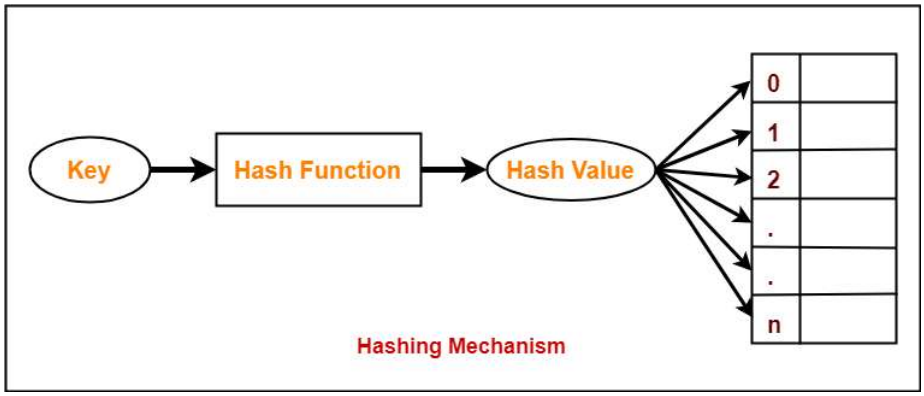
Hashing: -

Hashing is the process of converting a given key into another value. A **hash function** is used to generate the new value according to a mathematical algorithm. The result of a hash function is known as a **hash value** or simply, a **hash**.



Hashing can be used to build, search, or delete from a table.

The basic idea behind hashing is to take a field in a record, known as the **key**, and convert it through some fixed process to a numeric value, known as the **hash key**, which represents the position to either store or find an item in the table. The numeric value will be in the range of 0 to n-1, where n is the maximum number of slots (or **buckets**) in the table.

The fixed process to convert a key to a hash key is known as a **hash function**. This function will be used whenever access to the table is needed.

One common method of determining a hash key is the **division method** of hashing. The formula that will be used is:

hash key = key % number of slots in the table

Assume a table with 8 slots:

Hash key = key % table size

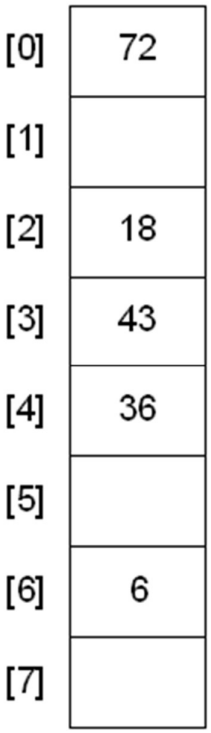
4 = 36 % 8

2 = 18 % 8

0 = 72 % 8

3 = 43 % 8

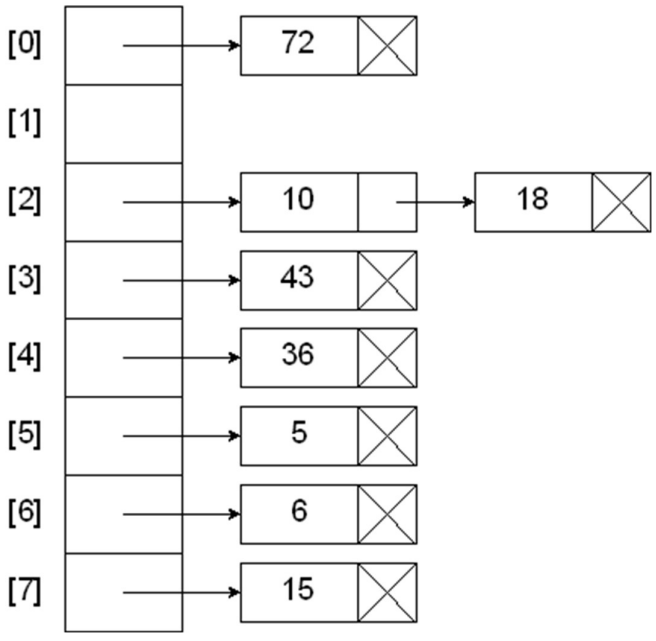
6 = 6 % 8



The problem with above is that there may me collisions. In that case, we can do something like below: -

Hash key = key % table size

4 = 36 % 8
2 = 18 % 8
0 = 72 % 8
3 = 43 % 8
6 = 6 % 8
2 = 10 % 8
5 = 5 % 8
7 = 15 % 8



The above is called as Open Addressing. The problem with above that say our numbers are pointing to the same bucket, then the linked list size increases and that the hash table has many empty buckets in it. We can overcome this by using closed addressing.

Hashing with Linear Probe

When using a linear probe, the item will be stored in the next available slot in the table, assuming that the table is not already full.

This is implemented via a linear search for an empty slot, from the point of collision. If the physical end of table is reached during the linear search, the search will wrap around to the beginning of the table and continue from there.

If an empty slot is not found before reaching the point of collision, the table is full.

[0]	72
[1]	
[2]	18
[3]	43
[4]	36
[5]	
[6]	6
[7]	

Add the keys 10, 5, and 15 to the previous table .

Hash key = key % table size

2 = 10 % 8

5 = 5 % 8

7 = 15 % 8

[0]	72
[1]	15
[2]	18
[3]	43
[4]	36
[5]	10
[6]	6
[7]	5

A problem with the linear probe method is that it is possible for blocks of data to form when collisions are resolved. This is known as **primary clustering**.

Hashing with Quadratic Probe

To resolve the primary clustering problem, **quadratic probing** can be used. With quadratic probing, rather than always moving one spot, move i^2 spots from the point of collision, where i is the number of attempts to resolve the collision.

89 % 10 = 9

18 % 10 = 8

49 % 10 = 9 – 1 attempt needed – 1^2 = 1 spot

58 % 10 = 8 – 3 attempts – 3^2 = 9 spots

69 % 10 = 9 – 2 attempts – 2^2 = 4 spots

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

Hashing with Double Hashing

Double hashing uses the idea of applying a second hash function to the key when a collision occurs. The result of the second hash function will be the number of positions form the point of collision to insert.

There are a couple of requirements for the second function:

- it must never evaluate to 0
- must make sure that all cells can be probed

A popular second hash function is: $Hash_2(key) = R - (key \% R)$ where R is a prime number that is smaller than the size of the table.

Table Size = 10 elements
$Hash_1(key) = key \% 10$
$Hash_2(key) = 7 - (k \% 7)$

Insert keys : 89, 18, 49, 58, 69

Hash(89) = 89 % 10 = 9

Hash(18) = 18 % 10 = 8

Hash(49) = 49 % 10 = 9 a collision !
= 7 – (49 % 7)
= 7 positions from [9]

Hash(58) = 58 % 10 = 8
= 7 – (58 % 7)
= 5 positions from [8]

Hash(69) = 69 % 10 = 9
= 7 – (69 % 7)
= 1 position from [9]

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