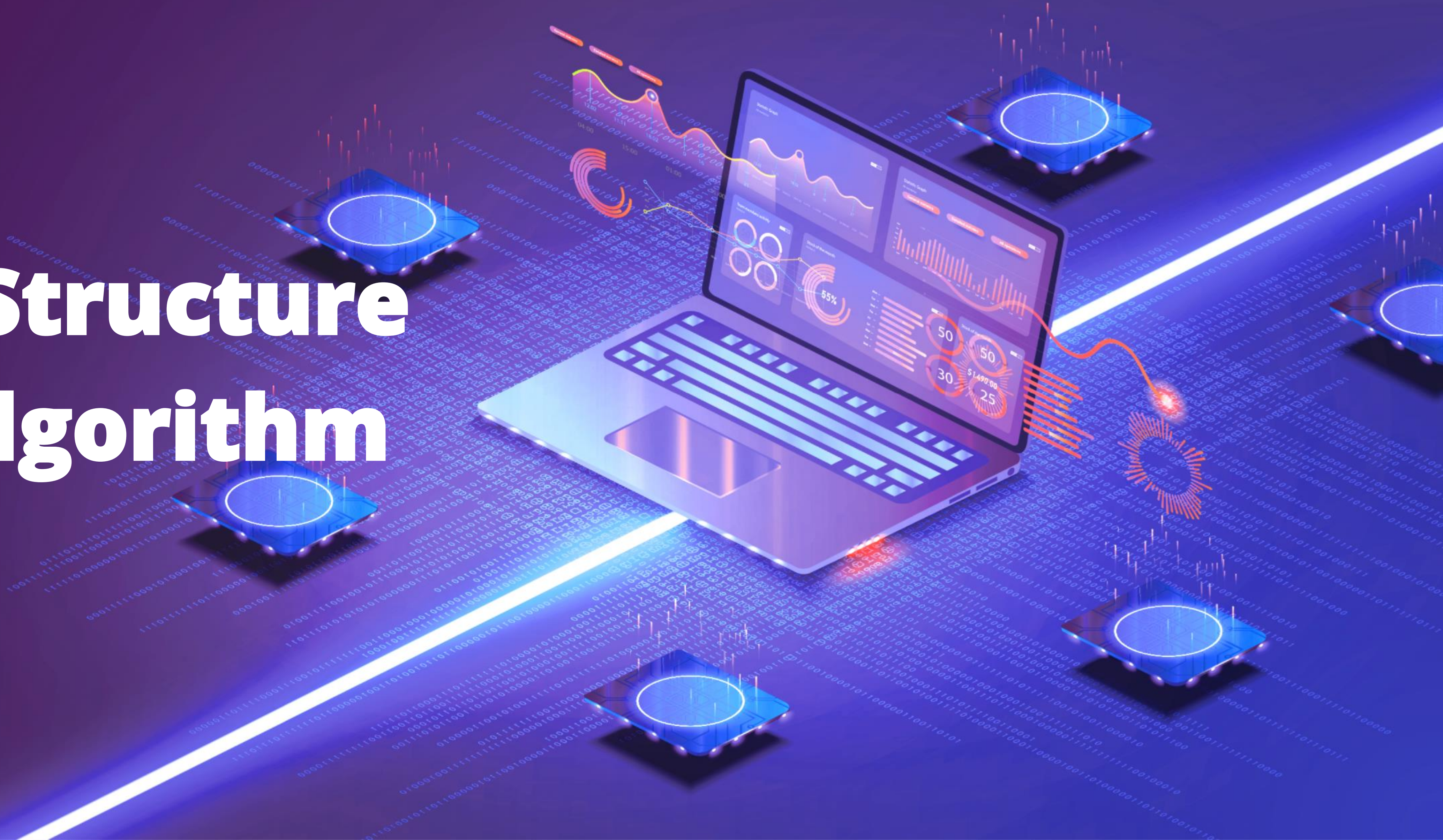
WILEY

ENABLING DISCOVERY | POWERING EDUCATION | SHAPING WORKFORCES

Data Structure and Algorithm



DATA STRUCTURES ALGORITHMS

HASH

Hash Data Structure

Let's learn what hash table is. Also, you will find working examples of hash table operations in C, C++, Java and Python.

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.

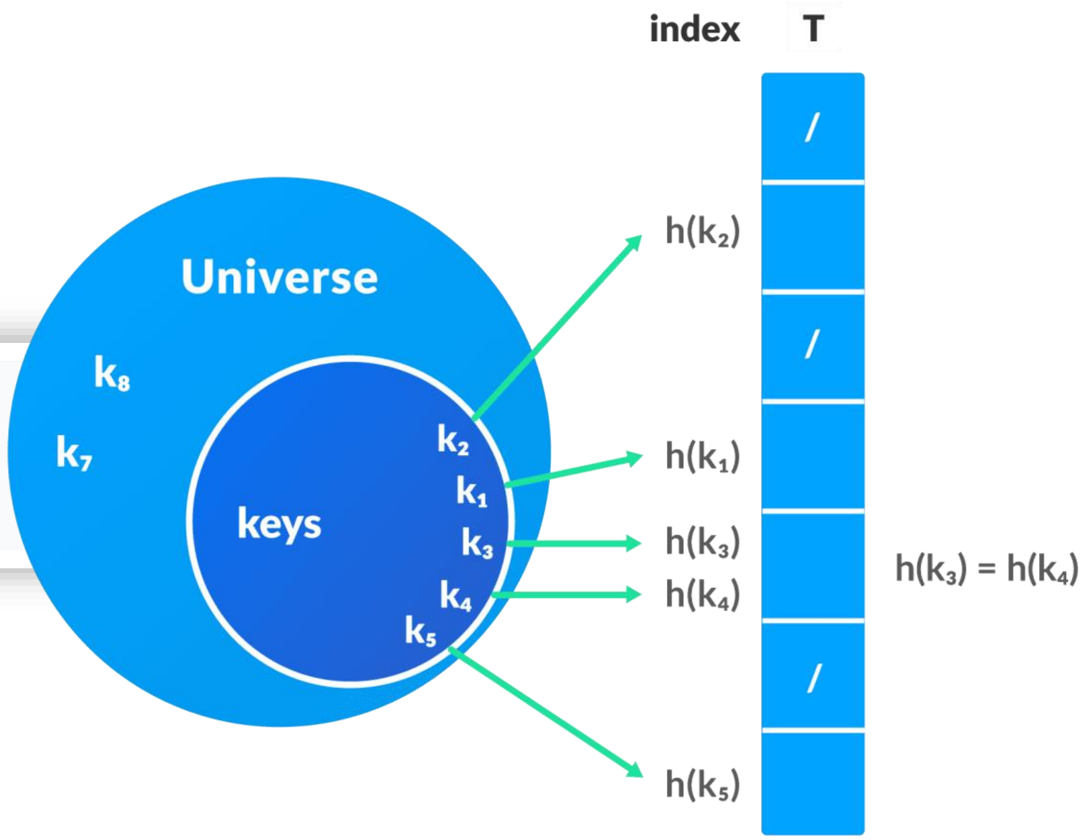
Key and Value in Hash table

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`.



Hash Collision

When the hash function generates the same index for multiple keys, there will be a conflict (what value to be stored in that index). This is called a **hash collision**

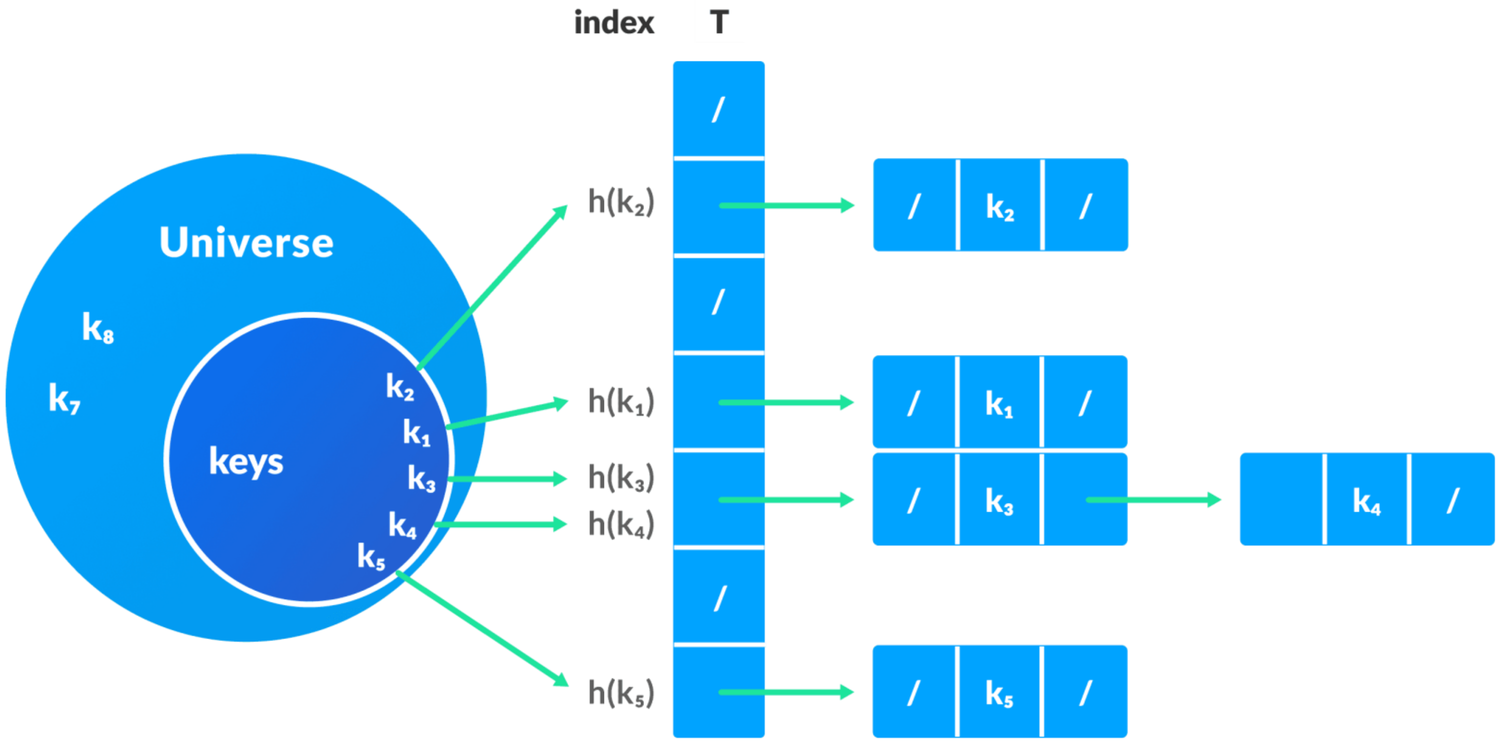
We can resolve the hash collision using one of the following techniques.

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

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.



Pseudocode for operations

```
chainedHashSearch(T, k)
  return T[h(k)]
chainedHashInsert(T, x)
  T[h(x.key)] = x //insert at the head
chainedHashDelete(T, x)
  T[h(x.key)] = 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

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

`h(k, i) = (h'(k) + i) mod m`

where

- `i = {0, 1, ….}`
- `h'(k)` is a new hash function

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

The problem with linear probing is that a cluster of adjacent slots is filled. When inserting a new element, the entire cluster must be traversed. This adds to the time required to perform operations on the hash table.

ii. Quadratic Probing

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

$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$

where,

- c_1 and c_2 are positive auxiliary constants,
- $i = \{0, 1, \ldots\}$

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.

$h(k, i) = (h_1(k) + i h_2(k)) \bmod m$

Good Hash Functions

A good hash function may not prevent the collisions completely however it can reduce the number of collisions.

Here, we will look into different methods to find a good hash function

1. Division Method

If `k` is a key and `m` is the size of the hash table, the hash function `h()` is calculated as:

`h(k) = k mod m`

For example, If the size of a hash table is `10` and `k = 112` then `h(k) = 112` mod `10 = 2`. The value of `m` must not be the powers of `2`. This is because the powers of `2` in binary format are `10, 100, 1000, …`. When we find `k mod m`, we will always get the lower order p-bits.

```
if m = 22, k = 17, then h(k) = 17 mod 22 = 10001 mod 100 = 01
if m = 23, k = 17, then h(k) = 17 mod 22 = 10001 mod 100 = 001
if m = 24, k = 17, then h(k) = 17 mod 22 = 10001 mod 100 = 0001
if m = 2p, then h(k) = p lower bits of m
```

2. Multiplication Method

$h(k) = \lfloor m(kA \bmod 1) \rfloor$

where,

- `kA mod 1` gives the fractional part `kA`,
- `⌊ ⌋` gives the floor value
- `A` is any constant. The value of `A` lies between 0 and 1. But, an optimal choice will be `≈ (√5-1)/2` suggested by Knuth.

3. Universal Hashing

In Universal hashing, the hash function is chosen at random independent of keys.

Applications of Hash Table

Hash tables are implemented where

- constant time lookup and insertion is required
- cryptographic applications
- indexing data is required