Comp 251: Assignment 1

Answers must be returned online by February 1st (11:59pm), 2017.

- Your solution must be returned electronically on MyCourse.
- Written answers and programming questions must be returned in two separate submission folders on MyCourse.
- The only format accepted for written answers are PDF or text files (i.e. .txt r .rtf). PDF files must open on SOCS computers. Any additional files (e.g. images) must be included in the PDF.
- Do not submit a compressed repository with all your files. Upload instead each PDF or text file individually.
- The solution of programming questions must be written in java. Your program should compile and execute of SOCS computers. Java files that do not compile or execute properly on SOCS computer will not be graded.
- To some extent, collaborations are allowed. These collaborations should not go as far as sharing code or giving away the answer. You must indicate on your assignments the names of the persons with who you collaborated or discussed your assignments (including members of the course staff). If you did not collaborate with anyone, you write "No collaborators" at the beginning of your document. If asked, you should be able to orally explain your solution to a member of the course staff.
- Unless specified, all answers must be justified.
- When applicable, your pseudo-code should be commented and indented.
- The clarity and presentation of your answers is part of the grading. Be neat!
- Violation of all rules above may result in penalties or even absence of grading (Please, refer to the course webpage for a full description of the policy).
- Partial answers will receive credits.
- The course staff will answer questions about the assignment during office hours or in the online forum at `https://osqa.cs.mcgill.ca/`. We urge you to ask your questions as early as possible. We cannot guarantee that questions asked less than 24h before the submission deadline will be answered in time.

Exercise 1 (40 points) We wish to study the execution time of hashing with the division and multiplication methods we covered in class:

$$\begin{aligned} \text{Division method :} \quad & f(k) = k \mod D \\ \text{Multiplication method:} \quad & g(k) = (A \cdot k \mod 2^w) >> (w - r) \end{aligned}$$

Where r and w are two integers s.t. $w > r$, and A is a random number s.t. $2^{w-1} < A < 2^w$.

Let n be the number of key to insert, and m the number of slots in the hash tables. We remind you that the value $\alpha = \frac{n}{m}$ is called the load factor. In the following, we set $m = D = 2^r - 1$.

We want to estimate the number of collisions with respect to the choice of values of w, r, and α. More specifically, your task to write a java program that will take as input (i.e. in the java command line) the values r and w (in this order), and where the values of α will vary from 0.2 to 4 with a step of 0.2. The program will simulate the insertion of n keys and count the number of collision for each method (i.e. division and multiplication).

We provide you a template file `COMP251HW1.java` that you will complete. In particular, this file contains a function `generateRandomNumberInRange(double min, double max)` that will enable you to generate random number within a specified range. We also provide you a jar file `JavaPlotBuilder.jar` to visualize your results (you can run this file with the command line `java -jar JavaPlotBuilder.jar`).

Your first task is to implement the two java methods `divisionMethodImpl` and `multiplicationMethodImpl` in `COMP251HW1.java`. These methods will generate n random keys in the range $[0, 2^r[$ and return the number of collisions. Note that the value of A must be updated with w.

A typical command line to run this program will be `java COMP251HW1 8 32`, where $r = 8$ and $w = 32$. The output data will be stored in a csv file generated by the class `COMP251HW1`. You will be able to visualize your data using the jar file `JavaPlotBuilder.jar` (run the command line `java -jar JavaPlotBuilder.jar` and open the csv file with the GUI).

Your second task is to try several (well chosen) sets of values of w and r. Discuss the behaviour of the hash functions with variation of the parameters w, r, and α. Use your results to decide (and justify with your graphs) which method works best. You do not need to make many experiments. Here, we simply ask you to make a reasonable choices for the variables w and r observe the behaviour of the two methods with α, and discuss your results.

Exercise 2 (15 points) Show the execution of *heapsort* on $A = \langle 6, 4, 3, 5, 1, 2 \rangle$. Heaps must be represented as arrays. Indicate the operation made on the heap for each step of the algorithm, and show the full status of the array and variables.

Exercise 3 (15 points) We use a random hash function h to hash n distinct keys to an array T of size m. Collisions are solved with chaining. What is the expected number of clashes? Write your results using the big-oh notation and the load factor.

Exercise 4 (15 points) We implement binary trees with a dynamic data structure such that each node `x` is represented by a object that has a field `father[x]` pointing to the father, `left[x]` pointing to the left child, and `right[x]` to the right child. Let `B` be a binary tree and `x` one node of this tree. Write the pseudo-code for the procedure `RotateRight(B, x)` (right rotation) we have seen in class. This function will replace `x` by its left child, and place `x` at the root of the right subtree of the new root of the rotated tree. The right subtree of the left child of `x` becomes the left subtree of `x` (See the slides of the lectures on AVL trees).
Note: The node `x` can be any node of the tree. Your algorithm should handle all special cases (e.g. `x` is a leaf). We remind you that your pseudo-code should be commented and fully indented.

Exercise 5 (15 points) Give a recurrence for the number of possible binary search trees with n keys. You may assume the keys are $1, 2, \cdots, n$.
Hint: the root of the binary search tree can be any one of the n keys. You need to consider each of these n possiblities.