Examen de Programación 3
19 de diciembre de 2020

En recuadros con este formato aparecerán aclaraciones que cumplen una función explicativa pero que no eran requeridos como parte de la solución.

Ejercicio 1 (30 puntos)

Ud. fue electo/a intendente de un departamento. Como tal debe cumplir con sus promesas electorales: repavimentar caminos y cuidar las arcas de la comuna. En el departamento hay n centros urbanos y m caminos. Cada camino une un par distinto de centros urbanos. El costo de repavimentar un camino es proporcional a su longitud, y se conoce la longitud de cada camino en kms. No necesariamente existe un camino entre todo par de centros urbanos. Sin embargo, para todo par de centros urbanos, c_1, c_2, es posible llegar de c_1 a c_2 siguiendo una secuencia de caminos.

Su objetivo es seleccionar un conjunto de caminos a repavimentar de forma tal que:

1. Dado un par de centros urbanos c_1, c_2, es posible transportarse de uno a otro usando únicamente caminos repavimentados (aunque no tiene por qué ser a través del recorrido más corto posible).
2. Se repavimenta el menor número total de kilómetros posible que permita cumplir con el punto anterior.

Se pide:

(a) Proponga un algoritmo que resuelve el problema. Debe admitir una implementación cuyo tiempo de ejecución es $O(m \log n)$. Reescriba cualquier algoritmo que use de los estudiados en el curso.

(b) Muestre que su algoritmo admite una implementación cuyo tiempo de ejecución es $O(m \log n)$. Repita los argumentos vistos en el curso. No es necesario demostrar la complejidad de operaciones de *TAD* conocidos.

Solución:

(a) Podemos modelizar el problema como un grafo $G = (V, E)$, con costos en las aristas, donde los nodos son los centros urbanos y las aristas los caminos. El costo de cada arista viene dado por la longitud del camino correspondiente. Por hipótesis el grafo del modelo es conexo, porque para todo par de centros urbanos es posible llegar de uno a otro siguiendo una secuencia de caminos.

En términos de este modelo, lo que buscamos es un conjunto de aristas T tal que (V, T) es conexo (por el requisito 1), y la suma de los costos de sus aristas es lo menor posible (por el requisito 2). Esta es precisamente la definición de un árbol de cubrimiento mínimo para G.

El algoritmo de la figura 1 detalla esta idea. Para la subrutina *MST* se puede utilizar el algoritmo de Prim, o el de Kruskal, ambos estudiados en el curso. El algoritmo, que se pide como parte del ejercicio, no está incluido en este texto, pero puede consultarse en la sección 4.5 del libro de referencia.

```
1 Algorithm repavimentado
2     Hacer G = (V, E), con V = {c_1, ..., c_n} y E inicialmente vacío
3     foreach camino entre dos centros urbanos, c_i y c_j do
4         Agregar a G una arista (c_i, c_j) con costo igual a la distancia de c_i a c_j
5     Crear T = MST(G)
6     Devolver T
7 end
```

Figura 1: Algoritmo para obtener conjunto de caminos a repavimentar.

(b) Construir una representación de listas de adyacencia de G a partir de los datos de la realidad requiere tiempo $O(m + n)$, ya que el paso 2 requiere tiempo $O(n)$ para crear n listas vacías, y el paso 3 se repite

m veces, y cada iteración realiza la inserción de un elemento en las listas de adyacencia de c_i y de c_j, lo cual requiere tiempo $O(1)$.

Como G es conexo, tenemos que $n \leq m + 1$, por lo cual $O(m + n)$ es $O(m)$. Concluimos entonces que los pasos 2–4 del algoritmo requieren tiempo $O(m)$. El paso 5 se puede implementar eficientemente de manera que el tiempo de construcción de T sea $O(m \log n)$. El análisis específico es requerido en el ejercicio, pero no está incluido en este texto; puede consultarse en las secciones 4.5 y 4.6 del libro de referencia.

El tiempo total de ejecución del algoritmo se descompone entonces en dos partes: el tiempo empleado en los pasos 2–4, que es $O(m)$, y el tiempo empleado en la construcción de T, que es $O(m \log n)$. Como este último es asintóticamente más grande que el primero, el tiempo total de ejecución del algoritmo es $O(m \log n)$.

Ejercicio 2 (35 puntos)

Una empresa multinacional de biotecnología desarrolló una vacuna para prevenir la enfermedad COVID-19.

La empresa posee n laboratorios, $\ell_1 \dots \ell_n$, productores de dosis de la vacuna, ubicados en distintos puntos del mundo. Cada laboratorio ℓ tiene la capacidad de producir $c(\ell)$ dosis mensuales.

Existen k países, $p_1 \dots p_k$, interesados en la vacuna. Cada país p demanda $d(p)$ dosis mensuales.

Para abaratar costos, no cualquier laboratorio puede proveer dosis de vacunas a cualquier país. Se evitará por ejemplo el transporte intercontinental de las dosis. Se conoce para cada par laboratorio-país, (ℓ, p), si p puede adquirir dosis provenientes de ℓ o no; cuando sí es posible, no hay limitantes de abastecimiento de ℓ a p más que la propia capacidad de producción de ℓ.

(a) Dé un algoritmo que responda si es posible satisfacer la demanda mensual de todos los países cumpliendo con las restricciones del problema. En caso de responder afirmativamente, el algoritmo también devuelve una matriz A, de dimensiones $n \times k$, tal que $A_{i,j}$ indica cuántas dosis mensuales debe proveer el laboratorio ℓ_i al país p_j.

(b) Demuestre que si su algoritmo responde afirmativamente, entonces existe efectivamente una forma de satisfacer la demanda mensual de todos los países cumpliendo con las restricciones del problema (no debe demostrar el recíproco).

Solución:

(a) El ejercicio se puede modelizar mediante el problema de flujo máximo.

El algoritmo de la figura 2 construye una red de flujo con $n + k + 2$ nodos: uno por cada laboratorio ℓ_i, uno por cada país p_j, un nodo fuente s, y un nodo sumidero t. Cada unidad de flujo (s, ℓ_i, p_j, t) modeliza una vacuna que provee el laboratorio ℓ_i al país p_j.

> En la figura 2 se define la red de flujo G de forma algorítmica, a través de los pasos 2–10. Alternativamente podría darse una definición matemática de G previamente, y construir la red en un único paso del algoritmo aludiendo a su definición.

```
1  Algorithm vacunas
2      Hacer G = (V, E), con V = {s, t, ℓ_1, ..., ℓ_n, p_1, ..., p_k} y E inicialmente vacío
3      for i = 1 to n do
4          agregar a G una arista (s, ℓ_i) con capacidad c(ℓ_i)
5      for j = 1 to k do
6          agregar a G una arista (p_j, t) con capacidad d(p_j)
7      for i = 1 to n do
8          for j = 1 to k do
9              if ℓ_i puede abastecer a p_j then
10                 agregar a G una arista (ℓ_i, p_j) con capacidad d(p_j)
11     Obtener f, un flujo de valor máximo sobre G, usando el algoritmo de Ford-Fulkerson
12     if d(p_j) = f((p_j, t)) para todo j, 1 ≤ j ≤ k then
13         for i = 1 to n do
14             for j = 1 to k do
15                 if (ℓ_i, p_j) ∈ E then
16                     A_ij = f(ℓ_i, p_j)
17                 else
18                     A_ij = 0
19         return (true, A)
20     else
21         return (false, −)
22 end
```

Figura 2: Algoritmo para obtener, si existe, una asignación de dosis a proveer por cada laboratorio a cada país de forma de cumplir con todas las restricciones del problema.

(b) Asumamos que el algoritmo responde afirmativamente, y veamos que una posible forma de satisfacer la demanda mensual de todos los países (tal cual se requiere en la parte a), es que el laboratorio ℓ_i provea a cada país p_j una cantidad A_{ij} de dosis.

Como la respuesta es afirmativa, por la condición del paso 12, el flujo f satisface $f((p_j, t)) = d(p_j)$ para todo j, $1 \leq j \leq k$. Como (p_j, t) es la única arista saliente de cada nodo p_j, $1 \leq j \leq k$, la propiedad de conservación implica que la suma de los valores de flujo de las aristas entrantes a p_j es también $d(p_j)$. Para cada nodo p_j que representa un país entran entonces $d(p_j)$ unidades de flujo (que corresponden a dosis mensuales, según el modelo), por lo que se satisface la demanda de todos los países.

De forma similar, por la propiedad de conservación, en cada nodo ℓ_i la suma de los valores de flujo en aristas salientes, que representa la cantidad total de dosis provistas por ℓ_i, no excede $c(\ell_i)$, que es la capacidad de la única arista entrante a ℓ_i. Por lo tanto, ningún laboratorio provee más dosis mensuales que las que es capaz de producir.

Finalmente, notamos que, por la condición en el paso 9, solo construimos aristas de la forma (ℓ_i, p_j) para pares donde ℓ_i puede abastecer a p_j. Por lo tanto, por la condición en el paso 15, los laboratorios solo abastecen países válidos.

Ejercicio 3 (35 puntos)

En un comercio de venta de productos se quiere encontrar una manera de devolver el cambio de cada compra con la menor cantidad de billetes posible. Para esto, se plantea resolver el siguiente problema. Sea un conjunto de n valores distintos de billete, $B = \{w_i\}_{1 \leq i \leq n}$, donde cada w_i es un entero positivo, el primer valor de billete es $w_1 = 1$, y el conjunto se encuentra ordenado de forma creciente, $w_1 < w_2 < \ldots < w_n$. Dado un valor de cambio a retornar, C, queremos determinar la menor cantidad posible de billetes, con valores en B, que suman C. Por ejemplo, si $B = \{1, 5\}$ y $C = 7$, hacen faltan 3 billetes: uno de valor 5 y dos de valor 1. La cantidad de billetes disponibles de cada valor es ilimitada.

(a) Considere el siguiente algoritmo *Greedy*.

1. Inicializar $R = C$ y $x = 0$.
2. Mientras R sea mayor que cero:
   tomar el mayor valor de billete w_j que no supere a R, incrementar x, y restar w_j a R.
3. Devolver x.

Dé una instancia del problema en la que el algoritmo Greedy propuesto no funciona.

(b) Escriba una relación de recurrencia para calcular la cantidad mínima de billetes para una instancia del problema. Justifique explicando la procedencia de cada término.

Sugerencia: Considere la función $OPT(i, R)$, $1 \leq i \leq n$, $R \geq 0$, definida como la menor cantidad posible de billetes, con valores en $\{w_1, w_2, \ldots, w_i\}$, que suman R.

(c) Escriba un algoritmo iterativo que utilice la técnica de Programación Dinámica para obtener la mínima cantidad de billetes con valores en B, necesarios para devolver un cambio de valor C.

Solución:

(a) Sea $B = \{1, 3, 4\}$ y $C = 6$. El algoritmo *Greedy* propuesto devuelve el valor 3 (el cambio devuelto está compuesto por un billete de valor 4 y dos de valor 1). Sin embargo, se puede devolver una menor cantidad de billetes si se seleccionan 2 billetes de valor 3.

(b)

Para $R = 0$, es claro que $OPT(i, R) = 0$ para todo i, $1 \leq i \leq n$, porque no es necesario devolver ningún billete. Por otra parte, para $i = 1$ solo es posible utilizar billetes de valor 1, por lo cual tenemos $OPT(i, R) = R$ para todo R, $1 \leq R \leq C$. Estas dos observaciones determinan los casos base definidos en (1) y (2). Luego, para $i > 1$ y $R > 0$, distinguimos dos casos:

No incluimos un billete de valor w_i:
En este caso, si no utilizamos ningún billete de valor w_i, entonces la mínima cantidad de billetes para devolver un cambio de valor R está dada por $OPT(i-1, R)$. Esta observación determina el caso (3) de la recurrencia. Notar que este es el único caso posible cuando $w_i > R$.

Incluimos un billete de valor w_i:
Si incluímos un billete de valor w_i, la mínima cantidad de billetes para devolver un cambio de valor R está dada por $1 + OPT(i, R - w_i)$, ya que además de este billete necesitamos $OPT(i, R - w_i)$ billetes para devolver el el resto del cambio de forma óptima. Notar que para incluir un billete de valor w_i, se tiene que cumplir que $R \geq w_i$. Esta observación, en combinación con la anterior, determinan el caso (4) de la recurrencia

$$OPT(i, 0) = 0, \quad 1 \leq i \leq n. \tag{1}$$

$$OPT(1, R) = R, \quad 1 \leq R \leq C. \tag{2}$$

$$OPT(i, R) = OPT(i-1, R), \quad 1 < i \leq n,\ 0 < R < w_i \tag{3}$$

$$OPT(i, R) = \text{mín}\{OPT(i-1, R), 1 + OPT(i, R - w_i)\}, \quad 1 < i \leq n,\ w_i \leq R \leq C \tag{4}$$

(c) El algoritmo se presenta en la figura 3. Aunque no se pide en la letra del problema, en la figura se ilustra también cómo construir una solución, esto es, un algoritmo que construye un arreglo X, donde X_i indica la cantidad de billetes de valor w_i que se usan en una solución.

```
1  Algorithm cambio
2      Hacer OPT[i, 0] = 0, para todo i, 1 ≤ i ≤ n
3      Hacer OPT[1, R] = R, para todo R, 1 ≤ R ≤ C
4      for R = 1 to C do
5          for i = 2 to n do
6              if R < w_i then
7                  OPT[i, R] = OPT[i − 1, R]
8              else
9                  OPT[i, R] = mín{OPT[i − 1, R], 1 + OPT[i, R − w_i]}
       // Solución de parte (c)
       Output: Cantidad mínima de billetes necesarios
10     return OPT[n, R]
       // Construcción de solución
       Output: Arreglo X con las cantidades de cada valor de billete
11     Hacer X[i] = 0 para todo i, 1 ≤ i ≤ n
12     Hacer i = n y R = C
13     while R > 0 do
14         if i = 1 then
15             X[1] = R
16             R = 0
17         else if R < w_i or OPT[i − 1, R] < 1 + OPT[i, R − w_i] then
18             i = i − 1
19         else
20             X[i] = X[i] + 1
21             R = R − w_i
22     return X
23 end
```

Figura 3: Algoritmo para determinar la cantidad mínima de billetes necesarios (hasta paso 10) y para determinar la cantidad necesaria de billetes de cada valor en una solución óptima (desde paso 11)