UNIT-2

Introduction

Hashing:- The implementation of the hash table is frequency called as hashing.

→ Hashing is a technique which can be used to perform the operations such as insertion, deletion, search / find operations. in average constant time of Big-oh ⇒ O(1)

→ The ideal hash table data structure is a fixed size array by dividing that hash table into the numbers of locations which is equal to the size of the hash table.

→ where each location is capable of storing the given keys into the hash table.

→ The hash table can be indexed from zero to n-1 (0 to n-1) where n is the size of the hash table.

→ The hash table implementation can be done by using hash function only. that is nothing can be done without hash function in hash table

Hashing organisation:-

The organization of the hash table can be done by using → bucket
→ hash function.

Bucket:- A bucket a storage location of the hash table.

→ The hash table can be divided into the no. of buckets depending on given hash table size.

bucket is capable of storing the given keys.

→ The hash table operations can be perform only with in the buckets of hash table.

Hash Function:- The hash function is a mapping function which maps the given search keys into the buckets of the hash table.

i.e., it is a function from search keys to hash table locations.

Types of Hashing:-

The Hashing can be divided into two types.

which are 1) static Hashing
2) Dynamic Hashing.

1) static Hashing:-

In static Hashing the size of the hash table is fixed.

i.e., no growing (or) shinking during the implementati of the hash table.

2) Dynamic Hashing:-

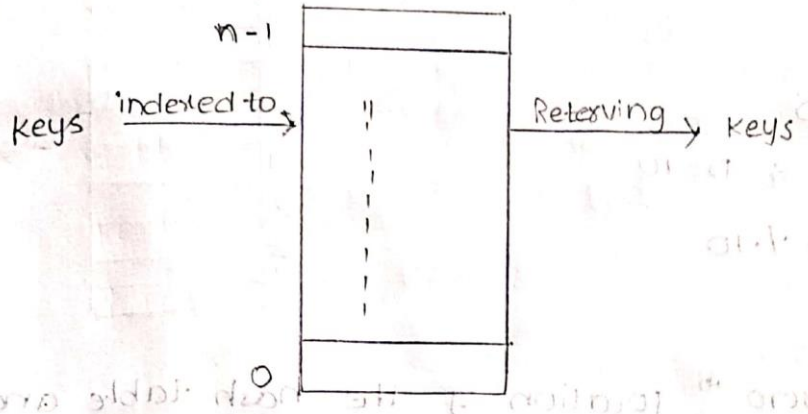
In Dynamic Hashing the size of the hash table is variable i.e., the size of the hash table may grow (or) shrink during the implementation of the hash table.

Static Hashing:

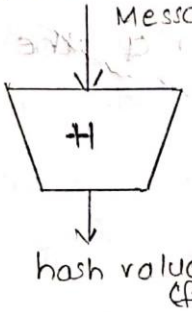
Hashing: Hashing is the process of indexed to and retriving the keys from hash table.

Hash Table: The hash table is of fixed size array in static hashing. The hash table contains the number of buckets ~~dividing~~ divided from 0 to n-1

→ The keys are inserted into and retrive from the hash table by using hash function.



Hash Function: The hash function is a mapping function which maps the given message of arbitary length to the buckets of the hash table by retriving output as hash value of fixed size.



Ex:- Insert the keys 50, 62, 78, 11, 15, 19, 47 into the hash table of size 10]X

The hash function can be ~~given~~ defined as $h(k) = k \% D$

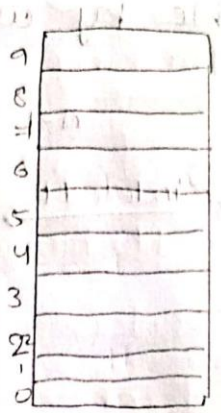
where k : key
D : size of hash table.

→ The hash function always returns a value is called as hash value (or) Message digest.

Ex:- Insert the keys 50, 62, 78, 11, 15, 19, 47 into the hash table of size 10.

→ The hash table can be created with 10 locations by indexing from 0 to 9. as the given size is of 10.

Hash table:-



For key 50:

$h(K) = K \% D$

where $k = 50$ & $D = 10$

$h(50) = 50 \% 10$

$= 0$

→ Go to the zero th location of the hash table and insert the key 50.

For key 62:

$h(62) = 62 \% 10$

$= 2$

→ Go to the 2nd location of the hash table and insert the key 62.

For key 78:

$h(78) = 78 \% 10 = 8$

→ Go to the 8th location of the hash table and insert the key 78.

For key 11:

$h(11) = 11 \% 10 = 1$

→ Go to the 1st location of the hash table and insert the key 11.

For key 15:

$h(15) = 15 \% 10 = 5$

→ Go to the 5th location of the hash table and insert the key 5.

For key 19:

$h(19) = 19 \% 10 = 9$

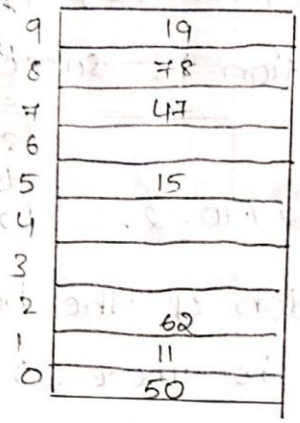
→ Go to the 9th location of the hash table and insert the key 9.

For key 47:

$$h(47) = 47 \% 10 = 7$$

→ Go to the 7th location of the hash table and insert the key 47.

After inserting all these keys to the hash table. The hash table is.



Delete the keys 15 and 44 from the above hash table.

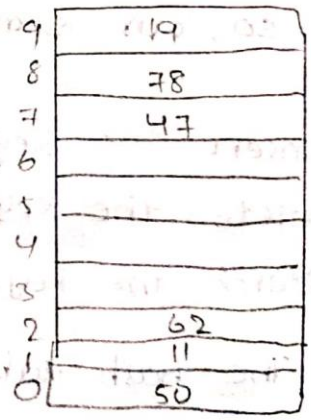
For key 15:

$$h(15) = 15 \% 10 = 5$$

→ Go to 5th location of the hash table and check if the given key is there (or) not.

→ The given key 15 exists in 5th location of the hash table. so delete the key 15 from 5th location of the hash table.

→ After deletion the hash table is.



For key 44:

$$h(44) = 44 \% 10 = 4.$$

→ Go to 4th location of the hash table and check whether the given key is ~~not~~ there (or) not.

→ In 4th location the given key 44 is not there so, No deletion can be done.

Search the keys 19, 62 & 59 in the above hash-table.

For key 19:

$$h(19) = 19 \% 10 = 9$$

→ Go to the 9th location of the hash-table and check whether the given key is there (or) not.

→ In 9th location the given key 19 is there.

so, our search operation is successfull.

For key 62:

$$h(62) = 62 \% 10 = 2.$$

→ Go to the 2nd location of the hash-table and Search the given key is there (or) not.

→ In second location the given key 62 is there.

so, search operation is succesfull.

For key 59:

$$h(59) = 59 \% 10 = 9$$

→ Go to the 9th location of the hash-table and search whether the given key is there (or) not.

→ In 9th location the given key 59 is not there.

so, our search operation is unsuccessfull.

Ex: Insert the keys 17, 41, 110, 7, 123, 148

Delete the keys 110, 140.

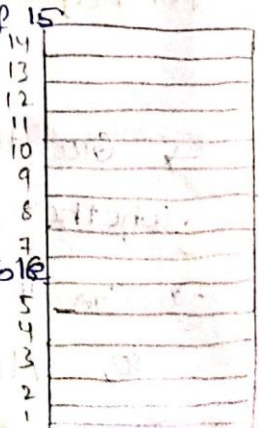
Search the keys 7, 75 in the hash table of size 15.

→ The hash-table can be created with 15 locations by indexing from 0 to 14 as the given size of 15 hash-table:-

For key 17:

$$h(K) = K \% D = 17 \% 15 = 2$$

→ Go to the 2nd location of the hash table and insert the key 17.



For key 41:

$$h(41) = 41 \% 15 = 11$$

→ Go to the 11th location of the hash table and insert the key 41.

For key 110:

$$h(110) = 110 \% 15 = 5$$

→ Go to the 5th location of the hash table and insert the key 110.

For key 7:

$$h(7) = 7 \% 15 = 7$$

→ Go to the 7th location of the hash table and insert the key 7.

For key 123:

$$h(123) = 123 \% 15 = 3$$

→ Go to the 3rd location of the hash table and insert the key 3.

For key 148:

$$h(148) = 148 \% 15 = 13.$$

→ Go to the 13th location of the hash table and insert the key 148.

After inserting all the keys to the hash table. The hash table will be

14	
13	148
12	
11	41
10	
9	
8	
7	7
6	
5	110
4	
3	123
2	17
1	
0	

Deleting the keys:

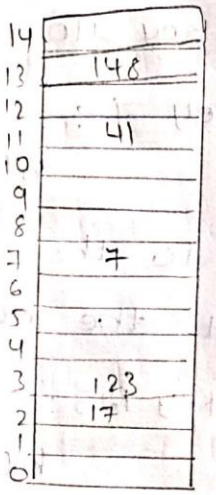
For key 110:

$$h(110) = 110 \% 15 = 5$$

→ Go to 5th location of hash table and check if the given key is there or not

→ The given 110 is exists in 5th location of the hash table. so delete the key 110 from 5th location of the hash table.

→ After deletion the hash table is



For key 140:

$$h(140) = 140 \% 15 = 5$$

→ Go to the 5th location of the hash table and check if the given key is there (or) not

→ The given 140 is not exists. so, no deletion can be done.

Search the keys:

For key 7:

$$h(7) = 7 \% 15 = 7$$

→ Go to the 7th location of hash table and check whether the given key is there (or) not.

→ In 7th location the given key 7 is there. so, our search operation is succesfull.

For key 75:

$$h(75) = 75 \% 15 = 0$$

→ Go to the 0th location of hash table & search the given key is there (or) not.

→ In 0th location, the given key '75' is not there so, our search operation is unsuccessful.

Feauters of Hash function:-

1. The Hash function genesates fixed length output data (Hash value)
2. The Hash function Converts orbitary length data to fixed length data, this process can be called as hashing the data
3. The hash functions can also be called as Compressed functions. As it compress the given large data into smaller representation.
4. The hash value generated by the hash function is the smaller representation of the keys that's why the hash value can be called as digest.
5. If the hash value is of n bit then those many n bits hash functions can be used to generate hash values.

→ The hash value ranges from 160 - 512 bits / 20 - 64 bytes.

Secure Hash Function (SHA):

While inserting the keys into the buckets of the hash table, if more than one key wants to enter into the same location then it uses secure hash functions to provide security by avoiding collisions.

→ The SHA Family available in four types of algorithms which are

- SHA-0
- SHA-1
- SHA-2
- SHA-3

SHA-0:

SHA-0 is the original ~~fun~~ version of secured hash function is ~~one~~ 160 bit and proposed by National Institue OF Standard Technologies (NIST).

* It had few weakness and didn't became the much popular.
* In this situation the NIST called for a new secure hash function in 1995 to correct the weakness of SHA-0.

SHA-1:

The SHA-1 algorithm designs a new secure hash function. in the year 1995. to release all the weakness occured in SHA-0.

* The SHA-1 hash function widely employable in applications and protocalls by including Secure Socket layer (SSL)
* The long term employability of SHA-1 is doubtfull.

so, the next version SHA-2 invited in the year 2003.

SHA-2:

SHA-2 secure hash function is the ~~storg~~ strongest hash function which can offers four variance of hash values.

They are

SHA - 224
SHA - 256
SHA - 384
SHA - 512 by depending the no of bits.

SHA-2 algorithm is not became that much popular as it uses still the design of SHA-0.

* In this situation the NIST develop a new secure hash function in October 2012.

SHA-3:

The SHA-3 secure Hash function develop by using Keccak algorithm as it offers many benfits such as efficient performance at good resistance from attacks.

** Collision Resolution / overflow Handling Techniques:-

Collision: When more than one key wants to enter into the same location of the hash table, then that situation can be called as Collision.

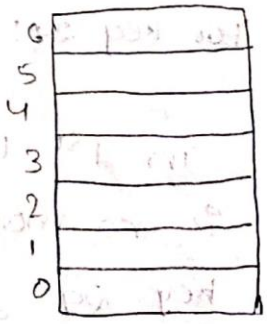
Overflow: If the key insertion exceeds the level of defined bucket capacity then it results in over flow condition.

Ex:- Insert the keys 43, 51, 15, 16, 29 of table size D=7

→ For key 43:

$h(k) = k \% D$

$h(43) = 43 \% 7 = 1$



→ Go to the 1st location and store 43

For key 51:

$h(51) = 51 \% 7 = 2$

→ Go to the 2nd location and ~~store~~ 51. check whether that location is empty (or) not.

For key 15:

$h(15) = 15 \% 7 = 1$

The second location is empty. so, insert the key 51 into the table.

For key 15:

$h(15) = 15 \% 7 = 1$

[→ Go to the 1st location]

Here the 1st location of the hash table was already occupied by key 43 and the key 15 also wants to enter into the same location of the hash table. so, it results in collision.

* consider the capacity of each bucket of the hash table is 2.
* When collision occur at 1st location some more space available to insert another key.

so, insert key 15 in 1st location of the hash table.

For key 16:

$h(16) = 16 \% 7 = 2.$

In 2nd location already the key 51 is there. so, it results in collision to insert another key. under this condition only insert key 16 as free space is available for that.

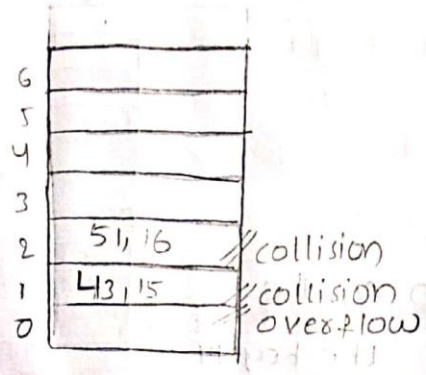
For key 29:

$h(29) = 29 \% 7 = 1$

In 1st location already 2 keys are there and there is no more space to insert another key but the key 29 also wants the same location to insert.

* Here if the insertion can be done it exceeds the defined capacity level by resulting in overflow condition.

The Hash Table becomes as



To avoid collision / overflow there are two types of hashing techniques.

They are 1) closed Hashing
2) open Hashing.

1) closed Hashing: If any collision occured in closed hashing then those collisions can be resolve by using Seperate chaining method.

→when collision occured in closed hashing then that location only can be used to resolve that collision closedly. i.e., The collide bucket only extended to resolve the collision.

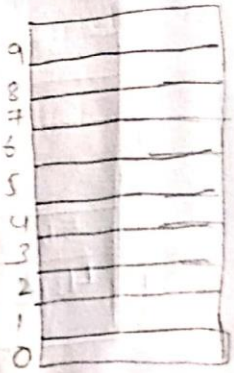
Seperate chaining Method: In seperate chaining method a chain can be created through single linked list which can be connected (or) linked to each and every bucket of the hash table.

→ When collision occured then the keys are inserted in the considered linked chain to that bucket.

→ If More keys are there to insert into the same location then the chain will be extended by assigning a new node to the next pointer variable.

→ If No more keys are there to insert then assign the next pointer to null.

Ex:- Insert the keys 17, 54, 27, 39, 4, 59, 24, 6 into the hash table of size 10.



→ For key 17:

$h(k) = k \% 10$

$h(17) = 17 \% 10 = 7$

→ Go to the 7th location of the hash table and insert the key 17 in data field of the node.

For key 54:

$h(54) = 54 \% 10 = 4$

→ Go to the 4th location of the hash table and insert the key 54 in data field of the node.

For key 27:

$h(27) = 27 \% 10 = 7$

→ Go to the 7th location of the and insert the key 27 by creating a new node which linked to the next pointer of previous node.

For key 39:

$h(39) = 39 \% 10 = 9$

→ Go to the 9th location of the hash table and insert the key 39 in data field of the node.

For key 4:

$h(4) = 4 \% 10 = 4$

→ Go to the 4th location and insert the key 4 by creating a new node which linked to the next pointer of previous node.

For key 59:

$h(59) = 59 \% 10 = 9$.

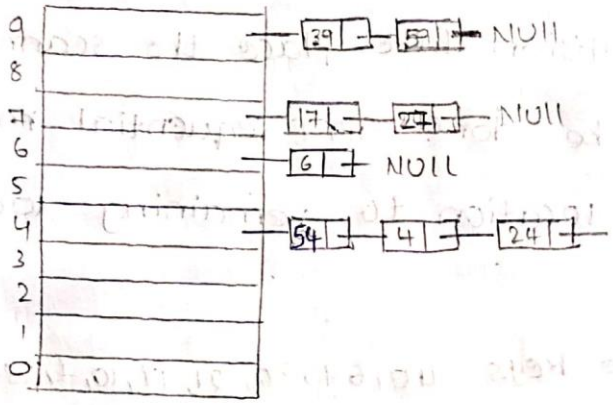
→ Go to the 9th location and insert the key 9 by creating a new node which linked to the next pointer of previous node.

For key 24: $h(24) = 24 \% 10 = 4$.

→ Go the 4th location and insert the key 24 by creating a new node which linked to the next pointer of previous node.

For key 6: $h(6) = 6 \% 10 = 6$

→ Go the 6th location of the hash table and insert the key 6 in data field of the node.



Drawbacks of seperate chaining Method:

1. The maintance of linked list for each and every bucket of the hash table is burden of the user.
2. It is very time consuming process and requires more space for the creation of nodes.

2) Open Hashing: In open hashing if any collision occures then we have to search for an open slot (empty location) to insert that collide key into the hash table.

→ The open hashing can be done by using three techniques

- Linear probing
- Quadratic probing
- Double probing.

Linear probing:

Probe: Probe is an attempt to find out an empty location to insert the collide key into the hash table.

→ In linear probing the probing can be done in ~~search~~ sequential order to find out an empty location when collision occured.

i.e., when collision takes place the scanning of the locations can be done in sequential manner from that collide location to remaining locations in hash table

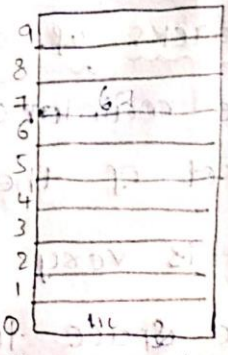
Ex:- Insert the keys 40, 67, 30, 21, 17, 10, 7, 28 into the hash table of size 10.

→ Consid

For key 40:

$h(K) = K \% D$

$= 40 \% 10 = 0$



→ Go to the 0^{th} location and insert the key 40 into hash table.

For key 67: $67 \% 10 = 7$

→ Go to the 7^{th} location and insert the key 67 into the hash table.

For key 30: $h(30) = 30 \% 10 = 0$.

→ Here the 0^{th} location already occupied by the key 40. so, it results in collision.

This collision can be resolve by finding out an empty location using linear probing to insert that collide key.

→ An empty location can be found by searching for an empty slot from that collide place to remaining locations from the hash table.

→ From collide location 0, the next location 1 is empty. so, the insert the collide key 30 in 1st location of the hash table.

For key 21: $h(21) = 21 \% 10 = 1$

→ Here the 1st location already occupied by the key 30. so, it results in collision.

→ From that 1st location the next open slot is 2. so, insert that collide key into 2nd location of the hash table.

For key 17: $h(17) = 17 \% 10 = 7$.

→ At 7th location collision occured. so, the next free slot from 7th location is location 8. so, insert that collide key into 8th location of the hash table.

For key 10: $h(10) = 10 \% 10 = 0$

→ At 0th location collision occured. so the next free slot from 0th location is location 3. so, insert that collide key into 3rd location of the hash table.

For key 7: $h(7) = 7 \% 10 = 7$

→ A 7th location collision occured. so the next free slot location is 9. so insert that collide key into 9th location of the hash table.

For key 28: $h(28) = 28 \% 10 = 8$.

→ Here the 8th location collision occured. so, the next free slot location is 4. so insert that collide key into 4th location of the hash table.

After linear probing the hash table is

9	7
8	17
7	67
6	
5	
4	28
3	10
2	21
1	30
0	40

The number of probs required for the collides keys are

h(30) = 1 probe

h(21) = 1 probe

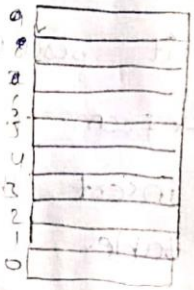
h(17) = 1 probe

h(10) = 3 probes

h(7) = 2 probes

h(28) = 6 probes

Ex:- Insert the keys 50, 35, 15, 37 into hash table of size 7.



→ For key 50: h(k) = k%.D

h(50) = 50%.7

= 1

→ Go to the 1st location of the hash table and insert the key 50 into hash table.

For key 35: h(35) = 35%.7

= 0

→ Go to the 0th location of the hash table and insert the key 35 into hash table.

For key 15: h(15) = 15%.7 = 1

→ Here 1st location already occupied by the key 50 so, it results collision.

so, the next free slot from 1st location is location 2. so, insert collide key 15 into 2nd location of hash table.

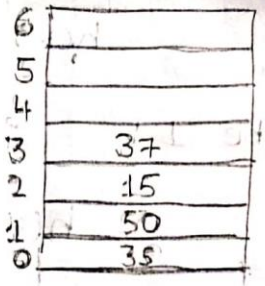
For key 37: h(37) = 37%.7 = 2

→ Here 2nd location already occupied. so, the next free slot from 2nd location is location 3. so, insert collide key 37 into 3rd location of hash table.

The number of probs required for the collides keys are

$h(15) = 1$ probe

$h(37) = 1$ probe.



Quadratic probing:

In Quadratic probing the number of probes to find out an empty location will be reduced by using a new hash function.

$$h(k) = (h(k) + i^2) \% D$$

where $i = 1, 2, 3, 4$

This Hash function can be used when Collision occured in the Hash table otherwise the normal hash function $h(k) = k \% D$

Can be used to insert the given keys into the hash table.

Ex:- Insert the keys 14, 36, 74, 12, 66, 82 into the hash table of size 10

→ For key 14:

$$h(14) = 14 \% 10 = 4$$

→ Go to the 4th location, and insert the key 14 in 4th location of the hash table.

For key 36: $h(36) = 36 \% 10 = 6$

→ Go to the 6th location and insert the key 36 in 6th location of the hash table.

For key 74: $h(74) = 74 \% 10 = 4$,

→ Here the collision occured at 4th location.

→ Resolve this collision using the hash function

$$h(k) = (h(k) + i^2) \% D$$

Probe 1: $i = 1$

$$h(74) = (4 + 1^2) \% 10 = 5$$

→ Go to 5th location and if it empty insert the key 74 into the hash table

For key 12:

$$h(12) = 12 \% 10 = 2$$

→ Go to 2nd location and if insert the key 12 into the 2nd location of hash table.

For key 66:

$$h(66) = 66 \% 10 = 6$$

→ Here the collision occured at 6th location.

→ Resolve this collision using hash function.

$$h(k) = (h(k) + i^2) \% D$$

probe 1: $i = 1$

$$h(66) = (6 + 1^2) \% 10 = 7 \% 10 = 7$$

→ Go to the 7th location and if it is empty insert the key 66 into the hash table.

For key 82:

$$h(82) = 82 \% 10 = 2$$

X

[→ Go to the 2nd location and insert the key 82 into 2nd location of hash table.] X

→ Here the collision occured at 2nd location,

→ Resolve this collision using hash function.

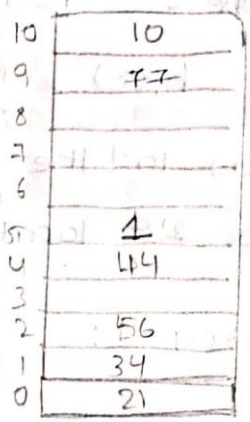
$$h(k) = (h(k) + i^2) \% D$$

probe 1: $i = 1$

$$h(82) = (2 + 1^2) \% 10 = 3 \% 10 = 3$$

→ Go to 3rd location and insert the key 82 in 3rd location of hash table.

Ex:- Insert the key 10, 21, 34, 44, 56, 1, 77 into hash table of size 11.



For key 10:

$$h(10) = 10 \% 11 = 10$$

→ Go to the 10th location and insert the key 10 into 10th location of hash table.

For key 21:

$$h(21) = 21 \% 11 = 10$$

→ Here the collision occured at 10th location.

→ Resolve this collision using hash function.

$$i=1 \Rightarrow h_i(k) = (h(k)+i^2) \% D = (10+1^2) \% 11 = 11 \% 11 = 0$$

→ Go to the 0th the location and insert the key 21 into 0th location of hash table.

For key 34:

$$h(34) = 34 \% 11 = 1$$

→ Go to the 1st location and insert the key 34 into 1st location of hash table.

For key 44:

$$h(44) = 44 \% 11 = 0$$

→ Here the collision occured at 0th location.

→ Resolve this collision using hash function.

probe 1: $i = 1$

$$h_i(k) = (h(k) + i^2) \% D = (0+1) \% 11 = 1 \% 11 = 1$$

probe 2: $i = 2$

$$h_i(k) = (h(k) + i^2) \% D = (0 + 2^2) \% 11 = 4 \% 11 = 4$$

→ Go to the 4th location and insert the key 44 into 4th location of hash table.

For key 56:

$$h(56) = 56 \% 11 = 1$$

→ Here the collision occured at 1st location.

→ Resolve this collision using hash function.

probe 1: $i = 1$

$$h_i(k) = (h(k) + i^2) \% 11 = (1 + 1^2) \% 11 = 2 \% 11 = 2$$

→ Go to the 2nd location and insert the key 56 into 2nd location of hash table.

For key 1:

$$h(1) = 1 \% 11 = 1$$

→ Here the collision occured at 1st location

→ Resolve this collision using hash table.

probe 1: $i = 1$

$$h_i(k) = (1 + 1^2) \% 11 = 2 \% 11 = 2$$

probe 2: $i = 2$

$$h_i(1) = (1 + 2^2) \% 11 = 5 \% 11 = 5$$

→ Go to the 5th location and insert the key 1 into 5th location of the hash table.

For key 77:

$$h(77) = 77 \% 11 = 0.$$

→ Here the collision occured at 0th location.

→ Resolve this collision using hash table.

probe 1: $i = 1$

$$h_i(k) = (0 + 1^2) \% 11 = 1 \% 11 = 1$$

probe 2: $i = 2$

$$h_i(k) = (0 + 2^2) \% 11 = 4 \% 11 = 4$$

probe 3: $i = 3$

$$h_i(k) = (0 + 3^2) \% 11 = 9 \% 11 = 9$$

→ Go to 9th location and insert the key 77 into the 9th location of the hash table.

Draw back :-

* In quadratic probing also there is no gasante to finding out an empty location when collision occured.

i.e., this technique gives only four chances to find out an empty location when collision takes place.

Double Hashing: In double hashing two hash functions are used to avoid the collisions while inserting the given keys into the hash table.

Intially all keys are inserted into the hash table using the first hash function

$$h_1(k) = k \% D$$

where K = key

D is the size of the table.

If there is any collision the double hashing provides second hash function

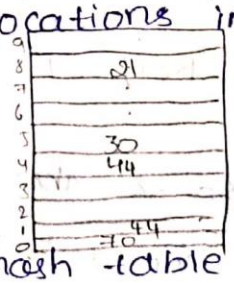
$$h_2(k) = R - (k \% R)$$

where R : The first prime < table size

To avoid the collisions.

Ex:- Insert the keys 70, 44, 30, 14, 21, 65 into the hash table of size 10.

→ Consides the hash table with 10 locations indexing from 0 to 9. and



For key 70: $h_1(70) = 70 \% 10 = 0$

→ Go to the 0th location of the hash table and insert the key 70 into 0th location of hash table.

For key 44: $h_1(44) = 44 \% 10 = 4$

→ Go to the 4th location of the hash table and insert the key 44 into 4th location of hash table

For key 30:

$$h_1(30) = 30 \% 10 = 0$$

→ The 0th location already occupied by the key 70. So, it results in collision.

→ To avoid this collision use 2nd hash function.

$$h_2(K) = R - (K \% R)$$

where R = 7 which is the first prime < table size

$$h_2(30) = 7 - (30 \% 7) = 7 - 2 = 5$$

→ Go 5 locations away from the collide location 0 which is location 5. So, insert the key in 5th location of the hash table.

For key 14: $h_1(14) = 14 \% 10 = 4$

→ The 4th location already occupied by the key 44 so, it results in collision.

→ To avoid this collision use 2nd hash function

$$h_2(14) = 7 - (14 \% 7) = 7 - 0 = 7$$

→ Move 7 locations away from 4th location which is location 1. So, insert the key 1st location of the hash table.

For key 21: $h_1(21) = 21 \% 10 = 1$

→ Go to the 1

→ The 1st location already occupied by the key 14, so, it results in collision,

$$h_2(21) = 7 - (21 \% 7) = 7$$

→ Move 7 locations away from 1st location which is location 7. So, insert the key 7th location of the hash table.

For key 65: $h_1(65) = 65 \% 10 = 5$

→ The 5th location already occupied by the key 30 so, it results in collision.

$$h_2(k) = 7 - (65 \% 7) = 7 - 2 = 5$$

→ Move 5 locations away from the collide location 5 which is location 0 this is not empty.

→ so, again move 5 locations away from 0th location and check the collision is empty as not to insert collide key into hash table.

→ continue this process until an empty location will found otherwise determined that these is no chance to insert the key into the hash table

→ For key 65 these is no empty location. i.e., no chance to insert the key into the hash table.

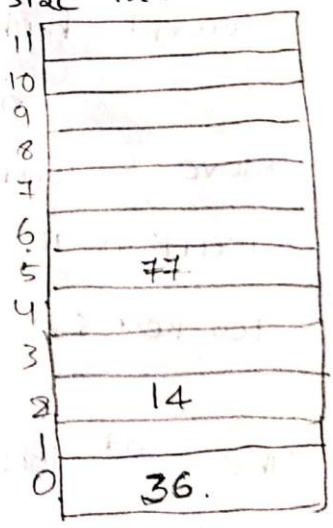
Ex:- Insert the keys 77, 36, 14, 5, 26, 62, 29 of size 12.

→ consider the hash table

For key 77:

$$h_1(k) = 77 \% 12 = 5$$

→ Go to the 5th location and insert the key 77 into 5th location of the hash table.



For key 36: $h_1(36) = 36 \% 12 = 0$

→ Go to the 0th location and insert the key 36 into 0th location of the hash table.

For key 14: $h_1(14) = 14 \% 12 = 2$

→ Go to the 2nd location and insert the key 14 into 2nd location of the hash table.

For key 5: $h_1(5) = 5 \% 12 = 5$

↳ Go to the 5th location check whether it is empty or not. Here 5th location was already occupied by the key 77, go to the next hash function.

$$h_2(5) = 7 - (5 \% 7) = 7 - 5 = 2.$$

Move 2 locations away from location 5 which is location 7. So, insert the key 5 into 7th location.

For key 26:

$$h(26) = 26 \% 12 = 2$$

The second location is already occupied by the key 14. So, go to the next hash function.

$$h_2(26) = 7 - (26 \% 7) = 7 - 5 = 2.$$

Move 2 locations above from the collide location.

For key 29:

$$h(29) = 29 \% 12 = 5$$

TO location 5 is already occupied with key 17. So, go to second hash function.

$$h_2(29) = 7 - (29 \% 7) = 7 - 1 = 6.$$

Move 6 locations away from the 5th location which is location 11. So, insert the key 29 at 11th location.

For key 62:

$$h(62) = 62 \% 12 = 2$$

The 2nd location is already occupied with the key 77 and go to next hash function.

$$h_2(62) = 7 - (62 \% 7) = 7 - 6 = 1$$

Move 1 location away from the collide location which is location 3 and insert the key 62 at 3rd location.

Drawbacks:

1. Finding of an empty location to insert a new element into the hash table when collision occur is difficult.

If more than half of the table gets full.

Rehashing:

when a new element inserted into the hash table it is very difficult in finding of an empty location by using open / closed hashing techniques.

* This problem can be achived by reconstructing a new hashtable with large size.
* In Rehashing the given keys are remapped into new constructed hash table with new hash function, of it's new increasing size.

The table can Rehash when

1) when the table gets more than half full.
2) when an insertion fails and finding of empty location.
3) when the table capacity reaches it defined load factor.

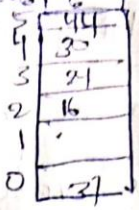
* In rehashing a new hash table can be constructed by doubling the previous size to its next prime number This is the size of new constructed hash table.

→ In this newly constructed hash-table the given keys are remapped with new table size. If any collision occured then use any open / closed hashing techniques to resolve the collision.

Generally the linear probing we to resolve the collisions.

Ex:- Insert the keys 16, 24, 30, 44, 6, 31 into the hashtable of size 7.

Consider the hash-table with 7 location. Indexing from 0-6.



For key 16: $h(16) = 16 \% 7 = 2$

Go to the 2nd location insert the key 16 into the hashtable.

For key 24: $h(24) = 24 \% 7 = 3$

Go to the 3rd location insert the key 24 into the hash-table.

For key 30: $h(30) = 30 \% 7 = 2$.

→ In 2nd location already key 16 is there, it results in collision.

→ TO resolve this collision we linear probing. By using linear probing the next empty slot to insert key 30 is location 4, so insert the collide key into 4 location of hash table

For key 44: $h(44) = 44 \% 7 = 2$

In 2nd location already key 16 is there, it results in collision;

TO resolve this collision use linear probing.

By using linear probing the next empty slot to insert key 44 is location 5. so, insert the collide key into 5 location of hash-table.

For key 6: $h(6) = 6 \% 7 = 6$.

→ Go to the 6th location insert the key 6 into the hashtable.

For key 31: $h(31) = 31 \% 7 = 3$.

In 3rd location already key 24 is there, it results in collision.

To resolve this collision use linear probing. By using linear probing the next empty slot to insert key 31 into 0th location.

Here, the table is more than half full. Now reconstruct the hashtable by doubling the size with its next prime number is 17.

Table size: $7+7 = 14$ Prime 17

The new hash function is

$h(k) = k \% 17$, Now, re map the keys

For key 16: $h(k) = k \% 17 = 16 \% 17 = 16$.

Go to the 16th location of the hashtable & insert the key 16.

For key 24: $h(24) = 24 \% 17 = 7$

Go to the 7th location of the hashtable and insert the key 24.

For key 30: $h(30) = 30 \% 17 = 13$

Go to the 13th location of the re-hashed table and insert the key 30.

For key 44: $h(44) = 44 \% 17 = 10$

Go to the 10th location of the re-hashed table and insert the key 44.

For key 6: $h(6) = 6\%17 = 6$.

Go to 6th location and insert the key 6.

For key 31: $h(31) = 31\%17 = 11$

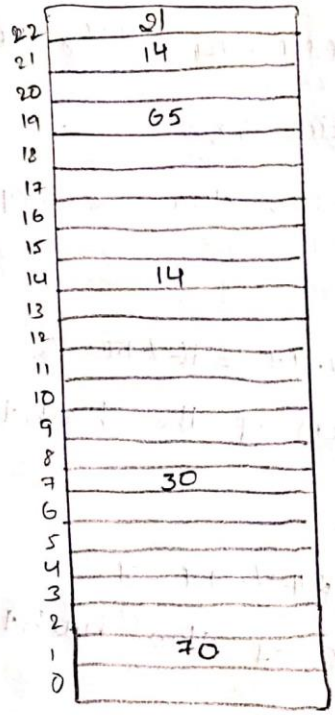
Go to the 11th location of the re-hash table and insert the key 31.

Insert the keys 70, 44, 30, 14, 21, 65 into the hash table of size 10.

As we done the insertion by double hashing previous there is no empty location for the key 65.

Now, re-construct the hash table by doubling the size with it's next prime number is 23.

Construct Hash table with size of 23.



For key 70: $h(70) = 70\%23 = 1$

Go to the 1st location of the hash table and insert the key 23.

for key 44: $h(44) = 44\%23 = 21$

Go to the 21st location of the hash table and insert the key 44.

for key 30: $h(30) = 30 \% 23 = 7$.

Go to the 7th location of the hash table and insert the key 30.

for key 14: $h(14) = 14 \% 23 = 14$

Go to the 14th location of the hash table and insert the key 14.

for key 21: $h(21) = 21 \% 23 = 21$.

The 21st location is already occupied by the key so, by linear probing the next empty slot is location 22. Insert at 22nd location.

for key 65: $h(65) = 65 \% 23 = 19$.

Go to the 19th location of the re-hashed table and insert the key 65.

Drawback:-

It is very expensive to construct a new Hashtable.

Extendible Hashing: In open (or) closed hashing techniques to perform any operation it requires several access on hash table.

→ To minimize the number of access a new technique will be used which is extendible hashing

→ In extendible hashing all operations are performed with few number of disk access.

→ In extendible hashing it requires two disk access only for find operation and very few number of access for insert and delete operations.

→ This extendible hashing can be done by using directorys and keys are inserted in binary format

→ Here the directory can be divided into 2^D sub directries are roots. For each root a leaf directory required to connect. This leaf can be called as disk.

→ The capacity of each leaf or disk is also 2^D. where D is the size of the directory.

→ When the insertion of binary format keys reaches it load factor of leaf then it requires to split the root into two directorys / roots by increasing its size by 1

→ After splitting the separate leafes are attached to the roots which requires the split operation and remaining all indicating with the same leafs of previous one.

Ex:- Insert the keys 11011O, 010101, 111010, 111011, 101100, 110011, 111101. into directory of size 2.

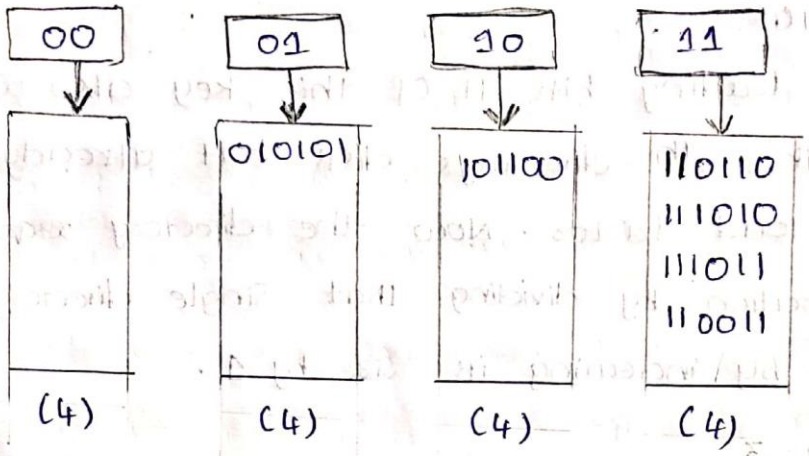
Given D = 2

So, divide the directory into 2^D subdirectorys / roots which are 00, 01, 10, 11

where $2^2 = 4$

The capacity of each leaf / disk is also $2^D = 2^2 = 4$

Now the extendible hashing divides the directory as



For the insertion of given keys into the directorys match appropriate directory / root with first two leading bits with keys. where the match will be found with the root insert the given key into its appropriate disk / leaf.

→ For key 110110:

The first two leading bits 11 of this key matches with the fourth directory

So, insert the key into its disk.

→ For key 010101 :

The first two leadings bits 01 of this key matches with the second directory. so, insert the key into its disk.

→ For key 111010 :

The first two leading bits 11 of this key matches with the fourth directory. so, insert the key into its disk.

→ For key 111011 :

The first two leading bits 11 of this key matches with the fourth directory. so, insert key into the disk.

→ For key 101100 :

The first two leading bits 10 of this key matches with the 3rd directory . so, insert the key into the disk

→ For key 110011 :

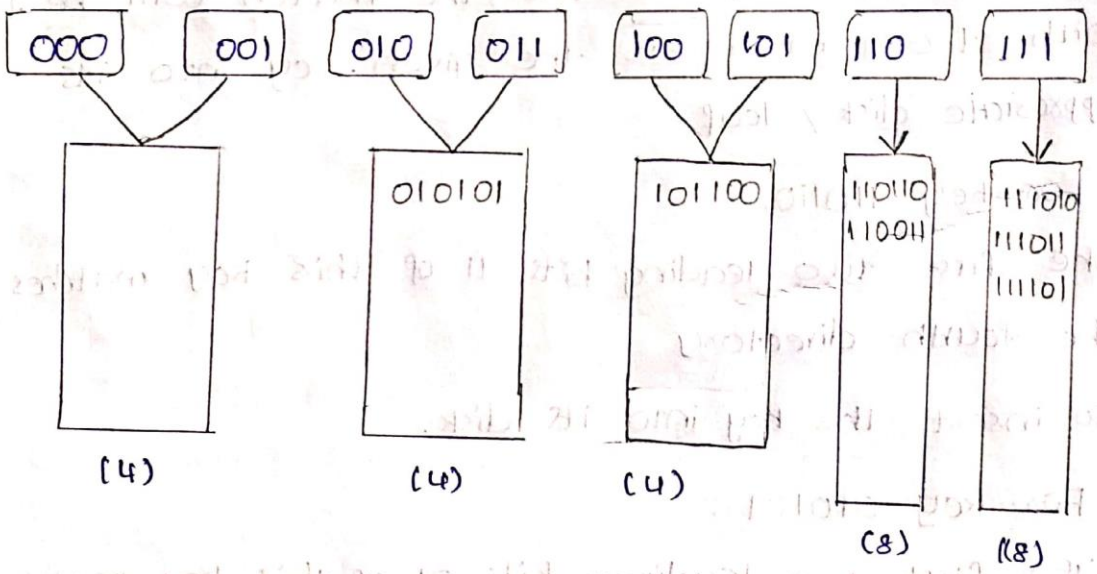
The first two leading bits 11 of this key matches with 4th directory . so, insert the key into the disk.

→ For key 111101 :

The first two leading bits 11 of this key also wants to occupy the 4th directorys disk but already it reaches its load factor . Now the directory requires the split operation by dividing that single directory into two directorys by increasing its size by 1.

Now $D = 3$

The directory can be divided into 2^D (8) subdirectorys / roots.



The capacity of directorys 00, 01, 10 is same of four elements and directory 11 increases its capacity from 4 to 8;

→ The given keys are remapped into the extended into extended directorys as shown in above table.

Dynamic Hashing: In Dynamic hashing the size of the hash table may grow or shrink during the performance of its operations.

Motivation of Dynamic Hashing:

To ensure good performance it requires to increase the size of the hash table where it reaches its load density of pre specified threshold

Example:

Consider the hash table of size 10 and pre specified threshold value is 4.

→ If insertion of the elements into the hash table reaches its pre specified threshold value 4 then increase the size of the hash table for good performance to insert remaining all elements without collision.

* The dynamic hashing can be done in two ways.

1. Directory less Dynamic Hashing
2. Dynamic Hashing with Directories.

1. Directory less Dynamic Hashing:

In Directory less Dynamic Hashing the hash table can be used to insert the given elements into its buckets.

→ Here the size of the hash table may grow when it reaches pre specified threshold (load factor) value (or) if any insertion fails (or) if table gets more than half full

Ex:- Rehashing.

2. Dynamic Hashing with Directories:

In this technique the directories will be used instead of using the hash tables.

→ The directory is a storage location where the data can be inserted.

→ Here the directory can be divided into 2^D roots and the keys are inserted into its seperately connected ~~leafs~~ leaves where the capacity of each leave is also 2^D.

Ex:- Extendible Hashing.