CE100 Algorithms and Programming II

Week-6 (Matrix Chain Order / LCS)

Spring Semester, 2021-2022

Download DOC, SLIDE, PPTX

<iframe width=700, height=500 frameBorder=0 src="../ce100-week-6-lcs.md_slide.html"></iframe>

Matrix Chain Order / Longest Common Subsequence

Outline

- Elements of Dynamic Programming
  - Optimal Substructure
  - Overlapping Subproblems

- Recursive Matrix Chain Order Memoization
    - Top-Down Approach
    - RMC
    - MemoizedMatrixChain
        - LookupC
    - Dynamic Programming vs Memoization Summary

- Dynamic Programming
  - Problem-2 : Longest Common Subsequence
    - Definitions
    - LCS Problem
    - Notations
    - Optimal Substructure of LCS
      - Proof Case-1
      - Proof Case-2
      - Proof Case-3

- A recursive solution to subproblems (inefficient)
- Computing the length of and LCS
  - LCS Data Structure for DP
  - Bottom-Up Computation
- Constructing and LCS
  - PRINT-LCS
  - Back-pointer space optimization for LCS length

- Most Common Dynamic Programming Interview Questions

Elements of Dynamic Programming

- When should we look for a DP solution to an optimization problem?
- Two key ingredients for the problem
  - Optimal substructure
  - Overlapping subproblems

DP Hallmark #1

- **Optimal Substructure**
  - A problem exhibits optimal substructure
    - if an optimal solution to a problem contains within it optimal solutions to subproblems
  - **Example:** *matrix-chain-multiplication*
    - Optimal parenthesization of $A_1 A_2 \ldots A_n$ that splits the product between A_k and A_{k+1}, contains within it **optimal soln's** to the problems of parenthesizing $A_1 A_2 \ldots A_k$ and $A_{k+1} A_{k+2} \ldots A_n$

Optimal Substructure

- Finding a suitable space of subproblems
  - Iterate on subproblem instances
  - **Example:** *matrix-chain-multiplication*
    - Iterate and look at the structure of optimal soln's to subproblems, sub-subproblems, and so forth
    - Discover that all subproblems consists of subchains of $\langle A_1, A_2, \dots, A_n \rangle$
    - Thus, the set of chains of the form $\langle A_i, A_{i+1}, \dots, A_j \rangle$ for $1 \leq i \leq j \leq n$
    - Makes a natural and reasonable space of subproblems

DP Hallmark #2

- **Overlapping Subproblems**
  - Total number of distinct subproblems should be **polynomial** in the input size
  - When a **recursive** algorithm revisits the same problem **over and over again**,
    - We say that the optimization problem has **overlapping subproblems**

Overlapping Subproblems

- **DP** algorithms typically take advantage of overlapping subproblems
  - by solving each problem once
  - then storing the solutions in a table
    - where it can be looked up when needed
  - using constant time per lookup

Overlapping Subproblems

- Recursive matrix-chain order

$$\text{RMC}(p, i, j)\{$$

$$\text{if } i = j \text{ then}$$

$$\text{return } 0$$

$$m[i, j] \leftarrow \infty$$

$$\text{for } k \leftarrow i \text{to } j - 1 \text{ do}$$

$$q \leftarrow \text{RMC}(p, i, k) + \text{RMC}(p, k + 1, j) + p_{i-1}p_kp_j$$

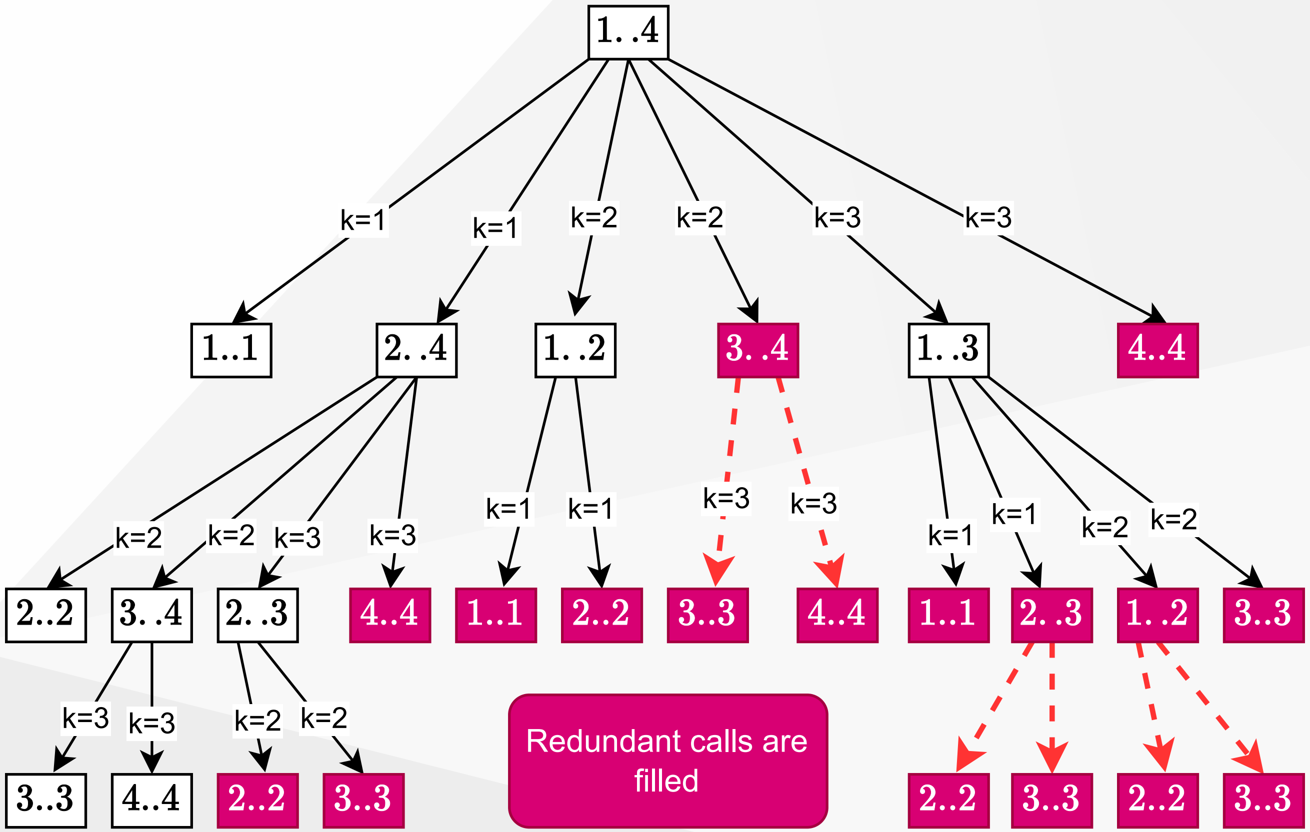
$$if\ q < m[i, j] \text{ then}$$

$$m[i, j] \leftarrow q$$

$$\text{return } m[i, j]\ \}$$

Direct Recursion: Inefficient!

- Recursion tree for $RMC(p, 1, 4)$
- Nodes are labeled with i and j values



Running Time of RMC

$T(1) \geq 1$

$T(n) \geq 1 + \sum_{k=1}^{n-1}(T(k) + T(n-k) + 1) \text{ for } n > 1$

- For $i = 1, 2, \ldots, n$ each term $T(i)$ appears twice
  - Once as $T(k)$, and once as $T(n-k)$
- Collect $n-1$, 1's in the summation together with the front 1

$$T(n) \geq 2\sum_{i=1}^{n-1} T(i) + n$$

- Prove that $T(n) = \Omega(2n)$ using the **substitution method**

Running Time of RMC: Prove that $T(n) = \Omega(2n)$

- Try to show that $T(n) \geq 2^{n-1}$ (**by substitution**)
- **Base case:** $T(1) \geq 1 = 2^0 = 2^{1-1}$ for $n = 1$
- **Ind. Hyp.:**

$$T(i) \geq 2^{i-1} \text{ for all } i = 1, 2, \ldots, n-1 \text{ and } n \geq 2$$

$$T(n) \geq 2\sum_{i=1}^{n-1} 2^{i-1} + n$$

$$= 2\sum_{i=1}^{n-1} 2^{i-1} + n$$

$$= 2(2^{n-1} - 1) + n$$

$$= 2^{n-1} + (2^{n-1} - 2 + n)$$

$$\Rightarrow T(n) \geq 2^{n-1} \quad \text{Q.E.D.}$$

Running Time of RMC: $T(n) \geq 2^{n-1}$

- **Whenever**
  - a recursion tree for the natural recursive solution to a problem contains the same subproblem repeatedly
  - the total number of different subproblems is small
    - it is a good idea to see if $DP(Dynamic\ Programming)$ can be applied

Memoization

- Offers the efficiency of the usual DP approach while maintaining **top-down** strategy
- Idea is to **memoize** the natural, but inefficient, **recursive algorithm**

Memoized Recursive Algorithm

- Maintains an **entry** in a **table** for the soln to each subproblem
- Each table entry contains **a special value** to indicate that the entry has yet to be filled in
- When the subproblem is **first encountered** its solution is **computed** and then **stored** in the table
- Each **subsequent** time that the subproblem encountered the value stored in the table is simply **looked up** and **returned**

Memoized Recursive Matrix-chain Order

- Shaded subtrees are looked-up rather than recomputing

MemoizedMatrixChain(p)
- $n \leftarrow length[p] - 1$
- for $i \leftarrow 1$ to n do
  - for $j \leftarrow 1$ to n do
    - $m[i,j] \leftarrow \infty$
- return LookupC$(p, 1, n) \implies$

$\implies$ LookupC(p, i, j)
- if $m[i,j] = \infty$ then
  - if $i = j$ then
    - $m[i,j] \leftarrow 0$
  - else
    - for $k \leftarrow i$ to $j-1$ do
      - $q \leftarrow$ LookupC(p, i, k) + LookupC$(p, k+1, j) + p_{i-1}p_kp_j$
      - if $q < m[i,j]$ then
        - $m[i,j] \leftarrow q$
- return $m[i,j]$

Memoized Recursive Algorithm

- The approach assumes that
  - The set of **all possible subproblem parameters** are known
  - The relation between the **table positions** and **subproblems** is established
- Another approach is to memoize
  - by using **hashing** with subproblem parameters as key

Dynamic Programming vs Memoization Summary (1)

- Matrix-chain multiplication can be solved in $O(n^3)$ time
  - by either a top-down memoized recursive algorithm
  - or a bottom-up dynamic programming algorithm
- Both methods exploit the **overlapping subproblems** property
  - There are only $\Theta(n^2)$ different subproblems in total
  - Both methods **compute** the soln to **each problem once**
- **Without memoization** the natural **recursive** algorithm runs in **exponential time** since subproblems are solved repeatedly

Dynamic Programming vs Memoization Summary (2)

- **In general practice**
  - If all subproblems must be solved at once
    - a bottom-up **DP algorithm always outperforms** a top-down memoized algorithm by a constant factor
  - because, bottom-up **DP** algorithm
    - Has no overhead for recursion
    - Less overhead for maintaining the table
  - **DP: Regular** pattern of **table accesses** can be exploited to reduce the time and/or space requirements even further
  - **Memoized:** If some problems need not be solved at all, it has the advantage of avoiding solutions to those subproblems

Problem 3: Longest Common Subsequence

Definitions

- A **subsequence** of a given sequence is just the **given sequence** with **some elements** (possibly none) **left out**
- **Example:**
  - $X = \langle A, B, C, B, D, A, B \rangle$
  - $Z = \langle B, C, D, B \rangle$
    - Z is a subsequence of X

Problem 3: Longest Common Subsequence

Definitions

- **Formal definition:** Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$, sequence $Z = \langle z_1, z_2, \ldots, z_k \rangle$ is a subsequence of X
  - if $\exists$ a **strictly increasing sequence** $\langle i_1, i_2, \ldots, i_k \rangle$ of indices of X such that $x_{i_j} = z_j$ for all $j = 1, 2, \ldots, k$, where $1 \leq k \leq m$
- **Example:** $Z = \langle B, C, D, B \rangle$ is a subsequence of $X = \langle A, B, C, B, D, A, B \rangle$ with the **index sequence** $\langle i_1, i_2, i_3, i_4 \rangle = \langle 2, 3, 5, 7 \rangle$

Problem 3: Longest Common Subsequence

Definitions

- If Z is a subsequence of both X and Y, we denote Z as a **common subsequence** of X and Y.
- **Example:**

$$X = \langle A, B^*, C^*, B, D, A^*, B \rangle$$
$$Y = \langle B^*, D, C^*, A^*, B, A \rangle$$

- $Z_1 = \langle B^*, C^*, A^* \rangle$ is a common subsequence (**of length 3**) of X and Y.
- **Two longest common subsequence (LCSs)** of X and Y?
  - $Z2 = \langle B, C, B, A \rangle$ of length 4
  - $Z3 = \langle B, D, A, B \rangle$ of length 4
    - *The optimal solution value = 4*

Longest Common Subsequence (LCS) Problem

- **LCS problem:** Given two sequences
  - $X = \langle x_1, x_2, \ldots, x_m \rangle$ and
  - $Y = \langle y_1, y_2, \ldots, y_n \rangle$, find the **LCS** of $X \& Y$
- **Brute force approach:**
  - Enumerate all subsequences of X
  - Check if each subsequence is also a subsequence of Y
  - Keep track of the **LCS**
  - What is the complexity?
  - There are 2^m subsequences of X
    - **Exponential runtime**

Notation

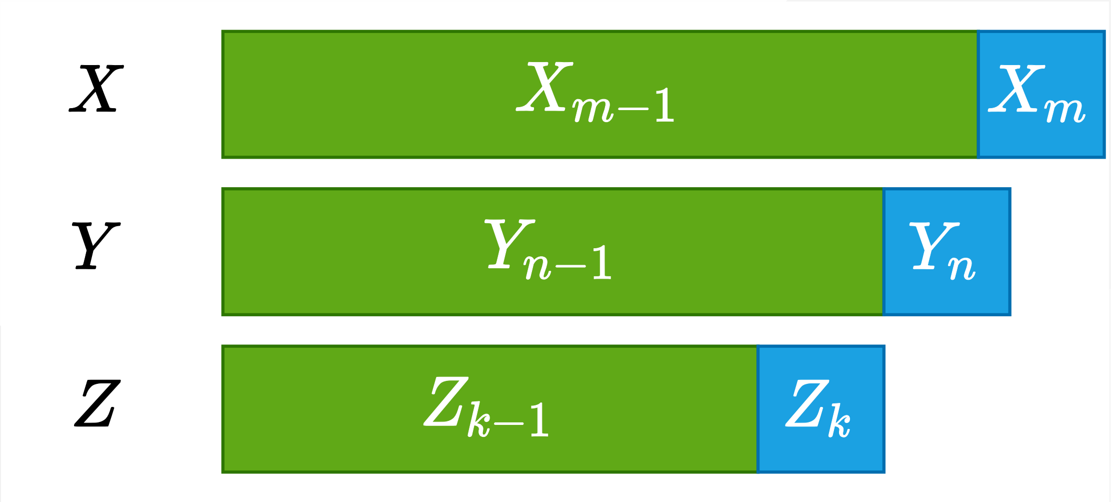
- **Notation:** Let X_i denote the i^{th} prefix of X
  - i.e. $X_i = \langle x_1, x_2, \ldots, x_i \rangle$
- **Example:**

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

$$X_4 = \langle A, B, C, B \rangle$$
$$X_0 = \langle \rangle$$

Optimal Substructure of an LCS

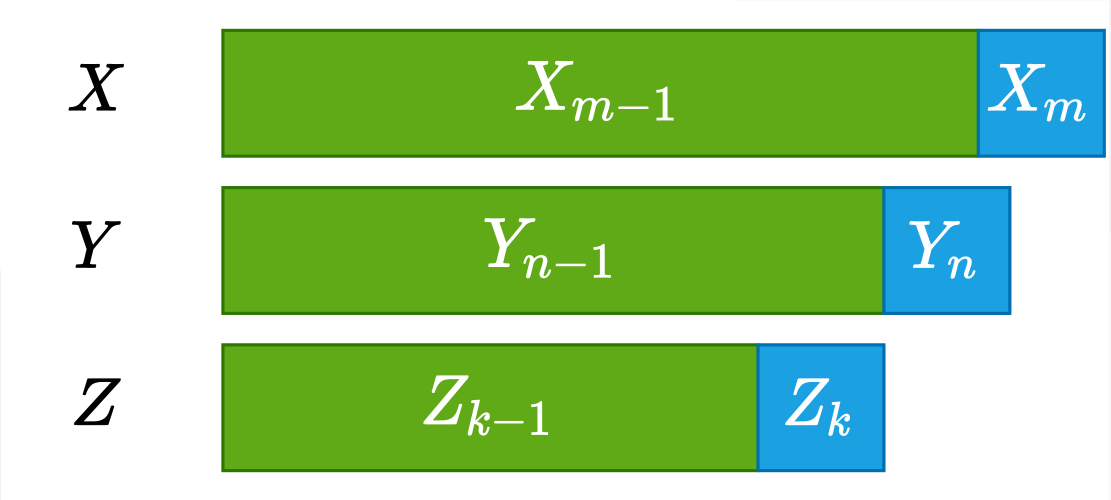
- Let $X =< x1, x2, \dots, xm >$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$ are given
- Let $Z = \langle z_1, z_2, \dots, z_k \rangle$ be an **LCS** of X and Y



- **Question 1:** If $x_m = y_n$, how to define the optimal substructure?
  - We must have $z_k = x_m = y_n$ and
  - $Z_{k-1} = \text{LCS}(X_{m-1}, Y_{n-1})$

Optimal Substructure of an LCS

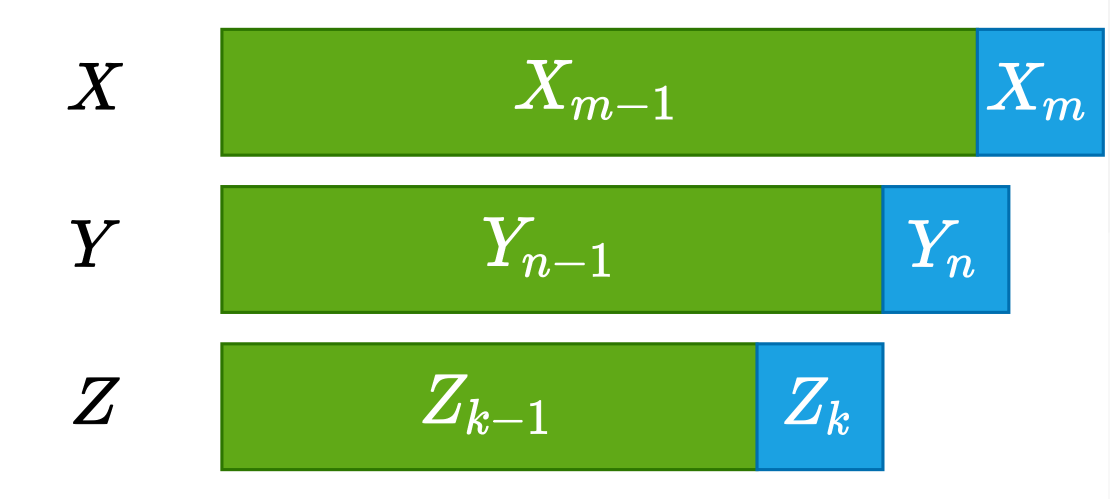
- Let $X =< x1, x2, \ldots, xm >$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ are given
- Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be an **LCS** of X and Y



- **Question 2:** If $x_m \neq y_n$ and $z_k \neq x_m$, how to define the optimal substructure?
  - We must have $Z = \text{LCS}(X_{m-1}, Y)$

Optimal Substructure of an LCS

- Let $X =< x1, x2, \ldots, xm >$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ are given
- Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be an **LCS** of X and Y



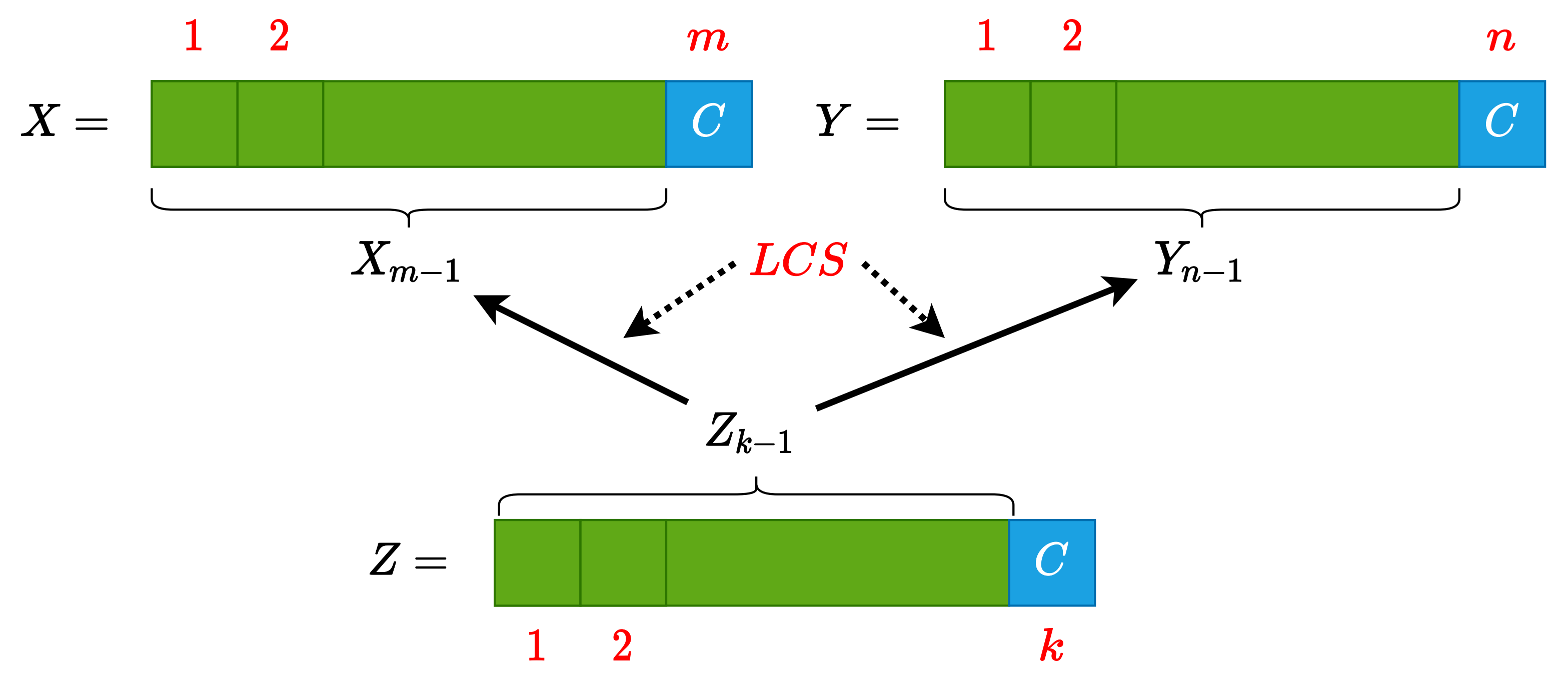
- **Question 3:** If $x_m \neq y_n$ and $z_k \neq y_n$, how to define the optimal substructure?
  - We must have $Z = \text{LCS}(X, Y_{n-1})$

Theorem: Optimal Substructure of an LCS

- Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and Y = <y1, y2, ..., yn> are given
- Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be an **LCS** of X and Y
- **Theorem:** Optimal substructure of an LCS:
  - 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}
  - If $x_m \neq y_n$ and $z_k \neq x_m$
    - then Z is an **LCS** of X_{m-1} and Y
  - If $x_m \neq y_n$ and $z_k \neq y_n$
    - then Z is an **LCS** of X and Y_{n-1}

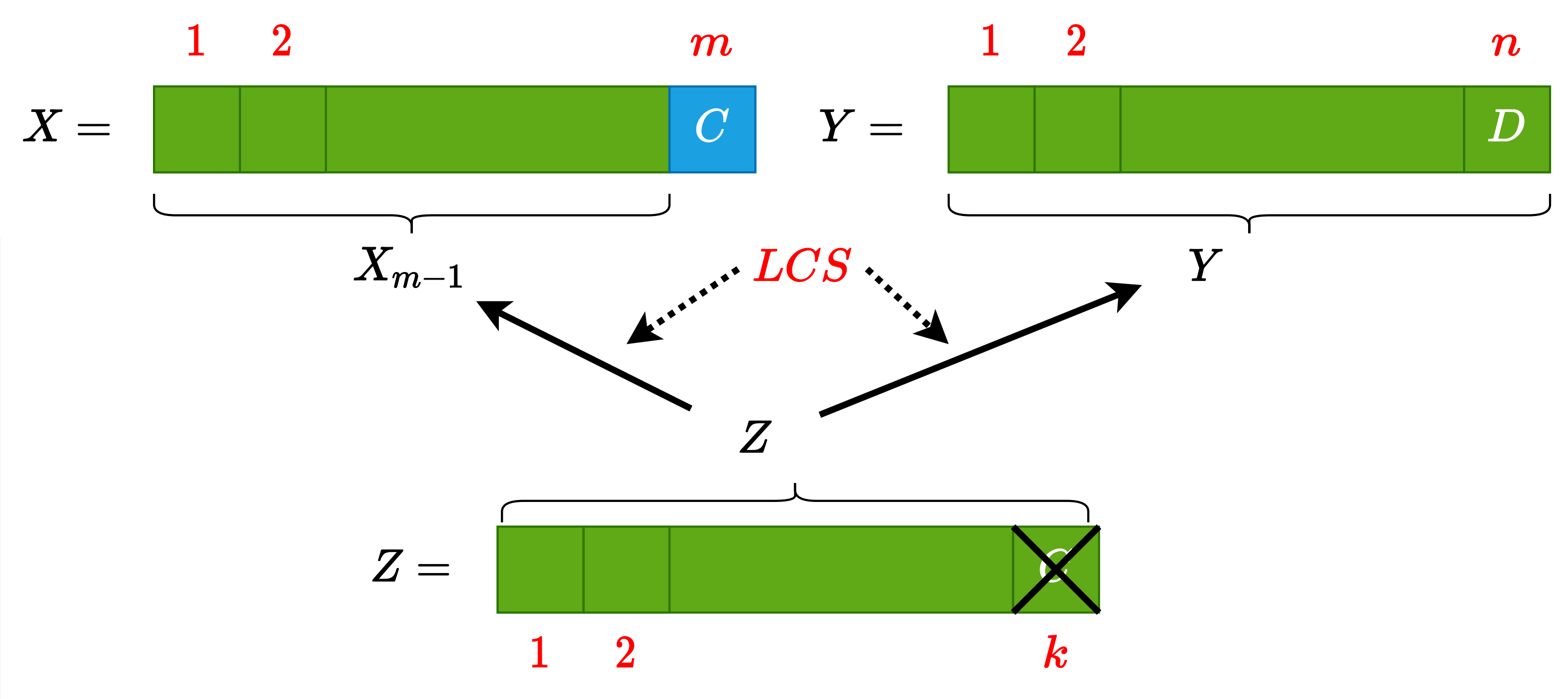
Optimal Substructure Theorem (case 1)

- 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}



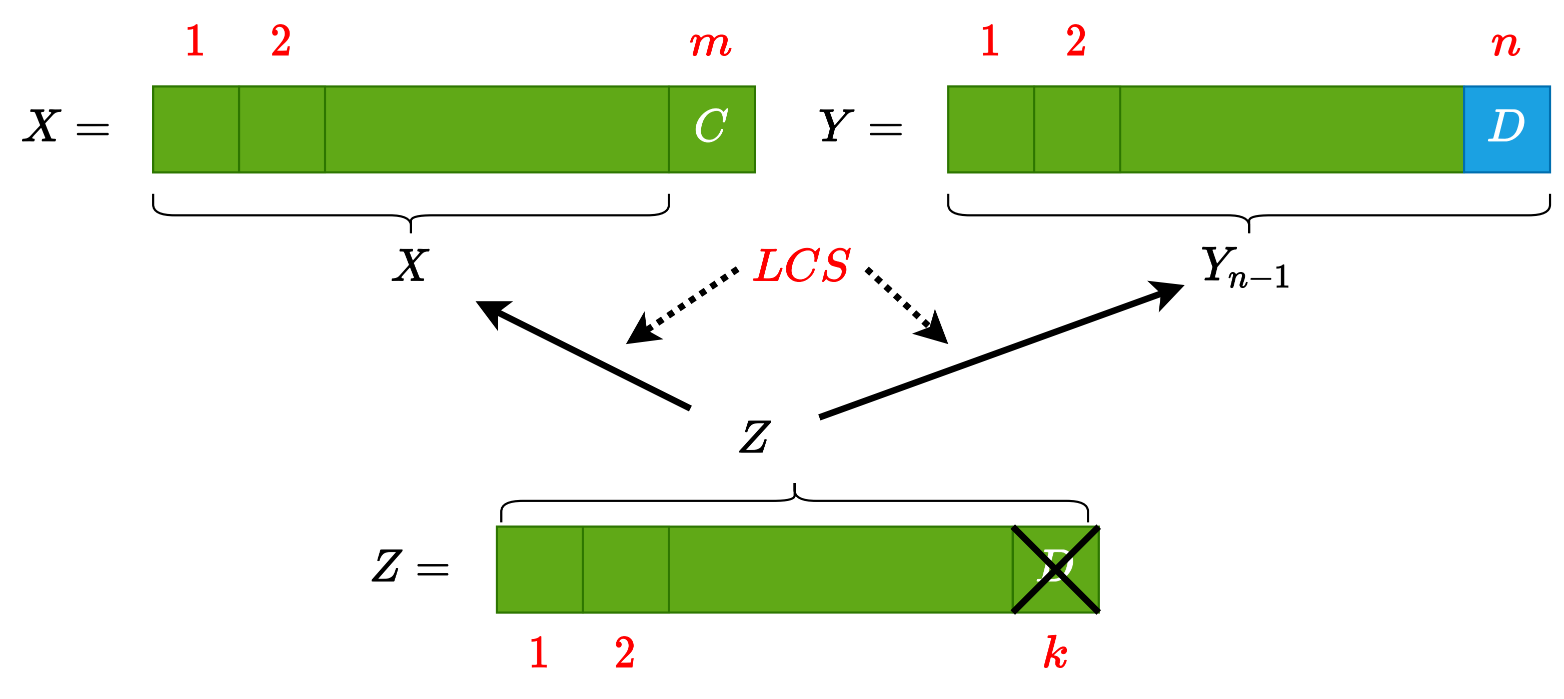
Optimal Substructure Theorem (case 2)

- If $x_m \neq y_n$ and $z_k \neq x_m$ then Z is an **LCS** of X_{m-1} and Y



Optimal Substructure Theorem (case 3)

- If $x_m \neq y_n$ and $z_k \neq y_n$ then Z is an **LCS** of X and Y_{n-1}



Proof of Optimal Substructure Theorem (case 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}
- **Proof:** If $z_k \neq x_m = y_n$ then
  - we can append $x_m = y_n$ to Z to obtain a common subsequence of length $k+1 \implies$ **contradiction**
  - Thus, we must have $z_k = x_m = y_n$
  - Hence, the prefix Z_{k-1} is a **length-**($k-1$) **CS** of X_{m-1} and Y_{n-1}
- **We have to show that** Z_{k-1} is in fact an **LCS** of X_{m-1} and Y_{n-1}
- **Proof by contradiction:**
  - **Assume that** $\exists$ a CS W of X_{m-1} and Y_{n-1} with $|W| = k$
  - Then appending $x_m = y_n$ to W produces a **CS** of length $k+1$

Proof of Optimal Substructure Theorem (case 2)

- If $x_m \neq y_n$ and $z_k \neq x_m$ then Z is an **LCS** of X_{m-1} and Y
- **Proof :** If $z_k \neq x_m$ then Z is a CS of X_{m-1} and Y_n
  - **We have to show that** Z is in fact an **LCS** of X_{m-1} and Y_n
- **(Proof by contradiction)**
  - Assume that $\exists$ a CS W of X_{m-1} and Y_n with $|W| > k$
  - Then W would also be a CS of X and Y
  - Contradiction to the assumption that
    - Z is an LCS of X and Y with $|Z| = k$
- **Case 3:** Dual of the proof for (case 2)

A Recursive Solution to Subproblems

- Theorem implies that there are one or two subproblems to examine
- **if** $x_m = y_n$ **then**
  - we must solve the subproblem of finding an **LCS** of $X_{m-1} \& Y_{n-1}$
  - appending $x_m = y_n$ to this **LCS** yields an **LCS** of $X \& Y$
- **else**
  - we must solve **two subproblems**
    - finding an **LCS** of $X_{m-1} \& Y$
    - finding an **LCS** of $X \& Y_{n-1}$
  - longer of these two **LCS**s is an **LCS** of $X \& Y$
- **endif**

Recursive Algorithm (Inefficient)

$$
\begin{aligned}
&\mathrm{LCS}(X, Y)\ \{\\
&\quad m \leftarrow length[X]\\
&\quad n \leftarrow length[Y]\\
&\quad \text{if } x_m = y_n \text{ then}\\
&\qquad Z \leftarrow \mathrm{LCS}(X_{m-1}, Y_{n-1}) \triangleright \text{solve one subproblem}\\
&\qquad \text{return } \langle Z, x_m = y_n \rangle \triangleright \text{append } x_m = y_n \text{ to } Z\\
&\quad else\\
&\qquad Z^{'} \leftarrow \mathrm{LCS}(X_{m-1}, Y) \triangleright \text{solve two subproblems}\\
&\qquad Z^{''} \leftarrow \mathrm{LCS}(X, Y_{n-1})\\
&\qquad \text{return longer of } Z^{'} \text{ and } Z^{''}\\
&\}
\end{aligned}
$$

A Recursive Solution

- $c[i,j]$: length of an **LCS** of X_i and Y_j

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

Computing the Length of an LCS

- We can easily write an **exponential-time recursive algorithm** based on the given recurrence. $\Longrightarrow$ **Inefficient!**
- How many distinct subproblems to solve?
    - $\Theta(mn)$
- **Overlapping subproblems property:** Many subproblems share the same sub-subproblems.
    - **e.g.** Finding an **LCS** to $X_{m-1}\&Y$ and an **LCS** to $X\&Y_{n-1}$
    - has the sub-subproblem of finding an **LCS** to $X_{m-1}\&Y_{n-1}$
- Therefore, we can use **dynamic programming**.

Data Structures

- Let:
  - $c[i, j]$: length of an **LCS** of X_i and Y_j
  - $b[i, j]$: direction towards the table entry corresponding to the optimal subproblem solution chosen when computing $c[i, j]$.
  - Used to simplify the construction of an optimal solution at the end.
- Maintain the following tables:
  - $c[0 \ldots m, 0 \ldots n]$
  - $b[1 \ldots m, 1 \ldots n]$

Bottom-up Computation

- **Reminder:**

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

- How to choose the order in which we process $c[i,j]$ values?
- The values for $c[i-1,j-1]$, $c[i,j-1]$, and $c[i-1,j]$ must be computed before computing $c[i,j]$.

Bottom-up Computation

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

Need to process:

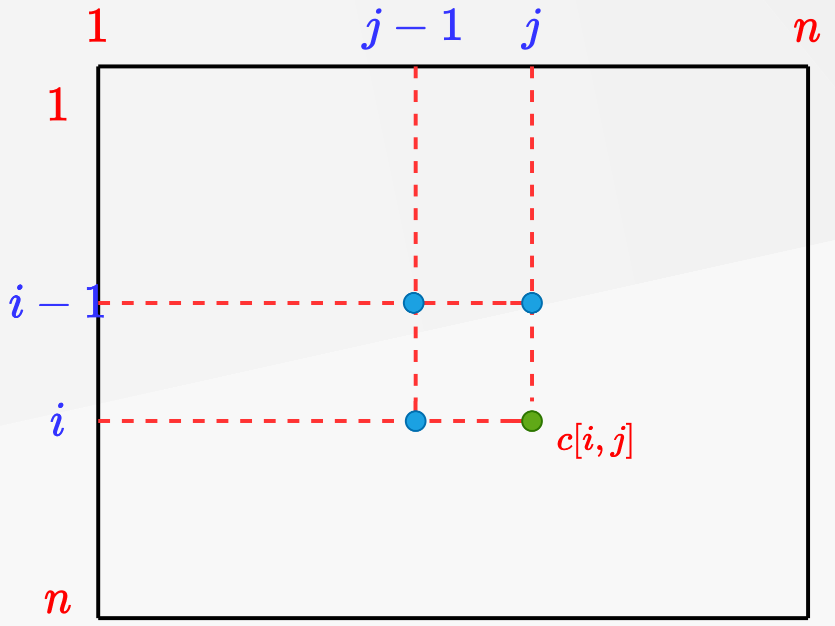
$c[i,j]$

after computing:

$c[i-1,j-1]$,

$c[i,j-1]$,

$c[i-1,j]$



Bottom-up Computation

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

$$\Downarrow$$

for $i \leftarrow 1$ to m

  for $j \leftarrow 1$ to n

    $\dots$

    $\dots$

    $c[i,j] = \cdots$

Computing the Length of an LCS

$$
\left.
\begin{array}{l}
LCS-LENGTH(X,Y) \\
\quad m \leftarrow length[X]; n \leftarrow length[Y] \\
\quad \text{for } i \leftarrow 0 \text{ to } m \text{ do } c[i,0] \leftarrow 0 \\
\quad \text{for } j \leftarrow 0 \text{ to } n \text{ do } c[0,j] \leftarrow 0 \\
\quad \text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
\quad\quad \text{for } j \leftarrow 1 \text{ to } n \text{ do} \\
\quad\quad\quad \text{if } x_i = y_j \text{ then} \\
\quad\quad\quad\quad c[i,j] \leftarrow c[i-1,j-1]+1 \\
\quad\quad\quad\quad b[i,j] \leftarrow \text{“} \nwarrow \text{”} \\
\quad\quad\quad \text{else if } c[i-1,j] \geq c[i,j-1] \\
\quad\quad\quad\quad c[i,j] \leftarrow c[i-1,j] \\
\quad\quad\quad\quad b[i,j] \leftarrow \text{“} \uparrow \\
\quad\quad\quad \text{else} \\
\quad\quad\quad\quad c[i,j] \leftarrow c[i,j-1] \\
\quad\quad\quad\quad b[i,j] \leftarrow \text{“} \leftarrow \text{”}
\end{array}
\right\}
\frac{\text{Total Runtime} = \Theta(mn)}{\text{Total Space} = \Theta(mn)}
$$

Computing the Length of an LCS-1

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 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-2

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0						
3 C	0						
4 B	0						
5 D	0						
6 A	0						
7 B	0						

Computing the Length of an LCS-3

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0						
4 B	0						
5 D	0						
6 A	0						
7 B	0						

Computing the Length of an LCS-4

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
$1\ A$	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
$2\ B$	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
$3\ C$	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
$4\ B$	0						
$5\ D$	0						
$6\ A$	0						
$7\ B$	0						

Computing the Length of an LCS-5

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
$1\ A$	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
$2\ B$	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
$3\ C$	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
$4\ B$	0	$\nwarrow$ 1					
$5\ D$	0						
$6\ A$	0						
$7\ B$	0						

Computing the Length of an LCS-6

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	$1\ B$	$2\ D$	$3\ C$	$4\ A$	$5\ B$	$6\ A$
$0 x_i$	0	0	0	0	0	0	0
$1\ A$	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
$2\ B$	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
$3\ C$	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
$4\ B$	0	$\nwarrow$ 1	$\uparrow$ 1				
$5\ D$	0						
$6\ A$	0						
$7\ B$	0						

Computing the Length of an LCS-7

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4 B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2			
5 D	0						
6 A	0						
7 B	0						

Computing the Length of an LCS-8

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4 B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2		
5 D	0						
6 A	0						
7 B	0						

Computing the Length of an LCS-9

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4 B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	
5 D	0						
6 A	0						
7 B	0						

Computing the Length of an LCS-10

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4 B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5 D	0						
6 A	0						
7 B	0						

Computing the Length of an LCS-11

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4 B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5 D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6 A	0						
7 B	0						

Computing the Length of an LCS-12

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
$1\ A$	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
$2\ B$	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
$3\ C$	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
$4\ B$	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
$5\ D$	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
$6\ A$	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
$7\ B$	0						

Computing the Length of an LCS-13

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

- Running-time = $O(mn)$ since each table entry takes $O(1)$ time to compute

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\nwarrow 1$
2 B	0	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$
3 C	0	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\uparrow 2$	$\uparrow 2$
4 B	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\leftarrow 3$
5 D	0	$\uparrow 1$	$\nwarrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 3$
6 A	0	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\uparrow 3$	$\nwarrow 4$
7 B	0	$\nwarrow 1$	$\uparrow 2$	$\uparrow 3$	$\uparrow 3$	$\nwarrow 4$	$\uparrow 4$

Computing the Length of an LCS-14

Operation of LCS-LENGTH on the sequences

$$X = \langle \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} \rangle$$

$$Y = \langle \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} \rangle$$

- Running-time = $O(mn)$ since each table entry takes $O(1)$ time to compute
- LCS of $X \& Y = \langle B, C, B, A \rangle$

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4 B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5 D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6 A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7 B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

Constructing an LCS

- The b table returned by **LCS-LENGTH** can be used to quickly construct an **LCS** of $X\&Y$
- Begin at $b[m, n]$ and trace through the table following arrows
- Whenever you encounter a "$\nwarrow$" in entry $b[i, j]$ it implies that $x_i = y_j$ is an element of **LCS**
- The elements of **LCS** are encountered in **reverse order**

Constructing an LCS

- The recursive procedure PRINT-LCS prints out LCS in proper order
- This procedure takes $O(m+n)$ time since at least one of i and j is decremented in each stage of the recursion

$$
\begin{aligned}
&\text{PRINT-LCS}(b, X, i, j) \\
&\quad \text{if } i = 0 \text{ or} j = 0 \text{ then} \\
&\quad \text{return} \\
&\quad \text{if } b[i, j] = \text{“} \nwarrow \text{”} \text{ then} \\
&\quad\quad \text{PRINT-LCS}(b, X, i-1, j-1) \\
&\quad\quad \text{print } x_i \\
&\quad \text{else if } b[i, j] = \text{“} \uparrow \text{”} \text{ then} \\
&\quad\quad \text{PRINT-LCS}(b, X, i-1, j) \\
&\quad \text{else} \\
&\quad\quad \text{PRINT-LCS}(b, X, i, j-1)
\end{aligned}
$$

- **The initial invocation:** $\text{PRINT-LCS}(b, X, length[X], length[Y])$

Do we really need the b table (back-pointers)?

- **Question:** From which neighbor did we expand to the highlighted cell?
- **Answer:** Upper-left neighbor,because $X[i] = Y[j]$.

$\downarrow i/j \rightarrow$	$0\ y_j$	$1\ B$	$2\ D$	$3\ C$	$4\ A$	$5\ B$	$6\ A$
$0\ x_i$	0	0	0	0	0	0	0
$1\ A$	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
$2\ B$	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
$3\ C$	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
$4\ B$	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
$5\ D$	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
$6\ A$	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
$7\ B$	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

Do we really need the b table (back-pointers)?

- **Question:** From which neighbor did we expand to the highlighted cell?
- **Answer:** Left neighbor, because $X[i] \neq Y[j]$ and $LCS[i, j-1] > LCS[i-1, j]$.

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 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	↑ 3	↑ 3	↖ 4	↑ 4

Do we really need the b table (back-pointers)?

- **Question:** From which neighbor did we expand to the highlighted cell?
- **Answer:** Upper neighbor,because $X[i] \neq Y[j]$ and $LCS[i, j-1] = LCS[i-1, j]$. *(See pseudo-code to see how ties are handled.)*

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4 B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5 D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6 A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7 B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

Improving the Space Requirements

- We can eliminate the b table altogether
  - each $c[i,j]$ entry depends only on 3 other c table entries: $c[i-1,j-1]$, $c[i-1,j]$ and $c[i,j-1]$
- Given the value of $c[i,j]$:
  - We can determine in $O(1)$ time which of these 3 values was used to compute $c[i,j]$ without inspecting table b
  - We save $\Theta(mn)$ space by this method
  - However, space requirement is still $\Theta(mn)$ since we need $\Theta(mn)$ space for the c table anyway

What if we store the last 2 rows only?

- To compute $c[i, j]$, we only need $c[i-1, j-1]$, $c[i-1, j]$,and $c[i-1, j-1]$
- So, we can store only the last two rows.

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4 B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5 D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6 A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7 B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

What if we store the last 2 rows only?

- To compute $c[i, j]$, we only need $c[i-1, j-1]$, $c[i-1, j]$, and $c[i-1, j-1]$
- So, we can store only the last two rows.

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\nwarrow 1$
2 B	0	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$
3 C	0	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\uparrow 2$	$\uparrow 2$
4 B	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\leftarrow 3$
5 D	0	$\uparrow 1$	$\nwarrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 3$
6 A	0	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\uparrow 3$	$\nwarrow 4$
7 B	0	$\nwarrow 1$	$\uparrow 2$	$\uparrow 3$	$\uparrow 3$	$\nwarrow 4$	$\uparrow 4$

What if we store the last 2 rows only?

- To compute $c[i, j]$, we only need $c[i-1, j-1]$, $c[i-1, j]$, and $c[i-1, j-1]$
- So, we can store only the last two rows.
- This reduces space complexity from $\Theta(mn)$ to $\Theta(n)$.
- Is there a problem with this approach?

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4 B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5 D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6 A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7 B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

What if we store the last 2 rows only?

- **Is there a problem with this approach?**
  - We cannot construct the optimal solution because we cannot backtrace anymore.
  - This approach works if we only need the length of an LCS, not the actual LCS.

$\downarrow i/j \rightarrow$	$0 y_j$	1 B	2 D	3 C	4 A	5 B	6 A
$0 x_i$	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4 B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5 D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6 A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7 B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

Problem 4 Optimal Binary Search Tree

- Problem-1: Longest Increasing Subsequence
  - https://www.geeksforgeeks.org/longest-increasing-subsequence-dp-3/
  - https://en.wikipedia.org/wiki/Longest_increasing_subsequence#:~:text=In computer science%2C the longest,not necessarily contiguous%2C or unique.
  - https://www.youtube.com/watch?v=22s1xxRvy28&ab_channel=StableSort

- Problem-2: Edit Distance
  - https://www.geeksforgeeks.org/edit-distance-dp-5/
  - https://www.youtube.com/watch?v=tU2f2JwHmfQ&feature=youtu.be&ab_channel=PrepForTech
  - Recursive
    - https://www.youtube.com/watch?v=8Q2IEIY2pDU&ab_channel=BenLangmead
  - DP
    - https://www.youtube.com/watch?v=0KzWq118UNI&ab_channel=BenLangmead
    - https://www.youtube.com/watch?v=eAVGRWSryGo&ab_channel=BenLangmead

- Problem-3: Partition a set into two subsets such that the difference of subset sums is minimum
  - https://www.geeksforgeeks.org/partition-a-set-into-two-subsets-such-that-the-difference-of-subset-sums-is-minimum/
- Problem-4: Count number of ways to cover a distance
  - https://www.geeksforgeeks.org/count-number-of-ways-to-cover-a-distance/
- Problem-5: Find the longest path in a matrix with given constraints
  - https://www.geeksforgeeks.org/find-the-longest-path-in-a-matrix-with-given-constraints/

- Problem-6: Subset Sum Problem
  - https://www.geeksforgeeks.org/subset-sum-problem-dp-25/
- Problem-7: Optimal Strategy for a Game
  - https://www.geeksforgeeks.org/optimal-strategy-for-a-game-dp-31/
- Problem-8: 0-1 Knapsack Problem
  - https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/

- Problem-9: Boolean Parenthesization Problem
  - https://www.geeksforgeeks.org/boolean-parenthesization-problem-dp-37/
- Problem-10: Shortest Common Supersequence
  - https://www.geeksforgeeks.org/shortest-common-supersequence/
  - https://en.wikipedia.org/wiki/Shortest_common_supersequence_problem
- Problem-11: Partition Problem
  - https://www.geeksforgeeks.org/partition-problem-dp-18/
- Problem-12: Cutting a Rod
  - https://www.geeksforgeeks.org/cutting-a-rod-dp-13/

- Problem-13: Coin Change
  - https://www.geeksforgeeks.org/coin-change-dp-7/
- Problem-14: Word Break Problem
  - https://www.geeksforgeeks.org/word-break-problem-dp-32/
- Problem-15: Maximum Product Cutting
  - https://www.geeksforgeeks.org/maximum-product-cutting-dp-36/

- Problem-16: Dice Throw
  - https://www.geeksforgeeks.org/dice-throw-dp-30/
- Problem-17: Box Stacking Problem
  - https://www.geeksforgeeks.org/box-stacking-problem-dp-22/
- Problem-18: Egg Dropping Puzzle
  - https://www.geeksforgeeks.org/egg-dropping-puzzle-dp-11/

References

- Introduction to Algorithms, Third Edition | The MIT Press
- Bilkent CS473 Course Notes (new)
- Bilkent CS473 Course Notes (old)

$$-End - Of - Week - 6 - Course - Module-$$