COMP 2140 Lab 2 — Recursion and Linked Lists

Helen Cameron

Week of October 17, 2016

Objective

To write both recursive and iterative methods to decide if a linked list contains more than n items.

Exercise

File `Lab02.java` contains a nearly-complete application that builds several linked lists of numbers, then tests if the linked lists contains more than n numbers (for various values of n). It times how long each test takes, and prints out the timings and the results returned. You are asked to complete the `hasMoreThanRecursive` method and the `hasMoreThanIterative` method (more information below). You are not permitted to change anything else in `Lab02.java`.

The `LinkedList` class and its `Node` class: The linked list class consists of a pointer (called `top`) to the dummy header node in the list — the list has a dummy header and a dummy trailer.

Inside the linked list class is a `private Node` class with `public` instance members, `item` and `next`. Because the `Node` class is `private` inside the `LinkedList` class, no code outside the `LinkedList` class can access or know about `Node`s. Because `item` and `next` are `public`, you can access the `item` and `next` of any node anywhere inside the `LinkedList` class. (This structure is not good object-oriented practice. However, it is very simple and will allow you to transfer your knowledge to non-object-oriented languages easily.)

The methods you must write: Each of method `hasMoreThanRecursive(int n)` and method `hasMoreThanIterative(int n)` consists of two parts:

1. A helper method in the `Node` class that returns `true` when a non-empty list contains more than n numbers and `false` otherwise. This part is what you will complete using the headers I have provided (more details below).
2. A `public` driver method in the `LinkedList` class. This part is already written for you. Its task is to correctly handle the case when the list is empty (simply return true if $n < 0$ and `false` if $n \geq 0$), and, when the list is NOT empty, to call the helper method in the `Node` class on the first non-dummy `Node` and to return whatever the helper method returns.

Here's what the helper methods should do:

The iterative helper method in the `Node` class: Method `hasMoreThanIterative(int n)` is called on the first non-dummy `Node` in the list — that is, it is passed a pointer to the first non-dummy `Node` in the list in the implicit parameter `this`. It must use a `while`-loop to go through all the `Node`s in the list until it has either seen more than n non-dummy nodes (its result should be `true`) or reaches the end of the list without seeing more than n non-dummy nodes (its result should be `false`).

The recursive helper method in the `Node` class: Method `hasMoreThanRecursive(int n)` is called on some non-dummy `Node` in the list — that is, it is passed a pointer to some non-dummy `Node` in the list in the implicit parameter `this`. Its task is to return `true` if there are more than n non-dummy `Node`s from `this` `Node` to the end of the list (including `this`).

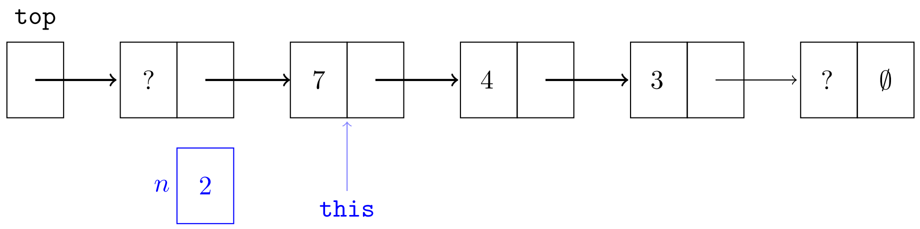
Base case 1: If $n < 1$, then return `true` — since `this` is at least one non-dummy `Node` in the list, we have more than n non-dummy `Node`s.

The recursive case: If $n \geq 1$ and there are more non-dummy `Node`s after `this`, the method should make a recursive call to find out if there are more than $n - 1$ non-dummy `Node`s after `this`:

- If there are more than $n - 1$ non-dummy `Node`s after `this` — that is, if the recursive call returns `true` —, then return `true` because there are more than n non-dummy `Node`s if you include `this`;
- Otherwise, if there are NOT more than $n - 1$ non-dummy `Node`s after `this` — that is, if the recursive call returns `false` —, then there are NOT more than n non-dummy `Node`s if you include `this`, so return `false`.

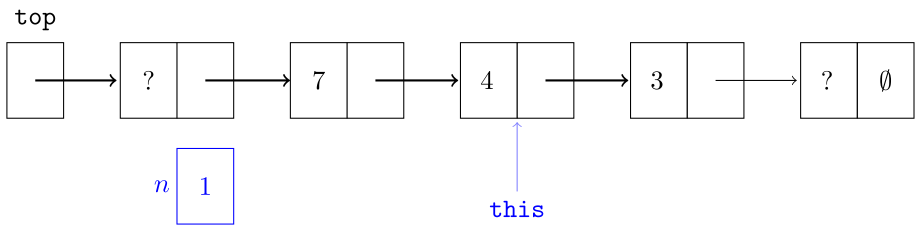
Base case 2: If $n \geq 1$ and there are no non-dummy `Node`s after `this`, return `false`.

The recursive method illustrated: Suppose n is 2 in the public driver method. The first call to helper method `hasMoreThanRecursive` has $n = 2$ and its `this` pointing to the first non-dummy `Node`:



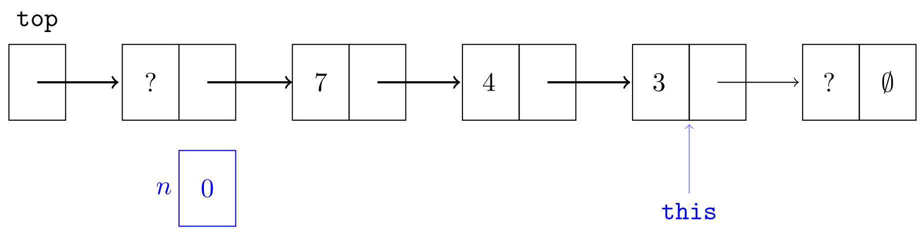
The first call makes a recursive call, since $n > 1$ and there is at least one non-dummy `Node` after its `this`. It passes $n - 1 = 1$ to the recursive call.

The second call to method `hasMoreThanRecursive` has its `this` pointing to the second non-dummy `Node`:



The second call makes a recursive call, since $n \geq 1$ and there is at least one non-dummy `Node` after its `this`.

The third call to method `hasMoreThanRecursive` has $n = 0$ and its `this` pointing to the third non-dummy `Node`:



The third call realizes that it is a base case because $n < 1$ and it knows that there is at least one non-dummy `Node` (`this`). Thus, it simply returns `true`.

The second call returns `true` (what it received from the third call).

Then the first call returns `true` (what it received form the second call). Since $n = 2$ and there are more than two non-dummy `Node`s in the list, it returns the correct result.

Reminder: Each method should have only ONE `return` statement. You should not have any `break` or `continue` statements.