Gebze Technical University
Computer Engineering

CSE 222
2017 Spring

HOMEWORK 7 REPORT

VAKHID BETRAKHMADOV
141044086

Course Assistant: Nur Banu Albayrak

1.Problem Solution Approach

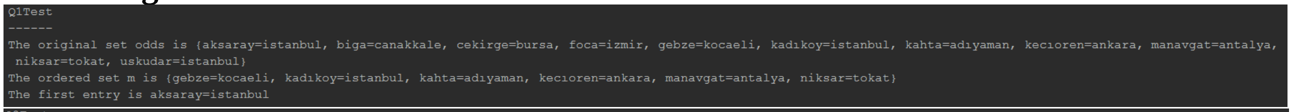
Q1:
In the first question we were supposed to write BinaryNavMap class which would implement NavigableMap interface. NavigableMap interface extends Map interface. This means that we had to implement a class similar to the TreeMap class in Java. In order to do that, I had to study first the Java TreeMap API. TreeMap in Java uses Red-Black tree data structure to implement TreeMap. That is a self-balancing Binary Search Tree, which guaranties O(log(n)) time performance for access, insertion and deletion of data. In our case, we have used just Binary Search Tree data structure, so expected performance is still O(log(n)) but might sometimes go up O(n). In order to impelement BinaryNavMap completely, I had to implement a lot of inner classes which implement Set and NavigableSet interfaces. This inner classes allowed a lot of code reuse later on. BinaryNavMap class works just like TreeMap, except for the put method in the view returned by the subMap method (just had no time left). I have tested both BinaryNavMap and TreeMap parallely on the same operations.
Note: subMap(1,true,1false) or subSet(1,true,1false) yields undefined behavior.

Q2.
In the second question we were supposed to implement hash table Map using chaining strategy, but using other hash table maps instead of LinkedLists. This was a very easy task to do. I have taken the book’s source code, made all methods in it private and implemented my own adapter methods. Internally structure stayed almost intact, except for a new array of HashTableChaining references. Performance for access, addition and removal of any data has not changed and remained constant.

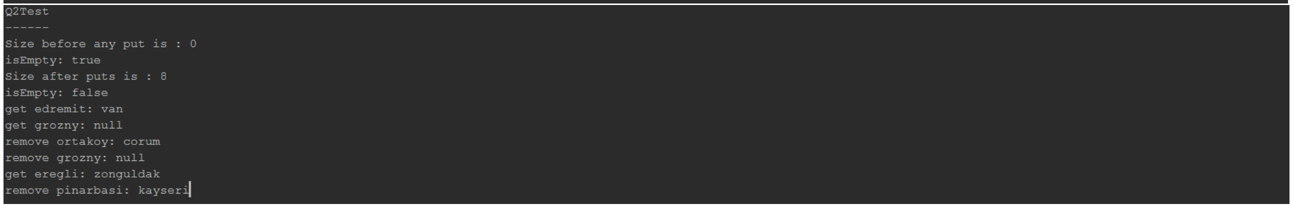
2.Test Cases

I have used test cases supplied, and passed them all successfully. I have also written my own tests. They are quite self-descriptive and need no explanations.

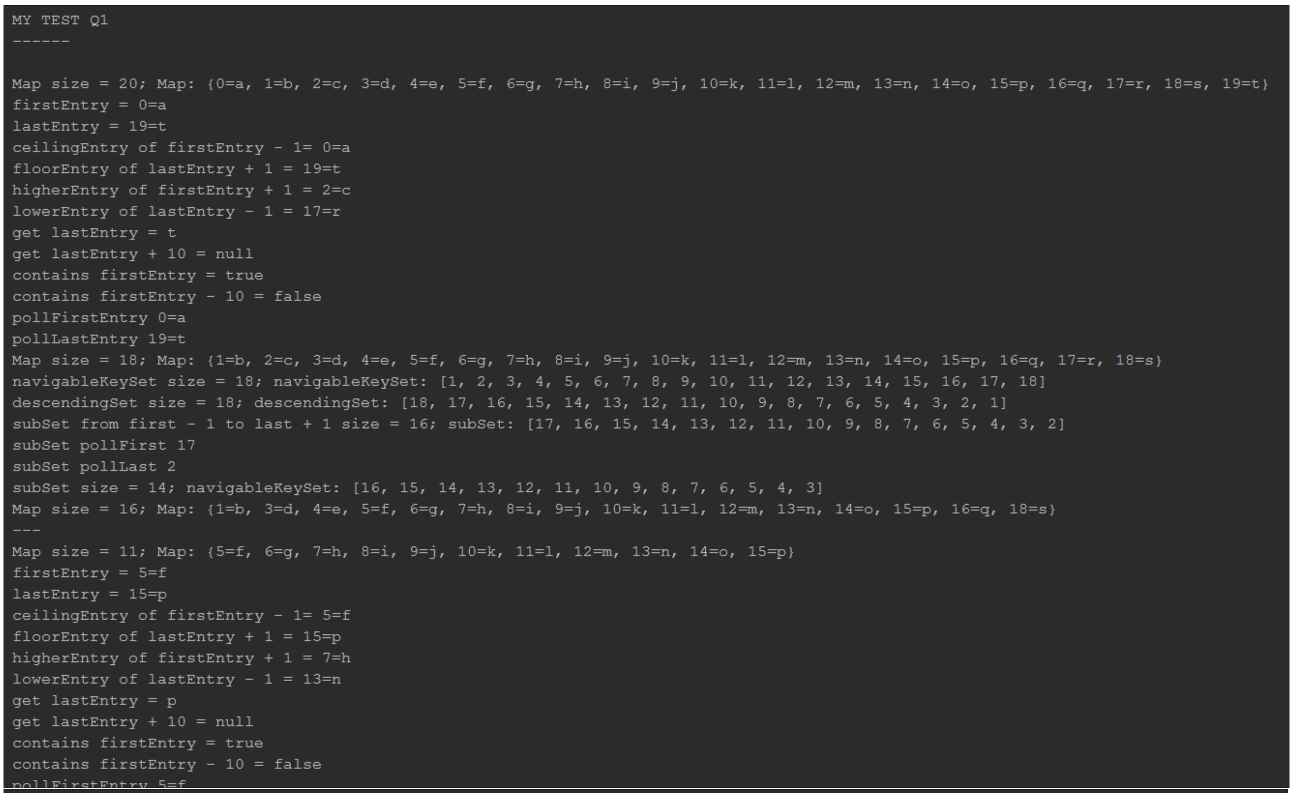
2.Running Command and Results



```
Q1Test
------
The original set odds is {aksaray=istanbul, biga=canakkale, cekirge=bursa, foca=izmir, gebze=kocaeli, kadıkoy=istanbul, kahta=adıyaman, kecıoren=ankara, manavgat=antalya,
 niksar=tokat, uskudar=istanbul}
The ordered set m is {gebze=kocaeli, kadıkoy=istanbul, kahta=adıyaman, kecıoren=ankara, manavgat=antalya, niksar=tokat}
The first entry is aksaray=istanbul
```

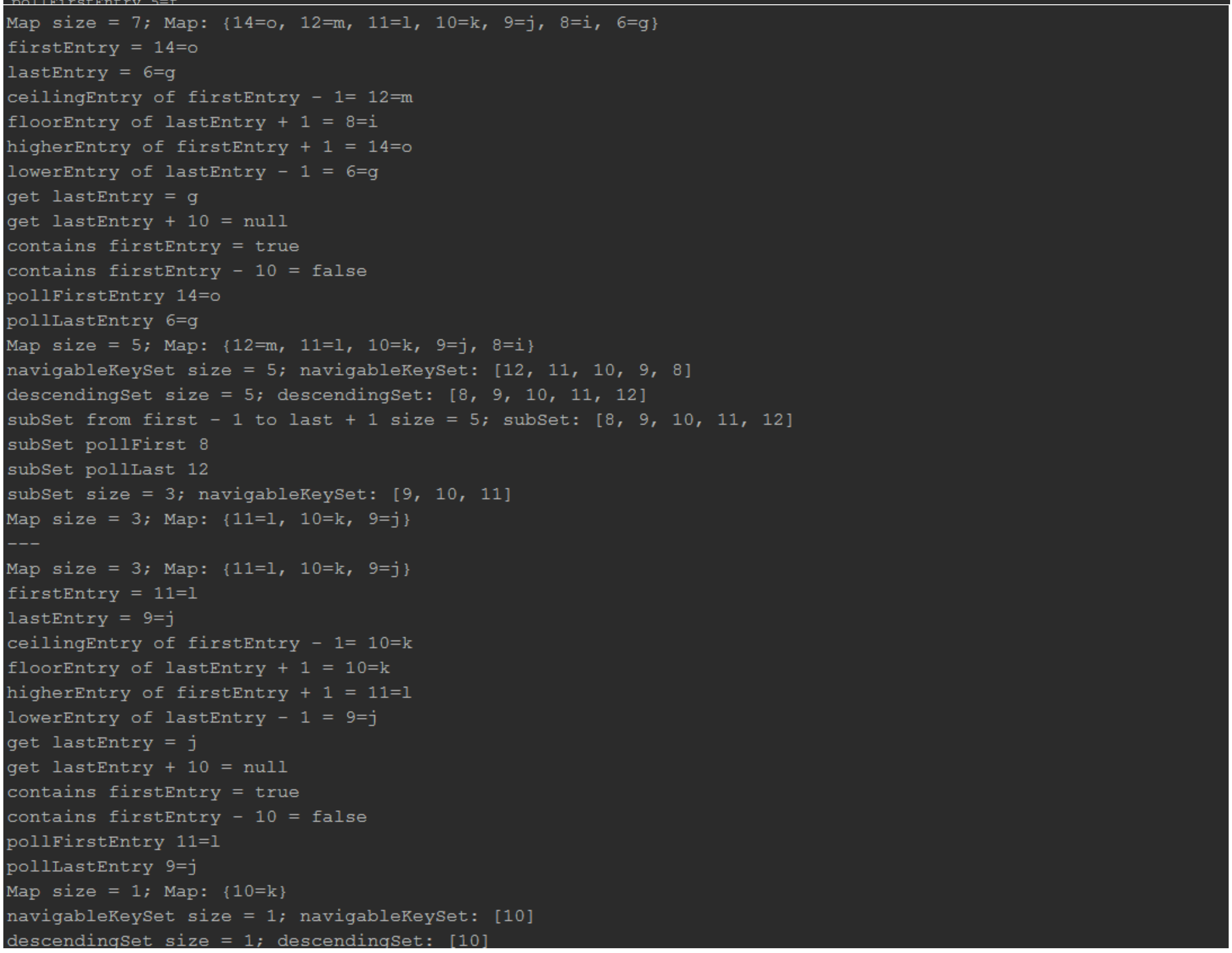


```
Q2Test
------
Size before any put is : 0
isEmpty: true
Size after puts is : 8
isEmpty: false
get edremit: van
get grozny: null
remove ortakoy: corum
remove grozny: null
get eregli: zonguldak
remove pinarbasi: kayseri
```



```
MY TEST Q1
------

Map size = 20; Map: {0=a, 1=b, 2=c, 3=d, 4=e, 5=f, 6=g, 7=h, 8=i, 9=j, 10=k, 11=l, 12=m, 13=n, 14=o, 15=p, 16=q, 17=r, 18=s, 19=t}
firstEntry = 0=a
lastEntry = 19=t
ceilingEntry of firstEntry - 1= 0=a
floorEntry of lastEntry + 1 = 19=t
higherEntry of firstEntry + 1 = 2=c
lowerEntry of lastEntry - 1 = 17=r
get lastEntry = t
get lastEntry + 10 = null
contains firstEntry = true
contains firstEntry - 10 = false
pollFirstEntry 0=a
pollLastEntry 19=t
Map size = 18; Map: {1=b, 2=c, 3=d, 4=e, 5=f, 6=g, 7=h, 8=i, 9=j, 10=k, 11=l, 12=m, 13=n, 14=o, 15=p, 16=q, 17=r, 18=s}
navigableKeySet size = 18; navigableKeySet: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
descendingSet size = 18; descendingSet: [18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
subSet from first - 1 to last + 1 size = 16; subSet: [17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]
subSet pollFirst 17
subSet pollLast 2
subSet size = 14; navigableKeySet: [16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3]
Map size = 16; Map: {1=b, 3=d, 4=e, 5=f, 6=g, 7=h, 8=i, 9=j, 10=k, 11=l, 12=m, 13=n, 14=o, 15=p, 16=q, 18=s}
---
Map size = 11; Map: {5=f, 6=g, 7=h, 8=i, 9=j, 10=k, 11=l, 12=m, 13=n, 14=o, 15=p}
firstEntry = 5=f
lastEntry = 15=p
ceilingEntry of firstEntry - 1= 5=f
floorEntry of lastEntry + 1 = 15=p
higherEntry of firstEntry + 1 = 7=h
lowerEntry of lastEntry - 1 = 13=n
get lastEntry = p
get lastEntry + 10 = null
contains firstEntry = true
contains firstEntry - 10 = false
pollFirstEntry 5=f
```



```
Map size = 7; Map: {14=o, 12=m, 11=l, 10=k, 9=j, 8=i, 6=g}
firstEntry = 14=o
lastEntry = 6=g
ceilingEntry of firstEntry - 1= 12=m
floorEntry of lastEntry + 1 = 8=i
higherEntry of firstEntry + 1 = 14=o
lowerEntry of lastEntry - 1 = 6=g
get lastEntry = g
get lastEntry + 10 = null
contains firstEntry = true
contains firstEntry - 10 = false
pollFirstEntry 14=o
pollLastEntry 6=g
Map size = 5; Map: {12=m, 11=l, 10=k, 9=j, 8=i}
navigableKeySet size = 5; navigableKeySet: [12, 11, 10, 9, 8]
descendingSet size = 5; descendingSet: [8, 9, 10, 11, 12]
subSet from first - 1 to last + 1 size = 5; subSet: [8, 9, 10, 11, 12]
subSet pollFirst 8
subSet pollLast 12
subSet size = 3; navigableKeySet: [9, 10, 11]
Map size = 3; Map: {11=l, 10=k, 9=j}
---
Map size = 3; Map: {11=l, 10=k, 9=j}
firstEntry = 11=l
lastEntry = 9=j
ceilingEntry of firstEntry - 1= 10=k
floorEntry of lastEntry + 1 = 10=k
higherEntry of firstEntry + 1 = 11=l
lowerEntry of lastEntry - 1 = 9=j
get lastEntry = j
get lastEntry + 10 = null
contains firstEntry = true
contains firstEntry - 10 = false
pollFirstEntry 11=l
pollLastEntry 9=j
Map size = 1; Map: {10=k}
navigableKeySet size = 1; navigableKeySet: [10]
descendingSet size = 1; descendingSet: [10]
```