

Teoría de Algoritmos

Capítulo 4: Programación Dinámica

Tema 12: Aplicaciones

- Otras aplicaciones de la P.D.
 - Multiplicación encadenada de matrices
 - El Problema de la Secuencia de Máxima longitud
 - El problema del “play-off”

El Problema de la Multiplicación encadenada de matrices

- Dadas n matrices A_1, A_2, \dots, A_n con A_i de dimensión $d_{i-1} \times d_i$
- Determinar el orden de multiplicación para minimizar el número de multiplicaciones escalares.
- Suponemos que la multiplicación de una matriz $p \times q$ por otra $q \times r$ requiere pqr multiplicaciones escalares

El Problema de la Multiplicación encadenada de matrices

- Un producto de matrices se dice que esta completamente parentizado si esta 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.

El Problema de la Multiplicación encadenada de matrices

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

$$A = \begin{matrix} & A_1 & A_2 & A_3 & A_4 \\ & 10 \times 20 & 20 \times 50 & 50 \times 1 & 1 \times 100 \end{matrix}$$

El Problema de la Multiplicación encadenada de matrices

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

$$\text{Costo}(A_3 \times A_4) = 50 \times 1 \times 100$$

$$\text{Costo}(A_2 \times (A_3 \times A_4)) = 20 \times 50 \times 100$$

$$\text{Costo}(A_1 \times (A_2 \times (A_3 \times A_4))) = 10 \times 20 \times 100$$

Costo total = 125000 multiplicaciones

Orden 2 $(A_1 \times (A_2 \times A_3)) \times A_4$

$$\text{Costo}(A_2 \times A_3) = 20 \times 50 \times 1$$

$$\text{Costo}(A_1 \times (A_2 \times A_3)) = 10 \times 20 \times 1$$

$$\text{Costo}((A_1 \times (A_2 \times A_3)) \times A_4) = 10 \times 1 \times 100$$

Costo total = 2200 multiplicaciones

El Problema de la Multiplicación encadenada de matrices

Principio de Optimalidad

- Si $(A_1 \times (A_2 \times A_3)) \times A_4$ es optimal para $A_1 \times A_2 \times A_3 \times A_4$,
- Entonces $(A_1 \times (A_2 \times A_3))$ es optimal para $A_1 \times A_2 \times A_3$
- Razón:
- Si hubiera una solución mejor para el subproblema, podríamos usarla en lugar de la anterior, lo que sería una contradicción sobre la optimalidad de $(A_1 \times (A_2 \times A_3)) \times A_4$

Recuento del numero de parentizaciones

La enumeración de todas las parentizaciones posibles no proporciona un método eficiente.

Notemos el numero 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 $k+1$ siguientes) para cualquier $k = 1, 2, \dots, n-1$, y entonces parentizar las dos subsucesiones resultantes independientemente, obtenemos la recurrencia:

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

Recuento del numero de parentizaciones

La solución de esa ecuación es la sucesión de los Números de Catalan (Eugène Charles Catalan, 1814-1894)

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

Donde

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

es $\Omega(4^n/n^{3/2})$,

Por tanto el numero de soluciones es exponencial en n y, consiguientemente, el método de la fuerza bruta es una pobre estrategia para determinar la parentización optimal de una cadena de matrices.

La Parentización optimal

- Si la parentización optimal de $A_1 \times A_2 \times \dots \times A_n$ se parte entre A_k y A_{k+1} , entonces

parentización optimal
para
 $A_1 \times A_2 \times \dots \times A_n$

{
parentización optimal
para $A_1 \times \dots \times A_k$
parentización optimal
para $A_{k+1} \times \dots \times A_n$

Lo unico que no conocemos es el valor de k

Podemos probar con todos los valores posibles de k , y auquel que devuelva el minimo, es el que escogemos.

La Parentización optimal

- Suponemos que
 - A_1 tiene dimension $p_0 \times p_1$
 - A_2 tiene dimension $p_1 \times p_2$
 - A_i tiene dimension $p_{i-1} \times p_i$
- $A_i \dots A_j$ tiene una solución $p_{i-1} \times p_j$
- Sea $m[i, j]$ el minimo numero de operaciones escalares para $A_i \dots A_j$
- La solución que buscamos es $m[1, n]$

Estructura de los sub-problemas

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

$m[1,3]$

$m[4,6]$

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

$p_0 \times p_3$

$p_3 \times p_6$

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

Estructura de los sub-problemas

$(A_1 A_2 A_3 A_4 A_5 A_6)$

$m[1,6] = ?$

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

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

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

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

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

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

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

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

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

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

$m[1,6] = \min \text{ de}$



Formulación Recursiva

- Las anteriores expresiones nos llevan a que

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

- Donde

- $m[i, k]$ = Costo optimo de $A_i \times \dots \times A_k$
- $m[k+1, j]$ = Costo optimo de $A_{k+1} \times \dots \times A_j$
- $p_{i-1}p_kp_j$ = Costo de $(A_i \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_j)$

Formulación Recursiva

Matrices-Encadenadas-Recursivo (p,i,j)

If $i = 1$

Then Return 0

$m[i,j] = \infty$

For $k = 1$ to $j - 1$ do

$q =$ Matrices-Encadenadas-Recursivo (p,i,k) +
 + Matrices-Encadenadas-Recursivo (p,k+1,j)
 + $p_{i-1} \cdot p_k \cdot p_j$

 If $q < m[i,j]$

 Then $m[i,j] = q$

Return $m[i,j]$

Eficiencia del algoritmo

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

$$T(n) = \sum_{k=1}^{n-1} (T(k) + T(n-k) + O(1))$$
$$= \Omega(2^n)$$

Inaceptable

Muchos sub-problemas se solapan

Eficiencia del algoritmo

- la anterior ecuación puede reescribirse como

$$T(n) = 2 \cdot \sum_{i=1..n-1} T(i) + n$$

- Ahora bien, como

$$T(1) \geq 1 = 2^0$$

- inductivamente para $n \geq 2$ tenemos

$$\begin{aligned} T(n) &= 2 \cdot \sum_{i=1..n-1} 2^{i-1} + n \geq \\ &\geq 2 \cdot \sum_{i=0..n-1} 2^i + n \geq 2(2^{n-1} - 1) + n = \\ &= (2^n - 2) + n \geq 2^{n-1} \end{aligned}$$

- Con lo que se demuestra que el tiempo es al menos exponencial en n .

PD: Solución del problema

- En lugar de calcular las soluciones de la recurrencia anterior de forma recursiva, resolveremos de acuerdo con la tercera etapa de la aplicación de la técnica de la PD.
- El siguiente algoritmo supone que A_i tiene dimensiones $p_{i-1} \cdot p_i$, para cualquier $i = 1, 2, \dots, n$.
- El input es una sucesión $\{p_0, p_1, \dots, p_n\}$ de longitud $n+1$, es decir $\text{leng}[p] = n+1$.
- El procedimiento usa
 - una tabla auxiliar $m[1..n, 1..n]$ para ordenar los $m[i, j]$ costos y
 - una tabla auxiliar $s[1..n, 1..n]$ que almacena que índice de k alcanza el costo optimal al calcular $m[i, j]$.

PD: Solución del problema

Orden-Cadena-Matrices (p)

$n = \text{leng}[p] - 1$

For $i = 1$ to n do $m[i,i] = 0$

For $l = 2$ to n do

 For $i = 1$ to $n - l + 1$ do

$j = i + l - 1$

$m[i,j] = \infty$

 For $k = i$ to $j - 1$ do

$q = m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j$

 If $q < m[i,j]$

 Then $m[i,j] = q; s[i,j] = k$

Return m y s .

Una simple inspeccion a la estructura encajada de los lazos en el anterior algoritmo demuestra que este tiene una eficiencia de $O(n^3)$, ya que hay 3 lazos en i , j y k que, en el peor caso, pueden llegar a tomar el valor n .

Construcción de una Solución Óptima

- Aunque el anterior algoritmo determina el número óptimo de multiplicaciones, no explica cómo multiplicar las matrices.
- La siguiente etapa de la técnica de la DP es la de la construcción de una solución óptima a partir de la información calculada.
- En nuestro caso, usamos la tabla $s[1..n, 1..n]$ para determinar la mejor forma de multiplicar las matrices. Cada elemento $s[i, j]$ almacena el valor k tal que divide óptimamente la parentización de $A_{i-1} \times A_i \times \dots \times A_j$. Por tanto, sabemos que la multiplicación óptima de matrices final para calcular $A_{1..n}$ es $A_{1..s[1,n]} \times A_{s[1,n]+1..n}$.
- La anterior multiplicación de matrices puede efectuarse recursivamente, puesto que $s[1, s[1, n]]$ determina la última multiplicación de matrices al calcular A , y $s[s[1, n]+1, n]$ la última multiplicación de matrices al calcular $A_{1..s[1,n]}$.

Construcción de una Solución Óptima

- El siguiente procedimiento recursivo calcula el producto encadenado de matrices $A_{i..j}$ dadas las matrices de la cadena $A = \{A_1, A_2, \dots, A_n\}$, la tabla s calculada por el algoritmo Orden-Cadena-Matrices, y los índices i y j .
- El algoritmo usa el procedimiento clásico MULT (A, B) de multiplicación de dos matrices.

Multiplica-Cadena-Matrices (A, s, i, j)

If $j > i$

Then $X = \text{Multiplica-Cadena-Matrices}(A, s, i, s[i, j])$

$Y = \text{Multiplica-Cadena-Matrices}(A, s, s[i, j] + 1, j)$

Return MULT (X, Y)

Else Return A_i

Sacar ventajas del enfoque

- El enfoque de la PD,
 1. Naturaleza n-etápica del problema
 2. Verificación del POB
 3. Planteamiento de una recurrencia
 4. Cálculo de la solución (enfoque adelantado o atrasado)
- sugiere un modo de abordar algunos problemas de alta dimensionalidad, que son planteables por medio de tablas de datos.
- La idea es aprovechar la estructura encajada de los subproblemas

El Problema de la Subsecuencia de Mayor Longitud (SML)

- Tenemos dos secuencias: X, Y
- ¿Cuál es la sub-secuencia comun a X e Y de longitud mayor?

- Ejemplo

Si

$X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

entonces la sub-secuencia de mayor longitud comun a X e Y es

$\langle B, C, B, A \rangle$ o $\langle B, D, A, B \rangle$

El Problema de la Subsecuencia de Mayor Longitud (SML)

- Es un problema de la maxima actualidad por sus aplicaciones en el proyecto del Genoma Humano, y en general en Bioinformática
- Secuencias identicas en distintos elementos pueden proporcionarnos una informacion muy relevante.
- El algoritmo de la fuerza bruta consume demasiado tiempo:
- Habria que contrastar 2^m subsecuencias de x contra los n elementos de y
- Eso daría un algoritmo de tiempo $O(n 2^m)$

El Problema de la Subsecuencia de Mayor Longitud (SML)

- Supongamos que

$$X = \langle x_1, x_2, \dots, x_m \rangle$$

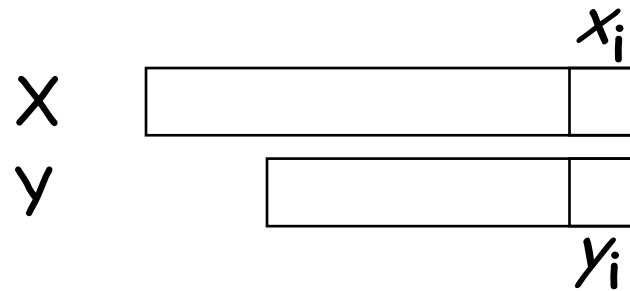
$$Y = \langle y_1, y_2, \dots, y_n \rangle$$

y que la sub-secuencia comun de mayor longitud es

$$Z = \langle z_1, z_2, \dots, z_k \rangle$$

- Si $x_m = y_n$ Entonces $z_k = x_m = y_n$ y z_{k-1} es una SML de x_{m-1} e y_{n-1}
- En otro caso o bien Z es una SML de X_{m-1} o una SML de X e Y_{n-1}

Una solución recursiva



$$x_i = y_j$$

$$SML(X, Y) = SML(X_{i-1}, Y_{j-1}) + x_i$$

$$x_i \neq y_j$$

$$SML(X, Y) = SML(X_{i-1}, Y_j)$$

$$0$$

$$SML(X, Y) = SML(X_i, Y_{j-1})$$

$$l(i, j) = \begin{cases} 0 & \text{si } i=j=0 \\ l(i-1, j-1) + 1 & \text{si } i, j > 0 \text{ y } x_i=y_j \\ \max(l(i-1, j), l(i, j-1)) & \text{si } i, j > 0 \text{ y } x_i \neq y_j \end{cases}$$

Ejemplo


- Consideramos las secuencias
 $X = \langle 0, 1, 1, 0, 1, 0, 0, 1 \rangle$
- y
 $Y = \langle 1, 1, 0, 1, 1, 0 \rangle$
- Entonces disponemos una tabla inicial vacía como,

		j	0	1	2	3	4	5	6
i		y_j	1	1	0	1	1	0	
0	x_i								
1	0								
2	1								
3	1								
4	0								
5	1								
5	0								
7	0								
8	1								

La primera tabla

- En primer lugar completamos con ceros los bordes de la tabla.

		j	0	1	2	3	4	5	6
i		y_j	1	1	0	1	1	0	
0	x_i	0	0	0	0	0	0	0	0
1	0	0							
2	1	0							
3	1	0							
4	0	0							
5	1	0							
5	0	0							
7	0	0							
8	1	0							

- Si $x_i = y_j$ entonces ponemos  en esa casilla, junto con el valor $l(i - 1, j - 1) + 1$
- En otro caso pondremos el mayor de los valores $l(i - 1, j)$ o $l(i, j - 1)$ en la casilla con la flecha apropiada.

La primera tabla

- Es facil calcular la primera fila, comenzando a partir de la posición (1, 1):

		0	1	2	3	4	5	6
		y_j	1	1	0	1	1	0
i	x_i							
0		0	0	0	0	0	0	0
1	0	0	↑ 0	↑ 0	↖ 1	← 1	← 1	↖ 1
2	1	0						
3	1	0						
4	0	0						
5	1	0						
5	0	0						
7	0	0						
8	1	0						

La primera tabla

- Despues de completar fila por fila, podemos alcanzar el array final:

		j	0	1	2	3	4	5	6
i	x_i	y_j	1	1	0	1	1	0	
			0	0	0	0	0	0	0
1	0	0	↑ 0	↑ 0	↖ 1	← 1	← 1	↖ 1	
2	1	0	↖ 1	↖ 1	↑ 1	↖ 2	↖ 2	← 2	
3	1	0	↖ 1	↖ 2	← 2	↖ 2	↖ 3	← 3	
4	0	0	↑ 1	↑ 2	↖ 3	← 3	↑ 3	↖ 4	
5	1	0	↖ 1	↖ 2	↑ 3	↖ 4	↖ 4	↑ 4	
6	0	0	↑ 1	↑ 2	↖ 3	↑ 4	↑ 4	↖ 5	
7	0	0	↑ 1	↑ 2	↖ 3	↑ 4	↑ 4	↖ 5	
8	1	0	↑ 1	↖ 2	↑ 3	↖ 4	↖ 5	↑ 5	

Determinando la SML

- La SML puede encontrar (al contrario) trazando el camino de las flechas a partir de $l(m, n)$. Cada flecha diagonal encontrada nos da un nuevo elemento de la SML.
- El algoritmo es $O(mn)$, que es mucho mejor que los algoritmos de fuerza bruta, que están en $O(n^{2^m})$
- $SML(8,6) = 5$
- Procediendo de este modo encontramos que la SML es 11010
- Notese que si al final del algoritmo (donde tenemos una posibilidad de elegir libremente) hubieramos elegido de $l(8, 6)$ a $l(8, 5)$ habríamos encontrado una SML diferente: 11011

El problema del Play Off

- Supongamos dos equipos A y B que juegan una final en la que se quiere saber cual sera el primero en ganar n partidos, para cierto n particular, es decir, deben jugar como mucho $2n-1$ partidos.
- El problema del play off es tal final cuando el numero de partidos necesarios es $n = 4$.
- Se puede suponer que ambos equipos son igualmente competentes y que, por tanto, la probabilidad de que A gane algun partido concreto es $1/2$.

El problema del Play Off

- Sea $P(i,j)$ la probabilidad de que sea A el ganador final si A necesita i partidos para ganar y B j .
- Antes del primer partido, la probabilidad de que gane A es $P(n,n)$.
- $P(0,i) = 1$ cuando $1 \leq i \leq n$ y $P(i,0) = 0$ cuando $1 \leq i \leq n$. $P(0,0)$ no esta definida.
- Finalmente, como A puede ganar cualquier partido con probabilidad $1/2$ y perderlo con identica probabilidad

$$P(i,j) = [P(i-1,j) + P(i,j-1)]/2, i \geq 1, j \geq 1$$

es decir,

$$\begin{aligned} P(i,j) &= 1 && \text{si } i = 0 \text{ y } j > 0 \\ &= 0 && \text{si } i > 0 \text{ y } j = 0 \\ &= [P(i-1,j) + P(i,j-1)]/2 && \text{si } i > 0 \text{ y } j > 0 \end{aligned}$$

- y podemos calcular $P(i,j)$ recursivamente.

El problema del Play Off

- Sea $T(k)$ el tiempo necesario en el peor de los casos para calcular $P(i,j)$, con $i + j = k$.
- Con este metodo recursivo tenemos que,
$$T(1) = c$$
$$T(k) = 2T(k-1) + d, k > 1$$
- donde c y d son constantes .
- $T(k)$ consume un tiempo en $O(2^k) = O(4^n)$, si $i = j = n$
- No es un metodo demasiado práctico cuando se supone un n grande.
- Intentamos sacar ventaja de la estructura encajada de los sub-problemas (enfoque matricial)

El problema del Play Off

- Otra forma de calcular $P(i,j)$ es rellenando una tabla.

$$P(i,j) = \begin{cases} 1 & \text{si } i = 0 \text{ y } j > 0 \\ 0 & \text{si } i > 0 \text{ y } j = 0 \\ [P(i-1,j)+P(i,j-1)]/2 & \text{si } i > 0 \text{ y } j > 0 \end{cases}$$

- La ultima fila son todos ceros y la ultima columna todos unos por la definicion de las dos primeras lineas de $P(i,j)$.
- Cualquier otra casilla de la tabla es la media de la casilla anterior y la que esta a su derecha.

El problema del Play Off

- Una forma valida de rellenar la tabla es proceder en diagonales, comenzando por la esquina sureste y procediendo hacia arriba a la izquierda a lo largo de las diagonales, que representan casillas con valores constantes de $i+j$

$1/2$	$2/32$	$13/16$	$15/16$	1	4
$11/32$	$1/2$	$11/16$	$7/8$	1	3
$3/16$	$5/16$	$1/2$	$3/4$	1	2
$1/16$	$1/8$	$1/4$	$1/2$	1	1
0	0	0	0	XXX	0
4	3	2	1	0	

$$P(1,1) = (P(0,1) + P(1,0))/2 = (1+0)/2 = 1/2$$

La siguiente funcion realiza estos calculos,

El problema del Play Off

Algoritmo Playoff

Begin

For $s := 1$ to $i+j$ do begin

$P[0,s] = 1.0;$

$P[s,0] = 0.0;$

For $k = 1$ to $s-1$ do

$P[k,s-k] = (P[k-1,s-k] + P[k,s-k-1])/2.0$

end;

Return ($P[i,j]$)

End;

El lazo mas externo se lleva un tiempo $O(\sum s)$, para $s = 1, 2, \dots, n$, es decir, un tiempo en $O(n^2)$ cuando $i+j = n$.

Por tanto, el uso de la PD supone un tiempo $O(n)$, muy inferior al del metodo directo.

Bioinformática: Futuro de la PD

- Hay muchas definiciones:
 - La Bioinformatica trata la gestion y el consiguiente uso de la información biológica, con especial énfasis en la información genética.
 - La Ciencia de la Información aplicada a la Biología produce el campo que se denomina Bioinformática.
 - La Bioinformática trata de la organización y el análisis de los datos biológicos.
- Bioinformatica vs. Biología
Computacional vs. Bio-computación