CSCE 221 Cover Page
Programming Assignment #5

Bonus Due Date: November 27th, 11:59pm
Final Due Date: December 2nd, 11:59pm

First Name: MUHAMMAD TAHA **Last Name**: HAQQANI **UIN:** 426004967

Any assignment turned in without a fully completed cover page will NOT BE GRADED.
Please list all below all sources (people, books, webpages, etc) consulted regarding this assignment:

CSCE 221 Students	Other People	Printed Material	Web Material (URL)	Other
1.	1.	1.	1.	1.
2.	2.	2.	2.	2.
3.	3.	3.	3.	3.
4.	4.	4.	4.	4.
5.	5.	5.	5.	5.

Recall that University Regulations, Section 42, define scholastic dishonesty to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion. Please consult the Aggie Honor System Office for additional information regarding academic misconduct – it is your responsibility to understand what constitutes academic misconduct and to ensure that you do not commit it.

I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received nor given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment.

Today's Date: 12/01/19

Printed Name (in lieu of a signature): MUHAMMAD TAHA HAQQANI

Introduction

- Implement the hash table with Chaining, Linear Probing and Double Hashing versions
- Create a word-counting application using the three different implementations of Hash Table and output the number of occurrences for each word
- Compare the performance of each implementation using graphs and discuss the time complexity, advantages, disadvantages, and results of the experiment

Hashing:

It is a technique that is used to uniquely identify a specific object from a group of similar objects. In hashing, large keys are converted into small keys using hash functions. The values are then stored in a data structure called hash table. Hashing is popular because of their average time complexity of O(1) which is better than any other ADT.

Hash Table:

Hash Table is an implementation of an associative array ADT. An associative array is a collection of key-value pairs. Hash Table is a data structure that maps keys to their associated values using a hash function. Hashing is a process done by the hash function in which the key is used to compute an index in the hash table at which the key is mapped to the value. The hash function can take key of any type and compute the index at which it will be mapped to the hash table. A hash table has the following features:

- Used to index large amounts of data
- Key-to-value mappings are unique
- Address of each key is calculated using the key itself
- Hashing algorithm/function is used to calculate the index of a key at which the key is inserted
- Collisions occur when the hash function computes the same index value for two different keys
- Collisions can be resolved with open or closed addressing
- Hashing is widely used in database indexing, compilers, caching, password authentication, etc.

Hash Function

A hash function is a function that is used to map a data set of an arbitrary size to a data set of a fixed size which falls into the hash table. The values returned from the hash function are called hash codes or hashes. A hash function has the following objectives:

- Distribute keys uniformly in the table
- Uses all input data

- Should minimize collisions
- Easy to calculate
- Resolve any collisions

Collision Resolution:

- Open addressing: resolving by placing keys in address other than the calculated address
    - Linear probing
    - Double hashing
    - Quadratic probing
    - Plus 3 Rehash
- Closed addressing: resolving by placing keys in address same as calculated address
    - Chaining: insert key in the same index in a chain using a list data structure

Theoretical Analysis

Chaining

Chaining is also called closed addressing because keys are stored at the same address in a list. Every time there is a collision, the key is inserted into a linked list or vector that contains other keys which had the same address. To find a key, the list found at the address is traversed. The worst case happens when all the keys are inserted at the same index. This means that to insert or remove a key, the whole list will need to be traversed. This will result in O(n) time complexity. For average case or best case, the time complexity will be O(1) since the keys are evenly distributed.

Advantages:

- No fear of exhausting hash table buckets
- Keys are stored at the calculated address from the hash function
- Accessing an element can be quicker than linear probing if the distribution is uniform
- Does not require the whole table to be searched

Disadvantages:

- Worst case time complexity is O(n) when the list is large
- If the load factor is low, open addressing can be better
- Extra space required to store the list

Linear Probing

Linear probing is a form of open addressing in which collisions are resolved by placing the key at the next available address. To search for a key, the calculated address is consulted first and then the next addresses are checked to find the key. The search stops when the key is found or the next address is empty.

Advantages:

- easy implementation
- Uses less space than chaining
- Data is in one place in the array, which means that access can be quicker

Disadvantages:

- fear of exhausting hash table buckets
- Leads to primary clustering because keys are present at consecutive locations in one part of the table
- Worst case time complexity is O(n) when the whole array needs to be searched

Double Hashing

In double hashing, there are two hashing functions. It is another form of open addressing. The second hash function is used to provide an offset value in case the first function results in a collision. A good double hash function must never evaluate to zero and must make sure than all cells can be probed.

Advantages:

- Allows smaller tables than linear probing since load factor is higher
- No clustering

Disadvantages:

- higher cost of computing the next probe as two hash functions need to be computed

Summary:

Operation	Chaining	Linear Probing	Double Hashing
Insert	O(1)	O(n)	O(n)
Remove	O(n)	O(n)	O(n)

Fig: Total Worst-Case Time Complexities for insert and remove operations

Experimental Setup

Machine Specifications:

- Processor: Intel(R) Core (TM) i7-7500U-CPU @2.70 GHz 2.90 GHz
- RAM: 8.00 GB
- System Type: 64-bit Operating System

Implementation:

- Since we know our data beforehand, the table size was fixed to the number of unique elements
- The table size was chosen such that the load factor is always less than 0.5 to increase the likelihood of hitting the average case
- For chaining, elements were added in an array which contained pointers to a node which formed a linked list
- For linear probing and double hashing, a fixed array was used
- The hash function used was:

  Hash1 = Hash1 * 5 + ASCII value of each character in the string

  The number 5 was used because it is a prime number and a larger prime number was not used because it would lead to very large integers and would require importation of libraries to hold such large integers.

- The second hash function used for double hashing was:
  Hash2 = prime – (Hash2 % prime)
  This was then combined with the first hash function:
  Hash = (Hash1 + Hash2 * i) % tableSize
  A prime number was used to distribute as evenly as possible. Secondly, the prime number chosen was ¾ the size of the table to keep the values within one part of the table. At the end, the combination is such that the index produced will be very distinct since it is being added with the first hash function.
- To test the program, 1,000,000 random words from dictionary.txt were inserted and the time taken was recorded after every 10,000 words.
- The test was conducted 3 times and an average was taken
- A graph was created for running time against input size for all three implementations

Experimental Analysis

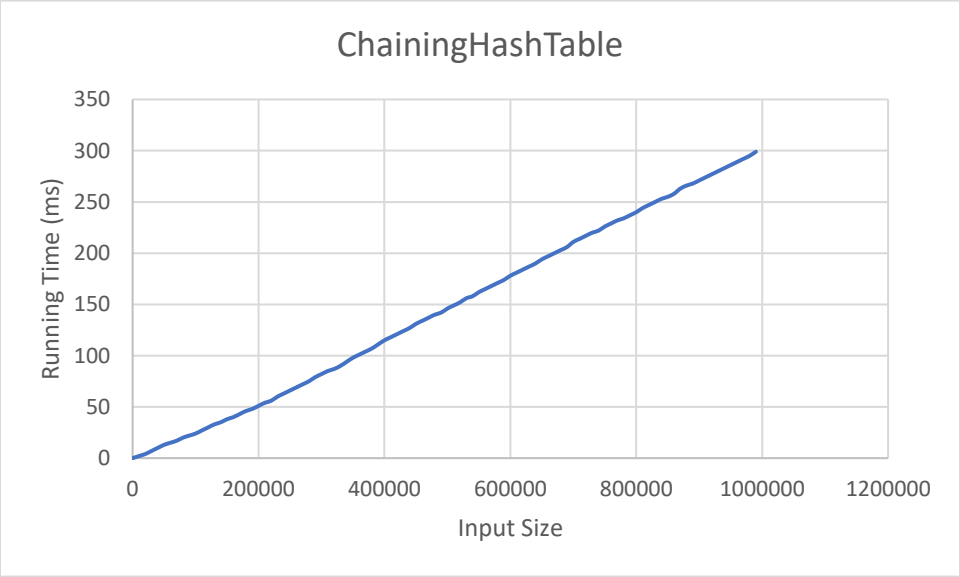

Fig: Running Time for Insertions in ChainingHashTable

From the graph we can clearly see that the running time for insertions in Chaining is O(n) for n insertions and the amortized cost is O(1) for each single insert operation.

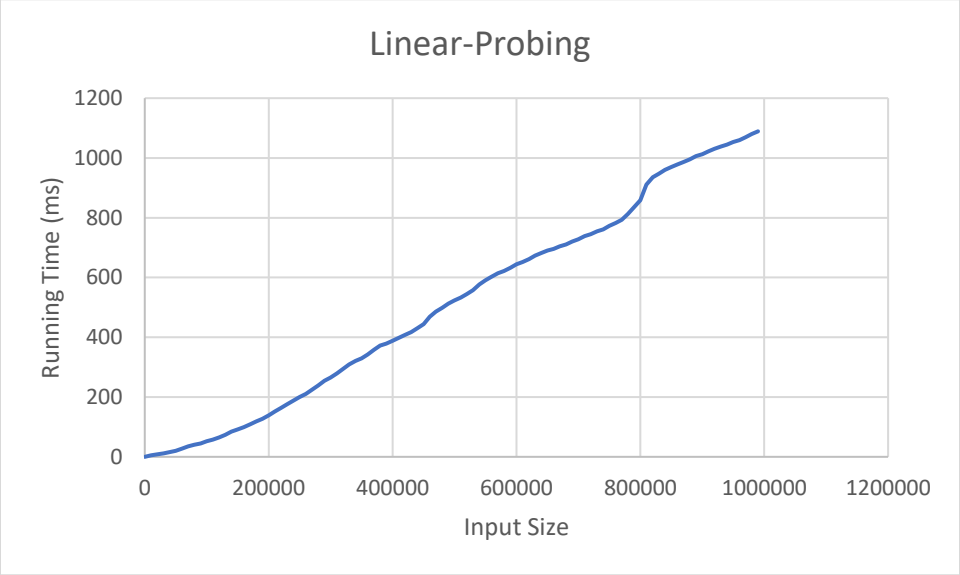

Fig: Running Time for Insertions in LinearProbingHashTable

From the graph, we see that the running time for insertions in Linear probing is O(n) and the amortized cost is still O(1). However, the running time is slower than chaining as the input size increases.

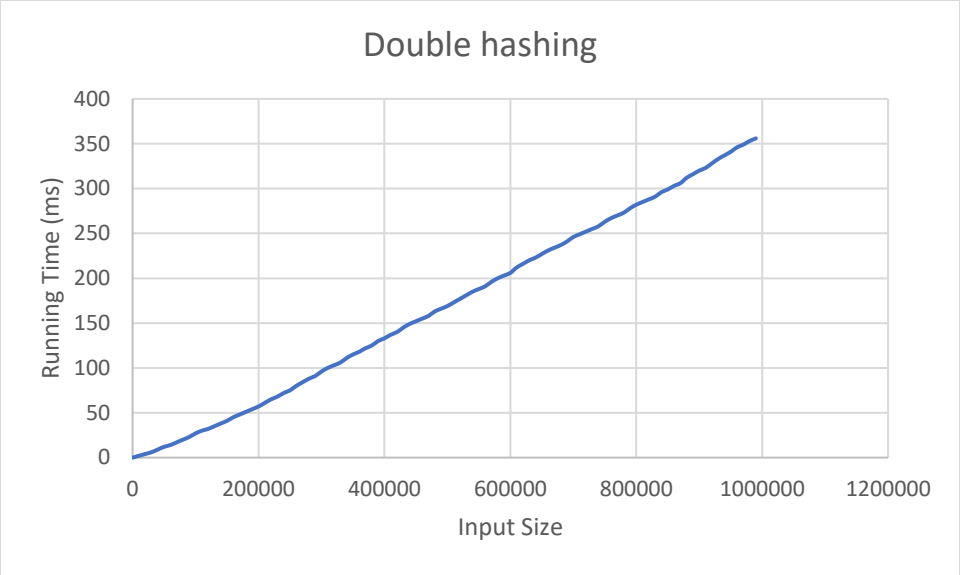

Fig: Running Time for Insertions in DoubleHashTable

From the graph above, the running time for Double-hashing technique was seen to be O(n) for n insert operations and the amortized cost was O(1). This was faster than linear-probing but slightly slower than chaining.

Conclusions

The results of the experiment supported our theoretical discussion. The following points were observed in this experiment:

- The amortized cost for insertions is O(1) for all Hash Table implementations
- Chaining was the fastest technique to insert n elements in the table
- Linear probing was the slowest technique to insert n elements in the table
- Chaining took the most space compared to the other two techniques
- The hash function and table size play a huge role in achieving O(1) amortized time complexity
- For large input sizes, linear probing becomes significantly slower and is ineffective

Limitations & Improvements:

- Table Size
    - Since the table size was fixed and chosen to be large enough to maintain a load factor < 0.5, the performance of all three implementations cannot be effectively compared because the collision rate was very low.
    - A better approach would be to have the table to be dynamic to save space and then test to see how well each technique deals with collisions.

- Input
    - The input chosen was random and more experimentation is needed to effectively compare the implementations.
    - Input should be made with strings having the same ASCII values and then strings having all different values and then the running times should be compared

- Hash Function
    - The hash function chosen was very effective and reduced the collision rates. This meant that most of the time the insertions were taking only O(1) time.
    - A dummy hash function should be used to allow more collisions in order to test how each implementation handles collisions. Different hash functions should be used and then the implementations should be tested.