树的直径

定义

树的直径是指树中任意两个节点之间的最长路径的长度（边数或距离）。直径的两个端点称为直径端点。

性质与应用

- **图论基础**：用于分析树的结构特征
- **网络优化**：数据中心、通信网络中的延迟评估
- **算法设计**：分治、动态规划的基础
- **竞技编程**：树形DP问题中的常见模式

算法步骤

方法一：两次BFS/DFS

> ⚠ 若存在负权边，则无法使用两次 DFS 的方式求解直径。

1. 从任意节点 `u` 开始进行BFS/DFS，找到距离 `u` 最远的节点 `v`
2. 从节点 `v` 再进行一次BFS/DFS，找到距离 `v` 最远的节点 `w`
3. `v` 到 `w` 的距离即为树的直径

伪代码示意

1. 从任意节点开始进行BFS
2. 记录访问过的节点和距离
3. 找到距离最远的节点 `v`
4. 从节点 `v` 再次进行BFS
5. 找到距离最远的节点 `w`
6. 返回 `v` 到 `w` 的距离作为直径

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
constexpr int maxn = 5e5 + 7;
vector<pair<int,int>> g[maxn]; // {to, weight}
int n, u, v;

// 从起点 dfs 找最远点（返回 {distance, node}）
pair<int,int> dfs_far(int u, int fa, int dist){
    pair<int,int> best = {dist, u};
    for(auto [to,w]: g[u]){
        if(to == fa) continue;
        auto cur = dfs_far(to, u, dist + w);
        if(cur.first > best.first) best = cur;
```

```
    }
    return best;
}

signed main(){
    cin.tie(0)->sync_with_stdio(0);
    cin >> n;
    for(int i = 1; i <= n - 1; ++i){
        cin >> u >> v;
        g[u].push_back({v, 1});
        g[v].push_back({u, 1});
    }
    // 任意点 1 -> 找到最远点 a
    auto a = dfs_far(1, 0, 0).second;
    // 从 a 出发找到最远点 b，距离即直径
    auto diam = dfs_far(a, 0, 0).first;
    cout << diam << '\n';
    return 0;
}
//Written By GPT-5 mini
```

方法二：树形DP

对每个节点计算经过该节点的最长路径，维护全局最大值。

伪代码示意

1. 选择任意节点作为根进行DFS
2. 对每个节点计算其子树中的最长路径
3. 计算经过该节点的最长路径（两条子树路径之和）
4. 更新全局最大直径
5. 回溯至父节点
6. 返回最大直径

参考代码

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
constexpr int maxn = 5e5 + 7;
vector<pair<int,int>> g[maxn]; // {to, weight}
int n, u, v;
int diam = 0;

// 返回以 u 为起点向下的最大长度
int dfs_dp(int u, int fa){
    int mx1 = 0, mx2 = 0; // 子路径中最大的两个
    for(auto [to,w]: g[u]){
        if(to == fa) continue;
        int d = dfs_dp(to, u) + w;
        if(d > mx1)mx2 = mx1,mx1 = d;
        else if(d > mx2) mx2 = d;
    }
```

```
    diam = max(diam, mx1 + mx2); // 经过 u 的最长路径
    return mx1;
}

signed main(){
    cin.tie(0)->sync_with_stdio(0);
    cin >> n;
    for(int i = 1; i <= n - 1; ++i){
        cin >> u >> v;
        g[u].push_back({v, 1});
        g[v].push_back({u, 1});
    }
    dfs_dp(1, 0);
    cout << diam << '\n';
    return 0;
}
```