Carlos Mc Gregor c.mcgregormuro@mail.utoronto.ca

CSC373S: Algorithm Design, Analysis & Complexity

Lecture 09

Wednesday January 25, 2017
based on notes by Denis Pankratov

Dynamic Programming (*continue*)

Elements of a DP Problem

1. Optimal substructure prop.
2. Overlapping subproblems prop.

Elements of a DP Solution:

1. Semantic array
2. Computation array
3. Correctness: equivalence of (1) & (2)
4. Pseudocode
5. Runtime

Implementation:

1. *Iteratively*: DP table
   - Sometimes faster because you avoid overhead of function calls
   - Allows additional memory optimization techniques
2. *Recursively*: Memoization
   - Easier to code (because you do not need to figure out order in which you will implement the elements)
   - Sometimes faster because you solve only subproblem needed for the final answer

Longest Increasing Subsequence

Input: A - array of n integers
Output: Longest Increasing Subsequence [LIS] (not necessarily contiguous) of A

Example: [5, **2**, 8, 6, **3**, **6**, **9**, 7] $= A$, largest LIS is $[2, 3, 6, 9]$.

Semantic Array:
$C[i] =$ length LIS from $A[1], \ldots, A[i]$ that ends in $A[i]$.
Answer: $\max_{1 \leq i \leq n} C[i]$

Computational Array:
$C[i] = 1 + \max\{C[j] | j < i \& A[j] < A[i]\}$ (as a convention, $\max \varnothing = 0$)

Correctness:
LIS ending in $A[i]$ includes $A[i]$ & LIS ending in $A[j]$ for $j < i$ & $A[j] < A[i]$ by cut & paste.

Optimal Substructure Argument:

```
def LIS(A, n):
    init C, P - arrays of length n each
    # P[i] = Index j such that LIS A[1..i]
    #          is A[i] + LIS A[1..j] if such j exists.
    #          Index i, otherwise.
    for i = 1 to n:
        C[i] = 1
        P[i] = 1
        for j = 1 to i - 1:
            if A[j] < A[i] and 1 + C[j] > C[i]:
                C[i] = 1 + C[j]
                P[i] = j

    # ml is max value in C, k is its index
    ml = 1
    k = 1

    # Longest subsequence ends in k
    for i = 2 to n:
        if C[i] > ml:
            ml = C[i]
            k = i

    sol = list w/ element {k}
    while P[k] != k:
        # Append in front new values as they arrive
        sol.prepend(P[k])
```

```
        k = P[k]

    return sol
```

<u>Runtime:</u> $\mathcal{O}(n^2)$
<u>Exercise:</u> find a different DP solution w/ runtime $\mathcal{O}(n \log n)$.

Longest Common Subsequence (LCS)

Input: X - string w/ m characters, Y - string w/ n characters
Output: Longest Common Subsequence (LCS)

<u>Example:</u>

$$X = [\text{S}, \textbf{P}, \text{R}, \textbf{I}, \textbf{N}, \text{G}, \text{T}, \text{I}, \text{M}, \textbf{E}]$$
$$Y = [\textbf{P}, \textbf{I}, \text{O}, \textbf{N}, \textbf{E}, \text{E}, \text{R}]$$

LCS = **PINE**.
Semantic Array:
$C[i,j]$ = length LCS of $X[1..i], Y[1..j], 0 \leq i \leq m, 0 \leq j \leq n$
<u>Answer:</u> $C[m,n]$

Computational Array:

$$C[i,j] = \max \begin{cases} 1 + C[i-1, j-1], & \text{if } X[i] = Y[j] \\ C[i-j, j] \\ C[i, j-1] \end{cases}$$

<u>Base Case:</u> $C[i,0] = C[0,j] = 0, 0 \leq i \leq m, 0 \leq j \leq n$.
<u>Equivalence:</u>
LCS $X[1..i]$ and $Y[1..j]$ either ends in $X[i]$ (possible if $X[i] = Y[j]$), which implies LCS $X[1..i-1]$ and $Y[1..j-1]$, or does not end in $X[i] \rightarrow$ LCS $X[1..i-1]\&Y[1..j]$, or does not end in $Y[j] \rightarrow$ LCS $X[1..i]\&Y[1..j-1]$.

<u>Algorithm:</u>

```
def LCS(X, Y, m, n):
  init array C of size (m + 1) * (n + 1), 0-based index

  for i = 0 to m:
    C[i, 0] = 0

  for j = 0 to n:
    C[0, j] = 0
```

```
    for i = 1 to m:
        for j = 1 to n:
            C[i, j] = 0
            if X[i] = Y[j]:
                C[i, j] = 1 + C[i-1, j-1]
            C[i, j] = max(C[i,j], C[i-1, j], C[i, j-1])

    return C[m,n]
```

Runtime: $\mathcal{O}(mn)$
Exercise: add book-keeping to construct LCS itself (*see Figure 1 below*)

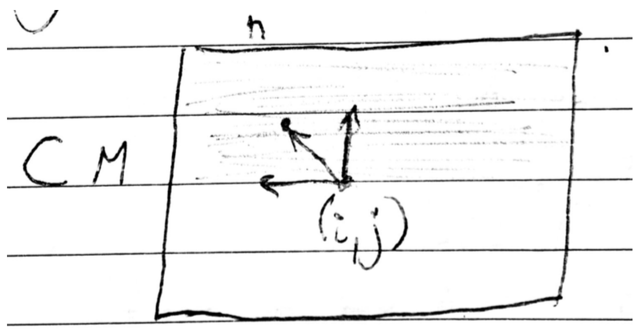

Figure 1: Visualization of Book-Keeping construction of LCS