Fundamentos de análisis y diseño de algoritmos

Programación dinámica

LCS (Longest Common Subsequence)

Multiplicación de matrices

Propiedades generales de la programación dinamica

Al igual que la técnica de *Dividir y conquistar*, la programación dinámica es una técnica para resolver problemas a partir de la solución de subproblemas y la combinación de esas soluciones

A diferencia de *Dividir y conquistar*, la programación dinámica es aplicable cuando los subproblemas no son independientes

Un algoritmo que sigue esta técnica resuelve cada subproblema una sola vez y guarda su respuesta en una tabla.

La programación dinámica se aplica para resolver problema de optimización:

- Problemas en los que se pueden encontrar muchas soluciones
- · Cada solución tiene un valor asociado
- Se busca una solución que tenga un valor asociado que sea óptimo (máximo o mínimo) entre las muchas soluciones que pueden existir

Desarrollar un algoritmo usando programación dinámica consta de los siguientes pasos:

- · Caracterizar la estructura de la solución óptima
- · Definir recursivamente el valor de una solución óptima
- Calcular el valor de una solución óptima de manera bottom-up
- Construir una solución óptima a partir de la información calculada

Multiplicación de sucesión de matrices (MSM)

Suponga que desea multiplicar las matrices A_1, A_2, A_3 , de dimensiones 10x100, 100x5 y 5x50 respectivamente.

¿Cuántas formas existen de realizar la multiplicación?

Multiplicación de sucesión de matrices (MSM)

Suponga que desea multiplicar las matrices A_1, A_2, A_3 , de dimensiones 10x100, 100x5 y 5x50 respectivamente.

Se tienen dos formas:

$$A_1.(A_2.A_3)$$

$$(A_1.A_2)A_3$$

Se busca una solución que minimice la cantidad de multiplicaciones necesarias. Para conocer esta cantidad, analice el algoritmo MATRIX-MULTIPLY(A,B)

```
MATRIX-MULTIPLY(A,B)

if columns[A]\neqrows[B]

then error "incompatible dimensions"

else for i\leftarrow1 to rows[A]

do for j\leftarrow1 to columns[B]

do C[i,j] \leftarrow 0

for k\leftarrow1 to columns[A]

do C[i,j] \leftarrow C[i,j] + A[i,k].B[k,j]
```

Se busca una solución que minimice la cantidad de multiplicaciones necesarias. Para conocer esta cantidad, analice el algoritmo MATRIX-MULTIPLY(A,B)

```
MATRIX-MULTIPLY(A,B)

if columns[A]\neqrows[B]

then error "incompatible dimensions"

else for i\leftarrow1 to rows[A]

do for j\leftarrow1 to columns[B]

do C[i,j] \leftarrow 0

for k\leftarrow1 to columns[A]

do C[i,j] \leftarrow C[i,j] + A[i,k].B[k,j]
```

Sea A de dimensiones pxq y B de dimensiones qxr, la cantidad de multiplicaciones es pqr

Calcule la cantidad de multiplicaciones para las soluciones al problema dado

Multiplicar las matrices A_1, A_2, A_3 , de dimensiones 10x100, 100x5 y 5x50 respectivamente.

Se tienen dos formas:

$$A_{1}.(A_{2}.A_{3})$$
 $100150 - 100150$
 $(A_{1}.A_{2})A_{3}$ $25000 + 50000 - 75000$
 $101100150 - 7500$
 $101100150 - 7500$
 $101100150 - 7500$

Multiplicar las matrices A_1, A_2, A_3 , de dimensiones 10x100, 100x5 y 5x50 respectivamente.

Se tienen dos formas:

$$\cdot A_1.(A_2.A_3)$$

La cantidad de multiplicaciones de A_2 . A_3 es 100x5x50=2500, lo cual genera una matriz de 100x50.

Por lo que A_1 . $(A_2.A_3)$ lleva $10\times100\times50 + 2500=75000$ multiplicaciones

$$\cdot (A_1.A_2)A_3$$

La cantidad de multiplicaciones de $A_1.A_2$ es 10x100x5=5000, lo cual genera una matriz de 10x5.

Por lo que $(A_1.A_2)A_3$ lleva 10x5x50 + 5000=7500 multiplicaciones

Multiplicar las matrices A_1, A_2, A_3 , de dimensiones 10x100, 100x5 y 5x50 respectivamente.

Se tienen dos formas:

$$\cdot A_1.(A_2.A_3)$$

La cantidad de multiplicaciones de $A_2.A_3$ es 100x5x50=2500, lo cual genera una matriz de 100x50.

Por lo que A_1 . $(A_2.A_3)$ lleva $10\times100\times50 + 2500=75000$ multiplicaciones

$$\cdot (A_1.A_2)A_3$$

La cantidad de multiplicaciones de $A_1.A_2$ es 10x100x5=5000, lo cual genera una matriz de 10x5.

Por lo que $(A_1.A_2)A_3$ lleva 10x5x50 + 5000=7500 multiplicaciones

La solución óptima es (A1.A2)A3

Problema: Multiplicación de una sucesión de matrices

Entrada: una sucesión $\langle A_1, ..., A_n \rangle$ de n matrices, donde A_i tiene dimensiones $p_{i-1}xp_i$

Salida: la manera óptima de multiplicar las matrices

Una solución ingenua al problema MSM consiste en:

- Enumerar todas las posibles maneras de multiplicar las n matrices, calculando su costo (cantidad de multiplicaciones)
- · Escoger la de menor costo (menos multiplicaciones)

El número de soluciones es exponencial sobre n, la cantidad de matrices a multiplicar

1. Caracterizar la estructura de una solución óptima

Sea $A_{i..j}$ la matriz que resulta de evaluar $A_i...A_j$

•Una solución óptima para calcular $A_{1..n}$ se divide en una solución óptima para calcular $A_{1..k}$ y una solución óptima para calcular $A_{k+1..n}$

El costo de una solución óptima es la suma de: AS AG)

- ·Costo de una solución óptima para A_{1..k}
- ·Costo de una solución óptima para $A_{k+1...n}$
- •Costo de multiplicar $A_{1..k}$ por $A_{k+1...n}$

Caracterización de la solución optima

- 1) Una solución óptima esta compuesta por soluciones óptimas de los subproblemas
- 2) Se define la recursión. Responder ¿Como se combinan los subproblemas para encontrar la solución óptima?

Para el caso de las multiplicación de matrices.

Este a su vez se parte en dos problemas (multiplicar dos matrices a la vez)

1) Ai ... Ak

Pi-1Pi ... Pk-1Pk Este subproblema da una matriz de que tamaño Pi-1Pk 2) Ak+1, ... Aj

PkPk+1.. Pj-1Pj

Este problema da una matriz de tamaño PkPj

Si estos problemas tienen solución optima, entonces la solución global es optima (Garantía de programación dinámica)

Componemos: Recusiva:

Solución optima: Costo(Ai...Ak) + Costo(Ak+1,...Aj)+Pi-1*Pk*Pj

1. Caracterizar la estructura de una solución óptima

Toda solución óptima para el problema $A_{1..n}$ contiene dentro de sí, soluciones óptimas para los subproblemas encontrados

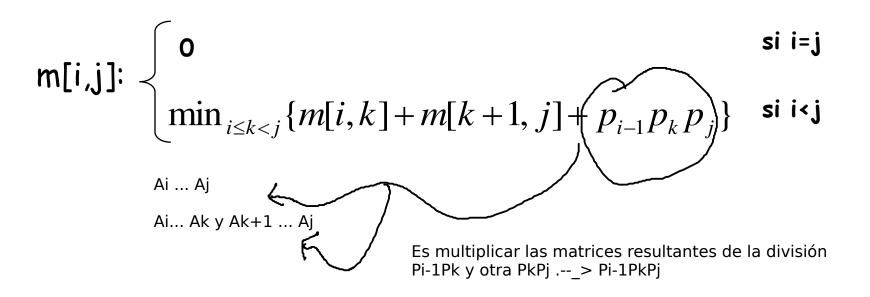
Esta propiedad de subestructuras óptimas dentro de soluciones óptimas es un indicador de la aplicabilidad de la técnica de programación dinámica

2. Definir recursivamente el valor de una solución óptima m[i,j]: mínimo número de multiplicaciones necesarias para calcular Ai..j

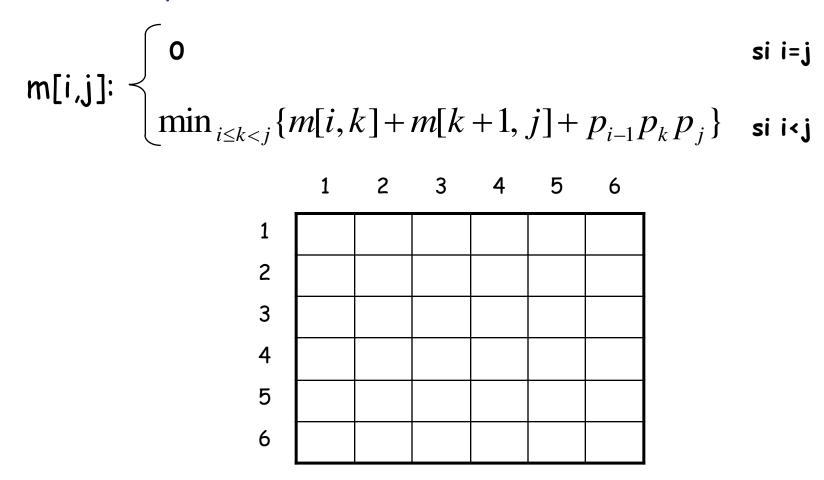
- ·Se mantiene una matriz para ir almacenando los resultados óptimos de los subproblemas
- ·El problema original es calcular m[1,n]

En este punto agregamos la estructura de memorización.

2. Definir recursivamente el valor de una solución óptima



3. Calcular el valor de una solución óptima de manera bottom-up



3. Calcular el valor de una solución óptima de manera bottom-up

$$m[i,j]: \begin{cases} 0 & \text{si } i=j \\ \min_{i \leq k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{si } i < j \end{cases}$$

Porque comenzamos por la soluciones trivales y terminamos en la solución general (completa)

3. Calcular el valor de una solución óptima de manera bottom-up

$$\mathsf{m[i,j]:} \begin{cases} \mathsf{0} & \mathsf{si} \ \mathsf{i=j} \\ \min_{i \leq k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \} & \mathsf{si} \ \mathsf{i$$

•Solo se define m cuando i<j

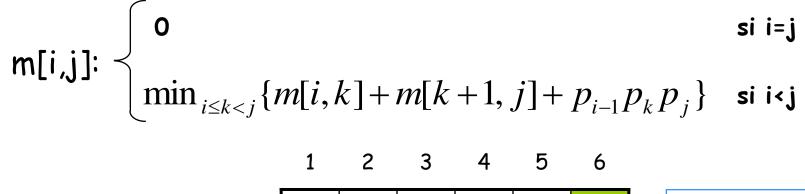
·Las celdas con X no se calcularán

5

6

1	2	3	4	5	6
0					
X	0				
X	X	0			
X	X	X	0		
X	X	X	X	0	
X	X	X	X	X	0

3. Calcular el valor de una solución óptima de manera bottom-up



3

5

6

_	-	_		•		
	0					
	X	0				
Ī	X	X	0			
Î	X	X	X	0		
I	X	X	X	X	0	
	X	X	X	X	X	0

- m[1,6] es el valor que se quiere calcular
- De qué depende m[1,6]?

3. Calcular el valor de una solución óptima de manera

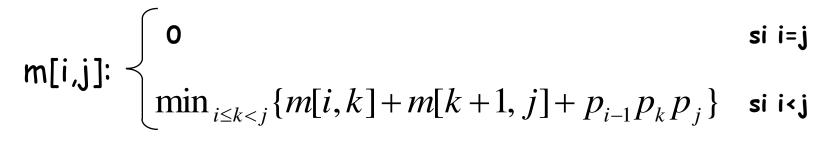
bottom-up También es necesario agregar una estructura que almacena la elección que se realizó en cada. caso el valor del k que se escogio

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

	1	2	3	4	5	6
1	0					
2	X	0				
3	X	X	0			
4	X	X	X	0		
5	X	X	X	X	0	
6	X	X	X	X	X	0

```
m[1,6]=min {
    m[1,1] + m[2,6] + p_0p_1p_6
    m[1,2] + m[3,6] + p_0p_2p_6
    m[1,3] + m[4,6] + p_0p_3p_6
    m[1,4] + m[5,6] + p_0p_4p_6
    m[1,5] + m[6,6] + p_0p_5p_6
```

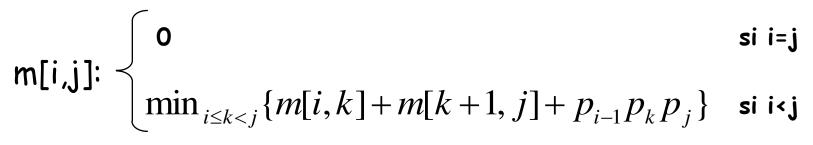
3. Calcular el valor de una solución óptima de manera bottom-up



	1	2	3	4	5	6
1	0					
2	X	0				
3	X	X	0			
4	X	X	X	0		
5	X	X	X	X	0	
6	X	X	X	X	X	0

```
m[1,6]=min \{ \\ m[1,1] + m[2,6] + p_0p_1p_6 \\ m[1,2] + m[3,6] + p_0p_2p_6 \\ m[1,3] + m[4,6] + p_0p_3p_6 \\ m[1,4] + m[5,6] + p_0p_4p_6 \\ m[1,5] + m[6,6] + p_0p_5p_6 \\ \}
```

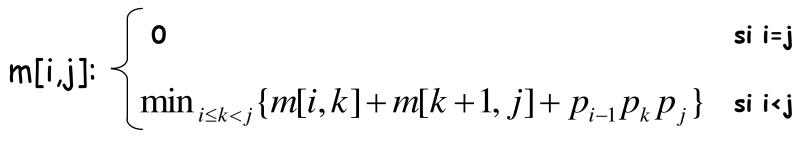
3. Calcular el valor de una solución óptima de manera bottom-up



	1	2	3	4	5	6
1	0					
2	X	0	_			
3	X	X	Ø	X		
4	X	X	X	0	X	
5	X	X	X	X	Ŕ	
6	X	X	X	X	X	0

¿De qué depende m[3,5]?

3. Calcular el valor de una solución óptima de manera bottom-up



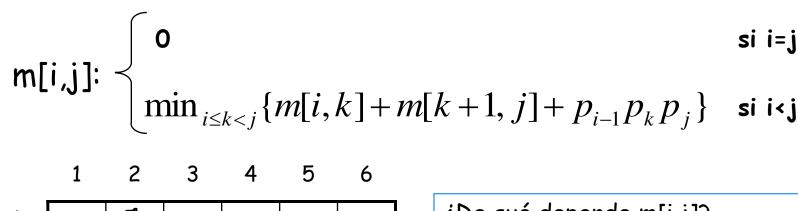
	1	2	3	4	5	6
1	0					
2	X	0				
3	X	X	0			
4	X	X	X	0		
5	X	X	X	X	0	
6	X	X	X	X	X	0

```
m[3,5]={

m[3,3] + m[4,5] + p_2p_3p_5

m[3,4] + m[5,5] + p_2p_3p_5
}
```

3. Calcular el valor de una solución óptima de manera bottom-up



1	2	3	4	5	6
G-	-p				
X	p -	Q	X		
X	X	þ	0		
X	X	X	Q		
X	X	X	X	0	
X	X	X	X	X	0

3

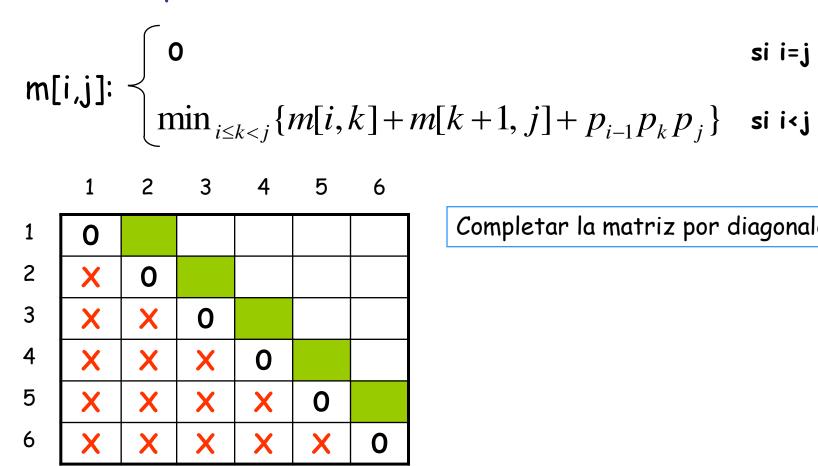
5

6

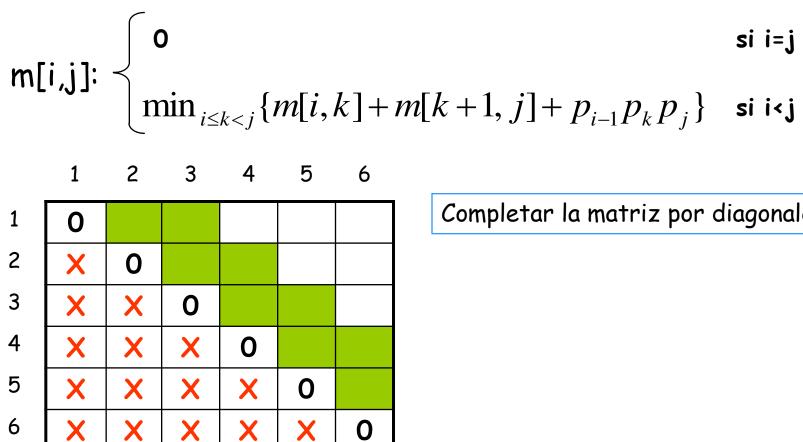
¿De qué depende m[i,j]?

¿Cómo se debería completar/llenar la matriz para que los cálculos necesarios ya se hayan realizado?

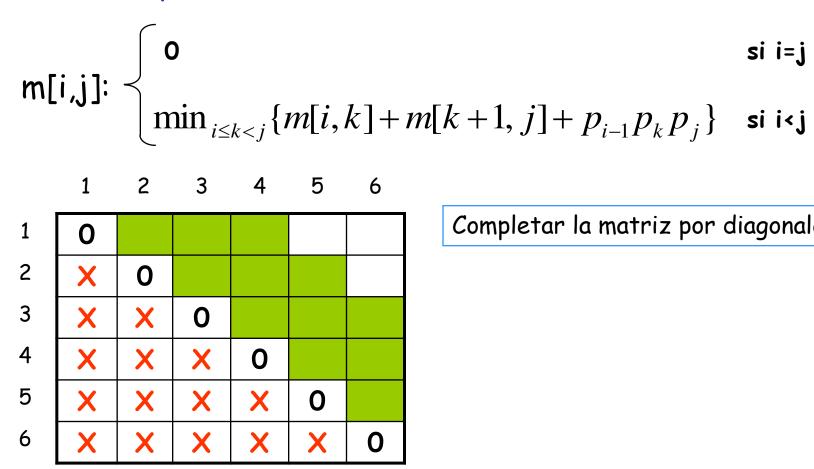
3. Calcular el valor de una solución óptima de manera bottom-up



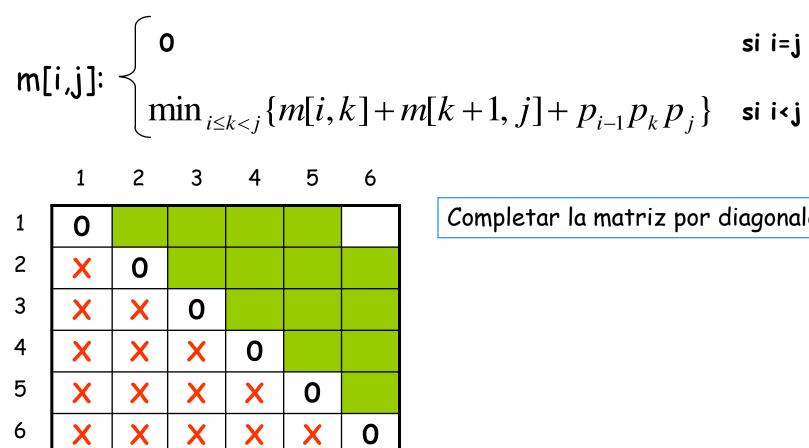
3. Calcular el valor de una solución óptima de manera bottom-up



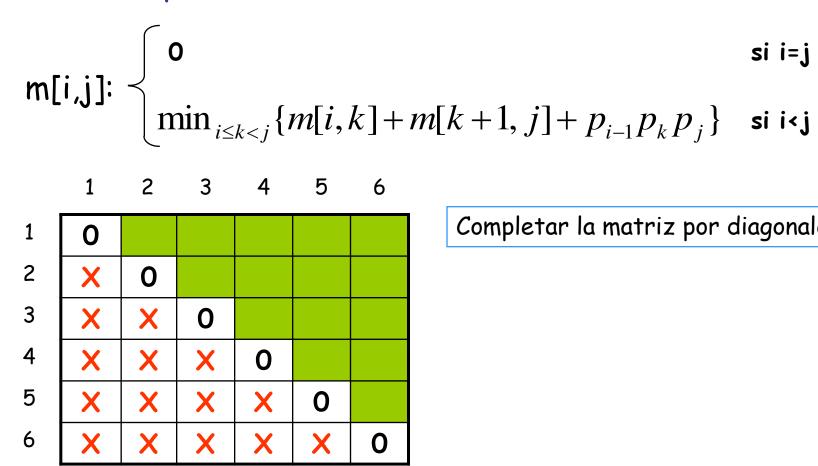
3. Calcular el valor de una solución óptima de manera bottom-up



3. Calcular el valor de una solución óptima de manera bottom-up



3. Calcular el valor de una solución óptima de manera bottom-up



3. Calcular el valor de una solución óptima de manera bottom-up

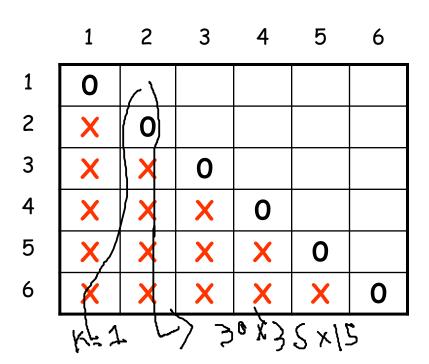
```
MATRIX-CHAIN-ORDER(p)
 n \leftarrow length[p]-1
 for i←1 to n
    do m[i,i]\leftarrow0
 for I←2 to n
    do for i←1 to n-l+1
           do j←i+l-1
             m[i,j] \leftarrow \infty
             for k←i to j-1
                do q \leftarrow m[i,k]+m[k+1,j]+p_{i-1}p_kp_i
                    if q<m[i,j]
```

then $m[i,j] \leftarrow q$

return m

3. Calcular el valor de una solución óptima de manera bottom-up

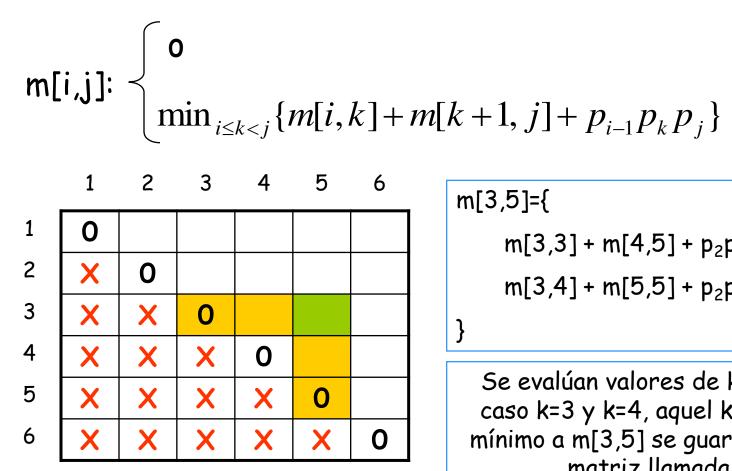
Ustedes deben almacenar 1) K (decisión) y 2) Costo $\mathbf{m[i,j]} : \begin{cases} \mathbf{0} & \text{si } \mathbf{i} = \mathbf{j} \\ \min_{i \leq k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\} & \text{si } \mathbf{i} < \mathbf{j} \end{cases}$



Completar m para las siguientes matrices: 12,							
A1 30×35							
A2	35×15						
A3	15×5						
A4	5×10						
A 5	16×20						
A 6	26×25						
	12 by 12k						

4. Construir una solución óptima a partir de la información calculada

Hasta ahora solo se tiene la cantidad óptima de multiplicaciones, falta mostrar la solución, esto es, el resultado de multiplicar las matrices en el orden óptimo



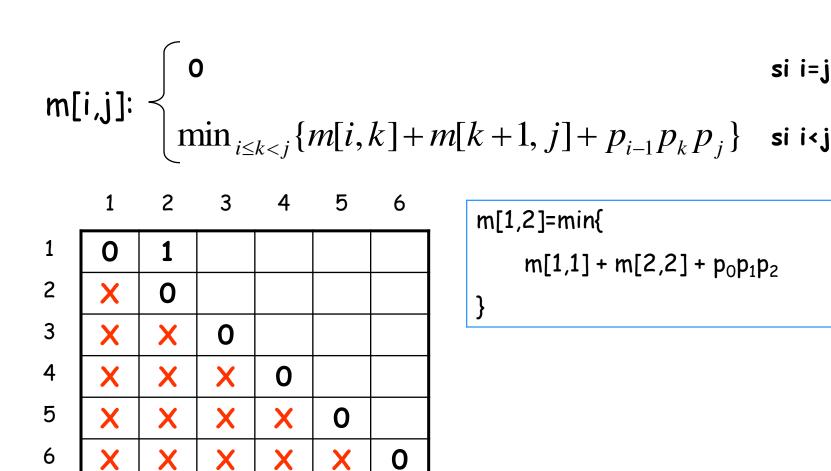
```
m[3,5]={
    m[3,3] + m[4,5] + p_2p_3p_5
    m[3,4] + m[5,5] + p_2p_3p_5
```

si i=j

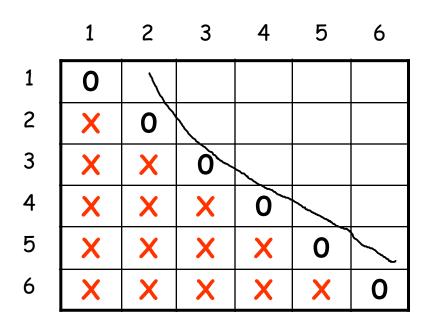
Se evalúan valores de k, en este caso k=3 y k=4, aquel k que haga mínimo a m[3,5] se guarda en otra matriz llamada s

```
MATRIX-CHAIN-ORDER(p)
 n \leftarrow length[p]-1
 for i←1 to n
    do m[i,i]\leftarrow0
 for I←2 to n
     do for i \leftarrow 1 to n-l+1
            do j←i+l-1
               m[i,j] \leftarrow \infty
               for k←i to j-1
                 do q \leftarrow m[i,k]+m[k+1,j]+p_{i-1}p_kp_i
                      if q<m[i,j]</pre>
                         then m[i,j] \leftarrow q
                                s[i,j]←k
```

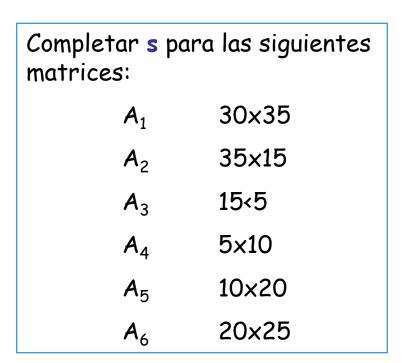
return m and s



Matriz s, almacena los valores de k



Matriz s, almacena los valores de k



4. Construir una solución óptima a partir de la información calculada

En s, cada celda s[i,j] almacena el valor k tal que la multiplicación de $A_i.A_{i+1}...A_j$ es óptima, esto es,

$$A_{i}...A_{k}A_{k+1}...A_{j}$$

4. Construir una solución óptima a partir de la información calculada

```
MATRIX-CHAIN-MULTIPLY(A,s,i,j)

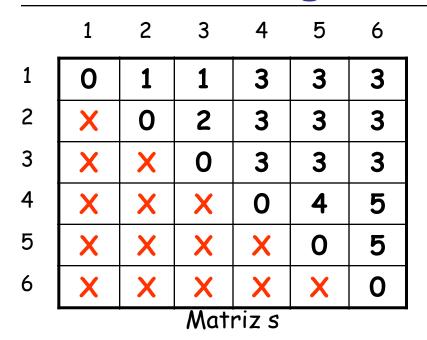
if i<j

then X \leftarrow MATRIX-CHAIN-MULTIPLY(A,s,i,s[i,j])

Y \leftarrow MATRIX-CHAIN-MULTIPLY(A,s,s[i,j]+1,j)

return MATRIX-MULTIPLY(X,Y)

else return A_i
```



```
      MATRIX-CHAIN-MULTIPLY(A,s,1,6)

      A_1
      30 \times 35

      A_2
      35 \times 15

      A_3
      15 \times 5

      A_4
      5 \times 10

      A_5
      10 \times 20

      A_6
      20 \times 25
```

```
MATRIX-CHAIN-MULTIPLY(A,s,i,j)

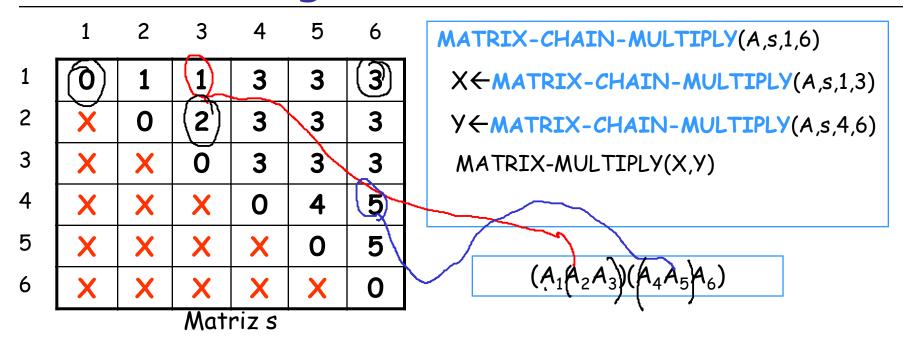
if i<j

then X←MATRIX-CHAIN-MULTIPLY(A,s,i,s[i,j])

Y←MATRIX-CHAIN-MULTIPLY(A,s,s[i,j]+1,j)

return MATRIX-MULTIPLY(X,Y)

else return A<sub>i</sub>
```



```
MATRIX-CHAIN-MULTIPLY(A,s,i,j)

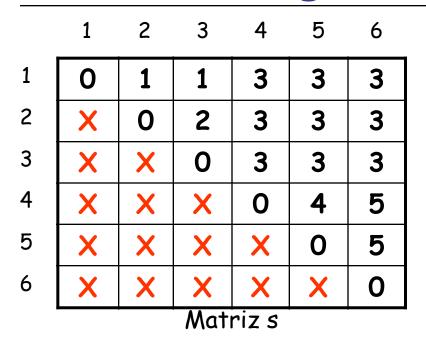
if i<j

then X←MATRIX-CHAIN-MULTIPLY(A,s,i,s[i,j])

Y←MATRIX-CHAIN-MULTIPLY(A,s,s[i,j]+1,j)

return MATRIX-MULTIPLY(X,Y)

else return A;
```



```
MATRIX-CHAIN-MULTIPLY(A,s,1,3)

X'←MATRIX-CHAIN-MULTIPLY(A,s,1,1)

Y'←MATRIX-CHAIN-MULTIPLY(A,s,2,3)

MATRIX-MULTIPLY(X',Y')
```

 $(A_1(A_2A_3))(A_4A_5A_6)$

```
MATRIX-CHAIN-MULTIPLY(A,s,i,j)

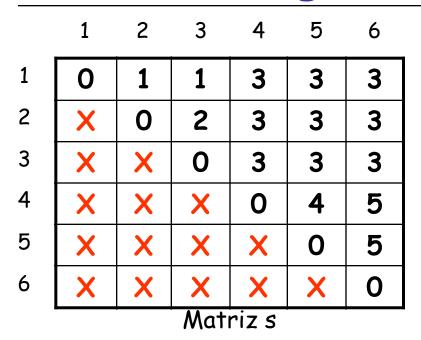
if i<j

then X←MATRIX-CHAIN-MULTIPLY(A,s,i,s[i,j])

Y←MATRIX-CHAIN-MULTIPLY(A,s,s[i,j]+1,j)

return MATRIX-MULTIPLY(X,Y)

else return A<sub>i</sub>
```



```
MATRIX-CHAIN-MULTIPLY(A,s,4,6)

X←MATRIX-CHAIN-MULTIPLY(A,s,4,5)

Y←MATRIX-CHAIN-MULTIPLY(A,s,6,6)

MATRIX-MULTIPLY(X,Y)
```

 $(A_1(A_2A_3))((A_4A_5)A_6)$

```
MATRIX-CHAIN-MULTIPLY(A,s,i,j)

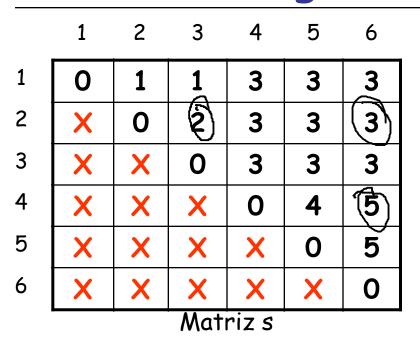
if i<j

then X←MATRIX-CHAIN-MULTIPLY(A,s,i,s[i,j])

Y←MATRIX-CHAIN-MULTIPLY(A,s,s[i,j]+1,j)

return MATRIX-MULTIPLY(X,Y)

else return A;
```



Encuentre la multiplicación óptima para $A_2A_3A_4A_5A_6$

 $\left(A_{2}A_{3}A_{4}A_{5}A_{6}\right)$

```
MATRIX-CHAIN-MULTIPLY(A,s,i,j)

if i<j

then X←MATRIX-CHAIN-MULTIPLY(A,s,i,s[i,j])

Y←MATRIX-CHAIN-MULTIPLY(A,s,s[i,j]+1,j)

return MATRIX-MULTIPLY(X,Y)

else return A;
```