# California State University, Fresno

# DEPARTMENT OF COMPUTER SCIENCE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **Algorithms & Data Structures** | | | Semester: | **Spring 2022** |
|  | | | | | |
| Points |  | Document author: | **MingkuanPang** | | |
|  | Author’s email: | **Yafking20 @mail.fresnostate.edu. email** | | |
| Laboratory number: | **07 – Hash Table** | | |
|  | | | | | |

**1. Statement of Objectives**

This lab asks for implementation of hash table and use it to count occurrences of words in the input text file. The output should be the number of unique words in the input file and the list of the ratios

cost[i]/updates.

**2. Experimental Procedure**

**Class hash array:**

Text

Description automatically generated

**Global variable HASHSIZE:**

There is a global variable HASHSIZE in the class hash\_array header file which has been assigned to 4001. Because we already knew there is 3684 unique words in the input file, we can make our hash\_array size to be 4001 which is the primary number that close to the target that we want to store.

**Prvate variables in the hash\_array:**

**Hash\_table:**

In the class hash\_array, the hash\_table is a string array that stores the hash variable (the words in the input file.) with size of HASHSIZE for the hash\_array. It will be initialized with value “NIL”.

**Hash\_a:**

The hash\_a stores the count occurrences for each unique words in the input file with size of HASHSIZE and it will be initialized with 0.

**Len\_of\_array:**

The len\_of\_array represents the size of hash\_a which means the amount of the elements in the hash\_a that do not equal to 0. It will be initialized with 0.

**Number\_array\_access:**

The number\_array\_access represents how many times the array access needs to be made in one update of the content of hash\_a. It will be initialized with 0.

**Updates [HASHSIZE]:**

The updates is an integer array with size of HASHSIZE. It stores the number of updates that were done while the size of hash\_a is equal to i. It will be initialized with 0.

**Cost [HASHSIZE]:**

The cost is an integer array with size of HASHSIZE. It stores the total number of probres that were done while the size of hash array is equal to i. It will be initialized with 0.

**Public methods in hash\_array:**

**int size ():**

Text

Description automatically generated

the size method just returns the current size of the hash array.

**Update(string):**

A screenshot of a computer

Description automatically generated with medium confidence

The update method takes a string parameter and update the content of the hash array with the parameter. First it calculates the hash key by using hash\_func. Then start searching the available position in the hash\_table with the hash key. During the searching, if it found the variable is already in the hash table, then it will not make any change on the hash\_table.

If it found the value on the current position is ‘NIL’ that means the position is available, then it inserts the parameter into hash\_table. If reached the end of the hash table then rest the key to be 0.

meanwhile, records how many times array access has been made.

**Hash\_func(string):**

Text

Description automatically generated

The hash\_func method takes a string parameter and calculates its hash value by hash algorithm and return the value as a result. The way it calculates its hash value is sum the ascii values of each letter in the string and mod it with HASHSIZE.

**Probes ():**

Text

Description automatically generated

The probes returns how many times array access were made in the last update method call.

**Get\_costs ():**

A screenshot of a computer screen

Description automatically generated with low confidence

The get\_costs returns an array cost which cost [i] represents the total number of probes that were done while the size of the hash array is equal to i.

**Get\_updates():**

Text

Description automatically generated

It returns an array updates which updates[i] represents the number of updates that were done while the size of the hash array is equal to i.

**3. Analysis**

**Main function:**

Text

Description automatically generated

The main function reads the content of a txt file “RomeoAndJuliet.txt” and put it into a hash array.

The hash array only holds unique word. After finish reading the txt file, it will output how many unique words that file contains which is the size of the hash array. Then it will output cost[i]/update[i], which is the total number of probes were done in each update while the size of hash array is equal to i.

**Output**

**a.**



**b.**

Background pattern

Description automatically generated

Due to the size of hash array is very large, the screen shot is just the part of output for part B

**4. Encountered Problems**

To be honest, I did not encounter any problems while doing this lab.

**5. Conclusions**

From this experiment, I realized that when we know the number of elements to store, and the number is not very large, we can make an appropriate size of hash table to store the data and use linear probing to insert the data. That is very easy to set up than other insertion method such as double hashing or chaining. Although linear probing is simple to implement, it does not provide as many probe sequences as double hashing, so double hashing can avoid collision more effectively.

**6. References**

I did not use any reference in this lab.