Dynamic Programming

General idea

- 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.

Review (Fibonacci numbers)

- $F(n) = F(n-1) + F(n-2)$ for $n > 1$
- $F(0) = 0$, $F(1) = 1$.

Review Knapsack Problem

$$F(i, j) = \begin{cases} \max\{F(i-1, j),\ v_i + F(i-1, j-w_i)\} & \text{if } j - w_i \geq 0, \\ F(i-1, j) & \text{if } j - w_i < 0. \end{cases}$$

$$F(0, j) = 0 \text{ for } j \geq 0 \quad \text{and} \quad F(i, 0) = 0 \text{ for } i \geq 0.$$

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.

Example

- 5, 1, 2, 10, 6, 2.

General Idea

- 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.
- The largest amount we can get from the first group is equal to $c_n + F(n - 2)$—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 $F(n)$.

Recurrence

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

Solve example with DP

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

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-collecting problem

- Several coins are placed in cells of an $n \times m$ 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.

General Idea

- 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

recourence

$$F(i, j) = \max\{F(i-1, j), F(i, j-1)\} + c_{ij} \quad \text{for } 1 \le i \le n, \ 1 \le j \le m$$

$$F(0, j) = 0 \ \text{ for } 1 \le j \le m \quad \text{and} \quad F(i, 0) = 0 \ \text{ for } 1 \le i \le n,$$

where $c_{ij} = 1$ if there is a coin in cell (i, j), and $c_{ij} = 0$ otherwise.

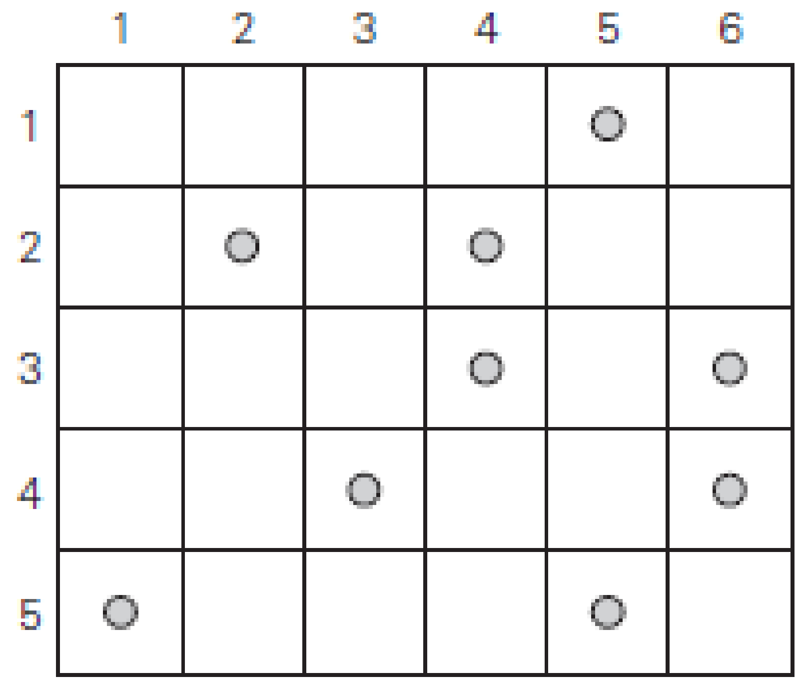
PseudoCode

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)

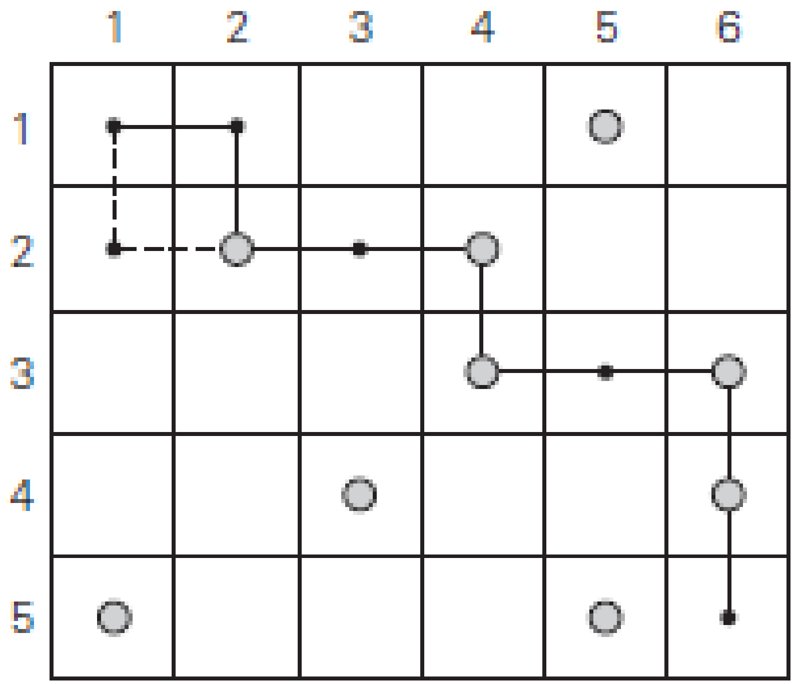
$F[1, 1] \leftarrow C[1, 1]$; **for** $j \leftarrow 2$ **to** m **do** $F[1, j] \leftarrow F[1, j-1] + C[1, j]$
for $i \leftarrow 2$ **to** n **do**
  $F[i, 1] \leftarrow F[i-1, 1] + C[i, 1]$
  for $j \leftarrow 2$ **to** m **do**
    $F[i, j] \leftarrow \max(F[i-1, j], F[i, j-1]) + C[i, j]$
return $F[n, m]$

Example



Solution

	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**



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 $d1<d2 < \ldots <dm$.
- Here, we consider a dynamic programming algorithm for the general case, assuming availability of unlimited quantities of coins for each of the m denominations
- $d1< d2 < \ldots < dm$ where $d1 = 1$.
- Let $F(n)$ be the minimum number of coins whose values add up to n; it is convenient to define $F(0) = 0$. The amount n can only be obtained by adding one coin of denomination dj to the amount $n - dj$ for $j = 1, 2, \ldots, m$ such that $n \geq dj$.
- Therefore, we can consider all such denominations and select the one minimizing $F(n - dj) + 1$. Since 1 is a constant, we can, of course, find the smallest $F(n - dj)$ first and then add 1 to it.

recurrence

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

$$F(0) = 0.$$

pseudocode

```
ALGORITHM ChangeMaking(D[1..m], n)
    //Applies dynamic programming to find the minimum number of coins
    //of denominations d_1 < d_2 < ··· < 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] ← 0
    for i ← 1 to n do
        temp ← ∞; j ← 1
        while j ≤ m and i ≥ D[j] do
            temp ← min(F[i − D[j]], temp)
            j ← j + 1
        F[i] ← temp + 1
    return F[n]
```

example

$F[0] = 0$

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**