CS608 - Fall 2020 - Assignment #3

Assigned: October 4th, 2020
Due: October 11th, 2020 (Extra credit: December 1st, 2020)
NO LATE SUBMISSIONS.
Non-coding answers may be resubmitted once, after receiving feedback on the first attempt.

Collaboration policy: The goal of assignment is to give you practice in mastering the course material. Consequently, you are encouraged to collaborate with others. In fact, students who form study groups generally do better on exams than do students who work alone. If you do work in a study group, however, you owe it to yourself and your group to be prepared for your study-group meeting. Specifically, you should spend at least 30–45 minutes trying to solve each problem beforehand. If your study group is unable to solve a problem, it is your responsibility to get help from the instructor before the assignment is due.

For this assignment, you can form a team of up to three members. Each team must write up each problem solution and/or code any programming assignment without external assistance, even if you collaborate with others outside your team for discussions. You are asked to identify your collaborators outside your team. **If you did not work with anyone outside your team, you must write "Collaborators: none."** If you obtain a solution through research (e.g., on the web), acknowledge your source, but write up the solution in your own words. **It is a violation of this policy to submit a problem solution that any member of the team cannot orally explain to the instructor.** No other student or team may use your solutions; this includes your writing, code, tests, documentation, etc. It is a violation of this policy to permit anyone other than the instructor and yourself read-access to the location where you keep your solutions.

Submission Guidelines: Your team has to submit your work on Blackboard (no email) by the due date. Only one submission per team is necessary. For each class in the programming assignments you must use the header template provided in Blackboard. Make sure that you identify your team members in the header, and any collaborators outside your team, if none, write "none". Your code must follow the Java formatting standards posted in Blackboard. Format will also be part of your grade. To complete the submission, you have to upload two files to Blackboard: your source file and your class file. Your answers to questions that do not require coding must be included in the remarks section of the header. **The submission will not be accepted in any other format.**

Style and Correctness: Keep in mind that your goal is to communicate. Full credit will be given only to the correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Assignment #3 Grading Rubric

Coding:

Program characteristic	Program feature	Credit possible		
Design 30%	Algorithm	30%		
Functionality 30%	Program runs without errors	20%		
	Correct result given	10%		
Input 15%	User friendly, typos, spacing	10%		
	Values read in correctly	5%		
Output 15%	Output provided	10%		
	Proper spelling, spacing, user friendly	5%		
Format 10%	Comments, name	5%		
	Indentation	5%		
	TOTAL	100%		

Non-coding:
Embedded in questions.

1(20)	2(20)	3(40)	4(20)	**TOTAL**(100)	EC

Assignment:

The intended usage of a data structure is crucial to choose the most efficient one. Common operations include insertions/deletions, queries[1], and range queries[2]. Suppose that the application requirements are such that the crucial operations are insertions and queries. We know that hash tables have $O(1)$ query time, provided that the hash function distributes the entries uniformly. But also a good choice of the initial table size is crucial to avoid costly re-hashings when new entries are added. AVL trees are Binary Search Trees that balance themselves through rotations. Because they are balanced, the query time is $O(\log n)$. But the order in which the entries are added is also important to avoid the worst-case $O(\log n)$ rotations per insertion. The purpose of this homework is to test experimentally the insertion and query performance of a separate-chaining hash table and an AVL tree, drawing conclusions from the results observed.

Assuming that each entry is a pair `<key,value>`, where the key is used to index the entries, do the following.

1. **(20 points)** Make a conjecture for the asymptotic running time of *(a)* adding n entries with consecutive keys in a separate-chaining hash table (**the time to insert all, not each of them**) and *(b)* searching for a key that is not in the table. Justify your conjectures using the running times detailed above and any other assumptions you make. *(Partial credit for one time. No credit without justification or no reference to the times given above.)*

2. **(20 points)** Make a conjecture for the asymptotic running time of *(a)* adding n entries with consecutive keys in an AVL tree (**the time to insert all, not each of them**) and *(b)* searching for a key that is not in the tree. Justify your conjectures using the running times detailed above and any other assumptions you make. *(Partial credit for one time. No credit without justification or no reference to the times given above.)*

3. **(40 points)** Write a program that does the following.

    - Create an instance of the `Hashtable` class from the Java API.

[1]Access operations, such as read or contains.

[2]Returning multiple items.

Make the initial table size of the hash table 1000 and the load factor[3] 0.75 (which has been shown experimentally to be optimal).

- Create an instance of the `AVLtree` class attached with this homework.
- Measure the running time of adding various numbers of entries (as required by the table below) to the hash table and the AVL tree.
- For each of those cases, measure the running time of searching for a key that is not in the hash table, and do the same for the AVL tree.

Fill in the following charts. **Adjust the values of n as needed according to your platform to obtain at least 4 measurements.** You do not need to use the same range for all rows.

construction time	$n = 10^2$	$n = 10^3$	$n = 10^4$	$n = 10^5$	$n = 10^6$
Hash table					
Tree					

search time	$n = 10^2$	$n = 10^3$	$n = 10^4$	$n = 10^5$	$n = 10^6$
Hash table					
Tree					

4. **(20 points)** How does **these measurements** compare with your conjecture in parts 1 and 2? If the results differ from your conjecture, investigate the reason by looking carefully at the code of `Hashtable` and `AVLtree` provided and explain what might have happened. *(Partial credit if less than four cases are analyzed. No credit if no reference to the measured values is given. No credit for comparison between hash table and tree.)*

5. **Extra Credit** Assume that, after the initial n insertions, the only operations applied on the data are queries. Then, using the AVL tree after the initial insertions may introduce some space overhead due to the references to children. Your task is to implement a tree in an array, dump the data contained in the AVL tree into such data structure, and

[3]The load factor is the occupancy threshold for rehashing. That is, if the number of items in the table is more than the table size times the load factor, the object rehashes the table increasing the capacity.

give an approximate function of n for the space used in the AVL tree and the array-tree.