CPSC 320 Notes: DP in 2-D

March 11, 2015

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 `snow` and `naomi` is the length 2 string `no`.

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

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

2. Now consider the two strings `stable` and `marriage`. Describe the relationship of their LCS with the LCS of `stabl` and `marriag` (strings `A` and string `B` with their last letters removed).

3. Given two strings `A` and `B` of length $n > 0$ and $m > 0$, break the problem of finding the `LCS(A[1..n], B[1..m])` down into a recurrence over smaller problems:

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

  the ________________ of

    ___________________________________________,

    ___________________________________________, and

    ___________________________________________
```

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

5. Complete the following table to find the length of the LCS of `tycoon` and `country` using dynamic programming and your recurrence:

	c	o	u	n	t	r	y
t							
y							
c							
o							
o							
n							

6. 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.

7. Analyse the efficiency of your algorithm in terms of runtime and (additional, beyond the parameters) memory use. You may assume the strings are of length n and m, where $n \leq m$ (without loss of generality).

8. 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.

1 Challenge

1. **Prove** that if two strings end in the same letter, you can ignore all but one of the "options" in your recurrence.

2. 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.)