hw#3_hui_duan

September 29, 2019

0.1 111. Minimum Depth of Binary Tree

```
In [ ]: class Solution:
            def minDepth(self, root: TreeNode) -> int:
                if root == None: return 0
                if root.left == None and root.right != None:
                    return self.minDepth(root.right)+1
                if root.right == None and root.left != None:
                    return self.minDepth(root.left)+1
                return min(self.minDepth(root.right),self.minDepth(root.left) )+1
```

0.2 112. Path Sum

```
In [ ]: class Solution:
            def hasPathSum(self, root: TreeNode, sum: int) -> bool:
                if not root: return False
                if not root.left and not root.right:
                    if sum==root.val: return True
                    else: return False
                else:
                    return self.hasPathSum(root.left, sum-root.val) or self.hasPathSum(root.righ
```

0.3 102. Binary Tree Level Order Traversal

```
In [ ]: class Solution:
            def dfs(self, root, level, res):
                if root == None:
                    return []
                if len(res) < level+1:
                    res.append([])
                res[level].append(root.val)
                self.dfs(root.right, level+1, res)
                self.dfs(root.left, level+1, res)

            def levelOrder(self, root):
                res = []
                self.dfs(root, 0, res)
                return res
```

0.4 841. Keys and Rooms

```
In [ ]: class Solution:
            def dfs(self, rooms, i, visited):
                visited[i] = 1
                for key in rooms[i]:
                    if not visited[key]:
                        self.dfs(rooms, key, visited)

            def canVisitAllRooms(self, rooms):
                visited = [0 for i in range(len(rooms))]
                self.dfs(rooms, 0, visited)
                return len(rooms) == sum(visited)
```

0.5 743. Network Delay Time

```
In [ ]: class Solution:
            def networkDelayTime(self, times, N, K):
                dis = [float('inf')] * N
                dis[K - 1] = 0
                for i in range(N):
                    for time in times:
                        u = time[0] - 1
                        v = time[1] - 1
                        w = time[2]
                        dis[v] = min(dis[v], dis[u] + w)
                if float('inf') in dis:
                    return -1
                else: return max(dis)
```

0.6 301. Remove Invalid Parentheses

```
In [ ]: class Solution(object):
            def checkValid(self, s):
                count = 0
                for item in s:
                    if item == '(': count += 1
                    elif item == ')':
                        if count == 0: return False
                        count -= 1
                return count == 0

            def removeInvalidParentheses(self, s):
                if not s: return []
                stack = [s]
```

```
result = []
vis = set([s])
found = False
while stack:
    cur = stack.pop(0)
    if self.checkValid(cur):
        found = True
        result.append(cur)
    elif not found:
        for i in xrange(len(cur)):
            if cur[i] == '(' or cur[i] == ')':
                t = cur[:i] + cur[i + 1:]
                if t not in vis:
                    stack.append(t)
                    vis.add(t)
return result
```