

CCES – College of Computer Engineering and Sciences

Prince Sattam Bin Abdulaziz University

CS3401 – Algorithm Design and Analysis

Chapter 6

Dynamic Programming

Academic Year 2026 – 2027



Dr. Khalil Chebil

A white capital letter 'E' centered within a white circle.

Dr. Esraa Eldesouky

Chapter Overview

Introduction

Coin-Row Problem

Change-Making Problem

0-1 Knapsack

Optimal Binary Search Trees

Warshall's & Floyd's Algorithms

Coin-Collecting Problem

1

Section 1

Introduction

The Fibonacci Trap

Naive recursive Fibonacci:

- 1: `fib( $n$ ):`
- 2: `if  $n \leq 1$  then return  $n$`
- 3: `end if`
- 4: `return fib( $n-1$ ) + fib( $n-2$ )`

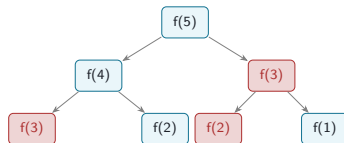
Recurrence: $T(n) = T(n-1) + T(n-2) + \mathcal{O}(1)$

$$T(n) \in \Theta(\varphi^n), \varphi \approx 1.618$$

Exponential time — `fib(50)` makes $\approx 10^{10}$ calls!

Root cause

The same sub-problem (e.g. `fib(10)`) is computed *exponentially many times*.



Recomputed!

Dynamic Programming — The Fix

Dynamic Programming (DP)

An algorithm design technique for solving problems with **overlapping sub-problems** by computing each sub-problem *once* and storing the result in a table. “Programming” here means *planning* (not coding).

Two approaches:

- ▶ **Tabulation (bottom-up)**: fill table from smallest sub-problems up to the target; iterative; no recursion overhead
- ▶ **Memoisation (top-down)**: recursive, but cache each result; compute only what is needed

Fibonacci with DP:

Fill table $F[0..n]$ left to right:

Key conditions for DP

1. **Optimal substructure**: optimal solution is built from optimal solutions of sub-problems
2. **Overlapping sub-problems**: sub-problems share common sub-sub-problems (unlike D&C)

DP vs. Divide & Conquer

Section 2

Coin-Row Problem

Coin-Row Problem (House Robber)

Problem

Given a row of n coins with values $c_1, c_2, \dots, c_n$, pick up the maximum total value subject to: *no two adjacent coins may be selected*.

Recurrence: Let $F(i) = \text{max value from coins } 1..i$:

$$F(i) = \max\{c_i + F(i-2), F(i-1)\}$$

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

Example: [5, 1, 2, 10, 6, 2]

i	0	1	2	3	4	5	6
c	—	5	1	2	10	6	2
F	0	5	5	7	15	15	17

Trace:

- ▶ $F(2) = \max(1 + 0, 5) = 5$
- ▶ $F(3) = \max(2 + 5, 5) = 7$
- ▶ $F(4) = \max(10 + 5, 7) = 15$
- ▶ $F(5) = \max(6 + 7, 15) = 15$
- ▶ $F(6) = \max(2 + 15, 15) = 17$

Optimal selection: trace back — pick coins at positions 1, 4, 6 (values 5, 10, 2).

$\mathcal{O}(n)$

Section 3

Change-Making Problem

Change-Making

Problem: Find the minimum number of coins from denominations $d_1 < d_2 < \dots < d_m$ that sum to amount n .

Recurrence:

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

Why greedy fails: denominations $\{1, 3, 4\}$, amount 6.

Greedy: $4 + 1 + 1 = 3$ coins.

DP optimal: $3 + 3 = 2$ coins.

DP table for $\{1, 3, 4\}$, $n = 6$:

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

Computation:

- ▶ $F(3) = \min\{F(2), F(0)\} + 1 = 1$
- ▶ $F(4) = \min\{F(3), F(1), F(0)\} + 1 = 1$
- ▶ $F(6) = \min\{F(5), F(3), F(2)\} + 1 = 2$

Complexity: $\mathcal{O}(nm)$

4

Section 4

0-1 Knapsack

0-1 Knapsack — DP Solution

Setup: n items, weights w_i , values v_i , capacity W .

$F(i, j)$ = max value using first i items and capacity j .

Recurrence:

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

$F(0, j) = 0$ for all j ; $F(i, 0) = 0$ for all i .

Example ($W = 5$):

	Capacity j					
	0	1	2	3	4	5
$\emptyset$	0	0	0	0	0	0
$w_1=2, v_1=12$	0	0	12	12	12	12
$w_2=1, v_2=10$	0	10	12	22	22	22
$w_3=3, v_3=20$	0	10	12	22	30	32
$w_4=2, v_4=15$	0	10	15	25	30	37

Optimal value: **37** (items 1, 2, 4)

$\mathcal{O}(nW)$

Section 5

Optimal Binary Search Trees

Optimal BST

Problem: Given n keys $a_1 < \dots < a_n$ with search probabilities $p_1, \dots, p_n$, build a BST that minimises average comparisons.

Recurrence: $C[i, j] = \min$ comparisons for $a_i, \dots, a_j$:

$$C[i, j] = \min_{i \leq k \leq j} \{ C[i, k-1] + C[k+1, j] \} + \sum_{s=i}^j p_s$$

$$C[i, i-1] = 0; \quad C[i, i] = p_i$$

Example: A, B, C, D with $p = 0.1, 0.2, 0.4, 0.3$

$i \backslash j$	0	1	2	3	4
1	0	.1	.4	1.1	1.7
2		0	.2	.8	1.4
3			0	.4	1.0
4				0	.3
5					0

Optimal avg

comparisons: 1.7

$\mathcal{O}(n^3)$

Section 6

Warshall's & Floyd's Algorithms

Warshall's Algorithm — Transitive Closure

Problem: Given a digraph, compute the $n \times n$ boolean matrix T where $T[i][j] = 1$ iff there is a path from vertex i to vertex j .

Recurrence:

$$R^{(k)}[i,j] = R^{(k-1)}[i,j] \vee (R^{(k-1)}[i,k] \wedge R^{(k-1)}[k,j])$$

At iteration k : does a path $i \rightarrow j$ exist using only vertices $1, \dots, k$ as intermediaries?

Algorithm

```
1:  $R^{(0)} \leftarrow A$ 
2: for  $k \leftarrow 1$  to  $n$ 
3:   for  $i \leftarrow 1$  to  $n$ 
4:     for  $j \leftarrow 1$  to  $n$ 
5:        $R[i,j] \leftarrow R[i,j] \text{ or } (R[i,k] \text{ and } R[k,j])$ 
6:     end for
7:   end for
8: end for
9: return  $R$ 
```

Complexity: $\Theta(n^3)$ In-place
overwrite possible

Floyd's Algorithm — All-Pairs Shortest Paths

Problem: Find shortest-path distances between every pair of vertices in a weighted digraph (no negative cycles).

Recurrence:

$$D^{(k)}[i, j] = \min\{D^{(k-1)}[i, j], D^{(k-1)}[i, k] + D^{(k-1)}[k, j]\}$$

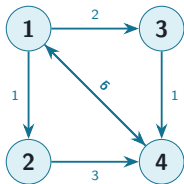
$D^{(0)} = W$ (weight matrix; ∞ for no edge).

Algorithm

```
1:  $D \leftarrow W$ 
2: for  $k \leftarrow 1$  to  $n$ 
3:   for  $i \leftarrow 1$  to  $n$ 
4:     for  $j \leftarrow 1$  to  $n$ 
5:        $D[i, j] \leftarrow \min(D[i, j],$ 
6:                            $D[i, k] + D[k, j])$ 
7:     end for
8:   end for
9: end for
10: return  $D$ 
```

Complexity: $\Theta(n^3)$ In-place
overwrite possible

Floyd's Algorithm — Example



$D^{(0)}$ (initial):

$$\begin{pmatrix} 0 & 1 & 2 & 5 \\ \infty & 0 & \infty & 3 \\ \infty & \infty & 0 & 1 \\ 6 & \infty & \infty & 0 \end{pmatrix}$$

$D^{(4)}$ (final):

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 9 & 0 & 11 & 3 \\ 7 & 8 & 0 & 1 \\ 6 & 7 & 8 & 0 \end{pmatrix}$$

E.g., shortest path $2 \rightarrow 1 = 9$ (via $4 \rightarrow 1$, cost $3 + 6$).

Section 7

Coin-Collecting Problem

Coin-Collecting on a Grid

Problem: A robot at cell $(1, 1)$ collects coins while moving only *right* or *down* to reach (n, m) . Each cell has c_{ij} coins.

Recurrence:

$$F(i, j) = \max\{F(i-1, j), F(i, j-1)\} + c_{ij}$$

$$F(0, j) = F(i, 0) = 0 \text{ (boundary)}$$

Example (5×6 board):

	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

Max coins =

5

$O(nm)$

Chapter 6 Summary

Problems solved with DP:

Problem	Complexity
Fibonacci	$\Theta(n)$
Coin-Row	$\Theta(n)$
Change-Making	$\mathcal{O}(nm)$
0-1 Knapsack	$\Theta(nW)$
Coin-Collecting	$\Theta(nm)$
Optimal BST	$\Theta(n^3)$
Warshall / Floyd	$\Theta(n^3)$

DP recipe

1. Define the sub-problem and its parameter(s)
2. Write the recurrence (and base cases)
3. Fill the DP table bottom-up (or use memoisation)
4. Trace back to find the optimal solution

Next chapter

Chapter 7: Greedy Algorithms — make locally optimal choices and prove global optimality.