CSE 620: Project 5
Alignment Algorithms
Due: Nov. 1, 2013

In this assignment you will implement two variations on the traditional alignment algorithm. One is a simple variation of the global alignment algorithm we looked at in class. The other requires some more substantial additions.

There are three types of alignments we will look at here:

- **Global alignment**: the algorithm we looked at in class. An implementation of the Needleman-Wunch algorithm has been provided.
- **Local alignment**: Where global alignment requires that you align the entirety of the two sequences, local alignment allow you to trim off the ends – picking the two best substrings for alignment.
- **Affine gap scoring**: In the alignment problem we reviewed, each gap incurred a single fixed penalty. In affine scoring, we have separate penalties for the *occurrence* of a gap and the *length* of a gap. That is: let *o* be the gap *occurrence* penalty and *c* be the gap *continuation penalty*. If we have *n* consecutive gap symbols in one alignment, we take a total penalty of $o + c*n$. For example, the alignment:

  ```
  AAACCCAAA----AAA
  AAA---AAATTTAAA
  ```

  would incur a total gap penalty of $(o + 3n) + (o + 4n) = 2o+7c$. (For a total score of $6m - 2o - 7c$, where *m* is the score for aligning an A to an A.)

  Note that to avoid an occurrence penalty they must be consecutive *on the same sting*. For example:

  ```
  A-GA
  AT-A
  ```

  would incur a penalty of $2o + 2$.

Affine scoring reflects the idea that the unlikely part of an insertion is that an insertion happens at all. The chance of a 10 base insertion is not that much smaller than the chance of a single base insertion, hence the penalty for a 10 base insertion should not be ten times that of a single base insertion.[1]

[1] In commenting on the idea that the existence of a large turtle is amazing, Terry Prachett wrote: "There is nothing amazing about size. Turtles are amazing…But the fact that there's a big turtle is far less amazing than the fact that there is a turtle *anywhere*." This is the same principle.

As an example, consider the sequences: AAATTTAGATTT and CCCTTTGTTTCCC. Assume all matches are worth 2 points, all mismatches are penalized at 1 point, and all gaps are penalized at 3 points. A optimal global alignment would be:

```
AAATTTAGATTT---
CCCTTT-G-TTTCCC
```

An optimal local alignment might be:

```
TTTAGATTT
TTT-G-TTT
```

If we are to move to local affine scoring, setting *o=10* and *c=1*, then this is no longer optimal. A better alignment would be:

```
TTTGAGTTT
TTTA--TTT
```

In this case, we accept a mismatch penalty of 1 and a total gap penalty of 12, which is much better than earning the 2 points for a match at the cost of 22 in gap penalties.

Assignment:

1. In alignment.py, write the code for the SmithWaterman algorithm, used to calculate the best local alignment of two strings given a scoring matrix C and a single gap penalty. The algorithm should return the aligned substrings and the score of the alignment. Note: this is a minor modification of the NeedlmanWunsch algorithm that has been provided for you. You can cut-and-paste the Needleman-Wunsch code earlier in the file and modify a few lines to get this working.
2. In alignment.py, write the code for the SmithWatermanAffine, used to calculate the best local alignment given an affine scoring system as explained above. Note: this requires a much more extensive change to the Needleman-Wunsch code.

Resources: I've provided you with the following files and code:

- Blosum62.txt: Contains a standard protein scoring matrix.
- DNA1.txt: Contains a DNA scoring matrix.
- DNA2.txt: Contains an alternative DNA scoring matrix.
- alignment_util.py: Contains potentially useful code, including:
    - **readScoringMatrix**: Parse the above files and returns a scoring matrix for use in the NeedlemanWunsch algorithm.
    - **scoreAlignment**: Given an alignment, a scoring matrix S, and a gap penalty *g*, calculates the score of the alignment. (Useful for verifying your score.)

- **scoreAlignmentAffine**: Given an alignment, a scoring matrix S, and affine gap penalties *o* and *s*, calculates the score of the alignment. (Useful for verifying your score.)

Working restrictions: You are expected to write and submit your own code. However, you **are** allowed to:

1. Discuss this, on an algorithmic level, with your classmates.
2. Use outside resources (Internet and text) in researching the algorithmic solutions.