Mohammad Ali Jinnah University

Chartered by Government of Sindh - Recognized by HEC

Assignment 02

Name: Muhamad Fahad

Id: FA19-BSSE-0014

Subject: Data Structures and Algorithms Lab (CS 2511)

Lab Title: Balance Tree

Section: AM

Teacher: MUHAMMAD MUBASHIR KHAN

Date: Thursday, January 7, 2021

1. Implement AVL tree in Java.

Code:

```
package com.company.Tree;

import java.util.Random;

public class AVL_Tree {
    public Node root;

    public final static class Node{
        int value;
        int balance;
        Node left, right;

        public Node(int item) {
            value = item;
            left = right = null;
        }
    }

    void Inorder(Node node) {
        if (node == null)
            return;

        Inorder(node.left);
        System.out.println(node.value);
        Inorder(node.right);

    }  // left,root,right
    void Postorder(Node node) {
        if (node == null)
            return;

        Postorder(node.left);
        Postorder(node.right);

        System.out.print(node.value + "("+getBalance(node)+") ->");
    }  // left, right, root
    void Preorder(Node node) {
        if (node == null)
            return;

        System.out.print(node.value+" , ");
        Preorder(node.left);
        Preorder(node.right);

    }   // root, left, right

    void updateHeight(Node n) {
        n.balance = height(n.left) - height(n.right);
    }
    int height(Node n) {
        return n == null ? -1 : n.balance;
    }
```

```
int getBalance(Node N) {
    if (N == null)
        return 0;
    return height(N.left) - height(N.right);
}
private int max(int a, int b) {
    return (a > b) ? a : b;
}

Node rotateLeft(Node root) {
    Node newNode = root.right;
    Node temp = newNode.left;
    newNode.left = root;
    root.right = temp;

    // Update heights
    root.balance = max(height(root.left), height(root.right)) + 1;
    newNode.balance = max(height(newNode.left), height(newNode.right)) + 1;

    return newNode;
}
Node rotateRight(Node root) { // 5
    Node NewNode = root.left;   // 4.3.2.1
    Node temp = NewNode.right;  //null
    NewNode.right = root;       //
    root.left = temp;

    // Update heights
    root.balance = max(height(root.left), height(root.right)) + 1;
    NewNode.balance = max(height(NewNode.left), height(NewNode.right)) + 1;

    return NewNode;
}

Node rebalance(Node root){
    updateHeight(root);
    int balance = getBalance(root);

    if (balance > 1) {
        if (height(root.right.right) > height(root.right.left)) {
            root = rotateLeft(root);
            System.out.println("hi right");
        }
        else {
            root.right = rotateRight(root.right);
            root = rotateLeft(root);
        }
    } else if (balance < -1) {
        System.out.println("hi left");

        if (height(root.left.left) > height(root.left.right))
            root = rotateRight(root);
        else {
            root.left = rotateLeft(root.left);
            root = rotateRight(root);
        }
```

```
    }
    return root;
}

Node insert(Node node, int key) {
    if (node == null)
        return (new Node(key));

    if (key < node.value)
        node.left = insert(node.left, key);
    else if (key > node.value)
        node.right = insert(node.right, key);
    else
        return node;

    node.balance = 1 + max(height(node.left),
            height(node.right));

    int balance = getBalance(node);
    if (balance > 1) {
        if(key < node.left.value)
            return rotateRight(node);
        else if(key > node.left.value) {
            node.left = rotateLeft(node.left);
            return rotateRight(node);
        }
    }

    if (balance < -1){
        if(key > node.right.value)
            return rotateLeft(node);
        else if(key < node.right.value){
            node.right = rotateRight(node.right);
            return rotateLeft(node);
        }
    }

    return node;
}
void insert(int value){
    root = insert(root,value);
}

Node minValueNode(Node node) {
    Node current = node;

    while (current.left != null)
        current = current.left;

    return current;
}

Node deleteNode(Node root, int key) {
    if (root == null)
        return root;
```

```
        if (key < root.value)
            root.left = deleteNode(root.left, key);

        else if (key > root.value)
            root.right = deleteNode(root.right, key);

        else {
            Node temp = null;
            if ((root.left == null) || (root.right == null)){
                temp = (temp == root.left)?root.right:root.left;

                if (temp == null) {
                    temp = root;
                    root = null;
                }
                else
                    root = temp;
            }
            else {
                temp = minValueNode(root.right);
                root.value = temp.value;
                root.right = deleteNode(root.right, temp.value);
            }
        }

        if (root == null)
            return root;

        root.balance = max(height(root.right), height(root.left)) + 1;
        int balance = getBalance(root);

        if (balance > 1) {
            if(key < root.left.value)
                return rotateRight(root);
            else if(key > root.left.value) {
                root.left = rotateLeft(root.left);
                return rotateRight(root);
            }
        }
        if (balance < -1){
            if(key > root.right.value)
                return rotateLeft(root);
            else if(key < root.right.value){
                root.right = rotateRight(root.right);
                return rotateLeft(root);
            }
        }

        return root;
    }
    void delete(int key){
        root = deleteNode(root,key);
    }
}
```

```
class Test{
    public static void main(String[] args) {
//        Random rand = new Random();

        AVL_Tree tree = new AVL_Tree();
        for (int i = 0; i < 12; i++)
            tree.insert(i);

//            tree.insert(rand.nextInt(i));

        System.out.print("\nPreorder: ");
        tree.Preorder(tree.root);

        tree.delete(5);
        tree.delete(9);
        tree.delete(2);

        System.out.println("\nDeleting tree Node(5,9,2)....");
        System.out.println("Deleted :)");

        System.out.print("\nPreorder: ");
        tree.Preorder(tree.root);

//            System.out.print("\nPostorder: ");
//            tree.Postorder(tree.root);
    }
}
```

Output:

```
Preorder: 7 , 3 , 1 , 0 , 2 , 5 , 4 , 6 , 9 , 8 , 10 , 11 ,
Deleting tree Node(5,9,2)....
Deleted :)

Preorder: 7 , 3 , 1 , 0 , 6 , 4 , 10 , 8 , 11 ,
Process finished with exit code 0
```

2. Implement Red-Back tree in Java.

Code:

```
package com.company.Tree;


import java.util.Scanner;

public class RedBlackTree {

    private final int RED = 0;
    private final int BLACK = 1;

    private class Node {

        int key = -1, color = BLACK;
        Node left = nil, right = nil, parent = nil;

        Node(int key) {
            this.key = key;
        }
    }

    private final Node nil = new Node(-1);
    private Node root = nil;

    public void printTree(Node node) {
        if (node == nil) {
            return;
        }
        printTree(node.left);
        System.out.print(((node.color==RED)?"Color: Red ":"Color: Black ")+"Key: "+node.key+" Parent:
"+node.parent.key+"\n");
        printTree(node.right);
    }

    private Node findNode(Node findNode, Node node) {
        if (root == nil) {
            return null;
        }

        if (findNode.key < node.key) {
            if (node.left != nil) {
                return findNode(findNode, node.left);
            }
        } else if (findNode.key > node.key) {
            if (node.right != nil) {
                return findNode(findNode, node.right);
            }
        } else if (findNode.key == node.key) {
            return node;
        }
        return null;
    }
```

```
private void insert(Node node) {
    Node temp = root;
    if (root == nil) {
        root = node;
        node.color = BLACK;
        node.parent = nil;
    } else {
        node.color = RED;
        while (true) {
            if (node.key < temp.key) {
                if (temp.left == nil) {
                    temp.left = node;
                    node.parent = temp;
                    break;
                } else {
                    temp = temp.left;
                }
            } else if (node.key >= temp.key) {
                if (temp.right == nil) {
                    temp.right = node;
                    node.parent = temp;
                    break;
                } else {
                    temp = temp.right;
                }
            }
        }
        fixTree(node);
    }
}

//Takes as argument the newly inserted node
private void fixTree(Node node) {
    while (node.parent.color == RED) {
        Node uncle = nil;
        if (node.parent == node.parent.parent.left) {
            uncle = node.parent.parent.right;

            if (uncle != nil && uncle.color == RED) {
                node.parent.color = BLACK;
                uncle.color = BLACK;
                node.parent.parent.color = RED;
                node = node.parent.parent;
                continue;
            }
            if (node == node.parent.right) {
                //Double rotation needed
                node = node.parent;
                rotateLeft(node);
            }
            node.parent.color = BLACK;
            node.parent.parent.color = RED;
            //if the "else if" code hasn't executed, this
            //is a case where we only need a single rotation
            rotateRight(node.parent.parent);
```

```
        } else {
            uncle = node.parent.parent.left;
            if (uncle != nil && uncle.color == RED) {
                node.parent.color = BLACK;
                uncle.color = BLACK;
                node.parent.parent.color = RED;
                node = node.parent.parent;
                continue;
            }
            if (node == node.parent.left) {
                //Double rotation needed
                node = node.parent;
                rotateRight(node);
            }
            node.parent.color = BLACK;
            node.parent.parent.color = RED;
            rotateLeft(node.parent.parent);
        }
    }
    root.color = BLACK;
}

void rotateLeft(Node node) {
    if (node.parent != nil) {
        if (node == node.parent.left) {
            node.parent.left = node.right;
        } else {
            node.parent.right = node.right;
        }
        node.right.parent = node.parent;
        node.parent = node.right;
        if (node.right.left != nil) {
            node.right.left.parent = node;
        }
        node.right = node.right.left;
        node.parent.left = node;
    } else {
        Node right = root.right;
        root.right = right.left;
        right.left.parent = root;
        root.parent = right;
        right.left = root;
        right.parent = nil;
        root = right;
    }
}

void rotateRight(Node node) {
    if (node.parent != nil) {
        if (node == node.parent.left) {
            node.parent.left = node.left;
        } else {
            node.parent.right = node.left;
        }

        node.left.parent = node.parent;
```

```
        node.parent = node.left;
        if (node.left.right != nil) {
            node.left.right.parent = node;
        }
        node.left = node.left.right;
        node.parent.right = node;
    } else {//Need to rotate root
        Node left = root.left;
        root.left = root.left.right;
        left.right.parent = root;
        root.parent = left;
        left.right = root;
        left.parent = nil;
        root = left;
    }
}

//Deletes whole tree
void deleteTree(){
    root = nil;
}

//Deletion Code .

//This operation doesn't care about the new Node's connections
//with previous node's left and right. The caller has to take care
//of that.
void transplant(Node target, Node with){
    if(target.parent == nil){
        root = with;
    }else if(target == target.parent.left){
        target.parent.left = with;
    }else
        target.parent.right = with;
    with.parent = target.parent;
}

boolean delete(Node z){
    if((z = findNode(z, root))==null) return false;
    Node x;
    Node y = z;
    int y_original_color = y.color;

    if(z.left == nil){
        x = z.right;
        transplant(z, z.right);
    }else if(z.right == nil){
        x = z.left;
        transplant(z, z.left);
    }else{
        y = treeMinimum(z.right);
        y_original_color = y.color;
        x = y.right;
        if(y.parent == z)
            x.parent = y;
        else{
```

```
            transplant(y, y.right);
            y.right = z.right;
            y.right.parent = y;
        }
        transplant(z, y);
        y.left = z.left;
        y.left.parent = y;
        y.color = z.color;
    }
    if(y_original_color==BLACK)
        deleteFixup(x);
    return true;
}

void deleteFixup(Node x){
    while(x!=root && x.color == BLACK){
        if(x == x.parent.left){
            Node w = x.parent.right;
            if(w.color == RED){
                w.color = BLACK;
                x.parent.color = RED;
                rotateLeft(x.parent);
                w = x.parent.right;
            }
            if(w.left.color == BLACK && w.right.color == BLACK){
                w.color = RED;
                x = x.parent;
                continue;
            }
            else if(w.right.color == BLACK){
                w.left.color = BLACK;
                w.color = RED;
                rotateRight(w);
                w = x.parent.right;
            }
            if(w.right.color == RED){
                w.color = x.parent.color;
                x.parent.color = BLACK;
                w.right.color = BLACK;
                rotateLeft(x.parent);
                x = root;
            }
        }else{
            Node w = x.parent.left;
            if(w.color == RED){
                w.color = BLACK;
                x.parent.color = RED;
                rotateRight(x.parent);
                w = x.parent.left;
            }
            if(w.right.color == BLACK && w.left.color == BLACK){
                w.color = RED;
                x = x.parent;
                continue;
            }
            else if(w.left.color == BLACK){
```

```
                w.right.color = BLACK;
                w.color = RED;
                rotateLeft(w);
                w = x.parent.left;
            }
            if(w.left.color == RED){
                w.color = x.parent.color;
                x.parent.color = BLACK;
                w.left.color = BLACK;
                rotateRight(x.parent);
                x = root;
            }
        }
    }
    x.color = BLACK;
}

Node treeMinimum(Node subTreeRoot){
    while(subTreeRoot.left!=nil){
        subTreeRoot = subTreeRoot.left;
    }
    return subTreeRoot;
}

public void consoleUI() {
    Scanner scan = new Scanner(System.in);
    int item;
    Node node;

    while (true) {
        System.out.println("\n1.- Add items\n"
                + "2.- Delete items\n"
                + "3.- Check items\n"
                + "4.- Print tree\n"
                + "5.- Delete tree\n");
        int choice = scan.nextInt();

        switch (choice) {
            case 1:
                System.out.print("Enter: ");
                item = scan.nextInt();
                node = new Node(item);
                insert(node);
                printTree(root);
                break;
            case 2:
                System.out.print("Enter: ");
                item = scan.nextInt();
                System.out.print("\nDeleting item: " + item);
                System.out.println((delete(new Node(item))?": deleted!":": does not exist!"));
                printTree(root);
                break;
            case 3:
                System.out.print("Enter: ");
                item = scan.nextInt();
                System.out.println("\nfind Node: " + item);
```

```
                System.out.println((findNode(new Node(item), root) != null) ? "found" : "not found");
                break;
            case 4:
                printTree(root);
                break;
            case 5:
                deleteTree();
                System.out.println("Tree deleted!");
                break;
        }
      }
    }

    public static void main(String[] args) {
      RedBlackTree rbt = new RedBlackTree();
      rbt.consoleUI();
    }
}
```

Output:

```
"C:\Program Files\Java\jdk-13.0.2

1.- Add items
2.- Delete items
3.- Check items
4.- Print tree
5.- Delete tree

1
Enter: 5
Color: Black Key: 5 Parent: -1
```

```
1
Enter: 5
Color: Black Key: 5 Parent: -1

1.- Add items
2.- Delete items
3.- Check items
4.- Print tree
5.- Delete tree

1
Enter: 2
Color: Red Key: 2 Parent: 5
Color: Black Key: 5 Parent: -1

1.- Add items
2.- Delete items
3.- Check items
4.- Print tree
5.- Delete tree

1
Enter: 6
Color: Red Key: 2 Parent: 5
Color: Black Key: 5 Parent: -1
Color: Red Key: 6 Parent: 5

1.- Add items
2.- Delete items
3.- Check items
4.- Print tree
5.- Delete tree

2
Enter: 6

Deleting item: 6: deleted!
Color: Red Key: 2 Parent: 5
Color: Black Key: 5 Parent: -1

1.- Add items
2.- Delete items
3.- Check items
4.- Print tree
5.- Delete tree

3
Enter: 5

find Node: 5
found
```

```
1.- Add items
2.- Delete items
3.- Check items
4.- Print tree
5.- Delete tree

4
Color: Red Key: 2 Parent: 5
Color: Black Key: 5 Parent: -1

1.- Add items
2.- Delete items
3.- Check items
4.- Print tree
5.- Delete tree

5
Tree deleted!

1.- Add items
2.- Delete items
3.- Check items
4.- Print tree
5.- Delete tree
```