图论-最近公共祖先

一、AcWing 1172. 祖孙询问（倍增法）

【题目描述】

给定一棵包含n个节点的有根无向树，节点编号互不相同，但不一定是$1 \sim n$。

有m个询问，每个询问给出了一对节点的编号x和y，询问x与y的祖孙关系。

【输入格式】

输入第一行包括一个整数n表示节点个数；

接下来n行每行一对整数a和b，表示a和b之间有一条无向边。如果b是-1，那么a就是树的根；

第$n+2$行是一个整数m表示询问个数；

接下来m行，每行两个不同的正整数x和y，表示一个询问。

【输出格式】

对于每一个询问，若x是y的祖先则输出1，若y是x的祖先则输出2，否则输出0。

【数据范围】

$1 \le n, m \le 4 \times 10^4$

$1 \le$ 每个节点的编号 $\le 4 \times 10^4$

【输入样例】

```
10
234 -1
12 234
13 234
14 234
15 234
16 234
17 234
18 234
19 234
233 19
5
```

```
234 233
233 12
233 13
233 15
233 19
```

【输出样例】

```
1
0
0
0
2
```

【分析】

本题为LCA的模板题，求LCA时我们首先需要预处理两个数组：

- $fa[i][j]$：表示从i开始，向上走2^j步所能走到的节点，若不存在则为0。
  求解方式：当$j=0$时，$fa[i][j]=i$的父节点；当$j \neq 0$时，$fa[i][j]=fa[fa[i][j-1]][j-1]$，即将$2^j$的路径分为两段$2^{j-1}$的路径。
- $depth[i]$：表示节点i的深度，规定根结点的深度为1，其子节点的深度为2，以此类推，即深度为到根结点的距离加一。

最近公共祖先的求解步骤：

1. 先将两个点跳到同一层。基于二进制拼凑的思想，假设节点x的深度大于y的深度，此时需要将x往上跳，那么我们从高位往低位枚举2^k，如果$depth[fa[x][k]] \geq depth[y]$那么就需要将$2^k$选进来，即需要向上跳（$x=fa[x][k]$），如果$depth[fa[x][k]] < depth[y]$，那么就不能跳，否则就跳到$y$的上面了。
2. 如果此时两个点不是同一个点，则让两个点同时往上跳，一直跳到他们的公共祖先的下一层。由于$fa[x][k]=fa[y][k]$时我们并不能判断这个点就是x,y的最近公共祖先，只能判断是公共祖先；而$fa[x][k] \neq fa[y][k]$时一定能判断这两个点还没跳到公共祖先，则可以一起往上跳。同样还是利用二进制的思想，从高到低枚举k，最后这两个点一定会跳到最近公共祖先的下一层，然后返回$fa[x][0]$即为答案。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
```

```
using namespace std;

const int N = 40010, M = N << 1;
int e[M], ne[M], h[N], idx;
int fa[N][16], depth[N];//2^16>40000
int n, m;

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

void bfs(int root)
{
    memset(depth, 0x3f, sizeof depth);
    depth[0] = 0, depth[root] = 1;
    queue<int> Q;
    Q.push(root);
    while (Q.size())
    {
        int t = Q.front();
        Q.pop();
        for (int i = h[t]; ~i; i = ne[i])
        {
            int j = e[i];
            if (depth[t] + 1 < depth[j])
            {
                depth[j] = depth[t] + 1;//子节点的深度等于父节点深度+1
                Q.push(j);
                fa[j][0] = t;//j向上跳2^0个节点就是跳到父节点t
                for (int k = 1; k <= 15; 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 = 15; k >= 0; k--)//将更下面的节点跳到与另一个节点同一层的位置
        if (depth[fa[a][k]] >= depth[b]) a = fa[a][k];

    if (a == b) return a;//如果在同一层时为同一点说明已经找到两点的最近公共祖
```

```
先
    for (int k = 15; k >= 0; k--)//否则两个点一起向上跳,跳至最近公共祖先的下
一个点
        if (fa[a][k] != fa[b][k]) a = fa[a][k], b = fa[b][k];
    return fa[a][0];//这个点的父节点即为两点的最近公共祖先
}

int main()
{
    cin >> n;
    memset(h, -1, sizeof h);
    int root = 0;
    while (n--)
    {
        int a, b;
        cin >> a >> b;
        if (b == -1) root = a;
        else add(a, b), add(b, a);
    }
    bfs(root);//预处理fa与depth
    cin >> m;
    while (m--)
    {
        int a, b;
        cin >> a >> b;
        int p = lca(a, b);//求出a,b两点的最近公共祖先p
        if (p == a) cout << 1 << endl;
        else if (p == b) cout << 2 << endl;
        else cout << 0 << endl;
    }
    return 0;
}
```

二、AcWing 1171. 距离（Tarjan）

【题目描述】

给出n个点的一棵树，多次询问两点之间的最短距离。

注意：

- 边是无向的。
- 所有节点的编号是$1, 2, \dots, n$。

【输入格式】

第一行为两个整数n和m。n表示点数，m表示询问次数；

接下来$n-1$行，每行三个整数x,y,k，表示点x和点y之间存在一条边长度为k；

再接下来m行，每行两个整数x,y，表示询问点x到点y的最短距离。

树中结点编号从$1\sim n$。

【输出格式】

共m行，对于每次询问，输出一行询问结果。

【数据范围】

$2\leq n\leq 10^4$

$1\leq m\leq 2\times 10^4$

$0<k\leq 100$

$1\leq x,y\leq n$

【输入样例1】

```
2 2
1 2 100
1 2
2 1
```

【输出样例1】

```
100
100
```

【输入样例2】

```
3 2
1 2 10
3 1 15
1 2
3 2
```

【输出样例2】

```
10
25
```

【分析】

由于树中任意两点间的路径是唯一的，因此两点间的距离即为最短距离。那么如何求出树中任意两点的距离呢？可以使用数组$dis[i]$表示节点i距离根节点的距离，设x,y的最近公共祖先为p，则x,y的距离即为$dis[x]+dis[y]-2*dis[p]$。

接下来就是Tarjan算法（离线求解LCA）：

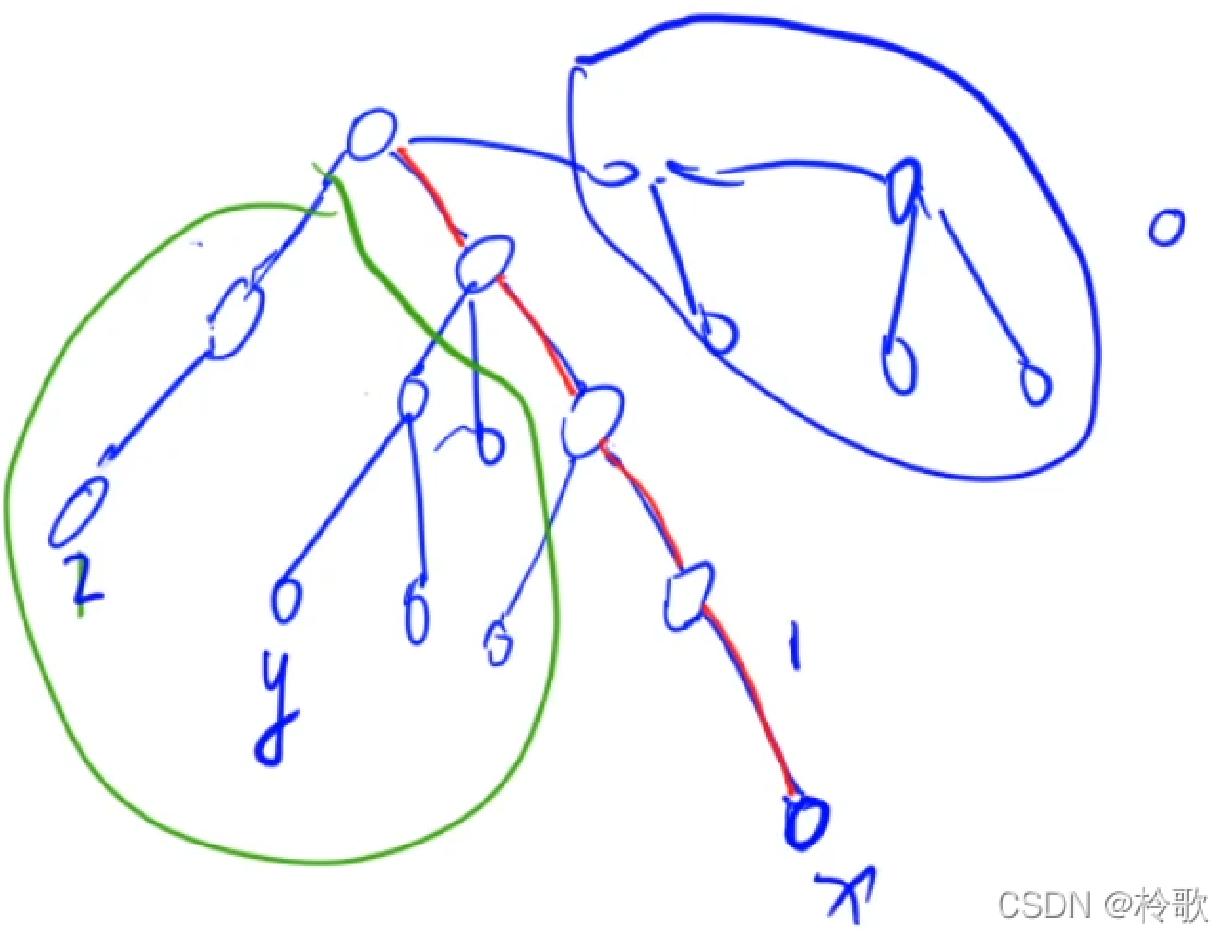
在线做法：读一个询问，处理一个，输出一个；

离线做法：读完全部询问，再全部处理完，再全部输出。

在深度优先遍历时，将所有点分成三大类

- 2号点：代表已经访问并结束回溯
- 1号点：代表正在访问
- 0号点：代表还没有访问过

其中所有2号点和正在搜索的1号点的路径中的其它点已经通过并查集合并成一个集合，2号点（绿色区域）所在连通块的根节点即为当前搜索路径（红色路径）上的其中一点，也就是2号点在这个根节点的子树上。那么这个根节点即为当前正在搜索的点x与根节点子树中任意一个2号点y的最近公共祖先。



因此使用Tarjan算法离线求解LCA的整体步骤如下：

1. 先求出所有点到根节点的距离$dis[]$，设x号点和y号点的最近公共祖先是p，则x和y的最近距离等于$dis[x] + dis[y] - 2 * dis[p]$
2. 在深度优先遍历1号点中的u点时，需要把与u有关的查询的另外一个点的最短距离进行计算并存储，最后把u点合并到上一节点的集合

【Tarjan写法$O(n+m)$】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;

typedef pair<int, int> PII;
const int N = 10010, M = N << 1;
int e[M], ne[M], d[M], h[N], idx;
int dis[N], st[N];//dis表示每个点到根节点的距离,st表示每个点的类型
int pre[N], res[N << 1];//res[i]表示第i个查询结果
int n, m;
vector<PII> query[N];//query[i]的第一个数为与i相关的查询的另一个点,第二个数为
```

```
第几个查询

void add(int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

int find(int k)
{
    if (pre[k] == k) return k;
    return pre[k] = find(pre[k]);
}

void dfs(int u, int fa)
{
    for (int i = h[u]; ~i; i = ne[i])
        if (e[i] != fa)
        {
            dis[e[i]] = dis[u] + d[i];
            dfs(e[i], u);
        }
}

void tarjan(int x)
{
    st[x] = 1;//当前正在搜索x
    for (int i = h[x]; ~i; i = ne[i])
        if (!st[e[i]])//如果x还未被搜索则进行搜索
        {
            tarjan(e[i]);
            pre[e[i]] = x;//搜索完后将子节点合并到父节点的集合中
        }
    for (auto t : query[x])//遍历与x有关的查询
    {
        int y = t.first, id = t.second;
        if (st[y] == 2)//如果另一个点已经搜索完且回溯了,则可以计算两点的LCA
            res[id] = dis[x] + dis[y] - 2 * dis[find(y)];
    }
    st[x] = 2;//已经搜索完x且已经回溯
}

int main()
{
```

```
    cin >> n >> m;
    memset(h, -1, sizeof h);
    for (int i = 1; i <= n; i++) pre[i] = i;
    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)//如果a==b则res[i]=0
        {
            query[a].push_back({ b, i });
            query[b].push_back({ a, i });
        }
    }
    dfs(1, -1);//设1号点为根节点,dfs求出其它点到1号点的距离
    tarjan(1);
    for (int i = 0; i < m; i++) cout << res[i] << endl;
    return 0;
}
```

【倍增写法（预处理$O(nlogn)$，查询$O(mlogn)$）】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;

const int N = 10010, M = N << 1;
int e[M], ne[M], d[M], h[N], idx;
int fa[N][14], depth[N], dis[N];
int n, m;

void add(int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

```

```
void dfs(int u, int fa)
{
    for (int i = h[u]; ~i; i = ne[i])
        if (e[i] != fa)
        {
            dis[e[i]] = dis[u] + d[i];
            dfs(e[i], u);
        }
}

void bfs(int root)
{
    memset(depth, 0x3f, sizeof depth);
    depth[0] = 0, depth[root] = 1;
    queue<int> Q;
    Q.push(root);
    while (Q.size())
    {
        int t = Q.front();
        Q.pop();
        for (int i = h[t]; ~i; i = ne[i])
        {
            int j = e[i];
            if (depth[t] + 1 < depth[j])
            {
                depth[j] = depth[t] + 1;
                Q.push(j);
                fa[j][0] = t;
                for (int k = 1; k <= 13; k++) fa[j][k] = fa[fa[j][k - 1]][k - 1];
            }
        }
    }
}

int lca(int a, int b)//倍增法在线求LCA
{
    if (depth[a] < depth[b]) swap(a, b);
    for (int k = 13; k >= 0; k--)
        if (depth[fa[a][k]] >= depth[b]) a = fa[a][k];
    if (a == b) return a;
    for (int k = 13; 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 >> 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);
    }
    dfs(1, -1);//设1号点为根节点
    bfs(1);//预处理fa与depth数组
    for (int i = 0; i < m; i++)
    {
        int a, b;
        cin >> a >> b;
        cout << dis[a] + dis[b] - 2 * dis[lca(a, b)] << endl;
    }
    return 0;
}
```

三、AcWing 356. 次小生成树

【题目描述】

给定一张N个点M条边的无向图，求无向图的严格次小生成树。

设最小生成树的边权之和为sum，严格次小生成树就是指边权之和大于sum的生成树中最小的一个。

【输入格式】

第一行包含两个整数N和M。

接下来M行，每行包含三个整数x, y, z，表示点x和点y之前存在一条边，边的权值为z。

【输出格式】

包含一行，仅一个数，表示严格次小生成树的边权和。（数据保证必定存在严格次小生成树）

【数据范围】

$N \le 10^5, M \le 3 \times 10^5$

【输入样例】

```
5 6
1 2 1
1 3 2
2 4 3
3 5 4
3 4 3
4 5 6
```

【输出样例】

```
11
```

【分析】

本题使用LCA倍增的方式求解新加入非树边的两点a, b间的最大边和严格次大边。除了fa与$depth$之外，我们还需要预处理两个数组：

- $d1[i][j]$：表示每个点i跳2^j路径上的最大边权
- $d2[i][j]$：表示每个点i跳2^j路径上的次大边权

如何预处理这两个数组呢？

还是利用分段的思想，节点i跳2^j个节点可以看成先跳2^{j-1}个节点再跳2^{j-1}个节点。即整段的最大与次大值一定在两个子段的最大与次大值中，即$d1[i][j], d2[i][j]$在$d1[i][j-1], d2[i][j-1], d1[fa[i][j-1]][j-1], d2[fa[i][j-1]][j-1]$中。

如下图所示，假设节点x, y的最近公共祖先为lca，我们在求解LCA时会将x, y不断上跳，在跳的过程中将$d1[x \to lca], d2[x \to lca], d1[y \to lca], d2[y \to lca]$记录下来，则$x, y$之间的最大边与次大边一定在这些$d1, d2$的值之中。

```
      lca
       o
      / \
     o   o
    /     \
   o       o
  /        y
 o
 x
```

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;

typedef long long LL;
const int N = 100010, M = 300010, INF = 0x3f3f3f3f;
int e[M], ne[M], d[M], h[N], idx;
int fa[N][17], depth[N], pre[N];
int d1[N][17], d2[N][17];//d1(2)[i][j]表示每个点i跳2^j路径上的最大(次大)边权
int n, m;

struct Edge
{
    int x, y, w;
    bool flag;//标记是否为最小生成树中的边
    bool operator< (const Edge& t) const
    {
        return w < t.w;
    }
}s[M];

void add(int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

int find(int k)
{
    if (pre[k] == k) return k;
    return pre[k] = find(pre[k]);
}

LL kruskal()
{
    for (int i = 1; i <= n; i++) pre[i] = i;
    sort(s, s + m);
    LL res = 0;

    for (int i = 0; i < m; i++)
```

```
    {
        int px = find(s[i].x), py = find(s[i].y), w = s[i].w;
        if (px != py) pre[px] = py, res += w, s[i].flag = true;
    }
    return res;
}

void build()
{
    memset(h, -1, sizeof h);
    for (int i = 0; i < m; i++)
        if (s[i].flag) add(s[i].x, s[i].y, s[i].w), add(s[i].y, s[i].x, s[i].w);
}

void bfs(int root)
{
    memset(depth, 0x3f, sizeof depth);
    depth[0] = 0, depth[root] = 1;
    queue<int> Q;
    Q.push(root);
    while (Q.size())
    {
        int t = Q.front();
        Q.pop();
        for (int i = h[t]; ~i; i = ne[i])
        {
            int j = e[i];
            if (depth[t] + 1 < depth[j])
            {
                depth[j] = depth[t] + 1;
                Q.push(j);
                fa[j][0] = t;
                d1[j][0] = d[i], d2[j][0] = -INF;//j和父节点t之间没有次大值
                for (int k = 1; k <= 16; k++)
                {
                    int anc = fa[j][k - 1];//anc表示从i跳2^(k-1)到达的节点
                    fa[j][k] = fa[anc][k - 1];
                    int dis[4] = { d1[j][k - 1], d2[j][k - 1], d1[anc][k - 1], d2[anc][k - 1] };

                    //整段的最大次大路径在两个子段的最大次大路径中找
```

```
                    d1[j][k] = d2[j][k] = -INF;
                    for (int u = 0; u < 4; u++)
                    {
                        if (dis[u] > d1[j][k]) d2[j][k] = d1[j][k], d1[j][k] = dis[u];
                        else if (dis[u] < d1[j][k] && dis[u] > d2[j][k]) d2[j][k] = dis[u];
                    }
                }
            }
        }
    }
}

int lca(int a, int b, int w)
{
    int dis[N << 1], cnt = 0;//dis保存两个点上跳到LCA时经过的所有最大次大路径
    if (depth[a] < depth[b]) swap(a, b);
    for (int k = 16; k >= 0; k--)
        if (depth[fa[a][k]] >= depth[b])
        {
            dis[cnt++] = d1[a][k];
            dis[cnt++] = d2[a][k];
            a = fa[a][k];
        }
    if (a != b)
    {
        for (int k = 16; k >= 0; k--)
            if (fa[a][k] != fa[b][k])
            {
                dis[cnt++] = d1[a][k];
                dis[cnt++] = d2[a][k];
                dis[cnt++] = d1[b][k];
                dis[cnt++] = d2[b][k];
                a = fa[a][k], b = fa[b][k];
            }
        dis[cnt++] = d1[a][0], dis[cnt++] = d1[b][0];//两点到LCA最后一步的距离
    }
    int dis1 = -INF, dis2 = -INF;
    for (int i = 0; i < cnt; i++)

        if (dis[i] > dis1) dis2 = dis1, dis1 = dis[i];
```

```
            else if (dis[i] < dis1 && dis[i] > dis2) dis2 = dis[i];
    if (w > dis1) return w - dis1;
    return w - dis2;
}

int main()
{
    cin >> n >> m;
    for (int i = 0; i < m; i++) cin >> s[i].x >> s[i].y >> s[i].w;
    LL sum = kruskal();
    build();
    bfs(1);//设1号点为根节点
    LL res = 1e18;
    for (int i = 0; i < m; i++)
        if (!s[i].flag) res = min(res, sum + lca(s[i].x, s[i].y, s[i].w));
    cout << res << endl;
    return 0;
}
```

四、AcWing 352. 闇の連鎖（树上差分）

【题目描述】

传说中的暗之连锁被人们称为Dark。

Dark是人类内心的黑暗的产物，古今中外的勇者们都试图打倒它。

经过研究，你发现Dark呈现无向图的结构，图中有N个节点和两类边，一类边被称为主要边，而另一类被称为附加边。

Dark有$N-1$条主要边，并且Dark的任意两个节点之间都存在一条只由主要边构成的路径。

另外，Dark还有M条附加边。

你的任务是把Dark斩为不连通的两部分。

一开始Dark的附加边都处于无敌状态，你只能选择一条主要边切断。

一旦你切断了一条主要边，Dark就会进入防御模式，主要边会变为无敌的而附加边可以被切断。

但是你的能力只能再切断Dark的一条附加边。

现在你想要知道，一共有多少种方案可以击败Dark。

注意，就算你第一步切断主要边之后就已经把Dark斩为两截，你也需要切断一条附加边才算击败了Dark。

【输入格式】

第一行包含两个整数N和M。

之后$N-1$行，每行包括两个整数A和B，表示A和B之间有一条主要边。

之后M行以同样的格式给出附加边。

【输出格式】

输出一个整数表示答案。

【数据范围】

$N \le 100000, M \le 200000$，数据保证答案不超过$2^{31}-1$

【输入样例】

```
4 1
1 2
2 3
1 4
3 4
```

【输出样例】

```
3
```

【分析】

在没有附加边的情况下，我们发现这是一颗树，那么再添加条附加边(u,v)后，会造成(u,v)之间产生一个环。

如果我们第一步截断了(u,v)之间的一条路，那么我们第二次只能截掉(u,v)之间的附加边，才能使其不连通。

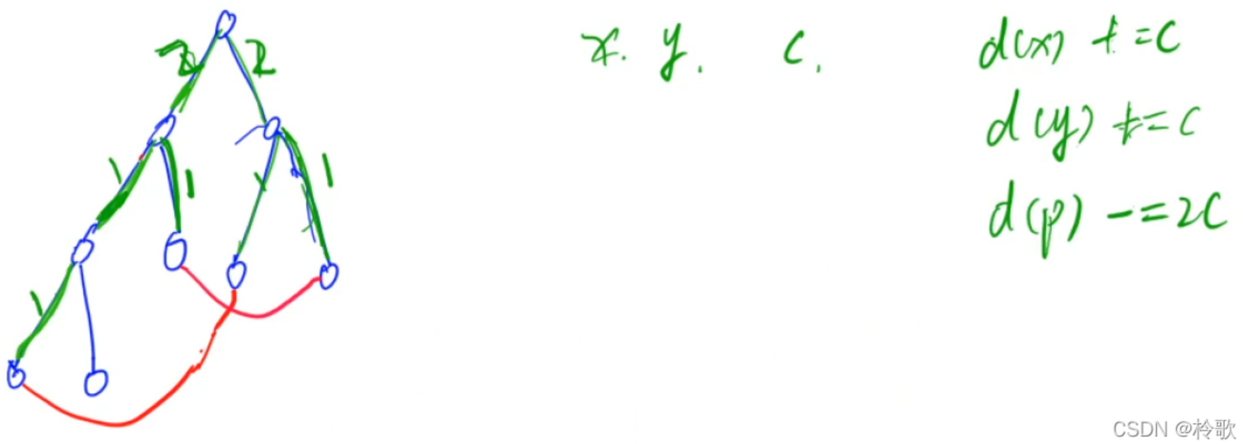
我们将每条附加边(u,v)称为将(u,v)之间的所有主要边覆盖了一遍，因此我们只需要统计出每条主要边被覆盖了几次即可。

- 对于只被覆盖一次的主要边，第二次我们只能切断(u,v)这条附加边，方法唯一，因此$res+1$；
- 如果我们第一步切断了被覆盖0次的主要边，那么我们已经将其分为两个连通块，那么第二部只需要在m条附加边中任选一条即可，因此$res+m$；

- 如果第一步截到被覆盖超过两次的主要边，无论切断哪条附加边都无法将其分为两部分，因此$res + 0$。

那么怎么标记(u, v)之间的主要边被覆盖了几次呢，那么我们可以使用树上差分，是解决此类问题的经典套路。

我们想，如果添加一条边，那么只会对$u \to lca(u, v) \to v$上的主要边产生影响，我们需要将$u \to v$之间的所有主要边的覆盖次数加一，对于新加的(u, v)这条附加边，我们将u节点的权值$+1$，v节点的权值$+1$，再将节点$lca(u, v)$的权值-2，这样就实现了将u, v之间的所有连边权值都$+1$的操作，画图很好理解。最后我们进行一遍DFS求出以每个节点为根节点的子树的权值，那么这个值就是当前节点与其父节点的连边所被覆盖的次数，根据覆盖次数按上述方法累加答案即可。



【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;

const int N = 100010, M = N << 1;
int e[M], ne[M], h[N], idx;
int fa[N][17], depth[N], b[N];//b为节点的差分数组
int n, m, res;

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

void bfs(int root)
```

```
{
    memset(depth, 0x3f, sizeof depth);
    depth[0] = 0, depth[root] = 1;
    queue<int> Q;
    Q.push(root);
    while (Q.size())
    {
        int t = Q.front();
        Q.pop();
        for (int i = h[t]; ~i; i = ne[i])
        {
            int j = e[i];
            if (depth[t] + 1 < depth[j])
            {
                depth[j] = depth[t] + 1;
                Q.push(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 u, int v)
{
    if (depth[u] < depth[v]) swap(u, v);
    for (int k = 16; k >= 0; k--)
        if (depth[fa[u][k]] >= depth[v]) u = fa[u][k];
    if (u == v) return u;
    for (int k = 16; k >= 0; k--)
        if (fa[u][k] != fa[v][k]) u = fa[u][k], v = fa[v][k];
    return fa[u][0];
}

int dfs(int u, int father)
{
    int sum = b[u];//sum表示以u为根节点的子树权值之和
    for (int i = h[u]; ~i; i = ne[i])
        if (e[i] != father)
        {
            int s = dfs(e[i], u);//s表示以u的子节点为根节点的子树和

            if (s == 0) res += m;
```

```
            else if (s == 1) res++;
            sum += s;
        }
    return sum;
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    for (int i = 0; i < n - 1; i++)
    {
        int u, v;
        cin >> u >> v;
        add(u, v), add(v, u);
    }
    bfs(1);
    for (int i = 0; i < m; i++)
    {
        int u, v;
        cin >> u >> v;
        b[u]++, b[v]++, b[lca(u, v)] -= 2;
    }
    dfs(1, -1);
    cout << res << endl;
    return 0;
}
```