Assignment 6

**Valeria Fionda, Mouna Kacimi,
Werner Nutt, Simon Razniewski**

Priority Queues and Hashing

Instructions: Your assignment should represent your own effort. However, you are not expected to work alone. It is fine to discuss the exercises and try to find solutions together, but each student shall write down and submit his/her solutions separately. It is good academic standard to acknowledge collaborators, so if you worked together with other students, please list their names.
Be prepared to present your solution at the lab. If you are not able to explain your solution, this will be considered as if you had not done your work at all.
You can write up your answers by hand (provided your handwriting is legible) or use a word processing system like Latex or Word. Experience shows that Word is in general difficult to use for this kind of task.
For a programming task, your solution must contain (i) an explanation of your solution to the problem, (ii) the Java code, in a form that we can run it, (iii) instructions how to run it. Also put the source code into your solution document. For all programming tasks, it is not allowed to use any external libraries ("import") or advanced built-in API functions (for example, `String.indexof("a")` or `String.substring(1,5)`), if not stated otherwise.
Please, include name, matriculation number and email address in your submission.

1. Priority Queues

In the lecture, we have introduced priority queues as an abstract data type that supports the operations

- Insert
- ExtractMax.

With Insert, one inserts a new value into the queue (or, more generally, an object one of whose attributes is the key attribute for the queue). ExtractMax returns the maximal value currently in the queue and deletes that value from the queue. (More generally, it returns an object with the maximal value and deletes that object from the queue.)

One way to realize priority queues, is to use heaps based on arrays. In this exercise, you are asked to realize a priority queue with binary trees. A priority queue element has a point "root" to an element of class Node. Nodes themselves are instances of the classe Node defined as follows:

```
class Node{
   int key;
   Node left;
   Node right;
   Node parent;
   int lcount; // Number of nodes in the left subtree
   int rcount; // Number of nodes in the right subtree}
```

Using the above data structure, implement priority queues as binary trees that have the heap property. Your implementation should support the following methods:

1. `bool IsEmpty()`
2. `void Insert(int value)`: inserts a value in the queue.
   Hints: Insertion should keep, as far as possible, the binary tree balanced. Each node holds information about how many nodes are there in its right and left subtrees. You can use this information to put a new node into the subtree that has fewer elements. Clearly, when inserting a new value in a new node, you also have to update these counters. If you have found out where to insert the new node in your tree as a leaf, the insertion of the new value may violate the tree property. Use the technique explained in the lecture to maintain.
3. `int ExtractMax()`: returns the maximum in the queue and deletes it.
   Hints: The maximum value in a heap is in the root. If you delete the root, this leaves a hole that needs to be refilled. To do that, find a leaf to be dropped, put its key value into the root node, and delete the leaf. The new value in the root may violate the heap property. Use the technique from the algorithm "heapify" to correct such violations.

Describe your implementation in a separate document from the code. Test it and report on the tests.

(10 Points)

2. Optimizing Chaining

Instead of using lists for storing entries with the same hash values, it is also possible to use binary search trees.

1. What are in this case the worst-case running times for Insert, Find and Delete? Do they improve?

2. What are in this case the average running times for Insert, Find and Delete? Do they improve?

(5 Points)

3. Hashing for Word Counting

Counting different words is a basic task in linguistic analysis. For example in the first two lines of this exercise, there are 27 words, of which 23 are different.

1. How can you effectively count the number of different words in a text using a hash table?

2. Implement a hash table of your choice (with open addressing or chaining) with 2^{14} cells and test it with the following three hash functions for words:

   - String.hashcode()
   - String.substring(0,4).hashcode()
   - a hash function chosen by yourself, that may only use the single characters of the word (you may use the function Character.hashcode())

   on the following three texts:

   - the book Genesis[1]
   - Romeo and Juliet by Shakespeare[2]
   - the European regulation about banana quality standards[3].

3. For each text, count how many different words are in the text. Also count the percentage of different words wrt. the total number of words in each text (e.g., 6.000 different words in 8.000 words in total $= 75\%$). Also count the number of collisions for each hash function and explain differences that you find. How well did your own hash function perform?

---

[1] `http://en.wikisource.org/wiki/Bible_%28King_James%29/Genesis`

[2] `http://en.wikisource.org/wiki/The_Tragedy_of_Romeo_and_Juliet`

[3] `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31994R2257:EN:HTML`

Hints:

- It is advisable to copy the texts from Wikisource into textfiles, then read the textfiles linewise in Java (using classes such as BufferedReader, InputStreamReader or similar)
- To identify the words in the text, you may use the Java class StringTokenizer
- Ignore sentence delimiters (“.”, “,”, “\” , “!”, “?”, “;”), numbers and other special characters.
- Also ignore cases, you may use the function `String.toLowerCase()` for that purpose.

(15 Points)

Submission: Until Wed, 22 May 2013, 11:59 pm, to

`dsa-submissions AT inf DOT unibz DOT it.`

Submit your work in two files, one PDF document and one .tar or .jar file with your code.