**HASHING**

Store data in key-value pair

It used to reduce time complexity in searching and accessing data

**OBJECTIVE : need to optimize searching time**

Example :- 4 2 6 1 3

How can we store data?

Data Structure searching time insertion time

In an array : o(n) o(1)

Sorted array : o(log(n)) o(n)

(performing Binary Search i.e logn )

Linked List : o(n) o(1)

Balanced BST : o(log(n)) o(log(n))

**But we want searching and insertion in o(1)**

**Hashing:** Now we still store data in an array but in non sequence manner

Like lets store data in arr[i]=i

Here searching and sorting is in o(1);

DRAWBACK: lets say need to store data :- 3 2 1 4000

Here we need array of size 4001 i.e lots of memory is wasted

Lets say a better approach :- lets pass an integer in HA(Hashing function) and give index to store data

Lets say= HA=i%10=index;

To store 3 => 3

To store 45 =>5

To store 1 =>1

To store 4000 =>0

Now can overcome issue of wasting huge memory but there is another drawback->

How to store 1,11,21,31 etc.

This is called collision

Collision: Single key mapping to multiple values

Collision Resolution: To solve linear hashing

It is of major 2 types

* Separate Chaining
* Open Addressing

Open addressing

* Linear Probing
* Quadratic probing
* Double Hashing

Separate Chaining

**Linear Probing:** if key found for storing value is already occupied then go to next unoccupied space linearly and store data there.

Example:- need to store 1,11,21

In an array we store

To store 1=> 1

To store 11=>2(as 1 is occupied)

To store 21 =>3(as 1 and 2 is occupied)

But as array reaches to end index then it starts again at 0 index

DRAWBACK: in worst case, insertion is o(n)

DRAWBACK: in worst case, selection is o(n)

**Quadratic Probing:** if key found for storing value is already occupied then go to next unoccupied space non-linearly and store data there.

Hf(key)=key%10

**Here HA=index=hf(key)+i+i^2**

Here index is to store data;

**Where i is the attempt to store data, i is incremented every time.**

Like store 1 , 11, 21

To store 1=> 1

(1+0+0)

To store 11=>3(as 1 is occupied)

(1+0+0)=1(occupied)

(1+1+1)=3(Not occupied)

To store 21 =>7(as 1 and 2 is occupied)

(1+0+0)=1(occupied)

(1+1+1)=3(occupied)

(1+2+4)=7(Not occupied)

Drawback: same type of value-> following same path everytime

**Double Hashing:** here key is generated through the combination of 2 hashing functions

**Here HA=index= hf1(key)+hf2(key)**

(can store specific amount of data only)

Lets store value at key which square root of value like

Key: 1 6 3 4 5 9

Value: 1 36 09 16 25 81

**SEPARATE CHAINING**

What we exactly do in separate chaining -> create linked list at each index

**Time complexity** =>

DRAWBACK:- what if linked list got bigger then it will take longer time to be searched

* To overcome Drawback => we will calculate load factor
* Like N node and b space so
* Load factor=n/b=0.8(let say 8/10)
* As load factor goes above 0.75 then we will re-hash

What is re-hash?

* Decrease Load Factor
* i.e Increase Size of bucket(array)
* size of an array (bucket) is increased by 1.5 times of previous one

Re-hash means each node is hashed again depending on new hashed function.

**HF=index=key % size**

Like when size is 10 -> 24 goes to 4

After rehashing

Like when size is 15 -> 24 goes to 9