UNIVERSIDAD DE CONCEPCION
FACULTAD DE CIENCIAS
FISICAS Y MATEMATICAS
DEPARTAMENTO DE INGENIERIA MATEMATICA

Optimización III (525151)

Pauta de Evaluación 1

De los siguientes problemas entregue la solución de solamente **dos** de ellos. Cada pregunta vale el mismo puntaje, i.e. 30 ptos cada una. Puede usar sólo resultados vistos en clase.

P1) Determine el valor de verdad de las siguientes proposiciones. Justifique su respuesta.

a) Si $f(n) = O(g(n))$, entonces $2^{f(n)} = O(2^{g(n)})$.

b) Si Q y R son problemas de decisión y $R \in$ P, entonces $Q \leq_p R \iff Q \leq_p \overline{R}$.

c) El siguiente problema CYCLE-VERTEX es polinomial.

CYCLE-VERTEX: Dado $G = (V, E)$ un digrafo y $v \in V$ ¿Existe un ciclo en G que contiene a v?

P2) Se define el siguiente problema en grafos no dirigidos.

TREE-EXISTENCE: Dado V un conjunto de n vértices distintos y $E \subseteq \{\{u, v\} : u \neq v \in V\}$ un conjunto de aristas con extremos en V. ¿Existe un árbol $T = (V_T, E_T)$ tal que $V_T = V$ y $E \subseteq E_T$?

a) Se sabe que el número de árboles distintos con un conjunto dado de n vértices está dado por la fórmula $f(n) = n^{n-2}$. Muestre que $\forall k \in \mathbb{N} : \ f(n) = \Omega(n^k)$ y $f(n) \neq O(n^k)$.

b) Muestre que el siguiente algoritmo A resuelve el problema TREE-EXISTENCE, pero no es polinomial.

Algorithm A

Input: V un conjunto de n vértices distintos y
$E \subseteq \{\{u, v\} : u \neq v \in V\}$ un conjunto de aristas

```
for all T = (V_T, E_T) árbol con V_T = V do
    if E ⊆ E_T then
        return s
    end if
end for
return n
```

c) Dé un ejemplo de algoritmo polinomial que resuelva TREE-EXISTENCE.

P3) Sea $G = (V, E)$ un digrafo y $\sigma : V \to \{-1, 0, 1\}$ una función etiqueta en los nodos de G. Un vértice $v \in V$ se dice no negativo si $\sigma(v) \in \{0, 1\}$. (G, σ) se dice no negativo si todo ciclo de G contiene al menos un vértice no negativo. Se define el siguiente problema de decisión:

DIGRAFO NO NEGATIVO (DNN): Dado $G = (V, E)$ un digrafo y $k \in \mathbb{N}$ ¿Existe $\sigma : V \to \mathbb{R}$ una función etiqueta tal que (G, σ) es no negativo y el número de vértices no negativo es menor o igual a k?

a) Muestre que FEEDBACK VERTEX SET $\leq_p$ DIGRAFO NO NEGATIVO.

b) Deduzca que DIGRAFO NO NEGATIVO es NP-completo.

Solución:

P1) *a*) (10 ptos.) **Falso:** Sea $f(n) = 2n$ y $g(n) = n$. Como $\forall n \in \mathbb{N},\ f(n) = 2n \leq 2n = 2g(n)$, entonces $f(n) = O(g(n))$. Sin embargo,

$$\lim_n \frac{2^{f(n)}}{2^{g(n)}} = \lim_n \frac{2^{2n}}{2^n} = \lim_n 2^n = \infty \Longrightarrow 2^{f(n)} \neq O(2^{g(n)}).$$

(Recordar que por resultado visto en clase si $\lim_n \frac{\alpha(n)}{\beta(n)} = \infty$, entonces $\alpha(n) \neq O(\beta(n)) \ \wedge \ \alpha(n) = \Omega(\beta(n))$).

b) (10 ptos.) **Verdadero:** Sea Q y R problemas de decisión con $R \in$ P y que puede ser resuelto por un algoritmo polinomial A_R. Probemos que $Q \leq_p R \Longrightarrow Q \leq_p \overline{R}$. Sea $z \neq \bar{z} \in I_R,\ R(z) = s \wedge R(\bar{z}) = n$. Luego, mostremos que $R \leq_p \bar{R}$. En efecto, definamos la función $g : I_R \to I_{\bar{R}}(= I_R)$ definida por $\forall y \in I_R$:

$$g(y) = \begin{cases} z & \text{si } R(y) = n, \\ \bar{z} & \text{si } R(y) = s \end{cases}$$

Así se tiene que $\forall y \in I_R,\ R(y) = s \iff R(g(y)) = n \iff \bar{R}(g(y)) = s$ (notar que si $R(g(y)) = n$, entonces $g(y) = \bar{z} \Longrightarrow R(y) = s$). Además, g puede ser calculada mediante el siguiente algoritmo A_g.

Algorithm A_g

Input: $y \in I_R$

```
1: if A_R(y) = s then
2:     return z̄
3: else
4:     return z
5: end if
```

Obviamente, A_g es polinomial pues A_R es polinomial. Por lo tanto, g es una reducción polinomial y así $R \leq_p \bar{R}$. Finalmente, como $Q \leq_p R$ y $R \leq_p \bar{R}$, se tiene por resultado visto en clase (transitividad de $\leq_p$) que $Q \leq_p \bar{R}$.
La implicancia contraria, es decir $Q \leq_p \bar{R} \Longrightarrow Q \leq_p R$ se obtiene de forma análoga a lo anterior, considerando $\bar{R}$ en lugar de R.

c) (10 ptos.) Sea el siguiente algoritmo Cycle-vertex que recibe como entrada un digrafo $G = (V, E)$ y un vértice $v \in V$.

Algorithm Cycle-vertex(G,v)

Input: $G = (V, E)$ un digrafo y $v \in V$

```
1: for all u ∈ V do
2:     if PATH(G, u, v) = s ∧ (v, u) ∈ E then
3:         return s
4:     end if
5: end for
6: return n
```

Recordar que existe un ciclo en G que contiene al vértice v si y sólo si existe un camino de algún vértice $u \in V$ (eventualmente $u = v$) a v y $(v, u) \in E$. Por consiguiente, (G, v) es instancia afirmativa de CYCLE-VERTEX si y sólo si Cycle-vertex(G,v)=s, ie el algoritmo Cycle-vertex resuelve el problema CYCLE-VERTEX.
Por otro lado, el ciclo for en el peor de los casos realiza $O(|V|)$ iteraciones y en cada iteración ejecuta PATH cuyo tiempo de ejecución es $O(|V|^2)$ y la comparación $(v, u) \in E$ realiza $O(|E|)$ operaciones elementales. Luego el tiempo de ejecución del algoritmo es $O(|V|) \cdot (O(|V|^2) + O(|E|))$ que es $O(|V|) \cdot (O(|V|^2) + O(|V|^2)) = O(|V|^3)$ (pues $|E| = O(|V|^2)$). Como el tamaño de la entrada es $size(G, v) = size(G) + size(v) = O(|V|^2) + O(1) = O(|V|^2)$, entonces Cycle-vertex es una algoritmo polinomial.

P2) *a*) (10 ptos.) Sea $k \in \mathbb{N}$ fijo y definamos la función $g(n) = n^k, \forall n \in \mathbb{N}$. Luego, se tiene que:

$$\lim_n \frac{f(n)}{g(n)} = \lim_n \frac{n^{n-2}}{n^k} = \lim_n n^{n-2-k} = \infty \Longrightarrow f(n) = \Omega(g(n)) \ \wedge \ f(n) \neq O(g(n)).$$

Por lo tanto, $\forall k \in \mathbb{N},\ f(n) = \Omega(n^k) \wedge f(n) \neq O(n^k)$. (Recordar que $\lim_n n^{n-2-k} = \lim_{n \geq 2+k} n^{n-2-k} = \infty$.)

b) (10 ptos.) El algoritmo A recibe como entrada una instancia (V, E) del problema TREE-EXISTENCE y chequea para todos los árboles posibles $T = (V_T, E_T)$ cuyo conjunto de vértices es V, i. e. $V_T = V$, si alguno de ellos contiene las aristas E. Si alguno de estos árboles verifica la condición $E \subseteq E_T$, entonces el algoritmo A retorna s y retorna n en caso contrario. Luego, A resuelve el problema TREE-EXISTENCE, i. e. $\forall (V, E) \in I_{TREE-EXISTENCE},\ A(V, E) = s$ si y sólo si existe árbol $T = (V_T, E_T)$ tal que $V_T = V$ y $E \subseteq E_T$.
Por otro lado, el número de iteraciones del ciclo for es, en el peor caso, del orden del número de árboles distintos con conjunto de vértices V que es, según *a*), igual a n^{n-2} donde $|V| = n$. En cada iteración la comparación $E \subseteq E_T$ requiere $O(|E|) \cdot O(|E_T|) = O(|V|^4) = O(n^4)$ (pues $|E| = O(|V|^2)$ y $|E_T| = O(|V|^2)$). Luego, el tiempo de ejecución de A es $O(n^{n-2}) \cdot O(n^4) = O(n^{n+2})$. Como el tamaño de la entrada de A es $size(V) + size(E) = O(|V|) + O(|V|^2) = O(|V|^2) = O(n^2)$, se tiene que el tiempo de ejecución de A no es polinomial en el tamaño de su entrada, pues de serlo existiría $r \in \mathbb{N},\ n^{n-2} + n^4 = O((n^2)^r) = O(n^{2r}) \Longrightarrow n^{n-2} = O(n^{2r})$, lo que es contradictorio con *a*).
(notar que $n^{n-2} + n^4 = O(n^{2^r}) \Longleftrightarrow \exists c > 0, n_0 \in \mathbb{N},\ \forall n \geq n_0 :\ (n^{n-2}) \leq n^{n-2} + n^4 \leq c(n^{2r})$, entonces $n^{n-2} = O(n^{2r})$).

c) (10 ptos.) Sea el siguiente algoritmo Tree-existence, donde los algoritmos PATH y Tree son los vistos en clase para grafos no dirigidos.

Algorithm Tree-existence(V,E)

Input: V un conjunto de n vértices distintos y $E \subseteq \{\{u, v\} : u \neq v \in V\}$ un conjunto de aristas

```
1:  G ← (V, E)
2:  if Tree(G)=s then
3:     return s
4:  end if
5:  for all u, v ∈ V, u ≠ v do
6:     if PATH(G,u,v)=n then
7:        G ← G ∪ {u, v}
8:     end if
9:  end for
10: if Tree(G)=s then
11:    return s
12: else
13:    return n
14: end if
```

Veamos que el algoritmo Tree-existence resuelve el problema TREE-EXISTENCE. Sea (V, E) una instancia de TREE-EXISTENCE.
Si Tree-existence(V,E)=s, entonces Tree(G)=s en la línea 2 o en la línea 10. En cualquier caso $G = (V, E)$ es un árbol que contiene a E, luego TREE-EXISTENCE(V, E)=s. Supongamos ahora que TREE-EXISTENCE(V, E)=s, entonces existe un árbol $T = (V_T, E_T)$ tal que $V_T = V$ y $E \subseteq E_T$. Si el grafo de entrada $G = (V, E)$ definido en la línea 1 es árbol, entonces el algoritmo Tree-existence con entrada (V, E) retorna s en la línea 3. Supongamos ahora que $G = (V, E)$ no es árbol. Como sabemos que existe un árbol $T = (V_T, E_T)$ tal que $V_T = V$ y $E \subseteq E_T$, entonces $G = (V, E)$ de la línea 1 no tiene ciclo pues de haber un ciclo formado por las aristas de E, entonces T tendría también un ciclo ya que contiene las aristas de E, lo que es contradictorio con que T sea un árbol. Veamos ahora que G después del ciclo for tampoco tiene ciclo en este caso. Supongamos por el contrario que existe C ciclo en G. Como el conjunto E no forma ciclo, entonces existe $\{u, v\} \notin E$ que pertenece al ciclo C y que fue agregado por la línea 7 a G en algún momento de la iteración del for. Sin pérdida de generalidad, supondremos que $\{u, v\}$ es la última arista en C que fue agregada a G según la línea 7. Luego, justo antes de agregar $\{u, v\}$ y crear el ciclo C había un camino en C entre u y v, lo que es contradictorio con la línea 6. Por lo tanto, no existe ciclo en G después del ciclo for. Por otro lado, G resultante del ciclo for es también obviamente conexo (existe camino entre cualquier par de vértices distintos). En consecuencia, G es árbol que contiene a E y así en la línea 10 el algoritmo Tree-existence retornará s. En resumen, el algoritmo Tree-existence resuelve el problema TREE-EXISTENCE.

Por otro lado, la definición de G en la línea 1 requiere $O(|V|)^2$ operaciones elementales mediante por ejemplo la definición de la matriz de adyacencia de G. La línea 2 requiere ejecutar el algoritmo Tree que como vimos en clase realiza $O(|V|^4)$ operaciones elementales. El ciclo for realiza en el peor caso $O(|V|^2)$

iteraciones, y en cada iteración se ejecuta el algoritmo PATH que realiza $O(|V|^2)$ operaciones elementales más la operación de asignación de la línea 7 que significa cambiar dos valores en la matriz de adyacencia, i. e. $O(1)$ operaciones elementales. Por último, en la línea 10 se ejecuta el algoritmo Tree que realiza $O(|V|^4)$ operaciones elementales. En resumen, el número de operaciones elementales del algoritmo es $O(|V|)^2 + O(|V|^4) + O(|V|^2) \cdot (O(|V|^2) + O(1)) + O(|V|^4) = O(|V|^4)$ operaciones elementales. Como el tamaño de la entrada (V, E) es $size(V, E) = |V| + |E| = O(|V|^2)$, entonces el algoritmo Tree-existence es polinomial.

P3) *a*) (15 ptos.) Sea $f : I_{FVS} \to I_{DNN}$ la función definida por: $\forall G = (V, E)$ digrafo y $k \in \mathbb{N}$, $f(G, k) = (G, k)$, i. e. f es la función identidad donde $I_{FVS} = I_{DNN}$ son los conjuntos de instancias del problema FEEDBACK VERTEX SET (FVS) y DIGRAFO NO NEGATIVO (DNN) respectivamente, y que en este caso son iguales. Como f es la función identidad entonces el algoritmo que recibe como entrada una instancia (G, k) y da como salida lo mismo es $O(1)$ y por lo tanto polinomial. Mostremos ahora que $\forall (G, k) \in I_{FVS},\ FVS(G, k) = s \iff DNN(G, k) = s$. En efecto, supongamos que existe $V' \subseteq V$ un FVS de G con $|V'| \leq k$. Luego, definiendo la función etiqueta $\sigma : V \to \{-1, 0, 1\}$ por:

$$\forall v \in V, \quad \sigma(v) = \begin{cases} 1 & \text{si } v \in V', \\ -1 & \text{si } v \notin V. \end{cases}$$

se tiene que todo ciclo de G contiene un vértice de V' y por ende un vértice no negativo. Así, (G, σ) es un digrafo no negativo. Además, el número de vértices no negativos es $|V'|$ que por hipótesis es menor o igual a k. Por lo tanto, (G, k) es instancia afirmativa de DNN. Supongamos ahora que (G, k) es instancia afirmativa de DNN. Luego, existe una función etiqueta $\sigma : V \to \{-1, 0, 1\}$ tal que (G, σ) es un digrafo no negativo y el número de vértices no negativos es menor o igual a k. Definamos, $V' := \{v \in V :\ \sigma(v) \in \{0, 1\}\}$. Luego, como cada ciclo de G contiene un vértice no negativo, entonces V' es un FVS de G. Además, como el número de vértices no negativos es menor o igual a k, entonces $|V'| \leq k$. De aquí, (G, k) es instancia afirmativa de FVS.
Por lo tanto, f es una reducción polinomial y así FEEDBACK VERTEX SET (FVS) $\leq_p$ DIGRAFO NO NEGATIVO (DNN).

b) (15 ptos.) De lo visto en clase sabemos que FEEDBACK VERTEX SET es NP-completo y en particular es NP-Hard, de *a*) se tiene entonces que DIGRAFO NO NEGATIVO es también NP-Hard. Probemos ahora que DIGRAFO NO NEGATIVO es NP. De la parte *a*) se tiene que como la función $f : I_{FVS} \to I_{DNN}$ es la identidad, entonces la misma función $f : I_{DNN} \to I_{FVS}$ (con $I_{DNN} = I_{FVS}$) definida por $\forall G = (V, E)$ digrafo y $k \in \mathbb{N}$, $f(G, k) = (G, k)$ es una reducción polinomial de DIGRAFO NO NEGATIVO en FEEDBACK VERTEX SET., i. e. DIGRAFO NO NEGATIVO $\leq_p$ FEEDBACK VERTEX SET. Luego, por resultado visto en clase, DIGRAFO NO NEGATIVO es NP y por consiguiente es NP-completo.
Otra forma de probar que DIGRAFO NO NEGATIVO es NP es construir un certificado polinomial análogo al construído para el problema FVS.