树

Tan Yiqing

2025 年 10 月 31 日

1 树的基本概念

1.1 树的定义

树是一个 n 个结点的有限集合。当 n = 0 时，称为空树。如果非空，则满足以下条件：

- 有且仅有一个特殊的结点称为根结点。
- 当 n>1 时，其余结点可分为 m(m≥1) 个互不相交的有限集合 T1,T2,...,Tm，其中每个集合本身又是一棵树，称为根结点的子树。

这意味着树的很多算法是使用递归实现的。

1.2 树的术语

- 结点的度：结点所拥有的子树的个数。
- 树的度：树中各结点度的最大值。
- 叶结点：度为 0 的结点，也称为终端结点。
- 分支结点：度不为 0 的结点，也称为非终端结点。
- 结点的层次：根结点的层数为 1；对其他结点，若某结点在第 k 层，那么其孩子结点在第 k + 1 层。
- 树的高度：树中所有结点的最大层次。
- 层序编号：从上到下、从左到右对树中结点进行编号。
- 孩子、双亲：一个结点拥有的子树，其根结点称为这个结点的孩子；而同时该结点亦为其孩子的双亲（比如 B 是 A 的孩子；A 是 B 的双亲）
- 祖先、子孙：从根结点到某一结点 p 的路径上所有结点（不包括结点 p）都是结点 p 的祖先，而结点 p 是这些结点的后代。
- 兄弟：具有同一个双亲的结点称为兄弟。
- 有序树、无序树：如果树中每个结点的子树有一个从左到右的次序，则称该树为有序树，否则称为无序树。
- 森林：m(m≥0) 棵互不相交的树的集合称为森林。当 m=0 时，称为空森林；当 m=1 时，森林中只有一棵树，此时森林退化为树。
- 路径：从结点 p 到结点 q 所经过的结点及边的序列称为从 p 到 q 的路径。
- 路径长度：路径上边的数目。

1.3 与线性结构的区别

- 线性结构中每个结点最多只有一个前驱和一个后继，而树中每个结点可以有多个孩子。
- 线性结构中只有一个结点没有前驱 (头结点)，而树中只有一个结点没有双亲 (根结点)。
- 线性结构中只有一个结点没有后继 (尾结点)，而树中可以有多个叶结点。

1.4 树的遍历

遍历树是指按某种次序访问树中所有结点且每个结点只访问一次。常用的遍历方法有前序遍历、后序遍历和层次遍历。假设树为：

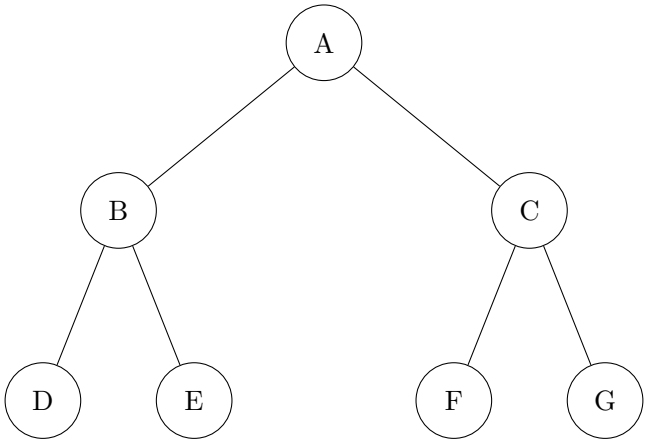

图 1: 树的示例

1.4.1 前序遍历

若空树，则遍历完成。先访问根结点，然后依次从左到右遍历各个子树。即：A-B-D-E-C-F-G。

1.4.2 后序遍历

若空树，则遍历完成。先依次从左到右遍历各个子树，然后访问根结点。即：D-E-B-F-G-C-A。

1.4.3 层次遍历

若空树，则遍历完成。先访问根结点，然后依次从上到下、从左到右遍历各个子树。该方法没有使用树本身的结构。即：A-B-C-D-E-F-G。

1.4.4 确定树的唯一性

知道前序和后序，可以唯一确定树。方法为：

- 1. 前序遍历的第一个结点一定是根结点。
- 2. 在后序遍历中找到根结点的位置，根结点左边的部分对应左子树，右边的部分对应右子树（如果有多棵子树，则依次划分）。
- 3. 在前序和后序序列中递归地对应每一棵子树，重复上述过程，直到所有结点都被确定。
- 4. 由于树的结构是有序的（即每个结点的孩子有顺序为从左往右，且不分左右），所以可以唯一还原整棵树。

举例：
已知前序遍历为：A B D E C F G
已知后序遍历为：D E B F G C A

1. 根结点为 A（前序第一个，后序最后一个）。
2. 在后序中，D E B 是 A 的左子树，F G C 是 A 的右子树。
3. 前序中，A 后面的 B D E 对应左子树，C F G 对应右子树。
4. 递归处理左子树和右子树，依次确定所有结点的父子关系。

2 树的实现

2.1 双亲表示法

双亲表示法使用一个一维数组来存储树中的结点。每个结点包含两个信息：结点的值和其双亲结点在数组中的下标。根结点的双亲下标通常设为-1 或其他特殊值表示没有双亲。

2.2 孩子表示法

孩子表示法使用一个一维数组来存储树中的结点，每个结点包含两个信息：结点的值和其第一个孩子结点在数组中的下标。根结点的孩子下标通常设为-1 或其他特殊值表示没有孩子。每个结点的兄弟结点通过下标关系来表示。

2.3 孩子兄弟表示法

孩子兄弟表示法使用一个一维数组来存储树中的结点。每个结点包含三个信息：结点的值、第一个孩子结点在数组中的下标和下一个兄弟结点在数组中的下标。根结点的孩子和兄弟下标通常设为-1 或其他特殊值表示没有孩子或兄弟。

3 二叉树

3.1 定义

二叉树是 n 个结点的有限集合。当集合为空时，称为空二叉树。非空时，则有一个根结点以及两颗互不相交的二叉树构成（分别为左子树、右子树）。

二叉树不是树!!!

核心区别：二叉树区分左右!!! 可以有右孩子没有左孩子。

3.2 术语

- 满二叉树：一棵深度为 k 的二叉树，若其所有叶结点都在第 k 层，且每个非叶结点都有两个孩子，则称该二叉树为满二叉树。
- 完全二叉树：设二叉树的深度为 k，除第 k 层外，其余各层上的结点数都达到最大值，第 k 层所有结点都连续集中在最左边，则称该二叉树为完全二叉树。
- 二叉搜索树：若它是一棵空树，或它的左子树上所有结点的值均小于根结点的值，且它的右子树上所有结点的值均大于根结点的值，并且它的左、右子树也分别为二叉搜索树，则称该二叉树为二叉搜索树。
- 平衡二叉树：一棵空树，或它的左、右子树都是平衡二叉树，且左、右子树的高度之差的绝对值不超过 1，则称该二叉树为平衡二叉树。
- 斜树：每个结点只有一个左或右孩子的二叉树称为斜树。

完全二叉树的判别（算法流程）：

- 只有右孩子没有左孩子，不是完全二叉树
- 出现了只有左孩子而没有右孩子，or 没有孩子，则后面只能出现叶子结点

3.3 性质

- 在二叉树的第 i 层上至多有 2^{i-1} 个结点 (i≥1)。
- 深度为 k 的二叉树至多有 2^k-1 个结点 (k≥1)。
- 对于任何一棵二叉树 T，设其叶结点数为 n0，度为 2 的结点数为 n2，则有 n0=n2+1。

3.4 完全二叉树特性

设完全二叉树的深度为 k，结点总数为 n，则有：

$$2^{k-1} \leq n < 2^k$$

高度至少为 $\lfloor \log_2(n) \rfloor + 1$，至多为 n。　　设完全二叉树的结点按层序从上到下、从左到右编号，则对于任意结点 i：

- 若 i=1，则结点 i 为根结点，无双亲。
- 若 i>1，则结点 i 的双亲为 $\left\lfloor \frac{i}{2} \right\rfloor$。
- 若 2i≤ n，则结点 i 的左孩子为 2i。
- 若 2i+1≤ n，则结点 i 的右孩子为 2i+1。

完全二叉树可以用顺序结构实现。

3.5 二叉树的遍历

二叉树的遍历包括前序遍历、中序遍历、后序遍历和层次遍历。假设二叉树为：

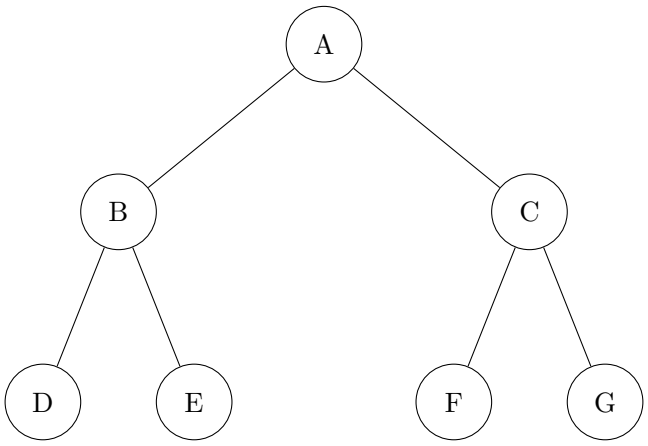

图 2: 二叉树的示例

3.5.1 前序遍历

前序遍历的顺序为：根结点 - 左子树 - 右子树。如：A-B-D-E-C-F-G。

3.5.2 中序遍历

中序遍历的顺序为：左子树 - 根结点 - 右子树。如：D-B-E-A-F-C-G。

3.5.3 后序遍历

后序遍历的顺序为：左子树 - 右子树 - 根结点。如：D-E-B-F-G-C-A。

3.5.4 层次遍历

层次遍历的顺序为：从上到下、从左到右。如：A-B-C-D-E-F-G。

3.5.5 确定**二叉树**的唯一性

知道前序和中序，可以唯一确定二叉树。方法为：

- 1. 前序遍历的第一个结点一定是根结点。
- 2. 在中序遍历中找到根结点的位置，根结点左边的部分对应左子树，右边的部分对应右子树。
- 3. 在前序和中序序列中递归地对应每一棵子树，重复上述过程，直到所有结点都被确定。
- 4. 由于二叉树的结构是有序的（即每个结点的左、右孩子有顺序），所以可以唯一还原整棵二叉树。

举例：已知前序遍历为：A B D E C F G 已知中序遍历为：D B E A F C G

1. 根结点为 A（前序第一个，后序最后一个）。
2. 在中序中，D B E 是 A 的左子树，F C G 是 A 的右子树。
3. 前序中，A 后面的 B D E 对应左子树，C F G 对应右子树。
4. 递归处理左子树和右子树，依次确定所有结点的父子关系。
5. 最终还原出唯一的二叉树结构。

后序加中序也可以唯一确定二叉树。方法为：

- 1. 后序遍历的最后一个结点一定是根结点。
- 2. 在中序遍历中找到根结点的位置，根结点左边的部分对应左子树，右边的部分对应右子树。
- 3. 在后序和中序序列中递归地对应每一棵子树，重复上述过程，直到所有结点都被确定。
- 4. 由于二叉树的结构是有序的（即每个结点的左、右孩子有顺序），所以可以唯一还原整棵二叉树。

举例：已知后序遍历为：D E B F G C A 已知中序遍历为：D B E A F C G

1. 根结点为 A（后序最后一个，中序第一个）。
2. 在中序中，D B E 是 A 的左子树，F C G 是 A 的右子树。
3. 后序中，D E B 对应左子树，F G C 对应右子树。
4. 递归处理左子树和右子树，依次确定所有结点的父子关系。
5. 最终还原出唯一的二叉树结构。

方法要点：找重复出现但顺序不同的部分来区分左、右、根。

但是**前序和后序**无法确定唯一二叉树，根本原因是二叉树区分左右。

Proposition：已知前序和中序遍历，可以唯一确定二叉树。

Proof：当结点是 0 或 1 时，结论明显。
假设对少于 n 个结点的二叉树结论成立，现考虑 n 个结点的二叉树。
设前序遍历序列为 $P_1, P_2, \ldots, P_n$，中序遍历序列为 $M_1, M_2, \ldots, M_n$。
根据前序定义，P_1 为根结点。那么存在 $1 \leq i \leq n$，使得 $M_i = P_1$。
按照中序的定义，$M_1, M_2, \ldots, M_{i-1}$ 为左子树的中序遍历序列，$M_{i+1}, M_{i+2}, \ldots, M_n$ 为右子树的中序遍历序列。
又根据前序的定义，$P_2, P_3, \ldots, P_i$ 为左子树的前序遍历序列，$P_{i+1}, P_{i+2}, \ldots, P_n$ 为右子树的前序遍历序列。
因为 $i - 1 < n$ 由归纳假设，左子树和右子树均可以唯一确定，因此整棵二叉树也可以唯一确定。
Q.E.D.

3.6 存储二叉树

3.7 顺序储存

一般只有完全二叉树使用顺序存储。

3.7.1 链式存储

每个结点由一个数据域和两个指针域组成，分别指向其左、右孩子结点的位置。

4 二叉树遍历的递归和非递归写法

4.1 前序遍历

4.1.1 递归写法

思路：

- 若结点为空，则返回。
- 访问根结点。
- 递归遍历左子树。
- 递归遍历右子树。
- 返回。

前序遍历的递归实现如下：

```
void preOrder(TreeNode* root) {
    if (root == NULL) return;
    visit(root);
    preOrder(root->left);
    preOrder(root->right);
}
```

4.1.2 非递归写法

思路：

- 使用栈来模拟递归过程。

- 初始化栈为空，当前结点为根结点。
- 当当前结点不为空或栈不为空时，重复以下步骤：
  - 当当前结点不为空时，访问当前结点并输出，将其右孩子入栈，然后将当前结点更新为其左孩子。
  - 当当前结点为空且栈不为空时，从栈中弹出一个结点，将其右孩子赋值给当前结点。

前序遍历的非递归实现可以使用栈来模拟递归过程：

```
void preOrder(TreeNode* root) {
    stack<TreeNode*> s;
    TreeNode* curr = root;
    while (curr != NULL || !s.empty()) {
        while (curr != NULL) {
            visit(curr);
            s.push(curr);
            curr = curr->left;
        }
        if (!s.empty()) {
            curr = s.top();
            s.pop();
            curr = curr->right;
        }
    }
}
```

4.2 中序遍历

4.2.1 递归写法

思路：

- 若结点为空，则返回。
- 递归遍历左子树。
- 访问根结点。
- 递归遍历右子树。
- 返回。

中序遍历的递归实现如下：

```
void inOrder(TreeNode* root) {
    if (root == NULL) return;
    inOrder(root->left);
    visit(root);
    inOrder(root->right);
}
```

4.2.2 非递归写法

思路：

- 使用栈来模拟递归过程。
- 初始化栈为空，当前结点为根结点。
- 当当前结点不为空或栈不为空时，重复以下步骤：
  - 当当前结点不为空时，将其入栈，然后将当前结点更新为其左孩子。
  - 当当前结点为空且栈不为空时，从栈中弹出一个结点，访问该结点并输出，然后将其右孩子赋值给当前结点。

中序遍历的非递归实现可以使用栈来模拟递归过程：

```
void inOrder(TreeNode* root) {
    stack<TreeNode*> s;
    TreeNode* curr = root;
    while (curr != NULL || !s.empty()) {
        while (curr != NULL) {
            s.push(curr);
            curr = curr->left;
        }
        if (!s.empty()) {
            curr = s.top();
            s.pop();
            visit(curr);
            curr = curr->right;
        }
    }
}
```

4.3 后序遍历

4.3.1 递归写法

思路：

- 若结点为空，则返回。
- 递归遍历左子树。
- 递归遍历右子树。
- 访问根结点。
- 返回。

后序遍历的递归实现如下:

```
void postOrder(TreeNode* root) {
    if (root == NULL) return;
    postOrder(root->left);
    postOrder(root->right);
```

```
    visit(root);
}
```

4.3.2 非递归写法

思路：

- 使用栈来模拟递归过程。(栈 + 指标，也就是区分是否第一次访问该结点)
- 初始化栈为空，当前结点为根结点，设置一个辅助指针 lastVisited 为 NULL。
- 当当前结点不为空或栈不为空时，重复以下步骤：
  - 当当前结点不为空时，将其入栈，然后将当前结点更新为其左孩子。
  - 当当前结点为空且栈不为空时，查看栈顶结点：
    - 如果栈顶结点的右孩子为空或已被访问过（即等于 lastVisited），则访问该结点并弹出，将 lastVisited 更新为该结点，当前结点设为 NULL。
    - 否则，将当前结点更新为栈顶结点的右孩子。

后序遍历的非递归实现可以使用栈来模拟递归过程：

```
void postOrder(TreeNode* root) {
    stack<TreeNode*> s;
    TreeNode* curr = root;
    TreeNode* lastVisited = NULL;
    while (curr != NULL || !s.empty()) {
        while (curr != NULL) {
            s.push(curr);
            curr = curr->left;
        }
        if (!s.empty()) {
            curr = s.top();
            if (curr->right == NULL || curr->right == lastVisited) {
                visit(curr);
                s.pop();
                lastVisited = curr;
                curr = NULL;
            } else {
                curr = curr->right;
            }
        }
    }
}
```

5 树与二叉树的等价性

5.1 树与二叉树之间的双射（左孩子-右兄弟表示法）

建立方法：

- 树 $\rightarrow$ 二叉树：对每个结点，将其第一个孩子作为左孩子；将每个结点的相邻兄弟用右孩子指针依次相连成右兄弟链；递归处理各子树。
- 二叉树 $\rightarrow$ 树：对每个结点，其左孩子是第一个孩子；从该孩子起沿右孩子指针依次枚举兄弟，作为当前结点的其余孩子；递归处理。

两步互为逆映射，故在有序根树与左孩子-右兄弟二叉树之间建立双射。

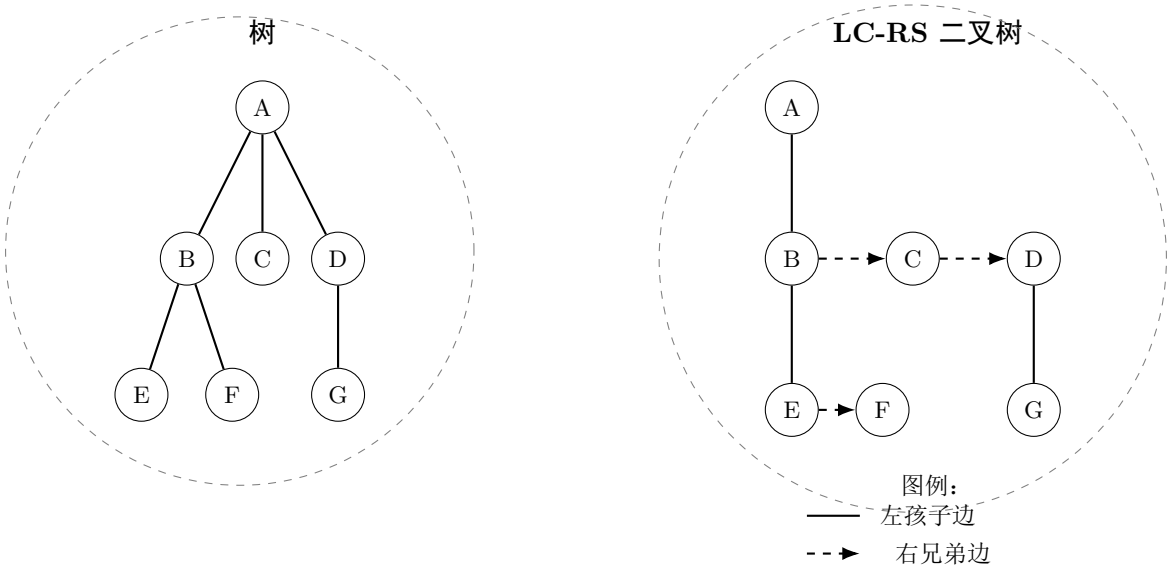

图 3: 树与二叉树（左孩子-右兄弟）的双射示意

5.2 遍历对应下的结点访问顺序

设树用“左孩子-右兄弟”法转为对应二叉树，则有：

- 树的前序遍历顺序 = 对应二叉树的前序遍历顺序。
- 树的后序遍历顺序 = 对应二叉树的中序遍历顺序。

以上图示中的示例（A 根，孩子依次为 B,C；B 的孩子为 D,E；C 的孩子为 F,G）：

- 树的前序：A, B, D, E, C, F, G
- 二叉树的前序：A, B, D, E, C, F, G
- 树的后序：D, E, B, F, G, C, A
- 二叉树的中序：D, E, B, F, G, C, A

注：层次遍历在该对应下不保持不变；上述“二叉树”均指左孩子-右兄弟表示得到的二叉树。

6 应用

6.1 哈夫曼树与编码

假设有一个字符串，其中只有字母 A，B，C，一般情况下我们用 ASCII 码，每个字符用了 8 个 bit 的空间，如何少用空间呢？

其实这里就只是三个不同的字符，所以每个用 2 个 bit 就行，如下表所示：

字符	编码
A	00
B	01
C	10

这个过程就叫做编码，如果知道每个字符出现的频率，我们可能做的更好。

6.1.1 编码：

数学上，一个编码的码字集合 C，原本的字符集 M，那么编码就是 M 到 C 的一个单射。计算机上，码字集合一般为 0,1 的数组川。

6.1.2 前缀编码：

所有码字都不是其他码字的前缀。它的优点是对给出的已编码串，可以唯一解出原文。

6.1.3 哈夫曼树

问题：给定一组字符与每个字符在文章中的使用次数，设计一种编码方案，使得编码后的文章长度最短。符合问题要求的编码就叫**哈夫曼编码**，用于生成哈夫曼编码的树就叫**哈夫曼树**。

其实每一个使用 0、1 编码的过程都可以看成是一棵二叉树，左子树代表 0，右子树代表 1。那么问题转化为寻找符合上述条件的最小带权长度 (weight path length, WPL) 的二叉树。

哈夫曼树的构造算法如下：

- 1. 对每个字符做一个结点，权重为使用频率，进而得到一个森林。
- 2. 若森林里的树的数目多于一棵时：
- 2a. 选出权 (如为单结点) 或叶子权值总和 (若为已合并过的树) 最小的两棵树。
- 2b. 构造一棵新树，其左、右孩子为上步选择的两棵树。
- 2c. 删掉那两颗树，并把新树加到森林中去。
- 3. 剩下的这棵树即为符合要求编码的对应哈夫曼树。(Rmk：答案不一定唯一，但 WPL 唯一)

哈夫曼算法的贪心选择性质：

> 设 C 为字符集，x 和 y 为其中两个频率最小的字符。则存在一棵最优哈夫曼树，其中 x 和 y 是最深的两个叶子结点。

Proof：设 T 为一棵最优哈夫曼树，a 和 b 为 T 中最深的两个叶子结点。
交换 x 与 a 的位置，得到新树 T'。则有：

$$\begin{aligned} WPL(T') - WPL(T) &= f(x)d_{T'}(x) + f(a)d_{T'}(a) - f(x)d_T(x) - f(a)d_T(a) \\ &= (f(a) - f(x))(d_T(x) - d_T(a)) \\ &\leq 0 \end{aligned}$$

同理，交换 y 与 b 的位置，得到新树 T''，则有：

$$\begin{aligned} WPL(T'') - WPL(T') &= f(y)d_{T''}(y) + f(b)d_{T''}(b) - f(y)d_{T'}(y) - f(b)d_{T'}(b) \\ &= (f(b) - f(y))(d_{T'}(y) - d_{T'}(b)) \\ &\leq 0 \end{aligned}$$

因此，$WPL(T'') \leq WPL(T') \leq WPL(T)$，且由于 T 为最优哈夫曼树，故 $WPL(T'') = WPL(T)$。
由此可见，存在一棵最优哈夫曼树，其中 x 和 y 是最深的两个叶子结点。
Q.E.D.