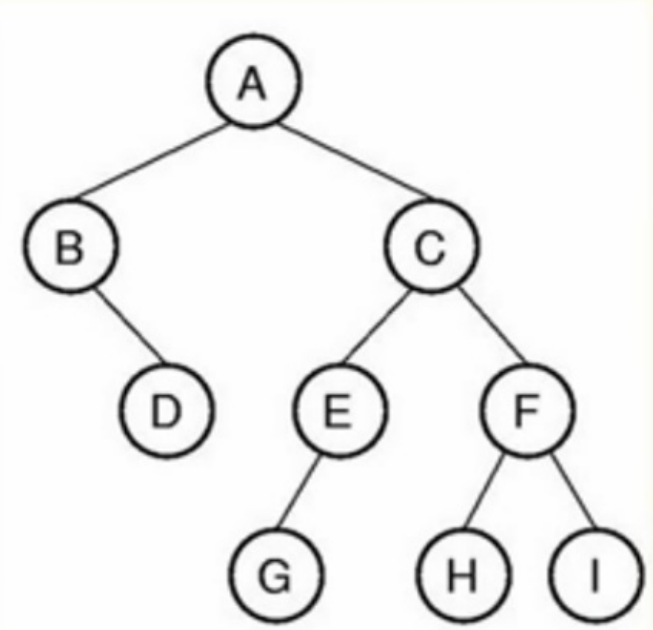
回溯法实现遍历二叉树

- **回溯法**
  - **概述**：回溯法都要借助 " **栈** " 实现 " 回溯 " （后退一步、再退一步.....对应于 **stack.pop()** ）、都要有大循环（每次循环代表每一步进行什么操作）且大循环的结束条件是栈为空
  - **模板**：

```
def preOrderIter(root):
    # 回溯法通过 " 栈 " 实现
    s = []
    node = root
    s.append(node)

  # 空栈 : False
  # 非空栈 : True
    while s :
       pass

-----------------------------------------------------------------------------------------------------

# 使用这种结构就不会出现 代码-1 中的问题
def preOrderIter (root):
    # 回溯法通过 栈 实现
    s = []
    node = root

    while True:
    # 空栈 : False
    # 非空栈 : True
    # 这里实现了空栈是循环结束的条件
        if not s:
            break
```

  - **思路**



    - **先序遍历**：访问当前节点（ **B** ）→ 将当前节点压栈 → 访问当前节点的左子树（左节点）→ 左子树（左节点）存在继续访问左子树 → 左子树不存在且栈不为空、弹栈（ **B 弹出** ）访问右子树（右节点）。
    - **中序遍历**：
    - **后续遍历**：
  - **代码-1：先序遍历**

```
node: A
stack: [<A>]

node: B
stack: [<A>, <B>]

node: D
stack: [<A>, <D>]

node: C
stack: [<C>]

node: E
stack: [<C>, <E>]

node: G
stack: [<C>, <E>, <G>]

node: F
stack: [<F>]

node: H
stack: [<F>, <H>]

node: I
stack: [<I>]
```

```
def preOrderIter(root):
    # 回溯法通过 栈 实现
    s = []
    node = root
  # 栈为空是大循环结束的条件，所以先向栈中添加一个节点使得能够开始循环，否则一开始栈为空不会进行循环
    s.append (node)

    while s :
        while node:
            print node
      # 前面的 s.append(node) 会导致第一个节点 A 又一次进入了栈中，此时栈中的情况 [A, A]
      # 这种模板还是有点儿问题的，放弃，使用老师的
            s.append(node)
      # 打印栈中元素
            print s
            node = node. left
        node = s. pop().right
```

```
def preOrderIter (root):
    s = []
    node = root

    while True:
        while node:
            print node
            s.append( node)
            node = node. left

    # if 的空栈判断必须放到这里
    # 如果放到 while node 循环之前会导致无法进行 while node 循环（一开始是空栈）
    # 如果放到 node = s.pop().right 之后会导致 A 出栈栈为空 跳出循环 OR 已经是空栈了还要弹栈 报错
        if not s:
```

```
            break

        node = s.pop().right
```

    - **代码-2：中序遍历**
    - **代码-3：后续遍历**
- **And So On**