Estructuras de Datos y Algoritmos – IIC2133

I3 - pauta

2 noviembre 2011

1. Tú quieres conducir un auto desde Santiago hasta Pueto Montt. Cuando el estanque de bencina del auto está lleno, te permite viajar n kilómetros. Tú tienes un mapa con las distancias entre estaciones de bencina en la ruta. Tu propósito es hacer el menor número posible de detenciones en el viaje.

a) Muestra que este problema tiene subestructura óptima.

(Suponemos que partimos con el estanque lleno y que las distancias entre pares consecutivos de estaciones de bencina son todas $\leq n$.)

Supongamos que tenemos la solución óptima, y que en esta solución, la primera detención es a los m kilómetros ($m \leq n$); en este punto, que llamaremos R, llenamos el estanque. El resto del recorrido tiene que ser una solución óptima (minimiza el número de detenciones) al problema de viajar desde R (m kilómetros al sur de Santiago) hasta Puerto Montt. La razón es que si no lo fuera, entonces podríamos encontrar otra secuencia de detenciones para viajar desde R hasta Puerto Montt que tuviera menos detenciones, y podríamos usar esta secuencia en el viaje desde Santiago hasta Puerto Montt, después de parar en R, reduciendo el número total de detenciones para este viaje, contradiciendo así la suposición de que la solución que tenemos es óptima.

b) Plantea una solución recursiva.

Dada la propiedad de subestructura óptima, podemos plantear la solución como sigue: Elegir de la mejor manera la próxima estación, y, de allí, encontrar una solución óptima al problema de viajar desde esa estación hasta Puerto Montt.

c) Justifica que en cualquier etapa de la recursión, una de las elecciones óptimas es la elección codiciosa.

La elección codiciosa es recorrer la mayor distancia que podamos antes de parar en una estación de bencina (y allí llenar nuevamente el estanque).

Supongamos que acabamos de reanudar el viaje depués de detenernos para llenar el estanque. Supongamos también que las siguientes estaciones son S, …, T, …, U, en orden de cercanía a nuestra posición actual, y que T es la más lejana que no está a más de n kilómetros de distancia. Y supongamos que en una solución óptima la próxima detención es en la estación S (más cercana que T).

Es fácil ver que en esta solución óptima podemos cambiar la elección de detenernos en S por la de detenernos en T, sin aumentar el número total de detenciones: si en la solución óptima, la próxima detención más allá de T es U, a la cual pudimos llegar desde S, entonces también podemos llegar a U desde T, ya que T está más cerca de U que S; y podemos llegar desde nuestra posición actual a T sin detenernos antes, ya que la distancia a T no es mayor que n kilómetros.

2. Siguiendo con el viaje de Santiago a Puerto Montt, supón que tienes autorización para poner letreros con propaganda de tu negocio en el camino. Los puntos en los que puedes poner los letreros son x_1, x_2, …, x_n, que representan las distancias desde Santiago, y tu modelo de predicción te dice que si pusieras un letrero en el punto x_k ganarías $r_k > 0$. Por otro lado, la autoridad vial exige que no puede haber dos letreros de un mismo negocio a menos de 5 km entre ellos.

Tu problema es determinar donde poner letreros, de manera de maximizar tu ganancia. P.ej., si los puntos son $\{ x_1, x_2, x_3, x_4 \} = \{ 6, 7, 12, 14 \}$ y las ganancias son $\{ r_1, r_2, r_3, r_4 \} = \{ 5, 6, 5, 1 \}$, entonces te conviene poner los letreros en x_1 y x_3 para ganar 10.

a) Muestra que este problema tiene subestructura óptima.

Consideremos una solución óptima. Ésta puede incluir poner un letrero en x_n o no. Si no lo incluye, entonces la solución óptima es la misma que para los puntos x_1, …, x_{n-1}. Si lo incluye, entonces, si sacamos x_n de la solución óptima, los puntos que quedan son una solución óptima para el camino de Santiago hasta 5 km antes de x_n.

b) Plantea una solución recursiva.

Definamos $s(j)$ como el punto más al sur que está a más de 5 km (al norte) de x_j. Es decir, si decidimos que x_j está en la solución, entonces x_1, x_2, …, $x_{s(j)}$ son aún posibilidades válidas, pero $x_{s(j)+1}$, …, x_{j-1} no. Si opt(j) es la ganancia del subconjunto óptimo de sitios entre x_1, …, x_j, entonces, de a):

$\text{opt}(j) = \max\{ r_j + \text{opt}(s(j)), \text{opt}(j-1) \}$

c) Justifica que una buena forma de resolver el problema es mediante programación dinámica.

Al tratar de resolver el problema recursivamente, aplicando la fórmula recursiva anterior de manera "top-down", vamos a tener que resolver un mismo subproblema opt(j) muchas veces. En cambio, podemos resolver la fórmula recursiva de manera "bottom-up," para $j = 2, 3, \ldots n$, sabiendo que $\text{opt}(0) = 0$ y $\text{opt}(1) = 1$.

3. Podemos usar las discrepancias en las tasas de cambio de monedas para transformar una unidad de una moneda en más de una unidad de la misma moneda. P.ej., si un euro compra 1.38 dólares, un dólar compra 500 pesos, y un peso compra 0.0015 euros, entonces, a partir de un euro podemos obtener $1.38 \times 500 \times 0.0015 = 1.035$ euros.

Supón que tienes n monedas $\langle c_1, c_2, \ldots, c_n \rangle$ y una tabla R de $n \times n$ tasas de cambio: una unidad de la moneda c_j compra $R[j][k]$ unidades de la moneda c_k. Queremos determinar si existe o no una secuencia de monedas $\langle c_a, c_b, \ldots, c_m \rangle$ tal que

$R[a][b] \times R[b][c] \times \ldots \times R[m][a] > 1$.

a) Muestra que este problema puede plantearse como un problema de determinar si en un grafo direccional existe un ciclo con costo total negativo. [Ayuda: Recuerda que $x > 1 \Rightarrow 1/x < 1$ y que $\log(ab) = \log(a) + \log(b)$.]

Construimos el siguiente grafo:

- cada vértice corresponde a una moneda;
- el costo de la arista dirigida (c_j, c_k) es $-\log(R[j][k])$.

¿Por qué? Si aplicamos la ayuda a la fórmula de arriba, queda, primero

$R[a][b] \times R[b][c] \times \ldots \times R[m][a] > 1 \Rightarrow 1/(R[a][b] \times R[b][c] \times \ldots \times R[m][a]) < 1.$

y luego,

$\log(1/(R[a][b] \times R[b][c] \times \ldots \times R[m][a])) < 0 \Rightarrow \log(1/ R[a][b]) + \log(1/ R[b][c]) + \ldots + \log(1/ R[m][a]) < 0$

$\Rightarrow -\log R[a][b] - \log R[b][c] - \ldots - \log R[m][a] < 0$

Es decir, si hay una secuencia de monedas como la que buscamos, entonces este grafo tiene un ciclo con costo total negativo.

b) Da un algoritmo para imprimir la secuencia, si existe.

Primero, hay que detectar el ciclo, para lo cual podemos aplicar el algoritmo de Bellman-Ford. Pero hay que asegurarse que partamos de un vértice desde el cual se pueda llegar al ciclo. Una posibilidad es agregar un nuevo vértice v al grafo con aristas direccionales con costo 0 a todos los otros vértices (las monedas). Ahora, en el **`for`** final del algoritmo, simplemente detectamos el primer vértice u cuyo valor d mejore; y de ahí seguimos los valores π hasta regresar a u.

4. Cambiemos levemente la definición de un árbol-B. Definamos el *orden* del árbol como el número máximo de hijos que puede tener un nodo. Entonces, un árbol-B de orden m tiene las siguientes propiedades: la raíz tiene al menos dos hijos, excepto si es una hoja; cada nodo interno (que no sea la raíz) tiene k–1 claves y k referencias a hijos, en que $\lceil m/2 \rceil \leq k \leq m$; cada hoja tiene k–1 claves, en que $\lceil m/2 \rceil \leq k \leq m$. P.ej., en un árbol-B de orden $m = 5$, cada nodo interno tiene entre 2 y 4 claves y entre 3 y 5 hijos, y cada hoja tiene entre 2 y 4 claves.

Supón que comenzamos con un árbol-B de orden 5 vacío.

a) Muestra el árbol después de insertar las claves 8, 14, 2, 15.

b) Muestra el árbol después de insertar las claves 3, 1, 16, 6, 5.

c) Muestra el árbol después de insertar las claves 27, 37, 18, 25, 7, 13, 20.