

# 1 DataStructure

## 1.1 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>0;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    }

```

## 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 {
16     memset(st, 0, sizeof(st));
17 }
18
19 void St1d::modify(int l, int r, int idx, int L, int R, ll v)
20 {
21     if(r < L || R < l) return;

```

```

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

```

```

85 ll St2d::query(int il, int ir, int jl, int jr, int idx, int i
86     , int j)
87 {
88     ll tot = 0;
89     if(i < il || ir < i) return 0;
90     if(il <= i && i <= ir) tot += st[idx].query(jl, jr, 1, j)
91         ;
92     if(il == i && ir == i) return tot;
93     int mid = (il+ir)/2;
94     tot += query(il, mid, jl, jr, idx*2, i, j);
95     tot += query(mid+1, ir, jl, jr, idx*2+1, i, j);
96     return tot;
97 }

```

## 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 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 }

```

## 1.5 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 {
9     return (st[idx].data*st[idx].mul%MD+(r-l+1)*st[idx].add%
10         MD)%MD;
11 }
12 void up(int l, int r, int idx)

```

```

13 {
14     int mid = l+(r-l)/2;
15     st[idx].data = (getval(l, mid, idx*2)+getval(mid+1, r,
16         idx*2+1))%MD;
17 }
18 void down(int l, int r, int idx)
19 {
20     st[idx].data = getval(l, r, idx);
21     int lson = idx*2, rson = idx*2+1;
22     if(l != r)
23     {
24         st[lson].mul = st[lson].mul*st[idx].mul%MD;
25         st[lson].add = (st[lson].add*st[idx].mul+st[idx].add)
26             %MD;
27         st[rson].mul = st[rson].mul*st[idx].mul%MD;
28         st[rson].add = (st[rson].add*st[idx].mul+st[idx].add)
29             %MD;
30     }
31     st[idx].mul = 1, st[idx].add = 0;
32 }
33 void buildst(int l, int r, int idx)
34 {
35     st[idx].mul = 1, st[idx].add = 0;
36     if(l == r)
37     {
38         st[idx].data = arr[l];
39         return;
40     }
41     int mid = l+(r-l)/2;
42     buildst(l, mid, idx*2);
43     buildst(mid+1, r, idx*2+1);
44     up(l, r, idx);
45 }
46 void add(int l, int r, int idx, int L, int R, int v) //操作L,
47     R
48 {
49     if(r < L || R < l) return;
50     if(L <= l && r <= R)
51     {
52         st[idx].add = (st[idx].add+v)%MD;
53         return;
54     }
55     down(l, r, idx);
56     int mid = l+(r-l)/2;
57     add(l, mid, idx*2, L, R, v);
58     add(mid+1, r, idx*2+1, L, R, v);
59     up(l, r, idx);
60 }
61 void mul(int l, int r, int idx, int L, int R, int v)
62 {
63     if(r < L || R < l) return;
64     if(L <= l && r <= R)
65     {
66         st[idx].add = st[idx].add*v%MD;
67         st[idx].mul = st[idx].mul*v%MD;
68         return;
69     }
70     down(l, r, idx);
71     int mid = l+(r-l)/2;
72     mul(l, mid, idx*2, L, R, v);
73     mul(mid+1, r, idx*2+1, L, R, v);

```

```

74     up(l, r, idx);
75 }
76
77 ll query(int l, int r, int idx, int L, int R)
78 {
79     if(r < L || R < l) return 0;
80     if(L <= l && r <= R)
81     {
82         return getval(l, r, idx);
83     }
84     down(l, r, idx);
85     int mid = l+(r-l)/2;
86     return (query(l, mid, idx*2, L, R)+query(mid+1, r, idx
87         *2+1, L, R))%MD;

```

## 1.6 Matrix

```

1 ll SZ,MOD;
2 const int MAXSZ=105;
3
4 struct Mat
5 {
6     ll m[MAXSZ][MAXSZ];
7     Mat(){memset(m, 0, sizeof(m));}
8 };
9
10 Mat matMul(const Mat &A, const Mat &B)
11 {
12     Mat rtn;
13     for(int i = 0; i < SZ; i++)
14         for(int k = 0; k < SZ; k++)
15             if(A.m[i][k])for(int j = 0; j < SZ; j++)
16                 {
17                     rtn.m[i][j]+=(A.m[i][k]*B.m[k][j]);
18                 }
19     return rtn;
20 }
21 //B is of size SZ
22 vector<ll> matMul(const Mat &A, const vector<ll> &B)
23 {
24     vector<ll> rtn(SZ,0);
25     for(int i = 0; i < SZ; i++)
26         for(int j = 0; j < SZ; j++)
27             rtn[i]=(rtn[i]+A.m[i][j]*B[j]);
28
29     return rtn;
30 }
31
32 Mat matPow(Mat& M, ll p)
33 {
34     if(p == 0)
35     {
36         Mat iden;
37         for(int i=0;i<SZ;i++)iden.m[i][i]=1;
38         return iden;
39     }
40     if(p == 1)return M;
41     Mat rtn = matPow(M, p/2);
42     if(p&1)return matMul(matMul(rtn, rtn), M);
43     else return matMul(rtn, rtn);
44 }

```

## 1.7 treap

```

1 struct Treap
2 {
3     int pri, sz;
4     int rev;
5     ll data, sum; // tag: make-same
6     Treap *lchild, *rchild;
7     Treap(ll d):pri(rand()), sz(1), rev(0), data(d), sum(d),
8         lchild(NULL), rchild(NULL)
9     {
10     }
11     inline void up();
12     inline void down();
13 };
14 inline int size(Treap *t) { return t? t->sz:0; }
15 inline ll get_data(Treap *t) { return t? t->data:0; }
16 inline ll get_sum(Treap *t) { return t? t->sum:0; }
17
18 inline void Treap::up()
19 {
20     if(lchild) lchild->down();
21     if(rchild) rchild->down();
22     sz = 1+size(lchild)+size(rchild);
23     sum = get_sum(lchild) + data + get_sum(rchild);
24 }
25
26 inline void Treap::down()
27 {
28     if(rev)
29     {
30         swap(mxpre, mxpost);
31         swap(lchild, rchild);
32         if(lchild) lchild->rev ^= 1;
33         if(rchild) rchild->rev ^= 1;
34         rev ^= 1;
35     }
36 }
37
38 Treap *merge(Treap *a, Treap *b)
39 {
40     if(!a || !b) return (a? a:b);
41     if(a->pri < b->pri)
42     {
43         a->down();
44         a->rchild = merge(a->rchild, b);
45         a->up();
46         return a;
47     }
48     else
49     {
50         b->down();
51         b->lchild = merge(a, b->lchild);
52         b->up();
53         return b;
54     }
55 }
56
57 void split(Treap *o, Treap *&a, Treap *&b, int k)
58 {
59     if(!o) a = b = NULL;
60     else
61     {
62         o->down();

```

```

63         if(k >= size(o->lchild)+1)
64         {
65             a = o;
66             split(o->rchild, a->rchild, b, k-size(o->lchild)-1);
67         }
68         else
69         {
70             b = o;
71             split(o->lchild, a, b->lchild, k);
72         }
73         o->up();
74     }
75 }

```

## 1.8 1d\_segTree

```

1 void buildst(int l, int r, int idx) //l, r是st的區間
2 {
3     if(l == r)
4     {
5         st[idx] = arr[l];
6         return;
7     }
8     int mid = (l+r)/2;
9     buildst(l, mid, idx*2);
10    buildst(mid+1, r, idx*2+1);
11    st[idx] = max(st[idx*2], st[idx*2+1]);
12 }
13
14 ll query(int l, int r, int idx, int L, int R) //L,R是操作的區間
15 {
16     if(r < L || R < l) return -INF;
17     if(L <= l && r <= R) return st[idx];
18     int mid = (l+r)/2;
19     return max(query(l, mid, idx*2, L, R), query(mid+1, r, 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     {
27         st[idx] += v; return;
28     }
29     int mid = (l+r)/2;
30     modify(l, mid, idx*2, x, v);
31     modify(mid+1, r, idx*2+1, x, v);
32     st[idx] = max(st[idx*2], st[idx*2+1]);
33 }

```

## 2 Flow

### 2.1 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[i].flow);
51                 flow+=f;
52                 e[i].flow-=f;
53                 e[i^1].flow+=f;
54                 if(minflow-flow<=1e-8)return flow;
55                 if(dis[start]>=ans)return flow;
56             }
57         }
58     }
59     if(--gap[dis[u]]==0)
60         dis[start]=ans;
61     dis[u]++;
62     gap[dis[u]]++;
63     return flow;
64 }
65 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%d",&n,&m)!=-1)
90     {
91         if(m==0){printf("1\n1\n");continue;}
92         start=0,end=n+m+1,ans=end+1;
93         for(i=0;i<m;i++)
94         {
95             scanf("%d%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
          之间误差的精度不超过1/(n*n)
99     {
100         mid=(Left+Right)/2;
101         makemap(mid);
102         flow=isap();//求出最大权值闭合图
103         if(flow<eps)//如果小于0, g值太大
104             Right=mid;
105         else Left=mid;
106     }
107     makemap(Left);//最大密度建图
108     isap();
109     memset(vis,false,sizeof(vis));
110     sum=0;
111     dfs1(start);
112     printf("%d\n",sum);
113     for(i=1;i<=n;i++)
114         if(vis[i]==true)//残留网络中源点能到达的点
115             printf("%d\n",i);
116 }
117 return 0;
118 }

```

## 2.2 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),
            r(cap){}
10    };
11    int g[MAXN];
12    vector<edge> e;
13    void init(int _n){
14        memset(g,-1,sizeof(int)*((n=_n)+1));
15        e.clear();
16    }
17    void add_edge(int u,int v,T cap,bool directed=false){
18        e.push_back(edge(v,g[u],cap));
19        g[u]=e.size()-1;
20        e.push_back(edge(u,g[v],directed?0:cap));
21        g[v]=e.size()-1;
22    }
23    int bfs(int s,int t){
24        memset(level,0,sizeof(int)*(n+1));
25        memcpy(cur,g,sizeof(int)*(n+1));
26        queue<int> q;
27        q.push(s);
28        level[s]=1;
29        while(q.size()){
30            int u=q.front();q.pop();
31            for(int i=g[u];~i;i=e[i].pre){
32                if(!level[e[i].v]&&e[i].r){
33                    level[e[i].v]=level[u]+1;
34                    q.push(e[i].v);
35                    if(e[i].v==t)return 1;
36                }
37            }
38        }
39        return 0;
40    }
41    T dfs(int u,int t,T cur_flow=INF){
42        if(u==t)return cur_flow;
43        T df;
44        for(int &i=cur[u];~i;i=e[i].pre){
45            if(level[e[i].v]==level[u]+1&&e[i].r){
46                if(df=dfs(e[i].v,t,min(cur_flow,e[i].r))){
47                    e[i].flow+=df;
48                    e[i^1].flow-=df;
49                    e[i].r-=df;
50                    e[i^1].r+=df;
51                    return df;
52                }
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 };

```

## 2.3 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)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;

```

```

62   PIS=ans=0;
63   while(modlabel()){
64       do memset(vis,0,sizeof(bool)*(n+1));
65       while(augment(S,INF));
66   }return ans;
67 }
68 };

```

## 3 Geometry

### 3.1 intercircle

```

1 vector<Point> interCir(Point o1, double r1, Point o2, double
  r2)
2 {
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
13    Point v_up=(Point){v.x*c-v.y*s,v.x*s+v.y*c};
14    Point v_down=(Point){v.x*c+v.y*s,-v.x*s+v.y*c};
15    return {o1+v_up,o1+v_down};
16 } // 求兩圓交點

```

### 3.2 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 } // 向量長度的平方
33 double rad(const Point b)const{
34     return fabs(atan2(fabs(cross(b)),dot(b)));
35 } // 兩向量的弧度
36 };

```

### 3.3 SegmentGeometry

```

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

```

```

45     {
46         p[0]=_p1;
47         p[1]=_p2;
48         if(p[1]<p[0])swap(p[0],p[1]);
49     }
50     P p[2];
51 };
52
53 bool par(seg& lhs, seg& rhs)
54 {return par((lhs.p[0],lhs.p[1],rhs.p[0],rhs.p[1]));}
55
56 P intersection(seg& lhs, seg& rhs)//p and q Must not be
  parallel
57 {return intersection((lhs.p[0],lhs.p[1],rhs.p[0],rhs.p[1]));}
58
59 bool on_seg(seg& sg, P q)
60 {return on_seg(sg.p[0],sg.p[1],q);}
61
62 bool overlap(seg s1, seg s2)
63 {
64     return par(s1,s2)&&
65     ( on_seg(s1,s2.p[0])||on_seg(s1,s2.p[1])||
66     on_seg(s2,s1.p[0])||on_seg(s2,s1.p[1]) );
67 }
68
69 bool is_intersect(seg s1, seg s2)
70 {
71     if(par(s1,s2))return false;
72     P p0 = intersection(s1,s2);
73     return on_seg(s1,p0)&&on_seg(s2,p0);
74 }
75
76 //make sure the vec is not vertical
77 double interpolate(seg& vec, double X)
78 {
79     double y0=vec.p[0].y,y1=vec.p[1].y,
80     x0=vec.p[0].x,x1=vec.p[1].x;
81     return y0+(y1-y0)*(X-x0)/(x1-x0);
82 }
83
84 //pts in clockwise order, p[N]=p[0]
85 bool in_poly(P* pol,int N,P pt)
86 {
87     double X = pt.x,Y=pt.y;
88     int pas=0;
89     for(int i=0;i<N;i++)
90     {
91         if(pol[i].x==pol[i+1].x)continue;
92         seg s0(pol[i],pol[i+1]);
93         //up or down?
94         double Y1 = interpolate(s0,X);
95         if(Y1<Y-EPS)continue;
96         double xl=min(pol[i].x,pol[i+1].x),xr=max(pol[i].x,
97         pol[i+1].x);
98         if(xl<X-EPS&&xr>=X-EPS)pas++;
99     }
100     return pas&1;
101 }
102
103 double dpseg(P p, P p1, P p2)//p to p1p2, p1!=p2
104 {
105     P v=p2-p1, v1=p-p1, v2=p-p2;
106     if( v.dot(v1) < EPS )return dis(p,p1);
107     if( v.dot(v2) > EPS )return dis(p,p2);
108     return fabs((p-p1).det(v))/len(v);
109 }

```

```

109 double dpseg(P p, seg s1)
110 {
111     return dpseg(p,s1.p[0],s1.p[1]);
112 }
113
114 double dsegseg(P p1, P p2, P p3, P p4)
115 {
116     if( is_intersect( seg(p1,p2), seg(p3,p4) ) )return 0;
117     return min( min( dpseg(p1,p3,p4),dpseg(p2,p3,p4) ), min(
118         dpseg(p3,p1,p2),dpseg(p4,p1,p2) ) );
119 }
120
121 double dsegseg(seg s1, seg s2)
122 {
123     return dsegseg( s1.p[0],s1.p[1],s2.p[0],s2.p[1] );
124 }

```

### 3.4 nearestDist

```

1 bool cmp_y(P a, P b)
2 {
3     return a.y < b.y;
4 }
5
6 bool cmp_x(P a, P b)
7 {
8     return a.x < b.x;
9 }
10
11 double dc(P *arr, int n)
12 {
13     if(n == 1) return INF;
14     int mid = n/2;
15     double cx = arr[mid].x;
16     double dist = min( dc(arr, mid), dc(arr+mid, n-mid) );
17     inplace_merge(arr, arr+mid, arr+n, cmp_y);
18     static vector<P> brr; brr.clear();
19     for(int i = 0; i < n; i++)
20     {
21         if(fabs(arr[i].x)-cx >= dist) continue;
22         for(int j = brr.size()-1; j >= 0; j--)
23         {
24             double dx = brr[j].x-arr[i].x;
25             double dy = brr[j].y-arr[i].y;
26             if(fabs(dy) >= dist) break;
27             dist = min(dist, sqrt(dx*dx+dy*dy));
28         }
29         brr.push_back(arr[i]);
30     }
31     return dist;
32 }
33
34 double nearestDist(P *arr, int n)
35 {
36     sort(arr, arr+n, cmp_x);
37     return dc(arr, n);
38 }

```

## 4 Graph

### 4.1 StronglyConnectedComponent

```

1 int V, E;
2 vector<int> G[MAXV];
3 vector<int> rG[MAXV];
4 vector<int> vs;//postorder
5 bool used[MAXV];
6 int comp[MAXV];//scc id, topologically ordered
7
8 void add_edge(int from, int to)
9 {
10     G[from].pb(to);
11     rG[to].pb(from);
12 }
13
14 void dfs(int u)//get postorder
15 {
16     used[u] = true;
17     for(int i=0;i<G[u].size();i++)
18         if(!used[G[u][i]])dfs(G[u][i]);
19     vs.pb(u);
20 }
21
22 void rdfs(int u, int k)
23 {
24     used[u]=true;
25     comp[u]=k;
26     for(int i=0;i<rG[u].size();i++)
27         if(!used[rG[u][i]])rdfs(rG[u][i],k);
28 }
29
30 int scc();//return scc cnt
31 {
32     memset(used,0,sizeof(used));
33     vs.clear();
34     FOR(v,1,V)if(!used[v])dfs(v);
35     memset(used,0,sizeof(used));
36     int k = 0;//sccID
37     FORD(i,V-1,0)
38         if(!used[ vs[i] ])
39         {
40             rdfs(vs[i],k);
41             k++;
42         }
43     return k;
44 }

```

### 4.2 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 }

```

### 4.3 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     {
16         V = _V;
17         FZ(el);
18         FZ(pr);
19         FZ(inq);
20         FZ(inp);
21         FZ(inb);
22         FZ(bk);
23         FZ(djs);
24         ans = 0;
25     }
26     void add_edge(int u, int v)
27     {
28         el[u][v] = el[v][u] = 1;
29     }
30     int lca(int u,int v)
31     {
32         memset(inp,0,sizeof(inp));
33         while(1)
34         {
35             u = djs[u];

```

```

36     inp[u] = true;
37     if(u == st) break;
38     u = bk[pr[u]];
39 }
40 while(1)
41 {
42     v = djs[v];
43     if(inp[v]) return v;
44     v = bk[pr[v]];
45 }
46 return v;
47 }
48 void upd(int u)
49 {
50     int v;
51     while(djs[u] != nb)
52     {
53         v = pr[u];
54         inb[djs[u]] = inb[djs[v]] = true;
55         u = bk[v];
56         if(djs[u] != nb) bk[u] = v;
57     }
58 }
59 void blo(int u, int v)
60 {
61     nb = lca(u, v);
62     memset(inb, 0, sizeof(inb));
63     upd(u);
64     upd(v);
65     if(djs[u] != nb) bk[u] = v;
66     if(djs[v] != nb) bk[v] = u;
67     for(int tu = 1; tu <= V; tu++)
68         if(inb[djs[tu]])
69         {
70             djs[tu] = nb;
71             if(!inq[tu])
72             {
73                 qe.push(tu);
74                 inq[tu] = 1;
75             }
76         }
77 }
78 void flow()
79 {
80     memset(inq, false, sizeof(inq));
81     memset(bk, 0, sizeof(bk));
82     for(int i = 1; i <= V; i++)
83         djs[i] = i;
84
85     while(qe.size()) qe.pop();
86     qe.push(st);
87     inq[st] = 1;
88     ed = 0;
89     while(qe.size())
90     {
91         int u = qe.front();
92         qe.pop();
93         for(int v = 1; v <= V; v++)
94             if(el[u][v] && (djs[u] != djs[v]) && (pr[u] != v))
95             {
96                 if((v == st) || ((pr[v] > 0) && bk[pr[v]] > 0))
97                     blo(u, v);
98                 else if(bk[v] == 0)
99                 {

```

```

100         bk[v] = u;
101         if(pr[v] > 0)
102         {
103             if(!inq[pr[v]]) qe.push(pr[v]);
104         }
105         else
106         {
107             ed = v;
108             return;
109         }
110     }
111 }
112 }
113 }
114 void aug()
115 {
116     int u, v, w;
117     u = ed;
118     while(u > 0)
119     {
120         v = bk[u];
121         w = pr[v];
122         pr[v] = u;
123         pr[u] = v;
124         u = w;
125     }
126 }
127 int solve()
128 {
129     memset(pr, 0, sizeof(pr));
130     for(int u = 1; u <= V; u++)
131         if(pr[u] == 0)
132         {
133             st = u;
134             flow();
135             if(ed > 0)
136             {
137                 aug();
138                 ans++;
139             }
140         }
141     return ans;
142 }
143 } gm;

```

## 4.4 ArticulationPoint

```

1 vector<int> G[MAXN];
2 int dfn[MAXN], low[MAXN], tim, ans;
3 void tarjan(int u, int p){ // p -> u
4     int child=0, cut_node=0;
5     dfn[u] = low[u] = ++tim;
6     for(auto v:G[u]){ // u -> v
7         if(!dfn[v]){ // tree edge
8             tarjan(v, u); child++;
9             low[u] = min(low[u], low[v]);
10            if(low[v] >= dfn[u]) cut_node=1;
11        }
12        else if(v!=p) //back edge
13            low[u] = min(low[u], dfn[v]);
14    }
15    if(p!=-1 && cut_node) ans++;
16 }

```

```

17     if(p== -1 && child>=2) ans++;
18 }
19 //how to call
20 for(int i=0; i<n; i++)
21     if(!dnf[i])
22         tarjan(i, -1);

```

## 4.5 TwoConnected

```

1 vector<int> G[MAXN];
2 vector<int> bcc[MAXN]; // bcc內的點
3
4 int dfn[MAXN], low[MAXN], tim, ans;
5 int st[MAXN], top;
6 int bccID[MAXN], bcc_cnt; // 每個點的bcc編號
7 bool is_cut[MAXN]; // 是否為割點, 割點的ID會被覆蓋
8
9 void tarjan(int u, int p){ // p->u
10     int child= 0, w;
11     dfn[u] = low[u] = ++tim;
12     st[top++] = u;
13     for(auto v:G[u]){ // u->v
14         if(!dfn[v]){
15             tarjan(v, u); child++;
16             low[u] = min(low[u], low[v]);
17             if(low[v] >= dfn[u]){
18                 is_cut[u] = true;
19                 do{
20                     w = st[--top];
21                     bccID[w] = bcc_cnt;
22                     bcc[bcc_cnt].push_back(w);
23                 }while(dfn[w] > dfn[v]);
24                 bccID[u] = bcc_cnt;
25                 bcc[bcc_cnt++].push_back(u);
26             }
27         }
28         else if(v != p)
29             low[u] = min(low[u], dfn[v]);
30     }
31     if(p == -1 && child<2)
32         is_cut[u] = false;
33 }
34 }

```

## 4.6 TarjanBridge

```

1 void tarjan(int u, int p){ // p -> u
2     dfn[u] = low[u] = ++tim;
3
4     for(auto v: G[u]){ // u->v
5         if(!dfn[v]){
6             tarjan(v, u);
7             low[u] = min(low[u], low[v]);
8             if(low[v] > dfn[u]){
9                 ans++;
10                E.push_back(edge(u, v));
11            }
12        }
13        else if(v != p)
14            low[u] = min(low[u], dfn[v]);
15    }

```



```

14         low[u] = min(low[u], dfn[v]);
15     }
16 }
17 struct edge{
18     int from, to;
19     edge(int u, int v):
20         from(u), to(v){}
21 };
22 for(int i=0; i<n; i++)
23     if(!dnf[i])
24         tarjan(i, -1);

```

## 4.7 MaxWeightPerfectMatch

```

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

```

```

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

```

## 4.8 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];
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 }

```

## 4.9 MaximalClique

```

1 #define MAXN 32
2 int n, m, Max;
3 ll v[MAXN], deg[MAXN]; //neighbors
4
5 void update_maximum(ll R)

```

```

6 {
7     int Size = 0;
8     while(R)
9     {
10         if(R&1)Size++;
11         R>>=1;
12     }
13     Max = max(Size, Max);
14 }
15
16 int pickPivot(ll P)
17 {
18     int pivot = -1, Max = -1;
19     memset(deg, 0, sizeof(deg));
20     for(int i = 0; i < n; i++)
21     {
22         if(P&(1LL<<i)) //i is in P
23         {
24             if(pivot == -1) //i = default pivot
25             {
26                 pivot = i;
27                 Max = deg[i];
28             }
29             for(int j = 0; j < i; j++)
30             {
31                 if((P&(1LL<<j))&&(v[i]&(1LL<<j)))
32                 {
33                     deg[i]++;
34                     if(deg[i] > Max)
35                     {
36                         Max = deg[i];
37                         pivot = i;
38                     }
39                     deg[j]++;
40                     if(deg[j] > Max)
41                     {
42                         Max = deg[j];
43                         pivot = j;
44                     }
45                 }
46             }
47         }
48     }
49     return pivot;
50 }
51
52 void BronKerbosch(ll R, ll P, ll X)
53 {
54     if(!P) //P is empty, no candidates left
55     {
56         if(!X)
57         {
58             //clique
59             update_maximum(R);
60         }
61         return;
62     }
63     int u = pickPivot(P|X);
64     for(int i = 0; i <= n-1; i++)
65     {
66         if(P&(~v[u])&(1LL<<i)) //vi is in P
67         {
68             BronKerbosch( R|(1LL<<i), P&v[i], X&v[i] );
69             P&=~(1LL<<i);
70             X|=(1LL<<i);
71         }

```



```

72     }
73 }
74
75 int main()
76 {
77     ios::sync_with_stdio(false);
78     cin.tie(0);
79     while(cin >> n)
80     {
81         cin >> m;
82
83         Max = 0;
84         FOR(i,0,n-1)v[i] = 0;
85
86         int a, b;
87         FOR(i,1,m)
88         {
89             cin >> a >> b;
90             v[a]|=(1LL<<b);
91             v[b]|=(1LL<<a);
92         }
93         BronKerbosch(0, (1LL<<n)-1, 0);
94         cout << Max << '\n';
95     }
96     return 0;
97 }

```

## 4.10 MinimumMeanCycle

```

1 #include<cstdio> //for DBL_MAX
2 int dp[MAXN][MAXN]; // 1-base,0(NM)
3 vector<tuple<int,int,int>> edge;
4 double mmc(int n){//allow negative weight
5     const int INF=0x3f3f3f3f;
6     for(int t=0;t<n;++t){
7         memset(dp[t+1],0x3f,sizeof(dp[t+1]));
8         for(const auto &e:edge){
9             int u,v,w;
10            tie(u,v,w) = e;
11            dp[t+1][v]=min(dp[t+1][v],dp[t][u]+w);
12        }
13    }
14    double res = DBL_MAX;
15    for(int u=1;u<=n;++u){
16        if(dp[n][u]==INF) continue;
17        double val = -DBL_MAX;
18        for(int t=0;t<n;++t){
19            val=max(val,(dp[n][u]-dp[t][u])*1.0/(n-t));
20        }
21        res=min(res,val);
22    }
23    return res;

```

## 4.11 匹配問題轉換

### 4.11.1 一般圖匹配問題轉換

1. 最大匹配邊數  $|+|$  最小邊涵蓋  $|=|V|$  (無孤立點)
2. 最大獨立集  $|+|$  最小點涵蓋  $|=|V|$
3. 最大權匹配  $\rightarrow$  最大權完美匹配: 用 0 邊補成完全圖

4. 最大權最大匹配  $\rightarrow$  最大權匹配: 先把所有邊加上  $|$  最負邊權重  $|+1$ , 得到新的圖  $G'$  上沒有任何負邊, 然後所有邊再加上  $G'$  上所有邊權重和, 這樣最大權匹配就會 = 最大權最大匹配。

## 4.12 BridgeConnected

```

1 struct edge{
2     int from, to;
3     edge(int u, int v):from(u), to(v){}
4 };
5 vector<edge> bridge;
6 vector<int> G[MAXN];
7 int dfn[MAXN], low[MAXN], tim;
8 int bccID[MAXN], bccCNT;
9 int st[MAXN], top;
10
11 void tarjan(int u, int p){ // p->u
12     dfn[u]= low[u]= ++tim;
13     st[top++]=u;
14     for(auto v:G[u]){ // u->v
15         if(!dfn[v]){
16             tarjan(v, u);
17             low[u]= min(low[u], low[v]);
18             if(low[v] > dfn[u]){
19                 bridge.push_back(edge(u, v));
20             }
21         } else if(v != p)
22             low[u]= min(low[u], dfn[v]);
23     }
24
25     if(dfn[u] == low[u]){
26         int w;
27         do{
28             w = st[--top];
29             bccID[w]= bccCNT;
30         }while(w != u);
31         bccCNT++;
32     }
33 }

```

## 4.13 HeavyLightDecomposition

```

1 const int MAX_N;
2 vector<int> link[MAX_N]; //edge
3
4 void dfs_build(int now, int fa, int *weight, int *depth, int *pa, int *son)
5 {
6     weight[now]=1;
7     son[now]=-1;
8     pa[now]=fa;
9     for(auto i:link[now])
10     {
11         if(i==fa) continue;
12         depth[i]=depth[now]+1;
13         dfs_build(i,now,weight,depth,pa,son);
14         if(son[now]==-1||weight[son[now]]<weight[i]) son[now]=i;
15         weight[now]+=weight[i];
16     }
17 }

```

```

18 void build_top(int now, int top,int *pa, int *son, int *link_top)
19 {
20     link_top[now]=top;
21     if(son[now]==-1) return;
22     build_top(son[now],top,pa,son,link_top);
23     for(auto i:link[now])
24     {
25         if(i==son[now]||i==pa[now]) continue;
26         build_top(i,i,pa,son,link_top);
27     }
28 }
29 inline void HLD(int *weight, int *depth, int *pa, int *son, int *link_top)
30 {
31     memset(son,-1,sizeof(int)*MAX_N);
32     depth[1]=1; //set node(1) as root
33     dfs_build(1,0,weight,depth,pa,son);
34     build_top(1,1,pa,son,link_top);
35 }
36 inline int find_lca(int x, int y, int *depth, int *pa, int *link_top)
37 {
38     int tx=link_top[x], ty=link_top[y];
39     while(tx!=ty)
40     {
41         if(depth[tx]<depth[ty])
42         {
43             swap(tx,ty);
44             swap(x,y);
45         }
46         tx=link_top[x=pa[x]];
47     }
48     return depth[x]<depth[y]?x:y;
49 }
50 //usage:
51 //build HeavyLightDecomposition: HLD
52 //find LCA(x,y): find_lca

```

## 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 }

```

```

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     for(int i = 0; i < n; ++i) {
43         fill(slack, slack + n, INF);
44         while(true) {
45             for(int j = 0; j < n; ++j) S[j] = T[j] = false;
46             if(match(i)) break;
47             else update();
48         }
49     }
50 }

```

## 4.15 tarjan

```

1 // copied from sylveon
2 // ap
3 void DFS(int v, int fa) //call DFS(v,v) at first
4 {
5     D[v] = L[v] = timestamp++; //timestamp > 0
6     int childCount = 0; //定理1 for root
7     bool isAP = false;
8
9     for (int w:adj[v])
10     {
11         if( w==fa ) continue;
12         if ( !D[w] ) // D[w] = 0 if not visited
13         {
14             DFS(w,v);
15             childCount++;
16             if (D[v]<=L[w]) isAP = true; //定理2
17             if (D[v]< L[w]) edgeBridge.emplace_back(v, w); //定理3
18             L[v] = min(L[v], L[w]);
19         }
20         L[v] = min(L[v], D[w]);
21     }
22     if ( v == fa && childCount < 2 ) isAP = false; //定理1, v == fa只是確認root
23     if ( isAP ) nodeAP.push_back(v);
24     return ;
25 }
26 // SCC
27 void DFS(int v, int fa) { //call DFS(v,v) at first
28     D[v] = L[v] = timestamp++; //timestamp > 0

```

```

29     st.push(v);
30
31     for (int w:adj[v]) {
32         if( w==fa ) continue;
33         if ( !D[w] ) { // D[w] = 0 if not visited
34             DFS(w,v);
35             L[v] = min(L[v], L[w]);
36         }
37         L[v] = min(L[v], D[w]);
38     }
39     if( D[v] == L[v] ) {
40         int x;
41         do{
42             x = st.top();
43             st.pop();
44             scc[x] = SCCID;
45         }while( x != v );
46         SCCID++;
47     }
48 }

```

## 4.16 dijkstra

```

1
2 struct edge{int to, cost;};
3 typedef pair<int, int> P; //first = min distance, second = v
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 }

```

## 4.17 spfa

```

1 typedef pair<int, ll> P;
2 #define idx first
3 #define w second
4 int vn, en;
5 vector<P> graph[N];
6 ll dist[N];
7
8 bool spfa() // return true if neg cycle
9 {
10     for(int i = 0; i < vn; i++) dist[i] = INF; dist[0] = 0;
11     int cnt[N] = {0};
12     bool inq[N] = {false};
13     queue<int> q; q.push(0); inq[0] = true;
14     while(!q.empty())
15     {
16         int s = q.front(); q.pop();
17         inq[s] = false;
18         for(auto e:graph[s])
19         {
20             if(dist[e.idx] > dist[s]+e.w)
21             {
22                 dist[e.idx] = dist[s]+e.w;
23                 if(++cnt[e.idx] >= vn) return true;
24                 if(!inq[e.idx])
25                 {
26                     inq[e.idx] = true;
27                     q.push(e.idx);
28                 }
29             }
30         }
31     }
32     return false;
33 }

```

## 4.18 MaxBiMatching

```

1 //注意：變數V
2 #define MAXV 505
3 int V; // # of vertex
4 vector<int> G[MAXV];
5 int match[MAXV];
6 int used[MAXV];
7
8 void add_edge(int u, int v)
9 {
10     G[u].pb(v);
11     G[v].pb(u);
12 }
13
14 bool dfs(int u)
15 {
16     used[u]=true;
17     for(int i = 0; i < G[u].size(); i++)
18     {
19         int v = G[u][i], w = match[v];
20         if(w<0 || !used[w]&&dfs(w))
21         {
22             match[u]=v;
23             match[v]=u;
24             return true;
25         }
26     }
27     return false;

```

```

28 }
29
30 int bip_match()
31 {
32     int res=0;
33     memset(match,-1,sizeof(match));
34     for(int v=0; v<V; v++)
35     {
36         if(match[v]<0)
37         {
38             memset(used,0,sizeof(used));
39             if(dfs(v))res++;
40         }
41     }
42     return res;
43 }

```

## 5 Math

### 5.1 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.

```

### 5.2 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++)

```

```

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

```

### 5.3 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 {
7     ll rtn = 1;
8     for(x=(x>=m?x%m:x);y>=1)
9     {
10        if(y&1) rtn = rtn*x%m;
11        x = x*x%m;
12    }
13    return rtn;
14 }
15 inline vector<ll> ntt(vector<ll> rtn, int Rev = 1)
16 {
17     int ntt_n = rtn.size();
18     for(int i=0,j=0;i<ntt_n;i++)
19     {
20         if(i>j) swap(rtn[i],rtn[j]);
21         for(int k=(ntt_n>>1);(j^=k)<k;k>=1);
22     }
23     for(int i=2,m=1;i<=ntt_n;i<=1,m++)
24     {
25         ll w = 1, wn = fpw(G,(P-1)>>m,P), u, t;
26         int mh = i>>1;
27         for(int j=0;j<mh;j++)
28         {
29             for(int k=j;k<ntt_n;k+=i)
30             {
31                 u = rtn[k], t = w*rtn[k+mh]%P;
32                 rtn[k] = (u+t)%P;
33                 rtn[k+mh] = (u-t+P)%P;

```

```

34     }
35     w = w*wn%P;
36 }
37 }
38 if(!~Rev)
39 {
40     for(int i=1;i<ntt_n/2;i++) swap(rtn[i],rtn[ntt_n-i]);
41     ll Revn = fpw(ntt_n,P-2,P);
42     for(int i=0;i<ntt_n;i++) rtn[i] = rtn[i]*Revn%P;
43 }
44 return rtn;
45 }
46 // 把原多項式包成long long的vector(poly), 並把項次拓展到2^i.
47 // 用ntt(poly)即可得到轉換後的結果.
48 // Rev為1時為NTT, 為-1時為InvNTT.

```

### 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 BigInt

```

1 #define MAX_N 1000
2 #define MAX 100000
3 #define MAX_LOG 5

```

```

4 class BigInt
5 {
6 public:
7     int sign;
8     long long m[MAX_N];
9     int l;
10    long long operator [](int i) const { return m[i]; }
11    long long &operator [](int i) { return m[i]; }
12    BigInt(){ l=1, m[0]=0; sign=1; }
13    BigInt(int x){ (*this)=x; }
14    BigInt(const char *s){ (*this)=s; }
15    BigInt operator =(int x)
16    {
17        if(x<0) x=-x, sign=-1;
18        else sign=1;
19        for(l=1, m[l-1]=x%MAX, x/=MAX; x; m[l++]=x%MAX, x/=MAX)
20            ;
21        if(sign==1&&l==1&&m[0]==0) sign=1;
22        return *this;
23    }
24    BigInt operator =(const char *t)
25    {
26        int i, j, len;
27        const char *s;
28        if(t[0]=='-') sign=-1, s=t+1;
29        else sign=1, s=t;
30        for(len=0; s[len]!='\0' && s[len]!='9'; len++);
31        for(l=(len+MAX_LOG-1)/MAX_LOG, i=0; i<l; i++)
32            for(m[i]=0, j=0; j<MAX_LOG; j++)
33                if(len-i*MAX_LOG-MAX_LOG+j>0)
34                    m[i]=m[i]*10+s[len-i*MAX_LOG-MAX_LOG+j]-'0';
35        if(sign==1&&l==1&&m[0]==0) sign=1;
36        return *this;
37    }
38    bool scan()
39    {
40        char s[MAX_N*MAX_LOG+10];
41        if(scanf("%s", s)==EOF) return 0;
42        else { *this=s; return 1; }
43    }
44    void print()
45    {
46        int i;
47        char s[8];
48        if(sign==-1) printf("-");
49        for(sprintf(s, "%08d", MAX_LOG), printf("%lld", m[l-1]), i=l-2; i>=0; printf(s, m[i]), i--);
50    }
51    bool operator <(const BigInt &x, const BigInt &y)
52    {
53        if(x.sign!=y.sign) return x.sign<y.sign;
54        int i;
55        if(x.l!=y.l) return (x.l<y.l) ^ (x.sign==-1);
56        for(i=x.l-1; i>=0 && x[i]!=y[i]; i--);
57        return (i>=0 && x[i]<y[i]) ^ (x.sign==-1);
58    }
59    bool operator ==(const BigInt &x, const BigInt &y)
60    {
61        if(x.sign!=y.sign) return false;
62        int i;
63        if(x.l!=y.l) return 0;
64        for(i=x.l-1; i>=0 && x[i]==y[i]; i--);
65        return i<0;
66    }
67    BigInt operator +(BigInt x, const BigInt &y)
68    {
69        int i;
70        long long h;
71        for(h=0, i=0; i<x.l || i<y.l || h; i++)
72        {
73            h+=(i<x.l)*x[i]*x.sign+(i<y.l)*y[i]*y.sign;
74            x[i]=h%MAX;
75            h/=MAX;
76        }
77        x.l=i;
78        for(; x.l>1 && !x[x.l-1]; x.l--);
79        x.sign=(x[x.l-1]>0?-1);
80        if(x[x.l-1]>0){ for(i=0; i<x.l; i++) if(x[i]<0) x[i+1]--, x[i]+=MAX; }
81        else for(i=0; i<x.l; i++) if(x[i]>0) x[i+1]++, x[i]-=MAX;
82        for(i=0; i<x.l; i++) x[i]*=x.sign;
83        if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
84        return x;
85    }
86    BigInt operator -(BigInt x, const BigInt &y)
87    {
88        int i;
89        long long h;
90        for(h=0, i=0; i<x.l || i<y.l || h; i++)
91        {
92            h+=(i<x.l)*x[i]*x.sign-(i<y.l)*y[i]*y.sign;
93            x[i]=h%MAX;
94            h/=MAX;
95        }
96        x.l=i;
97        for(; x.l>1 && !x[x.l-1]; x.l--);
98        x.sign=(x[x.l-1]>0?-1);
99        if(x[x.l-1]>0){ for(i=0; i<x.l; i++) if(x[i]<0) x[i+1]--, x[i]+=MAX; }
100        else for(i=0; i<x.l; i++) if(x[i]>0) x[i+1]++, x[i]-=MAX;
101        for(; x.l>1 && !x[x.l-1]; x.l--);
102        for(i=0; i<x.l; i++) x[i]*=x.sign;
103        if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
104        return x;
105    }
106    BigInt operator *(BigInt x, int y)
107    {
108        int i, sign=1;
109        long long h;
110        if(y<0) y=-y, sign=-1;
111        for(h=0, i=0; i<x.l || h; i++)
112        {
113            h+=(i<x.l)*x[i]*y;
114            x[i]=h%MAX;
115            h/=MAX;
116        }
117        for(x.l=i; x.l>1 && !x[x.l-1]; x.l--);
118        x.sign=(x.sign==sign?-1);
119        if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
120        return x;
121    }
122    BigInt operator /(BigInt x, int y)
123    {
124        int i, sign=1;
125        long long h;
126        if(y<0) y=-y, sign=-1;
127        for(h=0, i=x.l-1; i>=0; i--)
128        {
129            h=h*MAX+x[i];
130            x[i]=h/y;
131            h%=y;
132        }
133        for(; x.l>1 && !x[x.l-1]; x.l--);
134        x.sign=(x.sign==sign?-1);
135        if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
136        return x;
137    }
138    int operator %(BigInt x, int y)
139    {
140        int i;
141        long long h;
142        for(h=0, i=x.l-1; i>=0; i--)
143        {
144            h=h*MAX+x[i];
145            h%=y;
146        }
147        if(x.sign==-1) h=-h;
148        return h;
149    }
150    long long fl(double x) { return x<0?x-0.5:x+0.5; }
151    BigInt operator *(BigInt x, const BigInt &y)
152    {
153        if(y.l==1) return x*(y[0]*y.sign);
154        int i, N;
155        long long t;
156        vector<Complex> a, b;
157        for(i=0; i<x.l; i++) a.emplace_back(x[i]);
158        for(i=0; i<y.l; i++) b.emplace_back(y[i]);
159        for(N=1; N<x.l+y.l; N<=1);
160        while(N!=(int)(a.size())) a.emplace_back(0);
161        while(N!=(int)(b.size())) b.emplace_back(0);
162        a=fft(a), b=fft(b);
163        for(i=0; i<N; i++) a[i]=a[i]*b[i];
164        a=fft(a, -1);
165        for(i=0, t=0, x.l=0; i<N; i++)
166        {
167            t+=fl(a[i].x);
168            x[x.l++]=(t%MAX);
169            t/=MAX;
170        }
171        x[x.l++]=t;
172        for(; x.l>1 && !x[x.l-1]; x.l--);
173        x.sign=(x.sign==y.sign?-1);
174        if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
175        return x;
176    }
177    BigInt operator /(BigInt x, const BigInt &y)
178    {
179        if(y.l==1) return x/(y[0]*y.sign);
180        int i;
181        BigInt h;
182        for(h=0, i=x.l-1; i>=0; i--)
183        {
184            h=h*MAX+x[i];
185            if(h.l>y.l) x[i]=(h[h.l-1]*MAX+MAX+h[h.l-2]*MAX+h[h.l-3]);
186            ;
187            if(h.l==y.l) x[i]=(h[h.l-1]*MAX+MAX+h[h.l-2]);
188            x[i]/=(y[y.l-1]*MAX+y[y.l-2]);
189            for(; x[i] && h<y*(x[i]*y.sign); x[i]--);
190            h=h-(y*(x[i]*y.sign));
191        }
192        for(; x.l>1 && !x[x.l-1]; x.l--);
193        x.sign=(x.sign==y.sign?-1);
194        if(x.sign==1&&x.l==1&&x[0]==0) x.sign=1;
195        return x;
196    }
197    BigInt operator %(BigInt x, BigInt y)
198    {

```

```

197 if(y.l==1) return x%(y[0]*y.sign);
198 return x-(x/y)*y;
199 }

```

## 5.6 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 {
9     int n = A.size();
10    mat B(n, vec(n+1));
11    for(int i=0; i<n; i++) for(int j=0; j<n; j++) B[i][j]=A[i][j];
12
13    for(int i=0; i<n; i++) B[i][n]=b[i];
14
15    for(int i=0; i<n; i++)
16    {
17        int pivot=i;
18        for(int j=i; j<n; j++)
19        {
20            if(abs(B[j][i])>abs(B[pivot][i])) pivot=j;
21        }
22        swap(B[i], B[pivot]);
23        if(abs(B[i][i])<EPS) return vec(); //no/inf sol
24
25        for(int j=i+1; j<=n; j++) B[i][j]/=B[i][i];
26        for(int j=0; j<n; j++)
27        {
28            if(i!=j)
29            {
30                for(int k=i+1; k<=n; k++)
31                    B[j][k]-=B[i][i]*B[i][k];
32            }
33        }
34    }
35    vec x(n);
36    for(int i=0; i<n; i++) x[i]=B[i][n];
37    return x;
38 }

```

## 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  */
6 const int M = 100005;
7 ll sp[M], mobius[M];
8 void sieve()
9 {
10    for(ll i = 0; i < M; i++) sp[i] = i;
11    for(ll i = 2; i < M; i++) if(sp[i] == i)
12    {
13        for(ll j = i*i; j < M; j += i)

```

```

14        if(sp[j] == j) sp[j] = i;
15    }
16 }
17 void makeMobius()
18 {
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    {
23        for(ll j = i; j < M; j += i) mobius[j] = -mobius[j];
24        for(ll j = i*i; j < M; j += i*i) mobius[j] = 0;
25    }
26 }

```

## 5.8 extgcd

```

1 int extgcd(int a, int b, int &x, int &y)
2 {
3     int gcd = a;
4     if(b != 0)
5     {
6         gcd = extgcd(b, a%b, y, x);
7         y -= (a/b)*x;
8     }
9     else x = 1, y = 0;
10    return gcd;
11 }
12 //維護 a*x+b*y=gcd(a, b)

```

## 6 String

### 6.1 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 {
5     int len=strlen(S), k=0;
6     for(int i=0; i<len; i++)
7     {
8         if(Rank[i]==0) continue;
9         while(S[i+k] == S[SA[Rank[i]-1]+k]) k++;
10        Height[Rank[i]]=k;
11        if(k) k--;
12    } Height[0]=0;
13    for(int i=0; i<len; i++) dp_height[i][0]=Height[i];
14    for(int i=0; i<len; i++) for(int j=1; i+(1<<j)<len; j++)
15        dp_height[i][j]=min(dp_height[i][j-1], dp_height[i+(1<<(j-1))][j-1]);
16 }
17 int height_query(int x, int y)
18 {
19     int k=0;
20     while((1<<(k+1))<=y-x) k++;
21     return min(dp_height[x+1][k], dp_height[y-(1<<k)+1][k]);
22 }

```

## 6.2 AC-Automation

```

1 #define SZ 25000
2 int nx[SZ][26], spt;
3 int fl[SZ], efl[SZ], ed[SZ];
4 int newnode()
5 {
6     for(int i=0; i<26; i++) nx[spt][i]=0;
7     ed[spt]=0;
8     return spt++;
9 }
10 int add(char *s, int sptnow)
11 {
12     for(int i=0; s[i]; i++)
13     {
14         int tmp=s[i]-'a';
15         if(nx[sptnow][tmp]==0) nx[sptnow][tmp]=newnode();
16         sptnow=nx[sptnow][tmp];
17     }
18     ed[sptnow]=1;
19     return sptnow;
20 }
21 int bfsq[SZ], qs, qe;
22 void make_fl(int root)
23 {
24     fl[root]=efl[root]=qs=qe=0;
25     bfsq[qe++]=root;
26     while(qs!=qe)
27     {
28         int p=bfsq[qs++];
29         for(int i=0; i<26; i++)
30         {
31             int t=nx[p][i];
32             if(t==0) continue;
33             int tmp=fl[p];
34             for(; tmp&&nx[tmp][i]==0; tmp=fl[tmp]);
35             fl[t]=tmp?nx[tmp][i]:root;
36             efl[t]=ed[fl[t]]?fl[t]:efl[fl[t]];
37             bfsq[qe++]=t;
38         }
39     }
40 }

```

## 6.3 KMP

```

1 void failure_build(const char *p, int *fail)
2 {
3     for(int i=1, j=fail[0]=-1; p[i]; i++)
4     {
5         while(j>=0&&p[j+1]!=p[i]) j=fail[j];
6         if(p[j+1]==p[i]) j++;
7         fail[i]=j;
8     }
9 }
10 int KMP(const char *T, const char *P, int *fail)
11 {
12     failure_build(P, fail);
13     for(int i=0, j=-1; T[i]; i++)
14     {
15         while(j>=0&&P[j+1]!=T[i]) j=fail[j];
16         if(P[j+1]==T[i]) j++;
17         if(!P[j+1]) return i-j;

```

```

18 }
19 return -1;
20 }
21
22 //使用方法: KMP(主字串, 待匹配字串, failure array)
23 //回傳: 第一個完全匹配的位置

```

## 6.4 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
   *Rank)
5 {
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    {
12        if(SA[i]==len) pivot=j;
13        if(SA[i]<len)
14            out[j++]=in[(SA[i]+len-1)%len];
15    }
16    out[len]='|0';
17 }
18
19 inline void IBWT(char *in, char *out, int *tmp)
20 {
21     int len=strlen(in);
22     vector<int> idx[256];
23     for(int i=0;i<len;i++)
24         idx[in[i]].emplace_back(i);
25     for(int i=0,k=0;i<256;i++)
26         for(int j=0;j<(int)(idx[i].size());j++)
27             tmp[k++]=idx[i][j];
28     int p=pivot;
29     for(int i=0;i<len;i++)
30         out[i]=in[p=tmp[p]];
31     out[len]='|0';
32 }

```

## 6.5 Z-value

```

1 void Z_build(const char *S, int *Z)
2 {
3     Z[0]=0;
4     int b=0;
5     for(int i=1;S[i];i++)
6     {
7         if(Z[b]+b<i) Z[i]=0;
8         else Z[i]=min(Z[b]+b-i,Z[i-b]);
9         while(S[i+Z[i]]&&S[Z[i]]==S[i+Z[i]]) Z[i]++;
10        if(Z[i]+i>Z[b]+b) b=i;
11    }
12 }

```

## 6.6 SuffixArray

```

1 void SA_radix_sort(int *s, int *e, int *Rank, int rankcnt)
2 {
3     int box[MAX_N], tmp[MAX_N], len=e-s;
4     memset(box,0,sizeof(int)*rankcnt);
5     for(int i=0;i<len;i++) box[Rank[i]]++;
6     for(int i=1;i<rankcnt;i++) box[i]=box[i]+box[i-1];
7     for(int i=len-1;i>=0;i--) tmp[--box[Rank[s[i]]]]=s[i];
8     for(int i=0;i<len;i++) s[i]=tmp[i];
9 }
10
11 #define equal(a,b,c) c[a]!=c[b]||a+k>=len||c[a+k]!=c[b+k]
12 void SA_build(int *SA, int *Rank, char *S)
13 {
14     int ranktmp[MAX_N], len=strlen(S), rankcnt='z'+1;
15     for(int i=0;i<len;i++) Rank[i]=S[i];
16     for(int k=1;rankcnt!=len;k*=2)
17     {
18         for(int i=0;i<len;i++) SA[i]=(i+len-k)%len;
19         SA_radix_sort(SA+k, SA+len, Rank+k, rankcnt);
20         SA_radix_sort(SA, SA+len, Rank, rankcnt);
21         ranktmp[SA[0]]=0, rankcnt=0;
22         for(int i=1;i<len;i++)
23             ranktmp[SA[i]]=rankcnt+=equal(SA[i-1], SA[i], Rank);
24         rankcnt++;
25         for(int i=0;i<len;i++) Rank[i]=ranktmp[i];
26     }
27 #undef equal

```

## 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 {
16     if (visit[i] == 1 || sat[i] == 1) return true;
17     if (visit[i] == 2 || sat[i] == 2) return false;
18     visit[i] = 1;
19     visit[not(i)] = 2;
20     for (int j=0; j<N+N; ++j)
21         if (adj[i][j] && !dfs_try(j))
22             return false;
23     return true;
24 }
25
26 void dfs_mark(int i)

```

```

27 {
28     if (sat[i] == 1) return;
29     sat[i] = 1;
30     sat[not(i)] = 2;
31     for (int j=0; j<N+N; ++j)
32         if (adj[i][j])
33             dfs_mark(j);
34 }
35
36 void two_satisfiability()
37 {
38     // 一次輸入一個括號
39     memset(adj, false, sizeof(adj));
40     int a, b;
41     while (cin >> a >> b)
42     {
43         map[not(a)][b] = true;
44         map[not(b)][a] = true;
45     }
46
47     // 找出一組解
48     for (int i=0; i<N; ++i)
49     {
50         memset(visit, 0, sizeof(visit));
51         if (dfs_try(i)) {dfs_mark(i); continue;}
52
53         memset(visit, 0, sizeof(visit));
54         if (dfs_try(not(i))) {dfs_mark(not(i)); continue;}
55
56         // 無解則立即結束。
57         return;
58     }
59
60     // 印出一組解。
61     for (int i=1; i<N; ++i)
62         if (sat[i] == 1)
63             cout << i;
64         else /*if (sat[i] == 2)*/
65             cout << "not" << i;
66 }

```

### 7.2 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 {
21     for(int i = 1; i <= vn; i++)
22         graph[i].clear(), isCentroid[i] = false;
23 }
24
25 void buildTree()
26 {
27     for(int i = 1; i < vn; i++)
28     {
29         int u, v, l; scanf("%d %d %d", &u, &v, &l);
30         graph[u].push_back(P(v, l));
31         graph[v].push_back(P(u, l));
32     }
33 }
34
35 ll calSubsz(int v, int p)
36 {
37     subtreeSz[v] = 1;
38     for(auto c:graph[v])
39     {
40         if(isCentroid[c.idx] || c.idx == p) continue;
41         subtreeSz[v] += calSubsz(c.idx, v);
42     }
43     return subtreeSz[v];
44 }
45
46 P getCentroid(int v, int p, ll subsz)
47 {
48     P cen(-1, INF);
49     ll mxsonSz = -1;
50     for(auto c:graph[v])
51     {
52         if(c.idx == p || isCentroid[c.idx]) continue;
53         P res = getCentroid(c.idx, v, subsz);
54         if(res.w < cen.w) cen = res;
55         mxsonSz = max(mxsonSz, subtreeSz[c.idx]);
56     }
57     mxsonSz = max(mxsonSz, subsz - subtreeSz[v]);
58     if(mxsonSz < cen.w) cen = P(v, mxsonSz);
59     return cen;
60 }
61
62 void getDist(int v, int p, ll w)
63 {
64     if(w > k) return;
65     dist.push_back(w);
66     for(auto c:graph[v])
67     {
68         if(c.idx == p || isCentroid[c.idx]) continue;
69         getDist(c.idx, v, w+c.w);
70     }
71 }
72
73 ll calValidPair(int idx, ll w)
74 {
75     dist.clear();
76     getDist(idx, -1, w);
77     sort(dist.begin(), dist.end());
78     ll sum = 0;
79     for(int l = 0, r = dist.size()-1; l < r; )
80     {
81         if(dist[r]+dist[l] <= k) sum += r-l, l++;
82         else r--;
83     }
84     return sum;
85 }

```

```

86 }
87
88 ll treedc(int v)
89 {
90     ll sum = 0;
91     // find centroid
92     calSubsz(v, v);
93     int cen = getCentroid(v, v, subtreeSz[v]).idx;
94     isCentroid[cen] = true;
95
96     sum += calValidPair(cen, 0);
97     for(auto c:graph[cen])
98     {
99         if(isCentroid[c.idx]) continue;
100         sum -= calValidPair(c.idx, c.w);
101         sum += treedc(c.idx);
102     }
103     return sum;
104 }
105
106 int main()
107 {
108     while(scanf("%d %lld", &vn, &k) && vn && k)
109     {
110         init();
111         buildTree();
112         printf("%lld\n", treedc(1));
113     }
114     return 0;
115 }

```

### 7.3 definss

```

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 }

```



# ACM ICPC TEAM

## REFERENCE -

### NTHU\_FURRYFORCE

#### Contents

<b>1</b>	<b>DataSetructure</b>	<b>1</b>	<b>3</b>	<b>Geometry</b>	<b>5</b>	<b>5</b>	<b>Math</b>	<b>11</b>
1.1	BIT	1	3.1	intercircle	5	5.1	modeq	11
1.2	2d_st_tag	1	3.2	point	5	5.2	FFT	11
1.3	undo_disjoint_set	1	3.3	SegmentGeometry	5	5.3	NTT	11
1.4	disjoint_set	1	3.4	nearestDist	6	5.4	EulerPhi	11
1.5	1d_segTree_tag	2	<b>4</b>	<b>Graph</b>	<b>6</b>	5.5	BigInt	11
1.6	Matrix	2	4.1	StronglyConnectedComponent	6	5.6	GaussianJordan	13
1.7	treap	3	4.2	bellman_Ford	6	5.7	mobius	13
1.8	1d_segTree	3	4.3	MaxMatching	6	5.8	extgcd	13
<b>2</b>	<b>Flow</b>	<b>3</b>	4.4	ArticulationPoint	7	<b>6</b>	<b>String</b>	<b>13</b>
2.1	MaxDensitySubgraph	3	4.5	TwoConnected	7	6.1	LCP	13
			4.6	TarjanBridge	7	6.2	AC-Automation	13
			4.7	MaxWeightPerfectMatch	8	6.3	KMP	13
			4.8	lca	8	6.4	BWT	14
			4.9	MaximalClique	8	6.5	Z-value	14
			4.10	MinimumMeanCycle	9	6.6	SuffixArray	14
			4.11	匹配問題轉換	9	<b>7</b>	<b>other</b>	<b>14</b>
			4.11.1	一般圖匹配問題轉換	9	7.1	2sat	14
			4.12	BridgeConnected	9	7.2	PojTree	14
			4.13	HeavyLightDecomposition	9	7.3	definesss	15
			4.14	MaxWeightPerfectBiMatch	9			
			4.15	tarjan	10			
			2.2	dinic	4			
			2.3	MinCostMaxFlow	4			
			4.16	dijkstra	10			
			4.17	spfa	10			
			4.18	MaxBiMatching	10			