Programación 3
Año 2022

Práctico 9

Ejercicio 1 (Kleinberg & Tardos, Ex. 6.19)**.** Usted es consultor de un grupo secreto de personas cuyo trabajo consiste en monitorear y analizar señales eléctricas de barcos en la costa Atlántica. Ellos quieren un algoritmo eficiente para el siguiente problema: desenredar un entrelazado de dos señales conocidas.

Sean s, x, y cadenas binarias del mismo largo. Decimos que s es un *entrelazado* de x e y si sus símbolos se pueden particionar en dos secuencias (no necesariamente contiguas), s' y s'', tales que s' es un prefijo de x y s'' es un prefijo de y. Por ejemplo, si $x = 101101101$ e $y = 00100010$, entonces $s = 1\underline{001}\underline{000}11\underline{1}$ es un entrelazado de x e y, ya que la concatenación de los caracteres subrayados es un prefijo de y, y la concatenación de los restantes es un prefijo de x.

(a) Escriba un algoritmo de programación dinámica que, dadas tres cadenas de largo n, x, y, s, determina si s es un entrelazado de x e y.
Sugerencia: Observe que si s es un entrelazado de x e y, cada caracter c de s proviene o bien de x o de y. Defina una recurrencia para una función M tal que, para cada par de naturales i, j, $i+j \leq n$, el valor de $M(i,j)$ determina si $s_1 \dots s_{i+j}$ es un entrelazado de $x_1 \dots x_i$ y $y_1 \dots y_j$.

(b) Escriba un algoritmo que, si efectivamente s es un entrelazado de x e y, devuelve un conjunto de posiciones de s que provienen de x (el complemento necesariamente proviene de y).

Ejercicio 2 (Kleinberg & Tardos, Ex. 6.21)**.** Una compañía de inversiones está analizando el valor de las acciones de una empresa que cotiza en bolsa a lo largo de n días consecutivos. Los días considerados están numerados de 1 a n; para cada día i, $1 \leq i \leq n$, cada acción tiene un precio $p(i)$.

Para algunos valores, posiblemente grandes, de un parámetro k, quieren estudiar lo que llaman estrategias de k-disparos. Una estrategia de k-disparos es una colección de m pares de días $(c_1, v_1), ..., (c_m, v_m)$ donde $0 \leq m \leq k$ y

$$1 \leq c_1 < v_1 < c_2 < v_2 < ... < c_m < v_m \leq n\,. \tag{2.1}$$

Podemos interpretar esta secuencia como un conjunto de m intervalos disjuntos, durante los que los inversores compran 1000 acciones del stock (en el día c_i) y luego las venden (en el día v_i). Definimos entonces la *ganancia* de

una estrategia de k-disparos como la ganancia obtenida en las m transacciones de compra-venta realizadas, es decir

$$\texttt{ganancia}(E) = 1000 \times \sum_{i=1}^{m} \Big(p(v_i) - p(c_i) \Big), \tag{2.2}$$

donde E es una estrategia de k-disparos para un stock de una acción dada.

(a) Diseñe un algoritmo eficiente que, dada una secuencia de precios, determina una estrategia de k-disparos con la mayor ganancia posible para una acción dada. Asuma que siempre se cuenta con stock suficiente para comprar 1000 acciones en un día i, $1 \leq i \leq n$. El tiempo de ejecución de su algoritmo debe ser **polinomial en** n **y** k, es decir que no debe contener a k en el exponente de la expresión.

(b) Analice el tiempo de ejecución de su algoritmo en función de la cantidad de días n y la cantidad de disparos k.

Sugerencia: Para cada i, $0 \leq i \leq n$, calcule la mayor ganancia que es posible obtener mediante una estrategia de d-disparos, con $0 <= d <= k$, a lo largo de los días 1 hasta i. Para ello evalúe la conveniencia de vender las acciones del último disparo en el día i o no hacerlo.

Ejercicio 3 (Kleinberg & Tardos, Ex. 6.22)**.** Consideramos un grafo dirigido $G = (V, E)$ con aristas ponderadas. Los costos en las aristas pueden ser negativos o positivos, pero todo ciclo en G tiene costo positivo. Para medir la robustez en la conectividad de G queremos contar la *cantidad* de caminos de costo mínimo entre dos vértices. El objetivo de este ejercicio es construir un algoritmo que resuelva el problema en tiempo polinomial utilizando programación dinámica. El algoritmo toma como entrada un par de vértices de G, v, w, y da como resultado un número natural que es la cantidad de caminos de costo mínimo de v a w.

Definimos $OPT(i, s)$ como el costo del camino más corto de v a s usando *exactamente* i aristas. Definimos también $N(i, s)$ como la cantidad de esos caminos de costo mínimo.

(a) Tomando en cuenta que para llegar al nodo s en i pasos, $i > 0$, es necesario haber llegado a un nodo adyacente a s en $i - 1$ pasos, escriba una relación de recurrencia que defina $OPT(i, s)$ en función de los valores de $OPT(i - 1, t)$, $t \in V$.

(b) Asumiendo conocidos los valores de OPT, escriba una relación de recurrencia para $N(i, s)$, $i > 0$, $s \in V$, en función de los valores $N(i - 1, t)$, $t \in V$, y la función OPT.

(c) Observe que la longitud del camino más corto de v a w viene dada por

$$OPT(w) = \min_i \{OPT(i, w)\}, OPT(v) = 0$$

Defina de forma análoga cómo se calcula la cantidad de caminos $N(w)$ en función de los $N(i, w)$.

(d) Escriba un algoritmo que devuelve la cantidad de caminos más cortos. Para ello inicialice los valores de $OPT(i, e)$ y $N(i, e)$ como crea conveniente. Luego compute las recurrencias de las partes anteriores y devuelva el resultado.