Report Assignment 1

Harshit Pandit TCD23364360

This is my report for the Data Structures Assignment 1 for implementation of Hash Tables. Most of my implementation and programming work was done in Debian 12 Linux and my text editor is vim.

In the following Report I have defined how I implemented the hashing for each task. I have used some bits and pieces of the skeleton code which was provided to us like the macros, the type of the element and the hashing function.

Task 1

For the first task I created my own custom parser which takes CSV file as an input and iterates through each of the lines which in this case, turns out to be a list of Irish surnames. Each surname was present in one line each. The parser reads each of these names, copies them into a buffer and replaces the '\n' into '\0' so that we get the proper string for hashing.

The name is then passed on to 'appendHashTable' function which first searches the name in the hash table and checks whether the space at the hashed index is available or not. If it's not available, we are returned with the pointer that points to that memory. If we get a pointer, we simply update that data through the pointer by updating the frequency to one more.

In the case we're returned with NULL value, it indicates that there is a space available in that index. We dynamically allocate a memory for the Element, find out it's hash and check if the same hash value is empty or not. If it's not empty, then we allocate the next empty value. This is done using the while loop.

The search function works in a similar manner: - get the hash value of the name, keep checking if the allocated space is available or not using a while loop and return the element. If all the hashes are NULL, the condition is false, and we return NULL.

Task 2

Now changing the hashing the function, I somehow came up with my own custom hashing function. In here, we take the ASCII value of each of the char in the name string, find out it's remainder with (size of the array -1) and add that to the hash value. When the loop is finished, the hashed value is divided again with (size of array+1) and we return its remainder. In this way, the number of collisions comes down from 39 to 25. In this way, we can store each of the records more efficiently with a smaller number of collisions.

Task 3

Now here we were provided with another hash function hash3() which finds out the hash of a string in a slightly different method than hash1(). I can apply the formula

```
hash(x) = [ hash1(x) + i*hash2(x) ] % ARRAY_SIZE
```

We could use the hash function provided in task 1 and use it with hash3 function to get the final hashed index of the element to be placed.

Here 'i' denotes the number of attempts to get to the NULL value. So while iterating over the hash table, each time we face a non-empty and different name slot, we increase the value of 'i' by 1.

Calculating collisions, number of terms and the load factor

In all the above three tasks, we must calculate the number of terms, number of collisions and the load factor. The load factor simply is the ratio of number of terms that are filled to total number of slots available in the array.

To calculate the number of collisions, we define a variable `int collisions=0' as a global variable. We update the variable in insert function, while we are checking the NULL value to add the new element. The number increases if we are not found with an empty slot, and we have to move on to the next slot.

To calculate the number of terms, we simply iterate through the hash table and check if the slot is not NULL. If the slot is not null, we increase the value by one. The datatype of this variable is float in my case because I also want to calculate the load factor to avoid any loss of data during division which I encountered with.

The user input section

In the user input section, we simply put an infinite while loop and waited for the user to enter a name. If the name user entered contains space, that too was taken care of because I am using the gets() function which accepts spaces (For some odd reason, it was showing warning in my terminal and telling me to use fgets() instead which was not working) If you enter any name that is available in the Hash table or in the CSV file, it returns the name along with the frequency in the order formatted. If you enter any other name, it returns 0.