Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

AKIN ÇAM
151044007

Course Assistant: Fatma Nur Esirci

1 Worst RedBlack Tree

Kırmızı-Siyah ağaçlarla ilgili şu özellikler dikkate alınarak implementasyon yapılmıştır.

- Her node kırmızı ya da siyahtır.
- Root node daima siyahtır.
- Yeni eklenen node ilk eklendiği durumda kırmızıdır.
- Her root-leaf arasındaki yolda eşit sayıda siyah node bulunur.
- Alt alta iki kırmızı node bulunamaz.
- Kırmızı node her zaman siyah node çocuklara sahiptir.
- Null nodelar siyah olarak kabul edilir.

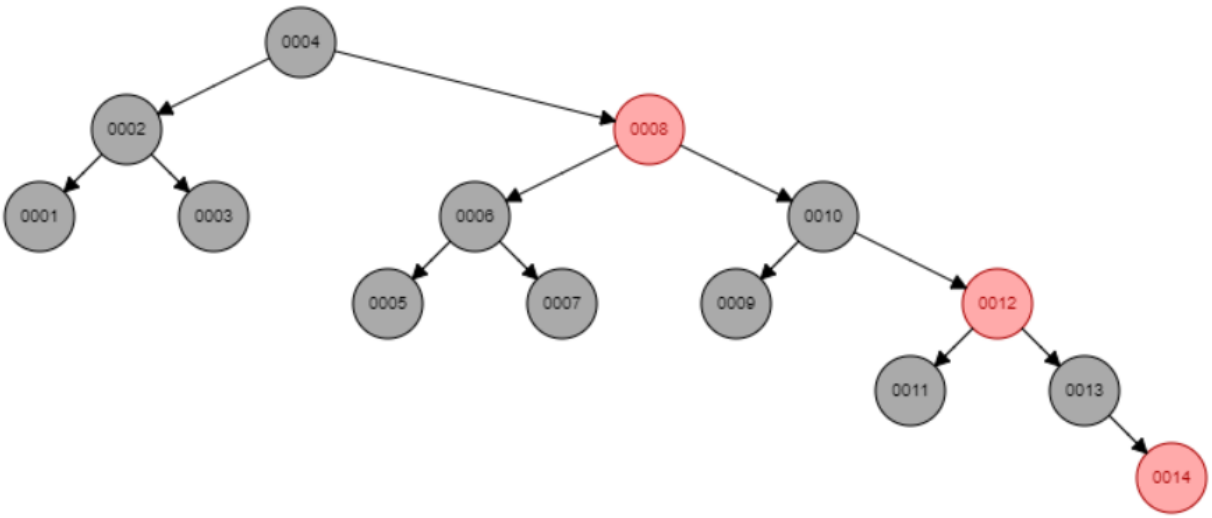
1.1 Problem Solution Approach

Problemi çözümüne şöyle karar verdim:
BinarySearch Tree dengeli olmadığı durumda çalışma zamanı lineerdir. Bu durum elemanlar sıralı eklendiğinde meydana gelir ve yükseklik bir yöne çok fazla ve ağaç dengesiz olur. RedBlackTree bunu önlemen için yapılmıştır ve denge sağlamak amacıyla yer değiştirme-renk ataması gibi özellikler eklenmiştir. Çalışma zamanı logn dir. RedBlack Tree yapısında yükseklik en uzun yoldaki node sayısı değilde o nodların bağlantılarıdır ve BinarySearchTree den 1 eksiktir. Bu şu anlama geliyor: Sıralı elemanlar eklendiğinde node sayısı az olmasına ve normalde dengeli bir ağaç yapısı olmasına rağmen ağaç bir yöne kaymasın diye düzenli yerdeğiştirme ve renk durumlarında değişiklikler olacak ağacın yüksekliği hep normalden daha fazla olacaktır
Bu durum RedBlack ağacı için en kötü durumu oluşturur. RedBlackTree yapısı bunu önlemek için yapılmış fakat node sayısı az olmasına rağmen yükseklik en üst seviyede olduğu için eleman ekleme arama gibi metotlar çalışma zamanı en kötü halini alacaktır. En az sayıda yani 22 node ile yüksekliği 6 olan en kötü RedBlack Tree oluşturulmuştur.

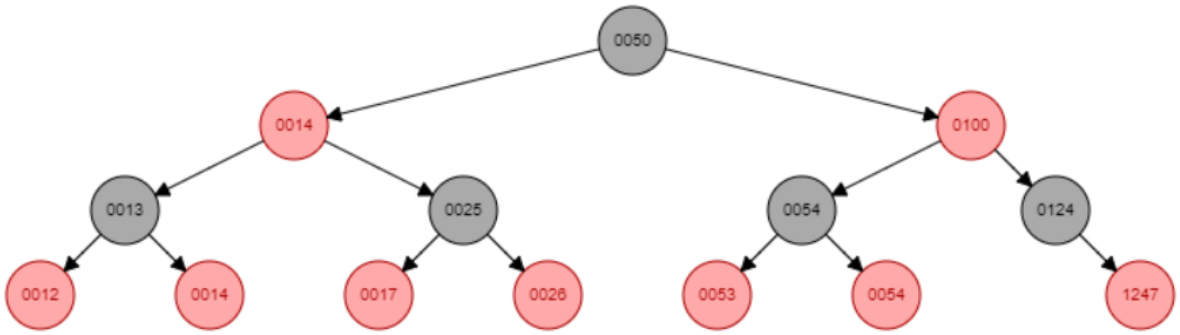
1.2 Test Cases

Elemanlar sıralı ekleneceği için ilk eleman rastgele alınır ve diğer elemanlar bundan büyük olacak şekilde eklenmektedir. En kötü durum olduğunu göstermek için rastgele 14 eleman eklenmiş ve ağacın yüksekliğine bakabilirsiniz. Diğer durumda 14 node sıralı biçimde eklenir ve sonuçlar aşağıda gösterilmiştir. (22 node eklendiğinde tüm elemanlar ekrana sığmıyor):

Sıralı Eklenen Elemanlar (yükseklik=5 node 14):



Sırasız Eklenen Elemanlar(Rastgele) (yükseklik=3 node 14):



Yüksekliği bulan bir metot yazılmıştır:
public int showHeight(){

eğer ağaç boş ise
0 döndür.
eğer ağaç boş değilse
sağ ağaç ile devam et yüksekliği artır.
sağ ağaç ile devam et yüksekliği artır.
yükseklikleri karşılaştır hangisi büyük ise onu döndür
}

1.3 Running Commands and Results

Sıralı Eklenen Elemanlar:

```
"C:\Program Files\Java\jdk1.8.0_161\bin\java
Black: 16427
  null
  null

Black: 16427
  null
  Red  : 16428
    null
    null

Black: 16428
  Red  : 16427
    null
    null
  Red  : 16429
    null
    null

Black: 16428
  Black: 16427
    null
    null
  Black: 16429
    null
    Red  : 16430
      null
      null

Black: 16428
  Black: 16427
    null
    null
  Black: 16430
    Red  : 16429
      null
      null
    Red  : 16431
      null
      null
```

```
Black: 16428
  Black: 16427
    null
    null
  Red  : 16430
    Black: 16429
      null
      null
    Black: 16431
      null
      Red  : 16432
        null
        null

Black: 16428
  Black: 16427
    null
    null
  Red  : 16430
    Black: 16429
      null
      null
    Black: 16432
      Red  : 16431
        null
        null
      Red  : 16433
        null
        null
```

```
Black: 16430
  Red  : 16428
    Black: 16427
      null
      null
    Black: 16429
      null
      null
  Red  : 16432
    Black: 16431
      null
      null
    Black: 16433
      null
      Red  : 16434
        null
        null

Black: 16430
  Black: 16428
    Black: 16427
      null
      null
    Black: 16429
      null
      null
  Black: 16432
    Black: 16431
      null
      null
    Black: 16434
      Red  : 16433
        null
        null
      Red  : 16435
        null
        null
```

```
Black: 16430
  Black: 16428
    Black: 16427
      null
      null
    Black: 16429
      null
      null
  Black: 16432
    Black: 16431
      null
      null
    Red  : 16434
      Black: 16433
        null
        null
      Black: 16435
        null
        Red  : 16436
          null
          null

Black: 16430
  Black: 16428
    Black: 16427
      null
      null
    Black: 16429
      null
      null
  Black: 16432
    Black: 16431
      null
      null
    Red  : 16434
      Black: 16433
        null
        null
      Black: 16436
        Red  : 16435
          null
          null
        Red  : 16437
          null
          null
```

```
Black: 16430
  Black: 16428
    Black: 16427
      null
      null
    Black: 16429
      null
      null
  Black: 16434
    Red  : 16432
      Black: 16431
        null
        null
      Black: 16433
        null
        null
    Red  : 16436
      Black: 16435
        null
        null
      Black: 16437
        null
        Red  : 16438
          null
          null
```

```
Black: 16430
  Black: 16428
    Black: 16427
      null
      null
    Black: 16429
      null
      null
  Red  : 16434
    Black: 16432
      Black: 16431
        null
        null
      Black: 16433
        null
        null
    Black: 16436
      Black: 16435
        null
        null
      Black: 16438
        Red  : 16437
          null
          null
        Red  : 16439
          null
          null
```

```
Black: 16430
  Black: 16428
    Black: 16427
      null
      null
    Black: 16429
      null
      null
  Red  : 16434
    Black: 16432
      Black: 16431
        null
        null
      Black: 16433
        null
        null
    Black: 16436
      Black: 16435
        null
        null
      Red  : 16438
        Black: 16437
          null
          null
        Black: 16439
          null
          Red  : 16440
            null
            null
```

```
Black: 16430
  Black: 16428
    Black: 16427
      null
      null
    Black: 16429
      null
      null
  Red  : 16434
    Black: 16432
      Black: 16431
        null
        null
      Black: 16433
        null
        null
    Black: 16436
      Black: 16435
        null
        null
      Red  : 16438
        Black: 16437
          null
          null
        Black: 16440
          Red  : 16439
            null
            null
          Red  : 16441
            null
            null
```

```
Black: 16430
  Black: 16428
    Black: 16427
      null
      null
    Black: 16429
      null
      null
  Red  : 16434
    Black: 16432
      Black: 16431
        null
        null
      Black: 16433
        null
        null
    Black: 16438
      Red  : 16436
        Black: 16435
          null
          null
        Black: 16437
          null
          null
      Red  : 16440
        Black: 16439
          null
          null
        Black: 16441
          null
          Red  : 16442
            null
            null
```

```
Black: 16434
  Red  : 16430
    Black: 16428
      Black: 16427
        null
        null
      Black: 16429
        null
        null
    Black: 16432
      Black: 16431
        null
        null
      Black: 16433
        null
        null
  Red  : 16438
    Black: 16436
      Black: 16435
        null
        null
      Black: 16437
        null
        null
    Black: 16440
      Black: 16439
        null
        null
      Black: 16442
        Red  : 16441
          null
          null
        Red  : 16443
          null
          null
```

```
Black: 16434
  Black: 16430
    Black: 16428
      Black: 16427
        null
        null
      Black: 16429
        null
        null
    Black: 16432
      Black: 16431
        null
        null
      Black: 16433
        null
        null
  Black: 16438
    Black: 16436
      Black: 16435
        null
        null
      Black: 16437
        null
        null
    Black: 16440
      Black: 16439
        null
        null
      Red  : 16442
        Black: 16441
          null
          null
        Black: 16443
          null
          Red  : 16444
            null
            null
```

```
Black: 16434
  Black: 16430
    Black: 16428
      Black: 16427
        null
        null
      Black: 16429
        null
        null
    Black: 16432
      Black: 16431
        null
        null
      Black: 16433
        null
        null
  Black: 16438
    Black: 16436
      Black: 16435
        null
        null
      Black: 16437
        null
        null
    Black: 16440
      Black: 16439
        null
        null
      Red  : 16442
        Black: 16441
          null
          null
        Black: 16444
          Red  : 16443
            null
            null
          Red  : 16445
            null
            null
```

```
Black: 16434
  Black: 16430
    Black: 16428
      Black: 16427
        null
        null
      Black: 16429
        null
        null
    Black: 16432
      Black: 16431
        null
        null
      Black: 16433
        null
        null
  Black: 16438
    Black: 16436
      Black: 16435
        null
        null
      Black: 16437
        null
        null
    Black: 16442
      Red  : 16440
        Black: 16439
          null
          null
        Black: 16441
          null
          null
      Red  : 16444
        Black: 16443
          null
          null
        Black: 16445
          null
          Red  : 16446
            null
            null
```

```
Black: 16430
  Black: 16428
    Black: 16427
      null
      null
    Black: 16429
      null
      null
  Black: 16432
    Black: 16431
      null
      null
    Black: 16433
      null
      null
Black: 16438
  Black: 16436
    Black: 16435
      null
      null
    Black: 16437
      null
      null
  Red  : 16442
    Black: 16440
      Black: 16439
        null
        null
      Black: 16441
        null
        null
    Black: 16444
      Black: 16443
        null
        null
      Black: 16446
        Red  : 16445
          null
          null
        Red  : 16447
          null
          null
```

```
Black: 16434
  Black: 16430
    Black: 16428
      Black: 16427
        null
        null
      Black: 16429
        null
        null
    Black: 16432
      Black: 16431
        null
        null
      Black: 16433
        null
        null
  Black: 16438
    Black: 16436
      Black: 16435
        null
        null
      Black: 16437
        null
        null
    Red  : 16442
      Black: 16440
        Black: 16439
          null
          null
        Black: 16441
          null
          null
      Black: 16444
        Black: 16443
          null
          null
        Red  : 16446
          Black: 16445
            null
            null
          Black: 16447
            null
            Red  : 16448
```

2 binarySearch method

Bu bölümde BTree Insert metodu için eksik olan binarySearch metodu yazılmıştır.

2.1 Problem Solution Approach

Private binarySearch metodu yazılmıştır. Bu metot yazılırken şunlar dikkate alınmıştır:
Burada elemanın yerleştirilebileceği index dikkate alınarak yazılmıştır.(elemanın hangi aralıkta bulunduğuna bakarak) BinarySearch te eleman sayısı/2 kadar arama yapılmaktadır.

```
private int binarySearch(E item, E[] data, int i, int size){
        count=0
        (while) count elemansayısı/2+1 e eşit olmadığı sürece devam et
                Ortadaki elemanın indexini bul.
                Ortadaki eleman ile aranan elemanı karşılaştır.
                Eğer aranan eleman ortadaki elemandan küçük ise
                        Ağacın solu dikkate alınacak şekilde aralıkları değiştir.
                Eğer aranan eleman ortadaki elemandan büyük ise
                        Ağacın sağı dikkate alınacak şekilde aralıkları değiştir.
                Eğer aranan eleman ortadaki elemana eşit ise
                        İndex i ortadaki eleman indexini atama yap döngüden çık
         Return index;
}
```

N tane eleman varsa çalışma zamanı için her seferinde yarısı olduğu için O(logn) diyebiliriz.

```
public E binarySearch(E item ){
        return binarySearch(item,root);
}
Private binarySearch(E item,Node<E> localRoot){
        Eğer Verilen node null ise
                null return eder
        üstteki binarySearch ile indexi bulur

        eğer verilen indexte aranan eleman varsa
                aranan elemanı return eder
        verilen indexte yoksa indexin child ı var mı ona bakılır
                yoksa null return eder
        verilen indexte yoksa indexin child ı var mı ona bakılır
                varsa o alt ağaç ile devam eder
}
```

Üstteki metodun çalışma zamanı logn dir. Bu method da üsteki metot dışında hep alt ağaç şeklinde devam eder bölerek logn dir. Çalışma zamanı O(logn^2) dir.

2.2 Test Cases

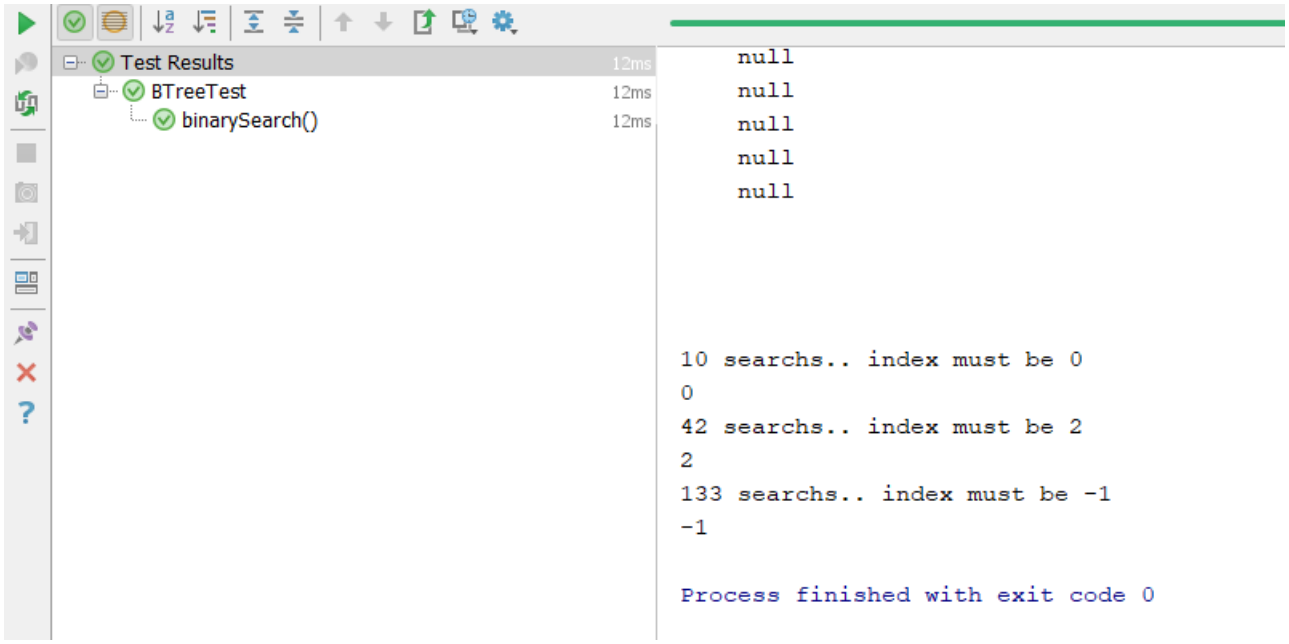
Public binarysearch method Test1→

```
System.out.println("10 searchs.. index must be 0");
bTree.binarySearch( item: 10);
assertEquals( expected: 0,bTree.getIndex());
System.out.println("13 searchs.. index must be 1");
bTree.binarySearch( item: 13);
assertEquals( expected: 1,bTree.getIndex());

System.out.println("133 searchs.. index must be -1");
bTree.binarySearch( item: 133);
assertEquals( expected: -1,bTree.getIndex());
```

```
"C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
22, 40
  10, 13
    null
    null
    null


  33
    null
    null


  41
    null
    null


10 searchs.. index must be 0
0
13 searchs.. index must be 1
1
133 searchs.. index must be -1
-1

Process finished with exit code 0
```

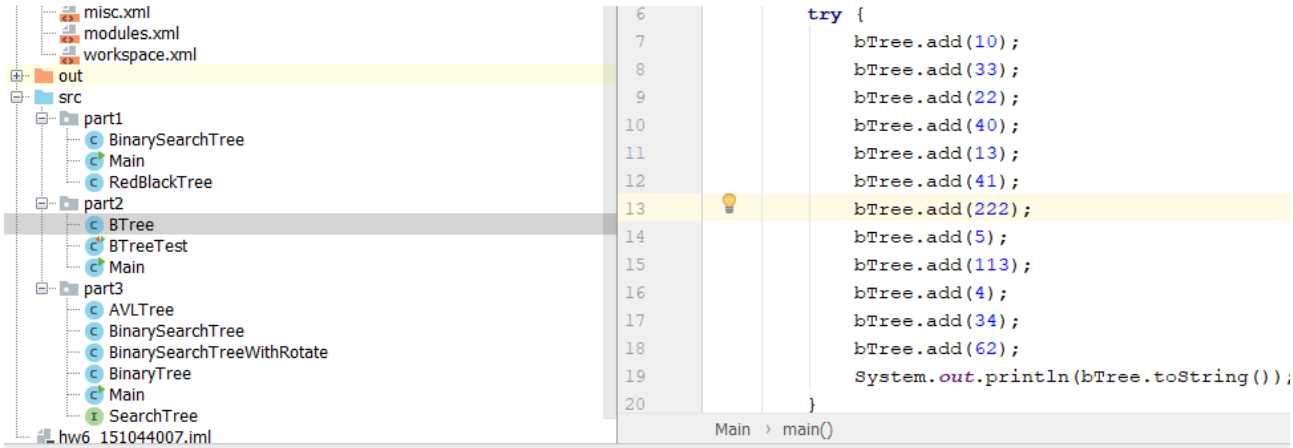
Public binarysearch method Test2→

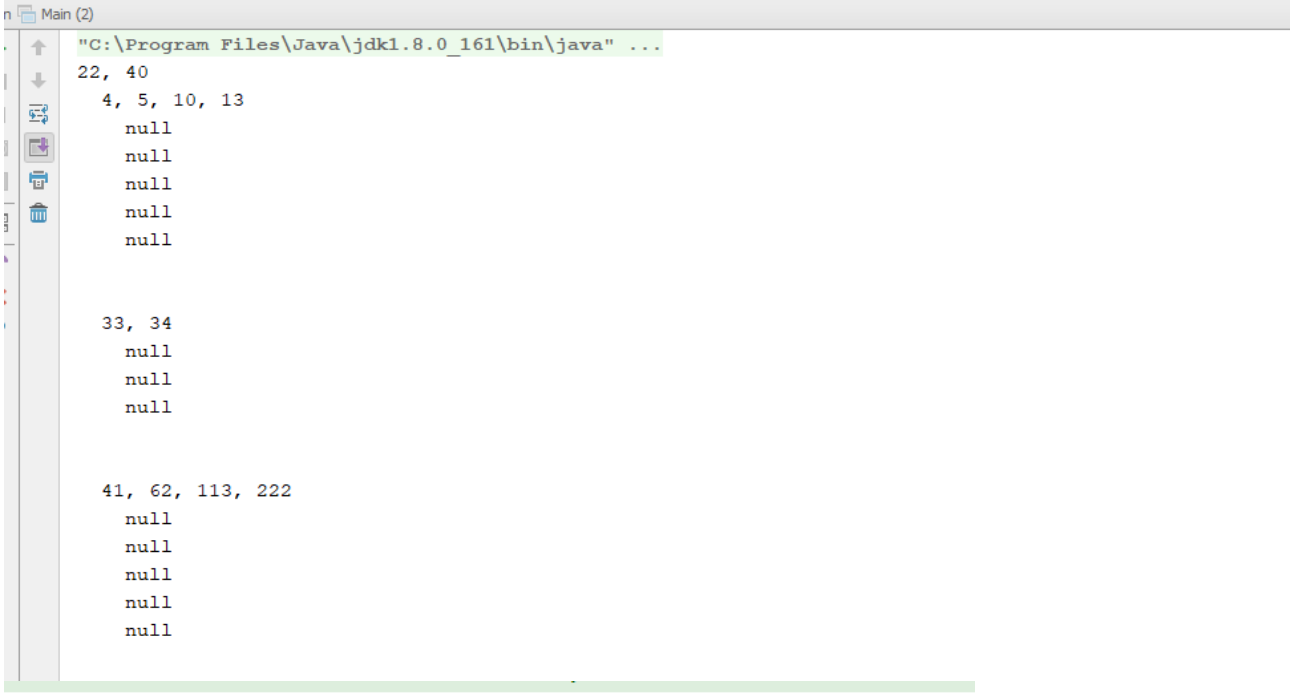
```
BTree<Integer> bTree=new BTree<>( order: 5);
bTree.add(10);
bTree.add(33);
bTree.add(22);
bTree.add(40);
bTree.add(41);
bTree.add(42);
bTree.add(43);
System.out.println(bTree.toString());
System.out.println("10 searchs.. index must be 0");
bTree.binarySearch( item: 10);
assertEquals( expected: 0,bTree.getIndex());
System.out.println("42 searchs.. index must be 2");
bTree.binarySearch( item: 42);
assertEquals( expected: 2,bTree.getIndex());

System.out.println("133 searchs.. index must be -1");
bTree.binarySearch( item: 133);
assertEquals( expected: -1,bTree.getIndex());
```

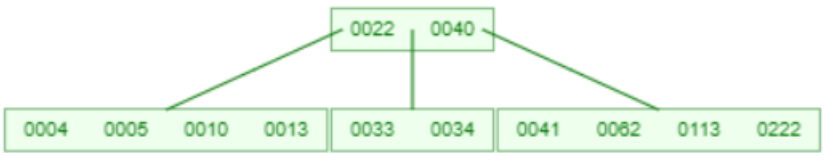


2.3 Running Commands and Results

Test1→


misc.xml
modules.xml
workspace.xml
out
src
part1
BinarySearchTree
Main
RedBlackTree
part2
BTree
BTreeTest
Main
part3
AVLTree
BinarySearchTree
BinarySearchTreeWithRotate
BinaryTree
Main
SearchTree
hw6_151044007.iml
try {
bTree.add(10);
bTree.add(33);
bTree.add(22);
bTree.add(40);
bTree.add(13);
bTree.add(41);
bTree.add(222);
bTree.add(5);
bTree.add(113);
bTree.add(4);
bTree.add(34);
bTree.add(62);
System.out.println(bTree.toString());
}
Main › main()


Main (2)
"C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
22, 40
4, 5, 10, 13
null
null
null
null
null
33, 34
null
null
null
41, 62, 113, 222
null
null
null
null
null


0022
0040
0004
0005
0010
0013
0033
0034
0041
0062
0113
0222

Test2→

```
i, s
  a, d, f
    null
    null
    null
    null

  k, l
    null
    null
    null

  t, v, w
    null
    null
    null
    null
```

3 Project 9.5 in book

Bu bölümde AVLTree delete metodu yazılmıştır. AVLTree dengeli bir ağaç yapısı oluşturur. Ekleme ve silme durumunda nodeların lokasyonları değişir ve dengeli hale getirilir.

3.1 Problem Solution Approach

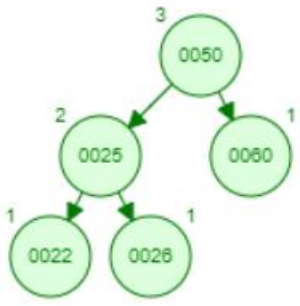
Consturctor metodunu tamamlayamadım.
Delete metodu yazılmıştır. Şu bilgiler dikkate alınmıştır:
rebalanceLeft rebalanceRight metotlarını delete için yeniden yazamadım. Tek delete metodunda her şeyi yapmaya çalıştım. Bu yüzden metot çok verimli olmadı.
Metot yinelemeli bir metotdur Add metodu ile ters işlemler yapılır.(add de denge sola ekleyince sola kayarsa delete tam tersi sağa kayar).
Yardımcı bir private metot kullanılmıştır.

```
private AVLNode<E> delete(AVLNode<E> node, E item){

        eğer node null ise
                silinen elemana null ataması yap bulunamadığı ve silinmediği için
                null return et

        eğer item karşılaştırılan node.data dan küçük ise
```

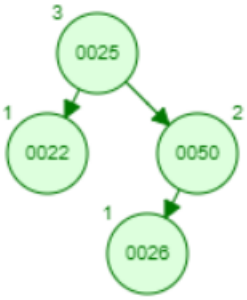
```
                sol alt ağaç için delete metodunu çağır(recursive)

            eğer control==1 ise yani eleman silinmiş silinen elemanın yerine
            yeni eleman gelmiş yani Remove yapılmış  ve ağaç dengesi yeniden dü-
            zenlenmesi gerekiyorsa(döndürme yapılarak denge sağlanır)
                sol alt ağaçta silme işlemi meydana geldiği için node un denge
            durumu artar. (Sol taraf eksi sağ taraf artı)(recursive olduğu için
            geri eski haline gelirken balance artar)
                ağaç dengelenir.

eğer item karşılaştırılan node.data dan büyük ise
                sağ alt ağaç için delete metodunu çağır(recursive)

        sol taraf için yapılanların tam tersi yapılır.


Eğer aranan eleman bulunduysa

        Return etmek için kaydedilir.
        Eğer solu yada sağı null ise node null olur
        Sağda en büyük eleman olduğu için eğer bir alt node un sağı varsa ola-
bildiğince en sağa gidip(while döngüsü) o node u silinen node un yerine koyar
aksi takdirde bir solundaki elemanı kendine atar ve kendini siler. Bu durumda
Solda yük dengesizliği olacağı için solu yeniden dengeler. Ama olabildiğince
sağa gittiği durumda denge işlemini uygulamaz çünkü sola doğru denge bozulmaz.
}

Çalışma zamanı için konuşacak olursak AVL tree nin Remove durumu kadar verimli
değildir. Hatta verimsiz bir metottur. Çalışma zamanı için şöyle söylenebilir.
Logn de elemanı bulur. Daha sonra elemanı bulduğunda bir while döngüsü ve en az
2 tane dengeyi sağlamak için işlem yapılır. Logn*n*n yani O(n^2*logn) diyebili-
riz.
```

3.2 Test Cases

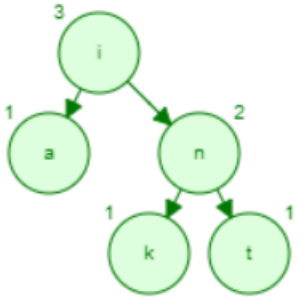
Test Edilen durumlar:
Eleman eklendikten sonra liste→



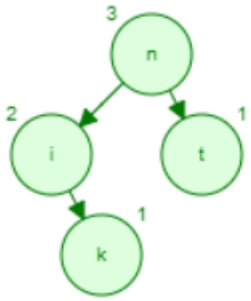
60 silindikten sonra liste→



Test2→

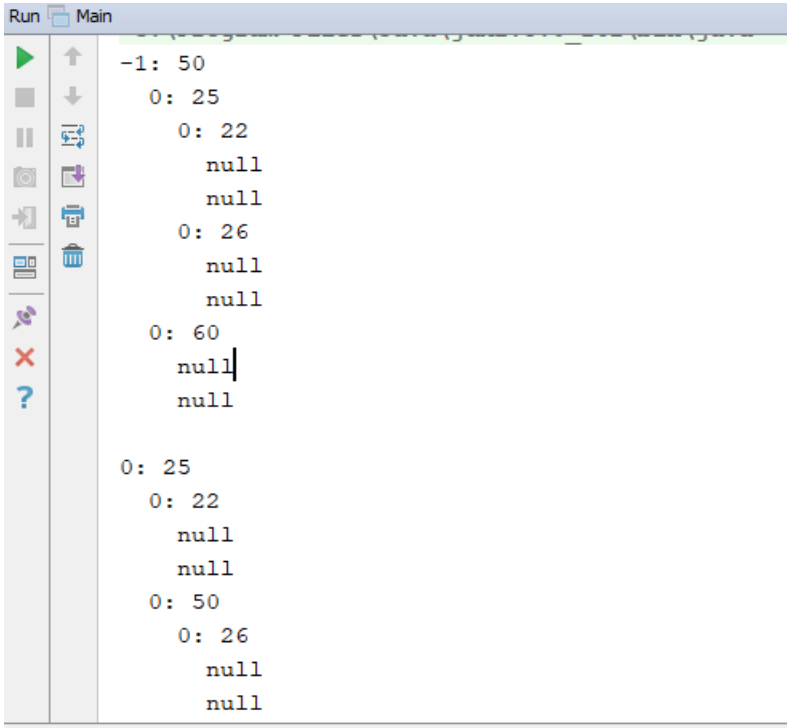


Remove ‘a’

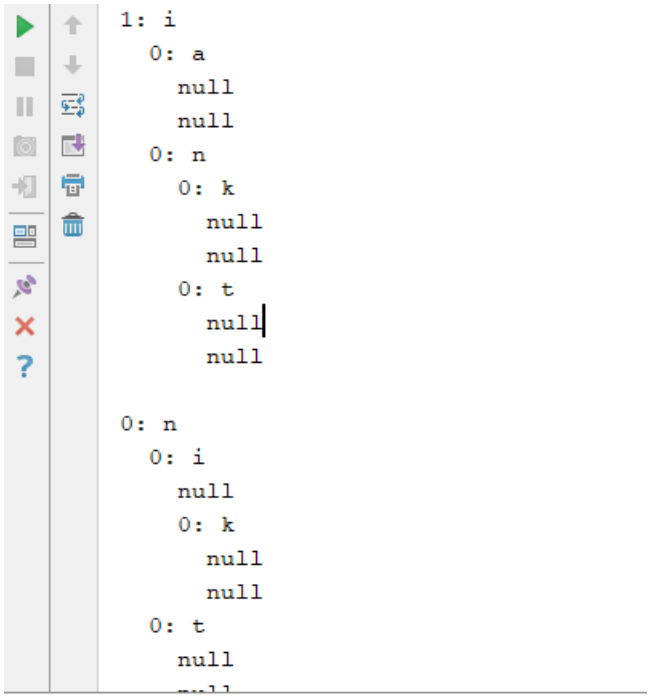


3.3 Running Commands and Results

TestCase 1→
50-60-25-22-26
TestCase2(Remove 60)→


Run
Main
-1: 50
0: 25
0: 22
null
null
0: 26
null
null
0: 60
null
null
0: 25
0: 22
null
null
0: 50
0: 26
null
null

Test2→


1: i
0: a
null
null
0: n
0: k
null
null
0: t
null
null
0: n
0: i
null
0: k
null
null
0: t
null