# Hashing

PAGE NO.: 11
10 | 11 | 2021

Hashing is a technique of mapping a large set of arbitrary data to tabular indexes using a hash function.

It is a method of representing dictionaries for large dataset.

It allows lookups, updating and retrieval operation to occur in a constant time ie. O(1)

## Why it is required?

Linear and Binary search perform lookup/search with complexity of O(n) and O(log n) resp. As the size of the dataset increases, these complexities also become significantly high, which is not acceptable.

We need a technique that is independent of the size of data. Hashing allows to do this in constant time O(1).

## Hash function -

A hash function is used for mapping each element of a dataset to indexes in the table.

## Hash table

Hash table is the data structure that stores elements in key-value pairs -

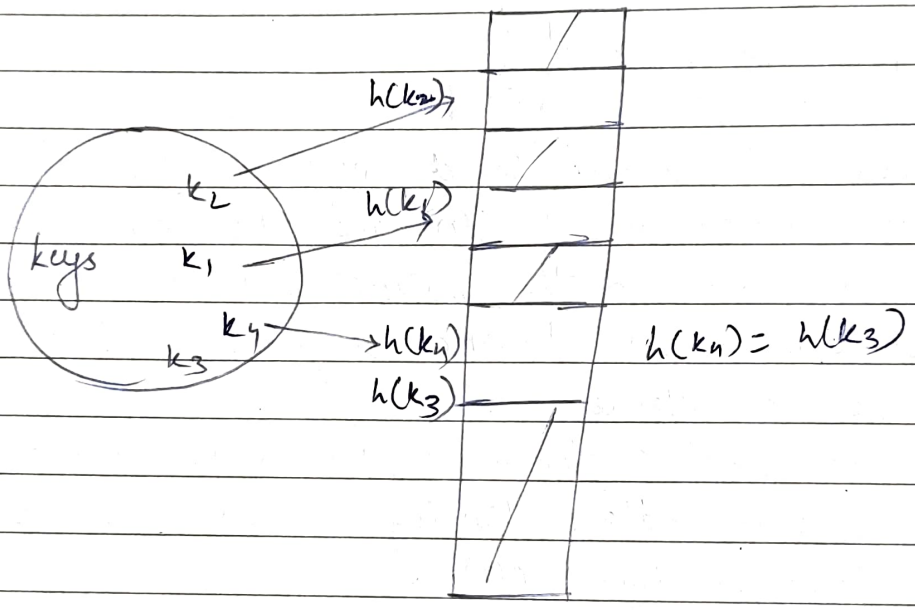
Key - unique integer that is used for indexing the values

Value - data that are associated with keys.

| key | data |

In a hash table, a new index is processed using a key. And, the element corresponding to that key is stored in the index. This 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'.




Search keys – 24, 52, 91, 67, 48, 83

Hash function -
- k mod n
- Mid square method
- Folding method

k mod 10

24 mod 10 = 4

| | |
|---|---|
| 0 | |
| 1 | 91 |
| 2 | 52 |
| 3 | 83 |
| 4 | 24 |
| 5 | |
| 6 | |
| 7 | 67 |
| 8 | 48 |

## * Division Method.

$h(k) = k \bmod m$

m = larger than numbers

to minimize collision, m should be prime or no. with small divisor.

Ex: 10, 15, 12 if m = 5

$h(10) = 10 \bmod 5 = 0$

$h(15) = 15 \bmod 5 = 0 \Rightarrow$ collision

$h(12) = 12 \bmod 5 = 2$

## * Mid-Square Method.

$h(k) = l$

key is squared ($k^2$).

'l' is obtained by deleting the digits from both ends.

| | | | |
|---|---|---|---|
| k - | 10 | 15 | 12 |
| $k^2$ - | 100 | 225 | 144 |
| l - | 0 | 2 | 4 |

## * Folding Method

$h(k) = k_1 + k_2 + k_3 + \dots + k_n$

- keys are partitioned into parts.
- parts are added together.

| | | | |
|---|---|---|---|
| k - | 10 | 21 | 11 |
| | 1, 0 | 2, 1 | 1, 1 |
| h(k) = | 1 | 3 | 2 |

## * Multiplication Method

1) Choose constant 'A' such that $0 < A < 1$
2) Multiply 'A' with 'key'.
3) Extract the fractional part of k·A
4) Multiply the result of above by size of hash table i.e. M.
5) Resulting hash value is obtained by taking the floor of the result in (4).

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

Ex:- $k = 12345$

$A = 0.35784$

$M = 100$

$$h(12345) = \text{floor}\left[100\,(12345 \times 0.35784 \bmod 1)\right]$$

$$= \text{floor}\left[100\,(4417.53485 \bmod 1)\right]$$

$$= \text{floor}\left[100 \times 0.53485\right]$$

$$= \lfloor 53.48 \rfloor$$

$$\boxed{h(12345) = 53}$$

* **Collision :-**

Since a hash fuction generates small no. for a given key, there is a possibility that two keys result in the same value. The situation where a newly inserted key maps to an already occupied slot in hash table is called collision.

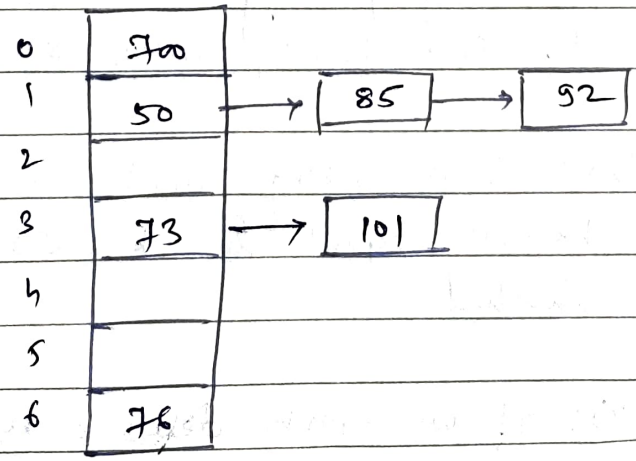
* **Collision Resolution Techniques.**

1. Separate Chaining
2. Open Addressing

**(1) Separate Chaining :-**

This is most popular and commonly used technique. Linked list data structure is used to implement this technique. When multiple keys are hashed into same index, then these elements are inserted into a singly-linked list (known as chain).

Ex :- $h(k) = k \bmod 7$ keys - 50, 700, 76, 85, 92, 73, 101



Advantages :-

* Easy to implement
* Hash table never fills up, We can always add more elements to chain.
* Less sensitive to hash function
* Mostly used when it is unknown how frequently keys are inserted or deleted.

Disadvantages:

* Wastage of space
* If chain becomes long, then searching can become O(n) in worst case
* Use extra spaces for links.

(d) Open Addressing.

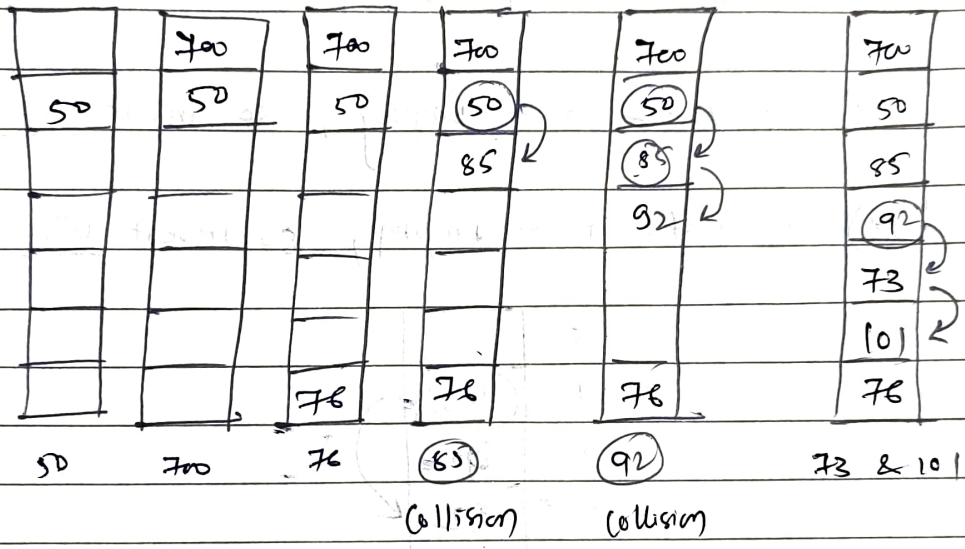
Here, all elements are stored in hash table itself. So, at any time, size of hash table must be greater than or equal to the no. of keys to be stored. This is based on probing.

- Insert (k) :- keep probing until an empty slot is found. Once found, insert k.
- Search (k) : keep probing until slot's key doesn't become equal to 'k' or an empty slot is reached.
- Delete (k): If we simply delete a key, then search may fail. So, deleted key's slots are marked as "deleted" Insert(k) can 'insert at deleted slot but search(k) doesn't stop at deleted slot.

A) Linear probing

Here, hash table is searched sequentially, that starts from the original location of hash. If location that we get is occupied, then we check for the next location.

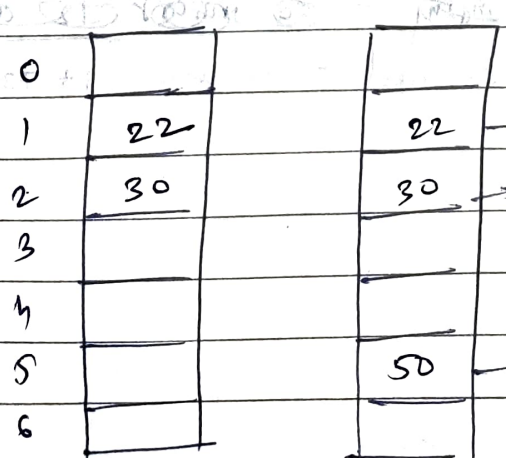
Ex: $h(k) = k \bmod 7$ keys are - 50, 700, 76, 85, 92, 73, 101.




B) Quadratic Probing

Here, we look for the $(i^{th})^2$ slot in $i^{th}$ iteration. We always start from original hash location. If location is occupied, then we check for other slots.

Ex: $h(k) = k \bmod 7$ Insert - 22, 30, 50.




50 % 7 = 1, occupied
so search - 1+1² = 2
Again occupied, so
search 1+2² = 4, unoccupied
Hence, Insert at 4th index

c) **Double Hashing**

In this techniques, the increment for probing sequence are computed by using another hash function.

If $h(x)$ is full, we check $[h(x) + i * h_2(x)] \% m$. where 'm' is table size. $i = 1, 2, \ldots$

Ex:- Hash table size = 7

$h_1(k) = k \bmod 7$ $\quad h_2(k) = 1 + k \bmod 5$

Insert - 27, 43, 92, 74

(1) $27 \% 7 = 6$ - Empty so insert 27 in 6th slot

(2) $43 \% 7 = 1$ : Empty. so insert 43 in 1st slot

(3) $92 \% 7 = 1$ - occupied so check -

$[h_1(k) + i * h_2(k)] \% 7$

$= [92 \% 7 + 1 * (1 + 92 \% 5)] \% 7$

$= [1 + 1 * (1+2)] \% 7 = 4 \% 7 = 4.$

Insert 92 at 4th slot

(4) $74 \% 7 = 4$ - occupied -

$[74 \% 7 + 1 * (74 \% 5)] \% 7$

$= [4 + 1 * (1+4)] \% 7 = 9 \% 7 = 2$

Insert 74 at 2nd slot

| | |
|---|---|
| 0 | |
| 1 | 43 |
| 2 | 74 |
| 3 | |
| 4 | 92 |
| 5 | |
| 6 | 27 |