Review and Comparison for Collision Resolution in a Hash Table

Aye Aye moe[a], Tin Tin Soe[b], Moe Moe Thein[a,b,*]

[a] *dawayeayemoee@gmail.com*
[b] *tintinsoe.kse2018@gmail.com*
[c] *cummthein74@gmail.com*
University of Computer Studies (Meiktila), Myanmar
University of Computer Studies (Meiktila), Myanmar
University of Computer Studies (Pyay), Myanmar

Abstract

Hash tables are quite common knowledge structures in computing. They supply economical key primarily based operations to insert and search for knowledge in containers hash table. In Computing, there are trade-offs associated to the utilization of hash tables. They are unhealthy selections once there's a necessity for type and choose operations. There are two main problems relating to the implementation of hash table analysis: the hashes operate and therefore the collision resolution mechanism. The hashes operate may be technique for the mathematical process that transforms a selected key into a selected table address. The collision resolution mechanism is performed for coping with keys that hash to identical address. During this paper, ways in which collision(linear probing, quadratic probing and double hashing) is resolved or enforced, comparison between them is created and conditions beneath that one technique is also desirable than others are printed.

Keywords**:** Hashing, Hash Function, Hash Table, Collision Resolution Strategies, Open Addressing Strategy, Load Factor;

1. Introduction

An Information structure might be a selected approach to storing and organizing data in a computer so that it can be used efficiently. In the algorithm analysis, data need to be adjusted data structures such as arrays, stacks, queues, trees, hashing and graphs. Different kinds of data structures are suited to several of applications. The two main ways of collision resolution in hash tables unit of measurement chaining (close addressing) and open addressing. The three main techniques beneath open addressing are linear probing, quadratic probing and double hashing. This paper work considers the open addressing technique of collision resolution, namely, linear probing, quadratic probing and double hashing. The algorithms were enforced in C++, and sample data information was applied. Review and comparison of their performance is created.
Hashing may be a technique that is used to uniquely identify a specific object from a group of similar objects. Hash table collision resolution techniques are largely advanced ideas for IT students. During this paper, hash table collision resolution approach is developed to supply a useful methodology for finding out basic operations that unit of measurement performed on Hash table.

The remainder of this paper is organized as follows. In Section 2, Hashing and hash function are explained. In Section 3, Collision resolution techniques of the study are presented. Section 4, Review and comparisons area unit explained. In Section 5, attracts Conclusions.

1.1 Hashing and Hash Function

For Hashing may be a technique that is used to uniquely identify a specific object from a group of similar objects. Assume that you simply have associate degree object and you wish to assign a key to it to create looking out simple. To store the key/value pair, you will use an easy array like a data structure where keys (integers) can be used directly as an index to store values. However, in cases where the keys are large and cannot be used directly as an associate degree index, you ought to use hashing.

In hashing, giant keys area unit born-again into tiny keys by exploitation hash functions. A hash function is any well-defined procedure or mathematical relation that converts an oversized, presumably variable-sized quantity of knowledge into a little data point, typically one number that will function associate degree index to associate degree array. The values area unit then holds on during an organization known as hash table. The thought of hashing is to distribute entries (key/value pairs) uniformly across an associate degree array. Every part is assigned a key (converted key). By exploitation that key you will access the part in O(1) time. Using the key, the algorithm (hash function) computes an associate degree index that implies wherever associate degree entry will be found or inserted. Hashing is enforced in two steps: An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it.

- An element is converted into an associate degree number by using a hash function. This part will be used as associate degree an index to store the initial part,, which falls into the hash table.
- The element is hold on within hash table wherever it will be quickly retrieved using hashed key.

  hash = hashfunc(key)

  index = hash % array_size

In this technique, the hash is freelance of the array size associate degree and it is then reduced to an index (a number between 0 and array_size - 1) by exploitation modulo operator (%). A hash function is any function that can be used to map a knowledge set of an arbitrary size to a knowledge set of a fixed size, which falls into the hash table. The values came by a hash function are called hash values, hash codes, hash sums, or just hashes. There are many types of hash functions, for the aim of this analysis, division technique is used. . During this technique the returned integer, *x* is to be divided by M, the size of the table. The reminder, that should be between 0 and M-1, are going to be accustomed specify the position of x within the table.

$h(x) = x \bmod M$

To achieve a decent hashing mechanism, it's necessary to own a decent hash function with the subsequent basic requirements:

- Easy to compute: It ought to be simple to compute and should not become a rule in itself.
- Uniform distribution: It ought to
- An event distribution across the hash table and will not result in clustering.
- Less collision: Collisions occur once pairs of components are mapped to an equivalent hash value. These ought to be avoided.

Hash functions are utilized in conjunction with hash tables to store and retrieve information things or information records. The hash function interprets the key related to every information or record into a hash code which is used to index the hash table. Once an item is to be added to the table, the hash code might index an empty slot (also called a bucket), in which case the item is added to the table there. If the hash code indexes a full slot, some reasonably of collision resolution is required: the new item could also be omitted (not added to the table), or replace the resent item, or it will be else to the table in another location by a given procedure. That procedure depends on the structure of the hash table: In *chained hashing*, every slot is the head of a linked list or chain, and items that collide at the slot are added to the chain. Chains may be kept in random order and searched linearly, or in serial order, or as a self-ordering list by frequency to speed hurry up access. In *open address hashing*, the table is probed ranging from the occupied slot in a given manner, typically by **linear probing, quadratic probing, or double hashing** until an open slot is located or the whole

table is probed (overflow).

1.2 Collision Resolution Techniques

All Hashing is a well-known searching approach. In this hashing, hash function is used to compute the hash value for a key. Hash value is then used as an index to store the key value in the hash table. When the hash value of a key maps to an already occupied bucket of the hash table, it is called as a Collision Resolution. Collision Resolution Techniques are the techniques used for resolving and handling the collision. When collisions occur, it is need to store the objects with colliding keys in alternative positions. Among the most widely used methods for resolving collisions is the method of *chaining*, in which the hash table positions are regarded as *buckets*, each one containing a pointer to a linked list (or other data structure) where the colliding objects will be located. It is a simple example to show that the average number of elements in each bucket is equal to the *load factor* of the hash table, defined as $\alpha=n/m$ for a hash table with m positions and n stored objects. Load factor is the ratio n/m between n, number of entries and m the size of its particular bucket array. As we shall see later in this paper work, with a best hash function, the average lookup cost is nearly constant as the load factor increases from 0 up to 0.7 or so. At that point of view, the probability of collisions and the cost of handling them increase.

If, when an element is added, it hashes to the same value as an already inserted element, then we have a collision and need to resolve it. There are several techniques for dealing with this:

-Separate chaining
-open addressing
    (1) Linear probing
    (2) Quadratic probing
    (3) Double hashing

Separate Chaining is an approach way to resolve collisions, but it has additional memory cost to store the structure of linked-lists. If entries datum are small (for instance integers) or there are no any values at all, then memory waste is corresponding to the dimensions of data itself. Once the hash table is predicated on the Open addressing strategy, all key-value pairs are stored in the hash table itself and there is no need for external organization.

1.2.1 Linear Probing

In open addressing, instead of in connected lists, all entry records area unit holds on at intervals the array itself. Once a new entry must be inserted, the hash index of the hashed value is computed and then the array is examined (starting with the hashed index). If the slot at the hashed index is unoccupied, then the entry record is inserted in slot at the hashed index else it yields in some probe sequence until it finds an unoccupied slot.
The probe sequence is that the sequence that's followed whereas traversing through entries. In various probe sequences, you may be able to have entirely completely different intervals between successive entry slots or probes. Once collusion occurs; the table is search consecutive for an empty slot. This is often accomplished using two values - one as a beginning value and one as an interval between consecutive values in standard arithmetic. The second value, which is the same for all keys and referred to as the step size, is repeatedly supplemental to the starting value until a free area is found, or the complete table is traversed.
Linear probing is once the interval between sequent probes is mounted (usually to 1). Let's assume that the hashed index for a selected entry is **index**. The inquisitor sequence for linear inquisitor will be:

index=index%hashTableSize
index=(index+1)%hashTableSize
index=(index+2)%hashTableSize
index = (index + 3) % hashTableSize and so on.

Example: Insert keys {89,18,49,58,69,78} with the hash function: h(x)= x mod 10 using linear probing. Use the table size 10.

Answer: -when x=89;
h(89)=89%10=9
insert key 89 in hash table in location 9
-when x=18;
h(18)=18%10=8
insert key 18 in hash table in location 8
-when x=49;
h(49)=49%10=9 (Collision)
so insert key 49 in hash table in next potential vacant location of 9 is 0.
-when x=58;
h(58)=58%10= 8 (Collision)
insert key 58 in hash table in next potential vacant location 8 is 1(since 9, 0 already contains values).
when x=69;
h(69)=69%10=9 (Collision)
insert key 69 in hash table in next potential vacant location 9 is 2(since 0,1 already contains values).
when x=78;
h(78)=78%10=8 (Collision)
search next vacant notice time for the table
which is 3 (since 0, 1, 2 contain values)
insert 78 at location 3.

Table 1: Hash table with keys mistreatment linear probing

0	49
1	58
2	69
3	78
4	
5	
6	
7	
8	18
9	89

Disadvantage of linear probing is:

-as long as table is sufficiently huge, a free cell frequently be found, however the time to undertake and do therefore can get quite huge.

-worse, though the table is comparatively empty, blocks of occupied cells begin forming. This result is thought as primary clustering.

-any key that hashes into the cluster would wish several makes an effort to resolve the collision, thus it'll argument the cluster.

1.2.2 Quadratic Probing

Quadratic probing is analogous to linear probing and then the entirely distinction is that the interval between consecutive probes or entry slots. Here, once the slot at a hashed index for associate entry record is already occupied, you would wish to start traversing until you discover associate unoccupied slot. The interval between slots is computed by inserting the sequent value of an arbitrary polynomial inside the initial hashed index.

Quadratic probing operates by taking the initial hash value and adding consecutive values of an arbitrary quadratic polynomial to the beginning value. The idea here is to skip regions inside the table with gettable clusters.

Let us assume that the hashed index for associate entry is **index** associated at **index** there's Associate in nursing occupied slot. The probe sequence is as follows:

index = index % hashTableSize
index = (index + 1^2) % hashTableSize
index = (index + 2^2) % hashTableSize
index = (index + 3^2) % hashTableSize
and so on.

Example: Insert keys ({89, 18, 49, 58, 69, 78} with the hash table size 10 using quadratic probing.
Solution: -when x=89;
h(89)=89%10=9
insert key 89 in hash table in location 9
-when x=18;
h(18)=18%10=8
insert key 18 in hash table in location 8
-when x=49;
h(49)=49%10=9 (Collision)
so use following hash function operate,
h1(49)= (9+1)%10=0
hence, insert key 49 in hash table in location 0
-when x=58;
h(58)=58%10= 8 (Collision)
so use following hash function,
$h_1(58)=(8+1)\%10=9$
again collision occur use once more the subsequent hash function,
$h_2(58)=(8+2^2)\%10=2$
insert key 58 in hash table in location 2
when x=69;
h(69)=69%10=9 (Collision)
so use following hash function,
$h_1(69)=(9+1)\%10=0$
again collision occurs use once more the subsequent hash function operate,
$h_2(69)=(9+2^2)\%10=3$
insert key 69 in hash table in location 3
when x=78;
h(78)=78%10=8 (Collision)
so use following hash function,
$h_1(78)=(8+10)\%10=9$; once more happens use yet again the next hash perform,

$h_2(78)=(8+2^2)\%10=2$;again collision happens figure following step
$h_3(78)=(8+3^2)\%10=7$
insert key 78 in hash table in location 7

Table 2: Hash table with search keys victimization quadratic probing

0	49
1	
2	58
3	69
4	
5	
6	
7	78
8	18
9	89

Quadratic probing may even be a collision resolution technique that eliminates the primary clustering problem take place in a linear probing. Although quadratic probing eliminates primary clustering, parts that hash to the constant location can probe the constant all completely different cells. This is often referred to as secondary clustering. Techniques that eliminates secondary clustering are available, the most fashionable is double hashing.

1.2.3 Double Hashing

Double hashing is analogous to linear probing and then the only distinction is that the interval between consecutive probes. Here, the interval between probes is computed by using two hash functions. Double hashing uses the concept of applying a second hash function h` (key) to the key once a collision happens. The results of the second hash function are the number of positions from the aim of collision to insert. There are some desires for the second function:

- it mustn't ever assess to zero
- have to be compelled to ensure that every one cells are probed

Let us say that the hashed index for an entry record is an index that is computed by one hash function and to boot the slot at that index is already occupied. You would like to start traversing in a specific probing sequence to appear for an unoccupied slot. The probing sequence will be:

index = (index + 1 * Hindex) % hashTableSize;
index = (index + 2 * Hindex) % hashTableSize; and so on.

In different methodology, to eliminate each sort of cluster the simplest way is double hashing. It involves two hash functions, $h_1(x)$ and $h_2(x)$, wherever $h_1(x)$ is primary hash function, is initial want to verify position of key and if it's occupied $h_2(x)$ is utilized. Example: $h_1(x)=x\%$ Tablesize; $h_2(x)= R-(x\%R)$, where R is prime

less than TableSize. So $h_i(x)=h_1(x)+i*h_2(x)\%TableSize$ is employed. The disadvantage of double hashing is it takes over time to work out hash function.

Example: Insert keys {89, 18, 49, 58} with the hash table size 10 using double hashing.

Solution:-when x=89;

h(89)=89%10=9

insert key 89 in hash table in location 9

-when x=18;

h(18)=18%10=8

insert key 18 in hash table in location 8

-when x=49;

h(49)=49%10=9 (Collision)

so use following hash function,

$h_1(49)=(9+1(7-(49\%7))\%10$

$=(9+(7-0))\%10=6$

hence insert key 49 in hash table in location 6

-when x=58;

h(58)=58%10=8 (Collision)

so use following hash function,

h1(58)=(8+1(7-(58%7))%10

=(8+(7-2))%10=3

insert 58 in the location 3.

Table 3: Hash table with search keys using double hashing

0	
1	
2	
3	58
4	
5	
6	49
7	
8	18
9	89

2. Review and Comparison Results

To compare the performance of the open addressing techniques, we have a tendency to thought-about considered adding student's registration numbers (an character set information type) in an exceedingly hash table enforced victimization C++ programming language, to watch the performance of the techniques as shown in the table-4 below. We have a tendency to take note of the amount of probes needed to resolve the collision occurred each time an insertion was made.

Table 4: results of variety of probes by every rule on a sample information

Registration number	Linear probing (probes)	Quadratic probing (probes)	Double hashing (probes)
K/FCS/05/356	0	0	0
K/FCS/05/214	4	2	2
K/FCS/05/117	0	0	0
K/FCS/05/714	0	0	0
K/FCS/05/735	1	1	3
K/FCS/05/821	0	0	0
K/FCS/05/434	2	3	0
K/FCS/05/578	1	1	0

As the variety of probes indicates the number of collisions, from the above table, linear probing has the very best variety of probes followed by quadratic probing. Double hashing has the smallest amount variety of probes hence minimum collisions. So, double hashing is the best followed by quadratic probing.
How may we have a tendency to qualify one rule is healthier than another? Primary concern can be the expansion of runtime as input set becomes larger. The runtime can be dependent on comparisons made, number of statements executed and varying implementations on completely different machines.

Some programs or algorithms perform simply fine with a little set of data to be processed. However they will perform very poorly with an outsized information set. It's helpful to understand which programs and algorithms might exhibit this behavior and avoid potential issues. Here we have a tendency to us that specialize in the speed of growth of needed computations because the amount of information grows. Hash function is expected to be independent of the dimension of the table, but as collision is inevitable, that property is not achieved. As we have seen, the efficiency of linear probing reduces drastically because the collision will increase. As a result of the matter of primary clustering, clearly, there are tradeoffs between memory efficiency and speed of access. Quadratic probing reduces the effect of clustering, however introduces another problem of secondary clustering. Whereas primary and secondary clustering affects the efficiency of linear and quadratic probing, clustering is totally avoided with double hashing. This makes double hashing best as far as clustering is concerned.

Since all the techniques are passionate about the amount of things within the table, then they are indirectly dependent on the load factor. If load factor exceeds 0.7 thresholds, table's speed drastically degrades. Indeed, length of probe sequence is proportional to (load Factor) / (1 – load Factor) value. Quadratic probing tends to be additional economical than linear probing if the number of items to be inserted is not greater than the half of the array, as a result of it eliminates clustering problem. Based on the above analyses, the subsequent table-5 is deduced.

Table 5: Summary of the algorithms performance

Probing Sequence	Primary Clustering	Secondary Clustering
Linear Probing	Yes	Yes
Quadratic Probing	No	Yes
Double Hashing	No	No

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

3. Conclusion

Hashing could be a search approach used once sorting isn't required and once interval is of essence. Although Hashing is associate economical methodology of looking out and insertion operations, there's continually time-space trade off. Once memory is not restricted, a key may be used as a memory address; in this case, the running intervals are going to be reduced. And once there is no time limitation, we will use consecutive search, thus there is no want of employing a key as a memory address, and thus, memory is reduced. Hashing gives a balance between these two ways – a simplest way to use an affordable quantity of each memory and time. The selection of a hash function depends on:
1. The character of keys and
2. The distribution of the numbers corresponding with the keys.
Best course of action:
3. Separate chaining: if the amount of records is not known in advance
4. Open addressing: if the amount of the records can be predicted and there is enough for memory on the market. In this paper work, load factor of open addressing is always less than or equals to one. To attain efficient insertion and searching operation 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 that the best collision resolution technique, once the scale of the hash table is prime number and it avoids clustering. Quadratic probing is also efficient but only when the records to be keep are not greater than the half of the table size. It's drawback of secondary clustering wherever two keys with the constant hash value probes the constant position. Linear probing is less complicated to implement and work with, however its potency tends to scale back drastically as the number of records approaches the scale of the array.

Acknowledgements

We thank the staff and our colleagues from the University of Computer Studies (Meiktila), Myanmar. This paper is supported by U Than Tun Naing, and by Daw Myat Myat Moe, Head of Natural Language in University of Computer Studies (Meiktila). I'd like to thank them for their constructive data and supporting.

References

[1] D.G Bruno, (1999); "*Data Structures and Algorithm With Object Oriented Design In C++"* (1* Ed). Addison Wes-ley 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 Scheldt (1998); *"C++:*
The Complete Reference" (3* Ed). McGraw-Hill Companies-Berkeley. PP. 833-841.
[4] TCSS 342 Lecture Notes, 2005. University of Washington.
[5] A. V. Aho, J.D. Ullman, and J.E. Hop croft, Data Structures and Algorithms, 1st ed. Addison-Wesley, 1983.
[6] http://en.wikipedia.org/wiki/Open_addressing