DATA STRUCTURES AND ALGORITHMS

DR SAMABIA TEHSIN

BS (AI)

HASHING

***Hashing** refers to the process of generating a fixed-size output from an input of variable size using the mathematical formulas known as hash functions. This technique determines an index or location for the storage of an item in a data structure.*

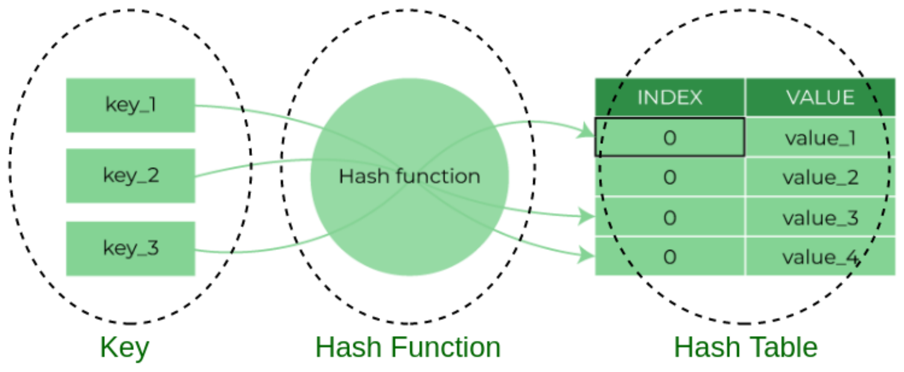
Components of Hashing

There are majorly three components of hashing:

1. **Key:** A **Key** can be anything string or integer which is fed as input in the hash function the technique that determines an index or location for storage of an item in a data structure.

2. **Hash Function:** The **hash function** receives the input key and returns the index of an element in an array called a hash table. The index is known as the **hash index**.

3. **Hash Table:** Hash table is a data structure that maps keys to values using a special function called a hash function. Hash stores the data in an associative manner in an array where each data value has its own unique index.

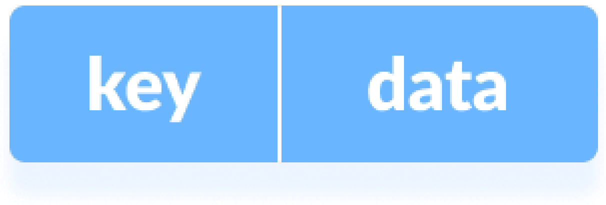


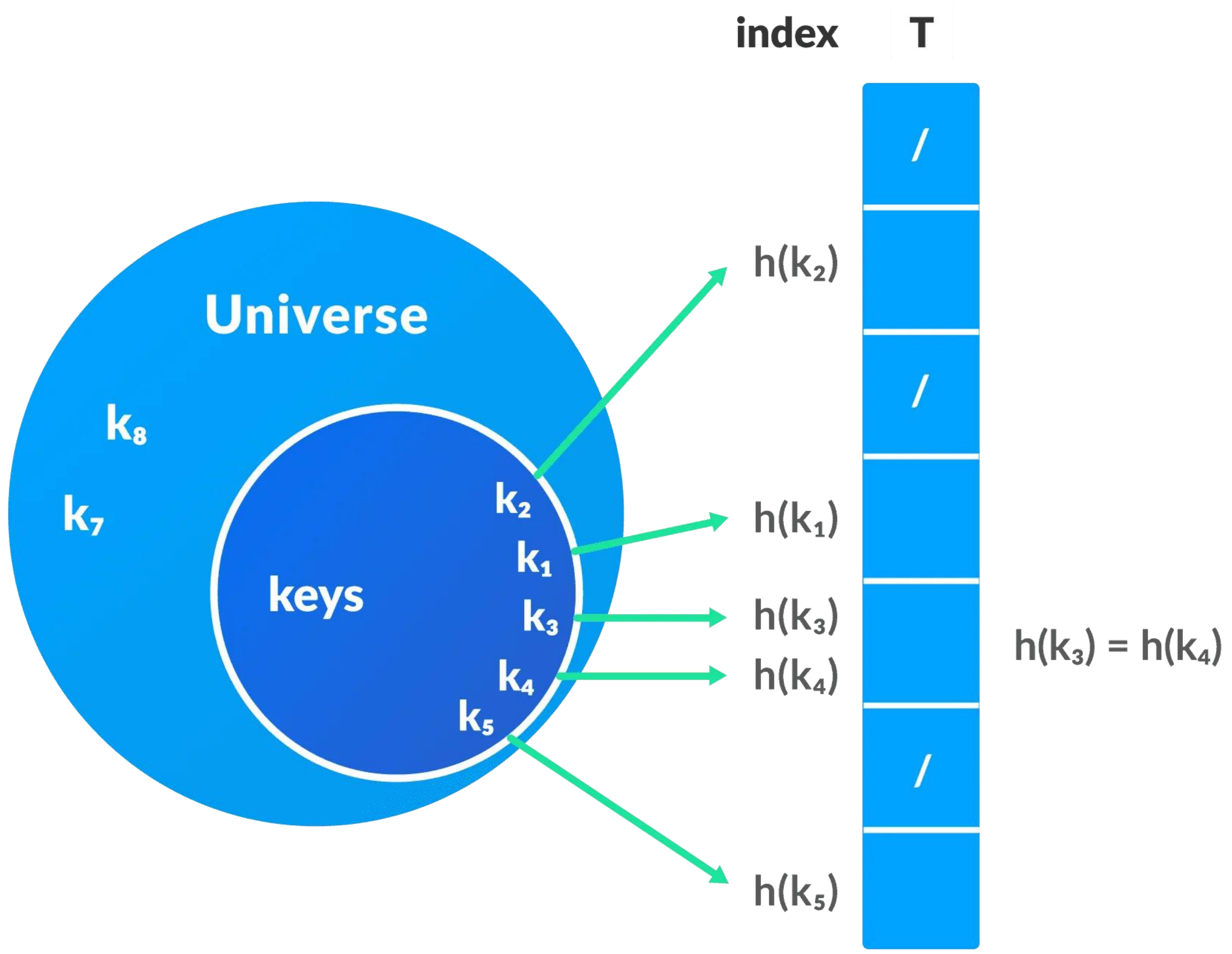
Components of Hashing

Hash Table

The Hash table data structure stores elements in key-value pairs where

- **Key**- unique integer that is used for indexing the values
- **Value** - data that are associated with keys.





Hashing (Hash Function)

In a hash table, a new index is processed using the keys. And, the element corresponding to that key is stored in the index. This process is called **hashing**.

Let k be a key and h(x) be a hash function.

Here, h(k) will give us a new index to store the element linked with k.

What is a Hash function?

The hash function creates a mapping between key and value, this is done through the use of mathematical formulas known as hash functions. The result of the hash function is referred to as a hash value or hash. The hash value is a representation of the original string of characters but usually smaller than the original.

Types of Hash functions:

There are many hash functions that use numeric or alphanumeric keys. This article focuses on discussing different hash functions:

1. Division Method.
2. Mid Square Method.
3. Folding Method.
4. Multiplication Method.

Division Method:

This is the most simple and easiest method to generate a hash value. The hash function divides the value k by M and then uses the remainder obtained.

Formula:

h(K) = k mod M

Here,
k is the key value, and
M is the size of the hash table.

It is best suited that M is a prime number as that can make sure the keys are more uniformly distributed. The hash function is dependent upon the remainder of a division.

Division Method

Example:

k = 12345

M = 95

h(12345) = 12345 mod 95

= 90

k = 1276

M = 11

h(1276) = 1276 mod 11

= 0

Division Method

Pros:

1. This method is quite good for any value of M.
2. The division method is very fast since it requires only a single division operation.

Cons:

1. This method leads to poor performance since consecutive keys map to consecutive hash values in the hash table.
2. Sometimes extra care should be taken to choose the value of M.

Mid Square Method

The mid-square method is a very good hashing method. It involves two steps to compute the hash value-

1.Square the value of the key k i.e. k^2

2.Extract the middle **r** digits as the hash value.

Formula:

h(K) = h(k x k)

Here,

k is the key value.

The value of **r** can be decided based on the size of the table.

Mid Square Method

Example:

Suppose the hash table has 100 memory locations. So r = 2 because two digits are required to map the key to the memory location.

k = 60
k x k = 60 x 60
        = 3600
h(60) = 60
The hash value obtained is 60

Mid Square Method

Pros:

1. The performance of this method is good as most or all digits of the key value contribute to the result. This is because all digits in the key contribute to generating the middle digits of the squared result.
2. The result is not dominated by the distribution of the top digit or bottom digit of the original key value.

Cons:

1. The size of the key is one of the limitations of this method, as the key is of big size then its square will double the number of digits.
2. Another disadvantage is that there will be collisions but we can try to reduce collisions.

Digit Folding Method

This method involves two steps:

1.Divide the key-value **k** into a number of parts i.e. **k1, k2, k3,….,kn**, where each part has the same number of digits except for the last part that can have lesser digits than the other parts.

2.Add the individual parts. The hash value is obtained by ignoring the last carry if any.

Formula:

k = k1, k2, k3, k4, ….., kn

s = k1+ k2 + k3 + k4 +….+ kn

h(K)= s

Here,

s is obtained by adding the parts of the key **k**

Digit Folding Method

Example:

k = 12345

k1 = 12, k2 = 34, k3 = 5

s = k1 + k2 + k3

  = 12 + 34 + 5

  = 51

h(K) = 51

Digit Folding Method

Note:
The number of digits in each part varies depending upon the size of the hash table. Suppose for example the size of the hash table is 100, then each part must have two digits except for the last part which can have a lesser number of digits.

Multiplication Method

This method involves the following steps:

1. Choose a constant value A such that $0 < A < 1$.
2. Multiply the key value with A.
3. Extract the fractional part of kA.
4. Multiply the result of the above step by the size of the hash table i.e. M.
5. The resulting hash value is obtained by taking the floor of the result obtained in step 4.

Multiplication Method

Formula:

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

Here,

M is the size of the hash table.

k is the key value.

A is a constant value.

Example:

k = 12345

A = 0.357840

M = 100

h(12345) = floor[100 (12345*0.357840 mod 1)]

= floor[100 (4417.5348 mod 1)]

= floor[100 (0.5348)]

= floor[53.48]

= 53

Multiplication Method

Pros:

The advantage of the multiplication method is that it can work with any value between 0 and 1, although there are some values that tend to give better results than the rest.

Cons:

The multiplication method is generally suitable when the table size is the power of two, then the whole process of computing the index by the key using multiplication hashing is very fast.

Properties of a Good hash function

- ❑A hash function that maps every item into its own unique slot is known as a perfect hash function. We can construct a perfect hash function if we know the items and the collection will never change but the problem is that there is no systematic way to construct a perfect hash function given an arbitrary collection of items. Fortunately, we will still gain performance efficiency even if the hash function isn’t perfect.
- ❑A good hash function should have the following properties:
  - ❑Efficiently computable.
  - ❑ Should uniformly distribute the keys (Each table position is equally likely for each.
  - ❑Should minimize collisions.

How does Hashing work?

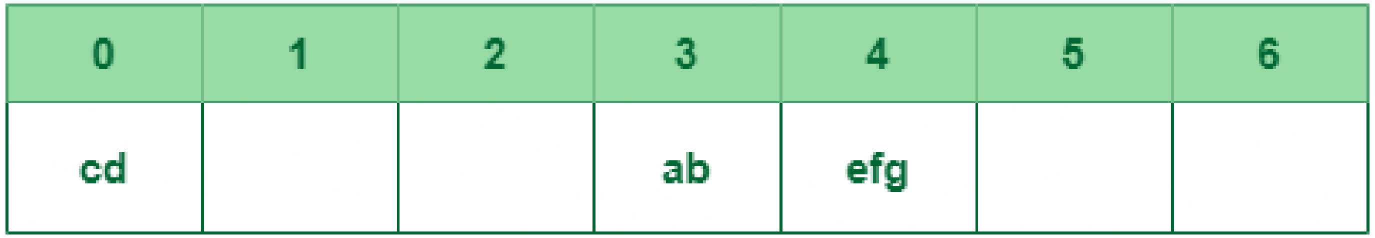
- **Step 1:** We know that hash functions (which is some mathematical formula) are used to calculate the hash value which acts as the index of the data structure where the value will be stored.

- **Step 2:** So, let’s assign
    - “a” = 1,
    - “b”=2, .. etc, to all alphabetical characters.

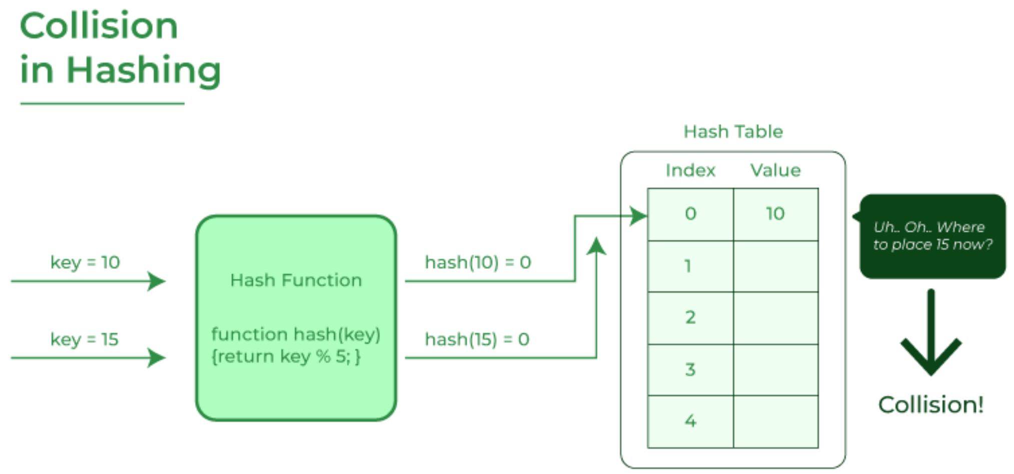
- **Step 3:** Therefore, the numerical value by summation of all characters of the string:
- “ab” = 1 + 2 = 3,
- “cd” = 3 + 4 = 7 ,
- “efg” = 5 + 6 + 7 = 18

How does Hashing work?

- **Step 4:** Now, assume that we have a table of size 7 to store these strings. The hash function that is used here is the sum of the characters in **key mod Table size**. We can compute the location of the string in the array by taking the **sum(string) mod 7**.
- **Step 5:** So we will then store
  - "ab" in 3 mod 7 = 3,
  - "cd" in 7 mod 7 = 0, and
  - "efg" in 18 mod 7 = 4.



0	1	2	3	4	5	6
cd			ab	efg		



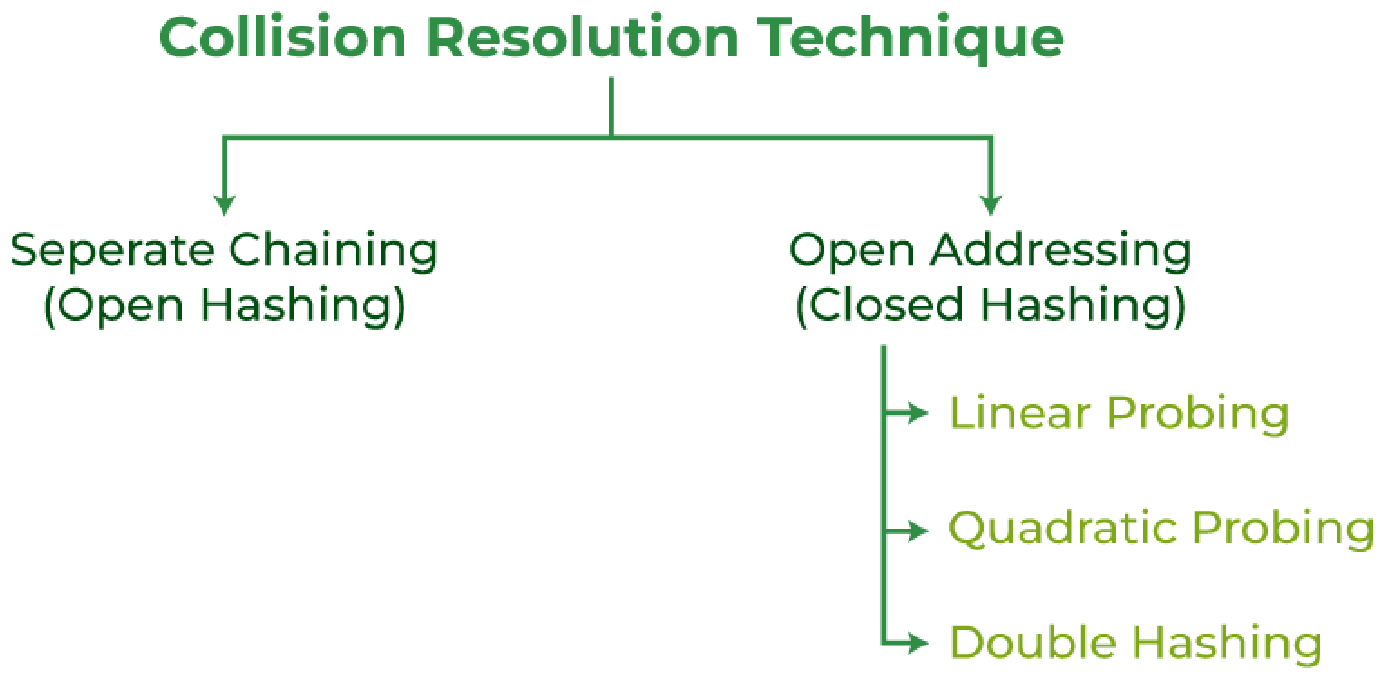
What is collision?

The hashing process generates a small number for a big key, so there is a possibility that two keys could produce the same value. The situation where the newly inserted key maps to an already occupied, and it must be handled using some collision handling technology.

Collision Resolution

We can resolve the hash collision using one of the following techniques.

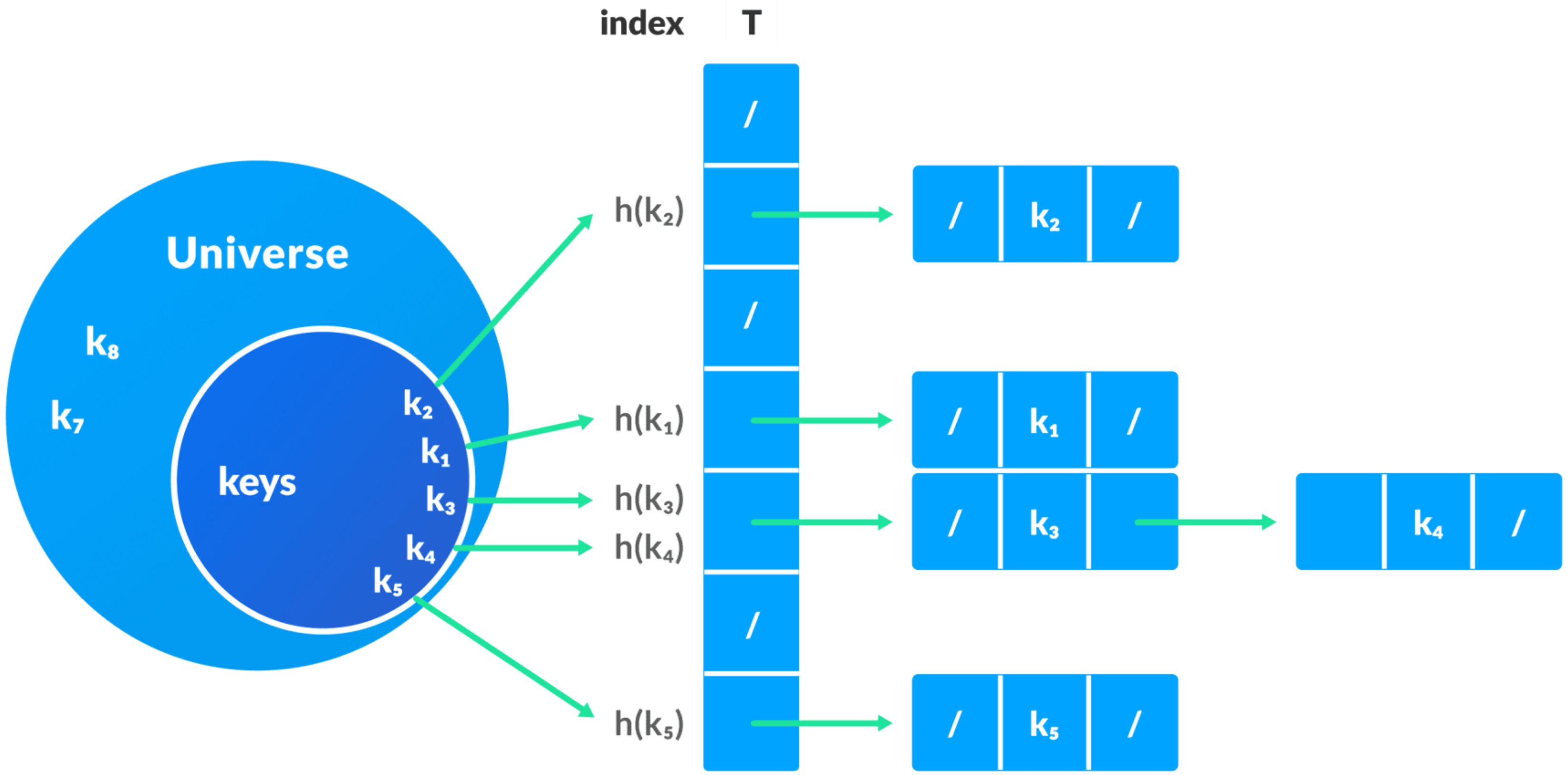
- Collision resolution by chaining
- Open Addressing: Linear/Quadratic Probing and Double Hashing


Collision Resolution Technique
Seperate Chaining (Open Hashing)
Open Addressing (Closed Hashing)
Linear Probing
Quadratic Probing
Double Hashing

1. Collision resolution by chaining

In chaining, if a hash function produces the same index for multiple elements, these elements are stored in the same index by using a doubly-linked list.

If j is the slot for multiple elements, it contains a pointer to the head of the list of elements. If no element is present, j contains NIL.



2. Open Addressing

Unlike chaining, open addressing doesn't store multiple elements into the same slot. Here, each slot is either filled with a single key or left NIL.
Different techniques used in open addressing are:

i. Linear Probing
ii. **Quadratic Probing**
iii. **Double hashing**

i.Linear Probing

In linear probing, collision is resolved by checking the next slot.

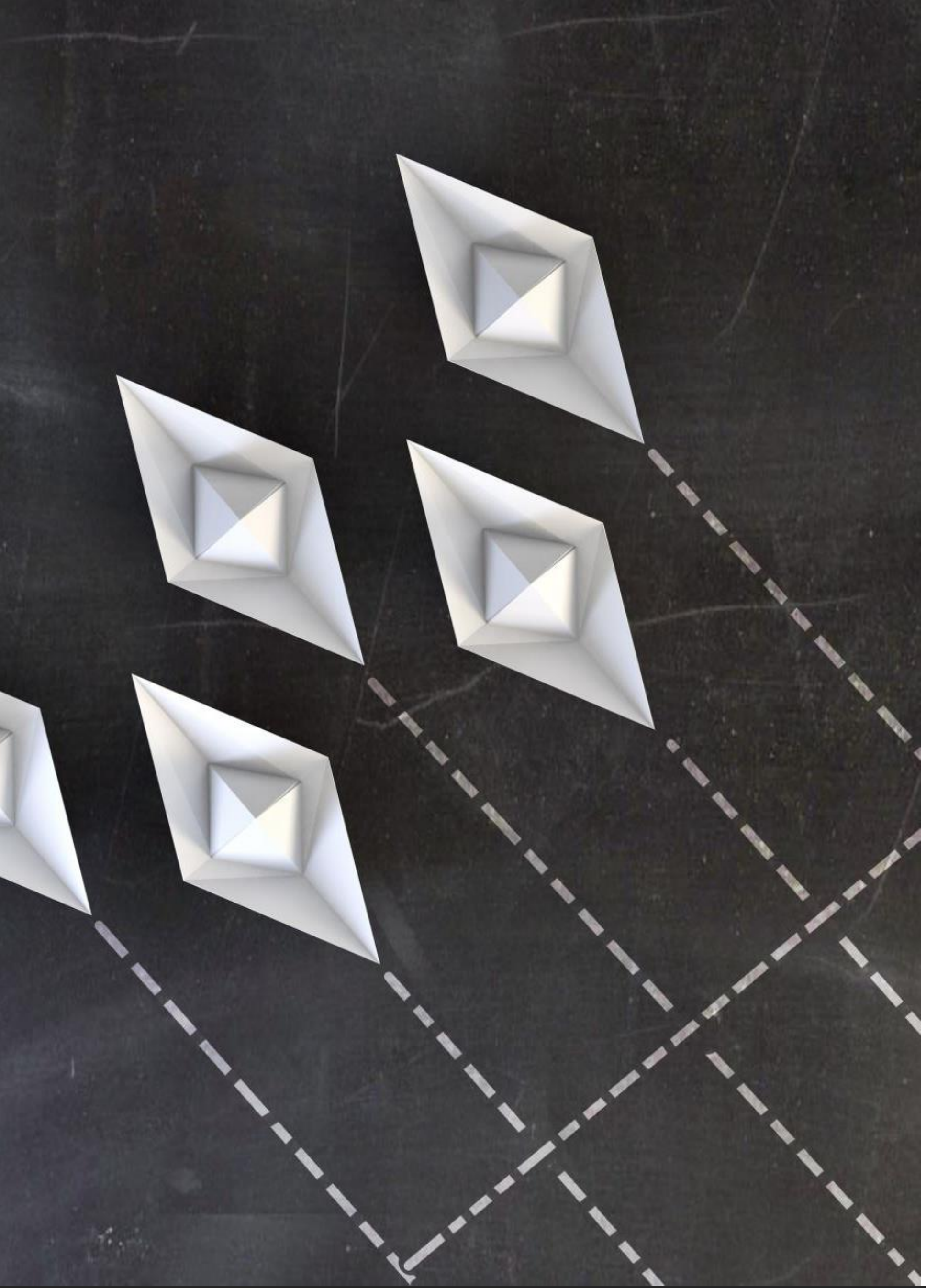
$h(k, i) = (h(k) + i) \bmod m$

where

- $i = \{0, 1, ...\}$

If a collision occurs at $h(k, 0)$, then $h(k, 1)$ is checked. In this way, the value of i is incremented linearly.

ii. Quadratic Probing



It works similar to linear probing but the spacing between the slots is increased (greater than one).

ii. Quadratic Probing

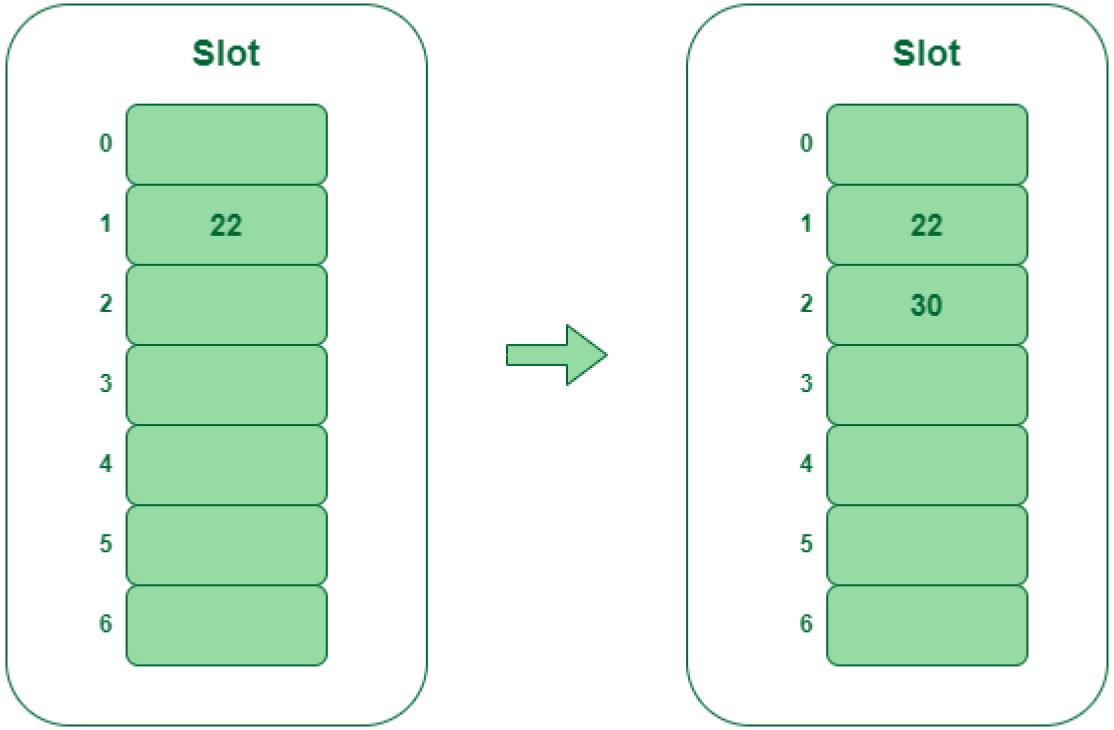
An example sequence using quadratic probing is:

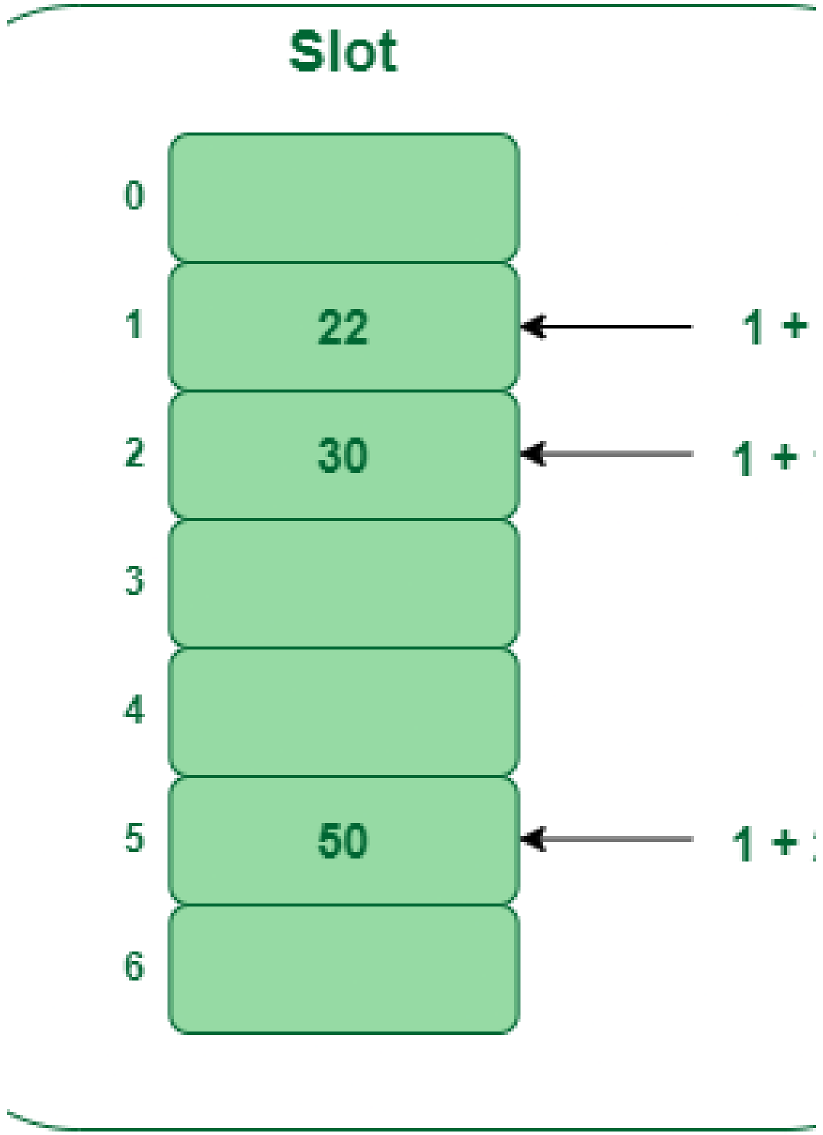
H + 1^2, **H** + 2^2, **H** + 3^2, **H** + 4^2……………………. **H** + k^2

This method is also known as the mid-square method because in this method we look for i^2'th probe (slot) in i'th iteration and the value of i = 0, 1, . . . n – 1. We always start from the original hash location. If only the location is occupied then we check the other slots.

Example:

Let us consider table Size = 7, hash function as Hash(x) = x % 7 and collision resolution strategy to be $f(i) = i^2$. Insert = 25, 33, and 105





Example

Inserting 50

Hash(50) = 50 % 7 = 1

- In our hash table slot 1 is already occupied. So, we will search for slot 1+1^2, i.e. 1+1 = 2,
- Again slot 2 is found occupied, so we will search for cell 1+2^2, i.e.1+4 = 5,
- Now, cell 5 is not occupied so we will place 50 in slot 5.

iii. Double hashing

If a collision occurs after applying a hash function h(k), then another hash function is calculated for finding the next slot.

- The first hash function is **h1(k)** which takes the key and gives out a location on the hash table. But if the new location is not occupied or empty then we can easily place our key.
- But in case the location is occupied (collision) we will use secondary hash-function **h2(k)** in combination with the first hash-function **h1(k)** to find the new location on the hash table.

Load Factor in Hashing

The load factor of the hash table can be defined as the number of items the hash table contains divided by the size of the hash table. Load factor is the decisive parameter that is used when we want to rehash the previous hash function or want to add more elements to the existing hash table.

```
Load Factor = Total elements in hash table/ Size of hash table
```

Rehashing?

As the name suggests, rehashing means hashing again. Basically, when the load factor increases to more than its predefined value (the default value of the load factor is 0.75), the complexity increases. So to overcome this, the size of the array is increased (doubled) and all the values are hashed again and stored in the new double-sized array to maintain a low load factor and low complexity.

Applications of Hash Data structure

- Hash is used in databases for indexing.
- Hash is used in disk-based data structures.
- Hash is used for cache mapping for fast access to the data.
- Hash is used in cryptography as a message digest.

Credits and Acknowledgements

https://www.gatevidyalay.com

https://www.programiz.com/