Problem 1

Implement a queue using 2 stacks.

a. Implement the `enqueue()` operation in O(1) time, and `dequeue()` in O(n) time
b. Implement the `enqueue()` operation in O(n) time, and `dequeue()` in O(1) time

References

- https://www.techiedelight.com/implement-a-queue-using-stack-data-structure/
- https://www.geeksforgeeks.org/queue-using-stacks/

Problem 2

Implement a stack using 2 queues.

a. Implement the `push()` operation in O(1) time, and `pop()` in O(n) time
b. Implement the `push()` operation in O(n) time, and `pop()` in O(1) time

References

- https://www.techiedelight.com/implement-stack-using-queue-data-structure/
- https://www.geeksforgeeks.org/implement-stack-using-queue/

Problem 3

Given a matrix of dimension m*n where each cell in the matrix can have values 0, 1 or 2 which has the following meaning:

- 0: Empty cell
- 1: Cells have fresh oranges
- 2: Cells have rotten oranges

Determine what is the minimum time required so that all the oranges become rotten. A rotten orange at index [i,j] can rot other fresh orange at indexes [i-1,j], [i+1,j], [i,j-1], [i,j+1] (up, down, left and right) in 1 day. If it is impossible to rot every orange then simply return -1.

Examples

1. Input for which all oranges will rot

```
Input:  arr[][C] = { {2, 1, 0, 2, 1}, {1, 0, 1, 2, 1}, {1, 0, 0, 2, 1}};
Output: 2

Explanation: At 0th time frame:
{2, 1, 0, 2, 1}
{1, 0, 1, 2, 1}
{1, 0, 0, 2, 1}
At 1st time frame:
{2, 2, 0, 2, 2}
{2, 0, 2, 2, 2}
{1, 0, 0, 2, 2}
At 2nd time frame:
{2, 2, 0, 2, 2}
```

```
{2, 0, 2, 2, 2}
{2, 0, 0, 2, 2}
```

2. Input for which sone orange will remain fresh

```
Input:  arr[][C] = { {2, 1, 0, 2, 1}, {0, 0, 1, 2, 1}, {1, 0, 0, 2, 1}}
Output: -1

Explanation: At 0th time frame:
{2, 1, 0, 2, 1}
{0, 0, 1, 2, 1}
{1, 0, 0, 2, 1}
At 1st time frame:
{2, 2, 0, 2, 2}
{0, 0, 2, 2, 2}
{1, 0, 0, 2, 2}
At 2nd time frame:
{2, 2, 0, 2, 2}
{0, 0, 2, 2, 2}
{1, 0, 0, 2, 2}
The 1 at the bottom left corner of the matrix is never rotten.
```

a. Easy method - Calculate what happens in 1 day in a loop iteration (while loop is best as number of iterations is not known in advance). In every iteration, visit every orange, and if it is rotten, mark its fresh neighbours by setting them to one more than the value of the rotten orange. For example, in the first iteration the rotten orange is marked 2 - so set the fresh neighbours which are 1, as 3 instead. In the next iteration, mark the fresh neighbours of 3 as 4, etc.

Maintain a flag to track is some orange was marked rotten in the current loop iteration. When marking some orange as rotten, set the flag. At the end of the loop, if none were marked rotten, then we are done, and we can break out of the loop.

After breaking out of the loop, go through the matrix and check if there is any fresh orange or not. Return -1 if there is something fresh. Else, the number of days to rot would be 2 less than the last number set.

b. Efficient method - This uses a queue. Check the reference for details.

References

- https://www.geeksforgeeks.org/minimum-time-required-so-that-all-oranges-become-rotten/