Recitation 5 - Stacks/Queues

Adi Iyer

24 Feb 2024

Stacks and Queue’s - When to use?

- A lot of parsing questions (valid parentheses, braces, parsing math expressions)
- Basically anything where future data can impact earlier - eg. similarities between 2 arrays - comparing each element is O(N^2) -> better off using a stack to do this
- Monotonically increasing / decreasing stack questions, where you eliminate all entries greater / less than on each iteration.

Q1

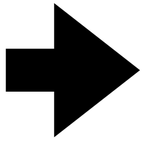
You are given a string s, which contains stars *. In one operation, you can: Choose a star in s. Remove the closest non-star character to its left, as well as remove the star itself. Return the string after all stars have been removed. Note: The input will be generated such that the operation is always possible. It can be shown that the r esulting string will always be unique.

Example 1:

```
Input: s = "leet**cod*e"
Output: "lecoe"
Explanation: Performing the removals from left to right:
- The closest character to the 1st star is 't' in
"leet**cod*e". s becomes "lee*cod*e".
- The closest character to the 2nd star is 'e' in
"lee*cod*e". s becomes "lecod*e".
- The closest character to the 3rd star is 'd' in
"lecod*e". s becomes "lecoe".
There are no more stars, so we return "lecoe".
```

Example 2:

```
Input: s = "erase*****"
Output: ""
Explanation: The entire string is removed, so we return
an empty string.
```



```java
public static String removeStars(String s) {
    Stack<Character> stack = new Stack<>();

    // if * pop, else add
    for (char c : s.toCharArray()) {
        if (c == '*') {
            if (!stack.isEmpty()) {
                stack.pop();
            }
        } else {
            stack.push(c);
        }
    }

    // create string out of it
    int len = stack.size();
    char[] result = new char[len];
    for (int i = len - 1; i >= 0; i--) {
        result[i] = stack.pop();
    }
    return new String(result);
}
```

Q2

Problem2. Implement Queue using Stacks

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and e mpty).

Implement the MyQueue class:

- void push(int x) Pushes element x to the back of the queue.
- int pop() Removes the element from the front of the queue and returns it.

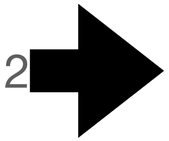
- int peek() Returns the element at the front of the queue.
- boolean empty() Returns true if the queue is empty, false otherwise.

Example 1:

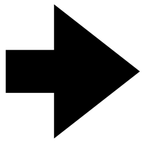
```
Input
["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 1, 1, false]

Explanation
MyQueue myQueue = new MyQueue();
myQueue.push(1); // queue is: [1]
myQueue.push(2); // queue is: [1, 2] (leftmost is front
of the queue)
myQueue.peek(); // return 1
myQueue.pop(); // return 1, queue is [2]
myQueue.empty(); // return false
```

Access last by picking up top of stack 2



Shift ONLY when stack 2 is empty



```java
public class MyQueue {

    private Stack<Integer> stack1 = new Stack<>();
    private Stack<Integer> stack2 = new Stack<>();

    public void push(int x) {
        stack1.push(x);
    }

    public int pop() {
        shiftStacks();
        return stack2.pop();
    }

    public int peek() {
        shiftStacks();
        return stack2.peek();}

    public boolean empty() {
        return stack1.isEmpty() && stack2.isEmpty();
    }

    private void shiftStacks() {
        if (stack2.isEmpty()) {
            while (!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
    }
```

Q3

We are given an array asteroids of integers representing asteroids in a row. For eac h asteroid, the absolute value represents its size, and the sign represents its directio n (positive meaning right, negative meaning left). Each asteroid moves at the same s peed. Find out the state of the asteroids after all collisions. If two asteroids meet, t he smaller one will explode. If both are the same size, both will explode. Two aster oids moving in the same direction will never meet.

Example 1:

```
Input: asteroids = [5,10,-5]
Output: [5,10]
Explanation: The 10 and -5 collide resulting in 10. The
5 and 10 never collide.
```

Example 2:

```
Input: asteroids = [8,-8]
Output: []
Explanation: The 8 and -8 collide exploding each other.
```

Example 3:

```
Input: asteroids = [10,2,-5]
Output: [10]
Explanation: The 2 and -5 collide resulting in -5. The
10 and -5 collide resulting in 10.
```

I- - is the key here

Keep moving forward

```java
public static int[] asteroidCollision(int[] asteroids) {
    Stack<Integer> stack = new Stack<>();
    int trash = 0;
    for (int i = 0; i < asteroids.length; i++) {
        if (asteroids[i] > 0 || stack.isEmpty() || stack.peek() < 0)
            stack.push(asteroids[i]);
        else if (stack.peek() < -asteroids[i]) {trash=stack.pop();i--;}
        else if (-asteroids[i]==stack.peek()) {trash=stack.pop();}//do nothing to stack,
    }
    int[] result = new int[stack.size()];
    for (int l = result.length - 1; l >= 0; l--)
        result[l] = stack.pop();
    return result;
}
```

Q4

Given two integer arrays pushed and popped each with distinct values, return true if this could have been the result of a sequence of push and pop operations on an initially empty stack, or false otherwise.

Example 1:

```
Input: pushed = [1,2,3,4,5], popped = [4,5,3,2,1]
Output: true
Explanation: We might do the following sequence:
push(1), push(2), push(3), push(4),
pop() -> 4,
push(5),
pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1
```

Example 2:

```
Input: pushed = [1,2,3,4,5], popped = [4,3,5,1,2]
Output: false
Explanation: 1 cannot be popped before 2.
```

```java
public static boolean validateStackSequences(int[] pushed, int[] popped) {
    Stack<Integer> stack = new Stack<>();
    int i = 0;
    for (int x : pushed) {
        stack.push(x);
        while (!stack.empty() && i < popped.length && stack.peek() == popped[i]) {
            stack.pop();
            i++;
        }
    }
    return i == popped.length;
}
```

Next Assignment - on Stacks

Out Sunday, you have a week

Look @ uploaded notes for more details..

Postfix Calculator

Introduction

In this lab, the goal is to create a calculator capable of evaluating mathematical expressions. The process consists of two fundamental steps: firstly, converting an infix expression to a post expression, and secondly, evaluating the resultant postfix expression.

In simple terms, an infix expression is the conventional way we write mathematical expressions, where operators like +,-,*, and / are placed between the operands (e.g., 3 + 4 * 5). On the other hand, a postfix expression, also known as Reverse Polish Notation (RPN), places operators after the operands (e.g., 3 4 5 * +). The significant advantage of postfix notation is the elimination of parenthesis and the ambiguity regarding the order of operations, making it a streamlined approach for computer evaluation.

Here's a breakdown of the task at hand:

Converter Class:

- This class is responsible for tacking an infix expression as a string input and converting it to a postfix expression.
- Example: The infix expression is "3 + 4 * 5" should be converted to "3 4 5 * +".

PostfixCalculator Class:

- This class is in charge of evaluating the postfix expression generated by the Converter class.
- Example: Given the postfix expression "3 4 5 * +" , the PostfixCalculator class should evaluate and return the result 23.