COL106 : Data Structures and Algorithms, Semester II 2024-25
Practice Programming Questions on Stack&Queues

January 2025

Instructions

- Please use the following questions as practice questions for learning about Stacks and Queue data structures.
- The questions with * next to them should be attempted during the lab sessions, and your solutions must be uploaded on moodlenew. Note that your submissions will not be evaluated, but will be used as a mark of your attendance. We will filter out all the submissions that are not from the lab workstations. So do not use your laptops for submitting the programs.

1 Questions

1. **Implement a Queue Using Two Stacks**

   Design a queue that uses two stacks to implement the standard queue operations: `enqueue`, `dequeue`, and `front`, while maintaining $O(1)$ **amortized time complexity** for each operation.

   Implementation: Construct a class `MyQueue` with the following methods:

   - `void enqueue(int x)`: Adds an element `x` to the end of the queue.
   - `void dequeue()`: Removes the element at the front of the queue.
   - `int front()`: Retrieves the element at the front of the queue.
   - `boolean isEmpty()`: Checks whether the queue is empty.

   Starter Code:

```
class MyQueue {
    private Stack<Integer> stack1; // For enqueue
    private Stack<Integer> stack2; // For dequeue

    public MyQueue() {
        stack1 = new Stack<>();
        stack2 = new Stack<>();
    }

    public void enqueue(int x) {
        // Your code here
    }

    public void dequeue() {
        // Your code here
```

```
    }

    public int front() {
        // Your code here
    }

    public boolean isEmpty() {
        // Your code here
    }

    // Optional: Helper method for transferring elements
    private void transferElements() {
        // Your code here
    }
}
```

Listing 1: Queue Using Two Stacks

Example:

- **Input:**

```
MyQueue queue = new MyQueue();
queue.enqueue(1);
queue.enqueue(2);
System.out.println(queue.front()); // 1
queue.dequeue();
System.out.println(queue.front()); // 2
```

- **Output:**

```
1
2
```

2. **Implement a MinStack**

Design a stack that supports standard stack operations (`push`, `pop`, `top`) in $O(1)$ time, and additionally supports retrieving the minimum element in the stack in $O(1)$ time.

Implementation: Construct a class `MinStack` with the following methods:

- `void push(int x)`: Pushes the element `x` onto the stack.
- `void pop()`: Removes the element on the top of the stack.
- `int top()`: Retrieves the element on the top of the stack.
- `int getMin()`: Retrieves the minimum element in the stack.

```
class MinStack {
    //Declare variable you need

    // Constructor
    public MinStack() {
        //Initialize variables here
    }

    // Implement the following methods:
```

```
    public void push(int x) {
        // Your code here
    }

    public void pop() {
        // Your code here
    }

    public int top() {
        // Your code here
    }

    public int getMin() {
        // Your code here
    }
}
```

Listing 2: MinStack Starter Code

3. **Decode an Encoded String**

   You are given an encoded string s that follows a specific pattern: `k[encoded_string]`, where k is a positive integer representing the number of times the `encoded_string` should be repeated to construct the decoded string. The encoded string contains letters and/or nested encoded strings. Decode s and return the decoded string using a stack implemented from scratch.

   Example:

   - **Input 1:**
     `s = "3[a2[c]]"`
   - **Result:**
     `"accaccacc"`
   - **Input 2:**
     `s = "2[abc]3[cd]ef"`
   - **Result:**
     `"abcabccdcdcdef"`

4. **Drone Programming and Movement Simulation** IIT Delhi recently deployed a drone for aerial surveillance on Independence Day. The drone starts at position $(0, 0, 0)$ and moves according to a programmed sequence of instructions. The instructions include: $+X$ (move one unit in the positive X-axis), $-X$ (move one unit in the negative X-axis), $+Y$ (move one unit in the positive Y-axis), $-Y$ (move one unit in the negative Y-axis), $+Z$ (move one unit in the positive Z-axis), $-Z$ (move one unit in the negative Z-axis), and $m(P)$ where $m > 0$ is an integer and P is a drone program to be repeated m times.

   Write a function `solve(String expr)` that takes a string `expr` as input and returns a list of 4 integers: the final X, Y, and Z coordinates of the drone, followed by the total distance traveled.

   Example

   - **Input 1:**
     `"2(+X+Y-Z)"`
   - **Result:**
     `[2,2,-2,6]`
   - **Input 2:**
     `s = "5(+X)10(-X)"`

- **Result:**
  `[-5,0,0,15]`

5. **Validate Stack sequence** You are given two integer arrays, `pushed` and `popped`, where:

   - `pushed` represents the order in which integers are pushed onto a stack.
   - `popped` represents the order in which integers are popped from the stack.

   Your task is to determine if the sequence of operations represented by the `pushed` array could result in the sequence represented by the `popped` array.

   ## Example

   Input:

   `pushed` = [1, 2, 3, 4, 5]
   `popped` = [4, 5, 3, 2, 1]

   Output:

   `true`

6. **Who's in Your Line of Sight?**

   Imagine you're at a student assembly, standing in a straight line with your friends. Everyone is eager to look ahead, but here's the twist—you're a little short-sighted! You can only see someone **if they are taller than you**, and even then, you can only see **one person at a time**. Let's solve some challenges to figure out who's in your line of sight!

   Challenge 1: The Next Tallest Friend

   For each person in the line, figure out who the next taller person is. If there's no one taller in sight, return -1.

   Expected Time Complexity: $O(N)$

   Example:

   - **Input:**
     $[6, 1, 3, 2, 4, 5, 2, 8]$
   - **Output:**
     $[8, 3, 4, 4, 5, 8, 8, -1]$

   Challenge 2: The Power of "Selective Vision"

   Now, you've unlocked a magical ability: you can **skip one person** in the line! Your task is to find the **second taller person** you can spot to your right. If no such person exists, return -1.

   Expected Time Complexity: $O(N)$

   Example:

   - **Input:**
     $[6, 1, 3, 2, 4, 5, 2, 8]$
   - **Output:**
     $[-1, 2, 5, 5, 8, -1, -1, -1]$

   Notes:

   - Heights are **unique**, so no two students are of the same height.

- Use your logic and coding skills to help these students figure out their line of sight!

‟

7. **Implement a MinQueue Using MinStack(From Q3)**

   Using the `MinStack` implementation, design a queue that supports the standard queue operations (`enqueue`, `dequeue`) in $O(1)$ or amortized $O(1)$ time, while also allowing retrieval of the minimum element in the queue in $O(1)$ time.

   Implementation: Construct a class `MinQueue` with the following methods:

   - `void enqueue(int x)`: Adds the element `x` to the rear of the queue.
   - `void dequeue()`: Removes the element at the front of the queue.
   - `int front()`: Retrieves the element at the front of the queue.
   - `int getMin()`: Retrieves the minimum element in the queue.

```
class MinQueue {
    private MinStack enqueueStack;
    private MinStack dequeueStack;

    // Constructor
    public MinQueue() {
        enqueueStack = new MinStack();
        dequeueStack = new MinStack();
    }

    // Implement the following methods:
    public void enqueue(int x) {
        // Your code here
    }

    public void dequeue() {
        // Your code here
    }

    public int front() {
        // Your code here
    }

    public int getMin() {
        // Your code here
    }

    // Helper method to transfer elements between stacks
    private void transferElements() {
        // Your code here
    }
}
```

Listing 3: MinQueue Starter Code

8. **The Coldplay Concert Dilemma**

   Coldplay is performing a grand concert, and you're responsible for organizing the VIP seating. Each row of seats is represented as an array of integers (`nums`), where each value indicates a fan's enthusiasm level. You are also given the Threshold of Excitement (`threshold`).

Your task is to find a group of fans seated together (a contiguous subarray of length `k`) such that **every fan's enthusiasm is greater than** `threshold/k`. If such a group exists, return the size of the subarray (`k`). If no such subarray exists, return -1.

Write a function `solve(int[] nums, int threshold)` that takes as input an array of integers `nums` and an integer `threshold`, and returns the size of any subarray where each element is greater than `threshold/size of subarray`.

Expected Time Complexity: $O(N)$

Example:

- **Input:**

  `nums = [1,3,4,3,1], threshold = 6`

- **Output:**

  `3`

- **Explanation:** The subarray [3, 4, 3] has length `k` $= 3$, and each value satisfies

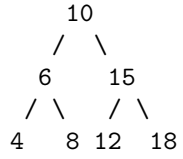
  $$\texttt{value} > \texttt{threshold/k} = 6/3 = 2.$$

  Hence, the answer is 3.

9. **Level-Order Traversal of a Binary Tree**

   A binary tree is a data structure with a root node, and every subsequent node can have 0/1/2 'children'. Each node has a left child and a right child (which may be null pointers (empty)). Each child node could be another sub-tree with its own children. You are provided with a `Node` class and a `BinaryTree` class. Your task is to implement the `printLevelOrderUsingQueue()` method to perform a **level-order traversal** of the binary tree using a **Queue**. Level order traversal means reading the tree from top to bottom, left to right in each height level.

   Expected Output: For the following tree:



   The level-order traversal should output:

   `10 6 15 4 8 12 18`

```java
//Starter code

import java.util.LinkedList;
import java.util.Queue;

class Node {
    int data;
    Node left, right;
```

```
9
10      Node(int data) {
11          this.data = data;
12          this.left = this.right = null;
13      }
14  }
15
16  class BinaryTree {
17      Node root;
18
19      // Function to perform level-order traversal using a queue
20      public void printLevelOrderUsingQueue() {
21          // Write your code here
22      }
23
24  public class Lab {
25      public static void main(String[] args) {
26          BinaryTree tree = new BinaryTree();
27
28          // Create a custom binary tree
29          tree.root = new Node(10);
30          tree.root.left = new Node(6);
31          tree.root.right = new Node(15);
32          tree.root.left.left = new Node(4);
33          tree.root.left.right = new Node(8);
34          tree.root.right.left = new Node(12);
35          tree.root.right.right = new Node(18);
36
37          // Print level-order traversal using queue
38          System.out.println("Level Order Traversal using Queue:");
39          tree.printLevelOrderUsingQueue();
40      }
41  }
```

Node Class: Represents a node in the binary tree with fields `data`, `left`, and `right`.

BinaryTree Class: Holds the root node of the binary tree. The `printLevelOrderUsingQueue()` method implements level-order traversal using a queue.

Level-Order Traversal Using Queue: To be implemented **Write code for the printLevelOrderUsingQueue() function**.

10. **Sum of maximum of all subarrays**

    Given an array arr[], the task is to find the sum of the maximum elements of every possible non-empty sub-arrays of the given array arr[] modulo 998244353.

    Example

    Input: arr[] = [1, 3, 2]

    Output: 15

    Explanation: All possible non-empty subarrays of [1, 3, 2] are 1, 3, 2, 1, 3, 3, 2 and 1, 3, 2. The maximum elements of the subarrays are 1, 3, 2, 3, 3, 3 respectively. The sum will be 15.

    Accepted Solution:

    - **Time Complexity:** $O(N)$
    - **Space Complexity:** $O(N)$

11. **Finding stock span**

We define the **span** of the stock's price in one day as the maximum number of consecutive days (starting from that day and going backward) for which the stock price was less than or equal to the price of that day.

Given an input arr[] containing stock prices for all days, you are supposed to output the span for the stock price of each day.

Example

Input: arr[] = [100, 80, 60, 70, 60, 75, 85]

Output: 1 1 1 2 1 4 6

Explanation:

- Span for day 1 is always 1.
- Span for day 2 is 1 since price decreased.
- Span for day 4 is 2 since price was less than or equal for 2 days (current day is also included).

Accepted Solution:

- **Time Complexity:** $O(N)$
- **Space Complexity:** $O(N)$