Department of Computer Engineering

BLG 335E
Analysis of Algorithms I
Project 3 Report

Fall 2018

Muhammed Raşit EROL

150150023

1 INTRODUCTION

In this project, it required that implement several hash functions on the given input files to store dataset in to a m-sized hash table. There are two input files: “vocab.txt” and “search.txt”.

2 IMPLEMENTATION

In this project, there are three types of hashing class which are called LinearHashing, DoubleHashing and UniversalHashing. They all are same properties in some cases. The difference between them is occurred on hashing function type. Hashing functions of them can be seen in the assignment pdf.

Constructors and destructors of these classes can be seen in the code. Several functions of that classes are called in the main in order to hashing and searching on given files. Hashed and searched file is read once in the beginning of the program. The values from that files are stored in the vector. The address of that vector is assigned into class pointers. Hence, all files can be used from all classes. Before hashing and searching size of input must be set. Because loops are works until that locations. Also, hash size can be change in the code. For each class, for two different m size firstly hashing and after that searching functions are called.

In my code, double and linear hashing is applied as mentioned in pdf. However, in the universal hashing, after collision is occurred linear hashing method is used. After created location with universal hashing is not empty then collision occurs and code start the locate the string with linear hashing.

Furthermore, in the searching linear and double search is applied same as adding new element into hash table. Also in the universal hashing, search is applied firstly using universal hashing following with the linear hashing.

In my code, line number is stored in the map with the read string. Hence, searched string is firstly searched in that map. If produced location is empty or is not searched string, then collision occurs and for both inserting and searching collisions are increased in this manner.

Therefore, in the linear and double hashing there is no collision in the searching and locating the string in the hash table. Because string locations are created with the line numbers. However, in the universal hashing locations are calculated randomly. Hence, there are collisions when inserting. Also, in the searching, first looked position is calculated randomly after that linear probing is used.

When unseen words are searching, loops are works until hash table size. Hence whole table is searched and still word is not found then loop is broken with the message which contains the unseen word.

In the running time on time program, I gave extra information about the status of the flowing of the program. So, when the program crash user can understand the breakpoint of that step.

C:\Users\mrero\source\repos\algo-proje-3\Debug\algo-proje-3.exe

```
Input file reading and creating hash tables for all type of hashing has been started!
-------------------------------------------------------------
13422 line is read from: vocab.txt
-------------------------------------------------------------
1500 line is read from: search.txt
-------------------------------------------------------------

LINEAR HASHING:
-------------------------------------------------------------

Creating Linear Hashing Table: ( m = 17863) :
Number of inserted element:( m = 17863) : 13422
Collision number when insertion with linear hashing ( m = 17863) : 0

Searching on Linear Hashing Table:: ( m = 17863) :
Number of found element:( m = 17863) : 1500
Collision number when searching with linear hashing ( m = 17863) : 0
-------------------------------------------------------------

Creating Linear Hashing Table: ( m = 21929) :
Number of inserted element:( m = 21929) : 13422
Collision number when insertion with linear hashing ( m = 21929) : 0

Searching on Linear Hashing Table:: ( m = 21929) :
Number of found element:( m = 21929) : 1500
Collision number when searching with linear hashing ( m = 21929) : 0
-------------------------------------------------------------
```

```
DOUBLE HASHING:
-------------------------------------------------------------
Creating Double Hashing Table: ( m = 17863) :
Determined prime number is : 17851
Number of inserted element:( m = 17863) : 13422
Collision number when insertion with linear hashing ( m = 17863) : 0

Searching on Double Hashing Table: ( m = 17863) :
Determined prime number is : 17851
Number of found element:( m = 17863) : 1500
Collision number when searching with double hashing ( m = 17863) : 0
-------------------------------------------------------------
Creating Double Hashing Table: ( m = 21929) :
Determined prime number is : 21911
Number of inserted element:( m = 21929) : 13422
Collision number when insertion with linear hashing ( m = 21929) : 0

Searching on Double Hashing Table: ( m = 21929) :
Determined prime number is : 21911
Number of found element:( m = 21929) : 1500
Collision number when searching with double hashing ( m = 21929) : 0
-------------------------------------------------------------
```

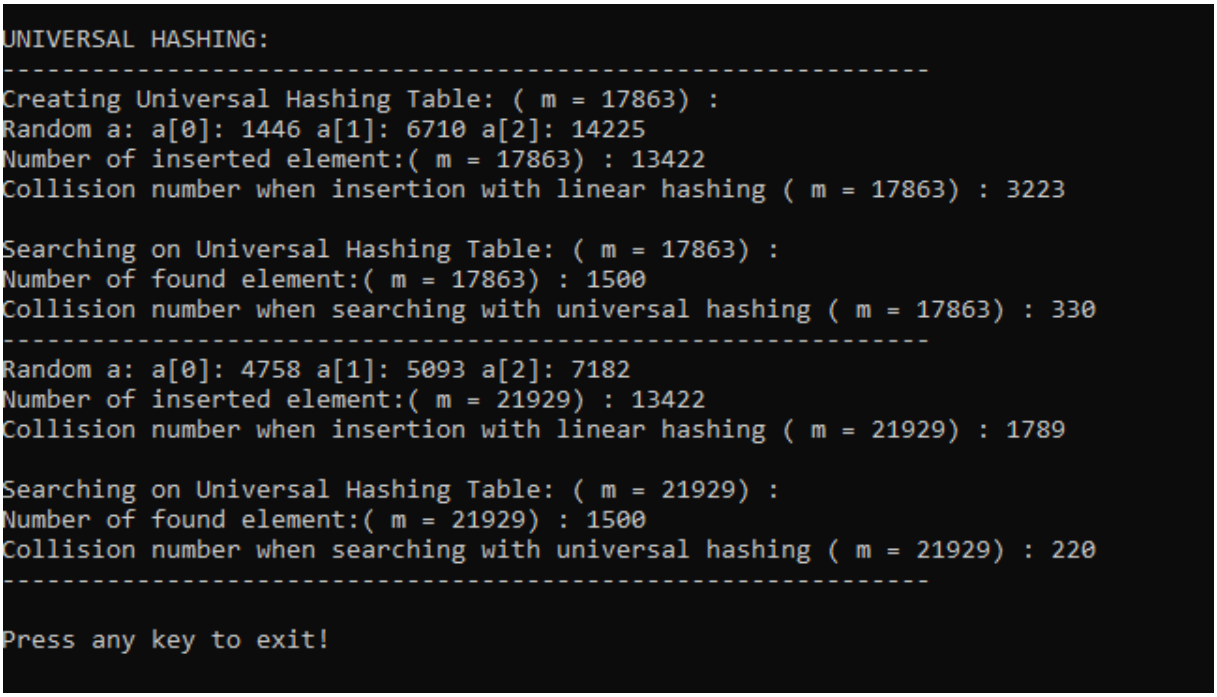

Figure 1 Output of the program

3 ANALYSIS

When the creating hash table collusion on the linear hashing is zero. Because we use line number to hash strings. In the project, each line is continuously increasing; hence, linear hash function generates different locations for each word. Also in searching, same approach is used.

Moreover, in the double hashing, same situation occurs. Because of the line numbers, searching and hashing is completed with the zero collision. Both for linear and double hashing key is the line number both work similarly.

However, in the universal hashing, location is determined randomly with decomposition. Hence each runtime, collision number can be change when inserting new string. Also in the searching, collision number is changed when I rerun the code. Furthermore, searching collision number is less than inserting collision number because searched word number is less than inserted words.

The result of the collisions can be seen in the Figure 2.

Insertion			
m	Linear	Double	Universal
17863	0	0	2432
21929	0	0	15658

Searching			
m	Linear	Double	Universal
17863	0	0	220
21929	0	0	1587

Figure 2: Result of collisions

4 COMPILING

The code can be compile like that:

- g++ *.cpp –o output.o –std=c++11
- ./output.o vocab.txt search.txt