③ Problemas NP-Completos:

Resumen. Hasta ahora tenemos los problemas de decisión "fáciles" que forman $\mathcal{P}$ y los problemas "de fácil verificación" que forman NP

Sabemos que $P \subseteq NP$ pero no si son iguales.

La clase anterior mencionamos un hecho SORPRENDENTE, que existen algunos problemas en NP que capturan TODOS los problemas NP en el siguiente sentido:

Def: Un lenguaje $L \subseteq \{0,1\}^*$ es NP-completo si cumple:

(1) $L \in NP$

(2) $\forall\ E \in NP$ tenemos $E \leqslant_P L$

Cualquier algoritmo para decidir L puede usarse para decidir CUALQUIER OTRO problema en NP, así que: (1) Deberíamos buscar algoritmos eficientes para problemas y (2) Si demostramos que un problema que nos interesa es NP-completo habremos probado que es computacionalmente difícil (Si $P \neq NP$ como muchos creemos)

Hoy daremos ejemplos de problemas NP-completos básicos (3-SAT) (sin demostración pues es un resultado difícil) y usando este demostraremos que otros problemas que no tienen relación aparente (CLIQUE y VERTEX-COVER) son NP-completos.

(3.1) Un problema NP-Completo:

Def: Una fórmula del Cálculo proposicional en 3-CNF (Conjunctive Normal Form) es así:

disyunción "o"

$$\underbrace{(x_1 \vee \overline{x_1} \vee \overline{x_3})}_{\text{Cláusula}} \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) = \varphi_1$$

Conjunción "y"

negación de x_1 "no x_1" $\neg x_1$

letras proposicionales

La fórmula está en 3-CNF si cada cláusula tiene exáctamente 3 letras proposicionales o sus negaciones

$$\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_k$$

número de cláusulas es k cualquiera

cada $C_i = x_{i,1}^{\pm} \vee x_{i,2}^{\pm} \vee x_{i,3}^{\pm}$

Teorema [Cook-Levin] El problema [3-SAT]: Dada una formula φ en 3-CNF existe una colección de valores de verdad que la haga verdadera? es NP-COMPLETO.

(La demostracion es extensa, consiste en codificar un verificador y los certificados mediante fórmulas, Cook recibe el Turing Award en 1982 por esto).

En nuestro ejemplo $x_1 = x_2 = 0$ y $x_3 = 1$ producen:

$$\varphi(0,0,1) = (0 \vee \bar{0} \vee 1) \wedge (\bar{0} \vee 0 \vee 1) \wedge (0 \vee 0 \vee 1) = 1 \wedge 1 \wedge 1 = 1$$

$$(x_1 \vee \overline{x_1} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) = \varphi_1$$

así que SI existe una asignación que valide a φ y la asignación sirve como certificado.

Encontrar tal asignación directamente a partir de la fórmula parece difícil... Si encontráramos un algo polynomial para ESTE problema, habríamos probado que P=NP !!

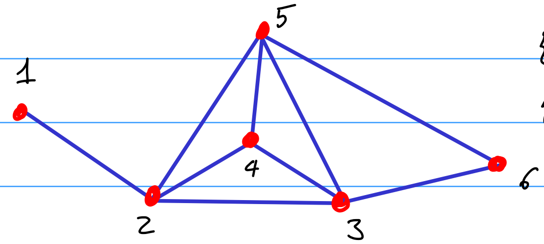
(3.2) Reducciones polinomiales:

Ahora usaremos [Cook-Levin] para demostrar que otros problemas son NP-completos. La idea es

Lema: Suponga que A es NP-completo y que $B \subseteq \{0,1\}^*$ cumple: (i) $B \in NP$ y (ii) $A \leq_p B$.
Entonces B también es NP-Completo.

Dem: Sea $L \in NP$ un lenguaje cualquiera.
Como A es NP-completo sabemos que $L \leq_p A$
Por hipótesis (ii) sabemos que $A \leq_p B$
y por transitividad de $\leq_p$ concluimos que $L \leq_p B$
Por hipótesis (i) sabemos que $B \in NP$ así que B es NP-completo.

Ejemplo: **Def:** Un clique en un grafo G es un subconjunto $W \subseteq V(G)$ que están todos conectados entre sí
(i.e. $\forall w_1, w_2 \in W \left(w_1 \neq w_2 \Rightarrow (w_1, w_2) \in E(G) \right)$)



$\{2,3,4,5\}$ son un clique y
$\{3,4,5,6\}$ NO son un clique.

Más formalmente el problema CLIQUE pregunta: Dado un grafo G y un entero positivo p, tiene G un clique con $\geq p$ vértices?

$$\underset{\{0,1\}^*}{\underset{\cap}{[CLIQUE]}} = \{ \langle G, p \rangle : G \text{ si tiene un clique de tamaño } \geq p \}$$

Mostraremos que [CLIQUE] es NP-COMPLETO en dos pasos:

(1) "fácil": $[CLIQUE] \in NP$. Para ello debemos aclarar:

ⓐ ¿Qué certificados usaremos? Los índices de los vértices $(i_1, \ldots, i_p)$ que forman el CLIQUE

(b) ¿Cómo funcionaría el verificador? Recorre las $\leq \binom{n}{2}$ parejas de vértices del certificado (a,b) y verifica que $(a,b) \in E(G)$, de lo contrario retorna 0. Esto requiere $O(n^2)$ pasos, polinomial en la longitud del input.

(2) "difícil" $[\text{3-SAT}] \leq_P [\text{CLIQUE}]$

Recuerda que esto significa que para toda fórmula $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ existen un grafo G y un entero p (que dependen de φ y pueden construirse en tiempo polinomial) tales que

$$\text{3-SAT}(\varphi) = 1 \iff \text{CLIQUE}\left(G(\varphi), p(\varphi)\right) = 1$$

Construir tales reducciones siempre requiere una idea:

$$\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$$

C_1: disyunción de tres letras $\ell_1^{(1)} \vee \ell_2^{(1)} \vee \ell_3^{(1)}$

cada $\ell_i^{(j)} = \pm$ variable

C_k: $\ell_1^{(k)}, \ell_2^{(k)}, \ell_3^{(k)}$

Crearemos un grafo $G(\varphi)$ y $p(\varphi)$ así:

Vértices: $3 \cdot k$ $\quad v_i^{(j)}$ para $j = 1, \ldots, k$ y $i = 1, 2, 3$

Aristas: Conectamos $v_i^{(j)}$ con $v_t^{(r)} \iff$ (1) $j \neq r$ y

(2) $v_i^{(j)}$ y $v_t^{(r)}$ son "consistentes" entre sí, es decir si $v_i^{(j)}$ no es $\neg v_t^{(r)}$

$p(\varphi) := k$ (el número de cláusulas).

Si tenemos una asignación de las letras proposicionales que valide la fórmula entonces debe haber al menos una escogida en cada cláusula y esas escogencias nunca son inconsistentes produciendo un $p(\varphi) = k$-clique en $G(\varphi)$. Recíprocamente un k-clique en $G(\varphi)$ elige exactamente un vértice de $v_i^{(r)}$ para $i = 1, 2, 3$ en cada $r = 1, \ldots, k$, llamelo $v_{i(r)}^{(r)}$. Seleccionando las letras proposicionales $v_{i(r)}^{(r)} = 1$ y las demás 0 obtenemos una asignación válida para φ.