Algorithm Design and Analysis

CSE 565

LECTURE 20

Dynamic Programming

- LCS

Sofya Raskhodnikova

10/12/2007

S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

Longest Common Subsequence (LCS)

- Given two sequences $x[1 \,..\, m]$ and $y[1 \,..\, n]$, find a longest subsequence common to them both.

"a" *not* "the"

x: A B C B D A B

y: B D C A B A

BCBA = LCS(x, y)

10/12/2007

S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

Brute-force LCS algorithm

Check every subsequence of $x[1 \,..\, m]$ to see if it is also a subsequence of $y[1 \,..\, n]$.

Analysis

- Checking = $O(n)$ time per subsequence.
- 2^m subsequences of x (each bit-vector of length m determines a distinct subsequence of x).

Worst-case running time = $O(n2^m)$

= exponential time.

10/12/2007

S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

Dynamic programming algorithm

Simplification:

1. Look at the *length* of a longest-common subsequence.
2. Extend the algorithm to find the LCS itself.

Notation: Denote the length of a sequence s by $|s|$.

Strategy: Consider ***prefixes*** of x and y.

- Define $c[i, j] = |\mathrm{LCS}(x[1 \,..\, i], y[1 \,..\, j])|$.
- Then, $c[m, n] = |\mathrm{LCS}(x, y)|$.

10/12/2007

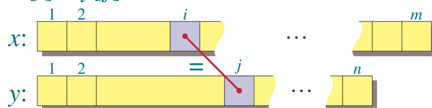
S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

Recursive formulation

$$c[i, j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x[i] = y[j], \\ \max\{c[i-1, j], c[i, j-1]\} & \text{otherwise.} \end{cases}$$

Base case: $c[i,j]=0$ if $i=0$ or $j=0$.

Case $x[i] = y[j]$:



The second case is similar.

10/12/2007

S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

Dynamic-programming hallmark #1

Optimal substructure

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

If $z = \mathrm{LCS}(x, y)$, then any prefix of z is an LCS of a prefix of x and a prefix of y.

10/12/2007

S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

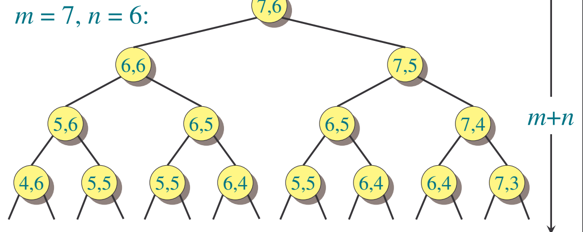
Recursive algorithm for LCS

LCS(x, y, i, j) // ignoring base cases
  if $x[i] = y[j]$
    then $c[i, j] \leftarrow$ LCS$(x, y, i-1, j-1) + 1$
    else $c[i, j] \leftarrow \max\{$ LCS$(x, y, i-1, j)$,
                LCS$(x, y, i, j-1)\}$
  return $c[i, j]$

Worse case: $x[i] \neq y[j]$, in which case the algorithm evaluates two subproblems, each with only one parameter decremented.

10/12/2007 — S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

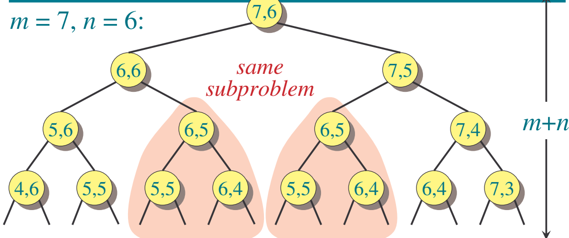
Recursion tree



Height = $m + n \Rightarrow$ work potentially exponential.

10/12/2007 — S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

Recursion tree



Height = $m + n \Rightarrow$ work potentially exponential, but we're solving subproblems already solved!

10/12/2007 — S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

Dynamic-programming hallmark #2

Overlapping subproblems
A recursive solution contains a "small" number of distinct subproblems repeated many times.

The number of distinct LCS subproblems for two strings of lengths m and n is only mn.

10/12/2007 — S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

Memoization algorithm

Memoization: After computing a solution to a subproblem, store it in a table. Subsequent calls check the table to avoid redoing work.

LCS(x, y, i, j)
  if $c[i, j] =$ NIL
    then if $x[i] = y[j]$
      then $c[i, j] \leftarrow$ LCS$(x, y, i-1, j-1) + 1$
      else $c[i, j] \leftarrow \max\{$ LCS$(x, y, i-1, j)$,
               LCS$(x, y, i, j-1)\}$
same as before

Time = $\Theta(mn)$ = constant work per table entry.
Space = $\Theta(mn)$.

10/12/2007 — S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

Dynamic-programming algorithm

IDEA:
Compute the table bottom-up.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

10/12/2007 — S. Raskhodnikova; based on slides by C. Leiserson and E. Demaine

CSE 565

Dynamic-programming algorithm

IDEA:

Compute the table bottom-up.

Time = $\Theta(mn)$.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

CSE 565

Dynamic-programming algorithm

IDEA:

Compute the table bottom-up.

Time = $\Theta(mn)$.

Reconstruct LCS by tracing backwards.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

CSE 565

Dynamic-programming algorithm

IDEA:

Compute the table bottom-up.

Time = $\Theta(mn)$.

Reconstruct LCS by tracing backwards.

Space = $\Theta(mn)$.

Exercise: $O(\min\{m, n\})$.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4