Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

EMİRE KORKMAZ
141044043

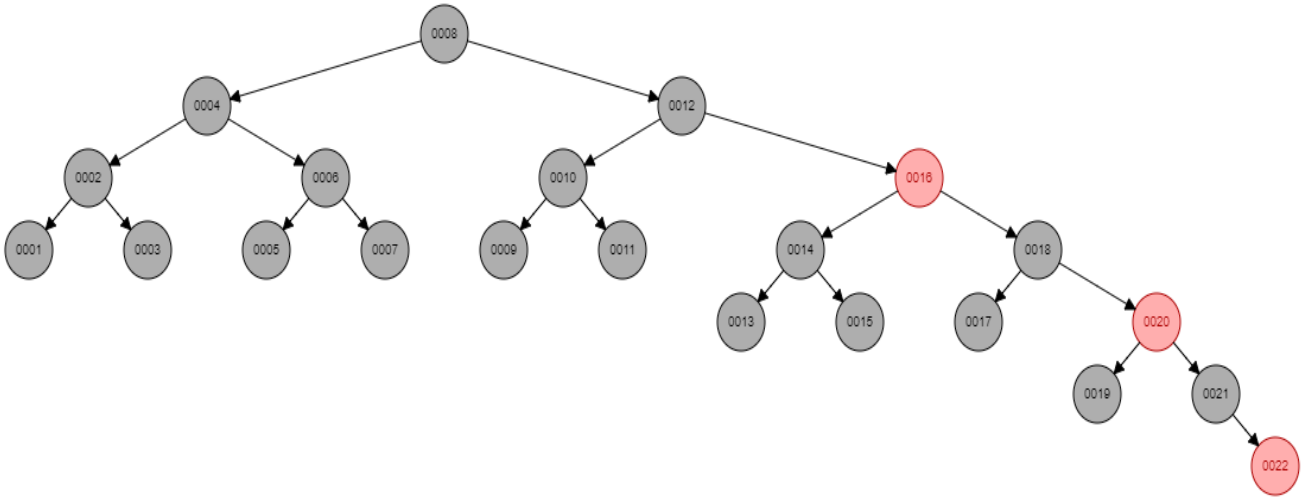
Course Assistant: Fatma Nur Esirci

1 Worst RedBlack Tree

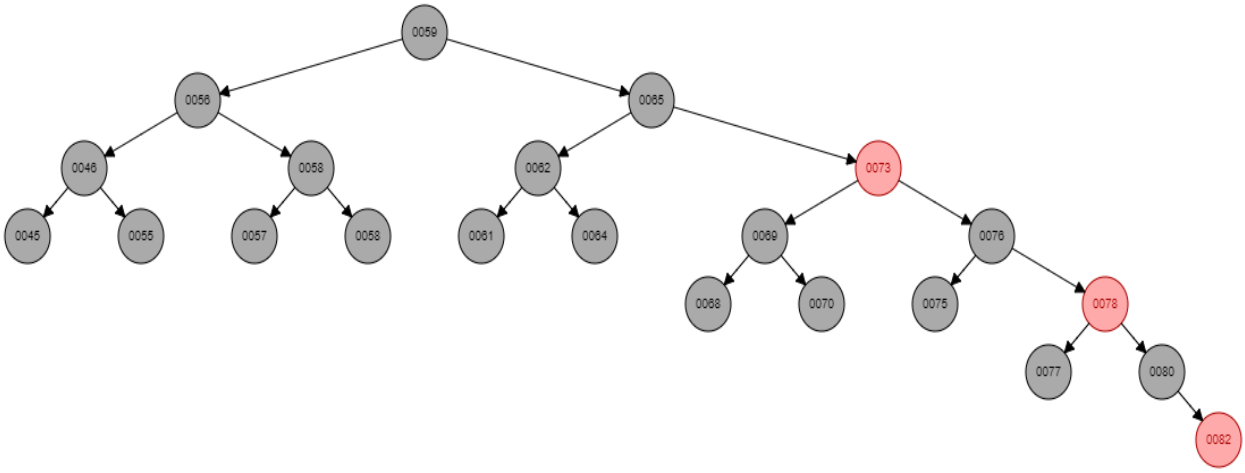
1.1 Problem Solution Approach

Red-Black Tree yapısının kuralları kullanılarak Red-Black Tree yapısında herhangi bir bozulma olmasına sebebiyet vermeden node'lar eklenilmiştir. 6 yüksekliğine ulaşıldığında bu yüksekliğe toplamda 22 node kullanılarak ulaşılmıştır.
-Verilen kodda eksik olan ve ekleme yapılırken yeni gelen çocuğun rengi kırmızı olacağı için önceki çocukları kırmızılarsa Red-Black Tree yapısına uymayacağı için siyaha çeviren moveBlackDown metodu yazılmıştır.
-**add** metodundaki eksik olan **item > localRoot.data** durumu için kullanılacak olan else bloğu da önceki if ve else if bloklarındaki yapıya uygun şekilde tamamlanmıştır.

1.2 Test Cases



Bizden yüksekliği 6 olan bir Red-Black Tree oluşturmamız isteniyor. Yüksekliği 6 olan bir Red-Black Tree yapısı 22 node kullanarak bu şekilde oluşturuldu.



Yüksekliği 6 olan bir başka Red-Black Tree yapısı yine 22 node kullanılarak oluşturuldu.

1.3 Running Commands and Results

Kendim oluştuduğum Red-Black Tree'yi aynı değerleri verek kodu kullanarak da yaptım.

```
Black: 8
  Black: 4
    Black: 2
      Black: 1
        null
        null
      Black: 3
        null
        null
    Black: 6
      Black: 5
        null
        null
      Black: 7
        null
        null
  Black: 12
    Black: 10
      Black: 9
        null
        null
      Black: 11
        null
        null
    Red  : 16
      Black: 14
        Black: 13
          null
          null
        Black: 15
          null
          null
      Black: 18
        Black: 17
          null
          null


      Red  : 20
        Black: 19
          null
          null
        Black: 21
          null
          Red  : 22
            null
            null
```

15 node kullanarak oluşturulan Red-Black Tree

```
Black: 27
  Black: 7
    Black: 5
      Red  : 2
        null
        null
      null
    Red  : 10
      Black: 8
        null
        null
      Black: 22
        null
        Red  : 23
          null
          null
  Black: 30
    Black: 28
      null
      null
    Red  : 42
      Black: 33
        null
        null
      Black: 48
        Red  : 43
          null
          null
        null
```

2 binarySearch method

2.1 Problem Solution Approach

Bu bölümde bizden kitaptaki B-Tree kodundaki eksik olan binarySearch metodunu yazmamız isteniyor. B-Tree yapısında binary search yapabilmek için her bir Node'da arama yapıldı. B-Tree yapısınının node'ları kullanılarak sırasıyla node'larda Binary Search yapılıp en yakın değer bulunduktan sonra çocuk node'a geçilerek aynı şekilde burada da Binary Search yapılarak aranan değer bulunana kadar bu işlem devam ettirilmiştir. Öncelikle ortadaki index'i bulup eşleşme varsa eşleşen index return edildi. Eğer aranılan değer, ortadaki index'teki değerden küçükse ve ortadaki aynı metod farklı parametrelerle yani ilk index'i 0 ve son index'i de ortadaki değer olacak şekilde bir daha çağırıldı. Eğer aranılan değer, ortadaki index'teki değerden büyükse aynı metod farklı parametrelerle yani ilk index'i ortadaki değer ve son index'i de uzunluk olacak şekilde bir daha çağırıldı. Aranan değerin node'daki iki değer arasında olduğu görülürse node'un çocuğu parametre olarak verilerek fonksiyon bir daha çağrılır aynı işlemler aranan değer bulunana kadar ya da olmadığı anlaşılana kadar sürer. İlk Node'da arama yapılırken büyük olan değere gelindiğinde diğer Node'a geçilme sebebi aranılan değerin Node'un çocuklarından birinde olduğunun belli olmasıdır. Farklı parametrelerle çağırınca aynı işlemler çocuk Node'da da gerçekleşecektir. Ve en sonunda aranılan değerin B-Tree'de olup olmadığı return edilen değer göre anlaşılacaktır. Bulunamaması durumuda -1 return ediliyor.

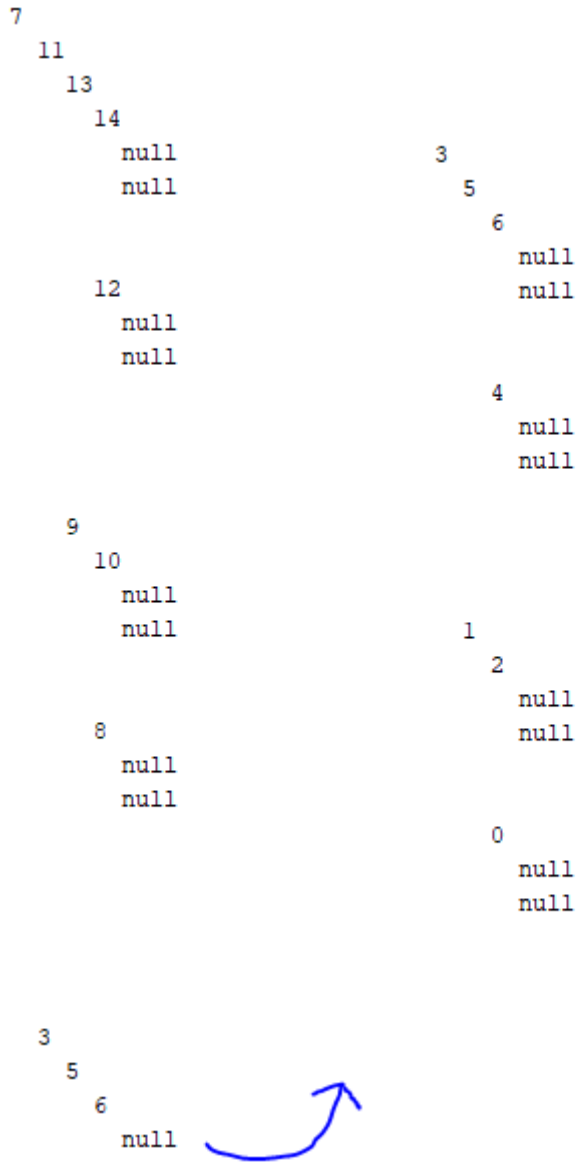
```
public int binarySearch(E item, E[] data, int first, int size){
  return helper(item, root, first, size);
}

private int helper(E item, Node<E> node, int first, int last)
```

2.2 Test Cases

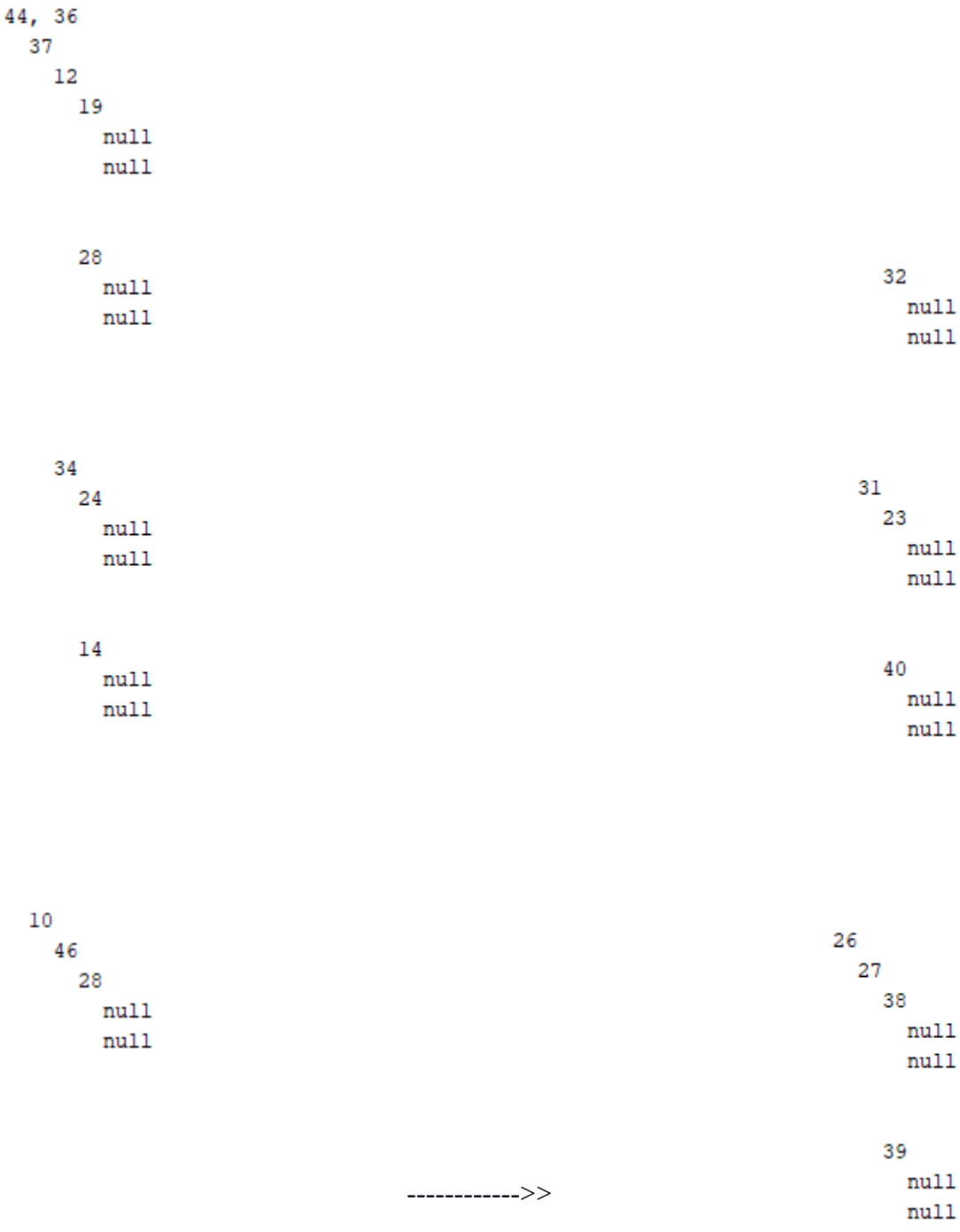
Try this code to search least 4 element on 2 different BTree. Report all of situations.

2.3 Running Commands and Results

İlk olarak 1 ile 15 arasındaki sayılar yerleştirildi.



İkinci olarak 25 tane rastgele sayı yerleştirildi.



```
5
  22
    null
    null

  33
    null
    null
```

Project 9.5 in book

2.4 Problem Solution Approach

Bu bölümde bizden AVL Tree class'ı için bir constructor, delete, decrementBalance, incrementBalance, rebalanceleft, rebalanceRight metodlarını yazmamız isteniyor.
-Constructor için AVL Tree olup olmadığını kontrol etmek için 2 tane yardımcı metod yazıldı. Bu metodlar **checkBalance** ve **findHeight** metodlarıdır. AVL Tree olma şartı olan dengeli olma şartı bu metodlar yardımıyla kontrol edilmiştir.
-RebalanceRight metodu yazılırken **rebalanceLeft** metodundan yardım alınıp tersi durum için gereken şeyler yazılmıştır.
-incrementBalance metodunda verilen node'ın balance değişkeni artırılıp ağacın dengesinin bozulup bozulmadığı kotrol edilmiştir. Eğer bozulmadıysa ağacın yüksekliği değişmemiştir yani increase değişkeni true olarak de decrease değişkeni de false olarak atanmıştır, diğer durumda yani ağacın dengesi bozulduğunda yükseklik artmıştır azalmamıştır. Buna göre increase ve decrease değişkenleri sırayla true ve false olarak atanır.
-add metodundaki en sondaki else bloğunu yani **item<localroot.data** durumunu yazmamız isteniyor. Bu blok yazılırken bir önceki else if bloğundan yani item>localroot.data durumundan yardım alınarak yazıldı.
-delete metodu Binary Search Tree'nin delete metofuna çok benzediği için Binary Search Tree metodunu kullanılarak yazıldı. Bir yardımcı metod kullanılarak ve bu metoda parametre olarak silinecek elemanın yanı sıra root da verilerek root'taki değerle silinecek değer karşılaştırılarak bir recursive metod yazıldı.

```
public Node<E> rightLeftRotate(Node<E> node)
```

Sağ ve sol döndürme gerek durumlar için yazılmıştır. Önce sağ tarafın çocuğu döndürülür daha sonra sol taraf döndürülür.

```
public Node<E> leftRightRotate(Node<E> node)
```

Sol ve sağ döndürme gerek durumlar için yazılmıştır. Önce sol tarafın çocuğu döndürülür daha sonra sağ taraf döndürülür.

```
public boolean checkBalance(BinaryTree.Node<E> node)
```

Verilen node'un sağ ve sol çocuklarının yüksekliklerine bakara dengeli olup olmadığını kontrol eder.

```
private int findHeight(BinaryTree.Node<E> node)
```

checkBalance metodununda verilen node'un yüksekliğini bulmak için yazıldı. Parametre olarak gelen node'un ağacın sonuna kadar yani null gelene kadar çocuklarını gezer ve yüksekliği döndürür.

2.5 Test Cases

Show that all methods your implemented run correctly. Report all of situations.

2.6 Running Commands and Results

```
Add method
1: 3
  0: 1
    0: 0
      null
      null
    0: 2
      null
      null
  0: 7
  Delete method
  0: 1
    0: 0
      null
      null
    0: 3
      0: 2
        null
        null
      0: 7
        0: 5
          0: 4
            null
            null
          0: 6
            null
            null
        1: 8
          null
          0: 9
            null
            null
```

Aynı zamanda BinaryTree olan
BinarySearchTree objesi oluşturup constructor’a parametre olarak verdim.

```
Constructor
Not an AVL Tree
```