数据结构

本周范围：二叉树、优先队列（堆）、霍夫曼树以及相关编程题

张天戈

4）对完全二叉树按从上到下、从左到右的顺序依次编号 $1, 2, \cdots, n$，则有以下关系：

① 当 $i > 1$ 时，结点 i 的双亲的编号为$\lfloor i/2 \rfloor$，即当 i 为偶数时，其双亲的编号为 $i/2$，它是双亲的左孩子；当 i 为奇数时，其双亲的编号为$(i-1)/2$，它是双亲的右孩子。

② 当 $2i \le n$ 时，结点 i 的左孩子编号为 $2i$，否则无左孩子。

③ 当 $2i+1 \le n$ 时，结点 i 的右孩子编号为 $2i+1$，否则无右孩子。

④ 结点 i 所在层次（深度）为 $\lfloor \log_2 i \rfloor + 1$。

1. 含有 n 个节点的三叉树的最小高度是多少？

补充：树的遍历及代码

先序 **1. 先序遍历** 根左右

先序遍历（PreOrder）的操作过程如下。

若二叉树为空，则什么也不做；否则，

1）访问根结点；

2）先序遍历左子树；

3）先序遍历右子树。

对应的递归算法如下：

```
void PreOrder(BiTree T){
    if(T!=NULL){
        visit(T);                  //访问根结点
        PreOrder(T->lchild);       //递归遍历左子树
        PreOrder(T->rchild);       //递归遍历右子树
    }
}
```

中序 **2. 中序遍历** 左根右

中序遍历（InOrder）的操作过程如下。

若二叉树为空，则什么也不做；否则，

1）中序遍历左子树；

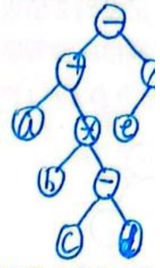
2）访问根结点；

3）中序遍历右子树。

对应的递归算法如下：

```
void InOrder(BiTree T){
    if(T!=NULL){
        InOrder(T->lchild);        //递归遍历左子树
        visit(T);                  //访问根结点
        InOrder(T->rchild);        //递归遍历右子树
    }
}
```

例：分析



后序 **3. 后序遍历** 左右根

后序遍历（PostOrder）的操作过程如下。

若二叉树为空，则什么也不做；否则，

1）后序遍历左子树；

2）后序遍历右子树；

3）访问根结点。

对应的递归算法如下：

```
void PostOrder(BiTree T){
    if(T!=NULL){
        PostOrder(T->lchild);      //递归遍历左子树
        PostOrder(T->rchild);      //递归遍历右子树
        visit(T);                  //访问根结点
    }
}
```

例2：求树的深

```
int treeDepth(
{ if(T==NU
    { return
  else
左  int l=
右  int r=
根  return
  }
}
```

5. 层次遍历

图 5.8 所示为二叉树的层次遍历，即按照箭头所指方向，按照 1, 2, 3, 4 的层次顺序，对二叉树中的各个结点进行访问。

要进行层次遍历，需要借助一个队列。先将二叉树根结点入队，然后出队，访问出队结点，若它有左子树，则将左子树根结点入队；若它有右子树，则将右子树根结点入队。然后出队，访问出队结点……如此反复，直至队列为空。

算法思想：①初始化一个辅助队列 ②根结点入队 ③若队列非空，则队头结点出队，访问该结点，并将其左、右孩子插入队尾。④重复③直至队列为空

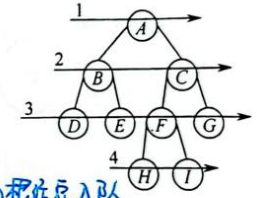


图 5.8　二叉树的层次遍历

二叉树的层次遍历算法如下：

```
void LevelOrder(BiTree T){
    InitQueue(Q);                      //初始化辅助队列
    BiTree p;
    EnQueue(Q,T);                      //将根结点入队
    while(!IsEmpty(Q)){                //队列不空则循环
        DeQueue(Q, p)                  //队头结点出队
        visit(p);                      //访问出队结点
        if(p->lchild!=NULL)
            EnQueue(Q,p->lchild);      //左子树不空，则左子树根结点入队
        if(p->rchild!=NULL)
            EnQueue(Q,p->rchild);      //右子树不空，则右子树根结点入队
    }
}
```

2. 二叉树的层次遍历序列为 ABCDEFGHIJ，中序遍历序列为 DBGEHJACIF，写出该二叉树的前序遍历序列。

堆的公式化定义

堆的定义如下，n 个关键字序列 L[1...n] 称为堆，当且仅当该序列满足：

① L(i)>=L(2i) 且 L(i)>=L(2i+1) 或

② L(i)<=L(2i) 且 L(i)<=L(2i+1)　$(1 \le i \le \lfloor n/2 \rfloor)$

可以将该一维数组视为一棵完全二叉树，满足条件①的堆称为大根堆（大顶堆），大根堆的最大元素存放在根结点，且其任一非根结点的值小于等于其双亲结点值。满足条件②的堆称为小根堆（小顶堆），小根堆的定义刚好相反，根结点是最小元素。图 8.4 所示为一个大根堆。

3. 简述什么是最大堆和最小堆？请画出将序列{11, 9, 3, 6, 7, 4, 5, 10, 8, 1, 2}存到一个完全二叉树中的情形，画出将其调整成最小堆的过程。

01 本周习题

4. 一个最大堆为（66，37，41，30，25，40，35，18），依次从中删除两个元素，写出最后得到的堆。

01 本周习题

1. 哈夫曼树的定义

在许多应用中，树中结点常常被赋予一个表示某种意义的数值，称为该结点的权。从树的根到任意结点的路径长度（经过的边数）与该结点上权值的乘积，称为该结点的带权路径长度。树中所有叶结点的带权路径长度之和称为该树的带权路径长度，记为

$$\boxed{\mathrm{WPL}=\sum_{i=1}^{n} w_i l_i}$$

式中，w_i 是第 i 个叶结点所带的权值，l_i 是该叶结点到根结点的路径长度。

在含有 n 个带权叶结点的二叉树中，其中带权路径长度（WPL）最小的二叉树称为哈夫曼树，也称最优二叉树。例如，图 5.34 中的 3 棵二叉树都有 4 个叶子结点 a, b, c, d，分别带权 7, 5, 2, 4，它们的带权路径长度分别为

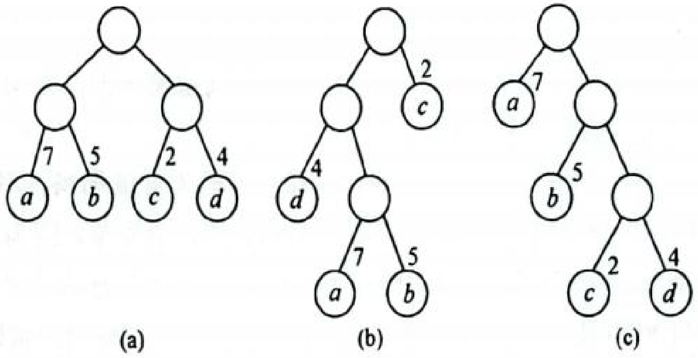


图 5.34　具有不同带权长度的二叉树

(a) WPL = 7×2 + 5×2 + 2×2 + 4×2 = 36。

(b) WPL = 4×2 + 7×3 + 5×3 + 2×1 = 46。

(c) WPL = 7×1 + 5×2 + 2×3 + 4×3 = 35。

其中，图 5.34(c)树的 WPL 最小。可以验证，它恰好为哈夫曼树。

2. 哈夫曼树的构造

给定 n 个权值分别为 $w_1, w_2, \cdots, w_n$ 的结点，构造哈夫曼树的算法描述如下：

1）将这 n 个结点分别作为 n 棵仅含一个结点的二叉树，构成森林 F。

2）构造一个新结点，从 F 中选取两棵根结点权值最小的树作为新结点的左、右子树，并且将新结点的权值置为左、右子树上根结点的权值之和。

3）从 F 中删除刚才选出的两棵树，同时将新得到的树加入 F 中。

4）重复步骤 2）和 3），直至 F 中只剩下一棵树为止。

5. 有一份电文中共使用 6 个字符：A、B、C、D、E、F，它们的出现频率依次为 10、6、5、2、15、4，试画出对应的赫夫曼树（请按左子树根节点的权小于等于右子树根节点的权的次序构造，左 0 右 1），并求出每个字符的赫夫曼编码。

6. 二叉树的带权路径长度（WPL）是二叉树中所有叶结点的带权路径长度之和，给定一棵二叉树 T，采用二叉链表存储，节点结构为：

left	weight	right

其中叶节点的 weight 域保存该结点的非负权值。设 root 为指向 T 的根节点的指针，设计求 T 的 WPL 的算法。

要求：给出算法的基本设计思想，并使用 C 或 C++语言，给出二叉树结点的数据类型定义，实现算法，关键之处给出注释，最后分析所编写代码的运行时间复杂度与空间复杂度。

01 本周习题

（1） 算法的设计思想：

递归遍历二叉树，利用一个参数同时对深度进行计数。叶结点的带权路径长度=该结点的weigth值*该结点的深度。每个叶结点的带权路径长度都可以求出，二叉树的WPL值=树中全部叶结点的带权路径长度之和=根结点左子树中全部叶结点的带权路径长度之和+根结点右子树中全部叶结点的带权路径长度之和。递归进行求和，即可求出二叉树的带权路径长度。

（2） 算法中使用的二叉树结点的数据类型定义如下：

```
typedef struct BTnode
{
	unsigned int weight; //结点的非负权值
	struct BTnode * lchild，  * rchild; //左右指针
}BTnode;
```

（3） 算法实现：

```
int main()
{
    return WPL(root,0); //初始化深度，调用 WPL 函数
}
int WPL(BTnode * root,int d) //其中 d 为结点深度
{
    if(root->lchild==NULL&&root->rchild==NULL)//root 为叶子结点
        return (root->weight * d); //返回该叶子结点的带权路径长度
    else
        return(WPL(root-> lchild,d+1)+WPL(root->rchild,d+1));
    /*返回左右子树中全部叶结点的带权路径长度之和*/
}
```

Q&A