Examen de Programación 3
20 de julio de 2021

Ejercicio 1 (35 puntos)

Considere un proyecto compuesto por un conjunto A de k actividades, cuya duración total estimada es de n días, comenzando desde el día 1. Cada actividad $i \in A$ tiene una duración de d_i días y debe realizarse en el período comprendido entre los días s_i y f_i (incluyéndolos). Para todo $i \in A$ se cumple $1 \leq s_i \leq f_i \leq n$, y $f_i - s_i + 1 \geq d_i$, es decir, existen al menos d_i días comprendidos entre s_i y f_i.

Cada día se elige una única actividad de las que resta terminar y se trabaja exclusivamente en ella durante todo ese día. Una actividad no necesariamente debe completarse en días consecutivos, sino que su ejecución puede intercalarse con la de otras actividades. El proyecto finaliza cuando todas las actividades se han completado.

El problema consiste en asignar, de ser posible, los días en los que debe realizarse cada actividad, de forma tal que todas puedan completarse dentro del plazo estimado para el proyecto, además de satisfacerse las restricciones de comienzo y fin de cada una.

(a) Modele el problema mediante una red de flujo. Dé un algoritmo que a partir de dicha red, responda si existe o no una asignación que satisface las restricciones del problema. En caso de responder afirmativamente, el algoritmo debe devolver también una asignación factible. No es necesario que reescriba algoritmos estudiados en el curso.

(b) Demuestre que el tiempo de ejecución de su algoritmo es $O(kn^2)$. Puede usar sin demostración resultados contenidos en el material teórico del curso.

Solución:

(a) Modelamos el problema como una red de flujo $G = (V, E)$, donde el conjunto de vértices V se construye de la siguiente manera.

- Para cada actividad $i \in A$ y día $j \in \{1, \ldots, n\}$, se agrega a un nodo i y j respectivamente.
- Se agrega un nodo fuente s y un nodo terminal t.

Por otro lado, el conjunto de aristas E queda definido de la siguiente manera.

- Para cada actividad $i \in A$, se agrega una arista (s, i) con capacidad d_i.
- Para cada actividad $i \in A$ y día $j \in \{1, \ldots, n\}$, tal que $s_i \leq j \leq f_i$, se agrega una arista (i, j) con capacidad igual a 1.
- Para cada día $j \in \{1, \ldots, n\}$, se agrega una arista (j, t) con capacidad igual a 1.

Por último, definimos D como la suma de las duraciones de todas las tareas, es decir, $D = \sum_{i=1}^{k} d_i$. El algoritmo que determina si existe una asignación factible consiste en los siguientes pasos.

1. Construir la red de flujo G definida anteriormente.
2. Obtener el flujo máximo f de la red G invocando el algoritmo de Ford-Fulkerson.
3. Si el valor del flujo $v(f)$ es igual a D, se responde afirmativamente, y a cada actividad i se le asignan los días j para los cuales la arista $e = (i, j)$ cumple que $f(e) > 0$. En cualquier otro caso se responde que no existe una asignación factible.

(b) Representando la red de flujo G con listas de adyacencia, la contrucción del grafo puede realizarse en tiempo $O(|V| + |E|)$. Para esto podemos enumerar los vértices que representan días con índices de 1 a n, los vértices que representan actividades con índices de $n + 1$ a $n + k$, el vértice s con índice 0 y el vértice t con índice $n + k + 1$. De esta forma, cada una de las inserciones de aristas que requiere la construcción de G se realiza en tiempo $O(1)$ ya que los índices asociados a los vértices involucrados se determinan en tiempo $O(1)$.

Observar que la cantidad de vértices $|V|$ es igual a $k + n + 2$, que es $O(n + k)$. Además, hay k aristas de la forma (s, i), a lo sumo kn aristas de la forma (i, j), y n aristas de la forma (j, t), por lo que la cantidad de aristas $|E|$ es $O(kn)$. Por to tanto, el tiempo que toma el paso 1 es $O(kn)$.

Por otra parte, el tiempo del paso 2 está dado por la invocación al algoritmo de Ford-Fulkerson, cuyo tiempo es $O(|E|C)$, siendo C la capacidad de un corte cualquiera de G. En nuestro caso, $|E|$ es $O(kn)$, y eligiendo el corte $s - t$ dado por $(V \setminus \{t\}, \{t\})$, de capacidad igual a n, queda demostrado que el tiempo de ejecución de este paso es $O(kn^2)$.

Con respecto al paso 3, chequear la condición del valor del flujo requiere tiempo $O(1)$ si dicho valor se mantiene durante la ejecución del algoritmo de Ford-Fulkerson. Adicionalmente, la construcción de una asignación factible requiere tiempo $O(kn)$, puesto que en el peor caso se debe examinar el flujo que circula por cada arista de la forma (i, j), para $i \in A$, $j \in \{1, \ldots, n\}$.

De lo anterior, se desprende que el tiempo de cada paso puede acotarse por $O(kn^2)$, de lo que se concluye que tiempo de ejecución total del algoritmo es $O(kn^2)$.

Ejercicio 2 (35 puntos)

Consideramos el cálculo del producto de n matrices, $A_1 \times A_2 \times \ldots \times A_n$, donde cada matriz A_i tiene dimensiones $d_{i-1} \times d_i$, es decir, d_{i-1} filas y d_i columnas, $1 \leq i \leq n$. Notar que para $1 \leq i < j \leq n$, la multiplicación $A_i \times A_{i+1} \times \ldots \times A_j$ da como resultado una matriz de dimensiones $d_{i-1} \times d_j$.

Cada multiplicación matricial de la forma $U \times V$, donde U tiene dimensiones $p \times q$ y V tiene dimensiones $q \times r$, requiere pqr multiplicaciones escalares con el algoritmo de multiplicación estándar.

La cantidad de multiplicaciones escalares requeridas para calcular $A_1 \times A_2 \times \ldots \times A_n$ depende de cómo se agrupen las multiplicaciones matriciales usando la propiedad asociativa. Por ejemplo, para tres matrices

$$A_1 \text{ de } 10 \times 30\,, \quad A_2 \text{ de } 30 \times 5\,, \quad A_3 \text{ de } 5 \times 60\,, \qquad (d_0, d_1, \ldots, d_3 = 10, 30, 5, 60)\,,$$

- $(A_1 \times A_2) \times A_3$ requiere $\overbrace{(10 \times 30 \times 5)}^{U = A_1 \times A_2} + \overbrace{(10 \times 5 \times 60)}^{U \times A_3} = 4500$ multiplicaciones escalares, y
- $A_1 \times (A_2 \times A_3)$ requiere $\underbrace{(30 \times 5 \times 60)}_{U = A_2 \times A_3} + \underbrace{(10 \times 30 \times 60)}_{A_1 \times U} = 27000$ multiplicaciones escalares.

Se pide:

(a) Especifique una relación de recurrencia para determinar la mínima cantidad posible de multiplicaciones escalares requeridas para el cálculo de $A_1 \times A_2 \times \ldots \times A_n$, en función de la secuencia de naturales $d_0, d_1, \ldots, d_n$. La minimización es entre todas las posibles formas de asociar los factores mediante paréntesis. Justifique su respuesta indicando el origen de cada uno de los términos de la recurrencia.

Sugerencia: Defina una función de la forma $OPT(i, j)$, donde i, j delimitan un subproblema del problema original.

(b) Según la recurrencia de la parte anterior, escriba un algoritmo iterativo de Programación Dinámica para obtener dicha cantidad mínima de multiplicaciones escalares.

Solución:

(a) $Definimos OPT(i, j)$ como el mínimo número de multiplicaciones escalares requeridas para realizar la multiplicación $A_i \times A_{i+1} \times \ldots \times A_j$, con $1 \leq i \leq j \leq n$.

Tenemos $OPT(i, i) = 0$, ya que no se requiere ninguna multiplicación escalar para una cantidad nula de multiplicaciones matriciales. Esto justifica el paso base de la recurrencia, (1).

Para $1 \leq i < j \leq n$, calcular $A_i \times A_{i+1} \times \ldots \times A_j$ implica:

1. calcular $U = A_i \times A_{i+1} \times \ldots \times A_k$, para cierto k, $i \leq k < j$,
2. calcular $V = A_{k+1} \times A_{k+2} \times \ldots \times A_j$,
3. calcular $U \times V$.

La realización de los pasos 1 y 2 de forma óptima requiere, por definición de OPT, $OPT(i, k)$ y $OPT(k+1, j)$ multiplicaciones escalares respectivamente. El paso 3 requiere $d_{i-1} d_k d_j$ multiplicaciones escalares. Por lo tanto, para realizar la menor cantidad posible de multiplicaciones escalares debemos elegir k de forma de minimizar $OPT(i, k) + OPT(k+1, j) + d_{i-1} d_k d_j$. Esto da lugar al paso inductivo de la recurrencia, (2).

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

$$OPT(i, j) = \min_{i \leq k < j} \left\{ OPT(i, k) + OPT(k+1, j) + d_{i-1} d_k d_j \right\}, \quad 1 \leq i < j \leq n \tag{2}$$

(b) Se presenta el algoritmo en la figura 1.

```
1 Algorithm Mínimo multiplicaciones
2     Hacer OPT[i, i] = 0 para todo i, 1 ≤ i ≤ n
      /* ℓ representa la cantidad de multiplicaciones matriciales en
         A_i × A_{i+1} × ... × A_j.                                    */
3     for ℓ = 1 to n − 1 do
4         for i = 1 to n − ℓ do
5             Hacer j = i + ℓ
6             Hacer OPT[i, j] = mín_{i≤k<j} { OPT[i, k] + OPT[k + 1, j] + d_{i−1} d_k d_j }
7     return OPT[1, n]
8 end
```

Figura 1: Algoritmo para determinar la cantidad mínima de multiplicaciones escalares para calcular $A_1 \times A_2 \times \ldots \times A_n$.

Ejercicio 3 (30 puntos)

Considere el código de un programa P compuesto por N líneas numeradas de 1 a N. Se quiere analizar si el programa funciona correctamente, ejecutando una serie de corridas de prueba. Para esto se cuenta con un conjunto de M tests, numerados de 1 a M, donde cada test i, $1 \leq i \leq M$, ejecuta un subconjunto L_i de líneas de P, $L_i \subseteq \{1, 2, \ldots, N\}$. Considere el problema de decisión *TEST* definido de la siguiente manera: Dada la cantidad de líneas de programa, N, y los conjuntos L_i, $1 \leq i \leq M$, correspondientes a M tests, ¿existe un conjunto de a lo sumo k tests que ejecute al menos una vez cada línea de P?

(a) Demuestre que *Vertex Cover* $\leq_P$ *TEST*. Repita cualquier argumento que utilice de los estudiados en el curso.

(b) Demuestre que *TEST* es $\mathcal{NP}$-Completo.

Solución:

(a) Consideramos una instancia arbitraria (G, k) de *Vertex Cover*, con $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_n\}$, y $E = \{e_1, e_2, \ldots, e_m\}$. Construimos una instancia (L, N, k') de *TEST*, donde $L = \{L_i\}_{1 \leq i \leq M}$, haciendo corresponder líneas de programa con aristas de G y tests con vértices de G. Específicamente, definimos (L, N, k') de la siguiente forma:

1. La cantidad de líneas del programa es $N = m$.
2. La cantidad de tests es $M = n$.
3. Para cada i, $1 \leq i \leq M$, definimos $L_i = \Big\{j \in \{1, \ldots, N\} : e_j \text{ es incidente a } v_i \text{ en G}\Big\}$.
4. Definimos $k' = k$.

Veamos que esta transformación se puede implementar de tal forma que requiere tiempo polinomial en el tamaño de la representación de la instancia de *Vertex Cover*. La construcción de L_i en el paso 3 implica recorrer las aristas incidentes a v_i, lo cual requiere tiempo $O(n)$ ya que v_i tiene a lo sumo $n-1$ vértices adyacentes. Por lo tanto, el tiempo total requerido por el paso 3 es $O(mn)$ y, como los pasos 1, 2 y 4 requieren tiempo $O(1)$, concluimos que el tiempo total de ejecución de la transformación es polinomial en n y m. Esto a su vez implica que el tiempo total de ejecución es también polinomial en el tamaño de la representación de la instancia de *Vertex Cover*.

Vamos a probar que G tiene una Vertex Cover de a lo sumo k nodos si y solo si en (L, N, k') existe un conjunto de a lo sumo k' tests que ejecutan al menos una vez cada una de las N líneas del programa.

Supongamos que G tiene un Vertex Cover $S = \{v_{i_1}, v_{i_2}, \ldots, v_{i_r}\}$, con $r \leq k$. Afirmamos que el conjuntos de tests $T = \{i_1, i_2, \ldots, i_r\}$, cuyo tamaño r no supera k' porque $k' = k$, ejecuta todas las líneas de programa. En efecto, para j arbitrario, $1 \leq j \leq N$, el hecho de que S sea un Vertex Cover implica que la arista e_j es incidente a algún vértice $v_{i_q} \in S$. Por lo tanto, por la definición de L_{i_q} en el paso 3 de la transformación, se cumple que la línea j es ejecutada por el test i_q de T. Como j es arbitrario, nuestra afirmación está probada.

Supongamos ahora que $T = \{i_1, i_2, \ldots, i_r\}$ es un conjunto de r tests, $r \leq k$, que ejecuta todas las líneas de programa. Afirmamos que $S = \{v_{i_1}, v_{i_2}, \ldots, v_{i_r}\}$ es un Vertex Cover de G. Sea e_j, $1 \leq j \leq m$, una arista arbitraria de G. Como j es un índice entre 1 y N, existe q, $1 \leq q \leq r$, tal que el test i_q de T ejecuta la línea j, es decir, se cumple $j \in L_{i_q}$. Por lo tanto, por la definición de L_{i_q} en el paso 3 de la transformación, se cumple que e_j es incidente a v_{i_q}, que es un vértice perteneciente a S. En consecuencia, como e_j es una arista arbitraria, S es un Vertex Cover de G.

Concluimos entonces que es posible resolver *Vertex Cover* a través de la transformación que acabamos de definir, que requiere tiempo polinomial, y una única invocación a un algoritmo que resuelve *TEST*. Esto demuestra que *Vertex Cover* $\leq_P$ *TEST*.

(b) Como *Vertex Cover* es $\mathcal{NP}$-Completo, por la parte anterior solo resta probar que *TEST* está en $\mathcal{NP}$. Para esto, definimos un certificado para una instancia (L, N, k) como un conjunto de tests, definido mediante un arreglo booleano de tamaño M, donde la i-ésima posición del arreglo indica si el test i pertenece o no al certificado.

Afirmamos que el algoritmo de la figura 2 es un certificador eficiente para *TEST*. El algoritmo recibe una representación binaria de una instancia (L, N, k) del problema y una tira de M bits que representa un certificado C.

```
1 Algorithm Certificador-TEST((L, N, k), C)
2     if |C| > k then return false
3     Sea P' un arreglo booleano de tamaño N
4     Inicializar cada posición de P' en false
5     foreach i en C do
6         foreach j en L_i do
7             Hacer true la posición j de P'
8     if todas las posiciones de P' son true then return true
9     else return false
```

Figura 2: Certificador eficiente para *TEST*.

Consideremos una instancia arbitraria (L, N, k) de *TEST*, $L = \{L_i\}_{1 \leq i \leq M}$. Si (L, N, k) es una instancia SÍ, entonces existe un conjunto W de a lo sumo k tests que ejecuta al menos una vez cada línea de programa. El certificado $C = W$ es de largo polinomial en el tamaño de la instancia (L, N, k), ya que su largo, M, es igual a la cantidad de conjuntos L_i que forman parte de L. Con este certificado, la condición del paso 2 del algoritmo certificador es falsa porque $|W| \geq k$. Por otro lado, como el conjunto de tests W ejecuta al menos una vez cada línea de programa, el paso 7 del algoritmo se va a ejecutar al menos una vez por cada posición de P'. De esta manera, la condición del paso 8 va a ser **true**, y el certificador va a retornar **true**.

Por otra parte, si existe una tira C que hace que el certificador responda **true**, entonces la instancia es SÍ. Efectivamente, para que el certificador responda **true** debe cumplirse que C contiene al menos k elementos, porque de lo contrario se devolvería **false** en el paso 2. Además, el subconjunto de tests en C debe ejecutar todas las líneas de programa al menos una vez, porque de lo contrario existiría al menos una posición del arreglo P' con valor **false** durante la ejecución del paso 8, lo que haría que el algoritmo certificador retornara **false**.

Concluimos que (L, N, k) es una instancia SÍ si y solo si existe un certificado C de largo M (que es polinomial en el tamaño de la instancia) que hace que el algoritmo del la figura 2 responda **true** para las entradas (L, N, k), C.

Respecto al tiempo de ejecución del algoritmo certificador, el paso 7 se ejecuta no más de MN veces, mientras que los pasos 3, 4, y 8 se ejecutan en tiempo $O(N)$, y el resto en tiempo $O(1)$. Por lo tanto, el tiempo de ejecución del algoritmo certificador es polinomial en el tamaño de la entrada.