UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS212: Practical 2

Release: Thursday 21 February 2019, 18:00
Deadline: Friday 22 February 2019, 18:00

Objectives

This practical has the following objectives:

- To learn how to use recursion for navigating a linked list data structure
- To learn how to implement a self organising list

Instructions

Complete the task below. Certain classes have been provided for you in the *files* zip archive of the practical. You have also been given a main file which will test some code functionality, but it is by no means intended to provide extensive test coverage. You are encouraged to edit this file and test your code more thoroughly. Remember to test boundary cases. Upload **only** the given source files with your changes in a zip archive before the deadline. Please comment your name **and** student number in at the top of each file.

Task 1: Self Organising List [23]

A self-organising list reorganises elements as they are accessed in an attempt to improve search efficiency. If certain elements are accessed more frequently than other elements, then it would be more efficient to move those elements closer to the head of the list. Common methods used by self-organising lists include the *Move-to-front method*, *Transpose method*, *Count method* and *Ordering method*. Refer to section 3.5 in the textbook. You will be required to implement a self-organising list using the *Transpose method*.

You have been given a partially implemented Self-organising list class and a node class to use. Your task is to implement the following methods according to the given specification:

`void insert(Integer key)`

Create a new node with the given key and insert it to the back of the list. The `calculateData()` method should be used to update all the node's `data` fields after an insert, as described below. If the key already exists, then do nothing.

`Boolean contains(Integer key)`

If a node exists with the given key, return `true`, otherwise return `false`.

`void Delete(Integer key)`

Remove the node with the given key. `calculateData()` should be called to update the `data` fields of all nodes. See description below. If the key does not exist, do nothing.

`void calculateData()`

This function is fully provided for you. Do not modify it.

`Integer getData(Integer key)`

Retrieve the data field for the node with the given key. The node should then be swapped with it's predecessor, unless it is the root. Since the data field depends on the order of the list, `calculateData()` needs to be called again. The value returned should be the node's `data` field before it was moved. If the key does not exist, return null.

Recursive Calculation and Traversal of Linked List

The `ListNode` class contains 3 fields:

- `Integer key` - The key used to search for the node
- `Integer data` - Data value generated using a recursive function
- `ListNode next` - The next node in the list. `null` indicates the end of the list.

The data field is to be calculated using the two recursive functions as follows:

$$dataRec(node) = \begin{cases} \text{node.key} & \text{if last node,} \\ \text{(node.key * sumRec(node)) - dataRec(node.next)} & \text{otherwise.} \end{cases}$$

$$sumRec(node) = \begin{cases} \text{node.key} & \text{if last node,} \\ \text{sumRec(node.next) + node.key} & \text{otherwise.} \end{cases}$$

These functions should be implemented in the following methods:

`public static int dataRec(ListNode node)`

Recursively calculate the `data` value for the given node, setting the `data` field to the calculated value before returning it.

`public static int sumRec(ListNode node)`

Recursively calculate the sum of `key`s starting with the given node to the end of the list.

You may use your own helper functions to assist in implementing the specification. However you may not modify any of the given method signatures.

Submission

You need to submit your source files on the Assignment website (assignments.cs.up.ac.za). All methods need to be implemented (or at least stubbed) before submission. Place all the source files including a makefile in a zip or tar/gzip archive named uXXXXXXXX.zip or uXXXXXXXX.tar.gz where XXXXXXXX is your student number. You have 24 hours to finish this practical, regardless of which practical session you attend. Upload your archive to the *2019 Prac 2 - Friday* slot on the Assignment website. Submit your work before the deadline. **No late submissions will be accepted!**