COMPARATIVE ANALYSIS OF LINEAR PROBING, QUADRATIC PROBING AND DOUBLE HASHING TECHNIQUES FOR RESOLVING COLLUSION IN A HASH TABLE

Saifullahi Aminu Bello[1] Ahmed Mukhtar Liman[2] Abubakar Sulaiman Gezawa[3] Abdurra'uf Garba[4] Abubakar Ado[5]

Abstract— Hash tables are very common data structures. They provide efficient key based operations to insert and search for data in containers. Like many other things in Computer Science, there are tradeoffs associated to the use of hash tables. They are not good choices when there is a need for sort and select operations. There are two main issues regarding the implementation of hash based containers: the hash function and the collision resolution mechanism. The hash function is responsible for the arithmetic operation that transforms a particular key into a particular table address. The collision resolution mechanism is responsible for dealing with keys that hash to the same address. In this research paper ways by which collision is resolved are implemented, comparison between them is made and conditions under which one techniques may be preferable than others are outlined.

Index Terms— Double hashing, hash function , hash table, linear probing , load factor, open addressing, quadratic probing,

1 INTRODUCTION

THE two main methods of collision resolution in hash tables are are chaining (close addressing) and open addressing. The three main techniques under open addressing are linear probing, quadratic probing and double hashing. This research work consider the open addressing technique of collision resolution, namely, Linear probing, Quadratic probing and double Hashing. The algorithms were implemented in c++, and sample data was applied. Comparison of their performance is made.

1.1 Hash Function: a hash function is any well-defined procedure or mathematical function that converts a large, possibly variable-sized amount of data into a small datum, usually a single integer that may serve as an index to an array [1]. The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes.
A good hash function should

- be simple/fast to compute
- map equal elements to the same index
- map different elements to different indexes
- have keys distributed evenly among indexes

There are many types of hash functions, for the purpose of this research, division method is used. . In this method the returned integer, *x* is to be divided by M, the size of the table. The reminder, which must be between 0 and M-1, will be use to specify the position of *x* in the table.
h(x) = x mod M

1.2 Load factor: is the ratio *n/m* between *n*, number of entries and *m* the size of its bucket array. As we shall see later in this research work, with a good hash function, the average lookup cost is nearly constant as the load factor increases from 0 up to 0.7 or so. Beyond that point, the probability of collisions and the cost of handling them increases.

1.3Linear probing: when collusion occurs, the table is search sequentially for an empty slot. This is accomplished using two values - one as a starting value and one as an interval between successive values in modular arithmetic. The second value, which is the same for all keys and known as the stepsize, is repeatedly added to the starting value until a free space is found, or the entire table is traversed.
The algorithm for this technique is
newLocation = (startingValue + stepSize) % arraySize
the stepsize takes the following value: 1, 2, 3, 4,….
Given an ordinary hash function H(x), a linear probing function would be:

$$H(x,i) = (H(x) + i) \pmod n$$

1.4 Quadratic Probing:
Quadratic probing operates by taking the original hash value and adding successive values of an arbitrary quadratic polynomial to the starting value. The idea here is to skip regions in the table with possible clusters. It uses the hash function of the form:

$$H(k, i) = (h`(k) + i^2) \bmod m \quad \text{for } i = 0, 1, 2, \ldots, m\text{-}1$$

1.5 Double hashing: uses the idea of applying a second hash function h`(key) to the key when a collision occurs.
The result of the second hash function will be the number of positions from the point of collision to insert.
There are some requirements for the second function:

- it must never evaluate to zero
- must make sure that all cells can be probed

The probing sequence is then computed as follows

$$Hi(x)=(h(x)+ih`(x)) \bmod m$$

Where h(x) is the original function, h`(x) the second function, i the number of collisions and m the table size.
So the table is searched as follows

• *Saifullahi Aminu Bello is working with Kano University of Science and Technolgy, Nigeria. And is currently pursuing masters degree program in computer science and technology in Liaoning University of Technology, China. E-mail: saifullahiabel@yahoo.com*

H0=(h(x)+0*h`(x)) mod m
H1=(h(x)+1*h`(x)) mod m
H2=(h(x)+2*h`(x)) mod m

And so on.

2 EXPERIMENT AND ANALYSIS

To compare the performance of the open addressing techniques, we considered inserting student's registration numbers(an alphanumeric data type) in a hash table implemented using c++ programming language, to monitor the performance of the techniques as shown in the table below. We took note of the number of probes needed to resolve the collision occurred each time an insertion was made.

Table 1: result of number of probes by each algorithm on a sample data

REGISTRATION NUMBER	Linear probing (probes)	Quadratic probing (probes)	Double hashing (probes)
KUST/SCI/05/356	0	0	0
KUST/SCI/05/214	4	2	2
KUST/SCI/05/117	0	0	0
KUST/SCI/05/714	0	0	0
KUST/SCI/05/735	1	1	3
KUST/SCI/05/821	0	0	0
KUST/SCI/05/434	2	3	0
KUST/SCI/05/578	1	1	0

As the number of probes indicates the number of collisions, from the above table, linear probing has the highest number of probes followed by quadratic probing. Double hashing has the least number of probes hence minimum collisions. So, double hashing is the most efficient followed by quadratic probing.

2.1 ALGORITHM COMPARISONS

How could we qualify one algorithm is better than another? Primary concern could be the growth of runtime as input set becomes larger. The runtime can be dependent on comparisons made, number of statements executed and varying implementations on different machines.

Some programs or algorithms perform just fine with a small set of data to be processed. But they may perform very poorly with a large data set. It's useful to understand which programs and algorithms might exhibit this behavior and avoid potential problems. Here we are focusing on the rate of growth of required computations as the quantity of data grows.

Hash function is expected to be independent of the size of the table, but as collision is inevitable, that property is rarely achieved. As we have seen, the efficiency of linear probing reduces drastically as the collision increases. Because of the problem of primary clustering, clearly, there are tradeoffs between memory efficiency and speed of access.

Quadratic probing reduces the effect of clustering, but introduces another problem of secondary clustering. While primary and secondary clustering affects the efficiency of linear and quadratic probing, clustering is completely avoided with double hashing. This makes double hashing most efficient as far as clustering is concerned.

Since all the techniques are dependent on the number of items in the table, then they are indirectly dependent on the load factor. If load factor exceeds 0.7 threshold, table's speed drastically degrades. Indeed, length of probe sequence is proportional to (load Factor) / (1 – load Factor) value (D.G Bruno, 1999).

Quadratic probing tends to be more efficient than linear probing if the number of items to be inserted is not greater than the half of the array, because it eliminates clustering problem.

Based on the above analyses, the following table is deduced

Table 2: Summary of the algorithms performance

PROBING SEQUENCE	PRIMARY CLUSTERING	CAPACITY LIMIT	SIZE RES
Linear probing	Yes	None	None
Quadratic probing	No	$\lambda<$	M must b
Double hashing	No	None	M must b

At best case, each of the technique works at O(1). But this is only achieved when there is no collision. But as collision occurs, linear probing tends to be less efficient so is quadratic probing and double hashing.

3 CONCLUSION

Hashing is a search method used when sorting is not needed and when access time is of essence.Though Hashing is an efficient method of searching and insertion, there is always time-space trade off. When memory is not limited, a key can be used as a memory address, in that case, access time will be reduced. And when there is no time limitation, we can use sequential search, so there is no need of using a key as a memory address, thus, memory is minimized.

Hashing – gives a balance between these two extremes – a way to use a reasonable amount of both memory and time.The choice of a hash function depends on:

1. The nature of keys and the
2. The distribution of the numbers corresponding to the keys.

Best course of action:

3. separate chaining: if the number of records is not known in advance
4. open addressing: if the number of the records can be predicted and there is enough memory available

From what we have seen in this research work, load factor of open addressing is always less than or equals to 1. To achieve efficient insertion and searching the load factor should be less than 0.75 for linear probing and double hashing, and must be less than or equals 0.5 for quadratic probing.

Double hashing is the most efficient collision technique, when the size of the table is prime number and it avoids clustering. Quadratic probing is also efficient but only when the records to be stored are not greater than the half of the table. It has problem of secondary clustering where two keys with the same hash value probes the same position. Linear probing is easier to implement and work with, but its efficiency tends to reduce drastically as the number of records approaches the size of the array.

REFERENCES

[1] D.G BRUNO, (1999); "*DATA STRUCTURES AND ALGORITHM WITH OBJECT ORIENTED DESIGN IN C++*" (1* Ed). Addison Wesley Publishing Company-America. PP. 225-248.

[2] JOHN R. HUBBARD, (2000); "*DATA STRUCTURES WITH C++*" (1* Ed). McGraw-Hill Companies -New York. PP. 161-165.

[3] HERBERT SCHILDT(1998); "*C++:THE COMPLETE REFERENCE*"(3* Ed). McGraw-Hill Companies-Berkeley. PP. 833-841.

[4] Tenenbaum, Aaron M.; Langsam, Yedidyah; Augenstein, Moshe J. (1990), *Data Structures Using C*, Prentice Hall, pp. 456–461, pp. 472, ISBN 0-13-199746-7

[5] TCSS 342 Lecture Notes, 2005. University of Washington

[6] http://en.wikipedia.org/wiki/Open_addressing