Project 1: The Josephus Problem

1 The Problem

This project is to warm you up about Java programming after this winter break. Later on, we will come back to this problem to provide an alternative solution, based on a more sophisticated data structure.

A story goes like this: After a fierce battle, some soldiers were caught by the enemy, who decided to hang all of those soldiers except one, who would be sent back as a messenger ☹.

More specifically, starting with the first among \( n \) soldiers sitting around a circle, numbered from 1 to \( n \), every \( k \)th soldier, \( k \in [2, n] \), will be hanged except the last one, who would be the survivor. For example, with four soldiers, and \( k = 2 \), the second, the fourth, and finally, the third, will be hanged, in this order, but the first will be the spared.

It was said that a smart soldier thought of a way to avoid being killed ☻. The question is where did he sit to avoid being hanged?

The more general problem is to determine the survivor number, \( J(n, k) \), i.e., where you should sit so that you will always be the survivor, if you are one of \( n \) people sitting around a circle, and every \( k \)th person will be killed, starting from the first one?

Below are a few values for \( J(n, 2) \).

<table>
<thead>
<tr>
<th>( n )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( J(n, 2) )</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

To echo the above example, for \( n = 4, k = 2 \), starting with 1, the order of the “killing” is \((2, 4, 3)\), and the survivor number, \( J(4, 2) \), is 1.

2 What to do?

Write a program, based on the Java Array type, that contains a method \( J(n, k) \), among others, which is to calculate the survivor number, \( J(n, k) \). As sample outputs, your program has to

1. determine where you should sit, if you are among 40 people for whom all but the last one will be hanged, and every other person \( (k = 2) \), will be hanged. You also have to figure out the order in which the 39 poor guys will be chosen;

2. determine \( J(n, 2) \) for \( n = 2 \) up to \( n = 100 \). Make sure that you will verify your result with the above initial values for \( n \in [1, 6] \); and

3. for \( k = 3, 4, \) and 12, calculate \( J(n, k) \) for \( n = 10, 50, \) and 100.
3 Is there a different/better way to do it?

As we mentioned in the syllabus, what we will do in this course is to find a “best” solution to a problem. Later on, we will see how to use a different data structure, *doubly linked circular linked list*, to solve this Josephus problem.

By a “better” solution, we often mean a solution that takes less time and/or less space. The time it takes to use a *Java Array type* to solve this problem necessarily takes \( \Theta(n) \) to finish, i.e., in proportional to \( n \).

For more discussion of this Josephus’ problem, as well as a \( \Theta(1) \) (constant time) solution for \( J(n, 2) \), please check out [1, § 1.3], or check out the link in the project page. Send in a short summary for this alternative, and more efficient, approach.

4 What to send in?

Email me your entire source code in .java, together with a lab report, containing answers to the questions as posed in Sections 2 and 3.

References