CME4408 INTRODUCTION TO NATURAL LANGUAGE PROCESSING

Assignment 2

İsmail Berkin Serin

2016510059

In this report the implementation of Minimum Edit Distance (MED) in **Java** programming language will be explained in detail with many examples and runtime analysis.

Since the implementation is done in Java, **Swing** GUI is implemented in order to produce easy to use application.

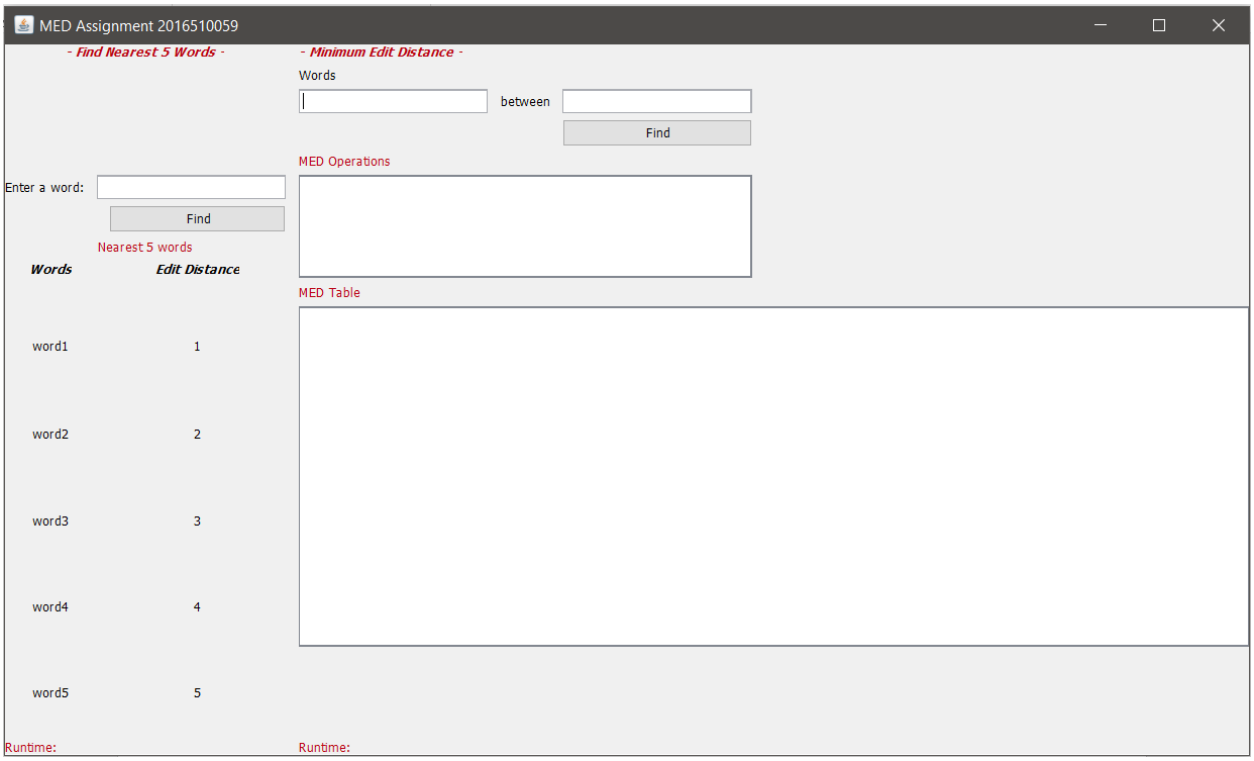
***** The .jar file for this implementation assumes that the sozluk.txt belongs in the same directory as the jar file. *****

There are three main classes.

```java
import javax.swing.*;

public class Main {


    public static void main(String[] args) throws ClassNotFoundException, UnsupportedLookAndFeelException, InstantiationException,

        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());


        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                MedGui mg = new MedGui();
                mg.setVisible(true);
            }
        });


    }
}
```

The main class is required to call the MedGui graphical user interface panel. The panel is set to visible. There are no other functions for this class. For a better look for the application the UIManager is called so the general theme for each operating systems will be applied to the application. The function is also tied to a runnable thread for convenience.

The most important class is the MedGui.java class. This class holds all the necessary components and necessary functions for MED analysis.



This is the general outlook for the application. Further explanation for the function in MedGui class:

```
static int min(int x, int y, int z)
{
    if (x <= y && x <= z)
        return x;
    if (y <= x && y <= z)
        return y;
    else
        return z;
}
```

This min function takes three integers and returns the minimum value amongst them. This function is particularly useful when backtracking and table creation.

This editDistDP function is the main function which is implemented by Dynamic Programming technique. This function takes four arguments including two different strings namely the words and their lengths.

```
static int[][] editDistDP(String str1, String str2, int m,
                          int n)
{
    // Create a table to store results of subproblems
    int[][] dp = new int[m + 1][n + 1];

    // Fill d[][] in bottom up manner
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            // If first string is empty, only option is
            // to insert all characters of second string
            if (i == 0)
                dp[i][j] = j; // Min. operations = j

                // If second string is empty, only option is
                // to remove all characters of second string
            else if (j == 0)
                dp[i][j] = i; // Min. operations = i

                // If last characters are same, ignore last
                // char and recur for remaining string
            else if (str1.charAt(i - 1)
                    == str2.charAt(j - 1))
                dp[i][j] = dp[i - 1][j - 1];
```

```
                // If the last character is different,
                // consider all possibilities and find the
                // minimum
            else
                dp[i][j] = 1
                        + min(dp[i][j - 1], // Insert
                        dp[i - 1][j], // Remove
                        dp[i - 1]
                                [j - 1]); // Replace
        }
    }

    return dp;
}
```

In the else part, the minimum of the upper, left upper and left neighbors are found and according to that value the Insertion, Removal or Replace operation is done. All of the operations cost 1. Since the Swing GUI is used the user enters two words and those two words are used to calculate minimum edit distance between them. When the find button is pressed the necessary operations are done and the operations needed to calculate MED is written along with the table to the GUI.

```
findButton1.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        medOperationsText.setText("");
        long startTime = System.nanoTime();
        String word1, word2, table = "";
        word1 = medWord1.getText();
        word2 = medWord2.getText();
        int[][] dp = editDistDP(word1,word2,word1.length(),word2.length());
        String [][] temp = new String[word1.length() +2][word2.length() +2];
        temp[0][0] = "";
        temp[1][0] = "#";
        temp[0][1] = "#";

        for (int i = 0; i < word1.length(); i++) {
            temp[i+2][0] = String.valueOf(word1.charAt(i));
        }
        for (int i = 0; i < word2.length(); i++) {
            temp[0][i+2] = String.valueOf(word2.charAt(i));
        }

        for (int i = 0; i < dp.length; i++) {
            for (int j = 0; j < dp[i].length; j++) {
                temp[i+1][j+1] = String.valueOf(dp[i][j]);
            }
        }
```

```
int k = word1.length()+1;
int l = word2.length()+1;
int insert = 0;
int remove = 0;
int replace = 0;

while(!(k == 1 && l == 1)){
    if(l == 1){
        temp[k - 1][l] = temp[k - 1][l] + String.valueOf("\u2193");
        if(temp[k][l].substring(0,temp[k][l].length() - 1) != temp[k - 1][l].substring(0,temp[k - 1][l].length() - 1)){
            remove++;
        }
        k = k-1;
        continue;
    }
    int val = min(Integer.parseInt(temp[k][l - 1]), // Insert
            Integer.parseInt(temp[k - 1][l]), // Remove
            Integer.parseInt(temp[k - 1][l - 1])); // Replace
    if (val == Integer.parseInt(temp[k - 1 ][l - 1])){
        String a,b;
        temp[k - 1][l - 1] = temp[k - 1][l - 1] +String.valueOf("\u2198");
        if (temp[k][l].length() == 1){
            a = temp[k][l];
        }else{
            a = temp[k][l].substring(0,temp[k][l].length() - 1);
        }
        b = temp[k - 1][l - 1].substring(0,temp[k - 1][l - 1].length() - 1);
```

After all the necessary operations the table is created and the values are written to the corresponding areas on the GUI:

```
tableText.setText(table);
long endTime   = System.nanoTime();
long totalTime = (endTime - startTime) / 1000000;
runtimeMEDlabel.setText("Runtime: " + totalTime + " ms");
medOperationsText.setText(medOperationsText.getText() + insert +
        " Insert Operations \n" + remove + " Remove Operations \n" + replace + " Replace Operations \n" + "\n");
medOperationsText.setText(medOperationsText.getText() + "Minimum Edit Distance: " + temp[word1.length()+1][word2.length()+1]+
```

The runtime of this MED calculation is also written on the GUI in milliseconds. Total operations are shown to the user.

The other find button is used to find the nearest 5 words for a user given word.

```
public void actionPerformed(ActionEvent e) {
    long startTime = System.nanoTime();
    String word;
    word = textField1.getText();
    Charset iso88599charset = Charset.forName("ISO-8859-9");
    try {
        String s = Files.readString(Path.of( first: "sozluk.txt"), iso88599charset);
        String dictionary[] = s.split( regex: "\r\n");
        HashMap<String, Integer> map = new HashMap<>();
        for (String item: dictionary) {
            map.put(item,editDistDP(word,item,word.length(),item.length())[word.length()][item.length()]);
        }

        Object[] a = map.entrySet().toArray();
        Arrays.sort(a, new Comparator() {
            public int compare(Object o1, Object o2) {
                return ((Map.Entry<String, Integer>) o1).getValue()
                        .compareTo(((Map.Entry<String, Integer>) o2).getValue());
            }
        });
```

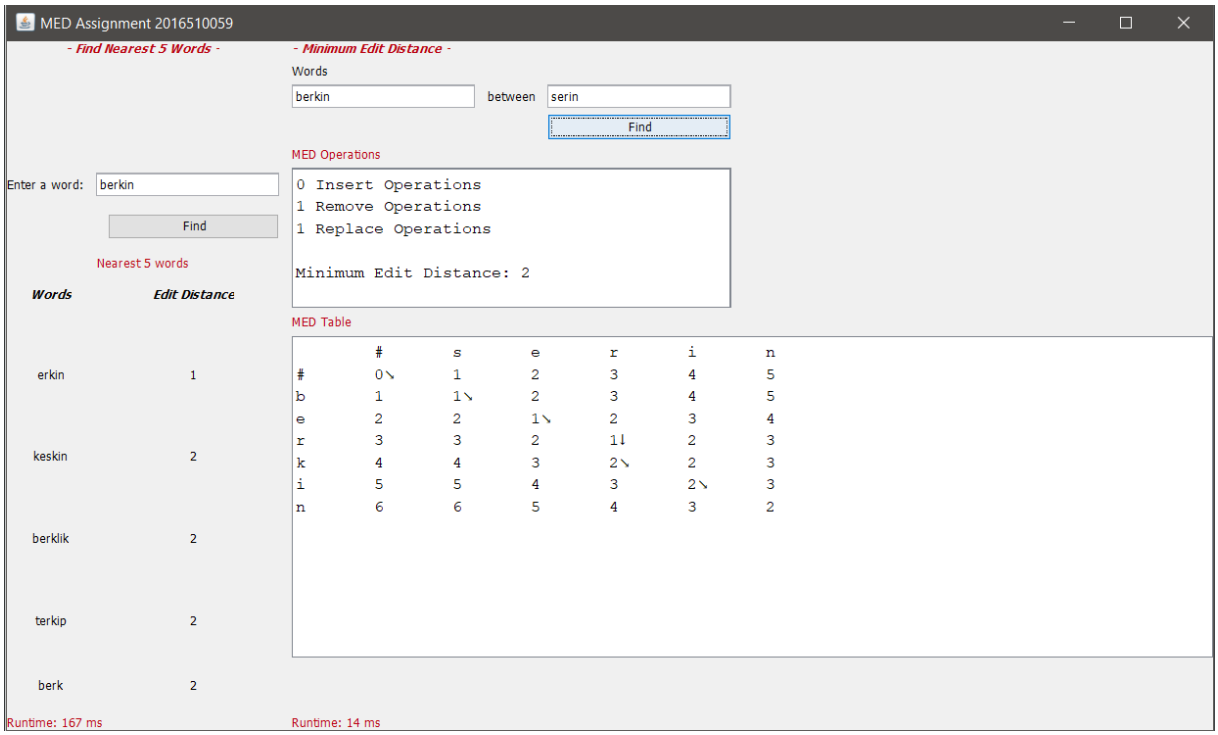
The dictionary sozluk.txt is read line by line and split into words with the encoding iso-8859-9. So there are no problems with the Turkish characters. After that the comparison is made with all of the words in the dictionary. The results are held in a hashmap format and they are sorted by their values. So the nearest 5 words can be easily retrieved.

```
    word1.setText(((Map.Entry<String, Integer>) a[0]).getKey());
    ed1.setText(String.valueOf(((Map.Entry<String, Integer>) a[0]).getValue()));
    word2.setText(((Map.Entry<String, Integer>) a[1]).getKey());
    ed2.setText(String.valueOf(((Map.Entry<String, Integer>) a[1]).getValue()));
    word3.setText(((Map.Entry<String, Integer>) a[2]).getKey());
    ed3.setText(String.valueOf(((Map.Entry<String, Integer>) a[2]).getValue()));
    word4.setText(((Map.Entry<String, Integer>) a[3]).getKey());
    ed4.setText(String.valueOf(((Map.Entry<String, Integer>) a[3]).getValue()));
    word5.setText(((Map.Entry<String, Integer>) a[4]).getKey());
    ed5.setText(String.valueOf(((Map.Entry<String, Integer>) a[4]).getValue()));
    long endTime   = System.nanoTime();
    long totalTime = (endTime - startTime) / 1000000;
    runtimeLabel.setText("Runtime: " + totalTime + " ms");
} catch (IOException ioException) {
    ioException.printStackTrace();
}
```

Since the hashmap is already sorted the first 5 nearest words are written on the GUI with their corresponding MED to the given word. The runtime of this operation is also calculated in milliseconds and written on the GUI.

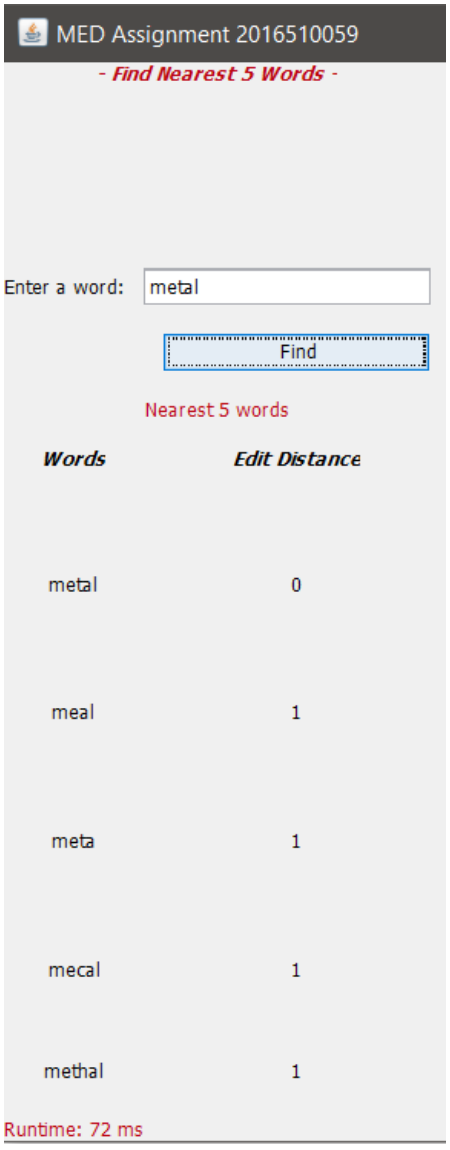
The last class is the class necessary for the Swing. The class is in the form of ".form".

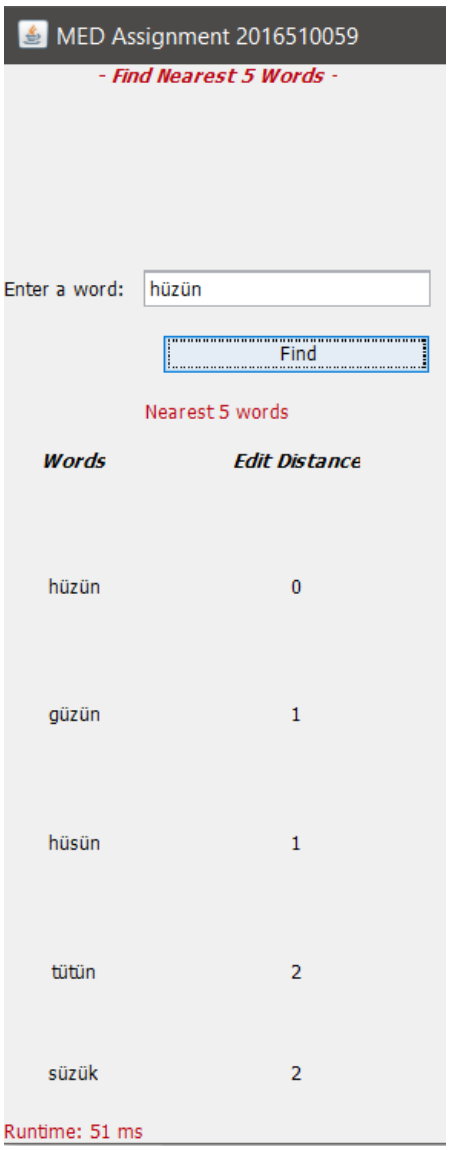
Test Results:

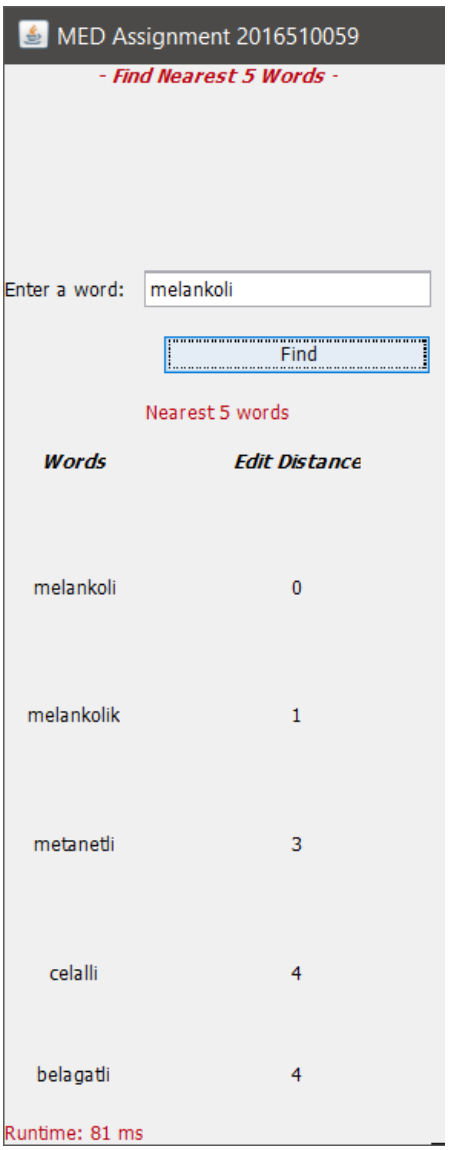


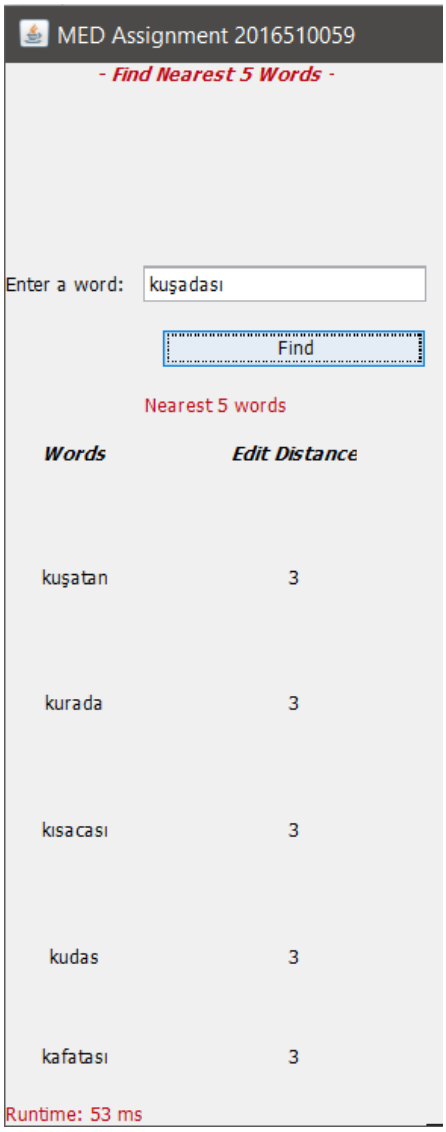
The runtime for the part1 which is finding the nearest 5 words is 167ms. The runtime for the part2 which is minimum edit distance between two words is 14ms. In total the average running time is 167ms + 14ms = 181ms.

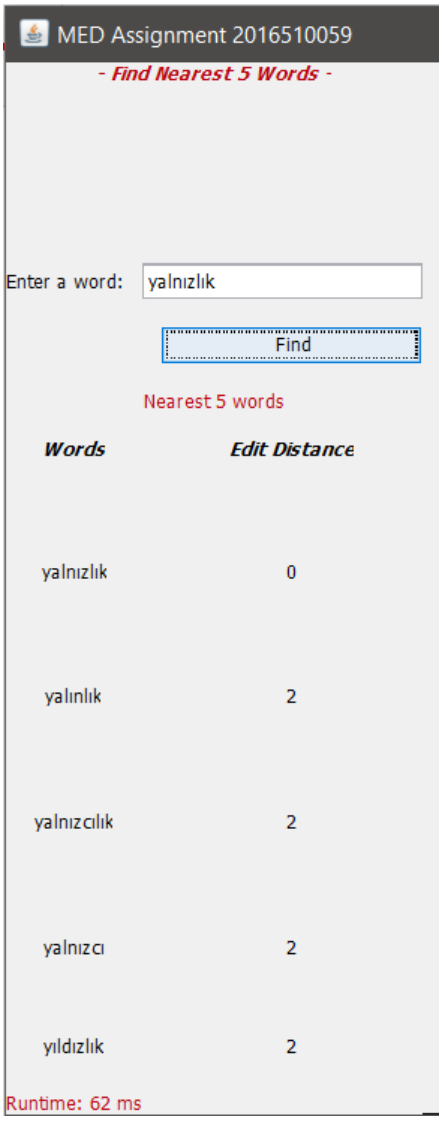
Alternative correct words list for at least 5 different words:





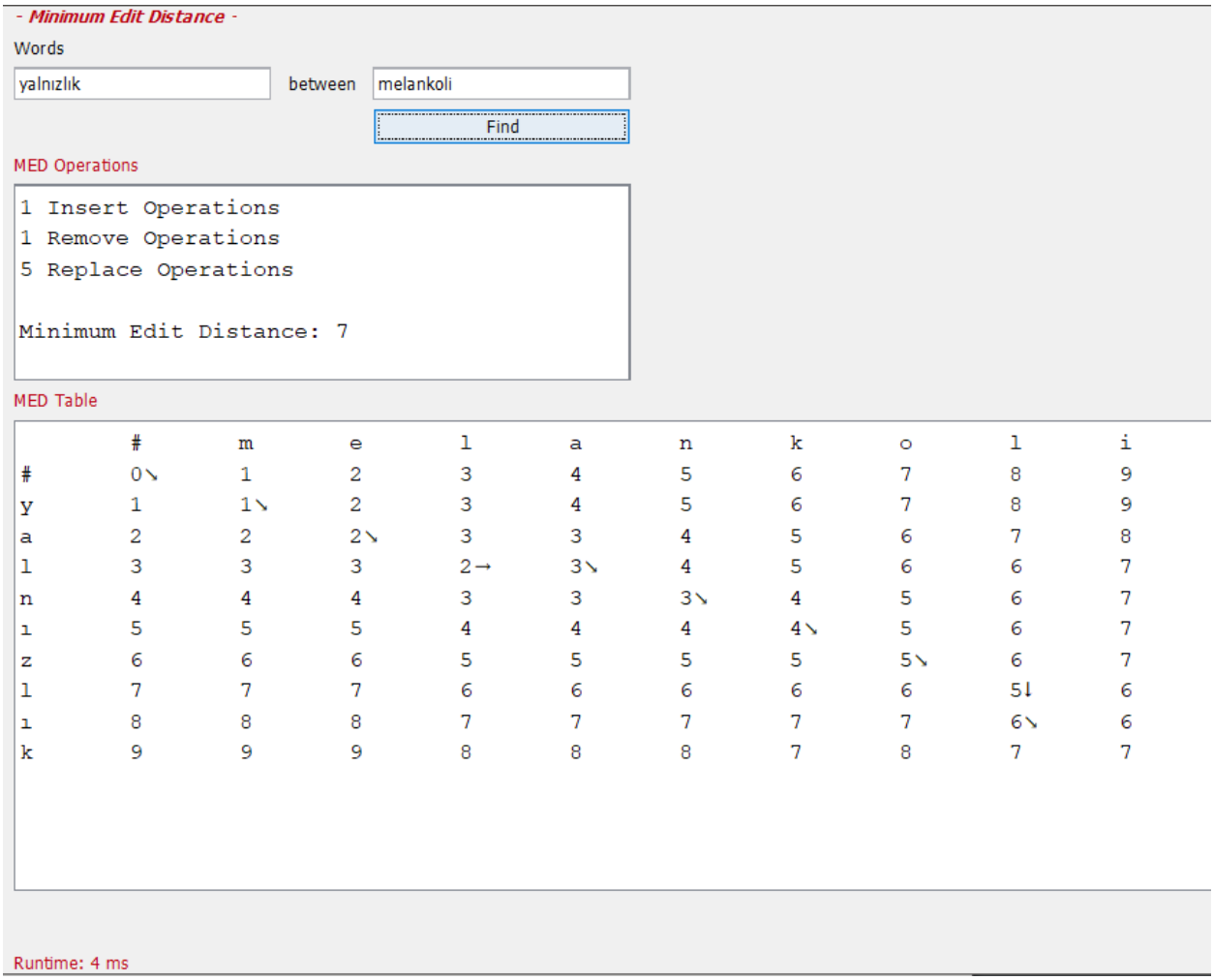




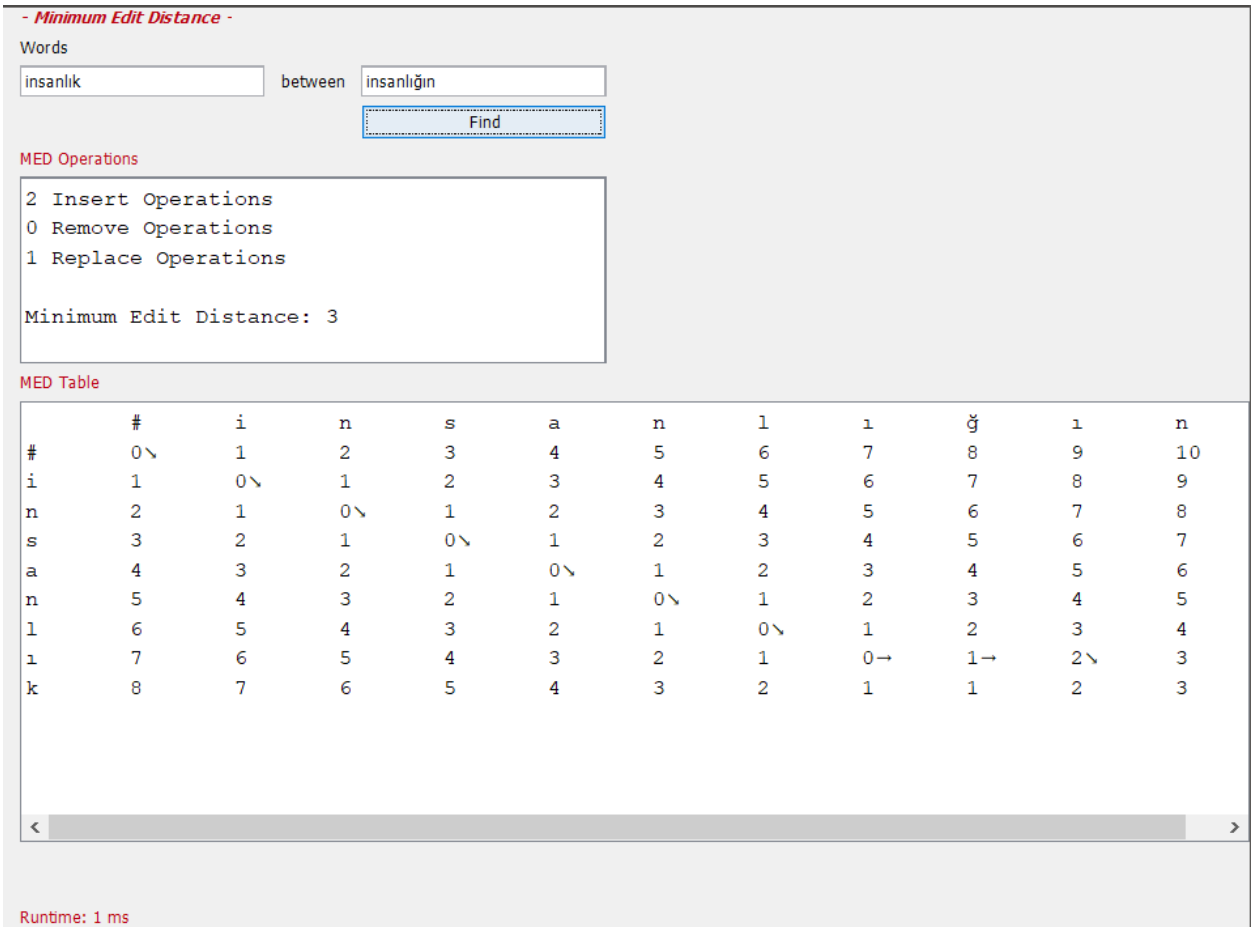


The results are shown above for 5 different words. The runtime is shown to the user in milliseconds along with the 5 nearest words and their Minimum Edit Distances. Some of the words have 0 MED since the words are exactly the same in the dictionary so it makes sense.

For the part two, where the user gives two words and their MED is calculated and shown to the user with the operations made. This part is run two times and the results are:



It can be seen from the result image that the MED between “yalnızlık” and “melankoli” is 7. There are 1 insert operation, 1 remove operation and 5 replace operations with the 7 total operations made. The whole process took 4ms to run.



The MED between the words “insanlık” and “insanlığın” is 3. 2 insert operations and 1 replace operation in total 3 operations are made. Whole process took 1ms to run. The MED table is shown to the user.

The project is fully working without any errors.