Examen de Programación 3
21 de diciembre de 2023

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)

(a) Considere una recorrida DFS de un grafo **dirigido**, G, y el árbol de recorrida correspondiente, T. Muestre con un contraejemplo que la siguiente afirmación es falsa en general:
Sea (u, w) una arista de G que no es arista de T. Entonces uno de u o w es un ancestro del otro en T.

(b) Considere la adaptación del algoritmo DFS recursivo de la figura 1, que toma como entrada un grafo **dirigido** $G = (V, E)$ y devuelve una secuencia de vértices S. Demuestre que si G es acíclico entonces S es un ordenamiento topológico de G.
Sugerencia: Puede usar sin demostrar la siguiente propiedad: Para una determinada invocación $DFS(u)$, todos los nodos que son marcados "*Explorados*" entre la invocación y el fin de esa llamada son descendientes de u en el árbol de recorrida T.

(c) Demuestre que el tiempo de ejecución de este algoritmo es $O(n + m)$, donde $n = |V|$ y $m = |E|$. Repita cualquier argumento que use de los estudiados en el curso.

```
1  Algorithm Recorrida (G)
2      Sea S secuencia vacía
3      explorado [u] ← false para todo u ∈ V
4      foreach u ∈ V do
5          if not explorado [u] then
6              Invocar DFS(u)
7      return S
8  Procedure DFS(u)
9      explorado [u] ← true
10     foreach w tal que (u, w) ∈ E do
11         if not explorado [w] then
12             Invocar DFS(w)
13     Insertar u al inicio de S
```

Figura 1: Algoritmo DFS con secuencia salida.

Solución:

(a) Si el grafo tiene solo una arista (u, w) y primero se invoca $DFS(w)$ se obtiene un bosque compuesto por dos nodos aislados.

Otro ejemplo. Las aristas son (u, w) y (z, w). Sea cual sea que se invoque primero entre $DFS(u)$ y $DFS(z)$, la arista que une al segundo con w no es arista de árbol, ni existe relación de ancestros entre sus vértices.

(b) Notar en primer lugar que cada $u \in V$ es agregado a S una y solo una vez en el paso 13. Una vez ejecutado el paso 3, ningún otro paso asigna false a ningún elemento del arreglo explorado. Por lo tanto, como cada ejecución de $DFS(u)$ comienza por asignar explorado[u] = true, y las invocaciones a DFS en los pasos 6 y 12 solo se realizan si explorado[u] = false, entonces $DFS(u)$ se invoca a lo sumo una vez para cada nodo u. Además, como todos los nodos son recorridos en el ciclo del paso 4, $DFS(u)$ se invoca al menos una vez para cada nodo u.

Sea (u, w) una arista del grafo. Cuando se invoca la llamada $DFS(u)$ ocurre uno de estos dos casos excluyentes:

Todavía no se había invocado $DFS(w)$ Entonces $DFS(w)$ se invocará antes de que termine la llamada $DFS(u)$ (a más tardar cuando se evalúe la arista (u, w) al recorrer las aristas salientes de u, línea 12) y por lo tanto terminará antes, por lo que u quedará antes que w en S.

Ya se había invocado $DFS(w)$ Veamos que la llamada $DFS(w)$ ya había terminado por lo que w ya estaba en S, y por lo tanto u quedará antes.

Si $DFS(w)$ no hubiera terminado entonces u sería marcado "Explorado" entre la invocación y el fin de la llamada $DFS(w)$. Esto implicaría que u sería descendiente de w en el árbol (propiedad en sugerencia de letra), por lo que habría un camino desde w hasta u, que junto a la arista (u, w) formaría un ciclo. Esto es una contradicción ya que el grafo es acíclico.

Por lo tanto se cumple que para cualquier arista (u, w) el vértice u queda antes que w en la secuencia S, por lo que S es un ordenamiento topológico.

(c) Para S usamos una representación de lista enlazada y para G una representación de lista de adyacencia.

Como argumentamos en la parte anterior, $DFS(u)$ se invoca a lo sumo una vez para cada nodo u. Esto demuestra que el tiempo acumulado en todas las ejecuciones de los pasos 9 y 13 es $O(n)$, ya que cada ejecución de uno de estos pasos requiere tiempo $O(1)$.

Por otra parte, la cantidad total de ejecuciones del ciclo del paso 10 está acotado por $\sum_{u \in V} n_u$, donde n_u es el grado de salida de u. En esta suma, cada arista contribuye con una unidad al grado de salida de exactamente un vértice, por lo cual se cumple $\sum_{u \in V} n_u = |E|$. Usando una representación de lista de adyacencia para G, cada iteración del ciclo del paso 10, excluyendo el tiempo dentro de llamadas recursivas, requiere tiempo $O(1)$, de donde concluimos que el tiempo total insumido entre todas las ejecuciones de DFS es $O(n + m)$.

Finalmente observamos que el resto de los pasos que no involucran llamadas a DFS, es decir los pasos 1 – 5, requieren tiempo $O(n)$. En efecto, la construcción de la lista S en el paso 2 requiere tiempo $O(1)$, la inicialización del arreglo explorado en el paso 3 requiere tiempo $O(n)$, y el ciclo del paso 4 se repite n veces, donde cada iteración requiere tiempo $O(1)$ excluyendo el tiempo dentro de las llamadas a DFS que ya fueron contabilizadas.

Por lo tanto podemos descomponer el tiempo de ejecución del algoritmo en dos términos, uno para las ejecuciones de los pasos 1 – 5, que es $O(n)$, y otro para el tiempo total de ejecución de todas las llamadas a DFS, que es $O(n + m)$. Esta es una suma de una cantidad constante de términos que son $O(n + m)$ y por lo tanto el tiempo total de ejecución es $O(n + m)$.

Ejercicio 2 (35 puntos)

Consideramos una secuencia $S = x_1, \ldots, x_n$ de números enteros, no necesariamente ordenada y posiblemente con elementos repetidos. Una *subsecuencia creciente* de S es una secuencia $P = x_{i_1}, x_{i_2}, \ldots, x_{i_k}$, donde los índices están ordenados de forma ascendente, $i_1 < i_2 < \ldots < i_k$, y los elementos también, $x_{i_1} < x_{i_2} < \ldots < x_{i_k}$. Por ejemplo, para $S = 4, 2, 9, 2, 10, 6, 5, 7, 8, 3$, algunas posibles subsecuencias crecientes son $P = 2, 3$; $P' = 4, 9, 10$; $P'' = 4, 5, 7, 8$ y $P''' = 2, 6, 7, 8$.

Queremos encontrar una subsecuencia creciente de largo máximo. En el ejemplo, tanto P'' como P''' son de largo máximo (entre otras).

(a) Defina una relación de recurrencia que permita calcular el largo máximo de las subsecuencias crecientes de S. Justifique su respuesta explicando la procedencia de cada término.

Sugerencia: Considere una función $OPT(j)$ definida para $1 \leq j \leq n$ como el largo máximo de las subsecuencias crecientes de $x_1, x_2, \ldots, x_j$ que terminan en x_j (es decir, el largo máximo de las subsecuencias crecientes $P = x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ en las cuales $i_k = j$).

(b) Dé un algoritmo **iterativo** eficiente, usando la técnica de Programación Dinámica, para calcular el largo máximo de las subsecuencias crecientes de S.

(c) Dé un algoritmo eficiente para obtener una subsecuencia creciente de S de largo máximo. Puede usar estructuras de datos construidas en el algoritmo de la parte anterior.

Solución:

(a) Para $1 \leq j \leq n$ definimos $OPT(j)$ como en la sugerencia, y definimos $S_j = \{i : 1 \leq i < j, x_i < x_j\}$. Con esta notación, la siguiente ecuación define una recurrencia para $OPT(j)$, $1 \leq j \leq n$.

$$OPT(j) = \begin{cases} 1, & \text{si } S_j = \emptyset, \\ 1 + \max\{OPT(i) : i \in S_j\}, & \text{si } S_j \neq \emptyset \end{cases} \quad (1)$$

Si $S_j = \emptyset$, entonces x_j es menor o igual que todos sus predecesores, por lo cual la única subsecuencia creciente posible que termina en x_j está formada únicamente por x_j. Esta subsecuencia tiene largo 1, como se establece en el paso base. Por otra parte, si existe una subsecuencia creciente de largo mayor a 1, $P = x_{i_1}, x_{i_2}, \ldots, x_{i_k}$, con $i_k = j$, entonces $P' = x_{i_1}, x_{i_2}, \ldots, x_{i_{k-1}}$ es una subsecuencia creciente que termina en i_{k-1} y se cumple que $i_{k-1} \in S_j$, ya que tenemos $x_{i_{k-1}} < x_{i_k} = x_j$. Por lo tanto una subsecuencia creciente que termina en x_j de largo máximo se obtiene concatenando un elemento a una subecuencia creciente P' que termina en un elemento de S_j de largo máximo, como se establece en el paso inductivo.

Notar que para $j = 1$ se cumple $S_j = \emptyset$, y para $j > 1$ todos los elementos de S_j son menores que j, por lo cual la recurrencia está bien definida.

(b) El siguiente algoritmo de Programación Dinámica calcula el largo máximo de las subsecuencias crecientes de la secuencia $S = x_1, \ldots, x_n$. La función OPT se calcula en el arreglo *largos*. En el algoritmo, inicialmente se asigna para cada j el valor $largos[j] = 1$ determinado por el paso base. A continuación, en el ciclo del paso 3, este valor se sobrescribe si $S_j \neq \emptyset$.

Data: Secuencia $S = x_1, \ldots, x_n$
Result: Largo máximo de las subsecuencias crecientes de S

```
1 for j ← 1 to n do
2     largos[j] ← 1
3     for i ← 1 to j − 1 do
4         if S[i] < S[j] and largos[i] + 1 > largos[j] then
5             largos[j] ← largos[i] + 1
6         end
7     end
8 end
9 return máx{largos[j] : 1 ≤ j ≤ n}
```

(c) A continuación, se presenta un algoritmo iterativo eficiente que retorna una subsecuencia creciente de largo máximo de S usando el arreglo *largos*:

Data: S y el arreglo *largos*
Result: P: subsecuencia creciente de largo máximo de S

```
1 j ← arg máx{largos[j] : 1 ≤ j ≤ n}
2 P ← (S[j])
3 for i ← j − 1 downto 1 do
4     if S[i] < S[j] y largos[i] + 1 = largos[j] then
5         Agregar S[i] al inicio de P
6         j ← i
7     end
8 end
9 return P
```

Ejercicio 3 (35 puntos)

Sea $G = (V, E)$ grafo con $2d$ vértices, $d \geq 3$. Decimos que G es una *cometa* si se cumple:

1. G contiene un subgrafo S que es un *clique* de d vértices (o sea, un grafo completo),
2. los d vértices de $V \setminus S$ forman un camino simple con un extremo conectado a un único vértice de S.

El otras palabras, los vértices del grafo pueden nombrarse como $\{v_1, \ldots, v_d\}$ (los vértices que forman el clique), y $(v_{d+1}, \ldots, v_{2d})$ (los vértices que forman el camino simple), y las aristas del grafo son:

- (v_i, v_j) para cada i, j tal que $1 \leq i, j \leq d$, $i \neq j$,
- (v_i, v_{i+1}) para cada i tal que $d + 1 \leq i \leq 2d - 1$,
- (v_i, v_{d+1}) para algún i tal que $1 \leq i \leq d$.

En la figura 2 se muestra un ejemplo de un grafo cometa.

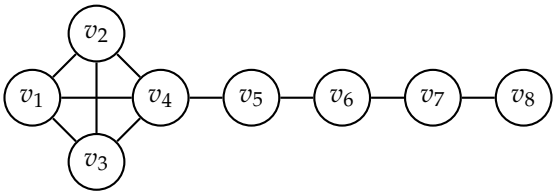

Figura 2: G es una cometa con $d = 4$: $\{v_1, v_2, v_3, v_4\}$ forma un *clique* y los restantes vértices un camino simple conectado a v_4.

Definimos el problema *COMETA* de la siguiente manera: dado un grafo G y un natural $d \geq 3$, ¿existe un subgrafo de G de al menos $2d$ vértices que sea una cometa? Por ejemplo, el grafo de la figura 3 con el natural $d = 3$ son una instancia SÍ del problema, mientras que el mismo grafo pero con $d = 4$ representan una instancia NO.

Demuestre que *COMETA* es $\mathcal{NP}$-completo. Una instancia del problema se representa mediante una matriz de adyacencia para G y un número natural d mayor o igual a 3. Enuncie cualquier argumento que utilice de los estudiados en el curso.

Sugerencia: utilice $CLIQUE_3$ como problema de referencia. El problema $CLIQUE_3$, que puede asumir que es $\mathcal{NP}$-completo, es: dados un grafo G y un natural $k \geq 3$, ¿existe un subgrafo de G de al menos k vértices que sea un clique? Puede asumir que una instancia de $CLIQUE_3$ se representa mediante una matriz de adyacencia para G y un número natural k mayor o igual a 3.

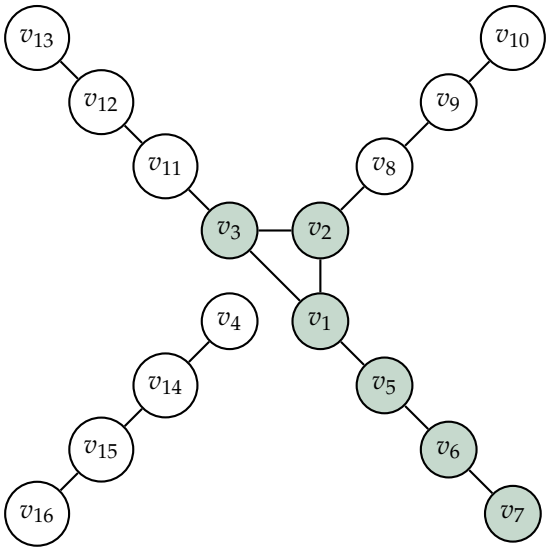

Figura 3: Los vértices sombreados de G forman un subgrafo que es una cometa con $d = 3$.

Solución:

Para demostrar que *COMETA* es $\mathcal{NP}$-completo se debe demostrar que:

- *COMETA* es $\mathcal{NP}$;
- todo problema en $\mathcal{NP}$ se puede reducir en tiempo polinomial a *COMETA*. Demostraremos que $CLIQUE_3$, que sabemos que es $\mathcal{NP}$-completo se puede reducir en tiempo polinomial a *COMETA*, con lo cual por propiedad transitiva quedará demostrado el punto.

Demostración de que *COMETA* es $\mathcal{NP}$

Vamos a demostrar que existe un algoritmo certificador eficiente, B, del problema *COMETA*. El algoritmo tiene como entrada la instancia s y un certificado t. Si el resultado de la ejecución de B es `SÍ` entonces s es una instancia `SÍ`. Si s es una instancia `SÍ` entonces existe al menos un certificado t con el cual el resultado de la ejecución de B es `SÍ`. El tamaño de t está acotado superiormente por un polinomio en el tamaño de s, $|t| \leq p(|s|)$, donde p es un polinomio. El tiempo de ejecución es polinomial.

La instancia s representa un grafo G y un entero d mayor o igual a 3. Vamos a suponer que el grafo está representado con una matriz de adyacencias. El certificado t es una lista de vértices de G.

El algoritmo verifica que t contiene $2d$ vértices distintos de G. Luego verifica que para cada par de los primeros d vértices, y por cada par de vértices consecutivos desde el $d-$ésimo hasta el último hay una arista en G. Devuelve `SÍ` si y solo si se cumple lo anterior.

```
1  Algorithm B (s,t)
       Data: s es un grafo G y un entero d mayor o igual a 3
2      Si t no contiene 2d vértices distintos terminar y devolver NO
3      foreach i, j ∈ {1..d}, i ≠ j do
4          Si (t_i, t_v) no es arista de G termina y devolver NO
5      end
6      foreach i ∈ d + 1..2d do
7          Si (t_{i-1}, t_i) no es arista de G terminar y devolver NO
8      end
9      return SI
10 end
```

Supongamos que el certificador devuelve `SÍ`. Eso significa que t representa $2d$ vértices distintos, los primeros d forman un clique en G, y los otros d forman un camino simple conectado al clique. Por lo tanto es una cometa de $2d$ vértices, y (G, d) es una instancia `SÍ` de `COMETA`.

Supongamos que $s = (G, d)$ es una instancia `SÍ` de `COMETA`. Entonces hay un subconjunto de $2d$ vértices distintos de G que forman una cometa. Si se construye t ubicando en las primeras d posiciones los vértices que forman el clique, con el vértice que está conectado al camino simple en la posición d, y los otros d de vértices de t son los que forman el camino simple, el certificador devuelve `SÍ`.

Si la instancia es `SÍ` el grafo tiene al menos $2d$ vértices, por lo que su tamaño es $\Omega(d^2)$. Por lo tanto el tamaño de t, que es $\Theta(d)$, es polinómico en el tamaño de s. El tiempo de ejecución del algoritmo consiste en verificar la longitud de t, lo que se hace en tiempo $O(d)$, verificar que los $2d$ vértices son diferentes, que también se puede cumplir en tiempo $O(d)$, y las verificaciones de la pertenencia de $\Theta(d^2)$ aristas al grafo, cada una de las cuales se puede hacer en tiempo constante. Por lo tanto el tiempo de ejecución es $O(d^2)$, que es polinómico en el tamaño de la entrada.

Demostración de que $CLIQUE_3$ se puede reducir en tiempo polinomial a *COMETA*

Se mostrará que existe una reducción polinomial del problema $\mathcal{NP}$-completo $CLIQUE_3$ a *COMETA*. Dada una instancia (G, k) de $CLIQUE_3$ nuestra reducción comienza verificando que $k \leq n$, donde n es la cantidad de vértices de G. Si esto no se cumple se responde NO. En caso contrario, se transforma en una instancia (G', d) de *COMETA* de la siguiente forma:

- Se toma d igual a k.
- El grafo G' es una copia de G, a la que por cada vértice v se le agrega un camino simple de d vértices, con uno de los extremos del camino adyacente a v. Los caminos son disjuntos.

Vamos a ver que

- una instancia de $CLIQUE_3$ se transforma en tiempo polinomial en una instancia de *COMETA*,
- una instancia `SÍ` de $CLIQUE_3$ se transforma en una instancia `SÍ` de *COMETA*,
- una instancia `NO` de $CLIQUE_3$ se transforma en una instancia `NO` de *COMETA*.

El grafo G' tiene $n(k+1)$ vértices y $m+nk$ aristas, donde m es la cantidad de aristas de G. Por lo tanto, para $k \leq n$, su tamaño es polinomial en el tamaño de G. La recorrida de G y construcción de G' se hacen en tiempo lineal en el tamaño de G', y la asignación de d es $O(1)$. Por la tanto la transformación se hace en tiempo lineal.

Demostramos que una instancia (G,k) `SÍ` de $CLIQUE_3$ se tranforma en una instancia (G',d) `SÍ` de *COMETA*. Si (G,k) es una instancia `SÍ` significa que existe un subconjunto U de k vértices que forman un clique. A ese subconjunto le corresponde por construcción un subconjunto U' de G' que también es un clique, de tamaño d. El subconjunto U' junto a cualquiera de los caminos simples de largo d agregados a cada uno de sus vértices forman un cometa de $2d$ vértices. Por lo tanto (G',d) es una instancia `SÍ` de `COMETA`.

Para demostrar que una instancia (G,k) `NO` de $CLIQUE_3$ se tranforma en una instancia (G',d) `NO` de *COMETA* vamos a demostrar el contrarecíproco, esto es, que si una instancia (G',d), resultado de la transformación de una instancia (G,k), es una instancia `SÍ` de *COMETA* entonces (G,k) es una instancia `SÍ` de $CLIQUE_3$. Si (G',d) es una instancia `SÍ` hay una cometa que incluye un subconjunto U' que es un clique de tamaño d. Los vértices de G' que no se corresponden a vértices de G son caminos simples. En cada uno de esos caminos hay un vértice con solo un adyacente, por lo que no puede pertenecer a un clique de tamaño mayor a 2. Los otros vértices del camino tiene dos adyacentes, que no son adyacentes entre sí, por lo que tampoco pueden pertenecer a un clique de tamaño mayor a 2. Por lo tanto el conjunto U' corresponde a un conjunto U de vértices de G. Por construcción U es un clique de tamaño k de G, por lo que (G,k) es una instancia S de $CLIQUE_3$.

Una transformación que no sirve es agregar un único camino simple, con un extremo adyacente a todos los vértices de G. Esto es porque (G,d) podría no ser una instancia `SÍ` de *CLIQUE* en la que hay un clique de tamaño $d-1$ adyacente a un camino simple, P, de d vértices. Al agregar el camino en el que un extremo es adyacente a todos los vértices de G se forma un clique de tamaño d, que junto con el camino P hacen que la instancia (G',d) sea una instancia `SÍ` de *COMETA*.