Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

HALİL ONUR ÇEÇEN
161044057

Course Assistant: Fatma Nur Esirci

1 Worst RedBlack Tree

In this part i’ve used source code of the book for RedBlackTree to create and plot worst RedBlack Tree in height of 6.

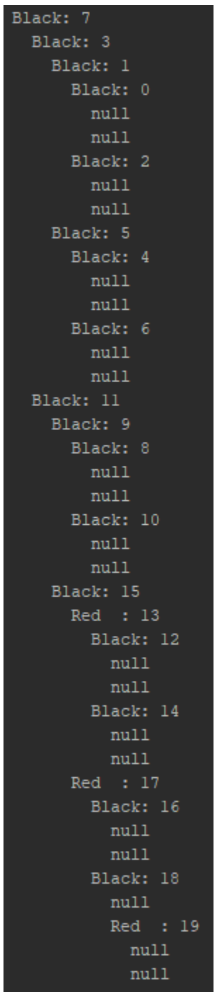
1.1 Problem Solution Approach

To create a worst RedBlack Tree i think it requires to have most black nodes in least amount of total nodes. To replicate that i’ve inserted 20 elements in order. That will result in a way that its left childs are always black consecutively and linear.

1.2 Test Cases

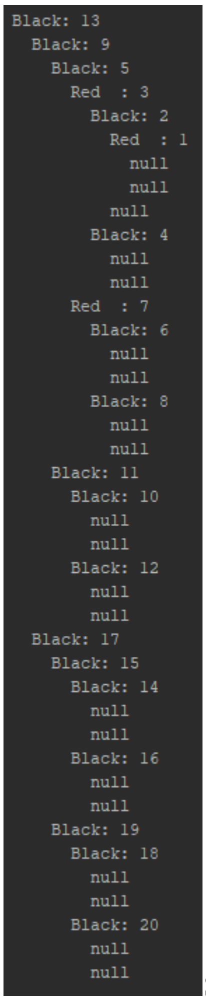
First case is in order of numbers from 0 to 19.
Second case in reverse order of numbers from 0 to 19.

1.3 Running Commands and Results



```
Black: 7
  Black: 3
    Black: 1
      Black: 0
        null
        null
      Black: 2
        null
        null
    Black: 5
      Black: 4
        null
        null
      Black: 6
        null
        null
  Black: 11
    Black: 9
      Black: 8
        null
        null
      Black: 10
        null
        null
    Black: 15
      Red  : 13
        Black: 12
          null
          null
        Black: 14
          null
          null
      Red  : 17
        Black: 16
          null
          null
        Black: 18
          null
          Red  : 19
            null
            null
```

First Test Case.



```
Black: 13
  Black: 9
    Black: 5
      Red  : 3
        Black: 2
          Red  : 1
            null
            null
          null
        Black: 4
          null
          null
      Red  : 7
        Black: 6
          null
          null
        Black: 8
          null
          null
    Black: 11
      Black: 10
        null
        null
      Black: 12
        null
        null
  Black: 17
    Black: 15
      Black: 14
        null
        null
      Black: 16
        null
        null
    Black: 19
      Black: 18
        null
        null
      Black: 20
        null
        null
```

Second Test Case.

2 binarySearch method

In this part we're asked to implement missing binarySearch method of BTree class from course source code.

2.1 Problem Solution Approach

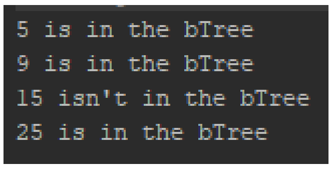
```
binarySearch(E item, E[]  data, int I, int size){
        int mid = (size – i) / 2 + I;
        if ( index  == data.length  ||
          (data[index] != null && data[index] == item)  ||
         Size == i )
                return index;
        if (data[index] .> item) return binarySearch(item,data,i,index-1);
        else return binarySearch(item,data,index+1,size);
```

In this method I've compared the item with middle element of data array and called the same method with middle to last index or first to middle index according to comparison. If compared element is the item or exceeded array limit or last and first size is same returned the current index. Thus, rest of the insert method can insert the element to its proper place.

2.2 Test Cases

In fact, simply adding an element and then checking its location proves the success of binarySearch method. But to complete the assignment I created two trees with elements inserted 1 to 11 and 20 to 30, respectively.
Since this method is private and works for insert method, I've overloaded this method to work like an adaptor for public use and checked 4 elements if they're in the tree or not.

2.3 Running Commands and Results



```
5 is in the bTree
9 is in the bTree
15 isn't in the bTree
25 is in the bTree
```

Checked these elements. 15 wasn't in that tree.

3 Project 9.5 in book

In this part we've asked to implement rest of AVLTree class and add another constructor to check if given Binary Tree is AVLTree or not. This constructor does nothing else but checking tree is AVL or not.

3.1 Problem Solution Approach

In this part I have coded 3 methods to check a given Binary Tree is AVL or not. To do so I checked its roots balance. If its balanced then it's an AVL tree. To do so, I used recursive getHeight and isAvl methods. And called them in constructor. Note that this constructor just checks if given tree is AVL or not. Does nothing else.
For implementation part of AVLTree class I've completed add method similarly to its item < data case. To do so I needed rebalanceRight and decrementBalance so I've implemented them as well by mirroring rebalanceLeft and incrementBalance respectively.
To implement delete method I've tried to mirror add method as well but I had issues with rebalancing tree on critical cases as shown in assignment pdf. So, to complete AVLTree class I've used book authors source codes to implement delete, findLargestChild, rebalanceLeftRight, rebalanceRightLeft and deleteNode methods. But after I've seen these method I can say to overcome given critical situations I realized that rebalanceRight and left wasn't enough on their own and under certain circumstances nodes needed to rotate in different combinations.

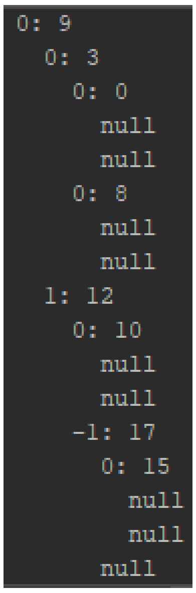
NOTE: Since book codes have them I cannot represent them as my own code. So, I do not expect to be graded from this portion of part 3. I've provided them just to show that I've understanded which changes must be made to balance tree after removal of certain important nodes. Constructor portion of this part(avlCheck, isAVL, getHeight) and completion/implementation of add, incrementBalance, rebalanceRight methods are provided by myself.

3.2 Test Cases

To test AVL checking i creted an AVL tree and a BinarySearchTree.

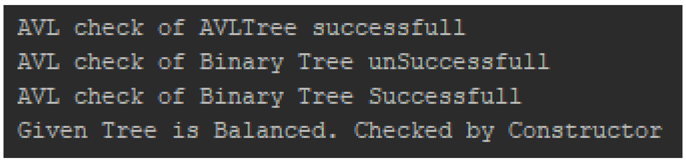
- Filled AVL tree randomly to test Add method.
- Checked the AVLtree to see if it's AVL or not.
- Filled Binary Search Tree unbalanced and checked if its AVL or not
- Filled its left side to be balanced and checked again if its AVL or not
- Constructed a new AVLTree from BinaryTree to check if its AVL or not

3.3 Running Commands and Results



```
0: 9
  0: 3
    0: 0
      null
      null
    0: 8
      null
      null
  1: 12
    0: 10
      null
      null
    -1: 17
      0: 15
        null
        null
      null
```

Randomly adds to AVLTree by calling add method.



```
AVL check of AVLTree successfull
AVL check of Binary Tree unSuccessfull
AVL check of Binary Tree Successfull
Given Tree is Balanced. Checked by Constructor
```

Checks an AVLTree whether if it’s an AVL tree or not by checkAVL static method.
Checks a BinarySearchTree whether if it’s an AVL tree or not by checkAVL static method.
Checks a BinarySearchTree whether if it’s an AVL tree or not by checkAVL static method.
Checks a BinarySearchTree whether if it’s an AVL tree or not by AVLTree constructor.