

Guia 3 Complejidad

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 deficiencia 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 el 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:

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

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

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

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 podría no ser NP-hard, no se si es mas difícil que cualquier NP

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

Falso Γ podría no ser NP, Por ejemplo Halt

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

Falso Π podría 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 reducción para ver que uno sea NPC vemos que sea igual o mas difícil 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 segurísimo es mas difícil que cualquier NP, y creo que el complemento de halting tampoco es computable así que también.

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

Por hipótesis $SAT \in P$:

Existe un programa $SAT(\phi)$ que devuelve V si se puede satisfacer ϕ , F caso contrario.

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

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

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

También se podría escribir reducción para ver $SAT(\phi) \leq_p SAT-R(\phi, r, k)$:

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

```
f(phi):
    return < phi, [], 0 >
```

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 asignación válida.

El programa corre en tiempo polinomial, pues hace cantidad de variables iteraciones, que es polinomial respecto de ϕ . En cada iteración 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 sako 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, sako 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 NP-completo

Quiero ver:

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

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

Con H_k un grafo completo de tamaño k .

Veamos que vale:

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

\Rightarrow)

$\langle g, k \rangle \in \text{CLIQUE} \Rightarrow G$ 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 $G_k \Rightarrow \langle G, H_k \rangle \in \text{SUBGRAPHISOMORPHISM}$

\Leftarrow)

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

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

Ejercicio 8 Considerar los siguientes dos lenguajes:

SHORTEST PATH (SP)

$SP = \{\langle G, s, t, k \rangle : G \text{ es un grafo pesado con dos nodos } s \text{ y } t \text{ tales que hay un recorrido de } s \text{ a } t \text{ de peso menor o igual a } k\}$

ELEMENTARY SP

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

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:

```
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 igual 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

Considerar los siguientes dos lenguajes:

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

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 $\text{sum}(\text{certificado}) = k$

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**.