

**课程名称：数据结构与算法**

**第一次上机实验报告**

**姓 名： 刘瑞欣**

**学 号： U202016818**

**班 级： 财务管理2001班**

**任课教师： 刘干**

**目录**

[0. 二叉树 3](#_Toc132034905)

[0.1 递归方法建立二叉树 3](#_Toc132034906)

[0.2 非递归方法前序遍历二叉树 4](#_Toc132034907)

[0.3 递归方法前序/中序/后序遍历二叉树 5](#_Toc132034908)

[1. 将二叉树对称交换，即求二叉树镜像 6](#_Toc132034909)

[1.1 算法一：递归算法 6](#_Toc132034910)

[1.2 算法二：非递归算法——利用队列 6](#_Toc132034911)

[2. 求二叉树的高度，即深度 7](#_Toc132034912)

[2.1 算法一：递归算法 7](#_Toc132034913)

[2.2 算法二：非递归算法 8](#_Toc132034914)

[3. 已知前序序列和中序序列，构造二叉树 9](#_Toc132034915)

[4. 分层遍历二叉树，即按层次从上往下，从左往右顺序访问 11](#_Toc132034916)

[5. 求二叉树的宽度，即求最大结点数的层所具有的结点数 11](#_Toc132034917)

[6. 二叉树中的结点个数 13](#_Toc132034918)

[7. 非递归方法中序、后续遍历二叉树 13](#_Toc132034919)

[8. 求二叉树中第K层的结点个数 15](#_Toc132034920)

[8.1 算法一：层序遍历法 15](#_Toc132034921)

[8.2 算法二：递归算法 16](#_Toc132034922)

[9. 求二叉树中叶子结点的个数 17](#_Toc132034923)

[9.1 算法一：非递归算法 17](#_Toc132034924)

[9.2 算法二：递归算法 18](#_Toc132034925)

[10. 判断两个二叉树结构是否相同 18](#_Toc132034926)

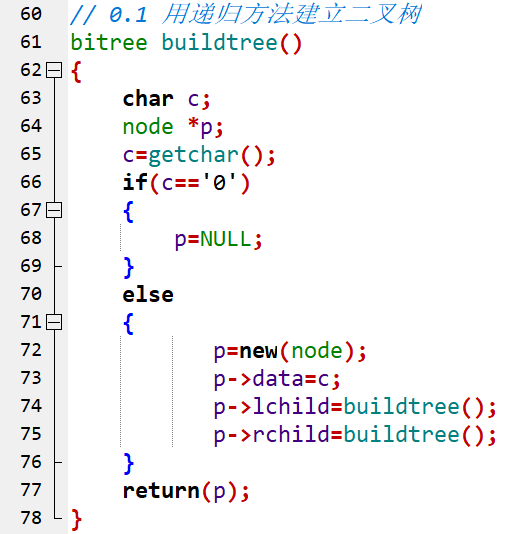
[11. 求二叉树中两个结点的最低公共祖先结点 19](#_Toc132034927)

# 0. 二叉树

## 0.1 递归方法建立二叉树

算法思想：输入数据，非空就创建新的节点，将数据赋给新的节点，然后从左边开始递归，如果是空，就再从右边递归，右边空就返回上一层。

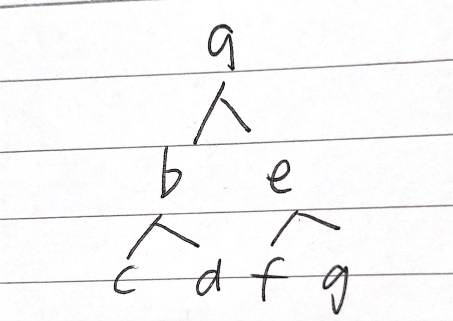
子函数代码：



时间复杂度：O(n)

空间复杂度：O(n)

运行结果：在主函数中输入abc00d00ef00g000，得到的结果为下图所示，之后都都将使用这个二叉树作为实验代码测试对象。



## 0.2 非递归方法前序遍历二叉树

算法思想：利用栈来存放，在访问完t->data后，将t入栈，遍历左子树；遍历完左子树返回时，栈顶元素应为t，出栈，再遍历t的右子树。

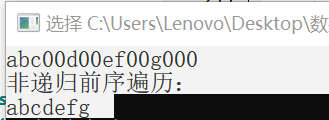
子函数代码：



时间复杂度：O(n)

空间复杂度：O(n)

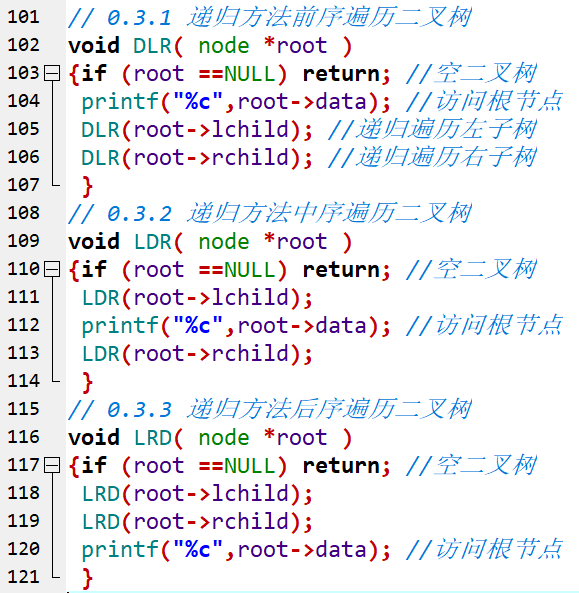
运行结果：



## 0.3 递归方法前序/中序/后序遍历二叉树

算法思想：利用递归算法，按照前中后序的顺序依次对左子树、右子树与根节点进行遍历。

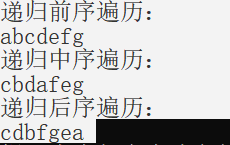
子函数代码：



时间复杂度：O(n)

空间复杂度：O(n)

运行结果：



# 1. 将二叉树对称交换，即求二叉树镜像

## 1.1 算法一：递归算法

算法思想：递归算法，先把左子树中的节点放到rchild中，然后再把右边结点放到lchild中，这个过程中要先把左节点暂存到一个地方。

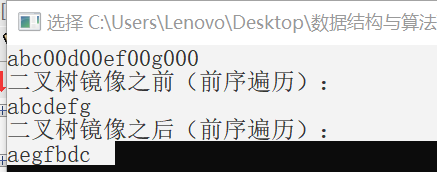
子函数代码：



时间复杂度：O(n)

空间复杂度：O(n)

运行结果：



## 1.2 算法二：非递归算法——利用队列

算法思想：利用队列先进先出的特性，层次遍历每一层，然后再队列里面做互换，然后再输出。

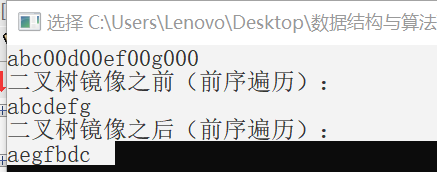
子函数代码：



时间复杂度：O(n)

空间复杂度：O(1)

运行结果：

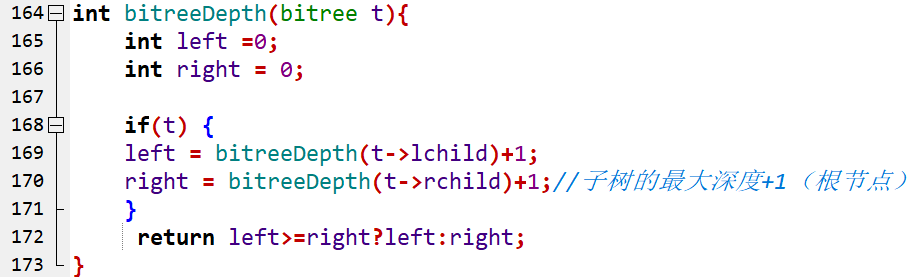


# 2. 求二叉树的高度，即深度

## 2.1 算法一：递归算法

算法思想：递归算法，为了求得树的深度，可以先求左右子树的深度，取二者较大者加1即是树的深度。

子函数代码：



时间复杂度：O(n)

空间复杂度：O(n)

运行结果：



## 2.2 算法二：非递归算法

算法思想：利用层序遍历，当当前节点数小于队列的size的时候，说明这一层没有结束，继续推进；否则说明已经到了这一层的末尾。

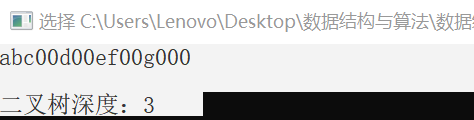
子函数代码：



时间复杂度：O(n²)

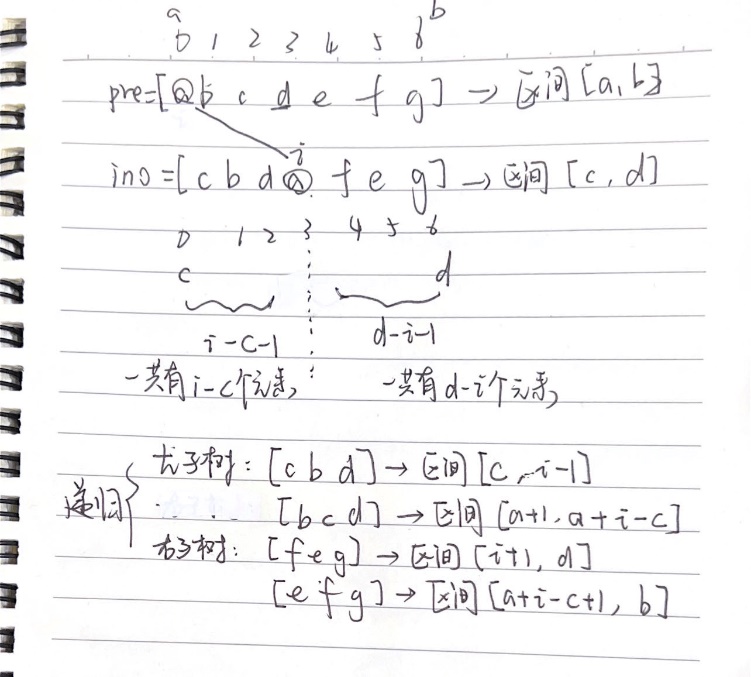
空间复杂度：O(1)

运行结果：

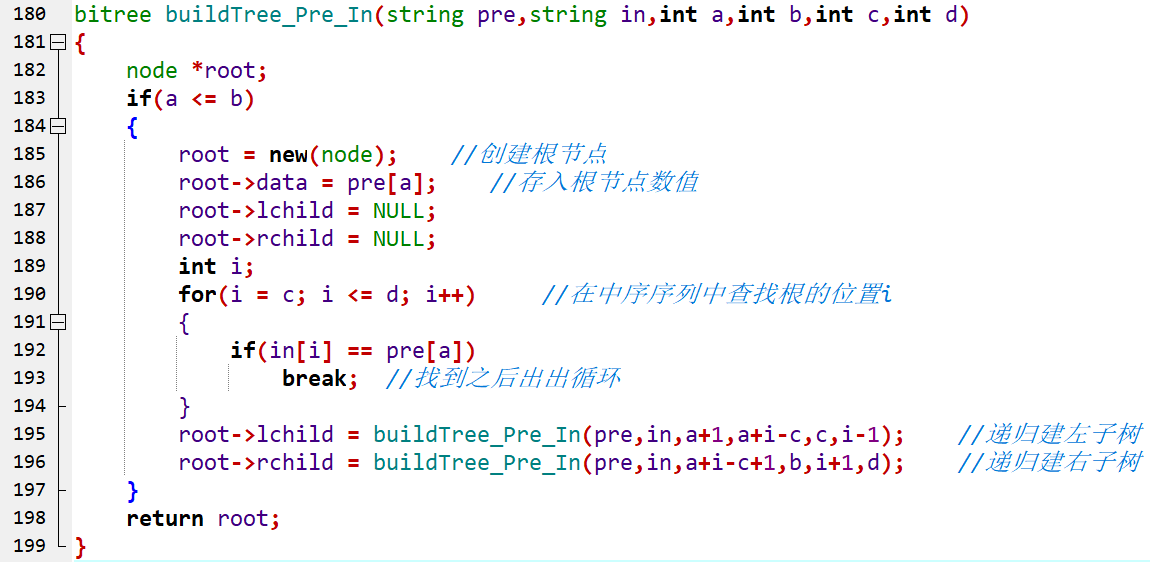


# 3. 已知前序序列和中序序列，构造二叉树

算法思想：可以知道先序序列的第一个是根节点，在中序序列中找到根节点，根节点左边的都是左子树，右边的就是右子树的成员，再进行函数递归，重新赋值后再对子树进行一样的操作，最后构造出一颗二叉树。



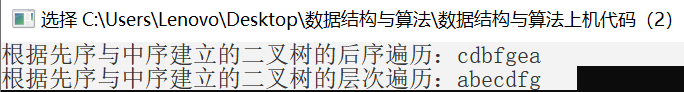
子函数代码：



时间复杂度：O(n)

空间复杂度：O(n)

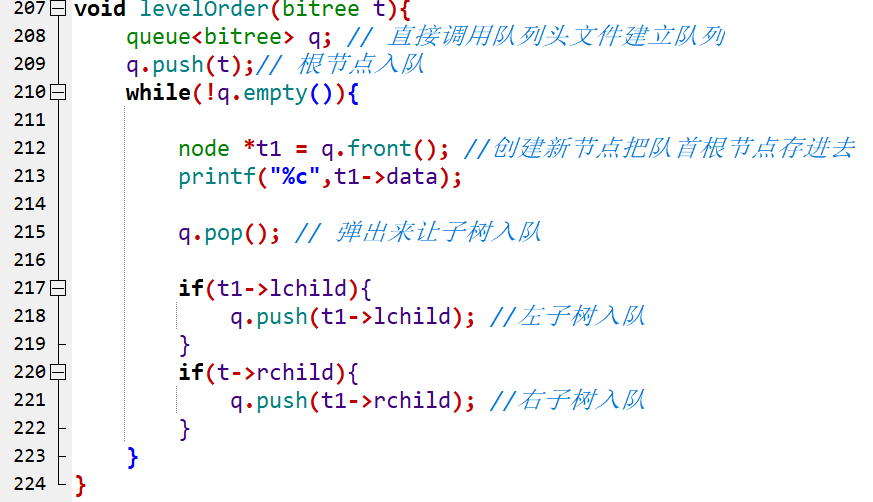
运行结果：



# 4. 分层遍历二叉树，即按层次从上往下，从左往右顺序访问

算法思想：利用队列先进先出的性质，先把根节点入队，然后让根节点的左子树和右子树同时入队，然后依次出队。

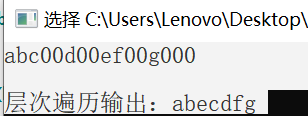
子函数代码：



时间复杂度：O(n)

空间复杂度：O(n)

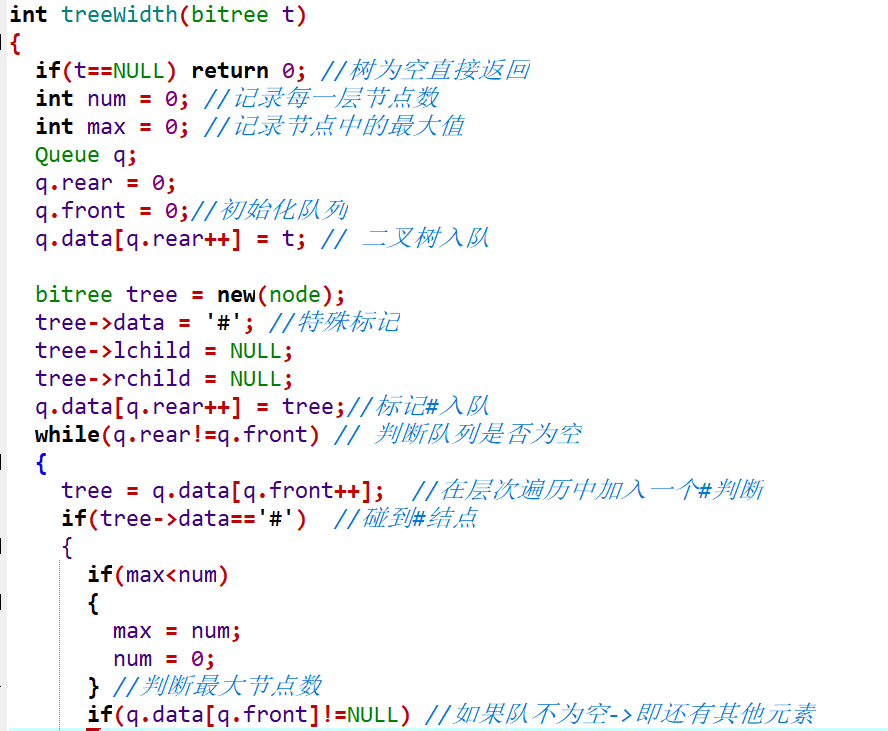
运行结果：

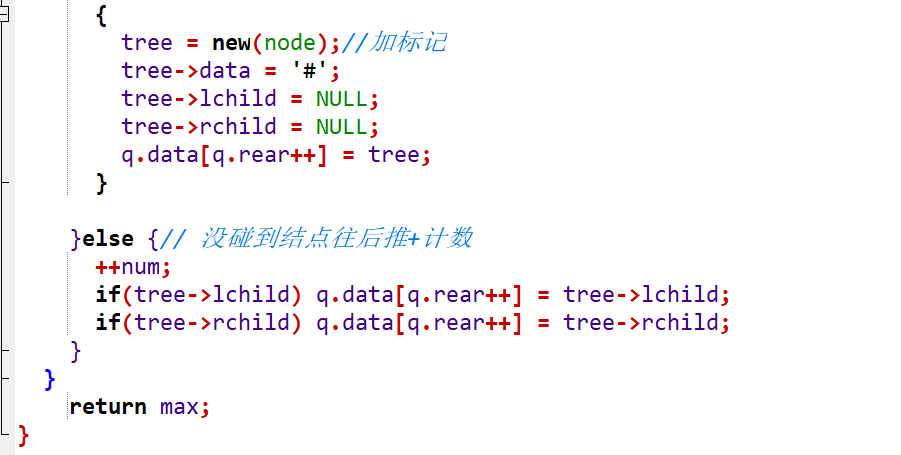


# 5. 求二叉树的宽度，即求最大结点数的层所具有的结点数

算法思想：层次遍历二叉树，设定一个特别的标记#来标记每一层的最右边，代表一层遍历的结束，但是用标记会导致陷入无限循环，因为最后剩下的那个一定是#，因此每当遍历到#时，还要看队列中是否还有其他元素。

子函数代码：

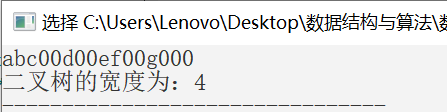




时间复杂度：O(n)

空间复杂度：O(1)

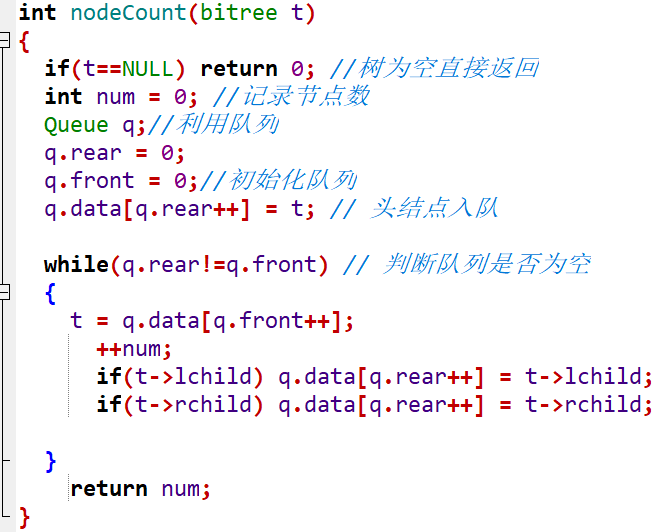
运行结果：



# 6. 二叉树中的结点个数

算法思想：层次遍历二叉树，设置一个计数器即可。

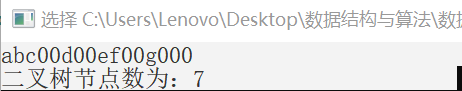
子函数代码：



时间复杂度：O(n)

空间复杂度：O(1)

运行结果：



# 7. 非递归方法中序、后续遍历二叉树

算法思想：利用辅助栈来实现。中序遍历与前序相似，主要在于后序遍历中如何区分上一节点的位置，后序遍历的顺序是左->右->根，若是根节点位于左子树，则需跳过根节点，先进入右子树，右子树访问完了再回头访问根节点。

子函数代码：

非递归中序遍历代码：



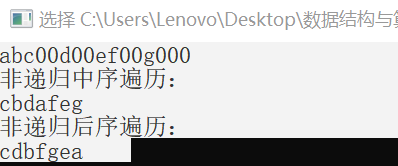
非递归后序遍历代码：



时间复杂度（中序/后序）：O(n)

空间复杂度（中序/后序）：O(n)

运行结果：

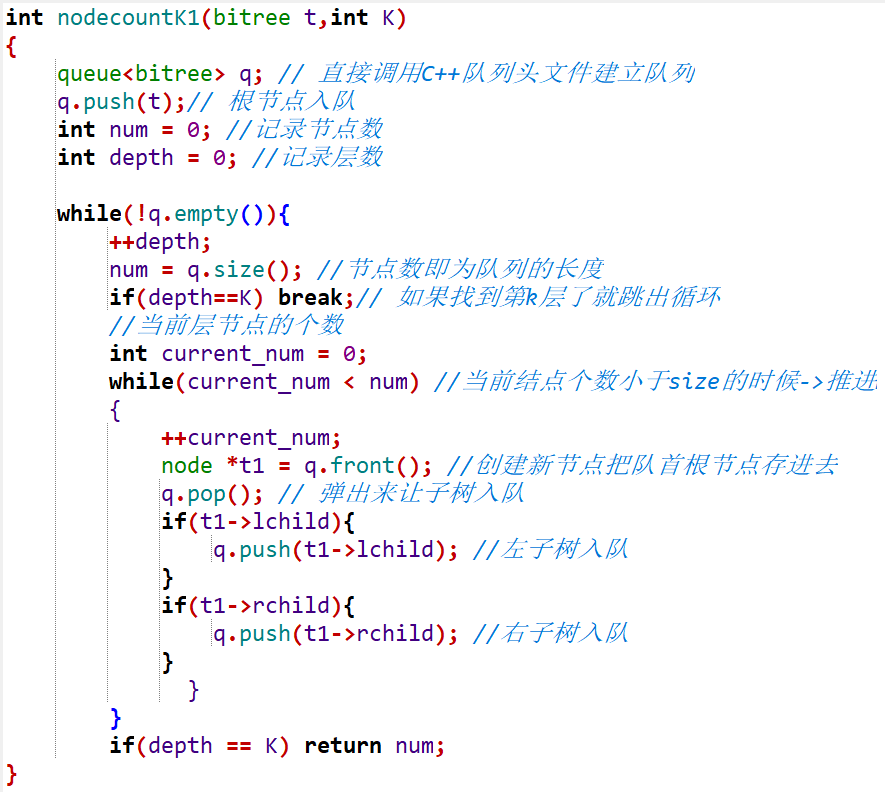


# 8. 求二叉树中第K层的结点个数

## 8.1 算法一：层序遍历法

算法思想：采用层序遍历，判断队列长度，如果当当前节点数小于队列size则继续推进，层数depth+1，当计数器depth等于K时，跳出循环，并返回该节点数。

子函数代码：



时间复杂度：O(n²)

空间复杂度：O(1)

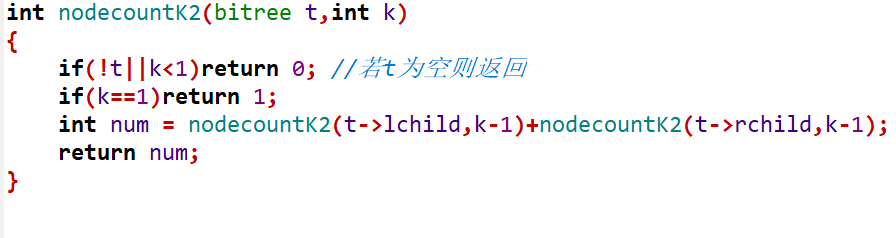
运行结果：



## 8.2 算法二：递归算法

算法思想：直接运用递归算法计算第k层的结点个数

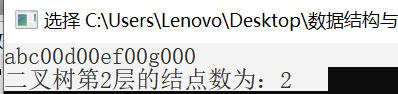
子函数代码：



时间复杂度：O(n)

空间复杂度：O(n)

运行结果：

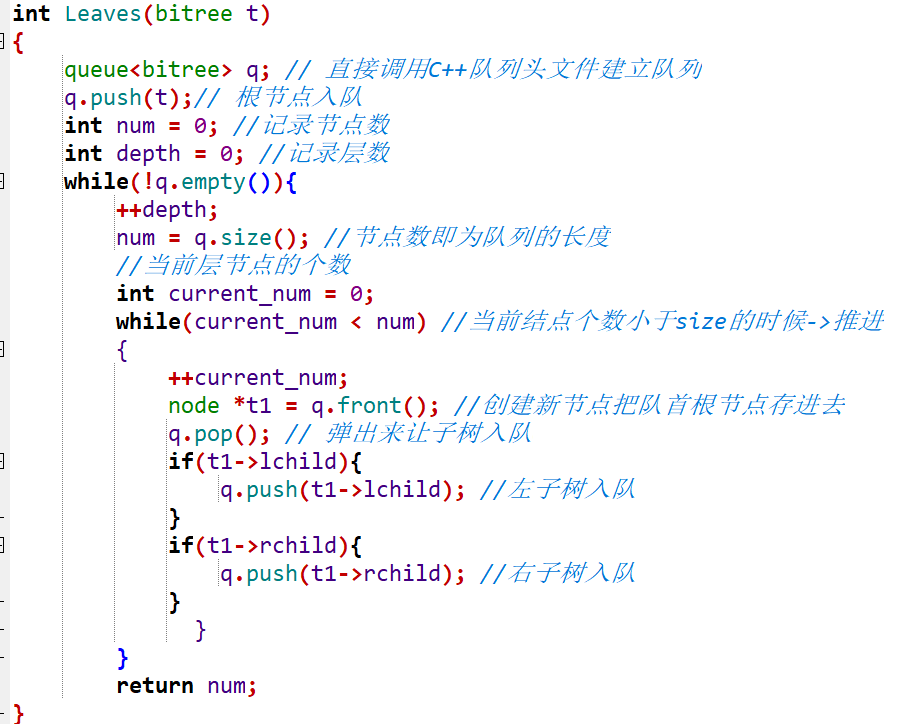


# 9. 求二叉树中叶子结点的个数

## 9.1 算法一：非递归算法

算法思想：利用层序遍历，，判断队列长度，如果当当前节点数小于这一层的队列size则继续推进，层数depth+1，直到最后一层，并返回该节点数。

子函数代码：



时间复杂度：O(n²)  
空间复杂度：O(1)

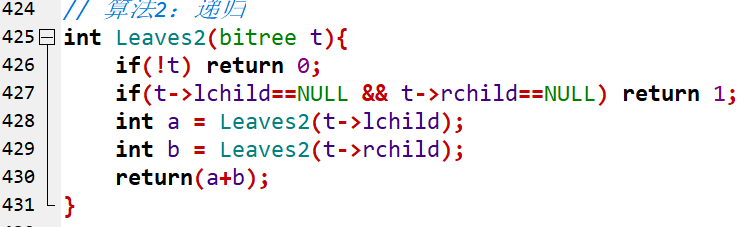
运行结果：



## 9.2 算法二：递归算法

算法思想：通过利用递归，分别求出左子树的叶子结点个数以及右子树的叶子结点个数，并进行加总。

子函数代码：



时间复杂度：O(n)

空间复杂度：O(n)

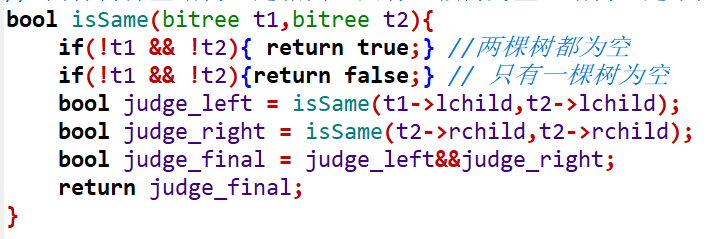
运行结果：



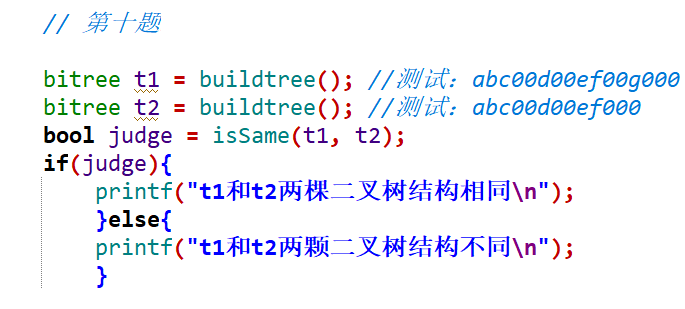
# 10. 判断两个二叉树结构是否相同

算法思想：如果一棵二叉树可以通过若干次左右孩子互换就变成另一个二叉树，那么我们称这两棵树是同构的，两棵树都为空的话，结构一定相同。只有一颗树为空，结构一定不同。递归，把全部的子树都比较一遍。

子函数代码：



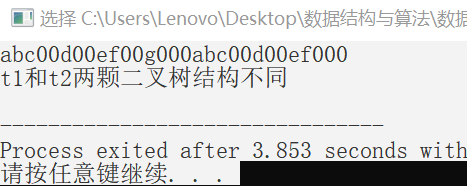
主函数部分输入：



时间复杂度：O(n)

空间复杂度：O(n)

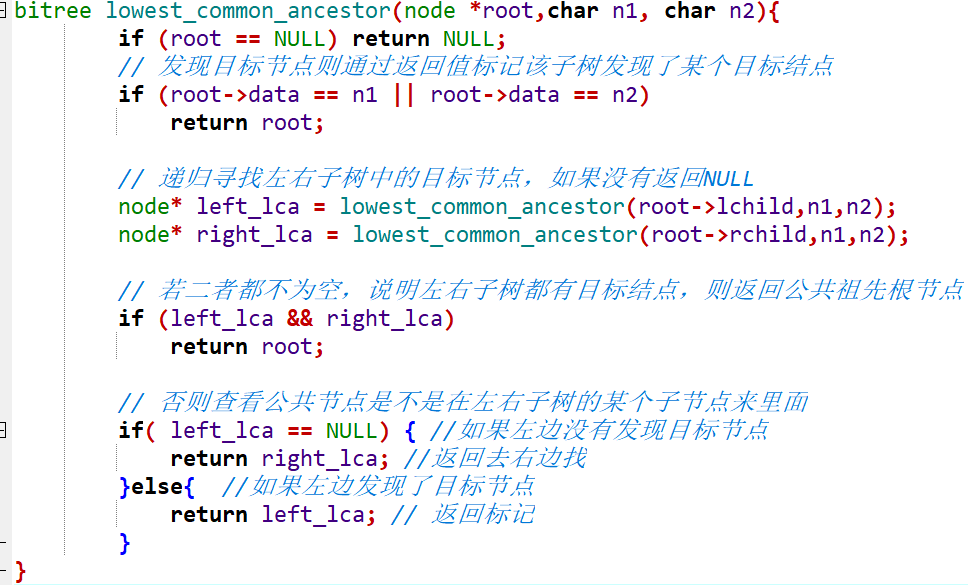
运行结果：



# 11. 求二叉树中两个结点的最低公共祖先结点

算法思想：利用递归思想，输入的两个结点为递归查找的目标节点，若根的左右子树中找到了目标节点，返回值标记这个子树发现了某个结点，否则继续向左右子树递归查找。如果两个子树均存在目标节点，那么说明最低公共祖先必然是根节点；如果只有一个子树存在目标节点，说明最低公共祖先节点存在于子树中。

子函数代码：



时间复杂度：O(n)

空间复杂度：O(n)

运算结果：

主函数输入为

