


Análisis y Diseño de Algoritmos

# NP-completitud

## Cook-Levin, 3-SAT, gadgets, suma de subconjuntos y la frontera

**Gerson Garrido**

Semana 14

UNIVERSIDAD NACIONAL DE INGENIERÍA

# Hoy

**Sesión A**

1. NP-difícil y NP-completo
2. Cook-Levin y la receta
3. De SAT a 3-SAT
4. De 3-SAT a conjunto independiente

**Sesión B**

5. De 3-SAT a 3-coloreo
6. De cobertura de vértices a suma de subconjuntos
7. La mochila, y $O(n^5)$
8. Un mapa de problemas y la frontera

> **Lectura**
>
> Apuntes de la semana 14. Código en `code/week-14`.

SECCIÓN 01

# NP-difícil y NP-completo

01



# El hueco de la semana pasada

Ya tenemos: P, NP y $A \leq_p B$, con reducciones que se componen.

# El hueco de la semana pasada

Ya tenemos: P, NP y $A \leq_p B$, con reducciones que se componen.

También: conjunto independiente, cobertura de vértices y clique se transforman uno en otro.

# El hueco de la semana pasada

Ya tenemos: P, NP y $A \leq_p B$, con reducciones que se componen.

También: conjunto independiente, cobertura de vértices y clique se transforman uno en otro.

**Pero**

Una reducción solo traslada la dificultad. Sin un primer problema difícil, solo sabemos que “son igual de difíciles entre sí”.

# El hueco de la semana pasada

Ya tenemos: P, NP y $A \leq_p B$, con reducciones que se componen.

También: conjunto independiente, cobertura de vértices y clique se transforman uno en otro.

**Pero**

Una reducción solo traslada la dificultad. Sin un primer problema difícil, solo sabemos que “son igual de difíciles entre sí”.

Hoy: el primer problema, y una cadena de construcciones que sale de él.

# Dos definiciones

**NP-difícil** *(NP-hard)*

$X$ es NP-difícil si todo problema $A$ de NP cumple $A \leq_p X$.

# Dos definiciones

**NP-difícil** *(NP-hard)*

$X$ es NP-difícil si todo problema $A$ de NP cumple $A \leq_p X$.

**NP-completo** *(NP-complete)*

$X$ es NP-completo si está en NP y es NP-difícil.

# Dos definiciones

**NP-difícil** *(NP-hard)*

$X$ es NP-difícil si todo problema $A$ de NP cumple $A \leq_p X$.

**NP-completo** *(NP-complete)*

$X$ es NP-completo si está en NP y es NP-difícil.

**Teorema.** Si $X$ es NP-completo: $X$ está en P $\iff$ P = NP.

# Dos definiciones

**NP-difícil** *(NP-hard)*

$X$ es NP-difícil si todo problema $A$ de NP cumple $A \leq_p X$.

**NP-completo** *(NP-complete)*

$X$ es NP-completo si está en NP y es NP-difícil.

**Teorema.** Si $X$ es NP-completo: $X$ está en P $\iff$ P = NP.

- $\Rightarrow$: cualquier $A$ de NP se transforma a $X$ y se resuelve. Un polinomio de un polinomio es un polinomio. Luego NP $\subseteq$ P.

# Dos definiciones

**NP-difícil** *(NP-hard)*

$X$ es NP-difícil si todo problema $A$ de NP cumple $A \leq_p X$.

**NP-completo** *(NP-complete)*

$X$ es NP-completo si está en NP y es NP-difícil.

**Teorema.** Si $X$ es NP-completo: $X$ está en P $\iff$ P = NP.

- $\Rightarrow$: cualquier $A$ de NP se transforma a $X$ y se resuelve. Un polinomio de un polinomio es un polinomio. Luego NP $\subseteq$ P.
- $\Leftarrow$: $X$ está en NP, que sería igual a P.

# Dos definiciones

**NP-difícil** *(NP-hard)*

$X$ es NP-difícil si todo problema $A$ de NP cumple $A \leq_p X$.

**NP-completo** *(NP-complete)*

$X$ es NP-completo si está en NP y es NP-difícil.

**Teorema.** Si $X$ es NP-completo: $X$ está en P $\iff$ P = NP.

- $\Rightarrow$: cualquier $A$ de NP se transforma a $X$ y se resuelve. Un polinomio de un polinomio es un polinomio. Luego NP $\subseteq$ P.
- $\Leftarrow$: $X$ está en NP, que sería igual a P.

Si un NP-completo está en P, todos lo están. Sin resolver: la mayoría *cree* que P $\neq$ NP.

# NP-difícil no es lo mismo que NP-completo

El **problema de la parada** *(halting problem)*: ¿este programa termina con esta entrada?

# NP-difícil no es lo mismo que NP-completo

El **problema de la parada** *(halting problem)*: ¿este programa termina con esta entrada?

- **Es NP-difícil.** Dada $\varphi$, se imprime un programa que prueba todas las asignaciones: termina si alguna cumple, y si no, entra en un ciclo infinito. No se corre, solo se escribe. Y todo NP se reduce a SAT.

# NP-difícil no es lo mismo que NP-completo

El **problema de la parada** *(halting problem)*: ¿este programa termina con esta entrada?

- **Es NP-difícil.** Dada $\varphi$, se imprime un programa que prueba todas las asignaciones: termina si alguna cumple, y si no, entra en un ciclo infinito. No se corre, solo se escribe. Y todo NP se reduce a SAT.
- **No está en NP.** Todo problema de NP se decide probando certificados. Este es **indecidible**, como verán en Teoría de la Computación.

# NP-difícil no es lo mismo que NP-completo

El **problema de la parada** *(halting problem)*: ¿este programa termina con esta entrada?

- **Es NP-difícil.** Dada $\varphi$, se imprime un programa que prueba todas las asignaciones: termina si alguna cumple, y si no, entra en un ciclo infinito. No se corre, solo se escribe. Y todo NP se reduce a SAT.
- **No está en NP.** Todo problema de NP se decide probando certificados. Este es **indecidible**, como verán en Teoría de la Computación.

**Optimización**

"¿Cuál es el mayor conjunto independiente?" no es de decisión, así que no entra en la definición de $\leq_p$. Se le dice NP-difícil en un sentido más amplio: un algoritmo polinomial para él daría P = NP.

SECCIÓN 02

# Cook-Levin y la receta

02

# Cook-Levin

**Teorema de Cook-Levin, enunciado**

SAT es NP-completo, incluso restringido a fórmulas en CNF.

# Cook-Levin

**Teorema de Cook-Levin, enunciado**
SAT es NP-completo, incluso restringido a fórmulas en CNF.

En este capítulo, **SAT significa SAT en CNF**.

# Cook-Levin

**Teorema de Cook-Levin, enunciado**
SAT es NP-completo, incluso restringido a fórmulas en CNF.

En este capítulo, **SAT significa SAT en CNF**.

**Intuición.** Si $A$ está en NP, tiene un verificador polinomial.

# Cook-Levin

**Teorema de Cook-Levin, enunciado**
SAT es NP-completo, incluso restringido a fórmulas en CNF.

En este capítulo, **SAT significa SAT en CNF**.

**Intuición.** Si $A$ está en NP, tiene un verificador polinomial.

- Su ejecución es una tabla de bits, con una fila por paso.

# Cook-Levin

**Teorema de Cook-Levin, enunciado**
SAT es NP-completo, incluso restringido a fórmulas en CNF.

En este capítulo, **SAT significa SAT en CNF**.

**Intuición.** Si $A$ está en NP, tiene un verificador polinomial.

- Su ejecución es una tabla de bits, con una fila por paso.
- "Cada fila sale bien de la anterior" es una regla local, y se escribe con cláusulas.

# Cook-Levin

**Teorema de Cook-Levin, enunciado**
SAT es NP-completo, incluso restringido a fórmulas en CNF.

En este capítulo, **SAT significa SAT en CNF**.

**Intuición.** Si *A* está en NP, tiene un verificador polinomial.

- Su ejecución es una tabla de bits, con una fila por paso.
- "Cada fila sale bien de la anterior" es una regla local, y se escribe con cláusulas.
- La fórmula se cumple si y solo si hay un certificado que el verificador acepta.

# Cook-Levin

> **Teorema de Cook-Levin, enunciado**
> SAT es NP-completo, incluso restringido a fórmulas en CNF.

En este capítulo, **SAT significa SAT en CNF**.

**Intuición.** Si *A* está en NP, tiene un verificador polinomial.

- Su ejecución es una tabla de bits, con una fila por paso.
- "Cada fila sale bien de la anterior" es una regla local, y se escribe con cláusulas.
- La fórmula se cumple si y solo si hay un certificado que el verificador acepta.

La demostración va en Teoría de la Computación.

# La receta

**Teorema.** Si $X$ está en NP, $A$ es NP-completo y $A \leq_p X$, entonces $X$ es NP-completo.
Porque todo $Y$ de NP cumple $Y \leq_p A \leq_p X$.

# La receta

**Teorema.** Si $X$ está en NP, $A$ es NP-completo y $A \leq_p X$, entonces $X$ es NP-completo.
Porque todo $Y$ de NP cumple $Y \leq_p A \leq_p X$.

1. **Está en NP:** certificado de tamaño polinomial, verificable en tiempo polinomial.

# La receta

**Teorema.** Si $X$ está en NP, $A$ es NP-completo y $A \leq_p X$, entonces $X$ es NP-completo.
Porque todo $Y$ de NP cumple $Y \leq_p A \leq_p X$.

1. **Está en NP:** certificado de tamaño polinomial, verificable en tiempo polinomial.
2. **Elegir** un NP-completo conocido $A$.

# La receta

**Teorema.** Si $X$ está en NP, $A$ es NP-completo y $A \leq_p X$, entonces $X$ es NP-completo.
Porque todo $Y$ de NP cumple $Y \leq_p A \leq_p X$.

1. **Está en NP:** certificado de tamaño polinomial, verificable en tiempo polinomial.
2. **Elegir** un NP-completo conocido $A$.
3. **Transformar** de $A$ hacia $X$: del conocido al nuevo.

# La receta

**Teorema.** Si $X$ está en NP, $A$ es NP-completo y $A \leq_p X$, entonces $X$ es NP-completo.
Porque todo $Y$ de NP cumple $Y \leq_p A \leq_p X$.

1. **Está en NP:** certificado de tamaño polinomial, verificable en tiempo polinomial.
2. **Elegir** un NP-completo conocido $A$.
3. **Transformar** de $A$ hacia $X$: del conocido al nuevo.
4. **Dos direcciones:** sí en $A \Rightarrow$ sí en $X$, y sí en $X \Rightarrow$ sí en $A$.

# La receta

**Teorema.** Si $X$ está en NP, $A$ es NP-completo y $A \leq_p X$, entonces $X$ es NP-completo.
Porque todo $Y$ de NP cumple $Y \leq_p A \leq_p X$.

1. **Está en NP:** certificado de tamaño polinomial, verificable en tiempo polinomial.
2. **Elegir** un NP-completo conocido $A$.
3. **Transformar** de $A$ hacia $X$: del conocido al nuevo.
4. **Dos direcciones:** sí en $A \Rightarrow$ sí en $X$, y sí en $X \Rightarrow$ sí en $A$.
5. **Tiempo polinomial** de la transformación, en bits.

# La receta

**Teorema.** Si $X$ está en NP, $A$ es NP-completo y $A \leq_p X$, entonces $X$ es NP-completo.
Porque todo $Y$ de NP cumple $Y \leq_p A \leq_p X$.

1. **Está en NP:** certificado de tamaño polinomial, verificable en tiempo polinomial.
2. **Elegir** un NP-completo conocido $A$.
3. **Transformar** de $A$ hacia $X$: del conocido al nuevo.
4. **Dos direcciones:** sí en $A \Rightarrow$ sí en $X$, y sí en $X \Rightarrow$ sí en $A$.
5. **Tiempo polinomial** de la transformación, en bits.

**Error común**

Olvidar el paso 1. El problema de la parada cumple del 2 al 5, y no es NP-completo.

SECCIÓN 03

# De SAT a 3-SAT

03



# 3-SAT

**3-SAT, en este curso**

Fórmula en CNF con **exactamente tres literales por cláusula, de tres variables distintas**.

| $k$ | se reemplaza por |
|---|---|

# 3-SAT

**3-SAT, en este curso**

Fórmula en CNF con **exactamente tres literales por cláusula, de tres variables distintas**.

“A lo sumo tres” también es NP-completo: está en NP y contiene a este caso.

| $k$ | se reemplaza por |
|---|---|

# 3-SAT

**3-SAT, en este curso**

Fórmula en CNF con **exactamente tres literales por cláusula, de tres variables distintas**.

"A lo sumo tres" también es NP-completo: está en NP y contiene a este caso.
**La transformación, por cláusula.** Primero se limpia: fuera los repetidos, y se borra toda cláusula con $x$ y $\neg x$.

| $k$ | se reemplaza por |
|---|---|

# 3-SAT

**3-SAT, en este curso**

Fórmula en CNF con **exactamente tres literales por cláusula, de tres variables distintas**.

"A lo sumo tres" también es NP-completo: está en NP y contiene a este caso.
**La transformación, por cláusula.** Primero se limpia: fuera los repetidos, y se borra toda cláusula con $x$ y $\neg x$.

| $k$ | se reemplaza por |
|---|---|
| 0 | las 8 cláusulas sobre tres variables nuevas |

# 3-SAT

**3-SAT, en este curso**

Fórmula en CNF con **exactamente tres literales por cláusula, de tres variables distintas**.

"A lo sumo tres" también es NP-completo: está en NP y contiene a este caso.
**La transformación, por cláusula.** Primero se limpia: fuera los repetidos, y se borra toda cláusula con $x$ y $\neg x$.

| $k$ | se reemplaza por |
|---|---|
| 0 | las 8 cláusulas sobre tres variables nuevas |
| 1 | $(\ell \vee y_1 \vee y_2)(\ell \vee y_1 \vee \neg y_2)(\ell \vee \neg y_1 \vee y_2)(\ell \vee \neg y_1 \vee \neg y_2)$ |

# 3-SAT

**3-SAT, en este curso**

Fórmula en CNF con **exactamente tres literales por cláusula, de tres variables distintas**.

"A lo sumo tres" también es NP-completo: está en NP y contiene a este caso.
**La transformación, por cláusula.** Primero se limpia: fuera los repetidos, y se borra toda cláusula con $x$ y $\neg x$.

| $k$ | se reemplaza por |
|---|---|
| 0 | las 8 cláusulas sobre tres variables nuevas |
| 1 | $(\ell \vee y_1 \vee y_2)(\ell \vee y_1 \vee \neg y_2)(\ell \vee \neg y_1 \vee y_2)(\ell \vee \neg y_1 \vee \neg y_2)$ |
| 2 | $(\ell_1 \vee \ell_2 \vee y_1)(\ell_1 \vee \ell_2 \vee \neg y_1)$ |

# 3-SAT

**3-SAT, en este curso**

Fórmula en CNF con **exactamente tres literales por cláusula, de tres variables distintas**.

"A lo sumo tres" también es NP-completo: está en NP y contiene a este caso.
**La transformación, por cláusula.** Primero se limpia: fuera los repetidos, y se borra toda cláusula con $x$ y $\neg x$.

| $k$ | se reemplaza por |
|---|---|
| 0 | las 8 cláusulas sobre tres variables nuevas |
| 1 | $(\ell \vee y_1 \vee y_2)(\ell \vee y_1 \vee \neg y_2)(\ell \vee \neg y_1 \vee y_2)(\ell \vee \neg y_1 \vee \neg y_2)$ |
| 2 | $(\ell_1 \vee \ell_2 \vee y_1)(\ell_1 \vee \ell_2 \vee \neg y_1)$ |
| 3 | la misma |

# 3-SAT

**3-SAT, en este curso**

Fórmula en CNF con **exactamente tres literales por cláusula, de tres variables distintas**.

"A lo sumo tres" también es NP-completo: está en NP y contiene a este caso.
**La transformación, por cláusula.** Primero se limpia: fuera los repetidos, y se borra toda cláusula con $x$ y $\neg x$.

| $k$ | se reemplaza por |
|---|---|
| 0 | las 8 cláusulas sobre tres variables nuevas |
| 1 | $(\ell \vee y_1 \vee y_2)(\ell \vee y_1 \vee \neg y_2)(\ell \vee \neg y_1 \vee y_2)(\ell \vee \neg y_1 \vee \neg y_2)$ |
| 2 | $(\ell_1 \vee \ell_2 \vee y_1)(\ell_1 \vee \ell_2 \vee \neg y_1)$ |
| 3 | la misma |
| $\geq 4$ | una cadena |

# La cadena

$(\ell_1 \vee \ell_2 \vee \ell_3 \vee \cdots \vee \ell_k)$ se reemplaza por $k - 2$ cláusulas, con $k - 3$ variables nuevas:

$$
\begin{aligned}
&(\ell_1 \vee \ell_2 \vee y_1) \\
&(\neg y_1 \vee \ell_3 \vee y_2) \\
&(\neg y_2 \vee \ell_4 \vee y_3) \\
&\quad \vdots \\
&(\neg y_{k-3} \vee \ell_{k-1} \vee \ell_k)
\end{aligned}
$$

# La cadena

$(\ell_1 \vee \ell_2 \vee \ell_3 \vee \cdots \vee \ell_k)$ se reemplaza por $k - 2$ cláusulas, con $k - 3$ variables nuevas:

$$
\begin{aligned}
&(\ell_1 \vee \ell_2 \vee y_1) \\
&(\neg y_1 \vee \ell_3 \vee y_2) \\
&(\neg y_2 \vee \ell_4 \vee y_3) \\
&\quad \vdots \\
&(\neg y_{k-3} \vee \ell_{k-1} \vee \ell_k)
\end{aligned}
$$

**Qué dice $y_j$**

"El literal verdadero está más adelante." Cada eslabón: si me dijeron eso, es este literal, o sigue más adelante.

# La cadena

$(\ell_1 \vee \ell_2 \vee \ell_3 \vee \cdots \vee \ell_k)$ se reemplaza por $k - 2$ cláusulas, con $k - 3$ variables nuevas:

$$
\begin{aligned}
&(\ell_1 \vee \ell_2 \vee y_1)\\
&(\neg y_1 \vee \ell_3 \vee y_2)\\
&(\neg y_2 \vee \ell_4 \vee y_3)\\
&\quad\vdots\\
&(\neg y_{k-3} \vee \ell_{k-1} \vee \ell_k)
\end{aligned}
$$

**Qué dice $y_j$**

"El literal verdadero está más adelante." Cada eslabón: si me dijeron eso, es este literal, o sigue más adelante.

La salida no es equivalente, porque tiene otras variables. Es **equisatisfactible**: una se puede cumplir si y solo si la otra se puede cumplir.

# Corre

build/three_sat_demo, extracto

```
entrada: (x1 ∨ x2 ∨ x3 ∨ x4 ∨ x5 ∨ x6)
salida, con 9 variables y 4 cláusulas:
  (x1 ∨ x2 ∨ x7)
  (¬x7 ∨ x3 ∨ x8)
  (¬x8 ∨ x4 ∨ x9)
  (¬x9 ∨ x5 ∨ x6)
  fuerza bruta: entrada satisfacible, salida satisfacible

entrada: (x1 ∨ x1 ∨ ¬x2 ∨ x1)
salida, con 3 variables y 2 cláusulas:
  (x1 ∨ ¬x2 ∨ x3)
  (x1 ∨ ¬x2 ∨ ¬x3)
  fuerza bruta: entrada satisfacible, salida satisfacible
```

Una cláusula con $x$ y $\neg x$ desaparece: siempre se cumple.

# De $C$ a la cadena

Sea $\ell_i$ un literal verdadero de $C$. Damos $y_j = V$ si $i \geq j + 2$, y $F$ si no.

# De $C$ a la cadena

Sea $\ell_i$ un literal verdadero de $C$. Damos $y_j = V$ si $i \geq j + 2$, y $F$ si no.

- $(\ell_1 \vee \ell_2 \vee y_1)$: si $i \leq 2$, por $\ell_i$. Si $i \geq 3$, $y_1 = V$.

# De $C$ a la cadena

Sea $\ell_i$ un literal verdadero de $C$. Damos $y_j = V$ si $i \geq j + 2$, y $F$ si no.

- $(\ell_1 \vee \ell_2 \vee y_1)$: si $i \leq 2$, por $\ell_i$. Si $i \geq 3$, $y_1 = V$.
- $(\neg y_{j-1} \vee \ell_{j+1} \vee y_j)$: si $i \leq j$, por $\neg y_{j-1}$. Si $i = j + 1$, por $\ell_{j+1}$. Si $i \geq j + 2$, por $y_j$.

# De $C$ a la cadena

Sea $\ell_i$ un literal verdadero de $C$. Damos $y_j = V$ si $i \geq j + 2$, y $F$ si no.

- $(\ell_1 \vee \ell_2 \vee y_1)$: si $i \leq 2$, por $\ell_i$. Si $i \geq 3$, $y_1 = V$.
- $(\neg y_{j-1} \vee \ell_{j+1} \vee y_j)$: si $i \leq j$, por $\neg y_{j-1}$. Si $i = j + 1$, por $\ell_{j+1}$. Si $i \geq j + 2$, por $y_j$.
- $(\neg y_{k-3} \vee \ell_{k-1} \vee \ell_k)$: si $i \leq k - 2$, por $\neg y_{k-3}$. Si no, por $\ell_i$.

# De $C$ a la cadena

Sea $\ell_i$ un literal verdadero de $C$. Damos $y_j = V$ si $i \geq j + 2$, y $F$ si no.

- $(\ell_1 \vee \ell_2 \vee y_1)$: si $i \leq 2$, por $\ell_i$. Si $i \geq 3$, $y_1 = V$.
- $(\neg y_{j-1} \vee \ell_{j+1} \vee y_j)$: si $i \leq j$, por $\neg y_{j-1}$. Si $i = j + 1$, por $\ell_{j+1}$. Si $i \geq j + 2$, por $y_j$.
- $(\neg y_{k-3} \vee \ell_{k-1} \vee \ell_k)$: si $i \leq k - 2$, por $\neg y_{k-3}$. Si no, por $\ell_i$.

| con $i = 4$ | regla | valor | eslabón |
|---|---|---|---|
| $y_1 = x_7$ | $4 \geq 3$ | $V$ | $(x_1 \vee x_2 \vee x_7)$ por $x_7$ |
| $y_2 = x_8$ | $4 \geq 4$ | $V$ | $(\neg x_7 \vee x_3 \vee x_8)$ por $x_8$ |
| $y_3 = x_9$ | $4 \geq 5$, no | $F$ | $(\neg x_8 \vee x_4 \vee x_9)$ por $x_4$ |
| | | | $(\neg x_9 \vee x_5 \vee x_6)$ por $\neg x_9$ |

# La cadena en C++

three_sat.h

```
// Una cadena: y dice "el literal verdadero está más adelante".
int y = ++out.variables;
out.clauses.push_back({c[0], c[1], y});
for (std::size_t i = 2; i + 2 < k; i++) {
    int next = ++out.variables;
    out.clauses.push_back({-y, c[i], next});
    y = next;
}
out.clauses.push_back({-y, c[k - 2], c[k - 1]});
```

Cada variable nueva se pide con `++out.variables`: ninguna se comparte con otra cláusula. Los literales son enteros, y $-y$ es la negación de $y$.

# De la cadena a $C$, y el costo

Supongamos que se cumple toda la cadena y que **todos** los $\ell_i$ son falsos.

# De la cadena a $C$, y el costo

Supongamos que se cumple toda la cadena y que **todos** los $\ell_i$ son falsos.

1. El primer eslabón obliga a $y_1 = V$.

# De la cadena a $C$, y el costo

Supongamos que se cumple toda la cadena y que **todos** los $\ell_i$ son falsos.

1. El primer eslabón obliga a $y_1 = V$.
2. Si $y_{j-1} = V$, el siguiente eslabón tiene $\neg y_{j-1}$ y $\ell_{j+1}$ falsos, y obliga a $y_j = V$.

# De la cadena a $C$, y el costo

Supongamos que se cumple toda la cadena y que **todos** los $\ell_i$ son falsos.

1. El primer eslabón obliga a $y_1 = V$.
2. Si $y_{j-1} = V$, el siguiente eslabón tiene $\neg y_{j-1}$ y $\ell_{j+1}$ falsos, y obliga a $y_j = V$.
3. Por inducción, $y_{k-3} = V$.

# De la cadena a $C$, y el costo

Supongamos que se cumple toda la cadena y que **todos** los $\ell_i$ son falsos.

1. El primer eslabón obliga a $y_1 = V$.
2. Si $y_{j-1} = V$, el siguiente eslabón tiene $\neg y_{j-1}$ y $\ell_{j+1}$ falsos, y obliga a $y_j = V$.
3. Por inducción, $y_{k-3} = V$.
4. El último eslabón queda con sus tres literales falsos. Contradicción. ■

# De la cadena a $C$, y el costo

Supongamos que se cumple toda la cadena y que **todos** los $\ell_i$ son falsos.

1. El primer eslabón obliga a $y_1 = V$.
2. Si $y_{j-1} = V$, el siguiente eslabón tiene $\neg y_{j-1}$ y $\ell_{j+1}$ falsos, y obliga a $y_j = V$.
3. Por inducción, $y_{k-3} = V$.
4. El último eslabón queda con sus tres literales falsos. Contradicción. ■

**Por cláusula** alcanza, porque las variables nuevas de cada una son solo suyas. Los largos 0, 1 y 2 se revisan directamente.

# De la cadena a $C$, y el costo

Supongamos que se cumple toda la cadena y que **todos** los $\ell_i$ son falsos.

1. El primer eslabón obliga a $y_1 = V$.
2. Si $y_{j-1} = V$, el siguiente eslabón tiene $\neg y_{j-1}$ y $\ell_{j+1}$ falsos, y obliga a $y_j = V$.
3. Por inducción, $y_{k-3} = V$.
4. El último eslabón queda con sus tres literales falsos. Contradicción. ■

**Por cláusula** alcanza, porque las variables nuevas de cada una son solo suyas. Los largos 0, 1 y 2 se revisan directamente.
**Costo.** Una cláusula de $k$ literales da a lo sumo $\max(k, 8)$ cláusulas: lineal.

# De la cadena a $C$, y el costo

Supongamos que se cumple toda la cadena y que **todos** los $\ell_i$ son falsos.

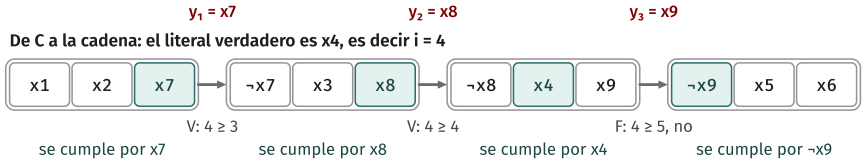
1. El primer eslabón obliga a $y_1 = V$.
2. Si $y_{j-1} = V$, el siguiente eslabón tiene $\neg y_{j-1}$ y $\ell_{j+1}$ falsos, y obliga a $y_j = V$.
3. Por inducción, $y_{k-3} = V$.
4. El último eslabón queda con sus tres literales falsos. Contradicción. ■

**Por cláusula** alcanza, porque las variables nuevas de cada una son solo suyas. Los largos 0, 1 y 2 se revisan directamente.
**Costo.** Una cláusula de $k$ literales da a lo sumo $\max(k, 8)$ cláusulas: lineal.

> **¿Y hasta dos literales?**
> Un eslabón necesita mensaje recibido, literal y mensaje enviado: tres cosas.
> 2-SAT está en P.

# Las dos direcciones, en el ejemplo

Entrada: (x1 ∨ x2 ∨ x3 ∨ x4 ∨ x5 ∨ x6). Con k = 6 literales: k − 3 = 3 variables nuevas y k − 2 = 4 cláusulas.




En verde, los literales verdaderos. Cada y es un mensaje: "el literal verdadero está más adelante".

Las variables nuevas solo aparecen en la cadena de esta cláusula.

SECCIÓN 04

# De 3-SAT a conjunto independiente

04

# La transformación

Una fórmula tiene dos clases de reglas, y el grafo, una clase de arista por cada una.

# La transformación

Una fórmula tiene dos clases de reglas, y el grafo, una clase de arista por cada una.

- **Un triángulo por cláusula,** con un nodo por literal: “a lo sumo uno por cláusula”.

# La transformación

Una fórmula tiene dos clases de reglas, y el grafo, una clase de arista por cada una.

- **Un triángulo por cláusula,** con un nodo por literal: “a lo sumo uno por cláusula”.
- **Una arista entre literales opuestos:** “no se contradigan”.

# La transformación

Una fórmula tiene dos clases de reglas, y el grafo, una clase de arista por cada una.

- **Un triángulo por cláusula,** con un nodo por literal: “a lo sumo uno por cláusula”.
- **Una arista entre literales opuestos:** “no se contradigan”.
- $k$ **= número de cláusulas:** “uno en cada cláusula”.

# La transformación

Una fórmula tiene dos clases de reglas, y el grafo, una clase de arista por cada una.

- **Un triángulo por cláusula,** con un nodo por literal: “a lo sumo uno por cláusula”.
- **Una arista entre literales opuestos:** “no se contradigan”.
- $k$ **= número de cláusulas:** “uno en cada cláusula”.

Elegir un nodo significa: este literal es verdadero, y por él se cumple esta cláusula.

# La transformación

Una fórmula tiene dos clases de reglas, y el grafo, una clase de arista por cada una.

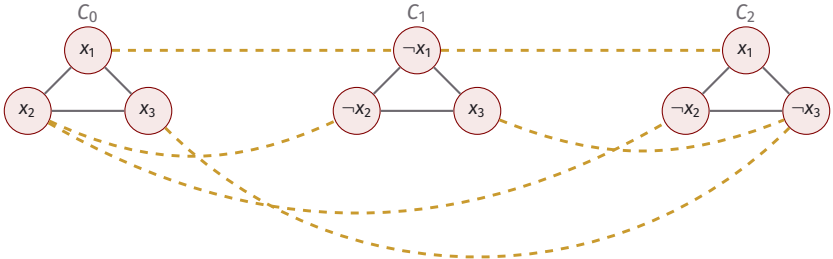
- **Un triángulo por cláusula,** con un nodo por literal: “a lo sumo uno por cláusula”.
- **Una arista entre literales opuestos:** “no se contradigan”.
- $k$ **= número de cláusulas:** “uno en cada cláusula”.

Elegir un nodo significa: este literal es verdadero, y por él se cumple esta cláusula.

> **Está en NP**
> El certificado es el conjunto: se revisan sus aristas y su tamaño.

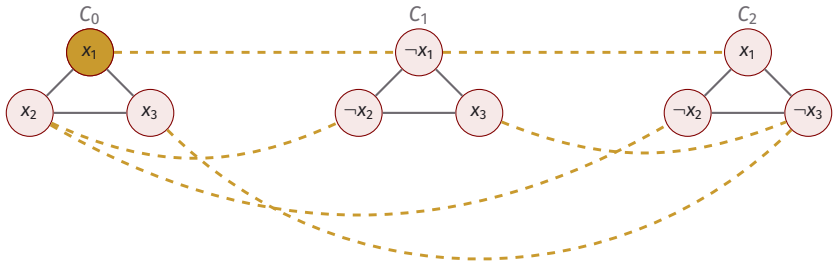
# Un ejemplo seguido en detalle

$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$, con $k = 3$.



# Un ejemplo seguido en detalle

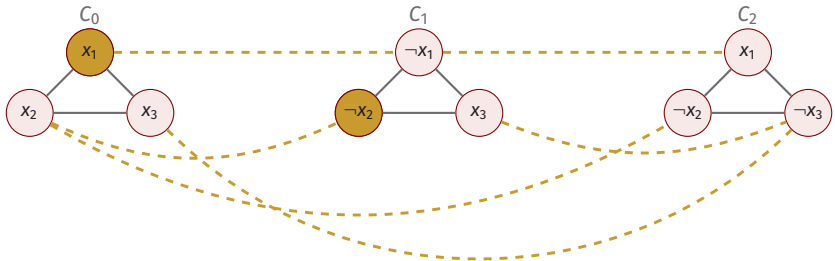
$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$, con $k = 3$.




- En $C_0$, $x_1$: nada elegido todavía.

# Un ejemplo seguido en detalle

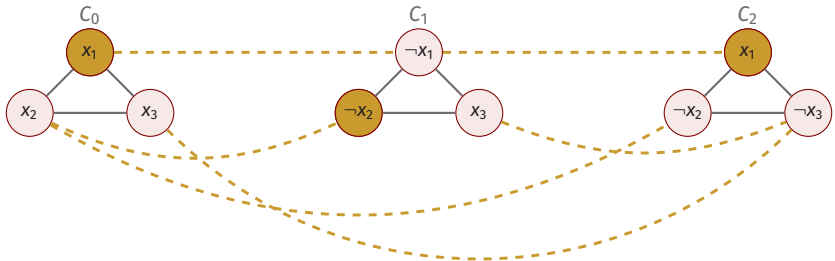
$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$, con $k = 3$.




- En $C_0$, $x_1$: nada elegido todavía.
- En $C_1$, $\neg x_2$: $\neg x_1$ está unido al $x_1$ elegido.

# Un ejemplo seguido en detalle

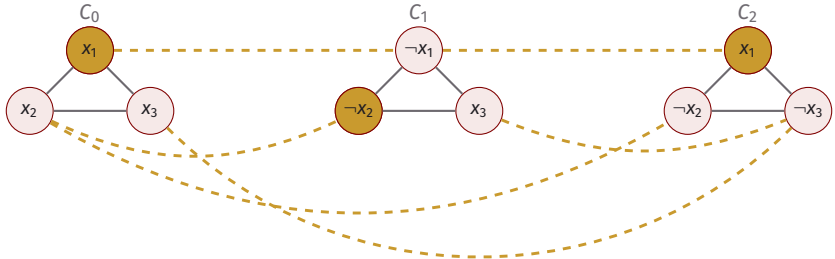
$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$, con $k = 3$.




- En $C_0$, $x_1$: nada elegido todavía.
- En $C_1$, $\neg x_2$: $\neg x_1$ está unido al $x_1$ elegido.
- En $C_2$, $x_1$ otra vez: mismo literal, sin arista.

# Un ejemplo seguido en detalle

$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$, con $k = 3$.




- En $C_0$, $x_1$: nada elegido todavía.
- En $C_1$, $\neg x_2$: $\neg x_1$ está unido al $x_1$ elegido.
- En $C_2$, $x_1$ otra vez: mismo literal, sin arista.
- **Lectura:** $x_1 = V$, $x_2 = F$. $x_3$ da igual.

# Las dos direcciones

**De una asignación a un conjunto.**

# Las dos direcciones

**De una asignación a un conjunto.**

- En cada cláusula hay un literal verdadero: se elige un nodo con él. Son $m$ nodos.

# Las dos direcciones

**De una asignación a un conjunto.**

- En cada cláusula hay un literal verdadero: se elige un nodo con él. Son $m$ nodos.
- Sin aristas de triángulo, porque hay uno por cláusula.

# Las dos direcciones

**De una asignación a un conjunto.**

- En cada cláusula hay un literal verdadero: se elige un nodo con él. Son $m$ nodos.
- Sin aristas de triángulo, porque hay uno por cláusula.
- Sin aristas de opuestos, porque $x$ y $\neg x$ no son los dos verdaderos.

# Las dos direcciones

**De una asignación a un conjunto.**

- En cada cláusula hay un literal verdadero: se elige un nodo con él. Son $m$ nodos.
- Sin aristas de triángulo, porque hay uno por cláusula.
- Sin aristas de opuestos, porque $x$ y $\neg x$ no son los dos verdaderos.

**De un conjunto a una asignación.**

# Las dos direcciones

**De una asignación a un conjunto.**

- En cada cláusula hay un literal verdadero: se elige un nodo con él. Son $m$ nodos.
- Sin aristas de triángulo, porque hay uno por cláusula.
- Sin aristas de opuestos, porque $x$ y $\neg x$ no son los dos verdaderos.

**De un conjunto a una asignación.**

- A lo sumo uno por triángulo, y hay $m$ triángulos: **exactamente uno** en cada uno.

# Las dos direcciones

**De una asignación a un conjunto.**

- En cada cláusula hay un literal verdadero: se elige un nodo con él. Son $m$ nodos.
- Sin aristas de triángulo, porque hay uno por cláusula.
- Sin aristas de opuestos, porque $x$ y $\neg x$ no son los dos verdaderos.

**De un conjunto a una asignación.**

- A lo sumo uno por triángulo, y hay $m$ triángulos: **exactamente uno** en cada uno.
- Su literal es verdadero. No se contradice, porque habría una arista.

# Las dos direcciones

**De una asignación a un conjunto.**

- En cada cláusula hay un literal verdadero: se elige un nodo con él. Son $m$ nodos.
- Sin aristas de triángulo, porque hay uno por cláusula.
- Sin aristas de opuestos, porque $x$ y $\neg x$ no son los dos verdaderos.

**De un conjunto a una asignación.**

- A lo sumo uno por triángulo, y hay $m$ triángulos: **exactamente uno** en cada uno.
- Su literal es verdadero. No se contradice, porque habría una arista.
- Cada cláusula tiene un literal verdadero. ■

# Las dos direcciones

**De una asignación a un conjunto.**

- En cada cláusula hay un literal verdadero: se elige un nodo con él. Son $m$ nodos.
- Sin aristas de triángulo, porque hay uno por cláusula.
- Sin aristas de opuestos, porque $x$ y $\neg x$ no son los dos verdaderos.

**De un conjunto a una asignación.**

- A lo sumo uno por triángulo, y hay $m$ triángulos: **exactamente uno** en cada uno.
- Su literal es verdadero. No se contradice, porque habría una arista.
- Cada cláusula tiene un literal verdadero. ■

Tiempo: $3m$ nodos y $O(m^2)$ aristas. Por la semana 13, cobertura de vértices y clique también son NP-completos.

# Pregunta de repaso 2, y aristas de menos

build/is_demo, extracto

```
Dos cláusulas (pregunta de repaso 2):
(x1 ∨ ¬x2 ∨ x3) ∧ (¬x1 ∨ x2 ∨ ¬x3)
  nodos: 0:x1 1:¬x2 2:x3 3:¬x1 4:x2 5:¬x3
  aristas de literales opuestos: 0-3 1-4 2-5
  k = 2, conjunto independiente máximo: 2 (0:x1, 4:x2)
  fuerza bruta sobre la fórmula: satisfacible, x1=F x2=F x3=F
  lectura del conjunto: x1=V x2=V x3=F, ¿cumple? sí

Sin solución: las ocho cláusulas sobre tres variables.
  24 nodos, 72 aristas, k = 8, máximo: 7

Si faltan aristas, en esa misma fórmula insatisfacible:
  sin literales opuestos: máximo 8, k = 8
  sin triángulos:         máximo 12, k = 8
```

Sin alguna clase de aristas, se rompe la dirección de vuelta: inventa una solución que no existe.

SECCIÓN 05

# De 3-SAT a 3-coloreo

05



# 3-coloreo, y qué es un gadget

**3-coloreo**

¿Se puede dar uno de tres colores a cada nodo, con colores distintos en los extremos de cada arista?

# 3-coloreo, y qué es un gadget

**3-coloreo**

¿Se puede dar uno de tres colores a cada nodo, con colores distintos en los extremos de cada arista?

**Está en NP:** el certificado es el color de cada nodo, y se revisa arista por arista.

# 3-coloreo, y qué es un gadget

**3-coloreo**

¿Se puede dar uno de tres colores a cada nodo, con colores distintos en los extremos de cada arista?

**Está en NP:** el certificado es el color de cada nodo, y se revisa arista por arista.

**Gadget**

Pieza pequeña de la instancia nueva, diseñada para portarse como una regla de la original.

# 3-coloreo, y qué es un gadget

**3-coloreo**

¿Se puede dar uno de tres colores a cada nodo, con colores distintos en los extremos de cada arista?

**Está en NP:** el certificado es el color de cada nodo, y se revisa arista por arista.

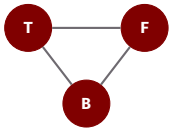
**Gadget**

Pieza pequeña de la instancia nueva, diseñada para portarse como una regla de la original.

Necesitamos tres: qué es verdadero y falso, la **paleta**; una **variable**; y un **O**.

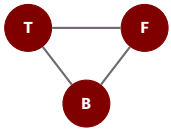
# La paleta y la variable

**Paleta.** Un triángulo T, F, B: tres colores distintos.



# La paleta y la variable

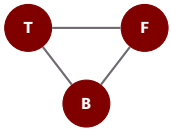
**Paleta.** Un triángulo T, F, B: tres colores distintos.




Los colores no traen nombre: **llamamos** T al color del nodo T. Así, “verdadero” significa tener ese color.

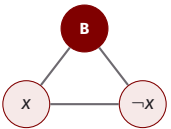
# La paleta y la variable

**Paleta.** Un triángulo T, F, B: tres colores distintos.



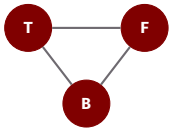

Los colores no traen nombre: **llamamos** T al color del nodo T. Así, “verdadero” significa tener ese color.

**Variable.** Dos nodos, unidos entre sí y con B.



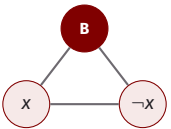
# La paleta y la variable

**Paleta.** Un triángulo T, F, B: tres colores distintos.




Los colores no traen nombre: **llamamos** T al color del nodo T. Así, “verdadero” significa tener ese color.

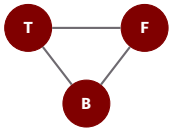
**Variable.** Dos nodos, unidos entre sí y con B.




Ninguno puede ser B, y no pueden ser iguales. Uno es T y el otro F.

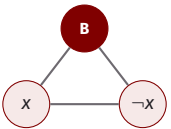
# La paleta y la variable

**Paleta.** Un triángulo T, F, B: tres colores distintos.



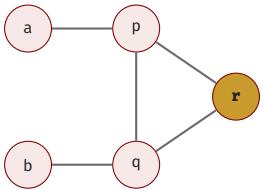

Los colores no traen nombre: **llamamos** T al color del nodo T. Así, “verdadero” significa tener ese color.

**Variable.** Dos nodos, unidos entre sí y con B.

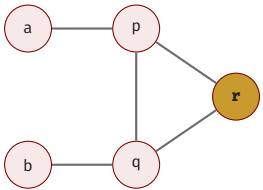



Ninguno puede ser B, y no pueden ser iguales. Uno es T y el otro F.
**Eso es una asignación, sin contradicciones.**

# La compuerta O

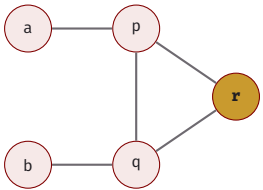


# La compuerta O




El triángulo $p$, $q$, $r$ usa los tres colores.

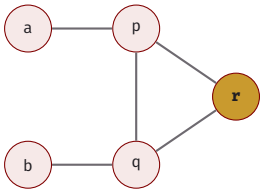
# La compuerta O




El triángulo $p$, $q$, $r$ usa los tres colores.

**Propiedad 1.** Si $a$ y $b$ son F, entonces $r$ es F, siempre.
$p \neq F$ y $q \neq F$, y el F le toca a $r$.

# La compuerta O

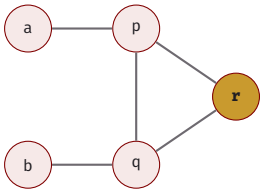



El triángulo $p$, $q$, $r$ usa los tres colores.

**Propiedad 1.** Si $a$ y $b$ son F, entonces $r$ es F, siempre. $p \neq F$ y $q \neq F$, y el F le toca a $r$.

**Propiedad 2.** Si alguna entrada es T, y las dos son T o F, $r$ puede ser T.

# La compuerta O



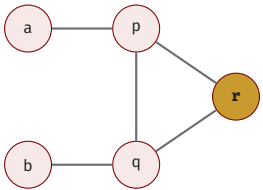

El triángulo $p$, $q$, $r$ usa los tres colores.

**Propiedad 1.** Si $a$ y $b$ son F, entonces $r$ es F, siempre. $p \neq F$ y $q \neq F$, y el F le toca a $r$.

**Propiedad 2.** Si alguna entrada es T, y las dos son T o F, $r$ puede ser T.

| a | b | p | q | r |
|---|---|---|---|---|
| T | T | B | F | T |
| T | F | F | B | T |
| F | T | B | F | T |

# La compuerta O




El triángulo $p$, $q$, $r$ usa los tres colores.

**Propiedad 1.** Si $a$ y $b$ son F, entonces $r$ es F, siempre. $p \neq F$ y $q \neq F$, y el F le toca a $r$.

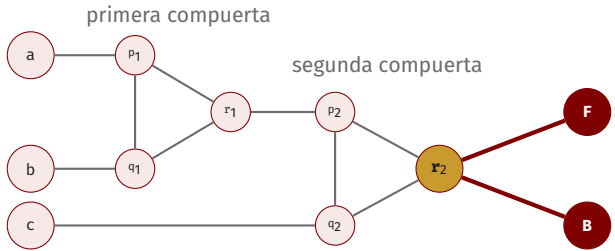
**Propiedad 2.** Si alguna entrada es T, y las dos son T o F, $r$ puede ser T.

| a | b | p | q | r |
|---|---|---|---|---|
| T | T | B | F | T |
| T | F | F | B | T |
| F | T | B | F | T |

No es un O exacto: con una entrada T, $r$ también admite otros colores. Solo necesitamos estas dos propiedades.

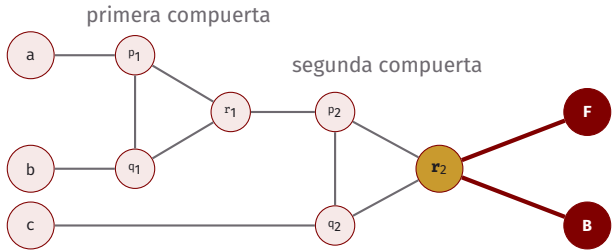
# El gadget de cláusula

$(a \vee b \vee c)$ es dos compuertas encadenadas, y la salida va unida a F y a B.



# El gadget de cláusula

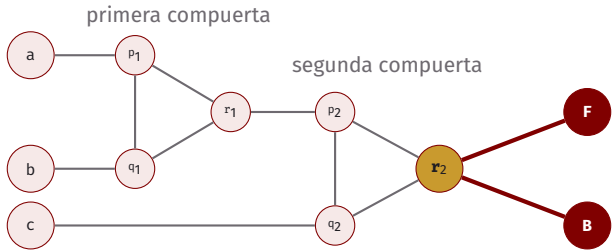
$(a \vee b \vee c)$ es dos compuertas encadenadas, y la salida va unida a F y a B.




**Lema.** Con la paleta y $a$, $b$, $c$ pintados de T o F, se puede completar si y solo si alguno es T.

# El gadget de cláusula

$(a \vee b \vee c)$ es dos compuertas encadenadas, y la salida va unida a F y a B.

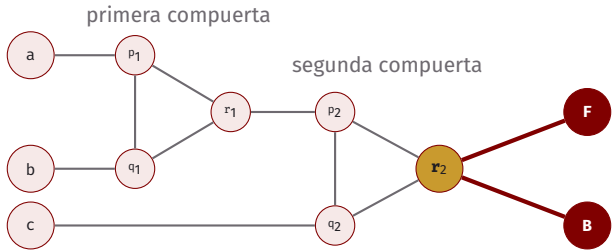



**Lema.** Con la paleta y $a$, $b$, $c$ pintados de T o F, se puede completar si y solo si alguno es T.

- **Todos F:** $r_1 = F$, luego $r_2 = F$, y $r_2$ está unido a F. Imposible.

# El gadget de cláusula

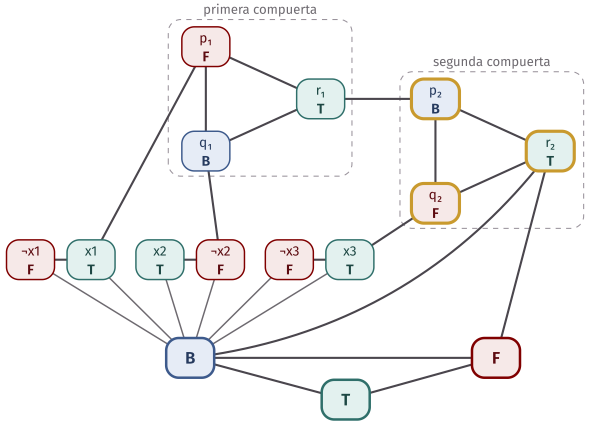
$(a \vee b \vee c)$ es dos compuertas encadenadas, y la salida va unida a F y a B.




**Lema.** Con la paleta y $a$, $b$, $c$ pintados de T o F, se puede completar si y solo si alguno es T.

- **Todos F:** $r_1 = F$, luego $r_2 = F$, y $r_2$ está unido a F. Imposible.
- **Alguno T:** se pinta $r_1$ de T o F, y luego $r_2 = T$, por la Propiedad 2. ■

# El gadget: un caso que se completa y uno que no




| a | b | c | coloreos |
|---|---|---|---|
| F | F | F | 0 |
| F | F | T | 2 |
| F | T | F | 2 |
| F | T | T | 4 |
| T | F | F | 2 |
| T | F | T | 4 |
| T | T | F | 2 |
| T | T | T | 4 |

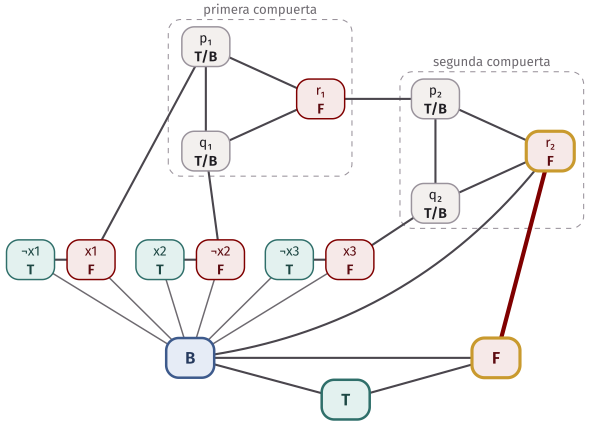
de 729 posibles

a = x1, b = ¬x2, c = x3

Segunda compuerta, fila T T: $p_2$ = B, $q_2$ = F, $r_2$ = T.

$r_2$ = T no choca con F ni con B. Es uno de los 4 coloreos de la entrada T F T.

# El gadget: un caso que se completa y uno que no




| a | b | c | coloreos |
|---|---|---|---|
| F | F | F | 0 |
| F | F | T | 2 |
| F | T | F | 2 |
| F | T | T | 4 |
| T | F | F | 2 |
| T | F | T | 4 |
| T | T | F | 2 |
| T | T | T | 4 |

de 729 posibles

a = x1, b = ¬x2, c = x3

Pero $r_2$ está unida al nodo F, que también tiene color F: es imposible.

Por eso la entrada F F F tiene 0 coloreos de 729, y la cláusula obliga a tener un T.

# Las 8 entradas, enumeradas

El gadget: de 729 coloreos

| a | b | c | coloreos |
|---|---|---|---|
| F | F | F | 0 |
| F | F | T | 2 |
| F | T | F | 2 |
| F | T | T | 4 |
| T | F | F | 2 |
| T | F | T | 4 |
| T | T | F | 2 |
| T | T | T | 4 |

Una compuerta

| a | b | r |
|---|---|---|
| F | F | F |
| F | T | T F B |
| T | F | T F B |
| T | T | T |

# Las 8 entradas, enumeradas

El gadget: de 729 coloreos

| a | b | c | coloreos |
|---|---|---|---|
| F | F | F | 0 |
| F | F | T | 2 |
| F | T | F | 2 |
| F | T | T | 4 |
| T | F | F | 2 |
| T | F | T | 4 |
| T | T | F | 2 |
| T | T | T | 4 |

Una compuerta

| a | b | r |
|---|---|---|
| F | F | F |
| F | T | T F B |
| T | F | T F B |
| T | T | T |

**Ojo**

La salida $r_1$ puede ser B. La vuelta solo usa la Propiedad 1, y a la ida se elige T o F.

# La reducción completa

Paleta, un gadget por variable, y uno por cláusula con los nodos de sus literales como entradas.

# La reducción completa

Paleta, un gadget por variable, y uno por cláusula con los nodos de sus literales como entradas.

**De una asignación a un coloreo.** Literales verdaderos de T, falsos de F. Cada cláusula tiene un T, y el Lema completa su gadget.

# La reducción completa

Paleta, un gadget por variable, y uno por cláusula con los nodos de sus literales como entradas.

**De una asignación a un coloreo.** Literales verdaderos de T, falsos de F. Cada cláusula tiene un T, y el Lema completa su gadget.

**De un coloreo a una asignación.**

# La reducción completa

Paleta, un gadget por variable, y uno por cláusula con los nodos de sus literales como entradas.

**De una asignación a un coloreo.** Literales verdaderos de T, falsos de F. Cada cláusula tiene un T, y el Lema completa su gadget.

**De un coloreo a una asignación.**

1. La paleta usa tres colores distintos. **Los llamamos** T, F, B.

# La reducción completa

Paleta, un gadget por variable, y uno por cláusula con los nodos de sus literales como entradas.

**De una asignación a un coloreo.** Literales verdaderos de T, falsos de F. Cada cláusula tiene un T, y el Lema completa su gadget.

**De un coloreo a una asignación.**

1. La paleta usa tres colores distintos. **Los llamamos** T, F, B.
2. Cada par $x$, $\neg x$ tiene un T y un F. $x$ es verdadera si su nodo tiene el color T.

# La reducción completa

Paleta, un gadget por variable, y uno por cláusula con los nodos de sus literales como entradas.

**De una asignación a un coloreo.** Literales verdaderos de T, falsos de F. Cada cláusula tiene un T, y el Lema completa su gadget.

**De un coloreo a una asignación.**

1. La paleta usa tres colores distintos. **Los llamamos** T, F, B.
2. Cada par $x$, $\neg x$ tiene un T y un F. $x$ es verdadera si su nodo tiene el color T.
3. Cada gadget de cláusula está coloreado, y por el Lema alguna entrada es T. ■

# La reducción completa

Paleta, un gadget por variable, y uno por cláusula con los nodos de sus literales como entradas.

**De una asignación a un coloreo.** Literales verdaderos de T, falsos de F. Cada cláusula tiene un T, y el Lema completa su gadget.

**De un coloreo a una asignación.**

1. La paleta usa tres colores distintos. **Los llamamos** T, F, B.
2. Cada par $x$, $\neg x$ tiene un T y un F. $x$ es verdadera si su nodo tiene el color T.
3. Cada gadget de cláusula está coloreado, y por el Lema alguna entrada es T. ■

**Tamaño:** $3 + 2n + 6m$ nodos, y $3 + 3n + 12m$ aristas.

# Corre, y qué arista hace falta

build/coloring_demo, extracto

```
(x1 ∨ x2 ∨ x3) ∧ (¬x1 ∨ ¬x2 ∨ x3) ∧ (x1 ∨ ¬x2 ∨ ¬x3)
  27 nodos, 48 aristas
  ¿3-coloreable? sí, fuerza bruta sobre la fórmula: satisfacible
  lectura del coloreo: x1=V x2=V x3=V, ¿cumple? sí

Si faltan aristas, en la fórmula de ocho cláusulas, que no tiene solución:
  variables sin arista a B:      3-coloreable
  salida sin arista a F:         3-coloreable
  salida sin arista a B:         no 3-coloreable
```

# Corre, y qué arista hace falta

build/coloring_demo, extracto

```
(x1 ∨ x2 ∨ x3) ∧ (¬x1 ∨ ¬x2 ∨ x3) ∧ (x1 ∨ ¬x2 ∨ ¬x3)
  27 nodos, 48 aristas
  ¿3-coloreable? sí, fuerza bruta sobre la fórmula: satisfacible
  lectura del coloreo: x1=V x2=V x3=V, ¿cumple? sí

Si faltan aristas, en la fórmula de ocho cláusulas, que no tiene solución:
  variables sin arista a B:      3-coloreable
  salida sin arista a F:         3-coloreable
  salida sin arista a B:         no 3-coloreable
```

- Sin la arista a B, $x$ puede ser B, y $x$ y $\neg x$ quedan “los dos activos”.

# Corre, y qué arista hace falta

build/coloring_demo, extracto

```
(x1 ∨ x2 ∨ x3) ∧ (¬x1 ∨ ¬x2 ∨ x3) ∧ (x1 ∨ ¬x2 ∨ ¬x3)
  27 nodos, 48 aristas
  ¿3-coloreable? sí, fuerza bruta sobre la fórmula: satisfacible
  lectura del coloreo: x1=V x2=V x3=V, ¿cumple? sí

Si faltan aristas, en la fórmula de ocho cláusulas, que no tiene solución:
  variables sin arista a B:      3-coloreable
  salida sin arista a F:         3-coloreable
  salida sin arista a B:         no 3-coloreable
```

- Sin la arista a B, $x$ puede ser B, y $x$ y $\neg x$ quedan “los dos activos”.
- Sin la arista a F, tres entradas falsas ya no molestan.

# Corre, y qué arista hace falta

build/coloring_demo, extracto

```
(x1 ∨ x2 ∨ x3) ∧ (¬x1 ∨ ¬x2 ∨ x3) ∧ (x1 ∨ ¬x2 ∨ ¬x3)
  27 nodos, 48 aristas
  ¿3-coloreable? sí, fuerza bruta sobre la fórmula: satisfacible
  lectura del coloreo: x1=V x2=V x3=V, ¿cumple? sí

Si faltan aristas, en la fórmula de ocho cláusulas, que no tiene solución:
  variables sin arista a B:      3-coloreable
  salida sin arista a F:         3-coloreable
  salida sin arista a B:         no 3-coloreable
```

- Sin la arista a B, $x$ puede ser B, y $x$ y $\neg x$ quedan “los dos activos”.
- Sin la arista a F, tres entradas falsas ya no molestan.
- La arista de la salida a B **no hace falta**: la vuelta solo usó $r_2 \neq F$. Se deja porque se lee mejor. Ejercicio 7.

SECCIÓN 06

# De cobertura de vértices a suma de subconjuntos

06

# Suma de subconjuntos

**Suma de subconjuntos** ***(subset sum)***

Dada una lista de enteros positivos, en binario, y un objetivo $t$: ¿alguna parte suma exactamente $t$?

# Suma de subconjuntos

**Suma de subconjuntos** ***(subset sum)***

Dada una lista de enteros positivos, en binario, y un objetivo $t$: ¿alguna parte suma exactamente $t$?

**Está en NP:** el certificado son las posiciones elegidas.

# Suma de subconjuntos

**Suma de subconjuntos** ***(subset sum)***

Dada una lista de enteros positivos, en binario, y un objetivo $t$: ¿alguna parte suma exactamente $t$?

**Está en NP:** el certificado son las posiciones elegidas.

Reducimos desde **cobertura de vértices**: ¿hay a lo sumo $k$ nodos que toquen todas las aristas?

# Suma de subconjuntos

**Suma de subconjuntos** *(subset sum)*

Dada una lista de enteros positivos, en binario, y un objetivo $t$: ¿alguna parte suma exactamente $t$?

**Está en NP:** el certificado son las posiciones elegidas.

Reducimos desde **cobertura de vértices**: ¿hay a lo sumo $k$ nodos que toquen todas las aristas?

**La dificultad**

En una suma no hay condiciones lógicas. Lo único que controlamos es qué dígitos salen.

# La tabla, en base 4

**Un dígito por arista, y uno arriba.** Si $k > V$, se usa $k = V$.

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 0 | 1 | 0 | 0 | 1 | 1 |
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| nodo 3 | 1 | 1 | 0 | 0 | 0 |
| holgura $e_0$ | 0 | 0 | 0 | 0 | 1 |
| holgura $e_1$ | 0 | 0 | 0 | 1 | 0 |
| holgura $e_2$ | 0 | 0 | 1 | 0 | 0 |
| holgura $e_3$ | 0 | 1 | 0 | 0 | 0 |
| objetivo | 2 | 2 | 2 | 2 | 2 |

# La tabla, en base 4

**Un dígito por arista, y uno arriba.** Si $k > V$, se usa $k = V$.

- **Número de nodo:** 1 arriba, y 1 en cada arista que toca.

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 0 | 1 | 0 | 0 | 1 | 1 |
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| nodo 3 | 1 | 1 | 0 | 0 | 0 |
| holgura $e_0$ | 0 | 0 | 0 | 0 | 1 |
| holgura $e_1$ | 0 | 0 | 0 | 1 | 0 |
| holgura $e_2$ | 0 | 0 | 1 | 0 | 0 |
| holgura $e_3$ | 0 | 1 | 0 | 0 | 0 |
| objetivo | 2 | 2 | 2 | 2 | 2 |

# La tabla, en base 4

**Un dígito por arista, y uno arriba.** Si $k > V$, se usa $k = V$.

- **Número de nodo:** 1 arriba, y 1 en cada arista que toca.
- **Holgura de una arista:** 1 en esa arista.

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 0 | 1 | 0 | 0 | 1 | 1 |
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| nodo 3 | 1 | 1 | 0 | 0 | 0 |
| holgura $e_0$ | 0 | 0 | 0 | 0 | 1 |
| holgura $e_1$ | 0 | 0 | 0 | 1 | 0 |
| holgura $e_2$ | 0 | 0 | 1 | 0 | 0 |
| holgura $e_3$ | 0 | 1 | 0 | 0 | 0 |
| objetivo | 2 | 2 | 2 | 2 | 2 |

# La tabla, en base 4

**Un dígito por arista, y uno arriba.** Si $k > V$, se usa $k = V$.

- **Número de nodo:** 1 arriba, y 1 en cada arista que toca.
- **Holgura de una arista:** 1 en esa arista.
- **Objetivo:** $k$ arriba, y 2 en cada arista.

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 0 | 1 | 0 | 0 | 1 | 1 |
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| nodo 3 | 1 | 1 | 0 | 0 | 0 |
| holgura $e_0$ | 0 | 0 | 0 | 0 | 1 |
| holgura $e_1$ | 0 | 0 | 0 | 1 | 0 |
| holgura $e_2$ | 0 | 0 | 1 | 0 | 0 |
| holgura $e_3$ | 0 | 1 | 0 | 0 | 0 |
| objetivo | 2 | 2 | 2 | 2 | 2 |

# La tabla, en base 4

**Un dígito por arista, y uno arriba.** Si $k > V$, se usa $k = V$.

- **Número de nodo:** 1 arriba, y 1 en cada arista que toca.
- **Holgura de una arista:** 1 en esa arista.
- **Objetivo:** $k$ arriba, y 2 en cada arista.

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 0 | 1 | 0 | 0 | 1 | 1 |
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| nodo 3 | 1 | 1 | 0 | 0 | 0 |
| holgura $e_0$ | 0 | 0 | 0 | 0 | 1 |
| holgura $e_1$ | 0 | 0 | 0 | 1 | 0 |
| holgura $e_2$ | 0 | 0 | 1 | 0 | 0 |
| holgura $e_3$ | 0 | 1 | 0 | 0 | 0 |
| objetivo | 2 | 2 | 2 | 2 | 2 |

Con $e_0 = 0$-1, $e_1 = 0$-2, $e_2 = 1$-2 y $e_3 = 2$-3, y $k = 2$.

# La tabla, en base 4

**Un dígito por arista, y uno arriba.** Si $k > V$, se usa $k = V$.

- **Número de nodo:** 1 arriba, y 1 en cada arista que toca.
- **Holgura de una arista:** 1 en esa arista.
- **Objetivo:** $k$ arriba, y 2 en cada arista.

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 0 | 1 | 0 | 0 | 1 | 1 |
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| nodo 3 | 1 | 1 | 0 | 0 | 0 |
| holgura $e_0$ | 0 | 0 | 0 | 0 | 1 |
| holgura $e_1$ | 0 | 0 | 0 | 1 | 0 |
| holgura $e_2$ | 0 | 0 | 1 | 0 | 0 |
| holgura $e_3$ | 0 | 1 | 0 | 0 | 0 |
| objetivo | 2 | 2 | 2 | 2 | 2 |

Con $e_0 = 0$-1, $e_1 = 0$-2, $e_2 = 1$-2 y $e_3 = 2$-3, y $k = 2$.

Pedir 2 en una columna: dos extremos, o un extremo y su holgura. **Cero extremos no se puede:** es la cobertura escrita en dígitos.

# Sumemos por columnas, con $k = 2$

| arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|
| | | | | |

# Sumemos por columnas, con $k = 2$

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| | | | | | |
| | | | | | |
| | | | | | |

# Sumemos por columnas, con $k = 2$

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| | | | | | |

# Sumemos por columnas, con $k = 2$

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| holgura $e_0$ | | | | | 1 |

# Sumemos por columnas, con $k = 2$

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| holgura $e_0$ | | | | | 1 |
| holgura $e_1$ | | | | 1 | |

# Sumemos por columnas, con $k = 2$

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| holgura $e_0$ | | | | | 1 |
| holgura $e_1$ | | | | 1 | |
| holgura $e_3$ | | 1 | | | |

# Sumemos por columnas, con $k = 2$

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| holgura $e_0$ | | | | | 1 |
| holgura $e_1$ | | | | 1 | |
| holgura $e_3$ | | 1 | | | |
| **suma** | **2** | **2** | **2** | **2** | **2** |

# Sumemos por columnas, con $k = 2$

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| holgura $e_0$ | | | | | 1 |
| holgura $e_1$ | | | | 1 | |
| holgura $e_3$ | | 1 | | | |
| **suma** | **2** | **2** | **2** | **2** | **2** |

$e_2$ tiene sus dos extremos elegidos, y no usa holgura. Las otras tres tienen uno, y usan la suya.

# Sumemos por columnas, con $k = 2$

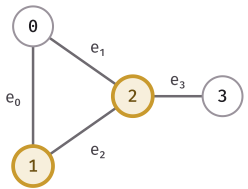
| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ |
|---|---|---|---|---|---|
| nodo 1 | 1 | 0 | 1 | 0 | 1 |
| nodo 2 | 1 | 1 | 1 | 1 | 0 |
| holgura $e_0$ | | | | | 1 |
| holgura $e_1$ | | | | 1 | |
| holgura $e_3$ | | 1 | | | |
| **suma** | **2** | **2** | **2** | **2** | **2** |

$e_2$ tiene sus dos extremos elegidos, y no usa holgura. Las otras tres tienen uno, y usan la suya.

En decimal: $273 + 340 + 1 + 4 + 64 = 682$. Los nodos elegidos, 1 y 2, son la cobertura.

# La misma suma, junto al grafo

**El grafo, con k = 2**




Cobertura: los nodos 1 y 2.

$e_2$ tiene sus dos extremos elegidos: no usa holgura.

$e_0$, $e_1$ y $e_3$ tienen uno, y usan la suya.

| | arriba | $e_3$ | $e_2$ | $e_1$ | $e_0$ | decimal |
|---|---|---|---|---|---|---|
| nodo 0 | 1 | 0 | 0 | 1 | 1 | 261 |
| **nodo 1** | 1 | 0 | 1 | 0 | 1 | 273 |
| **nodo 2** | 1 | 1 | 1 | 1 | 0 | 340 |
| nodo 3 | 1 | 1 | 0 | 0 | 0 | 320 |
| **holgura $e_0$** | 0 | 0 | 0 | 0 | 1 | 1 |
| **holgura $e_1$** | 0 | 0 | 0 | 1 | 0 | 4 |
| holgura $e_2$ | 0 | 0 | 1 | 0 | 0 | 16 |
| **holgura $e_3$** | 0 | 1 | 0 | 0 | 0 | 64 |
| **suma = objetivo** | 2 | 2 | 2 | 2 | 2 | 682 |
| **unos de la lista** | 4 | 3 | 3 | 3 | 3 | |

Pedir 2 en una columna: dos extremos, o un extremo y su holgura. Cero extremos no llegan.

En cada columna de arista hay tres 1 en toda la lista: 3 < 4, y por eso nunca hay acarreos.

```
273 + 340 + 1 + 4 + 64 = 682
```

# Por qué no hay acarreos

**La cuenta**

En cada columna de arista, entre **todos** los números hay exactamente tres 1: los dos extremos y la holgura. Aunque se elijan todos, la columna suma $3 < 4$.

# Por qué no hay acarreos

**La cuenta**

En cada columna de arista, entre **todos** los números hay exactamente tres 1: los dos extremos y la holgura. Aunque se elijan todos, la columna suma $3 < 4$.

La columna de arriba puede pasar de 3, pero no tiene nada encima que pisar.

# Por qué no hay acarreos

**La cuenta**

En cada columna de arista, entre **todos** los números hay exactamente tres 1: los dos extremos y la holgura. Aunque se elijan todos, la columna suma $3 < 4$.

La columna de arriba puede pasar de 3, pero no tiene nada encima que pisar.

**El hecho central.** Sean $c$ los números de nodo elegidos y $d_j$ los 1 en la columna $e_j$:

$$\text{suma} = c \cdot 4^E + \sum_j d_j \cdot 4^j, \quad \text{con} \sum_j d_j \cdot 4^j < 4^E.$$

# Por qué no hay acarreos

**La cuenta**

En cada columna de arista, entre **todos** los números hay exactamente tres 1: los dos extremos y la holgura. Aunque se elijan todos, la columna suma $3 < 4$.

La columna de arriba puede pasar de 3, pero no tiene nada encima que pisar.

**El hecho central.** Sean $c$ los números de nodo elegidos y $d_j$ los 1 en la columna $e_j$:

$$\text{suma} = c \cdot 4^E + \sum_j d_j \cdot 4^j, \quad \text{con} \sum_j d_j \cdot 4^j < 4^E.$$

La escritura en base 4 es única. Entonces, suma $= t$ si y solo si $c = k$ **y todo** $d_j = 2$.

# Las dos direcciones

**De una cobertura a una elección.**

# Las dos direcciones

**De una cobertura a una elección.**

1. Una cobertura de tamaño a lo sumo $k$ se completa hasta tener exactamente $k$ (se puede, porque $k \leq V$).

# Las dos direcciones

**De una cobertura a una elección.**

1. Una cobertura de tamaño a lo sumo $k$ se completa hasta tener exactamente $k$ (se puede, porque $k \leq V$).
2. Se eligen esos $k$ números de nodo: $c = k$.

# Las dos direcciones

**De una cobertura a una elección.**

1. Una cobertura de tamaño a lo sumo $k$ se completa hasta tener exactamente $k$ (se puede, porque $k \leq V$).
2. Se eligen esos $k$ números de nodo: $c = k$.
3. Cada arista tiene 1 o 2 extremos elegidos. Si tiene 1, se agrega su holgura. Todo $d_j = 2$.

# Las dos direcciones

**De una cobertura a una elección.**

1. Una cobertura de tamaño a lo sumo $k$ se completa hasta tener exactamente $k$ (se puede, porque $k \leq V$).
2. Se eligen esos $k$ números de nodo: $c = k$.
3. Cada arista tiene 1 o 2 extremos elegidos. Si tiene 1, se agrega su holgura. Todo $d_j = 2$.

**De una elección a una cobertura.**

# Las dos direcciones

**De una cobertura a una elección.**

1. Una cobertura de tamaño a lo sumo $k$ se completa hasta tener exactamente $k$ (se puede, porque $k \leq V$).
2. Se eligen esos $k$ números de nodo: $c = k$.
3. Cada arista tiene 1 o 2 extremos elegidos. Si tiene 1, se agrega su holgura. Todo $d_j = 2$.

**De una elección a una cobertura.**

1. Por el hecho central, hay $k$ números de nodo, y todo $d_j = 2$.

# Las dos direcciones

**De una cobertura a una elección.**

1. Una cobertura de tamaño a lo sumo $k$ se completa hasta tener exactamente $k$ (se puede, porque $k \leq V$).
2. Se eligen esos $k$ números de nodo: $c = k$.
3. Cada arista tiene 1 o 2 extremos elegidos. Si tiene 1, se agrega su holgura. Todo $d_j = 2$.

**De una elección a una cobertura.**

1. Por el hecho central, hay $k$ números de nodo, y todo $d_j = 2$.
2. La holgura aporta a lo sumo 1. Luego, al menos un extremo de cada arista está elegido. ■

# Las dos direcciones

**De una cobertura a una elección.**

1. Una cobertura de tamaño a lo sumo $k$ se completa hasta tener exactamente $k$ (se puede, porque $k \leq V$).
2. Se eligen esos $k$ números de nodo: $c = k$.
3. Cada arista tiene 1 o 2 extremos elegidos. Si tiene 1, se agrega su holgura. Todo $d_j = 2$.

**De una elección a una cobertura.**

1. Por el hecho central, hay $k$ números de nodo, y todo $d_j = 2$.
2. La holgura aporta a lo sumo 1. Luego, al menos un extremo de cada arista está elegido. ■

**Tamaño:** $V + E$ números de $2E + \log V + O(1)$ bits. Polinomial, en bits.

# ¿Hace falta la base 4?

build/subset_demo, extracto

```
En base 2 los dígitos se pisan:
  aristas 0-1 y 2-3, k = 0, cobertura mínima 2
  números: 5 5 6 6 1 2, objetivo 6
  la suma dice sí, con el número 6 (posición 2)

Búsqueda de respuestas falsas en otras bases:
  base 2: hay una respuesta falsa
  base 3: ninguna respuesta falsa en 4000 grafos al azar
  base 4: ninguna respuesta falsa en 4000 grafos al azar
```

# ¿Hace falta la base 4?

build/subset_demo, extracto

```
En base 2 los dígitos se pisan:
  aristas 0-1 y 2-3, k = 0, cobertura mínima 2
  números: 5 5 6 6 1 2, objetivo 6
  la suma dice sí, con el número 6 (posición 2)

Búsqueda de respuestas falsas en otras bases:
  base 2: hay una respuesta falsa
  base 3: ninguna respuesta falsa en 4000 grafos al azar
  base 4: ninguna respuesta falsa en 4000 grafos al azar
```

- En base 2 no se puede escribir un dígito 2, y el objetivo ya no dice lo que queremos.

# ¿Hace falta la base 4?

build/subset_demo, extracto

```
En base 2 los dígitos se pisan:
  aristas 0-1 y 2-3, k = 0, cobertura mínima 2
  números: 5 5 6 6 1 2, objetivo 6
  la suma dice sí, con el número 6 (posición 2)

Búsqueda de respuestas falsas en otras bases:
  base 2: hay una respuesta falsa
  base 3: ninguna respuesta falsa en 4000 grafos al azar
  base 4: ninguna respuesta falsa en 4000 grafos al azar
```

- En base 2 no se puede escribir un dígito 2, y el objetivo ya no dice lo que queremos.
- En base 3 una columna puede llevar 1, y aun así no apareció ninguna falla. Eso no es una demostración: el ejercicio 6 la pide, y la base 3 también sirve.

# ¿Hace falta la base 4?

build/subset_demo, extracto

```
En base 2 los dígitos se pisan:
  aristas 0-1 y 2-3, k = 0, cobertura mínima 2
  números: 5 5 6 6 1 2, objetivo 6
  la suma dice sí, con el número 6 (posición 2)

Búsqueda de respuestas falsas en otras bases:
  base 2: hay una respuesta falsa
  base 3: ninguna respuesta falsa en 4000 grafos al azar
  base 4: ninguna respuesta falsa en 4000 grafos al azar
```

- En base 2 no se puede escribir un dígito 2, y el objetivo ya no dice lo que queremos.
- En base 3 una columna puede llevar 1, y aun así no apareció ninguna falla. Eso no es una demostración: el ejercicio 6 la pide, y la base 3 también sirve.
- Con la base 4, la demostración no tiene que hablar de acarreos.

# ¿No contradice $O(nW)$?

make bench

| aristas | n | bits del objetivo | objetivo W | n·W | entrada en bits | respuesta | segundos |
|---|---|---|---|---|---|---|---|
| 8 | 16 | 19 | 305834 | 4893344 | 219 | sí | 0.003 |
| 9 | 18 | 21 | 1485482 | 26738676 | 273 | sí | 0.017 |
| 10 | 20 | 23 | 5941930 | 118838600 | 333 | sí | 0.077 |
| 11 | 22 | 25 | 27962026 | 615164572 | 399 | sí | 0.399 |
| 12 | 24 | 27 | 111848106 | 2684354544 | 471 | sí | 1.726 |
| 13 | 26 | 29 | 514501290 | 13377033540 | 549 | sí | 8.650 |

Filas 8 a 13 de la salida. Tiempos de una corrida con `-O2` en el contenedor: dependen de la máquina.

# ¿No contradice $O(nW)$?

```
make bench
```

| aristas | n | bits del objetivo | objetivo W | n·W | entrada en bits | respuesta | segundos |
|---|---|---|---|---|---|---|---|
| 8 | 16 | 19 | 305834 | 4893344 | 219 | sí | 0.003 |
| 9 | 18 | 21 | 1485482 | 26738676 | 273 | sí | 0.017 |
| 10 | 20 | 23 | 5941930 | 118838600 | 333 | sí | 0.077 |
| 11 | 22 | 25 | 27962026 | 615164572 | 399 | sí | 0.399 |
| 12 | 24 | 27 | 111848106 | 2684354544 | 471 | sí | 1.726 |
| 13 | 26 | 29 | 514501290 | 13377033540 | 549 | sí | 8.650 |

Filas 8 a 13 de la salida. Tiempos de una corrida con `-O2` en el contenedor: dependen de la máquina.

- $O(nW)$ es pseudopolinomial: $W$ se escribe en $\log W$ bits.

# ¿No contradice $O(nW)$?

make bench

| aristas | n | bits del objetivo | objetivo W | n·W | entrada en bits | respuesta | segundos |
|---|---|---|---|---|---|---|---|
| 8 | 16 | 19 | 305834 | 4893344 | 219 | sí | 0.003 |
| 9 | 18 | 21 | 1485482 | 26738676 | 273 | sí | 0.017 |
| 10 | 20 | 23 | 5941930 | 118838600 | 333 | sí | 0.077 |
| 11 | 22 | 25 | 27962026 | 615164572 | 399 | sí | 0.399 |
| 12 | 24 | 27 | 111848106 | 2684354544 | 471 | sí | 1.726 |
| 13 | 26 | 29 | 514501290 | 13377033540 | 549 | sí | 8.650 |

Filas 8 a 13 de la salida. Tiempos de una corrida con `-O2` en el contenedor: dependen de la máquina.

- $O(nW)$ es pseudopolinomial: $W$ se escribe en $\log W$ bits.
- De 12 a 13 aristas, $n \cdot W$ y el tiempo se multiplican por 5. La entrada, de 471 a 549 bits.

# ¿No contradice $O(nW)$?

make bench

| aristas | n | bits del objetivo | objetivo W | n·W | entrada en bits | respuesta | segundos |
|---|---|---|---|---|---|---|---|
| 8 | 16 | 19 | 305834 | 4893344 | 219 | sí | 0.003 |
| 9 | 18 | 21 | 1485482 | 26738676 | 273 | sí | 0.017 |
| 10 | 20 | 23 | 5941930 | 118838600 | 333 | sí | 0.077 |
| 11 | 22 | 25 | 27962026 | 615164572 | 399 | sí | 0.399 |
| 12 | 24 | 27 | 111848106 | 2684354544 | 471 | sí | 1.726 |
| 13 | 26 | 29 | 514501290 | 13377033540 | 549 | sí | 8.650 |

Filas 8 a 13 de la salida. Tiempos de una corrida con `-O2` en el contenedor: dependen de la máquina.

- $O(nW)$ es pseudopolinomial: $W$ se escribe en $\log W$ bits.
- De 12 a 13 aristas, $n \cdot W$ y el tiempo se multiplican por 5. La entrada, de 471 a 549 bits.
- **Por qué muchos dígitos:** cada arista necesita su columna, sin mezclarse.

# ¿No contradice $O(nW)$?

make bench

| aristas | n | bits del objetivo | objetivo W | n·W | entrada en bits | respuesta | segundos |
|---|---|---|---|---|---|---|---|
| 8 | 16 | 19 | 305834 | 4893344 | 219 | sí | 0.003 |
| 9 | 18 | 21 | 1485482 | 26738676 | 273 | sí | 0.017 |
| 10 | 20 | 23 | 5941930 | 118838600 | 333 | sí | 0.077 |
| 11 | 22 | 25 | 27962026 | 615164572 | 399 | sí | 0.399 |
| 12 | 24 | 27 | 111848106 | 2684354544 | 471 | sí | 1.726 |
| 13 | 26 | 29 | 514501290 | 13377033540 | 549 | sí | 8.650 |

Filas 8 a 13 de la salida. Tiempos de una corrida con `-O2` en el contenedor: dependen de la máquina.

- $O(nW)$ es pseudopolinomial: $W$ se escribe en $\log W$ bits.
- De 12 a 13 aristas, $n \cdot W$ y el tiempo se multiplican por 5. La entrada, de 471 a 549 bits.
- **Por qué muchos dígitos:** cada arista necesita su columna, sin mezclarse.
- Con $W$ polinomial siempre, la tabla de la semana 7 daría P = NP. En unario, estaría en P.

SECCIÓN 07

# La mochila, y $O(n^5)$

07



# La mochila es NP-completa

**Decisión:** ¿hay objetos con peso a lo sumo $W$ y valor al menos $V$?

# La mochila es NP-completa

**Decisión:** ¿hay objetos con peso a lo sumo $W$ y valor al menos $V$?

- **Está en NP:** los objetos elegidos.

# La mochila es NP-completa

**Decisión:** ¿hay objetos con peso a lo sumo $W$ y valor al menos $V$?

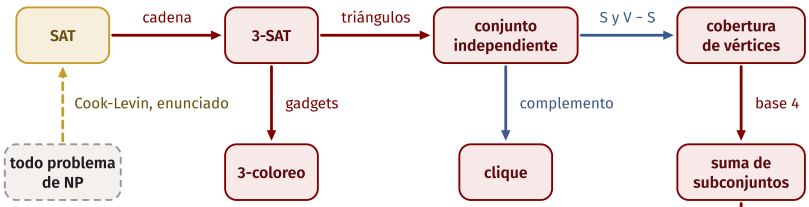
- **Está en NP:** los objetos elegidos.
- **Desde suma de subconjuntos:** peso $= x$, valor $= x$, y $W = V = t$.

# La mochila es NP-completa

**Decisión:** ¿hay objetos con peso a lo sumo $W$ y valor al menos $V$?

- **Está en NP:** los objetos elegidos.
- **Desde suma de subconjuntos:** peso $= x$, valor $= x$, y $W = V = t$.
- **Dos direcciones:** una parte que suma $t$ pesa y vale $t$. Y lo que pesa $\leq t$ y vale $\geq t$ es la misma suma: es $t$.

# La cadena completa, de Cook-Levin a la mochila




Una flecha de A a B dice A $\leq_p$ B: va del conocido al nuevo.

demostrada en este capítulo — — todo NP se reduce a SAT

de la semana 13

Todos están en NP y les llega una cadena desde SAT: por eso son NP-completos.

# Si mañana alguien resuelve 3-SAT en $O(n^5)$

**Conclusión:** P = NP, y la mochila en decisión se resuelve en tiempo polinomial, en bits.

# Si mañana alguien resuelve 3-SAT en $O(n^5)$

**Conclusión:** P = NP, y la mochila en decisión se resuelve en tiempo polinomial, en bits.

**La flecha al revés**

La cadena dice 3-SAT $\leq_p$ mochila. Eso sirve para resolver 3-SAT con una mochila, no al revés.

# Si mañana alguien resuelve 3-SAT en $O(n^5)$

**Conclusión:** P = NP, y la mochila en decisión se resuelve en tiempo polinomial, en bits.

**La flecha al revés**

La cadena dice 3-SAT $\leq_p$ mochila. Eso sirve para resolver 3-SAT con una mochila, no al revés.

**Hace falta mochila $\leq_p$ 3-SAT,** y existe por otra razón:

# Si mañana alguien resuelve 3-SAT en $O(n^5)$

**Conclusión:** P = NP, y la mochila en decisión se resuelve en tiempo polinomial, en bits.

**La flecha al revés**

La cadena dice 3-SAT $\leq_p$ mochila. Eso sirve para resolver 3-SAT con una mochila, no al revés.

**Hace falta mochila $\leq_p$ 3-SAT,** y existe por otra razón:

1. La mochila está en NP.

# Si mañana alguien resuelve 3-SAT en $O(n^5)$

**Conclusión:** P = NP, y la mochila en decisión se resuelve en tiempo polinomial, en bits.

**La flecha al revés**

La cadena dice 3-SAT $\leq_p$ mochila. Eso sirve para resolver 3-SAT con una mochila, no al revés.

**Hace falta mochila $\leq_p$ 3-SAT,** y existe por otra razón:

1. La mochila está en NP.
2. Por Cook-Levin, mochila $\leq_p$ SAT.

# Si mañana alguien resuelve 3-SAT en $O(n^5)$

**Conclusión:** P = NP, y la mochila en decisión se resuelve en tiempo polinomial, en bits.

**La flecha al revés**

La cadena dice 3-SAT $\leq_p$ mochila. Eso sirve para resolver 3-SAT con una mochila, no al revés.

**Hace falta mochila $\leq_p$ 3-SAT,** y existe por otra razón:

1. La mochila está en NP.
2. Por Cook-Levin, mochila $\leq_p$ SAT.
3. Y SAT $\leq_p$ 3-SAT.

# Si mañana alguien resuelve 3-SAT en $O(n^5)$

**Conclusión:** P = NP, y la mochila en decisión se resuelve en tiempo polinomial, en bits.

**La flecha al revés**

La cadena dice 3-SAT $\leq_p$ mochila. Eso sirve para resolver 3-SAT con una mochila, no al revés.

**Hace falta mochila $\leq_p$ 3-SAT,** y existe por otra razón:

1. La mochila está en NP.
2. Por Cook-Levin, mochila $\leq_p$ SAT.
3. Y SAT $\leq_p$ 3-SAT.

Una mochila de $s$ bits da una fórmula de tamaño $p(s)$: costo $O(p(s)^5)$. **Polinomial, pero no $O(s^5)$.** El exponente lo deciden las reducciones.

# De la decisión a la optimización

Búsqueda binaria sobre el valor, con la decisión como caja negra: a lo sumo $\lceil \log_2(\sum v + 1) \rceil$ llamadas.

build/knapsack_demo, extracto

```
objetos (peso, valor): (4, 10) (3, 7) (2, 4) (5, 12) (1, 1)
capacidad 9, suma de valores 34
óptimo por búsqueda binaria: 22, con 5 llamadas a la decisión
óptimo por la tabla de la semana 7: 22
objetos elegidos, con otra llamada por objeto: 0 3
```

# De la decisión a la optimización

Búsqueda binaria sobre el valor, con la decisión como caja negra: a lo sumo $\lceil \log_2(\sum v + 1) \rceil$ llamadas.

build/knapsack_demo, extracto

```
objetos (peso, valor): (4, 10) (3, 7) (2, 4) (5, 12) (1, 1)
capacidad 9, suma de valores 34
óptimo por búsqueda binaria: 22, con 5 llamadas a la decisión
óptimo por la tabla de la semana 7: 22
objetos elegidos, con otra llamada por objeto: 0 3
```

Para saber qué objetos llevar: se pregunta uno por uno si sobra.

# De la decisión a la optimización

Búsqueda binaria sobre el valor, con la decisión como caja negra: a lo sumo $\lceil \log_2(\sum v + 1) \rceil$ llamadas.

build/knapsack_demo, extracto

```
objetos (peso, valor): (4, 10) (3, 7) (2, 4) (5, 12) (1, 1)
capacidad 9, suma de valores 34
óptimo por búsqueda binaria: 22, con 5 llamadas a la decisión
óptimo por la tabla de la semana 7: 22
objetos elegidos, con otra llamada por objeto: 0 3
```

Para saber qué objetos llevar: se pregunta uno por uno si sobra.

$\sum v$ se escribe en pocos bits: las llamadas no pasan del tamaño de la entrada, salvo una constante.

SECCIÓN 08

# Un mapa de problemas y la frontera

08

# ¿Desde cuál reducir?

Empiecen por el que tenga la misma forma: las piezas de la transformación salen solas.

| Si el problema se trata de… | Reduzcan desde |
|---|---|

# ¿Desde cuál reducir?

Empiecen por el que tenga la misma forma: las piezas de la transformación salen solas.

| Si el problema se trata de… | Reduzcan desde |
|---|---|
| Decisiones sí o no con reglas lógicas | 3-SAT |

# ¿Desde cuál reducir?

Empiecen por el que tenga la misma forma: las piezas de la transformación salen solas.

| Si el problema se trata de… | Reduzcan desde |
|---|---|
| Decisiones sí o no con reglas lógicas | 3-SAT |
| Elegir mucho sin conflictos | Conjunto independiente |

# ¿Desde cuál reducir?

Empiecen por el que tenga la misma forma: las piezas de la transformación salen solas.

| Si el problema se trata de… | Reduzcan desde |
|---|---|
| Decisiones sí o no con reglas lógicas | 3-SAT |
| Elegir mucho sin conflictos | Conjunto independiente |
| Elegir un grupo donde todos encajen | Clique |

# ¿Desde cuál reducir?

Empiecen por el que tenga la misma forma: las piezas de la transformación salen solas.

| Si el problema se trata de… | Reduzcan desde |
|---|---|
| Decisiones sí o no con reglas lógicas | 3-SAT |
| Elegir mucho sin conflictos | Conjunto independiente |
| Elegir un grupo donde todos encajen | Clique |
| Elegir poco que toque todo | Cobertura de vértices o de conjuntos |

# ¿Desde cuál reducir?

Empiecen por el que tenga la misma forma: las piezas de la transformación salen solas.

| Si el problema se trata de… | Reduzcan desde |
|---|---|
| Decisiones sí o no con reglas lógicas | 3-SAT |
| Elegir mucho sin conflictos | Conjunto independiente |
| Elegir un grupo donde todos encajen | Clique |
| Elegir poco que toque todo | Cobertura de vértices o de conjuntos |
| Repartir en grupos con incompatibilidades | 3-coloreo |

# ¿Desde cuál reducir?

Empiecen por el que tenga la misma forma: las piezas de la transformación salen solas.

| Si el problema se trata de… | Reduzcan desde |
|---|---|
| Decisiones sí o no con reglas lógicas | 3-SAT |
| Elegir mucho sin conflictos | Conjunto independiente |
| Elegir un grupo donde todos encajen | Clique |
| Elegir poco que toque todo | Cobertura de vértices o de conjuntos |
| Repartir en grupos con incompatibilidades | 3-coloreo |
| Encontrar un orden o un recorrido | Ciclo o camino hamiltoniano |

# ¿Desde cuál reducir?

Empiecen por el que tenga la misma forma: las piezas de la transformación salen solas.

| Si el problema se trata de… | Reduzcan desde |
|---|---|
| Decisiones sí o no con reglas lógicas | 3-SAT |
| Elegir mucho sin conflictos | Conjunto independiente |
| Elegir un grupo donde todos encajen | Clique |
| Elegir poco que toque todo | Cobertura de vértices o de conjuntos |
| Repartir en grupos con incompatibilidades | 3-coloreo |
| Encontrar un orden o un recorrido | Ciclo o camino hamiltoniano |
| Llegar a un total exacto con números | Suma de subconjuntos, partición |

# ¿Desde cuál reducir?

Empiecen por el que tenga la misma forma: las piezas de la transformación salen solas.

| Si el problema se trata de… | Reduzcan desde |
|---|---|
| Decisiones sí o no con reglas lógicas | 3-SAT |
| Elegir mucho sin conflictos | Conjunto independiente |
| Elegir un grupo donde todos encajen | Clique |
| Elegir poco que toque todo | Cobertura de vértices o de conjuntos |
| Repartir en grupos con incompatibilidades | 3-coloreo |
| Encontrar un orden o un recorrido | Ciclo o camino hamiltoniano |
| Llegar a un total exacto con números | Suma de subconjuntos, partición |

Partición y cobertura de conjuntos son ejercicios. Ciclo y camino hamiltoniano están citados, sin demostrar: CLRS demuestra el ciclo.

# Cuatro pares

| Fácil | Por qué | Difícil |
|---|---|---|

# Cuatro pares

| Fácil | Por qué | Difícil |
|---|---|---|
| 2-SAT | Son implicaciones: componentes | 3-SAT |

# Cuatro pares

| Fácil | Por qué | Difícil |
|---|---|---|
| 2-SAT | Son implicaciones: componentes | 3-SAT |
| 2-coloreo | Un color obliga a los vecinos: BFS | 3-coloreo |

# Cuatro pares

| Fácil | Por qué | Difícil |
|---|---|---|
| 2-SAT | Son implicaciones: componentes | 3-SAT |
| 2-coloreo | Un color obliga a los vecinos: BFS | 3-coloreo |
| Ciclo euleriano | Grados pares y conexión | Ciclo hamiltoniano |

# Cuatro pares

| Fácil | Por qué | Difícil |
|---|---|---|
| 2-SAT | Son implicaciones: componentes | 3-SAT |
| 2-coloreo | Un color obliga a los vecinos: BFS | 3-coloreo |
| Ciclo euleriano | Grados pares y conexión | Ciclo hamiltoniano |
| Camino más largo en un DAG | Orden topológico | En general |

# Cuatro pares

| Fácil | Por qué | Difícil |
|---|---|---|
| 2-SAT | Son implicaciones: componentes | 3-SAT |
| 2-coloreo | Un color obliga a los vecinos: BFS | 3-coloreo |
| Ciclo euleriano | Grados pares y conexión | Ciclo hamiltoniano |
| Camino más largo en un DAG | Orden topológico | En general |

**Qué tienen en común**

En los tres primeros, una decisión obliga a otra, o la condición es casi local. En el cuarto, lo ya usado no se puede volver a alcanzar. Los problemas difíciles no ofrecen esa ventaja, que sepamos.

# 2-SAT: el grafo de implicaciones

$(a \vee b)$ dice: si $a$ es falso, $b$ es verdadero. Y si $b$ es falso, $a$ es verdadero.

# 2-SAT: el grafo de implicaciones

$(a \vee b)$ dice: si $a$ es falso, $b$ es verdadero. Y si $b$ es falso, $a$ es verdadero.

**Grafo de implicaciones** ***(implication graph)***

$2n$ nodos, uno por literal. $(a \vee b)$ aporta $\neg a \to b$ y $\neg b \to a$.

# 2-SAT: el grafo de implicaciones

$(a \vee b)$ dice: si $a$ es falso, $b$ es verdadero. Y si $b$ es falso, $a$ es verdadero.

**Grafo de implicaciones** ***(implication graph)***

$2n$ nodos, uno por literal. $(a \vee b)$ aporta $\neg a \to b$ y $\neg b \to a$.

Una asignación que cumple la fórmula no tiene un camino de un literal verdadero a uno falso.

# 2-SAT: el grafo de implicaciones

$(a \vee b)$ dice: si $a$ es falso, $b$ es verdadero. Y si $b$ es falso, $a$ es verdadero.

**Grafo de implicaciones** *(implication graph)*

$2n$ nodos, uno por literal. $(a \vee b)$ aporta $\neg a \to b$ y $\neg b \to a$.

Una asignación que cumple la fórmula no tiene un camino de un literal verdadero a uno falso.

**Cláusulas raras:**

- $(x \vee x)$: dos veces $\neg x \to x$. Obliga a $x$.

# 2-SAT: el grafo de implicaciones

$(a \vee b)$ dice: si $a$ es falso, $b$ es verdadero. Y si $b$ es falso, $a$ es verdadero.

**Grafo de implicaciones** *(implication graph)*

$2n$ nodos, uno por literal. $(a \vee b)$ aporta $\neg a \to b$ y $\neg b \to a$.

Una asignación que cumple la fórmula no tiene un camino de un literal verdadero a uno falso.

**Cláusulas raras:**

- $(x \vee x)$: dos veces $\neg x \to x$. Obliga a $x$.
- $(x \vee \neg x)$: lazos $x \to x$ y $\neg x \to \neg x$. No dice nada.

# 2-SAT: el grafo de implicaciones

$(a \vee b)$ dice: si $a$ es falso, $b$ es verdadero. Y si $b$ es falso, $a$ es verdadero.

**Grafo de implicaciones** *(implication graph)*

$2n$ nodos, uno por literal. $(a \vee b)$ aporta $\neg a \to b$ y $\neg b \to a$.

Una asignación que cumple la fórmula no tiene un camino de un literal verdadero a uno falso.

**Cláusulas raras:**

- $(x \vee x)$: dos veces $\neg x \to x$. Obliga a $x$.
- $(x \vee \neg x)$: lazos $x \to x$ y $\neg x \to \neg x$. No dice nada.

Con tres literales no pasa: si $a$ es falso, todavía quedan dos opciones.

# 2-SAT: el algoritmo

pseudocódigo

```
algoritmo dos_sat(formula)
    construir el grafo de implicaciones
    comp ← componentes(grafo)
    para cada variable x hacer
        si comp[x] = comp[no x] entonces
            devolver "no se puede"
        fin si
        valor[x] ← comp[x] < comp[no x]
    fin para
    devolver valor
fin algoritmo
```

# 2-SAT: el algoritmo

pseudocódigo

```
algoritmo dos_sat(formula)
    construir el grafo de implicaciones
    comp ← componentes(grafo)
    para cada variable x hacer
        si comp[x] = comp[no x] entonces
            devolver "no se puede"
        fin si
        valor[x] ← comp[x] < comp[no x]
    fin para
    devolver valor
fin algoritmo
```

- `componentes` es Kosaraju-Sharir de la semana 6: salen **de sumidero a fuente**.

# 2-SAT: el algoritmo

pseudocódigo

```
algoritmo dos_sat(formula)
    construir el grafo de implicaciones
    comp ← componentes(grafo)
    para cada variable x hacer
        si comp[x] = comp[no x] entonces
            devolver "no se puede"
        fin si
        valor[x] ← comp[x] < comp[no x]
    fin para
    devolver valor
fin algoritmo
```

- `componentes` es Kosaraju-Sharir de la semana 6: salen **de sumidero a fuente**.
- Numeradas así, toda arista $u \to v$ cumple comp[$u$] $\geq$ comp[$v$].

# 2-SAT: el algoritmo

pseudocódigo

```
algoritmo dos_sat(formula)
    construir el grafo de implicaciones
    comp ← componentes(grafo)
    para cada variable x hacer
        si comp[x] = comp[no x] entonces
            devolver "no se puede"
        fin si
        valor[x] ← comp[x] < comp[no x]
    fin para
    devolver valor
fin algoritmo
```

- `componentes` es Kosaraju-Sharir de la semana 6: salen **de sumidero a fuente**.
- Numeradas así, toda arista $u \to v$ cumple $\text{comp}[u] \geq \text{comp}[v]$.
- **Costo:** $2n$ nodos y $2m$ aristas: $\Theta(n + m)$.

# 2-SAT: la demostración

**Si $x$ y $\neg x$ están en la misma componente, no hay solución.**

# 2-SAT: la demostración

**Si $x$ y $\neg x$ están en la misma componente, no hay solución.**

- Hay caminos $x \rightsquigarrow \neg x$ y $\neg x \rightsquigarrow x$. Uno de los dos va de verdadero a falso.

# 2-SAT: la demostración

**Si $x$ y $\neg x$ están en la misma componente, no hay solución.**

- Hay caminos $x \rightsquigarrow \neg x$ y $\neg x \rightsquigarrow x$. Uno de los dos va de verdadero a falso.

**Si no, la asignación funciona.** Un literal $\ell$ es verdadero si y solo si $\text{comp}[\ell] < \text{comp}[\neg\ell]$.

# 2-SAT: la demostración

**Si $x$ y $\neg x$ están en la misma componente, no hay solución.**

- Hay caminos $x \rightsquigarrow \neg x$ y $\neg x \rightsquigarrow x$. Uno de los dos va de verdadero a falso.

**Si no, la asignación funciona.** Un literal $\ell$ es verdadero si y solo si $\text{comp}[\ell] < \text{comp}[\neg\ell]$.

- Supongamos $(a \vee b)$ con $a$ y $b$ falsos: $\text{comp}[a] > \text{comp}[\neg a]$ y $\text{comp}[b] > \text{comp}[\neg b]$.

# 2-SAT: la demostración

**Si $x$ y $\neg x$ están en la misma componente, no hay solución.**

- Hay caminos $x \rightsquigarrow \neg x$ y $\neg x \rightsquigarrow x$. Uno de los dos va de verdadero a falso.

**Si no, la asignación funciona.** Un literal $\ell$ es verdadero si y solo si $\text{comp}[\ell] < \text{comp}[\neg \ell]$.

- Supongamos $(a \vee b)$ con $a$ y $b$ falsos: $\text{comp}[a] > \text{comp}[\neg a]$ y $\text{comp}[b] > \text{comp}[\neg b]$.
- Las aristas $\neg a \to b$ y $\neg b \to a$ dan $\text{comp}[\neg a] \geq \text{comp}[b]$ y $\text{comp}[\neg b] \geq \text{comp}[a]$.

# 2-SAT: la demostración

**Si $x$ y $\neg x$ están en la misma componente, no hay solución.**

- Hay caminos $x \rightsquigarrow \neg x$ y $\neg x \rightsquigarrow x$. Uno de los dos va de verdadero a falso.

**Si no, la asignación funciona.** Un literal $\ell$ es verdadero si y solo si $\text{comp}[\ell] < \text{comp}[\neg\ell]$.

- Supongamos $(a \vee b)$ con $a$ y $b$ falsos: $\text{comp}[a] > \text{comp}[\neg a]$ y $\text{comp}[b] > \text{comp}[\neg b]$.
- Las aristas $\neg a \to b$ y $\neg b \to a$ dan $\text{comp}[\neg a] \geq \text{comp}[b]$ y $\text{comp}[\neg b] \geq \text{comp}[a]$.

$$\text{comp}[a] > \text{comp}[\neg a] \geq \text{comp}[b] > \text{comp}[\neg b] \geq \text{comp}[a]$$

# 2-SAT: la demostración

**Si $x$ y $\neg x$ están en la misma componente, no hay solución.**

- Hay caminos $x \rightsquigarrow \neg x$ y $\neg x \rightsquigarrow x$. Uno de los dos va de verdadero a falso.

**Si no, la asignación funciona.** Un literal $\ell$ es verdadero si y solo si $\text{comp}[\ell] < \text{comp}[\neg\ell]$.

- Supongamos $(a \vee b)$ con $a$ y $b$ falsos: $\text{comp}[a] > \text{comp}[\neg a]$ y $\text{comp}[b] > \text{comp}[\neg b]$.
- Las aristas $\neg a \to b$ y $\neg b \to a$ dan $\text{comp}[\neg a] \geq \text{comp}[b]$ y $\text{comp}[\neg b] \geq \text{comp}[a]$.

$$\text{comp}[a] > \text{comp}[\neg a] \geq \text{comp}[b] > \text{comp}[\neg b] \geq \text{comp}[a]$$

Contradicción. También cubre $a = b$ y $b = \neg a$. ■

# 2-SAT: la demostración

**Si $x$ y $\neg x$ están en la misma componente, no hay solución.**

- Hay caminos $x \rightsquigarrow \neg x$ y $\neg x \rightsquigarrow x$. Uno de los dos va de verdadero a falso.

**Si no, la asignación funciona.** Un literal $\ell$ es verdadero si y solo si $\text{comp}[\ell] < \text{comp}[\neg\ell]$.

- Supongamos $(a \vee b)$ con $a$ y $b$ falsos: $\text{comp}[a] > \text{comp}[\neg a]$ y $\text{comp}[b] > \text{comp}[\neg b]$.
- Las aristas $\neg a \to b$ y $\neg b \to a$ dan $\text{comp}[\neg a] \geq \text{comp}[b]$ y $\text{comp}[\neg b] \geq \text{comp}[a]$.

$$\text{comp}[a] > \text{comp}[\neg a] \geq \text{comp}[b] > \text{comp}[\neg b] \geq \text{comp}[a]$$

Contradicción. También cubre $a = b$ y $b = \neg a$. ■

**Intuición**

Lo verdadero arrastra todo lo que alcanza. Conviene que esté cerca de los sumideros.

# 2-SAT: corre

build/two_sat_demo, extracto

```
(x1 ∨ x2) ∧ (¬x1 ∨ x3) ∧ (¬x2 ∨ ¬x3) ∧ (¬x3 ∨ ¬x1)
  aristas: x1→x3 x1→¬x3 ¬x1→x2 x2→¬x3 ¬x2→x1 x3→¬x2 x3→¬x1 ¬x3→¬x1
  componentes, en el orden en que salen (de sumidero a fuente): {¬x1 x2 ¬x3} {x1 ¬x2 x3}
  x verdadera si comp[x] < comp[¬x]: x1=F x2=V x3=F, ¿cumple? sí
  con el orden al revés:            x1=V x2=F x3=V, ¿cumple? no

(x1 ∨ x2) ∧ (x1 ∨ ¬x2) ∧ (¬x1 ∨ x2) ∧ (¬x1 ∨ ¬x2)
  componentes, en el orden en que salen (de sumidero a fuente): {x1 ¬x1 x2 ¬x2}
  insatisfacible; fuerza bruta: insatisfacible
```

# 2-SAT: corre

build/two_sat_demo, extracto

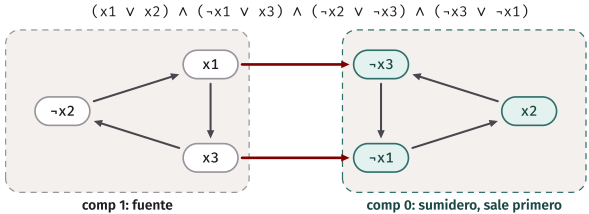
```
(x1 ∨ x2) ∧ (¬x1 ∨ x3) ∧ (¬x2 ∨ ¬x3) ∧ (¬x3 ∨ ¬x1)
  aristas: x1→x3 x1→¬x3 ¬x1→x2 x2→¬x3 ¬x2→x1 x3→¬x2 x3→¬x1 ¬x3→¬x1
  componentes, en el orden en que salen (de sumidero a fuente): {¬x1 x2 ¬x3} {x1 ¬x2 x3}
  x verdadera si comp[x] < comp[¬x]: x1=F x2=V x3=F, ¿cumple? sí
  con el orden al revés:             x1=V x2=F x3=V, ¿cumple? no

(x1 ∨ x2) ∧ (x1 ∨ ¬x2) ∧ (¬x1 ∨ x2) ∧ (¬x1 ∨ ¬x2)
  componentes, en el orden en que salen (de sumidero a fuente): {x1 ¬x1 x2 ¬x2}
  insatisfacible; fuerza bruta: insatisfacible
```

**Error común**

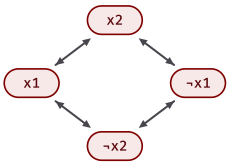
Tomar la regla de memoria. En la versión de CLRS las componentes salen al revés, y la comparación se invierte. Gana el literal más cercano a los sumideros.

# 2-SAT: las componentes, dibujadas




De la componente 1 a la 0: x1 → ¬x3 y x3 → ¬x1

En verde, lo verdadero: $\ell$ lo es si comp[$\ell$] < comp[$\neg\ell$]. Da x1 = F, x2 = V, x3 = F, y las cuatro cláusulas se cumplen.




(x1 ∨ x2) ∧ (x1 ∨ ¬x2) ∧ (¬x1 ∨ x2) ∧ (¬x1 ∨ ¬x2)

Una sola componente: {x1, ¬x1, x2, ¬x2}.

x1 y ¬x1 están en la misma, y x2 y ¬x2 también.

Hay un camino de x a ¬x y otro de ¬x a x: sea cual sea el valor de x, uno de los dos va de verdadero a falso. Insatisfacible.

# 2-coloreo, Euler y el camino más largo

**2-coloreo.** El primer color obliga a los vecinos, y así hasta el final. Un BFS falla si y solo si hay un ciclo impar. Con tres colores, el vecino aún tiene dos opciones.

# 2-coloreo, Euler y el camino más largo

**2-coloreo.** El primer color obliga a los vecinos, y así hasta el final. Un BFS falla si y solo si hay un ciclo impar. Con tres colores, el vecino aún tiene dos opciones.

**Ciclo euleriano:** usa *cada arista* una vez. Existe si y solo si todos los grados son pares y las aristas están en una sola componente.

# 2-coloreo, Euler y el camino más largo

**2-coloreo.** El primer color obliga a los vecinos, y así hasta el final. Un BFS falla si y solo si hay un ciclo impar. Con tres colores, el vecino aún tiene dos opciones.

**Ciclo euleriano:** usa *cada arista* una vez. Existe si y solo si todos los grados son pares y las aristas están en una sola componente.

- Necesario: cada entrada tiene su salida, y un ciclo no salta de componente.

# 2-coloreo, Euler y el camino más largo

**2-coloreo.** El primer color obliga a los vecinos, y así hasta el final. Un BFS falla si y solo si hay un ciclo impar. Con tres colores, el vecino aún tiene dos opciones.

**Ciclo euleriano:** usa *cada arista* una vez. Existe si y solo si todos los grados son pares y las aristas están en una sola componente.

- Necesario: cada entrada tiene su salida, y un ciclo no salta de componente.
- Suficiente: se camina hasta atascarse, que solo puede ser al inicio, y se empalman ciclos.

# 2-coloreo, Euler y el camino más largo

**2-coloreo.** El primer color obliga a los vecinos, y así hasta el final. Un BFS falla si y solo si hay un ciclo impar. Con tres colores, el vecino aún tiene dos opciones.

**Ciclo euleriano:** usa *cada arista* una vez. Existe si y solo si todos los grados son pares y las aristas están en una sola componente.

- Necesario: cada entrada tiene su salida, y un ciclo no salta de componente.
- Suficiente: se camina hasta atascarse, que solo puede ser al inicio, y se empalman ciclos.

**Ciclo hamiltoniano:** usa *cada nodo* una vez. NP-completo.

# 2-coloreo, Euler y el camino más largo

**2-coloreo.** El primer color obliga a los vecinos, y así hasta el final. Un BFS falla si y solo si hay un ciclo impar. Con tres colores, el vecino aún tiene dos opciones.

**Ciclo euleriano:** usa *cada arista* una vez. Existe si y solo si todos los grados son pares y las aristas están en una sola componente.

- Necesario: cada entrada tiene su salida, y un ciclo no salta de componente.
- Suficiente: se camina hasta atascarse, que solo puede ser al inicio, y se empalman ciclos.

**Ciclo hamiltoniano:** usa *cada nodo* una vez. NP-completo.

**Camino más largo.** En un DAG, sus sucesores se calculan primero. En general, se pide un camino simple: tiene $V - 1$ aristas si y solo si es hamiltoniano. El subproblema tendría que recordar qué nodos ya se usaron.

# Pregunta de repaso 4

| Problema | Por qué |
|---|---|

build/frontier_demo, extracto

```
¿Ciclo que use cada arista exactamente una vez?
  dos triángulos con un nodo común: grados y conexión sí, fuerza bruta sí
  dos triángulos separados        : grados y conexión no, fuerza bruta no
  triángulo con una cola          : grados y conexión no, fuerza bruta no
```

# Pregunta de repaso 4

| Problema | | Por qué |
|---|---|---|
| Camino más largo en un DAG | fácil | programación dinámica |

build/frontier_demo, extracto

```
¿Ciclo que use cada arista exactamente una vez?
  dos triángulos con un nodo común: grados y conexión sí, fuerza bruta sí
  dos triángulos separados        : grados y conexión no, fuerza bruta no
  triángulo con una cola          : grados y conexión no, fuerza bruta no
```

# Pregunta de repaso 4

| Problema | | Por qué |
|---|---|---|
| Camino más largo en un DAG | fácil | programación dinámica |
| 3-coloreo | difícil | NP-completo |

build/frontier_demo, extracto

```
¿Ciclo que use cada arista exactamente una vez?
  dos triángulos con un nodo común: grados y conexión sí, fuerza bruta sí
  dos triángulos separados        : grados y conexión no, fuerza bruta no
  triángulo con una cola          : grados y conexión no, fuerza bruta no
```

# Pregunta de repaso 4

| Problema | | Por qué |
|---|---|---|
| Camino más largo en un DAG | fácil | programación dinámica |
| 3-coloreo | difícil | NP-completo |
| ¿Ciclo que use cada arista exactamente una vez? | fácil | es Euler: lean “arista” |

build/frontier_demo, extracto

```
¿Ciclo que use cada arista exactamente una vez?
  dos triángulos con un nodo común: grados y conexión sí, fuerza bruta sí
  dos triángulos separados        : grados y conexión no, fuerza bruta no
  triángulo con una cola          : grados y conexión no, fuerza bruta no
```

# Pregunta de repaso 4

| Problema | | Por qué |
|---|---|---|
| Camino más largo en un DAG | fácil | programación dinámica |
| 3-coloreo | difícil | NP-completo |
| ¿Ciclo que use cada arista exactamente una vez? | fácil | es Euler: lean “arista” |
| 2-SAT | fácil | componentes |

build/frontier_demo, extracto

```
¿Ciclo que use cada arista exactamente una vez?
  dos triángulos con un nodo común: grados y conexión sí, fuerza bruta sí
  dos triángulos separados        : grados y conexión no, fuerza bruta no
  triángulo con una cola          : grados y conexión no, fuerza bruta no
```

# Pregunta de repaso 4

| Problema | | Por qué |
|---|---|---|
| Camino más largo en un DAG | fácil | programación dinámica |
| 3-coloreo | difícil | NP-completo |
| ¿Ciclo que use cada arista exactamente una vez? | fácil | es Euler: lean “arista” |
| 2-SAT | fácil | componentes |

build/frontier_demo, extracto

```
¿Ciclo que use cada arista exactamente una vez?
  dos triángulos con un nodo común: grados y conexión sí, fuerza bruta sí
  dos triángulos separados        : grados y conexión no, fuerza bruta no
  triángulo con una cola          : grados y conexión no, fuerza bruta no
```

Dos triángulos separados tienen todos los grados pares: la conexión es lo que se olvida.

SECCIÓN 09

# NP-completo no quiere decir irresoluble

09

# Sí se puede resolver

make solver, extracto. Tiempos de minisat: dependen de la máquina

```
 variables cláusulas  semilla        resultado     segundos
       300       1278        2   insatisfacible      9.69586

El principio del palomar: una paloma más que huecos.
    huecos  variables cláusulas        resultado     segundos
         9         90        415   insatisfacible      2.38231
        10        110        561     más de 30 s            -
```

# Sí se puede resolver

make solver, extracto. Tiempos de minisat: dependen de la máquina

```
 variables cláusulas  semilla        resultado     segundos
       300       1278        2   insatisfacible      9.69586

El principio del palomar: una paloma más que huecos.
    huecos  variables cláusulas        resultado     segundos
         9         90        415   insatisfacible      2.38231
        10        110        561     más de 30 s            -
```

Segundos con $2^{300}$ asignaciones. Más de 30 s con 110 variables del palomar.

# Sí se puede resolver

make solver, extracto. Tiempos de minisat: dependen de la máquina

```
 variables cláusulas  semilla        resultado     segundos
       300       1278        2   insatisfacible      9.69586

El principio del palomar: una paloma más que huecos.
    huecos  variables cláusulas        resultado     segundos
         9         90        415   insatisfacible      2.38231
        10        110        561     más de 30 s             -
```

Segundos con $2^{300}$ asignaciones. Más de 30 s con 110 variables del palomar.

**Error común**

“Es NP-completo, así que no se puede resolver.” Sí se puede: lo difícil es garantizar que sea rápido en todos los casos.

SECCIÓN 10

# Cierre

10

# Lo que hay que llevarse

- **NP-completo:** en NP y NP-difícil. Si uno cae, caen todos. El problema de la parada es NP-difícil y no es NP-completo.
- **La receta:** está en NP, elegir, transformar del conocido al nuevo, dos direcciones, tiempo polinomial.
- **SAT a 3-SAT:** cadena con "está más adelante". Equisatisfactible.
- **3-SAT a independiente:** un triángulo por cláusula, aristas entre opuestos, $k = m$.
- **3-SAT a 3-coloreo:** paleta, variable y dos compuertas O por cláusula.
- **Cobertura a suma:** base 4, un dígito por arista. Tres 1 por columna: sin acarreos.
- $O(nW)$**:** pseudopolinomial. No contradice nada.
- **3-SAT en** $O(n^5)$**:** P = NP. La mochila, por Cook-Levin. El exponente no tiene por qué ser 5.
- **Frontera:** 2-SAT, 2-coloreo, Euler y el DAG son fáciles, cada uno por su razón.
- **NP-completo no es irresoluble:** falta la garantía, no el algoritmo.

# Para la próxima clase

- Leer los apuntes de la semana 14, y hacer sus ejercicios. La semana 15: qué hacer con un problema difícil.
- **Práctica 4** en la sesión de laboratorio. Opcional: `lab/two_sat`, asignar turnos cumpliendo pedidos. Lean los límites.

**Preguntas**

¿Qué dos cosas hay que demostrar para probar que un problema es NP-completo?
Corran la reducción de 3-SAT a conjunto independiente con una fórmula de dos cláusulas.
¿Por qué la reducción a suma de subconjuntos necesita números con muchos dígitos?
Clasifiquen como fácil o difícil: camino más largo en un DAG; 3-coloreo; ¿tiene el grafo un ciclo que use cada arista exactamente una vez?; 2-SAT.
Si mañana alguien resuelve 3-SAT en $O(n^5)$, ¿qué pasa con la mochila?