|  |
| --- |
| 哈尔滨工业大学(深圳) |
| **《数据结构》实验报告** |
|  |
| 实验三  树型结构及其应用（根据不同实验内容做修改）  学 院:   |  |  | | --- | --- | | 姓 名: |  | | 学 号: |  | | 专 业: |  | | 日 期: |  | |

# 问题分析

（1）题目要求按照层序遍历建立一个二叉树，并且输出其前序中序后序遍历产生的序列。根据题目要求，若将二叉树补上空位，补为满二叉树后，假设某个节点按层序遍历后的下标为，则其左子树（若不存在即为空树）按照层序遍历后的下标为，右子树下标为。因此可以采用数组方式来实现层序遍历二叉树。而前序遍历即为先打印根节点再依次前序遍历左右子树；中序遍历即先遍历左子树，再打印根节点，最后中序遍历右子树；后序遍历即先后序遍历左右子树，再打印根节点。

（2）题目要求对给定的二叉树求其最大路径和，即所有路径中各节点的权值之和的最大值。此时只需记录每层节点的各个最大路径和的最大值，从底层开始，每层进行递归到根节点即可求出整棵树的最大路径和。

（3）题目要求对给定的二叉树求其所有左子叶的权重之和。从根节点开始，对左右子树求左子叶的和，并且用一个变量记录左子叶的和，如此递归即可求得所有左子叶的和。

（4）题目要求将给定的二叉树进行翻转。对根节点，若其为叶子节点，则达到递归终点，无需操作；若其不为叶子节点，则其必有左右子树其中之一，对其左右子树分别进行调用交换函数后，将左右子树（若其中一个不存在，则为空树）进行调换。递归结束后即得到这个二叉树的翻转。

# 二、详细设计

## 2.1 设计思想

（1）层序遍历建立二叉树：

用一个树节点数组记录补充之后的满二叉树的节点。若给定数据为-1，表明根节点为空，则此树为空树，返回。树不为空时，首先先将所有节点放入数据，形成一个补充后的满二叉树。若,表明此节点为空节点，令此节点为；否则对此节点赋值。对所有节点赋值后，建立节点之间的联系，若，则此节点为空节点，不建立联系；若此节点不为空，若，则其左子树不为空，此时令为的左子树，若，则其右子树不为空，此时令为的右子树。由于最后一层叶子节点的标志都大于，叶子节点不存在左右子树，无需建立关系，则当（对取整）时，停止循环。

前序遍历：

若树不为空，则先打印根节点，再前序遍历左右子树。若树为空，则达到递归终点，直接结束程序。

中序遍历：

若树不为空，先中序遍历左子树，再打印根节点，最后中序遍历右子树。若树为空，则达到递归终点，直接结束程序。

后序遍历：

若树不为空，则先后序遍历左右子树，再打印根节点。若树为空，则达到递归终点，直接结束程序。

（2）若输入的节点的左右子树为空，则其为叶子节点，返回值；若不为叶子节点，则对左右节点的maxPathSum函数返回值进行比较，返回较大的值与此节点的值的和。

（3）建立函数时设立一个sign值，sign值为1表示此节点为根节点或者左子树节点，sign值为0表示此节点为右子树节点。若左右子树为空且sign为1，则其为左节点或者根节点，返回此节点的值；若左右子树为空且sign为0，则其为右节点，不返回其值，返回0。若左右子树不全为空时：若左子树不为空，则将sum加上其左子树的左子叶权重之和，并将向下传递的sign值设为1；若右子树不为空，则将sum加上其右子树的左子叶权重之和，并将向下传递的sign值设为0。最后返回sum值。

（4）首先设置一个中间节点t用于交换左右子树的中间节点。若此树为空，则释放t后返回。若此树不为空：若此节点左右子树都为空，则此节点为叶子节点，无需操作，释放t后返回此节点；若此节点左右子树不为空，则此节点不为叶子节点，对其左右子树分别调用此翻转函数后，对此节点的左右子树进行调换，释放t后返回此节点即可、

## 2.2 存储结构及操作

(1) 存储结构（一般为自定义的数据类型，比如单链表，栈等。）

树节点TreeNode，内包含树节点的id（int），树节点的值val（int），指向左子树的指针\*left（struct TreeNode），指向右子树的指针\*right（struct TreeNode）

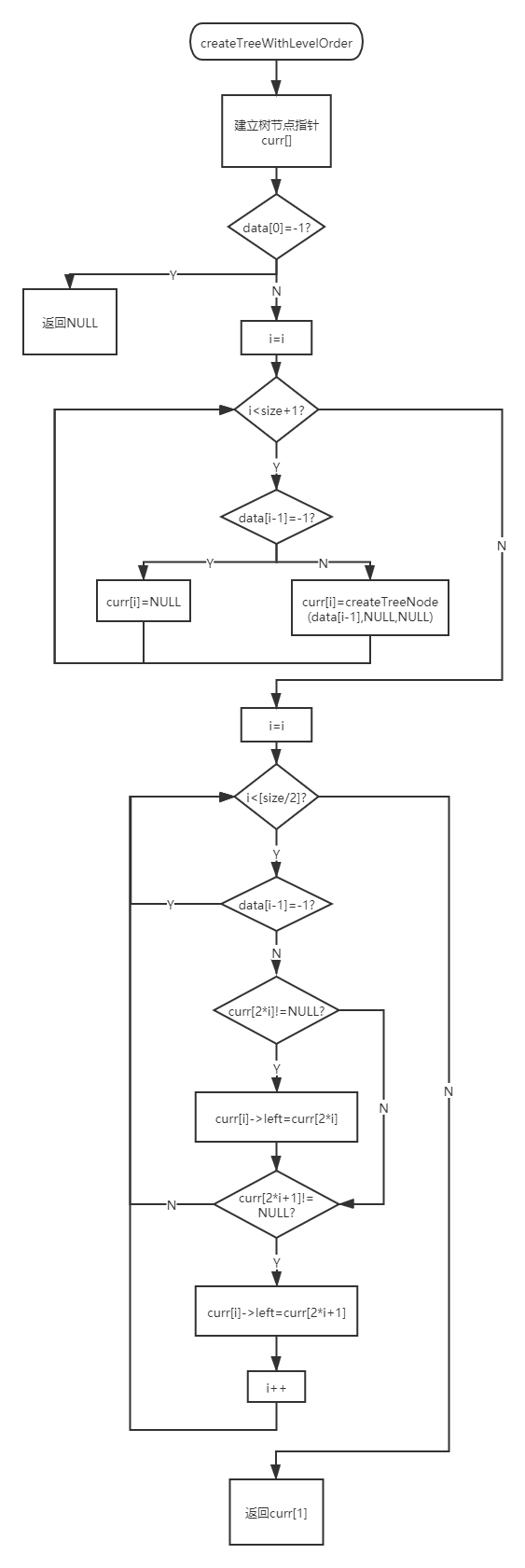
(2) 涉及的操作（一般为自定义函数，可不写过程，但要注明该函数的含义。）

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 参数 | 功能 | 返回值 |
| createTreeWithLevelOrder | 层序遍历读入的数组data[]  数组长度size | 通过层序遍历  构建二叉树 | 树节点指针 |
| preOrderTraverse | 树节点指针root | 前序遍历二叉树 |  |
| inOrderTraverse | 树节点指针root | 中序遍历二叉树 |  |
| postOrderTraverse | 树节点指针root | 后序遍历二叉树 |  |
| maxPathSum | 树节点指针root | 求二叉树的最大路径和 | 最大路径和（整型值） |
| sumOfLeftLeaves | 树节点指针root  标记值sign | 求二叉树的所有左子叶权重之和 | 左子叶权重之和  （整型值） |
| invertTree | 树节点指针root | 求二叉树的镜像 | 树节点指针 |

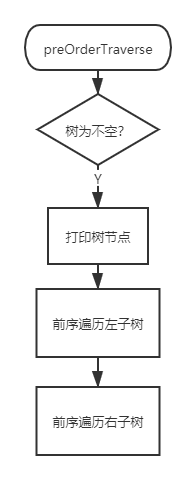
## 2.3 程序整体流程

画出整体流程，及核心算法流程。

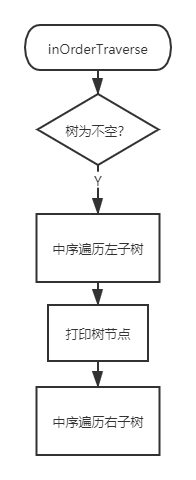
（1）层序遍历建立二叉树



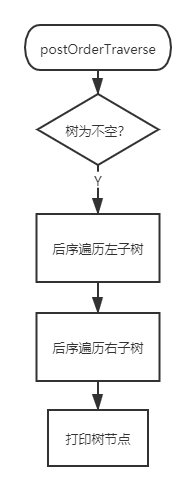
（2）前序遍历二叉树



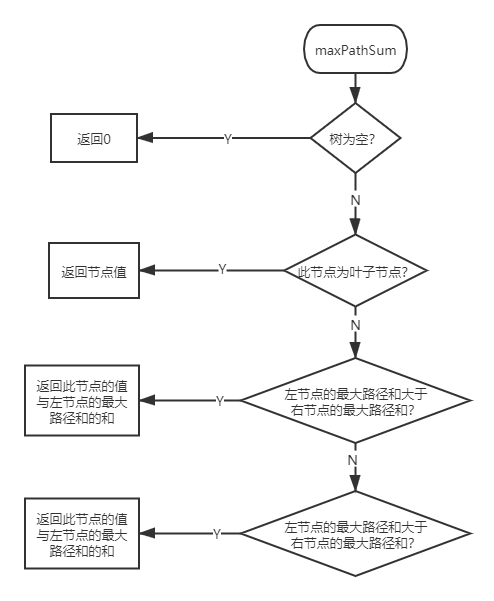
（3）中序遍历二叉树



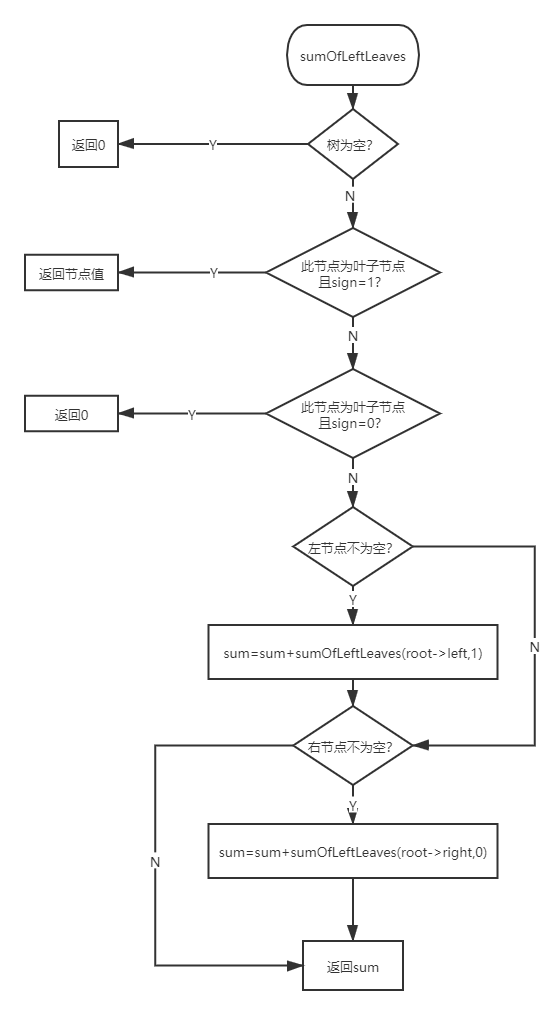
（4）后序遍历二叉树



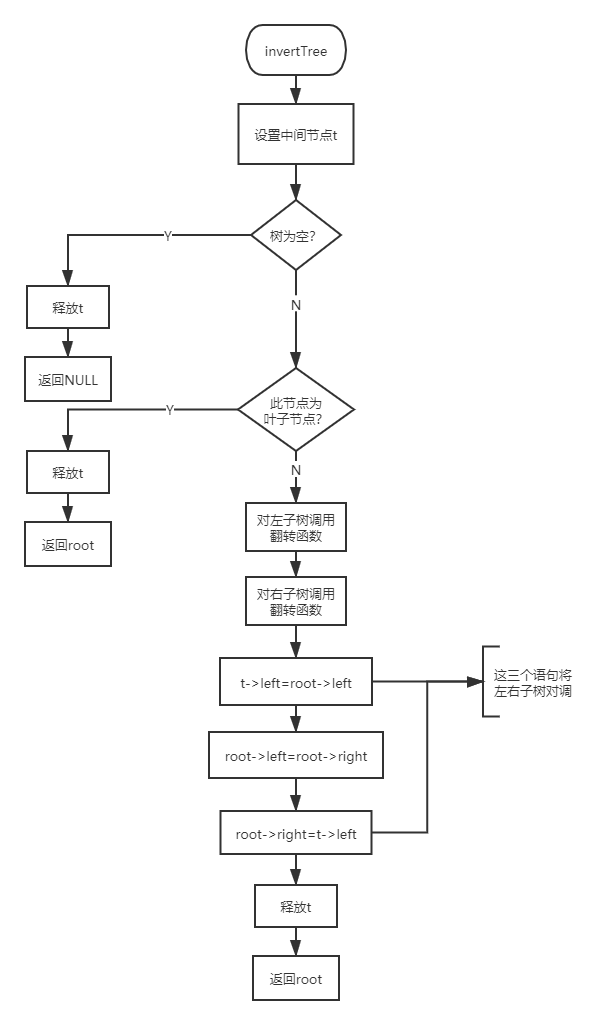
（5）求最大路径和



（6）求左子叶权重之和



（7）翻转二叉树



# 三、用户手册

程序接受数据的方式为从一个名为“test.txt”的程序中读取

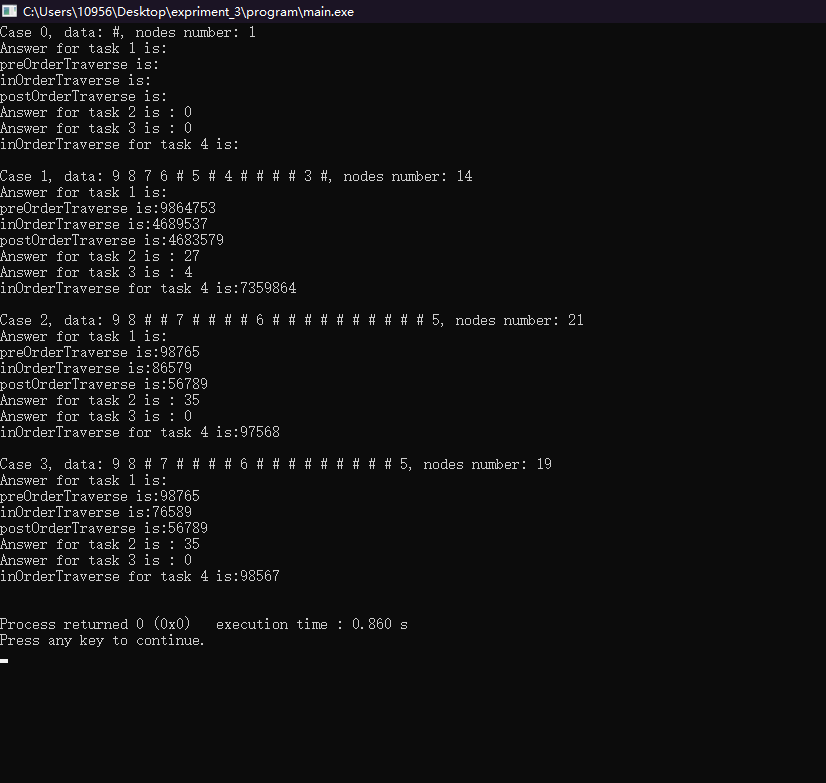
输入格式如下：

（1）首先输入树补为完全二叉树后的层序遍历序列的长度值，然后换行

（2）按照层序遍历的方式，将补为完全二叉树的层序遍历序列打入文件中，空子树的位置打入#，其余位置打入树节点的值，然后换行

（3）若需输入其他树，则重复（1）（2）操作。若不再输入其他树，则结束输入，关闭文件。

# 四、结果



# 五、总结

该实验涉及到的数据结构和算法，以及遇到的问题和收获。

1. 数据结构：树，数组
2. 算法：
   1. 设计通过层序遍历构建二叉树的程序时，运用了curr[i]的左右孩子节点分别为curr[2\*i]，curr[2\*i+1]，即通过数组来通过层序遍历来构建二叉树
   2. 前中后序遍历时，运用了前序中序后序遍历的递归算法
   3. 求最大路径和时运用了深度优先的比较算法
   4. 求左子叶权重和时运用了递归的方法，并且设置了一个向下传递的sign值来表示下一个节点为左节点或右节点，从而能够轻松地记录左节点的值
   5. 通过运用后序的递归算法，先对左右子树进行调用翻转函数，再对此节点的左右孩子节点进行对调。
3. 问题：
   1. 在层序遍历构建二叉树时树节点坐标和读入数据坐标未对齐，导致二叉树构建错误
   2. 在前序中序后序遍历二叉树时忘记判断树是否为空，即未设置递归终点，程序陷入死循环
   3. 在层序遍历构建二叉树时忘记判断左右子树是否为空就建立关系，导致树的建立错误
4. 收获
   1. 熟悉了树的前序中序后序算法并且将其运用于其他算法中
   2. 熟悉了递归算法的设计
   3. 熟悉了使用数组进行辅助来建立二叉树的方法