树的直径

基本概念

树的直径：树中两个节点之间的最长路径

相关概念：

- **直径长度**：最长路径的边数或权重和
- **直径路径**：构成直径的节点序列
- **性质**：树的直径可能不唯一，但长度相同

树的直径性质

重要性质

1. **直径的端点**：从任意节点出发进行 BFS/DFS，最远的节点一定是直径的一个端点
2. **直径的唯一性**：直径的长度唯一，但路径可能有多条
3. **直径的中点**：直径路径的中点（可能为节点或边）是树的中心

数学表达：

- 对于树 $T = (V, E)$，直径 $D = \max_{u,v \in V} dist(u, v)$
- 其中 $dist(u, v)$ 表示节点 u 和 v 之间的最短距离

两次 DFS/BFS 方法

算法思想

核心原理：从任意节点出发找到最远点 u，再从 u 出发找到最远点 v，则 u 到 v 的路径就是直径

算法步骤：

1. 任选节点 x，通过 DFS/BFS 找到距离 x 最远的节点 u
2. 从节点 u 出发，通过 DFS/BFS 找到距离 u 最远的节点 v
3. u 到 v 的路径就是树的直径

时间复杂度：$O(n)$，其中 n 为节点数

奇思妙学 信息学竞赛

两次 DFS 实现（无权树）

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

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

    // 第一次 DFS：从节点1出发，找到最远点u
    max_dist = -1;
    dfs(1, -1, 0);
    int u = endpoint;

    // 第二次 DFS：从u出发，找到最远点v
    max_dist = -1;
    dfs(u, -1, 0);
    int v = endpoint;

    cout << "直径端点：" << u << " 和 " << v << endl;
    cout << "直径长度：" << max_dist << endl;

    return 0;
}
```

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

int h[N], e[M], ne[M], idx;
int n;
int dist[N];  // 存储距离
int max_dist, endpoint;  // 最大距离和端点

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

// DFS 遍历，找到距离 start 最远的节点
void dfs(int u, int father, int depth) {
    dist[u] = depth;

    // 更新最大距离和端点
    if (depth > max_dist) {
        max_dist = depth;
        endpoint = u;
    }

    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (j == father) continue;  // 避免回父节点
        dfs(j, u, depth + 1);
    }
}
```

奇思妙学 信息学竞赛

两次 BFS 实现（无权树）

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

int h[N], e[M], ne[M], idx;
int n;
int dist[N];
bool visited[N];

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
// ...
int main() {
    memset(h, -1, sizeof h);
    cin >> n;

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

    // 第一次 BFS
    int u = bfs(1);
    // 第二次 BFS
    int v = bfs(u);

    cout << "直径端点: " << u << " 和 " << v << endl;
    cout << "直径长度: " << dist[v] << endl;

    return 0;
}
```

```cpp
// BFS 遍历, 返回最远节点
int bfs(int start) {
    memset(visited, false, sizeof visited),memset(dist, 0, sizeof dist);

    queue<int> q;
    q.push(start),visited[start] = true,dist[start] = 0;

    int farthest = start;

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        // 更新最远节点
        if (dist[u] > dist[farthest])
            farthest = u;

        for (int i = h[u]; i != -1; i = ne[i]) {
            int j = e[i];
            if (!visited[j]) {
                visited[j] = true;
                dist[j] = dist[u] + 1;
                q.push(j);
            }
        }
    }

    return farthest;
}
```

带权树的直径

带权树的两遍 DFS

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

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

    max_dist = -1;
    dfs(1, -1, 0);
    int u = endpoint;

    max_dist = -1;
    dfs(u, -1, 0);
    int v = endpoint;

    cout << "直径端点: " << u << " 和 " << v << endl;
    cout << "直径长度: " << max_dist << endl;

    return 0;
}
```

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

int h[N], e[M], w[M], ne[M], idx;
int n;
int dist[N];  // 存储距离（带权）
int max_dist, endpoint;

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

void dfs(int u, int father, int depth) {
    dist[u] = depth;

    if (depth > max_dist) {
        max_dist = depth;
        endpoint = u;
    }

    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (j == father) continue;
        dfs(j, u, depth + w[i]);  // 累加权值
    }
}
```

奇思妙学 信息学竞赛

树形 DP 方法

算法思想

核心思路：对于每个节点，计算经过该节点的最长路径

状态定义：

- $dp[u]$：以 u 为根的子树中，从 u 出发的最长路径长度
- ans：全局答案，记录直径长度

状态转移：

对于节点 u，考虑所有子节点 v：

- 更新 $dp[u] = \max(dp[u], dp[v] + w(u, v))$
- 用最长路径和次长路径更新 ans

奇思妙学 信息学竞赛

树形 DP 实现

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

int h[N], e[M], w[M], ne[M], idx;
int n;
int dp[N];  // dp[u] 表示从u出发向下最远距离
int ans;    // 直径长度

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

// ...

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

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

    ans = 0;
    dfs(1, -1);

    cout << "直径长度: " << ans << endl;

    return 0;
}
```

```cpp
void dfs(int u, int father) {
    int max1 = 0, max2 = 0;  // 最长路径和次长路径

    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (j == father) continue;

        dfs(j, u);

        int distance = dp[j] + w[i];

        // 更新最长和次长路径
        if (distance > max1) {
            max2 = max1;
            max1 = distance;
        } else if (distance > max2) {
            max2 = distance;
        }
    }

    dp[u] = max1;  // 从u出发的最长路径
    ans = max(ans, max1 + max2);  // 经过u的最长路径
}
```

两种方法对比

算法特性比较

特性	两次 DFS/BFS	树形 DP
时间复杂度	$O(n)$	$O(n)$
空间复杂度	$O(n)$	$O(n)$
实现难度	简单	中等
输出信息	端点 + 长度	仅长度
适用场景	需要知道端点	仅需长度

注意： 两边DFS不能处理负边权！

选择建议

使用两次 DFS/BFS 的情况：

- 需要知道「直径的具体端点」
- 需要输出直径路径
- 实现简单，代码直观

使用树形 DP 的情况：

- 只需要「直径长度」
- 需要与其他树形 DP 问题结合
- 在复杂树结构中更灵活

奇思妙学 信息学竞赛

直径路径记录

记录直径路径（两次 DFS 方法）

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

int h[N], e[M], ne[M], idx;
int n;
int dist[N], pre[N];  // pre数组记录前驱节点
int max_dist, endpoint;
vector<int> diameter_path;

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

奇思妙学 信息学竞赛

```
void dfs(int u, int father, int depth) {
    dist[u] = depth;
    pre[u] = father;  // 记录父节点，重点在此

    if (depth > max_dist) {
        max_dist = depth;
        endpoint = u;
    }

    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (j == father) continue;
        dfs(j, u, depth + 1);
    }
}
// 构建直径路径
void build_path(int start, int end) {
    diameter_path.clear();
    int cur = end;
    while (cur != start) {
        diameter_path.push_back(cur);
        cur = pre[cur];
    }
    diameter_path.push_back(start);
}
```

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

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

    max_dist = -1;
    dfs(1, -1, 0);
    int u = endpoint;

    max_dist = -1;
    dfs(u, -1, 0);
    int v = endpoint;

    build_path(u, v);

    cout << "直径端点: " << u << " 和 " << v << endl;
    cout << "直径长度: " << max_dist << endl;
    cout << "直径路径: ";
    for (int i = diameter_path.size() - 1; i >= 0; i--) {
        cout << diameter_path[i] << " ";
    }
    cout << endl;
}
```

应用实例

问题：树的中心

定义： 树上到其他所有节点最大距离最小的节点

求解方法：

1. 找到直径 $u-v$
2. 在直径路径上找到中点

```cpp
// 在直径路径上找中心
int find_center(const vector<int>& path) {
    int len = path.size();
    if (len % 2 == 1) {
        return path[len / 2];  // 奇数长度，中心是中间节点
    } else {
        // 偶数长度，中心可以是中间两个节点的任意一个
        // 或者需要进一步判断
        return path[len / 2 - 1];  // 返回前一个
    }
}
```

复杂度分析

时间复杂度

算法	时间复杂度	说明
两次 DFS	$O(n)$	每个节点访问一次
两次 BFS	$O(n)$	每个节点访问一次
树形 DP	$O(n)$	每个节点处理一次

空间复杂度

所有方法都是 $O(n)$，主要用于存储树结构和辅助数组。

奇思妙学 信息学竞赛

注意事项

实现细节

1. **图的存储**：使用邻接表存储树结构
2. **避免循环**：DFS 时需要记录父节点避免回环
3. **初始化**：确保数组正确初始化
4. **边界情况**：处理 $n = 1$ 的情况

常见错误

1. **忘记建双向边**：树是无向图，需要添加双向边
2. **数组越界**：确保数组大小足够
3. **未初始化**：全局变量需要正确初始化