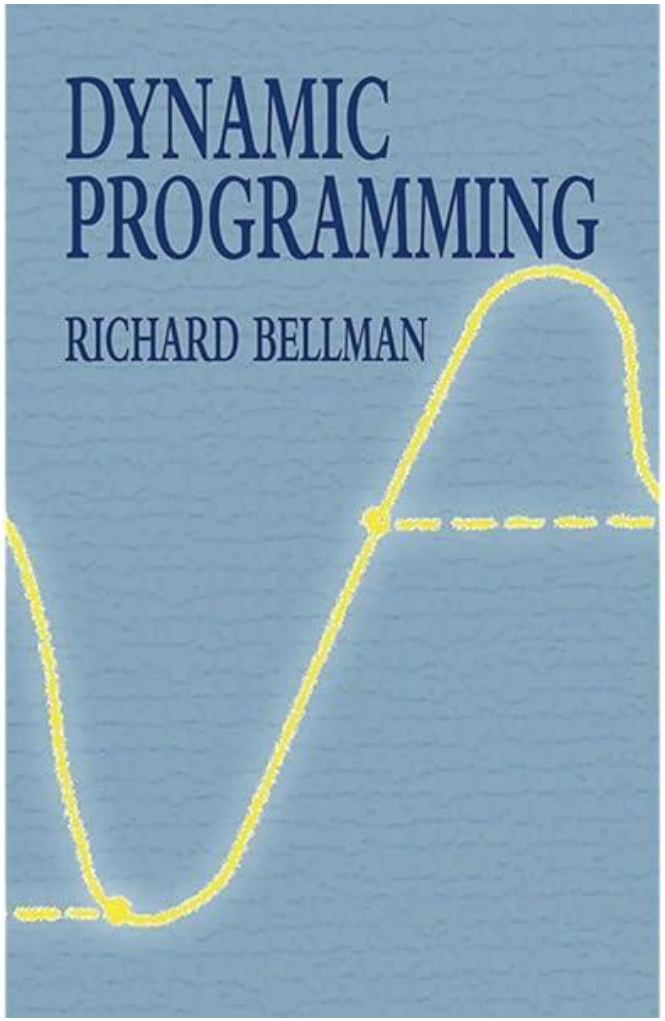
Dynamic Programming

Those who cannot remember the past
are condemned to repeat it.

-Dynamic Programming

Dynamic Programming



Idea of Dynamic Programming was first used in 1951 by Richard Bellman

Dynamic Programming: What

Dynamic programming is both a mathematical optimization method and an algorithmic paradigm.

Dynamic Programming: HOW

It divides the problem into smaller subproblems in recursive manner

Stores the intermediate computations into table(Memoization)

In future, Whenever these computations are needed it lookup the table and fetch the value instead of recomputing it from the scratch

Dynamic Programming: When

When a problem has:

- ❑ **Optimal Substructure Property**
- ❑ **Overlapping Subproblem**

1.Optimal Substructure Property

1. An optimal solution can be constructed from optimal solutions of its subproblems

Dynamic Programming: When

2.Overlapping Sub-Problem

Overlapping subproblems means that we're recomputing the same thing more than once

Dynamic Programming

Principal of optimality

An optimal solution to a problem can be constructed from optimal solutions to its subproblems

OR

In other words, if you have a complex problem that can be broken down into smaller subproblems, the best way to solve the overall problem is to find the best solution for each subproblem and then use those solutions to construct the optimal solution for the whole problem.

This principle is a key idea behind dynamic programming algorithms

Dynamic Programming

Applications

- Finding Binomial Coefficient
- 0/1 Knapsack Problem
- Matrix Chain multiplication
- Longest Common Subsequence
- All pair Shortest Path: Floyd Warshall Algorithm
- Largest Divisible Set

Dynamic Programming

Binomial Co-efficient

A binomial coefficient C(n, k) can be defined as the coefficient of x^k in the expansion of $(1 + x)^n$

OR

A binomial coefficient C(n, k) also gives the number of ways, disregarding order, that k objects can be chosen from among n objects

Finding Binomial Coefficient

Recursive Formulation

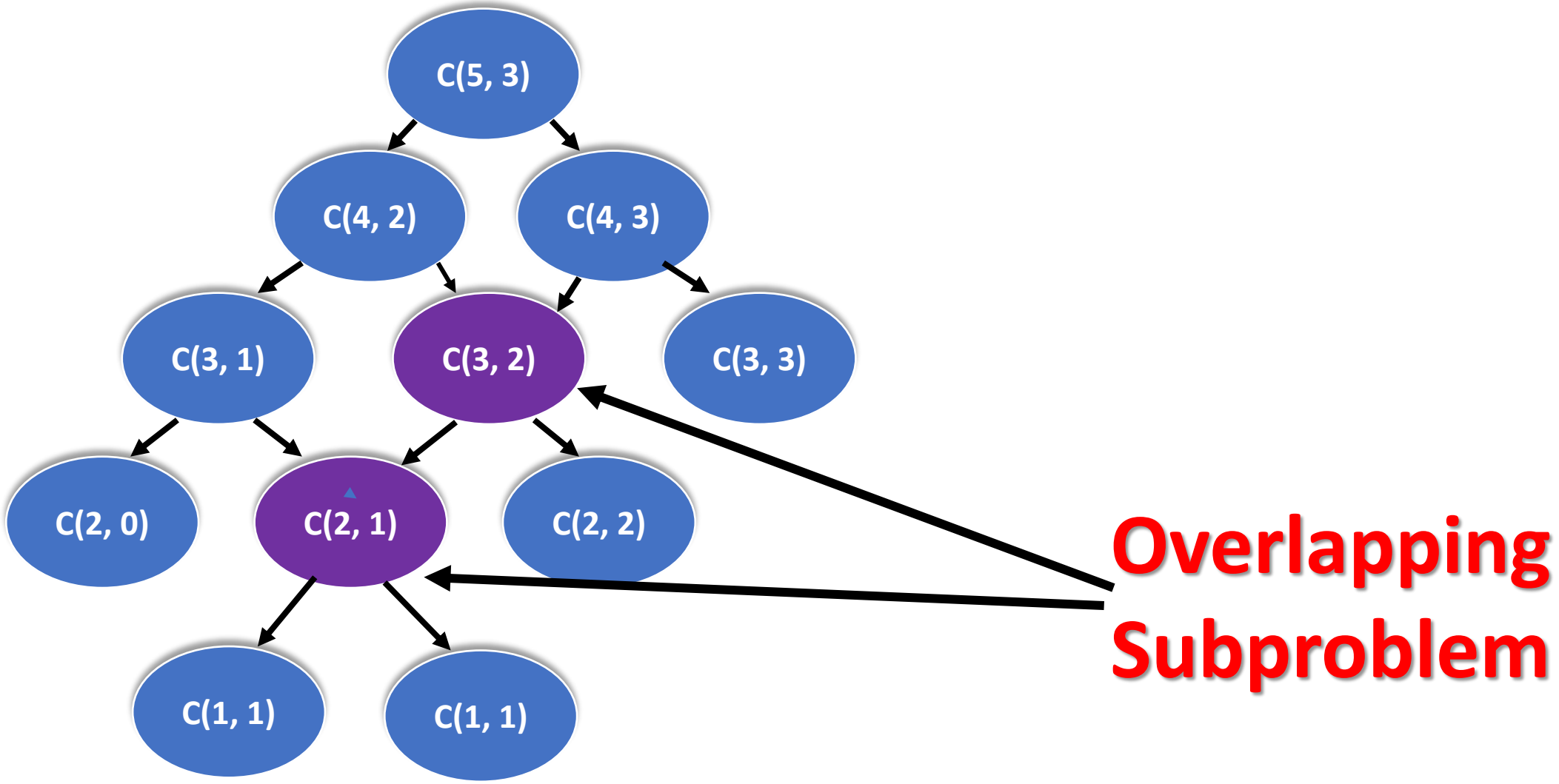
Base Case:

$$C(n,0) = C(n, n) = 1$$

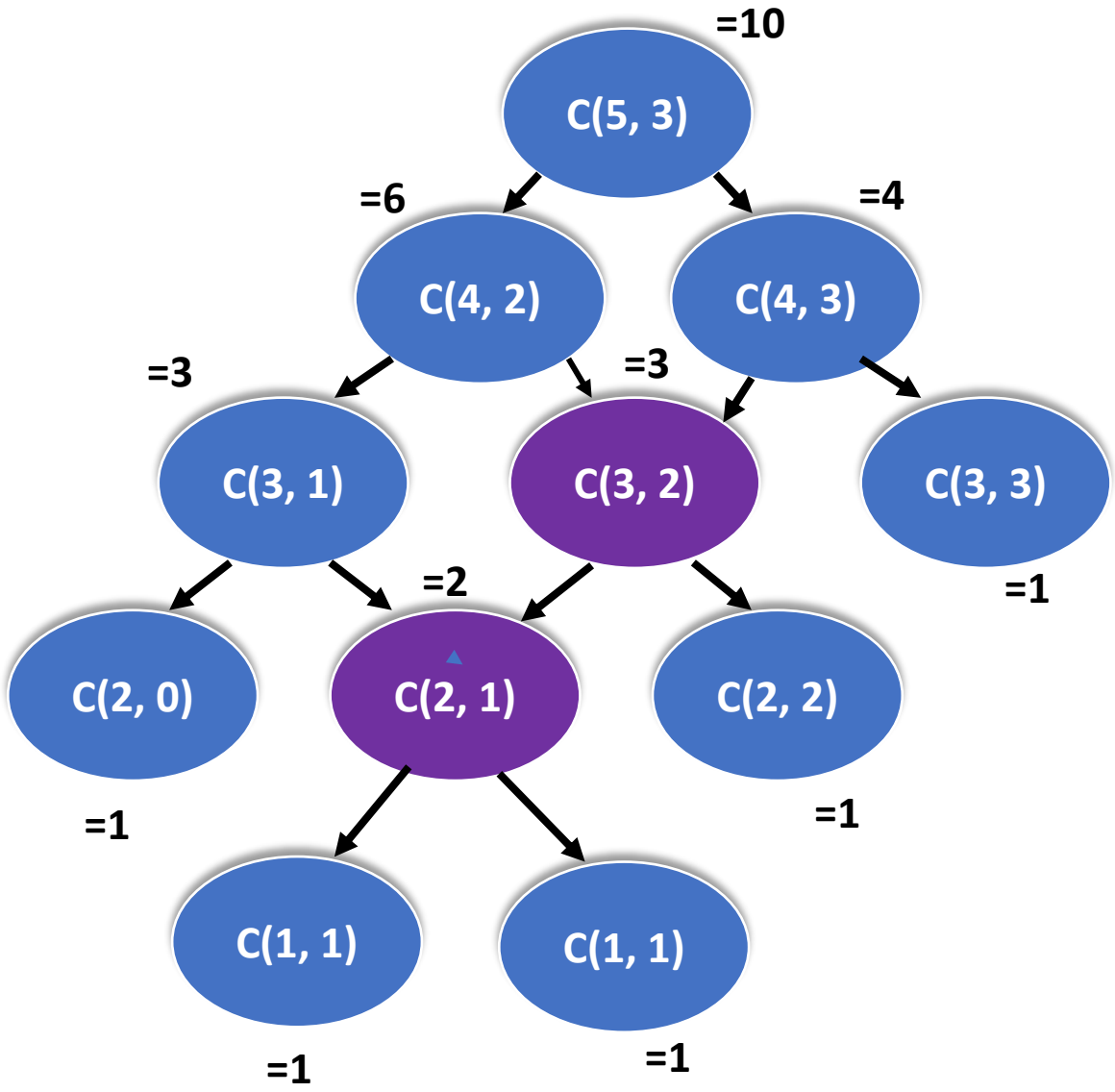
Recursive Case:

$$C(n, k) = C(n-1, k) + C(n-1, k-1)$$

Finding Binomial Coefficient



Finding Binomial Coefficient



$$C(5, 3) = 10$$

Finding Binomial Coefficient

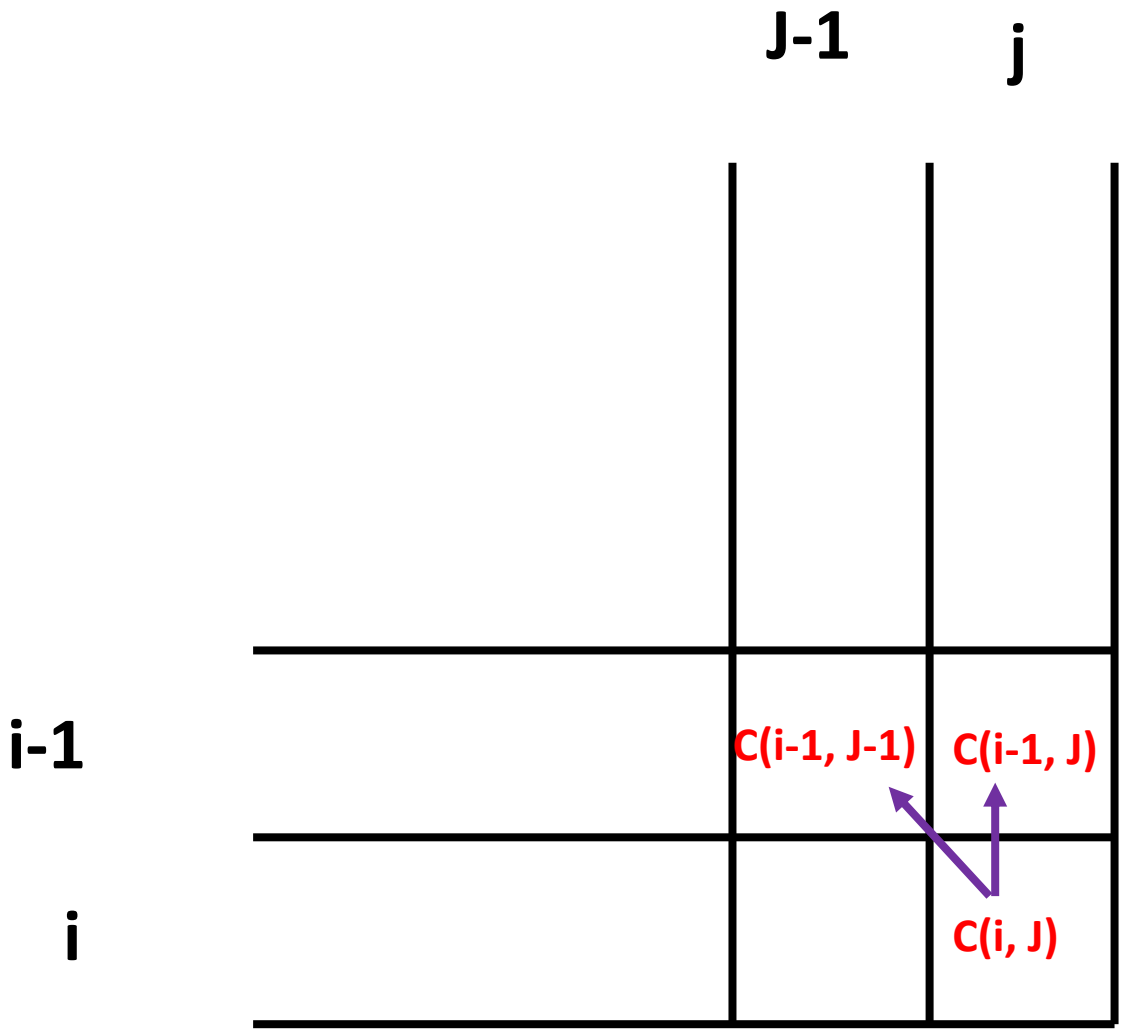
Algorithm 1 DP approach on Pascal's recursion.

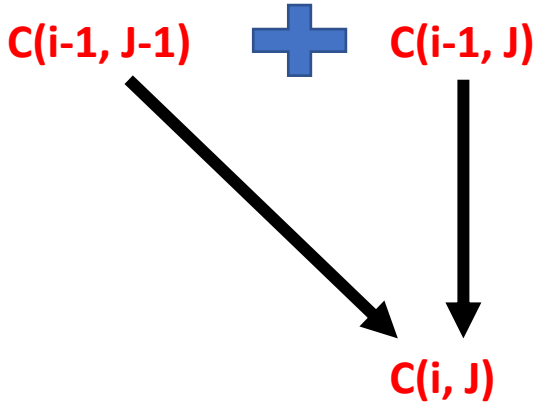
input : values of n and k

output: the computed binomial coefficient $\binom{n}{k}$

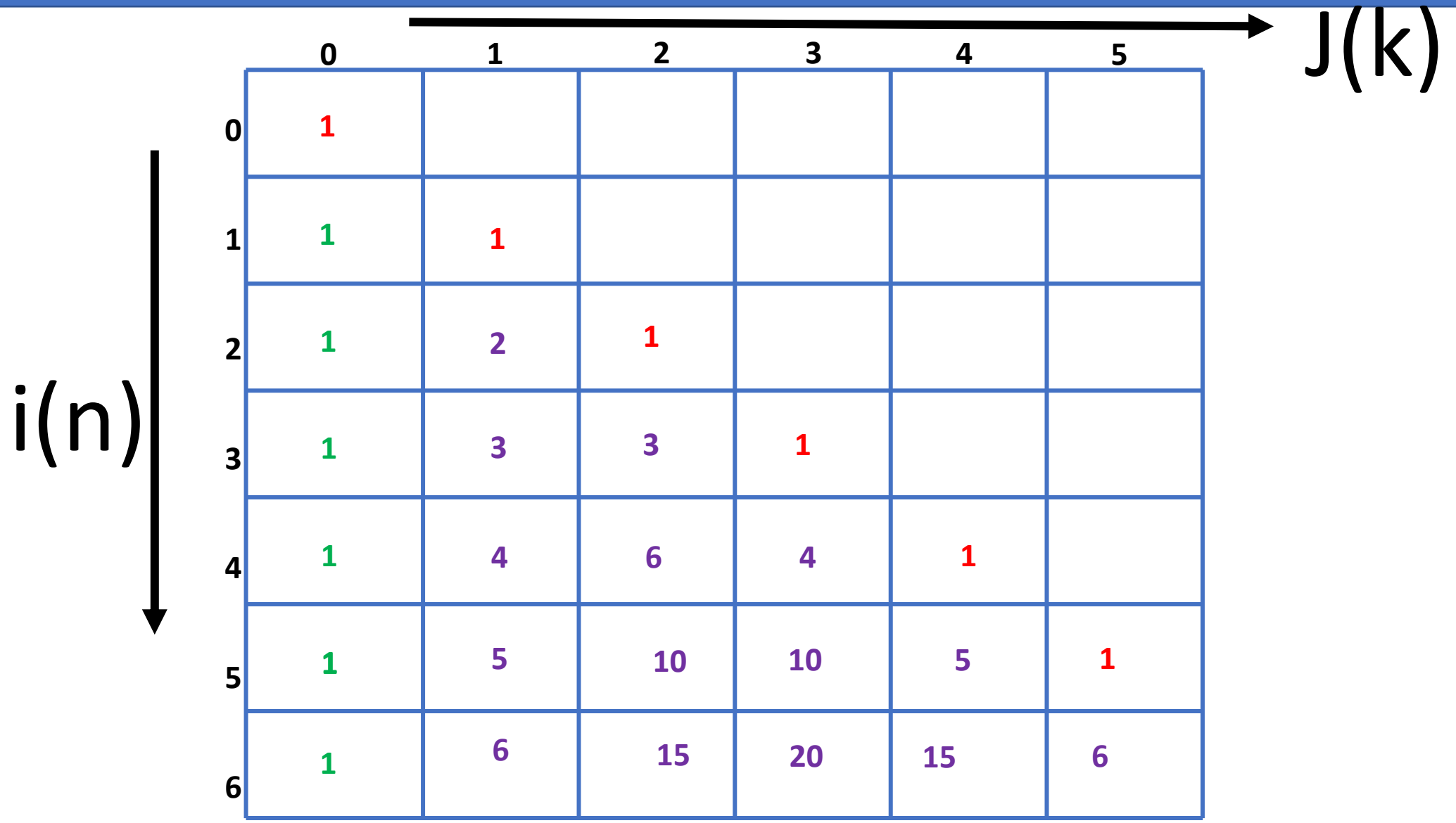
```
for i ← 0 to n do
    fill each row at a time, computing only the values that will be used for
    j ← 0 to min(i, k) do
        if j = 0 ∨ j = n then
            C[i, j] ← 1
        else
            use the recursive relation C[i, j] ← C[i − 1, j − 1] + C[i − 1, j]
        end
    end
end
return C[n, k]
```

Finding Binomial Coefficient





Finding Binomial Coefficient



Finding Binomial Coefficient

Time Complexity = O(Size of table x Time to fill one slot)

= O(nxn x O(1))

= $O(n^2)$

Space Complexity = O(Size of table) = $O(n^2)$

0/1 Knapsack Problem

Maximize $\sum_{1 \le i \le n} p_i * x_i$

Subject to $\sum_{i=1}^{n} w_i * x_i \le M$

Where $x_i \epsilon \{0, 1\}$ and $1 \le i \le n$

0/1 Knapsack Problem

In general n objects with their weight and profit are given to us. A knapsack with capacity M is also available.

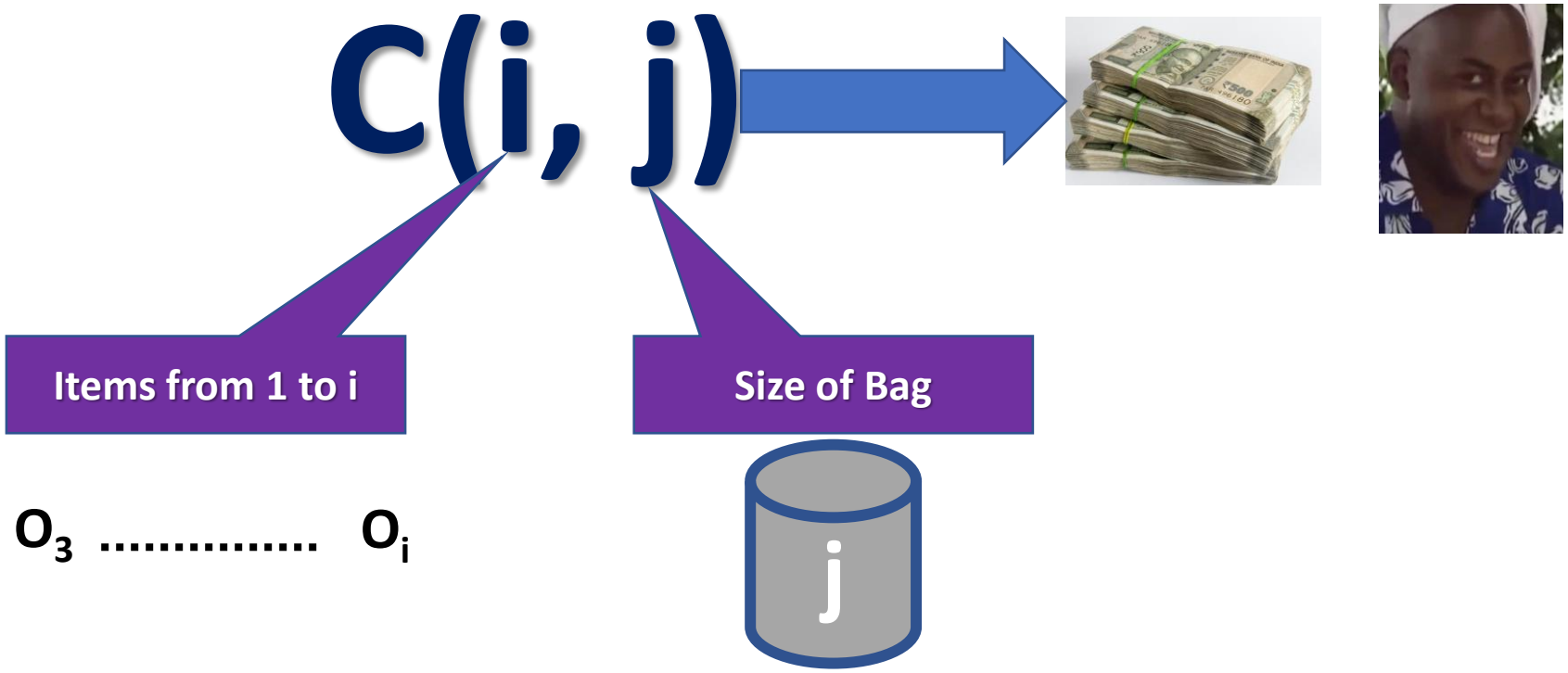
O_i	O_1	O_2	O_3	.....	O_i	.....	O_n
W_i	w_1	w_2	w_3	.....	w_i	.....	w_n
P_i	p_1	p_2	p_3	.....	p_i	.....	p_n

Goal: Fill the knapsack with given objects in such a way so that overall earned profit is maximum.

0/1 Knapsack Problem

Recursive Formulation:

C(i, j) = The ***maximum profit*** of the selected items if we can take **items 1 to i** and **size of bag j**



0/1 Knapsack Problem

Recursive Formulation:

C(i, j) = The ***maximum profit*** of the selected items if we can take **items 1 to i** and **size of bag j**

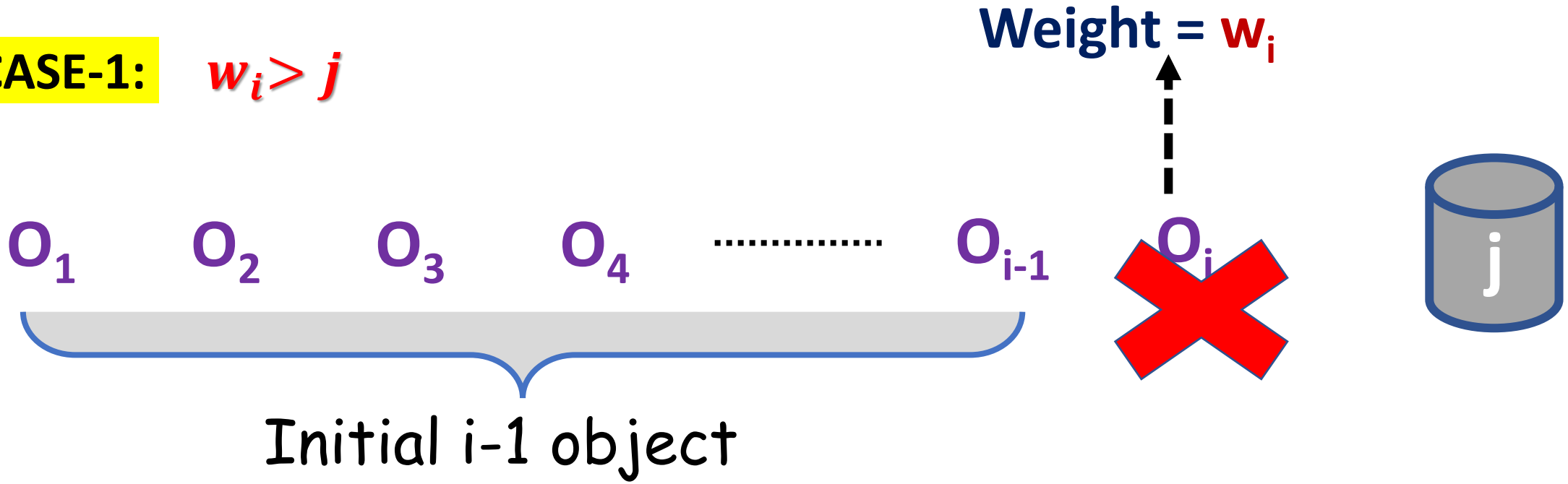
Base Case :

C(i, j) = 0 if i=0 or j=0

0/1 Knapsack Problem

Recursive Case :

CASE-1: $w_i > j$

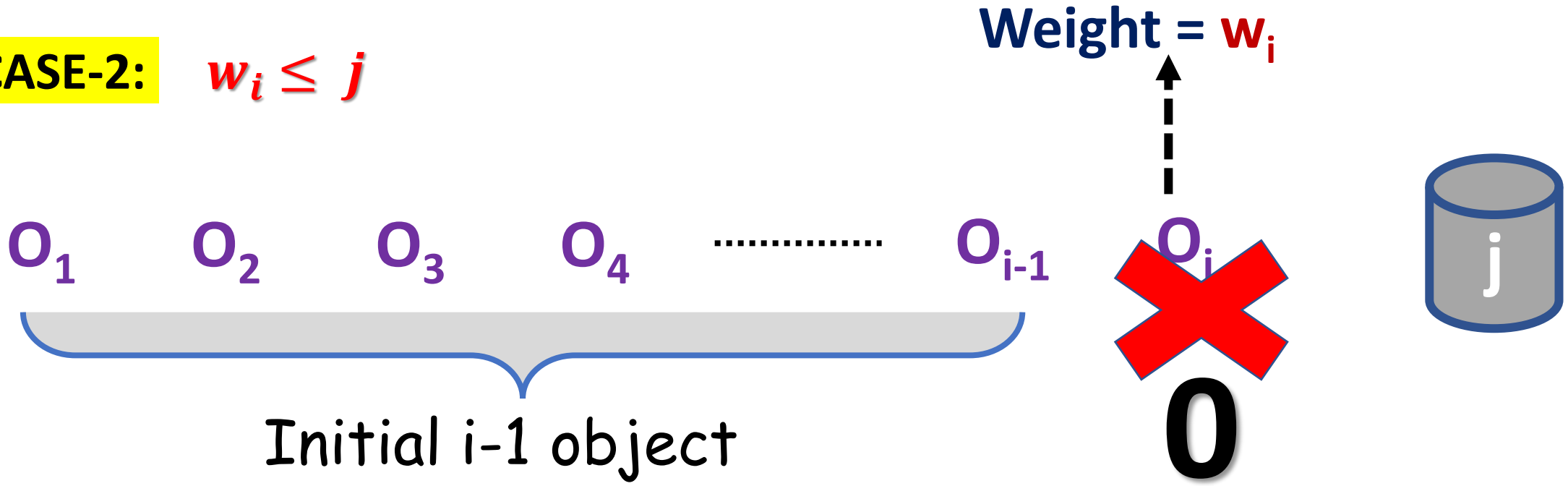


$$C(i, j) = C(i-1, j)$$

0/1 Knapsack Problem

Recursive Case :

CASE-2: $w_i \leq j$



C(i, j) = C(i-1, j)

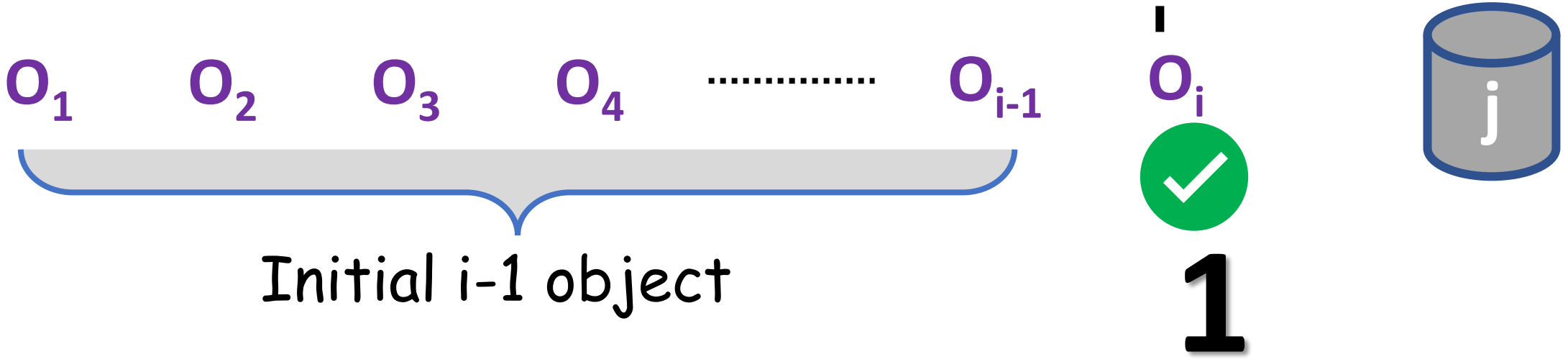
0/1 Knapsack Problem

Recursive Case :

CASE-2: $w_i \leq j$

Profit= p_i

Weight = w_i



$$C(i, j) = C(i-1, j- w_i) + p_i$$

0/1 Knapsack Problem

Recursive Formulation:

C(i, j) = The maximum profit of the selected items if we can take items 1 to i and size of bag j.

Base Case :

$$C(i, j) = 0 \text{ if } i=0 \text{ or } j=0$$

Recursive Case :

$$C(i, j) = \begin{cases} c(i-1, j) & \text{if } w_i > j \\ Max \begin{cases} c(i-1, j) & \text{if } w_i \leq j \\ c(i-1, j-w_i) + p_i \end{cases} \end{cases}$$

0/1 Knapsack Problem

For the given set of items and knapsack capacity = 5 kg, find the optimal solution for the 0/1 knapsack problem making use of dynamic programming approach.

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

0/1 Knapsack Problem

C(i, j)	0	1	2	3	4	5
0	0	0	0	0	0	0
O1 1	0	?				
O2 2	0					
O3 3	0					
O4 4	0					

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

C(1, 1) = C(0, 1) as w1 = 2 > j=1

= 0

0/1 Knapsack Problem

C(i, j)

		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	?			
O2	2	0					
O3	3	0					
O4	4	0					

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

$$C(1,2) = \max\{ C(0, 2), C(0, 0) + 3 \} \quad \text{as } w1 = 2 = j = 2$$
$$= \max\{ 0, 0 + 3\} = 3$$

0/1 Knapsack Problem

C(i, j)		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	?		
O2	2	0					
O3	3	0					
O4	4	0					

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

$$C(1,3) = \max\{ C(0, 3), C(0, 1) + 3 \} \quad \text{as } w1 = 2 < j = 3$$
$$= \max\{ 0, 0 + 3\} = 3$$

0/1 Knapsack Problem

C(i, j)		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0					
O3	3	0					
O4	4	0					

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

0/1 Knapsack Problem

C(i, j)

		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	?				
O3	3	0					
O4	4	0					

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

C(2, 1) = C(1, 1) as w2 =3 > j =1

= 0

0/1 Knapsack Problem

C(i, j)

		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	?			
O3	3	0					
O4	4	0					

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

C(2, 2) = C(1, 2) *as w2 =3 > j =2*

= 3

0/1 Knapsack Problem

C(i, j)

		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	3	?		
O3	3	0					
O4	4	0					

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

$C(2, 3) = max\{C(1, 3), C(1, 0) + 4$ as $w2 = 3 = j = 3$

$= max\{3 , 0 + 4\} = 4$

0/1 Knapsack Problem

C(i, j)		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	3	4	4	
O3	3	0					
O4	4	0					

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

$$C(2, 4) = \max\{C(1, 4), C(1, 1) + 4 \quad \text{as } w2 = 3 > j = 4$$

$$= \max\{3 , 0 + 4\} = 4$$

0/1 Knapsack Problem

C(i, j)		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	3	4	4	?
O3	3	0					
O4	4	0					

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

$$C(2, 5) = \max\{C(1, 4), C(1, 2) + 4 \quad \text{as } w2 = 3 > j = 4$$
$$= \max\{3 , 3 + 4\} = 7$$

0/1 Knapsack Problem

C(i, j)

		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	3	4	4	7
O3	3	0					
O4	4	0					

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

0/1 Knapsack Problem

C(i, j)

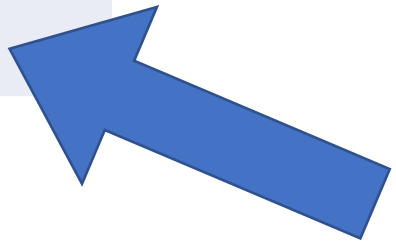
		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	3	4	4	7
O3	3	0	0	0	3	5	7
O4	4	0	0	0	3	5	7

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

0/1 Knapsack Problem

C(i, j)		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	3	4	4	7
O3	3	0	0	0	3	5	7
O4	4	0	0	0	3	5	7

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6



Maximum Profit = 7

0/1 Knapsack Problem

```
function knapsackDP(weights, values, capacity, n):
1.      C = a new table of size (n+1) x (capacity+1)
2.      for i from 0 to n:
3.          C[i][0] = 0
4.      for j from 0 to capacity:
5.          C[0][j] = 0
6.      for i from 1 to n:
7.          for j from 1 to capacity:
8.              if weights[i] > w:
9.                  C[i][j] = C[i-1][j]
10.             else:
11.                 C[i][j] = max(C[i-1][j], C[i-1][j - w_i]+p_i)
12.     return C[n][capacity]
```

0/1 Knapsack Problem

Time Complexity = O(Size of Table x Time to fill one slot)

= O(m x n x O(1))

= **O(mn)**

m = Number of items

n = Capacity of knapsack

0/1 Knapsack Problem

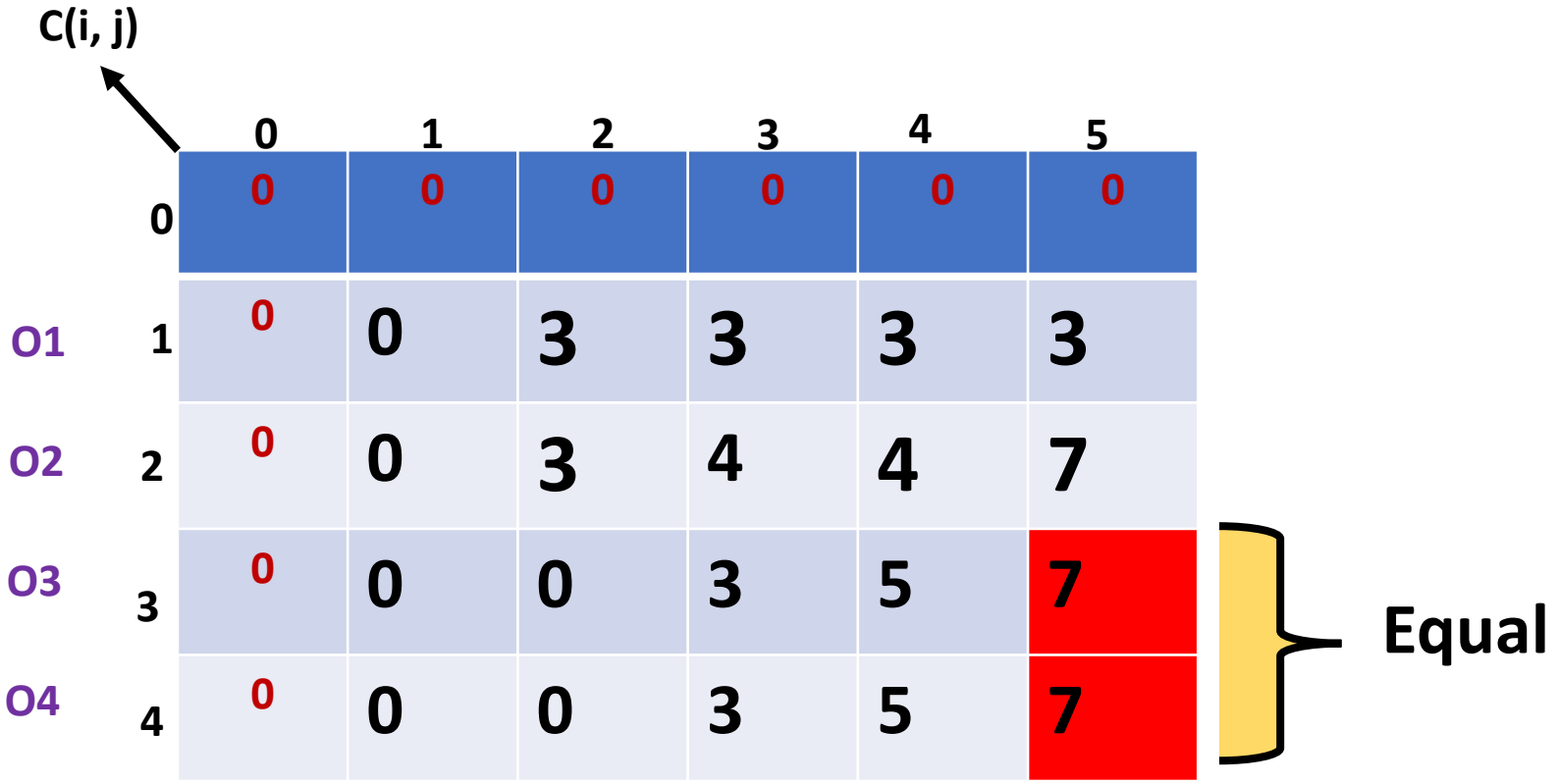
Trace back to find the item

C(i, j)

		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	3	4	4	7
O3	3	0	0	0	3	5	7
O4	4	0	0	0	3	5	7

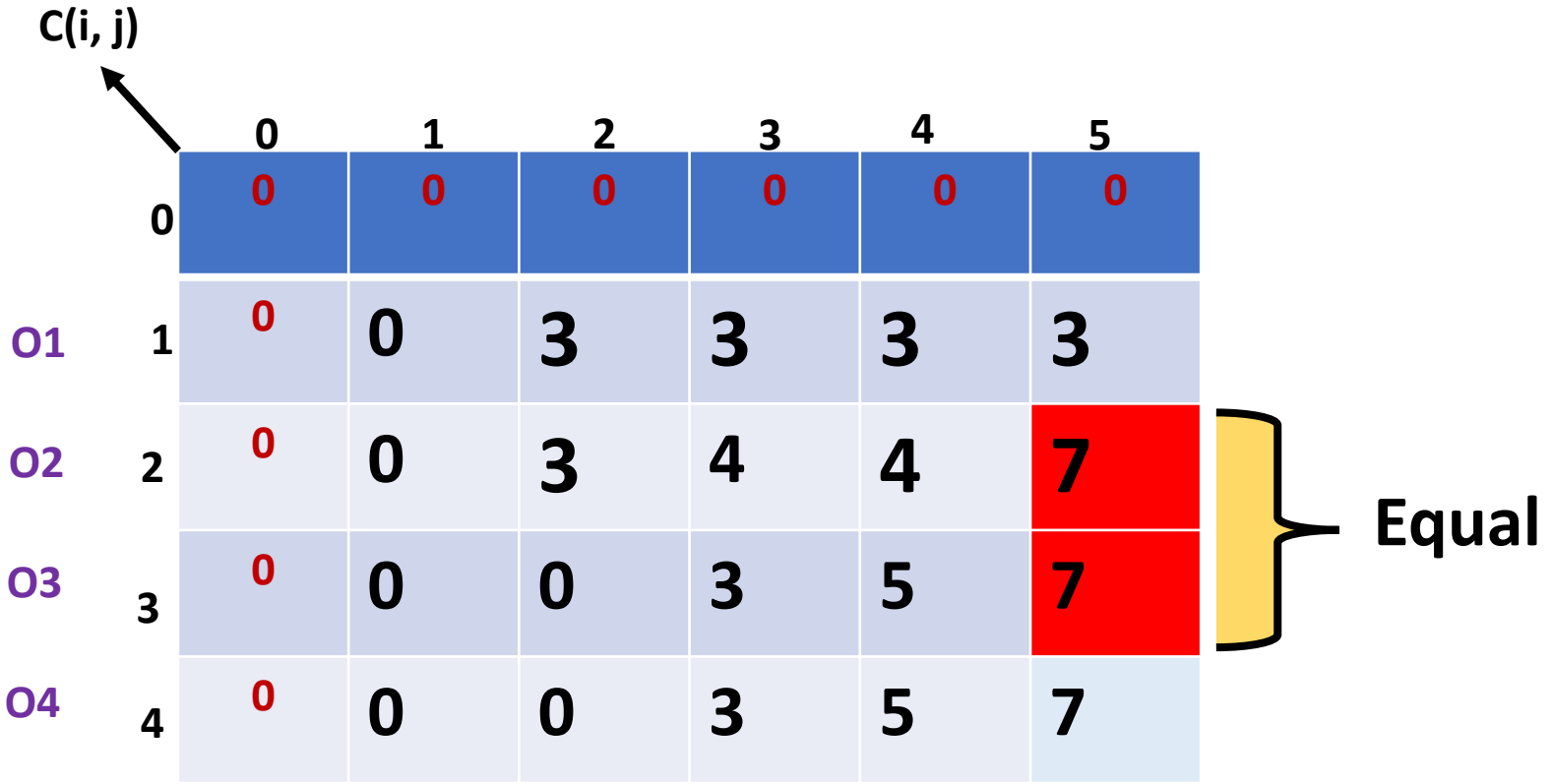
0/1 Knapsack Problem

Trace back to find the item



0/1 Knapsack Problem

Trace back to find the item



0/1 Knapsack Problem

Trace back to find the item

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

C(i, j)

		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	3	4	4	7
O3	3	0	0	0	3	5	7
O4	4	0	0	0	3	5	7

Not Equal

O2 is selected

0/1 Knapsack Problem

Trace back to find the item

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

C(i, j)

		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	3	4	4	7
O3	3	0	0	0	3	5	7
O4	4	0	0	0	3	5	7

0/1 Knapsack Problem

Trace back to find the item

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

C(i, j)

		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	3	4	4	7
O3	3	0	0	0	3	5	7
O4	4	0	0	0	3	5	7

Not Equal

O1 is selected

0/1 Knapsack Problem

Trace back to find the item

ITEM	WEIGHT	PROFIT
1	2	3
2	3	4
3	4	5
4	5	6

C(i, j)		0	1	2	3	4	5
	0	0	0	0	0	0	0
O1	1	0	0	3	3	3	3
O2	2	0	0	3	4	4	7
O3	3	0	0	0	3	5	7
O4	4	0	0	0	3	5	7

O_1 and O_2 is selected

0/1 Knapsack Problem

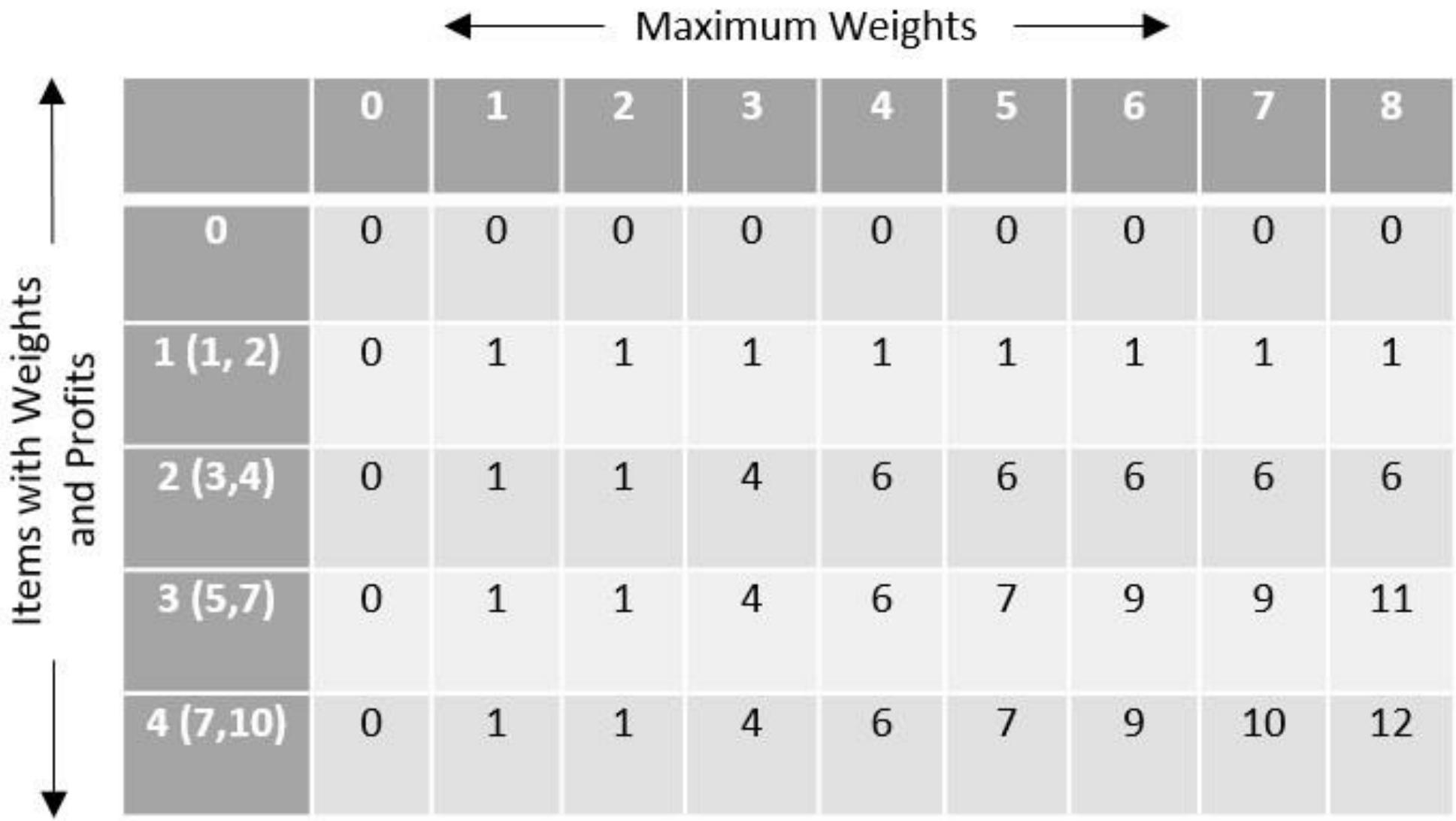
Trace back to find the item

Starting from the **bottom-right corner** of the table (i.e., C[n][W]), follow these steps:

1. If `C[i][w] != C[i-1][w]`, it means that item i was selected. Add item i to the list of selected items.
2. Move to the cell `C[i-1][w - weight[i]]` and repeat the process for the remaining capacity and items.
3. Continue this process until you reach the **top-left corner** of the table (i.e., C[0][0]).

Let us consider that the capacity of the knapsack is W = 8 and the items are as shown in the following table.

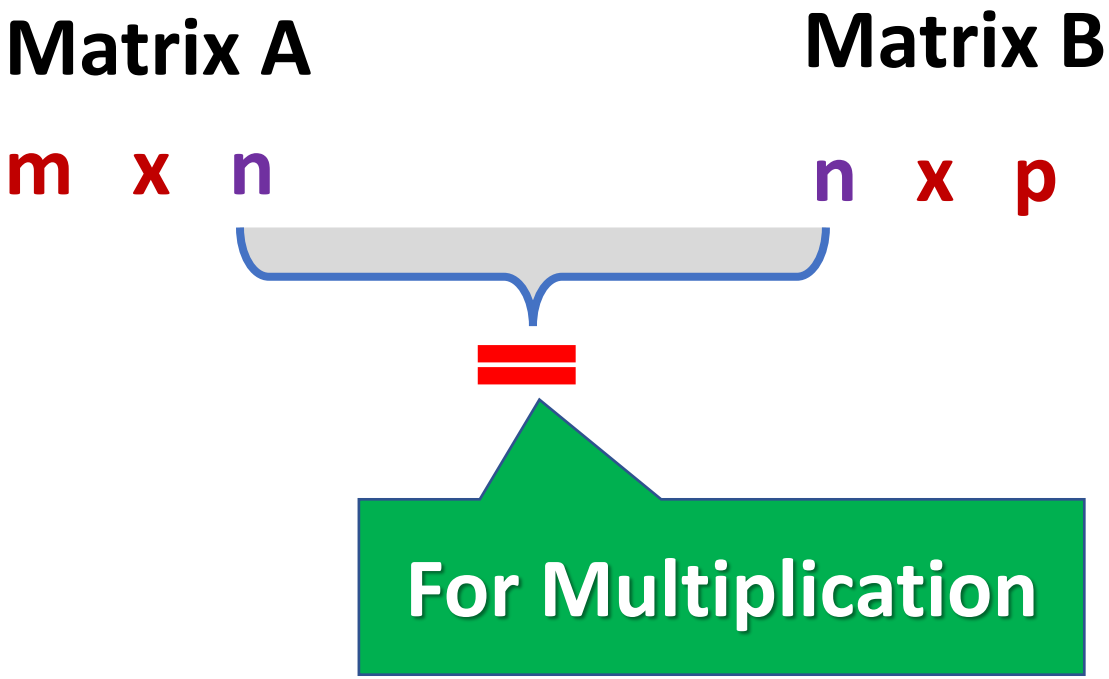
Item	**A**	**B**	**C**	**D**
Profit	2	4	7	10
Weight	1	3	5	7



	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1 (1, 2)	0	1	1	1	1	1	1	1	1
2 (3,4)	0	1	1	4	6	6	6	6	6
3 (5,7)	0	1	1	4	6	7	9	9	11
4 (7,10)	0	1	1	4	6	7	9	10	12

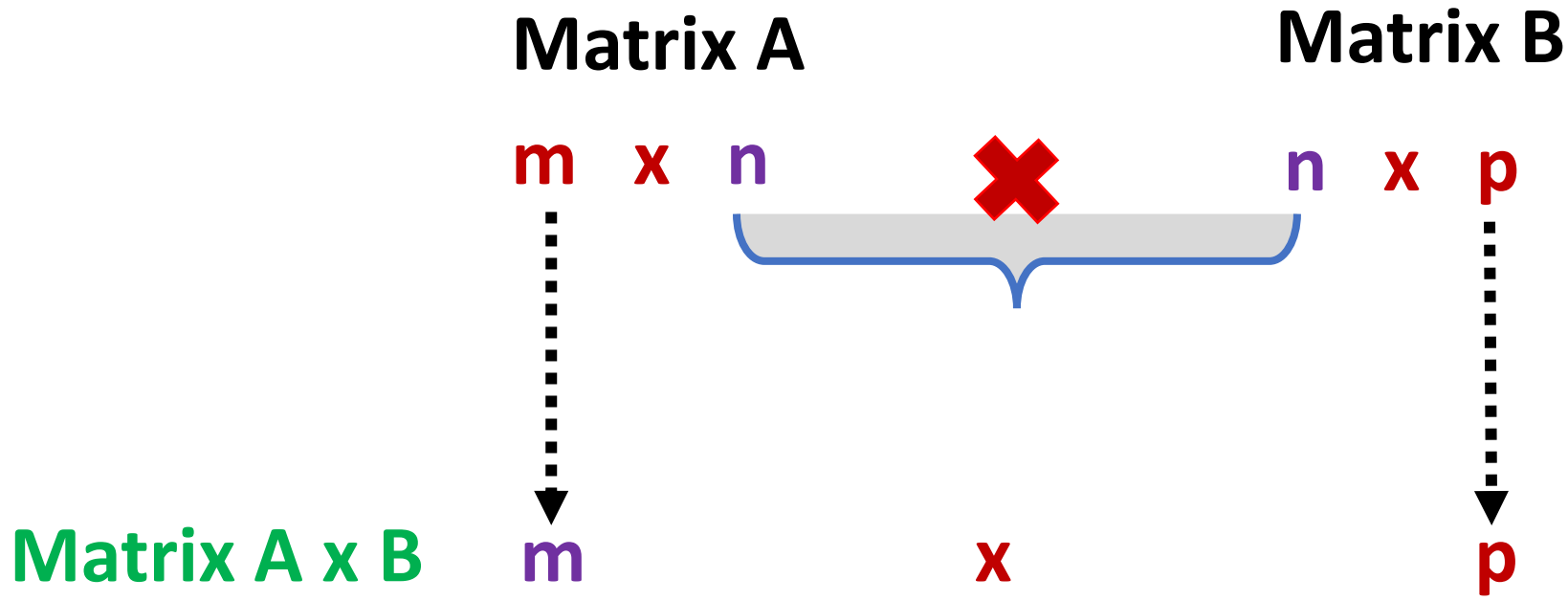
Matrix Chain Multiplication

Basics:



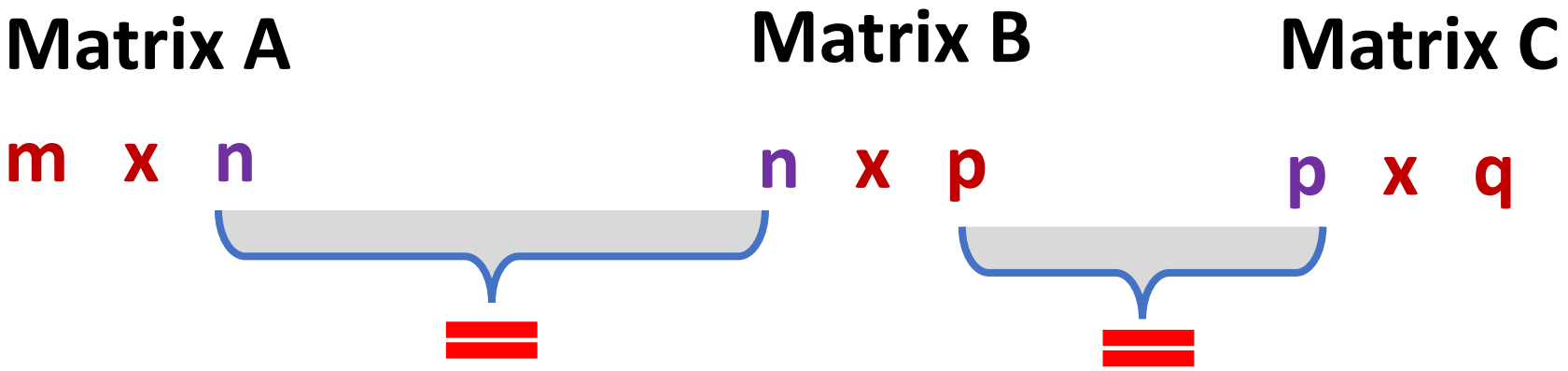
Matrix Chain Multiplication

Basics:



Matrix Chain Multiplication

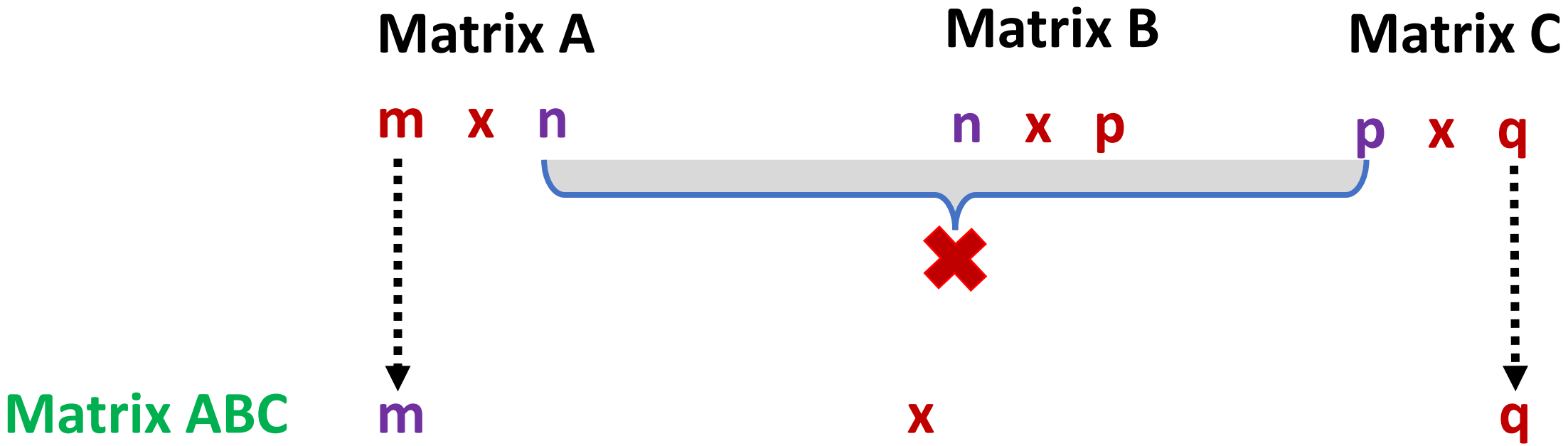
Basics:



Matrix ABC

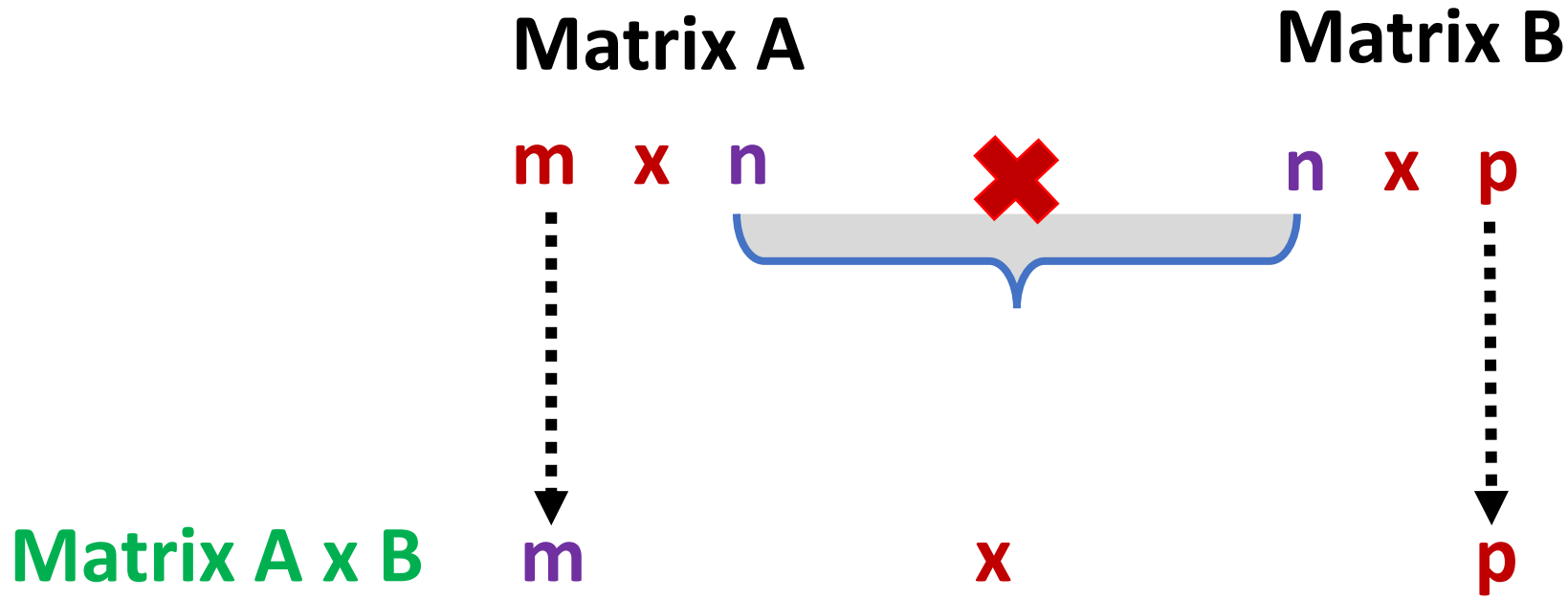
Matrix Chain Multiplication

Basics:



Matrix Chain Multiplication

Basics:



Number of multiplication operation: ***mnp***

Matrix Chain Multiplication

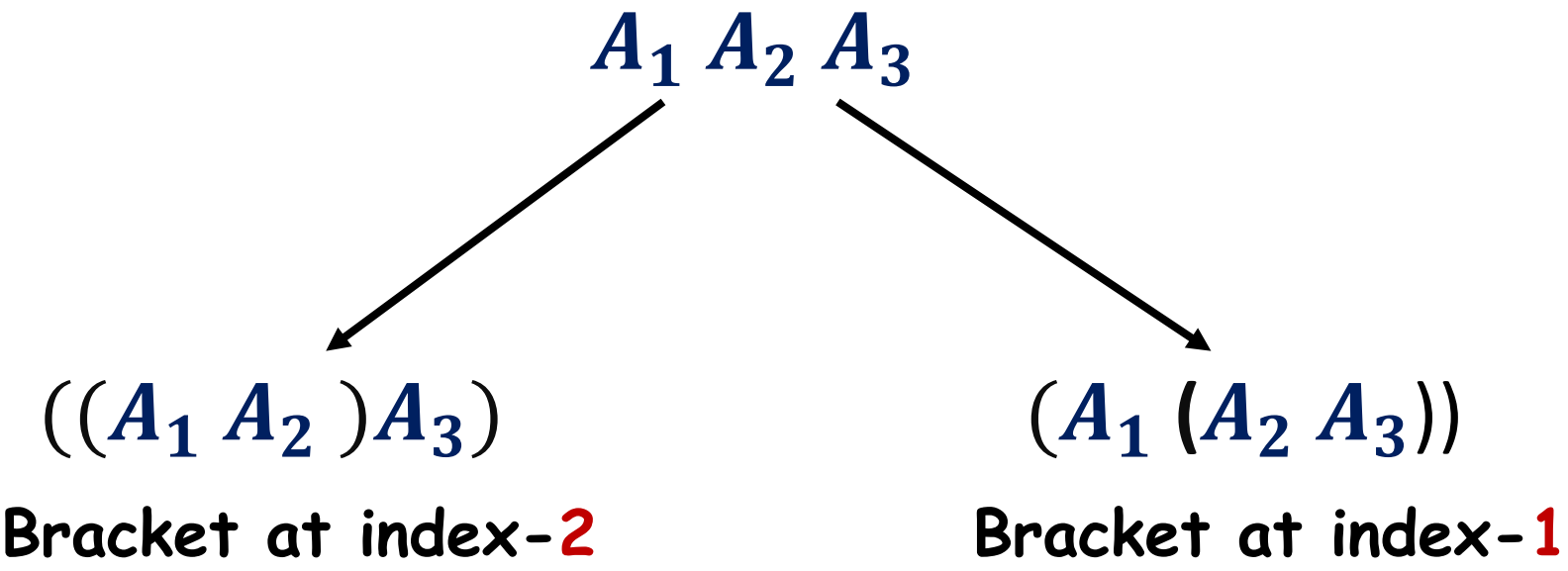
Let us consider 3 matrices

$A_1 \equiv 3 \times 2$
$A_2 \equiv 2 \times 5$
$A_3 \equiv 5 \times 4$

Goal: We want to compute $A_1\ A_2\ A_3$ using **minimum** number of multiplication

Matrix Chain Multiplication

$A_1 \equiv 3 \times 2$ $A_2 \equiv 2 \times 5$ $A_3 \equiv 5 \times 4$



Bracket at index-2

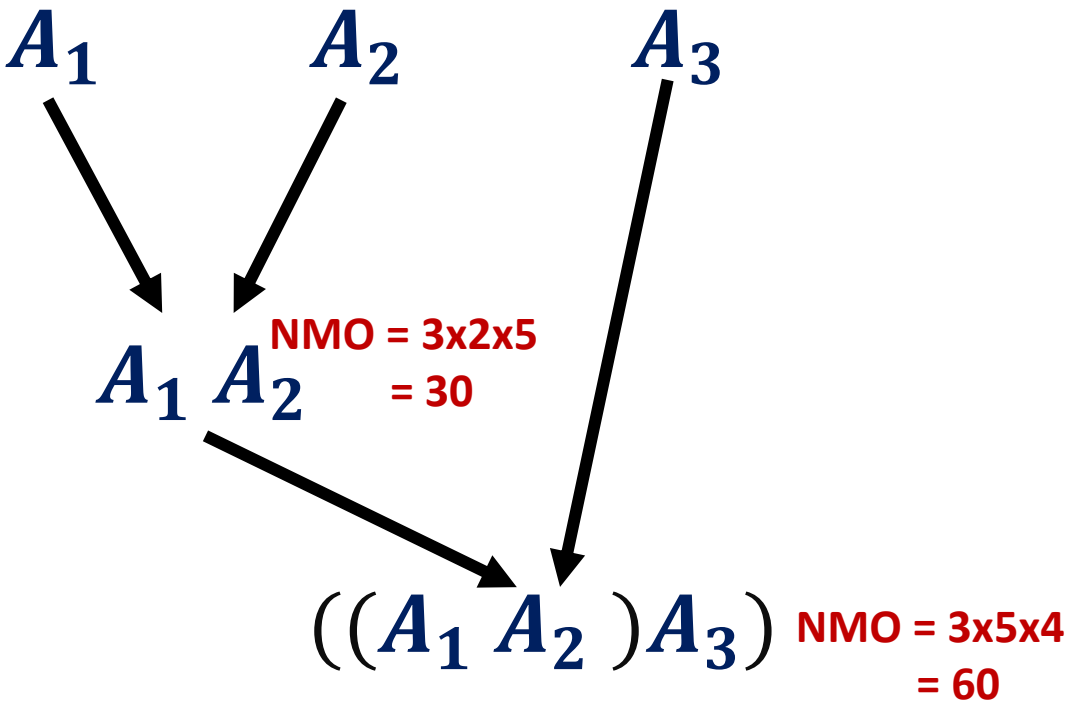
Bracket at index-1

Number of ways to multiply the matrices = Number of ways to put brackets

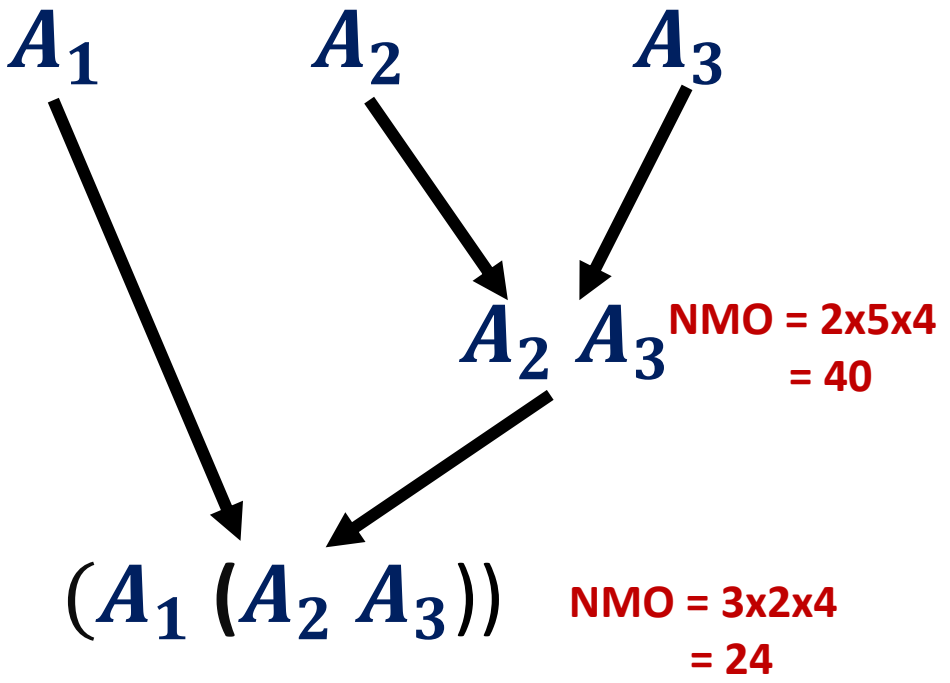
NOTE: No bracket at index 3(last index)

Matrix Chain Multiplication

$A_1 \equiv 3 \times 2$ $A_2 \equiv 2 \times 5$ $A_3 \equiv 5 \times 4$ $A_1A_2 \equiv 3 \times 5$ $A_2A_3 \equiv 2 \times 4$



Total No. of Multiplication operation = 30 + 60
= 90



Total No. of Multiplication operation = 40 + 24
= 64

Matrix Chain Multiplication

In general

Given:

A set of n matrices A_1, A_2, A_3,……..,A_n with their size array [p_0,p_1,p_2,p_3,…….,p_n]

Goal: We want to compute $A_1\ A_2\ A_3$……. A_n using **minimum** number of multiplication operation

$$A_1 \equiv p_0 \times p_1$$
$$A_2 \equiv p_1 \times p_2$$
$$A_3 \equiv p_2 \times p_3$$
$$\vdots$$
$$A_i \equiv p_{i-1} \times p_i$$
$$\vdots$$
$$A_n \equiv p_{n-1} \times p_n$$

Matrix Chain Multiplication

Recursive Formulation:

m[i, j] = minimum number of multiplication operation to compute $A_iA_{i+1}\ldots\ldots\ldots\ldots A_j$

m[1, n] = minimum number of multiplication operation to compute $A_1A_2\ldots\ldots\ldots\ldots A_n$ (*SOLUTION*)

m[2, 4] = minimum number of multiplication operation to compute $A_2A_3A_4$

Matrix Chain Multiplication

Recursive Formulation:

Base Case:

If **i=j** then we have **single** matrix

$$m[i, j] = 0$$

$$m[1, 1] = m[2, 2] = m[3, 3] = 0$$

Matrix Chain Multiplication

Recursive Case:

Let's put the bracket at k^{th} index

$$i \leq k < j$$

$$A_i \equiv p_{i-1} \times p_i$$
$$A_k \equiv p_{k-1} \times p_k$$
$$A_{k+1} \equiv p_k \times p_{k+1}$$
$$A_j \equiv p_{j-1} \times p_j$$

m[i, j]

$$(A_i A_{i+1} \ldots\ldots\ldots\ldots\ldots A_k)(A_{k+1} A_{k+2} \ldots\ldots\ldots\ldots\ldots A_j)$$

m[i, k]

Size= $p_{i-1} \times p_k$

m[k+1, j]

Size= $p_k \times p_j$

NMO= $p_{i-1} \times p_k \times p_j$

Matrix Chain Multiplication

Recursive Formulation:

Recursive Case:

$$m[i, j] = \min_{i \le k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

Matrix Chain Multiplication

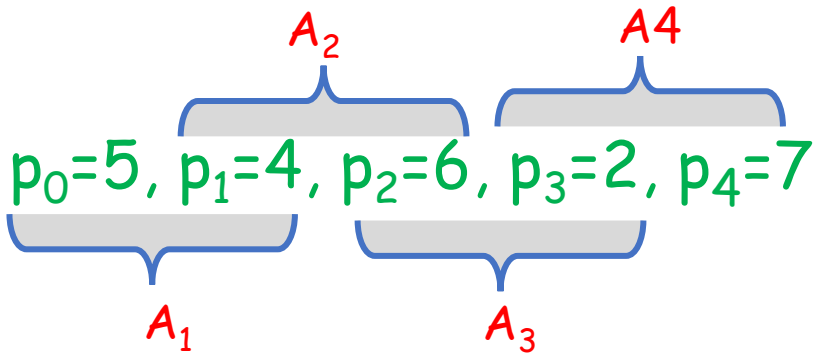
Recursive Formulation:

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \end{cases}$$

Matrix Chain Multiplication

Matrices: $A_1, A_2\ A_3, A_4$

Dimension of Matrices: $p_0=5, p_1=4, p_2=6, p_3=2, p_4=7$



Matrix Chain Multiplication

m

	1	2	3	4
1				
2				
3				
4				

s

	1	2	3	4
1				
2				
3				
4				

$p_0=5$
$p_1=4$
$p_2=6$
$p_3=2$
$p_4=7$

Matrix Chain Multiplication

m

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

s

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

$p_0=5$
$p_1=4$
$p_2=6$
$p_3=2$
$p_4=7$

Diagonal Entries are 0's due to base case (i=j)

Matrix Chain Multiplication

m

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

s

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

$p_0=5$
$p_1=4$
$p_2=6$
$p_3=2$
$p_4=7$

$m[1,2] = \min(m[1,1] + m[2,2] + P_0.P_1.P_2)$ **k=1**

$= \min(0 + 0 + 5.4.6 = 120)$

$= \min(120)$

$= 120$

Matrix Chain Multiplication

m

	1	2	3	4
1	0	120		
2		0	?	
3			0	
4				0

s

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

$p_0=5$
$p_1=4$
$p_2=6$
$p_3=2$
$p_4=7$

$m[2,3] = \min(m[2,2] + m[3,3] + P_1.P_2.P_3)$ **k=2**

$= \min(0 + 0 + 4.6.2 = 48)$

$= \min(48)$

$= 48$

Matrix Chain Multiplication

m

	1	2	3	4
1	0	120		
2		0	48	
3			0	?
4				0

s

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

$p_0=5$
$p_1=4$
$p_2=6$
$p_3=2$
$p_4=7$

$m[3,4] = \min(m[3,3] + m[4,4] + P_2.P_3.P_4)$ **k=1**

$= \min(0 + 0 + 6.2.7 = 84)$

$= \min(84)$

$= 84$

Matrix Chain Multiplication

m

	1	2	3	4
1	0	120	?	
2		0	48	
3			0	84
4				0

s

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

$p_0=5$
$p_1=4$
$p_2=6$
$p_3=2$
$p_4=7$

$$m[1,3] = min \begin{cases} m[1,1] + m[2,3] + p_0 p_1 p_3 & k = 1 \\ m[1,2] + m[3,3] + p_0 p_2 p_3 & k = 2 \end{cases}$$

k=1

$$m[1,3] = min \begin{cases} 0 + 48 + 5.4.2 & = 48 + 40 = 88 \\ 120 + 0 + 5.6.2 & = 120 + 60 = 180 \end{cases}$$

= 88

Matrix Chain Multiplication

m

	1	2	3	4
1	0	120	88	
2		0	48	?
3			0	84
4				0

s

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

$p_0=5$
$p_1=4$
$p_2=6$
$p_3=2$
$p_4=7$

$$m[2,4] = min \begin{cases} m[2,2] + m[3,4] + p1p2p4 & k = 2 \\ m[2,3] + m[4,4] + p1p3p4 & k = 3 \end{cases}$$

k=3

$$m[2,4] = min \begin{cases} 0 + 84 + 4.6.7 & = 84 + 168 = 252 \\ 48 + 0 + 4.2.7 & = 48 + 56 = 104 \end{cases}$$

= 104

Matrix Chain Multiplication

m

	1	2	3	4
1	0	120	88	?
2		0	48	104
3			0	84
4				0

s

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

$p_0=5$
$p_1=4$
$p_2=6$
$p_3=2$
$p_4=7$

$$m[1,4] = min \begin{cases} m[1,1] + m[2,4] + p0p1p4 & k = 1 \\ m[1,2] + m[3,4] + p0p2p4 & k = 2 \\ m[1,3] + m[4,4] + p0p3p4 & k = 3 \end{cases}$$

$$m[1,4] = min \begin{cases} 0 + 104 + 5.4.7 & = 104 + 140 = 244 \\ 120 + 84 + 5.6.7 & = 120 + 84 + 210 = 414 \\ 88 + 0 + 5.2.7 & = 88 + 70 = 158 \end{cases}$$

k=3

= 158

Matrix Chain Multiplication

m

	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84
4				0

s

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

Minimum number of multiplication operation to compute $A_1A_2A_3A_4$ = 158

Matrix Chain Multiplication

Time Complexity = O(*Size of matrix* x *Cost to fill one slot*)

= O(n.n.O(n))

= $\mathbf{O(n^3)}$

Space Complexity = O(*Size of matrix*)

= O(n.n)

= $\mathbf{O(n^2)}$

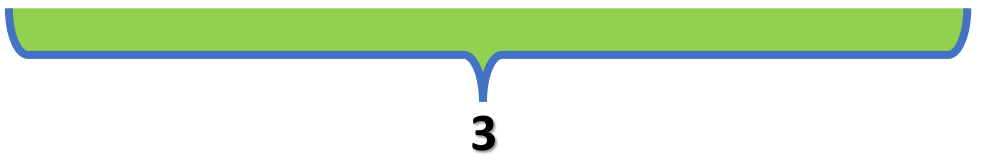
Matrix Chain Multiplication

```
MATRIX-CHAIN-ORDER(p)
 1  n ← length[p] − 1
 2  for i ← 1 to n
 3       do m[i, i] ← 0
 4  for l ← 2 to n                ▷ l is the chain length.
 5       do for i ← 1 to n − l + 1
 6            do j ← i + l − 1
 7               m[i, j] ← ∞
 8               for k ← i to j − 1
 9                    do q ← m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
10                       if q < m[i, j]
11                          then m[i, j] ← q
12                               s[i, j] ← k
13  return m and s
```

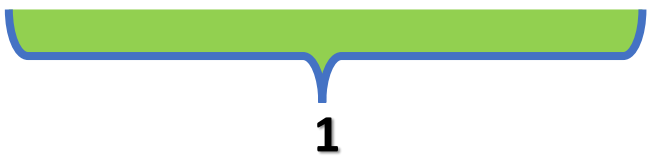
Matrix Chain Multiplication

How to get order of optimal multiplication table ?

$A_1 \quad A_2 \quad A_3 \quad A_4$



$(A_1 \quad A_2 \quad A_3) \quad A_4$



$(A_1 \quad (A_2 \quad A_3)) \quad A_4$

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

$The\ final\ order = ((A_1\ (A_2\ A_3))\ A_4)$

Matrix Chain Multiplication

How to get order of optimal multiplication table ?

```
function constructOrder(s, i, j):
1.      if i == j:
2.                  return "A" + i
3.      k = s[i][j]
4.      leftOrder = constructOrder(s, i, k)
5.      rightOrder = constructOrder(s, k + 1, j)
6.      return "(" + leftOrder + " x " + rightOrder + ")"
```

Longest Common Subsequence

Basics:

Subsequence: A subsequence Y of a sequence X can be obtained by dropping 0 or more elements from sequence X

$X = < A, B , B, A, C >$

$X = < A, B , B, A, C >$ $\qquad$ $Y = < A, B , A, C >$

Y is a subsequence of sequence X

Longest Common Subsequence

Basics:

X = < A, B , B, A, C> Y = < B , A, C>

X = < A, B , B, A, C> Y = < B , A>

X = < A, B , B, A, C> Y = < >

Longest Common Subsequence

Basics:

How many subsequences of a sequence having n symbols are possible?

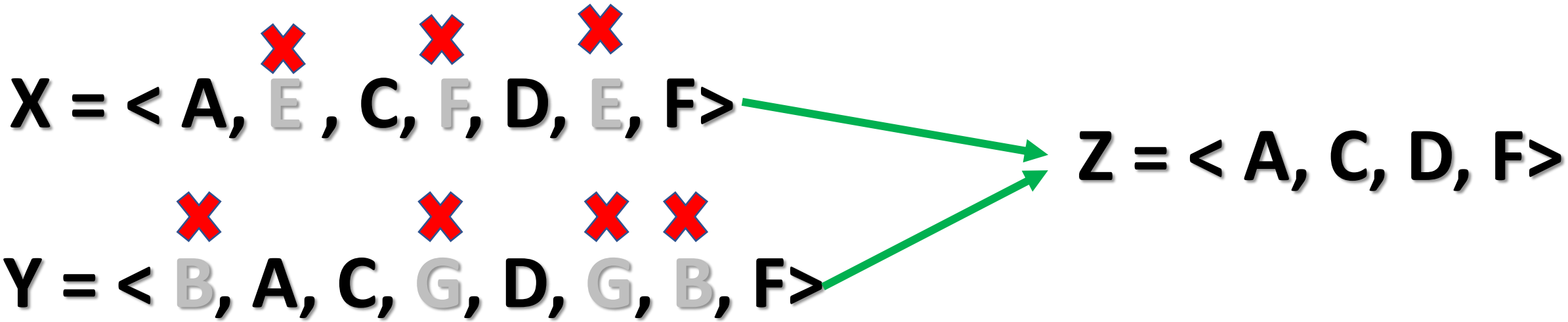
Ans: 2^n

Longest Common Subsequence

Basics:

Common subsequence: A sequence Z is a common subsequence of sequences X and Y if Z is a subsequence of both X and Y

X = < A, E , C, F, D, E, F> Y = < B, A, C, G, D, G, B, F>



Longest Common Subsequence

Basics:

Longest Common subsequence(LCS): A common subsequence Z of sequence X and Y whose length is longest

X = < A, E , C, F, D, E, F> Y = < B, A, C, G, D, G, B, F>

X = < A, E , C, F, D, E, F>

Y = < B, A, C, G, D, G, B, F>

X = < A, C, D, F>

LCS

Longest Common Subsequence

Problem Statement:

Given:

Two sequences X and Y are given to us

$X = \langle x_1, x_2, x_3, \ldots\ldots\ldots\ldots\ldots, x_m \rangle$

$Y = \langle y_1, y_2, y_3, \ldots\ldots\ldots\ldots\ldots, y_n \rangle$

Goal:

Find the **LCS** of X and Y

Longest Common Subsequence

Recursive Formulation:

LCS(i, j) = Length of LCS of sequences

$\langle x_1, x_2, x_3, \ldots\ldots\ldots\ldots, x_i \rangle$

$\langle y_1, y_2, y_3, \ldots\ldots\ldots\ldots, y_j \rangle$

Goal:

Find the LCS of X and Y

Longest Common Subsequence

Base Case:

$LCS(0, j) = 0$

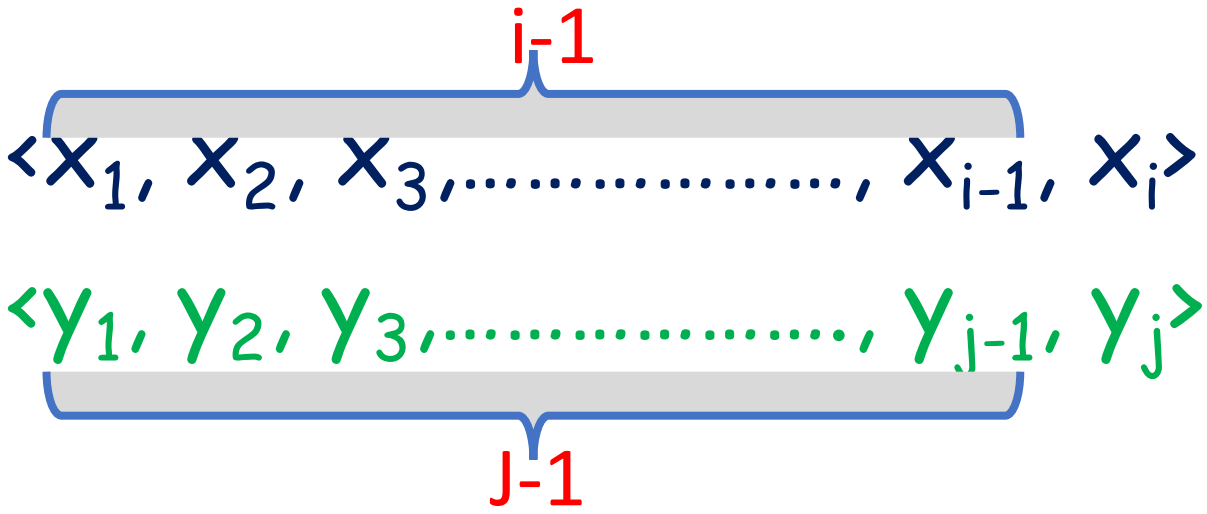
$LCS(i, 0) = 0$

$LCS(i, j) = 0$ if $i=0$ or $j=0$

Longest Common Subsequence

Recursive Case:

Case-1: $x_i = y_j$

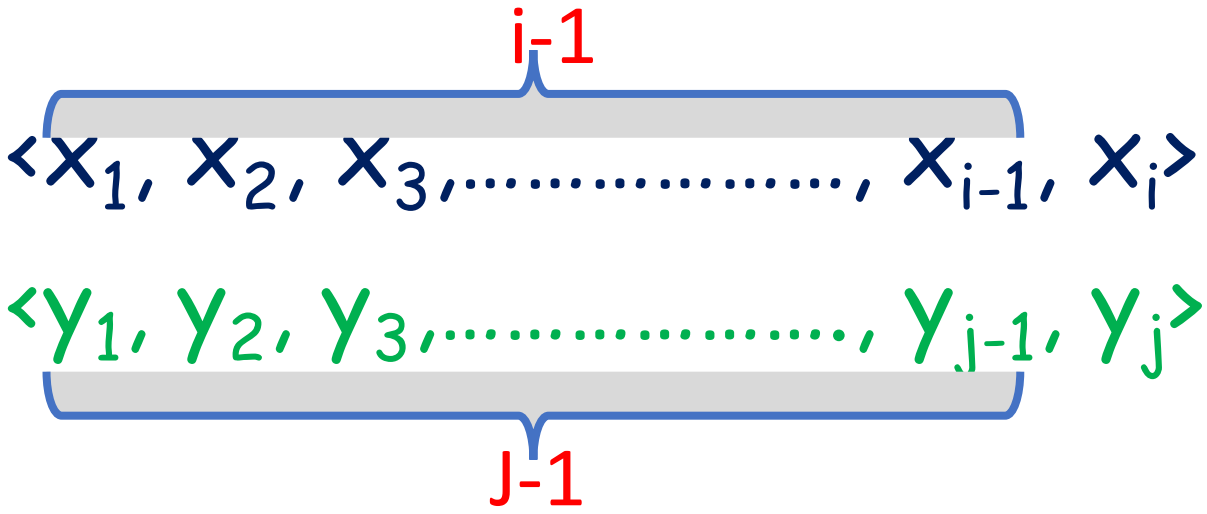


$$LCS(i, j) = LCS(i-1, j-1) + 1$$

Longest Common Subsequence

Recursive Case:

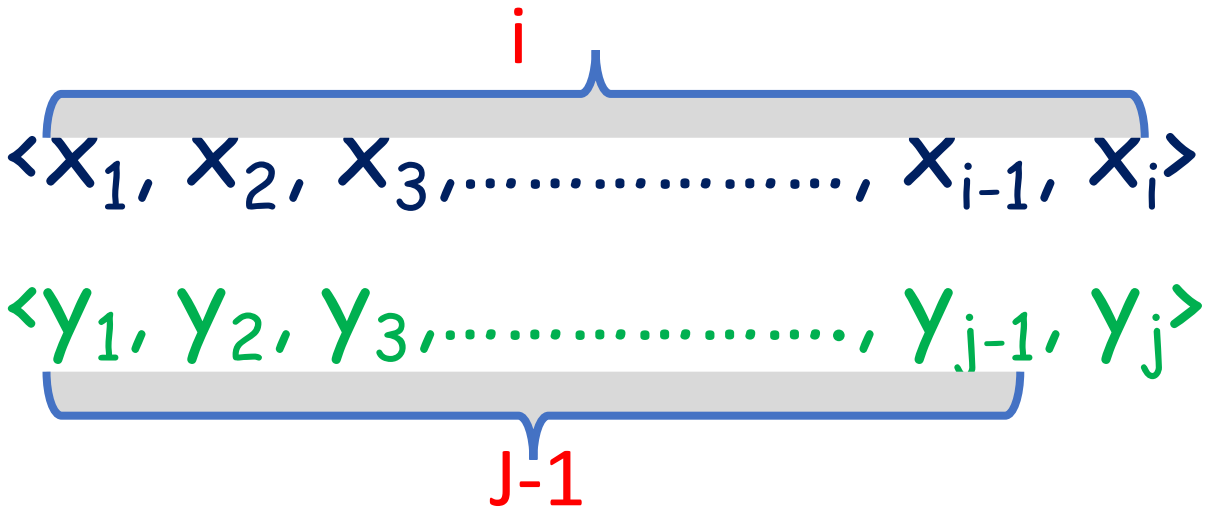
Case-2: $x_i \neq y_j$



Longest Common Subsequence

Recursive Case:

Case-2.1:

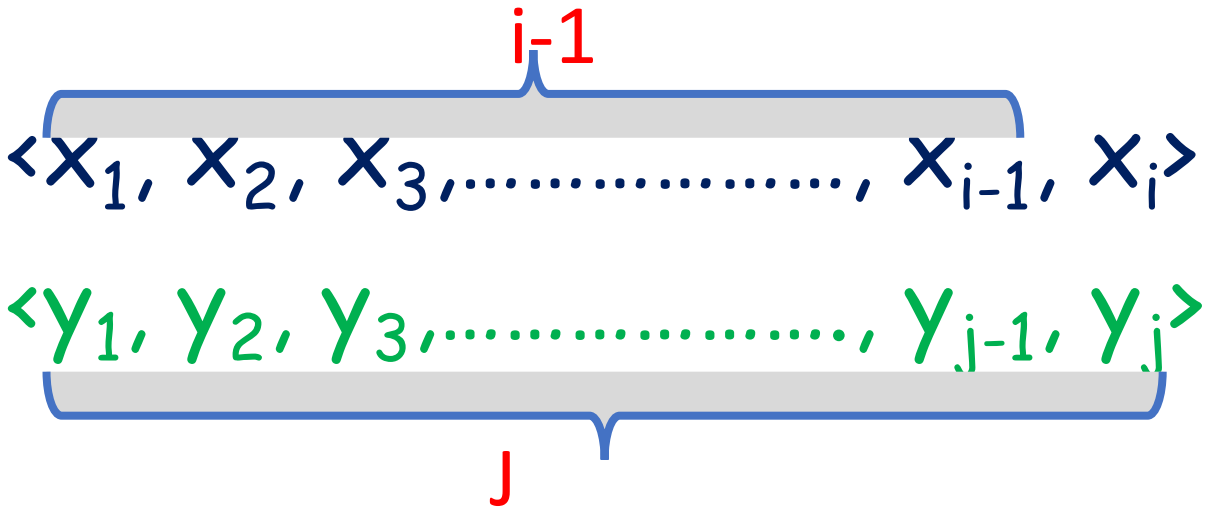


$$LCS(i, j) = LCS(i, j-1)$$

Longest Common Subsequence

Recursive Case:

Case-2.2:



$$LCS(i, j) = LCS(i-1, j)$$

Longest Common Subsequence

Recursive Case:

Case-2: $x_i \neq y_j$

$\langle x_1, x_2, x_3, \ldots\ldots\ldots\ldots\ldots, x_{i-1}, x_i \rangle$

$\langle y_1, y_2, y_3, \ldots\ldots\ldots\ldots\ldots, y_{j-1}, y_j \rangle$

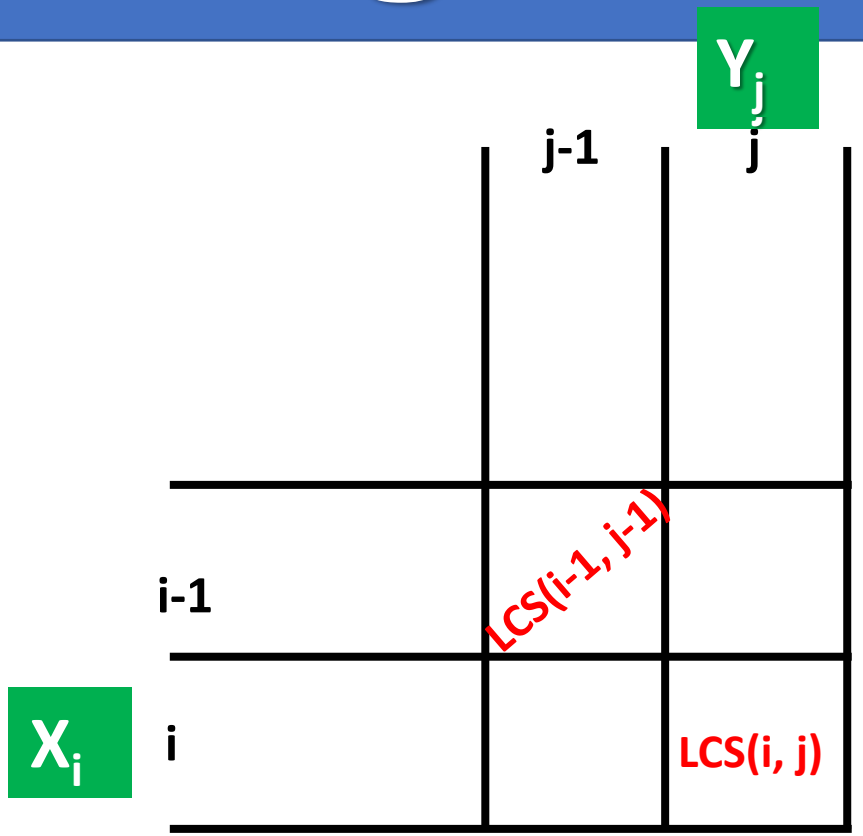
$$LCS(i,j) = max \begin{cases} LCS(i, j-1) \\ LCS(i-1, j) \end{cases}$$

Longest Common Subsequence

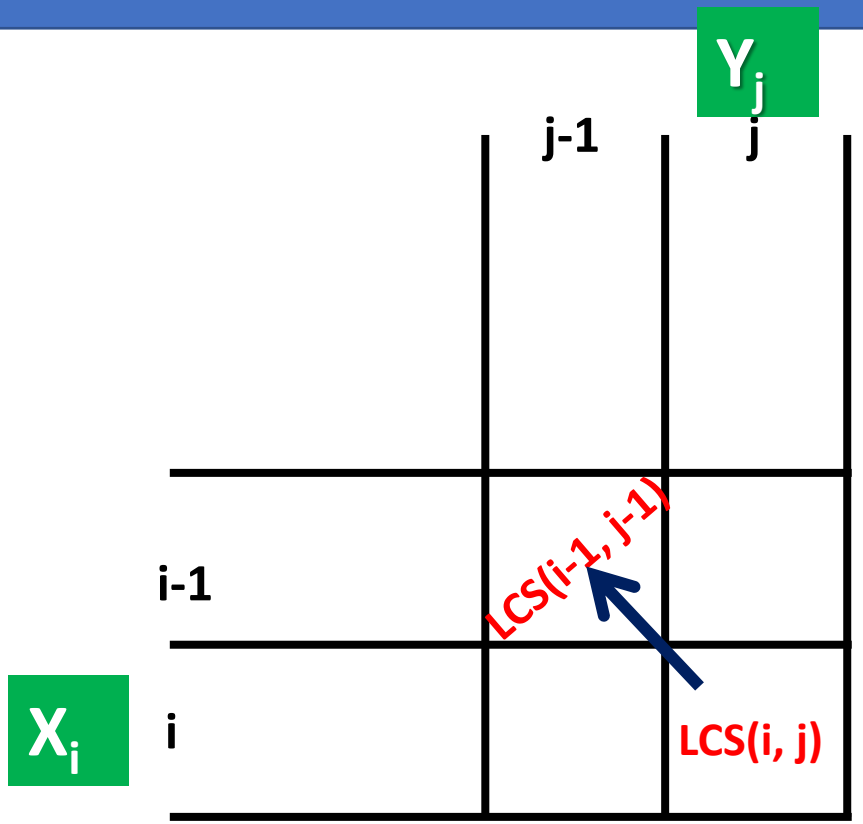
Recursive Formulation:

$$LCS(i,j) = \begin{cases} 0 & if\ i = 0\ or\ j = 0 \\ LCS(i-1, j-1) + 1 & if\ x_i = y_j \\ max \begin{cases} LCS(i, j-1) & if\ x_i \neq y_j \\ LCS(i-1, j) \end{cases} \end{cases}$$

Longest Common Subsequence



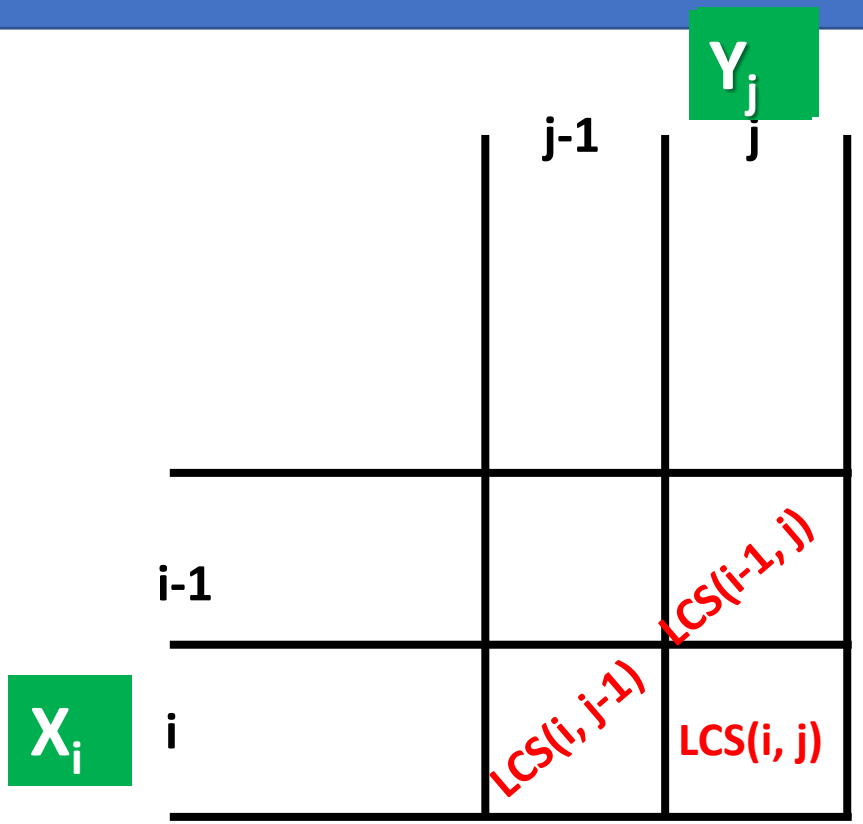
Longest Common Subsequence



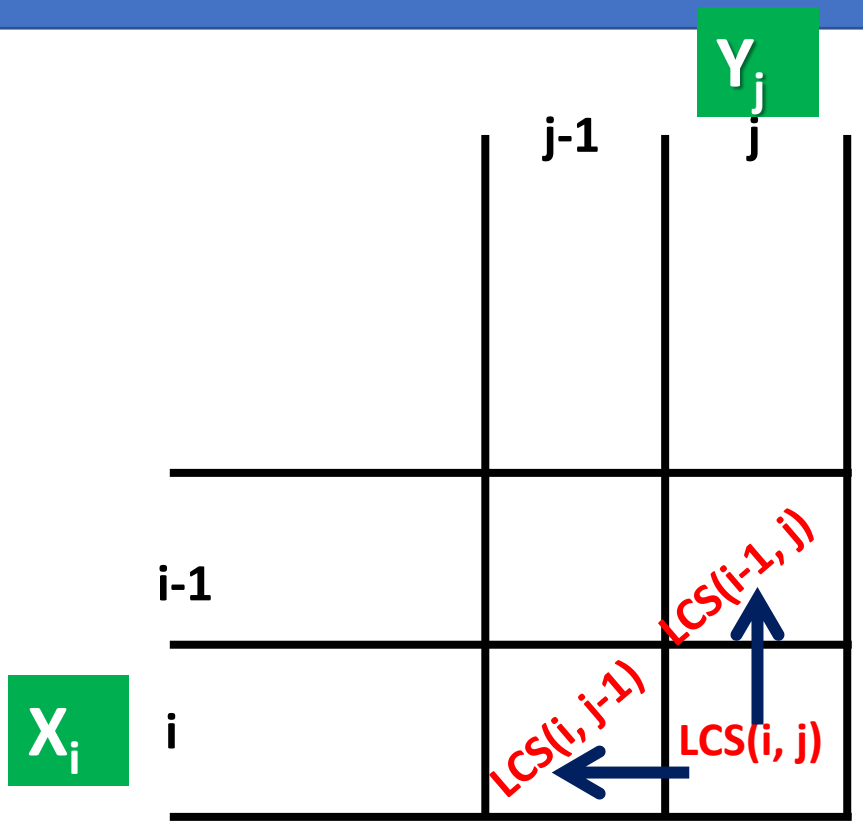
X_i = Y_j

$$LCS(i, j) = LCS(i-1, j-1) + 1$$

Longest Common Subsequence



Longest Common Subsequence



X_i != Y_j

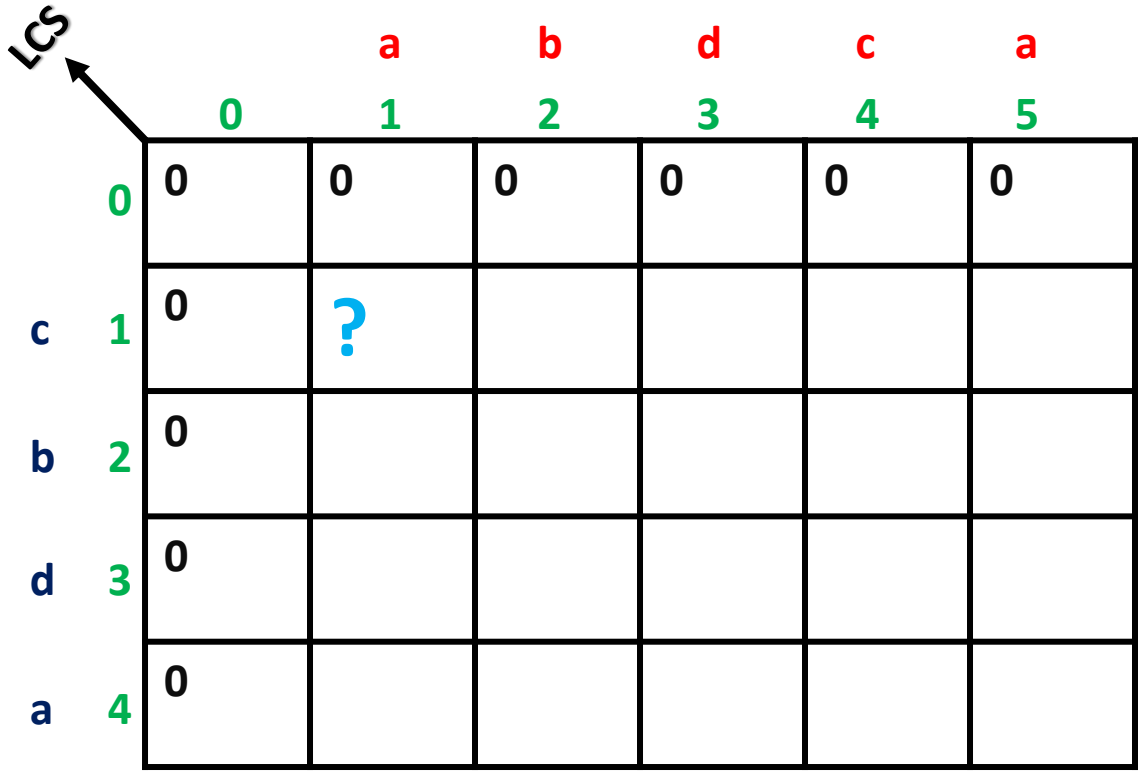
$$LCS(i,j) = max \begin{cases} LCS(i, j-1) \\ LCS(i-1, j) \end{cases}$$

Longest Common Subsequence

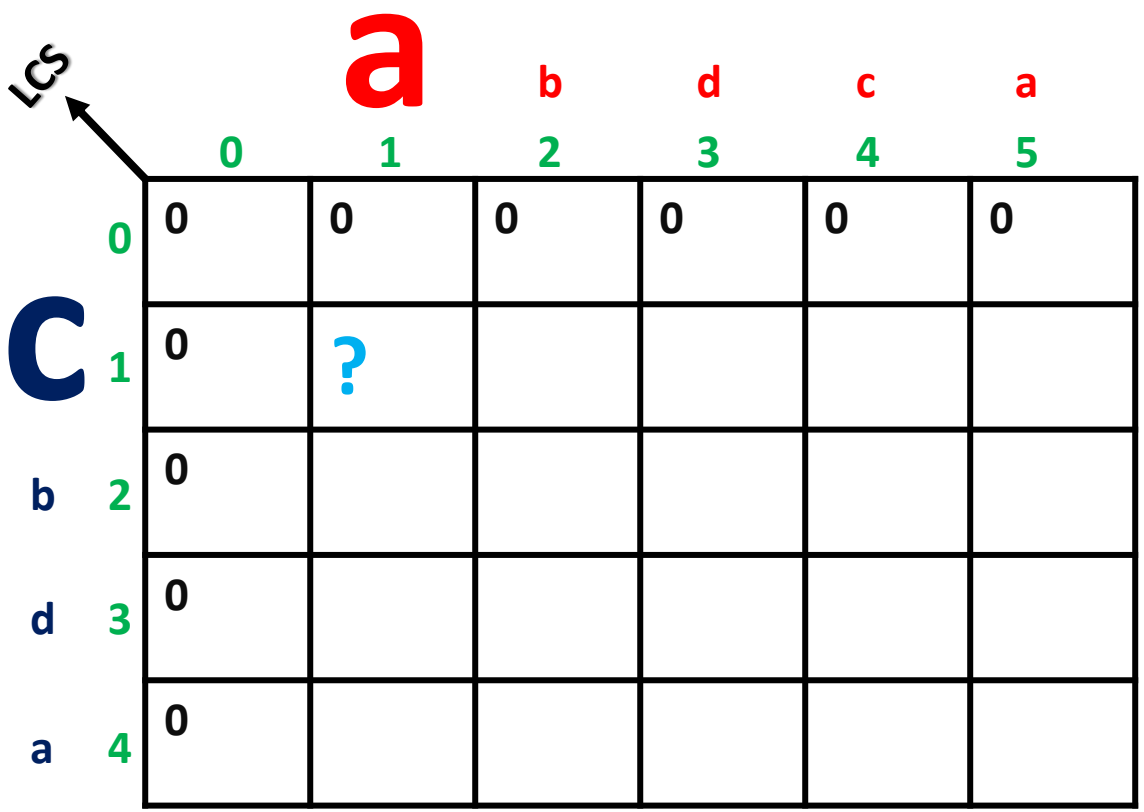
Let us consider the following 2 strings:

X = cbda

Y = abdca



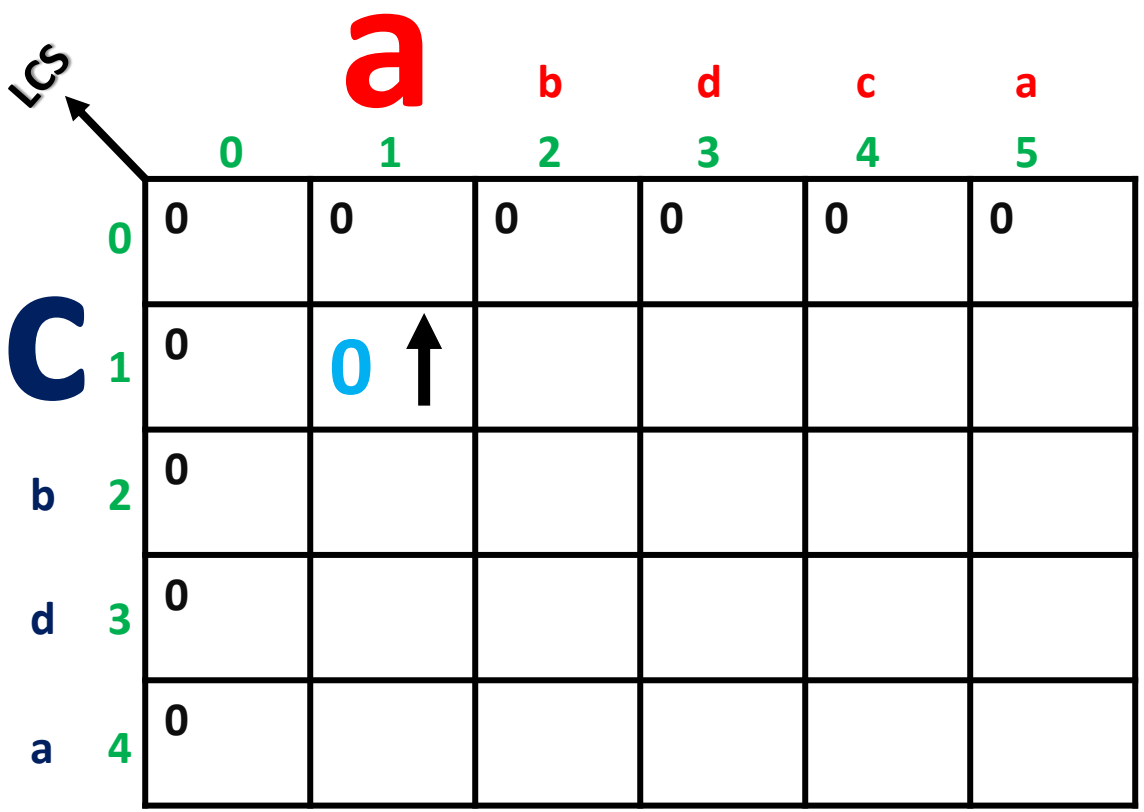
Longest Common Subsequence



$$c \neq a$$

$$LCS(1,1) = max \begin{cases} LCS(1,0) = 0 \\ LCS(0,1) = 0 \end{cases}$$
$$= 0$$

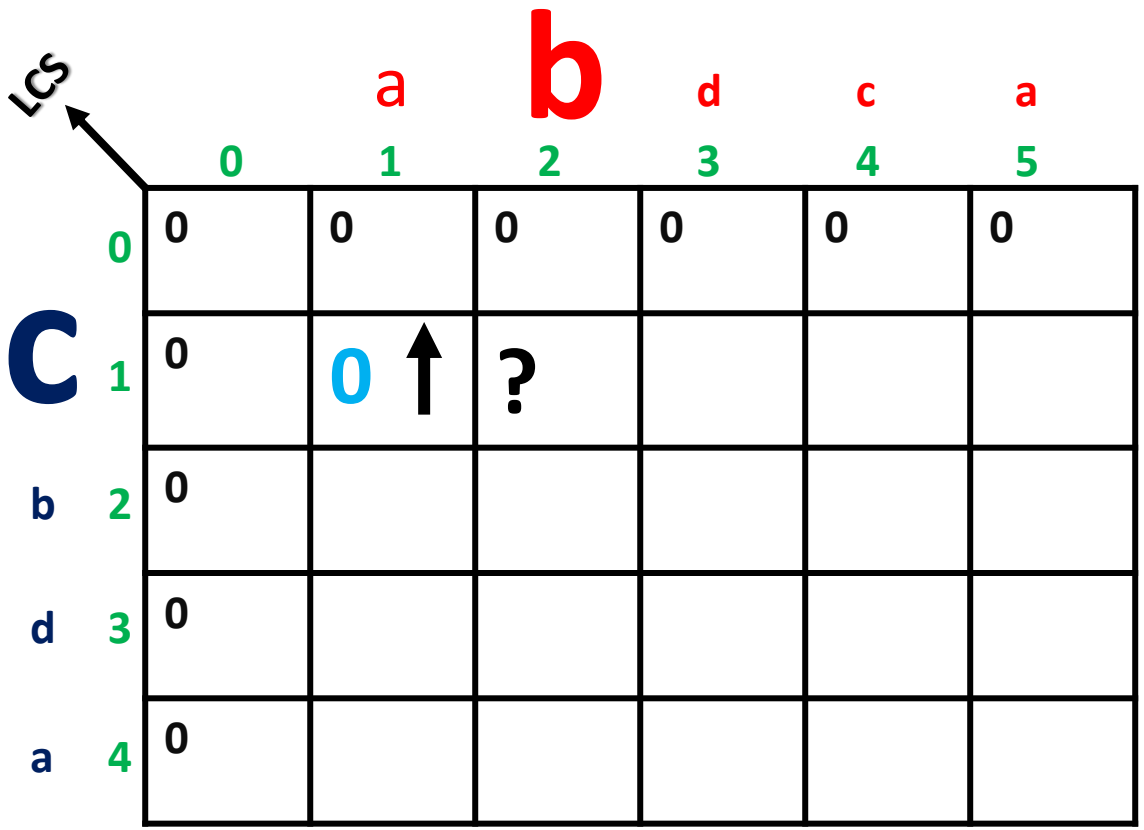
Longest Common Subsequence



$$c \neq a$$

$$LCS(1,1) = max \begin{cases} LCS(1,0) = 0 \\ LCS(0,1) = 0 \end{cases}$$
$$= 0$$

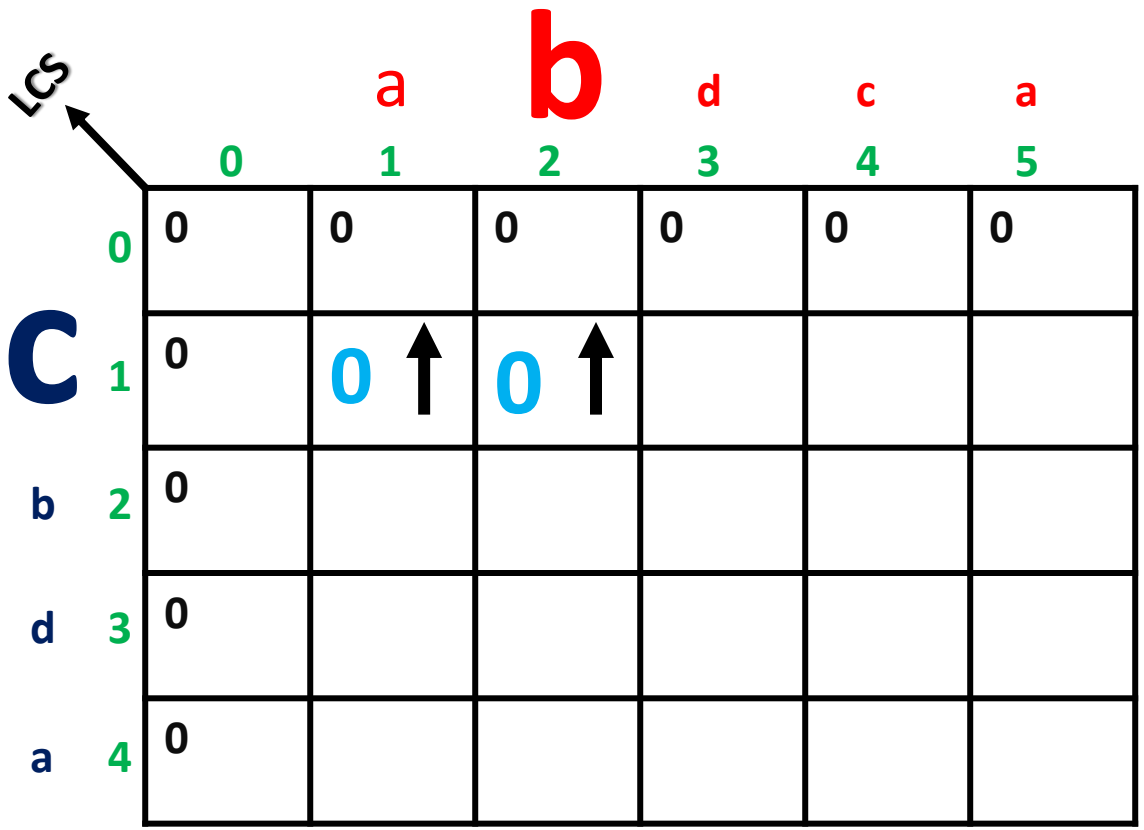
Longest Common Subsequence



$$c \neq b$$

$$LCS(1,2) = max \begin{cases} LCS(1,1) = 0 \\ LCS(0,2) = 0 \end{cases} = 0$$

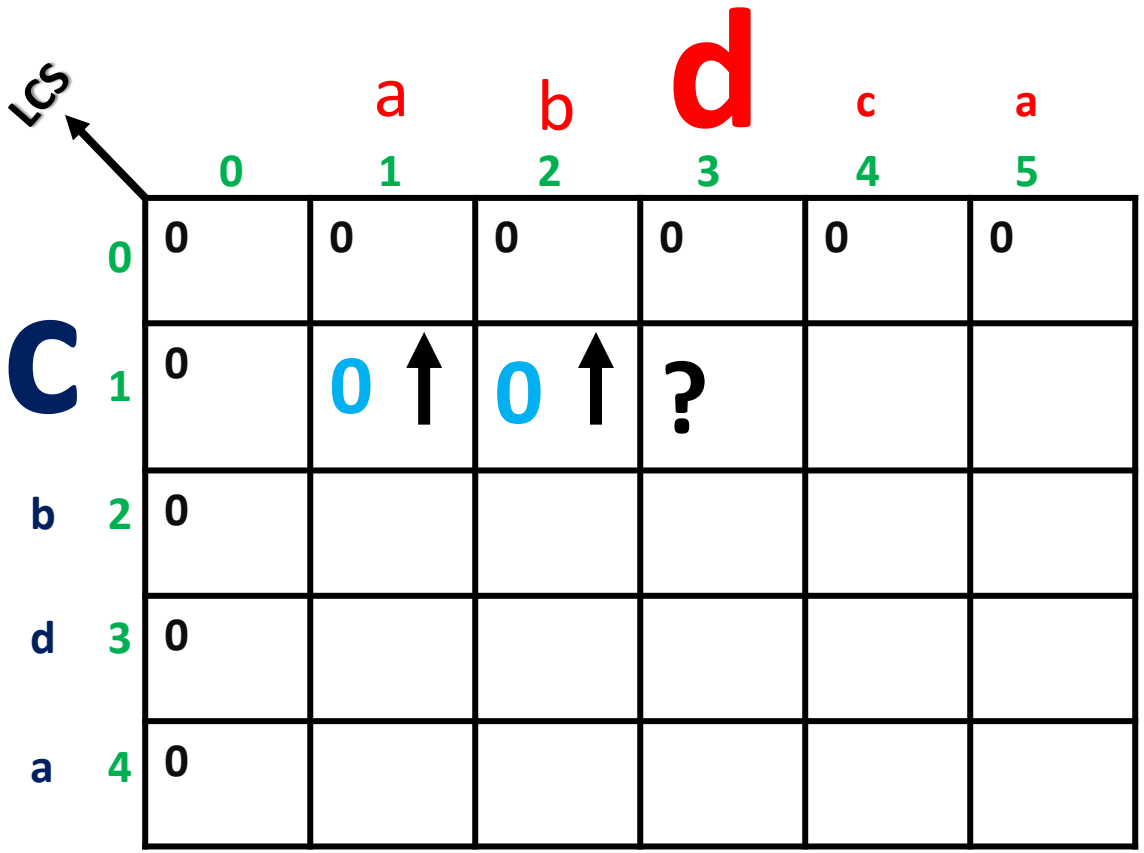
Longest Common Subsequence



$$c \neq b$$

$$LCS(1,2) = max \begin{cases} LCS(1,1) = 0 \\ LCS(0,2) = 0 \end{cases} = 0$$

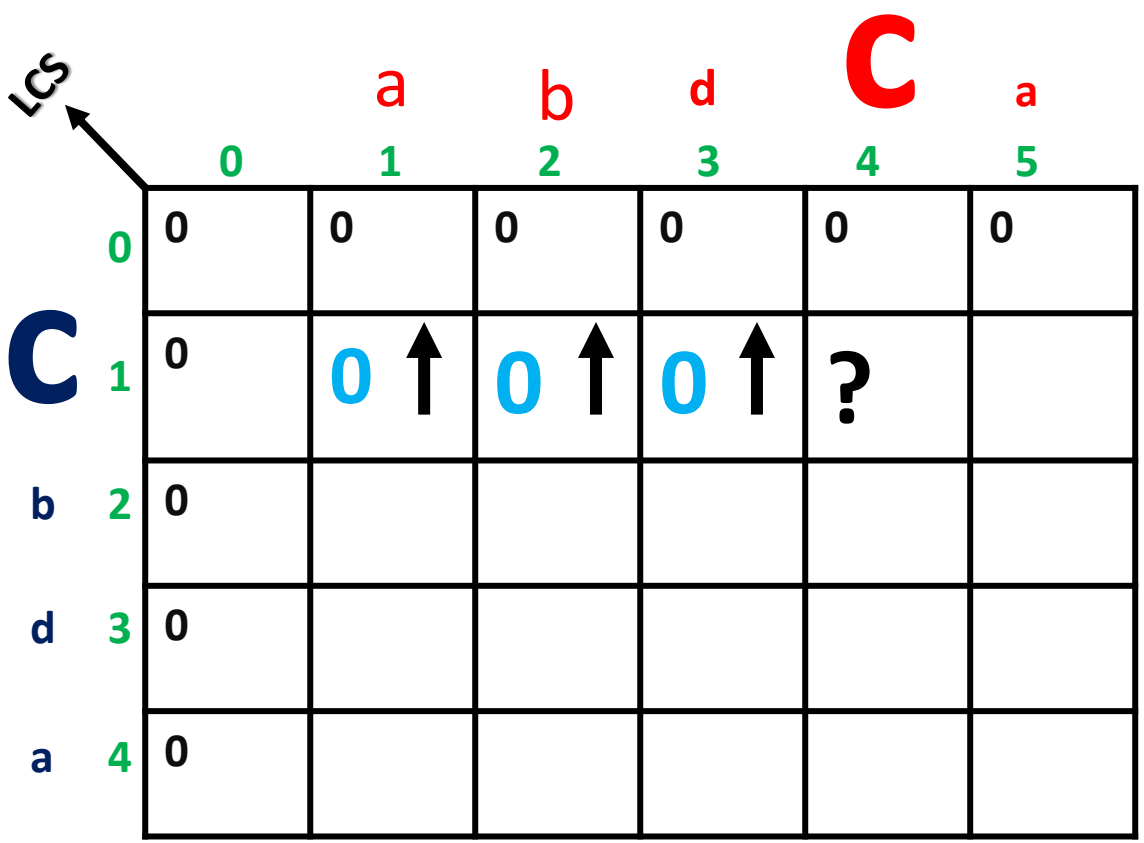
Longest Common Subsequence



$$c \neq d$$

$$LCS(1,3) = max \begin{cases} LCS(1,2) = 0 \\ LCS(0,3) = 0 \end{cases}$$
$$= 0$$

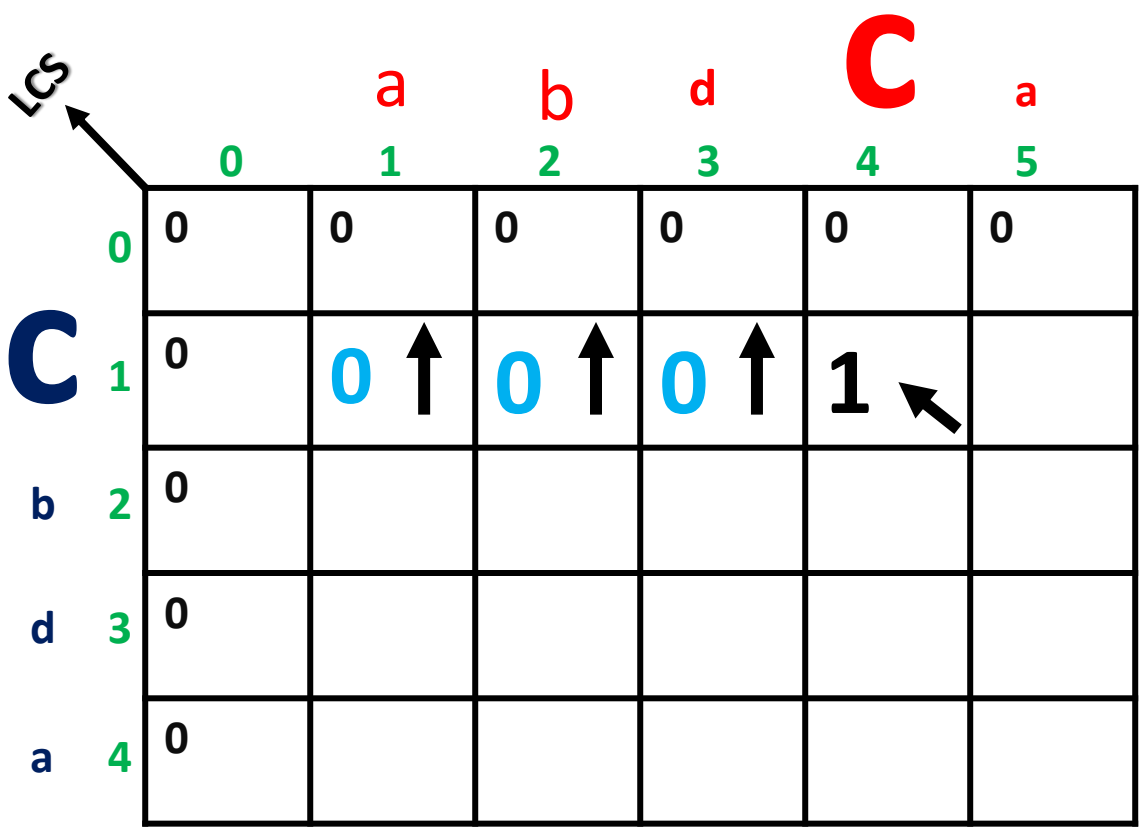
Longest Common Subsequence



$$C = C$$

$$LCS(1,4) = LCS(0,3) + 1$$
$$= 0 + 1 = 1$$

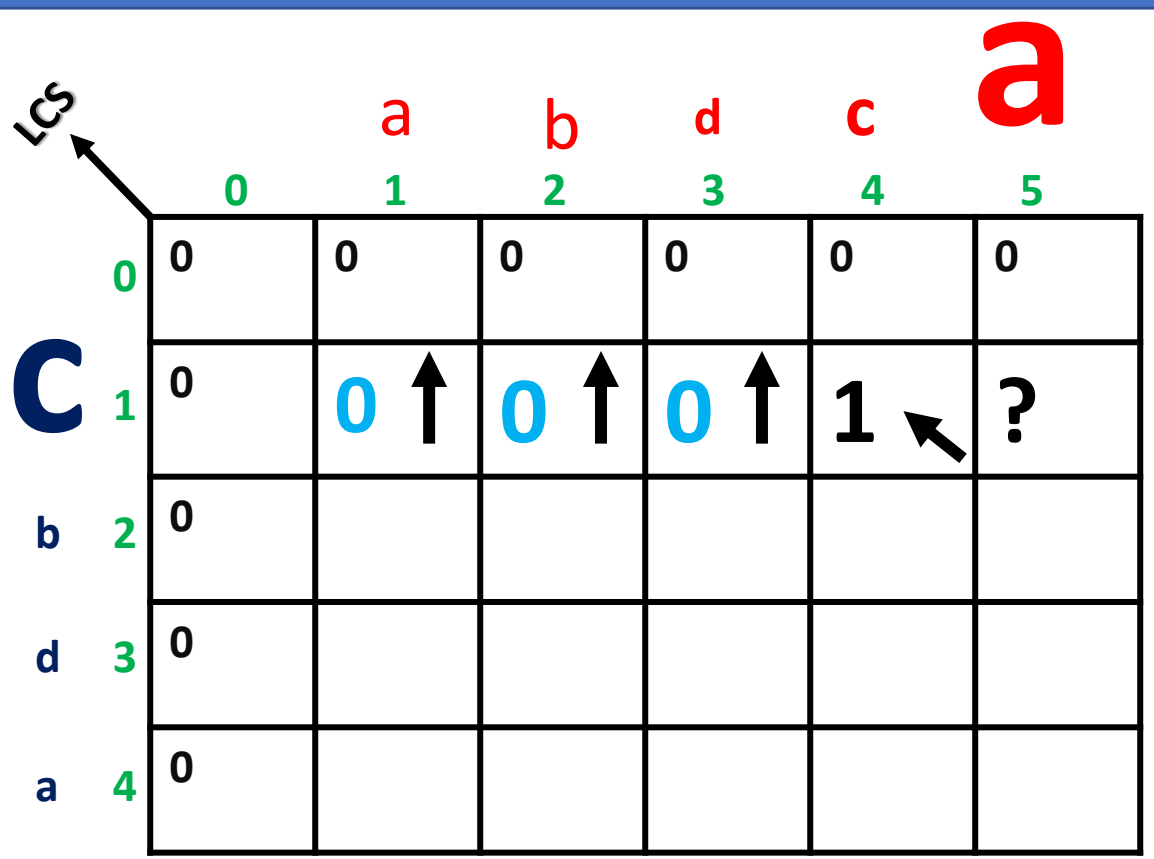
Longest Common Subsequence



$$c = c$$

$$LCS(1,4) = LCS(0,3) + 1$$
$$= 0 + 1 = 1$$

Longest Common Subsequence

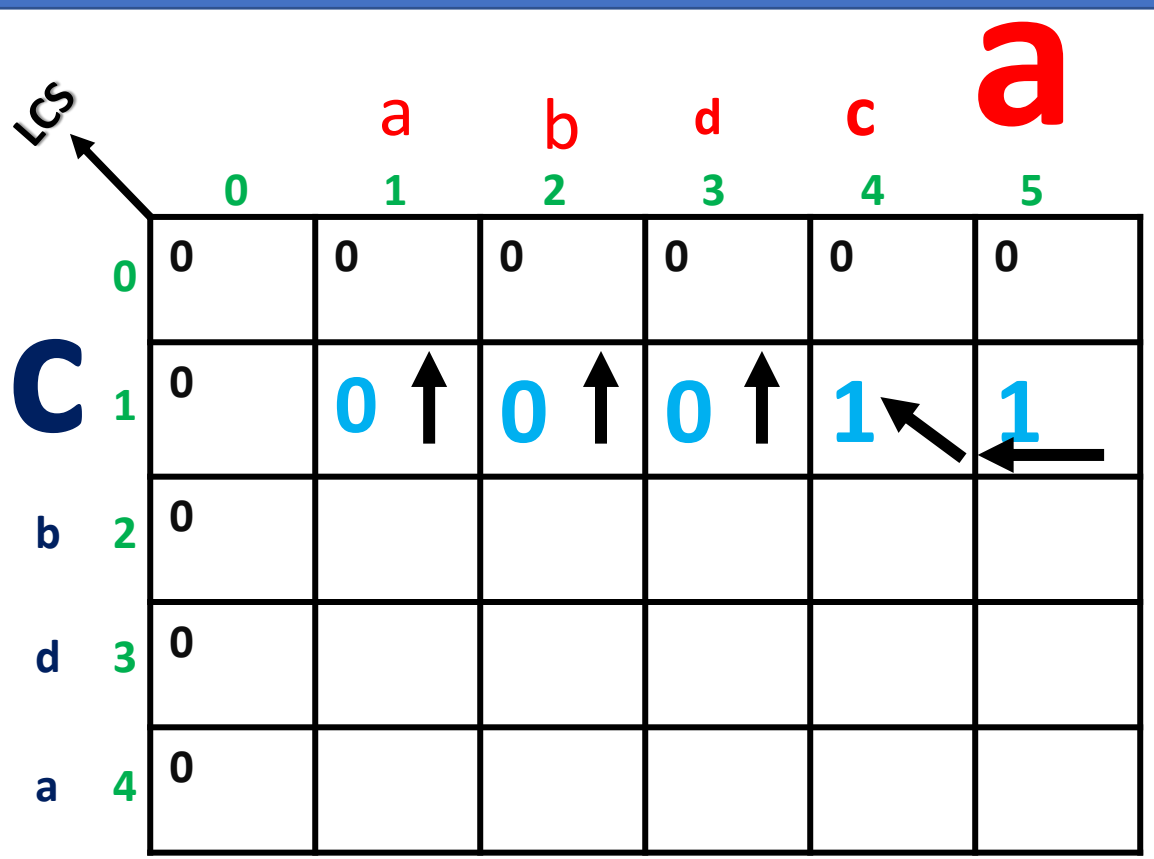


$$c \neq a$$

$$LCS(1,5) = max \begin{cases} LCS(1,4) = 1 \\ LCS(0,5) = 0 \end{cases}$$

$$= 1$$

Longest Common Subsequence

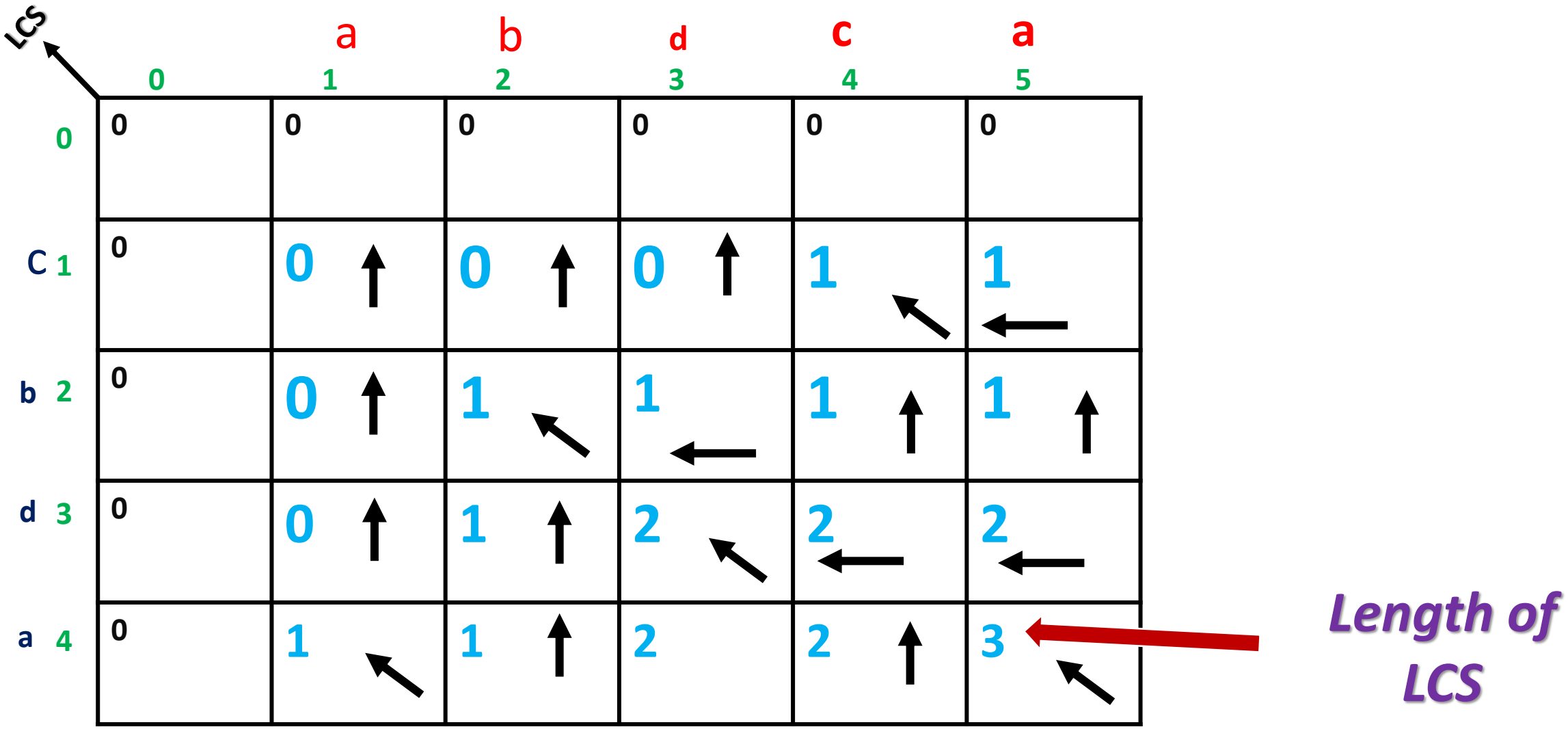


$$c \neq a$$

$$LCS(1,5) = max\begin{cases} LCS(1,4) = 1 \\ LCS(0,5) = 0 \end{cases}$$

$$= 1$$

Longest Common Subsequence



Longest Common Subsequence

Time Complexity = O(*Size of Table* x *Time to fill one slot*)

= O((m+1)(n+1)x O(1))

= O(mn)

= $O(n^2)$ if m = n

Space Complexity = O(*Size of Table*)

= O(*mn*) = $O(n^2)$ if m=n

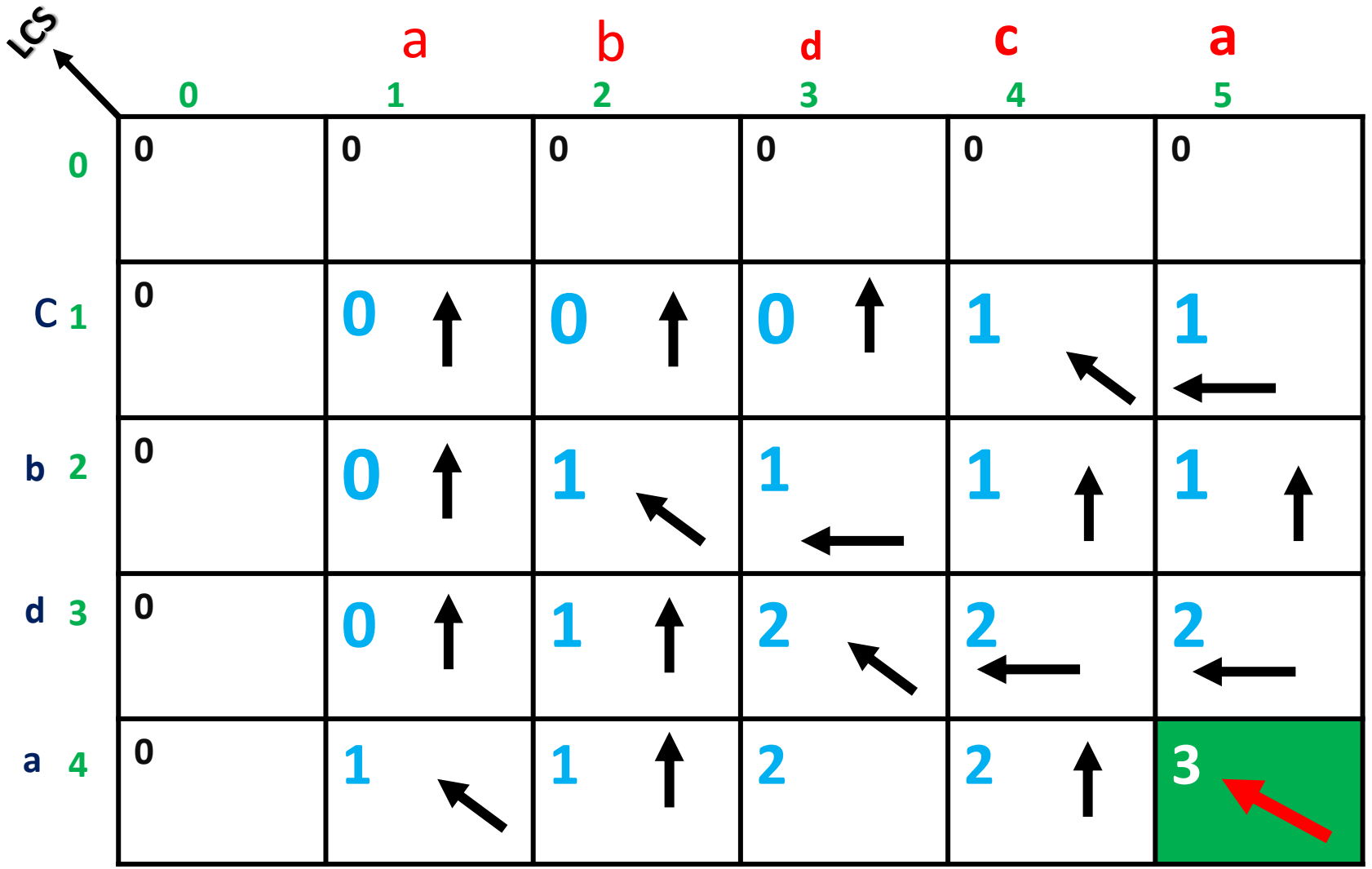
Longest Common Subsequence

```
LCS-LENGTH(X, Y)
 1  m = X.length
 2  n = Y.length
 3  let b[1..m, 1..n] and c[0..m, 0..n] be new tables
 4  for i = 1 to m
 5      c[i, 0] = 0
 6  for j = 0 to n
 7      c[0, j] = 0
 8  for i = 1 to m
 9      for j = 1 to n
10          if x_i == y_j
11              c[i, j] = c[i − 1, j − 1] + 1
12              b[i, j] = "↖"
13          elseif c[i − 1, j] ≥ c[i, j − 1]
14              c[i, j] = c[i − 1, j]
15              b[i, j] = "↑"
16          else c[i, j] = c[i, j − 1]
17              b[i, j] = "←"
18  return c and b
```

Longest Common Subsequence

How to get any one of the LCS from Table ?

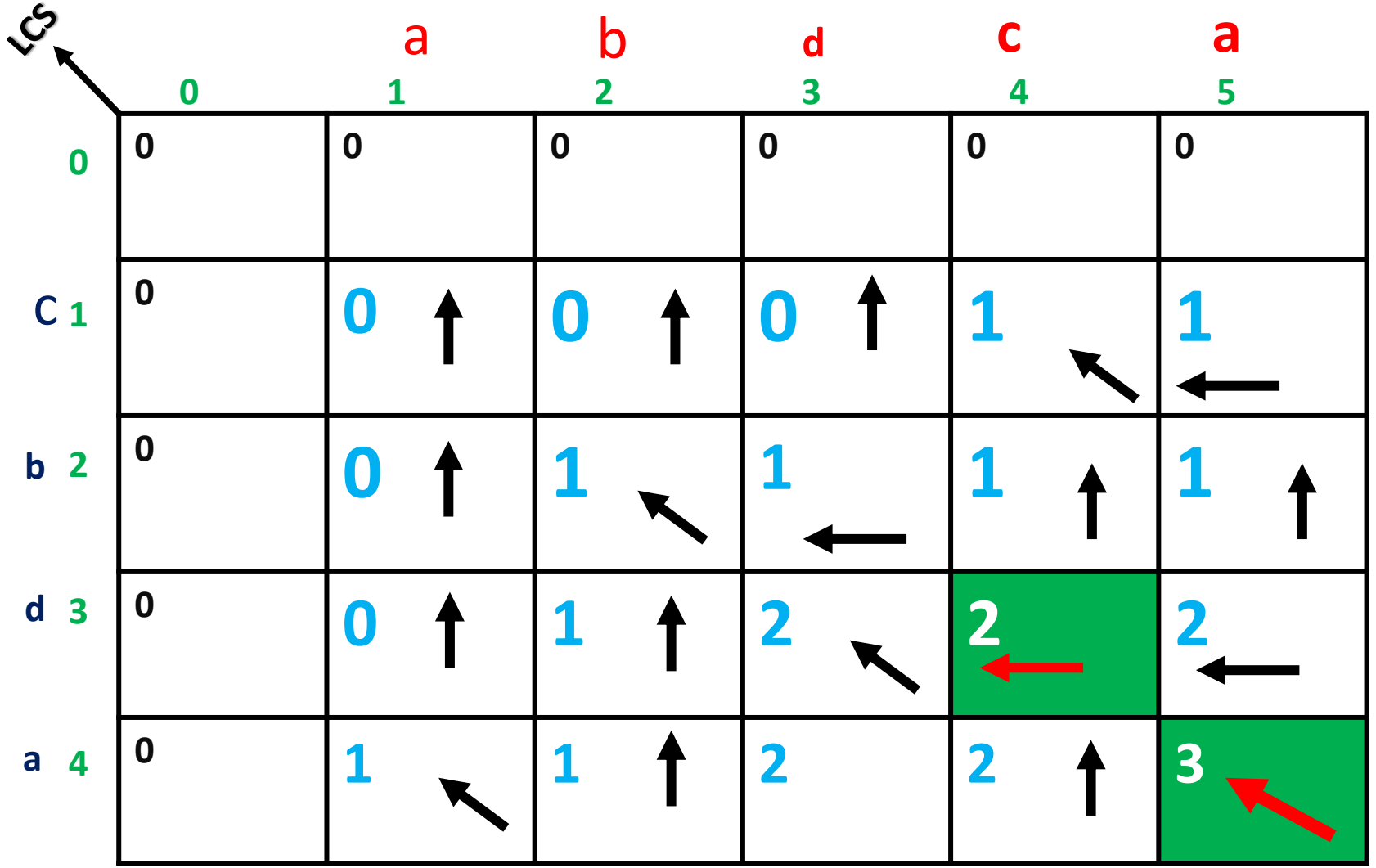
Longest Common Subsequence



LCS

a

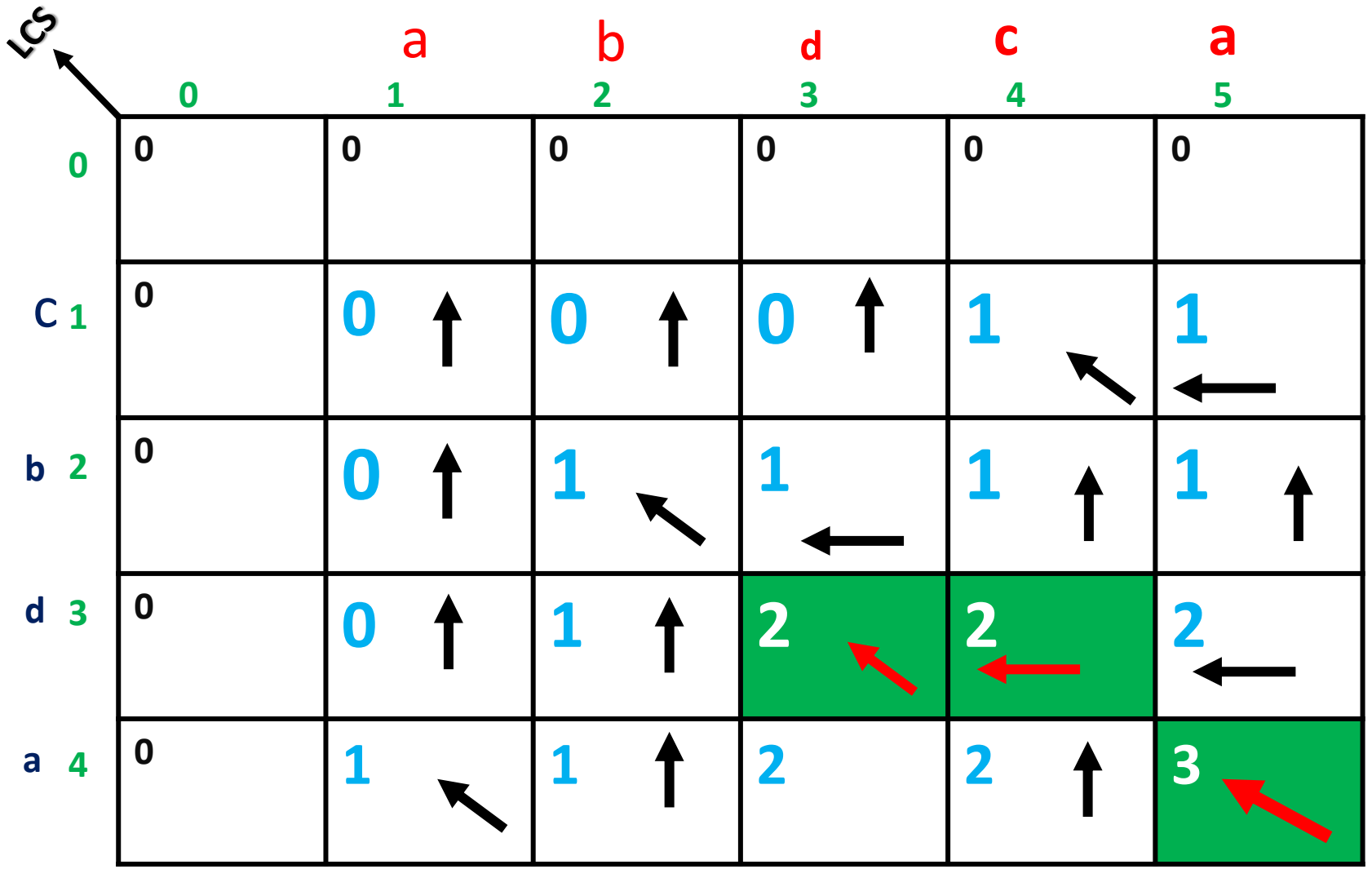
Longest Common Subsequence



LCS

a

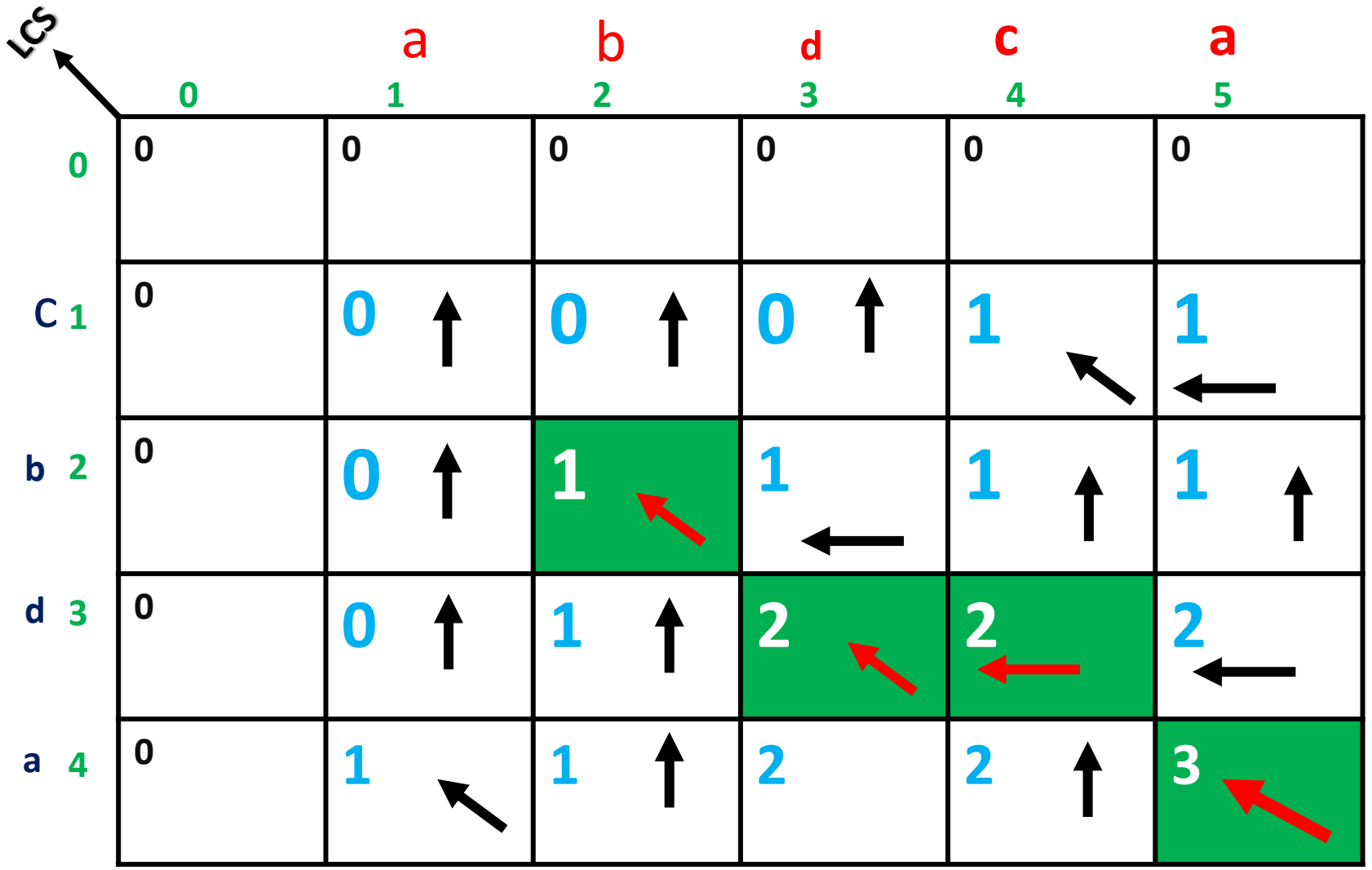
Longest Common Subsequence



LCS

d a

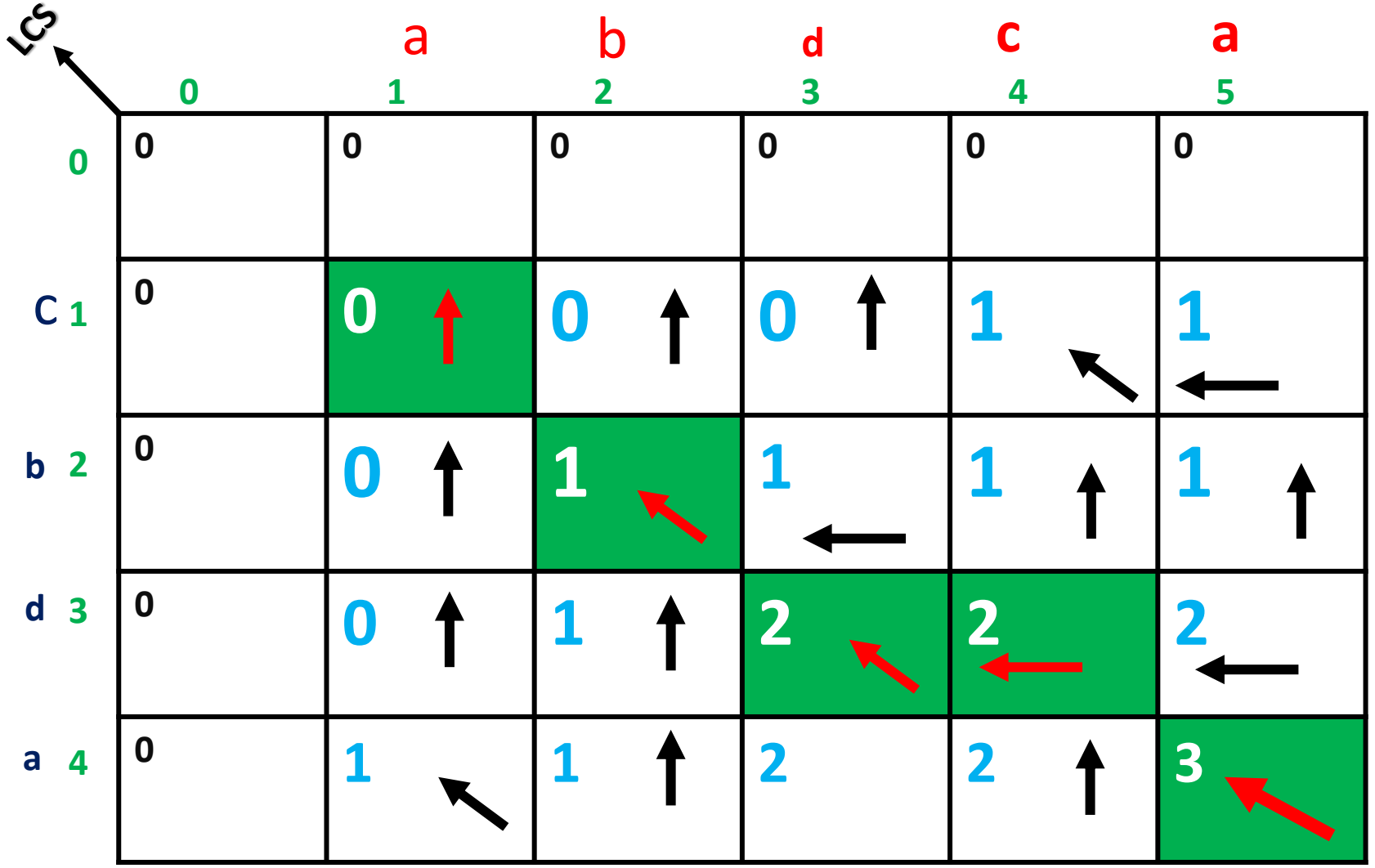
Longest Common Subsequence



LCS

b d a

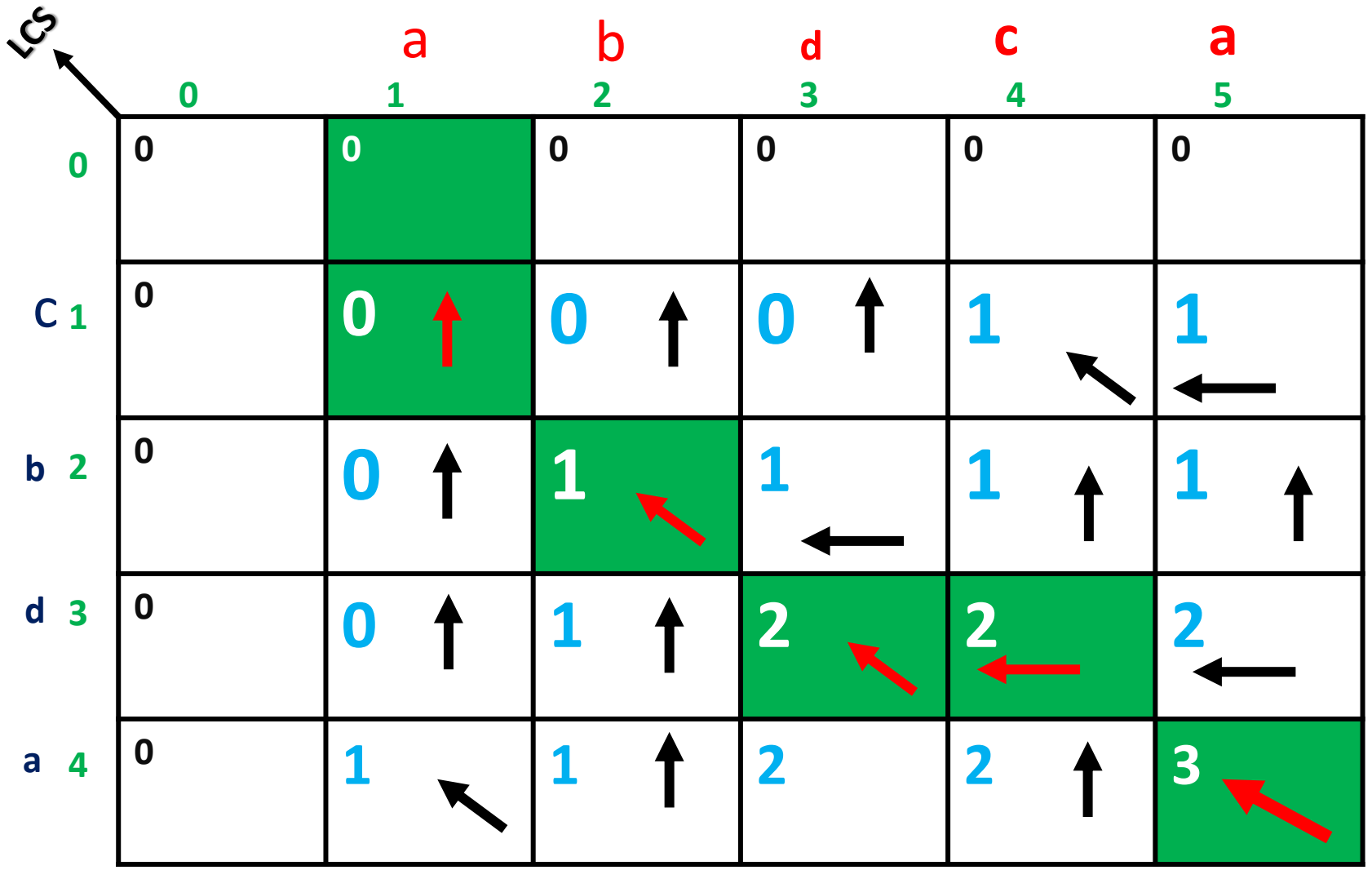
Longest Common Subsequence



LCS

b d a

Longest Common Subsequence



LCS

b d a

Longest Common Subsequence

How to get any one of the LCS from Table ?

```
PRINT-LCS(b, X, i, j)
 1 if i = 0 or j = 0 then
 2     return
 3 if b[i, j] = "↖" then
 4     PRINT-LCS(b, X, i − 1, j − 1)
 5     print x_i
 6 else if b[i, j] = "↑" then
 7     PRINT-LCS(b, X, i − 1, j)
 8 else
 9     PRINT-LCS(b, X, i, j − 1)
```

Longest Common Subsequence

How to get any one of the LCS from Table ?

Time Complexity = O(*Length of LCS*)
= O(L)

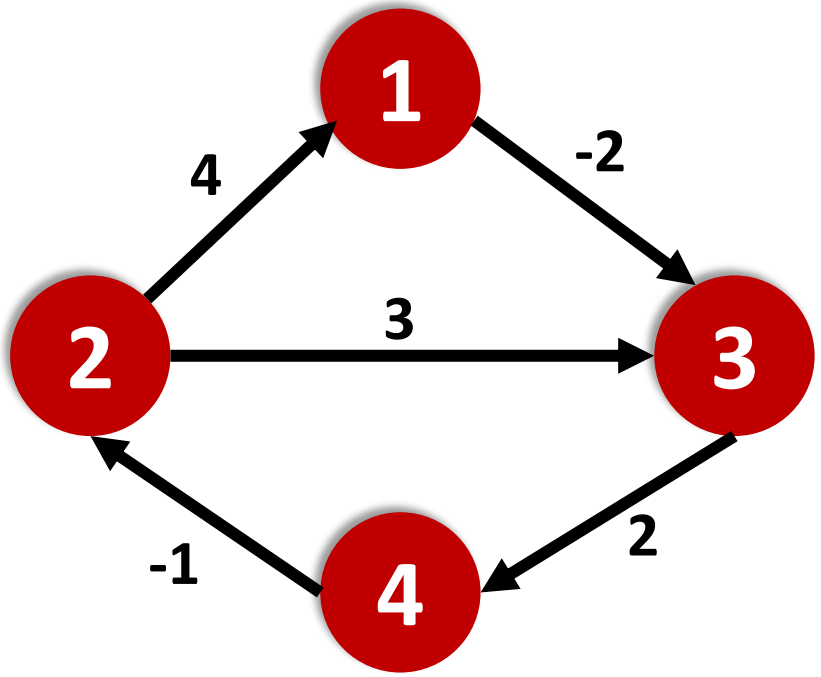
All Pair Shortest Path Problem

Given:

A weighted directed graph

Goal:

Compute the cost of shortest path b/w all pair of vertices



Source	Destination	Shortest Path(SP)	Cost(SP)
1	1	?	?
1	2	?	?
1	3	?	?
1	4	?	?
2	1	?	?
2	2	?	?
2	3	?	?
2	4	?	?
3	1	?	?
3	2	?	?
3	3	?	?
3	4	?	?
4	1	?	?
4	2	?	?
4	3	?	?
4	4	?	?

Floyd Warshall Algorithm

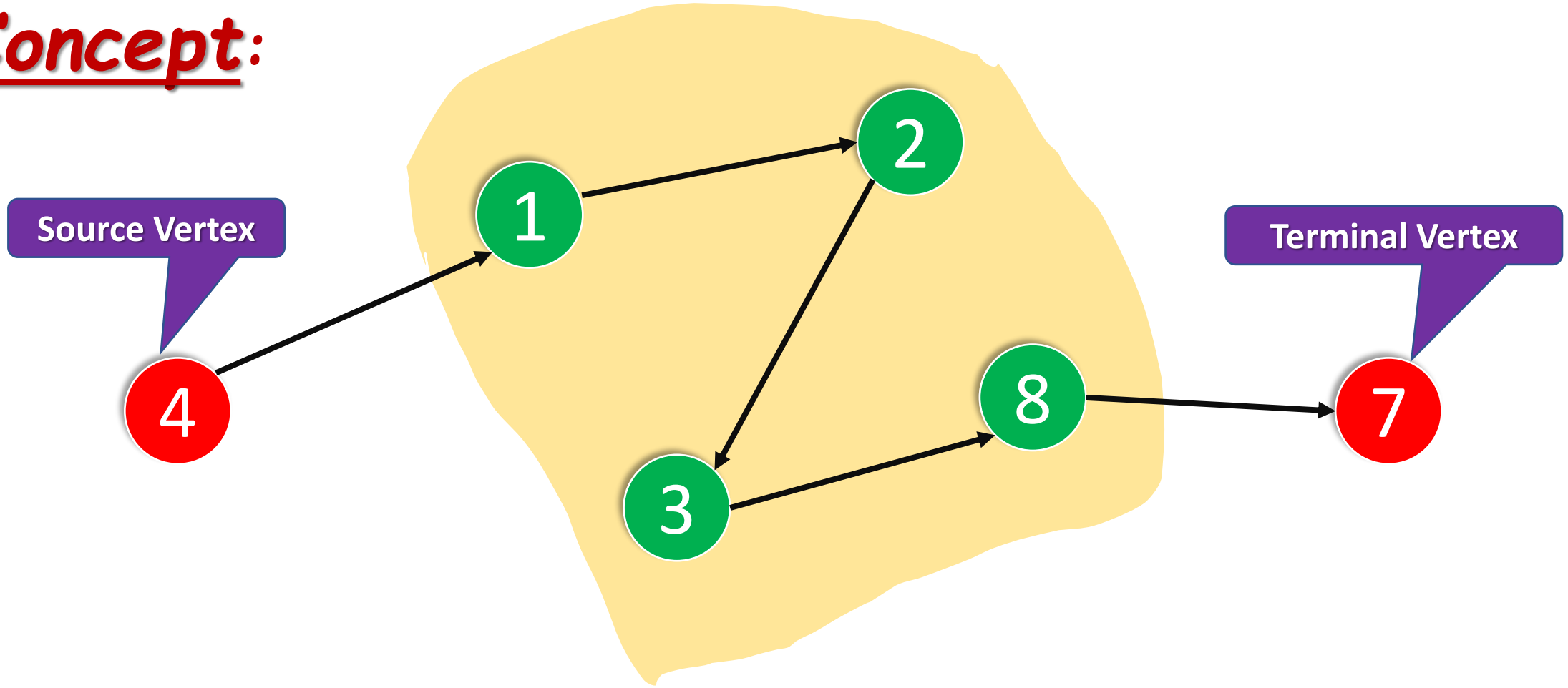
Floyd-Warshall's Algorithm(1962)

Floyd

Warshall

Floyd-Warshall's Algorithm

Concept:



Floyd-Warshall's Algorithm

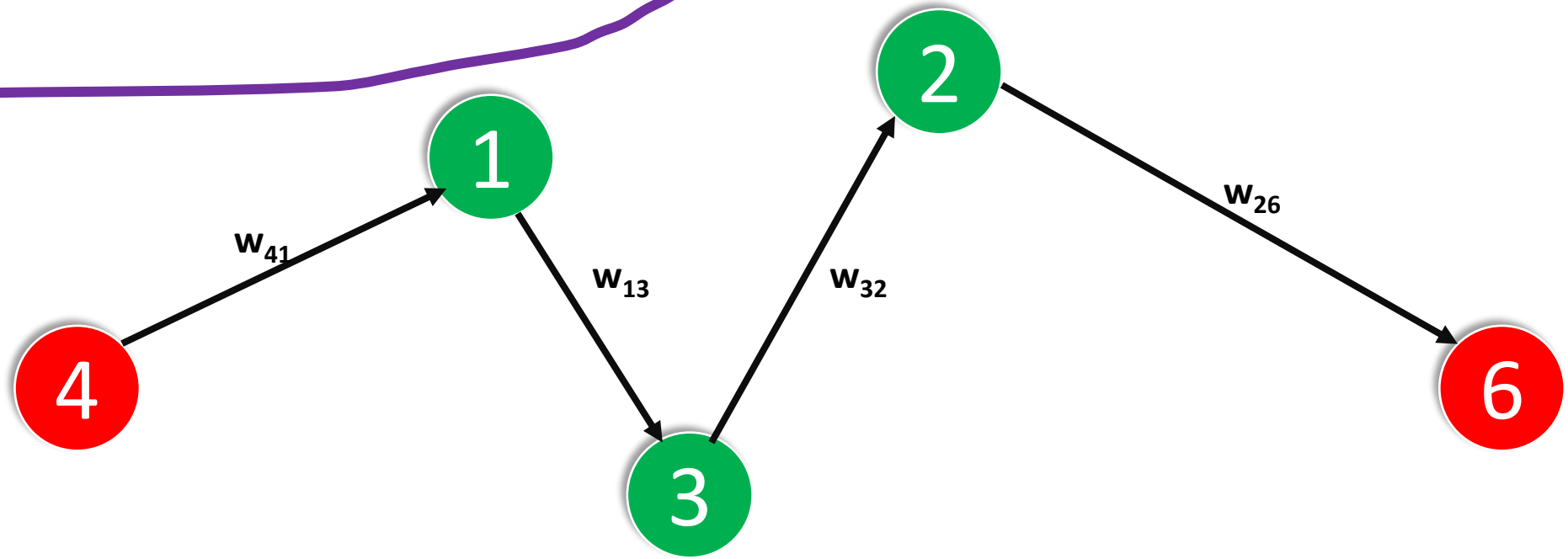
Recursive Formulation:

Vertices are numbered as {1, 2, 3, 4,……,n}

d_{ij}^{k} = *Cost of shortest path b/w* **i** *and* **j** *where the set of intermediate verticesis* $\{1, 2, 3, \ldots\ldots, k\}$

Floyd-Warshall's Algorithm

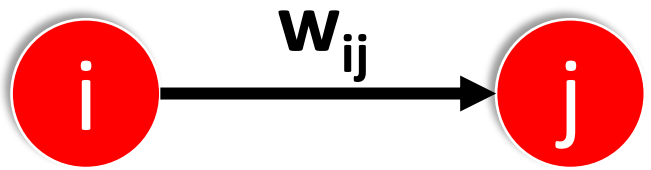
d_{46}^{3} = *Cost of shortest path b/w 4 and 6 where the set of intermediate verticesis* $\{1, 2, 3\}$



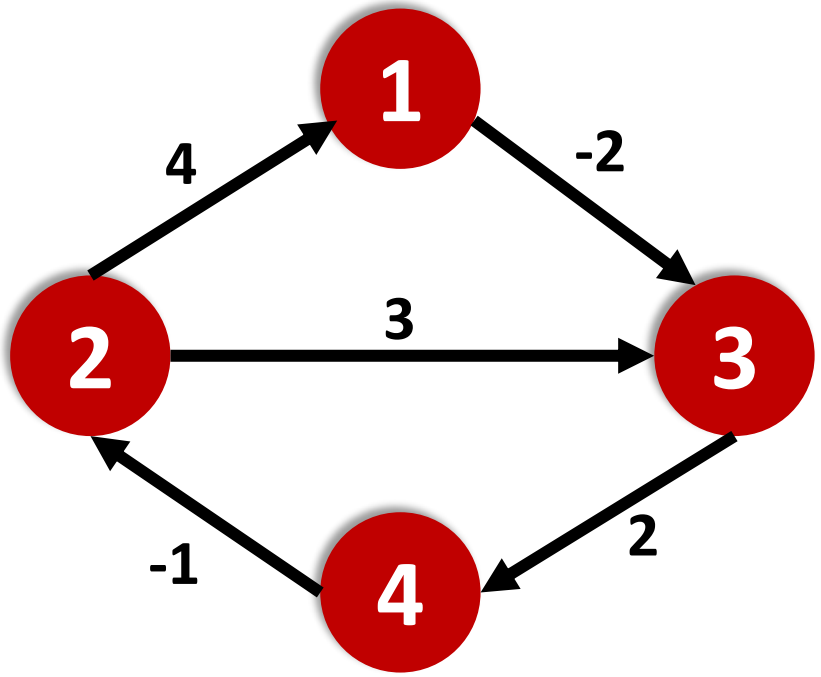
Floyd-Warshall's Algorithm

Base Case:

$$d_{ij}^{0} = w_{ij}$$



Floyd-Warshall's Algorithm



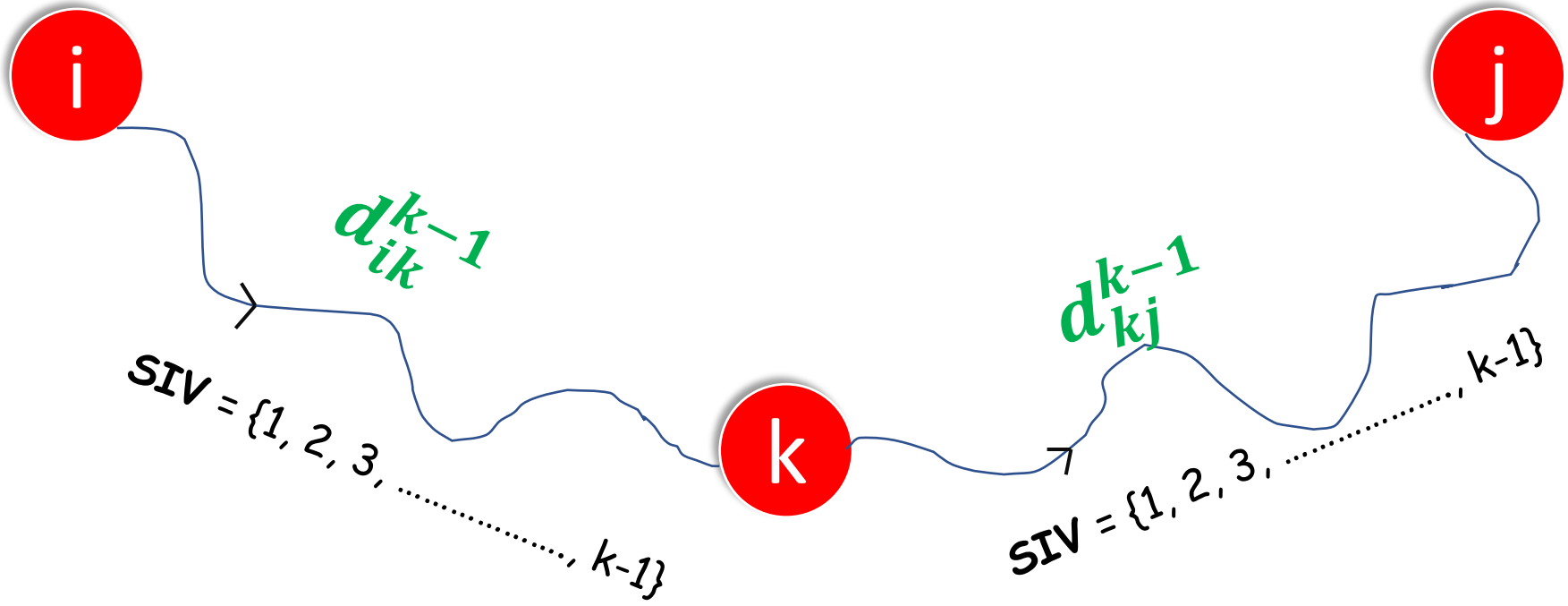
k=0		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	-1	∞	0

D^0

Floyd-Warshall's Algorithm

Recursive Case: K is the intermediate vertex

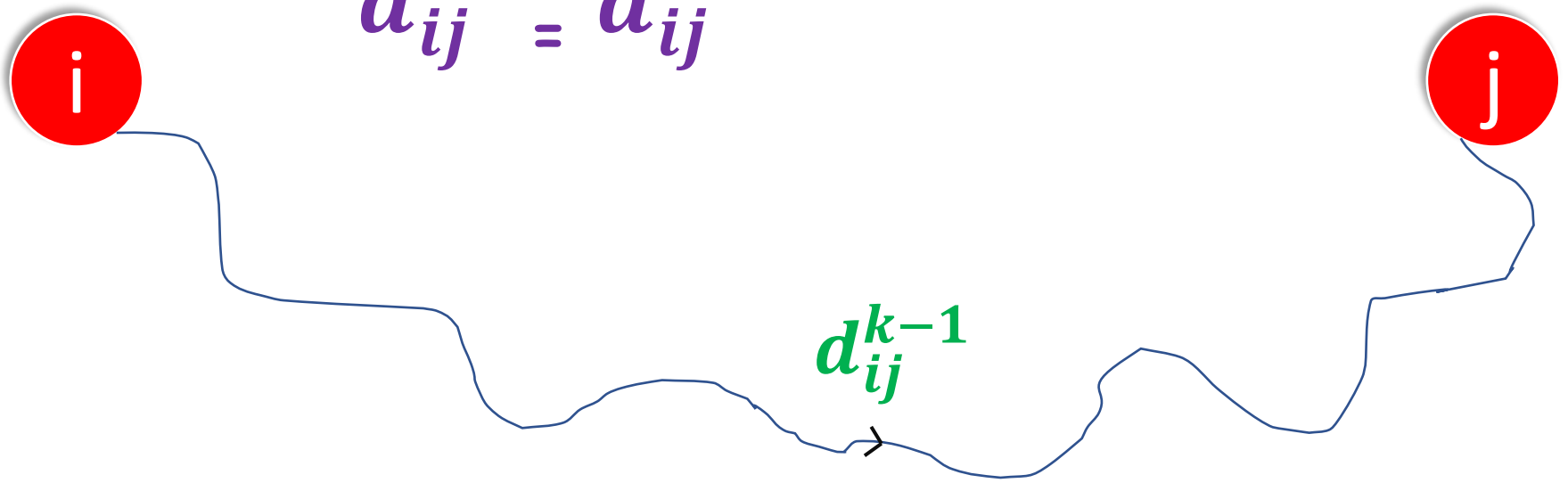
$$d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$$



Floyd-Warshall's Algorithm

Recursive Case: K is the **NOT** an intermediate vertex

$$d_{ij}^{k} = d_{ij}^{k-1}$$



SIV = {1, 2, 3, ……., k-1}

Floyd-Warshall's Algorithm

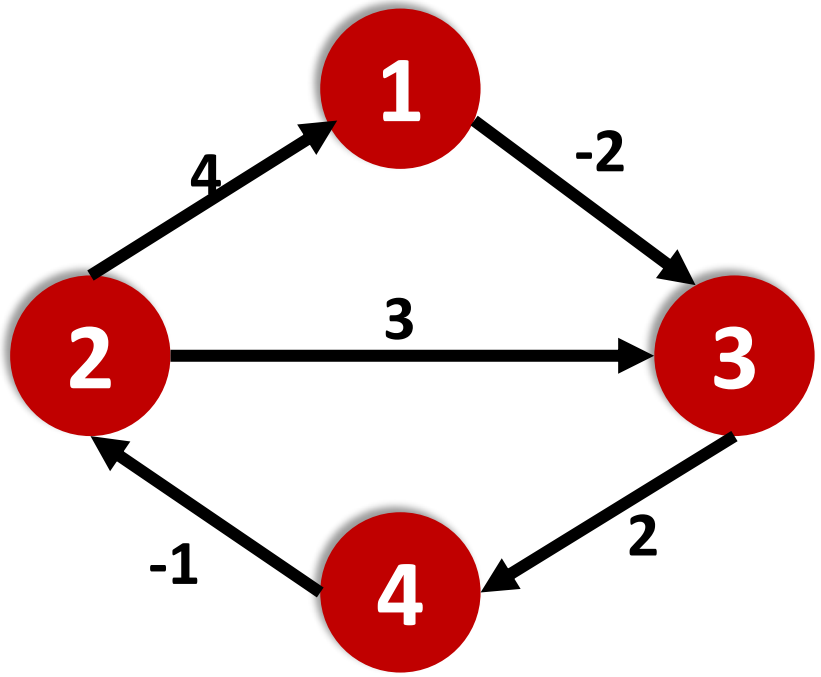
Recursive Formulation:

$$d_{ij}^{k} = \begin{cases} w_{ij} & k=0 \\ min \begin{cases} d_{ij}^{k-1} & \text{If k is NOT an intermediate vertex} \\ d_{ik}^{k-1} + d_{kj}^{k-1} & \text{If k is an intermediate vertex} \end{cases} \end{cases}$$

Floyd-Warshall's Algorithm

```
FLOYD-WARSHALL(W, n)
1  D^(0) = W
2  for k = 1 to n
3      let D^(k) = (d_ij^(k)) be a new n × n matrix
4      for i = 1 to n
5          for j = 1 to n
6              d_ij^(k) = min {d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1)}
7  return D^(n)
```

Floyd-Warshall's Algorithm



k=0		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	-1	∞	0

D^0

Floyd-Warshall's Algorithm

k=0		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	-1	∞	0

D^0

k=1		j			
		1	2	3	4
i	1	?			
	2				
	3				
	4				

D^1

$$d_{11}^1 = min \begin{cases} d_{11}^0 \\ d_{11}^0 + d_{11}^0 \end{cases}$$

$$d_{11}^1 = min \begin{cases} 0 \\ 0 \end{cases}$$

$$d_{11}^1 = 0$$

Floyd-Warshall's Algorithm

k=0		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	-1	∞	0

D^0

k=1		j			
		1	2	3	4
i	1	0			
	2				
	3				
	4				

D^1

$$d_{11}^1 = min \begin{cases} d_{11}^0 \\ d_{11}^0 + d_{11}^0 \end{cases}$$

$$d_{11}^1 = min \begin{cases} 0 \\ 0 \end{cases}$$

$$d_{11}^1 = 0$$

Floyd-Warshall's Algorithm

k=0		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	-1	∞	0

D^0

k=1		j			
		1	2	3	4
i	1	0	?		
	2				
	3				
	4				

D^1

$$d_{12}^1 = min \begin{cases} d_{12}^0 \\ d_{11}^0 + d_{12}^0 \end{cases}$$

$$d_{12}^1 = min \begin{cases} \infty \\ 0 + \infty \end{cases}$$

$$d_{12}^1 = \infty$$

Floyd-Warshall's Algorithm

k=0		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	-1	∞	0

D^0

k=1		j			
		1	2	3	4
i	1	0	∞		
	2				
	3				
	4				

D^1

$$d_{12}^1 = min \begin{cases} d_{12}^0 \\ d_{11}^0 + d_{12}^0 \end{cases}$$

$$d_{11}^1 = min \begin{cases} \infty \\ 0 + \infty \end{cases}$$

$$d_{11}^1 = \infty$$

Floyd-Warshall's Algorithm

k=0		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	-1	∞	0

D^0

k=1		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2				
	3				
	4				

D^1

Floyd-Warshall's Algorithm

k=0		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	-1	∞	0

D^0

k=1		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	?			
	3				
	4				

D^1

$$d_{21}^1 = min \begin{cases} d_{21}^0 \\ d_{21}^0 + d_{11}^0 \end{cases}$$

$$d_{11}^1 = min \begin{cases} 4 \\ 4 + 0 \end{cases}$$

$$d_{11}^1 = 4$$

Floyd-Warshall's Algorithm

k=0		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	-1	∞	0

D^0

k=1		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4			
	3				
	4				

D^1

$$d_{21}^1 = min \begin{cases} d_{21}^0 \\ d_{21}^0 + d_{11}^0 \end{cases}$$

$$d_{11}^1 = min \begin{cases} 4 \\ 4 + 0 \end{cases}$$

$$d_{11}^1 = 4$$

Floyd-Warshall's Algorithm

k=0		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	3	∞
	3	∞	∞	0	2
	4	∞	-1	∞	0

D^0

Floyd-Warshall's Algorithm

k=1		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	2	∞
	3	∞	∞	0	2
	4	∞	-1	∞	0

D^1

Floyd-Warshall's Algorithm

k=2		j			
		1	2	3	4
i	1	0	∞	-2	∞
	2	4	0	2	∞
	3	∞	∞	0	2
	4	3	-1	1	0

D^2

Floyd-Warshall's Algorithm

k=3		j			
		1	2	3	4
i	1	0	∞	-2	0
	2	4	0	2	4
	3	∞	∞	0	2
	4	3	-1	∞	0

D^3

Floyd-Warshall's Algorithm

k=4		j			
		1	2	3	4
i	1	0	-1	-2	0
	2	4	0	2	4
	3	5	1	0	2
	4	3	-1	1	0

D^4

Floyd-Warshall's Algorithm

```
FLOYD-WARSHALL(W, n)
1  D^(0) = W
2  for k = 1 to n
3      let D^(k) = (d_ij^(k)) be a new n × n matrix
4      for i = 1 to n
5          for j = 1 to n
6              d_ij^(k) = min {d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1)}
7  return D^(n)
```

Floyd-Warshall's Algorithm

Time Complexity = $O(n^3)$

Space Complexity = $O(n^2)$