Assignment: Staying Hydrated During Hakofos

Avraham Leff

September 23, 2022

1 Assignment-Specific Packaging

The general packaging is unchanged from the basic "Homework Requirements" (see slides from first lecture and "Homework Policies for COM 2545" on Piazza).

This assignment's "DIR" **must be named** *HydratedHakofos*, and your application's Java class **must be named** *HydratedHakofos.java*. Your write-up file **must be named** *HydratedHakofos.pdf*.

2 Motivation

This assignment is intended to give you practice coding an algorithm and explaining its order of growth.

3 Background

Your favorite synagogue is really, really, large: as people perform *hakofot* (or *"hakofos"*) people have been known to get dehydrated and even faint! In response, the synagogue's board has decided to designate tables that dispense water along the circular track taken by worshipers.

Specifically, there are n tables (**tables are labeled 1..n**) , arranged on a circle around the synagogue. Crowd control mandates that you can only move in a single direction from table to table; the board has decided that all travel must proceed in the clockwise direction. The idea is that worshipers begin at $table_1$, pick up a drink that's supposed to get them to (at least) $table_{i+1}$, and then travel to the next table where they can get their next drink of water and thus stay hydrated.

So far, so good. The problem is that the "board being the board", the tables are not equidistant from one another, so $table_i$ may dispense a *different amount of water* than $table_j$. To make things even more complicated, the *amount of water needed* to make it to the next table may also differ from table to table.

3.1 The Problem

The synagogue approaches you with the following problem: *"at which table should worshipers begin their hakofos so they can traverse the entire circuit without getting dehydrated."*

1. Worshipers begin their circuit with no water.
2. At $table_i$, they can pick up $availableWater_i$ cups of water.
3. We also know that $waterRequired_i$ cups of water are required to travel from $table_i$ to $table_{i+1}$ without fainting.
4. Worshipers are allowed to accumulate a "water surplus" if the amount of available water at $table_i$ exceeds the amount of water required to reach $table_{i+1}$.

They hire you to devise an algorithm that determines

- Is there **any table** from which worshipers can begin hakofos such that they complete successfully (i.e., worshipers never have a negative amount of water (a "zero level" is OK)).
- If beginning at a given table **does result** in a successful hakofos, return the **lowest numbered table** from the set of such possible tables.

4 Requirements

Your implementation must exhibit an $O(n)$ order of growth. In addition, it must "by the clock" perform well. Because the latter criterion is hard to define across computers, I'll only place a small weight on it. Qualitatively: the implementation should appear "fast" even on large input such as $n = 2^{26}$.

Aside from using the JDK, for this assignment, you may **only use code written by yourself**!

The assignment consists of both "programming" and "non-programming" components.

4.1 Non-Programming

This portion of the assignment is worth 20%.

Your writeup must contain, in this order:

1. In no more than three paragraphs, explain how your implementation meets the required order-of-growth.

Your explanation must sketch your design in sufficient detail that your "Big-O" claims follows naturally. Don't give me the code since I can see that for myself! Pseudo-code is the right level of detail. If appropriate, consider incorporating a diagram into your discussion.

This component will be graded on your *"communication skills"*: clarity, correctness, succinctness, and "convincingness".

You may "explain by reference" to material covered in lecture or textbook or to "well-known" CS-knowledge. If you choose to use a "reference to covered material", you **must offer a convincing explanation** that this material is relevant to solving our problem.

4.2 Programming

This portion of the assignment is worth 80%, and will be graded for "correctness" and ability to meet the "Big-O" requirements at scale.

Please review the general requirements for a programming assignment! I've tried to reduce the chances of "mistakes" occurring through the use of a "skeleton interface", but ultimately, **you are responsible** for ensuring that I can compile and test your code without incident.

1. Begin by downloading a skeleton `HydratedHakofos.java` class from this git repository.
2. Then, implement the stubbed methods of *HydratedHakofos*.