CMPSC 112
Introduction to Computer Science II
Fall 2016

Laboratory Assignment Ten: A Queue-Based Solution to the Josephus Problem

Introduction

Many real-world programs use the queue abstract data type (ADT) to solve a problem. For instance, as discussed in class, an operating system may use a round-robin scheduler that leverages a queue to determine which process it should execute next. In this laboratory assignment, you will use a linked-list-based queue, called the `NodeQueue`, to run a "simulator" that can "solve" the Josephus problem. Then, you will analyze the source-code of the simulator to determine its worst-case time complexity. Finally, you will extend the solver so that it accepts command-line arguments and then use this enhanced version to conduct a comprehensive study of the method's time efficiency.

Learning About the Queue Abstract Data Type

Before you start to extend the implementation of the `JosephusSolver`, you should take some time to learn more about the queue ADT described in Chapter 6. What are the methods that the queue provides? What is the difference between an `ArrayQueue` and a `NodeQueue`? What is an example of a problem that can be solved using the queue? Your response to the previous question should include a brief introduction to the problem and a discussion of how the queue is incorporated into the solution. For instance, you could explain a program that you have previously implemented that manipulated a queue to solve a problem. Alternatively, you can study books, articles, and online sources to investigate a well-known problem that can be solved through the use of a queue. Please see the instructor if you have questions about the tasks for this part of the assignment.

Using Simulation to Solve the Josephus Problem

Many children have played the game of "hot potato" that requires them to sit in a circle and pass an object around the circle until a leader rings a bell or stops playing some music. At this point in the game, the child holding the potato must stop playing the game and, after the remaining children move closer together in the circle, the game continues in the same fashion. This game play continues until there is only one child left, who is then declared the winner.

It turns out that this "game" was first recorded by Flavius Josephus, a first century Jewish historian who was a solider in a Jewish-Roman war. It is possible to use a queue to "simulate" the process of playing this game and then quickly determine who will win. After typing the command `git pull` in the `cs112F2016-share` repository, you will see that there is now a `labs/lab10/` directory that contains a `build.xml` file, a `net-datastructures-5-0.jar` Java archive, and `JosephusSolver.java` file in the `src/edu/allegheny/solver/` directory. You can run this program by typing "`ant JosephusSolver`" in the terminal. What output do you see?

If you study `JosephusSolver.java` you will notice that it contains a `main` method that creates different groups of children and then runs the solver by specifying a group of children and a stopping point for each round. While this approach is suitable for demonstrating the use of the solver, it

is not ideal if you need to run a series of experiments to evaluate the efficiency of this technique. Therefore, you should modify the program so that it accepts two command-line arguments: the number of children involved in the simulated game and the number of steps for which each round must be run. Next, you should enhance the program so that it automatically generates a list of "children" and calls the solver with both an instance of the child-populated `NodeQueue` and the steps parameter. Finally, you should add timing code to the program that can calculate how long it takes to run the solver. What is the program's output? How did you determine it worked correctly?

Evaluating the Efficiency of the Solver

After you have finished enhancing the `JosephusSolver` by adding command lines and timing code, you should carefully study the method called `Josephus`. Using the big-O notation, what is the worst-case time complexity of this method? In addition to stating your chosen complexity, you should clearly justify your choice. Does this analysis suggest that this solver is efficient or inefficient?

Finally, you should conduct a doubling-based empirical study to evaluate the run-time efficiency of the `JosephusSolver`. To develop an accurate understanding of the solver's performance, you should pick different values for `k` and run the solver with different groups of children of increasing size. For each value of `k` and size of the group, you should run the solver ten times and calculate the arithmetic mean of the execution times. Make sure that you pick values for `k` and the size of the group that will give an accurate picture of worst-case time efficiency. Next, you should create a table of data or a graph that summarizes your results. Does the data from your experimental evaluation confirm the worst-case time complexity that you derived? Why or why not?

Summary of the Required Deliverables

This assignment invites you to submit a signed and printed version of the following deliverables:

1. A description of a problem that can be solved by using the queue abstract data type.
2. The fully commented version of `JosephusSolver.java` and any other Java files you create.
3. The output from five separate runs of `JosephusSolver`, demonstrating its correctness.
4. A written report that provides a response to all of the questions posed in this assignment.
5. A justified statement of the worst-case time complexity for the `Josephus` method.
6. A written analysis of the `JosephusSolver`'s efficiency when it is run in different configurations.
7. A reflective commentary on the challenges that you faced when completing this assignment.

Along with turning in a printed version of these deliverables, you should ensure that everything is also available in the repository that is named according to the convention `cs112F2016-<your user name>`. Please note that students in the class are responsible for completing and submitting their own version of this assignment. While it is acceptable for members of this class to have high-level conversations, you should not share source code or full command lines with your classmates. Deliverables that are nearly identical to the work of others will be taken as evidence of violating the Honor Code. Please see the instructor if you have questions about the policies for this assignment.