**Data Structure Implementation – Hashing**

**Part 1 – Hashing Overview:**

Suppose we are searching for an item in an array. If the array is not sorted, the search might require examining each and all elements of the array O(n). If the array is sorted, we can use the binary search, and therefore reduce the worse-case runtime complexity to O(log n). We could search even faster if we know in advance the index at which that value is located in the array. Suppose we do have that magic function that would tell us the index for a given value. With this magic function our search is reduced to just one probe, giving us a constant runtime O(1). Such a function is called a hash function . A hash function is a function which when given a key, generates an address in the table.

Part 2 – Hashing Data Structures:

|  |  |
| --- | --- |
| **Data Structure** | **Characteristics** |
| Map |  |
| Hash Set | * Underlying data structure for HashSet is hashmap. * As it implements the Set Interface, duplicate values are not allowed. * Objects that you insert in HashSet are not guaranteed to be inserted in same order. Objects are inserted based on their hash code |
| Hash Map | * HashMap is non synchronized. It is not-thread safe and can’t be shared between many threads without proper synchronization code whereas * Allows one nul key and multiple null values * Generally, better to use hash map for performance if you don’t need thread synchronization. |
| Hash Table | * Hashtable is synchronized. It is thread-safe and can be shared with many threads. * Doesn’t allow null key or null values |
| Linked Hash Set |  |

|  |
| --- |
| **Hash Set Implementation:**  **public** **class** HashSet<E> **extends** AbstractSet<E> **implements** Set<E>{  **private** **transient** HashMap<E,Object> map;  **private** **static** **final** Object *PRESENT* = **new** Object();    **public** HashSet() {  map = **new** HashMap<E,Object>();  }    **public** Iterator<E> iterator() {  **return** map.keySet().iterator();  }    **public** **int** size() {  **return** map.size();  }    **public** **boolean** isEmpty() {  **return** map.isEmpty();  }    **public** **boolean** contains(Object o) {  **return** map.containsKey(o);  }    **public** **boolean** add(E e) {  **return** map.put(e, *PRESENT*)==**null**;  }    **public** **boolean** remove(Object o) {  **return** map.remove(o)==*PRESENT*;  }    **public** **void** clear() {  map.clear();  }  }  **Explanation:** Hash Set is really just a hash map. For example, if you declare a hash set of strings, it creates a hashmap of <String, Object>. Every time you add an item to the hash set, it puts your item as key and PRESENT as the value in the hash map. When you call “contains”, it calls “containsKey” and looks for your string. Iterator returns an iterator of all the keys in the hashmap. |