

Cs445 — Homework #5 Solutions

Dynamic Programming

Due: 4 April 2007 during class meeting.

- Run the LCS algorithm to find the LCS of $X = \text{"ABBACCBA"}$ and $Y = \text{"BABCA"}$. Show how you are using the the dynamic programming table, and how you are reconstructing from this table the LCS itself.

Answer:

When we run the LCS algorithm we fill in the dynamic programming table as follows. In each cell we record both the subsequence length and the direction of the choice that was made, as shown by arrows.

	ϵ	B	A	B	C	A
ϵ	0	\leftarrow^a 0	\leftarrow 0	\leftarrow 0	\leftarrow 0	\leftarrow 0
A	\uparrow 0	\uparrow 0	\swarrow^b 1	\leftarrow 1	\leftarrow 1	\swarrow 1
B	\uparrow 0	\swarrow 1	\uparrow^c 1	\swarrow 2	\leftarrow 2	\leftarrow 2
B	\uparrow 0	\swarrow 1	\uparrow 1	\swarrow^d 2	\uparrow 2	\uparrow 2
A	\uparrow 0	\uparrow 1	\swarrow 2	\uparrow^e 2	\uparrow 2	\swarrow 3
C	\uparrow 0	\uparrow 1	\uparrow 2	\uparrow^f 2	\swarrow 3	\uparrow 3
C	\uparrow 0	\uparrow 1	\uparrow 2	\uparrow 2	\swarrow^g 3	\uparrow 3
B	\uparrow 0	\swarrow 1	\uparrow 2	\swarrow 3	\uparrow^h 3	\uparrow 3
A	\uparrow 0	\uparrow 1	\swarrow 2	\uparrow 3	\uparrow 3	\swarrow^i 4

To reconstruct a subsequence of maximum length, we trace a path from lower right corner to upper left corner, following the arrows. The arrows of this path

are marked $a, b, c, d, e, f, g, h, i$. The diagonal arrows of this path (b, d, g, i) correspond to the letters of a subsequence $ABCA$, which is common to X and Y . (However, there is more than one longest common subsequence.)

2. Show how to compute $\text{LCS}(X, Y)$ using memory $\Theta(\min\{|X|, |Y|\})$ and time $O(|X| \cdot |Y|)$

Answer:

The stock LCS algorithm requires memory $\Theta(|X||Y|)$ and run time $O(|X||Y|)$. We wish to modify it to use less memory, without changing the time complexity. Notice that only two rows – the current row i and previous row $i - 1$ – are used in the computation of an element $c[i, j]$ in the memoization table. Thus we can discard data in rows $0..i - 2$. To further reduce memory requirements, we should associate the smaller of the two strings (i.e. $\min\{|X|, |Y|\}$) with the table rows. Without loss of generality, let's assume X is the shortest string; we can always swap X and Y as necessary to achieve this. Consider the following adaptation of LCS:

```

LCS(X, Y)
  /* Our table reduces to 2 arrays c of length |X| */
  for j ← 0..|X| loop
    c[0, j] ← 0
  c[1, 0] ← 0

  ci-1 ← c[0]; ci ← c[1]
  for i ← 1..|Y| loop
    for j ← 1..|X| loop
      if Y[i] = X[j] then
        ci[j] ← ci-1[j - 1] + 1
      else
        ci[j] ← max(ci-1[j], ci[j - 1])

    /* Swap pointers to overwrite obsolete row */
    ctemp ← ci-1; ci-1 ← ci; ci ← ctemp

  return ans ← ci-1

```

Analysis: The running time of our algorithm remains asymptotically unchanged. We've simply included a set of pointer-swap operations for each of the $|X|$ executions of the outer **for** loop. This step is necessary to transfer the row c_i , computed at iteration i , to the "hold" array c_{i-1} , which will be needed in the next iteration $i + 1$. The run time is thus $O((|X| + C)|Y|) = O(|X||Y|)$.

Our memory complexity is simply $2 \times |X| + C$, where X was given as the shortest string, and so our memory requirement must be $\Theta(\min\{|X|, |Y|\})$. Note that, if we wanted the actual LCS itself rather than just its length, we would still need to maintain the entire table during memoization.

3. You are given two strings X and Y , with m and n characters resp. Both strings are above the alphabet Σ . You are also given an array $Cost$ that specifies for each character $c \in \Sigma$ the **cost** of c .

Suggest an $\Theta(n^2)$ time algorithm that finds the a common subsequence of X and Y , such that the sum of costs of the characters that appears in Z is as large as possible. Here $n = |X| = |Y|$.

Answer:

We base our algorithm on LCS, and use a square table with $n+1$ cells on a side. Each cell (i, j) of the table will store the maximum cost of a subsequence of a length- i prefix of X and a length- j prefix of Y . So cell (n, n) therefore will store the maximum cost of a subsequence common to X and Y . Just like LCS, there is no common subsequence when comparing anything with an empty string, so we initialize the table by filling row 0 and column 0 with zeros. We fill in the table in the same order as for LCS, but at each cell (i, j) we compute the cell entry as follows.

If characters $X[i]$ and $Y[j]$ differ, then we can compute the max-cost subsequence of the prefixes like so:

$$Cell[i, j] \leftarrow \max(Cell[i - 1, j], Cell[i, j - 1])$$

The motivation here is the same as with LCS: we ignore the last character on one of the prefixes, and take the better solution for the smaller problem.

Whereas if $X[i] = Y[j]$ then we may choose one of the above options, or we can match the last characters of the prefix and combine that cost with the cost of the common subsequence when both prefixes are shortened by one:

$$Cell[i, j] \leftarrow \max(Cell[i - 1, j], Cell[i, j - 1], Cell[i - 1, j - 1] + Cost[X[i]])$$

The motivation here is that we don't know which choice will yield maximum cost. The $Cost$ array might have some negative entries, for all we know.

Each cell takes constant time to compute, so the algorithm needs $O(n^2)$ time to fill in all the cells. To reconstruct the actual subsequence, we store arrows in the table, and trace a path from cell (n, n) to cell $(0, 0)$; each diagonal arrow along the path indicates a character of the common subsequence. The extra effort does not increase the time complexity.

4. Let n be an **even integer**. Give an example of strings X and Y for which there are $\Omega(2^{n/2})$ different common subsequences of X and Y , both of maximal length. Assume n is an even number.

Answer:

Let string X consist of the repeating substrings “ AB ” and “ CD ”, such that $X = “ABCDABCDABCD\dots”$. Let Y consist of the substrings “ BA ” and “ DC ”, i.e. the reflections of “ AB ” and “ CD ” respectively, such that $Y = “BADCBADCBADC\dots”$.

Notice that, for the i th occurrence of the pair “ AB ” in X , we can match either “ A ” or “ B ” to a letter in the corresponding i th occurrence of “ BA ” in Y , but we can’t match both “ A ” and “ B ” to that pair simultaneously. An analogous argument applies to the pairs “ CD ”. This construction yields $\Omega(2^{n/2})$ common subsequences for X and Y .

5. Use the matrix multiplication algorithm to find the optimal way to compute A_1A_2, A_3, A_4 where $d_0 = 10, d_1 = 10, d_2 = 20, d_3 = 30, d_4 = 40$.

Answer:

We fill in a table of the costs, which ultimately yields the following:

	1	2	3	4
1	0	2000	8000	20000
2		0	6000	18000
3			0	24000
4				0

Entries $(1,1)$, $(2,2)$, $(3,3)$, and $(4,4)$ are zero, and entries $(1,2)$, $(2,3)$, and $(3,4)$ are products which do not involve any choice. The remaining entries, however, require decisions.

Entry $(1,3)$ corresponds to product $(A_1A_2)A_3$ with a cost of 8000, which is better than product $A_1(A_2A_3)$, which has a cost of 9000.

Entry $(2,4)$ corresponds to product $(A_2A_3)A_4$ with a cost of 18000, which is better than product $A_2(A_3A_4)$, which has a cost of 32000.

Entry $(1,4)$ correspond to product $((A_1A_2)A_3)A_4$ with a cost of 20000, which is better than both products $(A_1A_2)(A_3A_4)$, which costs 34000, and $A_1((A_2A_3)A_4)$, which costs 22000. This is the optimal way to compute the matrix product.

6. Run the edit distance algorithm on the following example: $X = \text{“ABBACCBA”}$ and $Y = \text{“BABCA”}$.

Answer:

There are multiple solutions to this problem. As the costs of the insert, delete, and replace operations were not explicitly given by the problem statement, we'll arbitrarily define them as follows:

$$\text{insCost} = i$$

$$\text{delCost} = d$$

$$\text{repCost} = r = d + i, \text{ where } d = i$$

The edit distance table is then given by

	""	A	B	B	A	C	C	B	A
""	0	1d	2d	3d	4d	5d	6d	7d	8d
B	1i	1i+1d	1d	2d	3d	4d	5d	6d	7d
A	2i	1i	1d+1i	2d+1i	2d	3d	4d	5d	6d
B	3i	2i	1i	1d+1i	2d+1i	3d+1i	4d+1i	4d	5d
C	4i	3i	2i	1d+2i	2d+2i	2d+1i	3d+1i	4d+1i	4d+2i
A	5i	4i	3i	1d+3i	1d+2i	2d+2i	3d+2i	4d+2i	4d+1i

An optimal editing path is shown in grey. Notice that, for this table, other optimal-cost paths are indeed possible.