Fundamentos de análisis y diseño de algoritmos

Programación dinámica

LCS (Longest Common Subsequence)

Multiplicación de matrices

Propiedades generales de la programación dinamica

Programación dinámica

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.

Programación dinámica

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

Programación dinámica

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

Programación dinámica

Multiplicación de sucesión de matrices (MSM)

$AB \neq BA$

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?

$A_1 A_2 A_3$

$(A_1 A_2) A_3$ $\qquad$ $A_1 (A_2 A_3)$

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, 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]
```

Programación dinámica

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

$p \times q \quad q \times r$

$pqr = O(n^3)$

```
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 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.

p x q   q x r

Se tienen dos formas:

10x100 $A_1.(A_2.A_3)$ → 25000 + 50000 = 75000

100x50

$(A_1.A_2)A_3$ = 5000 + 2500 = 7500

10x5   5x50

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=2500, lo cual genera una matriz de 100x50.

Por lo que $A_1.(A_2.A_3)$ lleva 10x100x50 + 2500=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=2500, lo cual genera una matriz de 100x50.

Por lo que $A_1.(A_2.A_3)$ lleva 10x100x50 + 2500=**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

¿Cual es la complejidad de la solución ingenua?

$M = A_1 A_2 \dots A_n$ $\quad (A_1 A_2)(A_3 A_4) \dots$

$(A_1 A_2) \dots A_n$

$A_1 \quad A_2 \quad A_3$

$\left\{ \begin{array}{l} A_1 A_2 \quad\quad A_2 A_3 \\ A_3 A_4 \dots \end{array} \right.$

Tama 2

$\underbrace{(A_1 A_2)}_{x} \underbrace{(A_3 A_4)}_{y}$

Tamaño 4

$A_1 A_2 A_3 A_4$

$(A_1 A_2)(A_3 A_4)$ ✓

$A_1 (\underbrace{A_2 A_3}^{A_X}) A_4$ $\qquad A_1 A_X A_4$

$A_1 A_2 A_3 A_4 A_5$

$(A_1 A_2)(A_3 A_4) A_5$

$A_X A_4 A_5$ $\qquad A_1 (A_2 A_3)(A_4 A_5)$

$(A_1 A_2) A_3 (A_4 A_5)$

$(A_1 A_2)(A_3 A_4)(A_5 A_6) \sim$ caso 3

$A_1 (A_2 A_3)(A_4 A_5) A \sim$ caso 4

$(A_1 A_2) A_3 (A_4 A_5) A_6$

$\sim$ caso 4

$T(n) = T(n-3) + 2T(n-2) + 2 \qquad T(3) = 2$

$$2$$

$$T(n-3) \qquad T(n-2) \qquad T(n-2)$$

Solución es exponencial r^n

Programación dinámica

Problema: Multiplicación de una sucesión de matrices

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

Salida: la manera óptima de multiplicar las matrices

Programación dinámica

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

Programación dinámica

1. Caracterizar la estructura de una solución 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 $A_{k+1..n}$

El costo de una solución óptima es la suma de:

- 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 $A_{k+1...n}$

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 $A_{i..j}$

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

Programación dinámica

2. Definir recursivamente el valor de una solución óptima

A_i A_n

p_1 p_2

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

$A_i, A_{i+1}, \ldots A_j$

min $((A_i\, A_{i+1})(\cdots)\quad (A_i\, A_{i+1}\, A_{i+2}) \cdots (A_i\, A_{i+1}\, A_{i+2}\, A_{i+3}) \cdots)$

$(A_i \ldots)A_j$

Programación dinámica

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

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

A_i A_j

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

$(A_1 A_2 A_3)$

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

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

	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

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

Programación dinámica

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

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

$1 \le k < 2$

$A_1 \ldots A_6$

$1 \le k < 6$

$(A_2 \ldots A_4)$

$(A_4 \ldots A_5)$

	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 {

$m[1,1] + m[2,6] + p_0 p_1 p_6$

$m[1,2] + m[3,6] + p_0 p_2 p_6$

$m[1,3] + m[4,6] + p_0 p_3 p_6$

$m[1,4] + m[5,6] + p_0 p_4 p_6$

$m[1,5] + m[6,6] + p_0 p_5 p_6$

}

Programación dinámica

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

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

	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

P_{i-1}, P_i

P_0 P_1 P_2 P_3 P_4 P_5 P_6

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]={ K

i j

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

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

}

$A_3 A_4 A_5$

$A_3 (A_4 A_5)$

$(A_3 A_4) A_5$

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
```

Programación dinámica

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

	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{

$m[1,1] + m[2,2] + p_0 p_1 p_2$

}

$0 + 0 + 30 \times 35 \times 15$

$(A_1 A_2)$

Programación dinámica

15750

	1	2	3	4	5	6
1	0	1				
2	X	0	2			
3	X	X	0	3		
4	X	X	X	0	4	
5	X	X	X	X	0	5
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	15x5
A_4	5x10
A_5	10x20
A_6	20x25

$$M[1,2] = \min \begin{cases} M[1,1] + M[2,2] + p_{i-1}\, p_k\, p_j \\ 0 + 0 + 30 \times 35 \times 15 \quad p_0\, p_1\, p_2 \end{cases}$$

1

15750

$(A_1)(A_2)$

$$(A_i \cdots A_k)(A_{k+1} \cdots A_j)$$

$A_{i,k}$

$A_2 A_3$

$i \leq k$

$$m[1,3] = \begin{cases} m[1,1] + m[2,3] + p_0 p_1 p_3 & 10000 + 3000 + 7500 = 20500 \\ m[1,2] + m[3,3] + p_0 p_2 p_3 & 10000 + 20000 + 1000 = 31000 \end{cases}$$

1, 2

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

```
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_i
```

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

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

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

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_i

$(A_1 A_2 A_3)(A_4 A_5 A_6)$

1,3 4,6

$(A_1 (A_2 A_3))((A_4 A_5) A_6)$

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

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

X←MATRIX-CHAIN-MULTIPLY(A,s,1,3)

Y←MATRIX-CHAIN-MULTIPLY(A,s,4,6)

MATRIX-MULTIPLY(X,Y)

$(\underline{A_1A_2A_3})(\underline{A_4A_5A_6})$

```
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_i
```

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

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_i

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

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_i

Programación dinámica

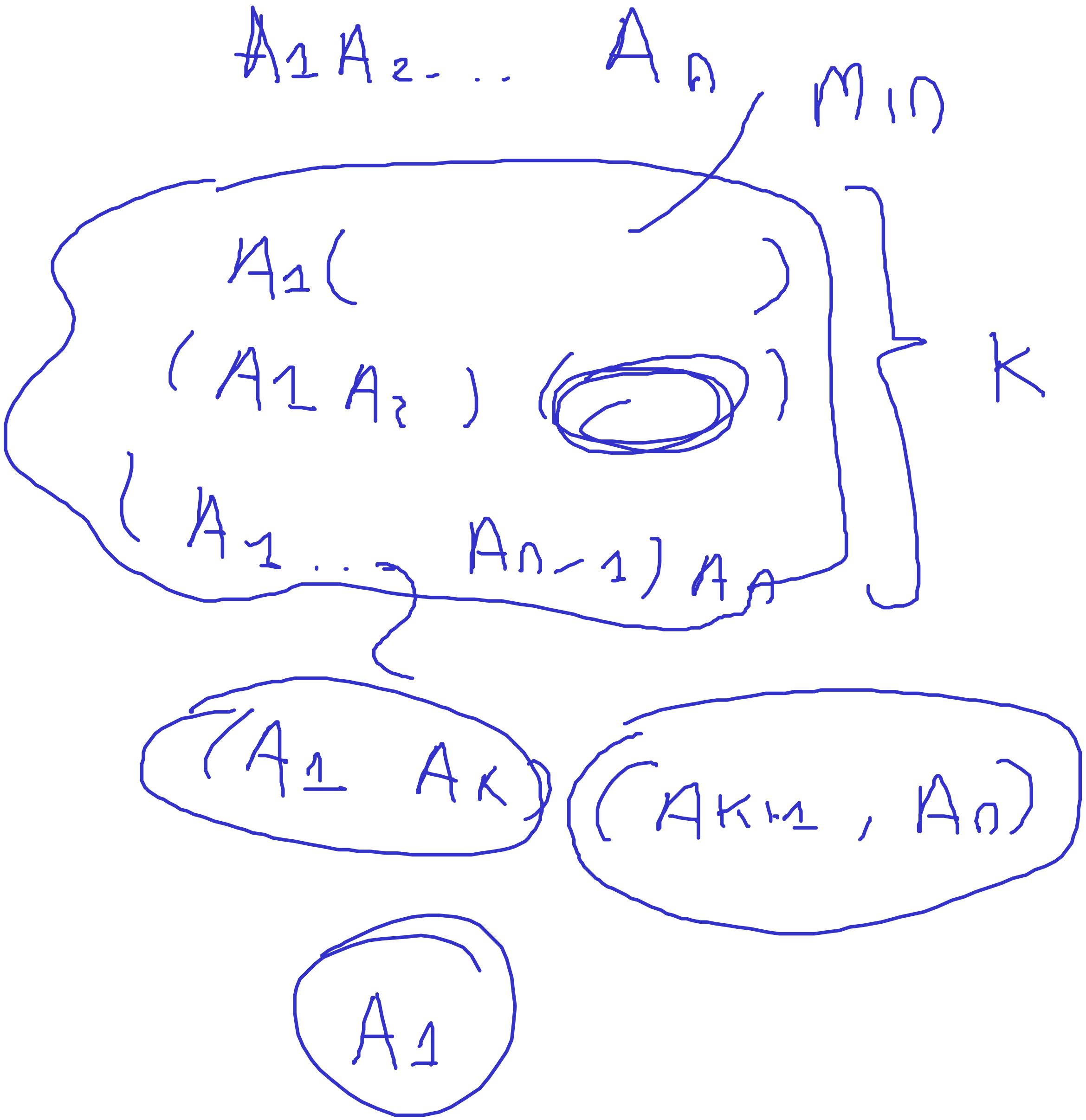
	1	2	3	4	5	6
1	**0**	**1**	**1**	**3**	**3**	**3**
2	X	**0**	**2**	**3**	**3**	**3**
3	X	X	**0**	**3**	**3**	**3**
4	X	X	X	**0**	**4**	**5**
5	X	X	X	X	**0**	**5**
6	X	X	X	X	X	**0**

Matriz s

Encuentre la multiplicación óptima para $A_2A_3A_4A_5A_6$

$A_2A_3A_4A_5A_6$

```
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_i
```



1) Estrategia de dividir, los subproble
mas son INDEPENDIENTES,no depende
de cómo divide. Existe un criterio
para elegiri un subproblema (General)

2) Caracterización del problema

(Dada un k, el problema se parte
en dos, y las soluciones optimas de
estos pertenecen a la solución optima
global)

3) Caraterización de una estructura
recursiva (Matriz), en esta estructura
mapeamos los subproblemas

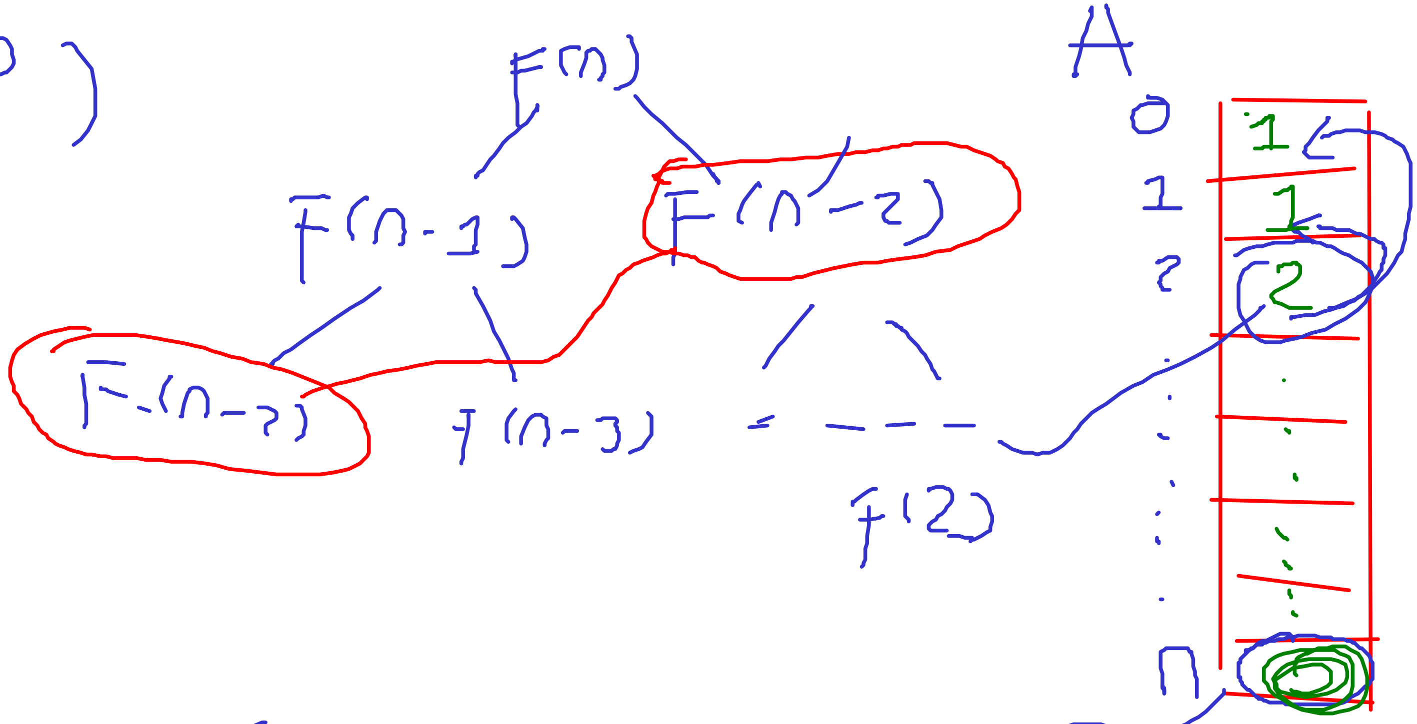
4) Calculo de la soluciones de forma
bottom - up (Soluciones triviales a la
general)

5) Obtenemos la solución de forma
top - down.

$$F(n) = F(n-1) + F(n-2)$$

$F(0) = 1$
$F(1) = 1$

$O(2^n)$



$$A[i] = \begin{cases} 0 & i=0, i=1 \quad F(n) \\ A[i-1] + A[i-2] & i \geq 2 \end{cases}$$

F(12)

0	1
1	1
2	2
3	3
4	5
5	8
6	13
7	21
8	34
9	55
10	89
11	144
12	333