CSCI 3104 Fall 2022
Instructors: Prof. Grochow and Chandra Kanth Nagesh

Problem Set 10

Due Date .... November 14, 2022
Name .... **Tyler Huynh**
Student ID **109603994**
Collaborators .... **N/A**

Contents

Instructions

- The solutions **must be typed**, using proper mathematical notation. We cannot accept hand-written solutions. Useful links and references on LaTeXcan be found here on Canvas.
- 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 are welcome and encouraged to collaborate with your classmates, as well as consult outside resources. You must **cite your sources in this document. 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, Reddit, StackExchange, etc., for help on an assignment is a violation of the Honor Code.
- You **must** virtually sign the Honor Code (see Section Honor Code). Failure to do so will result in your assignment not being graded.

Honor Code (Make Sure to Virtually Sign the Honor Pledge)

Problem HC. On my honor, my submission reflects the following:

- My submission is in my own words and reflects my understanding of the material.
- Any collaborations and external sources have been clearly cited in this document.
- I have not posted to external services including, but not limited to Chegg, Reddit, StackExchange, etc.
- I have neither copied nor provided others solutions they can copy.

In the specified region below, clearly indicate that you have upheld the Honor Code. Then type your name.

Honor Code. I, **Tyler Huynh** on my honor pledge that my submission is a reflection of my own understanding of the material, any and all collaborations/sources have been properly cited, I have not posted any material to external sources, and I have not copied other solutions as my own. □

26 Standard 26 – Hash Tables

26.1 Problem 1

Consider the following hash function. Let U be the universe of strings composed of the characters from the alphabet $\Sigma = [\texttt{A}, \ldots, \texttt{Z}]$, and let the function $f(x_i)$ return the index of a letter $x_i \in \Sigma$, e.g., $f(\texttt{A}) = 1$ and $f(\texttt{Z}) = 26$. Finally, for an m-character string $x \in \Sigma^m$, define $h(x) = ([\sum_{i=1}^{m} f(x_i)] \mod \ell)$, where ℓ is the number of buckets in the hash table. That is, our hash function sums up the index values of the characters of a string x and maps that value onto one of the ℓ buckets.

Suppose this is going to be used to hash words from a large body of English text.

List at least 4 reasons why $h(x)$ is a bad hash function relative to the ideal behavior of uniform hashing.

Answer. Referenced from **Levet notes**
We know that the best performance for a hashing function is to be able to achieve "random-like" operations such that $h(x)$ satisfies the uniform hashing assumptions.

In regards to this question $h(x)$ is a bad hash function relative to the ideal behaviour of uniform hashing for the following reasons:

- **Reason 1:** For this hash function we can see that the runtime for this hashing function will take $\Theta(m)$ times to run since to sum up the index values of the characters of a string x and map that onto ℓ buckets it must iterate through the cells within the hashing function to do so first. Where m represents the number of characters in the string. We know that an optimized has function should take $\Theta(1)$, to run.
- **Reason 2:** The third reason would be that there exists a large amount of possibilities to where collisions can exist, such that it does not minimimize collisions, take for example the following strings of: D, AC, CA, BB, AAAA all the indexes sum to 4, such that it hashes to $4 mod \ell$. From this our hashing function will hash these strings to the same bucket, although these strings are different from each other.
- **Reason 3:** The fourth reason that $h(x)$ is a bad hash function is that no two strings are truly independent within U or the universe of strings. Since no two strings are completely independent or different than there is a chance that they can both be used by our hashing function, such that it will map similar strings onto the same buckets creating a collision. Take for example, the string Tyler and Relyt, both of these strings are of the same length and contains the same characters, but the hashing function will not distribute the values uniformly into ℓ buckets because the hashing function will distribute these two strings into the same bucket.
- **Reason 4:** Another reason would be that since there exists a large amount of collisions within our buckets and that the hash function does not distribute the strings uniformly, this would increase the runtime of the operations of lookup and deletion, since there exists so many collisions.

□

26.2 Problem 2

Consider a chaining hash table A with b buckets that holds data from a fixed, finite universe U. Recall the definition of worst-case analysis, and consider starting with A empty and inserting n elements into A under the assumption that $|U| \leq bn$.

(a) What is the worst case for the number of elements that collide in a single bucket? Give an exact answer and justify it. **Do not assume the uniform hashing assumption for this question.**

Answer. Referenced from **Levet notes**
The worst case in this scenario would be if all n number of elements were to collide with each other in a single bucket. Such that when the hashing algorithm enters the data from the finite universe U, all the elements will collide with each other in the bucket since they are all hashed under the same key into the same bucket b. $\square$

(b) Calculate the worst-case total cost of these n insertions into A, and give your answer as $\Theta(f(n))$ for a suitable function f. Justify your answer.

Answer. Referenced from **Levet notes**
To calculate the worst-case total cost of these n insertions into A, we must first find the upper bound of the worst case for the insertion operation in hash tables by the definition of worst-case analysis. We can see that expected or the best case for insertion will be $\Theta(1)$, since it is a chaining hash table. The worst case total cost of the insertion operation, will be $\Theta(n)$, since we will being n where each insertion will take $\Theta(1)$ times to run this operation. $\square$

(c) **For this part only, assume the uniform hashing assumption, and that the elements added were chosen uniformly at random from U.** After the n insertions, suppose that m `find` operations are performed. What is the total cost of these m find operations? Give your answer as $\Theta(f(n))$ for a suitable function f, and justify your answer.

Answer. Referenced from **Levet notes**
Assuming that the uniform hashing assumption is valid and that the elements are chosen uniformly at random from U, after n insertions, the m find operations performed will have a total cost of, $\Theta(m + n)$. I came to this conclusion because assuming that the hash function satisfies the uniform hashing assumption, we know that the average case runtime for the lookup operation will be $\Theta(1 + \frac{n}{m})$, taking into account that after n operations have been performed, the m find operations would have to be account for. $\square$

26.3 Problem 3

Hash tables and balanced binary trees can be both be used to implement a dictionary data structure, which supports insertion, deletion, and lookup operations. In balanced binary trees containing n elements, the runtime of all operations is $\Theta(\log n)$.

For each of the following three scenarios, compare the average-case performance of a dictionary implemented with a hash table (which resolves collisions with chaining using doubly-linked lists) to a dictionary implemented with a balanced binary tree.

(a) A hash table with hash function $h_1(x) = 1$ for all keys x.

Answer. Referenced from **Levet notes**
For this question we know that the runtime for a balanced binary tree containing n elements will have a runtime of $\Theta(\log n)$ for all operations. In a dictionary implementing a hash table while also using chaining to solve collisions which makes use of doubly-linked lists, we know that the runtime of the insertion, deletion, and lookup operations will be:

Insertion: $\Theta(1)$
Deletion: $\Theta(n)$
Lookup: $\Theta(n)$

From the above it is shown that the hash function will hash all keys to 1, such that all of these keys will be on the same index. From this I will now compare the average-case performance of a dictionary implementing a hash table to a dictionary implementing a balanced-binary tree, such that:

- **Insertion:** The runtime of the insertion operation for a hash table that is using chaining to solve collisions while using a doubly linked list will have a runtime of $\Theta(1)$. $\Theta(1)$ will run faster than the runtime of the insertion operation from a balanced binary tree such that:

$$\Theta(1) < \Theta(\log n)$$

- **Deletion:** The runtime of the deletion operation for a hash table that is using chaining to solve collisions while using a doubly linked list will have a runtime of $\Theta(n)$. $\Theta(n)$ will take more time to run than the deletion operation from a balanced binary tree such that:

$$\Theta(n) > \Theta(\log n)$$

- **Lookup:** The runtime of the lookup operation for a hash table that is using chaining to solve collisions while using a doubly linked list will have a runtime of $\Theta(n)$.$\Theta(n)$ will take more time to run than the lookup operation from a balanced binary tree such that:

$$\Theta(n) > \Theta(\log n)$$

□

(b) A hash table with a hash function h_2 that satisfies the Simple Uniform Hashing Assumption, and where the number m of buckets is $\Theta(n)$.

Answer. Referenced from **Levet notes**
For this question we know that the runtime for the insertion operation in a dictionary implementing a hash table while also using chaining to solve collisions which makes use of doubly-linked lists, the runtime of insertion, deletion, and lookup operations considering that the runtime of both the deletion and lookup operations have been changed since h_2 satisfies the Simple Uniform Hashing Assumption, such that the runtime of all three operations will be:

Insertion: $\Theta(1)$
Deletion: For this operation the runtime will be different because h_2 satisfies the Simple Uniform Hashing Assumption, such that the runtime for this operation will be: $\Theta(1 + \alpha)$, where α represents the the number of elements n over the number of m buckets such that $\alpha = \frac{n}{m} = \frac{n}{n} = 1$. Then the total runtime for this operation will be $\Theta(1 + \alpha) = \Theta(2)$.
Lookup: For this operation the runtime will be different because h_2 satisfies the Simple Uniform Hashing Assumption, such that the runtime for this operation will be: $\Theta(1 + \alpha)$, where α represents the the number of elements n over the number of m buckets such that $\alpha = \frac{n}{m} = \frac{n}{n} = 1$. Then the total runtime for this operation will be $\Theta(1 + \alpha) = \Theta(2)$.

From the above I will now compare the average-case performance of a dictionary implementing a hash table that solves collisions with a doubly linked list to a dictionary implementing a balanced-binary tree, where h_2 satisfies the Simple Uniform Hashing Assumption such that:

- **Insertion:** The runtime of the insertion operation for a hash table that is using chaining to solve collisions while using a doubly linked list will have a runtime of $\Theta(1)$ since the operation is using constant time to run. $\Theta(1)$ will run faster than the runtime of the insertion operation from a balanced binary tree such that:

$$\Theta(1) < \Theta(\log n)$$

- **Deletion:** The runtime of the deletion operation for a has table that satisfies the Simple Uniform Hashing Assumption will have a runtime of $\Theta(2)$, such that the runtime of the deletion operation in a hashing function will run faster than the deletion in balanced binary tree, as follows:

$$\Theta(2) < \Theta(\log n)$$

- **Lookup:** The runtime of the deletion operation for a has table that satisfies the Simple Uniform Hashing Assumption will have a runtime of $\Theta(2)$, such that the runtime of the lookup operation in a hashing function will run faster than the lookup in balanced binary tree, as follows:

$$\Theta(2) < \Theta(\log n)$$

□

(c) A hash table with a hash function h_3 that satisfies the Simple Uniform Hashing Assumption, and where the number m of buckets is $\Theta(n^{3/4})$.

Answer. Referenced from **Levet notes**
For this question we can see that the runtime of deletion and lookup will be $\Theta(1 + \alpha)$ because h_3 satisfies the Simple Uniform Hashing Assumption, such that the runtime for deletion and lookup considering that the number of m buckets is $\Theta(n^{3/4})$, will be:

Insertion: $\Theta(1)$

Deletion: For deletion since it satisfies the Simple Uniform Hashing Assumption, we know that the runtime will be $\Theta(1+\alpha)$. Now we must find α first so considering that the equation for αt is $\alpha = \frac{n}{m}$, where $m = n^{\frac{3}{4}}$. We can solve for α, such that $\alpha = \frac{n}{n^{3/4}}$, from this $\alpha = n^{1/4}$, such that the runtime will be $\Theta(1+n^{1/4})$

Lookup: For lookup since it satisfies the Simple Uniform Hashing Assumption, we know that the runtime will be $\Theta(1+\alpha)$. Now we must find α first so considering that the equation for αt is $\alpha = \frac{n}{m}$, where $m = n^{\frac{3}{4}}$. We can solve for α, such that $\alpha = \frac{n}{n^{3/4}}$, from this $\alpha = n^{1/4}$, such that the runtime will be $\Theta(1+n^{1/4})$

From the above now we can compare the average-case performance of a dictionary implemented with a hash table and a dictionary implemented with a balanced binary tree, such that:

- **Insertion:** The runtime of the insertion operation for a hash table that is using chaining to solve collisions while using a doubly linked list will have a runtime of $\Theta(1)$ since the operation is using constant time to run. $\Theta(1)$ will grow larger than the runtime of the insertion operation from a balanced binary tree such that:

$$\Theta(1) > \Theta(\log n)$$

  Thus, we know that the balanced binary tree will perform this operation slower.

- **Deletion:** The runtime of the deletion operation for a has table that satisfies the Simple Uniform Hashing Assumption will have a runtime of $\Theta(1+n^{1/4})$, I will do a comparison test and make use of L'hopital's rule on both function to see which function will grow faster, such that:

$$\begin{aligned}
\lim_{n\to\infty} \frac{1+n^{1/4}}{\log n} &= \lim_{n\to\infty} \frac{1+n^{1/4}}{\log n} = \frac{\infty}{\infty} \qquad \textit{Indeterminate} \\
&= \frac{\frac{1}{4n^{3/4}}}{\frac{1}{n\ln(10)}} \\
&= \frac{n\ln(10)}{4n^{3/4}} = \frac{\infty}{\infty} \\
&= \frac{\ln(10)}{\frac{3}{n^{1/4}}} \\
&= \frac{ln(10)n^{1/4}}{3} \\
&= \frac{\infty}{3} \\
&= \infty
\end{aligned}$$

  From the above comparison test and use of L'hopital's rule we can see that $\Theta(\log n)$ of the balanced binary tree will grow slower than $\Theta(1+n^{1/4})$ of the hash table, such that:

$$\Theta(1+n^{1/4}) > \Theta(\log n)$$

  Thus, we know that the balanced binary tree will perform this operation faster.

- **Lookup:** The runtime of the lookup operation for a hash table that satisfies the Simple Uniform Hasing Assumption will have a runtime of $\Theta(1+n^{1/4})$, from our work on the deletion operation of a hash table, since both of these operations have the same runtime we can say that:

$$\Theta(1+n^{1/4}) > \Theta(\log n)$$

Thus, we know that the balanced binary tree will perform this operation faster.

$\square$