Hashing

Introduction to Hashing:

Before we learn about hashing, first let's talk about mapping**.**

What is Mapping ?

Mapping is the way of linking 2 things together, referred to as “Key” and “Value”.

The **key** is like a label that is used to identify something unique. For example, in a dictionary, the word is the key that helps you find the definition.

The **value** is the information that is associated with the key. For example, in a dictionary, the definition is the value that is associated with the word.

Example of Key-Value pair:

A simple example of a key-value pair is a dictionary, where the word is the key and the definition of that word is the value. For example, consider the following key-value pairs:

Key: "Apple" → Value: "A round fruit with red, green, or yellow skin and a white flesh."

Key: "Banana" → Value: "A long curved fruit with a yellow skin and soft sweet flesh."

Here, the keys are the words "Apple" and "Banana" and the corresponding values are their definitions. These key-value pairs allow you to look up the definition of a word quickly and easily.

Example of Key-Value pair:

A simple example of a key-value pair is a student's record, where the student's name is the key and their corresponding grade is the value. For example, consider the following key-value pairs:

Key: "John" → Value: "B+"
Key: "Mary" → Value: "A-"

Here, the keys are the students' names, "John" and "Mary", and the corresponding values are their grades. These key-value pairs allow you to quickly look up a student's grade by using their name as the key.

Example of Key-Value pair:

A simple example of a key-value pair in the context of a restaurant is a menu item, where the dish name is the key and its price is the value. For example, consider the following key-value pairs:

Key: “Pizza” → Value: “400”

Key: “Maggi” → Value: “60”

Key: “Soup” → Value: “100”

Key: “Lassi” → Value: “80”

Key: “Burger” → Value: “40”

Key: “Cold Drink” → Value: “30”

Here, the keys are the names of the menu items, “Pizza”, “Maggi”, “Burger”, “Soup”, ”Lassi” and “Cold Drink”, and the corresponding values are their prices.

These key-value pairs allow the customer to easily find the prices of menu item by using its name as the key.

Now, let's talk about this Mapping in terms of Programing:

In the 3 example of mapping which is on the restaurant menu. Suppose you want to find the prices of any particular item, let’s say you want to see the price of a Burger on the menu.

So, you have to scan the entire menu to see the price of the burger i.e., you have to look one by one at the menu items until you find the Burger.

So, in programming, this type of searching is known as Linear Searching and Linear Search is having time complexity O(n).

Now if you ask the waiter the price of the burger, then the waiter will quickly tell you that the price of the burger is Rs 40. In Constant time waiter will tell you the prices of burger, because the waiter works in the restaurant, and he knows the price of every food item which is sold by the restaurant.

So, the mapping of food item with its price is present inside the brain of the waiter.

> So, just like waiter in above scenario, We need a data structure that will store the mapping of key-value pairs and supports the following operation in O(1) i.e., constant time. Operations are: Searching, Insertion, and Deletion.

Hash Table is the data structure which supports these operation (Searching, Insertion and Deletion) in an average case of Constant Time O(1).

Hash Table Data Structure:

1. A **hash table** is a data structure that allows you to store and retrieve key-value pairs quickly and efficiently.
2. Hash Table is a unordered data structure.
3. The keys in a Hash Table are all unique so that there will be a one-to-one relationship between a key and a value.
4. Hash Table Supports Insertion, Deletion, and searching of key-value pair in an average case of O(1) i.e., Constant time, but the worst case time complexity of these operation in Hash Table is O(n).

A hash table is a data structure that is typically implemented as an array of linked lists. Each element in the array corresponds to a bucket (that why the array in Hash Table is also called as **Bucket Array**), which contains a linked list of key-value pairs that have been hashed to that particular bucket.

When a new key-value pair is added to the hash table, the hash function is used to compute the index of the array where the key-value pair should be stored. If there are no other key-value pairs stored at that index, the new key-value pair is simply added to the linked list at that location. If there are other key-value pairs stored at that index, the linked list is searched to see if the key already exists in the table. If it does, the value associated with that key is updated. If the key is not found, a new key-value pair is added to the end of the linked list.

When searching for a particular key-value pair, the hash function is again used to compute the index of the array where the key-value pair should be stored. The linked list at that location is then searched to see if the key exists in the table. If it does, the corresponding value can be retrieved. If the key is not found in the linked list, then the key-value pair does not exist in the hash table.

Hash Function:

As we know that Hash Table is an array data structure and to store key-value pairs in an array, we treat array indices as keys and store the values to key at a particular index or key.

But the problem is, in the array we have indices and these indices are in integers. So what if the key is not an integer, so how will you store its value in the array ?

So, Hash Function is used to convert a key into an integer value and these hash function are written by programmers.

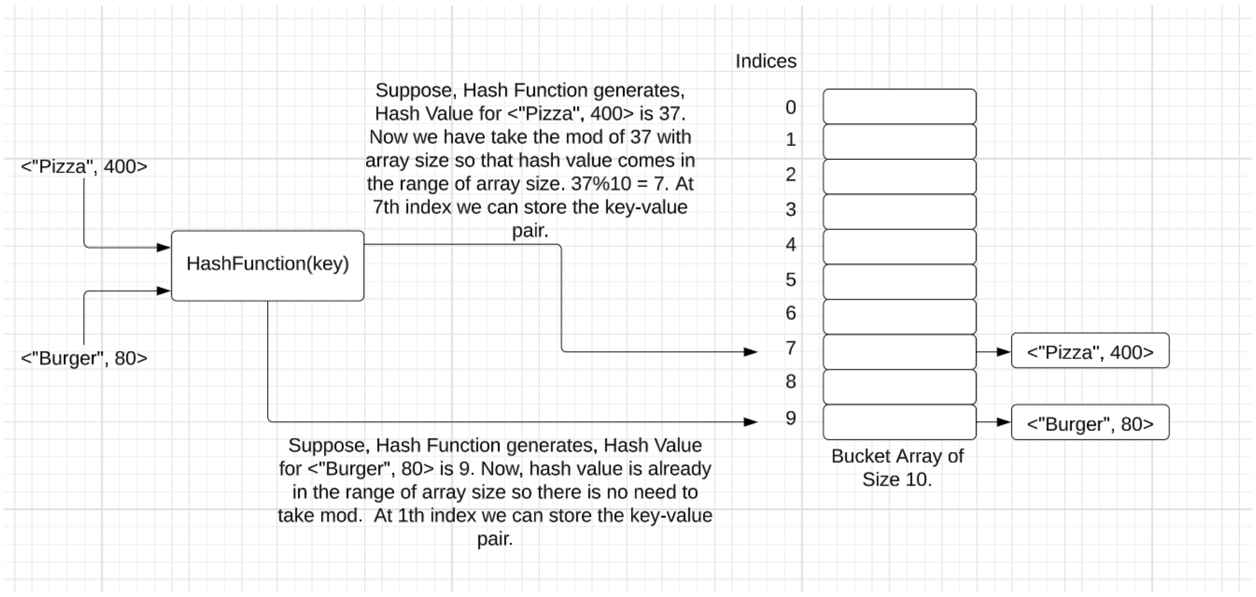
This is a responsibility of programmer to write a hash function of any type of key, because a key can be of type integer, string, character etc.

Hash Function if a combination of 2 things:

1. **Hash Code**
    a. To store any key-value pair in Bucket Array, your key must be in integer type because in the array we have indexes and these indexes are in the form of integers.
    b. And if the key is not an integer, we have to convert the key into an integer value and this task is done by Hash Code.
    c. Hash Code is a function that converts the key into an integer value, and after converting the key into an integer value, the Hash Code function returns this value (integer value) and this value (integer value) is called a **Hash Value.**
2. **Compression Function**
    a. Our Array is of particular size and it may possible that the hash value is not in the range of array size.
    b. Example: Suppose that our array size is 10, in 10 size array, indexes are from 0 to 9. Now suppose a hash function returns a hash value and that hash value is 16.

c. Compression function is used to compress the Hash Value so that the hash value will lie in the range of array indices.

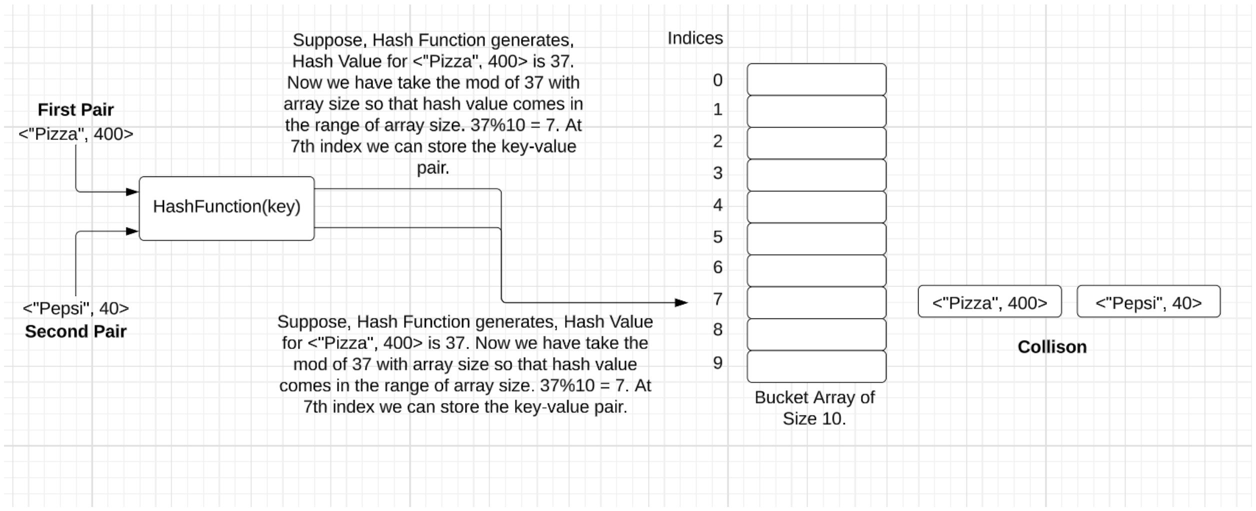
How Hash Function Works:



Indices
0
1
2
3
4
5
6
7
8
9
Suppose, Hash Function generates, Hash Value for <"Pizza", 400> is 37. Now we have take the mod of 37 with array size so that hash value comes in the range of array size. 37%10 = 7. At 7th index we can store the key-value pair.
<"Pizza", 400>
HashFunction(key)
<"Burger", 80>
<"Pizza", 400>
<"Burger", 80>
Bucket Array of Size 10.
Suppose, Hash Function generates, Hash Value for <"Burger", 80> is 9. Now, hash value is already in the range of array size so there is no need to take mod. At 1th index we can store the key-value pair.

What is Collision ?

In hashtable, a collision occurs when two or more keys have the same hash value and hence they are mapped to the same index in the hashtable. Collisions are a common problem in hashtables, especially as the number of items stored in the hashtable increases.


First Pair
<"Pizza", 400>
HashFunction(key)
<"Pepsi", 40>
Second Pair
Suppose, Hash Function generates, Hash Value for <"Pizza", 400> is 37. Now we have take the mod of 37 with array size so that hash value comes in the range of array size. 37%10 = 7. At 7th index we can store the key-value pair.
Suppose, Hash Function generates, Hash Value for <"Pizza", 400> is 37. Now we have take the mod of 37 with array size so that hash value comes in the range of array size. 37%10 = 7. At 7th index we can store the key-value pair.
Indices
0
1
2
3
4
5
6
7
8
9
Bucket Array of Size 10.
<"Pizza", 400>
<"Pepsi", 40>
Collison

Collision Handling:

There are different ways to handle collisions in a hashtable, including:

1. **Daisy Chaining or Closed Addressing.**
    a. Separate Chaining.
2. Open Addressing
    a. Linear Probing (Linear Search)
    b. Quadratic Probing (Non-Linear Search)
    c. Double Hashing (use multiple hash function).

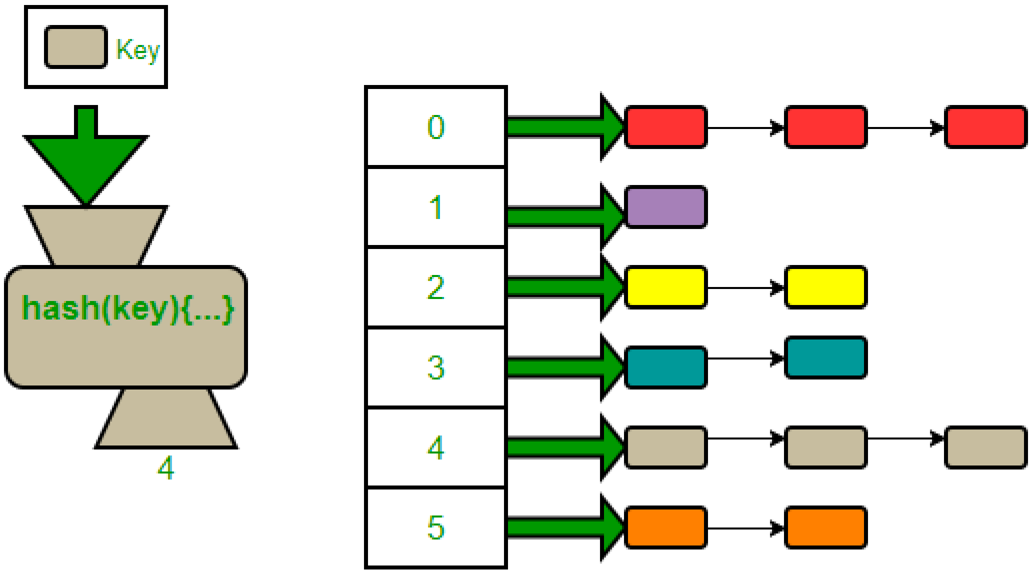
Daisy Chaining or Closed Addressing:

Daisy chaining, also known as closed addressing, is a technique used in hashtables to handle situations when two or more items have the same hash value, which is called a collision.

Daisy chaining or closed addressing is a collision resolution strategy used in hashtables. It is also known as separate chaining.

In daisy chaining, each bucket in the hash table is implemented as a linked list. So, basically in daisy chaining, our array is an array of linked list. When a collision occurs, the new item is inserted into the linked list at the corresponding bucket. This means that each bucket can contain multiple items, and each item stores a reference to the next item in the linked list.

When searching for an item in the hashtable, the hash function is used to determine the bucket where the item should be located. Then, the linked list at that bucket is traversed until the item with the matching key is found.



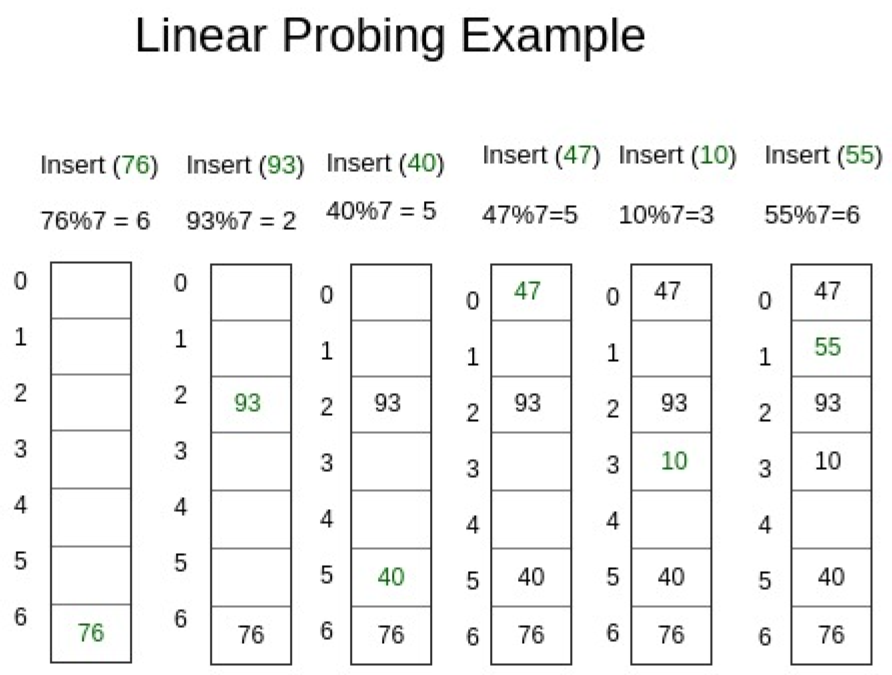
Open Addressing:

Open addressing is a collision resolution strategy used in hashtables. In open addressing, when a collision occurs (i.e., two or more keys have the same hash value), the hashtable looks for the next available slot and stores the item there.

The idea behind open addressing is to store all items in the hashtable itself, without using additional data structures like linked lists. This means that each bucket in the hashtable can store only one item.

Open addressing is a collision resolution technique used in hashtables, where the hashtable looks for the next available slot in the array when a collision occurs. Here are some common types of open addressing:

1. Linear Probing: In linear probing, when a collision occurs, the hashtable looks for the next available slot by checking the subsequent buckets in the array in a linear fashion until an empty bucket is found.



2. Quadratic Probing: In quadratic probing, the hashtable looks for the next available slot by checking the subsequent buckets in the array using a quadratic function. The quadratic function is of the form (i^2), where i is the number of times the hashtable has probed for the next available slot.

What is Hashing ?

Hashing is not a data structure, hashing is a technique used to quickly retrieve data or search for a specific data item from a large collection of data. The goal of hashing is to provide fast access to data by quickly finding the location of the data in memory.

1. Hashing uses Hash Table data structure to store key-value pairs.
2. And as we know that, Hash Table Supports Insertion, Deletion, and searching of key-value pair in an average case of O(1) i.e., Constant time, but the worst case time complexity of these operation in Hash Table is O(n).

Components of Hashing:

1. Hash Table
2. Hash Function
3. Collisions
4. Collision Resolution Techniques.

Load Factor:

Load factor is a concept used in hashtable data structures. It represents the ratio of the number of elements stored in the hashtable to the total number of buckets available in the hashtable. In simple words, it tells us how full the hashtable is.

```
Load Factor = number of elements in hash table / Size of Hash Table
```

For example, if we have a hashtable with 100 buckets and 70 elements are stored in it, then the load factor would be 0.7 (70/100). This means that the hashtable is 75% full.

The load factor is important because it affects the performance of the hashtable. When the load factor is high, it means the hashtable is becoming crowded, and the likelihood (likelihood means the chance of something happening) of collisions increases. Collisions occur when two or more elements map to the same bucket, which can cause slower search and insertion times.

Therefore, it's a good practice to keep the load factor below a certain threshold. Typically, a load factor of around 0.7 is considered a good balance between memory usage and performance. If the load factor exceeds the threshold, we can either increase the size of the hashtable or use a different collision resolution strategy to improve performance.

Rehashing:

Rehashing is the process of resizing a hashtable by creating a new, larger array and then re-inserting all the items from the old array into the new one. Rehashing is necessary when the load factor of the hashtable exceeds a certain threshold, indicating that the hashtable is becoming too full and may experience a significant number of collisions.

The steps involved in rehashing are as follows:

1. Create a new array with a larger size than the current array.
2. Iterate through all the items in the current array and re-insert them into the new array using the hashtable's hash function.
3. Update the hashtable's pointer to the new array, and free the memory allocated for the old array.

Rehashing can be an expensive operation, especially if the hashtable contains a large number of items. However, it is necessary to maintain the performance and efficiency of the hashtable by reducing the likelihood of collisions.