Hashmaps

```
public class MapNode<K, V> {
     K key;
     V value;
     MapNode<K, V> next;

     public MapNode(K key, V value) {
          this.key = key;
          this.value = value;
     }
}

public class MyHashmap<K, V> {
     ArrayList<MapNode<K, V>> bucket;
     int count;
     int sizeOfBucket;

     public MyHashmap() {
          bucket = new ArrayList<>();
          sizeOfBucket = 5;
          for (int i = 0; i < sizeOfBucket; i++) {
               bucket.add(null);
          }
     }

     // 0th helper method to get the index of array list
     private int getBucketIndex(K key) {
          int hc = key.hashCode();
          return hc % sizeOfBucket;
     }

     // 1st method
     public void insert(K key, V value) {
          int bucketIndex = getBucketIndex(key);
          MapNode<K, V> head = bucket.get(bucketIndex);
          // checking if an element is there if present then update
          while (head != null) {
               if (head.key.equals(key)) {
                    head.value = value;
                    return;
               }
               head = head.next;
          }
// if an element is not present, then insert it at 0th position in a LL
          head = bucket.get(bucketIndex);
          MapNode<K, V> newNode = new MapNode<>(key, value);
          newNode.next = head;
          bucket.set(bucketIndex, newNode);
          count++;
     //double loadFactor = (1.0 * count) / sizeOfBucket;
          if (loadFactor() >= 0.7) {
               rehash();
          }
     }
/* help to resize the bucket to make sure that there are minimum number of
elements in bucket Array we have rehash means resize the bucket array */

    private void rehash() {
        ArrayList<MapNode<K, V>> temp = bucket;
        bucket = new ArrayList<>();
        sizeOfBucket = sizeOfBucket * 2;
        count = 0;
        for (int i = 0; i < sizeOfBucket; i++) {
            bucket.add(null);
        }
        for (int i = 0; i < temp.size(); i++) {
            MapNode<K, V> head = temp.get(i);
            while (head != null) {
                K key = head.key;
                V value = head.value;
                insert(key, value);
                head = head.next;
            }
        }
    }

    // 2nd method
    public int size() {
        return count;
    }

    // 3rd get value
        public V getValue(K key) {
        int bucketIndex = getBucketIndex(key);
        MapNode<K, V> head = bucket.get(bucketIndex);
    // checking if an element is there if present then return its value
        while (head != null) {
            if (head.key.equals(key)) {
                return head.value;
            }
            head = head.next;
        }
        return null;
    }

    // 4th method
    public boolean isEmpty() {
        return count == 0;
    }

// calculating load factor
// count= number of entries & sizeOfBucket= size of the array
// we have to keep loadFactor <= 0.7
    public double loadFactor() {
        return (1.0 * count) / sizeOfBucket;
    }
}
```

```java
// 5th method remove key
public V removeKey(K key) {
     int bucketIndex = getBucketIndex(key);
     MapNode<K, V> head = bucket.get(bucketIndex);
     MapNode<K, V> previous = bucket.get(bucketIndex);
     while (head != null) {
          if (head.key.equals(key)) {
               if (previous != null) {
                    previous.next = head.next;
               }
               else {
                    bucket.set(bucketIndex, head.next);
               }
               count--;
               return head.value;
          }
          previous = head;
          head = head.next;
     }
     return null;
}
```