二叉树

二叉树遍历

递归

- 前序遍历
- 中序遍历
- 后序遍历

栈

- 前序遍历
- 中序遍历
- 后序遍历

Morris

- 前序遍历
- 中序遍历
- 后序遍历

102. 二叉树的层序遍历 - BFS

思路

按层走, 先遍历高度为0的, 再遍历高度为1, ..., 正是 bfs 的搜索顺序

步骤:

1. 将本层的结点加入到队列 q 中
2. 对 队列q 中的结点进行遍历, 将值加入到结果中, 同时将 q 中结点的子结点加入到新队列 p 中
3. 循环结束时, q = p, 以遍历下一层.

代码

```
class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        if not root:
            return []
        ans, q = [], [root]
        while q:
```

```
            ans.append(list())
            p = []
            for i in range(len(q)):
                node = q[i]
                ans[-1].append(node.val)
                if node.left:
                    p.append(node.left)
                if node.right:
                    p.append(node.right)
            q = p
        return ans
```

```
func levelOrder(root *TreeNode) [][]int {
    q := []*TreeNode{root}
    res := [][]int{}
    if root == nil {
        return res
    }
    for i := 0; len(q) > 0; i++ {
        res = append(res, []int{})
        p := []*TreeNode{}
        for j := 0; j < len(q); j++ {
            node := q[j]
            res[i] = append(res[i], node.Val)
            if node.Left != nil {
                p = append(p, node.Left)
            }
            if node.Right != nil {
                p = append(p, node.Right)
            }
        }
        q = p
    }
    return res
}
```

```
# 队列排序
class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        ans = []
        l = list()
        def bfs(root, i):
            if not root:
                return
            l.append((i, root.val))
            bfs(root.left, i+1)
            bfs(root.right, i+1)
```

```
        bfs(root, 0)
        pre = -1
        l.sort(key=lambda x: x[0])
        for x in l:
            if x[0] != pre:
                ans.append(list())
                pre = x[0]
            ans[-1].append(x[1])
        return ans
```