

## Guia 3 Complejidad

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

Por hipótesis  $SAT \in P$ :

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

Por ejemplo  $SAT-R(\phi, [True, False], 2)$  verifica si  $\phi$  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  $\phi$ . 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.