GEBZE TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CSE222/505 – Spring 2021

Homework 5 Report

Yakup Talha Yolcu

1801042609

1. PART 1
    (1) PROBLEM SOLUTIONS APPROACH

In this part we are asked to write a custom iterator class that MapIterator iterator the HashMap keys. Firstly, I used the Java's HashMap class in java.util package. I have MyHashMap class that extends the HashMap class.

```
public class MyHashMap<K, V> extends HashMap<K,V> implements Iterable<K>{
```

MyHashMap class should implement the Iterable interface to use range for loop. I am using range for loop to test my MapIterator class. MyHashMap class have 2 methods and no data fields.

```
public MapIterator<K> iterator() {
    if(isEmpty()) {
        throw new NullPointerException("This map is empty");
    }
    return new MapIterator<>();
}
```

```
public MapIterator<K> iterator(K key) {
    if(isEmpty()) {
        throw new NullPointerException("This Map is empty");
    }
    return new MapIterator<>(key);
}
```

As it needed, we should be able to start iteration with given key value. This methods returns an iterator to iterate through the HashMap.

MapIterator is inner class of MyHashMap class.

```
public class MapIterator <K> implements Iterator<K> {
```

I've made it public because I wanted to use it in main function.

MapIterator has 4 data fields that are :

```
/** values to iterate through */
private K[] kSet;
/** cursor */
private int cur=-1;
/** last key returned */
private K lastItemreturned=null;
/** represents the iterated elements */
private boolean[] iterated;
```

kSet array holds the current HashMaps elements. I designed the Iterator so that when it is created, insertions and removals do not have effect on the iterator.

Int cur holds the current index that iterator will be iterate.

lastItemreturned represents the last Item returned by the iterator.

Boolean iterated array keeps the elements are iterated or not information.

```java
public MapIterator(){
    try {
        check_for_empty();
        iterated=new boolean[kSet.length];
    }
    catch (NullPointerException ne) {
        System.out.println("This map is empty");
    }
}
```

No parameter constructor checks whether hashmap is empty or not. If it is empty, Null Pointer Exception will be thrown. Otherwise needed data field adjustments will be done.

```java
/** key parameter constructor that starts the iterator from given key ...*/
public MapIterator(K key){
    try {
        check_for_empty();

    }
    catch (NullPointerException ne) {
        System.out.println("This map is empty");
    }
    int i=0;
    /*...*/
    if(containsKey(key)) {
        /*...*/
        for(K key_temp:kSet) {
            if(key_temp.equals(key)) {
                break;
            }
            i++;
        }
        /*...*/
        iterated=new boolean[kSet.length];
        cur+=i;
        lastItemreturned=kSet[cur];
    }
}
```

Key parameter constructor takes a key as a parameter and checks whether hashmap is empty or not. If it is empty, Null Pointer Exception will be thrown. Otherwise needed data field adjustments will be done.

Firstly if hashmap contains the given key, it starts from that key. At first next(), this key will be returned. Otherwise (if it does not contain the key) It starts from the first key as if no parameter constructor.

```java
@SuppressWarnings("unchecked")
private void check_for_empty() throws NullPointerException{
    if(MyHashMap.super.isEmpty()) {
        throw new NullPointerException();
    }
    Set<K> temp_set= (Set<K>) keySet();
    if(temp_set.isEmpty()) {
        throw new NullPointerException();
    }
    kSet = (K[])temp_set.toArray();
}
```

Check for empty method checks whether hashmap is empty or not. If it is empty, NullPointerExcepiton will be thrown. Otherwise **kSet array** initialized. I retrieve the hashmap's data here. First I created a temporary set. I get the data from the hashmap by using hashmap's keyset method. Then I used the Set's toArray method.

```java
public boolean hasNext() {
    for(int i=0;i<iterated.length;i++) {
        if(!iterated[i]) {
            return true;
        }
    }
    return false;
}
```

hasNext method The method returns True if there are still not-iterated key/s in the Map, otherwise returns False.

```
public K next() {
    if(!hasNext()) {
        return kSet[0];
    }
    if(cur>=kSet.length-1) {
        cur=-1;
    }
    iterated[cur+1]=true;
    lastItemreturned=kSet[++cur];
    return lastItemreturned;
}
```

Next method The function returns the next key in the Map. It returns the first key when there is no not-iterated key in the Map.

```
public K prev() {
    if(cur==-1) {
        cur=kSet.length-1;
    }
    iterated[cur]=true;
    lastItemreturned=kSet[cur--];
    return lastItemreturned;
}
```

prev method The iterator points to the previous key in the Map. It returns the last key when the iterator is at the first key.

(2) TEST CASES

TEST CASE	TEST SCENARIO	RESULT
Hashmap is empty	HashMap does not have any element	Null Pointer Exception will be thrown
HashMap has element	HashMap's elements are iterated	HashMap's elements are iterated successfully

(3) RUNNING COMMANDS AND RESULTS

```
TESTING WITH INPUT SIZE 0

java.lang.NullPointerException: This map is empty
TESTING WITH INPUT SIZE 20

ITERATION WITH RANGE FOR LOOP

3392 ,4961 ,578 ,1538 ,3042 ,1769 ,2282 ,3052 ,2128 ,112 ,1968 ,4401 ,2547 ,2292 ,2902 ,2041 ,347 ,830 ,3935 ,2591 ,
------
```

```
ITERATION WITH NEXT
3392 ,4961 ,578 ,1538 ,3042 ,1769 ,2282 ,3052 ,2128 ,112 ,1968 ,4401 ,2547 ,2292 ,2902 ,2041 ,347 ,830 ,3935 ,2591 ,
COMPARING ELEMENTS AFTER ITERATION WITH RANGE FOR LOOP AND ITERATION WITH NEXT
ALL ELEMENTS ARE SAME, TEST PASSED


------
```

```
ITERATION WITH PREVIOUS

2591 ,3935 ,830 ,347 ,2041 ,2902 ,2292 ,2547 ,4401 ,1968 ,112 ,2128 ,3052 ,2282 ,1769 ,3042 ,1538 ,578 ,4961 ,3392 ,
COMPARING ELEMENTS AFTER ITERATION WITH PREVIOUS AND ITERATION WITH NEXT
ALL ELEMENTS ARE SAME, TEST PASSED

START VALUE OF ITERATIONS WITH START VALUE: 830

------

ITERATION WITH NEXT

830 ,3935 ,2591 ,3392 ,4961 ,578 ,1538 ,3042 ,1769 ,2282 ,3052 ,2128 ,112 ,1968 ,4401 ,2547 ,2292 ,2902 ,2041 ,347 ,
------

ITERATION WITH PREVIOUS

347 ,2041 ,2902 ,2292 ,2547 ,4401 ,1968 ,112 ,2128 ,3052 ,2282 ,1769 ,3042 ,1538 ,578 ,4961 ,3392 ,2591 ,3935 ,830 ,
COMPARING ELEMENTS AFTER ITERATION WITH START VALUE KEY NEXT AND START VALUE KEY PREVIOUS
ALL ELEMENTS ARE SAME, TEST PASSED

INPUT SIZE 20 TEST PASSED, INPUT SIZE WILL BE INCREMENTED
```

Other results can be seen in the terminal when program is executed. I tested the input size 100 and 500 and they passed.

2. PART 2

- **PROBLEM SOLUTION APPROACH**

  In this part we are expected to have HashMap as Chain, TreeSet Chain and Quadratic Probing forms. In Chain we are using LinkedList array to hold elements. Load threshold=3.0 in this class. In case of collusion, we have LinkedList to hold all collided elements. At that index we are searching the linkedlist to find the element. As in chain method, we are using same technique in TreeSet mode but only difference is we are holding elements in a TreeSet instead of LinkedList. To do that, all key values and Entry class should be comparable to storage elements in TreeSet.

  In Quadratic Probing, we are holding a next index to trace of collided elements. We are trying to find an empty place from key.hashcode%length+1 to key.hashcode%length+n^2 n=>iteration number.

  When removing an element if this element has an index, we should replace it with next element and remove that element from its previous place. I've solve this problem recursively. I did this process until element does not have any next element.

  If load factor is reached or exceeded, we need to rehash the table by increasing the size 2*n+1. Load factor is 0.75 in this case.

```java
private int find(Object key) {
    int index=key.hashCode()%table.length;
    if(index<0) {
        index+=table.length;
    }
    while ( (table[index] != null) && (!key.equals(table[index].key))) {
        if(table[index].next<0) {
            index=find_quadratic(key,index);
        }
        else {
            index=table[index].next;
        }
    }
    return index;
}
```

```java
private int find_quadratic(Object key,int index) {
    int next=index;
    int i=0;
    while (table[index] != null && !table[index].equals(key)) {
        index=(key.hashCode()%table.length)+(int)Math.pow(i+1,2);
        if(index>=table.length) {
            index%=table.length;
        }
        while(index<0) {
            index+=table.length;
        }
        i+=1;
    }
    table[next].next=index;
    return index;
}
```

- **TEST CASES**

TEST CASE	TEST SCENARIO	RESULT
Hashmap is filling	Put method call	Elements added successfully
Removing element from hashmap	Remove method call	Element is removed successfully
		Element couldn't removed

TESTING WITH DIFFERENT TYPES

```
KWHashMapChain<String,Integer> kwHashMapChain=new KWHashMapChain<>( input_size: 15);
KWHashMapChainTreeSet<String,Integer> kwHashMapChainTreeSet=new KWHashMapChainTreeSet<>( input_size: 15);
KWHashMapCoalesced<String,Integer> kwHashMapCoalesced=new KWHashMapCoalesced<>( input_size: 15);

System.out.println("\n\nTESTING DIFFERENT TYPES\n\nCHAIN:");
test_string(kwHashMapChain);
System.out.println("TREESET:");
test_string(kwHashMapChainTreeSet);
System.out.println("COALESCED:");
test_string(kwHashMapCoalesced);
```

```
public static void test_string(KWHashMap<String,Integer> map) {
    map.put("CSE232",90);
    map.put("CSE222",80);
    map.put("CSE101",85);
    map.put("CSE102",45);
    map.put("CSE108",90);
    map.put("CSE107",89);

    System.out.println("PRINTING MAP:\n"+map);

}
```

TESTING WITH LOW SIZE

```
    KWHashMapChain<Integer,Integer> kwHashMapChain=new KWHashMapChain<>( input_size: 15);
    KWHashMapChainTreeSet<Integer,Integer> kwHashMapChainTreeSet=new KWHashMapChainTreeSet<>( input_size: 15);
    KWHashMapCoalesced<Integer,Integer> kwHashMapCoalesced=new KWHashMapCoalesced<>( input_size: 15);

    System.out.println("FIRSTLY LOW INPUT SIZE TO SEE EASILY");

    for(int i=0;i<30;i++) {
        kwHashMapChain.put(i,i+1);
        kwHashMapChainTreeSet.put(i,i+1);
        kwHashMapCoalesced.put(i,i+1);
    }
    System.out.println("\n\nCHAIN:\n");
    System.out.println(kwHashMapChain);
    System.out.println("\n\nTREESET:\n");
    System.out.println(kwHashMapChainTreeSet);
    System.out.println("\n\nCOALESCED:\n");
    System.out.println(kwHashMapCoalesced);

    System.out.println("REMOVING 30 NUMBERS");
    for(int i=0;i<30;i++) {
        kwHashMapChain.remove(i);
        kwHashMapChainTreeSet.remove(i);
        kwHashMapCoalesced.remove(i);
    }
```

TESTING HIGH SIZE

```
System.out.println("ADDING 5K NUMBERS");

for(int i=30;i<5030;i++) {
    kwHashMapChain.put(i,i+1);
    kwHashMapChainTreeSet.put(i,i+1);
    kwHashMapCoalesced.put(i,i+1);
}

System.out.println("REMOVING 5K NUMBERS");
for(int i=30;i<5030;i++) {
    kwHashMapChain.remove(i);
    kwHashMapChainTreeSet.remove(i);
    kwHashMapCoalesced.remove(i);
}
```

TESTING PDF EXAMPLE

```
public static void test_coalesced_pdf_example() {
    KWHashMapCoalesced<Integer,Integer> kwchcoalesced=new KWHashMapCoalesced<>( input_size: 10);

    int[] input={3, 12, 13, 25, 23, 51, 42};

    for(int i:input) {
        kwchcoalesced.put(i,i);
    }
    System.out.println("\nBEFORE REMOVING 13\n");
    System.out.println(kwchcoalesced);
    kwchcoalesced.remove( key: 13);
    System.out.println("\nAFTER REMOVING 13\n");
    System.out.println(kwchcoalesced);
}
```

- **RUNNING COMMANDS AND RESULTS**

```java
KWHashMapChain<Integer,Integer> kwch=new KWHashMapChain<>( input_size: 10);
KWHashMapChainTreeSet<Integer,Integer> kwchtreeset=new KWHashMapChainTreeSet<>( input_size: 10);
KWHashMapCoalesced<Integer,Integer> kwchcoalesced=new KWHashMapCoalesced<>( input_size: 10);

int[] input={3, 12, 13, 25, 23, 51, 42};

for(int i:input) {
    kwchcoalesced.put(i,i);
    kwch.put(i,i);
    kwchtreeset.put(i,i);
}
System.out.println(kwch);

System.out.println(kwchtreeset);

System.out.println(kwchcoalesced);
```

INDEX	VALUE
0	null
1	[51]
2	[42, 12]
3	[23, 13, 3]
4	null
5	[25]
6	null
7	null
8	null
9	null

INDEX	VALUE
0	null
1	[51]
2	[12, 42]
3	[3, 13, 23]
4	null
5	[25]
6	null
7	null
8	null
9	null

INDEX	VALUE	NEXT
0	null	
1	51	NULL
2	12	6
3	3	4
4	13	7
5	25	NULL
6	42	NULL
7	23	NULL
8	null	
9	null	

```
Duration for adding 100 elements in HashTableChain:: 0.1717 ms
Duration for adding 100 elements in HashTableTreeSetChain:: 0.2173 ms
Duration for adding 100 elements in HashTableCoalesced:: 0.079 ms

Duration for Get() 100 elements in HashTableChain:: 0.1389 ms
Duration for Get() 100 elements in HashTableTreeSetChain:: 0.1135 ms
Duration for Get() 100 elements in HashTableCoalesced:: 0.0743 ms

Duration for Removing 100 elements in HashTableChain:: 0.1924 ms
Duration for Removing 100 elements in HashTableTreeSetChain:: 0.1842 ms
Duration for Removing 100 elements in HashTableCoalesced:: 0.8219 ms

Duration for Removing NOT EXIST 100 elements in HashTableChain:: 0.092 ms
Duration for Removing NOT EXIST 100 elements in HashTableTreeSetChain:: 0.092 ms
Duration for Removing NOT EXIST 100 elements in HashTableCoalesced:: 0.1714 ms


88 PUT SUCCESSFUL 88
TEST PASSED
162 PUT SUCCESSFUL 162
TEST PASSED
283 PUT SUCCESSFUL 283
TEST PASSED
399 PUT SUCCESSFUL 399
TEST PASSED
```

TESTING WITH DIFFERENT TYPES

CHAIN:
PRINTING MAP:

INDEX	VALUE
0	null
1	null
2	null
3	null
4	null
5	[[CSE101,85]]
6	[[CSE102,45]]
7	null
8	null
9	[[CSE222,80]]
10	[[CSE232,90]]
11	[[CSE107,89]]
12	[[CSE108,90]]
13	null
14	null

TREESET:
PRINTING MAP:

INDEX	VALUE
0	null
1	null
2	null
3	null
4	null
5	[[CSE101,85]]
6	[[CSE102,45]]
7	null
8	null
9	[[CSE222,80]]
10	[[CSE232,90]]
11	[[CSE107,89]]
12	[[CSE108,90]]
13	null
14	null

COALESCED:
PRINTING MAP:

INDEX	VALUE	NEXT
0	null	
1	null	
2	null	
3	null	
4	null	
5	CSE101	NULL
6	CSE102	NULL
7	null	
8	null	
9	CSE222	NULL
10	CSE232	NULL
11	CSE107	NULL
12	CSE108	NULL
13	null	
14	null	

TESTING WITH LOW INPUT SIZE

CHAIN:

INDEX	VALUE
0	[[15,16], [0,1]]
1	[[16,17], [1,2]]
2	[[17,18], [2,3]]
3	[[18,19], [3,4]]
4	[[19,20], [4,5]]
5	[[20,21], [5,6]]
6	[[21,22], [6,7]]
7	[[22,23], [7,8]]
8	[[23,24], [8,9]]
9	[[24,25], [9,10]]
10	[[25,26], [10,11]]
11	[[26,27], [11,12]]
12	[[27,28], [12,13]]
13	[[28,29], [13,14]]
14	[[29,30], [14,15]]

TREESET:

INDEX	VALUE
0	[[0,1], [15,16]]
1	[[1,2], [16,17]]
2	[[2,3], [17,18]]
3	[[3,4], [18,19]]
4	[[4,5], [19,20]]
5	[[5,6], [20,21]]
6	[[6,7], [21,22]]
7	[[7,8], [22,23]]
8	[[8,9], [23,24]]
9	[[9,10], [24,25]]
10	[[10,11], [25,26]]
11	[[11,12], [26,27]]
12	[[12,13], [27,28]]
13	[[13,14], [28,29]]
14	[[14,15], [29,30]]

COALESCED:

INDEX	VALUE	NEXT
0	0	NULL
1	1	NULL
2	2	NULL
3	3	NULL
4	4	NULL
5	5	NULL
6	6	NULL
7	7	NULL
8	8	NULL
9	9	NULL
10	10	NULL
11	11	NULL
12	12	NULL
13	13	NULL
14	14	NULL
15	15	NULL
16	16	NULL
17	17	NULL
18	18	NULL
19	19	NULL
20	20	NULL
21	21	NULL
22	22	NULL