# Hashing

## What is Hashing?

Method of sorting and indexing the data. It allows large number of data to be indexed using keys commonly called formula.

**Allocation of magic numbers**

Abc 🡪 5

XYZ 🡪 18

def 🡪 7

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | …. | 5 |  | 7 |  | 18 |  |  |
|  |  | abc |  | def |  | xyz |  |  |

## Why Hashing?

Time efficient: Search - O(1)/O(n)

**Terminology**:

**Hash Function**: Helps in mapping data of arbitrary size to fixed size.

**Key**: Input given by the user

**Hash Value**: Values returned by Hash function. It can be also called hash codes, digits or hashes.

**Hash table**: Associates array abstract structure, a structure that gets mapped as key to value.

**Collision**: 2 or more different keys of hash function gets to same output.

# Introduction to hash function

Technique to get the hash value from the mentioned character or string

Summation of all character of ASCII code

Divisor

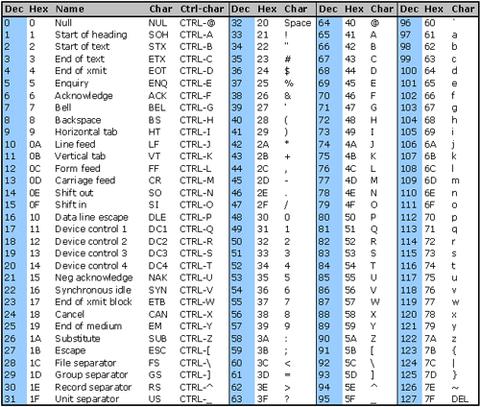
Modulus

**Algorithm**:

Loop through the charcters

Add the Ascii values

Hash value= Sum%divisor



# Collision in Hashing

Collision is 2 keys mapping to same value creating confusion of storing values

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | …. | 5 | 6 | 7 |  | 18 |  |  |
|  |  | abc | thf | def |  | xyz |  |  |

# Steps to avoid this:

In case of collision: Create a linked list at every key. So that we can store the value at the same key.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | …. | 5 | 5 | 7 |  | 18 |  |  |
|  |  | 111 | thf | def |  | xyz |  |  |

|  |  |  |
| --- | --- | --- |
| 111 |  |  |
| Abc - null |  |  |

Collision Resolution Technique

1. Direct chaining
2. Open addressing
3. Linear probing
4. Quadratic probing
5. Double Hashing

# Collision Resolution Technique

We have 2 types:

1. Direct chaining
2. Open Addressing

* Linear Probing
* Double Hashing
* Quadratic probing

Abc- 2 def- 2 xyz -2

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | …. | 2 | 3 | 4 | 5 | 6 |  |  |
|  |  |  |  |  |  |  |  |  |

## Direct chaining:

Creates a linked list and stores the collision values.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | …. | 2 | 3 | 4 | 5 | 6 |  |  |
|  |  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

## Linear Probing:

If collision, check the next cell and fill the value in the next value.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | …. | 2 | 3 | 4 | 5 | 6 |  |  |
|  |  |  |  |  |  |  |  |  |

## Quadratic probing:

Abc- 2 def- 2 xyz -2

If collision, add Square of 1,2,3… with the key and insert the value.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | …. | 2 | 3 | 4 | 5 | 6 |  |  |
|  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | …. | 2 | 3 | 4 | 5 | 6 |  |  |
|  |  |  |  |  |  |  |  |  |

## Double probing:

If collision, it creates another Hash function and creates the key .

Add the key with existing key and store the value, if case of further collision, multiply the key with 2, 3, 4 with the key generated by the secondary function.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | …. | 2 | 3 | 4 | 5 | 6 |  |  |
|  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | …. | 2 | 3 | 4 | 5 | 6 |  |  |
|  |  |  |  |  |  |  |  |  |

# What happens when collision is full

Collision Resolution Technique

1. Direct chaining – Do not have this issue
2. Open addressing
3. Linear probing
4. Quadratic probing
5. Double Hashing

Direct Chaining:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

Open addressing:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  |  |  |  |  |  |  |  |

# Collision resolution techniques compared

## Direct chaining:

No issues with size

Chances of creating a big LL creating problem with Time complexity

## Open Addressing:

Easy Implementation

Size gets exhausted

Cases of usage:

Input Size is known -> Open addressing

More deletion/Searching -> Direct chaining

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  |  |  |  |  |  |  |  |

# Practical use of hashing

Password verification

File storage at the disk

# Hashing vs Other Data Structure:

