图论-有向图的强连通分量

一、AcWing 1174. 受欢迎的牛

【题目描述】

每一头牛的愿望就是变成一头最受欢迎的牛。

现在有N头牛，编号从$1\sim N$，给你M对整数(A,B)，表示牛A认为牛B受欢迎。

这种关系是具有传递性的，如果A认为B受欢迎，B认为C受欢迎，那么牛A也认为牛C受欢迎。

你的任务是求出有多少头牛被除自己之外的所有牛认为是受欢迎的。

【输入格式】

第一行两个数N,M；

接下来M行，每行两个数A,B，意思是A认为B是受欢迎的（给出的信息有可能重复，即有可能出现多个A,B）。

【输出格式】

输出被除自己之外的所有牛认为是受欢迎的牛的数量。

【数据范围】

$1\le N\le 10^4$

$1\le M\le 5\times 10^4$

【输入样例】

```
3 3
1 2
2 1
2 3
```

【输出样例】

```
1
```

【样例解释】

只有第三头牛被除自己之外的所有牛认为是受欢迎的。

【分析】

首先我们将题意抽象成一个有向图，如果A认为B受欢迎，那么连一条$A \to B$的有向边。建好图后最简单的方法是可以在反图上枚举每个点，然后判断该点是否能到达其它所有点，如果可以那么答案加一，但是这样会超时。

假设我们的图是一个拓扑图，那么如果出度为0的点大于1个，则答案一定为0，因为出度为0的点互相之间一定不认为对方受欢迎，那么就不存在任何一个点受其它所有点的欢迎。如果不是这种情况，那么答案就是出度为0的点所在的强连通分量中点的数量。

因此本题需要一个cnt数组维护每个强连通分量中点的数量，以及一个out数组维护每个强连通分量的出度。我们在做完Tarjan算法之后可以枚举每个点u，对于每个点我们再枚举它的每条边(u, v)，如果u和v不在一个强连通分量中，那么就将u所在的强连通分量的出度加一。这样就无需将缩点后的图建出来了。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 10010, M = 50010;
int e[M], ne[M], h[N], idx;
int dfn[N], low[N], timestamp, scc_cnt;
int stk[N], cnt[N], id[N], out[N];
bool in_stk[N];
int n, m, tt, res;

void add(int u, int v)
{
    e[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void tarjan(int u)
{
    dfn[u] = low[u] = ++timestamp;
    stk[++tt] = u, in_stk[u] = true;
    for (int i = h[u]; ~i; i = ne[i])

    {
```

```
        int j = e[i];
        if (!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }
        else if (in_stk[j]) low[u] = min(low[u], dfn[j]);
    }
    if (dfn[u] == low[u])
    {
        int t;
        scc_cnt++;
        do
        {
            t = stk[tt--];
            in_stk[t] = false;
            id[t] = scc_cnt;
            cnt[scc_cnt]++;
        } while (t != u);
    }
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    while (m--) { int a, b; cin >> a >> b; add(a, b); }
    for (int i = 1; i <= n; i++)
        if (!dfn[i]) tarjan(i);
    for (int u = 1; u <= n; u++)
        for (int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if (id[u] != id[j])//如果u和j不在一个强连通分量中则缩点后的图需要连接两个强连通分量
                out[id[u]]++;//u所在的强连通分量出度加一
        }
    int cnt0 = 0;//出度为0的强连通分量的数量
    for (int i = 1; i <= scc_cnt; i++)
        if (!out[i])
        {
            cnt0++;

            res = cnt[i];
```

```
                if (cnt0 > 1) { res = 0; break; }
            }
        cout << res << endl;
        return 0;
}
```

二、AcWing 367. 学校网络

【题目描述】

一些学校连接在一个计算机网络上，学校之间存在软件支援协议，每个学校都有它应支援的学校名单（学校A支援学校B，并不表示学校B一定要支援学校A）。

当某校获得一个新软件时，无论是直接获得还是通过网络获得，该校都应立即将这个软件通过网络传送给它应支援的学校。

因此，一个新软件若想让所有学校都能使用，只需将其提供给一些学校即可。

现在请问最少需要将一个新软件直接提供给多少个学校，才能使软件能够通过网络被传送到所有学校？

最少需要添加几条新的支援关系，使得将一个新软件提供给任何一个学校，其他所有学校就都可以通过网络获得该软件？

【输入格式】

第1行包含整数N，表示学校数量。

第$2\sim N+1$行，每行包含一个或多个整数，第$i+1$行表示学校i应该支援的学校名单，每行最后都有一个0表示名单结束（只有一个0即表示该学校没有需要支援的学校）。

【输出格式】

输出两个问题的结果，每个结果占一行。

【数据范围】

$2\leq N\leq 100$

【输入样例】

```
5
2 4 3 0
4 5 0
0
0
1 0
```

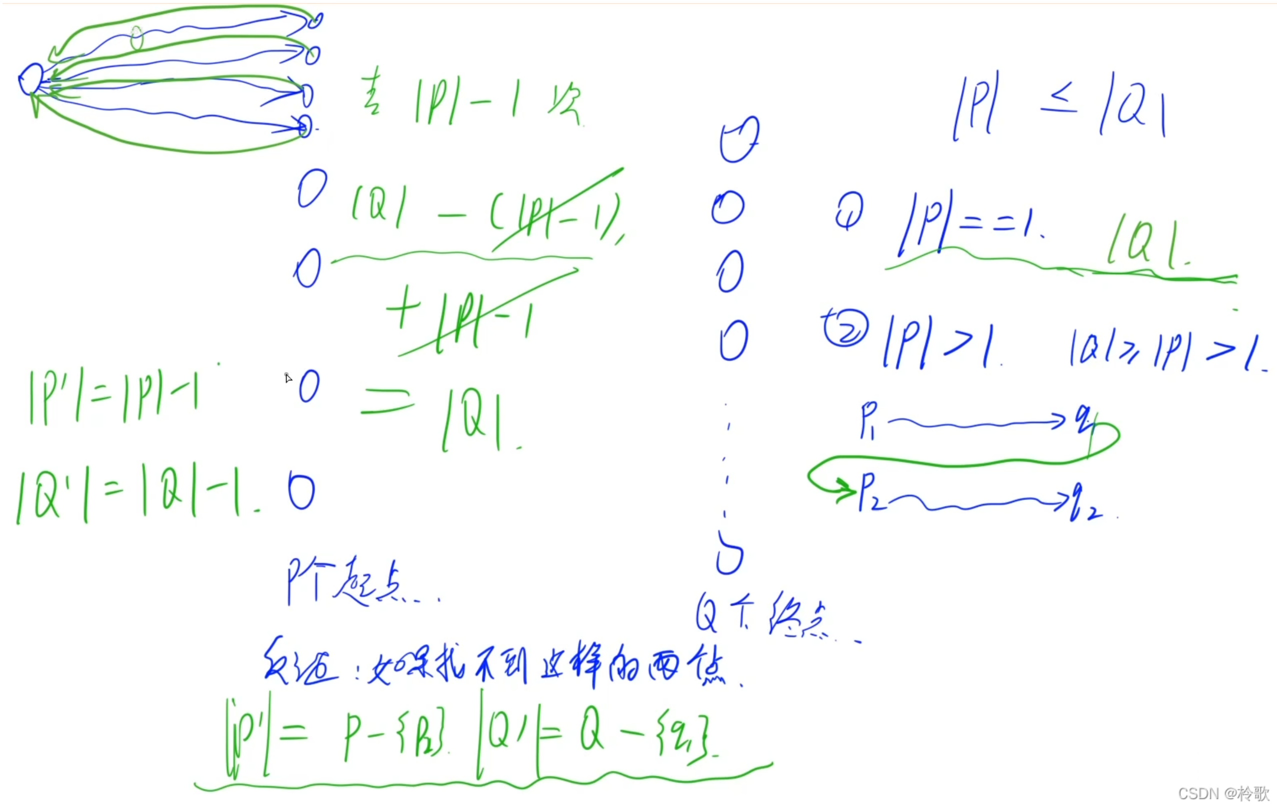
【输出样例】

```
1
2
```

【分析】

将原图转化成DAG图之后假设有P个起点，Q个终点（入度为0的点为起点，出度为0的点为终点），那么我们只需要给每个起点发一个软件即可让所有学校都能收到，因此第一问的答案就为P。

第二问是问我们至少需要加几条边能够使这个DAG图也变成一个强连通分量，答案是$max(P,Q)$，但是需要特判一下如果原图就只有一个强连通分量那么需要连的边数为0。



【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 110, M = 10010;
int e[M], ne[M], h[N], idx;
```

```
int dfn[N], low[N], timestamp, scc_cnt;
int stk[N], id[N], in[N], out[N];
bool in_stk[N];
int n, tt;

void add(int u, int v)
{
    e[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void tarjan(int u)
{
    dfn[u] = low[u] = ++timestamp;
    stk[++tt] = u, in_stk[u] = true;
    for (int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if (!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }
        else if (in_stk[j]) low[u] = min(low[u], dfn[j]);
    }
    if (dfn[u] == low[u])
    {
        int t;
        scc_cnt++;
        do
        {
            t = stk[tt--];
            in_stk[t] = false;
            id[t] = scc_cnt;
        } while (t != u);
    }
}

int main()
{
    cin >> n;
    memset(h, -1, sizeof h);
    for (int i = 1; i <= n; i++)

    {
```

```cpp
            int x;
            while (cin >> x, x) add(i, x);
        }
        for (int i = 1; i <= n; i++)
            if (!dfn[i]) tarjan(i);
        for (int u = 1; u <= n; u++)
            for (int i = h[u]; ~i; i = ne[i])
            {
                int j = e[i];
                if (id[u] != id[j]) out[id[u]]++, in[id[j]]++;
            }
        int cnt1 = 0, cnt2 = 0;//分别表示入度为0的点和出度为0的点的数量
        for (int i = 1; i <= scc_cnt; i++)
        {
            if (!in[i]) cnt1++;
            if (!out[i]) cnt2++;
        }
        cout << cnt1 << endl;
        if (scc_cnt == 1) cout << 0 << endl;//特判原图只有一个强连通分量
        else cout << max(cnt1, cnt2) << endl;
        return 0;
}
```

三、AcWing 1175. 最大半连通子图

【题目描述】

一个有向图$G=(V,E)$称为半连通的（Semi-Connected），如果满足：$\forall u,v\in V$，满足$u\rightarrow v$或$v\rightarrow u$，即对于图中任意两点u,v，存在一条u到v的有向路径或者从v到u的有向路径。

若$G'=(V',E')$满足，E'是E中所有和V'有关的边，则称G'是G的一个导出子图。

若G'是G的导出子图，且G'半连通，则称G'为G的半连通子图。

若G'是G所有半连通子图中包含节点数最多的，则称G'是G的最大半连通子图。

给定一个有向图G，请求出G的最大半连通子图拥有的节点数K，以及不同的最大半连通子图的数目C。

由于C可能比较大，仅要求输出C对X的余数。

【输入格式】

第一行包含三个整数N,M,X。N,M分别表示图G的点数与边数，X的意义如上文所述；

接下来M行，每行两个正整数a,b，表示一条有向边(a,b)。

图中的每个点将编号为$1\sim N$，保证输入中同一个(a,b)不会出现两次。

【输出格式】

应包含两行。

第一行包含一个整数K，第二行包含整数$C \bmod X$。

【数据范围】

$1\le N\le 10^5$

$1\le M\le 10^6$

$1\le X\le 10^8$

【输入样例】

```
6 6 20070603
1 2
2 1
1 3
2 4
5 6
6 4
```

【输出样例】

```
3
3
```

【分析】

先把所有强连通求出来并缩点后得到拓扑图，然后将新图建立出来，注意建新图的时候需要判重，因为题意中不同的最大半连通子图是指至少有一个点不同，而不是有一条边不同，选择导出子图的时候是先选择点，然后把和这些点有关的边全部选中，因此如果有多条边，那么其实导出子图也只算一种，所以如果不判重的话我们就会算多次。

由于Tarjan是逆DFS序，因此建完新图后我们从标号最大的点开始访问一定是满足拓扑序的，那么我们使用数组$f[i]$表示走到i时路径上的权值之和的最大值，$g[i]$表示满足最大值时的路径条数，$cnt[i]$表示某个强连通分量（即新图的某个点）中点的数量，则新图上每个点的权值即为$cnt[i]$。

我们在转移状态的时候有以下两种情况需要更新（假设从i走到j）：

- $f[i] + cnt[j] > f[j]$：那么更新$f[j] = f[i] + cnt[j], g[j] = g[i]$；
- $f[i] + cnt[j] = f[j]$：那么更新$g[j] += g[i]$。

最后我们再遍历一遍新图找出最大的$f[i]$以及这个值的所有方案数之和即可。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <set>
using namespace std;

typedef pair<int, int> PII;
const int N = 100010, M = 2000010;//缩点后还要再建一次图,最坏情况下边数是两倍
int e[M], ne[M], h[N], hs[N], idx;//hs为缩点后的邻接表表头
int dfn[N], low[N], id[N], cnt[N], timestamp, scc_cnt;
int f[N], g[N], stk[N];
bool in_stk[N];
int n, m, p, tt;

void add(int h[], int u, int v)
{
    e[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void tarjan(int u)
{
    dfn[u] = low[u] = ++timestamp;
    stk[++tt] = u, in_stk[u] = true;
    for (int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if (!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }
        else if (in_stk[j]) low[u] = min(low[u], dfn[j]);
    }
    if (dfn[u] == low[u])
    {
```

```
        int t;
        scc_cnt++;
        do
        {
            t = stk[tt--];
            in_stk[t] = false;
            id[t] = scc_cnt;
            cnt[scc_cnt]++;
        } while (t != u);
    }
}

int main()
{
    scanf("%d%d%d", &n, &m, &p);
    memset(h, -1, sizeof h);
    memset(hs, -1, sizeof h);
    while (m--) { int a, b; scanf("%d%d", &a, &b); add(h, a, b); }
    for (int i = 1; i <= n; i++)
        if (!dfn[i]) tarjan(i);
    set<PII> has_choose;
    for (int u = 1; u <= n; u++)
        for (int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if (id[u] != id[j] && !has_choose.count({ id[u], id[j] }))
                add(hs, id[u], id[j]), has_choose.insert({ id[u], id[j]
});
        }
    for (int u = scc_cnt; u; u--)//从大到小遍历缩点后的图一定是拓扑序的
    {
        if (!f[u]) f[u] = cnt[u], g[u] = 1;
        for (int i = hs[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if (f[u] + cnt[j] > f[j]) f[j] = f[u] + cnt[j], g[j] = g[u];
            else if (f[u] + cnt[j] == f[j]) g[j] = (g[j] + g[u]) % p;
        }
    }
    int res = 0, sum = 0;
    for (int i = 1; i <= scc_cnt; i++)
        if (f[i] > res) res = f[i], sum = g[i];

        else if (f[i] == res) sum = (sum + g[i]) % p;
```

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

四、AcWing 368. 银河

【题目描述】

银河中的恒星浩如烟海，但是我们只关注那些最亮的恒星。

我们用一个正整数来表示恒星的亮度，数值越大则恒星就越亮，恒星的亮度最暗是1。

现在对于N颗我们关注的恒星，有M对亮度之间的相对关系已经判明。

你的任务就是求出这N颗恒星的亮度值总和至少有多大。

【输入格式】

第一行给出两个整数N和M。

之后M行，每行三个整数T, A, B，表示一对恒星(A, B)之间的亮度关系。恒星的编号从1开始。

- 如果$T = 1$，说明A和B亮度相等。
- 如果$T = 2$，说明A的亮度小于B的亮度。
- 如果$T = 3$，说明A的亮度不小于B的亮度。
- 如果$T = 4$，说明A的亮度大于B的亮度。
- 如果$T = 5$，说明A的亮度不大于B的亮度。

【输出格式】

输出一个整数表示结果。

若无解，则输出-1。

【数据范围】

$N \le 100000, M \le 100000$

【输入样例】

```
5 7
1 1 2
2 3 2
4 4 1
3 4 5
5 4 5
2 3 5
4 5 1
```

【输出样例】

```
11
```

【分析】

本题与差分约束中的《糖果》这题完全一样，但是差分约束由于使用SPFA求最长路和正环，因此效率不稳定，而如果使用强连通分量来做那么时间复杂度是稳定$O(n)$的，而使用强连通分量求解需要具备一些性质。

本题中的边权只有0和1，强连通分量中的任意两点相互间都可以到达，那么如果一个强连通分量中有一条边的权值大于0，也就是为1的话，那么在这个强连通分量中一定存在正环，也就是无解。

如果有解的话那么每个强连通分量中所有的边权一定都是0，那么我们以标号从大到小的顺序枚举强连通分量，这样满足拓扑序，然后更新dis即可。最后求解所有dis之和的时候注意需要将每个强连通分量的dis乘上这个分量中点的数量cnt。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

typedef long long LL;
const int N = 100010, M = 600010;
int e[M], ne[M], d[M], h[N], hs[N], idx;
int dfn[N], low[N], timestamp;
int stk[N], id[N], cnt[N], dis[N], scc_cnt;
bool in_stk[N];
int n, m, tt;

```

```cpp
void add(int h[], int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

void tarjan(int u)
{
    dfn[u] = low[u] = ++timestamp;
    stk[++tt] = u, in_stk[u] = true;
    for (int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if (!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }
        else if (in_stk[j]) low[u] = min(low[u], dfn[j]);
    }
    if (dfn[u] == low[u])
    {
        int t;
        scc_cnt++;
        do
        {
            t = stk[tt--];
            in_stk[t] = false;
            id[t] = scc_cnt;
            cnt[scc_cnt]++;
        } while (t != u);
    }
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    memset(hs, -1, sizeof hs);
    for (int i = 1; i <= n; i++) add(h, 0, i, 1);
    while (m--)
    {
        int t, a, b;

        cin >> t >> a >> b;
```

```
        if (t == 1) add(h, a, b, 0), add(h, b, a, 0);
        else if (t == 2) add(h, a, b, 1);
        else if (t == 3) add(h, b, a, 0);
        else if (t == 4) add(h, b, a, 1);
        else add(h, a, b, 0);
    }
    tarjan(0);//0可以走到其它所有点
    for (int u = 0; u <= n; u++)
        for (int i = h[u]; ~i; i = ne[i])
        {
            int j = e[i];
            if (id[u] == id[j] && d[i] > 0) { cout << -1 << endl; return 0; }
            else if (id[u] != id[j]) add(hs, id[u], id[j], d[i]);
        }
    for (int u = scc_cnt; u; u--)//因为满足拓扑序因此按顺序遍历一遍更新最长路即可
        for (int i = hs[u]; ~i; i = ne[i])
            dis[e[i]] = max(dis[e[i]], dis[u] + d[i]);
    LL res = 0;
    for (int i = 1; i <= scc_cnt; i++) res += (LL)dis[i] * cnt[i];
    cout << res << endl;
    return 0;
}
```