# CSC 411
# Design and Analysis of Algorithms

---

## Chapter 8 - Part 1

Dynamic Programming

Instructor: Minhee Jun

# Dynamic Programming

- Dynamic Programming is a general algorithm design technique for solving problems defined by recurrences with overlapping subproblems

- Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems and later assimilated by CS

- "Programming" here means "planning"

- Main idea:
  - set up a recurrence relating a solution to a larger instance to solutions of some smaller instances
  - solve smaller instances ***once***
  - record solutions in a table
  - extract solution to the initial instance from that table

# Examples of DP algorithms

- Fibonaccl numbers problem
- Three basic examples
  - Coin row problem
  - Change-Making problem
  - Coin-collecting problem
- Knapsack problem & memory functions
- Optimal binary search tree
- Warshall’s and Floyd’s Algorithms

# Example: Fibonacci numbers

- Recall definition of Fibonacci numbers:

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

  $F(0) = 0$

  $F(1) = 1$

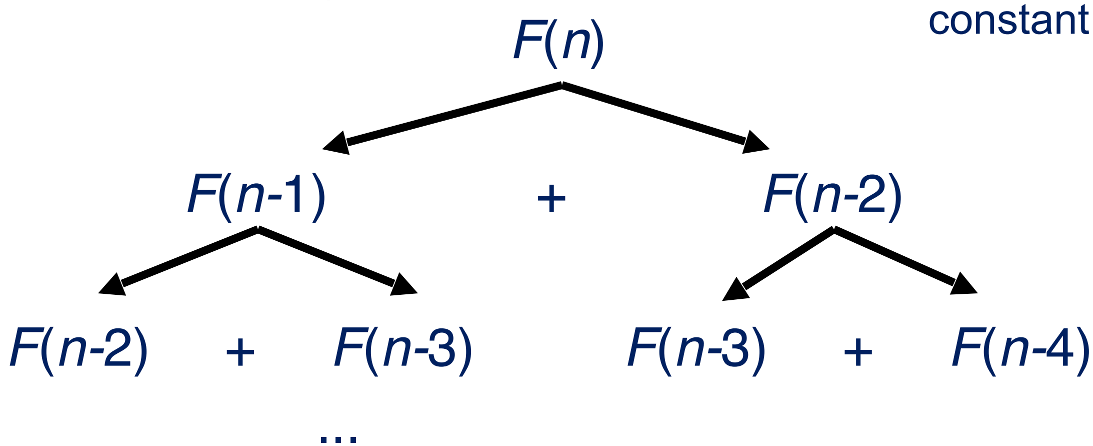
- Computing the $n^{th}$ Fibonacci number recursively (top-down):

**ALGORITHM** $F(n)$

//Computes the $n$th Fibonacci number recursively by using its definition
//Input: A nonnegative integer $n$
//Output: The $n$th Fibonacci number
**if** $n \le 1$ **return** $n$
**else return** $F(n-1) + F(n-2)$

Time Efficiency?

Linear second-order recurrences with constant coefficient (Appendix B)



# Example: compute F(5)

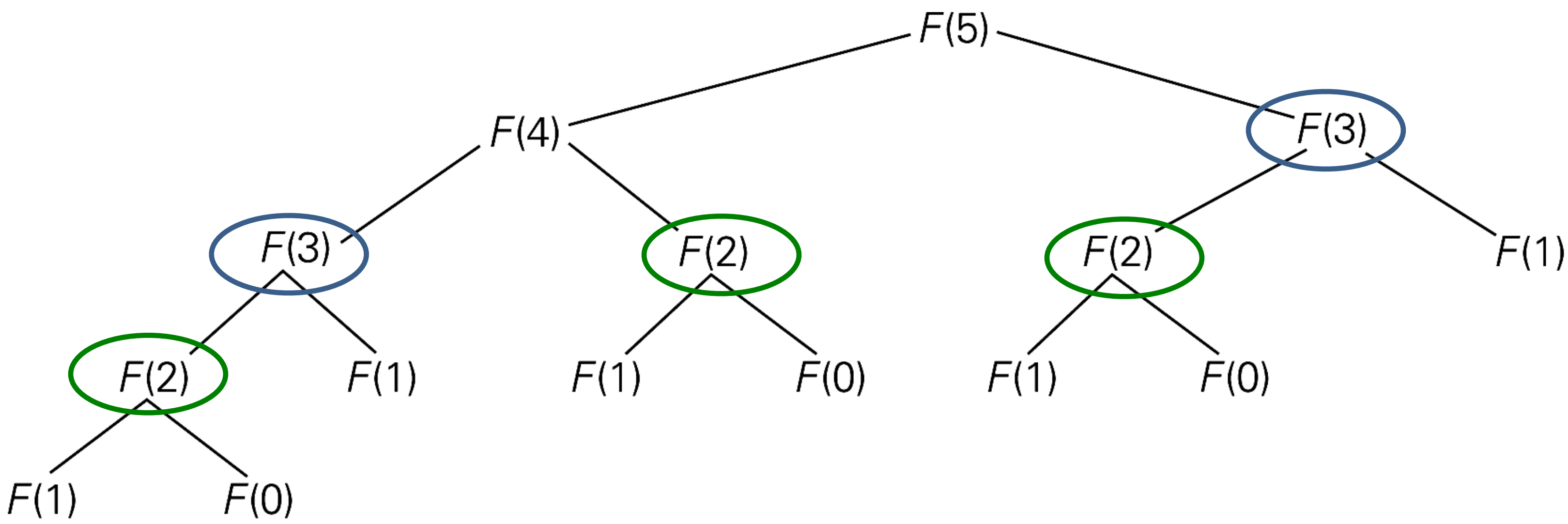



**FIGURE 2.6** Tree of recursive calls for computing the 5th Fibonacci number by the definition-based algorithm

# Example: Fibonacci numbers (cont.)

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

$F(0) = 0$

$F(1) = 1$

$F(2) = 1+0 = 1$

…

$F(n\text{-}2) =$

$F(n\text{-}1) =$

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

**ALGORITHM** *Fib*($n$)

```
//Computes the nth Fibonacci number iteratively by using its definition
//Input: A nonnegative integer n
//Output: The nth Fibonacci number
F[0] ← 0; F[1] ← 1
for i ← 2 to n do
    F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

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

Efficiency:

- time: O(n)
- space: one dimensional array with size n

See section 2.5 for a single-loop pseudocode

# Example: Computing a binomial coefficient

Binomial coefficients are coefficients of the binomial formula: $(a+b)^n$

$$(a+b)^n = C(n,0)a^n b^0 + \cdots + C(n,k)a^{n-k}b^k + \cdots + C(n,n)a^0 b^n$$

- Find $C(n,k)$ **for any given *n and k*.** How?
- Find the recurrence

$$C(n,k) = C(n-1,\ k-1) + C(n-1,\ k) \text{ for } n > k > 0$$

$$C(n,0) = 1, \quad C(n,n) = 1 \text{ for } n \geq 0$$

Value of $C(n,k)$ can be computed by filling a table:

| | 0 | 1 | 2 | . . . | *k*-1 | *k* |
|---|---|---|---|---|---|---|
| 0 | 1 | | | | | |
| 1 | 1 | 1 | | | | |
| 2 | | 2 | | | | |
| . | | | | | | |
| . | | | | | | |
| *n*-1 | | | | | $C(n\text{-}1, k\text{-}1)$ | $C(n\text{-}1, k)$ |
| *n* | | | | | | $C(n, k)$ |

# Computing *C*(*n*,*k*): pseudocode and analysis

**ALGORITHM** *Binomial*$(n, k)$
//Computes $C(n, k)$ by the dynamic programming algorithm
//Input: A pair of nonnegative integers $n \geq k \geq 0$
//Output: The value of $C(n, k)$
**for** $i \leftarrow 0$ **to** $n$ **do**
  **for** $j \leftarrow 0$ **to** $\min(i, k)$ **do**
    **if** $j = 0$ **or** $j = i$
      $C[i, j] \leftarrow 1$
    $\boxed{\textbf{else } C[i, j] \leftarrow C[i-1, j-1] + C[i-1, j]}$
**return** $C[n, k]$

- Time efficiency: $\Theta(nk)$
- Space efficiency: $\Theta(nk)$

# Example: Coin-row Problem by DP

- There are a row of $n$ coins whose values are some positive integers $c_1, c_2, \cdots, c_n$, not necessarily distinct.
  - The goal is to pick up the maximum amount of money
  - Constraint: no two coins adjacent in the initial row can be picked up.
  - $F(n)$: the maximum amount that can be picked up from the row of $n$ coins.

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

# Coin-row Problem by DP: 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** $F[n]$

- Efficiency: $\Theta(n)$

# Coin-row Problem by DP: Example

- The coin row: 5, 1, 2, 10, 6, 2

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

| 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 | | | |

# Coin-row Problem by DP: Example

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

| 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** |

# Change-making Problem by DP

- Give change for amount $n$ using the minimum # coins of denominations $d_1 < d_2 < \cdots < d_m$, where $d_1 = 1$.
  - $F(n)$: the minimum # coins whose values add up to $n$
  - $F(0) = 0$

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

$$F(0) = 0.$$

# Change-making Problem by DP: 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]
```

- Time efficiency: $O(nm)$

# Change-making Problem by DP: Example

- Amount n =6, and coin denominations 1, 3, 4

$F[0] = 0$

| $n$ | 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 | | | |

# Change-making Problem by DP: Example

- Amount n =6, and coin denominations 1, 3, 4

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

| $n$ | 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** |

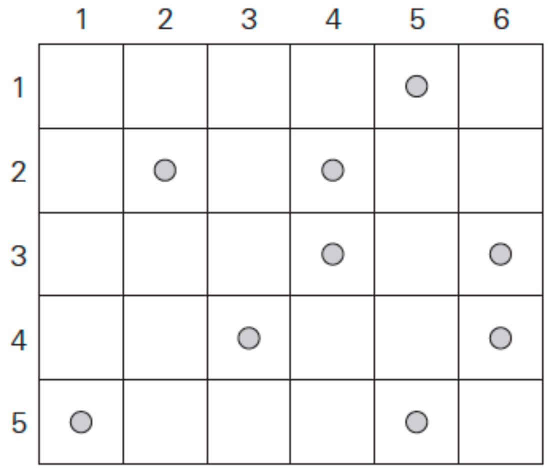
# Coin-collecting Problem by DP

- Several coins are placed in cells of an $n$ by $m$ 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 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.
  - $F(i, j)$: the largest # coins the robot can collect and bring to the cell $(i, j)$ in the $i$th row and $j$th column of the board.

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

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

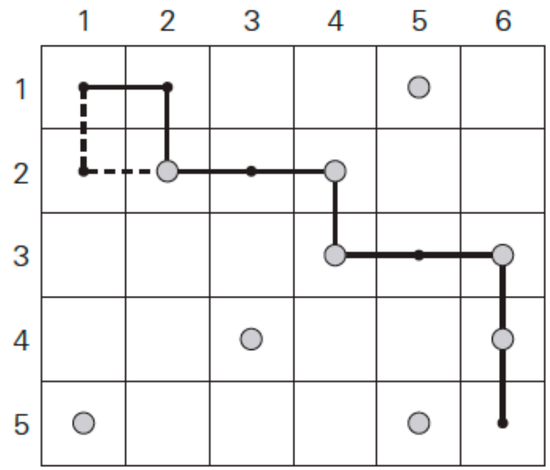
# Coin-collecting Problem by DP: Example




(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)

# Coin-collecting Problem by DP: 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** $F[n, m]$

- Time efficiency: $\Theta(nm)$
- Space efficiency: $\Theta(nm)$

# Exercise 8.1