**Design Hash Table - Solution**

Here are C++ and Java solutions for your reference. In our solution, we use an array to represent the hash set. Each element in the array is a bucket. And in each bucket, we use the array list (or vector in C++) to store all the values.

*Hash Set*

|  |
| --- |
| class MyHashSet {  private final int MAX\_LEN = 100000; // the amount of buckets  private List<Integer>[] set; // hash set implemented by array    /\*\* Returns the corresponding bucket index. \*/  private int getIndex(int key) {  return key % MAX\_LEN;  }    /\*\* Search the key in a specific bucket. Returns -1 if the key does not existed. \*/  private int getPos(int key, int index) {  // Each bucket contains a list.  List<Integer> temp = set[index];  if (temp == null) {  return -1;  }  // Iterate all the elements in the bucket to find the target key.  for (int i = 0; i < temp.size(); ++i) {  if (temp.get(i) == key) {  return i;  }  }  return -1;  }    /\*\* Initialize your data structure here. \*/  public MyHashSet() {  set = (List<Integer>[])new ArrayList[MAX\_LEN];  }    public void add(int key) {  int index = getIndex(key);  int pos = getPos(key, index);  if (pos < 0) {  // Add new key if key does not exist.  if (set[index] == null) {  set[index] = new ArrayList<Integer>();  }  set[index].add(key);  }  }    public void remove(int key) {  int index = getIndex(key);  int pos = getPos(key, index);  if (pos >= 0) {  // Remove the key if key exists.  set[index].remove(pos);  }  }    /\*\* Returns true if this set did not already contain the specified element \*/  public boolean contains(int key) {  int index = getIndex(key);  int pos = getPos(key, index);  return pos >= 0;  }  }  /\*\*  \* Your MyHashSet object will be instantiated and called as such:  \* MyHashSet obj = new MyHashSet();  \* obj.add(key);  \* obj.remove(key);  \* boolean param\_3 = obj.contains(key);  \*/ |

*Hash Map*

|  |
| --- |
| import javafx.util.Pair;  class MyHashMap {  private final int MAX\_LEN = 100000; // the amount of buckets  private List<Pair<Integer, Integer>>[] map; // hash map implemented by array    /\*\* Returns the corresponding bucket index. \*/  private int getIndex(int key) {  return key % MAX\_LEN;  }    /\*\* Search the key in a specific bucket. Returns -1 if the key does not existed. \*/  private int getPos(int key, int index) {  // Each bucket contains a list.  List<Pair<Integer, Integer>> temp = map[index];  if (temp == null) {  return -1;  }  // Iterate all the elements in the bucket to find the target key.  for (int i = 0; i < temp.size(); ++i) {  if (temp.get(i).getKey() == key) {  return i;  }  }  return -1;  }  /\*\* Initialize your data structure here. \*/  public MyHashMap() {  map = (List<Pair<Integer, Integer>>[])new ArrayList[MAX\_LEN];  }    /\*\* value will always be positive. \*/  public void put(int key, int value) {  int index = getIndex(key);  int pos = getPos(key, index);  if (pos < 0) {  // Add new (key, value) pair if key is not existed.  if (map[index] == null) {  map[index] = new ArrayList<Pair<Integer, Integer>>();  }  map[index].add(new Pair(key, value));  } else {  // Update the value if key is existed.  map[index].set(pos, new Pair(key, value));  }  }    /\*\* Returns the value to which the specified key is mapped, or -1 if this map contains no mapping for the key \*/  public int get(int key) {  int index = getIndex(key);  int pos = getPos(key, index);  if (pos < 0) {  return -1;  } else {  return map[index].get(pos).getValue();  }  }    /\*\* Removes the mapping of the specified value key if this map contains a mapping for the key \*/  public void remove(int key) {  int index = getIndex(key);  int pos = getPos(key, index);  if (pos >= 0) {  map[index].remove(pos);  }  }  }  /\*\*  \* Your MyHashMap object will be instantiated and called as such:  \* MyHashMap obj = new MyHashMap();  \* obj.put(key,value);  \* int param\_2 = obj.get(key);  \* obj.remove(key);  \*/ |

*More*

Let's take a look at the operation "remove". After we find out the position of the element, we need to remove the element from the array list.

Let's assume that we are going to remove the ith element and the size of the array list is n.

The strategy used in the built-in function is to move all the elements after ith element one position forward. That is to say, you have to move n - i times. So the time complexity to remove an element from an array list will be O(n).

Consider different value of i. In average, we will move ((n - 1) + (n - 2) + ... + 1 + 0) / n = (n - 1) / 2 times.

Hopefully, there are two solutions to reduce the time complexity from O(n) to O(1).

**1. Swap**

There is a tricky strategy we can use. First, swap the element which we want to remove with the last element in the bucket. Then remove the last element. By this way, we successfully remove the element in O(1) time complexity.

**2. Linked List**

Another way to achieve this goal is to use a linked list instead of an array list. By this way, we can remove the element in O(1) time complexity without modifying the order in the list.