

《数据结构与算法》实验报告

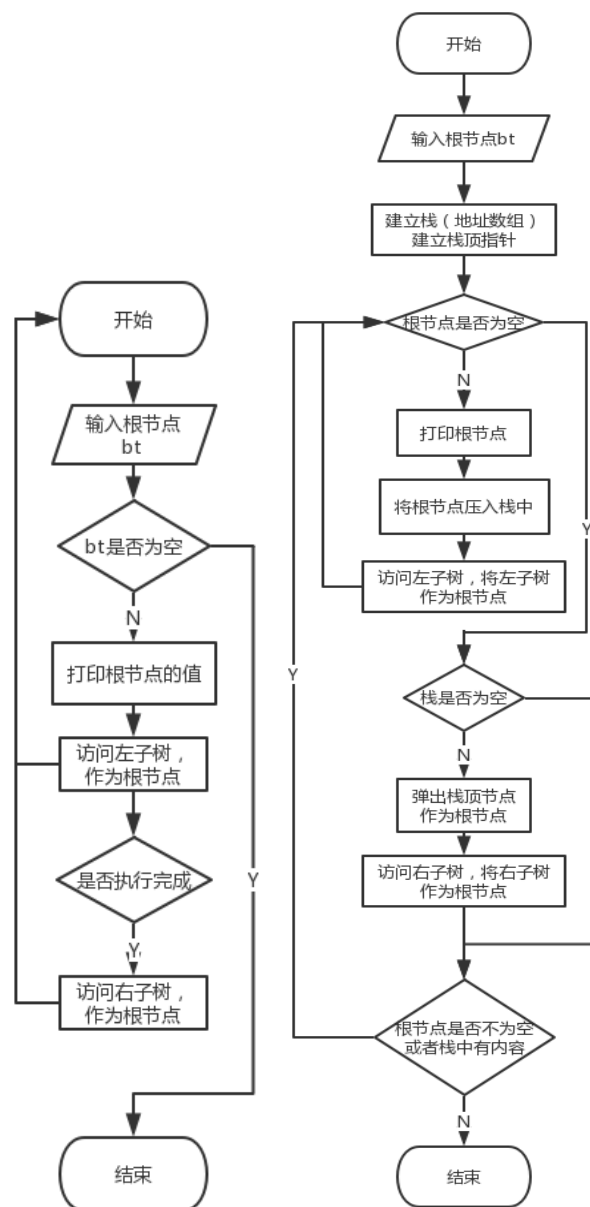
学生姓名		院（系）	计算机科学与技术
学 号		专 业	
实验时间		实验地点	
实验项目	实验 2/4：树型结构的建立与遍历		
<p>实验目的：将课程的基本原理、技术和方法与实际应用相结合，训练和提高学生组织、存储和处理信息的能力，以及复杂问题的数据结构设计能力和程序设计能力，培养软件设计与开发所需要的实践能力。</p> <p>实验要求：灵活运用基本的数据结构和算法知识，对实际问题进行分析和抽象；结合程序设计的一般过程和方法为实际问题设计数据结构和有效算法；用高级语言对数据结构和算法进行编程实现、调试，测试其正确性和有效性。</p>			
<p>实验内容：</p> <p>树型结构的遍历是树型结构算法的基础，本实验要求编写程序演示二叉树的存储结构的建立方法和遍历过程。</p> <ol style="list-style-type: none"> 1) 编写建立二叉树的二叉链表存储结构（左右链表示）的程序，并以适当的形式显示和保存二叉树； 2) 采用二叉树的二叉链表存储结构，编写程序实现二叉树的先序、中序和后序遍历的递归和非递归算法以及层序遍历算法，并以适当的形式显示和保存二叉树及其相应的遍历序列； 3) 给定一个二叉树， 编写算法完成下列应用:（二选一） <ol style="list-style-type: none"> a) 判断其是否为完全二叉树； b) 求二叉树中任意两个结点的公共祖先。 			
<p>数据结构定义：</p> <pre>/*二叉树结构，左右链表定义*/ typedef struct Node { struct Node * lchild; struct Node * rchild; char data; }* BTREE, Node; /*队列定义*/ struct QUEUE {</pre>			

```

BTREE data[MAX];
int front;
int rear;
};

```

算法设计与分析（要求画出核心内容的程序流程图）：



先序遍历的递归（左）与非递归（右）

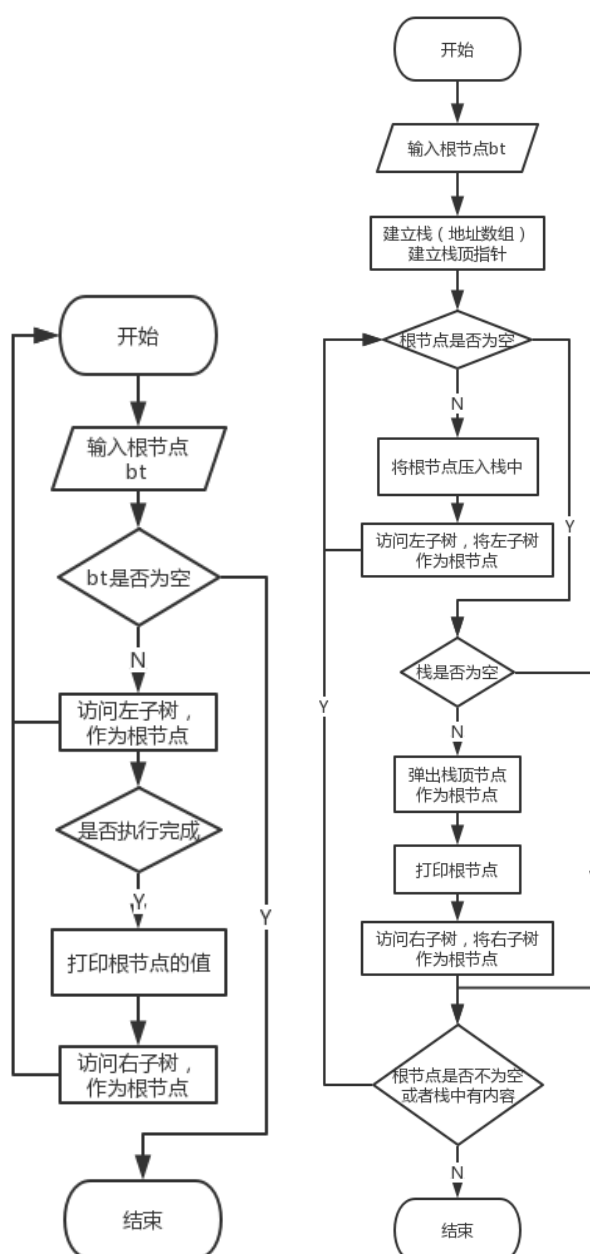
递归算法：

先打印当前根节点，再依次调用递归函数，分别输入左子树与右子树，时间复杂度为 $O(n)$

非递归算法：

使用栈来保存之前遍历节点，此时先输出根节点,再将根节点及其左路子树（左子树及其左子树的左子树等）存入栈中，直到左子树为空，再访问其右子树，并将右子树作为根节点寻找左路子树压入栈中，重复，这样实现先输出根节点，

在输出左子树，最后输出右子树顺序。节点遍历一次，时间复杂度为 $O(n)$



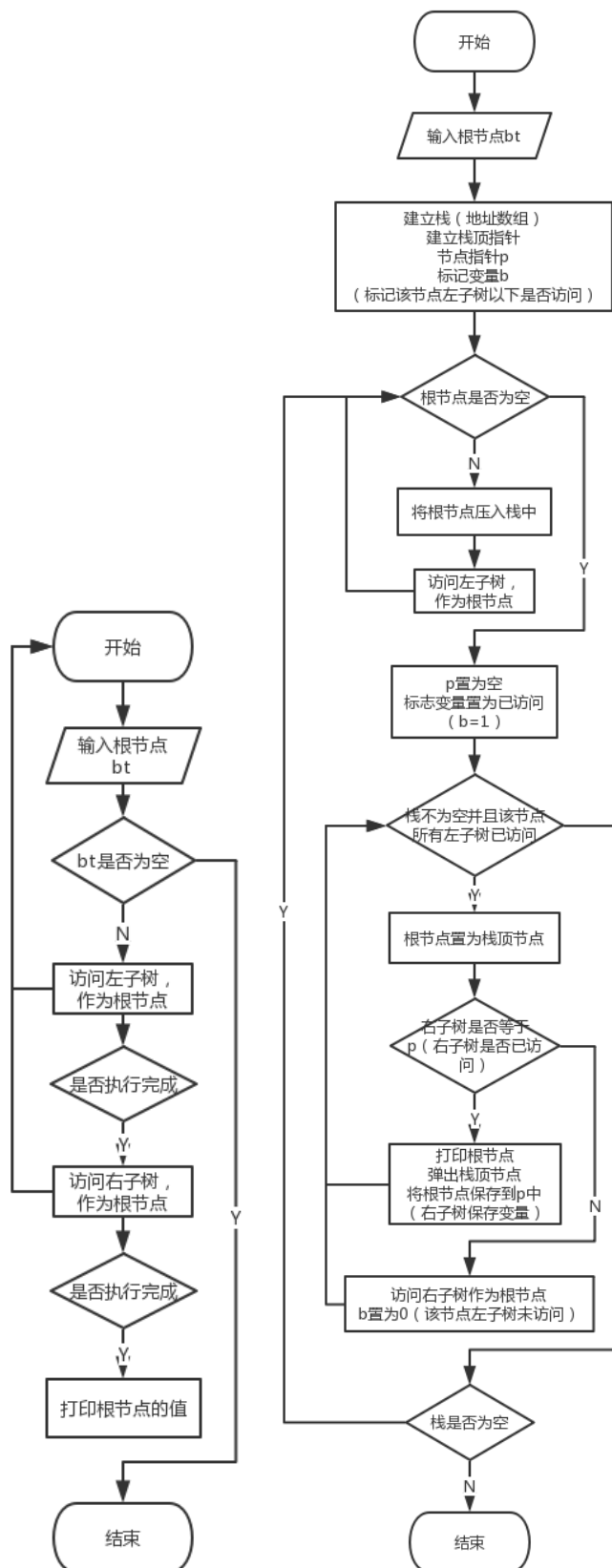
中序遍历的递归（左）与非递归（右）

递归算法：

先调用递归函数输入左子树，再打印当前根节点，最后调用递归函数，输入右子树，时间复杂度为 $O(n)$

非递归算法：

使用栈来保存之前遍历节点，此时先将根节点及其左路子树存入栈中，直到左子树为空，再输出根节点，再访问其右子树，并将右子树作为根节点寻找左路子树存入栈中，重复以上动作，这样实现先输出左子树，在输出根节点，最后输出右子树顺序。节点遍历一次，时间复杂度为 $O(n)$



后序遍历的递归（左）与非递归（右）

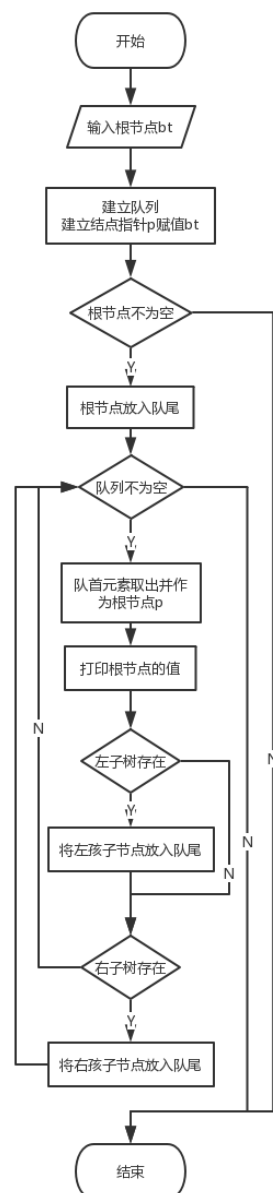
递归算法：

先调用递归函数输入左子树，再调用递归函数，输入右子树，最后打印当前

根节点，时间复杂度为 $O(n)$

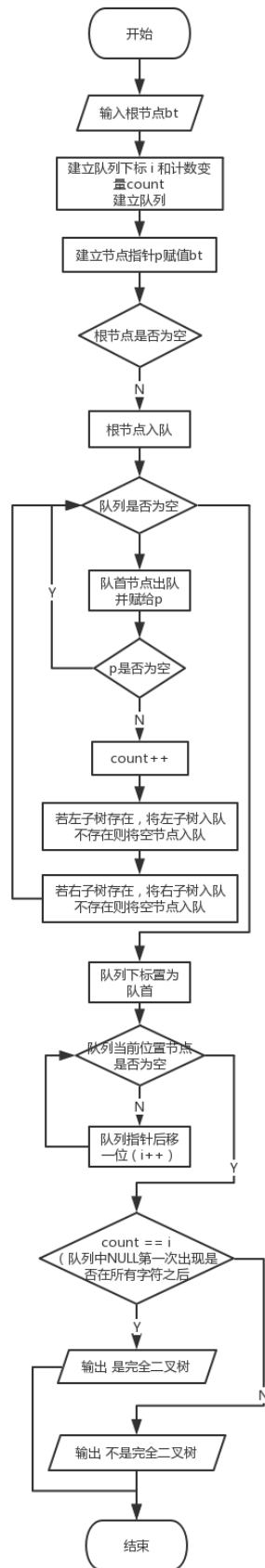
非递归算法：

使用栈来保存之前遍历节点，同时需要建立两个标识符，一个是保存已访问右子树节点的地址变量 p ，一个是标记当前节点的左路子树是否都已访问并压栈的标志变量 b 。此时先将根节点及其左路子树存入栈中，直到左子树为空，标记 b 为 1（即左路子树全入栈）， p 为 NULL（默认无右子树）。若栈不为空，接着将根节点置为栈顶节点，判断右子树是否已访问（ $=p$ ），如果已访问，则输出根节点，并将当前节点当作父节点的右子树保存到 p ，如果未访问（ $\neq p$ ），则将右子树作为根节点， b 置为 0（此节点左路子树未访问），重复上述操作。节点遍历一次，时间复杂度为 $O(n)$



层序遍历

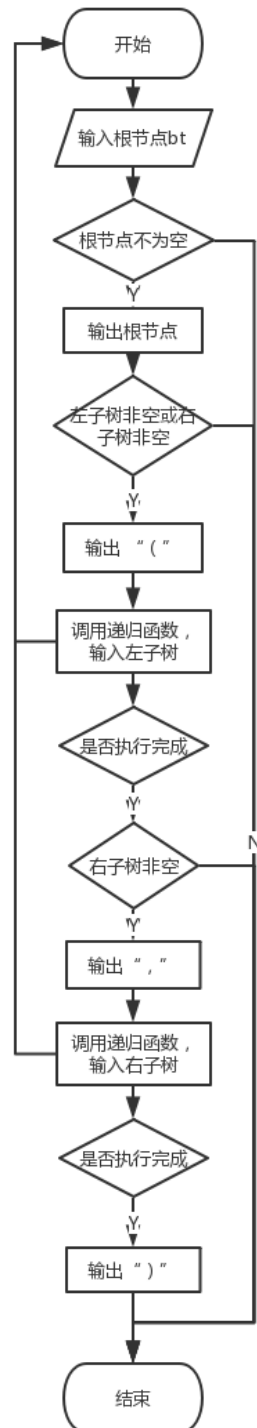
算法说明：设置队列，将根节点入队，之后每从队列中弹出一个节点，就将此节点的左子树节点和右子树节点入队，重复即实现层序遍历。时间复杂度为 $O(n)$



判断完全二叉树

算法说明：完全二叉树根据性质，如果用层序遍历的算法，将二叉树所有元素排

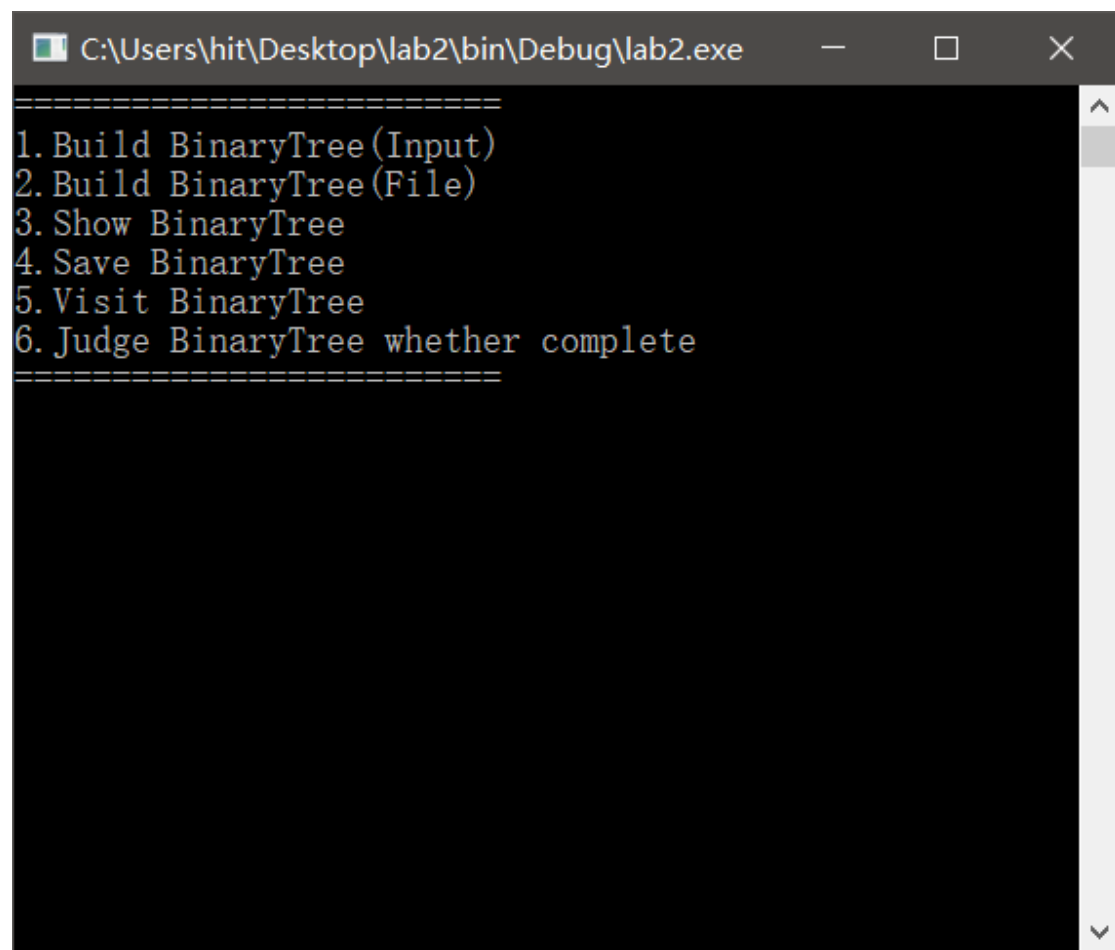
入队列中时,空节点 NULL 应该出现在所有字符的最后,即 a,b,c.....NULL,NULL,如果 NULL 出现在字符中间,则不是完全二叉树。根据这个方法,统计在二叉树中第一次 NULL 出现的位置 (i),如果不等于字符个数 (count),则不是完全二叉树。



广义表表示二叉树

算法说明: 又广义表定义可得上述算法

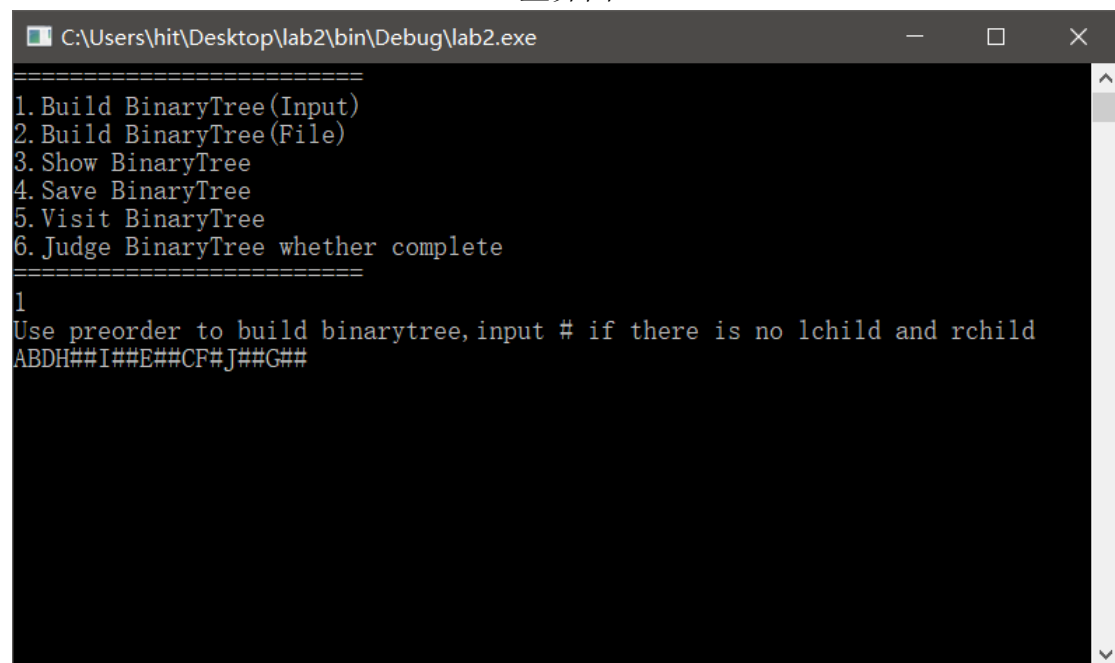
实验测试结果及结果分析:



A screenshot of a Windows application window titled "C:\Users\hit\Desktop\lab2\bin\Debug\lab2.exe". The window contains a menu with six options, each preceded by a number and a period. The options are: 1. Build BinaryTree(Input), 2. Build BinaryTree(File), 3. Show BinaryTree, 4. Save BinaryTree, 5. Visit BinaryTree, and 6. Judge BinaryTree whether complete. The menu is enclosed in a box with a title bar and standard Windows window controls.

```
=====
1. Build BinaryTree(Input)
2. Build BinaryTree(File)
3. Show BinaryTree
4. Save BinaryTree
5. Visit BinaryTree
6. Judge BinaryTree whether complete
=====
```

主界面



A screenshot of the same application window, now showing the input phase. The menu is still visible, but the user has selected option 1. Below the menu, the text "1" is entered, followed by a prompt: "Use preorder to build binarytree, input # if there is no lchild and rchild". The user has entered the input string "ABDH##I##E##CF#J##G##".

```
=====
1. Build BinaryTree(Input)
2. Build BinaryTree(File)
3. Show BinaryTree
4. Save BinaryTree
5. Visit BinaryTree
6. Judge BinaryTree whether complete
=====
1
Use preorder to build binarytree, input # if there is no lchild and rchild
ABDH##I##E##CF#J##G##
```

手动输入二叉树


```
C:\Users\hit\Desktop\lab2\bin\Debug\lab2.exe

=====
1. Build BinaryTree(Input)
2. Build BinaryTree(File)
3. Show BinaryTree
4. Save BinaryTree
5. Visit BinaryTree
6. Judge BinaryTree whether complete
=====
3
A(B(D(H, I), E), C(F(, J), G))
```

广义表显示二叉树

```
C:\Users\hit\Desktop\lab2\bin\Debug\lab2.exe

=====
1. Build BinaryTree(Input)
2. Build BinaryTree(File)
3. Show BinaryTree
4. Save BinaryTree
5. Visit BinaryTree
6. Judge BinaryTree whether complete
=====
5
=====
1. Preorder
2. Inorder
3. Postorder
4. Level list
=====
1
Recursion is: ABDHIECFJG
Nonrecursion is: ABDHIECFJG
Save complete
```

先序遍历二叉树（递归与非递归）

```
选择C:\Users\hit\Desktop\lab2\bin\Debug\lab2.exe

=====
1. Build BinaryTree(Input)
2. Build BinaryTree(File)
3. Show BinaryTree
4. Save BinaryTree
5. Visit BinaryTree
6. Judge BinaryTree whether complete
=====
5
=====
1. Preorder
2. Inorder
3. Postorder
4. Level list
=====
2
Recursion is: HDIBFAFJCG
Nonrecursion is: HDIBFAFJCG
Save complete
```

中序遍历（递归与非递归）

```
C:\Users\hit\Desktop\lab2\bin\Debug\lab2.exe

=====
1. Build BinaryTree(Input)
2. Build BinaryTree(File)
3. Show BinaryTree
4. Save BinaryTree
5. Visit BinaryTree
6. Judge BinaryTree whether complete
=====
5
=====
1. Preorder
2. Inorder
3. Postorder
4. Level list
=====
3
Recursion is: HIDEBJFGCA
Nonrecursion is: HIDEBJFGCA
Save complete
```

后序遍历（递归与非递归）

```
C:\Users\hit\Desktop\lab2\bin\Debug\lab2.exe

=====
1. Build BinaryTree(Input)
2. Build BinaryTree(File)
3. Show BinaryTree
4. Save BinaryTree
5. Visit BinaryTree
6. Judge BinaryTree whether complete
=====
5
=====
1. Preorder
2. Inorder
3. Postorder
4. Level list
=====
4
ABCDEFGHJIJ
Save complete
```

层序遍历

```
C:\Users\hit\Desktop\lab2\bin\Debug\lab2.exe
=====
1. Build BinaryTree(Input)
2. Build BinaryTree(File)
3. Show BinaryTree
4. Save BinaryTree
5. Visit BinaryTree
6. Judge BinaryTree whether complete
=====
6
ABCDEFGHIJ
Not a complete binarytree
```

判断是否是完全二叉树 (ABDH##I##E##CF#J##G##) (不是)

```
C:\Users\hit\Desktop\lab2\bin\Debug\lab2.exe
=====
1. Build BinaryTree(Input)
2. Build BinaryTree(File)
3. Show BinaryTree
4. Save BinaryTree
5. Visit BinaryTree
6. Judge BinaryTree whether complete
=====
6
AB
Is a complete binarytree
```

判断是否是完全二叉树 (AB###) (是)

问题及解决方法:
1. 如何判断队列为空? 设置队列的两个指针, 队首和队尾, 当队首位置超过队尾时, 队列即为空。
2. 如何实现文件输入二叉树? 将先序遍历算法改造, 遇到#符号作为空节点判断即可。
3. 如何判断完全二叉树? 比较层序遍历中空节点出现位置, 若出现在字符中间, 则不是完全二叉树。
源程序名称: 1163450201.c

注意: 正文文字为宋体小 4 号, 图中文字为宋体 5 号。行距为多倍行距 1.25。

源程序与此报告打包提交, 压缩包采用学号命名。