505. The Maze II

```
from heapq import *
class Solution:
    def shortestDistance(self, maze: List[List[int]], start: List[int], destination: List[int]) -> int:
        directions = [[0, -1], [0, 1], [1, 0], [-1, 0]]
        q = []
        distances = {}
        heappush(q, (0, start[0], start[1]))
        rows = len(maze)
        cols = len(maze[0])
        while q:
            d, x, y = heappop(q)
            if (x, y) in distances and distances[(x,y)] < d:
                continue
            distances[(x,y)] = d
            maze[x][y] = 2
            for d_x, d_y in directions:
                new_x, new_y, new_d = x + d_x, y + d_y, d + 1
                while 0 <= new_x < rows and 0 <= new_y < cols and maze[new_x][new_y] != 1:
                    new_x += d_x
                    new_y += d_y
                    new_d += 1
                new_x -= d_x
                new_y -= d_y
                new_d -= 1
                if maze[new_x][new_y] == 0:
                    heappush(q, (new_d, new_x, new_y))
        return distances[tuple(destination)] if tuple(destination) in distances else -1
```

426. Convert Binary Search Tree to Sorted Doubly Linked List

```
class Solution:
    def treeToDoublyList(self, root: 'Node') -> 'Node':
        self.head = None
        self.prev = None
        def traverse(node):
            if not node:
                return None
            traverse(node.left)
            if not self.head:
                self.head = node
            if self.prev:
                self.prev.right = node
            node.left = self.prev
            self.prev = node
            traverse(node.right)
        if not root:
            return None
        traverse(root)
        self.head.left = self.prev
        self.prev.right = self.head
        return self.head
```

Iteration:

```
class Solution:
    def treeToDoublyList(self, root: 'Node') -> 'Node':
        head = None
        prev = None
        if not root:
            return None
        stack, node = [], root
        while stack or node:
            while node:
                stack.append(node)
                node = node.left
            node = stack.pop()
            if not head:
                head = node
            if prev:
                prev.right = node
            node.left = prev
            prev = node
            node = node.right
        head.left = prev
        prev.right = head
        return head
```

105. Construct Binary Tree from Preorder and Inorder Traversal

```
class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
        if not preorder:
            return None
        root = TreeNode(preorder[0])
        idx = inorder.index(preorder[0])
        inorder_left = inorder[:idx]
        inorder_right = inorder[idx + 1:]
        preorder_left = preorder[1:1+len(inorder_left)]
        preorder_right = preorder[1+len(inorder_left):]
        root.left = self.buildTree(preorder_left, inorder_left)
        root.right = self.buildTree(preorder_right, inorder_right)
        return root
```
109. Convert Sorted List to Binary Search Tree
```
class Solution:
    def sortedListToBST(self, head: ListNode) -> TreeNode:
        arr = []
        while head:
            arr.append(head.val)
            head = head.next
        def helper(l, r):
            if l > r:
                return None
            mid = (l + r) // 2
            node = TreeNode(arr[mid])
            node.left = helper(l, mid - 1)
            node.right = helper(mid + 1, r)
            return node
        return helper(0, len(arr) - 1)
```