**Programming Project Report**

Name: Edgar Alcocer

Date: Nov – 21 - 2022

**Academic Integrity Statement:** I pledge that I have neither given nor received unauthorized help on this programming assignment.

**Problem Statement:**

The goal of this programming project is to give me experience with hash tables and their functionalities. My task was to optimize the hash table so words could be looked up as quickly as possible. The program inputs were reading a file which contained the top 10,000 words on google search.

The outputs of the program were the hash table which contained the (string, integer) pair, I have also added a count for the number of collisions to determine how fast the hash function works. To check the errors that were coming up within the program I have used a lot of cout statements to help me figure out what part of the program was giving the error. To make sure I was getting the correct amount of data into my hash table I implemented a counter for every time a word is inserted into the table.

**Design:**

The design of the program was to create three files, main.cpp, hash2.h, and hash2.cpp. An additional file was added named google10000.txt which contained all the words that were needed to include inside the hash table. The data structure I used for this project was a vector and a hash table. The vector oversees the storing the words that were being read into the program, it was named wordStorage. After the words have been read into the file It was time to hash the index of the words into a hash table. I create a HashTable which was double the size of the number of words in the words file, in this case it would be 20,000.

The algorithms I used were three different hash functions that could be changed in order to have less collisions, my final version of the program has the function that has the least amount of collisions. Having to choose the hash function with less collision helps the speed of the different functions the HashTable class provides, such as insertion, searching, and deleting.

**Implementation:**

The sample code I started with was the code Dr.Gauch provided. I was given the main.cpp, hash2, hash2.cpp, and google10000.txt files to begin the project. The implementation of this program begins by creating an empty hash table that is double the size of the google10000.txt file which would be 20,000. I then created a for loop which inserted a key and value pair into the hash table. The key value pair are read in from the vector which had previously stored the .txt information of the top 10,000 words on google. The hash2.cpp file contained the various functions that can be done with the hashTable, such as inserting, searching, and deleting. The hash function and the hash2 function had. To be edited to get a runtime that has considerably less collisions.

I extended and adapted his code by implementing a read and process function, a counter which detected the number of collision when calling a hash function. I have also managed to improve the hash function to reduce the number of collisions that happen when the hash function is called, I changed the num being multiplied to 1312, 1311, and 131. Finally, I have improved the hash2 function which was originally a linear probing algorithm and changed it to instead of going to the next variable spot in a linear order, I switched it to where the order of moving elements would be either 99, 7, or 13, this was to reduce the collision. The results of these changing functions are in the results section of the report. I have also provided the different hash functions in comments in the program. The development timeline of this project took around five days to complete with no errors in the vinal version of the program.

**Testing:**

I tested my program by consistently adding cout statements which allowed me to see where in the program I was receiving errors. This has cut down my debugging process immensely. The normal inputs I used for this program was the google10000.txt words that are being passed onto a vector and then being hashed into the hash table to be used later in the program whether its searching or deleting a word in the hash table.

There were no special cases I had to use because of the already given .txt file which gave me the inputs needed to demonstrate this program. Everything has worked as expected the inserting function was my base for the implementation of the different hash changes. I then added the function into the search and deleting methods.

**Sample Input/Output:**

(Hash = 1312, Hash2 = 99)

-----Welcome to Project 6-----

size of the hash table is 10000

The number of collisions for inserting are 12288

The number of collisions for search are 6043

The number of collisions for search are 12296

(Hash = 1311, Hash2 = 7)

-----Welcome to Project 6-----

size of the hash table is 10000

The number of collisions for inserting are 5393

The number of collisions for search are 2742

The number of collisions for deleting are 5392

(Hash = 131, Hash2 = 13)

-----Welcome to Project 6-----

size of the hash table is 10000

The number of collisions for inserting are 5243

The number of collisions for search are 2736

The number of collisions for deleting are 5242

**Sample Code**

//-----------------------------------------------------------

// Primary hash function

//-----------------------------------------------------------

int HashTable::Hash(string key)

{

int num = 42;

for (int i = 0; i < int(key.length()); i++)

{

// num = (num \* 1312 + key[i]) % Size;

// num = (num \* 1311 + key[i]) % Size;

num = (num \* 131 + key[i]) % Size;

}

return num;

}

//-----------------------------------------------------------

// Secondary hash function

//-----------------------------------------------------------

int HashTable::Hash2(int index)

{

// cout << "collision\n";

NumCollision++;

// return (index + 99) % Size;

// return (index + 7) % Size;

return (index + 13) % Size;

}

**Conclusions:**

The result of this program was a success, I have learned the implantation of inserting, searching, and deleting. I have also learned how to change the hash function to optimize them better to reduce the collisions. What I would do differently next time would be to implement a menu system to help me debug easier and having the ability to choose between the three different hash functions. The project took around 5 days to complete with the desired outcome.