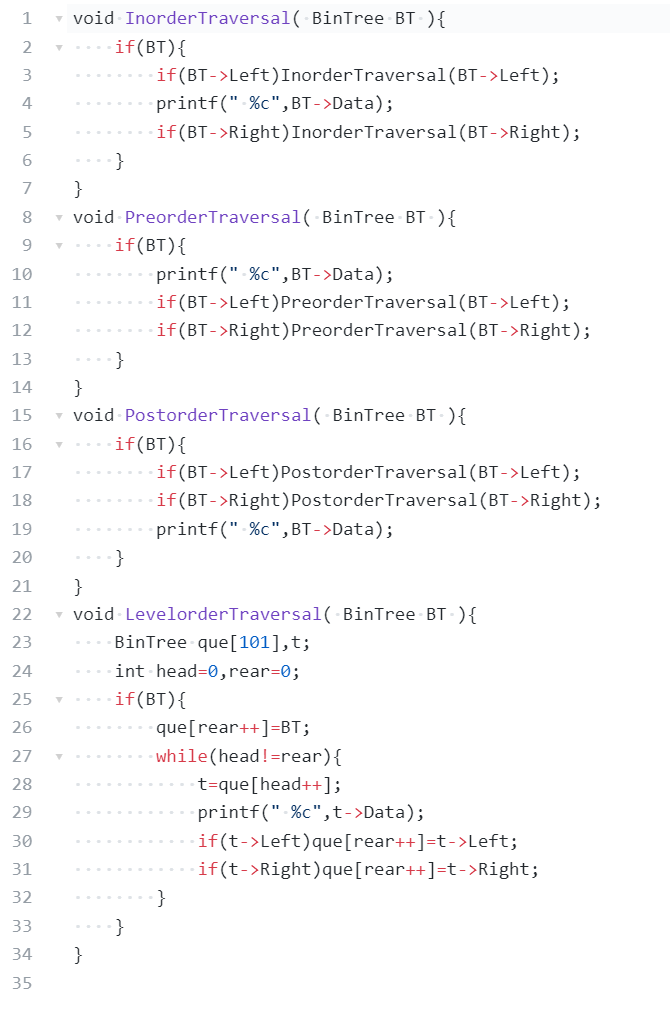
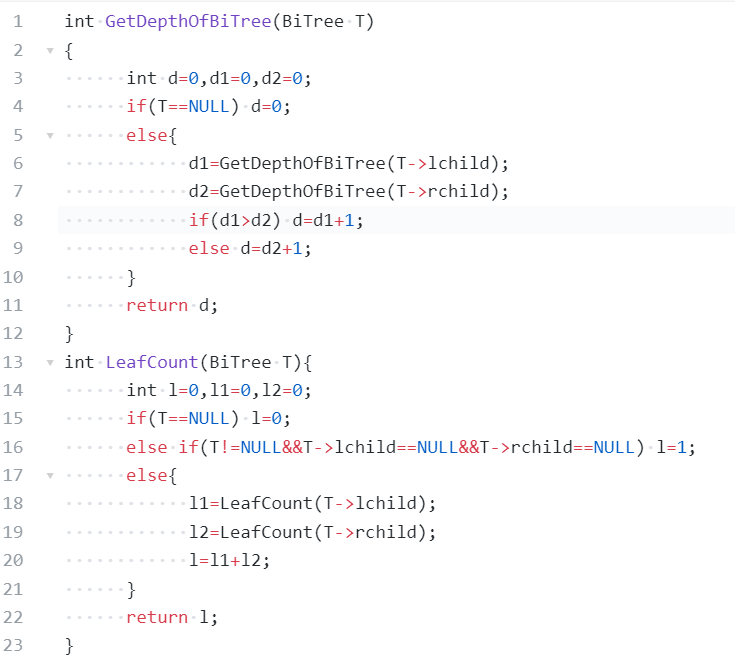
班级 计算机203 学号 202007020625 姓名 於俊涛

**实验题目：**

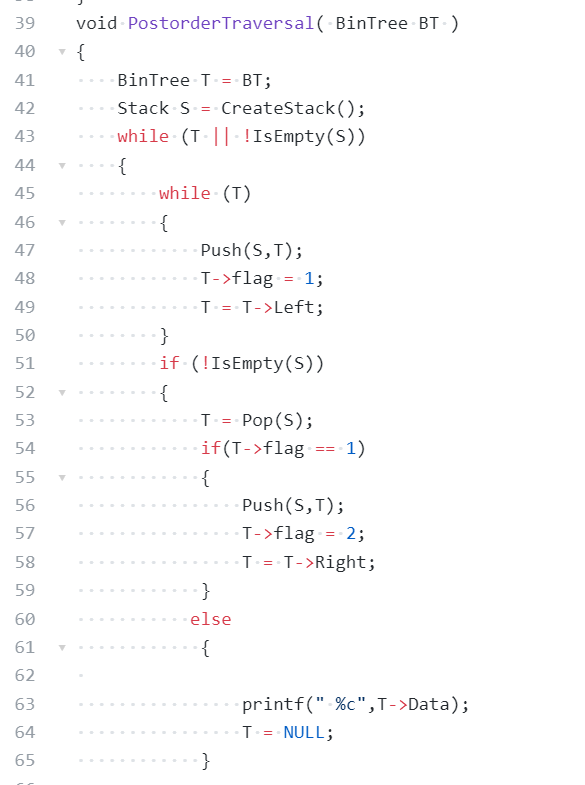
**一、概述**

算法实现技术：

二叉树求深度与叶子数↓ 递归遍历↓



非递归遍历↓



实验完成情况：设计出的代码顺利完成了任务要求；

结论：求二叉树的深度与叶子结点数用链式存储结构更方便；遍历二叉树采用递归方式设计出的代码更简洁；非递归遍历二叉树借助了堆栈的定义；

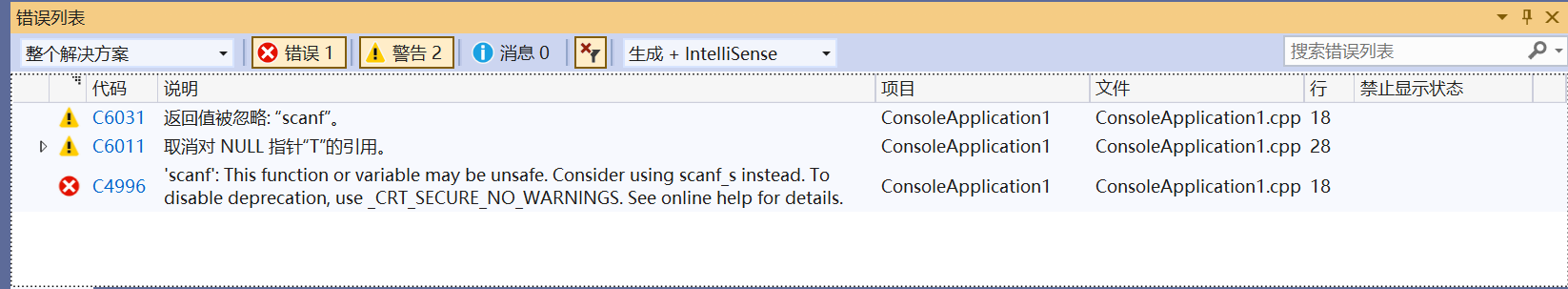
亮点：采用递归遍历方式输出根结点、左，右子树时套用了函数本身，加快了遍历的速度；

不足：非递归后续遍历时多访问了一次右节点增加了一个for循坏，耗时增多；

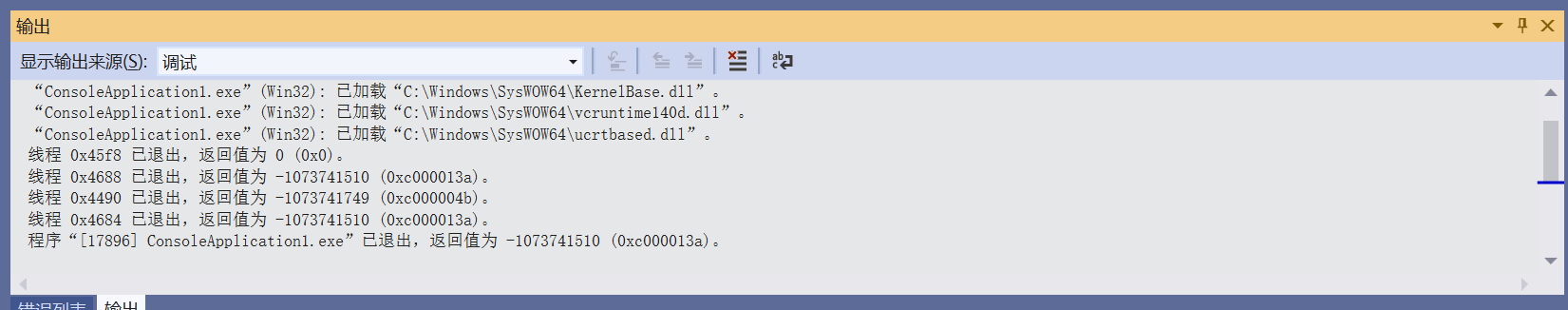
**二、实验过程**

1.调试分析

Scanf函数在vs不兼容；



将scanf函数用scanf\_s替代；

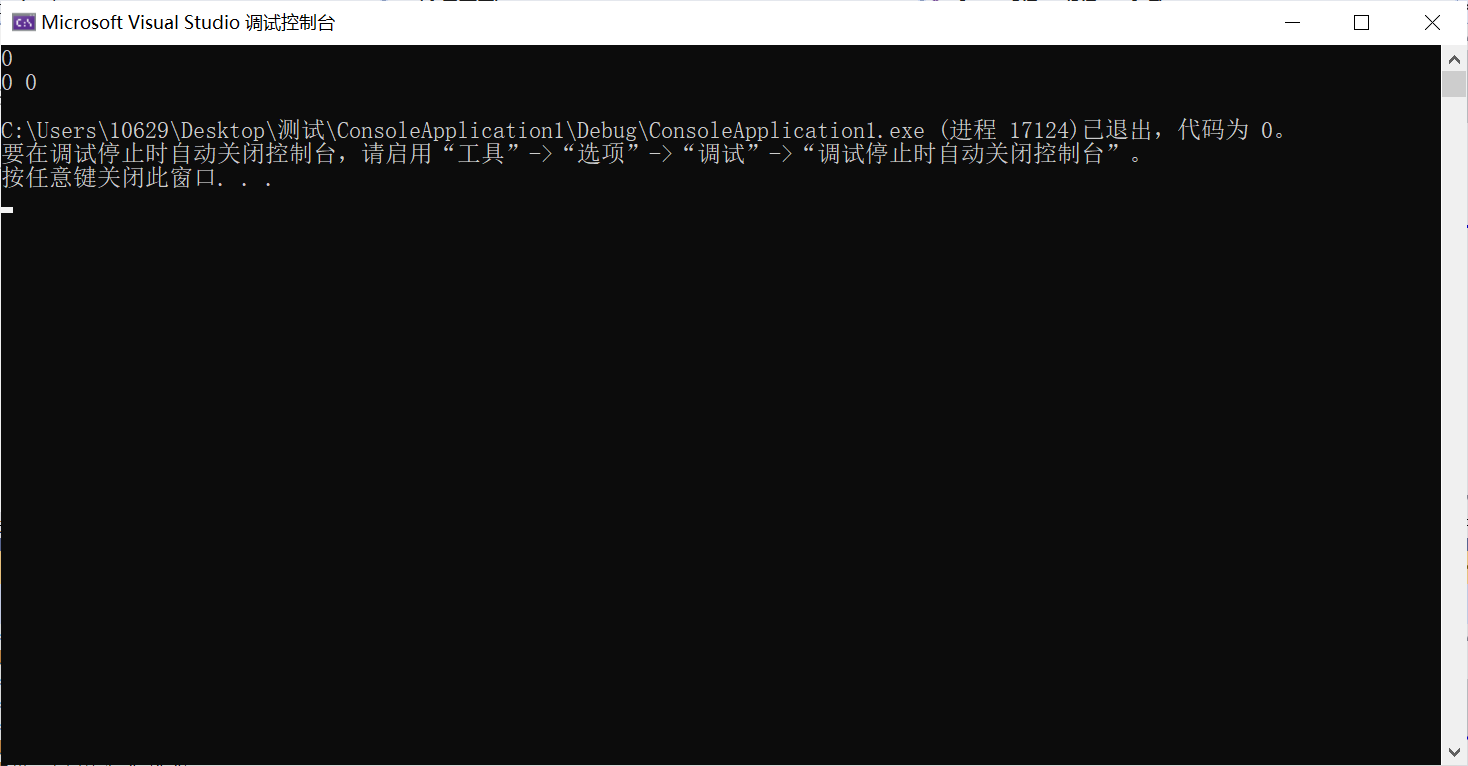


2.测试过程

实验一：

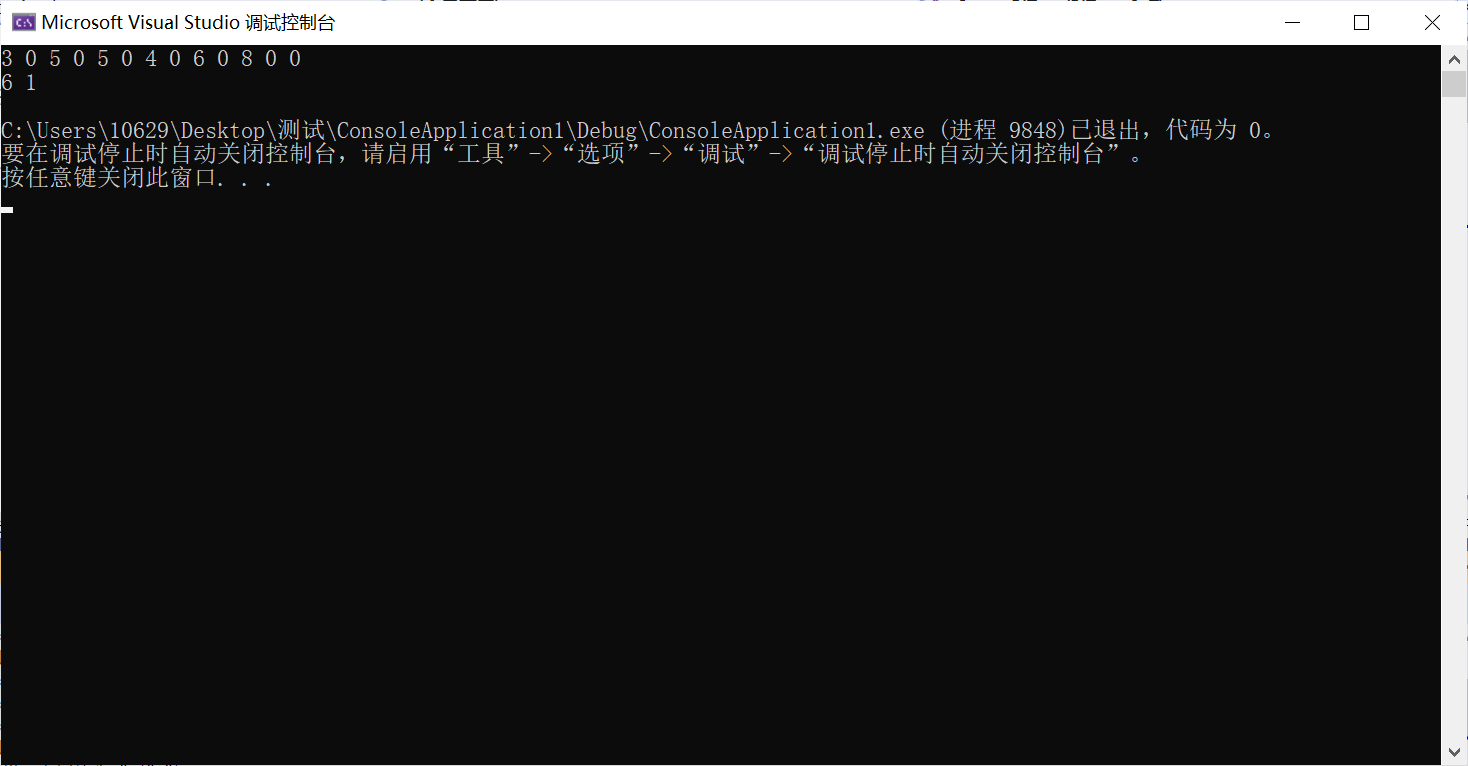
测试一：0；

0 0；



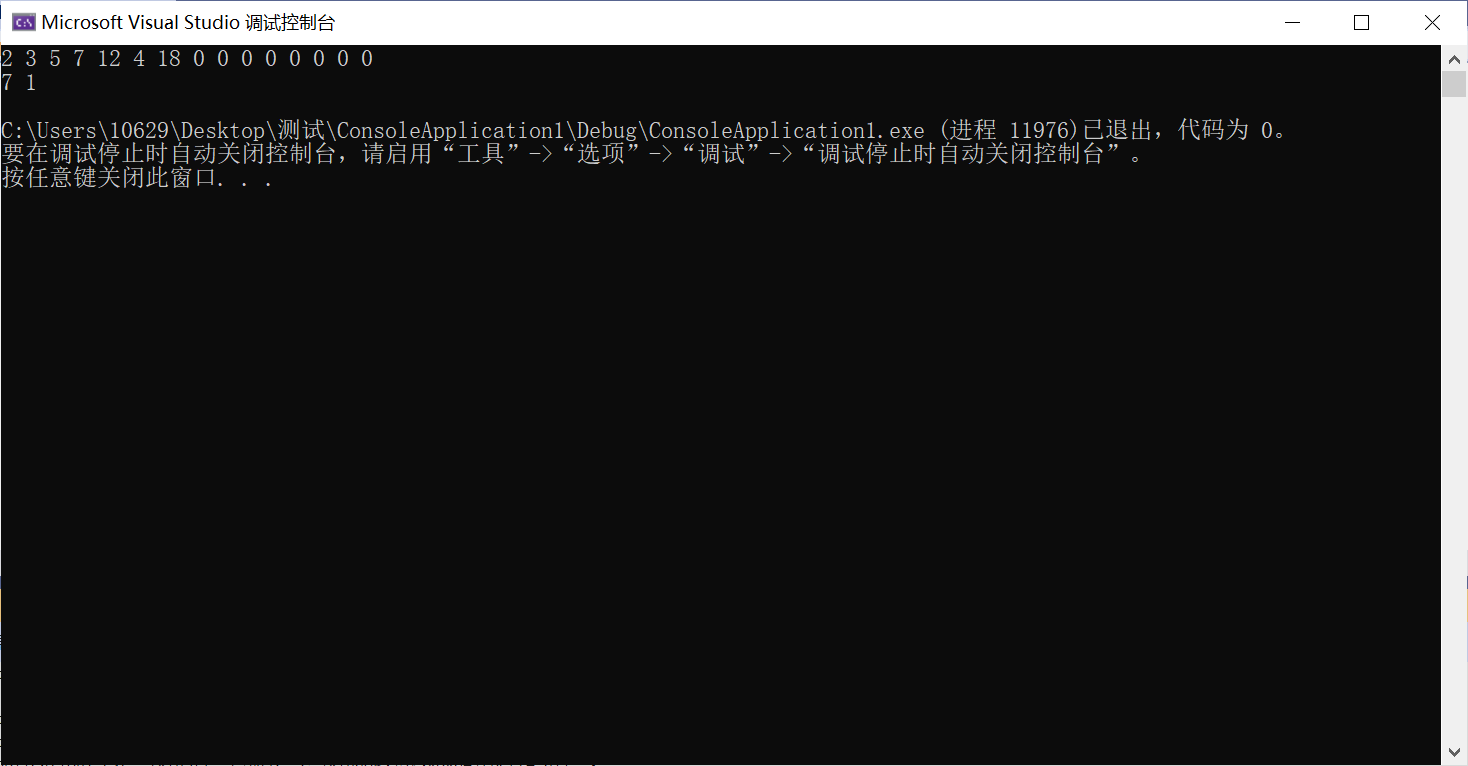
测试二：3 0 5 0 5 0 4 0 6 0 8 0 0；

6 1；



测试三：2 3 5 7 12 4 18 0 0 0 0 0 0 0 0；

7 1；



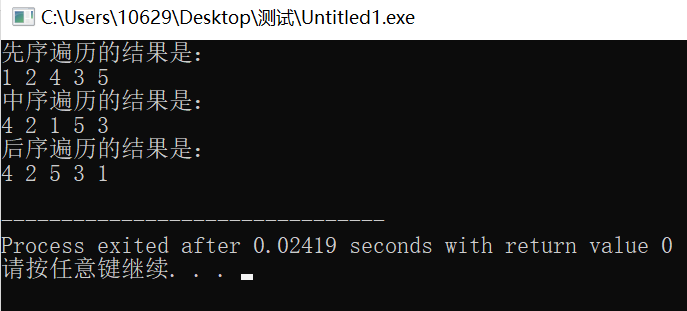
实验二：

测试一：

1

2 3

4 5

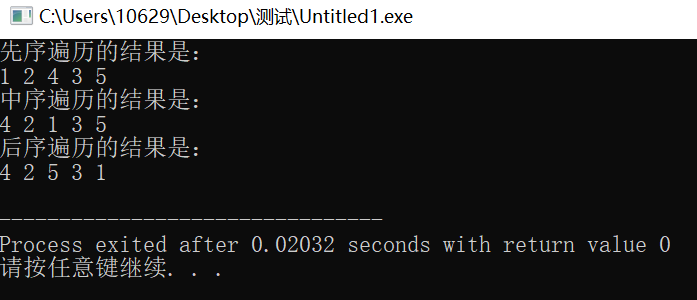


测试二：

1

2 3

4 5

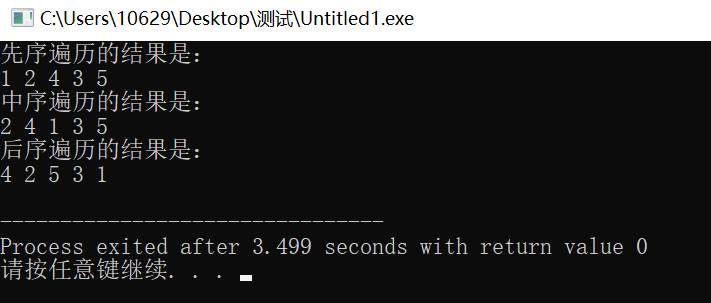


测试三：

1

2 3

4 5



**三、评价分析**

1.实验结果分析

实验一

测试数据一：0；

测试结果一：0 0；

分析：空二叉树，空树没有结点。叶子，深度自然也就为零。因此输出0 0

测试数据二：2 3 5 7 12 4 18 0 0 0 0 0 0 0 0

测试结果二：7 1

分析：该二叉树是左单支树，其结点有七个，2 3 5 7 12 4 18，叶子结点是18，在最底层。所以输出7 1

测试数据三：3 0 5 0 5 0 4 0 6 0 8 0 0

测试结果三：6 1

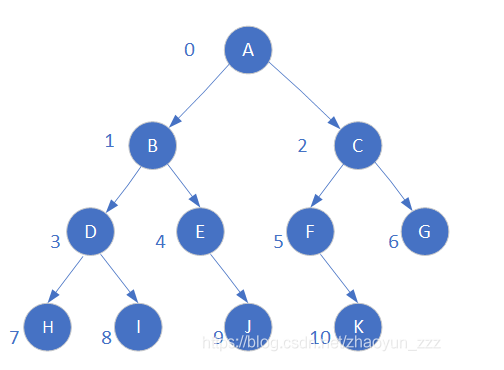
分析：该二叉树是右单支树，其结点有6个，3 5 5 4 6 8，叶子结点是8，在最底层所以输出6 1；

实验二：

仅改变叶子结点的位置，只改变一种遍历算法的结果而对另两种遍历算法没有影响；

2.算法性能评价

**四、总结与体会**



先序遍历

访问根结点

访问左子树的左子树，再访问左子树中的右子树

访问右子树的左子树，再访问右子树中的右子树

任意子树输出顺序为：父结点——左子结点——右子结点

如上图先序遍历顺序为：A-B-D-H-I-E-J-C-F-K-G

中序遍历

先访问左子树中的左子树，再访问左子树中的右子树

访问根结点。

后访问右子树中的左子树，再访问右子树中的右子树

任意子树输出顺序为：左子结点——父结点——右子结点

如上图中序遍历顺序为：H-D-I-B-E-J-A-F-K-C-G

后序遍历

先访问左子树中的左子树，再访问左子树中的右子树

再访问右子树中的左子树，再访问右子树中的右子树

访问根结点

任意子树输出顺序为：左子结点——右子结点——父结点

如上图后序遍历顺序为：H-I-D-J-E-B-K-F-G-C-A

层次遍历

访问根结点，即第1层，设为 i 层

访问i+1层的结点，从左到右顺序访问

层次遍历输出顺序为：根结点—— i 层结点从左到右—— i +1层结点从左到右

如上图层次遍历顺序为：A-B-C-D-E-F-G-H-I-J-K