二叉树的层序遍历 （BFS）

核心思想，遍历到每一层，

每一个节点的孩子用队列记录，先进先出

然后把每一层的下一层节点数量记录下来

每push到队列一次，节点数量就 -1

如果下一层节点数量归零，就意味着这层遍历结束，

进入到下一层

In [ ]:
```python
# 长度法

from typing import List, Optional
from collections import deque

# 定义二叉树的节点类
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        # 如果树是空的，直接返回空列表
        if not root:
            return []

        # 初始化队列，用于存储每一层的节点
        queue = deque([root])
        # 用于存储最终的层次遍历结果
        result = []

        # 当队列不为空时，说明还有未遍历的节点
        while queue:
            # 当前层的节点值
            current_level = []

            # 遍历当前层的所有节点
            for _ in range(len(queue)):  # len(queue) 是当前层的节点数
                # 取出队首节点
                node = queue.popleft()
                # 将当前节点的值添加到当前层
                current_level.append(node.val)

                # 如果有左子节点，加入队列
                if node.left:
                    queue.append(node.left)
                # 如果有右子节点，加入队列
                if node.right:
                    queue.append(node.right)

            # 当前层结束后，将结果添加到最终结果中
            result.append(current_level)

        return result
;
```

In [ ]:
```python
class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        if not root:
            return []

        levels = []

        def traverse(node, level):
            if not node:
                return

            if len(levels) == level:
                levels.append([])

            levels[level].append(node.val)
            traverse(node.left, level + 1)
            traverse(node.right, level + 1)

        traverse(root, 0)
        return levels
```

层序遍历（反转方向）：最后反转一下结果数列 [::-1]

In [ ]:
```python
# while记录每层大小法
```

```
from collections import deque

class Solution:
    def levelOrderBottom(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        queue = deque([root])
        result = []

        while queue:
            level = []
            for _ in range(len(queue)):
                cur = queue.popleft()
                if cur.left:
                    level.append(cur.left)
                if cur.right:
                    level.append(cur.right)

            result.append(level)
        return result[::-1]
```

层序遍历 (右视图)：每层判断一下是不是最后一个

In [ ]:
```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:
        if not root:
            return []

        queue = deque([root])
        result = []

        while queue:
            level_size = len(queue)
            for i in range(level_size):
                node = queue.popleft()
                if i == level_size - 1:
                    result.append(node.val)

                # 注意要加这个，如果不加，之后queue就空了
                # 注意这个循环在for loop内部
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)

        return result
```

637.二叉树的层平均值

In [1]:
```
from collections import deque

class Solution:
    def averageOfLevels(self, root: TreeNode) -> List[List[int]]:

        if not root:
            return []
        queue = deque([root])
        result = []

        while queue:
            level = []
            for _ in range(len(queue)):
                node = queue.popleft()
                level.append(node.val)
                # 注意要加这个，如果不加，之后queue就空了
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)

            result.append(sum(level)/len(level))

        return result
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 3
      1 from collections import deque
----> 3 class Solution:
      4     def averageOfLevels(self, root: TreeNode) -> List[List[int]]:
      6         if not root:

Cell In[1], line 4, in Solution()
      3 class Solution:
----> 4     def averageOfLevels(self, root: TreeNode) -> List[List[int]]:
      6         if not root:
      7             return []

NameError: name 'TreeNode' is not defined
```

429.N叉树的层序遍历

In [ ]:
```python
class Solution:
    def levelOrder(self, root: 'Node') -> List[List[int]]:
        if not root:
            return []

        result = []
        queue = collections.deque([root])

        while queue:
            level_size = len(queue)
            level = []

            for _ in range(level_size):
                node = queue.popleft()
                level.append(node.val)

                # 就是这里有不同，遍历了所有child，而不是左右了
                for child in node.children:
                    queue.append(child)

            result.append(level)

        return result
```

```
Running cells with 'Python 3.13.1' requires the ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'c:/Python313/python.exe -m pip install ipykernel -U --user --force-reinstall'
```

In [ ]:
```python
# LeetCode 429. N-ary Tree Level Order Traversal
# 递归法
class Solution:
    def levelOrder(self, root: 'Node') -> List[List[int]]:
        if not root: return []
        result=[]
        def traversal(root,depth):
            if len(result)==depth:result.append([])
            result[depth].append(root.val)
            if root.children:
                for i in range(len(root.children)):
                    traversal(root.children[i],depth+1)

        traversal(root,0)
        return result
```

515.在每个树行中找最大值

就是每行记录一下最大值

In [ ]:
```python
# Definition for a binary tree node.

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

from collections import deque
from typing import Optional, List

class Solution:
    def largestValues(self, root: Optional[TreeNode]) -> List[int]:

        if not root:
            return []
        queue = deque([root])
        result = []

        while queue:
            level = []
            for _ in range(len(queue)):
```

```
                node = queue.popleft()
                level.append(node.val)
                # 注意要加这个，如果不加，之后queue就空了
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)

            result.append(max(level))

        return result
```

116.填充每个节点的下一个右侧节点指针

117.填充每个节点的下一个右侧节点指针II

In [12]:
```
#  第一次尝试


"""
# Definition for a Node.
class Node:
    def __init__(self, val: int = 0, left: 'Node' = None, right: 'Node' = None, next: 'Node' = None):
        self.val = val
        self.left = left
        self.right = right
        self.next = next
"""

class Solution:
    def connect(self, root: 'Optional[Node]') -> 'Optional[Node]':
        if not root:
            return root
        queue = deque([root])

        while queue:
            level = [];
            queue_len = len(queue)
            for i in range(queue_len):
                node = queue.popleft()
                level.append(node)

                if len(level) > 1:
                    level[-2].next = node

                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)

            level[-1].next = None

        return root
```

In [ ]:
```
# 更简洁的逻辑

class Solution:
    def connect(self, root: 'Node') -> 'Node':
        if not root:
            return root

        queue = collections.deque([root])

        while queue:
            level_size = len(queue)
            prev = None

            for i in range(level_size):
                node = queue.popleft()

                if prev:
                    prev.next = node

                prev = node

                if node.left:
                    queue.append(node.left)

                if node.right:
                    queue.append(node.right)

        return root
```

In [ ]:
```
from collections import deque

class Solution:
    def connect(self, root: 'Node') -> 'Node':
        if not root:
            return root
```

```
        queue = collections.deque([root])

        while queue:
            level_size = len(queue)
            prev = None

            for i in range(level_size):
                node = queue.popleft()

                # 这是很重要的一个范式
                if prev:
                    prev.next = node

                prev = node

                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)

        return root
```

In [ ]:

104.二叉树的最大深度

In [ ]:
```
class Solution:
    def connect(self, root: 'Node') -> 'Node':
        if not root:
            return 0

        queue = collections.deque([root])
        level_n = 0
        while queue:
            level_size = len(queue)
            prev = None

            for i in range(level_size):
                node = queue.popleft()

                if node.left:
                    queue.append(node.left)

                if node.right:
                    queue.append(node.right)

            level_n += 1

        return level_n
```

111.二叉树的最小深度

只有当左右孩子都为空的时候，才说明遍历的最低点了

In [ ]:
```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def minDepth(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0

        queue = collections.deque([root])
        level_n = 0
        while queue:
            level_size = len(queue)
            prev = None

            for i in range(level_size):
                node = queue.popleft()

                if node.left:
                    queue.append(node.left)
                    level_n += 1

                elif node.right:
                    queue.append(node.right)
                    level_n += 1

                else:
                    return level_n + 1


        return level_n
```