Design and Analysis of Algorithms

Dina El-Manakhly, Ph. D.

dina_almnakhly@science.suez.edu.eg

Dynamic programming (introduction)

- The Fibonacci numbers F(n) are defined as follows:

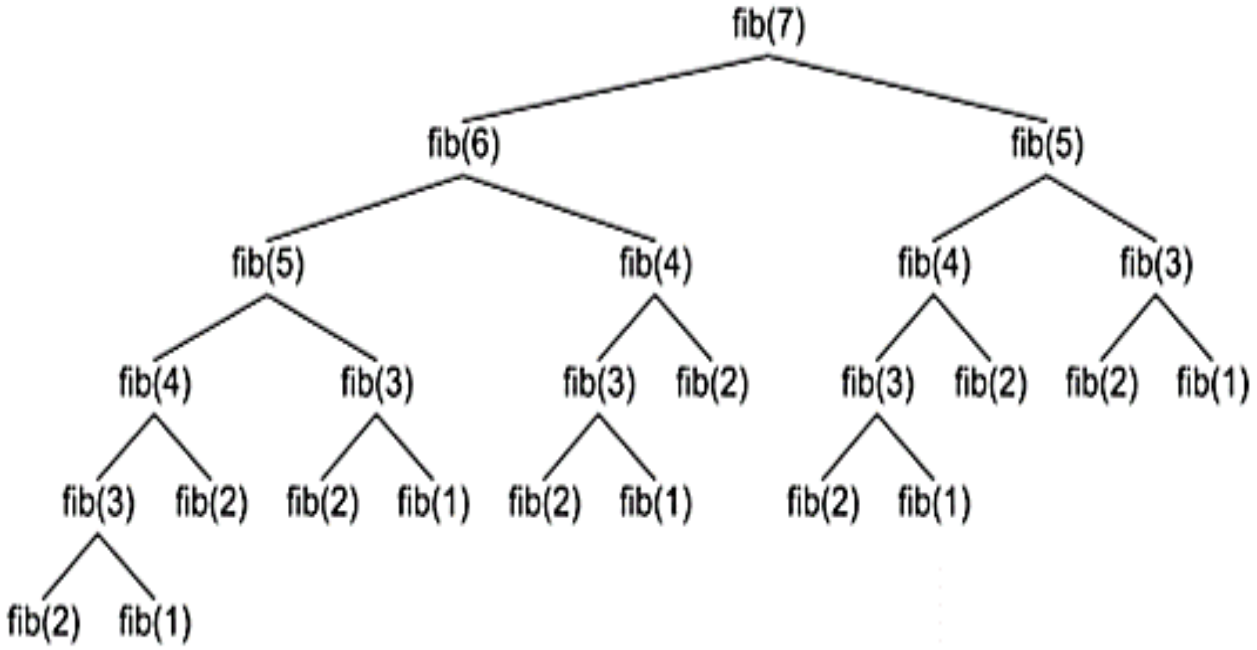
$$f(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f(n-1) + f(n-2) & n \geq 2 \end{cases}$$

- Fibonacci numbers are: 0,1,1, 2, 3, 5, 8, 13,….
- Each number in the sequence is the sum of the two preceding numbers.

Dynamic programming (introduction)

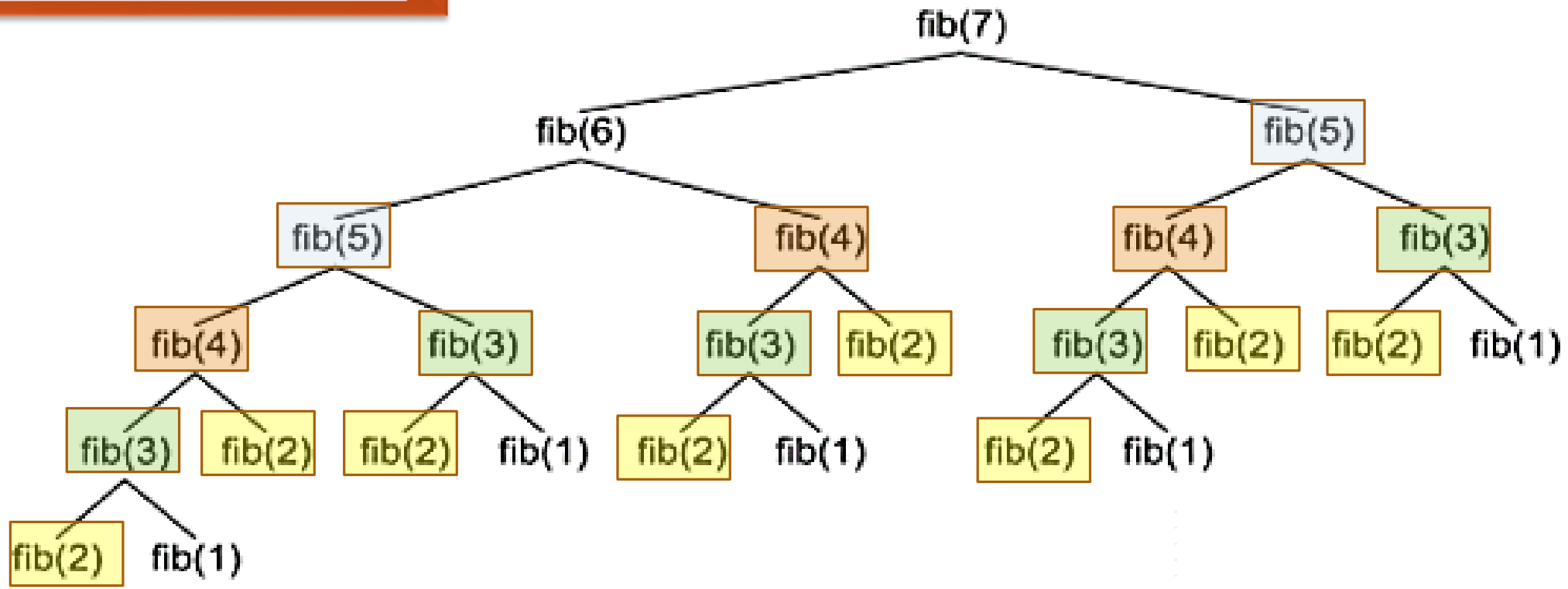
Algorithm Fibonacci(n) Recursively

```
Begin
 if (n=0) then return 0
 else if  (n=1) then return 1
 else return Fibonacci(n-1)+ Fibonacci(n-2)
End.
```



The running time required to compute $f(n)$ is exponential in the value of n ($O(2^n)$).

Observations



- Overlapping sub problems: where the solutions of the same sub-problems are required again and again.
- Optimal substructure: optimal solution to problem consists of optimal solutions to sub problems.

Computing the n^{th} Fibonacci number using bottom-up iteration and recording results:

$F(0) = 0$
$F(1) = 1$
$F(2) = 1+0 = 1$
$F(3) = 1+1=2$
…
$F(n\text{-}2) =$
$F(n\text{-}1) =$
$F(n) = F(n\text{-}1) + F(n\text{-}2)$

0	1	1	……	$F(n\text{-}2)$	$F(n\text{-}1)$	$F(n)$

What if we solve it recursively?

Efficiency:

- Time: O(n)

Dynamic programming

- Dynamic Programming is a technique in computer programming that helps to efficiently solve a class of problems that have overlapping sub problems and optimal substructure property.

DP is used to solve problems with the following characteristics:

(1) Optimal sub-structure: an optimal solution to the problem contains within it optimal solutions to sub-problems.

(2) Overlapping sub problems: there exist some places where we solve the same sub problem more than once.

Dynamic Programming is an algorithm design technique for optimization problems: often minimizing or maximizing.

Dynamic programming

- Dynamic programming works by storing the result of sub-problems so that when their solutions are required, they are at hand and we do not need to recalculate them.
- This technique of storing the value of sub-problems is called memoization. By saving the values in the array, we save time for computations of sub-problems we have already come across.

Divide and Conquer (D&C) & Dynamic programming (DP) methods:

- Similarity
  1. Both: split a problem into separate sub-problems of small size.
  2. Both: solve problems by combining solutions to sub-problems.
- Dissimilarity
  1. DP sub-problems are dependent (overlaps), while D&C sub-problems are independent.
  2. D&C top down, while DP bottom up techniques.

Greedy method (G) & Dynamic programming (DP) method:

- Similarity
  1. Both techniques solve optimization problems.
  2. Both techniques exhibit optimal substructure.

- Dissimilarity
  1. In DP many sequences are generated, while in G technique only one decision sequence is generated.
  2. G method is easier than DP. DP is more expensive. Therefore, we first try greedy algorithm. If it fails then try dynamic programming.

Different Types of Dynamic Programming Algorithms

1. Longest Common Subsequence
2. Floyd-Warshall Algorithm

Longest Common Subsequence

- The longest common subsequence (LCS) is defined as the longest subsequence that is common to all the given sequences, provided that the elements of the subsequence are not required to occupy consecutive positions within the original sequences.
- If S1 and S2 are the two given sequences then, Z is the common subsequence of S1 and S2 if Z is a subsequence of both S1 and S2. Furthermore, Z must be a **strictly increasing sequence** of the indices of both S1 and S2.

  In a strictly increasing sequence, the indices of the elements chosen from the original sequences must be in ascending order in Z.

  If S1 = {B, C, D, A, A, C, D}

Then, {A, D, B} cannot be a subsequence of S1 as the order of the elements is not the same (i.e.. not strictly increasing sequence).

Example

If
S1 = {B, C, D, A, A, C, D}
S2 = {A, C, D, B, A, C}

Then, common subsequences are
{B, C},
{C, D, A, C},
{D, A, C},
{A, A, C},
{A, C},
{C, D},

Among these subsequences, {C, D, A, C} is the longest common subsequence.

Using Dynamic Programming to find the LCS

The following steps are followed for finding the longest common subsequence:

1. Create a table of dimension n+1*m+1 where n and m are the lengths of A and B respectively. The first row and the first column are filled with zeros.

2. If the character corresponding to the current row and current column are matching, then fill the current cell by adding one to the diagonal element. Point an arrow to the diagonal cell. Let L[i,j] denote the length of a longest common subsequence.

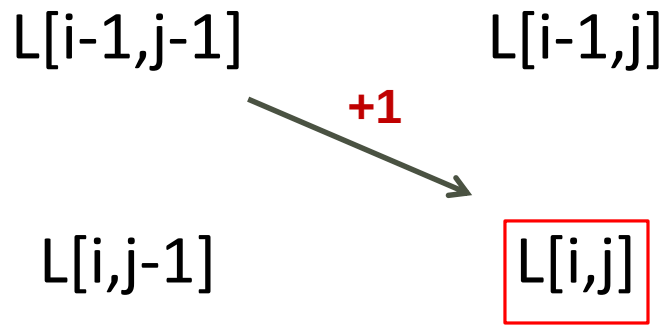
$$\text{If } a_i = b_j \text{ then } L[i,j] = L[i-1,j-1] + 1$$

3. Else take the maximum value from the previous column and previous row element for filling the current cell. Point an arrow to the cell with maximum value. If they are equal, point to any of them.

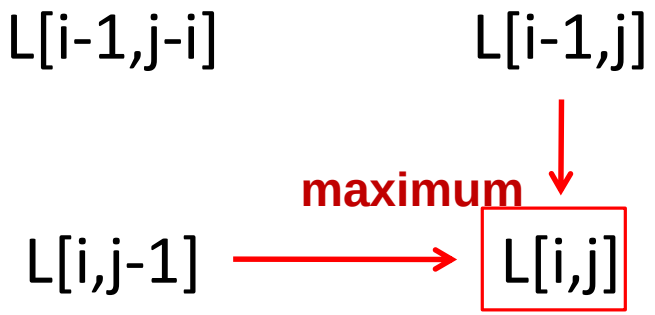
$$\text{If } a_i \neq b_j \text{ then } L[i,j] = \text{Max}\{L[i,j-1], L[i-1,j]\}$$

Step 2 and 3

If $a_i=b_j$ then L[i,j]=L[i-1,j-1]+1.



If $a_i \neq b_j$ then L[i,j]= Max{L[i,j-1],L[i-1,j]}



Example 1

Given two strings A=lion and B=line. Find LCS.

Find the length of the longest common subsequence

B

A

	ε	l	i	n	e
ε					
l					
l					
o					
n					

	ε	**l**	**i**	**n**	**e**
ε	0	0	0	0	0
l	0				
l	0				
o	0				
n	0				

Given two strings A=lion and B=line. Find LCS.

	ε	l	i	n	e
ε	0	0	0	0	0
l	0	1	1	1	1
i	0				
O	0				
n	0				

Given two strings A=lion and B=line. Find LCS.

	ε	l	i	n	e
ε	0	0	0	0	0
l	0	1	1	1	1
i	0	1	2	2	2
o	0				
n	0				

Given two strings A=lion and B=line. Find LCS.

	ε	l	i	n	e
ε	0	0	0	0	0
l	0	1	1	1	1
i	0	1	2	2	2
o	0	1	2	2	2
n	0				

Given two strings A=lion and B=line. Find LCS.

	ε	l	i	n	e
ε	0	0	0	0	0
l	0	1	1	1	1
i	0	1	2	2	2
o	0	1	2	2	2
n	0	1	2	3	3

The bottom right corner is the length of the LCS

Example 2

Find the length of the longest common subsequence

Given two strings A=alignment and B=algorithm. Find LCS.

		a	l	i	g	n	m	e	n	t
	0	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1	1
l	0									
g	0									
o	0									
r	0									
i	0									
t	0									
h	0									
m	0									

Given two strings A=alignment and B=algorithm. Find LCS.

		a	l	i	g	n	m	e	n	t
	0	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1	1
l	0	1	2	2	2	2	2	2	2	2
g	0									
o	0									
r	0									
i	0									
t	0									
h	0									
m	0									

Given two strings A=alignment and B=algorithm. Find LCS.

		a	l	i	g	n	m	e	n	t
	0	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1	1
l	0	1	2	2	2	2	2	2	2	2
g	0	1	2	2	3	3	3	3	3	3
o	0									
r	0									
i	0									
t	0									
h	0									
m	0									

Given two strings A=alignment and B=algorithm. Find LCS.

		a	l	i	g	n	m	e	n	t
	0	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1	1
l	0	1	2	2	2	2	2	2	2	2
g	0	1	2	2	3	3	3	3	3	3
o	0	1	2	2	3	3	3	3	3	3
r	0									
i	0									
t	0									
h	0									
m	0									

Given two strings A=alignment and B=algorithm. Find LCS.

		a	l	i	g	n	m	e	n	t
	0	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1	1
l	0	1	2	2	2	2	2	2	2	2
g	0	1	2	2	3	3	3	3	3	3
o	0	1	2	2	3	3	3	3	3	3
r	0	1	2	2	3	3	3	3	3	3
i	0									
t	0									
h	0									
m	0									

Design and Analysis of Algorithms

Given two strings A=alignment and B=algorithm. Find LCS.

		a	l	i	g	n	m	e	n	t
	0	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1	1
l	0	1	2	2	2	2	2	2	2	2
g	0	1	2	2	3	3	3	3	3	3
o	0	1	2	2	3	3	3	3	3	3
r	0	1	2	2	3	3	3	3	3	3
i	0	1	2	3	3	3	3	3	3	3
t	0									
h	0									
m	0									

Given two strings A=alignment and B=algorithm. Find LCS.

		a	l	i	g	n	m	e	n	t
	0	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1	1
l	0	1	2	2	2	2	2	2	2	2
g	0	1	2	2	3	3	3	3	3	3
o	0	1	2	2	3	3	3	3	3	3
r	0	1	2	2	3	3	3	3	3	3
i	0	1	2	3	3	3	3	3	3	3
t	0	1	2	3	3	3	3	3	3	4
h	0									
m	0									

Given two strings A=alignment and B=algorithm. Find LCS.

		a	l	i	g	n	m	e	n	t
	0	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1	1
l	0	1	2	2	2	2	2	2	2	2
g	0	1	2	2	3	3	3	3	3	3
o	0	1	2	2	3	3	3	3	3	3
r	0	1	2	2	3	3	3	3	3	3
i	0	1	2	3	3	3	3	3	3	3
t	0	1	2	3	3	3	3	3	3	4
h	0	1	2	3	3	3	3	3	3	4
m	0									

Given two strings A=alignment and B=algorithm. Find LCS.

		a	l	i	g	n	m	e	n	t
	0	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1	1
l	0	1	2	2	2	2	2	2	2	2
g	0	1	2	2	3	3	3	3	3	3
o	0	1	2	2	3	3	3	3	3	3
r	0	1	2	2	3	3	3	3	3	3
i	0	1	2	3	3	3	3	3	3	3
t	0	1	2	3	3	3	3	3	3	4
h	0	1	2	3	3	3	3	3	3	4
m	0	1	2	3	3	3	4	4	4	4

The bottom right corner is the length of the LCS

Example 3

Find the longest common subsequence and the length

The first sequence

X

The second sequence

Y

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

The first sequence

X

A	C	A	D	B

The second sequence

Y

C	B	D	A

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

The first sequence

X

A	C	A	D	B

The second sequence

Y

C	B	D	A

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

The first sequence

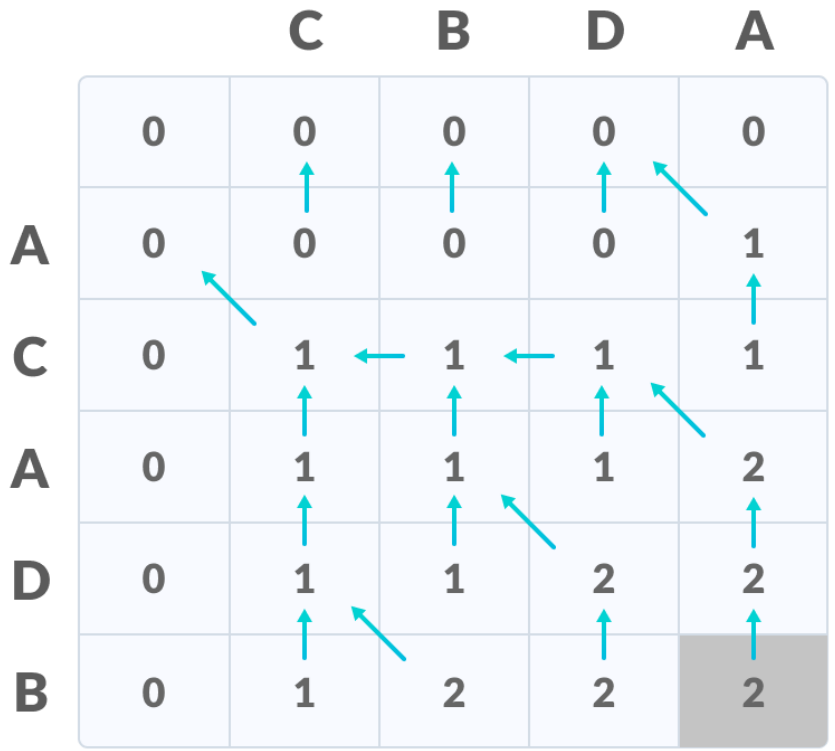
X

A	C	A	D	B

The second sequence

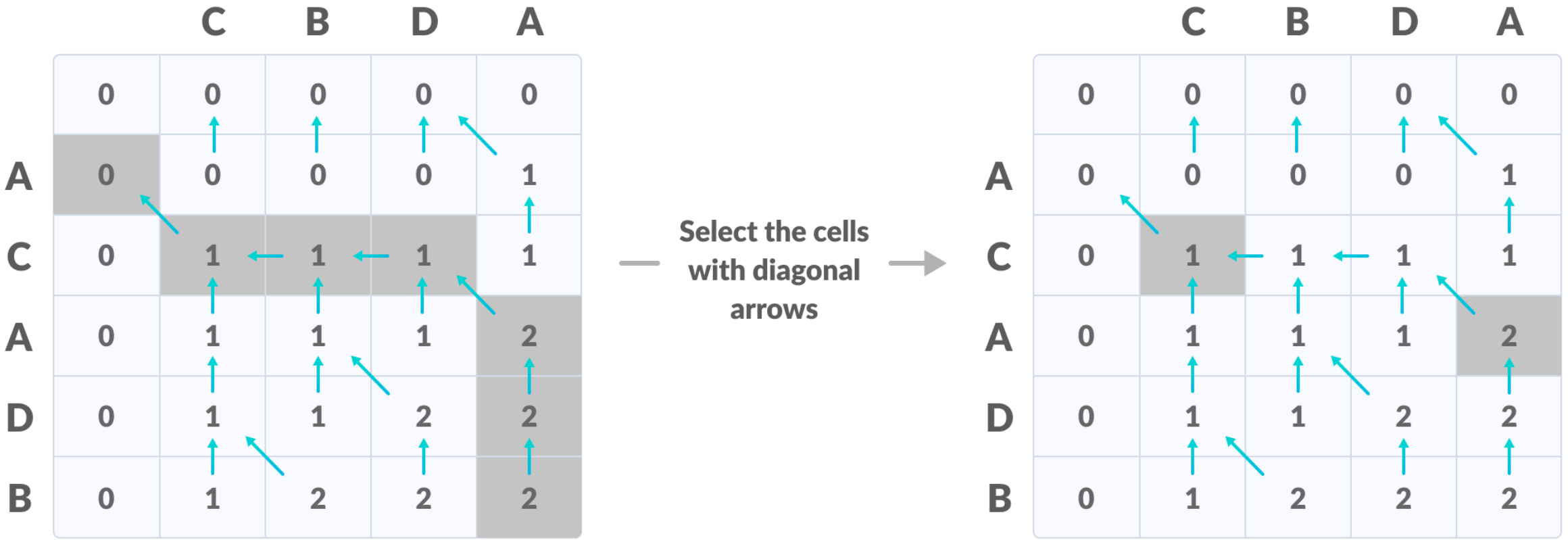
Y

C	B	D	A



The bottom right corner is the length of the LCS

In order to find the longest common subsequence, start from the last element and follow the direction of the arrow. The elements corresponding to () symbol form the longest common subsequence.



Algorithm

Algorithm LCS
Input: two strings A and B of lengths n and m respectively.
Output: The length of the longest common subsequences of A and B.

```
Begin
  for i=0 to n do
    L[i,0]=0
 for j=0 to m do
    L[0,j]=0
 for i=1 to n do
    for j=1 to m do
        if ai = bj then L[i,j]=L[i-1,j-1]+1
        else L[i,j]=max{ L[i,j-1],L[i-1,j] }
  return L[i,j]
End.
```

Time efficiency: $O(n\ m)$.

Home Work
(Dynamic Programming)

Find the longest subsequence common between:

(1) “human” and “chimpanzee”.

(2) “tail” and “tall”.

(3) “common” and “coin”.