CMPS 101: Winter 2016: HW 4 (Update 2)

Due: 4th March 2016

• The assignment is to be attempted in groups of two. If you choose to not work with a partner, one point will be automatically deducted from your score.

• Each group needs to submit only one set of solutions. Solutions need to be handed in the class on the due date.

• LaTeX is preferred, but neatly handwritten solutions will also be accepted. All solutions need to be handed over in the class before the beginning of the lectures.

• The names of the group members, and their UCSC ID (@ucsc.edu email address) should prominently be written on the upper left corner of the first page.

• Multiple sheets should be stapled together in the upper left corner.

• Solutions to the problems should be clearly labeled with the problem number.

• Although no points are given for neatness, illegible and/or poorly organized solutions can be penalized at the graders option.

• Clearly acknowledge sources, and mention if you discussed the problems with other students or groups. In all cases, the course policy on collaboration applies, and you should refrain from getting direct answers from anybody or any source. If in doubt, please ask the instructors or TAs.

**Question 1 (1 + 1 + 1 points):** Suppose you want to implement a hash table using open addressing. However, you are required to support insertions, lazy deletions, and searches. Write pseudo-code for the following three functions:

1. `hash-delete(T, k)`
2. `hash-insert(T, k, v)`
3. `hash-search(T, k)`
Here $T$ is the hash table, $k$ is the key and $v$ is the corresponding value. The search should return a NIL if the key does not exist in the table and the value of the key otherwise. The insert function should handle the following three cases: If the key $k$ already exists in the hash table, then update its value to $v$. On the other hand, if the key $k$ does not exist in $T$, then insert it with value $v$. Finally, if the table is full, then return a NIL. You may assume that you have access to a suitable hash function $h(\cdot, \cdot)$.

**Question 2 (1 + 1 points):** Recall that in the class we learned about two different strategies for hashing. In the first strategy, we use chaining (lets call it strategy $C$) and in the second we use open addressing (lets call it strategy $O$). For the questions below you can assume that both strategies use the same hash function and hash table size. Moreover, assume that the set of keys is fixed, and that the same sequence of insert, search, and delete operations are performed on both tables. Finally, you may assume that strategy $O$ uses quadratic probing.

- A primary collision happens when two records map to the same slot in the table. In other words, when you invoke the hash function for the first time and find a target location, a primary collision happens if that location already contains one (in the case of $O$) or more (in the case of $C$) keys. Prove or disprove the following claim: Strategy $C$ always results in a lower or equal number of primary collisions than strategy $O$.

- In the case of a successful search, strategy $C$ has to traverse through a linked list in order to find the the key being searched for. On the other hand, strategy $O$ has to search through slots in the table, in order to find the key. Prove or disprove the following claim: In the case of a successful search, the number of slots searched by strategy $O$ is larger than or equal to the number of nodes in the linked list traversed by strategy $C$.

For both the above sub-problems, you need to provide elementary proofs, without relying on theorems from the book. In particular, note that we are not doing a average case analysis. The set of keys, and the sequence of insert, search, and delete operations is fixed, and the same for both strategies.

**Question 3 (1 + 1 point)** Write pseudocode for an algorithm that takes a binary search tree (BST) as input along with two keys $a < b$, and prints out all keys $c$ in the tree that lie in the interval $[a, b]$ in sorted order. What is the time complexity of your algorithm?

- Can you perform a similar range query on a hash table efficiently? If yes, give pseudocode and a time complexity analysis. If no, justify your answer.
Changelog:

- Update 1: Problem 1: Clarified the behavior of insert if the table is full.
- Update 1: Problem 2: Fixed a typo in part 2 of the problem. Clarified that strategy $O$ uses quadratic probing. Clarified the definition of primary collision. Made it clear that the proofs need to be elementary.
- Update 1: Problem 3: Clarified that the keys in the range $[a, b]$ need to be printed in sorted order.
- Update 2: Problem 2: Part 1: Clarified that the number of collisions can be lower or equal.