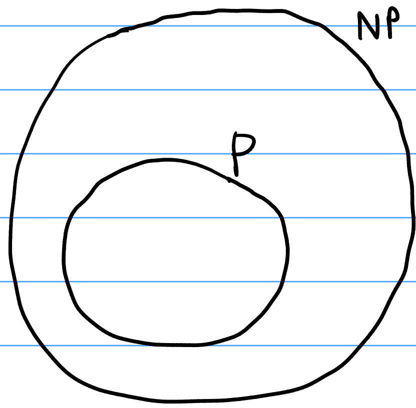
Clase $P = \{ A \subseteq \Sigma^* \mid A$ es decidible en tiempo polinomial $\}$

Clase $NP = \{ A \subseteq \Sigma^* \mid A$ es verificable en tiempo polinomial $\}$

$= \{ A \subseteq \Sigma^* \mid A$ es decidible en tiempo polinomial por una M.T.N. $\}$



Algunos problemas de la clase NP son "especiales".

Existen lenguajes $A \in NP$ tales que si supiéramos decidir A en tiempo polinomial, eso nos permitiría decidir en tiempo polinomial TODOS los lenguajes de la clase NP.

Lenguajes NP-completos

- Dado un lenguaje A con estas características, podríamos probar que $NP \subseteq P$ si encontráramos un algoritmo para decidir A en tiempo polinomial.

- A veces nos puede ser útil demostrar que un lenguaje es NP-completo.

- Tenemos variables $(x, y, z, \dots)$ que pueden tomar valores booleanos ($0 = \text{False}$, $1 = \text{True}$).

- A partir de las variables podemos formar fórmulas lógicas usando los conectivos $\underset{\text{and}}{\wedge}, \underset{\text{or}}{\vee}, \underset{\text{not}}{\neg}$.

- Por ejemplo:

$$\varphi = (\underbrace{\neg x}_{\bar{x}} \wedge y) \vee (x \wedge \underbrace{\neg z}_{\bar{z}})$$

$$\boxed{\begin{array}{l} x \mapsto 0 \\ y \mapsto 1 \\ z \mapsto 0 \end{array}}$$

- Dada una fórmula φ, decimos que φ es <u>SATISFACTIBLE</u> si se puede encontrar una asignación de variables a valores booleanos que hace verdadera a la fórmula.

- Por ejemplo $x \wedge \bar{x}$ no es satisfactible.

<u>Def</u>. El lenguaje SAT se define así:

$$SAT = \{ \langle \varphi \rangle \mid \varphi \text{ es una fórmula satisfactible} \}.$$

<u>Obs</u>. El lenguaje SAT es decidible.

Va a ser un ejemplo de lenguaje NP-completo.

<u>Teorema</u> (Cook-Levin).

Si $SAT \in P$, entonces $NP \subseteq P$.
$(P = NP)$.

(Demostración más tarde en esta clase).

Reducciones polinomiales

Habíamos visto dos nociones de reducción entre lenguajes:

- $A \leq_m B$ ("many-one") (B decidible $\Rightarrow$ A decidible)

- $A \leq_T B$ ("Turing-reducibilidad") (B decidible $\Rightarrow$ A decidible)

<u>Def</u>. Una función $f : \Sigma^* \longrightarrow \Sigma^*$ es <u>computable en tiempo polinomial</u> si existe una M.T. M que siempre termina, tiene tiempo de ejecución en peor caso polinomial y además

$\forall w \in \Sigma^*$. $M(w)$ termina con la palabra $f(w)$ escrita en la cinta.

<u>Def</u>. Dados dos lenguajes $A, B \subseteq \Sigma^*$ decimos que <u>A se reduce polinomialmente a B</u> (se escribe $A \leq_p B$) si existe una función $f : \Sigma^* \to \Sigma^*$ computable en tiempo polinomial tal que:

$$\forall w \in \Sigma^*. \quad w \in A \iff f(w) \in B.$$

<u>Obs</u>. Si $A \leq_p B$ entonces $A \leq_m B$.

<u>Obs</u>. Si $A \leq_p B$ y B es decidible, A es decidible.

Teorema. Si $A \leq_p B$ y $B \in P$ entonces $A \in P$.

Dem. • Supongamos que $A \leq_p B$.
Es decir, existe una función $f : \Sigma^* \to \Sigma^*$ computable en tiempo polinomial tal que $(\forall w \in \Sigma^*. \quad w \in A \Leftrightarrow f(w) \in B)$.

• Supongamos que $B \in P$.
Es decir, existe una M.T. M de tiempo polinomial que decide B.

• Construyamos una M.T. M' que decide A en tiempo polinomial:

$\underline{M'(w)}$: 1) Calcular $f(w)$.
2) Ejecutar $M(f(w))$.

• $M'(w) = \text{acepta} \Leftrightarrow M(f(w)) = \text{acepta}$
$\Leftrightarrow f(w) \in B$
$\Leftrightarrow w \in A$

• Análogamente, $M'(w) = \text{rechaza} \Leftrightarrow w \notin A$.
• Por lo tanto M' decide A.

• Si w es una palabra de largo n, $|w| = n$.
Calcular $f(w)$ cuesta a lo sumo n^k, para algún k, pues f es computable en tiempo polinomial.

• ¿Cuánto cuesta calcular $M(f(w))$?
- M tiene costo polinomial en el tamaño de la entrada.
- $\underline{\text{ojo}}$: la entrada no es w sino $f(w)$. $|f(w)| \leq n^k$.

• Por lo tanto, el costo de calcular $M(f(w))$ es $(n^k)^r = n^{k \cdot r}$.

Def.

- Un literal es una variable booleana afirmada o negada.

Ej: $x \quad y \quad z \quad \overline{x} \quad \overline{y} \quad \overline{z}$

$\neg x \quad \neg y \quad \neg z$

- Una cláusula es una disyunción de literales.

Ej: $x \vee \overline{y} \vee z$

- Una fórmula está en forma normal conjuntiva (FNC) si es una conjunción de cláusulas.

Ej. $(x \vee \overline{y} \vee z) \wedge (z \vee \overline{x}) \wedge (y \vee \overline{z})$ es una fórmula en FNC.

$(x \vee \overline{y}) \wedge \neg(z \vee x)$ No está en FNC.

- Una fórmula está en 3FNC si está en FNC y además todas las cláusulas tienen exactamente 3 literales.

Ej. $(x \vee y \vee y) \wedge (\overline{x} \vee y \vee y) \wedge (\overline{x} \vee z \vee \overline{z})$

está en 3FNC.

Def.

$3SAT = \{ \langle \varphi \rangle \mid \varphi$ es una fórmula en 3FNC satisfactible $\}$.

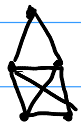
Obs. $3SAT \subseteq SAT$

Recordemos:

$$\text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ es un grafo no dirigido que tiene una } k\text{-clique} \}$$

Una clique de tamaño k.

subgrafo completo.

Ej. $G =$ 

tiene una 4-CLIQUE

pero no tiene una 5-CLIQUE.

Por lo tanto $\langle G, 4 \rangle \in \text{CLIQUE}$

pero $\langle G, 5 \rangle \notin \text{CLIQUE}$.

Teorema. $3SAT \leq_p \text{CLIQUE}$

Dem.. Por definición, queremos hallar una función $f : \Sigma^* \longrightarrow \Sigma^*$ computable en tiempo polinomial tal que

$$\langle \varphi \rangle \in SAT \iff \underbrace{f(\langle \varphi \rangle)}_{= \langle G, k \rangle} \in \text{CLIQUE}.$$

- Nos van a interesar las fórmulas en 3FNC.

Si φ no está en 3FNC, definimos $f(\varphi) = \langle :, 2 \rangle$

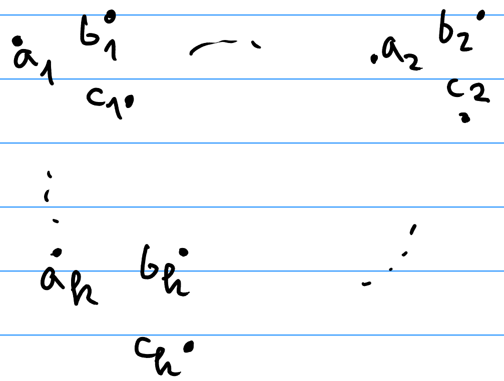
$\langle \varphi \rangle \notin SAT$ y $f(\varphi) \notin \text{CLIQUE}$.

- Si tenemos una fórmula φ que está en 3FNC,

$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$$

- La fórmula es la conjunción de k cláusulas.
- Vamos a construir un grafo de $3k$ vértices, y nos va a interesar la pregunta de si tiene o no una k-clique.

- El grafo tendrá un vértice por cada literal. ($3k$ vértices en total).

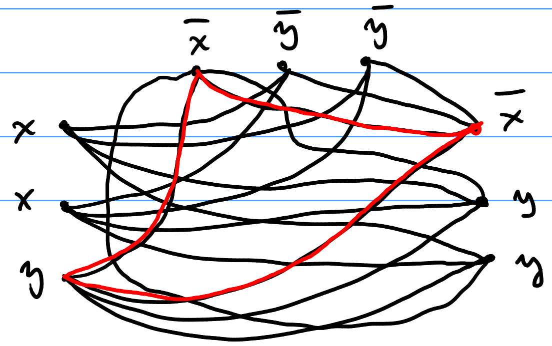


- Todos los pares de vértices del grafo van a estar conectados por una arista, excepto en dos casos:
  1) Si los dos vértices pertenecen a la misma terna (vienen de la misma cláusula).
  2) Si los dos vértices corresponden a literales opuestos (afirmación y negación de una misma variable).

Ejemplo: $\varphi = (x \vee x \vee y) \wedge (\bar{x} \vee \bar{y} \vee \bar{y}) \wedge (\bar{x} \vee y \vee y)$ en 3FNC

$k = 3$

$f(\varphi) = \langle G, 3 \rangle$

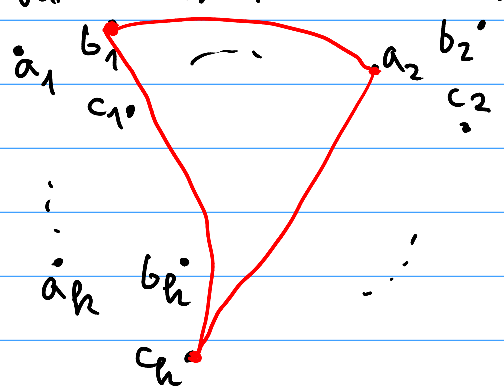


· Veamos que $\langle \varphi \rangle \in 3SAT$
si y sólo si $f(\varphi) \in CLIQUE$.

($\Longrightarrow$) · Supongamos que φ es satisfactible.
· Consideremos una asignación de variables a valores booleanos que hace verdadera a la fórmula.
· En cada cláusula hay al menos un literal que tiene valor verdadero.

$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \dots \wedge (a_k \vee b_k \vee c_k)$$

Estos k vértices van a estar conectados entre sí,



porque pertenecen a ternas distintas y no corresponden a literales opuestos.

Por lo tanto el grafo tiene una k-clique.

($\Longleftarrow$) Supongamos que el grafo tiene una k-clique.
· Los vértices de la k-clique corresponden a ternas distintas.

· Podemos elegir una asignación de variables que le otorgue valor verdadero a los literales de la clique.

· No puede haber una contradicción, no puede ser que x y $\overline{x}$ estén ambos en la clique.

· A las variables que no aparecen en la clique, se les puede dar cualquier valor.
· Esta asignación de variables hace verdadera a φ. $\langle \varphi \rangle \in 3SAT$.

Conclusión:

$$3SAT \leq_p CLIQUE.$$

Si tuviéramos un algoritmo para resolver CLIQUE en tiempo polinomial, podríamos resolver 3SAT en tiempo polinomial.

Def (NP-completitud).

Sea $A \subseteq \Sigma^*$ un lenguaje.
Decimos que A es NP-completo si:

1) $A \in NP$.
2) $\forall B \subseteq \Sigma^*.\ B \in NP \Rightarrow B \leqslant_p A$.] — A es NP-hard.

Teorema. Si A es NP-completo y $A \in P$ entonces $P = NP$.

Dem. Sup. que A es NP-completo y $A \in P$.
Veamos que $P = NP$.

($\subseteq$) Ya habíamos visto que $P \subseteq NP$.

($\supseteq$) Sea $B \in NP$. Como A es NP-completo, $B \leqslant_p A$.
Por lo tanto $B \in P$. Por lo tanto $NP \subseteq P$.

Teorema. Si A es NP-completo, $B \in NP$ y $A \leqslant_p B$ entonces B es NP-completo.

Dem. Sup. que A es NP-completo, $B \in NP$ y $A \leqslant_p B$.
Veamos que B es NP-completo.

1) $B \in NP$ ✓

2) Veamos que $(\forall C \in NP.\ C \leqslant_p B)$.
Sea $C \in NP$.
Como A es NP-completo,
$C \leqslant_p A$.

$$C \leqslant_p A \leqslant_p B$$

Por lo tanto $C \leqslant_p B$. ✓

Enunciado alternativo:

Si $SAT \in P$ entonces $P = NP$.

Teorema (Cook-Levin).

SAT es NP-completo.

$SAT = \{\langle\varphi\rangle \mid \varphi \text{ es satisfactible}\}$.

Dem.

Veamos que SAT cumple con la definición de NP-Completo.

1) SAT $\in$ NP.

Veamos que SAT se puede verificar en tiempo polinomial.

$V(\langle\varphi\rangle, c)$:

- El certificado debe ser una lista de variables $[x_1, x_2, \ldots, x_n]$. (Si no, rechazar).
- Evaluar la fórmula φ dándole valor verdadero a las variables que estén en la lista y valor falso a las demás variables.
- Si la fórmula tiene valor verdadero, aceptar. Si no, rechazar.

$$\langle\varphi\rangle \in SAT \iff \exists c.\ V(\langle\varphi\rangle, c) = \text{acepta}$$

SAT es NP-hard

2) Veamos que

si $A \in NP$ entonces $A \leq_p SAT$.

. Supongamos que $A \in NP$.

Es decir, existe una M.T.N. N tal que N decide A en tiempo polinomial.

- Supongamos que dada una cadena $w \in \Sigma^*$, $|w| = n$ la máquina N termina su ejecución en tiempo $|w|^k = n^k$.

- Una cadena $w \in \Sigma^*$ es aceptada por la máquina N si existe una secuencia de configuraciones:

$$\left.\begin{array}{l} \overbrace{\alpha_0 \qquad\qquad\qquad}^{n^k} \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{array}\right\} m \leq n^k$$

de tal modo que:
1) α_0 es la configuración inicial, $\alpha_0 = q_0 w$.
2) α_m está en un estado final.
3) Hay una transición de α_i a α_{i+1} $\forall i \in 0..(m-1)$.

. Vamos a querer transformar una cadena $w \in \Sigma^*$ en una fórmula φ que sea satisfactible si y sólo si existe una secuencia de configuraciones de la máquina N que acepta la cadena w.

Una tabla de ejecución es una matriz de $n^k \times n^k$

En cada fila hay una configuración de la máquina:

n^k columnas

n^k filas

#	q_0	a	b	c	a	a	⊔	⊔	...	⊔	⊔	#
#	z	q_3	b	c	a	a	⊔	⊔	...			#
#												#

- Vamos a tener variables de la forma x_{ijs} donde

  $1 \leq i \leq n^k$ es un número de fila

  $1 \leq j \leq n^k$ es un número de columna

  s es un símbolo $\underbrace{s \in \Gamma \cup Q \cup \{\#\}}_{C}$

- La variable x_{ijs} representa el hecho de que en la fila i, columna j de la tabla se encuentra el símbolo s.

- Vamos a construir una fórmula:

$$\varphi = \varphi_{cell} \land \varphi_{start} \land \varphi_{accept} \land \varphi_{move}$$

$$\underline{\varphi_{cell}} = \bigwedge_{i=1..n^k} \bigwedge_{j=1..n^k} \left(\bigvee_{s \in C} x_{ijs} \right) \land \left(\bigwedge_{s \in C} \bigwedge_{\substack{t \in C \\ t \neq s}} \overline{x_{ijs}} \lor \overline{x_{ijt}} \right)$$

"La tabla está correctamente armada".

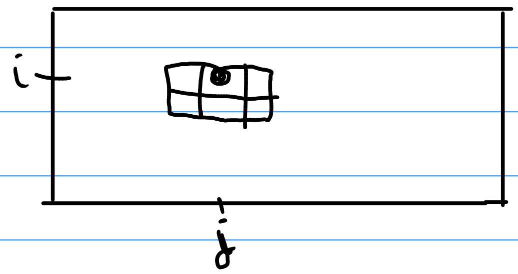
$$\underline{\varphi_{start}} = x_{11\#} \land x_{12q_0} \land x_{13w_1} \land x_{14w_2} \land \dots \land x_{1(n+2)w_n}$$
$$\land\, x_{1(n+3)⊔}$$
$$\vdots$$
$$\land\, x_{1(n^k-1)⊔}$$
$$\land\, x_{1n^k\#}$$

"La primera fila de la tabla es la configuración inicial de N cuando se la alimenta con w".

$$\varphi_{accept} = \bigvee_{i=1..n^k} \bigvee_{j=1..n^k} x_{ij\,q_{accept}}$$

"La máquina acepta la cadena de entrada".

$$\varphi_{move} = \bigwedge_{i=1..n^k} \bigwedge_{j=1..n^k}$$ (La ventana de 2 filas y 3 columnas centrada en (i,j) corresponde a una transición posible de la máquina N).



"Las sucesivas filas de la tabla corresponden a transiciones de la máquina N".

a	b	c
?	b	?

#	b	c
#	b	?

q_1	b	c
z	q_7	?

q_1	b	c
?	b	?

- Dada una palabra $w \in \Sigma^*$ hemos construido la codificación de una fórmula $f(w)$.
- Vale que $w \in A \iff f(w) \in SAT$ $\qquad A \leq_m SAT$
- Además, la función f es computable en tiempo polinomial. Y tenemos entonces

$$A \leq_p SAT.$$

. **Corolario.** 3SAT es NP-completo.

$$(x_1 \vee x_2 \vee x_3 \vee \dots \vee x_n)$$

$$\Big\Downarrow$$

3FNC

$$(x_1 \vee x_1 \vee a_1) \wedge (\overline{a_1} \vee x_2 \vee a_2) \wedge (\overline{a_2} \vee x_3 \vee a_3) \wedge \dots \wedge (\overline{a_{n-1}} \vee x_n \vee x_n)$$

. **Corolario.** CLIQUE es NP-completo.

Ya vimos que $\underbrace{3SAT}_{\text{NP-completo}} \leq_p CLIQUE \in NP$