Algorithms and Data Structures

Matrix Chain Multiplication (cont), Longest Common Subsequence

Dynamic Programming

Dynamic Programming

The paradigm of dynamic programming:

Given a problem P, define a sequence of subproblems, with the following properties:

Dynamic Programming

The paradigm of dynamic programming:

Given a problem P, define a sequence of subproblems, with the following properties:

The subproblems are ordered from the smallest to the largest.

Dynamic Programming

The paradigm of dynamic programming:

Given a problem P, define a sequence of subproblems, with the following properties:

The subproblems are ordered from the smallest to the largest.

The largest problem is our original problem P.

Dynamic Programming

The paradigm of dynamic programming:

Given a problem P, define a sequence of subproblems, with the following properties:

The subproblems are ordered from the smallest to the largest.

The largest problem is our original problem P.

The optimal solution of a subproblem can be constructed from the optimal solutions of sub-sub-problems. (*Optimal Substructure*).

Dynamic Programming

The paradigm of dynamic programming:

Given a problem P, define a sequence of subproblems, with the following properties:

The subproblems are ordered from the smallest to the largest.

The largest problem is our original problem P.

The optimal solution of a subproblem can be constructed from the optimal solutions of sub-sub-problems. (*Optimal Substructure*).

Solve the subproblems from the smallest to the largest. When you solve a subproblem, store the solution (e.g., in an array) and use it to solve the larger subproblems.

Matrix Chain Multiplication

Matrix Chain Multiplication

We have a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices (not necessarily square) to be multiplied.

Matrix Chain Multiplication

We have a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices (not necessarily square) to be multiplied.

The goal is to compute the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$.

Matrix Chain Multiplication

We have a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices (not necessarily square) to be multiplied.

The goal is to compute the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$.

For that we will use the *standard* algorithm for matrix multiplication:

```
RECTANGULAR-MATRIX-MULTIPLY(A, B, C, p, q, r)
1  for i = 1 to p
2      for j = 1 to r
3          for k = 1 to q
4              c_ij = c_ij + a_ik · b_kj
```

Matrix Chain Multiplication

We have a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices (not necessarily square) to be multiplied.

The goal is to compute the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$.

For that we will use the *standard* algorithm for matrix multiplication:

```
RECTANGULAR-MATRIX-MULTIPLY(A, B, C, p, q, r)
1  for i = 1 to p
2      for j = 1 to r
3          for k = 1 to q
4              c_ij = c_ij + a_ik · b_kj
```

The running time is the number of scalar multiplications, i.e., pqr.

Matrix Chain Multiplication

We have a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices (not necessarily square) to be multiplied.

The goal is to compute the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$.

Matrix Chain Multiplication

We have a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices (not necessarily square) to be multiplied.

The goal is to compute the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$.

The order of multiplication matters!

Matrix Chain Multiplication

We have a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices (not necessarily square) to be multiplied.

The goal is to compute the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$.

The order of multiplication matters!

Consider $\langle A_1, A_2, A_3 \rangle$ with dimensions 10×100, 100×5, and 5×50.

Matrix Chain Multiplication

We have a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices (not necessarily square) to be multiplied.

The goal is to compute the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$.

The order of multiplication matters!

Consider $\langle A_1, A_2, A_3 \rangle$ with dimensions 10×100, 100×5, and 5×50.

If we do $((A_1 \cdot A_2) \cdot A_3)$ we have $10 \cdot 100 \cdot 5 = 5000$ scalar multiplications + $10 \cdot 5 \cdot 50 = 2500$ scalar multiplications for a total of 7500.

Matrix Chain Multiplication

We have a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices (not necessarily square) to be multiplied.

The goal is to compute the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$.

The order of multiplication matters!

Consider $\langle A_1, A_2, A_3 \rangle$ with dimensions 10×100, 100×5, and 5×50.

If we do $((A_1 \cdot A_2) \cdot A_3)$ we have $10 \cdot 100 \cdot 5 = 5000$ scalar multiplications + $10 \cdot 5 \cdot 50 = 2500$ scalar multiplications for a total of 7500.

If instead we do $(A_1 \cdot (A_2 \cdot A_3))$ we have $100 \cdot 5 \cdot 50 = 25000$ scalar multiplications + $10 \cdot 100 \cdot 50 = 50000$ scalar multiplications for a total of 75000.

Full Parenthesisation

A product of matrices is *fully parenthesised* if it is either a single matrix, or a product of two fully parenthesised matrix products, surrounded by parenthesis.

Full Parenthesisation

A product of matrices is *fully parenthesised* if it is either a single matrix, or a product of two fully parenthesised matrix products, surrounded by parenthesis.

Matrix Chain Multiplication problem: Given a chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices where for $i = 1,2,\ldots,n$, matrix A_i has dimension $p_{i-1} \times p_i$, find a full parenthesisation of the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$ that minimises the number of scalar multiplications.

Full Parenthesisation

A product of matrices is *fully parenthesised* if it is either a single matrix, or a product of two fully parenthesised matrix products, surrounded by parenthesis.

Matrix Chain Multiplication problem: Given a chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices where for $i = 1,2,\ldots,n$, matrix A_i has dimension $p_{i-1} \times p_i$, find a full parenthesisation of the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$ that minimises the number of scalar multiplications.

We can think of the input as a sequence of dimensions $\langle p_0, p_1, p_2, \ldots, p_n \rangle$.

A Dynamic Programming algorithm (bottom-up)

```
MATRIX-CHAIN-ORDER(p, n)
 1  let m[1:n, 1:n] and s[1:n − 1, 2:n] be new tables
 2  for i = 1 to n                          // chain length 1
 3      m[i, i] = 0
 4  for l = 2 to n                          // l is the chain length
 5      for i = 1 to n − l + 1              // chain begins at A_i
 6          j = i + l − 1                   // chain ends at A_j
 7          m[i, j] = ∞
 8          for k = i to j − 1              // try A_{i:k} A_{k+1:j}
 9              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
10              if q < m[i, j]
11                  m[i, j] = q             // remember this cost
12                  s[i, j] = k             // remember this index
13  return m and s
```

Computing a solution

We have the following relation:

$$M[i,j] = \begin{cases} 0, & \text{if } i = j, \\ \min_{k \in [i,j)} \{M[i,k] + M[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j\,. \end{cases}$$

We could now define a recursive algorithm straightforwardly using this.

```
RECURSIVE-MATRIX-CHAIN(p, i, j)
1  if i == j
2      return 0
3  m[i, j] = ∞
4  for k = i to j − 1
5      q = RECURSIVE-MATRIX-CHAIN(p, i, k)
           + RECURSIVE-MATRIX-CHAIN(p, k + 1, j)
           + p_{i−1} p_k p_j
6      if q < m[i, j]
7          m[i, j] = q
8  return m[i, j]
```

Memoization

We write the procedure recursively in a natural manner, but we modify it to save the result of each subproblem in a table.

The procedure first checks if we have computed the solution already in our table.

If we have, it uses it without calling itself recursively.

If we have not, then it recurses to compute it.

A Dynamic Programming algorithm (top-down)

```
MEMOIZED-MATRIX-CHAIN(p, n)
1  let m[1:n, 1:n] be a new table
2  for i = 1 to n
3       for j = i to n
4            m[i, j] = ∞
5  return LOOKUP-CHAIN(m, p, 1, n)
```

```
LOOKUP-CHAIN(m, p, i, j)
1  if m[i, j] < ∞
2       return m[i, j]
3  if i == j
4       m[i, j] = 0
5  else for k = i to j − 1
6            q = LOOKUP-CHAIN(m, p, i, k)
                 + LOOKUP-CHAIN(m, p, k + 1, j) + p_{i−1} p_k p_j
7            if q < m[i, j]
8                 m[i, j] = q
9  return m[i, j]
```

Longest Common Subsequence

Longest Common Subsequence

Motivation: A strand of DNA consists of a string of bases, with possible values Adenine, Cytosine, Guanine, and Thymine.

We can express a strand of DNA as a string, e.g.,

$S_1 =$ ACCGGTCGAGTGCGCGGAAGCCGGCCAA

$S_2 =$ GTCGTTCGGAATGCCGTTGCTCTGTAAA

We would like to compare two strands to see how similar they are (in order to see how similar two organisms are).

Various way to do that. Here: Find the longest possible strand S_3 which is a subsequence of both S_1 and S_2.

Longest Common Subsequence

Longest Common Subsequence

Given two sequences X and Y, find a common subsequence Z that is as long as possible.

Longest Common Subsequence

Given two sequences X and Y, find a common subsequence Z that is as long as possible.

$Z = \langle z_1, z_2, \ldots, z_k \rangle$ is a *subsequence* of $X = \langle x_1, x_2, \cdots, x_m \rangle$ if there exists a strictly increasing sequence $\langle i_1, i_2, \ldots, i_k \rangle$ of indices of X such that for all $j = 1,2,\ldots, k$ we have $x_{i_j} = z_j$.

Longest Common Subsequence

Given two sequences X and Y, find a common subsequence Z that is as long as possible.

$Z = \langle z_1, z_2, \ldots, z_k \rangle$ is a *subsequence* of $X = \langle x_1, x_2, \cdots, x_m \rangle$ if there exists a strictly increasing sequence $\langle i_1, i_2, \ldots, i_k \rangle$ of indices of X such that for all $j = 1,2,\ldots, k$ we have $x_{i_j} = z_j$.

Example: $Z = \langle B, C, D, B \rangle$ is a subsequence of $X = \langle A,\ B, C, B, D, A, B \rangle$.

Brute Force

Brute Force

We could enumerate all subsequences of X and check for each one of them if it is also a subsequence of Y.

Brute Force

We could enumerate all subsequences of X and check for each one of them if it is also a subsequence of Y.

How many subsequences does X have?

Brute Force

We could enumerate all subsequences of X and check for each one of them if it is also a subsequence of Y.

How many subsequences does X have?

Each subsequence is a subset of the indices $\{1,2,\ldots,m\}$.

Brute Force

We could enumerate all subsequences of X and check for each one of them if it is also a subsequence of Y.

How many subsequences does X have?

Each subsequence is a subset of the indices $\{1,2,\ldots,m\}$.

Therefore there are 2^m of them.

Brute Force

We could enumerate all subsequences of X and check for each one of them if it is also a subsequence of Y.

How many subsequences does X have?

Each subsequence is a subset of the indices $\{1,2,\ldots,m\}$.

Therefore there are 2^m of them.

Exponential time.

Optimal Substructure

Optimal Substructure

Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith *prefix* of X (for $i = 1,2,\ldots, m$) is defined as $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Optimal Substructure

Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith *prefix* of X (for $i = 1,2,\ldots,m$) is defined as $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Example: $X = \langle A, B, C, B, D, A, B \rangle$, $X_4 = \langle A, B, C, B \rangle$ and $X_0 = \langle \rangle$.

Optimal Substructure

Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$, the ith *prefix* of X (for $i = 1,2,\ldots, m$) is defined as $X_i = \langle x_1, x_2, \ldots, x_i \rangle$.

Example: $X = \langle A, B, C, B, D, A, B \rangle$, $X_4 = \langle A, B, C, B \rangle$ and $X_0 = \langle \rangle$.

Theorem (Optimal Substructure):
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be two sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y. Then.

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}.
2. If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is an LCS of X_{m-1} and Y.
3. 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 (Optimal Substructure):
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be two sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y. Then.

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}.
2. If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is an LCS of X_{m-1} and Y.
3. 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 (Optimal Substructure):
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be two sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y. Then.

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}.
2. If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$ and $z_k \neq y_n$, then Z is an LCS of X and Y_{n-1}.

1. $X = \langle x_1, x_2, \ldots, x_{m-1}, A \rangle$, $Y = \langle y_1, y_2, \ldots, y_{n-1}, A \rangle \Rightarrow Z = \langle z_1, z_2, \ldots, z_{k-1}, A \rangle$.

Optimal Substructure

Theorem (Optimal Substructure):
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be two sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y. Then.

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}.
2. If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$ and $z_k \neq y_n$, then Z is an LCS of X and Y_{n-1}.

1. $X = \langle x_1, x_2, \ldots, x_{m-1}, A \rangle$, $Y = \langle y_1, y_2, \ldots, y_{n-1}, A \rangle \Rightarrow Z = \langle z_1, z_2, \ldots, z_{k-1}, A \rangle$.

2. $X = \langle x_1, x_2, \ldots, x_{m-1}, A \rangle$, $Y = \langle y_1, y_2, \ldots, y_{n-1}, B \rangle$,
$Z = \langle z_1, z_2, \ldots, z_{k-1}, \bar{A} \rangle$

Optimal Substructure

Theorem (Optimal Substructure):
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be two sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y. Then.

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}.
2. If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$ and $z_k \neq y_n$, then Z is an LCS of X and Y_{n-1}.

1. $X = \langle x_1, x_2, \ldots, x_{m-1}, A \rangle$, $Y = \langle y_1, y_2, \ldots, y_{n-1}, A \rangle \Rightarrow Z = \langle z_1, z_2, \ldots, z_{k-1}, A \rangle$.

2. $X = \langle x_1, x_2, \ldots, x_{m-1}, A \rangle$, $Y = \langle y_1, y_2, \ldots, y_{n-1}, B \rangle$,
$Z = \langle z_1, z_2, \ldots, z_{k-1}, \bar{A} \rangle$

3. $X = \langle x_1, x_2, \ldots, x_{m-1}, A \rangle$, $Y = \langle y_1, y_2, \ldots, y_{n-1}, B \rangle$,
$Z = \langle z_1, z_2, \ldots, z_{k-1}, \bar{B} \rangle$

Optimal Substructure

Optimal Substructure

Proof:

(1) Assume by contradiction that $z_k \neq x_m = y_n$.

Optimal Substructure

Proof:

(1) Assume by contradiction that $z_k \neq x_m = y_n$.

We could add $x_m = y_n$ to Z to obtain a common subsequence of length $k + 1$, a contradiction.

Optimal Substructure

Proof:

(1) Assume by contradiction that $z_k \neq x_m = y_n$.

We could add $x_m = y_n$ to Z to obtain a common subsequence of length $k + 1$, a contradiction.

Consider the prefix Z_{k-1}: it is a common subsequence of length $k - 1$, and we would like to show that it is a LCS of X_{m-1} and Y_{n-1}.

Optimal Substructure

Proof:

(1) Assume by contradiction that $z_k \neq x_m = y_n$.

We could add $x_m = y_n$ to Z to obtain a common subsequence of length $k + 1$, a contradiction.

Consider the prefix Z_{k-1}: it is a common subsequence of length $k - 1$, and we would like to show that it is a LCS of X_{m-1} and Y_{n-1}.

Assume by contradiction that there exists a longer common subsequence W of X_{m-1} and Y_{n-1} that has length at least k.

Optimal Substructure

Proof:

(1) Assume by contradiction that $z_k \neq x_m = y_n$.

We could add $x_m = y_n$ to Z to obtain a common subsequence of length $k + 1$, a contradiction.

Consider the prefix Z_{k-1}: it is a common subsequence of length $k - 1$, and we would like to show that it is a LCS of X_{m-1} and Y_{n-1}.

Assume by contradiction that there exists a longer common subsequence W of X_{m-1} and Y_{n-1} that has length at least k.

If we append $x_m = y_n$ to W we obtain a common subsequence of X and Y of length $k + 1$ contradicting the fact that Z is a LCS.

Optimal Substructure

Theorem (Optimal Substructure):
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be two sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y. Then.

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}.
2. If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is an LCS of X_{m-1} and Y.
3. 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 (Optimal Substructure):
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be two sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y. Then.

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}.
2. If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is an LCS of X_{m-1} and Y.
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:
(2) Since $z_k \neq x_m$, Z is a common subsequence of X_{m-1} and Y. Assume by contradiction that it is no a LCS.

Optimal Substructure

Theorem (Optimal Substructure):
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be two sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y. Then.

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}.
2. If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is an LCS of X_{m-1} and Y.
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:
(2) Since $z_k \neq x_m$, Z is a common subsequence of X_{m-1} and Y. Assume by contradiction that it is no a LCS.

Let W be a longer common subsequence, of length at least $k + 1$.

Optimal Substructure

Theorem (Optimal Substructure):
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be two sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y. Then.

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}.
2. If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is an LCS of X_{m-1} and Y.
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:
(2) Since $z_k \neq x_m$, Z is a common subsequence of X_{m-1} and Y. Assume by contradiction that it is no a LCS.

Let W be a longer common subsequence, of length at least $k + 1$.

But W is a common subsequence of X and Y, contradicting the fact that Z is a LCS.

Constructing a recursion

Constructing a recursion

By the theorem, we have:

Constructing a recursion

By the theorem, we have:

- If $x_m = y_n$, then we need to find a LCS of X_{m-1} and Y_{n-1}.

Constructing a recursion

By the theorem, we have:

- If $x_m = y_n$, then we need to find a LCS of X_{m-1} and Y_{n-1}.
- Otherwise, we need to solve two subproblems:

Constructing a recursion

By the theorem, we have:

- If $x_m = y_n$, then we need to find a LCS of X_{m-1} and Y_{n-1}.
- Otherwise, we need to solve two subproblems:
  - Find a LCS of X_{m-1} and Y.

Constructing a recursion

By the theorem, we have:

- If $x_m = y_n$, then we need to find a LCS of X_{m-1} and Y_{n-1}.
- Otherwise, we need to solve two subproblems:
  - Find a LCS of X_{m-1} and Y.
  - Find a LCS of X and Y_{n-1}.

Constructing a recursion

By the theorem, we have:

- If $x_m = y_n$, then we need to find a LCS of X_{m-1} and Y_{n-1}.
- Otherwise, we need to solve two subproblems:
  - Find a LCS of X_{m-1} and Y.
  - Find a LCS of X and Y_{n-1}.
  - Choose the one that is longer.

Constructing a recursion

Constructing a recursion

Let $C[i, j]$ be the length of a LCS of X_i and Y_j.

Constructing a recursion

Let $C[i, j]$ be the length of a LCS of X_i and Y_j.

If $i = 0$ or $j = 0$, the LCS has length 0.

Constructing a recursion

Let $C[i, j]$ be the length of a LCS of X_i and Y_j.

If $i = 0$ or $j = 0$, the LCS has length 0.

$C[i,0] = C[0,j]$ for all i, j.

Constructing a recursion

Let $C[i, j]$ be the length of a LCS of X_i and Y_j.

If $i = 0$ or $j = 0$, the LCS has length 0.

$C[i,0] = C[0,j]$ for all i, j.

Otherwise we consider the two cases of the previous slide:

Constructing a recursion

Let $C[i,j]$ be the length of a LCS of X_i and Y_j.

If $i = 0$ or $j = 0$, the LCS has length 0.

$C[i,0] = C[0,j]$ for all i, j.

Otherwise we consider the two cases of the previous slide:

- If $x_i = x_j$, we have $C[i,j] = C[i-1,j-1] + 1$

Constructing a recursion

Let $C[i, j]$ be the length of a LCS of X_i and Y_j.

If $i = 0$ or $j = 0$, the LCS has length 0.

$C[i,0] = C[0,j]$ for all i, j.

Otherwise we consider the two cases of the previous slide:

- If $x_i = x_j$, we have $C[i, j] = C[i - 1,j - 1] + 1$
- If $x_i \neq x_j$, we have $C[i, j] = \max\{C[i, j - 1], C[i - 1,j]\}$

Constructing a recursion

Let $C[i, j]$ be the length of a LCS of X_i and Y_j.

If $i = 0$ or $j = 0$, the LCS has length 0.

$C[i,0] = C[0,j]$ for all i, j.

Otherwise we consider the two cases of the previous slide:

- If $x_i = x_j$, we have $C[i, j] = C[i - 1,j - 1] + 1$
- If $x_i \neq x_j$, we have $C[i, j] = \max\{C[i, j - 1], C[i - 1,j]\}$

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

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```


A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0		

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0			
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0			
0			
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0	★		
0			
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0	★	★	
0			
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0	★	★	★
0			
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0	★	★	★
0	★		
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0	★	★	★
0	★	★	
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0	★	★	★
0	★	★	★
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0	★	★	★
0	★	★	★
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0	★	★	★
0	★	★	★
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0	★	★	★
0	★	★	★
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0	★	★	★
0	★	★	★
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0 ←	★	★	★
0	★	★	★
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```

0	0	0	0
0	★	★	★
0	★	★	★
0			
0			

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0						
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                Is x_1 = y_1?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0						
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                Is x_1 = y_1?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0						
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                Is x_1 = y_1?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0						
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$

$C[1,0] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                Is x_1 = y_1?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0						
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$

$C[1,0] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                Is x_1 = y_1?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0					
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$

$C[1,0] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0					
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$

$C[1,0] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0					
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$

$C[1,0] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                Is x_1 = y_2?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0					
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$

$C[1,0] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                Is x_1 = y_2?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0					
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                Is x_1 = y_2?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0					
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                 Is x_1 = y_2?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0					
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                 Is x_1 = y_2?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0				
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0				
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	*D*	*C*	*A*	*B*	*A*
	0	0	0	0	0	0	0
A	0	0	0				
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0			
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                Is x_1 = y_4?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	*D*	*C*	*A*	*B*	*A*
	0	0	0	0	0	0	0
A	0	0	0	0			
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                    Is x_1 = y_4?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0			
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                    Is x_1 = y_4?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0	1		
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0	1		
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0	1		
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	*D*	*C*	*A*	*B*	*A*
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0						
C	0						
B	0						
D	0						
A	0						
B	0						

$C[0,1] = 0$ $C[0,2] = 0$

$C[1,0] = 0$ $C[1,1] = 0$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0	1	1	1	1	2	2
C	0	1	1				
B	0						
D	0						
A	0						
B	0						

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	*D*	*C*	*A*	*B*	*A*
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0	1	1	1	1	2	2
C	0	1	1				
B	0						
D	0						
A	0						
B	0						

Next to compute: $C[3,3]$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                Is x_3 = y_3?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	*D*	*C*	*A*	*B*	*A*
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0	1	1	1	1	2	2
C	0	1	1				
B	0						
D	0						
A	0						
B	0						

Next to compute: $C[3,3]$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

Is $x_3 = y_3$?

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0	1	1	1	1	2	2
C	0	1	1				
B	0						
D	0						
A	0						
B	0						

Next to compute: $C[3,3]$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                    Is x_3 = y_3?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0	1	1	1	1	2	2
C	0	1	1				
B	0						
D	0						
A	0						
B	0						

Next to compute: $C[3,3]$

Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j                    Is x_3 = y_3?
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0	1	1	1	1	2	2
C	0	1	1	2			
B	0						
D	0						
A	0						
B	0						

Next to compute: $C[3,3]$

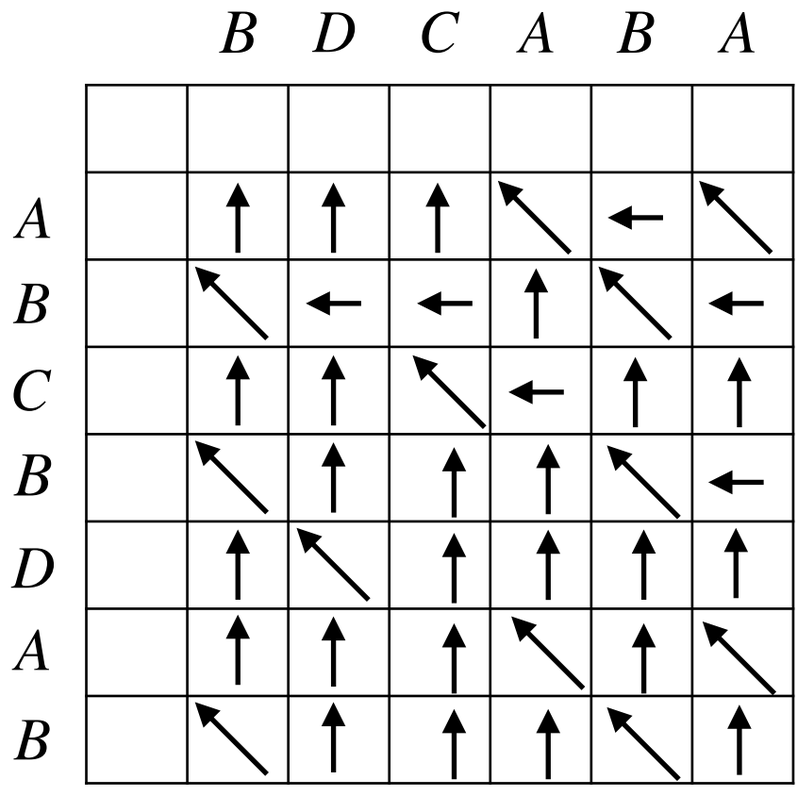
Example

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

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

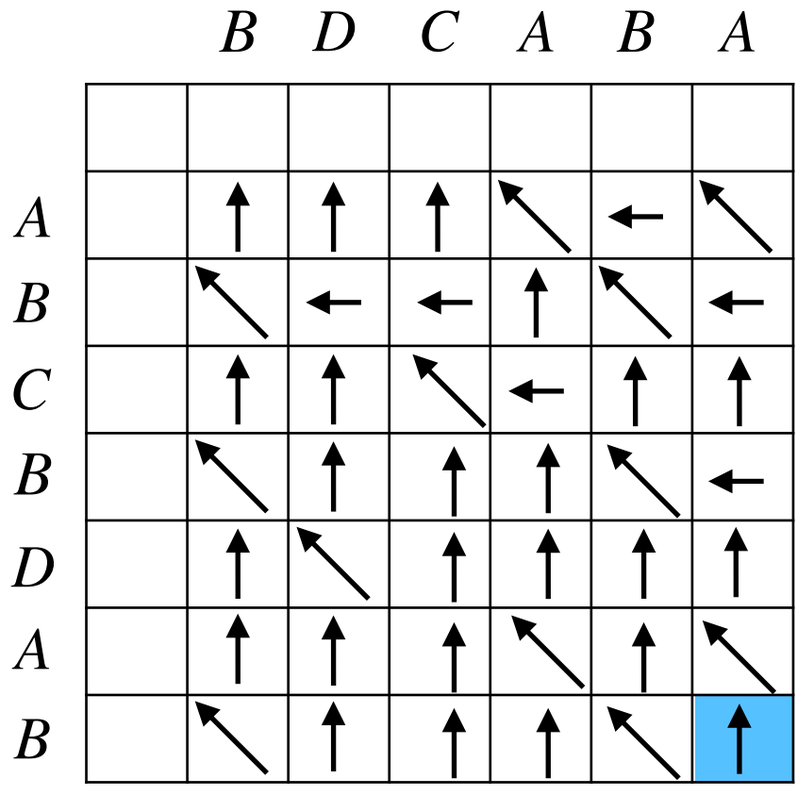
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



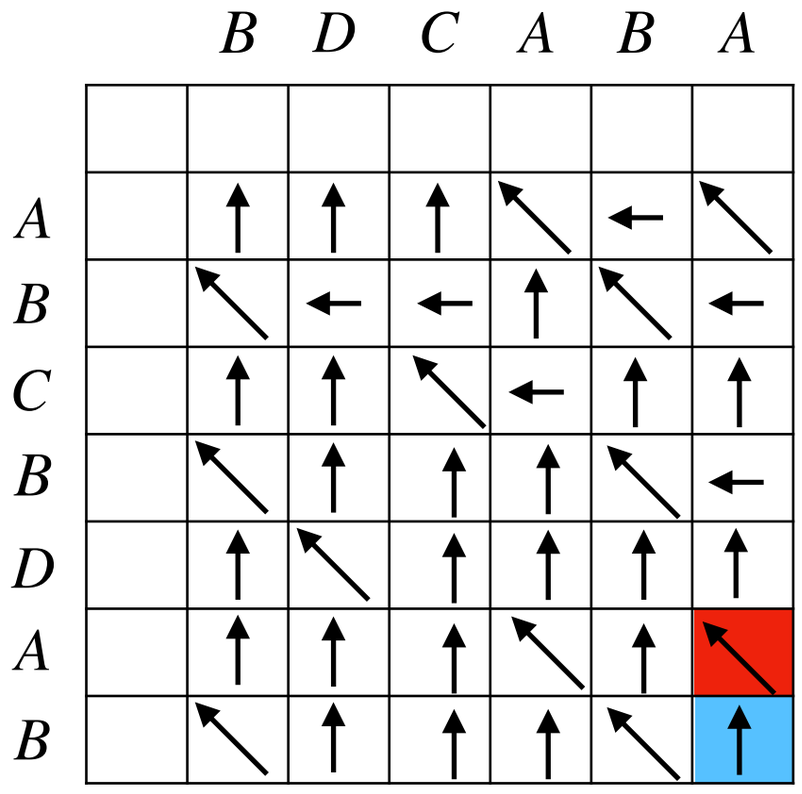
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



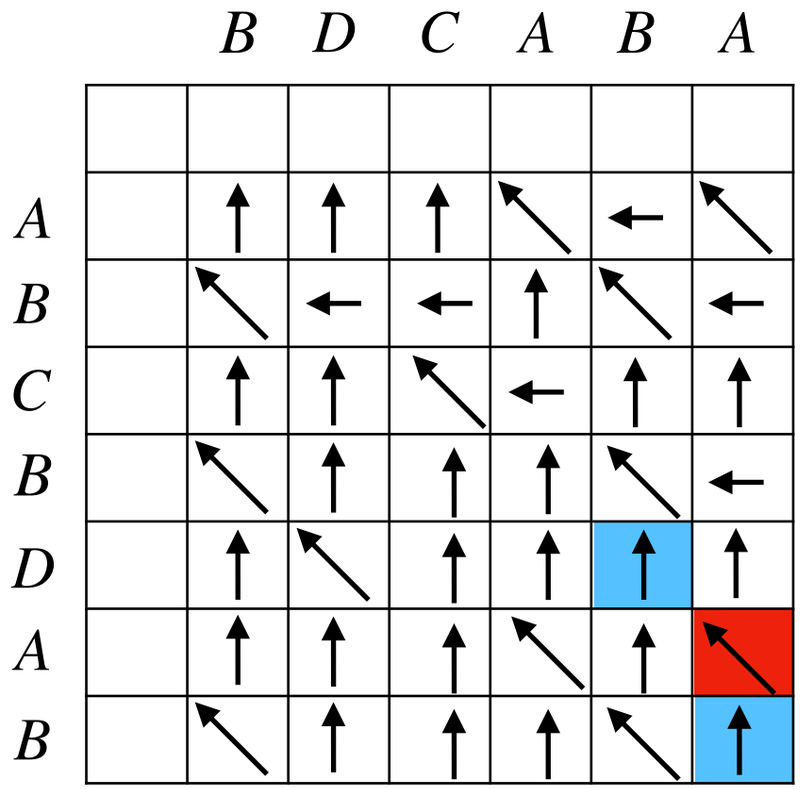
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



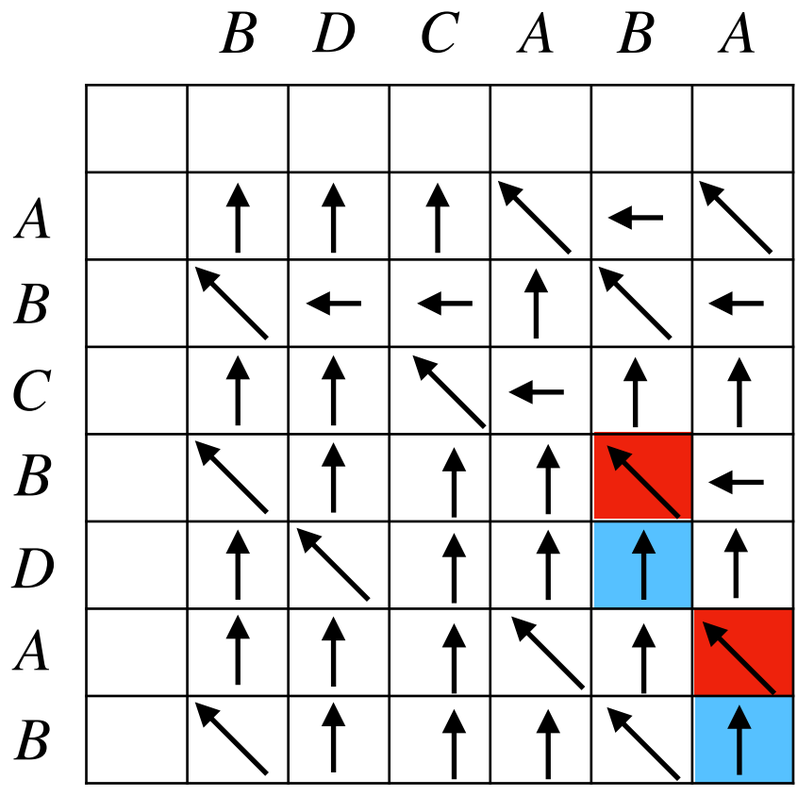
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



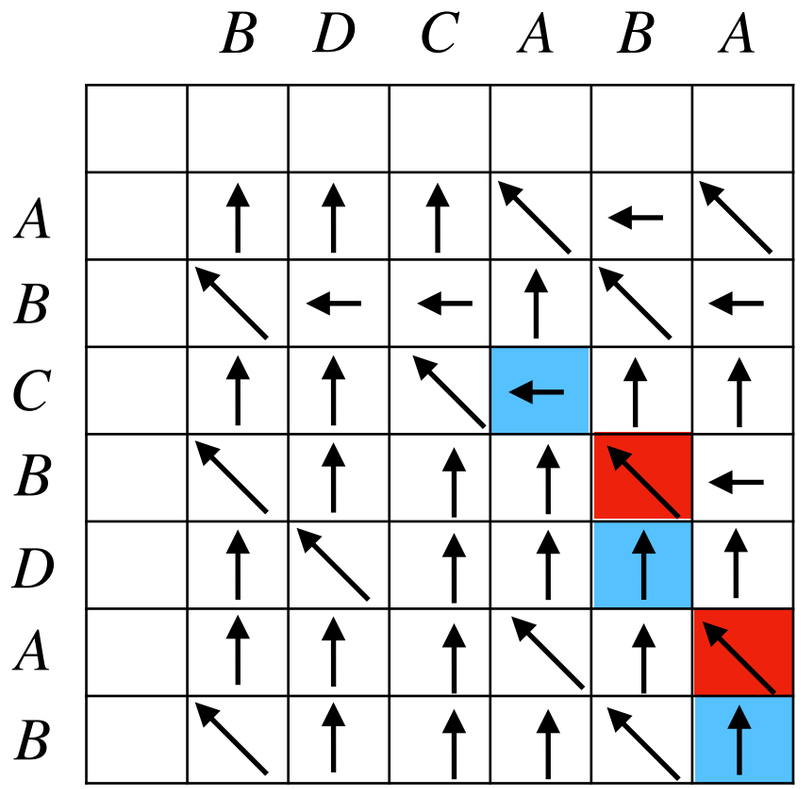
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



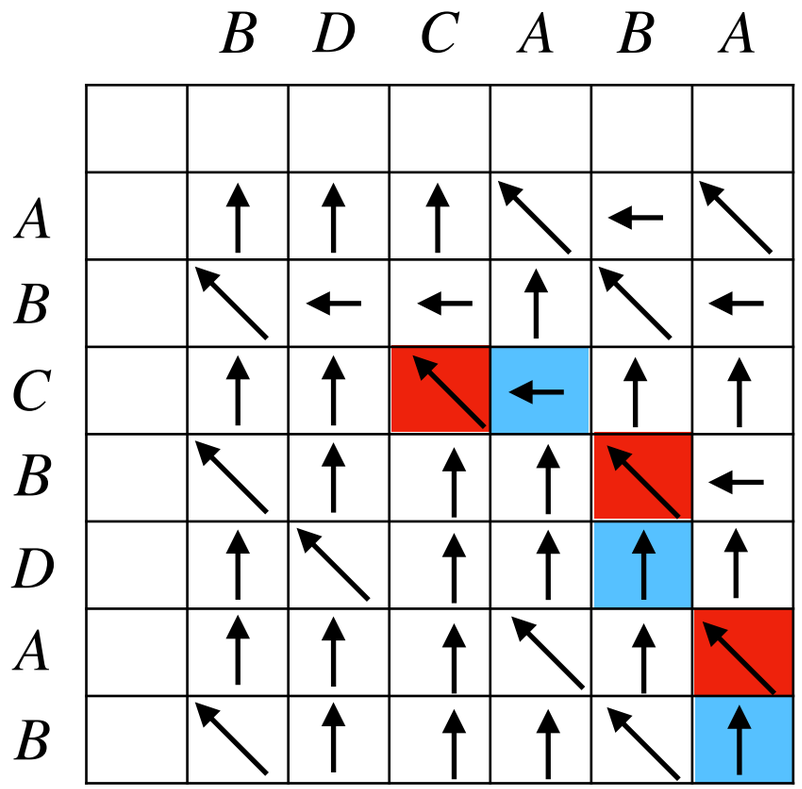
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



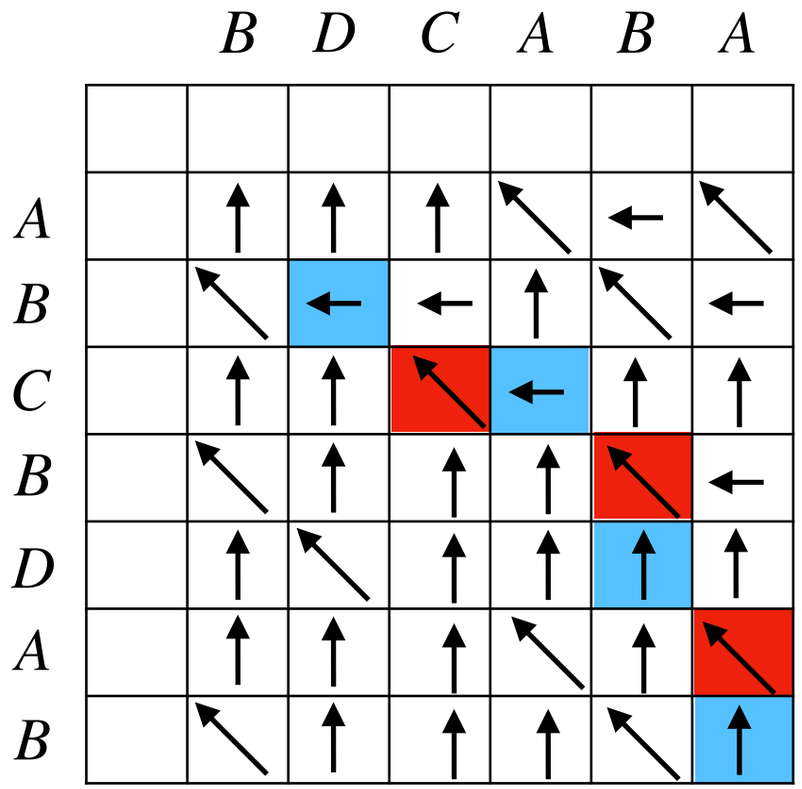
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



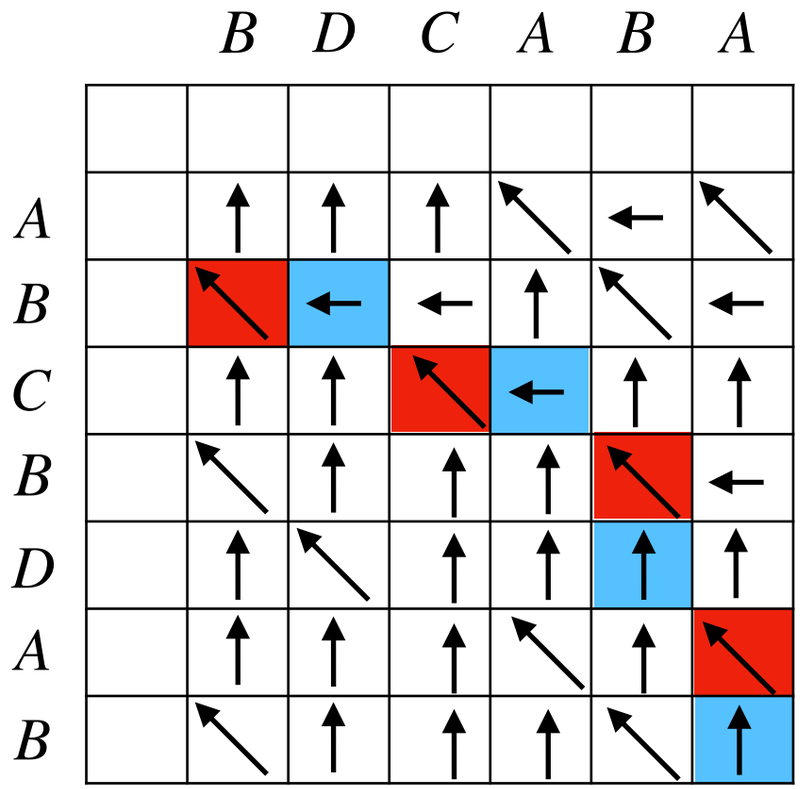
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



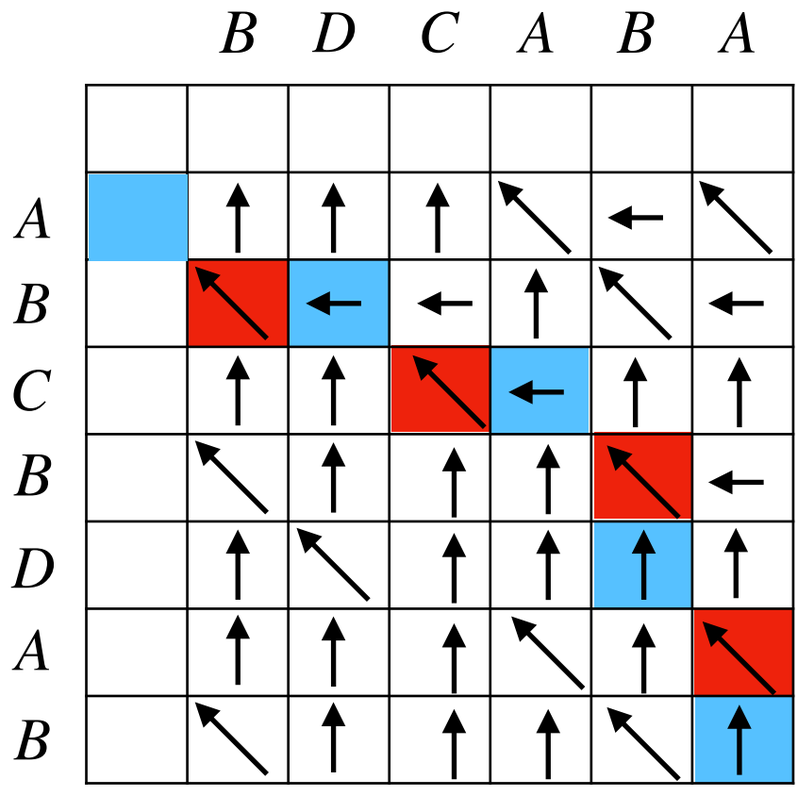
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return                  // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i               // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



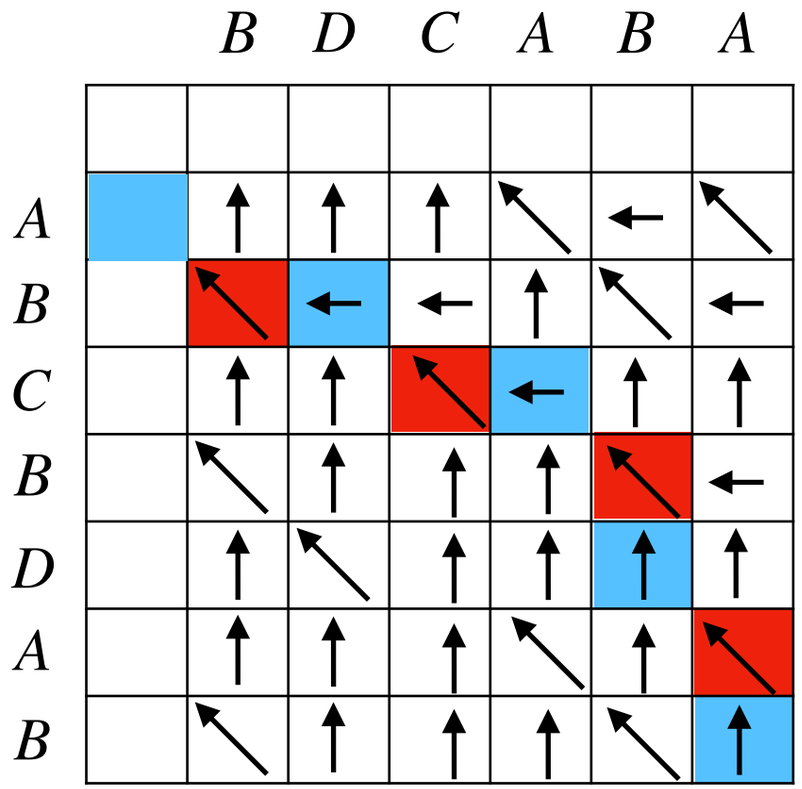
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



Example

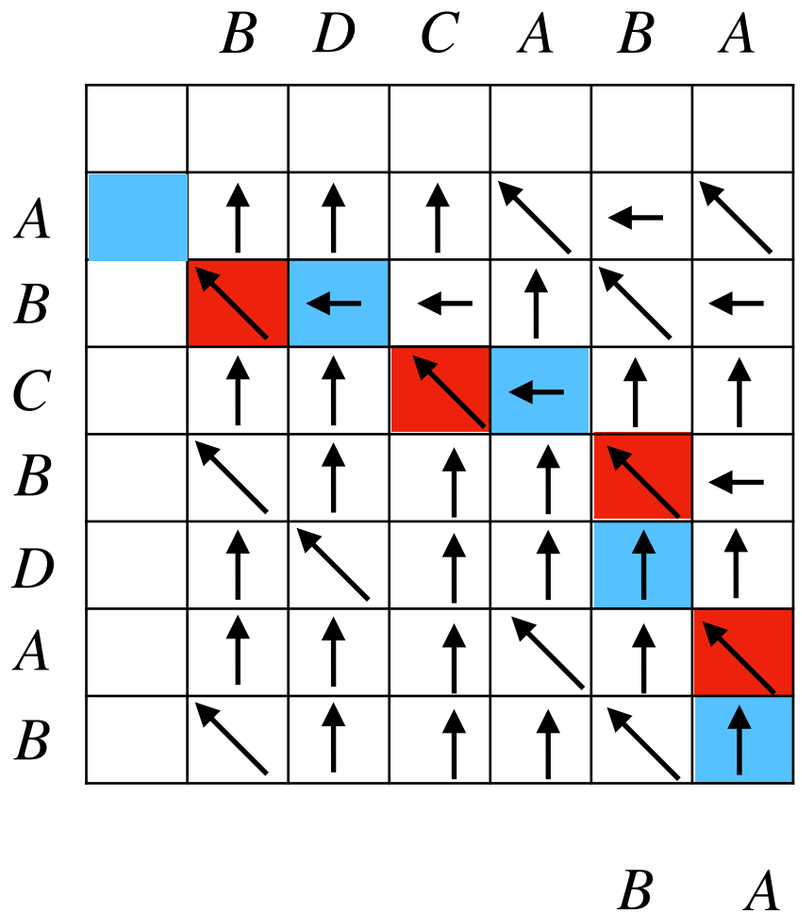
```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



A

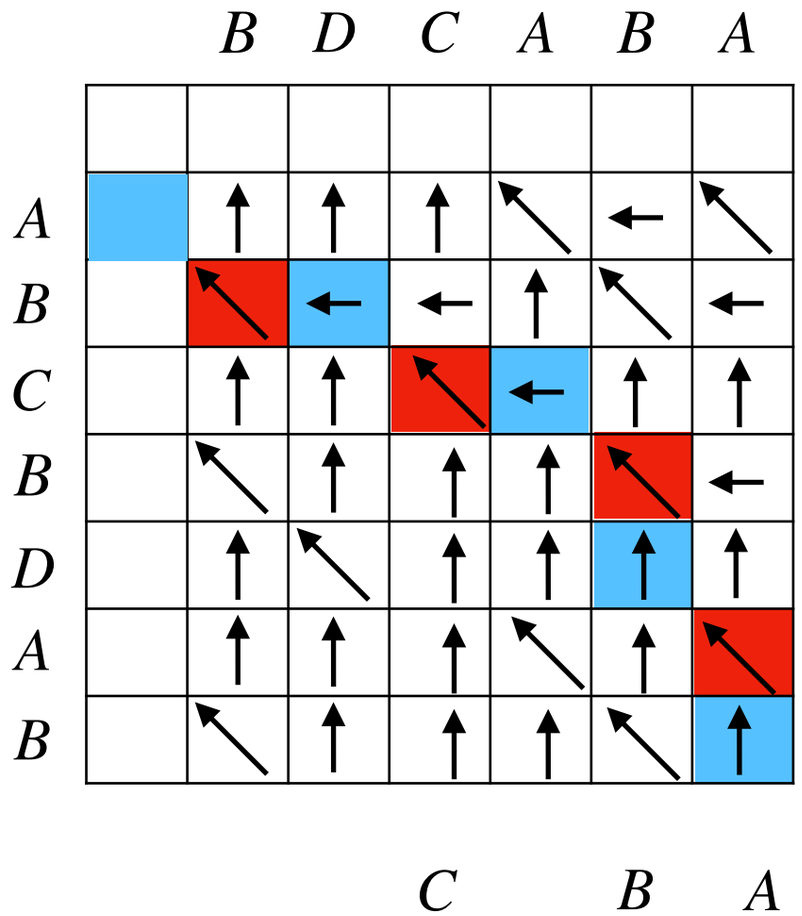
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



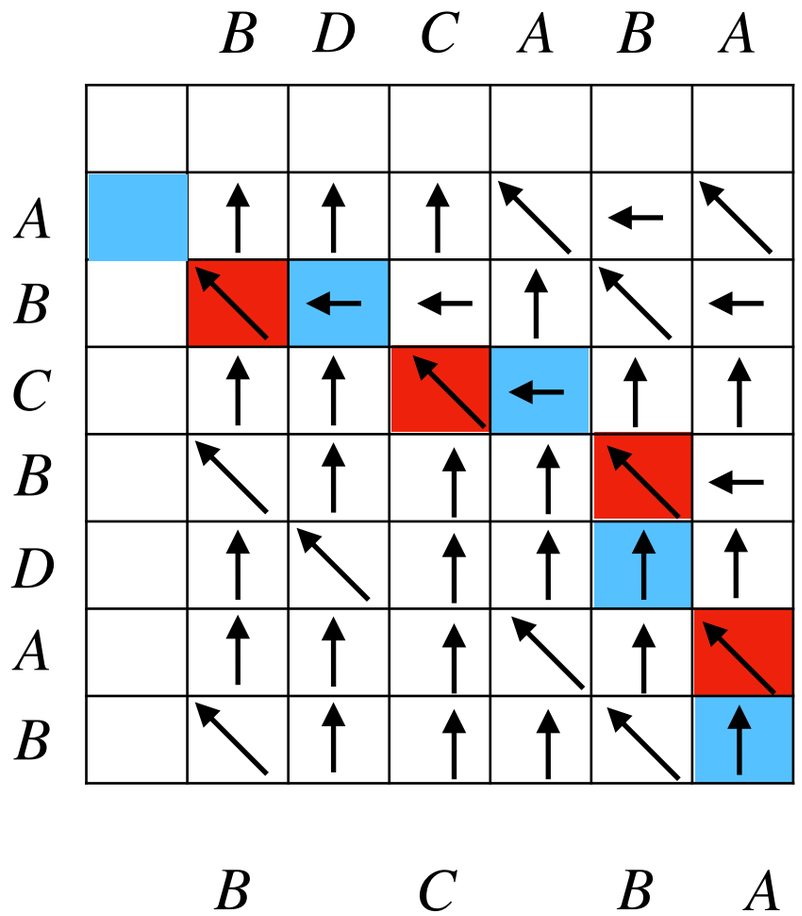
Example

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



Example

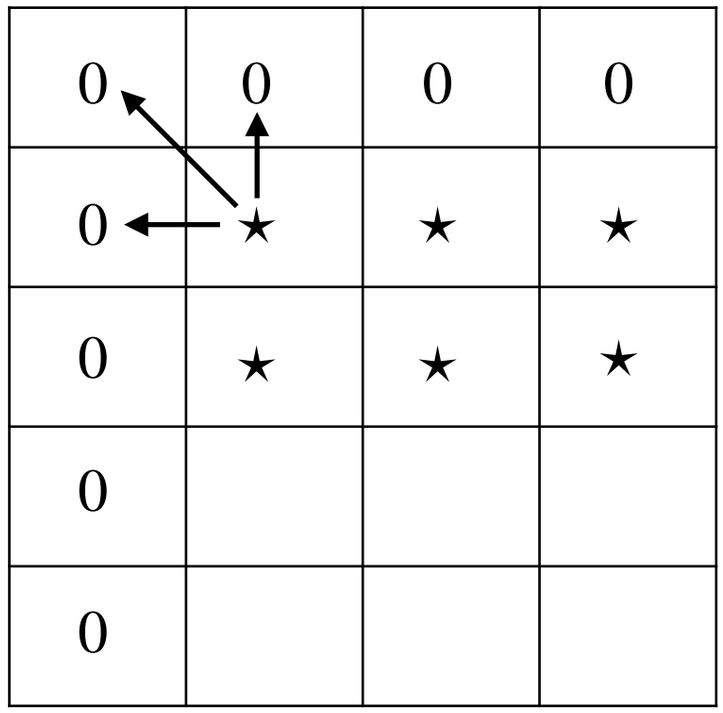
```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b

PRINT-LCS(b, X, i, j)
 1  if i == 0 or j == 0
 2      return              // the LCS has length 0
 3  if b[i, j] == "↖"
 4      PRINT-LCS(b, X, i − 1, j − 1)
 5      print x_i           // same as y_j
 6  elseif b[i, j] == "↑"
 7      PRINT-LCS(b, X, i − 1, j)
 8  else PRINT-LCS(b, X, i, j − 1)
```



A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2      return              // the LCS has length 0
3  if b[i, j] == "↖"
4      PRINT-LCS(b, X, i − 1, j − 1)
5      print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7      PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```

do we need this?

0	0	0	0
0	★	★	★
0	★	★	★
0			
0			

A Dynamic Programming Algorithm

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1:m, 1:n] and c[0:m, 0:n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m          // compute table entries in row-major order
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

can we use less space here?

0	0	0	0
0	★	★	★
0	★	★	★
0			
0			

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2      return              // the LCS has length 0
3  if b[i, j] == "↖"
4      PRINT-LCS(b, X, i − 1, j − 1)
5      print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7      PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```

do we need this?