Hashing

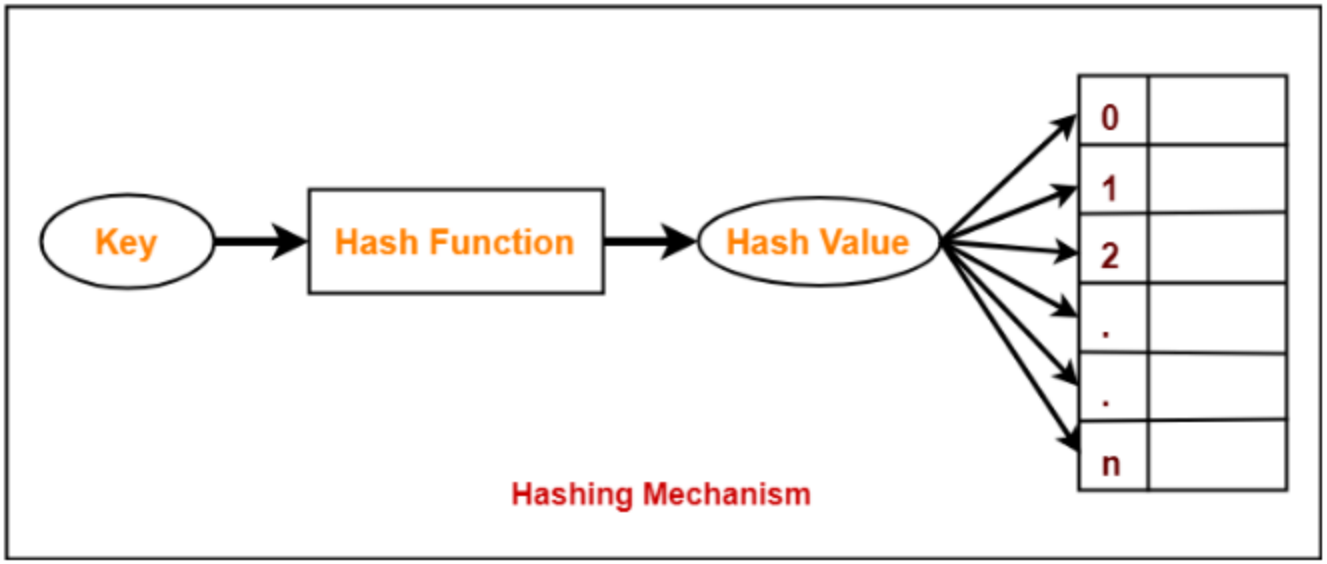
- Hashing is a well-known technique to search any particular element among several elements.
- It minimizes the number of comparisons while performing the search.
- Unlike other searching techniques,
  - Hashing is extremely efficient.
  - The time taken by it to perform the search does not depend upon the total number of elements.
  - It completes the search with constant time complexity O(1).

Hashing Mechanism

- An array data structure called as **Hash table** is used to store the data items.
- Based on the hash key value, data items are inserted into the hash table.

Hash Key value

- Hash key value is a special value that serves as an index for a data item.
- It indicates where the data item should be be stored in the hash table.
- Hash key value is generated using a hash function.



Hash Function

- Hash function is a function that maps any big number or string to a small integer value.
- Hash function takes the data item as an input and returns a small integer value as an output.
- The small integer value is called as a hash value.
- Hash value of the data item is then used as an index for storing it into the hash table.

Types of Hash Functions

- There are various types of hash functions available such as-
- Mid Square Hash Function
- Division Hash Function
- Folding Hash Function etc

- It depends on the user which hash function wants to use.

Properties of Hash Function

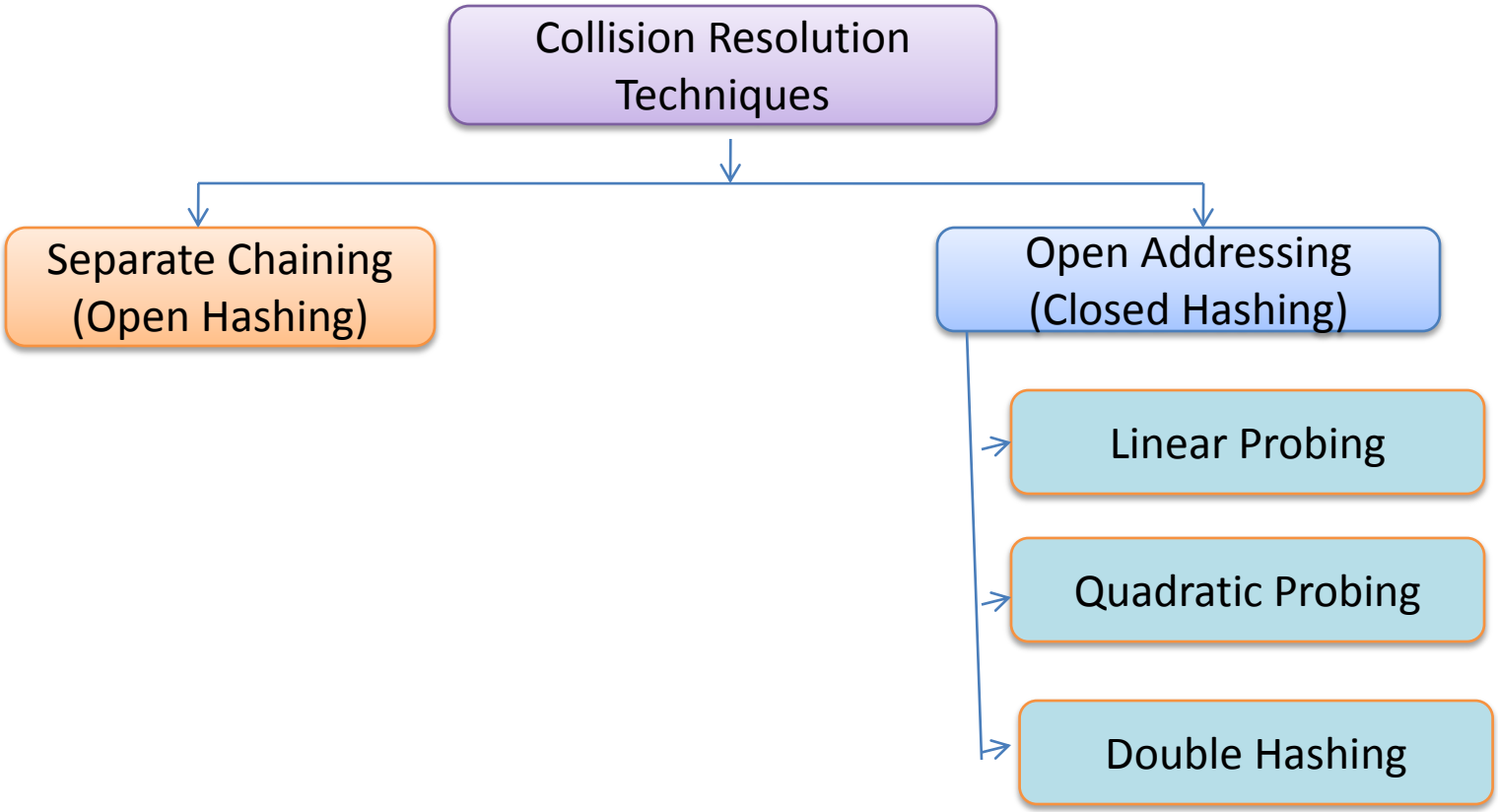
- The properties of a good hash function are-
  - It is efficiently computable.
  - It minimizes the number of collisions.
  - It distributes the keys uniformly over the table.

Collision in Hashing

- Hash function is used to compute the hash value for a key.
- Hash value is then used as an index to store the key in the hash table.
- Hash function may return the same hash value for two or more keys.
- When the hash value of a key maps to an already occupied bucket of the hash table, it is called as a **Collision**.

Collision Resolution Techniques

- Collision Resolution Techniques are the techniques used for resolving or handling the collision.



Separate Chaining

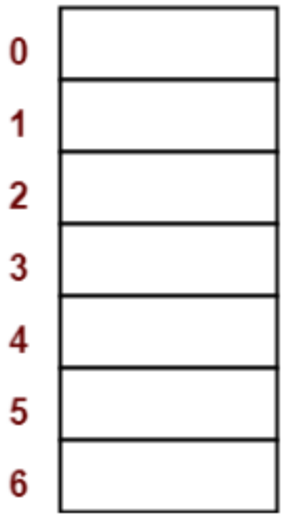
- To handle the collision,
  - This technique creates a linked list to the slot for which collision occurs.
  - The new key is then inserted in the linked list.
  - These linked lists to the slots appear like chains.
  - That is why, this technique is called as **separate chaining**.

Example-Separate Chaining

- Using the hash function ‘key mod 7’, insert the following sequence of keys in the hash table-
- 50, 700, 76, 85, 92, 73 and 101

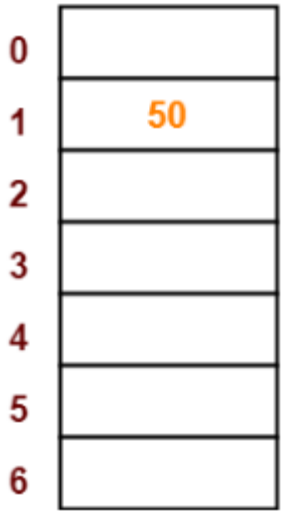
Step-1

- Draw an empty hash table.
- For the given hash function, the possible range of hash values is [0, 6].
- So, draw an empty hash table consisting of 7 buckets as-



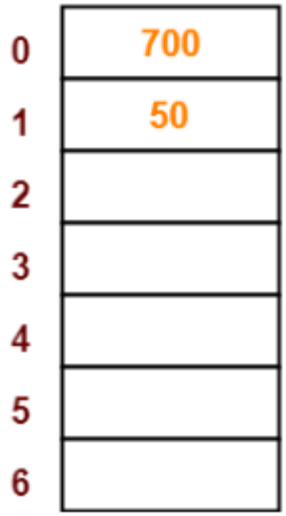
Step-2

- Insert the given keys in the hash table one by one.
- The first key to be inserted in the hash table = 50.
- Bucket of the hash table to which key 50 maps = 50 mod 7 = 1.
- So, key 50 will be inserted in bucket-1 of the hash table as-



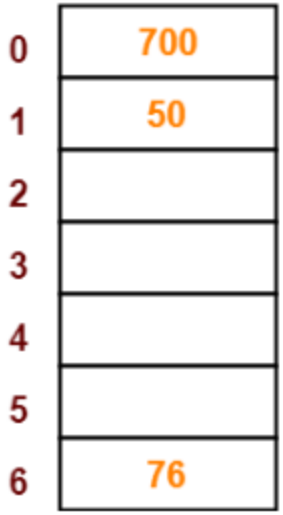
Step-3

- The next key to be inserted in the hash table = 700.
- Bucket of the hash table to which key 700 maps = 700 mod 7 = 0.
- So, key 700 will be inserted in bucket-0 of the hash table as-



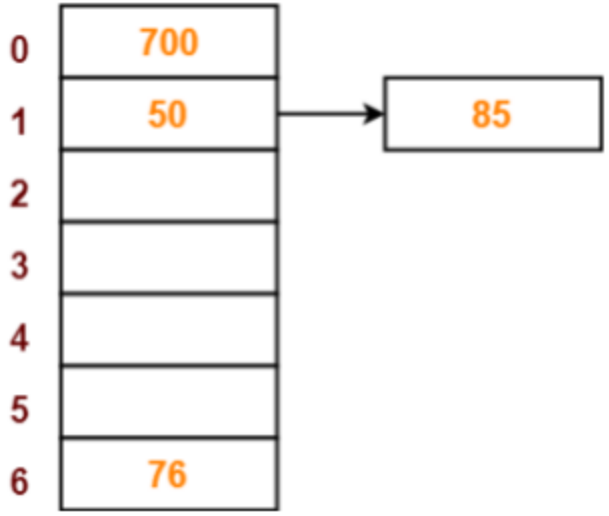
Step-4

- The next key to be inserted in the hash table = 76.
- Bucket of the hash table to which key 76 maps = 76 mod 7 = 6.
- So, key 76 will be inserted in bucket-6 of the hash table as-



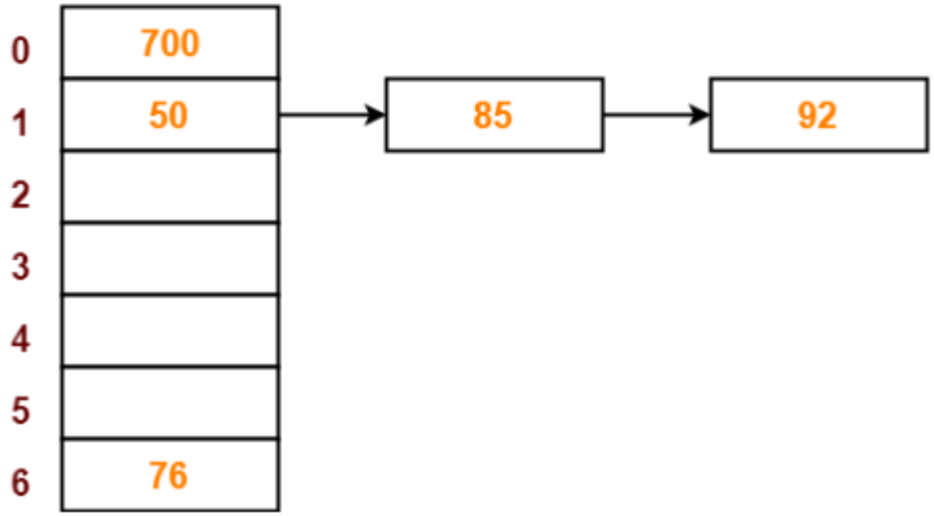
Step-5

- The next key to be inserted in the hash table = 85.
- Bucket of the hash table to which key 85 maps = 85 mod 7 = 1.
- Since bucket-1 is already occupied, so collision occurs.
- Separate chaining handles the collision by creating a linked list to bucket-1.
- So, key 85 will be inserted in bucket-1 of the hash table as-



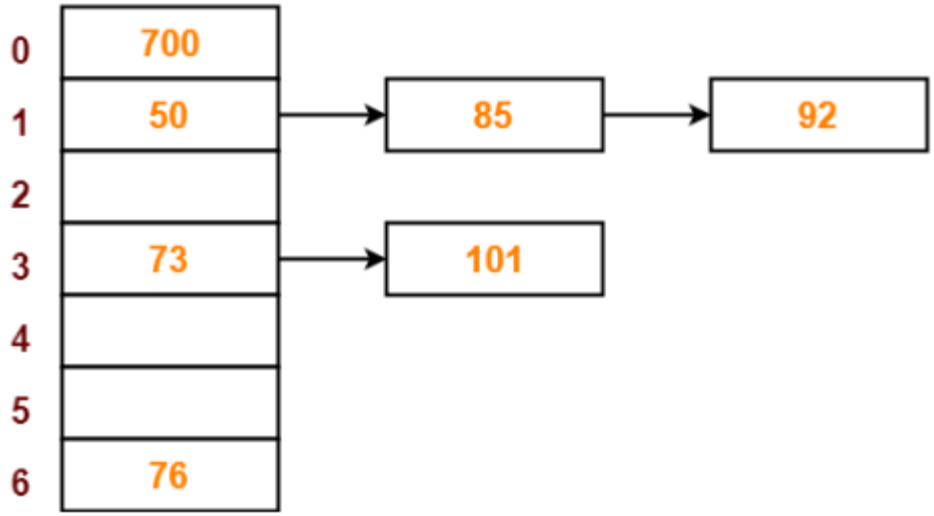
Step-6

- The next key to be inserted in the hash table = 92.
- Bucket of the hash table to which key 92 maps = 92 mod 7 = 1.
- Since bucket-1 is already occupied, so collision occurs.
- Separate chaining handles the collision by creating a linked list to bucket-1.
- So, key 92 will be inserted in bucket-1 of the hash table as-



Step-7

- The next key to be inserted in the hash table = 101.
- Bucket of the hash table to which key 101 maps = 101 mod 7 = 3.
- Since bucket-3 is already occupied, so collision occurs.
- Separate chaining handles the collision by creating a linked list to bucket-3.
- So, key 101 will be inserted in bucket-3 of the hash table as-



Open Addressing

- In open addressing,
  - Unlike separate chaining, all the keys are stored inside the hash table.
  - No key is stored outside the hash table.

- Techniques used for open addressing are-
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

Operations in Open Addressing

- Insert Operation:
  - Hash function is used to compute the hash value for a key to be inserted.
  - Hash value is then used as an index to store the key in the hash table.

- In case of collision,
  - Probing is performed until an empty bucket is found.
  - Once an empty bucket is found, the key is inserted.
  - Probing is performed in accordance with the technique used for open addressing.

Open Addressing

- Search Operation:
- To search any particular key,
  - Its hash value is obtained using the hash function used.
  - Using the hash value, that bucket of the hash table is checked.
  - If the required key is found, the key is searched.
  - Otherwise, the subsequent buckets are checked until the required key or an empty bucket is found.
  - The empty bucket indicates that the key is not present in the hash table.

Open Addressing

- Delete Operation:
  - The key is first searched and then deleted.
  - After deleting the key, that particular bucket is marked as “deleted”.

1. Linear Probing

- In linear probing,
  - When collision occurs, we linearly probe for the next bucket.
  - We keep probing until an empty bucket is found.

- **Advantage-**

  - It is easy to compute.

- **Disadvantage-**

  - The main problem with linear probing is clustering.
  - Many consecutive elements form groups.
  - Then, it takes time to search an element or to find an empty bucket.

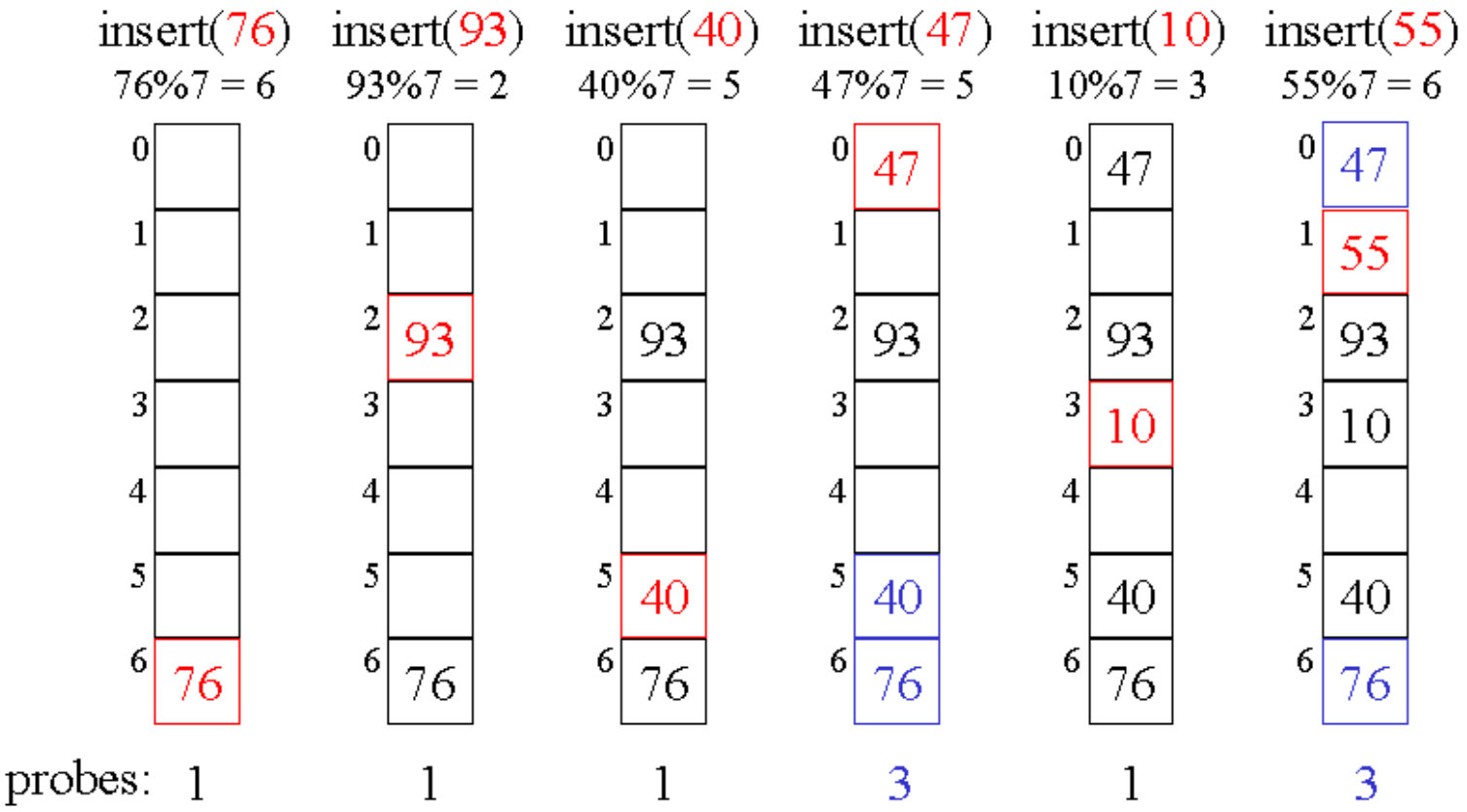
Linear Probing

$$f(i) = i$$

- Probe sequence is
  - h(k) mod size
  - h(k) + 1 mod size
  - h(k) + 2 mod size
  - …
- findEntry using linear probing:

```
bool findEntry(const Key & k, Entry *& entry) {
  int probePoint = hash1(k);
  do {
    entry = &table[probePoint];
    probePoint = (probePoint + 1) % size;
  } while (!entry->isEmpty() && entry->key != k);
  return !entry->isEmpty();
}
```

Example- Linear Probing



2. Quadratic Probing

- In quadratic probing,
- When collision occurs, we probe for i^2‘th bucket in i^{th} iteration.
- We keep probing until an empty bucket is found.

Quadratic Probing

$$f(i) = i^2$$

- Probe sequence is
  - h(k) mod size
  - (h(k) + 1) mod size
  - (h(k) + 4) mod size
  - (h(k) + 9) mod size
  - …
- findEntry using quadratic probing:

```
bool findEntry(const Key & k, Entry *& entry) {
  int probePoint = hash1(k), numProbes = 0;
  do {
    entry = &table[probePoint];
    numProbes++;
    probePoint = (probePoint + 2*numProbes - 1) % size;
  } while (!entry->isEmpty() && entry->key != key);
  return !entry->isEmpty();
}
```

Example – Quadratic Probing