LCA

倍增

时间复杂度:预处理$nlogn$, 查询$logn$

$fa[i][j]$表示从i开始, 向上走2^j步所能走到的节点, $0 \le j \le logn$

$depth[i]$表示i节点的深度

哨兵：如果从i开始跳2^j步会跳过根节点, 那么$fa[i][j] = 0, depth[0] = 0$

步骤:

- 先将两个点跳到同一层
- 让两个点同时网上跳,一直跳到他们最近公共祖先的**下一层**

代码

```
#include<iostream>
#include<cstring>

using namespace std;

const int N = 40010,M = N * 2;

int n,m;
int h[N],e[M],ne[M],idx;
int depth[N],fa[N][16];
int q[N];

void add(int a,int b)
{
    e[idx] = b,ne[idx] = h[a],h[a] = idx++;
}

void bfs(int root)
{
    memset(depth,0x3f,sizeof depth);
    depth[0] = 0,depth[root] = 1;
    int hh = 0,tt = -1;
    q[++tt] = root;

    while(hh <= tt)
    {
        int t = q[hh++];
```

```
        for(int i = h[t]; ~i;i = ne[i])
        {
            int j = e[i];
            if(depth[j] > depth[t] + 1)
            {
                depth[j] = depth[t] + 1;
                q[++tt] = j;
                fa[j][0] = t;

                //倍增求fa
                for(int k = 1;k <= 15;k++)
                    fa[j][k] = fa[fa[j][k-1]][k-1];
            }
        }
    }
}


int lca(int a,int b)//不妨设a比较低
{
    if(depth[a] < depth[b]) swap(a,b);

    for(int k = 15;k >= 0;k--)
        if(depth[fa[a][k]] >= depth[b])
            a  = fa[a][k];
    if(a == b) return a;

    for(int k = 15; k >= 0;k--  )
        if(fa[a][k] != fa[b][k])
        {
            a = fa[a][k];
            b = fa[b][k];
        }
    return fa[a][0];
}


int main()
{
    cin>>n;
    int root = 0;
    memset(h,-1,sizeof h);

    for(int i = 0;i < n;i++)
    {
        int a,b;
        cin>>a>>b;
        if(b == -1) root = a;
        else add(a,b),add(b,a);
    }

    bfs(root);//求深度和fa

    cin>>m;
```

```
    while(m--)
    {
        int a,b;
        cin>>a>>b;
        int p = lca(a,b);
        if(p == a) puts("1");
        else if(p == b) puts("2");
        else puts("0");
    }

    return 0;
}
```

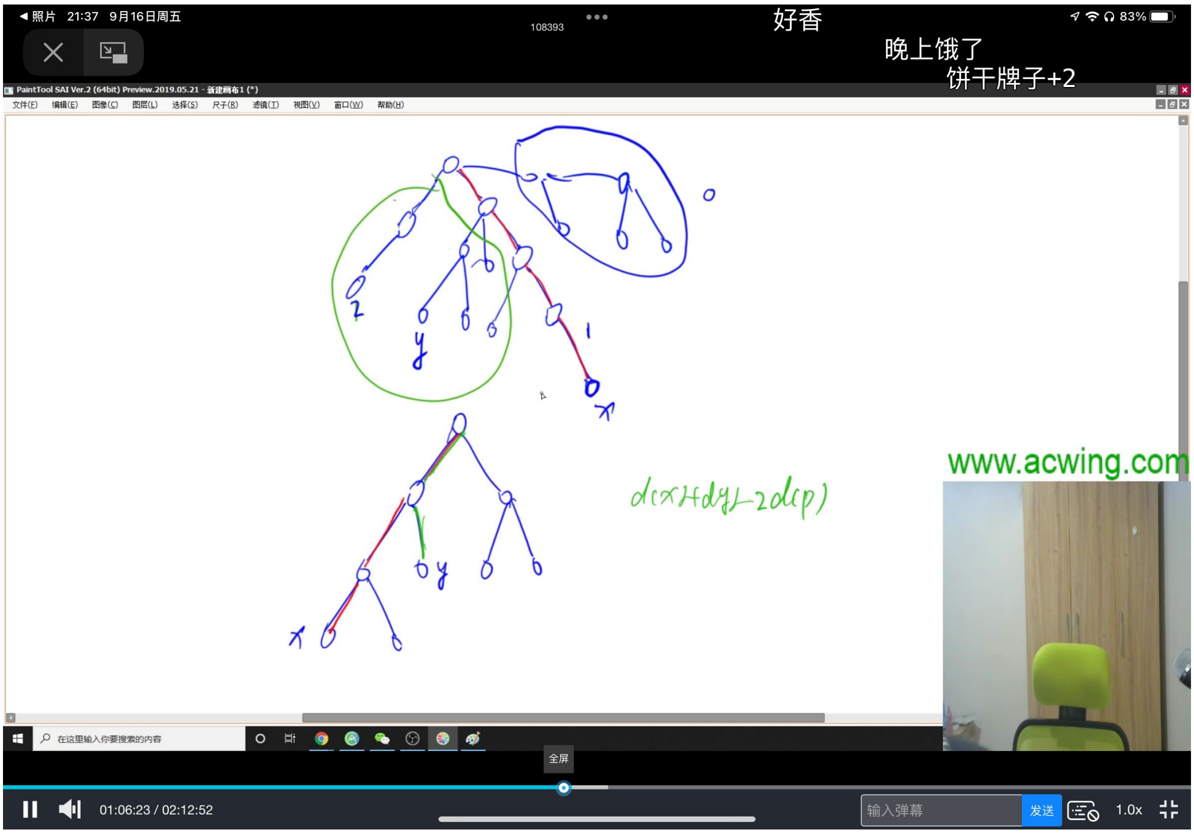
Tarjan-离线求LCA

时间复杂度:$O(n+m)$

步骤:

在dfs时,将所有的点分成三大类:

- [1]已经遍历过,且回溯过的点
- [2]正在搜索的分支
- [3]还未访问过的点



可发现`正在访问的点`的点与`已经访问`的点的LCA是已经访问过的节点的祖宗结点。

代码

> 树上两点之间的最短距离等于d[x] + d[y] - 2d[p] // d[u]为u到根节点距离,p是Lca(x,y)

```
#include<iostream>
#include<vector>
#include<cstring>

using namespace std;

typedef pair<int,int> PII;

const int N = 10010,M = N * 2;

int n,m;
int h[N],e[M],ne[M],w[M],idx;
int dist[N];
int p[N];
int ans[M];
int st[N];
vector<PII> query[N];// first存查询的另外一个点,second存查询编号

void add(int a,int b,int c)
{
    e[idx] = b,w[idx] = c,ne[idx] = h[a],h[a] = idx++;
}

int find(int x)
{
    if(p[x] != x) p[x] = find(p[x]);
```

```
    return p[x];
}


void dfs(int u,int fa)
{
    for(int i = h[u]; ~i ; i = ne[i])
    {
        int j = e[i];
        if(j == fa) continue;
        dist[j] = dist[u] + w[i];
        dfs(j,u);
    }
}

void tarjan(int u)
{
    st[u] = 1;//正在访问

    for(int i = h[u]; ~i ;i = ne[i])
    {
        int j = e[i];
        if(!st[j])
        {
            tarjan(j);
            p[j] = u;
        }
    }

    for(auto item : query[u])
    {
        int y = item.first,id = item.second;
        if(st[y] == 2)
        {
            int temp = find(y);
            ans[id] = dist[u] + dist[y] - dist[temp] * 2;
        }
    }

    st[u] = 2;
}


int main()
{
    cin>>n>>m;
    memset(h,-1,sizeof h);

    for (int i = 0; i < n - 1; i ++ )
    {
        int a, b, c;
        cin>>a>>b>>c;
        add(a, b, c), add(b, a, c);
    }
```

```
    for(int i = 0;i < m;i++)
    {
        int a,b;
        cin>>a>>b;
        if(a != b)
        {
            query[a].push_back({b,i});
            query[b].push_back({a,i});
        }
    }

    for(int i = 1;i <= n;i++) p[i] = i;

    dfs(1,-1);
    tarjan(1);

    for(int i = 0;i < m;i++) cout<<ans[i]<<endl;

    return 0;
}
```

树上差分:快速给树上每一条边加上1

```
d[x] += c, d[y] += c,d[p] -= 2c // p是x和y的lca,d[x]为以x为根节点,子结点权值和。
```

> 遍历每一条树边,要是当前树边被一个**非树边**连成环,则需要看两刀,被两条及以上连成环,则砍两刀不改变连通性,若是没有被**非树边**连成环,则可以砍**任意**一条非树边。

代码

```
#include<iostream>
#include<cstring>

using namespace std;

const int N = 100010,M = N * 2;

int n,m;
int h[N],e[M],ne[M],idx;
int depth[N],fa[N][17];
int d[N];//把边的差分赋给点，点的值代表的是某条边终点的覆盖情况
int q[N];
int ans;

void add(int a,int b)
{
```

```
    e[idx] = b,ne[idx] = h[a],h[a] = idx++;
}


void bfs()
{
    memset(depth,0x3f,sizeof depth);
    depth[0] = 0,depth[1] = 1;
    int hh = 0,tt = -1;
    q[++tt] = 1;

    while(hh <= tt)
    {
        int t = q[hh++];
        for (int i = h[t]; ~i; i = ne[i])
        {
            int j = e[i];
            if (depth[j] > depth[t] + 1)
            {
                depth[j] = depth[t] + 1;
                q[ ++ tt] = j;
                fa[j][0] = t;
                for (int k = 1; k <= 16; k ++ )
                    fa[j][k] = fa[fa[j][k - 1]][k - 1];
            }
        }
    }
}

int lca(int a,int b)
{
    if(depth[a] < depth[b]) swap(a,b);

    for(int k = 16;k >= 0;k--)
        if(depth[fa[a][k]] >= depth[b])
            a = fa[a][k];

    if(a == b) return a;
    for(int k = 16;k >= 0;k--)
        if(fa[a][k] != fa[b][k])
        {
            a = fa[a][k];
            b = fa[b][k];
        }
    return fa[a][0];
}

int dfs(int u,int father)
{
    int res = d[u];
    for(int i = h[u] ; ~i ;i = ne[i])
    {
        int j = e[i];
        if(j != father)
        {
```

```
            int s = dfs(j,u);
            if(s == 0) ans += m;
            else if(s == 1) ans ++;
            res += s;
        }
    }

    return res;
}

int main()
{
    cin>>n>>m;
    memset(h,-1,sizeof h);

    for(int i = 0;i < n - 1;i++)
    {
        int a,b;
        cin>>a>>b;
        add(a,b),add(b,a);
    }

    bfs();

    for(int i = 0;i < m;i++)
    {
        int a,b;
        cin>>a>>b;
        int p = lca(a,b);
        d[a]++,d[b]++,d[p] -= 2;
    }

    dfs(1,-1);
    cout<<ans<<endl;

    return 0;
}
```

有向图的强连通分量

对于一个**有向图**,联通分量:对于分量中的任意两点u、v,必然可以从u走到v,且从v走到u

强连通分量:极大联通分量

半连通:u->v **或** v->u

> 有向图 + 缩点 -> 有向无环图(DAG)

模板流程:

[1]遍历每个点,没被遍历过(!dfn[u])的话就遍历一下。[2.tarjan]每次先求一下时间戳,将当前点放到栈中,标记当前点在栈中,然后遍历一下当前点的所有邻边:**如果**当前点没被遍历过的话,就遍历一下,更新一下low[u],**否则**这个点如果在栈中,就更新一下low值。然后如果u是最上边的点,找出所有强连通分量中的点。

时间戳dfn、low、timestamp

scc_cnt---强联通分量个数

size---每个scc里点的个数

dout每个scc出度

id是强连通分量的编号

除了建边数组,所有数组中大小都是**点的数量**

```
int n,m;
int h[N],e[M],ne[M],idx;
int dfn[N],low[N],timestamp;//时间戳
int stk[N],top;
bool in_stk[N];
int id[N],scc_cnt,Size[N];//强联通分量编号、数量、大小
int dout[N];

void tarjan(int u)
{
    dfn[u] = low[u] = ++timestamp;
    stk[++top] = u,in_stk[u] = true;
    for(int i = h[u]; ~i ; i = ne[i])
    {
        int j = e[i];
        if(!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u],low[j]);
        }
        else if(in_stk[j]) low[u] = min(low[u],dfn[j]);
    }

    if(dfn[u] == low[u])
    {
        ++scc_cnt;
        int y;
        do{
            y = stk[top--];
            in_stk[y] = false;
            id[y] = scc_cnt;
            Size[scc_cnt]++;
        }while(y != u);
```

```
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    while (m -- )
    {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
    }

    for (int i = 1; i <= n; i ++ )
        if (!dfn[i])
            tarjan(i);
}
```

结论1

做完tarjan强连通分量编号**递减**的顺序一定是拓扑序

结论2

设强连通分量图的起点有p个,终点有q个。 只需从**p个起点**往终点走,就能将图中的所有点走完。还需要添加Max(p,q)条边,使图中所有点,都能到达任意一点。

(注意判断只有一个强联通分量时,不需要添边)

结论3

极大半联通子图等于强连通拓扑图中**最长**不分叉链,点数最多。

tarjan后建图+拓扑序dp

技巧判重边: `unordered_set<ll> S;// (u,v) -> u * 1000000 + v`

```
#include<cstring>
#include<iostream>
#include<unordered_set>

using namespace std;

typedef long long ll;

const int N = 100010,M = 2000010;

int n,m,mod;
```

```
int h[N],hs[N],e[M],ne[M],idx;
int dfn[N],low[N],timestamp;
int stk[N],top;
bool in_stk[N];
int id[N],scc_cnt,siz[N];
int f[N],g[N];//最大、方案数

void add(int h[],int a,int b)
{
    e[idx] = b,ne[idx] = h[a],h[a] = idx++;
}

void tarjan(int u)
{
    dfn[u] = low[u] = ++ timestamp;
    stk[++top] = u,in_stk[u] = true;

    for(int i = h[u]; ~i ; i = ne[i])
    {
        int j = e[i];
        if(!dfn[j])
        {
            tarjan(j);
            low[u] = min(low[u],low[j]);
        }
        else if(in_stk[j]) low[u] = min(low[u],dfn[j]);
    }

    if(dfn[u] == low[u])
    {
        ++scc_cnt;
        int y;
        do{
            y = stk[top--];
            in_stk[y] = false;
            id[y] = scc_cnt;
            siz[scc_cnt]++;
        }while(y != u);
    }
}
int main()
{
    memset(h,-1,sizeof h);
    memset(hs,-1,sizeof hs);

    cin>>n>>m>>mod;
    while(m--)
    {
        int a,b;
        cin>>a>>b;
        add(h,a,b);
    }

    for(int i = 1;i <= n;i++)
        if(!dfn[i])
```

```
            tarjan(i);

    unordered_set<ll> S;// (u,v) -> u * 1000000 + v
    for(int i = 1;i <= n;i++)
        for(int j = h[i]; ~j;j = ne[j])
        {
            int k = e[j];
            int a = id[i],b = id[k];
            ll hash = a * 1000000ll + b;
            if(a != b && !S.count(hash))
            {
                add(hs,a,b);
                S.insert(hash);
            }
        }


    for(int i = scc_cnt ; i ; i --)
    {
        if(!f[i])
        {
            f[i] = siz[i];
            g[i] = 1;
        }

        for(int j = hs[i] ; ~j ;j = ne[j])
        {
            int k = e[j];
            if(f[k] < f[i] + siz[k])
            {
                f[k] = f[i] + siz[k];
                g[k] = g[i];
            }
            else if(f[k] == f[i] + siz[k])
                g[k] = (g[k] + g[i]) % mod;
        }
    }

    int maxf = 0,sum = 0;

    for (int i = 1; i <= scc_cnt; i ++ )
        if (f[i] > maxf)
        {
            maxf = f[i];
            sum = g[i];
        }
        else if (f[i] == maxf) sum = (sum + g[i]) % mod;

    cout<<maxf<<endl<<sum;
    return 0;
}
```