

Hashing , Dictionary, and Map

Abdul Ghafoor

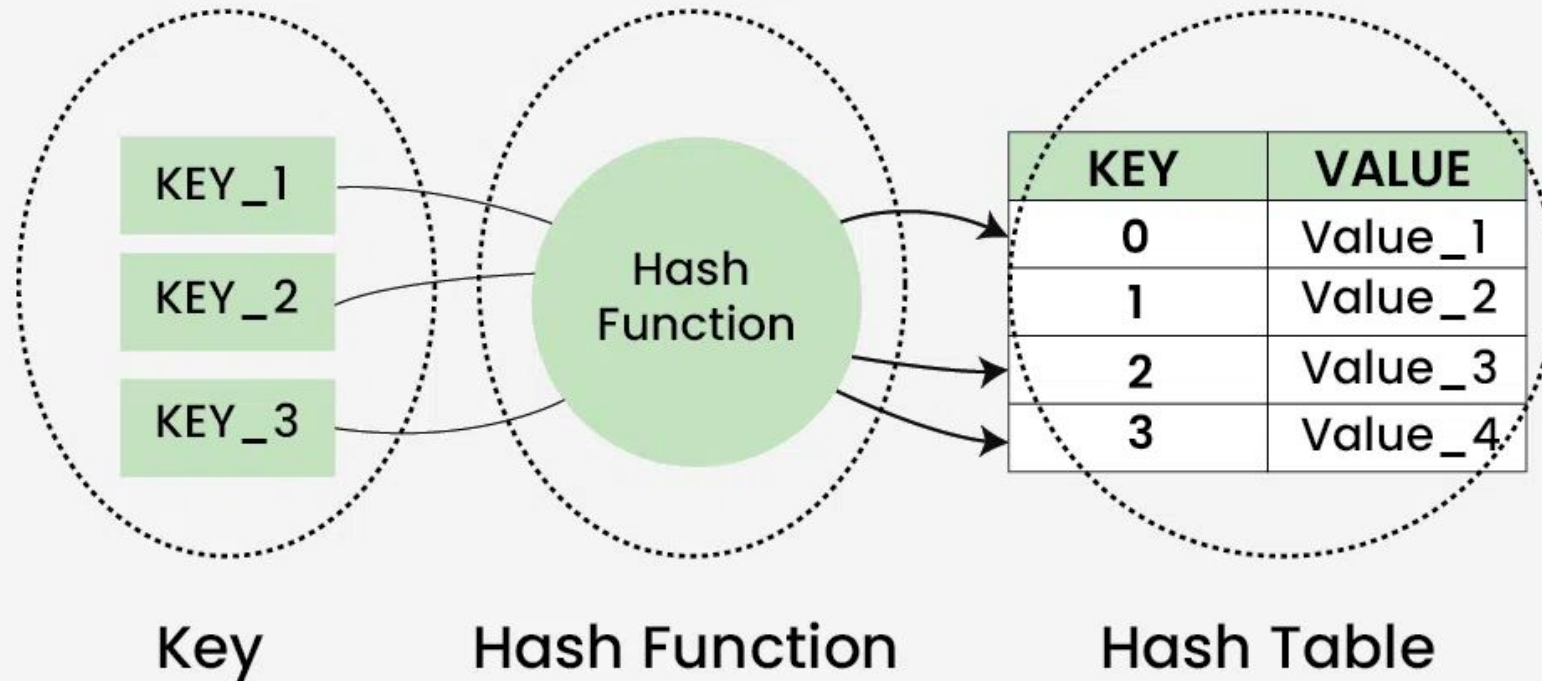
Hashing

Hashing in Data Structures refers to the process of transforming a given key to another value. It involves mapping data to a specific index in a hash table using a hash function that enables fast retrieval of information based on its key.

The transformation of a key to the corresponding value is done using a **Hash Function** and the value obtained from the hash function is called **Hash**



Components of Hashing



Components of Hashing

There are majorly three components of hashing:

Key: A **Key** can be anything string or integer which is fed as input in the hash function the technique that determines an index or location for storage of an item in a data structure.

Hash Function: The **hash function** receives the input key and returns the index of an element in an array called a hash table. The index is known as the **hash index** .


Hash Table: Hash table is a data structure that maps keys to values using a special function called a hash function. Hash stores the data in an associative manner in an array where each data value has its own unique index.

How does Hashing work?

Suppose we have a set of strings {"ab", "cd", "efg"} and we would like to store it in a table.



Step 1: We know that hash functions (which is some mathematical formula) are used to calculate the hash value which acts as the index of the data structure where the value will be stored.



Step 2: So, let's assign

"a" = 1,

"b"=2, .. etc, to all alphabetical characters.

How does Hashing work?

Step 3: Therefore, the numerical value by summation of all characters of the string:


$$\text{"ab"} = 1 + 2 = 3,$$

$$\text{"cd"} = 3 + 4 = 7 ,$$

$$\text{"efg"} = 5 + 6 + 7 = 18$$

How does Hashing work?

Step 4: Now, assume that we have a table of size 7 to store these strings. The hash function that is used here is the sum of the characters in **key mod Table size** . We can compute the location of the string in the array by taking the **sum(string) mod 7** .



Step 5: So we will then store

“ab” in $3 \bmod 7 = 3$,

“cd” in $7 \bmod 7 = 0$, and

“efg” in $18 \bmod 7 = 4$.



Mapping Key with indices of Array

0	1	2	3	4	5	6
cd			ab	egf		

Java.util.Dictionary Class in Java

The `java.util.Dictionary` class in Java is an abstract class that represents a collection of key-value pairs, where keys are unique and are used to access the values.

The Dictionary class is an abstract class and cannot be instantiated directly. Instead, it provides the basic operations for accessing the key-value pairs stored in the collection, which are implemented by its concrete subclass `java.util.Hashtable`.

```
import java.util.Dictionary;  
import java.util.Hashtable;
```

```
public class test {  
    public static void main(String[] args) {  
        Dictionary<Integer, String> dictionary = new Hashtable<>();  
        dictionary.put(1, "One");  
        dictionary.put(2, "Two");  
  
        System.out.println("Value for key 1: " + dictionary.get(1));  
    }  
}
```

Map

The Map interface replaced the Dictionary class starting from Java 1.2 as part of the Java Collections Framework.

Map provided a more flexible, modern, and type-safe way to handle key-value pairs, making Dictionary largely obsolete.

```
import java.util.HashMap;  
import java.util.Map;
```

```
public class test {  
    public static void main(String[] args) {  
        Map<Integer, String> map = new HashMap<>();  
        map.put(1, "One");  
        map.put(2, "Two");  
  
        System.out.println(map.get(1));  
    }  
}
```

java.util package

The java.util package contains a variety of data structures (DS) that are part of the Java Collections Framework as well as other utility classes. Here's a comprehensive list of the main data structures from the java.util package:

java.util package : List Interface and Implementations

The List interface represents an ordered collection of elements, where duplicate elements are allowed.

ArrayList: A resizable array implementation of the List interface.

LinkedList: A doubly-linked list implementation of the List interface.

Vector: A growable array of objects (part of legacy collections).

Stack: A last-in, first-out (LIFO) stack (extends Vector and is now outdated).

java.util package : Set Interface and Implementations

The Set interface represents an unordered collection of unique elements.

HashSet: A Set implementation based on a hash table, allowing for fast lookups.

LinkedHashSet: A Set implementation that maintains insertion order.

TreeSet: A Set implementation that stores elements in a sorted order using a red-black tree.

java.util package : Queue Interface and Implementations

The Queue interface represents a collection designed for holding elements in a queue, typically used in FIFO (First-In-First-Out) order.

PriorityQueue: A priority queue that stores elements in a sorted order based on their natural ordering or a custom comparator.

LinkedList: Can also be used as a Queue (implements both List and Queue).

ArrayDeque: A resizable array implementation of the Deque interface (can be used as a queue or stack).

java.util package : Map Interface and Implementations

The Map interface represents a collection of key-value pairs, where each key is unique.

HashMap: A hash table-based implementation of the Map interface (non-synchronized).

LinkedHashMap: A Map implementation that maintains insertion order.

TreeMap: A Map implementation that stores key-value pairs in a sorted order.

ConcurrentHashMap: A thread-safe version of HashMap designed for concurrent access.

WeakHashMap: A Map that holds keys as weak references, allowing them to be garbage collected when no longer in use.

