CSE 527: Computational Biology **Assignment #1**
January 13, 2000 **due: Thursday, January 27**

Choose as many of the following problems as you care to work on, and take each as far as you can. There are a few that I don't know how to solve. I don't expect you to tackle all the problems, nor even all the problems that I know how to solve. I will get much more excited about promising partial progress on an open problem than about long solutions to all the routine problems. Keep your answers clear and concise. If you use references, please include citations.

1. Give an explicit example of two strings S and T, a scoring function, and a linear affine gap penalty function for which the optimal global alignment (without the extra gap penalty function), the optimal global alignment with the gap penalty function, and the optimal local alignment (without the extra gap penalty function) are all different. Show these three optimal alignments. Try to select your example to illustrate how very different these three optimal alignments can be.
2. Given strings S and T, the *longest common subsequence problem* is to find the longest string that is a subsequence of both S and T, and the *longest common substring problem* is to find the longest string that is a (contiguous) substring of both S and T. Show that these problems are special cases of problems discussed in lectures, and conclude what the running time is for the algorithms that solve them. Can you devise faster algorithms for either of these special cases?
3. In this problem you will derive the $O(n + m)$ space and $O(nm)$ time algorithm for computing optimal global alignments discussed in the lecture notes. First of all, explain how retaining only two consecutive rows of the table is sufficient to compute the *value* of an optimal global alignment, and how much space this uses.

   Assume for simplicity that n is a power of 2. The key to reconstructing the alignment itself is to find a column index k^* such that some optimal alignment path (in the sense of Section 4.1.2) passes through entry $(n/2, k^*)$. For any string S, let S^R denote string S written in reverse. Given two strings S and T, let $V^R(i, j)$ denote the value of an optimal global alignment of the first i characters of S^R and the first j characters of T^R. How much time and space does it take to compute $V(n/2, k)$ for all $0 \leq k \leq m$ and to compute $V^R(n/2, m - k)$ for all $0 \leq k \leq m$? How could you compute k^* efficiently from these values?

   You have now reduced the problem to two similar but smaller subproblems, namely finding the path from $(0, 0)$ to $(n/2, k^*)$ and finding the path from $(n/2, k^*)$ to (n, m). Show that solving these two subproblems "recursively" (i.e., by using the same method on each of them) results in an algorithm with the time and space bounds required.

   What modifications are necessary to this algorithm to make it work for computing optimal local alignments? For computing optimal alignments with gaps under the affine gap model discussed in the lecture notes?

4. Under certain circumstances it is reasonable to assume that gaps at the ends of alignments are less serious than gaps in the middle. To take an extreme, let's say such end gaps have zero cost. Give an efficient algorithm to find optimal global alignments under this assumption.
5. For the cDNA matching problem discussed in the lecture notes, investigate what is known about the distribution of intron and exon lengths in a eukaryote of your choice, and devise a gap penalty function that models this distribution well. Design and analyze an efficient algorithm to find optimal alignments with this gap penalty function.
6. Design an algorithm to find an optimal global alignment when the gap penalty function is an arbitrary function $g(q)$ of the gap length q. (Consider how you want to treat two adjacent gaps, and how this affects your recurrences. There are two ways you can handle this, and either one is justifiable.) Show that your algorithm runs in time $O(n^2m + nm^2)$ (assuming that $g(q)$ is computed by a supplied subroutine in constant time). Are there interesting special classes of gap penalty functions (other than the affine functions of Section 4.2.2) for which you can do substantially better?
7. Suppose you wanted to extend the optimal alignment algorithm to handle "inversion" mutations, in which a contiguous substring of a DNA sequence is replaced by its reverse complement (i.e., as though it were moved to the complementary DNA strand). It might be reasonable to have an affine inversion penalty, that is, some constant penalty per inversion, plus a penalty that grows linearly with the size of the inversion. Devise an efficient algorithm to find optimal alignments in this generalized setting. If possible, you would also like to handle substitutions, insertions, and deletions in the reversed portions.
8. How would you find long repeated substrings in a sequence? Start by defining what this means more carefully. Exact repeats? Approximate repeats? Gapped or gapless? Two occurrences or several occurrences? Adjacent (tandem) occurrences or separated occurrences? Repeats in the same orientation or reverse-complemented?

The following definitions of alternative valuation schemes for multiple string alignments are needed for the problems that follow.

Definition: Given a multiple alignment M of a set of strings, the *consensus character* of column i of M is the character that minimizes the sum of distances to it from all the characters in column i. Let $d(i)$ be this sum. The *consensus string* is the concatenation of all the consensus characters, in column order. The *alignment error* of M is $\sum_{i=1}^{l} d(i)$, where l is the number of columns in the alignment.

Definition: Given a set of strings, an *optimal consensus multiple alignment* is a multiple alignment that minimizes the alignment error.

Definition: Given a set $\mathcal{T}$ of strings and another string S, the *consensus error* of S is

$$E(S) = \sum_{T \in \mathcal{T}} D(S, T),$$

where $D(S, T)$ denotes the value of an optimal alignment of S and T.

Definition: Given a set $\mathcal{T}$ of strings, an *optimal Steiner string* S^* is a string that minimizes the consensus error $E(S^*)$. Note that S^* need not be in $\mathcal{T}$.

Definition: Given a set $\mathcal{T}$ of strings, an *optimal Steiner multiple alignment* is constructed by beginning with an optimal Steiner string S^*, and then adding the strings of $\mathcal{T}$ one at a time as in the approximation algorithm of Lecture 6.

9. Prove that removal of the spaces from the consensus string of an optimal consensus multiple alignment results in an optimal Steiner string. Conversely, prove that removal of the row for S^* from an optimal Steiner multiple alignment results in an optimal consensus multiple alignment. Is there a similar equivalence with optimal sum-of-pairs multiple alignments?

10. Given a set of strings, what is the time complexity of computing an optimal Steiner string?

11. Given a set of strings and an integer E, what is the time complexity of determining if there is a multiple alignment with alignment error at most E?

12. In a *local multiple alignment* a substring is selected from each string, and only the substrings contribute to the value of the alignment. Can you find a good approximation algorithm or heuristic for this problem?

13. The approximation algorithm for multiple string alignment given in the lecture notes is not used in practice. Instead, the following "greedy" method is sometimes used. Begin by aligning the two strings S and T with the minimum value of $D(S,T)$, where $D(S,T)$ denotes the value of an optimal alignment of S and T. Then successively merge in the string S with smallest distance $D(S,T)$ from any of the strings T already in the multiple alignment, as in the approximation algorithm in the lecture notes (but running the dynamic programming algorithm on S and T instead of S and a fixed string S_1). If you know Prim's algorithm for computing minimum spanning trees, this method is finding a multiple alignment consistent with a minimum spanning tree formed from the pairwise D distances.

    What is the running time of this algorithm? What can you prove about the quality of the alignment it produces? What if, instead of using a string T already in the multiple alignment, you use the consensus string of the multiple alignment?