GTU Department of Computer Engineering

CSE 222/505 – Spring 2021

Homework 5 Report

ŞEYMA NUR CANBAZ

171044076

PART1:

1. PROBLEM SOLUTION APPROACH:

First I created a class called MapIterator. I needed a hashMap to navigate this class as an iterator. So I got the hashMap object from the constructor and assigned it to the table array. If the hashMap is not empty and the key value is not given in the constructor, I had to iterate all the elements from the beginning. That's why I kept index and counter information. While the index keeps the current position information, the counter increases for each iterated element. So I can see if I have iterated all the elements in the table. (If the key value is given as a parameter and we want to navigate the whole table starting from that element, counter is increased). I look at the counter while finding the next and previous values. For the hasNext method, if counter == table.length, the entire table is navigated. For the hasPrevious method, if counter == 0, it means the entire table has been navigated. With the index value, the current value is returned.

2. TEST CASES:

- If the given hasMap is empty and while iterate it
- No parameter constructor (iterator starts from beginning)
- Constructor with key (iterator starts from given key)

3. RUNNING COMMAND AND RESULTS:

- When hash map is empty and while iterate it, throws exception.

```
If given hash map is empty:
java.util.NoSuchElementException: Hash map is empty
```

- While iterator starts from beginning (while no given key)

```
Hash map:
{50=50, 20=20, 40=40, 10=10, 30=30}

While starts from beginning:

******* NEXT ********
50
20
40
10
30

******* PREV ********
30
10
40
20
50
```

- While iterator starts from given key (given key)

```
Hash map:
{50=50, 35=35, 20=20, 40=40, 25=25, 10=10, 30=30, 15=15}

While starts from 25:

******* NEXT ********
25
10
30
15
50
35
20
40

******* PREV ********
40
20
35
50
30
10
25
40
```

PART2:

1. PROBLEM SOLUTION APPROACH:

First, I implemented the chaining classes in the book. To implement TreeSet, I made the inner class Comparable in the treeSetChain class. I created a CoalescedHashMap class for coalesced hashing. In the inner Entry class of this class, I kept the nextIndex value to access the next element.

I checked that the key given by the getIndexOf method is not in the table and found its index.

I used auxiliary methods as 'delete' and 'move' in remove method. Delete method connects elements before and after the key to be deleted, and connects them to each other and shifts each one to the place of the deleted element (with the move method). For this, I found the prevIndex value of the key to be deleted and checked whether its nextIndex is full. If there is no other key after the key to be deleted, but there is a preceding key, I kept the nextIndex of the previous key as -1. If the key to be deleted doesn't have the previous and next, I deleted it directly in the remove method.

In the put method, I found the index that should be added with the find method first. While finding this index, if there is collision, I checked the next index with quadratic probing. If it is empty, find method returns this index. If it is not empty, I kept the index value (target) of the next element (when it was null) and returned it. If the next is not null, I calculated the index with quadratic probilng until it is null. If table.length is not enough, I called the rehash () method for increase the table length.

2. TEST CASES:

- Insert item
- Delete item
- Delete non exist item
- Insert item with same key
- Check map is empty or not

3. RUNNING COMMAND AND RESULTS:

- Check map is empty or not.
- Add 1-A
- Check map is empty or not.
- Add 2-C
- Add 3-D
- Add 1-a
- Get the value of 2
- Get the value of 4 (non exist)
- Delete 3
- Delete 4 (non exist)

```
**** HASH TABLE CHAIN ******

The map is empty
[1 - A]

The map is not empty.
[1 - A]
[2 - C]

[1 - A]
[2 - C]
[3 - D]

[1 - a]
[2 - C]
[3 - D]

The value of key(2): C
The value of key(4): null
The value of removed key(3): D
[1 - a]
[2 - C]

The value of removed key(4): null
```

```
**** TREE SET CHAIN ******

The map is empty.
[[1 - A]]

The map is not empty.
[[1 - A]]
[[2 - C]]

[[1 - A]]
[[2 - C]]
[[3 - D]]

[[1 - a]]
[[2 - C]]
[[3 - D]]

The value of key(2): C
The value of key(4): null
The value of removed key(3): D
[[1 - a]]
[[2 - C]]

The value of removed key(4): null
```

Coalesced Hash Map:

- Add 3,12,13,25,23,51,42,
- Get the value of key 23
- Remove 13

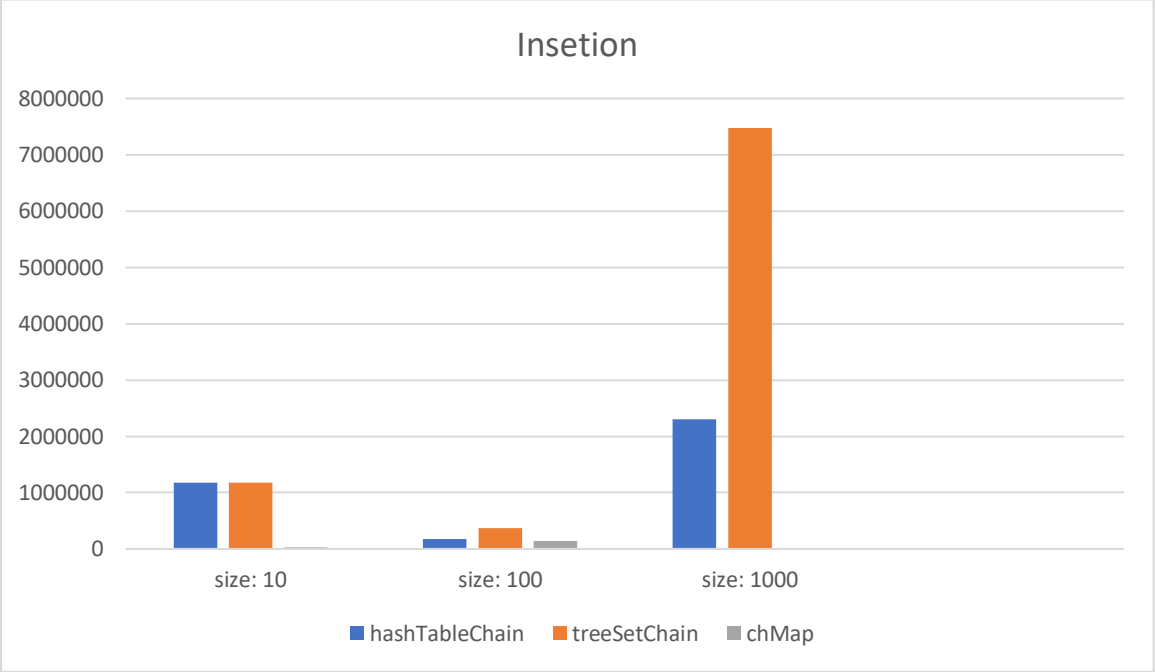
```
**** COALESCED HASH MAP ******

Add 3
Index: 0        Key: null       Value: null     NextIndex: null
Index: 1        Key: null       Value: null     NextIndex: null
Index: 2        Key: null       Value: null     NextIndex: null
Index: 3        Key: 3          Value: 3        NextIndex: -1
Index: 4        Key: null       Value: null     NextIndex: null
Index: 5        Key: null       Value: null     NextIndex: null
Index: 6        Key: null       Value: null     NextIndex: null
Index: 7        Key: null       Value: null     NextIndex: null
Index: 8        Key: null       Value: null     NextIndex: null
Index: 9        Key: null       Value: null     NextIndex: null

Add 12
Index: 0        Key: null       Value: null     NextIndex: null
Index: 1        Key: null       Value: null     NextIndex: null
Index: 2        Key: 12         Value: 12       NextIndex: -1
Index: 3        Key: 3          Value: 3        NextIndex: -1
Index: 4        Key: null       Value: null     NextIndex: null
Index: 5        Key: null       Value: null     NextIndex: null
Index: 6        Key: null       Value: null     NextIndex: null
Index: 7        Key: null       Value: null     NextIndex: null
Index: 8        Key: null       Value: null     NextIndex: null
Index: 9        Key: null       Value: null     NextIndex: null

Add 13
Index: 0        Key: null       Value: null     NextIndex: null
Index: 1        Key: null       Value: null     NextIndex: null
Index: 2        Key: 12         Value: 12       NextIndex: -1
Index: 3        Key: 3          Value: 3        NextIndex: 4
Index: 4        Key: 13         Value: 13       NextIndex: -1
Index: 5        Key: null       Value: null     NextIndex: null
Index: 6        Key: null       Value: null     NextIndex: null
Index: 7        Key: null       Value: null     NextIndex: null
Index: 8        Key: null       Value: null     NextIndex: null
Index: 9        Key: null       Value: null     NextIndex: null
```

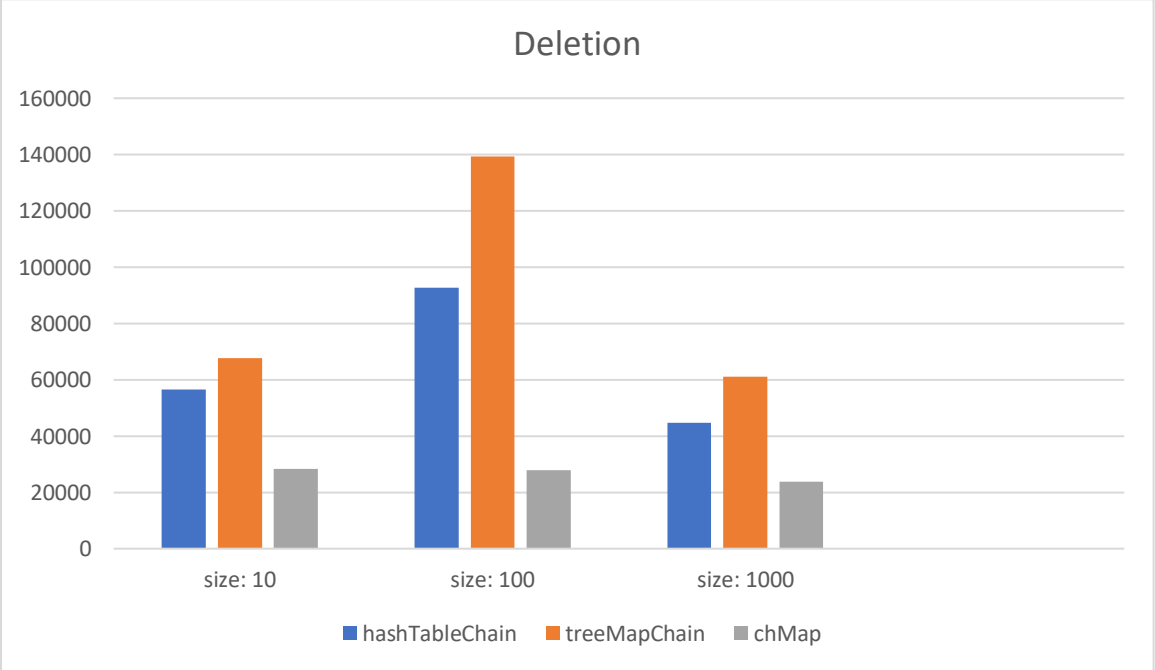
```
Add 25
Index: 0        Key: null       Value: null     NextIndex: null
Index: 1        Key: null       Value: null     NextIndex: null
Index: 2        Key: 12         Value: 12       NextIndex: -1
Index: 3        Key: 3          Value: 3        NextIndex: 4
Index: 4        Key: 13         Value: 13       NextIndex: -1
Index: 5        Key: 25         Value: 25       NextIndex: -1
Index: 6        Key: null       Value: null     NextIndex: null
Index: 7        Key: null       Value: null     NextIndex: null
Index: 8        Key: null       Value: null     NextIndex: null
Index: 9        Key: null       Value: null     NextIndex: null

Add 23
Index: 0        Key: null       Value: null     NextIndex: null
Index: 1        Key: null       Value: null     NextIndex: null
Index: 2        Key: 12         Value: 12       NextIndex: -1
Index: 3        Key: 3          Value: 3        NextIndex: 4
Index: 4        Key: 13         Value: 13       NextIndex: 7
Index: 5        Key: 25         Value: 25       NextIndex: -1
Index: 6        Key: null       Value: null     NextIndex: null
Index: 7        Key: 23         Value: 23       NextIndex: -1
Index: 8        Key: null       Value: null     NextIndex: null
Index: 9        Key: null       Value: null     NextIndex: null

Add 51
Index: 0        Key: null       Value: null     NextIndex: null
Index: 1        Key: 51         Value: 51       NextIndex: -1
Index: 2        Key: 12         Value: 12       NextIndex: -1
Index: 3        Key: 3          Value: 3        NextIndex: 4
Index: 4        Key: 13         Value: 13       NextIndex: 7
Index: 5        Key: 25         Value: 25       NextIndex: -1
Index: 6        Key: null       Value: null     NextIndex: null
Index: 7        Key: 23         Value: 23       NextIndex: -1
Index: 8        Key: null       Value: null     NextIndex: null
Index: 9        Key: null       Value: null     NextIndex: null
```

```
Add 51
Index: 0        Key: null       Value: null     NextIndex: null
Index: 1        Key: 51         Value: 51       NextIndex: -1
Index: 2        Key: 12         Value: 12       NextIndex: -1
Index: 3        Key: 3          Value: 3        NextIndex: 4
Index: 4        Key: 13         Value: 13       NextIndex: 7
Index: 5        Key: 25         Value: 25       NextIndex: -1
Index: 6        Key: null       Value: null     NextIndex: null
Index: 7        Key: 23         Value: 23       NextIndex: -1
Index: 8        Key: null       Value: null     NextIndex: null
Index: 9        Key: null       Value: null     NextIndex: null

Add 42
Index: 0        Key: null       Value: null     NextIndex: null
Index: 1        Key: 51         Value: 51       NextIndex: -1
Index: 2        Key: 12         Value: 12       NextIndex: 6
Index: 3        Key: 3          Value: 3        NextIndex: 4
Index: 4        Key: 13         Value: 13       NextIndex: 7
Index: 5        Key: 25         Value: 25       NextIndex: -1
Index: 6        Key: 42         Value: 42       NextIndex: -1
Index: 7        Key: 23         Value: 23       NextIndex: -1
Index: 8        Key: null       Value: null     NextIndex: null
Index: 9        Key: null       Value: null     NextIndex: null

Key: 23 - Value: 23

Remove 13
Index: 0        Key: null       Value: null     NextIndex: null
Index: 1        Key: 51         Value: 51       NextIndex: -1
Index: 2        Key: 12         Value: 12       NextIndex: 6
Index: 3        Key: 3          Value: 3        NextIndex: 4
Index: 4        Key: 23         Value: 23       NextIndex: -1
Index: 5        Key: 25         Value: 25       NextIndex: -1
Index: 6        Key: 42         Value: 42       NextIndex: -1
Index: 7        Key: null       Value: null     NextIndex: null
Index: 8        Key: null       Value: null     NextIndex: null
Index: 9        Key: null       Value: null     NextIndex: null
```

```
*** Inserting Time ***
HashTableChain (length 10) is 1172801 ns
TreeSetChain (length 10) is 1176101 ns
CoalescedHasphMap (length 10) is 26300 ns

*** Existing Time ***
HashTableChain (length 10): 41300 ns
TreeSetChain (length 10): 55901 ns
CoalescedHasphMap (length 10): 20599 ns

*** Non-existing Time ***
HashTableChain (length 10): 4100 ns
TreeSetChain (length 10): 2800 ns
CoalescedHasphMap (length 10): 3000 ns

*** Removing time ***
HashTableChain (length 10): 56501 ns
TreeSetChain (length 10): 67700 ns
CoalescedHasphMap (length 100): 28400 ns

*** Inserting Time ***
HashTableChain (length 100): 177500 ns
TreeSetChain (length 100): 364899 ns
CoalescedHasphMap (length 100): 143600 ns

*** Existing Time ***
HashTableChain (length 100): 14800 ns
TreeSetChain (length 100): 5601 ns
CoalescedHasphMap (length 100): 3000 ns

*** Non-existing Time ***
HashTableChain (length 100): 3200 ns
TreeSetChain (length 100): 6200 ns
CoalescedHasphMap (length 100): 4000 ns

*** Removing time ***
HashTableChain (length 100): 92800 ns
TreeSetChain (length 100): 139401ns
CoalescedHasphMap (length 100): 27900 ns
```

```
*** Inserting Time ***
HashTableChain (length 1000): 2307201 ns
TreeSetChain (length 1000): 7471500 ns
CoalescedHasphMap (length 1000): 2032500 ns

*** Existing Time ***
HashTableChain (length 1000): 35901 ns
TreeSetChain (length 1000): 35600 ns
CoalescedHasphMap (length 1000): 2700 ns

*** Non-existing Time ***
HashTableChain (length 1000): 7400 ns
TreeSetChain (length 1000): 8100 ns
CoalescedHasphMap (length 1000): 5000 ns

*** Removing time ***
HashTableChain (length 1000): 44800 ns
TreeSetChain (length 1000): 61000 ns
CoalescedHasphMap (length 100): 23701 ns
```





Looking at the data, it is seen that the performance of treeSetChain (hash chain using tree set) is worst. (Because the treeSet is sorted, the operations take longer than the others). It is seen that the most efficient is the coalesced map. Since quadratic probing and seperate chaining are used, it is much easier to find a place in the table for the elements to be added or to access the elements

in the table. Because every time it takes quadratic time. The hashTableChain appears to be in between. Since just linked list is used, its performance is lower than coalesced map.