

CO202 – Software Engineering – Algorithms
Dynamic Programming

Ben Glocker
Huxley Building, Room 377
b.glocker@imperial.ac.uk

Algorithm Design

How to design an (efficient) algorithm?

Structure the problem!*

Make use of **algorithmic schemes/design paradigms**

- Incremental Approach
- Divide and Conquer (last week)
- **Dynamic Programming** (this week)
- Greedy Algorithms (next week)

*Take some time and think!

Dynamic Programming Principle

- DP solves problems by **combining** the solutions to **subproblems** (similar to Divide and Conquer)
- In D&C subproblems do not overlap: disjoint partitioning
- In DP subproblems **overlap**: subproblems share subsubproblems
- A DP algorithm solves each subproblem just once and **saves its answer in a table**, avoiding recomputations

The term “Programming” refers to a **tabular** method, not to writing computer code. DP is typically applied to **optimisation problems**.

Optimisation Problems

- Such problems can have many possible solutions
- Each solution has a value (e.g. a *cost* or an *energy*)
- Optimisation seeks for a solution with the optimal (minimum or maximum) value: an^* optimal solution

*Note: there could be several solutions with the same optimal value

Dynamic Programming

When developing a DP algorithm, follow four steps:

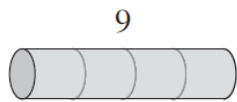
1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution, typically in a bottom-up fashion
4. Construct an optimal solution from computed information

Example: Rod Cutting

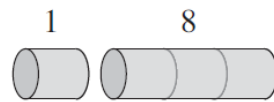
Given a rod of length n metres and a table of prices p_i for $i = 1, 2, \dots, n$, determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces.

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

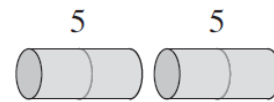
Possible cuts for rod of length $n = 4$



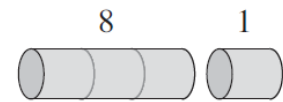
(a)



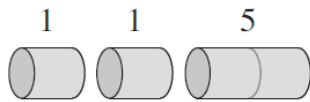
(b)



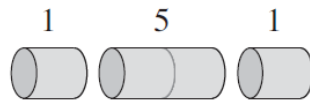
(c)



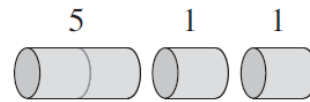
(d)



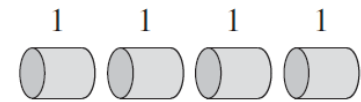
(e)



(f)



(g)



(h)

[Cormen] p.361

Example: Rod Cutting (cont'd)

A rod of length n can be cut up in 2^{n-1} different ways*

*we assume only integer-valued cuts are possible

- An optimal solution cuts the rod into k pieces, for some $1 \leq k \leq n$

- Optimal decomposition is then

$$n = i_1 + i_2 + \cdots + i_k$$

- Providing maximal revenue

$$r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$$

Example: Rod Cutting (cont'd)

Optimal solutions for all $1 \leq i \leq 10$

Max Revenue	Optimal Decomposition
$r_1 = 1$	$1 = 1$ (no cuts)
$r_2 = 5$	$2 = 2$ (no cuts)
$r_3 = 8$	$3 = 3$ (no cuts)
$r_4 = 10$	$4 = 2 + 2$
$r_5 = 13$	$5 = 2 + 3$
$r_6 = 17$	$6 = 6$ (no cuts)
$r_7 = 18$	$7 = 1 + 6$ or $7 = 2 + 2 + 3$
$r_8 = 22$	$8 = 2 + 6$
$r_9 = 25$	$9 = 3 + 6$
$r_{10} = 30$	$10 = 10$ (no cuts)

Recursive Rod Cutting

The maximum revenue r_n can be determined by considering the revenues for shorter rods:

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1)$$

or more compact (with $r_0 = 0$)

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

The rod-cutting problem exhibits **optimal substructure**: optimal solutions to a problem incorporate optimal solutions to related subproblems.

Recursive Rod Cutting (cont'd)

Top-down Implementation

CUT-ROD(p,n)

```
1: if n == 0                # if no rod
2:     return 0              # then no revenue
3: q = -∞                    # initialise revenue
4: for i = 1 to n            # search for maximum
5:     q = max(q, p[i] + CUT-ROD(p,n-i))
6: return q
```

What is the running time of the CUT-ROD algorithm?

$$T(n) = 2^n$$

Bad, bad...really bad!

Recursive Rod Cutting (cont'd)

Exponential running time of CUT-ROD

$$T(n) = \begin{cases} 1, & \text{if } n = 0 \\ 1 + \sum_{j=0}^{n-1} T(j), & \text{if } n > 0 \end{cases}$$

Assumption: $T(n) = 1 + \sum_{j=0}^{n-1} T(j) = 2^n$

Base case: $T(0) = 1 = 2^0$

Holds.

Show for $n+1$: $T(n+1) = 1 + \sum_{j=0}^n T(j)$

$$= 1 + \sum_{j=0}^{n-1} T(j) + T(n)$$

$$= 2T(n)$$

$$= 2(2^n)$$

$$= 2^{n+1}$$

Extract $T(n)$ from the summation

Get twice the same term

Insert assumption

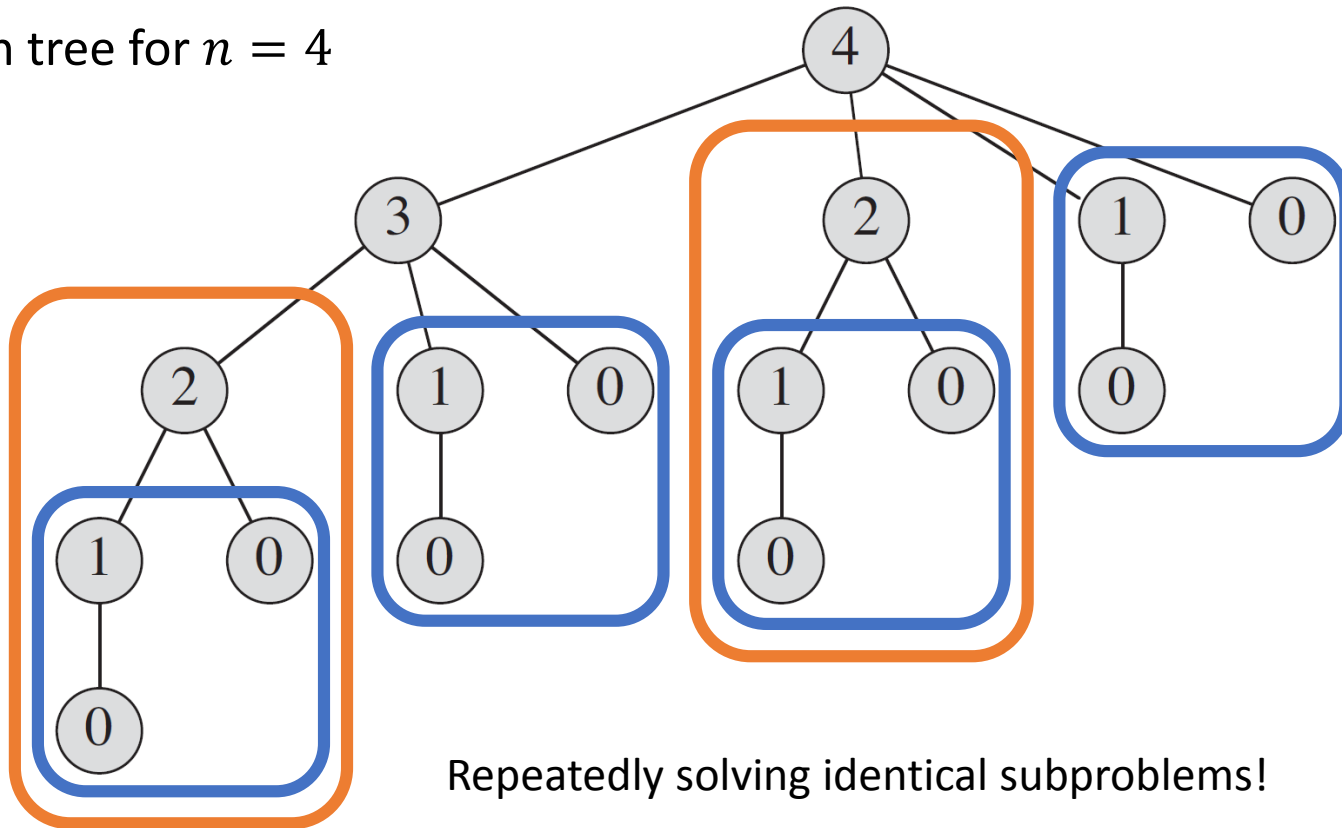
Done.

■

Inefficiency of Recursive Rod Cutting

Why is CUT-ROD so inefficient?

Recursion tree for $n = 4$



[Cormen] p.364

The Dynamic Programming Approach

- Subproblems are solved **once**, and solutions are **saved**
- If we encounter a subproblem again, just **look up** the solution
- Dynamic programming uses additional memory, often yielding a **time-memory trade-off**
- Two strategies for implementing a DP algorithm:
 1. **Top-down with memoization***
 2. **Bottom-up**

*Note: not a typo, memoization comes from *memo*.

Top-Down with Memoization

MEMOIZED-CUT-ROD(p, n)

```
1: let  $r[0..n]$  be a new array
2: for  $i = 0$  to  $n$ 
3:    $r[i] = -\infty$ 
4: return MEMOIZED-CUT-ROD-AUX( $p, n, r$ )
```

MEMOIZED-CUT-ROD-AUX(p, n, r)

```
1: if  $r[n] \geq 0$ 
2:   return  $r[n]$ 
3: if  $n == 0$ 
4:    $q = 0$ 
5: else
6:    $q = -\infty$ 
7:   for  $i = 1$  to  $n$ 
8:      $q = \max(q, p[i] + \text{MEMOIZED-CUT-ROD-AUX}(p, n-i, r))$ 
9:  $r[n] = q$ 
10: return  $q$ 
```

CUT-ROD(p, n)

```
1: if  $n == 0$ 
2:   return 0
3:  $q = -\infty$ 
4: for  $i = 1$  to  $n$ 
5:    $q = \max(q, p[i] + \text{CUT-ROD}(p, n-i))$ 
6: return  $q$ 
```

Bottom-Up Version of Rod Cutting

BOTTOM-UP-CUT-ROD(p, n)

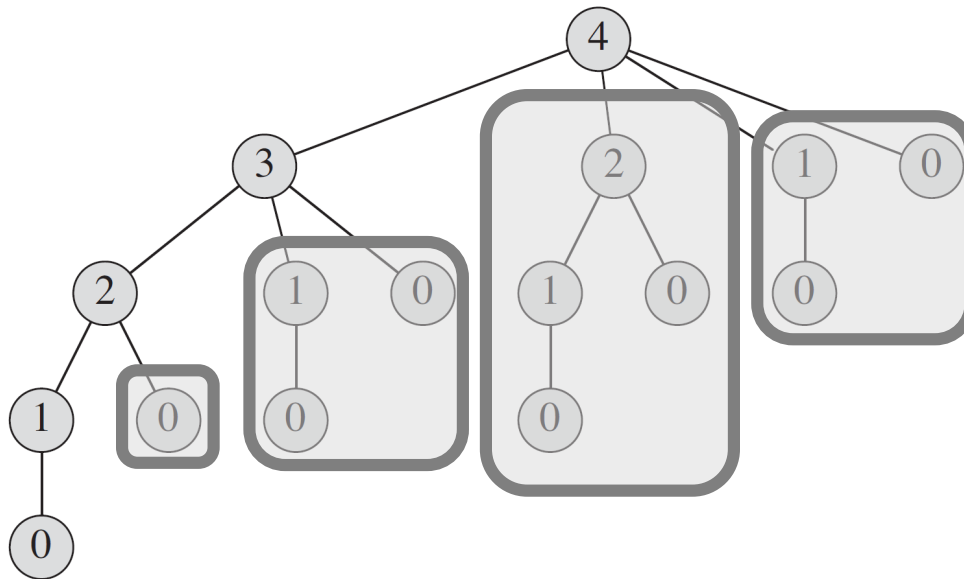
```
1: let  $r[0..n]$  be a new array
2:  $r[0] = 0$ 
3: for  $j = 1$  to  $n$ 
4:      $q = -\infty$ 
5:     for  $i = 1$  to  $j$ 
6:          $q = \max(q, p[i] + r[j-i])$ 
7:      $r[j] = q$ 
8: return  $r[n]$ 
```

What is the running time of BOTTOM-UP-CUT-ROD?

$$T(n) = n^2$$

Memoization

- Memoization allows us to make top-down recursive algorithms as efficient as bottom-up approaches
- A memoized recursive algorithm maintains an entry in a table for the solution to each subproblem



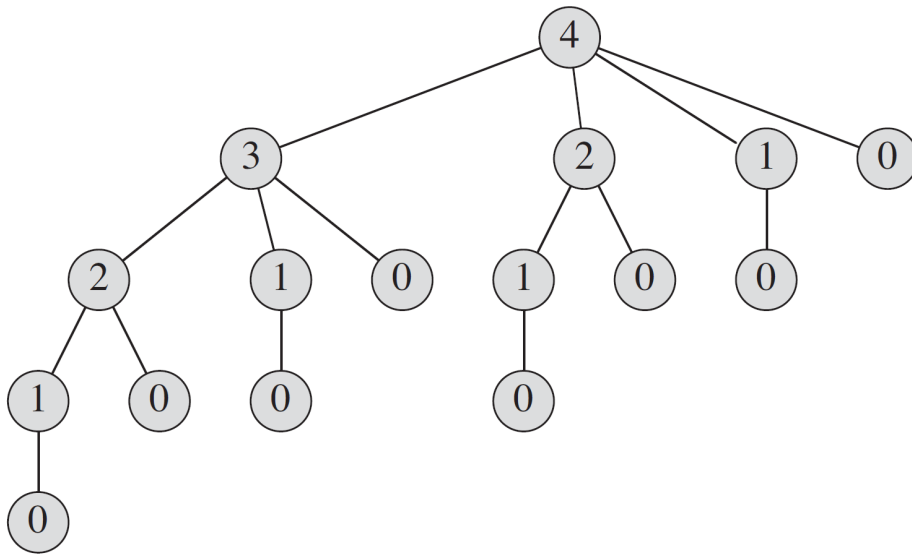
Top-down vs. Bottom-up

Top-down with memoization and bottom-up have the same asymptotic running time

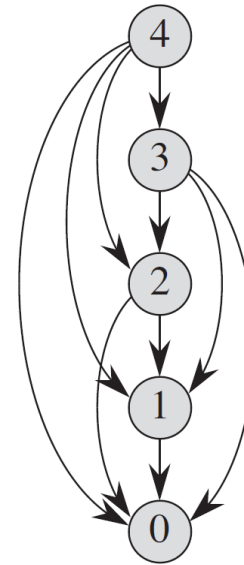
- Bottom-up is usually more efficient by a constant factor because there is no overhead for recursive calls
- Bottom-up might benefit from optimal memory access
- Top-down can avoid to compute solutions of subproblems that are not required
- Top-down implementation is closer to the natural, recursive definition of the problem

Subproblem Graphs

Rod cutting with $n = 4$



Recursion Tree



Subproblem Graph

Reconstructing a Solution

Compute not only the maximum revenue, but also record the solution (i.e. the cut off pieces)

EXTENDED-BOTTOM-UP-CUT-ROD(p, n)

```
1: let  $r[0..n]$  and  $s[1..n]$  be a new array
2:  $r[0] = 0$ 
3: for  $j = 1$  to  $n$ 
4:      $q = -\infty$ 
5:     for  $i = 1$  to  $j$ 
6:         if  $q < p[i] + r[j-i]$ 
7:              $q = p[i] + r[j-i]$ 
8:              $s[j] = i$ 
9:      $r[j] = q$ 
10: return  $r$  and  $s$ 
```

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$		1	2	3	2	2	6	1	2	3	10

Print a Solution

Print the list of piece sizes for an optimal decomposition of a rod of length n

```
PRINT-CUT-ROD-SOLUTION(p,n)
```

```
1: (r,s) = EXTENDED-BOTTOM-UP-CUT-ROD(p,n)
```

```
2: while n > 0
```

```
3:     print s[n]
```

```
4:     n = n - s[n]
```

Key Elements of Dynamic Programming

When should we consider DP?

- **Optimal substructure:** Optimal solution to a problem contains optimal solutions to subproblems
- **Overlapping subproblems:** The space of subproblems must be “small”. Subproblems are solved over and over, rather than generating new subproblems

Longest Common Subsequence (LCS)

Consider the following two strands of DNA

A strand of DNA is a string over the finite set $\{A, C, G, T\}$, standing for bases adenine, cytosine, guanine, and thymine.

$$S_1 = \text{ACCGGTCGAGTGCGCGGAAGCCGGCCGAA}$$
$$S_2 = \text{GTCGTTCGGAATGCCGTTGCTCTGTAAA}$$

How similar are these two strands?

One way of defining similarity is to look for bases that appear in same order, but not necessarily consecutively

$$S_3 = \text{GTCGTCGGAAGCCGGCCGAA}$$

Formal Definition of Subsequence

Given a sequence

$$X = \langle x_1, x_2, \dots, x_m \rangle$$

Another sequence

$$Z = \langle z_1, z_2, \dots, z_k \rangle$$

is a **subsequence** of X if there exists a strictly increasing sequence $\langle i_1, i_2, \dots, i_k \rangle$ of indices of X such that for all $j = 1, 2, \dots, k$ we have $x_{i_j} = z_j$.

Example: $Z = \langle B, C, D, B \rangle$ is a subsequence of $X = \langle A, B, C, B, D, A, B \rangle$ with corresponding index sequence $\langle 2, 3, 5, 7 \rangle$.

Common Subsequence

Given two sequences X and Y , then the sequence Z is a **common subsequence** if it is a subsequence of X and Y .

Example: $X = \langle A, B, C, B, D, A, B \rangle$ and $Y = \langle B, D, C, A, B, A \rangle$ then $Z = \langle B, C, A \rangle$ is a common subsequence. What is the LCS?

The LCS Problem

Given two sequence X and Y , we wish to find a maximum-length common subsequence. We will solve this using DP!

Reminder: Dynamic Programming

When developing a DP algorithm, follow 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
4. Construct an optimal solution

LCS – Step 1: Characterize the Structure

Brute-force: Enumerate all subsequences of X and check each whether it is also a subsequence of Y . **Impractical!**

Definition of prefix: The i th prefix of X , for $i = 0, 1, \dots, m$, is defined as $X_i = \langle x_1, x_2, \dots, x_i \rangle$ with X_0 being the empty sequence

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_7 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_6 = \langle B, D, C, A, B, A \rangle$$

$$Z_4 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_7 = \langle A, B, C, B, D, A, \textcolor{red}{B} \rangle$$

$$Y_6 = \langle B, D, C, A, B, A \rangle$$

$$Z_4 = \langle B, C, A, \textcolor{red}{B} \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_7 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_5 = \langle B, D, C, A, B, A \rangle$$

$$Z_4 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_7 = \langle A, B, C, B, D, A, \textcolor{red}{B} \rangle$$

$$Y_5 = \langle B, D, C, A, \textcolor{red}{B}, \textcolor{gray}{A} \rangle$$

$$Z_4 = \langle B, C, A, \textcolor{red}{B} \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_6 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_4 = \langle B, D, C, A, B, A \rangle$$

$$Z_3 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_6 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_4 = \langle B, D, C, A, B, A \rangle$$

$$Z_3 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_5 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_3 = \langle B, D, C, A, B, A \rangle$$

$$Z_2 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_5 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_3 = \langle B, D, C, A, B, A \rangle$$

$$Z_2 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_4 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_3 = \langle B, D, C, A, B, A \rangle$$

$$Z_2 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_4 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_3 = \langle B, D, C, A, B, A \rangle$$

$$Z_2 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_3 = \langle A, B, C, \textcolor{gray}{B}, \textcolor{gray}{D}, \textcolor{blue}{A}, \textcolor{blue}{B} \rangle$$

$$Y_3 = \langle B, D, C, \textcolor{blue}{A}, \textcolor{blue}{B}, \textcolor{gray}{A} \rangle$$

$$Z_2 = \langle B, C, \textcolor{blue}{A}, \textcolor{blue}{B} \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_3 = \langle A, B, \textcolor{red}{C}, \textcolor{gray}{B}, \textcolor{gray}{D}, \textcolor{blue}{A}, \textcolor{blue}{B} \rangle$$

$$Y_3 = \langle B, D, \textcolor{red}{C}, \textcolor{blue}{A}, \textcolor{blue}{B}, \textcolor{gray}{A} \rangle$$

$$Z_2 = \langle B, \textcolor{red}{C}, \textcolor{blue}{A}, \textcolor{blue}{B} \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_2 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_2 = \langle B, D, C, A, B, A \rangle$$

$$Z_1 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_2 = \langle A, \textcolor{red}{B}, \textcolor{blue}{C}, \textcolor{gray}{B}, \textcolor{gray}{D}, A, \textcolor{blue}{B} \rangle$$

$$Y_2 = \langle B, D, \textcolor{blue}{C}, \textcolor{blue}{A}, \textcolor{blue}{B}, \textcolor{gray}{A} \rangle$$

$$Z_1 = \langle \textcolor{red}{B}, \textcolor{blue}{C}, \textcolor{blue}{A}, \textcolor{blue}{B} \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_2 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_1 = \langle B, D, C, A, B, A \rangle$$

$$Z_1 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_2 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_1 = \langle B, D, C, A, B, A \rangle$$

$$Z_1 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

$$X_1 = \langle A, B, C, B, D, A, B \rangle$$

$$Y_0 = \langle B, D, C, A, B, A \rangle$$

$$Z_0 = \langle B, C, A, B \rangle$$

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

Proof: (Part 1)

- 1) If $z_k \neq x_m$, then we could append $x_m = y_n$ to Z and get an LCS with $k + 1$, which is a contradiction to Z being LCS of X and Y .

The prefix Z_{k-1} with length $k - 1$ is LCS of X_{m-1} and Y_{n-1} . Suppose there is LCS W that is greater than $k - 1$, then appending $x_m = y_n$ to W produces an LCS with length $> k$, which is a contradiction.

LCS – Step 1: Characterize the Structure

Optimal substructure of LCS (Theorem):

Given $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ and let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be any LCS of X and Y

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is 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 LCS of X_{m-1} and Y
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is LCS of X and Y_{n-1}

Proof: (Part 2 & 3)

- 2) If $z_k \neq x_m$, then Z is a common subsequence of X_{m-1} and Y .
If there were a common subsequence W of X_{m-1} and Y with length $> k$, then W would also be a common subsequence of X_m and Y , which is a contradiction with Z being LCS of X and Y .
- 3) The proof is symmetric to (2).



LCS – Step 2: Recursive Solution

There are two cases:

1. If $x_m = y_n$, find LCS of X_{m-1} and Y_{n-1} and append $x_m = y_n$
2. Otherwise, solve two subproblems
 - a) Find LCS of X_{m-1} and Y
 - b) Find LCS of X and Y_{n-1}

Whichever LCS is longer, it is also an LCS of X and Y

Overlapping-subproblems property:

In order to find LCS of X and Y we may need to find LCSs of X and Y_{n-1} and X_{m-1} and Y . Each of these subproblems has the subsubproblem of finding LCS of X_{m-1} and Y_{n-1} .

LCS – Step 2: Recursive Solution (cont'd)

Let $c[i, j]$ be the length of an LCS of sequences X_i and Y_j

$$c[i, j] = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0 \\ c[i - 1, j - 1] + 1, & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(c[i, j - 1], c[i - 1, j]), & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

LCS – Step 3: Computing a Solution (Bottom-up)

LCS-LENGTH(X,Y)

```
1: m = X.length
2: n = Y.length
3: let b[1..m,1..n] and c[0..m,0..n] be new tables
4: for i = 1 to m                                # initialize first column
5:     c[i,0] = 0
6: for j = 0 to n                                # initialize first row
7:     c[0,j] = 0
8: for j = 1 to n
9:     for i = 1 to m
10:        if  $x_i == y_j$                         # case 1
11:            c[i,j] = c[i-1,j-1] + 1
12:            b[i,j] = ↖
13:        elseif c[i-1,j] ≥ c[i,j-1]             # case 2a
14:            c[i,j] = c[i-1,j]
15:            b[i,j] = ↑
16:        else                                    # case 2b
17:            c[i,j] = c[i,j-1]
18:            b[i,j] = ←
19: return c and b
```


Exercise: Compute 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							

Exercise: Compute 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						

```

8: for j = 1 to n
9:   for i = 1 to m
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
17:       $c[i,j] = c[i,j-1]$ 
18:       $b[i,j] = \leftarrow$ 

```

Exercise: Compute 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					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

```

8: for j = 1 to n
9:   for i = 1 to m
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
17:       $c[i,j] = c[i,j-1]$ 
18:       $b[i,j] = \leftarrow$ 

```

Exercise: Compute 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					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

```

8: for j = 1 to n
9:   for i = 1 to m
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
17:       $c[i,j] = c[i,j-1]$ 
18:       $b[i,j] = \leftarrow$ 

```

Exercise: Compute 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					
2	B	0	1					
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

```

8: for j = 1 to n
9:   for i = 1 to m
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
17:       $c[i,j] = c[i,j-1]$ 
18:       $b[i,j] = \leftarrow$ 

```

Exercise: Compute 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					
2	B	0	↖ 1					
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

```

8: for j = 1 to n
9:   for i = 1 to m
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
17:       $c[i,j] = c[i,j-1]$ 
18:       $b[i,j] = \leftarrow$ 

```

Exercise: Compute 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					
2	B	0	↖ 1					
3	C	0	1					
4	B	0						
5	D	0						
6	A	0						
7	B	0						

```

8: for j = 1 to n
9:   for i = 1 to m
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
17:       $c[i,j] = c[i,j-1]$ 
18:       $b[i,j] = \leftarrow$ 

```

Exercise: Compute 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						

```

8: for j = 1 to n
9:   for i = 1 to m
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
17:       $c[i,j] = c[i,j-1]$ 
18:       $b[i,j] = \leftarrow$ 

```


Exercise: Compute 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	↑	↑	↑ ↖	1 ←	1 ←	1 ↖
2	B	0	↖	←	←	1 ↑	2 ↖	2 ←
3	C	0	↑	↑	↖	2 ←	2 ↑	2 ↑
4	B	0	↖	↑	↑	2 ↑	3 ↖	3 ←
5	D	0	↑	↖	↑	2 ↑	3 ↑	3 ↑
6	A	0	↑	↑	↑	↖	3 ↑	4 ↖
7	B	0	↖	↑	↑	3 ↑	4 ↖	4 ↑

Exercise: Compute 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

Exercise: Compute 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

Exercise: Compute 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

Exercise: Compute LCS

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

```

8: for j = 1 to n
9:   for i = 1 to m
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
17:       $c[i,j] = c[i,j-1]$ 
18:       $b[i,j] = \leftarrow$ 

```

Exercise: Compute 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

Exercise: Compute 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

LCS – Step 4: Construct a Solution

```
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 xi
6:  elseif b[i,j] == ↑
7:      PRINT-LCS(b,X,i-1,j)
8:  else
9:      PRINT-LCS(b,X,i,j-1)
```


Approximate String Matching

How similar are SUNNY and SNOWY?

- LCS has length 3: SNY
- **LCS Distance:** How many operations does it need to transform one string into the other? Allowed operations are **DELETE** and **INSERT**

S	U	N	N	-	-	Y
S	-	N	-	O	W	Y

Distance: 4

Levenshtein Distance



Allowed operations are **DELETE**, **INSERT** and **REPLACE**

S	U	N	N	-	Y
S	-	N	O	W	Y

Distance: 3

$$d[i, j] = \begin{cases} \max(i, j), & \text{if } i = 0 \text{ or } j = 0 \\ \min \begin{cases} d[i - 1, j] + 1 \\ d[i, j - 1] + 1 \\ d[i - 1, j - 1] + \mathbf{1}_{(x_i \neq y_j)} \end{cases} & , \quad \text{otherwise} \end{cases}$$

Levenshtein Distance (cont'd)

LEV-DISTANCE(X,Y)

```
1: m = X.length
2: n = Y.length
3: let d[0..m,0..n] be new table
4: for i = 1 to m
5:     d[i,0] = i
6: for j = 0 to n
7:     d[0,j] = j
8: for j = 1 to n
9:     for i = 1 to m
10:         c = xi == yj ? 0 : 1
11:         d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
12: return d[m,n]
```

Exercise: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i										
1	I										
2	M										
3	P										
4	E										
5	R										
6	I										
7	A										
8	L										

Exercise: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i	0	1	2	3	4	5	6	7	8	9
1	I	1									
2	M	2									
3	P	3									
4	E	4									
5	R	5									
6	I	6									
7											
8											
9											

```
8: for j = 1 to n
9:   for i = 1 to m
10:    c =  $x_i == y_j$  ? 0 : 1
11:    d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i	0	1	2	3	4	5	6	7	8	9
1	I	1	R 1								
2	M	2									
3	P	3									
4	E	4									
5	R	5									
6	I	6									
7	C	7									
8	A	8									
9	L	9									

```

8: for j = 1 to n
9:     for i = 1 to m
10:        c = xi == yj ? 0 : 1
11:        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)

```

Exercise: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i	0	1	2	3	4	5	6	7	8	9
1	I	1	R	1							
2	M	2	R	D	2						
3	P	3									
4	E	4									
5	R	5									
6	I	6									
7	C	7									
8	A	8									
9	L	9									

```

8: for j = 1 to n
9:     for i = 1 to m
10:        c = xi == yj ? 0 : 1
11:        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)

```

Exercise: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i	0	1	2	3	4	5	6	7	8	9
1	I	1	R 1								
2	M	2	R D 2								
3	P	3	R D 3								
4	E	4									
5	R	5									
6	I	6									
7	C	7									
8	A	8									
9	L	9									

```

8: for j = 1 to n
9:     for i = 1 to m
10:        c = xi == yj ? 0 : 1
11:        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)

```


Exercise: Compute Levenshtein Distance

[illegible]

```

8: for j = 1 to n
9:     for i = 1 to m
10:        c = xi == yj ? 0 : 1
11:        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)

```

Exercise: Compute Levenshtein Distance

I	M	P	E	R	I	-	A	L
E	M	P	I	R	I	C	A	L
REPLACE	KEEP	KEEP	REPLACE	KEEP	KEEP	INSERT	KEEP	KEEP

	j	0	1	2	3	4	5	6	7	8	9		
i		y_j	E	M	P	I	R	I	C	A	L		
0	x_i	0	I 1	I 2	I 3	I 4	I 5	I 6	I 7	I 8	I 9		
1	I	1	D 1	R 1	I 2	I 3	K 3	I 4	I 5	I 6	I 7	I 8	
2	M	2	D 2	R 2	D 2	K 1	I 2	I 3	I 4	I 5	I 6	I 7	I 8
3	P	3	D 3	R 3	D 3	D 2	K 1	I 2	I 3	I 4	I 5	I 6	I 7
4	E	4	D 4	K 3	D 3	D 2	R 2	I 3	I 4	I 5	I 6	I 7	
5	R	5	D 5	D 4	R 4	D 3	D 2	R 3	I 3	I 4	I 5	I 6	
6	I	6	D 6	D 5	R 5	D 4	K 3	D 3	K 2	I 3	I 4	I 5	
7	A	7	D 7	D 6	R 6	D 5	D 4	R 4	D 3	R 3	K 3	I 4	
8	L	8	D 8	D 7	R 7	D 6	D 5	R 5	D 4	R 4	D 4	K 3	

Conclusions

- DP is a powerful algorithm design technique that can sometimes reduce exponential running time to polynomial (e.g. rod cutting, LCS, ...)
- DP is employed in many applications
 - Bioinformatics (computational genomics, DNA matching)
 - Spell checker (via approximate string matching)
 - Time series analysis (speech recognition)
 - Economics

References

Books

- **[Cormen] Introduction to Algorithms**
T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. MIT Press. 2009 (3rd Edition)
- **[Sedgewick] Algorithms**
R. Sedgewick, K. Wayne. Addison-Wesley. 2011 (4th Edition)
- **[Dasgupta] Algorithms**
S. Dasgupta, C. Papadimitriou, U. Vazirani. McGraw-Hill Higher Education. 2006

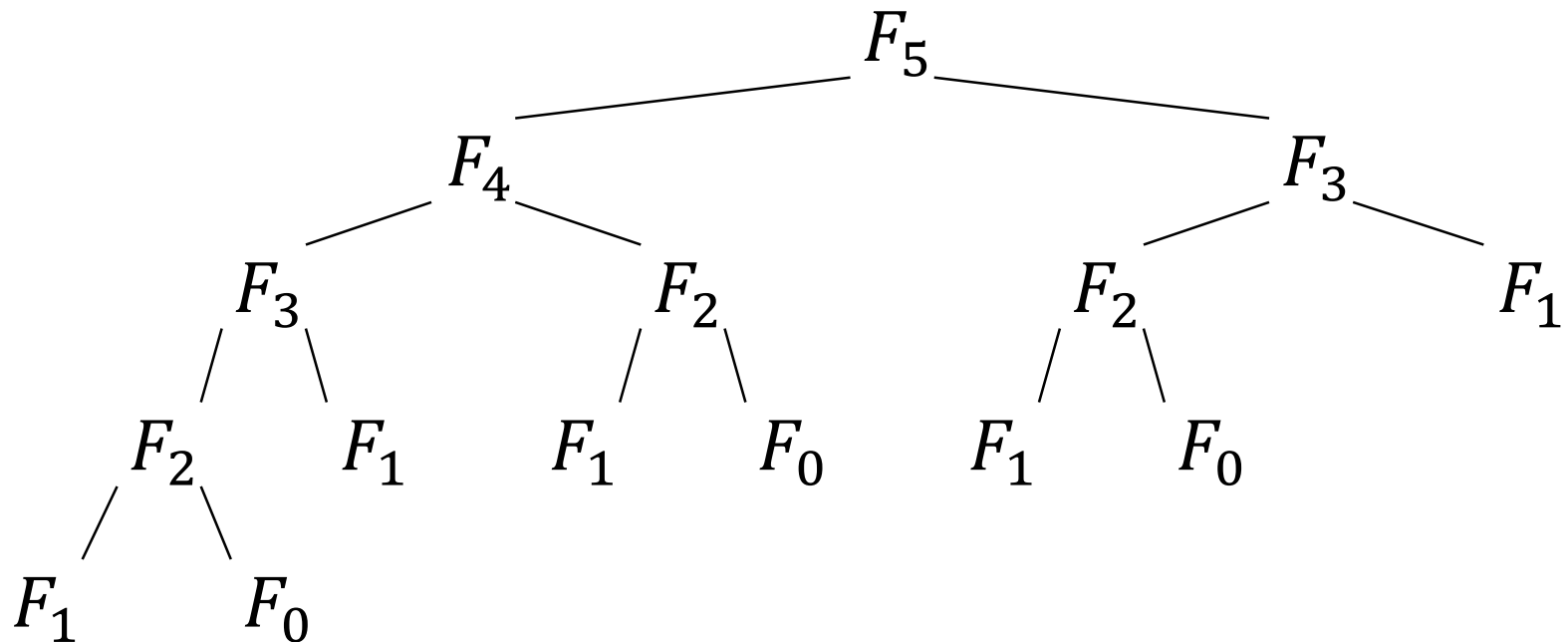
Online

- <http://algs4.cs.princeton.edu/lectures/>
- <https://www.coursera.org/courses?query=algorithms>

Exercise: Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{otherwise} \end{cases}$$



Exercise: Fibonacci Sequence (cont'd)

NAÏVE-FIBONACCI(n)

```
1: if n == 0
2:     return 0
3: if n == 1
4:     return 1
5: return FIBONACCI(n-1) + FIBONACCI(n-2)
```

Running time of NAÏVE-FIBONACCI:

$$T(n) = O(2^{0.694n})$$

Exercise: Bottom-Up Fibonacci

BOTTOM-UP-FIBONACCI(*n*)

1: **if** *n* == 0

2: **return** 0

3: let *f*[0..*n*] be a new array

4: *f*[0] = 0

5: *f*[1] = 1

6: ?

7: ?

8: ?

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{otherwise} \end{cases}$$

What is the running time of BOTTOM-UP-FIBONACCI?

Exercise: Bottom-Up Fibonacci

BOTTOM-UP-FIBONACCI(*n*)

1: **if** *n* == 0

2: **return** 0

3: let *f*[0..*n*] be a new array

4: *f*[0] = 0

5: *f*[1] = 1

6: **for** *i* = 2 **to** *n*

7: *f*[*i*] = *f*[*i*-1] + *f*[*i*-2]

8: **return** *f*[*n*]

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{otherwise} \end{cases}$$

What is the running time of BOTTOM-UP-FIBONACCI?

$T(n) = O(n)$...or?