473 }
478     474 }
479     475 }
480     476 }
481     477 }
482     478 }
483     479 }
484     480 }
485     481 }
486     482 }
487     483 }
488     484 }
489     485 }
490     486 }
491     487 }
492     488 }
493     489 }
494     490 }
495     491 }
496     492 }
497     493 }
498     494 }
499     495 }
500     496 }
501     497 }
502     498 }
503     499 }
504     500 }
505     501 }
506     502 }
507     503 }
508     504 }
509     505 }
510     506 }
511     507 }
512     508 }
513     509 }
514     510 }
515     511 }
516     512 }
517     513 }
518     514 }
519     515 }
520     516 }
521     517 }
522     518 }
523     519 }
524     520 }
525     521 }
526     522 }
527     523 }
528     524 }
529     525 }
530     526 }
531     527 }
532     528 }
533     529 }
534     530 }
535     531 }
536     532 }
537     533 }
538     534 }
539     535 }
540     536 }
541     537 }
542     538 }
543     539 }
544     540 }
545     541 }
546     542 }
547     543 }
548     544 }
549     545 }
550     546 }
551     547 }
552     548 }
553     549 }
554     550 }
555     551 }
556     552 }
557     553 }
558     554 }
559     555 }
560     556 }
561     557 }
562     558 }
563     559 }
564     560 }
565     561 }
566     562 }
567     563 }
568     564 }
569     565 }
570     566 }
571     567 }
572     568 }
573     569 }
574     570 }
575     571 }
576     572 }
577     573 }
578     574 }
579     575 }
580     576 }
581     577 }
582     578 }
583     579 }
584     580 }
585     581 }
586     582 }
587     583 }
588     584 }
589     585 }
590     586 }
591     587 }
592     588 }
593     589 }
594     590 }
595     591 }
596     592 }
597     593 }
598     594 }
599     595 }
600     596 }
601     597 }
602     598 }
603     599 }
604     600 }
605     601 }
606     602 }
607     603 }
608     604 }
609     605 }
610     606 }
611     607 }
612     608 }
613     609 }
614     610 }
615     611 }
616     612 }
617     613 }
618     614 }
619     615 }
620     616 }
621     617 }
622     618 }
623     619 }
624     620 }
625     621 }
626     622 }
627     623 }
628     624 }
629     625 }
630     626 }
631     627 }
632     628 }
633     629 }
634     630 }
635     631 }
636     632 }
637     633 }
638     634 }
639     635 }
640     636 }
641     637 }
642     638 }
643     639 }
644     640 }
645     641 }
646     642 }
647     643 }
648     644 }
649     645 }
650     646 }
651     647 }
652     648 }
653     649 }
654     650 }
655     651 }
656     652 }
657     653 }
658     654 }
659     655 }
660     656 }
661     657 }
662     658 }
663     659 }
664     660 }
665     661 }
666     662 }
667     663 }
668     664 }
669     665 }
670     666 }
671     667 }
672     668 }
673     669 }
674     670 }
675     671 }
676     672 }
677     673 }
678     674 }
679     675 }
680     676 }
681     677 }
682     678 }
683     679 }
684     680 }
685     681 }
686     682 }
687     683 }
688     684 }
689     685 }
690     686 }
691     687 }
692     688 }
693     689 }
694     690 }
695     691 }
696     692 }
697     693 }
698     694 }
699     695 }
700     696 }
701     697 }
702     698 }
703     699 }
704     700 }
705     701 }
706     702 }
707     703 }
708     704 }
709     705 }
710     706 }
711     707 }
712     708 }
713     709 }
714     710 }
715     711 }
716     712 }
717     713 }
718     714 }
719     715 }
720     716 }
721     717 }
722     718 }
723     719 }
724     720 }
725     721 }
726     722 }
727     723 }
728     724 }
729     725 }
730     726 }
731     727 }
732     728 }
733     729 }
734     730 }
735     731 }
736     732 }
737     733 }
738     734 }
739     735 }
740     736 }
741     737 }
742     738 }
743     739 }
744     740 }
745     741 }
746     742 }
747     743 }
748     744 }
749     745 }
750     746 }
751     747 }
752     748 }
753     749 }
754     750 }
755     751 }
756     752 }
757     753 }
758     754 }
759     755 }
760     756 }
761     757 }
762     758 }
763     759 }
764     760 }
765     761 }
766     762 }
767     763 }
768     764 }
769     765 }
770     766 }
771     767 }
772     768 }
773     769 }
774     770 }
775     771 }
776     772 }
777     773 }
778     774 }
779     775 }
780     776 }
781     777 }
782     778 }
783     779 }
784     780 }
785     781 }
786     782 }
787     783 }
788     784 }
789     785 }
790     786 }
791     787 }
792     788 }
793     789 }
794     790 }
795     791 }
796     792 }
797     793 }
798     794 }
799     795 }
800     796 }
801     797 }
802     798 }
803     799 }
804     800 }
805     801 }
806     802 }
807     803 }
808     804 }
809     805 }
810     806 }
811     807 }
812     808 }
813     809 }
814     810 }
815     811 }
816     812 }
817     813 }
818     814 }
819     815 }
820     816 }
821     817 }
822     818 }
823     819 }
824     820 }
825     821 }
826     822 }
827     823 }
828     824 }
829     825 }
830     826 }
831     827 }
832     828 }
833     829 }
834     830 }
835     831 }
836     832 }
837     833 }
838     834 }
839     835 }
840     836 }
841     837 }
842     838 }
843     839 }
844     840 }
845     841 }
846     842 }
847     843 }
848     844 }
849     845 }
850     846 }
851     847 }
852     848 }
853     849 }
854     850 }
855     851 }
856     852 }
857     853 }
858     854 }
859     855 }
860     856 }
861     857 }
862     858 }
863     859 }
864     860 }
865     861 }
866     862 }
867     863 }
868     864 }
869     865 }
870     866 }
871     867 }
872     868 }
873     869 }
874     870 }
875     871 }
876     872 }
877     873 }
878     874 }
879     875 }
880     876 }
881     877 }
882     878 }
883     879 }
884     880 }
885     881 }
886     882 }
887     883 }
888     884 }
889     885 }
890     886 }
891     887 }
892     888 }
893     889 }
894     890 }
895     891 }
896     892 }
897     893 }
898     894 }
899     895 }
900     896 }
901     897 }
902     898 }
903     899 }
904     900 }
905     901 }
906     902 }
907     903 }
908     904 }
909     905 }
910     906 }
911     907 }
912     908 }
913     909 }
914     910 }
915     911 }
916     912 }
917     913 }
918     914 }
919     915 }
920     916 }
921     917 }
922     918 }
923     919 }
924     920 }
925     921 }
926     922 }
927     923 }
928     924 }
929     925 }
930     926 }
931     927 }
932     928 }
933     929 }
934     930 }
935     931 }
936     932 }
937     933 }
938     934 }
939     935 }
940     936 }
941     937 }
942     938 }
943     939 }
944     940 }
945     941 }
946     942 }
947     943 }
948     944 }
949     945 }
950     946 }
951     947 }
952     948 }
953     949 }
954     950 }
955     951 }
956     952 }
957     953 }
958     954 }
959     955 }
960     956 }
961     957 }
962     958 }
963     959 }
964     960 }
965     961 }
966     962 }
967     963 }
968     964 }
969     965 }
970     966 }
971     967 }
972     968 }
973     969 }
974     970 }
975     971 }
976     972 }
977     973 }
978     974 }
979     975 }
980     976 }
981     977 }
982     978 }
983     979 }
984     980 }
985     981 }
986     982 }
987     983 }
988     984 }
989     985 }
990     986 }
991     987 }
992     988 }
993     989 }
994     990 }
995     991 }
996     992 }
997     993 }
998     994 }
999     995 }
1000    996 }

```

```

232   T ans=1e99;
233   for(int i=0;i<n;++i){
234       while((P[l]-P[l+1]).cross(Q[r+1]-Q[r])<0)r=(r+1)%m;
235       ans=min(ans,line<T>(P[l],P[l+1]).seg_dis2(line<T>(Q[r],
236           Q[r+1])));
237       l=(l+1)%n;
238   }
239   return P.pop_back(),Q.pop_back(),ans;
240 }
241 static char sign(const point<T>&t){
242     return (t.y==0?t.x:t.y)<0;
243 }
244 static bool angle_cmp(const line<T>& A,const line<T>& B){
245     point<T> a=A.p2-A.p1,b=B.p2-B.p1;
246     return sign(a)<sign(b)||((sign(a)==sign(b)&&a.cross(b)>0);
247 }
248 int halfplane_intersection(vector<line<T> > &s){//半平面交
249     sort(s.begin(),s.end(),angle_cmp); //線段左側為該線段半
250     面
251     int L,R,n=s.size();
252     vector<point<T> > px(n);
253     vector<line<T> > q(n);
254     q[L=R=0]=s[0];
255     for(int i=1;i<n;++i){
256         while(L<R&&s[i].ori(px[R-1])<=0)--R;
257         while(L<R&&s[i].ori(px[L])<=0)+L;
258         q[++R]=s[i];
259         if(q[R].parallel(q[R-1])){
260             --R;
261             if(q[R].ori(s[i].p1)>0)q[R]=s[i];
262         }
263         if(L<R)px[R-1]=q[R-1].line_intersection(q[R]);
264     }
265     while(L<R&&q[L].ori(px[R-1])<=0)--R;
266     p.clear();
267     if(R-L<=1)return 0;
268     px[R]=q[R].line_intersection(q[L]);
269     for(int i=L;i<R;++i)p.push_back(px[i]);
270     return R-L+1;
271 }
272 template<typename T>
273 struct triangle{
274     point<T> a,b,c;
275     triangle(){
276         triangle(const point<T> &a,const point<T> &b,const point<T>
277             &c):a(a),b(b),c(c){}
278     }
279     T area()const{
280         T t=(b-a).cross(c-a)/2;
281         return t>0?t:-t;
282     }
283     point<T> barycenter()const{//重心
284         return (a+b+c)/3;
285     }
286     point<T> circumcenter()const{//外心
287         static line<T> u,v;
288         u.p1=(a+b)/2;
289         u.p2=point<T>(u.p1.x-a.y+b.y,u.p1.y+a.x-b.x);
290         v.p1=(a+c)/2;
291         v.p2=point<T>(v.p1.x-a.y+c.y,v.p1.y+a.x-c.x);
292         return u.line_intersection(v);
293     }
294     point<T> incenter()const{//內心
295         T A=sqrt((b-c).abs2()),B=sqrt((a-c).abs2()),C=sqrt((a-b).
296             abs2());
297         return point<T>(A*a.x+B*b.x+C*c.x,A*a.y+B*b.y+C*c.y)/(A+B
298             +C);
299     }
300 }
301 template<typename T>
302 struct point3D{
303     T x,y,z;
304     point3D(){
305         point3D(const T&x,const T&y,const T&z):x(x),y(y),z(z){}
306     }
307     point3D operator+(const point3D &b)const{
308         return point3D(x+b.x,y+b.y,z+b.z);}
309     point3D operator-(const point3D &b)const{
310         return point3D(x-b.x,y-b.y,z-b.z);}
311     point3D operator*(const T &b)const{
312         return point3D(x*b,y*b,z*b);}
313     point3D operator/(const T &b)const{
314         return point3D(x/b,y/b,z/b);}
315     bool operator==(const point3D &b)const{
316         return x==b.x&&y==b.y&&z==b.z;}
317     T dot(const point3D &b)const{
318         return x*b.x+y*b.y+z*b.z;}
319     point3D cross(const point3D &b)const{
320         return point3D(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);}
321     T abs2()const{//向量長度的平方
322         return dot(*this);}
323     T area2(const point3D &b)const{//和b、原點圍成面積的平方
324         return cross(b).abs2()/4;}
325 }
326 template<typename T>
327 struct line3D{
328     point3D<T> p1,p2;
329     line3D(){
330         line3D(const point3D<T> &p1,const point3D<T> &p2):p1(p1),p2
331             (p2){}
332     }
333     T dis2(const point3D<T> &p,bool is_segment=0)const{//點跟直
334         線/線段的距離平方
335         point3D<T> v=p2-p1,v1=p-p1;
336         if(is_segment){
337             point3D<T> v2=p-p2;
338             if(v.dot(v1)<=0)return v1.abs2();
339             if(v.dot(v2)>=0)return v2.abs2();
340         }
341         point3D<T> tmp=v.cross(v1);
342         return tmp.abs2()/v.abs2();
343     }
344     pair<point3D<T>,point3D<T> > closest_pair(const line3D<T> &
345         l)const{
346         point3D<T> v1=(p1-p2),v2=(l.p1-l.p2);
347         point3D<T> N=v1.cross(v2),ab(p1-l.p1);
348         //if(N.abs2()==0)return NULL;平行或重合
349         T tmp=N.dot(ab),ans=tmp*tmp/N.abs2(); //最近點對距離
350         point3D<T> d1=p2-p1,d2=l.p2-l.p1,D=d1.cross(d2),G=l.p1-p1
351             ;
352         T t1=(G.cross(d2)).dot(D)/D.abs2();
353         T t2=(G.cross(d1)).dot(D)/D.abs2();
354         return make_pair(p1+d1*t1,l.p1+d2*t2);
355     }
356     bool same_side(const point3D<T> &a,const point3D<T> &b)
357         const{
358         return (p2-p1).cross(a-p1).dot((p2-p1).cross(b-p1))>0;
359     }
360 }
361 template<typename T>
362 struct plane{
363     point3D<T> p0,n; //平面上的點和法向量
364     plane(){
365         plane(const point3D<T> &p0,const point3D<T> &n):p0(p0),n(n)
366             {}
367     }
368     T dis2(const point3D<T> &p)const{//點到平面距離的平方
369         T tmp=(p-p0).dot(n);
370         return tmp*tmp/n.abs2();
371     }
372     point3D<T> projection(const point3D<T> &p)const{
373         return p-n*(p-p0).dot(n)/n.abs2();
374     }
375     point3D<T> line_intersection(const line3D<T> &l)const{
376         T tmp=n.dot(l.p2-l.p1); //等於0表示平行或重合該平面
377         return l.p1+(l.p2-l.p1)*(n.dot(p0-l.p1)/tmp);
378     }
379     line3D<T> plane_intersection(const plane &pl)const{
380         point3D<T> e=n.cross(pl.n),v=n.cross(e);
381         T tmp=pl.n.dot(v); //等於0表示平行或重合該平面
382         point3D<T> q=p0+(v*(pl.n.dot(pl.p0-p0))/tmp);
383         return line3D<T>(q,q+e);
384     }
385 };
386 template<typename T>
387 struct triangle3D{
388     point3D<T> a,b,c;
389     triangle3D(){
390         triangle3D(const point3D<T> &a,const point3D<T> &b,const
391             point3D<T> &c):a(a),b(b),c(c){}
392     }
393     bool point_in(const point3D<T> &p)const{//點在該平面上的投
394         影在三角形中
395         return line3D<T>(b,c).same_side(p,a)&&line3D<T>(a,c).
396             same_side(p,b)&&line3D<T>(a,b).same_side(p,c);
397     }
398 };
399 template<typename T>
400 struct tetrahedron{//四面體
401     point3D<T> a,b,c,d;
402     tetrahedron(){
403         tetrahedron(const point3D<T> &a,const point3D<T> &b,const
404             point3D<T> &c,const point3D<T> &d):a(a),b(b),c(c),d(d)
405             {}
406     }
407     T volume6()const{//體積的六倍
408         return (d-a).dot((b-a).cross(c-a));
409     }
410     point3D<T> centroid()const{
411         return (a+b+c+d)/4;
412     }
413     bool point_in(const point3D<T> &p)const{
414         return triangle3D<T>(a,b,c).point_in(p)&&triangle3D<T>(c,
415             d,a).point_in(p);
416     }
417 };
418 template<typename T>
419 struct convexhull3D{
420     static const int MAXN=1005;
421     struct face{
422         int a,b,c;
423         face(int a,int b,int c):a(a),b(b),c(c){}
424     };
425     vector<point3D<T>> pt;
426     vector<face> ans;

```



```

408 int fid[MAXN][MAXN];
409 void build(){
410     int n=pt.size();
411     ans.clear();
412     memset(fid,0,sizeof(fid));
413     ans.emplace_back(0,1,2); //注意不能共線
414     ans.emplace_back(2,1,0);
415     int ftop = 0;
416     for(int i=3, ftop=1; i<n; ++i, ++ftop){
417         vector<face> next;
418         for(auto &f:ans){
419             T d=(pt[i]-pt[f.a]).dot((pt[f.b]-pt[f.a]).cross(pt[f.
420                 c]-pt[f.a]));
421             if(d<=0) next.push_back(f);
422             int ff=0;
423             if(d>0) ff=ftop;
424             else if(d<0) ff=-ftop;
425             fid[f.a][f.b]=fid[f.b][f.c]=fid[f.c][f.a]=ff;
426         }
427         for(auto &f:ans){
428             if(fid[f.a][f.b]>0 && fid[f.a][f.b]!=fid[f.b][f.a])
429                 next.emplace_back(f.a,f.b,i);
430             if(fid[f.b][f.c]>0 && fid[f.b][f.c]!=fid[f.c][f.b])
431                 next.emplace_back(f.b,f.c,i);
432             if(fid[f.c][f.a]>0 && fid[f.c][f.a]!=fid[f.a][f.c])
433                 next.emplace_back(f.c,f.a,i);
434         }
435         ans=next;
436     }
437     point3D<T> centroid()const{
438         point3D<T> res(0,0,0);
439         T vol=0;
440         for(auto &f:ans){
441             T tmp=pt[f.a].dot(pt[f.b].cross(pt[f.c]));
442             res=res+(pt[f.a]+pt[f.b]+pt[f.c])*tmp;
443             vol+=tmp;
444         }
445         return res/(vol*4);
446     }
447 };

```

### 3.6 nearestDist

```

1 bool cmp_y(P a, P b){
2     return a.y < b.y;
3 }
4
5 bool cmp_x(P a, P b){
6     return a.x < b.x;
7 }
8
9 double dc(P *arr, int n){
10     if(n == 1) return INF;
11     int mid = n/2;
12     double cx = arr[mid].x;
13     double dist = min( dc(arr, mid), dc(arr+mid, n-mid) );
14     inplace_merge(arr, arr+mid, arr+n, cmp_y);
15     static vector<P> brr; brr.clear();
16     for(int i = 0; i < n; i++){
17         if(fabs(arr[i].x)-cx >= dist) continue;
18         for(int j = brr.size()-1; j >= 0; j--){
19             double dx = brr[j].x-arr[i].x;

```

```

20             double dy = brr[j].y-arr[i].y;
21             if(fabs(dy) >= dist) break;
22             dist = min(dist, sqrt(dx*dx+dy*dy));
23         }
24         brr.push_back(arr[i]);
25     }
26     return dist;
27 }
28
29 double nearestDist(P *arr, int n){
30     sort(arr, arr+n, cmp_x);
31     return dc(arr, n);
32 }

```

### 3.7 convexHull

```

1 struct ConvexHull {
2     vector<Point> h; // hull
3     static bool cmp(const Point &lhs, const Point &rhs) {
4         if(lhs.x == rhs.x) return lhs.y < rhs.y;
5         else return lhs.x < rhs.x;
6     }
7     // p : points, h : return hull
8     void buildHull(vector<Point> &p) {
9         int n = p.size(), m = 0;
10        sort(p.begin(), p.end(), cmp);
11        h.resize(n+1);
12        for(int i = 0; i < n; ++i){
13            while(m>=2&&(h[m-1]-h[m-2]).cross(p[i]-h[m-2])<=0)--m;
14            h[m++]=p[i];
15        }
16        for(int i = n-2, t = m+1; i >= 0; --i) {
17            while(m>=t&&(h[m-1]-h[m-2]).cross(p[i]-h[m-2])<=0)--m;
18            h[m++]=p[i];
19        }
20        if(h.size()>1)--m;
21        h.resize(m);
22    }
23 };

```

```

15 stack<int> stk;
16 vector<bool> inStk;
17 vector<int> scc; // scc[v] = id of scc
18 void reportSCC(int v){
19     int x;
20     do{
21         x = stk.top(); stk.pop();
22         inStk[x] = false;
23         scc[x] = sccIdx;
24     } while(x != v);
25     sccIdx++;
26 }
27 void dfs(int v) {
28     low[v] = dep[v] = ++ts;
29     stk.push(v); inStk[v] = true;
30     for(auto c : graph[v]){
31         if(dep[c] == 0){ // not visited
32             dfs(c);
33             low[v] = min(low[v], low[c]);
34         }
35         if(inStk[c]) low[v] = min(low[v], dep[c]);
36     }
37     if(low[v] == dep[v]) reportSCC(v);
38 }
39 public:
40     void init(int v){
41         vn = v, ts = 0, sccIdx = 0;
42         graph.resize(v);
43         low.resize(v, 0);
44         dep.resize(v, 0);
45         scc.resize(v, 0);
46         inStk.resize(v, false);
47     }
48     void addEdge(int u, int v){
49         graph[u].emplace_back(v);
50     }
51     void run(){
52         for(int v = 0; v < vn; v++){
53             if(dep[v] == 0) dfs(v);
54         }
55         int getSCCId(int v) { return scc[v]; }
56 };

```

### 4.2 lca

```

1 const int MAXN=100000; // 1-base
2 const int MLG=17; //log2(MAXN)+1;
3 int pa[MLG+2][MAXN+5];
4 int dep[MAXN+5];
5 vector<int> G[MAXN+5];
6 void dfs(int x, int p=0){ //dfs(root);
7     pa[0][x]=p;
8     for(int i=0; i<=MLG; ++i)
9         pa[i+1][x]=pa[i][pa[i][x]];
10    for(auto &i:G[x]){
11        if(i==p) continue;
12        dep[i]=dep[x]+1;
13        dfs(i,x);
14    }
15 }
16 inline int jump(int x, int d){
17     for(int i=0; i<=MLG; ++i)
18         if((d>>i)&1) x=pa[i][x];

```

## 4 Graph

### 4.1 SCC

```

1 /*
2  * SCC in an directed graph
3  * usage : init(), addEdge(), run()
4  * 0-base graph
5  */
6 class DirectedTarjan
7 {
8 private:
9     int vn;
10    int ts; // dfs timestamp
11    int sccIdx;
12    vector<vector<int>> graph;
13    vector<int> low;
14    vector<int> dep;

```

```

19     return x;
20 }
21 inline int find_lca(int a,int b){
22     if(dep[a]>dep[b])swap(a,b);
23     b=jump(b,dep[b]-dep[a]);
24     if(a==b)return a;
25     for(int i=MLG;i>=0;--i){
26         if(pa[i][a]!=pa[i][b]){
27             a=pa[i][a];
28             b=pa[i][b];
29         }
30     }
31     return pa[0][a];
32 }

```

### 4.3 bellman\_Ford

```

1 struct edge{ int from, to, cost; };
2 #define INF 2147483647
3
4 edge es[100];
5
6 int d[100]; //min distance
7 int V, E, s, f;
8
9 bool bellman_ford() // return true if there is negative loop
10 {
11     for(int i = 0; i < V; i++) d[i] = INF;
12     d[s] = 0;
13
14     for(int i = 0; i < V; i++)
15     {
16         for(int j = 0; j < E; j++)
17         {
18             edge e = es[j];
19             if(d[e.from] != INF && d[e.to] > d[e.from] + e.
20                 cost)
21             {
22                 d[e.to] = d[e.from] + e.cost;
23                 if(i == V - 1) return true; //got neg loop
24             }
25             if(d[e.to] != INF && d[e.from] > d[e.to] + e.cost
26                 )
27             {
28                 d[e.from] = d[e.to] + e.cost;
29                 if(i == V - 1) return true; //got neg loop
30             }
31         }
32     }
33     return false;
34 }

```

### 4.4 MaxMatching

```

1 #define FZ(x) memset(x,0,sizeof(x))
2 struct GenMatch // 1-base
3 {
4     static const int MAXN = 250;
5     int V;
6     bool el[MAXN][MAXN];

```

```

7     int pr[MAXN];
8     bool inq[MAXN],inp[MAXN],inb[MAXN];
9     queue<int> qe;
10     int st,ed;
11     int nb;
12     int bk[MAXN],djs[MAXN];
13     int ans;
14     void init(int _V){
15         V = _V;
16         FZ(el);
17         FZ(pr);
18         FZ(inq);
19         FZ(inp);
20         FZ(inb);
21         FZ(bk);
22         FZ(djs);
23         ans = 0;
24     }
25     void add_edge(int u, int v){
26         el[u][v] = el[v][u] = 1;
27     }
28     int lca(int u,int v){
29         memset(inp,0,sizeof(inp));
30         while(1){
31             u = djs[u];
32             inp[u] = true;
33             if(u == st) break;
34             u = bk[pr[u]];
35         }
36         while(1){
37             v = djs[v];
38             if(inp[v]) return v;
39             v = bk[pr[v]];
40         }
41         return v;
42     }
43     void upd(int u){
44         int v;
45         while(djs[u] != nb){
46             v = pr[u];
47             inb[djs[u]] = inb[djs[v]] = true;
48             u = bk[v];
49             if(djs[u] != nb) bk[u] = v;
50         }
51     }
52     void blo(int u,int v){
53         nb = lca(u,v);
54         memset(inb,0,sizeof(inb));
55         upd(u);
56         upd(v);
57         if(djs[u] != nb) bk[u] = v;
58         if(djs[v] != nb) bk[v] = u;
59         for(int tu = 1; tu <= V; tu++)
60             if(inb[djs[tu]]){
61                 djs[tu] = nb;
62                 if(!inq[tu]){
63                     qe.push(tu);
64                     inq[tu] = 1;
65                 }
66             }
67     }
68     void flow(){
69         memset(inq,false,sizeof(inq));
70         memset(bk,0,sizeof(bk));
71         for(int i = 1; i <= V; i++)
72             djs[i] = i;

```

```

73
74     while(qe.size()) qe.pop();
75     qe.push(st);
76     inq[st] = 1;
77     ed = 0;
78     while(qe.size()){
79         int u = qe.front();
80         qe.pop();
81         for(int v = 1; v <= V; v++){
82             if(el[u][v] && (djs[u] != djs[v]) && (pr[u]
83                 != v)){
84                 if((v == st) || ((pr[v] > 0) && bk[pr[v]]
85                     > 0))
86                     blo(u,v);
87                 else if(bk[v] == 0){
88                     bk[v] = u;
89                     if(pr[v] > 0){
90                         if(!inq[pr[v]]) qe.push(pr[v]);
91                     }
92                     else{
93                         ed = v;
94                         return;
95                     }
96                 }
97             }
98         }
99     }
100     void aug(){
101         int u,v,w;
102         u = ed;
103         while(u > 0){
104             v = bk[u];
105             w = pr[v];
106             pr[v] = u;
107             pr[u] = v;
108             u = w;
109         }
110     }
111     int solve(){
112         memset(pr,0,sizeof(pr));
113         for(int u = 1; u <= V; u++){
114             if(pr[u] == 0){
115                 st = u;
116                 flow();
117                 if(ed > 0){
118                     aug();
119                     ans ++;
120                 }
121             }
122         }
123     }
124     return ans;
125 }
126 } gm;

```

### 4.5 MinimumMeanCycle

```

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

```

```
1 #define MAXN 32
2 int n, m, Max;
3 ll v[MAXN], deg[MAXN]; //neighbors
```

```

5 void update_maximum(ll R){
6     int Size = 0;
7     while(R){
8         if(R&1)Size++;
9         R>>=1;
10    }
11    Max = max(Size, Max);
12 }
13
14 int pickPivot(ll P){
15     int pivot = -1, Max = -1;
16     memset(deg, 0, sizeof(deg));
17     for(int i = 0; i < n; i++){
18         if(P&(1LL<<i)){//i is in P
19             if(pivot == -1){//i = default pivot
20                 pivot = i;
21                 Max = deg[i];
22             }
23             for(int j = 0; j < i; j++){
24                 if((P&(1LL<<j))&&(v[i]&(1LL<<j)))){
25                     deg[i]++;
26                     if(deg[i] > Max){
27                         Max = deg[i];
28                         pivot = i;
29                     }
30                     deg[j]++;
31                     if(deg[j] > Max){
32                         Max = deg[j];
33                         pivot = j;
34                     }
35                 }
36             }
37         }
38     }
39     return pivot;
40 }
41
42 void BronKerbosch(ll R, ll P, ll X){
43     if(!P){//P is empty, no candidates left
44         if(!X){
45             //clique
46             update_maximum(R);
47         }
48         return;
49     }
50     int u = pickPivot(P|X);
51     for(int i = 0; i <= n-1; i++){
52         if(P&(~v[u])&(1LL<<i)){//vi is in P
53             BronKerbosch( R|(1LL<<i), P&v[i], X&v[i] );
54             P&=~(1LL<<i));
55             X|=(1LL<<i);
56         }
57     }
58 }
59
60 int main(){
61     ios::sync_with_stdio(false);
62     cin.tie(0);
63     while(cin >> n){
64         cin >> m;
65
66         Max = 0;
67         FOR(i,0,n-1)v[i] = 0;
68
69         int a, b;

```

[illegible]



```

24     }
25     }
26 }
27 return false;
28 }

```

### 4.13 dijkstra

```

1 struct edge{int to, cost;};
2 typedef pair<int, int> P; //first = min distance, second = v
3     id
4 #define f first
5 #define s second
6 #define INF 2147483647
7
8 int V, E, S, F;
9 vector<edge> G[100];
10 int d[100];
11
12 void dijkstra()
13 {
14     priority_queue<P, vector<P>, greater<P>> q;
15     fill(d, d + V, INF);
16     d[S] = 0;
17     q.push(P(0, S));
18
19     while(!q.empty())
20     {
21         P p = q.top(); q.pop();
22         int v = p.s;
23         if(d[v] < p.f) continue;
24         for(int i = 0; i < G[v].size(); i++)
25         {
26             edge e = G[v][i];
27             if(d[e.to] > d[v] + e.cost)
28             {
29                 d[e.to] = d[v] + e.cost;
30                 q.push(P(d[e.to], e.to));
31             }
32         }
33     }
34 }

```

### 4.14 MaxWeightPerfectBiMatch

```

1 const int maxn = 500 + 3, INF = 0x3f3f3f3f;
2 int n, W[maxn][maxn];
3 int mat[maxn];
4 int Lx[maxn], Ly[maxn], slack[maxn];
5 bool S[maxn], T[maxn];
6
7 inline void tension(int &a, const int b) {
8     if(b < a) a = b;
9 }
10
11 inline bool match(int u) {
12     S[u] = true;
13     for(int v = 0; v < n; ++v) {
14         if(T[v]) continue;

```

```

15         int t = Lx[u] + Ly[v] - W[u][v];
16         if(!t) {
17             T[v] = true;
18             if(mat[v] == -1 || match(mat[v])) {
19                 mat[v] = u;
20                 return true;
21             }
22         }else tension(slack[v], t);
23     }
24     return false;
25 }
26
27 inline void update() {
28     int d = INF;
29     for(int i = 0; i < n; ++i)
30         if(!T[i]) tension(d, slack[i]);
31     for(int i = 0; i < n; ++i) {
32         if(S[i]) Lx[i] -= d;
33         if(T[i]) Ly[i] += d;
34     }
35 }
36
37 inline void KM() {
38     for(int i = 0; i < n; ++i) {
39         Lx[i] = Ly[i] = 0; mat[i] = -1;
40         for(int j = 0; j < n; ++j) Lx[i] = max(Lx[i], W[i][j]
41             );
42     }
43     for(int i = 0; i < n; ++i) {
44         fill(slack, slack + n, INF);
45         while(true) {
46             for(int j = 0; j < n; ++j) S[j] = T[j] = false;
47             if(match(i)) break;
48             else update();
49         }
50     }

```

## 5 Math

### 5.1 extgcd

```

1 int extgcd(int a, int b, int &x, int &y){
2     int gcd = a;
3     if(b != 0)
4         gcd = extgcd(b, a%b, y, x), y -= (a/b)*x;
5     else x = 1, y = 0;
6     return gcd;
7 }
8 //維護 a*x+b*y=gcd(a, b)

```

### 5.2 NTT

```

1 typedef long long ll;
2
3 const ll P = (479<<21)+1;
4 const ll G = 3;

```

```

5 inline ll fpw(ll x, ll y, ll m){
6     ll rtn = 1;
7     for(x=(x>=m?x%m:x);y>=1){
8         if(y&1) rtn = rtn*x%m;
9         x = x*x%m;
10    }
11    return rtn;
12 }
13 inline vector<ll> ntt(vector<ll> rtn, int Rev = 1){
14     int ntt_n = rtn.size();
15     for(int i=0,j=0;i<ntt_n;i++){
16         if(i>j) swap(rtn[i],rtn[j]);
17         for(int k=(ntt_n>>1);(j^=k)<k;k>=1);
18     }
19     for(int i=2,m=1;i<=ntt_n;i<=1,m++){
20         ll w = 1, wn = fpw(G,(P-1)>>m,P), u, t;
21         int mh = i>1;
22         for(int j=0;j<mh;j++){
23             for(int k=j;k<ntt_n;k+=i){
24                 u = rtn[k], t = w*rtn[k+mh]%P;
25                 rtn[k] = (u+t)%P;
26                 rtn[k+mh] = (u-t)%P;
27             }
28             w = w*wn%P;
29         }
30     }
31     if(!~Rev){
32         for(int i=1;i<ntt_n/2;i++) swap(rtn[i],rtn[ntt_n-i]);
33         ll Revn = fpw(ntt_n,P-2,P);
34         for(int i=0;i<ntt_n;i++) rtn[i] = rtn[i]*Revn%P;
35     }
36     return rtn;
37 }
38 // 把原多項式包成 long long 的 vector (poly), 並把項次拓展到 2^i.
39 // 用 ntt(poly) 即可得到轉換後的結果.
40 // Rev 為 1 時為 NTT, 為 -1 時為 InvNTT.

```

### 5.3 GaussianJordan

```

1 const double EPS = 1e-8;
2 typedef vector<double> vec;
3 typedef vector<vec> mat;
4
5 //solve Ax=b
6 //if no sol/inf sol, return vec of size 0
7 vec gauss_jordan(const mat& A, const vec& b){
8     int n = A.size();
9     mat B(n, vec(n+1));
10    for(int i=0;i<n;i++)for(int j=0;j<n;j++)B[i][j]=A[i][j];
11
12    for(int i=0;i<n;i++)B[i][n]=b[i];
13
14    for(int i=0;i<n;i++){
15        int pivot=i;
16        for(int j=i;j<n;j++){
17            if(abs(B[j][i])>abs(B[pivot][i]))pivot=j;
18        }
19        swap(B[i],B[pivot]);
20        if(abs(B[i][i])<EPS)return vec();//no/inf sol
21
22        for(int j=i+1;j<n;j++)B[i][j]/=B[i][i];
23        for(int j=0;j<n;j++){

```

```

24     if(i!=j){
25         for(int k=i+1;k<=n;k++)
26             B[j][k]-=B[j][i]*B[i][k];
27     }
28 }
29 }
30 vec x(n);
31 for(int i=0;i<n;i++)x[i]=B[i][n];
32 return x;
33 }

```

## 5.4 EulerPhi

```

1 //find in O(sqrt(N))
2
3 int euler_phi(int N)
4 {
5     int res=N;
6     for(int i=2;i*i<=N;i++)
7     {
8         if(N%i==0)
9         {
10             res=res/i*(i-1);
11             for(;N%i==0;N/=i);
12         }
13     }
14     if(N!=1)res=res/N*(N-1);//self=prime
15     return res;
16 }
17
18 //tabulate in O(MAXN)
19
20 int euler[MAXN];
21
22 void euler_phi2()
23 {
24     for(int i=0;i<MAXN;i++)euler[i]=i;
25     for(int i=2;i<MAXN;i++)
26     {
27         if(euler[i]==i)
28         {
29             for(int j=i;j<MAXN;j+=i)
30             {
31                 euler[j]=euler[j]/i*(i-1);
32             }
33         }
34     }
35 }

```

## 5.5 FFT

```

1 const double PI = acos(-1.0);
2 struct Complex
3 {
4     double x,y;
5     Complex(){}
6     Complex(double a):x(a),y(0){}
7     Complex(double a, double b):x(a),y(b){}
8     Complex operator+ (const Complex &a){ return Complex(x+a.x,
9         y+a.y); }

```

```

9     Complex operator- (const Complex &a){ return Complex(x-a.x,
10         y-a.y); }
11     Complex operator* (const Complex &a){ return Complex(x*a.x-
12         y*a.y,x*a.y+y*a.x); }
13 };
14 inline vector<Complex> fft(vector<Complex> rtn, int Rev = 1)
15 {
16     int fft_n = rtn.size();
17     for(int i=0,j=0;i<fft_n;i++)
18     {
19         if(i>j) swap(rtn[i],rtn[j]);
20         for(int k=(fft_n>>1);(j^=k)<k;k>>=1);
21     }
22     for(int i=2,m;i<=fft_n;i<=1)
23     {
24         m = i>>1;
25         for(int j=0;j<fft_n;j+=i)
26         {
27             for(int k=0;k<m;k++)
28             {
29                 Complex y = rtn[j+k+m]*Complex(cos(2*PI/i*k), Rev*sin
30                     (2*PI/i*k));
31                 rtn[j+k+m] = rtn[j+k]-y;
32                 rtn[j+k] = rtn[j+k]+y;
33             }
34         }
35     }
36     for(int i=0;~Rev&&i<fft_n;i++)
37         rtn[i].x = rtn[i].x/fft_n;
38     return rtn;
39 }
40 // Complex的x為實部, y為虛部.
41 // 把原多項式包成Complex的vector(poly), 並把項次拓展到2^i, 用
42 // fft(poly)即可得到轉換後的結果.
43 // Rev為1時為FFT, 為-1時為InvFFT.

```

## 5.6 BigInt

```

1 #define MAX_N 1000
2 #define MAX 100000
3 #define MAX_LOG 5
4 class BigInt{
5 public:
6     int sign;
7     long long m[MAX_N];
8     int l;
9     long long operator [](int i) const { return m[i]; }
10    long long &operator [](int i) { return m[i]; }
11    BigInt() { l=1, m[0]=0; sign=1; }
12    BigInt(int x){ (*this)=x; }
13    BigInt(const char *s){ (*this)=s; }
14    BigInt operator =(int x){
15        if(x<0) x=-x, sign=-1;
16        else sign=1;
17        for(l=1, m[l-1]=x%MAX, x/=MAX; x; m[l++]=x%MAX, x/=MAX)
18            ;
19        if(sign==1&&l==1&&m[0]==0) sign=1;
20        return *this;
21    }
22    BigInt operator =(const char *t){
23        int i, j, len;
24        const char *s;

```

```

25        if(t[0]=='-') sign=-1, s=t+1;
26        else sign=1, s=t;
27        for(len=0; s[len]>='0' && s[len]<='9'; len++);
28        for(l=(len+MAX_LOG-1)/MAX_LOG, i=0; i<l; i++)
29            for(m[i]=0, j=0; j<MAX_LOG; j++)
30                if(len-i*MAX_LOG-MAX_LOG+j>=0)
31                    m[i]=m[i]*10+s[len-i*MAX_LOG-MAX_LOG+j]-'0';
32        if(sign==1&&l==1&&m[0]==0) sign=1;
33        return *this;
34    }
35    bool scan(){
36        char s[MAX_N*MAX_LOG+10];
37        if(scanf("%s", s)==EOF) return 0;
38        else { *this=s; return 1; }
39    }
40    void print(){
41        int i;
42        char s[8];
43        if(sign==1) printf("-");
44        for(sprintf(s, "%0%lld", MAX_LOG), printf("%lld", m[l-1]), i=l-2; i>=0; printf(s, m[i]), i--);
45    }
46    bool operator <(const BigInt &x, const BigInt &y){
47        if(x.sign!=y.sign) return x.sign<y.sign;
48        int i;
49        if(x.l!=y.l) return (x.l<y.l) ^ (x.sign==1);
50        for(i=x.l-1; i>=0 && x[i]!=y[i]; i--);
51        return (i>=0 && x[i]<y[i]) ^ (x.sign==1);
52    }
53    bool operator ==(const BigInt &x, const BigInt &y){
54        if(x.sign!=y.sign) return false;
55        int i;
56        if(x.l!=y.l) return 0;
57        for(i=x.l-1; i>=0 && x[i]!=y[i]; i--);
58        return i<0;
59    }
60    BigInt operator +(BigInt x, const BigInt &y){
61        int i;
62        long long h;
63        for(h=0, i=0; i<x.l || i<y.l || h; i++){
64            h+=(i<x.l)*x[i]*x.sign+(i<y.l)*y[i]*y.sign;
65            x[i]=h%MAX;
66            h/=MAX;
67        }
68        x.l=i;
69        for(; x.l>1 && !x[x.l-1]; x.l--);
70        x.sign=(x[x.l-1]>0?1:-1);
71        if(x[x.l-1]>0){ for(i=0; i<x.l; i++) if(x[i]<0) x[i+1]--, x[i]+=MAX; }
72        else for(i=0; i<x.l; i++) if(x[i]>0) x[i+1]++, x[i]-=MAX;
73        for(i=0; i<x.l; i++) x[i]*=x.sign;
74        if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
75        return x;
76    }
77    BigInt operator -(BigInt x, const BigInt &y){
78        int i;
79        long long h;
80        for(h=0, i=0; i<x.l || i<y.l || h; i++){
81            h+=(i<x.l)*x[i]*x.sign-(i<y.l)*y[i]*y.sign;
82            x[i]=h%MAX;
83            h/=MAX;
84        }
85        x.l=i;
86        for(; x.l>1 && !x[x.l-1]; x.l--);
87        x.sign=(x[x.l-1]>0?1:-1);

```



```

88 if(x[x.l-1]>0){ for(i=0; i<x.l; i++) if(x[i]<0) x[i+1]--, x[i] += MAX; }
89 else for(i=0; i<x.l; i++) if(x[i]>0) x[i+1]++, x[i] -= MAX;
90 for(; x.l>1 && !x[x.l-1]; x.l--);
91 for(i=0; i<x.l; i++) x[i]*=x.sign;
92 if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
93 return x;
94 }
95 BigInt operator *(BigInt x, int y){
96     int i, sign=1;
97     long long h;
98     if(y<0) y=-y, sign=-1;
99     for(h=0, i=0; i<x.l || h; i++){
100         h+=(i<x.l)*x[i]*y;
101         x[i]=h/MAX;
102         h/=MAX;
103     }
104     for(x.l=i; x.l>1 && !x[x.l-1]; x.l--);
105     x.sign=(x.sign==sign?-1);
106     if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
107     return x;
108 }
109 BigInt operator /(BigInt x, int y){
110     int i, sign=1;
111     long long h;
112     if(y<0) y=-y, sign=-1;
113     for(h=0, i=x.l-1; i>=0; i--){
114         h=h*MAX+x[i];
115         x[i]=h/y;
116         h%=y;
117     }
118     for(; x.l>1 && !x[x.l-1]; x.l--);
119     x.sign=(x.sign==sign?-1);
120     if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
121     return x;
122 }
123 int operator %(BigInt x, int y){
124     int i;
125     long long h;
126     for(h=0, i=x.l-1; i>=0; i--){
127         h=h*MAX+x[i];
128         h%=y;
129     }
130     if(x.sign==1) h=-h;
131     return h;
132 }
133 long long fl(double x) { return x<0?x-0.5:x+0.5; }
134 BigInt operator *(BigInt x, const BigInt &y){
135     if(y.l==1) return x*(y[0]*y.sign);
136     int i, N;
137     long long t;
138     vector<Complex> a, b;
139     for(i=0; i<x.l; i++) a.emplace_back(x[i]);
140     for(i=0; i<y.l; i++) b.emplace_back(y[i]);
141     for(N=1; N<x.l+y.l; N<=1);
142     while(N!=(int)(a.size())) a.emplace_back(0);
143     while(N!=(int)(b.size())) b.emplace_back(0);
144     a=fft(a), b=fft(b);
145     for(i=0; i<N; i++) a[i]=a[i]*b[i];
146     a=fft(a,-1);
147     for(i=0, t=0, x.l=0; i<N; i++){
148         t+=fl(a[i].x);
149         x[x.l++] = t/MAX;
150         t/=MAX;
151     } x[x.l++] = t;
152     for(; x.l>1 && !x[x.l-1]; x.l--);

```

```

x.sign=(x.sign==y.sign?-1);
if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
return x;
}
BigInt operator /(BigInt x, const BigInt &y){
    if(y.l==1) return x/(y[0]*y.sign);
    int i;
    BigInt h;
    for(h=0, i=x.l-1; i>=0; i--){
        h=h*MAX+x[i];
        if(h.l>y.l) x[i]=(h[h.l-1]*MAX*MAX+h[h.l-2]*MAX+h[h.l-3]);
        if(h.l==y.l) x[i]=(h[h.l-1]*MAX+h[h.l-2]);
        x[i]/=(y[y.l-1]*MAX+y[y.l-2]);
        for(; x[i] && h<y*(x[i]*y.sign); x[i]--);
        h=h-(y*(x[i]*y.sign));
    }
    for(; x.l>1 && !x[x.l-1]; x.l--);
    x.sign=(x.sign==y.sign?-1);
    if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
    return x;
}
BigInt operator %(BigInt x, BigInt y){
    if(y.l==1) return x%(y[0]*y.sign);
    return x-(x/y)*y;
}

```

## 5.7 mobius

```

1 /* Mobius Function
2  * m(x) = 0, x has repeated factors
3  * m(x) = 1, x = 1
4  * m(x) = (-1)^k, x is the product of k distinct primes
5  * f(n) = sum(g(d)) for d|n,
6  * g(n) = sum( mu(n/d)f(d) ) = sum( mu(d)f(n/d) )
7 */
8 const int M = 100005;
9 ll sp[M], mobius[M];
10 void sieve(){
11     for(ll i = 0; i < M; i++) sp[i] = i;
12     for(ll i = 2; i*i < M; i++) if(sp[i] == i)
13     {
14         for(ll j = i*i; j < M; j += i)
15             if(sp[j] == j) sp[j] = i;
16     }
17 }
18 void makeMobius(){
19     for(ll i = 0; i < M; i++) mobius[i] = 1;
20     mobius[0] = 0;
21     for(ll i = 2; i < M; i++) if(sp[i] == i){
22         for(ll j = i; j < M; j += i) mobius[j] = -mobius[j];
23         for(ll j = i*i; j < M; j += i*i) mobius[j] = 0;
24     }
25 }

```

## 5.8 modeq

```

1 ll solve(ll *a, ll *b, ll *m, int N)
2 {
3     ll k = 0, h = 1, gcd, n, t, ar;

```

```

4     for (ll i = 0; i < N; i++)
5     {
6         gcd = extgcd(a[i] * h, m[i], ar, t);
7         t = (b[i] - a[i] * k) / gcd, n = abs(m[i] / gcd);
8         if (t * gcd != b[i] - a[i] * k) return -1;
9         t = ((ar * t) % n + n) % n;
10        k += h * t, h *= n, k %= h;
11    }
12    return (k % h + h) % h;
13 }
14 // 解n組a*x=b%m, 回傳x.
15 // 回傳的是最小非負整數解。無解回傳-1.

```

## 6 String

### 6.1 BWT

```

1 // use with suffix array
2 int pivot;
3 // BWT array size must be double of the data size
4 inline void BWT(char *tmp, char *in, char *out, int *SA, int
5     *Rank){
6     int len=strlen(in);
7     for(int i=0;i<len;i++) tmp[i]=tmp[i+len]=in[i];
8     tmp[len*2]='|0';
9     SA_build(SA,Rank,tmp);
10    for(int i=0, j=0;i<2*len;i++){
11        if(SA[i]==len) pivot=j;
12        if(SA[i]<len)
13            out[j++]=in[(SA[i]+len-1)%len];
14    }
15    out[len]='|0';
16 }
17 inline void IBWT(char *in, char *out, int *tmp){
18     int len=strlen(in);
19     vector<int> idx[256];
20     for(int i=0;i<len;i++)
21         idx[in[i]].emplace_back(i);
22     for(int i=0,k=0;i<256;i++){
23         for(int j=0;j<(int)(idx[i].size());j++)
24             tmp[k++]=idx[i][j];
25     }
26     int p=pivot;
27     for(int i=0;i<len;i++)
28         out[i]=in[p=tmp[p]];
29     out[len]='|0';
}

```

### 6.2 SuffixArray

```

1 void SA_radix_sort(int *s, int *e, int *Rank, int rankcnt){
2     int box[MAX_N], tmp[MAX_N], len=e-s;
3     memset(box,0,sizeof(int)*rankcnt);
4     for(int i=0;i<len;i++) box[Rank[i]]++;
5     for(int i=1;i<rankcnt;i++) box[i]=box[i]+box[i-1];
6     for(int i=len-1;i>=0;i--) tmp[--box[Rank[s[i]]]]=s[i];
7     for(int i=0;i<len;i++) s[i]=tmp[i];
}

```

```

8 }
9 #define equal(a,b,c) c[a]!=c[b]||a+k>=len||c[a+k]!=c[b+k]
10 void SA_build(int *SA, int *Rank, char *S){
11     int ranktmp[MAX_N], len=strlen(S), rankcnt='z'+1;
12     for(int i=0;i<len;i++) Rank[i]=S[i];
13     for(int k=1;rankcnt!=len;k*=2){
14         for(int i=0;i<len;i++) SA[i]=(i+len-k)%len;
15         SA_radix_sort(SA+k, SA+len, Rank+k, rankcnt);
16         SA_radix_sort(SA, SA+len, Rank, rankcnt);
17         ranktmp[SA[0]]=0, rankcnt=0;
18         for(int i=1;i<len;i++)
19             ranktmp[SA[i]]=rankcnt++equal(SA[i-1], SA[i], Rank);
20         rankcnt++;
21         for(int i=0;i<len;i++) Rank[i]=ranktmp[i];
22     }
23 }
24 #undef equal

```

## 6.3 AC-Automation

```

1 #define SZ 25000
2 int nx[SZ][26], spt;
3 int fl[SZ], efl[SZ], ed[SZ];
4 int newnode(){
5     for(int i=0;i<26;i++) nx[spt][i]=0;
6     ed[spt]=0;
7     return spt++;
8 }
9 int add(char *s, int sptnow){
10     for(int i=0;s[i];i++){
11         int tmp=s[i]-'a';
12         if(nx[sptnow][tmp]==0) nx[sptnow][tmp]=newnode();
13         sptnow=nx[sptnow][tmp];
14     }
15     ed[sptnow]=1;
16     return sptnow;
17 }
18 int bfsq[SZ], qs, qe;
19 void make_fl(int root){
20     fl[root]=efl[root]=qs=qe=0;
21     bfsq[qe++]=root;
22     while(qs!=qe){
23         int p=bfsq[qs++];
24         for(int i=0;i<26;i++){
25             int t=nx[p][i];
26             if(t==0) continue;
27             int tmp=fl[p];
28             for(;tmp&&nx[tmp][i]==0;tmp=fl[tmp]);
29             fl[t]=tmp?nx[tmp][i]:root;
30             efl[t]=ed[fl[t]]?fl[t]:efl[fl[t]];
31             bfsq[qe++]=t;
32         }
33     }
34 }

```

## 6.4 LCP

```

1 //build query in O(nlogn), query LCP(i,j) in O(1)
2 int dp_height[MAX_N][20];
3 void height_build(int *SA, int *Rank, char *S, int *Height){

```

```

4     int len=strlen(S), k=0;
5     for(int i=0;i<len;i++){
6         if(Rank[i]==0) continue;
7         while(S[i+k] == S[SA[Rank[i]-1]+k]) k++;
8         Height[Rank[i]]=k;
9         if(k) k--;
10    } Height[0]=0;
11    for(int i=0;i<len;i++) dp_height[i][0]=Height[i];
12    for(int i=0;i<len;i++) for(int j=1;i+(1<<j)<len;j++){
13        dp_height[i][j]=min(dp_height[i][j-1], dp_height[i+(1<<j-1)])[j-1]);
14    }
15    int height_query(int x, int y){
16        int k=0;
17        while((1<<(k+1))<=y-x) k++;
18        return min(dp_height[x+1][k], dp_height[y-(1<<k)+1][k]);
19    }

```

## 6.5 Z-value

```

1 void Z_build(const char *S, int *Z){
2     Z[0]=0;
3     int b=0;
4     for(int i=1;S[i];i++){
5         if(Z[b]+b<i) Z[i]=0;
6         else Z[i]=min(Z[b]+b-i,Z[i-b]);
7         while(S[i+Z[i]]&&S[Z[i]]==S[i+Z[i]]) Z[i]++;
8         if(Z[i]+i>Z[b]+b) b=i;
9     }
10 }

```

## 6.6 KMP

```

1 void failure_build(const char *p, int *fail){
2     for(int i=1, j=fail[0]=-1; p[i]; i++){
3         while(j>=0&&p[j+1]!=p[i]) j=fail[j];
4         if(p[j+1]==p[i]) j++;
5         fail[i]=j;
6     }
7 }
8 int KMP(const char *T, const char *P, int *fail){
9     failure_build(P, fail);
10    for(int i=0, j=-1; T[i]; i++){
11        while(j>=0&&P[j+1]!=T[i]) j=fail[j];
12        if(P[j+1]==T[i]) j++;
13        if(!P[j+1]) return i-j;
14    }
15    return -1;
16 }
17
18 //使用方法: KMP(主字串, 待匹配字串, failure array)
19 //回傳: 第一個完全匹配的位置

```

## 7 other

### 7.1 2sat

```

1 const int N = 10; // 變數數量
2 bool adj[20][20]; // adjacency matrix
3 int visit[20]; // DFS visit record
4 int sat[20]; // 解
5
6 int not(int a) {return a<N ? a+N : a-N;}
7
8 // 另外一種方式
9 /*
10 int not(int a) {return a&1 ? a : a+1;}
11 int not(int a) {return a^1;}
12 */
13
14 bool dfs_try(int i){
15     if (visit[i] == 1 || sat[i] == 1) return true;
16     if (visit[i] == 2 || sat[i] == 2) return false;
17     visit[i] = 1;
18     visit[not(i)] = 2;
19     for (int j=0; j<N+N; ++j)
20         if (adj[i][j] && !dfs_try(j))
21             return false;
22     return true;
23 }
24
25 void dfs_mark(int i){
26     if (sat[i] == 1) return;
27     sat[i] = 1;
28     sat[not(i)] = 2;
29     for (int j=0; j<N+N; ++j)
30         if (adj[i][j])
31             dfs_mark(j);
32 }
33
34 void two_satisfiability(){
35     // 一次輸入一個括號
36     memset(adj, false, sizeof(adj));
37     int a, b;
38     while (cin >> a >> b){
39         map[not(a)][b] = true;
40         map[not(b)][a] = true;
41     }
42
43     // 找出一組解
44     for (int i=0; i<N; ++i){
45         memset(visit, 0, sizeof(visit));
46         if (dfs_try(i)) {dfs_mark(i); continue;}
47
48         memset(visit, 0, sizeof(visit));
49         if (dfs_try(not(i))) {dfs_mark(not(i)); continue;}
50
51         // 無解則立即結束。
52         return;
53     }
54
55     // 印出一組解。
56     for (int i=1; i<N; ++i)
57         if (sat[i] == 1)
58             cout << i;

```

```

59     else /*if (sat[i] == 2)*/
60         cout << "not" << i;
61 }

```

## 7.2 CppHashTrick

```

1 struct custom_hash {
2     static uint64_t splitmix64(uint64_t x) {
3         x += 0x9e3779b97f4a7c15;
4         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
5         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
6         return x ^ (x >> 31);
7     }
8     // size_t operator()(YourType in) const;
9     // YourType可以放各種type/struct, 只要寫對應到x身上的方法就
10    行
11    // (std::hash, non-symmetric function)
12    size_t operator()(uint64_t in) const {
13        uint64_t x = in;
14        static const uint64_t FIXED_RANDOM = chrono::
15        steady_clock::now().time_since_epoch().count();
16        return splitmix64(x + FIXED_RANDOM);
17    };
18 unordered_map<long long, int, custom_hash> ump;
19 unordered_set<long long, custom_hash> ust;

```

## 7.3 definesss

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define pb push_back
4 #define pii pair<int,int>
5 #define pll pair<ll, ll>
6 #define pil pair<int,ll>
7 #define pli pair<ll,int>
8 #define ppi pair<pii,int>
9 #define pip pair<int,pii>
10 #define pdd pair<double, double>
11 #define f first
12 #define s second
13 #define MOD 1000000007
14 #define mkp make_pair
15 #define M_PI 3.14159265358979323846
16 #define FOR(i,l,r) for (int i=l;i<=r;i++)
17 #define LOR(i,l,r) for (ll i=l;i<=r;i++)
18 #define FORD(i,r,l) for (int i=r;i>=l;i--)
19 #define LORD(i,r,l) for (ll i=r;i>=l;i--)
20 #define INF 1000000000
21 #define CL(x) memset(x,0,sizeof(x))
22 typedef long long ll;
23
24 int main()
25 {
26     ios::sync_with_stdio(false);
27     cin.tie(0);
28
29     return 0;
30 }

```

## 7.4 PojTree

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 typedef long long int ll;
5 typedef pair<int, ll> P;
6 #define idx first
7 #define w second
8
9 const int N = 10004;
10 const ll INF = (1ll << 60);
11
12 int vn;
13 ll k;
14 vector<P> graph[N];
15 vector<int> dist;
16 ll subtreeSz[N];
17 bool isCentroid[N];
18
19 void init(){
20     for(int i = 1; i <= vn; i++)
21         graph[i].clear(), isCentroid[i] = false;
22 }
23
24 void buildTree(){
25     for(int i = 1; i < vn; i++){
26         int u, v, l; scanf("%d %d %d", &u, &v, &l);
27         graph[u].push_back(P(v, l));
28         graph[v].push_back(P(u, l));
29     }
30 }
31
32 ll calSubsz(int v, int p){
33     subtreeSz[v] = 1;
34     for(auto c:graph[v]){
35         if(isCentroid[c.idx] || c.idx == p) continue;
36         subtreeSz[v] += calSubsz(c.idx, v);
37     }
38     return subtreeSz[v];
39 }
40
41 P getCentroid(int v, int p, ll subsz){
42     P cen(-1, INF);
43     ll mxsonSz = -1;
44     for(auto c:graph[v]){
45         if(c.idx == p || isCentroid[c.idx]) continue;
46         P res = getCentroid(c.idx, v, subsz);
47         if(res.w < cen.w) cen = res;
48         mxsonSz = max(mxsonSz, subtreeSz[c.idx]);
49     }
50     mxsonSz = max(mxsonSz, subsz-subtreeSz[v]);
51     if(mxsonSz < cen.w) cen = P(v, mxsonSz);
52     return cen;
53 }
54
55 void getDist(int v, int p, ll w){
56     if(w > k) return;
57     dist.push_back(w);
58     for(auto c:graph[v]){
59         if(c.idx == p || isCentroid[c.idx]) continue;
60         getDist(c.idx, v, w+c.w);
61     }
62 }
63 }

```

```

64
65 ll calValidPair(int idx, ll w){
66     dist.clear();
67     getDist(idx, -1, w);
68     sort(dist.begin(), dist.end());
69     ll sum = 0;
70     for(int l = 0, r = dist.size()-1; l < r; ){
71         if(dist[r]+dist[l] <= k) sum += r-l, l++;
72         else r--;
73     }
74     return sum;
75 }
76
77 ll treedc(int v){
78     ll sum = 0;
79     // find centroid
80     calSubsz(v, v);
81     int cen = getCentroid(v, v, subtreeSz[v]).idx;
82     isCentroid[cen] = true;
83
84     sum += calValidPair(cen, 0);
85     for(auto c:graph[cen])
86     {
87         if(isCentroid[c.idx]) continue;
88         sum += calValidPair(c.idx, c.w);
89         sum += treedc(c.idx);
90     }
91     return sum;
92 }
93
94 int main(){
95     while(scanf("%d %lld", &vn, &k) && vn && k)
96     {
97         init();
98         buildTree();
99         printf("%lld\n", treedc(1));
100     }
101     return 0;
102 }

```

4.13	dijkstra . . . . .	13
4.14	MaxWeightPerfectBiMatch . . . . .	13
<b>5</b>	<b>Math</b>	<b>13</b>
5.1	extgcd . . . . .	13
5.2	NTT . . . . .	13
5.3	GaussianJordan . . . . .	13
5.4	EulerPhi . . . . .	14
5.5	FFT . . . . .	14
5.6	BigInt . . . . .	14
5.7	mobius . . . . .	15
5.8	modeq . . . . .	15
<b>6</b>	<b>String</b>	<b>15</b>
6.1	BWT . . . . .	15
6.2	SuffixArray . . . . .	15
6.3	AC-Automation . . . . .	16
6.4	LCP . . . . .	16
6.5	Z-value . . . . .	16
6.6	KMP . . . . .	16
<b>7</b>	<b>other</b>	<b>16</b>
7.1	2sat . . . . .	16
7.2	CppHashTrick . . . . .	17
7.3	definss . . . . .	17
7.4	PojTree . . . . .	17