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)]

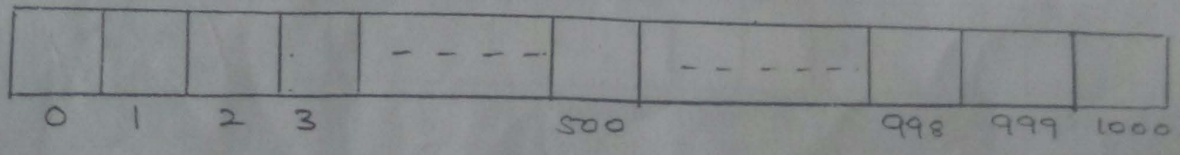
Hash function and Hash table :-

Hashing uses a 'Hash function' to map keys into position 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



* To Search the Students record with ID number 951002, 951500, 951998

$f(k) = k - 951000$

$f(951002) = 951002 - 951000$

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

$f(951998) = 951998 - 951000$

$= 998 \rightarrow$ The student record with ID number 951998 found at position '998' of hash table

$f(951500) = 951500 - 951000$

$= 500 \rightarrow$ 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

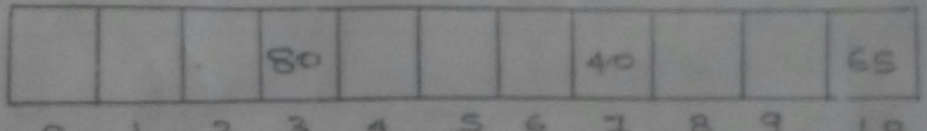
Bucket :- Each position in the hash table is called a 'Bucket'

* 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 % 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 :-

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 divi-der 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 a position 3

16 49 82 at 5 [∵ f(k) = k % D]

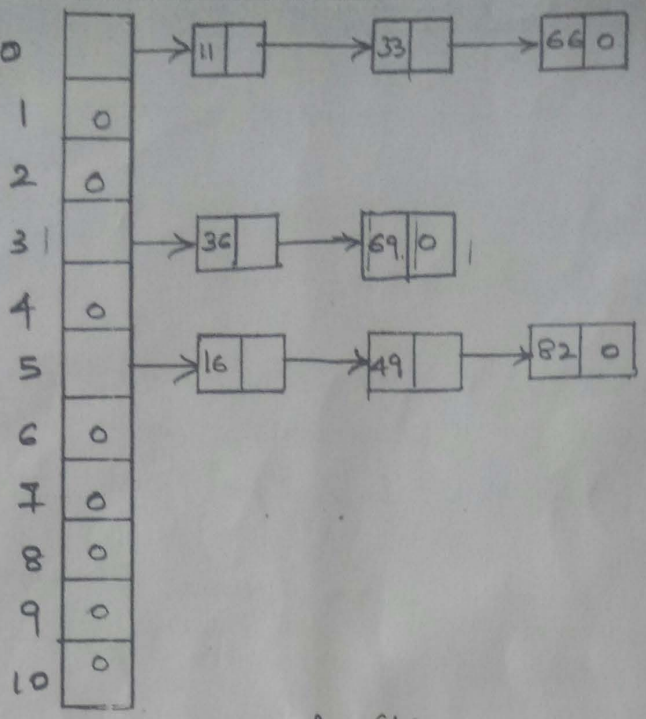
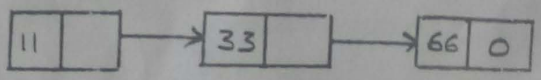


fig (b)

* 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 operations requires extra time and tends to slow down the algorithm.

To overcome this disadvantage linear open addressing technique is implemented.

Linear open addressing :-

* In this method, records that produce 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 starts 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 is/ shown below. The with key values {89, 18, 49, 58, 69} 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 co
* As the name indicates it does double hash functi
when collision exists

The logic used in mapping the Elements using double hashing is

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

Two Hash functions

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

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

where 'i' is the number of times collision occurs

* By using double hash functions we can easily search out the empty buckets and Hereby avoiding the Sequen-ntial 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

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

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

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

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

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

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

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

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

* 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.
* Further 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)

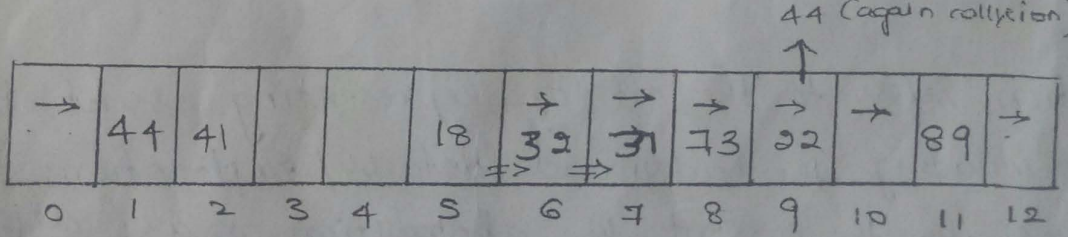
It does two hash functions when collision occurs. Hence the name double Hashing

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

$h_1(k) = k \% 13$

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



18 → $h_1(k) = 18 \% 13$
$= 5$

41 → $h_1(k) = 41 \% 13$
$= 2$

22 → $h_1(k) = 22 \% 13$
$= 9$

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

$h_2(k) = 8 - (44 \% 8)$ (shown in →)
$= 8 - 4$
$= 4$ (Move 4 locations from the point of collision

9th location → again collision So Move again 4 locations. Location 1

89

$h_1(k) = 89 \% 13$

$= 11$

32

$h_1(k) = 32 \% 13$

$= 6$

31

$h_1(k) = 31 \% 13$

$= 5$ Collision

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

$= 8 - 7$

$= 1$ → Move 1 Location from the point of collision

6th Location, again collision, move 1 Location 7th Location is free

73

$h_1(k) = 73 \% 13$

$= 8$