The Josephus Problem

Dr. Andrew Rosen

Due Feb 25th at 6am

Abstract

In this lab you will solve the Josephus Problem by implementing the Circular Linked List provided to you. This text is adapted from Programming Project 5 of Chapter 2 in Koffman and Wolfgang.

1 The Josephus Problem

The Josephus problem is named after the historian Titus Flavius Josephus (37 - 100 CE). Josephus participated in the Jewish uprising against the Roman Empire. This revolt ended exactly as other attempts to stand against the might of Rome did – poorly.

During the siege of Yodfat, Josephus and forty other soldiers were about to be captured by the Romans. They decided they would rather die than become Roman slaves and committed suicide by drawing lots. Josephus and one other soldier remained in the end, decided that a life under the Romans as a slave was better than no life, and surrendered. Josephus became a slave and gained his freedom a few years later. He later went on to write a number of documents which give insight to the religions of that time period.

While the exact nature of the lot-drawing process is unknown, the following approach is generally thought to be how it's done.

1.1 The Problem

Soldiers form a circle and count around the circle some predetermined number. When this number is reached, that person receives a lot and leaves the circle. The count starts over with the next person.

Build a Circular Linked List with the provided code to represent the circle of people and simulate the problem. Your program should use two parameters, n people and k, where k is the number of counts. In other words, we successively remove the kth person from the list.

If you need more information, Wikipedia has a pretty decent article.

2 The Code

To complete the task, you need to fill in the missing code. I've included code to create an `Iterator`.

An `Iterator` is an object that iterates over another object – in this case, a circular linked list. You can use the `.next()` method to advance the `Iterator` to the next item (the first time you call it, the iterator will travel to node at index 0). Using iterator's `.remove()` removes the node the iterator is currently at.

Say that we had a `CircularLinkedList` that looked like this:

```
A ==> B ==> C ==> D ==> E ==>
```

Calling `.next()` three times will advance the iterator to index 2. Calling `.remove()` once will remove the node at index 2.

```
A ==> B ==> D ==> E ==>
```

Calling `.remove()` once more will remove the node now at index 2.

```
A ==> B ==> E ==>
```

The `Iterator` methods handle wrapping around the `CircularLinkedList`. **Be sure to create the iterator using `l.iterator()` and after you've added all the nodes to the list**, where `l` is your `CircularLinkedList`.

3 Examples

Here are examples of rings with n people and every kth person is removed from the ring.
For a ring of $n = 5$ and the count $k = 2$:[1]

```
1 ==> 2 ==> 3 ==> 4 ==> 5 ==>
1 ==> 3 ==> 4 ==> 5 ==>
1 ==> 3 ==> 5 ==>
3 ==> 5 ==>
3
```

For a ring of $n = 13$ and $k = 2$

```
1 ==> 2 ==> 3 ==> 4 ==> 5 ==> 6 ==> 7 ==> 8 ==> 9 ==> 10 ==> 11 ==> 12 ==> 13 ==>
1 ==> 3 ==> 4 ==> 5 ==> 6 ==> 7 ==> 8 ==> 9 ==> 10 ==> 11 ==> 12 ==> 13 ==>
1 ==> 3 ==> 5 ==> 6 ==> 7 ==> 8 ==> 9 ==> 10 ==> 11 ==> 12 ==> 13 ==>
1 ==> 3 ==> 5 ==> 7 ==> 8 ==> 9 ==> 10 ==> 11 ==> 12 ==> 13 ==>
1 ==> 3 ==> 5 ==> 7 ==> 9 ==> 10 ==> 11 ==> 12 ==> 13 ==>
1 ==> 3 ==> 5 ==> 7 ==> 9 ==> 11 ==> 12 ==> 13 ==>
1 ==> 3 ==> 5 ==> 7 ==> 9 ==> 11 ==> 13 ==>
3 ==> 5 ==> 7 ==> 9 ==> 11 ==> 13 ==>
3 ==> 7 ==> 9 ==> 11 ==> 13 ==>
3 ==> 7 ==> 11 ==> 13 ==>
3 ==> 7 ==> 11 ==>
3 ==> 11 ==>
11
```

[1] We are starting with 1 as opposed to 0.