**Assignment 1: Hash Tables**

Department of Computer Engineering

Dr Mélanie Bouroche

**Submitted By:**

Aniket Agarwal

Student ID: 17317437

**WRITE UP:**

Hashing:

* Data records are stored in a hash table.
* The position of a data record in the hash table is determined by its key.
* A hash function maps keys to positions in the hash table.
* If a hash function maps two keys to the same position in the hash table, then a **collision** occurs.

Linear Probing:

* During insert of key k to position p: If position p contains a different key, then examine positions p+1, p+2, etc.\* until an empty position is found and insert k there.
* During a search for key k at position p: If position p contains a different key, then examine positions p+1, p+2, etc.\* until either the key is found or an unused position is encountered.

Double Hashing:

* Double hashing is a [computer programming](https://en.wikipedia.org/wiki/Computer_programming) technique used in [hash tables](https://en.wikipedia.org/wiki/Hash_table) to resolve [hash collisions](https://en.wikipedia.org/wiki/Hash_collision), in cases when two different values to be searched for produce the same hash key.
* It is a popular [collision](https://en.wikipedia.org/wiki/Hash_collision)-resolution technique in [open-addressed](https://en.wikipedia.org/wiki/Open_addressing) hash tables.
* Double hashing is implemented in many popular [libraries](https://en.wikipedia.org/wiki/Library_(computing)).

Load Factor:

* The load factor (alpha) of a hash table with n elements is given by the following formula: α = n / table (length).
* Thus, 0 < α < 1 for linear probing.( α can be greater than 1 for other collision resolution methods)
* For linear probing, as α approaches 1, the number of collisions increases

**TASK 1:**

My approach to the problem:

* Retrieve data from the file and store it in the linked list
* Indexing of the data on the basis of linear probing with the help of hash function as given in the Assignment
* Calculating the **No. of Collisions** and **Total No. of terms** and the **Load Factor** and printing it.
* Then searching the file on the basis of hash value for the name in the list and displaying it with its frequency.

**TASK 2:**

My approach to the problem:

* General approach to the problem remains the same as was in the task 1.
* Modifications to the **hash function** were made by me to **reduce the no. of collisions**.
* Modifications to the Hash Function:-

The range we are using is zero to 99991 so that our table is large enough and the prime number size reduces the probability of collisions from different strings hashing to the same value. We want to generate a hash code that is the size of our table. So we mod the calculated hash to ensure that it is in the proper range of our hash table entries. 99991 is a prime number which provides better characteristics than a non-prime number table size.

* Algorithm used is djb2
* This algorithm (k=33) was first reported by dan bernstein many years ago in comp.lang.c. Another version of this algorithm (now favored by bernstein) uses xor: hash (i) = hash (i - 1) \* 33 ^ str[i]; the magic of number 33 (why it works better than many other constants, prime or not) has never been adequately explained.

**TASK 3:**

My approach to the problem:

* General approach to the problem remains the same as was in the task 1 and task 2.
* Modifications to the **hash function** by using the method of double hashing were made by me to **reduce the no. of collisions**.

**TASK 4:**

My approach to the problem:

* General approach to the problem remains the same as was in the task 3.
* Modifications to the **hash function** by using the method of double hashing were made by me to **reduce the no. of collisions**.