圆方树

算法简介

一种用于将图的问题转化为树上问题进行求解的算法。

算法实现

首先给出圆方树相关定义：

对无向图求双连通分量，然后给每个双连通分量建一个新点。

特别的，这里点双连通分量的定义是不存在割点的最大子图，然后只有一个点的图需要自行特殊处理。

每个双连通分量代表的方点向原图中属于该点双连通分量的点连一条边，得到一棵树，称为圆方树。

同时称双连通分量代表的点为方点，原图中的点为圆点。

具体实现稍微修改一下求点双连通分量的代码即可，记得由于点双连通分量最多有 $O(n)$ 个要开双倍数组。

```
vector<int> g[MAXN<<1];
int node_cnt;
int low[MAXN],dfs_id[MAXN],dfs_t,bcc_id[MAXN],bcc_cnt;
vector<int> bcc[MAXN];
stack<pair<int,int> >Stack;
bool iscut[MAXN];
void dfs(int u,int fa){
    low[u]=dfs_id[u]=++dfs_t;
    blk_sz++;
    int child=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        if(!dfs_id[v]){
            Stack.push(make_pair(u,v));
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfs_id[u]){
                iscut[u]=true;
                pair<int,int> temp;
                bcc[++bcc_cnt].clear();
                while(true){
                    temp=Stack.top();Stack.pop();
                    if(bcc_id[temp.first]!=bcc_cnt){
                        bcc_id[temp.first]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.first);
                    }
```

```
                    if(bcc_id[temp.second]!=bcc_cnt){
                        bcc_id[temp.second]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.second);
                    }
                    if(temp.first==u&&temp.second==v)
                    break;
                }
                node_cnt++;//就加了几行
                for(int node_id:bcc[bcc_cnt]){
                    g[node_cnt].push_back(node_id);
                    g[node_id].push_back(node_cnt);
                }
            }
            child++;
        }
        else if(dfs_id[v]<dfs_id[u]){
            Stack.push(make_pair(u,v));
            low[u]=min(low[u],dfs_id[v]);
        }
    }
    if(u==fa&&child<2)
    iscut[u]=false;
}
void find_bcc(int n){
    mem(dfs_id,0);
    mem(iscut,0);
    mem(bcc_id,0);
    dfs_t=bcc_cnt=0;
    node_cnt=n;
    _rep(i,1,n){
        if(!dfs_id[i])
        dfs(i,i);
    }
}
```

算法例题

例题一

洛谷p4630

题意

给定一个图，不保证连通。求有多少三元组 (s,c,f) 满足 s,c,f 互异且存在一条从 s 出发经过 c 到达 f 的简单路径。

题解

首先给定点双连通分量的一个性质，任取一个点双连通分量中互异的三个点 (s,c,f) 一定存在一条从 s 出发经过 c 到达 f 的简单路径。

该性质由来见 证明

然后又可以得出结论任取两个圆点 s,f 则满足条件的 c 正好是与 s,f 在圆方树的路径经过的方点相邻的圆点构成的集合(注意减去 s,f 本身)。

不妨设每个方点的点权为该点双连通分量的大小，每个圆点权值为 -1。则不难发现 s,f 对应的合法的 c 数量就是 s,f 在圆方树路径上的点权和。

考虑树形 dp 求解经过每个点的路径个数计算贡献即可，时间复杂度 $O(n+m)$

```
const int MAXN=1e5+5,MAXM=2e5+5;
struct Edge{
    int to,next;
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
vector<int> g[MAXN<<1];
int node_cnt,blk_sz,val[MAXN<<1];
int low[MAXN],dfs_id[MAXN],dfs_t,bcc_id[MAXN],bcc_cnt;
vector<int> bcc[MAXN];
stack<pair<int,int> >Stack;
bool iscut[MAXN];
void dfs(int u,int fa){
    low[u]=dfs_id[u]=++dfs_t;
    blk_sz++;
    int child=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        if(!dfs_id[v]){
            Stack.push(make_pair(u,v));
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfs_id[u]){
                iscut[u]=true;
                pair<int,int> temp;
                bcc[++bcc_cnt].clear();
                while(true){
                    temp=Stack.top();Stack.pop();
                    if(bcc_id[temp.first]!=bcc_cnt){
                        bcc_id[temp.first]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.first);
```

```
                    }
                    if(bcc_id[temp.second]!=bcc_cnt){
                        bcc_id[temp.second]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.second);
                    }
                    if(temp.first==u&&temp.second==v)
                    break;
                }
                val[++node_cnt]=bcc[bcc_cnt].size();
                for(int node_id:bcc[bcc_cnt]){
                    g[node_cnt].push_back(node_id);
                    g[node_id].push_back(node_cnt);
                }
            }
            child++;
        }
        else if(dfs_id[v]<dfs_id[u]){
            Stack.push(make_pair(u,v));
            low[u]=min(low[u],dfs_id[v]);
        }
    }
    if(u==fa&&child<2)
    iscut[u]=false;
}
int sz[MAXN<<1];
LL ans;
void dfs2(int u,int fa){
    if(val[u]==-1)sz[u]=1;
    for(int v:g[u]){
        if(v==fa)continue;
        dfs2(v,u);
        ans+=2LL*sz[u]*sz[v]*val[u];
        sz[u]+=sz[v];
    }
    ans+=2LL*sz[u]*(blk_sz-sz[u])*val[u];
}
void solve(int rt){
    blk_sz=0;
    dfs(rt,rt);
    dfs2(rt,rt);
}
void find_bcc(int n){
    mem(dfs_id,0);
    mem(iscut,0);
    mem(bcc_id,0);
    dfs_t=bcc_cnt=0;
    node_cnt=n;
    _rep(i,1,n)
    val[i]=-1;
    _rep(i,1,n){
```

```
        if(!dfs_id[i])
        solve(i);
    }
}
int main(){
    int n=read_int(),m=read_int();
    while(m--){
        int u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    find_bcc(n);
    enter(ans);
    return 0;
}
</hidden>
```