

**Gebze Technical University
Computer Engineering**

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

**MESUT BUNALDI
111044077**

Course Assistant: Fatma Nur Esirci

1 Worst RedBlack Tree

1.1 Problem Solution Approach

Kırmızı-siyah ağaçlar (red-black trees) tanım itibariyle ikili arama ağaçlarıdır (binary search tree) ve bu anlamda, herhangi bir düğümün solunda kendisinden küçük ve sağında ise büyük verilerin durması beklenir. Ağaçta ayrıca her düğüm için bir renk özelliği tutulur. Yani bir düğüm kırmızı veya siyah renk özelliği taşıyabilir. Ağaçtaki düğümlerin taşımı gereken bu özellikler aşağıdaki şekilde sıralanabilir:

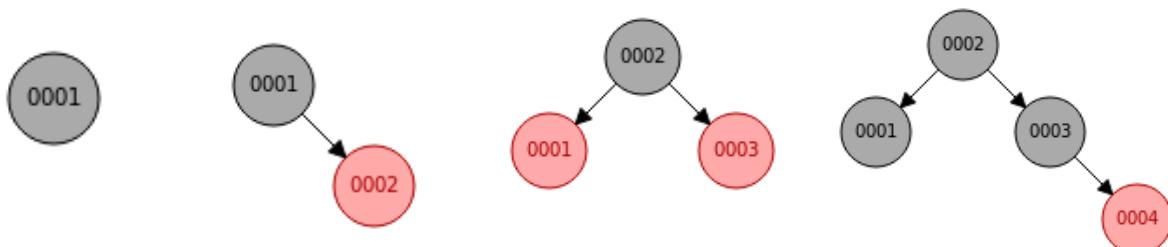
1. Ağaçtaki her düğüm kırmızı ya da siyahdır
2. Kök düğüm (root node) her zaman için siyahdır.
3. Bütün yaprak düğümler (leaf nodes) siyahdır
4. Herhangi bir kırmızı düğümün bütün çocukları siyahdır.
5. Herhangi bir düğümden, yaprak düzgüme kadar gidilen bütün yollarda eşit sayıda siyah düğüm bulunur.

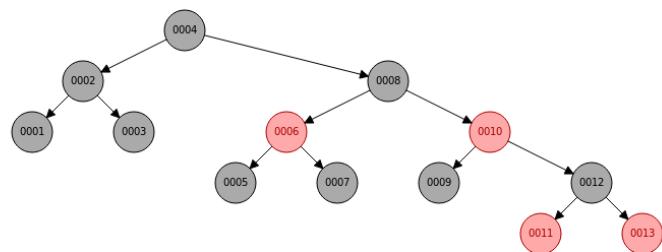
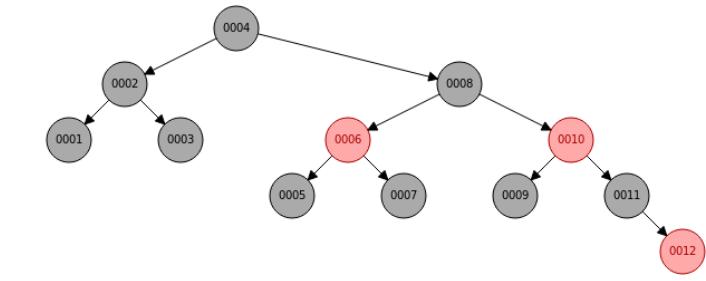
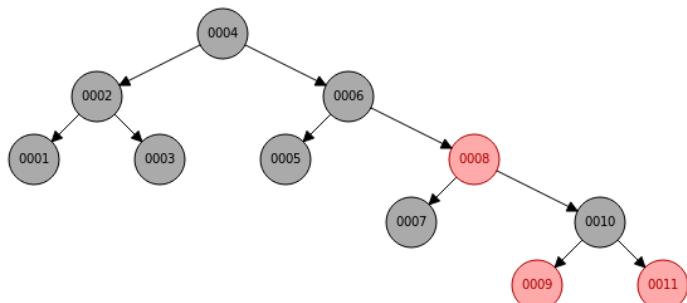
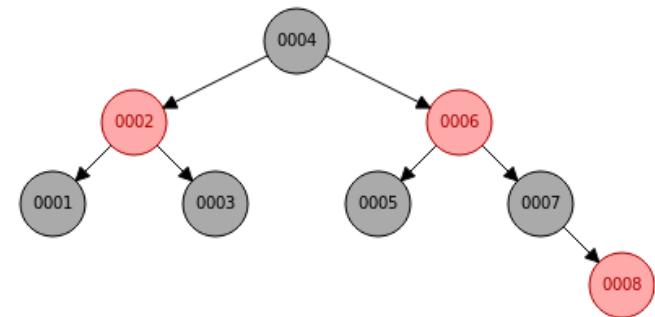
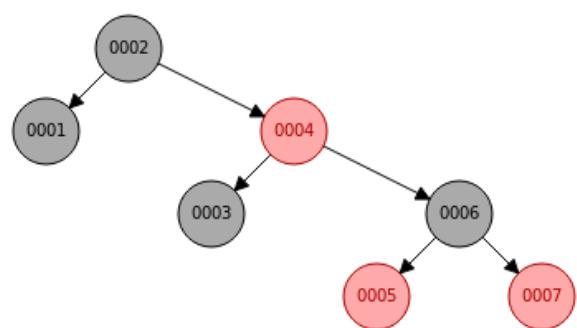
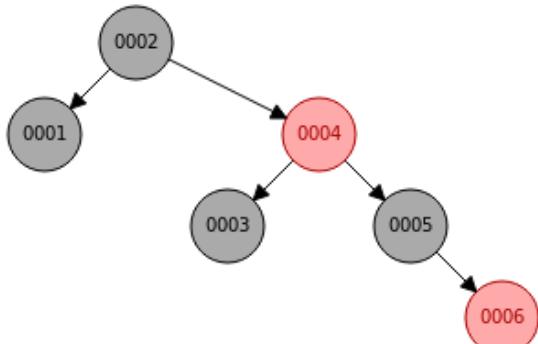
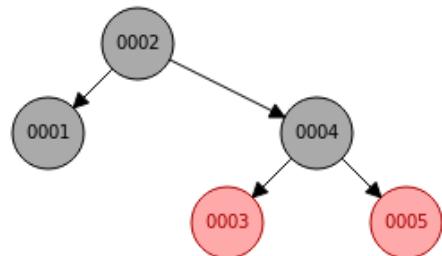
6 yüksekliğine sahip worst RedBlack Tree oluşturabilmek için en az 14 ekleme yapılması gerekmektedir. Bu ekleme işlemi Q1 package içerisinde main class içerisinde for döngüsü yardımı ile oluşturulmuş olan RedBlackTree içerisinde eklenmiş ve sonuç olarak 6 yüksekliğine sahip RedBlackTree yapısı elde edilmiştir.

1.2 Test Cases

-1'den 14 'e kadar olan sayıların RedBlackTree üzerine eklenmesi sonucunda 6 yüksekliğinde RedBlackTree elde edilmiştir.

-14'den 1'e kadar olan sayıların RedBlackTree üzerine eklenmesi sonucunda yine 6 yüksekliğinde RedBlackTree elde edilmiştir.





1.3 Running Commands and Results

```
public class main
{
    public static void main(String[] args){
        RedBlackTree<Integer> redBlackTree=new RedBlackTree<>();
        for (int i = 0; i <14 ; i++) {
            redBlackTree.add(i);
        }
        System.out.println(redBlackTree.toString());
    }
}
```

```
/home/jupiter/Documents/jdk1.8.0_161/bin/java ...
Black: 3
  Black: 1
    Black: 0
      null
      null
    Black: 2
      null
      null
  Red : 7
    Black: 5
      Black: 4
        null
        null
      Black: 6
        null
        null
    Black: 9
      Black: 8
        null
        null
    Red : 11
      Black: 10
        null
        null
      Black: 12
        null
        Red : 13
          null
          null

Process finished with exit code 0
```

Red/Black Tree

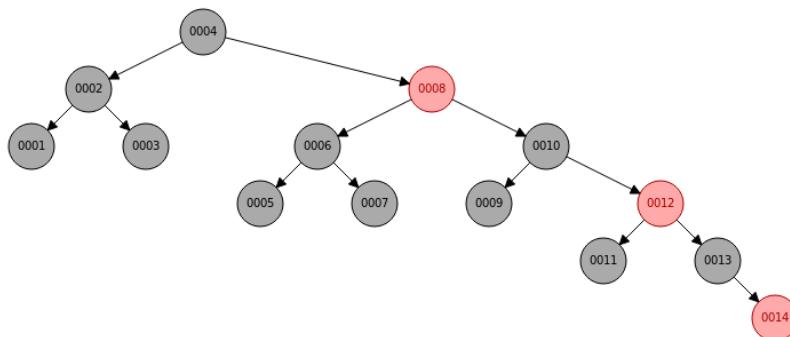
Insert

Delete

Find

Print

Show Null Leaves



Animation Completed

Skip Back Step Back Pause Step Forward Skip Forward Animation Speed W: 1000 H: 500 Change Canvas Size Move Controls

2 binarySearch method

This part about Question2 in HW6

2.1 Problem Solution Approach

BTree'ye ait binarySearch fonksiyonu implementasyonu Recursion konusunda işlemiş olduğumuz binary search metodu dına çok benzemektedir.

```
private int binarySearch(E target, E[] items, int first, int last)
```

Bu fonksiyon aracılığı ile ekleme yapacağımız düğümün index değerini elde ederiz. Böylece istediğimiz düğümü olması gereken yere ekleyebiliriz.

First: başlangıç indeksi

Last: son indeks

middle: medyan

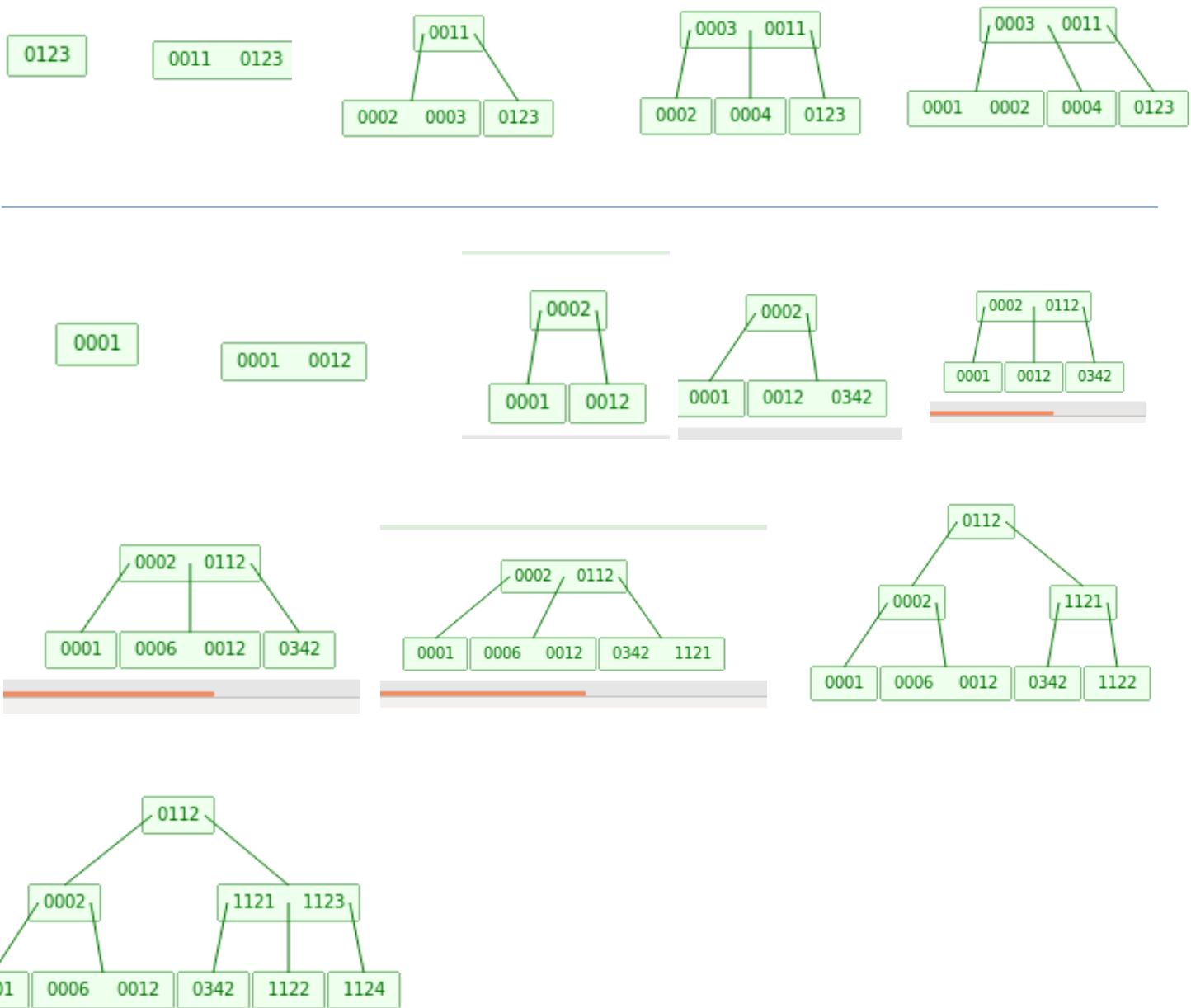
target: eklenmek istenen eleman

compResult :target ile medyan ve küçükse medyanın bir önceki elemanı ya da medyan ve büyükse medyanın bir sonraki elemanı ile karşılaştırma yapılması sonucu oluşan değer.

Recursive BinarySearch fonksiyonu :

```
if (first >= last)
    return first
else
    middle = (last-first) / 2
    compResult = target.compareTo(items[middle])
    if (compResult <0)
        return binarySearch( target,items, first, middle - 1)
    else if (compResult >0) {
        return binarySearch( target,items, middle + 1, last)
    else
        return middle
```

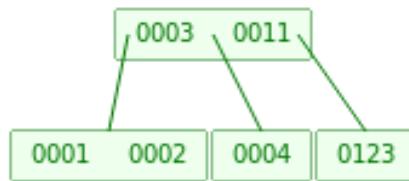
2.2 Cases



2.3 Running Commands and Results

```
BTree<Integer> bTree=new Btree<>(3);  
  
bTree.add(123);  
  
bTree.add(11);  
bTree.add(2);  
bTree.add(3);  
bTree.add(4);  
bTree.add(1);  
System.out.println(bTree.toString());  
BTree<Integer> bTree2=new BTree<>(6);  
bTree2.add(1);  
bTree2.add(12);  
bTree2.add(2);  
bTree2.add(342);  
bTree2.add(112);  
bTree2.add(1121);  
bTree2.add(1122);  
bTree2.add(1123);  
bTree2.add(1124);  
System.out.println(bTree2.toString());
```

```
/home/jupiter/Documents/jdk1.8.0_161/bin/java  
3, 11  
 1, 2  
   null  
   null  
   null  
  
 4  
   null  
   null  
  
123  
  null  
  null  
  
  
12, 1121  
 2, 1  
   null  
   null  
   null  
  
112, 342  
  null  
  null  
  null  
  
1122, 1123, 1124  
  null  
  null  
  null  
  null
```



3 Project 9.5 in book

3.1 Problem Solution Approach

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

```

function decrementBalance(AVLNode < E > node)
    node.balance--
    if (node.balance == AVLNode.BALANCED)
        increase = false

```

```

function incrementBalance(AVLNode < E > node)
    node.balance++
    if (node.balance > AVLNode.BALANCED)
        increase = true;
        decrease = false;
    else
        increase = false
        decrease = true

```

```

function rebalanceRight(AVLNode < E > localRoot)

    AVLNode < E > rightChild = (AVLNode < E > ) localRoot.right

    if (rightChild.balance < AVLNode.BALANCED)
        AVLNode < E > rightLeftChild = (AVLNode < E > ) rightChild.left
        if (rightLeftChild.balance > AVLNode.BALANCED)
            rightChild.balance = AVLNode.BALANCED
            rightLeftChild.balance = AVLNode.BALANCED
            localRoot.balance = AVLNode.LEFT_HEAVY

        else if (rightLeftChild.balance < AVLNode.BALANCED)

```

```

        rightChild.balance = AVLNode.RIGHT_HEAVY
        rightLeftChild.balance = AVLNode.BALANCED
        localRoot.balance = AVLNode.BALANCED
    else
        rightChild.balance = AVLNode.BALANCED
        rightLeftChild.balance = AVLNode.BALANCED
        localRoot.balance = AVLNode.BALANCED
        increase = false
        decrease = true
        localRoot.right = rotateRight(rightChild)
        return (AVLNode < E > ) rotateLeft(localRoot)
else
    rightChild.balance = AVLNode.BALANCED
    localRoot.balance = AVLNode.BALANCED
    increase = false
    decrease = true
    return (AVLNode < E > ) rotateLeft(localRoot)

```

```

function rebalanceLeftR(AVLNode < E > localRoot)
AVLNode < E > leftChild = (AVLNode < E > ) localRoot.left
if (leftChild.balance > AVLNode.BALANCED)

    AVLNode < E > leftRightChild = (AVLNode < E > ) leftChild.right
    if (leftRightChild.balance < AVLNode.BALANCED)
        leftChild.balance = AVLNode.LEFT_HEAVY
        leftRightChild.balance = AVLNode.BALANCED
        localRoot.balance = AVLNode.BALANCED

    else if (leftRightChild.balance > AVLNode.BALANCED)
        leftChild.balance = AVLNode.BALANCED
        leftRightChild.balance = AVLNode.BALANCED
        localRoot.balance = AVLNode.RIGHT_HEAVY

    else
        leftChild.balance = AVLNode.BALANCED
        localRoot.balance = AVLNode.BALANCED

    increase = false
    decrease = true
    // Perform double rotation
    localRoot.left = rotateLeft(leftChild)
    return (AVLNode < E > ) rotateRight(localRoot)

if (leftChild.balance < AVLNode.BALANCED)
    leftChild.balance = AVLNode.BALANCED
    localRoot.balance = AVLNode.BALANCED
    increase = false
    decrease = true

```

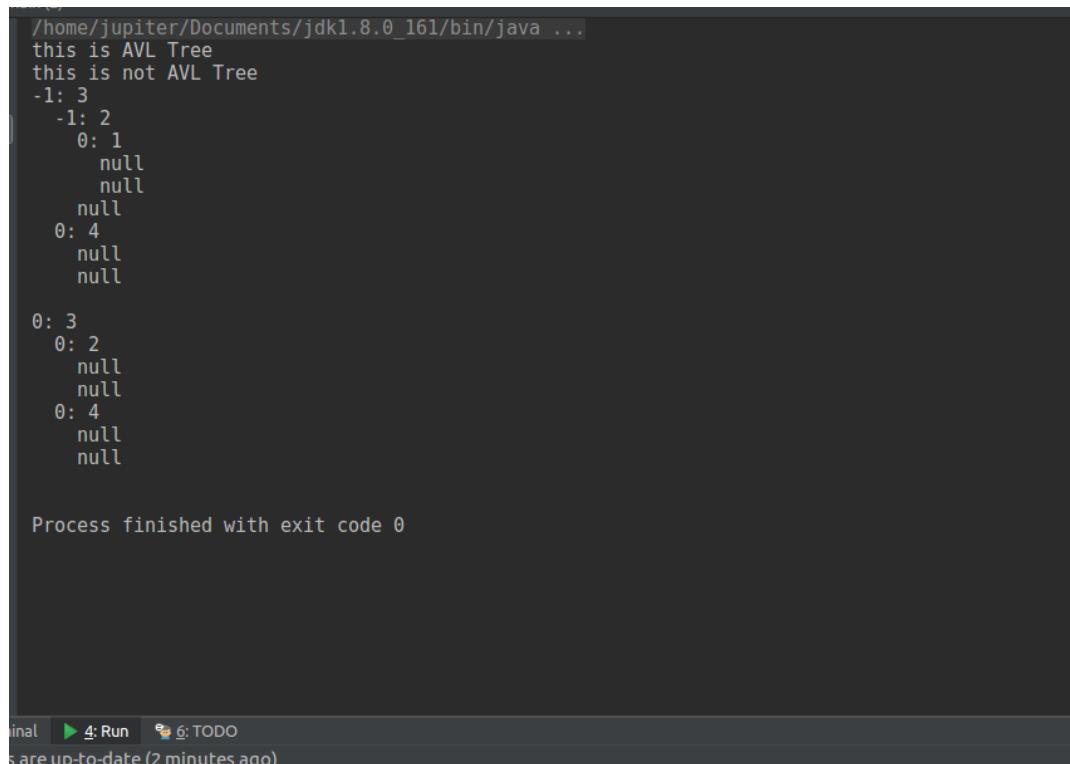
```
        else
            leftChild.balance = AVLNode.RIGHT_HEAVY
            localRoot.balance = AVLNode.LEFT_HEAVY

        return (AVLNode < E > ) rotateRight(localRoot)
```

3.2 Test Cases

Yukarıda kodu verilmiş olan tüm fonksiyonlar delete ve add fonksiyonları içinde kullanılmakta ve results kısmında da görüldüğü gibi doğru sonuçlar almamızı sağlamaktadır. Ayrıca AVLTree sınıfına ait BinaryTree parametresi alan bir constructor yazılmıştır. Çıktıdan da anlaşılacağı üzere bu constructor parametre olarak gönderilen tree nin AVL olup olmadığını kontrol etmekte ve çıktı olarak konsola bu bilgiyi yazmaktadır.

3.3 Running Commands and Results



The screenshot shows a terminal window with the following output:

```
/home/jupiter/Documents/jdk1.8.0_161/bin/java ...
this is AVL Tree
this is not AVL Tree
-1: 3
 -1: 2
  0: 1
   null
   null
  null
 0: 4
  null
  null

 0: 3
  0: 2
   null
   null
  0: 4
   null
   null

Process finished with exit code 0
```

At the bottom of the terminal window, there is a status bar with the following information:

- final
- ▶ 4: Run
- ⌚ 6: TODO
- Changes are up-to-date (2 minutes ago)