12 Dynamic Programming

21 January 2024

Prof. Dr. Sebastian Wild

CS566 (Wintersemester 2024/25)
Philippps-Universität Marburg

version `2025-01-21 14:00`

Learning Outcomes

Unit 12: ***Dynamic Programming***

1. Be able to apply the DP paradigm to solve new problems.

Outline

12 Dynamic Programming

12.1 Elements of Dynamic Programming

Introduction

applicable to many problems

- *Dynamic Programming (DP)* is a powerful algorithm **design pattern** for exact solutions to **optimization** problems
- Some commonalities with Greedy Algorithms, but with an element of brute force added in

  DP = "careful brute force" (Erik Demaine)

- often yields polynomial time, but usually not linear time algorithms
- for many problems the *only* way we know to build efficient algorithms

- **Naming fun:** The term "dynamic programming", due to Richard Bellman from around 1953, does not refer to computer programming; rather to a program (= plan, schedule) changing with time. It seems to have been at least partly marketing babble devoid of technical meaning . . .

Plan of the Unit

1. Abstract steps of DP (briefly)
2. Details on a concrete example (*matrix chain multiplication*)
3. More examples!

The 6 Steps of Dynamic Programming

1. Define **subproblems** (and relate to original problem)
2. **Guess** (part of solution) $\rightsquigarrow$ local brute force
3. Set up **DP recurrence** (for quality of solution)
4. Recursive implementation with **Memoization**
5. Bottom-up **table filling** (topological sort of subproblem dependency graph)
6. **Backtracing** to reconstruct optimal solution

- Steps 1–3 require insight / creativity / intuition;
  Steps 4–6 are mostly automatic / same each time

$\rightsquigarrow$ Correctness proof usually at level of DP recurrence

👍 running time too! worst case time = #subproblems · time to find single best guess

When does DP (not) help?

- *No Silver Bullet*
  DP is the most widely applicable design technique, but can't *always* be applied

1. Vitally important for DP to be correct:

   Bellman's Optimality Criterion

   For a *correctly guessed* fixed part of the solution,
   ***any* optimal solution to the corresponding subproblems**
   must yield an *optimal solution* to the overall problem (once combined).

When does DP (not) help?

- *No Silver Bullet*
  DP is the most widely applicable design technique, but can't *always* be applied

1. Vitally important for DP to be correct:

   Bellman's Optimality Criterion

   For a *correctly guessed* fixed part of the solution,
   ***any* optimal solution to the corresponding subproblems**
   must yield an *optimal solution* to the overall problem (once combined).

   at most polynomial in n

2. Also, the total **number of different subproblems** should be *"small"*
   (DP potentially still works correctly otherwise, but won't be *efficient*.)

12.2 DP & Matrix Chain Multiplication

The Matrix-Chain Multiplication Problem

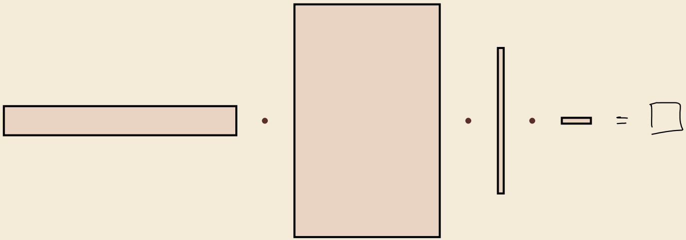
Consider the following exemplary problem

- We have a product $M_0 \cdot M_1 \cdot \dots \cdot M_{n-1}$ of n matrices to compute
- Since (matrix) multiplication is associative, it can be evaluated in different orders.
- For non-square matrices of different sizes, different order can change costs dramatically
  - Assume elementary matrix multiplication algorithm:
  - ⇝ Multiplying $a \times b$-matrix with $b \times c$ matrix costs $a \cdot b \cdot c$ integer multiplications
- **Given:** Row and column counts $\overset{r}{\cancel{c}}[0..n)$ and $\overset{c}{\cancel{r}}[0..n)$ with $r[i+1] = c[i]$ for $i \in [0..n-1)$
  (corresponding to matrices $M_0, \dots, M_{n-1}$ with $M_i \in \mathbb{R}^{r[i] \times c[i]}$)
- **Goal:** parenthesization of the product chain with minimal cost

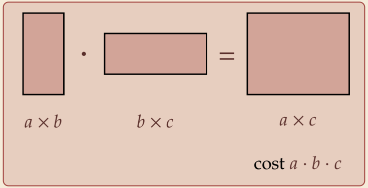
really a binary tree with n leaves!

$(M_0 (M_1 \; M_2))$

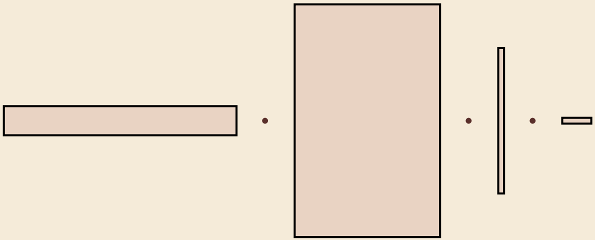
Matrix-Chain Multiplication – Example



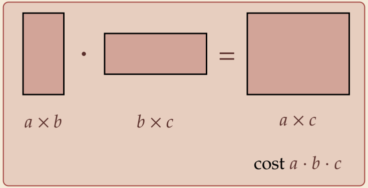
M_0 10×80 · M_1 80×50 · M_2 50×2 · M_3 2×10



Matrix-Chain Multiplication – Example



M_0	M_1	M_2	M_3
10×80	80×50	50×2	2×10



Parenthesization	Cost (integer multiplications)		
$M_0 \cdot (M_1 \cdot (M_2 \cdot M_3))$	$1000 + 40\,000 + 8000$	$=$	$49\,000$
$M_0 \cdot ((M_1 \cdot M_2) \cdot M_3)$	$8000 + 1600 + 8000$	$=$	$17\,600$
$(M_0 \cdot M_1) \cdot (M_2 \cdot M_3)$	$40\,000 + 1000 + 5000$	$=$	$46\,000$
$(M_0 \cdot (M_1 \cdot M_2)) \cdot M_3$	$8000 + 1600 + 200$	$=$	$9\,800$
$((M_0 \cdot M_1) \cdot M_2) \cdot M_3$	$40\,000 + 1000 + 200$	$=$	$41\,200$

first or last operation

Greedy fails both ways!