CSCI 3104 Fall 2021
Instructors: Profs. Grochow and Waggoner

Midterm 2- Standard 26

Due Date .... TODO
Name .... **john blackburn**
Student ID **jobl2177**

Contents

1 Instructions

- The solutions **should be typed**, using proper mathematical notation. We cannot accept hand-written solutions. Here's a short intro to LaTeX.
- You should submit your work through the **class Canvas page** only. Please submit one PDF file, compiled using this LaTeX template.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please submit this document with no fewer pages than the blank template (or Gradescope has issues with it).
- You **may not collaborate with other students**. **Copying from any source is an Honor Code violation. Furthermore, all submissions must be in your own words and reflect your understanding of the material.** If there is any confusion about this policy, it is your responsibility to clarify before the due date.
- Posting to **any** service including, but not limited to Chegg, Discord, Reddit, StackExchange, etc., for help on an assignment is a violation of the Honor Code.
- You **must** virtually sign the Honor Code (see Section 2). Failure to do so will result in your assignment not being graded.

2 Honor Code (Make Sure to Virtually Sign)

Problem 1.

- My submission is in my own words and reflects my understanding of the material.
- I have not collaborated with any other person.
- I have not posted to external services including, but not limited to Chegg, Discord, Reddit, StackExchange, etc.
- I have neither copied nor provided others solutions they can copy.

Agreed (john blackburn). □

3 Standard 25- Hashing and Collisions

Problem 2. Consider a hash table designed to store integers, using the hash function $h(k) = k \; mod \; 3$ for all keys k for a table of size 3. (Resolve collisions by chaining with a linked list.) You have three scenarios:

- Scenario 1: keys are randomly drawn from integers that are divisible by 3.
- Scenario 2: keys are randomly drawn from integers of the form $3m + 1$ where m is an integer.
- Scenario 3: keys are randomly drawn from *all* integers

Do the following.

(a) For which scenario does the hash function $h(k)$ perform better? Please **explain/justify** your answer.

Answer. Scenario 3 will have the best performance because it will spread the keys far more evenly than the other two scenarios. In scenario 1, if all keys are divisible by 3 that means that when they're plugged into the hash function $h(k)$ they will all get put into the same table slot, because any integer that is divisible by 3 will equal 0 when modulated by 3. A similar scenario follows for scenario 2. Any integer that is multiplied by 3 then has 1 added to it that is then modulated by 3 will always result in 1, so all the keys in this scenario are stored in the same bucket. Scenario 3, however spreads the keys much more evenly throughout the table since all integers are chosen from random. □

(b) In each of the three applications, does the hash function $h(k)$ satisfy the uniform hashing property? Please **explain/justify** your answer.

Answer. Scenario 1 does not satisfy the uniform hashing property because it places all keys into the same bucket rather than spreading them evenly through all buckets. All keys are placed in the bucket at location 0 for the reason that I described in part A. As for scenario 2, It also does not satisfy the uniform hashing property because it also puts all keys in the same bucket as per my description in part A. Scenario 3, does satisfy the uniform hashing property because it fills all the buckets equally and randomly. It chooses keys at random from all integers and modulates those integers by 3 and therefore we will see keys in all the available buckets spreadout randomly and roughly equally. □

(c) Suppose you have n keys in total for each application. What is the resulting load factor for each application?

Answer. Load factor is equivalent to the total keys in our application over the size of our table.

Scenario 1: Since we only have a single table index that is used our actual table size is 1. So the load factor is $n/1 = n$

Scenario 2: We again effectively only have a single table entry ever used so the table size is 1. So the load factor is $n/1 = n$

Scenario 3: We have a table size of 3 in this case because we do actually use all the table in this case. So the load factor is $(n/3)$. □

(d) Suppose you have n keys in total for each application. What are the time complexities of the dictionary operations: add, delete, and find, respectively?

Answer. Scenario 1: effectively have a linked list implementation

add: O(1), it will always take O(1) for adding in a hash table even in the worst case.

The delete time is always 1+ load factor, we found load factor above so: delete: $O(1+n) = O(n)$

same applies for find

find: $O(1+n) = O(n)$

Scenario 2: effectively have a linked list implementation

add: O(1), it will always take O(1) for adding in a hash table even in the worst case.

The delete time is always 1+ load factor, we found load factor above so: delete: $O(1+n) = O(n)$

same applies for find

find: $O(1+n) = O(n)$

Scenario 3: keys spread throughout the table

add: O(1), it will always take O(1) for adding in a hash table even in the worst case.

The delete time is always 1+ load factor, we found load factor above so: delete: $O(1+n/3)$

same applies for find

find: $O(1+n/3)$ □