Guia 3 Complejidad Computacional 1 Cuatrimestre 2025

Juan DElia

Ejercicio 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) P $\subseteq$ NP y P $\subseteq$ coNP

P $\subseteq$ NP ?

Si, si los puedo resolver en tiempo polinomial puedo dar un certificado y verificarlo en tiempo polinomial

P $\subseteq$ coNP ? Si. Tengo que ver $\forall \Pi \in P \Rightarrow \Pi \in coNP$

puedo decir, por defiicion de coNP:

$\forall \Pi \in P \Rightarrow \Pi \in coNP \Rightarrow \Pi c \in NP$

Que es verdadero porque Π pertenece a NP y esta cerrado por ele complemento.

b) Si P=NP entonces coNP = NP

veo primero $NP \subseteq coNP$:

como P = NP:

$$NP \subseteq coNP \iff P \subseteq coNP$$

Que vimos que es verdadero en el inciso anterior.

ahora veo $coNP \subseteq NP$, quiero ver:

$$\forall \Pi \in coNp \Rightarrow \Pi \in NP$$

Desarrollo el lado izquierdo hasta llegar:

$$\forall \Pi \in coNp \Rightarrow \Pi^c \in NP \Rightarrow^{P=NP} \Pi^c \in P$$

Como la clase P es cerrada por el complemento (sigue la cadena de $\Rightarrow$):

$$\Rightarrow \Pi \in P \Rightarrow^{P=NP} \Pi \in NP$$

Como queria ver.

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

Falso, Halt (cualquier problema que no sea NP creo)

d) Si coNP = NP , SAT $\in$ coNP

Verdadero, basta con ver SAT $\in$ NP que vimos en clase que es verdad (dar certificado y maquina…).

e) Si coNP $\subseteq$ NP, entonces NP = coNP

Para que sean iguales basta con ver NP $\subseteq$ coNP (la vuelta ya esta en la hipotesis):

quiero ver que:

$\forall \Pi \in NP \Rightarrow \Pi \in coNP$

Como por hipotesis todo elemento de coNP esta en NP (desarrollo la expresion izquierda):

$\forall \Pi \in NP \Rightarrow \Pi \in coNP \Rightarrow \Pi \in NP$

Que es verdadero siempre.

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

Veamos si es cierto.

Recordar que para L ser en NPC cumple: - $L \in NP$ - Es NP-Hard: $\forall L'.L' \in NP$ vale $L' \leq_p L$

vale $\Pi \leq_p \Gamma$?:

como $\Gamma \in$ NPC es NP-hard y $\Pi \in$ NP $\Rightarrow \Pi \leq_p \Gamma$

$\Gamma \leq_p \Pi$?:

como $\Pi \in$ NPC es NP-hard y $\Gamma \in$ NP $\Rightarrow \Gamma \leq_p \Pi$

Entonces Vale.

Ejercicio 3 Sean Π y Γ dos lenguajes tq: $\Pi \leq_p \Gamma$ que se puede inferir?

a) $\Pi \in P \Rightarrow \Gamma \in P$

Falso

b) $\Gamma \in P \Rightarrow \Pi \in P$

Verdadero

c) $\Gamma \in NPC \Rightarrow \Pi \in NPC$

Falso, Π podria no ser NP-hard, no se si es mas dificil que cualquier NP

d) $\Pi \in NPC \Rightarrow \Gamma \in NPC$

Falso Γ podria no ser NP, Por ejemplo Halt

e) $\Gamma \in NPC$ **y** $\Pi \in NP \Rightarrow \Pi \in NPC$

Falso Π podria no ser hard

f) $\Pi \in NPC$ **y** $\Gamma \in NP \Rightarrow \Gamma \in NPC$

Verdadero

g) Π **y** Γ **no pueden pertenecer ambos a NPC**

Falso, cada vez que hacemos una reduccion para ver que uno sea NPC vemos que sea igual o mas dificil que otro NPC

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

a) Si P = NP, entonces todo problema NP-completo es polinomial

Verdadero. Si un problema esta en NPC, por definicion esta en NP. Entonces por hipotesis esta en P.

b) Si P = NP, entonces todo problema NP-hard es polinomial.

Falso. Los problemas que no sean NP y sean NP-Hard no van a ser Polinomiales, por ejemplos los de la clase exp o Halt.

c) Si las clases NP-completo y coNP-completo son disjuntas entonces P $\neq$ NP.

Es verdadero, supongamos por el absurdo que P=NP: P=NP $\rightarrow$ coNP = NP (ver ejercicio 1)

Absurdo! Por hipotesis NP-completo y coNP-completo son disjuntas. Por lo tanto P debe ser distinto a NP

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

Verdadero.

Halting segurisimo es mas dificil que cualquier NP, y creo que el complemento de halting tampoco es computable asi que tambien.

Ejercicio 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.

Por hipotesis SAT $\in$ P:

Existe un programa SAT(ϕ) que devuele V si se puede satisfacer ϕ, F caso contrario.

Puedo definir tambien SAT-R(ϕ,r,k) que toma k restricciones y restricciones es una lista de boleanos tq: r_i es el valor que debe tener la x_i variable de ϕ.

Por ejemplo SAT-R(ϕ,[True,False],2) verifica si ϕ es satisfactible con la primera variable siendo T y la segunda F.

Sigue perteneciendo a NP (y por hipótesis a P) ya que la unica diferencia es que cuando se verifica el certificado (valuacion tq la formula sea satisfactible) hay que chequear que las primeras k variables del certificado sean las mismas que las restricciones.

Con esto en mente defino un algoritmo:

```
def asignacion(phi):
    if SAT(phi):
        asignacion = []
        for i in range(len(phi.variables)):
            if SAT-R(phi,asignacion + [True],i+1):
                asignacion.push(True)
            else:
                asignacion.push(False)
        return asignacion
    else:
        print("no hay asignacion valida")
```

Basicamente va dejando fijas las variables y chequea si fijando Verdadero se puede satisfacer, sino la fija falso. Hace eso con todas hasta obtener una asignacion valida.

El programa corre en tiempo polinomial, pues hace cantidad de variables iteraciones, que es polinomial respecto de ϕ. En cada iteracion corre SAT-R que por hipótesis es polinomial.

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

Por hipotesis tengo una maquina polinomial Clique(g,k) que me dice si G tiene una clique de tamaño k.

Busco primero el tamaño maximo de clique en G. Para cada nodo n chequeo si G-n sigue teniendo clique de tamaño maximo. Si se mantiene significa que n es dispensable asi que lo saco del grafo. Cuando termine de recorrer solo quedan nodos "indispensables" que forman la clique asi que devuelvo los nodos restantes en el grafo.

```
def max_clique(g):
    tamaño = 0
    for i in range(|g.v|+1):
        if clique(g,i):
            tamaño += 1
        else:
            break

    for nodo in g.v:
        if(clique(g.sin_nodo(nodo),tamaño)):
            // si hay clique de tamaño maximo en el grafo sin n, saco a n
            g.sacar_nodo(n)
    return g.v // retorno los que quedaron, es decir los que forman la clique
```

Calcular el tamaño maximo es polinomial, por cada nodo del grafo que es polinomial respecto a su tamaño realizo una operación polinomial (clique).

Despues Simplemente vuelvo a recorrer la lista y hago operaciones polinomiales.

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

Quiero ver:

$$\langle g, k \rangle \in CLIQUE \iff f(\langle g, k \rangle) \in SUBGRAPHISOMORPHISM$$

F va a tomar G, k y deolver G, H_k

Con H_k un grafo completo de tamaño k.

Veamos que vale:

$$\langle g, k \rangle \in CLIQUE \iff \langle G, H_k \rangle \in SUBGRAPHISOMORPHISM$$

$\Rightarrow$)

$\langle g, k \rangle \in CLIQUE \Rightarrow$ G tiene subgrafo completo de tamaño k, lo llamo G_k

Notar que G_k es isomorfo a **cualquier** grafo completo de tamaño k.

$\Rightarrow G$ es un grafo y H_k es isomorfo al grafo inducido de G $G_k \Rightarrow \langle G, H_k \rangle \in SUBGRAPHISOMORPHISM$

$\Leftarrow$)

$\langle G, H_k \rangle \in SUBGRAPHISOMORPHISM \Rightarrow G$ grafo y H_k es isomorfo a un grafo inducido de G

$\Rightarrow$ G tiene un subgrafo de tamaño k completo $\Rightarrow \langle G, k \rangle \in CLIQUE$

Ejercicio 8 Considerar los siguientes dos lenguajes (consultar):

SHORTEST PATH (SP)

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

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?

a)

Para ver que Shortest Path pertenece a P propongo:

```python
def shortest_path(g,s,t,k):
    vector_distancia = dijkstraa(g,s)
    dist_s_t = vector_distancia[t]
    return dist_s_t <= k
```

Que es polinomial porque el algoritmo de dijkstraa es polinomial

b)

Pertenece a NPC?

Certificado: Camino sin repetir nodos de s a t con peso menor a k

Verificador: Chequear que efectivamente ese camino esta en el grafo, que no tiene nodos repetidos y que la longitud del certificado es menor o igal a K.

Es NP-Hard?

Dijkstraa no se puede forzar a no repetir nodos

No se que reduccion puedo hacer con los lenguajes que sabemos que son NPC, ninguno tiene que ver con caminos… Consultar

Ejercicio 9 Probar que $L = \emptyset$ y L^c no son NPC

Para que sean NPC tienen que ser NP (lo son ambos son polinomiales) y ser NP-hard osea que para todo $L' \in NP$ vale que $L' \leq_p L$

a) veamos que no vale la reduccion para $\emptyset$

$$x \in L' \iff f(x) \in \emptyset$$

Esto es absurdo, la parte derecha del si solo si siempre va a ser falsa. Ningun elemento pertenece al vacio

b) veamos que no vale la reduccion para el complemento, llamemoslo U

$$x \in L' \iff f(x) \in U$$

Tambien es absurdo. Para que sea verdad tendria que valer que todo x pertenece a L' arbitrario.

Ejercicio 10 (consultar)

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.

Es NP?

Certificado: El subconjunto que suma k

Verificador: recorrer el certificado sumando cada elemento, hay que chequear que cada uno pertenece al conjunto original y que sum(certificado) = k

no le encuentro la vuelta a la reduccion

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**.

Ejercicio 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?

Certificado: las dos valuaciones. Verificador: Reemplazar variable a variable una vez para cada valuacion y ver que dan verdadero.

(es lo mismo que verificar SAT pero dos veces)

DOUBLE-SAT es NP hard?

Veamos la siguiente reduccion:

$$\phi \in SAT \iff f(\phi) \in DOUBLE-SAT$$

Idea:

Si $\phi(v) \to V \Rightarrow \phi(v) \wedge V \to V \to \phi(v) \wedge (z_1 \vee \neg z_1) \to V$

Entonces mi funcion (trivialmente polinomial) va a ser:

f(ϕ) $\to$ $\phi \wedge (z_1 \vee \neg z_1)$ (z_1 variable fresca) y quiero ver que:

$$\phi \in SAT \iff \phi \wedge (z_1 \vee \neg z_1) \in DOUBLE-SAT$$

$\Rightarrow$) a $\phi \wedge (z_1 \vee \neg z_1)$ lo voy a notar ϕ'

$\phi \in SAT \Rightarrow \exists v.\phi(v) \to V \Rightarrow (\phi'(v+\{z_1 = V\}) \to V \wedge \phi'(v+\{z_1 = F\}) \to V) \Rightarrow \phi' \in DOUBLE-SAT$

$\Leftarrow$)

$\phi' \in DOUBLE-SAT \Rightarrow \exists v_1 v_2.\phi'(v_1) \to V \wedge \phi'(v_2) \to V \Rightarrow^{nota_1} \phi(v_1) \to T \Rightarrow \phi \in SAT$

$nota_1$:

El implica vale porque ϕ' esta en CNF $C_1 \wedge C_2 \wedge ... \wedge C_n$. Si una valuacion es verdadera para $C_1 \wedge C_2 \wedge ... \wedge C_n$, sigue siendo verdadera para cualquier formula producto de sacarle alguna C_i porque es una cadena de conjunciones.

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

Recordatorio = $\binom{n}{k} = O(n^k)$

Propongo el siguiente algoritmo:

1. Armar todos los subconjuntos de vertices de tamaño k. $O(n^k)$
2. Para cada subconjunto, recorrer la matriz para chequear si estan todos conectados. $O(n^k.n^2)$

complejidad total: $O(n^k.n^2)$ que es polinomial con k fijo.

Ejercicio 13 El problema HALF-CLIQUE consiste en determinar si un grafo G de tamaño n tiene un completo de tamaño n/2. Sabiendo que CLIQUE es NP-completo, demostrar que HALF-CLIQUE es NPcompleto. ¿Por qué este resultado no contradice el hecho de que k-CLIQUE es polinomial para todo k?

$HALF - CLIQUE \in NP$?:

Certificado: Conjunto de nodos que forman la half clique

Verificador: Ver que esten todos conectaadosy que el |certificado| $\geq$ n/2

Es NP-Hard? Veo la reduccion:

$< G, k >\in CLIQUE \iff f(< G, k >) \in HALF - CLIQUE$

Idea para la funcion:

Si g es mas chico que n/2 puedo armar g' agregando nuevos nodos que formen una clique y que cada nodo se conecte a los originales de g. De esta manera si en G tenia una clique de tamaño k, g' va a tener clique de tamaño k+x, siendo x el tamaño de la clique nueva.

Cuantos nodos tiene que tener esa clique de x nodos? Veamos que se tiene que cumplir lo siguiente:

$$n' = n + x$$
$$\frac{n'}{2} = k + x$$

Siendo n' la cantidad de nodos de G'. Quiero que $\frac{n'}{2}$ sea la cantidad de nodos de la clique original mas los que tenga que agregar

Es un despeje sencillo y obtenemos:

$$x = n - 2k$$

$$n' = n + n - 2k = 2n - 2k$$

Por lo que el tamaño de la CLIQUE en G' va a ser:

$$\frac{n'}{2} = n - k$$

Propongo la siguiente funcion:

CORREGIR LA VUELTA NO ANDA FALTA EL CASO ELSE AGREGANDO NODOS SUELTOS

```
f(G,k):
    if k < len(G.v):
        return el grafo "G'" igual a G y con una clique nueva de tamaño (n-k)/2, donde cada
        esta conectado a cada nodo de G
    else:
        return G
```

Veamos que es verdadero:

$$< G, k > \in CLIQUE \iff G' \in HALF - CLIQUE$$

$\Rightarrow$)

$$< G, k > \in CLIQUE \Rightarrow^{nota_1} < G', k+n-2k > \in CLIQUE \Rightarrow \; < G', n-k > \in CLIQUE \Rightarrow$$

$$< G', \frac{n'}{2} > \in CLIQUE \Rightarrow \; < G' > \in HALF - CLIQUE$$

$nota_1$: G' tiene clique con x (x=n-2k) mas nodos que G por como la defini.

$\Leftarrow$)

$$G' \in HALF - CLIQUE \Rightarrow < G', n - k > \in CLIQUE \Rightarrow^{nota_2}$$

$$< G, (n - k) - (n - 2k) > \in CLIQUE \Rightarrow < G, k > \in CLIQUE$$

$nota_2$: G tiene clique con x menos nodos que en G' (x=n-2k).

Ejercicio 14 Demostrar que TAUTOLOGY es coNP-completo

$$TAUTOLOGY^c = \{\langle\phi\rangle : \exists v.\phi(v) \vdash F\}$$

Ver que TAUTOLOGY esta en coNP es ver que su complemento esta en np:

Certificado: Valuacion para ϕ que cuando se evalua da falso.

Verificador: reemplazar cada variable de ϕ por su valor en la valuacion. Claramente polinomial

Ahora hay que ver que $TAUTOLOGY^c$ es np hard. Voy a reducrir SAT a $TAUTOLOGY^c$:

$$\phi \in SAT \iff f(\phi) \in TAUTOLOGY^c$$

Si una formula es satisfacible, tiene al menos una valuacion que la hace verdadera. El complemento de esa formula entonces tendra al menos una valuacion que la haga falsa.

Propongo f(ϕ) $\rightarrow \neg\phi$ (negar una formula es polinomial)

veamos que vale:

$$\phi \in SAT \iff \phi' = \neg\phi \in TAUTOLOGY^c$$

$\Rightarrow$)

$$\phi \in SAT \Rightarrow \exists v.\phi(v) \vdash V \Rightarrow \neg\phi(v) \vdash F \Rightarrow \phi'(v) \vdash F \Rightarrow \phi'(v) \in TAUTOLOGY^C$$

$\Leftarrow$)

$$\phi' \in TAUTOLOGY^C \Rightarrow \neg\phi \in TAUTOLOGY^C \Rightarrow \exists v.\neg\phi(v) \vdash F \Rightarrow \phi(v) \vdash V \Rightarrow \phi \in SAT$$