Problem 1

We solve this problem as a dynamic program. We will maintain and fill a 3-dimensional array `wv` of size $n_1 n_2 m$, where $n_1 = |S_1|$, $n_2 = |S_2|$, and $m = |T|$. Let $\texttt{iw}(x, y, z)$ be 1 if there is an occurence of $S_1[x..n_1]$ and $S_2[y..n_2]$ interwoven in $T[z..m]$, possibly with spaces, and 0 otherwise. We compute the value of $\texttt{iw}(x, y, z)$ recursively. The solution is given by the value of $\texttt{iw}(1, 1, 1)$.

$$\texttt{iw}(x,y,z) = \begin{cases} 0 & \text{if } z > m \\ \texttt{iw}(x+1,y,z+1) + \texttt{iw}(x,y+1,z+1) & \text{if } S_1[x] = T[z] \text{ and } S_2[y] = T[z] \\ \texttt{iw}(x+1,y,z+1) & \text{if } S_1[x] = T[z] \text{ and } S_2[y] \neq T[z] \\ \texttt{iw}(x,y+1,z+1) & \text{if } S_1[x] \neq T[z] \text{ and } S_2[y] = T[z] \\ \texttt{iw}(x,y,z+1) & \text{if } S_1[x] \neq T[z] \text{ and } S_2[y] \neq T[z] \end{cases}$$

Here, "+" denotes logical OR. The algorithm takes $O(n_1 n_2 m)$ time.

Problem 2

We use a modification of the usual dynamic program for edit distance. We denote by $RM(i, j)$ the Row Minimum, or the minimum edit distance over all columns $\leq j$ for the row i. That is, $RM(i, j) = \min_{j' \leq j} W(i, j')$. Likewise we define the Column Minimum to be $CM(i, j) = \min_{i' \leq i} W(i', j)$.

Now, the recurrences are given by:

$$W(i,j) = \min \begin{cases} W(i-1, j-1) \\ W(i-1, j) + 1 \\ W(i, j-1) + 1 \\ RM(i, j-k) + k \\ CM(i-k, j) + k \end{cases}$$

$$RM(i,j) = \min \begin{cases} W(i,j) \\ RM(i, j-1) \end{cases}$$

$$CM(i,j) = \min \begin{cases} W(i,j) \\ CM(i-1, j) \end{cases}$$

Problem 3

We construct a graph on $m+1$ vertices. Each strint S_i has a vertex V_i corresponding to it. In addition there is a special vertex V_0. There is a directed edge between every ordered pair of vertices. An edge directed from V_i to V_j carries a weight equal to the length of S_j minus the length of the longest prefix of S_j that is a suffix of S_i. This edge signifies the increase in length when S_j is appended to S_i with the maximum overlap possible.

An edge directed from V_0 to V_i has length $|S_i|$, and an edge directed from V_i to V_0 has length 0.

Now consider a directed tour of this graph that starts and ends at the node V_0. This tour defined a superstring containing each of the strings S_i, that can be constructed by appending the strings in the order in which they occur in the tour to each other, with the maximum overlap possible. The edge lengths are defined such that the length of this tour is exactly the length of the corresponding superstring. Thus the shortest superstring problem is exactly the problem of finding a shortest directed tour of this graph, or the directed-TSP.

The shortest superstring problem can be shown to be NP-hard via a reduction from TSP, or vertex cover. Details can be found in Maier and Storer, *"A note on the complexity of the superstring problem"*, Report No. 223, 1977, Computer Science Laboratory, Princeton University.

Problem 4

Proof of Lemma: A string of length t has exactly $t - k + 1$ k-tuples, one starting at each position, except for the last $k - 1$ positions. Each individual character is contained in k of these k-tuples. Thus, l mismatches invalidate a total of at most lk k-tuples (mismatches near the boundary invalidate fewer tuples). All the remaining k-tuples match. So the two strings have at least $t - k + 1 - lk = t - (l + 1)k + 1$ k-tuples in common.

Using this lemma, and given a value of k, we can determine the number of k-tuples in each string in the database that match the given query. Strings that have very few matches can be eliminated right away, reducing the search space considerably.

Problem 6

A. We first compute the best local alignment score (and its position) using the $O(n + m)$ space solution (without keeping track of the actual alignment). For this we need to modify the recursion of the dynamic program to compute the best local alignment rather than the edit distance of the entire string. Then, knowing the position of the best local alignment, we run Hirshberg's algorithm on the corresponding substrings that give the best local alignment.

B. We need to show that for any S_1, S_2 and S_3, $D(S_1, S_2) \leq D(S_1, S_3) + D(S_2, S_3)$. We have $D(S_1, S_2) = \sum_i \delta(S_1[i], S_2[i]) \leq \sum_i \{\delta(S_1[i], S_3[i]) + \delta(S_2[i], S_3[i])\} = D(S_1, S_3) + D(S_2, S_3)$. The inequality follows from the fact that δ satisfies the triangle inequality.

   In order to find the approximately best solution, we use the following algorithm. For every S_i, we compute $E(S_i)$, and pick the one with minimum cost.

   Let T be the optimal solution and S_i be the string minimizing $D(T, S_i)$. We will show that $E(S_i) \leq 2E(T)$. Then, since we pick the string with the minimum E, our cost is at most $2E(T)$, and we are done.

   We have $D(T, S_i) \leq D(T, S_j)\, \forall j$, be definition. Using triangle inequality, $D(S_i, S_j) \leq D(S_i, T) + D(S_j, T) \leq 2D(S_j, T)$. Summing over all j, we get $E(S_i) \leq 2E(T)$.