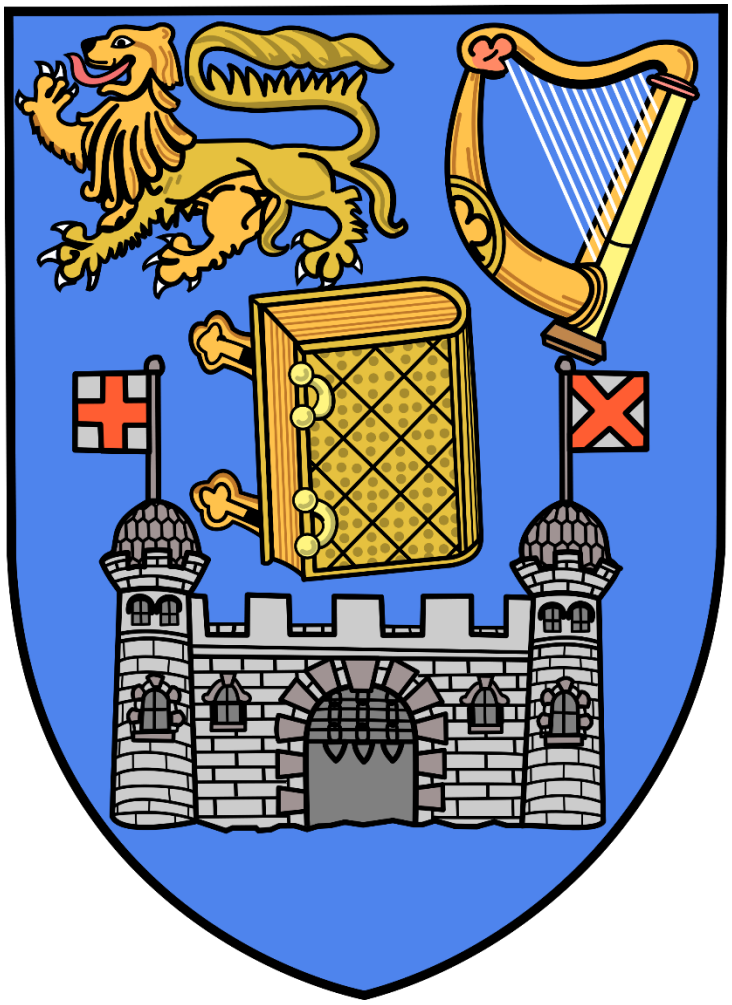

3D5A – Hash Tables Assignment

Name: Donal Tuohy

Student No: 14313774

Date: 26/10/16

Abstract

Firstly, an explanation and analysis of hash table created for the assignment. Two different collision solutions are used:

- Linear probing
- Double Hashing

Linear probing results in a build-up of adjacent blocks as more collisions occur. This is called clustering. Double hashing is a much more widely used technique as it reduces collisions and clustering effects.

Secondly, the open addressing method of Coalesced Chaining is examined and reviewed. This is another type of collision resolution in which every slot has a secondary slot where the index of the next collision key is stored. It minimises the effects of clustering but requires complicated code and also deleting data is difficult.

Introduction

Hash Tables and their corresponding hash functions are used in many types of modern day programs such as college databases to online phonebooks. A hash table, in many cases, is the most efficient way of storing data as there is little or no iterations through the data to find a specific index. This is because of the hash function associated with a table. It returns a specific integer created by using the characters in a string. The trick to creating an efficient hash function is trying to make every piece of data its own unique index. When two different pieces of data or keys have the same index, this is called a collision. When designing a hash function the aim is to have an algorithm that results in as little collisions as possible. For the assignment, students had to create a double hashing function which is a common way to deal with these collisions.

Implementation and Analysis

The first part of the assignment was to implement a double-hashing hash table in C. To begin a simple linear probing hash function was created. It can be seen in *Figure 1* that the algorithm takes in an array of characters. It then takes each individual character and sums their ASCII values. One noticeable problem with this however is that if a name has the same characters in different order, they will both have the same index (e.g. “Teela” and “Aleeta”). This was addressed by adding the ASCII value of the first letter to the sum. To keep the index

within the bounds of the hash table, the remainder of the sum divided by the size of the hash table, ***M***. If a collision occurs, the function moves one down to the next slot in the table and so on and so forth until there is a free slot.

To implement the double hashing feature to this table, a second hash function was created. This one used the same basis but was adjusted slightly.

It takes the sum of all character's ASCII values and subtracts the first value. The function then returns the remainder of this value when divided by half the table size. This value is then used as the amount of spaces to step when a collision occurs. It is divided to half the size of the table so that it will not just return to the same slot.

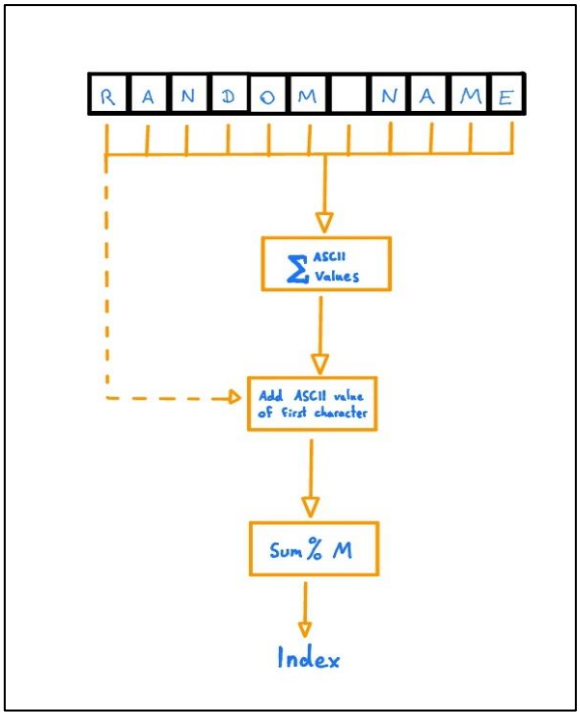

Figure 1: Linear Probing Hash Function

The **load factor** of the table is found by:

$$Load\ Factor = \frac{n}{k} = \frac{number\ of\ entries}{number\ of\ buckets}$$

If this table is full, the load factor is equal to one. This is because there is no chaining and there is only one bucket per entry.

Coalesced Chaining

This method of collision resolution is a combination of two other methods:

- *Separate chaining* is when each index has a separate bucket and keys with the same index are stored in some sort of list.
- *Open addressing* is the process of searching for empty locations where collision members can be stored.

Coalesced chaining requires there to be two array slots per key as the index to the next key of the same index is the link between these colliding strings. A collision is dealt with by taking the key that is already indexed and probing from the bottom of the table until an empty array slot is found. The index of this slot is then inserted back into the original keys secondary slot. It is easier to explain by looking at Figure 2[1].

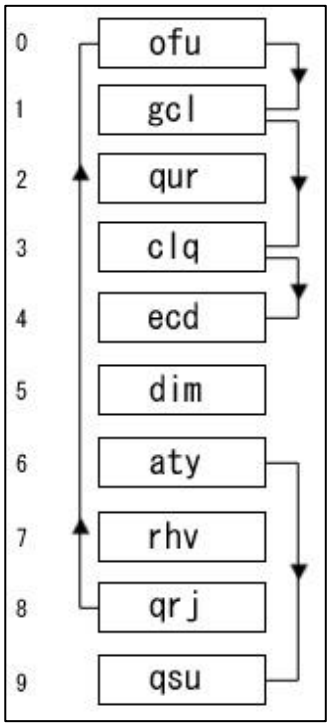

Figure 2: Coalesced Hash Table featuring links between separate buckets.

[1] Figure 2: Taken from Wikipedia article on coalesced hashing. https://en.wikipedia.org/wiki/Coalesced_hashing

In this diagram, links are shown between certain indexes. This means that a collision has occurred between the two linked buckets and if you want to find a word which shares index you can transverse across these links until you get to the bucket you are searching for. The step length does not need to be known or constant as it has a direct link to the next key of that index. It has the characteristics of a separate chaining hash table but does not store more than one key per bucket. This is combined with the open addressing idea of traversing to empty buckets. When discussing efficiency of this method, there are both positives and negatives. It means that buckets with a different index do not have to be checked like in linear probing. If a poor hash function is used which cause a lot of collisions, the worst case scenario will go towards O(n) as it will have to pass through every bucket.

Discussions

From part 1 of this assignment, the different types of hashing have become clear. Linear probing is a useful single array that can easily be accessed and copied easily. Double hashing, however is a very useful technique for reducing clustering than is associated with linear or quadratic probing. As each key has its own step value, big blocks of keys are unlikely to build up beside each other.

Both of these methods are completely fine for small tables but once you scale it up to larger databases with thousands maybe millions of buckets they will be very inefficient and might not work at all. From my reading online, hash tables, their hash functions and the efficiency of them is a massive area of study. Every day programmers are working on making their hash functions better.

The hash function is in essence, the base of security encryption[2] in all modern day programs and apps. Popular apps such as Whatsapp and Facebook use encryption functions which have the same idea as a hash function. A string is passed into the function and it returns a series of characters which are unique to that function which can be seen simplified in Figure 3[3].

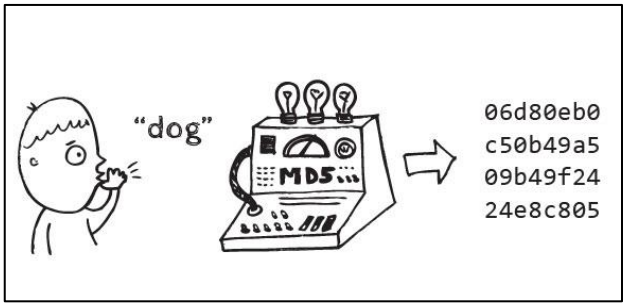

Figure 3: Simplistic explanation of a cryptographic hash function.

Conclusion

It has been interesting to learn about hash tables and how they are such a current topic. One can see that there are always advantages and disadvantages that come with every method and more algorithm options should be considered when addressing future projects.

[2] Cryptographic hash functions: Information sourced from "Tutorials Point"- https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm

[3] Figure 3: Taken from "The Inside Out Security Blog" - https://blog.varonis.com/the-definitive-guide-to-cryptographic-hash-functions-part-1/