COP-3530 Data Structures,

Programming Assignment 1:
Running Time Calculation, Lists, Stacks and Queues

Due Date: June 03 at 11:59 PM

This assignment has three parts: 1) Implementing a queue data structure using a circular array, 2) implementing a queue data structure using two stacks, and 3) comparing the running times of "enqueue" and "dequeue" operations of both implementations in two given scenarios.

1 Constructing CAQueue: a Queue using a Circular Array

As mentioned in the the slides, you can implement a queue using a circular array. In this implementation, *CAQueue* is a Java class with two key methods: *enqueue* and *dequeue* and two main private instance fields *front* and *back* storing the indices of front and back objects in the circular array. You must use an object of type *java.util.ArrayList* as the underlying data structure which stores the objects in the queue.

2 Constructing SSQueue: a Queue using two Stacks

In this part, you need to first implement the java class *MyStack* and its push and pop methods to store/remove objects in a First-In-Last-Out manner to/from an object of type *java.util.ArrayList*. Then, you should implement the class *SSQueue* and its enqueue and dequeue operations using two stacks of type *MyStack* in the following way:

- The enqueue operation can be done by simply pushing the explicit input parameter of the method in the first stack.
- The dequeue operation pops an element from the second stack. If the second stack is empty, but the first one is not empty, you need to transfer *all* of the elements stored in the first stack to the second one. After emptying the first stack, you simply pop one element from the second stack to complete the dequeue operation.

3 Efficiency Comparison of two Implemented Queues

In this part, you need to write a program that compares the efficiency of the queue implementations in parts 1 and 2. To do so, you need to find and compare the running times of the following two scenarios for both queue implementations:

3.1 Scenario 1: Alternating Sequence of Enqueues and Dequeues

For every $n \in \{20, 50, 100, 1000, 10000, 100000, 1000000\}$, do the following:

1. add the first n non-negative integers to the queue.
2. long startTime = System.nanoTime();
3. for every $i \in \{0, 1, \ldots, n-1\}$, do the following:
   (a) enqueue($i + n$)
   (b) dequeue()
4. long endTime = System.nanoTime();
5. long totalTime = endTime - startTime;
6. System.out.println(n + ", " + totalTime);

3.2 Scenario 2: Random Sequence of Enqeueus and Dequeues

For every $n \in \{20, 50, 100, 1000, 10000, 100000, 1000000\}$, do the following:

1. add the first n non-negative integers to the queue.
2. long startTime = System.nanoTime();
3. for every $i \in \{0, 1, \ldots, n-1\}$, do the following:
   (a) if (Math.random() $<$ 0.5) enqueue($i + n$);
   (b) else dequeue();
4. long endTime = System.nanoTime();
5. long totalTime = endTime - startTime;
6. System.out.println(n + ", " + totalTime);

4 Submissions

You need to submit a *.zip* file compressing the following folder:

- all the Java source file(s).
- An image showing the plot of running times (in nanoseconds) of scenario 1 vs. n when CAQueue is applied (x-axis is n and y-axis is the time).

- An image showing the plot of running times (in nanoseconds) of scenario 1 vs. n when SSQueue is applied.
- An image showing the plot of running times (in nanoseconds) of scenario 2 vs. n when CAQueue is applied.
- An image showing the plot of running times (in nanoseconds) of scenario 2 vs. n when SSQueue is applied.

Please note that you can place some or all the four aforementioned plots in one coordinate system.