Introduction

Hashing is an important data structure designed to solve the problem of efficiently finding and storing data in an array. For example, if you have a list of 20000 numbers, and you have given a number to search in that list- you will scan each number in the list until you find a match.

It requires a significant amount of your time to search in the entire list and locate that specific number. This manual process of scanning is not only time-consuming but inefficient too. With hashing in the data structure, you can narrow down the search and find the number within seconds.

This blog will give you a deeper understanding of the hash method, hash tables, and linear probing with examples.

What is Hashing in Data Structure?

Hashing in the data structure is a technique of mapping a large chunk of data into small tables using a hashing function. It is also known as the message digest function. It is a technique that uniquely identifies a specific item from a collection of similar items.

It uses hash tables to store the data in an array format. Each value in the array has assigned a unique index number. Hash tables use a technique to generate these unique index numbers for each value stored in an array format. This technique is called the hash technique. You only need to find the index of the desired item, rather than finding the data. With indexing, you can quickly scan the entire list and retrieve the item you wish. Indexing also helps in inserting operations when you need to insert data at a specific location. No matter how big or small the table is, you can update and retrieve data within seconds.

Hashing in a data structure is a two-step process.

1. The hash function converts the item into a small integer or hash value. This integer is used as an index to store the original data.
2. It stores the data in a hash table. You can use a hash key to locate data quickly.

Examples of Hashing in Data Structure

The following are real-life examples of **hashing in the data structure** –

- In schools, the teacher assigns a unique roll number to each student. Later, the teacher uses that roll number to retrieve information about that student.
- A library has an infinite number of books. The librarian assigns a unique number to each book. This unique number helps in identifying the position of the books on the bookshelf.

Hash Function

The hash function in a data structure maps arbitrary size of data to fixed-sized data. It returns the following values: a small integer value (also known as hash value), hash codes, and hash sums.

hash = hashfunc(key)

index = hash % array_size

The hash function must satisfy the following requirements:

- A good hash function is easy to compute.
- A good hash function never gets stuck in clustering and distributes keys evenly across the hash table.
- A good hash function avoids collision when two elements or items get assigned to the same hash value.

Hash Table

Hashing in data structure uses hash tables to store the key-value pairs. The hash table then uses the hash function to generate an index. Hashing uses this unique index to perform insert, update, and search operations.

How does Hashing in Data Structure Works?

In hashing, the hashing function maps strings or numbers to a small integer value. Hash tables retrieve the item from the list using a hashing function. The objective of hashing technique is to distribute the data evenly across an array. Hashing assigns all the elements a unique key. The hash table uses this key to access the data in the list. Hash table stores the data in a key-value pair. The key acts as an input to the hashing function. Hashing function then generates a unique index number for each value stored. The index number keeps the value that corresponds to that key. The hash function returns a small integer value as an output. The output of the hashing function is called the hash value.

Let us understand **hashing in a data structure** with an example. Imagine you need to store some items (arranged in a key-value pair) inside a hash table with 30 cells.
The values are: (3,21) (1,72) (40,36) (5,30) (11,44) (15,33) (18,12) (16,80) (38,99)
The hash table will look like the following:

Serial Number	Key	Hash	Array Index
1	3	3%30 = 3	3
2	1	1%30 = 1	1
3	40	40%30 = 10	10
4	5	5%30 = 5	5
5	11	11%30 = 11	11
6	15	15%30 = 15	15
7	18	18%30 = 18	18
8	16	16%30 = 16	16
9	38	38%30 = 8	8

Collision Resolution Techniques

Hashing in data structure falls into a collision if two keys are assigned the same index number in the hash table. The collision creates a problem because each index in a hash table is supposed to store only one value. **Hashing in data structure** uses several collision resolution techniques to manage the performance of a hash table.

Linear Probing

Hashing in data structure results in an array index that is already occupied to store a value. In such a case, hashing performs a search operation and probes linearly for the next empty cell.

Linear Probing Example

Imagine you have been asked to store some items inside a hash table of size 30. The items are already sorted in a key-value pair format. The values given are: (3,21) (1,72) (63,36) (5,30) (11,44) (15,33) (18,12) (16,80) (46,99).

The hash(n) is the index computed using a hash function and T is the table size. If slot index = (hash(n) % T) is full, then we look for the next slot index by adding 1 ((hash(n) + 1) % T). If (hash(n) + 1) % T is also full, then we try (hash(n) + 2) % T. If (hash(n) + 2) % T is also full, then we try (hash(n) + 3) % T.
The hash table will look like the following:

Serial Number	Key	Hash	Array Index	Array Index after Linear Probing
1	3	3%30 = 3	3	3
2	1	1%30 = 1	1	1
3	63	63%30 = 3	3	4
4	5	5%30 = 5	5	5
5	11	11%30 = 11	11	11
6	15	15%30 = 15	15	15
7	18	18%30 = 18	18	18
8	16	16%30 = 16	16	16
9	46	46%30 = 8	16	17

Double Hashing

The double hashing technique uses two hash functions. The second hash function comes into use when the first function causes a collision. It provides an offset index to store the value.
The formula for the double hashing technique is as follows:
(firstHash(key) + i * secondHash(key)) % sizeOfTable
Where i is the offset value. This offset value keeps incremented until it finds an empty slot.
For example, you have two hash functions: h1 and h2. You must perform the following steps to find an empty slot:

1. Verify if hash1(key) is empty. If yes, then store the value on this slot.
2. If hash1(key) is not empty, then find another slot using hash2(key).
3. Verify if hash1(key) + hash2(key) is empty. If yes, then store the value on this slot.

4. Keep incrementing the counter and repeat with hash1(key)+2hash2(key), hash1(key)+3hash2(key), and so on, until it finds an empty slot.

Double Hashing Example

Imagine you need to store some items inside a hash table of size 20. The values given are: (16, 8, 63, 9, 27, 37, 48, 5, 69, 34, 1).

h1(n)=n%20
h2(n)=n%13
n h(n, i) = (h1 (n) + ih2(n)) mod 20

n	**h(n,i) = (h'(n) + i2) %20**
16	I = 0, h(n,0) = 16
8	I = 0, h(n,0) = 8
63	I = 0, h(n,0) = 3
9	I = 0, h(n,0) = 9
27	I = 0, h(n,0) = 7
37	I = 0, h(n,0) = 17
48	I = 0, h(n,0) = 8 I = 0, h(n,1) = 9 I = 0, h(n,2) = 12
5	I = 0, h(n,0) = 5
69	I = 0, h(n,0) = 9 I = 0, h(n,1) = 10
34	I = 0, h(n,0) = 14
1	I = 0, h(n,0) = 1