Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки Кафедра ІПІ

Звіт

з лабораторної роботи №7
з дисципліни «Алгоритми та структури даних 2. Структури даних»

«Бінарні дерева пошуку»

Виконав(ла) *ІП-22, Андреєва Уляна Андріївна*

(шифр, прізвище, ім'я, по батькові)

Перевірила *Халус Олена Андріївна*

(прізвище, ім'я, по батькові)

Київ 2023

ЗМІСТ

Практичне завдання №7

“Бінарні дерева пошуку”

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи -розв'язати наступну задачу: переписати значення вузлів у бінарному дереві та знайти суми послідовних вузлів у цьому дереві.

2 ЗАВДАННЯ

У даній роботі необхідно виконати два завдання:

1. Перетворити вхідне бінарне дерево у бінарне дерево пошуку

На вхід подається деяке бінарне дерево, із фіксованою структурою (тобто зв’язками між вузлами, їх батьком та нащадками). Необхідно переписати значення вузлів дерева таким чином, щоби:
a) їх нові значення брались тільки з того набору, який присутній у вхідному дереві;
b) зберігалась внутрішня структура дерева (зв’язки між вузол-батько та вузол-нащадки).

2. Пошук сум послідовних вузлів в дереві

Після того, як вхідне дерево перетворене на бінарне дерево пошуку, необхідно розв’язати
наступну задачу. Додатково задається деяке число S. В отриманому бінарному дереві пошуку необхідно знайти всі такі монотонні шляхи (які не обов'язково йдуть від кореня, але всі прямують згори вниз), що сума значень вузлів, які належать знайденим шляхам, дорівнює числу S.

3.1 Псевдокод алгоритму

Обхід упорядкованого бінарного дерева (In-order Traversal)

```
procedure inOrderTraversal(root, values)
    if root = null then
        return

    inOrderTraversal(root.left, values)
    values.add(root.value)
    inOrderTraversal(root.right, values)
```

Заміна значень вузлів бінарного дерева

```
procedure replaceNodeValues(root, iterator)
    if root = null then
        return

    replaceNodeValues(root.left, iterator)
    root.value = iterator.next()
    replaceNodeValues(root.right, iterator)
```

Пошук шляхів в бінарному дереві з заданою сумою

```
function findPaths(root, targetSum)
    paths = empty list
    currentPath = empty list
    findPathsRecursive(root, targetSum, currentPath, paths)
    return paths

procedure findPathsRecursive(node, targetSum, currentPath, paths)
    if node = null then
        return

    currentPath.add(node.value)

    currentSum = 0
    for i = currentPath.size() - 1 downto 0 do
        currentSum = currentSum + currentPath[i]
        if currentSum = targetSum then
            paths.add(copy of currentPath[i to end])

    findPathsRecursive(node.left, targetSum, currentPath, paths)
    findPathsRecursive(node.right, targetSum, currentPath, paths)

    currentPath.removeLast()
```

3.2 Вихідний код

```java
import java.io.*;
import java.util.*;

public class BinarySearchTreeConverter {
    public static void convertToBST(Node root) {
        List<Integer> values = new ArrayList<>();
        inOrderTraversal(root, values);
        quickSort(values, 0, values.size() - 1);

        Iterator<Integer> iterator = values.iterator();
```

```
        replaceNodeValues(root, iterator);
    }

    private static void quickSort(List<Integer> values, int low, int high)
{
        if (low < high) {
            int pivotIndex = partition(values, low, high);
            quickSort(values, low, pivotIndex - 1);
            quickSort(values, pivotIndex + 1, high);
        }
    }

    private static int partition(List<Integer> values, int low, int high) {
        int pivot = values.get(high);
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (values.get(j) < pivot) {
                i++;
                swap(values, i, j);
            }
        }

        swap(values, i + 1, high);
        return i + 1;
    }

    private static void swap(List<Integer> values, int i, int j) {
        int temp = values.get(i);
        values.set(i, values.get(j));
        values.set(j, temp);
    }

    private static void inOrderTraversal(Node root, List<Integer> values) {
        if (root == null) {
            return;
        }

        inOrderTraversal(root.left, values);
        values.add(root.value);
        inOrderTraversal(root.right, values);
    }

    private static void replaceNodeValues(Node root, Iterator<Integer>
iterator) {
        if (root == null) {
            return;
        }

        replaceNodeValues(root.left, iterator);
        root.value = iterator.next();
        replaceNodeValues(root.right, iterator);
    }

    public static void printTree(Node root) {
        printTreeRecursive(root, 0);
    }

    private static void printTreeRecursive(Node current, int level) {
        if (current != null) {
            printTreeRecursive(current.right, level + 1);
```

```
            for (int i = 0; i < level; i++) {
                System.out.print("    ");
            }

            System.out.println(current.value);

            printTreeRecursive(current.left, level + 1);
        }
    }

    public static List<List<Integer>> findPaths(Node root, int targetSum) {
        List<List<Integer>> paths = new ArrayList<>();
        List<Integer> currentPath = new ArrayList<>();
        findPathsRecursive(root, targetSum, currentPath, paths);
        return paths;
    }

    private static void findPathsRecursive(Node node, int targetSum,
List<Integer> currentPath, List<List<Integer>> paths) {
        if (node == null)
            return;

        currentPath.add(node.value);

        int currentSum = 0;
        for (int i = currentPath.size() - 1; i >= 0; i--) {
            currentSum += currentPath.get(i);
            if (currentSum == targetSum)
                paths.add(new ArrayList<>(currentPath.subList(i,
currentPath.size())));
        }

        findPathsRecursive(node.left, targetSum, currentPath, paths);
        findPathsRecursive(node.right, targetSum, currentPath, paths);

        currentPath.remove(currentPath.size() - 1);
    }
}

class Main{
    static String readFile(String fileName){
        String fileContent = "";

        try {
            File file = new File(fileName);
            Scanner scanner = new Scanner(file);

            while (scanner.hasNextLine()) {
                fileContent = scanner.nextLine();
            }

            scanner.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        return fileContent;
    }

    static int[] convertToInts(String fileContent){
```

```
        String[] numbersString = fileContent.split(" ");
        int[] numbers = new int[numbersString.length];
        for (int i = 0; i < numbersString.length; i++) {
            numbers[i] = Integer.parseInt(numbersString[i]);
        }

        return numbers;
    }

    static void writePathsToFile(List<List<Integer>> paths){
        String fileName = "output.txt";

        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(fileName))) {
            for (List<Integer> row : paths) {
                for (int i = 0; i < row.size(); i++) {
                    writer.write(Integer.toString(row.get(i)));
                    if (i != row.size() - 1) {
                        writer.write(" ");
                    }
                }
                writer.newLine();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        String fileName = args[0];
        int targetSum = Integer.parseInt(args[1]);

        String fileContent = readFile(fileName);
        System.out.println("Input.txt: " + fileContent);

        int[] numbers = convertToInts(fileContent);

        Node root = BinaryTreeConverter.convertToBST(numbers);

        System.out.println("Binary tree from an array:");
        BinarySearchTreeConverter.printTree(root);

        BinarySearchTreeConverter.convertToBST(root);

        System.out.println("\nConverted binary search tree:");
        BinarySearchTreeConverter.printTree(root);

        List<List<Integer>> paths =
BinarySearchTreeConverter.findPaths(root, targetSum);

        System.out.println("Paths with sum " + targetSum + ":");
        for (List<Integer> path : paths) {
            System.out.println(path);
        }

        writePathsToFile(paths);
    }
}
```

```
public class BinaryTreeConverter {
    private static int index;

    public static Node convertToBST(int[] arr) {
        index = 0;
        return buildBST(arr);
    }

    private static Node buildBST(int[] arr) {
        if (index >= arr.length || arr[index] == 0) {
            index++;
            return null;
        }

        Node node = new Node(arr[index]);
        index++;

        node.left = buildBST(arr);
        node.right = buildBST(arr);

        return node;
    }

    public static void inOrderTraversal(Node root) {
        if (root == null) {
            return;
        }

        inOrderTraversal(root.left);
        System.out.print(root.value + " ");
        inOrderTraversal(root.right);
    }


}
```

```
class Node {
    int value;
    Node left;
    Node right;

    public Node(int value) {
        this.value = value;
        this.left = null;
        this.right = null;
    }
}
```

3.2.1 Програмна реалізація

Input_txt(10a)

input.txt

```
1 4 6 10 0 0 0 7 0 8 0 0 2 5 0 0 3 9 0 0 0
```

Solution

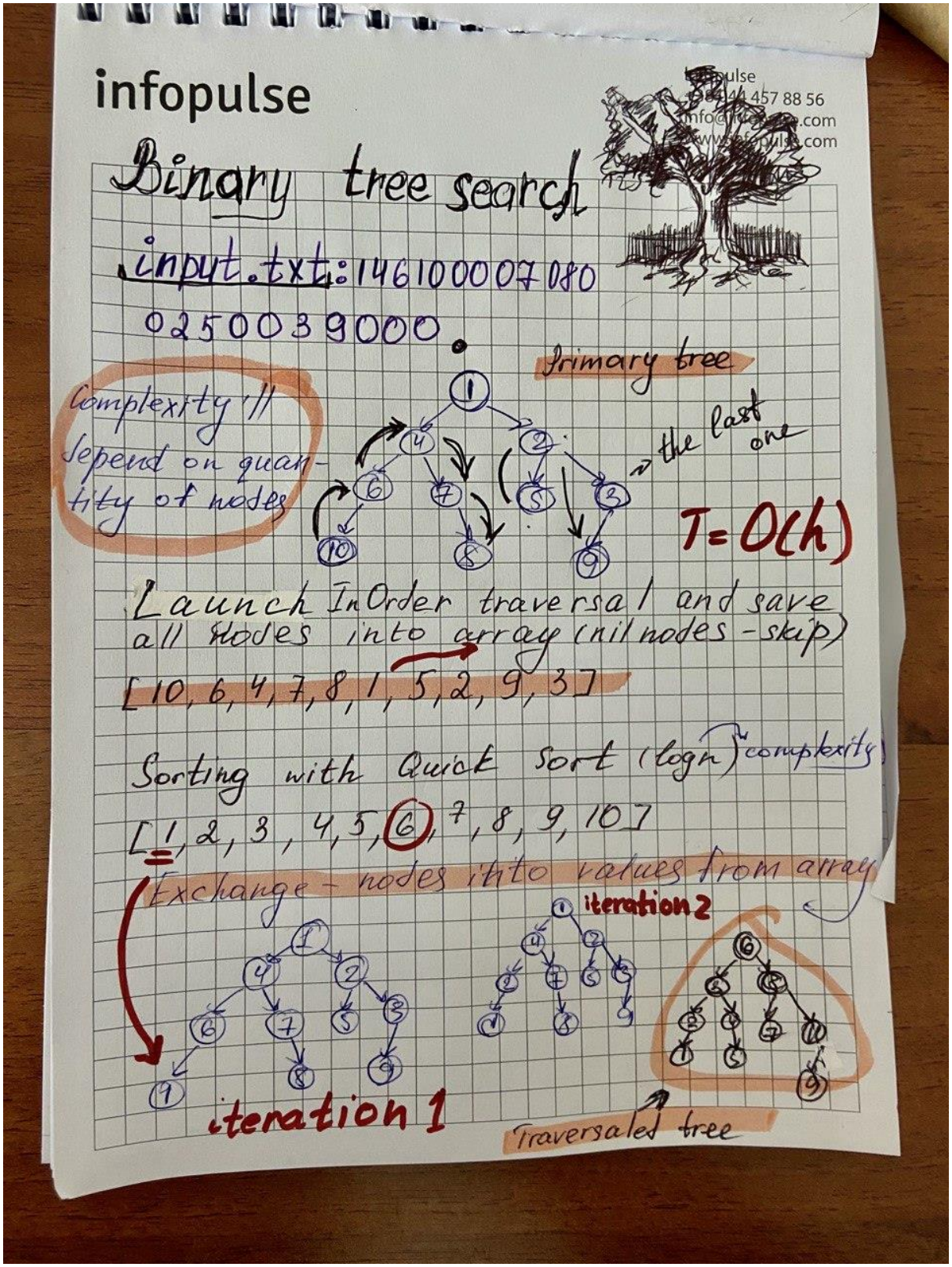
```
mac@MacBook-Pro-mac src % java Main input.txt 9
Input.txt: 1 4 6 10 0 0 0 7 0 8 0 0 2 5 0 0 3 9 0 0 0
Binary tree from an array:
        3
            9
    2
        5
1
            8
        7
    4
        6
            10

Converted binary search tree:
        10
            9
    8
        7
6
            5
        4
    3
        2
            1
Paths with sum 9:
[6, 3]
[4, 5]
[9]
mac@MacBook-Pro-mac src %
```

Output_txt(10a)

output.txt

```
6 3
4 5
9
```

Manual solution



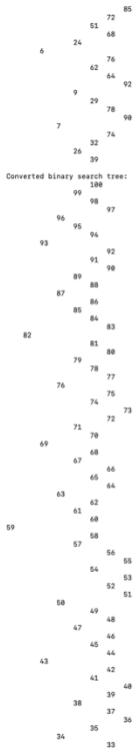
Input_100b.txt(example)

input.txt

```
1 2 6 7 26 39 0 0 32 0 74 0 0 9 29 78 90 0 0 0 0 62 64 92 0 0 0 76 0 0 14 24 0 51 68 0 0 72 0 85 0 88
0 0 36 56 91 0 0 0 46 69 0 0 47 61 0 0 0 3 11 13 0 23 31 0 0 100 0 0 40 49 0 87 95 0 0 0 63 79 0 80 0
0 86 0 0 19 22 71 97 0 0 82 0 0 54 59 0 0 0 25 34 48 58 0 0 57 0 0 53 73 0 0 0 52 0 0 4 8 15 65 84 0
0 96 0 0 18 20 43 0 0 83 0 0 38 0 0 17 41 67 0 0 45 60 0 89 0 0 70 0 0 33 50 66 0 0 0 42 93 0 0 0 5
16 21 55 77 0 0 0 94 0 0 30 75 0 0 35 81 0 0 99 0 0 10 37 44 0 0 0 12 27 98 0 0 0 28 0 0
```

Solution

```
mac@MacBook-Pro-mac src % java Main input.txt 78
Input.txt: 1 2 6 7 26 39 0 0 32 0 74 0 0 9 29 78 90 0 0 0 0 62 64 92 0 0 0 76 0 0 14 24 0 51 68 0 0 72 0 85 0 88 0 0 36 56 91 0 0 0 46 69 0 0 47 61 0 0 0 3 11 13 0 23 31 0 0 100 0 0 40 49 0 87 95 0 0 0 63 79 0 80 0 0 86 0 0 19 22 71 97 0 0 82 0 0 54 59 0 0 0 25 34 48 58 0 0 57 0 0 53 7
3 0 0 0 52 0 0 4 8 15 65 84 0 0 96 0 0 18 20 43 0 0 83 0 0 38 0 0 17 41 67 0 0 45 60 0 89 0 0 70 0 0 33 50 66 0 0 0 42 93 0 0 0 5 16 21 55 77 0 0 0 94 0 0 30 75 0 0 35 81 0 0 99 0 0 10 37 44 0 0 0 12 27 98 0 0 0 28 0 0
Binary tree from an array:
                    28
                12
                    27
                        98
            10
                37
                    44
        5
                        99
                    35
                        81
                30
                    75
            16
                    94
                21
                    55
                        77
    4
                    42
                        93
                33
                    50
                        66
            17
                        70
                    45
                            89
                        60
                41
                    67
        8
                    38
                18
                        83
                    20
                        43
            15
                    96
                65
                    84
1
                    52
                25
                        53
                            73
                    34
                            57
                        48
                            58
            19
                    54
                        59
                22
                        82
                    71
                        97
        3
                        86
                    63
                            80
                        79
                40
                        87
                            95
                    49
            11
                        100
                    23
                        31
                13
    2
                        47
                            61
                    46
                        69
                36
                    56
                        91
            14
                                88
                            85
                        72
                    51
                        68
```




```
Paths with sum 78:
[29, 14, 5, 9, 8, 7, 6]
[24, 26, 28]
[78]
```

Output_100b.txt(example)

output.txt

```
29 14 5 9 8 7 6
24 26 28
78
```

Обхід бінарного дерева в прямому (pre-order) порядку означає, що спочатку відвідується кореневий вузол, потім вузол лівого піддерева і нарешті вузол правого піддерева. У коді цей обхід реалізований методом **printPreOrderRecursive()**, який виводить значення кожного вузла у вказаному порядку.

Обхід бінарного дерева в впорядкованому (in-order) порядку означає, що спочатку відвідується вузол лівого піддерева, потім кореневий вузол і нарешті вузол правого піддерева. У коді цей обхід реалізований методом

printInOrderRecursive(), який також виводить значення кожного вузла у вказаному порядку.

ВИСНОВОК

У даній задачі використовується бінарне дерево для зберігання послідовності чисел. Кожне число вставляється у відповідну позицію в дереві. Далі виконується обхід бінарного дерева та знаходяться всі можливі послідовності чисел, які мають задану суму. Результати зберігаються у вигляді списку списків.