Hash Tables – Assignment 1

3D5

Name: Oisín Cullen |Student number: 20332292 | Date: 13/10/2022

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Task 1

My first assignment works as follows:

The main function calls load file which starts parsing a csv file into strings of data. Once the data is parsed the string is then brought into the "Add or Increment" function.

Add or increment has 2 possibilities, we first use a search function to check the hash table and check if that string is stored in that location of the table. If the search function returns Null we know that the string is not yet stored and call on the insert function.

If Null is NOT returned, we know that the search function found the string in the table and hence we do not need to add it again if this is the case we get the key value of the string and start or search until we find its actual position in the table once it is found we have a counter which counts the collisions occurred and the frequency of names stored. Both the frequency and collisions get stored in the location where the name is actually located in the hash table.

When **Insert** is called it finds the key value of the string and does a double check of search to make sure that the string is not stored in the array. If search returns Null we know that the string is not present. Inert then checks if there is space at the key position if there is nothing stored the name is stored and the function is returned. If this does not happen it cycles through the array till a free space is found.

Search is a very simple function which gets the key value of the string and checks from that position down the array until it hits a Null space in the array or if it returns to its start value of the search. (if key cycles to end of array let =0 and continue search). If the key cycles down the array and finds a Null value, we know that the value was not inserted and we subsequently return NULL. If the string is found in the array, we return the position in which the string was found.

Hash Function the hash function is a function that creates a unique integer value depending on the characters of the string.

Create Element Works by returning a pointer to an address of memory which has allocated space to fit our struct.

To get information about the hash table I made a function called "**Print Stats**" it adds up the total amount of collisions that occurred, counts how many terms were actually stored in the array and calculates the load on the array (Numterms/array size).

Comments: Some alterations had to be done to the load file function so that it ignored whitespaces, the same was done to the scanf function so that when the user put in a ' ' it wouldn't stop reading the input.

https://stackoverflow.com/questions/33313517/why-cant-scanf-read-white-spaces

Task 2

The functionality of task 2 is identical to task one the only difference is the hash function.

We were tasked with implementing "a better hash function". Before I explain my choice of function I think it is important to define a good hash function.

A good hash function is:

- A function which is easy to implement.
- A function that will spread the strings out evenly reducing the amount of different strings having the same key value.
- Ideally not computationally heavy so our time complexity and space complexity is respectively balanced.

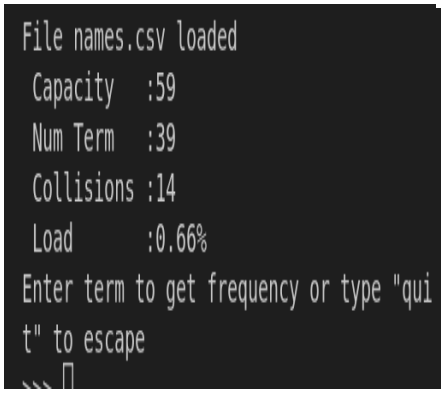
The hash function I chose to use was a polynomial rolling hash function. The reason why I chose this was after analyzing the data set of Irish names certain traits became apparent. The first was that there was **high repeatability** at the start of Irish names, For example in our data set there was a lot of Mac X or O' Xs so a good way to reduce the affect of having the same start of each name is by weighing those unique start values away from the rest and then letting the back of their name have an affect on the hash function to decide exactly where each Mac or O name will fall. This essentially moves these names to their own unique regions in the table where the end of their name will decide their exact location(or that's the idea!!) separate from each other and become more unique.

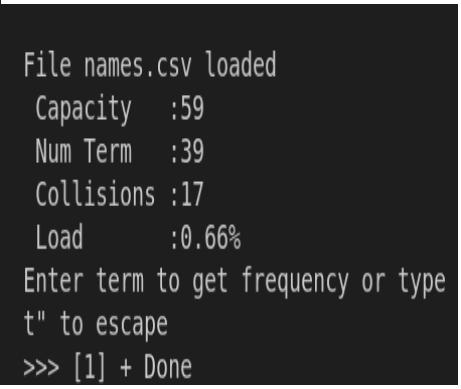
$$\text{hash(s)} = \text{s}[0] + \text{s}[1] \cdot p + \text{s}[2] \cdot p^2 + \cdots + \text{s}[n-1] \times p^{n-1} \mod m$$

I decided to modify my constants p and m until a good appeared, choosing classical prime numbers and exponentials respectively

Comments:

In all I found my function a better implementation of the hash function especially on higher load percentages. In all my function was more effective then my first hash function. Whilst I do keep looking for certain characters the scaling works as the first exponent being multiplied by a constant





Assingment 2 Assingment 1

Credit:

https://www.geeksforgeeks.org/string-hashing-using-polynomial-rolling-hash-function/

Also spoke to Jakub Ptrisunski and Nollaig mchugh about the hash function, Jakub about the repeatability of Irish names and Nollaig about length sorting options.

pstrusij@tcd.ie

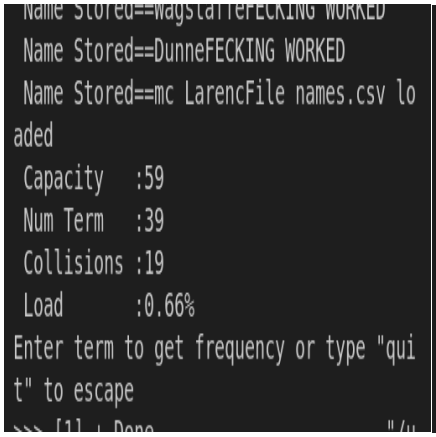
mchughno@tcd.ie

Assignment 3:

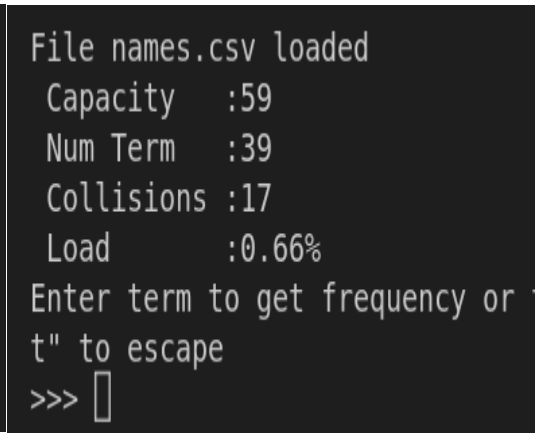
Task 3 works very similar to a task 1 and 2, the only real differences is that the search and insert function is altered so that if NULL isn't found the key is updated and finds an checks if the new key position is NULL if so the search or insert function responds accordingly. One problem I had with this task was when I nested the 2 hash functions into a main hash function would cause errors or issues. I believe this could be due to the amount of nesting occurring and maybe a double pointer should have been used to remedy this. However I found that if I called the 2 functions separately and brought their integer values into the function it worked fine.

Which is better double hash or linear probing?

In the data set of 59 I found that linear probing was more effective, however this could be due to bugs or what I believe is because due to lack of space. If we increase the space both systems perform the same, however if a better secondary hash function was selected for this task this again would improve the efficiency of the data.



Task 3



Task 1

https://www.geeksforgeeks.org/double-hashing/

Task 4

I was unable to start task four this week due to time constraints.