Multiplicación Encadenada de Matrices

- Dadas n matrices $A_1, A_2, \ldots, A_n$ con A_i de dimensión d_{i-1} x d_i
- Determinar el orden de multiplicación para minimizar el numero de multiplicaciones escalares.
- Suponemos que la multiplicación de una matriz p x e por otra e x r requiere per multiplicaciones escalares

$$C[i, j] = \sum_{k=1}^{e} A[i,k] * B[k, j]$$

Multiplicación encadenada de matrices

- Un producto de matrices se dice que está completamente parentizado si está constituido por una sola matriz, o por el producto completamente parentizado de dos matrices, cerrado por paréntesis.

- La multiplicación de matrices es asociativa, y por tanto todas las parentizaciones producen el mismo resultado.

Multiplicación Encadenada de Matrices

- ¿Por qué es importante la parentización?

- Ejemplo
  B es 3 × 100
  C es 100 × 5
  D es 5 × 5

  (B*C)*D necesita 1500 + 75 = 1575 operaciones
  B*(C*D) necesita 1500 + 2500 = 4000 operaciones

Multiplicación Encadenada de Matrices

- El producto $A_1\ A_2\ A_3\ A_4$ puede parentizarse completamente de 5 formas distintas
  - $(A_1\ (A_2(A_3A_4)))$
  - $(A_1\ ((A_2A_3)A_4))$
  - $((A_1\ A_2)(A_3A_4))$
  - $((A_1\ (A_2A_3))A_4)$
  - $(((A_1\ A_2)A_3)A_4)$

A =	$\mathbf{A_1}$	$\mathbf{A_2}$	$\mathbf{A_3}$	$\mathbf{A_4}$
	10 x 20	**20 x 50**	**50 x 1**	**1 x 100**

Multiplicación Encadenada de Matrices

Orden 1 $A_1 \times (A_2 \times (A_3 \times A_4))$

Costo($A_3 \times A_4$) = 50 x 1 x 100

Costo($A_2 \times (A_3 \times A_4)$) = 20 x 50 x 100

Costo($A_1 \times (A_2 \times (A_3 \times A_4))$) = 10 x 20 x 100

Costo total = 125000 multiplicaciones

Orden 4 $(A_1 \times (A_2 \times A_3)) \times A_4$

Costo($A_2 \times A_3$) = 20 x 50 x 1

Costo($A_1 \times (A_2 \times A_3)$) = 10 x 20 x 1

Costo($(A_1 \times (A_2 \times A_3)) \times A_4$) = 10 x 1 x 100

Costo total = 2200 multiplicaciones

Multiplic. Encadenada de Matrices

- Se puede plantear mediante PD? Se cumple el POB?
- La secuencia de decisiones es dónde se coloca el primer paréntesis, dónde el segundo,...
  - La solución óptima se puede definir en términos de soluciones óptimas a problemas de tamaño menor.
  - Necesariamente tiene que haber una multiplicación final (la de mayor nivel) en el cálculo de la solución óptima.
  - Supongamos que la última multiplicación se realiza en la posición i:

$$(A_1 * \ldots * A_i) * (A_{i+1} * \ldots * A_n).$$

  - Si hubiera una solución mejor para algún subproblema, podríamos usarla en lugar de la anterior y obtendríamos una solución mejor que la óptima. Contradicción.

Recuento del número de parentizaciones

La enumeración de todas las parentizaciones posibles no proporciona un método eficiente.
Notemos el número de parentizaciones de una sucesión de n matrices por P(n).
Como podemos dividir una sucesión de n matrices en dos (las k primeras y las n-k siguientes) para cualquier k = 1,2,...,n-1, y entonces parentizar las dos subsucesiones resultantes independientemente, obtenemos la recurrencia:

$$P(n) = \begin{cases} 1 & \text{si } n = 1 \\ \sum_{k=1..n-1} P(k) \times P(n-k) & \text{si } n \geq 2 \end{cases}$$

Recuento del número de parentizaciones

La solución de esa ecuación es la sucesión de los Números de Catalan (que también cuenta el número de árboles binarios con n+1 hojas)

$$P(n) = C(n-1)$$

Donde

$$C(n) = (n+1)^{-1} C_{2n,n} = \frac{\binom{2n}{n}}{n+1}$$

es de orden exponencial, $4^n/n^{3/2}$,

Por tanto el método de la fuerza bruta es una pobre estrategia para determinar la parentización optimal de una cadena de matrices.

Multiplic. Encadenada de Matrices

- Sea $p_{k-1}p_k$ la dimensión de la matriz A_k
- Problema: Multiplicar $(A_1A_2...A_kA_{k+1}A_{k+2}\ldots A_n)$
- Supongamos que parentizamos en k
  – $(A_1A2...A_k)$ $(A_{k+1}A_{k+2}\ldots A_n)$

- Si llamamos N[i,j] al número de operaciones necesarias para multiplicar $A_i*A_{i+1}*\ldots*A_j$.

$$\mathbf{N[1,n] = N[1,k]+N[k+1,n]+p_0p_kp_n.}$$

Definición Recursiva Solución Optimal:

- Si i=j entonces

  $N[i,j] = 0$

- Para $1 <= i < j <= n$

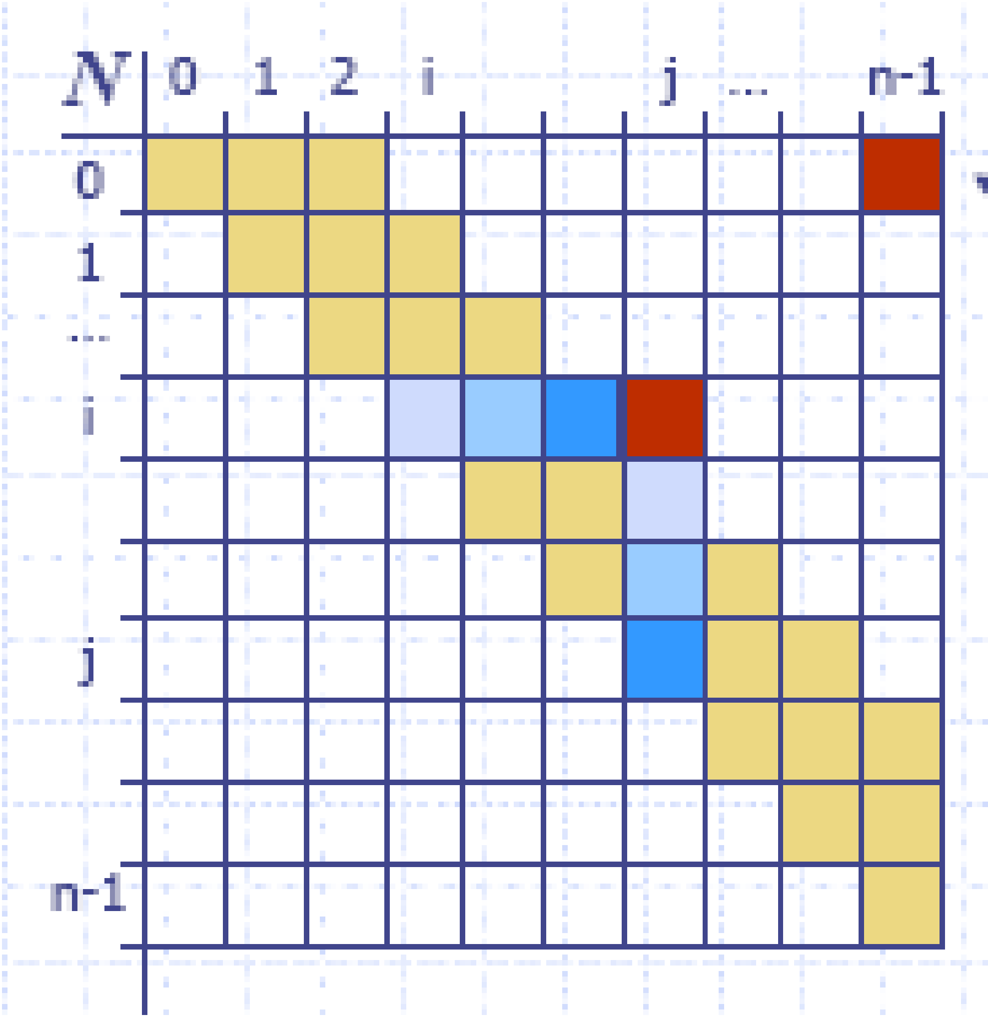
  $N[i,j]= \min\{i<=k<j\} (N[i,k]+N[k+1,j]+p_{i-1}p_kp_j)$

A partir de la definición recursiva se puede ver como se produce el solapamiento de los subproblemas

Primero se solucionan los suproblemas triviales (tamaño 0) para en cada paso ir resolviendo subproblemas de tamaño 1,2,3,…

Tenemos $O(n^2)$ subproblemas distintos

Para calcular N[i,j] necesitamos los valores almacenados en la fila i columna j



Multiplicación encadenada de matrices

- **Tablas usadas por el algoritmo.**
  - Sea **N** una matriz [1..n, 1..n] de enteros. El algoritmo usará la mitad de la matriz.

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

- **Forma de rellenar la tabla.**
  - Inicializar la matriz. Para todo i, desde 1 hasta n. N[i, i] = 0
  - Aplicar la ecuación de recurrencia por diagonales.

$$N[i, j] = \min_{i \leq k < j} (N[i, k] + N[k+1, j] + p_{i-1}p_k p_j)$$

- **Ejemplo**. n= 4, p = (10, 20, 50, 1, 100)

	j= 1	2	3	4
i=1	0	10.000	1.200	2.200
2		0	1.000	3.000
3			0	5.000
4				0

Multiplicación encadenada de matrices

- ¿Cuál es el orden de complejidad de este algoritmo?

  Para calcular cada valor necesitamos O(n)

  El algoritmo final es de orden O(n^3)

- En la posición **N[1, n]** tenemos almacenado el número mínimo de multiplicaciones escalares necesario (para la ordenación que es óptima). Necesitamos calcular cuál es esta ordenación óptima.
- Usar una matriz auxiliar **Mejork [1..n, 1..n]** en la que se almacene el índice donde se alcanzó el mínimo (mejor valor de **k)** para cada subproblema.
- En el **ejemplo anterior**.

Mejork	j= 1	2	3	4
i=1	-	1	1	3
2		-	2	3
3			-	3
4				-

Cálculo de la solución

Mejork	j= 1	2	3	4
i=1	-	1	1	3
2		-	2	3
3			-	3
4				-

Mejork[1,4]=3, luego los subproblemas son $A_{1..3}$ y $A_{4..4}$ o sea $(A_1A_2A_3)A_4$
Mejork[1,3]=1, luego tenemos $A_{1..1}$ y $A_{2..3}$ o sea $A_1(A_2A_3)$

Luego la parentización óptima es $((A_1(A_2A_3))A_4)$

Algoritmo

- MultCadenaMatrices(n)
  - for i= 1 to n

    N[i,i] = 0
  - for l = 2 to n

    for i = 1 to n-l+1

        j=i+l-1

        N[i,j] = inf.

        for k=i to j-1

            q=N[i,k] + N[k+1,j] + p[i-1]p[k]p[j]

            if q < N[i,j]

                N[i,j] = q

                Mejork[i,j] = k

- MultplCadMatrices(A, s, i, j)

```
if j>i
    x = MultplCadMatrices (A, s, i, s[i,j])
    y = MultplCadMatrices (A, s, s[i,j]+1, j)
    return MultMatrices(x, y)
else return Ai
```

s representa a Mejork y MultMatrices es la multiplicación estándar de 2 matrices.

Subsecuencia Común de Mayor Longitud (LCS)

Dadas dos secuencias de símbolos X e Y, ¿cuál es la subsecuencia común a X e Y de longitud mayor?

Ej: X= {A B C B D A B }, Y= {B D C A B A}

Subsec. Común de Mayor longitud :

X = A **B** **C** **B** D **A** B

Y = **B** D **C** A **B** **A**

También puede ser BDAB

Subsecuencia Común de Mayor Longitud (LCS)

Aplicaciones en bioinformática (genómica) y en comparación de ficheros (diff).

Un algoritmo de fuerza bruta compararía cualquier subsecuencia de X con los símbolos de Y.

Si $|X| = m$, $|Y| = n$, hay que contrastar 2^m subsecuencias de X contra los n elementos de Y.

Eso daría un algoritmo de orden $O(n2^m)$

LCS: POB?

- Sin embargo, LCS tiene *subestructuras optimales*: las soluciones a los subproblemas son parte de la solución final.

  X = A **B** **C** **B** D **A** B

  Y = **B** D **C** A **B** **A**

  Si BCBA es solución optimal => BCB debe ser solución optimal para qué subproblemas

  Para X' = ABCBD (quitamos AB a X por la dcha.) y

  Y' = BDCAB (quitamos A a Y por la dcha.)

  Si BCB no es optimal para X' e Y' entonces BCBA no puede ser optimal para X e Y

- Subproblemas: "Encontrar LCS para pares de prefijos de X e Y"

LCS: subproblemas

- Definimos X_i, Y_j los prefijos de X e Y de longitud i y j respectivamente

$$X = \langle x_1, x_2, \ldots , x_m \rangle, Y = \langle y_1, y_2, \ldots , y_n \rangle$$

$$X_i = \langle x_1, x_2, \ldots , x_i \rangle, Y_j = \langle y_1, y_2, \ldots , y_j \rangle$$

- Definimos $c[i,j]$ la longitud de LCS para X_i e Y_j
- Entonces, LCS de X e Y será $c[m,n]$

LCS

Supongamos que la sub-secuencia común de mayor longitud LCS de X e Y es

$$Z_k = <z_1, z_2, \ldots, z_k>$$

Si $x_m = y_n$ entonces $z_k = x_m = y_n$ y Z_{k-1} es una LCS de X_{m-1} e Y_{n-1}

En otro caso o bien Z_k es una LCS de X_{m-1} e Y_n o una LCS de X_m e Y_{n-1} (si $z_k = x_m$ entonces y_n no puede estar en LCS, de ahí que Z_k sea LCS de X_m e Y_{n-1}; si $z_k \,!= x_m$, de ahí que Z_k sea LCS de X_{m-1} e Y_n)

$$c[i,j] = \begin{cases} c[i-1, j-1] + 1 & \text{si } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{en otro caso} \end{cases}$$

LCS: Definición Recursiva

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{si } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & encasocontrario \end{cases}$$

- *Inicio: i = j = 0* (subcadena vacía de x e y)

  (*c[0,0] = 0*)

- LCS de la cadena vacía y cualquier otra cadena es vacía, por tanto para cada par i,j:

  c[0, j] = c[i,0] = 0

Algoritmo LCS

LCS-Length(X, Y)

1. m = length(X) // get the # of symbols in X
2. n = length(Y) // get the # of symbols in Y
3. for i = 1 to m c[i,0] = 0 // special case: Y_0
4. for j = 1 to n c[0,j] = 0 // special case: X_0
5. for i = 1 to m // for all X_i
6. for j = 1 to n // for all Y_j
7. if (x[i] == y[j])
8. c[i,j] = c[i-1,j-1] + 1
9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c

Eficiencia O(nm)

Ejemplo LCS

- X = ABCB
- Y = BDCAB

LCS(X, Y) = BCB

X = A **B** **C** **B**

Y = **B** D **C** A **B**

Ejemplo LCS (0)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	**B**	**D**	**C**	**A**	**B**
0	Xi							
1	**A**							
2	**B**							
3	**C**							
4	**B**							

X = ABCB; m = |X| = 4
Y = BDCAB; n = |Y| = 5

Ejemplo LCS (1)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**					
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

for i = 1 to m   c[i,0] = 0
for j = 1 to n   c[0,j] = 0

Ejemplo LCS (2)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**				
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Ejemplo LCS (3)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**		
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

if (X_i == Y_j)

c[i,j] = c[i-1,j-1] + 1

else c[i,j] = max(c[i-1,j], c[i,j-1])

Ejemplo LCS (4)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

if (X_i == Y_j)
 c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

Ejemplo LCS (5)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1** →	**1**
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Ejemplo LCS (6)

ABCB
BDCAB

i		j: 0	1	2	3	4	5
		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**				
3	**C**	**0**					
4	**B**	**0**					

if (X_i == Y_j)
c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

Ejemplo LCS (7)

ABCB
BDCAB

i		j: 0	1	2	3	4	5
		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	
3	**C**	**0**					
4	**B**	**0**					

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Ejemplo LCS (8)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**					
4	**B**	**0**					

if (X_i == Y_j)
c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

Ejemplo LCS (10)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**			
4	**B**	**0**					

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Ejemplo LCS (11)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**		
4	**B**	**0**					

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Ejemplo LCS (12)

ABCB
BDCAB

j		0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**					

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Ejemplo LCS (13)

ABCB
BDCAB

j		0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**	**1**				

if (X_i == Y_j)
c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

Ejemplo LCS (14)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**	**1**	**1**	**2**	**2**	

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Ejemplo LCS (15)

ABCB
BDCAB

i	j	0	1	2	3	4	5
		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**	**1**	**1**	**2**	**2**	**3**

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Cómo encontrar la subsecuencia LCS

Cada $c[i,j]$ depende de $c[i-1,j]$ *y* $c[i,j-1]$

O bien de $c[i-1, j-1]$

Por tanto, a partir del valor $c[i,j]$ podremos averiguar cómo se determinó

2	2
2	3

Por ejemplo

$c[i,j] = c[i-1,j-1] +1 = 2+1=3$

Cómo encontrar la subsecuencia LCS

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{if } x[i]=y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

- Empezamos desde *c[m,n]* y vamos hacia atrás
- Si *c[i,j] = c[i-1, j-1]+1* y *x[i]=y[j]*, guardamos *x[i]* (porque *x[i]* pertenece a LCS): i=i-1; j=j-1
- Si *c[i,j] = c[i, j-1]* : j=j-1
- Si *c[i,j] = c[i-1, j]* : i=i-1
- Si i=0 ó j=0 (alcanzamos el principio), devolvemos los caracteres almacenados en orden inverso.

Encontramos LCS

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**	**1**	**1**	**2**	**2**	**3**

Encontramos LCS

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1** ←	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2** ←	**2**	**2**
4	**B**	**0**	**1**	**1**	**2**	**2**	**3**

LCS : **B C B**

Caminos mínimos: Algoritmo de Floyd

Problema:
Calcular el camino más corto que une cada par de vértices de un grafo, considerando que no hay pesos negativos.

Posibles soluciones:

- Por fuerza bruta (de orden exponencial).
- Aplicar el algoritmo de Dijkstra para cada vértice.
- Algoritmo de Floyd (programación dinámica).

Caminos mínimos: POB?

- Camino mínimo entre dos vértices i y j
- Se puede plantear mediante PD? Se cumple el POB?
- La secuencia de decisiones es cuál es el primer vértice del camino, cuál el segundo,...
- Si el camino mínimo de i a j pasa por k, entonces los caminos de i a k y de k a j son también mínimos (si no lo fueran encontraríamos un camino de i a j mejor que el mínimo)
- Luego de cumple el principio de optimalidad de Bellman

Definición recursiva

D_k(i,j): valor del camino más corto de i a j usando sólo los k primeros vértices del grafo, {1,2,...,k} como nodos intermedios.

Expresión recursiva:

$$D_k(i,j) = \min\{D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j)\}$$

Caso base:

$$D_0(i,j) = c_{ij}$$

Matriz de adyacencia, con c_{ij}=infinito si no hay arco de i a j

La solución del problema original el D_n(i,j)

Algoritmo de Floyd (1962)

```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        D[i][j] = coste(i,j);

for (k=1; k<=n; k++)
    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
            if (D[i][k] + D[k][j] < D[i][j] )
               D[i][j] = D[i][k] + D[k][j];
```

En la k-esima iteración los valores de la fila k y la columna k no cambian pues D[k,k]=0. Por eso se puede utilizar una única matriz D en vez de una para D_k y otra para D_{k-1}

Orden de eficiencia $O(n^3)$

$$D_k(i,j) = \min\{D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j)\}$$

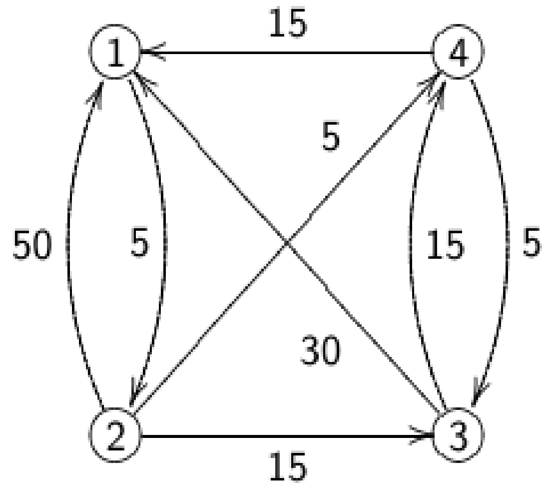
Floyd: cálculo de la solución

- Si además de conocer el valor del camino mínimo queremos conocer el camino en sí, empleamos otra matriz P, y el bucle interno del algoritmo sería:

```
if (D[i][k] + D[k][j] < D[i][j] )
                D[i][j] = D[i][k] + D[k][j];
                P[i][j] = k;
```

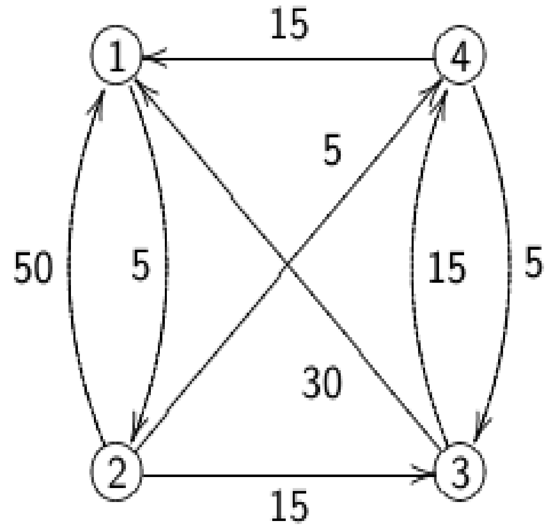
- Si al terminar, P[i][j]=0 el camino es el directo de i a j
- Si P[i][j]=k, entonces el camino pasa por k. Se analizan P[i][k] y P[k][j]

Floyd: ejemplo



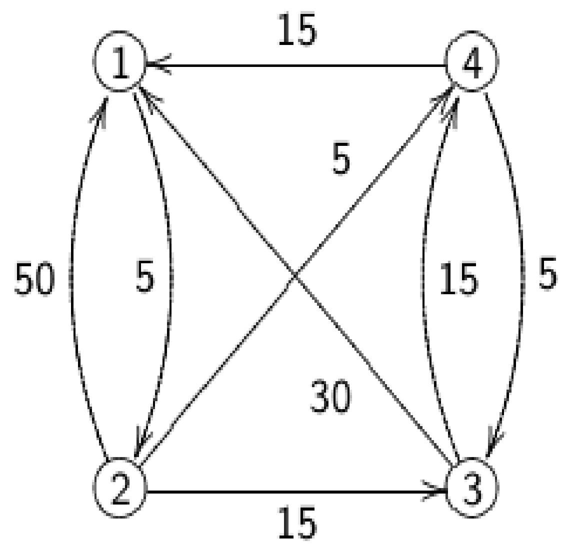
$$D_0 = D = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix}$$

Floyd: ejemplo, k=1



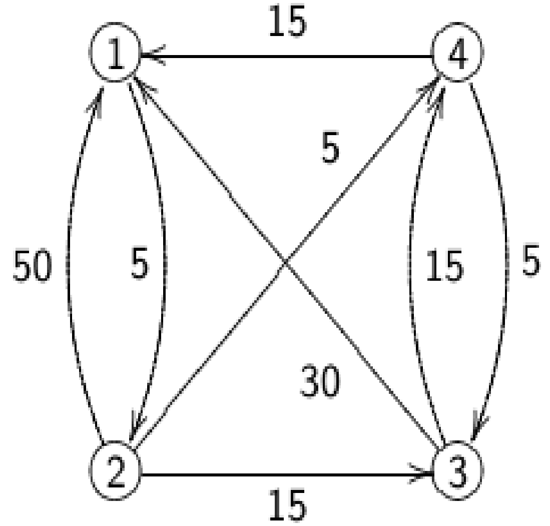
$$D_0 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix} \qquad D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Floyd: ejemplo, k=2



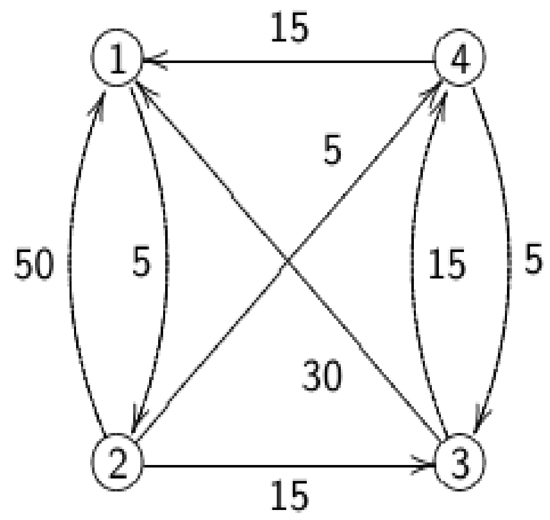
$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \qquad D_2 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Floyd: ejemplo, k=3



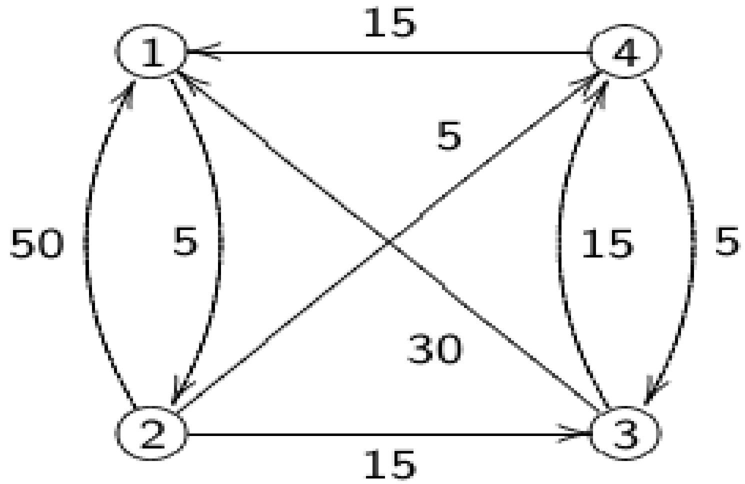
$$D_2 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \qquad D_3 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 45 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Floyd: ejemplo, k=4



$$D_3 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 45 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \qquad D_4 = \begin{pmatrix} 0 & 5 & 15 & 10 \\ 20 & 0 & 10 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

Floyd: cálculo de la solución



$$P = \begin{pmatrix} 0 & 0 & 4 & 2 \\ 4 & 0 & 4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$