ADSA Assignment 3

[redacted]

OS: Ubuntu 20.04 running inside WSL on Windows 10.
Compiler version: Python 3.8.5 [GCC 7.3.0] Anaconda, Inc. on linux

1. Problem Description

Given two sequences, target and query, we have to find the minimum cost of aligning the two given sequences with the condition that the query sequence should be lexicographically smallest.

2. Solution Description

We implement the Needleman Wunsch dynamic programming algorithm to find the global sequence alignment.

2.1 Forming the array

We first create an 2D array of size (x+1, y+1) where x is the length of target sequence and y is the length of the query sequence. Each row and column indicates a unique character in the sequence. We create an extra row and column to store gap costs for each element. The (0,0) element contains the cost 0.

We then start filling the table using the following conditions:

- If the characters are the same in both sequences, no cost is incurred and we copy the cost from diagonally top left element to the current element.
- If the characters are not same, either a gap cost or mismatch cost will be incurred. To fill this, we find the minimum of gap cost in both query and target sequences as well as the mismatch cost. We fill the current element with the minimum cost of these three.

The pseudocode is given below:

```
Alignment(x,y):
table[0. . . m][0. . .n]
table[i,0] = i * gap_cost for each i
table[0,j] = j * gap_cost for each j
for i = 1,.........., n
     for j = 1,.........., m
          if x[i] == y[j]
               table[i][j] = table[i-1][j-1]
          else
               table[i][j] = min(
```

```
table[i-1][j-1] + mismatch_cost,
table[i-1][j] + gap_cost,
table[i][j-1] + gap_cost)
```

2.2 Backtracking

Once the table is created and filled with the gap costs and mismatch costs we need to backtrack from the bottom right corner to find the aligned sequence. To make sure we get the lexicgraphically smallest query sequence we need to make sure that _ is added to query sequences before adding to target sequence. As the _ is the smallest character possible, we will be making sure we are getting the smallest possible sequence.

The steps to backtrack are necessarily to be followed in the order given otherwise we will not get lexicograpphically smallest sequence.

- If the current characters in the sequences are same, we add those to alignment strings and move to next character.
- If the left element + gap cost is equal to the current element in the table, add _ to query alignment and current character in target sequence to target alignment.
- If the top element + gap cost is equal to the current element in the table, add the current character in query sequence to query alignment and _ to target alignment.
- If all other conditions fail, we will have to incur a mismatch cost. We check if the diagonally top left element + mismatch cost is equal to current element and add current characters in both the target and query sequences to target alignment and query alignment.
- If no conditions satisfy, we have wrongly calculated our table in the first step and results will be wrong. This case has not been handled.

Repeat the above steps until we reach either the top or left border of the table. Once we get there, there are no more characters left in either the target or query sequences. We can now just add all the remaining characters in target sequence or query sequence to target alignment or query alignment respectively.

The psuedocode is given below:

```
Backtracking(x,y):
i = x.len
j = y.len
m = align_x.len
n = align_y.len

if x[i] == y[j]
      align_x[m] = x[i]
      align_y[n] = y[i]
      x--, y--, i--, j--
```

```
else if table[i][j-1] + gap_cost == table[i][j]
      align_x[m] = '_'
      align_y[n] = y[j-1]
      m--, n--, j--

else if table[i-1][j] + gap_cost == table[i][j]
      align_x[m] = x[i-1]
      align_y[n] = '_'
      m--, n--, i--

else if table[i-1][j-1] + mismatch_cost == table[i][j]
      align_x[m] = x[i]
      align_y[n] = y[j]
      m--, n--, i--, j--

while m > 0
      if i > 0
            i--
            align_x[m] = x[i]
            m--
      else
            align_x[m] = '_'
            m--

while n > 0
      if j > 0
            j--
            align_y[n] = y[j]
            n--
      else
            align_y[n] = '_'
            n--
```

2.3 Results

Once the above steps are done, the penalty cost is calculated and stored in the bottom right element of the table. Additionally, the alignment sequences have been calculated for both the target and the query string. We need to just print these three as our standard output.

2.4 Time complexity

In filling the table we need to visit each and every element in both the sequences. Hence The time complexity is *O(mn)* for filling the table.

To backtrack the and find the aligned sequence, we need to move across the table for the length of the sequences. In worst case, we will need to move the length of both sequences combined, i.e. m+n. Since constant work is done in each iteration, we will need just *O(m+n)* to backtrack the alignment.

Hence the total cost can be calculated as, *O(mn)* + *O(m+n)*.
Since, mn > m+n we can say the cost for this algorithm is *O(mn)*.