

HashMap Implementation

Java Code

```
import java.util.*;

public class HashMapCode {
    static class HashMap<K,V> { //generics
        private class Node {
            K key;
            V value;

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

        private int n; //n - nodes
        private int N; //N - buckets
        private LinkedList<Node> buckets[]; //N = buckets.length

        @SuppressWarnings("unchecked")
        public HashMap() {
            this.N = 4;
            this.buckets = new LinkedList[4];
            for(int i=0; i<4; i++) {
                this.buckets[i] = new LinkedList<>();
            }
        }

        private int hashFunction(K key) {
            int bi = key.hashCode();
            return Math.abs(bi) % N;
        }

        private int searchInLL(K key, int bi) {
            LinkedList<Node> ll = buckets[bi];
```

```

        for(int i=0; i<ll.size(); i++) {
            if(ll.get(i).key == key) {
                return i; //di
            }
        }

        return -1;
    }

    @SuppressWarnings("unchecked")
    private void rehash() {
        LinkedList<Node> oldBucket[] = buckets;
        buckets = new LinkedList[N*2];

        for(int i=0; i<N*2; i++) {
            buckets[i] = new LinkedList<>();
        }

        for(int i=0; i<oldBucket.length; i++) {
            LinkedList<Node> ll = oldBucket[i];
            for(int j=0; j<ll.size(); j++) {
                Node node = ll.get(j);
                put(node.key, node.value);
            }
        }
    }

    public void put(K key, V value) {
        int bi = hashFunction(key);
        int di = searchInLL(key, bi); //di = -1

        if(di == -1) { //key doesn't exist
            buckets[bi].add(new Node(key, value));
            n++;
        } else { //key exists
            Node node = buckets[bi].get(di);
            node.value = value;
        }
    }

    double lambda = (double)n/N;

```

```
        if(lambda > 2.0) {
            rehash();
        }
    }

    public boolean containsKey(K key) {
        int bi = hashFunction(key);
        int di = searchInLL(key, bi); //di = -1

        if(di == -1) { //key doesn't exist
            return false;
        } else { //key exists
            return true;
        }
    }

    public V remove(K key) {
        int bi = hashFunction(key);
        int di = searchInLL(key, bi); //di = -1

        if(di == -1) { //key doesn't exist
            return null;
        } else { //key exists
            Node node = buckets[bi].remove(di);
            n--;
            return node.value;
        }
    }

    public V get(K key) {
        int bi = hashFunction(key);
        int di = searchInLL(key, bi); //di = -1

        if(di == -1) { //key doesn't exist
            return null;
        } else { //key exists
            Node node = buckets[bi].get(di);
            return node.value;
        }
    }
}
```

```
public ArrayList<K> keySet() {
    ArrayList<K> keys = new ArrayList<>();

    for(int i=0; i<buckets.length; i++) { //bi
        LinkedList<Node> ll = buckets[i];
        for(int j=0; j<ll.size(); j++) { //di
            Node node = ll.get(j);
            keys.add(node.key);
        }
    }
    return keys;
}

public boolean isEmpty() {
    return n == 0;
}

}

public static void main(String args[]) {
    HashMap<String, Integer> map = new HashMap<>();
    map.put("India", 190);
    map.put("China", 200);
    map.put("US", 50);

    ArrayList<String> keys = map.keySet();
    for(int i=0; i<keys.size(); i++) {
        System.out.println(keys.get(i)+" "+map.get(keys.get(i)));
    }

    map.remove("India");
    System.out.println(map.get("India"));
}
}
```