

Guia 2 Complejidad

Ejercicio 1 Probar que los siguientes lenguajes están en P

a) $\text{COPRIME} = \{\langle a, b \rangle : (a, b) = 1, \text{ es decir, } a \text{ y } b \text{ son coprimos}\}$

```
def coprime(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 $|b| = \log_2(b)$. $\Rightarrow a = 2^{|a|}$

Con $a > b$ la complejidad es:

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

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) $\text{TREE} = \{\langle G \rangle : G \text{ es un grafo conexo sin ciclos}\}$

Algoritmo:

1. DFS en G marcando 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 $\log n$ y cada arista $\log n$ Digamos $|G| = |V| + |E| = n \log n + m \log n$

$1 \rightarrow O((n \log n)^2)$ en matriz de adyacencia $2 \rightarrow O(n \log n)$ recorrer la lista de nodos

Es polinomial

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

Tomo cualquiera $L_1, L_2 \in P$ 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 = $\{ \langle G, s, t \rangle : G \text{ es un grafo con dos nodos } s \text{ y } t \text{ tales que hay un camino hamiltoniano de } s \text{ a } t \}$

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

Es polinomial tiene tamaño $O(|V|)$ o $O(n \log n)$.

```
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 camino 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** = $\{n : n \in \mathbb{N} \text{ es primo}\}$

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 polinomial
        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** = $\{\langle G, k \rangle : G \text{ es un grafo tal que todos sus ciclos simples tienen } k \text{ o menos vértices}\}$

El complemento de un para todo es un existe negando la propiedad. Entonces el complemento de $\{\langle G, k \rangle : \text{tq existe un ciclo simple con mas de } k \text{ vertices}\}$

Veo que sea NP:

Certificado: Camino de vertices de longitud $> k$ que fomrman un ciclo. Tiene longitud $O(n \log n)$

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 nodos repetidos

c) **TAUTOLOGY** = $\{\langle \phi \rangle : \phi \text{ es tautología}\}$

El complemento es que exista una valuacion que haga la formula falsa. Veo que

sea NP: Certificado: Una valuacion que al evaluarla en ϕ de falso. Su longitud es

polinomial respecto a ϕ , pues si ϕ tiene n variables distintas el certificado tendra

longitud n . Verificador: Recorre el certificado (es $O(|certificado|)$), reemplaza

cada una de las variables en ϕ , evalua ϕ y verifica que sea falsa (polinomial

respecto de ϕ)

Ejercicio 5

\Rightarrow)

Por hipótesis hay camino de longitud par $\leq k$ de s a t en G , en general:

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

Por la definición de G' existe en G' un camino análogo

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

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.

Así que si el camino de G tiene longitud par, el camino en G' al ser análogo también y por consiguiente su último nodo tiene que tener p en la segunda componente. (Y ambos son de misma longitud así que se cumple que es de longitud $\leq k$ por hipótesis)

$$\langle G', (s, p), (t, p), k \rangle \in PATH$$

\Leftarrow)

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

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

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

Entonces existe un camino análogo también de longitud $\leq k$ en G de forma:

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

Que al ser análogo también tiene longitud par.

$$\langle G, s, t, k \rangle \in EVEN - PATH$$

Queda demostrado

Ejercicio 6

a

Dada una instancia $\langle X \rangle$ de 2-PARTITION, se define la instancia $\langle R, r_1, \dots, r_{|X|} \rangle$ de RECTANGLE PACKING donde R tiene base $|X|$ y altura 2, y r_i es un rectángulo de base x_i y altura 1, para cada $1 \leq i \leq |X|$. Demostrar que $\langle X \rangle \in 2\text{-PARTITION} \iff \langle R, r_1, \dots, r_{|X|} \rangle \in \text{RECTANGLE PACKING}$.

Veamos la ida \rightarrow :

Quiero ver que los subrectángulos r_i cubren al rectángulo R .

Para que la instancia $\in \text{RECTANGLE PACKING}$, los r_i deben poder cubrir R . Como cada r_i tiene área x_i el conjunto de las áreas es X . Por hipótesis se que lo puedo dividir como:

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

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

Veamos la vuelta \leftarrow : Ahora por hipótesis se que tengo un rectángulo R y que los r_i con altura 1 y base x_i cubren completamente a R .

$$\sum_{x \in X} x = \text{Área del Rectángulo}$$

Como la hipótesis es que los r_i pueden cubrir al rectángulo (que tiene altura 2) y se que todos los rectángulos tienen altura 1, pensando geoméricamente, puedo partir al rectángulo R en dos rectángulos de altura 1. Llamo al conjunto de las áreas de cada partición X_1 y X_2 . Como lo parti a la mitad:

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

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

$$\text{Por lo mismo } X_1 \cup X_2 = X$$

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 reducción sería algo como:

$$\langle x \rangle \in 2-partition \iff f(\langle x \rangle) \in RECTANGLEPACKING$$

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 Π a Π^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: $CONNECTED = \{\langle G, s, t \rangle : G \text{ es un digrafo y } s \text{ y } t \text{ dos nodos de } G \text{ tales que hay un recorrido de } s \text{ a } t\}$ Para un digrafo G , sea H el digrafo que tiene un vértice (S, v) para cada $S \subseteq V(G)$ y cada $v \in V(G)$, donde $(S, v) \rightarrow (R, w)$ es una arista de H si y solo si $w \notin S$, $R = S \cup \{w\}$ y $v \rightarrow w$ es una arista de G .

a)

Demostrar que $\langle G, s, t \rangle \in HAMPATH \iff \langle H, (\{s\}, s), (V(G), t) \rangle \in CONNECTED$.

\Rightarrow) Si hay Hamiltoniano en G de s a t hay un camino de forma:

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

tal que:

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

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

$$(\{s\}, s) \rightarrow (\{s, v_1\}, v_1) \rightarrow (\{s, v_1, v_2\}, v_2) \rightarrow (\{s, v_1, v_2, \dots, v_n\}, v_n) \rightarrow (\{s, v_1, v_2, \dots, v_n, t\}, t) = (V(g), v_n)$$

$$\Rightarrow \langle H, (s, s), (V(g), t) \rangle \in CONNECTED$$

\Leftarrow)

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:

$$(\{s\}, s) \rightarrow (\{s, v_1\}, v_1) \rightarrow (\{s, v_1, v_2\}, v_2) \rightarrow (\{s, v_1, v_2, \dots, v_n\}, v_n) \rightarrow (\{s, v_1, v_2, \dots, v_n, t\}, t)$$

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

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

$$\Rightarrow \langle G, s, t \rangle \in \text{HAMPATH}$$

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 :

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

Es exponencial respecto al tamaño de la entrada.

Ejercicio 9 (a consultar)

Si x pertenece a L quiero que $f(x)$ termine así 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 determinísticas

Si un problema esta en NP, por definicion, hay una maquina no determinística 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 configuracion hay dos transiciones posibles δ_1 y δ_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 **determinística** 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 $O(t \cdot 2^t)$

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

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

(Como halting no es decidable no pertenece a NP)