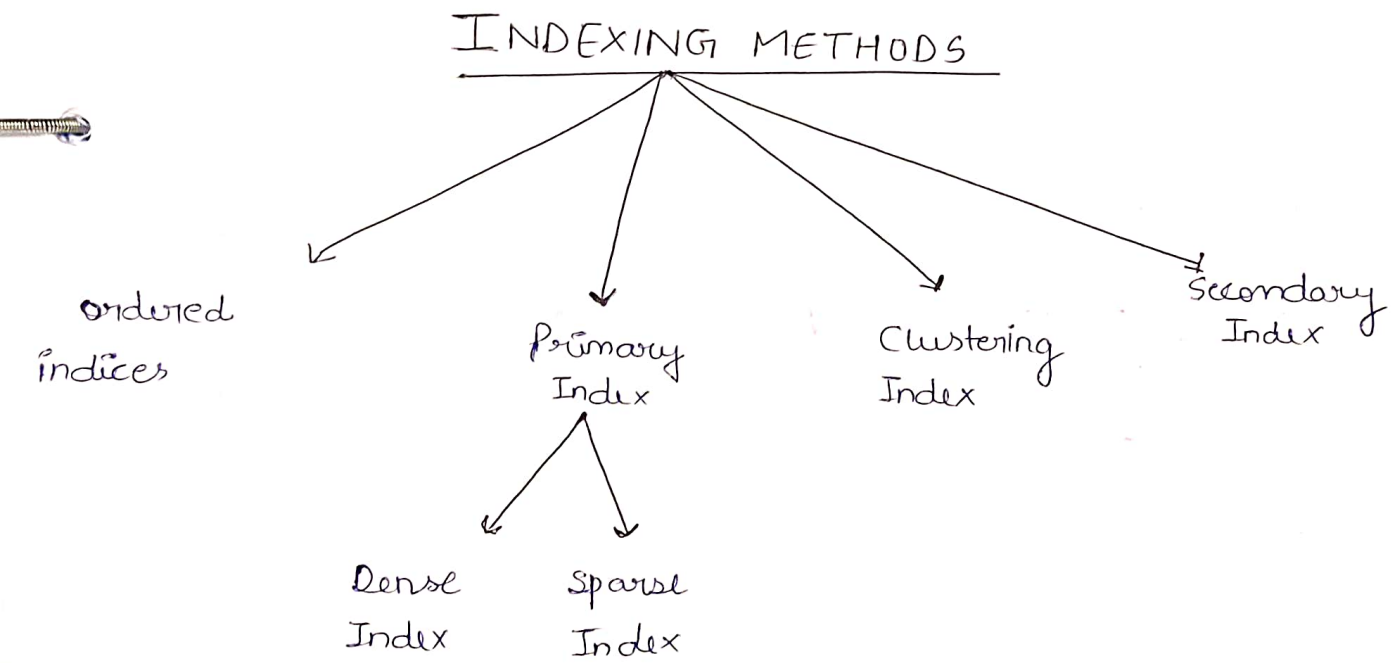
INDEXING

Indexing is used to optimize the performance of a database by minimizing the number of disk Accesses required when a query is processed.

Index is a type of data structure. It is used to locate & Access the data in a database table Quickly.

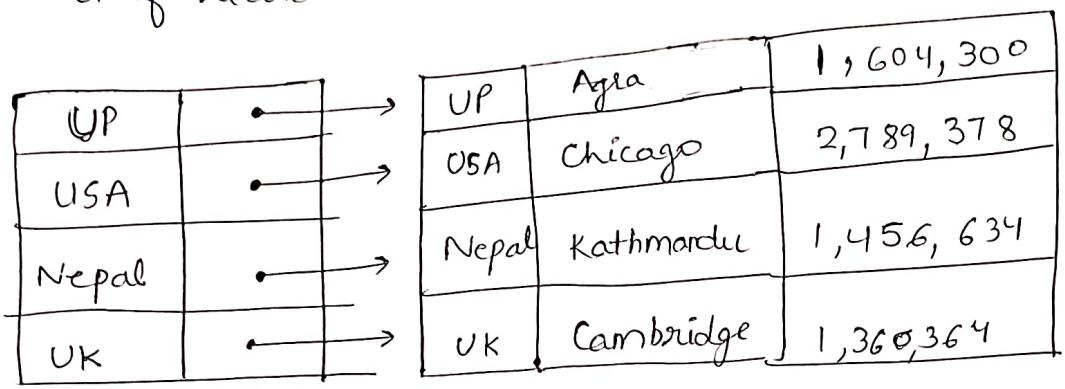


1) Ordered Indices :- The indices are usually sorted to make Searching faster. The indices which are sorted are known as ordered indices.

2) Primary Index :- If the index is created on the basis of the primary key of the table, then it is known as primary indexing.

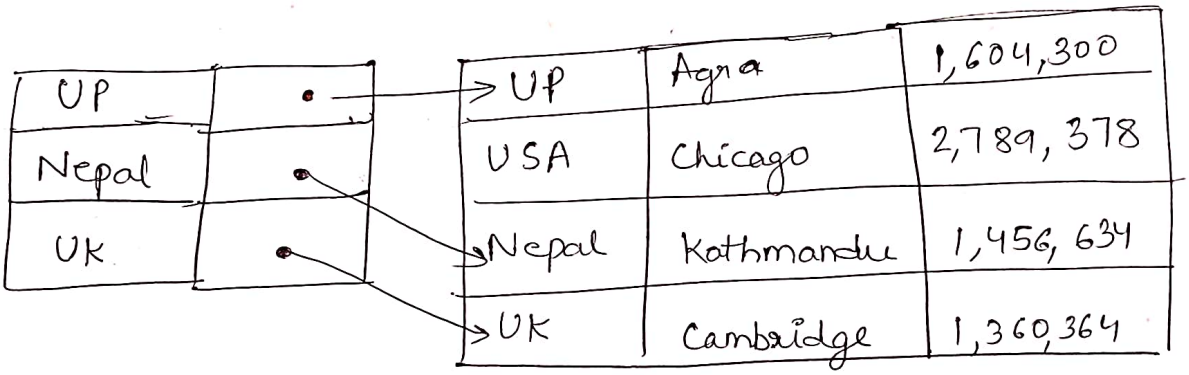
(a) Dense Index :- The dense index Contains an index record for every Search key Value in the data file.

→ In this, the Number of records in the index table is Same as the number of records in the main table.



(b) Sparse Index :- In the data File, index record appears only for a few items. Each item points to a block.

In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



What is a File?

A File is named a Collection of related information that is recorded on Secondary storage Such as magnetic disks, magnetic tapes etc.

What is File organization?

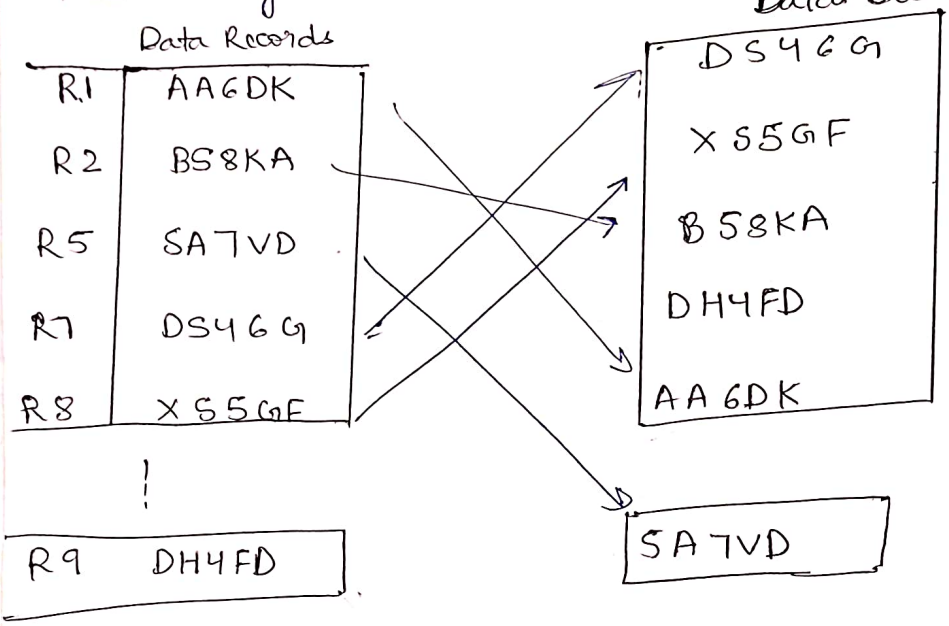
Storing the Files in a Certain order is called File Organization. File organization refers to the logical relationships among various records that constitute the File.

Types of File Organizations

1. Sequential File Organization.
2. Heap File organization
3. Hash File organization
4. B^+ tree File organization
5. ISAM [Indexed Sequential Access Method].

ISAM [Indexed Sequential Access Method]

In this method, records are stored in the File using the primary Key. An index value is generated for each primary Key and mapped with the record. This index contains the Address of the record in the File.



Advantages of ISAM:

→ In this Method, each record has the Address of its data block, Searching a record in a huge database is quick & easy.

Disadvantages of ISAM:

→ This method requires extra space in the disk to store the index value.

→ When the new records are inserted, then these Files have to be reconstructed to maintain the sequence.

B-Tree

B-tree is a specialized m-way tree that can be widely used for disk Access. A B-tree of order m can have at most (m-1) keys and m children. one of the main reason of using B tree is its capability to store large no. of keys in a single node & large key values by keeping the height of the tree Relatively small.

Operations

Insertion

Step1: If the tree is empty, a root node is allocated, & we will insert the key.

Step2: we will then update the allowed No. of keys in the node.

Step3: We will then search for the appropriate node for the insertion of the element.

Step4: If the node is filled, we will follow the below steps.

Step 4.1: Insert the data elements in Ascending order.

Step 4.2: once the data elements exceed their limit, we will split the node at the median.

Step 4.3: we will then push the median key upwards, making the left keys the left child nodes and the right keys the right child nodes.

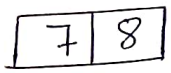
Step 5 : If the node is not full, we will follow the below steps.

Step 5.1 : We will insert the node in Ascending order.

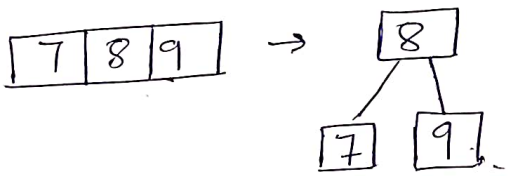
Example Construct a tree of order @ 3

7, 8, 9, 10, 11, 16, 21, 18.

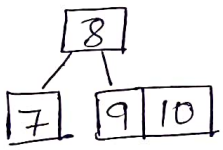
Inserting 7



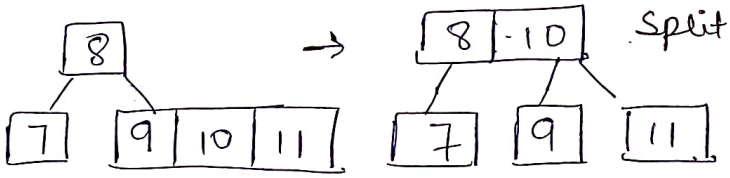
Inserting 8



Split & Inserting 9

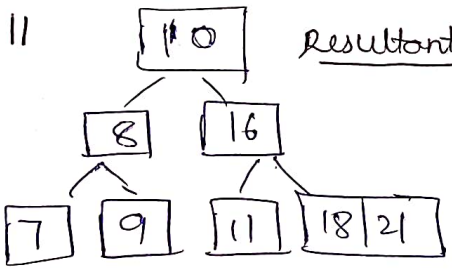


Inserting 10

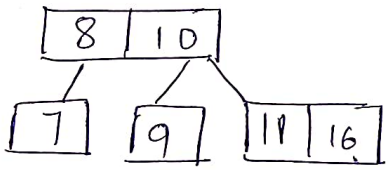


Split & Insert 11

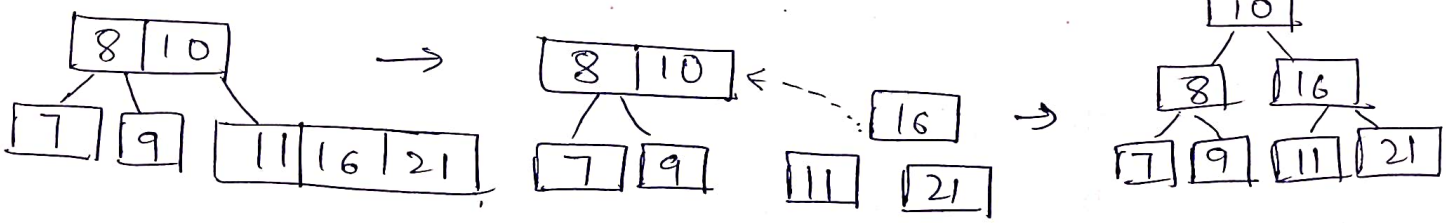
Inserting 18



Resultant Tree



Inserting 16



Split & Insert 21

Deletion from a B-tree

Deleting an element on a B-tree consists of three main events:

1) Searching the node
2) Deleting the key
3) Balancing the tree if required.

Underflow

This condition occurs when a node contains less than the minimum number of keys it should hold.

Important terms before learning Deletion

1) Inorder Predecessor

The largest key on the left child of a node is called its inorder predecessor.

2) Inorder Successor

The smallest key on the right child of a node is called its inorder successor.

Important Facts About a B tree of degree m.

1) A node can have a maximum of m children (i.e 3)
2) A node can contain a maximum of (m-1) keys (i.e 2)
3) A node should have a minimum of $\lceil m/2 \rceil$ children. (i.e. 2)
4) A node (except root node) should contain a minimum of $\lceil m/2 - 1 \rceil$ keys. (i.e 1)

CASE I

Ste The Key to be deleted lies in the leaf. There are two cases for it.

Ste 1. The deletion of the key does not violate the property of the minimum number of keys a node should hold.

Ex 2. The deletion of the key violates the property of the minimum number of keys a node should hold.

In this case, we borrow a key from its immediate neighboring sibling node in the order of left to right.

Else, Check to borrow from the immediate right sibling node.

If both the immediate sibling nodes already have a minimum No. of keys, then merge the node with either left sibling node or right sibling node. This is done through parent node.

CASE II

If the key to be deleted lies in the internal node, the following cases occur.

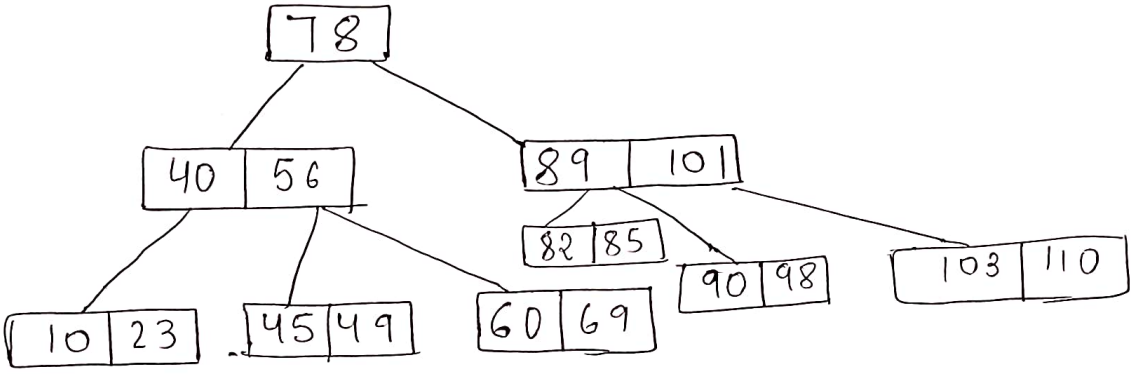
1. The internal node, which is deleted, is replaced by an inorder predecessor in the left child has more than minimum No. of keys.

2. The internal node, which is deleted, is replaced by an inorder successor if the right child has more than minimum No. of keys.

3. If either child has exactly a minimum No. of keys then, merge the left and the right children.

After merging if the parent node has less than the minimum No. of keys then, look for the siblings as in Case I.

Searching In B- Tree



This Searching is similar to that in Binary Search tree.

For example, if we search an item 49 in the following B Tree.

The process will be:

1) compare item 49 with Root node 78. Since 49<78 hence, move to its left sub-tree.

2) Since, 40<49<56, traverse right sub-tree of 40.

3) 49>45, move to right. Compare 49.

4) match found, return.

CASE III

In this case, the height of the tree shrinks. If that key lies in an internal node, & the deletion of the key leads to a fewer no. of keys in the node (i.e less than the minimum required), then look for the inorder predecessor & the inorder successor. If both the children contain a minimum No. of keys then, borrowing cannot take place. This leads to Case II i.e merging the children.

Again, look for the sibling to borrow a key. But, if the sibling also has only a minimum no. of keys then, merge the node with the sibling along with the parent. Arrange the children Accordingly (increasing order).

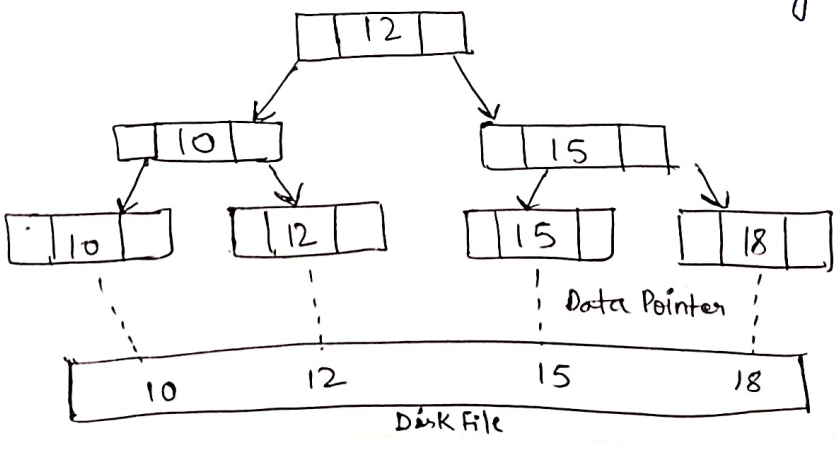
B++ Tree

B + trees are extensions of B-Tree designed to make the insertion, deletion & searching operations more efficient.

B+ trees store records in leaf nodes and keys in internal nodes.

B+ tree is that all leaf nodes are connected to eachother in a single linked list format and a data pointer is available to point to the data present in disk file.

The size of main memory is limited, B+ trees act as the data storage for the records that Couldn't be stored in the main memory. For this, the internal nodes are stored in the main memory and the leaf nodes are stored in the secondary memory storage.



Insertion Operation

The insertion to a B+ tree starts at a leaf node.

Step1- Calculate the max number and min number of keys to be added onto the B+ tree node.

order = 4	order (m) = 4
(m) Max children = 4	(m) max children = 4
$(\frac{m}{2})$ Min children = 2	min children = $\frac{m}{2} = 2$
$(m-1)$ Max keys = 3	max keys = $(m-1) = 3$
$(\frac{m-1}{2})$ Min keys = 1	min keys = $(\frac{m}{2})-1 = 1$

Step2 - Insert the elements one by one accordingly into a leaf node until it exceeds the maximum key number.

Step3 - The node is split into half where the left child consists of minimum number of keys & the remaining keys are stored in the right child.

Step4 - But if the internal node also exceeds the max keys property, the node is split in half where the left child consists of the minimum keys and remaining keys are stored in the right child However, the smallest no. in the right child is made the parent

Step5 - If both the leaf node and internal node have the max keys, both of them are split in the similar manner and the smallest key in the right child is added to the parent node.

Deletion Operation

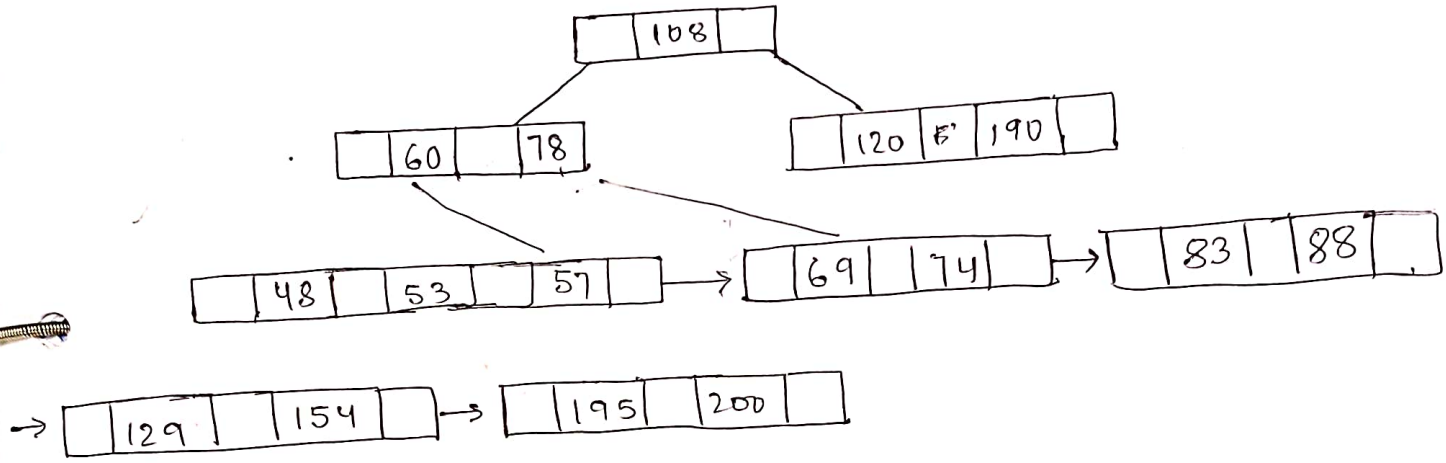
Step1: Delete the key and data from the leaves.

Step2: If the leaf node contains less than minimum number of elements, merge down the node with its sibling and delete the key in between them.

Step 3: If the index node contains less than minimum number of elements, merge the node with the sibling and move down the key in between them.

Example

Delete the Key 200 from the B+ tree shown in the diagram.



Searching Operation

Following steps are followed to search for data in a B+ tree of order M, Let the data to be searched be K.

1. Start from the root node. Compare K with the keys at the root node [K1, K2, K3, ... K] (m-1).
2. If K < K1, go to the left child of the root node.
3. Else if K == K1, Compare K2. If K < K2, K lies between K1 and K2. So, search in the left child of K2.
4. if K > K2, go for K3, K4, ... K m-1 as in steps 2 and 3.
5. Repeat the above steps until a leaf node is reached.
6. If K exists in the leaf node, return true else return false.

Hashing

The hashing technique utilizes an auxillary hash table to store the data records using a hash function.

There are 2 key components in hashing:

Hash Table: A hash table is an array or data structure and its size is determined by the total volume of data records present in the database. Each memory location in a hash table is called a 'bucket' or hash indice and stores a data record's exact location and can be accessed through a hash function.

Bucket: A bucket is a memory location (index) in the hash table that stores the data record. These buckets generally store a disk block, which further stores multiple records. It is also known as the hash index.

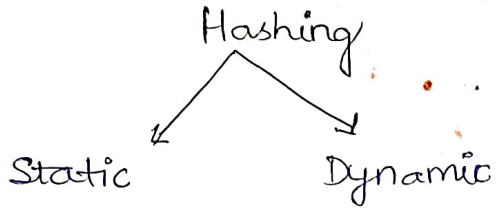
Hash Function: A hash function is a mathematical equation or algorithm that takes one data record's primary key as input and computes the hash index as output.

Hashing In DBMS

In huge databases it is very impractical to search all the index values to get the desired data as searching all the index values ~~to get the desired data as searching all the inde~~ through all the levels is not an efficient way to reach destination data block to retrieve the desired data.

Therefore, hashing in DBMS is an effective technique used to directly search the location of data without using index structure. It uses a hash function (a mathematical fun^{n}) to find the exact location of a record in the minimum amnt of time.

Types of Hashing in DBMS



Static Hashing

In static hashing, for a key K, has function h(x) always generates the same memory address. For example, if $h(x) = x \% 7$ then for any x the value will always be the same.

Operations in static Hashing

Insertion - To insert a new record in the table we will calculate the bucket address for search key K, and the record will be stored there.

Search - To search a record we will use the same hash func^n to get the bucket address of the record and retrieve the data from there.

Deletion - To delete a record from the table we will Firstly search for that record and then simply delete it from that location.

Static hashing → open hashing
Static hashing → closed hashing

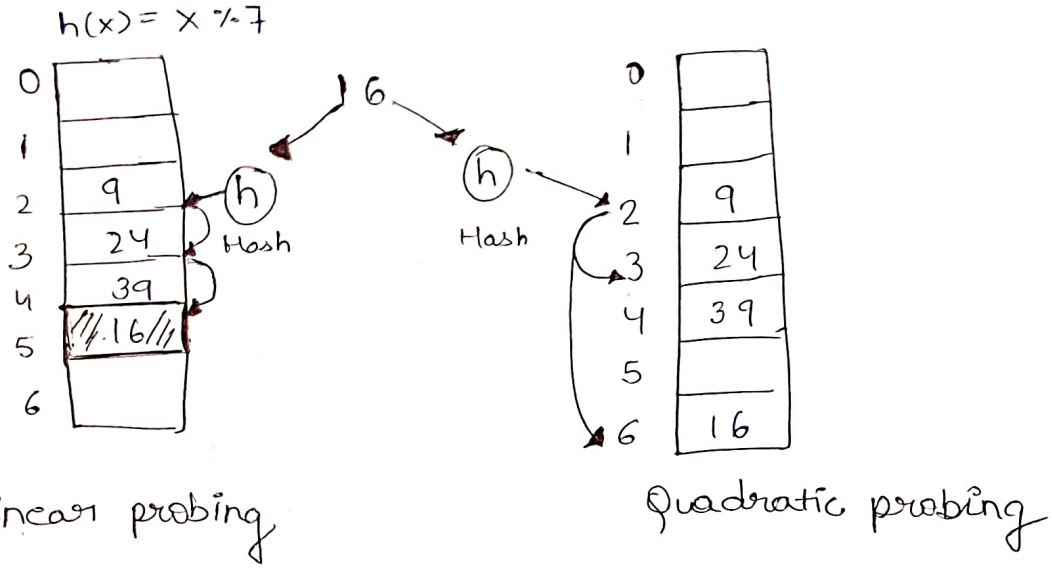
Open hashing

In this, whenever a collision occurs, we probe for next empty bucket to enter new record.

We can check ordering by two methods :-

* Linear probing
* Quadratic probing

Example -

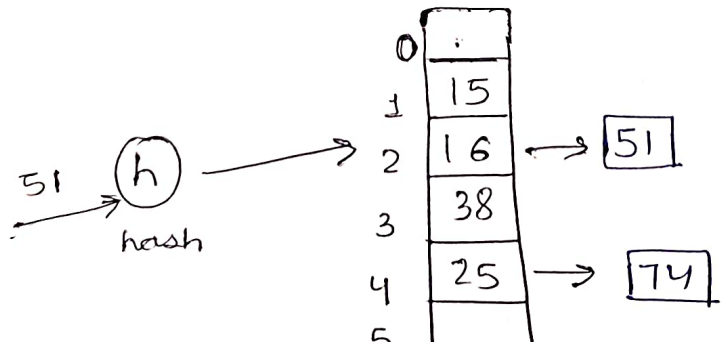


Linear probing

Quadratic probing

Closed hashing

In this hashing, the collision condition is handled by linking the new record after the previous one, due to which it is also termed as "Hashing with Separate Chaining".



Dynamic Hashing

The Main disadvantage of static hashing is that it does not expand or shrink as the size of the database expands or shrinks.

To counter this problem, we use dynamic hashing which allows on demand addition & removal of data buckets. It is also known as extended hashing.

This method allows efficient insertion and deletion operations.

Searching A Key —

1) First, Calculate the hash address of the key.
2) Check how many bits are used in the directory, and these bits are called as i.
3) Take the least significant i bits of the hash address. This gives an index of the directory.
4) Now using the index, go to the directory and find bucket address where the record might be.

Inserting A new record

1) Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
2) If there is still space in that bucket, then place the record in it.
3) If the bucket is full, then we will split the bucket and redistribut the records.

Advantages of dynamic hashing

1. In this, performance does not decrease as the data grows in the system. it simply increases the size of memory to accommodate the data.

2. Memory is well utilized as it grows and shrinks with the data. There will not be any unused memory lying.

Disadvantages of dynamic hashing

1. In this method, if the data size increases then the bucket size is also increased. These addresses of data will be maintained in the bucket address table. This is because the data address will keep changing as bucket grow and shrink. if there is a huge increase in data, maintaining the bucket address table becomes tedious.

2. In this case, the bucket overflow situation will also occur. But it might take little time to reach this situation than Static hashing.