**Міністерство освіти і науки України**

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

**Факультет інформатики та обчислювальної техніки**

**Кафедра автоматизованих систем обробки інформації**

**і управління**

**Звіт**

з лабораторної роботи № 5 з дисципліни

«Алгоритми та структури даних 2. Структури даних»

**«Деревовидні структури даних»**

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2021

Зміст

1. Мета лабораторної роботи 3
2. ЗаВдання 4
3. Виконання 9
   1. Псевдокод алгоритмів 9
   2. Програмна реалізація 10
      1. Вихідний код 10
      2. Приклади роботи 15

Висновок 17

Критерії оцінювання 18

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи формалізації та імплементації алгоритмів побудови та обробки базових деревовидних структур даних.

# ЗаВдання

Розробити алгоритм розв’язання задачі відповідно до варіанту. Виконати програмну реалізацію задачі. Не використовувати вбудовані деревовидні структури даних (контейнери). Зробити висновок по лабораторній роботі.

**Варіанти завдань.**

1. Задано фрагмент програми мовою С++. Надрукувати в алфавітному порядку всі ідентифікатори цієї програми, вказавши для кожного з них число входжень у текст програми. Для збереження ідентифікаторів використати структуру типу дерева, елементами якого є ідентифікатор і число його входжень у текст.
2. Побудувати дерево, елементами якого є дійсні числа. Обчислити середнє арифметичне усіх його елементів.
3. Побудувати дерево, що відображає формулу (a\*(b+с))/d, де коренем дерева та його підкоренями є операції "\*, +, -, /", а листками - змінні a, b, c, d. Надрукувати відповідне дерево .
4. Побудувати дерево, елементами якого є числа. Надрукувати дерево. Визначити кількість від'ємних та додатніх елементів дерева.
5. Побудувати двійкове дерево, елементами якого є символи. Визначити, чи знаходиться у цьому дереві елемент, значення якого вводиться з клавіатури. Якщо елемент знайдений, то підрахувати число його входжень.
6. Побудувати двійкове дерево пошуку з літер заданого рядка. Видалити з дерева літери, що зустрічаються більше одного разу. Вивести елементи дерева, що залишилися, при його постфіксному обході.
7. Заданий текст. Підрахувати кількість повторень кожного слова. Побудувати дерево із слів тексту. Слова, що зустрічаються найчастіше розмістити на верхньому рівні, на інших рівнях дерева розмістити слова з меншою кількістю повторень.
8. Побудувати бінарне дерево, елементами якого є дійсні числа. Знайти значення найбільшого елемента цього дерева та надрукувати його.
9. Заданий рядок символів латинського алфавіту. Побудувати дерево, в якому значеннями вершин є символи, що розміщуються на рівнях відповідно до кількості їх повторень у рядку.
10. Побудувати дерево, елементами якого є цілі числа. Визначити кількість вершин на n-му його рівні та кількість рівнів.
11. Побудувати дерево, що відображає формулу ((a+b)\*c-d), де коренем дерева та його підкоренями є операції " +, -, \*, /", а листками є змінні a, b, c, d . Вивести значення дерева-формули. Надрукувати відповідні піддерева у1=а+b, у2=у1\*с, y3=y2-d.
12. Побудувати дерево наступного виду:

, де n -додатне ціле число

n

/ \

n-1 n-1

/ \ / \

..........

2 2

/ \ / \

1 1 1 1

1. Побудувати бінарне дерево для зберігання даних виду: деталь, її кількість, постачальник. Забезпечити виконання операцій додавання нового елемента у дерево в діалоговому режимі та визначення постачальника найбільшої кількості деталей.
2. Побудувати дерево, елементами якого є символи. Визначити максимальну глибину дерева (число гілок на найбільшому з маршрутів від кореня дерева до листків).
3. Побудувати бінарне дерево, елементами якого є цілі числа. Підрахувати кількість вершин на n-му рівні цього дерева (нульовий рівень - корінь цього дерева) та надрукувати ці елементи.
4. Побудувати дерево, що відображає формулу ((a+b)/c)\*d, де коренем дерева та його підкоренями є операції, а листками - змінні. Ввести значення змінних та визначити значення дерева-формули. Надрукувати відповідні піддерева, наприклад: у1=а+b, y2:= y1/c, y3=y2\*d.
5. Відповідно до виразу, що вводиться з клавіатури та може містити операції +,-,\*,/, побудувати дерево-формулу та обчислити значення цієї формули.
6. Побудувати бінарне дерево, елементами якого є слова. Знайти у ньому значення слова, введеного з клавіатури, визначивши номер відповідного рівня.
7. Побудувати дерево, елементами якого є символи. Знайти довжину шляху (число гілок) від кореня до значення символа, введеного з клавіатури. Різновид дерева вибрати самостійно.
8. Побудувати дерево наступного типу:

, де n -додатне ціле число

n

/ \

2 2

/ \ / \

..........

n-1 n-1

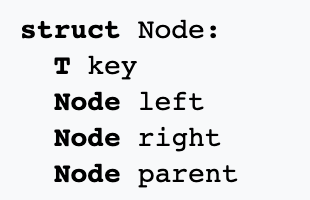
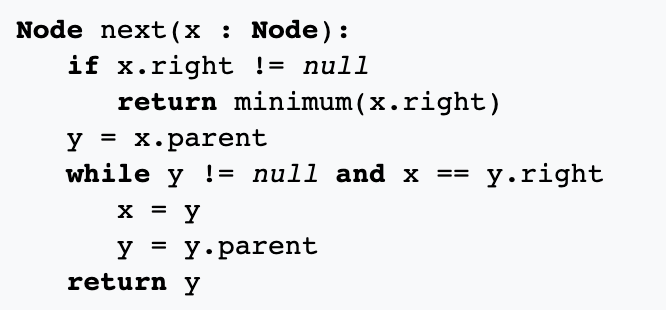
/ \ / \

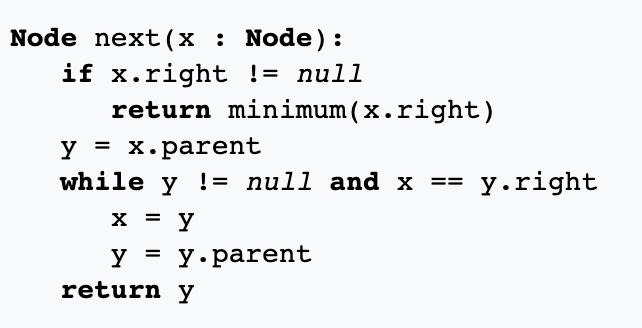
n n n n

1. Побудувати два бінарних дерева, елементами якого є цілі числа. Об'єднати їх, уникаючи дублювання елементів в сумарному дереві.
2. Побудувати дерево, елементами якого є дійсні числа. Поміняти місцями найбільше та найменше значення дерева.
3. Побудувати бінарне дерево для зберігання даних виду: найменування товару, його кількість, вартість одиниці. Забезпечити виконання операцій додавання нового елемента у дерево в діалоговому режимі та підрахунку загальної вартості вказаного товару.
4. Побудувати двійкове дерево пошуку, в вершинах якого знаходяться слова введені з клавіатури. Визначити кількість вершин дерева, що містять слова, які починаються на зазначену букву.
5. Побудувати дерево, елементами якого є цілі числа. Визначити кількість вузлових вершин (не листків) даного дерева та надрукувати їх координати (номер рівня та номер гілки).
6. Написати програму, що будує дерево-формулу та перетворює в ньому всі піддерева, що відповідають формулам ((f1±f2)\*f3), на піддерева виду ((f1\*f3)±(f2\*f3)).
7. Побудувати дерево, елементами якого є символи. Визначити і вивести на друк усі термінальні вершини (листя) цього дерева.
8. Побудувати і вивести на екран бінарне дерево наступного виразу: 9 +  8 \* (7 + (6 \* (5 + 4) – (3 – 2)) +1)). Реалізувати постфіксний, інфіксний та префіксний обходи дерева і вивести відповідні вирази на екран.
9. Побудувати двійкове дерево пошуку, в вершинах якого знаходяться цілі числа. Визначити максимальну висоту дерева, тобто число ребер в найдовшому шляху від кореня до листків.
10. Побудувати двійкове дерево пошуку, в вершинах якого знаходяться цілі числа. Визначити висоту дерева, тобто число ребер в шляху від кореня до заданої вершини.
11. Побудувати дерево, елементами якого є цілі числа. Видалити з дерева усі листки, надрукувати початкове та модифіковане дерево.
12. Побудувати двійкове дерево пошуку, в вершинах якого знаходяться цілі числа. Визначити чи є дерево збалансованим.
13. Побудувати бінарне дерево, елементами якого є цілі числа. Вивести усі вузли, які мають лише правого потомка і усі вузли, які мають лише лівого потомка.
14. Побудувати дерево, елементами якого є цілі числа. Видалити з дерева усі вузли, які є батьками листків, надрукувати початкове та модифіковане дерево.
15. Реалізувати структуру даних бінарна піраміда на основі бінарного дерева, передбачити можливість додавання та видалення елементів.

# Виконання

## Псевдокод алгоритмів





## Програмна реалізація

### Вихідний код

package DocumentWordBinaryTree;

import java.io.IOException;

import java.nio.charset.Charset;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.nio.file.Files;

import java.nio.file.LinkOption;

import java.util.Collections;

import java.util.LinkedHashMap;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import java.util.logging.Level;

import java.util.logging.Logger;

import java.util.stream.Collectors;

import static java.util.stream.Collectors.toMap;

import java.util.stream.Stream;

public class Main {

public static void main(String[] args) {

if (args.length < 1) {

System.out.println("Please provide a valid file path as an argument");

System.exit(-1);

}

Path path = Paths.get(args[0]);

// check for txt file as an input in the argument

if (!isTextFile(path)) {

System.out.println("Only txt files are supported, please provide a link for a txt file");

System.exit(-1);

}

if (!Files.exists(path, LinkOption.NOFOLLOW\_LINKS) && !Files.isReadable(path) && !Files.isRegularFile(path, LinkOption.NOFOLLOW\_LINKS)) {

System.out.println("ERROR: the file does not exist \n "

+ "OR the file is not accessible \n"

+ "OR path is invalid");

System.exit(-1);

}

wordFreqCount(path);

}

private static boolean isTextFile(Path path) {

String pathString = path.toString();

String txtString = pathString.substring(pathString.length() - 4);

return txtString.equals(".txt");

}

private static void wordFreqCount(Path path) {

try {

String content = new String(Files.readAllBytes(path), Charset.forName("UTF-8"));

Stream<String> stream = Stream.of(content.toLowerCase().split("\\s+")).parallel();

Map<String, Long> wordFreq = stream.collect(Collectors.groupingBy(String::toString, Collectors.counting()));

if (wordFreq.isEmpty()) {

System.out.println("Unfortunately your text file is empty!!\nPlease provide a text file with some content");

System.exit(-1);

}

Map<String, Long> sortedWordFreq = wordFreq.entrySet().stream()

.sorted(Entry.comparingByValue())

.collect(toMap(Entry::getKey, Entry::getValue,

(e1, e2) -> e1, LinkedHashMap::new));

createBinaryTree(sortedWordFreq);

} catch (IOException ex) {

Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);

}

}

private static void createBinaryTree(Map<String, Long> map) {

LinkedList<Node> nodeList = new LinkedList<>();

for (Map.Entry<String, Long> entry : map.entrySet()) {

Node node = new Node(entry.getKey(), entry.getValue());

nodeList.add(node);

}

int verticalIteration = getVerticalIteration(nodeList.size());

List<Node> replacedNodeList = new LinkedList<>();

for (int i = 0; i < verticalIteration; i++) {

for (int j = 0; j < nodeList.size() - 1; j += 2) {

Node nodeLeft = nodeList.get(j);

Node nodeRight = nodeList.get(j + 1);

Node newNode = new Node("Lvl:" + (i + 2), nodeLeft.getValue() + nodeRight.getValue(), nodeLeft, nodeRight);

nodeLeft.setParent(newNode);

nodeRight.setParent(newNode);

replacedNodeList.add(newNode);

}

if (nodeList.size() % 2 == 1) {

replacedNodeList.add(nodeList.get(nodeList.size() - 1));

}

Collections.sort(replacedNodeList, (o1, o2) -> o1.getValue().compareTo(o2.getValue()));

nodeList.clear();

nodeList.addAll(replacedNodeList);

replacedNodeList.clear();

}

Node binaryTree = nodeList.get(0);

binaryTree.print();

}

private static int getVerticalIteration(int arraysize) {

boolean isLastIteration = false;

int currentArraySize = arraysize;

int iterationCount = 0;

while (isLastIteration == false) {

iterationCount++;

currentArraySize = (currentArraySize % 2 == 1) ? (currentArraySize /= 2) + 1 : (currentArraySize /= 2);

isLastIteration = (currentArraySize == 1);

}

return iterationCount;

}

}

package DocumentWordBinaryTree;

import java.util.LinkedList;

public class Node {

private String key;

private Long value;

private Node parent;

private Node left;

private Node right;

public Node(String key, Long value) {

this.key = key;

this.value = value;

this.parent = null;

this.left = null;

this.right = null;

}

public Node(String key, Long value, Node left, Node right) {

this.key = key;

this.value = value;

this.left = left;

this.right = right;

this.parent = null;

}

public String getKey() {

return key;

}

public Long getValue() {

return value;

}

public Node getParent() {

return parent;

}

public Node getLeft() {

return left;

}

public Node getRight() {

return right;

}

public void setKey(String key) {

this.key = key;

}

public void setValue(Long value) {

this.value = value;

}

public void setParent(Node parent) {

this.parent = parent;

}

public void setLeft(Node left) {

this.left = left;

}

public void setRight(Node right) {

this.right = right;

}

public LinkedList<Node> getChilds() {

LinkedList<Node> nodeList = new LinkedList<>();

if (this.left != null) {

nodeList.add(this.left);

}

if (this.right != null) {

nodeList.add(this.right);

}

return nodeList;

}

public void print() {

print("",this, false);

}

private void print(String prefix, Node node, boolean isLeft) {

if (node != null) {

System.out.println (prefix + (isLeft ? "|-- " : "\\-- ") + node.getKey() + "=" + node.getValue());

print(prefix + (isLeft ? "| " : " "), node.right, true);

print(prefix + (isLeft ? "| " : " "), node.left, false);

}

}

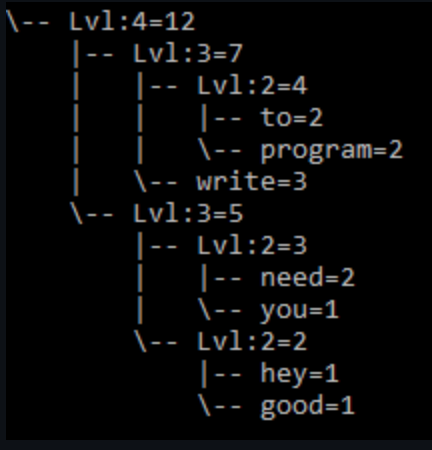
}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1 –

**Example output**

text = Hey you need need to to write write write good program program

Висновок

При виконанні даної лабораторної роботи ми вивчили роботу бинарних древ

**Бинарное дерево поиска** (англ. *binary search tree, BST*) — структура данных для работы с [упорядоченными множествами](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A3%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D1%87%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE).

Бинарное дерево поиска обладает следующим свойством: если

x — узел бинарного дерева с ключом k, то все узлы в левом поддереве должны иметь ключи, меньшие k а в правом поддереве большие k

Критерії оцінювання

За умови здачі лабораторної роботи до 27.04.2020 включно максимальний бал дорівнює – 5. Після 27.04.2020 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 10%;
* програмна реалізація алгоритму – 80%;
* висновок – 10%.