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

Optimización III (525551)

Ejercicios de algoritmos polinomiales

P1) *a*) Construya un algoritmo polinomial que resuelva el problema:
SCONNECTED: Dado $G = (V, A)$ un grafo dirigido ¿Es G fuertemente conexo? ¿Cuántas componentes fuertemente conexas posee?

b) A partir de la parte **a)**, construya un algoritmo polinomial que resuelva el problema:
ACYCLIC DIRECTED: Dado $G = (V, A)$ un grafo dirigido ¿Es G acíclico?

P2) Para cada uno de los siguientes problemas, muestre un algoritmo polinomial que lo resuelva. Justifique su respuesta.

a) **TREE**: Dado $G = (V, E)$ un grafo no dirigido ¿Es G un árbol?

b) **PATH-VERTEX**: Dado $G = (V, E)$ un grafo dirigido y los vértices $u, v, w \in V$ ¿Existe un camino de u a v en G que no pase por w?

P3) 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(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: for all T = (V_T, E_T) árbol con V_T = V do
2:     if E ⊆ E_T then
3:         return s
4:     end if
5: end for
6: return n
```

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

Solución:

P1) *a*) Una forma de resolver el problema es usar el algoritmo de exploración de grafos para determinar los nodos alcanzables desde un vértice inicial cualquiera v_1 y usar un algoritmo que genere una exploración en reversa al de exploración de grafos para determinar los nodos que alcanzan v_1, para ello, es suficiente cambiar la palabra sucesor por predecesor en el algoritmo de exploración visto en clase.
Así, si ambos conjuntos de vértices marcados que son alcanzados por v_1 y llegan a v_1 son iguales al conjunto completo de vértices entonces el grafo sería fuertemente conexo y retornar (s, 1 , V), donde la segunda y tercera componente de la salida resuelve la segunda pregunta. En cambio, si este no lo fuera se construye el conjunto de vértices S que contiene los nodos no marcados y que no pertenezcan a una componente fuertemente conexa y que denotaremos por $comp[i]$. Esto se ejecutará hasta definir todos los vértices en una componente, lo cual ocurrirá hasta que $S = \emptyset$. Tras salir del ciclo while se retorna n (no es un grafo fuertemente conexo), el número de componentes fuertemente conexas y los vértices de las componentes obtenidas.

Algorithm $SCONNECTED(G)$

Input: $G = (V, A)$ grafo dirigido donde $V = \{v_1, v_2, ..., v_n\}$

```
1: (M1, pred) ← ExploracionGrafo(G, v1)
2: (M2, suc) ← ExploracionReversaGrafo(G, v1)
3: if V = M1 ∧ V = M2 then
4:    return s,1,V
5: else
6:    i ← 1
7:    comp[i] ← M1 ∩ M2
8:    S ← V \ comp[i]
9:    while S ≠ φ do
10:      Seleccionar v ∈ S
11:      (M1, pred) ← ExploracionGrafo(G, v)
12:      (M2, suc) ← ExploracionReversaGrafo(G, v)
13:      i ← i + 1
14:      comp[i] ← M1 ∩ M2
15:      S ← S \ comp[i]
16:   end while
17: end if
18: return n, i, comp[:]
```

Respecto al tiempo de ejecución, tanto el algoritmo ExploracionGrafo y ExploracionReversaGrafo se ejecutan en tiempo $O(|V| + |E|) = O(|V|^2)$, la línea 3 se ejecuta en el peor caso en tiempo $O(|V|^2)$, las líneas 6,10 y 13 son consideradas operaciones elementales, las operaciones en las líneas 7, 8, 14 y 15 se ejecutan en el peor caso $O(|V|^2)$ cada una y el ciclo while se ejecuta en tiempo $O(|V|)$. Por todo lo anterior, el tiempo de ejecución del algoritmo SCONNECTED viene dado por:

$$\begin{aligned} O(|V| + |E|) + O(|V|^2) + O(|V|^2) + O(|V|) \cdot [O(|V| + |E|) + O(|V|^2)] \\ &= O(|V|^2) + O(|V|) \cdot O(|V|^2) \\ &= O(|V|^3) \end{aligned}$$

Por lo tanto, el tiempo de ejecución del algoritmo SCONNECTED es polinomial respecto al tamaño de entrada ($O(size(G)) = O(|V|^2)$).

b) Notemos que el problema de existencia de un ciclo en G es equivalente a probar la existencia de una componente fuertemente conexa no trivial, es decir, de al menos dos vértices. Esto puede ser verificado probando que G es fuertemente conexo o posee menos de $n = |V|$ componentes fuertemente conexas. A continuación, se presenta un algoritmo que verifica lo anterior.

Algorithm $ACYCLIC(G)$

Input: $G = (V, A)$ grafo dirigido, donde $V = \{v_1, v_2, ..., v_n\}$ y $n \geq 2$.

1: $(q, i, comp[:]) \leftarrow SCONNECTED(G)$
2: **if** $i < n$ **then**
3: **return** n
4: **else**
5: **return** s
6: **end if**

Respecto al tiempo de ejecución, notemos que las líneas 2, 3 y 5 se consideran operaciones elementales por lo cual el tiempo de ejecución viene dado exclusivamente por la línea 1, la cual se ejecuta en tiempo $O(|V|^3)$ e implicando que ACYCLIC sea polinomial respecto a su tamaño de entrada ($O(size(G)) = O(|V|^2)$).

P2) *a*) Recordar que dado $G = (V, E)$ un grafo no dirigido, se tiene que:

$$G \text{ es árbol } \iff G \text{ es conexo y sin ciclos } \iff G \text{ es conexo y } |E| = |V| - 1.$$

Luego, un algorimo polinomial (pero sin duda no optimal) que resuelve el problema TREE es el siguiente:

Algorithm Tree(G)

Input: $G = (V, E)$ un grafo no dirigido

1: **for all** $u, v \in V,\ u \neq v$ **do**
2: **if** $PATH(G, u, v) = n$ **then**
3: **return** n
4: **end if**
5: **end for**
6: **if** $|E| \neq |V| - 1$ **then**
7: **return** n
8: **end if**
9: **return** s

Luego, obviamente Tree(G)=s si y sólo si G es árbol. Por otro lado, el ciclo for de la línea 1 se ejecuta en el peor caso $O(|V|^2)$ veces. En cada iteración, se ejecuta el algoritmo PATH que es $O(|V|^2)$. Luego, el ciclo for ejecuta en el peor caso $O(|V|^4)$ operaciones elementales. Por otro lado, la comparación de la línea 6 requiere contar los elementos de E y de V. Ambas operaciones se pueden hacer en el peor caso con $O(|E|)$ y $O(|V|)$ operaciones elementales, respectivamente. Por lo tanto, el tiempo de ejecución de Tree es $O(|V|^4) + O(|E|) + O(|V|) = O(|V|^4)$, i.e es polinomial en el tamaño de G ($size(G) = O(|V|^2)$).

b) El problema PATH-VERTEX puede ser resuelto de muchas formas distintas. A continuación, un posible algoritmo que resuelve PATH-VERTEX en tiempo polinomial:

Algorithm Path-vertex(G,u,v,w)

Input: $G = (V, E)$ un grafo dirigido y $u, v, w \in V$

1: $G' \leftarrow G - w$
2: **if** $PATH(G', u, v) = s$ **then**
3: **return** s
4: **else**
5: **return** n
6: **end if**

Luego, Path-vertex$(G, u, v, w) = s \iff$ existe un camino de u a v en $G' = G - w \iff$ existe un camino de u a v en G que no pasa por w.
Observar que la línea 1 requiere simplemente eliminar una fila y una columna de la matriz de adyacencia de G, i.e. se puede hacer en $O(|V|)$ operaciones elementales. Por otro lado, PATH requiere $O(|V|^2)$ operaciones elementales. Por lo tanto, el tiempo de ejecución de Path-vertex es $O(|V|^2)$, ie. es un algoritmo polinomial.

P3) *a*) 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) 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 $r \in \mathbb{N}$, $n^{n-2} \cdot n^4 = n^{n+2} = O((n^2)^r) = O(n^{2r}) \Longrightarrow n^{n-2} = O(n^{n+2}) = O(n^{2r})$, lo que es contradictorio con *a*).

c) 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 elementa-

les. Como el tamaño de la entrada (V, E) es $size(V, E) = |V| + |E| = O(|V|^2)$, entonces el algoritmo Tree-existence es polinomial.