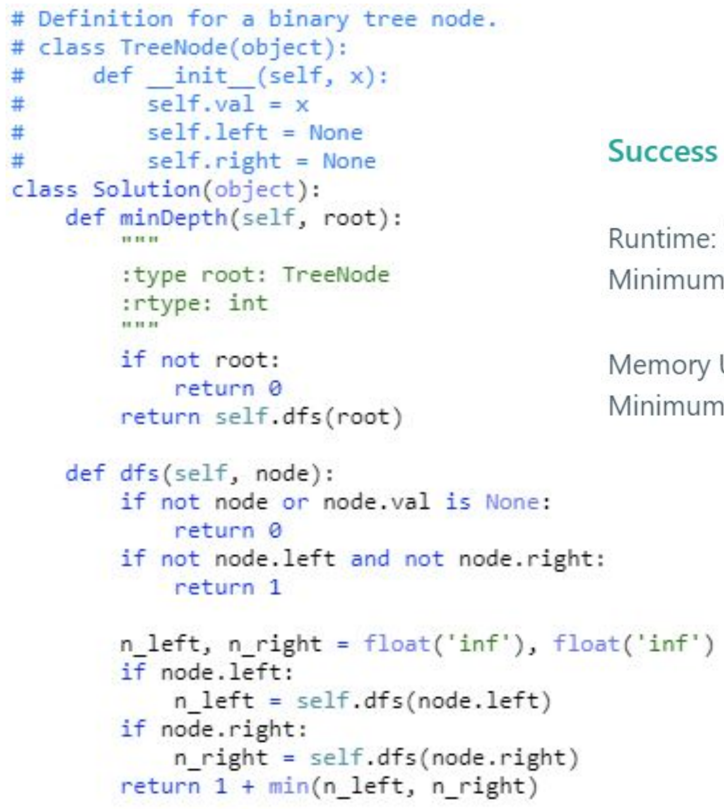
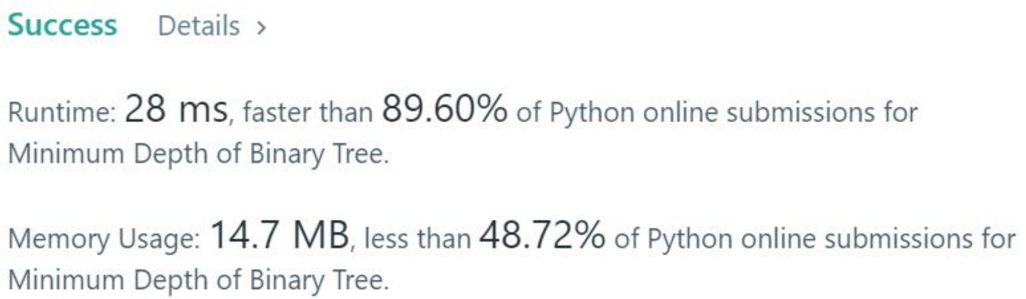
LC 111: Minimum Depth of Binary Tree



```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution(object):
    def minDepth(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if not root:
            return 0
        return self.dfs(root)

    def dfs(self, node):
        if not node or node.val is None:
            return 0
        if not node.left and not node.right:
            return 1

        n_left, n_right = float('inf'), float('inf')
        if node.left:
            n_left = self.dfs(node.left)
        if node.right:
            n_right = self.dfs(node.right)
        return 1 + min(n_left, n_right)
```

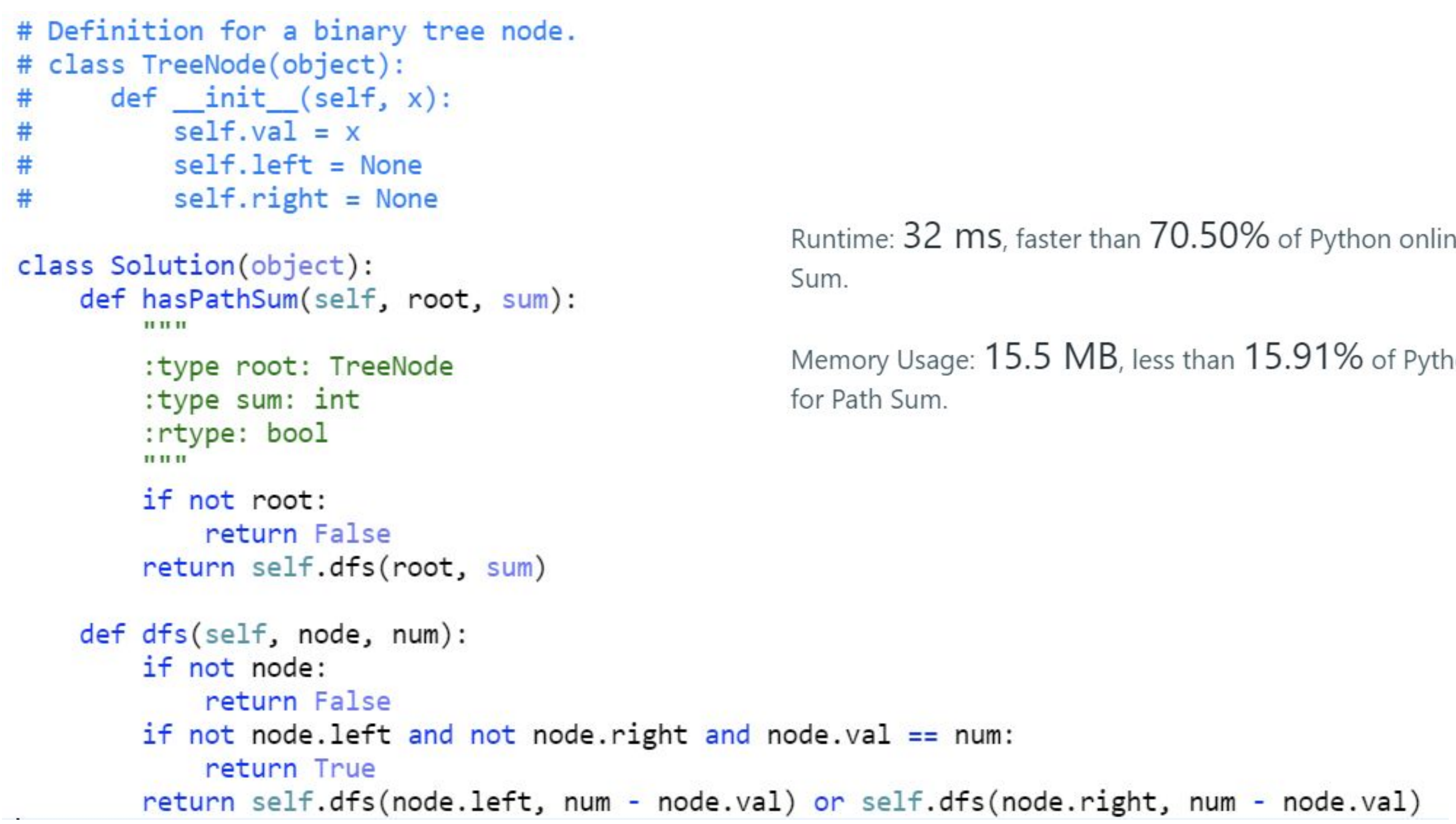


Success   Details ›

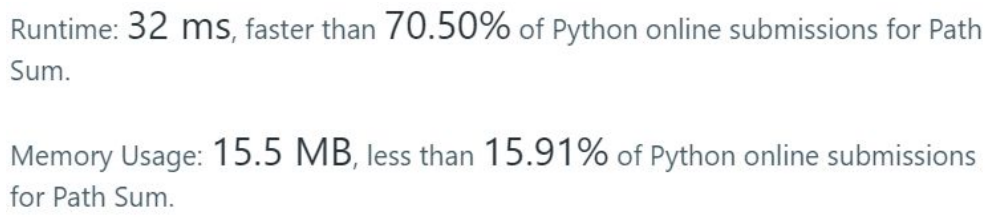
Runtime: 28 ms, faster than 89.60% of Python online submissions for Minimum Depth of Binary Tree.

Memory Usage: 14.7 MB, less than 48.72% of Python online submissions for Minimum Depth of Binary Tree.

LC 112: Path Sum



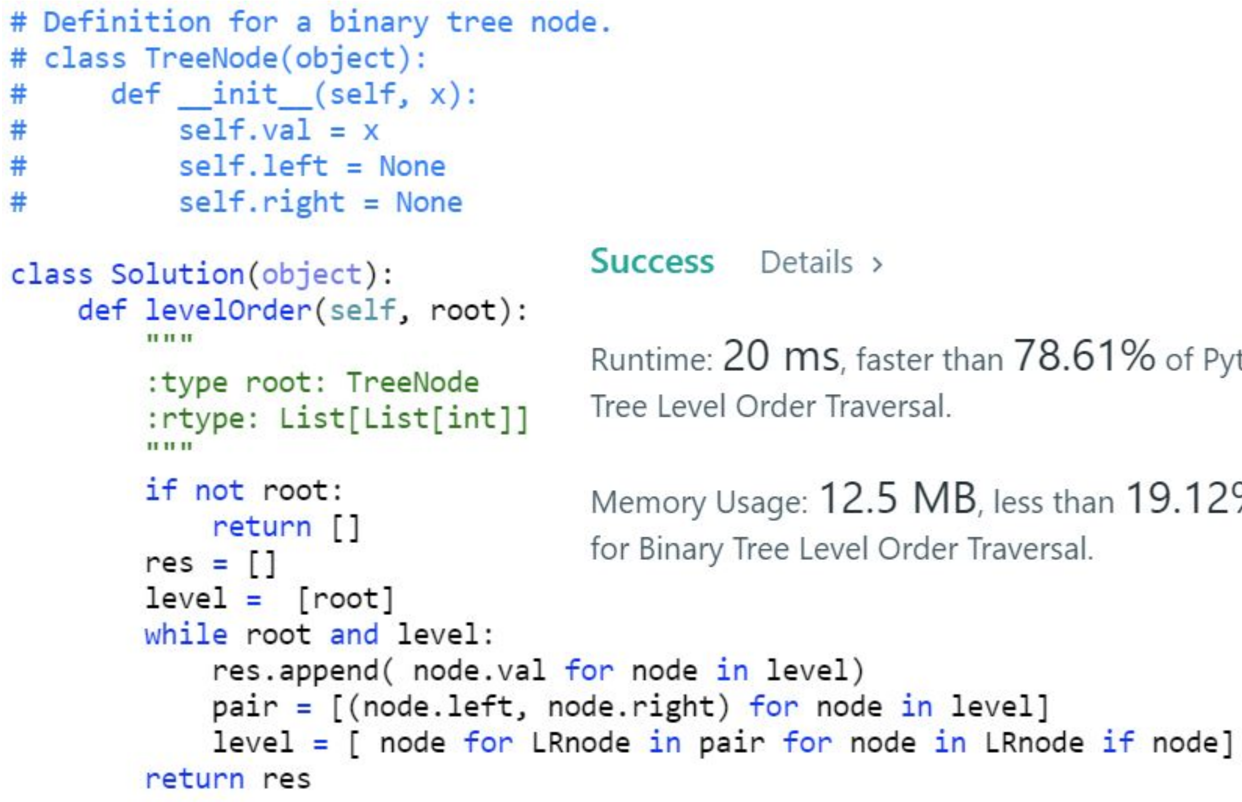
```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def hasPathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: bool
        """
        if not root:
            return False
        return self.dfs(root, sum)

    def dfs(self, node, num):
        if not node:
            return False
        if not node.left and not node.right and node.val == num:
            return True
        return self.dfs(node.left, num - node.val) or self.dfs(node.right, num - node.val)
```



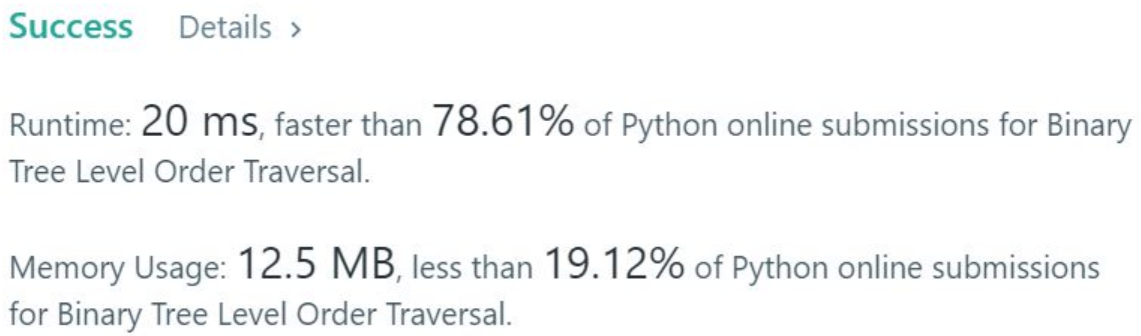
Runtime: 32 ms, faster than 70.50% of Python online submissions for Path Sum.

Memory Usage: 15.5 MB, less than 15.91% of Python online submissions for Path Sum.

LC 102: Binary Tree Level Order Traversal



```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def levelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root:
            return []
        res = []
        level =  [root]
        while root and level:
            res.append( node.val for node in level)
            pair = [(node.left, node.right) for node in level]
            level = [ node for LRnode in pair for node in LRnode if node]
        return res
```

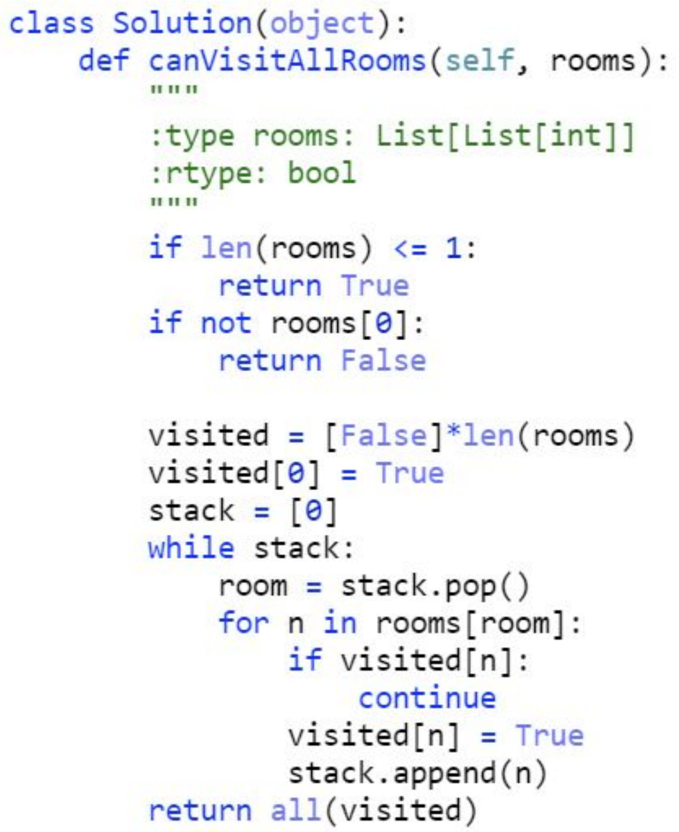


Success Details ›

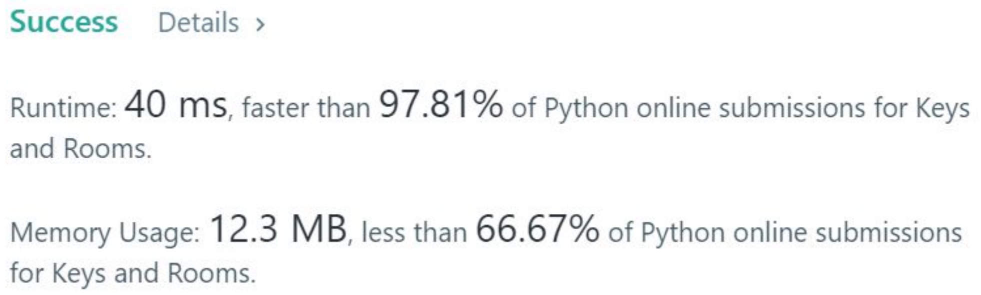
Runtime: 20 ms, faster than 78.61% of Python online submissions for Binary Tree Level Order Traversal.

Memory Usage: 12.5 MB, less than 19.12% of Python online submissions for Binary Tree Level Order Traversal.

LC 841: Keys and Rooms



```python
class Solution(object):
    def canVisitAllRooms(self, rooms):
        """
        :type rooms: List[List[int]]
        :rtype: bool
        """
        if len(rooms) <= 1:
            return True
        if not rooms[0]:
            return False

        visited = [False]*len(rooms)
        visited[0] = True
        stack = [0]
        while stack:
            room = stack.pop()
            for n in rooms[room]:
                if visited[n]:
                    continue
                visited[n] = True
                stack.append(n)
        return all(visited)
```

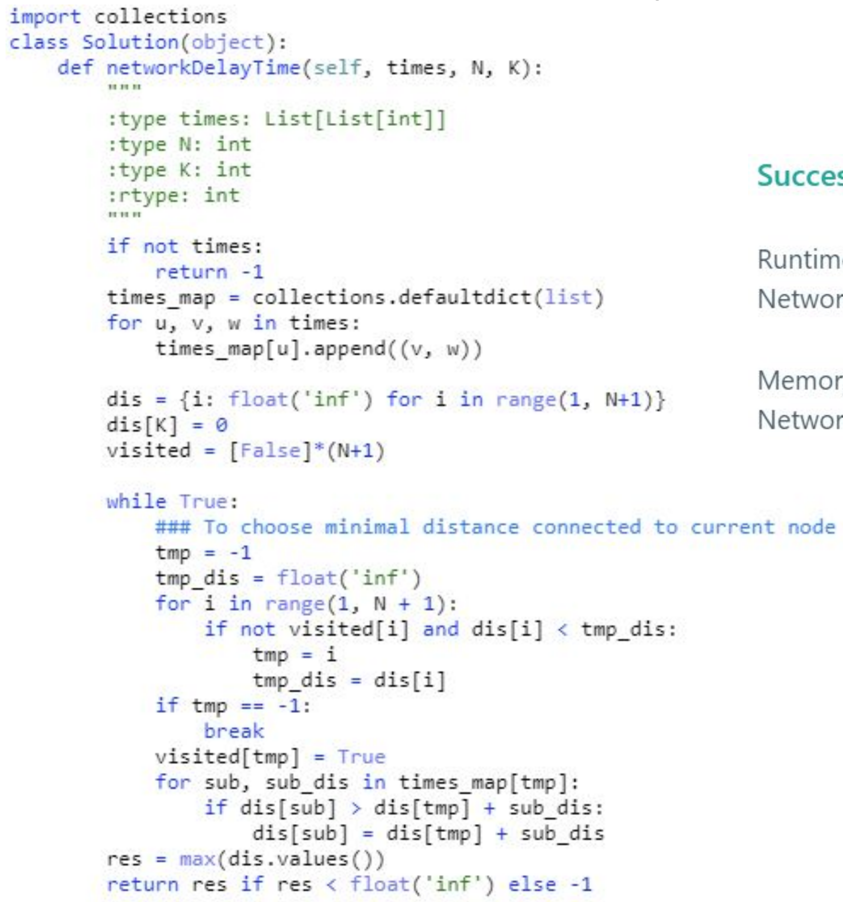


Success Details ›

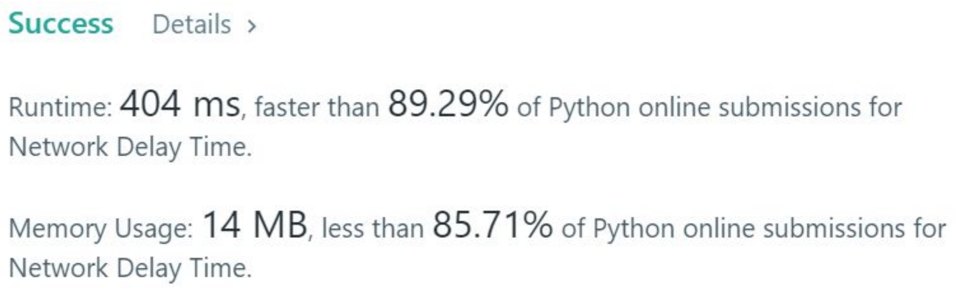
Runtime: 40 ms, faster than 97.81% of Python online submissions for Keys and Rooms.

Memory Usage: 12.3 MB, less than 66.67% of Python online submissions for Keys and Rooms.

LC 743: Network Delay Time



```python
import collections
class Solution(object):
    def networkDelayTime(self, times, N, K):
        """
        :type times: List[List[int]]
        :type N: int
        :type K: int
        :rtype: int
        """
        if not times:
            return -1
        times_map = collections.defaultdict(list)
        for u, v, w in times:
            times_map[u].append((v, w))

        dis = {i: float('inf') for i in range(1, N+1)}
        dis[K] = 0
        visited = [False]*(N+1)

        while True:
            ### To choose minimal distance connected to current node
            tmp = -1
            tmp_dis = float('inf')
            for i in range(1, N + 1):
                if not visited[i] and dis[i] < tmp_dis:
                    tmp = i
                    tmp_dis = dis[i]
            if tmp == -1:
                break
            visited[tmp] = True
            for sub, sub_dis in times_map[tmp]:
                if dis[sub] > dis[tmp] + sub_dis:
                    dis[sub] = dis[tmp] + sub_dis
        res = max(dis.values())
        return res if res < float('inf') else -1
```

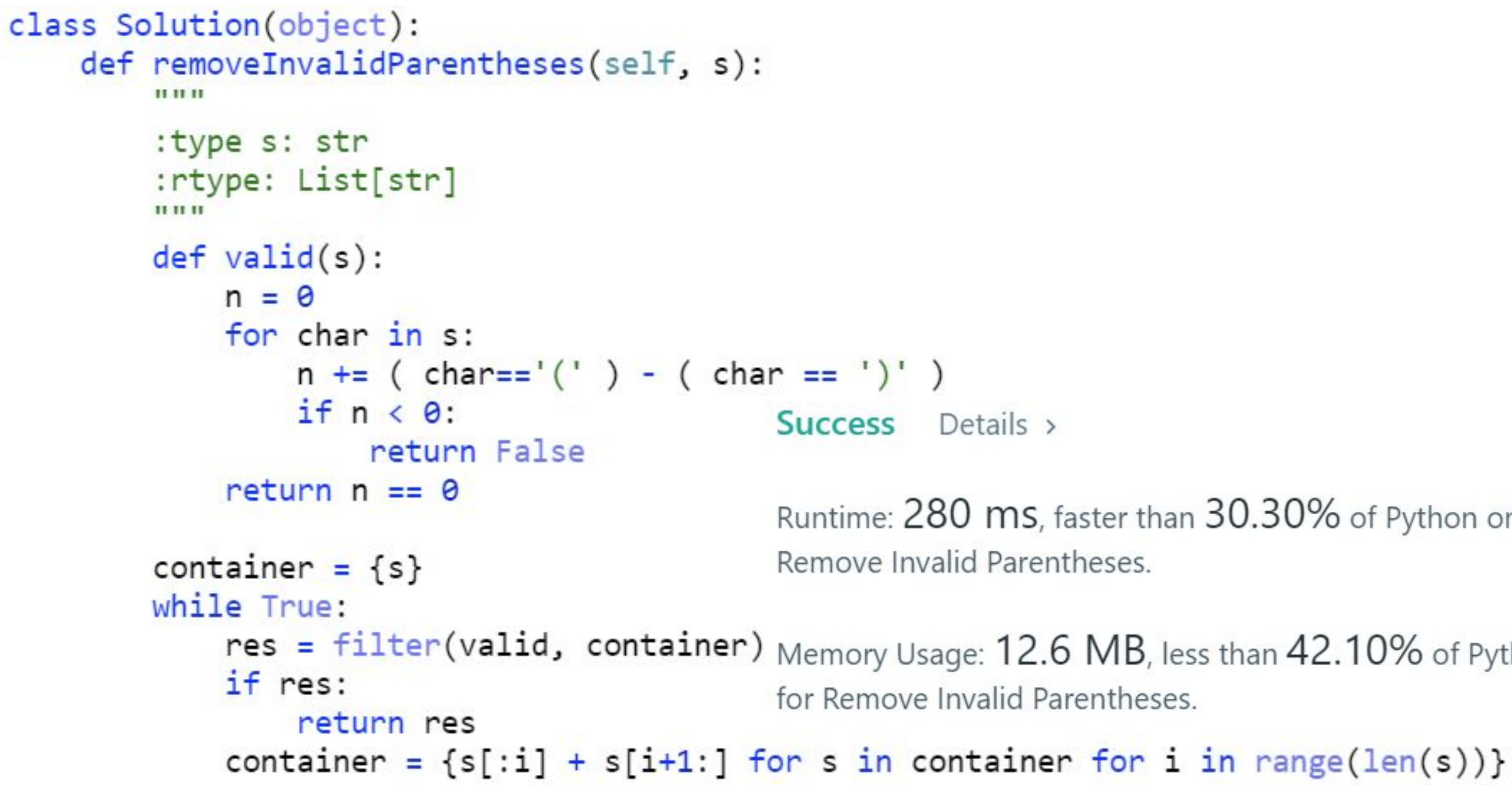


Success   Details ›

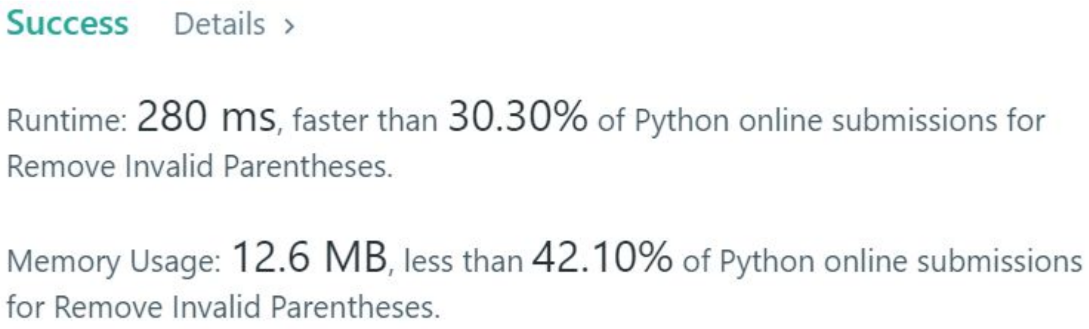
Runtime: 404 ms, faster than 89.29% of Python online submissions for Network Delay Time.

Memory Usage: 14 MB, less than 85.71% of Python online submissions for Network Delay Time.

LC 112: Remove Invalid Parentheses



```python
class Solution(object):
    def removeInvalidParentheses(self, s):
        """
        :type s: str
        :rtype: List[str]
        """
        def valid(s):
            n = 0
            for char in s:
                n += ( char=='(' ) - ( char == ')' )
                if n < 0:
                    return False
            return n == 0

        container = {s}
        while True:
            res = filter(valid, container)
            if res:
                return res
            container = {s[:i] + s[i+1:] for s in container for i in range(len(s))}
```



Success Details ›

Runtime: 280 ms, faster than 30.30% of Python online submissions for Remove Invalid Parentheses.

Memory Usage: 12.6 MB, less than 42.10% of Python online submissions for Remove Invalid Parentheses.