CIS 121 - Data Structures and Algorithms
Homework Assignment 10

Given: April 10, 2018 **Due:** April 16, 2018

Note: This homework is out of **80 points** and is due **electronically on Gradescope and Canvas** on Monday, April 16 by 11:59PM EST. For late submissions, please refer to the Late Submission Policy on the course webpage. You may use a maximum of 2 late days on this homework.

A. **Gradescope**: You must select the appropriate pages on Gradescope. Gradescope makes this easy for you: before you submit, it asks you to associate pages with the homework questions. Failing to do so will get you points off, which cannot be argued against after the fact. Gradescope may prompt you with a warning to select your cover page, but please ignore this warning.

B. **LaTeX**: You must use the `hw121.cls` template Latex provided on the course website, or a penalty will be incurred. Handwritten solutions or solutions not typeset in Latex will not be accepted.

C. **Solutions**: Please write concise and clear solutions; you will get only a partial credit for correct solutions that are either unnecessarily long or not clear. Please refer to the Written Homework Guidelines for all the requirements.

D. **Algorithms**: Whenever you present an algorithm, your answer must include 3 separate sections:

1. A precise description of your algorithm in English. No pseudocode, no code.
2. Proof of correctness of your algorithm.
3. Analysis of the running time complexity of your algorithm.

E. **Collaboration**: You are allowed to discuss **ideas** for solving homework problems in groups of up to 3 people but *you must write your solutions independently*. Also, you must write on your homework the names of the people with whom you discussed. For a clarification on the collaboration policy, please see Piazza @547.

F. **Outside Resources**: Finally, you are not allowed to use *any* material outside of the class notes and the textbook. Any violation of this policy may seriously affect your grade in the class. If you're unsure if something violates our policy, please ask.

1. [20pts - Binary String Tree] Given two strings $a = a_0a_1 \dots a_p$ and $b = b_0b_1 \dots b_q$, where each a_i and each b_j is in some ordered set of characters, we say that a string is *lexicographically less than* string b if either:

1. there exists an integer j, where $0 \leq j \leq \min(p, q)$, such that $a_i = b_i$ for all $i = 0, 1, \dots, j-1$ and $a_j < b_j$, or
2. $p < q$ and $a_i = b_i$ for all $i = 0, 1, \dots, p$.

For example, if a and b are bit strings, then $10100 < 10110$ by rule 1 (letting $j = 3$) and $10100 < 101000$ by rule 2. This is similar to the ordering used in English-language dictionaries.

The *binary string tree* data structure shown in the figure below stores the bit strings $1011, 10, 011, 100$, and 0. When searching for a key $a = a_0a_1 \dots a_p$, we go left at a node of depth i if $a_i = 0$ and right if $a_i = 1$. Let S be a set of distinct binary strings whose lengths sum to n. Show how to use a binary string tree to sort S lexicographically in $\Theta(n)$ time. For the example in the figure below, the output of the sort should be the sequence $0, 011, 10, 100, 1011$.

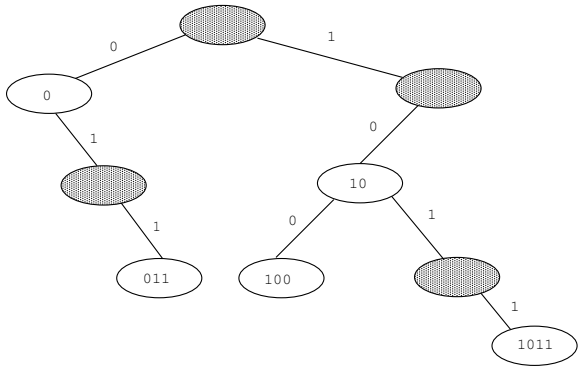

Figure 1: A binary string tree storing the bit strings 1011, 10, 011, 100, 0. Each node's key can be determined by traversing the path from the root to that node. One can store the key at the node. Nodes are heavily shaded if the keys corresponding to them are not in the tree; such nodes are present only to establish a path to other nodes.

2. [30pts - AVL Trees] Extend the AVL tree data structure to implement the following method for an ordered dictionary D in $O(\log n)$ time.

countInRange(k_1,k_2): compute and return the number of items in D with key k such that $k_1 \leq k \leq k_2$.

Note that this method returns a single integer, and that the values k_1 and k_2 may or may not be in the tree. You may use at most $O(1)$ extra storage per node in the tree. If you decide to store any new values/fields, you must describe (briefly) how you maintain those new values/fields upon insertion and deletion. Moreover, any modifications you make

cannot change the asymptotic runtimes of INSERT, SEARCH, or DELETE. As always, prove the runtime and correctness of your modifications/algorithm.

3. [30pts - Hashing] Consider two hash tables T_1 and T_2, both having the same number of slots m. With T_1, we resolve collisions using chaining, where each chain is a doubly-linked list and insertions are done at the front of the list. With the hash table T_2, we resolve collisions using linear probing, and when we delete an element, we replace it with a special marker. We use the same hash function for both the tables.

Let $S : O_1, O_2, \ldots, O_n$ be a sequence of INSERT, DELETE, and SEARCH operations, and suppose we perform S on both T_1 and T_2. For $1 \leq i \leq n$, let $C_{1,i}$ be the number of element comparisons made when performing O_i on T_1, and let $C_{2,i}$ be the number of element comparisons made when performing O_i on T_2.

A. Give an example of a sequence S in which $C_{1,i} > C_{2,i}$, for some $1 \leq i \leq n$, where O_i is a SEARCH operation. You may choose a specific value of $m \geq 5$ and a specific hash function h in your answer.

B. Let S be any sequence of INSERT, DELETE, and SEARCH operations. Suppose we perform S on both T_1 and T_2, and then perform a SEARCH operation where each item in the table is equally likely to be searched for. Let Z_1 be the expected number of comparisons made when performing the SEARCH operation on T_1, and let Z_2 be the expected number of comparisons made when performing the SEARCH operation on T_2. Prove that $Z_1 \leq Z_2$. In this case, the table size m and the hash function h are unspecified: you cannot choose values for m and h like in part (a). However, you may assume the hash function h satisfies the uniform hashing assumption.

Don't forget to submit to Canvas and to select pages on Gradescope!