2020/06/05_数据结构_第6课_二叉搜索树的遍历

笔记本: 数据结构
创建时间: 2020/6/6 星期六 9:52
作者: ileemi
标签: 二叉搜索树的遍历

二叉搜索树的遍历

1、**前序遍历**：先自己，再左孩子，再右孩子
2、**中序遍历**：先左孩子，再自己，再右孩子
3、**后续遍历**：先左孩子，再有孩子，再自己
4、**层次遍历**：从上层到下层，每层从左到右依次遍历

下面三种遍历方式了解即可，这里先讨论上面的四种遍历方式：

5、**逆前序遍历**
6、**逆中序遍历**
7、**逆后续遍历**

前序遍历

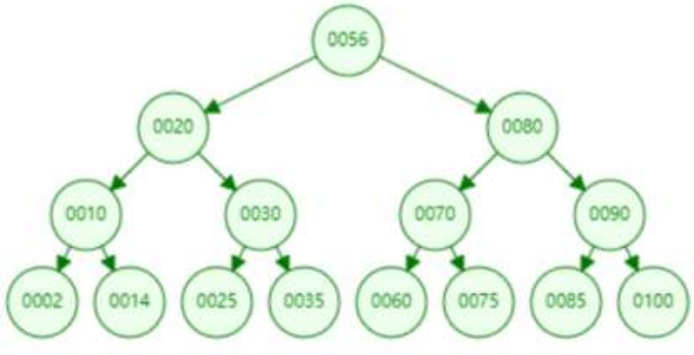
二叉树中序遍历的实现思想是：

- 访问根结点；
- 访问当前结点的左子树；
- 访问当前结点的右子树；

先自己，再左孩子，再右孩子

递归

示例数据：
56 20 80 10 30 70 90 2 14 25 35 60 75 85 100



使用递归前序遍历输出的数据依次是（参考上图）：

56 20 10 2 14 30 25 35 80 70 60 75 90 85 100

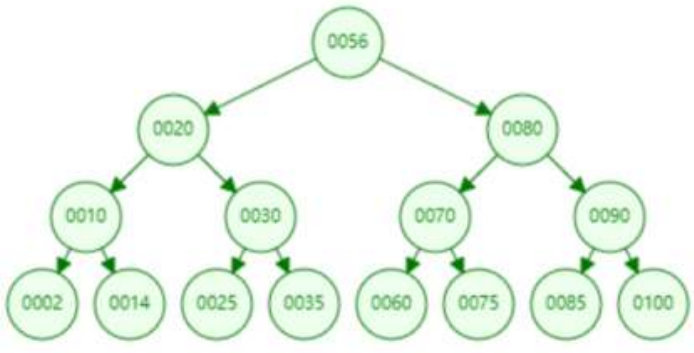
非递归

递归的底层实现依靠的是 栈 存储结构，因此，二叉树的先序遍历既可以直接采用递归思想实现，也可以使用栈的存储结构去模拟递归的思想实现。

首先需要对其进行分析：

找规律：
结点输出时，将该结点的右孩子存储起来，当左孩子输出后，其右孩子从存储数据的地方出来，每次结点输出时，都将右孩子存储起来。

示例数据：
56 20 80 10 30 70 90 2 14 25 35 60 75 85 100



输出	存入数据
56	80
20	80 30
10	80 30 14
02	80 30 14
14	80 30
30	80 35
25	80 35
35	80
80	90
70	90 75
60	90 75
75	90
90	100
85	100
100	nullptr

编写程序，程序输出结果：

前序-用递归：56 20 10 2 14 30 25 35 80 70 60 75 90 85 100
前序-非递归：56 20 10 2 14 30 25 35 80 70 60 75 90 85 100

中序遍历（常用）

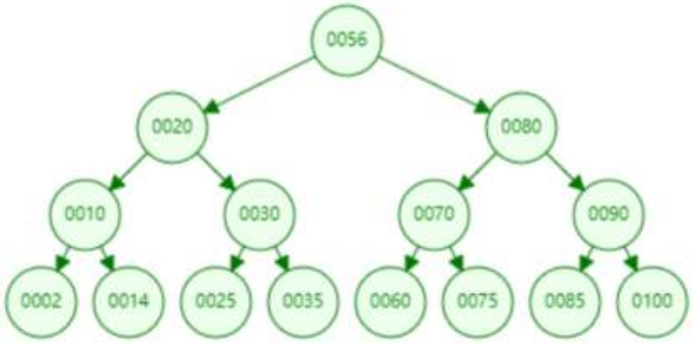
二叉树中序遍历的实现思想是：

- 访问当前结点的左子树；
- 访问根结点；
- 访问当前结点的右子树；

先左孩子，再自己，再右孩子

递归

示例数据：
56 20 80 10 30 70 90 2 14 25 35 60 75 85 100



使用递归中序遍历输出的数据依次是（参考上图）：

2 10 14 20 25 30 35 **56** 60 70 75 80 85 90 100

非递归

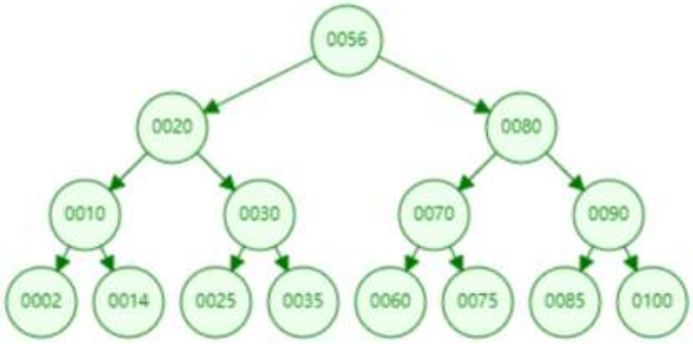
中序遍历的非递归方式实现思想：

从根结点开始，遍历所有结点的左孩子，也就是遍历的同时依次将其都压入栈内，当遍历结束，就是最后一个结点没有左孩子时，这个时候从栈中取出刚才依次存储的数据，取出来一个，之后访问取出来数据的结点的右孩子，重复上述步骤，如果右孩子是叶子结点，就输出数据，之后再次从栈中输出数据。

非递归的中序遍历：

存储	输出
56	
56 20	
56 20 10	
56 20 10 02	
56 20 10	02
56 20	02 10
56 20 14	02 10
56 20	02 10 14
56	02 10 14 20
56 30	02 10 14 20
56 30 25	02 10 14 20
56 30	02 10 14 20 25
56	02 10 14 20 25 30
56 35	02 10 14 20 25 30
56	02 10 14 20 25 30 35
	02 10 14 20 25 30 35 56
80	02 10 14 20 25 30 35 56
80 70	02 10 14 20 25 30 35 56
80 70 60	02 10 14 20 25 30 35 56
80 70	02 10 14 20 25 30 35 56 60
80	02 10 14 20 25 30 35 56 60 70
80 75	02 10 14 20 25 30 35 56 60 70
80	02 10 14 20 25 30 35 56 60 70 75
	02 10 14 20 25 30 35 56 60 70 75 80
90	02 10 14 20 25 30 35 56 60 70 75 80
90 85	02 10 14 20 25 30 35 56 60 70 75 80
90	02 10 14 20 25 30 35 56 60 70 75 80 85
	02 10 14 20 25 30 35 56 60 70 75 80 85 90
100	02 10 14 20 25 30 35 56 60 70 75 80 85 90
	02 10 14 20 25 30 35 56 60 70 75 80 85 90 100

示例数据：
56 20 80 10 30 70 90 2 14 25 35 60 75 85 100



手动模拟：

非递归的中序遍历：

存储	输出
56	
56 20	
56 20 10	
56 20 10 02	
56 20 10	02
56 20	02 10
56 20 14	02 10
56 20	02 10 14
56	02 10 14 20
56 30	02 10 14 20
56 30 25	02 10 14 20
56 30	02 10 14 20 25
56	02 10 14 20 25 30
56 35	02 10 14 20 25 30
56	02 10 14 20 25 30 35
	02 10 14 20 25 30 35 56
80	02 10 14 20 25 30 35 56
80 70	02 10 14 20 25 30 35 56
80 70 60	02 10 14 20 25 30 35 56
80 70	02 10 14 20 25 30 35 56 60
80	02 10 14 20 25 30 35 56 60 70
80 75	02 10 14 20 25 30 35 56 60 70
80	02 10 14 20 25 30 35 56 60 70 75
	02 10 14 20 25 30 35 56 60 70 75 80
90	02 10 14 20 25 30 35 56 60 70 75 80
90 85	02 10 14 20 25 30 35 56 60 70 75 80
90	02 10 14 20 25 30 35 56 60 70 75 80 85
	02 10 14 20 25 30 35 56 60 70 75 80 85 90
100	02 10 14 20 25 30 35 56 60 70 75 80 85 90
	02 10 14 20 25 30 35 56 60 70 75 80 85 90 100

后序遍历

二叉树中序遍历的实现思想是：

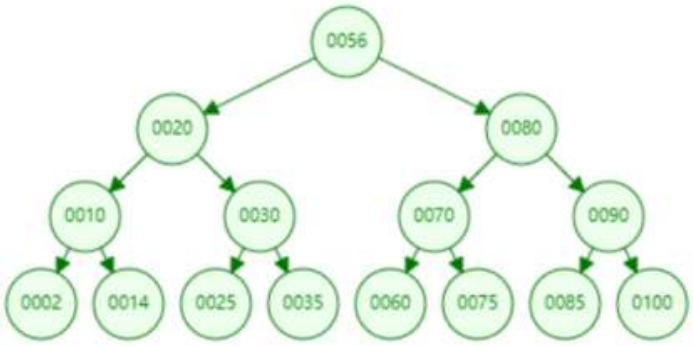
- 访问当前结点的左子树；
- 访问当前结点的右子树；
- 访问根结点；

先左孩子，再右孩子，再自己

递归

示例数据：

56 20 80 10 30 70 90 2 14 25 35 60 75 85 100



使用递归后序遍历输出的数据依次是（参考上图）：

2 14 10 25 35 30 20 60 75 70 85 100 90 80 **56**

非递归

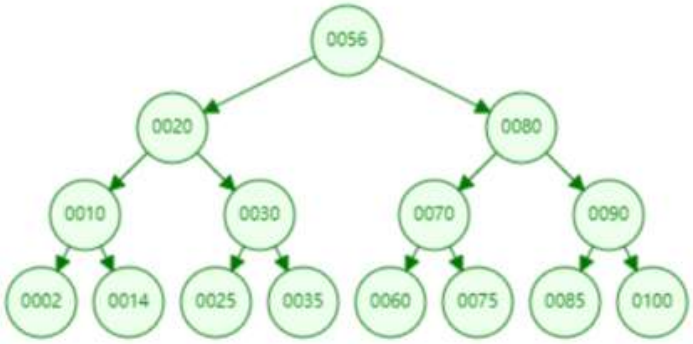
递归算法底层的实现使用的是栈存储结构，所以可以直接使用栈写出相应的非递归算法。

后续遍历是在输出处理完当前结点左右孩子之后，再处理当前结点的。

处理当前结点的时机：需要判断上次处理的数据是不是当前结点的右孩子。
示例：

- 在将上述数据的根节点下的左孩子都入栈后，最后一个结点为叶子结点，弹出。
- 再次从栈顶弹出一个数据，此时就需要判断，该结点的右孩子是否已经处理，发现上次处理的右孩子是其左孩子的右孩子。所以此时这个结点的右孩子还没有进行处理，该结点不进行处理，需要先去处理其右孩子。
- 或者当该结点的右孩子为空的时候，也可以处理该结点。

示例数据：
56 20 80 10 30 70 90 2 14 25 35 60 75 85 100



手动模拟：

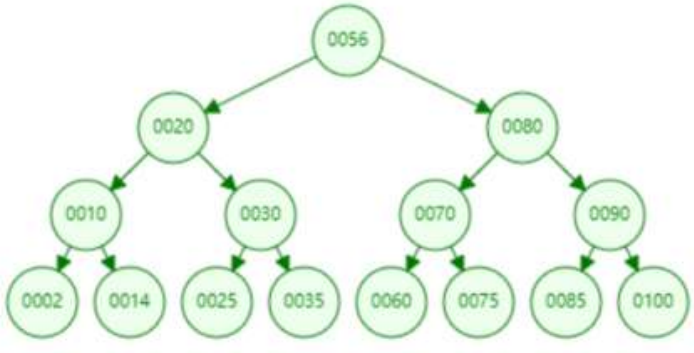
存储	输出	
56 20 10 02		
56 20 10	02	
56 20 10	02	-->栈上取出10, 不能处理, 再放入栈中
56 20 10 14	02	
56 20 10	02 14	
56 20	02 14 10	
56 20	02 14 10	-->栈上取出20, 不能处理, 再放入栈中
56 20 30	02 14 10	
56 20 30 25	02 14 10	
56 20 30	02 14 10 25	
56 20 30	02 14 10 25	-->栈上取出30, 不能处理, 再放入栈中
56 20 30 35	02 14 10 25	
56 20 30	02 14 10 25 35	
56 20	02 14 10 25 35 30	
56	02 14 10 25 35 30 20	
56	02 14 10 25 35 30 20	-->栈上取出56, 不能处理, 再放入栈中
56 80 70 60	02 14 10 25 35 30 20	
56 80 70	02 14 10 25 35 30 20 60	
56 80 70	02 14 10 25 35 30 20 60	-->栈上取出70, 不能处理, 再放入栈中
56 80 70 75	02 14 10 25 35 30 20 60	
56 80 70	02 14 10 25 35 30 20 60 75	
56 80	02 14 10 25 35 30 20 60 75 70	
56 80	02 14 10 25 35 30 20 60 75 70	-->栈上取出80, 不能处理, 再放入栈中
56 80 90 85	02 14 10 25 35 30 20 60 75 70	
56 80 90	02 14 10 25 35 30 20 60 75 70 85	
56 80 90	02 14 10 25 35 30 20 60 75 70 85	-->栈上取出90, 不能处理, 再放入栈中
56 80 90 100	02 14 10 25 35 30 20 60 75 70 85	
56 80 90	02 14 10 25 35 30 20 60 75 70 85 100	
56 80	02 14 10 25 35 30 20 60 75 70 85 100 90	
56	02 14 10 25 35 30 20 60 75 70 85 100 90 80	
	02 14 10 25 35 30 20 60 75 70 85 100 90 80 56	

层次遍历

按照二叉树中的层次从左到右依次遍历每层中的结点。

具体的实现思路：通过使用队列的数据结构，从树的根结点开始，依次将其左孩子和右孩子依次入队。之后当，每个结点需要出队时，都将该结点的左孩子和右孩子入队，直到数中所有的结点都出队，出队结点的先后顺序就是层次遍历的最终结果。

示例数据：
56 20 80 10 30 70 90 2 14 25 35 60 75 85 100



根据上图 其层次遍历的实现过程如下：

- 根结点 56 入队（队列：56）
- 根结点 56 出队，其左孩子 20 和 右孩子 80 入队（队列：20 80）
- 结点 20 出队，其左孩子 10 和 右孩子 30 入队（队列：80 10 30）
- 结点 80 出队，其左孩子 70 和 右孩子 90 入队（队列：10 30 70 90）
- 结点 10 出队，其左孩子 2 和 右孩子 14 入队（队列：30 70 90 2 14）
- 结点 30 出队，其左孩子 25 和 右孩子 35 入队（队列：70 90 2 14 25 35）
- 结点 70 出队，其左孩子 60 和 右孩子 75 入队（队列：90 2 14 25 35 60 75）
- 结点 90 出队，其左孩子 85 和 右孩子 100 入队（队列：2 14 25 35 60 75 85 100）
- 结点 02 出队，其没有左孩子和右孩子
- 结点 14 出队，其没有左孩子和右孩子
- 结点 25 出队，其没有左孩子和右孩子
- 结点 35 出队，其没有左孩子和右孩子
- 结点 60 出队，其没有左孩子和右孩子
- 结点 75 出队，其没有左孩子和右孩子
- 结点 85 出队，其没有左孩子和右孩子
- 结点 100 出队，其没有左孩子和右孩子

最后输出结果：56 20 80 10 30 70 90 2 14 25 35 60 75 85 100