Análisis y Diseño de Algoritmos.

Sesión 10. 28 de Octubre de 2015.

Maestría en Sistemas Computacionales.

Por: Hugo Iván Piza Dávila.

¿Qué veremos hoy?

- Programación Dinámica
  - Cálculo del Coeficiente Binomial
  - Problema del Cambio
  - Principio de Optimalidad
  - Problema de la Mochila

Coeficiente binomial

- Sabemos que la siguiente ecuación calcula el número de subconjuntos de *k* elementos que se pueden construir a partir de un conjunto de *n* elementos, también llamado *coeficiente binomial*:

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

- El mismo valor se puede calcular de manera implícita así:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Coeficiente binomial

- Consideremos el siguiente ejemplo ilustrativo:

$$\binom{4}{3} = \frac{4!}{3!\,1!} = 4 = \binom{3}{2} + \binom{3}{3}$$

- Los subconjuntos que se pueden formar:

1. {a, b, d}
2. {a, c, d}
3. {b, c, d}
4. {a, b, c}

Los primeros tres subconjuntos son los que se forman con $n = 3$, $k = 2$, si omitimos al 4° elemento d. El último subconjunto no utiliza al 4° elemento, y es el que se forma con n = 3, k = 3.

Calculando el coeficiente binomial

- Podemos diseñar un algoritmo muy expresivo que, dados los valores de n, k, calcula el coeficiente binomial de acuerdo a la siguiente definición recursiva, inspirada en la técnica de *divide y vencerás*:

$$\binom{n}{k} = \begin{cases} 1 & k = 0 \text{ ó } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 0 & en\ otro\ caso \end{cases}$$

Calculando el coeficiente binomial

- Si ejecutamos la función recursiva con n = 4, k = 2, realizaríamos las siguientes llamadas:
  - (4,2)
  - (3,1), (3,2)
  - (2,0),(2,1) (2,1), (2,2)
  - (1,0),(1,1) (1,0),(1,1)
- Obsérvese que la llamada (n = 2, k = 1) se efectuó dos veces, lo que implicó duplicación de toda la recursión con raíz (2, 1).
- En total se realizan 11 llamadas. Sin repetición serían 8.

Calculando el coeficiente binomial

N	K = $^1/_2$N	# de subconjuntos	Llamadas
2	1	2	3
4	2	6	11
6	3	20	39
8	4	70	139
10	5	252	503
12	6	924	1847
14	7	3432	6863
16	8	12,870	25,739
18	9	48,620	97,239
20	10	184,756	369,511
30	15	155’117,520	310’235,039

Calculando el coeficiente binomial

- De acuerdo a la tabla anterior, podemos concluir que la complejidad temporal del algoritmo recursivo que calcula el coeficiente binomial se puede expresar así:

$$t(n,k) = 2\binom{n}{k} - 1 \in \Omega\binom{n}{k}$$

- El crecimiento acelerado en el número de llamadas recursivas hace inviable el uso de este algoritmo.
- Este fenómeno se presenta en la versión recursiva de *Fibonacci*.

Desventaja de *Divide y vencerás*

- Es una técnica *top-down*.
  - Primero ataca la instancia completa.
  - Esta se divide en instancias más pequeñas.
  - Conforme el algoritmo avanza, cada instancias se sigue dividiendo en instancias más pequeñas hasta llegar a la trivialidad.
- En muchos problemas, esta técnica nos lleva a instancias que se traslapan → algoritmos ineficientes.
- Si aprovechamos la división en problemas más pequeños pero nos cercioramos de que no haya instancias repetidas podemos llegar a un algoritmo muy eficiente.

Programación Dinámica (PD)

- Es una técnica *bottom-up*.
  - Comienza con las instancias más pequeñas (simples, triviales).
  - Combina las soluciones de instancias pequeñas para obtener la solución de una instancia de tamaño cada vez mayor.
  - Al final llegamos a la solución de la instancia original.
  - Adecuado cuando los problemas pequeños son **reutilizables**.
- La combinación de las soluciones implica registrar las soluciones de las instancias pequeñas.
  - Utiliza arreglos de *D* dimensiones, donde *D* es el número de argumentos necesarios para crear una instancia.

Coeficiente Binomial con PD

- Cada instancia de coeficiente binomial utiliza dos argumentos: N, K.
  - Esto sugiere el uso de una matriz de N x K
- En muchos casos, es conveniente contemplar los casos base de cada argumento (K = 0, N = 0).
  - Por tanto, la matriz será de (N + 1) x (K + 1)
- La celda [n][k] deberá almacenar el coeficiente binomial obtenido con los argumentos (N = n, K = k).

Coeficiente Binomial con PD

- Matriz para N = 4, K = 2, versión 1:

N K	0	1	2
0	1		
1	1	1	
2	1		1
3	1		
4	1		Solución

$$\binom{n}{k} = \begin{cases} \mathbf{1} & \boldsymbol{k = 0 \text{ ó } k = n} \\ \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 0 & \textit{en otro caso} \end{cases}$$

Coeficiente Binomial con PD

- Matriz para N = 4, K = 2, versión 2:

N K	0	1	2
0	1	0	0
1	1	1	0
2	1		1
3	1		
4	1		Solución

$$\binom{n}{k} = \begin{cases} 1 & k = 0 \text{ ó } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ \mathbf{0} & \textit{en otro caso} \end{cases}$$

Coeficiente Binomial con PD

- Matriz para N = 4, K = 2, versión 3:

N K	0	1	2
0	1	0	0
1	1	1	0
2	1	**2**	1
3	1	**3**	**3**
4	1	**4**	6

$$\binom{n}{k} = \begin{cases} 1 & k = 0 \text{ ó } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 0 & \textit{en otro caso} \end{cases}$$

Se formó el triángulo de Pascal

Coeficiente Binomial con PD

- Es fácil saber que la complejidad temporal y espacial del algoritmo que calcula el Coeficiente Binomial utilizando Programación Dinámica son proporcionales a N x K.
- ¿Cuál es la complejidad espacial de la versión *divide y vencerás*?
  - No se usa una matriz pero se tienen que almacenar (en una pila) los argumentos de todas las llamadas recursivas pendientes.

Problema del Cambio

- Entregar la mínima cantidad de monedas que sumen el cambio solicitado.
- Tipos de monedas disponibles = {1, 4, 6, …}
- Sabemos que el algoritmo voraz no siempre dará la solución correcta:
  - Cambio = 9. Solución = {6, 1, 1, 1}.
  - Solución correcta = {4, 4, 1}
- Hay que construir una tabla con resultados intermedios (instancias más pequeñas del problema), suponiendo menos cambio y menos tipos de monedas.

Problema del Cambio

- El diseño de la matriz considera:
  - Cambio de $0.00 a C = $9.00 en intervalos de 1
  - Cada denominación de moneda disponible. Supone que la primera es de $1.00 para garantizar que existe una solución.

M C	0	1	2	3	4	5	6	7	8	9
1										
4										
6										

Problema del Cambio

- Si el cambio es $0.00, no se entregan monedas.

M C	0	1	2	3	4	5	6	7	8	9
1	0									
4	0									
6	0									

Problema del Cambio

- Si sólo se cuenta con monedas de $1.00, el número de monedas será igual al cambio solicitado.

M C	0	1	2	3	4	5	6	7	8	9
1	0	1	2	3	4	5	6	7	8	9
4	0									
6	0									

Problema del Cambio

- Si el cambio es menor que la denominación de la moneda, la solución estará en la fila anterior.

M C	0	1	2	3	4	5	6	7	8	9
1	0	1	2	3	4	5	6	7	8	9
4	0	1	2	3	x	y				
6	0	1	2	3	x	y				

Problema del Cambio

- En otro caso, evaluar el resultado que se obtiene al utilizar una moneda de denominación mayor.
- ¿Qué implica menos monedas?
  - No usar la moneda de denominación mayor: nos quedamos con el resultado de la fila anterior
  - Sí usarla. ¿Cómo sabemos?
    - Si al cambio le restamos el valor de la denominación actual, nos moveremos a una celda que ya tiene el resultado final ☺.
    - Al resultado obtenido le sumamos 1 moneda más: la moneda de mayor denominación.

Problema del Cambio

- Si el cambio a entregar es $6.00, ¿qué es mejor?
  - El resultado usando monedas de menor denominación (6).
  - Restar al cambio la denominación actual (6 – 4 = 2), y sumar uno al resultado de la columna 1 (2 + 1 = 3).

M C	0	1	2	3	4	5	6	7	8	9
1	0	1	2	3	4	5	6	7	8	9
4	0	1	2	3	1	2	3			
6	0	1	2	3	1	2				

Problema del Cambio

- Sólo en las últimas dos celdas fue mejor el resultado obtenido con la denominación anterior.
- Se necesitan mínimo tres monedas para dar $9.00 de cambio con las denominaciones {1, 4, 6}.

M C	0	1	2	3	4	5	6	7	8	9
1	0	1	2	3	4	5	6	7	8	9
4	0	1	2	3	1	2	3	4	2	3
6	0	1	2	3	1	2	1	2	2	**3**

Problema del Cambio

- ¿Y cómo sabemos cuáles monedas fueron elegidas?
  1. Comenzamos en la celda donde está el resultado (m = 6, c = 9)
  2. Si tiene el mismo valor que la fila anterior, nos subimos.
  3. Si no, incrementamos el número de monedas del valor actual y restamos al cambio el valor de la moneda.
  4. Regresamos al paso 2 con la celda correspondiente a los valores actuales de moneda y cambio mientras no lleguemos a la primer fila.
  5. El número de monedas de $1.00 estará indicada en la primer fila.

M C	0	1	2	3	4	5	6	7	8	9	#
1	0	1	2	3	4	5	6	7	8	9	1
4	0	1	2	3	1	2	3	4	2	3	2
6	0	1	2	3	1	2	1	2	2	3	0

Principio de Optimalidad

- En una secuencia óptima de decisiones, cada sub-secuencia debe ser también óptima.
- De manera natural, supusimos que se cumple esta regla en la solución del Problema del Cambio
  - Al calcular el valor de la celda [n, c] como el menor entre las celdas [n – 1,c] y 1 + [n, c – d_n], supusimos que dichas celdas representan la forma óptima de resolver el problema con los argumentos que representan.
  - Aunque sólo nos interesa la celda [N, C], damos por hecho que todas las celdas deben representar decisiones óptimas.

Problema de la Mochila

- Puede ser resuelto de forma óptima sin partir objetos con Programación Dinámica.
- Cada fila representa un objeto y cada columna un valor de peso: de 0 a la capacidad de la mochila.
- Para solucionarse con Programación Dinámica, los pesos deben *discretizarse*. Por ejemplo, definirse con enteros.
- La celda (j, k) almacenará el máximo valor obtenido con los primeros j+1 objetos y con una mochila de capacidad k.

Problema de la Mochila

- Consideremos una mochila con 11 kg de capacidad y los siguientes objetos:

Valor	Peso
$1.00	1
$6.00	2
$18.00	5
$22.00	6
$28.00	7

Problema de la Mochila

- No podemos guardar ningún objeto en una mochila con 0 kilos de capacidad.

O P	0	1	2	3	4	5	6	7	8	9	10	11
0 (1k)	0											
1 (2k)	0											
2 (5k)	0											
3 (6k)	0											
4 (7k)	0											

Problema de la Mochila

- Si el peso del primer objeto no supera la capacidad actual de la mochila, el valor obtenido es el del primer objeto.

O P	0	1	2	3	4	5	6	7	8	9	10	11
0 (1k)	0	1	1	1	1	1	1	1	1	1	1	1
1 (2k)	0											
2 (5k)	0											
3 (6k)	0											
4 (7k)	0											

Problema de la Mochila

- Si el peso del objeto actual supera la capacidad actual de la mochila no lo consideramos: nos quedamos con el valor obtenido con los objetos anteriores.

O P	0	1	2	3	4	5	6	7	8	9	10	11
0 (1k)	0	1	1	1	1	1	1	1	1	1	1	1
1 (2k)	0	1										
2 (5k)	0	1	x	x	x							
3 (6k)	0	1	x	x	x	x						
4 (7k)	0	1	x	x	x	x	x					

Problema de la Mochila

- En otro caso nos preguntamos, ¿qué sería mejor?
  - No considerar al objeto actual.
    - Nos quedamos con el valor de la fila anterior.
  - Sí considerarlo.
    - ¿Qué valor teníamos en la fila anterior si restamos al peso actual el peso del objeto que se pretende considerar?
    - Al valor obtenido sumar el del objeto a considerar.

Problema de la Mochila

- Recordando que el segundo objeto vale $6.00.

O P	0	1	2	3	4	5	6	7	8	9	10	11
0 (1k)	0	1	1	1	1	1	1	1	1	1	1	1
1 (2k)	0	1	6	7	7	7	7	7	7	7	7	7
2 (5k)	0	1	6	7	7							
3 (6k)	0	1	6	7	7	x						
4 (7k)	0	1	6	7	7	x	x					

Problema de la Mochila

- Recordando que el segundo objeto vale $18.00.

O P	0	1	2	3	4	5	6	7	8	9	10	11
0 (1k)	0	1	1	1	1	1	1	1	1	1	1	1
1 (2k)	0	1	6	7	7	7	7	7	7	7	7	7
2 (5k)	0	1	6	7	7	18	19	24	25	25	25	25
3 (6k)	0	1	6	7	7	18						
4 (7k)	0	1	6	7	7	18	x					

Problema de la Mochila

- Recordando que el tercer objeto vale $22.00.

O P	0	1	2	3	4	5	6	7	8	9	10	11
0 (1k)	0	1	1	1	1	1	1	1	1	1	1	1
1 (2k)	0	1	6	7	7	7	7	7	7	7	7	7
2 (5k)	0	1	6	7	7	18	19	24	25	25	25	25
3 (6k)	0	1	6	7	7	18	22	24	28	29	29	40
4 (7k)	0	1	6	7	7	18	22					

Problema de la Mochila

- Recordando que el tercer objeto vale $28.00.
- El máximo valor obtenido es $40.00 (objetos 2, 3).
- La lista de objetos seleccionados se obtiene de manera semejante al problema del cambio.

O P	0	1	2	3	4	5	6	7	8	9	10	11
0 (1k)	0	1	1	1	1	1	1	1	1	1	1	1
1 (2k)	0	1	6	7	7	7	7	7	7	7	7	7
2 (5k)	0	1	6	7	7	18	19	24	25	25	25	25
3 (6k)	0	1	6	7	7	18	22	24	28	29	29	40
4 (7k)	0	1	6	7	7	18	22	28	29	34	35	40