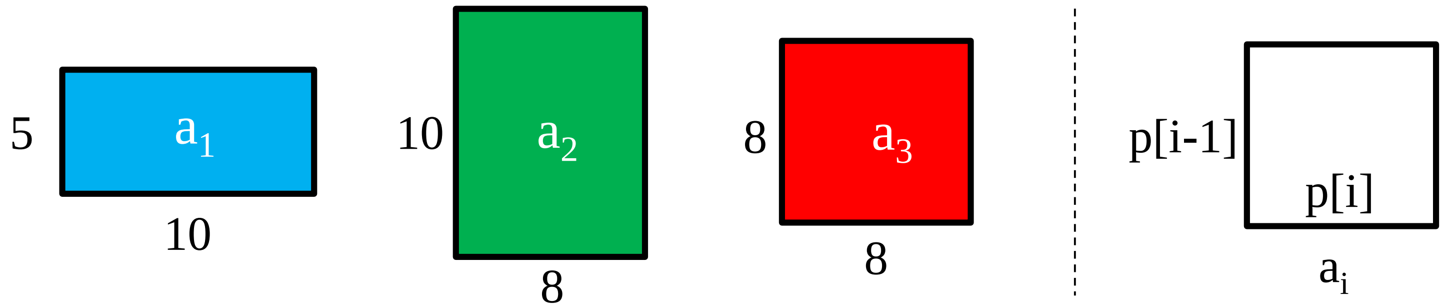
- **Matrix-Chain Multiplication**

Input: A sequence of matrices $a_1, a_2, \ldots a_n$ and dimensions p[0… n]

Output: The minimum number of multiplications required

Example:

p = [5, 10, 8, 8]



Revisit:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{22} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

number of multiplications: 2 * (2 * 2)

General case: $A[m_1, n_1] * B[m_2, n_2] = C[m_1, n_2]$, $n_1 = m_2$

number of multiplications: $\mathbf{n_1 * (m_1 * n_2)}$

- $\underbrace{(a_1[5,10] * a_2[10,8])}_{[5,8]} * a3[8,8]$

  10 * (5 * 8) + 8 * (5 * 8) = **720**

- $a_1[5,10] * \underbrace{(a_2[10,8] * a3[8,8])}_{[10,8]}$

  8 * (10 * 8) + 10 * (5 * 8) = **1040**

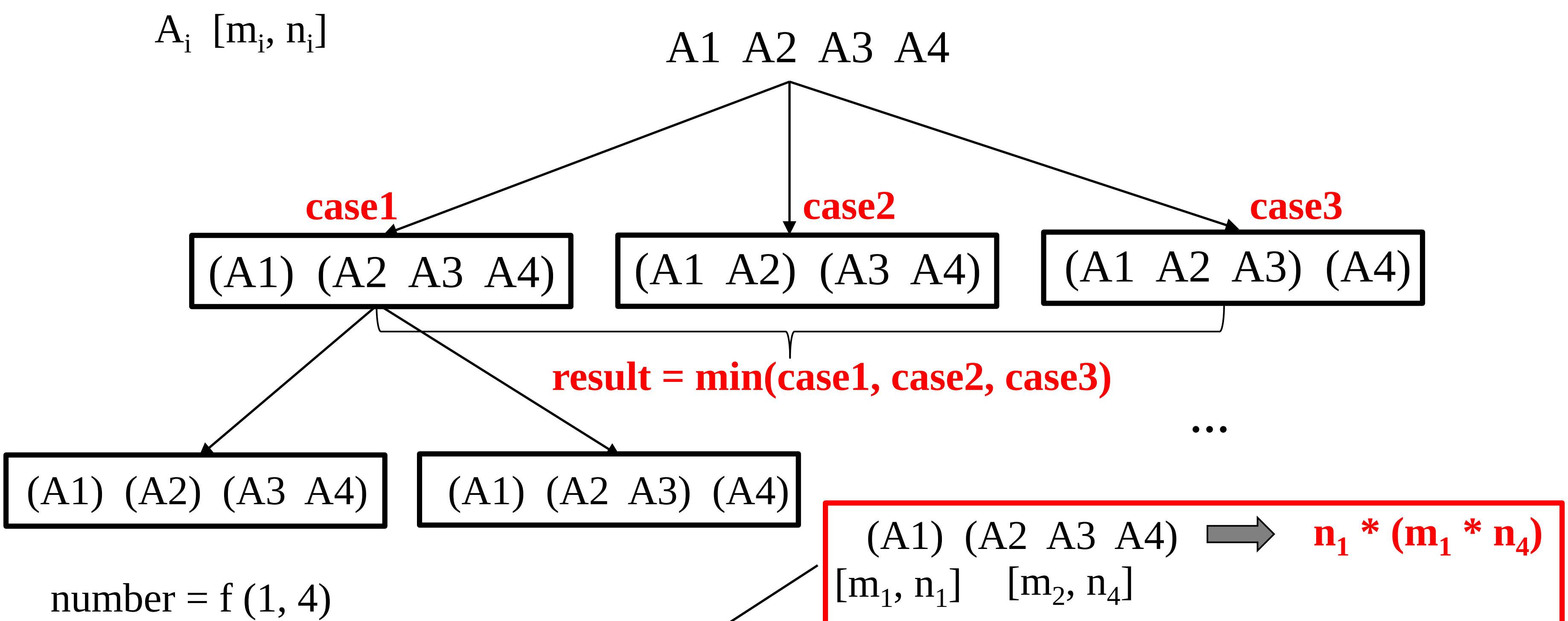
- Not perform the multiplications, but to decide in which **order** to perform the multiplications.

	Time complexity	Space complexity
Naive Recursion:	$O(4^n / n^{3/2})$	$O(n)$
DP:	$O(n^3)$	$O(n^2)$

1) Naive Recursion

General case: $A[m_1, n_1] * B[m_2, n_2] = C[m_1, n_2]$, $n_1 = m_2$

number of multiplications: $\mathbf{n_1 * (m_1 * n_2)}$



number = f (1, 4)

f (1, 4) = min($f(1, 1) + f(2, 4) + n_1 * (m_1 * n_4)$, **case1**

$f(1, 2) + f(3, 4) + n_2 * (m_1 * n_4)$, **case2**

$f(1, 3) + f(4, 4) + n_3 * (m_1 * n_4)$) **case3**

1) Naive Recursion

Pseudocode:

matrixChainMultiplication (p[0..n])

Input: : A dimension array p[0… n] of sequence of matrices
Output: : The minimum number of multiplications

```
return countMultiplication(p, 1, n)
```

countMultiplication (p[0..n], i, j)

```
if i = j do return 0
result = inf
for k ← i to j - 1 do
    count ← (countMultiplication(p, i, k) +
                countMultiplication(p, k+1, j) +
                p[k] * p[i-1] * p[j])
    if count < result do result ← count
return result
```

p[0… n] p[i-1] p[i]

a_i

$$f(1, 4) = \min(f(1, 1) + f(2, 4) + n_1 * (m_1 * n_4),$$
$$f(1, 2) + f(3, 4) + n_2 * (m_1 * n_4),$$
$$f(1, 3) + f(4, 4) + n_3 * (m_1 * n_4))$$

i = 1 k = 1 j = 4

(A1) (A2 A3 A4)

k = 2

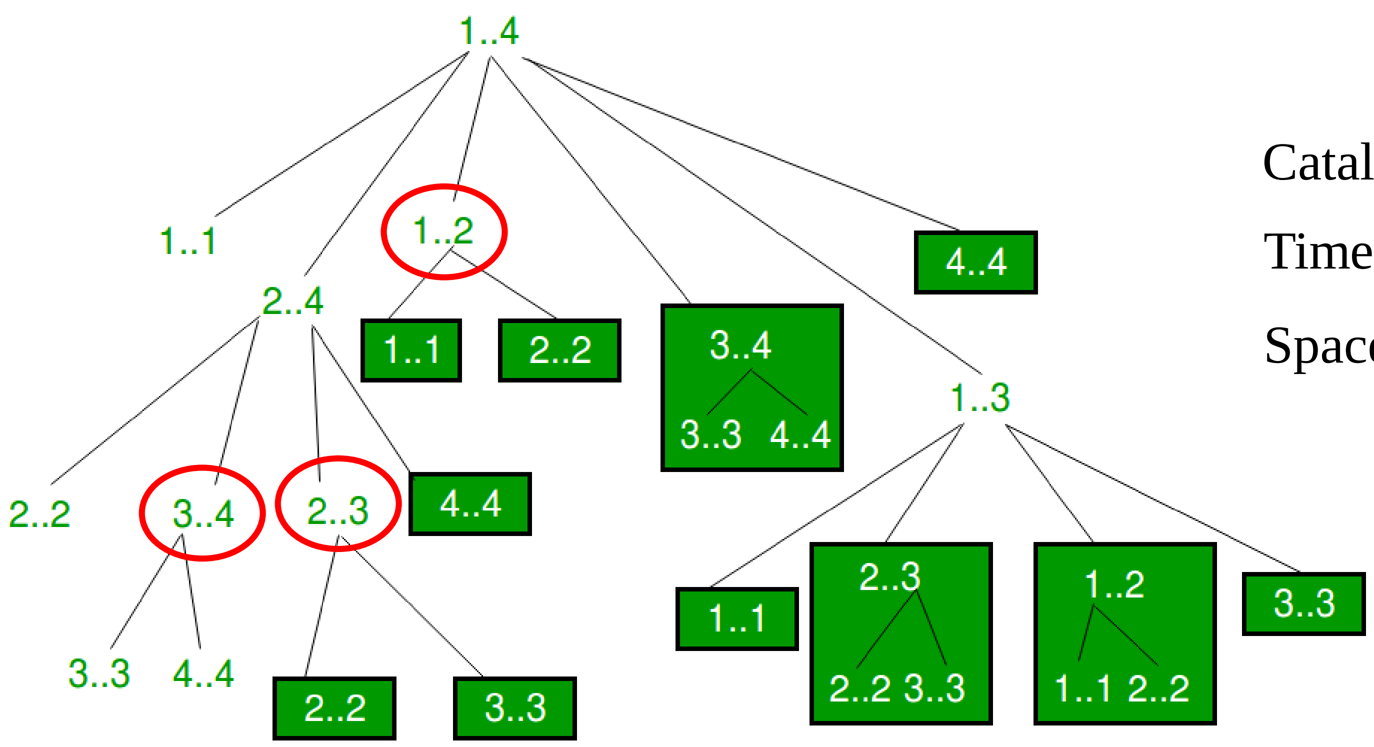
(A1 A2) (A3 A4)

k = 3

(A1 A2 A3) (A4)

1) Naive Recursion

Recursion tree:



Catalan number

Time complexity: $O(4^n / n^{3/2})$

Space complexity: $O(n)$

https://www.geeksforgeeks.org/matrix-chain-multiplication-dp-8/

2) Dynamic Programming

- Subproblem:

$M[i, j]$ = Minimum number of multiplications needed to compute the matrix-chain product $a_i \times a_{i+1} \times \cdots \times a_j$, where the size of matrix a_i is $p[i-1] \times p[i]$.

Compute $M[1, n]$.

https://www3.cs.stonybrook.edu/~pramod.ganapathi/doc/algorithms/Algo-DynamicProgramming.pdf

2) Dynamic Programming

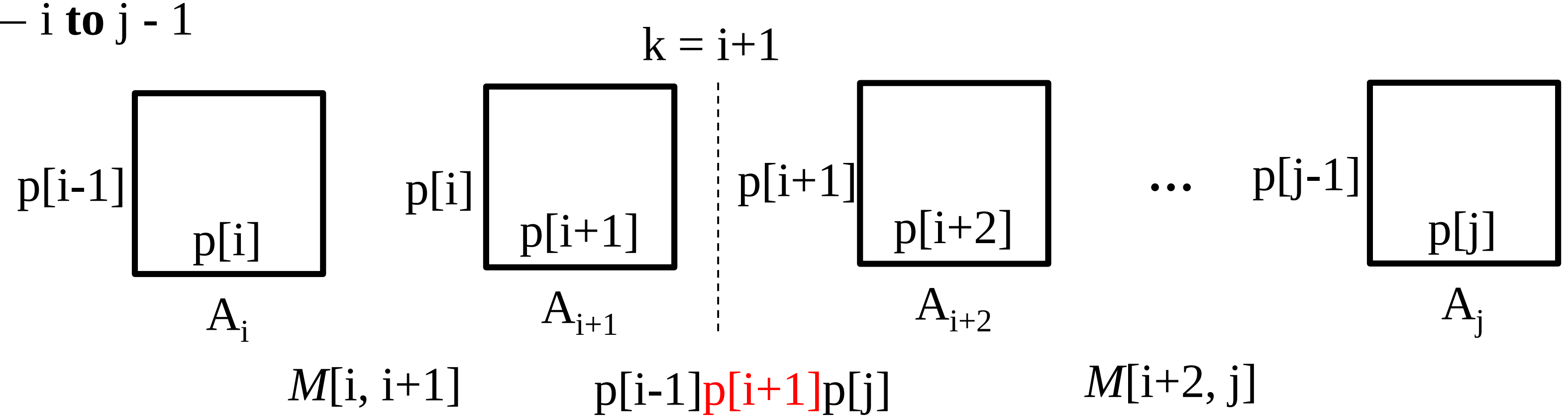
$A[m_1, n_1] * B[m_2, n_2] = C[m_1, n_2], n_1 = m_2$

number of multiplications: $\mathbf{n_1 * (m_1 * n_2)}$

- Recurrence:

$$M[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min \begin{Bmatrix} M[i,\overset{k=i}{i}] + M[\overset{k+1}{i+1}, j] + p[i-1]p[\overset{k=i}{i}]p[j], \\ M[i,\underset{k=i+1}{i+1}] + M[\underset{k+1}{i+2}, j] + p[i-1]p[\underset{k=i+1}{i+1}]p[j], \\ \cdots \\ M[i,j-1] + M[j,j] + p[i-1]p[j-1]p[j], \end{Bmatrix} & \text{if } i < j. \end{cases}$$

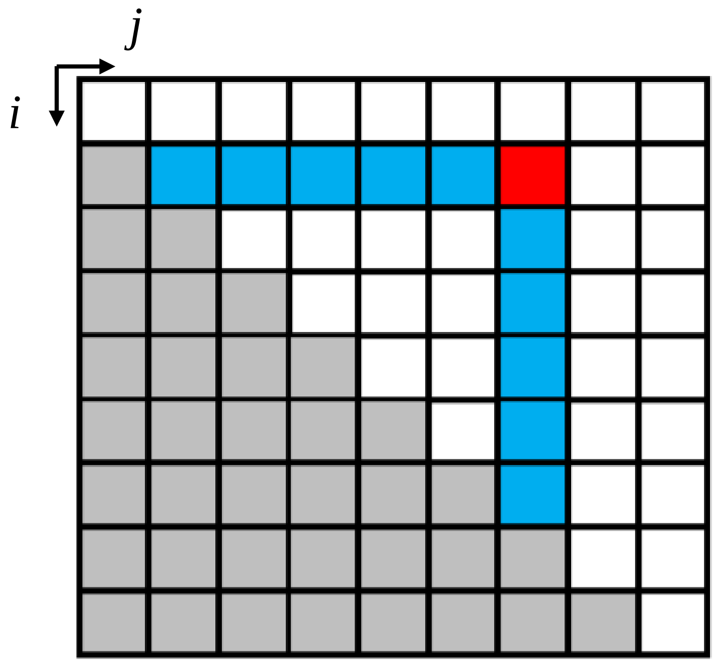
k ← i **to** j - 1



https://www3.cs.stonybrook.edu/~pramod.ganapathi/doc/algorithms/Algo-DynamicProgramming.pdf

2) Dynamic Programming

- Dependency:



$$M[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min \begin{cases} M[i,i] + M[i+1,j] + p[i-1]p[i]p[j], \\ M[i,i+1] + M[i+2,j] + p[i-1]p[i+1]p[j], \\ \dots \\ M[i,j-1] + M[j,j] + p[i-1]p[j-1]p[j], \end{cases} & \text{if } i < j. \end{cases}$$

https://www3.cs.stonybrook.edu/~pramod.ganapathi/doc/algorithms/Algo-DynamicProgramming.pdf

2) Dynamic Programming

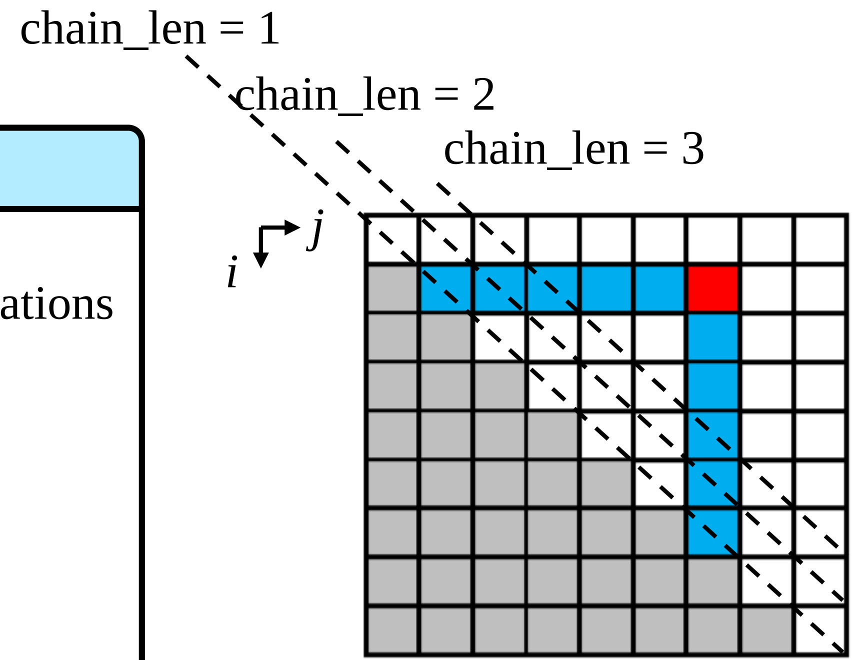
- Pseudocode:

matrixMultiplication (p[0..n])

Input: : A dimension array p[0.. n]

Output: : The minimum number of multiplications

```
for i ← 1 to n do
    M[i, i] ← 0
for chain_len ←  2 to n – 1 do
    for i ← 1 to n - chain_len + 1 do
        j ← i + chain_len – 1
        M[i, j] ← inf
        for k ← i to j - 1 do
            M[i, j] ← min(M[i , j],
                          M[i, k] + M[k + 1, j] +
                          p[i-1] * p[k] * p[j])
return M[1, n]
```



Time complexity: $O(n^3)$

Space complexity: $O(n^2)$

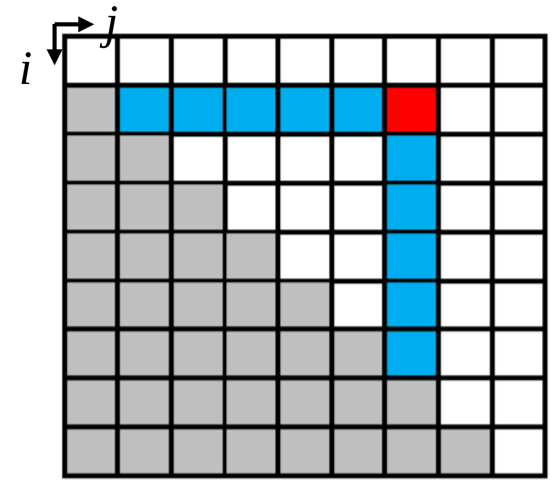
https://www3.cs.stonybrook.edu/~pramod.ganapathi/doc/algorithms/Algo-DynamicProgramming.pdf

2) Dynamic Programming

- Tables

i	0	1	2	3	4
P[i]	1	2	3	4	3

M[i, j]	j = 1	j = 2	j = 3	j = 4
i = 1	0	6	18	**30**
i = 2		0	24	48
i = 3			0	36
i = 4				0



min(M[i, k] + M[k + 1, j] + p[i-1] * p[k] * p[j])

ex. M[1, 3] = min(M[1, 1] + M[2, 3] + p[0] * p[1] * p[2], 0 + 24 + 6 = 30

M[1, 2] + M[3, 3] + p[0] * p[2] * p[3]) 6 + 0 + 12 = 18

→ M[1, 3] = 18

	Time complexity	Space complexity
Naive Recursion:	$O(4^n / n^{3/2})$	$O(n)$
DP:	$O(n^3)$	$O(n^2)$

There are more efficient algorithms. Ex. Hu & Shing. $O(n \log n)$

Hu, T. C., and M. T. Shing. 1981.