Análisis y Diseño de Algoritmos II

Jesús Alexander Aranda Ph.D Robinson Duque, Ph.D
Juan Francisco Díaz, Ph. D

Universidad del Valle

jesus.aranda@correounivalle.edu.co
robinson.duque@correounivalle.edu.co
juanfco.diaz@correounivalle.edu.co
Programa de Ingeniería de Sistemas
Escuela de Ingeniería de Sistemas y Computación

Programación Dinámica

Caso de Estudio

Multiplicación de Sucesión de Matrices

Programación dinámica

Los problemas de optimización donde se aplica programación dinámica se caracterizan por:

- **Subestructura óptima**: la solución (óptima) de un problema se descompone en soluciones (óptimas) de algunos de sus subproblemas.
- **Solapamiento de subproblemas**: Hay subproblemas diferentes que comparten subsubproblemas.

Programación dinámica

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?

Programación dinámica

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:

$A_1.(A_2.A_3)$

$(A_1.A_2)A_3$

Programación dinámica

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

```
MATRIX-MULTIPLY(A,B)
 if columns[A]≠rows[B]
    then error "incompatible dimensions"
    else for i←1 to rows[A]
            do for j←1 to columns[B]
                do C[i,j]←0
                   for k←1 to columns[A]
                     do C[i,j]←C[i,j] + A[i,k].B[k,j]
```

Programación dinámica

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]≠rows[B]
    then error "incompatible dimensions"
    else for i←1 to rows[A]
            do for j←1 to columns[B]
                do C[i,j]←0
                   for k←1 to columns[A]
                     do C[i,j]←C[i,j] + A[i,k].B[k,j]
```

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

Programación dinámica

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:

$A_1.(A_2.A_3)$

$(A_1.A_2)A_3$

Programación dinámica

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

Se tienen dos formas:

- $A_1.(A_2.A_3)$

La cantidad de multiplicaciones de $A_2.A_3$ es 100x5x50=25000, lo cual genera una matriz de 100x50.

Por lo que $A_1.(A_2.A_3)$ lleva 10x100x50 + 25000=75000 multiplicaciones

- $(A_1.A_2)A_3$

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

Programación dinámica

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

Se tienen dos formas:

- $A_1.(A_2.A_3)$

La cantidad de multiplicaciones de $A_2.A_3$ es 100x5x50=25000, lo cual genera una matriz de 100x50.

Por lo que $A_1.(A_2.A_3)$ lleva 10x100x50 + 25000=**75000** multiplicaciones

- $(A_1.A_2)A_3$

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

Programación dinámica

Problema: A partir de una sucesión $\langle A_1, \ldots, A_n \rangle$ de n matrices, donde A_i tiene dimensiones $p_{i-1} x p_i$ determinar la manera de multiplicarlas tal que se minimice el número de multiplicaciones escalares.

Entrada: $\langle p_0, p_1, p_2, \ldots, p_n \rangle$ que indican las dimensiones de todas las matrices

(Ejemplo, $\dim(A_1) = p_0 x p_1$, $\dim(A_2) = p_1 x p_2$ ...$\dim(A_n) = p_{n-1} x p_n$)

Salida: la manera óptima de multiplicar las matrices

Programación dinámica

Problema:

Entrada: $\langle p_0, p_1, p_2, \ldots, p_n \rangle$ que indican las dimensiones de todas las matrices $\langle A_1, \ldots, A_n \rangle$

(Ejemplo, $\dim(A_1) = p_0 \times p_1$, $\dim(A_2) = p_1 \times p_2$ ...$\dim(A_n) = p_{n-1} \times p_n$)

Salida: la manera óptima de multiplicar las matrices

Instancias	Solución	Valor de la Solución
<10, 100, 5, 50>	$(A_1.A_2)A_3$	7500
<20, 50, 30>	$A_1.A_2$	30000

Programación dinámica

Sucesión de Multiplicación de Matrices (Fuerza Bruta)

Entrada: $\langle p_0, p_1, p_2, \ldots, p_n \rangle$ que indican las dimensiones de todas las matrices $\langle A_1, \ldots, A_n \rangle$

(Ejemplo, $\dim(A_1) = p_0 x p_1$, $\dim(A_2) = p_1 x p_2$ ...$\dim(A_n) = p_{n-1} x p_n$)

Salida: la manera óptima de multiplicar las matrices

Solución Inicial:

- 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 total de maneras de multiplicar es exponencial sobre n, la cantidad de matrices a multiplicar

Programación dinámica

Sucesión de Multiplicación de Matrices (Programación Dinámica)

Problema:

Entrada: $\langle p_0, p_1, p_2, \ldots, p_n \rangle$ que indican las dimensiones de todas las matrices $\langle A_1, \ldots, A_n \rangle$

(Ejemplo, dim(A_1) = $p_0 x p_1$, dim(A_2) = $p_1 x p_2$ …dim (A_n) = $p_{n-1} x p_n$)

Salida: la manera óptima de multiplicar las matrices

Al aplicar programación dinámica sobre un problema de optimización se siguen los siguientes pasos:

1) Caracterizar la estructura de la solución óptima (subestructura óptima).
2) Definir recursivamente el valor de la solución óptima.
3) Calcular el valor de la solución óptima (algoritmo).
4) Construir la solución óptima (algoritmo).

Programación dinámica

1. Caracterización de la estructura de una solución óptima (subestructura óptima)

Sea $A_{i:j}$ la matriz que resulta de evaluar $A_i \ldots A_j$

- **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 $A_{k+1:n}$**

Programación dinámica

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

Programación dinámica

2. Definir recursivamente el valor de una solución óptima

m[i,j]: mínimo número de multiplicaciones necesarias para calcular Ai:j (es decir, el valor de la solución del problema de matrices desde Ai hasta Aj)

- Se mantiene una matriz para ir almacenando los resultados óptimos de los subproblemas
- El valor de la solución del problema original corresponde a m[1,n]

Programación dinámica

2. Definir recursivamente el valor de una solución óptima

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

Programación dinámica

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

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

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

Programación dinámica

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

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

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

Programación dinámica

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

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

- Solo se define m cuando i<j
- Las celdas con X no se calcularán

	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

Programación dinámica

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

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

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

Programación dinámica

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

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

	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] + $p_0p_5p_6$

}

Programación dinámica

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

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

	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] + $p_0p_5p_6$

}

Programación dinámica

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

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

	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

¿De qué depende m[3,5]?

Programación dinámica

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

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

	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] + $p_2p_3p_5$

}

Programación dinámica

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

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

	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

¿De qué depende m[i,j]?

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

Programación dinámica

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

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

	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

Completar la matriz por diagonales

Programación dinámica

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

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

	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

Completar la matriz por diagonales

Programación dinámica

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

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

	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

Completar la matriz por diagonales

Programación dinámica

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

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

	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

Completar la matriz por diagonales

Programación dinámica

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

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

	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

Completar la matriz por diagonales

Programación dinámica

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

```
MATRIX-CHAIN-ORDER(p)
 n←length[p]-1
 for i←1 to n
    do m[i,i]←0
 for l←2 to n
     do for i←1 to n-l+1
            do j←i+l-1
               m[i,j]←∞
               for k←i to j-1
                  do q←m[i,k]+m[k+1,j]+p_{i-1}p_k p_j
                      if q<m[i,j]
                         then m[i,j]←q
return m
```

Programación dinámica

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

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

	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

Completar m para las siguientes matrices:

A1	30x35
A2	35x15
A3	15x5
A4	5x10
A5	10x20
A6	20x25

Programación dinámica

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

Programación dinámica

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

	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] + $p_2p_3p_5$

}

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

Programación dinámica

```
MATRIX-CHAIN-ORDER(p)
  n←length[p]-1
  for i←1 to n
     do m[i,i]←0
  for l←2 to n
     do for i←1 to n-l+1
           do j←i+l-1
              m[i,j]←∞
              for k←i to j-1
                 do q←m[i,k]+m[k+1,j]+p_{i-1}p_k p_j
                    if q<m[i,j]
                       then m[i,j]←q
                            s[i,j]←k
return m and s
```

Costo Computacional Temporal

Calcular cada posición de la matriz cuesta $O(n)$

Calcular toda la matriz $O(n^3)$

Costo Computacional Espacial

$\theta\ (n^2)$

Programación dinámica

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

	1	2	3	4	5	6
1	0	1				
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

Matriz s, almacena los valores de k

m[1,2]=min{

m[1,1] + m[2,2] + $p_0p_1p_2$

}

Programación dinámica

	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

Matriz s, almacena los valores de k

Completar **s** para las siguientes matrices:

A_1	30x35
A_2	35x15
A_3	15<5
A_4	5x10
A_5	10x20
A_6	20x25

Programación dinámica

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,

$A_i...A_kA_{k+1}...A_j$

Programación dinámica

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

```
PRINT-OPTIMAL-PARENS(s,i,j)
if i=j
   then print "A"i
   else  print "("
         PRINT-OPTIMAL-PARENS(s,i,s[i,j])
         PRINT-OPTIMAL-PARENS(s,s[i,j]+1, j)
          print ")"
```

Programación dinámica

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

Matriz s

PRINT-OPTIMAL-PARENS (s,1,6)

A_1	30x35
A_2	35x15
A_3	15x5
A_4	5x10
A_5	10x20
A_6	20x25

```
PRINT-OPTIMAL-PARENS(s,i,j)
if i=j
   then print "A"i
   else  print "("
        PRINT-OPTIMAL-PARENS(s,i,s[i,j])
        PRINT-OPTIMAL-PARENS(s,s[i,j]+1, j)
         print ")"
```

Programación dinámica

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

Matriz s

```
PRINT-OPTIMAL-PARENS(s,1,6)
  "(" PRINT-OPTIMAL-PARENS(s,1,3)
   PRINT-OPTIMAL-PARENS(s,4,6) ")"
```

$((A_1A_2A_3)(A_4A_5A_6))$

```
PRINT-OPTIMAL-PARENS(s,i,j)
if i=j
   then print "A"i
   else  print "("
        PRINT-OPTIMAL-PARENS(s,i,s[i,j])
        PRINT-OPTIMAL-PARENS(s,s[i,j]+1, j)
         print ")"
```