



Algoritmos y Estructuras de Datos II

Clase 14

Carreras:

Licenciatura en Informática

Ingeniería en Informática

2024

Viaje en Tren



- Para viajar en el Tren de la Costa desde Capital al Delta del Tigre hay $n=11$ estaciones numeradas de: 1 (Maipú) a 11(Delta).



- En cada una de ellas se puede seguir en el tren o bajar y subir en el próximo tren para ir a cualquier otra estación entre las siguientes.

Viaje en Tren



- Existe un cuadro de tarifas que indica el costo del viaje de la estación i a la estación j para cualquier estación de partida i y cualquier estación de llegada j más adelante en el viaje ($i < j$).

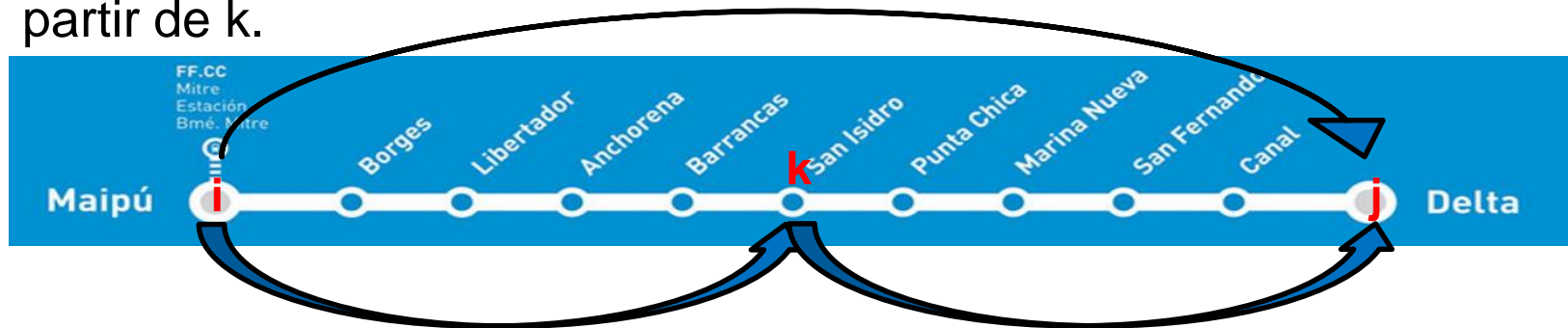
Cuadro tarifario:

ESTACIONES	Av. Maipu	Borges	Libertador	J. Anchorena	Las Barrancas	San Isidro R.	Punta Chica	Marina Nueva	San Fernando R.	Canal San Fernando	Delta
Av. Maipu	0										
Borges	-	0									
Libertador	-	-	0			i, j					
J. Anchorena	-	-	-	0							
Las Barrancas	-	-	-	-	0						
San Isidro R.	-	-	-	-	-	0					
Punta Chica	-	-	-	-	-	-	0				
Marina Nueva	-	-	-	-	-	-	-	0			
San Fernando R.	-	-	-	-	-	-	-	-	0		
Canal San Fernando	-	-	-	-	-	-	-	-	-	0	
Delta	-	-	-	-	-	-	-	-	-	-	0

Viaje en Tren



- Puede suceder que un viaje de i a j sea más caro que una sucesión de viajes cortos, en cuyo caso si se quiere ahorrar, se toma el primer tren hasta la estación k y un segundo tren para continuar a partir de k .



- No hay costo adicional en cambiar de tren.
- El problema que se plantea es diseñar un algoritmo eficiente usando la técnica de programación dinámica que determine el **costo mínimo para viajar entre cada par de ciudades i, j rumbo al Tigre** ($i < j$) y determinar el costo del algoritmo en función de n ($n=11$ en este ejemplo).

Viaje en Tren

Solución:

- **Entrada:**

A una matriz triangular superior de orden n .

$A(i,j)$: costo del pasaje para ir de la estación i a la j (directamente sin bajar del tren).

- **Salida:**

C una matriz triangular superior de orden n .

$C(i,j)$: costo mínimo para ir de la estación i a la estación j (con posible escala y trasbordo).

Viaje en Tren

- El costo de *quedarse en la estación*, se representa en la diagonal principal de la matriz de costos:

$$C(i,i)=0$$

- El costo de *ir de una estación a la siguiente* estación está dado por la matriz de tarifas, y representada por la primera diagonal sobre la principal:

$$C(i,i+1)=A(i,i+1)$$

- Para viajar de la estación i a la estación j se puede hacer *con una primera parada* en la estación k , entonces:

$$C(i,j)=A(i,k)+C(k,j) \quad , \quad i < k \leq j$$

Viaje en Tren

- Generalizando se obtiene:

$$\begin{array}{ll} C(i,j)=0 & \text{si } i=j \\ C(i,j)=A(i,k)+C(k,j) & \text{si } i < k \leq j \end{array}$$

- Esta última contempla el viaje directo cuando $k=j$.
- Debe satisfacer el principio de óptimo, entonces:

$$\begin{array}{ll} C(i,j)=0 & \text{si } i=j \\ C(i,j)=\underset{i < k \leq j}{\text{Minimo}} \{ A(i,k)+C(k,j) \} & \text{si } i < j \end{array}$$

Ejercitación: escribir el algoritmo y calcular su costo

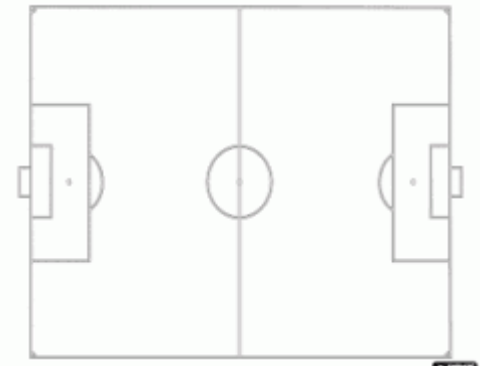
Apuestas entre dos equipos

- Sean A y B dos equipos de fútbol que compiten en un campeonato..
- Juegan como máximo $2n-1$ partidos para ver quien es el primero en ganar n partidos para un dado n .
- Se supone que A y B son igualmente competentes, de modo que cada uno tiene un 50% de posibilidad de ganar cada partido.
- Se supone también que no hay empates.

Calcular $P(i,j)$, donde:

$P(i,j)$ es la probabilidad de que A gane el campeonato, considerando que:

- A necesita ganar i partidos para resultar ganador
- B necesita ganar j partidos mas para ser ganador.



Apuestas entre dos equipos

- Antes del primer partido, la probabilidad de que gane el equipo A es $P(n,n)$. Ambos equipos necesitan todavía ganar n partidos para ganar el campeonato.

- Si el equipo A ha ganado el campeonato, no necesita más victorias, entonces:

$$P(0, j) = 1, \quad 1 \leq j \leq n$$

- Si el equipo B ha ganado el campeonato, no necesita más victorias, entonces:

$$P(i, 0) = 0, \quad 1 \leq i \leq n$$

- $P(0,0)$ no tiene sentido porque significaría que A y B son ganadores.
- Si $i \geq 1$ y $j \geq 1$ entonces debe jugarse al menos un partido más, y los dos equipos ganan la mitad de las veces.

Apuestas entre dos equipos

- Entonces $P(i,j)$ debe ser el promedio entre las dos situaciones: de que A gane el mundial si gana próximo partido y que gane el mundial aunque pierda el siguiente partido:

$$P(i,j) = (P(i-1,j) + P(i,j-1)) / 2$$

- Resumiendo:

$$P(i,j) = 1$$

$$\text{si } i=0 \quad j>0$$

$$P(i,j) = 0$$

$$\text{si } i>0 \quad j=0$$

$$P(i,j) = (P(i-1,j) + P(i,j-1)) / 2$$

$$\text{si } i \geq 1 \quad j \geq 1$$

Apuestas entre dos equipos

Se puede calcular $P(i,j)$ con la siguiente función recursiva:

Función $P(i,j)$: entero ≥ 0 x entero $\geq 0 \rightarrow$ real ≥ 0

Si $i=0$ entonces

Retorna 1

sino si $j=0$ entonces

Retorna 0

sino

Retorna ($P(i-1,j) + P(i,j-1)$) / 2

Fin

Apuestas entre dos equipos

Costo de la función:

$$\begin{array}{ll} T(i,j)=c & \text{si } i=0 \text{ or } j=0 \\ T(i,j)=T(i-1,j)+T(i,j-1)+d & \text{si } i>0 \text{ } j>0 \end{array}$$

Se puede demostrar que: $T(i,j) \in O(2^{i+j})$

Si $i=j=n$ entonces puedo decir: $T(n) \in O(4^n)$

También se puede demostrar que: $T(n) \in \Omega(4^n/n)$

Esto indica que el cálculo de P mediante la función recursiva es ineficiente.

Apuestas entre dos equipos

- Una forma más eficiente de calcular $P(i,j)$ es *llenar una tabla* que almacena las probabilidades y no las recalcula.
- La primera fila ($i=0$) se llena con 1, la primera columna ($j=0$) con 0.

$$P(i,j) = 1 \quad \text{si } i=0 \quad j>0$$

$$P(i,j) = 0 \quad \text{si } i>0 \quad j=0$$

- Los demás elementos se calculan con la fórmula recurrente como promedio del elemento de la fila anterior y de la columna anterior.

$$P(i,j) = (P(i-1,j) + P(i,j-1)) / 2 \quad \text{si } i>0 \quad j>0$$

Apuestas entre dos equipos

Un algoritmo usando programación dinámica:

Función **Apuesta**(n): entero $\geq 0 \rightarrow$ real ≥ 0

Auxiliar : $P(0..n, 0..n)$ // se llena por diagonales

Para $s=1$ hasta n hacer // completa hasta la primera diagonal

$P(0,s)=1$; $P(s,0)=0$

Para $k=1$ hasta $s-1$ hacer

$P(k,s-k) \leftarrow (P(k-1,s-k) + P(k,s-k-1)) / 2.0$

Para $t=1$ hasta n hacer // completa desde la primera diagonal

Para $k=0$ hasta $n-t$ hacer

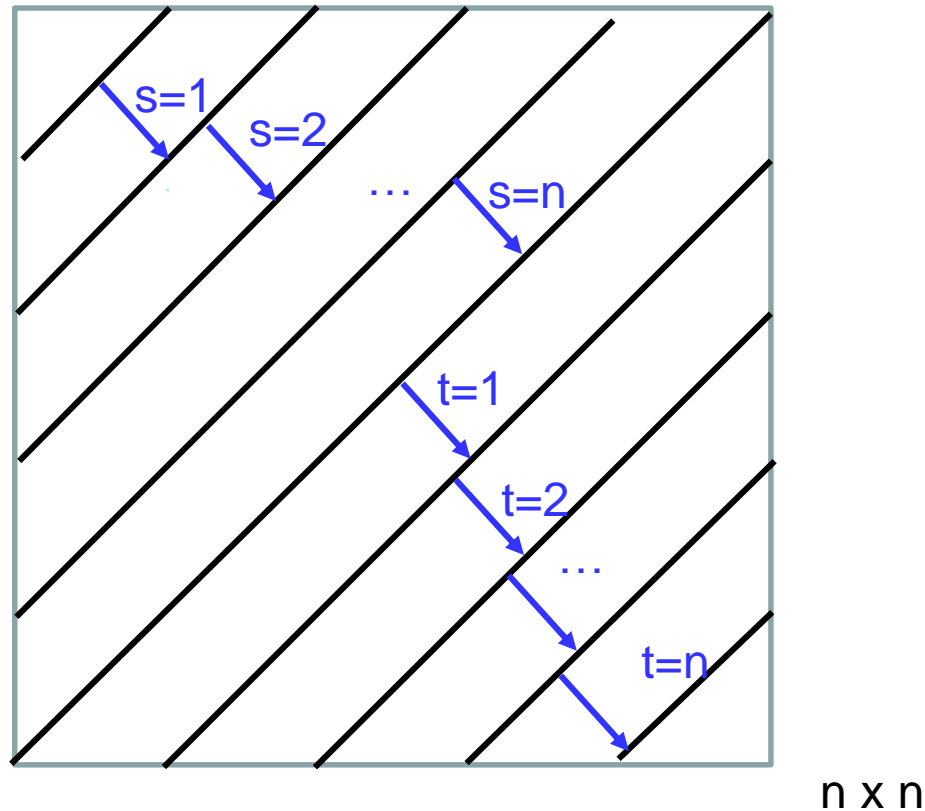
$P(t+k,n-k) \leftarrow (P(t+k-1,n-k) + P(t+k,n-k-1)) / 2.0$

Retorna $P(n,n)$

Fin

Apuestas entre dos equipos

Esquema de iteración del algoritmo Apuesta:



Apuestas entre dos equipos

Esquema de iteración del algoritmo para el ejemplo $n=4$:

s=1 x
s=2 x
s=3 x
s=4 x
t=1 x
t=2 x
t=3 x
t=4 x

	0	1	2	3	4
0		x	x	x	x
1	x	x	x	x	x
2	x	x	x	x	x
3	x	x	x	x	x
4	x	x	x	x	x

Apuestas entre dos equipos

Ejemplo: para $n=4$ $s=1$

$i \backslash j$	0	1	2	3	4
0		1			
1	0				
2					
3					
4					

Apuestas entre dos equipos

Ejemplo: para $n=4$ $s=2$

$i \backslash j$	0	1	2	3	4
0		1	1		
1	0	1/2			
2	0				
3					
4					

Apuestas entre dos equipos

Ejemplo: para $n=4$ $s=3$

$i \backslash j$	0	1	2	3	4
0		1	1	1	
1	0	$1/2$	$3/4$		
2	0	$1/4$			
3	0				
4					

Apuestas entre dos equipos

Ejemplo: para $n=4$ $s=4$

$i \setminus j$	0	1	2	3	4
0		1	1	1	1
1	0	$1/2$	$3/4$	$7/8$	
2	0	$1/4$	$1/2$		
3	0	$1/8$			
4	0				

Apuestas entre dos equipos

Ejemplo: para $n=4$ $t=1$

$i \backslash j$	0	1	2	3	4
0		1	1	1	1
1	0	$1/2$	$3/4$	$7/8$	$15/16$
2	0	$1/4$	$1/2$	$11/16$	
3	0	$1/8$	$5/16$		
4	0	$1/16$			

Apuestas entre dos equipos

Ejemplo: para $n=4$ $t=2$

$i \setminus j$	0	1	2	3	4
0		1	1	1	1
1	0	$1/2$	$3/4$	$7/8$	$15/16$
2	0	$1/4$	$1/2$	$11/16$	$13/16$
3	0	$1/8$	$5/16$	$1/2$	
4	0	$1/16$	$3/16$		

The diagram illustrates a sequence of bets starting from the cell (1, 15/16) and moving down-left to the cell (4, 3/16). The cells (2, 13/16), (3, 1/2), and (4, 3/16) are circled in blue, indicating the path of the bets. Blue arrows show the sequence of moves: from (1, 15/16) to (2, 13/16), then to (3, 11/16), then to (4, 5/16), then to (3, 1/2), then to (4, 3/16), and finally to (4, 1/16).

Apuestas entre dos equipos

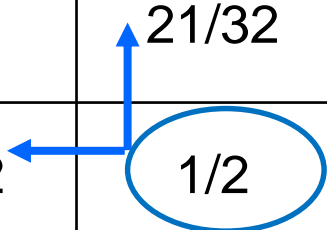
Ejemplo: para $n=4$ $t=3$

$i \setminus j$	0	1	2	3	4
0		1	1	1	1
1	0	$1/2$	$3/4$	$7/8$	$15/16$
2	0	$1/4$	$1/2$	$11/16$	$13/16$
3	0	$1/8$	$5/16$	$1/2$	$21/32$
4	0	$1/16$	$3/16$	$11/32$	

Apuestas entre dos equipos

Ejemplo: para $n=4$ $t=4$

$i \setminus j$	0	1	2	3	4
0		1	1	1	1
1	0	$1/2$	$3/4$	$7/8$	$15/16$
2	0	$1/4$	$1/2$	$11/16$	$13/16$
3	0	$1/8$	$5/16$	$1/2$	$21/32$
4	0	$1/16$	$3/16$	$11/32$	$1/2$



Apuestas entre dos equipos

Ejemplo: para $n=4$

$i \setminus j$	0	1	2	3	4
0		1	1	1	1
1	0	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{7}{8}$	$\frac{15}{16}$
2	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{11}{16}$	$\frac{13}{16}$
3	0	$\frac{1}{8}$	$\frac{5}{16}$	$\frac{1}{2}$	$\frac{21}{32}$
4	0	$\frac{1}{16}$	$\frac{3}{16}$	$\frac{11}{32}$	$\frac{1}{2}$

Apuestas entre dos equipos

Costo de la función Apuesta:

El algoritmo completa una matriz $n \times n$ por diagonales, de modo que el tiempo de ejecución es:

$$T(n) \in O(n^2)$$

Además necesita memoria para almacenar la tabla proporcional a $O(n^2)$

Multiplicación Óptima De Matrices

Producto de A_{mxq} por una matriz $B_{q \times n}$ es una matriz $C_{m \times n}$ dada por:

$$c_{i,j} = \sum_{k=1}^q a_{i,k} \cdot b_{k,j} \quad 1 \leq i \leq m \quad 1 \leq j \leq n$$

Algorítmicamente un lazo triple anidado hace $m \times n \times q$ multiplicaciones y es de la forma:

Para $i=1, m$ hacer

Para $j=1, n$ hacer

$c(i,j) \leftarrow 0$

Para $k=1, q$ hacer

*$c(i,j) \leftarrow c(i,j) + a(i,k) * b(k,j)$*

Multiplicación Óptima De Matrices

Producto de más de dos matrices:

$$M = M_1 M_2 \dots M_n$$

La **multiplicación de matrices es asociativa**, de modo que se puede calcular el producto matricial de distintas maneras, y todas darán la misma respuesta.

Sin embargo la **multiplicación de matrices no es conmutativa**, así que no se permite modificar el orden de las matrices involucradas.

Objetivo: calcular la matriz producto M de n matrices **minimizando el número total de multiplicaciones escalares a realizar**.

Este problema ya fue planteado, y se demostró que los algoritmos greedy presentados no encuentran solución en todos los casos.

Multiplicación Óptima De Matrices

Ejemplo: Sean 4 matrices:

A de 13×5

B de 5×89

C de 89×3

D de 3×34

El número de multiplicaciones al efectuar el producto:

AB	5785 multiplicaciones
(AB)C	3471 multiplicaciones
((AB)C)D	1324 multiplicaciones
TOTAL=	10582 multiplicaciones

Multiplicación Óptima De Matrices

Además de la forma anterior existen otras que se listan a continuación con su correspondiente número total de multiplicaciones:

$((AB)C)D$	10582 multiplicaciones
$(AB)(CD)$	54201 multiplicaciones
$(A(BC))D$	2856 multiplicaciones
$A((BC)D)$	4055 multiplicaciones
$A(B(CD))$	26418 multiplicaciones

Observar que en el tercer caso se hace muchas menos que en el segundo.

Multiplicación Óptima De Matrices

Solución por la *fuerza bruta*:

Calcular el costo de cada una de las opciones posibles y elegir la mejor entre ellas antes de multiplicar.

Ya se ha visto que para valores grandes de n esta estrategia es inútil, pues crece exponencialmente con n

El numero de opciones posibles sigue la sucesión de los números de Catalán:

$$T(n) = \sum_{i=1}^{n-1} T(i)T(n-i) = \frac{1}{n} \binom{2n-2}{n-1} \in \Theta\left(\frac{4^n}{\sqrt{n}}\right)$$

N	1	2	3	4	5	10	15
T(n)	1	1	2	5	14	4862	2674440

Multiplicación Óptima De Matrices

Solución con programación dinámica

Cada M_i es de dimensión $d_{i-1}d_i$ ($1 \leq i \leq n$),

Realizar la multiplicación $M_i \times M_{i+1}$, requiere un total de $d_{i-1}d_i d_{i+1}$ multiplicaciones escalares.

$M(i,j)$: número mínimo de multiplicaciones necesarias

para el cálculo del producto de $M_i \times M_{i+1} \dots \times M_j$ con $1 \leq i \leq j \leq n$.

La solución al problema planteado estará en: $M(1,n)$.

Multiplicación Óptima De Matrices

Ecuación en recurrencia que define la solución:

Se asocia las matrices de la siguiente manera:

$$\underbrace{(M_i \ M_{i+1} \dots M_k)}_{M(i, k)} \underbrace{(M_{k+1} \ M_{k+2} \dots M_j)}_{M(k+1, j)}$$

$$\underbrace{\hspace{10em}}_{d_{i-1} d_k d_j}$$

El valor de $M(i, j)$ viene dado por la expresión:

$$M(i, j) = M(i, k) + M(k+1, j) + d_{i-1} d_k d_j$$

Aplicando el principio de óptimo, k puede tomar cualquier valor entre i y $j-1$. $M(i, j)$, deberá elegir el más favorable de entre todos ellos:

$$M(i, j) = \underset{i \leq k < j}{\text{Min}} \{ M(i, k) + M(k+1, j) + d_{i-1} d_k d_j \}$$

Multiplicación Óptima De Matrices

Relación en recurrencia:

$$\begin{aligned} M(i, j) &= 0 && \text{si } i = j \\ M(i, j) &= \underset{i \leq k < j}{\text{Min}} \{ M(i, k) + M(k+1, j) + d_{i-1}d_kd_j \} && \text{en otro caso.} \end{aligned}$$

Para resolver esta ecuación en un tiempo de complejidad polinómico:

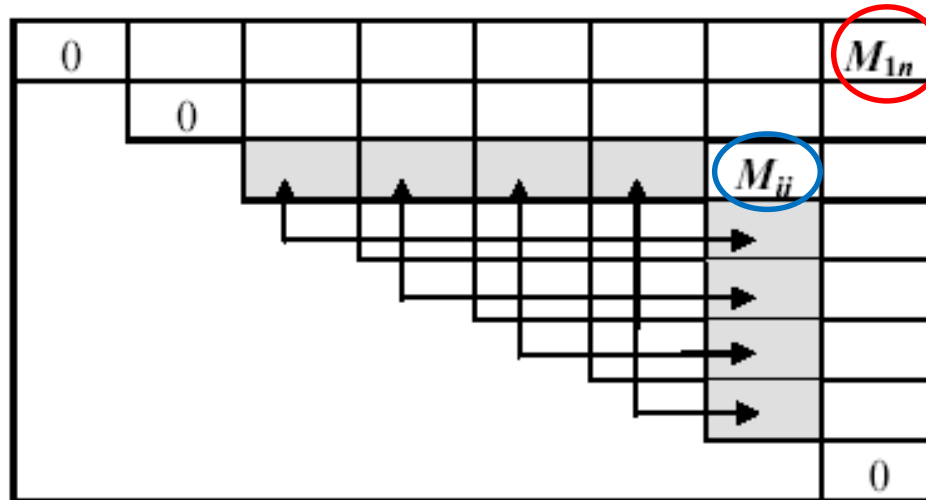
- Crear **una tabla** en la que se vayan almacenando los valores:

$$M(i, j) \ (1 \leq i \leq j \leq n)$$

A partir de las condiciones iniciales reutilizar los valores calculados en los pasos anteriores.

Multiplicación Óptima De Matrices

- Esta tabla se irá completando por diagonales sabiendo que los elementos de la diagonal principal son todos cero.



$$M(i, j) = 0$$

$$M(i, j) = \text{Min}_{i \leq k < j} \{ M(i, k) + M(k+1, j) + d_{i-1}d_kd_j \}$$

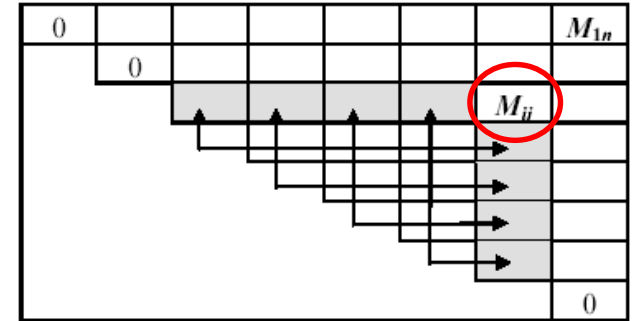
si $i = j$

en otro caso.

- La solución se encuentra en el extremo superior derecho, que indica el número de multiplicaciones escalares buscado: **$M(1, n)$** .

Multiplicación Óptima De Matrices

- Cada elemento $M(i,j)$ será el valor mínimo de entre todos los pares $(M(i,k) + M(k+1,j))$ señalados con la línea de doble flecha en la figura, más el aporte de la última multiplicación $(d_{i-1}d_kd_j)$.



- Los valores que requiere el cálculo de $M(i,j)$ y que el algoritmo reutiliza para conseguir un tiempo de ejecución aceptable se encuentran sombreados.
- Al ir rellenando la tabla por diagonales se asegura que esta información $(M(i,k), M(k+1,j))$ está disponible cuando se necesita, pues cada $M(i,j)$ utiliza para su cálculo todos los elementos anteriores de su fila y todos los de su columna por debajo suya.

Multiplicación Óptima De Matrices

- Si además se quiere conocer cómo es la asociación que corresponde a este óptimo es necesario conservar para cada elemento $M(i,j)$ el valor de k para el cual la expresión:

$$M(i,k) + M(k+1,j) + d_{i-1}d_kd_j \text{ es mínima,}$$

construyendo otra tabla que se denomina *Indice*.

El algoritmo *Matriz* permite la creación de las tablas *M* e *Indice*:

- En la tabla *M* se almacenan los valores del número mínimo de multiplicaciones
- En la tabla *Indice* la información necesaria para construir la asociación óptima.

Multiplicación Óptima De Matrices

$M(i,j)$ (1..n,1..n) : matriz de enteros

$Indice$ (1..n,1..n) : matriz de enteros

d (0..n) : vector de enteros

Algoritmo **Matriz**

ENTRADA: d, n

SALIDA: $M, Indice$

Para $i=1, n$ hacer

$M(i,i) \leftarrow 0$

Para $j=1, n-1$ hacer

Para $i=1, n-j$ hacer

$(min, kmin) \leftarrow \text{Minimo}(d, M, i, i+j)$

$M(i, i+j) \leftarrow min$

$Indice(i, i+j) \leftarrow kmin$

Retorna $(M, Indice)$

Fin

Multiplicación Óptima De Matrices

Función **Minimo**: calcula el mínimo de la expresión en recurrencia y devuelve:

- el valor de este mínimo,
- el valor de k para el que se lo alcanza

Función **Minimo**(d, M, i, j): vector x matriz x $\text{ent} \geq 0$ x $\text{ent} \geq 0 \rightarrow (\text{ent} \geq 0$ x $\text{ent} \geq 0$)

$min \leftarrow \infty$

Para $k=i, j-1$ Hacer

$aux \leftarrow M(i, k) + M(k+1, j) + d(i-1) * d(k) * d(j)$

Si $aux < min$ entonces

$min \leftarrow aux$

$kmin \leftarrow k$

Retorna ($min, kmin$)

Fin

Multiplicación Óptima De Matrices

Costo del algoritmo:

- La función **Matriz** tiene dos iteraciones anidadas y una llamada a la función Minimo.
- La función **Minimo** tiene una complejidad del orden del valor de j.
- Por tanto el tiempo de ejecución del algoritmo es:

$$\sum_{j=1}^{n-1} (n-j)j = n \sum_{j=1}^{n-1} j - \sum_{j=1}^{n-1} j^2 = \frac{n^3 - n}{6}$$

La **complejidad temporal** es de orden **$O(n^3)$** .

La **complejidad espacial** del algoritmo es de orden **$O(n^2)$** .

Multiplicación Óptima De Matrices

El algoritmo **Orden** sirve para reconstruir la solución a partir de la tabla **Indice**. Da un listado de manera de multiplicar las matrices para obtener ese valor mínimo:

Algoritmo Orden(*Indice, i, j*) : matriz $x \text{ ent} \geq 0$ $x \text{ ent} \geq 0 \rightarrow$ tira de char

Si $i=j$ entonces

 Escribir (“M”)

 Escribir(*i*)

sino

$k \leftarrow \text{Indice}(i, j)$

 Escribir("(");

Orden (*Indice, i, k*)

 Escribir("x");

Orden (*Indice, k+1, j*)

 Escribir(")")

Fin

Multiplicación Óptima De Matrices

El algoritmo Matriz encuentra el valor de una solución óptima junto con una de las formas de obtenerla. Sin embargo, existen casos en donde puede haber más de una solución óptima, esto es, distintas formas de multiplicar las matrices para obtener el valor óptimo.

Ejemplo: Dadas las matrices $M_1(10 \times 10)$, $M_2(10 \times 50)$ y $M_3(50 \times 50)$.

Existen dos formas de asociarlas para multiplicarlas con la misma cantidad de multiplicaciones:

$$(M_1 M_2) M_3 = 10 \cdot 10 \cdot 50 + 10 \cdot 50 \cdot 50 = 30000$$

$$M_1 (M_2 M_3) = 10 \cdot 50 \cdot 50 + 10 \cdot 10 \cdot 50 = 30000$$

Es posible modificar el algoritmo para que encuentre **todas las soluciones** que llevan al valor óptimo, y para esto se usa la tabla *Indice*. Generalmente, esta modificación tiene poco interés desde un punto de vista práctico.

Multiplicación Óptima De Matrices

Ejemplo: $A(13 \times 5)$, $B(5 \times 89)$, $C(89 \times 3)$ y $D(3 \times 34)$ $d=(13,5,89,3,34)$.

Para $j=1$:

$$m_{12} = d_0 \cdot d_1 \cdot d_2 = 13 \times 5 \times 89 = 5785,$$

$$m_{23} = d_1 \cdot d_2 \cdot d_3 = 5 \times 89 \times 3 = 1335,$$

$$m_{34} = d_2 \cdot d_3 \cdot d_4 = 89 \times 3 \times 34 = 9078.$$

Para $j=2$:

$i \setminus j$	1	2	3	4
1	0	5785	1530	2856
2		0	1335	1845
3			0	9078
4				0

$$m_{13} = \min \begin{cases} m_{11} + m_{23} + d_0 \cdot d_1 \cdot d_3 \\ m_{12} + m_{33} + d_0 \cdot d_2 \cdot d_3 \end{cases} = \min(1530, 9256) = 1530$$

$$m_{24} = \min \begin{cases} m_{22} + m_{34} + d_1 \cdot d_2 \cdot d_4 \\ m_{23} + m_{44} + d_1 \cdot d_3 \cdot d_4 \end{cases} = \min(24208, 1845) = 1845$$

Para $j=3$:

$$m_{14} = \min \begin{cases} m_{11} + m_{24} + d_0 \cdot d_1 \cdot d_4 \\ m_{12} + m_{34} + d_0 \cdot d_2 \cdot d_4 \\ m_{13} + m_{44} + d_0 \cdot d_3 \cdot d_4 \end{cases} = \min(4055, 54201, 2856) = 2856$$

Programación Dinámica con Recursión

- Aun cuando los algoritmos de programación dinámica son eficientes, hay algo poco convincente en el enfoque ascendente.
- Un método descendente, ya sea Divide & Conquer, refinamiento sucesivo o recursión parece más natural, sobre todo para aquellos a quienes se ha enseñado a desarrollar diseños y programas de esta manera.
- Una razón de más peso es que el enfoque ascendente lleva a calcular valores que podrían ser totalmente irrelevantes.
- **¿Se puede alcanzar la misma eficiencia en una versión descendente del algoritmo?**

Programación Dinámica con Recursión

Ejemplo: multiplicación matricial.

Sustituir la tabla M por una función fm que se calcule recursivamente:

$$fm(i,j) = M(i,j) \quad \text{para } 1 \leq i \leq j \leq n$$

La función **fm** se escribe con las reglas para calcular M :

Función $fm(i,j,d)$: entero ≥ 1 x entero ≥ 1 x vector \rightarrow entero ≥ 0

Si $i=j$ entonces

 retorna 0

sino

$valor \leftarrow \infty$

 Para $k=i, j-1$ hacer

$valor \leftarrow \min(valor, fm(i,k,d) + fm(k+1,j,d) + d(i-1)*d(k)*d(j))$

 retorna $valor$

Fin

Programación Dinámica con Recursión

Costo del algoritmo:

- Para averiguar cuantas multiplicaciones escalares se necesitan para calcular el producto de n matrices $M = M_1 M_2 \dots M_n$, se invoca a $fm(1, n, d)$.
- Esta función fm , tiene 2 llamadas recursivas. Esto implica el cálculo de 2 casos en forma independiente que se solapan.
- Se puede concluir que el tiempo para multiplicar n matrices es:
$$T(n) \in \Omega(2^n)$$
$$T(n) \in O(3^n)$$

(ver detalles en la bibliografía)
- Resulta entonces mucho más ineficiente que el algoritmo de programación dinámica que ya se desarrolló.

Programación Dinámica con Recursión

¿Será posible combinar las ventajas de la eficiencia de la programación dinámica y la sencillez de la recursión?

- Las **funciones con memoria** son funciones recursivas a las que se les agrega almacenamiento en forma de tabla.
- La tabla se inicializa en un valor especial que indica que no se ha calculado esa entrada todavía.
- Luego se inspecciona primero la tabla para ver si la entrada ya ha sido calculada, si es así se usa ese valor, sino se calcula su valor y se lo almacena.
- De esa forma no se calcula dos veces la función para el mismo conjunto de parámetros.

Programación Dinámica con Recursión

Ejemplo: multiplicación matricial.

La función **fm-tabla** usa una *tabla* como variable global
Debe estar ya inicializada en: $tabla(i,j) \leftarrow (-1)$, $\forall i, \forall j$

Función **fm-tabla(i,j,d):entero ≥ 1 x entero ≥ 1 x vector \rightarrow entero ≥ 0**

Si $i=j$ entonces retorna 0

sino si $tabla(i,j) \geq 0$ entonces retorna $tabla(i,j)$

sino

$valor \leftarrow \infty$

Para $k=i, j-1$ hacer

$valor \leftarrow \min(valor, \text{fm-tabla}(i,k,d) + \text{fm-tabla}(k+1,j,d) + d(i-1)*d(k)*d(j))$

$tabla(i,j) \leftarrow valor$

retorna $valor$

Fin

Programación Dinámica con Recursión

Ejemplo: Mochila 0/1

Datos: n objetos, vector de pesos: $p(1..n)$, vector de beneficio: $b(1..n)$, peso máximo mochila: M

Objetivo: *llenar la mochila de tal manera que se maximice el beneficio de los objetos transportados, respetando la limitación de la capacidad impuesta.*

Solución: vector X

$x_i = 0$ si el objeto i no se carga en la mochila

$x_i = 1$ si el objeto i si se carga en la mochila

Maximizar la cantidad: $\sum_{i=1}^n b_i x_i$ Restricción: $\sum_{i=1}^n p_i x_i \leq M$

Programación Dinámica con Recursión

Ejemplo: Mochila 0/1

- Para resolver este problema con programación dinámica se prepara una tabla $V(1..n, 0..M)$ que tiene una *fila* por cada *objeto* disponible (desde 0 a n) y una *columna* para cada *peso* (desde 0 a M).

Objeto / Limite de peso	0	1	2	3	j	...	M
Objeto 1: p_1, b_1							
Objeto 2: p_2, b_2							
<u>Objeto i</u>					<u>$V(i,j)$</u>		
...							
Objeto n: p_n, b_n							<u>$V(n,M)$</u>

$$V(i,j) = \text{Maximo}(V(i-1, j), V(i-1, j-p_i)+b_i)$$

Programación Dinámica con Recursión

Ejemplo: Mochila 0/1

Para poder aplicar la *recursión* de manera eficiente se usa un almacenamiento en tabla.

La función **m-tabla** usa como variables globales de Entrada:

Vector de pesos: $p(i)$, $i=1,n$

Vector de beneficios: $b(i)$, $i=1,n$

Una matriz auxiliar: **tabla(1..n,1..M)** inicializada en:

$tabla(i,j) \leftarrow 0$, primera fila y primera columna

$tabla(i,j) \leftarrow -1$, $i=1,n$, $j=1,M$

La función **m-tabla(i,j)** retorna como salida: el mayor beneficio posible cargando peso máximo j y usando objetos desde $1..i$

Para resolver el problema planteado se invoca con **m-tabla(n,M)**.

Programación Dinámica con Recursión

Ejemplo: Mochila 0/1

Función **m-tabla**(i, j):entero ≥ 1 x entero $\geq 1 \rightarrow$ entero ≥ 0

Si $tabla(i, j) < 0$ entonces // solo se calcula si no esta almacenado

 Si $j < p(i)$ entonces

$valor \leftarrow \text{m-tabla}(i-1, j)$

 sino

$valor \leftarrow \text{maximo} (\text{m-tabla}(i-1, j), b(i) + \text{m-tabla}(i-1, j-p(i)))$

$tabla(i, j) \leftarrow valor$

retorna $tabla(i, j)$

Fin

Costo del algoritmo: Para construir la tabla se necesita $O(n \cdot M)$ en almacenamiento y en tiempo de ejecución. Además se usa memoria para el stack de recursión.

Programación Dinámica con Recursión

Ejemplo: Mochila 0/1

n=4 objetos: $p_i=(2,1,3,2)$ $b_i=(12,10,20,15)$ $M=5$

Tabla (0..4,0..5)

Objeto / peso	0	1	2	3	4	5
	0	0	0	0	0	0
Objeto 1: p1=2 b1=12	0	0	12	12	12	12
Objeto 2: p2=1 b2=10	0	-	12	22	-	22
Objeto 3: p3=3 b3=20	0	-	-	22	-	32
Objeto 4: p4=2 b4=15	0	-	-	-	-	37

$$tabla(i,j) \leftarrow \text{maximo} (\text{m-tabla}(i-1,j), b(i) + \text{m-tabla}(i-1,j-p(i)))$$

Se calculan solo 11 de los 20 valores no triviales de la tabla

El valor de $tabla(1,2)$ no se recalcula, sino que se devuelve el almacenado.

Programación Dinámica



Trabajo Práctico no. 6