ADSA Programming Assignment 3

EE19MTECH01008

13 December 2020

OS: Linux (Ubuntu 20.04)
Gcc version: 9.3.0

1 Problem

In this assignment, we will look at an application of Dynamic Programming to align DNA sequences. In the global DNA sequence alignment problem, you are given as input two DNA sequences S 1 , S 2 , which are strings using the four characters A, C, T, G. The strings are of lengths m, n, where possibly m 6 = n. Given two DNA sequences, the goal is to compute the best possible (minimum) alignment cost between them.

Assumptions: Input strings are valid.

1.1 Alignment and scoring

Consider the target and query sequences (our first sequence will always be called the target and second the query): AGGGCCT, TGGCA respectively. Note that these are of different lengths, so one natural way to pair them would be:

A G G G C C T
T G G_ C_ T

Here, is called a gap, that can be inserted anywhere in either string, to make the rest of sequences align better. There is a penalty s(x, y) for matching character x to character y. Consider the following scoring function:

$$s(x,y) = \begin{cases} 0, & \text{if x=y} \\ 2, & \text{if x} \neq \text{y} \\ 1, & \text{if x=\_ or y = \_} \end{cases} \tag{1}$$

Note that a smaller cost means better alignment. The cost of aligning a character with the gap character is usually denoted by γ, and is called the gap-penalty. According to the above scoring rules, the pairing given above has an alignment cost of 2 + 1 + 1 = 4.

Consider aligning the target string: AACAGTTACC and query string: TAAGGTCA.
Two possible alignments and corresponding scores are given: Score:

A A C A G T T A C C
T A A G G T C A _ _
2+0+2+2+0+0+2+0+1+1 = 10
A A C A G T T A C C
T A _ A G G T _ C A
Score: 2 0 1 0 0 2 0 1 0 2 = 8

Thus, the second alignment is preferred. Note we may decide to insert gap characters in both target or query string to minimize the alignment scores, and not just in the string with smaller length. Now consider another example: two possible alignments of AGTACG and ACATAG. Both have an alignment cost of 4, which, you can check, is the minimum alignment cost between the two strings.

Alignment 1:
A_GTACG
ACATA_G
Alignment 2:
A__GTACG
ACA_TA_G

1.2 Problem to solve

Given two sequences, write code to compute the cost and alignment pattern of the optimal alignment. The scoring s(x, y) function will be specified as part of the input, along with the target and query sequences. The output should be (each on a new line):

1. The optimal alignment score (a single integer)
2. The pattern with gaps of the target sequence that is matched (without spaces)
3. The pattern with gaps of the query sequence that is matched (without spaces)

Important criterion: As seen in the last example above, there may be multiple output patterns possible with the same alignment score. You are to output the alignment pattern in which the matched query pattern is lexicographically the least, when read from left to right. You should consider the character ordering: _ < A < C < G < T. The gap character is considered the smallest. e.g. ACC TG is smaller then ACCT G, because is smaller than T. This ordering makes sense even for strings of unequal length: AT CGG is smaller than ATCC.

2 Solution

Needleman Wunsch Global DNA Sequence Alignment algorithm is used to solve this problem [1]. The major steps in the algorithm are:

1. Initialize a score matrix with column and rows corresponding to the given target and query sequence

2. Calculation of scores and filling in the matrix to obtain the optimal score of alignment

3. Trace-back to get the optimal pattern for both target and query sequence to get the optimal score.

The implementation is based on these steps. There are primarily 3 main function int he code: **initmat()**, **scoremat()** and **traceback()**.
Time Compexity: O(mn)
Space Complexity: O(mn)
Here, "m" and "n" are the length of the target and query sequence respectively.

2.1 Algorithm Explanation:

1. **Initialization of the score matrix:** The **initmat()** function initializes a score matrix of (m+1)*(n+1) where m and n are the lengths of the target and query sequences respectively. The first cell is set to 0. The first column and row is filled with gap penalty being added to the previous value.

2. **Filling scores:** The **scoremat()** function fills the score matrix. The corresponding characters in the target and query strings are matched with each other and a penalty is calculated based on the mismatch or match. The final value to be put in the individual locations in the matrix is the minimum of left, top and diagonal values. According the algorithm the optimal score is obtained on the bottom right corner of the matrix after the whole matrix is filled.

3. **Traceback:** The **traceback()** fucntion is required to get the optimal pattern so as to get the optimal score. There are certain rules pertaining to this function. While tracing back we need to find the cell from where the current value came from. If the current value came from top then insert a gap in optimal query sequence. If the current value came from diagonal then add the corresponding letter in both the optimal target and query sequences. If the current value came from left then add a gap optimal target sequence.
   Another case where the lengths of the target and the query sequence may not be similar is also handled. If the length of the query sequence is greater than target sequence, we add spaces in optimal target sequence and add character of query sequence in the optimal query sequence. If the length of the target sequence is larger then query sequence add spaces in optimal

query sequence and add character of the target sequence in the optimal target sequence.

2.2 Pseudo-code:

Algorithm 1 Pseudo code of the solution

```
Read target sequence length and query sequence length (tarlen, quelen)
Read gap penalty, mismatch penalty (gap,mismatch)
Read target string (target)
Read query string (query)
string optimaltarget = ""
string optimalquery = ""
scorematrix(m+1, n+1)
initmat(scorematrix, tarlen, quelen, gap)
scoremat(scorematrix, gap, target, query, mismatch, tarlen, quelen)
traceback(scorematrix, gap, target, query, optimaltarget, optimalquery, mismatch, tarlen, quelen)
Return scorematrix[tarlen][quelen]
Return optimaltarget
Return optimalquery
```

References

[1] Wikipedia contributors. Needleman–wunsch algorithm — Wikipedia, the free encyclopedia, 2020. [Online; accessed 13-December-2020].