

**Gebze Technical University
ComputerEngineering**

CSE 222 -2018 Spring

HOMEWORK 6 REPORT

**BURAK ÖZDEMİR
141044027**

Course Assistant: Fatma Nur Esirci

1 WorstRedBlackTree

1.1 Problem Solution Approach

RedBlackTree ve RedBlackNode sınıfları kullanılmıştır. 6 yüksekliğindeki RedBlackTree ağacı worst case durumunda en az 14 node ile gerçekleştirilebilir. Worst Case durumu ağacı yüksekliği ile alakalı olduğu için her durumda $O(\log n)$ karmaşıklığındadır. Yükseklikte normal Binary ağactaki gibi hesaplanır.

1.2 Test Cases

FirstTree:

1 den 14 e kadar nodelar sırasıyla eklenmiştir. Ağac RedBlackTree genel kurallarına uyuyor. Visualization sitesinde denenmiştir. Yükseklik 6 dır. Worst Case durumunda yine $O(\log n)$ karmaşıklığı vardır.

SecondTree:

14 den 1 e kadar nodelar sırasıyla eklenmiştir. Ağac RedBlackTree genel kurallarına uyuyor. Visualization sitesinde denenmiştir. Yükseklik 6 dır. Worst Case durumunda yine $O(\log n)$ karmaşıklığı vardır.

1.3 Running Commands and Results

The image shows a screenshot of a Java IDE (IntelliJ IDEA) and a web browser displaying a Red/Black Tree visualization. The IDE window on the left shows the source code for a RedBlackTree class. The code includes methods for inserting nodes and printing the tree level by level. The main method inserts nodes from 14 down to 1. The output window shows the level-order traversal of the tree: 11(B) 7(R) 13(B) 5(B) 9(B) 12(B) 14(B) 3(R) 6(B) 8(B) 10(B) 2(B) 4(B) 1(R). The web browser window on the right shows the Red/Black Tree visualization site. The tree is displayed with nodes labeled with their binary representations (e.g., 0011, 0007, 0005, 0009, 0003, 0006, 0001, 0002, 0004, 0008, 0010, 0012, 0014). The tree has a height of 6.

Q1 Main Test

2 binarySearchmethod

2.1 Problem Solution Approach

Bu method 2 helper metodla çalışır . İlk helper method Node un elemanlarının içinde binarysearch yaparak işlemi gerçekleştirir.Eğer bulursa return edip işlemi sonlandırır . Eğer bulamazsa Nodu alt çocuklarının ilk elemanları ile karşılaştırır . Yani çocuklarında hepsine bakmıyarak yine logn karmaşıklığında bir işlem yaparak alt nodelara geçer ve tekrar ilk method çağırılır Bu şekilde devam eder .

```
-private boolean binarySearchInKeys(Bnode node ,int low,int high){
    If high is bigger than low
        Initilaze mid and assign (low +(high-low)/2)
        If key of node[mid] is equal to data
            Return true
        If key of node[mid] is smaller than data
            Return call binarySearchInKeys(node,mid+1,high,data)
        Else
            Return call binarySearchInKeys(node,low,mid-1,data)
    Return false
}

-private boolean binarySearchInChilds(Bnode root,int data){
    Initialize res and assign call BinarySearchInKeys(root,0,root.count-1,data)
    If res is equal to true
        Return true
    Initialize newChildNode and assign to -1
    Initialize i and assing to zero
    Initialize temp and ssign to root
    While child of temp[i] is not equal to null
        If child of temp[i] is not equal to null
            If child of temp[i+1] is not equal to null
                If key of child of temp[i][0] is smaller than data
                    And key of child of temp[i+1][0] is bigger than data

                    Assign i to newChildNode
                    Break
            Else
                Assing i to newChildOne
        I plus plus
    If newchildNode is equal to -1
        Return false
    Return call binarySearchInChilds(root.getChild(newChildNode),data)
}

+public boolean binarSearch(int data){
    Return binarySearchInChilds(root,data)
}
```

2.2 Test Cases

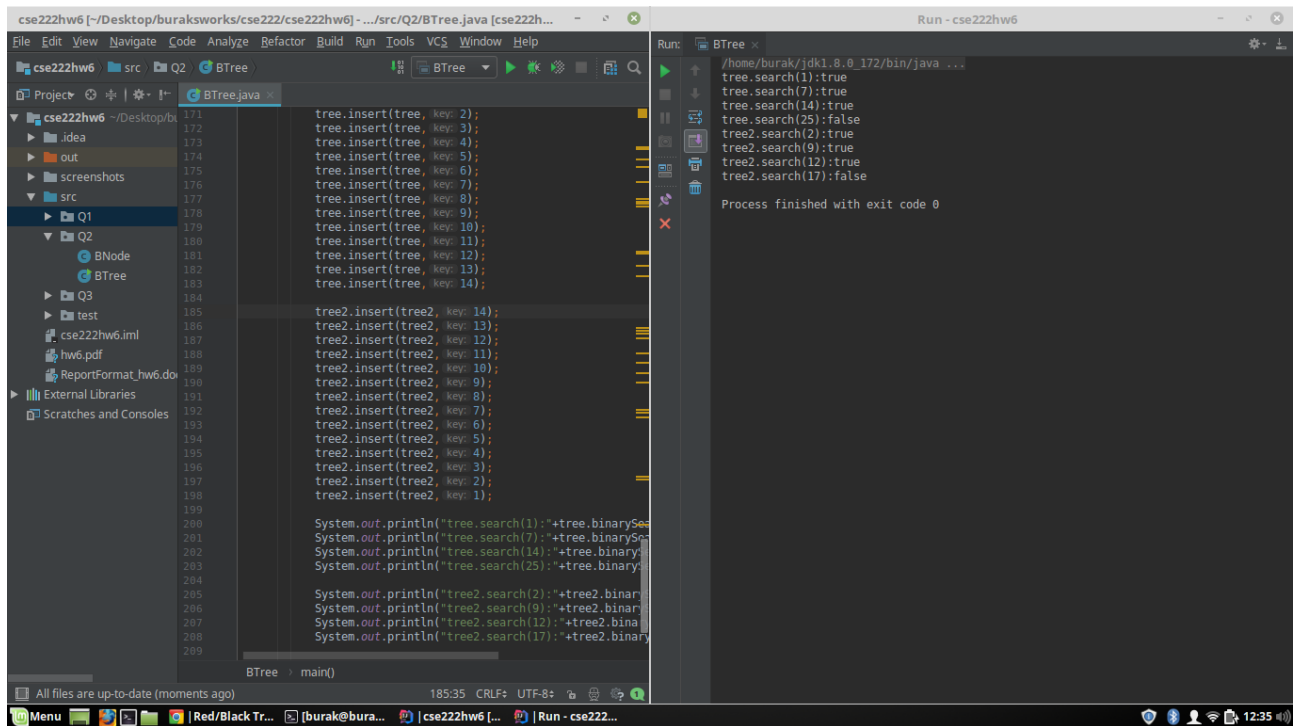
Tree1:

Ağaç 1 den 14 e kadar sayılarla doldurulmuştur . Sırasıyla 1 7 14 25 sayıları search edilmiştir .(Basarılı)

Tree2:

Ağaç 14 den 1 e kadar sayılarla doldurulmuştur . Sırasıyla 1 7 14 25 sayıları search edilmiştir .(Basarılı)

2.3 Running Commands and Results



```
tree.insert(tree, key: 2);
tree.insert(tree, key: 3);
tree.insert(tree, key: 4);
tree.insert(tree, key: 5);
tree.insert(tree, key: 6);
tree.insert(tree, key: 7);
tree.insert(tree, key: 8);
tree.insert(tree, key: 9);
tree.insert(tree, key: 10);
tree.insert(tree, key: 11);
tree.insert(tree, key: 12);
tree.insert(tree, key: 13);
tree.insert(tree, key: 14);

tree2.insert(tree2, key: 14);
tree2.insert(tree2, key: 13);
tree2.insert(tree2, key: 12);
tree2.insert(tree2, key: 11);
tree2.insert(tree2, key: 10);
tree2.insert(tree2, key: 9);
tree2.insert(tree2, key: 8);
tree2.insert(tree2, key: 7);
tree2.insert(tree2, key: 6);
tree2.insert(tree2, key: 5);
tree2.insert(tree2, key: 4);
tree2.insert(tree2, key: 3);
tree2.insert(tree2, key: 2);
tree2.insert(tree2, key: 1);

System.out.println("tree.search(1):"+tree.binarySearch(1));
System.out.println("tree.search(7):"+tree.binarySearch(7));
System.out.println("tree.search(14):"+tree.binarySearch(14));
System.out.println("tree.search(25):"+tree.binarySearch(25));

System.out.println("tree2.search(2):"+tree2.binarySearch(2));
System.out.println("tree2.search(9):"+tree2.binarySearch(9));
System.out.println("tree2.search(12):"+tree2.binarySearch(12));
System.out.println("tree2.search(17):"+tree2.binarySearch(17));
```

Q2 Main Test

3 Project 9.5 in book

3.1 Problem Solution Approach

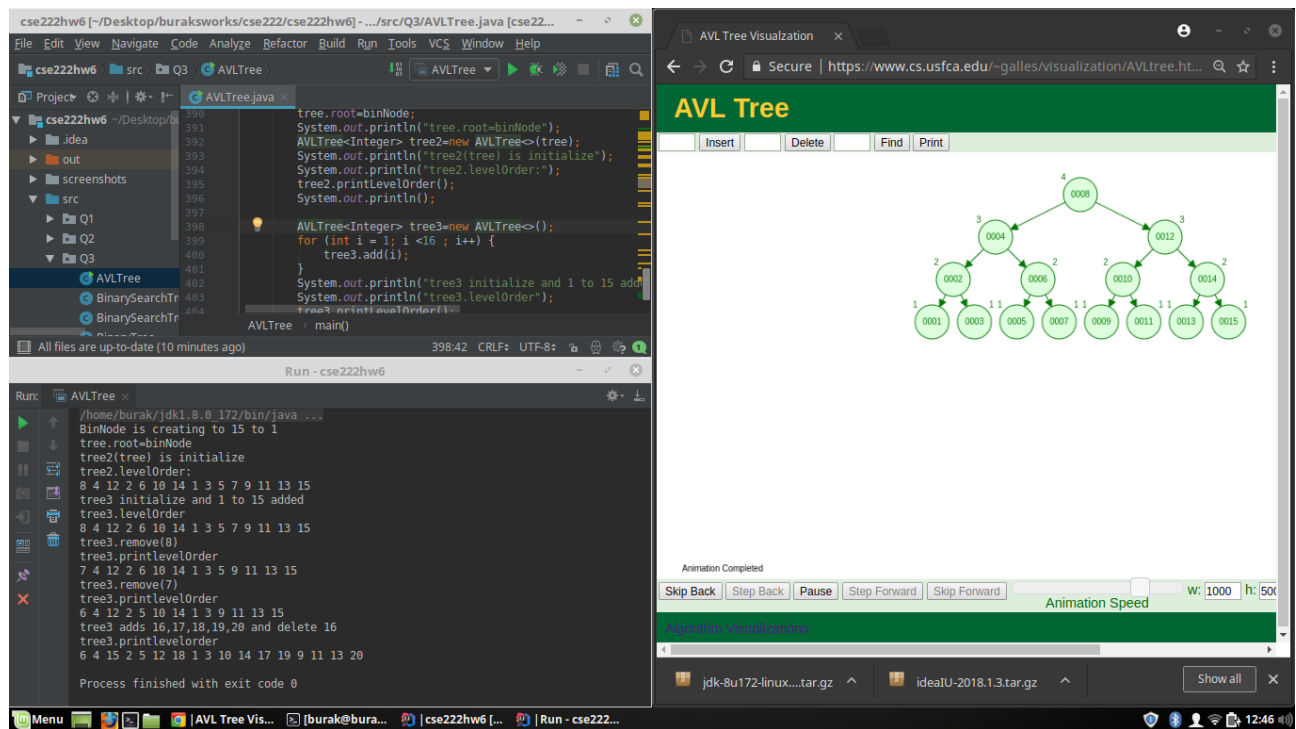
AVLTree sınıfı implement edildi. Constructor baksa bir binary tree alıyor ve kendisine göre düzenliyerek datafieldlarına yerleştiriyor. Gerekli metodlar implement edildi. Binarytree kontrolü constructor içinde ediliyor ve sonuc ekrana basılıyor.

3.2 Test Cases

Tree degiskenine surekli sol tarafa dogru giden bir node yapısı verildi . Daha sonra bu tree degiskeni tree2 yanı AVLTree yapısına constructor parametresi olarak verildi . Sınıf bu yapıyı

duzenliyerek kendisine uygun hale getirdi . Daha sonra tree3 degiskeni ile sınıfın add ve remove metodları test edildi .

3.3 RunningCommandsandResults



The screenshot shows an IDE (IntelliJ IDEA) on the left and a web browser on the right. The IDE displays the source code for `AVLTree.java` and the output of the program. The web browser shows the AVL Tree Visualization page with a tree diagram.

AVL Tree Visualization Page:

- Buttons: Insert, Delete, Find, Print
- Tree Diagram: A balanced AVL tree with root 0008. The tree structure is as follows:
 - Root: 0008 (BF 4)
 - Left child: 0004 (BF 3)
 - Right child: 0012 (BF 3)
 - Left child of 0004: 0002 (BF 2)
 - Right child of 0004: 0006 (BF 2)
 - Left child of 0012: 0010 (BF 2)
 - Right child of 0012: 0014 (BF 2)
 - Left child of 0002: 0001 (BF 1)
 - Right child of 0002: 0003 (BF 1)
 - Left child of 0006: 0005 (BF 1)
 - Right child of 0006: 0007 (BF 1)
 - Left child of 0010: 0009 (BF 1)
 - Right child of 0010: 0011 (BF 1)
 - Left child of 0014: 0013 (BF 1)
 - Right child of 0014: 0015 (BF 1)
- Animation Controls: Skip Back, Step Back, Pause, Step Forward, Skip Forward, Animation Speed (W: 1000, H: 500)

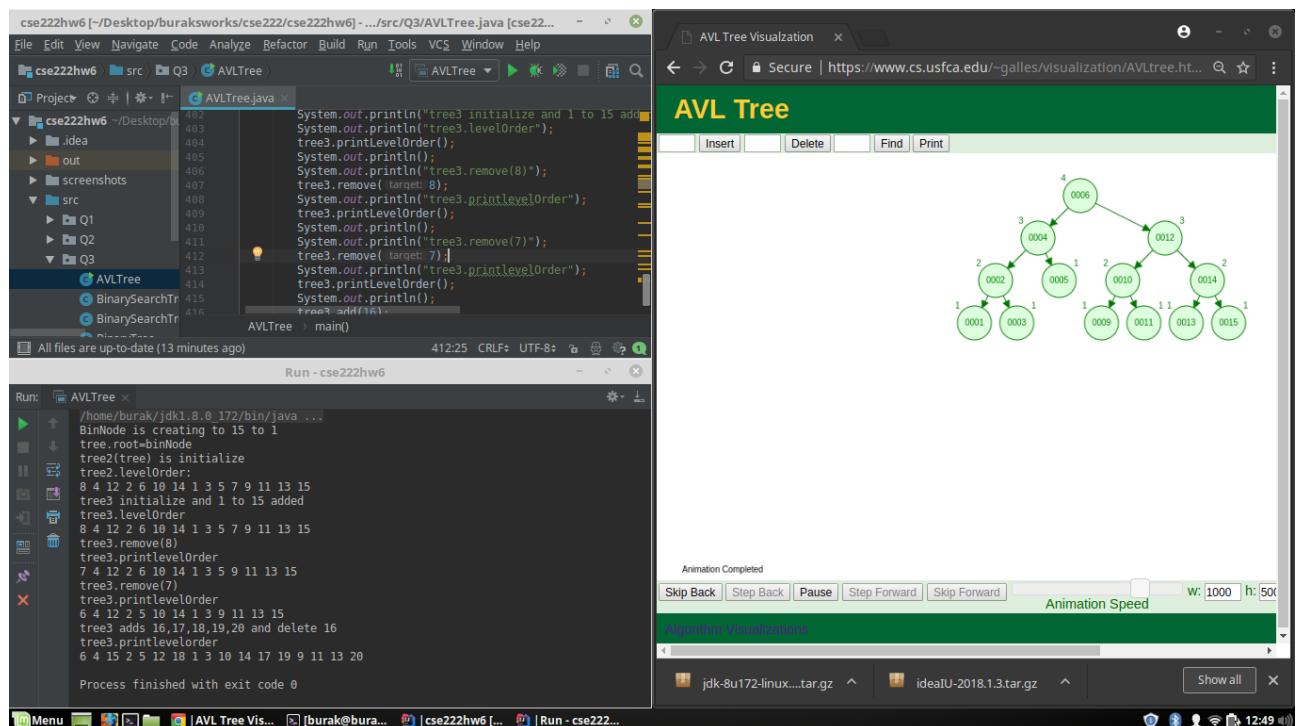
IDE Output:

```

tree.root=binNode;
System.out.println("tree.root=binNode");
AVLTree<Integer> tree2=new AVLTree<>(tree);
System.out.println("tree2(tree) is initialize");
System.out.println("tree2.levelOrder");
tree2.printLevelOrder();
System.out.println();
AVLTree<Integer> tree3=new AVLTree<>();
for (int i = 1; i < 16; i++) {
    tree3.add(i);
}
System.out.println("tree3 initialize and 1 to 15 add");
System.out.println("tree3.levelOrder");
tree3.printLevelOrder();
tree3.remove(8);
tree3.printLevelOrder();
tree3.remove(7);
tree3.printLevelOrder();
tree3.adds(16,17,18,19,20 and delete 16);
tree3.printLevelOrder();
tree3.remove(16);
tree3.printLevelOrder();

```

Q3 Main Test(1 den 15 e kadar ekleme)



The screenshot shows the same IDE and web browser setup as before, but with the AVL tree visualization updated to reflect the insertion of elements 1 through 15.

AVL Tree Visualization Page:

- Buttons: Insert, Delete, Find, Print
- Tree Diagram: A balanced AVL tree with root 0008. The tree structure is as follows:
 - Root: 0008 (BF 4)
 - Left child: 0004 (BF 3)
 - Right child: 0012 (BF 3)
 - Left child of 0004: 0002 (BF 2)
 - Right child of 0004: 0006 (BF 2)
 - Left child of 0012: 0010 (BF 2)
 - Right child of 0012: 0014 (BF 2)
 - Left child of 0002: 0001 (BF 1)
 - Right child of 0002: 0003 (BF 1)
 - Left child of 0006: 0005 (BF 1)
 - Right child of 0006: 0007 (BF 1)
 - Left child of 0010: 0009 (BF 1)
 - Right child of 0010: 0011 (BF 1)
 - Left child of 0014: 0013 (BF 1)
 - Right child of 0014: 0015 (BF 1)
- Animation Controls: Skip Back, Step Back, Pause, Step Forward, Skip Forward, Animation Speed (W: 1000, H: 500)

IDE Output:

```

System.out.println("tree3 initialize and 1 to 15 add");
System.out.println("tree3.levelOrder");
tree3.printLevelOrder();
System.out.println();
System.out.println("tree3.remove(8)");
tree3.remove(8);
System.out.println("tree3.printLevelOrder");
tree3.printLevelOrder();
System.out.println();
System.out.println("tree3.remove(7)");
tree3.remove(7);
System.out.println("tree3.printLevelOrder");
tree3.printLevelOrder();
System.out.println();
System.out.println("tree3.adds(16,17,18,19,20 and delete 16)");
tree3.adds(16,17,18,19,20 and delete 16);
tree3.printLevelOrder();
tree3.remove(16);
tree3.printLevelOrder();

```

Q3 Main Test(7 and 8 deleted)

```

402 System.out.println("tree3 initialize and 1 to 15 added");
403 System.out.println("tree3.levelOrder");
404 tree3.levelOrder();
405 System.out.println();
406 System.out.println("tree3.remove(8)");
407 tree3.remove(target: 8);
408 System.out.println("tree3.levelOrder");
409 tree3.levelOrder();
410 System.out.println();
411 System.out.println("tree3.remove(7)");
412 tree3.remove(target: 7);
413 System.out.println("tree3.levelOrder");
414 tree3.levelOrder();
415 System.out.println();
416 tree3.add(16);
417
AVLTree -> main()

```

Run - cse222hw6

```

/home/burak/jdk1.8.0_172/bin/java ...
BinNode is creating to 15 to 1
tree.root=binNode
tree2(tree) is initialize
tree2.levelOrder:
8 4 12 2 6 10 14 1 3 5 7 9 11 13 15
tree3 initialize and 1 to 15 added
tree3.levelOrder
8 4 12 2 6 10 14 1 3 5 7 9 11 13 15
tree3.remove(8)
tree3.levelOrder
7 4 12 2 6 10 14 1 3 5 9 11 13 15
tree3.remove(7)
tree3.levelOrder
6 4 12 2 5 10 14 1 3 9 11 13 15
tree3 adds 16,17,18,19,20 and delete 16
tree3.levelOrder
6 4 15 2 5 12 18 1 3 10 14 17 19 9 11 13 20
Process finished with exit code 0

```

Q3 Main Test(from 16 to 20 added)

```

414 tree3.levelOrder();
415 System.out.println();
416 tree3.add(16);
417 tree3.add(17);
418 tree3.add(18);
419 tree3.add(19);
420 tree3.add(20);
421 System.out.println("tree3 adds 16,17,18,19,20 and delete 16");
422 tree3.remove(target: 16);
423 System.out.println("tree3.levelOrder");
424 tree3.levelOrder();
425 System.out.println();
426
AVLTree -> main()

```

Run - cse222hw6

```

/home/burak/jdk1.8.0_172/bin/java ...
BinNode is creating to 15 to 1
tree.root=binNode
tree2(tree) is initialize
tree2.levelOrder:
8 4 12 2 6 10 14 1 3 5 7 9 11 13 15
tree3 initialize and 1 to 15 added
tree3.levelOrder
8 4 12 2 6 10 14 1 3 5 7 9 11 13 15
tree3.remove(8)
tree3.levelOrder
7 4 12 2 6 10 14 1 3 5 9 11 13 15
tree3.remove(7)
tree3.levelOrder
6 4 12 2 5 10 14 1 3 9 11 13 15
tree3 adds 16,17,18,19,20 and delete 16
tree3.levelOrder
6 4 15 2 5 12 18 1 3 10 14 17 19 9 11 13 20
Process finished with exit code 0

```

Q3 Main Test(deleted 16)