Guía 3

Ej 1:

1. Determinar cuáles de las siguientes afirmaciones son verdaderas y cuáles falsas. Demostrar aquellas que son verdaderas y dar contraejemplos para aquellas que son falsas.

   a) $\mathsf{P} \subseteq \mathsf{NP}$ y $\mathsf{P} \subseteq \mathsf{coNP}$.
   b) Si $\mathsf{P} = \mathsf{NP}$, entonces $\mathsf{coNP} = \mathsf{NP}$.
   c) Si $\mathsf{P} = \mathsf{NP}$, entonces todos los lenguajes pertenecen a P.
   d) Si $\mathsf{coNP} = \mathsf{NP}$, entonces $\mathsf{SAT} \in \mathsf{coNP}$.
   e) Si $\mathsf{coNP} \subseteq \mathsf{NP}$, entonces $\mathsf{NP} = \mathsf{coNP}$.

a) $P \subseteq NP$ y $P \subseteq coNP$ Verdadero

Para ver esto voy a tomar un L genérico tal que $L \in P$ y demostrar que $L \in NP$ y $L \in coNP$.

$P \subseteq NP$:

Tengo una M det. poly. que decide L.

Luego, tomo como certificado un c random (el cual no voy a usar pero tiene tamaño poly) y voy a tener como verificador a M' det. poly. donde se le pase la entrada y c como certificado

M': ⟨x, c⟩

ret M(x) → Es poly

$P \subseteq coNP$:

M decide L.

Luego, uso la misma idea que antes, pero ahora M' es de la forma:

M': ⟨x, c⟩

ret ¬M(x)

Ya que M' sería la máq. que decide $\bar{L}$ tal que $\bar{L} \in NP$.

b) Si $P = NP$, entonces $coNP = NP$ Verdadero

*Notar que sabemos que $P \subseteq NP$ pero no si $NP \subseteq P$

Si tengo que $P = NP$ y sé que $P = coP$, puedo decir que $NP = coP$, pero si en coP por definición tengo todos los complementos de los lenguajes en P (y a su vez los complementos de los lenguajes en NP) obtengo que $coP = coNP$, y como $P = coP$, $P = NP = coP = coNP$.

γ

c) Si $P = NP$, entonces todos los lenguajes pertenecen a P Falso

Sabemos que $ExpTime \notin P$ por jerarquía temporal. Esto hace imposible que todos los lenguajes pertenezcan a P.

Hay muchas formas de encarar esto igual, podría haber dicho que HALTING$\notin$P, también decir que la cantidad de lenguajes es no numerable y la cantidad de máquinas si lo es, etc...

d) Si coNP = NP, entonces SAT ∈ coNP Verdadero

Si coNP = NP, entonces $\forall L.\ L \in NP$ sii $L \in coNP$, en particular SAT está en NP, por lo cual está en coNP.

e) Si coNP ⊆ NP, entonces NP = coNP

Quiero probar que si $coNP \subseteq NP \Rightarrow NP \subseteq coNP$:

Sea $L \in NP$, $\overline{L} \in coNP$, pero como $coNP \subseteq NP$, $\overline{L} \in NP$, entonces $L \in coNP$ por definición de coNP, se encuentran todos los lenguajes que su complemento está en NP.

Ej 2:

2. ¿Es cierto que si dos lenguajes Π y Γ pertenecen a NPC entonces $\Pi \leq_p \Gamma$, y también $\Gamma \leq_p \Pi$? Justificar.

coNP-completo

Sí, es cierto.

Si π y $\Gamma \in$ NP-C, entonces:

$\pi, \Gamma \in NP$ y $\pi, \Gamma \in$ NP-hard ↳ $\forall L.\ L \in NP$, $L \leq_p \pi$ y $L \leq_p \Gamma$

$\pi \leq_p \Gamma$:

$x \in \pi$ sii $f(x) \in \Gamma$, esto se cumple porque $\pi \in NP$ y Γ es NP-hard

$\Gamma \leq_p \pi$:

$x \in \Gamma$ sii $f(x) \in \pi$, esto se cumple porque $\Gamma \in NP$ y π es NP-hard

Ej 3:

3. Sean Π y Γ dos lenguajes tales que $\Pi \leq_p \Gamma$. ¿Qué se puede inferir?

a) Si $\Pi \in$ P entonces $\Gamma \in$ P.
b) Si $\Gamma \in$ P entonces $\Pi \in$ P.
c) Si $\Gamma \in$ NPC entonces $\Pi \in$ NPC.
d) Si $\Pi \in$ NPC entonces $\Gamma \in$ NPC.
e) Si $\Gamma \in$ NPC y $\Pi \in$ NP entonces $\Pi \in$ NPC.
f) Si $\Pi \in$ NPC y $\Gamma \in$ NP entonces $\Gamma \in$ NPC.
g) Π y Γ no pueden pertenecer ambos a NPC.

a) Si $\pi \in P$, entonces $\Gamma \in P$ FALSO

Todo $L \in$ Recursive cumple que $L \leq_p$ HALTING y HALTING no es computable, por lo cual $\notin P$.

b) Si $\Gamma \in P$, entonces $\pi \in P$ Verdadera

Si $\pi \leq_p \Gamma$, entonces hay una f computable polinomialmente tal que:

$$x \in \pi \text{ sii } f(x) \in \Gamma$$

Luego, tengo que la máquina que decide π puede ser aplicar la máquina que decide f (tarda poly en función de x) y luego aplicar la máquina que decide Γ (es poly en función de $f(x)$, pero esta es poly en función de x, así que no hay problema).

c) Si $\Gamma \in NPC$, entonces $\Pi \in NPC$ Falso

Sé que si $\Gamma \in NPC$, entonces $\Pi \in NP$, pero no sé si $\Pi \in NP\text{-hard}$ porque por ejemplo podría suceder que $\Gamma \in P$ y se sigue cumpliendo lo anterior, pero no se sabe si Γ es NP-hard, porque si lo es entonces $P = NP$ y esto es una pregunta abierta.

d) Si $\Pi \in NPC$, entonces $\Gamma \in NPC$ Falso

Es muy similar a la justificación de a)

Γ podría ser HALTING, lo cual es NP-hard, pero no es NP (porque en NP no hay problemas no computables.

e) Si $\Gamma \in NPC$ y $\Pi \in NP$, entonces $\Pi \in NPC$ Falso

No necesariamente, aplica la misma justificación que en el c)

f) Si $\Pi \in NPC$ y $\Gamma \in NP$, entonces $\Gamma \in NPC$ Verdadero

Cubre el error que sucedía en el d)

Si $\Pi \in NPC$ y $\Gamma \in NP$, entonces sucede que $\Gamma \leq_p \Pi$, además como sé que $\Pi \leq_p \Gamma$ puedo decir que:

$$\forall L.\ L \in NP : L \leq_p \Pi \leq_p \Gamma \text{ si\! si } L \leq_p \Gamma$$

Ya que sería la composición de 2 funciones computables polinomialmente y esto ya habíamos visto que era polinomial en lo teórico.

g) Π y Γ no pueden pertenecer ambas a NPC Falso

Si sucede lo que pasa en f ambas pertenecen a NPC. También fue probado en el Ej 2 esto.

Ej 4:

4. Decir si las siguientes afirmaciones son verdaderas o falsas:

   a) Si P = NP, entonces todo problema NP-completo es polinomial.
   b) Si P = NP, entonces todo problema NP-hard es polinomial.
   c) Si las clases NP-completo y coNP-completo son disjuntas entonces P ≠ NP.
   d) HALTING es NP-hard y coNP-hard.

a) Si P=NP, entonces toda problema NP-C es polinomial Verdadera

Si, $\forall L.\ L \in NP \Leftrightarrow L \in P$. O sea, las $L \in NP$ que son NP-hard también se encuentran en P.

b) Si P=NP, entonces toda problema NP-hard es polinomial Falso

HALTING es NP-hard y si P=NP, sigue siendo un lenguaje indecidible, por lo cual no es polinomial y sigue siendo NP-hard.

c) Si NP-C y coNP-C son disjuntas, entonces $P \neq NP$. Verdadera. ($NP \cap coNP = \emptyset$)

Sé que si $P = NP \Rightarrow NP = coNP$. Por esto, si: $NP \neq coNP$ no podría suceder que $P = NP$.

$$S \Rightarrow t \text{ si\! si } \neg S \vee t \text{ si\! si } \neg t \Rightarrow \neg S$$

d) HALTING es NP-hard y coNP-hard Verdadero

En la guía pasada demostramos que si $L \in RECURSIVE \Rightarrow L \leq_p HALTING$ (PZEQ), como se que si $L \in NP$ ó $L \in coNP$, entonces $L \in RECURSIVE$ sé que paso A, o sea, HALTING es NP-hard y coNP-hard.

Ej 5:

5. Suponiendo que P = NP, diseñar un algoritmo polinomial que dada una fórmula booleana ϕ encuentre una asignación que la satizfaga, si es que ϕ es satisfacible.

Si P=NP, entonces tengo una máq. N det. poly. que decide SAT.

Ahora bien, paso a escribir el algoritmo pedido:

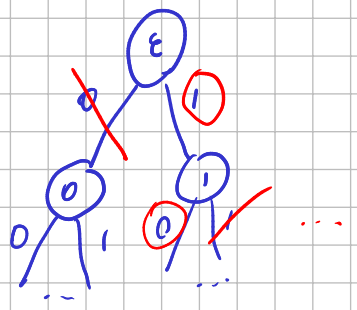
Algoritmo: $\langle \phi \rangle$

```
m := cantidad de var. proposicionales de ϕ        O(|ϕ|)
res := ε.                                          → ε = cadena vacía  O(|ϕ|)

While (m ≠ 0):                                     → Le agrego el 0 a la valuación final   O(|ϕ|) iteraciones
    res.add(0)
    IF (¬N(ϕ|res)):                                → ϕ evaluada en res (no sé si es correcta la notación)
                                                   → Si se vuelve no satisfacible le cambio el último 0 por un 1
        res[|res|-1] = 1

ret res
```

Visual:



Ej 6:

6. Suponiendo que P = NP, diseñar un algoritmo polinomial que dado un grafo G retorne una clique de tamaño máximo de G.

Si P=NP tengo una N máq. det. poly. tal que N decide el problema CLIQUE $\langle G,K \rangle$ de la práctica 2.

Algoritmo: $\langle G \rangle$

```
//Primero busco el tamaño máximo de la clique

K := |V|

while (K ≠ 0)

    IF (N(⟨G,K⟩))                  → Si encuentro un tamaño de clique que es el mayor en G salgo del loop
        break

    K--
```

//Ahora sí busco los nodos de la clique (de manera similar al pto. anterior)

```
i := 1                                   //Iterador sobre los nodos de V   O(|V|)
res := {}                                //Nodos de la clique              O(|V|)
G' := G

while(K≠0):                              //Si no tengo todos los nodos de la clique
  res := res + {V[i]}                    //Agrego otro nodo
  G' := <V'\V[i], E'\{edges de V[i]}>    //Grafo sin el nodo que se está viendo si pertenece a la clique.

  i++

  if(¬N(<G', K>)):                       //Si no existe más la clique máx el que agregue pertenece a la clique máxima.

    K--

  else:                                  //Si sigue existiendo la máxima clique posible, puse uno que podría no estar en el grafo para que exista

    res := res \ {V[i-1]}
```

Ej 7:

7. Sabiendo que CLIQUE es NP-completo, demostrar que SUBGRAPH ISOMORPHISM es NP-completo.

SUBGRAPH ISOMORPHISM: $\{\langle G, H\rangle : G$ es un grafo y H es isomorfo a un subgrafo inducido de $G\}$

CLIQUE: $\{\langle G, K\rangle : G$ es un grafo con subgrafo completo de tamaño $\geq K\}$

Ya demostré en P2E3F que S-I $\in$ NP, ahora QVQ S-I es NP-hard, esto lo logro probando que CLIQUE $\leq_p$ S-I

$x \in$ CLIQUE sii $f(x) \in$ S-I con f computable polinomialmente.

Ahora, ¿qué sería esta f?:

f toma $\langle G, K\rangle$ y declara su salida $(\langle G', H\rangle)$ de manera que $G' = G$ y H = grafo completo de K nodos.

Esta f se computa poly. ya que hacer $G' = G$ toma $O(|V|^2)$ y H toma $O(|V|^2)$.

Luego, tengo que probar que se cumple el sii:

$\Rightarrow$)
Si $\langle G, K\rangle \in$ CLIQUE, entonces existe una clique de tamaño K en el grafo G, luego, hay un grafo isomorfo a esta clique, el cual es un grafo completo de K nodos, o sea, isomorfo a H.

$\Leftarrow$)
Si $\langle G', H\rangle \in$ S-I significa que hay una clique en G' de tamaño $|V_H|$, como sé por construcción que $G' = G$ y $|V_H| = K$, esta significa que $\langle G, K\rangle \in$ CLIQUE.

γ

Ej 8:

8. Considerar los siguientes dos lenguajes:

- SHORTEST PATH (SP) $= \{\langle G, s, t, k\rangle : G$ es un grafo pesado con dos nodos s y t tales que hay un recorrido de s a t de peso menor o igual a $k\}$
- ELEMENTARY SP$= \{\langle G, s, t, k\rangle : G$ es un grafo pesado con dos nodos s y t tales que hay un camino simple de s a t de peso menor o igual a $k\}$

Demostrar que ELEMENTARY SHORTEST PATH es NP-completo y que SHORTEST PATH está en P. ¿Cuál de los dos problemas resuelven los algoritmos de camino mínimo vistos en TDA?

SP ∈ P: Utilizo Dijkstra ó Bellman-Ford para resolverlo (vistas en TDA), lo cuales poly.

E-SP ∈ NP-C:

Para esto debo ver q' E-SP ∈ NP y E-SP ∈ NP-Hard

E-SP ∈ NP:

Certificado: Conjunto de nodos tal que su longitud sea menor o igual a $|V|-1$ (porque es un camino simple, o sea, sin loops).

Algoritmo: $\langle G, s, t, K, u\rangle$ ← Certificado

- Verifico q' s esté en la primera posición de u y t en la última.
- Veo que u no tenga nodos repetidos y que su longitud sea menor a $|V|$
- Veo que todo elemento de u $\in V$.
- Veo que para $1 \leq i < |u|-1$: $(u[i], u[i+1]) \in E$
- Veo que el mínimo peso de las aristas del camino sea $\leq K$. (Pq' si no el camino tiene un peso más grande.) (Corte mínimo(?).

E-SP ∈ NP-HARD:

Puedo usar HAMPATH para la reducción $\leq_p$ (HAMPATH es NP-hard)

$$\text{HAMPATH} \leq_p \text{E-SP}$$

$$\langle G, s, t\rangle \in \text{HAMPATH} \text{ sii } f(\langle G\rangle) \in \text{E-SP}$$

Para ver como hacer una f que de $\langle G, s, t\rangle$ conseguir $\langle G', s', t', K\rangle$ y que se cumpla el sii

A ver, lo más intuitivo es hacer $G' = G$, $s' = s$, $t' = t$, el tema es que G' tiene aristas con pesas.

¿Qué pesas le pongo a las aristas entonces?

Bueno, tengo que ver qué K debería poner también. Entonces, mi K debería ser uno que me dia que si o si pasé por todos los nodos.

Si le pongo peso -1 a cada arista y $K = -n+1$, entonces si el camino es $\leq K$, no es aceptado pq' si o si no es un camino simple (igual esto se chequea fuera de ver el flujo del camino), además si el camino es $> K$, no pasó por todos los nodos (esto si lo restringe el K).

Buenísimo, quedó $G' = G$ con pesas -1 en las aristas, $s' = s$, $t' = t$, $K = -n+1$. ∎

Ej 9:

9. Probar que $\mathcal{L} = \emptyset$ y $\mathcal{L}^c$ no son NP-completos.

Si no son NP-completas, entonces existe un lenguaje $\pi \in NP$ tal q' no se cumple que

$$x \in \pi \text{ si: } x \in \mathcal{L} \text{ y } x \in \pi \text{ si } x \in \mathcal{L}^c$$

¿Cuál es el problema?

A la pregunta $x \in \mathcal{L}$ la respuesta siempre es 0, pq' no tiene ninguna palabra en el lenguaje (y para $x \in \mathcal{L}^c$ siempre es 1).

Bueno, nos conflictúa esto porque si: $x \in \pi = 1$ la ida no vale para ninguna f en $x \in \mathcal{L}$, y si $x \in \pi = 0$, la ida no vale para ninguna f, en $x \in \mathcal{L}^c$. Y así se rompió todo, por lo q' los lenguajes triviales no pueden estar en NP-C.

Ej 10:

10. Considerar los siguientes dos lenguajes:

- SUBSET-SUM $= \{\langle v_1, \ldots, v_n, k\rangle$: existe un subconjunto $V \subseteq \{v_1, \ldots, v_n\}$ tal que $\sum_{v \in V} v = k\}$
- UNARY-SUBSET-SUM $= \{\langle v_1, \ldots, v_n, 1^k\rangle : \langle v_1, \ldots, v_n, k\rangle \in$ SUBSET-SUM$\}$

a) Probar que SUBSET-SUM $\in$ NPC.

b) Probar que UNARY-SUBSET-SUM $\in$ P.

c) Concluir que la codificación de los números afecta la complejidad de los problemas. En general, si un problema sigue siendo NP-completo cuando los números de la entrada se representan en unario entonces el problema se considera **fuertemente** NP-completo.

a) S-S $\in$ NP-C:

S-S $\in$ NP:

Certificado: Conjunto de números de longitud $\leq n$

Algoritmo: $\langle\langle v_1, \ldots, v_n, K\rangle, u\rangle$

- Verifico que todo u_i tal que $u_i \in u$ pase que: $u_i = v_j$ para algún j de $1 \leq j \leq n$.
- Verifico que en u no haya más de un elemento que en $\{v_1, \ldots, v_n\}$.
- Veo que $\sum_i u_i = K$.

S-S $\in$ NP-hard:

Para ver esto tomo un problema NP-hard y veo q' tengo una f pdy computable tal q'

$$x \in \mathcal{L} \text{ si: } f(x) \in S\text{-}S$$

Uso $\mathcal{L}$ = 2-PARTITION (Esto lo obtenés desp. de replantearte cuál de todos los problemas q' conocés es NP-C)

La f que voy a plantear es la siguiente:

Dado $\langle X\rangle$ tengo que hacer $\{v_1, \ldots, v_n\}$ y un K.

Esto se ve bastante sencillo de hacer, tomo $\{v_1, \ldots, v_n\} = X$ y $K = \frac{\text{sum}(x)}{2}$ y esto es pdy

Veo q' valga el s::

Tengo $\langle X \rangle \in L$ y q.q. $\langle v_1, \dots, v_n, K \rangle \in S\text{-}S$.

$\langle X \rangle \in L$ s:i Existe una 2-partición de X tal que ambas sumen la mitad de sum(X)
s:i $v_1, \dots, v_n$ se puede partir en 2 cjtos tal que ambos sumen K, o sea la mitad de sum(X).
s:i $v_1, \dots, v_n$ tiene 1 subcjto que sume K (pq' si uno suma esta, el resto del cjto debe sumar K también).
s:i $\langle v_1, \dots, v_n, k \rangle \in S\text{-}S$.

Otra forma sería usar 3SAT (Más dificil)

$X \in 3SAT$ s:i $f(X) \in S\text{-}S$

Idea: Usar las codificaciones en bits de los números para llegar a algo q' signifique q' X es 3SAT

$(\underbrace{l_{11} \vee l_{12} \vee l_{13}}) \wedge \dots \wedge (\dots)$ = z cláusulas
$X_1 \| X_2 \| X_3$ = m variables libres

Supongo que cada X_i tiene una codificación en binario

Ahora voy a hacer una codificación para X_i y otra para $\neg X_i$

Para cada cláusula creo un número con 1's en las posiciones q' correspondan a los literales en la cláusula

El K debe tener 1's en las posiciones correspondientes a las cláusulas

	X_1	X_2	...	X_m	C_1	C_2	...	C_n
T_2	0	1	0	...	0	*	*	...
F_2	0	1	0	...	0	*	*	...

Si hay un 1 en C_j, X_2 está en C_j

– Example description for:
$(u_1 \vee \neg u_3 \vee \neg u_4) \wedge (\neg u_1 \vee u_2 \vee \neg u_4)$

	u_1	u_2	u_3	u_4	c_1	c_2
T_1	1	0	0	0	1	0
F_1	1	0	0	0	0	1
T_2	0	1	0	0	0	1
F_2	0	1	0	0	0	0
T_3	0	0	1	0	0	0
F_3	0	0	1	0	1	0
T_4	0	0	0	1	0	0
F_4	0	0	0	1	1	1

La suma binaria que vamos a usar no tiene carry a la siguiente columna por lo cual vamos a asumir q' cada bit de columna en realidad está en base 10.

Para tener un K fija necesita tener en cuenta q' las variables pueden tener apariciones en varios literales y varias cláusulas, para atajar esto uso lo siguiente

– Going back to our example:

$(u_1 \vee \neg u_3 \vee \neg u_4) \wedge (\neg u_1 \vee u_2 \vee \neg u_4)$

	u_1	u_2	u_3	u_4	c_1	c_2
T_1	1	0	0	0	1	0
F_1	1	0	0	0	0	1
T_2	0	1	0	0	0	1
F_2	0	1	0	0	0	0
T_3	0	0	1	0	0	0
F_3	0	0	1	0	1	0
T_4	0	0	0	1	0	0
F_4	0	0	0	1	1	1
$S1_1$	0	0	0	0	1	0
$S2_1$	0	0	0	0	2	0
$S1_2$	0	0	0	0	0	1
$S2_2$	0	0	0	0	0	2
Target :	1	1	1	1	4	4

Para más detalles... googlearla ^^

b) U-S-S $\in P$

La combinatoria de todos los subconjuntos q' puedo tomar serían partes de $\{v_1, \dots, v_n\}$ La cuál son 2^n elementos.

Ahora bien, si K es lo suficientemente grande ($|K| \geq 2^n$) puedo hacer la combinatoria de $\{v_1, \dots, v_n\}$ en tiempo poly, porque básicamente sería 2^n (lo cuál es poly a $n^2 \cdot 2^n$.

↓

n elementos (v_i) y tomar n bits por elemento pq' tienen q' poder codificar 2^n.

c)

Cuanto más tiempo se tarde en leer la entrada generalmente se tiene libertad de hacer más cosas al ser que el tiempo del algoritmo se mide proporcionalmente a el tamaño de la entrada.

Ej 11:

11. El problema DOUBLE-SAT consiste en deteminar si una formula proposicional ϕ tiene al menos dos valuaciones que la satisfacen. Demostrar que DOUBLE-SAT es NP-completo.

DOUBLE-SAT $\in$ NP:

Haga el algoritmo y declara el certificado:

Certificado: 2 valuaciones (v, v') con n bits (donde n es la cantidad de var. proposicionales de φ).

Algoritmo: $\langle \varphi, u \rangle$

- Chequear q' u sea de la forma (v, v') con $v, v' \in \{0,1\}^n$
- Retornar: $(v \vDash \varphi \wedge v' \vDash \varphi)$.

DOUBLE-SAT $\in$ NP-hard:

Lo más intuitiva sería reducir con SAT tal q':

$$x \in SAT \text{ si: } f(x) \in D\text{-}SAT$$

¿Qué hace f?

Por cada φ, tal que $x = \varphi$, obtengo φ' donde $\varphi' = \varphi \vee \varphi$ donde φ tiene la misma estructura q' φ pero con variables frescas.

Ej: $\varphi = x_1 \wedge (x_2 \vee x_3)$
$\varphi' = (x_1 \wedge (x_2 \vee x_3)) \vee (y_1 \wedge (y_2 \vee y_3))$

Esto es polinomial pq' lo más costoso sería poner φ' en CNF y esto es poly pq' es hacer una distributiva.

Esta f vale pq' cumple que si $\varphi \in SAT$ $\varphi' \in 2\text{-}SAT$, ya q' suponiendo q' $v \vDash \varphi$, $vw \vDash \varphi'$ y $wv \vDash \varphi'$ con $w = \{0,1\}^n$ cualquiera.

Ej 12:

12. Probar que k-CLIQUE está en P para cualquier $k \in \mathbb{N}$. Concluir que dejar parámetros fijos puede cambiar la complejidad de los problemas.