

# 二叉树的定义

---

二叉树 (Binary Tree) 是另一种树形结构

一颗二叉树是结点的一个有限集合，该集合或者为空，或者是由一个根节点加上两颗分别称为左子树和右子树的、互不相交的二叉树组成

## 特点

- 二叉树的特点是每个结点至多只有两颗子树（即二叉树中不存在度大于2的结点）；
- 并且二叉树的子树有左右之分，其次序不能任意颠倒

## 两种特殊形态的二叉树

- 满二叉树：一颗深度为k且有 $2^k - 1$ 的二叉树称为满二叉树
- 完全二叉树：深度为k的二叉树，前k - 1层是满的，最后一层从右往左连续缺失若干个结点

# 二叉树的性质

---

## 性质1

在二叉树的第i ( $i \geq 1$ ) 层至多有  $2^{i-1}$  个结点

## 性质2

深度为k的二叉树至多有  $2^k - 1$  个结点 ( $k \geq 1$ )

$$\sum_{i=0}^k 2^{i-1} = 2^0 + 2^1 + \dots + 2^{i-1} + \dots + 2^{k-1} \leftarrow$$

## 性质3

叶子结点（度为1）个数为 $n_0$ ，度为2的结点个数为 $n_1$ ，度为2的结点个数为 $n_2$

结论：

- $n_0 = n_2 + 1$
- 或 叶子结点数等于度为2的结点个数+1

证明：

- 总的结点个数 $n = n_0 + n_1 + n_2$
- 设B为总分支数，则 $n = B + 1 = 2 \cdot n_2 + n_1 + 1$
- 则 $n_0 + n_1 + n_2 = 2 \cdot n_2 + n_1 + 1$

- 化简:  $n_0 = n_2 + 1$

## 性质4

具有 $n$ 个结点的完全二叉树深度为  $\text{floor}(\log_2(n)) + 1$

### 证明

- 前 $h - 1$ 层是一个满二叉树, 共有 $2^{(h-1)} - 1$ 个结点
- $h$ 层的完全二叉树至多有 $2^h - 1$ 个结点
- 所以 $h$ 层完全二叉树结点数的范围为 $2^{(h-1)} - 1 \leq n \leq 2^h - 1$
- 两边同时加一, 不等式不变  $2^{(h-1)} \leq n < 2^h$
- 取对数  $h-1 \leq n < h$ , 又 $h$ 是整倍数
- 因此有 $h = \text{floor}(\log_2(n)) + 1$

## 性质5

如果对一颗有 $n$ 个结点的完全二叉树 (其深度为 $\text{floor}(\log_2(n)) + 1$ ) 的结点按层序编号 (从第1层到第 $\text{floor}(\log_2(n)) + 1$ 层, 每层从左到右编号), 则对任一结点 $i$  ( $1 \leq i \leq n$ ), 有

- 如果 $i = 1$ , 则结点 $i$ 是二叉树的根, 无双亲
- 如果 $i > 1$ , 则其双亲 $\text{PARENT}(i)$ 是结点 $\text{floor}(i / 2)$
- 如果 $2*i > n$ , 则结点 $i$ 无左孩子 (结点 $i$ 为叶子结点); 否则其左孩子 $\text{LCHILD}(i)$ 是结点 $2*i$
- 如果 $(2*i + 1) > n$ , 则结点 $i$ 无右孩子 (结点 $i$ 为叶子结点); 否则其右孩子 $\text{RCHILD}(i)$ 是结点 $2*i + 1$

# 二叉树的存储结构

## 顺序存储结构

用一组地址连续的存储单元依次自上而下、自左至右存储完全二叉树的结点元素;

即将完全二叉树上编号为 $i$ 的结点元素存储在落上定义的一维数组中下标为 $i-1$ 的分量中

这种顺序存储结构仅仅适用于完全二叉树; 因为在最坏的情况下, 一个深度为 $k$ 且只有 $k$ 个结点的单支树 (树中不存在度为2的结点) 却需要长度为  $2^k - 1$  的一维数组

## 链式存储结构

由二叉树的定义可知, 二叉树的结点由一个数据元素和分别指向其左、右子树的两个分支构成

则表示二叉树的链表中的结点至少包含三个域: 数据域、左、右指针域

在含有 $n$ 个结点的二叉链表中有 $n+1$ 个空链域

- $n$ 个结点的左右链域共有 $2n$ 个
- $n$ 个结点有 $n-1$ 个分支 (也就是链域存储的信息数)
- 所以空链域有  $2*n - (n-1) = n + 1$  个

```
typedef struct BinaryNode {
    ElementType data; //数据域
    struct BinaryNode *lchild; //左孩子
    struct BinaryNode *rchild; //右孩子
} BinaryNode, *BinaryTree;
```

## 遍历二叉树 (traversing binary tree)

按照某条搜索路径巡访树中的每个结点，使得每个结点均被访问一次，而且仅被访问一次

- 根节点D
- 左节点L
- 右节点R

### 先序遍历

DLR; 根左右

- 访问根节点
- 先序遍历左子树
- 先序遍历右子树

```
Status PreOrderTraverse(BinaryTree tree){
    if(tree == NULL){
        return FALSE;
    }else{
        printf("%2d",tree->data);
        PreOrderTraverse(tree->leftChild);
        PreOrderTraverse(tree->rightChild);
        return TRUE;
    }
}
```

中序后序遍历也都大致相似，不再赘述

- 中序遍历; LDR; 左根右
- 后序遍历; LRD; 左右根

### 创建二叉树

```
/**
 * 按照先序的次序构造二叉树; 空结点输入空格
 */
Status CreateBinaryTree(BinaryTree &tree){
    char ch;
    scanf(&ch);
```

```

    if(ch == ' '){
        tree = NULL;
    }else{
        tree = (BinaryTree)malloc(sizeof(Binarynode));

        if(tree == NULL){
            exit(0);
        }

        //生成根节点
        tree->data = ch;
        //构造左子树
        CreateBinaryTree(tree->leftChild);
        //构造右子树
        CreateBinaryTree(tree->rightChild);
    }

    return TRUE;
}

```

## 先序和中序构造二叉树

对于任意一颗树而言，前序遍历的形式总是

- [根节点, [左子树的前序遍历结果], [右子树的前序遍历结果]]

即根节点总是前序遍历的第一个结点

而中序遍历的形式总是

- [[左子树的前序遍历结果], 根节点, [右子树的前序遍历结果]]

只要我们在中序遍历中定位到根节点，那么我们就可以分别知道左子树和右子树的结点数目；

我们可以对应到前序遍历的结果中，对上述形式中的所有左右括号进行定位；可以递归地构造除左右子树，再将这两颗子树接到根结点的左右位置

## 线索二叉树

遍历二叉树得到某种次序的线性序列，这实质上是对一个非线性结构进行线性化操作，使这些结点（除第一个和最后一个外）在这些序列中有且仅有一个直接前驱和直接后继；

```

typedef struct Node{
    ElementType      data;          //数据域
    int               LTag;          //LTag=0指示结点的左孩子，否则指示结点的前驱
    struct Node       leftChild;    //左指针域；具体作用取决于LTag
    int               RTag;          //RTag=0指示结点的右孩子，否则指示结点的后继
    struct Node       rightChild;   //右指针域；具体作用取决于RTag
} ThreadedNode, *ThreadedBinaryTree;

```

## 线索

以这种结点结构构成的二叉链表作为二叉树的存储结构，叫做线索链表，其中指向结点前驱和后继的指针，叫做线索；加上线索的二叉树叫做线索二叉树（Threaded Binary Tree）

## 线索化

对二叉树以某种次序遍历使其变为线索二叉树的过程叫做线索化

## 头结点

为了方便起见，在二叉树的线索链表上也添加一个头结点，并令其lchild域的指针指向二叉树的根结点，其rchild域的指针指向某种次序的最后一个结点

## 非递归的中序遍历

```
#define LINK      1
#define THREAD    0

/**
 * 有头结点；
 * lchild指向根结点
 * rchild指向中序遍历序列的最后一个结点
 */

Status InOrderTraverse(ThreadedBinaryTree headNode){
    //头结点的lchild指向根结点；令current指向二叉树的根结点
    ThreadedBinaryTree current = headNode->leftChild;

    //循环结束条件；
    //头结点的rchild指向中序遍历序列的最后一个结点
    //如果current等于头结点，说明已经访问了最后一个结点
    while(current != headNode){

        //左转到最左结点
        while(current->LTag == LINK){
            current = current->leftChild;
        }

        printf("%2d",current->data);

        //如果右标志为线索且不等于头结点，那么访问
        while((current->RTag == THREAD) && (current->rightChild != headNode)){
            current = current->rightChild;
            printf("%2d",current->data);
        }

        current = current->rightChild;
    }
}
```