МІНІСТЕРСТВО ОБОРОНИ УКРАЇНИ

ВІЙСЬКОВИЙ ІНСТИТУТ ТЕЛЕКОМУНІКАЦІЙ ТА ІНФОРМАТИЗАЦІЇ

КАФЕДРА КІБЕРЗБЕЗПЕКИ

**ЗВІТ**

про виконання практичної роботи:

з дисципліни «Прикладна криптологія»

**РОЗРАХУНОК ЧАСТОТ СИМВОЛІВ ТА БІГРАМ ВІДКРИТИХ ТЕКСТІВ**

Дата 04.04.2024

**Виконав:** курсант 314 навчальної групи

старший солдат\_\_\_\_\_\_\_\_\_\_\_КОКУРА І.В.

**Перевiрив:** старший викладач кафедри 33

капітан\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ПРИЙМА О.О.

Код програми:  
import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.GridPane;

import javafx.stage.FileChooser;

import javafx.stage.Stage;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

public class TextFilterGUI extends Application {

private TextArea inputTextArea;

private TextArea outputTextArea;

private ComboBox<String> filterModeComboBox;

private String alphabet = "абвгдеєжзиіїйклмнопрстуфхцчшщьюя";

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("Text Filter");

GridPane grid = new GridPane();

grid.setPadding(new Insets(10, 10, 10, 10));

grid.setVgap(5);

grid.setHgap(5);

Label inputLabel = new Label("Input Text:");

GridPane.setConstraints(inputLabel, 0, 0);

inputTextArea = new TextArea();

inputTextArea.setPrefRowCount(5);

inputTextArea.setWrapText(true);

GridPane.setConstraints(inputTextArea, 0, 1);

Label outputLabel = new Label("Filtered Text:");

GridPane.setConstraints(outputLabel, 0, 2);

outputTextArea = new TextArea();

outputTextArea.setPrefRowCount(5);

outputTextArea.setEditable(false);

outputTextArea.setWrapText(true);

GridPane.setConstraints(outputTextArea, 0, 3);

Button chooseFileButton = new Button("Choose File");

chooseFileButton.setOnAction(e -> chooseFile(primaryStage));

GridPane.setConstraints(chooseFileButton, 1, 0);

filterModeComboBox = new ComboBox<>();

filterModeComboBox.getItems().addAll("Filter with Normal Alphabet", "Filter with Alphabet and Space");

filterModeComboBox.setValue("Filter with Normal Alphabet");

GridPane.setConstraints(filterModeComboBox, 1, 1);

Button filterButton = new Button("Filter Text");

filterButton.setOnAction(e -> filterText());

GridPane.setConstraints(filterButton, 1, 2);

grid.getChildren().addAll(inputLabel, inputTextArea, outputLabel, outputTextArea, chooseFileButton, filterModeComboBox, filterButton);

Scene scene = new Scene(grid, 500, 300);

primaryStage.setScene(scene);

primaryStage.show();

}

private void chooseFile(Stage primaryStage) {

FileChooser fileChooser = new FileChooser();

FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter("Text Files (\*.txt)", "\*.txt");

fileChooser.getExtensionFilters().add(extFilter);

fileChooser.setTitle("Open Text File");

File selectedFile = fileChooser.showOpenDialog(primaryStage);

if (selectedFile != null) {

try {

BufferedReader reader = new BufferedReader(new FileReader(selectedFile));

StringBuilder text = new StringBuilder();

String line;

while ((line = reader.readLine()) != null) {

text.append(line).append("\n");

}

reader.close();

inputTextArea.setText(text.toString());

} catch (IOException ex) {

showErrorAlert("Error reading file", "An error occurred while reading the file.");

}

}

}

private void filterText() {

try {

String inputText = inputTextArea.getText();

String selectedMode = filterModeComboBox.getValue();

String filteredText;

if (selectedMode.equals("Filter with Normal Alphabet")) {

filteredText = filterNormalAlphabet(inputText, alphabet);

} else {

filteredText = filterWithSpaceAlphabet(inputText, alphabet);

}

outputTextArea.setText(filteredText);

// Calculate frequencies

calculateCharacterFrequenciesWithSpace(filteredText);

calculateCharacterFrequenciesWithoutSpace(filteredText);

calculateBigramFrequenciesWithSpaceOverlap(filteredText);

calculateBigramFrequenciesWithSpaceNoOverlap(filteredText);

calculateBigramFrequenciesNoSpaceOverlap(filteredText);

calculateBigramFrequenciesNoSpaceNoOverlap(filteredText);

// Calculate text coincidence index

double textCoincidenceIndexWithSpace = calculateTextCoincidenceIndex(filteredText);

double textCoincidenceIndexWithoutSpace = calculateTextCoincidenceIndex(filteredText.replaceAll("\\s+", ""));

outputTextArea.appendText("\n\nText Coincidence Index (with spaces): " + String.format("%.4f", textCoincidenceIndexWithSpace));

outputTextArea.appendText("\nText Coincidence Index (without spaces): " + String.format("%.4f", textCoincidenceIndexWithoutSpace));

} catch (Exception ex) {

showErrorAlert("Error filtering text", "An error occurred while filtering the text.");

}

}

private void calculateCharacterFrequenciesWithSpace(String text) {

Map<Character, Integer> frequencies = new HashMap<>();

int totalChars = 0;

for (char c : text.toCharArray()) {

if (Character.isLetter(c) || c == ' ') {

c = Character.toLowerCase(c);

frequencies.put(c, frequencies.getOrDefault(c, 0) + 1);

totalChars++;

}

}

StringBuilder result = new StringBuilder("Character Frequencies (with spaces):\n");

for (char c : frequencies.keySet()) {

double frequency = (double) frequencies.get(c) / totalChars;

result.append(c).append(": ").append(String.format("%.2f", frequency \* 100)).append("%\n");

}

outputTextArea.appendText("\n\n" + result.toString());

}

private void calculateCharacterFrequenciesWithoutSpace(String text) {

Map<Character, Integer> frequencies = new HashMap<>();

int totalChars = 0;

for (char c : text.toCharArray()) {

if (Character.isLetter(c)) {

c = Character.toLowerCase(c);

frequencies.put(c, frequencies.getOrDefault(c, 0) + 1);

totalChars++;

}

}

StringBuilder result = new StringBuilder("Character Frequencies (without spaces):\n");

for (char c : frequencies.keySet()) {

double frequency = (double) frequencies.get(c) / totalChars;

result.append(c).append(": ").append(String.format("%.2f", frequency \* 100)).append("%\n");

}

outputTextArea.appendText("\n\n" + result.toString());

}

private void calculateBigramFrequenciesWithSpaceOverlap(String text) {

Map<String, Integer> frequencies = new HashMap<>();

int totalBigrams = 0;

for (int i = 0; i < text.length() - 1; i++) {

char first = Character.toLowerCase(text.charAt(i));

char second = Character.toLowerCase(text.charAt(i + 1));

if (Character.isLetter(first) && Character.isLetter(second) || first == ' ' || second == ' ') {

String bigram = String.valueOf(first) + second;

frequencies.put(bigram, frequencies.getOrDefault(bigram, 0) + 1);

totalBigrams++;

}

}

StringBuilder result = new StringBuilder("Bigram Frequencies (with spaces, overlapping):\n");

for (String bigram : frequencies.keySet()) {

double frequency = (double) frequencies.get(bigram) / totalBigrams;

result.append(bigram).append(": ").append(String.format("%.2f", frequency \* 100)).append("%\n");

}

outputTextArea.appendText("\n\n" + result.toString());

}

private void calculateBigramFrequenciesWithSpaceNoOverlap(String text) {

Map<String, Integer> frequencies = new HashMap<>();

int totalBigrams = 0;

for (int i = 0; i < text.length() - 1; i += 2) {

char first = Character.toLowerCase(text.charAt(i));

char second = Character.toLowerCase(text.charAt(i + 1));

if (Character.isLetter(first) && Character.isLetter(second) || first == ' ' || second == ' ') {

String bigram = String.valueOf(first) + second;

frequencies.put(bigram, frequencies.getOrDefault(bigram, 0) + 1);

totalBigrams++;

}

}

StringBuilder result = new StringBuilder("Bigram Frequencies (with spaces, non-overlapping):\n");

for (String bigram : frequencies.keySet()) {

double frequency = (double) frequencies.get(bigram) / totalBigrams;

result.append(bigram).append(": ").append(String.format("%.2f", frequency \* 100)).append("%\n");

}

outputTextArea.appendText("\n\n" + result.toString());

}

private void calculateBigramFrequenciesNoSpaceOverlap(String text) {

Map<String, Integer> frequencies = new HashMap<>();

int totalBigrams = 0;

for (int i = 0; i < text.length() - 1; i++) {

char first = Character.toLowerCase(text.charAt(i));

char second = Character.toLowerCase(text.charAt(i + 1));

if (Character.isLetter(first) && Character.isLetter(second)) {

String bigram = String.valueOf(first) + second;

frequencies.put(bigram, frequencies.getOrDefault(bigram, 0) + 1);

totalBigrams++;

}

}

StringBuilder result = new StringBuilder("Bigram Frequencies (without spaces, overlapping):\n");

for (String bigram : frequencies.keySet()) {

double frequency = (double) frequencies.get(bigram) / totalBigrams;

result.append(bigram).append(": ").append(String.format("%.2f", frequency \* 100)).append("%\n");

}

outputTextArea.appendText("\n\n" + result.toString());

}

private void calculateBigramFrequenciesNoSpaceNoOverlap(String text) {

Map<String, Integer> frequencies = new HashMap<>();

int totalBigrams = 0;

for (int i = 0; i < text.length() - 1; i += 2) {

char first = Character.toLowerCase(text.charAt(i));

char second = Character.toLowerCase(text.charAt(i + 1));

if (Character.isLetter(first) && Character.isLetter(second)) {

String bigram = String.valueOf(first) + second;

frequencies.put(bigram, frequencies.getOrDefault(bigram, 0) + 1);

totalBigrams++;

}

}

StringBuilder result = new StringBuilder("Bigram Frequencies (without spaces, non-overlapping):\n");

for (String bigram : frequencies.keySet()) {

double frequency = (double) frequencies.get(bigram) / totalBigrams;

result.append(bigram).append(": ").append(String.format("%.2f", frequency \* 100)).append("%\n");

}

outputTextArea.appendText("\n\n" + result.toString());

}

private double calculateTextCoincidenceIndex(String text) {

Map<Character, Integer> frequencies = new HashMap<>();

int totalChars = 0;

for (char c : text.toCharArray()) {

if (Character.isLetter(c)) {

c = Character.toLowerCase(c);

frequencies.put(c, frequencies.getOrDefault(c, 0) + 1);

totalChars++;

}

}

double index = 0;

for (int freq : frequencies.values()) {

index += (double) freq \* (freq - 1);

}

index /= (double) totalChars \* (totalChars - 1);

return index;

}

private String filterNormalAlphabet(String text, String alphabet) {

StringBuilder result = new StringBuilder();

for (char c : text.toCharArray()) {

if (alphabet.indexOf(Character.toLowerCase(c)) != -1) {

result.append(Character.toLowerCase(c));

}

}

return result.toString();

}

private String filterWithSpaceAlphabet(String text, String alphabet) {

StringBuilder result = new StringBuilder();

boolean spaceFound = false;

for (char c : text.toCharArray()) {

if (Character.isWhitespace(c)) {

if (!spaceFound) {

result.append(' ');

spaceFound = true;

}

} else if (alphabet.indexOf(Character.toLowerCase(c)) != -1) {

result.append(Character.toLowerCase(c));

spaceFound = false;

}

}

return result.toString().trim();

}

private void showErrorAlert(String title, String message) {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();

}

public static void main(String[] args) {

launch(args);

}

}

Висновок:

Під час цієї роботи було створено програму на мові Java з графічним інтерфейсом, яка здійснює фільтрацію тексту за заданим алфавітом, обчислює частоти символів та біграм, а також індекс відповідності тексту.

1. **Фільтрація тексту:** Програма дозволяє користувачеві вибирати текстовий файл з комп'ютера та застосовувати фільтрацію тексту в режимі звичайного алфавіту або з алфавітом і пробілом. Результати виводяться у відповідних текстових полях.
2. **Обчислення частот символів та біграм:** Програма розраховує частоти символів і біграм у відфільтрованому тексті, враховуючи різні варіанти обробки пробілів та перетину символів.
3. **Обчислення індексу відповідності тексту:** Функція була додана для обчислення індексу відповідності тексту, що дозволяє оцінити ступінь однорідності символів у тексті. Ця метрика допомагає визначити, наскільки текст є схожим на мову, для якої було створено алфавіт.