

Міністерство освіти і науки України
**Національний технічний університет України «Київський
політехнічний інститут імені Ігоря Сікорського»**
Факультет інформатики та обчислювальної техніки Кафедра ІІІ
Звіт

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

Виконав(ла) ІІІ-22, Андреева Уляна Андріївна
(шифр, прізвище, ім'я, по батькові)

Перевірила Халус Олена Андріївна
(прізвище, ім'я, по батькові)

Київ 2023

ЗМІСТ

МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
ЗАВДАННЯ	4
ВИКОНАННЯ	9
1. 3.1 ПСЕВДОКОД АЛГОРИТМУ	9
2. 3.2 ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	9
3.2.1 Вихідний код	9
ВИСНОВОК	11

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:

- а) їх нові значення брались тільки з того набору, який присутній у вхідному дереві;
- б) 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

```
method insert(value)
    root = insertRec(root, value)

method insertRec(current, value)
    if value == 0 then
        return current
    end if
    if current == null then
        return new Node(value)
    end if
```

```

if value < current.value then
    current.left = insertRec(current.left, value)
else if value > current.value then
    current.right = insertRec(current.right, value)
end if

return current

```

(root)

```

method printInOrderRecursive(current)
    if current != null then
        printInOrderRecursive(current.left)
        print current.value
        printInOrderRecursive(current.right)
    end if

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

    currentPath.add(node.value)

    currentSum = 0
    for i = currentPath.size() - 1 to 0 step -1 do
        currentSum += currentPath.get(i)
        if currentSum == targetSum then
            paths.add(new ArrayList<>(currentPath.subList(i,
currentPath.size())))
        end if
    end for

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

    currentPath.remove(currentPath.size() - 1)
method findPaths(targetSum)
    paths = new ArrayList<>()
    currentPath = new ArrayList<>()
    findPathsRecursive(root, targetSum, currentPath, paths)
    return paths

```

3.2 Вихідний код

```
import java.util.ArrayList;
import java.util.List;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

class BinaryTree {
    Node root;

    public BinaryTree() {
        this.root = null;
    }

    public void insert(int value) {
        this.root = insertRec(this.root, value);
    }

    private Node insertRec(Node current, int value) {
        if (value == 0) {
            return current;
        }
        if (current == null) {
            return new Node(value);
        }

        if (value < current.value) {
            current.left = insertRec(current.left, value);
        } else if (value > current.value) {
            current.right = insertRec(current.right, value);
        }

        return current;
    }

    public void printInOrder() {
        printInOrderRecursive(this.root);
    }

    private void printInOrderRecursive(Node current) {
        if (current != null) {
            printInOrderRecursive(current.left);
            System.out.print(current.value + " ");
            printInOrderRecursive(current.right);
        }
    }

    private 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);
    }

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

    public void printPreOrder() {
        printPreOrderRecursive(this.root);
    }

    private void printPreOrderRecursive(Node current) {
        if (current != null) {
            System.out.print(current.value + " ");
            printPreOrderRecursive(current.left);
            printPreOrderRecursive(current.right);
        }
    }
}

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);

        BinaryTree tree = new BinaryTree();

        for (int number : numbers) {
            tree.insert(number);
        }

        System.out.println("BinaryTree in Inorder traversal: ");
        tree.printInOrder();
        System.out.println();

        System.out.println("BinaryTree Preorder traversal:");
        tree.printPreOrder();
        System.out.println();

        List<List<Integer>> paths = tree.findPaths(targetSum);

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

        writePathsToFile(paths);
    }
}

```

```

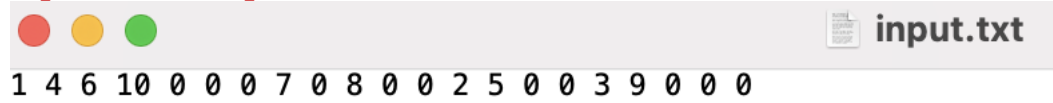
class Node {
    int value;
    Node left;
    Node right;

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

```

3.2.1 Програмна реалізація

Input_txt(example)



input.txt

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

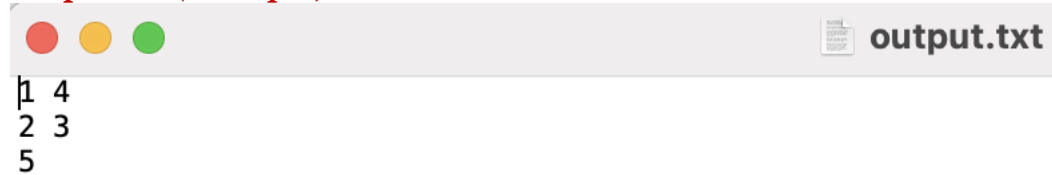
Solution

```

[mac@MacBook-Pro-mac src % java Main input.txt 5
Input.txt: 1 4 6 10 0 0 0 7 0 8 0 0 2 5 0 0 3 9 0 0 0
BinaryTree in Inorder traversal:
1 2 3 4 5 6 7 8 9 10
BinaryTree Preorder traversal:
1 4 2 3 6 5 10 7 8 9
Paths with sum 5:
[1, 4]
[2, 3]
[5]

```

Output_txt(example)

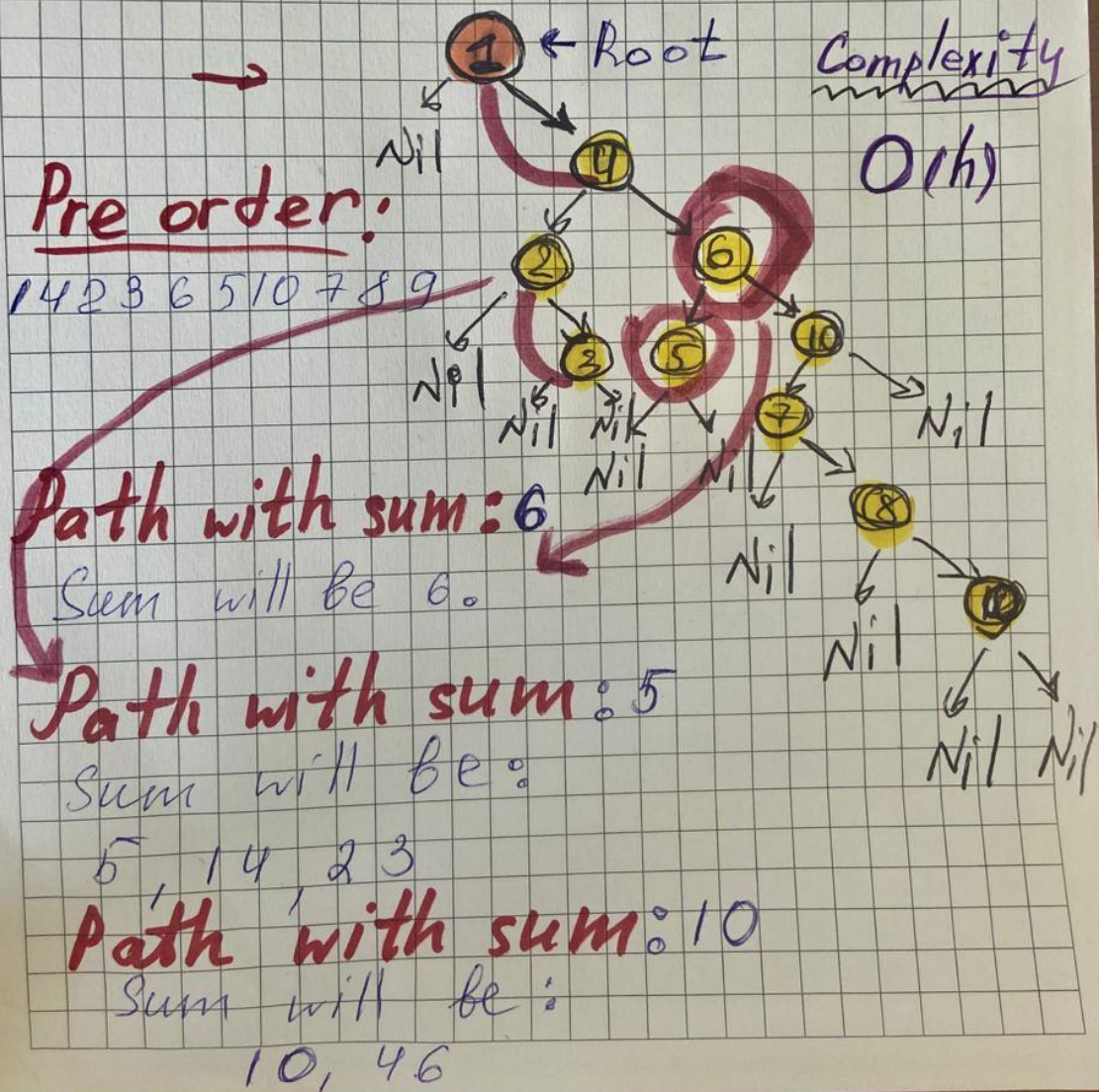


output.txt

1 4
2 3
5

Binary tree search

① Input txt = 1 4 6 10 00070800
250039000, (0-Null)



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

Output_100b.txt(example)

```
output.txt
6 7 26 39
78
```

Input_10b.txt()

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

Solution

```
[mac@MacBook-Pro-mac src % java Main input.txt 7
Input.txt: 1 2 5 0 0 7 0 9 0 0 3 4 6 0 0 0 8 10 0 0 0
BinaryTree in Inorder traversal:
1 2 3 4 5 6 7 8 9 10
BinaryTree Preorder traversal:
1 2 5 3 4 7 6 9 8 10
Paths with sum 7:
[2, 5]
[3, 4]
[7]
```

Output_10b.txt()

```
output.txt
2 5
3 4
7
```


Manual solution

infopulse

② Binary tree search

input 106.txt:

1 2 5 0 0 4 0 9 0 0 3 4 6 0 0 0 8 1 0 0 0 0

Expectation - calculate sum 7

Complexity $O(h)$

output 106 (7).txt

2 5
7
3 4

```
graph TD; 1((1)) --> 2((2)); 2 --> 5((5)); 5 --> 7((7)); 7 --> 9((9)); 9 --> 3((3)); 3 --> 4((4)); 4 --> 6((6)); 6 --> 8((8)); 8 --> 10((10));
```

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.