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

Yunus Emre Öztürk
1901042619

1. PROBLEM SOLUTION APPROACH

Part 1: In part 1 I used keySet method of the HashMap structure in Java. Because of the prev method, I should've track all of my moves in the keyset.

```
//this stack will save the returned key by the next() to access them later if
user call prev()
private ArrayDeque<K> remainPrev = new ArrayDeque<>();
```

As iterator of keySet moves, I put the items that I passed to a stack and I used that stack structure if user wants to use prev.

```
//this stack will save the returned key by the prev() to access them later if
user call next()
private ArrayDeque<K> remainNext = new ArrayDeque<>();
```

As user uses prev I stored the return elements from the prev to another Stack structure and I used it if user wants to go next element.

When user wants to go next element I checked if the Stack that I stored the elements is empty or not, if it is empty I used keySet iterator, if it is not I pop the item from Stack and returned that item to User. If iterator and Stack that I used has no element left method returns the first element in the set.

For prev method if stack that I used for prev is empty I returned the last element.

If user wants to start from another element other than first I used my remainNext stack and I moved the keySet iterator until program finds the given key, as I moved the iterator I saved the elements from iterator to remainNext stack. After that if user wants to go next program will travel through the remainNext first and then it will continue to use iterator. In this case first element will be the given key and last element will be the last element returned by the iterator.

Part 2: For the linkedlist chain part I implemented the methods with similar way in the textbook.

For the part used TreeSet for chains I used ceiling method of the TreeSet while searching in the chain, because if I use iterator it will go ascending order and it will take O(n) time complexity and it will act like, like I implemented chain structure with linkedlist, with ceiling/floor method I decreased this time complexity to O(logn) which can be seen in screenshots from my worst scenario case (if hashcode of all given elements are same):

```
Worst case test (10000elements (with same hashcode)):
Time took in miliseconds for putting items in HashMapListChain: 115
Time took in miliseconds for putting items in HashMapTreeSetChain: 6
Time took in miliseconds for putting items in HashMapCoalesced: 1052


Time took in miliseconds for getting items in HashMapListChain: 76
Time took in miliseconds for getting items in HashMapTreeSetChain: 3
Time took in miliseconds for getting items in HashMapCoalesced: 364


Time took in miliseconds for removing items in HashMapListChain: 64
Time took in miliseconds for removing items in HashMapTreeSetChain: 5
Time took in miliseconds for removing items in HashMapCoalesced: 158
**********************************************************
```

In the part used Coalesced method I stored indices of next item in chain in inner Entry class as integer. When user puts new item, hashmap looks for the index specified with key if the place is full program uses quadratic probing technique, after it finds an empty place it puts the new element there and connects that element with the end of the chain of the original place. For remove if place that will be removed has next element it replaces the place with that next element. If it doesn't have a next element program will delete the place and if there is a entry that points to it, removes that point.

For all parts I used similar rehash method which uses the put methods of them.

2. TEST CASES / RUNNING AND RESULTS

```
**********************************Iterable Hash Map*******************************
List of elements in set [97, 33, 67, 68, 4, 8, 42, 80, 17, 84, 21, 23, 25, 28, 60, 94, 62]
iter.next from start to end:
97, 33, 67, 68, 4, 8, 42, 80, 17, 84, 21, 23, 25, 28, 60, 94, 62,
20 random next and prev:
next: 97
next: 97
next: 97
next: 97
next: 97
next: 97
prev: 62
prev: 94
next: 94
prev: 94
prev: 60
next: 60
next: 94
prev: 94
next: 94
next: 62
prev: 62
prev: 94
next: 94
prev: 94
```

```
iter.next from start to end with iterator starts from middle element:
21, 84, 17, 80, 42, 8, 4, 68, 67, 33, 97, 23, 25, 28, 60, 94, 62,
20 random next and prev with iterator starts from middle element:
next: 21
prev: 21
prev: 62
prev: 62
prev: 62
prev: 62
next: 21
next: 84
next: 17
next: 80
prev: 80
prev: 17
prev: 84
next: 84
prev: 84
next: 84
next: 17
prev: 17
prev: 84
next: 84
```

Part 2:

```
Average case test (1000elements (random)):
Time took in miliseconds for putting items in HashMapListChain: 3
Time took in miliseconds for putting items in HashMapTreeSetChain: 4
Time took in miliseconds for putting items in HashMapCoalesced: 1


Time took in miliseconds for getting items in HashMapListChain: 1
Time took in miliseconds for getting items in HashMapTreeSetChain: 0
Time took in miliseconds for getting items in HashMapCoalesced: 1


Time took in miliseconds for removing items in HashMapListChain: 1
Time took in miliseconds for removing items in HashMapTreeSetChain: 1
Time took in miliseconds for removing items in HashMapCoalesced: 0
*******************************************************
```

```
Average case test (10000elements (random)):
Time took in miliseconds for putting items in HashMapListChain: 4
Time took in miliseconds for putting items in HashMapTreeSetChain: 5
Time took in miliseconds for putting items in HashMapCoalesced: 4


Time took in miliseconds for getting items in HashMapListChain: 2
Time took in miliseconds for getting items in HashMapTreeSetChain: 2
Time took in miliseconds for getting items in HashMapCoalesced: 1


Time took in miliseconds for removing items in HashMapListChain: 2
Time took in miliseconds for removing items in HashMapTreeSetChain: 3
Time took in miliseconds for removing items in HashMapCoalesced: 1
*******************************************************
```

```
Average case test (100000elements (random)):
Time took in miliseconds for putting items in HashMapListChain: 13
Time took in miliseconds for putting items in HashMapTreeSetChain: 23
Time took in miliseconds for putting items in HashMapCoalesced: 6


Time took in miliseconds for getting items in HashMapListChain: 14
Time took in miliseconds for getting items in HashMapTreeSetChain: 16
Time took in miliseconds for getting items in HashMapCoalesced: 3


Time took in miliseconds for removing items in HashMapListChain: 5
Time took in miliseconds for removing items in HashMapTreeSetChain: 5
Time took in miliseconds for removing items in HashMapCoalesced: 3
*******************************************************
```

```
Worst case test (100elements (with same hashcode)):
Time took in miliseconds for putting items in HashMapListChain: 1
Time took in miliseconds for putting items in HashMapTreeSetChain: 1
Time took in miliseconds for putting items in HashMapCoalesced: 1


Time took in miliseconds for getting items in HashMapListChain: 1
Time took in miliseconds for getting items in HashMapTreeSetChain: 0
Time took in miliseconds for getting items in HashMapCoalesced: 1


Time took in miliseconds for removing items in HashMapListChain: 1
Time took in miliseconds for removing items in HashMapTreeSetChain: 0
Time took in miliseconds for removing items in HashMapCoalesced: 0
*******************************************************
```

```
Worst case test (1000elements (with same hashcode)):
Time took in miliseconds for putting items in HashMapListChain: 28
Time took in miliseconds for putting items in HashMapTreeSetChain: 5
Time took in miliseconds for putting items in HashMapCoalesced: 14


Time took in miliseconds for getting items in HashMapListChain: 33
Time took in miliseconds for getting items in HashMapTreeSetChain: 0
Time took in miliseconds for getting items in HashMapCoalesced: 3


Time took in miliseconds for removing items in HashMapListChain: 32
Time took in miliseconds for removing items in HashMapTreeSetChain: 1
Time took in miliseconds for removing items in HashMapCoalesced: 1
*******************************************************
```

```
Worst case test (10000elements (with same hashcode)):
Time took in miliseconds for putting items in HashMapListChain: 115
Time took in miliseconds for putting items in HashMapTreeSetChain: 6
Time took in miliseconds for putting items in HashMapCoalesced: 1052


Time took in miliseconds for getting items in HashMapListChain: 76
Time took in miliseconds for getting items in HashMapTreeSetChain: 3
Time took in miliseconds for getting items in HashMapCoalesced: 364


Time took in miliseconds for removing items in HashMapListChain: 64
Time took in miliseconds for removing items in HashMapTreeSetChain: 5
Time took in miliseconds for removing items in HashMapCoalesced: 158
*******************************************************
```

```
Trying to get non-existing element:
1001 not found in HashMapListChain, test passed.
1001 not found HashMapTreeSetChain, test passed.
1001 not found HashMapCoalesced, test passed.
1002 not found in HashMapListChain, test passed.
1002 not found HashMapTreeSetChain, test passed.
1002 not found HashMapCoalesced, test passed.
1003 not found in HashMapListChain, test passed.
1003 not found HashMapTreeSetChain, test passed.
1003 not found HashMapCoalesced, test passed.
1004 not found in HashMapListChain, test passed.
1004 not found HashMapTreeSetChain, test passed.
1004 not found HashMapCoalesced, test passed.
1005 not found in HashMapListChain, test passed.
1005 not found HashMapTreeSetChain, test passed.
1005 not found HashMapCoalesced, test passed.
```

```
Trying to remove non-existing element:
1001 not found in HashMapListChain, test passed.
1001 not found HashMapTreeSetChain, test passed.
1001 not found HashMapCoalesced, test passed.
1002 not found in HashMapListChain, test passed.
1002 not found HashMapTreeSetChain, test passed.
1002 not found HashMapCoalesced, test passed.
1003 not found in HashMapListChain, test passed.
1003 not found HashMapTreeSetChain, test passed.
1003 not found HashMapCoalesced, test passed.
1004 not found in HashMapListChain, test passed.
1004 not found HashMapTreeSetChain, test passed.
1004 not found HashMapCoalesced, test passed.
1005 not found in HashMapListChain, test passed.
1005 not found HashMapTreeSetChain, test passed.
1005 not found HashMapCoalesced, test passed.
```

```
Trying to add null:
NullPointerException throwed, test passed.
```

```
Coalesced map example:
hash val       key      next
3    1222      1222       4
4    1291      1291       null
9    3919      3919       null
```