DESIGN AND ANALYSIS OF ALGORITHMS

CS 4120/5120

DP – LONGEST COMMON SUBSEQUENCE

AGENDA

- Longest common subsequence
  - Problem definition
  - Building a model using abstraction
  - Solve the problem using DP

ELEMENTS OF DP
BRIEF REVIEW

- The four elements of dynamic programming
  - Two key ingredients
    - Optimal substructure
    - Overlapping subproblems
  - Reconstructing a solution
  - Memoization

SIMILARITY OF DNA STRANDS

- A strand of DNA consists of a string of molecules called **bases**.
  - ***A***denine, ***G***uanine, ***C***ytosine, and ***T***hymine.
- Representing each of these bases by its initial letter, we can express a strand of DNA as a string over the finite set $\{A, C, G, T\}$.

- For example, below are two DNA strands of two organism.
  - $S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA$
  - $S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA$

SIMILARITY OF DNA STRANDS
APPLICATION

- In medical/biological studies, people often need to compare two DNA strands to determine how "similar" they are.
- We might say two DNA strands are similar if
  - one is the substring of the other, or
  - the number of changes needed to turn one into the other is small, or
  - there exist a third strand S_3 in which the bases in S_3 appear in each of S_1 and S_2.
    - The bases must appear in the same order, but not necessarily consecutively.

We formalize this notation of similarity as the ***longest-common-subsequence (LCS) problem***.

SIMILARITY OF DNA STRANDS APPLICATION (CONT'D)

- Consider the LCS previously defined.
  - There exist a third strand S_3 in which the bases in S_3 appear in each of S_1 and S_2.
    - The bases must appear in the same order, but not necessarily consecutively.

- Identify the S_3 of the two DNA strands (below).
  - $S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA$
  - $S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA$

LCS PROBLEM DEFINITION THE SUBSEQUENCE

- Given a sequence $X = < x_1, x_2, \dots, x_m >$, another sequence $Z = < z_1, z_2, \dots, z_k >$ is a ***subsequence*** of X if there exists a strictly increasing sequence $< i_1, i_2, \dots, i_k >$ of indices of X such that for all $j = 1, 2, \dots, k$, we have $x_{i_j} = z_j$ (the subscript of x_{i_j} is i_j).
- Example
  - Consider two sequences $Z = < \overset{1}{B}, \overset{2}{C}, \overset{3}{D}, \overset{4}{B} >$ and $X = < \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} >$.
  - Z is a subsequence of X as there exists a strictly increasing sequence of indices of X $\underline{< 2, 3, 5, 7 >}$ that for all $j = 1, 2, \dots, k$ we have $x_{i_j} = z_j$.
    - In this example, $k =$ __4__, $i_1 =$ __2__, $i_2 =$ __3__, $i_3 =$ __5__, $i_4 =$ __7__
    - $x_{i_1} = x_{\underline{2}} =$ __B__, $x_{i_2} = x_{\underline{3}} =$ __C__, $x_{i_3} = x_{\underline{5}} =$ __D__, $x_{i_4} = x_{\underline{7}} =$ __B__.

LCS PROBLEM DEFINITION THE SUBSEQUENCE PRACTICE

- Given a sequence $X = < x_1, x_2, \dots, x_m >$, another sequence $Z = < z_1, z_2, \dots, z_k >$ is a ***subsequence*** of X if there exists a strictly increasing sequence $< i_1, i_2, \dots, i_k >$ of indices of X such that for all $j = 1, 2, \dots, k$, we have $x_{i_j} = z_j$ (the subscript of x_{i_j} is i_j).
- Consider a sequence $X = \{A,\ B,\ D,\ D,\ D,\ C,\ D,\ E,\ F,\ D,\ C,\ C,\ B\}$. Suppose Z is a subsequence of X with corresponding index sequence $< 1, 4, 6, 8, 12 >$.
  - $k =$ <u>5</u>
  - $< i_1, i_2, \dots, i_k > = <$ <u>$1, 4, 6, 8, 12$</u> $>$.
  - $Z = <$ <u>A, D, C, E, C</u> $>$.

LCS PROBLEM DEFINITION THE COMMON SUBSEQUENCE

- Given two sequences X and Y, we say that a sequence Z is a ***common subsequence*** of X and Y if Z is a subsequence of both X and Y.
- Example
  - Consider two sequences: $X = < A, B, C, B, D, A, B >$ and $Y = < B, D, C, A, B, A >$.
  - Sequence $< B, C, A >$ is ***a* common subsequence** of X and Y.

THE PROBLEM DEFINITION OF THE LONGEST COMMON SUBSEQUENCE

- In the ***longest-common-subsequence (LCS) problem***, we are given two sequences

$$X =< x_1, x_2, \dots, x_m > \text{ and } Y =< y_1, y_2, \dots, y_n > \text{ and}$$

  wish to find a ***maximum*** length ***common subsequence*** of X and Y.

- In the problem definition,
  - $X.length =$ ________________, and the elements of sequence X are denoted by **lowercase** $\boldsymbol{x}$.
  - $Y.length =$ ________________, and the elements of sequence Y are denoted by **lowercase** $\boldsymbol{y}$.
  - The lengths of X and Y are not necessarily the same.

THE LONGEST-COMMON-SUBSEQUENCE PROBLEM

- Example
  - Consider two sequences $X = < A, B, C, B, D, A, B >$ and $Y = < B, D, C, A, B, A >$.
  - List the ***common subsequence***s of X and Y.

    $< A, B >, < A, B, A >$ (2)
    $< B, C >, < B, C, B >, < \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{B}, \boldsymbol{A} >, < B, C, A >, < B, C, A, B >, \cancel{< B, C, B >}$ (5)
    $< C, B >, < C, B, A >, < C, A >, < C, A, B >$ (4)
    $< B, D >, < B, D, A >, < \boldsymbol{B}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{B} >, < B, A >, < B, A, B >, < B, B >$ (6)
    $< A, B >$ (1)
  - The ***longest common subsequence*** of X and Y is $Z = \underline{< B, C, B, A >}$, or $Z = \underline{< B, D, A, B >}$.

SOLVING THE LCS PROBLEM

- The problem description has the phrase ***maximum length***, which indicates this is an ***optimization problem***.
  - We shall now begin the steps of developing a DP.
    - We will find the two key ingredients of DP along the way.

DYNAMIC PROGRAMMING CHECKLIST

- Here is a checklist of the qualifications of a DP problem.
  - ❑ Optimization problem
  - ❑ Two key ingredients
    - ❑ Optimal substructure
    - ❑ Overlapping subproblems

OPTIMAL SUBSTRUCTURE
NOTATIONS

- Let $X =< x_1, x_2, \ldots, x_m >$ and $Y =< y_1, y_2, \ldots, y_n >$ be two sequence.
- Let $Z =< z_1, z_2, \ldots, z_k >$ be any LCS of X and Y.
- Given a sequence $X =< x_1, x_2, \ldots, x_m >$, we define the ith ***prefix*** of X, for $i = \mathbf{0}, 1, \ldots, m$, as $X_i =< x_1, x_2, \ldots, x_i >$
  - X_0 is the empty sequence.

OPTIMAL SUBSTRUCTURE NOTATIONS PRACTICE

- Consider the two sequences $P = < \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} >$ and $Q = < \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B} >$.
- Sequence $Z = < B, D, A, B >$ is an LCS of P and Q.
- Fill out the following blanks.
  - The sequence of indexes of P corresponding to Z is $\underline{< 2, 5, 6, 7 >}$ or $\underline{< 4, 5, 6, 7 >}$.
  - The sequence of indexes of Q corresponding to Z is $\underline{< 1, 2, 4, 5 >}$.
  - $p_3 = \underline{C}$, $p_6 = \underline{A}$,
  - $P_3 = \underline{< A, B, C >}$, $P_6 = \underline{< A, B, C, B, D, A >}$,
  - $q_4 = \underline{A}$, $Q_4 = \underline{< B, D, C, A >}$, $Q_0 = \underline{\emptyset}$.

DISCOVERING THE OPTIMAL SUBSTRUCTURE

- **General steps**
  - **Step 1**: A solution to the problem consists of making a choice.
  - **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.
  - **Step 3**: Given this choice, you determine which subproblems ensue and how to best characterize the resulting space of subproblems.
  - **Step 4**: Show the solutions to the subproblems used within an optimal solution to the problem must themselves be optimal by using a "cut-and-paste" technique.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 1

- **Step 1**: A solution to the problem consists of making a choice.

- Consider the two pairs of sequences shown below.
  - Pair #1: $P = < A, B, C, B, D, A, B >$ and $Q = < B, D, C, A, B >$.
  - Pair #2: $X = < A, B, C, B, D, A, B >$ and $Y = < B, D, C, A, B, A >$.
- Observations
  - Pair #1: Both sequences end in the same letter B.
  - Pair #2: The two sequences, X and Y, end in different letters.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 1 (CONT'D)

- **Step 1**: A solution to the problem consists of making a choice.

- Consider the two pairs of sequences shown below.
  - Pair #1: $P = < A, B, C, B, D, A, B >$ and $Q = < B, D, C, A, B >$. $\Longrightarrow$ A possible LCS is $< \cdots, \ldots, \cdots, B >$
  - Pair #2: $X = < A, B, C, B, D, A, B >$ and $Y = < B, D, C, A, B, A >$.
- **Making a choice**
  - Pair #1: Obviously, the ending letter B is included in an LCS of P and Q.
  - Pair #2: Use Z to denote an LCS of X and Y. Either Z and X do not end in the same letter, or Z and Y do not end in the same letter.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 2

- **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.
  - At this point, you do not concern yourself with how to determine this choice.

- Unlike the rod-cutting problem or the matrix-chain multiplication problem, there is not a one-size-fits-all characterization the problem.
- We need to characterize the problem case-by-case.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 2 (CONT'D)

- **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.
  - At this point, you do not concern yourself with how to determine this choice.
- **Case 1**: $P = < \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} >$ and $Q = < \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B} >, p_7 = q_5 = B.$
  - Suppose that we are given the choice that including $p_7 = q_5 = B$ in an LCS of P and Q.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 2 (CONT'D)

- **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.
  - At this point, you do not concern yourself with how to determine this choice.

- **Case 2**: $X = < \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{\cancel{B}} >$ and $Y = < \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A} >$,where $x_7 \neq y_6$.
  - Use $Z =< z_1, z_2, \ldots, z_k >$ to denote an LCS of X and Y, where
  - Suppose that we are given the choice that $\boldsymbol{x_7}$ is not included in the LCS Z.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 2 (CONT'D)

- **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.
  - At this point, you do not concern yourself with how to determine this choice.

- **Case 3**: $X = < \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} >$ and $Y = < \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{\cancel{A}} >$, where $x_7 \neq y_6$.
  - Use $Z =< z_1, z_2, \dots, z_k >$ to denote an LCS of X and Y, where
  - Suppose that we are given the choice that y_6 is not included in the LCS Z.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 3

- **Step 3**: Given this choice, you determine which subproblems ensue and how to best characterize the resulting space of subproblems.

- **Case 1**: $P = <\overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B}>$ and $Q = <\overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}>$, $p_7 = q_5 = B$.
  - Suppose that we are given the choice that including $p_7 = q_5 = B$ in an LCS of P and Q.
  - The subproblem can be formulated as finding an LCS of
    - $< A, B, C, B, D, A, \cancel{B} >$ or $P_{7-1} = P_6$ and
    - $< B, D, C, A, \cancel{B} >$ or $Q_{5-1} = Q_4$.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 3 (CONT'D)

- **Step 3**: Given this choice, you determine which subproblems ensue and how to best characterize the resulting space of subproblems.

- **Case 2**: $X = <\overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{\cancel{B}}>$ and $Y = <\overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{A}>$, where $x_7 \neq y_6$.
  - Use $Z =< z_1, z_2, \ldots, z_k >$ to denote an LCS of X and Y, where
  - Suppose that we are given the choice that $\boldsymbol{x_7}$ is not included in the LCS Z.
  - The subproblem can be formulated as finding an LCS of
    - $< A, B, C, B, D, A, \cancel{B} >$ or $X_{7-1} = X_6$ and
    - $< B, D, C, A, B, A >$ or Y.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 3 (CONT'D)

- **Step 3**: Given this choice, you determine which subproblems ensue and how to best characterize the resulting space of subproblems.

- **Case 3**: $X = < \overset{1}{A}, \overset{2}{B}, \overset{3}{C}, \overset{4}{B}, \overset{5}{D}, \overset{6}{A}, \overset{7}{B} >$ and $Y = < \overset{1}{B}, \overset{2}{D}, \overset{3}{C}, \overset{4}{A}, \overset{5}{B}, \overset{6}{\cancel{A}} >$, where $x_7 \neq y_6$.
  - Use $Z =< z_1, z_2, \dots, z_k >$ to denote an LCS of X and Y, where
  - Suppose that we are given the choice that $\boldsymbol{y_6}$ is not included in the LCS Z.
  - The subproblem can be formulated as finding an LCS of
    - $< A, B, C, B, D, A, B >$ or X __________ and
    - $< B, D, C, A, B, \cancel{A} >$ or $Y_{6-1} = Y_5$ __________.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 3 (CONT'D)

- **Step 3**: Given this choice, you determine which subproblems ensue and how to best characterize the resulting space of subproblems.

- **Characterization**
  - Let $X =< x_1, x_2, \ldots, x_m >$ and $Y =< y_1, y_2, \ldots, y_n >$, and let $Z =< z_1, z_2, \ldots, z_k >$ be any LCS of X and Y.
    - **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}.
    - **Case 2**: If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
    - **Case 3**: If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4

- **Step 4**: Show the solutions to the subproblem used within an optimal solution to the problem must themselves be optimal by using a "cut-and-paste" technique.
  - Prove the correctness of the characterizations of the three cases.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4 CASE 1

- **Step 4**: The proof of the optimality of the solution to the subproblem.

- Let $X = < x_1, x_2, \dots, x_m >$ and $Y = < y_1, y_2, \dots, y_n >$, and let $Z = < z_1, z_2, \dots, z_k >$ be any LCS of X and Y.
  - **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}.
    - The optimality of the characterization is ***two-fold***
      - $\boldsymbol{z_k = x_m = y_n}$
      - Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4 CASE 1 - 1

- **Step 4**: The proof of the optimality of the solution to the subproblem.

- Let $X =< x_1, x_2, \ldots, x_m >$ and $Y =< y_1, y_2, \ldots, y_n >$, and let $Z =< z_1, z_2, \ldots, z_k >$ be any LCS of X and Y.
  - **Case I**: 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 of $\boldsymbol{z_k = x_m = y_n}$, i.e., the ending letter of the LCS Z is also the ending letter of X and Y.

    i. **Assume** that ______________________________.

    Assuming the opposite of the goal

    ii. We can ***append*** ___________ to ______ to **create a new** LCS Z', and $Z'.length =$ __________.

    iii. Obviously, $Z'.length$ _________ $k = Z.length$, **contradicting the supposition** that _____ is an LCS of X and Y.

    iv. **Therefore**, __________________.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4 CASE 1 - 2

- **Step 4**: The proof of the optimality of the solution to the subproblem.

- Let $X =< x_1, x_2, \ldots, x_m >$ and $Y =< y_1, y_2, \ldots, y_n >$, and let $Z =< z_1, z_2, \ldots, z_k >$ be any LCS of X and Y.
  - **Case I**: 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}.

Assuming the opposite of the goal

i. **Assume** that ____.

ii. There exist an LCS of X_{m-1} and Y_{n-1}, denoted by W, $W.length >$ __________.

iii. We can **construct a new** LCS Z' by appending ___________ to _____, $Z'.length =$ ___________ $+1$.

iv. Obviously, $Z'.length$ _____ $k = Z.length$, **contradicting the supposition** that _____ is an LCS of X and Y.

v. **Therefore**, ____.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4 CASE 2

- **Step 4**: The proof of the optimality of the solution to the subproblem.

- Let $X =< x_1, x_2, \dots, x_m >$ and $Y =< y_1, y_2, \dots, y_n >$, and let $Z =< z_1, z_2, \dots, z_k >$ be any LCS of X and Y.
  - **Case 2**: If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
    - i. **Assume** that ____.
    - ii. There exist an LCS of X_{m-1} and Y, denoted by W.
    - iii. Then W itself is also a ***common sequence*** of _____ and _____.
    - iv. Obviously, $W.length >$ ____, **contradicting the supposition** that ______ is an LCS of X and Y.
    - v. **Therefore**, ____.

Assuming the opposite of the goal

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4 CASE 3

- **Step 4**: The proof of the optimality of the solution to the subproblem.

- Let $X =< x_1, x_2, \dots, x_m >$ and $Y =< y_1, y_2, \dots, y_n >$, and let $Z =< z_1, z_2, \dots, z_k >$ be any LCS of X and Y.
  - **Case 3**: If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.
    - i. **Assume** that ____.
    - ii. There exist an LCS of ________ and _______, denoted by W.
    - iii. Then W itself is also a ***common sequence*** of _____ and _____.
    - iv. Obviously, $W.length >$ ____, **contradicting the supposition** that ______ is an LCS of X and Y.
    - v. **Therefore**, ____.

Assuming the opposite of the goal

DISCOVERING THE OPTIMAL SUBSTRUCTURE, DONE

- **Theorem 15.1 (Optimal substructure of an LCS)**
  - Let $X =< x_1, x_2, \dots, x_m >$ and $Y =< y_1, y_2, \dots, y_n >$, and let $Z =< z_1, z_2, \dots, z_k >$ be any LCS of X and Y.
    - **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}.
    - **Case 2**: If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
    - **Case 3**: If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

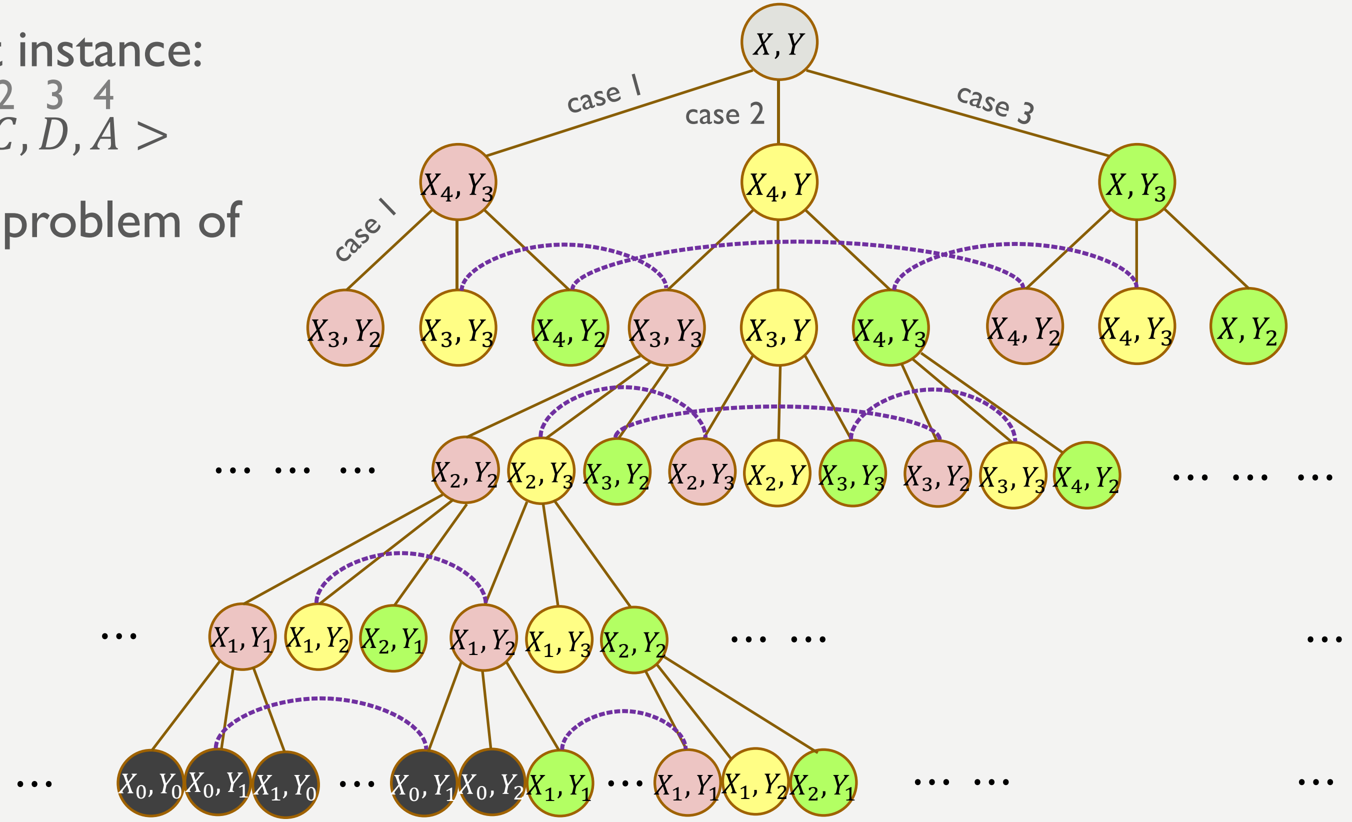
- **The longest Z of all three cases is an LCS of X and Y.**

DYNAMIC PROGRAMMING CHECKLIST

- Here is a checklist of the qualifications of a DP problem.
  - ❑ Optimization problem
  - ❑ Two key ingredients
    - ❑ Optimal substructure
    - ❑ Overlapping subproblems

DISCOVER OVERLAPPING SUBPROBLEMS

- Draw the subproblem graph for input instance:
  - $X = \langle \overset{1}{C}, \overset{2}{B}, \overset{3}{D}, \overset{4}{A}, \overset{5}{B} \rangle$ and $Y = \langle \overset{1}{B}, \overset{2}{C}, \overset{3}{D}, \overset{4}{A} \rangle$
- Each vertex X, Y represents the (sub)problem of finding an LCS of X and Y
  - Each vertex has a degree ≤ 3.
    - **Case 1**: **Left** child represents the subproblem for X_{m-1} and Y_{n-1}.
    - **Case 2**: **Middle** child represents the subproblem for X_{m-1} and Y.
    - **Case 3**: **Right** child represents the subproblem for X and Y_{n-1}.



DYNAMIC PROGRAMMING CHECKLIST

- Here is a checklist of the qualifications of a DP problem.
  - ☐ Optimization problem
  - ☐ Two key ingredients
    - ☐ Optimal substructure
    - ☐ Overlapping subproblems

APPLYING DP
STEP 1

- **Step 1**: Characterize the structure of an optimal solution
  - Discover the ***optimal substructure*** of the problem.

- **Theorem 15.1 (Optimal substructure of an LCS)**
  - Let $X =< x_1, x_2, \dots, x_m >$ and $Y =< y_1, y_2, \dots, y_n >$, and let $Z =< z_1, z_2, \dots, z_k >$ be any LCS of X and Y.
    - **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}.
    - **Case 2**: If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
    - **Case 3**: If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

APPLYING DP
STEP 2

- **Step 2**: Recursively define the ***value*** of an optimization.
  - Take advantage of the optimal substructure to recursively compute the optimal ***value***.
  - The ***value*** of an optimization in the LCS problem means the length of an LCS of the two inputs.

- Define $c[i,j]$ to be the length of an LCS of the sequences X_i and Y_j.
  - Theorem 15.1 (Optimal substructure of an LCS) mapped onto input X_i and Y_j.
    - **Case 1**: If $x_i = y_j$, then $z_k = x_i = y_j$ and Z_{k-1} is an LCS of X_{i-1} and Y_{j-1}. $\Longrightarrow c[i,j] = c[i-1, j-1] + 1$
    - **Case 2**: If $x_i \neq y_j$, then $z_k \neq x_i$ implies that Z is an LCS of X_{i-1} and Y. $\Longrightarrow c[i,j] = c[i-1, j]$ if $x_i \neq y_j$
    - **Case 3**: If $x_i \neq y_j$, then $z_k \neq y_j$ implies that Z is an LCS of X and Y_{j-1}. $\Longrightarrow c[i,j] = c[i, j-1]$ if $x_i \neq y_j$

APPLYING DP
STEP 2 (CONT'D)

- **Step 2**: Recursively define the ***value*** of an optimization.
  - Take advantage of the optimal substructure to recursively compute the optimal ***value***.
  - The ***value*** of an optimization in the LCS problem means the length of an LCS of the two inputs.

- Define $c[i, j]$ to be the length of an LCS of the sequences X_i and Y_j.

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

APPLYING DP
STEP 3

- **Step 3**: Compute the ***value*** of an optimal solution.

- The LCS-LENGTH algorithm
  - **Input**
    - $X =< x_1, x_2, \dots x_{m>}$ and
    - $Y =< y_1, y_2, \dots, y_n >$
  - **Bottom-up** strategy
  - **Memoziation**
  - **Problem solved** by line 8 ~ 17

	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
5	$c[i, 0] = 0$
6	**for** $j = 0$ **to** n
7	$c[0, j] = 0$
8	**for** $i = 1$ **to** m
9	**for** $j = 1$ **to** n
10	**if** $x_i == y_j$ // case 1
11	$c[i, j] = c[i-1, j-1] + 1$
12	$b[i, j] =$"↖"
13	**elseif** $c[i-1, j] \geq c[i, j-1]$ // case 2
14	$c[i, j] = c[i-1, j]$
15	$b[i, j] =$ "↑"
16	**else** $c[i, j] = c[i, j-1]$ // case 3
17	$b[i, j] =$ "←"
18	**return** c and b

APPLYING DP
STEP 3 (CONT'D)

- **Step 3**: Compute the ***value*** of an optimal solution.

- The table $\boldsymbol{c[0..m, 0..n]}$
  - $m + 1$ rows and $n + 1$ columns
  - An entry $\boldsymbol{c[i, j]}$ stores the length of an LCS of sequences X_i and Y_j.

LCS-LENGTH (X, Y)
$m = X.length$
$n = Y.length$
let $b[1..m, 1..n]$ and $\boldsymbol{c[0..m, 0..n]}$ be new tables
for $i = 1$ **to** m
$c[i, 0] = 0$
for $j = 0$ **to** n
$c[0, j] = 0$
for $i = 1$ **to** m
for $j = 1$ **to** n
if $x_i == y_j$
$\boldsymbol{c[i, j]} = c[i-1, j-1] + 1$
$b[i, j] =$ "↖"
elseif $c[i-1, j] \geq c[i, j-1]$
$\boldsymbol{c[i, j]} = c[i-1, j]$
$b[i, j] =$ "↑"
else $\boldsymbol{c[i, j]} = c[i, j-1]$
$b[i, j] =$ "←"
return c and b

APPLYING DP
STEP 3 (CONT'D)

- **Step 3**: Compute the ***value*** of an optimal solution.

- The table $\boldsymbol{b[1..m, 1..n]}$
  - m rows and n columns
  - The table stores **the choices** made when computing the length of an LCS.
  - An entry $\boldsymbol{b[i, j]}$ stores the choice that lead to the values in entry $\boldsymbol{c[i, j]}$.

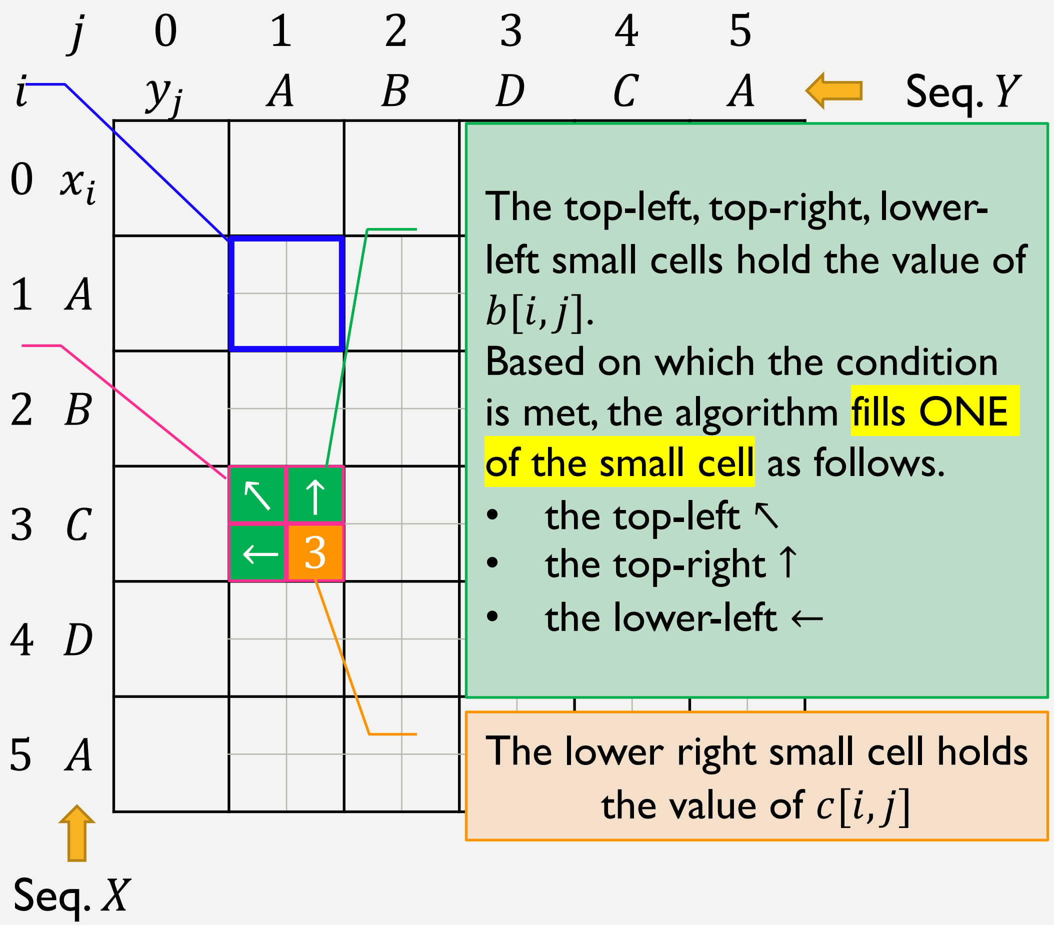
```
LCS-LENGTH (X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

MAINTAINING THE TWO TABLES $b[1..m, 1..n]$ AND $c[0..m, 0..n]$

- The two tables are maintained in one place.
- Input
  - $X = < A, B, C, D, A >$
    - Stored in rows
  - $Y = < A, B, D, C, A >$
    - Stored in columns
- The following slides show the execution of LCS-LENGTH on X and Y.

Each "big" cell with **thick** border corresponds to one entry of the c table and the b table.

Each "big" cell further splits into four small cells.



LCS-LENGTH INITIALIZATION

- Initialization

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

```
LCS-LENGTH (X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

LCS-LENGTH IN ACTION
ITERATION 1×1

- Compute the ***optimal*** value
  - $i = 1$
  - $j = 1$

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

```
LCS-LENGTH (X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

LCS-LENGTH IN ACTION
ITERATION 1×2

- Compute the ***optimal*** value
  - $i = 1$
  - $j = 2$

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

```
LCS-LENGTH (X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

LCS-LENGTH IN ACTION
ITERATION 1×3

- Compute the ***optimal*** value
  - $i = 1$
  - $j = 3$

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

```
LCS-LENGTH (X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

LCS-LENGTH IN ACTION
ITERATION 1×4

- Compute the ***optimal*** value
  - $i = 1$
  - $j = 4$

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

```
LCS-LENGTH (X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

LCS-LENGTH IN ACTION
ITERATION 1×5

- Compute the ***optimal*** value
  - $i = 1$
  - $j = 5$

	j	0	1	2	3	4	5
i		y_j	A	B	D	C	A
0	x_i	0	0	0	0	0	0
1	A	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1
2	B	0					
3	C	0					
4	D	0					
5	A	0					

LCS-LENGTH (X, Y)

```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

LCS-LENGTH IN ACTION
ITERATION 2×1

- Compute the ***optimal*** value
  - $i = 2$
  - $j = 1$

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

```
LCS-LENGTH (X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

LCS-LENGTH IN ACTION
ITERATION 2×2

- Compute the ***optimal*** value
  - $i = 2$
  - $j = 2$

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

LCS-LENGTH (X, Y)

```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

LCS-LENGTH IN ACTION
ITERATIONS TO GO

- Compute the ***optimal*** value
  - $i = 2$
  - $j = 3, 4, 5$
  - $\dots$
  - $i = 5$
  - $j = 1, 5$

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

LCS-LENGTH (X, Y)

```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

BACK TO APPLYING DP
STEP 3 (CONT'D)

- **Step 3**: Compute the ***value*** of an optimal solution.

- Running time of LCS-LENGTH
  $T(n) = \Theta($ __________ $)$

```
LCS-LENGTH (X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

LCS-LENGTH (X, Y)
PRACTICE #1

- Which of the following c, b tables can the resulting table of running the LCS-LENGTH algorithm? Choose all that fit.
  - For chosen table(s), given an example of the input instance?

i \ j	0 (y_j)	1	2	3	4	5
0 (x_i)	0	0	0	0	0	0
1	0	↖ 1	↖ 1	↖ 1	↖ 1	↖ 1
2	0	↖ 1	↖ 1	↖ 1	↖ 1	↖ 1
3	0	↖ 1	↖ 1	↖ 1	↖ 1	↖ 1
4	0	↖ 1	↖ 1	↖ 1	↖ 1	↖ 1
5	0	↖ 1	↖ 1	↖ 1	↖ 1	↖ 1

A

i \ j	0 (y_j)	1	2	3	4	5
0 (x_i)	0	0	0	0	0	0
1	0	↖ 1	← 1	← 1	← 1	← 1
2	0	↑ 1	↖ 2	← 1	← 1	← 1
3	0	↑ 1	↑ 1	↖ 3	← 1	← 1
4	0	↑ 1	↑ 1	↑ 1	↖ 4	← 1
5	0	↑ 1	↑ 1	↑ 1	↑ 1	↖ 5

B

i \ j	0 (y_j)	1	2	3	4	5
0 (x_i)	0	0	0	0	0	0
1	0	← 1	← 1	← 1	← 1	← 1
2	0	← 1	← 1	← 1	← 1	← 1
3	0	← 1	← 1	← 1	← 1	← 1
4	0	← 1	← 1	← 1	← 1	← 1
5	0	← 1	← 1	← 1	← 1	← 1

C

LCS-LENGTH (X, Y)
$m = X.length$
$n = Y.length$
let $\boldsymbol{b}[\mathbf{1}..\boldsymbol{m}, \mathbf{1}..\boldsymbol{n}]$ and $\boldsymbol{c}[\mathbf{0}..\boldsymbol{m}, \mathbf{0}..\boldsymbol{n}]$ be new tables
for $i = 1$ **to** m
$c[i, 0] = 0$
for $j = 0$ **to** n
$c[0, j] = 0$
for $i = 1$ **to** m
for $j = 1$ **to** n
if $x_i == y_j$
$c[i, j] = c[i-1, j-1] + 1$
$b[i, j] =$ "↖"
elseif $c[i-1, j] \geq c[i, j-1]$
$c[i, j] = c[i-1, j]$
$b[i, j] =$ "↑"
else $c[i, j] = c[i, j-1]$
$b[i, j] =$ "←"
return c and b

LCS-LENGTH (X, Y) PRACTICE #2

- Which of the following c, b tables can the resulting table of running the LCS-LENGTH algorithm? Choose all that fit.
  - For chosen table(s), given an example of the input instance?

	j	0	1	2	3	4
i		y_j				
0	x_i	0	0	0	0	0
1		0	↖ 1	← 1	← 1	← 1
2		0	↑ 1	↖ 2	← 2	← 2
3		0	↑ 1	↑ 2	↖ 3	← 3
4		0	↑ 1	↑ 2	↑ 3	↖ 4
5		0	↑ 1	↑ 2	↑ 3	↑ 4

A

	j	0	1	2	3	4
i		y_j				
0	x_i	0	0	0	0	0
1		0	← 0	← 0	← 0	← 0
2		0	← 0	← 0	← 0	← 0
3		0	← 0	← 0	← 0	← 0
4		0	← 0	← 0	← 0	← 0
5		0	← 0	← 0	← 0	← 0

B

	j	0	1	2	3	4
i		y_j				
0	x_i	0	0	0	0	0
1		0	↑ 1	← 1	← 1	← 1
2		0	↑ 1	↑ 1	↑ 1	↑ 1
3		0	↑ 1	↑ 1	↑ 1	↑ 1
4		0	↑ 1	↑ 1	↑ 1	↑ 1
5		0	↑ 1	↑ 1	↑ 1	↑ 1

C

LCS-LENGTH (X, Y)
$m = X.length$
$n = Y.length$
let $\boldsymbol{b}[\mathbf{1}..\boldsymbol{m}, \mathbf{1}..\boldsymbol{n}]$ and $\boldsymbol{c}[\mathbf{0}..\boldsymbol{m}, \mathbf{0}..\boldsymbol{n}]$ be new tables
for $i = 1$ **to** m
$c[i, 0] = 0$
for $j = 0$ **to** n
$c[0, j] = 0$
for $i = 1$ **to** m
for $j = 1$ **to** n
if $x_i == y_j$
$c[i, j] = c[i-1, j-1] + 1$
$b[i, j] =$ "↖"
elseif $c[i-1, j] \geq c[i, j-1]$
$c[i, j] = c[i-1, j]$
$b[i, j] =$ "↑"
else $c[i, j] = c[i, j-1]$
$b[i, j] =$ "←"
return c and b

APPLYING DP
STEP 4

- **Step 4**: Construct the optimal solution from the computed information.

- PRINT-LCS algorithm
  - Print the LCS found by LCS-LENGTH algorithm.
  - The initial call is PRINT-LCS (b, X, m, n)
    - $m = X.length$
    - $n = Y.length$

PRINT-LCS (b, X, i, j)
if $i == 0$ or $j == 0$
return
if $b[i, j] ==$ "↖"
PRINT-LCS $(b, X, i-1, j-1)$
print x_i
elseif $b[i, j] ==$ "↑"
PRINT-LCS $(b, X, i-1, j)$
else PRINT-LCS $(b, X, i, j-1)$

APPLYING DP STEP 4 ON A REAL TABLE

```
PRINT-LCS (b, X, i, j)
if i == 0 or j == 0
        return
if b[i, j] == "↖"
        PRINT-LCS (b, X, i − 1, j − 1)
        print x_i
elseif b[i, j] == "↑"
        PRINT-LCS (b, X, i − 1, j)
else PRINT-LCS (b, X, i, j − 1)
```

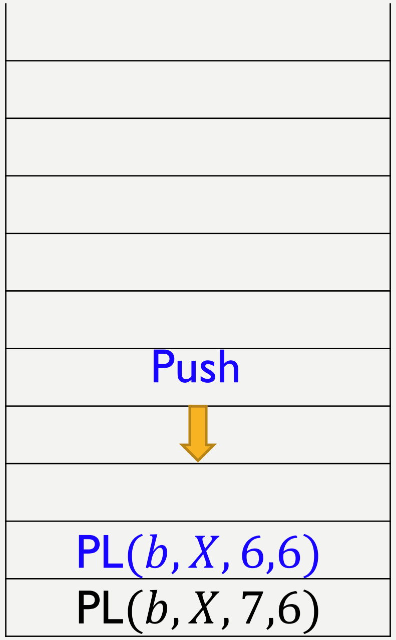
- **Step 4**: Construct the optimal solution from the computed information.
- The c, b table displayed on the lower-right corner is the resulting table of running LCS-LENGTH (X, Y)
  - $X =< A, B, C, B, D, A, B >$
  - $Y = < B, D, C, A, B, A >$
- To construct the LCS found by LCS-LENGTH (X, Y), we need to call PRINT-LCS $(b, X,$ ___ , ___ $)$

	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

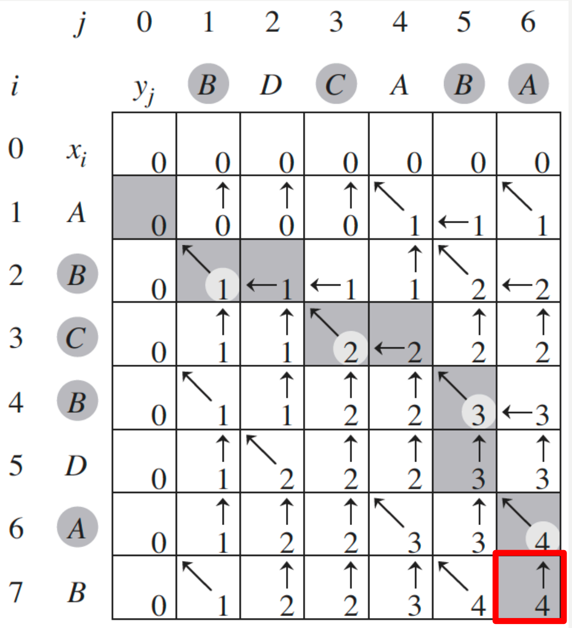
APPLYING DP STEP 4 ON A REAL TABLE

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

- The CPU will start at computing PRINT-LCS $(b, X, 7, 6)$
- Output



Stack



APPLYING DP STEP 4 ON A REAL TABLE

- The CPU will start at computing PRINT-LCS $(b, X, 7, 6)$
- Output

```
PRINT-LCS (b, X, i, j)
if i == 0 or j == 0
        return
if b[i, j] == "↖"
        PRINT-LCS (b, X, i − 1, j − 1)
        print x_i
elseif b[i, j] == "↑"
        PRINT-LCS (b, X, i − 1, j)
else PRINT-LCS (b, X, i, j − 1)
```

Push

PL$(b, X, 5,5)$

PL$(b, X, 6,6)$

PL$(b, X, 7,6)$

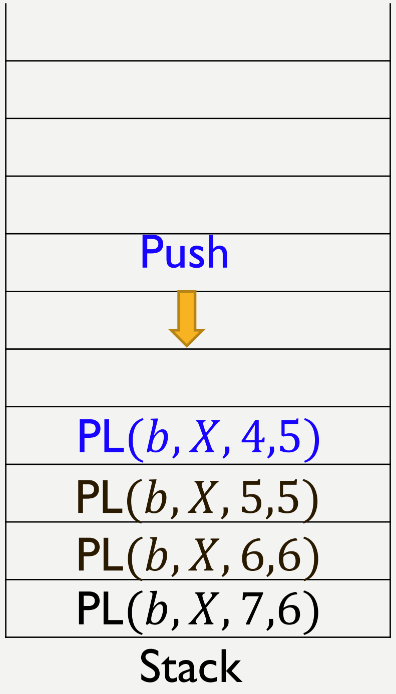
Stack

	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

APPLYING DP STEP 4 ON A REAL TABLE

- The CPU will start at computing PRINT-LCS $(b, X, 7, 6)$
- Output

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

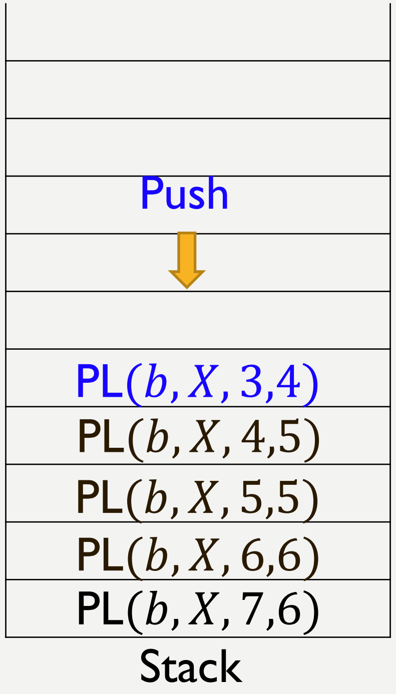


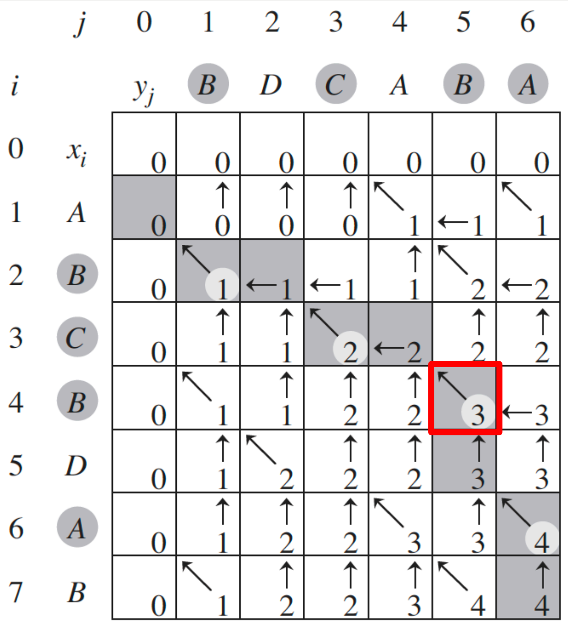
i \ j		0	1	2	3	4	5	6
	y_j		B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

APPLYING DP STEP 4 ON A REAL TABLE

- The CPU will start at computing PRINT-LCS $(b, X, 7, 6)$
- Output

PRINT-LCS (b, X, i, j)
if $i == 0$ or $j == 0$
return
if $b[i,j] ==$ "↖"
PRINT-LCS $(b, X, i-1, j-1)$
print x_i
elseif $b[i,j] ==$ "↑"
PRINT-LCS $(b, X, i-1, j)$
else PRINT-LCS $(b, X, i, j-1)$

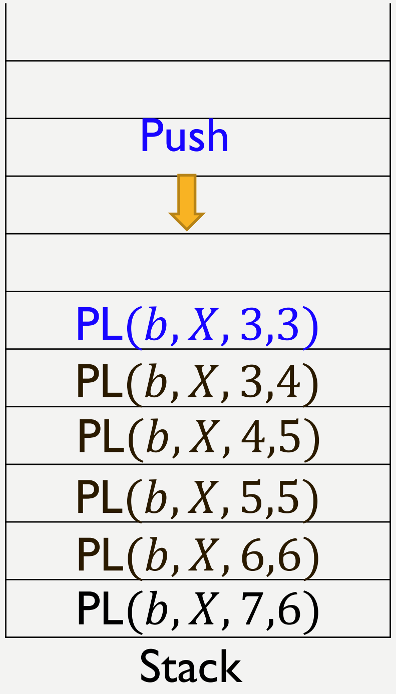


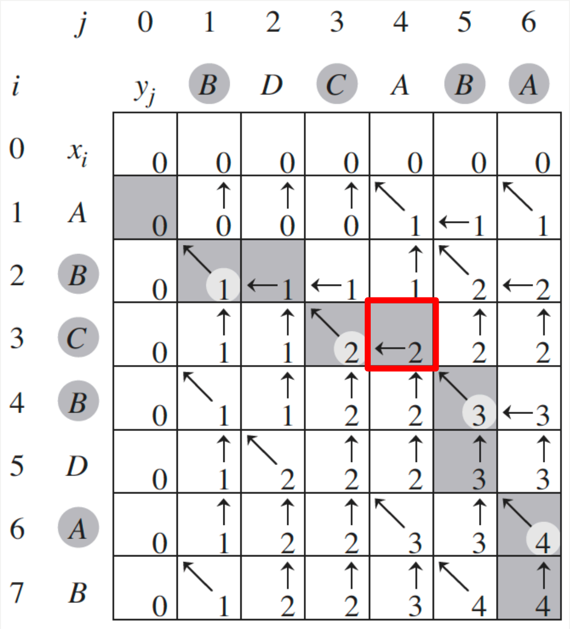


APPLYING DP STEP 4 ON A REAL TABLE

```
PRINT-LCS (b, X, i, j)
if i == 0 or j == 0
        return
if b[i, j] == "↖"
        PRINT-LCS (b, X, i − 1, j − 1)
        print x_i
elseif b[i, j] == "↑"
        PRINT-LCS (b, X, i − 1, j)
else PRINT-LCS (b, X, i, j − 1)
```

- The CPU will start at computing PRINT-LCS $(b, X, 7, 6)$
- Output





APPLYING DP STEP 4 ON A REAL TABLE

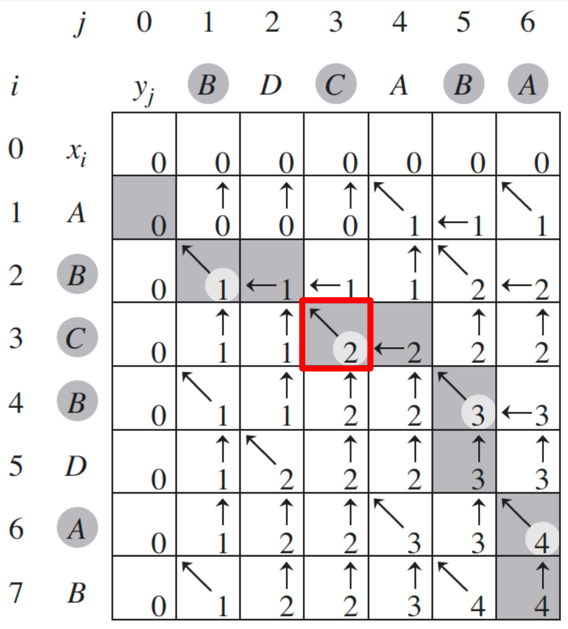
- The CPU will start at computing PRINT-LCS $(b, X, 7, 6)$
- Output

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

Push

PL$(b, X, 2,2)$
PL$(b, X, 3,3)$
PL$(b, X, 3,4)$
PL$(b, X, 4,5)$
PL$(b, X, 5,5)$
PL$(b, X, 6,6)$
PL$(b, X, 7,6)$

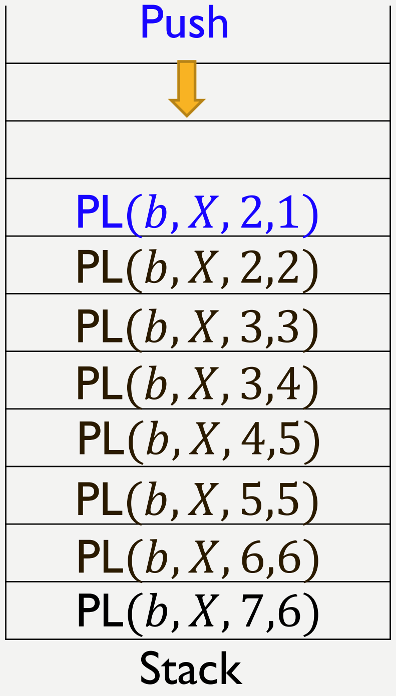
Stack

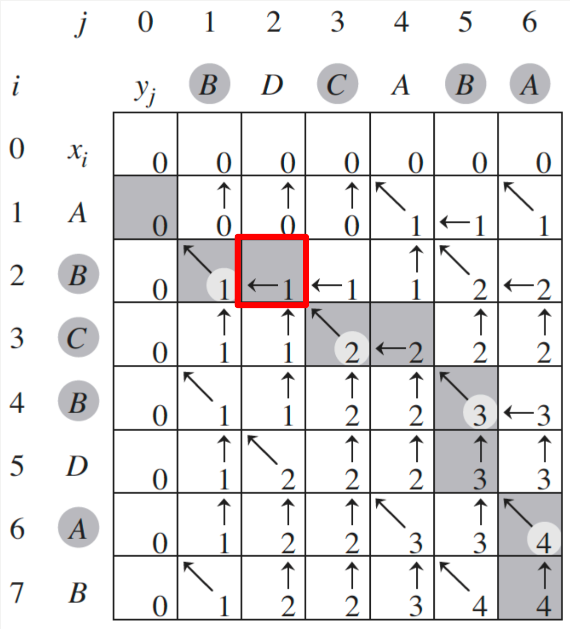


APPLYING DP STEP 4 ON A REAL TABLE

```
PRINT-LCS (b, X, i, j)
if i == 0 or j == 0
        return
if b[i, j] == "↖"
        PRINT-LCS (b, X, i − 1, j − 1)
        print x_i
elseif b[i, j] == "↑"
        PRINT-LCS (b, X, i − 1, j)
else PRINT-LCS (b, X, i, j − 1)
```

- The CPU will start at computing PRINT-LCS $(b, X, 7, 6)$
- Output





APPLYING DP STEP 4 ON A REAL TABLE

- The CPU will start at computing PRINT-LCS $(b, X, 7, 6)$
- Output

PRINT-LCS (b, X, i, j)
if $i == 0$ or $j == 0$
return
if $b[i, j] ==$ "↖"
PRINT-LCS $(b, X, i-1, j-1)$
print x_i
elseif $b[i, j] ==$ "↑"
PRINT-LCS $(b, X, i-1, j)$
else PRINT-LCS $(b, X, i, j-1)$

Push

Stack
PL$(b, X, 1,0)$
PL$(b, X, 2,1)$
PL$(b, X, 2,2)$
PL$(b, X, 3,3)$
PL$(b, X, 3,4)$
PL$(b, X, 4,5)$
PL$(b, X, 5,5)$
PL$(b, X, 6,6)$
PL$(b, X, 7,6)$

Stack

	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

APPLYING DP STEP 4 ON A REAL TABLE

```
PRINT-LCS (b, X, i, j)
if i == 0 or j == 0
        return
if b[i, j] == "↖"
        PRINT-LCS (b, X, i − 1, j − 1)
        print x_i
elseif b[i, j] == "↑"
        PRINT-LCS (b, X, i − 1, j)
else PRINT-LCS (b, X, i, j − 1)
```

- The CPU will start at computing PRINT-LCS $(b, X, 7, 6)$
- Output

  B C B A

- A letter in the found LCS is printed before the stack pops out a PRINT-LCS (b, X, i, j) where $b[i, j]$ holds a "↖"

Pop

Stack
PL$(b, X, 1{,}0)$
PL$(b, X, 2{,}1)$
PL$(b, X, 2{,}2)$
PL$(b, X, 3{,}3)$
PL$(b, X, 3{,}4)$
PL$(b, X, 4{,}5)$
PL$(b, X, 5{,}5)$
PL$(b, X, 6{,}6)$
PL$(b, X, 7{,}6)$

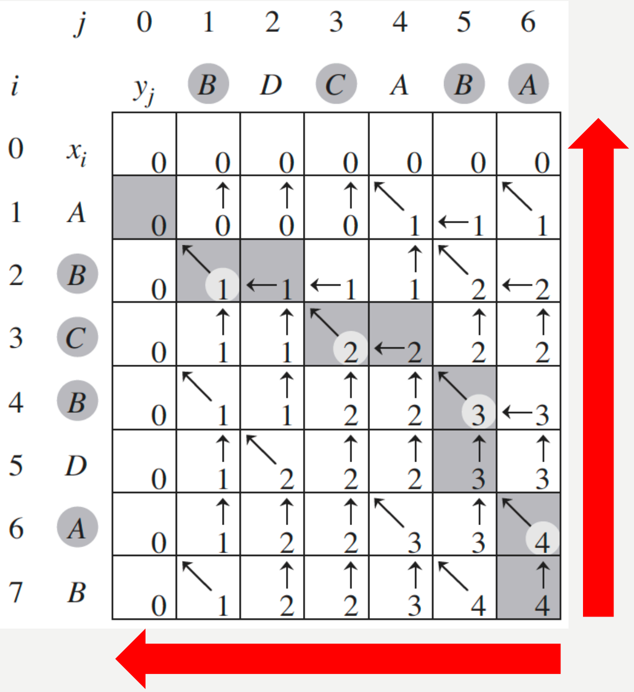
Stack

i \ j		0	1	2	3	4	5	6
	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

APPLYING DP STEP 4
THE RUNNING TIME

- **Step 4**: Construct the optimal solution from the computed information.

- Consider instance $X =< A, B, C, B, D, A, B >$ and $Y =< B, D, C, A, B, A >$
  - $m =$ _____, and $n =$ _____
  - The algorithm starts at the lower-right entry and ends at the top-left entry.
  - It visits _______ rows and _______ columns.



APPLYING DP STEP 4
THE RUNNING TIME

- **Step 4**: Construct the optimal solution from the computed information.

- Consider $X =< x_1, x_2, \dots, x_m >$ and $Y = < y_1, y_2, \dots, y_n >$
- The running time of the PRINT-LCS algorithm is $T(n) =$ (______).

i \ j		0	1	2	3	4
		y_j				
0	x_i	0	0	0	0	0
1		0	↖ 1	← 1	← 1	← 1
2		0	↑ 1	↖ 2	← 2	← 2
3		0	↑ 1	↑ 2	↖ 3	← 3
4		0	↑ 1	↑ 2	↑ 3	↖ 4
5		0	↑ 1	↑ 2	↑ 3	↑ 4

i \ j		0	1	2	3	4
		y_j				
0	x_i	0	0	0	0	0
1		0	↑ 1	← 1	← 1	← 1
2		0	↑ 1	↑ 1	↑ 1	↑ 1
3		0	↑ 1	↑ 1	↑ 1	↑ 1
4		0	↑ 1	↑ 1	↑ 1	↑ 1
5		0	↑ 1	↑ 1	↑ 1	↑ 1

PROJECT 01

- Quicksort on IDENTICAL, SORTED, REVERSE-SORTED datasets
  - Stack size limit or recursion limit problem with **python** and **Java** implementations

- Report the issue in your project report
  - Do some research on the cause of problem
    - Reference the sites, textbook pages your analysis is based off.
  - Show the limit of your program, i.e., maximum data sizes that can be quick-sorted.

- No homework due tonight.

NEXT UP
GREEDY STRATEGY

REFERENCE

- Screenshots are taken from the textbook.