Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

SEFA NADİR YILDIZ
131044031

Fatma Nur Esirci

1 Double Hashing Map

1.1 Pseudocode and Explanation

Write pseudocode and explanation about code design. Indicate what you are using that interfaces, classes, structures, etc.

public class DoubleHashingMap< K, V> implements Map<K, V> write class

```
initialize doubleHashTable
initialize TABLE_CAPACITY
initialize OCCUPANCY_RATE_TABLE
initialize TABLE_CAPACITY
initialize hashFunction2
initialize deletingElements
initialize standingElements
```

private static class keyValuePair< K, V> implements Map.Entry<K, V> write nested class

public int size()
    returns the number of entries in the map

public boolean isEmpty()
    returns true if map is empty

public boolean containsKey(Object key)
    for loop until doubleHashTable.length
        check doubleHashTable[i] is null
        check doubleHashTable[i].getDeletedState() not equals 1
      and check doubleHashTable[i].getKey().equals(key)
        returns true if this map contains a mapping for the specified

  return false f this map don't contain a mapping for the specified

public boolean containsValue(Object value)
    for loop until doubleHashTable.length
        check doubleHashTable[i] is null
        check doubleHashTable[i].getDeletedState() not equals 1
      and check doubleHashTable[i].getValue().equals(value)
        returns true if this map maps one or more keys to the specified value

  return false if this map don't map one or more keys to the specified value

public V get(Object key)
    call findIndex(key);
    check doubleHashTable[index] not equals null
        if doubleHashTable[index] not equals null
            return return doubleHashTable[index].value
        if doubleHashTable[index] equals null
            return null

```
public V put(K key, V value)
        call  findIndex(key)
        if  doubleHashTable[index] equals  null
                constructor new keyValuePair<>(key, value);
        check  loadFactor

        if loadFactor greater than  OCCUPANCY_RATE_TABLE
                call  reallocDoubleHashing()
        else
                return old value


public V remove(Object key)
        call  findIndex(key)
        if  doubleHashTable[index] is  null
                return null
        decrease number of elements
        return old value


public void putAll(Map<? extends K, ? extends V> m)

        for loop until m  size
                determine keys and values
                put keys and values to the map


public void clear()
        clear all of the mappings from this map


public Set<K> keySet()
        create a set object
        for loop map length
                if  doubleHashTable[i] not equals null and
                if  doubleHashTable[i].getDeletedState() not equals 1
                        add keys to set
        return set


public Collection<V> values()
        create a collection
        for loop map length
                if  doubleHashTable[i] not equals null and
                if  doubleHashTable[i].getDeletedState() not equals 1
                        add values to collection
        return collection
```

```
public Set<Entry<K, V>> entrySet()
        create set
        for loop map length
                create  keyValuePair object
                if  doubleHashTable[i] not equals null and
                if  doubleHashTable[i].getDeletedState() not equals 1
                        add keys to set
                        add values to set
        return set


private int findIndex(Object key)
        find hash code of the key
        check hash code

        if  doubleHashTable[tableIndex1] not equals null and
        if  key not equals map key.equals
                find hash code2
                check hash code2
                call  calculateSecondHash()

        return hash code



private int calculateSecondHash(int hash1, int hash2, int multiplication)
        calculate return hash1 + multiplication * hash2 % hashFunction1;



private void reallocDoubleHashing()
        create temp table
        assign map table to temp table
        increase temp table size
        assign temp table to map table

        put keys and values to map table


public String toString()
        create  StringBuilder object
        append keys to StringBuilder object
        append values to StringBuilder object

        return StringBuilder object
```

Double Hashing Map oluşturmak için öncelikle **public class DoubleHashingMap< K, V> implements Map<K, V>** sınıfımı oluşturdum. Bu sınıfı Map sınıfından generic olarak implement ettim. Bu sınıfın objesini oluştururken K ve V tipinde iki tip gerekmektedir. Yani DoubleHashingMap sınıfım <String, Integer>, <Integer, Integer> gibi yapılar da generic iki tip tutabilmektedir.

Bu sınıfın içinde bir de **private static class keyValuePair< K, V> implements Map.Entry<K, V>** sınıfı bulunmaktadır. Bu sınıfı Map.Entry sınıfından implement ettim.
private K key;
private V value;
private int deletedState;

keyValuePair sınıfı yukarıda görülen 3 tip veriyi tutmaktadır. **deletedState** değişkenini mapten silinen keylerin konumunu tutmak için kullandım.

DoubleHashingMap sınıfını Map sınıfından implement ettiğim için 11 tane methodu istenilene göre yeniden dizayn edip yazdım.

size() methodu

Bu methodta map sınıfımda kaç tane eleman olduğunu return ettim.

isEmpty() methodu

Bu methodta map sınıfımın boş olup olmadığını kontrol ettim.

containsKey(Object key)

Bu methodta parametreden gelen key'in map sınıfımda olup olmadığını kontrol ettim.

containsValue(Object value)

Bu methodta parametreden gelen değerin map sınıfımda olup olmadığını kontrol ettim.

get(Object key)

Bu method parametreden gelen key'i map sınıfımda arar ve eşleşen key değeri bulursa bu key'e karşılık gelen değerin geri dönmesini kontrol ettim.

put(K key, V value)
Bu methodta gelen key'e özel bir hash kod üreterek, bu hash koda karşılık gelen konumun map sınıfımda dolu veya boş olmasını kontrol ederek key ve value çiftini duruma göre map sınıfıma yerleştirme işlemi yaptım. Eğer map sınıfımın kapasitesi 0.75 oranına yaklaştıysa map sınıfımın alanını yeniden güncelledim.

remove(Object key)

Bu methodta parametreden gelen key değerine göre hash kod üretip, bu hash koda karşılık gelen konum map sınıfımdaki key ile eşleşiyorsa bu key değeri ve bu key'e karşılık gelen değer map sınıfımdan silinir.

putAll(Map<? extends K, ? extends V> m)

Bu methodta parametreden gelen map'in içindeki key ve value değerlerini **put(K key, V value)** methodunu kullanarak map sınıfıma yerleştirme işlemlerini yaptım.

clear()

Bu methodu map sınıfımın içeriğini temizlemek için kullandım.

KeySet()

Bu methodta map sınıfımdaki key'leri belirleyerek bir set oluşturdum ve key'leri seti yapısına ekleyerek return ettim.

values()

Bu methodta map sınıfımdaki value'lari belirleyerek bir collection yapısı oluşturdum ve bu collection yapısına ekleyerek return ettim.

EntrySet()

Bu methodta bulunan key ve value değerlerinin set görüntüsünü oluşturarak return ettim.

findIndex(Object key)

Bu methodta gelen key'e göre özel bir hash kod ürettim. Bu özel koda karşılık gelen konumda çakışma meydana geldiyse double hashing uygulayarak uygun bir hash kod ürettim.

calculateSecondHash(int hash1, int hash2, int multiplication)

Çakışma olması durumunda double hashing uygulayıp yeni bir hash kod üretmek için bu methodu kullandım. **findIndex(Object key)** methodu içinde çağırdım.

ReallocDoubleHashing()

Bu methodta, map sınıfımın alanı 0,75 doluluk oranını aştığı zaman alanımın boyutunu arttırdım.

ToString()

Bu methodta map sınıfımın key ve value çitlerini ekrana basmak için @override işlemi kullandım.

1.2 Test Cases

```
DoubleHashingMap<String, Integer> mapObject = new DoubleHashingMap<>();

    mapObject.put("toyota", 123);
    mapObject.put("mercedes", 248);
    mapObject.put("fiat", 347);

    Set<Map.Entry<String, Integer>> tableEntrySet = new HashSet<>();
    tableEntrySet = mapObject.entrySet();

    Iterator iterator = tableEntrySet.iterator();
    System.out.println("size:" + tableEntrySet.size());

    System.out.println(tableEntrySet.iterator().next().getKey() + "->" +
       tableEntrySet.iterator().next().getValue());

    System.out.println(tableEntrySet.iterator().next().getKey() + "->" +
       tableEntrySet.iterator().next().getValue());

    System.out.println(tableEntrySet.iterator().next().getKey() + "->" + t
       ableEntrySet.iterator().next().getValue());

    System.out.println(mapObject.toString());
```

2 Recursive Hashing Set

2.1 Pseudocode and Explanation

Write pseudocode and explanation about code design. Indicate what you are using that interfaces, classes, structures, etc.

public class RecursiveHashingSet<T> implements Set<T> write this class

```
initialize numberOfKeys;
initialize ecursiveHashTable;
initialize  CAPACITY = 10;
```

private static class Node<T> write nested class

public int size()
    returns the number of entries in the map

public boolean isEmpty()
    returns true if map is empty

public boolean contains(Object o)
    call recursive method contains(Node<T>[] recursiveHashTable, Object o)

private boolean contains(Node<T>[] recursiveHashTable, Object o)
    if recursiveHashTable is null
        return false;
    find hash code of Object o

    if recursiveHashTable[index] not equals null and
    if recursiveHashTable[index].getData() not equals null
        if Object o equals data
            return true;
        else
            if ecursiveHashTable[index].next not equals null
            call contains (recursive) and return

    return false

public Object[] toArray()
    call recursive method toArray(Node<T>[] recursiveHashTable, Object[] container, int index)

```
private Object[] toArray(Node<T>[] recursiveHashTable, Object[] container, int index)
        if  container is null
                create new array
        for loop until  recursiveHashTable length
                if recursiveHashTable[i] not equlas null and
                if recursiveHashTable[i].getData() not equals null
                        add data to array
                        if recursiveHashTable[i].next  not equals null
                                call toArray ( recursive )
        return container array;


public <T> T[] toArray(T[] a)
        call recursive method toArray(Node<T>[] recursiveHashTable, Object[] container, int index)


public boolean add(T e)
        call contains method
        check contains
                call recursive method add(Node<T>[] recursiveHashTable, T e)
        return true;


private Node<T>[] add(Node<T>[] recursiveHashTable, T e)
        if  recursiveHashTable is null)
                create new array
        find hash code of e

        call add ( recursive)
        return  recursiveHashTable


public boolean remove(Object o)
        check current size
        call recursive method remove(Node<T>[] recursiveHashTable, Object o)
        check new size
        if  current size greater tha new size
                return true
        return false


private Node<T>[] remove(Node<T>[] recursiveHashTable, Object o)
        call contains for Object o
         if recursiveHashTable equals null
                 return recursiveHashTable
        find hash code of Object o

        call remove ( recursive )

        return  recursiveHashTable
```

```
public boolean containsAll(Collection<?> c)
        if  c equals null |or c is empty
                return false;
        for loop until end element of c
                check same element
        if all elements are same
                return true
        return false


public boolean addAll(Collection<? extends T> c)

        if  c equals null or c is empty
                return false;
        for loop until end element of c
                call add method
        return true;


public boolean retainAll(Collection<?> c)
        call toArray method
        create collection object

        if  c equals null or c is empty
                return false;
        determine instersection set and fill to collection

        call removeAll methodl


public boolean removeAll(Collection<?> c)
        if  c equals null or c is empty
                return false;
        for loop until end element of c
                call remove method
        return true;


public void clear()
        clear all of the elements from this set


Iterator<T> iterator()
        return new iterator class object


class setIterator implements Iterator<T> write nested class for Iterator<T> iterator() method
        implement public boolean hasNext()
        implement public T next()
```

Chaining hash table kullanarak set sınıfı tanımlamak için **RecursiveHashingSet<T> implements Set<T>** generic bir sınıf oluşturdum ve bu sınıfımı Set sınıfından implement ettim. Chainin table yapısını uygulamak için **private static class Node<T>** sınıfı oluşturdum.

Node sınıfının içinde **T data, Node<T>[] next** olarak iki tane veri tuttum. Set sınıfıma ekleyeceğim verileri tutmak için private **Node<T>[] recursiveHashTable** dizisi tanımladım. Set sınıfımı yazarken şu methodları implement ettim.

size() methodu

Bu methodta set sınıfımda kaç tane eleman olduğunu return ettim.

isEmpty() methodu

Bu methodta set sınıfımın boş olup olmadığını kontrol ettim.

contains(Object o) methodu

Bu yöntemde parametreden gelen değerin set snıfımda olup olmadığını bu method içerisinde recursive **contains(Node<T>[] recursiveHashTable, Object o)** methodunu çağırarak kontrol ettim. Parametreden gelen değer sınıfımın içinde bulunuyorsa true return ettim.

object[] toArray() methodu

Bu yöntemde set sınıfımdaki elemanlarımı bu method içerisinde recursive **toArray(Node<T>[] recursiveHashTable, Object[] container, int index)** methodunu çağırarak **Object[] container** dizisine ekleyerek return ettim.

toArray(T[] a) methodu

Bu yöntemde parametreden gelen boş diziye set sınıfımdaki elemanlarımı bu method içerisinde recursive olarak **toArray(Node<T>[] recursiveHashTable, Object[] container, int index)** methodunu çağırarak T**[] a** dizisine ekleyerek return ettim.

add(T e) methodu

Bu yöntemde öncelikli olarak **contains methodunu** çağırıp parametreden gelen değerin set sınıfımda olup olmadığını kontrol ettim. Eğer yoksa **add(Node<T>[] recursiveHashTable, T e)** recursive methodunu çağırarak set sınıfıma ekleme yaptım.

remove(Object o) methodu

Bu yöntemde öncelikli olarak **contains methodunu** çağırıp parametreden gelen değerin set sınıfımda olup olmadığını kontrol ettim. Eğer parametreden gelen değer set sınıfımda bulunuyorsa **remove(Node<T>[] recursiveHashTable, T e)** recursive methodunu çağırarak set sınıfımdan **Object o** değerini çıkarttım. Çıkarma işlemi başarılı olduğunda ise true return ettim.

containsAll(Collection<?> c) methodu

Bu yöntemde parametreden gelen c collection yapısının içerisindeki elemanları **contains methodu** kontrol ettim. Eğer set sınıfım c collection yapısının tüm elemanlarını içeriyorsa true return ettim.

addAll(Collection<? extends T> c) methodu

Bu yöntemde parametreden gelen c collection yapısının içerisindeki elemanları **add methodu** kullanarak set sınıfıma dahil ettim. Eğer c collection yapısının tüm elemanları doğru bir şekilde set sınıfıma eklendiyse true return ettim.

retainAll(Collection<?> c) methodu

Bu yöntemde **Object[] elements = toArray();** işlemini yaptıktan sonra parametreden gelen c collection yapısyla elements dizinin kesişim kümesi elemanlarını belirledim **ArrayList<Object> intersection** yapısına ekledim ve **removeAll(intersection)** ile ortak olmayan elemanları set sınıfımdan uzaklaştırdım.

removeAll(Collection<?> c) methodu

Bu yöntemde parametreden gelen c collection yapısının içerisindeki elemanları **remove methodu** kullanarak set sınıfımdan çıkarttım. Eğer c collection yapısının tüm elemanları doğru bir şekilde set sınıfımdan çıkarıldıysa true return ettim.

clear() methodu

Bu yöntemde tüm set sınıfımın içeriğine null ataması yaparak temizledim.

toString() methodu

Bu yöntemde set sınıfımın değerlerini **toString(Node<T>[] recursiveHashTable) recursive methodu** ile ekrana yazdırdım.

iterator() methodu

Bu yöntemde set elemanlarım üzerinde gezen bir iterator tanımlamak için **class setIterator implements Iterator<T>** sınıfını iterator sınıfından implement ederek yazdım ve içinde **hasNext()** ile **next()** methodunu implement ettim.

2.2 Test Cases

Try this code least 2 different hash table size and 2 different sequence of keys. Report all of situations.

```
System.out.println("---------Recursive Hashing Set---------");
RecursiveHashingSet<Integer> setObject = new RecursiveHashingSet<>();
setObject.add(14);
setObject.add(24);
setObject.add(94);
setObject.add(12);

setObject.toString();



if (setObject.contains(15)) {
   System.out.println("contains 15");
} else {
   System.out.println("don't contain 15");
}



if (setObject.contains(24)) {
   System.out.println("contains 24");
} else {
   System.out.println("don't contain 24");
}


if (setObject.remove(15)) {
   System.out.println("success remove 15");
} else {
   System.out.println("unseccess remove 15");
}
if (setObject.remove(12)) {
   System.out.println("success remove 12 ");
} else {
   System.out.println("unsuccess remove 12 ");
}


setObject.toString();


System.out.println("-----------------addAll----------------");
ArrayList<Integer> collection1 = new ArrayList<>();
collection1.add(7);
collection1.add(34);
setObject.addAll(collection1);
setObject.toString();
```

```
System.out.println("-----------------removeAll---------------");
ArrayList<Integer> collection = new ArrayList<>();
collection.add(94);
collection.add(34);
setObject.removeAll(collection);
setObject.toString();

System.out.println("-----------------toArray----------------");
Object[] container1 = setObject.toArray();


for (int i = 0; i < container1.length; i++) {
   System.out.println(container1[i]);
}

System.out.println("--------------containsAll----------------");
ArrayList<Integer> collection3 = new ArrayList<>();
collection3.add(14);
collection3.add(24);


if (setObject.containsAll(collection3)) {
   System.out.println("collection contains all elements");
} else {
   System.out.println("collection don't contain all elements");
}


System.out.println("--------------retainAll----------------");
ArrayList<Integer> collection4 = new ArrayList<>();
collection4.add(14);
collection4.add(24);

setObject.retainAll(collection4);
setObject.toString();


System.out.println("------------toArray(T[] a)-------------");
Object[] container2 = null;
container2 = setObject.toArray(container2);
for (int i = 0; i < container2.length; i++) {
   System.out.println(container2[i]);
}

System.out.println("---------------iterator----------------");
Iterator<Integer> iter = setObject.iterator();

while (iter.hasNext()) {
   Object a = iter.next();
   System.out.println(a);
}
```

3 Sorting Algortihms

3.1 MergeSort with DoubleLinkedList

This part about Question3 in HW5

3.1.1 Pseudocode and Explanation

Write pseudocode and explanation about code design. Indicate what you are using that interfaces, classes, structures, etc.

public class MergeSortDoubleLinkedList<T extends Comparable<T>> write this class

define private Node<T> head

private static class Node<T> write nested class

```
private Node<T> separate(Node<T> head)
        if head equals null
                return null
        call separate(Node<T> middle_1, Node<T> middle_2) method


private Node<T> separate(Node<T> middle_1, Node<T> middle_2)

        if middle_1.next not equals null and middle_1.next.next not equals null
                return call separate for  middle_1.next.next and  middle_2.next


private Node<T> mergeSort(Node<T> left)

        if  left equals  null
                return left;

        if  left.next equals  null
                return left;
        call  separate method for left
        call recursive mergeSort for left
        call recursive mergeSort for right

        return and call merge method for left and right


private Node<T> merge(Node<T> left, Node<T> right)
        if left equals null or right equals null
                return null
        compare data of left and data of right

        call recursive merge(left.next, right)
        call recursive merge(left, right.next)
```

Bu partta merge sort with double linked list oluşturmak için
public class MergeSortDoubleLinkedList<T extends Comparable<T>> generic sınıfını oluşturdum. T tipini başka verilerle karşılaştırabilmek için **T extends Comparable<T>** yaptım.

LinkedList yapısını gösterebilmek için **private static class Node<T>** sınıfı tanımladım ve içerisinde **T data, Node<T> next, Node<T> previous** datalarını tuttum.

separate(Node<T> head) methodu

Dizimi parçalara ayırabilmek için bu method içerisinde **separate(Node<T> middle_1, Node<T> middle_2) recursive methodumu** çağırdım ve separete işlemimi gerçekleştirdim.

mergeSort(Node<T> left) methodu

Sort işlemini gerçekleştirirken **merge(Node<T> left, Node<T> right) recursive methodumdan** yararlandım. Recursive methodum içerisinde comparaTo kullanarak sıralama işlemini kontrol ettim.

```
int compare = left.data.compareTo(right.data);
  if (compare < 0) {
     left.next = merge(left.next, right);
     left.next.previous = left;
     left.previous = null;
     return left;
  } else {
     right.next = merge(left, right.next);
     right.next.previous = right;
     right.previous = null;
     return right;
  }
```

printArray(Node<T> node) methodu

Bu yöntemde karışık sırada gelen yapıyı merge sort işlemi yapıldıktan sonraki halini ekrana basmak için kullandım.

----------1--------
Heap Sort
1. run time: 1108193
Insertion Sort
1. run time: 629657
Merge Sort
1. run time: 545411
Quick Sort
1. run time: 739163
---------2---------
Heap Sort
2. run time: 324446
Insertion Sort
2. run time: 45006
Merge Sort
2. run time: 417593
Quick Sort
2. run time: 1068519
---------3---------
Heap Sort
3. run time: 662944
Insertion Sort
3. run time: 46180
Merge Sort
3. run time: 154617
Quick Sort
3. run time: 1317243
---------4---------
Heap Sort
4. run time: 1536452
Insertion Sort
4. run time: 62885
Merge Sort
4. run time: 282806
Quick Sort
4. run time: 5246151

---------5---------
Heap Sort
5. run time: 2480873
Insertion Sort
5. run time: 64580
Merge Sort
5. run time: 333119
Quick Sort
5. run time: 6177556

---------6---------

Heap Sort
6. run time: 3368546
Insertion Sort
6. run time: 134738
Merge Sort
6. run time: 506230
Quick Sort
6. run time: 61858582
---------7---------
Heap Sort
7. run time: 7493927
Insertion Sort
7. run time: 408512
Merge Sort
7. run time: 1171426
Quick Sort
7. run time: 48884294
---------8---------
Heap Sort
8. run time: 6605502
Insertion Sort
8. run time: 2311746
Merge Sort
8. run time: 3123746
Quick Sort
8. run time: 43439774
---------9---------
Heap Sort
9. run time: 8790159
Insertion Sort
9. run time: 1014423
Merge Sort
9. run time: 4654526
Quick Sort
9. run time: 112429744
---------10---------
Heap Sort
10. run time: 14071698
Insertion Sort
10. run time: 1199871
Merge Sort
10. run time: 6988890
Quick Sort
10. run time: 181609999
Heap Sort Average Time: 4644274
Insertion Sort Average Time: 591759
Merge Sort Average Time: 1817836
Quick Sort Average Time: 46277102
Heap Sort Worst Case   : size->5000 time->8785193
Insertion Sort Worst Case: size->10000 time->968636
Merge Sort Worst Case   : size->10000 time->1822162
Quick Sort Worst Case   : size->5000 time->55007657

4 Comparison the Analysis Results