

## Ejercicio 1

Supón que un algoritmo  $A$ , que resuelve un problema  $\Pi$ , tiene un tiempo de ejecución de  $T_A(n) = 20n^3 + 10n + 16$ , además supón que un algoritmo  $B$ , que resuelve el mismo problema  $\Pi$ , tiene un tiempo de ejecución de  $T_B(n) = 22n^2 \log_2(8n)$ . ¿Qué algoritmo es mejor?. Justifica tu respuesta respondiendo las siguientes preguntas:

- ¿Para qué valores de  $n$  el algoritmo  $A$  es mejor que el  $B$  ?
- ¿Para qué valores de  $n$  el algoritmo  $B$  es mejor que el  $A$  ?

Inicialmente veamos cómo se comportan estas dos funciones para los primeros 30 números naturales:

n	T_A	T_B	
1	46	66	Es mejor T_A
2	196	352	Es mejor T_A
3	586	907,8225751	Es mejor T_A
4	1336	1760	Es mejor T_A
5	2566	2927,060452	Es mejor T_A
6	4396	4423,290301	Es mejor T_A
7	6946	6260,328606	Es mejor T_B
8	10336	8448	Es mejor T_B
9	14686	10994,80635	Es mejor T_B
10	20116	13908,24181	Es mejor T_B
11	26746	17195,00697	Es mejor T_B
12	34696	20861,1612	Es mejor T_B
13	44086	24912,23487	Es mejor T_B
14	55036	29353,31442	Es mejor T_B
15	67666	34189,10845	Es mejor T_B
16	82096	39424	Es mejor T_B
17	98446	45062,08874	Es mejor T_B
18	116836	51107,22541	Es mejor T_B
19	137386	57563,04031	Es mejor T_B
20	160216	64432,96724	Es mejor T_B
21	185446	71720,26364	Es mejor T_B
22	213196	79428,02788	Es mejor T_B
23	243586	87559,21404	Es mejor T_B
24	276736	96116,64481	Es mejor T_B
25	312766	105103,0226	Es mejor T_B
26	351796	114520,9395	Es mejor T_B
27	393946	124372,8858	Es mejor T_B
28	439336	134661,2577	Es mejor T_B
29	488086	145388,3644	Es mejor T_B

Figure 1: Pueden ver la tabla original dando click aquí [aqui](#)

Como podemos observar, cuando  $1 \leq n \leq 6$  sucede que el tiempo de ejecución de  $T_A$  es menor que el de  $T_B$ , lo que implica que dentro de este intervalo de valores el algoritmo  $A$  es mejor que el  $B$ . Por otro lado, cuando  $n \geq 7$  sucede que el tiempo de ejecución de  $T_B$  es menor que el de  $T_A$ , lo que implica que en ese intervalo de valores el algoritmo  $B$  es mejor que el algoritmo  $A$ .

Por lo que nuestra respuesta sería que el algoritmo  $B$  es mejor. Esto se debe al siguiente análisis: El algoritmo  $A$  es mejor que el algoritmo  $B$  sólo en seis valores de  $n$  (donde el

número de posibles valores para  $n$  es la cardinalidad de  $\mathbb{N}$ ). Si nuestras entradas siempre fueran valores menores o iguales a 6, entonces claramente la respuesta sería que el algoritmo  $A$  es mejor; pero esa condición no la podemos asegurar en ningún caso y como el algoritmo  $B$  será el mejor en la mayoría de los casos, entonces nuestra conclusión es que en general el algoritmo  $B$  es el mejor.

---

## Ejercicio 2

¿Cuáles de las siguientes afirmaciones son verdades y cuáles falsas? Justifica tu respuesta en cada caso

- (a) Si  $f(n) = n^k$  y  $g(n) = c^n$ , para algunas constantes  $c > 1, k \geq 0$ , entonces  $g(n) = \Omega(f(n))$  (FALSO)

Sabemos que  $\Omega(f(n)) = g(n)$  si existen  $c', n_0$  constantes positivas tal que

$$\forall n \geq n_0 : 0 \leq c'f(n) \leq g(n)$$

es decir que a partir de  $n_0$  la función  $c'f$  queda dominada por la función  $g$ .

Entonces basta encontrar  $n_0$  y  $c'$  constantes positivas tales que cumplan que para toda  $n \geq n_0$  se cumpla que:

$$c'(n^k) \leq c^n$$

- **CASO**  $c \geq k$  : Sea  $n_0 = 1$  y  $c' = 1$ .

Notemos ahora que si  $c \geq k$  siempre podemos afirmar que

$$n^k \leq c^n$$

- **CASO**  $c < k$  : En este caso, a menos que  $c'$  sea variable (por ejempplo  $c' = c^n$ ) tendríamos que la desigualdad buscada se cumpliría, pero dado que  $c'$  tiene que ser constante no es posible).

Por tanto podemos concluir que la afirmación es verdadera sólo cuando  $c \geq k$ .

(b) Si  $f(n) = n!$  y  $g(n) = 2^n$ ,  $f(n) = O(g(n))$  (FALSO)

Sabemos que  $f(n) = O(g(n))$  si existen  $c, n_0$  constantes positivas tal que

$$\forall n \geq n_0 : 0 \leq |f(n)| \leq c|g(n)|$$

es decir que a partir de  $n_0$  la función  $c|g|$  domina a la función  $|f|$ .

Para demostrar que es falso basta demostrar que para toda  $c$  y para toda  $n$  se tiene que

$$|g(n)| = (2^n) < c|f(n)| = c|n!|$$

Sea pues  $c$  una constante positiva mayor a 1 y demostremos por inducción que

$$|2^n| < c|n!|$$

PASO BASE:  $n = 0$

En este caso  $2^0 = 1$  y  $0! = 1$  por lo que dado que  $c$  es positiva (mayor a 1) se cumple que

$$1 = |2^0| < c|1| = c$$

HIPÓTESIS DE INDUCCIÓN: Supongamos que

$$|2^n| < c|n!|$$

y demostremos que

$$|2^{n+1}| < c|(n+1)!|$$

Observemos que  $2^{n+1} = 2^n(2)$  y  $(n+1)! = n!(n+1)$ , usando la H.I. tenemos que

$$|2^n| < c|n!|$$

y dado que  $2 < n+1$  para toda  $n > 1$  y  $c$  es positiva, podemos afirmar que

$$|2^{n+1}| \leq c|(n+1)!|$$

que es lo que queríamos.  $\square$

(c) Si  $f(n) = O(n)$  y  $g(n) = \Theta(n)$ , entonces:

- $f(n) + g(n) = O(n)$  (VERDADERO)

Dado que  $f$  está acotada superiormente por algún múltiplo de la función lineal  $n$ , mientras que  $g$  está acotada tanto superior como inferiormente por múltiplos de la función lineal  $n$  (es decir,  $g$  y  $n$  crecen de la misma forma), por tanto la función que suma  $f$  y  $g$  debe estar acotada superiormente por algún múltiplo de la función  $n$ .

- $f(n)g(n) = O(n^2)$  (VERDADERO)

Dado que  $f$  está acotada superiormente por algún múltiplo de la función lineal  $n$ , mientras que  $g$  está acotada tanto superior como inferiormente por múltiplos de la función lineal  $n$  (es decir,  $g$  y  $n$  crecen de la misma forma), por tanto la función que multiplica  $f$  y  $g$  debe estar acotada superiormente por al más algún múltiplo de la función  $n^2$ .

- $f(n) - g(n) = O(1)$  (FALSO)

Sea  $f(n) = 2n$  y  $g(n) = n$ . Así,  $f(n) - g(n) = 2n - n = n$  y  $n \neq O(1)$  pues la función 1 no domina a ningún múltiplo de la función  $n$ .

(d) Existen funciones  $f, g$  tales que no se cumple que  $f(n) = O(g(n))$  ni  $f(n) = \Omega(g(n))$

Para que la afirmación sea falsa tenemos que demostrar que para toda función  $f, g$  se tiene o bien que  $f(n) = O(g(n))$  o bien que  $f(n) = \Omega(g(n))$  o ambas.

Para demostrar que es verdadera tenemos que mostrar que existen funciones  $f, g$  tales que no se cumple que  $f(n) = O(g(n))$  ni  $f(n) = \Omega(g(n))$ .

El ejemplo que la ayudante da en clase solo muestra que existen funciones  $f, g$  ( $f = g = n^2$ ) tales que  $f(n) = O(g(n))$  y  $f(n) = \Omega(g(n))$  pero esa no es la negación de la afirmación d) (la negación del existe  $x$  tal que  $P$  es para todo  $x$  no  $P$ ) sino la negación de la afirmación siguiente:

Para toda  $f$  y  $g$  se tiene que no es cierto que  $f(n) = O(g(n))$  o no es cierto que  $f(n) = \Omega(g(n))$ . ...(\*)

---

Así, suponiendo que el enunciado d) era (\*) (y para que sea congruente con el ejemplo que dio la ayudante) la afirmación es FALSA y el contraejemplo efectivamente es el que se vió en clase:

$$g(n) = f(n) = n^2.$$

Observemos que  $f(n) = O(g(n))$  pues existen constantes  $c = 1, n_0 = 1$  tal que

$$\forall n \geq n_0 : 0 \leq |f(n)| \leq |f(n)|$$

De manera análoga observemos que  $f(n) = \Omega(g(n))$  pues existen constantes  $c = 1, n_0 = 1$  tal que

$$\forall n \geq n_0 : 0 \leq f(n) \leq f(n)$$

Por tanto tenemos que  $f(n) = O(g(n))$  y  $f(n) = \Omega(g(n))$ .

### Ejercicio 3

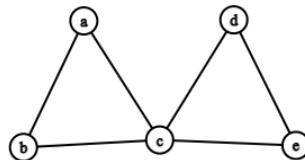
Da ejemplos de gráficas que cumplan lo siguiente:

- **Contiene un Ciclo Euleriano, pero no Ciclos Hamiltonianos**

La siguiente gráfica tiene un ciclo euleriano pues el camino CE pasa por cada arista una y solo una vez:

$CE = ca, ab, bc, cd, de, ec$

Sin embargo no tiene ciclos hamiltoniano, ya que cualquier camino que visite todos los vértices tendría que visitar el vérifice central (c) al menos 2 veces.

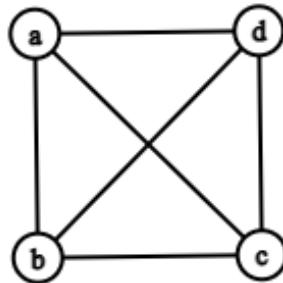


- **Contiene un Ciclo Hamiltoniano, pero no Ciclos Eulerianos**

La siguiente gráfica tiene un Ciclo Hamiltoniano pues el camino CH pasa por cada vértice una y solo una vez:

$$CH = abcda$$

Sin embargo no tiene ciclos eulerianos ya que cada uno de sus vértices tiene grado impar.

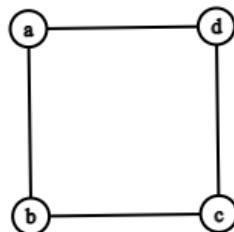


- **Contiene al menos un Ciclo Hamiltoniano y un Ciclo Euleriano**

La siguiente gráfica tiene un ciclo hamiltoniano y un ciclo eureliano :

$$CH = abcda$$

$$CE = ab, bc, cd, da$$



- **No contiene Ciclos Hamiltonianos ni Ciclos Eulerianos**

La siguiente gráfica no tiene ciclos ni hamiltonianos ni eurelianos pues al ser solo una arista no tiene ciclos de ningún tipo.



## Ejercicio 4

Investiga en qué consiste el problema de árboles de Steiner. ¿Es un problema de decisión, búsqueda u optimización?

Enuncia una versión del problema de decisión de árboles de Steiner y una versión del problema de optimización de árboles de Steiner. Para cada caso, menciona cuáles son los parámetros, y da algunos ejemplares concretos y sus soluciones.

**¿En qué consiste el problema de árboles de Steiner.?**

**PROBLEMA DE STEINER:** Dado un grafo  $G = (V, E)$ , un subconjunto de vértices  $S \subseteq V$ , una función de pesos  $f : E \rightarrow Z$ , minimizar el costo del árbol que conecta todos los elementos de  $S$ . Es decir, encontrar el árbol cuya suma de los pesos de sus aristas sea el mínimo. Así, el problema de árboles de Steiner nos pide encontrar un árbol cuya suma de los pesos de sus aristas sea mínimo.

**¿Es un problema de decisión, búsqueda u optimización?**

Es un problema tanto de búsqueda, decisión y optimización pues aunque está enunciado como un problema de optimización siempre podemos llevarlo a la forma de un problema de búsqueda o de decisión pues:

1. Búsqueda: Es un problema de búsqueda pues se trata de buscar el árbol de Steiner dada una gráfica.
2. Optimización: Por definición es un problema de optimización ya que se trata de buscar el árbol con costo mínimo.
3. Decisión: Es un problema de decisión pues también nos podemos preguntar si dada una gráfica, existe o no un árbol de Steiner.

**Enuncia una versión del problema de decisión de árboles de Steiner:**

**PROBLEMA DE STEINER (DECISIÓN):** Dado un grafo  $G = (V, E)$ , un subconjunto de vértices  $S \subseteq V$ , una función de pesos  $f : E \rightarrow Z$ , Determinar si  $G$  tiene un árbol de Steiner, es decir ¿Existe un árbol con costo mínimo?

Ejemplar:

Sea  $G_1 = (V, E)$ ,

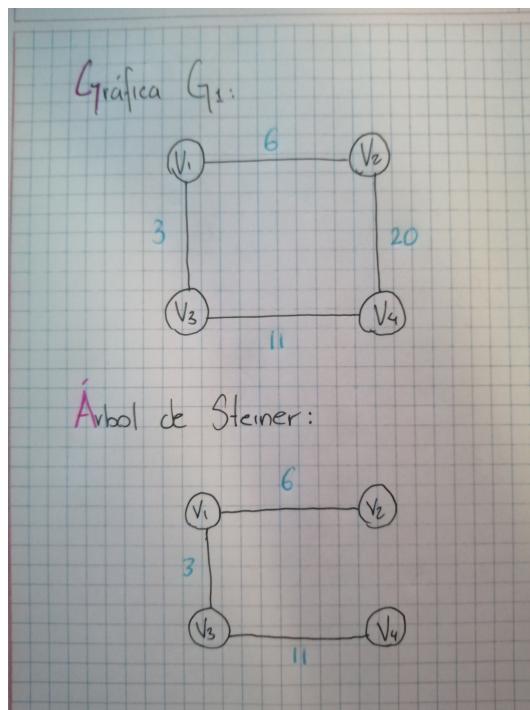
$$V = \{v_1, v_2, v_3, v_4\},$$

$$E = \{v_1v_2, v_1v_3, v_3v_4, v_2v_4\}$$

$$S \subseteq V \text{ donde } S = \{v_1, v_2, v_3, v_4\}$$

$$\text{y pesos } p(v_1v_2) = 6, \quad p(v_2v_4) = 20, \quad p(v_1v_3) = 3, \quad p(v_3v_4) = 11$$

¿Existe un árbol con costo mínimo? (Ver imagen siguiente)



**Enuncia una versión del problema de optimización de árboles de Steiner:**

**PROBLEMA DE STEINER (OPTIMIZACIÓN):** Dado un grafo  $G = (V, E)$ , un subconjunto de vértices  $S \subseteq V$ , una función de pesos  $f : E \rightarrow Z$ , minimizar el costo del árbol que conecta todos los elementos de  $S$ . Es decir, encontrar el árbol cuya suma de los pesos de sus aristas sea mínimo.

Ejemplar:

Sea  $G_1 = (V, E)$ ,

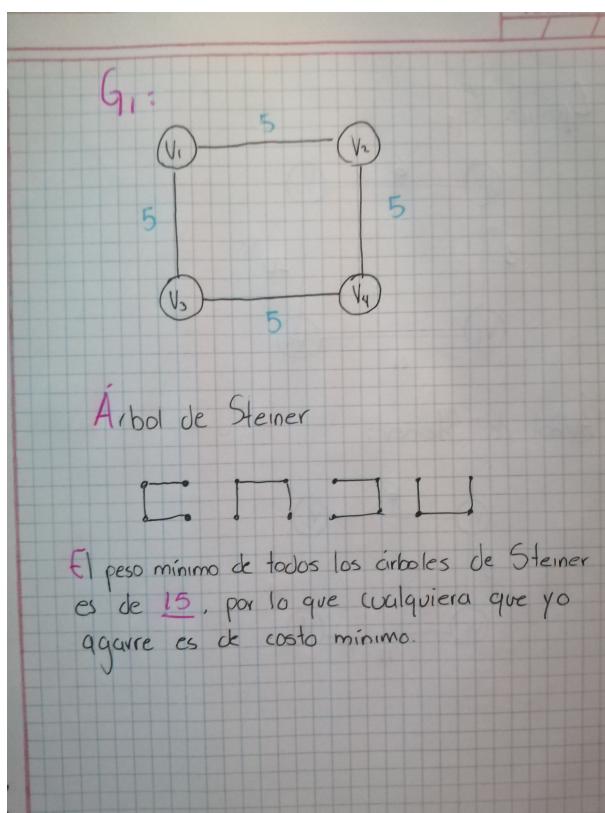
$V = \{v_1, v_2, v_3, v_4\}$ ,

$E = \{v_1v_2, v_1v_3, v_3v_4, v_2v_4\}$

$S \subseteq V$  donde  $S = \{v_1, v_2, v_3, v_4\}$

y pesos  $p(v_1v_2) = 5$ ,  $p(v_2v_4) = 5$ ,  $p(v_1v_3) = 5$ ,  $p(v_3v_4) = 5$

Pregunta: Encontrar un árbol de costo mínimo.



## Ejercicio 5

Estudiamos problemas de optimización en términos de sus versiones en lenguajes. Considera el problema del CLAN, en su versión de optimización, y muestra que existe un algoritmo de tiempo polinomial para el problema original (de búsqueda) sí y solo sí existe un algoritmo de tiempo polinomial para la versión del problema de decisión.

La idea detrás de la demostración es ir agregando vértices a un subconjunto  $H$  (que contenga un CLAN) de la grafica  $G$  hasta que eventualmente ya no sea una CLAN o en su defecto la gráfica  $G$  sea el CLAN.

### Decisión a búsqueda

Supongamos entonces que tenemos un algoritmo *caja negra*  $N$ , la cual crece exponencialmente de acuerdo al numero de vértices, que nos dice si la grafica  $H$  tiene un CLAN, así entonces por cada vertice agregado a  $H$  usaremos de nuevo a la caja negra  $N$ .

Sea una grafica  $H$  tal que  $N$  responde si  $H$  contiene un clan  $c = \{v_1, v_2, \dots, v_n\}$ , notemos que  $n \leq |V|$  Tenemos que toda grafica tiene un clan trivial, ya sea un vértice mismo o dos vértices que estén unidos por una arista, un clan de tamaño 2. Tomemos  $H' = H$

Ahora un vértice cualquiera de  $H'$  y preguntamos a  $N$  si el vértice  $v_i$  pertenece al clan  $C$  de  $H'$ . Como toda gráfica  $H'$  contiene un clan trivial la respuesta que nos dará será un *sí*

Como el vértice  $v_i$  está en  $C$  entonces agregamos otro vértice a  $C$  y preguntamos a la *caja negra*  $N$  si  $H' + v_{i+1}$  sigue teniendo un Clan.

Los posibles resultados de  $N$  son:

- Sí

$H' + v_{i+1}$  sigue teniendo un clan, pues las aristas que inciden en el vértice agregado también incidirán en todos los demás vértices, así que podemos agregarlo sin ningún problema

- No

$H' + v_{i+1}$  no tiene clan, entonces todo clan en  $H'$  no puede contener a este vértice. Continuamos probando para todos los demás vértices en  $H'$

Por lo cual podemos afirmar que la grafica  $H'$  con la que terminamos tiene a un clan por la manera en la que la construimos.

Como nuestro algoritmo  $N$  nos dice en tiempo polinomial si tiene un clan, podemos decir que el tiempo que tomará está acotado por un polinomio  $P(n)$ .

### Búsqueda a desición

Para demostrar el regreso es trivial, ya que el mismo concepto de buscar algo nos induce a responder si algo está o no. Supongamos una gráfica  $H$  y la existencia de una caja negra que nos puede decir que  $H$  tiene un clan maximo.

Preguntamos entonces si el clan es el maximo, las posibles respuestas son *sí* y *no*, por lo cual lo convierte en un problema de desición

## Referencias

Notación asintótica <https://cs.famaf.unc.edu.ar/~hoffmann/md18/04.html>

Notas de ayudante, (Definición de la  $\Omega$ ,  $\Theta$ ,  $O$  )

Árbol de Steiner [https://es.wikipedia.org/wiki/%C3%81rbol\\_de\\_Steiner](https://es.wikipedia.org/wiki/%C3%81rbol_de_Steiner)

Árbol de Steiner [https://hmnn.wiki/es/Steiner\\_tree\\_problem](https://hmnn.wiki/es/Steiner_tree_problem)

Problema del Clan <https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/dna-computing/clique.htm>

Problema del clan [https://en.wikipedia.org/wiki/Clique\\_problem](https://en.wikipedia.org/wiki/Clique_problem)