|  |
| --- |
| 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  学 院: 计算机科学与技术学院   |  |  | | --- | --- | | 姓 名: | 苏亦凡 | | 学 号: | 200111229 | | 专 业: | 计算机科学与技术 | | 日 期: | 2021-04-16 | |

# 一、问题分析

(1) 题目1

按层次遍历建立二叉树，并输出该二叉树的前序遍历、 中序遍历和后序遍历的序列。需考虑判空以及空节点之后的占位符的处理方法，以及如何避免用户的非法输入对程序产生影响。

(2) 题目2

给定一棵二叉树，路径定义为从树的根节点到叶子结点的任意路径，求取该二叉树的最大路径和，路径和定义为一条路径中各节点的权值之和。需要考虑判空以及对边界情况的处理，考虑采用哪种遍历方式。

(3) 题目3

给定一棵二叉树，求取该二叉树的所有左子叶权重之和，左子叶被定义为二叉树叶子结点中属于左子树的节点。需考虑判空及边界情况的处理，考虑采用哪种遍历方式。

(4) 题目4

给定求取该树的镜像并输出翻转后二叉树的中序遍历。需考虑判空并考虑采用哪种遍历方式。

# 二、详细设计

## 2.1 设计思想

(1)题目1：

考虑到层次遍历的特性，需使用辅助队列并利用队列的First in First out 特性建立二叉树。先建立根节点并令根节点入队。接下去每次循环令队列元素出队一次，然后读取下两个数组元素，建立该节点的左右孩子节点，循环直到队列为空或数组的下标超限。

为了避免多余的操作，在节点建立时将其两个指针域初始化为NULL。

需要注意的是遇到占位的-1时虽然无需申请空间建立节点，当仍需要使NULL进入队列占位，当遇到NULL出队时，为避免对NULL访问域导致程序出错，可采取以下逻辑：易知接下去两个元素必为-1，可令数组下标连续自增两次，将两个NULL连续入队。

为避免数组下标越界，在每次访问数组元素前判断数组下标是否越界，若越界则结束循环。

对于前序、中序、后序遍历，采用递归的思想，并设置当输入的根节点为NULL时返回即可。

(2)题目2：

在模板中给出的函数原型中，函数的输入除了根节点外还有一个整形的变量sum，且在main函数中调用此函数时传入的sum值为0。因此可将sum的意义定义为从原树的根节点到该子树根节点的路径长。求取路径长应采取深度优先的遍历方法。

(3)题目3：

该问题的关键是遍历方法的选取以及左子叶的判定方法。

对于子叶的搜素，我们可选择优先遍历。

左子叶的判定可采取如下方法：对某节点的左孩子节点不为空，且该左孩子节点的左右孩子均为空，则该左孩子节点为左子叶节点。

在递归时利用如下性质：若一个节点为某树子树的左子叶，则该节点也为该树的左子叶。

考虑到效率问题可使用三目运算符，考虑到程序的易读性则使用if的判断分支。

(4)题目4：

该问题的关键是如何设计递归函数。

利用镜像树与原数左右子树对调且取镜像，可设计递归函数如下：建立根节点，值与原数相同，该根节点的左指针指向原树根子树的右子树的镜像树；右指针指向原树根子树的左子树的镜像树。

## 2.2 存储结构及操作

(1) 存储结构：

主要结构：树的二叉链表储存形式。辅助结构：队列的链表形式。

typedef struct TreeNode

{

int id;

int val;

struct TreeNode \*left;

struct TreeNode \*right;

} TreeNode, \*TreeNodePtr;

typedef struct ListNode

{

struct TreeNode \*node;

struct ListNode \*next;

} ListNode, \*ListNodePtr;

typedef struct Queue

{

ListNodePtr dummyHead;

ListNodePtr tail;

int size;

} \*QueuePtr;

(2) 涉及的操作：

队列操作：

ListNodePtr createListNode(TreeNodePtr node, ListNodePtr next);

TreeNodePtr createTreeNode(int val, TreeNodePtr left, TreeNodePtr right);

QueuePtr InitQueue();

void EnQueue(QueuePtr queue, TreeNodePtr node);

void DeQueue(QueuePtr queue);

bool QueueEmpty(QueuePtr queue);

TreeNodePtr GetHead(QueuePtr queue);

树操作：

void destoryTree(TreeNodePtr root);

TreeNodePtr createTreeWithLevelOrder(int \*data, int size);

void preOrderTraverse(TreeNodePtr root);

void inOrderTraverse(TreeNodePtr root);

void postOrderTraverse(TreeNodePtr root);

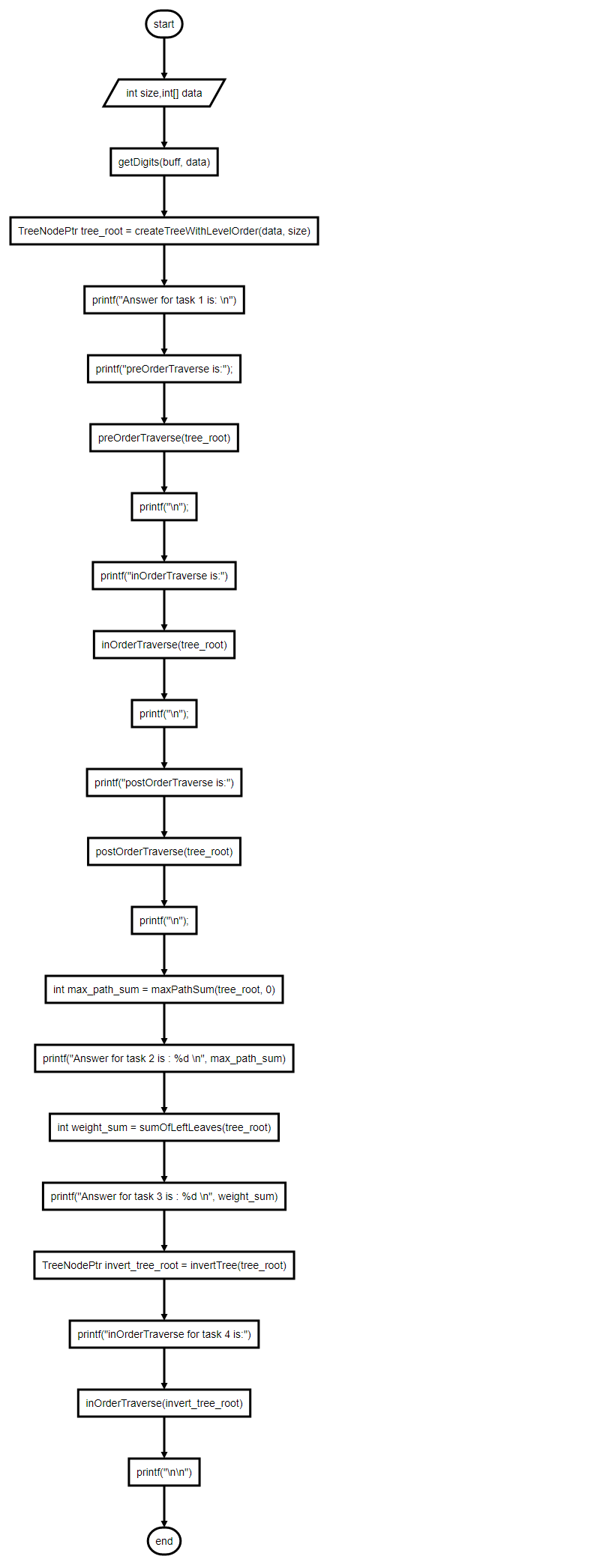
int maxPathSum(TreeNodePtr root, int sum);

int sumOfLeftLeaves(TreeNodePtr root);

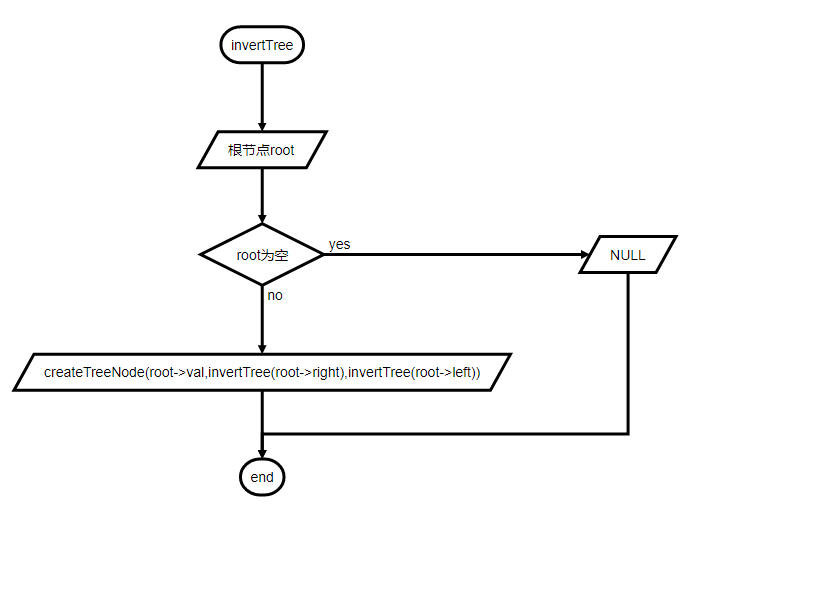
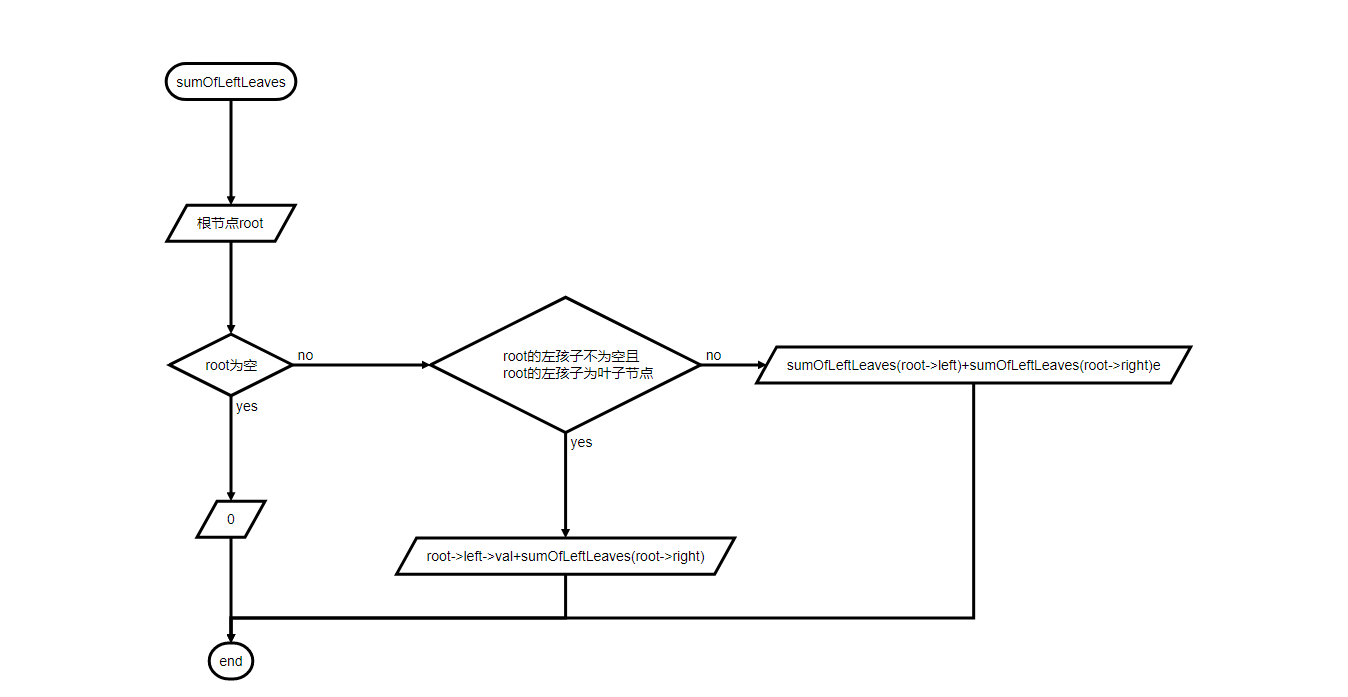
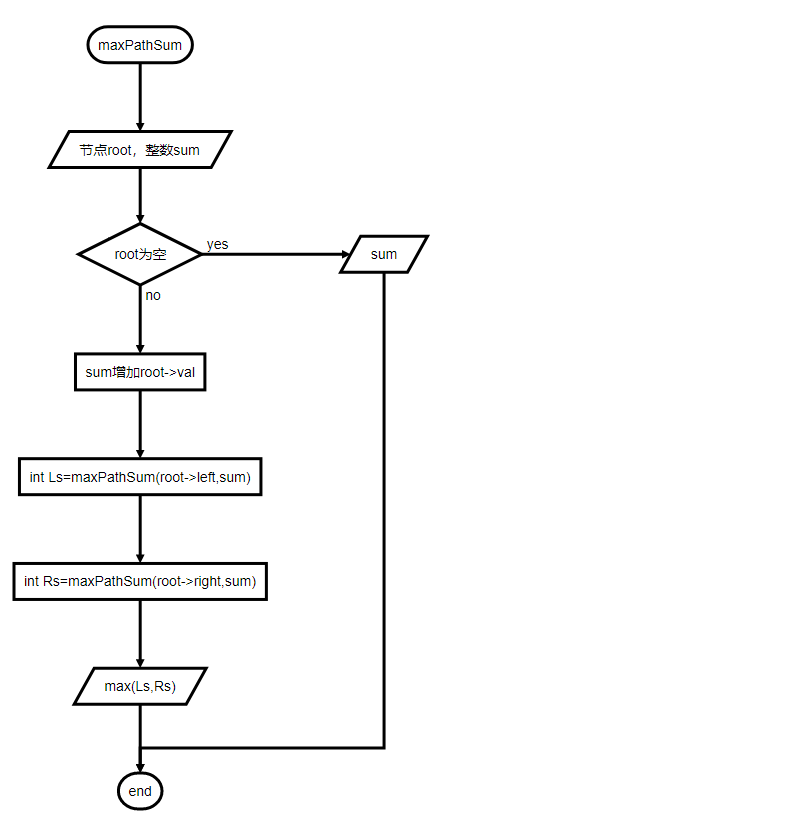
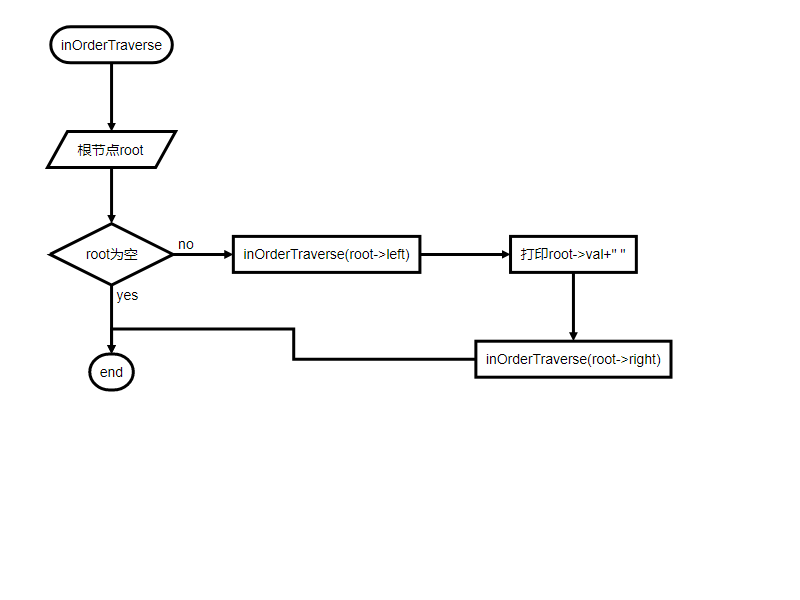
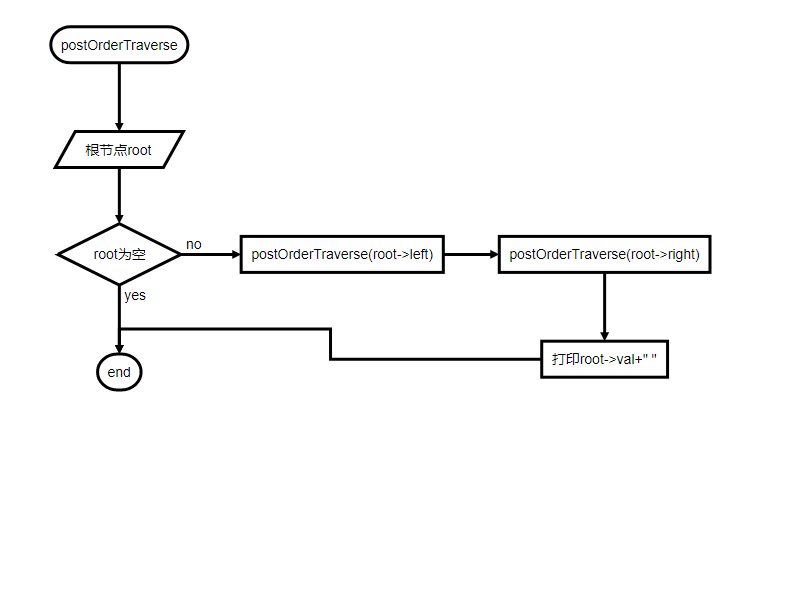
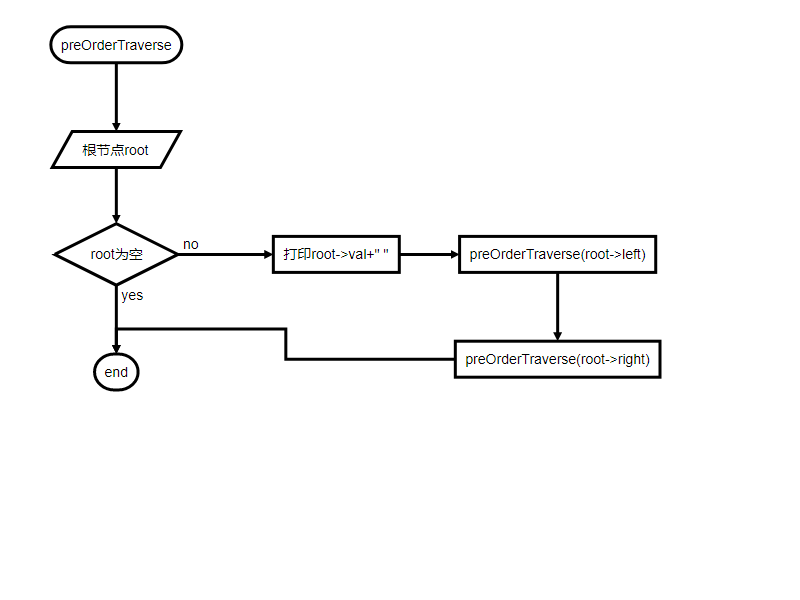
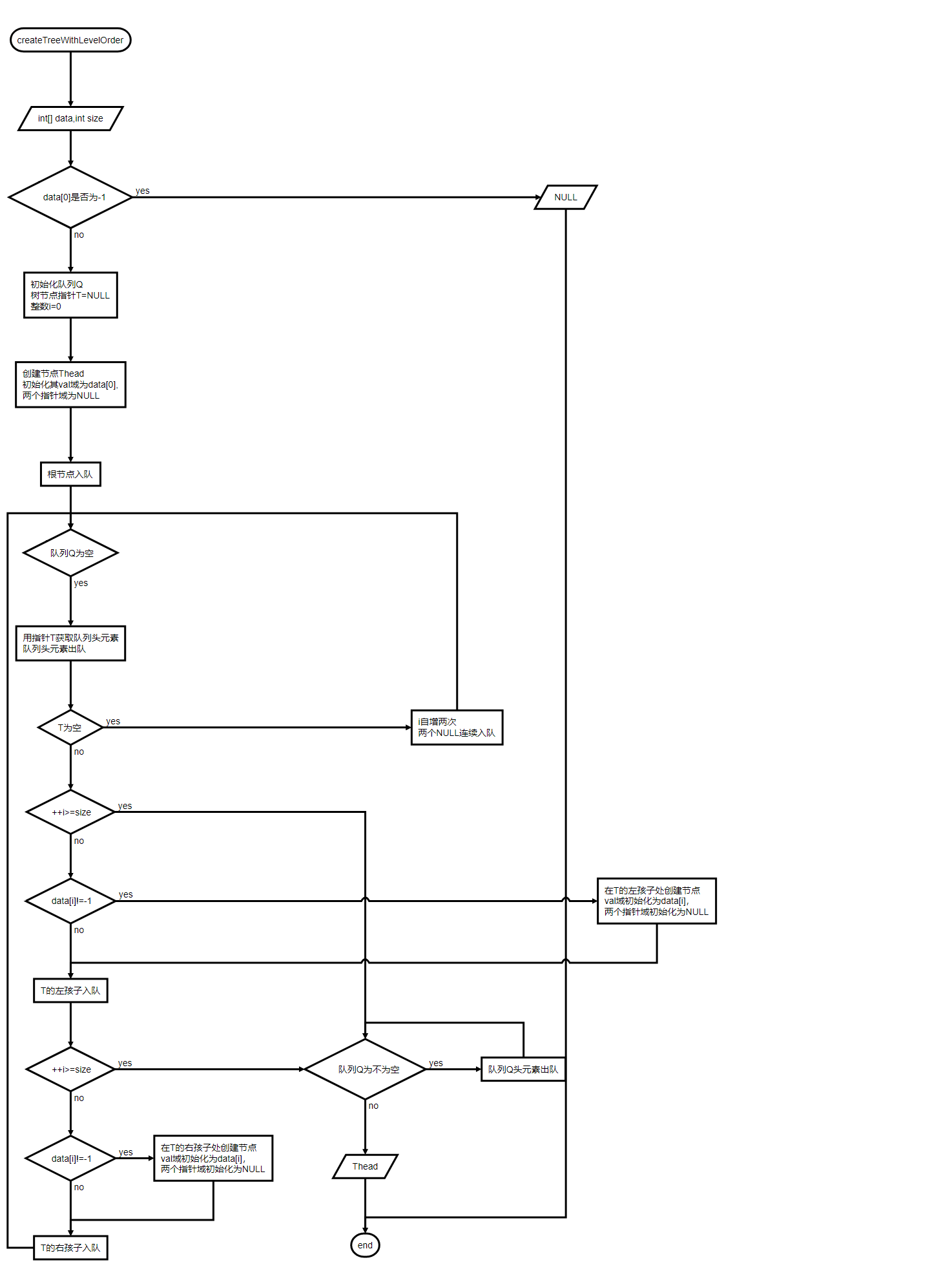
TreeNodePtr invertTree(TreeNodePtr root);

## 2.3 程序整体流程

源码及原图保存在flowchart文件夹中。

整体流程：仅画出对一组数据的处理流程

函数算法流程：



# 三、用户手册

(1)输入数据的方式：

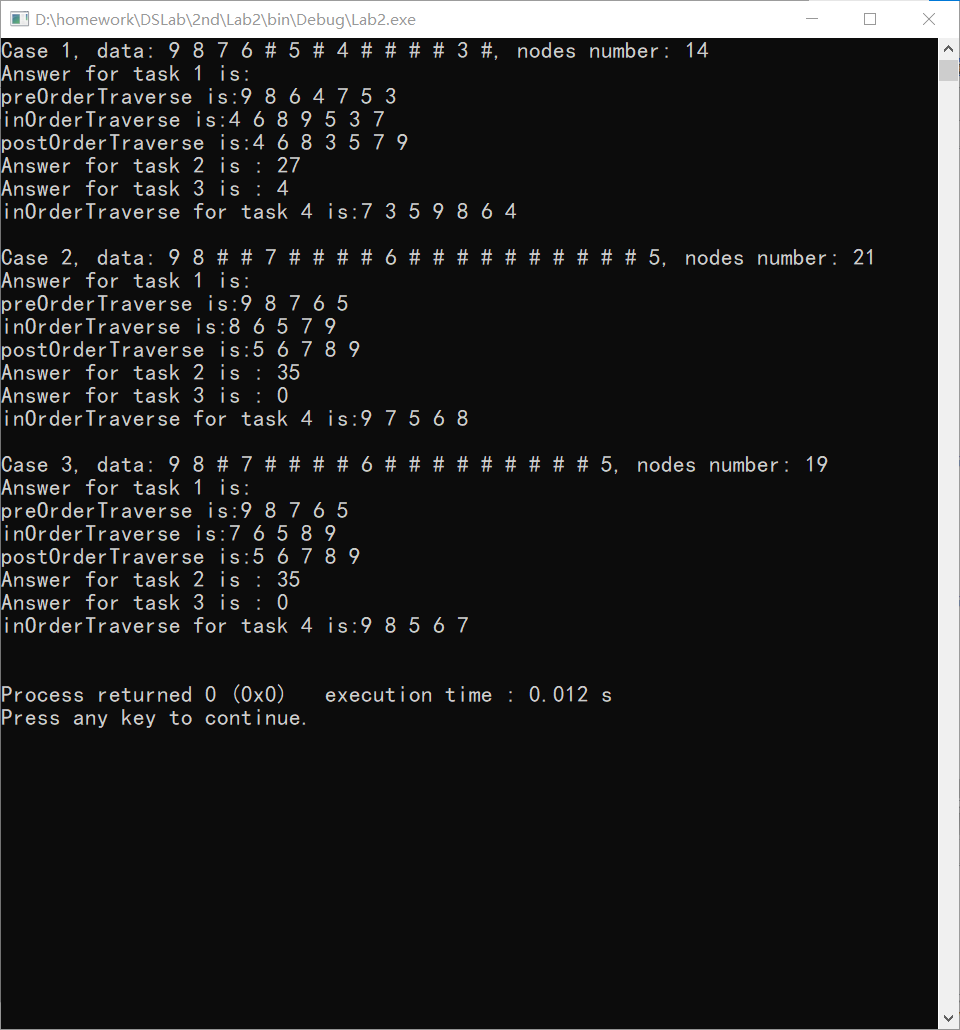
使用同级目录的test.txt文件进行输入。规范为：

1. 第一行为输入的序列长度。
2. 第二行为层序遍历序列，用#表示空。
3. 若有多组输入换行后重复以上两个步骤。

(2)实现各种功能的操作方式：

1. 按层次遍历建立链表：  
   调用createTreeWithLevelOrder函数，输入为转换后的层次遍历序列整形数组名，整形的序列长度，返回值为指向根节点的树节点指针。
2. 按前序、中序或后序遍历树并输出遍历序列：  
   分别调用preOrderTraverse, inOrderTraverse或postOrderTraverse，函数的输入为指向根节点的树节点指针。
3. 求取二叉树的最大路径和：  
   调用maxPathSum函数，函数的输入为指向根节点的树节点指针，整形0；输出为整形的最大路径和。
4. 求二叉树的所有左子叶权重之和：  
   调用sumOfLeftLeaves函数，函数的输入为指向根节点的树节点指针；输出为整形的左子叶权重和。
5. 求取二叉树的镜像：  
   调用invertTree函数，函数的输入为指向根节点的树节点指针；输出为指向镜像二叉树根节点的树节点指针。

# 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。

该实验主要涉及树这一数据结构，并使用二叉链表进行储存；同时使用了链表形式的队列作为辅助。算法主要有BFS与DFS，并大量使用递归的思想。

本次实验的难点主要在于任务一的建立二叉树。该任务无法向其他问题一样用递归解决，故操作过程相对繁琐，很容易出现纰漏，如：数组越界、对NULL访问域等、忘记释放队列内存等等。故这项任务很考验耐心与细致程度，当然，若发生错误亦考察debug的能力。

另一个困难仍然出现在实验报告中，由于实验的总代码量不小，故流程图的绘制也是相对繁琐，我采用了markdown基于flowchart.js的流程图绘制方法，问题也得到解决。Markdown源码以及生成的图片保存在flowchart文件夹中。

本次实验是对树的相关知识的回顾与复习，加深了对与树相关的算法的理解，锻炼了动手编程的能力，总体来说，收获颇丰。