CISC-471 WINTER 2015

HOMEWORK 7

Please work on these problems and be prepared to share your solutions with classmates in class on Wednesday March 4. **NOTE** To provide an incentive for you to do this work you may submit it to me either hard copy or electronically either in class on March 4, or electronically before class on March 4. Your work will not be marked nor will it be returned. However, I will keep it, and use it to possibly raise your grade when it comes time to submit the final grades, For example if you do poorly on the final and have consistently handed in homework I may raise your grade.

Programming

Write a program that implements the algorithmic solution for problems 6.21, 6.22, and 6.23 given below. I will ask students to demonstrate their program when answering those problems. There is a dynamic programming solution for each of those problems, (and obviously for 6.20), so programs that solve one of these problems should be able to solve any of the others with minor modification.

Problems

These questions come from *An Introduction to Bioinformatics Algorithms* by Neil C. Jones and Pavel A. Pevzner.

Problem 6.20: Consider the sequences v = TACGGGTAT and w = GGACGTACG. Assume that the match premium is +1 and that the mismatch and indel penalties are -1.

- Fill out the dynamic programming table for a global alignment between v and w. Draw arrows in the cells to store the backtrack information. What is the score of the optimal global alignment and what alignment does this score correspond to?
- Fill out the dynamic programming table for a local alignment between v and w. Draw arrows in the cells to store the backtrack information. What is the score of the optimal local alignment in this case and what alignment achieves this score?

Problem 6.21: For a pair of strings $v = v_1, ..., v_n$ and $w = w_1, ..., w_m$, define M(v, w) to be the matrix whose (i, j)th entry is the score of the optimal global alignment which aligns the character v_i with the character w_j . Give an O(nm) algorithm which computes M (v, w).

Problem 6.22: Define an overlap alignment between two sequences $v = v_1, ..., v_n$ and $w = w_1, ..., w_m$ to be an alignment between a suffix of v and a prefix of w. For example, if v = TATATA and w = AAATTT, then a (not necessarily optimal) overlap alignment between v and w is shown below:

```
ATA
AAA
```

An optimal overlap alignment is an alignment that maximizes the global alignment score between $v_i, ..., v_n$ and $w_1, ...w_j$, where the maximum is taken over all suffixes $v_i, ..., v_n$ of v and all prefixes $w_1, ...w_j$ of w. Give an algorithm which computes the optimal overlap alignment, and runs in time O(nm). Explain how to fill in the first row and column of the dynamic programming table and give a recurrence to fill in the rest of the table. Give a method to find the best alignment once the table is filled in.

Problem 6.23: Suppose that we have sequences $v = v_1, ..., v_n$ and $w = w_1, ..., w_m$, where v is longer than w. We wish to find a substring of v which best matches all of w. Global alignment wont work because it would try to align all of v. Local alignment wont work because it may not align all of w. Therefore this is a distinct problem which we call the *fitting problem.* Fitting a sequence w into a sequence v is a problem of finding a substring v' of v that maximizes the score of alignment s(v', w) among all substrings of v. For example, if v = GTAGGCTTAAGGTTA and w = TAGATA, the best alignments might be as shown in the table below.

	global	local	fitting
v w	GTAGGCTTAAGGTTA -TAG----A---T-A	TAG TAG	TAGGCTTA TAGA--TA
score	-3	3	2

The scores are computed as 1 for match, -1 for mismatch or indel. Note that the optimal local alignment is not a valid fitting alignment. On the other hand, the optimal global alignment contains a valid fitting alignment, but it achieves a suboptimal score among all fitting alignments.

Give an algorithm which computes the optimal fitting alignment. Explain how to fill in the first row and column of the dynamic programming table and give a recurrence to fill in the rest of the table. Give a method to find the best alignment once the table is filled in. The algorithm should run in time O(nm).