

Hashing

* Hashset :-

- * contains all unique elements & removes duplicate.
- * elements in the set are not ordered (jumbled)
- * *** Complexity :- Insert/Add $\rightarrow O(1)$; Search/contain $\rightarrow O(1)$; Delete/remove $\rightarrow O(1)$;

Array ($O(N)$)
BST ($O(\log N)$)

```

* HashSet<Integer> set = new HashSet<>();  $\rightarrow$  same as arraylist
set.add(1);
set.add(2);
set.add(1);  $\rightarrow$  no need (won't consider duplicate)
set.contains(1) // true

```

* Iterator : * import iterator first
 hasNext \rightarrow next * doesn't guarantee order of items when iterated
 will be same next time.

* Use case :- for iteration in set HashSet

```

Iterator it = Set.iterator();
while (it.hasNext()) {
    print(it.next());
}

```

O/P
1
2

* HashMap

- * Must have key, value pair where key should be unique
- * HashMap<String, Integer> map = new HashMap<>();

\swarrow
key
(countries)

\searrow
value
(population)

```

* Inserting  $\rightarrow$  map.put("India", 70);
               map.put("USA", 90);
               map.put("China", 70);

```

```

* print(map);  $\rightarrow$  O/P

```

{China=70, US=90, India=70} \rightarrow order not guaranteed.

✓ * Updating → new value with same key name ∴ `map.put("china", 200);`
(value only)

✓ Note:- in Updating
 → exist: updates value of key respectively
 → not exist: adds new key, value pair.

✓ * Searching (key) → `map.containsKey("china"); // true`

✓ * Get (value) → `map.get("china"); // 200`
 ↓
 from key `map.get("Indonesia"); // null`

Two types of 'for' loop:- 1. `int[] arr = {12, 15, 18};`
 `for (int i=0; i<arr.length; i++){`
 `print(arr[i] + " ");`
 `}`

{for each}

2. `for (int val : arr){`
 `print (val + " ");`
 `}`

* 1. `for (Map.Entry<String, Integer> e : map.entrySet()){`
 `print(e.getKey());`
 `print(e.getValue());`
 `}`

✓ 2. keyset → `Set<String> keys = map.keySet();`
 `for (String key : keys){`
 `print (key + " " + map.get(key));`
 gives value
 `}`

✓ * Deletion → `map.remove("china");`
 `print(map); ✓`

* Implementation (in next Page)

V.V.V imp for interviews

* Implementation

* functions \Rightarrow put, get, containskey, remove, ~~size~~^{search}, keyset

\downarrow
all in $O(1)$ complexities

* Implemented as array of LL.

* Basic Declaration:-

```
import java.util.*;
```

```
public class Hashcode {
```

Generics (data type is dependable)

static (no need to instantiate)

```
static class HashMap<K,V> {
```

check

```
private class Node {
```

datatype of key is K

& value is V

dependable on

input

count of both

```
    K key;
```

```
    V value;
```

```
public Node (K key, V val) {
```

```
    this.key = key;
```

```
    this.value = val;
```

```
}
```

no. of nodes inside bucket

bucket.length = N

```
private int n;
```

```
private int N;
```

!(int[] arr)

```
private LinkedList<Node> [] buckets;
```

(index are buckets)

index 0 \rightarrow

nodes in LinkedList
n

since nodes are stored

Due to generics, Java mein type nhi define
karo chahoga but not in L.L. used

```
@SuppressWarnings("unchecked")
```

```
public HashMap() {
```

initialised bucket size with 4

```
    this.N = 4;
```

"we brain"

initialised bucket with LL size 4

```
    this.buckets = new LinkedList[4];
```

```
    for (i=0 to 4) {
```

same indices of L.L.
lagi diye

```
        this.buckets[i] = new new LinkedList<>();
```

```
    }
```


✓* PUT

T.C. = $O(1)$, worst = $O(n)$

- Get the bucket index "bi" i.e. index of buckets array.
- " " data "di" i.e. at bi jo LL stored hai uska index
∴ we need bi and key to get di.
- If di gives -1 ∴ key doesn't exist ∴ add new node at that bi and update the size of 'n'
- Else if key exist ∴ update value at di.

return type = void, arg = K key, V val

int bi = hashFunction(key);int di = searchin LL(key, bi);

if (di == -1) { // key doesn't exist → add new node + increase n

✓ buckets[bi].add(new Node(key, val));

property of LL

y n++;

else { // key exists → get the value then place new one

✓ Node data = buckets[bi].get(di);

data.value = val;

y

Just to make sure ek hi index ← double lambda = (double) n / N;

† bada LL banta na chala jaye

if (lambda > 2.0) {

(let k=2)

threshold

rehash();

}

written like
this ∴ double
(must)

✓* hashFunction → return type = int, arg = K key

int bi = key.hashCode();

return Math.abs(bi) % N;

- hashCode generates unique code, here it will generate based on key
(ranges like 12765 or -314411)
- We get bi positive values only, that too in range of N.

* searchinLL → return type = int, arg = key, int bi.

provide LL inside bi → `LinkedList<Node> ll = buckets[bi];`

`for (i=0 to ll.size()-1) {`

✓ * `if (ll.get(i).key == key) {`

`return i; → 'di' found`

`} return -1; → 'di' does not exist`

✓ * rehash function → return type = void, no args

firstly jo buckets me tha use old bucket hi daalo
buckets ka size double karo

saare naye buckets hi linked list banao

aur har old bucket ke index pe jo ll hai use get karo
aur put kar do ~~new~~ waha pe old wale dit.

① `LinkedList<Node> oldBucket = buckets;`

② `buckets = new LinkedList[N*2];`

③ `for (i=0 to buckets.length or N*2) {`

`buckets[i] = new LinkedList<>();`

④ * `for (i=0 to oldBucket.length) {`

`LinkedList<Node> ll = oldBucket[i];`

`for (j=0 to ll.size()-1) {`

`Node node = ll.get(j);`

`put(node.key, node.value);`

✓ ⑤ main fn. :- psum → `HashMap<String, Integer> map = new HashMap<>();`
generics will take the datatypes.

`map.put("India", 190); ... more put`

`ArrayList<String> keys = map.keySet();`

`for (i=0 to keys.size()-1) {`

* → `print(keys.get(i) + " " + map.get(keys.get(i)));`

`map.remove("India");`

`print(map.get("India")); → null.`

✓ * GET return type = "V", arg = K key
 int bi = hashFunction(key);
 int di = searchInLL(key, bi);
 if (di == -1) {
 return null; → key doesnot exist
 } else {
 get the node from ← Node node = buckets[bi].get(di);
 ↳ if key exists → return node.value; → get the value.
 }

✓ * Contains key return type = boolean, arg = K key
 int bi = ✓
 int di = ✓
 if (di == -1) {
 return false;
 } else {
 return true;
 }
 ↳ viewing purposes

✓ * Remove return type = V, arg = K key.
 int bi = ✓
 int di = ✓
 if (di == -1) { return null; } → DNS, no remove
 else {
 Node node = buckets[bi].remove(di);
 * n--;
 return node.value; → viewing purposes.

✓ * IsEmpty boolean, no arg
 return n == 0; → if no nodes in hashmap then Empty.

✓ * keySet return = ArrayList<K>, no arg
 ArrayList<K> keys = new ArrayList<>();
 for (i = 0 to buckets.length) {
 LL<node> ll = buckets[i];
 for (j = 0 to ll.size)
 Node node = ll.get(j);
 keys.add(node.key);
 }
 return keys;

Sabko add krdo key list
 me aur return kardo.

↳ v. easy just
 understand
 conceptually