Examen de Programación 3
7 de febrero de 2024

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

Ejercicio 1 (32 puntos)

Una empresa distribuidora de electricidad planea conectar a un conjunto de proveedores de electricidad, $P = \{p_1, p_2, \ldots, p_n\}$, instalando cables de distribución entre ellos. Conociendo la distancia $d_{i,j}$ entre cada par de proveedores (p_i, p_j), $i \neq j$, busca cablear una red, conexa sin ciclos, que **maximice** el largo total del cableado instalado, de forma de disminuir interferencias de generación. Las distancias son simétricas: $d_{i,j} = d_{j,i}$.

(a) Proponga un algoritmo que resuelva el problema. Este debe admitir una implementación cuyo tiempo de ejecución es $O(n^2 \log n)$. Reescriba cualquier algoritmo que use de los estudiados en el curso.

Sugerencia: Reduzca el problema a uno en el cual puede aplicar directamente un algoritmo conocido.

(b) Demuestre la corrección de su algoritmo. Recuerde que puede usar sin necesidad de reescribir resultados estudiados en el curso.

Solución:

(a) El problema se puede modelar mediante un grafo $G = (V, E)$ en el que los vértices representan los proveedores, $V = P$, y las aristas representan los posibles tendidos de cable. Ponderamos cada arista (p_i, p_j) por la distancia entre los proveedores p_i, p_j. El grafo es conexo dado que existe arista entre todo par de vértices distintos.

Buscar establecer una red, conexa sin ciclos, de mayor distancia total de instalación implica determinar en el modelo un subconjunto de aristas T de E, tales que el grafo (V, T) es conexo, sin ciclos y de mayor ponderación total en sus aristas. Lo que implica que T es un árbol de cubrimiento de G de mayor valor total de ponderaciones de sus aristas.

Determinar un árbol de cubrimiento de mayor valor total de las ponderaciones de sus aristas es equivalente a determinar un árbol de cubrimiento mínimo (MST) en el mismo grafo pero con costos complementarios con respecto a la máxima distancia entre proveedores (ver demostración en parte siguiente).

El algoritmo de la figura 1 implementa la resolución del problema a partir del problema MST. Para la subrutina *MST* se podría utilizar el algoritmo de Prim o el de Kruskal (Sección 4.5 de Kleinberg y Tardos).

```
1 Algorithm MST-maximo
2     Sea M > máx{d_ij}_{1≤i<j≤n} arbitrario
3     Hacer G = (V, E), con V = P y E = ∅
4     foreach i, j con 1 ≤ i < j ≤ n do
5         Hacer c_ij = M − d_ij
6         Agregar a G una arista (p_i, p_j) con costo c_ij
7     Hacer T = MST(G)
8     return T
9 end
```

Figura 1: Algoritmo para obtener una red conexa, sin ciclos, de mayor distancia total de instalación.

(b) Sea T el árbol devuelto por nuestro algoritmo y T' un árbol de cubrimiento de G arbitrario.

Tenemos

$$\begin{aligned}
\sum_{(p_i,p_j)\in T} d_{ij} &= \sum_{(p_i,p_j)\in T} \Big(M - c_{ij}\Big) && (1)\\
&= (n-1)M - \sum_{(p_i,p_j)\in T} c_{ij} && (2)\\
&\geq (n-1)M - \sum_{(p_i,p_j)\in T'} c_{ij} && (3)\\
&= \sum_{(p_i,p_j)\in T'} \Big(M - c_{ij}\Big) && (4)\\
&= \sum_{(p_i,p_j)\in T'} d_{ij}\,, && (5)
\end{aligned}$$

donde (1) y (5) surgen de la definición de c_{ij} en el paso 5 del algoritmo, (2) y (4) se obtienen del hecho de que T y T' tiene $n-1$ aristas por ser árboles, y (3) es consecuencia de que T es un MST de G.

Como T' es arbitrario, se concluye que la suma de las ponderaciones de las aristas de T es al menos tan grande como la de cualquier otro árbol de cubrimiento de G.

Ejercicio 2 (33 puntos)

Sea S una cadena no vacía $s_1 s_2 \dots s_n$ de elementos diferentes. Consideramos las concatenaciones de prefijos no nulos de S, es decir, cadenas de la forma $s_1 s_2 \dots s_{\ell_1} s_1 s_2 \dots s_{\ell_2} \dots s_1 s_2 \dots s_{\ell_m}$, con $1 \leq \ell_i \leq n$ y $m \geq 1$, donde ℓ_i denota el largo del i-ésimo prefijo de la concatenación. Por ejemplo, si $S = ebdac$ algunas posibles concatenaciones son e, eee, $ebda$, $ebde$, y $eebe$.

Sea X una cadena $x_1 x_2 \dots x_n$ de largo n. Se quiere construir una concatenación de prefijos no nulos de S tal que la cantidad de posiciones en las que coincide con X sea máxima. Es decir, se quiere construir $Y = y_1 y_2 \dots y_n$, concatenación de prefijos no nulos de S, tal que $|\{i : 1 \leq i \leq n, x_i = y_i\}|$ sea máxima. Por ejemplo si $X = ddbe$, entonces $ebda$ no coincide en ninguna posición, $ebde$ coincide en la posición 4, y $eebe$ coincide en las posiciones 3 y 4, y es la que coincide en más posiciones.

(a) Especifique una relación de recurrencia que permita resolver el problema mediante Programación Dinámica. Justifique explicando la procedencia de cada término.

Sugerencia: Considere una función $OPT(j)$ definida como la máxima cantidad de coincidencias que puede haber entre el prefijo de largo j de X y una concatenación de prefijos no nulos de S.

(b) Dé un algoritmo iterativo eficiente para calcular la máxima cantidad de coincidencias entre X y alguna concatenación de prefijos no nulos de S.

Solución:

(a) Para $0 \leq j \leq n$ definimos $OPT(j)$ como la máxima cantidad de coincidencias que puede haber entre el prefijo de largo j de X, denotado X^j, y una concatenación de prefijos no nulos de S. Para $j = 0$, no hay ninguna coincidencia posible con la cadena nula X^0, lo cual justifica el paso base en (6).

Para $1 \leq j \leq n$, consideremos una concatenación de prefijos no nulos de S, de largo acumulado total j, y sea i el largo acumulado de todos estos prefijos exceptuando el último. Denotamos con $c(i, j)$ la cantidad de coincidencias entre $x_{i+1} \dots x_j$ y $s_1 \dots s_{j-i}$, es decir,

$$c(i, j) = |\{k : i < k \leq j, x_k = s_{k-i}\}| .$$

La cantidad de coincidencias con X^j es la suma de la cantidad de coincidencias del último prefijo, que está dado por $c(i, j)$, más la cantidad de coincidencias de los prefijos anteriores con X^i, que es $OPT(i)$ si se eligen de forma óptima. El paso inductivo (7) surge de maximizar entre todas las posibles elecciones de i.

$$OPT(0) = 0 , \tag{6}$$

$$OPT(j) = \max_{0 \leq i < j} \Big\{ OPT(i) + c(i, j) \Big\}, 1 \leq j \leq n . \tag{7}$$

(b) El algoritmo de la figura 2 devuelve la cantidad de coincidencias.

```
1  Algorithm PrefijosConcatenadosOptimo (S = s_1 s_2 ... s_n, X = x_1 x_2 ... x_n)
2      for j = 1 to n do
3          for i = 0 to j − 1 do
4              Hacer c[i, j] = |{k : i < k ≤ j, x_k = s_{k−i}}|
5      Hacer OPT[0] = 0
6      for j = 1 to n do
7          Hacer OPT[j] = 0
8          for i = 0 to j − 1 do
9              Hacer OPT[j] = máx{OPT[j], OPT[i] + c[i, j]}
10     return OPT[n]
11 end
```

Figura 2: Algoritmo para calcular la máxima cantidad de coincidencias entre X y alguna concatenación de prefijos no nulos de S.

Ejercicio 3 (35 puntos)

Una carrera universitaria tiene acotada la cantidad de estudiantes que pueden asignarse a sus cursos electivos y la cantidad de cursos que pueden cursar sus estudiantes. Además, los horarios de algunos cursos se superponen entre sí, por lo cual no se pueden cursar simultáneamente.

Sean $\mathcal{E}$ y $\mathcal{C}$ los conjuntos de estudiantes y cursos, respectivamente. Para cada estudiante $e \in \mathcal{E}$, sea C_e el conjunto de los cursos que solicita cursar y sea p_e la cantidad máxima de cursos que puede cursar. Para cada curso $c \in \mathcal{C}$, sea q_c la cantidad máxima de estudiantes que se le puede asignar. Además, conocemos una partición de $\mathcal{C}$, denotada $\mathcal{S}$, tal que para cada $S \in \mathcal{S}$ los cursos de S se superponen en el tiempo y por lo tanto no pueden cursarse simultáneamente, mientras que cursos pertenecientes a diferentes conjuntos de $\mathcal{S}$ no se superponen entre sí.

Se desea asignar a cada estudiante e un conjunto de cursos A_e satisfaciendo las siguientes restricciones:

1. Solo se asignan cursos solicitados, es decir, $A_e \subseteq C_e$ para todo $e \in \mathcal{E}$.
2. La cantidad de cursos asignados a cada estudiante e no supera p_e.
3. A ningún curso c se le asignan más de q_c estudiantes.
4. A ningún estudiante se le asignan cursos que se superponen, es decir, $|A_e \cap S| \leq 1$ para todo $e \in \mathcal{E}$ y todo $S \in \mathcal{S}$.

(a) Proponga un algoritmo de tiempo polinomial en $|\mathcal{E}|$ y $|\mathcal{C}|$ que establezca una asignación de estudiantes a cursos en las condiciones estipuladas, maximizando la cantidad total de asignaciones $\sum_{e \in \mathcal{E}} |A_e|$. Recuerde que no es necesario reescribir algoritmos contenidos en el material teórico del curso.
Sugerencia: Considere una red de flujo $G = (V, E)$ con $V = \{s, t\} \cup \mathcal{E} \cup W \cup C$, donde $W = \mathcal{E} \times \mathcal{S}$.

(b) Demuestre que el tiempo de ejecución de su algoritmo es polinomial en $|\mathcal{E}|$ y $|\mathcal{C}|$. Recuerde que puede usar resultados del material teórico.

Solución:

(a) Para cada $c \in \mathcal{C}$ denotamos con S_c al único conjunto de $\mathcal{S}$ al que pertenece c. Definimos una red de flujo $G = (V, E)$ con $V = \{s, t\} \cup \mathcal{E} \cup W \cup C$, donde $W = \mathcal{E} \times \mathcal{S}$. Las aristas de E y sus capacidades se definen de la siguiente manera:

1. Para cada $e \in \mathcal{E}$ agregamos una arista (s, e) con capacidad p_e.
2. Para cada $e \in \mathcal{E}$ y cada $S \in \mathcal{S}$ tal que $S \cap C_e \neq \emptyset$ agregamos una arista (e, w) con capacidad 1, con $w \in W$ definido como $w = (e, S)$.
3. Para cada $e \in \mathcal{E}$ y cada $c \in C_e$, agregamos una arista (w, c) con capacidad 1, donde $w \in W$ se define como $w = (e, S_c)$.
4. Para cada $c \in C$ agregamos una arista (c, t) con capacidad q_c.

El algoritmo para resolver el problema es el siguiente:

1. Construir la red de flujo sobre G definida anteriormente.
2. Obtener el flujo máximo de s a t en la red invocando el algoritmo de Ford-Fulkerson.
3. Definir inicialmente $A_e = \emptyset$ para todo $e \in \mathcal{E}$. Para cada arista con flujo positivo de la forma (w, c), con $c \in \mathcal{C}$ y $w = (e, S_c) \in W$, agregar c a A_e.

(b) Observar que $|V| = 2 + |\mathcal{E}| + |W| + |\mathcal{C}|$, y además $|W| = |\mathcal{E}| \times |\mathcal{S}| \leq |\mathcal{E}| \times |\mathcal{C}|$. Por lo tanto $|V| \leq 2 + |\mathcal{E}| + |\mathcal{E}| \times |\mathcal{C}| + |\mathcal{C}|$ y como $|E| \leq |V|^2$ concluimos que el tamaño de G es polinomial en $|\mathcal{E}|$ y $|\mathcal{C}|$. Su construcción en el paso 1, por lo tanto, requiere tiempo polinomial en $|\mathcal{E}|$ y $|\mathcal{C}|$.

El corte (A, B), con $A = \{s\} \cup \mathcal{E}$ y $B = V \setminus A$ tiene capacidad acotada superiormente por $|\mathcal{E}| \times |\mathcal{S}|$. Esto es consecuencia de que todas las aristas de A a B son de la forma (e, w), con a lo sumo $|\mathcal{S}|$ asistas partiendo de cada $e \in \mathcal{E}$ a distintos $w \in W$, y todas estas aristas tienen capacidad 1 (ver inciso 2 de la definición de E). Como $|\mathcal{S}| \leq |\mathcal{C}|$, concluimos que la capacidad del corte no es superior a $|\mathcal{E}| \times |\mathcal{C}|$, que es polinomial en $|\mathcal{E}|$ y $|\mathcal{C}|$. Como la cantidad de iteraciones del algoritmo de Ford-Fulkerson está acotada por la capacidad de cualquier corte, y cada iteración requiere tiempo $O(|E|)$, concluimos que el tiempo de ejecución del paso 2 es polinomial en $|\mathcal{E}|$ y $|\mathcal{C}|$.

El paso 3 implica construir $|\mathcal{E}|$ conjuntos vacíos lo cual, representando estos conjuntos mediante listas por ejemplo, requiere tiempo $O(|\mathcal{E}|)$. Adicionalmente se recorre parte de las aristas del grafo G, lo cual requiere tiempo polinomial en $|\mathcal{E}|$ y $|\mathcal{C}|$, agregando cuando corresponde un elemento

a un conjunto A_e, lo cual requiere tiempo $O(1)$. En total el paso 3 requiere tiempo polinomial en $|\mathcal{E}|$ y $|\mathcal{C}|$.

En conclusión, el algoritmo consta de 3 pasos, cada uno de los cuales requiere tiempo polinomial en $|\mathcal{E}|$ y $|\mathcal{C}|$, por lo cual el tiempo total es también polinomial en $|\mathcal{E}|$ y $|\mathcal{C}|$.