Examen de Programación 3
5 de febrero de 2021

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)

Considere el algoritmo de la figura 1 para formar un emparejamiento estable entre dos conjuntos de personas, M y W, ambos de tamaño n (versión estándar del problema, sin ninguna de las generalizaciones estudiadas). A diferencia del algoritmo de Gale-Shapley, cada vez que se elige un $m \in M$ libre, este realiza una ronda de propuestas comenzando siempre desde su opción de mayor preferencia.

```
1  Algorithm SimilGS
2      Inicialmente p está libre para todo p ∈ M ∪ W
3      while existe m ∈ M libre do
4          Elegir m ∈ M libre arbitrario, y hacer i = 1
5          while m está libre do
6              Sea w ∈ W en i-ésimo lugar en la lista de preferencias de m; m se propone a w
7              if w está libre then
8                  Emparejar m con w
9              else
10                 Sea m' la actual pareja de w
11                 if w prefiere a m antes que a m' then
12                     Separar a w de m' y emparejar m con w
13                 else
14                     w rechaza a m; Incrementar i
```

Figura 1: Algoritmo para formar un emparejamiento estable.

Analizando el algoritmo notamos que se cumplen los siguientes dos hechos (no necesita demostrarlos):

1. Para $w \in W$ se cumple que una vez que w forma una pareja con alguien, nunca vuelve a estar libre, y sus parejas solo mejoran a lo largo de la ejecución del algoritmo.
2. En ningún momento ocurre que alguien esté en pareja con más de una persona.

Se pide:

(a) Muestre que el algoritmo termina. Puede asumir sin demostración que el ciclo del paso 5 siempre termina.
Sugerencia: Para $w \in W$, $i \geq 1$, sea $R_i(w)$ la posición de la pareja de w en su lista de preferencia al final de la i-ésima iteración del ciclo del paso 3, o $R_i(w) = n + 1$ si w está libre. Definimos $R_0(w) = n + 1$ para todo $w \in W$. Considere la suma $\sum_{w \in W} R_i(w)$ como medida de avance.

(b) Muestre que el algoritmo produce un emparejamiento estable.

Solución:

(a) Consideramos la ejecución del algoritmo para cierta instancia del problema. Para $w \in W$, $i \geq 1$, denotamos con $R_i(w)$ a la posición de la pareja de w en su lista de preferencia al final de la i-ésima iteración del ciclo del paso 3, o $R_i(w) = n + 1$ si w está aún libre. Definimos $R_0(w) = n + 1$ para todo $w \in W$.

El ciclo del paso 5 termina con la formación de una pareja (m, w), ya sea en el paso 8 o en el paso 12. En cualquiera de los dos casos $R_i(w)$ es estrictamente menor que $R_{i-1}(w)$, $i \geq 1$. Además, la pareja de w' no se modifican para $w' \in W \setminus \{w\}$, por lo cual se cumple $R_{i-1}(w') = R_i(w')$. En consecuencia, la

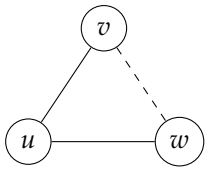
suma $\sum_{w \in W} R_i(w)$ disminuye estrictamente con cada iteración del ciclo del paso 3. Como se cumple $R_i(w) \geq 1$ para todo $w \in W$, esta suma está acotada inferiormente por n, lo cual implica que el ciclo termina en no más de $O(n^2)$ iteraciones porque el valor inicial de esta suma es $n(n+1)$.

(b) El resultado es un emparejamiento, por el hecho 2. Además es perfecto, ya que la condición de parada del ciclo del paso 3 implica que no existe $m \in M$ libre, y esto junto con el hecho 2 implica que tampoco existe $w \in W$ libre.

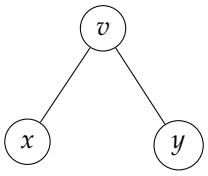
Vamos a probar que es estable. Sea m arbitrario y w su pareja al final del algoritmo. Todo $w' \in W$ al que m prefiere antes que a w lo rechazó durante la ejecución del ciclo del paso 5 en el cual m se emparejó con w. Por lo tanto, por el hecho 1, todos estos w' terminan en pareja con alguien a quien prefieren antes que a m y en consecuencia m no puede configurar ninguna inestabilidad.

Ejercicio 2 (35 puntos)

Un vértice de un grafo se denomina **punto de articulación** si el grafo resultante que se obtiene al borrarlo, junto a sus aristas incidentes, contiene más componentes conexas que el grafo original. Ver ejemplos en figura 2.



Grafo donde v no es punto de articulación; al eliminar v sigue habiendo una única componente conexa, $\{u, w\}$.



Grafo donde v es punto de articulación; al eliminar v se obtienen dos componentes conexas, $\{x\}$, $\{y\}$.

Figura 2: Dos ejemplos de grafos. Las aristas punteadas no pertenecen a un árbol de recorrido DFS.

(a) Sea v un vértice de un grafo G y supongamos que v tiene al menos dos hijos, x, y, en un árbol de recorrido DFS, T. Sean, x', y' dos vértices cualesquiera de los subárboles de x, y, respectivamente, en T (x' podría ser x o cualquiera de sus descendientes, y lo mismo para y' con respecto a y.) ¿Es posible que (x', y') sea una arista de G? Enuncie (sin demostrar) la propiedad teórica de árboles DFS que justifica la respuesta.

(b) Dé un algoritmo basado en **un solo** recorrido DFS para determinar si un vértice v dado como entrada es punto de articulación de un grafo G, no necesariamente conexo, con n vértices y m aristas. El tiempo de ejecución debe ser $O(m + n)$. **Reescriba DFS**. No es necesario que muestre la corrección de su algoritmo.

(c) Muestre que su algoritmo admite una implementación cuyo tiempo de ejecución es $O(m + n)$. Repita los argumentos vistos en el curso.

Solución:

(a) No es posible (ver enunciado (3.7) en libro del curso). Si (x', y') es una arista de G entonces uno de los vértices debe ser ancestro del otro en un árbol de recorrido DFS.

(b) El problema se puede resolver realizando un recorrido, basado en DFS, de los vértices del grafo a partir de vértice v. Si el vértice v tiene más de un hijo en el árbol resultante del recorrido entonces v es punto de articulación de G. Las aristas de un grafo que no pertenecen a un árbol de recorrido DFS solo vinculan vertices en relación ancestral del árbol. Por lo cual no existen aristas entre los hijos de

v ni entre sus descendientes. El algoritmo de la Figura 3 resuelve el problema.

```
1  Algorithm EsPuntoArticulacion(grafo (V, E), vertice v)
2      Hacer visitado [u] := false para todo u ∈ V
3      Hacer padre [u] := null para todo u ∈ V
4      canthijo := 0
5      Invocar DFS(v)
6      foreach u ∈ V do
7          if padre [u] = v then
8              canthijo := canthijo + 1
9      if canthijo > 1 then
10         return (El vértice es punto de articulación)
11     else
12         return (El vértice no es punto de articulación)

13 Procedure DFS(vertice u)
14     visitado [u] := true
15     foreach w : (u, w) ∈ E do
16         if visitado [w] = false then
17             padre [w] := u
18             Invocar DFS(w)
```

Figura 3: Algoritmo basado en DFS para determinar si v es punto de articulación.

(c) El algoritmo es implementado recursivamente. En el siguiente análisis se asume una representación de lista de adyacencia para el grafo, lo cual permite implementar el bucle del paso 15 iterando sobre la lista de adyacentes de v. El procedimiento DFS es invocado para cada vértice a lo sumo una vez, ya que el vértice pasa de no-visitado a visitado en el paso 14 y la condición del paso 16 evita que se repita la invocación. Para cada vértice se exploran sus vértices adyacentes, por lo que cada arista (x, y) se indaga dos veces, una vez desde x y otra desde y. O sea, si la cantidad de vértices adyacentes al vértice u es n_u, el bucle del paso 15 se ejecuta a lo sumo $\sum_{u \in V} n_u = 2m$ veces. Por lo tanto $O(m + n)$ es una cota superior para el tiempo total insumido en todas las invocaciones a DFS a partir de la invocación inicial del paso 5.

Por otra parte los pasos 2, 3 y el ciclo del paso 6 requieren tiempo $O(n)$ y el resto de los pasos requieren tiempo $O(1)$. Sumando el tiempo requerido para el paso 5, que es $O(m + n)$, obtenemos como resultado una cota superior $O(m + n)$ para el tiempo total de ejecución del algoritmo.

Ejercicio 3 (35 puntos)

Considere una red social de la cual forman parte un conjunto de n personas, $P = \{p_i\}_{1 \leq i \leq n}$. En esta red social, cada persona p_i tiene un conjunto de amigos, A_i, que también forman parte de la red. La relación de amistad cumple que si p_i es amigo de p_j, entonces p_j es amigo de p_i. Definimos el problema de decisión *RED-SOCIAL* de la siguiente manera: Dado un número natural k y una red social $R = (P, A)$, definida por un conjunto de personas $P = \{p_i\}_{1 \leq i \leq n}$ y sus respectivos conjuntos de amigos, $A = (A_1, A_2, \ldots, A_n)$, ¿existe un conjunto de personas de tamaño al menos k que sean todas amigas entre sí?

(a) Demuestre que *Independent Set* $\leq_P$ *RED-SOCIAL*. Repita cualquier argumento que utilice de los estudiados en el curso.

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

Solución:

(a) Consideramos una instancia arbitraria (G, k) de *Independent Set*, con $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_n\}$. Construimos una instancia (R, k') de *RED-SOCIAL*, con $R = (P, A)$, de la siguiente forma:

1. Definimos $P = V$.
2. Para cada i, $1 \leq i \leq n$, definimos A_i como el conjunto de todos los vértices de V que no son adyacentes a v_i.
3. Definimos $k' = k$.

Notar que, como la relación de adyacencia entre vértices de un grafo es simétrica, se cumple que la relación de amistad en esta definición de R también lo es, tal como exige la definición del problema *RED-SOCIAL*.

Observamos que la transformación se puede implementar de tal forma que requiere tiempo polinomial, ya que cada uno de sus tres pasos (una cantidad constante) requieren tiempo polinomial. En primer lugar, la construcción de A_i en el paso 2 puede implementarse mediante una búsqueda exhaustiva entre todos los vértices v_j, $1 \leq j \leq n$, incluyendo en A_i solo aquellos que no son adyacentes a v_i. Determinar si dos vértices son adyacentes requiere tiempo $O(n)$, y esta operación se realiza $O(n^2)$ veces (para i y j variando de 1 a n), por lo cual el tiempo de ejecución para el paso 2 es $O(n^3)$. Por otra parte, la construcción de un conjunto de tamaño n en el paso 1 requiere tiempo $O(n)$ y el paso 3 requiere tiempo $O(1)$. En conclusión, el tiempo total es polinomial en n, y por lo tanto también polinomial en el tamaño de la representación de la instancia de *Independent Set*.

Vamos a probar que G tiene un conjunto independiente de al menos k nodos si y solo si en R existe un conjunto de al menos k' personas amigas entre sí.

Supongamos que G tiene un conjunto independiente S de tamaño al menos k. Afirmamos que S es un conjunto de personas de R que son todas amigas entre sí (y tiene tamaño al menos k' porque $k' = k$). En efecto, para $v \in S$ y $w \in S$ arbitrarios, la definición de conjunto independiente implica que v y w no son adyacentes, por lo cual, por el paso 2, son amigos en R. Supongamos ahora que W es un conjunto de al menos k' personas de R que son amigas entre sí. Vamos a mostrar que W es un conjunto independiente en G (de tamaño al menos k, porque $k = k'$). Para $p \in W$ y $q \in W$ arbitrarios, el hecho de que sean amigas implica, por el paso 2, que p y q no son adyacentes. Como son arbitrarias, se cumple que ningún par de vértices en W son adyacentes entre sí, y por lo tanto W es un conjunto independiente.

Concluimos entonces que es posible resolver *Independent Set* a través de la transformación que acabamos de definir, que requiere tiempo polinomial, y una única invocación a un algoritmo que resuelve *RED-SOCIAL*. Esto demuestra que *Independent Set* $\leq_P$ *RED-SOCIAL*.

(b) Como *Independent Set* es $\mathcal{NP}$-Completo, por la parte anterior solo resta probar que *RED-SOCIAL* está en $\mathcal{NP}$. Definimos un certificado para una instancia con un conjunto $P = \{p_i\}_{1 \leq i \leq n}$ de personas como un subconjunto de P, representado como un arreglo booleano de tamaño n, donde la i-ésima posición del arreglo indica si p_i pertenece o no al certificado.

Afirmamos que el algoritmo de la figura 4 es un certificador eficiente para *RED-SOCIAL*. El algoritmo recibe una representación binaria de una instancia (R, k) del problema, y una tira de n bits que

representa un certificado C. El tiempo de ejecución del algoritmo es $O(n^3)$, ya que el ciclo del paso 3 se repite $O(n^2)$ veces y cada ejecución del paso 4 requiere tiempo $O(n)$ (para recorrer la el conjunto de no más de n amigos de q). Este tiempo de ejecución es polinomial en el tamaño de las entradas del algoritmo, tal como establece la definición de certificador eficiente.

```
1 Algorithm Certificador-RED-SOCIAL((R, k), C)
2     if |C| < k then return false
3     foreach (p, q) ∈ C × C, p ≠ q do
4         if p no está en el conjunto de amigos de q then return false
5     return true
```

Figura 4: Certificador eficiente para *RED-SOCIAL*.

Consideremos una instancia arbitraria (R, k) de *RED-SOCIAL*, con $R = (P, A)$, $P = \{p_i\}_{1 \leq i \leq n}$. Si (R, k) es una instancia SÍ, entonces existe un conjunto W de al menos k personas de R que son amigas entre sí. El certificado $C = W$ es de largo polinomial en el tamaño de la instancia (R, k), ya que su largo, n, es igual a la cantidad de conjuntos A_i que forman parte de R. Con este certificado, el certificador responde **true**, ya que la condición del paso 2 es falsa porque $|W| \geq k$, y todas las evaluaciones del paso 4 también son falsas porque las personas de W son todas amigas entre sí. 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, y todas las personas contenidas en C son amigas entre sí, porque de lo contrario alguna de las ejecuciones del paso 4 devolvería **false**.

Concluimos que (R, k) es una instancia SÍ si y solo si existe un certificado C de largo n (que es polinomial en el tamaño de la instancia) que hace que el algoritmo del la figura 4 responda **true** para las entradas (R, k), C.