CMSC 423: Project #2: Finding Several Good Local Alignments

Due: Tuesday, December 8th at 11:59pm.

In this second project, you will implement local alignment with affine gaps along with a scheme for finding several good alignments, not just the best. The project will be done in groups of 2 — these groups can be different from those in part 1. Some of your grade for the project will be based on a confidential survey completed by your partner.

Format of the command: Your program will be run using the following command:

```
java multilocal.jar score.matrix in.fasta out.fasta k
```

where k is an integer that gives the desired number of alignments. Your program must compile on the submit server. It should output into `out.fasta` the k best local alignments of the two sequences in `in.fasta` (see below).

Format of input: The format of the `score.matrix` file will be the same as in Project #1. The format of the `in.fasta` file is the same as in Project #1 as well, except that it can contain only 2 sequences. You should handle errors the same way as in Project #1 as well.

Format of the output: You will find k local alignments (where k is given as the last parameter on the command line). The output file will be in the same multi-FASTA format as in Project #1. The output file should contain $2k$ sequences, where sequences 1 and 2 give the best local alignment, sequences 3 and 4 give the second best local alignment, and so on.

Algorithm to find the top-k different local alignments: A particular local alignment can be specified by the edges that it uses during the traceback. Two local alignments are said to be *disjoint* if they do not use any of the same edges. Your algorithm should output the best alignment, and then repeatedly output the next best alignment *that is disjoint* from all the alignments output so far, stopping when a total of k alignments (including the best) have been output.

A simple way to do this is the following algorithm:

```
ForbiddenEdges = []
For i = 1 .. k:
   Fill in the dynamic programming matrix, disallowing the use of any ForbiddenEdges
   Traceback, adding every visited edge to ForbiddenEdges
```

In other words, we recompute the dynamic programming matrix k times, but when we fill in a cell, we don't consider any options in the recurrence that would cause us to use a previously used edge. This runs in time $O(knmf)$, where the length of the sequences are n and m and f is some function that accounts for the time spent searching the `ForbiddenEdges` list when filling in the matrix.

Grading: There are faster ways of finding the top-k alignments than the simple algorithm above. 20% of your grade will be based on the speed of your algorithm and implementation. **You should submit, along with your code, a short document (PDF or plain text file) named README.txt or README.pdf that describes your approach to solving the problem, why it is correct, and its advantages and disadvantages.**

Code Reuse: You can reuse any of the code your team wrote for Project #1 and you can use the code that will be posted on the website. You cannot use anyone else's Project #1 code.

Submission: You will submit using the CS submit server: `submit.cs.umd.edu`. There will be no tests on the submission server — it is used only for timestamping.