CPE 231 Algorithms

# Dynamic Programming

Dr. Taweechai Nuntawisuttiwong

# Contents

# Dynamic Programming

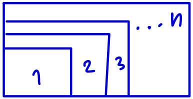
# Dynamic Programming

**Definition**

Dynamic Programming (DP) is a technique for solving problems with overlapping subproblems and optimal substructure.

- Introduced by Richard Bellman in the 1950s as a method for optimizing multistage decision processes.
- The term "programming" refers to "planning" rather than computer coding.

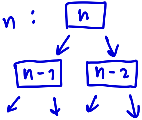
# Key Concepts of Dynamic Programming



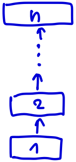

DP $\neq$ recursive fn.
(direct)

- **Overlapping Subproblems**: Solving the same subproblems repeatedly in naive recursive solutions.
- **Memoization**: Storing solutions to subproblems to avoid redundant calculations. recursive but not direct recursive
- Two Approaches:
  - **Top-down** (Memoization): Recursive approach with caching.
  - **Bottom-up** (Tabulation): Iterative approach by solving all smaller subproblems first.

Top-down :




Bottom-up :

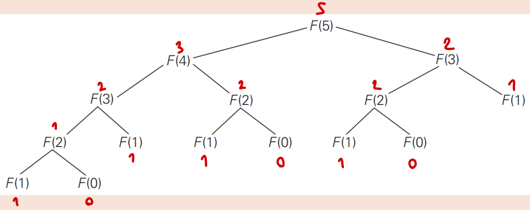


# Example - Fibonacci Sequence

Fibonacci Recurrence: $F(n) = F(n-1) + F(n-2), \; n > 1.$
Initial Conditions: $F(0) = \cancel{1}, \; F(1) = 1.$
0

Top-down

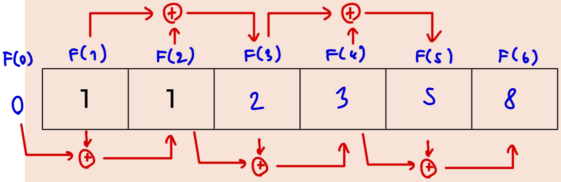
**Use recurrence directly**




15 ครั้ง

Bottom-Up

**Use dynamic programming**




4 ครั้ง

# Applications and Principle of Optimality

- **Principle of Optimality**: Solutions to subproblems contribute to the overall optimal solution.
- **Applications**:
  - Optimization problems like shortest paths in graphs, knapsack, matrix chain multiplication.

    shortest path, TSP, VRP
  - Real-world examples: Text formatting, image resizing, and engineering optimizations.
- Dynamic Programming is a powerful strategy for solving complex optimization problems efficiently.

TSP $\longrightarrow$ n cities $\longrightarrow$ feasible solution n!

DP $\longrightarrow$ n! memory unit

# Three Basic Examples

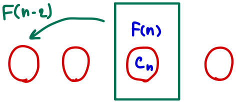
# Coin-Row Problem

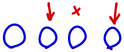
- Maximize the amount collected from a row of coins without picking two adjacent coins.
  - Given a row of $n$ coins, each with a positive integer value $c_i$. มีค่าเท่ากันได้
- Define $F(n)$ as the maximum amount collectible from the first $n$ coins.

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

หยิบเหรียญที่ n ไปรวมกับค่าของเหรียญก่อนหน้า 2 ครั้ง

  - Base cases: $F(0) = 0$, $F(1) = c_1$.
  - With last coin: Add $c_n$ and skip the adjacent coin.
  - Without last coin: Take the solution from $F(n-1)$.
- Complexity
  - Time: $O(n)$
  - Space: $O(n)$



# Coin-Row Problem

**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
$F[0] \leftarrow 0; \quad F[1] \leftarrow C[1]$
**for** $i \leftarrow 2$ **to** $n$ **do**
    $F[i] \leftarrow \max(C[i] + F[i-2],\ F[i-1])$
**return** $F[n]$

# Coin-Row Problem

## Example: Coin values [5, 1, 2, 10, 6, 2]

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

Base case

$F[0] = 0$ , $F(1) = 5$

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| C | + | 5 | 1 | 2 | 10 | 6 | 2 |
| F | 0 | 5 | 5 | | | | |

1

$$F[2] = \max\{\overset{c_2}{1}+\overset{F(0)}{0}, \overset{F(1)}{5}\} = 5$$

# Coin-Row Problem

## Example: Coin values [5, 1, 2, 10, 6, 2]

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

$$F[3] = \max\{\underbrace{2}_{c_3}+\underbrace{5}_{F(1)}, \underbrace{5}_{F(2)}\} = 7$$

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

$$F[4] = \max\{\underbrace{10}_{c_4}+\underbrace{5}_{F(2)}, \underbrace{7}_{F(3)}\} = 15$$

# Coin-Row Problem

## Example: Coin values [5, 1, 2, 10, 6, 2]

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

$$F[5] = \max\{\overset{C(5)}{6}+\overset{F(3)}{7}, \underset{F(4)}{15}\} = 15$$

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

$$F[6] = \max\{\overset{C(6)}{2}+\overset{F(4)}{15}, \underset{F(5)}{15}\} = 17$$

# Change-Making Problem

- Use the minimum number of coins to make a given amount $n$.
  - Given a target amount $n$ and denominations $d_1 < d_2 < \ldots < d_m$.
- Define $F(n)$ as the minimum number of coins to make $n$.

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

จำนวนเงิน ค่าของเหรียญ

  - Base case: $F(0) = 0$.
- Complexity
  - Time: $O(nm)$
  - Space: $O(n)$

# Change-Making Problem

**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] ← 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]
```

# Change-Making Problem

## Example: amount $n$ = 6 and denominations 1, 3, 4.

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

$$F[0] = 0$$

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

$$F[1] = \min\{F(1-\overset{1}{\cancel{d_j}})\} + 1 = F(0)+1 = 1$$

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

$$F[2] = \min\{F(2-\overset{1}{\cancel{d_j}})\} + 1 = F(1)+1 = 2$$

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

$$F[3] = \min\{F(3-\overset{3,1}{\cancel{d_j}})\} + 1 = \min\{F(0), F(2)\} + 1$$
$$= 1$$

# Change-Making Problem

## Example: amount $n$ = 6 and denominations 1, 3, 4.

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

$$F[4] = \min\{F(4-\overset{1,3,4}{\cancel{d_j}})\}+1 = \min\{F(0), F(1), F(3)\}+1$$
$$= F(0)+1 = 1$$

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

$$F[5] = \min\{F(5-\overset{1,3,4}{\cancel{d_j}})\}+1 = \min\{F(1), F(2), F(4)\}+1$$
$$= F(4)+1 = 2$$

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

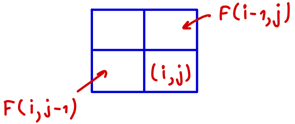
$$F[6] = \min\{F(6-\overset{1,3,4}{\cancel{d_j}})\}+1 = \min\{F(2), F(3), F(5)\}+1$$
$$= F(3)+1 = 2$$

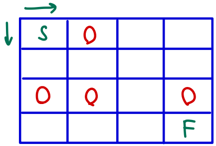
# Coin-Collecting Problem

- Collect the maximum number of coins on an $n \times m$ grid by moving only right or down.
  - A grid with coins in specific cells (either 0 or 1).
- Define $F(i, j)$ as the maximum coins collectible to reach cell $(i, j)$.

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

  - Base cases: $F(0, j) = 0$ for $1 \le j \le m$, $F(i, 0) = 0$ for $1 \le i \le n$.
  - The robot moves from the cell above or left.
- Complexity
  - Time: $O(nm)$
  - Space: $O(nm)$



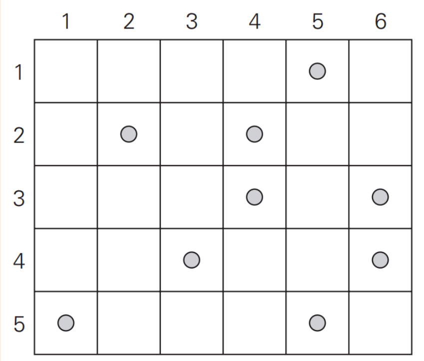



# Coin-Collecting Problem

```
ALGORITHM  RobotCoinCollection(C[1..n, 1..m])
    //Applies dynamic programming to compute the largest number of
    //coins a robot can collect on an n × 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] ← C[1, 1];   for j ← 2 to m do F[1, j] ← F[1, j − 1] + C[1, j]
    for i ← 2 to n do
        F[i, 1] ← F[i − 1, 1] + C[i, 1]
        for j ← 2 to m do
            F[i, j] ← max(F[i − 1, j], F[i, j − 1]) + C[i, j]
    return F[n, m]
```

# Coin-Collecting Problem

## Example:




| | 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 |

Solution

# Coin-Collecting Problem

## Example:

<u>Tracing</u> :

1) $F(i-1,j) > F(i,j-1)$

บน > ซ้าย

$\downarrow$ $(i,j)$

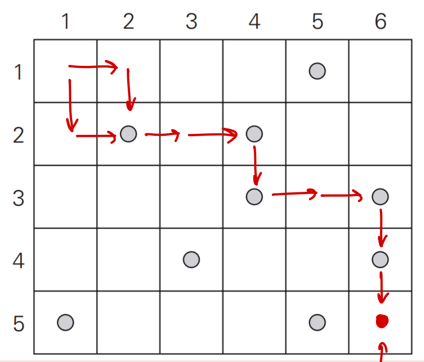
2) $F(i-1,j) < F(i,j-1)$

บน < ซ้าย $\longrightarrow (i,j)$

3) $F(i-1,j) = F(i,j-1)$

$\downarrow$

$\longrightarrow (i,j)$




Start tracing

# The Knapsack Problem and Memory Functions

# Knapsack Problem

- Find the subset of items that maximizes value without exceeding the capacity.
  - Given $n$ items with weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$, and a knapsack of capacity $W$.
- Define $F(i, j)$: Maximum value achievable with the first $i$ items and capacity $j$.

ไม่เพิ่ม        เพิ่ม item i

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

  - Base cases: $F(0, j) = 0$ for $j \geq 0$, $F(i, 0) = 0$ for $i \geq 0$.
  - Case 1: If the item fits, compare the value including vs. excluding it.
  - Case 2: If the item does not fit, skip it.
- Complexity
  - Time: $O(nW)$
  - Space: $O(nW)$
  - Time (find the compositional of an optimal solution): $O(n)$

# Knapsack Problem

**Example: Consider the instance given by the following data**

capacity $W = 5.$

| item | weight | value |
|---|---|---|
| 1 | 2 | $12 |
| 2 | 1 | $10 |
| 3 | 3 | $20 |
| 4 | 2 | $15 |

# Knapsack Problem

→ solution path

## Example:

$$F(i,j) = \begin{cases} \max\{\overset{\text{ไม่เพิ่ม}}{F(i-1,j)}, \overset{\text{เพิ่ม item } i}{v_i + F(i-1, j-w_i)}\} & \text{if } j - w_i \geq 0, \text{ น้ำหนักไม่เกิน} \\ F(i-1,j) \text{ ไม่ใส่ของเพิ่ม} & \text{if } j - w_i < 0. \text{ น้ำหนักเกิน} \end{cases}$$

item : $S = \{1, 2, 4\}$

**capacity $j$**

F(2-1,2-1) + 10    F(3-1,2-1)

| | i | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $w_1 = 2$ $v_1 = 12$ | 1 | 0 | $\boxed{0}$ | $\boxed{12}$ | 12 | 12 | 12 |
| $w_2 = 1$ $v_1 = 10$ | 2 | 0 | 10 | 12 | 22 | 22 | 22 |
| $w_3 = 3$ $v_1 = 20$ | 3 | 0 | 10 | 12 | 22 | 30 | 32 |
| $w_4 = 2$ $v_1 = 15$ | 4 | 0 | 10 | 15 | 25 | 30 | $\boxed{37}$ |

+10 +10 +20 +20 +15 +15

37 > 32

4 ∈ S

Solution

# Memory Functions

- Reduce unnecessary computations in dynamic programming by combining top-down and bottom-up methods.
- Memory Function Approach: Use a top-down recursion with memoization.
  - Check the table before calculating a subproblem.
  - Compute only if not already solved, then store in the table.
- Avoid solving unneeded subproblems and reduce recursive calls.

# Memory Functions

**ALGORITHM** $MFKnapsack(i, j)$

//Implements the memory function method for the knapsack problem
//Input: A nonnegative integer $i$ indicating the number of the first
//  items being considered and a nonnegative integer $j$ indicating
//  the knapsack capacity
//Output: The value of an optimal feasible subset of the first $i$ items
//Note: Uses as global variables input arrays $Weights[1..n]$, $Values[1..n]$,
//and table $F[0..n, 0..W]$ whose entries are initialized with $-1$'s except for
//row 0 and column 0 initialized with 0's

**if** $F[i, j] < 0$
  **if** $j < Weights[i]$
    $value \leftarrow MFKnapsack(i-1, j)$
  **else**
    $value \leftarrow \max(MFKnapsack(i-1, j),$
      $Values[i] + MFKnapsack(i-1, j - Weights[i]))$
  $F[i, j] \leftarrow value$
**return** $F[i, j]$

# Memory Functions

**Example:**

**capacity $j$**

| | i | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $w_1 = 2 \quad v_1 = 12$ | 1 | 0 | $\boxed{0}$ | $\boxed{12}$ | $\boxed{12}$ | $\boxed{12}$ | $\boxed{12}$ |
| $w_2 = 1 \quad v_1 = 10$ | 2 | 0 | | $\boxed{12}$ | $\boxed{22}$ | | $\boxed{22}$ |
| $w_3 = 3 \quad v_1 = 20$ | 3 | 0 | | | $\boxed{32}$ | | $\boxed{32}$ |
| $w_4 = 2 \quad v_1 = 15$ | 4 | 0 | | | | | $\boxed{37}$ |