# Problem Set 3: Dynamic Programming

Handed out Wednesday, February 12. Due at the start of class Friday, February 21.

**Homework Information:** Some of the problems are probably too long to attempt the night before the due date, so plan accordingly. No late homework will be accepted. You must cite any sources used outside of the course materials and textbook, including anyone you discussed the problems with.

**1.** (4 pts) Show the results of the DP algorithm for computing the longest common subsequence (LCS) as presented in class on the following input:

$$X = \langle \text{TACACT} \rangle \qquad Y = \langle \text{ATCAT} \rangle$$

(a) Show the contents of the lcs table (see Lecture 10). Also, show the matrix of helper values to obtain the solution. I would suggest representing them as arrows (see, Slide 5 from Lecture 10), but you may represent them as you like, as long as you explain your convention.

(b) Which sequence do you get if you apply the function get-lcs-sequence as given in class?

**2.** (3 pts) We have discussed two different types of alignment in class (global alignment, local alignment) and different scoring matrices that can be used for each. For the following scenarios, determine which of these alignments would be the best approach, and what sort of scoring matrix/gap penalty you would use. Make sure to explain why you made the choices you did. If you think the best alignment approach from the set of choices doesn't fit the problem well, describe any modification to the alignment that you think would be better.

1. You are given two protein sequences for two distantly related primates. You aren't exactly sure what the proteins sequences are, so you want to find out if there are any conserved protein domains shared between the two sequences.
2. You are given DNA sequence for a particular gene from two different members of the same species. You would like to determine how similar the gene is between the individuals.
3. You receive a mystery email that says it's from `leroy@rhodes.edu` and has a title of "Align me!". The email contains two fasta files, each containing a single protein sequence of similar length. You suspect that Leroy the Lynx may have embedded a short secret message somewhere inside the protein sequences that you can only find once you align the files.

**3.** (4 pts) We define an *overlap alignment* between two sequences $\mathbf{v} = v_1 v_2 \ldots v_n$ and $\mathbf{w} = w_1 w_2 \ldots w_m$ to be an alignment between a suffix of $\mathbf{v}$ and a prefix of $\mathbf{w}$. For example, if $\mathbf{v} = TATATA$ and $\mathbf{w} = AAATTT$, then a (not necessarily optimal) overlap alignment between $\mathbf{v}$ and $\mathbf{w}$ is:

```
ATA
AAA
```

The optimal overlap alignment is an alignment that maximizes the global alignment score between $v_i \ldots v_n$ and $w_1 \ldots w_j$, where the maximum is taken over all suffixes $v_i \ldots v_n$ of $\mathbf{v}$ and all prefixes $w_1 \ldots w_j$ of $\mathbf{w}$. **Give a dynamic programming algorithm that computes the optimal overlap alignment between sequences v and w given a gap score $\sigma$ and scoring matrix $\delta$.** In particular, make sure your algorithm addresses the following points.

- Clearly specify any dynamic programming table the algorithm uses, and how large it is.
- Present your base, recurrence relation, the order in which your algorithm fills out the table and where the final solution is found in the table.
- Includes a description on how to get the optimal score and the alignment that corresponds to that score.

Furthermore, you should provide some brief explanation about why you think your algorithm is correct (you are welcome to write a full proof if you wish, but that is not required) and and explanation of your algorithms asymptotic running time.

**4.** (4 pts) A string **x** is called a *supersequence* of a string **v** if **v** is a subsequence of **x**. (Recall from the reading that a subsequence of a string **s** is a string **t** that can be derived from **s** by deleting 0 or more characters without changing the order of the remaining characters). For example, `ABLUE` is a supersequence for `BLUE` and `ABLE` (`BLUE` and `ABLE` and both subsequences of `ABLUE`). This problem, and others similar to it, often show up when analyzing a set of related DNA/RNA or Protein sequences.

Given strings **v** and **w**, devise a dynamic programming algorithm to find the shortest supersequence for both **v** and **w**. In particular, make sure your algorithm addresses the following points.

- Clearly specify any dynamic programming table the algorithm uses, and how large it is.
- Present your base, recurrence relation, the order in which your algorithm fills out the table and where the final solution is found in the table.

If you are curious, you can also explain how your algorithm actually constructs the superstring, once it has filled out the DP table, but you do not need to include this to get full credit.