CS 4102: Algorithms

Lecture 12: Dynamic Programming

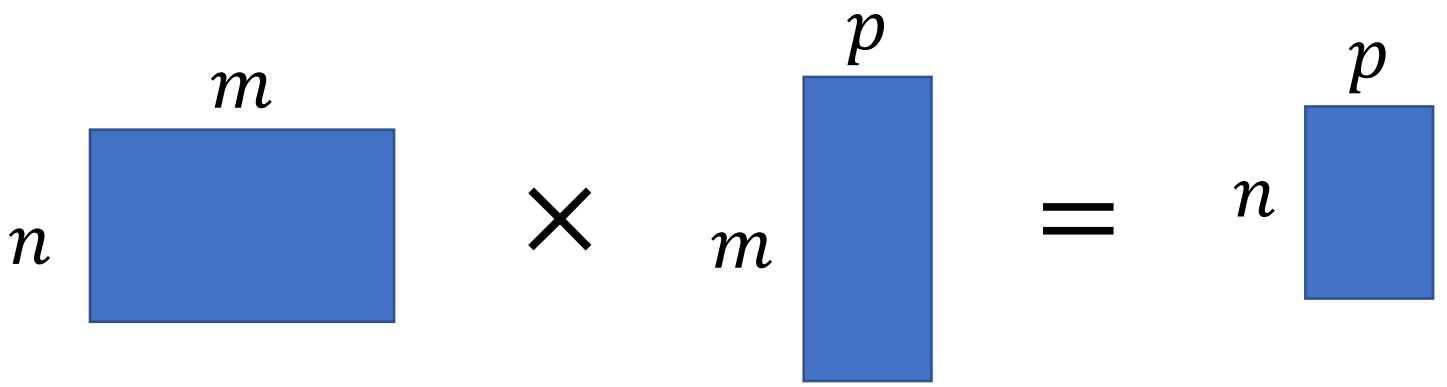
David Wu

Fall 2019

Warm-Up

How many arithmetic operations are required to multiply a $n \times m$ matrix with a $m \times p$ matrix?

Warm-Up



m multiplications and additions per element of output matrix

$n \cdot p$ elements to compute (in output matrix)

Total cost: $m \cdot n \cdot p$

Today's Keywords

Dynamic Programming

Matrix Chaining

Seam Carving

Longest Common Subsequence

CLRS Readings: Chapter 14

Homework

- **HW4** due **Saturday, October 12, 11pm**
  - Divide and conquer, sorting, and dynamic programming
  - Written (use LaTeX!) – Submit both **zip** and **pdf** (two separate attachments)!

- **Midterm Exam:** Tuesday October 15 (in class)

- Regrade office hours
  - Thursdays 11am-12pm @ Rice 210
  - Thursdays 4pm-5pm @ Rice 501

Dynamic Programming

Requires optimal substructure

- Solution to larger problem contains the solutions to smaller ones

General Blueprint:

1. Identify recursive structure of the problem
   - What is the “last thing” done?
2. Select a good order for solving subproblems
   - “Top Down:” Solve each problem recursively
   - “Bottom Up:” Iteratively solve each problem from smallest to largest

Log Cutting (Review)

Given: a log of length n, a list (of length n) of prices P

Problem: Find the best way to cut the log

$P[i]$ is the price of a cut of size i

Price:	1	5	8	9	10	17	17	20	24	30
Length:	1	2	3	4	5	6	7	8	9	10

Problem formulation: Find lengths $\ell_1, \ldots, \ell_k$ that maximizes $\sum_{i \in [k]} P[\ell_i]$ and such that $\sum_{i \in [k]} \ell_i = n$

Brute Force: $O(2^n)$

Dynamic Programming

Requires optimal substructure

- Solution to larger problem contains the solutions to smaller ones

General Blueprint:

1. Identify recursive structure of the problem
   - What is the “last thing” done?
2. Select a good order for solving subproblems
   - “Top Down:” Solve each problem recursively
   - “Bottom Up:” Iteratively solve each problem from smallest to largest

Dynamic Programming

Requires optimal substructure

- Solution to larger problem contains the solutions to smaller ones

General Blueprint:

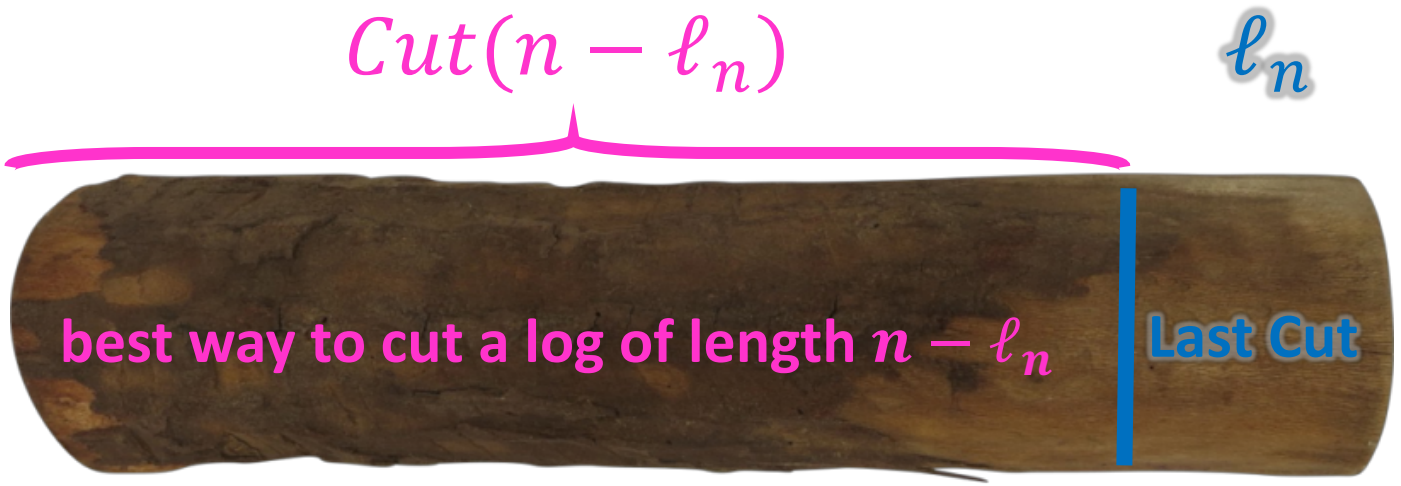
1. Identify recursive structure of the problem
   - What is the "last thing" done?
2. Select a good order for solving subproblems
   - "Top Down:" Solve each problem recursively
   - "Bottom Up:" Iteratively solve each problem from smallest to largest

Step 1: Identify Recursive Structure

$P[i] =$ value of a cut of length i

$\text{Cut}(n) =$ value of best way to cut a log of length n

$$\text{Cut}(n) = \max \begin{cases} \text{Cut}(n-1) + P[1] \\ \text{Cut}(n-2) + P[2] \\ \vdots \\ \text{Cut}(0) + P[n] \end{cases}$$



Dynamic Programming

Requires optimal substructure

- Solution to larger problem contains the solutions to smaller ones

General Blueprint:

1. Identify recursive structure of the problem
   - What is the "last thing" done?
2. Select a good order for solving subproblems
   - "Top Down:" Solve each problem recursively
   - "Bottom Up:" Iteratively solve each problem from smallest to largest

2. Select a Good Order for Solving Subproblems

Solve smallest subproblem first

$\text{Cut}(0) = 0$

$\text{Cut}(i)$:	0						
Length:	**0**	**1**	**2**	**3**	**4**	**5**	**6**

Price:	1	18	24	36	50	50
Length:	**1**	**2**	**3**	**4**	**5**	**6**

2. Select a Good Order for Solving Subproblems

Solve smallest subproblem first

$$\text{Cut}(1) = \text{Cut}(0) + P[1]$$

$\text{Cut}(i)$:	0	1					
Length:	**0**	**1**	**2**	**3**	**4**	**5**	**6**

Price:	1	18	24	36	50	50
Length:	**1**	**2**	**3**	**4**	**5**	**6**

2. Select a Good Order for Solving Subproblems

Solve smallest subproblem first

$$\text{Cut}(2) = \max \begin{cases} \text{Cut}(1) + P[1] \\ \text{Cut}(0) + P[2] \end{cases}$$

$\text{Cut}(i)$:	0	1	18				
Length:	**0**	**1**	**2**	**3**	**4**	**5**	**6**

Price:	1	18	24	36	50	50
Length:	**1**	**2**	**3**	**4**	**5**	**6**

2. Select a Good Order for Solving Subproblems

Solve smallest subproblem first

$$\text{Cut}(3) = \max \begin{cases} \text{Cut}(2) + P[1] \\ \text{Cut}(1) + P[2] \\ \text{Cut}(0) + P[3] \end{cases}$$

$\text{Cut}(i)$:	0	1	18	24			
Length:	**0**	**1**	**2**	**3**	**4**	**5**	**6**

Price:	1	18	24	36	50	50
Length:	**1**	**2**	**3**	**4**	**5**	**6**

2. Select a Good Order for Solving Subproblems

Solve smallest subproblem first

$$\text{Cut}(n) = \max \begin{cases} \text{Cut}(n-1) + P[1] \\ \text{Cut}(n-2) + P[2] \\ \vdots \\ \text{Cut}(0) + P[n] \end{cases}$$

$\text{Cut}(i)$:	0	1	18	24	36	50	54
Length:	**0**	**1**	**2**	**3**	**4**	**5**	**6**

Price:	1	18	24	36	50	50
Length:	**1**	**2**	**3**	**4**	**5**	**6**

Log Cutting Pseudocode

```
initialize memory C
cut(n):
    C[0] = 0
    for i = 1 to n:
        best = 0
        for j = 1 to i:
            best = max(best, C[i-j] + P[j])
        C[i] = best
    return C[n]
```

Run Time: $O(n^2)$

Finding the Cuts

This procedure told us the profit, but not the cuts themselves

Idea: remember the choice that you made, then backtrack

Remembering the Choices

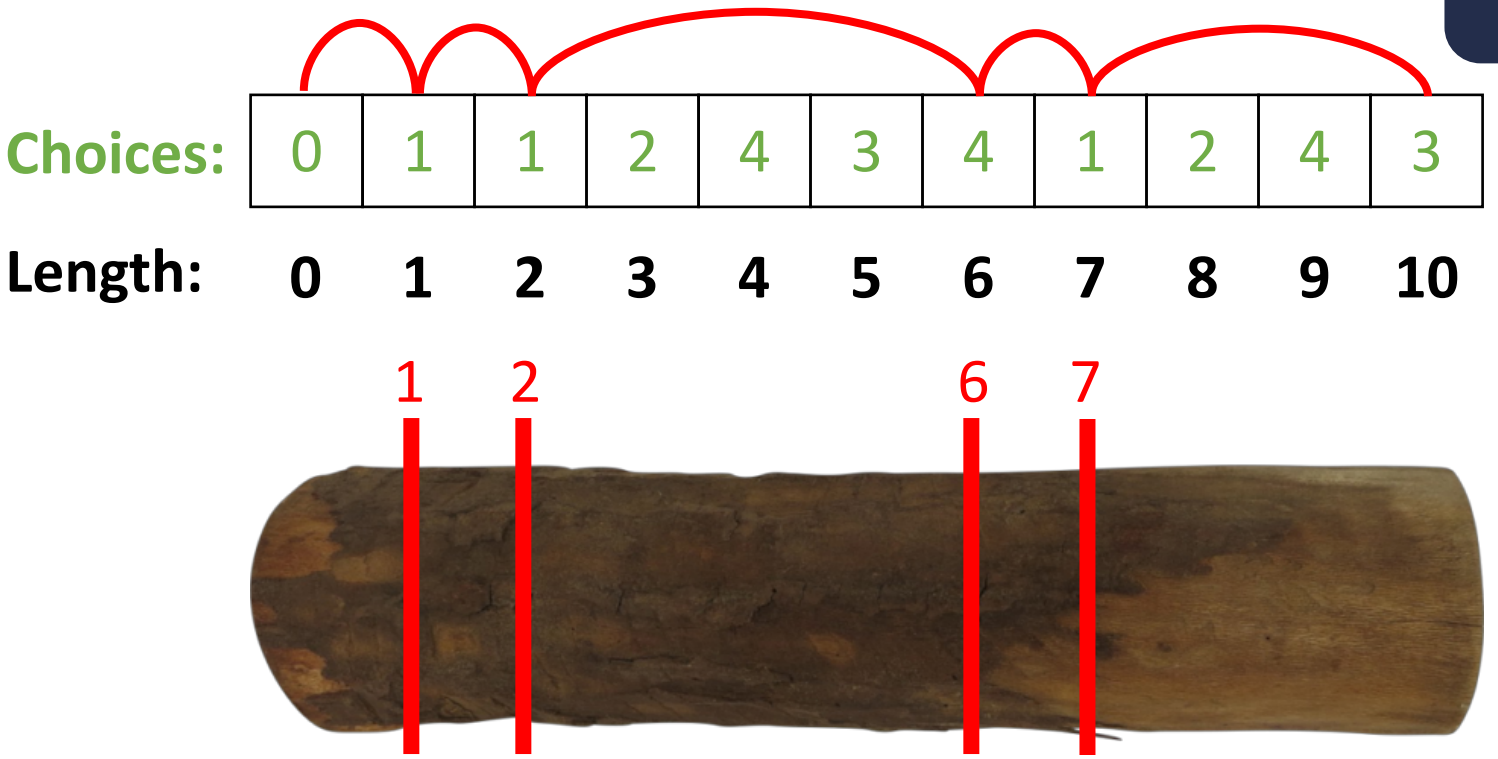
```
initialize memory C, choices
cut(n):
    C[0] = 0
    for i = 1 to n:
        best = 0
        for j = 1 to i:
            if best < C[i-j] + P[j]:
                best = C[i-j] + P[j]
                choices[i] = j
        C[i] = best
    return C[n], choices
```

size of the last cut

Reconstruct the Cuts

Backtrack through the choices:

Optimal cut for log of length 10 is to first cut segment of length 3



Note: this choices array is an example (does not correspond to previous cost array!)

Backtracking Pseudocode

```
i = n
while i > 0:
    print choices[i]
    i = i - choices[i]
```

Dynamic Programming

Requires optimal substructure

- Solution to larger problem contains the solutions to smaller ones

General Blueprint:

1. Identify recursive structure of the problem
   - What is the “last thing” done?
2. Select a good order for solving subproblems
   - “Top Down:” Solve each problem recursively
   - “Bottom Up:” Iteratively solve each problem from smallest to largest
3. Save solution to each subproblem in memory

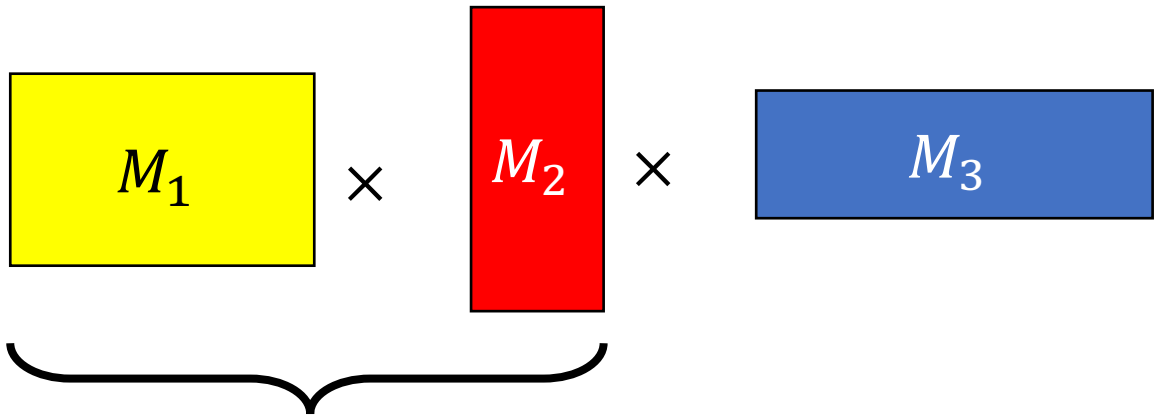
Matrix Chaining

Problem: Given a sequence of matrices $M_1, \ldots, M_n$, what is the most efficient way to multiply them?

Remember: matrix multiplication is associative

Order Matters!

$n_1 \times n_2$ $n_2 \times n_3$ $n_3 \times n_4$

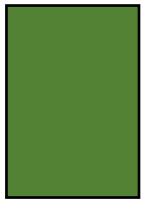


$n_1 = 7$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$

Total operations:

2520



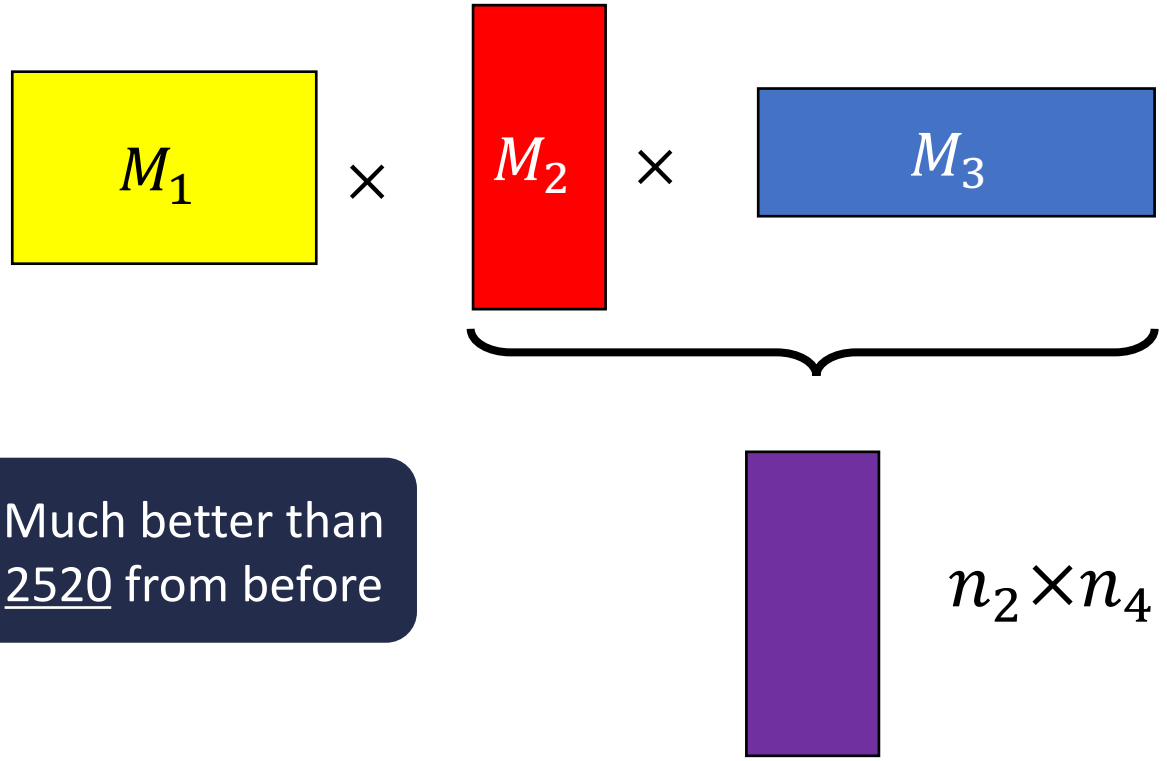
$n_1 \times n_3$

$(M_1 \times M_2) \times M_3$

- requires $n_1 n_2 n_3 + n_1 n_3 n_4$ operations

Order Matters!

$n_1 \times n_2$ $n_2 \times n_3$ $n_3 \times n_4$



$n_1 = 7$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$

Total operations:

2160

Much better than 2520 from before

$n_2 \times n_4$

$M_1 \times (M_2 \times M_3)$

- requires $n_1 n_2 n_4 + n_2 n_3 n_4$ operations

Dynamic Programming

Requires optimal substructure

- Solution to larger problem contains the solutions to smaller ones

General Blueprint:

1. Identify recursive structure of the problem
   - What is the “last thing” done?
2. Select a good order for solving subproblems
   - “Top Down:” Solve each problem recursively
   - “Bottom Up:” Iteratively solve each problem from smallest to largest
3. Save solution to each subproblem in memory

Dynamic Programming

Requires optimal substructure

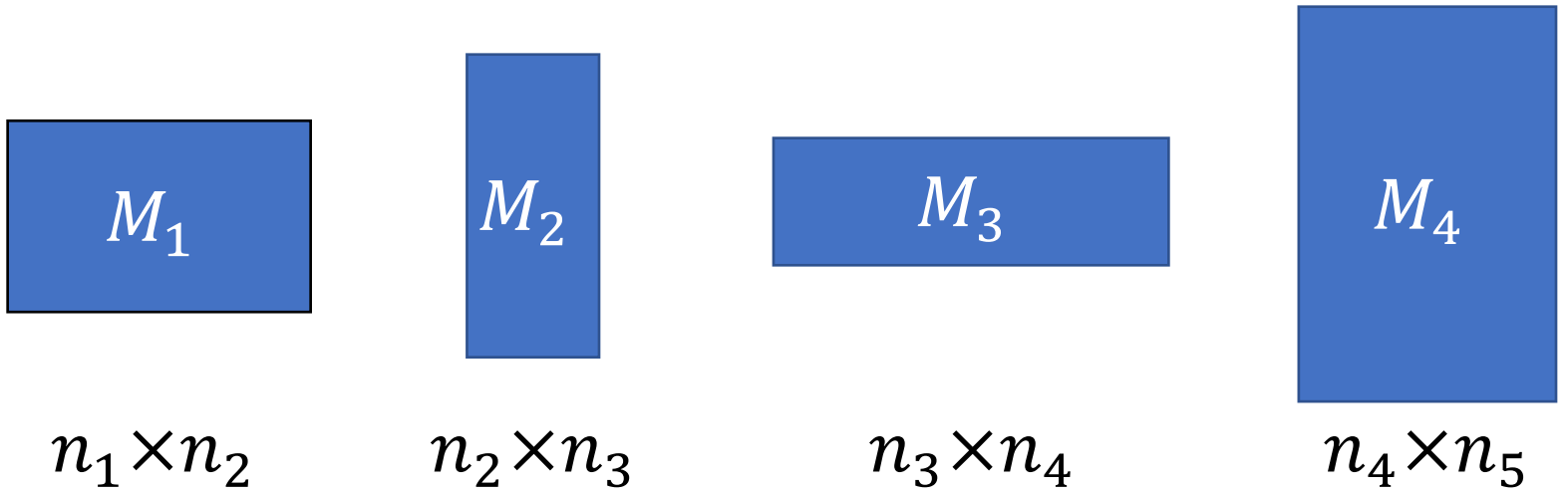
- Solution to larger problem contains the solutions to smaller ones

General Blueprint:

1. Identify recursive structure of the problem
   - What is the "last thing" done?
2. Select a good order for solving subproblems
   - "Top Down:" Solve each problem recursively
   - "Bottom Up:" Iteratively solve each problem from smallest to largest
3. Save solution to each subproblem in memory

1. Identify Recursive Structure

$\text{Best}(1, n)$ = cheapest way to multiply together M_1 through M_n



1. Identify Recursive Structure

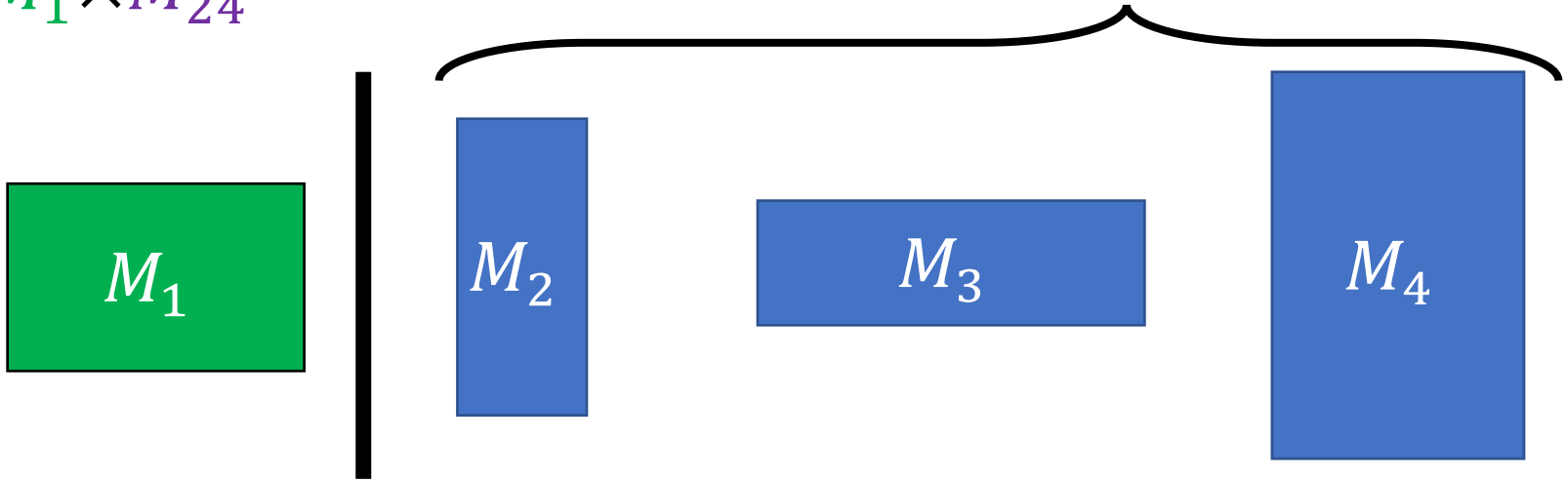
$\text{Best}(1,n) =$ cheapest way to multiply together M_1 through M_n

$$\text{Best}(1,4) = \min \begin{cases} \text{Best}(2,4) + n_1 n_2 n_5 \\ \vdots \end{cases}$$

Operations to compute M_{24} and then multiply M_1 with M_{24}

$n_2 \times n_5$

Last product: $M_1 \times M_{24}$

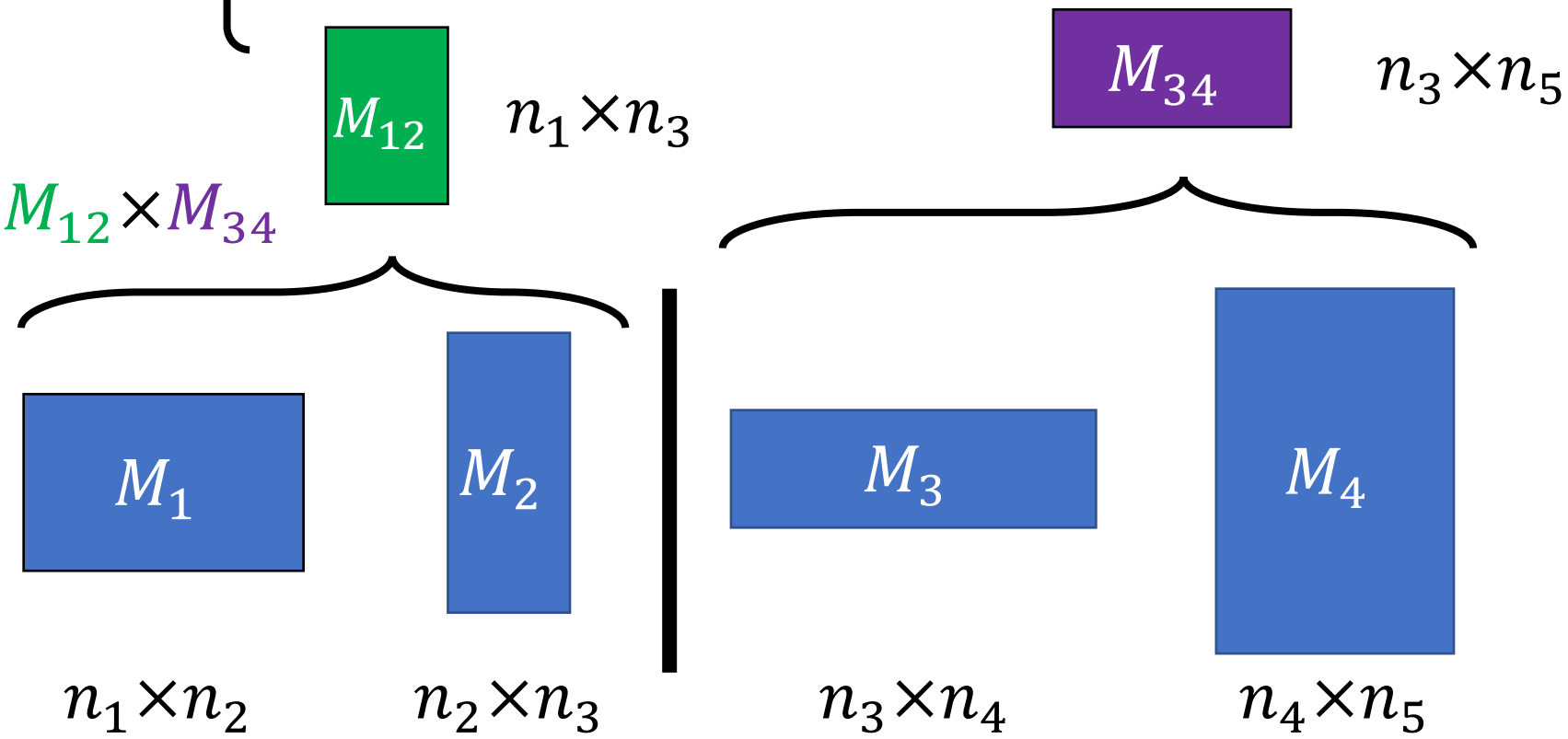


1. Identify Recursive Structure

$\text{Best}(1, n) =$ cheapest way to multiply together M_1 through M_n

$$\text{Best}(1,4) = \min \begin{cases} \text{Best}(2,4) + n_1 n_2 n_5 \\ \text{Best}(1,2) + \text{Best}(3,4) + n_1 n_3 n_5 \end{cases}$$

Last product: $M_{12} \times M_{34}$

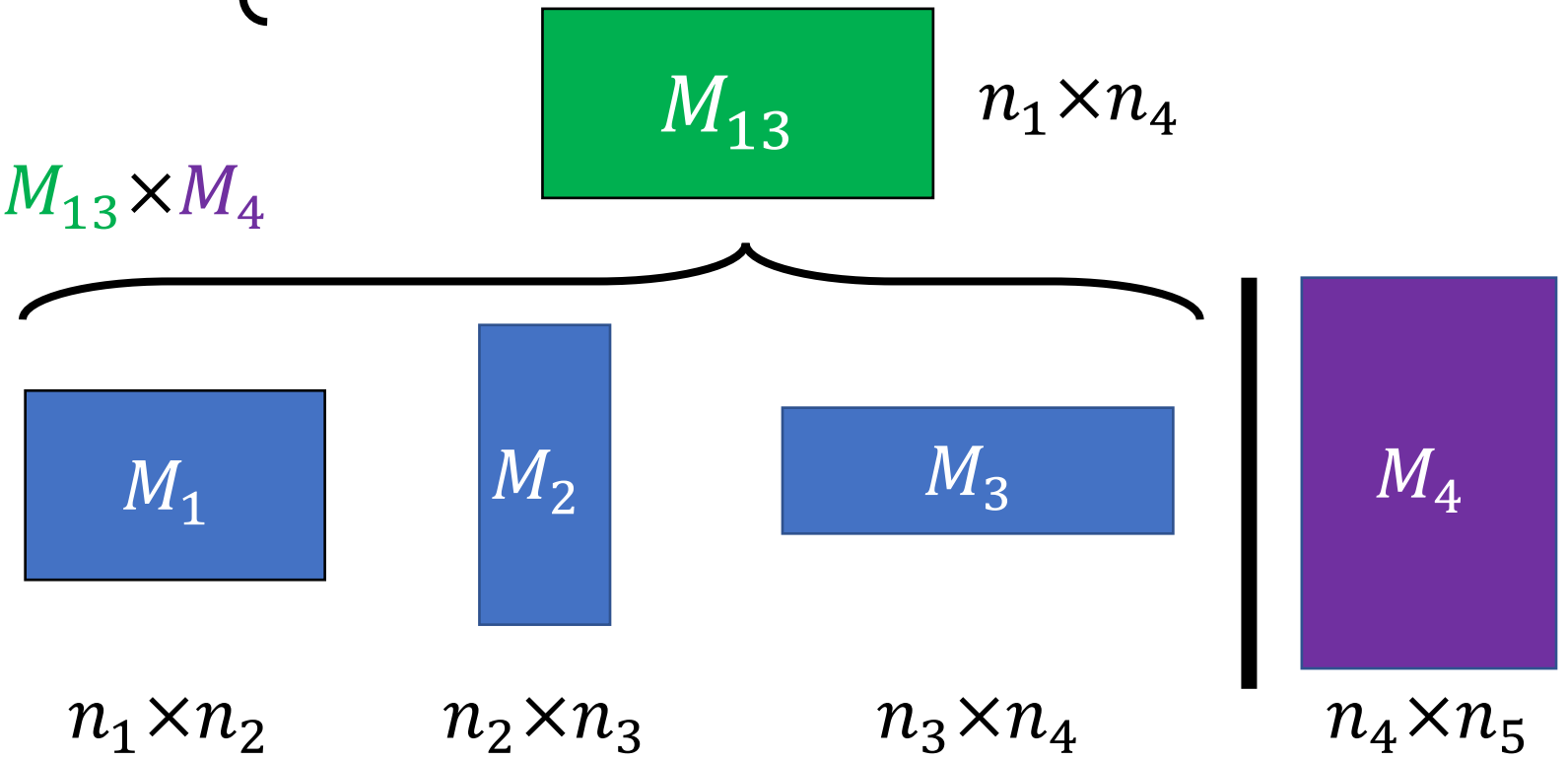


1. Identify Recursive Structure

$\text{Best}(1,n) =$ cheapest way to multiply together M_1 through M_n

$$\text{Best}(1,4) = \min \begin{cases} \text{Best}(2,4) + n_1 n_2 n_5 \\ \text{Best}(1,2) + \text{Best}(3,4) + n_1 n_3 n_5 \\ \text{Best}(1,3) + n_1 n_4 n_5 \end{cases}$$

Last product: $M_{13} \times M_4$



1. Identify Recursive Structure

More generally:

$\text{Best}(i, j) =$ cheapest way to multiply together M_i through M_j

Possible ways to compute $M_i \times M_{i+1} \times \cdots \times M_j$

$M_i \times M_{i+1,j} = M_i \times (M_{i+1} \times \cdots \times M_j)$

$M_{i,i+1} \times M_{i+2,j} = (M_i \times M_{i+1}) \times (M_{i+2} \times \cdots \times M_j)$

$M_{i,i+2} \times M_{i+3,j} = (M_i \times M_{i+1} \times M_{i+2}) \times (M_{i+3} \times \cdots \times M_j)$

$\vdots \qquad \vdots \qquad \vdots$

$M_{i,j-1} \times M_j = (M_i \times \cdots \times M_{j-1}) \times M_j$

Position of the "split" changes

1. Identify Recursive Structure

More generally:

$\text{Best}(i, j) =$ cheapest way to multiply together M_i through M_j

Possible ways to compute $M_i \times M_{i+1} \times \cdots \times M_j$

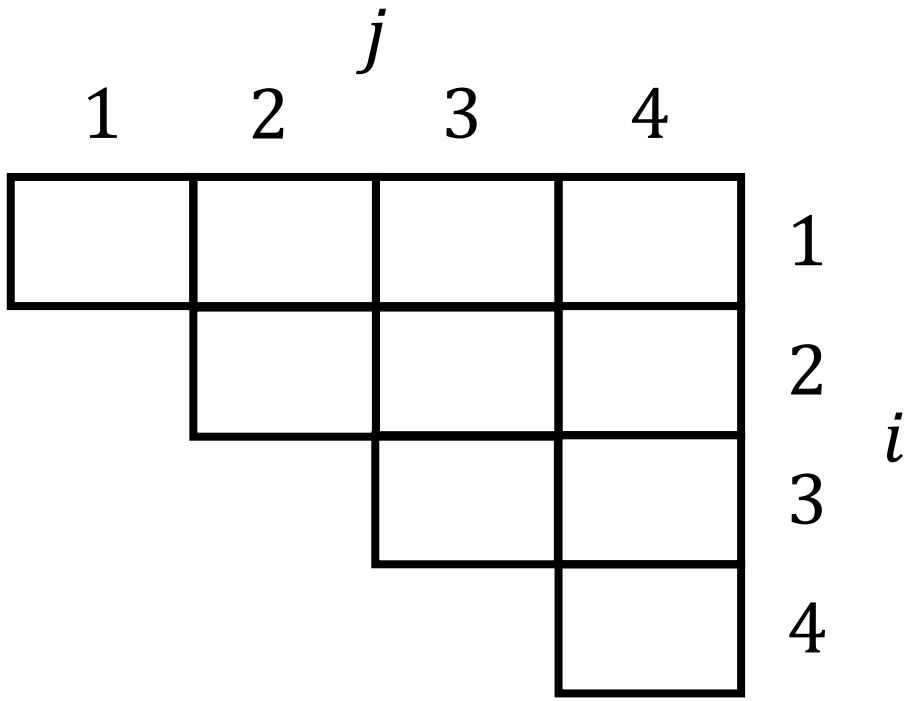
$$\text{Best}(i,j) = \min \begin{cases} \text{Best}(i,i) + \text{Best}(i+1,j) + n_i n_{i+1} n_{j+1} \\ \text{Best}(i,i+1) + \text{Best}(i+2,j) + n_i n_{i+2} n_{j+1} \\ \text{Best}(i,i+2) + \text{Best}(i+3,j) + n_i n_{i+3} n_{j+1} \\ \\ \text{Best}(i,j-1) + \text{Best}(j,j) + n_i n_j n_{j+1} \end{cases}$$

$$\text{Best}(i,i) = 0$$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

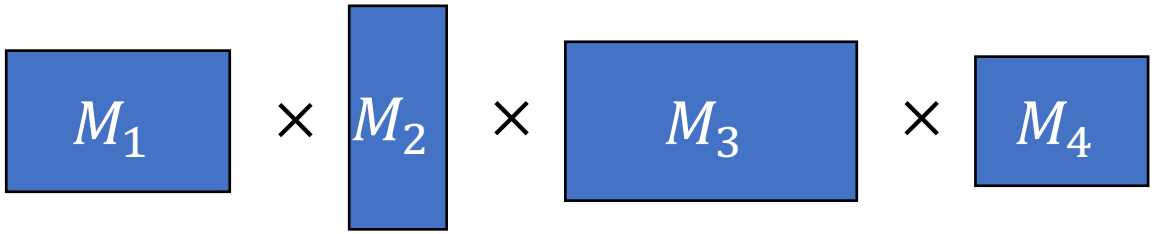
1. Identify Recursive Structure

$$\text{Best}(i,i) = 0 \qquad \text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$$



$n_1 = 5$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$ $n_5 = 6$



Dynamic Programming

Requires optimal substructure

- Solution to larger problem contains the solutions to smaller ones

General Blueprint:

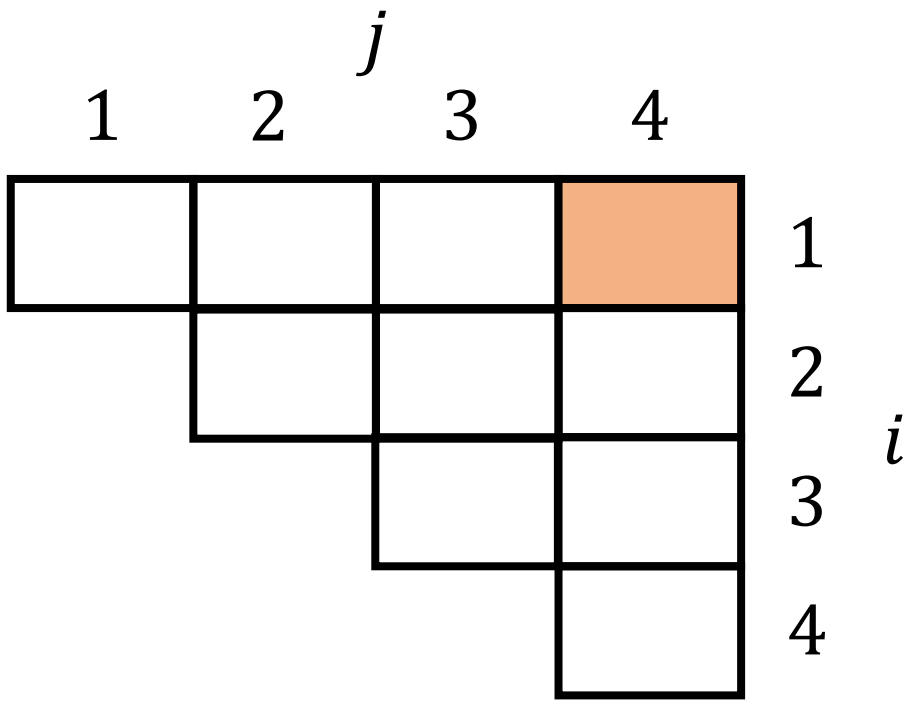
1. Identify recursive structure of the problem
   - What is the “last thing” done?
2. Select a good order for solving subproblems
   - “Top Down:” Solve each problem recursively
   - “Bottom Up:” Iteratively solve each problem from smallest to largest
3. Save solution to each subproblem in memory

2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 4$



$n_1 = 5$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$ $n_5 = 6$

$M_1 \times M_2 \times M_3 \times M_4$

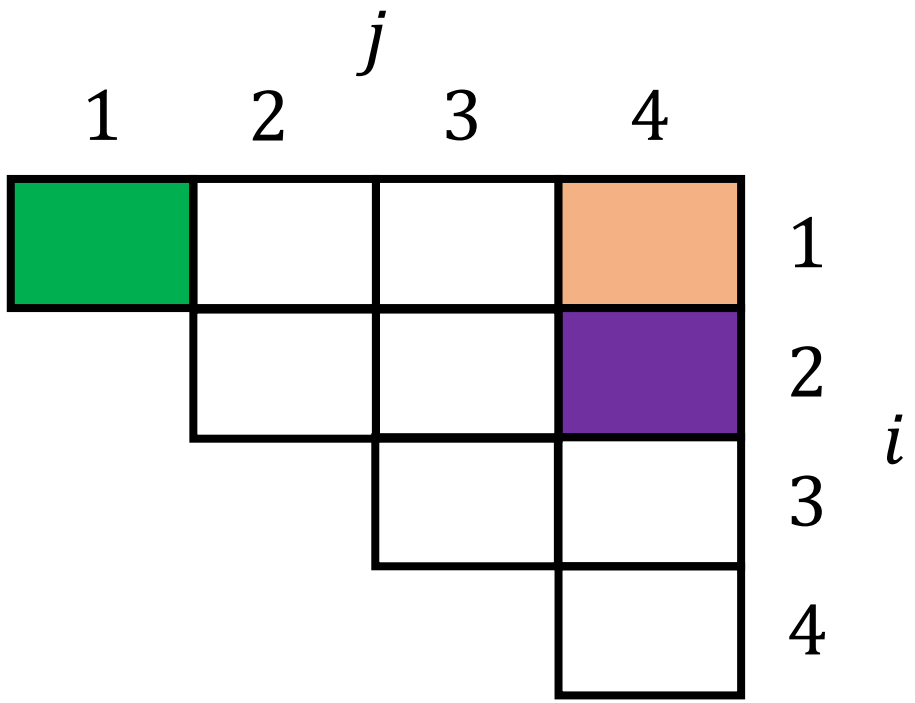
2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

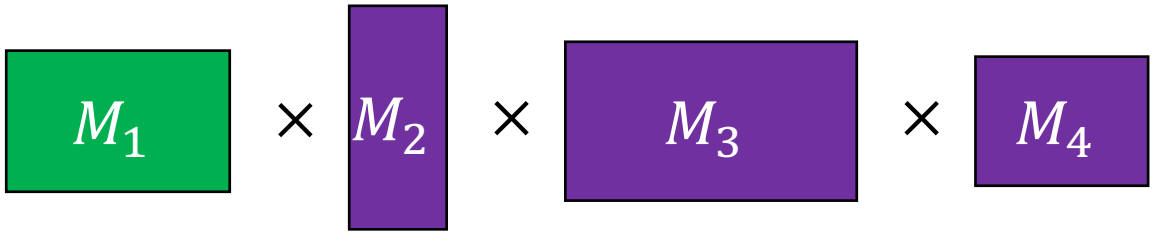
$i = 1, j = 4$

$k = 0$



$n_1 = 5 \quad n_2 = 10$

$n_3 = 20 \quad n_4 = 8 \quad n_5 = 6$



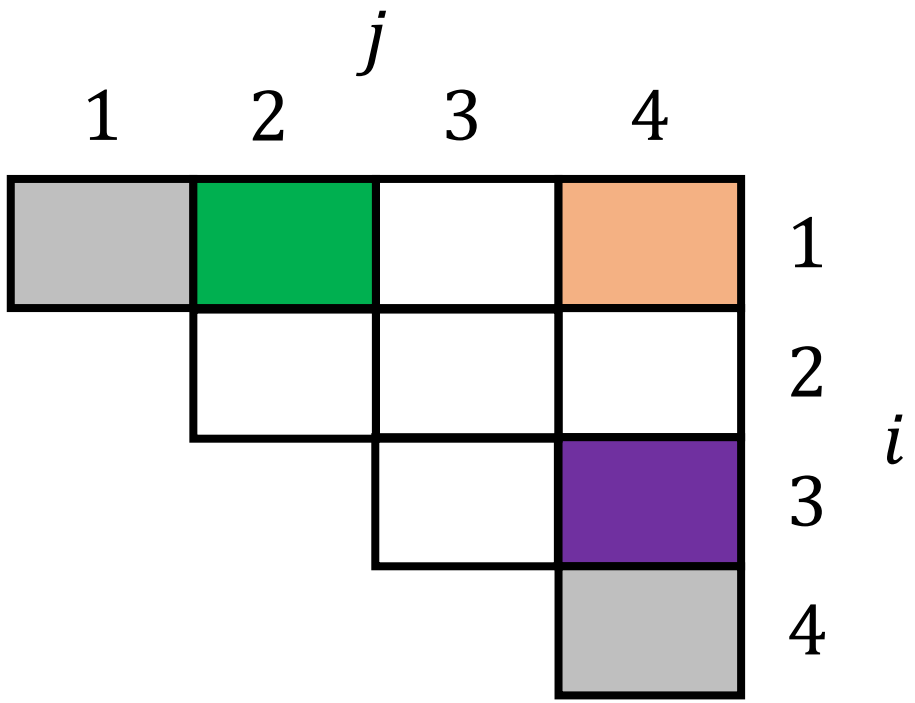
2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 4$

$k = 1$



$n_1 = 5$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$ $n_5 = 6$

$M_1 \times M_2 \times M_3 \times M_4$

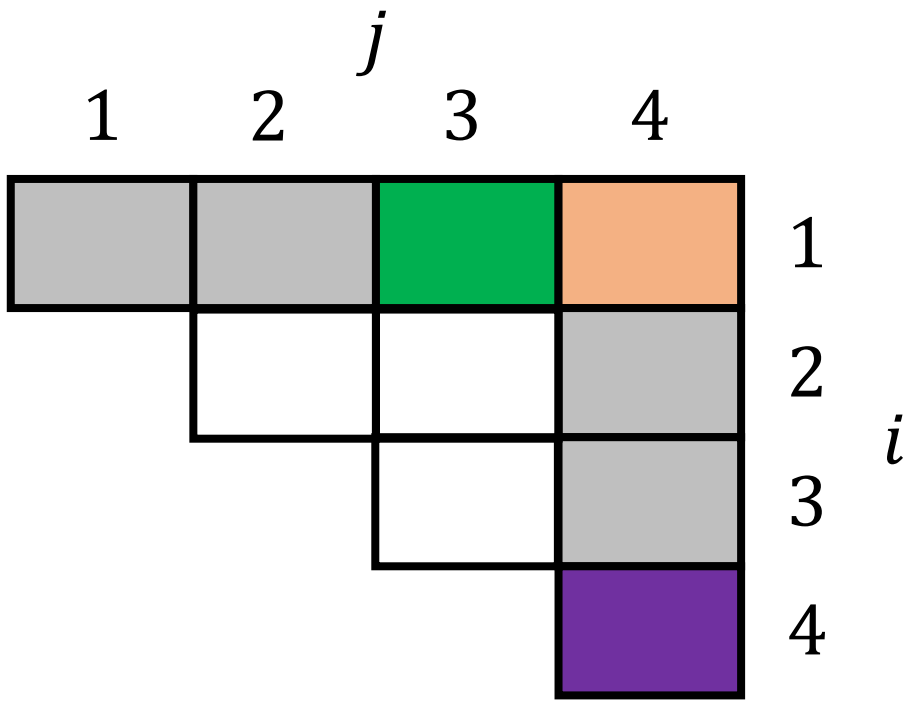
2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 4$

$k = 2$



$n_1 = 5$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$ $n_5 = 6$

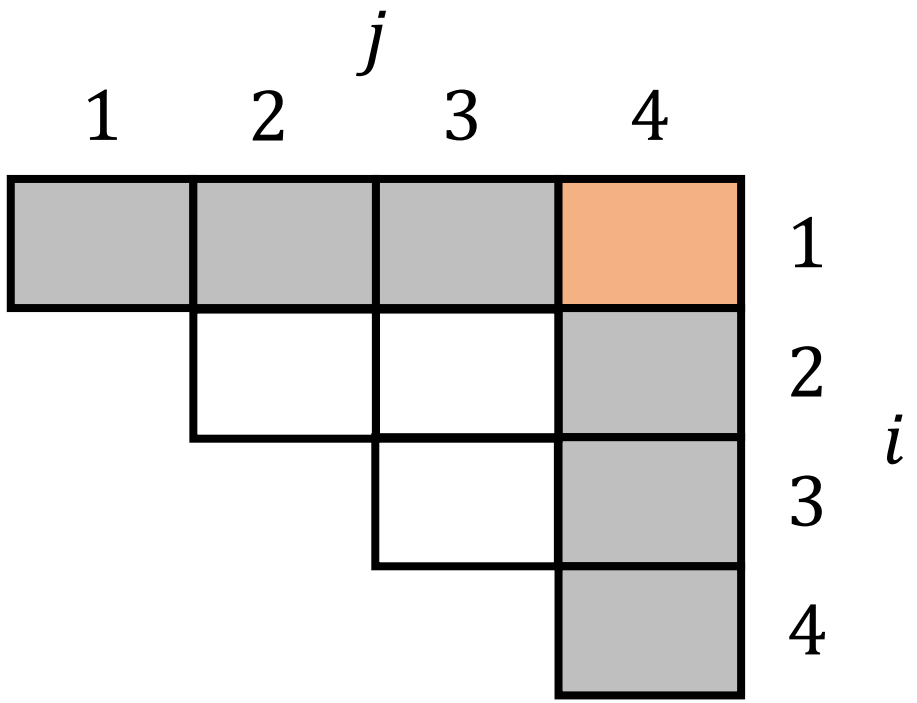
$M_1 \times M_2 \times M_3 \times M_4$

2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

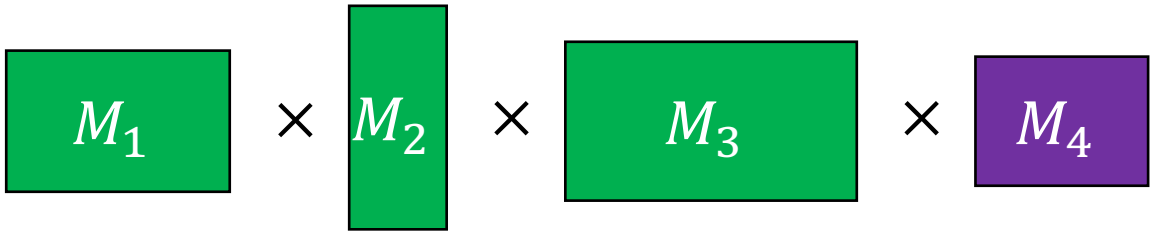
$i = 1, j = 4$



Observation: Value depends on values to its <u>left</u> and <u>below</u>

$n_1 = 5$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$ $n_5 = 6$



2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 4$

i \ j	1	2	3	4
1	1	5	8	10
2		2	6	9
3			3	7
4				4

Observation: Value depends on values to its left and below

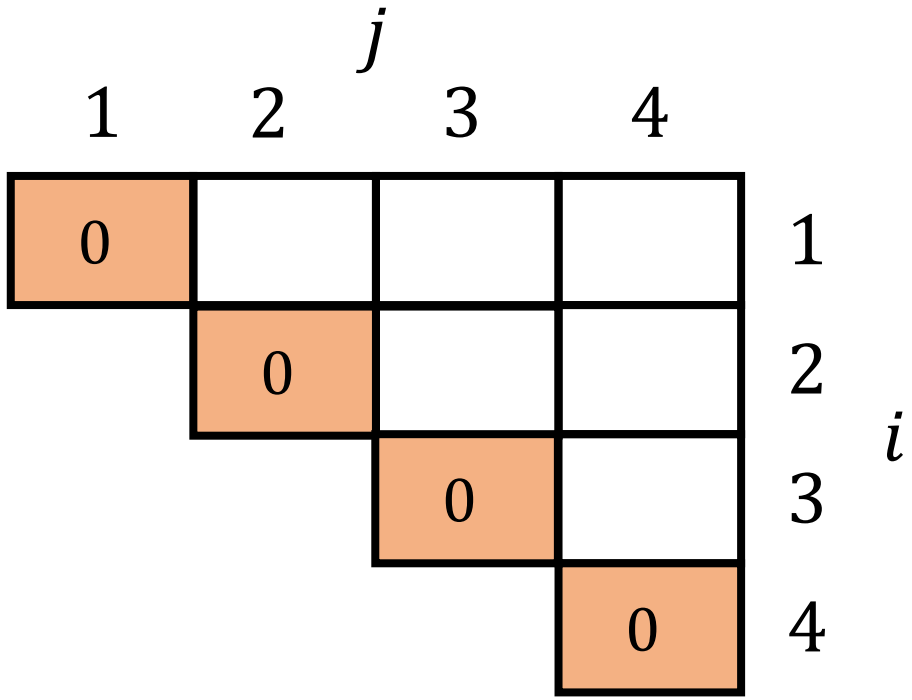
Order: Fill values along diagonal

$n_1 = 5$ $n_2 = 10$ $n_3 = 20$ $n_4 = 8$ $n_5 = 6$

$M_1 \times M_2 \times M_3 \times M_4$

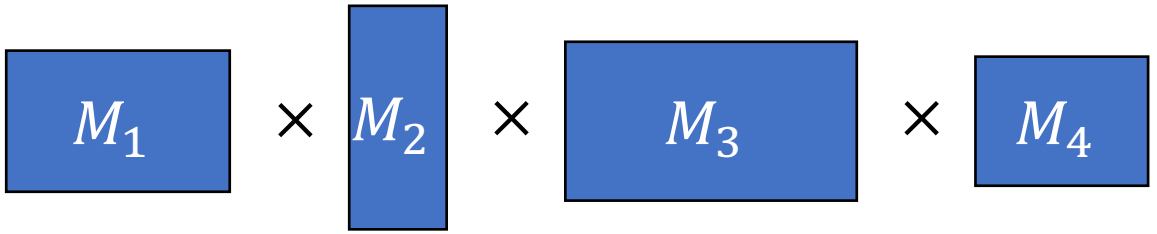
2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$ $\quad \text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$



$n_1 = 5$ $\quad n_2 = 10$

$n_3 = 20$ $\quad n_4 = 8$ $\quad n_5 = 6$



2. Select a Good Order for Solving Subproblems

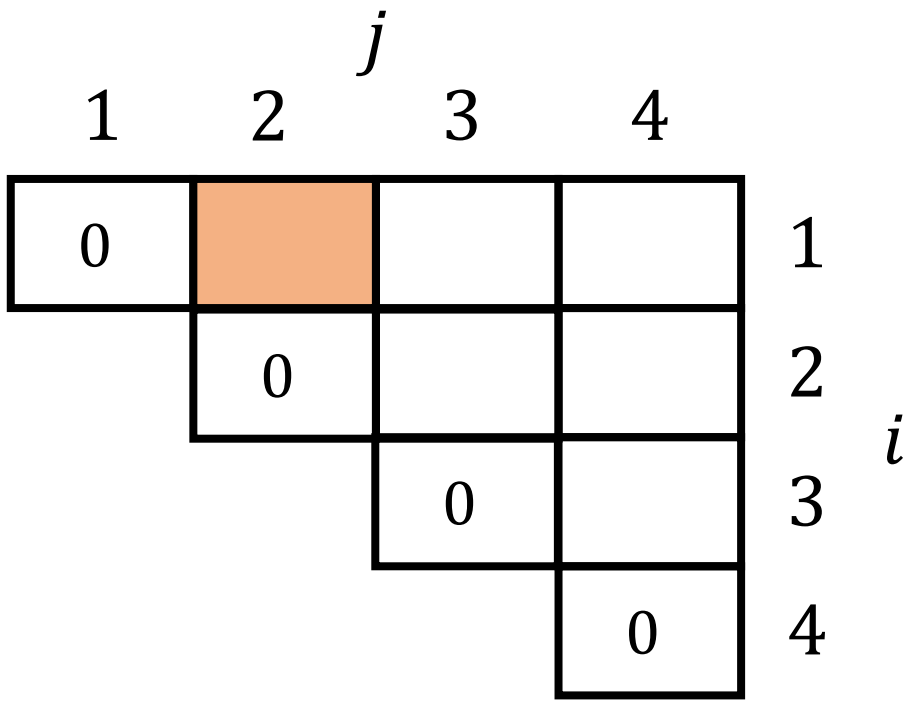
$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 2$

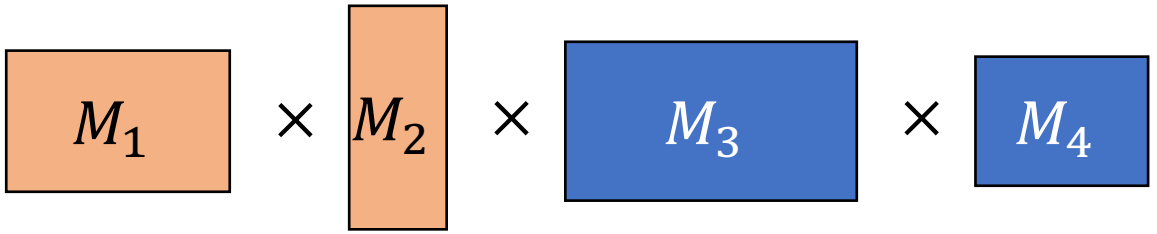
$k = 0$

$n_1 n_2 n_3 = 1000$



$n_1 = 5 \quad n_2 = 10$

$n_3 = 20 \quad n_4 = 8 \quad n_5 = 6$



2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

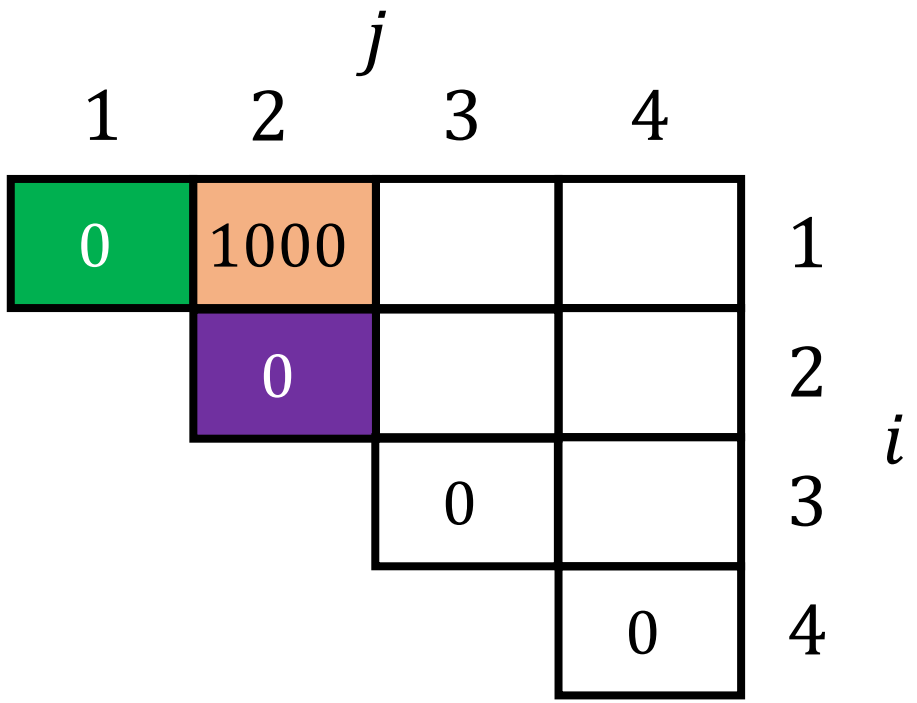
$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 2$

$k = 0$

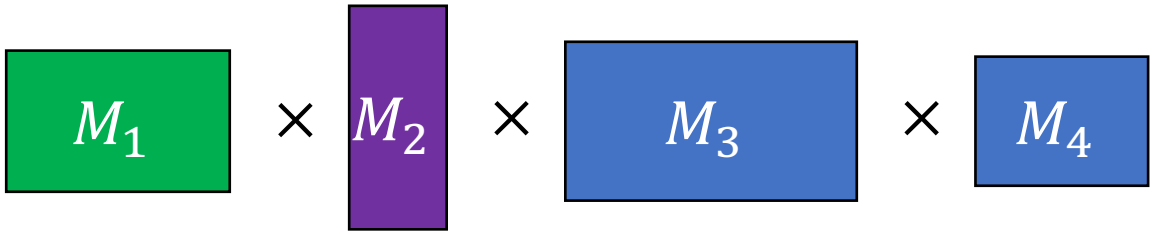
$n_1 n_2 n_3 = 1000$

$\text{Cost} = 1000$



$n_1 = 5 \quad n_2 = 10$

$n_3 = 20 \quad n_4 = 8 \quad n_5 = 6$



2. Select a Good Order for Solving Subproblems

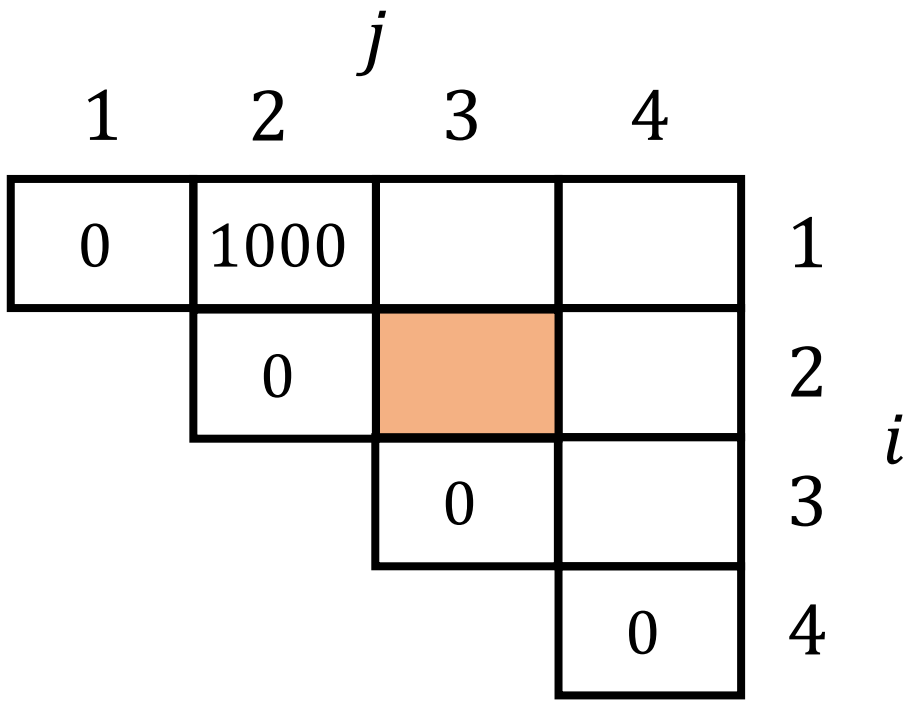
$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$$

$i = 2, j = 3$

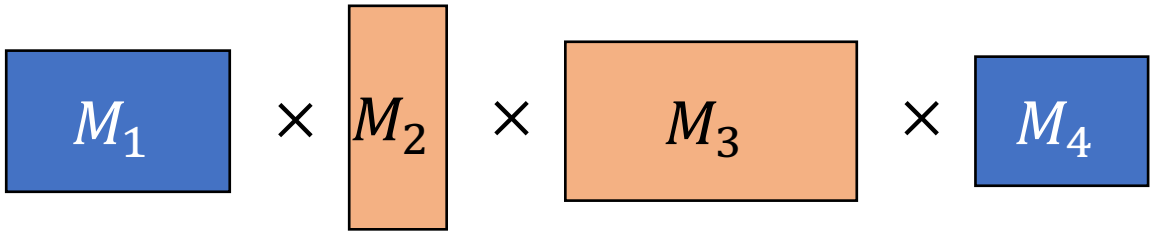
$k = 0$

$n_2 n_3 n_4 = 1600$



$n_1 = 5$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$ $n_5 = 6$



2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$$

$i = 2, j = 3$

$k = 0$

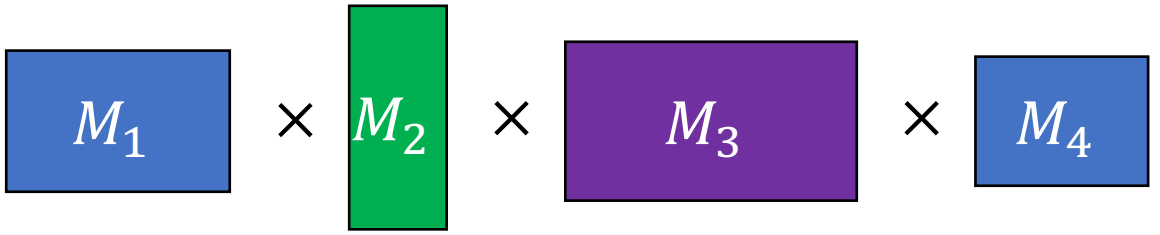
$n_2 n_3 n_4 = 1600$

$\text{Cost} = 1600$

i \ j	1	2	3	4
1	0	1000		
2		0	1600	
3			0	
4				0

$n_1 = 5$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$ $n_5 = 6$



2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$$

$i = 2, j = 3$

$k = 0$

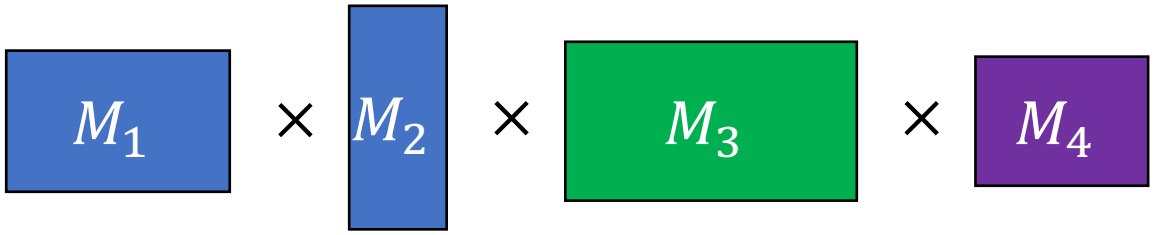
$n_3 n_4 n_5 = 960$

$\text{Cost} = 960$

i \ j	1	2	3	4
1	0	1000		
2		0	1600	
3			0	960
4				0

$n_1 = 5$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$ $n_5 = 6$



2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 3$

i \ j	1	2	3	4
1	0	1000		
2		0	1600	
3			0	960
4				0

$n_1 = 5$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$ $n_5 = 6$

$M_1 \times M_2 \times M_3 \times M_4$

2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 3$

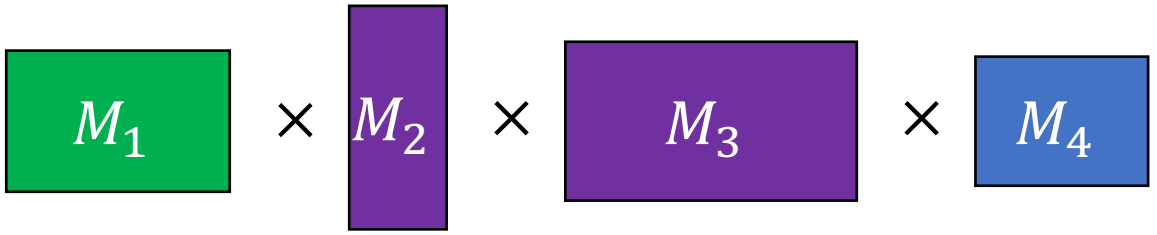
$k = 0$

$n_1 n_2 n_4 = 400$

$\text{Cost} = 2000$

i \ j	1	2	3	4
1	0	1000		
2		0	1600	
3			0	960
4				0

$n_1 = 5$ $n_2 = 10$ $n_3 = 20$ $n_4 = 8$ $n_5 = 6$



2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 3$

$k = 0$

$n_1 n_2 n_4 = 400$

$\text{Cost} = 2000$

i \ j	1	2	3	4
1	0	1000	1800	
2		0	1600	
3			0	960
4				0

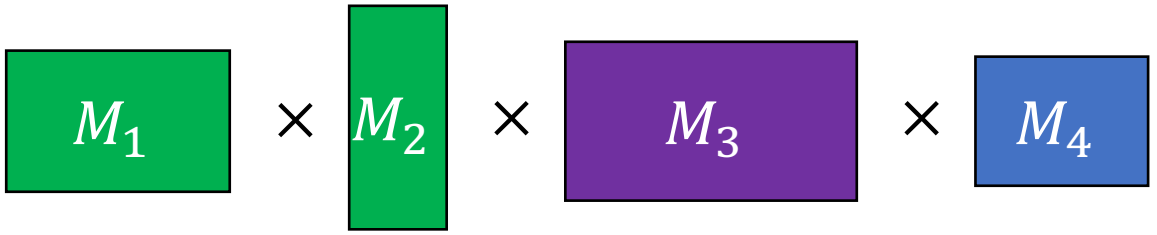
$k = 1$

$n_1 n_3 n_4 = 800$

$\text{Cost} = 1800$

$n_1 = 5 \quad n_2 = 10$

$n_3 = 20 \quad n_4 = 8 \quad n_5 = 6$



2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$$

$i = 2, j = 4$

i \ j	1	2	3	4
1	0	1000	1800	
2		0	1600	
3			0	960
4				0

$n_1 = 5$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$ $n_5 = 6$

$M_1 \times M_2 \times M_3 \times M_4$

2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$$

$i = 2, j = 4$

$k = 0$

$n_2 n_3 n_5 = 1200$

$\text{Cost} = 2160$

i \ j	1	2	3	4
1	0	1000	1800	
2		0	1600	
3			0	960
4				0

$n_1 = 5$ $n_2 = 10$

$n_3 = 20$ $n_4 = 8$ $n_5 = 6$

$M_1 \times M_2 \times M_3 \times M_4$

2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

$i = 2, j = 4$

$k = 0$

$n_2 n_3 n_5 = 1200$

$\text{Cost} = 2160$

i \ j	1	2	3	4
1	0	1000	1800	
2		0	1600	2080
3			0	960
4				0

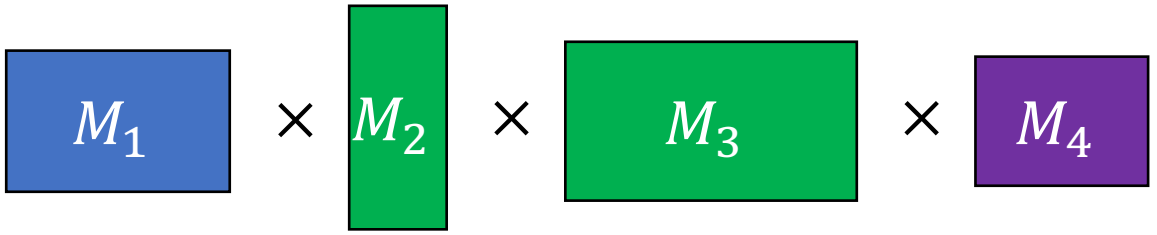
$k = 1$

$n_2 n_4 n_5 = 480$

$\text{Cost} = 2080$

$n_1 = 5 \quad n_2 = 10$

$n_3 = 20 \quad n_4 = 8 \quad n_5 = 6$



2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 4$

i \ j	1	2	3	4
1	0	1000	1800	
2		0	1600	2080
3			0	960
4				0

$n_1 = 5$ $n_2 = 10$
$n_3 = 20$ $n_4 = 8$ $n_5 = 6$

$M_1 \times M_2 \times M_3 \times M_4$

2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 4$

$k = 0$

$n_1 n_2 n_5 = 300$

$\text{Cost} = 2380$

i \ j	1	2	3	4
1	0	1000	1800	
2		0	1600	2080
3			0	960
4				0

$n_1 = 5$ $n_2 = 10$ $n_3 = 20$ $n_4 = 8$ $n_5 = 6$

$M_1 \times M_2 \times M_3 \times M_4$

2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 4$

$k = 0$

$n_1 n_2 n_5 = 300$

$\text{Cost} = 2380$

i \ j	1	2	3	4
1	0	1000	1800	
2		0	1600	2080
3			0	960
4				0

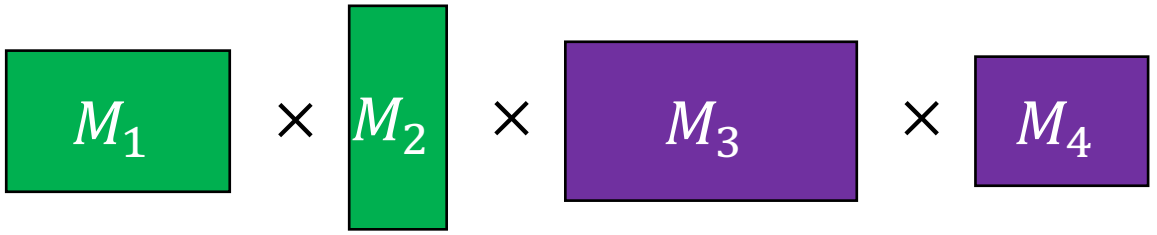
$k = 1$

$n_1 n_3 n_5 = 600$

$\text{Cost} = 2560$

$n_1 = 5 \quad n_2 = 10$

$n_3 = 20 \quad n_4 = 8 \quad n_5 = 6$



2. Select a Good Order for Solving Subproblems

$\text{Best}(i,i) = 0$

$$\text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

$i = 1, j = 4$

$k = 0$

$n_1 n_2 n_5 = 300$

$\text{Cost} = 2380$

i \ j	1	2	3	4
1	0	1000	1800	2040
2		0	1600	2080
3			0	960
4				0

$k = 1$

$n_1 n_3 n_5 = 600$

$\text{Cost} = 2560$

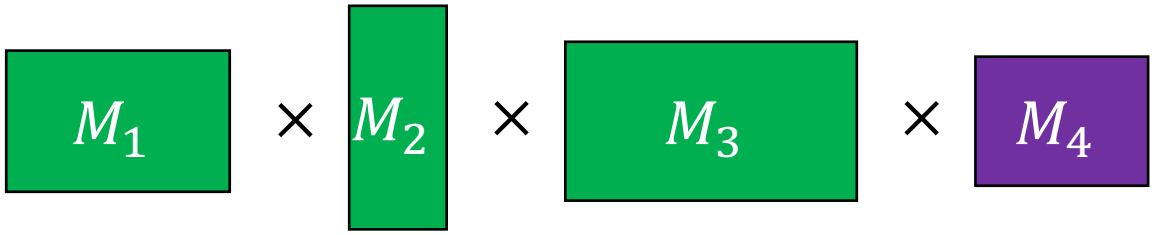
$k = 2$

$n_1 n_4 n_5 = 240$

$\text{Cost} = 2040$

$n_1 = 5 \quad n_2 = 10$

$n_3 = 20 \quad n_4 = 8 \quad n_5 = 6$



Dynamic Programming

Requires optimal substructure

- Solution to larger problem contains the solutions to smaller ones

General Blueprint:

1. Identify recursive structure of the problem
   - What is the “last thing” done?
2. Select a good order for solving subproblems
   - “Top Down:” Solve each problem recursively
   - “Bottom Up:” Iteratively solve each problem from smallest to largest
3. Save solution to each subproblem in memory

Run Time

1. Initialize $\text{Best}[i, i]$ to be all 0s
2. Starting at the main diagonal, working to the upper-right, fill in each cell using:
   - $\text{Best}(i, i) = 0$
   - $\text{Best}(i, j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i, i+k) + \text{Best}(i+k+1, j) + n_i n_{i+k+1} n_{j+1}$

$\Theta(n^2)$ cells in the array

$\Theta(n)$ options per cell

$\Theta(n^3)$ overall run time

Backtrack to Find the Best Order

$$\text{Best}(i,i) = 0 \qquad \text{Best}(i,j) = \min_{k=0,\ldots,j-i-1} \text{Best}(i,i+k) + \text{Best}(i+k+1,j) + n_i n_{i+k+1} n_{j+1}$$

"remember" which choice of k was the minimum at each cell

i \ j	1	2	3	4
1	0	1000 (0)	1800 (1)	2040 (2)
2		0	1600 (0)	2080 (1)
3			0	960 (0)
4				0

$n_1 = 5$ $n_2 = 10$
$n_3 = 20$ $n_4 = 8$ $n_5 = 6$