Algorítmica

Capítulo 4: Programación Dinámica

Tema 12: Aplicaciones

- Otras aplicaciones de la P.D.
  - Multiplicación encadenada de matrices
  - Enfoque tabular de la PD

El Problema de la Multiplicación encadenada de matrices

- Dadas n matrices $A_1, A_2, \ldots, A_n$ 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 pxq por otra qxr requiere pqr multiplicaciones escalares

El Problema de la Multiplicación encadenada de matrices

- Un producto de matrices se dice que está 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.

Programación Dinámica

Aplicaciones

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
  - $(A_1\ (A_2(A_3A_4)))$
  - $(A_1\ ((A_2A_3)A_4))$
  - $((A_1\ A_2)(A_3A_4))$
  - $((A_1\ (A_2A_3))A_4)$
  - $(((A_1\ A_2)A_3)A_4)$

A =	A_1	A_2	A_3	A_4
	10 x 20	20 x 50	50 x 1	1 x 100

El Problema de la Multiplicación encadenada de matrices

Orden 1 A_1 x (A_2 x (A_3 x A_4))

Costo(A_3 x A_4) = 50 x 1 x 100

Costo(A_2 x (A_3 x A_4)) = 20 x 50 x 100

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

Costo total = 125000 multiplicaciones

Orden 2 (A_1 x (A_2 x A_3)) x A_4

Costo(A_2 x A_3) = 20 x 50 x 1

Costo(A_1 x (A_2 x A_3)) = 10 x 20 x 1

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

Costo total = 2200 multiplicaciones

El Problema de la Multiplicación encadenada de matrices

Principio del Óptimo

- Si $(A_1 \times (A_2 \times A_3)) \times A_4$ es óptima para $A_1 \times A_2 \times A_3 \times A_4$,
- Entonces $(A_1 \times (A_2 \times A_3))$ es óptima 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 condición de óptimo de $(A_1 \times (A_2 \times A_3)) \times A_4$

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 k+1 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 (Eugène Charles Catalan, 1814-1894)

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

Donde

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

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

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

La Parentización óptima

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

parentización óptima para $A_1 \times A_2 \times \ldots \times A_n$ $\left\{ \begin{array}{l} \text{parentización óptima para } A_1 \times \ldots \times A_k \\ \text{parentización óptima para } A_{k+1} \times \ldots \times A_n \end{array} \right.$

Lo único que no conocemos es el valor de k

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

La Parentización óptima

- Suponemos que
  A_1 tiene dimensión p_0 x p_1
  A_2 tiene dimensión p_1 x p_2
  A_i tiene dimension p_{i-1} x p_i
- $A_i \ldots A_j$ tiene dimensión p_{i-1} x p_j
- Sea m[i, j] el mínimo número de operaciones escalares para $A_i \ldots A_j$
- La solución que buscamos es m[1,n]

Estructura de los sub-problemas

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

$m[1,3]$ $m[4,6]$

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

$p_0 \times p_3$ $p_3 \times p_6$

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

Estructura de los sub-problemas

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

$m[1\ ,6] = ?$

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

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

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

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

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

$m[1,6] = \min$ de

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

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

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

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

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

Formulación Recursiva

- Las anteriores expresiones nos llevan a que

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

- Donde

- $m[i, k]$ = Costo óptimo de A_i x … x A_k
- $m[k+1, j]$ = Costo óptimo de A_{k+1} x … x A_j
- $p_{i-1}p_kp_j$ = Costo de $(A_i$ x … x $A_k)$ x $(A_{k+1}$x…x $A_j)$

Formulación Recursiva

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

If i = j

Then Return 0

m[i,j] = ∞

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

$$m[i,j] = \min_{i \le k < j} \{ \mathrm{m}[\mathrm{i},\mathrm{k}] + \mathrm{m}[\mathrm{k}+1,\mathrm{j}] + \mathrm{p}_{\mathrm{i-1}}\mathrm{p}_{\mathrm{k}}\mathrm{p}_{\mathrm{j}} \}$$

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

$$= \Omega(2^n)$$

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

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

- 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 $i = 1,2,\ldots,n$.
- El input es una sucesión $\{p_o,p_1,\ldots,p_n\}$ de longitud n+1, es decir long[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 óptima al calcular m[i,j].

PD: Solución del problema

```
Orden-Cadena-Matrices (p)
n = leng[p] - 1
For i = 1 to n do m[i,i] = 0
For l = 2 to n do
    For i = 1 to 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 y s.
```

Una simple inspección 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 l, i 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 optimal de multiplicaciones, no explica como multiplicar las matrices.
- La siguiente etapa de la técnica de la PD 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 optimamente la parentización de $A_{i-1} \times A_{i+1} \times \ldots \times A_j$ Por tanto, sabemos que la multiplicación optimal 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 $A = \{A_1, A_2, \ldots, A_n\}$, la tabla s calculada por el algoritmo Orden-Cadena-Matrices, y los indices 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 = Multiplica-Cadena-Matrices (A,s,i,s[i,j])
        Y = Multiplica-Cadena-Matrices (A,s,s[i,j]+1,j)
        Return MULT (X,Y)
Else Return A_i
```

El enfoque tabular

- El enfoque de la PD sugiere un modo de abordar algunos problemas de alta dimension, que son planteables por medio de tablas de datos.
- La idea es aprovechar la estructura encajada de los subproblemas, aún sabiendo que no se verifica el POB para el problema que estemos considerando.
- Pero el "modus operandi" sugiere un enfoque tabular para resolver algunos problemas
- Algunos casos problemas interesantes que pueden resolverse con este enfoque tabular son
- El del Play Off, el del Turista en Manhattan, el del Cálculo de la Secuencia de ADN más larga, el Cálculo de Números de Fibonacci…

El problema del Play Off

- Supongamos dos equipos A y B que juegan una final en la que se quiere saber cual será 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 número 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 algún 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 idéntica probabilidad

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

es decir,

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

- 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 método 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 método demasiado práctico cuando se supone un n grande.
- Intentamos sacar ventaja de la estructura encajada de los sub-problemas (enfoque matricial)
- No es un problema en el que se pueda aplicar PD, pero si se puede aplicar la metodología que subyace en la PD por estar encajados los sub-problemas.

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 última fila son todos ceros y la última columna todos unos por la definición de las dos primeras líneas de P(i,j).
- Cualquier otra casilla de la tabla es la media de la casilla anterior y la que está a su derecha.

El problema del Play Off

- Una forma válida 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	21/32	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 función realiza estos cálculos,

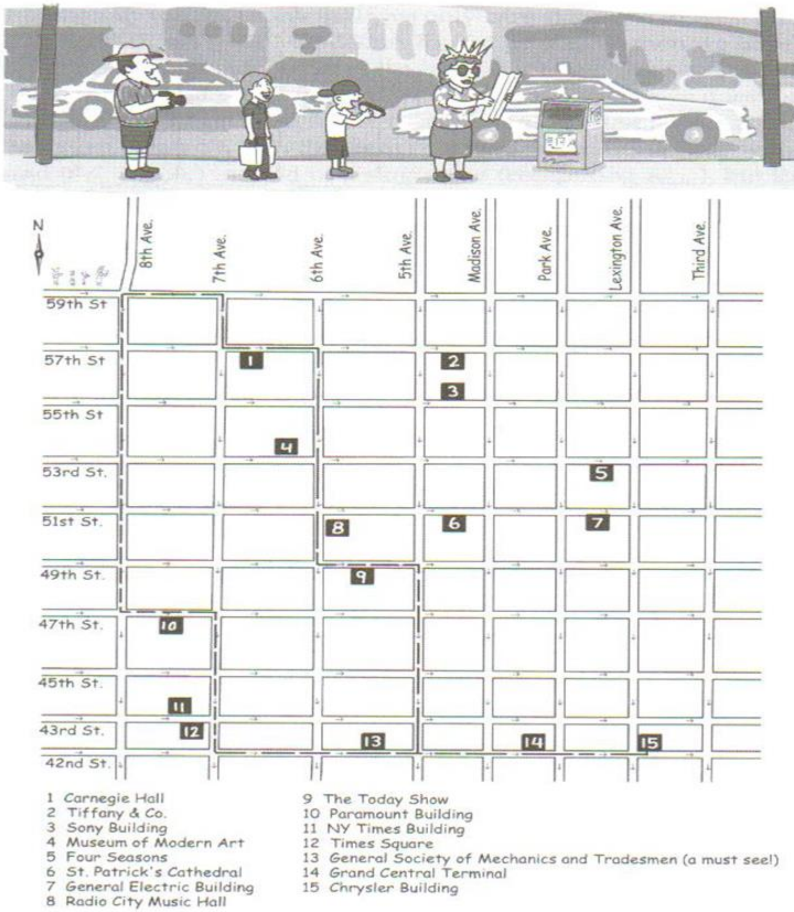
El problema del Play Off

```
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 que es $O(\Sigma s)$, para $s = 1,2,..,n$, es decir, un tiempo en $O(n^2)$ cuando $i+j = n$.

Por tanto, el uso de la PD supone un tiempo $O(n^2)$, muy inferior al del método directo.

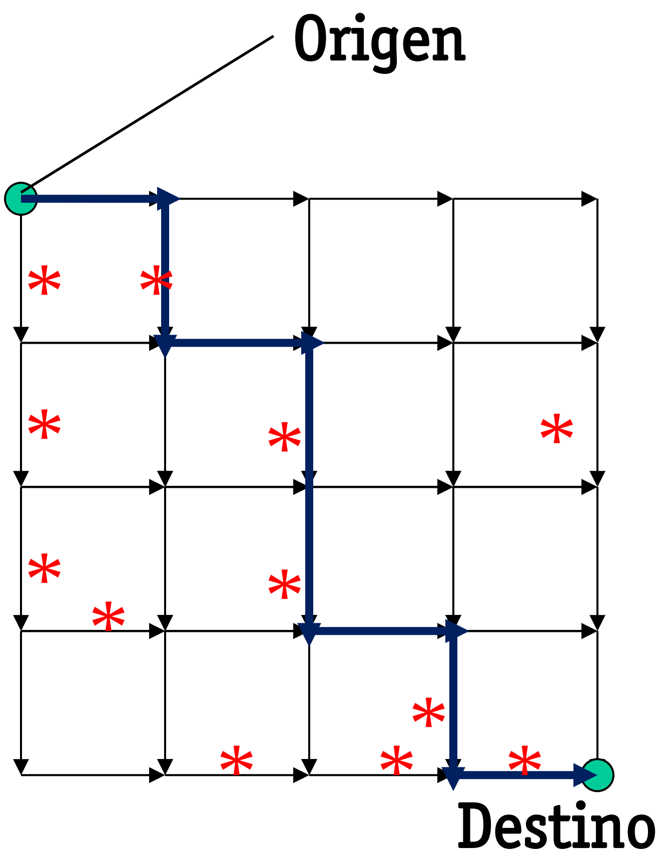
Ejercicio: El Problema del Turista en Manhattan*



* An introduction to bioinformatics algorithms. N. C. Jones and P. Pevzner, 2004

Ejercicio: El Problema del Turista en Manhattan*

Se trata de encontrar un camino (desde un origen a un destino) tal que caminando solo hacia el sur y hacia el este, nos permita visitar el mayor número lugares turísticos (*) en el mapa de Manhattan.



El problema, como el del Play-Off, no verifica el POB, pero igual que aquel es n-etápico y se le puede aplicar el enfoque matricial que sugiere la PD.

Otras aplicaciones de la Programación Dinámica

- Distancia entre dos series temporales: $O(n^2)$
  DTW (Dynamic Time Warping)
- Análisis Sintáctico: $O(n^3)$
  - Algoritmo CKY (Cocke, Kasami y Younger)
  - Algoritmo de Earley
- Algoritmo de Viterbi: HMM (Hidden Markov Models)
  - Decodificación de señales (p. ej. Modems, GSM, Wi-Fi, …)
  - Procesamiento de lenguaje natural
  - Bioinformática
- Optimización de consultas en bases de datos relacionales
- Comparación de ficheros con diff
- …