河南省实验中学
HENAN EXPERIMENTAL HIGH SCHOOL

数据结构基础：树结构

2026年5月

课程导航

模块	内容
1	🌱 初识树——基本概念与术语
2	🌿 存储之道——多种存储方法
3	🚶 遍历之旅——先根/后根/层次
4	🌳 进阶应用——直径、重心、LCA、树形DP
5	✍️ 巩固练习——基础题+OJ实战

从线性到非线性

线性结构	非线性结构
数组	**树**
链表	图
队列	

树结构：一对多的层次关系，模拟现实中的“树”

树的定义

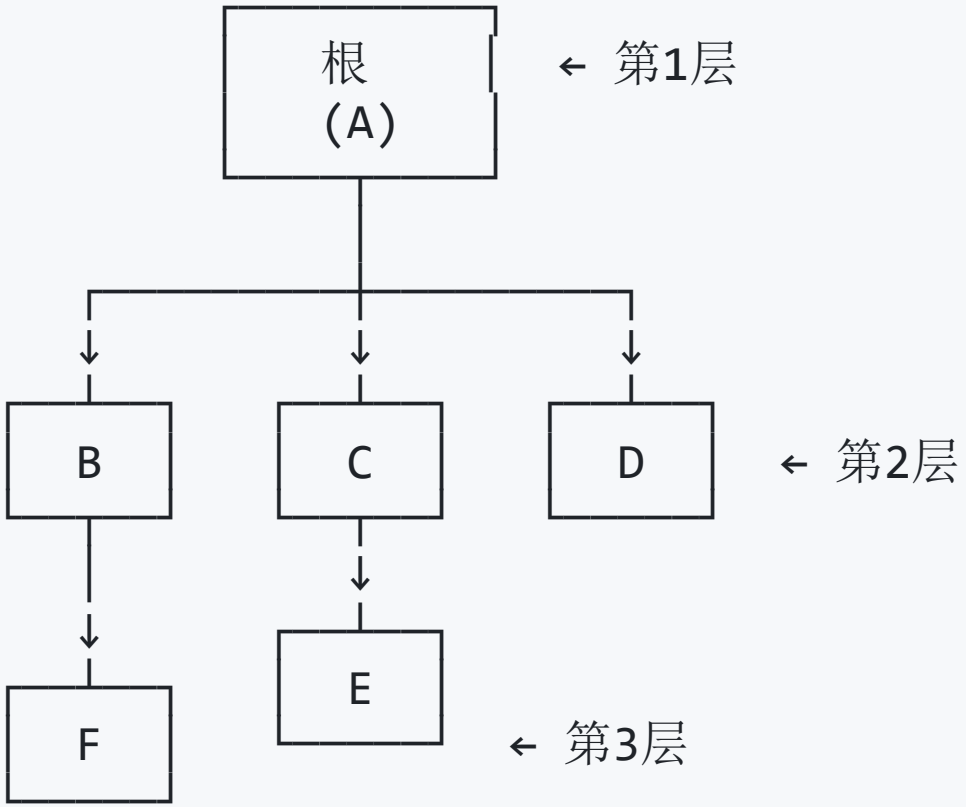
树 (Tree) 是 n (n ≥ 0) 个结点的有限集合。

非空树必须满足：

- ✅ 有且仅有一个称为 **根 (Root)** 的结点
- ✅ 其余结点分成 **互不相交** 的有限集合 $T_1, T_2, \ldots, T_m$，每个集合本身又是一棵树，称为根的 **子树**

一棵有 n 个结点的树，恰好有 **n-1** 条边

树的结构图示



层次分明，每个结点可以有任意多个孩子

树的基本术语

术语	含义
根 (Root)	没有父结点的结点
父结点 (Parent)	直接前驱
子结点 (Child)	直接后继
兄弟 (Sibling)	同一父结点的结点
叶子 (Leaf)	没有子结点的结点
结点的度 (Degree)	子结点的个数
树的度	所有结点度的最大值
层次 (Level)	根为第1层，向下递增
深度/高度	最大层次数

一般树的存储方法

方法	实现	适用场景
双亲表示法	`parent[i]` 存储父结点	自底向上递推
邻接表法	`vector<int> G[N]`	最常用，万能
孩子兄弟表示法	`child[i]` + `sib[i]`	多叉树转二叉树（初赛常见）

💡 **竞赛推荐：邻接表法** —— 空间 O(n)，方便遍历所有孩子

邻接表法详解

```cpp
#include <vector>
using namespace std;

const int N = 100005;
vector<int> G[N];   // G[u] 存储 u 的所有子结点

// 添加一条边（父结点 u，子结点 v）
G[u].push_back(v);

// 若需要同时记录父结点，可额外开 parent 数组
int parent[N];
parent[v] = u;
```

邻接表是存储一般树的标准方式，简洁高效

双亲表示法

```
int parent[N];  // parent[i] = 结点 i 的父结点编号
// 根结点的 parent[root] = 0（或 -1）

// 查询结点 u 的所有子结点：需遍历所有结点 O(n)
for (int v = 1; v <= n; v++) {
    if (parent[v] == u) {
        // v 是 u 的孩子
    }
}
```

特点：

- ✅ 找父结点 O(1)
- ❌ 找孩子需要遍历整个数组

孩子兄弟表示法

```
int child[N];   // 第一个孩子
int sib[N];     // 下一个兄弟（右兄弟）

// 初始化: child[u] = -1, sib[u] = -1
// 添加孩子 v 到结点 u
if (child[u] == -1) child[u] = v;
else {
    sib[v] = child[u];
    child[u] = v;
}
```

每个结点只存两个指针：**第一个孩子** + **下一个兄弟**

任何一般树都能用这种方式转化为二叉树（左孩子右兄弟）

树的遍历——核心思想

遍历 = 按照某种顺序访问树中每个结点恰好一次

遍历方式	核心	数据结构
深度优先遍历 (DFS)	先往深处走，再回溯	递归 / 栈
广度优先遍历 (BFS)	按层逐层推进	队列

DFS 和 BFS 是树结构上最基础的算法，所有进阶应用都离不开它们

深度优先遍历 (DFS) - 一般树

```
void dfs(int u) {
    // 访问当前结点 u（先根序）
    cout << u << " ";
    // 递归访问所有子结点
    for (int v : G[u]) {
        dfs(v);
    }
}
```

若将 `cout` 放在递归之后，则为**后根遍历**（先孩子，后根）

- 时间复杂度：O(n)
- 空间复杂度：O(树高)（递归栈）

一般树的两种 DFS 次序

遍历类型	访问顺序	示例树 (A -> B,C,D; B->E,F)
先根遍历	根 → 子树1 → 子树2 → ...	A, B, E, F, C, D
后根遍历	子树1 → 子树2 → ... → 根	E, F, B, C, D, A

先根遍历与二叉树的先序遍历类似，但不需要区分左右

应用：

- 先根：复制树结构，前缀表达式
- 后根：释放树结点，后缀表达式

广度优先遍历 (BFS) - 层次遍历

```
#include <queue>

void bfs(int root) {
    queue<int> q;
    q.push(root);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        cout << u << " ";           // 按层输出
        for (int v : G[u]) {
            q.push(v);
        }
    }
}
```

- 使用队列，逐层访问
- 可同时计算每个结点的**层次（深度）**
- 时间复杂度 O(n)，空间复杂度 O(宽度)

遍历的应用示例

问题： 求树的高度（从根到最远叶子的距离）

```
int height(int u) {
    int maxH = 0;
    for (int v : G[u]) {
        maxH = max(maxH, height(v));
    }
    return maxH + 1;
}
```

遍历的应用示例

问题： 求树的结点总数

```
int countNodes(int u) {
    int sum = 1;  // 当前结点
    for (int v : G[u]) {
        sum += countNodes(v);
    }
    return sum;
}
```

树上的许多问题都可以通过 DFS + 后根遍历（自底向上）解决

树的简单应用——建树与基础统计

例题：找树根和孩子（一本通 T1336）

给定 n 个结点和 m 条父子边，输出树的根、孩子最多的结点及其孩子。

输入	输出
8 7 4 1 4 2 1 3 1 5 2 6 2 7 2 8	4 2 6 7 8

思路：

- 使用 parent 数组记录父结点，根结点 parent = 0
- 使用 childCount 数组记录每个结点的子结点个数
- 遍历找根和最大值

建树问题——参考代码

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, m;
    cin >> n >> m;
    vector<int> parent(n+1, 0);
    vector<int> childCount(n+1, 0);

    for (int i = 0; i < m; i++) {
        int x, y;
        cin >> x >> y;
        parent[y] = x;   childCount[x]++;
    }

    int root = 0;
    for (int i = 1; i <= n; i++)
        if (parent[i] == 0) root = i;

    int maxNode = 1;
    for (int i = 1; i <= n; i++)
        if (childCount[i] > childCount[maxNode]) maxNode = i;

    cout << root << endl << maxNode << endl;
    for (int i = 1; i <= n; i++)
        if (parent[i] == maxNode) cout << i << " ";
    return 0;
}
```

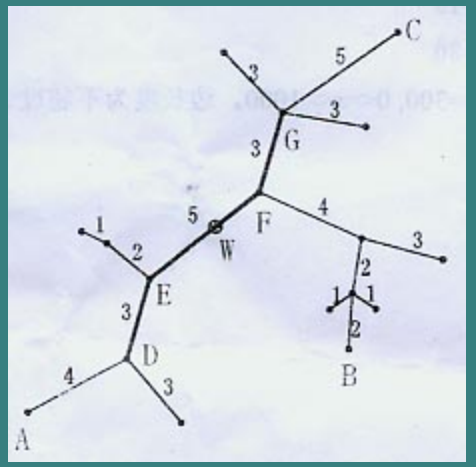
进阶应用——树的直径

给定一棵树，树中每条边都有一个权值，树中两点之间的距离定义为连接两点的路径边权之和

树中最远的两个节点之间的距离被称为树的直径，连接这两点的路径被称为树的最长链，通常称为树的“直径”

即直径是一个数值概念，也可代指一条路径

直径的性质：从边权值均为正的树上任意一个点出发所到达的最远的点一定是树的直径两个端点之一。



进阶应用——树的直径

求法（两遍 DFS/BFS）：

1. 从任意结点出发，找到最远的结点 u
2. 从 u 出发，找到最远的结点 v
3. 路径 (u, v) 即为直径，长度 = d[v]

> 证明：直径的端点一定是叶子，且第一次 DFS 找到的 u 必是直径的一个端点

树的直径——代码实现（BFS）

```
int d[N];  // 距离数组

int bfs(int start) {//适用于有根或无根树
    memset(d, -1, sizeof(d));
    queue<int> q;
    q.push(start);
    d[start] = 0;
    int farthest = start;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v : G[u]) {
            if (d[v] == -1) {
                d[v] = d[u] + 1;
                q.push(v);
                if (d[v] > d[farthest]) farthest = v;
            }
        }
    }
    return farthest;
}

int getDiameter() {
    int u = bfs(1);
    int v = bfs(u);
    return d[v];  // 直径长度
}
```

进阶应用——树的重心

重心：以该点为根时，所有子树的大小（结点数）的最大值最小

性质：

- 重心将树分成若干部分，每部分大小 ≤ n/2
- 树中所有点到重心的距离之和最小
- 重心最多两个，且相邻

进阶应用——树的重心

求法： DFS 计算子树大小，同时维护“最大子树大小”的最小值

```
int sz[N], maxSub[N], center = 0, minMax = INF;
void dfs_center(int u, int fa) {
    sz[u] = 1;
    maxSub[u] = 0;
    for (int v : G[u]) {
        if (v == fa) continue;
        dfs_center(v, u);
        sz[u] += sz[v];
        maxSub[u] = max(maxSub[u], sz[v]);
    }
    maxSub[u] = max(maxSub[u], n - sz[u]); // 上方部分
    if (maxSub[u] < minMax) {
        minMax = maxSub[u];
        center = u;
    }
}
```

进阶应用——最近公共祖先 LCA

LCA (Lowest Common Ancestor)：两个结点的公共祖先中深度最大的那个

应用： 树上两点距离 = depth[u] + depth[v] - 2*depth[lca]

算法	预处理	单次查询	特点
倍增法	O(n log n)	O(log n)	在线，最常用
Tarjan	O(n)	O(1)	离线并查集
树剖	O(n)	O(log n)	代码稍长

重点掌握**倍增法**

倍增 LCA——预处理

```
int f[N][LOG];  // f[i][j]: i 的 2^j 级祖先
int depth[N];

void dfs_lca(int u, int fa) {
    depth[u] = depth[fa] + 1;
    f[u][0] = fa;
    for (int j = 1; j < LOG; j++)
        f[u][j] = f[f[u][j-1]][j-1];
    for (int v : G[u])
        if (v != fa) dfs_lca(v, u);
}
```

倍增 LCA——查询

```
int lca(int x, int y) {
    if (depth[x] < depth[y]) swap(x, y);
    // 将 x 上移到与 y 同一深度
    int diff = depth[x] - depth[y];
    for (int j = 0; j < LOG; j++)
        if (diff & (1 << j)) x = f[x][j];
    if (x == y) return x;
    // 一起上移直到父结点相同
    for (int j = LOG-1; j >= 0; j--)
        if (f[x][j] != f[y][j]) {
            x = f[x][j];
            y = f[y][j];
        }
    return f[x][0];
}
```

时间复杂度 O(log n)，非常实用

进阶应用——树形DP

在树上做动态规划，通过 DFS 自底向上合并子问题

经典例题：「没有上司的舞会」

- 每个结点有权值，不能同时选相邻的结点
- 求最大权值和

状态	含义	转移
f[u][0]	不选 u，以 u 为根的子树的最大值	f[u][0] = Σ max(f[v][0], f[v][1])
f[u][1]	选 u，以 u 为根的子树的最大值	f[u][1] = val[u] + Σ f[v][0]

答案 = max(f[root][0], f[root][1])

树形DP代码框架

```
int val[N];
int f[N][2];

void dfs_dp(int u, int fa) {
    f[u][1] = val[u];
    f[u][0] = 0;
    for (int v : G[u]) {
        if (v == fa) continue;
        dfs_dp(v, u);
        f[u][1] += f[v][0];
        f[u][0] += max(f[v][0], f[v][1]);
    }
}

// 调用: dfs_dp(root, 0);
// 答案: max(f[root][0], f[root][1])
```

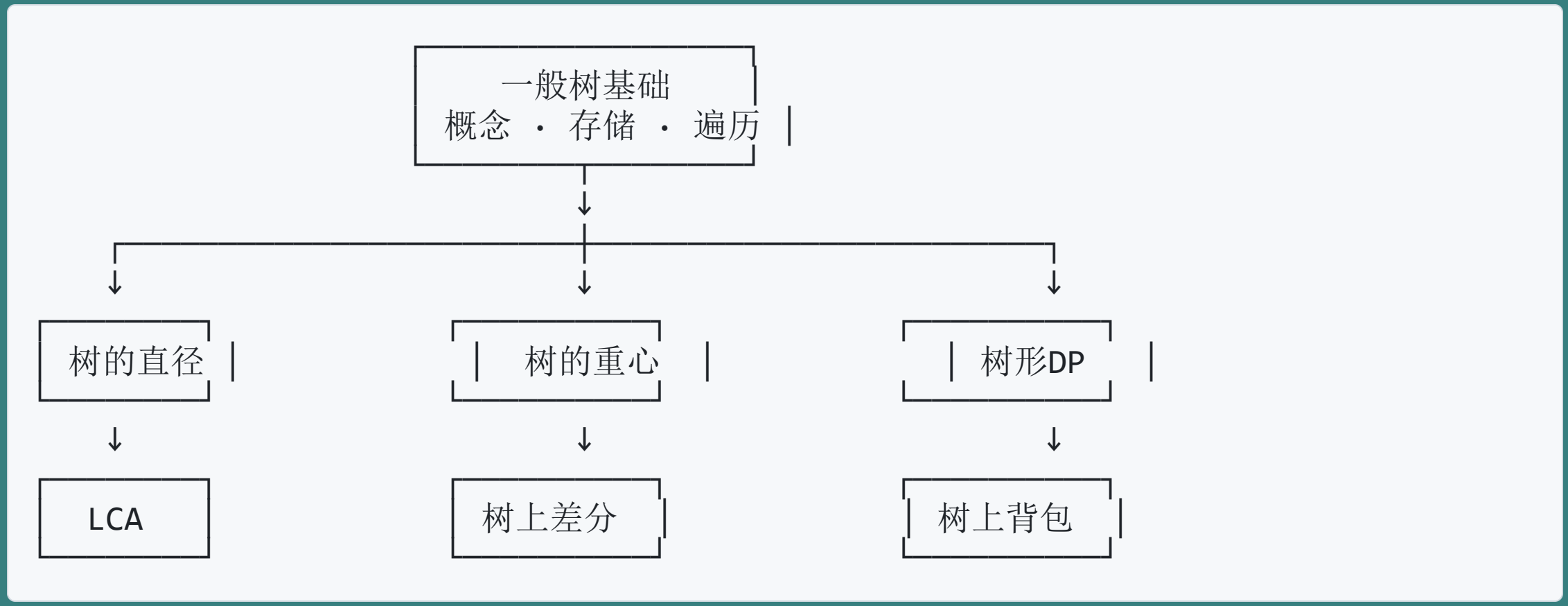
树形DP是信息学竞赛的重要考点，务必掌握

OJ实战练习题

来源	题目	难度	考察点
一本通 T1336	找树根和孩子	★	建树、统计
洛谷 P1352	没有上司的舞会	★★	树形DP
洛谷 P5906	树的直径	★★	两次BFS
洛谷 P1395	会议（树的重心）	★★	重心 + 距离和
洛谷 P3379	【模板】最近公共祖先（LCA）	★★★	倍增法

> 建议顺序：找树根和孩子 → 树形DP → 直径 → 重心 → LCA

知识体系总结



所有应用都建立在 DFS/BFS 的基础之上

寄语

树形结构是信息学竞赛的半壁江山，

不要畏惧递归，不要害怕 DFS——
每一个 OI 高手，都是从一棵小树苗成长起来的。

保持好奇心，多画图，多调试，
你一定会爱上这个层次分明的世界！

参考资料

1. **OI Wiki** – 树基础 / 树的直径 / 树的重心 / LCA / 树形DP
   https://oi-wiki.org/graph/tree-basic/
2. **信息学奥赛一本通** – 第6章 树结构
3. **洛谷题库** – 树结构相关题目
   P1352, P5906, P1395, P3379 等
4. **《算法竞赛入门经典》** – 第11章 树