Minimum Edit Distance(MED) Algorithm

I. Introduction

Minimum edit distance (MED) or Levenshtein algorithm is used to rate the similarity between two sequences. In practice it is used to rate the similarity between words in search results.

In this study, a Levenshtein algorithm was implemented in Java. The program takes input as text and lists all words written incorrect with their first 5 alternative correct. It uses a text file as a sample word pool to perform these operations. The program outputs were analyzed separately according to the total running time and words - alternative correct words list for 20 words.

II. Definition of the Algorithm

A. *Methods, Inputs, and Outputs of the Program*

The following methods were used in the program:

- distance method- MED algorithm was used in this method. The actual process. Comparing is done by this method. Distance method is hearth of the code [1].
- main method –
  - ✓ Taking inputs,
  - ✓ File operations, [2]
  - ✓ Calculating total running time, [3]

  are included in this method.
- Inputs- Any text.
- Outputs- Total running time and all words written incorrect with their first 5 alternative correct words.

III. Screenshots of the Program

The program screenshots are shown in the following.

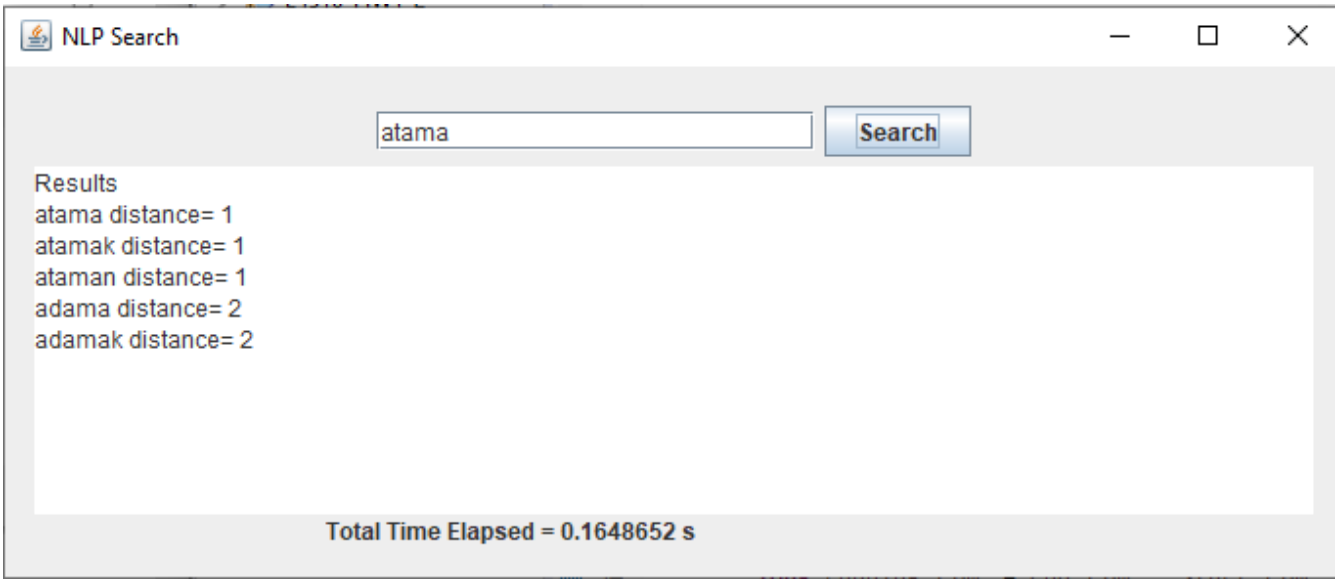

Figure 1: Outputs and total running time for the test word “atama”.

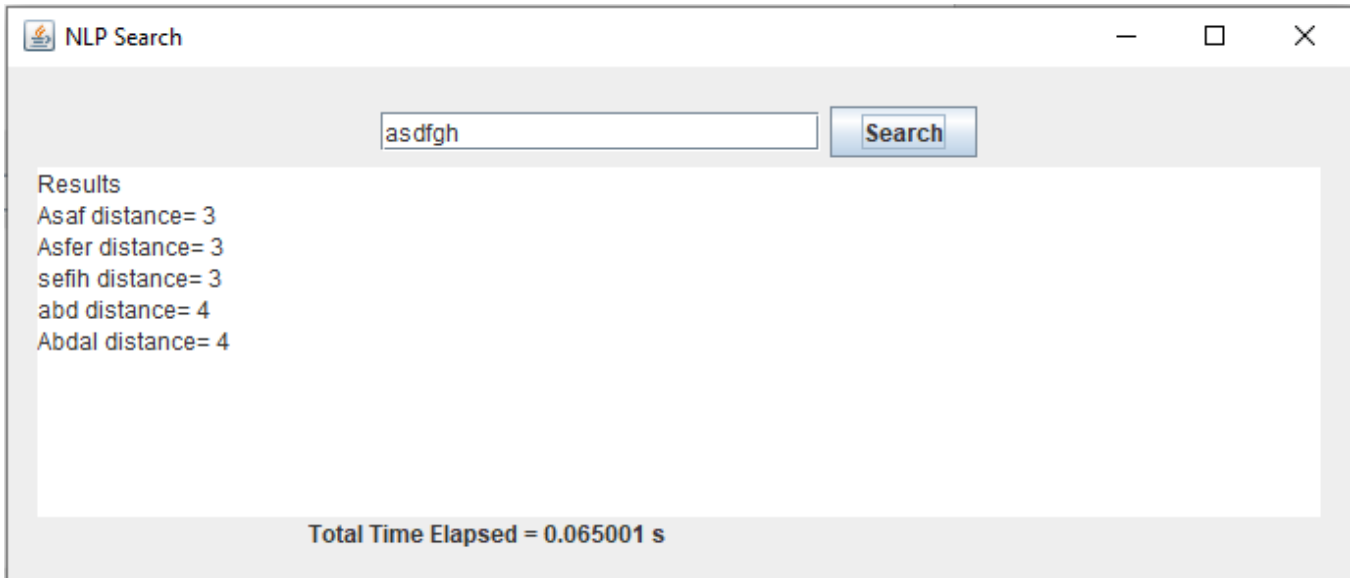

Figure 2: Outputs and total running time for a test word that generated randomly.

IV. TEST RESULTS

Table 1: Test words, their alternatives and total running times for each word.

Test words	**Alternative words for test word**	**Total running time(s)**
çocukluk	bozukluk distance= 2 çabukluk distance= 2 çocukçu distance= 2 çocuklar distance= 2 çoğunluk distance= 2	0.0202
çcklk	çıkak distance= 2 çıkık distance= 2 cılk distance= 2 çekik distance= 2 çökek distance= 2	0.0163
rabıta	rabıta distance= 0 rabıt distance= 1 zabıta distance= 1 rabıtalı distance= 2 rasıt distance= 2	0.0169
rabı	rab distance= 1 rabıt distance= 1 rakı distance= 1 rakı distance= 1 araba distance= 2	0.0154
radikallşmek	radikalleşmek distance= 1 adileşmek distance= 4 dikleşmek distance= 4 kadifeleşmek distance= 4 radikalizm distance= 4	0.0225
radikllşmk	radikalleşmek distance= 3 adileşmek distance= 4 dikleşmek distance= 4 radikal distance= 4 radikalizm distance= 4	0.0209
meyusiyet	meyusiyet distance= 0 memuriyet distance= 2 mesuliyet distance= 2 mezuniyet distance= 2	0.0254

	haysiyet distance= 3	
myusiytek	meyusiyet distance= 3 maksi etek distance= 4 mersiye distance= 4 meyus etmek distance= 4 musibet distance= 4	0.0200
mezcetmek	mezcetmek distance= 0 bezetmek distance= 2 cezbetmek distance= 2 dercetmek distance= 2 menetmek distance= 2	0.0218
mezctmek	mezcetmek distance= 1 bezetmek distance= 2 menetmek distance= 2 mercimek distance= 2 ahzetmek distance= 3	0.0237
mıhlamak	mıhlamak distance= 0 mıhlanmak distance= 1 çınlamak distance= 2 cızlamak distance= 2 dışlamak distance= 2	0.0335
mıhlak	ahlak distance= 2 çıplak distance= 2 ıtlak distance= 2 dılak distance= 2 fırlak distance= 2	0.0176
istekle	istekle distance= 0 istekli distance= 1 iskemle distance= 2 istek distance= 2 isteka distance= 2	0.0256
isteklekle	içtenlikle distance= 3 istekle distance= 3 isteklenmek distance= 3 içtenlikle distance= 3 destekleme distance= 4	0.0216

çocuk bakıcısı	çocuk bakıcısı distance= 0 çocuk bakıcı distance= 2 at bakıcısı distance= 5 at bakıcısı distance= 5 çocuk yazını distance= 5	0.0316
çock ba	çokça distance= 3 acaba distance= 4 çıkma distance= 4 caba distance= 4 çaba distance= 4	0.0170
çocukça	çocukçu distance= 1 çocukçu distance= 1 çabukça distance= 2 çocuk distance= 2 çocuklar distance= 2	0.0172
çoukça	çokça distance= 1 çabukça distance= 2 çocukçu distance= 2 çokçu distance= 2 çocukçu distance= 2	0.0185
fakir	fakir distance= 0 bakir distance= 1 fakr distance= 1 fikir distance= 1 hakir distance= 1	0.0166
fakitl	akil distance= 2 akit distance= 2 fail distance= 2 fakat distance= 2 fakir distance= 2	0.0177

As shown in the Table 1, the program tested with 20 different words. The running times were computed for each word in seconds. A word was written correctly. Then, random changes were made in the letters and results were reached. These steps were performed for 10 different words. A total of 20 test words, alternative lists and running times were obtained. At the end of the study, it was found that with the same order, the similarity of letters increased and the distance and running times decreased.

REFERENCES

[1] https://rosettacode.org/wiki/Levenshtein_distance#Java

[2] https://stackoverflow.com/questions/1844688/how-to-read-all-files-in-a-folder-from-java

[3] https://github.com/belenyasin/Java-Gecen-Sure-Hesaplama