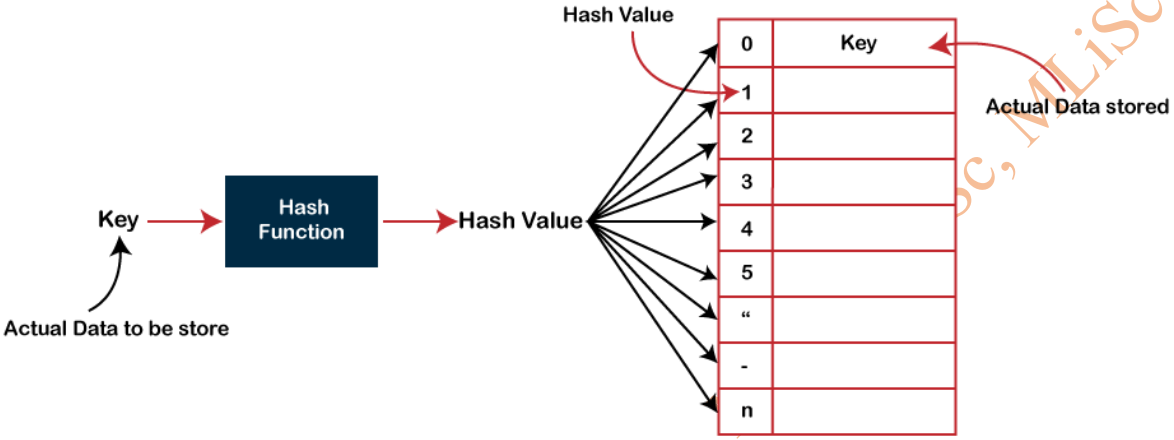
Hashing:

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.



Properties of the Hash Table:

- All the keys we add to the Hash Table must implement the Hash Code method. It provides a unique code to every key.
- Hash Table can be considered as an array of lists where each index of the list contains a bucket.
- We can add only the unique keys in the Hash Table.
- One cannot add null keys or values.
- The initial default capacity of the HashTable class is 11 whereas load factor is 0.75.

Public key:

- Public key often termed 'asymmetric' key, is a type of key only used for data encryption.
- The mechanism is relatively slower because the public key is an open key. The common uses of public keys include the functions of cryptography, the transfer of bitcoins, and securing online sessions.
- However, Public key functions along with a set of private keys, and thus, the overall security is not compromised.

Private key:

- The private key is employed in both encryption and decryption.
- Each party that sends or receives sensitive information that has been encrypted shares a key.
- Due to the fact that both parties share it, the private key is also referred to as “symmetric”.
- A private key is typically a long, impossible-to-guess string of bits generated at random or artificially random.

SSH public key:

- SSH uses both a public and a private key.
- SSH is a set of keys that can be used to authenticate and decrypt a communication sent from a distance.
- Both the distant servers and the stakeholders have access to the public key.

Types Of Hash Function:

There exist several types of hash function in data structures. We will focus on some of the most common hashing techniques present.

1. Division Hashing
2. Mid-Square Hashing
3. Digit Folding Method
4. Multiplication Hashing

Division Method:

The division method is the simplest and easiest method used to generate a hash value. In this hash function, the value of k is divided by M and uses the remainder as obtained.

Formula - h(K) = k mod M

(where k = key value and M = the size of the hash table)

Example -

k = 1320
M = 11
h (1320) = 1320 mod 11
= 0

Mid Square Method:

The steps involved in computing this hash method include the following

1. Squaring the value of k (like k*k)
2. Extract the hash value from the middle r digits.

Formula - h(K) = h(k x k)

(where k = key value)

Or

This method involves squaring the key and extracting some digits from the middle of the resulting value. This is followed by taking the modulo of that value with the number of buckets in the table. This method is used when the key is a non-negative integer.

Example:

Let's take the hash table with 200 memory locations and r = 2, as decided on the size of the mapping in the table.

k = 50

Therefore,

k = k x k

= 50 x 50

= 2500

Thus,

h(50) = 50

Digit Folding Method:

This method involves taking the key, breaking it down into groups of digits, and then summing the digits in each group. The resulting value is then taken modulo the number of buckets in the table. This method can be used when the key is an integer or a string.

Or

There are two steps in this method:

1. The key-value k should be divided into a specific number of parts, such as k1, k2, k3,..., kn, each having the very same number of digits aside from the final component, which may have fewer digits than the remaining parts.
2. Add each component separately. The last carry, if any, is disregarded to determine the hash value.

Formula:

k = k1, k2, k3, k4, ….., kn

s = k1+ k2 + k3 + k4 +….+ kn

h(K)= s

(Where, s = addition of the parts of key k)

Example:

k = 54321

k1 = 54; k2 = 32; k3 = 1

Therefore,

s = k1 + k2 + k3

= 54 + 32+ 1

= 87

Thus,

h (k) = 87

Multiplication Method:

This method involves multiplying the key by a constant value and then taking the fractional part of the result. The fractional part is multiplied by the number of buckets in the table, and the floor of that value is taken as the hash code.

Steps to follow:

1. Pick up a constant value A (where 0 < A < 1)
2. Multiply A with the key value
3. Take the fractional part of kA
4. Take the result of the previous step and multiply it by the size of the hash table, M.

Formula - h(K) = floor (M (kA mod 1))

(Where, M = size of the hash table, k = key value and A = constant value)

Example:

k = 1234
A = 0.35784
M = 100
So,
h (1234) = floor [100(1234 x 0.35784 mod 1)]
= floor [100 (441.57456 mod 1)]
= floor [100 (0. 57456)]
= floor [57.456]
= 57
Thus,
h (1234) = 57

Static and Dynamic Hashing.

What is static and dynamic hashing in data structure?

Static hashing is a hashing technique that allows users to perform lookups on a finalized dictionary set (all objects in the dictionary are final and not changing). In contrast, dynamic hashing is a hashing technique in which the data buckets are added and removed dynamically and on demand.

What is the basic difference between static hashing and dynamic hashing?

- The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks.
- Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing is also known as extended hashing.

Static Hashing:

It is a hashing technique that enables users to lookup a definite data set. Meaning, the data in the directory is not changing, it is "Static" or fixed. In this hashing technique, the resulting number of data buckets in memory remains constant.

Operations Provided by Static Hashing:

Static hashing provides the following operations:

Delete:

Search a record address and delete a record at the same address or delete a chunk of records from records for that address in memory.

Insertion:

While entering a new record using static hashing, the hash function (h) calculates bucket address "h(K)" for the search key (k), where the record is going to be stored.

Search:

A record can be obtained using a hash function by locating the address of the bucket where the data is stored.

Update:

It supports updating a record once it is traced in the data bucket.

Advantages:

Static hashing is advantageous in the following ways:

- Offers unparalleled performance for small-size databases.
- Allows Primary Key value to be used as a Hash Key.

Disadvantages:

Static hashing comes with the following disadvantages:

- It cannot work efficiently with the databases that can be scaled.
- It is not a good option for large-size databases.
- Bucket overflow issue occurs if there is more data and less memory.

Dynamic Hashing:

It is a hashing technique that enables users to lookup a dynamic data set. Means, the data set is modified by adding data to or removing the data from, on demand hence the name 'Dynamic' hashing. Thus, the resulting data bucket keeps increasing or decreasing depending on the number of records.

In this hashing technique, the resulting number of data buckets in memory is ever-changing.

Operations:

Dynamic hashing provides the following operations:

Delete:

Locate the desired location and support deleting data (or a chunk of data) at that location.

Insertion:

Support inserting new data into the data bucket if there is a space available in the data bucket.

Query:

Perform querying to compute the bucket address.

Update:

Perform a query to update the data.

Advantages:

Dynamic hashing is advantageous in the following ways:

- It works well with scalable data.
- It can handle addressing large amount of memory in which data size is always changing.
- Bucket overflow issue comes rarely or very late.

Disadvantages:

Dynamic hashing comes with the following disadvantage:

- The location of the data in memory keeps changing according to the bucket size.
- Hence if there is a phenomenal increase in data, then maintaining the bucket address table becomes a challenge.

Differences between Static and Dynamic Hashing:

Key Factor	Static Hashing	Dynamic Hashing
Form of Data	Fixed-size, non-changing data.	Variable-size, changing data.
Result	The resulting Data Bucket is of fixed-length.	The resulting Data Bucket is of variable-length.
Bucket Overflow	Challenge of Bucket overflow can arise often depending upon memory size.	Bucket overflow can occur very late or doesn't occur at all.
Complexity	Simple	Complex

What is a Hash Collision?

A hash collision happens when the same hash value is produced for two different input values by a hash algorithm. But it's important to point out that collisions aren't a problem; they're a fundamental aspect of hashing algorithms.

Collisions occur because different hashing techniques in data structure convert every input into a fixed-length code, regardless of its length. Since there are an endless number of inputs and a limited number of outputs, the hashing algorithms will eventually produce repeating hashes.

Types of Hashing:

There are two primary hashing techniques in a data structure.

- Open hashing / separate chaining / closed addressing
- Closed hashing / open addressing

1. Open Hashing / Separate Chaining:

- Separate chaining is the most used collision hashing technique in data structures that uses a lined list.
- Any two or more components that meet at the same point are chained together to form a single-linked list known as a chain.
- Every linked list member that hashes is chained to the same position here.
- Also known as closed addressing, open hashing is used to avoid any hash collisions, using an array of linked lists in order to resolve the collision.

Example:

- In a simple hash function, h(key) = key mod table size.
- Let us take a simple hash table with table size = 6
- Sequence keys = 50, 600, 74, 113, 14, 99

2. Closed hashing (Open addressing):

- Open addressing stores all entry records within the array itself, as opposed to linked lists.
- The phrase 'open addressing' refers to the notion that the hash value of an item does not identify its location or address.
- In order to insert a new entry, the array is first checked before computing the hash index of the hashed value, starting with the hashed index.
- If the space at the hashed index is empty, the entry value is inserted there; otherwise, some probing sequences are used until an empty slot is found.

In Closed hashing, there are three techniques that are used to resolve the collision:

1. **Linear Probing**
2. **Quadratic Probing**
3. **Double-Hashing**

Linear Probing:

- Linear probing involves systematically checking the hash table from its very beginning.
- A different site is searched if the one received is already occupied. In linear probing,
- the interval between the probes is usually fixed (generally, to a value of 1).

- The formula for linear probing: **index = key % hashTableSize**
- 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 the next slot index is calculated by adding 1 ((hash(n) + 1) % T).

The sequence goes as:

index = (hash(n) % T)

(hash(n) + 1) % T

(hash(n) + 2) % T

(hash(n) + 3) % T … and so on.

Example:

For a hash table, Table Size = 20

Keys = 3,2,46,6,11,13,53,12,70,90

SL. NO	KEY	HASH	INDEX	INDEX (AFTER LINEAR PROBING)
1	3	3%20	3	3
2	2	2%20	2	2
3	46	46%20	6	6
4	6	6%20	6	7
5	11	11%20	11	11
6	13	13%20	13	13
7	53	53%20	13	14

SL. NO	KEY	HASH	INDEX	INDEX (AFTER LINEAR PROBING)
8	12	12%20	12	12
9	70	70%20	10	10
10	90	90%20	10	11

Quadratic Probing:

- The only distinction between linear and quadratic probing is the space between succeeding probes or entry slots.
- When a hashed index slot for an entry record is already taken, you must start traversing until you discover an open slot.
- The spacing between slots is calculated by adding each subsequent value of any arbitrary polynomial in the initial hashed index.

The formula for quadratic probing: **index = index % hashTableSize**

- 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 the next slot index is calculated by adding 1 (hash(n) + 1 x 1) % T

The sequence goes as -

- index = (hash(n) % T)
- (hash(n) + 1 x 1) % T
- (hash(n) + 2 x 2) % T
- (hash(n) + 3 x 3) % T … and so on

Example:

- For a hash table, Table Size = 7
- Keys = 22,30,50
-

SL. NO	KEY	HASH	INDEX	INDEX (AFTER QUADRATIC PROBING)
1	22	22%7	1	1
2	30	30%7	2	2
3				
4				
5	50	50%7	1	1(1+2 x 2)
6				

Double-Hashing:

- It is another hash function that determines the intervals between probes.
- An optimized method of reducing clustering is double hashing.
- An additional hash function is used to calculate the increments for the probing sequence.

The formula for double hashing - **(first hash(key) + i * secondHash(key)) % size of the table**

The sequence goes as follows -

- index = hash(x) % S
- (hash(x) + 1*hash2(x)) % S
- (hash(x) + 2*hash2(x)) % S
- (hash(x) + 3*hash2(x)) % S … an so on

Example:

- For a hash table, Table Size = 7
- Keys = 27,43,98,72

SL. NO	KEY	HASH	INDEX	INDEX (AFTER DOUBLE HASHING)
1	43	43%7	1	
2	92	92%7	6	[h1(92) + i * (h2(92)] % 7 = [6 + 1 * (1 + 92 % 5)] % 7 = 9 % 7 = 2
3				
4				
5	72	72%7	2	[h1(72) + i * (h2(72)] % 7 = [2 + 1 * (1 + 72 % 5)] % 7 = 5 % 7 = 5
6	27	27%7	6