**GTU Department of Computer Engineering**

**CSE 222/505 - Spring 2021**

**Homework #4 Report**

**Gökbey Gazi KESKİN**

**1901042631**

1. **Problem Solution Approach**

**Part 1)**

The problem definition in part 1 is to iterate through a HashMap with a custom iterator It also should be able to start from somewhere in middle (with Key constructor) so I had to keep the elements which have been iterated. But open-addressing hash table has lots of empty elements. So I kept 2 data fields in private inner class Iterator. First is count. It indicates the amount of already iterated elements. Second is tableIndex. It indicates the position which the iterator is at. Iterator’s next methods iterates until table[tableIndex] is not null. When it finds a non-null entry, it increments the count by one and returns the value. This way, even the iterator is at the end of the hash table, when count is less then the size of the HashMap, it goes to first index and keeps iterating.

**Part2)**

1. **Chained Hashtable with LinkedList**

Problem definition of this part is to create a hash table which has linked lists as elements of the table and keep the entries with same (hash value % table.length) in same linked list. Methods which must find the element’s exact position such has get,put,remove finds the index at the hashtable at first. Then iterates through the linked list if the linked list is not null. Everything else are the same with every type of Hashtable.

1. **Chained Hashtable with TreeSet**

In this assignment, the main difference between TreeSet and LinkedList is LinkedList is iterable no matter what. But in order to iterate through a TreeSet, elements have to be comparable. Because Nodes of the TreeSet has smaller entries to left and higher entries to right. In order to accomplish this, I made Entry class comparable. And I compared the keys of this class. So, keys should also be comparable.

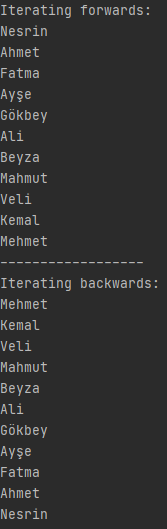
1. **Coalesced Hashtable**

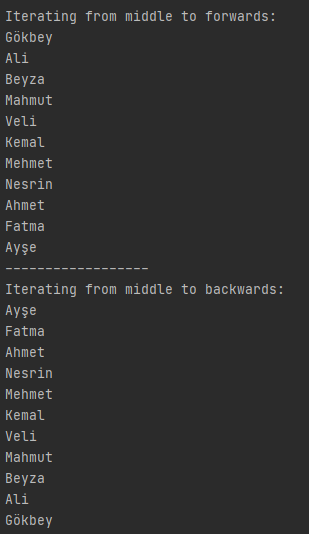
Coalesced hashtable is basically a quadratic probing, open addressing hashtable which indicates the index of the next entry with same (hash value % table.length). In order to do this, I added an additional data field to private inner class next. When a deletion is happened next entry is copied to current entry and next entry is setted to DELETED if there is a next element of the entry which will be deleted. While putting a new entry, program looks for the next entries if the current one is occupied.

1. **Test Cases & Results**

**Cases are written on the screenshots.**

**Part 1)**

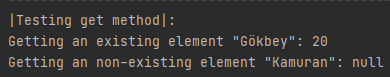
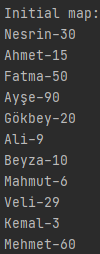
****

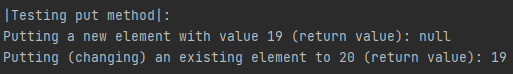


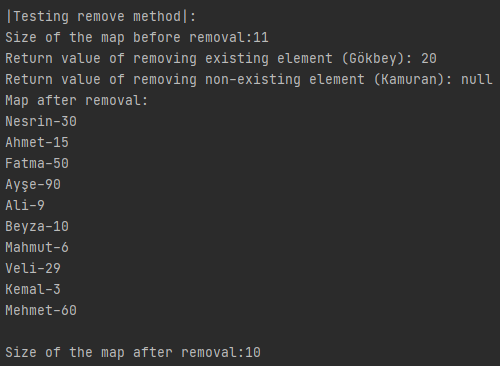
**Part 2)**

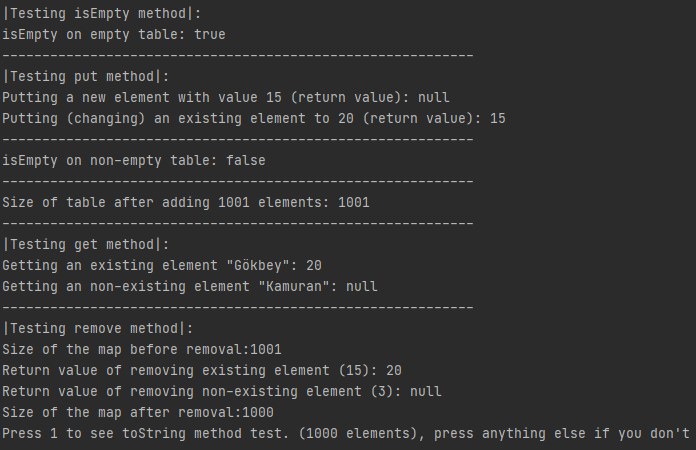
** 2.1 with small amount of data)**

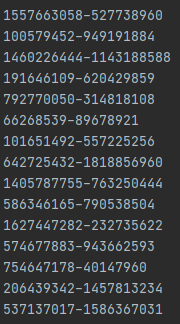
****







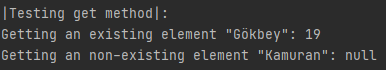
** 2.1 with big amount of data)**

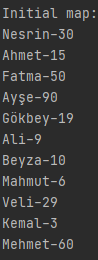
****

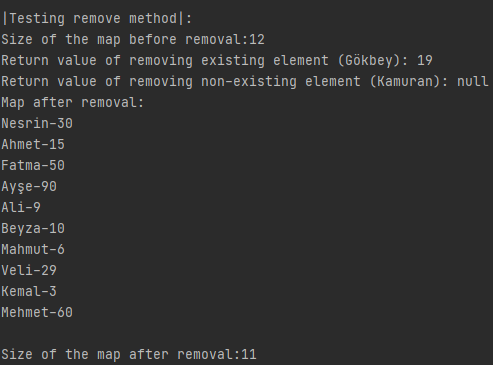
**…….**

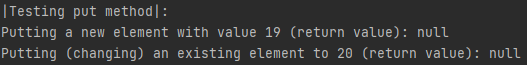
**2.2 with small amount of data)**



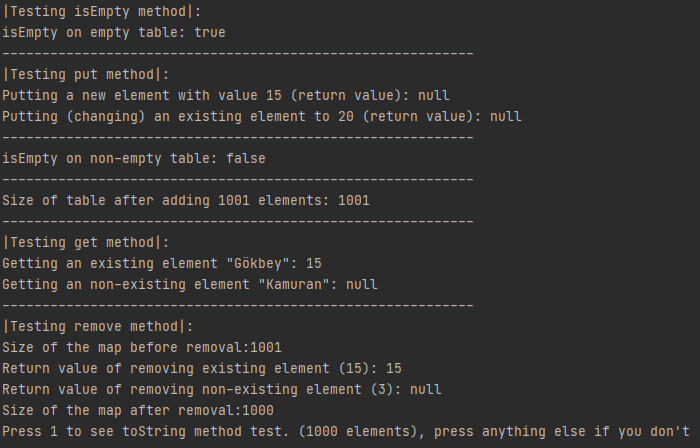


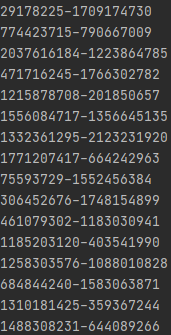






**2.2 with big amount of data)**

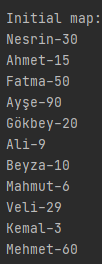
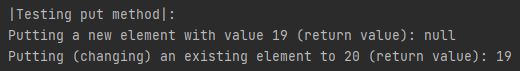
****

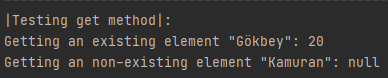


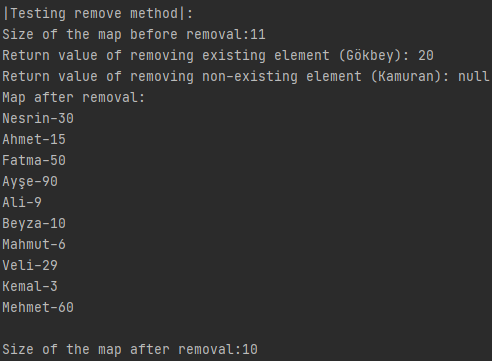
**…**

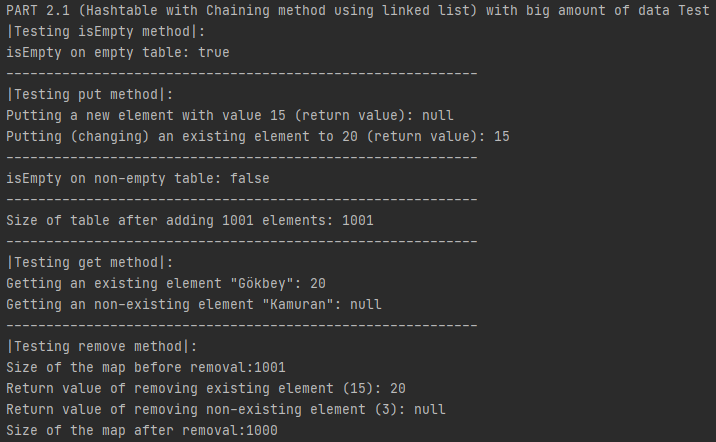
**2.3 with small amount of data)**

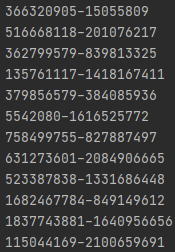
****

****

****

****

**2.3 with big amount of data)**

** …**

1. **Time Comparisons**

****

****

****