CS 2302

Fall 2019

Lab Report #5

John Rodriguez

Due: November 1, 2019

Professor: Olac Fuentes

TA: Anindita Nath

Introduction

For lab 5 we were given the task to compare the running times of the two data structures like in lab 4 where we used a Binary Search Tree (BST) and a B-Tree. This time however we used the two types of hash tables, one using chaining and the other using linear probing. Then we get the word embeddings to compare two words from the hash tables and record the running times. We were also given the task to implement 5 given hash functions and create one of our own for both types of hash tables. The purpose of this lab is to make us comfortable when working with the two types of hash tables.

Proposed Solution Design and Implementation

I approached this lab by first creating the WordEmbedding class, the hash table using linear probing class, and the hash table using chaining class. I then focused on the implementing the functions needed for the inserting objects of WordEmbedding type into both types of tables and then created the 5 hash functions that we were given the task of implementing and then created a custom hash function that has the ASCII value of the last character of k % size of table and implemented it into both hash table classes. I then created a function that uses one of the six hash functions depending on the number of the user’s selection. The last function I created for both table classes was one that searches the hash table for a string and then returns the embedding for that word.

I then added two functions I had created from lab 4 which one of them being called only\_letters which checks if a string only contains alphabetical letters. The other function is called words\_diffrence which returns how similar two words are by using their embedding and then returns that number. I then implemented the menu I created from lab 4 but change the information to be about hash tables and then also added an additional menu that prompts the user to select the type of hash function they want to use when constructing their hash table.

The program then opens the “glove.6B.50d.txt” file and then reads then file line by line separating the word from the embedding and making it of the object type WordEmbedding and inserts it into the type of hash table the user selected and then shows the time it took to construct the hash table. The program then searches for words similarity using words from a custom file I created called “word file” which holds two words separated by a space on each row. It then searches for the two words in the hash table and then returns the embedding of both and then uses the words\_diffrence function to return the numerical similarity between the words and then shows the time it took to get the similarity of the word pairs.

Experimental Results

**Option 1: Binary Search Tree**

I tested my program by selecting option 1 to construct a hash table using chaining using information from the “glove.6B.50d.txt” file and then search for words from the “word file.txt” in the hash table to find the similarity between them. I tested the 6 different hash functions we implemented into our program.

**Hash Function 1:**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of text

Description automatically generatedA close up of text on a white background

Description automatically generatedA screenshot of a cell phone

Description automatically generated**

**Hash Function 2:**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of text

Description automatically generated**

**A close up of text on a white background

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**Hash Function 3:**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of text

Description automatically generated**

**A close up of text on a white background

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**Hash Function 4:**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of text

Description automatically generated**

**A close up of text on a white background

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**Hash Function 5:**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of text

Description automatically generated**

**A close up of text on a white background

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**Hash Function 6:**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of text

Description automatically generated**

**A close up of text on a white background

Description automatically generated**

A screenshot of a cell phone

Description automatically generated

**Option 2: Hash Table using Linear Probing**

I tested my program by selecting option 2 to construct a hash table using linear probing using information from the “glove.6B.50d.txt” file and then search for words from the “word file.txt” in the hash table to find the similarity between them. I tested the 6 different hash functions we implemented into our program but hade to make the table size only 100 because it takes far too much time with a larger table size.

**Hash Function 1:**

**A screenshot of a cell phone

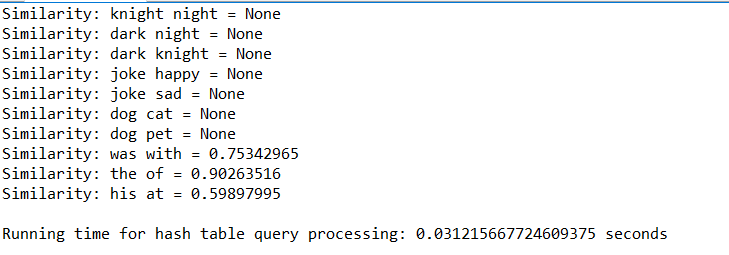
Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

****

**Hash Function 2:**

**A screenshot of a cell phone

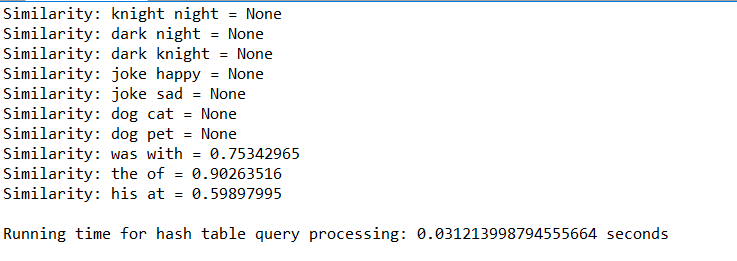
Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

****

**Hash Function 3:**

**A screenshot of a cell phone

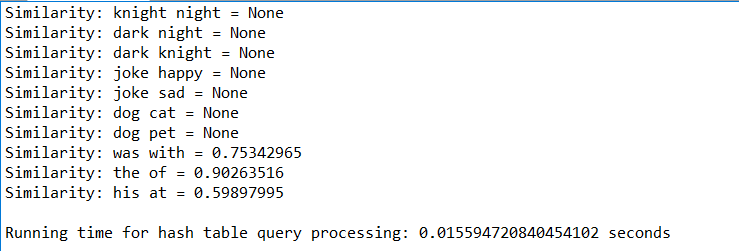
Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

****

**Hash Function 4:**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**Hash Function 5:**

**A screenshot of a cell phone

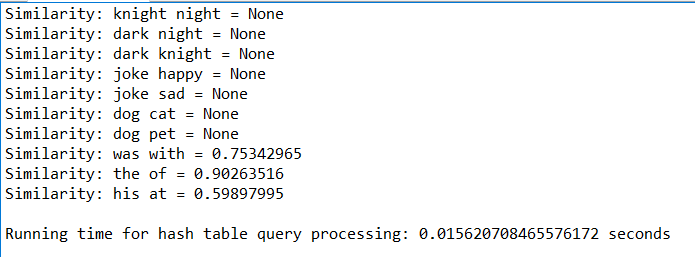
Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

****

**Hash Function 6:**

**A screenshot of a social media post

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

A screenshot of a cell phone

Description automatically generated

**Running Times Table for Hash Table using Chaining:**

|  |  |  |
| --- | --- | --- |
|  | Running Time Table Construction | Running Time Query Process |
| Function 1 | 174.1270592212677 | 0.03131699562072754 |
| Function 2 | 89.31177544593811 | 0.03124237060546875 |
| Function 3 | 19.403873205184937 | 0.01564764976501465 |
| Function 4 | 11.705357789993286 | 0.03126883506774902 |
| Function 5 | 7.649629354476929 | 0.0312650203704834 |
| Function 6 | 114.44851064682007 | 0.03126716613769531 |

**Running Times Table for Hash Table using Linear Probing (Table Size 100):**

|  |  |  |
| --- | --- | --- |
|  | Running Time Table Construction | Running Time Query Process |
| Function 1 | 18.530704975128174 | 0.031215667724609375 |
| Function 2 | 18.664013862609863 | 0.031217336654663086 |
| Function 3 | 18.23460602760315 | 0.015594720840454102 |
| Function 4 | 18.57987880706787 | 0.03124260902404785 |
| Function 5 | 19.943190097808838 | 0.015620708465576172 |
| Function 6 | 18.514169931411743 | 0.03126835823059082 |

**Running Times Table for BST and B-Tree:**

|  |  |  |
| --- | --- | --- |
| Tree Type | Running Time Tree Construction | Running Time Query Process |
| BST Test1 | 10.09720754623413 seconds | 0.031238079071044922 seconds |
| BST Test2 | 9.914376974105835 seconds | 0.017843961715698242seconds |
| B-Tree Test1 | 12.657171964645386 seconds | 0.015622138977050781 seconds |
| B-Tree Test2 | 15.335143327713013 seconds | 0.031242847442626953 seconds |

Conclusion

This lab helped me become more comfortable when using hash tables that use chaining and linear probing. I also found that the hash tables that uses chaining was able to construct much faster than the hash tables that uses linear probing which took so long I thought my program had crashed. The fastest running time of the hash tables which was the hash table that used chaining and the recursive hash function. When comparing the running times of the fastest hash table to the fastest tree running time the hash table was faster.

Appendix

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a social media post

Description automatically generatedA screenshot of a social media post

Description automatically generatedA screenshot of a social media post

Description automatically generatedA screenshot of a social media post

Description automatically generatedA screenshot of a social media post

Description automatically generated

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class