华南农业大学信息学院

综合性、设计性实验

起止日期:2025/06/07

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | | 数学与信息学院 | **专业班级** | 计算机科学与技术1班 | | | | | | **学号** | | 202425310114 | | | | | **姓名** | | 梁子祺 | | |
| **实**  **验**  **题**  **目** | 实现平衡二叉排序树的各种算法 | | | | | | | | | | | | | | | | | | | |
| **自**  **我**  **评**  **价** | **项 目** | | | | | **算法设计** | | | **独立完成情况** | | | | | **算法熟练程度** | | | | | | **测试通过** |
| 成功 | | 失败 | 独立 | | 帮助 | | | 掌握 | | 了解 | | 不懂 | |
| 插入新结点 | | | | | ✔️ | |  | ✔️ | |  | | | ✔️ | |  | |  | | ✔️ |
| 前序、中序、后序遍历二叉树（递归） | | | | | ✔️ | |  | ✔️ | |  | | | ✔️ | |  | |  | | ✔️ |
| 前序、中序、后序遍历的非递归算法 | | | | | ✔️ | |  | ✔️ | |  | | | ✔️ | |  | |  | | ✔️ |
| 层次遍历二叉树 | | | | | ✔️ | |  | ✔️ | |  | | | ✔️ | |  | |  | | ✔️ |
| 在二叉树中查找给定关键字 | | | | | ✔️ | |  | ✔️ | |  | | | ✔️ | |  | |  | | ✔️ |
| 交换各结点的左右子树 | | | | | ✔️ | |  | ✔️ | |  | | | ✔️ | |  | |  | | ✔️ |
| 求二叉树的深度 | | | | | ✔️ | |  | ✔️ | |  | | | ✔️ | |  | |  | | ✔️ |
| 叶子结点数 | | | | | ✔️ | |  | ✔️ | |  | | | ✔️ | |  | |  | | ✔️ |
| 删除某结点 | | | | ✔️ | |  | | ✔️ | |  | | ✔️ | |  | | |  | | ✔️ |
| * + A---------完成实验要求的全部功能并运行通过，算法有一定的新意，程序代码符合书写规范，实验报告叙述清晰完整，有详尽的分析和总结。   + B---------完成实验要求的全部功能，程序代码符合书写规范，实验报告叙述 清晰完整。 * C---------完成实验要求的大部分功能，实验报告良好。 * D---------未按时完成实验，或者抄袭。 | | | | | | | | | | | | | | | | | | | |
| **成绩** |  | | | | | | | | | | | | | | | | | | | |

教师签名：杨秋妹

**一、分析题目要求**

**(1)程序设计任务：**

本程序需要实现二叉排序树的插入，递归排序，非递归排序，查找，交换子树，求深度，统计叶子节点数，删除结点等功能。

**(2)函数定义：**

1.TreeNode\* createNode(int val)

参数：一个整数 val，表示新节点的数据值。

返回值类型：TreeNode\*，指向新创建的节点指针。

  功能：创建一个新节点，并将其左右子指针初始化为空。

2.TreeNode\* insert(TreeNode\* node, int val)

  参数：当前节点指针 node，待插入的数据 val。

  返回值类型：TreeNode\*，返回插入后树的根节点指针。

  功能：将新数据插入到二叉排序树中，保持其有序结构。

3.void preorderRecursive(TreeNode\* node)

  参数：当前节点指针 node。

  返回值类型：void。

  功能：递归方式对二叉树进行前序遍历（根 -> 左 -> 右）。

4.void inorderRecursive(TreeNode\* node)

  参数：当前节点指针 node。

  返回值类型：void。

  功能：递归方式进行中序遍历（左 -> 根 -> 右）。

5.void postorderRecursive(TreeNode\* node)

  参数：当前节点指针 node。

  返回值类型：void。

  功能：递归方式进行后序遍历（左 -> 右 -> 根）。

6.void preorderNonRecursive(TreeNode\* root)

  参数：根节点指针 root。

  返回值类型：void。

  功能：利用栈实现非递归的前序遍历。

7.void inorderNonRecursive(TreeNode\* root)

  参数：根节点指针 root。

  返回值类型：void。

  功能：使用栈实现非递归的中序遍历。

8.void postorderNonRecursive(TreeNode\* root)

  参数：根节点指针 root。

  返回值类型：void。

  功能：利用两个栈实现非递归的后序遍历。

9.void levelOrder(TreeNode\* root)

  参数：根节点指针 root。

  返回值类型：void。

  功能：使用队列进行层次遍历（按层输出每个节点）。

10.int search(TreeNode\* node, int key)

  参数：当前节点指针 node，关键字 key。

  返回值类型：int，找到返回 1，未找到返回 0。

  功能：在二叉排序树中查找给定的关键字。

11.void swapLeftRight(TreeNode\* node)

  参数：当前节点指针 node。

  返回值类型：void。

  功能：递归地交换树中所有节点的左右子树。

12.int getDepth(TreeNode\* node)

  参数：当前节点指针 node。

  返回值类型：int，表示树的最大深度。

  功能：计算并返回当前树的最大深度（高度）。

13.int getLeafCount(TreeNode\* node)

  参数：当前节点指针 node。

  返回值类型：int，叶子结点数量。

  功能：统计树中所有没有子节点的结点数。

14.TreeNode\* deleteNode(TreeNode\* node, int val)

  参数：当前节点指针 node，要删除的值 val。

  返回值类型：TreeNode\*，删除后的树根节点指针。

  功能：在二叉排序树中删除指定值的结点，并保持有序结构。

15.TreeNode\* findMin(TreeNode\* node)

  参数：当前节点指针 node。

  返回值类型：TreeNode\*，最小值结点指针。

  功能：查找并返回当前树或子树中最小值所在的结点。

16.void clearInputBuffer()

  参数：无。

  返回值类型：void。

  功能：清空输入缓冲区，防止非法输入干扰程序。

17.Stack\* createStack(int capacity)

  参数：整数 capacity，表示栈的容量。

  返回值类型：Stack\*，新建栈的指针。

  功能：初始化一个指定容量的栈，用于非递归遍历。

18.void push(Stack\* stack, TreeNode\* node)

  参数：栈指针 stack，树结点指针 node。

  返回值类型：void。

  功能：将结点压入栈顶。

19.TreeNode\* pop(Stack\* stack)

  参数：栈指针 stack。

  返回值类型：TreeNode\*，返回栈顶结点。

  功能：弹出栈顶结点。

20.bool isStackEmpty(Stack\* stack)

  参数：栈指针 stack。

  返回值类型：bool，返回栈是否为空。

  功能：判断栈是否为空。

21.void freeStack(Stack\* stack)

  参数：栈指针 stack。

  返回值类型：void。

  功能：释放栈所占用的内存。

22.Queue\* createQueue(int capacity)

  参数：整数 capacity，表示队列的容量。

  返回值类型：Queue\*，新建队列的指针。

  功能：创建一个容量固定的队列，用于层次遍历。

23.void enqueue(Queue\* queue, TreeNode\* node)

  参数：队列指针 queue，树结点指针 node。

  返回值类型：void。

  功能：将节点加入队列尾部。

24.TreeNode\* dequeue(Queue\* queue)

  参数：队列指针 queue。

  返回值类型：TreeNode\*，返回队头结点。

  功能：从队列头部取出一个结点。

25.bool isQueueEmpty(Queue\* queue)

  参数：队列指针 queue。

  返回值类型：bool。

  功能：判断队列是否为空。

26.void freeQueue(Queue\* queue)

  参数：队列指针 queue。

  返回值类型：void。

  功能：释放队列占用的内存空间。

27.void insertMultiple(TreeNode\*\* root)

参数：指向根节点指针的指针TreeNode\*\* root。

  返回值类型：void。

  功能：批量插入多个结点。用户输入插入数量和各结点值，依次插入到二叉排序树中。

**(3) 输入的形式和输入值的范围：**

程序在命令行形式运行，在开始菜单中选择对应的不同编号选择不同的操作。节点数设置为1000，建立的结点不能大于1000，在输入其他有逻辑错误时系统会自动忽略。输入数据时系统以有序形式插入树中。

**(4) 输出的形式：**

每完成一个操作会打印出对应的结果，以方便验证。

**(5) 程序所能达到的功能：**

本程序实现了基于二叉排序树的数据结构操作，实现了二叉排序树的插入，递归排序，非递归排序，查找，交换子树，求深度，统计叶子节点数和删除结点的功能，充分满足题目的要求。

**二、解题思路**

1．插入新结点

思路：使用递归方式插入数据，遵循 BST 左小右大的性质。

用到的数据结构：树（递归调用）

伪代码：

function insert(node, val):

if node == NULL:

return new Node(val)

if val < node.data:

node.left = insert(node.left, val)

else if val > node.data:

node.right = insert(node.right, val)

return node

2．递归遍历（前序、中序、后序）

思路：使用递归实现三种遍历方式。

用到的数据结构：树（递归）

伪代码：

function inorder(node):

if node != NULL:

inorder(node.left)

print node.data

inorder(node.right)

(中序遍历演示，后三行交换可实现前序后序遍历)

3. 非递归遍历（前序、中序、后序）

思路：使用栈模拟递归过程，避免系统调用栈。

用到的数据结构：栈（手动模拟调用过程）

伪代码：

function inorderNonRecursive(root):

stack = empty stack

current = root

while current != NULL or stack not empty:

while current != NULL:

stack.push(current)

current = current.left

current = stack.pop()

print current.data

current = current.right

(中序遍历演示，后三行交换可实现前序后序遍历)

4. 层次遍历

思路：利用队列先进先出的特点，逐层输出节点。

用到的数据结构：队列

伪代码：

function levelOrder(root):

queue = empty queue

queue.enqueue(root)

while queue not empty:

node = queue.dequeue()

print node.data

if node.left:

queue.enqueue(node.left)

if node.right:

queue.enqueue(node.right)

5. 查找关键字

思路：递归查找关键字，比较大小决定搜索方向。

用到的数据结构：树（递归）

伪代码：

function search(node, key):

if node == NULL:

return 0

if key == node.data:

return 1

else if key < node.data:

return search(node.left, key)

else:

return search(node.right, key)

6. 交换左右子树

思路：递归交换所有结点的左右子指针。

用到的数据结构：树（递归）

伪代码：

function swap(node):

if node != NULL:

swap(node.left)

swap(node.right)

temp = node.left

node.left = node.right

node.right = temp

7. 计算树的深度

思路：递归求左右子树最大深度，取较大值 +1。

用到的数据结构：树（递归）

伪代码：

function getDepth(node):

if node == NULL:

return 0

left = getDepth(node.left)

right = getDepth(node.right)

return max(left, right) + 1

8. 统计叶子结点数量

思路：若左右子树皆为空，则是叶子，递归累加。

用到的数据结构：树（递归）

伪代码：

function countLeaf(node):

if node == NULL:

return 0

if node.left == NULL and node.right == NULL:

return 1

return countLeaf(node.left) + countLeaf(node.right)

9. 删除结点

思路： (1)若待删节点无子树，直接删除

(2)若只有一边子树，用其子树代替

(3)若有两个子树，用右子树中的最小值替换当前节点，然后递归删除替代节点。

用到的数据结构：树（递归）

伪代码：

function deleteNode(node, val):

if node == NULL:

return NULL

if val < node.data:

node.left = deleteNode(node.left, val)

else if val > node.data:

node.right = deleteNode(node.right, val)

else:

if node.left == NULL:

return node.right

else if node.right == NULL:

return node.left

temp = findMin(node.right)

node.data = temp.data

node.right = deleteNode(node.right, temp.data)

return node

**三、调试分析**

**(1)** **在实验开发与调试过程中，主要遇到以下问题：**

输入非法数据后程序异常，在检查代码逻辑后发现是输入非整数或字符时，程序进入了死循环或读取错误。通过增加 clearInputBuffer() 函数，在输入错误时清空缓冲区并给出提示，避免非法数据污染后续输入。

非递归遍历栈使用不当，在检查代码逻辑后发现是初期在中序非递归遍历中，栈为空或指针未及时更新，导致死循环或空指针异常。修改代码通过严格检查栈的使用逻辑和节点指针状态，最终实现正确遍历。

本程序的设计遵循着模块化的原则，各操作单独调用函数，便于系统维护。

**(2)** **测试样例**

测试组1：10 5 15 3 7 12 18

前序遍历：前序遍历(递归): 10 5 3 7 15 12 18

中序遍历：中序遍历(递归): 3 5 7 10 12 15 18

后序遍历：后序遍历(递归): 3 7 5 12 18 15 10

非递归输出同上。

层次遍历：层次遍历: 10 5 15 3 7 12 18

找关键字（如查找7）： 请输入要查找的关键字: 7

查找成功: 找到关键字 7

求树的深度：二叉树的深度: 3

求叶子结点数：叶子结点数: 4

删除结点（如删除15）后的中序遍历：3 5 7 10 12 18

测试组2：8 4 12 2 6 10 14 1 3 5 7 9 11 13 15

前序遍历：前序遍历(递归): 8 4 2 1 3 6 5 7 12 10 9 11 14 13 15

中序遍历：中序遍历(递归): 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

后序遍历：后序遍历(递归): 1 3 2 5 7 6 4 9 11 10 13 15 14 12 8

非递归输出同上。

层次遍历：层次遍历: 8 4 12 2 6 10 14 1 3 5 7 9 11 13 15

找关键字（如查找7）： 请输入要查找的关键字: 7

查找成功: 找到关键字 7

求树的深度：二叉树的深度: 4

求叶子结点数：叶子结点数: 8

删除结点（如删除15）后的中序遍历：1 2 3 4 5 6 7 8 9 10 11 12 13 14

测试组3：20 10 30 5 15 25 35 2 7 12 17 22 27 32 37

前序遍历：前序遍历(递归): 20 10 5 2 7 15 12 17 30 25 22 27 35 32 37

中序遍历：中序遍历(递归): 2 5 7 10 12 15 17 20 22 25 27 30 32 35 37

后序遍历：后序遍历(递归): 2 7 5 12 17 15 10 22 27 25 32 37 35 30 20

非递归输出同上。

层次遍历：层次遍历: 20 10 30 5 15 25 35 2 7 12 17 22 27 32 37

找关键字（如查找9）： 请输入要查找的关键字: 9

查找失败: 未找到关键字9

求树的深度：二叉树的深度: 4

求叶子结点数：叶子结点数: 8

删除结点（如删除15）后的中序遍历：2 5 7 10 12 17 20 22 25 27 30 32 35 37

**(3)** **算法效率分析**

可以用c++中的类来对代码进行优化，是代码更简便。也可以可进一步引入 AVL树 或 红黑树 等平衡树结构，实现插入、删除后自动保持平衡，提升最坏情况下效率。

**(4)** **经验体会**

通过本次综合性实验，我更加深入地掌握了二叉树结构的基本原理，特别是递归与非递归算法的区别与实现方式。同时也熟练运用了堆栈和队列两种数据结构在遍历过程中的应用。更熟练地掌握了课程内容。充分达到了掌握递归算法，掌握堆栈算法，掌握队列算法，掌握树型查找和提高对树型结构的综合应用能力的实验要求。

**四、附录**

**代码文件：**

**程序文件：**

**CMake配置文件：**