

Course «Algorithms» · Lecture 6

Dynamic Programming: the Idea and Three Classic Examples

Textbook: A. Levitin, Section 8.1

Today’s goal

Learn to solve a problem by **solving each subproblem once** and storing the answer in a table.

- when dynamic programming works
- a five-step recipe
- three classic examples: coin row, change making, coin collecting

From Fibonacci to a general technique

- Naive recursive Fibonacci takes **exponential** time
- Yet only $n + 1$ different values F(0), ..., F(n) exist
- It is slow because it solves the **same subproblem** again and again

> **Dynamic programming:** solve each subproblem once, record it in a table, build the answer from the table.

"Programming" means planning in a table, not writing code. Richard Bellman, 1950s.

When does it work?

Overlapping subproblems

The same subproblems appear many times. Mergesort has none, so storing results gives nothing there.

Optimal substructure

An optimal solution contains optimal solutions of its subproblems. Fails for the *longest simple path*.

The standard recipe

1. **Define the subproblem:** what exactly F(i) or F(i, j) means
2. **Write the recurrence** from the *last decision*
3. **Initial conditions** for the smallest subproblems
4. **Order of evaluation:** bottom-up
5. **Recover the solution** by tracing back

Time = number of table entries × work per entry.

Example 1: the coin-row problem

A row of *n* coins with values $c_1, \ldots, c_n$. Pick the maximum total so that **no two adjacent coins** are taken.

Subproblem: F(n) is the best amount from the first *n* coins.
Last decision: take coin *n* or not?

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

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

Coin row: 5, 1, 2, 10, 6, 2

i	0	1	2	3	4	5	6
c_i		5	1	2	10	6	2
F(i)	0	5	5	7	15	15	17

$$F(3) = \max\{2 + F(1),\ F(2)\} = \max\{7,\ 5\} = 7$$

$$F(4) = \max\{10 + F(2),\ F(3)\} = \max\{15,\ 7\} = 15$$

$$F(6) = \max\{2 + F(4),\ F(5)\} = \max\{17,\ 15\} = 17$$

Which coins give 17?

- F(6) = 17 came from coin 6 + F(4)
- F(4) = 15 came from coin 4 + F(2)
- F(2) = F(1): coin 2 skipped, coin 1 taken

> Optimal set: $\{c_1, c_4, c_6\} = \{5, 10, 2\}$, total **17**

Time **Θ(n)**, space **Θ(n)**; only the last two values are needed, so **Θ(1)** space for the value alone.

Example 2: change making

Give change for amount *n* with the **fewest coins** of denominations $d_1 < \ldots < d_m$, with $d_1 = 1$.

Greedy fails: coins 1, 3, 4 and n = 6
greedy: 4 + 1 + 1 (3 coins) · optimum: **3 + 3 (2 coins)**

Last decision: which coin is the last one?

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

Change making: coins 1, 3, 4 and n = 6

i	0	1	2	3	4	5	6
F(i)	0	1	2	1	1	2	2

$$F(5) = \min\{F(4),\ F(2),\ F(1)\} + 1 = \min\{1,\ 2,\ 1\} + 1 = 2$$

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

- Trace back 6 → 3 → 0: the answer is 3 + 3
- Time **Θ(nm)**: **pseudo-polynomial**, since n takes only $\log_2 n$ bits

Example 3: coin collecting

A robot goes from the top-left to the bottom-right cell of an $n \times m$ board, moving only **right or down**, and collects as many coins as possible.

Last decision: did we come from above or from the left?

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

$$F(0, j) = 0, \qquad F(i, 0) = 0$$

c_{ij} = 1 if cell (i, j) has a coin, 0 otherwise.

Coin collecting: a 4 × 4 board

Board (1 = coin)

	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	1	0
4	1	0	0	1

Table F

	1	2	3	4
1	0	1	1	2
2	1	1	2	2
3	1	2	3	3
4	2	2	3	4

One best path: (1,1) → (1,2) → (1,3) → (2,3) → (3,3) → (3,4) → (4,4). Time and space **Θ(nm)**.

What the three examples share

	Coin row	**Change making**	**Coin collecting**
Table	1-D, F(i)	1-D, F(i)	2-D, F(i, j)
Goal	maximize	minimize	maximize
Last decision	take coin n or not	which coin is last	from above or left
Time	Θ(n)	Θ(nm)	Θ(nm)
Brute force	exponential	exponential	exponential

Always ask: what was the **last decision**, and what must be true of the rest?

DP and other techniques

- **Divide and conquer:** independent subproblems; DP: overlapping ones
- **Greedy:** one local choice, never revised; DP tries all choices for the last step
- **Brute force:** all candidates; DP is brute force that shares work
- **Top-down** with memoization: next lecture

Summary

- DP needs overlapping subproblems and optimal substructure
- Recipe: subproblem, recurrence from the last decision, initial conditions, order, trace back
- Coin row **Θ(n)**, change making **Θ(nm)**, coin collecting **Θ(nm)**
- Greedy change making fails for coins 1, 3, 4 and n = 6
- Change making is pseudo-polynomial

Review questions

1. Which two properties make DP applicable? Why does Fibonacci benefit and mergesort not?
2. Fill the coin-row table for 3, 12, 4, 6, 9, 2 and find an optimal set.
3. Coins 1, 5, 7 and n = 10: show greedy is not optimal; compute the DP table.
4. Why is change making $\Theta(nm)$, and why is that not polynomial?
5. How many robot paths exist on an $n \times m$ board?