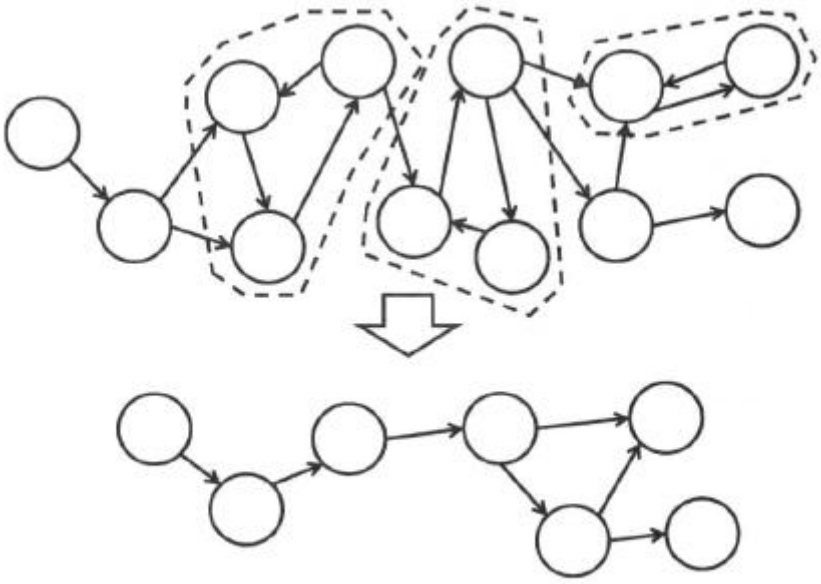
有向图强连通分量的 Tarjan 算法

一、定义

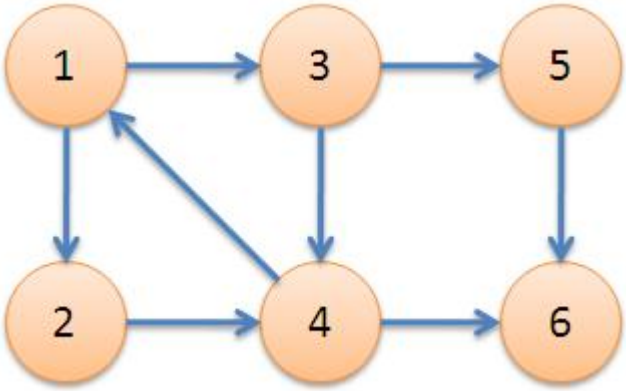
在**有向图 G** 中，如果两个顶点 u,v 间存在一条路径 u 到 v 的路径且也存在一条 v 到 u 的路径，则称这两个顶点 u,v 是**强连通的**(strongly connected)。如果有向图 G 的每两个顶点都强连通，称 G 是一个**强连通图**。有向非强连通图的**极大强连通子图**，称为**强连通分量**(strongly connected components)。若将有向图中的强连通分量都缩为一个点，则原图会形成一个 DAG（有向无环图）。

极大强连通子图：G 是一个极大强连通子图当且仅当 G 是一个强连通子图且不存在另一个强连通子图 G’使得 G 是 G’的真子集。



虚线包围的部分构成一个强连通分量

下图中，子图{1,2,3,4}为一个强连通分量，因为顶点 1,2,3,4 两两可达。{5},{6}也分别是两个强连通分量。



二、Tarjan 算法

Tarjan 算法是基于对图深度优先搜索(DFS)的算法，**每个强连通分量为搜索树中的一棵子树（的一部分）**。搜索时，把当前搜索树中未处理的结点加入一个栈，回溯时可以判断栈顶到栈中的结点是否为一个强连通分量。

DFS 过程中遇到的四种边：

树枝边：DFS 时经过的边，即 DFS 搜索树上的边

前向边：与 DFS 方向一致，从某个结点指向其某个子孙的边

后向边：与 DFS 方向相反，从某个结点指向其某个祖先的边

横叉边：从某个结点指向搜索树中另一子树中的某结点的边

定义 DFN(u)为结点 u 搜索的次序编号(时间戳)，Low(u)为 u 或 u 的子树（经过最多一条后向边或栈中横叉边）能够回溯到的最早的栈中结点的次序号。由定义可以得出:

Low(u)=Min

{

DFN(u),

Low(v),(u,v)为树枝边，u 为 v 的父结点

DFN(v),(u,v)为后向边或指向栈中结点的横叉边

}

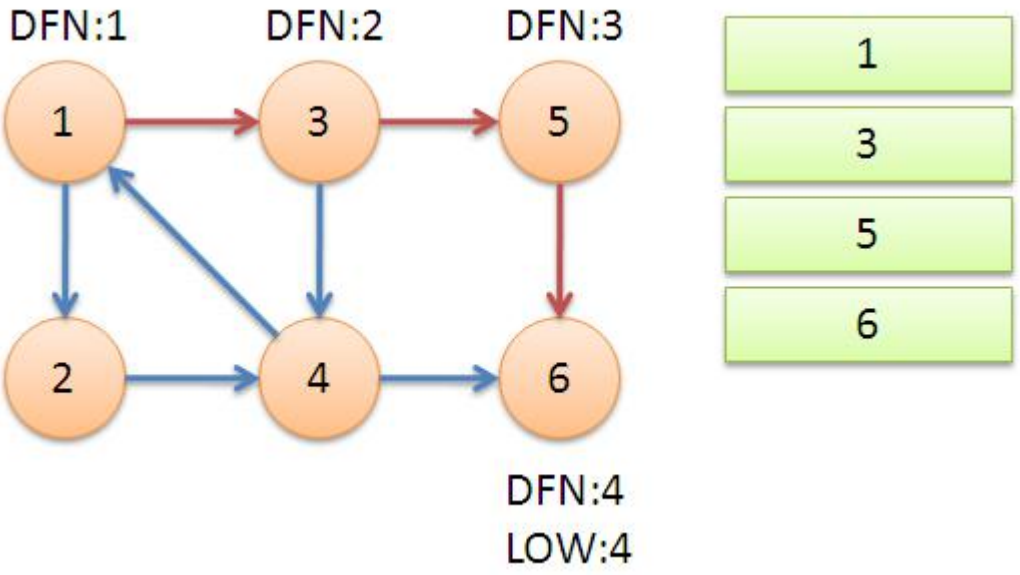
当结点 u 的搜索过程结束后，若 DFN(u)=Low(u)，则以 u 为根的搜索子树上所有还在栈中的结点是一个强连通分量。

算法伪代码如下:

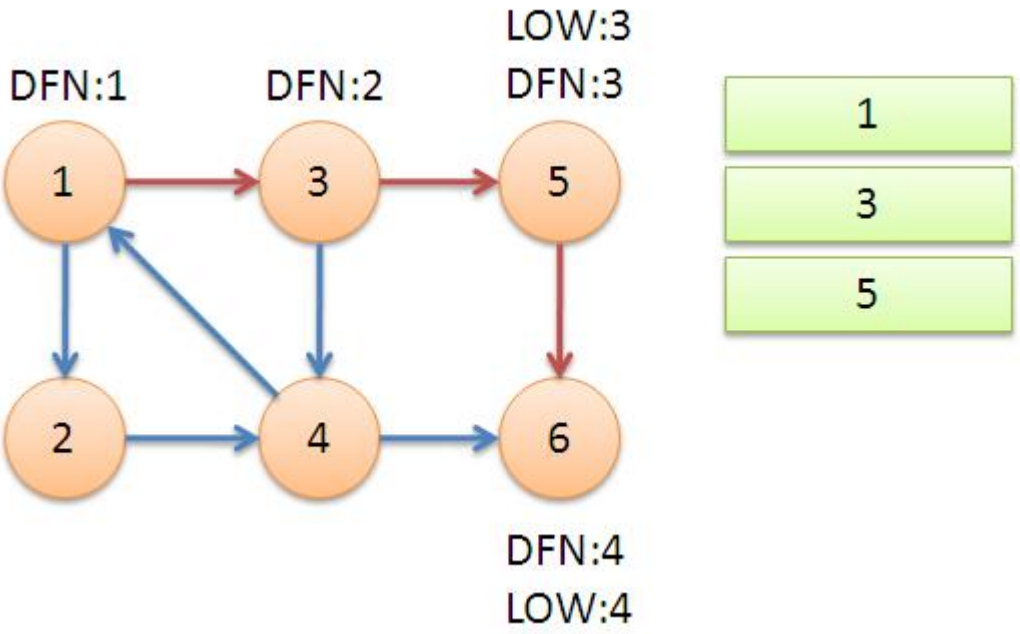
```
tarjan(u)
{
    DFN[u]=Low[u]=++Index                    // 为结点 u 设定次序编号和 Low 初值
    Stack.push(u)                            // 将结点 u 压入栈中
    for each (u, v) in E                     // 枚举每一条边
        if (v is not visted)                 // 如果结点 v 未被访问过
            tarjan(v)                        // 继续向下找
            Low[u] = min(Low[u], Low[v])
        else if (v in S)                     // 如果结点 v 还在栈内
            Low[u] = min(Low[u], DFN[v])
    if (DFN[u] == Low[u])                    // 如果结点 u 是强连通分量的根
        repeat
            v = S.pop                        // 将 v 退栈，为该强连通分量中一个顶点
            print v
        until (u== v)
}
```

三、算法流程演示

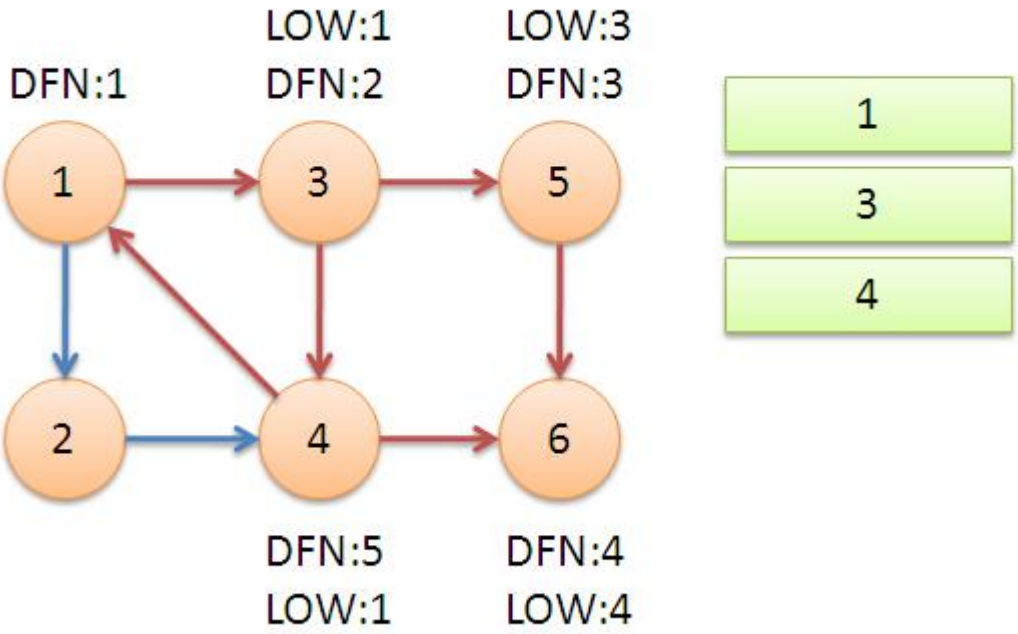
从结点 1 开始 DFS，把遍历到的结点加入栈中(1->3->5->6)。搜索到结点 u=6 时，DFN[6]=LOW[6]，找到了一个强连通分量。退栈到 u=v 为止，{6}为一个强连通分量。



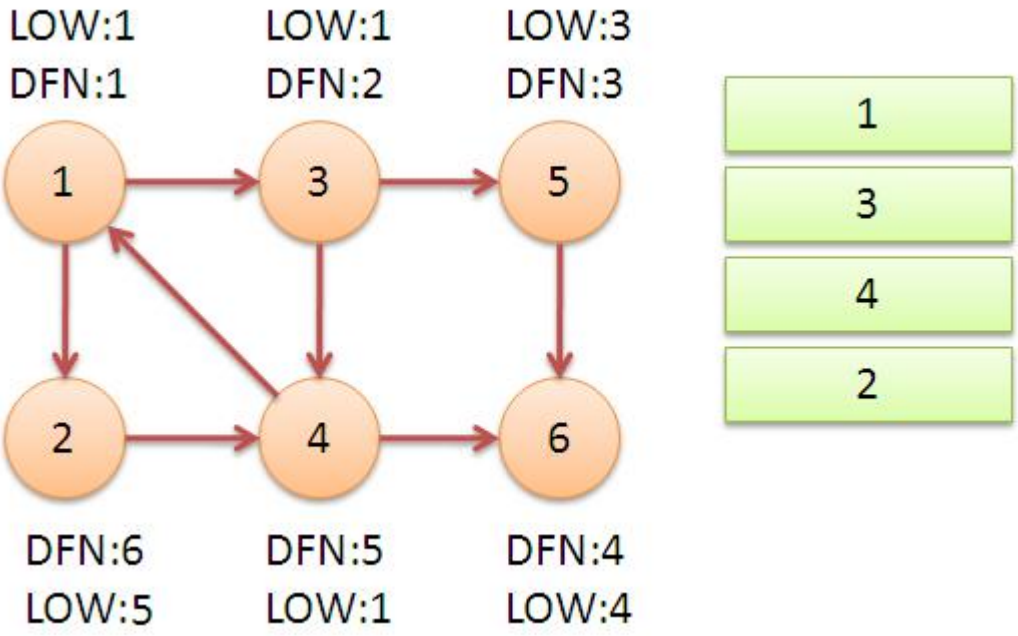
返回结点 5，发现 DFN[5]=LOW[5]，退栈后{5}为一个强连通分量。



返回结点 3，继续搜索到结点 4，把 4 加入栈。发现结点 4 向结点 1 有后向边，结点 1 还在栈中，所以 LOW[4]=1。结点 6 已经出栈，(4,6)是指向非栈中结点的横叉边，因此不更新 LOW[4]返回 3，(3,4)为树枝边，所以 LOW[3]=LOW[4]=1.



继续回到结点 1，最后访问结点 2。访问边(2,4)，4 还在栈中，所以 LOW[2]=DFN[4]=5. 返回 1 后，发现 DFN[1]=LOW[1]，把栈中结点全部取出，组成一个连通分量{1,3,4,2}.



至此，算法结束。经过该算法，求出了图中全部的三个强连通分量{1,3,4,2},{5},{6}.

可以发现，运行 Tarjan 算法的过程中，**每个顶点都被访问了一次，且只进出了一次栈，每条边也只被访问了一次，所以该算法的时间复杂度为 O(N+M).**

四、例题

题目大意：

Popular Cows （POJ No.2186）

每头牛都想成为牛群中的红人。给定 N 头牛的牛群和 M 个有序对(A, B)。(A, B)表示牛 A 认为牛 B 是红人。该关系具有传递性，所以如果牛 A 认为牛 B 是红人，牛 B 认为牛 C 是红人，那么牛 A 也认为牛 C 是红人。不过，给定的有序对中可能包含(A, B)和(B, C)，但不包含(A, C)。求被其他所有牛认为是红人的牛的总数。

限制条件

- $1 \leqslant N \leqslant 10000$
- $1 \leqslant M \leqslant 50000$
- $1 \leqslant A, B \leqslant N$

样例

输入

```
N = 3
M = 3
(A, B) = {(1, 2), (2, 1), (2, 3)}
```

输出

```
1（3号牛）
```

分析：

考虑以牛为顶点的有向图，对每个有序对(A, B)连一条从A到B的有向边。那么，被其他所有牛认为是红人的牛对应的顶点，也就是从其他所有顶点都可达的顶点。虽然这可以通过从每个顶点出发搜索求得，但总的复杂度却是$O(NM)$，是不可行的，必须要考虑更为高效的算法。

假设有两头牛A和B都被其他所有牛认为是红人。那么显然，A被B认为是红人，B也被A认为是红人，即存在一个包含A、B两个顶点的圈，或者说，A、B同属于一个强连通分量。反之，如果一头牛被其他所有牛认为是红人，那么其所属的强连通分量内的所有牛都被其他所有牛认为是红人。由此，我们把图进行强连通分量分解后，至多有一个强连通分量满足题目的条件。而按前面介绍的算法进行强连通分量分解时，我们还能够得到各个强连通分量拓扑排序后的顺序，唯一可能成为解的只有拓扑序最后的强连通分量。所以在最后，我们只要检查这个强连通分量是否从所有顶点可达就好了。该算法的复杂度为$O(N+M)$，足以在时限内解决原题。

代码：

```cpp
#include <cstdio>

const int N = 1e4 + 3;
const int M = 5e4 + 3;
int n, m, times, sum, top, ans;
int d[N], num[N], dfn[N], low[N], bel[N], stk[N];
// num 表示这个强连通分量的点数，bel 表示这个点所属的强连通分量编号
int ecnt, adj[N], nxt[M], go[M];
bool ins[N];

```

```cpp
inline void addEdge(const int &u, const int &v) {
  nxt[++ecnt] = adj[u], adj[u] = ecnt, go[ecnt] = v;
}

inline void Tarjan(const int &u) {
  ins[u] = true, stk[++top] = u;
  low[u] = dfn[u] = ++times;
  for (int e = adj[u], v; e; e = nxt[e]) {
    if (!dfn[v = go[e]]) {
      Tarjan(v);
      if (low[v] < low[u]) low[u] = low[v];
    } else if (ins[v] && dfn[v] < low[u])
      low[u] = dfn[v];
  }
  if (dfn[u] == low[u]) {
    int v;
    num[bel[u] = ++sum] = 1, ins[u] = false;
    while (v = stk[top--], v != u) ++num[bel[v] = sum], ins[v] = false;
  }
}

int main() {
  scanf("%d%d", &n, &m);
  for (int i = 1; i <= m; ++i) {
    int a, b;
    scanf("%d%d", &a, &b);
    addEdge(b, a);              // a 认为 b 是红人则建 b->a 的边(与上面题解有所不同)
  }
  for (int i = 1; i <= n; ++i)
    if (!dfn[i]) Tarjan(i);
  for (int i = 1; i <= n; ++i)                  // 求缩点后新图中每个新点的入度
    for (int e = adj[i]; e; e = nxt[e])
      if (bel[i] != bel[go[e]]) ++d[bel[go[e]]];
  for (int i = 1; i <= sum; ++i) {
    if (!d[i] && !ans) ans = num[i];            // 只有入度为 0 的点才可能是答案
    else if (!d[i]) {                           // 若有不止一个点入度为 0 则无解
      ans = -1; break;
    }
  }
  if (ans == -1) puts("0");
  else printf("%d\n", ans);
  return 0;
}
```

练习题

POJ 3180 ; POJ 1236

上述内容参考：

1.https://www.byvoid.com/blog/scc-tarjan/

2.挑战程序设计竞赛(第二版)

图的割点、桥与双连通分量

一、定义

点连通度与边连通度：

在一个**无向连通图**中，如果有一个顶点集合 V，删除顶点集合 V，以及与 V 中顶点相连（至少有一端在 V 中）的所有边后，原图**不连通**，就称这个点集 V 为**割点集合**。

一个图的**点连通度**的定义为：最小割点集合中的顶点数。

类似的，如果有一个边集合，删除这个边集合以后，原图不连通，就称这个点集为**割边集合**。

一个图的**边连通度**的定义为：最小割边集合中的边数。

双连通图、割点与桥：

如果一个无向连通图的**点连通度大于 1**，则称该图是**点双连通的**(point biconnected)，简称双连通或重连通。一个图有**割点**，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为**割点**(cut point)，又叫关节点(articulation point)。一个图可能有多个割点。

如果一个无向连通图的**边连通度大于 1**，则称该图是**边双连通的**(edge biconnected)，简称双连通或重连通。一个图有**桥**，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为**桥**(bridge)，又叫关节边(articulation edge)。一个图可能有多个桥。

可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，均既可指点双连通，又可指边双连通。（但这并不意味着它们等价）

双连通分量（分支）：在图 G 的所有子图 G'中，如果 G'是双连通的，则称 G'为双连通子图。如果一个双连通子图 G'它不是任何一个双连通子图的真子集，则 G'为极大双连通子图。双连通分量(biconnected component)，或重连通分量，就是图的极大双连通子图。特殊的，点双连通分量又叫做块。

二、Tarjan 算法

与有向图求强连通分量的 Tarjan 算法类似，只需通过求 DFN 与 LOW 值来得出割点与桥。

对图深度优先搜索(DFS)，定义 DFN(u)为 u 在搜索树（以下简称为树）中被遍历到的次序号。定义 Low(u)为 u 或 u 子树中的结点经过最多一条后向边能追溯到的最早的树中结点次序号。

根据定义，则有：

Low(u) = Min

{

DFN(u)

DFN(v) (u,v)为后向边(返祖边) 等价于 DFN(v)<DFN(u)**且 v 不为 u 的父亲结点**

Low(v) (u,v)为树枝边(父子边)

}

一个顶点 u 是割点，当且仅当满足(1)或(2)：

(1) u 为树根，且 u 有多于一个子树。因为无向图 DFS 搜索树中不存在横叉边，所以若有多个子树，这些子树间不会有边相连。

(2) u 不为树根，且满足存在(u,v)为树枝边（即 u 为 v 在搜索树中的父亲），并使得 DFN(u)<=Low(v).（因为删去 u 后 v 以及 v 的子树不能到达 u 的其他子树以及祖先）

一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足 DFN(u)<Low(v).（因为 v 想要到达 u 的父亲必须经过(u,v)这条边，所以删去这条边，图不连通）

实现时，因为有重边的问题，所以需要将一条无向边拆为两条编号一样的有向边，用邻接表进行存储。在判断(u,v)是否为后向边时要注意是树枝边的反向边还是新的一条反向边。

三、求双连通分量

下面要分开讨论点双连通分量与边双连通分量的求法。

对于点双连通分量，实际上在求割点的过程中就能顺便把每个点双连通分支求量。建立一个栈，存储当前双连通分量，在搜索图时，每找到一条树枝边或后向边（非横叉边），就把这条边加入栈中。如果遇到某时满足 DFN(u)<=Low(v)，说明 u 是一个割点，同时把边从栈顶一个个取出，直到遇到了边(u,v)，取出的这些边与其相连的点，组成一个点双连通分量。割点可以属于多个点双连通分量，其余点和每条边只属于且属于一个点双连通分量。

对于边双连通分量，求法更为简单。只需在求出所有的桥以后，把桥边删除，原图变成了多个连通块，则每个连通块就是一个边双连通分量。桥不属于任何一个边双连通分量，其余的边和每个顶点都属于且只属于一个边双连通分量。可以用并查集实现。

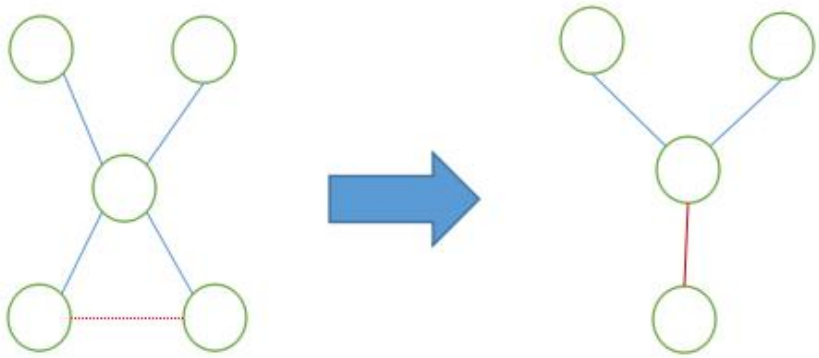
一个有桥的连通图，如何把它通过加边变成边双连通图？方法为首先求出所有的桥，然后删除这些桥边，剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶

点，再把桥边加回来，最后的这个图一定是一棵树，边连通度为 1。

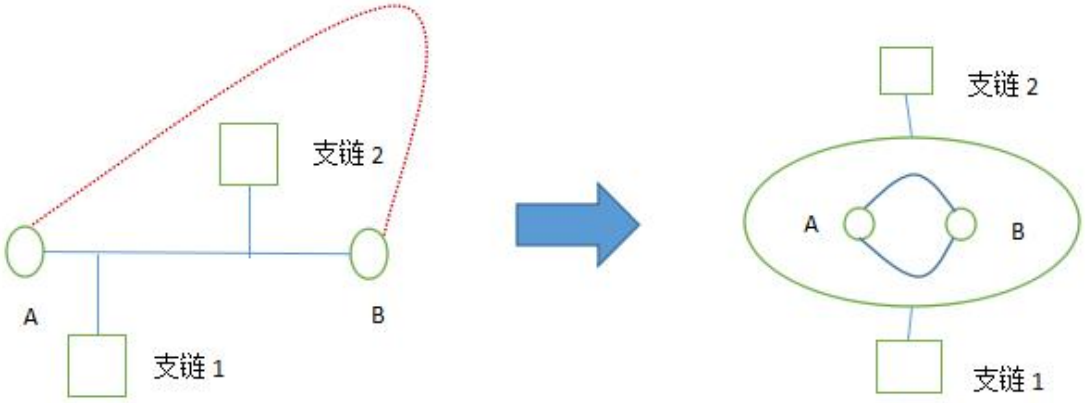
统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 leaf。则至少在树上添加(leaf+1)/2 条边，就能使树达到边二连通，所以至少添加的边数就是(leaf+1)/2。

证明：首先在一个边双里的点之间的连边是不会减少桥的数目的。因此先缩点。

考虑我们每次找两个叶子节点连边，但是如果随便找的话，连边之后重缩点可能会出现新的叶子。



注意到这种问题只出现在，两个叶子之间的路径上只有至多一条支链。



上图即为两条支链，连边后不会出现新叶子。

考虑至少有 4 个叶子的情况，选 4 个，设为 ABCD。首先考察 A 到 B 的路径和 C 到 D 的路径，如果有一条路径上其他边至少 2 条，则缩这两个点可以减少两个叶子。假设 A 到 B 和 C 到 D 的路径上都只有 1 条其他边（不能没有，否则不连通）。

如图所示，AD 路径上至少有两条其他边，于是建一条连接 AD 的边可以删去两个叶子。

边界情况：

叶子=1：不需要边

叶子=2：链，一条边

叶子=3：需要两条边

所以我们按照上述方式选取，可以保证不出现新叶子。

所以结论就是：叶子数=1 答案为 0 否则为(叶子数+1)/2.

四、例题

Redundant Paths (POJ 3177)

题目大意：

有 F 个牧场，现在一个牧群经常需要从一个牧场迁移到另一个牧场。奶牛们已经厌烦老是走同一条路，所以有必要再新修几条路，这样它们从一个牧场迁移到另一个牧场时总是可以选择至少两条独立的路。现在 F 个牧场的任何两个牧场之间已经至少有一条路了，奶牛们需要至少有两条。

给定现有的 R 条直接连接两个牧场的路，计算至少需要新修多少条直接连接两个牧场的路，使得任何两个牧场之间至少有两条独立的路。两条独立的路是指没有公共边的路，但可以经过同一个中间顶点。

1<=F<=5000 ; F-1<=R<=10000

分析：

题目要求任意两点间至少有两条没有公共边的路，也就是说所要求的图是一张边双连通图。根据上面所提到的，将一张有桥图通过加边变成边双连通图，至少要加(leaf+1)/2 条边。因此对于本题，我们求出所有桥，将桥删去后得出所有的边双连通分量，将它们缩为点后找

出叶子数，进而求出答案。

代码：

```cpp
#include <cstdio>

const int N = 5003;
const int M = 10005;
int n, m, du[N], anc[N];
int tt, from[N], low[N], dfn[N];
int ecnt = 1, adj[N], nxt[M * 2], go[M * 2], st[M * 2];
    // ecnt 初值为 1,那么 e=(u,v)的另一方向(v,u)的边的编号即为 e^1.
bool cut[M * 2];

int ufs(int u) {
  if (anc[u] == u) return u;
  return anc[u] = ufs(anc[u]);
}

inline void addEdge(int u, int v) {
  nxt[++ecnt] = adj[u], adj[u] = ecnt, go[ecnt] = v, st[ecnt] = u;
  nxt[++ecnt] = adj[v], adj[v] = ecnt, go[ecnt] = u, st[ecnt] = v;
}

void Tarjan(int u) {
  low[u] = dfn[u] = ++tt;
  for (int e = adj[u], v; e; e = nxt[e]) {
    if (e == (from[u] ^ 1)) continue;  // 树枝边需要跳过不予考虑
    if (!dfn[v = go[e]]) {
      from[v] = e;
      Tarjan(v);
      if (low[v] < low[u]) low[u] = low[v];
      if (low[v] > dfn[u]) cut[from[v]] = cut[from[v] ^ 1] = true;  // 找到桥
    } else if (dfn[v] < low[u]) {
      low[u] = dfn[v];
    }
  }
}

int main() {
  scanf("%d%d", &n, &m);
  for (int i = 1; i <= m; ++i) {
    int a, b;
    scanf("%d%d", &a, &b);
    addEdge(a, b);
```

```
42.  }
43.  Tarjan(1);
44.
45.  for (int i = 1; i <= n; ++i) anc[i] = i;
46.  for (int i = 2; i <= ecnt; i += 2)   // 用并查集实现缩点
47.    if (!cut[i]) anc[ufs(go[i])] = ufs(st[i]);
48.  for (int i = 2; i <= ecnt; i += 2)   // 桥的两端是不同边双分量
49.    if (cut[i]) ++du[ufs(go[i])], ++du[ufs(st[i])];
50.
51.  int ans = 0;
52.  for (int i = 1; i <= n; ++i)
53.    if (ufs(i) == i && du[i] == 1)
54.      ++ans;
55.  printf("%d\n", (ans + 1) / 2);
56.  return 0;
57. }
```

练习题

POJ 1523 ; POJ 2942

上述内容参考：

1.https://www.byvoid.com/blog/biconnect/

2-SAT

一、定义

给定一个布尔方程，判断是否存在一组布尔变量的取值方案，使得整个方程值为真的问题，被称为布尔方程的可满足性问题(SAT).SAT 问题是 NP 完全的，但对于一些特殊形式的 SAT 问题我们可以有效求解。

我们将下面这种布尔方程称为合取范式：

$(a \vee b \vee c \vee \cdots) \wedge (d \vee e \vee f \vee \cdots) \wedge \cdots$

其中 a,$b,c,\cdots$ 称为文字，它是一个布尔变量或其否定。像 $(a \vee b \vee c \vee \cdots)$ 这样用 $\vee$ 连接的部分称为子句。如果合取范式的每个子句中的文字个数都不超过两个，那么对应的 SAT 问题又称为 2-SAT 问题。

2-SAT 的一些例子：

$(a \vee b) \wedge (\neg a)$
$(a \vee \neg b) \wedge (\neg a \vee c) \wedge (b \vee \neg c)$
$(a \vee b) \wedge (\neg a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg b)$

二、解法

对于给定的 2-SAT 问题，首先利用 $\Rightarrow$ (蕴含)将每个子句 $(a \vee b)$ 改写成等价形式 $(\neg a \Rightarrow b \wedge \neg b \Rightarrow a)$.这样原布尔公式就变成了把 $(a \Rightarrow b)$ 形式的布尔公式用 $\wedge$ 连接起来的形式。

对每个布尔变量 x 构造两个顶点分别代表 x 与 $\neg x$.以 $\Rightarrow$ 关系为边建立有向图。若在此图中 a 点能到达 b 点，就表示 a 为真时 b 也一定为真。因此该图中同一个强连通分量中所含的所有变量的布尔值均相同。

若存在某个变量 x ,代表其 x 与 $\neg x$ 的两个顶点在同一个强连通分量中，则原布尔表达式的值无法为真。反之若不存在这样的变量，那么我们先将原图中所有的强连通分量缩为一个点，构出一个新图，新图显然是一个拓扑图，我们求出它的一个拓扑序。那么对于每个变量 x，让

x 所在的强连通分量（新图中的点）的拓扑序在 $\neg x$ 所在的强连通分量之后 $\Leftrightarrow$ x 为真

就是使得原布尔表达式取值为真的一组合适布尔变量赋值。注意到 Tarjan 算法所求的强连通分量就是按拓扑序的逆序得出的，因此不需要真的缩点建新图求拓扑序，直接利用强连通分量的编号来当做顺序即可。

若布尔变量的个数为 n，子句的个数为 m，那么用 Tarjan 求解这个 2-SAT 问题的时间复杂度即为 $O(n+m)$.

三、证明

首先利用子句构出来的图具有**对称性**。即若存在 $a \to b$ 的边，则也一定存在 $\neg b \to \neg a$ 的边。这里的对称指的是逆否命题的对称。对于单独的 x 这种子句，我们构出来的边是 $\neg x \to x$，而它的对称边还是它自己。

显然这种对称性具有传递性，即若有 $a \to b, b \to c$ 则可得出 $a \to c$，且根据对称性我们也能得出 $\neg c \to \neg a$.也就是说若能从 a 点到达 b 点，则一定也能从 $\neg b$ 点到达 $\neg a$ 点。

有了对称性与传递性不难得出我们所求出的强连通分量也具有对称性。也就是若 a, b 在同一个强连通分量 S 里，则 $\neg a, \neg b$ 也肯定在同一强连通分量 S' 里。S 与 S' 是相互矛盾的，选了 S（也就是将 S 内的元素取值设为真），那么就不能选 S'.

考虑将强连通分量缩成点以后的新图，新图也具有对称性，即若新图中 $S1$ 能到达点 $S2$，则能推出 $S2'$ 能到达 $S1'$（可以根据原图中点的对称性与传递性推出）。也就是说 S 的后继与 S' 的前驱也是相互对称的（互为矛盾点）。同时也代表，选择了 $S1$，那么就要选择 $S2$，并且 $S1'$ 与 $S2'$ 都不能选择。

考虑若按拓扑序从前往后选择，将当前可选的连通分量 S 与 S 所能到达的所有后继 S_i 内的元素的取值置为真，并将它们的矛盾点 S' 与 S_i' 都置为不可选，根据对称性 S_i' 都是 S' 的前驱，因此这么选不会出现矛盾。这么做在大多数情况下是对的，但是对于这种情况：$\neg x \to x$，选择会出现矛盾，因此我们按从后往前的顺序，这样就能使得选择合法。

四、例题

Wedding(POJ3648)

题目大意：

有一对新人结婚，邀请了 n-1 对夫妇去参加婚礼。婚礼上所有人要坐在一张很长的桌子的两边。所有的夫妇（包括新郎新娘）两人不能坐在同一边。还有 m 对人，对于每对人(a,b)，a、b 两人不能同时坐在新郎一边，但可以同时坐在新娘这边或是分两边坐。如果存在一种可行的方案，输出与新娘同侧的人（任意一种方案即可）。否则输出无解。

n <= 1000 , m <= 10^5

分析：

由于所有的夫妇要分两边坐，所以要么丈夫坐在新娘同侧要么妻子坐在新娘同侧，将夫妇的位置安排看做变量，则它只有两个相互排斥的取值，也就是一个布尔变量。而另外的 m 对限制其实就是限制两个人必须有至少一人在新娘同侧，就是一种$(a \vee b)$的限制。每对限制都只涉及两个人，所有限制都要被满足（即用 ∧ 连接所有限制），因此这是个 2-SAT 问题。

考虑对每个夫妇建两个点 x1,x2，x1 表示妻子与新郎同侧，x2 则表示丈夫与新郎同侧。那么对于一对关系(a,b)，a 向 b’,b 向 a’各连一条边，表示若 a 与新郎同侧，那么 b 不能与新郎同侧，也就是 b’一定要与新郎同侧。反之亦然。输出时反过来，即若与新郎同侧的是妻子，那么与新娘同侧的（也就是需要输出的人）就是丈夫。

代码：

```
#include <cstdio>

const int N = 1005;
const int M = 100005;
int n, m, SCC, id[N * 2];
int tt, top, stk[N * 2], dfn[N * 2], low[N * 2];
int ecnt, adj[N * 2], nxt[M * 4], go[M * 4];
bool ins[N * 2];

inline int neg(int x) {
  if (x <= n) return x + n;
  return x - n;
}

inline void addEdge(int u, int v) {
  nxt[++ecnt] = adj[u], adj[u] = ecnt, go[ecnt] = v;
}

void Tarjan(int u) {
  dfn[u] = low[u] = ++tt;
  ins[u] = true, stk[++top] = u;
```

```
  for (int e = adj[u], v; e; e = nxt[e]) {
    if (!dfn[v = go[e]]) {
      Tarjan(v);
      if (low[v] < low[u]) low[u] = low[v];
    } else if (ins[v] && dfn[v] < low[u]) {
      low[u] = dfn[v];
    }
  }
  if (dfn[u] == low[u]) {
    int v;
    id[u] = ++SCC, ins[u] = false;
    while (v = stk[top--], v != u) id[v] = SCC, ins[v] = false;
  }
}

void init() {
  ecnt = SCC = tt = 0;
  for (int i = 1; i <= n * 2; ++i)
    dfn[i] = adj[i] = 0;
}

void buildGraph() {
  int u, v;
  char a, b;
  for (int i = 1; i <= m; ++i) {
    scanf("%d%c %d%c", &u, &a, &v, &b);
    ++u, ++v;
    if (a == 'h') u += n;    // [1...n]表示妻子，[n+1...2n]表示丈夫
    if (b == 'h') v += n;
    addEdge(u, neg(v));
    addEdge(v, neg(u));
  }
  addEdge(1, 1 + n);     // 限制与新郎同侧的一定是新郎
}

int main() {
  while (scanf("%d%d", &n, &m), n > 0 || m > 0) {
    init();
    buildGraph();
    for (int i = 1; i <= n * 2; ++i)
      if (!dfn[i]) Tarjan(i);

    bool flag = true;
    for (int i = 1; i <= n && flag; ++i)
```

```
66.       if (id[i] == id[i + n])
67.         flag = false;
68.
69.     if (!flag) {
70.       puts("bad luck");
71.       continue;
72.     }
73.     for (int i = 2; i <= n; ++i)
74.       printf("%d%c%c", i - 1, (id[i] > id[i + n]) ? 'w' : 'h', " \n"[i == n])
    ;   // 按一般定义，是判断(id[i] < id[i + n])，然后取 i 所对应的解，这里输出时要反过
    来，因此我们直接把判断条件给反过来
75.     if (n < 2) printf("\n");
76.   }
77.   return 0;
78. }
```

练习题

POJ 3678 ; POJ 2749

上述内容参考：

1.挑战程序设计竞赛

欧拉回路

一、定义

设$G=(V,E)$是一个图。

欧拉回路　图G中经过**每条边一次**并且**仅一次的回路**称作欧拉回路。

欧拉路径　图G中经过每条边一次并且仅一次的路径称作欧拉路径。

欧拉图　存在**欧拉回路**的图称为欧拉图。

半欧拉图　存在欧拉路径但不存在欧拉回路的图称为半欧拉图。

二、性质与定理

在以下讨论中，假设图G不存在孤立点（度为0）；否则，先将所有孤立点从图中删除。显然，这样做并不会影响图G中欧拉回路的存在性。

我们经常需要判定一个图是否为欧拉图（或半欧拉图），并且找出一条欧拉回路（或欧拉路径）。对于无向图有如下结论：

定理1　无向图G为欧拉图，当且仅当G为连通图且所有顶点的度为偶数。

证明　**必要性**。设图G的一条欧拉回路为C。由于C经过图G的每一条边，而图G没有孤立点，所以C也经过图G的每一个顶点，G为连通图成立。而对于图G的任意一个顶点v，C经过v时都是从一条边进入，从另一条边离开，因此C经过v的关联边的次数为偶数。又由于C不重复地经过了图G的每一条边，因此v的度为偶数。

充分性。假设图G中不存在回路，而G是连通图，故G一定是树，那么有$|E|=|V|-1$。由于图G所有顶点的度为偶数而且不含孤立点，那么图G的每一个顶点的度至少为2。由握手定理，有$|E|=\frac{1}{2}\sum_{v\in V}d(v)\geq|V|$，与假设相矛盾。故图$G$中一定存在回路。设图$G$中边数最多的一条简单回路(边没有重复出现)为

$$C=\{e_1=(v_0,v_1),e_2=(v_1,v_2),\ldots,e_m=(v_{m-1},v_0)\}$$

下面证明回路C是图G的欧拉回路。

假设C不是欧拉回路，则C中至少含有一个点v_k，该点的度大于C经过该点的关联边的次数。令$v_0'=v_k$，从v_0'出发有一条不属于C的边$e_1'=(v_0',v_1')$。若$v_1'=v_0'$，则顶点v_0'自

身构成一个环，可以将其加入C中形成一个更大的回路；否则，若$v_1' \neq v_0'$，由于v_1'的度为偶数，而C中经过v_1'的关联边的次数也是偶数，所以必然存在一条不属于C的边$e_2' = (v_1', v_2')$。依此类推，存在不属于C的边$e_3' = (v_2', v_3'), \ldots, e_k' = (v_{k-1}', v_0')$。故$C' = \{e_1', e_2', \ldots, e_k'\}$是一条新的回路，将其加入$C$中可以形成一个更大的回路，这与$C$是图$G$的最大回路的假设相矛盾。故$C$是图$G$的欧拉回路。

由定理 1 可以立即得到一个用于判定半欧拉图的推论：

推论 1　无向图G为半欧拉图，当且仅当G为连通图且除了两个顶点的度为奇数之外，其它所有顶点的度为偶数。

证明　必要性。设图G的一条欧拉路径为

$$P = \{e_1 = (v_0, v_1), e_2 = (v_1, v_2), \ldots, e_m = (v_{m-1}, v_m)\}$$

由于P经过图G的每一条边，而图G没有孤立点，所以P也经过图G的每一个顶点，G为连通图成立。对于顶点v_0，P进入v_0的次数比离开v_0的次数少 1；对于顶点v_m，P进入v_m的次数比离开v_m的次数多 1：故v_0和v_m的度为奇数。而对于其它任意一个顶点$v_k (v_k \neq v_0, v_k \neq v_m)$，$P$进入$v_k$的次数等于离开$v_k$的次数，故$v_k$的度为偶数。

充分性。设v_1, v_2是图G中唯一的两个度为奇数的顶点。给图G加上一条虚拟边$e_0 = (v_1, v_2)$得到图G'，则图G'的每一个顶点度均为偶数，故图G'中存在欧拉回路C'。从C'中删去e_0得到一条从v_1到v_2的路径P，P即为图G的欧拉路径。

对于有向图，可以得到类似的结论：

定理 2　有向图G为欧拉图，当且仅当G的基图[1]连通，且所有顶点的入度等于出度。

推论 2　有向图G为半欧拉图，当且仅当G的基图连通，且存在顶点u的入度比出度大 1、v的入度比出度小 1，其它所有顶点的入度等于出度。

这两个结论的证明与定理 1 和推论 1 的证明方法类似，这里不再赘述。

注意到定理 1 的证明是构造性的，可以利用它来寻找欧拉回路。下面以无向图为例，介绍求欧拉回路的算法。

首先给出以下两个性质：

---

[1] 忽略有向图所有边的方向，得到的无向图称为该有向图的基图。

性质 1　设 C 是欧拉图 G 中的一个简单回路，将 C 中的边从图 G 中删去得到一个新的图 G'，则 G' 的每一个极大连通子图都有一条欧拉回路。

证明　若 G 为无向图，则图 G' 的各顶点的度为偶数；若 G 为有向图，则图 G' 的各顶点的入度等于出度。

性质 2　设 C_1、C_2 是图 G 的两个没有公共边，但有至少一个公共顶点的简单回路，我们可以将它们合并成一个新的简单回路 C'。

证明　只需按如图 3 所示的方式合并。

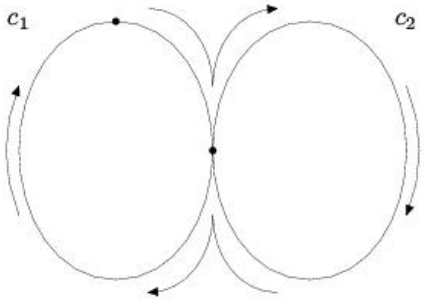


图 3

由此可以得到以下求欧拉图 G 的欧拉回路的算法：

1　在图 G 中任意找一个回路 C；

2　将图 G 中属于回路 C 的边删除；

3　在残留图的各极大连通子图中分别寻找欧拉回路；

4　将各极大连通子图的欧拉回路合并到 C 中得到图 G 的欧拉回路。

该算法的伪代码如下：

```
Procedure Euler-circuit ( start );
Begin
  For 顶点 start 的每个邻接点 v  Do
  If 边 (start, v) 未被标记 Then Begin
     将边 (start, v) 作上标记;
     将边 (v, start) 作上标记;                    //1
     Euler-circuit ( v );
     将边 (start, v) 加入栈 S;
```

```
  End;
End;
```

最后依次取出栈 S 每一条边而得到图 G 的欧拉回路。(也就是边出栈序的逆序)

由于该算法执行过程中每条边最多访问两次，因此该算法的时间复杂度为 $O(|E|)$ 。

如果图 G 是有向图，我们仍然可以使用以上算法，只需将标记有//1 的行删去即可。

三、例题

欧拉回路(UOJ 117)

题目大意：

给定一张图(有向或无向)，请你找出图中的一个欧拉回路。

分析：

直接套用模板即可。

代码：

```cpp
#include <cstdio>
#include <vector>

const int N = 100000 + 10, E = 10 * N;

int type, n, m;

int adj[N], to[E], next[E];

void link(int a, int b) {
  static int cnt = 2;
  to[cnt] = b;
  next[cnt] = adj[a];
  adj[a] = cnt++;
}

bool flag[E];

std::vector<int> ans;

void dfs(int a) {
  for (int &i = adj[a]; i; i = next[i]) {
    int b = to[i], c = (type == 1 ? (i / 2) : (i - 1));
```

```
24.     bool sig = i & 1;
25.     if (flag[c]) continue;
26.     flag[c] = true;
27.     dfs(b);
28.     if (type == 1) ans.push_back(sig ? -c : c); else ans.push_back(c);
29.   }
30. }
31. 
32. int main() {
33.   scanf("%d%d%d", &type, &n, &m);
34.   static int in[N], out[N];
35.   for (int i = m; i--;) {
36.     int a, b;
37.     scanf("%d%d", &a, &b);
38.     link(a, b);
39.     ++out[a], ++in[b];
40.     if (type == 1) link(b, a);
41.   }
42.   if (type == 1) {
43.     for (int i = 1; i <= n; ++i) if ((in[i] + out[i]) & 1) return puts("NO"),
    0;
44.   } else {
45.     for (int i = 1; i <= n; ++i) if (in[i] != out[i]) return puts("NO"), 0;

46.   }
47.   for (int i = 1; i <= n; ++i) {
48.     if (adj[i]) {
49.       dfs(i);
50.       break;
51.     }
52.   }
53.   if (ans.size() != m) return puts("NO"), 0;
54.   puts("YES");
55.   for (int i = m - 1; i >= 0; --i) printf("%d ", ans[i]);
56.   return 0;
57. }
```


上述内容参考：

1.2009 年中国国家集训队论文集，欧拉回路性质与应用探究，仇荣琦