COMP8547 Advanced Computing Concepts – Fall 2023
Assignment 2

Variant 4 - *Plagiarism Detection using the Edit Distance Algorithm*

Source Code with Explanation:

```
package PlagiarismDetector;  // Here, I'm declaring the package named "PlagiarismDetector."

import java.io.BufferedReader;  // Here, I'm importing the  necessary Java classes and libraries.
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Scanner;
import java.util.Set;

public class PlagiarismDetector { // Here is the class for PlagiarismDetector
   public static void main(String[] args) { // Main function
      Scanner scanner = new Scanner(System.in);  // I'm creating a Scanner object for user input.

      System.out.print("Please Enter the number of Text Files that you would like to check Plagiarism: ");
      int nmbr_of_filess = scanner.nextInt();  // Here, I'm reading the number of text files to compare.
      scanner.nextLine();  // To enter a newline character.

      System.out.print("Please Enter the path in which you would like to check the text files you need to check: ");
      String dirr_paath = scanner.nextLine();  // Inputing this to avoid file not found exception as it will read the directory path where text files are located.

      List<String> filee_cntnts = new ArrayList<>();  // I'm initializing a list to store file contents.
      File directory = new File(dirr_paath);  // I'm creating a File object for the specified directory.
      File[] filees_in_dirr = directory.listFiles((dir, name) -> name.toLowerCase().endsWith(".txt"));
// Here, I'm listing all .txt files in the directory to check the txt files and not other text file types like docx, etc.

      if (filees_in_dirr == null || filees_in_dirr.length == 0) {
         System.out.println("Sorry mate! There aren't any files in the directory you specified. Try again!");
```

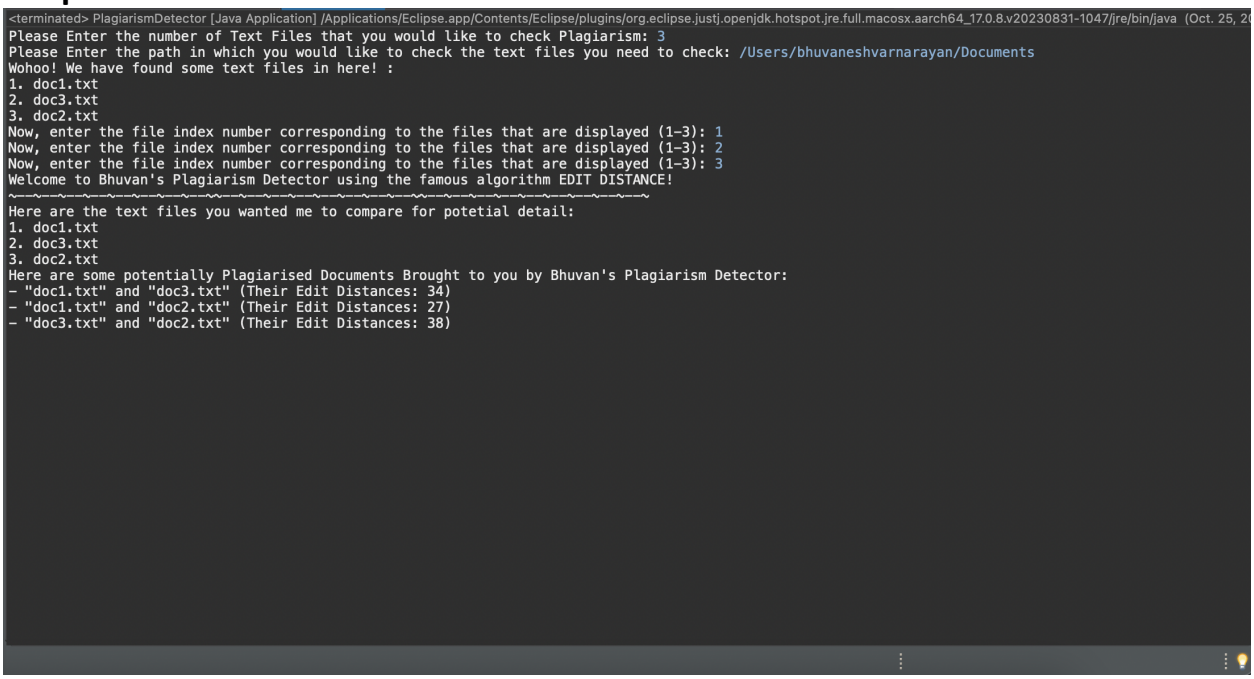
```
            scanner.close(); // In this block, I'm trying to explain that the directory does not contain any txt files to compare
            return;
        }

        System.out.println("Wohoo! We have found some text files in here! :");
        for (int indxr = 0; indxr < filees_in_dirr.length; indxr++) {
            System.out.println((indxr + 1) + ". " + filees_in_dirr[indxr].getName());  // Displaying the text files that have been found.
        }

        for (int indxr = 0; indxr < nmbr_of_filess; indxr++) {
            System.out.print("Now, enter the file index number corresponding to the files that are displayed (1-" + filees_in_dirr.length + "): ");
            int fileee_nmbr = scanner.nextInt();  // I'm giving the user the ability to enter the file index number displayed on the screen rather than entering the file name to reduce ambiguity.
            scanner.nextLine();  // To enter a newline character.

            if (fileee_nmbr < 1 || fileee_nmbr > filees_in_dirr.length) {
                System.out.println("Check the screen and enter a valid number!");
                indxr--; // Decrementing the i variable to retry input if the number is invalid.
            } else {
                String filee_paath = filees_in_dirr[fileee_nmbr - 1].getPath();  // To get the path of the selected file.
                String content1 = read_txt_filee(filee_paath);  // To read the content of the file.
                if (content1 != null) {
                    filee_cntnts.add(content1);  // Adds all the file content to the list.
                } else {
                    System.out.println("We regret to inform you that the file that you wanted me to read failed miserably: " + filee_paath); // Sends this message if fails.
                }
            }
        }

        // Given the ability to adjust to a higher threshold_valuees for similarity scores
        int threshold_valuees = 60; // Users can adjust this value as needed

        // Here, I'm setting a stricter minimum sequence length for reporting
        int minimum_Sequence_lngth = 50; // Users can adjust this value as needed

        // Users can define some of the common stop words
        Set<String> stopWords = new HashSet<>();
        stopWords.add("the");
        stopWords.add("and");
```

```
        stopWords.add("he");
        stopWords.add("she");
        stopWords.add("are");
        stopWords.add("was");
        stopWords.add("is");
        stopWords.add("that");
        // We can add more stop words as needed in here.

        // Compare the text documents and identify potential plagiarism
        System.out.println("Welcome to Bhuvan's Plagiarism Detector using the famous algorithm EDIT DISTANCE!"); // Intro of the app
        System.out.println("~--~--~--~--~--~--~--~--~~--~--~--~--~--~--~--~--~~--~--~--~--~--~--~--~--~--~"); // Decorations
        System.out.println("Here are the text files you wanted me to compare for potetial detail:");
        for (int indxr = 0; indxr < nmbr_of_filess; indxr++) {
            System.out.println((indxr + 1) + ". " + filees_in_dirr[indxr].getName());  // Here it displays the selected text files.
        }

        System.out.println("Here are some potentially Plagiarised Documents Brought to you by Bhuvan's Plagiarism Detector:");
        for (int indxr = 0; indxr < filee_cntnts.size(); indxr++) {
            for (int j_indxr = indxr + 1; j_indxr < filee_cntnts.size(); j_indxr++) {
                int editDistance = calculateEditDistance(filee_cntnts.get(indxr), filee_cntnts.get(j_indxr), stopWords, minimum_Sequence_lngth);  // This calculates the edit distance.
                if (editDistance <= threshold_valuees) {
                    System.out.println("- \"" + filees_in_dirr[indxr].getName() + "\" and \"" + filees_in_dirr[j_indxr].getName() + "\" (Their Edit Distances: " + editDistance + ")");
                }
            }
        }

        scanner.close();
    }

    private static String read_txt_filee(String filee_paath) {
        StringBuilder content = new StringBuilder();  // Creating a StringBuilder to store the content of the text file.
        try (BufferedReader reader = new BufferedReader(new FileReader(filee_paath))) {
            String line_22;
            while ((line_22 = reader.readLine()) != null) {
                content.append(line_22).append("\n");  // Read each line of the text file and append it to the content.
```

```
        }
    } catch (IOException e) {
        return null;  // This will return null when a  file is not found or any errors.
    }
    return content.toString();  // This returns the content of the text file as a single string.
}

  private static int calculateEditDistance(String text1, String text2, Set<String> stopWords, int minimum_Sequence_lngth) {
    String[] wrdss_1 = text1.split("\\s+");  // Split the first text into an array of words.
    String[] wrdss_2 = text2.split("\\s+");  // Split the second text into an array of words.

    int m = wrdss_1.length;  // Get the length of the first text.
    int n = wrdss_2.length;  // Get the length of the second text.

    int[][] dp = new int[m + 1][n + 1];  // Create a 2D array to store edit distance values.

    for (int indxr = 0; indxr <= m; indxr++) {
        for (int j_indxr = 0; j_indxr <= n; j_indxr++) {
            if (indxr == 0) {
                dp[indxr][j_indxr] = j_indxr;  // Initialize the first row with incremental values.
            } else if (j_indxr == 0) {
                dp[indxr][j_indxr] = indxr;  // Initialize the first column with incremental values.
            } else {
                int cost = wrdss_1[indxr - 1].equals(wrdss_2[j_indxr - 1]) ? 0 : 1;  // Calculate the cost of replacing a word.

                if (stopWords.contains(wrdss_1[indxr - 1]) || stopWords.contains(wrdss_2[j_indxr - 1])) {
                    // If a word is a stop word, consider a different cost.
                    dp[indxr][j_indxr] = Math.min(dp[indxr - 1][j_indxr - 1] + cost, Math.min(dp[indxr - 1][j_indxr] + 1, dp[indxr][j_indxr - 1] + 1));
                } else {
                    // If not a stop word, use the standard cost.
                    dp[indxr][j_indxr] = Math.min(dp[indxr - 1][j_indxr - 1] + cost, Math.min(dp[indxr - 1][j_indxr] + 1, dp[indxr][j_indxr - 1] + 1));
                }
            }
        }
    }

    int editDistance = dp[m][n];  // The final edit distance value.

    // Adjust the edit distance based on the minimum sequence length
```

```
        int maximum_lenngth = Math.max(m, n);  // Find the maximum length of the two texts.
        if (maximum_lenngth < minimum_Sequence_lngth) {
            return editDistance;  // If the maximum length is below the threshold_valuees, return the unadjusted edit distance.
        } else {
            return (editDistance * minimum_Sequence_lngth) / maximum_lenngth;  // Adjust the edit distance based on the minimum sequence length and return it.
        }
    }
}
```

Output:



Explanation:

- The program first prompts the user to enter the number of text files they want to check for plagiarism.
- After entering the number of files, the code prompts the user to enter the directory where these files are located.
- After entering the directory, if it's unsuccessful, it will prompt a message saying there aren't any files in the directory.
- If successful, it finds text files in the specified directory, in this case three files, and displays them. These files are named "doc1.txt," "doc3.txt," and "doc2.txt."
- The user is prompted to select files by entering their corresponding index numbers. In this case, the user selects all three files.

- After input, the program welcomes the user to “Bhuvan's Plagiarism Detector” and displays a decorative separator.
- It then lists the text files the user wants to compare again: "doc1.txt," "doc3.txt," and "doc2.txt."
- Finally, the program performs plagiarism detection using the Edit Distance algorithm and displays the potentially plagiarized documents along with their calculated edit distances. Here's a breakdown of the detected plagiarized documents:
    - "doc1.txt" and "doc3.txt" have an edit distance of 34.
    - "doc1.txt" and "doc2.txt" have an edit distance of 27.
    - "doc3.txt" and "doc2.txt" have an edit distance of 38.

To put it simply, the program compared the text files and found that "doc1.txt" and "doc3.txt" have somewhat similar content, with an edit distance of 34. Similarly, there were similarities between "doc1.txt" and "doc2.txt" with an edit distance of 27, and between "doc3.txt" and "doc2.txt" with an edit distance of 38. Lower edit distances indicate a higher level of similarity between the documents, which could suggest potential plagiarism.