DOKUZ EYLUL UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CME2201 DATA STRUCTURES
REPORT
Homework – II

RedBlackTree and Augmented RedBlackTree

by
Mehmet Hamdi Uslu
2400000145

Lecturers
Asst.Prof. Zerrin IŞIK
Res.Asst. Ali CÜVİTOĞLU
IZMIR

18.12.2018

1.Introduction

We are expected to implement a balanced binary search tree (specifically a Red-Black (RB) - -Tree) which performs operations and queries displayed below. Then you are required to improve it by augmenting the tree structure. You must insert each line in “people.txt” to both RB-Tree and Augmented RB-tree. Assignment is coded in Java.

2.Tasks

Task 1: Design and code a RBTree data structure with the following capabilities:

INSERT: insert an input item (each line in ‘people.txt’)

GETNUMSMALLER1: get the number of people who are born before a given input date

GETNUMSMALLER2: get the number of people who are born before a person with a given id

GETMAX: get maximum aged (oldest) person and print the id and birthdate of the person

GETMIN: get minimum aged (youngest) person and print the id and birthdate of the person

-GETNUM: get number of all people in the tree by using inorder tree walk.

PRINT: display all items of the tree by applying inorder tree walk.

Task 2: The program should run the methods in the following order for testing.

Insert all elements in the input file (people.txt)

Print the result of GETNUMSMALLER1 for the item with value 7/6/1991

Print the result of GETNUMSMALLER2 for the item with value 9988

Print the maximum age (with the birthdate and id)

Print the minimum age (with the birthdate and id)

Print the number of all items (GETNUM)

Task 3: Improve the RB−Tree class by augmenting the structure so that each node holds an additional field: the number of all nodes (age) that are smaller than the current node’s age. Note that you need to update this value on each insert operation. You must update some of operations for this Augmented RB-tree. Not all operations will be the same as in the original RB-tree. Consider the additional field and plan which operations should be updated.

3. Design and Coding

Firstly, we have been given a base code as a working ground. I started designing RB Tree part before Augmented RB Tree because I could extend and use every function and algorithm on Augmented RB Tree. All functions on this assignment runs on generic forms. Base functions I used are can be seen on Figure-1:

```
void case1 (String line) throws IOException {
void case2 (String entry) {
void case3 (int ids) {
void case4 () {
void case5 () {
void case6 () {
void case7 () {
void case8(String file) throws IOException {

public <I> void consoleUI() throws IOException {

private void ReadDoc(String file) throws IOException {

private void Inserter(String readline) throws IOException {

private void getNumSmaller1(String entry) {

private void getNumSmaller2(int id) {

private void RecursiveFindbyAge(Node temp, double age) {

private double AgeFinder(String dateold) {

private Node treeMinimum(Node subTreeRoot) {

private Node treeMaximum(Node subTreeRoot) {

private void howmanyPeopleinTree(Node rooter) {

void RBTchecklist() {
```

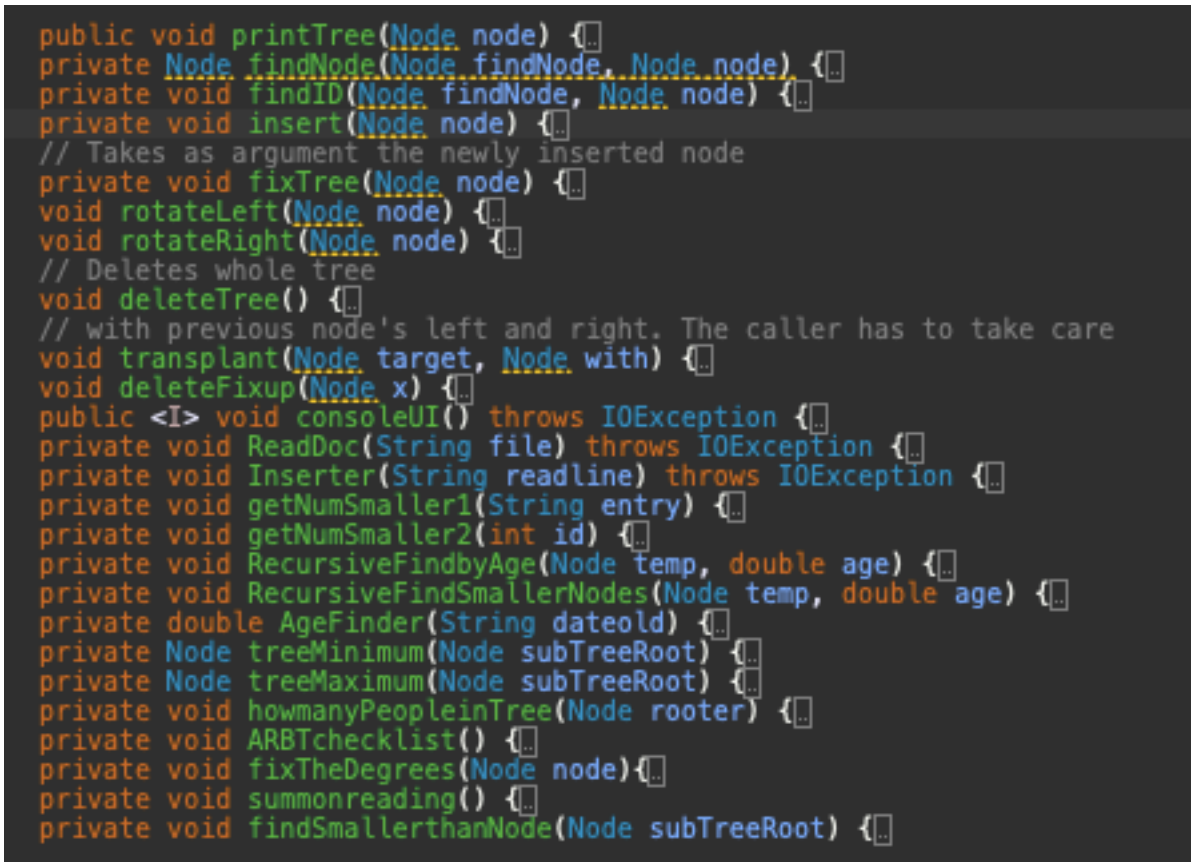
Figure -1

Void cases seen on figure-1 are the callsigns for functions menu created on AUG-RB-Tree. Each one when called on UI they are printed/processed simultaneously for both simultaneously AUG-RB-Tree and RB-Tree.

Secondly, I started working on implementing needed algorithms on AUG-RB Tree as can be seen on figure-2

Functions changed on AUG-RB-Tree are:

- getNumSmaller1:
- getNumSmaller2



```
public void printTree(Node node) {
private Node findNode(Node findNode, Node node) {
private void findID(Node findNode, Node node) {
private void insert(Node node) {
// Takes as argument the newly inserted node
private void fixTree(Node node) {
void rotateLeft(Node node) {
void rotateRight(Node node) {
// Deletes whole tree
void deleteTree() {
// with previous node's left and right. The caller has to take care
void transplant(Node target, Node with) {
void deleteFixup(Node x) {
public <I> void consoleUI() throws IOException {
private void ReadDoc(String file) throws IOException {
private void Inserter(String readline) throws IOException {
private void getNumSmaller1(String entry) {
private void getNumSmaller2(int id) {
private void RecursiveFindbyAge(Node temp, double age) {
private void RecursiveFindSmallerNodes(Node temp, double age) {
private double AgeFinder(String dateold) {
private Node treeMinimum(Node subTreeRoot) {
private Node treeMaximum(Node subTreeRoot) {
private void howmanyPeopleinTree(Node rooter) {
private void ARBTchecklist() {
private void fixTheDegrees(Node node){
private void summonreading() {
private void findSmallerthanNode(Node subTreeRoot) {
```

Figure -2

these values can be calculated easier than RB-Tree on AUG-RB-Tree reason of that is every node on AUG-RB-Tree holds additional information about binary tree which is the number of people who are born before a given input date.

Additionally couple really important functions were added and those are:

- findSmallerthanNode: which is, function that sends needed parameters to RecursiveFindSmallerNodes.
- RecursiveFindSmallerNodes: which is, finds smaller nodes than given node.
- FixTheDegres: which is, the fixing function for extra information that hold by AUG-RB-Tree.
- SummonReading: which is, callsigns of reading document functions for both AUG RB-Tree and RB-Tree.

You can see diffrences between AUG-RB-Tree and RB-Tree nodes on Figure 3-4 .

```
private class Node<I, A, B, C> {

    double key = -1, color = BLACK;
    I id;
    A age;
    B bdate;
    C scount;
    Node left = nil, right = nil, parent = nil;

    Node(final double key, final I id, final A age, final B bdate, final C scount) {

        this.key = key;
        this.id = id;
        this.age = age;
        this.bdate = bdate;
        this.scount = scount;
    }
```

Figure -4

```
private class Node<I, A, B> {

    double key = -1, color = BLACK;
    I id;
    A age;
    B bdate;
    Node left = nil, right = nil, parent = nil;
    int lolo = 0;


    Node(final double key, final I id, final A age, final B bdate) {

        this.key = key;
        this.id = id;
        this.age = age;
        this.bdate = bdate;
    }
```

Figure -3

All other functions are can work on inherited from RB-Tree class such as:

- ReadDoc: which is, reading of documents for both trees.
- Inserted: which is, singular inserter of tree.
- RecursiveFindbyAge: which is, age finder for both trees
- Tree Max-Min: which is, max value and min value of trees.
- howmanyPeopleinTree: which is, Node counter of both classes.
- CheckLists: which is, printers of needed launch screen statistics.

These were the everything used in this assigned shortly and finally my main class can be seen on Figure-5 and void cases that are summoned on AUG-RB-Tree UI can be seen on Figure-6 and Figure-7 which are called and used on UI.

```
public static void main(String[] args) throws IOException {
    AugmentedRedBlackTree augrbt = new AugmentedRedBlackTree();
    augrbt.consoleUI();
}
```

Figure-5

```
void case1 (String line) throws IOException {
    Inserter(line);
}
void case2 (String entry) {

    try {
        getNumSmaller1(entry);

    } catch (Exception e) {

    }
}
void case3 (int ids) {
    try {
        getNumSmaller2(ids);
    } catch (Exception e) {
        // TODO: handle exception
    }
}
```

Figure -6

```
switch (choice) {
case 1:
    System.out.print("Please enter ID,dd/MM/yyyy format: ");
    String line = scan.next();

    rbtt.case1(line);

    Inserter(line);
    break;
case 2:
    String entry = null;
    System.out.print("Please enter dd/MM/yyyy format: ");
    try {

        entry = scan.next();
        System.out.println("RBT OUTPUT");
        rbtt.case2(entry);
        System.out.println("AUG-RBT OUTPUT");
        getNumSmaller1(entry);

    } catch (Exception e) {
        System.out.println("Please use designated format");
    }

    break;
```

Figure -7

As conclusion, AUG-RB-Tree makes couple methods run faster and more effective reason of this difference is every information needed is hold on Node of tree so, tree lets user print them or use them on a different way user needs for their future desires. This is an advantage. However, any mistake on this process can be lethal, node's counter which is embedded on node runs recursively any mistake/bug can poison every node on tree. When used well Augmented Structures are more efficient.