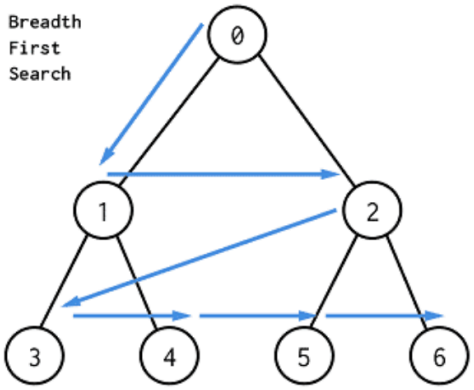
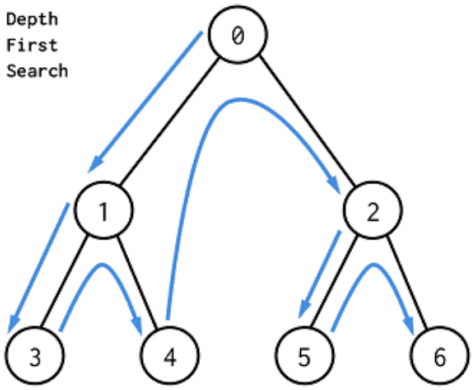




BFS使用队列，把每个还没有搜索到的点依次放入队列，然后再弹出队列的头部元素当做当前遍历点。

BFS总共有两个模板：

如果不需要确定当前遍历到了哪一层，BFS模板如下。

```
while queue 不空：
    cur = queue.pop()
    for 节点 in cur的所有相邻节点：
        if 该节点有效且未访问过：
            queue.push(该节点)
```

如果要确定当前遍历到了哪一层，BFS模板如下。

这里增加了level表示当前遍历到二叉树中的哪一层了，也可以理解为在一个图中，现在已经走了多少步了。size表示在当前遍历层有多少个元素，也就是队列中的元素数，我们把这些元素一次性遍历完，即把当前层的所有元素都向外走了一步。

```
level = 0
while queue 不空：
    size = queue.size()
    while (size --) {
        cur = queue.pop()
        for 节点 in cur的所有相邻节点：
            if 该节点有效且未被访问过：
                queue.push(该节点)
    }
     level ++;
```

102 二叉树的层序遍历：

给你一个二叉树，请你返回其按 **层序遍历** 得到的节点值。 （即逐层地，从左到右访问所有节点）。

示例：
二叉树：`[3,9,20,null,null,15,7]`，

```
    3
   / \
  9  20
    /  \
   15   7
```

返回其层次遍历结果：

```
[
  [3],
  [9,20],
  [15,7]
]
```

```python
def levelOrder(self, root):
        if not root :
            return []
        res = []
        queue = [root]
        while queue:
            # 获取当前队列的长度，这个长度相当于 当前这一层的节点个数
            size = len(queue)
            tmp = []
             # 将队列中的元素都拿出来(也就是获取这一层的节点)，放到临时list中
             for _ in xrange(size):
                 r = queue.pop(0)
                 tmp.append(r.val)
                 # 如果节点的左/右子树不为空，则放入队列中
                 if r.left:
                     queue.append(r.left)
                 if r.right:
                     queue.append(r.right)
             # 将临时list加入最终的结果中
             res.append(tmp)
         return res
```

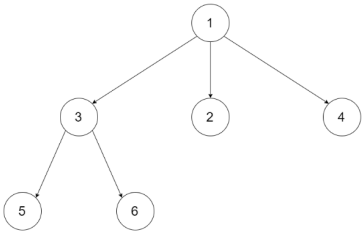
时间复杂度O（n）

空间复杂度是 O（内）

429 N叉树的层序遍历，BFS

给定一个 N 叉树，返回其节点值的*层序遍历*。(即从左到右，逐层遍历)。

例如，给定一个 `3叉树` ：



返回其层序遍历:

```
[
     [1],
     [3,2,4],
     [5,6]
]
```

```python
def levelOrder(self, root):
        """
        :type root: Node
        :rtype: List[List[int]]
        """
        if not root:
            return []
        res = []
        queue = [root]
         while queue:
             # 获取当前队列的长度，这个长度相当于 当前这一层的节点个数
             size = len(queue)
             tmp = []
             for i in range(size):
                 cur = queue.pop(0)
                 tmp.append(cur.val)
                 if cur.children :
                     queue.extend(cur.children)
             res.append(tmp)
         return res
```

DFS 使用的是递归

```python
def levelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
```

```
        """
        if not root :
            return []
        res = []
        def dfs(index, r):
             # res是[[1],[2,3]], index是3，就再插入一个空list到res中
             if len(res) < index :
                 res.append([])
             # 将当前节点值加入到res中，index是当前层，假设index是3，节点值是5，
             # res是[[1],[2,3],[]], 加入后变成[[1],[2,3],[5]]
             res[index -1].append(r.val)
             if r.left:
                 dfs(index+1, r.left)
             if r.right:
                 dfs(index+1, r.right)
         dfs(1,root)
         return res
```

类比 N叉树的层序遍历

```
def levelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root :
            return []
        res = []
        def dfs(index, r):
             # res是[[1],[2,3]], index是3，就再插入一个空list到res中
             if len(res) < index :
                 res.append([])
             # 将当前节点值加入到res中，index是当前层，假设index是3，节点值是5，
             # res是[[1],[2,3],[]], 加入后变成[[1],[2,3],[5]]
             res[index -1].append(r.val)
             if r.left:
                 dfs(index+1, r.left)
             if r.right:
                 dfs(index+1, r.right)
         dfs(1,root)
         return res
```

时间复杂度：O(n)

空间复杂度：O(h)．h树的深度