GIT Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework 5 Report

SERDİL ANIL ÜNLÜ
1801042672

1-PROBLEM SOLUTIONS APPROACH:

PART1:

- I extend the HashMap in java and I create the MapIterator class as inner class in MyHashMap class.
- I have written all the necessary methods inside this MapIterator class. (Except MapIterator iterator() and MapIterator iterator(K key)).
- And I used above two iterators to iterate with zero constructors and constructors for last task of part 1 .
- In the hasNext() method, it returns true until the length of the Kvalue value is over.
- I create the getIndex method to compare the key and index.
- I can use the Next() method with both the hasNext() method and the iterator to navigate all values.
- With the prev() method, I print all previous values by going backward step by step.
- I have defined the MapIterator iterators outside of the MapIterator class and created two ways: Normal constructor and zero constructor.

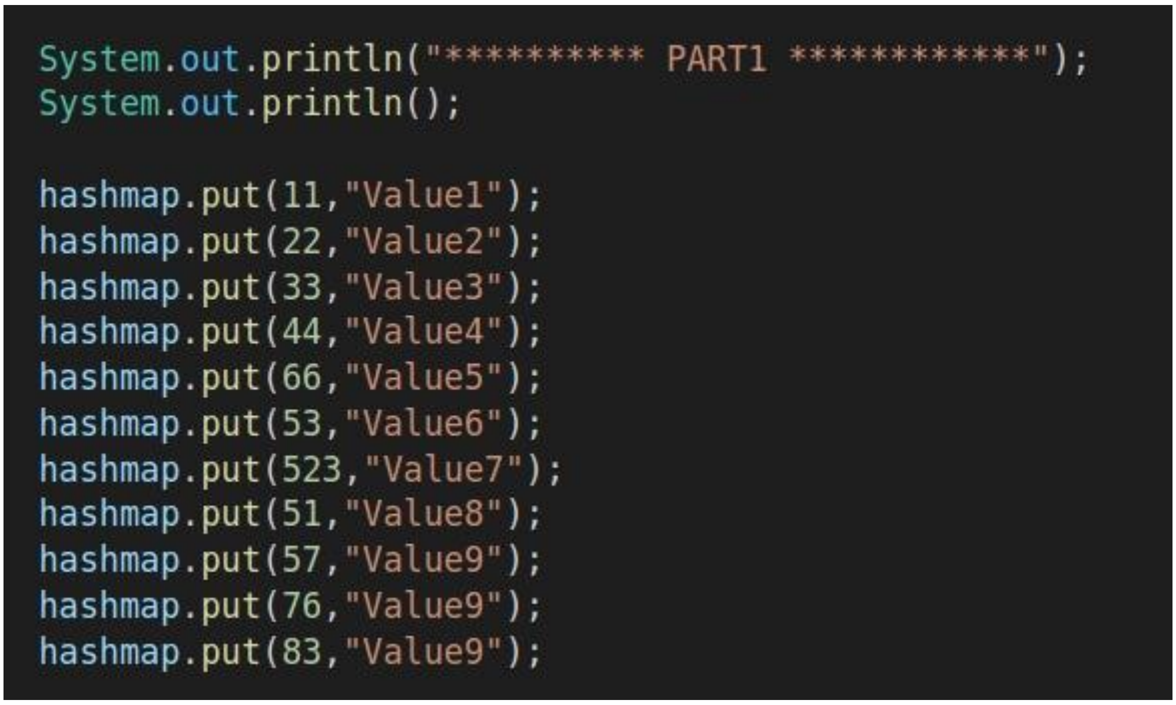
PART2:

- For the tasks in Part2, I first implemented the KWHashMap class in all classes that I will create for this part.
- First of all, I set up HashTableChain class to use the hashing with linked list and I set the Entry class inside this class. I wrote the required setter and getter methods of key and value into the entry class. I applied the Chaining technique together with the put and remove methods.
- While I was applying the Chaining technique to TreeSet hashing, I created a class named HashtableChainTreeSet,

different from the one I did on the linked list. In the class I created for a linked list in this class, I typed TreeSet where there is a linked list. I changed the places where I used the addfirst method with the add method. Finally,I wrote the compareTo method for the Entry class.

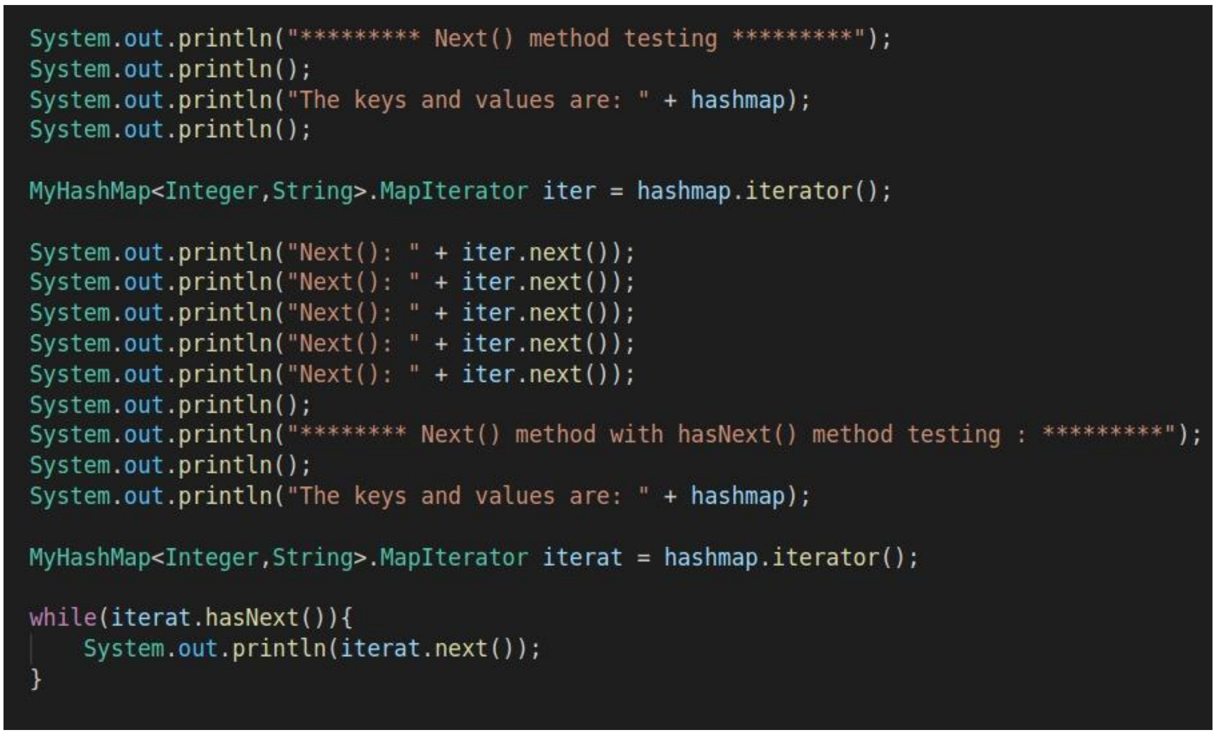
- I created the CoalescedHash class for the last batch of part2. I have used open addressing for the Hash table implementation in this class. I used the hashCode method for the put function . If the table is 0 and the loadfactor is greater than threshold, I used the rehash method and quadratically increased the probe when adding.

2- TEST CASES:



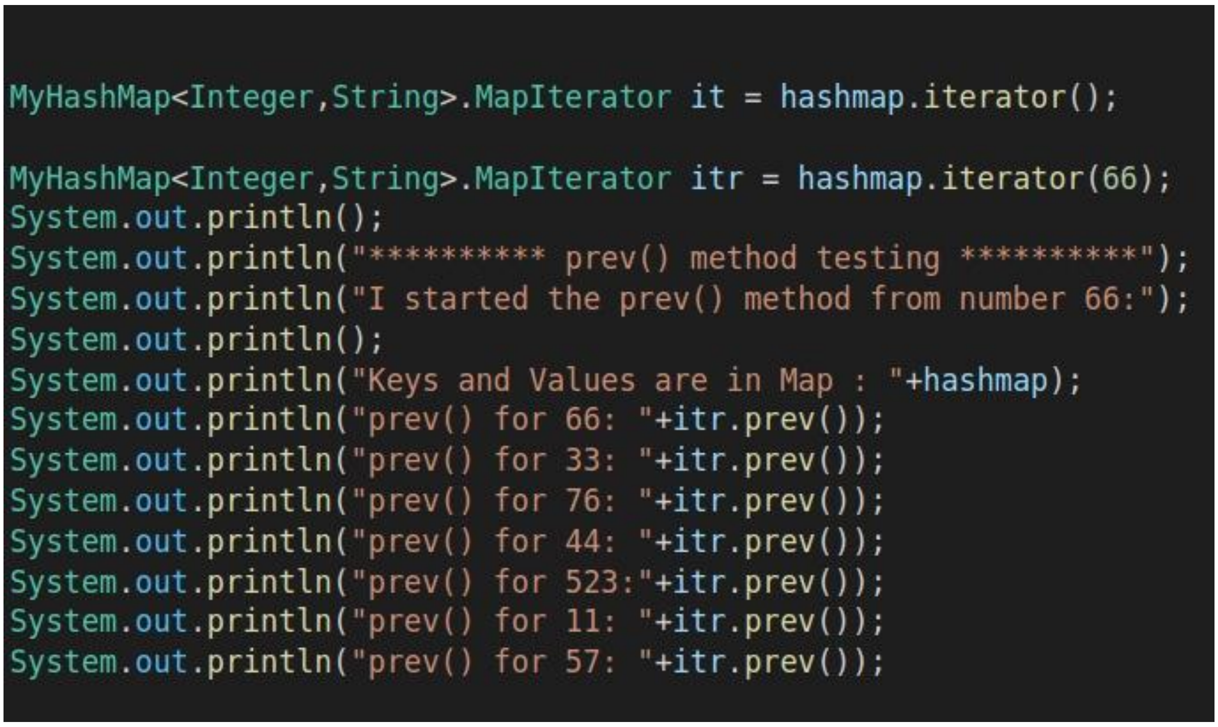
```
System.out.println("********** PART1 ************");
System.out.println();

hashmap.put(11,"Value1");
hashmap.put(22,"Value2");
hashmap.put(33,"Value3");
hashmap.put(44,"Value4");
hashmap.put(66,"Value5");
hashmap.put(53,"Value6");
hashmap.put(523,"Value7");
hashmap.put(51,"Value8");
hashmap.put(57,"Value9");
hashmap.put(76,"Value9");
hashmap.put(83,"Value9");
```

Here, I add elements to hashmap via put function to try part1.



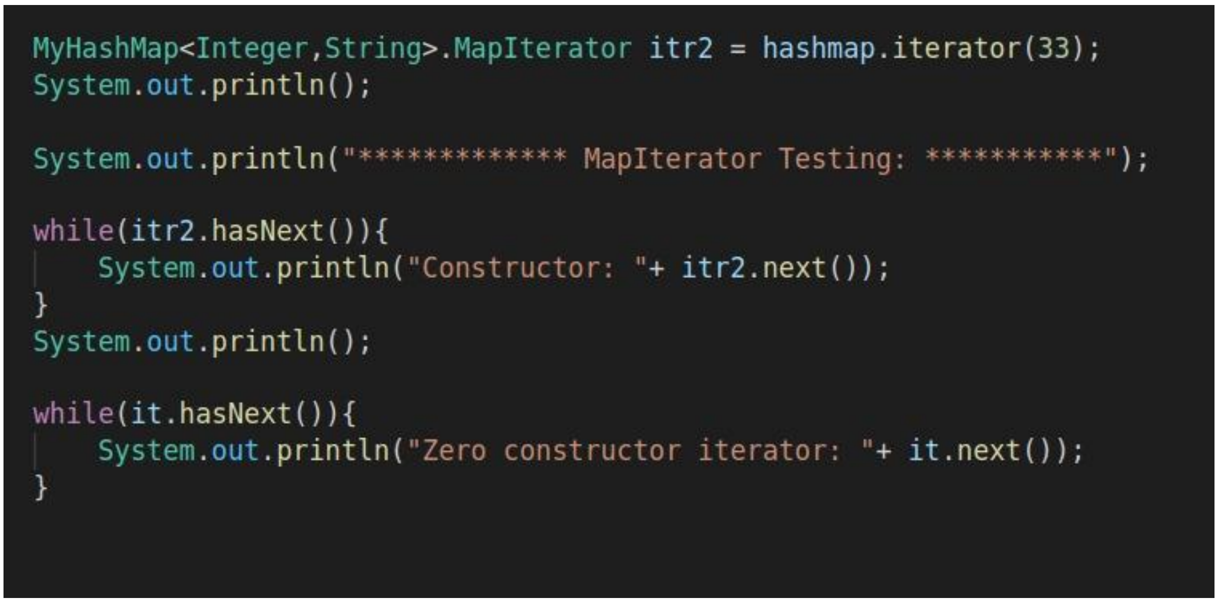
```java
System.out.println("********* Next() method testing *********");
System.out.println();
System.out.println("The keys and values are: " + hashmap);
System.out.println();

MyHashMap<Integer,String>.MapIterator iter = hashmap.iterator();

System.out.println("Next(): " + iter.next());
System.out.println("Next(): " + iter.next());
System.out.println("Next(): " + iter.next());
System.out.println("Next(): " + iter.next());
System.out.println("Next(): " + iter.next());
System.out.println();
System.out.println("******** Next() method with hasNext() method testing : *********");
System.out.println();
System.out.println("The keys and values are: " + hashmap);

MyHashMap<Integer,String>.MapIterator iterat = hashmap.iterator();

while(iterat.hasNext()){
    System.out.println(iterat.next());
}
```

After adding an element to the hashmap here, I navigate the next() method with both the hasNext method and the elements in the map with the iterator.



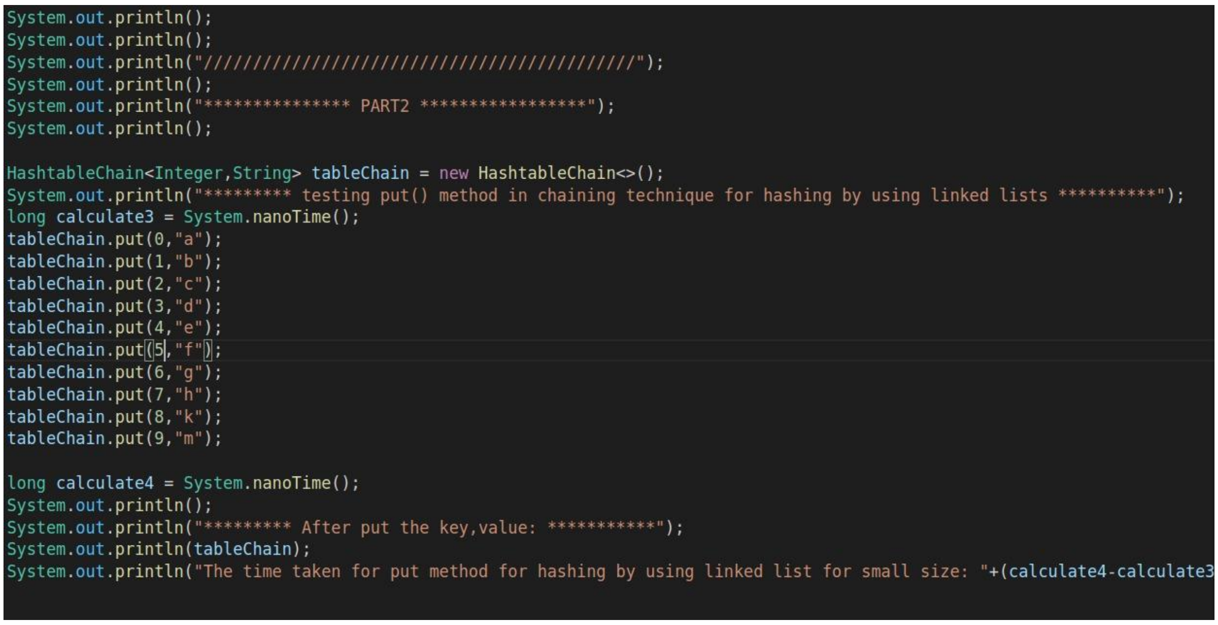
```java
MyHashMap<Integer,String>.MapIterator it = hashmap.iterator();

MyHashMap<Integer,String>.MapIterator itr = hashmap.iterator(66);
System.out.println();
System.out.println("********** prev() method testing **********");
System.out.println("I started the prev() method from number 66:");
System.out.println();
System.out.println("Keys and Values are in Map : "+hashmap);
System.out.println("prev() for 66: "+itr.prev());
System.out.println("prev() for 33: "+itr.prev());
System.out.println("prev() for 76: "+itr.prev());
System.out.println("prev() for 44: "+itr.prev());
System.out.println("prev() for 523:"+itr.prev());
System.out.println("prev() for 11: "+itr.prev());
System.out.println("prev() for 57: "+itr.prev());
```

I also use the prev method on the elements I added, I go back to the beginning and go back to the last elements.



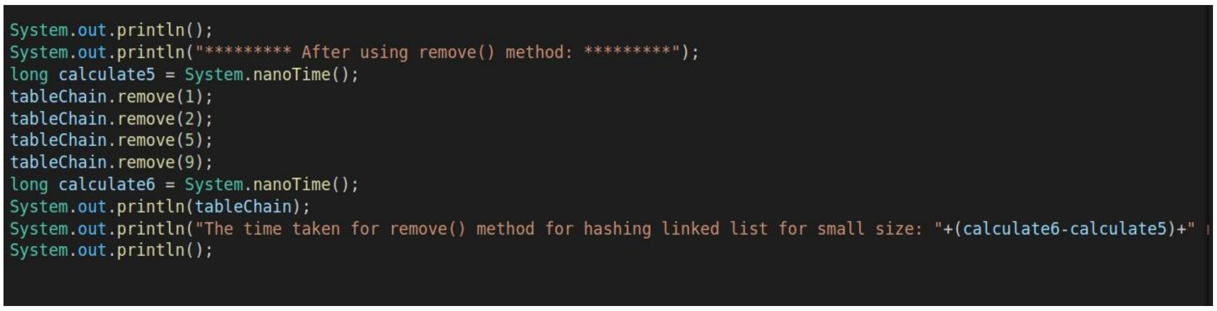
```java
MyHashMap<Integer,String>.MapIterator itr2 = hashmap.iterator(33);
System.out.println();

System.out.println("************* MapIterator Testing: ***********");

while(itr2.hasNext()){
    System.out.println("Constructor: "+ itr2.next());
}
System.out.println();

while(it.hasNext()){
    System.out.println("Zero constructor iterator: "+ it.next());
}
```

Thanks to zero constructors and constructors, I use MapIterator and navigate elements.



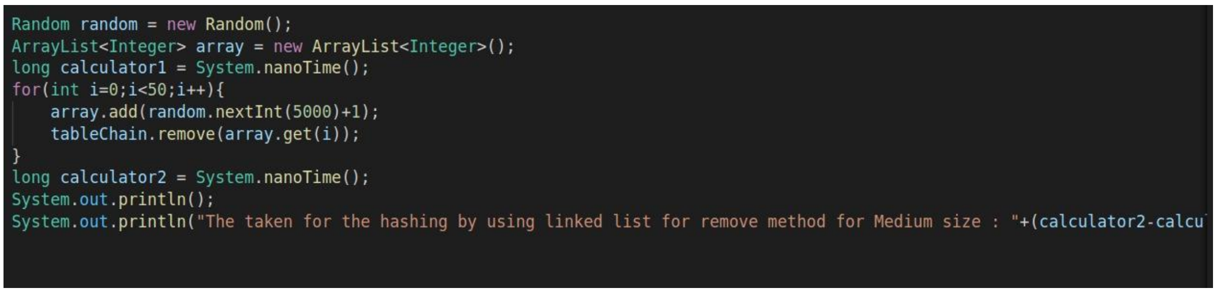
```java
System.out.println();
System.out.println();
System.out.println("////////////////////////////////////////////");
System.out.println();
System.out.println("*************** PART2 *****************");
System.out.println();

HashtableChain<Integer,String> tableChain = new HashtableChain<>();
System.out.println("********* testing put() method in chaining technique for hashing by using linked lists **********");
long calculate3 = System.nanoTime();
tableChain.put(0,"a");
tableChain.put(1,"b");
tableChain.put(2,"c");
tableChain.put(3,"d");
tableChain.put(4,"e");
tableChain.put(5,"f");
tableChain.put(6,"g");
tableChain.put(7,"h");
tableChain.put(8,"k");
tableChain.put(9,"m");

long calculate4 = System.nanoTime();
System.out.println();
System.out.println("********* After put the key,value: ***********");
System.out.println(tableChain);
System.out.println("The time taken for put method for hashing by using linked list for small size: "+(calculate4-calculate3
```

I’m adding element to the tableChain for the 1st part of part2. In addition, I take 10 elements I added in small size and measure their performance.



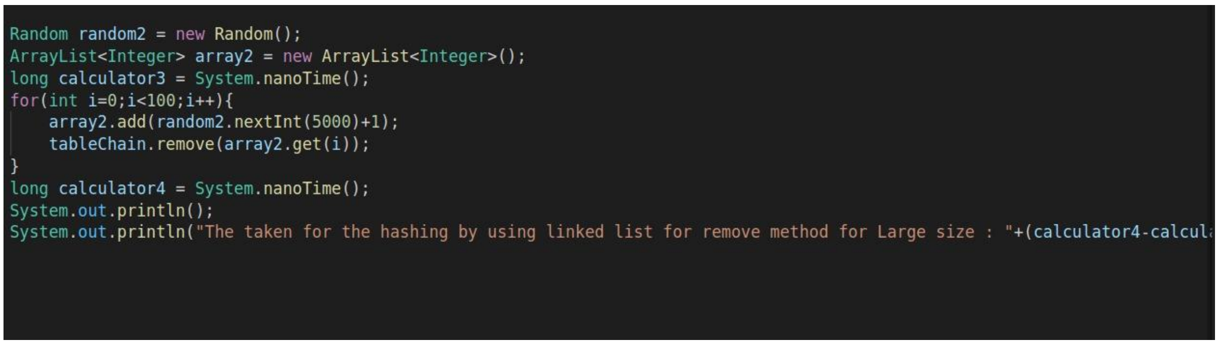
```java
System.out.println();
System.out.println("********* After using remove() method: *********");
long calculate5 = System.nanoTime();
tableChain.remove(1);
tableChain.remove(2);
tableChain.remove(5);
tableChain.remove(9);
long calculate6 = System.nanoTime();
System.out.println(tableChain);
System.out.println("The time taken for remove() method for hashing linked list for small size: "+(calculate6-calculate5)+"
System.out.println();
```

Here, too, I delete some of the elements I added to the tableChain and measure the time it takes, again in small size.



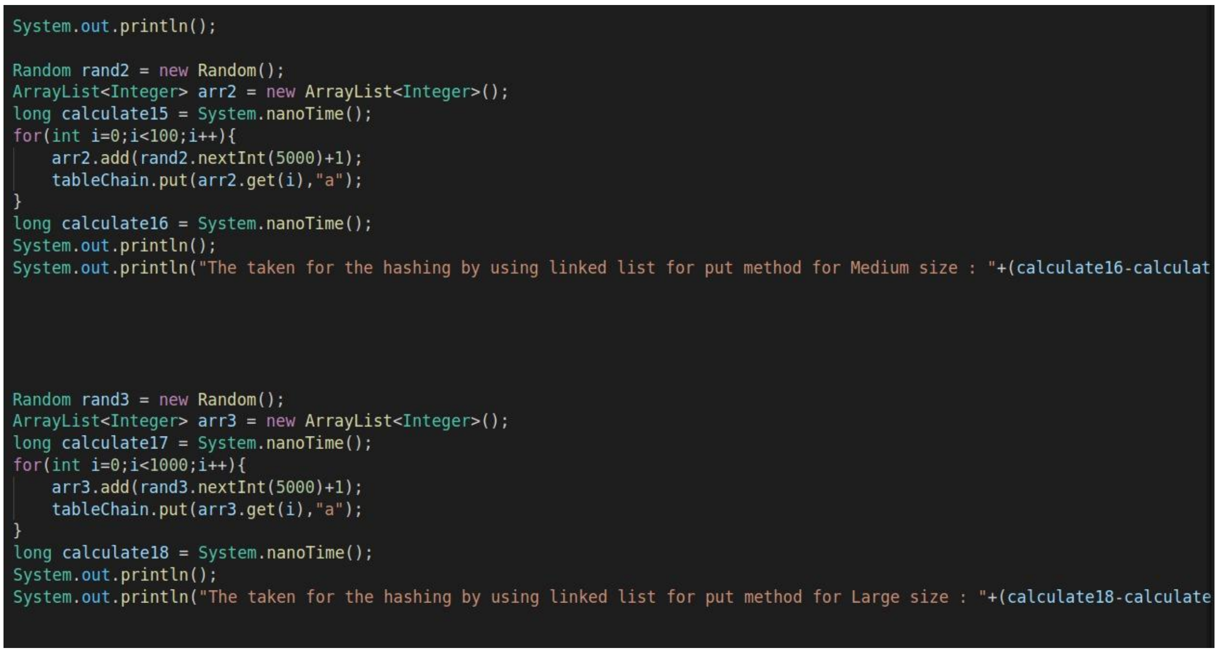
```java
Random random = new Random();
ArrayList<Integer> array = new ArrayList<Integer>();
long calculator1 = System.nanoTime();
for(int i=0;i<50;i++){
    array.add(random.nextInt(5000)+1);
    tableChain.remove(array.get(i));
}
long calculator2 = System.nanoTime();
System.out.println();
System.out.println("The taken for the hashing by using linked list for remove method for Medium size : "+(calculator2-calcu
```

Here , I am deleting the tableChain (linked list) medium size.



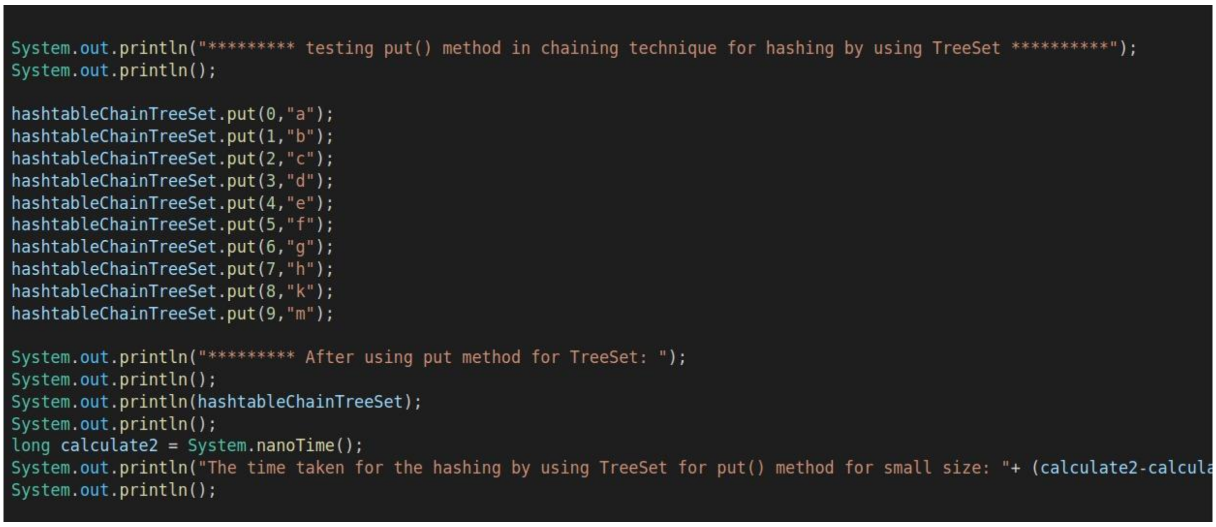
```java
Random random2 = new Random();
ArrayList<Integer> array2 = new ArrayList<Integer>();
long calculator3 = System.nanoTime();
for(int i=0;i<100;i++){
    array2.add(random2.nextInt(5000)+1);
    tableChain.remove(array2.get(i));
}
long calculator4 = System.nanoTime();
System.out.println();
System.out.println("The taken for the hashing by using linked list for remove method for Large size : "+(calculator4-calcul
```

Here, I am deleting the tableChain (linked list) large size.

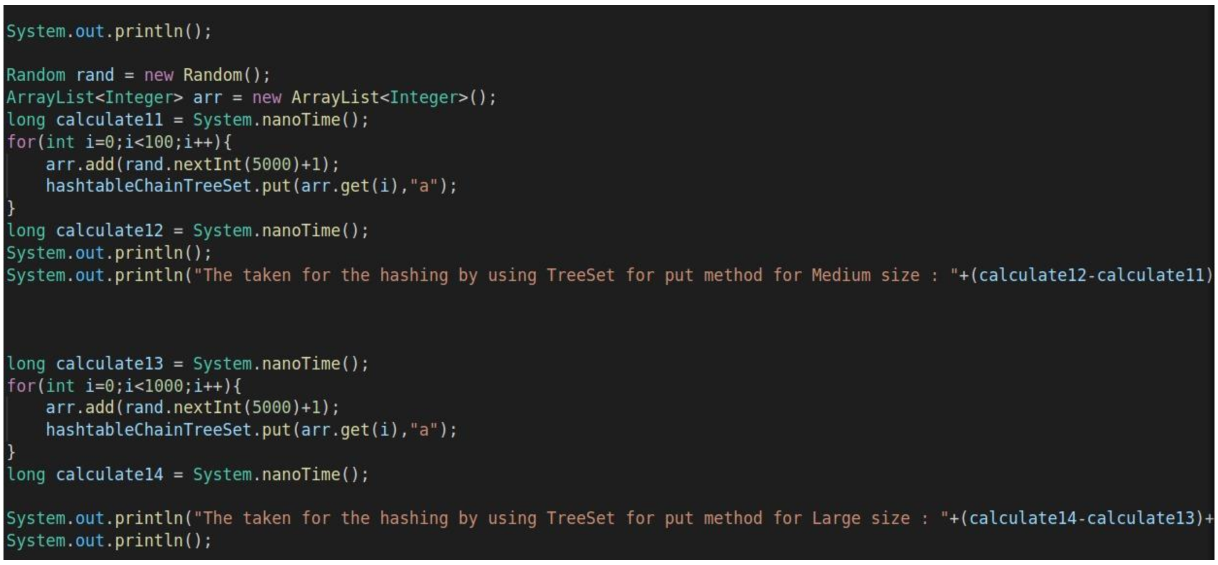


```
System.out.println();

Random rand2 = new Random();
ArrayList<Integer> arr2 = new ArrayList<Integer>();
long calculate15 = System.nanoTime();
for(int i=0;i<100;i++){
    arr2.add(rand2.nextInt(5000)+1);
    tableChain.put(arr2.get(i),"a");
}
long calculate16 = System.nanoTime();
System.out.println();
System.out.println("The taken for the hashing by using linked list for put method for Medium size : "+(calculate16-calculat



Random rand3 = new Random();
ArrayList<Integer> arr3 = new ArrayList<Integer>();
long calculate17 = System.nanoTime();
for(int i=0;i<1000;i++){
    arr3.add(rand3.nextInt(5000)+1);
    tableChain.put(arr3.get(i),"a");
}
long calculate18 = System.nanoTime();
System.out.println();
System.out.println("The taken for the hashing by using linked list for put method for Large size : "+(calculate18-calculate
```

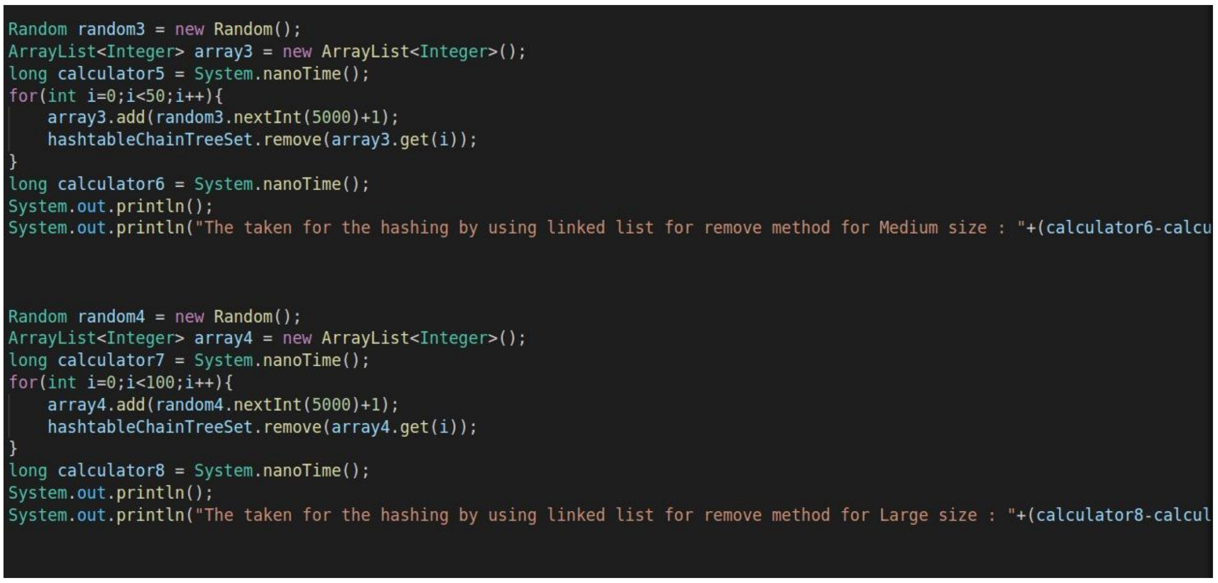
For the put method in tableChain, I also generated a random number and measured the time taken by the medium and large sizes as well as the small size.



```
System.out.println("********* testing put() method in chaining technique for hashing by using TreeSet **********");
System.out.println();

hashtableChainTreeSet.put(0,"a");
hashtableChainTreeSet.put(1,"b");
hashtableChainTreeSet.put(2,"c");
hashtableChainTreeSet.put(3,"d");
hashtableChainTreeSet.put(4,"e");
hashtableChainTreeSet.put(5,"f");
hashtableChainTreeSet.put(6,"g");
hashtableChainTreeSet.put(7,"h");
hashtableChainTreeSet.put(8,"k");
hashtableChainTreeSet.put(9,"m");

System.out.println("********* After using put method for TreeSet: ");
System.out.println();
System.out.println(hashtableChainTreeSet);
System.out.println();
long calculate2 = System.nanoTime();
System.out.println("The time taken for the hashing by using TreeSet for put() method for small size: "+ (calculate2-calcula
System.out.println();
```

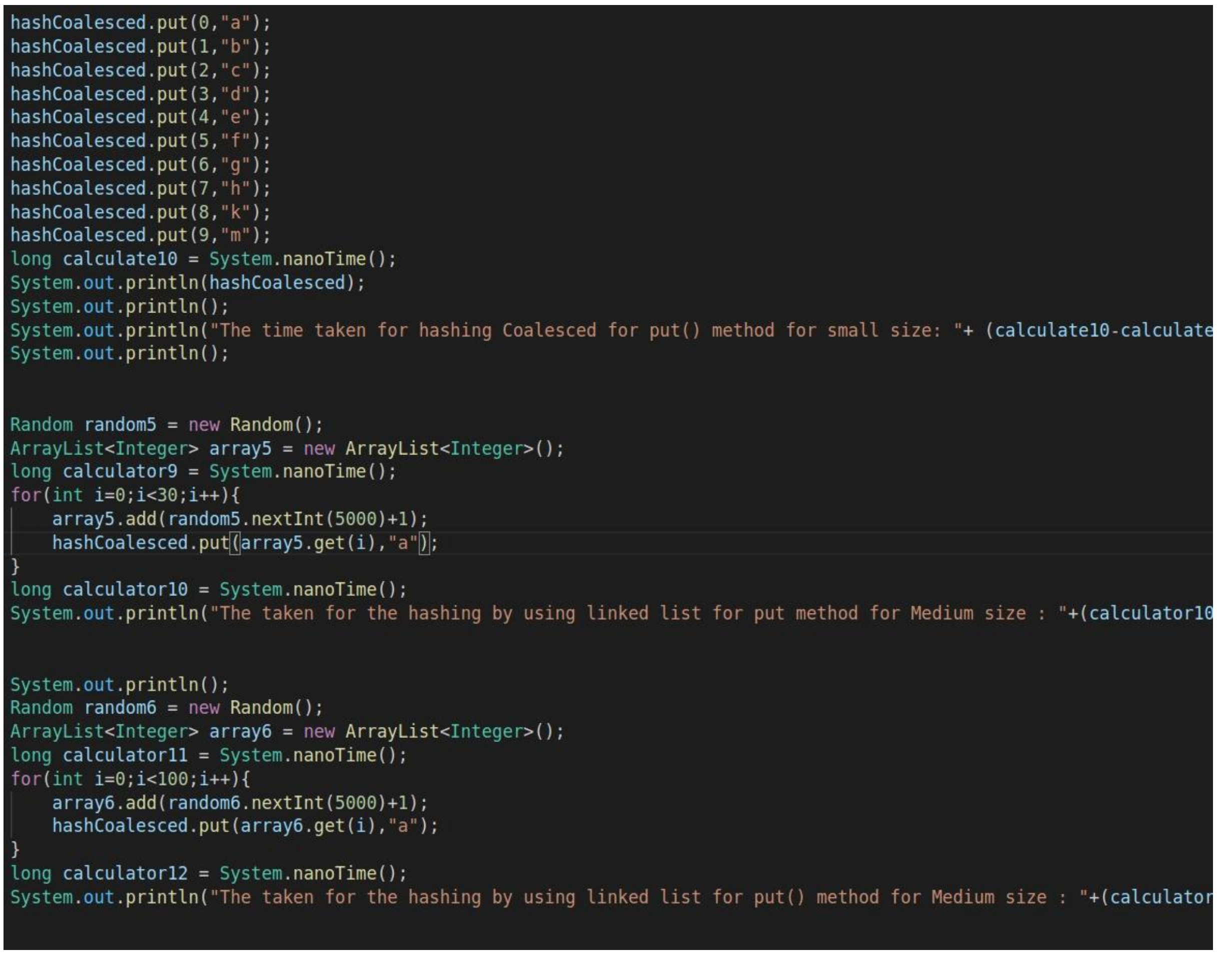


```
System.out.println();

Random rand = new Random();
ArrayList<Integer> arr = new ArrayList<Integer>();
long calculate11 = System.nanoTime();
for(int i=0;i<100;i++){
    arr.add(rand.nextInt(5000)+1);
    hashtableChainTreeSet.put(arr.get(i),"a");
}
long calculate12 = System.nanoTime();
System.out.println();
System.out.println("The taken for the hashing by using TreeSet for put method for Medium size : "+(calculate12-calculate11)



long calculate13 = System.nanoTime();
for(int i=0;i<1000;i++){
    arr.add(rand.nextInt(5000)+1);
    hashtableChainTreeSet.put(arr.get(i),"a");
}
long calculate14 = System.nanoTime();

System.out.println("The taken for the hashing by using TreeSet for put method for Large size : "+(calculate14-calculate13)+
System.out.println();
```



```
Random random3 = new Random();
ArrayList<Integer> array3 = new ArrayList<Integer>();
long calculator5 = System.nanoTime();
for(int i=0;i<50;i++){
    array3.add(random3.nextInt(5000)+1);
    hashtableChainTreeSet.remove(array3.get(i));
}
long calculator6 = System.nanoTime();
System.out.println();
System.out.println("The taken for the hashing by using linked list for remove method for Medium size : "+(calculator6-calcu


Random random4 = new Random();
ArrayList<Integer> array4 = new ArrayList<Integer>();
long calculator7 = System.nanoTime();
for(int i=0;i<100;i++){
    array4.add(random4.nextInt(5000)+1);
    hashtableChainTreeSet.remove(array4.get(i));
}
long calculator8 = System.nanoTime();
System.out.println();
System.out.println("The taken for the hashing by using linked list for remove method for Large size : "+(calculator8-calcul
```

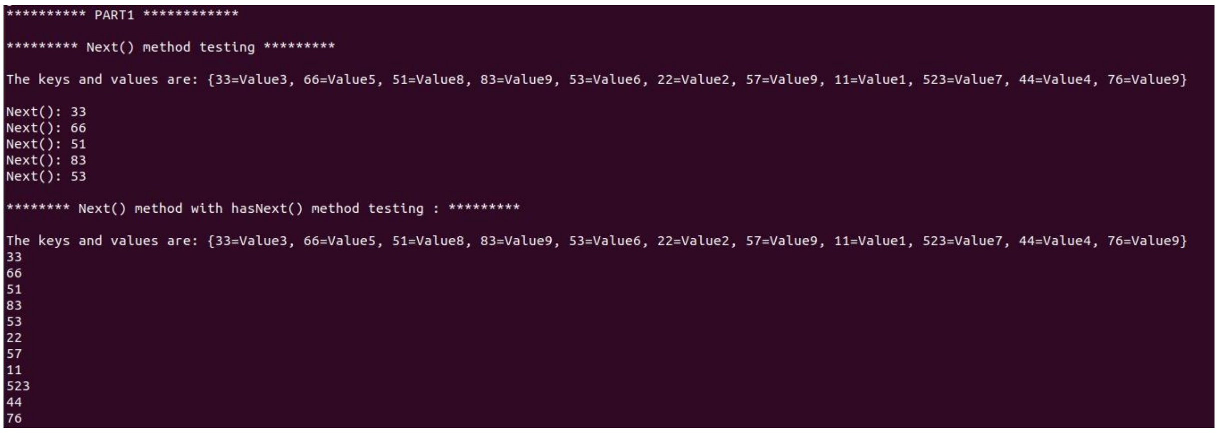
This time I am applying the same operations that I did for the tableChain on the hashtableChainTreeSet. In other words, I generate random numbers and perform operations with medium and large scale remove and put methods as well as small size operations and measure the time taken by these operations.



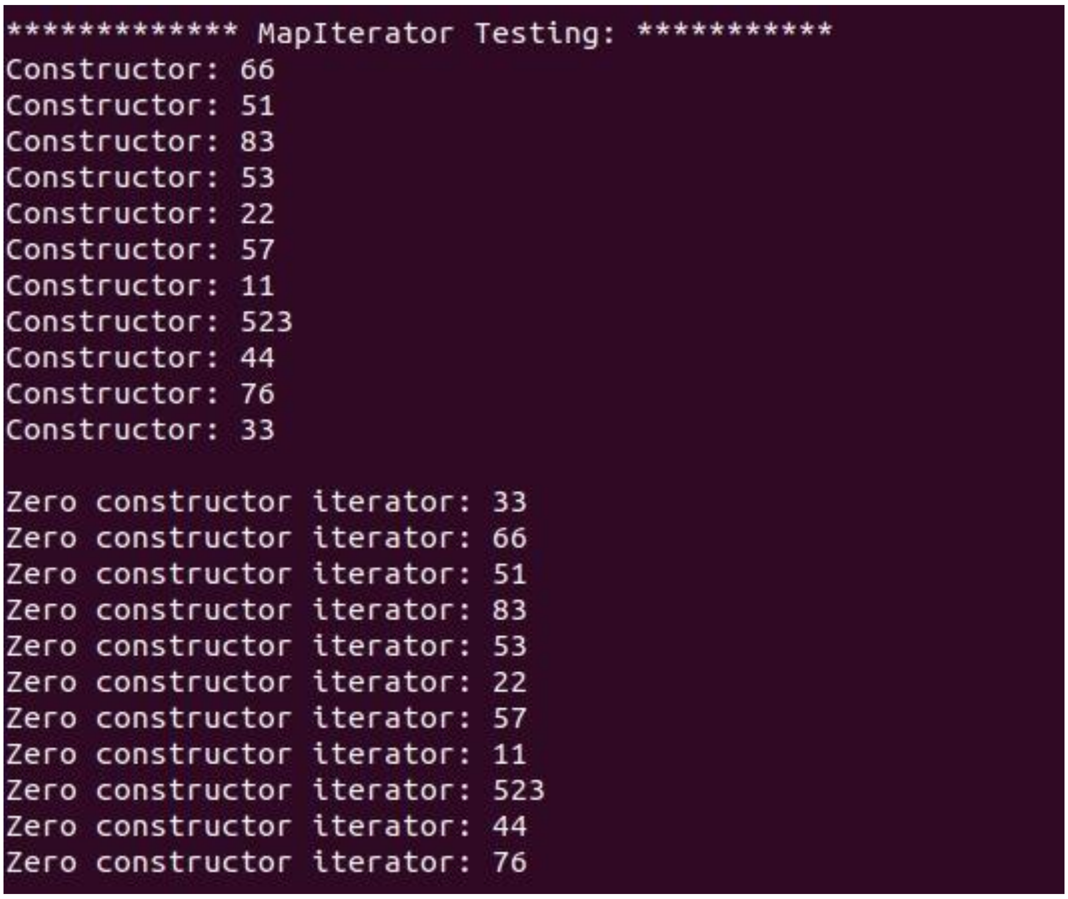
```
hashCoalesced.put(0,"a");
hashCoalesced.put(1,"b");
hashCoalesced.put(2,"c");
hashCoalesced.put(3,"d");
hashCoalesced.put(4,"e");
hashCoalesced.put(5,"f");
hashCoalesced.put(6,"g");
hashCoalesced.put(7,"h");
hashCoalesced.put(8,"k");
hashCoalesced.put(9,"m");
long calculate10 = System.nanoTime();
System.out.println(hashCoalesced);
System.out.println();
System.out.println("The time taken for hashing Coalesced for put() method for small size: "+ (calculate10-calculate
System.out.println();


Random random5 = new Random();
ArrayList<Integer> array5 = new ArrayList<Integer>();
long calculator9 = System.nanoTime();
for(int i=0;i<30;i++){
    array5.add(random5.nextInt(5000)+1);
    hashCoalesced.put(array5.get(i),"a");
}
long calculator10 = System.nanoTime();
System.out.println("The taken for the hashing by using linked list for put method for Medium size : "+(calculator10


System.out.println();
Random random6 = new Random();
ArrayList<Integer> array6 = new ArrayList<Integer>();
long calculator11 = System.nanoTime();
for(int i=0;i<100;i++){
    array6.add(random6.nextInt(5000)+1);
    hashCoalesced.put(array6.get(i),"a");
}
long calculator12 = System.nanoTime();
System.out.println("The taken for the hashing by using linked list for put() method for Medium size : "+(calculator
```

I have implemented the operations I have applied in the other two hash implementation in hashCoalesced.

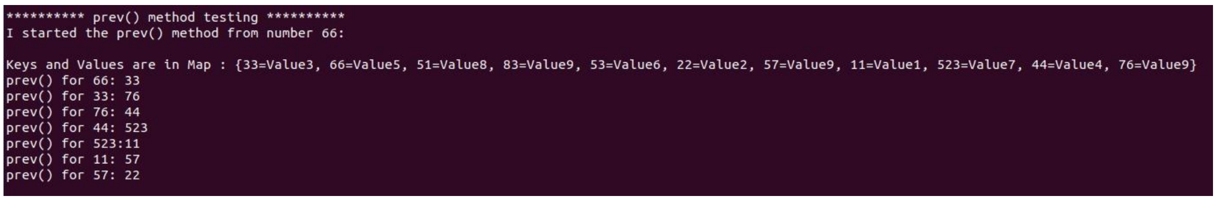
3-RUNNING COMMAND AND RESULTS:



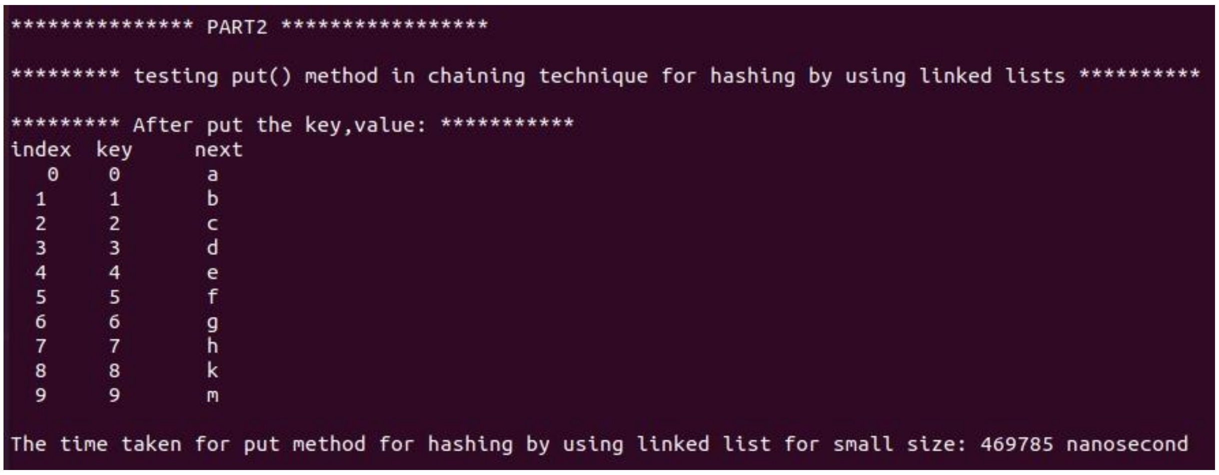
```
********** PART1 ************

********* Next() method testing *********

The keys and values are: {33=Value3, 66=Value5, 51=Value8, 83=Value9, 53=Value6, 22=Value2, 57=Value9, 11=Value1, 523=Value7, 44=Value4, 76=Value9}

Next(): 33
Next(): 66
Next(): 51
Next(): 83
Next(): 53

******** Next() method with hasNext() method testing : *********

The keys and values are: {33=Value3, 66=Value5, 51=Value8, 83=Value9, 53=Value6, 22=Value2, 57=Value9, 11=Value1, 523=Value7, 44=Value4, 76=Value9}
33
66
51
83
53
22
57
11
523
44
76
```



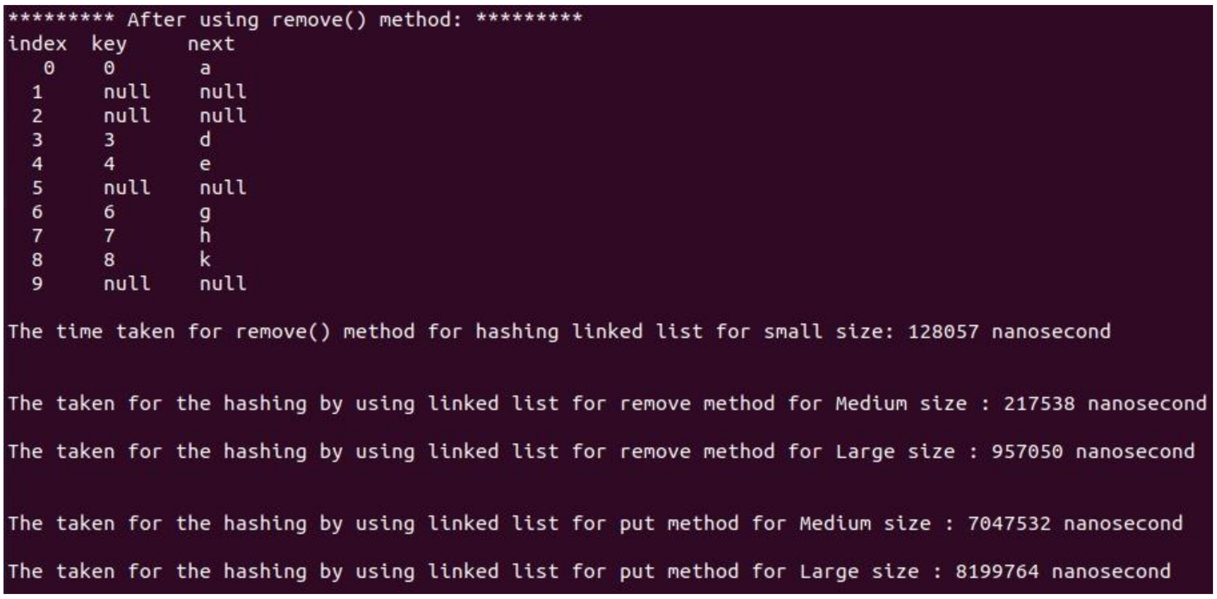
```
************** MapIterator Testing: ***********
Constructor: 66
Constructor: 51
Constructor: 83
Constructor: 53
Constructor: 22
Constructor: 57
Constructor: 11
Constructor: 523
Constructor: 44
Constructor: 76
Constructor: 33

Zero constructor iterator: 33
Zero constructor iterator: 66
Zero constructor iterator: 51
Zero constructor iterator: 83
Zero constructor iterator: 53
Zero constructor iterator: 22
Zero constructor iterator: 57
Zero constructor iterator: 11
Zero constructor iterator: 523
Zero constructor iterator: 44
Zero constructor iterator: 76
```



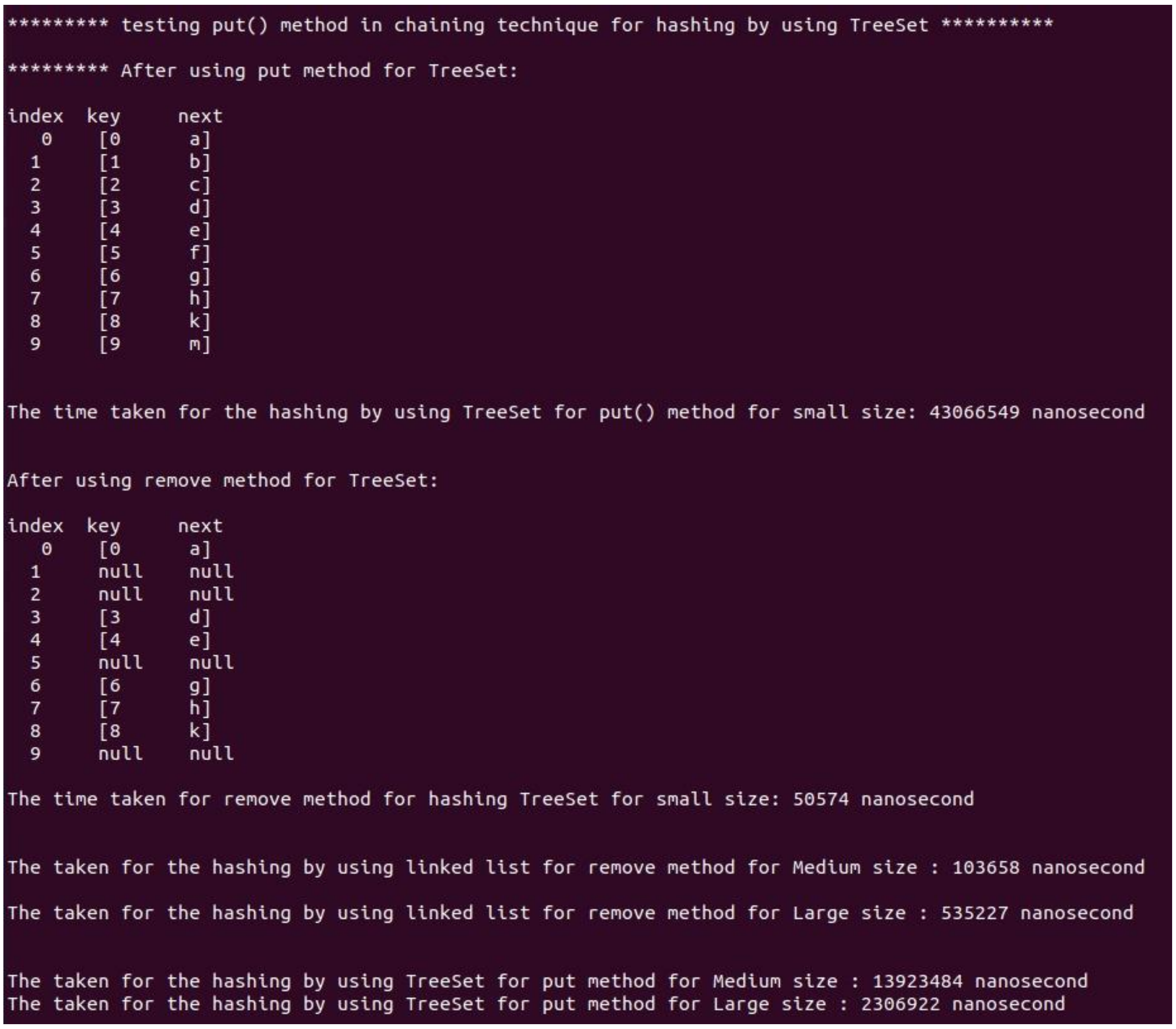
```
********** prev() method testing **********
I started the prev() method from number 66:

Keys and Values are in Map : {33=Value3, 66=Value5, 51=Value8, 83=Value9, 53=Value6, 22=Value2, 57=Value9, 11=Value1, 523=Value7, 44=Value4, 76=Value9}
prev() for 66: 33
prev() for 33: 76
prev() for 76: 44
prev() for 44: 523
prev() for 523:11
prev() for 11: 57
prev() for 57: 22
```



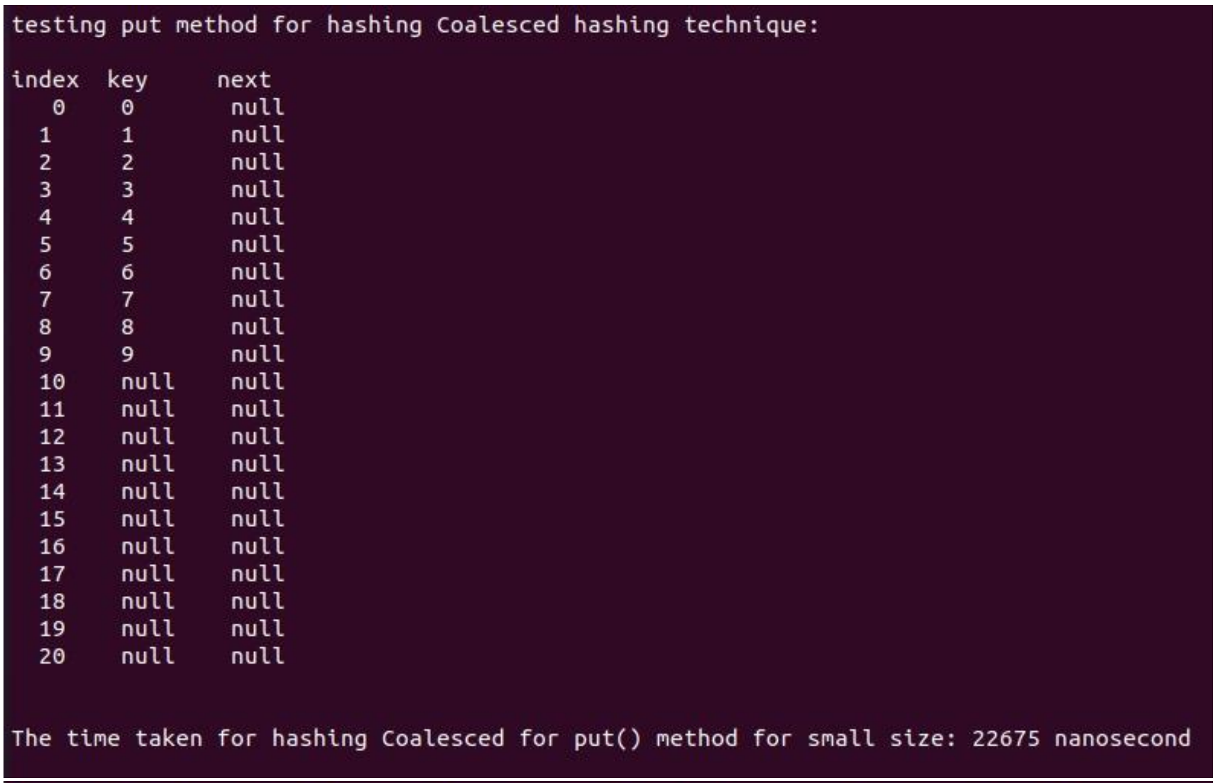
```
*************** PART2 *****************

********* testing put() method in chaining technique for hashing by using linked lists **********

********* After put the key,value: ***********
index  key     next
   0    0       a
  1     1       b
  2     2       c
  3     3       d
  4     4       e
  5     5       f
  6     6       g
  7     7       h
  8     8       k
  9     9       m

The time taken for put method for hashing by using linked list for small size: 469785 nanosecond
```



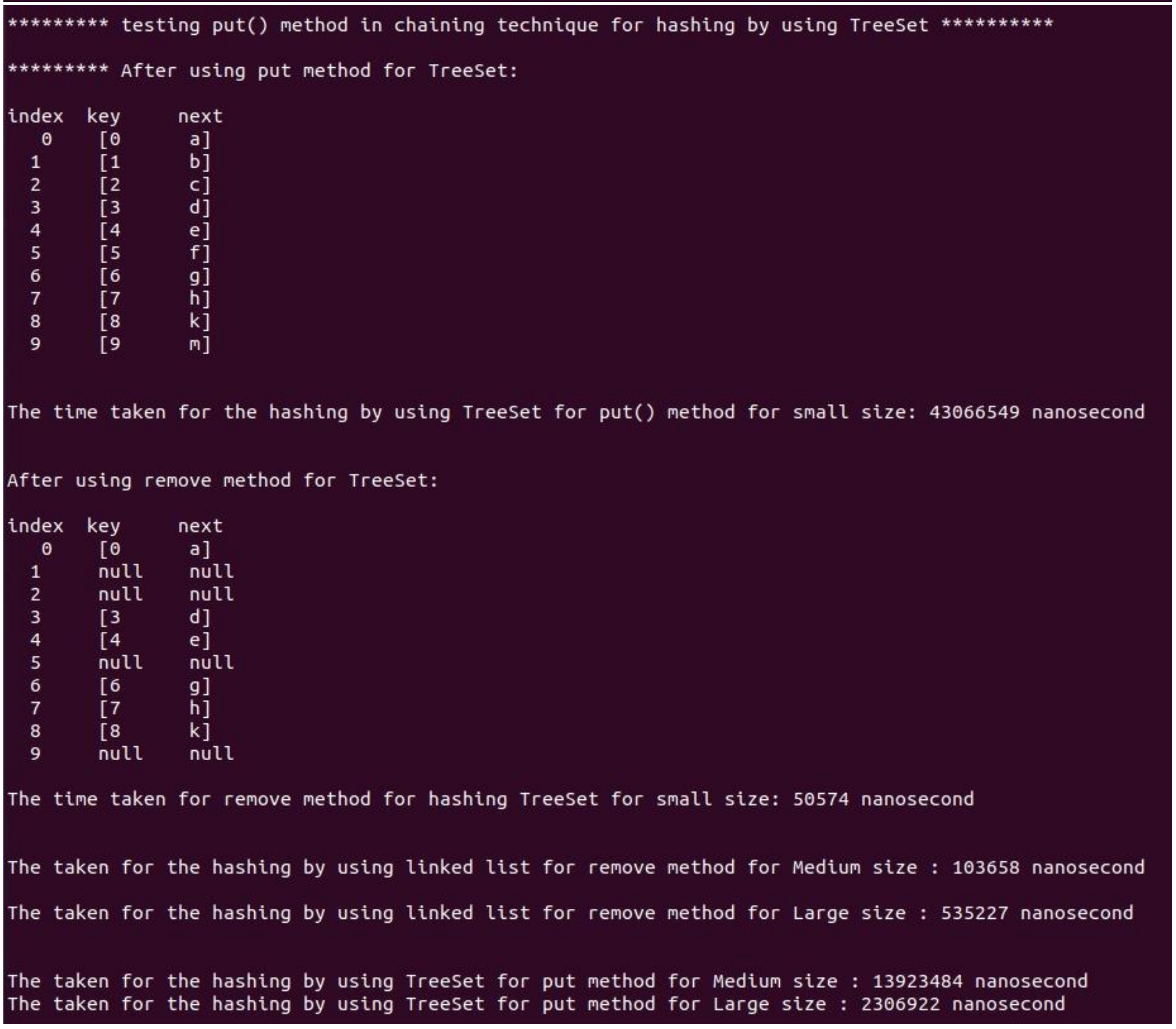
```
********* After using remove() method: *********
index  key     next
   0    0       a
  1     null    null
  2     null    null
  3     3       d
  4     4       e
  5     null    null
  6     6       g
  7     7       h
  8     8       k
  9     null    null

The time taken for remove() method for hashing linked list for small size: 128057 nanosecond


The taken for the hashing by using linked list for remove method for Medium size : 217538 nanosecond

The taken for the hashing by using linked list for remove method for Large size : 957050 nanosecond


The taken for the hashing by using linked list for put method for Medium size : 7047532 nanosecond

The taken for the hashing by using linked list for put method for Large size : 8199764 nanosecond
```



```
********* testing put() method in chaining technique for hashing by using TreeSet **********

********* After using put method for TreeSet:

index  key     next
   0    [0      a]
  1     [1      b]
  2     [2      c]
  3     [3      d]
  4     [4      e]
  5     [5      f]
  6     [6      g]
  7     [7      h]
  8     [8      k]
  9     [9      m]


The time taken for the hashing by using TreeSet for put() method for small size: 43066549 nanosecond


After using remove method for TreeSet:

index  key     next
   0    [0      a]
  1     null    null
  2     null    null
  3     [3      d]
  4     [4      e]
  5     null    null
  6     [6      g]
  7     [7      h]
  8     [8      k]
  9     null    null

The time taken for remove method for hashing TreeSet for small size: 50574 nanosecond


The taken for the hashing by using linked list for remove method for Medium size : 103658 nanosecond

The taken for the hashing by using linked list for remove method for Large size : 535227 nanosecond


The taken for the hashing by using TreeSet for put method for Medium size : 13923484 nanosecond
The taken for the hashing by using TreeSet for put method for Large size : 2306922 nanosecond
```



```
testing put method for hashing Coalesced hashing technique:

index  key     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    null    null
  11    null    null
  12    null    null
  13    null    null
  14    null    null
  15    null    null
  16    null    null
  17    null    null
  18    null    null
  19    null    null
  20    null    null


 The time taken for hashing Coalesced for put() method for small size: 22675 nanosecond
```



```
********* testing put() method in chaining technique for hashing by using TreeSet **********

********* After using put method for TreeSet:

index  key     next
   0    [0      a]
  1     [1      b]
  2     [2      c]
  3     [3      d]
  4     [4      e]
  5     [5      f]
  6     [6      g]
  7     [7      h]
  8     [8      k]
  9     [9      m]


The time taken for the hashing by using TreeSet for put() method for small size: 43066549 nanosecond


After using remove method for TreeSet:

index  key     next
   0    [0      a]
  1     null    null
  2     null    null
  3     [3      d]
  4     [4      e]
  5     null    null
  6     [6      g]
  7     [7      h]
  8     [8      k]
  9     null    null

The time taken for remove method for hashing TreeSet for small size: 50574 nanosecond


The taken for the hashing by using linked list for remove method for Medium size : 103658 nanosecond

The taken for the hashing by using linked list for remove method for Large size : 535227 nanosecond


The taken for the hashing by using TreeSet for put method for Medium size : 13923484 nanosecond
The taken for the hashing by using TreeSet for put method for Large size : 2306922 nanosecond
```