POJ1144

题目描述（POJ1144）：电话公司正在建立一个新的电话网络，每个地方都有一个电话交换机（编号为 1～N）。线路是双向的，并且总是将两个地方连接在一起，在每个地方，线路都终止于电话交换机。从每个地方都可以通过线路到达其他地方，但不需要直接连接，可以进行多次交换。有时在某个地方发生故障，会导致交换机无法运行。在这种情况下，除了无法到达失败的地方，还可能导致其他地方无法相互连接。这个地方（发生故障的地方）是至关重要的。请写程序来查找所有关键位置的数量。

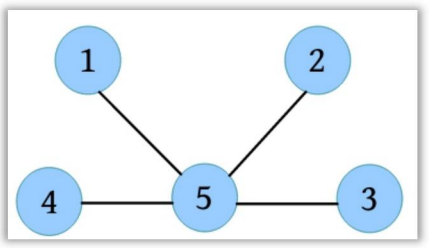
输入：输入包含多个测试用例。每个测试用例都描述一个网络。每个测试用例的第 1 行都是 N（N<100）。接下来最多 N 行中的每一行都包含一个地点的编号，后面是该地方可以直达的地点的编号。每个测试用例都以一条仅包含 0 的行结束。N=0 时输入结束，不处理。

输出：对每个测试用例，都单行输出关键位置的数量。

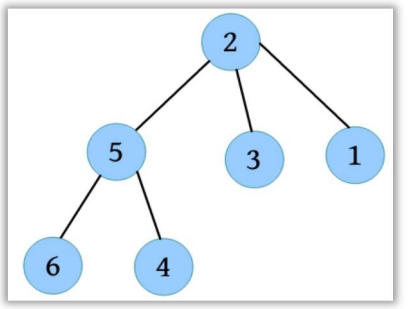
输入样例	输出样例
5 5 1 2 3 4 0 6 2 1 3 5 4 6 2 0 0	1 2

题解：

• 输入样例 1，构建的图如下图所示。在该图中有 1 个关键点 5。



• 输入样例 2，构建的图如下图所示。在该图中有两个关键点，分别是 2 和 5。



本题比较简单，就是求割点数，利用 Tarjan 算法求解即可。

算法代码：

```
void tarjan(int u,int root){//求割点
   dfn[u]=low[u]=++num;
   int flag=0;
   for(int i=head[u];i;i=e[i].next){
      int v=e[i].to;
      if(!dfn[v]){
         tarjan(v,u);
         low[u]=min(low[u],low[v]);
         if(low[v]>=dfn[u]){
            flag++;
            if(u!=root||flag>1)//u 不是根或者 u 是根但至少有两个子节点满足条件
               cut[u]=true;
         }
      }
      else
         low[u]=min(low[u],dfn[v]);
   }
}

for(int i=1;i<=n;i++)//统计割点的数量
   if(cut[i])
      ans++;
```

POJ3352

题目描述（POJ3352）：热带岛屿负责道路的人们想修理和升级岛上各个旅游景点之间的道路。道路本身也很有趣，它们从不在交叉路口汇合，而是通过桥梁和隧道相互交叉或相互通过。通过这种方式，每条道路都在两个特定的旅游景点之间运行，这样游客就不会迷失。不幸的是，当建筑公司在特定道路上工作时，该道路在任何一个方向都无法使用。如果在两个旅游景点之间无法同行，则即使建筑公司在任何特定时间只在一条道路上工作，也可能出现问题。

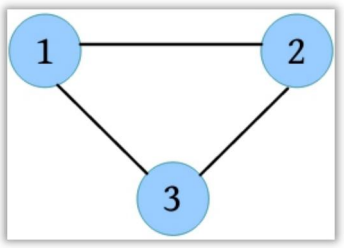
道路部门已经决定在景点之间建造新的道路，以便在最终配置中，如果任何一条道路正在建设，则仍然可以使用剩余的道路在任意两个旅游景点之间旅行。我们的任务是找到所需的最少数量的新道路。

输入：输入的第 1 行将包括正整数 n（3≤n≤1000）和 r（2≤r≤1000），其中 n 是旅游景点的数量，r 是道路的数量。旅游景点的编号为 1～n。以下 r 行中的每一行都将由两个整数 v 和 w 组成，表示在 v 和 w 的景点之间存在道路。请注意，道路是双向的，在任何两个旅游景点之间最多有一条道路。此外，在目前的配置中，可以在任意两个旅游景点之间旅行。

输出：单行输出需要添加的最少道路数量。

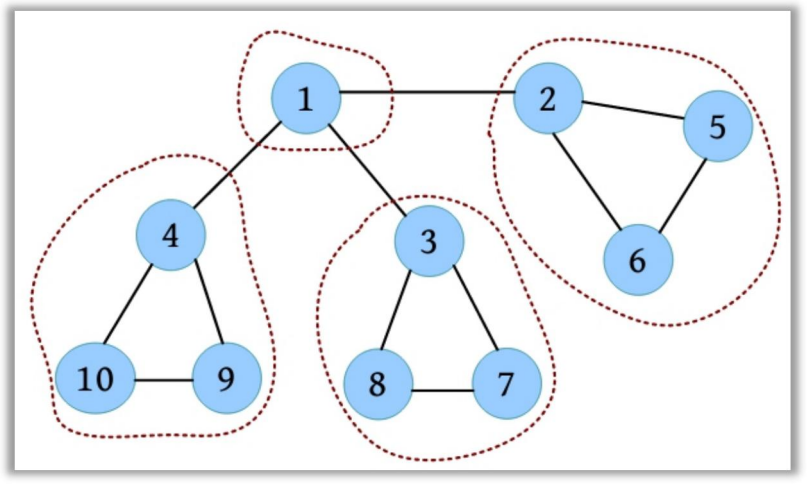
输入样例	输出样例
10 12 1 2 1 3 1 4 2 5 2 6 5 6 3 7 3 8 7 8 4 9 4 10 9 10 3 3 1 2 2 3 1 3	2 0

题解：输入样例 2，构建的图如下图所示。不需要添加加新的道路，就可以保证在一条道路维修时，可以通过其他道路到达任何一个景点。因此至少需要添加 0 条边。

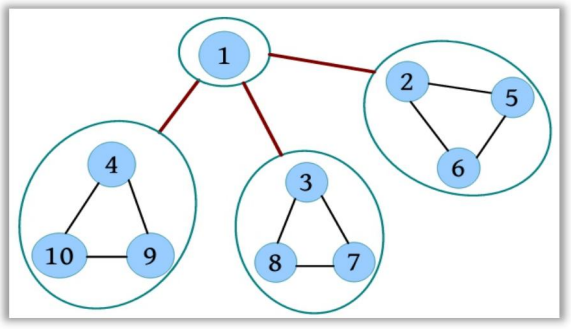


如何求解至少添加多少条边呢？

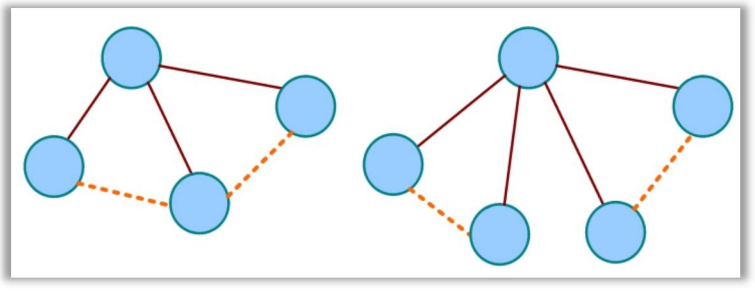
（1）如果在无向图中不存在桥，则称它为边双连通图。如果节点在一个边双连通分量中，则不需要添加边。因此需要求解边双连通分量。在边双连通分量之间需要添加边。输入样例 1，其边双连通分量如下图所示。



（2）将每个连通分量都缩成一个点，如下图所示。



（3）求解需要添加的新路数量。如果度为 1 的节点数为 k，则至少添加(k+1)/2 条边。例如，对 3 个度为 1 的节点至少需要加两条边，对 4 个度为 1 的节点至少需要加两条边。



• 为什么要统计叶子（度为 1 的节点）呢？

连通分量缩点后得到一个棵树，在树中任意两个点间添加一条边都会产生一个回路。在两个叶子之间添加一条边，则叶子和一些分支节点一起构成一个回路。而在分支节点之间添加一条边，产生的回路不会包含叶子。因此通过连接叶子可以添加最少的边，使每个节点都在回路中。

• 为什么添加的边数为(k+1)/2？

实际上，如果度为 1 的点数为偶数 k，那么直接两两添加一条边即可，即 k/2；如果为奇数，则在 k-1 点两两添加一条边，在最后一个点再添加一条边，即(k-1)/2+1=(k+1)/2。k 为偶数时，k/2=(k+1)/2，因此统一为(k+1)/2。

1. 算法设计

（1）先运行 Tarjan 算法，求解边双连通分量。

（2）缩点。检查每个节点 u 的每个邻接点 v，若 low[u]!=low[v]，则将这个连通分量点 low[u]的度加 1，degree[low[u]]++，同一个连通分量中的节点 low[]相同。

（3）统计度为 1 的点的个数为 leaf，添加的最少边数为(leaf+1)/2。

2. 算法实现

```
void tarjan(int u,int fa){
   dfn[u]=low[u]=++num;
   for(int i=head[u];i;i=e[i].next){
      int v=e[i].to;
      if(v==fa)
         continue;
      if(!dfn[v]){
         tarjan(v,u);
         low[u]=min(low[u],low[v]);
      }
      else
         low[u]=min(low[u],dfn[v]);
   }
}
//缩点并统计叶子数，输出答案
for(int u=1;u<=n;u++)
   for(int i=head[u];i;i=e[i].next){
      int v=e[i].to;
      if(low[u]!=low[v])
         degree[low[u]]++;
   }
int leaf=0;
for(int i=1;i<=n;i++)//统计叶子数
   if(degree[i]==1)
      leaf++;
cout<<(leaf+1)/2<<endl;
```

POJ2553

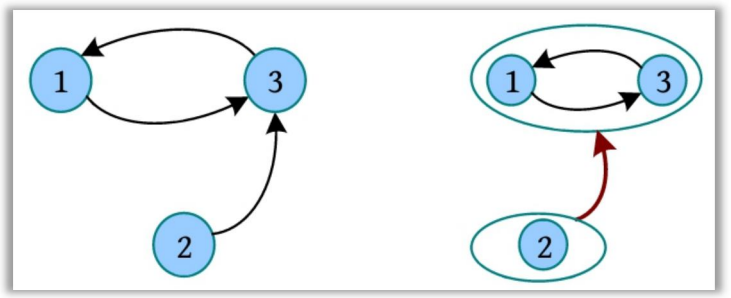
题目描述（POJ2553）：对于有向图 G 中任意一个节点 v，如果节点 v 可以到达节点 w，那么节点 w 都可以到达节点 v，那么节点 v 是一个 sink 节点。图 G 的底部是由图 G 中所有的 sink 节点构成的，请按顺序输出图 G 底部的所有 sink 节点，如果没有 sink 节点，则输出一个空行。

输入：输入包含几个测试用例，每个测试用例都对应一个有向图 G。每个测试用例都以整数 v（1≤v≤5000）开始，表示图 G 的节点数，节点编号为 1～v。接下来是非负整数 e，然后是 e 对节点编号 v_1, w_1 ,..., v_e, w_e，其中(v_i , w_i) 表示一条边。在最后一个测试用例后跟着一个 0。

输出：单行输出图底部的所有 sink 节点。如果没有，则输出一个空行。

输入样例	输出样例
3 3 1 3 2 3 3 1 2 1 1 2 0	1 3 2

题解：输入样例 1，构建的图如下图所示。图中的 sink 节点为 1 和 3。



求解强连通分量，并对强连通分量缩点，计算缩点的出度，出度为 0 的强连通分量中的所有节点均为 sink 节点。

1. 算法设计

（1）先采用 Tarjan 算法求解有向图的强连通分量，标记强连通分量号。

（2）检查每个节点 u 的每个邻接点 v，若强连通分量不同，则将 u 的连通分量号的出度加 1。

（3）检查每个节点 u，若其连通分量号的出度为 0，则输出该节点。

2. 算法实现

```
void tarjan(int u){//求强连通分量
   low[u]=dfn[u]=++num;
   ins[u]=true;
   s.push(u);
   for(int i=head[u];i;i=e[i].next){
      int v=e[i].to;
      if(!dfn[v]){
         tarjan(v);
         low[u]=min(low[u],low[v]);
      }
      else if(ins[v])
         low[u]=min(low[u],dfn[v]);
   }
   if(low[u]==dfn[u]){
      int v;
      do{
         v=s.top();
         s.pop();
         belong[v]=id;
         ins[v]=false;
      }while(v!=u);
      id++;
   }
}
//执行 Tarjan 算法，缩点并统计出度，输出出度为 0 的节点
for(int i=1;i<=n;i++)
   if(!dfn[i])
      tarjan(i);
for(int u=1;u<=n;u++)
   for(int i=head[u];i;i=e[i].next){
      int v=e[i].to;
      if(belong[u]!=belong[v])
         out[belong[u]]++;
   }
int flag=1;
for(int i=1;i<=n;i++)
   if(!out[belong[i]]){
      if(flag)//在第 1 个数前不加空格
         flag=0;
      else
         cout<<" ";
      cout<<i;
```

POJ1236

题目描述（POJ1236）：许多学校连接到计算机网络，这些学校之间已达成协议：每所学校都有一份学校名单，其中包括分发软件的学校（接收学校）。注意，即使学校 B 出现在学校 A 的分发列表中，学校 A 也不一定出现在学校 B 的列表中。编写程序，计算必须收到新软件副本的最少学校数量，以便软件根据协议到达网络中的所有学校（子任务 1）。作为进一步的任务，希过将新软件的副本发送到任意学校，使该软件覆盖网络中的所有学校。为了实现这一目标，可能必须通过新成员扩展接收者列表。请计算必须进行的最小数量的扩展，以便发送新软件到任意学校，它将到达所有其他学校（子任务 2）。一个扩展表示将一个新成员引入一所学校的接收者名单。

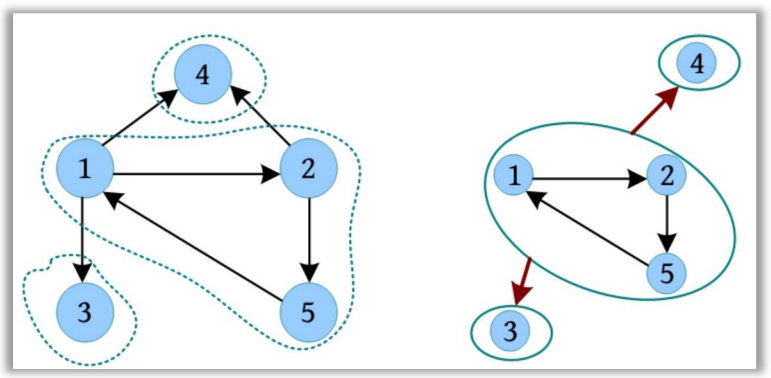
输入：第 1 行包含 1 个整数 N，表示网络中的学校数（2≤N≤100）。学校由前 N 个正整数标识。接下来的 N 行，每一行都描述了接收者列表，第 i+1 行包含学校 i 的接收者的标识符。每个列表都以 0 结尾。空列表在行中仅包含 0。

输出：输出包括两行。第 1 行应包含子任务 1 的解，第 2 行应包含子任务 2 的解。

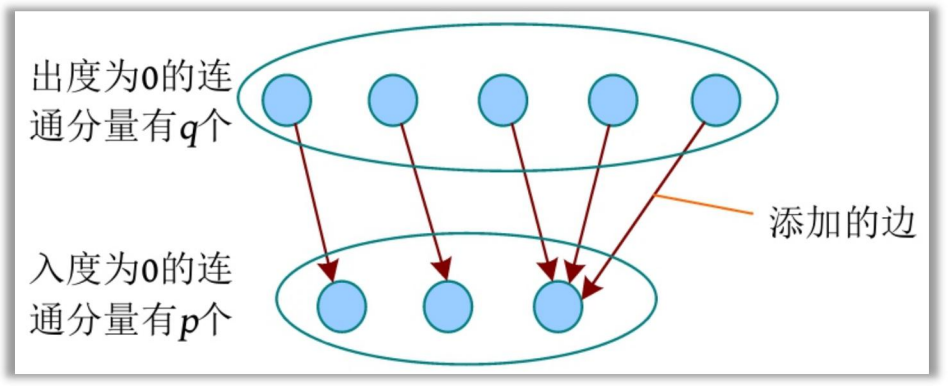
输入样例	输出样例
5 2 4 3 0 4 5 0 0 0 1 0	1 2

题解：求解过程如下。

（1）求解子任务 1：至少发送给多少个学校，才能让软件到达所有学校呢？实际上，求强连通分量并缩点后，每个入度为 0 的强连通分量都必须收到一个新软件副本。输入样例 1，构建的图如下图所示，其中包含 3 个强连通分量，缩点后入度为 0 的强连通分量有 1 个，至少发送给 1 个学校即可，即 1、2、5 中的任意一个学校。



（2）求解子任务 2：至少添加多少个接收关系，才能实现发送给任意一个学校，所有学校都能收到？也就是说，每个强连通分量都必须既有入度，又有出度。对入度为 0 的强连通分量，至少添加一个入度；对出度为 0 的强连通分量，至少添加一个出度。添加的边数为 max(p,q)，如下图所示。



特殊情况：若只有一个强连通分量，则至少分发给 1 个学校，需要添加的边数为 0。

1. 算法设计

（1）采用 Tarjan 算法求解强连通分量，标记连通分量号。

（2）检查每个节点 u 的每个邻接点 v，若连通分量号不同，则 u 连通分量号出度加 1，v 连通分量号入度加 1。

（3）统计入度为 0 的连通分量数 p 及出度为 0 的连通分量数 q，求 max(p,q)。

2. 算法实现

```
void tarjan(int u) {//求解有向图的强连通分量
   low[u]=dfn[u]=++num;
   vis[u]=true;
   s.push(u);
   for(int i=head[u];i;i=e[i].next){
      int v=e[i].to;
      if(!dfn[v]){
         tarjan(v);
         low[u]=min(low[u],low[v]);
      }
      else if(vis[v])
         low[u]=min(low[u],dfn[v]);
   }
   if(low[u]==dfn[u]){
      int v;
      id++;
      do{
         v=s.top();
         s.pop();
         belong[v]=id;
         vis[v]=false;
}
//执行 Tarjan 算法，缩点并统计输入和出度，求入度和出度为 0 的节点数
for(int i=1;i<=n;i++)
   if(!dfn[i])
      tarjan(i);
for(int u=1;u<=n;u++)
   for(int i=head[u];i;i=e[i].next){
      int v=e[i].to;
      if(belong[u]!=belong[v]){
         in[belong[v]]++;
         out[belong[u]]++;
      }
   }
if(id==1){//特殊情况判断
   cout<<1<<endl;
   cout<<0<<endl;
   return 0;
}
int ans1=0,ans2=0;
for(int i=1;i<=id;i++){
   if(!in[i])
      ans1++;
   if(!out[i])
      ans2++;
}
cout<<ans1<<endl;
cout<<max(ans1,ans2)<<endl;
```