Algorithms and Data Structures

Longest Common Subsequence, Subset Sum

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.

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: $C[3,3]$

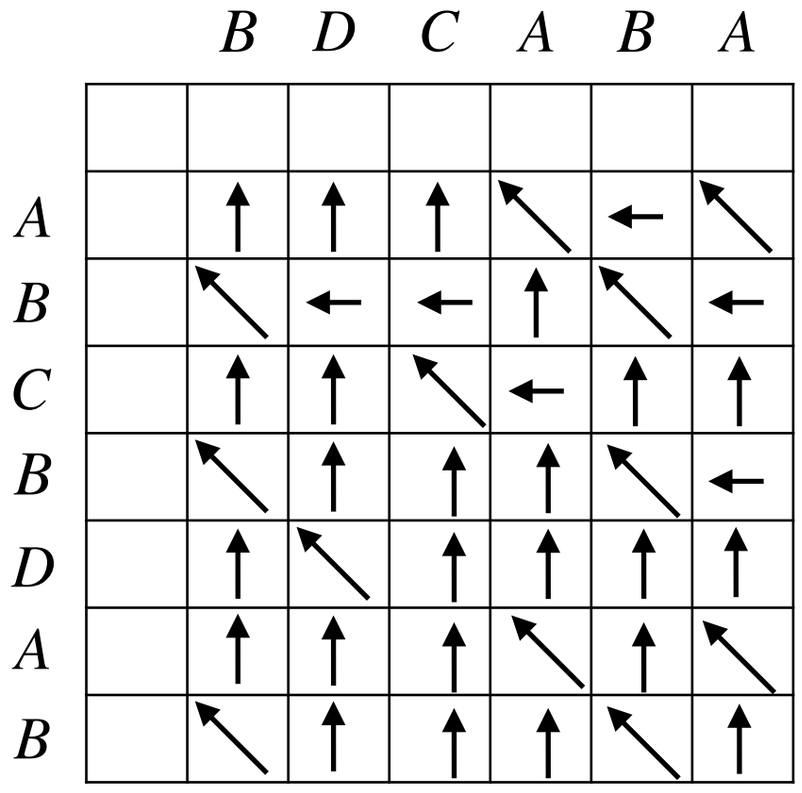
Example

```
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

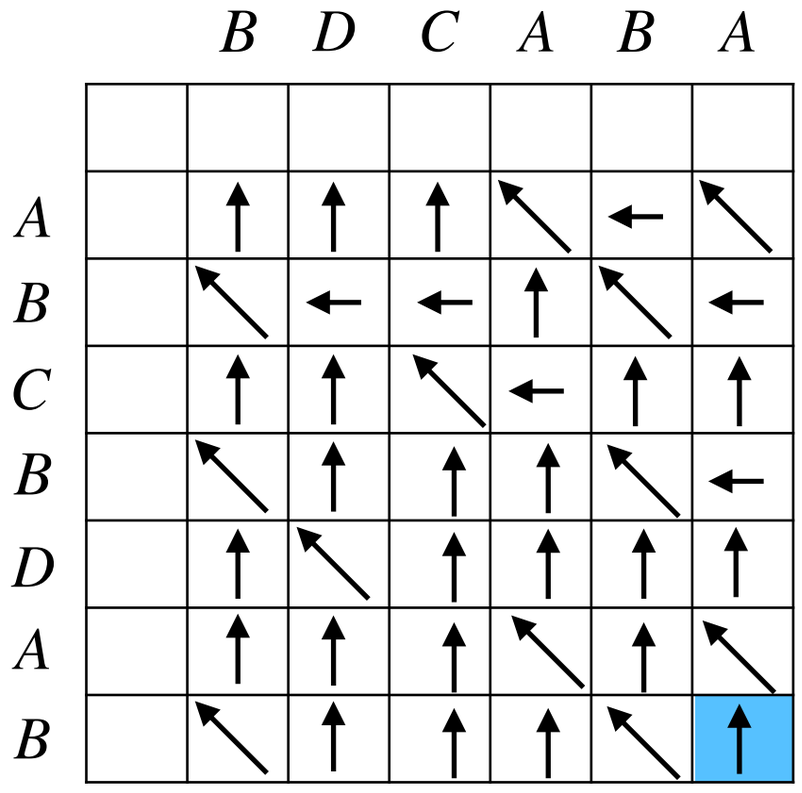
Example

```
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)
```



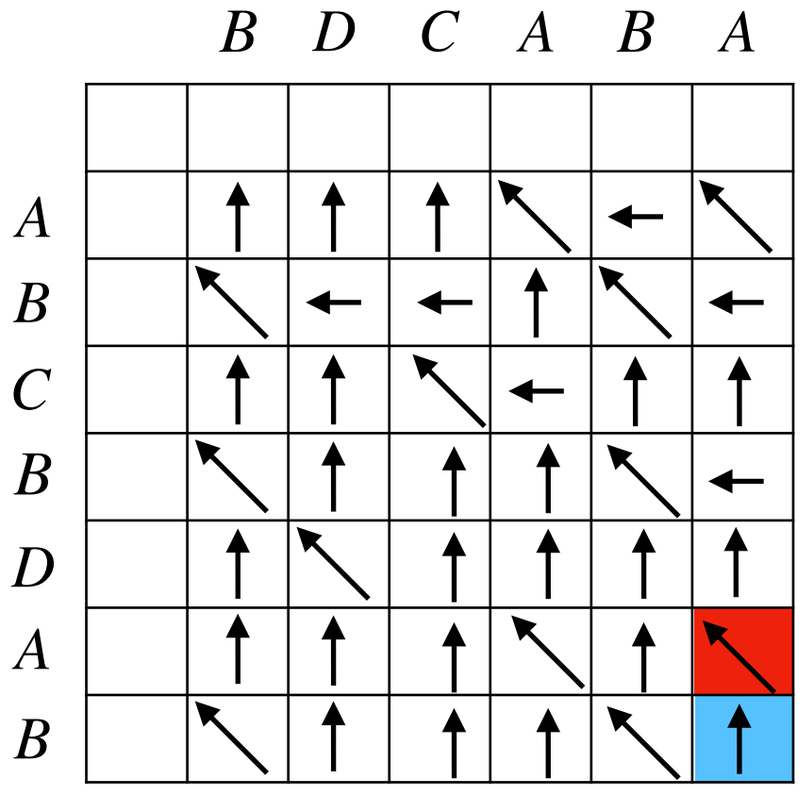
Example

```
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)
```



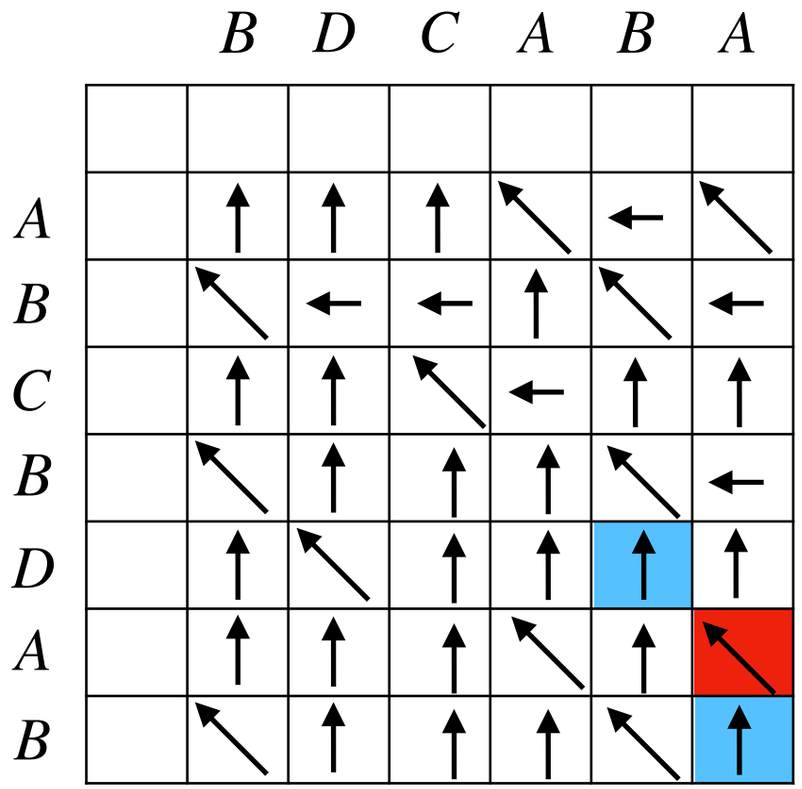
Example

```
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)
```



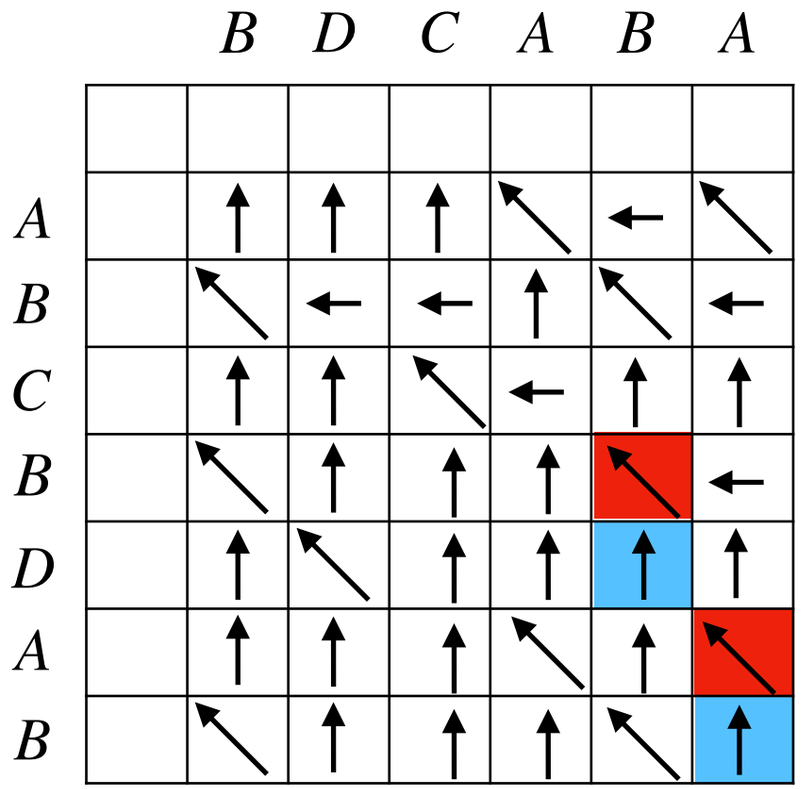
Example

```
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)
```



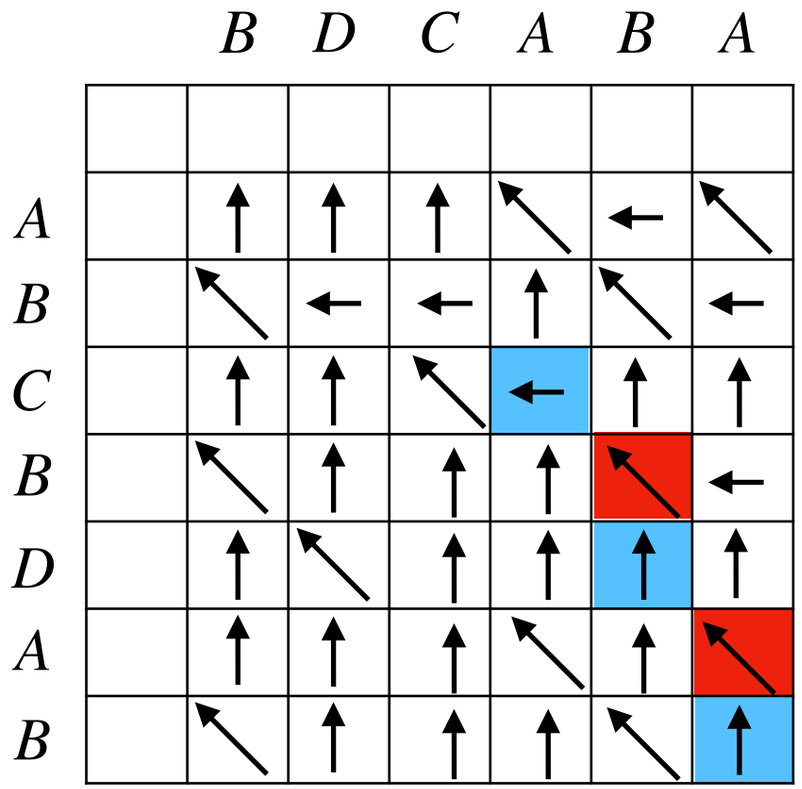
Example

```
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)
```



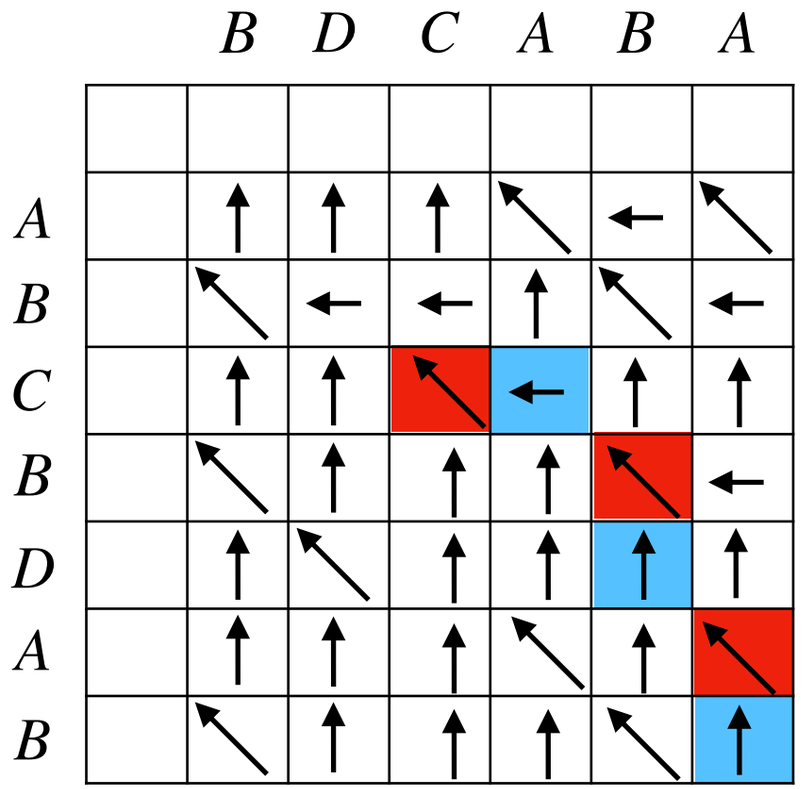
Example

```
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)
```



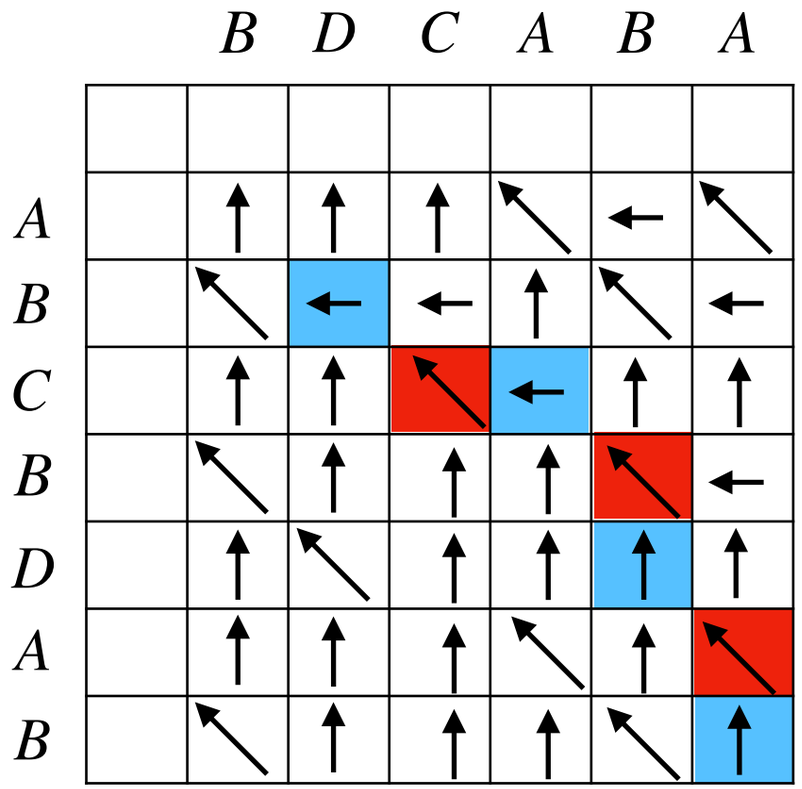
Example

```
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)
```



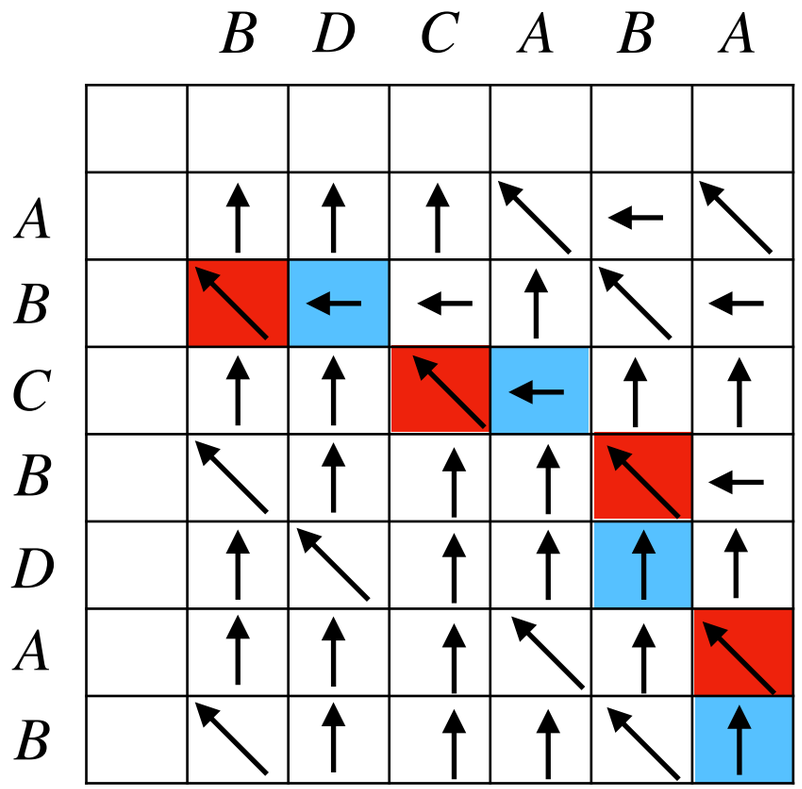
Example

```
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)
```



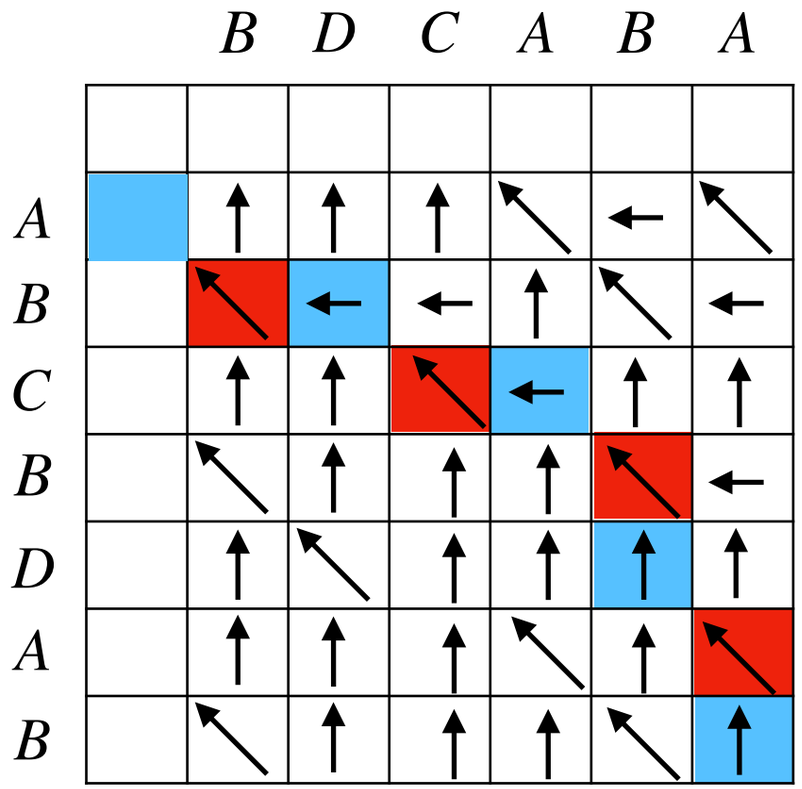
Example

```
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)
```



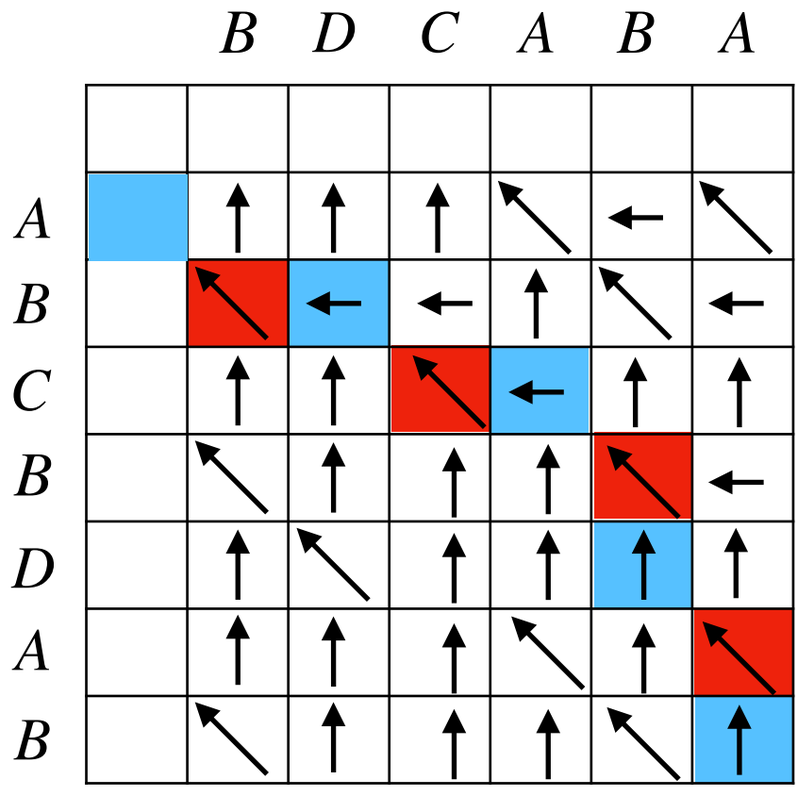
Example

```
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

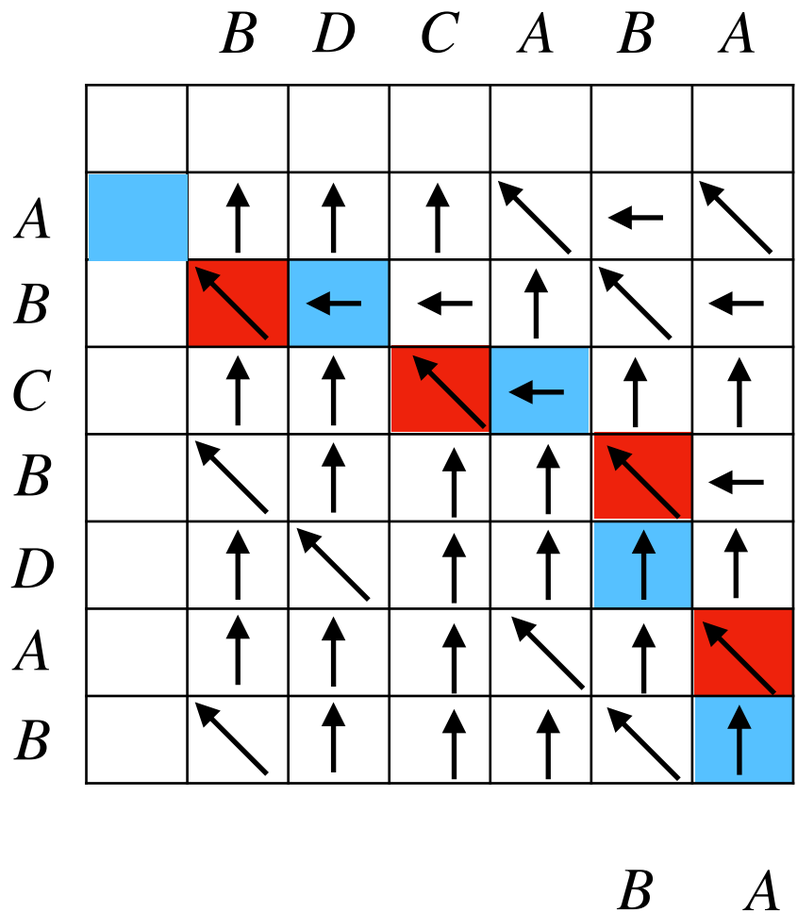
```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



A

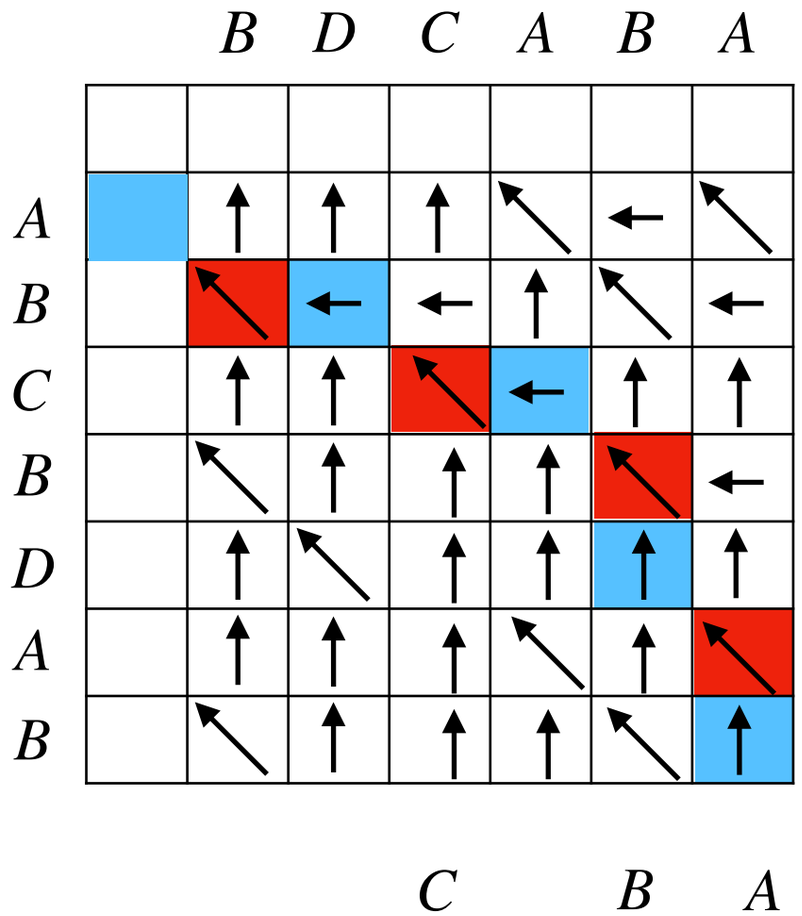
Example

```
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)
```



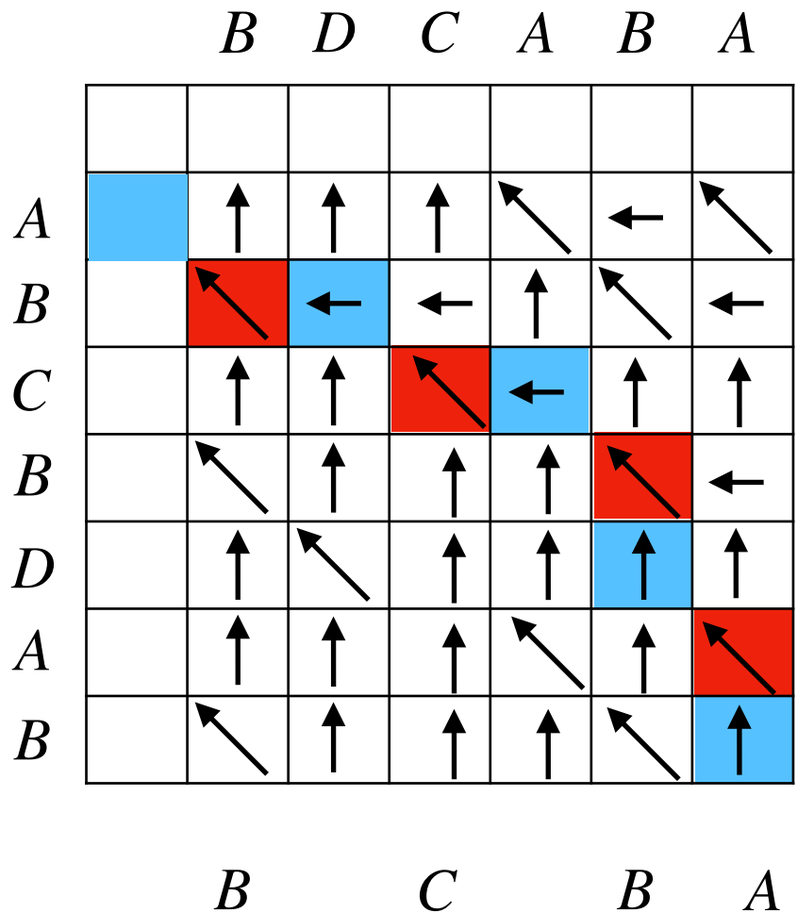
Example

```
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

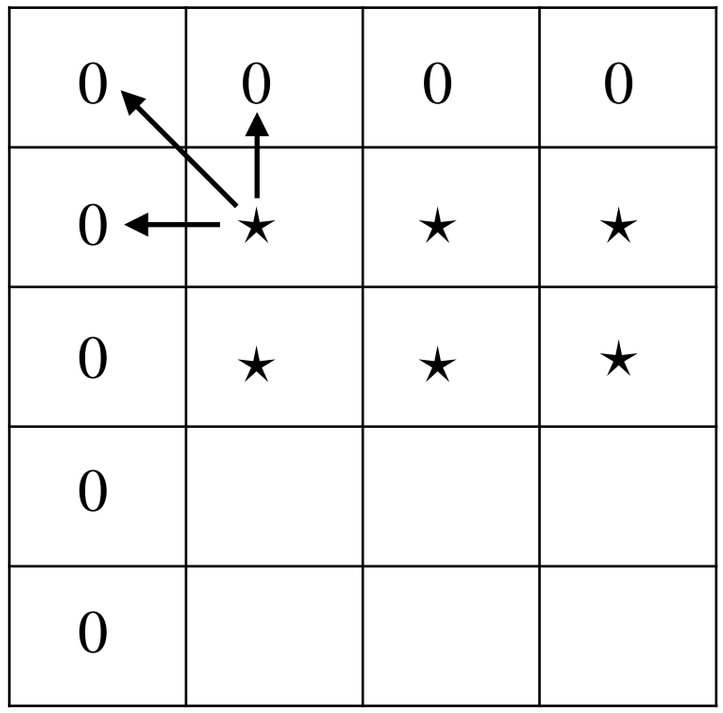
```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return              // the LCS has length 0
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i           // same as y_j
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



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?

The subset sum problem

We are given a set of n items {*1*, *2*, … , *n*}.

Each item *i* has a non-negative weight w_i.

We are given a bound W.

Goal: Select a subset S of the items such that

$\sum_{i \in N} w_i \leq W$ and $\sum_{i \in N} w_i$ is maximised.

Dynamic Programming

Dynamic Programming

We need to identify the appropriate subproblems to use in order to solve the main problem.

Dynamic Programming

We need to identify the appropriate subproblems to use in order to solve the main problem.

Let O_i be the optimal solution to the subset be the optimal solution to the subset sum problem, using a subset of {*1*, *2*, … , *i*}, and let OPT(*i*) be its value.

Dynamic Programming

We need to identify the appropriate subproblems to use in order to solve the main problem.

Let O_i be the optimal solution to the subset be the optimal solution to the subset sum problem, using a subset of {*1*, *2*, … , *i*}, and let OPT(*i*) be its value.

Hence O is O_n, and OPT = OPT(*n*)

Dynamic Programming

We need to identify the appropriate subproblems to use in order to solve the main problem.

Let O_i be the optimal solution to the subset be the optimal solution to the subset sum problem, using a subset of {*1*, *2*, … , *i*}, and let OPT(*i*) be its value.

Hence O is O_n, and OPT = OPT(*n*)

Should item *n* be in the optimal solution O or not?

Dynamic Programming

We need to identify the appropriate subproblems to use in order to solve the main problem.

Let O_i be the optimal solution to the subset be the optimal solution to the subset sum problem, using a subset of {*1*, *2*, … , *i*}, and let OPT(*i*) be its value.

Hence O is O_n, and OPT = OPT(*n*)

Should item *n* be in the optimal solution O or not?

If no, then OPT(*n-1*) = OPT(*n*)

Dynamic Programming

We need to identify the appropriate subproblems to use in order to solve the main problem.

Let O_i be the optimal solution to the subset be the optimal solution to the subset sum problem, using a subset of {*1*, *2*, … , *i*}, and let OPT(*i*) be its value.

Hence O is O_n, and OPT = OPT(*n*)

Should item *n* be in the optimal solution O or not?

If no, then OPT(*n-1*) = OPT(*n*)

If yes, ?

If n is in O

If n is in O

What information do we get about the other items?

If *n* is in O

What information do we get about the other items?

There is no reason to a-priori exclude any remaining item, unless adding it would exceed the weight.

If *n* is in O

What information do we get about the other items?

There is no reason to a-priori exclude any remaining item, unless adding it would exceed the weight.

The only information that we really get is that we now have weight W - w_n left.

What we really need

What we really need

To find the optimal value OPT(n), we need

What we really need

To find the optimal value OPT(n), we need

The optimal value OPT(n-1) if n is not in O.

What we really need

To find the optimal value OPT(*n*), we need

The optimal value OPT(*n-1*) if *n* is not in O.

The optimal value of the solution on input {*1*, *2* , …, *n-1*} and w = W - w_n .

What we really need

To find the optimal value OPT(*n*), we need

The optimal value OPT(*n-1*) if *n* is not in O.

The optimal value of the solution on input {*1*, *2* , …, *n-1*} and w = W - w_n .

How many subproblems do we need?

What we really need

To find the optimal value OPT(*n*), we need

The optimal value OPT(*n-1*) if *n* is not in O.

The optimal value of the solution on input {*1*, *2* , …, *n-1*} and w = W - w_n .

How many subproblems do we need?

One for each initial set {*1*, *2* , …, *i*} of items and each possible value for the remaining weight w.

Subproblems

Subproblems

Assumptions:

Subproblems

Assumptions:

W is an integer.

Subproblems

Assumptions:

W is an integer.

Every w_i is an integer.

Subproblems

Assumptions:

W is an integer.

Every w_i is an integer.

We will have one subproblem for each $i=0,1,\ldots,n$ and each integer $0 \le w \le W$.

Subproblems

Assumptions:

W is an integer.

Every w_i is an integer.

We will have one subproblem for each $i=0,1,\ldots,n$ and each integer $0 \le w \le W$.

Let OPT(i,w) be the value of the optimal solution on subset $\{1, 2, \ldots, i\}$ and maximum allowed weight w.

Subproblems

Subproblems

Using this notation, what are we looking for?

Subproblems

Using this notation, what are we looking for?

OPT(*n*,W)

Subproblems

Using this notation, what are we looking for?

OPT(*n*,W)

Should item *n* be in the optimal solution O or not?

Subproblems

Using this notation, what are we looking for?

OPT(*n*,W)

Should item *n* be in the optimal solution O or not?

If no, then OPT(*n*,W) = OPT(*n-1*,W).

Subproblems

Using this notation, what are we looking for?

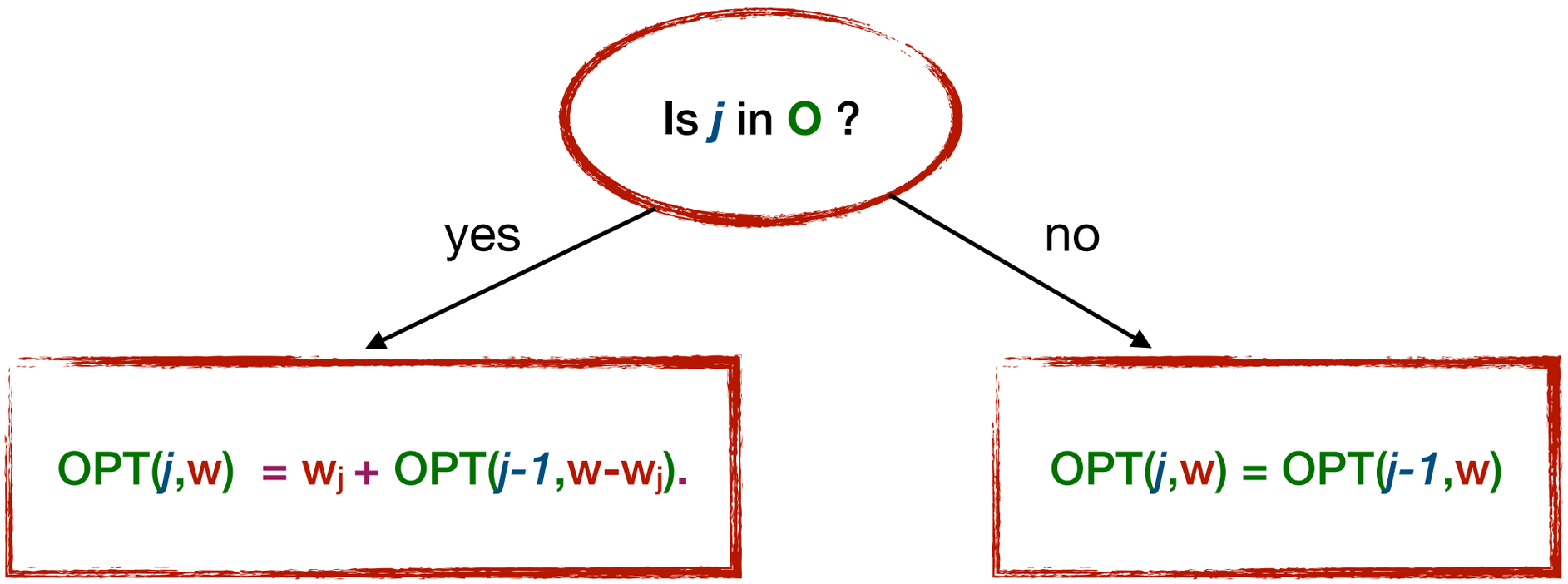
OPT(*n*,W)

Should item *n* be in the optimal solution O or not?

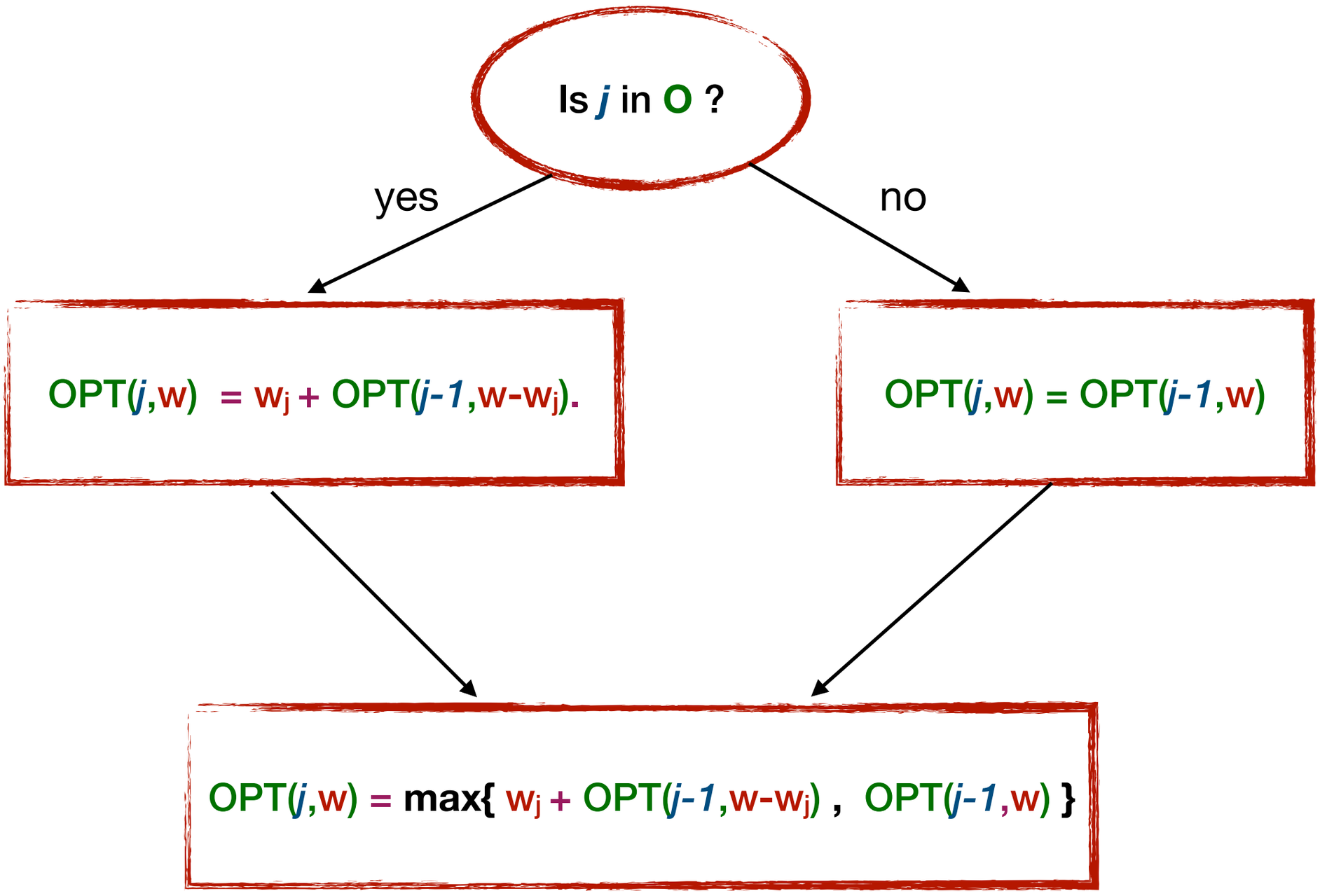
If no, then OPT(*n*,W) = OPT(*n-1*,W).

If yes, then OPT(*n*,W) = w_n + OPT(*n-1*,W-w_n).

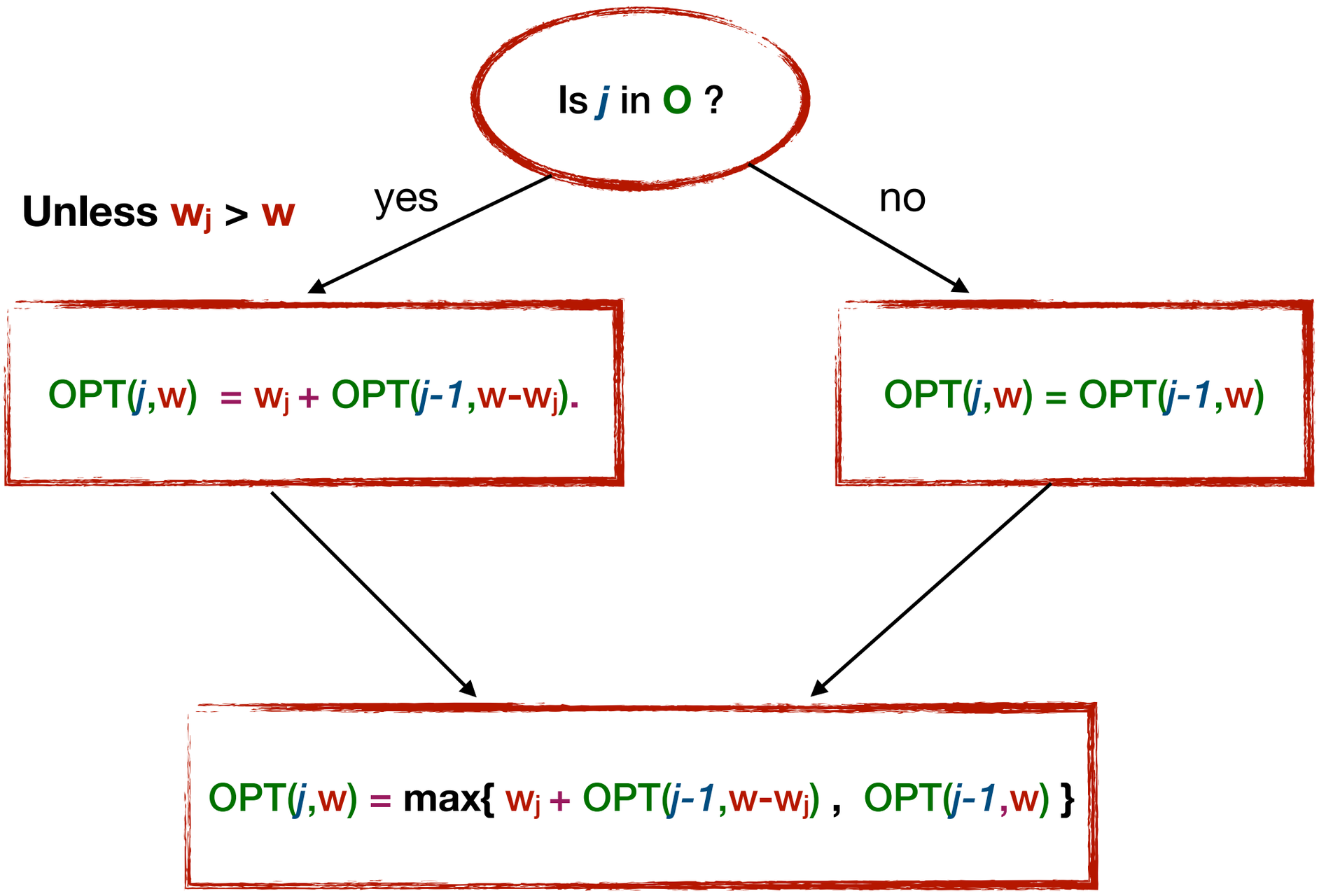
Subproblems



Subproblems



Subproblems



Algorithm

Algorithm **SubsetSum**(*n*,W)

Array M=[*0 … n*, 0 … W]
Initialise M[*0*, w] = 0, for each w = 0, 1 , … , W

For i = *1*, *2*, … , *n*
For w = 0 , … , W
If (w_i > w) * *If the item does not fit* *\
M[*i*, w] = M[*i-1*, w]
Else
M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
EndIf

Return M[*n*, W]

Two dimensional array

n	0										
n-1	0										
…	0										
…	0										
i	0										
i-1	0										
…	0										
2	0										
1	0										
0	0	0	0	0	0	0	0	0	0	0	0
	0	1	…	$w-w_i$			w				W

Two dimensional array

											OPT(*n*,W)
n	0										
n-1	0										
…	0										
…	0										
i	0										
i-1	0										
…	0										
2	0										
1	0										
0	0	0	0	0	0	0	0	0	0	0	0
	0	1	…	$w-w_i$			w				W

Two dimensional array

n	0										
n-1	0										
…	0										
…	0										
i	0										
i-1	0										
…	0										
2	0										
1	0										
0	0	0	0	0	0	0	0	0	0	0	0
	0	1	…	$w-w_i$			w				W

OPT(*n*,W)

Two dimensional array

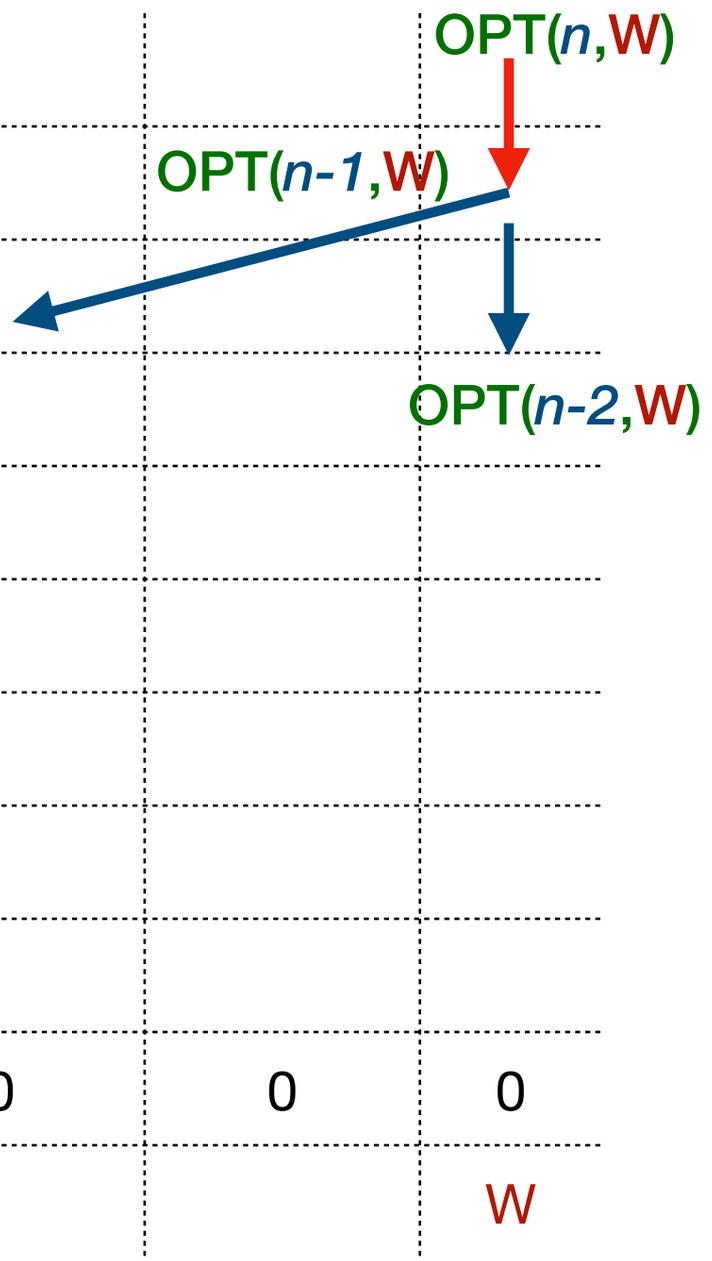
n	0										
n-1	0										
…	0										
…	0										
i	0										
i-1	0										
…	0										
2	0										
1	0										
0	0	0	0	0	0	0	0	0	0	0	0
	0	1	…	$w-w_i$			w				W

OPT(*n*,W)

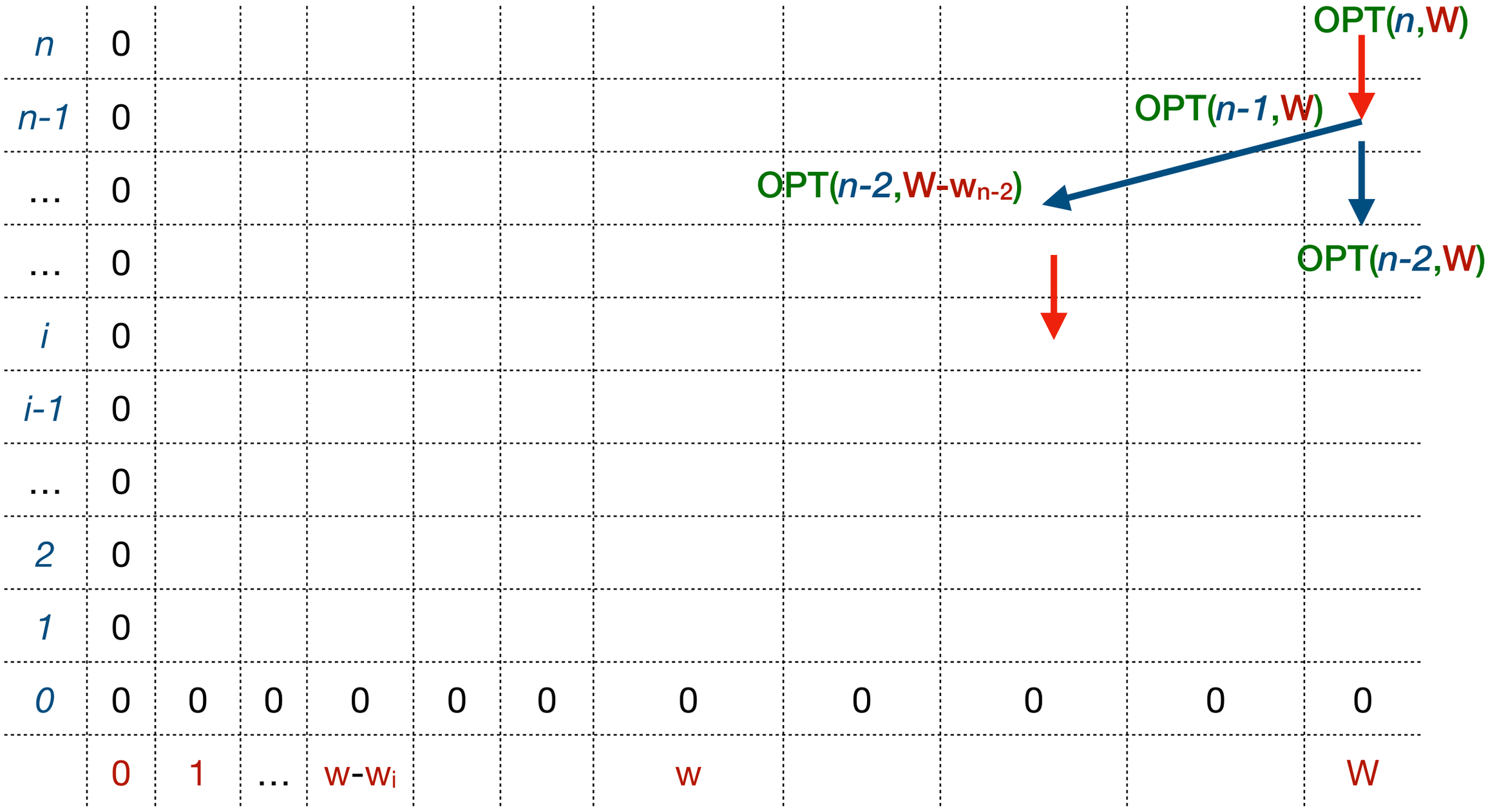
OPT(*n-1*,W)

Two dimensional array

n	0											
n-1	0											
…	0											
…	0											
i	0											
i-1	0											
…	0											
2	0											
1	0											
0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	…	$w-w_i$			w					W

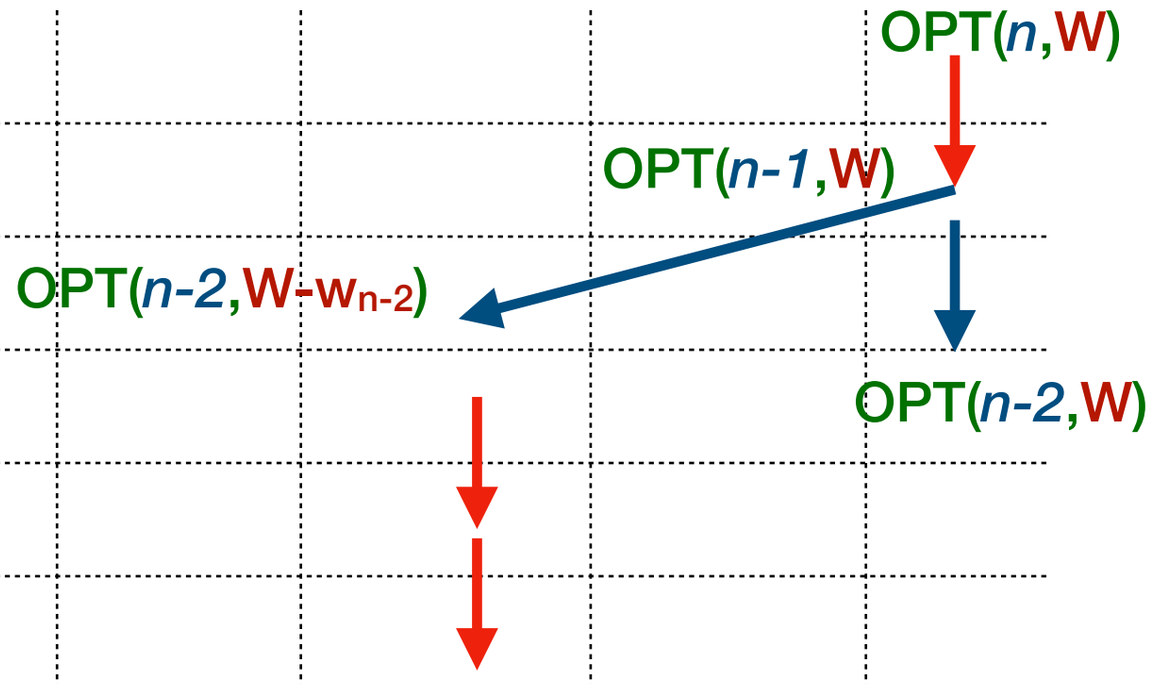


Two dimensional array

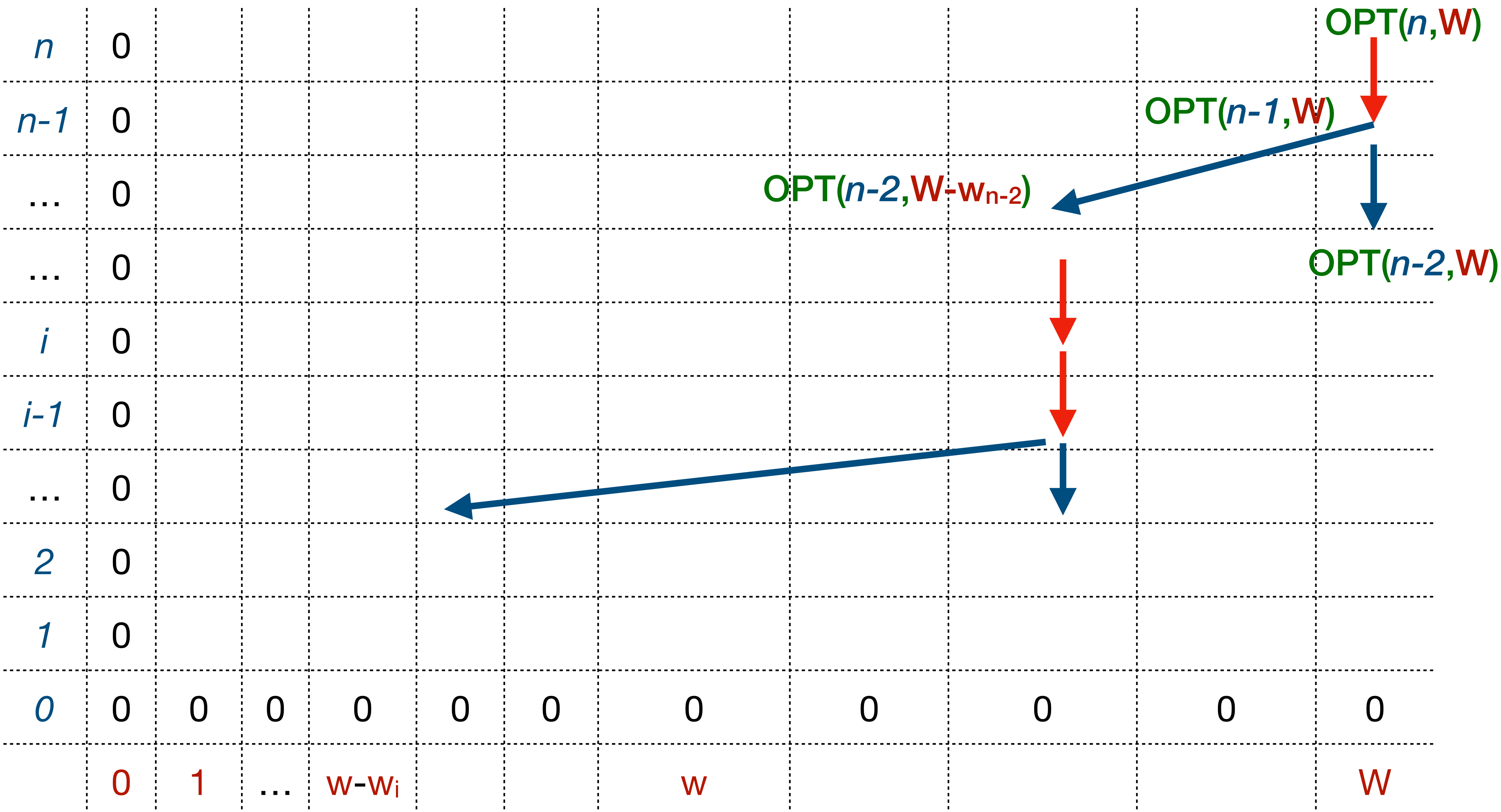


Two dimensional array

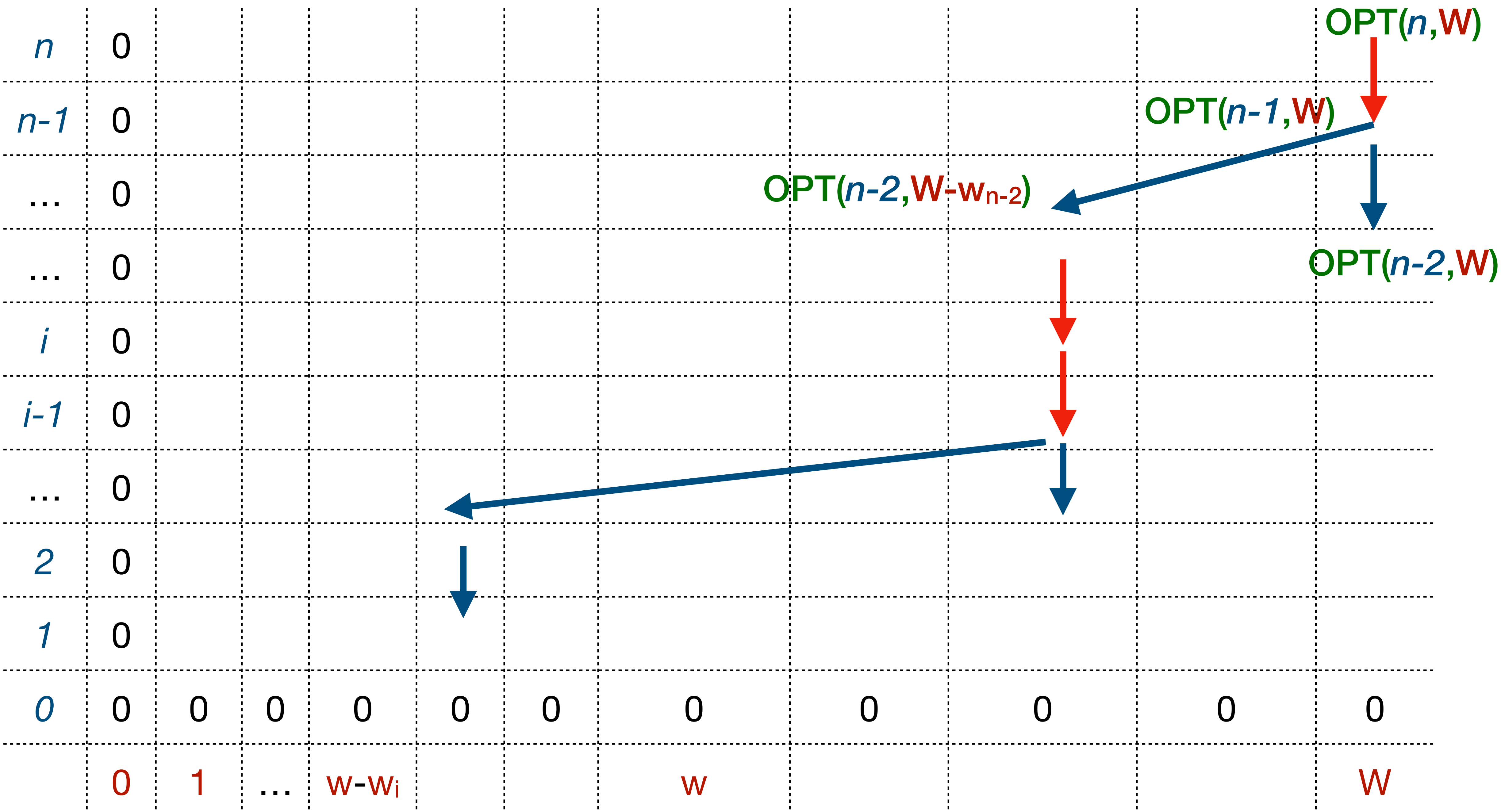
n	0										
n-1	0										
…	0										
…	0										
i	0										
i-1	0										
…	0										
2	0										
1	0										
0	0	0	0	0	0	0	0	0	0	0	0
	0	1	…	$w-w_i$			w				W



Two dimensional array



Two dimensional array



Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1							
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

Array M=[*0* … *n*, 0 … W]

Initialise M[*0*, w] = 0, for each w = 0, 1 , … , W

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1							
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1							
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1							
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1							
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W

If (w_i > w) **doesn't fit**

M[*i*, w] = M[*i-1*, w]

Else

M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**

EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1	0						
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1	0						
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w) **doesn't fit**
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1	0	0					
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1	0	0					
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w) **fits**
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

i \ w	0	1	2	3	4	5	6
3							
2							
1	0	0					
0	0	0	0	0	0	0	0

For w = 0 , … , W
  If (w_i > w) **fits**
    M[i, w] = M[i-1, w]
  Else
    M[i, w] = **max{**M[i-1, w] , w_i + M[i-1, w-w_i]**}**
  EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1	0	0					
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
  If (w_i > w) **fits**
    M[*i*, w] = M[*i-1*, w]
  Else
    M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
  EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1	0	0	2				
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2							
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2	0	0					
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2	0	0	2				
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2	0	0	2	2			
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2	0	0	2	2	4		
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3							
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

i							
3	0	0	2	2	4	4	4
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3	0	0	2				
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3	0	0	2	3			
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3	0	0	2	3	4		
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3	0	0	2	3	4	5	
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

Example

n=3, W=6, w_1 = w_2 = 2 and w_3 = 3.

3	0	0	2	3	4	5	5
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For $w = 0, \ldots, W$

    If ($w_i > w$)
        $M[i, w] = M[i-1, w]$
    Else
        $M[i, w] = \mathbf{max\{}M[i-1, w], w_i + M[i-1, w-w_i]\mathbf{\}}$
    EndIf

Example

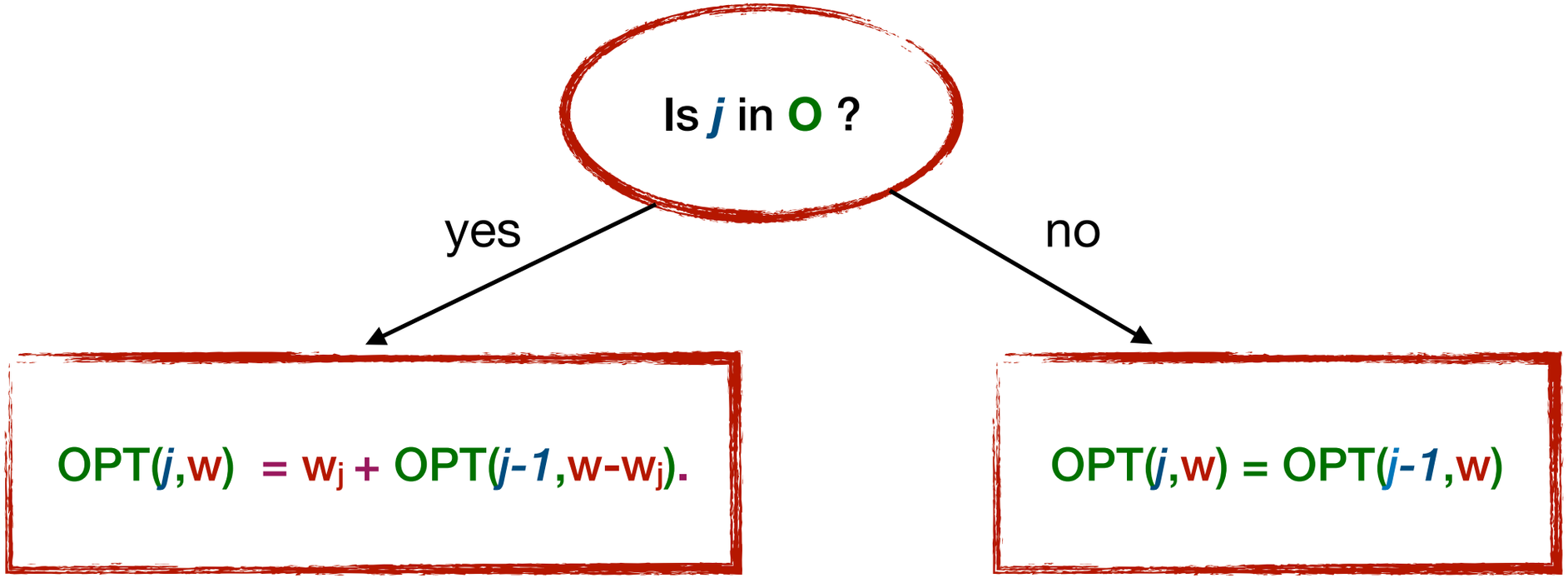
n=3, W=6, $w_1 = w_2 = 2$ and $w_3 = 3$.

Optimal value

3	0	0	2	3	4	5	5
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

For w = 0 , … , W
    If (w_i > w)
        M[*i*, w] = M[*i-1*, w]
    Else
        M[*i*, w] = **max{**M[*i-1*, w] , w_i + M[*i-1*, w-w_i]**}**
    EndIf

From values to solutions



Running Time

Running Time

Similar to weighted interval scheduling.

Running Time

Similar to weighted interval scheduling.

We are building up a table M of solutions (instead of an array).

Running Time

Similar to weighted interval scheduling.

We are building up a table M of solutions (instead of an array).

We compute each value M(*i*, w) of the table in O(1) time using the previous values.

Running Time

Similar to weighted interval scheduling.

We are building up a table M of solutions (instead of an array).

We compute each value M(*i*, w) of the table in O(1) time using the previous values.

What is the running time overall?

Running Time

Similar to weighted interval scheduling.

We are building up a table M of solutions (instead of an array).

We compute each value M(*i*, w) of the table in O(1) time using the previous values.

What is the running time overall?

How many entries does the table M have?

Running Time

Running Time

SubsetSum(n,W) runs in time O(nW).

Running Time

SubsetSum(n,W) runs in time O(nW).

Is this a polynomial time algorithm?

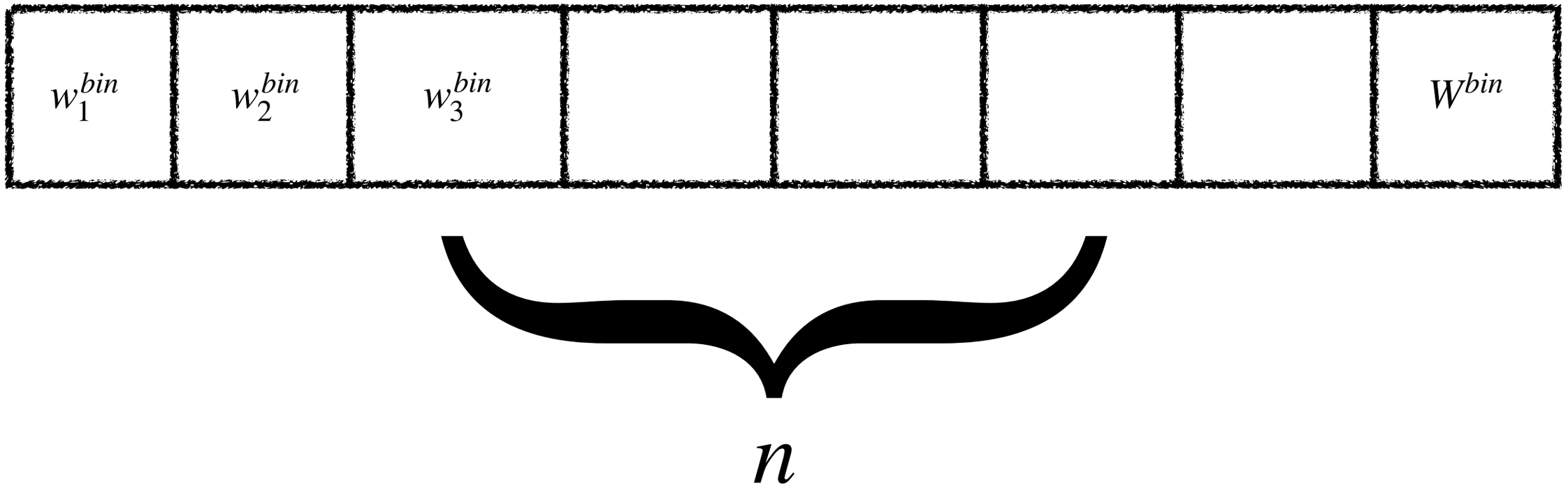
Running Time

SubsetSum(n,W) runs in time O(nW).

Is this a polynomial time algorithm?

Not quite, because it depends on W.

Our input



Running Time

SubsetSum(n,W) runs in time O(nW).

Is this a polynomial time algorithm?

Running Time

SubsetSum(n,W) runs in time O(nW).

Is this a polynomial time algorithm?

It is *pseudopolynomial*, as it runs in time polynomial in n and the unary representation of W.

Running Time

SubsetSum(n,W) runs in time O(nW).

Is this a polynomial time algorithm?

It is *pseudopolynomial*, as it runs in time polynomial in n and the unary representation of W.

It is fairly efficient, if in the numbers involved in the input are reasonably small.

Should we be happy?

Should we be happy?

Pseudopolynomial is good in some cases.

Should we be happy?

Pseudopolynomial is good in some cases.

But why not polynomial?

Should we be happy?

Pseudopolynomial is good in some cases.

But why not polynomial?

How hard is it to design a polynomial time algorithm for subset sum?

Should we be happy?

Pseudopolynomial is good in some cases.

But why not polynomial?

How hard is it to design a polynomial time algorithm for subset sum?

Hard enough to justify a reward of 1 million dollars!

Sl

BUSINESS INSIDER

If you can solve this math problem you'll get a $1 million prize — and change internet security as we know it

Pseudopolynomial is good in some cases.

But why not polynomial?

How hard is it to design a polynomial time algorithm for subset sum?

Hard enough to justify a reward of 1 million dollars!

Sl

BUSINESS INSIDER

If you can solve this math problem you'll get a $1 million prize — and change internet security as we know it

Pseudopolynomial is good in some cases.

But why not polynomial?

How hard is it to design a polynomial time algorithm for subset sum?

Hard enough to justify a reward of 1 million dollars!

Subset sum is NP-hard!

Sl

BUSINESS INSIDER

If you can solve this math problem you'll get a $1 million prize — and change internet security as we know it

Pseudopolynomial is good in some cases.

But why not polynomial?

How hard is it to design a polynomial time algorithm for subset sum?

Hard enough to justify a reward of 1 million dollars!

Subset sum is NP-hard!

More about that later on in the course.