# 树与图的深度优先遍历

给定一颗树，树中包含 $n$ 个结点（编号 $1 \sim n$）和 $n-1$ 条无向边。

请你找到树的重心，并输出将重心删除后，剩余各个连通块中点数的最大值。

重心定义：重心是指树中的一个结点，如果将这个点删除后，剩余各个连通块中点数的最大值最小，那么这个节点被称为树的重心。

**输入格式**

第一行包含整数 $n$，表示树的结点数。

接下来 $n-1$ 行，每行包含两个整数 $a$ 和 $b$，表示点 $a$ 和点 $b$ 之间存在一条边。

**输出格式**

输出一个整数 $m$，表示将重心删除后，剩余各个连通块中点数的最大值。

**数据范围**

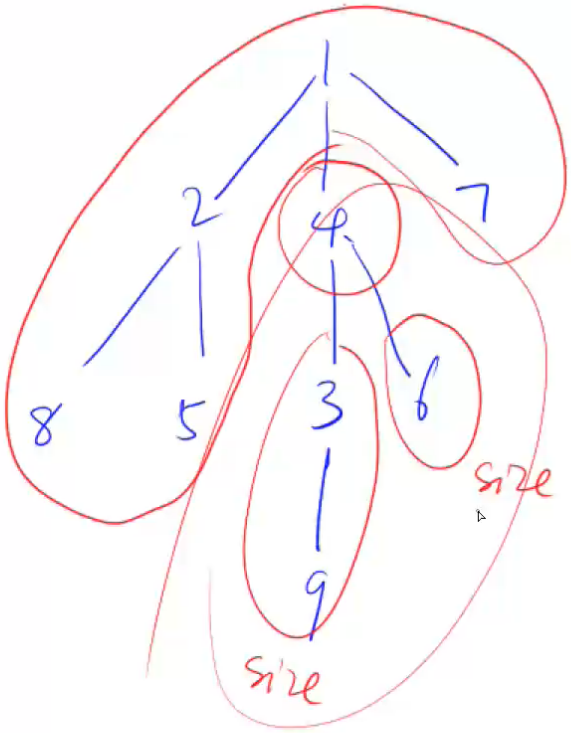
$1 \leq n \leq 10^5$

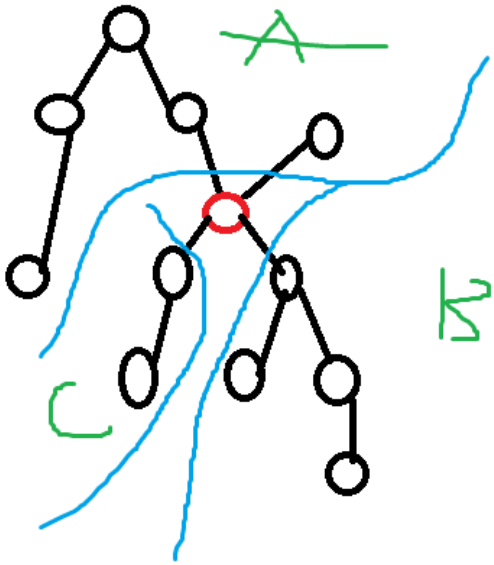
**输入样例**

```
9
1 2
1 7
1 4
2 8
2 5
4 3
3 9
4 6
```

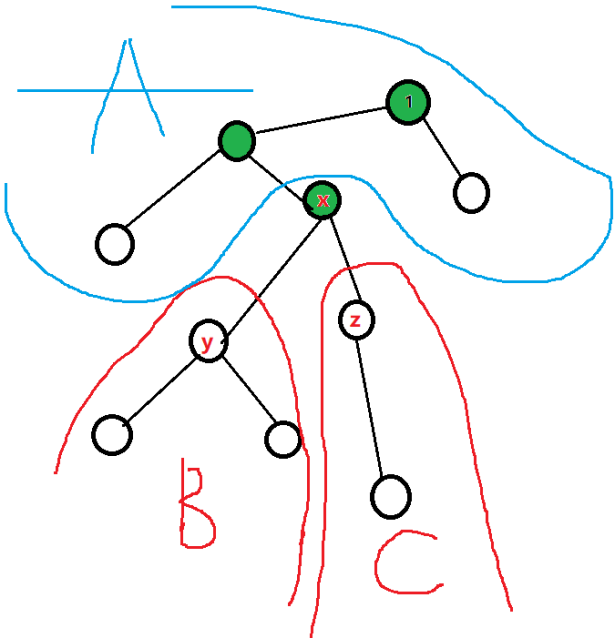
**输出样例：**

```
4
```




如左图所示，假设我将红色的点删掉，则会将整棵树划分为三颗子树，分别为A,B,C；假设树的节点数量为n，B子树的节点个数为size(B),C子树的节点个数为size(C),则我们所要求的删掉一个节点后各个联通块的最大值，就是求出size(B),size(C)和n-(size(B)+size(C)+1)的最大值




如左图所示，假设dfs函数已经递归到了节点x,对于点x来说，去掉该点后图中共有三个联通块：A、B和C.
B联通块的大小有size(B) = dfs(y),C联通块的大小为size(C) = dfs(z).所以dfs(x)的返回值也就是以点x为根的子树的有sum = 1 + size(B) + size(C).
这样我们就得到了A联通块的大小size(A) = n - sum.
最后得出最大的联通块的数量
max(size(A),size(B),size(C)).

y总代码：

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

using namespace std;

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

int n;
int h[N], e[M], ne[M], idx;
int ans = N;
bool st[N];

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

int dfs(int u)
{
    st[u] = true;

    int size = 0, sum = 0;
    for (int i = h[u]; i != -1; i = ne[i])
    {
        int j = e[i];
        if (st[j]) continue;

        int s = dfs(j);
        size = max(size, s);
        sum += s;
    }

    size = max(size, n - sum - 1);
    ans = min(ans, size);
```

```
    return sum + 1;
}

int main()
{
    scanf("%d", &n);

    memset(h, -1, sizeof h);

    for (int i = 0; i < n - 1; i ++ )
    {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b), add(b, a);
    }

    dfs(1);

    printf("%d\n", ans);

    return 0;
}
```



理解：

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

using namespace std;

const int N = 1e5 + 10; //数据范围是10的5次方
const int M = 2 * N; //以有向图的格式存储无向图，所以每个节点至多对应2n-2条边

int h[N]; //邻接表存储树，有n个节点，所以需要n个队列头节点
int e[M]; //存储元素
int ne[M]; //存储列表的next值
int idx; //单链表指针
int n; //题目所给的输入，n个节点
int ans = N; //表示重心的所有的子树中，最大的子树的结点数目

bool st[N]; //记录节点是否被访问过，访问过则标记为true

//a所对应的单链表中插入b  a作为根
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}

// dfs 框架
/*
void dfs(int u){
    st[u]=true; // 标记一下，记录为已经被搜索过了，下面进行搜索过程
```

```
    for(int i=h[u];i!=-1;i=ne[i]){
        int j=e[i];
        if(!st[j]) {
            dfs(j);
        }
    }
}
*/

//返回以u为根的子树中节点的个数，包括u节点
int dfs(int u) {
    int res = 0; //存储 删掉某个节点之后，最大的连通子图节点数
    st[u] = true; //标记访问过u节点
    int sum = 1; //存储 以u为根的树 的节点数，包括u，如图中的4号节点

    //访问u的每个子节点
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        //因为每个节点的编号都是不一样的，所以 用编号为下标 来标记是否被访问过
        if (!st[j]) {
            int s = dfs(j);  // u节点的单棵子树节点数 如图中的size值
            res = max(res, s); // 记录最大联通子图的节点数
            sum += s; //以j为根的树 的节点数
        }
    }

    //n-sum 如图中的n-size值，不包括根节点4;
    res = max(res, n - sum); // 选择u节点为重心，最大的 连通子图节点数
    ans = min(res, ans); //遍历过的假设重心中，最小的最大联通子图的 节点数
    return sum;
}

int main() {
    memset(h, -1, sizeof h); //初始化h数组 -1表示尾节点
    cin >> n; //表示树的结点数

    // 题目接下来会输入，n-1行数据，
    // 树中是不存在环的，对于有n个节点的树，必定是n-1条边
    for (int i = 0; i < n - 1; i++) {
        int a, b;
        cin >> a >> b;
        add(a, b), add(b, a); //无向图
    }

    dfs(1); //可以任意选定一个节点开始 u<=n

    cout << ans << endl;

    return 0;
}
```