Natural Language Processing
Home Assignment 1

Problem 1:

*Compute the edit distance (using insertion cost = 1, deletion cost = 1, substitution cost = 1) of **"leda"** to **"deal"**. Show your work (using the edit distance grid)* We derive the edit distance between words **"leda"** and **"deal"**, according to the next algorithm [1]:

```
function MIN-EDIT-DISTANCE(source, target) returns min-distance

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix distance[n+1,m+1]

  # Initialization: the zeroth row and column is the distance from the empty string
      D[0,0] = 0
      for each row i from 1 to n do
         D[i,0] ← D[i-1,0] + del-cost(source[i])
      for each column j from 1 to m do
         D[0,j] ← D[0,j-1] + ins-cost(target[j])

  # Recurrence relation:
  for each row i from 1 to n do
      for each column j from 1 to m do
         D[i,j] ← MIN( D[i−1,j] + del-cost(source[i]),
                       D[i−1,j−1] + sub-cost(source[i],target[j]),
                       D[i,j−1] + ins-cost(target[j]))
  # Termination
  return D[n,m]
```

Figure 1: The minimum edit distance algorithm (Dynamic Programming). The divergent costs can be applied (i.e., $\forall x, ins - cost(x) = 2$). Assume that there is no cost for the substitution of the letter by itself or $sub - cost(x, x) = 0$) [1].

Table 1 illustrates the obtained edit distance grid for **"leda"** and **"deal"**.

Table 1: Edit distance grid for **"leda"** and **"deal"**

Src	#	**l**	**e**	**d**	**a**
#	0	1	2	3	4
d	1	1	2	3	3
e	2	2	1	2	3
a	3	2	2	2	3
l	4	3	3	2	**3**

Hence, the Levenshtein distance between *"leda"* and *"deal"* is 3:

l	**e**	**d**	**a**	
d	**e**		**a**	**l**

Problem 2:

*Figure out whether **"drive"** is closer to **"brief"** or to **"divers"** and what the edit distance is to each. You may use any version of distance that you like.*
To begin with, we determine the edit distance (using insertion cost = 1, deletion cost = 1, substitution cost = 1) among *drive* and *brief* (**Table 2**).

Table 2: Edit distance grid for **"drive"** and **"brief"**

Src	#	**b**	**r**	**i**	**e**	**f**
#	0	1	2	3	4	5
d	1	1	2	3	4	5
r	2	2	1	2	3	4
i	3	3	2	1	2	3
v	4	4	3	2	2	3
e	5	5	4	3	2	**3**

Next, we derive edit distance between **"drive"** and **"divers"** (**Table 3**).

Table 3: Edit distance grid for **"drive"** and **"divers"**

Src	#	**d**	**i**	**v**	**e**	**r**	**s**
#	0	1	2	3	4	5	6
d	1	0	1	2	3	4	5
r	2	1	1	2	3	3	4
i	3	2	1	2	3	4	4
v	4	3	2	1	2	3	4
e	5	4	3	2	1	2	**3**

The Levenshtein distance between "drive" and "brief" is 3:

d	**r**	**i**	**v**	**e**	
b	**r**	**i**		**e**	**f**

The Levenshtein distance between "drive" and "divers" is 3:

d	r	i	v	e		
d		i	v	e	r	s

*Thus, the edit distance between **"drive"** and **"belief"** is the same as "drive" and **"divers"**.*

Problem 3:

Now implement a minimum edit distance algorithm and use your hand-computed results to check your code.

Figure 2 demonstrates the code for implementing minimum edit distance algorithm.

```
def levenshteinDistance(temp, temp2):
    if len(temp) > len(temp2):
        temp = temp2
        temp2 = temp

    dist = range(len(temp) + 1)

    for ind, ind2 in enumerate(temp2):
        temp_dist = [ind+1]
        for iind, iind2 in enumerate(temp):
            if iind2 != ind2:
                temp_dist.append(1 + min((dist[iind], \
                                    dist[iind + 1], temp_dist[-1])))
            else:
                temp_dist.append(dist[iind])

        dist = temp_dist

    return dist[-1]
```

Figure 2: The minimum edit distance algorithm (Dynamic Programming) (using insertion cost = 1, deletion cost = 1, substitution cost = 1)

Figure 3 displays the results of the experiments with *drive, divers, brief, belief, leda* and *deal.* The same results were obtained manually.

```
In [61]: levenshteinDistance("leda", "divers")
Out[61]: 5

In [62]: levenshteinDistance("drive", "brief")
Out[62]: 3

In [63]: levenshteinDistance("drive", "divers")
Out[63]: 3

In [64]: levenshteinDistance("leda", "deal")
Out[64]: 3

In [65]: levenshteinDistance("drive", "belief")
Out[65]: 5
```

Figure 3: Experimenting with the implemented minimum edit distance algorithm

References

[1] Jurafsky D., Martin J., Speech and Language Processing, An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Stanford University, University of Colorado at Boulder, 2017, p. 28-30