使用非递归实现二叉树的遍历

主讲人 令狐冲

二叉树三种遍历

先序遍历 Pre-order

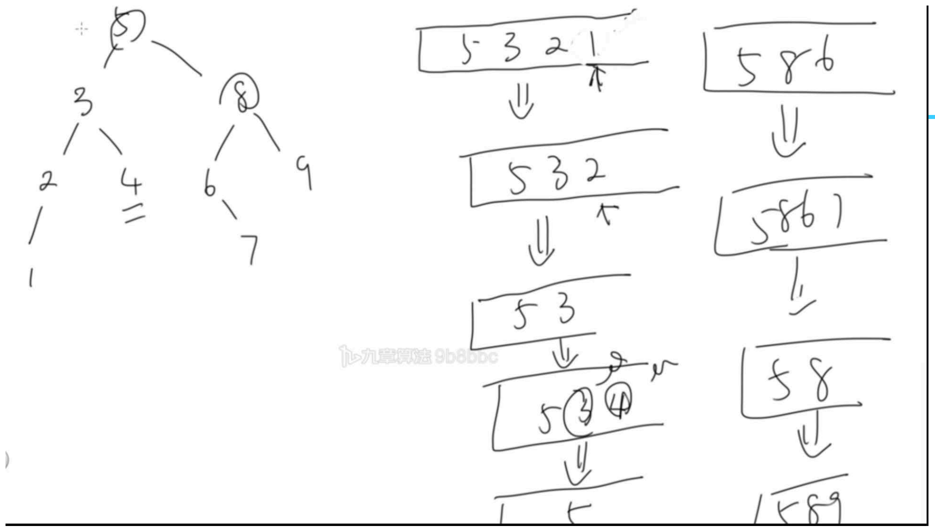
中序遍历 In-order

后序遍历 Post-order（分治法）

Binary Search Tree Iterator

https://www.lintcode.com/problem/binary-search-tree-iterator/

通过实现 hasNext 和 next 两个方法，从而实现
二叉查找树的中序遍历迭代器



实现要点

递归 → 非递归，意味着自己需要控制原来由操作系统控制的**栈**的进进出出

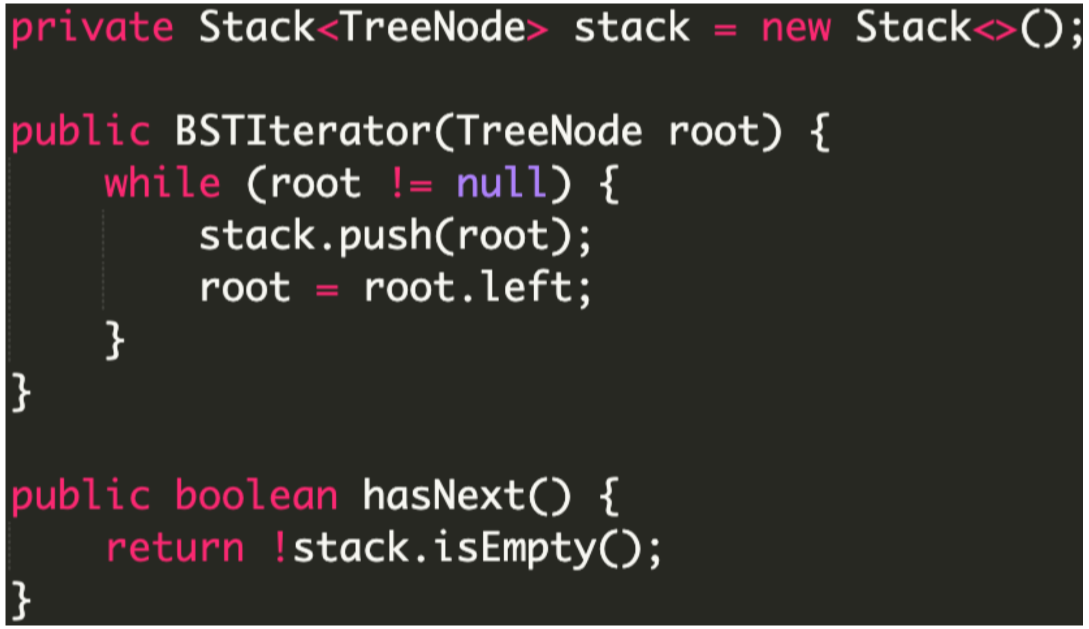
如何找到最小的第一个点？最左边的点即是

如何求出一个二叉树节点在中序遍历中的下一个节点？

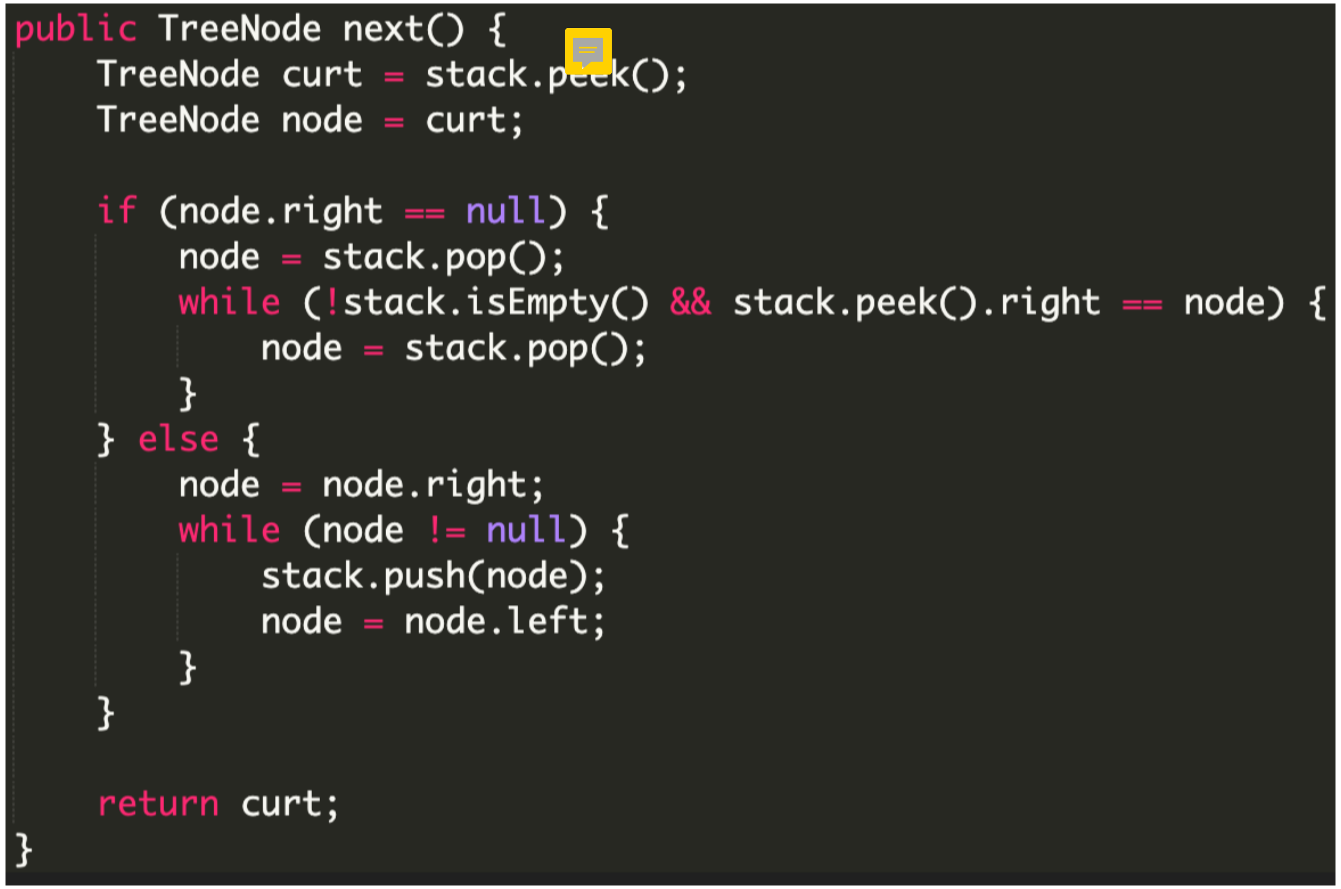
在 stack 中记录从根节点到当前节点的整条路径

下一个点=右子树最小点 or 路径中最近一个通过左子树包含当前点的点

Java 代码

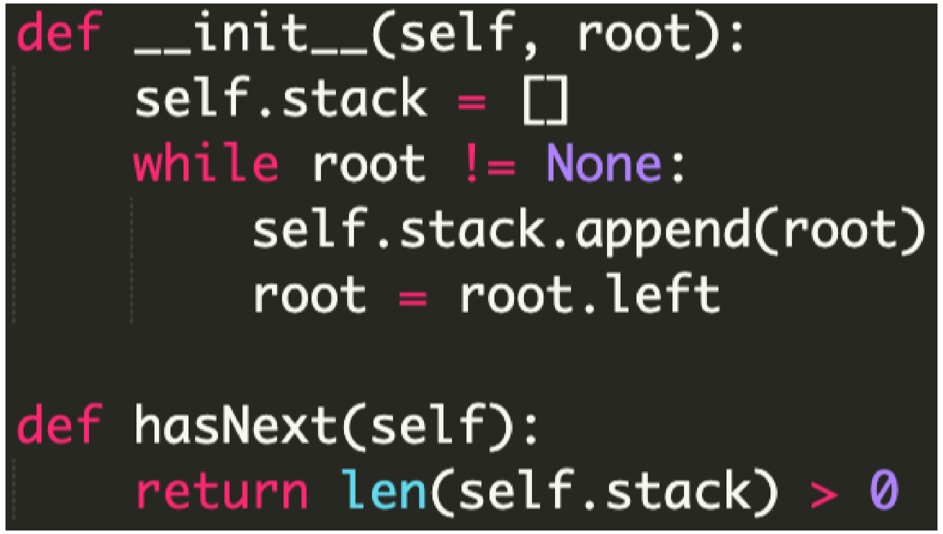


```java
private Stack<TreeNode> stack = new Stack<>();

public BSTIterator(TreeNode root) {
    while (root != null) {
        stack.push(root);
        root = root.left;
    }
}

public boolean hasNext() {
    return !stack.isEmpty();
}
```

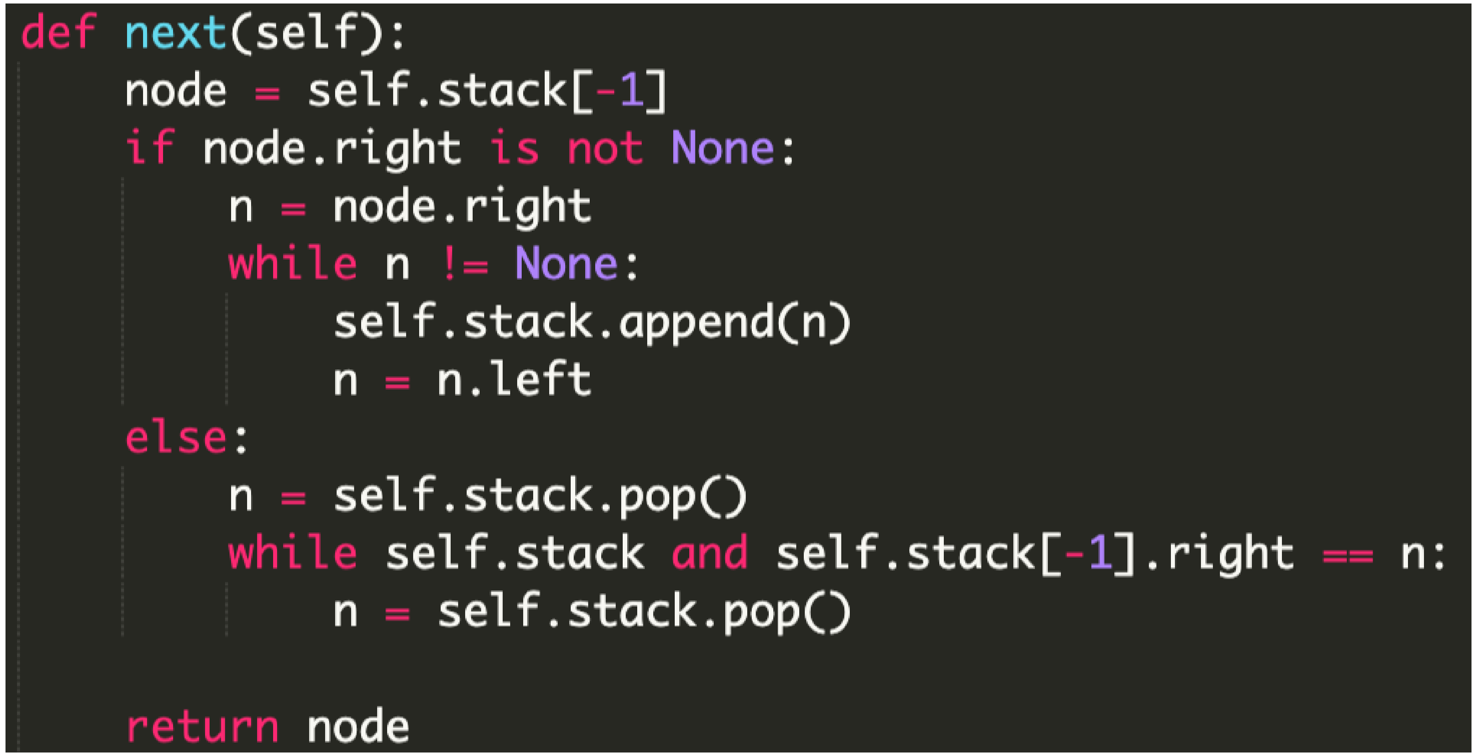


```java
public TreeNode next() {
    TreeNode curt = stack.peek();
    TreeNode node = curt;

    if (node.right == null) {
        node = stack.pop();
        while (!stack.isEmpty() && stack.peek().right == node) {
            node = stack.pop();
        }
    } else {
        node = node.right;
        while (node != null) {
            stack.push(node);
            node = node.left;
        }
    }

    return curt;
}
```

Python 代码



```python
def __init__(self, root):
    self.stack = []
    while root != None:
        self.stack.append(root)
        root = root.left

def hasNext(self):
    return len(self.stack) > 0
```



```python
def next(self):
    node = self.stack[-1]
    if node.right is not None:
        n = node.right
        while n != None:
            self.stack.append(n)
            n = n.left
    else:
        n = self.stack.pop()
        while self.stack and self.stack[-1].right == n:
            n = self.stack.pop()

    return node
```

一种更简单的实现方式

在 stack 中不保存哪些已经被 iterator 访问过的节点
即如果 iterate 到了这个节点，即便右子树还未完全遍历
也从 stack 里踢出

简单的实现方式代码

```python
class BSTIterator:
    def __init__(self, root):
        self.stack = []
        self.find_most_left(root)

    def find_most_left(self, node):
        while node:
            self.stack.append(node)
            node = node.left

    def hasNext(self):
        return bool(self.stack)

    def next(self):
        node = self.stack.pop()
        if node.right:
            self.find_most_left(node.right)
        return node
```

```java
public Stack<TreeNode> stack;

public BSTIterator(TreeNode root) {
    stack = new Stack<>();
    findMostLeft(root);
}

private void findMostLeft(TreeNode node) {
    while (node != null) {
        stack.add(node);
        node = node.left;
    }
}

public boolean hasNext() {
    return !stack.isEmpty();
}

public TreeNode next() {
    TreeNode node = stack.pop();
    if (node.right != null) {
        findMostLeft(node.right);
    }
    return node;
}
```

这里的 stack 算全局变量么？

类内部不同的函数都可以访问 stack 和修改