CSE 373: Analysis of Algorithms

Lectures 23 & 24
(Dynamic Programming)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2014

Dynamic Programming vs. Divide-and-Conquer

- Dynamic programming, like the divide-and-conquer method, solves problems by combining solutions to subproblems
- Divide-and-conquer algorithms
  - partition the problem into disjoint subproblems,
  - solve the subproblems recursively, and
  - then combine their solutions to solve the original problem
- In contrast, dynamic programming applies when the subproblems overlap — that is, when subproblems share subsubproblems
- A dynamic-programming algorithm solves each subsubproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time it solves each subsubproblem

Dynamic Programming

When developing a dynamic-programming algorithm, we follow a sequence of four steps:

1) Characterize the structure of an optimal solution.
2) Recursively define the value of an optimal solution.
3) Compute the value of an optimal solution, typically in a bottom-up fashion.
4) Construct an optimal solution from computed information.

If we need only the value of an optimal solution, and not the solution itself, then we can omit step 4.

If we perform step 4, we sometimes maintain additional information during step 3 so that we can easily construct an optimal solution.

Longest Common Subsequence (LCS)

A *subsequence* of a sequence *X* is obtained by deleting zero or more symbols from *X*.

Example:

$X =$ *abcba*

$Z =$ *bca* $\leftarrow$ obtained by deleting the 1st '*a*' and the 2nd '*b*' from *X*

A *Longest Common Subsequence* (*LCS*) of two sequence *X* and *Y* is a sequence *Z* that is a subsequence of both *X* and *Y*, and is the longest among all such subsequences.

Given *X* and *Y*, the *LCS problem* asks for such a *Z*.

Optimal Substructure of an LCS

Given two sequences: $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$

Let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y.

For $0 \le i \le m$, let $X_i = \langle x_1, x_2, \dots, x_i \rangle$. We define Y_i and Z_i similarly.

Then

(1) If $x_m = y_n$,
then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.

(2) If $x_m \neq y_n$,
then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.

(3) If $x_m \neq y_n$,
then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

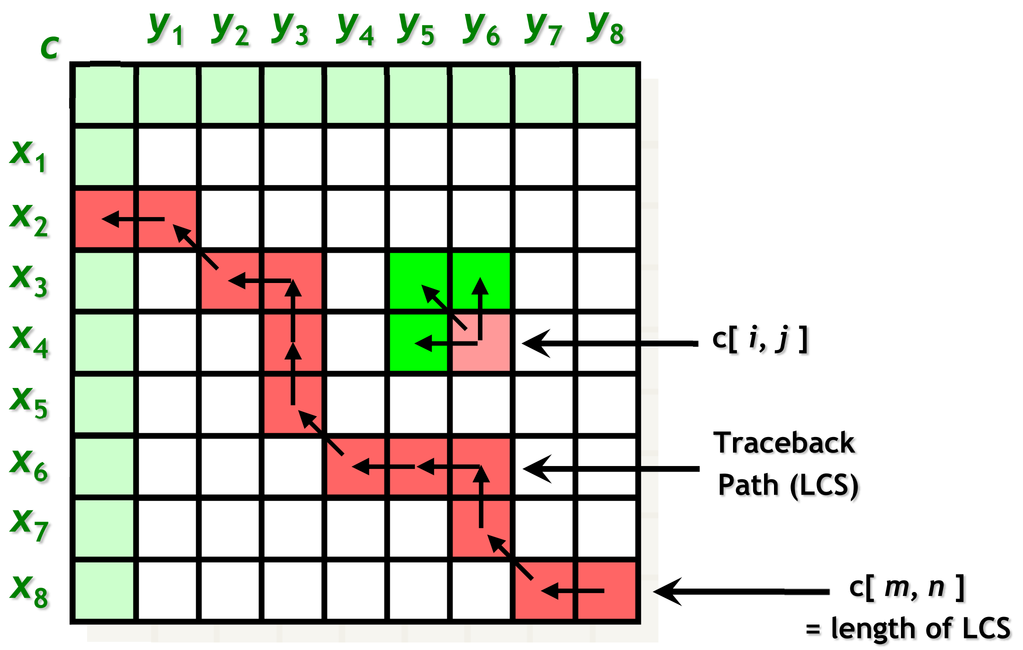
The LCS Recurrence

Given two sequences: $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$

For $0 \le i \le m$ and $0 \le j \le n$,

let $c[i,j]$ be the length of an LCS of X_i and Y_j. Then

$$c[i,j] = \begin{cases} 0, & if\ i = 0 \lor j = 0, \\ c[i-1, j-1] + 1, & if\ i,j > 0 \land x_i = y_j, \\ \max\{c[i, j-1], c[i-1, j]\}, & otherwise. \end{cases}$$



Computing the Length of an LCS

```
LCS-LENGTH ( X, Y )
1.  $m = X.length$
2.  $n = Y.length$
3.  let $b[1 \dots m, 1 \dots n]$ and $c[0 \dots m, 0 \dots n]$ be new tables
4.  for $i = 1$ to $m$
5.      $c[i, 0] = 0$
6.  for $j = 0$ to $n$
7.      $c[0, j] = 0$
8.  for $i = 1$ to $m$
9.      for $j = 1$ to $n$
10.         if $x_i == y_j$
11.             $c[i, j] = c[i-1, j-1] + 1$
12.             $b[i, j] =$ “$\nwarrow$”
13.         elseif $c[i-1, j] \geq c[i, j-1]$
14.             $c[i, j] = c[i-1, j]$
15.             $b[i, j] =$ “$\uparrow$”
16.         else $c[i, j] = c[i, j-1]$
17.             $b[i, j] =$ “$\leftarrow$”
```

Running time $= \Theta(mn)$

Computing the Length of an LCS

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i							
1	A							
2	B							
3	C							
4	B							
5	D							
6	A							
7	B							

Computing the Length of an LCS

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

Computing the Length of an LCS

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Computing the Length of an LCS

	j	0	1	2	3	4	5	6
i		y_j	Ⓑ	D	Ⓒ	A	Ⓑ	Ⓐ
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	Ⓑ	0	↖ ①	← 1	← 1	↑ 1	↖ 2	← 2
3	Ⓒ	0	↑ 1	↑ 1	↖ ②	← 2	↑ 2	↑ 2
4	Ⓑ	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ ③	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	Ⓐ	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ ④
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Constructing an LCS

```
PRINT-LCS ( b, X, i, j )

1.  if $i == 0$ or $j == 0$
2.      return
3.  if $b[i,j] ==$ “↖”
4.          PRINT-LCS ( $b, X, i-1, j-1$ )
5.          print $x_i$
6.  elseif $b[i,j] ==$ “↑”
7.          PRINT-LCS ( $b, X, i-1, j$ )
8.  else PRINT-LCS ( $b, X, i, j-1$ )
```

Running time $= O(m+n)$