

Assignment -7

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
```

```
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        findRightView(root,0,result);
        return result;
    }
    public void findRightView(TreeNode root, int level,List<Integer> result){
        if(root == null) return;
        if(level == result.size()) result.add(root.val);
        findRightView(root.right,level+1,result);
        findRightView(root.left,level+1,result);
    }
}
```

Screenshot of a LeetCode submission page for the problem "Binary Tree Right Side View".

The page shows the problem description, the user's solution code in Java, and the submission results.

Submission Details:

- Accepted: 27 / 27 Testcases passed
- arnabdas19... submitted at Apr 22, 2025 14:11
- Runtime: 0 ms, Beats 100.00%
- Memory: 42.29 MB, 43.95%

Code:

```
2  * Definition for a binary tree node.
3  * public class TreeNode {
4  *     int val;
5  *     TreeNode left;
6  *     TreeNode right;
7  *     TreeNode() {}
8  *     TreeNode(int val) { this.val = val; }
9  *     TreeNode(int val, TreeNode left, TreeNode right) {
10 *         this.val = val;
11 *         this.left = left;
12 *         this.right = right;
13 *     }
14 * }
15 */
16 class Solution {
17     public List<Integer> rightSideView(TreeNode root) {
18         List<Integer> result = new ArrayList<>();
19         findRightView(root,0,result);
20         return result;
21     }
22 }
```

Test Result:

- Accepted
- Runtime: 0 ms
- Case 1, Case 2, Case 3, Case 4

Input:

```
root = [1,2,3,null,5,null,4]
```

Question 2

```
class Solution {
    public int orangesRotting(int[][] grid) {
        if (grid == null || grid.length == 0) return -1;
        int rows = grid.length, cols = grid[0].length;
        Queue<int[]> queue = new LinkedList<>();
        int freshCount = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (grid[i][j] == 2)
                    queue.offer(new int[]{i, j});
                else if (grid[i][j] == 1)
                    freshCount++;
            }
        }
        if (freshCount == 0) return 0;
        int minutes = 0;
        int[][] directions = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
        while (!queue.isEmpty()) {
            int size = queue.size();
            boolean infected = false;
            for (int i = 0; i < size; i++) {
                int[] point = queue.poll();
                int x = point[0], y = point[1];
                for (int[] dir : directions) {
                    int nx = x + dir[0];
                    int ny = y + dir[1];
                    if (nx >= 0 && ny >= 0 && nx < rows && ny < cols && grid[nx][ny] == 1) {
                        grid[nx][ny] = 2;
                        queue.offer(new int[]{nx, ny});
                        freshCount--;
                        infected = true;
                    }
                }
            }
            if (infected) minutes++;
        }
        return freshCount == 0 ? minutes : -1;
    }
}
```

leetcode.com/problems/rotting-oranges/

Problem List

Description Accepted Editorial Solutions Sub Code

All Submissions

Accepted

arnabdas19... submitted at Apr 22, 2025 14:16

Runtime

1 ms Beats 100.00%

Analyze Complexity

Memory

42.53 49.71%

Case	Time	Memory
1	1ms	~42.53 MB
2	2ms	~49.71 MB
3	3ms	~42.53 MB
4	4ms	~42.53 MB
5	5ms	~42.53 MB
6	6ms	~42.53 MB

```

1 class Solution {
2     public int orangesRotting(int[][] grid) {
3         if (grid == null || grid.length == 0) return -1;
4         int rows = grid.length, cols = grid[0].length;
5         Queue<int[]> queue = new LinkedList<>();
6         int freshCount = 0;
7         for (int i = 0; i < rows; i++) {
8             for (int j = 0; j < cols; j++) {
9                 if (grid[i][j] == 2)
10                     queue.offer(new int[]{i, j});
11                 else if (grid[i][j] == 1)
12                     freshCount++;
13             }
14         }
15         if (freshCount == 0) return 0;
16         int minutes = 0;
17         int[][] directions = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
18         while (!queue.isEmpty()) {
19             int size = queue.size();
20             boolean infected = false;
```

Test Result

Accepted Runtime: 0 ms

- Case 1
- Case 2
- Case 3

Input

```

grid =
[[2,1,1],[1,1,0],[0,1,1]]
```

```

class Solution {
    public int[] findOrder(int numCourses, int[][] prerequisites) {
        List<Integer>[] graph = new ArrayList[numCourses];
        int[] indegree = new int[numCourses];
        for (int i = 0; i < numCourses; i++) {
            graph[i] = new ArrayList<>();
        }
        for (int[] pair : prerequisites) {
            int course = pair[0];
            int prereq = pair[1];
            graph[prereq].add(course);
            indegree[course]++;
        }
        Queue<Integer> queue = new LinkedList<>();
        for (int i = 0; i < numCourses; i++) {
            if (indegree[i] == 0)
                queue.offer(i);
        }
        int[] order = new int[numCourses];
        int index = 0;
        while (!queue.isEmpty()) {
            int course = queue.poll();
            order[index++] = course;

            for (int neighbor : graph[course]) {
                indegree[neighbor]--;
                if (indegree[neighbor] == 0)
                    queue.offer(neighbor);
            }
        }
        return index == numCourses ? order : new int[0];
    }
}

```

Accepted 40/40 test cases passed
arnabdas19... submitted at Apr 22, 2025 14:19

Solution

Runtime

4 ms Beat: 91.82%

Analyze Complexity

Memory

45.47 95.53%



```
class Solution {  
    public int[] findOrder(int numCourses, int[][] prerequisites) {  
        // ...  
    }  
}
```

```
16         if (indegree[i] == 0)  
17             queue.offer(i);  
18     }  
19     int[] order = new int[numCourses];  
20     int index = 0;  
21     while (!queue.isEmpty()) {  
22         int course = queue.poll();  
23         order[index++] = course;  
24     }  
25     for (int neighbor : graph[course]) {  
26         indegree[neighbor]--;  
27         if (indegree[neighbor] == 0)  
28             queue.offer(neighbor);  
29     }  
30 }  
31 return index == numCourses ? order : new int[0];  
32 }  
33 }
```

14/44 (34.1%) Solved

Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

numCourses =
2

Run Submit