CPS 2232 - Fall 2015 - Homework #6

Assigned: November 11th, 2015
Due: November 18th, 2015

Collaboration policy: The goal of homework 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 group is unable to solve a problem, it is your responsibility to get help from the instructor before the assignment is due. **You must write up each problem solution and/or code any programming assignment by yourself** without assistance, even if you collaborate with others to solve the problem. **You are asked to identify your collaborators.** If you did not work with anyone, 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 you cannot orally explain to the instructor.** No other student 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: You have to submit your work on Blackboard by the due date. For each of the programming assignments you must **use the header template provided in Blackboard**. The header must contain, your name, course number, semester, homework number, problem number, and **list of collaborators** (if any). Your answers to questions that do not require coding must be included in this header as well. Your code must follow the Java formatting standards posted in Blackboard. Format will also be part of your grade.

To submit your assignments, you have to produce a capture file following the steps below. **The submission will not be accepted otherwise.**

1. Initiate an output capture.
   (On eve, enter `startrec yourname-hwknumber-problemnumber`)

2. Display the source code.
   (On eve, enter `cat filename.java`)

3. Compile the source code.
   (On eve, enter `javac filename.java`)

4. Execute the program.
   (On eve, enter `java filename`)

5. End capture and generate the output file.
   (On eve, enter `Control-D`)

Homework 6

Programming Assignment Grading Rubric:

The following rubric applies only to the programming assignment.

Program characteristic	Program feature	Credit possible	**Part 3**
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%	Documentation: name, collaborators, header, etc.	5%	
	Clarity: comments, indentation, etc.	5%	
	TOTAL	100%	

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

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** and *(b)* searching for a key that is not in the table. Justify your conjecture using the running times detailed above and any other assumptions you make.

2. **(20 points)** Make a conjecture for the asymptotic running time of *(a)* adding n entries with consecutive keys in an **AVL tree** and *(b)* searching for a key that is not in the tree. Justify your conjecture using the running times detailed above and any other assumptions you make.

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

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

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

[2] Returning multiple items.

[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.

- 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 adjusting the values of n as needed according to your platform.

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 results 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` (`grepcode.com`) and `AVLtree` classes and explain what might have happened.