Example 2: Coin-row problem

There is a row of *n* coins whose values are some positive integers $c_1, c_2, \ldots, c_n$, 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.

E.g.: 5, 1, 2, 10, 6, 2. What is the best selection?

DP solution to the coin-row problem

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 without last coin – the max amount is ?

those with the last coin -- the max amount is ?

Thus we have the following recurrence

$F(n) = \max\{c_n + F(n-2), F(n-1)\}$ for $n > 1$,

$F(0) = 0$, $F(1)=c_1$

DP solution to the coin-row problem (cont.)

$F(n) = \max\{c_n + F(n-2),\ F(n-1)\}$ for $n > 1$,

$F(0) = 0,\ F(1)=c_1$

Index	0	1	2	3	4	5	6
Coins	--	5	1	2	10	6	2
F( )							

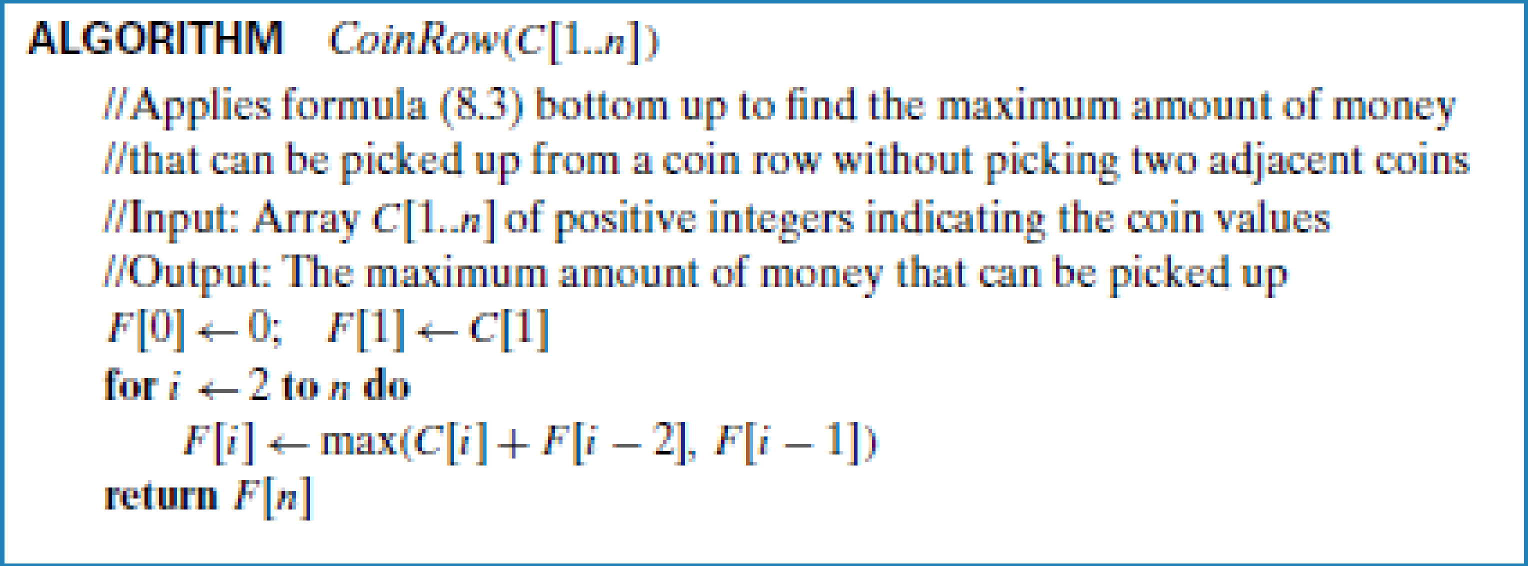
Max amount:

Coins of optimal solution:

Time efficiency:

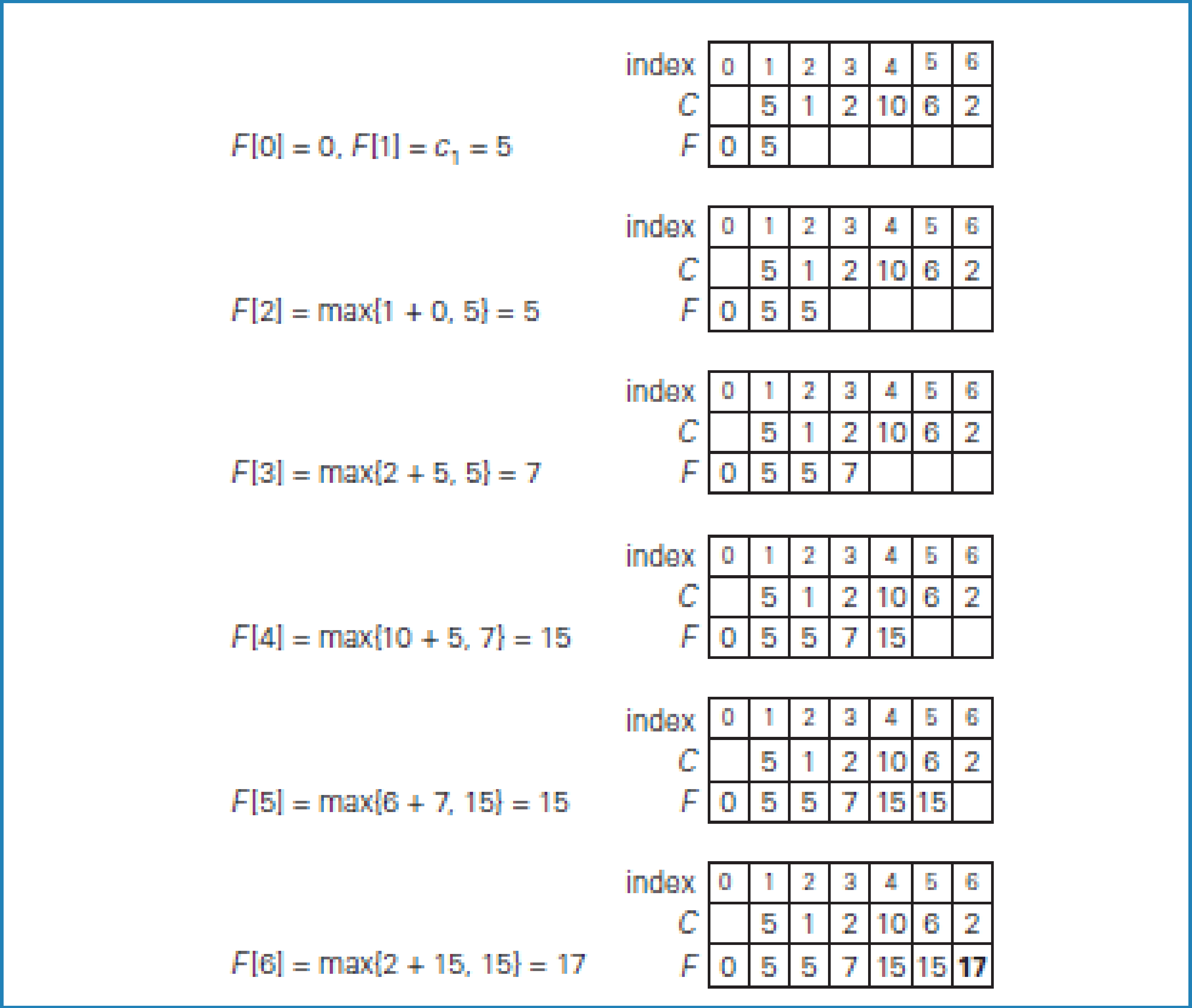
Space efficiency:

Algorithm



```
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 $F[n]$
```

Example



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

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		

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

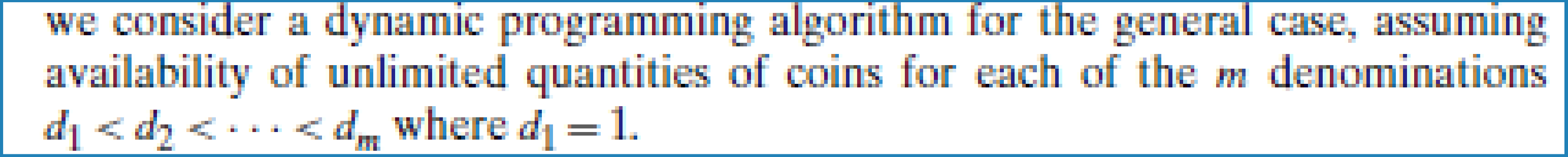
index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	

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

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

Coin change

- Give change for amount n using the minimum
- number of coins of denominations $d_1 < d_2 < \cdots < d_m$
- . For the coin denominations

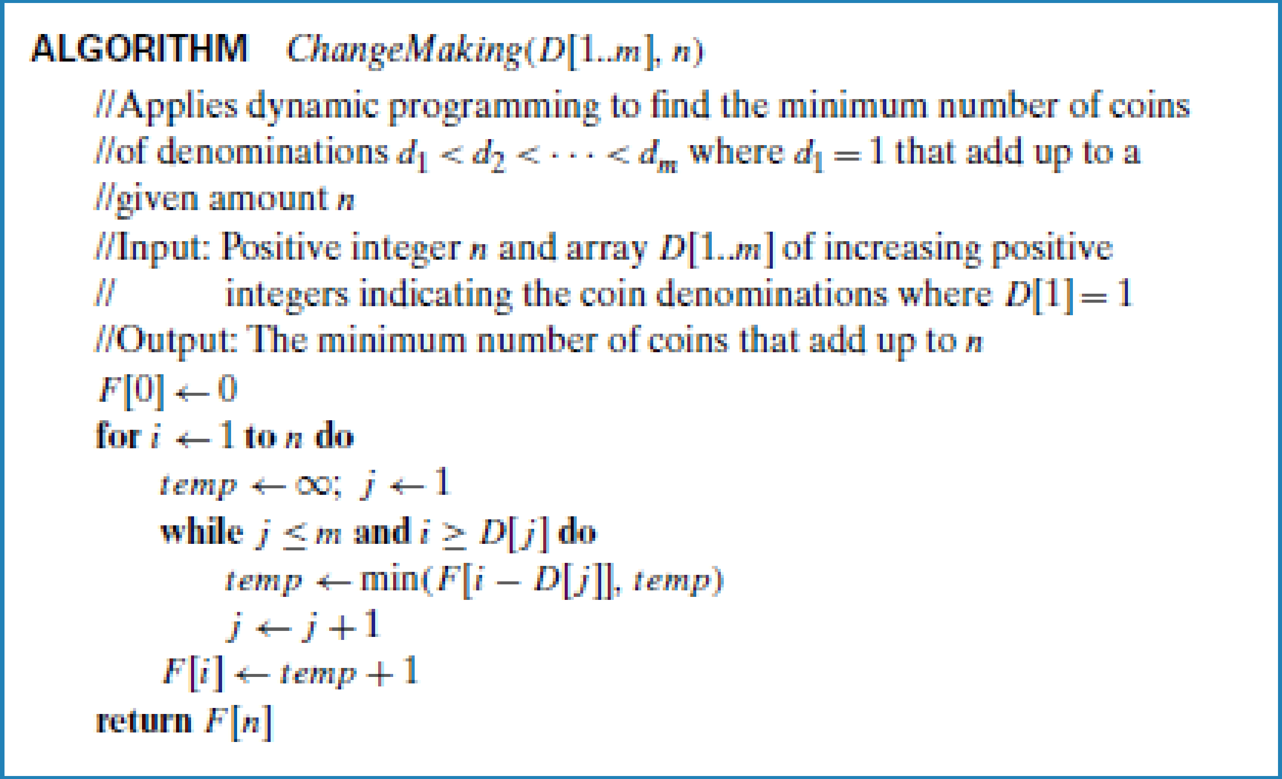


we consider a dynamic programming algorithm for the general case, assuming availability of unlimited quantities of coins for each of the m denominations $d_1 < d_2 < \cdots < d_m$ where $d_1 = 1$.

$$F(n) = \min_{j:\, n \geq d_j} \{F(n - d_j)\} + 1 \quad \text{for } n > 0,$$

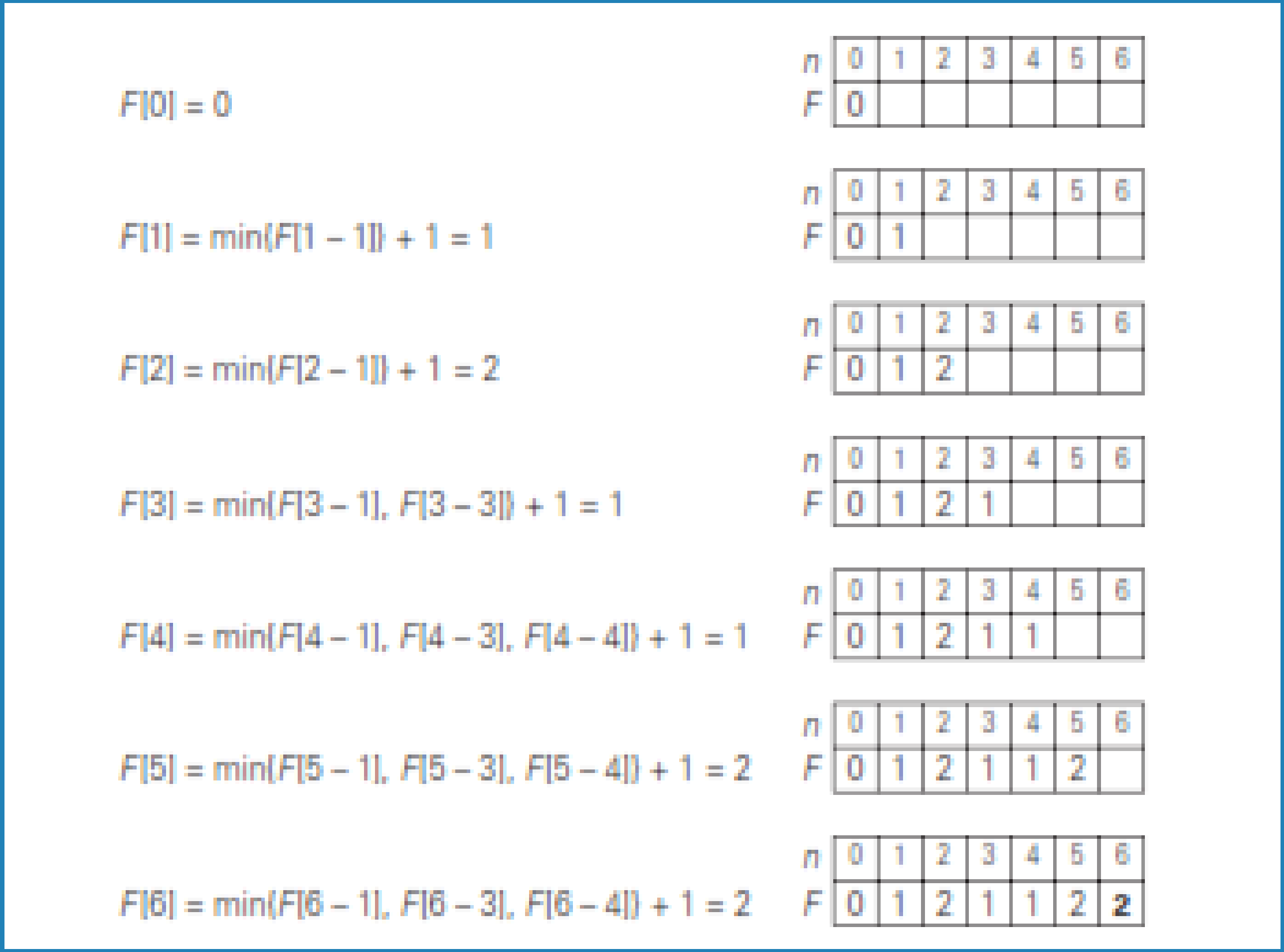
$$F(0) = 0.$$

Algorithm



```
ALGORITHM  ChangeMaking(D[1..m], n)
    //Applies dynamic programming to find the minimum number of coins
    //of denominations $d_1 < d_2 < \cdots < d_m$ where $d_1 = 1$ 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 $F[n]$
```

1,3,4 for change 6 & 1,2,3 n=5



$F[0] = 0$

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**