

## **Tarea 5**

Análisis y Diseño de Algoritmos

Estudiante:

Juan Diego Valencia Alomia

Código:

8977467

Docentes:

Camilo Rocha & Carlos Ramírez

Santiago de Cali, mayo 19, 2025

## 1.) Resposta (a)

Se sabe que

$$c \in \mathbb{R} \wedge c > 0, f \in O(g)$$

Teorema:

$$f(n) \log_2(f(n)^c) \in O(g(n) \log_2(g(n)^c))$$

Demostración

Se procede de manera directa partiendo de la definición de O.

De acuerdo con esto, es necesario encontrar ctes  $c_1$  y  $n_0$  tal que

$$\forall n \geq n_0 \rightarrow f(n) \log_2(f(n)^c) \leq c_1 g(n) \log_2(g(n)^c)$$

$$\cancel{f(n) \log_2(f(n)^c)} \leq \cancel{c_1} g(n) \log_2(g(n)^c)$$

$$f(n) \log_2(f(n)^c) \leq c_1 g(n) \log_2(g(n)^c)$$

$$c_1 \geq \frac{f(n) \log_2(f(n)^c)}{g(n) \log_2(g(n)^c)}$$

Como  $f \in O(g)$  entonces por definición de O:  $f(n) \leq c_2 g(n)$ , es decir que

la función  $g$  crece más o igual que  $f$ , por lo que se parte por casos

Caso 1:  $f = g$

$$c_1 \geq \frac{f(n) \log_2(f(n)^c)}{g(n) \log_2(g(n)^c)}$$

$$c_1 \geq \frac{f(n) \log_2(f(n)^c)}{f(n) \log_2(f(n)^c)}$$

$$c_1 \geq 1$$

Caso 2:  $f < g$

$$c_1 \geq \frac{f(n) \log_2(f(n)^c)}{g(n) \log_2(g(n)^c)}$$

Note que como  $g$  crece más rápido va a llevar la expresión a cero para valores grandes de  $n$ , por lo tanto

$$c_1 \geq 0$$

Como consecuencia hayamos un  $c_1$  para el cual se cumple la condición, es decir que la expresión se cumple para cualesquiera  $f$ ,  $g$  y  $c$  como se indica en el enunciado, tomando como no el que cumple  $f \in O(g)$  y  $c_1$  acorde al caso

2.)

Respuesta b

Justificación

La clase P consiste en todos los problemas que pueden ser resueltos en tiempo polinomial

Con respecto a su entrada

- a) Incorrecto, ya que los problemas en P si son resolubles
- c) Incorrecto, ya que no requieren tiempo exponencial sino, polinomial
- d) Aunque si se cumple que  $P \subseteq NP$ , esto no describe la clase P

Por tanto, la opción que mejor describe la clase P es la b ya que resulta que son problemas que son eficientemente solubles

3.) Respuesta c

Justificación

La clase NP se define como un conjunto de problemas para los cuales una solución o certificado propuesto puede ser verificado en tiempo polinomial

Por tanto, son incorrectos

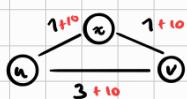
- a) No necesariamente pueden ser verificados en tiempo logarítmico
- b) Describe la clase P, no NP
- d) Incorrecta, ya que no todos los problemas en NP requieren tiempo exponencial para decidirse como los que pertenecen a P

Como consecuencia la respuesta correcta es la c) ya que expresa su principal definición

4.)

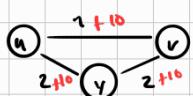
Respuestas b, c y d

Ejemplo A



Antes  $\pi: u \rightarrow x \rightarrow v$  (2)  $> \pi \neq \pi'$  Antes  $\pi: u \rightarrow v$  (1)  $> \pi = \pi'$   
Después  $\pi': u \rightarrow v$  (13)

Ejemplo B



Antes  $\pi: u \rightarrow v$  (1)  $> \pi = \pi'$  Despues  $\pi': u \rightarrow v$  (11)

- b) Es cierto (Ejemplo A)
- c) Es cierto (Ejemplo A y B)
- d) Es cierto (Ejemplo B)
- a) Erroneo (contra ejemplo A)

5.)

### Especificación del problema

Entrada: un arreglo  $A[0..N)$   $N \geq 0$  de enteros

Salida: un arreglo  $A'$  representando un min-heap

### Planteamiento de las funciones

Sea  $N = |A|$ ,  $x \in \mathbb{Z}$

$\text{init}(N)$ : crea una cola de prioridad

$\text{add}(x)$ : agrega  $x$  a la cola

$\text{min}()$ : retorna el valor mínimo de la cola

$\text{empty}()$ : ¿Está vacía la cola?

Sean  $H$  un arreglo,  $s \in \mathbb{Z} \wedge mx \in \mathbb{Z}$

**def init(N):**

```
global H, s, mx
H = []
s = 0
mx = N
```

**def add(x):**

```
global H, s, mx
```

if  $s \geq mx$ :

Print("maximas capacidad alcanzada")

else:

H.append(x)

s += 1

i = s - 1

p = (i - 1) // 2 # Padre

while  $i > 0 \wedge H[i] < H[p]$ :

$H[i], H[p] = H[p], H[i]$

i = p

p = (i - 1) // 2

**def empty():**

return s == 0

**def min():**

```
global H, s
if empty()
    Print("cola vacía")
else:
    ans = H[0]
    H[0] = H[s - 1]
    H.pop()
    s -= 1
```

i = 0

flag = True

while flag:

l = 2 \* i + 1

r = 2 \* i + 2

mn = i

if  $l < s \wedge H[l] < H[mn]$ :

mn = l

if  $r < s \wedge H[r] < H[mn]$ :

mn = r

if mn != i:

$H[i], H[mn] = H[mn], H[i]$

i = mn

else:

flag = False

return ans

### Ejemplo de uso

**def pq(A):**

```
N = len(A)
init(N)
```

for x in A:

add(x)

for i in range(N):

y = min()

Print(y)

$A = [5, 1, 9, -2, 7]$

$pq(A)$

out put:

-2, 1, 5, 7, 9

## 6.) Respuesta b

## Justificación

La b expone tanto la definición de P como la de NP, y los demás son errores ya que:

- a) En NP importa que se verifique en tiempo polinomial, no que se acepte
- c) En NP importa que se verifique en tiempo polinomial, no que no se acepte
- d) Los problemas en P si pueden ser decididos en tiempo polinomial

## 7.)

$TSP = \{ \langle G = (V, E, w) : E \rightarrow \mathbb{R}, K \rangle \mid "G \text{ es completo y tiene un circuito cuyo costo no supera } K" \}$

Teorema: TSP es NPC si Hamilton es NP-Hard

## Demostración

Se quiere ver que:

- $TSP \in NP \wedge TSP \in NP\text{-Hard}$

Lema I:  $TSP \in NP$

## Demostración

Se requiere construir un algoritmo de verificación en tiempo polinomial para TSP

Sea  $\langle G, w, K \rangle$  un certificado y A una secuencia de vértices

## Condiciones

1.  $|A| = |V| + 1$
2.  $A[0] = A[|A|] \wedge \{c[1], c[2], \dots, c[V] = V$
3.  $\sum_{i=0}^{|V|} w(c_i, c_{i+1}) \leq K$

## Complejidad

$O(|V|)$

$O(|V|)$

$O(|V|)$

Por lo tanto TSP puede verificarse en  $O(|V|)$  por lo que pertenece

a NP

Lema 2: TSP  $\in$  NP-Hard

Demonstración: Se mostrará que es posible reducir Hamilton a TSP en tiempo Polinomial

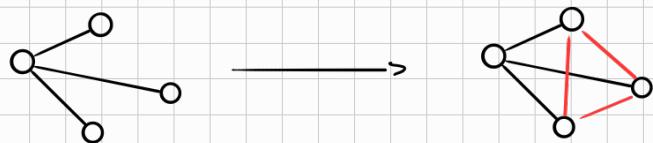
$$\text{Hamilton} \leq_p \text{TSP}$$

Sea  $f$  una función, que permite construir un grafo  $f(G) = G'$  a partir de un grafo  $G = (V, E)$ . Donde  $G'$  es un grafo completo  $G' = (V, E')$

tal que  $E' = \{(u, v) \mid u, v \in V \wedge u \neq v\}$  y con una función de peso  $w$  de la siguiente forma

$$w(u, v) = \begin{cases} 0 & \text{if } (u, v) \in E \\ 1 & \text{otherwise} \end{cases}$$

Ejemplo: los aristas negras tienen peso 0 y los rojas 1



Note que  $f$  se puede calcular en tiempo polinomial, ahora falta demostrar que  $f$  es una reducción

$$G(V, E) \xrightarrow{f} G'(V, E'), 0$$

Se procede por casos: sea  $x$  una entrada de Hamilton

1.  $x \in \text{Hamiltonian}$ :

Si  $x \in \text{Hamiltonian}$ , entonces  $\langle f(x), 0 \rangle \in \text{TSP}$ , luego  $x = G = (V, E)$  tiene un circuito Hamiltoniano de la siguiente manera

$$\vec{u}: u_0 \rightarrow u_1 \rightarrow u_2 \rightarrow \dots \rightarrow u_{|V|-1} \rightarrow u_0$$

Note que  $\vec{u}$  continua siendo un circuito en  $G'$ , además es un circuito de costo 0. Se concluye que  $\langle f(x), 0 \rangle \in \text{TSP}$

## 2. $x \notin \text{Hamilton}$

Si  $x \notin \text{Hamilton}$ , entonces  $\langle f(x), o \rangle \notin \text{TSP}$ , por equivalencia

Si  $\langle f(x), o \rangle \in \text{TSP} \rightarrow x \in \text{Hamilton}$

Supongamos que  $\langle f(x), o \rangle \in \text{TSP}$ , Por suposición hay un circuito

$$\vec{u} : u_0 \xrightarrow{o} u_1 \xrightarrow{o} u_2 \xrightarrow{o} \dots \xrightarrow{o} u_{|u|-1} \xrightarrow{o} u_0$$

Donde  $\vec{u}$  tiene costo 0, además note que  $\vec{u}$  es un circuito de Hamilton

Por tanto  $x \in \text{Hamilton}$

En conclusión,  $f$  es una reducción polinomial de Hamilton a TSP y  $\text{TSP} \in \text{NPC}$

8.)

Teorema:

Suponga que  $P \neq NP$ . Entonces, si un lenguaje  $L \in P$  entonces  $L \notin NP\text{-Hard}$  bajo reducciones polinómicas

Demonstración

Se parte por contradicción

$L \in P \wedge L \in NP\text{-Hard}$

Por definición de  $NP\text{-Hard}$   $\forall L \in NP\text{-Hard}$  existe una reducción polinómica  $f$  de  $L$  que

$$x \in L \xrightarrow{f} f(x) \in L$$

La función  $f$  se puede calcular en tiempo polinomial  $O(|x|^k)$  para algún  $k \in \mathbb{Z}$

Como  $L \in P$ , existe algún algoritmo  $A$  que decide si una entrada  $y \in L$  en tiempo polinomial, supongamos  $O(|y|^i)$  para algún  $i$

Algoritmo para  $L' \in NP$

Dado una entrada  $x$  para  $L'$

1. calcular  $f(x)$  ( $O(|x|^k)$ )

2. llamar  $A(f(x))$  ( $O(|f(x)|^i)$ )

3. Aceptar si  $A(f(x))$  acepta

$L'$  se decide en tiempo polinomial

Conclusión:

Se encontró un algoritmo polinomial para decidir un lenguaje  $L \in NP$ , lo que implica que  $NP \subseteq P$ , y como sabemos que  $P \subseteq NP$  entonces

$$P = NP$$

No obstante esto contradice nuestra suposición inicial.

Como consecuencia, no puede existir un lenguaje  $L \in P$  tal que que sea NP-Hard, a menos de que  $P = NP$ , dado a que asumimos  $P \neq NP$  entonces  $L \in P \rightarrow L \notin NP\text{-Hard}$