图论

弦图与区间图

记：
w(G) 团数 最大团的点数
x(G) 色数 最小染色(使相邻点颜色不同)的色数
a(G) 最大独立集
k(G) 最小团覆盖 用最少个数的团覆盖所有的点
有定理:
w(G) <= x(G)
a(G) <= k(G)

弦(chord)：连接环中不相邻的两个点的边
弦图(chordal graph)：一个无向图称为弦图当图中任意长度大于 3 的环都至少有一个弦。
弦图的每一个诱导子图一定是弦图。
弦图的任一个诱导子图不同构于 Cn (n > 3)

单纯点(simplicial vertex)：
设 N(v)表示与点 v 相邻的点集。
一个点称为单纯点当{v} + N(v)的诱导子图为一个团。
引理：
任何一个弦图都至少有一个单纯点，不是完全图的弦图至少有两个不相邻的单纯点。
完美消除序列(perfect elimination ordering)
一个点的序列(每个点出现且恰好出现一次)v1, v2, …, vn 满足
vi 在{vi, vi+1,…,vn}的诱导子图中为一个单纯点。
定理：一个无向图是弦图当且仅当它有一个完美消除序列。
证明：
充分性
由引理知任何一个弦图都至少有一个单纯点以及弦图的诱导子图都是弦图。可以使用数学归纳法假设当点数<n 的弦图一定有完美消除序列，那么点数为 n 的弦图的完美消除序列可以由一个单纯点加上剩余点的诱导子图的完美消除序列得到。
必要性
反证若无向图存在一个长度 > 3 的无弦环，不妨设环中在完美消除序列中出现在最前面的点为 v，设环中 v 与 v1, v2 相连，根据完美消除序列的性质知 v1, v2 相连，与环无弦矛盾。所以无向图为弦图。
两个求完美消除序列 O(m + n)的算法
字典序广度优先搜索(Lexicographic BFS)
从 n 到 1 的顺序依次给点标号。
每个点维护一个 list 记录与它相邻的已标号点的标号，list 中的标号按照按从大到小排序。
每次选择 list 字典序最大的未标号点标号。
LexBFS 与 BFS 不同在于每次扩展的节点加了特殊的顺序。

最大势算法 Maximum Cardinality Search(曾用过)
从 n 到 1 的顺序依次给点标号(标号为 i 的点出现在完美消除序列的第 i 个)。
设 label[i]表示第 i 个点与多少个已标号的点相邻，每次选择 label[i]最大的未标号的点进行标号。
判断一个序列是否为完美消除序列
设{vi+1,…,vn}中所有与 vi 相邻的点依次为 vj1, …, vjk。
只需判断 vj1 是否与 vj2, …, vjk 相邻即可。
时间复杂度: O(m + n)
弦图的极大团
设第 i 个点在弦图的完美消除序列第 p(i)个。
令 N(v) = {w | w 与 v 相邻且 p(w) > p(v)}
弦图的极大团一定是 v | N(v) 的形式。
判断 v | N(v) 是否为极大团
设 A = v | N(v) ，若存在 B = w | N(w) 使得 B 包含 A，则 A 不是极大团。
p(w) < p(v)
设 next(v)表示 N(v)中最前的点。令 w*表示所有满足 B 包含 A 的 w 中最后的一个点。
next(w*) = v (否则 next(w*)也是满足条件的 w)
Next(w) = v
B 包含 A 当且仅当|N(v)| + 1 ≤ |N(w)|
只需判断是否存在一个 w，满足 Next(w) = v 且|N(v)| + 1 ≤ |N(w)|即可。
时间复杂度：O(m + n)
弦图的点染色问题
用最少的颜色给每个点染色使得相邻的点染的颜色不同
完美消除序列从后往前依次给每个点染色，给每个点染上可以染的最小的颜色
证明:
设使用了 T 种颜色，则 T ≥ 色数
T = 团数 ≤ 色数
团数 = 色数 = T
时间复杂度： O(m + n)
弦图的最大独立集
完美消除序列从前往后能选就选。
弦图的最小团覆盖
设最大独立集为{p1 , p2 , …, pt}，则{p1∪N(p1), …, pt∪N(pt)}为最小团覆盖。
弦图 最大独立集数 = 最小团覆盖数

完美图
一个图 G 称为完美图若它的每一个诱导子图都满足 w(G) == x(G)
伴完美图的概念
一个图 G 称为伴完美图若它的每一个诱导子图都满足 a(G) == k(G)
完美图 = 伴完美图
弦图属于完美图

区间图

给定一些区间，定义一个相交图为每个顶点表示一个区间，两个点有边当且仅当两个区间的交集非空。

一个图为区间图当它是若干个区间的相交图。

区间图一定是弦图

[例题 1]给定 n 个区间，要求选择最多的区间，使得区间不互相重叠

Solution 区间图的最大独立集

区间图 G 的一个完美消除序列：

将所有的区间按照右端点从小到大排序。

图树剖分（不会来）

最小树形图 zhj

//最小树形图，就是给有向带权图中指定一个特殊的点 root，求一棵以 root 为根的有向生成树 T，并且 T 中所有边的总权值最小。

//题意：

//N 个点 M 条边的有向图

//找到点 x 使其能到达所有其它点

//并且需要经过的边权和最小

//边权为 0 或 1

//输出方案中边权为 1 的边

```
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
const int mm=120000;
struct ppp{
    int x,y,co,a,b,use,link;
} e[mm],q[1200000];
int i,j,k,l,n,m,be[mm],bo[mm],root,in[mm],pre[mm],ans,last[mm],
    nn,tot;
int Deric_MST(){
    root=1;
    int i,j,k,M,res=0;
    while(true){
        //zhao zui xiao ru du bian
        for(i=1;i<=n;++i)in[i]=1e9,bo[i]=be[i]=last[i]=0;
        for(i=1;i<=m;++i)if(e[i].x!=e[i].y   &&   e[i].y!=root   &&
e[i].co<in[e[i].y]){
            in[e[i].y]=e[i].co;
            pre[e[i].y]=e[i].x;
            last[e[i].y]=e[i].link;
```

```
        }
        for(i=1;i<=n;++i){
            if(i!=root && in[i]==1e9)return -1;
            if(i!=root)res+=in[i];
        }
    //  for(i=1;i<=n;++i)printf("%d %d\n",i,pre[i]);
    //  printf("\n");
        //suo dian
        M=0;
        for(i=1;i<=n;++i)if(i!=root){
            q[last[i]].use++;
            for(j=i;bo[j]!=i&&j!=root&&be[j]==0;j=pre[j])bo[j]=i;
            if(j!=root && be[j]==0){
                M++;
                for(k=j;pre[k]!=j;k=pre[k])be[k]=M;
                be[k]=M;
            }
        }
        if(M==0)return res;
        for(i=1;i<=n;++i)if(be[i]==0)be[i]=++M;
        //geng xing bian
        for(i=1;i<=m;++i){
            int v=e[i].y;
            e[i].x=be[e[i].x];
            e[i].y=be[e[i].y];
            if(e[i].x!=e[i].y){
                e[i].co-=in[v];
                q[++tot]=e[i];
                q[tot].use=0;
                q[tot].a=e[i].link;
                q[tot].b=last[v];
                e[i].link=tot;
            }
        }
        n=M;
        root=be[root];
//      printf("%d\n",n);
    }
}

int main(){
    freopen("input.txt","r",stdin);
    freopen("output.txt","w",stdout);
    scanf("%d%d",&n,&m);
```

```
    nn=n;
    for(i=1;i<=m;++i){
        scanf("%d%d%d",&e[i].x,&e[i].y,&e[i].co);
        q[++tot]=e[i];
        e[i].link=tot;
    }
    ans=Deric_MST();
    if(ans==-1)printf("-1\n");
    else{
        printf("%d\n",ans);
        for(i=tot;i>m;--i){
            q[q[i].a].use+=q[i].use;
            q[q[i].b].use-=q[i].use;
        }
        for(i=1;i<=m;++i)if(q[i].co==1   &&   q[i].use>0)printf("%d
",i);
    }
}
```

SPFA_DFS 实现 可盼负环

```
//dfs 实现 spfa
//可判断是否出现负环
//返回 0 表示出现负环
bool dfs (int x) {
    use[x] = 1;
    bool flag = 1;
    for (int i = st[x]; i; i = ne[i])
        if (dist[go[i]] < dist[x] + w[i]) {
            if (use[go[i]])
                return 0;
            dist[go[i]] = dist[x] + w[i];
            flag &= dfs (go[i]);
            if (!flag) return 0;
        }
    use[x] = 0;
    return 1;
}
```

Tarjan 求割点

```
struct EdgeTp {
    int adj;
    int next;
```

```
} g[MaxN];
int n, tot;
int first[MaxN], dfn[MaxN], low[MaxN], par[MaxN];
void addedge (int x, int y) {
    tot++;
    g[tot].adj = y;
    g[tot].next = first[x];
    first[x] = tot;
}
void init() {
    int x, y;
    scanf ("%d", &n);
    tot = 0;
    while (scanf ("%d %d", &x, &y) != EOF) {
        addedge (x, y);
        addedge (y, x);
    }
    tot = 0;
}
void dfs (int u, int p) {
    int v, t;
    dfn[u] = low[u] = ++tot;
    par[u] = p;
    t = first[u];
    while (t != 0) {
        v = g[t].adj;
        if (!dfn[v]) { //Ê÷Ö¦±ß
            dfs (v, u);
            low[u] = min (low[u], low[v]);
        } else if (v != p && dfn[v] < low[u]) //ºóÏò±ß
            low[u] = dfn[v];
        t = g[t].next;
    }
}
void solve() {
    int i, u, v, rc = 0, gdc = 0; //rcÎª¸ù½Úµã×ÓÊ÷Êý,gdcÎª¸îµãÊý
    bool flag[MaxN] = {false}; //±ê¼ÇÊÇ·ñÎª¸îµã
    dfs (1, 0);
    for (v = 2; v <= n; v++) {
        u = par[v];
        if (u == 1) //uÎªÊ÷¸ù
            rc++;
        else//u²»ÎªÊ÷¸ù
            if (dfn[u] <= low[v])
```

```
            flag[u] = true;
    }
    if (rc >= 2)
        flag[1] = true;
    for (i = 1; i <= n; i++)
        if (flag[i])
            gdc++;
    printf ("%d\n", gdc);
    for (i = 1; i <= n; i++)
        if (flag[i])
            printf ("%d\n", i);
}
int main() {
    freopen ("gd.in", "r", stdin);
    freopen ("gd.out", "w", stdout);
    init();
    solve();
    return 0;
}
```

Tarjan——GY:

```
//tot, last, num
//mark[N]表示点 x 所在强连通块序号
//dfn[N]表示 dfn 序号
//low[N]表示点 x 能到的点的最小 dfn
//sta[]为栈
//vis[]为点是否在栈内,是否出栈

void tarjan (int x) {
    dfn[x] = low[x] = ++tot;
    sta[++last] = x;
    vis[x] = 1;
    for (int i = st[x]; i; i = ne[i])
        if (vis[go[i]] == 0) {
            tarjan (go[i]);
            low[x] = min (low[x], low[go[i]]);
      } else if (vis[go[i]] == 1) low[x] = min (low[x], dfn[go[i]]);
    if (dfn[x] == low[x]) {
        ++num;
        int p;
        do {
            p = sta[last];
            mark[p] = num;
```

```
            vis[p] = 2;
            sta[last--] = 0;
        } while (p != x);
    }
}
```

Tarjan 求双联通分量

```
#define MAXN 1001
#define min(a,b) (a<b?a:b)

typedef pair<int, int> PAIR;

int p[MAXN], ecnt, n, m, dfn[MAXN], lowlink[MAXN], sign, st,
color[MAXN];
bool hate[MAXN][MAXN], mark[MAXN], ans[MAXN];
stack<PAIR> sta;

struct Edge {
    int v, next;
} edg[MAXN * MAXN];


void init() {
    ecnt = 0;
    memset (p, -1, sizeof (p) );
    memset (dfn, -1, sizeof (dfn) );
    memset (hate, false, sizeof (hate) );
    sign = 0;
    while (!sta.empty() )
        sta.pop();
    memset (ans, false, sizeof (ans) );
}

bool check_odd (int u, int col) { //¼ì²é¸Ã×Ó¿éÊÇ·ñÎª¶þ·ÖÍ¼
    int i, v;
    color[u] = col;
    for (i = p[u]; i != -1; i = edg[i].next) {
        v = edg[i].v;
        if (mark[v]) {
            if (color[v] == 0) {
                if (check_odd (v, -col) )
                    return true;
            } else if (col == color[v])
```

```
            return true;
        }
    }
    return false;
}
void dfs (int pre, int u) {
    int i, j, v, x, y;
    PAIR P;
    dfn[u] = lowlink[u] = ++sign;
    for (i = p[u]; i != -1; i = edg[i].next) {
        v = edg[i].v;
        if (v != pre && dfn[u] > dfn[v]) //±ßu-vÎ´·ÃÎÊ¹ý
            sta.push (make_pair (u, v) );
        if (dfn[v] == -1) { //¶¥µãvÎ´·ÃÎÊ¹ý
            dfs (u, v);
            lowlink[u] = min (lowlink[u], lowlink[v]);
            if (dfn[u] <= lowlink[v]) { //uÊÇ¸îµã£¬ÕÒµ½¸öØÁ¬Í¨·ÖÁ¿
                memset (mark, false, sizeof (mark) );
                st = u;
                do {
                    P = sta.top();
                    x = P.first;
                    y = P.second;
                    sta.pop();
                    mark[x] = mark[y] = true;
                } while (! ( (x == u && y == v) || (x == v && y ==
u) ) );
                memset (color, 0, sizeof (color) );
                if (check_odd (st, 1) )
                    for (j = 1; j <= n; j++)
                        ans[j] |= mark[j];
            }
        } else if (v != pre)
            lowlink[u] = min (lowlink[u], dfn[v]);
    }
}

int solve() {
    int i, anscnt = 0;
    for (i = 1; i <= n; i++) {
        if (dfn[i] == -1) {
            dfs (-1, i);
        }
    }
```

```
    for (i = 1; i <= n; i++)
        if (ans[i])
            anscnt++;
    return n - anscnt;
}


int main() {
    int i, u, v;
    freopen ("knight.in", "r", stdin);
    freopen ("knight.out", "w", stdout);
    while (1) {
        scanf ("%d%d", &n, &m);
        if (n == 0 && m == 0) break;
        init();
        for (i = 0; i < m; i++) {
            scanf ("%d%d", &u, &v);
            hate[u][v] = hate[v][u] = true;
        }
        for (u = 1; u <= n; u++) {
            for (v = u + 1; v <= n; v++) {
                if (!hate[u][v]) {
                    edg[ecnt].v = v;
                    edg[ecnt].next = p[u];
                    p[u] = ecnt++;
                    edg[ecnt].v = u;
                    edg[ecnt].next = p[v];
                    p[v] = ecnt++;
                }
            }
        }


        printf ("%d\n", solve() );
    }
    return 0;
}
```