CS2040 Tutorial 3

Week 5, starting 5 Sep 2022

Q1 Array- vs Reference-based Queue/Stack?

Does Java have an array-based queue, array-based stack, reference-based queue and reference-based stack? For each of them that exists, which methods does the Java API recommend to be used for stack/queue operations?

Q2 Queue Application – Digging for Gold

There is a long rectangular stretch of land that can be divided into **N** x 1 side-by-side squares. Each square contains 0 or more gold bars and 0 or more rocks. You are allowed to dig up ONE contiguous piece of land in the stretch that has at most **k** (non-negative) rocks (or none if not possible). A square will either be untouched or completely dug up

```
int findMostGoldBars(int[] golds, int[] rocks, int k)
```

Design an algorithm to find the highest number of gold bars that can be dug up, and implement the method above. The algorithm should run in O(**N**) time

e.g. golds = {3, 5, 2, 4, 1}, rocks = {1, 3, 2, 0, 1}

when **k** = 3, the answer is 7

when **k** = 1, the answer is 5

Q3 Stack Application – Expression Evaluation

In the Lisp programming language, each of the four basic arithmetic operators appears before an arbitrary number of operands, which are separated by spaces. The resulting expressions are enclosed in parentheses. There is only one operator in a pair of parentheses. The operators behave as follows:

- `( + a b c )` returns the sum of all the operands, and `( + )` returns 0.
- `( - a b c )` returns a - b - c - … and `( - a )` returns 0 - a.
  The minus operator must have at least one operand.
- `( * a b c )` returns the product of all the operands, and `( * )` returns 1.
- `( / a b c )` returns a / b / c / … and `( / a )` returns 1 / a, using **double division**
  The divide operator must have at least one operand.

You can form larger arithmetic expressions by combining these basic expressions using a fully parenthesized prefix notation. For example, the following is a valid Lisp expression:

```
( + ( - 6 ) ( * 2 3 4 ) )
```

The expression is evaluated successively as follows:

```
( + -6.0 ( * 2.0 3.0 4.0 ) )
( + -6.0 24.0 )
```

18.0

Design and implement an algorithm that uses up to 2 stacks to efficiently evaluate a legal Lisp expression composed of the four basic operators, integer operands, and parentheses. The expression is well formed (i.e. no syntax error), there will always be a space between 2 tokens, and we will not divide by zero.

Output the result, which will be one double value

Question 4 (Online Discussion) – Queue, Stack or Deque?

You are given an array of **N** integers. For every **k** (k > 0) consecutive numbers in the array, report:

a) The leftmost non-zero number out of the **k** numbers
b) The rightmost non-zero number out of the **k** numbers
c) The maximum of the **k** numbers

Solve each part independent of the other parts, in O(**N**) time
e.g. if the numbers are {0, 1, 0, 0, 2, 0, 4, 0, 1, 0, 3, 1, 0, 2} and **k** = 4, then outputs are:

a) 1 1 2 2 2 4 4 1 1 3 3
b) 1 2 2 4 4 1 1 3 1 1 2
c) 1 2 2 4 4 4 4 3 3 3 3

How do you tell whether a Queue, Stack or Deque can help you in each part?