Discussion 6

ECS 122A
May 8, 2019

I. A "Recipe" for Dynamic Programming

1. Characterize the structure of an optimal solution, and recursively define the value of an optimal solution. In other word, come up with a **formula**
2. Compute the value of an optimal solution in a **bottom-up** fashion, and make use of the computed information (**momoization**)

Recap: Largest Common Subsequence Problem

II. Dynamic Programming VS Divide-and-conquer

Dynamic Programming	**Divide-and-conquer**
Subproblems **overlap**	Subproblems are **disjoint**, mostly smaller instances of the **same** type
Use a lookup table and traceback table (memoization)	Solve the subproblems recursively
Bottom-up	Top-down

In the context of subproblems sharing subsubproblems, divide-and-conquer algorithm does more work than necessary, repeatedly solving the common subsubproblems while dynamic-programming algorithm solves each subsubproblem just once and then saves its answer in a lookup table, thereby avoiding the work of recomputing the answer every time it solves each subsubproblem. We can conclude that dynamic programming is a good approach to **optimization problem**, such as *max* and *min*.

Discussion 6

III. Dynamic Programming: Image Compression (Textbook Problem 15-8)

We are given a color picture consisting of an $m \times n$ array $A[1...m, 1...n]$ of pixels, where each pixel specifies a triple of red, green, and blue (RGB) intensities. Suppose that we wish to compress this picture slightly. Specifically, we wish to remove one pixel from each of the m rows, so that the whole picture becomes one pixel narrower (namely, $m \times n - 1$). To avoid disturbing visual effects, however, we require that the pixels removed in two adjacent rows be in the same or adjacent columns; the pixels removed form a "seam" from the top row to the bottom row where successive pixels in the seam are adjacent vertically or diagonally.

(a) Show that the number of such possible seams grows at least exponentially in m, assuming that $n > 1$.

(b) Suppose now that along with each pixel $A[i, j]$, we have calculated a real valued disruption measure $d[i, j]$ indicating how disruptive it would be to remove pixel $A[i, j]$. Intuitively, the lower a pixel's disruption measure, the more similar the pixel is to its neighbors. Suppose further that we define the disruption measure of a seam to be the sum of the disruption measures of its pixels.

Give an algorithm to find a seam with the lowest disruption measure. How efficient is your algorithm?

(*Hint*: If $A[i, j]$ is in the optimal seam, who else is in that seam too?)