Parcial de Programación 3
26 de noviembre de 2018

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

Ejercicio 1 (16 puntos)

Algunas asignaturas electivas de una carrera universitaria tienen acotada la cantidad máxima de estudiantes que pueden cursarlas en un semestre (denominada cupo). Se plantea resolver la asignación de estudiantes a asignaturas a través de un proceso centralizado, para lo cual se acota la cantidad de asignaturas que puede cursar cada estudiante en un semestre a un máximo de k.

Sean A el conjunto de asignaturas y E el conjunto de estudiantes. Para cada $a \in A$, sea q_a el cupo de la asignatura a y, para cada $e \in E$, sea S_e el conjunto de las asignaturas a las que se ha inscripto el estudiante e.

(a) Proponga un algoritmo de tiempo polinomial en $|A|$ y $|E|$ que establece una asignación de estudiantes a asignaturas en las condiciones estipuladas, maximizando la cantidad total de asignaciones de estudiantes a asignaturas (es decir, la suma sobre todas las asignaturas de la cantidad de estudiantes asignados a ella). Recuerde que no es necesario reescribir algoritmos contenidos en el material teórico del curso.

Sugerencia: Modele el problema mediante un problema de flujo máximo en red.

(b) Analice la complejidad del algoritmo propuesto. Recuerde que puede usar resultados del material teórico.

Solución:

(a) El problema se puede modelar mediante el problema de flujo máximo, y éste a su vez se puede resolver mediante el algoritmo de Ford-Fulkerson.

La red de flujo se establece mediante un nodo v_e por cada estudiante $e \in E$, un nodo w_a por cada asignatura $a \in A$, y una arista (v_e, w_a) de capacidad uno si el estudiante e se ha inscripto en la asignatura e, es decir si $a \in S_e$. Además, se establecen un super nodo s y aristas (s, v_e) de capacidad k que conectan a s con cada estudiante v_e, y se definen un super nodo t y aristas (w_a, t) de capacidad q_a que conectan a cada asignatura w_a con t.

Calculamos el flujo máximo usando el algoritmo de Ford-Fulkerson y asignamos un estudiante e a la asignatura a si la arista (v_e, w_a) transporta flujo de valor uno.

Existe una solución factible de x asignaciones de estudiantes a asignaturas en el problema original si y solo si el flujo máximo en la red tiene valor x.

(b) Sean $V = \{v_e | e \in E\}$, $W = \{w_a | a \in A\}$, y sea $C = k|E|$ la cota superior para el flujo máximo determinada por la capacidad del corte $(\{s\}, V \cup W \cup \{t\})$. El problema de flujo máximo en un grafo con m aristas, n nodos y cota de flujo máximo C puede ser resuelto mediante Ford-Fulkerson en tiempo $O((m+n)C)$, o si se asume que m es $\Omega(n)$, en tiempo $O(mC)$ (Enunciado KyT (7.5)). En nuestro caso, $m + n$ es $O(|E||A|)$ y C es $O(|E|)$, de donde concluimos que el tiempo de ejecución del algoritmo es polinomial en $|A|$ y $|E|$.

Una cota potencialmente más ajustada para el flujo máximo podría ser

$$C' = \text{mín}\{k|E|, \sum_{e \in E} |S_e|, \sum_{a \in A} q_a\},$$

donde los argumentos del operador mín$\{\}$ corresponden a las capacidades de los cortes $(\{s\}, V \cup W \cup \{t\})$, $(\{s\} \cup V, W \cup \{t\})$ y $(\{s\} \cup V \cup W, \{t\})$, respectivamente.

Ejercicio 2 (22 puntos)

Sea $G = (V, E)$ un grafo dirigido y sea t un vértice de G. Cada arista $(v, w) \in E$ tiene un *costo* positivo asociado que denotamos con c_{vw}; el *costo* de un camino en G es la suma de los costos de las aristas que lo componen.

Para cada vértice $v \in V$ queremos encontrar el costo mínimo entre los caminos de v a t que tienen **a lo sumo** K aristas, $0 \leq K < |V|$. Este costo mínimo se denota con $\text{OPT}(K, v)$.

Sea $\text{CMC}(i, v)$ el costo mínimo entre los caminos de v a t que tienen **exactamente** i aristas, $i \geq 0$, o $\text{CMC}(i, v) = \infty$ si no existe tal camino. Notar que $\text{CMC}(0, t) = 0$.

(a) Escriba una relación de recurrencia para CMC.

(b) Dé el valor de $\text{OPT}(K, v)$ en función de CMC.

(c) Dé un algoritmo iterativo que obtiene $\text{OPT}(K, v)$ para cada $v \in V$. El algoritmo debe admitir una implementación cuyo tiempo de ejecución es polinomial en $|V|, |E|$. No es necesario implementar operaciones elementales sobre conjuntos como contar la cantidad de elementos u obtener valores mínimos o máximos.

(d) Analice la complejidad del algoritmo propuesto.

Solución:

(a) Por la definición de CMC se cumple

$$\text{CMC}(0, v) = \begin{cases} 0, & v = t\,, \\ \infty, & v \neq t\,. \end{cases} \tag{1}$$

Para $i > 0$ y $v \in V$, sea C un camino de costo mínimo desde v a t con i aristas. C está compuesto por una arista que sale de v hacia otro nodo, digamos w, concatenado con un camino C' de $i - 1$ aristas desde w hasta t. El costo de C' debe ser $\text{CMC}(i - 1, w)$: no puede ser menor por definición de CMC y, si fuera mayor, C no sería de costo mínimo. El costo total de C está dado por la suma $\text{CMC}(i - 1, w) + c_{vw}$, de modo que el hecho de que C sea de costo mínimo, es decir igual a $\text{CMC}(i, v)$, implica que se cumple

$$\text{CMC}(i, v) = \min_{(v,w) \in E} \{\text{CMC}(i - 1, w) + c_{vw}\}\,, \quad 1 \leq i \leq K\,, \tag{2}$$

donde convenimos que el mínimo de un conjunto vacío es ∞.

(b) Un camino de costo mínimo de v a t que tienen a lo sumo K aristas es, en particular, un camino de costo mínimo de v a t que tienen exactamente i aristas para algún i, $0 \leq i \leq K$. Por lo tanto, se cumple

$$\text{OPT}(K, v) = \min_{0 \leq i \leq K} \{\text{CMC}(i, v)\}\,. \tag{3}$$

Una solución alternativa para este problema (que no es la que se pide en la letra del ejercicio) consiste en escribir directamente una relación de recurrencia para OPT.

$$\text{OPT}(0, v) = \begin{cases} 0, & v = t\,, \\ \infty, & v \neq t\,. \end{cases} \tag{4}$$

$$\text{OPT}(k, t) = 0\,, \quad 1 \leq k \leq K\,. \tag{5}$$

$$\text{OPT}(k, v) = \min_{(v,w) \in E} \left\{\text{OPT}(k - 1, w) + c_{vw}\right\}, \quad 1 \leq k \leq K, v \in V \setminus \{t\}\,. \tag{6}$$

(c)

```
1  Algorithm CaminosDeMenorCostoHastaK (K,G = (V,E))
2      foreach v ∈ V, v ≠ t do
3          CMC(0, v) = ∞
4      end
5      CMC(0, t) = 0
6      for 1 ≤ i ≤ K do
7          foreach v ∈ V do
8              CMC(i, v) = mín_{(v,w)∈E}{CMC(i − 1, w) + c_vw}
9          end
10     end
11     foreach v ∈ V do
12         OPT(K, v) = mín_{0≤i≤K}{CMC(i, v)}
13     end
14 end
```

(d) Sea $n = |V|$ y $m = |E|$.

El ciclo de la línea 2 requiere un tiempo de ejecución que es $O(n)$.

Asumiendo una representación de G mediante listas de adyacencia, el ciclo de la línea 6 requiere un tiempo de ejecución que es $O(K(n+m))$, porque, para cada valor de i, $1 \leq i \leq K$, cada vértice se procesa una sola vez y cada arista se considera una sola vez al procesar el vértice desde el cual sale.

> Con una representación del grafo mediante matriz de adyacencia el tiempo sería $O(Kn^2)$.

El ciclo de la línea 11 requiere un tiempo de ejecución que es $O(nK)$.

Por lo tanto, teniendo en cuenta que $K < n$, el tiempo de ejecución del algoritmo es de orden polinomial en n y m, ya que es una secuencia de ciclos cada uno de los cuales requiere un tiempo de ejecución polinomial en n y m.

Ejercicio 3 (20 puntos)

En el problema de Ubicación Optima de Plantas Industriales (*UOPI*), tenemos un conjunto de ubicaciones candidatas para instalar plantas industriales. Asociada a cada ubicación hay un *valor* entero, que representa la ganancia obtenida por instalar una planta en ese lugar, y nos interesa elegir ubicaciones de modo que la suma de las ganancias sea al menos t, donde t es un parámetro entero. La elección de las ubicaciones no es arbitraria porque algunas son excluyentes entre sí (por ejemplo no pueden estar demasiado cerca o sobre un mismo río).

Concretamente, modelamos una instancia de este problema mediante el entero t y un grafo G, donde cada vértice es una ubicación candidata y una arista (v_i, v_j) significa que las ubicaciones v_i y v_j son excluyentes entre sí (no pueden tener ambas una planta asignada). A cada vértice v_i asociamos un entero w_i que representa el valor de la ubicación v_i.

(a) Muestre que *UOPI* pertenece a la clase de problemas NP.

(b) Muestre que *UOPI* es NP-completo.
Sugerencia: Relacione *UOPI* con *Independent Set* (Conjunto Independiente).

Solución:

(a) Lo que se debe demostrar es que existe un algoritmo certificador, B, que toma como entrada una instancia s de *UOPI* y un string c y se cumple

- su tiempo de ejecución es polinomial,
- s es una instancia `SÍ` si y solo si existe al menos un certificado c, de tamaño polinomial en el tamaño de s, tal que si s y c son los parámetros con los que se corre B el resultado es `SÍ`.

La instancia s, consiste en un arreglo con los valores w_i, una lista de adyacentes para cada vértice, y el entero t. El certificado c es un string de bits (ceros y unos) de largo n, donde n es la cantidad de vértices del grafo.

El algoritmo B realiza dos verificaciones:

1. Suma los w_i para cada i tal que c_i es 1. Si esa suma es menor a t termina sin certificar.
2. Para cada i tal que c_i es 1 recorre la lista del vértice i. Si para algún j en alguna de las listas se cumple que c_j es 1, entonces termina sin crtificar. En otro caso, si c_j es 0 para cada vértice de cada lista, termina emitiendo el resultado `SÍ`.

Se debe notar que el tamaño de c, que es n, es polinómico en el tamaño de s, y que el tiempo de ejecución del algoritmo, que consiste en sumar algunos elementos de un arreglo y recorrer el grafo de entrada, es polinómico.

Interpretamos c como una selección de vértices, y por lo tanto de las ubicaciones que modelan, de tal forma que las ubicaciones seleccionadas son las que corresponden a los i para los que se cumple que c_i es 1.

Veamos que si el resultado de B es `SÍ` entonces s es una instancia `SÍ`. Si el resultado es `SÍ` significa que se cumplen las dos verificaciones. Entoces

1. la suma de los valores de las ubicaciones seleccionadas es al menos t, y
2. ningún par de ubicaciones seleccionadas son excluyentes porque para cada par (i, j) de ubicaciones excluyentes si i es seleccionada (o sea, si c_i es 1) j no es seleccionada (c_j es 0).

Por lo tanto, s es una instancia `SÍ`.

En el otro sentido veamos que si s es una instancia `SÍ` existe un certificado con el cual la ejecución de B da como resultado `SÍ`. Si es una instancia `SÍ` significa que hay un conjunto de ubicaciones cuya suma de valores es no menor a t y ningún par de ellas es excluyente. Entonces si c corresponde a esas ubicaciones (o sea, c_i es 1 si y solo si i es una de las ubicaciones) al correr B con parámetros s y t las dos condiciones se verifican por lo que el resultado es `SÍ`.

Con esto queda demostrado que *UOPI* pertenece a NP .

(b) Dada una instancia (G, V, k) de *Independent Set (INDSET)*, la transformamos en una instancia (G', V', W, t) de *UOPI* de la siguiente forma:

- $G' = G$,
- $W[i] = 1$ a todo vértice v_i de V,
- $t = k$.

Observamos en primer lugar que la transformación requiere un tiempo polinomial en el tamaño de la representación de la instancia original. En efecto, los grafos de ambas instancias coinciden y la generación del arreglo W que representa los valores de las plantas se hace en tiempo $O(n)$.

Mostramos ahora que una una instancia de *INDSET* tiene respuesta afirmativa si y solo si la instancia transformada de *UOPI* tiene respuesta afirmativa. Como los grafos G y G' son iguales, un subconjunto S de V es independiente en G si y solo si no hay exclusiones mutuas en el conjunto de plantas representadas por S. A su vez, como todas las plantas tienen valor 1 y $t = k$, la suma de las ganancias asociadas a las plantas representadas por S es mayor igual a t si y solo si $|S| \geq k$. Esto muestra la equivalencia entre las respuestas de las instancias originales de *INDSET* y las transformadas de *UOPI*.

Como *INDSET* es un problema NP-completo, por propiedad transitiva cualquier problema NP se puede reducir en tiempo polinomial a *UOPI*.

Esto, junto con la parte anterior, demuestra que *UOPI* es también un problema NP-completo.