二叉树+遍历：二叉树的中序遍历

中等/二叉树、中序遍历、莫里斯遍历

学习目标

拉勾教育
—互联网人实战大学—

- 掌握二叉树的基本分类
- 了解二叉树的前序、后序遍历的方法
- 掌握二叉树的中序遍历方法

题目描述

给定一棵二叉树，返回其中序遍历序列

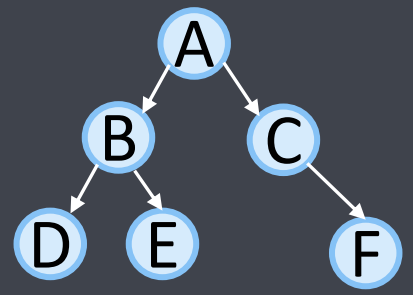
```
输入: [1,null,2,3]
   1
    \
     2
    /
   3

输出: [1,3,2]
```

一. Comprehend 理解题意

本题题目要求给定一棵二叉树，返回**中序遍历**序列

- 二叉树概念
- 中序遍历概念



[D,B,E,A,C,F]

一. Comprehend 理解题意

二叉树：

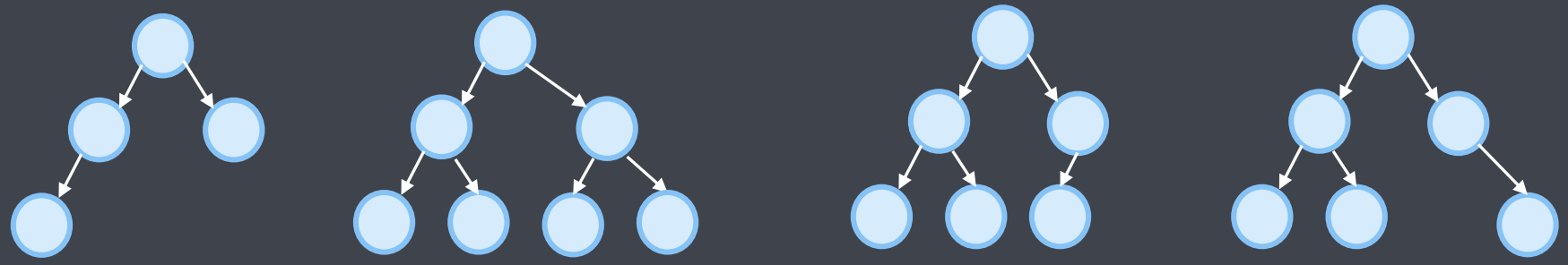
每个结点最多有两棵子树且左右子树的顺序不能调换

满二叉树：

如果二叉树中任意一个结点，或者是叶结点，或者有两棵非空子树，且叶结点都集中在二叉树最下面一层

完全二叉树：

若二叉树最多只有最下面两层结点的度可以小于2，且最下面一层结点都依次排列在该层最左边的位置上



二叉树　　满二叉树　　完全二叉树　　非完全二叉树

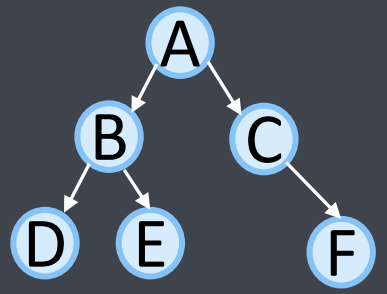
一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

二叉树的遍历

前序遍历：

1. 访问根结点
2. 以前序遍历方式遍历根结点的左子树
3. 以前序遍历方式遍历根结点的右子树

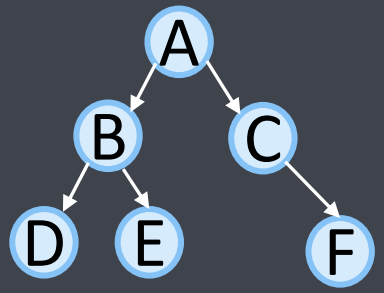


[A,B,D,E,C,F]

根->左->右

中序遍历：

1. 以中序遍历方式遍历根结点的左子树
2. 访问根结点
3. 以中序序遍历方式遍历根结点的右子树

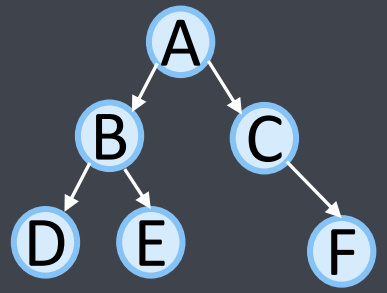


[D,B,E,A,C,F]

左->根->右

后序遍历：

1. 以后序遍历方式遍历根结点的左子树
2. 以后序遍历方式遍历根结点的右子树
3. 访问根结点



[D,E,B,F,C,A]

左->右->根

二. Choose 数据结构及算法思维选择

数据结构选择

题目已经限定为二叉树

算法思维选择

- 迭代

  迭代遍历所有节点

- 递归

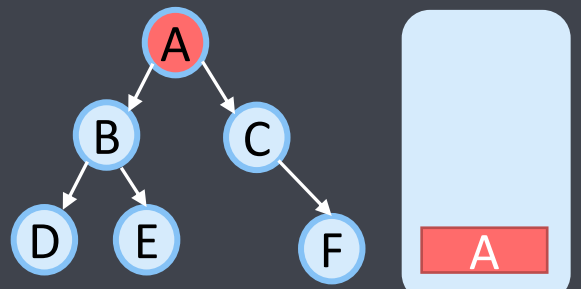
  DFS递归遍历所有节点

三. Code 基本解法及编码实现

解法一： 迭代

由于递归方式实现中序遍历相对简单，面试时常常要求使用迭代方法

递归中隐式的维护了一个栈，在迭代中需要显式的将这个栈模拟出来



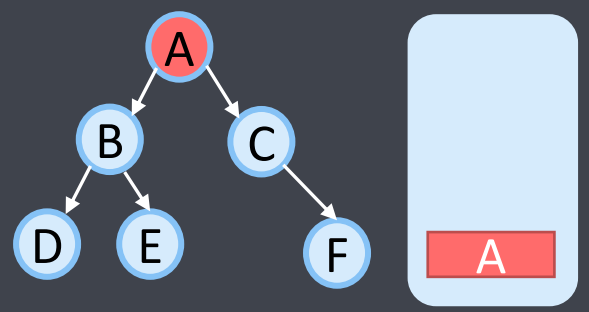
res=[]

思考：

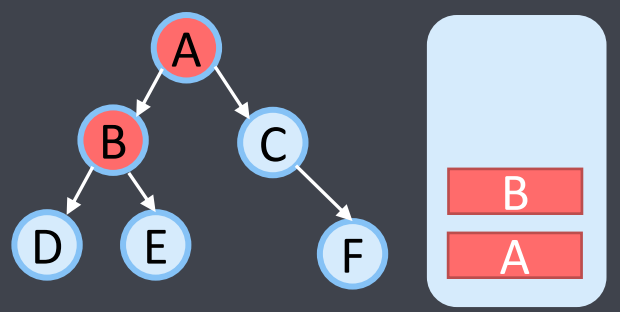
- 是否需要特别处理二叉树为空的情况？
- 入栈的顺序是什么？
- 为了得到中序遍历序列，出栈顺序是什么?

三. Code 基本解法及编码实现

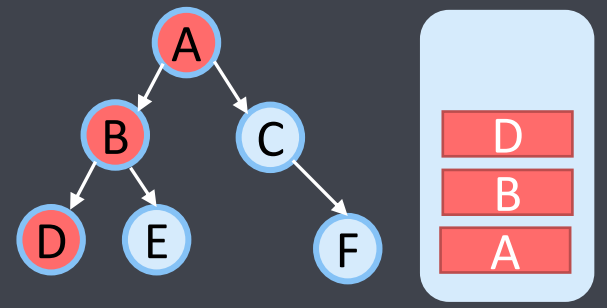
解法一： 迭代



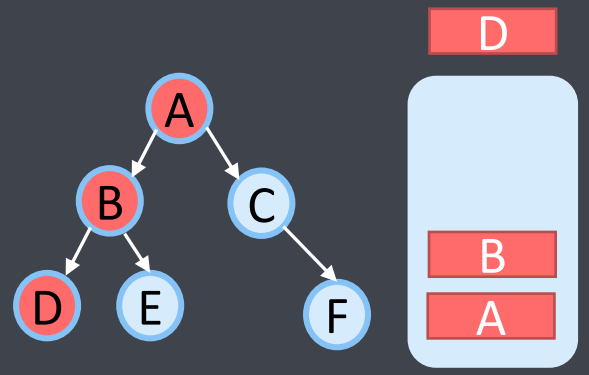
res=[]



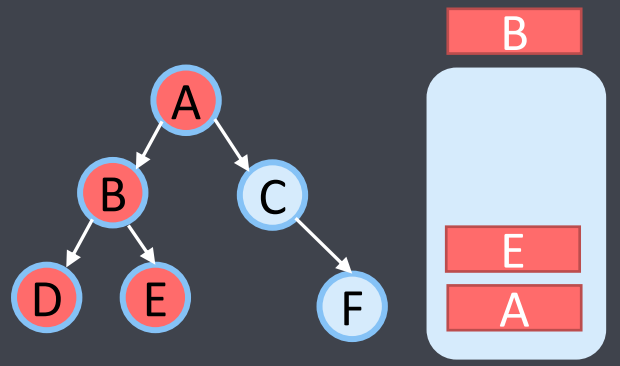
res=[]



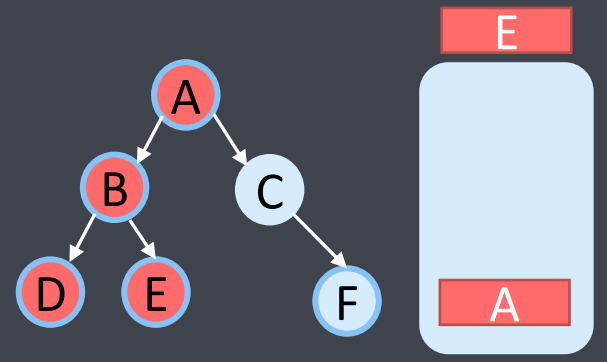
res=[]



res=[D]



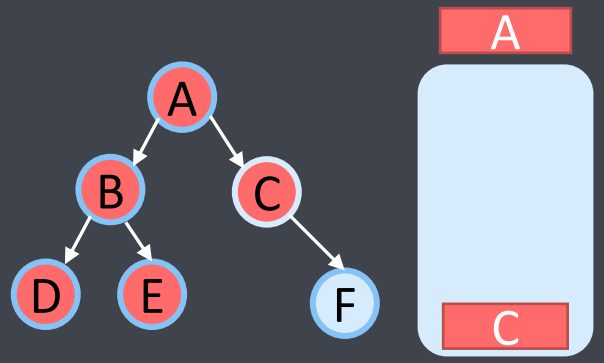
res=[D,B]



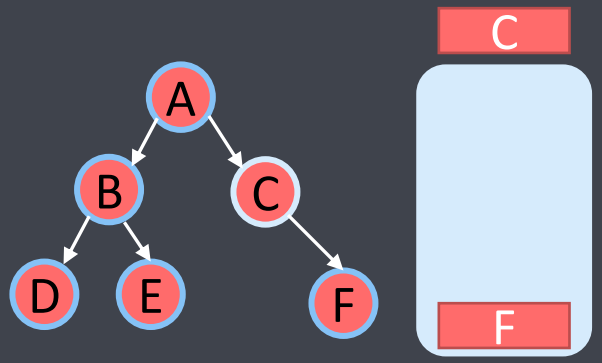
res=[D,B,E]

三. Code 基本解法及编码实现

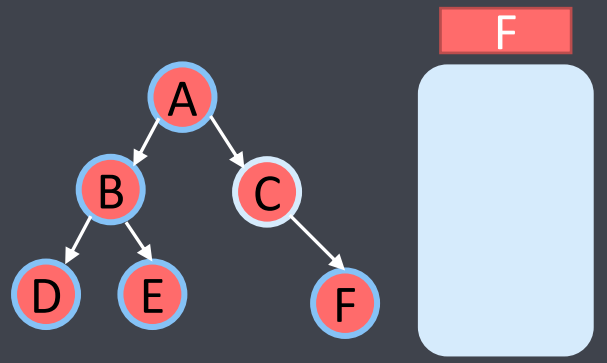
解法一： 迭代



res=[D,B,E,A]



res=[D,B,E,A,C]



res=[D,B,E,A,C,F]

三. Code 基本解法及编码实现

解法一： 迭代

时间复杂度：O(n)

- n为二叉树结点个数
- 每个结点只会被访问一次

空间复杂度：

- 取决于栈的深度
- 在二叉树为一条链的时候最大为O(n)

三. Code 基本解法及编码实现

解法一： 迭代编码实现


```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<Integer>();// 记录目标序列
        Deque<TreeNode> stk = new LinkedList<TreeNode>();// 显式模拟栈
       // 判断结点是否被访问完
       // 处理二叉树为空的特殊情况rrer
        while (root != null || !stk.isEmpty()) {
            // 根据中序遍历顺序，第一个结点是一棵树的最左边的结点
            while (root != null) {
                stk.push(root);
                root = root.left;//
            }
            root = stk.pop();  // 出栈
            res.add(root.val); // 更新目标序列
            root = root.right; // 根据中序遍历顺序，根节点之后应该遍历右结点
        }
        return res;
    }
}
```

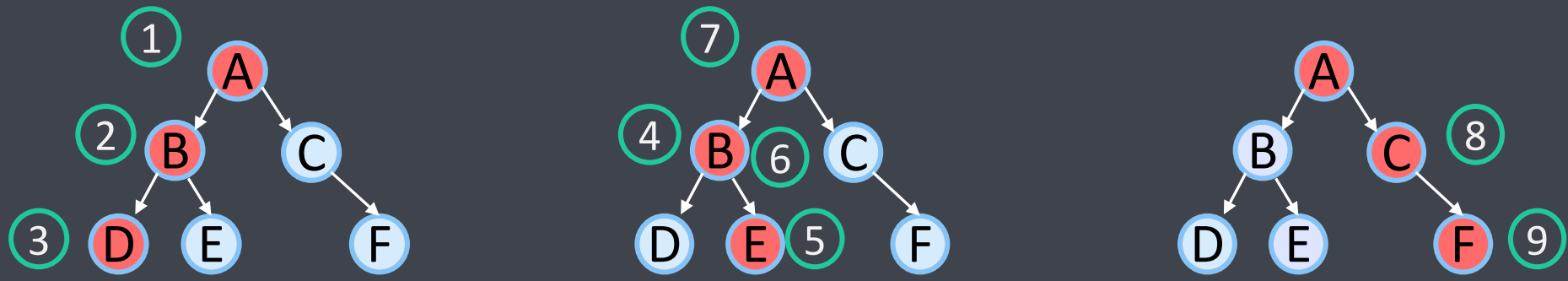
执行结果： 通过 显示详情 >

执行用时： 1 ms ，在所有 Java 提交中击败了 46.27% 的用户

内存消耗： 37 MB ，在所有 Java 提交中击败了 63.31% 的用户

三. Code 基本解法及编码实现

解法二： DFS(递归)



DFS(递归)访问各个结点的顺序

思考：如何在递归的访问的节点过程中得到目标序列[D,B,E,A,C,F] ？

三. Code 基本解法及编码实现

解法二： DFS(递归)边界和细节问题

边界问题

- 考虑树为空情况

细节问题

- 如何判断遍历的结点已经是叶子结点
- 何时访问节点才能得到目标序列

三. Code 基本解法及编码实现

解法二： DFS(递归)复杂度分析

时间复杂度：O(n)

- n是二叉树节点数目
- 每个结点会被遍历一次

空间复杂度：O(h)

- h是二叉树的高度
- 递归需要栈，而栈的深度取决于二叉树的高度

三. Code 基本解法及编码实现

解法二： DFS(递归)编码实现


```
class Solution {
     public List<Integer> inorderTraversal(TreeNode root) {
         List<Integer> res = new ArrayList<Integer>();// 记录目标序列
         inorder(root, res); // 递归遍历二叉树
         return res;
     }

     public void inorder(TreeNode root, List<Integer> res) {
         // 处理二叉树为空的边界问题
         // 判断遍历指针已经到达叶子结点下一层，可以返回
         if (root == null) {return; }

         inorder(root.left, res); // 先遍历左子树结点
         res.add(root.val);// 访问根结点
         inorder(root.right, res); // 再遍历右子树结点
     }
}
```

执行结果： 通过 显示详情 >

执行用时： 0 ms ，在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 37.3 MB ，在所有 Java 提交中击败了 12.73% 的用户

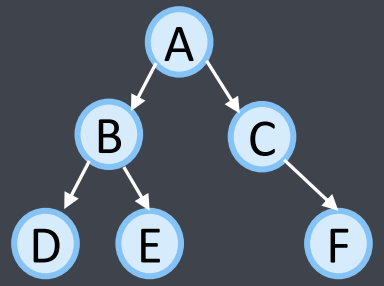
思考：前序遍历、后序遍历的递归代码实现？

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

中序遍历要求每个结点被遍历一次，最优时间复杂度为O(n)

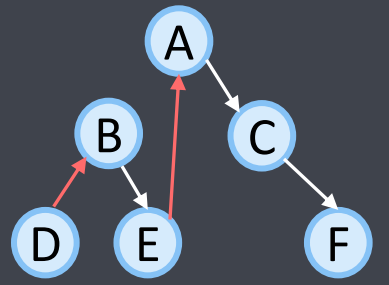
递归和迭代都需要记录中间状态，思考是否可以不用记录中间状态从而降低空间复杂度?



二叉树

D->B->E->A->C->F

目标序列



注意观察需要改变指向的节点的共同特点

- E是A的左子树的中序遍历最后一个点
- D是B的中序遍历最后一个点

五. Code 最优解思路及编码实现

最优解： 莫里斯遍历

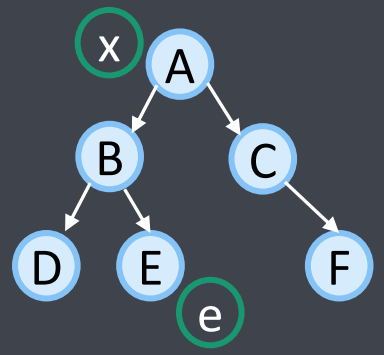
假设当前遍历至结点x：

1. 如果x没有左子树，将其加入目标序列，再遍历右子树，x=x.right
2. 如果x有左子树，找到左子树上中序遍历最后一个结点（记作ex-point）
    - 如果ex-point的右子树为空，则将其指向x，访问x的左子树，x=x.left
    - 如果ex-point右子树不为空，此时其右子树已经指向x，说明左子树已经遍历完成，将ex-point右子树置空，将x加入目标序列，然后访问x的右子树,x=x.right

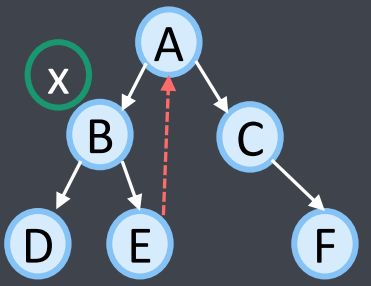
五. Code 最优解思路及编码实现

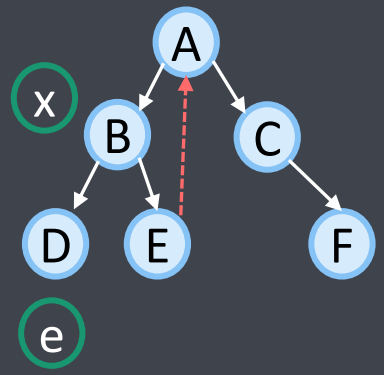
拉勾教育
—互联网人实战大学—

最优解： 莫里斯遍历

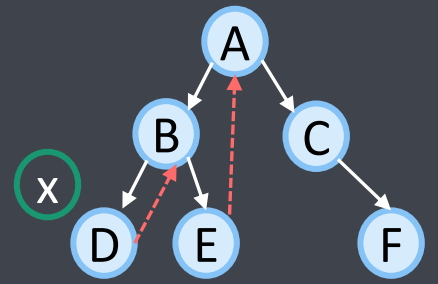


- x无左子树
  - 加入目标序列，x=x.right
- x有左子树，找到ex-point
  - ex-point.right = null, ex-point.right = x, x = x.left
  - ex-point.right!=null, visit x, x=x.right, ex-point.right=null





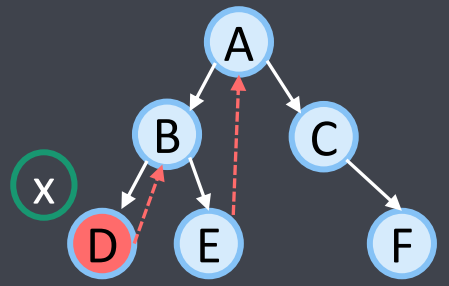
- x无左子树
  - 加入目标序列，x=x.right
- x有左子树，找到ex-point
  - if ex-point.right = null, ex-point.right = x, x = x.left
  - ex-point.right!=null, visit x, x=x.right, ex-point.right=null



五. Code 最优解思路及编码实现

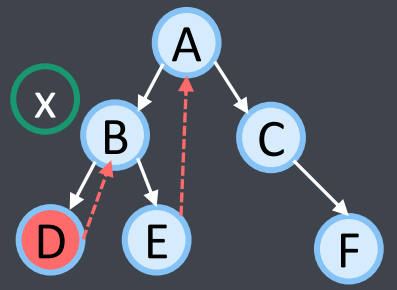
拉勾教育
—互联网人实战大学—

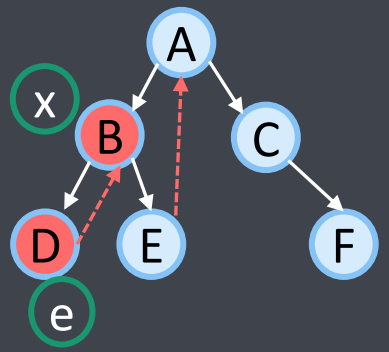
最优解： 莫里斯遍历



- X无左子树
  - 加入目标序列，x=x.right
- X有左子树，找到ex-point
  - ex-point.right = null, ex-point.right = x, x = x.left
  - ex-point.right!=null, visit x, x=x.right, ex-point.right=null

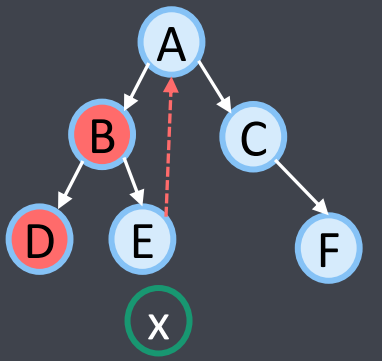
res=[D]





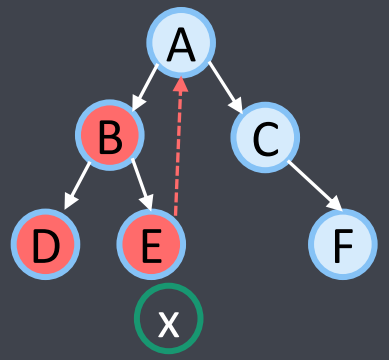
- X无左子树
  - 加入目标序列，x=x.right
- X有左子树，找到ex-point
  - ex-point.right = null, ex-point.right = x, x = x.left
  - ex-point.right!=null, visit x, x=x.right, ex-point.right=null

res=[D,B]



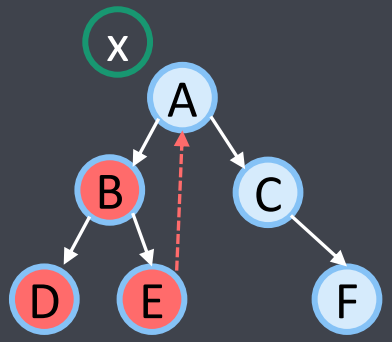
五. Code 最优解思路及编码实现

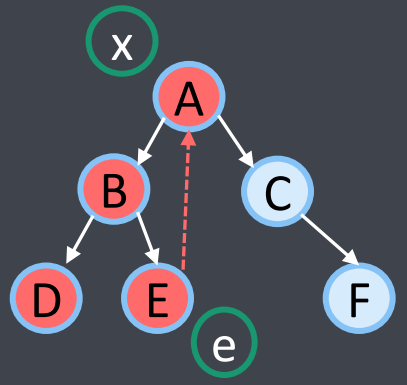
最优解： 莫里斯遍历



- x无左子树
  - 加入目标序列，x=x.right
- x有左子树，找到ex-point
  - ex-point.right = null, ex-point.right = x, x = x.left
  - ex-point.right!=null, visit x, x=x.right, ex-point.right=null

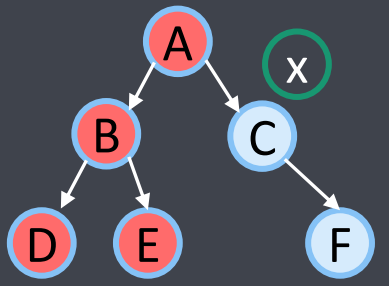
res=[D,B,E]





- x无左子树
  - 加入目标序列，x=x.right
- x有左子树，找到ex-point
  - ex-point.right = null, ex-point.right = x, x = x.left
  - ex-point.right!=null, visit x, x=x.right, ex-point.right=null

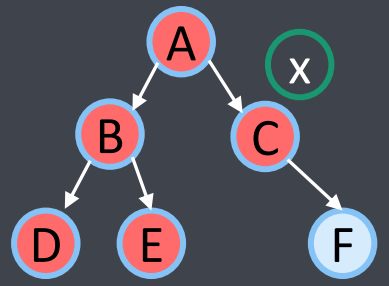
res=[D,B,E,A]



五. Code 最优解思路及编码实现

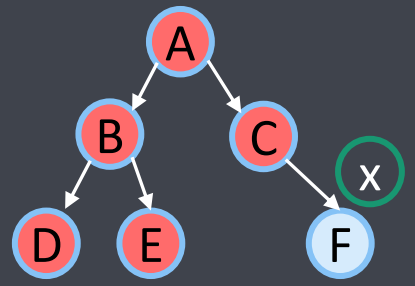
拉勾教育
—互联网人实战大学—

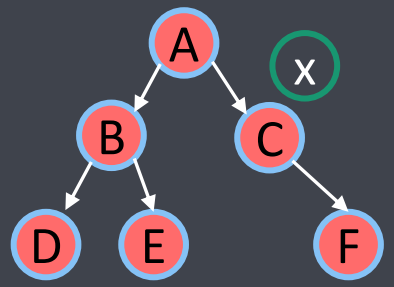
最优解： 莫里斯遍历



- x无左子树
  - 加入目标序列，x=x.right
- x有左子树，找到ex-point
  - ex-point.right = null, ex-point.right = x, x = x.left
  - ex-point.right!=null, visit x, x=x.right, ex-point.right=null

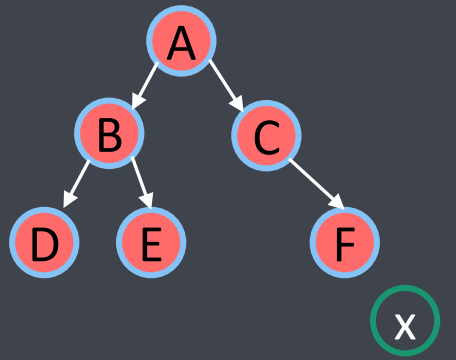
res=[D,B,E,A,C]





- X无左子树
  - 加入目标序列，x=x.right
- X有左子树，找到ex-point
  - ex-point.right = null, ex-point.right = x, x = x.left
  - ex-point.right!=null, visit x, x=x.right, ex-point.right=null

res=[D,B,E,A,E,F]



五. Code 最优解思路及编码实现

复杂度分析

时间复杂度：O(n)

- 每个结点只会被遍历1次

空间复杂度：O(1)

- 无需记录中间结点

五. Code 最优解思路及编码实现

编码实现

```
public List<Integer> inorderTraversal(TreeNode root) {
    List<Integer> res = new ArrayList<Integer>();
    TreeNode exPoint = null;
    while (root != null) {
        if (root.left != null) {
            exPoint = root.left;// exPoint 节点就是当前 root 节点向左走一步，然后一直向右走至无法走为止
            while (exPoint.right != null && exPoint.right != root) {
                exPoint = exPoint.right;
            }
            if (exPoint.right == null) {
                exPoint.right = root; // 让 exPoint 的右指针指向 root，继续遍历左子树
                root = root.left;
            } else { // 说明左子树已经访问完了，我们需要断开链接
                res.add(root.val);
                exPoint.right = null;
                root = root.right;
            }
        } else { // 如果没有左孩子，则直接访问右孩子
            res.add(root.val);
            root = root.right;
        }
    }
    return res;
}
```

执行结果： 通过 显示详情 >

执行用时： 0 ms ，在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 37 MB ，在所有 Java 提交中击败了 56.23% 的用户

六. Change 变形延伸

拉勾教育
—互联网人实战大学—

题目变形

- （练习）二叉树的前序遍历
- （练习）二叉树的后序遍历

延伸扩展

- 在面对一些树的题目的时候，可以考虑改变树本身的结构来解决求解目标

本题来源

- Leetcode 94 https://leetcode-cn.com/problems/binary-tree-inorder-traversal/

总结

- 掌握二叉树的基本分类
- 了解二叉树的前序、后序遍历的方法
- 掌握二叉树的中序遍历方法

课后练习

拉勾教育
—互联网人实战大学—

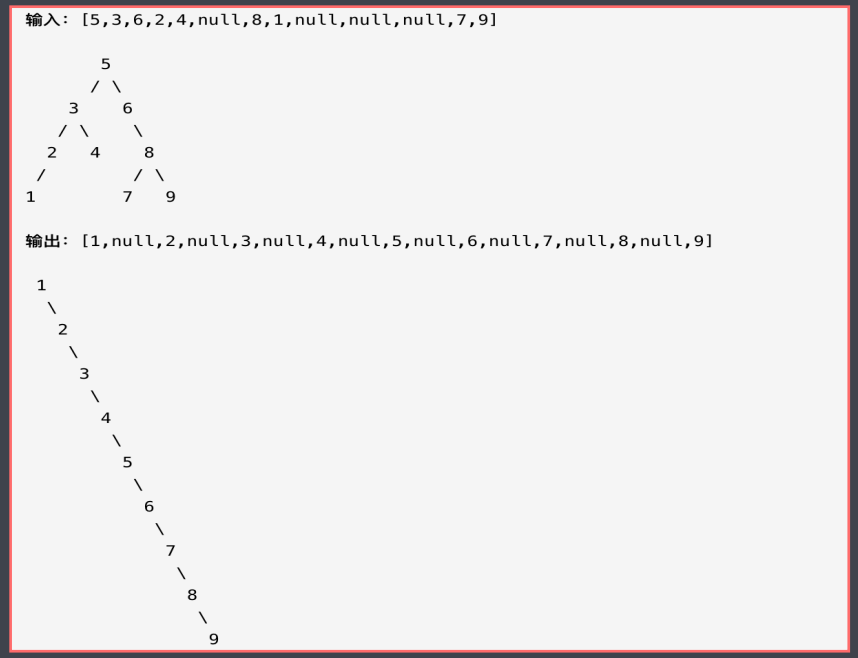
1. 递增顺序查找树(Leetcode 897/简单)
2. 路径总和（Leetcode 112 /简单）
3. 路径总和||（Leetcode 113 /中等）
4. 序列化二叉树（剑指offer 37/困难）

课后练习

拉勾教育
—互联网人实战大学—

1. 递增顺序查找树(Leetcode 897/简单)

   提示：给你一个树，请你 按中序遍历 重新排列树，使树中最左边的结点现在是树的根，并且每个结点没有左子结点，只有一个右子结点。



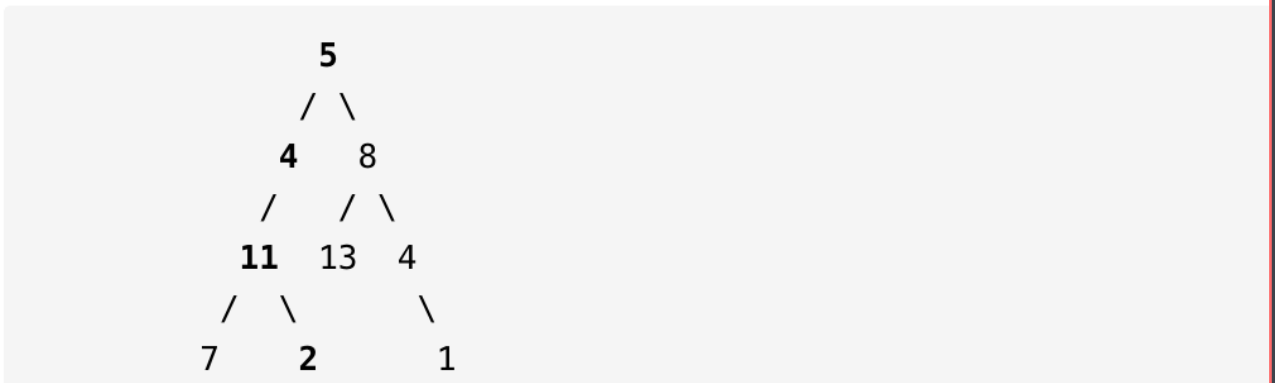
课后练习

拉勾教育
—互联网人实战大学—

2. 路径总和（Leetcode 112 /简单）

提示：给定一个二叉树和一个目标和，判断该树中是否存在根节点到叶子节点的路径，这条路径上所有节点值相加等于目标和。

示例:
给定如下二叉树，以及目标和 `sum = 22`，



返回 `true`，因为存在目标和为 22 的根节点到叶子节点的路径 `5->4->11->2`。

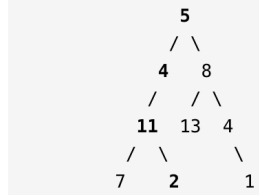
课后练习

拉勾教育
—互联网人实战大学—

3. 路径总和II（Leetcode 113 /中等）

提示：给定一个二叉树和一个目标和，找到所有从根节点到叶子节点路径总和等于给定目标和的路径。

示例:
给定如下二叉树，以及目标和 `sum = 22`，



返回 `true`，因为存在目标和为 22 的根节点到叶子节点的路径 `5->4->11->2`。

返回:

```
[
   [5,4,11,2],
   [5,8,4,5]
]
```

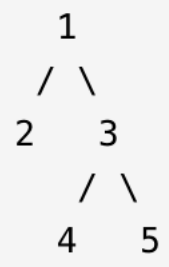
课后练习

拉勾教育
—互联网人实战大学—

4. 序列化二叉树（剑指offer 37/困难）

提示：请实现两个函数，分别用来序列化和反序列化二叉树。

你可以将以下二叉树：



序列化为 "[1,2,3,null,null,4,5]"

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容