Introduction:-

* **Dictionaries:-**

A Dictionaries is a collection of Elements; Each Element has a field called key and no two elements have the same key

* operations performed on a dictionary are
  a> Insert an Element with a specified key value
  b> Search the dictionary for an element with specified key value
  c> Delete an Element with a specified key value

* Abstract data type of dictionary {

instances
collection of Elements with distinct keys

operations
Create(): Create an empty dictionary
Search(k,x): return Element with key k in x; return false if the operation fails, true if it succeeds
Insert(x): Insert x into the dictionary
Delete(k,x): delete element with key k and return it in x

}

* **Accessing of Dictionary Elements:-**

1> Random Access:- Any Element in the dictionary can be retrieved by performing a search on its key

2> Sequential Access:- Elements are retrieved one by one in ascending order of the key field

* **Example for Dictionary:-**

Collection of students records in a class

→ (key, value) = (Student number, Exam marks)

* **Representation of Dictionary:-**

There are mainly three methods of representing Dictionary.

1) Linear List Representation
2) Skip List Representation
3) Hashing

1) Linear List Representation

A dictionary can be represented using linear list representation where the dictionary elements and their keys increases from left to right. The linear list representation again divided into two classes

(i) Sorted List
(ii) Sorted chain

(i) **Sorted List :-**

e_1	e_2	e_3	- - - - -	e_n

* It uses a formula based representation of a linear list

* $L = (e_1, e_2, e_3, \ldots, e_n)$

Each e_i is a pair (key, value)

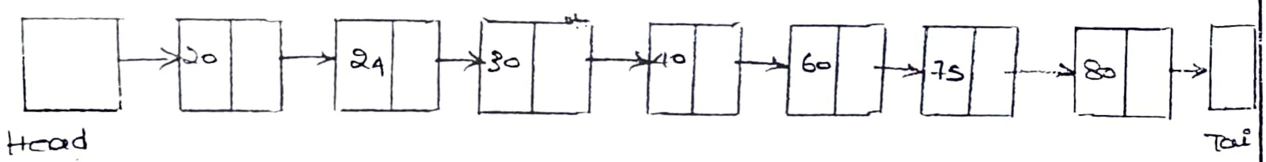
* To Search an 'n' Element in the dictionary it takes o(logn) time

* To make insertion, first verify that the dictionary doesn't already contain an Element with same key. This verification is done by performing Search operation with o(logn) time and next the insertion of Element is done by additional o(n) time as o(n) Elements must be moved to make room for the new Element.

* For deletion operation Searching is done for the Elem-ent which is to be deleted and then deletion is perfo-rmed.

(ii) Sorted chain:-

In this method Each node in the dictionary have two private numbers data and link. The Second Sorted array is represented as below.



Sorted array is provided by head and tail nodes as shown in the above figure. Head points to the Starting key value of the Element and tail node keep the Element with value larger than that of any other Element.

Eg:- Suppose we need to Search for the Students details whose Reg No is from 9000 to 9100 Then Head points to 9000 and tail points to 9100

In this method to search 'n' element in dictionary chain requires upto 'n' element comparision. This is time consuming method. To overcome this skip list representation of dictionary is implemented.

Skip List Representation

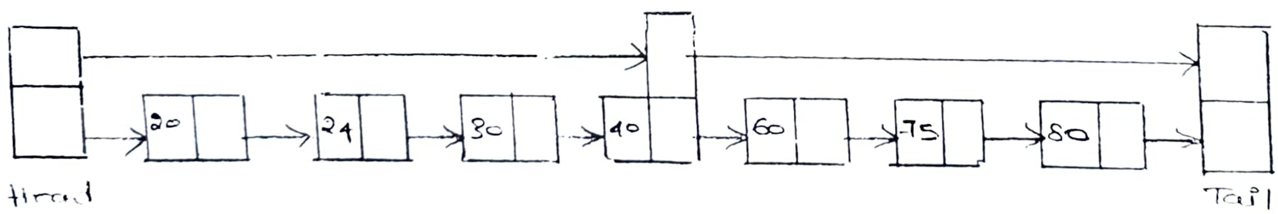
why skip lists:- The linear list with forward pointers is called skip list.

Although sorted chain takes o(n) time for search of 'n' element, we can improve the search performance of sorted chain by placing additional pointers in some of the chain nodes. These additional pointers permit us to skip over several nodes of the chain during a search operation. Thus it is no longer necessary to examine all chain nodes from left to right during a search.

The different operations performed on skip list ~~representation~~ are explained by considering below examples.

fig (a)

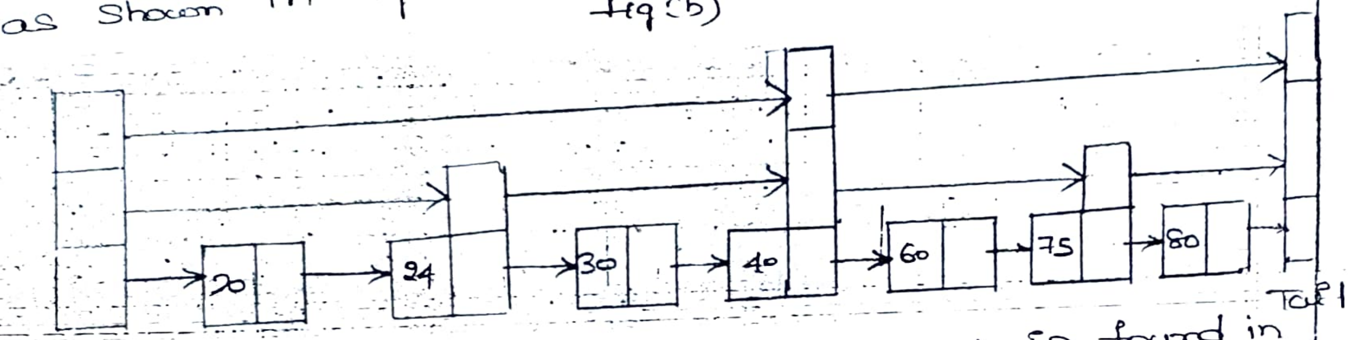
a)



* In this method the number of comparision is reduced to $\frac{n}{2}+1$ by keeping the pointer to the middle element.
* If we are looking for the smaller element than the element that pointer pointing, we need not want to search the second half.
* If we are looking for larger element we need compa-rision of elements only at the right half of the chain.

=> we can reduce the number of comparision by keeping pointers to the middle of Each half. It is as shown in fig (b)

fig (b)

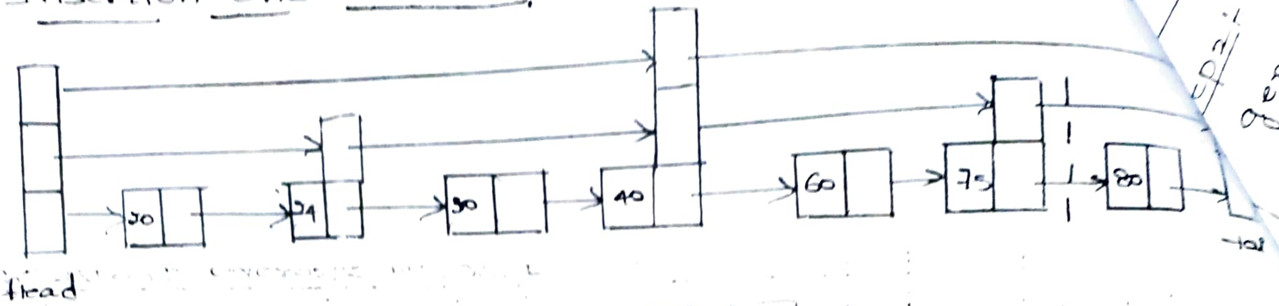


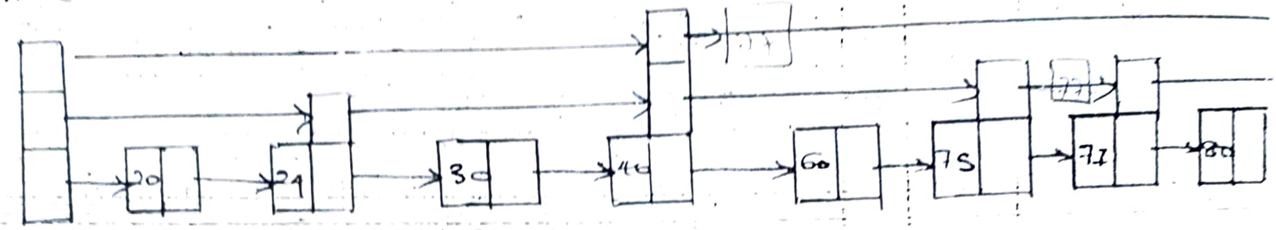
* For example to search 30, the Element is found in
$\Theta(1)$ time using Level - 2 chain. Since 30<40 the
Search continues by examing the middle Element
to left half. This is also done in $\Theta(1)$ time using
Level-1 chain. Since 30>24 the Search continues
by dropping into the Level 0 chain & compares with
next Element in this chain.

* To Search 77, the first comparision is with 40,
Since 77>40, drop to Level 1 chain and compare with
Element 75. 77> 75, drop to Level 0 chain &
compare with the next Element i.e, 80 in this
chain that comes just after 75. Therefore 77 is
not in a dictionary.

* For general n, the Level 0 chain includes all elements
The level 1 chain includes every Second Element; the
Level 2 chain Includes every fourth Element and the
Level i chain includes Every 2^i th Element.
The Level i chain comprises a Subset of Elements in
the Level i-1 chain.

Insertion and Deletion :- fig(a)





fig(b)

To insert an Element with value 77 in the fig(a)

* First Search operation is performed to make Sure no Element with this value is present.
* During this Search the level 2 pointer is associated with 75 and 75 is also 40, the level pointer is associated with 75 and 75 is also an Element with level 0. Since 77 is inserted after 75, level 0 and level 1 pointers get affected as shown in the dotted line
* To insert an Element we have to assign a level for the newly inserted element

Assigning of level :-

* In a regular skip List Structure, a fraction p of the Elements on the level i-1 chain are also on the level i chain. Therefore the probability that an Element is on level i-1 chain is also on the level i chain is 'p'

* Assigning level using Random number generation method

Step 1 :- we have to decide the maximum level in the dictionary by choosing the probability

Maximum level, $i = \lceil \log_{1/p} N - 1 \rceil$ where N = number of Elements in the dict

Step 2: Choose a uniform random number generator which generates random number in the range 0 through Randmax probability of next random number is ≤ cut off

where, cutoff = P * Rand_max.

Step 3: Generate a random number, If random number generated ≤ cut off, the new element should assign i-1 level

Step 4: Now we have to check whether the Element is also in i^{th} level For this generate another random number If the newly generated random number is ≤ cut off then it is also have level i.

Step 5: Continue the process untill the random number ≥ cut-off. If the condition is satisfied the Element assign only i-1 level

In the above example we have to insert 77 at level-1 Element so we have to satisfy the condition

Random number generated ≥ cut-off.

* After the insertion of 77 at level 1' Element the linked list structure is as shown in fig(b)

Deletion:- To delete 77 from the list, we have to do Search operation first Since the 77 lies at Right half there is no need of control of left

* The Level 1 and level 0 pointers are associated with 77 By chains when these pointers are changed to point to the Element after 77 In their respective chains, Structure a fig(c) is obtained. Thus deletion is performed

Disadvantages of skip List:

1) Skip List is ~~only~~ associated only with sorted array. for unsorted array it cannot be implemented.

2) O(1) performance is not possible.

To overcome the above disadvantages the new method of dictionary representation is implemented. The new one is called 'Hashing'

Hashing :-

Hashing is one of the method of representing dictionary

It is a technique used for performing insertion, deletion & Searching in constant average time. [O(1)]

Hash function and Hash table :-

- Hashing uses a 'Hash function' to map keys into positions in a table called the Hash table.

* In ideal situation if Element 'e' has the key 'k' and 'f' is the hash function then 'e' is stored in position f(k) of the table

* To Search for an Element with key 'k', compute f(k) and see if there is an Element at position f(k). If so, the Element is found. If not, the dictionary contains no Element with this key.

* To delete an element from dictionary, make position f(k) of the table Empty.

Example :- Consider a Students records dictionary. Instead of using Student name as a key, use their ID number. Let us assume 100 students in a class and their ID numbers will be in the range of 951000 and 952000.

The function f(k) is defined by $f(k) = k - 951000$ to map the Students ID position 0 through 1000 in a hash table of size 1000

				- - - -		- - - -			
0	1	2	3		500		998	999	1000

* To Search the Students record with ID number 951003, 951500, 951998

$f(k) = k - 951000$

$f(951002) = 951002 - 951000$

$= 2$ → The student record with ID number 951002 found position '2' of hash table

$f(951998) = 951998 - 951000$

$= 998$ → The student record with ID number 951998 found at position '998' of hashtable

$f(951500) = 951500 - 951000$

$= 500$ → The student record with ID number 951500 found at position 500 of hash table

* There are several methods for describing hash function. The most common method is Division method

Hashing by Division :- In this method hash function has the form

$$f(k) = k \% D$$

where 'k' is the key

'D' is the size (i.e, number of positions in hash table)

% is the modulo operator

Bucket :- Each position in the hash table is called 'Bucket'

* Below figure shows hash table 'ht' with 11 buckets numbered from 0 to 10. This table contains three Elements (80, 40, 65). Since there are 11 buckets the divisor used is 11. The three Elements are allocated as below

				80				40			65
ht	0	1	2	3	4	5	6	7	8	9	10

$80 \% 11 = 3$

$40 \% 11 = 7$

$65 \% 11 = 10$

fig(a)

Hash clash:-

Consider the hash table represented in fig (a). To insert 58 in the table the position is

$$f(58) = 58 \% 11 = 3$$

But the position '3' is already occupied by 80. This problem in hasing is described as 'Hash clash'

Hash clash ~~definition~~:- when two Elements have same hash value collision will occur in hash table. This is termed as hash clash.

* In general a bucket may contain space for more than one Element, so a collision may not create any difficulties. An overflow occur if their is not room in the home bucket for the new Element.

(Home bucket is the position occupied by the first Element)

There are mainly two methods to overcome hash clash

1) chaining or open hashing

2) Linear open addressing or closed hashing

1) Hashing with chaining:-

Hashing with chaining is one of the clash resolving technique in which all the Elements with same hash value is linked together with separate chain. The chain is represented by a pointer provided by Each bucket.

The fig(b) shows the chaining hashing. Here the divisor D is 11

Therefore 11, 33, 66 are placed in linked list pointing at position 0

36 and 69 are placed in linked list pointing at position 3

16 49 82 at 5 [∵ f(k) = k % D]

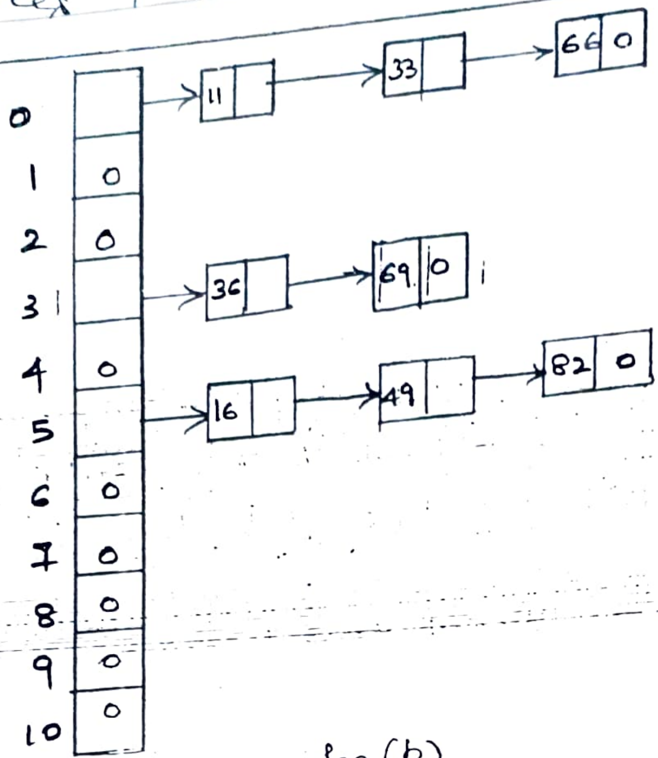
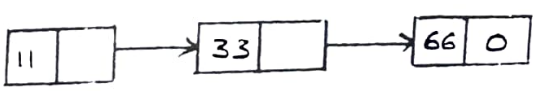


fig (b)

* To search for an element with key 'k' first compute the home bucket [k % D] for the key and then search the chain that begins at this bucket.

* To insert an element, first verify that the table doesn't already have element with same key. This search is limited to the chain for the home bucket of the new element. The elements are placed in ascending order of the key values in chain. [Linked list]

eg:-



* To delete an element with key k access the home bucket chain, search this chain for an element with given key and then delete the element.

Disadvantage of chaining:-

* Elements with same hash value starts appearing in the linked list structure as shown in fig (b). Insertion, deletion & search operation requires extra time and tends to slow down the algorithm.

To overcome this disadvantage linear open addressing tech is implemented.

Linear open addressing:

* In this method, records that produce a collision, are stored at an alternate position in the hash table. An alternate location is obtained by searching the hash table until an unused position is found. This process is called **probing**.

* There are mainly two probing techniques.
  * Linear probing
  * Double hashing

* Linear probing:

* In this method, whenever there is a collision, the record is stored at the next empty position in the hash table. The hash table is considered to be a circular array such that after the last location, the search proceeds from first location of the table.
* Although linear probing is a very simple approach, it has disadvantage of clustering.

Clustering:

When the table becomes half full, there is a tendency towards clustering. This means that records start to appear in long strings of consecutive cells with gaps b/w the strings. Therefore the sequential search for an empty position becomes very time consuming.

The implementation of hash table using linear probing with key values {89, 18, 49, 58, 69} ~~is shown below~~ having hash table size 10 is as shown below.

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

89 % 10 = 9
18 % 10 = 8
49 % 10 = 9
58 % 10 = 8
69 % 10 = 9

(ii) Double Hashing :-

* It is the best method of resolving hashing as it uses double hash function.
* As the name indicates it does double hashing when collision exists.

The logic used in mapping the Elements using double hashing is

$$h_i(K) = [h(K) + i * h_p(K)] \% \text{ Hash table size}$$

Two Hash functions

$$h(K) = K \% \text{ Hash table size}$$

$$h_p(K) = 1 + K \% (\text{Hash table size} - 1)$$

where 'i' is the number of times collision occurs

* By using double hash functions we can easily search out the empty buckets and thereby avoiding the sequential search for of empty bucket as in linear probing

The implementation of Double hashing for key values {89, 18, 49, 58, 69} with Hash table size 10 is as shown below.

$h(K) = 89 \% 10 = 9$

$h(K) = 18 \% 10 = 8$

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

$49 \rightarrow h(K) = 49 \% 10$
$= 9$ - collision

$\therefore h_p(K) = 1 + 49 \% (10-1)$
$= 1 + 49 \% 9$
$= 1 + 4$
$= 5$

$h_i(K) = [9 + 1 \times 5] \% 10$
$= 14 \% 10$
$= 4$

$58 \rightarrow h(K) = 58 \% 10$
$= 8 \rightarrow$ collision

$\therefore h_p(K) = 1 + 58 \% (10-1)$
$= 1 + 58 \% 9$
$= 1 + 4$
$= 5$

$h_i(K) = [8 + 1 \times 5] \% 10$
$= 13 \% 10$
$= 3$

$69 \rightarrow h(K) = 69 \%$
$= 9$ - collision

$h_p(K) = 1 + 69 \% (10-1)$
$= 1 + 69 \% 9$
$= 1 + 6$
7

$h_i(K) = [9 + 1 \times 7]$
$= 16 \% 10$
$= 6$

* The Search and Insertion operation using linear open addressing is easier. But the problem arises during deletion operation.
* when an element is deleted from the hash table, the empty position is filled by marked item called tombstone.
* Either insertion overwrite these tombstones, but Look up treat them as collision.
* with out these tombstones, we might insert two elements with the same hash value, then remove the first one and leave the second item.

Double Hashing (prefer this for double hashing)

It does two hash functions when collision occurs. Hence the name double Hashing

Example:- compute the double hashing to insert keys { 18, 41, 22, 44, 59, 32, 31, 73 } in a Hash table of size = 13

The two hash functions are

$h_1(k) = k \% 13$

$h_2(k) = 8 - (k \% 8)$

44 (again collision)

→	44	41			18	32	31	73	22	→	89	→
0	1	2	3	4	5	6	7	8	9	10	11	12

$18 \rightarrow h_1(k) = 18 \% 13$
$= 5$

$41 \rightarrow h_1(k) = 41 \% 13$
$= 2$

$22 \rightarrow h_1(k) = 22 \% 13$
$= 9$

$44 \rightarrow h_1(k) = 44 \% 13$
$= 5 \rightarrow$ collision

$h_2(k) = 8 - (44 \% 8)$ (Shown in →)
$= 8 - 4$
$= 4$ (Move 4 Locations from the point of collision So 9th Location → again collision. Location 1

Move again 4 Locations.

89

$h_1(k) = 89 \% 13$

$= 11$

32

$h_1(k) = 32 \% 13$

$= 6$

31

$h_1(k) = 31 \% 13$

$= 5$ Collision

$h_2(k) = 8 - (31 \% 8)$ (Shown in =>)

$= 8 - 7$

$= 1$ → Move 1 Location from the point of collision 6th Location, again collision, move 1 location 7th location is free

73

$h_1(k) = 73 \% 13$

$= 8$

ADT

Abstraction refers to the act of representing essential features without including the background details / explanations.

* Classes use the concept of abstraction & are defined as a list of abstract attributes such as size, weight & cost & functions to operate on these attributes.
* They encapsulate all the essential properties of the object that are to be created. These attributes are called data members & functions are called as member functions.