

Sample Write-up For Dynamic Programming

September 18, 2003

Below is a sample-write up for the string alignment problem that I covered in class. You can use this as a guideline when writing up your solutions for Homework 2. If you want some additional practice problems, just let me know and I can provide them. Two good practice problems are the matrix chain multiplication and the Longest Common Subsequence problems in the text. I recommend that you read the statement of these problems and then try to solve each on your own.

Here is the problem statement for the sequence alignment problem. You are given sequence D_1 of n_1 characters and sequence D_2 of n_2 characters. An alignment is defined by inserting any number of spaces in D_1 and D_2 so that the resulting strings D'_1 and D'_2 both have the same length (with the spaces included as part of the sequence). Each character of D'_1 (including each space as a character) has a corresponding character (matching or non-matching) in the same position in D'_2 . For a particular alignment A we say $cost(A)$ is the number of mismatches (where a space but does not match any of the original characters).

In our solution the general form of the subproblem is: Find the best alignment for the first i characters of D_1 and the first j characters of D_2 for $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$. Let $D(i)$ denote be the i th character in string D . Let $c[i, j]$ be the cost of an optimal alignment for $D_1(1), \dots, D_1(i)$ and $D_2(1), \dots, D_2(j)$. We can define $c[i, j]$ recursively as shown (where $c[n_1, n_2]$ gives the optimal cost to the original problem):

$$c[i, j] = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min\{c[i-1, j-1], c[i-1, j] + 1, c[i, j-1] + 1\} & \text{if } D_1(i) = D_2(j) \\ \min\{c[i-1, j-1], c[i-1, j], c[i, j-1]\} + 1 & \text{otherwise} \end{cases}$$

We now argue this recursive definition is correct. First we argue that there exists an optimal alignment where two spaces are never aligned with each other. Suppose not. Take an optimal alignment in which two spaces are aligned with each other. Modify this alignment by just removing pairs of aligned spaces from both D'_1 and D'_2 . This solution has the same cost and hence is also optimal. Let D'_1 and D'_2 be the resulting alignment. In this alignment either the last character of D'_1 is a space or it is the last character (character i) of D_1 and the last character of D'_2 is a space or it is the last character (character j) of D_2 . Since both last characters are not a space, one of the following three cases must apply:

1. The last character of D'_1 is $D_1(i)$ and the last character of D'_2 is $D_2(j)$.
2. The last character of D'_1 is $D_1(i)$ and the last character of D'_2 is a space.
3. The last character of D'_1 is a space and the last character of D'_2 is $D_2(j)$.

The base of the recurrence (when $j = 0$ or $i = 0$) is correct since when one string is empty the optimal alignment has length equal to that of the other string since each character must be aligned with a space. Next, consider when $D_1(i) = D_2(j)$. If case 1 occurs then the cost of the alignment is $c[i-1, j-1]$ since there is no penalty for the last character of the

alignment and what remains is to align $D_1(1), \dots, D_1(i-1)$ with $D_2(1), \dots, D_2(j-1)$. If case 2 occurs, then the cost is $c[i-1, j] + 1$ since the last character is a mismatch and what remains is to align $D_1(1), \dots, D_1(i-1)$ with $D_2(1), \dots, D_2(j)$. Finally, if case 3 occurs, then the cost is $c[i, j-1] + 1$ since the last character is a mismatch and what remains is to align $D_1(1), \dots, D_1(i)$ with $D_2(1), \dots, D_2(j-1)$. Observe that in all of these cases the cost for the original problem is either equal to the cost for the subproblem or one more than the cost of the subproblem. So clearly an optimal solution to the problem of aligning the first i characters of D_1 with the first j characters of D_2 must also optimally solve the subproblem. Hence, the optimal substructure property holds. When $D_1(i) \neq D_2(j)$ the only difference is that when the first case applies, the cost is instead $c[i-1, j-1] + 1$ since there is a mismatch in the last character. Hence, the recursive solution returns the cost of the optimal solution.

The time complexity is $O(n_1 \cdot n_2)$ since there are $n_1 \cdot n_2$ subproblems each of which is solved in constant time. Finally, the $c[i, j]$ matrix can be computed in row major order.

We now describe the more efficient solution (though asymptotically the same as above) given in class. The general subproblem form is the same as above, however, the recursive solution is instead:

$$c[i, j] = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ c[i-1, j-1] & \text{if } D_1(i) = D_2(j) \\ \min\{c[i-1, j-1], c[i-1, j], c[i, j-1]\} + 1 & \text{otherwise} \end{cases}$$

The only difference between the proof of correctness here and what was given above is to prove that when $D_1(i) = D_2(j)$ that there is an optimal solution for the subproblem P of aligning the first i characters of D_1 with the first j characters of D_2 that aligns $D_1(i)$ with $D_2(j)$. This solution is interesting in that you are using dynamic programming but then can obtain a more efficient solution by applying a greedy algorithm (and a proof for the greedy choice property) in one of the cases.

We now prove that when $D_1(i) = D_2(j)$ there is an optimal solution for the subproblem P of aligning the first i characters of D_1 with the first j characters of D_2 that aligns $D_1(i)$ with $D_2(j)$. Suppose not. Take an optimal alignment A for subproblem P in which $D_1(i)$ is not aligned with $D_2(j)$. Without loss of generality, assume that in A the last character of D'_1 is a space and the last character of D'_2 is $D_2(j)$. Let index k be the index of D'_1 which contains $D_1(i)$. Consider the alignment A' that is obtained by moving $D_1(i)$ from the k th element of D'_1 to the last element of D'_1 . Observe that since $D_1(i) = D_2(j)$ there is no mismatch here whereas A did have a mismatch. If the k th element of D'_2 is a space then A' is actually a better alignment of A contradicting the optimality of A . Otherwise, a mismatch is created in the k th element of the alignment but this is offset by the mismatched removed in the last character. Thus A' is an optimal alignment in which D'_1 ends with $D_1(i)$ and D'_2 ends with $D_2(i)$.

The asymptotic time complexity does not change since the cost for computing a subproblem is still $O(1)$. However, the constant hidden in the big-oh notation is reduced here.