estudios de postgrado
en computación

Análisis y Diseño de Algoritmos (AyDA)

Isabel Besembel Carrera

TÉCNICAS DE DISEÑO PROGRAMACIÓN DINÁMICA

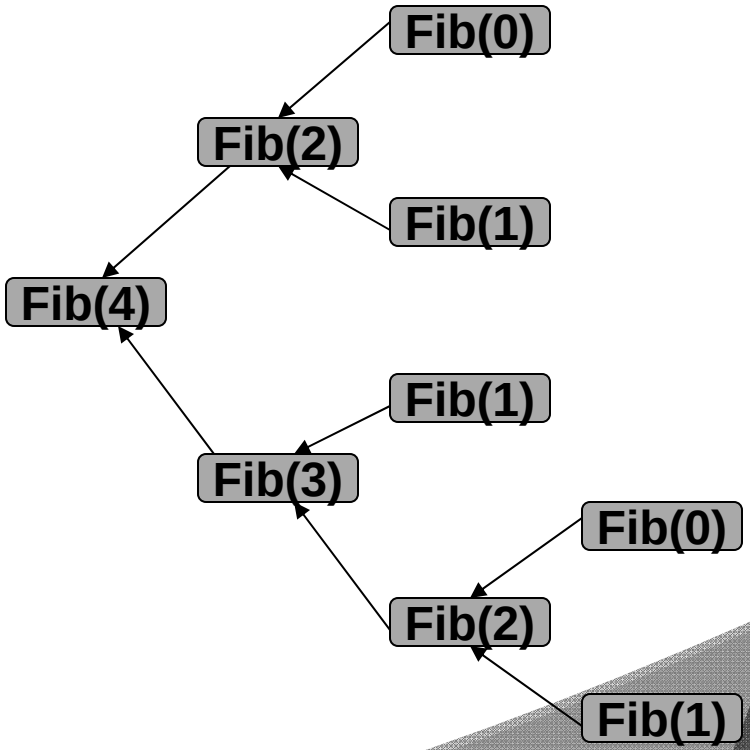
Estrategia 5

- Programación dinámica
  - Técnica utilizada para eliminar el trabajo redundante almacenando en un área de memoria especial las soluciones encontradas anteriormente
- Ejemplo 1:
  - Encontrar el n-ésimo número de la secuencia de Fibonacci

$$F(n) = \begin{cases} 0 \text{ si } n = 0 \\ 1 \text{ si } n = 1 \\ F(n-1) + F(n-2) \text{ si } n > 1 \end{cases}$$

Estrategia 5

- Ejemplo 1
- La solución obvia es utilizar la estrategia divide-y-vencerás, pero ella tiene una complejidad en tiempo del tipo exponencial
- El árbol de las llamadas para el caso n = 4



Estrategia 5

- La mejora del algoritmo se realiza con la programación dinámica, obteniéndose una complejidad O(n)
- La programación dinámica es una técnica de optimización de código útil, aunque no reduzca la complejidad asintótica

Junio,04	**fibonacci(Entero: n):Entero**	
{ pre: n ≥ 0 }		**{ pos: número de Fibonacci ≥ 0 }**
1 2 3	**v(0), v(1) = 0, 1** **[v(j) = v(j - 1) + v(j - 2)] j = 2, n** **regrese v(n)**	**-v: Arreglo(0..n)de Entero. Vector auxiliar que almacena los valores intermedios anteriores** **-j: Entero. Subíndice de v**
1 2	n=4, v=(0,1,1,2,3), v(4)=3 n=0, v=(0,1), v(0)=0	Caso exitoso Caso exitoso

Estrategia 5

Ejemplo 2. Multiplicación encadenada de matrices

- Dada una colección de n matrices 2D, calcular

$$A = A_0 . A_1 . A_2 \ldots . A_{n-1}$$

- Donde A_i es una matriz d_i x d_{i+1}, para i=0,1,…,n-1

$$\sum_{k=0}^{e-1} B[I,k] . C[k,j]$$

- Multiplicación de matrices es asociativa
  - B(2x10), C(10x50) y D(50x20)
  - B . (C . D) requiere 2x10x20+10x50x20=10400 multiplicaciones
  - (B . C) . D requiere 2x10x50+2x50x20=3000 multiplicaciones

Algoritmo de multiplicación de matrices

Sep,09 **multiplicacionMatrices(Arreglo[n,m]: A, Arreglo[p,q]: B):Arreglo[n,q]** **{ pre: m = p }** **{ pos: C=AxB }**		
1	**[[C(i,j) = 0** **[C(i, j) = C(i, j) + A(i, k) . B(k, j)] k = 1, m** **] j = 1, q** **] i = 1, n**	-C: Arreglo[n, q] . Matriz de 2 dimensiones resultado de la multiplicación de A por B. -j, i, k: Natural. Subíndices de columnas de B, filas de A y columnas de A o filas de B
1 2	Ejercicio…. Colocar los casos de prueba	

$O(n^3)$

Problema de la multiplicación encadenada de matrices

- E0: Determinar la parentación de la expresión que define el producto A que minimiza el número total de multiplicaciones escalares a realizar.
- El algoritmo de la fuerza bruta es exponencial igual a n-ésimo número Catalan $\Omega(4^n/n^{3/2})$
- Mejora:
  - El problema se puede dividir en subproblemas
  - Ni,j denota el mínimo número de multiplicaciones necesarias para calcular la subexpresión $A_i \cdot A_{i+1} \ldots A_j$

Problema de la multiplicación encadenada de matrices

- E1: E0 se transforma en calcular el valor de $N_{0,n-1}$
- Condición de optimalidad del subproblema: Es posible caracterizar una solución optima a un subproblema particular en términos de soluciones optimas a sus subproblemas
- Solución optima al subproblema Ni,j

$N_{i,j}=\min_{i\leq k<j}\{N_{i,k}+N_{k+1,j}+d_i d_{k+1} d_{j+1}\}$, donde $N_{i,i}=0$

Problema de la multiplicación encadenada de matrices

- En apariencia similar a divide-y-vencerás por la subdivisión en subproblemas, solo que ellos no son independientes
- Compartición de subproblemas hace que se deba construir la solución de abajo-hacia-arriba, almacenando las soluciones intermedias en una tabla con los $N_{i,j}$ ya calculados
- Inicio: $N_{i,i} = 0$ para $i = 0, 1, \ldots, n-1$

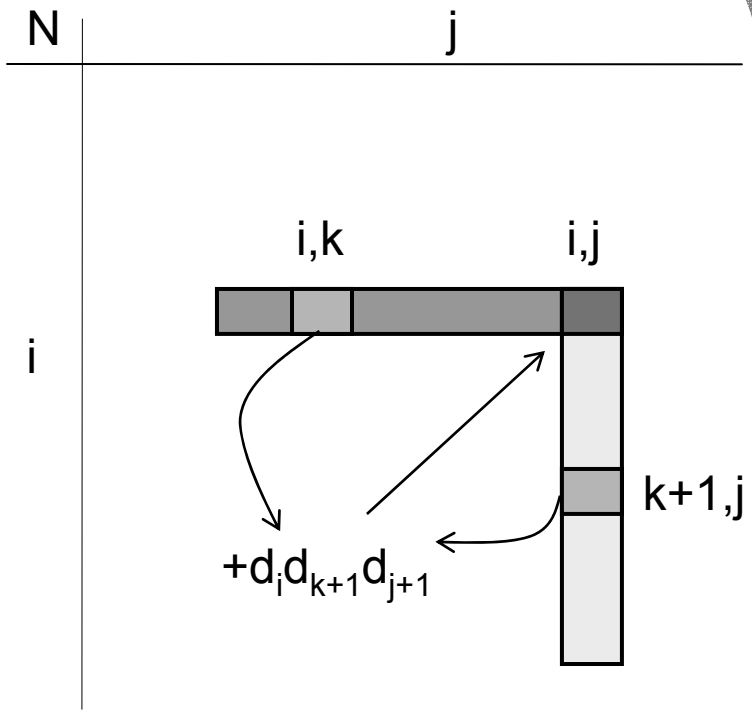
Algoritmo del orden de la multiplicación encadenada de matrices

Sep,09 **ordenMatricesEncadenadas(SecuenciaDeEntero: d, Natural: n):Entero** **{ pre: n ≥ 0 }** **{ pos: Ni,j ≥ 0 }**		
1 2	**[N(i,i)=0] i = 0, n-1** **[[j, N(i,j) = i+b, +∞** **[N(i,j)=min{N(i,j), N(i,k)+N(k+1,j)+d(i)d(k+1)d(j+1)}] k = i, j-1** **] i = 0, n-b-1** **] b = 1, n-1** N(i,j).sm=k	-d: Arreglo(0..n) de Entero. Secuencia -j, i, k, b: Entero. Subíndices de d y N
1 2	Ejercicio…. Colocar los casos de prueba	

- Calculo de $N_{0,n-1}$ con un algoritmo con 3 lazos anidados, $T(n)=O(n^3)$
- Teorema: Dado un producto encadenado de n matrices 2D, se puede calcular la parentación de tal cadena que arroja el mínimo número de multiplicaciones escalares en $O(n^3)$

Problema de la multiplicación encadenada de matrices

- Cómo saber la secuencia de parentización?
- Colocando el valor de k al encontrar el mínimo asociado al $N_{i,j}$



Multiplicación encadenada de matrices

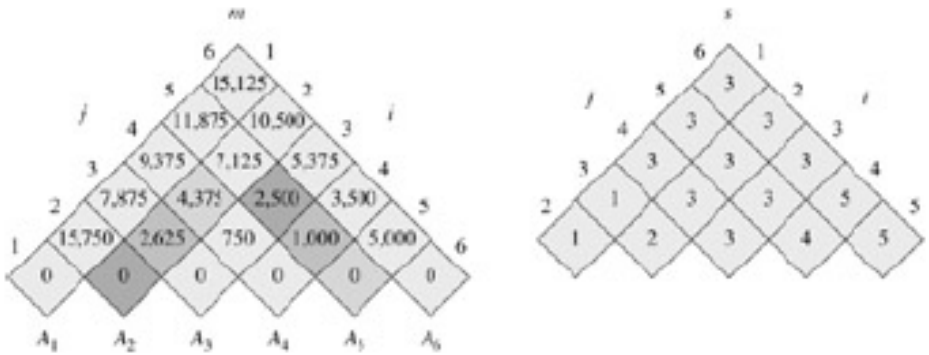


Figure 15.3: The m and s tables computed by MATRIX-CHAIN-ORDER for $n = 6$ and the following matrix dimensions:

matrix	dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

```
MATRIX-CHAIN-ORDER(p)
 1 n ← length[p] - 1
 2 for i ← 1 to n
 3      do m[i, i] ← 0
 4 for l ← 2 to n       ▸l is the chain length.
 5      do for i ← 1 to n - l + 1
 6             do j ← i + l - 1
 7                m[i, j] ← ∞
 8                for k ← i to j - 1
 9                    do q ← m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j
10                       if q < m[i, j]
11                          then m[i, j] ← q
12                               s[i, j] ← k
13 return m and s
```

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \end{cases}$$
$$= 7125\,.$$

Libro texto. 2da. Ed.

Multiplicación encadenada de matrices

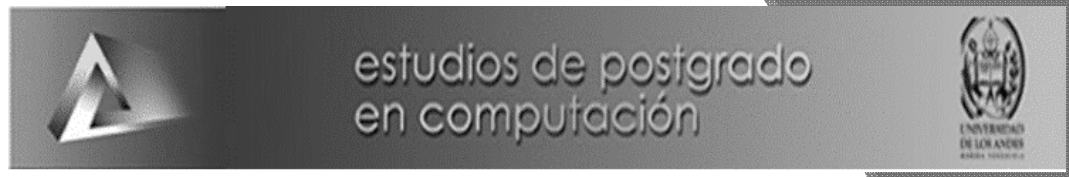
```
PRINT-OPTIMAL-PARENS(s, i, j)
1 if i = j
2     then print "A"i
3     else print "("
4          PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5          PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6          print ")"
```

Sep,09 **multiMatricesEncadenadas(Arreglo[n]De Arreglo[n, m]: A, Arreglo[n,n]: s, Natural: i, Natural: j):Arreglo[n,n]** **{ pre: |A| > 0 }**		**{ pos: |A| > 0 }**
1	**Si (i < j) entonces** **X = multiMatricesEncadenadas(A, s, i, s(i, j))** **Y = multiMatricesEncadenadas(A, s, s(i, j)+1, j)** **regrese multiplicacionMatrices(X, Y)** **sino** **regrese A_i** **fsi**	-X, Y: Arreglo[n, q] . Matriz de 2 dimensiones. -j, i: Natural. Subíndices de la matriz de parentación s
1 2	Ejercicio…. Colocar los casos de prueba	

Estrategia 5

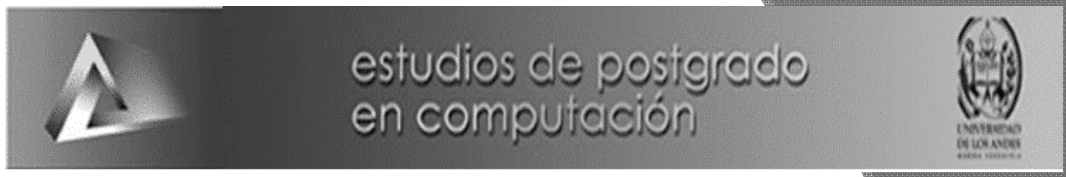
Programación dinámica

- Se utiliza principalmente en problemas de optimización
- Componentes:
  - **Subproblemas simples:** debe haber una forma de dividir el problema global en subproblemas con estructura similar al problema original
  - **Optimalidad de los subproblemas:** la solución optima del problema global debe ser una composición de las soluciones optimas de los subproblemas
  - **Solapamiento de los subproblemas**: soluciones optimas de subproblemas no relacionados pueden contener subproblemas en común, lo cual mejora la eficiencia



Ejemplo 3. Problema de la mochila

- Suponga un alpinista con su morral que conoce el peso máximo total W que puede cargar y con un conjunto S de items útiles que puede potencialmente cargar.
- Cada item tiene un peso $w_i \in$ Natural y un valor de beneficio b_i, que es el valor que el alpinista asigna al item I
- Problema: optimizar el valor total del conjunto T de items sin sobrepasar W
- Función objetivo: maximizar $\sum_{i \in T} b_i$ con $\sum_{i \in T} w_i \leq W$

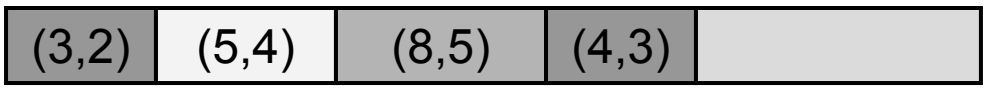


Problema de la mochila (0-1)

- Cada item se escoge o se rechaza completo
- Uso en el contexto de las subastas por Internet
- Método de la fuerza bruta: $O(2^n)$, enumerando todos los subconjuntos de S y seleccionando uno con el mayor beneficio entre los que no sobrepasen W
- Método con programación dinámica:
  - Enumerar los items en S como 1, 2, …, n y definir $\forall k \in \{1, 2, \ldots, n\}$ el subconjunto S_k={items $\in$ S etiquetados 1, 2, …, k}

Problema de la mochila (0-1)

- Cada subproblema k es la mejor manera de llenar el morral usando solo los items de S_k, pero no es claro como definir la solución optima de k en función de las soluciones optimas de los subproblemas
- Ejemplo: S={(3,2), (5,4), (8,5), (4,3), (10,9)} (peso, beneficio) con W=20



Problema de la mochila (0-1)

- Reformulación del problema: se agrega un parámetro w, donde B[k, w] máximo valor de S_k entre aquellos subconjuntos que tienen peso total exactamente w.
- B[0, w] = 0 $\forall w \leq W$ y en general se tiene

  B[k,w]= B[k-1, w] si $w_k > w$

  max{B[k-1, w], B[k-1, w-w_k]+b_k} en caso contrario
- Lo cual satisface la condición de optimización de los subproblemas y además el solapamiento de los mismos

Algoritmo del problema de la mochila 0-1

Sep,09	**mochila01(Conjunto: S, Natural: W)** **{ pre: |S| ≥ 0 }**	**{ pos: |B| ≥ 0 }**
1 2	**[B(w)=0] w = 0, W** **[[si (B(w-w_k)+b_k > B(w)) entonces** **B(w) = B(w-w_k)+b_k** **fsi** **] w = W, w_k, -1** **] k = 1, n**	-S: Conjunto de pares (b_i, w_i) -w, k: Natural. Subíndices de los items -B: ArregloDe Natural. Beneficio máximo de un subconjunto de S con peso total w
1 2	Ejercicio…. Colocar los casos de prueba	

- Algoritmo con 2 lazos anidados, el primero itera hasta n y el interno itera a lo sumo W
- Teorema: Dado W ∈ Natural y S con n items, cada uno con un beneficio y un peso ∈ Natural, se puede encontrar el beneficio más alto en el subconjunto de S con un peso total a lo sumo W en O(nW)

Algoritmo de tiempo pseudo-polinomial

- La complejidad depende de W, que no es proporcional al tamaño de la instancia del problema
- Si W es muy grande (2^n), entonces el algoritmo es asintóticamente mas lento que la fuerza bruta
- No es un algoritmo de tiempo polinómico, ya que no depende del tamaño del problema
- Se considera de tiempo pseudo-polinómico