The Design and Analysis of Algorithm

Dynamic Programming

Instructor: Sadullah Karimi
M.Sc. in CSE

An idea, like a ghost . . . must be spoken to a little before it will

explain itself.

—Charles Dickens (1812–1870)

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.
- Examples of this technique’s applications.
- Numerous other applications range from the optimal way of breaking text into lines to image resizing to a variety of applications to sophisticated engineering

problems.

- **Richard Bellman called it the *principle of optimality***

Three Basic Examples

- **EXAMPLE 1** *Coin-rowproblem* There is a row of *n* coins whose values are some positive integers *c*1, *c*2, . . . , *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.

$$F(n) = \max\{c_n + F(n-2), F(n-1)\} \quad \text{for } n > 1, \qquad (8.3)$$
$$F(0) = 0, \qquad F(1) = c_1.$$

Algorithm (pseudocode)

- **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** $F[n]$
- The application of the algorithm to the coin row of denominations 5, 1, 2, 10,
- 6, 2 is shown in Figure 8.1. It yields the maximum amount of 17. It is worth pointing

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

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**

FIGURE 8.1 Solving the coin-row problem by dynamic programming for the coin row 5, 1, 2, 10, 6, 2.

Change-making problem

EXAMPLE 2 Change-making problem Consider the general instance of the following well-known problem.

Give change for amount n using the minimum number of coins of denominations d1<d2 < . . .<dm. For the coin denominations used in the United States, as for those used in most if not all other countries, there is a very simple and efficient algorithm discussed in the next chapter.

Here, we consider a dynamic programming algorithm for the general case, assuming availability of unlimited quantities of coins for each of the m denominations d1< d2 < . . . < dm where d1 = 1.

$$\begin{aligned} F(n) &= \min_{j:n\geq d_j} \{F(n-d_j)\} + 1 \quad \text{for } n > 0, \\ F(0) &= 0. \end{aligned} \tag{8.4}$$

Algorithm (pseudocode)

- **ALGORITHM** *ChangeMaking*($D[1..m]$, n)
- //Applies dynamic programming to find the minimum number of coins
- //of denominations $d1 < d2 < \dots < dm$ where $d1 = 1$ that add up to a
- //given amount n
- //Input: Positive integer n and array $D[1..m]$ of increasing positive
- // integers indicating the coin denominations where $D[1] = 1$
- //Output: The minimum number of coins that add up to n
- $F[0] \leftarrow 0$
- **for** $i \leftarrow 1$ **to** n **do**
- $temp \leftarrow \infty$; $j \leftarrow 1$
- **while** $j \le m$ **and** $i \ge D[j]$ **do**
- $temp \leftarrow \min(F[i - D[j]], temp)$
- $j \leftarrow j + 1$
- $F[i] \leftarrow temp + 1$
- **return** $F[n]$
- **The time and space efficiencies of the algorithm are obviously *O(nm)* and *(n),* respectively**.

$$F[0] = 0$$

n	0	1	2	3	4	5	6
F	0						

$$F[1] = \min\{F[1-1]\} + 1 = 1$$

n	0	1	2	3	4	5	6
F	0	1					

$$F[2] = \min\{F[2-1]\} + 1 = 2$$

n	0	1	2	3	4	5	6
F	0	1	2				

$$F[3] = \min\{F[3-1], F[3-3]\} + 1 = 1$$

n	0	1	2	3	4	5	6
F	0	1	2	1			

$$F[4] = \min\{F[4-1], F[4-3], F[4-4]\} + 1 = 1$$

n	0	1	2	3	4	5	6
F	0	1	2	1	1		

$$F[5] = \min\{F[5-1], F[5-3], F[5-4]\} + 1 = 2$$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	

$$F[6] = \min\{F[6-1], F[6-3], F[6-4]\} + 1 = 2$$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	**2**

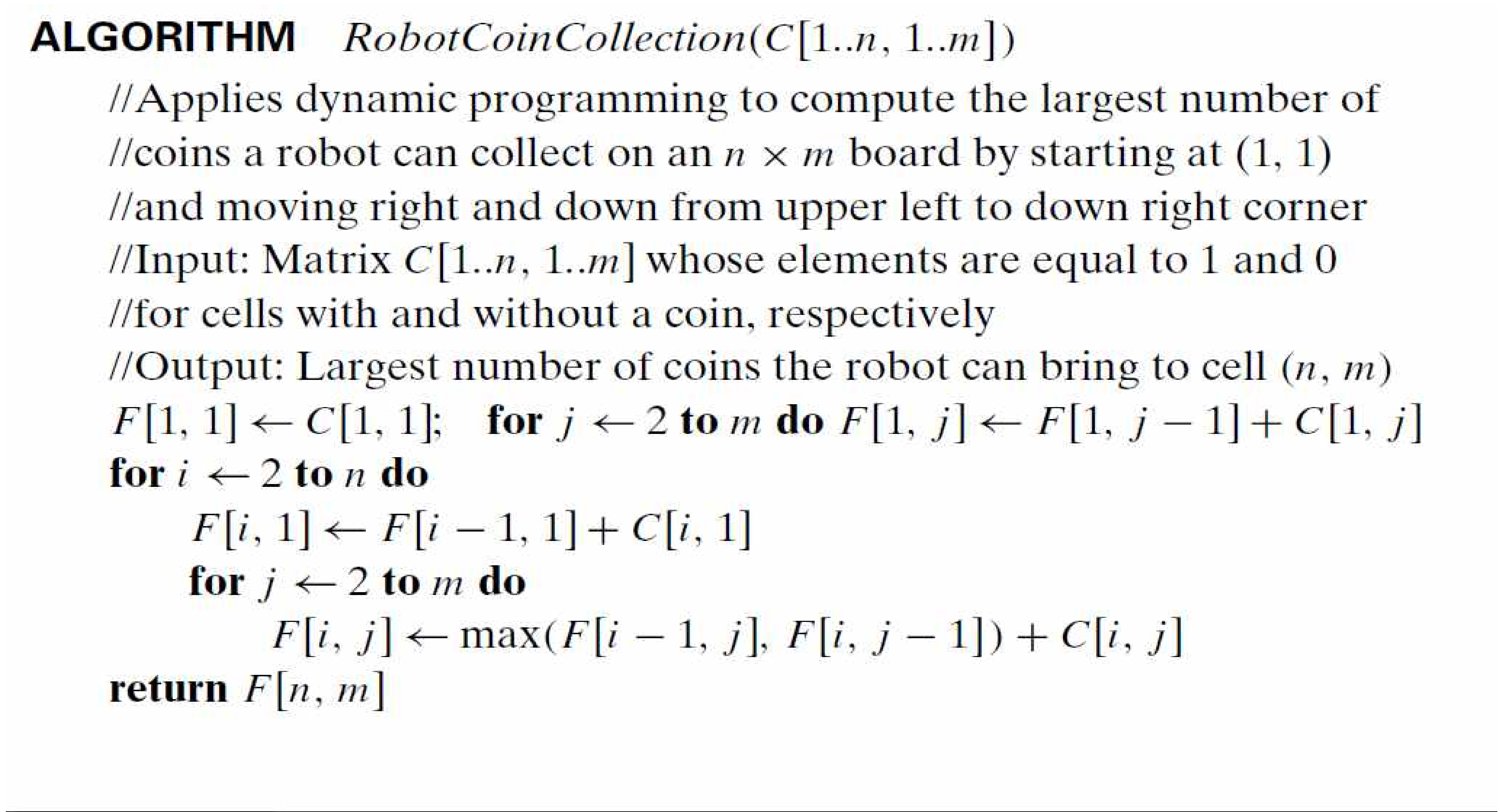
FIGURE 8.2 Application of Algorithm *MinCoinChange* to amount $n = 6$ and coin denominations 1, 3, and 4.

Coin-collecting problem

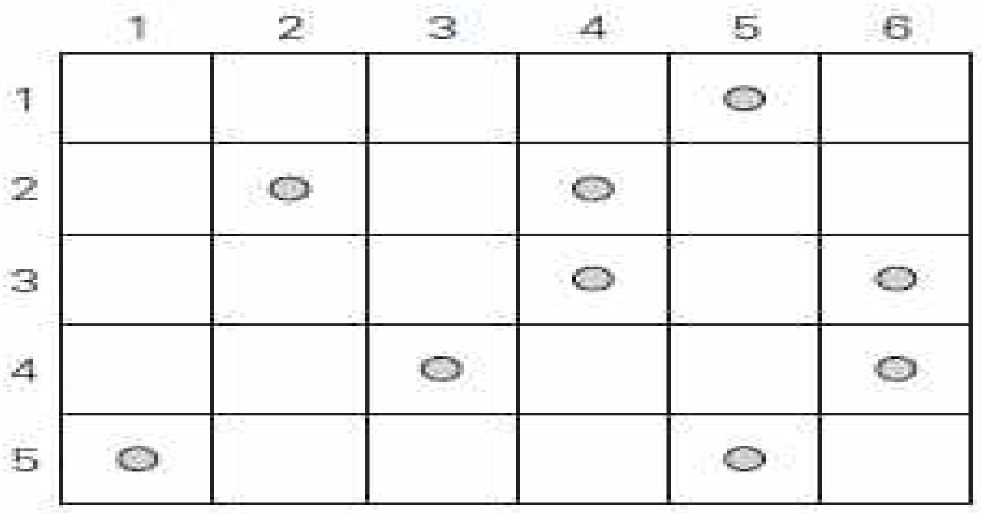
- EXAMPLE 3 Coin-collecting problem Several coins are placed in cells of an n × m board, no more than one coin per cell.
- A robot, located in the upper left cell of the board, needs to collect as many of the coins as possible and bring them to the bottom right cell.
- On each step, the robot can move either one cell to the right or one cell down from its current location.
- When the robot visits a cell with a coin, it always picks up that coin. Design an algorithm to find the maximum number of coins the robot can collect and a path it needs to follow to do this.
- Let F(i, j) be the largest number of coins the robot can collect and bring to the cell (i, j) in the ith row and jth column of the board.
- It can reach this cell either from the adjacent cell (i − 1, j) above it or from the adjacent cell (i, j − 1) to the left of it. The largest numbers of coins that can be brought to these cells are F(i − 1, j) and F(i, j − 1), respectively

$$\begin{aligned} &F(i, j) = \max\{F(i-1, j), F(i, j-1)\} + c_{ij} \quad \text{for } 1 \le i \le n, \ \ 1 \le j \le m \\ &F(0, j) = 0 \ \text{ for } 1 \le j \le m \quad \text{and} \quad F(i, 0) = 0 \ \text{ for } 1 \le i \le n, \end{aligned} \qquad \textbf{(8.5)}$$

Algorithm (pseudocode)



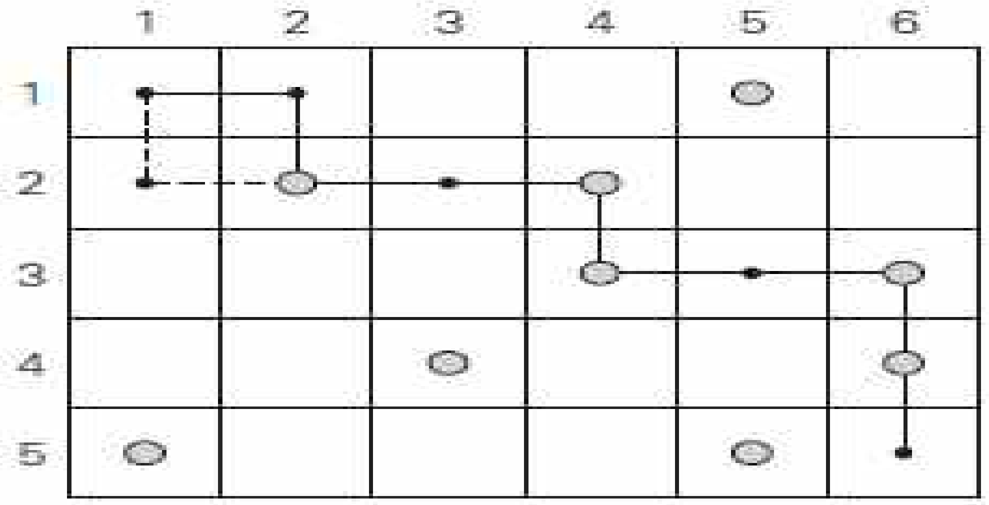
```
ALGORITHM RobotCoinCollection($C[1..n, 1..m]$)
    //Applies dynamic programming to compute the largest number of
    //coins a robot can collect on an $n \times m$ board by starting at (1, 1)
    //and moving right and down from upper left to down right corner
    //Input: Matrix $C[1..n, 1..m]$ whose elements are equal to 1 and 0
    //for cells with and without a coin, respectively
    //Output: Largest number of coins the robot can bring to cell $(n, m)$
    $F[1, 1] \leftarrow C[1, 1]$;   for $j \leftarrow 2$ to $m$ do $F[1, j] \leftarrow F[1, j-1] + C[1, j]$
    for $i \leftarrow 2$ to $n$ do
        $F[i, 1] \leftarrow F[i-1, 1] + C[i, 1]$
        for $j \leftarrow 2$ to $m$ do
            $F[i, j] \leftarrow \max(F[i-1, j], F[i, j-1]) + C[i, j]$
    return $F[n, m]$
```



(a)

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

(b)



(c)

FIGURE 8.3 (a) Coins to collect. (b) Dynamic programming algorithm results. (c) Two paths to collect 5 coins, the maximum number of coins possible.

Questions:

?