HASH TABLE

Speciality: O(1) time complexity, for all the dictionary operations.

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 | Value |

pairs.

Dictionary operations: INSERT, SEARCH AND DELETE

Different Names:

HashMap	Object	Dictionary
Java	JavaScript	Python / C#

- In HashMaps, we can't have duplicate Keys.

Methods: Map<Key (Data Type), Value (Data Type)> map = new HashMap<>();

i. map.put() // To add a key,value pair

ii. map.remove() // To remove

iii. map.get(Key) // Get the value at a particular key

iii map.containsKey() // Return a boolean if the key is in the Hash Table.

iv. map.containsValue() // It has to go over all the elements, to check whether the value exists or not.
↓
O(n)

v. map.keySet() // Returns all the keys

vi. map.entrySet() // Returns all the key value pairs.

Maps: Key → Value

Sets: Key

- A set only contains keys. And it doesn't allow duplicates.

Eg:

```
Set<Integer> set = new HashSet<>();
int[] numbers = {1,2,3,3,2,1,4};

for(int num : numbers)
    set.add(num);

System.out.println(set) //output: {1,2,3,4}.
```

Methods:

set.add()
set.remove()
set.removeAll()
set.contains()
set.containsAll()
set.size()
set.clear()

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 known as Hashing.

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 known as Hash Collision.

Simply: When two keys generates same index to store their values by going through some kind of hash function.

Resolving Hash Collisions: [hash 1 = Key % table_size]

(In Integer form)

- Closed / Separate Chaining: Elements are stored in the same index by using a doubly linked list.
- Open Addressing: By Probing (searching for an empty index).
  - Linear Probing: (hash 1 + i) % table_size
  - Quadratic Probing: (hash1 + i²) % table_size (squared)
  - Double Hash: (hash 1 + i * hash2) % table_size

    hash 2 (Key): prime − (Key % prime)

Some basic algorithms:

To find the frequency of all the keys in HashMap.

eg: Str = " a mango man"

```
Map<String, Integer> map = new HashMap<>();

for (char key: str.toCharArray())
{
    int count = map.containsKey(key) ? map.get(key) : 0;
    map.put(key, count +1);
}

System.out.println(map);
// Output: {a=3, _=2, m=2, n=2, g=1, o=1}
```

→ It will grab the value at key.

Count the frequency