

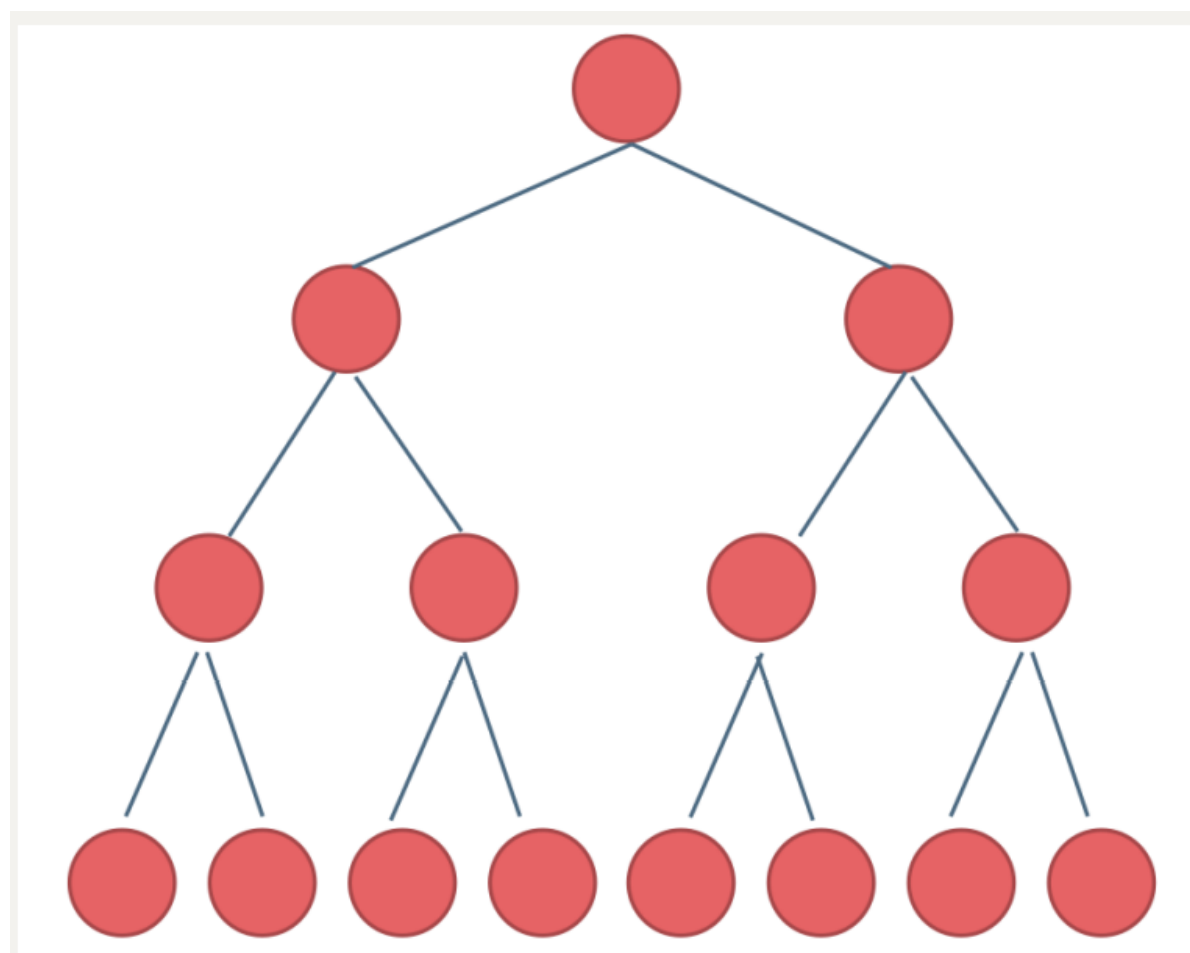
算法-二叉树Part1

开始二刷代码随想录

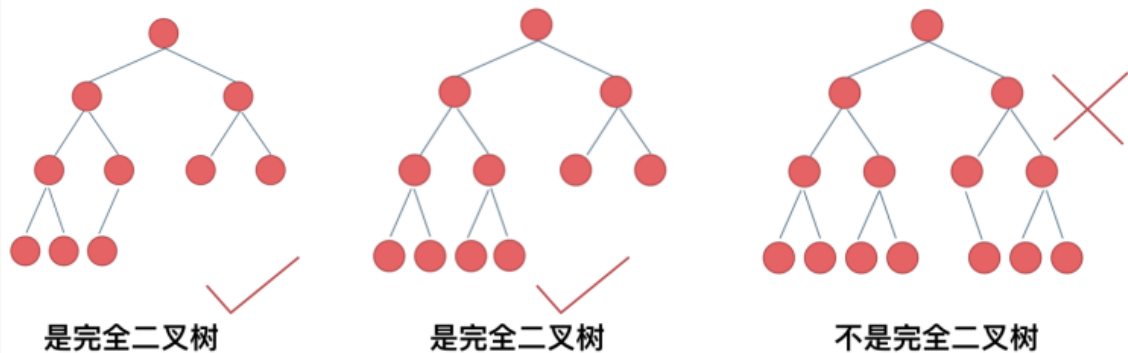
第一天-二叉树基本知识

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。

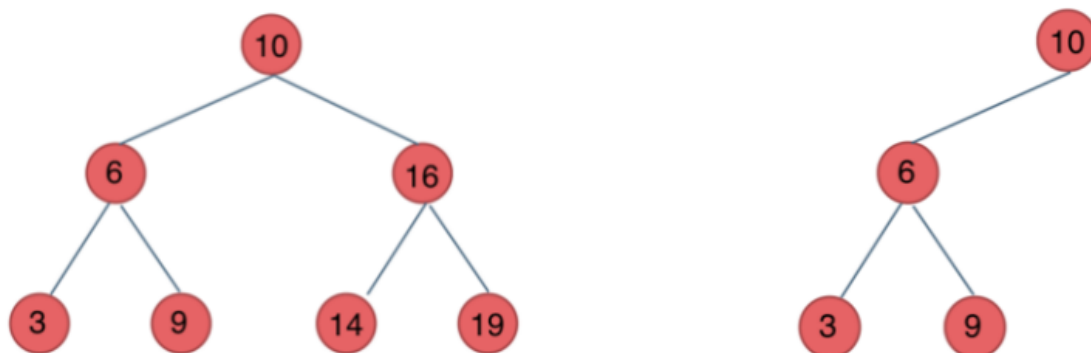


完全二叉树：在完全二叉树中，除了最底层节点可能没填满外，其余每层节点数都达到最大值，并且最下面一层的节点都集中在该层最左边的若干位置。若最底层为第 h 层，则该层包含 $1 \sim 2^{h-1}$ 个节点

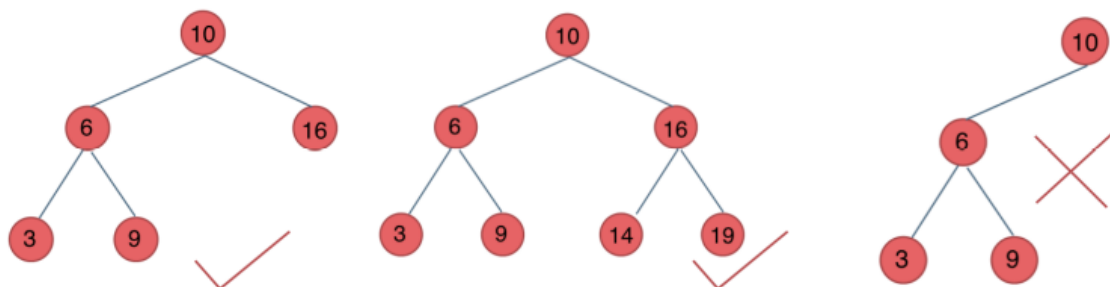


二叉搜索树：二叉搜索树是一个有序树

- 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，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 的增删操作时间复杂度是 $\log n$ ，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

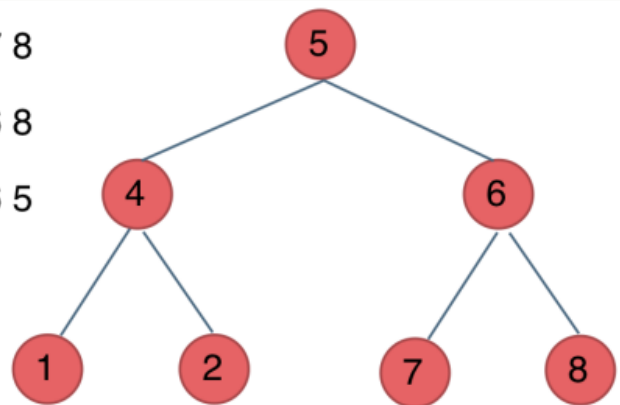
二叉树主要有两种遍历方式：

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

前序遍历（中左右）：5 4 1 2 6 7 8

中序遍历（左中右）：1 4 2 5 7 6 8

后序遍历（左右中）：1 2 4 7 8 6 5



栈与队列的另一个应用场景

- 栈其实就是递归的一种实现结构，也就是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

```
1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
```

```
7      *      TreeNode() : val(0), left(nullptr),
      right(nullptr) {}
8      *      TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
9      *      TreeNode(int x, TreeNode *left, TreeNode *right)
      : val(x), left(left), right(right) {}
10     * };
11     */
12     class Solution {
13     public:
14         void subprocess(TreeNode* cur,vector<int>& vec)
15         {
16             if(cur==nullptr)
17             {
18                 return;
19             }
20             subprocess(cur->left,vec);    //左
21             vec.push_back(cur->val);      //中
22             subprocess(cur->right,vec);    //右
23         }
24
25         vector<int> inorderTraversal(TreeNode* root)
26         {
27             vector<int> res;
28             subprocess(root,res);    //调用循环子进程
29             return res;
30         }
31     };
```

题目描述

讨论 (1.6K)

题解 (3.9K)

提交记录

× 关闭

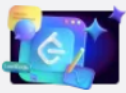
✔ 通过

更多挑战

• 验证二叉搜索树

• 二叉树的前序遍历

• 二叉树的后序遍历



2022 年度报告已生成
生成报告, 解锁你的年度关键词



所有状态



所有语言



通过

1 分钟前

C++



通过

2021.10.25

C++



选择标签

0/5

```
/**
 * 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;
    }
};
```

```
1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode() : val(0), left(nullptr),
8   *     right(nullptr) {}
9   *     TreeNode(int x) : val(x), left(nullptr),
10  *     right(nullptr) {}
11  *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
12  * };
13  */
14 class solution {
15 public:
16     vector<int> inorderTraversal(TreeNode* root)
```

```

15     {
16         //中序 左 中 右
17         vector<int> res;
18         stack<TreeNode*> st;
19         TreeNode* cur=root;
20         while(cur!= nullptr || !st.empty())
21         {
22             if(cur != nullptr)
23             {
24                 //指针访问结点 访问到最下层
25                 st.push(cur);
26                 cur = cur->left;
27             }
28             else
29             {
30                 cur=st.top();
31                 st.pop();
32                 res.push_back(cur->val);    //中
33                 cur=cur->right;
34             }
35         }
36         return res;
37     }
38 };

```

144-前序

```

1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode() : val(0), left(nullptr),
8   *     right(nullptr) {}
9   *     TreeNode(int x) : val(x), left(nullptr),
10    *     right(nullptr) {}

```

```

9      *      TreeNode(int x, TreeNode *left, TreeNode *right)
      : val(x), left(left), right(right) {}
10     * };
11     */
12     class solution {
13     public:
14         void subprocess(TreeNode* cur,vector<int>& vec)
15         {
16             if(cur==nullptr)
17             {
18                 return;
19             }
20             vec.push_back(cur->val);
21             subprocess(cur->left,vec);
22             subprocess(cur->right,vec);
23         }
24
25         vector<int> preorderTraversal(TreeNode* root)
26         {
27             vector<int> res;
28             subprocess(root,res);
29             return res;
30         }
31     };

```

```

1     class solution {
2     public:
3         vector<int> preorderTraversal(TreeNode* root)
4         {
5             //中 左 右
6             //非递归方式 使用栈来实现
7             stack<TreeNode*> st1;
8             vector<int> res;
9             if(root == nullptr)
10            {
11                return res;
12            }
13
14            st1.push(root);
15            while(!st1.empty())

```

```

16         {
17             TreeNode* node = st1.top(); //中间结点
18             st1.pop(); //弹出
19             res.push_back(node->val);
20             if(node->right)
21             {
22                 st1.push(node->right); //右 因为先进后
出的特性 所以先压右结点入栈
23             }
24             if(node->left)
25             {
26                 st1.push(node->left); //左
27             }
28         }
29         return res;
30     }
31 };

```

145-后序

```

1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode() : val(0), left(nullptr),
right(nullptr) {}
8   *     TreeNode(int x) : val(x), left(nullptr),
right(nullptr) {}
9   *     TreeNode(int x, TreeNode *left, TreeNode *right)
: val(x), left(left), right(right) {}
10  * };
11  */
12  class Solution {
13  public:
14      void subprocess(TreeNode* cur,vector<int>& vec)
15      {

```



```

16         if(cur==nullptr)
17         {
18             return;
19         }
20         subprocess(cur->left,vec);
21         subprocess(cur->right,vec);
22         vec.push_back(cur->val);
23     }
24
25     vector<int> postorderTraversal(TreeNode* root)
26     {
27         vector<int> res;
28         subprocess(root,res);
29         return res;
30     }
31 };

```

```

1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode() : val(0), left(nullptr),
right(nullptr) {}
8   *     TreeNode(int x) : val(x), left(nullptr),
right(nullptr) {}
9   *     TreeNode(int x, TreeNode *left, TreeNode *right)
: val(x), left(left), right(right) {}
10  * };
11  */
12  class Solution {
13  public:
14      vector<int> postorderTraversal(TreeNode* root)
15      {
16          stack<TreeNode*> st;
17          vector<int> res;
18          if(root == nullptr)
19          {
20              return res;

```

```
21     }
22     st.push(root);
23     while(!st.empty())
24     {
25         TreeNode* node = st.top();
26         st.pop();
27         res.push_back(node->val);
28         if(node->left)
29         {
30             st.push(node->left);
31         }
32         if(node->right)
33         {
34             st.push(node->right);
35         }
36     }
37     reverse(res.begin(), res.end());
38     return res;
39 }
40 };
```