割点和桥

割点

> 对于一个无向图，如果把一个点删除后这个图的极大连通分量数增加了，那么这个点就是这个图的割点（又称割顶）。

如何实现？

如果我们尝试删除每个点，并且判断这个图的连通性，那么复杂度会特别高。所以要介绍一个常用的算法Tarjan

首先，我们上一个图：

很容易看出割点是 2，而且这个图仅有这一个割点。

首先，我们按照 DFS 序给他打上时间戳（访问的顺序）。

这些信息被我们保存在一个叫做 num 的数组中。

还需要另外一个数组 low，用它来存储不经过其父亲能到达的最小的时间戳。

例如 low[2] 的话是 1，low[5] 和 low[6] 是 3。

然后我们开始 DFS我们判断某个点是否是割点的根据是：对于某个顶点 u如果存在至少一个顶点 $v$$u$ 的儿子），使得 $low_v\ge num_u$即不能回到祖先，那么 u 点为割点。

另外，如果搜到了自己（在环中），如果它有两个及以上的儿子，那么它一定是割点了，如果只有一个儿子，那么把它删掉，不会有任何的影响。比如下面这个图，此处形成了一个环，从树上来讲它有2个儿子：

我们在访问 1 的儿子的时候，假设先 DFS 到了 2，然后标记用过，然后递归往下，来到了 4，4 又来到了 3，当递归回溯的时候，会发现 3 已经被访问过了，所以不是割点。

更新 low 的伪代码如下：

```
如果 v 是u 的儿子 low[u] = min(low[u], low[v]);
否则
low[u] = min(low[u], num[v]);
```

例题

洛谷 P3388【模板】割点（割顶）

题意：求割点

代码：

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e4+5,M=2e5+5;
int head[N],nxt[M],to[M],tot,cnt;
int dfn[N],low[N];
bool flag[N];
void addedge(int u,int v){
    nxt[++tot]=head[u];
    head[u]=tot;
    to[tot]=v;
}
void Tarjan(int u,int fa){
    dfn[u]=low[u]=++cnt;
    int child=0;
    for(int i=head[u];i;i=nxt[i]){
        int v=to[i];
        if(v==fa) continue;
        if(!dfn[v]){
            Tarjan(v,u);
            low[u]=min(low[u],low[v]);
            child++;
            if(low[v]>=dfn[u]&&u!=fa) flag[u]=1;//非根结点且子结点不能以不通过父结点的方式到达父结点的祖先结点
        }
        else low[u]=min(low[u],dfn[v]);
    }
    if(u==fa&&child>=2) flag[u]=1;//是根结点且子结点数大于等于 2
}
int main(){
    int n,m;
    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;i++){
        int x,y;
        scanf("%d %d",&x,&y);
        addedge(x,y);
        addedge(y,x);
    }
    for(int i=1;i<=n;i++){
        if(!dfn[i]) Tarjan(i,i);
    }
    int ans=0;
    for(int i=1;i<=n;i++)
        if(flag[i]) ans++;
    printf("%d\n",ans);
    for(int i=1;i<=n;i++)
        if(flag[i])
            printf("%d ",i);
    return 0;
```

```
}
```

POJ 2117 Electricity

题意：选定一个结点，使得删掉它之后的连通块数量最多

题解：对于根结点，删掉它之后新增连通块为其儿子数-1；对于非根结点的割点，删掉它之后新增的连通块等于其儿子数。

代码：

```
#include<cstdio>
#include<vector>
#include<cstring>
using namespace std;
const int N=1e4+5;
vector<int>adj[N];
int n,m;
int dfn[N],low[N],cnt;
int block[N];
void init(){
    memset(block,0,sizeof(block));
    memset(dfn,0,sizeof(dfn));
    memset(low,0,sizeof(low));
    for(int i=1;i<N;i++) adj[i].clear();
    cnt=0;
}
void addedge(int u,int v){
    adj[u].push_back(v);
    adj[v].push_back(u);
}
void Tarjan(int u,int fa){
    low[u]=dfn[u]=++cnt;
    int child=0;
    for(int i=0;i<adj[u].size();i++){
        int v=adj[u][i];
        if(v==fa) continue;
        if(!dfn[v]){
            child++;
            Tarjan(v,u);
            low[u]=min(low[u],low[v]);
            if(fa!=u&&low[v]>=dfn[u]) block[u]++;
        }
        else low[u]=min(low[u],dfn[v]);
    }
    if(fa==u) block[u]=child-1;
}
int main(){
    while(scanf("%d %d",&n,&m)&&(n||m)){
        init();
```

```
        for(int i=1;i<=m;i++){
            int x,y;
            scanf("%d %d",&x,&y);
            x++;y++;
            addedge(x,y);
        }
        int parts=0;
        for(int i=1;i<=n;i++){
            if(!dfn[i]){
                parts++;
                Tarjan(i,i);
            }
        }
        int maxx=0;
        for(int i=1;i<=n;i++) maxx=max(maxx,parts+block[i]);
        printf("%d\n",maxx);
    }
    return 0;
}
```