

Fundamentos de análisis y diseño de algoritmos

Programación dinámica

LCS (Longest Common Subsequence)

Multiplicación de matrices

Propiedades generales de la programación dinámica

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)

Suponga que desea multiplicar las matrices A_1, A_2, A_3 , de dimensiones 10×100 , 100×5 y 5×50 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 10×100 , 100×5 y 5×50 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)`

`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 $p \times q$ y B de dimensiones $q \times r$, 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 10×100 , 100×5 y 5×50 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 10×100 , 100×5 y 5×50 respectivamente.

Se tienen dos formas:

- $A_1.(A_2.A_3)$

La cantidad de multiplicaciones de $A_2.A_3$ es $100 \times 5 \times 50 = 2500$, lo cual genera una matriz de 100×50 .

Por lo que $A_1.(A_2.A_3)$ lleva $10 \times 100 \times 50 + 2500 = 75000$ multiplicaciones

- $(A_1.A_2)A_3$

La cantidad de multiplicaciones de $A_1.A_2$ es $10 \times 100 \times 5 = 5000$, lo cual genera una matriz de 10×5 .

Por lo que $(A_1.A_2)A_3$ lleva $10 \times 5 \times 50 + 5000 = 7500$ multiplicaciones

Programación dinámica

Multiplicar las matrices A_1, A_2, A_3 , de dimensiones 10×100 , 100×5 y 5×50 respectivamente.

Se tienen dos formas:

- $A_1.(A_2.A_3)$

La cantidad de multiplicaciones de $A_2.A_3$ es $100 \times 5 \times 50 = 2500$, lo cual genera una matriz de 100×50 .

Por lo que $A_1.(A_2.A_3)$ lleva $10 \times 100 \times 50 + 2500 = 75000$ multiplicaciones

- $(A_1.A_2)A_3$

La cantidad de multiplicaciones de $A_1.A_2$ es $10 \times 100 \times 5 = 5000$, lo cual genera una matriz de 10×5 .

Por lo que $(A_1.A_2)A_3$ lleva $10 \times 5 \times 50 + 5000 = 7500$ multiplicaciones

La solución óptima es $(A_1.A_2)A_3$

Programación dinámica

Problema: Multiplicación de una sucesión de matrices

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

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 $A_i \dots A_j$

- 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

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

Programación dinámica

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

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

	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

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

	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

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

- 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

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

	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

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

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

$$\begin{aligned} m[1,6] = \min \{ & \\ & m[1,1] + m[2,6] + p_0p_1p_6 \\ & m[1,2] + m[3,6] + p_0p_2p_6 \\ & m[1,3] + m[4,6] + p_0p_3p_6 \\ & m[1,4] + m[5,6] + p_0p_4p_6 \\ & m[1,5] + m[6,6] + p_0p_5p_6 \\ & \} \end{aligned}$$

Programación dinámica

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

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

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

$$\begin{aligned} m[1,6] = \min \{ & \\ & m[1,1] + m[2,6] + p_0p_1p_6 \\ & m[1,2] + m[3,6] + p_0p_2p_6 \\ & m[1,3] + m[4,6] + p_0p_3p_6 \\ & m[1,4] + m[5,6] + p_0p_4p_6 \\ & m[1,5] + m[6,6] + p_0p_5p_6 \\ & \} \end{aligned}$$

Programación dinámica

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

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

	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

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

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

$m[3,5]=\{$
 $m[3,3] + m[4,5] + p_2p_3p_5$
 $m[3,4] + m[5,5] + p_2p_3p_5$
 $\}$

Programación dinámica

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

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

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

¿De qué depende $m[i,j]$?

¿Cómo se debería completar/llevar 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

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

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

Completar la matriz por diagonales

Programación dinámica

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

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

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

Completar la matriz por diagonales

Programación dinámica

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

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

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

Completar la matriz por diagonales

Programación dinámica

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

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

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

Completar la matriz por diagonales

Programación dinámica

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

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

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

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] + pi-1pkpj
        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

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

	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

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

	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] + pi-1pkpj
        if q < m[i,j]
          then m[i,j] ← q
          s[i,j] ← k
return m and s
```

Programación dinámica

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

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

$$m[1,2] = \min\{ \\ m[1,1] + m[2,2] + p_0p_1p_2 \\ \}$$

Matriz s, almacena los
valores de k

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 30×35

A_2 35×15

A_3 15×5

A_4 5×10

A_5 10×20

A_6 20×25

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,

$$A_i...A_k A_{k+1}...A_j$$

Programación dinámica

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

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

if $i < j$

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

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

 return $\text{MATRIX-MULTIPLY}(X, Y)$

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 $X \leftarrow \text{MATRIX-CHAIN-MULTIPLY}(A,s,i,s[i,j])$

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

 return $\text{MATRIX-MULTIPLY}(X,Y)$

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)

X ← **MATRIX-CHAIN-MULTIPLY**(A,s,1,3)

Y ← **MATRIX-CHAIN-MULTIPLY**(A,s,4,6)

MATRIX-MULTIPLY(X,Y)

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

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

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	×	0	2	3	3	3
3	×	×	0	3	3	3
4	×	×	×	0	4	5
5	×	×	×	×	0	5
6	×	×	×	×	×	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 $X \leftarrow \text{MATRIX-CHAIN-MULTIPLY}(A,s,i,s[i,j])$

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

return $\text{MATRIX-MULTIPLY}(X,Y)$

else return A_i