18340146_宋渝杰_实验 5

一、二叉树问题

算法分析：

因为要构造一个函数，用于交换二叉树所有的左右子节点，因此我采用层次遍历的方式，把二叉树所有的节点指针存入 vector 中，之后遍历 vector，交换所有节点的左右子节点即可（非递归方式）。

在实现该函数之前，构造了 struct Node 类型的节点，具有 int 类型的值（也是节点的编号）和 Node* 类型的左右节点；class 类型的 Tree，具有 Node* 类型的 root 根节点，构造函数，三种遍历函数，insert 插入子节点函数，以及一些相关的函数。

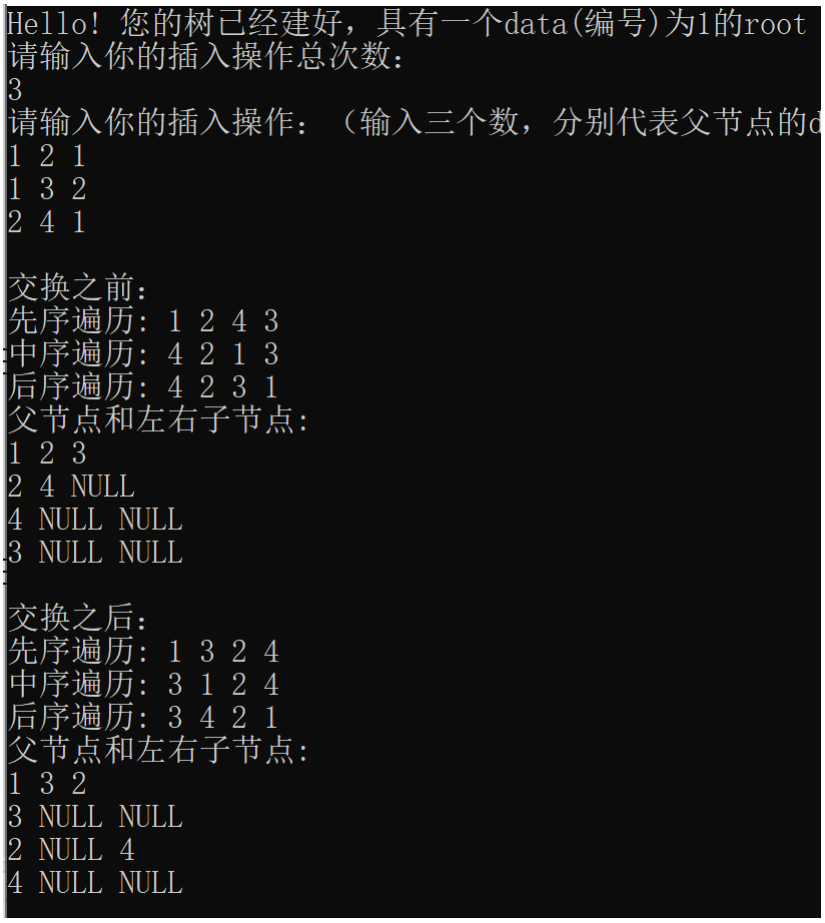
流程：

程序首先建立好了一棵树，只有 root 根节点，值（编号）为 1，之后程序提示输入插入操作的总次数、插入具体操作，完成之后程序便构造好了一棵二叉树，之后程序输出交换之前的树的各种遍历和表达方式，再自行调用交换函数，输出交换之后的各种遍历和表达方式。

程序测试：

1、较简单的二叉树：

（输入提示，先输入插入操作总次数 n，之后输入 n 行数据，每行第一个数为父节点的编号，第二个数为子节点的编号，第三个数决定是左子节点还是右子节点（1 代表左，2 代表右））



可以通过“父节点和左右子节点”画出树来检测数据，会发现满足插入和情况，遍历也是正确的。

2、复杂样例:

```
Hello! 您的树已经建好，具有一个data(编号)为1的root
请输入你的插入操作总次数:
7
请输入你的插入操作：（输入三个数，分别代表父节点的d
1 2 1
1 3 2
2 4 1
3 5 1
3 6 2
4 7 1
7 8 2

交换之前:
先序遍历: 1 2 4 7 8 3 5 6
中序遍历: 7 8 4 2 1 5 3 6
后序遍历: 8 7 4 2 5 6 3 1
父节点和左右子节点:
1 2 3
2 4 NULL
4 7 NULL
7 NULL 8
8 NULL NULL
3 5 6
5 NULL NULL
6 NULL NULL

交换之后:
先序遍历: 1 3 6 5 2 4 7 8
中序遍历: 6 3 5 1 2 4 8 7
后序遍历: 6 5 3 8 7 4 2 1
父节点和左右子节点:
1 3 2
3 6 5
6 NULL NULL
5 NULL NULL
2 NULL 4
4 NULL 7
7 8 NULL
8 NULL NULL
```

3、异常处理:

```
Hello! 您的树已经建好，具有一个data(编号)为1的root
请输入你的插入操作总次数:
4
请输入你的插入操作：（输入三个数，分别代表父节点的da
1 2 1
1 3 2
2 4 1
1 5 2

交换之前:
先序遍历: 1 2 4 3
中序遍历: 4 2 1 3
后序遍历: 4 2 3 1
父节点和左右子节点:
1 2 3
2 4 NULL
4 NULL NULL
3 NULL NULL

交换之后:
先序遍历: 1 3 2 4
中序遍历: 3 1 2 4
后序遍历: 3 4 2 1
父节点和左右子节点:
1 3 2
3 NULL NULL
2 NULL 4
4 NULL NULL
```

与第一个测试数据相比，插入操作中 1 5 2 指的是在编号为 1 的节点右边插入编号为 5 的节点，和前面的 1 3 2 冲突（编号为 1 的节点已经有了右子节点），因此这里不进行 1 5 2 的插入过程，因此最后的树与第一个测试数据相同。