Parcial de Programación 3
16 de octubre 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 (16 puntos)

Sea $G = (V, E)$ un grafo con costos en las aristas, todos distintos entre sí. Se quiere determinar si una arista dada (u, v) es la arista de mayor costo en **algún** ciclo.

(a) Sea $G' = (V, E')$ el grafo que se obtiene eliminando de G todas las aristas más costosas que (u, v). Muestre que (u, v) es la arista de mayor costo de algún ciclo en G si y solo si (u, v) pertenece a algún ciclo en G'.

(b) Dé un algoritmo que resuelve el problema. Debe admitir una implementación cuyo tiempo de ejecución es $O(m + n)$, donde n es la cantidad de vértices y m la cantidad de aristas de G.

Sugerencia: Considere el grafo que se obtiene a partir de G' eliminando la arista (u, v).

(c) Demuestre que su algoritmo admite una implementación cuyo tiempo de ejecución es $O(m + n)$.

Solución:

(a) Demostramos por separado cada una de las dos implicancias correspondientes al directo y al recíproco de la equivalencia de la tesis.

Si (u, v) es la arista de mayor costo en un ciclo de G, entonces pertenece a un ciclo de G': Sea C un ciclo de G del cual (u, v) es la arista de mayor costo. Como para obtener G' no se elimina (u, v) ni ninguna arista de menor costo, C es un ciclo de G' al cual pertenece (u, v).

Si (u, v) pertenece a un ciclo de G', entonces es la arista de mayor costo en un ciclo de G: Sea C un ciclo de G' al que pertenece (u, v). Por construcción, (u, v) es la arista de mayor costo de G' y por lo tanto también la de C. Como cada arista de G' es arista de G ($E' \subset E$), el ciclo C también es ciclo de G, del cual (u, v) es la arista de mayor costo.

(b) Sea G'' el grafo que se obtiene eliminando (u, v) de G'. Entonces (u, v) pertenece a un ciclo en G' si y solo si u y v pertenecen a la misma componente conexa en G''. Juntando esto con la parte anterior, el problema de determinar si (u, v) es la arista de mayor costo en G es equivalente al de determinar si u y v pertenecen a la misma componente conexa en G''.

Teniendo en cuenta que G'' es el grafo que se obtiene de G eliminando las aristas de costo mayor o igual al de (u, v), el siguiente algoritmo responde **Verdadero** si y solo si (u, v) es la arista de mayor costo en algún ciclo de G.

1. Construir G'' eliminando de G las aristas de costo mayor o igual al de (u, v).
2. Determinar si u y v están en la misma componente conexa en G''. Responder **Verdadero** en caso afirmativo, y **Falso** en caso contrario.

(c) Usamos la representación del grafo mediante listas de adyacencia. Los elementos de la lista mantienen, además del vértice vecino, el costo de la arista.

El paso 1 consiste en recorrer una vez cada una de las n listas, tal vez eliminando elementos. Cada eliminación se hace en tiempo $O(1)$, y la cantidad de elementos en todas las listas es $2m$. Por lo tanto el paso 1 requiere tiempo $O(m + n)$.

El paso 2 consiste en una recorrida (por ejemplo mediante DFS o BFS) que obtenga la componente conexa de u. Esto requiere tiempo $O(m + n)$. Si el resultado consiste en un arreglo booleano con una casilla por vértice, se determina en tiempo $O(1)$ si v pertenece a la misma componente conexa de u.

En total tenemos una cantidad finita de pasos, cada uno de los cuales requiere tiempo $O(m + n)$, de donde se desprende que el tiempo total de ejecución es $O(m + n)$.

Ejercicio 2 (17 puntos)

Debido al contexto actual de emergencia sanitaria, una institución educativa decidió implantar un sistema de avisos en los distintos salones donde se desarrollan las clases. El objetivo del sistema es emitir un mensaje que recuerde las medidas a ser tomadas para evitar nuevos contagios.

Cuando se emite el mensaje usando el sistema, este es reproducido simultáneamente en todos los salones donde se están desarrollando clases en ese momento. Debido a la relevancia del mensaje, la institución quiere asegurarse de que toda clase reciba el mensaje, es decir, que dentro del plazo de duración de cada clase el mensaje sea emitido al menos una vez. Adicionalmente, quiere minimizar la cantidad total de emisiones.

Entonces el problema consiste en, dado un conjunto

$$\mathcal{I} = \{(s_1, f_1), (s_2, f_2), \ldots, (s_n, f_n)\}$$

donde (s_i, f_i) representa una clase que comienza en tiempo s_i y finaliza en tiempo f_i, determinar un conjunto de cardinalidad mínima, $\mathcal{A} = \{a_1, ..., a_k\}$, de tiempos en los que emitir el mensaje de modo que se cumple:

$$\text{para todo } (s_i, f_i) \in \mathcal{I} \text{ existe } a_j \in \mathcal{A} \text{ tal que } s_i \leq a_j \leq f_i.$$

(a) Dé un algoritmo ávido (*greedy*) eficiente que resuelve el problema.

(b) Demuestre que su algoritmo es correcto.
Sugerencia: realice la prueba mostrando que su algoritmo "está siempre por delante" con respecto a una solución arbitraria.

Solución:

(a) El algoritmo de la figura 1 resuelve el problema.

```
1 Ordenar el conjunto I según f_i
2 A = {}
3 while I ≠ {} do
4     a = mín{f : (s, f) ∈ I}
5     agregar a a A
6     eliminar de I los pares (s, f) tales que s ≤ a ≤ f
7 end
8 return A
```

Figura 1: Algoritmo *greedy* para resolver el probelma

(b) En primer lugar notamos que nuestro algoritmo termina, porque en cada iteración se elimina al menos una clase de $\mathcal{I}$, y hay una cantidad finita de ellas. Más aún, solo se eliminan clases cubiertas por algún elemento de $\mathcal{A}$, de modo que cuando termina todas las clases son cubiertas (decimos que una clase (s, f) es *cubierta* por un tiempo de emisión a si $s \leq a \leq f$). Solo resta probar que la cardinalidad de $\mathcal{A}$ es mínima.

Sea $\mathcal{A} = \{a_1, \ldots, a_k\}$ el conjunto devuelto por nuestro algoritmo para una determinada instancia del problema, y sea $\mathcal{O} = \{o_1, \ldots, o_m\}$ un conjunto solución para esa instancia. Ambos conjuntos están enumerados de menor a mayor: $a_i < a_{i+1}$, $1 \leq i < k$, y $o_j < o_{j+1}$, $1 \leq j < m$.

Notar que el algoritmo construye el conjunto $\mathcal{A} = \{a_1, \ldots, a_k\}$ incorporando los tiempos $a_1, \ldots, a_k$ en ese orden, ya que cada ejecución del paso 4 elige siempre el mínimo tiempo de finalización entre las tareas que no han sido eliminadas de $\mathcal{I}$ en iteraciones anteriores, y esta sucesión de mínimos no puede decrecer.

Mostramos a continuación que nuestro algoritmo está "por delante" de $\mathcal{O}$ en el siguiente sentido.

Proposición 1.
Para todo i, $1 \leq i \leq \text{mín}\{m, k\}$, se cumple que $a_i \geq o_i$.

Demostración.
Probaremos el enunciado por inducción en i, para $1 \leq i \leq \text{mín}\{m, k\}$.

Veamos que el enunciado se cumple para $i = 1$. Nuestro algoritmo comienza eligiendo como tiempo para el primer aviso, a_1, el final más temprano entre todas las clases; sea (s, f) una clase con tal tiempo de finalización, es decir $f = a_1$. Como $\mathcal{O}$ es una solución, debemos tener $o_1 \leq f$, porque de lo contrario (s, f) no sería cubierta por o_1 y tampoco por ninguno de los otros tiempos de emisión en $\mathcal{O}$, que son mayores que o_1. Esto muestra que $a_1 \geq o_1$.

Asumamos ahora que el enunciado se cumple para $i = j$, para cierto j, $1 \leq j < \text{mín}\{m, k\}$, y veamos que se cumple para $i = j + 1$.

Cuando se elige a_{j+1} en el paso 4, existe una clase, (s, f), con $f = a_{j+1}$, que no fue eliminada de $\mathcal{I}$ en iteraciones anteriores. Esto implica que se cumple $s > a_j$, porque en caso contrario tendríamos $s \leq a_j < a_{j+1} = f$, y (s, f) habría sido eliminada a más tardar al agregar a_j a $\mathcal{A}$. A partir de este hecho, combinado con la hipótesis de inducción que establece que $a_j \geq o_j$, obtenemos $s > o_j$, de modo que (s, f) no es cubierta por o_j ni ninguno de los tiempos de emisión anteriores en $\mathcal{O}$. Concluimos entonces que debemos tener $o_{j+1} \leq f$, porque de lo contrario (s, f) tampoco sería cubierta por o_{j+1}, ni ninguno de los tiempos de emisión posteriores en $\mathcal{O}$. Recordando que $a_{j+1} = f$, esto muestra que $a_{j+1} \geq o_{j+1}$, lo cual concluye la demostración de la proposición.

□

Veremos que la proposición que acabamos de demostrar implica que se cumple $k \leq m$, de donde, como $\mathcal{O}$ es arbitraria, se concluye que nuestro algoritmo construye una solución de cardinalidad mínima. En efecto, supongamos por absurdo que $k > m$. Cuando nuestro algoritmo elige a_{m+1} en el paso 4, existe una clase, (s, f), con $f = a_{m+1}$, que no fue eliminada de $\mathcal{I}$ en iteraciones anteriores. Siguiendo el mismo razonamiento que en el paso inductivo en la demostración de la proposición anterior, esto implica que se cumple $s > a_m$, porque de lo contrario (s, f) habría sido eliminada a más tardar al agregar a_m a $\mathcal{A}$. Pero por la proposición anterior tenemos $a_m \geq o_m$, por lo cual se cumple $s > o_m$ y en consecuencia (s, f) no es cubierta por $\mathcal{O}$. Llegamos a un absurdo, de donde concluimos que se debe cumplir $k \leq m$.

Ejercicio 3 (17 puntos)

Sea $G = (V, E)$ un grafo con costos en las aristas, todos positivos y distintos entre sí, y sea T un árbol de cubrimiento de costo mínimo (MST) para G.

Sea $(u, v) \in T$ y sea $G' = (V, E')$ el grafo que resulta de eliminar la arista (u, v) de G (es decir, $E' = E \setminus \{(u, v)\}$). Suponemos que G' es conexo. Sea T' un MST para G'.

Sabiendo que se cumple $T \setminus \{(u, v)\} \subseteq T'$, nos interesa obtener T' eficientemente a partir de T, (u, v), y G, para lo cual se propone el siguiente algoritmo:

1. Sea S la componente conexa de u en $(V, T \setminus \{(u, v)\})$.
2. Sea (u', v') la arista de menor costo en G' tal que $u' \in S$ y $v' \in V \setminus S$.
3. Hacer $T' = (T \setminus \{(u, v)\}) \cup \{(u', v')\}$.

(a) Demuestre que el algoritmo es correcto.

(b) Demuestre que el algoritmo admite una implementación cuyo tiempo de ejecución es $O(m)$, $m = |E|$.

Solución:

(a) Por (4.17) en el libro de referencia del curso, la arista (u', v') definida en el paso 2 pertenece a T'. En consecuencia, como tenemos $T \setminus \{(u, v)\} \subseteq T'$, se cumple que $(T \setminus \{(u, v)\}) \cup \{(u', v')\}$ es un subconjunto de T'. Por otra parte (u', v') no pertenece a T, porque formaría un ciclo junto con (u, v) ya que ambas unen S con $V \setminus S$. Por lo tanto, $(T \setminus \{(u, v)\}) \cup \{(u', v')\}$ tiene el mismo tamaño que T, que es igual al de T', y por lo tanto debemos tener $T' = (T \setminus \{(u, v)\}) \cup \{(u', v')\}$ como establece el paso 3.

> Una demostración alternativa podría ser la siguiente.
>
> Como tenemos $T \setminus \{(u, v)\} \subseteq T'$ y tanto T como T' tienen $n - 1$ elementos, T' se obtiene agregando una arista a $T \setminus \{(u, v)\}$. Como (V, T') es conexo, dicha arista debe conectar S con $V \setminus S$, es decir, debe ser de la forma (u', v') con $u' \in S$ y $v' \in V \setminus S$. Cualquier arista de esta forma que sea agregada a $T \setminus \{(u, v)\}$ da como resultado un árbol de cubrimiento, pero solo la de menor costo entre todas ellas resulta en un MST.

(b) El paso 1 puede implementarse construyendo una representación de listas de adyacencia del grafo $(V, T \setminus \{(u, v)\})$, lo cual requiere tiempo $O(n)$ para inicializar n listas vacías y agregar las $n - 2$ aristas de T que son distintas de (u, v), donde $n = |V|$. A continuación obtenemos S ejecutando BFS o DFS a partir de u, lo cual requiere tiempo $O(n)$ ya que se ejecuta sobre un árbol, que tiene $O(n)$ aristas. Como representación de S utilizamos un arreglo booleano de tamaño n que indica la pertenencia o no de un vértice a S (por ejemplo el arreglo que marca los nodos visitados en BFS o DFS puede jugar este rol).

Para el paso 2 recorremos todas las aristas (x, y) de G, y para cada una verificamos si se cumple que $x \in S$, $y \notin S$, y $(x, y) \neq (u, v)$. Nos quedamos con la de menor costo que cumple estas condiciones, cada una de las cuales se pueden verificar tiempo $O(1)$ usando la representación elegida para S. En total este paso requiere entonces tiempo $O(m + n)$.

El paso 3 implica unir, por ejemplo en una lista, las aristas de $T \setminus \{(u, v)\}$ con la arista (u', v') determinada en el paso 2, lo cual requiere tiempo $O(n)$.

En total tenemos una cantidad finita de pasos, cada uno de los cuales requiere tiempo $O(m + n)$, que es $O(m)$ porque G es conexo y por lo tanto tenemos $n \leq m + 1$. Esto demuestra que el tiempo de ejecución total es $O(m)$.