5- DYNAMIC PROGRAMMING

5- Dynamic Programming

- Dynamic programming is an algorithm design technique with a rather interesting history. It was invented by a prominent U.S. mathematician, Richard Bellman, in the 1950s as a general method for optimizing multistage decision processes.
- Thus, the word “programming” in the name of this technique stands for “planning” and does not refer to computer programming.
- After proving its worth as an important tool of applied mathematics, dynamic programming has eventually come to be considered, at least in computer science circles, as a general algorithm design technique that does not have to be limited to special types of optimization problems.
- It is from this point of view that we will consider this technique here.

5- Dynamic Programming

- Dynamic programming is a technique for solving problems with overlapping subproblems.
- Typically, these subproblems arise from a recurrence relating a given problem’s solution to solutions of its smaller subproblems.
- Rather than solving overlapping subproblems again and again, dynamic programming suggests solving each of the smaller subproblems only once and recording the results in a table from which a solution to the original problem can then be obtained.

5- Dynamic Programming

- This technique can be illustrated by revisiting the Fibonacci numbers.
- The Fibonacci numbers are the elements of the sequence
- 0 – 1 – 1 – 2 – 3 – 5 – 8 – 13 – 21 – 34 ….
- which can be defined by the simple recurrence
- $F(n) = F(n-1) + F(n-2)$ for $n > 1$
- and two initial conditions
- $F(0) = 0 \quad F(1) = 1$

5- Dynamic Programming

- If we try to use recurrence directly to compute the nth Fibonacci number F (n), we would have to recompute the same values of this function many times.
- Note that the problem of computing F(n) is expressed in terms of its smaller and overlapping subproblems of computing F(n − 1) and F(n − 2).
- So we can simply fill elements of a one-dimensional array with the n + 1 consecutive values of F (n) by starting, in view of initial conditions, with 0 and 1 and using equation as the rule for producing all the other elements.
- Obviously, the last element of this array will contain F (n).

5- Dynamic Programming

- Note that we can, in fact, avoid using an extra array to accomplish this task by recording the values of just the last two elements of the Fibonacci sequence.
- This phenomenon is not unusual, and we shall encounter it in a few more examples in this chapter.
- Thus, although a straightforward application of dynamic programming can be interpreted as a special variety of space-for-time trade-off, a dynamic programming algorithm can sometimes be re-fined to avoid using extra space.

5- Dynamic Programming

- Certain algorithms compute the nth Fibonacci number without computing all the preceding elements of this sequence.
- It is typical of an algorithm based on the classic bottom-up dynamic programming approach, however, to solve all smaller subproblems of a given problem.
- One variation of the dynamic programming approach seeks to avoid solving unnecessary subproblems.
- This technique, exploits so-called memory functions and can be considered a top-down variation of dynamic programming.

5- Dynamic Programming

- Whether one uses the classical bottom-up version of dynamic programming or its top-down variation, the crucial step in designing such an algorithm remains the same: deriving a recurrence relating a solution to the problem to solutions to its smaller subproblems.
- The immediate availability of equation for computing the nth Fibonacci number is one of the few exceptions to this rule.

5- Dynamic Programming

- **Coin-row problem :** There is a row of n coins whose values are some positive integers c1,c2,...,cn, not necessarily distinct. The goal is to pick up the maximum amount of money subject to the constraint that no two coins adjacent in the initial row can be picked up.

- Let F (n) be the maximum amount that can be picked up from the row of n coins. To derive a recurrence for F (n), we partition all the allowed coin selections into two groups: those that include the last coin and those without it.

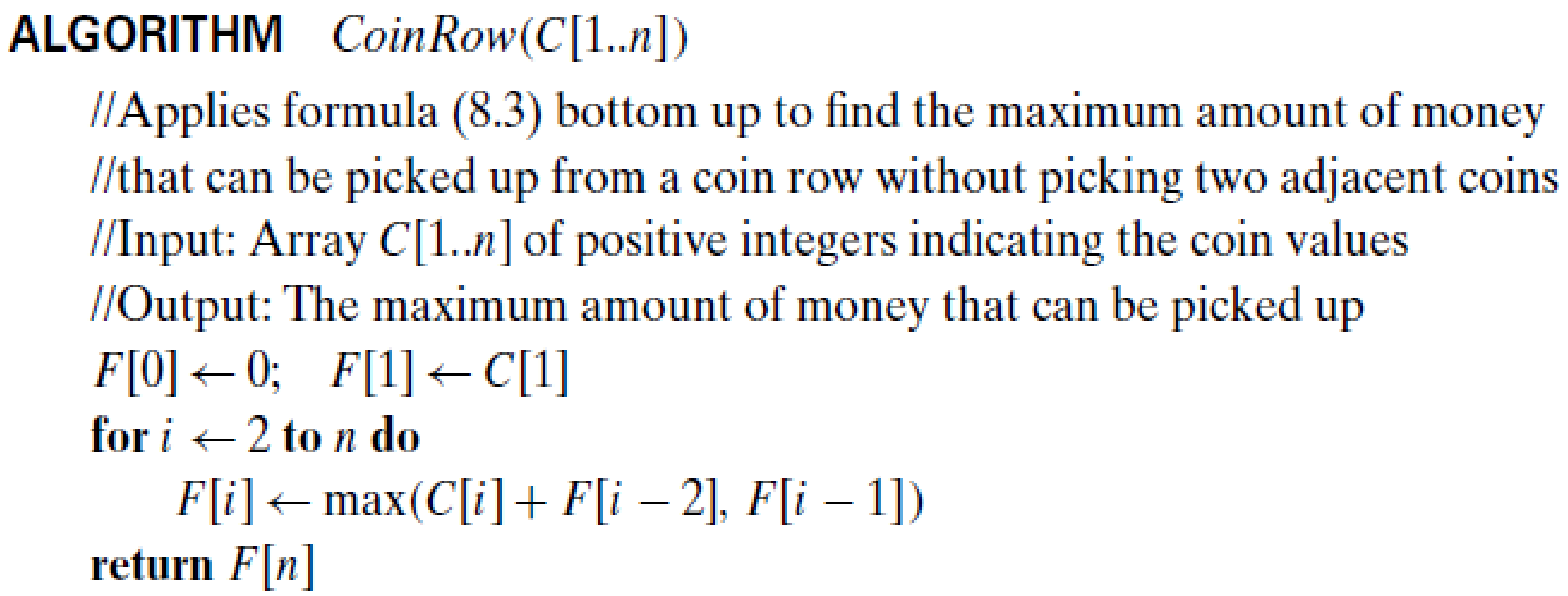
5- Dynamic Programming

- The largest amount we can get from the first group is equal to cn + F(n − 2)—the value of the nth coin plus the maximum amount we can pick up from the first n − 2 coins.

- The maximum amount we can get from the second group is equal to F(n − 1) by the definition of F (n). Thus, we have the following recurrence subject to the obvious initial conditions:

$$F(n) = \max\{c_n + F(n-2), F(n-1)\} \quad \text{for } n > 1,$$
$$F(0) = 0, \qquad F(1) = c_1.$$

5- Dynamic Programming



```
ALGORITHM  CoinRow(C[1..n])
    //Applies formula (8.3) bottom up to find the maximum amount of money
    //that can be picked up from a coin row without picking two adjacent coins
    //Input: Array C[1..n] of positive integers indicating the coin values
    //Output: The maximum amount of money that can be picked up
    $F[0] \leftarrow 0$;   $F[1] \leftarrow C[1]$
    for $i \leftarrow 2$ to $n$ do
        $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
    return $F[n]$
```

$F[0] = 0$, $F[1] = c_1 = 5$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

$F[4] = \max\{10 + 5, 7\} = 15$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15		

$F[5] = \max\{6 + 7, 15\} = 15$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	

$F[6] = \max\{2 + 15, 15\} = 17$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	**17**

Solving the coin-row problem by dynamic programming for the coin row 5, 1, 2, 10, 6, 2.

5- Dynamic Programming

- **What does dynamic programming have in common with divide-and-conquer? What is a principal difference between them?**

- **Solve the instance 5, 1, 2, 10, 6 of the coin-row problem.**

5- Dynamic Programming - Warshall’s and Floyd’s Algorithms

- Apply Warshall’s algorithm to find the transitive closure of the digraph de-fined by the following adjacency matrix:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- Prove that the time efficiency of Warshall’s algorithm is cubic.

5- Dynamic Programming - Warshall’s and Floyd’s Algorithms

- Solve the all-pairs shortest-path problem for the digraph with the following weight matrix:

$$\begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$