# Choosing Best Hashing Strategies and Hash Functions

Thesis submitted in partial fulfillment of the requirements for the award of Degree of

**Master of Engineering**

in

**Software Engineering**


By:

**Name: Mahima Singh**

**Roll No: 80731011**

Under the supervision of:

**Dr. Deepak Garg**

**Assistant Professor, CSED**

**&**

**Mr. Ravinder Kumar**

**Lecturer, CSED**



COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004


**JUNE 2009**

## Certificate

I hereby certify that the work which is being presented in the thesis report entitled, **"Choosing Best Hashing Strategies and Hash Functions"**, submitted by me in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Dr. Deepak Garg** and **Mr. Ravinder Kumar** and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

**(Mahima Singh)**

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

**(Dr. Deepak Garg)**
**Assistant Professor, CSED**
Thapar University
Patiala

**And**

**(Mr. Ravinder Kumar)**
**Lecturer, CSED**
Thapar University
Patiala

**Countersigned by:**

13/07/09.

**Dr. Rajesh Kumar Bhatia**
**Assistant Professor & Head**
Computer Science & Engineering. Department
Thapar University
Patiala.

15/7/09

**(Dr. R.K.SHARMA)**
**Dean (Academic Affairs)**
Thapar University,
Patiala.

## Acknowledgement

No volume of words is enough to express my gratitude towards my guide, **Dr. Deepak Garg Assistant Professor.** Who has been very concerned and has aided for all the material essential for the preparation of this thesis work. I would also thankful to **Mr. Ravinder Kumar** for his continual support, encouragement and invaluable suggestions towards the research area. They helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research-oriented venture.

I am thankful to **Dr. Rajesh Kumar Bhatia, Head of Computer Science & Engineering Department Thapar University Patiala** and **Mrs. Inderveer Channa, P.G. Coordinator** for providing us adequate environment, facility for carrying thesis work.

I would like to thank to all staff members who were always there at the need of hour and provided with all the help and facilities, which I required for the completion of my thesis.

I would also like to express my appreciation to my co-worker and my friends Jitender, Nupur, Mandeep, Aman, Mahima, Balaji for motivation and providing interesting work environment. It was great pleasure in working with them during this thesis work.

At last but not the least I would like to thank God and my parents for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

**Mahima Singh**

**Roll No. 80731011**

**M.E (Software Engineering)**

# Abstract

The most common use of hashing is in computing that one might expect hash functions is not well understood, and that choosing appropriate function should not be difficult. Recommendations are made for choosing hashing a strategy and hashing method and measuring its performance.

hashing. The report tries to find out the advantages and disadvantages of hashing. It discusses about hashing and its various components which are involved in hashing and states the need of using hashing i.e. for faster data retrieval. The report contains the study of hash table and the problem like collision which occur in managing the hash table and its solution to overcome the problem like separate chaining and open addressing, the various methods involve in open addressing like linear probing ,quadratic probing and double

It discusses the hash table, its basic operations and typical hash function operations and the problems for which hash tables are suitable and for which hash tables are not suitable. It contains the some common hashing methods like the division method, multiplication methods, folding method and other hashing method Midsquare, Addition, Digit Analysis which are used in different application of hashing.

And finally compare all the methods and techniques, so we can conclude that which methods are best suitable for which kind of problems. It helps to finding the best hashing technique and hash function is best for that problem. A hash function can be defined as a function that compresses a large message into a fixed small size 'message digest'. There are many hash function but the most widely use of them is in cryptographic applications. These cryptographic hash functions are based on the block ciphers

This thesis presents a survey on different types of the hash functions, different types of hashing methods ,hashing strategies and structural weaknesses of them or the

limitation of them that in which kind of problem they are suitable and for what they can’t be used.

Here comparison of various hash table techniques provides a great help for finding out the best suitable hashing for any problem. For it some popular techniques: open addressing, coalescent chaining and separate chaining are discussed. And all these comparison provides the base for our problem that is choosing the best hashing strategies and hash function.

# Table of Contents

**Certificate……** **i**

**Acknowledgment……** **ii**

**Abstract……...** **iii**

**Thesis Organisation……** **v**

**Table of Contents……** **vi**

**List of Tables……..** **ix**

**List of Figures……....** **x**

**Chapter 1:** Introduction

1.1 What is Hashing……... 1

1.2 Features of Hashing……..... 3

1.3 The Hash Table……. 3

1.3.1 Basic Operation…… 6

1.3.2 How Hash Table Works…… 7

1.3.3 Advantages…… 8

1.3.4 Disadvantages……... 8

1.4 Hash Function……... 10

1.4.1 Choice of Hash Function…….. 10

1.4.2 Choosing Hash Keys……. 11

1.4.3 Perfect Hashing……. 12

**Chapter 2 :** Problems in Hashing: Collision

2.1 Collision…… 13

2.1.1 Hash Collision…….. 13

2.1.2 Load factor…… 13

2.2 Resolving Collisions……. 14

2.2.1 By Chaining……. 14

2.2.2 Open Addressing……. 17
2.2.3 Advantages over Open Addressed Hash Tables…….. 21
2.2.4 Some Other Collision Resolution Techniques……. 22

**Chapter 3:** Hashing Types

3.1 Static Hashing……... 24
3.2 Dynamic Hashing……. 25
3.3 Characteristics of Good Hash Functions…….. 25
3.3.1 Basic Hashing Issues…… 25
3.4 Hashing Methods…….. 26
3.4.1 The Division Method ……..... 26
3.4.2 The Multiplication Method……..... 27
3.4.3 The Folding Method…… 28
3.4.4 Random-Number Generator…… 28
3.5 Other Hashing Methods…… 28
4.5.1 Midsquare…… 29
4.5.2 Addition……... 29
4.5.3 Digit Analysis…….. 30
3.6 Hashing Strategies ……... 30
3.6.1 Universal Hashing…… 30
3.6.2 Hashing with Polynomial…… 31
3.6.3 Cascade Hashing…….. 32
3.6.4 Cuckoo Hashing……..... 32
3.6.5 Geometric Hashing…….. 34
3.6.6 Cryptographic Hashing…… 35
3.6.7 Robust Hashing ……... 36

3.6.8 Bloom filters…… 37
3.6.9 String Hashing…… 38

**Chapter 4 :** Choosing Best Hashing Strategies and Hash Function

4.1 Comparisons between Linear probing and Double hashing……. 40
4.2 Comparisons: Open Addressing verses Separate Chaining…….. 40
4.3 Comparisons between Open Addressing Methods……... 40
4.4 Comparisons between the hashing methods……. 41
4.4.1 Chained Hashing……..... 41
4.4.2 Double Hashing……... 41
4.4.3 Linear Probing……. 41
4.4.4 Coalesced Chaining…….....…. 41
4.4.5 Cuckoo Hashing……... 42
4.4.6 Two-Way Chaining……. 42
4.5 What Constitutes a Good Hash Function……..... 43
4.6 Universal Hashing…… 44
4.7 Hashing with Polynomials…… 44
4.8 Cascade Hashing……... 45
4.9 Cuckoo Hashing…… 45
4.10 Geometric Hashing……. 46
4.11 Cryptographic Hashing……... 46
4.12 Robust Hashing…….. 46
4.12.1 Robust Audio Hashing……... 47
4.12.2 Robust Image Hashing……... 47
4.13 Bloom Filters…….. 47

4.14 String Hashing……..... 48
4.15 Best suited Hashing technique for a particular problem……. 48

**Chapter 5:** Results and Conclusion

5.1 Results……... 51
5.2 Conclusion…… 54

**Annexure**

I. References…… 56
II. Appendices……... 59
III. List of Publications……..... 63

# List of Tables

**Table 4.1:** Comparisons Table of Open Addressing methods……..... 40

**Table 4.2:** Best suited Hashing Technique for a particular problem……... 48

# List of Figures

**Figure 1.1** Hash Function and Hash Keys…… 5
**Figure 2.1** Load Factor……. 14
**Figure 2.2** Separate Chaining…….. 15
**Figure 2.3(a)** Coalesced Chaining…… 16
**Figure 2.3(b)** Coalesced Chaining……... 16
**Figure 2.4** Linear Probing…… 19
**Figure 2.5** Quadratic Probing……... 19
**Figure 2.6(a)** Double Hashing…….. 20
**Figure 2.6(b)** Double Hashing……. 21
**Figure 3.1** Cocukoo Hashing…… 33

# 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].

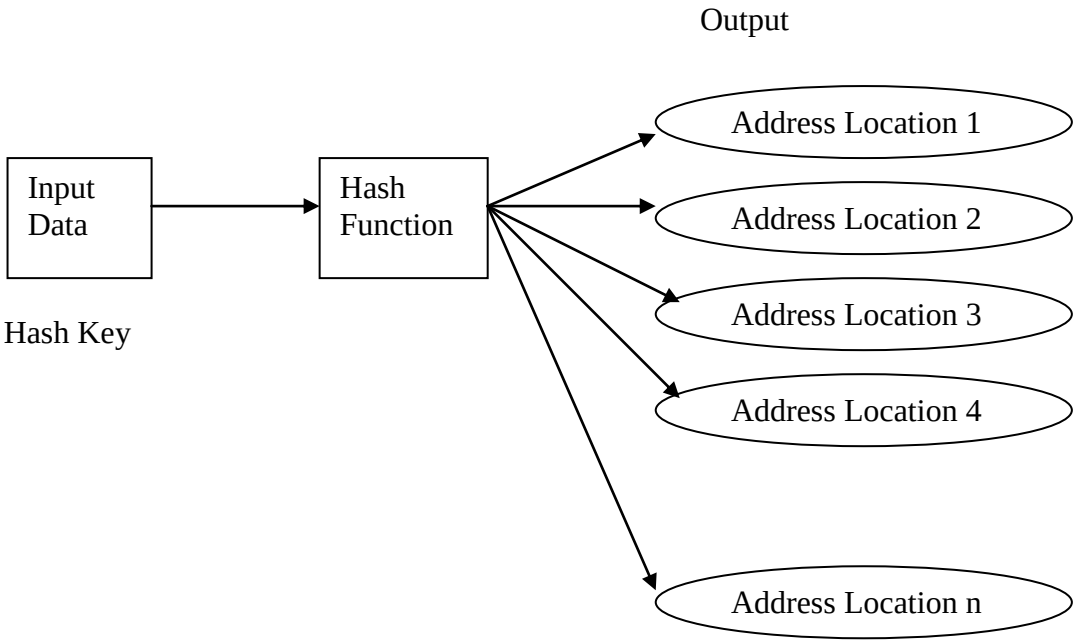


**Figure 1.1.Hash functions and hash keys**

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 [7].

# 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 [7].

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

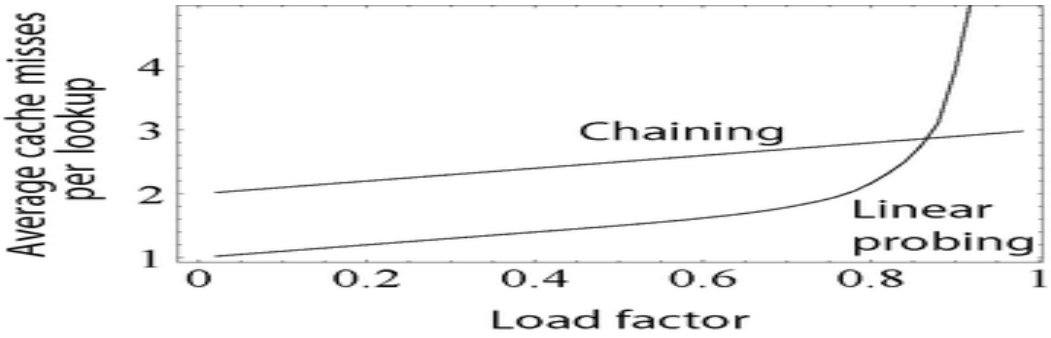


**Figure 2.1** This graph compares the average number of cache misses required to lookup elements in tables with chaining and linear probing. As the table passes the 80%-full mark, linear probing performance drastically degrades [6].

## 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
    - Separate chaining
    - Coalesced chaining
- Open Addressing: Array based implementation
    - Linear probing (linear search)
    - Quadratic probing (non linear search)
    - 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.

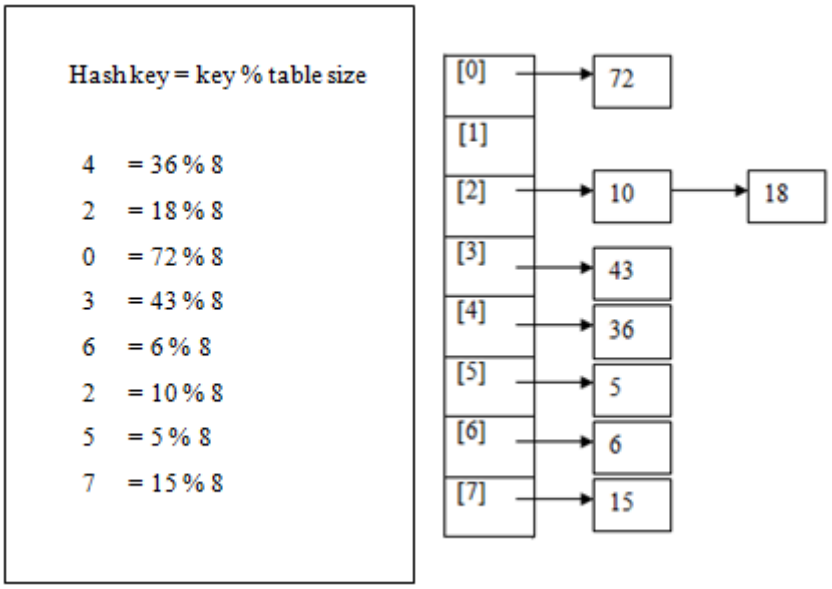


**Figure 2.2: Separate Chaining**

- **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:

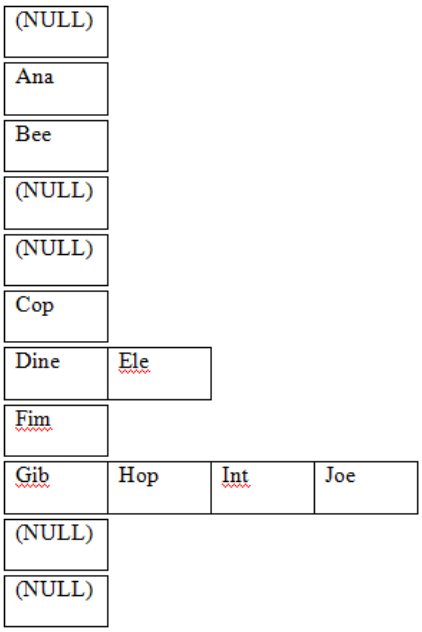


**Figure 2.3 (a): Coalesced chaining**

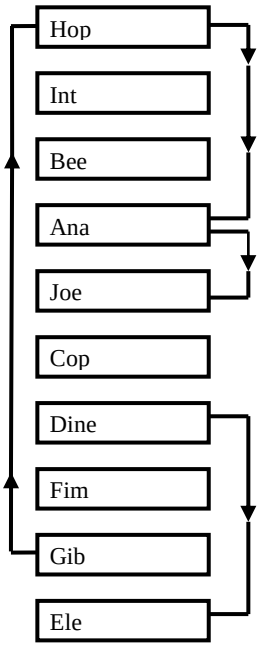


**Figure 2.3 (b): Coalesced chaining**

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,

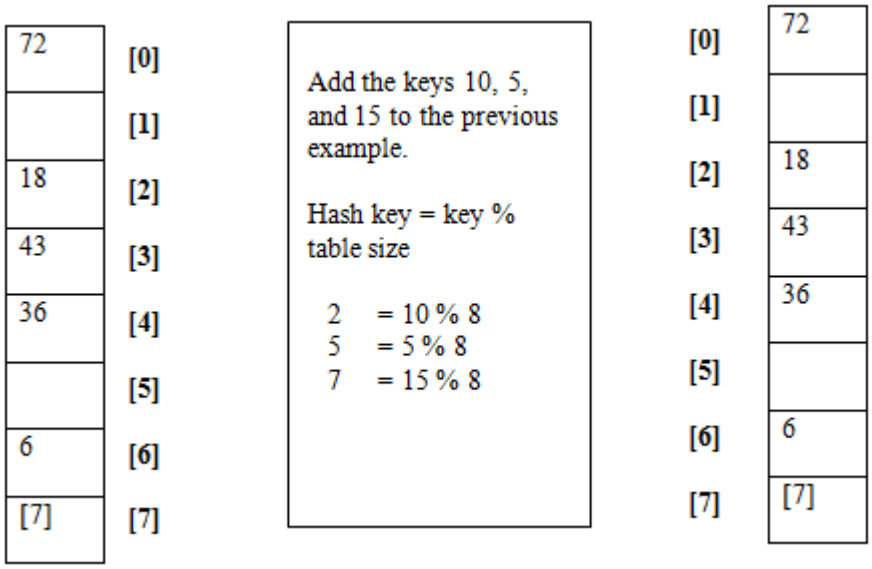


**Figure 2.4: Linear Probing**

**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.

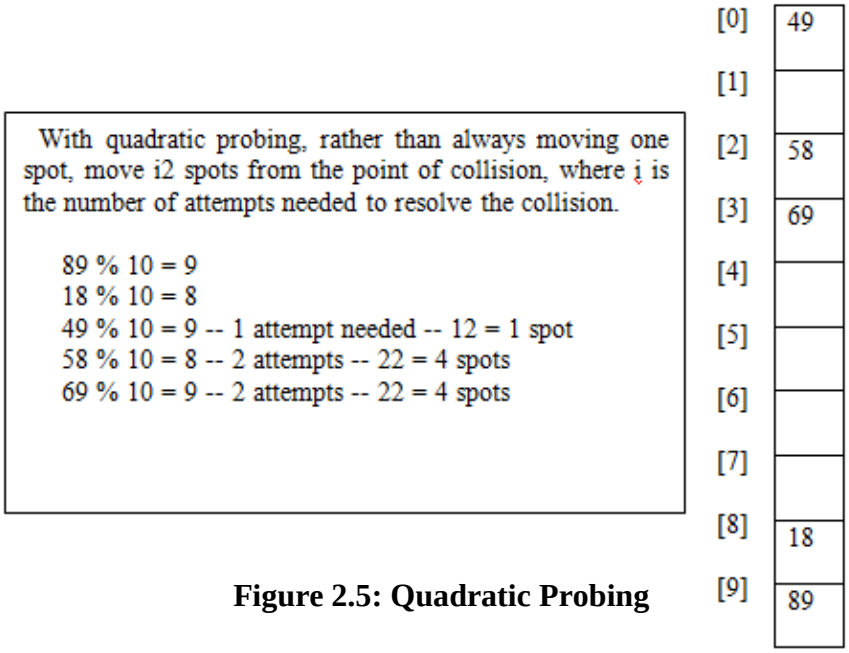


**Figure 2.5: Quadratic Probing**

**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.

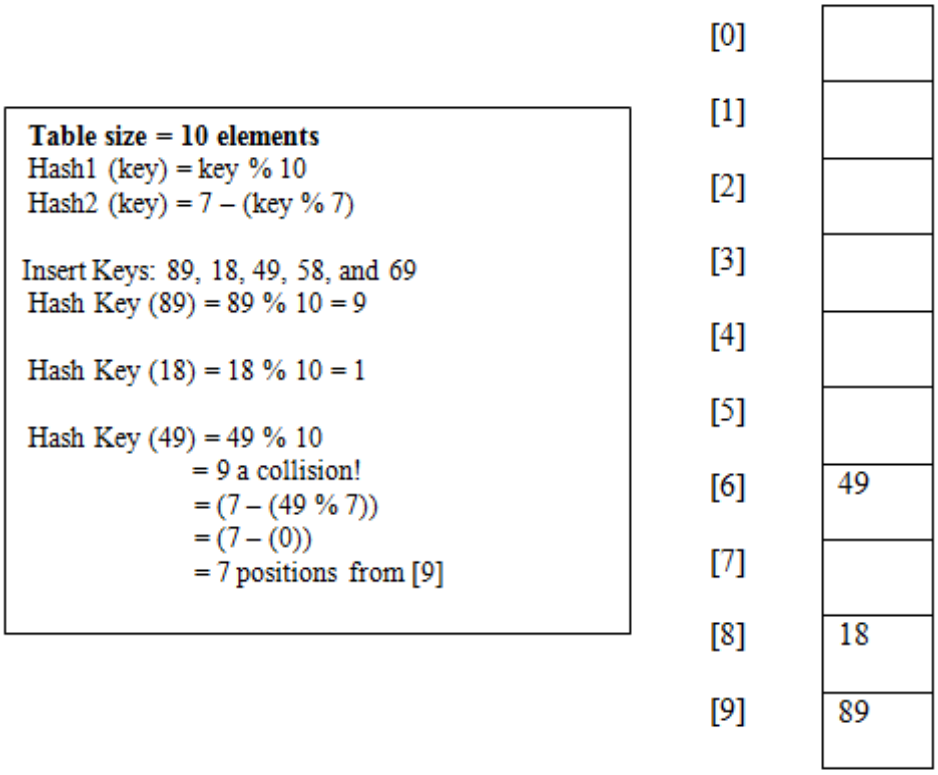


**Figure 2.6 (a): Double Hashing**

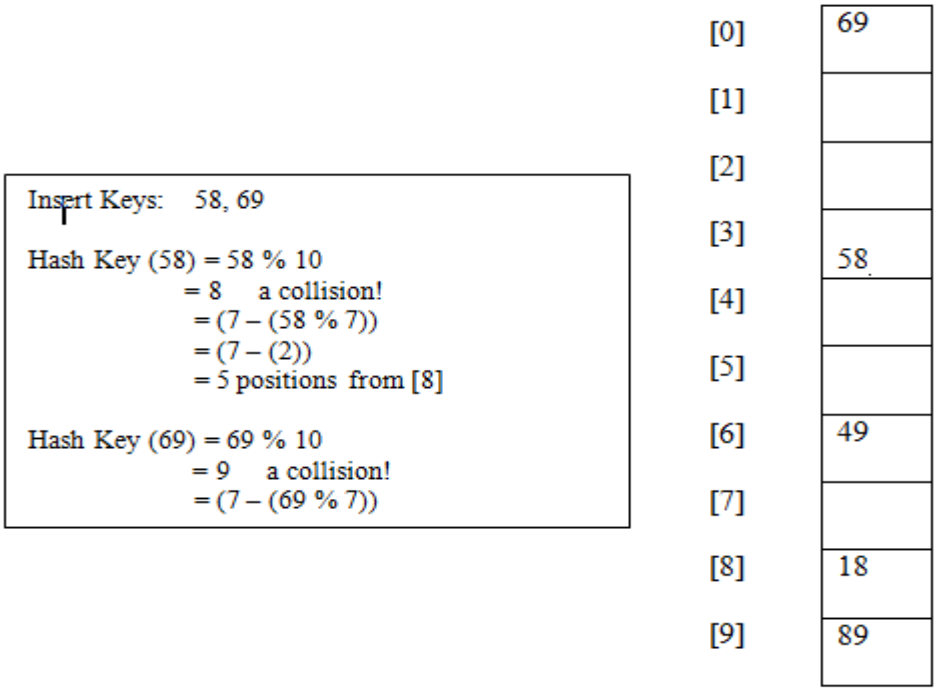


**Figure 2.6 (b): Double Hashing**

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.

When a collision occurs, elements with the same hash key will be chained together.

- A chain is simply a linked list of all the elements with the same hash key.
- The hash table slots will no longer hold a table element. They will now hold the address of a table element [7].

### 2.2.4 Some Other Collision Resolution Techniques

- **Cuckoo Hashing.**

In cuckoo hashing two memory accesses are used in lookups. After the first memory lookup becomes unsuccessful hashing is done by the second hash function. Comparatively it is much faster than chained hashing for small, cache-resident hash tables. Also the bucketized versions of cuckoo hashing (variants that use buckets that contain more than one key) is much more faster than conventional methods and it is used for large hash tables, when space utilization is high .When storing a large set of distinct keys, comparative to the other traditional hashing schemes tables, including linear probing ,the performance of the cuckoo hash table , for maintaining a large dictionary of integers in memory is recorded to be faster to build, search, and delete than the equivalent chained and array hash tables. Although the cuckoo hash table is space-intensive relative to the array hash table. But it was not as scalable, since the number of slots available, their capacity, and the extra vacant slots needed to prevent an irresolvable collision bound the total number of distinct keys storable. Hence, in order to cater for an unexpected increase in the number of distinct keys processed, for example, the cuckoo hash table will need to be re-sized which can be both expensive and space-intensive, particularly in a dynamic environment.

The bucketized cuckoo hash table was confirmed to be the slowest hash table, far inferior to both the chained and array hash tables if we consider the performance of hash tables under important skew access. Indeed, the greatest hash table for both distinct and skew data distribution was linear probing, though it too is not a scalable option relation to the array hash table; and even though more space-efficient than cuckoo hashing, it also requires a remaining of unfilled slots.

- **Two-Way Chaining**

The next method, Two-Way Chaining, can be described as two instances of chained hashing. A key is inserted in one of the two hash tables, namely the one where it hashes to the shorter chain. Two-way chaining is a novel hashing scheme that uses two independent truly uniform hash functions f and g to insert m keys into a hash table with n chains, where each key x is inserted into the shortest chain among the chains f(x) and g( x), breaking ties randomly. The two-way chaining paradigm is utilized to design efficient open addressing hashing schemes. If a longer list is needed, a rehash must be performed [32].

# CHAPTER 3
# Hashing Techniques

Hashing is a method to storing the data in an array so that storing, searching, inserting and deleting data is fast and efficient. The basic idea is not to search for the correct position of a record with comparisons but to compute the position within the array. All this is done on the basis of the given data, whether it is fixed or it can vary such as it will increase further or not, so that hashing can be implemented. There are two types of hashing: Static Hashing & Dynamic Hashing .If the data is fixed static hashing is beneficial but if its not fixed static hashing can give one problem so dynamic hashing is the next alternative for such type of data.

## 3.1 Static hashing

The set of keys is kept fixed and given in advance in static hashing. In static hashing the number of primary pages in the directory are kept fixed. Thus, when a bucket is full, an overflow bucket is needed to store any additional records that hash to the full bucket. This can be done with a link to an overflow page, or linked lists of overflow pages. The linked list can be separate for each bucket, or the same for all buckets that overflow. The original bucket is accessed first when search is done, and then the overflow buckets are accessed. There are also many keys that hash to the same bucket, locating a record may require accessing multiple pages on disk, which can affect the performance and can degrades it.

It have a constraint to the designer had to fix the size of the address space at file at the time of formation .All the parameters are kept constant such as capacity and no of bucket. However there a problem that is if the address space was selected extremely big, space would be exhausted and if the designer guessed too small, the number of overflows and the access time would rise that is one more additional disadvantage. The single solution was a luxurious reorganization of the complete file. The reorganization process first allocated a larger address space, then rehashed all the records (using a different hash function) to the larger address space, and finally released the old address space. Typically, this reorganization was a time-consuming process. Basically in static hashing there is a direct trade-off between space utilization

and access time in a dynamic environment. The smaller the space allocation, the more likely are overflows and poorer performance. The dynamic hashing schemes are used to reduce this space versus access-time trade-off and avoid the expensive remaking [16].

## 3.2 Dynamic hashing

The set of keys can change dynamically in this. Dynamic hashing is able to solve the problem of long searching of overflow buckets. In this type of hashing the size of the directory grows with the number of collisions to contain new records and avoid long overflow page chains. Extendible and Linear Hashing are two dynamic hashing techniques. Dynamic hashing has developed to the point at which useful implementations can be made. Dynamic hashing can expand and contract gracefully while maintaining reasonable access times and space utilization [18] [16] [33].

## 3.3. Characteristics of Good Hash Functions

A good hash function should following characteristics

- Minimize collision
- Be easy and quick to compute
- Distribute key values evenly in the hash table
- Use all the information provided in the key
- Have a high load factor for a given set of keys.

### 3.3.1 Basic Hashing Issues

How can a good hash function be found?

- The logic of the function must be clear and should be dependent upon the type of the key
- Besides it, the hash function should also depend upon the set of key values that will actually be hashed.
- The function should be made that it is easily computable and efficient too.

## 3.4 Hashing Methods

There are around seven common methods used in hashing, or the seven ways to insert values into a key accessed table. These are listed in no particular order that is given below.

- Division Method
- Multiplication Method
- Folding Method
- Length-dependent Method
- Midsquare Method
- Digit-Analysis
- Addition Method

This report discusses each of these with emphasis on the division, multiplication and folding methods. Also one method is there called as random method generator. The random number produced can be transformed to produce a valid hash value [7].

### 3.4.1 The Division Method

A hash function must guarantee that the number it returns is a valid index to one of the table cells. The simplest way to accomplish this is division method. In this an integer key is divided by the table size and the remainder is taken as the hash value.
It has been found that the best result with the division method is achieved when the table size is prime.

Algorithm: H ($x$) = $x$ mod $m$ + 1

Where: $m$ is some predetermined divisor integer (i.e., the table size), $x$ is the preconditioned key and mod stands for modulo.
Note that adding 1 is only necessary if the table starts at key 1 (if it starts at 0, the algorithm simplifies to (H ($x$) = $x$ mod $m$).
In the applet, we did not add 1.

So, in other words: given an item, divide the preconditioned key of that item by the table size (+1). The remainder is the hash key.

**Example:**

Given a hash table with 10 buckets, what is the hash key for 'Cat'?

Since 'Cat' = 131130 when converted to ASCII, then x = 131130.We are given the table size (i.e., m = 10, starting at 0 and ending at 9).

H(x) = x mod m

H (131130) = 131130 mod 10

= 0 (there is no remainder)

'Cat' is inserted into the table at address 0.

The Division method is distribution-independent.

### 3.4.2 The Multiplication Method

A different hash method is multiplicative method. This method is used in the applet. It multiplies of all the every single digits in the key jointly, and takes the remainder after dividing the resulting number by the table size.

In practical notation, the algorithm is:

H(x) = (a * b * c * d *....) mod m

Where: m is the table size, a, b, c, d, etc. are the individual digits of the item, and mod stands for modulo.

this can clearly understood this algorithm by applying it to an example.

**Example:**

Given a hash table of ten buckets (0 through 9), what is the hash key for 'Cat'?

Since 'Cat' = 131130 when converted to ASCII, then x = 131130

We are given the table size (i.e., m = 10).

The constant can be any number we want, let's use five (i.e., c = 5).

H (x) = (a * b * c * d *....) mod m

H (131130) = (1 * 3 * 1 * 1 * 3 * 0) mod 10

= 0 mod 10

= 0

'Cat' is inserted into the table at address 0.

Both of these Multiplication methods are distribution-independent.

### 3.4.3 The Folding Method

The folding method breaks up a key into precise segments that are added to form a hash value. And still another technique is to apply a multiplicative hash function to each segment individually before folding.

Algorithm: H (*x*) = (a + b + c) mod *m*

Where: a, b, and c represent the preconditioned key broken down into three parts, 'm' is the table size, and mod stands for modulo.

In other words: the sum of three parts of the preconditioned key is divided by the table size.

The remainder is the hash key.

**Example:**

Fold the key 123456789 into a hash table of ten spaces (0 through 9).

We are given *x* = 123456789 and the table size (i.e., *m* = 10).

Since we can break x into three parts any way we want to, we will break it up evenly.

Thus a = 123, b = 456, and c = 789.

H (*x*) = (a + b + c) mod *m*

H (123456789) = (123+456+789) mod 10

= 1368 mod 10

= 8

123456789 are inserted into the table at address 8.

The Folding method is distribution-independent.

### 3.4.4 Random-Number Generator

This is a scheme used for generating a pseudo-random numbers. Primarily, in digitized the state of a chaotic system is used to form a binary string. This binary string is subsequently hashed to construct a second binary string. This second binary string is now used to seed a pseudo-random number generator. The output of pseudo-random number generator is used in constructing a password or cryptographic key which is used in a security system.

The algorithm must ensure that:

- It always generates the same random value for a given key.

- It is unlikely for two keys to yield the same random value.

The random number produced can be transformed to produce a valid hash value.

Pseudo random number generators mix up a state like hash functions, but they don't take any input. They have a state of their own, and they just keep churning it. They produce random-looking sequences, which can be used as fake data [17].

## 3.5 Other Hashing Methods

A few of the many hashing methods that are also used for hashing are given below-

- Length-dependent Method
- Midsquare Method
- Digit-Analysis
- Addition Method

### 3.5.1Midsquare

In midsquare method, the key is multiply x by itself (i.e., x2) and select a number of digits from the middle of the result. How many digits you select will depend on your table size and the size of your hash key. If the square is considered as the decimal number, the table size must be a power of 10, whereas if it is considered as the decimal number, the table size must be a power of 2.Unforunately the midsquare. Method does not yield uniform hash values and does not perform as well as the multiplicative or the division method.

**For Example:**

To map the key 3121 into a hash table of size 1000, we square it 31212 = 9740641 and extract 406 as the hash value.

It can be more efficient with powers of 2 as hash table size. Works well if the keys do not contain a lot of leading or zeros. On-integer keys have to be pre-processed to obtain corresponding integer values.

### 3.5.2Addition

The sum of the digits of the preconditioned key is divided by the table size. The remainder is the hash key.

### 3.5.3. Digit Analysis

Certain digits of the preconditioned key are selected in a certain predetermined and consistent pattern.

There are many other hash functions each with its own advantages and disadvantages depending upon on the set of keys to be hashed. One consideration in choosing function is efficiency of calculation. It does not good to be able to find an object on the first try if that try takes longer than several trials in an alternative method. If keys are not integer they must be converted to the integer before applying one of foregoing hash function.

## 3.6 Hashing Strategies

### 3.6.1 Universal Hashing

For selecting a hash function say F universal hashing is used. Basically it is a randomized algorithm .The hash function F should have the following property: for any two distinct inputs x and y, the probability that F($x$) =F($y$) (i.e., that there is a hash collision between $x$ and $y$) is the same as if F was a random function. Therefore, if F has function values in a range of size r, the probability of any particular hash collision should be at most 1/r. There are universal hashing methods that give a function F that can be evaluated in a handful of computer instructions. Initially hashing started for implementing symbol tables and it is purely heuristic method. It plays an important role in many important constructions in abstract complexity theory and in cryptography too. And now, these constructions are generally used in many practices. Thus, having matured inside theory, hashing gets applied in ways the original symbol table implementers.

Randomized algorithms present a way to simply ensuring that choosing a random function from the class allows a proof that the probabilistic expectation for any set of inputs is that they will be distributed randomly. The main idea behind the universal

hashing is selecting a random hash function from a carefully designed class of hash function.Randamisation in some cases like quick sort ensures that no input will always evoke worst case behaviour. The algorithm can have different on each execution even with of same input that is possible only because of randomisation and hence can guarantee a good average case performance for the input.

Universal hashing has numerous uses in computer science, for example in cryptography and in implementations of hash tables. Since the function is randomly chosen, an adversary hoping to create many hash collisions is unlikely to succeed.

Universal hashing has been generalized in many ways, most notably to the notion of k-wise independent hash functions, where the function is required to act like a random function on any set of *k* inputs [2] [17].

### 3.6.2 Hashing with Polynomials

Basically it is the potential mathematical principles and structures that can provide the foundation for cryptographic hash functions, and also present a simple and efficiently computable hash function based on a non-associative operation with polynomials over a finite field of characteristic.

Polynomial hash functions are well recognized and frequently used in numerous applications. They become popular because of its certain performances they show. Even linear hash functions are expected to contain such performances. But, it's fairly advantageous that preferred hash functions to be reliable, i.e. they execute well with high probability; for several significant properties even higher degree polynomials were not known to be reliable. For some vital properties linear hash functions are not trustworthy. Even sometimes-quadratic hash functions might not be reliable. On the optimistic side, it is found that cubic hash functions to be reliable. But more generally, higher degree of the polynomial hash functions exhibits into higher reliability. There are some new class of hash functions, which enables to reduce the universe size in an efficient and simple manner. The reliability results and the new class of hash functions are used for some fundamental applications: improved and simplified reliable algorithms for perfect hash functions and real-time dictionaries, which use significantly less random bits, and tighter upper bound for the program size of perfect

hash functions. Hash functions are easy-to-compute compression functions that take a variable length input and convert it to a fixed-length output. Hash functions are used as compact representations, or digital fingerprints, of data and to provide message integrity [13].

### 3.6.3 Cascade Hashing

Cascade Hashing is a new dynamic hashing scheme, which is based on spiral storage. Actually multilevel double hashing schemes called cascade hashing. They use several levels of hash tables. In each table, we use the common double hashing scheme. Higher level hash tables work as fail-safes of lower level hash tables. This strategy is highly effective and efficient in reduce collisions in hash insertion. So it gains a constant worst case lookup time with a relatively high load factor (70%-85%).

In the implementation of M-level cascade hash table, M hash tables are there, and use limited double hashing in every level of table. Also the hash table size is half of its preceding hash table (the proportion 1=2 is chosen empirically). The total number of probes are limited and kept up to 12. Thus in every level, the probe number is p = 12=M. Here M is a factor of 12, so M 2 f1; 2; 3; 4; 6; 12g. If an item can't find a free cell in Level 1 in p probes, it will probe in Level 2, and if still with bad luck, it turns to search lower levels. If in any case crisis happens, the hash table can be enlarged and rehashed. The lookup procedure is similar to the insertion procedure. It also takes not more than 12 steps. Clearly, insertion and lookup both take at most 12 probes, so the time complexity of cascade hash table is O (1).
Specially, when M = 1, it's the ordinary (limited) double-hashing scheme. When
M = 12(one probe every level), it's the "multilevel adaptive hashing" scheme.

Obviously, if we permit a larger total probe count, we can achieve higher load factor. But the average speed will be slower. So a user can choose an appropriate configuration, which balances best between speed and space efficiency to him [20].

### 3.6.4 Cuckoo hashing

Cuckoo hashing is an efficient and practical dynamic dictionary .The main performance parameters are of course lookup time, update time, and space or the good memory utilization. It is best for the modern computer architectures and for

distributed environment and presents a significant improvement as compared to all other schemes. Basically for new generation a dynamic dictionary it is cuckoo hashing is the main basic concept which provides much better results.

The invariable factors concerned in cuckoo hashing are critical for numerous applications. Specifically, lookup time is a essential parameter. It is already known that, by using a simple universal hash function, the likely number of memory probes for all dictionary operations can be made arbitrarily close to 1 if a suitably sparse hash table is used. Therefore the challenge is to coalesce speed with a reasonable space usage that is effectively done by this method. Also cuckoo hashing is very simple to implement. The pattern of this hashing is shown in below example [34].

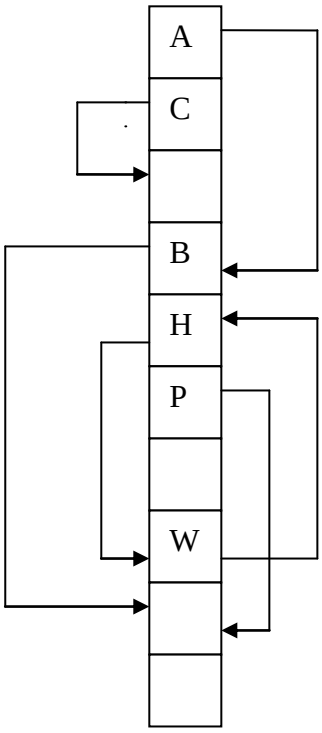


**Figure 3.1: Cuckoo hashing**

[The arrows show the alternative location of each key. A new item would be inserted in the location of A by moving A to its alternative location, currently occupied by B, and moving B to its alternative location which is currently vacant. Insertion of a new item in the location of H would not succeed. Since H is a part of cycle (together with W), the new item would get kicked out again]

These are the following properties of Cuckoo Hashing:

- Simple implementation.
- Lookups using two probes (optimal).
- Efficient in the average case.
- A practical, provably good hash function family is not known.

### 3.6.5 Geometric Hashing

Basically it is a general model based technique that cab be applied in different class of transformations. Here efficient partial matching between abject is done. So it has wide applications in various applications.

The basic concepts of geometric hashing are:

- Object feature Representation using the transformation invariants for the reorganisation of objects subject to any allowed transformation.

- For the efficient retrieval, these invariants are stored in hashing table, which is almost independent of the complexity of the model database.

- Robust matching is used which guarantees the reliable reorganisation.

This scheme is extensively used in computer graphics, computational geometry and in numerous other disciplines. Also used in solving numerous nearness problems in the plane or in 3-D space, for example in finding closest pairs in a set of points, comparable shapes in a list of shapes, comparable images in an image database, and so on. In these applications, the set of all inputs is some sort of metric space, and the hashing function can be interpreted as a dividing wall of that space into a grid of cells. The table is frequently an array having two or more indices (called a grid file, grid index, bucket grid, and similar names), and the hash function proceeds an index tuple. This exceptional case of hashing is known as geometric hashing and sometimes also called as the grid method.

The Geometric Hashing is the main paradigm for matching of a set of geometric features against a database as explained above. So very efficient in model based object recognition in computer vision like CT or MRI images of different persons, matching

of an individual fingerprint versus a database, matching the molecular surface of a receptor molecule against a data base of drugs and so on.

Only when the recognizable database objects have undergone transformations or when only partial information is present even at that time matching is possible. Also for low polynomial complexity this technique is highly efficient [21] [22].

### 3.6.6 Cryptographic hashing

It is the process that takes an arbitrary block of data and precedes a fixed-size bit string, the hash value, such that an accidental or intentional change to the data will change the hash value. The data to be encoded is called the "message", and their hash values are called the message digest or simply digest. Cryptographic hash function protect the integrity of data instead of protecting the integrity of arbitrary length of data, the whole emphasis is given to the protection of the integrity of significantly small bit strings. Therefore the data is hashed onto string or even the Cyclic Redundancy Checks (CRC). Basically CRC are used for the error detection but here it is assumed for the malicious and no longer random noise process [24].

The ideal cryptographic hash function has four main properties:

- it is easy to compute the hash value for any given message,
- it is infeasible to find a message that has a given hash,
- it is infeasible to modify a message without changing its hash,
- it is infeasible to find two different messages with the same hash.

Cryptographic hash functions have many information security applications, details are given below –

- in digital signatures,
- message authentication codes (MACs)
- And also in other forms of authentication.

Ordinary hash functions can also be used, to index data in hash tables; as fingerprints, for the detection of duplicate data or uniquely identify files; or as checksums for the detection of accidental data corruption. Also in information security contexts,

cryptographic hash values are sometimes called (digital) fingerprints, checksums, or just hash values, even though all these terms stand for functions with rather different properties and purposes.

Some minimum properties that a cryptographic hash function must have are:

- Preimage resistance: (One -way)

  given a hash h it should be hard to find any message m such that h = hash (m). This concept is related to that of one way function. Functions that lack this property are vulnerable to preimage attacks.

- Second preimage resistance(Weak Collision Resistance)

  given an input $m_1$, it should be hard to find another input, $m_2$ (not equal to $m_1$) such that hash($m_1$) = hash($m_2$). This property is sometimes referred to as weak collision resistance. Functions that lack this property are vulnerable to second preimage attacks.

- Collision resistance(Strong Collision resistance )

  it should be hard to find two different messages $m_1$ and $m_2$ such that hash ($m_1$) = hash ($m_2$). These type of pair is called a (cryptographic) hash collision, and this property is referred to as strong collision resistance. It requires a hash value at least twice as long as what is required for preimage-resistance, otherwise collisions may be found by a birthday attack.

These properties imply that not even a malicious adversary can replace or modify the input data without changing its digest. Thus, if two strings have the same digest, one can be very confident that they are identical [36].

### 3.6.7 Robust Hashing

- **Robust Audio Hashing**

Previously it was made for précis a long audio signal into a concise signature sequence that can be easily used to identify the original record. The output sequence is

denoted in the literature by alternate names, such as signature, fingerprint or perceptual hash values of the input that's called the robust audio hashing. The mapping tool of audio input to the signature is called perceptual hash function. Audio hashing methods were tested, to measure for robustness via non-malicious signal processing attacks on the one hand and to assess the uniqueness or randomness of the hash when audio files with different content [26].

- **Robust Image Hashing**

Basically it is related to cryptographic hash functions and it is sensitive only to perceptual change. Sign bit from the domain of Discrete Cosine Transform (DCT) s the basic concept for robust image hashing image robust hashing which is widely used in image processing and video processing, e.g. for compression or digital watermarking. The sign bit of this feature vector is extracted to form an intermediate hash. To derive a final hash the intermediate hash can be incorporated into some security mechanism. The advantages of the sign signal in DCT domain are verified in experiments evaluating robustness.

It is easy to be incorporated into image and video compression and watermarking due to the implementation in DCT domain. But subscriber cannot significantly alter some files without sacrificing the quality or utility of the data. This can be true of various files including image data, audio data, and computer code [27].

### 3.6.8 Bloom Filters

It allows for the state of existence of a very large set of likely type values to be represented with a comparably smaller piece of memory. Through the use of multiple distinct hash functions it can be achieved. And by allowing the result of a query for the subsistence of a particular type to have a certain quantity of fault. By varying the size of the table used for the Bloom filter and also by varying the number of hash functions this error can be controlled [28][29][30].

Some network-related applications in which bloom filters are used:

• Collaborating in overlay and peer-to-peer networks: Bloom filters can be used for summarizing content to aid collaborations in overlay and peer-toper networks.

• Resource routing: Bloom filters allow probabilistic algorithms for locating resources.

• Packet routing: Bloom filters provide a means to speed up or simplify packet routing protocols.
• Measurement: Bloom filters provide a useful tool for measurement infrastructures used to create data summaries in routers or other network devices.

### 3.6.9 String Hashing

String Hashing is fundamental operation which is widely used in applications where speed is critical. It is the process of reducing a pseudo- random number in specified range.

Following uses are given below of String Hashing-

- It is used in so many applications where fast access to distinct strings is required.
- It is mostly used for the data storage access and mainly within indexing of data and as a structural back end to associative containers (i.e.: hash tables)
- One more use of it is in spell checking [31].

# CHAPTER 4

# Choosing Best Hashing Strategies and Hash Function

Hash tables offer the much better results for constant-time searching of a large collection of data as compared to balanced binary search trees, which can be searched in $O$ (log $n$) time. So for the larger data sets its better to use hashing. But hashing is not a strategy to be employed lightly. If it is done well, it can easily outperform most other data structures in terms of search time; done poorly; its performance can degenerate to that of a linked list. So it's important to use the hashing scheme carefully. And for this it's necessary to choose an effective and efficient hash function for good use of hashing.

There are two criteria that should be considered when choosing hash functions for hash table.

1. Functions should be choosed in such a manner that approximate the assumptions of Simple Hashing i.e. that map roughly equal numbers of keys to each slot in the table and are unlikely to deviate from this behaviour for keys encountered in practice.
2. If the chosen hash table is open-addressed, it is ensure that the slot sequence for every key is of maximum length, eventually touching every slot in the table [15].

When a hash table technique is examined, the main characteristics, which are considered, are generally, in that order:

- the mean number of probes in a successful search (more rarely its standard deviation, or its full distribution),
- the (mean) number of probes before an unsuccessful search is detected,
- the amount of additional work needed to insert a new item,
- The impact of deleting an existing item, each event (hopefully) less frequent than the preceding ones.

These characteristics are generally expressed, exactly or approximately, in functions of "alpha" the load factor of the table. Where load factor is equal to the ratio of the number of items over the number of entries in the table.

So according to our need of problem and need technique can be choosed to avoiding the collision, which is the main problem of hashing. From all of above studies of the hashing methods and techniques, hash functions the conclusions come out.

## 4.1 Comparisons between Linear probing and Double hashing

The choice between linear probing and double hashing depends primarily on the cost of computing the hash function and on the load factor of the table. Both use few probes but double hashing take more time because it hashes to compare two hash functions for long keys.

## 4.2 Comparisons: Open Addressing verses Separate Chaining

It's somewhat complicated because we have to account the memory usage. Separate chaining uses extra memory for links. Although the Open Addressing extra memory implicitly within the table to terminate probe sequence. Open-addressed hash tables cannot be used if the data does not have unique keys. An alternative is use separate-chained hash tables.

## 4.3 Comparisons Table of Open Addressing methods

| **Linear Probing** | **Quadratic Probing** | **Double hashing** |
|---|---|---|
| Fastest among three | Easiest to implement and deploy | Makes more efficient use of memory |
| Use few probes | Uses extra memory for links and it does not probe all locations in the table. | Use few probes but take more time. |
| A problem occurs known as primary clustering | A problem occurs known as secondary clustering | More complicated to implement |
| Interval between probes is fixed - often at 1. | Interval between probes increases proportional to the hash value | Interval between probes is computed by another hash function |

**Table 4.1: Comparisons Table of Open Addressing methods**

## 4.4 Comparisons between the hashing methods:

**4.4.1 Chained Hashing**. Half of the allocated memory is used for the hash table, and half for list elements to increasing the performance and avoiding the collision. If the data structure runs out of free list elements, its size is doubled. We store the first key of each linked list directly in the hash table, as this often saves one cache miss. Having the first key in the hash table also slightly improves memory utilization, in the expected sense. This is because every non-empty linked list is one element shorter and because we expect more than half of the hash table cells to contain a linked list for the load factors considered here.

**4.4.2 Double Hashing**. For preventing the tables from clogging up with deleted cells, resulting in poor performance for unsuccessful lookups, all keys are rehashed when keys and "deleted" markers occupy 2/3 of the hash table. The fraction 2/3 was found to give a good trade off between the time for insertion and unsuccessful lookups.

**4.4.3 Linear Probing.** Hashing with linear probing was the best performer in terms of both time and space on all architectures. The reason for the space advantage is straightforward by have only one level of tables instead of two. And has was considerably faster, as far fewer rehashes were needed.

**4.4.4 Coalesced chaining.** This strategy is effective, efficient, and very easy to implement. Although sometimes the extra memory use might be prohibitive and for this problem the most common alternative is open addressing, but still its disadvantage is that it decreases the performance. More specifically, open addressing has the problem 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.

So to conquer the difficulty of clustering solitary solution can be coalesced hashing. Coalesced hashing uses a parallel technique as used in separate chaining, but in its place of allocating new nodes for the linked list, buckets in the table are used. The initial unfilled bucket in the table is considered a collision bucket. It avoids the property of primary and secondary clustering, and as a consequence, advantage of the capable search algorithm for separate chaining. As in open addressing, deletion from a

coalesced hash table is awkward and potentially expensive, and resizing the table is terribly expensive and should be done rarely, if ever [8].

**4.4.5 Cuckoo Hashing.** Here the two memory accesses are used in lookups. After the first memory lookup becomes unsuccessful hashing is done by the second hash function. Cuckoo hashing is much faster than chained hashing for small, cache-resident hash tables. Also the bucketized versions of cuckoo hashing are much faster than conventional methods also for large hash tables, when space utilization is high [14].

**4.4.6 Two-Way Chaining**. For inserting the keys, two independent uniform hash functions are used and each key is inserted on-line into the shorter chain, with ties broken down indiscriminately. The insertion time is still made constant, while the average search time cannot be more than twice of the average search time of conventional uniform hashing. Two-way chaining theory can be effectively used in obtaining many efficient algorithms the .It is somewhat nearer to cuckoo hashing but it is able to achieves constant worst-case insertion time, deterministically, and constant worst-case search time asymptotically almost definitely, when the load factor is made stable. Here the space consumption is linear as well.

After considering all the hashing methods any their strategy the common parameter comes out is space-time trade-off .It can be applied to all the problem of data storage, which the main building block for the hashing. So if we study about the data storage it will found that that if data is stored uncompressed, it takes more space but less time, than if the data were stored compressed (since compressing the data reduces the amount of space it takes, but it takes time to run the compression algorithm). Depending on the particular instance of the problem, either way is practical. Another example is displaying mathematical formulae on primarily text-based websites, such as Wikipedia. Storing only the Latex source and rendering it as an image every time the page is requested would be trading time for space - more time used, but less space. Rendering the image when the page is changed and storing the rendered images would be trading space for time - more space used, but less time. Also there are rare instances where it is possible to directly work with compressed data, such as in the

case of compressed bitmap indices, where it is faster to work with compression than without compression.

For good hash table performance a good hash function is essential. So good hash function should be able to make a tradeoffs between time and space. A poor choice of a hash function can lead to clustering, in which probability of keys mapping to the same hash bucket (i.e. a collision) is significantly greater than would be expected from a random function. A nonzero collision probability is practically impossible in any hash implementation, but it can be avoided or decreased. In addition, some hash functions are computationally expensive, so the amount of time (and, in some cases, memory) taken to compute the hash may be burdensome.

The most important thing in choosing a hash function is to make sure that it evenly spreads the space of possible keys onto the set of indices for the hash table. Secondly, it is advantageous if clusters in the space of possible keys are broken up so that we are likely to get a `continuous run'. Therefore, it is never a good idea to make the last (or even the first) few characters decisive, when defining hash functions of strings of characters. In order to avoid primary clustering, for choosing secondary hash functions, it has to make sure that different keys with the same primary position give different results when the second hash function is applied. Also one has to be careful to ensure that the second hash function cannot result in a number which has a non-trivial common divisor with the size of the hash table: Like for example if the hash table has size 10, and we get a second hash function which gives 2 as a result, then only half of the locations will be checked, which might result in failure (an endless loop, for example) while the table is still half empty. A simple way to avoid this is, always make the size of the table a prime number.

## 4.5 What Constitutes a Good Hash Function?

Before concluding further about hashing, consider in more detail the issue of choosing a hash function to perform the address calculations for a given application. However, this section presents a brief summary of the major concerns.

- A hash function should be easy and fast to compute.

If a hashing scheme is to perform table operations almost instantaneously, you certainly must be able to calculate the hash function rapidly. Most of the common

hash functions require only a single division (like the modulo function), a single multiplication, or some kind of "bit-level" operation on the internal representation of the search key. In all these cases, the requirement that the hash function be easy and fast to compute is satisfied.

- A hash function should evenly scatter the data throughout the hash table.

No matter what hash function you use, there is in general no way to avoid collisions entirely. (Of course, a perfect hashing function avoids collisions.) For example, to achieve the best performance from a separate chaining scheme should contain approximately the same number of items on its chain; that is, each chain should contain approximately N/Table Size items (and thus no chain should contain significantly more than N/Table Size items). To accomplish this goal, your hash function should scatter the search keys evenly throughout the hash table.
We cannot avoid collisions entirely [18].

Some more issues can be considered regarding a good hash function that is:

• Issues to consider with regard to how evenly a hash function scatters the search keys
  How well does the hash function scatter random data?
  How well does the hash function scatter non-random data?
•General requirements of a hash function
  The calculation of the hash function should involve the entire search key
  If a hash function uses modulo arithmetic, the base should be prime

## 4.6 Universal Hashing

Universal hashing is used widely in many computer science applications, for example in cryptography and in implementations of hash tables. Since in universal hashing the function is randomly choosed, there is some adverse hoping to create many hash collisions is unlikely to succeed [17].

## 4.7 Hashing with Polynomials

Basically it is the potential mathematical principles and structures that can provide the foundation for cryptographic hash functions, and also present a simple and efficiently computable hash function based on a non-associative operation with polynomials over a finite field of characteristic.

On the positive side, it is found that cubic hash functions are reliable. In a more general setting, higher degree of the polynomial hash functions translates into higher reliability. The reliability results and the new class of hash functions are used for some fundamental applications: improved and simplified reliable algorithms for perfect hash functions and real-time dictionaries, which use significantly less random bits, and tighter upper bound for the program size of perfect hash functions. Hash functions are easy-to-compute compression functions that take a variable length input and convert it to a fixed-length output. Hash functions are used as compact representations, or digital fingerprints, of data and to provide message integrity [13].

## 4.8 Cascade Hashing

Cascade Hashing is a new dynamic hashing scheme that depends upon spiral storage. Actually multilevel double hashing schemes called cascade hashing. They use several levels of hash tables. In each table, we use the common double hashing scheme. Higher-level hash tables work as fail-safes of lower level hash tables. By this strategy, it could effectively reduce collisions in hash insertion. Thus it gains a constant worst-case lookup time with a relatively high load factor (70%-85%) in random experiments. Different parameters of cascade hash tables are tested [20].

## 4.9 Cuckoo Hashing

The main concept behind this cuckoo hashing is to use two hash functions instead of only one. This locates two possible locations for each key in the hash table. To utilize a larger part of the capacity of the hash table efficiently while sacrificing some lookup and insertion speed generalizations of cuckoo hashing that uses more than 2 alternative hash functions can be expected. The use of just three hash functions enhances the load to 91%. The next generalization of cuckoo hashing consists in using more than one key per bucket. Using just 2 keys per bucket allows the load factor above 80% [34] [35].

The basic Cuckoo Hashing properties are:

- Simple implementation.
- Lookups using two probes (optimal).
- Efficient in the average case.

- A practical, provably good hash function family is not known.

## 4.10 Geometric Hashing

Geometric hashing is used in telecommunications (usually under the name vector quantization) to encode and compress multi-dimensional signals. The Geometric Hashing paradigm for matching of a set of geometric features against a database of such feature sets.

It is used in computer vision for matching geometric features against a database of such features, finds use in a number of other areas. When the recognizable database objects have undergone transformations or when only partial information is present even at that time matching is possible. For low polynomial complexity this technique is highly efficient [21][22].

## 4.11 Cryptographic Hashing

The ideal cryptographic hash function has four main properties:

- it is easy to compute the hash value for any given message,
- it is infeasible to find a message that has a given hash,
- it is infeasible to modify a message without changing its hash,
- it is infeasible to find two different messages with the same hash.

Cryptographic hash functions can be used in many information security applications, especially in digital signatures, message authentication codes (MACs), and other forms of authentication. to index data in hash tables; as fingerprints, to detect duplicate data or uniquely identify files, they can also be used as ordinary functions. Also as checksums to detect accidental data corruption these cab be good option. Indeed, in information security contexts, cryptographic hash values are sometimes called (digital) fingerprints, checksums, or just hash values, even though all these terms stand for functions with rather different properties and purposes.

## 4.12 Robust Hashing

Mainly robust hashing can be in mainly two fields that are in audio as well as in images too. So robust hashing is considered in two forms:

### 4.12.1 Robust Audio Hashing

It was made for précis a long audio signal into a concise signature sequence that can be used to identify the original record. The output sequence is denoted in the literature by alternate names, such as signature, fingerprint or perceptual hash values of the input that's called the robust audio hashing [26].

### 4.12.2 Robust Image Hashing

Sign bit from the domain of Discrete Cosine Transform (DCT) s the basic concept for robust image hashing. This technique is widely used in image processing and video processing, e.g. for compression or digital watermarking.
It is easy to be incorporated into image and video compression and watermarking due to the implementation in DCT domain. But subscriber cannot significantly alter some files without sacrificing the quality or utility of the data. This can be true of various files including image data, audio data, and computer code [27].

## 4.13 Bloom Filters

Bloom filter technique of hashing finds several applications, such as in efficient maintenance of differential files, space efficient storage of dictionaries, and parallel free-text searching It allows for the state of existence of a very large set of possible type values to be represented with a much smaller piece of memory. Through the use of multiple distinct hash functions it can be achieved. Bloom filters have a strong advantage over other data structures that is space, for representing sets, such as self-balancing binary search trees, tries, hash tables, or simple arrays or linked lists of the entries. Most of these require storing at least the data items themselves, which can require anywhere from a small number of bits, for small integers, to an arbitrary number of bits, such as for strings. One unusual property of bloom filter is that the time needed to either add items or to check whether an item is in the set is a fixed constant, completely independent of the number of items already in the set. No other constant-space set data structure has this property, but the average access time of sparse hash tables can make them faster in practice than some Bloom filters [28][29][30].

## 4.14 String Hashing

String Hashing is a fundamental operation used in many applications where fast access to distinct string is required. String hashing is the process of reducing a pseudo-random number in specified range. It is fundamental operation, which is widely used in applications where speed is critical. One more use of it is in spell checking [31].

## 4.15 Best suited Hashing technique for a particular problem

| | **Problems solved by hashing** | **Various types of Hashing used** |
|---|---|---|
| 1. | For traditional databases or in access method. The basic purpose of access methods is to retrieve and update data efficiently. Especially when record has to be retrieved in exactly one disk access [16][33]. | Dynamic hash functions |
| 2. | Password verification related applications. To authenticate a user, the password presented by the user is hashed and compared with the stored hash. This is sometimes referred to as one-way encryption [23].<br><br>For verification of message integrity. Determines whether any changes is to a message (or a file), by comparing message digests calculated before, and after, transmission [24].<br><br>For message digest for reliably identifying a file: several source code management systems, including Git, Mercurial and Monotone to uniquely identify them [25].<br><br>For security and performance reasons: most digital signature algorithms specify that only the digest of the message be "signed", not the entire message. Hash functions can also | Cryptographic hash functions |
| 3. | Problems like in computer graphics, computational geometry and many other disciplines. In these applications hashing function can be interpreted as a partition of that space into a grid of cells [21].<br><br>Problems of computer vision and structural alignment of proteins [22]. | Geometric hash functions<br>[Note: The technique is highly efficient and of low polynomial complexity.] |
| 4. | Problems related with content-based retrieval, monitoring, and filtering [27]. | Robust hash functions [Robust image hashing] |

| | | |
|---|---|---|
| | Problems of image processing and video processing, e.g. for compression or digital watermarking [27].<br><br>In authentication of both video data and still images and for integrity verification of visual multimedia [27]. | |
| 5. | Problems of most audio content identification systems [26].<br><br>For identification of the original record: summarizing a long audio signal into a concise signature sequence, which can then be used for identifying [26]. | Robust hash functions [Robust Audio Hashing] |
| 6. | Used in Web cache sharing*:* Collaborating Web caches use Bloom filters which make compact representations for the local set of cached files. Each cache periodically broadcasts its summary to all other members of the distributed cache [28].<br><br>Used in Query filtering and routing*:* The Secure wide-area Discovery Service, subsystem of Ninja project, organizes service providers in a hierarchy. Bloom filters are used as summaries for the set of services offered by a node [29].<br><br>Used in Compact representation of a differential file*:* a batch of database records are put in differential file contains to be updated.<br><br>Used in Free text searching*:* Basically, the set of words that appear in a text is succinctly represented using a Bloom filter [30].<br>For classification of various kinds of automobiles, for the purpose of re-detection in arbitrary scenes. The level of detection can be varied from just detecting a vehicle, to a particular model of vehicle, to a specific vehicle. | Bloom hash functions |
| 7. | Mainly within indexing of data and as a structural back end to associative containers (i.e.: hash tables) [31].<br><br>It *uses* special hash function for finding correct subtitles.<br><br>Used in spell checking. | String hash functions Used in the area of data storage access |

| | | |
|---|---|---|
| 8. | For the same theoretical properties as the classic dictionary a new hashing scheme called Cuckoo Hashing is used which is much more efficient than the traditional one. It combines the main two features speed with a reasonable space usage in more efficient manner [34]. | Cuckoo Hashing |

**Table 5.2: Best suited Hashing technique for a particular problem**

# CHAPTER 5
# Results and Conclusion

## 5.1 Results

After analysing all the methods and technique the results come out. Firstly we discussed the dynamic hashing scheme, which has an additional advantage that any record can be retrieved in exactly one disk access. It is used for traditional databases or in access method. The basic purpose of access methods is to retrieve and update data efficiently .So mainly in dictionaries where we need our results in one access we use this dynamic hashing scheme. The most efficient schemes in this class are linear hashing. But beyond this it has a fixed retrieval speed and the storage utilization is to be selected by the user. So mainly used in databases.

When we talked about the Cryptographic hash functions they are mainly used for the security reasons like information security applications, notably in digital signatures, message authentication codes (MACs), and other forms of authentication. As its major applications are Password verification related applications and to authenticate a user, the password presented by the user is hashed and compared with the stored hash. This is sometimes referred to as one-way encryption .For verification of message integrity cryptography can be used and determines whether any changes applied to a message or not, by comparing message digests calculated before, and after, transmission. It also used for message digest to identify a file in several source code management systems with reliability. Security and performance reasons like most digital signature algorithms specify that only the digest of the message be "signed", not the entire message. So it is concluded that cryptography is used for the problems in which security is the main concern.

Geometric hash functions are mainly developed in computer vision for matching geometric features against a database of having same features. Matching is possible only when partial information is present or the object recognisable database is transformed to some extent. The mostly computer vision research are done by the object recognization. For reorganizing an object, “object reorganization system” is developed so that it can be able to find the partially occlude object. So being above

reasons it can be used in problems like computer graphics, computational geometry and many other disciplines. In these applications hashing function can be interpreted as a partition of that space into a grid of cells or problems of computer vision and structural alignment of proteins.

Robust hash functions basically used in audio content identification systems as well as in image processing and video processing e.g. for compression or digital watermarking. Basically Robust image hashing is associated to cryptographic hash functions and it is responsive only to perceptual modifications and used in image processing and video processing, such as for compression or digital watermarking. Sign bit from the domain of Discrete Cosine Transform (DCT) s the basic concept for robust image hashing. Image hashing is also acknowledged as content-based digital signature, perceptual hashing, soft hashing, robust hashing, or image fingerprinting. The extracted hash could be used as a conventional digital signature, such as for authentication and tamper detection. The method gives good results but it depends on the intensity of the image. The reason of image hashing is the content classification. So it has to discriminate diverse content items. For implementing this objective discriminability of image hashes is essential to. The discriminability depends on the length of the considered hashes. The next significant factor influencing the discrimination is distance metric. The major three properties necessary for image hashing are: Robustness, Discriminability, and Security.

Robust audio hashing is based on a concise description of the time-frequency characteristics. It summarizes a long audio signal into a concise signature sequence, which is used to identify the original record. Robust hashing finds applications in database searching, broadcast monitoring, tamper proofing, data content authentication etc. For example, in database searching and broadcast monitoring, instead of comparing the whole sample set, hash sequence would suffice to identify the content. One more important application of this is in watermarking, where a content-dependent signature, coupled with ownership label is embedded in the document. This type of watermarking is resistant, among other things, to copy attack.

For database applications bloom filters are used very efficiently and in recent years they so become popular in the networking literature. A Bloom filter is a simple space-efficient randomized data structure for representing a set in order to support

membership queries. Bloom filters are used as summaries for the set of services offered by a node. So the problems in which it is beneficial to use bloom filters are web cache sharing, collaborating .Web caches use Bloom filters, which make compact representations for the local set of cached files. Each cache periodically broadcasts its summary to all other members of the distributed cache. Problems like query filtering and routing also use this technique. Bloom filters are used as summaries for the set of services offered by a node. Compact representation of a differential file is also one of an important use of it where a batch of database records are put in differential file contains to be updated. Free text searching is also a main implementation of bloom filter. Basically, the set of words that appear in a text is succinctly represented using a Bloom filter. It is also used for classification of various kinds of automobiles, for the purpose of re-detection in arbitrary scenes. The level of detection can be varied from just detecting a vehicle, to a particular model of vehicle, to a specific vehicle. Network-related applications are also one application of this. String Hashing is a fundamental operation used in many applications where fast access to distinct string is required. It is used in the area of data storage access like mainly within indexing of data and as a structural back end to associative containers. It uses special hash function for finding correct subtitles. String hashing is the process of reducing a pseudo- random number in specified range. It is fundamental operation, which is widely used in applications where speed is critical. One more use of it is in spell checking.

So from above discussion it is observed that for the different problems we need to choose a hash function which is most perfect to the problem and most efficient, like in databases we must choose the dynamic hashing especially when record must be retrieved exactly in one access and for security and performance reasons like message integrity, password verification one should go with the cryptographic hashing Similarly for image processing and for authentication for a video or a still image we go with the robust hashing is picked. Problem likes free text searching and redetection can be resolved by bloom hashing and for indexing of data string hashing will be most suitable. String hashing can solve problems like spell checking.

## 5.2 Conclusions

After the detailed study of hashing strategies and hash functions it is concluded that choosing a best hashing strategies and hash functions is purely concerned to the given problem. Choosing an effective hash function for a specific application is more an art than a science. So regarding to our problem we can choose the best-suited technique or hash function considering the above properties and the problems in which these techniques are used. We consider the problem and hashing strategies, then firstly we see the requirements of our problem, then we choose the best hashing technique. For example if we have less memory we cannot go with the separate chaining or the quadratic probing, it is advisable to choose linear probing. But if this constraint of less memory space is not there then we can go with the quadratic probing or the double hashing but again the double hashing become more complicated like we need few more probes required for it and hence more time will be consumed. If we consider the coalesced chaining, this strategy is effective, efficient, and very easy to implement, but however, sometimes the extra memory use might be prohibitive. And for this the most common alternative is open addressing, it degrades the performance to some extent. More specifically, open addressing has the problem of primary and secondary clustering.

Similarly choosing the best hash function a same criterion is considered like what is required and what kind of constraints we are having in our problem. For example if we are having problems like password verification, authenticate a user , verification of message integrity :determines whether any changes is to a message, for security and performance reasons, for message digest for reliably identifying a file: several source code management systems all problems related to network security we must use the cryptographic hash functions. For the problems like in computer graphics, computational geometry and many other disciplines geometric hash functions are best suitable. Also for the computer vision and structural alignments of proteins it can be used. Web cache sharing in which web caches, which uses bloom filters which make compact representations for the local set of cached files, in query filtering and routing, in Free text searching, for classification of various kinds of automobiles, for the purpose of re-detection in arbitrary scenes we cannot use such cryptography we should go with the other hashing function such as the bloom filters, it collaborate Web

caches which make compact representations for the local set of cached files. Each cache periodically broadcasts its summary to all other members of the distributed cache. Similarly string hash functions are used in the problems of spell checking and mainly within indexing of data and as a structural back end to associative containers. Basically string hash functions are used in the area of data storage access where fast access to distinct strings is required. For the problems related with content-based retrieval, monitoring, and filtering or the for compression or digital watermarking basically for the image processing and video processing we need the another type of hashing function called the Robust hash functions. [Robust image hashing]

This type of hashing function also solves the problems regarding the authentication of both video data and still images and for integrity verification of visual multimedia. Similarly for the most audio content identification systems robust hash function is used but this time robust audio hashing is beneficial. It is also good for summarizing a long audio signal into a concise signature sequence and its identification of the original record .For very simpler problems like for traditional databases or in access method we can easily go with the dynamic hash function. The basic purpose of access methods is to retrieve and update data efficiently. Especially when record has to be retrieved in exactly one disk access. But if we need much more efficient than the traditional one then we can choose cuckoo hashing having the same theoretical properties as the classic dictionary but combines the main two features speed with a reasonable space usage in more efficient manner.

So from above conclusion it can be seen very easily that choosing any hashing scheme or the hash function is entirely the matter of our need and requirements with the consideration of our boundation provided. Means it’s the choice of our requirements.

# ANNEXURE I

## References


1. Henry M. Walker, 'Abstract Data Types', Clarendon Press, 1988,4th Edition, Pg 129-143
2. Thomas H. Coremen, Charles E. Leiserson, Ronald L. Stein, "Introduction to Algorithms", 2nd edition, PHI, Pg 367,299,245,237,422,224,230
3. Robert Sedgewick,. Addison- Wesley, "Algorithms in C", 3rd edition, 1998,Chapter 14.
4. Josef Pieprzyk, Thomas Hardjono, Jennifer Seberry, "Fundamentals of computer security",Spinger, Pg 242-285
5. J. Zobel, S. Heinz, and H.E. Williams, "In-memory Hash Tables for Accumulating Text Vocabularies", Information Processing Letters, 271-277, December 2001
6. Mahima Singh, Deepak Garg, "Choosing Best Hashing Strategies and Hash Functions" ,IEEE International Advance Computing Conference (IACC'09) March 6-7 , 2009, Thapar University , Patiala.
7. Michael Main, "Data structures & other objects using Java", Addison Wesley, 1999 Original from the University of Michigan Digitized Nov 17, 2007
8. Richard Wiener, Lewis J. Pinson , "Fundamentals of OOP and data structures in Java" ,Cambridge University Press, 367-393
9. Dinesh P. Mehta, Sartaj Sahni , "Handbook of data structures and applications" ,CRC Press , Pg 1392.
10. Josef Pieprzyk, Babak Sadeghiyan, "Design of hashing algorithms" , Springer-Verlag, 1993 Original from the University of Michigan Digitized Nov 20, 2007
11. Yedidyah Langsam, Moshe J. Augenstein, Aaron M. Tenenbaum, "Data Structures Using C and C++",2nd Edition, Prentice Hall, 1996
12. Roy S. Ellzey , "Data structures for computer information systems", Science Research Associates, 1982 Original from the University of Michigan Digitized Nov 19, 2007,Pg 166

13. Crosby and Wallach's , “Denial of Service via Algorithmic Complexity Attacks”, Computational science and its applications: ICCSA 2006 , international conference, Glaskow, UK,
14. Frank Dehne, Jörg-Rüdiger Sack, Michiel Smid “Data Structure and Algorithms in java”,Springer
15. Michael B. Feldman , “Data Structure with ADA” , Reston Pub. Co., 1985 Original from the University of Michigann Digitized Jan 23, 2007, Pg 244
16. Fl J. Enbody H. C. Du , “Dynamic Hashing Schemes”, High performance computing for computational science, Vecpar 2002
17. Askitis, Nikolas (2009), "Fast and Compact Hash Tables for Integer Keys", Proceedings of the 32nd Australasian Computer Science Conference (ACSC 2009) 91
18. Paul Helman, Robert Veroff, Frank M. Carrano, Frank R. Carrano, “Intermediate Problem Solving and Data Structure’, 2nd Edition, Benjamin/Cummings
19. Donald Ervin Knuth, “The Art of Computer Programming: Fundamental algorithms”, Addison-Wesley, 1997
20. Peter Kjel lberg and Torben U. Zahle , “Cascade Hashing”, Computer Science Department, University of Copenhage
21. Haim J Wolfson , “Geometric Hashing An Overview” ,Tel Aviv University,Isidore Rigoutsos , IBM T.F. Watson Research Center
22. Chan-Yong Park, Sung-Hee Park, Dae-Hee Ki1, Soo-Jun Park,Man-Kyu Sung, Hong-Ro Lee, Jung-Sub Shin, and Chi-Jung Hwang, “Fast Protein Structure Alignment Algorithm Based on Local Geometric Similarity”
23. Mihir Bellare and Ran Canetti and Hugo Krawczyk. , “Keying Hash Functions for Message Authentication”, Springer-Verlag
24. Serge Vaudenay, “A Classical Introduction to Cryptoghaphy”,Springer,Pg 63
25. “Mihir Bellar and Ran Canetti, Hugo Krawczyk, “Keying Hash Function for Message Authentication”, Springer
26. Hamza Özer, Bülent Sankur, Nasir Memon, “Robust audio hashing for Audio Identification”, EUSIPCO 2004 (XII European Signal Processing conference),Sept 6-10,2004,Vienna,Austria.

27. Sign,Longjiang Yu and Shenghe Sun, “Image Robust Hashing based on DCT’, IIH-MSP’06,International Conference on Intelligent Information Hiding and multimedia Signal Processing.
28. E. Papapetrou, E. Pitoura , and K. Lillis , “Speeding –up Cache Lookups in Wireless Ad-Hoc Routing Using Bloom Filters”, the 16th Annual International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC 2005), Sept 11-13, 2005, Berlin, Germany,2005.
29. Andrei Broder, Michael Mitzenmacher, “Network Applications of BloomFilters: A Survey”, Internet Math 1(2003), no.4, 485-509
30. M.V. Ramakrishna, ‘Practical Performance of Bloom Filters and Parallel” , 6th International Conference on Distributed and Parallel Computing.
31. M V Ramakrishna,Justin Zobel , “Performance in Practice of String Hash Function’, International Conference on Database Systems for Advanced Applications’97 .
32. Ketan Dalal, Luc Devroy, Ebrahim Malalla and Erin Mcleish, “Two way chaining with Reassignment”
33. Per-Ake, Per-Åke Larson, “Linear Hashing with Separators-A Dynamic Hashing Scheme Achieving One-Access Retrieval”, University of Waterloo, Computer Science Dept., 1984
34. Rasmus Pagh Flemming Friche Rodler , “Cuckoo Hashing”,9th Annual European Symposium on Algorithms ,ESA 2001,Denmark.
35. Rasmus Pagh “Cuckoo Hashing for Undergraduates”, IT University of Copenhagen,March 27, 2006
36. http://www.inf.ed.ac.uk/teaching/courses/cs/0708/lecs/hashfuncs-6up.pdf.

# ANNEXURE II

# Appendices

**Bloom filter:** It is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. False positives are possible, but false negatives are not. Elements can be added to the set, but not removed .The more elements that are added to the set, the larger the probability of false positives. Bloom filters are incredibly space-efficient when the number of potential elements in the set is large. Hash functions are an essential ingredient of the Bloom filter.

**Collision**: When two keys hash to the same index, 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.

**Cascade Hashing**: It is a hashing method which is very effective, efficient, and very easy to implement. But it uses extra memory. It avoids the effects of primary and secondary clustering. So it is very good option if large memory is not a problem.

**Cuckoo Hashing**: It possesses the same theoretical properties as the classic dictionary but is much simple the space usage is similar to that of binary search trees. A special feature of our lookup procedure is that there are just two memory accesses, which are independent and can be done in parallel if this is supported by the hardware. It is very simple to implement.

**Cryptographic Hashing**: It is basically used in computer and network security. Like for Checksums, Authenticating Documents.

**CT:** geometric hashing uses the curve matching used in CT scans. Computed tomography (CT) is a medical imaging method employing tomography. Digital geometry processing is used to generate a three-dimensional image of the inside of an

object from a large series of two-dimensional X-ray images taken around a single axis of rotation.

**DTC:** A discrete cosine transform (DCT) expresses a sequence of finitely many data points in terms of a sum of cosine functions oscillating at different frequencies. DCTs are important to numerous applications in science and engineering, from lossy compression of audio and images

**Hashing**: Enables access to table items in time that is relatively constant and independent of the items.

Hashing is a method to store data in an array so that storing, searching, inserting and deleting data is fast. For this every record needs an unique key. The basic idea is not to search for the correct position of a record with comparisons but to compute the position within the array.

**Hash Function**: Maps the search key of a table item into a location that will contain the item!

These well-defined procedure or mathematical function which converts a large, possibly variable-sized amount of data into a small datum, usually a single integer that may serve as an index into an array. The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes.

**Hash Table**: An array that contains the table items, as assigned by a hash function

a hash table or hash map is a data structure that uses a hash function to efficiently map certain identifiers or keys (e.g., amit) to associated values (e.g., their telephone numbers). The hash function is used to transform the key into the index (the hash) of an array element where the corresponding value is to be sought. Also hash table is a storage location in memory or on disk that records the hashed values created by the hashing algorithm

**Load Factor**: the ratio n/s between n and the size s of its bucket array where n is the number of stored entries in the hash table. The performance of most collision resolution methods depends on this load factor.

**MAC**: A MAC is an Authentication tag (also called a checksum) derived by application of an authentication scheme, together with a secret key, to a message. MACs are computed and verified with the same key so the intended receiver, unlike digital signatures, can only verify them. MACs can be derived from various cryptographic techniques.

**MRI:** Magnetic Resonance Imaging (MRI), or nuclear magnetic resonance imaging (NMRI), is primarily a medical imaging technique most commonly used in radiology to visualize the internal structure and function of the body. The hashing technique is used in MRI.

**Perfect Hashing**: It guarantees that you for no collisions at all. It is possible when you know exactly what set of keys you are going to be hashing when you design your hash function. It's popular for hashing keywords for compilers.
Peer to Peer network: In its simplest form, a peer-to-peer (P2P) network is created when two or more PCs are connected and share resources without going through a separate server computer.

**Robust Image Hashing**: Image robust hashing is related to cryptographic hash function. In contrast to cryptographic hash functions this robust digest is sensitive only to perceptual change Image robust hashing is used in content-based retrieval, monitoring, and filtering

**Robust Audio Hashing**: These hash functions are based on a concise description of the time-frequency characteristics and used in audio content identification.

**String Hashing**: It is used where fast access to distinct strings is required. String hashing is the process of reducing a pseudo- random number in specified range. It is fundamental operation, which is widely used in applications where speed is critical. One more use of it is in spell checking.

**Web Cache**: It is the caching of web documents (e.g., HTML pages, images) in order to reduce bandwidth usage, server load, and perceived lag. A web cache stores copies

of documents passing through it; subsequent requests may be satisfied from the cache if certain conditions are met.

# ANNEXURE III

## List of Publications

**[1]** Mahima Singh, Dr. Deepak Garg, *"Choosing Best Hashing Strategies and Hash Functions"*, International Advance computing conference (IACC'09), March 2009 at Thapar University, Patiala.