CSE 6331: Dynamic Programming - Longest Common Subsequence

R. Wenger

Divide and Conquer

Divide and Conquer:

- Divide the problem into subproblems;
- Solve each of the subproblems;
- Combine to solve the original problem.

Dynamic Programming: Basic Idea

Dynamic Programmming:

- Divide the problem into MULTIPLE COMBINATIONS of subproblems;
- Solve each of the subproblems;
- Take the "best" combination of subproblems to solve the original problem.

Dynamic Programming: Overview

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 the computed information.

Longest Common Subsequence

Subsequence

Sequence:

$$A = (\cancel{A}, \cancel{A}, T, G, \cancel{C}, T, \cancel{A}, C, \cancel{A}, A, C).$$

Definition: **Subsequence** of A is a sequence derived from A by deleting elements from A without changing the order of the remaining elements.

Examples of subsequences of A:

(A, A, T, G, C)

(C, A, A, C)

(T, G, T, C, A, C)

(A, A, A, A, A)

(T)

$()$.

Common Subsequence

Sequences:

$$A = (A, A, T, G, C, T, A, C, A, A, C).$$
$$B = (C, A, A, A, G, C, C, G, A, G, C, T).$$

Definition: A **common subsequence** of A and B is a subsequence of both A and B.

Examples of common subsequences of A and B:

(A, C, T)
(A, G, A)
(T, G, A, C)
(A, A, A, A)
(T)
$()$.

Adenine

Cytosine

Guanine

Thymine

Longest Common Subsequence

Given two sequences:

$$A = (a_1, a_2, \ldots, a_n),$$
$$B = (b_1, b_2, \ldots, b_m),$$

find a longest common subsequence of A and B.

Example:

$$A = (A, A, T, G, C, T, A, C, A, A, C)$$
$$B = (C, A, A, A, G, C, C, G, A, G, C, T)$$

A, A, G, C, A, C

Longest Common Subsequence: Example 1

What is the longest common subsequence $\texttt{LCS}(A, B)$ of:

$$A = (A, A, T, G, C, T, A, C, A, A, C)$$
$$B = (C, A, A, A, G, C, C, G, A, G, C, T)$$

A, A, G, C, C, A C

Longest Common Subsequence: Example 2

What is the longest common subsequence ($\texttt{LCS}(A, B)$) of:

$$A = (C, A, T, C)$$
$$B = (A, T, A, C, G, C, A).$$

$A' = (C, A, T)$

$B' = (A, T, A, C, G, C)$

$LCS(A, B) = LCS(A', B)$ or $LCS(A, B')$

Longest Common Subsequence: Example 2

What is the longest common subsequence ($\mathtt{LCS}(A, B)$) of:

$$A = (C, A, T, C)$$
$$B = (A, T, A, C, G, C).$$

$A' = C, A, T$

$B' = A, T, A, C, G$

$\mathrm{LCS}(A, B) =$ either $\cancel{\mathrm{LCS}(A', B)}$

or $\cancel{\mathrm{LCS}(A, B')}$

or $\mathrm{LCS}(A', B') \circ (C)$

Longest Common Subsequence: Example 2

What is the longest common subsequence ($\texttt{LCS}(A, B)$) of:

$$A = (C, A, T, C)$$
$$B = (A, T, A, C, G, C).$$

A': C A T

B': A T A C G

$LCS(A, B) = LCS(A', B') \circ \{C\}$

Longest Common Subsequence: Recurrence Relation

What is longest common subsequence $\texttt{LCS}(A, B)$ of

$$A = (a_1, a_2, \ldots, a_n),$$
$$B = (b_1, b_2, \ldots, b_m).$$

If $a_n \neq b_m$, then:

- $\texttt{LCS}(A, B) = LCS(A, B')$ where $B' = (b_1, b_2, \ldots, b_{m-1})$, OR
- $\texttt{LCS}(A, B) = LCS(A', B)$ where $A' = (a_1, a_2, \ldots, a_{n-1})$.

Longest Common Subsequence: Recurrence Relation

What is longest common subsequence $\texttt{LCS}(A, B)$ of

$$A = (a_1, a_2, \ldots, a_n),$$
$$B = (b_1, b_2, \ldots, b_m).$$

If $a_n \neq b_m$, then:

- $\texttt{LCS}(A, B) = LCS(A, B')$ where $B' = (b_1, b_2, \ldots, b_{m-1})$, OR
- $\texttt{LCS}(A, B) = LCS(A', B)$ where $A' = (a_1, a_2, \ldots, a_{n-1})$.

If $a_n = b_m$, then

- $LCS(A, B) = LCS(A', B') \circ (a_n)$ where:
  $A' = (a_1, a_2, \ldots, a_{n-1})$ and $B' = (b_1, b_2, \ldots, b_{m-1})$.

Longest Common Subsequence: Recurrence Relation

$$A = (a_1, a_2, \ldots, a_n),$$
$$B = (b_1, b_2, \ldots, b_m).$$

Let $L(i,j)$ denote the length of the longest common subsequence of $A_i = (a_1, a_2, \ldots, a_i)$ and $B_j = (b_1, b_2, \ldots, b_j)$.

Assume $i \geq 1$ and $j \geq 1$.

If $a_i \neq b_j$, then:

- $\texttt{LCS}(A_i, B_j) = LCS(A_i, B_{j-1})$ OR
- $\texttt{LCS}(A_i, B_j) = LCS(A_{i-1}, B_j)$.
- $L(i,j) =$ $\max(L(i, j-1), L(i-1, j))$

Longest Common Subsequence: Recurrence Relation

$$A = (a_1, a_2, \ldots, a_n),$$
$$B = (b_1, b_2, \ldots, b_m).$$

Let $L(i, j)$ denote the length of the longest common subsequence of $A_i = (a_1, a_2, \ldots, a_i)$ and $B_j = (b_1, b_2, \ldots, b_j)$.

Assume $i \geq 1$ and $j \geq 1$.

If $a_i = b_j$, then:

- $\texttt{LCS}(A_i, B_j) = LCS(A_{i-1}, B_{j-1})$. $\circ\,(a_i)$
- $L(i, j) =$ $L(i-1, j-1) + 1$

Longest Common Subsequence: Boundary Conditions

$$A = (a_1, a_2, \ldots, a_n),$$
$$B = (b_1, b_2, \ldots, b_m).$$

Let $L(i, j)$ denote the length of the longest common subsequence of $A_i = (a_1, a_2, \ldots, a_i)$ and $B_j = (b_1, b_2, \ldots, b_j)$.

If $i = 0$ or $j = 0$, then:

- $\texttt{LCS}(A_i, B_j) = (\ )$
- $L(i, j) = 0$

Longest Common Subsequenc: Recurrence Relation

$$A_i = (a_1, a_2, \ldots, a_i),$$
$$B_j = (b_1, b_2, \ldots, b_j).$$

For $i \geq 1$ and $j \geq 1$:

$$L(i,j) = \begin{cases} \max(L(i-1,j), L(i,j-1)) & \text{if } a_i \neq a_j \\ L(i-1,j-1)+1 & \text{if } a_i = a_j. \end{cases}$$

For all i and j:

$$L(i,0) = 0.$$
$$L(0,j) = 0.$$

Dynamic Programming: Overview

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 the computed information.

Longest Common Subsequence: Algorithm

$$L(i,j) = \begin{cases} \max(L(i-1,j), L(i,j-1)) & \text{if } a_i \neq a_j \\ L(i-1,j-1)+1 & \text{if } a_i = a_j. \end{cases}$$

```
procedure LongestCommonSubsequence(A[ ], n, B[ ], m)
/* Arrays A[ ] and B[ ] represent sequences of length n and m   */
for i = 0 to n do  L[i, 0] ← 0;
for j = 0 to m do  L[0, j] ← 0;
for i = 1 to n do
    for j = 1 to m do
        if (A[i] = B[j]) then  L[i, j] = 1 + L[i − 1, j − 1] ;
        else  L(i, j) = max(L[i − 1, j], L[i, j − 1]) ;
    end
end
return L(n, m);
```

Longest Common Subsequence: Table

$L[i,j]$	0	1	2	3	...	$m-1$	m
0	0	0	0	0	...	0	0
1	0						
2	0						
3	0						
$\vdots$	$\vdots$						
$n-1$	0						
n	0						

Longest Common Subsequence: Table

	$L[i,j]$	0	C 1	A 2	T 3	C 4
	0	0	0	0	0	0
A	1	0	↑ 0	↖ 1	← 1	← 1
T	2	0	↑ 0	↑ 1	↖ 2	← 2
A	3	0	↑ 0	↖ 1	↑ 2	↑ 2
C	4	0	↖ 1	↑ 1	↑ 2	↖ 3
G	5	0				
C	6	0				
A	7	0				

Longest Common Subsequence: Running time

$$L(i,j) = \begin{cases} \max(L(i-1,j), L(i,j-1)) & \text{if } a_i \neq a_j \\ L(i-1,j-1)+1 & \text{if } a_i = a_j. \end{cases}$$

procedure `LongestCommonSubsequence`$(A[\,], n, B[\,], m)$

/ Arrays $A[\,]$ and $B[\,]$ represent sequences of length n and m */*

for $i = 0$ **to** n **do** $L[i,0] \leftarrow 0$;

for $j = 0$ **to** m **do** $L[0,j] \leftarrow 0$;

for $i = 1$ **to** n **do** $\leftarrow n$

  for $j = 1$ **to** m **do** $\leftarrow m$

    if $(A[i] = B[j])$ **then** $L[i,j] = 1 + L[i-1,j-1]$;

    else $L(i,j) = \max(L[i-1,j], L[i,j-1])$;

  end

end

return $L(n,m)$;

Running time: $C\,n\,m \cong n^2$ if $n \approx m$

Longest Common Subsequence: Print Subsequence

procedure LongestCommonSubsequence($A[\,], n, B[\,], m, L[\,;\,]$)
/* *Arrays* $A[\,]$ *and* $B[\,]$ *represent sequences of length* n *and* m */
/* $L[i,j]$ = *length of LCS of* $A_i[\,]$ *and* $B_j[\,]$ */

$S \leftarrow \emptyset$; /* *S is a stack* */
$i \leftarrow n$;
$j \leftarrow m$;
while $(i > 0)$ **and** $(j > 0)$ **do**
  if $(A[i] = B[j])$ **then** S.Push($A[i]$) ; $i \leftarrow i-1$; $j \leftarrow j-1$
  else if $L[i-1,j] > L[i,j-1]$ **then** $i \leftarrow i-1$;
  else $j \leftarrow j-1$; /* $L[i-1,j] \leq L[i,j-1]$ */
end
while $(S \neq \emptyset)$ **do**
  $x \leftarrow S$.Pop();
  Print x;
end

LCS: Print Subsequence - Running Time

procedure `LongestCommonSubsequence`($A[\,]$, n, $B[\,]$, m, $L[\,;\,]$)

/* *Arrays* $A[\,]$ *and* $B[\,]$ *represent sequences of length* n *and* m */

/* $L[i,j]$ = *length of LCS of* $A_i[\,]$ *and* $B_j[\,]$ */

$S \leftarrow \emptyset$; /* *S is a stack* */

$i \leftarrow n$;

$j \leftarrow m$;

while $(i > 0)$ **and** $(j > 0)$ **do**

  if $(A[i] = B[j])$ **then** S.`Push`($A[i]$) ; $i \leftarrow i-1$ $j \leftarrow j-1$

  else if $L[i-1,j] > L[i,j-1]$ **then** $i \leftarrow i-1$;

  else $j \leftarrow j-1$; /* $L[i-1,j] \leq L[i,j-1]$ */

end

while $(S \neq \emptyset)$ **do**

  $x \leftarrow S.\mathrm{Pop}()$;

  Print x;

end

Running time: $c(n+m) \in \Theta(n+m)$

Longest Common Subsequence: Table

	$L[i,j]$	0	C 1	A 2	T 3	C 4
	0	0	0	0	0	0
A	1	0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1
T	2	0	$\uparrow$ 0	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
A	3	0	$\uparrow$ 0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2
C	4	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\nwarrow$ 3
G	5	0	$\uparrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3
C	6	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\nwarrow$ 3
A	7	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 3

ATC

Longest Common Subsequence: Summary

$$
\begin{aligned}
A &= (a_1, a_2, \ldots, a_n) \\
B &= (b_1, b_2, \ldots, b_m) \\
A' &= (a_1, a_2, \ldots, a_{n-1}) \\
B' &= (b_1, b_2, \ldots, b_{m-1})
\end{aligned}
$$

- If $a_n \neq b_m$, then $LCS(A, B)$ equals $LCS(A, B')$ or $LCS(A', B)$.
- If $a_n = b_m$, then $LCS(A, B) = LCS(A', B') \circ (a_n)$.
- $$L(i,j) = \begin{cases} \max(L(i-1,j), L(i,j-1)) & \text{if } a_i \neq a_j \\ L(i-1,j-1)+1 & \text{if } a_i = a_j. \end{cases}$$
- Computing $L(n,m)$ and $LCS(A,B)$ takes $\Theta(nm)$ time.

Dynamic Programming - Memoization

Longest Common Subsequence: ~~Boundary Conditions~~

$$A = (a_1, a_2, \ldots, a_n),$$
$$B = (b_1, b_2, \ldots, b_m).$$

Let $L(i, j)$ denote the length of the longest common subsequence of $A_i = (a_1, a_2, \ldots, a_i)$ and $B_j = (b_1, b_2, \ldots, b_j)$.

If $i = 0$ or $j = 0$, then:

$$L(i, j) = 0.$$

If $i \geq 1$ and $j \geq 1$:

$$L(i, j) = \begin{cases} \max(L(i-1, j), L(i, j-1)) & \text{if } a_i \neq a_j \\ L(i-1, j-1) + 1 & \text{if } a_i = a_j. \end{cases}$$

Longest Common Subsequence: Recursive Algorithm

NOT DYNAMIC PROGRAMMING.
DO NOT USE THIS ALGORITHM.

```
procedure LCSRecursive(A[ ], n, B[ ], m)
/* Arrays A[ ] and B[ ] represent sequences of length n and m */
if (n = 0) or (m = 0) then return 0;
if (A[n] = B[m]) then
    k ← 1+ LCSRecursive(A, n − 1, B, m − 1);
    return k;
else
    k ← LCSRecursive(A, n − 1, B, m);
    k′ ← LCSRecursive(A, n, B, m − 1);
    return max(k, k′);
end
```

LCS: Recursive Algorithm - Recurrence Relation

NOT DYNAMIC PROGRAMMING.
DO NOT USE THIS ALGORITHM.

```
procedure LCSRecursive(A[ ], n, B[ ], m)
/* Arrays A[ ] and B[ ] represent sequences of length n and m          */
if (n = 0) or (m = 0) then return 0;
if (A[n] = B[m]) then
    k ← 1+ LCSRecursive(A, n − 1, B, m − 1);
    return k;
else
    k ← LCSRecursive(A, n − 1, B, m);
    k′ ← LCSRecursive(A, n, B, m − 1);
    return max(k, k′);
end
```

Worst case running time:
$T(n, m) = c + T(n-1, m) + T(n, m-1)$

LCS: Recursive Algorithm - Running Time

$T(n, m) = c + T(n-1, m) + T(n, m-1)$

$T(n-1, m) = c + T(n-2, m) + T(n-1, m-1) \leftarrow$

$T(n, m-1) = c + T(n-1, m-1) + T(n, m-2) \leftarrow$

$$\begin{aligned} T(n, m) &= c + T(n-1, m) + T(n, m-1) \\ &= c + (c + T(n-2, m) + T(n-1, m-1)) \\ &\quad + (c + T(n-1, m-1) + T(n, m-2)) \\ &\geq 2T(n-1, m-1) \end{aligned}$$

LCS: Recursive Algorithm - Running Time (cont)

$$T(n, m) \geq 2T(n-1, m-1)$$

Assuming $m = n$

$$T(n, n) \geq 2T(n-1, n-1)$$

$$\geq 2 \cdot 2T(n-2, n-2)$$

$$\geq \underbrace{2 \cdot 2 \cdots \quad 2}_{2^n} \cdot \cancel{T(0, 0)} \; c$$

$$2^n \in \Omega(2^n)$$

LCS: Recursive Algorithm

NOT DYNAMIC PROGRAMMING.
DO NOT USE THIS ALGORITHM.

procedure `LCSRecursive`($A[\,]$, n, $B[\,]$, m)
/* *Arrays $A[\,]$ and $B[\,]$ represent sequences of length n and m* */
if $(n = 0)$ **or** $(m = 0)$ **then return** 0;
if $(A[n] = B[m])$ **then**
  $k \leftarrow 1+$ `LCSRecursive`(A, $n-1$, B, $m-1$);
  return k;
else
  $k \leftarrow$ `LCSRecursive`(A, $n-1$, B, m);
  $k' \leftarrow$ `LCSRecursive`(A, n, B, $m-1$);
  return $\max(k, k')$;
end

Worst case running time:
$T(n, m) =$

LCS: Recursive Algorithm - Running Time

$L[i,j]$	0	1	2	3	...	$m-1$	m
0	0	0	0	0	...	0	0
1	0						
2	0						
3	0						
⋮	⋮						
$n-1$	0						
n	0						

LCS: Memoized Algorithm

procedure LCSMemoized($A[\,]$, n, $B[\,]$, m, $L[\,;\,]$)

/* *Arrays $A[\,]$ and $B[\,]$ represent sequences of length n and m* */

/* $L[i,j]$ = *length of LCS of $A_i[\,]$ and $B_j[\,]$* */

/* *MEMOIZATION: DO NOT RECOMPUTE $L[n,m]$* */

if ($L[n,m]$ is undefined) **then**

  if ($n = 0$) **or** ($m = 0$) **then** $L[n,m] \leftarrow 0$;

  else if ($A[n] = B[m]$) **then**

    $L[n,m] \leftarrow 1+$ LCS~~Recursive~~(A, $n-1$, B, $m-1$);

    Memoized

  else

    Memoized

    $k \leftarrow$ LCS~~Recursive~~(A, $n-1$, B, m);

    $k' \leftarrow$ LCS~~Recursive~~(A, n, B, $m-1$);

    $L[n,m] \leftarrow \max(k,k')$;

  end

    Memoized

end

return $L[n,m]$;

LCS: Memoized Algorithm

Input : Array $A[\,]$ representing a sequence of length A.
Array $B[\,]$ representing a sequence of length B.

```
for i = 0 to n do
    for j = 0 to m do
        LCS[i, j] ← undefined;
    end
end
LCSMemoized(A, n, B, m, L);
return L[n, m];
```

LCS: Memoized Algorithm

```
procedure LCSMemoized(A[ ], n, B[ ], m, L[ ; ])
/* Arrays A[ ] and B[ ] represent sequences of length n and m   */
/* L[i, j] = length of LCS of A_i[ ] and B_j[ ]                  */
/* MEMOIZATION: DO NOT RECOMPUTE L[n, m]                         */
if (L[n, m] is undefined) then
    if (n = 0) or (m = 0) then L[n, m] ← 0;
    else if (A[n] = B[m]) then
        L[n, m] ← 1+ LCSRecursive(A, n − 1, B, m − 1);
    else
        k ← LCSRecursive(A, n − 1, B, m);
        k′ ← LCSRecursive(A, n, B, m − 1);
        L[n, m] ← max(k, k′);
    end
end
return L[n, m];
```

Running time: $\Theta(n \times m)$

LCS: Memoized Algorithm - Running Time

$L[i,j]$	0	1	2	3	...	$m-1$	m
0	0	0	0	0	...	0	0
1	0						
2	0						
3	0						
$\vdots$	$\vdots$						
$n-1$	0						
n	0						

Memoized Recursion

Bottom Up Iterative.

LCS: Top Down Algorithm

```
procedure LCSMemoized(A[ ], n, B[ ], m, L[ ; ])
/* Arrays A[ ] and B[ ] represent sequences of length n and m   */
/* L[i, j] = length of LCS of A_i[ ] and B_j[ ]                 */
/* MEMOIZATION: DO NOT RECOMPUTE L[n, m]                        */
if (L[n, m] is undefined) then
    if (n = 0) or (m = 0) then L[n, m] ← 0;
    else if (A[n] = B[m]) then
        L[n, m] ← 1+ LCSRecursive(A, n − 1, B, m − 1);
    else
        k ← LCSRecursive(A, n − 1, B, m);
        k′ ← LCSRecursive(A, n, B, m − 1);
        L[n, m] ← max(k, k′);
    end
end
return L[n, m];
```

Algorithm is “Top down” - Start from top and recurse downward.

Longest Common Subsequence: Bottom Up Algorithm

$$L(i,j) = \begin{cases} \max(L(i-1,j), L(i,j-1)) & \text{if } a_i \neq a_j \\ L(i-1,j-1)+1 & \text{if } a_i = a_j. \end{cases}$$

procedure `LongestCommonSubsequence`($A[\,]$, n, $B[\,]$, m)
/* *Arrays A[ ] and B[ ] represent sequences of length n and m* */
for $i = 0$ **to** n **do** $L[i,0] \leftarrow 0$;
for $j = 0$ **to** m **do** $L[0,j] \leftarrow 0$;
for $i = 1$ **to** n **do**
    for $j = 1$ **to** m **do**
        if $(A[i] = B[j])$ **then** $L[i,j] = 1 + L[i-1,j-1]$;
        else $L(i,j) = \max(L[i-1,j], L[i,j-1])$;
    end
end
return $L(n,m)$;

Algorithm is "Bottom up" - Start from bottom and move up.

Rod Cutting: Recursive Algorithm

NOT DYNAMIC PROGRAMMING.
DON'T USE THIS ALGORITHM.

$r(n) = \max_{i=1,2,\ldots,n} \left(p_i + r(n-i) \right)$.

```
procedure CutRodRecursive(p[ ], n)
/* p[ ] is an array of n prices                */
/* p[i] is the price of a rod of length i      */
if (n = 0) then return (0);
q ← −∞;
for i = 1 to n do
    q′ ← p[i]+ CutRodRecursive(p[ ], n − i);
    q ← max (q, q′);
end
return (q);
```

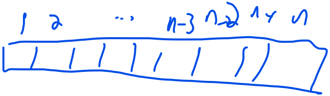
$T(n) = \sum_{i=1,2,\ldots,n-1} T(i)$.

Rod Cutting: Memoized Algorithm

```
procedure CutRodMemoized(p[ ], n, r[ ])
/* p[ ] is an array of n prices                              */
/* p[i] is the price of a rod of length i                    */
/* r[i] is optimal cost of cutting rod of length i           */
if (r[n] is undefined) then
    if (n = 0) then r[n] ← 0 ;
    else
        q ← −∞;
        for i = 1 to n do
            q′ ← p[i] + CutRodMemoized(p[ ], n − i);
            q ← max(q, q′);
        end
        r[n] ← q;
    end
end
return r[n];
```

Rod Cutting: Memoized Algorithm - Running Time

```
procedure CutRodMemoized(p[ ], n, r[ ])
/* p[ ] is an array of n prices                              */
/* p[i] is the price of a rod of length i                    */
if (r[n] is undefined) then
    if (n = 0) then r[n] ← 0 ;
    else
        q ← −∞;
        for i = 1 to n do      ← n
            q′ ← p[i]+ CutRodMemoized(p[ ], n − i);
            q ← max (q, q′);
        end
        r[n] ← q;
    end
end
return r[n];
```



Running time: $\cancel{\Theta(n)}$ $\sum_{i=1}^{n} ci \approx \frac{cn^2}{2} \in \Theta(n^2)$

Rod Cutting: Memoized Algorithm - Running Time (cont)

Dynamic Programming: Overview (Revised)

- Characterize the structure of an optimal solution.
- Recursively define the value of an optimal solution.
- Compute the value of an optimal solution, either:
  - Bottom up, OR
  - Top down (Memoized recursive algorithm)
- Construct an optimal solution from the computed information.