Computer Science 3A

Practical Assignment 8

20 April 2023

Time: 20 April 2023 — 17h00 Marks: 50

Practical assignments must be uploaded to `eve.uj.ac.za` **before** 17h00. Late submissions **will not be accepted**, and will therefore not be marked. You are **not allowed to collaborate** with any other student.

Heap-based term frequency indexing for improved keyword searching
One key aspect of natural language processing is the ability to represent text in a more succinct manner. Modern AI approaches this problem with large language models such as GPT-4 in order to create an embedding that encapsulates all the nuance present in language. However, if we go back to the basics the earlier days used term frequency as a representation. In order to achieve this we tokenize the text (split it up into words or n-grams) and then keep track how often certain terms come up and leverage that as the representation. This week's practical is centered around this. Heaps allow us then to sort these term frequencies in an efficient manner.

On the basic level, a heap is simply a binary tree where the following conditions have been placed on the tree:

- key(v) $\geq$ key(parent(v))

If the following key property holds then **heap-order** has been maintained. The other condition that we are dealing with is the **complete binary tree** property.

Insertion from a heap always occurs in the **last node**, this is the rightmost node on the lowest height level. When items are removed from the heap, the items are removed from the root. In order to maintain the binary tree as a complete binary tree, the item in the last node is swapped with the entry in the root, and then the last node is removed. In both cases, in order to maintain heap order, an `upheap` or `downheap` operation must be performed.

In the case of a heap implementation, all heaps that store more than one value must use **sentinel nodes** as the leaf nodes. This means that the definition for **isExternal** must be modified in some cases.

Part of creating the heap is locating the last node. This must be done in two places, when an item is inserted and when an item is removed.

The following algorithm can be used to locate the last node on insert:

1. If the current last node is a **left child** then the sibling on the current last node is the new last node.
2. If not then traverse up the tree until you locate a node that is a **left child** or the **root of the tree** maintain this as the **current node**
3. If the current node is **a left child** then the current node becomes the sibling of this node.
4. Traverse down the **left branch** of the tree from the current node until you reach an **External node**. This should be a sentinel node.
5. The current node should then be returned as the new last node.

When you are removing an item from the heap, the reverse of the operation is performed.

You must complete the following classes marked by:

```
//TODO: COMPLETE CODE HERE
```

You have been provided with the necessary classes that you can use to test the execution of your solution, the results of your test program should look as follows:

The following files must be submitted to EVE:

1. *studentnumber*_p8.zip

Marksheet

1. Heap: upheap [7]
2. Heap: downheap [8]
3. Heap: getLastNodeInsert [5]
4. Main: readTokens [10]
5. Main: buildHeap [10]
6. Compilation and Correct execution. [10]