Leetcode Bootcamp Homework W - 7

1. Binary Tree Right Side View

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

        # a deque for efficient append and pop operations
        q = deque([root])
        result = []

        # Standard BFS level-order traversal
        while q:
            # Get the number of nodes at the current level
            level_size = len(q)

            # Iterate through all nodes at the current level
            for i in range(level_size):
                node = q.popleft()

                if i == level_size - 1:
                    result.append(node.val)

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

        return result
```

2. Rotting Oranges

```
class Solution(object):
    def orangesRotting(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        rows, cols = len(grid), len(grid[0])
        # queue for BFS, storing (row, col)
        queue = deque()
        fresh_count = 0
        minutes = 0

        for r in range(rows):
            for c in range(cols):
                if grid[r][c] == 2:
                    queue.append((r, c))
                elif grid[r][c] == 1:
                    fresh_count += 1

        dir = [(0, 1), (0, -1), (1, 0), (-1, 0)]

        while queue and fresh_count > 0:
            size = len(queue)

            for _ in range(size):
                r, c = queue.popleft()

                for dr, dc in dir:
                    nr, nc = r + dr, c + dc

                    if (0 <= nr < rows and
                        0 <= nc < cols and
                        grid[nr][nc] == 1):

                        grid[nr][nc] = 2
                        fresh_count -= 1

                        queue.append((nr, nc))

            minutes += 1
```

```
            # If fresh_count is 0, all fresh oranges rotted, return the
    elapsed time.
            if fresh_count == 0:
                return minutes
            # If fresh_count is > 0, it means some fresh oranges were
    unreachable.
            else:
                return -1
```

3. Course Schedule II

```
class Solution(object):
    def findOrder(self, numCourses, prerequisites):
        """
        :type numCourses: int
        :type prerequisites: List[List[int]]
        :rtype: List[int]
        """
        graph = defaultdict(list)
        in_degree = [0] * numCourses

        for course, prereq in prerequisites:
            graph[prereq].append(course)

            in_degree[course] += 1

        queue = deque()

        for i in range(numCourses):
            if in_degree[i] == 0:
                queue.append(i)

        result_order = []

        while queue:
            prereq = queue.popleft()
            result_order.append(prereq)

            for dependent_course in graph[prereq]:
                in_degree[dependent_course] -= 1
```

```
            if in_degree[dependent_course] == 0:
                queue.append(dependent_course)

    if len(result_order) == numCourses:
        return result_order
    else:
        return []
```