# J题题解

## 题目

给你一个 n$n$ 个点，m$m$ 条边的无向图简单图。求割点数量，割边数量，极大点双连通分量数量，极大点双连通分量包含边数的最大值。

割点：在图中移除这个点后，存在一个点对在原图中连通在新图中不连通。

割边：在图中移除这条边后，存在一个点对在原图中连通在新图中不连通。

点双连通分量：原图的一个点数大于 11 的连通子图，不存在对于这个子图的割点。

极大点双连通分量：是点双连通分量，且任意新增一个点后不是点双连通分量。

不要问这题为什么又出现了，问就是no response

## Standard Input

第一行两个整数 n,m$n,m$，分别表示图的边数和点数。

接下来 m$m$ 行，每行两个整数 u,v$u,v$，表示一条边。

## Standard Output

一行四个整数，用空格分割，分别表示割点数量，割边数量，极大点双连通分量数量，极大点双连通分量包含边数的最大值。

## Samples

| Input | Output |
| --- | --- |
| `6 6 1 2 2 3 3 4 4 5 5 6 6 1` | `0 0 1 6` |
| `6 7 1 2 2 3 3 1 4 5 5 6 6 4 1 4` | `2 1 3 3` |

## Constraints

1\le n\le 10001≤$n$≤1000
0\le m\le \frac{n(n-1)}{2}0≤$m$≤2$n$($n$−1)
1\le u,v\le n1≤$u$,$v$≤$n$
保证无重边，无自环。

## Note

如果存在一条边 u,v$u,v$，那么 {u,v}{$u$,$v$} 也是一个点双连通分量。

| Problem ID | 2612 |
| --- | --- |
| Problem Title | tarjan |
| Time Limit | 1000 ms |
| Memory Limit | 256 MiB |
| Output Limit | 256 MiB |
| Source | 2021 UESTC ICPC Training for Graph |

# 题解

一眼题目：tarjan求割点割边，点双和点双内部的边数的数量。

tarjan是基于深度优先搜索算法，探究图的连通性问题，tarjan算法可以在线性时间内求出图的割点和桥，进一步可以求出无向图的点双，或者有向图的强连通分量，必经边和必经点。

## 求无向图的割点和桥

边没有方向的图就是无向图，下面介绍一下割点和桥的概念。

### 割点

若从图中删除节点 x 以及所有与 x 关联的边之后，图将被分成两个或两个以上的不相连的子图，那么称 x 为图的**割点**。

### 桥

若从图中删除边 e 之后，图将分裂成两个不相连的子图，那么称 e 为图的**桥**或**割边**。

### 时间戳

在深度优先搜索过程中遍历的先后顺序。用的$dfn[x]$表示

### 搜索树

在无向图中，我们以某一个节点 x 出发进行深度优先搜索，每一个节点只访问一次，所有被访问过的节点与边构成一棵树，我们可以称之为“无向连通图的搜索树”

### 追溯值

追溯值用来表示从当前节点 x 作为搜索树的根节点出发，能够访问到的所有节点中，时间戳最小的值即为low[x]。

### 无向图判定桥

在一张无向图中，判断边 e （其对应的两个节点分别为 u 与 v）是否为桥，需要其满足如下条件即可：dfn[u] < low[v]

它代表的是节点 u 被访问的时间，要优先于（小于）以下这些节点被访问的时间 —— low[v] 。

- 以节点 v 为根的搜索树中的所有节点
- 通过一条非搜索树上的边，能够到达搜索树的所有节点（在追溯值内容中有所解释）

是不是上面的两个条件很眼熟？对，其实就是前文提到的追溯值 —— low[v]。

### 求割点

首先选定一个根节点，从该根节点开始遍历整个图（使用DFS）。

对于根节点，判断是不是割点很简单——计算其子树数量，如果有2棵即以上的子树，就是割点。因为如果去掉这个点，这两棵子树就不能互相到达。

对于非根节点，判断是不是割点就有些麻烦了。我们维护两个数组$dfn[]$和$low[]$，$dfn[u]$表示顶点u第几个被（首次）访问，$low[u]$表示顶点u及其子树中的点，通过非父子边（回边），能够回溯到的最早的点（dfn最小）的dfn值（但不能通过连接u与其父节点的边）。对于边$(u,v)$，如果$low[v] \geq dfn[u]$，此时u就是割点。

但这里也出现一个问题：怎么计算$low[u]$。

假设当前顶点为u，则默认$low[u] = dfn[u]$，即最早只能回溯到自身。

有一条边$(u,v)$，如果v未访问过，继续DFS，DFS完之后，$low[u] = min(low[u], low[v])$;

如果v访问过（且u不是v的父亲），就不需要继续DFS了，一定有$dfn[v] < dfn[u]$，
$low[u] = min(low[u], dfn[v])$。

### 求点双

在Tarjan过程中维护一个栈，每次Tarjan到一个结点就将该结点入栈，回溯时若目标结点low值不小于当前结点dfn值就出栈直到目标结点（目标结点也出栈），将出栈结点和当前结点存入DCC

具体实现参考代码。

# 代码

```
// #include <bits/stdc++.h>
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cstdlib>
#include <vector>
using namespace std;
#define MAXN 2000005
int head[MAXN],ver[MAXN],nxt[MAXN],tot=1;
inline void add(int x,int y){ver[++tot]=y;nxt[tot]=head[x];head[x]=tot;}
int dfn[MAXN],low[MAXN],num;
bool bridge[MAXN];
int n,m;
int ans1,ans2,ans4;
void tarjan(int x,int  in_edge)
{
    dfn[x]=low[x]=++num;
    for(int i=head[x];i;i=nxt[i])
    {
        int y=ver[i];
        if(!dfn[y])
        {
            tarjan(y,i);
            low[x]=min(low[x],low[y]);
            if(low[y]>dfn[x])
                bridge[i]=bridge[i^1]=1;
        }
        else if(i!=(in_edge^1))
```

```
        {
            low[x]=min(low[x],dfn[y]);
        }
    }
}

int root;
bool cut[MAXN];
void tarjan2(int x)
{
    dfn[x]=low[x]=++num;
    int flag=0;
    for(int i=head[x];i;i=nxt[i])
    {
        int y=ver[i];
        if(!dfn[y])
        {
            tarjan2(y);
            low[x]=min(low[x],low[y]);
            if(low[y]>=dfn[x])
            {
                flag++;
                if(x!=root||flag>1)cut[x]=1;
            }
        }
        else low[x]=min(low[x],dfn[y]);
    }
}
int sta[MAXN],top,cnt;
vector<int>dcc[MAXN];
void tarjan3(int x)
{
    dfn[x]=low[x]=++num;
    sta[++top]=x;
    if(x==root&&head[x]==0)
    {
        // dcc[++cnt].push_back(x);
        return ;
    }
    int flag=0;
    for(int i=head[x];i;i=nxt[i])
    {
        int y=ver[i];
        if(!dfn[y])
        {
            tarjan3(y);
            low[x]=min(low[x],low[y]);
            if(low[y]>=dfn[x])
            {
                flag++;
                if(x!=root||flag>1)cut[x]=1;
                cnt++;
                int z;
                do{
                    z=sta[top--];
                    dcc[cnt].push_back(z);
                }while(z!=y);
                dcc[cnt].push_back(x);
```

```
            }
        }
        else low[x]=min(low[x],dfn[y]);
    }
}
bool vis[MAXN],is_dcc[MAXN];
int edge_cnt;
void dfs(int x) {
        vis[x] = 1;
        for(int i=head[x];i;i=nxt[i])
        {
            int v=ver[i];
            if (!is_dcc[v]) continue;
            ++edge_cnt;
            if (!vis[v])
                dfs(v);
        }
    }
int belong[MAXN],res[MAXN];
int main()
{
    // freopen("run.in","r",stdin);
    // freopen("opt.out","w",stdout);
    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;i++)
    {
        int x,y;
        scanf("%d %d",&x,&y);
        add(x,y),add(y,x);
    }
    num=0;
    for(int i=1;i<=n;i++)
    {
        if(!dfn[i])
            tarjan(i,0);
    }
    for(int i=2;i<tot;i+=2)
    {
        if(bridge[i])
        {
            // printf("%d %d\n",ver[i^1],ver[i]);
            ans2++;//求割边
        }
    }
    memset(dfn,0,sizeof(dfn));
    memset(low,0,sizeof(low));
    num=0;ans1=0;
    for(int i=1;i<=n;i++)
    {
        if(!dfn[i]){
            root=i;
            tarjan2(i);
        }
    }
    // int rs1=0;
    for(int i=1;i<=n;i++)
    {
        // if(cut[i])cout<<"gedian   "<<i<<endl;
```

```
        if(cut[i])ans1++;
    }
    memset(dfn,0,sizeof(dfn));
    memset(low,0,sizeof(low));
    memset(cut,0,sizeof(cut));
    num=0;top=0;cnt=0;
    for(int i=1;i<=n;i++)
    {
        if(!dfn[i])
        {
            root=i;
            tarjan3(i);
        }
    }
    int x;
    for (int i = 1; i <= cnt; ++i)
    {
        for (int j = 0; j < dcc[i].size(); ++j)
        {
            x = dcc[i][j];
            is_dcc[x] = 1;
        }
        x = dcc[i][0];
        edge_cnt = 0;
        dfs(x);
        ans4 = max(ans4, edge_cnt/2);
        for (int j = 0; j < dcc[i].size(); ++j) {
            x = dcc[i][j];
            is_dcc[x] = vis[x] = 0;
        }
    }
    cout<<ans1<<" "<<ans2<<" "<<cnt<<" "<<ans4<<endl;
}
```