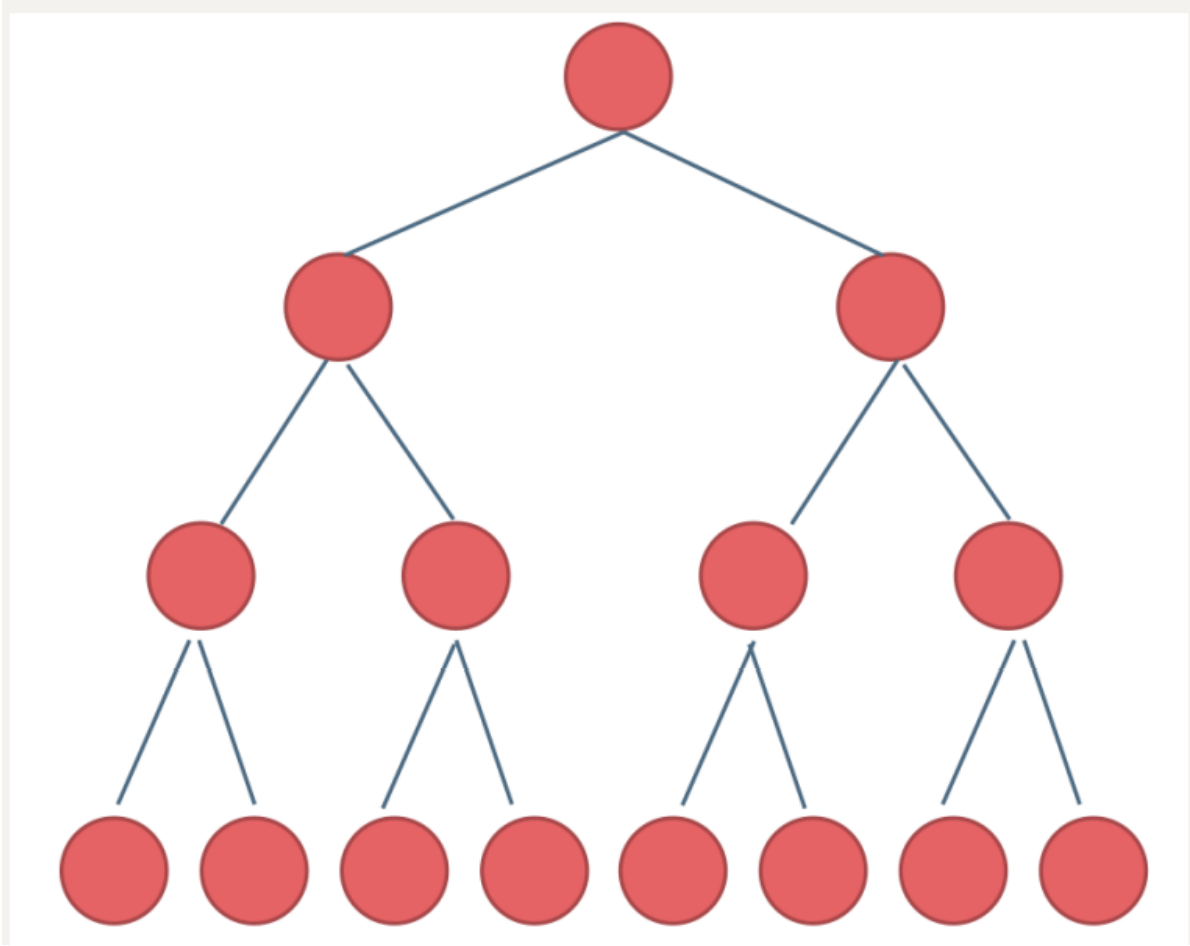
算法-二叉树Part1

开始二刷代码随想录

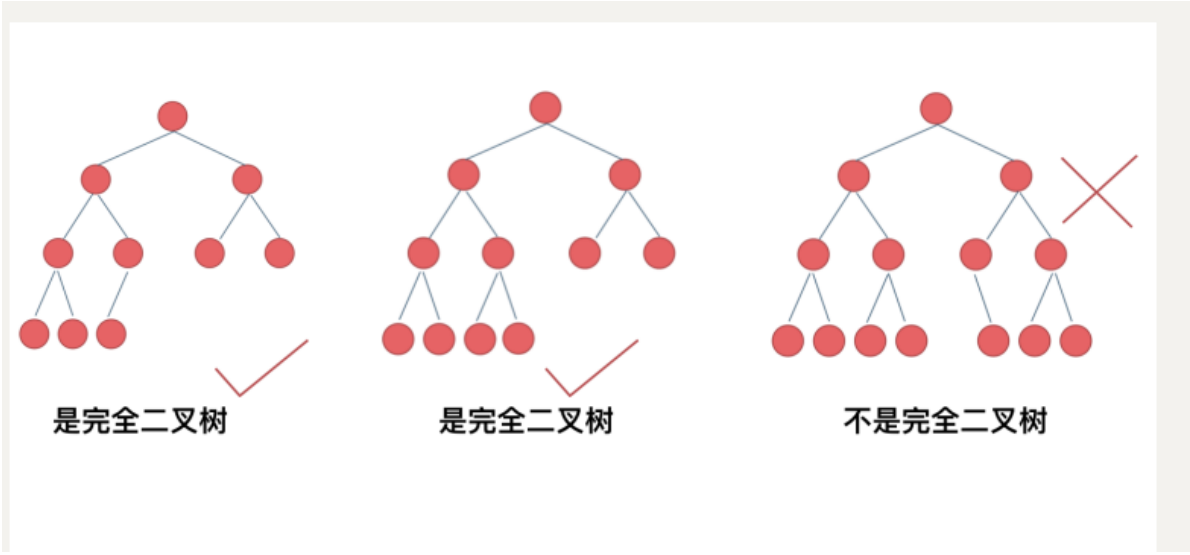
第一天-二叉树基本知识

术语分类

满二叉树：如果一棵二叉树只有度为0的结点和度为2的结点，并且度为0的结点在同一层 上，则这棵二叉树为满二叉树。

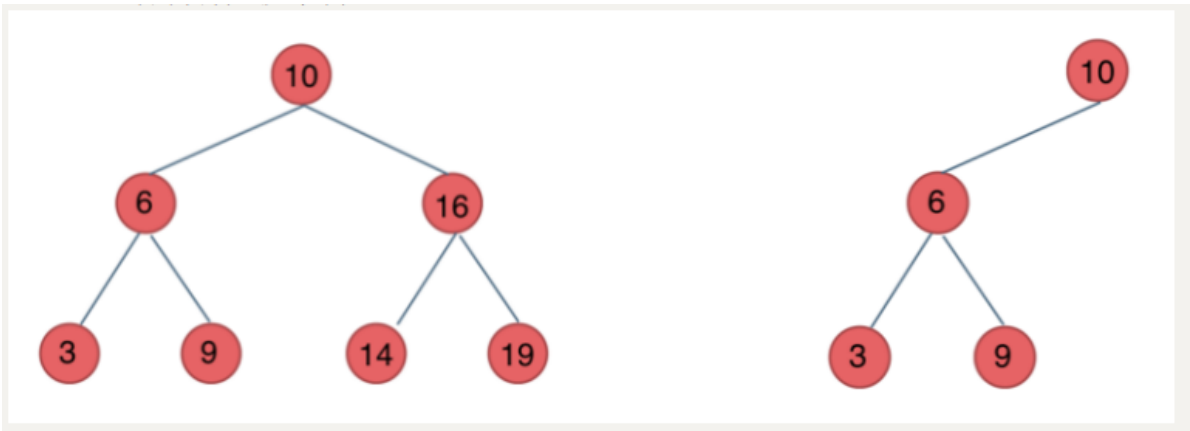


完全二叉树：在完全二叉树中，除了最底层节点可能没填满外，其余每层节点数 都达到最大值，并且最下面一层的节点都集中在该层最左边的若干位置。若最底层为第 h 层，则该层包含 1~ 2^h -1 个节点

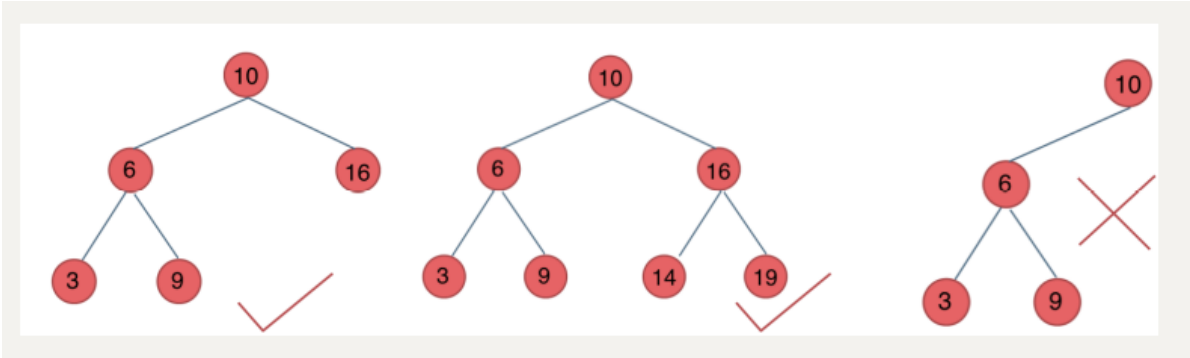


二叉搜索树：二叉搜索树是一个有序树

- 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值；
- 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值；
- 它的左、右子树也分别为二叉排序树



平衡二叉搜索树：又被称为AVL（Adelson-Velsky and Landis）树，且具有以下性质：它是 一棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡 二叉树。



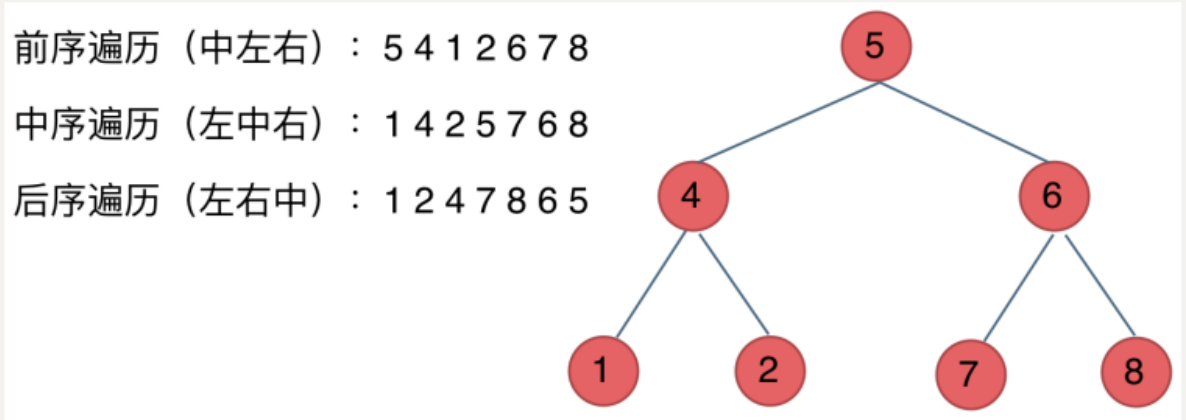
重要

C++中map、set、multimap，multiset的底层实现都是平衡二叉搜索树，所以map、set 的增删操作时间时间复杂度是logn，注意我这里没有说unordered_map、unordered_set， unordered_map、unordered_map底层实现是哈希表。

遍历

二叉树主要有两种遍历方式：

1. 深度优先遍历：先往深走，遇到叶子节点再往回走。
2. 广度优先遍历：一层一层的去遍历



栈与队列的另一个应用场景

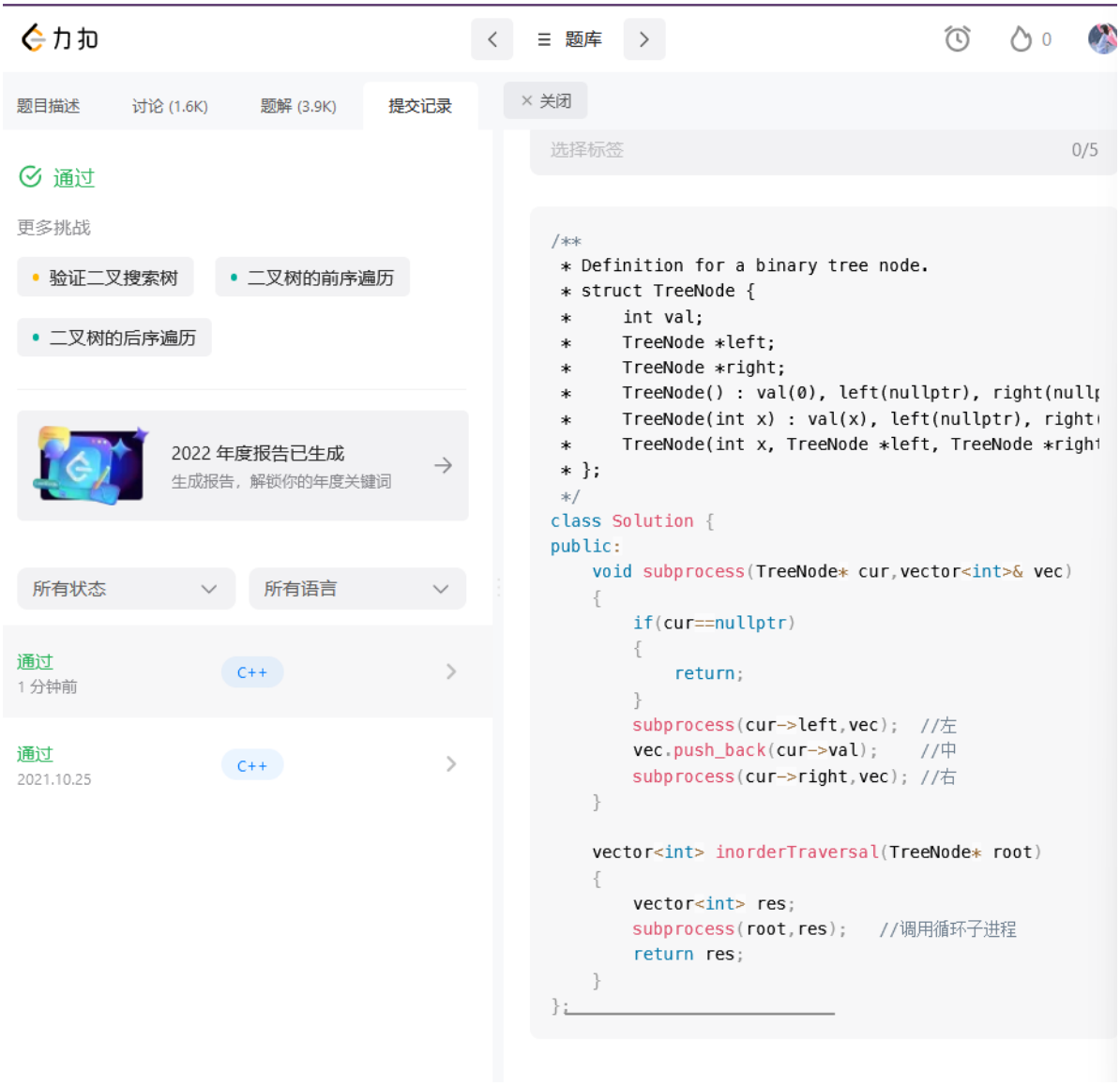
- 栈其实就是递归的一种是实现结构，也就说前中后序遍 历的逻辑其实都是可以借助栈使用非递归的方式来实现的
- 而广度优先遍历的实现一般使用队列来实现，这也是队列先进先出的特点所决定的，因为需 要先进先出的结构，才能一层一层的来遍历二叉树

94-中序遍历

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
```

```
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    void subprocess(TreeNode* cur,vector<int>& vec)
    {
        if(cur==nullptr)
        {
            return;
        }
        subprocess(cur->left,vec);  //左
        vec.push_back(cur->val);    //中
        subprocess(cur->right,vec); //右
    }

    vector<int> inorderTraversal(TreeNode* root)
    {
        vector<int> res;
        subprocess(root,res);   //调用循环子进程
        return res;
    }
};
```




```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root)
```

```
    {
        //中序 左 中 右
        vector<int> res;
        stack<TreeNode*> st;
        TreeNode* cur=root;
        while(cur!= nullptr || !st.empty())
        {
            if(cur != nullptr)
            {
                //指针访问结点 访问到最下层
                st.push(cur);
                cur = cur->left;
            }
            else
            {
                cur=st.top();
                st.pop();
                res.push_back(cur->val);    //中
                cur=cur->right;
            }
        }
        return res;
    }
};
```

144-前序

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
```

```
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    void subprocess(TreeNode* cur,vector<int>& vec)
    {
        if(cur==nullptr)
        {
            return;
        }
        vec.push_back(cur->val);
        subprocess(cur->left,vec);
        subprocess(cur->right,vec);
    }

    vector<int> preorderTraversal(TreeNode* root)
    {
        vector<int> res;
        subprocess(root,res);
        return res;
    }
};
```

```
class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root)
    {
        //中 左 右
        //非递归方式 使用栈来实现
        stack<TreeNode*> st1;
        vector<int> res;
        if(root == nullptr)
        {
            return res;
        }

        st1.push(root);
        while(!st1.empty())
```

```
        {
            TreeNode* node = st1.top();  //中间结点
            st1.pop();   //弹出
            res.push_back(node->val);
            if(node->right)
            {
                st1.push(node->right);   //右   因为先进后出的特性  所以先压右结点入栈
            }
            if(node->left)
            {
                st1.push(node->left);   //左
            }
        }
        return res;
    }
};
```

145-后序

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    void subprocess(TreeNode* cur,vector<int>& vec)
    {
```

```
        if(cur==nullptr)
        {
            return;
        }
        subprocess(cur->left,vec);
        subprocess(cur->right,vec);
        vec.push_back(cur->val);
    }

    vector<int> postorderTraversal(TreeNode* root)
    {
        vector<int> res;
        subprocess(root,res);
        return res;
    }
};
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<int> postorderTraversal(TreeNode* root)
    {
        stack<TreeNode*> st;
        vector<int> res;
        if(root == nullptr)
        {
            return res;
```

```
        }
        st.push(root);
        while(!st.empty())
        {
            TreeNode* node = st.top();
            st.pop();
            res.push_back(node->val);
            if(node->left)
            {
                st.push(node->left);
            }
            if(node->right)
            {
                st.push(node->right);
            }
        }
        reverse(res.begin(),res.end());
        return res;
    }
};
```