Data Structures & Algorithms

CHAPTER - I : INTRODUCTION

- **Definition :**

An data structure is a specialized format for organizing, processing, retrieving and storing data.

- **Basic Terminology :**
  - Data : Data are values or set of values.
  - Data Item : Data item refers to single unit of values.
  - Group Items : Data items that are divided into sub items are called as Group items.
  - Elementary Items : Data items that cannot be divided are called as Elementary Items.
  - Entity : An entity is that which contains certain attributes or properties, which may be assigned values.
  - Field : It is a single elementary unit of information representing an attribute of an entity.
  - Record : Record is a collection of field values of a given entity.
  - File : File is a collection of records of the entities in a given entity set.

- **Algorithm :**

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

- **Categories of algorithms :**
  - Search
  - Sort
  - Insert
  - Update
  - Delete

$c_1 g(n) \leq T(n) \leq c_2 g(n)$, for all n, $n \geq n_0$

Where, $T(n)$ - time execution of an algorithm for different problem sizes n.

$g(n)$ - mathematical representation of an algorithm as a function of problem size n.

This should be read as - 'T of n' is 'Theta' of 'g of n'.

The Theta Notation is more precise than Big oh and Omega notations.

The function $T(n) = \Theta(g(n))$ if $g(n)$ is both an upper and lower bound on $T(n)$.

Time complexity Examples :

1. $O(1)$ - constant computing time
2. $O(n)$ - Linear
3. $O(n^2)$ - Quadratic
4. $O(n^3)$ - Cubic
5. $O(2^n)$ - Exponential

Below table give quick view of various function grow with n.

log n	n	n log n	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296

Function Values.

One dimensional Representation :

Row	Column	A
1	3	6
1	5	9
2	1	2
2	4	7
2	5	8
2	7	4
3	1	10
4	3	12
6	4	3
6	7	5

1D Representation of Sparse Mat.

Linked Representation of Sparse Matrix :

Node structure = | Row Number | Column Number | Value | Pointer To Next Node |

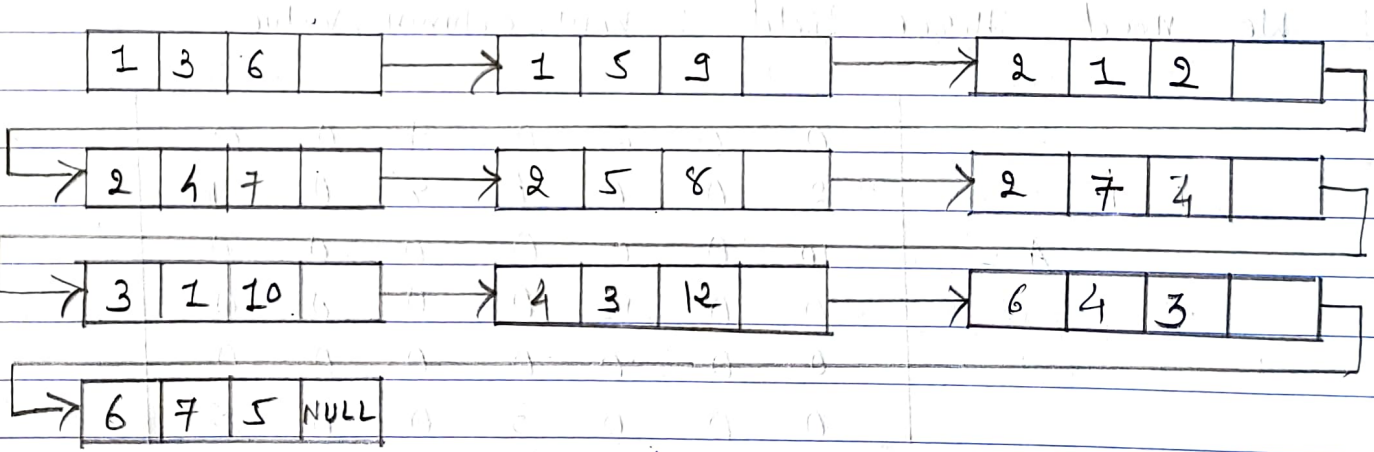
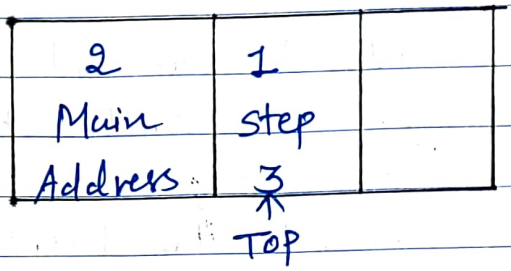


Fig : Linked Representation of Sparse Matrix

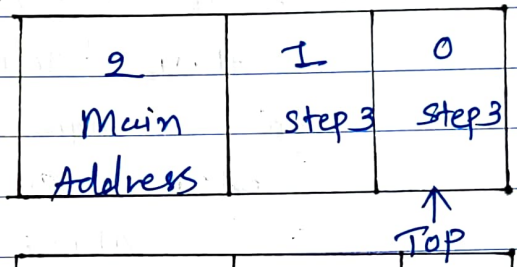
Level 2 (1st Recursive call)

1. PUSH (A, 1, (1, Step 3)
2. N ≠ 0
   PARM ← 0, ADDR ← step 3

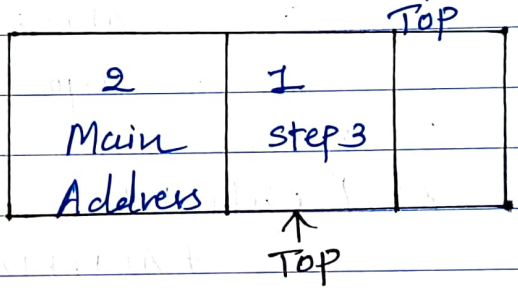


Level 3 (2nd Recursive call)

1. PUSH (A, 2, (0, step 3)
2. N = 0
   FACTORIAL ← 1

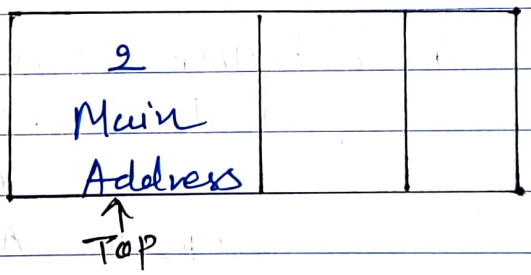


4. POP (A, 3)
   go to step 3



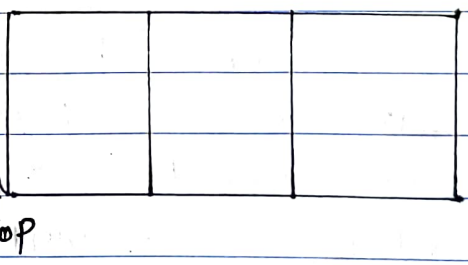
Return to level 2

3. FACTORIAL ← 1 * 1
4. POP (A, 2)
   go to step 3.



Return to level 1

3. FACTORIAL ← 2 * 1
4. POP (A, 1)
   go to main address



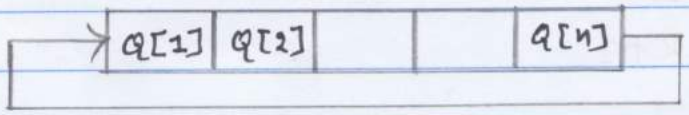
Tower of Hanoi : It is a recursive problem which has a historical basis in the ritual of the ancient Tower of Brahma.

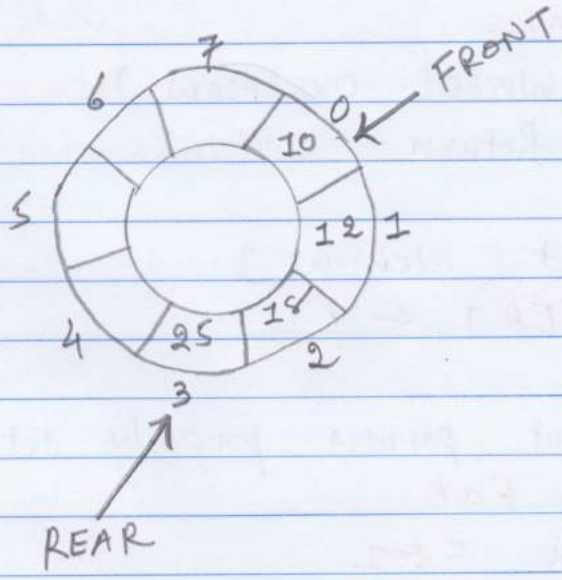
problem – Given N discs of decreasing size stacked on one needle and two empty

Circular Queue: A more suitable method of representing simple queue which prevents an excessive use of memory is to arrange the elements Q[1], Q[2], ... Q[n] in a circular fashion with Q[1] following Q[n], this is called Circular Queue.

In a circular Queue, the last node is connected back to the first node to make a circle.

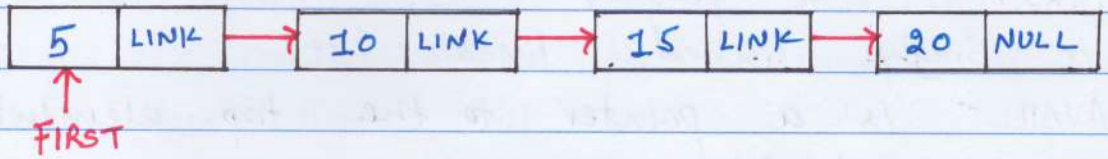
Circular Queue follows FIFO principle.
It is also known as RING BUFFER.



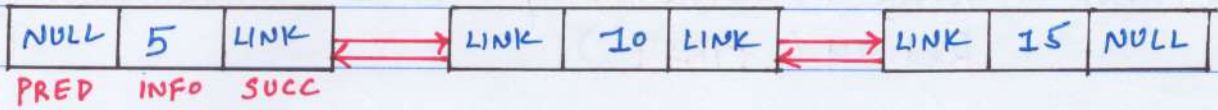


— Variants of linkedlist :—

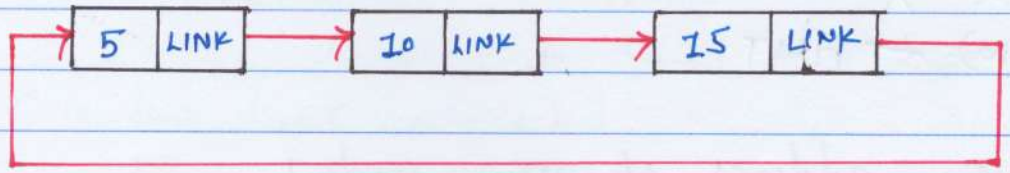
1. Singly Linked list : All the nodes are linked in sequential manner, such ordered list is called singly linked list.



2. Doubly - Linked List : All the nodes are linked together by multiple address link which help in traversing the successor node (NEXT NODE) and predecessor (PREVIOUS NODE) within the list.



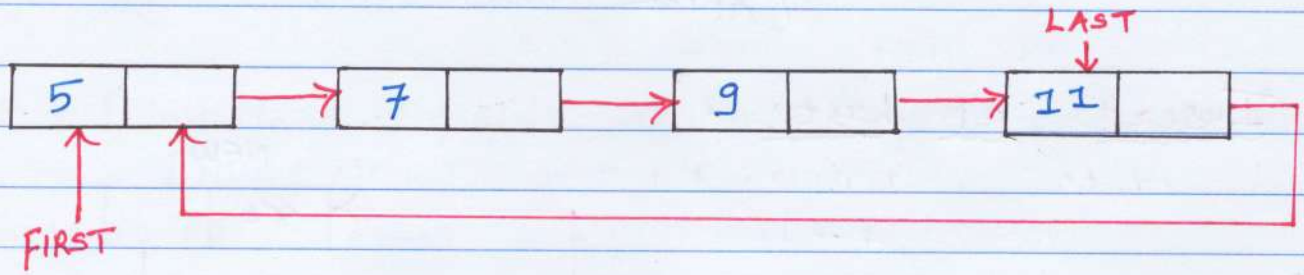
3. Circular linked List : This ordered list has no beginning and no end. Here, the last node consist the address of first node in singly linked list will create circular linked list.



7. [set link of last node and return]
PTR(NEW) ← NULL
Return (BEGIN).

★ ★ CIRCULAR LINKED LIST ★ ★

If we replace NULL pointer of the last node of singly linked list with the address of it's first node, it becomes circular linked linear list.



Here, FIRST - the address of first node.
LAST - the address of last node.

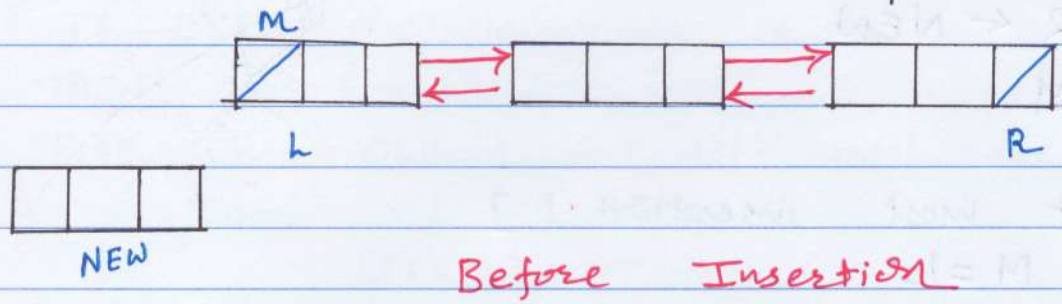
Advantages :-

- Every node is accessible from given node.
- It saves time when we have to go to the first node from the last node.

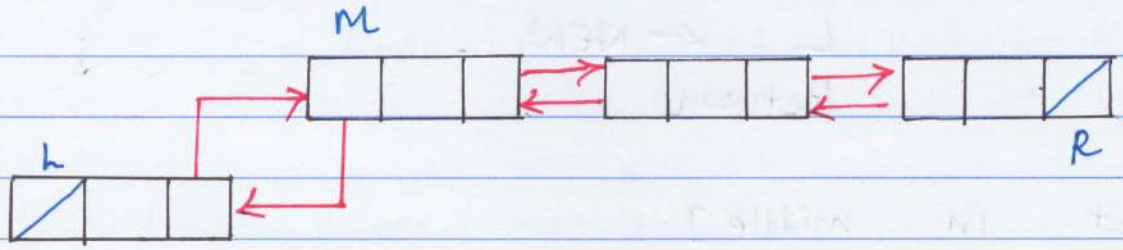
Dis-advantages :-

- Reversal is not easy in the linked list.
- Execution may go into infinite loop.
- Backward traversal is very costly.

~ Left most Insertion in Doubly Linked List :-



Before Insertion



After Insertion

Algorithm : DOU_INS(L,R,M,x)

- It inserts a new node in doubly linked list.
- Insertion is to be performed to the left of a specific node with its address given by the pointer variable M.

1. [create a New Empty Node]
   NEW ← NODE

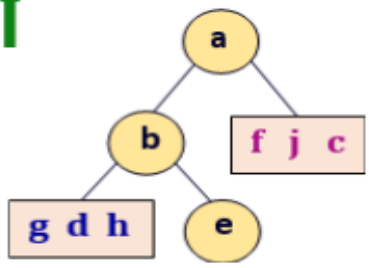
2. [copy information field]
   INFO(NEW) ← X

3. [Insert into an empty list]
   IF R = NULL
   THEN LPTR(NEW) ← NULL
   RPTR(NEW) ← NULL

Step 3

Next element under scan is d. Use currently scanned **element d** in left sub-tree of b to get further its left sub-tree and right sub-tree.

Preorder : a b d g h e c f j

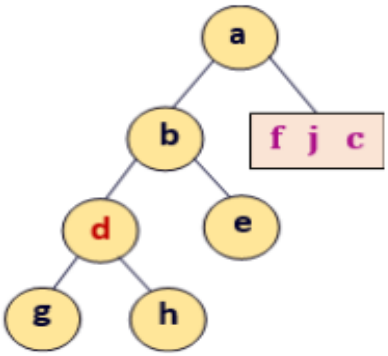


d is root node in the left subtree of b.

Left Subtree of b : g d h

d is the root

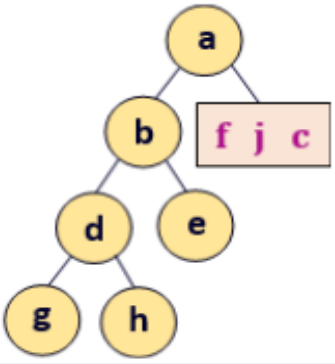
g is in the left subtree of d **h is in the right subtree of d**



Step 4

Next scanned element is g, and **g has no left or right child.** So stop and go for scanning next element.

Preorder : a b d g h e c f j



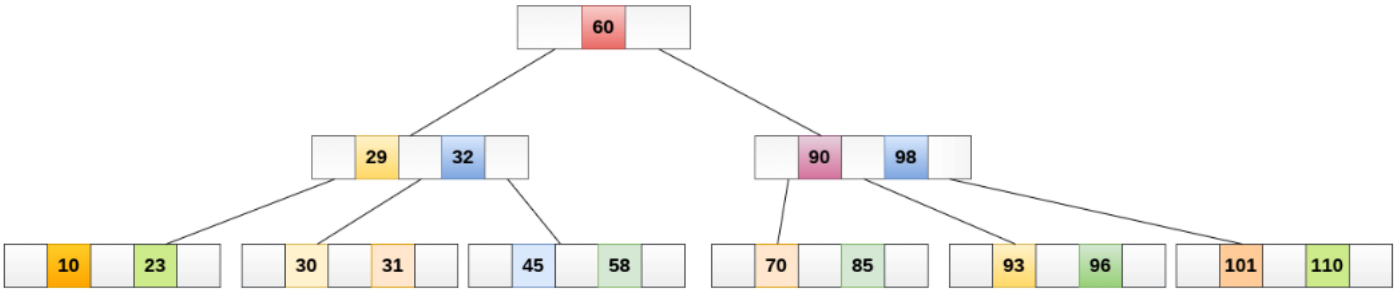
g is a single value node. Node g has no left or right child. So move to next step.

♣ 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 number of keys in a single node and large key values by keeping the height of the tree relatively small.
- ❖ A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.
  1. Every node in a B-Tree contains at most m children.
  2. Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
  3. The root nodes must have at least 2 nodes.
  4. All leaf nodes must be at the same level.

It is not necessary that, all the nodes contain the same number of children but, each node must have m/2 number of nodes.

- ❖ A B tree of order 4 is shown in the following image.



♣ B+ TREE:

- ❖ In B Tree, Keys and records both can be stored in the internal as well as leaf nodes. Whereas, in B+ tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values.
- ❖ The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient.
- ❖ B+ Tree are used to store the large amount of data which cannot be stored in the main memory. Due to the fact that, size of main memory is always limited, the internal nodes (keys to access records) of the B+ tree are stored in the main memory whereas, leaf nodes are stored in the secondary memory.
- ❖ The internal nodes of B+ tree are often called index nodes. A B+ tree of order 3 is shown in the following figure.

♣ Linear Search

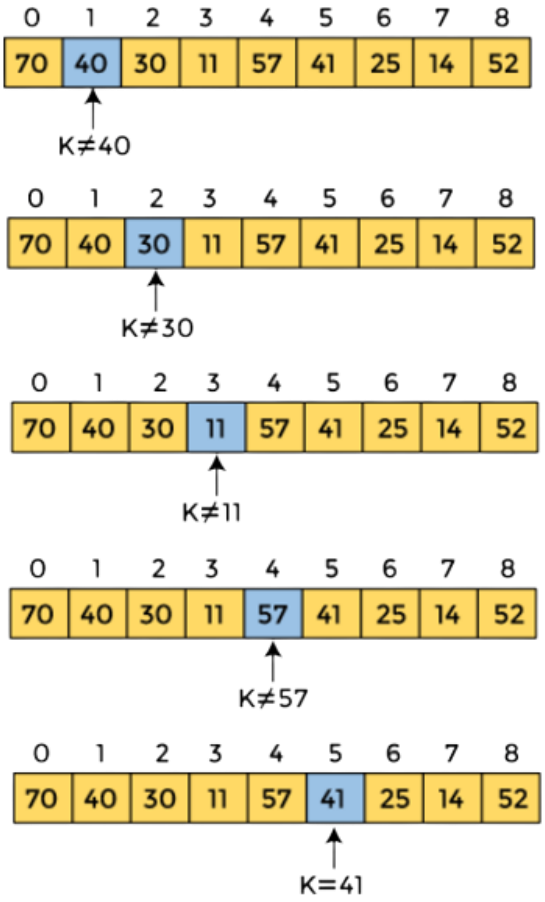
❖ Concept:

linear search or sequential search is a method for finding a particular value in a list that consists of checking every one of its elements, one at a time and in sequence, until the desired one is found. The efficiency of the sequential search is O(n).

❖ Algorithm:

```
Linear_Search
    for i=0 to last index of A:
        if A[i] equals key:
        return i
return -1
```

❖ Example:



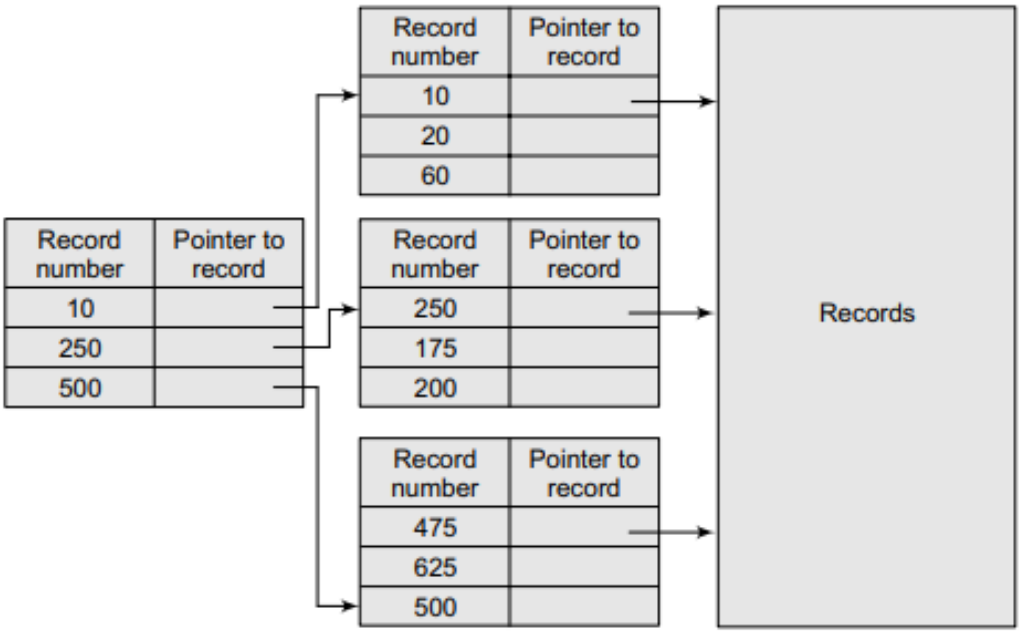


Fig: multi-level indices

❖ Hashed indices (Direct file organization):

- ❖ It is a common technique used for fast accessing of records on secondary storage.
- ❖ Records of a file are divided among buckets.
- ❖ A bucket is either one disk block or cluster of contiguous blocks.
- ❖ A hashing function maps a key into a bucket number. The buckets are numbered 0, 1,2...b-1.
- ❖ A hash function f maps each key value into one of the integers 0 through b - 1.
- ❖ If x is a key, f(x) is the number of bucket that contains the record with key x.
- ❖ The blocks making up each bucket could either be contiguous blocks or they can be chained together in a linked list.
- ❖ Translation of bucket number to disk block address is done with the help of **bucket directory**. It **gives** the **address of the first block** of the chained blocks in a linked list.
- ❖ Hashing is quite efficient in retrieving a record on hashed key. The average number of block accesses for retrieving a record.

$$= \mathbf{1}\ (\textbf{bucket directory}) + \frac{No\ of\ records}{No\ of\ buckets\ x\ No\ of\ records\ per\ block}$$

- ❖ Thus the **operation is b times faster** (b = number of buckets) than unordered file.
- ❖ To **insert a record with key value x**, the **new record** can added to the **last block** in the chain for bucket f(x). If the record does not fit into the existing block, record is stored in a new block and this new block is added at the end of the chain for bucket f(x).
- ❖ A well designed hashed structure requires two block accesses for most operations.