# CmpE 493-Information Retrieval
# Spring 2020
# Assignment 1-Spell Error Corrector

Muhammed Bera Kaya
2016400234

April 10, 2020

# Contents

# 1 Implementation

I used Java language for implementation.In this program, we have 4 classes:

1. Main Class
2. Matrix_calc Class: Given a pair of strings,this class creates their Damerau-Levenshtein distance matrix,calculates their edit distance and extracts the edit operations needed.
3. Corrector Class: The class where the main work is done.In this class,the dictionary and confusion matrices are created using corpus and spell-errors files.The spell error corrections are also done by methods in this class, by calculating probabilities using dictionary and confusion matrices for all candidate words.
4. Operation Class:Represents the edit operations.It has 3 fields:operation name, operand 1 and operand 2.Example:("insertion","a","b")

## 1.1 Matrix_calc Class

### 1.1.1 Producing Distance Matrix

The Levenshtein distance matrix algorithm that we have learned in class does not consider the "transposition" operation.That's why,I changed the algorithm a little bit. I searched for adjacent substitution operations with operands reversed (ex:sub[a,b] and sub[b,a] occurs adjacently) and converted them to be a single transposition operation.Let the distance matrix be matrix[w1.length()+1][w2.length+1](w1 and w2 are the 2 strings) as shown in the figure 1.For a given cell (matrix[j][i]) ,in the normal algorithm; we check the values of the left(matrix[j][i-1]),upper(matrix[j-1][i]) and upper left(matrix[j-1][i-1]) neighbors of the cell.In my algorithm additionally,I get the operands of the upper left cell(since substitution operation implies a diagonal movement) and check if the operands are the reversed version of the operands in the current cell.If so, I check if the upper left cell(matrix[j-1][i-1]) has a value that is equal to its upper left neighbor value plus one ((matrix[j-2][i-2])+1).Because,this would mean that the upper left cell value had been the result of a substitution operation.If so, while calculating the value of matrix[j][i], we do not add 1 to the value of the upper left cell(matrix[j-1][i-1]).There would be no extra cost of coming from that cell to current cell,since 2 adjacent reversed substitutions implies one transposition.

```java
public int[][] produceMatrix(){
    int[][] matrix=new int[w2.length()+1][w1.length()+1];
    for(int i=1;i<=w1.length();i++) {
        matrix[0][i]=i;
    }
    for(int i=1;i<=w2.length();i++) {
        matrix[i][0]=i;
    }
    for(int i=1;i<=w1.length();i++) {
        for(int j=1;j<=w2.length();j++) {
            char c1=w1.charAt(i-1);
            char c2=w2.charAt(j-1);
            int val=0;
            if(c1==c2) {
                val=Math.min(Math.min(matrix[j][i-1]+1, matrix[j-1][i]+1), matrix[j-1][i-1]);
            }else {
                val=Math.min(Math.min(matrix[j][i-1]+1, matrix[j-1][i]+1), matrix[j-1][i-1]+1);
            }

            //If sub[x,y] operation occurs right after sub[y,x] operation, it is considered as transposition[y,x] operation
            //Therefore, the distance value will not be incremented again.
            if(j>1&&i>1) {
                char c3=w1.charAt(i-2);
                char c4=w2.charAt(j-2);
                //Math.min(Math.min(matrix[j-2][i-1],matrix[j-1][i-2]), matrix[j-2][i-2])
                if(c1==c4&&c2==c3&&c1!=c2&&matrix[j-1][i-1]==matrix[j-2][i-2]+1) {
                    //System.out.println(c1 + " "+c2 + " "+ c3 + " "+ c4);
                    val=Math.min(Math.min(matrix[j][i-1]+1, matrix[j-1][i]+1), matrix[j-1][i-1] );
                }
            }

            matrix[j][i]=val;
        }
    }

    return matrix;
}
```

Figure 1: Produce distance matrix

### 1.1.2 Backtracking the distance matrix to extract operations

Starting from the m[w1.length()][w2.length()](m is the distance matrix calculated before;w1 and w2 are the 2 strings) , that is the bottom right corner of the matrix,we backtrack to the upper left corner of the matrix.I changed the normal algorithm that we have learned in class,since I should consider the transposition again.Normally,operands are not the same in cell (m[j][i]), one of the following conditions should be true:m[j][i]=m[j-1][i-1]+1 , m[j][i]=m[j][i-1]+1, m[j][i]=m[j-1][i]+1 . But, since we had changed the distance matrix production algorithm,these 3 conditions might not hold. That's why, if these 3 conditions are not true, I check if m[j][i]=m[j-1][i-1] that is if current cell value equals the upper left neighbor value. Because, that would mean that a transposition operation had occurred and the upper left neighbor value was not incremented,so it equals the current cell value.
In every iteration, I create an Operation object and save it in a Vector.I do not save any transposition operations here. I normalize the operation vector by removing adjacent reverse substitutions and adding a transposition operation in place of them.
Note: Insertion and deletion operations can have "#" character as operands.It represents the beginning of a word.

```java
public Vector<Operation> calculateEdits() {
		Vector<Operation> vec=new Vector<Operation>();
		int i=w1.length();
		int j=w2.length();
		while(i!=0||j!=0) {
			int curr=m[j][i];
			String c1;
			String c2;
			if(i!=0) c1=Character.toString(w1.charAt(i-1));
				else c1="#";		//# denotes the
beginning of the word
			if(j!=0) c2=Character.toString(w2.charAt(j-1));
				else c2="#";
			String opname="";
			if(i==0) {
				opname="del";	//deletion
				j--;
			}else if(j==0) {
				opname="ins";	//insertion
				i--;
			}else if(c1.equals(c2)) {
				if(curr==m[j-1][i-1]) {
					j--;i--;
					opname="copy";
				}else if(curr==m[j-1][i]+1) {
					j--;
					opname="del";
				}else if(curr==m[j][i-1]+1) {
					i--;
					opname="ins";
				}
			}else {
				if(curr==m[j-1][i-1]+1) {
					j--;i--;
					opname="rep";	//substitution
				}else if(curr==m[j-1][i]+1) {
					j--;
					opname="del";
				}else if(curr==m[j][i-1]+1) {
					i--;
					opname="ins";
				}else if(curr==m[j-1][i-1]) {
					j--;i--;
					opname="rep";
				}
			}
			Operation op=new Operation(opname,c1,c2);
			vec.add(op);
		}
		Collections.reverse(vec);
		this.ops=vec;
		normalizeOperations(this.ops);
		return vec;
	}
```

Figure 2: Backtracking distance matrix

## 1.2 Corrector Class

### 1.2.1 Creating the dictionary

I created the dictionary as a hashmap where the keys are distinct words in corpus and the values are the number of occurrences of a word. I read the lines in corpus.txt file one by one. I convert every line to lower case and replace all characters except the ones in alphabet and the apostrophe(') with a whitespace.I chose to keep the apostrophe, since it is used commonly in English words. After these operations, I split the lines by whitespace to extract words.For every word in a line I check if it is already in the dictionary.If not, I add it to dictionary and set its value as 1.If it is already in the dictionary, I increment its value by 1.

```java
public HashMap<String,Integer> getDictionaryFromFile(String filepath) throws IOException{

      File file = new File(filepath);
      HashMap<String, Integer> dictionary = new HashMap<String, Integer>();
      Vector<String> lines=Main.readFile(filepath);
      for(String line:lines) {
        line=line.toLowerCase();        //convert all words to lowercase
        //replace any character in corpus,which is not apostrophe or not in alphabet, with a whitespace.
        String b=line.replaceAll("[^a-zA-Z']", " ");
        String[] words=b.split("\\s+");     // get all words in corpus
        //put words in hashmap and calculate their occurrences
        for (String word:words) {

            if(dictionary.containsKey(word)) {
                dictionary.replace(word, dictionary.get(word)+1);
            }else {
                dictionary.put(word,1);
            }
        }

      }
      return dictionary;
}
```

Figure 3: Creating dictionary from corpus file

### 1.2.2 Parsing spell-errors

I parse the "spell-errors.txt" file similarly to create a hashmap where keys are the correct versions of a word and the values are hashmaps again.These value hashmap has the misspelled versions of a word as keys and number of times they had been misspelled as values.This spell-errors hashmap is used to create confusion matrices.

```java
public static HashMap<String,HashMap<String,Integer>> parseSpellErrors(String filepath) throws IOException{

      HashMap<String,HashMap<String,Integer>> errors=new HashMap<String,HashMap<String,Integer>>();
      File file = new File(filepath);
      Vector<String> lines=Main.readFile(filepath);
      for(String line:lines) {

            line=line.toLowerCase();
            String b=line.replaceAll("\\s+","");
            String[] words=b.split(":");
            if(words.length>1) {
                String errs=words[1];
                String[] errlist=errs.split(",");

                HashMap<String,Integer> errmap=new HashMap<String,Integer>();
                for(String err:errlist) {
                    String[]spl=err.split("\\*");
                    if(spl.length>1) {
                        Integer a=Integer.parseInt(spl[1]);
                        errmap.put(spl[0],a);

                    }else {
                        errmap.put(spl[0],1);
                    }
                }
                errors.put(words[0],errmap);
            }
      }

    return errors;
}
```

Figure 4: Parse spell errors

### 1.2.3 Producing confusion matrices

We have 4 confusion matrices:ins(insertion),del(deletion),rep(substitution),tr(transposition).For every key in "spellErrors" hashmap which was created before, we get the a misspelled version of it.We create a Matrix_calc object to extract the operations needed to convert the correct version to the misspelled version.We iterate through these operations and get their operation name and operands.We increase the value in the confusion matrix according to the operation.FOr example, if operation is ins(a,b), we increase the matrix cell value "ins[a][b]" by the number of times this misspelled version had been used.

```
public void produceConfusionMatrices(){

    for(String s:spellErrors.keySet()) {
        HashMap<String,Integer> map=spellErrors.get(s);
        for(String k:map.keySet()) {
            int count=map.get(k);
            Matrix_calc m=new Matrix_calc(k,s);
            Vector<Operation> ops=m.calculateEdits();
            for(Operation op:ops) {
                String opname=op.opname;
                String oper1=op.op1.toLowerCase();
                String oper2=op.op2.toLowerCase();
                int i1=alphabet.indexOf(oper1);
                int i2=alphabet.indexOf(oper2);
                int j1=alphabet2.indexOf(oper1);
                int j2=alphabet2.indexOf(oper2);
                if(opname.equals("ins")&&j1!=-1&&j2!=-1) {

                    ins[j2][j1]+=count;
                }else if(opname.equals("del")&&j1!=-1&&j2!=-1) {

                    del[j1][j2]+=count;
                }else if(opname.equals("rep")&&i1!=-1&&i2!=-1) {

                    rep[i2][i1]+=count;
                }else if(opname.equals("tr")&&i1!=-1&&i2!=-1) {

                    tr[i1][i2]+=count;
                }
            }


        }
    }


}
```

Figure 5: Produce confusion matrices

#### 1.2.4 Correcting a misspelled word

Now that we have our dictionary and confusion matrices,we can start correcting a misspelled word.First we have to find candidates for a given word in the dictionary. For this purpose, I generate all words which have an edit distance of 1 with the word and get the ones that are in my dictionary.After that I calculate the probability P(W)P(X$\|W)foreverycandidateandgettheonewiththemaximumprobability.Theprobabilitycalcul$

```java
public static Set<String> generateWords(String s){
    String alphabet="abcdefghijklmnopqrstuvwxyz";
    StringBuilder sb=new StringBuilder(s);
    Set<String> vec = new HashSet<String>();
    for(int i=0;i<=s.length();i++) {
            if(i!=s.length()) {
                //deletion
                sb=new StringBuilder(s);
                String del=sb.deleteCharAt(i).toString();
                vec.add(del);
                //transposition
                if(i!=s.length()-1) {
                    sb=new StringBuilder(s);
                    String s1=Character.toString(s.charAt(i));
                    String s2=Character.toString(s.charAt(i+1));
                    if(!s1.equals(s2)) {
                        String replace=s2+s1;
                        sb.replace(i, i+2, replace);
                        vec.add(sb.toString());
                    }
                }
            }
            for(int j=0;j<alphabet.length();j++) {
                //insertion
                sb=new StringBuilder(s);
                char c=alphabet.charAt(j);
                String ins=sb.insert(i,c).toString();
                vec.add(ins);

                //substitution
                sb=new StringBuilder(s);
                if(i<s.length()) {
                    if(c!=s.charAt(i)) {
                        String subs=sb.replace(i,i+1,Character.toString(c)).toString();
                        vec.add(subs);
                    }

                }
            }
    }
    return vec;
```

Figure 6: Generate all words that have an edit distance of 1 with a given word

```java
//Find words in dictionary which have an edit distance of 1 with a given string "errorWord"
public static Vector<String> findCandidatesInCorpus(HashMap<String,Integer> dictionary,String errorWord) {
    Vector<String> vec=new Vector<String>();
    for(String s:generateWords(errorWord)) {
        if(dictionary.containsKey(s)) {
            vec.add(s);
        }
    }
    return vec;
}
```

Figure 7: Get candidates for a misspelled word from dictionary

```java
//returns the correectWord for a given misspelled word using dictionary and confusion matrices
public String correctWord(String word) {
    double maxProb=0;
    String bestCorrection="";

    //calculate probability p(w)(px|w) for all candidates and return the one with the highest probability
    for(String s:findCandidatesInCorpus(dictionary,word)) {
        Vector<Operation> ops=new Matrix_calc(word,s).calculateEdits();
        double prob=0;
        for(Operation op:ops) {
            if(!op.opname.equals("copy")) {
                prob=calculateProbability(s,op.opname,op.op1,op.op2);
                break;
            }

        }
        if(prob>maxProb) {
            maxProb=prob;
            bestCorrection=s;
        }
    }
    return bestCorrection;
}
```

Figure 8: Return the correct version of a misspelled word

# 2 Confusion Matrices

| | # | ' | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # | 0 | 6 | 115 | 17 | 20 | 12 | 77 | 15 | 12 | 61 | 48 | 2 | 17 | 22 | 9 | 29 | 40 | 30 | 2 | 34 | 51 | 44 | 19 | 6 | 22 | 2 | 15 | 0 |
| ' | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 7 | 0 | 10 | 88 | 70 | 34 | 4 | 16 | 7 | 116 | 0 | 9 | 135 | 30 | 123 | 3 | 33 | 0 | 160 | 88 | 83 | 52 | 6 | 4 | 4 | 23 | 5 |
| b | 0 | 0 | 37 | 0 | 1 | 0 | 54 | 0 | 0 | 2 | 15 | 1 | 0 | 9 | 0 | 0 | 14 | 0 | 0 | 15 | 7 | 6 | 25 | 0 | 1 | 0 | 2 | 0 |
| c | 0 | 1 | 24 | 0 | 0 | 0 | 104 | 1 | 0 | 53 | 67 | 0 | 101 | 19 | 5 | 7 | 97 | 4 | 5 | 10 | 28 | 64 | 19 | 2 | 0 | 2 | 7 | 0 |
| d | 0 | 2 | 17 | 0 | 0 | 0 | 137 | 2 | 8 | 0 | 68 | 1 | 0 | 7 | 1 | 14 | 34 | 0 | 0 | 13 | 8 | 10 | 6 | 0 | 2 | 0 | 3 | 0 |
| e | 0 | 18 | 196 | 1 | 122 | 163 | 0 | 46 | 22 | 7 | 93 | 2 | 6 | 105 | 50 | 144 | 41 | 24 | 8 | 208 | 217 | 69 | 18 | 8 | 3 | 20 | 38 | 2 |
| f | 0 | 0 | 36 | 0 | 0 | 0 | 78 | 0 | 0 | 0 | 25 | 0 | 0 | 6 | 0 | 3 | 33 | 0 | 0 | 19 | 0 | 7 | 16 | 3 | 0 | 0 | 0 | 0 |
| g | 0 | 1 | 10 | 0 | 1 | 8 | 70 | 1 | 0 | 9 | 31 | 0 | 1 | 9 | 2 | 4 | 14 | 0 | 2 | 11 | 10 | 9 | 24 | 0 | 1 | 0 | 2 | 2 |
| h | 0 | 1 | 27 | 0 | 3 | 1 | 115 | 2 | 21 | 0 | 45 | 0 | 0 | 9 | 5 | 5 | 47 | 1 | 0 | 22 | 4 | 29 | 11 | 1 | 1 | 0 | 12 | 0 |
| i | 0 | 1 | 95 | 3 | 58 | 40 | 149 | 21 | 24 | 13 | 0 | 1 | 9 | 96 | 30 | 141 | 91 | 7 | 4 | 26 | 166 | 81 | 18 | 10 | 0 | 1 | 6 | 25 |
| j | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| k | 0 | 1 | 0 | 0 | 2 | 6 | 40 | 0 | 1 | 0 | 5 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 6 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| l | 0 | 0 | 52 | 0 | 4 | 4 | 237 | 2 | 5 | 1 | 109 | 0 | 1 | 0 | 2 | 8 | 32 | 3 | 0 | 9 | 13 | 15 | 10 | 2 | 4 | 0 | 38 | 0 |
| m | 0 | 1 | 47 | 9 | 0 | 6 | 145 | 0 | 3 | 3 | 53 | 0 | 2 | 0 | 0 | 42 | 26 | 1 | 0 | 11 | 9 | 6 | 15 | 0 | 0 | 1 | 2 | 0 |
| n | 0 | 14 | 49 | 2 | 76 | 88 | 267 | 6 | 60 | 11 | 131 | 0 | 9 | 19 | 14 | 0 | 24 | 2 | 0 | 12 | 83 | 141 | 28 | 3 | 2 | 0 | 6 | 3 |
| o | 0 | 0 | 48 | 4 | 33 | 7 | 43 | 14 | 13 | 10 | 20 | 1 | 2 | 39 | 60 | 63 | 0 | 24 | 0 | 85 | 43 | 21 | 176 | 4 | 59 | 1 | 4 | 0 |
| p | 0 | 0 | 22 | 0 | 6 | 0 | 104 | 0 | 0 | 53 | 40 | 0 | 0 | 16 | 1 | 3 | 16 | 0 | 0 | 30 | 5 | 23 | 8 | 0 | 0 | 0 | 2 | 0 |
| q | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 |
| r | 0 | 6 | 60 | 8 | 27 | 30 | 315 | 1 | 2 | 5 | 125 | 0 | 0 | 30 | 8 | 29 | 48 | 1 | 0 | 0 | 34 | 34 | 20 | 5 | 3 | 1 | 29 | 0 |
| s | 0 | 8 | 50 | 0 | 48 | 6 | 254 | 0 | 3 | 60 | 113 | 0 | 1 | 21 | 6 | 21 | 40 | 2 | 0 | 9 | 0 | 59 | 56 | 2 | 1 | 0 | 13 | 2 |
| t | 0 | 7 | 87 | 0 | 19 | 15 | 376 | 0 | 1 | 63 | 146 | 1 | 0 | 24 | 5 | 21 | 62 | 0 | 1 | 54 | 42 | 0 | 24 | 3 | 19 | 0 | 11 | 3 |
| u | 0 | 0 | 50 | 2 | 24 | 9 | 59 | 1 | 11 | 6 | 40 | 0 | 3 | 133 | 10 | 23 | 13 | 6 | 2 | 83 | 31 | 20 | 0 | 1 | 2 | 0 | 0 | 0 |
| v | 0 | 0 | 8 | 1 | 1 | 6 | 67 | 1 | 1 | 0 | 55 | 0 | 0 | 0 | 0 | 3 | 7 | 0 | 0 | 4 | 0 | 2 | 3 | 0 | 0 | 0 | 4 | 0 |
| w | 0 | 0 | 6 | 0 | 0 | 1 | 38 | 0 | 0 | 54 | 6 | 0 | 0 | 2 | 0 | 6 | 5 | 0 | 0 | 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| x | 0 | 0 | 1 | 0 | 25 | 0 | 9 | 0 | 1 | 8 | 7 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 50 | 4 | 0 | 0 | 0 | 0 | 0 | 4 |
| y | 0 | 8 | 3 | 0 | 0 | 1 | 23 | 0 | 4 | 0 | 11 | 0 | 0 | 1 | 2 | 7 | 1 | 1 | 0 | 1 | 17 | 1 | 2 | 0 | 1 | 0 | 0 | 0 |
| z | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 9: Insertion("#" represents beginning of string)

| | # | ' | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # | 0 | 2 | 185 | 48 | 117 | 33 | 143 | 42 | 29 | 78 | 151 | 9 | 58 | 19 | 38 | 60 | 69 | 219 | 3 | 94 | 136 | 78 | 43 | 31 | 63 | 2 | 11 | 0 |
| ' | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 2 | 0 | 52 | 361 | 70 | 89 | 99 | 120 | 16 | 301 | 2 | 7 | 473 | 63 | 392 | 48 | 254 | 4 | 252 | 153 | 245 | 183 | 19 | 4 | 4 | 28 | 0 |
| b | 0 | 0 | 64 | 0 | 6 | 6 | 133 | 0 | 9 | 3 | 40 | 0 | 0 | 28 | 0 | 8 | 29 | 0 | 0 | 53 | 9 | 22 | 28 | 2 | 0 | 0 | 3 | 0 |
| c | 0 | 0 | 118 | 1 | 0 | 7 | 188 | 19 | 3 | 281 | 292 | 0 | 47 | 66 | 26 | 36 | 230 | 4 | 1 | 60 | 64 | 118 | 73 | 2 | 2 | 0 | 7 | 0 |
| d | 0 | 6 | 71 | 1 | 5 | 0 | 151 | 4 | 29 | 1 | 136 | 2 | 1 | 20 | 6 | 29 | 26 | 5 | 1 | 36 | 33 | 34 | 20 | 4 | 2 | 0 | 18 | 0 |
| e | 0 | 21 | 423 | 7 | 233 | 367 | 0 | 35 | 31 | 25 | 211 | 28 | 0 | 202 | 70 | 364 | 90 | 21 | 47 | 319 | 390 | 187 | 120 | 10 | 5 | 19 | 52 | 6 |
| f | 0 | 0 | 62 | 3 | 62 | 1 | 75 | 0 | 10 | 4 | 161 | 0 | 0 | 48 | 2 | 18 | 47 | 0 | 0 | 125 | 2 | 40 | 28 | 0 | 0 | 0 | 6 | 0 |
| g | 0 | 1 | 83 | 5 | 5 | 10 | 191 | 2 | 0 | 39 | 71 | 0 | 0 | 20 | 8 | 62 | 20 | 0 | 0 | 72 | 19 | 14 | 186 | 1 | 0 | 0 | 9 | 1 |
| h | 0 | 2 | 54 | 0 | 1 | 49 | 238 | 0 | 5 | 0 | 61 | 0 | 0 | 29 | 7 | 17 | 70 | 2 | 0 | 38 | 33 | 51 | 33 | 1 | 0 | 0 | 13 | 0 |
| i | 0 | 1 | 180 | 10 | 158 | 43 | 199 | 41 | 176 | 70 | 0 | 0 | 2 | 117 | 106 | 263 | 141 | 65 | 3 | 46 | 220 | 194 | 31 | 19 | 1 | 0 | 3 | 7 |
| j | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 15 | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 0 | 0 | 0 | 0 |
| k | 0 | 6 | 0 | 0 | 0 | 36 | 94 | 0 | 7 | 0 | 14 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 0 | 1 | 22 | 2 | 0 | 0 | 0 | 0 | 1 | 0 |
| l | 0 | 4 | 83 | 0 | 17 | 42 | 358 | 5 | 29 | 6 | 200 | 0 | 0 | 0 | 4 | 34 | 128 | 23 | 0 | 18 | 38 | 25 | 10 | 23 | 2 | 0 | 89 | 2 |
| m | 0 | 7 | 138 | 44 | 18 | 26 | 236 | 6 | 12 | 8 | 120 | 0 | 1 | 10 | 0 | 112 | 71 | 73 | 0 | 28 | 27 | 34 | 26 | 5 | 1 | 0 | 4 | 0 |
| n | 0 | 79 | 254 | 0 | 195 | 197 | 547 | 15 | 124 | 15 | 343 | 5 | 3 | 59 | 23 | 0 | 101 | 5 | 7 | 15 | 284 | 349 | 56 | 19 | 4 | 27 | 32 | 0 |
| o | 0 | 12 | 92 | 23 | 58 | 39 | 102 | 23 | 74 | 19 | 72 | 0 | 10 | 105 | 187 | 163 | 0 | 123 | 0 | 186 | 61 | 95 | 382 | 22 | 111 | 15 | 36 | 2 |
| p | 0 | 1 | 112 | 0 | 56 | 19 | 240 | 1 | 1 | 75 | 90 | 0 | 2 | 79 | 3 | 44 | 78 | 0 | 0 | 189 | 11 | 76 | 13 | 1 | 0 | 1 | 6 | 0 |
| q | 0 | 0 | 2 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 26 | 0 | 0 | 0 | 0 | 0 |
| r | 0 | 42 | 299 | 4 | 79 | 48 | 667 | 13 | 43 | 68 | 354 | 3 | 3 | 72 | 49 | 111 | 154 | 11 | 8 | 0 | 93 | 144 | 75 | 7 | 0 | 1 | 36 | 2 |
| s | 0 | 6 | 129 | 2 | 394 | 14 | 341 | 16 | 11 | 119 | 299 | 0 | 21 | 30 | 6 | 45 | 109 | 73 | 26 | 73 | 0 | 243 | 173 | 1 | 49 | 0 | 42 | 0 |
| t | 0 | 48 | 288 | 17 | 54 | 31 | 890 | 12 | 15 | 66 | 390 | 0 | 1 | 111 | 11 | 112 | 158 | 1 | 3 | 194 | 155 | 0 | 115 | 11 | 9 | 0 | 32 | 0 |
| u | 0 | 12 | 119 | 28 | 93 | 29 | 68 | 28 | 42 | 20 | 169 | 0 | 0 | 108 | 25 | 105 | 18 | 44 | 2 | 152 | 84 | 61 | 0 | 3 | 0 | 0 | 0 | 1 |
| v | 0 | 0 | 50 | 1 | 6 | 5 | 152 | 0 | 9 | 0 | 60 | 0 | 0 | 11 | 0 | 54 | 23 | 0 | 0 | 7 | 3 | 2 | 3 | 0 | 1 | 0 | 1 | 0 |
| w | 0 | 1 | 26 | 0 | 1 | 11 | 54 | 0 | 2 | 112 | 21 | 0 | 0 | 15 | 3 | 6 | 21 | 0 | 0 | 13 | 8 | 4 | 2 | 1 | 0 | 0 | 1 | 0 |
| x | 0 | 0 | 16 | 5 | 38 | 2 | 19 | 0 | 12 | 53 | 32 | 0 | 0 | 1 | 2 | 1 | 1 | 9 | 4 | 1 | 5 | 8 | 1 | 0 | 0 | 0 | 1 | 0 |
| y | 0 | 11 | 5 | 0 | 18 | 7 | 35 | 0 | 8 | 5 | 11 | 0 | 0 | 2 | 21 | 9 | 5 | 2 | 0 | 7 | 28 | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| z | 0 | 0 | 0 | 0 | 0 | 1 | 6 | 0 | 1 | 0 | 3 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 10: Deletion("#" represents beginning of string)

| | ' | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ' | 0 | 2 | 0 | 8 | 1 | 24 | 0 | 0 | 0 | 12 | 0 | 0 | 3 | 0 | 30 | 3 | 0 | 0 | 3 | 1 | 9 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 1 | 0 | 24 | 103 | 56 | 1720 | 28 | 28 | 74 | 746 | 1 | 32 | 194 | 37 | 138 | 795 | 29 | 3 | 416 | 143 | 99 | 367 | 17 | 16 | 3 | 38 | 7 |
| b | 0 | 15 | 0 | 2 | 82 | 42 | 5 | 6 | 4 | 10 | 0 | 1 | 23 | 12 | 14 | 8 | 86 | 0 | 15 | 2 | 13 | 15 | 19 | 2 | 0 | 0 | 0 |
| c | 1 | 132 | 4 | 0 | 31 | 262 | 24 | 68 | 67 | 98 | 0 | 170 | 64 | 16 | 74 | 61 | 34 | 52 | 75 | 1252 | 333 | 126 | 20 | 8 | 85 | 21 | 14 |
| d | 0 | 75 | 90 | 18 | 0 | 163 | 6 | 106 | 28 | 38 | 15 | 16 | 81 | 19 | 85 | 23 | 15 | 0 | 78 | 84 | 426 | 13 | 10 | 8 | 1 | 32 | 2 |
| e | 27 | 1710 | 37 | 93 | 167 | 0 | 92 | 77 | 118 | 1722 | 7 | 73 | 381 | 74 | 242 | 504 | 47 | 2 | 364 | 635 | 465 | 483 | 31 | 37 | 6 | 262 | 13 |
| f | 0 | 33 | 5 | 24 | 10 | 78 | 0 | 7 | 67 | 16 | 0 | 0 | 18 | 6 | 19 | 17 | 43 | 2 | 77 | 24 | 48 | 9 | 118 | 4 | 7 | 5 | 0 |
| g | 0 | 70 | 9 | 121 | 108 | 132 | 23 | 0 | 27 | 38 | 34 | 37 | 35 | 12 | 63 | 39 | 9 | 19 | 58 | 40 | 50 | 21 | 10 | 3 | 1 | 42 | 8 |
| h | 0 | 83 | 5 | 55 | 27 | 134 | 181 | 25 | 0 | 69 | 4 | 86 | 54 | 8 | 44 | 74 | 17 | 3 | 82 | 50 | 149 | 43 | 24 | 28 | 3 | 14 | 0 |
| i | 4 | 1145 | 12 | 157 | 56 | 2252 | 47 | 52 | 270 | 0 | 2 | 10 | 217 | 72 | 255 | 346 | 28 | 5 | 316 | 300 | 210 | 321 | 19 | 11 | 4 | 244 | 21 |
| j | 0 | 2 | 0 | 8 | 56 | 3 | 0 | 75 | 1 | 0 | 0 | 0 | 3 | 2 | 0 | 2 | 1 | 1 | 4 | 0 | 2 | 2 | 0 | 0 | 0 | 4 | 0 |
| k | 0 | 9 | 1 | 97 | 7 | 42 | 1 | 22 | 14 | 3 | 0 | 0 | 8 | 2 | 9 | 16 | 0 | 2 | 10 | 6 | 22 | 2 | 0 | 4 | 1 | 1 | 0 |
| l | 0 | 293 | 15 | 61 | 55 | 445 | 28 | 18 | 37 | 188 | 1 | 20 | 0 | 15 | 107 | 143 | 17 | 5 | 265 | 50 | 160 | 128 | 18 | 15 | 2 | 36 | 3 |
| m | 0 | 81 | 11 | 19 | 26 | 155 | 6 | 9 | 15 | 61 | 0 | 3 | 22 | 0 | 573 | 62 | 20 | 0 | 46 | 17 | 35 | 36 | 6 | 9 | 5 | 13 | 1 |
| n | 1 | 166 | 8 | 60 | 108 | 386 | 14 | 43 | 76 | 168 | 0 | 10 | 85 | 395 | 0 | 124 | 22 | 2 | 153 | 73 | 171 | 104 | 12 | 20 | 9 | 21 | 1 |
| o | 0 | 688 | 13 | 84 | 36 | 643 | 10 | 34 | 65 | 288 | 5 | 31 | 103 | 28 | 135 | 0 | 20 | 4 | 180 | 61 | 45 | 375 | 12 | 19 | 4 | 21 | 3 |
| p | 0 | 59 | 70 | 41 | 7 | 77 | 59 | 7 | 28 | 61 | 0 | 15 | 12 | 22 | 33 | 23 | 0 | 0 | 40 | 34 | 83 | 18 | 7 | 0 | 4 | 11 | 0 |
| q | 0 | 7 | 0 | 79 | 1 | 7 | 0 | 24 | 5 | 5 | 0 | 22 | 1 | 1 | 1 | 9 | 7 | 0 | 1 | 2 | 8 | 1 | 0 | 0 | 0 | 0 | 3 |
| r | 7 | 258 | 13 | 48 | 51 | 397 | 21 | 29 | 72 | 183 | 2 | 12 | 117 | 25 | 143 | 145 | 29 | 1 | 0 | 77 | 99 | 160 | 16 | 36 | 3 | 53 | 2 |
| s | 1 | 106 | 6 | 524 | 40 | 394 | 17 | 26 | 80 | 119 | 0 | 13 | 45 | 32 | 92 | 55 | 20 | 0 | 114 | 0 | 182 | 76 | 7 | 19 | 45 | 87 | 90 |
| t | 6 | 142 | 10 | 263 | 330 | 434 | 54 | 52 | 157 | 148 | 6 | 47 | 151 | 39 | 244 | 63 | 40 | 4 | 150 | 345 | 0 | 42 | 12 | 5 | 9 | 29 | 4 |
| u | 2 | 460 | 5 | 140 | 34 | 674 | 17 | 63 | 89 | 400 | 1 | 42 | 104 | 30 | 84 | 486 | 17 | 7 | 243 | 152 | 70 | 0 | 13 | 157 | 2 | 39 | 1 |
| v | 0 | 15 | 19 | 6 | 20 | 17 | 82 | 1 | 8 | 6 | 1 | 0 | 14 | 5 | 11 | 11 | 0 | 0 | 19 | 11 | 28 | 12 | 0 | 11 | 0 | 2 | 0 |
| w | 0 | 14 | 2 | 8 | 12 | 31 | 4 | 2 | 19 | 7 | 0 | 2 | 28 | 9 | 24 | 22 | 0 | 0 | 42 | 17 | 9 | 70 | 5 | 0 | 0 | 7 | 0 |
| x | 0 | 10 | 0 | 51 | 1 | 6 | 1 | 16 | 1 | 4 | 0 | 6 | 0 | 10 | 3 | 3 | 3 | 0 | 9 | 68 | 6 | 4 | 3 | 0 | 0 | 1 | 14 |
| y | 1 | 56 | 3 | 27 | 34 | 558 | 2 | 52 | 37 | 276 | 0 | 4 | 44 | 2 | 43 | 18 | 6 | 0 | 86 | 76 | 53 | 30 | 2 | 4 | 2 | 0 | 3 |
| z | 0 | 0 | 0 | 9 | 5 | 4 | 0 | 2 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 95 | 1 | 0 | 1 | 0 | 2 | 0 | 0 |

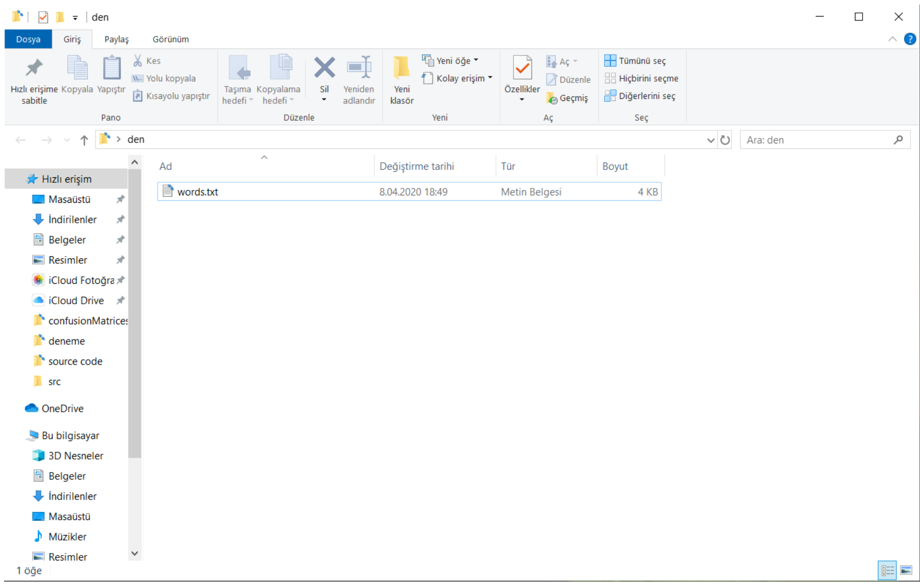
Figure 11: Substitution

| | ' | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ' | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 0 | 0 | 5 | 2 | 25 | 0 | 6 | 9 | 26 | 0 | 0 | 12 | 9 | 16 | 7 | 0 | 0 | 20 | 1 | 9 | 37 | 0 | 0 | 0 | 0 | 0 |
| b | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| c | 0 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 15 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 7 | 9 | 2 | 0 | 0 | 0 | 6 | 0 |
| d | 0 | 0 | 0 | 0 | 0 | 39 | 0 | 0 | 0 | 4 | 0 | 0 | 7 | 0 | 7 | 0 | 0 | 0 | 1 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| e | 0 | 6 | 0 | 3 | 17 | 0 | 5 | 6 | 9 | 118 | 0 | 5 | 100 | 11 | 19 | 6 | 6 | 0 | 125 | 30 | 29 | 4 | 3 | 0 | 0 | 4 | 0 |
| f | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| g | 0 | 3 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 1 | 0 | 6 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| h | 0 | 0 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 16 | 0 | 0 | 6 | 1 | 0 | 0 |
| i | 0 | 48 | 0 | 23 | 6 | 79 | 0 | 8 | 4 | 0 | 0 | 0 | 13 | 10 | 23 | 11 | 1 | 0 | 32 | 11 | 15 | 16 | 0 | 2 | 2 | 2 | 0 |
| j | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| k | 0 | 7 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| l | 0 | 14 | 5 | 4 | 1 | 72 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 0 | 1 | 15 | 5 | 0 | 0 | 0 | 4 | 4 | 0 | 1 | 0 | 1 | 0 |
| m | 0 | 2 | 0 | 0 | 0 | 19 | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 1 | 4 | 0 | 0 | 2 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| n | 0 | 17 | 0 | 0 | 0 | 20 | 0 | 12 | 0 | 15 | 0 | 2 | 0 | 2 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 5 | 0 | 5 | 0 | 0 | 0 |
| o | 0 | 3 | 1 | 2 | 1 | 4 | 0 | 1 | 6 | 24 | 0 | 0 | 13 | 5 | 10 | 0 | 2 | 0 | 28 | 1 | 4 | 7 | 0 | 4 | 0 | 0 | 0 |
| p | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 7 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| q | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 |
| r | 0 | 25 | 1 | 1 | 0 | 88 | 0 | 2 | 0 | 21 | 0 | 0 | 0 | 0 | 0 | 30 | 0 | 0 | 0 | 0 | 5 | 16 | 0 | 1 | 0 | 0 | 0 |
| s | 5 | 2 | 0 | 0 | 0 | 43 | 0 | 0 | 0 | 19 | 0 | 0 | 4 | 1 | 2 | 4 | 4 | 0 | 5 | 0 | 5 | 2 | 0 | 0 | 0 | 4 | 0 |
| t | 0 | 10 | 0 | 4 | 0 | 17 | 1 | 0 | 26 | 17 | 0 | 0 | 5 | 0 | 4 | 3 | 0 | 0 | 1 | 9 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| u | 0 | 12 | 0 | 5 | 8 | 17 | 1 | 4 | 1 | 4 | 0 | 0 | 9 | 1 | 1 | 13 | 1 | 1 | 11 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| v | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| w | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| y | 0 | 5 | 0 | 2 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 3 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| z | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

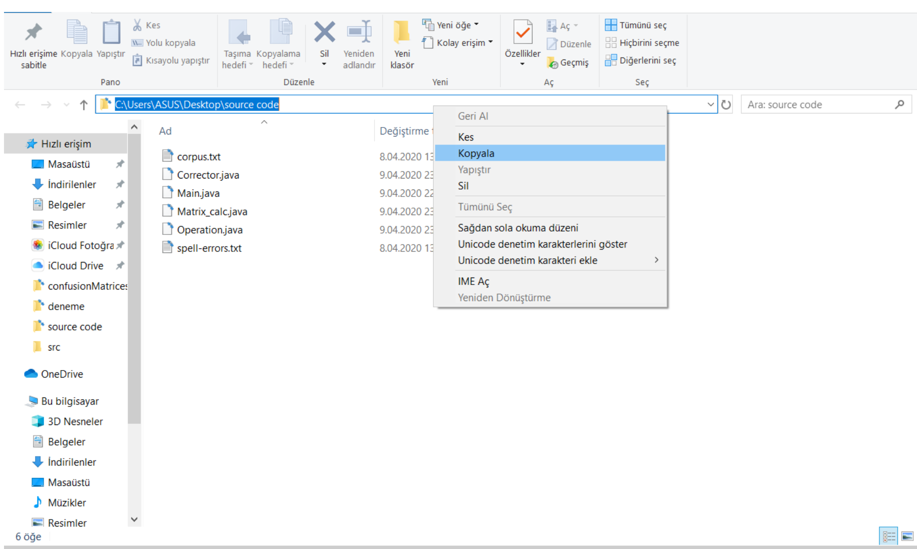
Figure 12: Transposition

# 3 Screenshots of Running the System

1. Put the input file in a folder after naming it ”words.txt”.

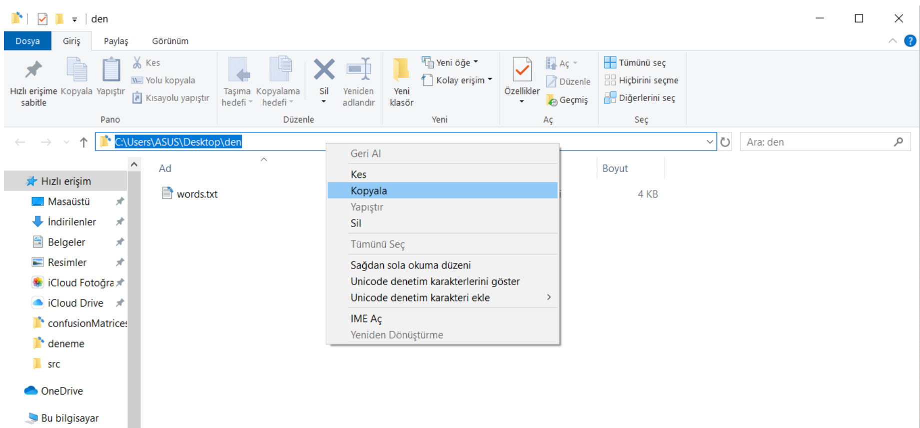



2. Open the ”source code” folder and copy the path(corpus.txt and spell-errors.txt files should be in this folder.If not,please copy them here).

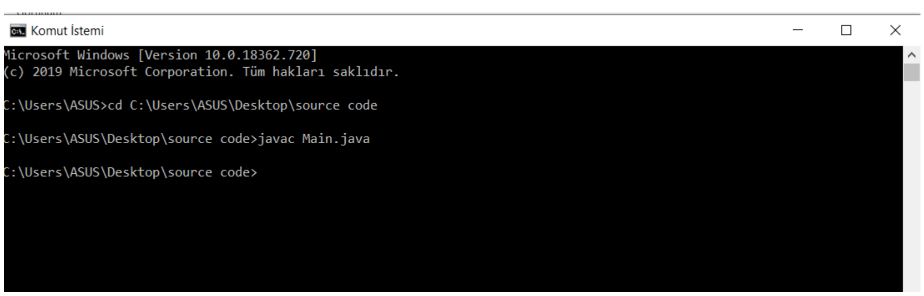


3. Open command line and "cd" into the source code directory.

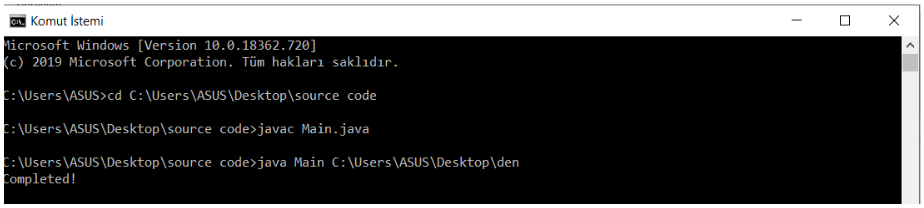

4. Open the folder of the input file and copy the path.



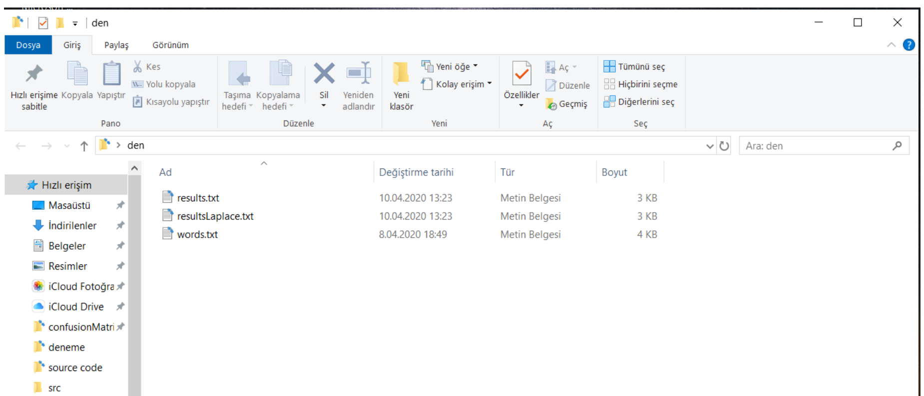

5. Open command line again and enter command "javac Main.java"



6. Then, enter command ”java Main $< inputfilefolderpath >$”.”Completed” will be printed if program finishes successfully.




7. Output files will be created in the input file folder.



# 4 Accuracy Scores

```
 RESULTS
Number of true corrections:294
Number of false corrections:20
Number of no corrections:70

Number of no corrections when the true correction is not in dictionary:49
Number of no corrections when the true correction has an edit distance higher than '1':21

Number of wrong corrections when the true correction is not in dictionary:7
Number of wrong corrections when the true correction has an edit distance higher than '1':1

Number of times the true correction is not in dictionary:56
Number of times the true correction has an edit distance higher than '1':22
---------------------------------
Laplace RESULTS
Number of true corrections:294
Number of false corrections:21
Number of no corrections:69

Number of no corrections when the true correction is not in dictionary:48
Number of no corrections when the true correction has an edit distance higher than '1':21

Number of wrong corrections when the true correction is not in dictionary:8
Number of wrong corrections when the true correction has an edit distance higher than '1':1

Number of times the true correction is not in dictionary:56
Number of times the true correction has an edit distance higher than '1':22
---------------------------------
```

# 5 Suggested Improvements

Most of the correct versions of the misspelled words that my error corrector could not correct was not in the corpus.With a bigger corpus(as demonstrated in "Accuracy Scores" section), the results would be much better. Also,if spell errors data was bigger, the confusion matrices would be more accurate.Then, the calculated probabilities would be more accurate too and ,as a result, the corrections would be better.