OBLIGATORIO
para mantener la condición de estudiante

UBA
Universidad de Buenos Aires

¡Recordá que tenés que censarte!

SIP - Sistema de Información Permanente

CENSO Y REMATRICULACIÓN
ESTUDIANTES 2015

- desde el 6 de Agosto hasta el 6 de Septiembre
- exclusivamente por internet

Algoritmos y Estructuras de Datos III

Segundo cuatrimestre 2015

Técnicas de diseño de algoritmos (segunda parte)

Programación dinámica

- Al igual que “dividir y conquistar”, el problema es dividido en subproblemas de tamaños menores que son más fácil de resolver. Una vez resueltos esto subproblemas, se combinan las soluciones obtenidas para generar la solución del problema original.
- Es bottom up y no es recursivo.
- Se guardan las soluciones de los subproblemas para no calcularlos más de una vez.

Programación dinámica

- **Principio de optimalidad de Bellman**:
  Un problema de optimización satisface el principio de optimalidad de Bellman si en una sucesión óptima de decisiones, cada subsucesión es a su vez óptima.
- Es decir, si miramos una subsolución de la solución óptima, debe ser solución del subproblema asociado a esa subsolución.
- El principio de optimalidad es condición necesaria para poder usar programación dinámica.
  - Se cumple en camino mínimo (sin distancias negativas).
  - No se cumple en camino máximo.

Ejemplo: Cálculo de coeficientes binomiales

- **Definición.** Si $n \geq 0$ y $0 \leq k \leq n$, definimos

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

- Algoritmo directo: Calcular $a := n!$, $b := k!$ y $c := (n-k)!$, y retornar a/bc. ¿Es un algoritmo efectivo? **No!** ¿Por qué?

- **Teorema.** Si $n \geq 0$ y $0 \leq k \leq n$, entonces

$$\binom{n}{k} = \begin{cases} 1 & \text{si } k = 0 \text{ o } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{si } 0 < k < n \end{cases}$$

Ejemplo: Cálculo de coeficientes binomiales

	0	1	2	3	4	$\cdots$	$k-1$	k
0	1							
1	1	1						
2	1	2	1					
3	1	3	3	1				
4	1	4	6	4	1			
$\vdots$	$\vdots$					$\ddots$		
$k-1$	1						1	
k	1							1
$\vdots$	$\vdots$							
$n-1$	1							
n	1							

Ejemplo: Cálculo de coeficientes binomiales

algoritmo *combinatorio*(*n*,*k*)

entrada: dos enteros n y k

salida: $\binom{n}{k}$

para $i = 1$ **hasta** n **hacer**
  $A[i][0] \leftarrow 1$
fin para
para $j = 0$ **hasta** k **hacer**
  $A[j][j] \leftarrow 1$
fin para
para $i = 2$ **hasta** n **hacer**
  para $j = 2$ **hasta** $\texttt{min}(i-1,\texttt{k})$ **hacer**
    $A[i][j] \leftarrow A[i-1][j-1] + A[i-1][j]$
  fin para
fin para
retornar $A[n][k]$

Ejemplo: Cálculo de coeficientes binomiales

- Función recursiva:
  - Complejidad $\Omega(\binom{n}{k})$.
- Programación dinámica:
  - Complejidad $O(nk)$.
  - Espacio $\Theta(k)$: sólo necesitamos almacenar la fila anterior a la que estamos calculando.

Recordemos: El problema de la mochila (0/1)

Datos de entrada:

- Capacidad $C \in \mathbb{Z}_+$ de la mochila (peso máximo).
- Cantidad $n \in \mathbb{Z}_+$ de objetos.
- Peso $p_i \in \mathbb{Z}_+$ del objeto i, para $i = 1, \ldots, n$.
- Beneficio $b_i \in \mathbb{Z}_+$ del objeto i, para $i = 1, \ldots, n$.

Problema: Determinar qué objetos debemos incluir en la mochila sin excedernos del peso máximo C, de modo tal de maximizar el beneficio total entre los objetos seleccionados.

Recordemos: El problema de la mochila

- Definimos $m(k, D)$ = valor óptimo del problema con los primeros k objetos y una mochila de capacidad D.
- Podemos representar los valores de este parámetro en una tabla de dos dimensiones:

m	0	1	2	3	4	...	C
0	0	0	0	0	0	...	0
1	0						
2	0						
3	0						
4	0				$m(k, D)$		
$\vdots$	$\vdots$						
n	0						$m(n, C)$

Recordemos: El problema de la mochila

- Sea $S^* \subseteq \{1, \ldots, k\}$ una solución óptima para la instancia (k, D).
- $$m(k, D) = \begin{cases} 0 & \text{si } k = 0 \\ 0 & \text{si } D = 0 \\ m(k-1, D) & \text{si } k \notin S^* \\ b_k + m(k-1, D - p_k) & \text{si } k \in S^* \end{cases}$$
- Tenemos entonces que:
  1. $m(k, D) = 0$, si $k = 0$ o $D = 0$.
  2. $m(k, D) = m(k-1, D)$, si $p_k > D$.
  3. $m(k, D) = \max\{m(k-1, D), b_k + m(k-1, D - p_k)\}$, en caso contrario.

Multiplicación de n matrices

$$M = M_1 \times M_2 \times \ldots M_n$$

Por la propiedad asociativa del producto de matrices esto puede hacerse de muchas formas. Queremos determinar la que minimiza el número de operaciones necesarias. Por ejemplo, si A es de 13×5, B de 5×89, C de 89×3 y D de 3×34, entonces

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

Multiplicación de n matrices

- La mejor forma de multiplicar todas las matrices es multiplicar las matrices 1 a i por un lado y las matrices $i+1$ a n por otro lado y luego multiplicar estos dos resultados para algún $1 \leq i \leq n-1$.

- En la solución óptima de $M = M_1 \times M_2 \times \ldots M_n$, estos dos subproblemas, $M_1 \times M_2 \times \ldots M_i$ y $M_{i+1} \times M_{i+2} \times \ldots M_n$ deben estar resueltos de forma óptima: se cumple el principio de optimalidad.

- Llamamos m_{ij} a la cantidad mínima de multiplicaciones necesarias para calcular $M_i \times M_{i+1} \times \ldots M_j$.

Multiplicación de n matrices

Suponemos que las dimensiones de las matrices están dadas por un vector $d \in \mathbb{N}^{n+1}$, tal que la matriz M_i tiene d_{i-1} filas y d_i columnas para $1 \leq i \leq n$. Entonces:

- Para $i = 1, 2, \ldots, n$, $m_{ii} = 0$
- Para $i = 1, 2, \ldots, n-1$, $m_{i,i+1} = d_{i-1} d_i d_{i+1}$
- Para $s = 2, \ldots, n-1$, $i = 1, 2, \ldots, n-s$,

$$m_{i,i+s} = \min_{i \leq k < i+s} (m_{ik} + m_{k+1,i+s} + d_{i-1} d_k d_{i+s})$$

La solución del problema es m_{1n}.

Algoritmos probabilísticos

- Algoritmos numéricos probabilísticos: Algoritmos basados en simulaciones, que encuentran una solución aproximada para problemas numéricos.
  - A mayor tiempo de proceso, mayor precisión en la respuesta.
  - Ejemplo: Cálculo de π arrojando dardos virtuales a un círculo inscripto en un cuadrado.
- Algoritmos de Montecarlo: Algoritmos que siempre proporcionan una respuesta, pero que puede no ser la correcta. La respuesta es correcta con alta probabilidad.
  - A mayor tiempo de proceso, mayor probabilidad de dar una respuesta correcta.
  - Ejemplo: determinar la existencia en un arreglo de un elemento mayor a un valor dado.

Algoritmos probabilísticos

- Algoritmos de Las Vegas: Algoritmos que si dan una respuesta entonces es correcta, pero pueden no dar ninguna respuesta.
  - A mayor tiempo de proceso, mayor probabilidad de obtener respuesta.
  - Ejemplo: Problema de las n reinas.
- Algoritmos de Sherwood: Algoritmos que "aleatorizan" un algoritmo determinístico donde hay una gran diferencia entre el peor caso y caso promedio.
  - Se espera que la aleatorización permita estar en un caso promedio con alta probabilidad, evitando así los peores casos.
  - Ejemplo: Quicksort con pivote seleccionado aleatoriamente.