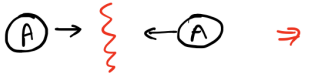
Tuesday, 30 July 2024
8:44PM

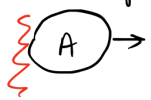
Queues + Stack Problem Solving Session

- → Rotten Oranges
- → Asteroid Collision
- → Sliding Window Maximum

Asteroid Collision

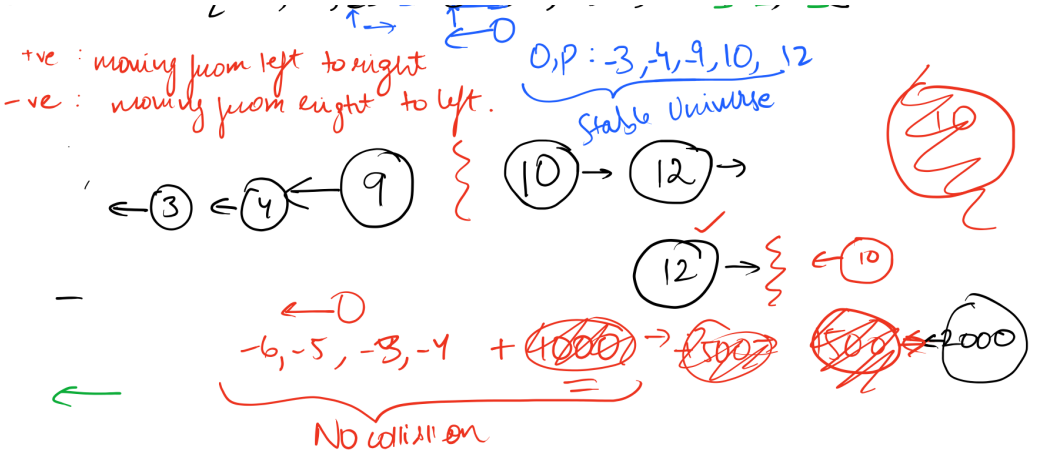


→ Smaller asteroid is destroyed
→ Larger remains unaffected
→ If both are same, both with explode

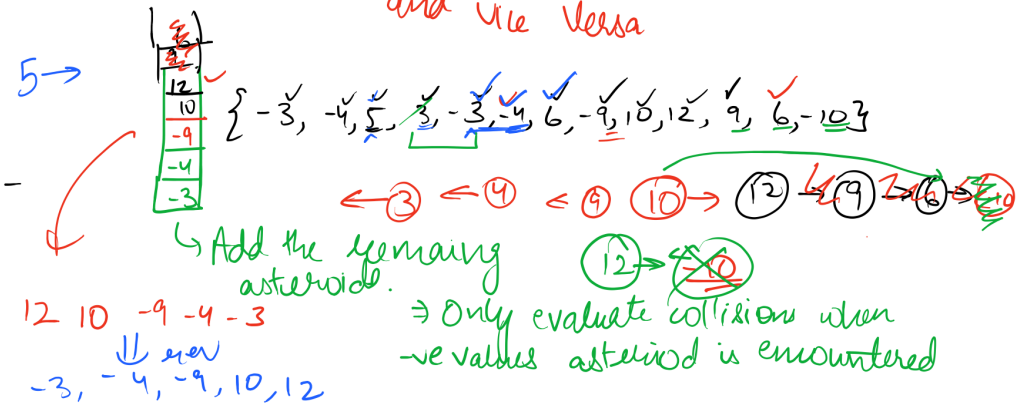


⇒ Remained Unaffected

[] asteroids = {-3, -4, 5, 3, -5, -4, 6, -9, 10, 12, 9, 6, -10}



Any +ve value will continue moving towards right if not encounter by -ve valued asteroid.
and vice versa

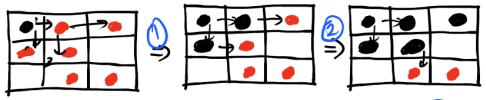


12 10 -9 -4 -3
= even
-3, -4, -9, 10, 12

TC: O(N) SC: O(N)

Rotting Oranges

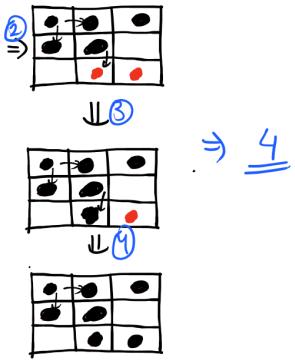
2, 1, 1
1, 1, 0
0, 1, 1



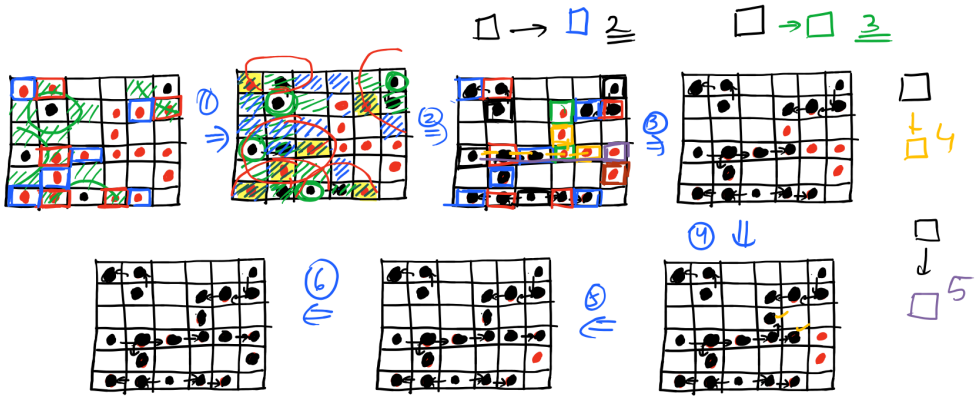
2 → Rotten Orange

1 → Fresh Orange

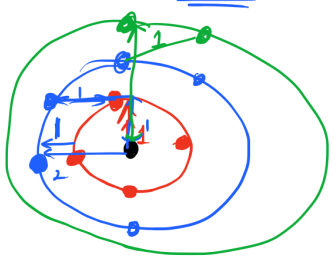
0 → Empty



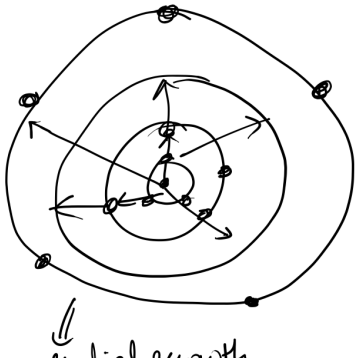
⇒ 4



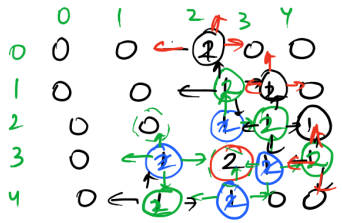
Ans ⇒ 6 min



Breadth First Search



(BFS)



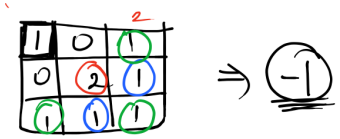
time = 0 → 1 → 2 → 3 → 4

Steps for BFS

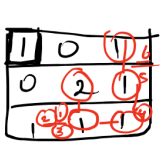
Queue → add all the srcs

→ try to remove each node, add new unvisited nodes in all 4 dirn

→ when all the nodes from a particular level are visited increase the time.



⇒ -1



Recursion ✗

2 | 2 | | (2 |

```
static class Pair {
    int v;
```

```
        int x;
        int y;

        Pair(int x, int y) {
                this.x = x;
                this.y = y;
        }
}

public static int orangesRotting(int[][] grid) {

        int m = grid.length;
        int n = grid[0].length;

        int time = 0;
        int countFresh = 0;
        Queue<Pair> queue = new LinkedList<>();
        for (int r = 0; r < m; r++) {
                for (int c = 0; c < n; c++) {
                        if (grid[r][c] == 1) {
                                countFresh++;
                        } else if (grid[r][c] == 2) {
                                // rotten orange
                                queue.add(new Pair(r, c));
                        }
                }
        }
        if (countFresh == 0) {
                return 0;
        }
        while (queue.size() != 0) {
                int rottenOrangesInCurrentLevel = queue.size();
                while (rottenOrangesInCurrentLevel-- > 0) {
                        Pair rp = queue.remove();
                        int r = rp.x;
                        int c = rp.y;
                        // up
                        if (r - 1 >= 0 && grid[r - 1][c] == 1) {
                                grid[r - 1][c] = 2;
                                queue.add(new Pair(r - 1, c));
                                countFresh--;
                        }
                        // down
                        if (r + 1 < m && grid[r + 1][c] == 1) {
                                grid[r + 1][c] = 2;
                                queue.add(new Pair(r + 1, c));
                                countFresh--;
                        }
                        // left
                        if (c - 1 >= 0 && grid[r][c - 1] == 1) {
                                grid[r][c - 1] = 2;
                                queue.add(new Pair(r, c - 1));
                                countFresh--;
                        }
                        // right
                        if (c + 1 < n && grid[r][c + 1] == 1) {
                                grid[r][c + 1] = 2;
                                queue.add(new Pair(r, c + 1));
                                countFresh--;
                        }
                }
                time++;
        }
        if (countFresh != 0) {
                // some fresh orange remains
                return -1;
        }
        return time - 1;
}
```

CF = 4 3 2 1 0

time = 0 1 2 3 4 → 5

(0,0) (1,0) (0,1) (0,2) (2,1) (2,2)

curr = 2 1 0

curr Level = 2 1 0

Level

Pair (0,0)
U D L R

Pair (1,0)
U D L R

Pair (0,1)
U D L R ✓

curr Level = 2 1 0

Pair (1,1) Pair = (0,2)

U D ✓ L R

U D ✗ L R

curr = 1 0

Pair (2,1)

U ✗ D ✗ L ✗ R ✓

curr L = 1

Pair (2,2)

U D L R

BFS → Queue

TC: O(MN)

SC: O(MN)