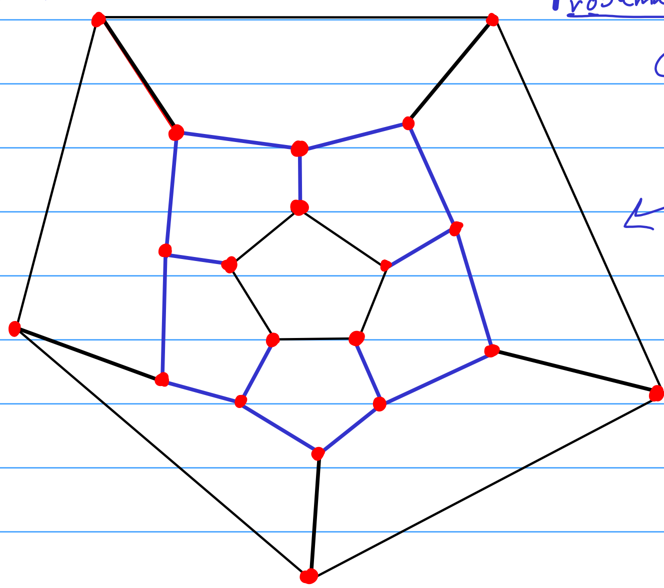
② La clase de complejidad NP:

A veces es más fácil verificar que una solución propuesta para un problema es correcta que construir esta solución desde cero

Ejemplo

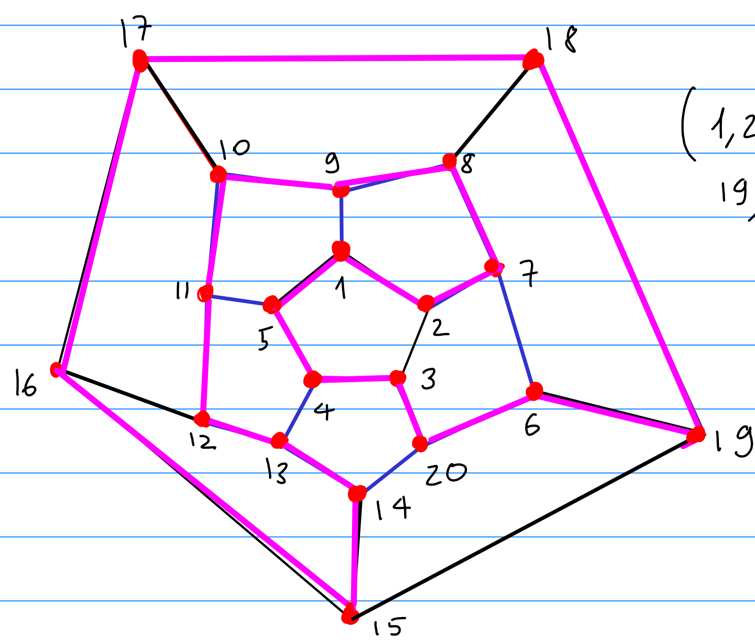


Problema 1: ¿Existe un ciclo hamiltoniano en este grafo?

"Ciclo que visita todo vértice exáctamente una vez"

(Es el grafo formado por los 12 pentágonos de un balón de fútbol)

Problema 2: Es — un ciclo hamiltoniano?

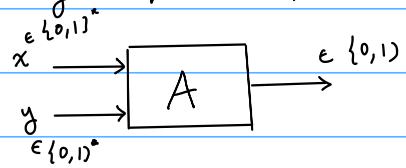


$(1,2,7,8,9,10,..,14,15,16,..,19,$
$19,6,20,3,4,5) = y$

la idea es llevar esta idea intuitiva como herramienta a la teoría de complejidad

(máquina de Turing determinista)

Def: Un verificador es un algoritmo A con dos inputs binarios y output en $\{0,1\}$



Def: El <u>lenguaje verificado</u> por un verificador A es

$$L = \{ x \in \{0,1\}^* : \exists\, y \in \{0,1\}^* \text{ con } A(x,y) = 1 \}$$

Lo anterior nos permite definir la clase de complejidad NP así

$$\mathbf{NP} = \left\{ \begin{array}{l} L \subseteq \{0,1\}^* : \exists\, k_L, c_L \in \mathbb{N} \text{ y un verificador } A_L(\cdot,\cdot) \text{ que cumplen:} \\ \quad (1)\ x \in L \iff \exists\, y \text{ con } |y| = O(|x|^{c_L}) \text{ con } A(x,y) = 1 \\ \quad (2)\ A(x,y) \text{ se ejecuta en tiempo } \underbrace{O\left((|x|+|y|)^{k_L}\right)}_{\text{polinomial en } |x|} \end{array} \right\}$$

Lenguajes verificables en tiempo polinomial (en particular el certificado debe tener longitud polinomial).

Obs: En principio, encontrar el certificado podría ser una tarea computacionalmente mucho más difícil, para demostrar que un problema está en NP basta definir qué tipo de certificados admitimos y un algoritmo que verifique la validez del certificado.

[CH]

Ejemplo: Demuestre que CICLO-HAMILTONIANO está en NP

Sea $\langle G \rangle$ la codificación fija de un grafo G.
Módulo factores polinomiales su longitud es $|x| = n+m$
donde $n = |V(G)|$ y $m = |E(G)|$.
Un certificado para [CH] es un ordenamiento de los vértices $y = (v_1, v_2, \dots, v_n)$ formando un ciclo Hamiltoniano.
Luego $|y| = n$.
El verificador hace lo siguiente:
(1) Recibe el input $(\vec{x}, \vec{y})$
(2) Chequea que $\vec{x} = \langle G \rangle$ para algún grafo G y que $\vec{y}$ sea una sucesión de vértices de G; contrario retorna 0.
(3) Verifica que $\vec{y} = (v_1, \dots, v_n)$ cumpla $(v_i, v_{i+1}) \in E(G)$ $i = 1, \dots, n-1$ y $(v_n, v_1) \in E(G)$ y que la lista $v_1, \dots, v_n$ no tiene repeticiones y en ese caso retorna 1, si alguna de estas falla retorna 0

tiempo del verificador $O(\underbrace{n+m}_{①} + \underbrace{n+m}_{②} + \underbrace{2n + n\log(n)}_{③})$

Por construcción $A(x,y) = 1 \iff$ x codifica un grafo G y y codifica un ciclo Hamiltoniano en G

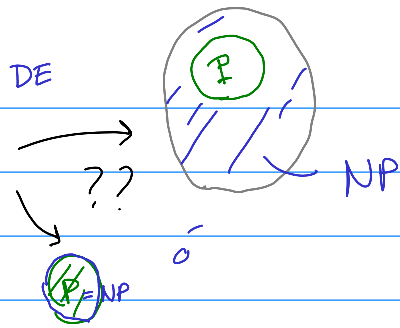
luego $\{x \in \{0,1\}^* : \exists\, y \;\; A(x,y) = 1\} = [CH]$
y por eso $[CH] \in NP$.

Lema: $P \subseteq NP$ (fácil...)

Dem: Si $L \in P$ existe k_L y un algoritmo $A_L(\cdot)$ que decide L. Definiremos $\tilde{A}(x,y) = A_L(x)$
$\{x \in \{0,1\}^* : \exists\, y$ con $\tilde{A}(x,y) = 1\} = L$

y $\tilde{A}$ ejecuta en tiempo polinomial en $|x|$ luego también en $|x|+|y|$.

LA PREGUNTA MÁS IMPORTANTE DE LA TEORÍA DE COMPUTACIÓN: ES $P = NP$?



③ Reducibilidad Polinomial:

Ahora queremos comparar la dificultad relativa de problemas de decisión (ignorando cantidades polinomiales). Definiremos esta dificultad relativa a nivel de lenguajes.

Def: Una función $f: \{0,1\}^* \longrightarrow \{0,1\}^*$ es calculable en tiempo polinomial si existe un entero k_f algoritmo A que cumple:

(1) $\forall x \in \{0,1\}^* \quad A(x) = f(x)$ y

(2) A calcula $A(x)$ en tiempo $O(|x|^{k_f})$

Def: $L_1 \leq_p L_2$

L_1 es "reducible en tiempo polinomial" a L_2

L_1 es "reducible módulo factores polinomiales" a L_2

si existe una función $f: \{0,1\}^* \longrightarrow \{0,1\}^*$ (función testigo de $L_1 \leq_p L_2$) calculable en tiempo polinomial que cumple

$$\forall x \in \{0,1\}^* \left(x \in L_1 \iff f(x) \in L_2 \right)$$

Qué significa esto? Si supiéramos que $[CH] \leq_p [B]$ para algún problema B y tuviéramos un buen algoritmo Solve_B para B y una función f podríamos resolver una instancia $\langle G \rangle$ de CH así: Solve_B$(f(\langle G \rangle))$

f es la función testigo de $[CH] \leq_p [B]$

Las funciones f son muy, muy importantes pues nos dan NUEVOS ALGORITMOS, permitiéndonos reutilizar el de [B] para resolver [CH].

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

(1) $C \in \mathbf{NP}$

(2) $\forall\ L \in NP,\ L \leqslant_P C$

En palabras, si hubiera un algoritmo rápido para C, podría usarse para construir algoritmos rápidos para TODO problema en NP.

Concluiremos con un hecho SORPRENDENTE:
Existen PROBLEMAS DE DECISIÓN NP-COMPLETOS.

Ejemplo: Existen valores $x_1, x_2, x_3 \in \{0,1\}$ que hagan esta fórmula verdadera?

$$\Big[\big[(\neg x_3) \wedge x_1 \wedge x_2\big] \wedge x_3\Big] \vee \Big[(x_1 \vee x_2 \quad \vee x_3)\Big] \vee (\neg x_3)$$

Más precisamente: ¿Dado un circuito booleano, existe una escogencia de las variables input que la haga verdadera? es **NP-completo**.

Ejemplo: CICLO HAMILTONIANO es **NP-completo**