

# DYNAMIC PROGRAMMING

Bùi Tiến Lên

2023



KHOA CÔNG NGHỆ THÔNG TIN  
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

# Contents

---



1. **Dynamic Programming**

2. **Dynamic Programming and Optimization Problems**

3. **Workshop**



# Dynamic Programming

- 
- ```

graph TD
    6((6)) --> 5((5))
    6 --> 4((4))
    5 --> 4_5((4))
    5 --> 3_5((3))
    4_5 --> 3_4_5((3))
    4_5 --> 2_4_5((2))
    3_5 --> 2_3_5((2))
    3_5 --> 1_3_5((1))
    2_4_5 --> 1_2_4_5((1))
    2_4_5 --> 0_2_4_5((0))
    2_3_5 --> 1_2_3_5((1))
    2_3_5 --> 0_2_3_5((0))
    3_4_5 --> 2_3_4_5((2))
    3_4_5 --> 1_3_4_5((1))
    2_2_4_5 --> 1_2_2_4_5((1))
    2_2_4_5 --> 0_2_2_4_5((0))
    2_1_3_5 --> 1_2_1_3_5((1))
    2_1_3_5 --> 0_2_1_3_5((0))
    1_3_5 --> 0_1_3_5((0))
    1_3_5 --> 1_1_3_5((1))
    2_1_2_4_5 --> 1_2_1_2_4_5((1))
    2_1_2_4_5 --> 0_2_1_2_4_5((0))
    1_1_3_5 --> 0_1_1_3_5((0))
    1_1_3_5 --> 1_1_1_3_5((1))
    0_2_4_5 --> 1_0_2_4_5((1))
    0_2_4_5 --> 0_0_2_4_5((0))
    0_2_3_5 --> 1_0_2_3_5((1))
    0_2_3_5 --> 0_0_2_3_5((0))
    0_1_3_5 --> 1_0_1_3_5((1))
    0_1_3_5 --> 0_0_1_3_5((0))
    0_2_1_3_5 --> 1_0_2_1_3_5((1))
    0_2_1_3_5 --> 0_0_2_1_3_5((0))
    0_1_1_3_5 --> 1_0_1_1_3_5((1))
    0_1_1_3_5 --> 0_0_1_1_3_5((0))
    0_0_2_4_5 --> 1_0_0_2_4_5((1))
    0_0_2_4_5 --> 0_0_0_2_4_5((0))
    0_0_2_3_5 --> 1_0_0_2_3_5((1))
    0_0_2_3_5 --> 0_0_0_2_3_5((0))
    0_0_1_3_5 --> 1_0_0_1_3_5((1))
    0_0_1_3_5 --> 0_0_0_1_3_5((0))
    0_0_2_1_3_5 --> 1_0_0_2_1_3_5((1))
    0_0_2_1_3_5 --> 0_0_0_2_1_3_5((0))
    0_0_1_1_3_5 --> 1_0_0_1_1_3_5((1))
    0_0_1_1_3_5 --> 0_0_0_1_1_3_5((0))
    0_0_0_2_4_5 --> 1_0_0_0_2_4_5((1))
    0_0_0_2_4_5 --> 0_0_0_0_2_4_5((0))
    0_0_0_2_3_5 --> 1_0_0_0_2_3_5((1))
    0_0_0_2_3_5 --> 0_0_0_0_2_3_5((0))
    0_0_0_1_3_5 --> 1_0_0_0_1_3_5((1))
    0_0_0_1_3_5 --> 0_0_0_0_1_3_5((0))
    0_0_0_2_1_3_5 --> 1_0_0_0_2_1_3_5((1))
    0_0_0_2_1_3_5 --> 0_0_0_0_2_1_3_5((0))
    0_0_0_1_1_3_5 --> 1_0_0_0_1_1_3_5((1))
    0_0_0_1_1_3_5 --> 0_0_0_0_1_1_3_5((0))
    0_0_0_0_2_4_5 --> 1_0_0_0_0_2_4_5((1))
    0_0_0_0_2_4_5 --> 0_0_0_0_0_2_4_5((0))
    0_0_0_0_2_3_5 --> 1_0_0_0_0_2_3_5((1))
    0_0_0_0_2_3_5 --> 0_0_0_0_0_2_3_5((0))
    0_0_0_0_1_3_5 --> 1_0_0_0_0_1_3_5((1))
    0_0_0_0_1_3_5 --> 0_0_0_0_0_1_3_5((0))
    0_0_0_0_2_1_3_5 --> 1_0_0_0_0_2_1_3_5((1))
    0_0_0_0_2_1_3_5 --> 0_0_0_0_0_2_1_3_5((0))
    0_0_0_0_1_1_3_5 --> 1_0_0_0_0_1_1_3_5((1))
    0_0_0_0_1_1_3_5 --> 0_0_0_0_0_1_1_3_5((0))
    0_0_0_0_0_2_4_5 --> 1_0_0_0_0_0_2_4_5((1))
    0_0_0_0_0_2_4_5 --> 0_0_0_0_0_0_2_4_5((0))
    0_0_0_0_0_2_3_5 --> 1_0_0_0_0_0_2_3_5((1))
    0_0_0_0_0_2_3_5 --> 0_0_0_0_0_0_2_3_5((0))
    0_0_0_0_0_1_3_5 --> 1_0_0_0_0_0_1_3_5((1))
    0_0_0_0_0_1_3_5 --> 0_0_0_0_0_0_1_3_5((0))
    0_0_0_0_0_2_1_3_5 --> 1_0_0_0_0_0_2_1_3_5((1))
    0_0_0_0_0_2_1_3_5 --> 0_0_0_0_0_0_2_1_3_5((0))
    0_0_0_0_0_1_1_3_5 --> 1_0_0_0_0_0_1_1_3_5((1))
    0_0_0_0_0_1_1_3_5 --> 0_0_0_0_0_0_1_1_3_5((0))
    0_0_0_0_0_0_2_4_5 --> 1_0_0_0_0_0_0_2_4_5((1))
    0_0_0_0_0_0_2_4_5 --> 0_0_0_0_0_0_0_2_4_5((0))
    0_0_0_0_0_0_2_3_5 --> 1_0_0_0_0_0_0_2_3_5((1))
    0_0_0_0_0_0_2_3_5 --> 0_0_0_0_0_0_0_2_3_5((0))
    0_0_0_0_0_0_1_3_5 --> 1_0_0_0_0_0_0_1_3_5((1))
    0_0_0_0_0_0_1_3_5 --> 0_0_0_0_0_0_0_1_3_5((0))
    0_0_0_0_0_0_2_1_3_5 --> 1_0_0_0_0_0_0_2_1_3_5((1))
    0_0_0_0_0_0_2_1_3_5 --> 0_0_0_0_0_0_0_2_1_3_5((0))
    0_0_0_0_0_0_1_1_3_5 --> 1_0_0_0_0_0_0_1_1_3_5((1))
    0_0_0_0_0_0_1_1_3_5 --> 0_0_0_0_0_0_0_1_1_3_5((0))
    0_0_0_0_0_0_0_2_4_5 --> 1_0_0_0_0_0_0_0_2_4_5((1))
    0_0_0_0_0_0_0_2_4_5 --> 0_0_0_0_0_0_0_0_2_4_5((0))
    0_0_0_0_0_0_0_2_3_5 --> 1_0_0_0_0_0_0_0_2_3_5((1))
    0_0_0_0_0_0_0_2_3_5 --> 0_0_0_0_0_0_0_0_2_3_5((0))
    0_0_0_0_0_0_0_1_3_5 --> 1_0_0_0_0_0_0_0_1_3_5((1))
    0_0_0_0_0_0_0_1_3_5 --> 0_0_0_0_0_0_0_0_1_3_5((0))
    0_0_0_0_0_0_0_2_1_3_5 --> 1_0_0_0_0_0_0_0_2_1_3_5((1))
    0_0_0_0_0_0_0_2_1_3_5 --> 0_0_0_0_0_0_0_0_2_1_3_5((0))
    0_0_0_0_0_0_0_1_1_3_5 --> 1_0_0_0_0_0_0_0_1_1_3_5((1))
    0_0_0_0_0_0_0_1_1_3_5 --> 0_0_0_0_0_0_0_0_1_1_3_5((0))
    0_0_0_0_0_0_0_0_2_4_5 --> 1_0_0_0_0_0_0_0_0_2_4_5((1))
    0_0_0_0_0_0_0_0_2_4_5 --> 0_0_0_0_0_0_0_0_0_2_4_5((0))
    0_0_0_0_0_0_0_0_2_3_5 --> 1_0_0_0_0_0_0_0_0_2_3_5((1))
    0_0_0_0_0_0_0_0_2_3_5 --> 0_0_0_0_0_0_0_0_0_2_3_5((0))
    0_0_0_0_0_0_0_0_1_3_5 --> 1_0_0_0_0_0_0_0_0_1_3_5((1))
    0_0_0_0_0_0_0_0_1_3_5 --> 0_0_0_0_0_0_0_0_0_1_3_5((0))
    0_0_0_0_0_0_0_0_2_1_3_5 --> 1_0_0_0_0_0_0_0_0_2_1_3_5((1))
    0_0_0_0_0_0_0_0_2_1_3_5 --> 0_0_0_0_0_0_0_0_0_2_1_3_5((0))
    0_0_0_0_0_0_0_0_1_1_3_5 --> 1_0_0_0_0_0_0_0_0_1_1_3_5((1))
    0_0_0_0_0_0_0_0_1_1_3_5 --> 0_0_0_0_0_0_0_0_0_1_1_3_5((
```



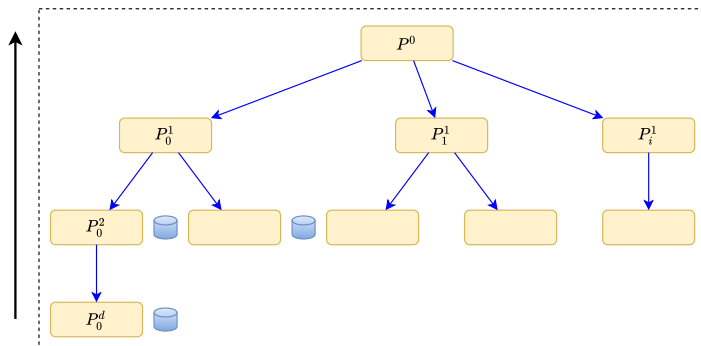
# Dynamic Programming

---

- Dynamic programming is similar to divide-and-conquer in that an instance of a problem is divided into smaller instances.
- However, in this approach we solve small instances first, **store** the results, and later, whenever we need a result, **look it up** instead of recomputing it.
- Dynamic programming is a **bottom-up** approach



# Dynamic Programming (cont.)





# Dynamic Programming (cont.)

The steps in the development of a dynamic programming algorithm are as follows:

1. **Establish** a recursive property that gives the solution to an instance of the problem.
2. **Solve** an instance of the problem in a bottom-up fashion by solving smaller instances first and storing the results
  - Using an array (or sequence of arrays)



# The Binomial Coefficient

- The **binomial coefficient** is given by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad \text{for } 0 \leq k \leq n \quad (1)$$

- We have recursive formula

$$\binom{n}{k} = \begin{cases} 1 & k = 0 \text{ or } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \end{cases} \quad (2)$$





# The Binomial Coefficient (cont.)

---

```
int bin(int n, int k) {  
    if ( k == 0 || k == n)  
        return 1;  
    else  
        return bin(n-1, k - 1) + bin(n - 1, k);  
}
```

- This algorithm is very inefficient.



# DP solution

- We define an array  $B[i][j]$  containing  $\binom{i}{j}$

1. Establish a recursive property.

$$B[i][j] = \begin{cases} 1 & j = 0 \text{ or } j = i \\ B[i-1][j-1] + B[i-1][j] & 0 < j < i \end{cases} \quad (3)$$

2. Solve an instance of the problem in a **bottom-up** fashion by computing the rows in  $B$  in sequence starting with the first row.



# DP solution (cont.)

|     | 0 | 1 | 2 | 3 | 4 | $j$ | $k$ |
|-----|---|---|---|---|---|-----|-----|
| 0   | 1 |   |   |   |   |     |     |
| 1   | 1 | 1 |   |   |   |     |     |
| 2   | 1 | 2 | 1 |   |   |     |     |
| 3   | 1 | 3 | 3 | 1 |   |     |     |
| 4   | 1 | 4 | 6 | 4 | 1 |     |     |
| $i$ |   |   |   |   |   |     |     |
| $n$ |   |   |   |   |   |     |     |

$$\begin{array}{ccc}
 B[i-1][j-1] & B[i-1][j] \\
 \downarrow & \downarrow \\
 \rightarrow & B[i][j]
 \end{array}$$

# Dynamic Programming and Optimization Problems





# Principle of Optimality

---

## Concept 1

A problem is said to satisfy the **Principle of Optimality** if the subsolutions of an optimal solution of the problem are themselves optimal solutions for their subproblems.

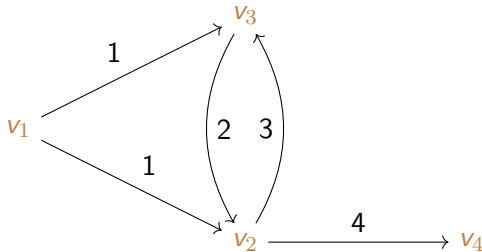
- In practice, it is necessary to show that the principle applies before assuming that an optimal solution can be obtained using dynamic programming.



# Longest Path Problem

Finding the longest simple paths from each vertex to all other vertices.

- The optimal (longest) simple path from  $v_1$  to  $v_4$  is  $\{v_1, v_3, v_2, v_4\}$ .
- However, the subpath  $\{v_1, v_3\}$  is not an optimal (longest) path from  $v_1$  to  $v_3$ .



# Dynamic Programming

---



1. **Establish** a recursive property that gives the optimal solution to an instance of the problem.
2. **Compute** the value of an optimal solution in a bottom-up fashion.
3. **Construct** an optimal solution in a bottom-up fashion.



# Chained Matrix Multiplication

- Multiply a  $2 \times 3$  matrix times a  $3 \times 4$  matrix, the resultant matrix is a  $2 \times 4$  matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 & 8 & 9 & 1 \\ 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 29 & 35 & 41 & 38 \\ 74 & 89 & 104 & 83 \end{bmatrix}$$

the total number of elementary multiplication is  $2 \times 3 \times 4 = 24$ .

- In general, to multiply an  $m \times n$  matrix times a  $n \times p$  matrix using the standard method, it is necessary to do

$$m \times n \times p \text{ elementary multiplications} \quad (4)$$





# Chained Matrix Multiplication (cont.)

- Consider the multiplication of the following four matrices:

$$\underbrace{A}_{20 \times 2} \times \underbrace{B}_{2 \times 30} \times \underbrace{C}_{30 \times 12} \times \underbrace{D}_{12 \times 8}$$

- Matrix multiplication is an associative operation, meaning that the order in which we multiply does not matter. There are five different orders in which we can multiply four matrices, each possibly resulting in a different number of elementary multiplications.

|            |                                                                          |           |
|------------|--------------------------------------------------------------------------|-----------|
| $A(B(CD))$ | $30 \times 12 \times 8 + 2 \times 30 \times 8 + 20 \times 2 \times 8$    | $= 3680$  |
| $(AB)(CD)$ | $20 \times 2 \times 30 + 30 \times 12 \times 8 + 20 \times 30 \times 8$  | $= 8880$  |
| $A((BC)D)$ | $2 \times 30 \times 12 + 2 \times 12 \times 8 + 20 \times 2 \times 8$    | $= 1232$  |
| $(AB)C)D$  | $20 \times 2 \times 30 + 20 \times 30 \times 12 + 20 \times 12 \times 8$ | $= 10320$ |
| $(A(BC))D$ | $2 \times 30 \times 12 + 20 \times 2 \times 12 + 20 \times 12 \times 8$  | $= 3120$  |



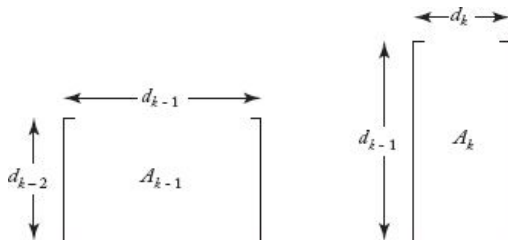
# Chained Matrix Multiplication (cont.)

Multiply  $n$  matrices:  $A_1, A_2, \dots, A_n$ .

- It is not hard to see that the principle of optimality applies in this problem.

Let

- $d_0$  be the number of rows in  $A_1$  and
- $d_k$  be the number of columns in  $A_k$  for  $1 \leq k \leq n$ , the dimension of  $A_k$  is  $d_{k-1} \times d_k$





# Solution to Chained Matrix Multiplication

- Let  $M[i][j]$  = minimum number of multiplications needed to multiply  $A_i$  through  $A_j$ , if  $i \leq j$ .
- Principle of optimality

$$\begin{aligned} M[i][i] &= 0 \\ M[i][j] &= \min_{i \leq k \leq j-1} (M[i][k] + M[k+1][j] + d_{i-1}d_kd_j) \quad \text{if } i < j \end{aligned} \quad (5)$$

# Solution to Chained Matrix Multiplication (cont.)



- Consider the multiplication of the following six matrices

$$\underbrace{A_1}_{5 \times 2} \times \underbrace{A_2}_{2 \times 3} \times \underbrace{A_3}_{3 \times 4} \times \underbrace{A_4}_{4 \times 6} \times \underbrace{A_5}_{6 \times 7} \times \underbrace{A_6}_{7 \times 8}$$

we have  $d_0 = 5, d_1 = 2, d_2 = 3, d_3 = 4, d_4 = 6, d_5 = 7, d_6 = 8$

$M =$

|   | 1 | 2  | 3  | 4   | 5   | 6   |
|---|---|----|----|-----|-----|-----|
| 1 | 0 | 30 | 64 | 132 | 226 | 348 |
| 2 |   | 0  | 24 | 72  | 156 | 268 |
| 3 |   |    | 0  | 72  | 198 | 366 |
| 4 |   |    |    | 0   | 168 | 392 |
| 5 |   |    |    |     | 0   | 336 |
| 6 |   |    |    |     |     | 0   |



# Workshop



# Quiz

---



## 1. What is the dynamic programming?

.....

.....

.....



# Exercises

---



- Write a program

# References

---



Deitel, P. (2016).

*C++: How to program.*

Pearson.



Gaddis, T. (2014).

*Starting Out with C++ from Control Structures to Objects.*

Addison-Wesley Professional, 8th edition.



Jones, B. (2014).

*Sams teach yourself C++ in one hour a day.*

Sams.