Programming for Bioinformatics | BIOL7200

Week 11 Exercise

October 29, 2019

This week's assignment consists of a main graded assignment and an additional assignment for bonus points. The first few pages describe the algorithm at length. The text below summarizes the algorithm.

For this week, assume that the user always specifies the input correctly. **Your script will be graded on the output produced and not how all the errors are handled.**

The CI for this week will also not be public, i.e. you will not be able to see the autograder. We will run your script using a CI after the submission deadline.

Again, please do not use any modules other than "sys". The CI will not install missing modules. **Do not use** input() for any input either, we do not handle this in the CI. The CI **will fail** if you do not follow these instructions.

Instructions for submission

- **This assignment is due Monday, November 4, 2019 at 11:59pm. Late submissions will receive a 0**
- Your code must be available on GitLab at the above time to be graded
- Name the Needleman-Wunsch algorithm script as `nwAlign.py`
- Optional bonus assignment: Name the Smith-Waterman algorithm script as `swAlign.py`
- Your code should run as `./nwAlign.py <seq1.fa> <seq2.fa>` OR `./swAlign.py <seq1.fa> <seq2.fa>`
- Both scripts should output their results to `STDOUT`
- DO NOT HARDCODE any file name!
- Please use the `#!/usr/bin/env python3` as your shebang

Main assignment: Needleman-Wunsch (NW) algorithm

Max score: 100 points

This is an example of another complex problem that has a rather simple solution. NW algorithm is a classical bioinformatics algorithm designed to obtain optimal global alignment for a given pair of sequences. The algorithm falls under the class of dynamic programming which in simple language is the class of algorithm that work by breaking a problem into subproblems, solving each subproblem and joining the solutions to reach the global solution.

The algorithm can be divided into three steps:

1. *Initialization*: Construction of the matrix with the two sequences as each axis and selection of a suitable scoring system. For simplicity, let's have three types of scores:
    a. Match = +1
    b. Mismatch = -1
    c. Gap = -1

2. *Matrix filling:* Filling the matrix based on the scoring system. This occurs one row at a time, starting from the topmost row. Each cell in the matrix derives the value from the adjacent cells located to the left, top-left or on top of the current cell. The match score is added or gap/mismatch penalty is subtracted from these adjacent cells and the maximum value is carried over to the current cell (Figure 1).

3. *Backtracking*: Once the matrix has been filled up, backtracking is done to compute the optimal alignment(s). The backtracking step starts from the very last cell filled in the matrix (the bottom-right cell) and proceeds to the first cell filled in matrix (the cell with 0 in the upper left corner of the matrices in Figure 1). This backtrack path is computed by moving through the adjacent cells (cells to the left, top-left and on top of the current cell) with the maximum score such that the path has the maximum total score (Figure 2). If multiple paths exist, then all of them are considered to be the optimal paths. This path is converted to an alignment by the following rule: the path moves diagonally to the left if there is a match or if the maximum score of the adjacent cells is present in the diagonal left cell. If either of these are true, the two corresponding characters from each sequence are aligned together. When the maximum score is obtained by moving horizontally, then a gap is introduced in the sequence on the vertical axis, and if the path moves vertically, then a gap is introduced in the sequence on the horizontal axis.

Backtracking rules:

1. Always take the diagonal when the diagonal is either (1) the highest score or (2) tied for highest score
2. If the diagonal is not the highest score, take the "Up" if it is either (1) the highest score or (2) tied for highest score.
3. Take the "Left" if the diagonal and "Up" are not the highest

For the purpose of this assignment, you will only observe a single optimal path with the above rules in the test sequences we use. You will not have to worry about multiple, optimal paths.

As before, do not think too much or you will spend most of the week doing this. Think simple, use arrays and loops. If you are not too confident with coding, start by writing pseudocode. When you have it right, then code it up. Ideally, designing the pseudocode should take 80% of your time, translating it into code should take 10% and testing should take 10%. If you aren't sincere with the pseudocode, you are going to have a bad time with this assignment.

Syntax: `./nwAlign.py <input FASTA file 1> <input FASTA file 2>`

Example usage: `./nwAlign.py seq1.fa seq2.fa`

seq1.fa contains:

```
>seq1.fa
ATTGCC
```

seq2.fa contains:

```
>seq2.fa
AGTCC
```

Output format:

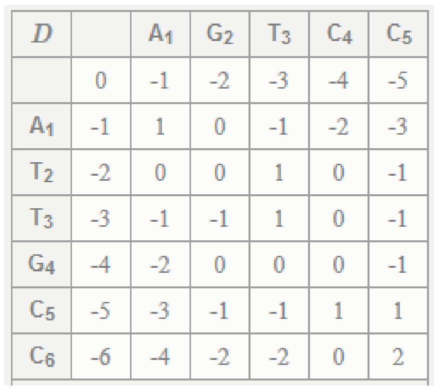
```
ATTGCC
|*| ||
AGT-CC
Alignment score: 2
```

"|" represents a match

"*"represents a mismatch

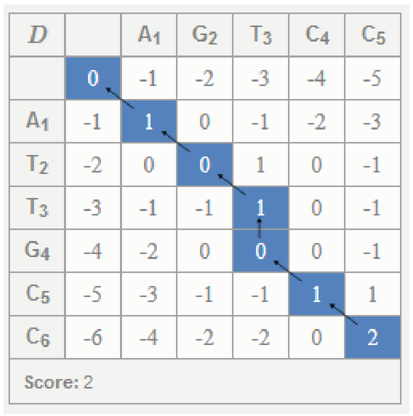
Print the output to STDOUT.

You can test out the generated matrix and alignment here: http://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Needleman-Wunsch. As specified above, for the purposes of this exercise we will only consider problems with only one possible alignment.



D		A_1	G_2	T_3	C_4	C_5
	0	-1	-2	-3	-4	-5
A_1	-1	1	0	-1	-2	-3
T_2	-2	0	0	1	0	-1
T_3	-3	-1	-1	1	0	-1
G_4	-4	-2	0	0	0	-1
C_5	-5	-3	-1	-1	1	1
C_6	-6	-4	-2	-2	0	2

Figure 1. Matrix filling step of dynamic programming



D		A_1	G_2	T_3	C_4	C_5
	0	-1	-2	-3	-4	-5
A_1	-1	1	0	-1	-2	-3
T_2	-2	0	0	1	0	-1
T_3	-3	-1	-1	1	0	-1
G_4	-4	-2	0	0	0	-1
C_5	-5	-3	-1	-1	1	1
C_6	-6	-4	-2	-2	0	2

Score: 2

Figure 2. Matrix backtracking step and the generation of optimal alignments

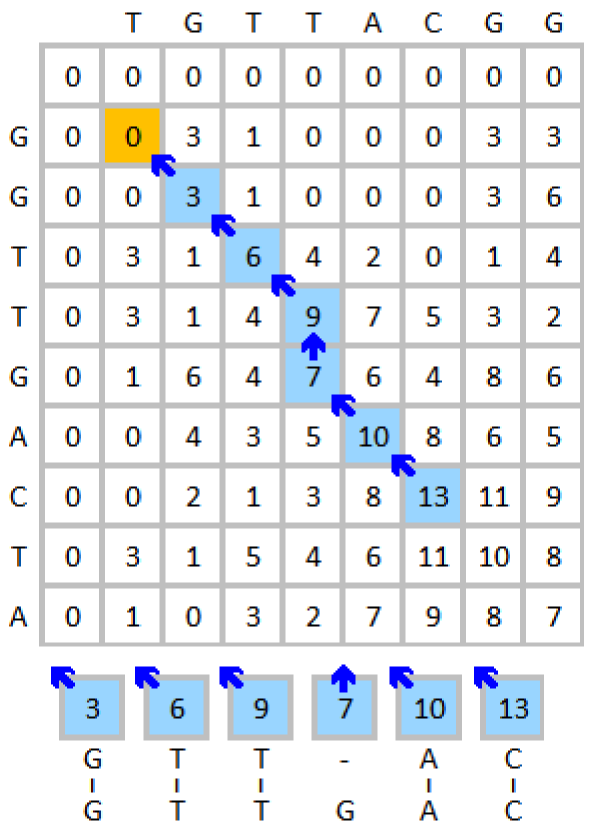
Bonus assignment: Smith-Waterman (SW) algorithm

Bonus score: 100 points

The Smith-Waterman algorithm is a variant of Needleman-Wunsch that is applicable for local alignments. The differences between the algorithms are:

1. There are no negative values within the matrix. If subtracting a gap penalty or mismatch score results in a negative value, the value for the cell becomes 0.
2. The traceback starts with the cell with the highest value and ends when a 0 valued cell is encountered during the traceback.
3. NOT TO BE IMPLEMENTED FOR THIS ASSIGNMENT: The next local alignment starts with second highest score and ends at the first encountered 0 valued in the path.

If you've coded NW smartly, this should be a straight-forward change within the code.



		T	G	T	T	A	C	G	G
	0	0	0	0	0	0	0	0	0
G	0	0	3	1	0	0	0	3	3
G	0	0	3	1	0	0	0	3	6
T	0	3	1	6	4	2	0	1	4
T	0	3	1	4	9	7	5	3	2
G	0	1	6	4	7	6	4	8	6
A	0	0	4	3	5	10	8	6	5
C	0	0	2	1	3	8	13	11	9
T	0	3	1	5	4	6	11	10	8
A	0	1	0	3	2	7	9	8	7

An example matrix using Smith-Waterman algorithm (source: https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm). The scoring scheme is +3 for match, -3 for mismatch and -2 for gap. Notice the lack of negative values and the traceback path starting from the middle of the matrix.

Syntax: `./swAlign.py <input FASTA file 1> <input FASTA file 2>`

Example usage: `./swAlign.py seq1.fa seq2.fa`

seq1.fa contains:

```
>seq1.fa
TGTTACGG
```

seq2.fa contains:

```
>seq2.fa
GGTTGACTA
```

Output format:

```
  GTT-AC
  ||| ||
  GTTGAC
  Alignment score: 1
```