Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

SEFA NADİR YILDIZ
131044031

Course Assistant: Fatma Nur Esirci

1 Worst RedBlack Tree

This part about Question1 in HW6

1.1 Problem Solution Approach

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

- Root node mutlaka siyah olmalıdır.
- Her düğüm ya kırmızı ya da siyahtır.
- Her yaprak (NULL) siyahtır.
- Bir düğüm kırmızıysa, o zaman her iki çocuğu da siyahtır.
- Düğümden soyundan bir yapraka giden her basit yol, aynı sayıda siyah düğüm içerir.

Bu 5 maddeyi kullanarak RedBlack Tree yapımı oluşturdum

public class RedBlackTree< E extends Comparable<E>> extends BinarySearchTreeWithRotate<E>

Görüldüğü üzere **RedBlackTree** sınıfı **BinarySearchTreeWithRotate** sınıfndan extend edilmiştir.

Interface olarak **SearchTree** ve **Serializable** interfaceleri kullanılmıştır.

RedBlack Tree yapısına ekleme yaparken **public boolean add(E item) methodu** ile yardımcı recursive method olan **private Node< E> add(RedBlackNode< E> localRoot, E item)** methodunu kullandım.

```
public boolean add
        if  root is null
                create root with new item
                assign false to red color of root
                return true;
        else
                new add item to root
                assign false to red color of root, root is always black.
                return boolean value;


private Node< E> add
        if compare item and localRoot.data equals zero
                item already in the tree.
                return localRoot;
        else if  compare item and localRoot.data less than zero
                if localRoot.left  is null
                        create new left child.
                        return localRoot;
```

```
            else
                    check for two red children, swap colors if found.
                    recursively add on the left.

            see whether the left child is now red
            if red color of left of root is equal true
                    if left of left of root is not null and red color of left of left of root is true
                    left-left grandchild is also red.

                    single rotation is necessary
                    .....
                    return rotate right

            else if right of left of root is not null  and red color of left of right of root is true
                    left-right grandchild is also red.
                    double rotation is necessary
                    .....
                    return rotate right

        return root;

    else
            item is greater than data of root

            if right of root is null
                    create new right child
                    .....
                    return root;
            else
                    check for two red children swap colors
                    recursively insert on the right
                    see if the right child is now red

                    if red color of right of root is true
                    if right of right of root is not null and red color of right of right of root is true

                            right-right grandchild is also red
                            single rotate is necessary
                            .....
                            return rotate left

            else if left of right of root is not null and red color of left of right is true
                    left-right grandchild is also red
                    double rotate is necessary
                    .....
                    return rotate left
```

private Node< E> fixupLeft, private Node< E> fixupRight, private E findLargestChild, private Node< E> findReplacement, private E removeFromRight, private E removeFromLeft, public E delete, private void moveBlackDown methodları da RedBlack Tree yapısı oluşturulurken kullanılmıştır.

1.2 Test Cases

Try this code least 2 different redblack tree. Report all of situations.

```
rbt_1.add(60);
rbt_1.add(100);
rbt_1.add(80);
rbt_1.add(160);
rbt_1.add(130);
rbt_1.add(90);
rbt_1.add(95);
rbt_1.add(115);
rbt_1.add(125);
rbt_1.add(120);
rbt_1.add(124);
rbt_1.add(122);
rbt_1.add(121);
rbt_1.add(123);
System.out.println(rbt_1.toString());

RedBlackTree<Integer> rbt_2 = new RedBlackTree<>();
rbt_2.add(1000);
rbt_2.add(100);
rbt_2.add(550);
rbt_2.add(2000);
rbt_2.add(1500);
rbt_2.add(4000);
rbt_2.add(3000);
rbt_2.add(6000);
rbt_2.add(5000);
rbt_2.add(8000);
rbt_2.add(7600);
rbt_2.add(9000);
rbt_2.add(8500);
rbt_2.add(9500);
System.out.println(rbt_2.toString());
```

Oluşturduğum redblack tree objelerine add yaparken kök düğümün her zaman için siyah, herhangi bir düğümden, yapraklara kadar uzanan herhangi bir yolda, eşit sayıda siyah düğüm olması ve bir kırmızı düğümün kırmızı çoçuğu bulunamaz ilkelerini kullandım. Bu durumlardan birisi gerçekleştiğinde ağaçtaki düğümlerin rengini değiştirdim ya da değiştirilemiyorsa ağaçta dengeleme yapmak için rotation işlemleri gerçekleştirildi.

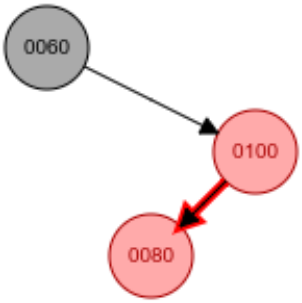
Yüksekliği 6 olan worst RedBlack Tree istendiği için her objeye 14 ekleme yapıp 6 yüksekliğine ulaşabildim. Worst olması içinde en çok rotate işleminin gerçekleşeceği elemanlar ekledim.

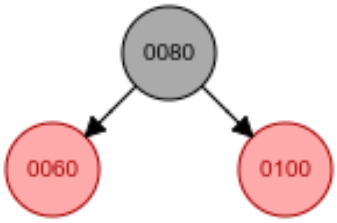
1.3 Running Commands and Results

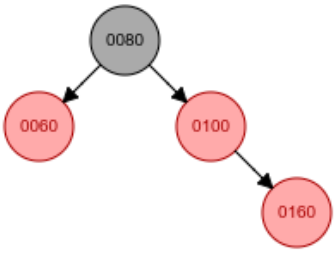
İlk oluşturduğum objeye add methodu ile eklediğim ağacın terminal çıktısı

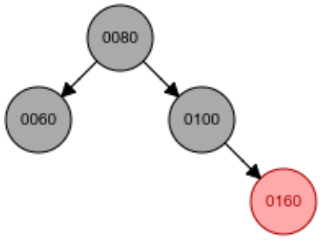
Black: 95
  Black: 80
    Black: 60
      null
      null
    Black: 90
      null
      null
  Red : 124
    Black: 115
      Black: 100
        null
        null
      Red : 121
        Black: 120
          null
          null
        Black: 122
          null
          Red : 123
            null
            null
    Black: 130
      Black: 125
        null
        null
      Black: 160
        null
        null

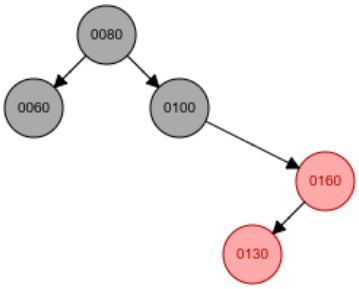
Animasyonda adım adım ağacın oluşturulması

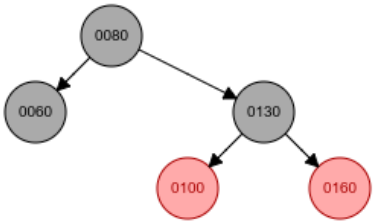




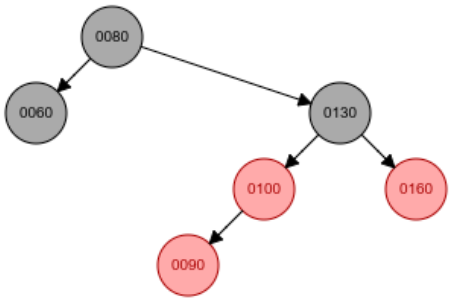


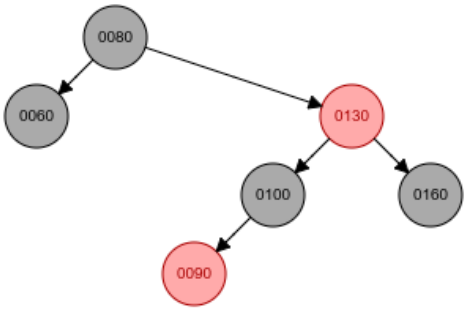


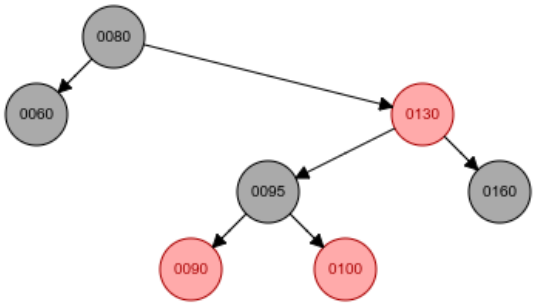


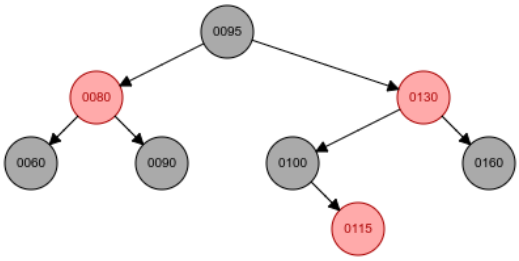


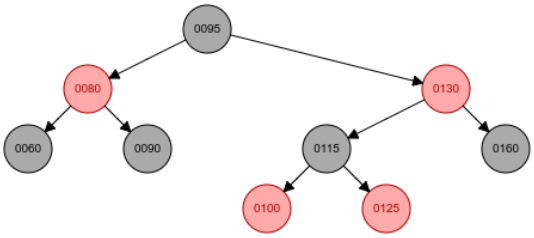
down from grandparent

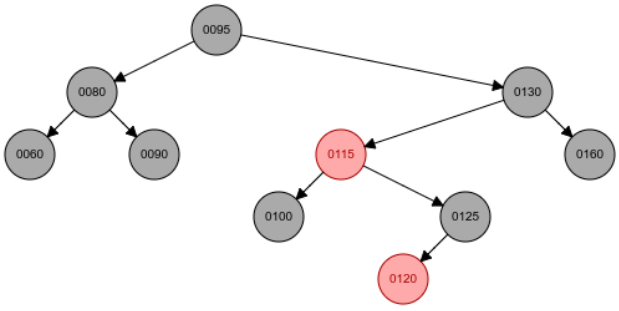


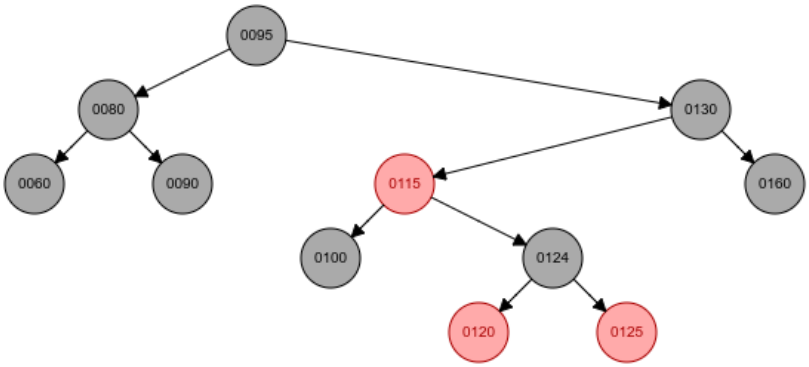

0080
0060
0130
0100
0160
0090

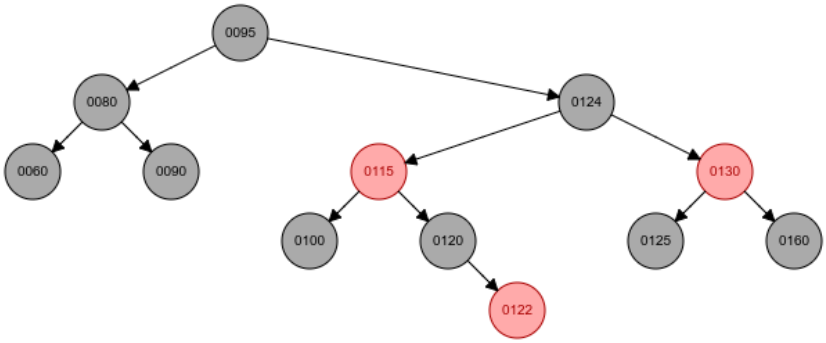

0080
0060
0130
0095
0160
0090
0100

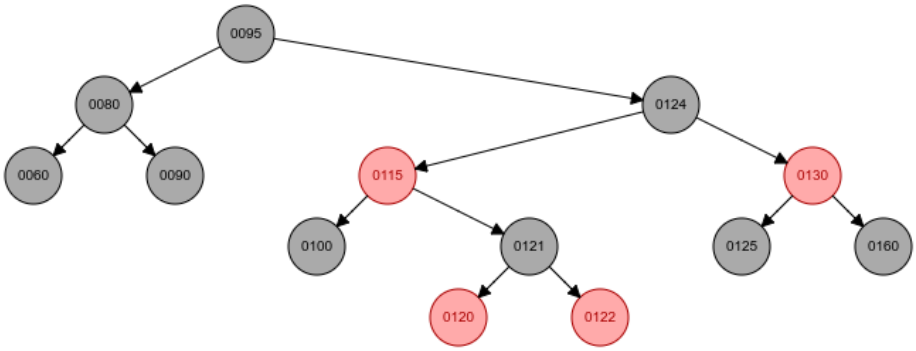

0095
0080
0130
0060
0090
0100
0160
0115

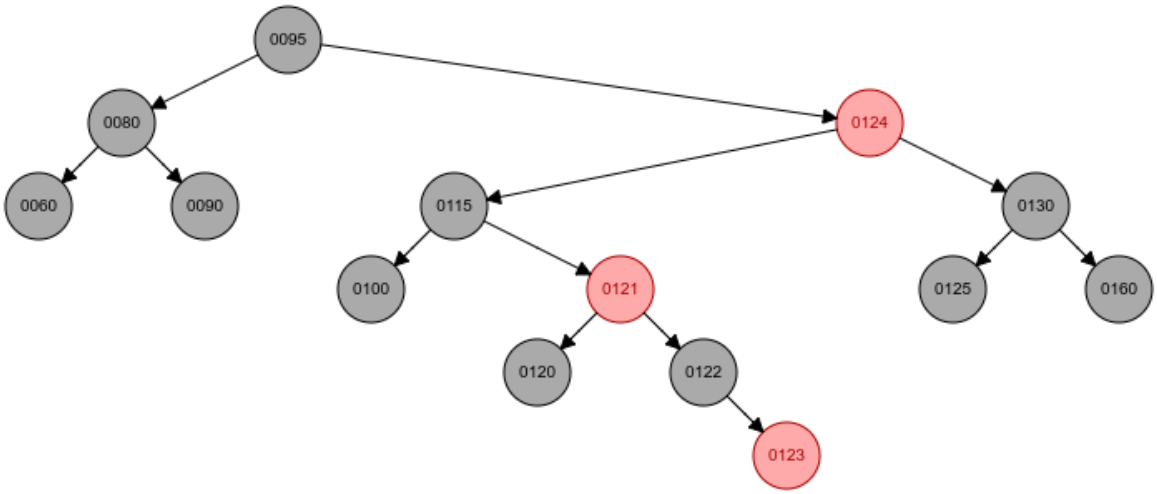

0095
0080
0130
0060
0090
0115
0160
0100
0125


0095
0080
0130
0060
0090
0115
0160
0100
0125
0120


0095
0080
0130
0060
0090
0115
0160
0100
0124
0120
0125


0095
0080
0124
0060
0090
0115
0130
0100
0120
0125
0160
0122

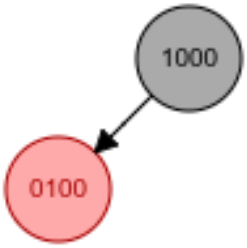

0095
0080
0124
0060
0090
0115
0130
0100
0121
0125
0160
0120
0122

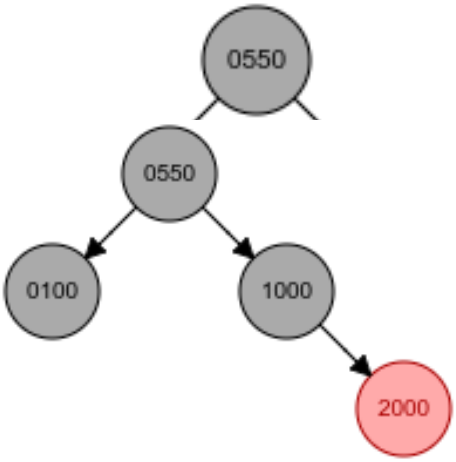

0095
0080
0124
0060
0090
0115
0130
0100
0121
0125
0160
0120
0122
0123

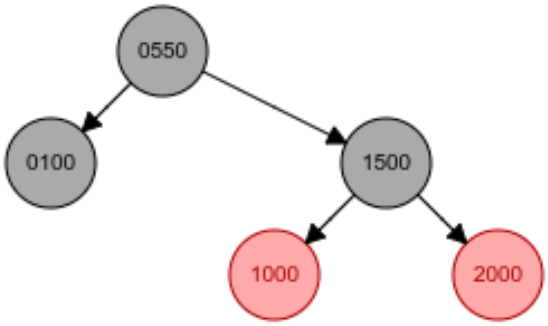
İkinci oluşturduğum objeye add methodu ile eklediğim ağacın terminal çıktısı

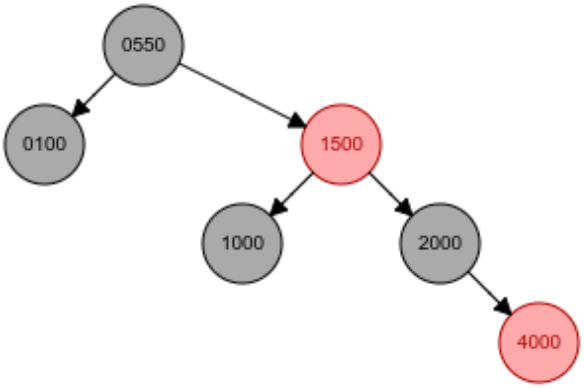
Black: 1500
  Black: 550
    Black: 100
      null
      null
    Black: 1000
      null
      null
  Red : 5000
    Black: 3000
      Black: 2000
        null
        null
      Black: 4000
        null
        null
    Black: 7600
      Black: 6000
        null
        null
      Red : 8500
        Black: 8000
          null
          null
        Black: 9000
          null
          Red : 9500
            null
            null

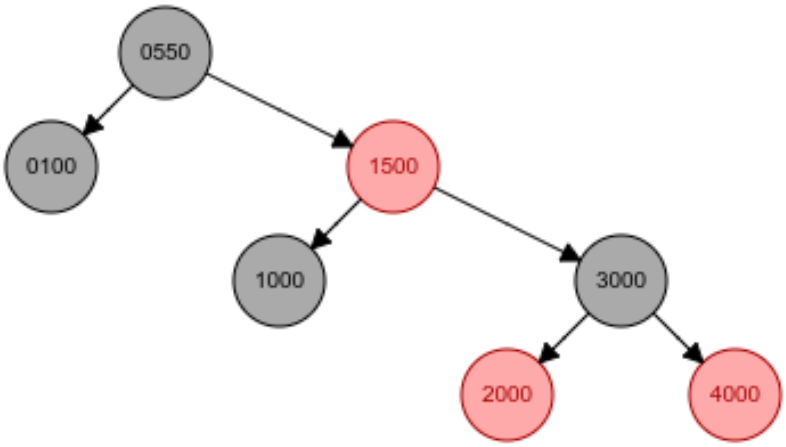
Animasyonda adım adım ağacın oluşturulması

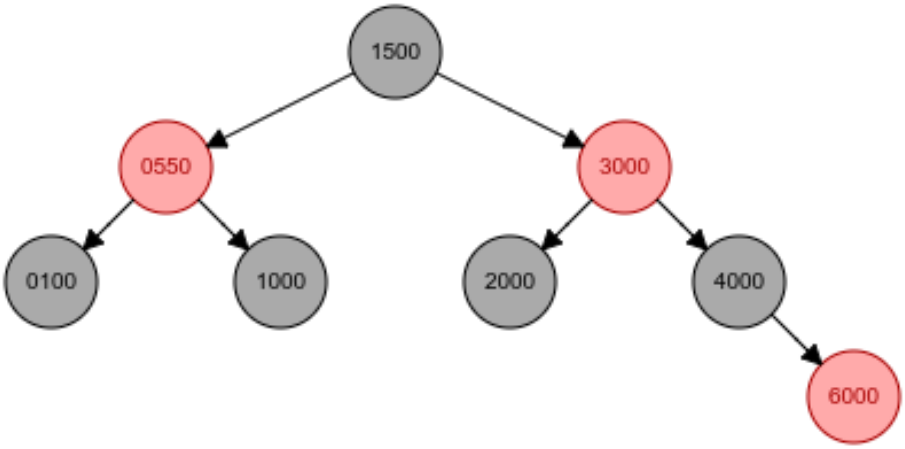


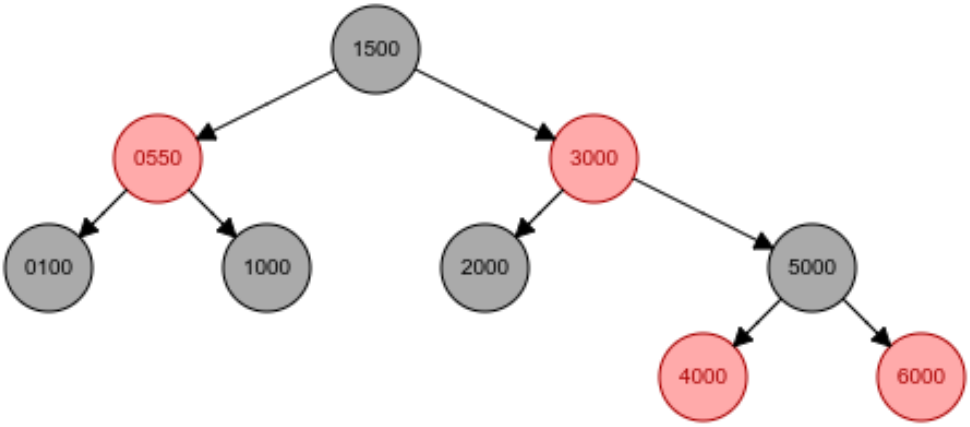

0550
0550
0100
1000
2000

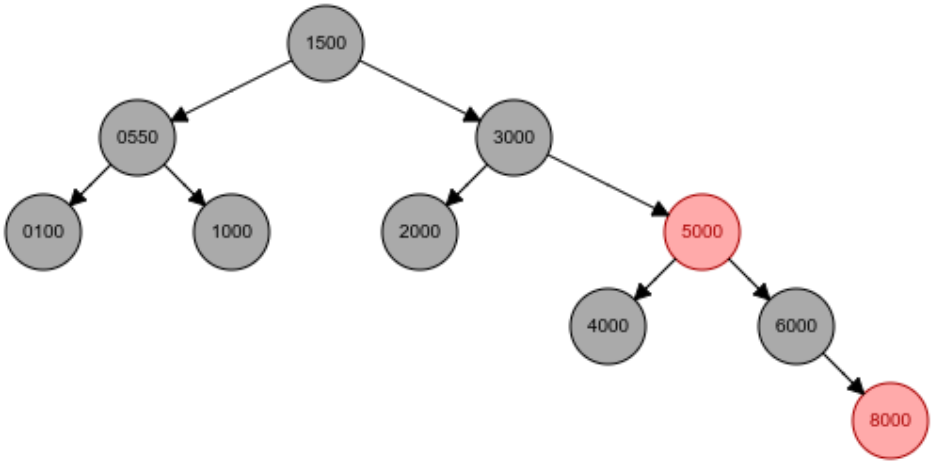

0550
0100
1500
1000
2000

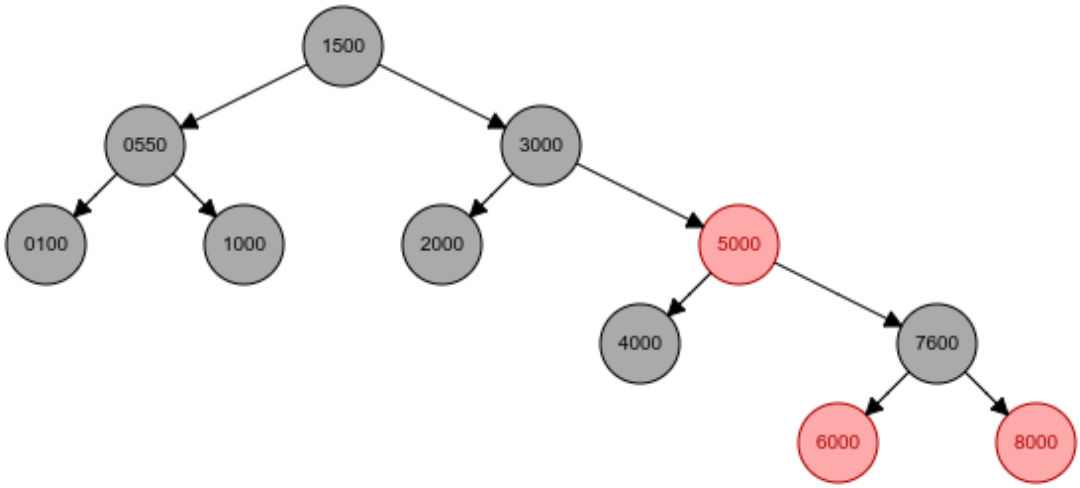

0550
0100
1500
1000
2000
4000

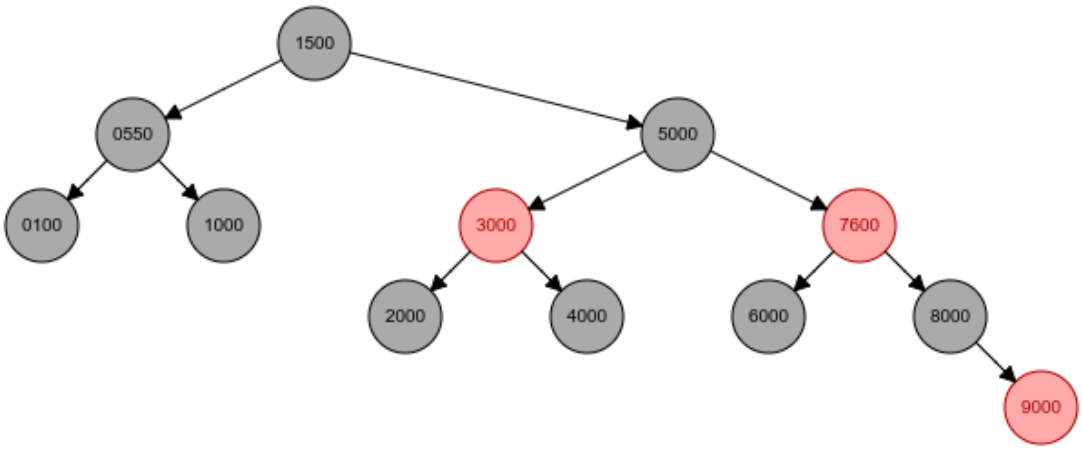

0550
0100
1500
1000
3000
2000
4000

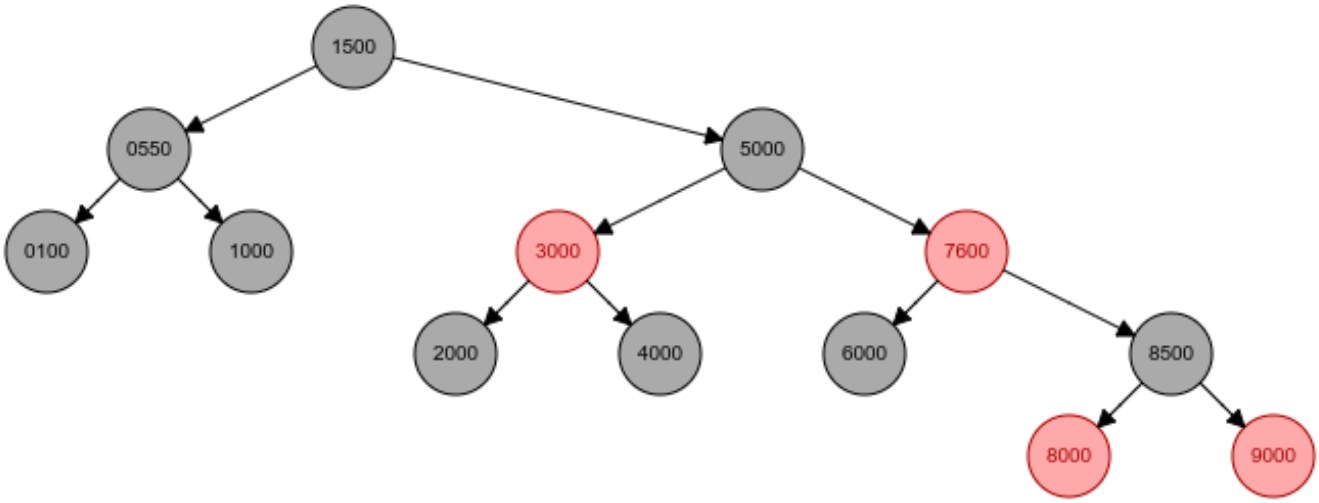

1500
0550
3000
0100
1000
2000
4000
6000

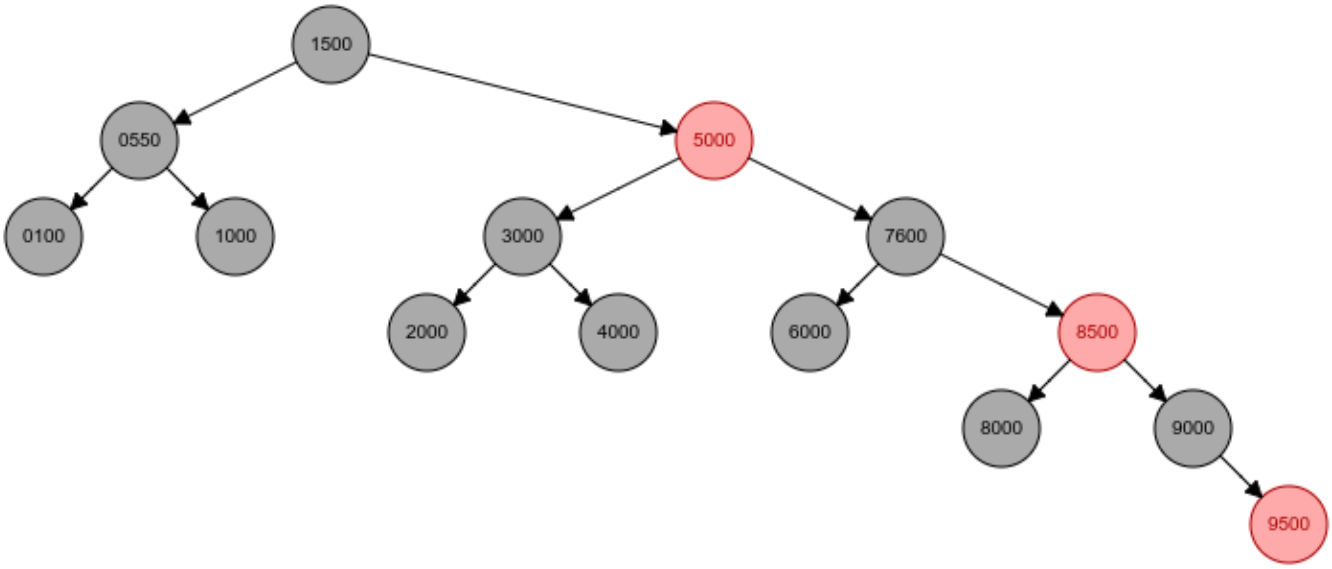

1500
0550
3000
0100
1000
2000
5000
4000
6000


1500
0550
3000
0100
1000
2000
5000
4000
6000
8000


1500
0550
3000
0100
1000
2000
5000
4000
7600
6000
8000


1500
0550
5000
0100
1000
3000
7600
2000
4000
6000
8000
9000


1500
0550
5000
0100
1000
3000
7600
2000
4000
6000
8500
8000
9000


1500
0550
5000
0100
1000
3000
7600
2000
4000
6000
8500
8000
9000
9500

2 binarySearch method

This part about Question2 in HW6

2.1 Problem Solution Approach

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

- Her düğümün en fazla m çocuğu bulunmalıdır.
- Root ve leaf düğümler haricindeki her düğümün en az m/2 adet elemanı bulunmalıdır.
- Bütün yapraklar aynı seviyede olmak zorundadır.
- Herhangi bir düğümde k çocuk bulunuyorsa k-1 elemanı gösteren anahtar bulunmalıdır.

Btree için bu 4 maddeden yola çıkarak Btree yapıma eklemeler yaptım.

Ekleme sırasında her node'un sahip olabileceği çocuk sayısı aşılırsa düğümün çoğalıp çoğalmadığını kontrol ettim.

Bir yaprağın seviyesi düşmesi durumunda yani daha yukarı çıkması veya daha sığ olması durumunda ağaçtaki yapısal değişiklikleri kontrol ettim.

BinarySearch method pseudocode

```
private int binarySearch(E item, E[] data, int startSize, int rootSize)
        if  startSize is greater than rootSize
                return startSize
        else
                calculate middle index
                (startSize + rootSize) / 2)
                compare item and element of data in the middle index

                if compare value equals zero
                        return middle index
                else if compare is less than zero
                        recursive call and return for startSize and middleIndex
                else
                        recursive call and return middleIndex+1 and rootSize
```

Bu yötemde recursive çagrı yaptığım için öncelikli olarak startSize ve rootSize karşılaştırarak base case oluşturdum. startSize, rootSize'dan büyük veya eşit ise startSize değerini return ettim.

Daha sonra startSize ile rootSize değerlenirini toplayıp 2'ye bölerek ortanca değeri buldum ve eklenecek elemanı bu ortanca değere karşılık gelen indeksteki değerle karşılaştırarak sonucun sıfırdan büyük veya küçük olmasına göre recursive çağrımı yaptım.

2.2 Test Cases

Try this code to search least 4 element on 2 different BTree. Report all of situations.

```
BTree<Integer> btree_1 = new BTree<>(5);
btree_1.add(10);
btree_1.add(22);
btree_1.add(30);
btree_1.add(40);
btree_1.add(5);
btree_1.add(7);
btree_1.add(8);
btree_1.add(13);
btree_1.add(15);
btree_1.add(18);
btree_1.add(20);
btree_1.add(26);
btree_1.add(27);
btree_1.add(32);
btree_1.add(35);
btree_1.add(38);
btree_1.add(42);
btree_1.add(46);

System.out.println(btree_1.toString());

System.out.println("35 searching...");
if (btree_1.contains(35)) {
   System.out.println(btree_1.find(35) + "found");
} else {
   System.out.println("35 didn't find");
}

System.out.println("7 searching...");
if (btree_1.contains(7)) {
   System.out.println(btree_1.find(7) + "found");
} else {
   System.out.println("7 didn't find");
}

System.out.println("100 searching...");
if (btree_1.contains(100)) {
   System.out.println(btree_1.find(100) + "found");
} else {
   System.out.println("100 didn't find");
}
System.out.println("46 searching...");
if (btree_1.contains(46)) {
   System.out.println(btree_1.find(46) + "found");
} else {
   System.out.println("46 didn't find");
}
```

```
BTree<Integer> btree_2 = new Btree<>(7);

btree_2.add(11);
btree_2.add(24);
btree_2.add(38);
btree_2.add(45);
btree_2.add(56);
btree_2.add(74);
btree_2.add(81);
btree_2.add(19);
btree_2.add(14);
btree_2.add(88);
btree_2.add(29);
btree_2.add(26);
btree_2.add(100);
btree_2.add(250);
btree_2.add(150);
btree_2.add(138);

System.out.println(btree_2.toString());
System.out.println("38 searching...");

if (btree_2.contains(38)) {
   System.out.println(btree_1.find(38) + "found");
} else {
   System.out.println("38 didn't find");
}

System.out.println("26 searching...");

if (btree_2.contains(26)) {
   System.out.println(btree_1.find(26) + "found");
} else {
   System.out.println("26 didn't find");
}

System.out.println("959 searching...");

if (btree_2.contains(959)) {
   System.out.println(btree_1.find(959) + "found");
} else {
   System.out.println("959 didn't find");
}

System.out.println("872 searching...");

if (btree_2.contains(872)) {
   System.out.println(btree_1.find(872) + "found");
} else {
   System.out.println("872 didn't find");
}
```

2.3 Running Commands and Results

btree_1 objesi terminal çıktısı

```
22
  8, 15
    5, 7
      null
      null
      null

    10, 13
      null
      null
      null

    18, 20
      null
      null
      null

  30, 38
    26, 27
      null
      null
      null

    32, 35
      null
      null
      null


    40, 42, 46
      null
      null
      null
      null
```

btree1 objesi için 4 elamanın aranmasının terminal çıktısı

```
35 searching...
35found

7 searching...
7found

100 searching...
100 didn't find

46 searching...
46found
```

btree_2 objesi terminal çıktısı

```
24, 45, 88
  11, 14, 19
    null
    null
    null
    null

  26, 29, 38
    null
    null
    null
    null

  56, 74, 81
    null
    null
    null
    null

  100, 138, 150, 250
    null
    null
    null
    null
    null
```

btree2 objesi için 4 elamanın aranmasının terminal çıktısı

```
38 searching...
38found

26 searching...
26found

959 searching...
959 didn't find

872 searching...
872 didn't find
```

3 Project 9.5 in book

This part about Question3 in HW6

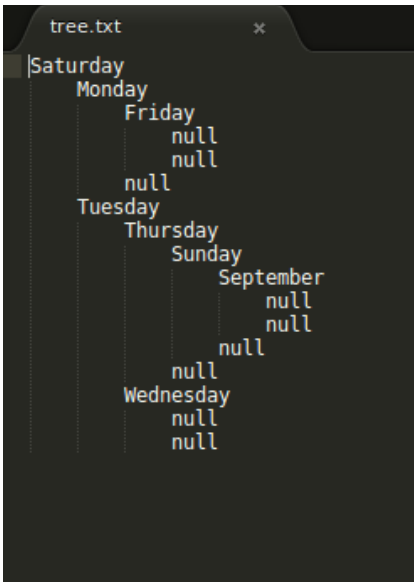
3.1 Problem Solution Approach

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

AVL Tree sınıfımın Binary Tree alan constructor'ını şu şekilde oluşturdum.

```
public AVLTree(BinaryTree<E> tree) {
        .....
        .....
 }
```

Mainde Binary Tree sınıfının readBinaryTree yöntemini kullanrak txt dosyasına yazdığım ağaç yapısının tree yapısına eklemenmesini sağladım.




```
fileRead = new FileReader("tree.txt");
bufRead = new BufferedReader(fileRead);

BinaryTree<String> readBinaryTree;

readBinaryTree =BinaryTree.readBinaryTree(bufRead);
```

public AVLTree(BinaryTree<E> tree) constructor methodu içinde Binary Tree de elemanları AVL Tree ye ekleyip dengeyi sağlamak için **addBinaryTreeToAVL** methodu yazdım.
Pseudocode yapısı şu şekildedir.

```
        private boolean addBinaryTreeToAVL(BinaryTree<E> tree) {
                if tree is not null
                        call add method and return value
                        sent data of tree to add method
                        recursive call  addBinaryTreeToAVL for left of tree
                        recursive call  addBinaryTreeToAVL for right of tree

                return false
```

Ağacın dengesizlik durumunda decrementBalance, incrementBalance, rebalanceleft, and rebalanceRight methodlarının çalışıp çalışmadığını kontrol edebilmek için şu değişkenkleri tanımladım.

```
private boolean decBalance;
private boolean incBalance;
private boolean reLeft;
private boolean reRight;
```

decBalance değişkenini decrementBalance methodu içinde, incBalance değişkenini incrementBalance methodu içinde, reLeft değişkenini rebalanceleft methodu içinde, reRight değişkenini rebalanceRight methodu içinde kullanarak rotate ve balance durumlarını kontrol ettim.

3.2 Test Cases

```
try {

        BufferedReader bufRead = null;
        FileReader fileRead = null;
        fileRead = new FileReader("tree.txt");
        bufRead = new BufferedReader(fileRead);

        BinaryTree<String> readBinaryTree = BinaryTree.readBinaryTree(bufRead);
        System.out.println(readBinaryTree.toString());
        AVLTree<String> avlTree = new AVLTree<>(readBinaryTree);
        System.out.println(avlTree.toString());

} catch (IOException e) {
        e.printStackTrace();
}
```

addBinaryTreeToAVL methodu ile işlem yapıldıktan sonra

```
if (reRight) {
        System.out.println("Ağacın sağ tarafı yeniden dengelenmiştir.");
}

if (incBalance) {
        System.out.println("Sağ tarafa ekleme yapılırken denge değerinde artma olmuştur.");
}

if (reLeft) {
        System.out.println("Ağacın sol tarafı yeniden dengelenmiştir.");
}

if (decBalance) {
        System.out.println("Sol tarafa ekleme yapılırken denge değerinde azalma
        olmuştur.");
}
```

3.3 Running Commands and Results

Show that test case results using screenshots.

Dosyadan okunan Binary Tree Ağaç Yapısı

Saturday
  Monday
    Friday
      null
      null
    null
  Tuesday
    Thursday
      Sunday
        September
          null
          null
        null
      null
    Wednesday
      null
      null

AVL Tree Yapısına eklendikten Sonra Dengeleme İşlemi

Ağacın sağ tarafı yeniden dengelenmiştir.
Sağ tarafa ekleme yapılırken denge değerinde artma olmuştur.
Ağacın sol tarafı yeniden dengelenmiştir.
Sol tarafa ekleme yapılırken denge değerinde azalma olmuştur.

1: Saturday
  -1: Monday
    0: Friday
      null
      null
    null
  0: Thursday
    -1: Sunday
      0: September
        null
        null
      null
    1: Tuesday
      null
      0: Wednesday
        null
        null