DOKUZ EYLÜL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

Natural Language Processing

Assignment – 2

by

2019510017 Ali Şiyar ARSLAN

Advisor

Assoc. Prof. Dr. Özlem AKTAŞ

December, 2023

İZMİR

Considering that the project does not open, I added a file with .jar extension under the dist folder.

1. Definition of Algorithm

```
private void list_correct_wordsActionPerformed(java.awt.event.ActionEvent evt) {
    CorrectWords correct_words=new CorrectWords();
    setVisible(false);
    correct_words.setVisible(true);
}

private void find_med_valueActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    Med med=new Med();
    setVisible(false);
    med.setVisible(true);
}
```

The user is first shown the home page and is directed to the correct words finding page or the med value finding page, depending on the button the user clicks. The visibility of the current home page is turned off and the visibility of the page you want to open is turned on.

```
public static ArrayList<String> vocabulary = new ArrayList();

public static String readTxt(String txtName) {

            String txt = "";

            try
            {
                    //the file to be opened for reading
                    FileInputStream fis=new FileInputStream(txtName);
                    Scanner sc=new Scanner(fis, "ISO-8859-9");    //file to be scanned

                    //returns true if there is another line to read
                    while(sc.hasNextLine())
                    {
                            vocabulary.add(sc.nextLine());
                    }
                    sc.close();     //closes the scanner
            }
            catch(IOException e)
            {
                    e.printStackTrace();
            }

            return txt;

    }
```

The readTxt function is used to read a specified text file and return its content as a String. The function uses txtName as a parameter, a String containing the name (file path) of the text file to be read. txt, which is an empty String, is created. A file reading process is initiated using FileInputStream and Scanner to read the text file named txtName. "ISO-8859-9" character set is used in this process. The file is scanned line by line (while(sc.hasNextLine())), each line is added to a collection called vocabulary. Scanner is closed (sc.close()). If an error occurs while reading the file, an IOException type exception is caught and printed on the screen

(e.printStackTrace()). txt, which is an empty String, is returned by the function. This function stores the contents of the file by adding each line in the text file to a collection called vocabulary.

```
public static int calculateLevenshteinDistance(String word1, String word2) {
    int len1 = word1.length();
    int len2 = word2.length();

    int[][] dp = new int[len1 + 1][len2 + 1];

    for (int i = 0; i <= len1; i++) {
        for (int j = 0; j <= len2; j++) {
            if (i == 0) {
                dp[i][j] = j;
            } else if (j == 0) {
                dp[i][j] = i;
            } else if (word1.charAt(i - 1) == word2.charAt(j - 1)) {
                dp[i][j] = dp[i - 1][j - 1];
            } else {
                dp[i][j] = 1 + Math.min(Math.min(dp[i - 1][j], dp[i][j - 1]), dp[i - 1][j - 1]);
            }
        }
    }

    return dp[len1][len2];
}
```

The calculateLevenshteinDistance function implements an algorithm used to calculate the Levenshtein distance between two words. Levenshtein distance refers to the minimum number of editing operations required to transform one word into another. These operations can be adding characters, removing characters, or changing characters. The function calculates the Levenshtein distance between two words using the dynamic programming method. Input Parameters: word1 and word2: Two words to compare. Matrix Creation: A matrix named int[][] dp is created. The number of rows of this matrix is determined as the length of word1 + 1, and the number of columns is determined as the length of word2 + 1. This extra row and column ensures that zero-length words are also taken into account. Assigning Initial Values: The first column of the matrix represents how many insertions are required if each character of word1 is inserted without word2. Therefore, the assignment dp[i][0] = i is made. The first row of the matrix represents how many insertions would be required if each character of word2 was inserted without word1. Therefore, the assignment dp[0][j] = j is made. Dynamic Programming Step: Comparing each character of two words and updating the matrix is done step by step. If word1.charAt(i - 1) equals word2.charAt(j - 1), these characters are matched and their previous state (dp[i - 1][j - 1]) is used. If it does not match, dp[i][j] is updated by taking the minimum of three different cases. These states come from the upper left (dp[i - 1][j - 1]), left (dp[i][j - 1]), and upper (dp[i - 1][j]) neighboring cells, respectively. Result: The value in the lower right corner of the matrix represents the Levenshtein distance between two words. This value is returned by the function. This method calculates all states step by step using a matrix to compare each character. Thanks to dynamic programming, recalculation of previous states is avoided, thus optimizing the calculation time.

```
public static ArrayList<String> findNearestWords(String target, ArrayList<String> vocabulary, int numAlternatives) {
    ArrayList<StringDistancePair> distances = new ArrayList<>();



    for (String word : vocabulary) {
        int distance = calculateLevenshteinDistance(target, word);
        distances.add(new StringDistancePair(word, distance));
    }

    distances.sort((pair1, pair2) -> Integer.compare(pair1.getDistance(), pair2.getDistance()));

    ArrayList<String> result = new ArrayList<>();
    for (int i = 0; i < numAlternatives && i < distances.size(); i++) {
        result.add(distances.get(i).getString());
    }

    return result;
}
```

The findNearestWords function uses Levenshtein distance to measure the similarity between a target word and a vocabulary word. First, it calculates the Levenshtein distances of the target word to each word and associates these distances with the words. Then, it sorts these distances from smallest to largest. Finally, it selects and returns the specified number of closest alternative words using the sorted list. This is used to evaluate similarities between two words and make suggestions.

```
private void list_5_correct_words_buttonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    long start = System.currentTimeMillis();
    String userInput = wordField.getText();

    String correct_words = "";
    ArrayList<String> alternatives = findNearestWords(userInput, HomePage.vocabulary, 5);

    correct_words += "<html><div style='font-family: Arial, sans-serif; font-size: 14pt; margin: 10px;'>";
    correct_words += "<strong>Alternative correct words for '" + userInput + "':</strong><br><br>";

    for (int i = 0; i < alternatives.size(); i++) {
        correct_words += "<span style='color: #007bff;'>" + (i + 1) + ".</span> " + alternatives.get(i) + "<br>";
    }

    correct_words += "</div></html>";

    long finish = System.currentTimeMillis();

    long timeElapsed = finish - start;
    timeLabel.setText("Finding first 5 alternative correct words according to "+ userInput +" took "+timeElapsed+" millisecond.");
    text_area.setText(correct_words);


}
```

The list_5_correct_words_buttonActionPerformed function performs an operation that finds the 5 closest alternatives of the word entered by a user and displays them as a list on the screen. The list_5_correct_words_buttonActionPerformed method represents a GUI (Graphical User Interface) event. In this method, a word input from the user (wordField.getText()) is received. Then, using the findNearestWords function, the 5 closest alternative words of this user input are found. A String named verify_words is created and edited in HTML format. This arrangement includes alternative words listed in color below the word the user entered. A text containing information about the time it takes to find alternative words (how long it takes to find them) is written on the timeLabel tag. This time calculates the time from the beginning to the end of the process. An area called text_area is used to display the content of correct_words. This field displays text in HTML format containing the alternative words found.

```
private void find_med_buttonActionPerformed(java.awt.event.ActionEvent evt) {

    long start = System.currentTimeMillis();

    String source = sourceTextField.getText();
    String target = targetTextField.getText();

    String[][] med_matrix =new String[source.length() + 2][target.length() + 2];

    for (int i = 0; i < source.length() + 2; i++) {
        for (int j = 0; j < target.length() + 2; j++) {
            if(i == 0 && j == 0){
                med_matrix[i][j] = "";
            }
            else if(i + j == 1){
                med_matrix[i][j] = "#";
            }
            else if(j == 0){
                med_matrix[i][j] = String.valueOf(source.charAt(i-2));
            }
            else if(i == 0){
                med_matrix[i][j] = String.valueOf(target.charAt(j-2));
            }
            else if(j == 1){
                med_matrix[i][j] = String.valueOf((i-1));
            }
            else if(i == 1){
                med_matrix[i][j] = String.valueOf((j-1));
            }
```

```
        else if(med_matrix[i][0].equals(med_matrix[0][j])){
            med_matrix[i][j] = med_matrix[i - 1][j - 1];
        }
        else{
            med_matrix[i][j] = String.valueOf(Math.min(Integer.parseInt(med_matrix[i - 1][j - 1]), Math.min(Integer.parseInt(med_matrix[i - 1][j ]), Integer.parseInt(med
        }
    }
}
```

```
    ArrayList<Integer> row_index_path = new ArrayList();
    ArrayList<Integer> column_index_path = new ArrayList();

    ArrayList<String> transactions = new ArrayList();

    int k = source.length() + 1;
    int l = target.length() + 1;

    while(!(k == 1 && l == 1)){

        row_index_path.add(k);
        column_index_path.add(l);

        int left = 999 ;
        int cross = 999 ;
        int top = 999 ;
        try {
            if(!(k == 1 && l != 1)){

                top = Integer.parseInt(med_matrix[k-1][l]);
            }

        } catch (NumberFormatException e) {

        }
        try {
            cross = Integer.parseInt(med_matrix[k-1][l-1]);

        } catch (NumberFormatException e) {

        }
```

```
        try {
            if(!(l == 1 && k != 1)){
                left = Integer.parseInt(med_matrix[k][l-1]);
            }


        } catch (NumberFormatException e) {

        }

        if(cross <= left && cross <= top ){

            if (!med_matrix[k][0].equals(med_matrix[0][l])) {
                transactions.add(0,"substitution " + med_matrix[k][0] +" with " + med_matrix[0][l] );
            }
            k--;
            l--;
        }else if(left <= top && left <= cross ){
            transactions.add(0,"insertion " + med_matrix[0][l]);
            l--;
        }else{
            transactions.add(0,"deletion " + med_matrix[k][0]);
            k--;
        }
    }

    row_index_path.add(1);
    column_index_path.add(1);
```

```
String transactions_str = "<html><div style='font-family: Arial, sans-serif; font-size: 14pt; margin: 10px;'>";
for (int i = 0; i < transactions.size(); i++) {
    transactions_str += "<span style='color: #007bff;'>" + (i + 1) + ".</span> " + transactions.get(i) + "<br>";
}

transactions_str += "</div></html>";


String matrixText = "<html><table border='1' cellpadding='10' style='font-size: 16pt;'>";
for (int i = 0; i < med_matrix.length; i++) {
    matrixText += "<tr>";
    for (int j = 0; j < med_matrix[i].length; j++) {

        if(row_index_path.get(row_index_path.size() - 1) == i && column_index_path.get(column_index_path.size() - 1) == j ){
            matrixText += "<td><span style='color: #f00;'>" + med_matrix[i][j] + "</span></td>";
            row_index_path.remove(row_index_path.size() - 1);
            column_index_path.remove(column_index_path.size() - 1);
        }
        else{
            matrixText += "<td>" + med_matrix[i][j] + "</td>";
        }


    }
    matrixText += "</tr>";
}
matrixText += "</table></html>";

String last = matrixText+transactions_str;
```

```
resultLabel.setText(matrixText);
transactionsLabel.setText(transactions_str);

long finish = System.currentTimeMillis();
long timeElapsed = finish - start;
timeLabelMed.setText("Process took "+timeElapsed+" millisecond.");
```

The find_med_buttonActionPerformed function implements an algorithm that finds the Minimum Edit Distance (MED) or Levenshtein distance between two texts and edit actions.

Two texts named source and target are received. A two-dimensional array named med_matrix is created. This array stores the Levenshtein distance and regularization operations in each cell. The dimensions of the matrix are determined by adding 2 to the length of both texts. Using two nested loops, each matrix cell is calculated. Cell values are determined as follows: cell (0, 0) represents an empty string. When i + j == 1, cell (i, j) represents the symbol #. When j == 0, cell (i, j) represents the corresponding character of the source text. When i == 0, cell (i, j) represents the corresponding character of the target text. When j == 1, cell (i, j) represents the number i-1. When i == 1, cell (i, j) represents the number j-1. If med_matrix[i][0] and med_matrix[0][j] are equal, cell (i, j) takes the value in the upper left corner. If not, cell (i, j) gets the value with 1 added to the minimum of the values in its upper left, left and upper neighbors. Each cell of the matrix shows the Levenshtein distance between two texts and the editing operations performed. The matrix resulting from the calculations is stored in a two-dimensional array called med_matrix.

Two ArrayLists named row_index_path and column_index_path are created. These lists are used to store cell indexes in the shortest edit path. An ArrayList named transactions is created. This list is used to store edit operations that have occurred. Two index variables named k and l are created. These indices will be used to move from the lower right corner of the matrix to the upper left corner. With the while(!(k == 1 && l == 1)) loop, the editing operations are followed up to the lower right corner of the matrix. At each step of the loop, the existing indices (k and l) are added to the row_index_path and column_index_path lists. The values of the top left, top, and left neighboring cells are taken and assigned to the variables left, cross, and top. During this process, NumberFormatException is used against possible error situations. By choosing the

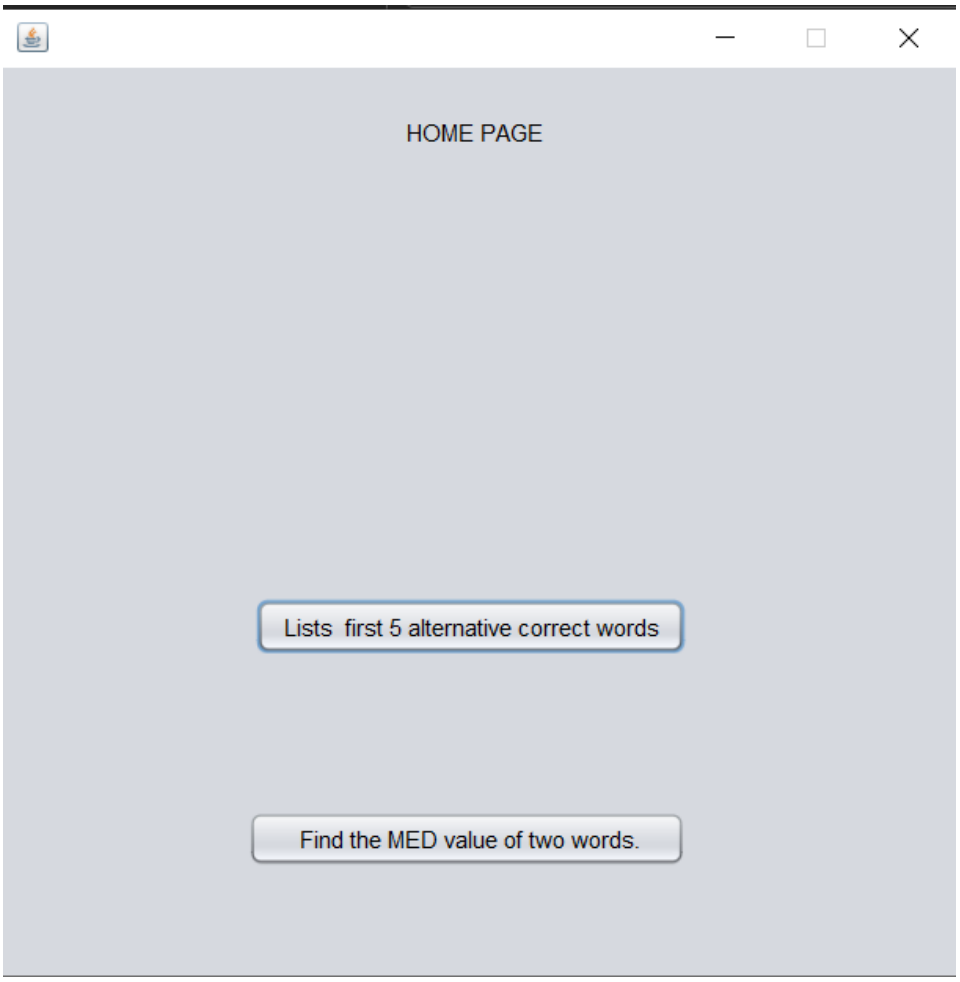
smallest value, it is determined which editing operation will be performed: If the cross is smallest, the "substitution" operation is performed and k and l are reduced by one unit. If left is the smallest, "insertion" is performed and only l is decreased by one unit. If ball is the smallest, "deletion" is performed and only k is reduced by one unit. The edited action is added to the transactions list. When the loop finishes, the indices (1, 1) of the upper left corner are added to the row_index_path and column_index_path lists. The row_index_path and column_index_path lists contain the indices of that path, while the transactions list contains the editing operations that occurred.

A String, transactions_str, is created. This String contains styled text in HTML format. Each edit operation in the transactions list is processed using a loop. Each edit action is added as a numbered list element. It is placed within HTML tags along with color and style settings. After the loop ends, the transactions_str text is completed by adding closing HTML tags.

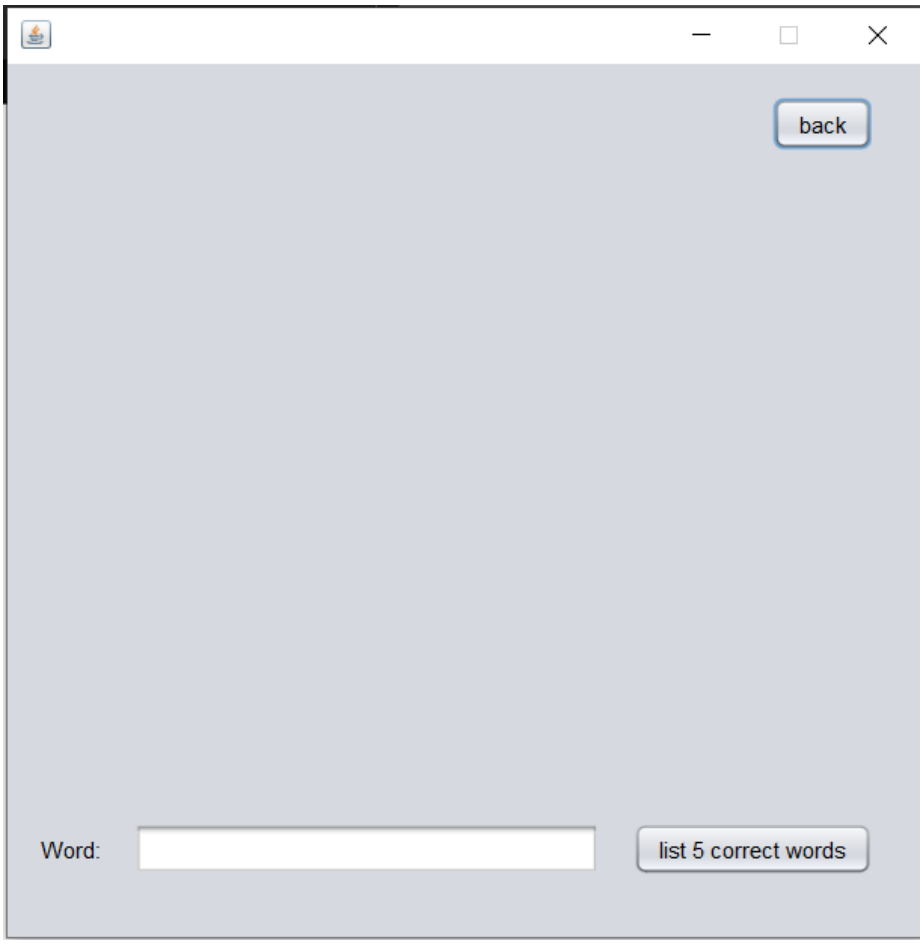
matrixText, which is a String, is created. This String contains a table in HTML format. Using two loops, med_matrix is traversed. Each cell is inserted as an HTML table cell. It is placed within HTML tags along with color and style settings. If the current cell corresponds to the last element in the row_index_path and column_index_path lists that show the shortest editing path, this cell is highlighted with a special color. Red color (#f00) is used for highlighting. As each row is completed, an HTML table row (<tr>) is created. After the loop ends, the matrixText text is completed by adding closing HTML tags.

2. Screenshots of program

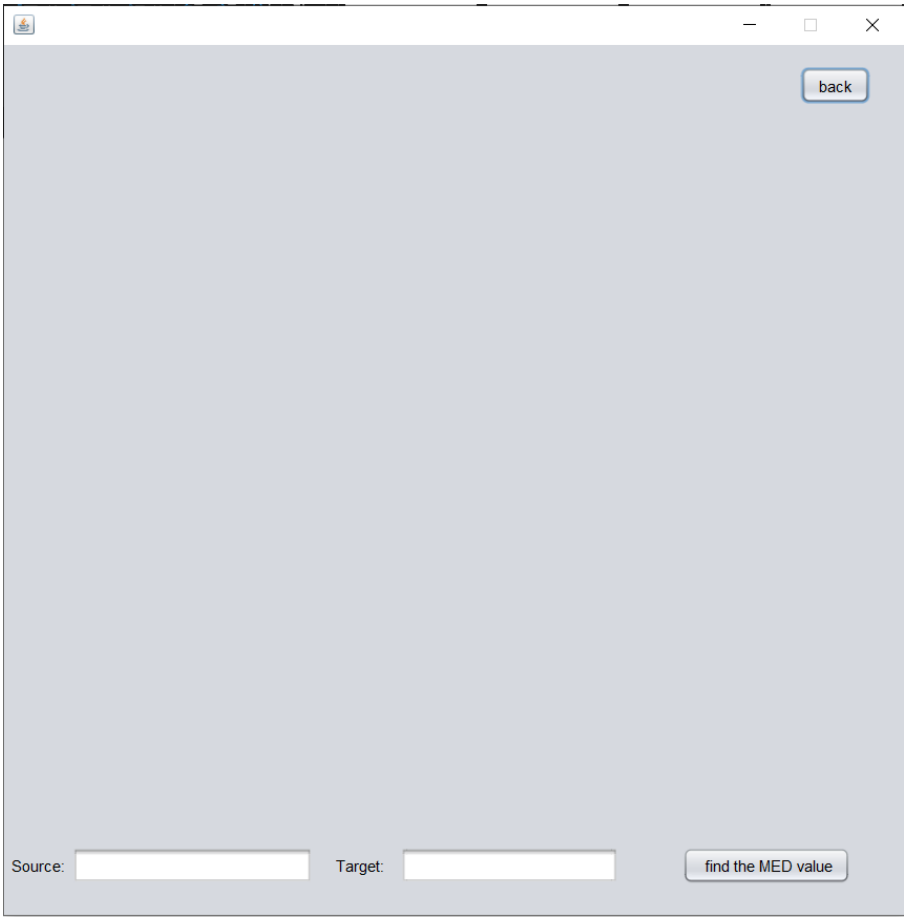
2.1 Home Page



2.2 List Correct Words Page



2.3 Med Value Page



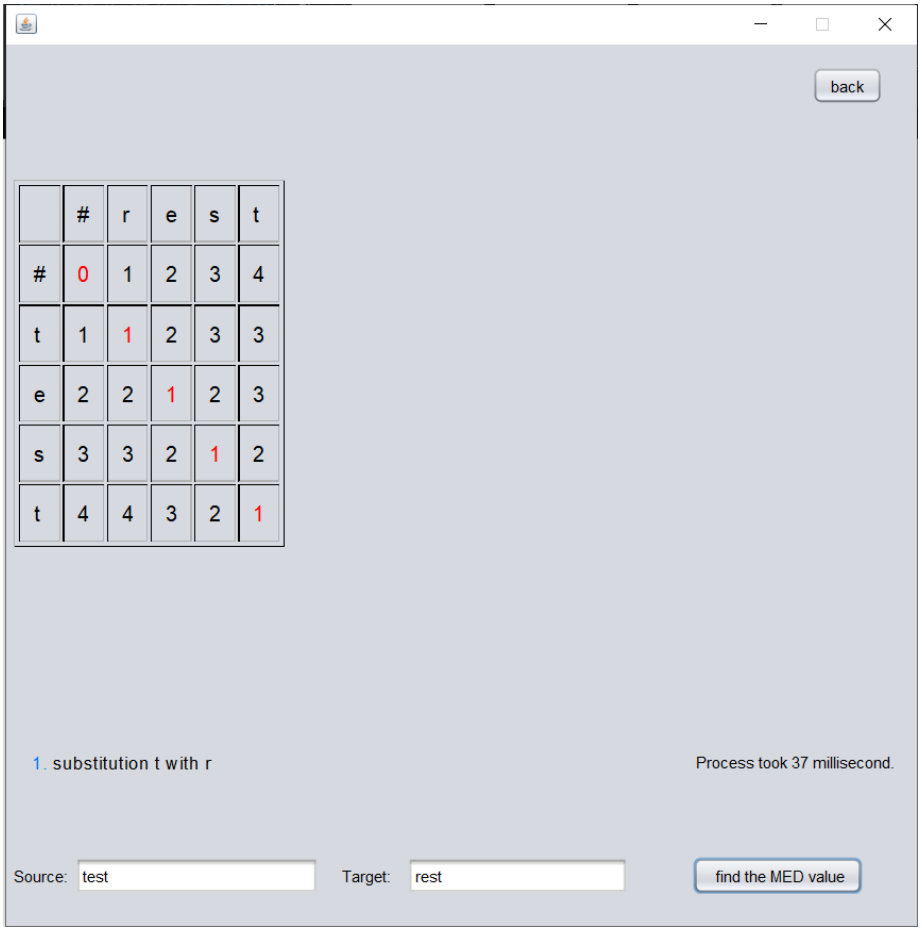
3. Test results

3.1 Correct Words Test for “Ali”

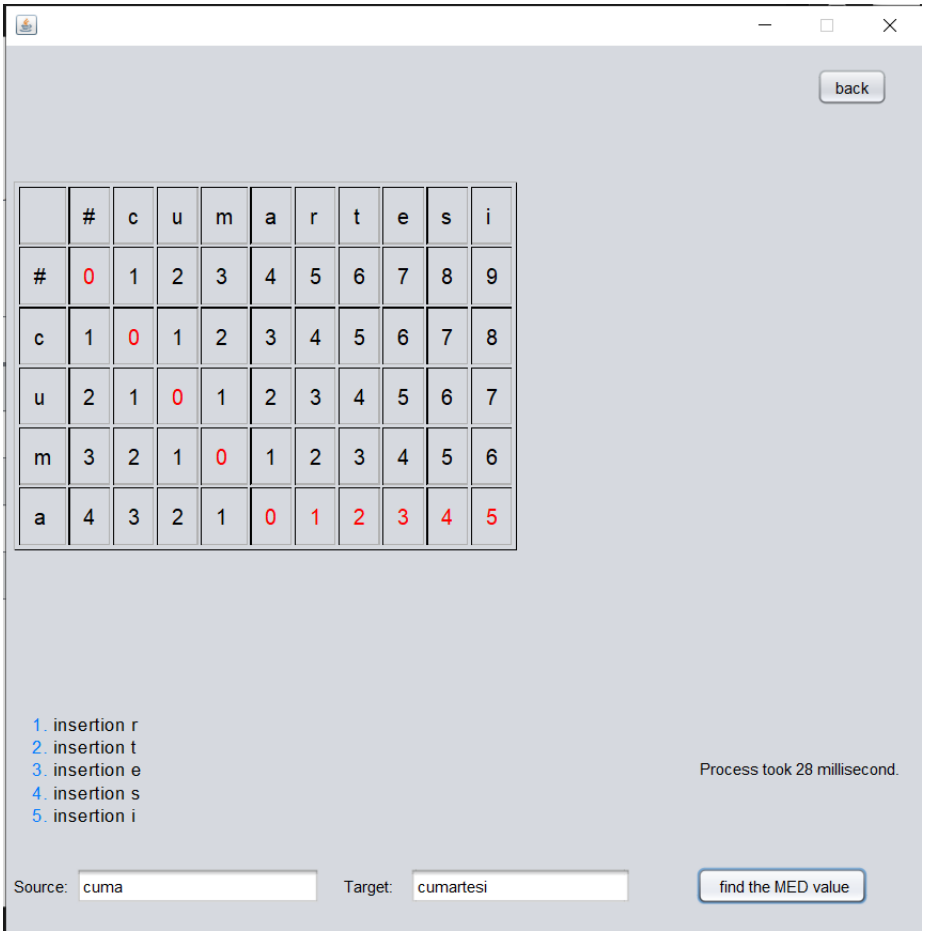
3.2 Correct Words Test for “Şiyar”

3.3 Correct Words Test for “arslan”

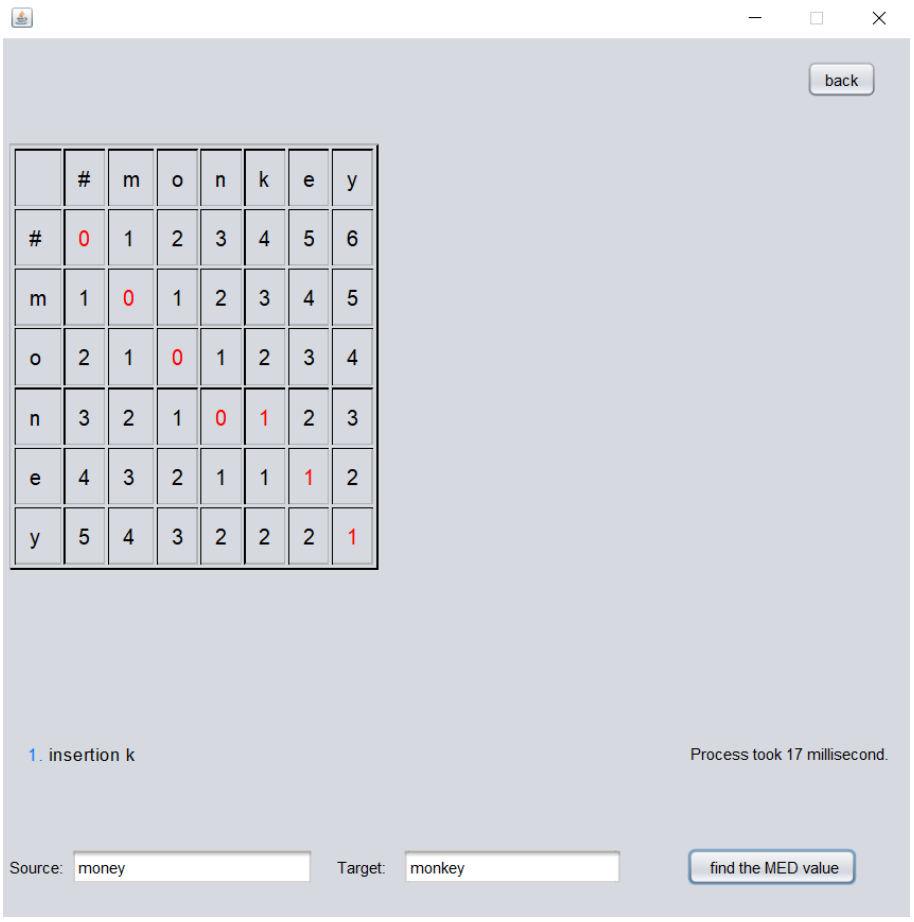
3.4 Finding Med Values Test for test – rest



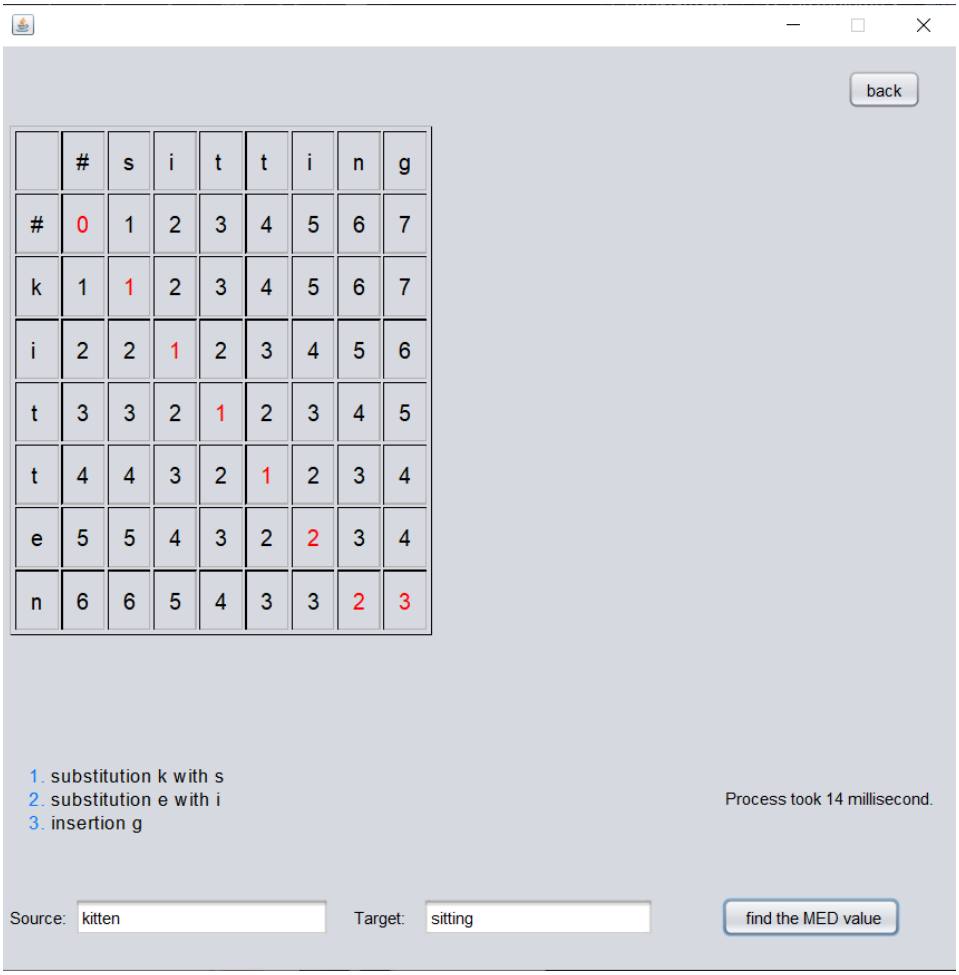
3.5 Finding Med Values Test for cuma – cumartesi



3.6 Finding Med Values Test for money – monkey

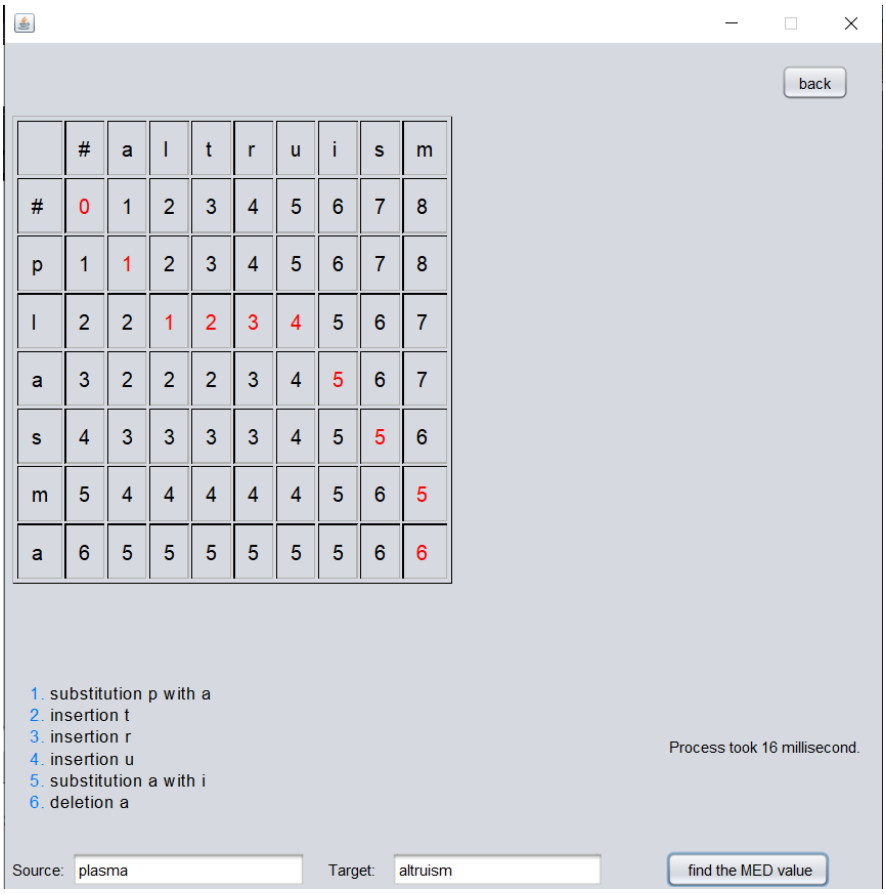


3.7 Finding Med Values Test for kitten – sitting



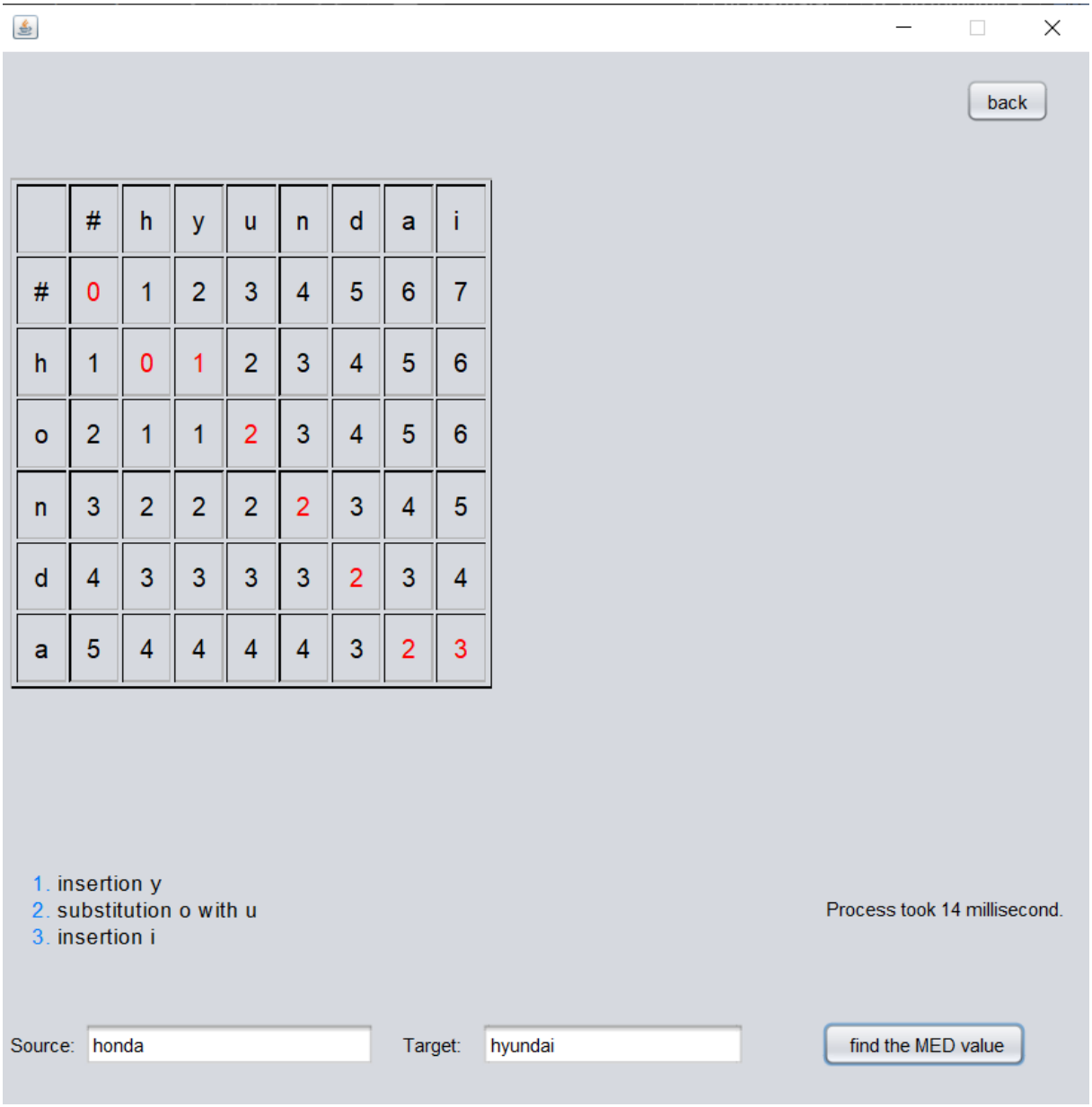
	#	s	i	t	t	i	n	g
#	0	1	2	3	4	5	6	7
k	1	1	2	3	4	5	6	7
i	2	2	1	2	3	4	5	6
t	3	3	2	1	2	3	4	5
t	4	4	3	2	1	2	3	4
e	5	5	4	3	2	2	3	4
n	6	6	5	4	3	3	2	3

3.8 Finding Med Values Test for plasma – altruism



	#	a	l	t	r	u	i	s	m
#	0	1	2	3	4	5	6	7	8
p	1	1	2	3	4	5	6	7	8
l	2	2	1	2	3	4	5	6	7
a	3	2	2	2	3	4	5	6	7
s	4	3	3	3	3	4	5	5	6
m	5	4	4	4	4	4	5	6	5
a	6	5	5	5	5	5	5	6	6

3.8 Finding Med Values Test for honda – hyundai



	#	h	y	u	n	d	a	i
#	0	1	2	3	4	5	6	7
h	1	0	1	2	3	4	5	6
o	2	1	1	2	3	4	5	6
n	3	2	2	2	2	3	4	5
d	4	3	3	3	3	2	3	4
a	5	4	4	4	4	3	2	3