# Guia 2 Complejidad Computacional 1 Cuatrimestre 2025

**Juan DElia**

## Ejercicio 1 Probar que los siguientes lenguajes están en P

### a) COPRIME = {⟨a, b⟩ : (a : b) = 1, es decir, a y b son coprimos}

```
def coprimo(a,b){
    tmp = 0
    while b != 0:
        tmp = b
        b = a % b
        a = tmp
    //hasta aca es el algoritmo de eculides para calcular mcd
    return mcd == 1
}
```

Es polinomial respecto de |a| + |b|. Con |a| = $log_2(a)$ y |b| = $log_2(b)$. ⇒ a = $2^{|a|}$

Con a > b la complejidad es:

O(log a) reemplazando con la def |a| → $O(log(2^{|a|})) = O(|a|.log(2)) = O(|a|)$

Notar que haciendo el algoritmo naive que es O(a): $O(a) = O(2^{|a|})$

Que es exponencial al tamaño de la entrada

La moraleja de este ejercicio es que si la entrada es un numero, **iterar todo el rango de ese numero es exponencial respecto a su tamaño**.

Es la idea que hay que usar en todos los incisos.

### c) TREE = {⟨G⟩ : G es un grafo conexo sin ciclos}

Algoritmo:

```
1. DFS en G marfcando los nodos, si encuentro uno ya marcado devuelvo falso.
2. Si termina el dfs, recorrer la lista de nodos y ver que esten
todos marcados para devolver true, caso contrario falso.
```

Complejidad: Si cada nodo se representa en logn y cada arista logm Digamos |G| = |V|+|E| = nlogn+mlogm

1 → $O((nlogn)^2)$ en matriz de adyacencia 2 → $O(nlogn)$ recorrer la lista de nodos

Es polinomial

## Ejercicio 2: Probar que la clase P está cerrada por unión, intersección y complemento

Tomo cualqueira $L_1, L_2 \in P$ y P1(x) P2(x) las maquinas que los reconocen en tiempo polinomial

Unión:

```
def unionL1_L2(x):
    return p1(l1) or p2(x)
```

IntersecciónL1_L2:

```
def interseccion(x):
    return p1(x) and p2(x)
```

Complemento:

```
def complemento_L1(x):
    return not p1(x)
```

Son los tres polinomiales porque en python son polinomiales

## Ejercicio 3 Probar que los siguientes lenguajes están en NP.

**HAMPATH = {⟨G, s, t⟩ : G es un grafo con dos nodos s y t tales que hay un camino hamiltoniando de s a t}**

Certificado: Lista de los nodos que forman el camino hamiltoneano (camino que pasa por todos los nodos si repetir)

Es polinomial tiene tamaño O(|V|) o O(nlogn).

```
def verificador(g,s,t,c):
    for cada nodo en el certificado-1:
        if(g[c[i]][c[i+1]]) == 0:
            return false // si no son adyacentes los nodos que tienen que serlo
                         //  para armar el caminio da falso
    return length(c) == length(g.V) and sinRepetidos(c) and c[0] == s and c[-1] == t
```

Esto corre en O(|c|) que es polinomial

**falta agregar. . .**

## Ejercicio 4 Probar que los siguientes problemas están en coNP

### a) PRIME= {n : n ∈ N es primo}

```
PRIME esta en coNP si solo si PRIME complemento esta en NP.
El complemento es el lenguaje de los numeros naturales que no son primos.
Veo que este en NP:
Certificado: lista de su factorizacion en primos.
El certificado es polinomial, a lo sumo tiene O(log(n)) factores primos.
Y se verifica en tiempo polinomial con el siguiente pseudocodigo:
def verificar(n, cert):
    res = 1
    for e in certificado:
        // ver que un nro sea primo es polinommial
        if(!esPrimo(e)){return false}
        res *= e
    return n == res and 1 not in cert and n not in cert

Corre en tiempo O(|cert|.polinomial) que es polinomial
```

### b) GIRTH= {⟨G, k⟩ : G es un grafo tal que todos sus ciclos simples tienen k o menos vértices}

```
El complemento de un para todo es un existe negando la propiedad. Entonces el complemento de
{<G,k> tq existe un ciclo simple con mas de k vertices}
Veo que sea NP:
Certificado: Camino de vertices de longitud > k que fomrman un ciclo. Tiene longitud O(nlogn
Verificador:
Hay que recorrer la lista de nodos y verificar que forman un ciclo (ver que sean adyacentes
Por ultimo hay que chequear que la longitud del certificado sea mayor a k y que no haya nodo
```

### c) TAUTOLOGY= {⟨$\phi$⟩ : $\phi$ es tautología}

El complemento es que exista una valuacion que haga la formula falsa. Veo que sea NP: Certificado: Una valuacion que al evaluarla en $\phi$ de falso. Su longitud es polinomial respecto a $\phi$, pues si $\phi$ tiene n varibles distintas el certificado tendra longitud n. Verificador: Recorre el certificado (es O(|certificado|)), reemplaza cada una de las variables en $\phi$, evalua $\phi$ y verifica que sea falsa (polinomial respecto de $\phi$)

## Ejercicio 5

$\Rightarrow$)

Por hipótesis hay camino de longitud par <= k de s a t en g, en general:

$s \rightarrow v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_n \rightarrow t$

Por la definicion de $G'$ existe en $G'$ un camino análogo

$(s, p) \rightarrow (v_1, i) \rightarrow (v_2, p) \rightarrow ... \rightarrow (t, p)$

Notar que por la alternancia en la segunda componente siempre que un camino termina en un nodo de tipo (n,p) el camino tiene longitud par y si termina en uno (n,i) tiene longitud impar.

Asi que si el camino de G tiene longitud par, el camino en G' al ser análogo tambien y por consecuente su último nodo tiene que tener p en la segunda componente. (Y ambos son de misma longitud asi que se cumple que es de longitud <= k por hipótesis)

$< G', (s, p), (t, p), k > \in PATH$

$\Leftarrow$)

Por hipótesis en $G'$ hay camino de longitud <= k de (s,p) a (t,p)

$(s, p) \rightarrow (v_1, i) \rightarrow (v_2, p) \rightarrow ... \rightarrow (t, p)$

EL camino tiene longitud par porque termina en un nodo (n,p) por la misma observacion de antes.

Entonces existe un camino análogo tambien de longitud <= k en G de forma:

$s \rightarrow v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_n \rightarrow t$

Que al ser análogo tambien tiene longitud par.

$< G, s, t, k > \in EVEN - PATH$

Queda demostrado

## Ejercicio 6

### a

Dada una instancia ⟨X⟩ de 2-PARTITION, se define la instancia ⟨R, r1, . . . , r|X|⟩ de RECTANGLE PACKING donde R tiene base P x ∈ X x/2 y altura 2, y ri es un rectángulo de base xi y altura 1, para cada 1 ≤ i ≤ |X|. Demostrar que ⟨X⟩ ∈ 2-PARTITION $\iff$ ⟨R, r1, . . . , r|X|⟩ ∈ RECTANGLE PACKING.

Veo la ida →:

Quiero ver que los subrectangulos $r_i$ cubren al rectangulo R.

Para que la instancia ∈ RECTANGLE PACKING, los $r_i$ deben poder cubrir R. como cada $r_i$ tiene area $x_i$ el conjunto de las areas es $X$. Por hipotesis se que lo puedo dividir como:

$$\sum_{x \in X_1} x = \sum_{x \in X_2} x = \sum_{x \in X} x/2$$

Asi que puedo dividir a todos los $r_i$ en dos rectangulos de altura 1 (los pongo todos uno al lado de otro). Y la suma de ambas es el area de R. Si pongo esos de rectangulos uno arriba de otro obtengo al R de altura 2 que cumple con area. (Notar que la base de cada area es $\sum_{x \in X} x/2$)

Veo la vuelta ←: Ahora por hipotesis se que tengo un rectangulo R y que los $r_i$ con altura 1 y base $x_i$ cubren completamente a R.

$\sum_{x \in X} x =$ Area del Rectangulo

Como la hipotesis es que los $r_i$ pueden cubrir al rectangulo (que tiene altura 2) y se que todos los rectangulos tienen altura 1, pensando geometricamente, puedo partir al rectangulo R en dos rectangulos de altura 1. Llamo al conjunto de las areas de cada particion $X_1$ y $X_2$. Como lo parti a la mitad:

$\sum_{x \in X_1} x = \sum_{x \in X_2} = \sum_{x \in X} x/2$

Como parti a la mitad el triangulo cada subconjunto tiene elementos distintos $X_1 \cap X_2 = \emptyset$

Por lo mismo $X_1 \cup X_2 = X$

### b

te la debo

### c

Mostrar que las reducciones implicadas por los puntos anteriores son polinomiales en función de los tamaños de las entradas.

La reduccion seria algo como:

$< x > \in 2 - partition \iff f(< x >) \in RECTANGLEPACKING$

Tengo que encontrar una f. Notar que los rectangulos tienen la pinta de (base,altura)

f(X): res = new tupla R = (sum(x)/2,2) res.agregar(R) for e in x: res.agregar(e,1) return res

## Ejercicio 7

Explicar por qué la identidad no es una reducción polinomial de un lenguaje $\Pi$ a $\Pi^c$.

Concluir que las nociones de NP y coNP son altamente sensibles a la "etiqueta" de la respuesta.

La identidad es f(x): return x, para que se cumpla la reduccion tendria que valer que:

$$x \in \Pi \iff x \in \Pi^c$$

Abs!

(No entendi la moraleja del ejercicio consultar)

## Ejercicio 8

Considerar el siguiente lenguaje: CONNECTED = {⟨G, s, t⟩ : G es un digrafo y s y t dos nodos de G tales que hay un recorrido de s a t} Para un digrafo G, sea H el digrafo que tiene un vértice (S, v) para cada S ⊆ V (G) y cada v ∈ V (G), donde (S, v) → (R, w) es una arista de H si y solo si w ∉ S, R = S ∪ {w} y v → w es una arista de G.

### a)

Demostrar que ⟨G, s, t⟩ ∈ HAMPATH ⟺ ⟨H,({s}, s),(V (G), t)⟩ ∈ CONNECTED.

⇒) Si hay Hamiltoneano en G de s a t hay un camino de forma:

$$s \to v_1 \to v_2 \to ... \to v_n \to t$$

tal que:

$V(g) = \{s, v_1, ...v_n, t\}$

Por la definicion de H, Hay camino analogo en H tq:

$$(\{s\}, s) \to (\{s, v_1\}, v_1) \to (\{s, v_1, v_2\}, v_2) \to (\{s, v_1, v_2, ..., v_n\}, v_n) \to (\{s, v_1, v_2, ..., v_n, t\}, t) = (V(g), v_n)$$

$$\Rightarrow < H, (s, s), (V(g), t) > \in CONNECTED$$

⇐)

Por como esta definido $H$ cada vez que se “camina” de un nodo a otro, el destino se agrega a la primera componente del vertice de H donde se forma un conjunto de los nodos ya visitados en el “camino actual” (quedan de la pinta (camino, nodo_actual)). Notar que en su definicion no permite tener nodos repetidos en el conjunto.

Entonces como por hipotesis hay camino de $(\{s\}, s)$ a $(V(g), t)$ necesariamente tuvo que haber recorrido todos los nodos del grafo de forma:

$$(\{s\}, s) \to (\{s, v_1\}, v_1) \to (\{s, v_1, v_2\}, v_2) \to (\{s, v_1, v_2, ..., v_n\}, v_n) \to (\{s, v_1, v_2, ..., v_n, t\}, t)$$

Por definicion de H a partir de G, el camino analogo existe en G:

$$s \to v_1 \to v_2 \to ... \to v_n \to t$$

$$\Rightarrow < G, s, t > \in HAMPATH$$

**queda demostrado**

**b)**

Mostrar que la reducción de HAMPATH a CONNECTED implicada por el punto anterior **no** es polinomial.

Se ve que no es polinomial porque las aristas del grafo H se forman a partir de todos los caminos posibles en cada arista. Mas precisamente por la definicion de $H$:

$|V(H)| = |\{(S, v) : S \subseteq V(G)\}| = n.2^n$

Es exponencial respecto al tamaño de la entrada.

## Ejercicio 9 (a consultar)

Si x pertenece a L quiero que f(x) termine asi pertenece a halting, sino quiero que se cuelgue

Llamo L a la maquina que reconoce $L$

```
def f(x):
    if(L(x) == 1):
        return(<M,x>)
    else:
        return <M',x>

def M(x):
    return 1

def M'(x):
    while(true)
```

## Ejercicio 10 Probar que NP $\subseteq$ RECURSIVE. Concluir que HALTING $\notin$ NP

**la demo esta basada en el apunte de santi, chequear la parte de maquinas no deterministicas**

Si un problema esta en NP, por definicion, hay una maquina no deterministica N que corre en tiempo polinomial tq: L(n) = L (es decir, x pertenece a L sii existe un computo aceptador de N a partir de x).

Digamos que cada computo posible de la maquina N, corre a lo sumo en tiempo T(n). Podemos representar todos los posibles computos de N como un grafo, donde cada nodo es una configuracion. (en estas maquinas desde cada configuaricion hay dos transiciones posibles $\delta_1$ y $\delta_2$). Cada nodo tiene dos hijos que corresponden a una posible evolucion del siguiente paso de esa configuracion.

Notar que con esta idea de codificacion, cualquier computo posible de la maquina N se puede codificar como cadenas de 1 y 0. Siendo 0 avanzar a la izquierda y 1 a la derecha.

Entonces todos los posibles computos se pueden simular de manera **deterministica** a partir de un nodo inicial. Habria que analizar $2^t$ computos (porque son cadenas de 01* de longitud t), siendo t la longitud maxima de un computo.

Como cada computo toma a lo sumo t tiempo (t pasos a realizar), podemos simular todos los computos en $O(t.2^t)$

Esto significa que todos los computos de una maquina no deterministica se pueden simular (aunque sea en tiempo exponencial), y como ya explique todo lenguaje en NP tiene una maquina no deterministica que lo decide.

En conclusion todo problema de NP, es decidible $\Rightarrow NP \subseteq RECURSIVE$

(Como halting no es decidible no pertenece a NP)