CPSC 320 Notes: DP in 2-D

November 3, 2018

The Longest Common Subsequence of two strings `A` and `B` is the longest string whose letters appear in order (but not necessarily consecutively) within both `A` and `B`. For example, the LCS of `eleanor` and `naomi` is the length 2 string `no` (or equivalently the length 2 string `ao`).

(Biologists: If these were DNA base or amino acid sequences, can you imagine how this might be a useful problem?)

1. Write at least three trivial or small instances and their solutions.

Now, working backward from the end (i.e., from the last letters, as with the change-making problem where we worked from the total amount of change desired down to zero), let's figure out the first choice we make as we break the problem down into smaller pieces:

2. Consider the two strings `tycoon` and `country`. Describe the relationship of the length of their LCS with the length of the LCS of `tycoon` and `countr` (the same string `A`, and string `B` with its last letter removed).

3. Now consider the two strings `compute` and `science`. Describe the relationship of the length of their LCS with the length of the LCS of `comput` and `scienc` (strings `A` and `B` with both of their last letters removed).

4. Given two strings `A` and `B` of length $n > 0$ and $m > 0$, break the problem of finding the length of the LCS `LLCS(A[1..n], B[1..m])` down into a recurrence over smaller problems. **USE** and generalize your work in the previous problems!

```
LLCS(A[1..n], B[1..m]) =

  if _____________________________ then

    return ______________________________________________________________

  else return the ________________ of

    _______________________________________________________________________ and

    _______________________________________________________________________
```

5. Given two strings `A` and `B`, if either has a length of `0`, what is the length of their LCS?

6. The previous two problems give a recurrence to solve `LLCS`. Does this recurrence repeatedly solve suproblems many times? (That is, might we want to use memoization or dynamic programming on it?) Sketch enough of the recursion tree to justify your answer.

7. Convert your recurrence into a memoized solution to the `LLCS` problem.

8. Complete the following table to find the length of the LCS of `tycoon` and `country` using your memoized solution. (The row and column headed with an ϵ, denoting the empty string, are for the trivial cases!)

	ϵ	c	co	cou	coun	count	countr	country
ϵ								
t								
ty								
tyc								
tyco								
tycoo								
tycoon								

9. Go back to the table and extract the actual LCS from it. Circle each entry of the table you have to inspect in constructing the LCS. Then, use the space below to write an algorithm that extracts the actual LCS from an `LLCS` table.

   Hint: you **always** need to look at the lower-right corner because that represents the solution to the full problem. The recurrence used to compute this entry references either a single one, or two other entries. Which one(s)? Which entry was the one the recurrence "chose"? What does that choice mean in terms of the actual solution?

```
// Note: len(A) = n, len(B) = m, and Table is a filled-in
//       (n+1)x(m+1) LLCS memoization table for A and B
ExplainLCS(A, B, Table):
```

10. Give an iterative dynamic programming solution that produces the same table as the memoized solution.

11. Analyze the efficiency of your memoized (question 7), DP (question 10), and "explain" (question 9) algorithms in terms of runtime and memory use (not including the space used by the parameters). You may assume the strings are of length n and m, where $n \leq m$ (without loss of generality).

12. If we only want the **length** of the LCS of `A` and `B` with lengths n and m, where $n \leq m$, explain how we can "get away" with using only $O(n)$ memory in the dynamic programming solution.

1 Challenge

1. Give a LCS algorithm that runs in the same asymptotic runtime as the one above, uses only $O(m+n)$ space (note that this is potentially more than the "space-efficient" version mentioned above), and returns not only the length of the LCS but the LCS itself. (Note: try this for yourself for a while, and then walk through the description of the awesome algorithm in section 6.7 if you need help.)

2. Now, let's use a different approach to save memory. WLOG, assume that the shorter string is of length n and the longer of length m. Say that we know the LCS of the two strings is only k letters shorter than the shorter of the two strings, i.e., length $n - k$. Give a LCS algorithm that uses space in $O(km)$