University of Windsor

School of Computer Science

60-654 Advanced Computing Concepts - Fall 2017
Lab Assignment 1

Deadline: **October 2, 2017 at 11:59pm**

Rules: This assignment must be submitted and will be evaluated **individually**. You are allowed to discuss about the problems and some ideas within a group or with other students. However, the source code for the programs should not be identical among those students.

You are required to use Java SE 8 and Eclipse to write your programs, and the data structures and implementations as discussed in class.

Objectives: The aim of this assignment is to help understand the main principles of hash tables and different types of search trees. Students will obtain hands on experience in working with hash tables and search trees and doing some experiments on them. A real application (optional) includes implementing some components of a simple Web search engine, including Web page dictionary, ranking and spellchecking by using hash tables, heaps and search trees, and the use of Java SE 8 and Eclipse.

Tasks:

1. Within a Java class, write a method that creates n random strings and inserts them in a hash table. The method should compute the average time for each insertion.
2. Write another method that finds n random strings in the hash table. The method should delete the string if found. It should also compute the average time of each search.
3. Repeat #1 and #2 with $n = 2^i$, $i = 1, \ldots, 20$. Place the numbers in a table and compare the results for Cuckoo, QuadraticProbing and SeparateChaining. Comment on the times obtained and compare them with the complexities as discussed in class.
4. Use the Java classes BinarySearchTree, AVLTree, RedBlackBST, SplayTree given in class. For each tree:
   a. Insert 100,000 integer keys, from 1 to 100,000 (in that order). Find the average time for each insertion.
   b. Do 100,000 searches of random integer keys between 1 and 100,000. Find the average time of each search.
   c. Delete all the keys in the trees, starting from 100,000 down to 1 (in that order). Find the average time of each deletion.
5. For each tree:
   a. Insert 100,000 keys between 1 and 100,000. Find the average time of each search.
   b. Repeat #4.b.
   c. Repeat #4.c but with random keys between 1 and 100,000. Note that not all the keys may be found in the tree.
6. Draw a table that contains all the average times found in #4 and #5. Comment on the results obtained and compare them with the worst-case and average-case running times of each operation for each tree. Which tree will you use in your implementations for real problems? Note: you decide on the format of the table (use your creativity to present the results in the best possible way).

Submission:

1. You must submit: (i) a report (in PDF or word), which contains the answers to all questions, (ii) all java files and classes needed to run your programs. Your programs should run in Java SE 8 in the Eclipse IDE. You should upload all this files to the Blackboard, as a **single** zip file.
2. In your report, you must provide written answers to all questions, including the programs' outputs, as required. Marks will be deducted if comments/explanations are missing.
3. Add the following note at the beginning of your report:
   "I confirm that I will keep the content of this assignment confidential. I confirm that I have not received any unauthorized assistance in preparing for or writing this assignment. I acknowledge that a mark of 0 may be assigned for copied work." + Name + SID
4. Any submission after the deadline will receive a penalty of 10% for the first 24 hrs, and so on, for up to three days. After three days, the mark will be *zero*.
5. Unlimited resubmissions are allowed. But keep in mind that we will consider the last submission. That means that if you resubmit after the deadline, a penalty will be applied, even if you submitted an earlier version in time.

Real application - challenge (optional):

1. You are given a set of 100 HTML documents downloaded from the World Wide Web Consortium main web site (http://www.w3.org/). They are provided in the attached file called "W3C Web Pages.zip". The subfolder called "Text" contains the files in text format.
2. Write a program/method that given an HTML file, it computes the frequency of each word in the file and stores all words and frequencies in a hash table. The key for the hash table is the word. Use one of the hash tables discussed in the lab. Suggestion: use a string tokenizer to extract the words from the file, and consider only words of the 26-letter alphabet and 10 digits, ignoring lower/upper case.
3. Simple Web search engine: Write a program that given a set of keywords as input, it ranks the web pages based on their score (score = sum of matches per keyword, or keyword * frequency). The program should list the top 10 pages with the best matches. Suggestion: create a heap for storing the pages, where the "key" is the score of each page. Choose the top 10 pages from the heap as in the Selection problem. Test your program with a few keywords.
4. A Web dictionary: Write a program/method that takes all 100 HTML files as input, and creates a dictionary that contains all words along with their frequencies. Store the dictionary in a splay tree.
5. Some stats: Design an algorithm that takes the dictionary (splay tree) as input and lists the 10 most and least frequent words in decreasing order of frequency. Write an efficient algorithm that does not search the entire tree. Explain why your algorithm is efficient and its complexity. Write a program for your algorithm, run it and show the list of the 10 most/least frequent words.
6. The spellchecker: Write a program that takes an HTML file as input and checks whether or not each word in the file is the dictionary. Insert (in the dictionary) the words that are not found – provide a list of these words. Test it with real pages from http://www.w3.org.