HASHING

Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.

Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.

The two types of hashing are separate chaining and open addressing

RE HASHING

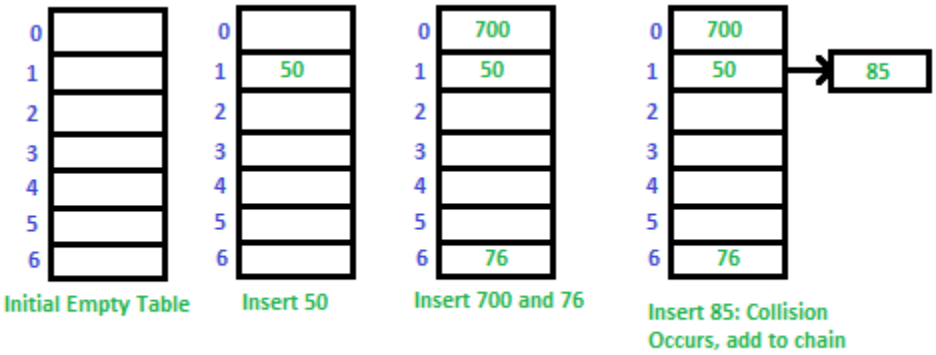
Rehashing is the process of recalculating the hash code of previously stored entries (key values pair) in order to shift them to larger size hash map when the threshold is reached or crossed.

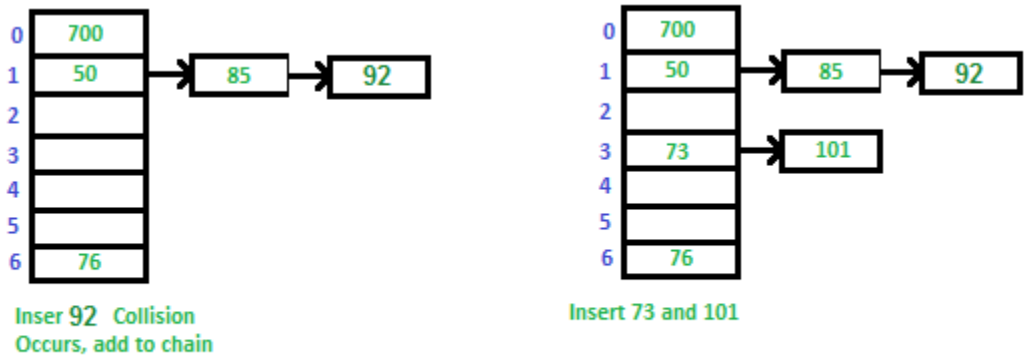
SEPARATE CHAINING

The idea behind separate chaining is to implement the array as a linked list called a chain. Separate chaining is one of the most popular and commonly used techniques in order to handle collisions.

The **linked list** data structure is used to implement this technique. So what happens is, when multiple elements are hashed into the same slot index, then these elements are inserted into a singly-linked list which is known as a chain.

Example: Let us consider a simple hash function as "**key mod 7**" and a sequence of keys as 50, 700, 76, 85, 92, 73, 101





OPEN ADDRESSING

It inserts the data into the hash table itself. The size of the hash table should be larger than the number of keys.

There are three types of open addressing. They are

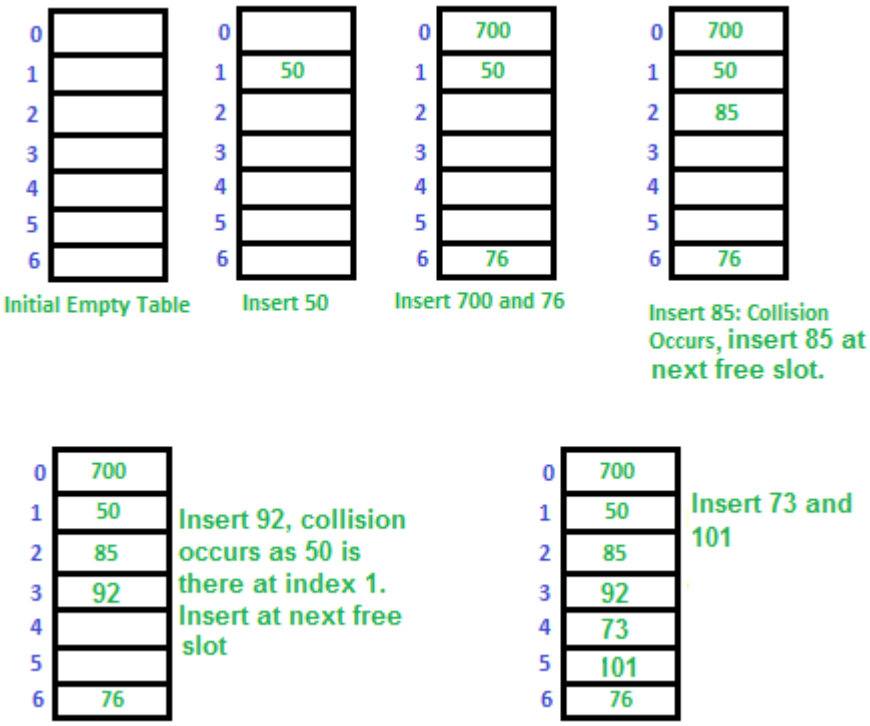
1. Linear probing
2. Quadratic probing
3. Double hashing

LINEAR PROBING

In this technique, if collision occurs, the element has to be inserted into the next free slot.

Let us consider a simple hash function as "key mod 7" and a sequence of keys as 50, 700, 76, 85, 92, 73, 101,

which means hash(key)= key% S, here S=size of the table =7,indexed from 0 to 6.



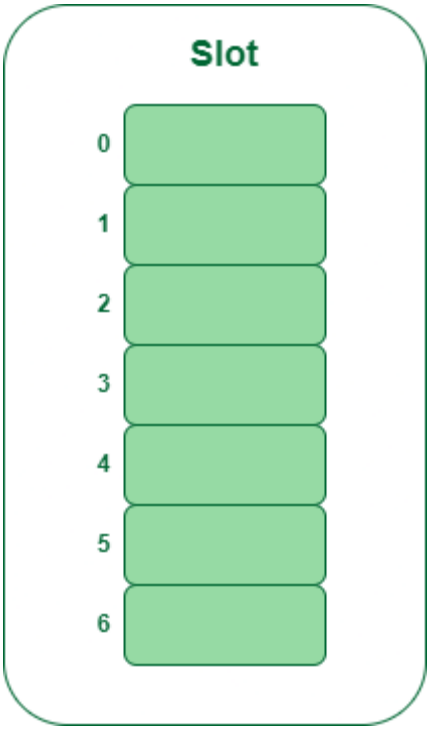
QUADRATIC PROBING

This method is also known as the **mid-square** method. In this method, we look for the **i^2'th** slot in the **i^{th}** iteration. We always start from the original hash location. If only the location is occupied then we check the other slots.
let hash(x) be the slot index computed using hash function.

*If slot hash(x) % S is full, then we try (hash(x) + 1*1) % S*
*If (hash(x) + 1*1) % S is also full, then we try (hash(x) + 2*2) % S*
*If (hash(x) + 2*2) % S is also full, then we try (hash(x) + 3*3) % S*

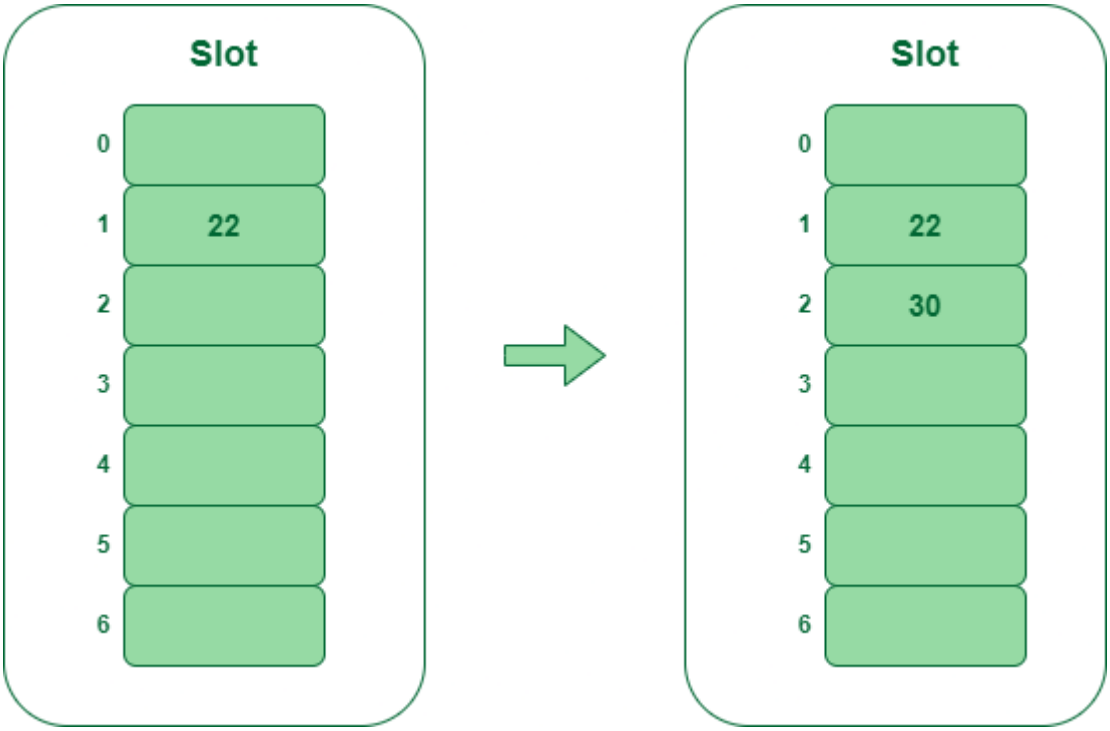
Example: Let us consider table Size = 7, hash function as Hash(x) = x % 7 and collision resolution strategy to be f(i) = i^2 . Insert = 22, 30, and 50.

- **Step 1:** Create a table of size 7.



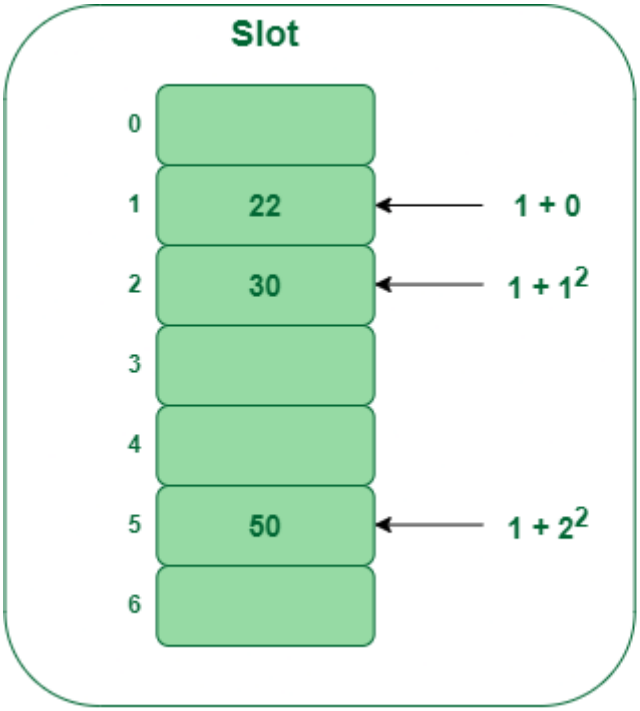
Hash table

- **Step 2** – Insert 22 and 30
    - Hash(22) = 22 % 7 = 1, Since the cell at index 1 is empty, we can easily insert 22 at slot 1.
    - Hash(30) = 30 % 7 = 2, Since the cell at index 2 is empty, we can easily insert 30 at slot 2.



Insert keys 22 and 30 in the hash table

- **Step 3:** Inserting 50
    - Hash(50) = 50 % 7 = 1
    - In our hash table slot 1 is already occupied. So, we will search for slot $1+1^2$, i.e. 1+1 = 2,
    - Again slot 2 is found occupied, so we will search for cell $1+2^2$, i.e.1+4 = 5,
    - Now, cell 5 is not occupied so we will place 50 in slot 5.



Insert key 50 in the hash table

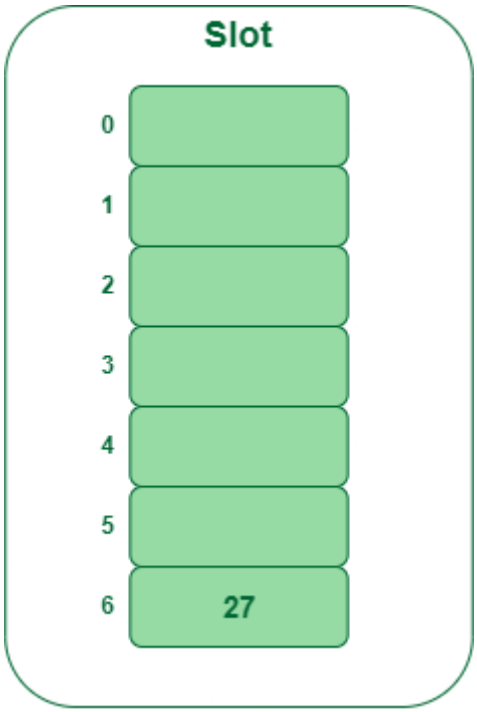
DOUBLE HASHING

In this technique, the increments for the probing sequence are computed by using another hash function. We use another hash function hash2(x) and look for the i*hash2(x) slot in the **i**th rotation.
let hash(x) be the slot index computed using hash function.

If slot hash(x) % S is full, then we try (hash(x) + 1*hash2(x)) % S
If (hash(x) + 1*hash2(x)) % S is also full, then we try (hash(x) + 2*hash2(x)) % S
If (hash(x) + 2*hash2(x)) % S is also full, then we try (hash(x) + 3*hash2(x)) % S

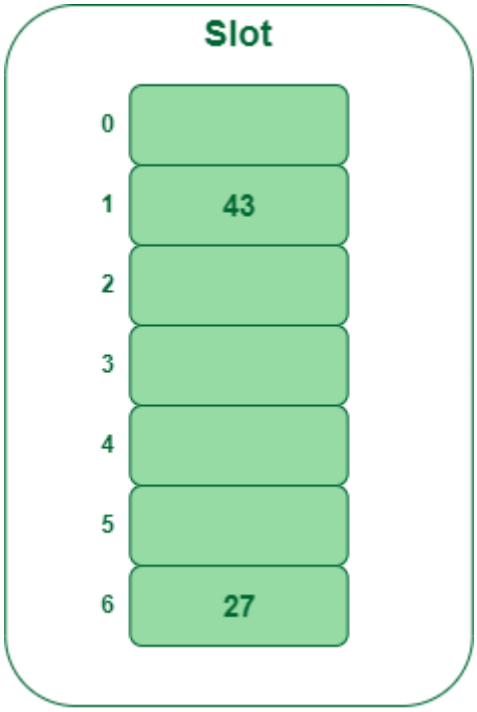
Example: Insert the keys 27, 43, 692, 72 into the Hash Table of size 7. where first hash-function is **h1(k) = k mod 7** and second hash-function is **h2(k) = 1 + (k mod 5)**

- **Step 1:** Insert 27
    - 27 % 7 = 6, location 6 is empty so insert 27 into 6 slot.



Insert key 27 in the hash table

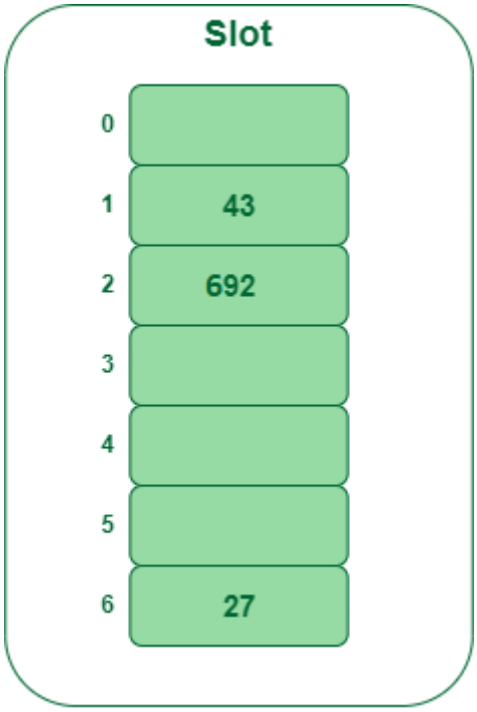
- **Step 2:** Insert 43
    - 43 % 7 = 1, location 1 is empty so insert 43 into 1 slot.



Insert key 43 in the hash table

- **Step 3:** Insert 692
    - 692 % 7 = 6, but location 6 is already being occupied and this is a collision
    - So we need to resolve this collision using double hashing.

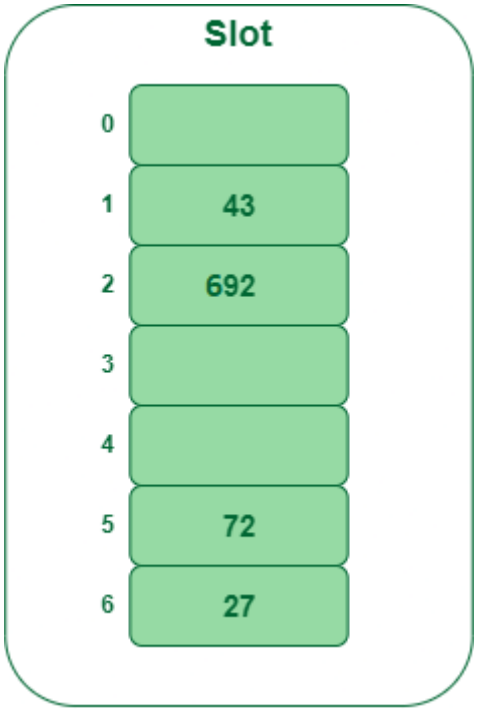
h_{new} = *[h1(692) + i * (h2(692)] % 7*
*= [6 + 1 * (1 + 692 % 5)] % 7*
= 9% 7
= 2
Now, as 2 is an empty slot,
so we can insert 692 into 2nd slot.



Insert key 692 in the hash table

- **Step 4:** Insert 72
    - 72 % 7 = 2, but location 2 is already being occupied and this is a collision.
    - So we need to resolve this collision using double hashing.

h_{new} = *[h1(72) + i * (h2(72)] % 7*
*= [2 + 1 * (1 + 72 % 5)] % 7*
= 5 % 7
= 5,
Now, as 5 is an empty slot,
so we can insert 72 into 5th slot.


Slot
0
1
43
2
692
3
4
5
72
6
27