

Recitation 06

Monotonic Stack/queue

Adithya Iyer, March 1

Queue

Problem1. Implement Stack using Queues

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

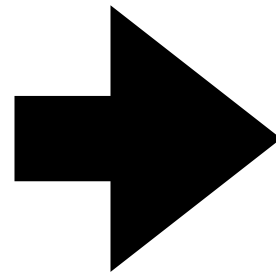
Implement the MyStack class:

`void push(int x)` Pushes element `x` to the top of the stack.

`int pop()` Removes the element on the top of the stack and returns it.

`int top()` Returns the element on the top of the stack.

`boolean empty()` Returns true if the stack is empty, false otherwise.



```
public MyStack() {
    q1 = new LinkedList<>();
    q2 = new LinkedList<>();
}

public void push(int x) {
    q2.offer(x);
    while (!q1.isEmpty()) {
        q2.offer(q1.poll());
    }
    Queue<Integer> temp = q1;
    q1 = q2;
    q2 = temp;
    topElement = x;
}

public int pop() {
    return q1.poll();
}

public int top() {
    return topElement;
}

public boolean empty() {
    return q1.isEmpty();
}
```


Monotonic Stack and Queue

Problem2. Given an integer array `nums`, return the number of non-empty subarrays with the leftmost element of the subarray not larger than other elements in the subarray. A subarray is a contiguous part of an array.

Example 1:

```
Input: nums = [1,4,2,5,3]
Output: 11
Explanation: There are 11 valid subarrays: [1],[4],[2],
[5],[3],[1,4],[2,5],[1,4,2],[2,5,3],[1,4,2,5],
[1,4,2,5,3].
```

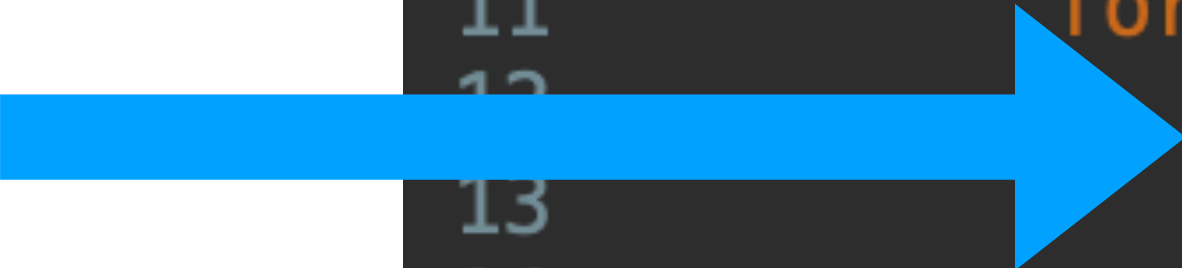
Example 2:

```
Input: nums = [3,2,1]
Output: 3
Explanation: The 3 valid subarrays are: [3],[2],[1].
```

Example 3:

```
Input: nums = [2,2,2]
Output: 6
Explanation: There are 6 valid subarrays: [2],[2],[2],
[2,2],[2,2],[2,2,2].
```

```
4
5 public class SubarrayCountUsingStack {
6     public static int countSubarrays(int[] nums) {
7         int n = nums.length;
8         Stack<Integer> stack = new Stack<>();
9         int totalCount = 0;
10
11         for (int i = n - 1; i >= 0; i--) {
12             while (!stack.isEmpty() && nums[stack.peek()] >= nums[i]) {
13                 stack.pop();
14             }
15             int nextSmallerIndex = stack.isEmpty() ? n : stack.peek();
16             totalCount += (nextSmallerIndex - i);
17             stack.push(i);
18         }
19
20         return totalCount;
21     }
22 }
```



Problem3.

There are n people standing in a queue, and they numbered from 0 to $n - 1$ in left to right order. You are given an array heights of distinct integers where heights[i] represents the height of the i th person.

A person can see another person to their right in the queue if everybody in between is shorter than both of them. More formally, the i th person can see the j th person if $i < j$ and $\min(\text{heights}[i], \text{heights}[j]) > \max(\text{heights}[i+1], \text{heights}[i+2], \dots, \text{heights}[j-1])$.

Return an array answer of length n where answer[i] is the number of people the i th person can see to their right in the queue.

Input: heights = [10,6,8,5,11,9]

Output: [3,1,2,1,1,0]

Explanation:

Person 0 can see person 1, 2, and 4.

Person 1 can see person 2.

Person 2 can see person 3 and 4.

Person 3 can see person 4.

Person 4 can see person 5.

Person 5 can see no one since nobody is to the right of them.

```
public static int[] canSeePersonsCount(int[] heights) {  
    int length = heights.length;  
    int[] counts = new int[length];  
    Deque<Integer> stack = new LinkedList<Integer>();  
    for (int i = length - 1; i >= 0; i--) {  
        int height = heights[i];  
        while (!stack.isEmpty()) {  
            int prevHeight = stack.peek();  
            counts[i]++;  
            if (prevHeight <= height)  
                stack.pop();  
            else  
                break;  
        }  
        stack.push(height);  
    }  
    return counts;  
}
```


Problem4. Given an array of integers arr, find the sum of min(b), where b ranges over every (contiguous) subarray of arr.

Example 1:

Input: arr = [3,1,2,4]

Output: 17

Explanation:

Subarrays are [3], [1], [2], [4], [3,1], [1,2], [2,4], [3,1,2], [1,2,4], [3,1,2,4].

Minimums are 3, 1, 2, 4, 1, 1, 2, 1, 1, 1.

Sum is 17.

Example 2:

Input: arr = [11,81,94,43,3]

Output: 444


```
public static int sumSubarrayMins(int[] arr) {
    int n = arr.length;
    int[] left = new int[n]; // Stores the leftmost index for each element
    int[] right = new int[n]; // Stores the rightmost index for each element
    Stack<Integer> stack = new Stack<>();

    // Calculate left boundaries
    for (int i = 0; i < n; i++) {
        while (!stack.isEmpty() && arr[stack.peek()] > arr[i]) {
            stack.pop();
        }
        left[i] = stack.isEmpty() ? -1 : stack.peek();
        stack.push(i);
    }

    // Clear the stack for reusing it
    stack.clear();

    // Calculate right boundaries
    for (int i = n - 1; i >= 0; i--) {
        while (!stack.isEmpty() && arr[stack.peek()] >= arr[i]) {
            stack.pop();
        }
        right[i] = stack.isEmpty() ? n : stack.peek();
        stack.push(i);
    }

    // Calculate the sum of minimums
    long ans = 0;
    int MOD = (int) 1e9 + 7;
    for (int i = 0; i < n; i++) {
        ans += (long) (i - left[i]) * (right[i] - i) % MOD * arr[i] % MOD;
        ans %= MOD;
    }

    return (int) ans;
}
```

About your next assignment...

- Calculate all primes until the number N
- How you'd naively do it? : check each number, with a `checkPrime(N)` method
- Complexity : $O(N^2)$
- What we want you to do? : make it more compact:

In this assignment you will use queues to implement an algorithm to calculate all primes, in order, up to a number n . Here is how that algorithm will work:

- 1) Initialize a queue called **numbers** filled with all of the numbers from 2 (since 1 is technically not prime) up to n . Initialize another empty queue called **primes**.
- 2) Remove the smallest element in **numbers** (the first element in the queue), call this **p**, and add it to the end of **primes**.
- 3) Remove all elements of **numbers** that are divisible by **p**. To do this, remove elements in the front of **numbers** one by one and add them to the end of **numbers** only if **p** does not divide them. If **numbers** is not empty, go back to step 2.
- 4) Print the elements in **primes**.