CHAPTER 1

Introduction

One of the fundamental problem in computer science is how to store information so that it can be searched and retrieved efficiently .We already have binary search trees that support operation such as INSERT, DELETE and RETRIEVAL in O (n log (n)) expected time in operations. So in many applications where we need these operations in that case hashing provides a way to reduce expected time to O (1).

The idea behind the hashing comes naturally if we approach the problem in the straightforward fashion and then work around the memory problem. Hashing is the most efficient scheme for locating and retrieving information‟s.

1.1 What is Hashing?

Hashing is a procedure that is used in sorting and retrieving the information about the database. This information is associated with key properties and makes use of individual character, numbers in the key itself. Hashing is a good technique for implementation in keyed tables [1].

In hashing the transformation of a string of characters into a frequently shorter fixed-length value or key that represents the original string is done. It‟s really tough to do the work in a faster manner like to discover the item using the shorter hashed key than to find it using the original value so for this reason hashing is very capable, so it is used to locate and retrieve items in a database. Moreover, it is also used in many encryption algorithms [2].

It is a technique used for storing and retrieving information (in main memory) as fast as possible and also used in performing optimal searches and retrievals because it increases speed, betters ease of transfer, get better retrieval, optimizes searching of data, reduces overhead. The main benefit of hashing is to optimize disk accesses and packing density. The packing density, approximately equal to a load factor. The main motive of hashing is to reduce disk space and access time by inserting and retrieving a record from the table in only one seeks. So for minimizing of this thing small hash table size must be used (that should be less than 10) [2] [3].

Hashing is a scheme of sorting and indexing data when we think about the case of databases. The hashing is mainly used to index the huge quantity of data using keywords or keys commonly created by complex formulas. Using hashing large amounts of information can be rapidly searched and listed.

When referring to security, hashing is a process of taking data, encrypting it, and creating unpredictable, irreversible output. There are many different types of hashing algorithms. MD2, MD5, SHA and SHA-1 are examples of hashing algorithms [4].

There are 4 key components involved in hashing:

1. Hash Table

2. Hashing and Hash Functions

3. Collisions

4. Collision Resolution Techniques.

The hash table is a storage location in memory or on disk that records the hashed values shaped by the hashing algorithm. Some storing known type of data is wanted. For creating a hash table certain number of buckets or storage locations.

Hashing is somewhat different from other type data structure such as binary trees, stacks, or lists because the data in a hash table does not have to be reorganized before being retrieved or inserted and in the other data structures, the items are stored either in the form of lists or trees. For larger data sets it can be a big problem, where a search and retrieval must travel through all nodes of a tree or all elements of a list. With a hash table, the size is set, so inserting or searching for an item is limited. On the other hand the time for storage and or retrieval with lists, trees or even stacks is related to the size of the data set [5].

All the element of data can be hashed in hash table and its size plays an important role in the efficiency of hash table. These tables contain a set number of buckets (storage spaces) and are stored in memory or on disk.

Items either strings or integers which are inserted into the hash table will differ and tackled in a diverse way. For example, if it is an integer it can be directly used by a hashing method to find a key. Alternatively, string item, is first converted to an integer value with the help of the ASCII conventions or some other consistently used technique (this is called 'preconditioning').

Preconditioning: Transforming a string to an integer with the help of ASCII conventions. String item has to be transformed to an integer prior to a key can be found. This process is known as preconditioning. Normally, ASCII conventions for transforming characters are used. ASCII values are assigned to the 26 letters starting at 11 (numbers 0 to 10 are first). Thus, 'A' is 11, 'B' is 12, and „C" is 13, and so on until 'Z' is 36. The numbers 37 and up are assigned to 'special characters' such as +, -, =, /, * and so on. For example, "Joe" is converted to

202515 using J=20, O=25 AND E=15.

A problem occurs in after preconditioning is that the consequential integer is too large to be stored in a table. To resolve this trouble we can use one hash process to attain a usable number and a second method to map the result to the table.

1.2 Features of Hashing

As hashing is the approach for storing and searching the data so the major working is done with the data .So main description of hashing are:

- ☐ Randomising: The spreading the data or records randomly over whole storage space.
- ☐ Collision: When two different key hashes to the same address space. This is the one major problem in hashing which will be discuses later chapter.

1.3 The Hash Table

The simplest way the hash table may be explained as a data structure that divides every element into equal-sized categories, or buckets, to permit quick access to the elements. The hash function finds that which element belongs to which bucket. A hash table can also be definite as data structure those acquaintances keys with values. The basic procedure is to support powerfully and finds the consequent value.

Basically it is the one-dimensional array indexed by an integer value computed by an index function called a hash function. Hash tables are sometimes referred to as scatter tables. Hash table are containers that represent a group of objects inserted at computed index locations. Each object inserted in the hash table is related with a hash index. The process of hashing involves the computation of an integer index (the hash index) for a given object (such as a string). If calculated correctly, the hash calculation (1) should be quick, and (2) when finished frequently for a set of keys to be inserted in a hash table should create hash indices consistently spread crossways the variety of index values designed for the hash table [6].

In algorithms a hash table or hash map is a data structure that uses a hash function to efficiently map certain identifiers or keys (student name) to associated values (e.g. that students enrolment no.).The hash function is used to transform the key into the index (the hash) of an array element (the slot or bucket) where the corresponding value is to be sought. Hash function are used to map each key to different address space but practically its not possible to create such a hash function that is able to do this and the problem called collision occurs. But still it is done up to greatest feasible case so that the chances of collision should be kept minimum. In a well-dimensioned hash table, the regular cost (number of instructions) for each lookup is self-determining of the number of elements stored in the table [8] [17].

But still having all the problems hash table are much more proficient in many cases comparative to all other data structures like search trees or any other table lookup structure. That the reason behind using hash tables in all kinds of computer software, particularly for associative arrays, database indexing, caches, and sets[8].

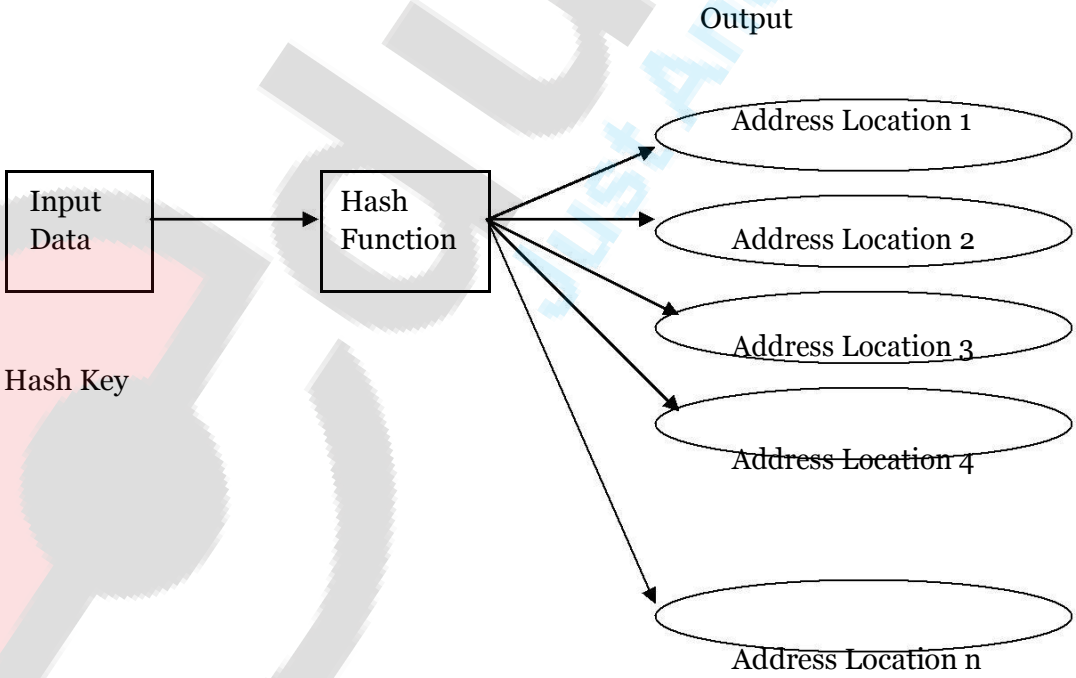

Figure 1.1.Hash functions and hash keys

When two different objects create the same hash index, it is referred as a collision.

Clearly the two objects cannot be located at the equivalent index position in the table. A collision resolution algorithm must be calculated to place the second object at a position separate from the first when their hash indices are alike.

The two primary problems associated with the creation of hash tables are:

1. The efficient hash function is designed so that it distributes the index values of inserted objects uniformly across the table.

2. The efficient collision resolution algorithm is designed so that it computes an alternative index for an object whose hash index corresponds to an object previously inserted in the hash table [8].

A hash table is an array based structure used to store “key, information” pairs. To accumulate an entry in a hash table, a hash function is functional to the key of the item being stored; frequent an index within the range of the hash table. The item is then stored in the table at that index position. Each index location in a hash table is called a bucket. To retrieve an item in a hash table, the same method is followed as used to store up the item.

Typical hash table operations are:

- *Initialization.*
- *Insertion.*
- *Retrieval.*
- *Deletion.*

1.3.1 Basic operation

Transforming the key into a hash, to situate the desired position by using a function, does working of hash table. A number that is used as an index in an array ("bucket") where the values should be. The number is transformed into the index by taking a modulo. The optimal hash function can vary widely for any given use of a hash table, however, depending on the nature of the key. Typical operations that can be done in hash table are insertion, deletion and lookup (although some hash tables are pre calculated so that no insertions or deletions, only lookups are done on a live system [7].

Problems for Which Hash Tables are not suitable are:

1. Problems for which data ordering is required.

   Certain operations are difficult and expensive to e implement because a hash table is an unordered data structure. Only if the keys are copied into a sorted data structure queries, proximity queries, sorted traversals and selection are possible. There are hash table implementations that keep the keys in order, but they are far from efficient.

2. Problems having multidimensional data.

3. Prefix searching especially if the keys are long and of variable-lengths.

4. Problems that have dynamic data:

Since open addressed hash tables are 1 Dim array its very difficult to be resized them, once they are allocated. Ahead of that implementing the table as a dynamic array and rehash all of the keys on every occasion the size changes can do it. This is an extremely luxurious operation. Separate-chained hash tables or dynamic hashing can be used as an alternate.

5. Problems in which the data does not have unique keys.

   If the data does not have unique keys open-addressed hash tables cannot be used. An alternative is use separate-chained hash tables.

1.3.2 How Hash Tables works?

A hash table works with transforming the key by means of a hash function into a hash, a number that is used as an index in an array to position the desired location where the values should be. Hash tables are generally used to implement many types of in-memory tables.

Mainly the hash tables are efficient for insertion of new entries, in expected time. Means it reduces the time for insertion. The main factor for the time spent in searching or the other operations involved in this are firstly the hash function and secondly the load on has table for both insertion and search approach time.

The most frequent operations on a hash table include insertion, deletion and lookup but some hash tables are pre calculated so the operations like insertions or deletions are not possible only lookups can be done on a live system. These operations are all performed in amortized constant time, which makes maintaining and accessing a huge hash table very efficient.

It is also likely to generate a hash table statically where for example there is a moderately restricted rigid set of input values such as the value in a single byte or perhaps two bytes from which an index can be constructed in a straight line. The hash table can also be used concurrently for tests of authority on the values that are disqualified.

Since two records cannot be stored in the same location so two keys hash cannot be indexed to the same location, an alternate location must be determined because two records cannot be stored in the same location .The process of finding an alternate location is called collision resolution. A collision resolution strategy ensures future key lookup operations that from no the query returns to the correct respective records and the problem of finding the same record on one location is solved.

A significant fraction of any hash table is selecting a resourceful collision resolution strategy. Consider the case imitative by means of the birthday paradox of a hash table containing million indices. Although a hash function be to output arbitrary indices homogeneously scattered over the array there is a 95% chance of a collision happening before it contain 2500 records. There are a number of collision resolution

techniques but the mainly admired are open addressing and chaining which will be

discussed in chapter 2 [9].

1.3.3 Advantages

- The main benefit of hash tables in excess of former table data structures is speed. This benefit is additional capable when the data is large (thousands or more). Hash tables becomes practically efficient when the greatest number of entries or the size of data is recognized or can be predicted in move forwards, so that the bucket array can be owed once with the most favourable size and there will be no require to be resized.

- One benefit of hashing is with the purpose of no index storage space is necessary, while inserting into other structures for instance trees does in general require an index. Such an index could be in the variety of a queue. In addition, hashing provides the advantage of rapid updates.

- The average lookup cost may reduce by a careful alternative of the hash function, bucket table size, and internal data structures if the set of key-value pairs is permanent and recognized earlier than instance (so insertions and deletions are not allowed). In particular, one may be able to plan a hash function that is collision-free, or even ideal. For this the keys need not be stored in the table [10] [11].

1.3.4 Disadvantage

- Hash tables are trickier to execute as compared to the efficient search trees. Choosing an effective hash function for a specific application is the mainly significant in creating hash table. In open-addressed hash tables it is fairly easy to create a poor hash function.

- Other problem in using hashing as an insert and retrieval tool is that it more often than not lacks locality and chronological retrieval by key. As result the insertion and retrieval becomes more indiscriminate.

- Another disadvantage is the inability to use duplicate keys. This is a problem when key values are very small (i.e. one or two digits).

- Even though operations on a hash table obtain constant time on regular, the charge of a good hash function be able to be considerably superior than the inner loop of

  the lookup algorithm for a in order list or search tree. Thus hash tables are not

  efficient when the number of entries is very tiny.

- Hash tables may be less efficient than tries for certain string processing applications, such as spell checking, Also, if every key is represented by a little sufficient number of bits, then, as an alternative of a hash table, one might use the key straight as the index into an array of values. Note that there are no collisions in this case.

- The entries stored in a hash table in a number of pseudo-random order can be enumerated powerfully. Therefore, there is no efficient way to efficiently situate an entry whose key is adjacent to a given key. Generally for separate sorting is required for catalogue all n entries in some specific order, whose cost is relative to log (n) for each entry. In contrast, ordered search trees encompass lookup and insertion cost proportional to log (n), but permit finding the adjacent key regarding the identical cost, and ordered enumeration of all entries at steady cost per entry. There may be no trouble-free approach to enumerate the keys, if the

keys are not stored (because the hash function is collision-free), that are present in the table at any known instant [12].

- ☐ Although the standard cost per operation is stable and moderately small but still the cost of a single operation could be rather high. Specifically an insertion or deletion operation might infrequently get time comparative to the number of entries, if the hash table uses dynamic resizing. This becomes a chief negative aspect in real-time or interactive applications [10].

- ☐ For the reason that hash tables cause access patterns that jump around, this be able to trigger microprocessor cache misses that cause elongated delays. Consequently in general hash tables demonstrate poor locality of orientation to be precise, the data to be accessed is scattered apparently at arbitrary in memory. Compact data structures for example arrays, searched with linear search, may possibly be faster if the table is moderately small and keys are integers or other small strings.

- ☐ Hash tables develop into quite inefficient when there are many collisions. While extremely uneven hash distributions are extremely unlikely to arise by chance, can cause excessive collisions, which may result in very poor performance (i.e., a denial of service attack) [13].

1.4 Hash Functions

Hash function is mathematical function or a process, which transform a huge, possibly variable-sized amount of data into a small, usually fixed-sized. The values get back by a hash function are called hash values or simply hashes, and usually take the form of a single integer represented in hexadecimal. Hash functions are most commonly used to speed up table lookup or data comparison tasks such as finding items in a database, detecting duplicated or similar records in a large file [7].

1.4.1 Choice of Hash Function

Choice of hash function is obviously is the matter of choice the need of problem means there are many parameters for choosing the hash function. But its not possible to choose exactly the perfect one because many problems are faced in selecting the hash function during the choice of hash function. So there may be three possible ways for it.

- ☐ Perfect Hash Function: There is no feasibility for this type of hash function if the data is large because practically it is not possible for huge data.

- ☐ Desirable Hash Function: For these hash function the address space should be small and collision should be kept very less or minimum.

- ☐ Trade-Off: But for above a tradeoffs should be maintained because for the larger data sets its very easy to avoid collision but the storage utilization becomes worst.

  So it‟s very important to maintain tradeoffs between them.

1.4.2 Choosing Hash Keys

One significant thought in selecting a hash key is the query design. In the predicates of queries there should be an EQUAL factor for each key column that will use the hash structure.

The subsequently significant concern is the allocation of key values. The most excellent key marks in a set of hash values that are consistently dispersed between the primary pages existing. The worst key marks in hash values that gather strongly in a fine range of primary pages, leaving others empty [7].

The next significant concern is that a key have to be unique. It possibly will be a unique single column value or a unique combination. Intended for constructing a key to be unique a hash key have to be non-volatile. when a key is need to be modified necessary only DELETE can be followed by an INSERT ,because the UPDATE statement can‟t be used by means of a hash key column. a range of columns can as well be used to generate a unique key, as in the subsequent example:

```
CREATE PUBLIC TABLE PurchDB.OrderItems
    OrderNumber      INTEGER          NOT NULL,
    ItemNumber       INTEGER          NOT NULL,
    VendPartNumber   CHAR(16),
    PurchasePrice    DECIMAL(10,2)    NOT NULL,
    OrderQty         SMALLINT,
    ItemDueDate      CHAR(8),
    ReceivedQty      SMALLINT
  UNIQUE HASH ON (OrderNumber, VendPartNumber) PAGES=101  IN OrderFS
```

For any hash table, the selected hash function has to be choosing for quick lookup, and it have to cause as minimum number of collision as it can. And if the keys are chosen in such a fashion that it is possible to get zero collisions this is called perfect hashing. Otherwise, the best you can do is to map an equal number of keys to each possible hash value and make sure that similar keys are not unusually likely to map to the same value.

1.4.3 Perfect Hashing

The hashing which ensures to get no more collisions at all is called as Perfect Hashing. A hash function that is injective that is, maps each valid input to a different hash values is said to be perfect. With such a function one can directly locate the desired entry in a hash table, without any additional searching.

The problem with perfect hash functions is that it is useful only in conditions anywhere the inputs are fixed and completely recognized in advance, such as mapping month names to the integers 0 to 11, or words to the entries of a dictionary. For the use in a hash table a suitable perfect function for a known set of n keys, can be establish in time relative to n, can be represented in less than $3n$ bits, and could be evaluated in a stable number of operations. Optimized executable code are emitted by the generators to estimate a perfect hash designed for a given input set .

CHAPTER 2

Problems in Hashing: Collision

2.1 Collision

Collision is the condition where two records are stored in the same location. But two records cannot be stored in same memory addresses; the process of finding an alternate location is called collision resolution. The impact of collisions depends on the application. For avoiding the collision hash functions should be choose cleverly.

- Checksums are the one that are designed to minimize the probability of collisions between similar inputs, without regard for collisions between very different inputs [14].

2.1.1 Hash collision

It is a condition in which a hash function gives the same hash code or hash table location for two different keys. Consider a case where two passwords encrypt to same value - thus there are two passwords that can be used to enter the system. Suppose there are numbers of forms that needs to be placed in sorted order by first letter of their surname. But if there are many people that have their names starts with same letter, then there will be more than one paper that needs to be stored in piles. In this case, the hashing system needs to cope with the hash collision described above. The first solution can be sorting them using second letter of the surname. Again there's a 1/26 possibility that there's more than one with same second letter.

2.1.2 Load factor

The presentation of Collision resolution methods does not depend openly on the number *n* of stored entries, but also dependent relative on the table's load factor. The load factor is the ratio *n/s* between *n* and the size *s* of its bucket array. The standard cost of lookup through a good quality hash function, is practically constant as the load factor increases from 0 up to 0.7 or so. Further than these points, the likelihood of collisions and their cost as well for handling them, both increase

As the load factor approaches zero, the size of the hash table increases with little improvement in the search cost, and memory is wasted.

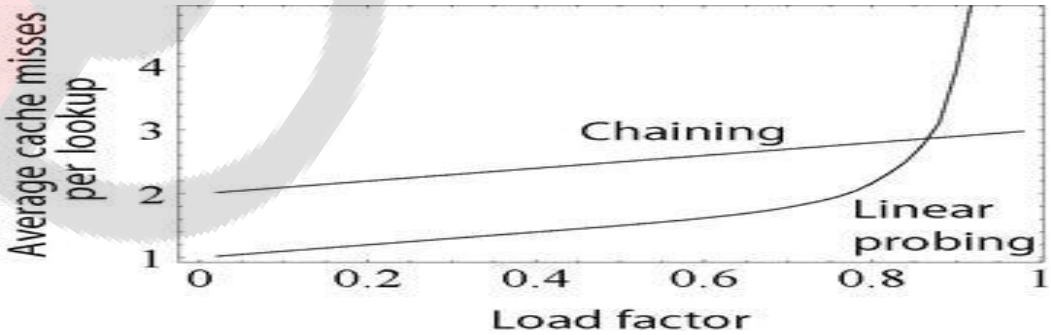

Figure 2.1 This graph compares the average number of cache misses required to lookup elements in tables with chaining and linear probing. As the table passes the 80%-full mark, linear probing performance drastically degrades [6].

2.2 Resolving Collisions

In collision resolution strategy algorithms and data structures are used to handle two hash keys that hash to the same hash keys. There are a number of collision resolution techniques, but the most popular are open addressing and chaining.

- ☐ Chaining: An array of link list application
  - o Separate chaining
  - o Coalesced chaining
- ☐ Open Addressing: Array based implementation o Linear probing (linear search)
  - o Quadratic probing (non linear search)
  - o Double hashing (use two hash functions)

2.2.1 By Chaining

Occasionally the chaining is also known as direct chaining; in its simplest form this procedure has a linked list of inserted records at every slot in the array references.

☐ **Separate Chaining**

Every linked list has each element that collides to the similar slot. Insertion need to locate the accurate slot, and appending to any end of the list in that slot wherever, deletion needs searching the list and removal.

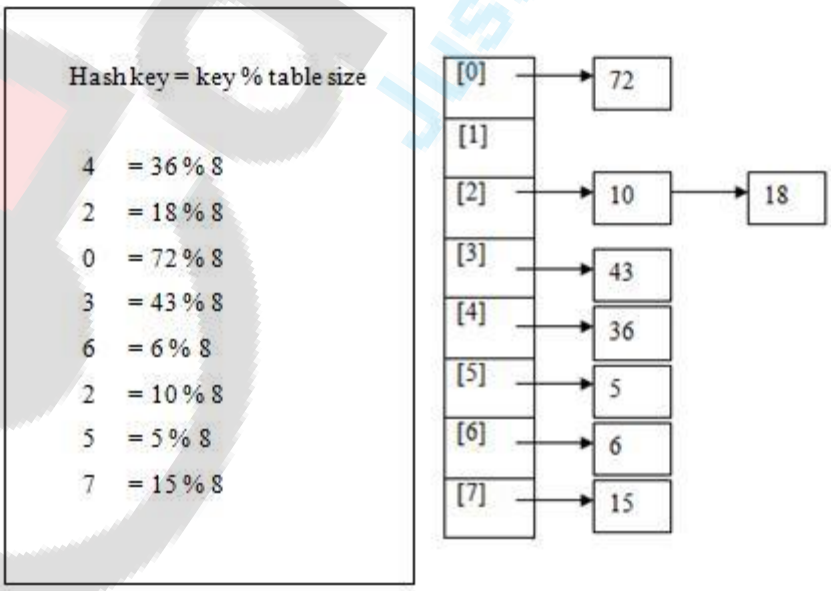

Figure 2.2: Separate Chaining

☐ **Coalesced Chaining**

Coalesced hashing is a scheme of collision resolution and it is a mix form of separate chaining and open addressing in a hash table. In a separately chaining a great quantity of recollection get wasted as in its hash table, items that hash to the same index are located on a list at that index, because the table itself have to be great enough to preserve a load factor that performs well (typically twice the expected number of items), and additional memory have to be used for all but the first item in a chain (unless list headers are used, in which case extra memory must be used for all items in a chain).

For example for a given sequence of randomly generated three character long strings, the following table would be generated with a table of size 10:

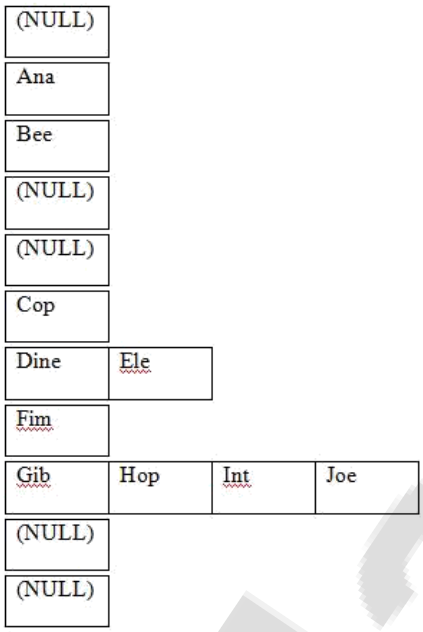

Figure 2.3 (a): Coalesced chaining

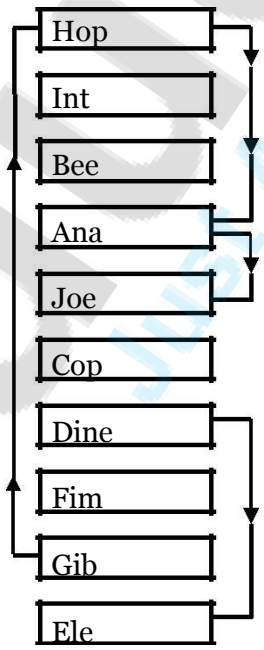

Figure 2.3 (b): Coalesced chaining

This scheme is successful, proficient, and very simple to put into practice. though, sometimes the additional memory employ might be a problem but an additional frequently used option is there, that is open addressing. It has a drawback that degrades the performance. Actually more specifically, open addressing has the

difficulty of primary and secondary clustering, where there are long sequences of used

buckets, and extra time is needed to calculate the next open bucket for a collision resolution.

Coalesced hashing is the one resolution to the clustering. A similar technique is used here as used in separate chaining, but as an alternative of locating a new nodes for the linked list, buckets are used in the table. The initial unfilled bucket in the table is called a collision bucket. When somewhere in the table collision occurs, the item is located in the collision bucket and a link is made connecting the colliding index and the collision bucket. After that to provide the next collision bucket, the next unfilled bucket is searched. As of this the consequence of primary and secondary clustering is minimized, and search times stay on well-organized. As the collision bucket moves in a expected prototype distinct to how the keys are hashed.

Coalesced chaining provides a good effort to avoiding the effects of primary and secondary clustering, and as a result can take advantage of the efficient search algorithm for separate chaining. For short chain, this strategy is very efficient and can be highly condensed, memory-wise [14].

2.2.2 Open Addressing

Open addressing hash tables be used to stock up the records straight inside the array. This approach is also known as closed hashing. This procedure is based on probing. A hash collision is resolute by probing, or searching through interchange locations in the array (the probe sequence) awaiting either the target record is establish, or an vacant array slot is establish, that‟s the sign of that there is no such key in the table [7].

Well known probe sequences include:

- **Linear probing** in which the interval between probes is fixed often at 1.

- **Quadratic Probing** in which the interval between probes increases proportional to the hash value (the interval thus increasing linearly and the indices are described by a quadratic function).

- **Double Hashing** in which the interval between probes is computed by another hash function.

The major tradeoffs between these methods are that linear probing has the best cache performance but is mainly responsive to clustering, even as double hashing has poor cache performance but exhibits nearly no clustering; quadratic probing cascade in-between in both areas. More computation is require in double hashing than other forms of probing.

A major influence open addressing hash table‟s performance is the load factor; that is, the proportion of the slots in the array that are used. As the load factor increases towards 100%, the number of probes that may be required to find or insert a given key raises abruptly. Probing algorithms may even fail to terminate, if once the table becomes full. Even with good hash functions, load factors are normally limited to 80%. A poor hash function can exhibit poor performance even at very low load factors by generating significant clustering. So both the load factor and hash function play important role here [7] [15].

☐ Linear Probing

Linear probing method is used for resolving hash collisions of values of hash functions by sequentially searching the hash table for a free location. This method's performance is more sensitive to the input distribution as compare to other methods like double hashing which will be discussed later.

The item will be stored in the next available slot in the table in linear probing also an assumption is made that the table is not already full. This is implemented via a linear search for an empty slot, from the point of collision. If the physical end of table is reached during the linear search, the search will again get start around to the beginning of the table and continue from there. The table is considered as full, if an empty slot is not found before reaching the point of collision,

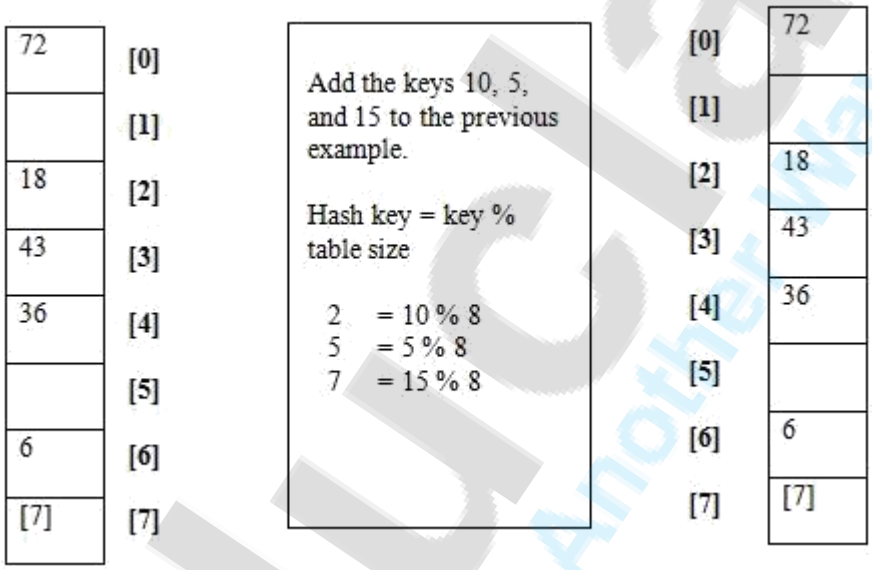

Figure 2.4: Linear Probing

Limitation:

A problem with the linear probe method is primary clustering. In primary clustering blocks of data may possibly be able to form collision. Several attempts may be required by any key that hashes into the cluster to resolve the collision.

☐ Quadratic Probing

To resolve the primary clustering problem, quadratic probing can be used.

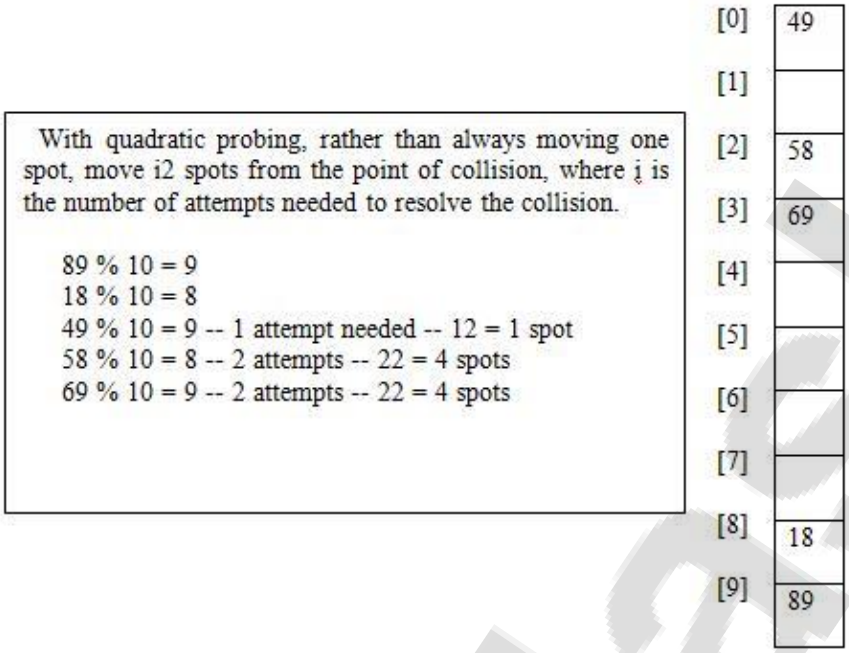

Figure 2.5: Quadratic Probing

Limitation:

Maximum half of the table can be used as substitute locations to resolve collisions.

Once the table gets more than half full, it"s really hard to locate an unfilled spot. This new difficulty is recognized as secondary clustering because elements that hash to the same hash key will always probe the identical substitute cells.

☐ **Double Hashing**

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

1. It must never evaluate to zero
2. Must make sure that all cells can be probed

A popular second hash function is: Hash2 (key) = R – (key mod R) where R is a Prime number smaller than the size of the table.

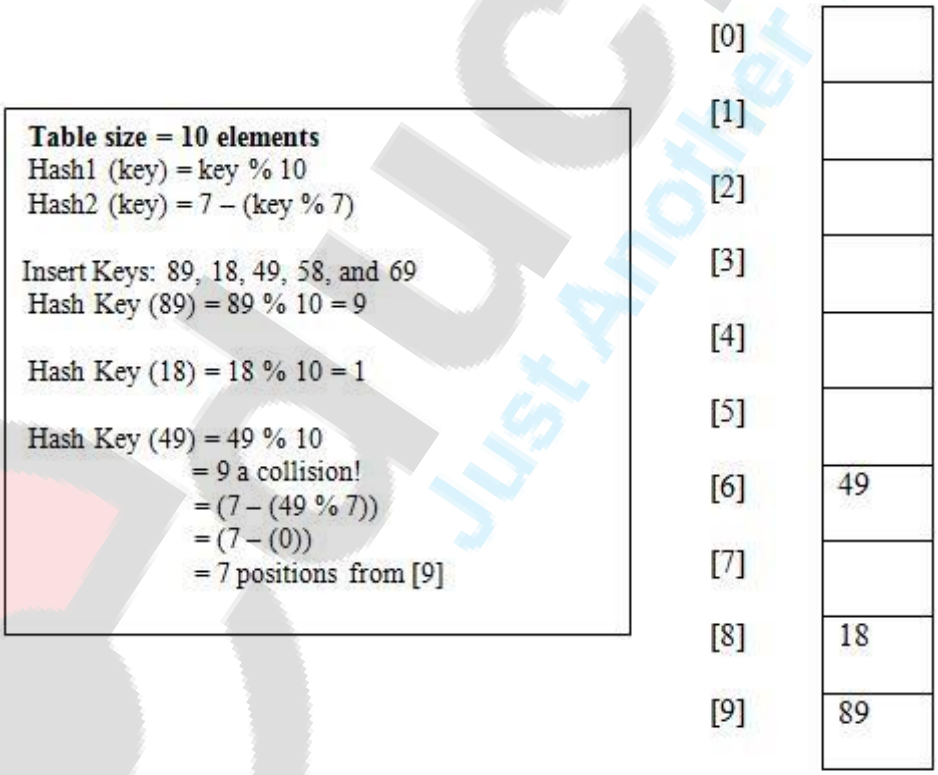

Figure 2.6 (a): Double Hashing

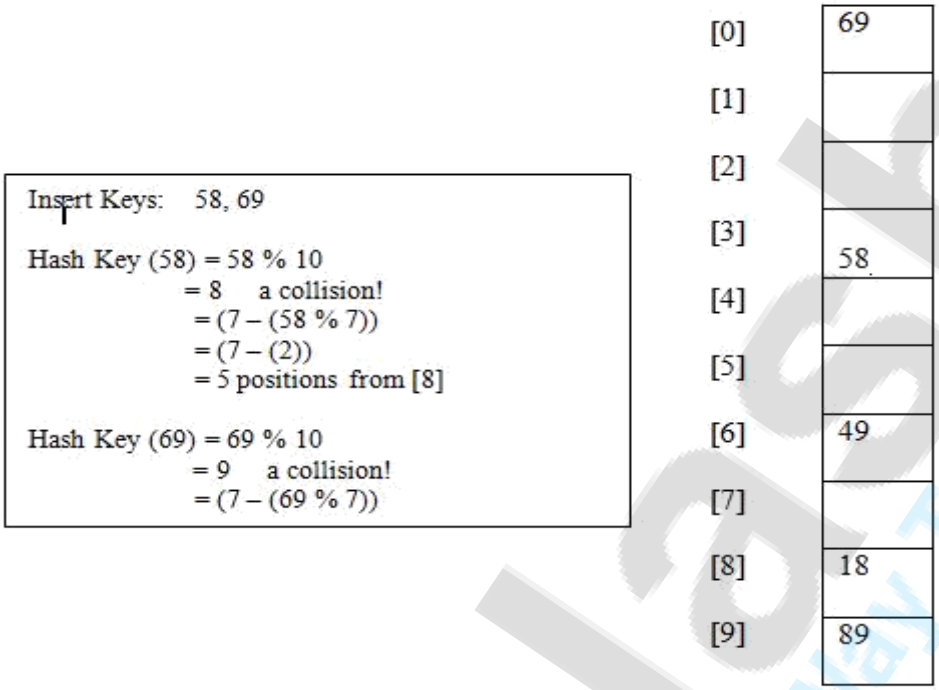

Figure 2.6 (b): Double Hashing

An efficient collision resolution strategy is an important part of any hash table. Regard as the subsequent case, derived using the birthday paradox, of a hash table containing 1 million indices. Although a hash function were to output random indices uniformly distributed over the array, there is a 95% chance of a collision occurring before it contains 2500 records [15].

2.2.3 Advantages over Open Addressed hash tables

The elimination function is straightforward and resizing the table can be delayed for a greatly longer time because performance degrade more gracefully even when every slot is used this is a chief benefit of chained hash tables above open addressed hash tables in that. In addition numerous chaining hash tables might not need resizing at all because performance degradation is linear as the table fills.

But besides that chained hash tables inherit the disadvantages of linked lists. When storing small records, the overhead of the linked list can be important. Traversing a linked list has poor cache performance is one more extra disadvantage of it