

UNIVERSIDAD NACIONAL DE INGENIERÍA

Análisis y Diseño de Algoritmos

P, NP y reducciones

Conjunto independiente, cobertura de vértices y SAT

Gerson Garrido

Semana 13

Hoy

Sesión A

1. Problemas parecidos, destinos distintos
2. Decisión y bits
3. P y NP
4. P contra NP
5. Reducciones polinomiales
6. La dirección correcta

Sesión B

7. Conjunto independiente y cobertura
8. Conjunto independiente y clique
9. El patrón de una reducción
10. SAT, CNF y un solucionador
11. Codificar: coloreo en DIMACS
12. Al menos uno y a lo sumo uno

> **Lectura**
>
> Apuntes de la semana 13. Código en `code/week-13`. Laboratorio en `lab/sudoku_sat`.

SECCIÓN 01

Problemas parecidos

01

Las reducciones cambian de oficio

La semana pasada: reducir para **resolver**. Un problema se convertía en un flujo.

Las reducciones cambian de oficio

La semana pasada: reducir para **resolver**. Un problema se convertía en un flujo.

Esta semana: hay problemas que nadie sabe resolver rápido.

Las reducciones cambian de oficio

La semana pasada: reducir para **resolver**. Un problema se convertía en un flujo.

Esta semana: hay problemas que nadie sabe resolver rápido.

- Nadie ha demostrado que sea imposible.

Las reducciones cambian de oficio

La semana pasada: reducir para **resolver**. Un problema se convertía en un flujo.

Esta semana: hay problemas que nadie sabe resolver rápido.

- Nadie ha demostrado que sea imposible.
- Tampoco hay un algoritmo.

Las reducciones cambian de oficio

La semana pasada: reducir para **resolver**. Un problema se convertía en un flujo.

Esta semana: hay problemas que nadie sabe resolver rápido.

- Nadie ha demostrado que sea imposible.
- Tampoco hay un algoritmo.
- Lo que hay es una red de reducciones entre ellos.

Las reducciones cambian de oficio

La semana pasada: reducir para **resolver**. Un problema se convertía en un flujo.

Esta semana: hay problemas que nadie sabe resolver rápido.

- Nadie ha demostrado que sea imposible.
- Tampoco hay un algoritmo.
- Lo que hay es una red de reducciones entre ellos.

> **Lo que vamos a decir**
>
> Son, en un sentido preciso, el mismo problema: quien resuelva uno rápido los resuelve a todos.

Cambiar una palabra

Camino mínimo

- Dijkstra, en tiempo polinomial.

Camino simple más largo

- Ningún algoritmo polinomial conocido.

2-coloreo

- BFS, en $O(V + E)$.

3-coloreo

- Ningún algoritmo polinomial conocido.

Cambiar una palabra

Camino mínimo

- Dijkstra, en tiempo polinomial.
- Un tramo de un camino mínimo también es mínimo.

2-coloreo

- BFS, en $O(V + E)$.

Camino simple más largo

- Ningún algoritmo polinomial conocido.
- Un tramo no tiene por qué serlo: hay que recordar qué nodos se usaron, como Held-Karp.

3-coloreo

- Ningún algoritmo polinomial conocido.

Cambiar una palabra

Camino mínimo

- Dijkstra, en tiempo polinomial.
- Un tramo de un camino mínimo también es mínimo.

2-coloreo

- BFS, en $O(V + E)$.

Camino simple más largo

- Ningún algoritmo polinomial conocido.
- Un tramo no tiene por qué serlo: hay que recordar qué nodos se usaron, como Held-Karp.

3-coloreo

- Ningún algoritmo polinomial conocido.

Cambiar una palabra

Camino mínimo

- Dijkstra, en tiempo polinomial.
- Un tramo de un camino mínimo también es mínimo.

2-coloreo

- BFS, en $O(V + E)$.
- Pintar un nodo obliga a cada vecino.

Camino simple más largo

- Ningún algoritmo polinomial conocido.
- Un tramo no tiene por qué serlo: hay que recordar qué nodos se usaron, como Held-Karp.

3-coloreo

- Ningún algoritmo polinomial conocido.
- A cada vecino todavía le quedan dos opciones.

Cambiar una palabra

Camino mínimo

- Dijkstra, en tiempo polinomial.
- Un tramo de un camino mínimo también es mínimo.

2-coloreo

- BFS, en $O(V + E)$.
- Pintar un nodo obliga a cada vecino.

Camino simple más largo

- Ningún algoritmo polinomial conocido.
- Un tramo no tiene por qué serlo: hay que recordar qué nodos se usaron, como Held-Karp.

3-coloreo

- Ningún algoritmo polinomial conocido.
- A cada vecino todavía le quedan dos opciones.

“No conozco un algoritmo rápido” no es lo mismo que “no existe”.

SECCIÓN 02

Decisión y bits

02

Problemas de decisión

Problema de decisión *(decision problem)*

Su respuesta es sí o no. Cada entrada es una **instancia**.

Optimización	**Decisión**
Conjunto independiente máximo	¿Hay uno de tamaño al menos k?
Cobertura de vértices mínima	¿Hay una de tamaño a lo sumo k?
Mínimo de colores	¿Se colorea con k colores?
Mochila de mayor valor	¿Peso $\leq W$ y valor $\geq V$?

Problemas de decisión

Problema de decisión *(decision problem)*

Su respuesta es sí o no. Cada entrada es una **instancia**.

Optimización	**Decisión**
Conjunto independiente máximo	¿Hay uno de tamaño al menos k?
Cobertura de vértices mínima	¿Hay una de tamaño a lo sumo k?
Mínimo de colores	¿Se colorea con k colores?
Mochila de mayor valor	¿Peso $\leq W$ y valor $\geq V$?

No se pierde nada aquí: búsqueda binaria sobre k da el óptimo con $O(\log n)$ preguntas.

El tamaño se mide en bits

“Polinomial” en qué: en b, los bits de la entrada.

El tamaño se mide en bits

"Polinomial" en qué: en b, los bits de la entrada.

- Un grafo sin números grandes: polinomial en $V + E$ es polinomial en b.

El tamaño se mide en bits

"Polinomial" en qué: en b, los bits de la entrada.

- Un grafo sin números grandes: polinomial en $V + E$ es polinomial en b.
- El número W ocupa $\log W$ bits. $O(nW)$ es polinomial en el **valor** de W, no en sus bits.

El tamaño se mide en bits

"Polinomial" en qué: en b, los bits de la entrada.

- Un grafo sin números grandes: polinomial en $V + E$ es polinomial en b.
- El número W ocupa $\log W$ bits. $O(nW)$ es polinomial en el **valor** de W, no en sus bits.
- Con números de c bits: $b \approx n \cdot c$, y el trabajo puede llegar a $n \cdot 2^c$.

El tamaño se mide en bits

"Polinomial" en qué: en b, los bits de la entrada.

- Un grafo sin números grandes: polinomial en $V + E$ es polinomial en b.
- El número W ocupa $\log W$ bits. $O(nW)$ es polinomial en el **valor** de W, no en sus bits.
- Con números de c bits: $b \approx n \cdot c$, y el trabajo puede llegar a $n \cdot 2^c$.
- Probar si N es primo con divisores hasta $\sqrt{N}$ cuesta $2^{b/2}$.

El tamaño se mide en bits

"Polinomial" en qué: en b, los bits de la entrada.

- Un grafo sin números grandes: polinomial en $V + E$ es polinomial en b.
- El número W ocupa $\log W$ bits. $O(nW)$ es polinomial en el **valor** de W, no en sus bits.
- Con números de c bits: $b \approx n \cdot c$, y el trabajo puede llegar a $n \cdot 2^c$.
- Probar si N es primo con divisores hasta $\sqrt{N}$ cuesta $2^{b/2}$.

Error común

Decir que algo es polinomial porque la fórmula no tiene exponentes, sin mirar si sus símbolos son cantidades de objetos o valores de números.

SECCIÓN 03

P y NP

03



La clase P

P

Problemas de decisión con un algoritmo correcto para toda instancia, que corre en $O(b^c)$ para alguna constante c.

¿Hay camino de s a t? ¿Hay un matching de tamaño k? ¿Es 2-coloreable?

La clase P

P

Problemas de decisión con un algoritmo correcto para toda instancia, que corre en $O(b^c)$ para alguna constante c.

¿Hay camino de s a t? ¿Hay un matching de tamaño k? ¿Es 2-coloreable?

- Es una propiedad del **problema**, no de un programa.

La clase P

P

Problemas de decisión con un algoritmo correcto para toda instancia, que corre en $O(b^c)$ para alguna constante c.

¿Hay camino de s a t? ¿Hay un matching de tamaño k? ¿Es 2-coloreable?

- Es una propiedad del **problema**, no de un programa.
- $O(b^{100})$ es polinomial: no significa "rápido".

La clase P

P

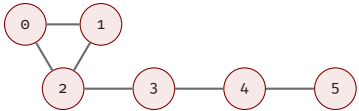
Problemas de decisión con un algoritmo correcto para toda instancia, que corre en $O(b^c)$ para alguna constante c.

¿Hay camino de s a t? ¿Hay un matching de tamaño k? ¿Es 2-coloreable?

- Es una propiedad del **problema**, no de un programa.
- $O(b^{100})$ es polinomial: no significa "rápido".
- Se usa porque es robusta, y porque un polinomio de un polinomio es un polinomio.

Comprobar es más fácil que encontrar

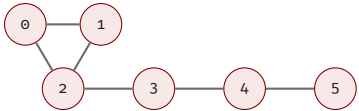
¿Hay un conjunto independiente de tamaño 3?



Comprobar es más fácil que encontrar

¿Hay un conjunto independiente de tamaño 3?

Encontrarlo parece difícil.

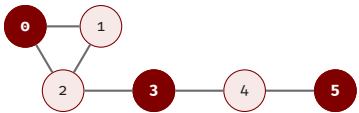


Comprobar es más fácil que encontrar

¿Hay un conjunto independiente de tamaño 3?

Encontrarlo parece difícil.

Si alguien dice “este es: $\{0, 3, 5\}$”, se revisa cada arista una vez.



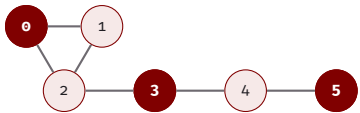
Comprobar es más fácil que encontrar

¿Hay un conjunto independiente de tamaño 3?

Encontrarlo parece difícil.

Si alguien dice “este es: $\{0, 3, 5\}$”, se revisa cada arista una vez.

Esa lista es un **certificado**.



La clase NP

NP

Un problema está en NP si hay un **verificador** V y un polinomio p tales que:

1. V recibe x y un **certificado** c de a lo sumo $p(|x|)$ bits, y corre en tiempo polinomial en $|x|$.
2. Si x es "sí", **existe** algún c con el que V acepta.
3. Si x es "no", V rechaza **todo** c.

La clase NP

NP

Un problema está en NP si hay un **verificador** V y un polinomio p tales que:

1. V recibe x y un **certificado** c de a lo sumo $p(|x|)$ bits, y corre en tiempo polinomial en $|x|$.
2. Si x es "sí", **existe** algún c con el que V acepta.
3. Si x es "no", V rechaza **todo** c.

Nadie dice de dónde sale c. El verificador solo lo revisa: nunca se deja engañar.

La clase NP

NP

Un problema está en NP si hay un **verificador** V y un polinomio p tales que:

1. V recibe x y un **certificado** c de a lo sumo $p(|x|)$ bits, y corre en tiempo polinomial en $|x|$.
2. Si x es "sí", **existe** algún c con el que V acepta.
3. Si x es "no", V rechaza **todo** c.

Nadie dice de dónde sale c. El verificador solo lo revisa: nunca se deja engañar.

Ya lo vimos: el corte que prueba que un flujo es máximo, en la semana 11.

Verificar un conjunto independiente: tamaño

pseudocódigo

```
algoritmo verificar_independiente(G, k, S)
    marca ← arreglo de V valores falso
    distintos ← 0
    para i desde 0 hasta |S| - 1 hacer
        si no marca[S[i]] entonces
            marca[S[i]] ← verdadero
            distintos ← distintos + 1
        fin si
    fin para
    si distintos < k entonces
        devolver falso
    fin si
```

El certificado es la lista S, de a lo sumo $V \log V$ bits. Un nodo repetido no se cuenta dos veces.

Verificar un conjunto independiente: aristas

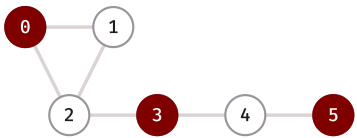
continúa

```
    para i desde 0 hasta E - 1 hacer
        (u, v) ← aristas[i]
        si marca[u] y marca[v] entonces
            devolver falso
        fin si
    fin para
    devolver verdadero
fin algoritmo
```

Se pasa una vez por los nodos de S y una vez por las aristas: $O(V + E)$.

El verificador con $k = 3$ y $S = \{0, 3, 5\}$

```
G, k = 3
certificado S = {0, 3, 5}
```



marca: ¿el nodo está en S?

0	1	2	3	4	5
sí	no	no	sí	no	sí

`distintos = 3` `k = 3`

aristas, en orden

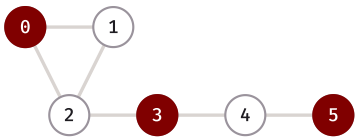
0-1	0-2	1-2	2-3	3-4	4-5

revisadas: 0 de 6

S[2] = 5: marca[5] ← verdadero, y distintos pasa a 3.

El verificador con $k = 3$ y $S = \{0, 3, 5\}$

G, k = 3
certificado S = {0, 3, 5}



marca: ¿el nodo está en S?

0	1	2	3	4	5
sí	no	no	sí	no	sí

distintos = 3 k = 3 3 ≥ 3

aristas, en orden

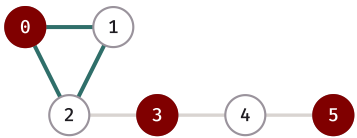
0-1	0-2	1-2	2-3	3-4	4-5

revisadas: 0 de 6

distintos = 3 no es menor que k = 3: S tiene tamaño suficiente, y sigue con las aristas.

El verificador con $k = 3$ y $S = \{0, 3, 5\}$

```
G, k = 3
certificado S = {0, 3, 5}
```



marca: ¿el nodo está en S?

0	1	2	3	4	5
sí	no	no	sí	no	sí

`distintos = 3` `k = 3`

aristas, en orden

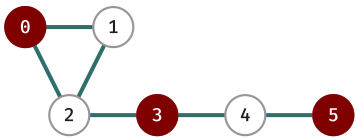
0-1	0-2	1-2	2-3	3-4	4-5

revisadas: 3 de 6

Arista 1-2: marca[1] = no y marca[2] = no. No tiene los dos extremos en S.

El verificador con $k = 3$ y $S = \{0, 3, 5\}$

```
G, k = 3
certificado S = {0, 3, 5}
```



marca: ¿el nodo está en S?

0	1	2	3	4	5
sí	no	no	sí	no	sí

`distintos = 3` `k = 3`

aristas, en orden

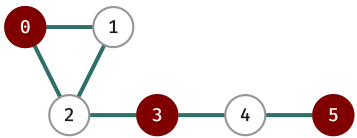
0-1	0-2	1-2	2-3	3-4	4-5

revisadas: 6 de 6

Arista 4-5: marca[4] = no y marca[5] = sí. No tiene los dos extremos en S.

El verificador con $k = 3$ y $S = \{0, 3, 5\}$

```
G, k = 3
certificado S = {0, 3, 5}
```



marca: ¿el nodo está en S?

0	1	2	3	4	5
sí	no	no	sí	no	sí

`distintos = 3` `k = 3`

aristas, en orden

0-1	0-2	1-2	2-3	3-4	4-5

revisadas: 6 de 6

`devolver verdadero`

El verificador acepta: S es la prueba. Una pasada por S y otra por las aristas: O(V + E).

Más certificados

Problema	Certificado	Qué se verifica
Cobertura de vértices	Lista de nodos	$\leq k$ nodos; cada arista toca a alguno
Clique	Lista de nodos	$\geq k$ nodos; todo par es vecino
Coloreo con k colores	Color de cada nodo	Ninguna arista con dos iguales
SAT	Valor de cada variable	Cada cláusula tiene un literal verdadero
Ciclo hamiltoniano	Orden de los nodos	Cada nodo una vez; pares seguidos son aristas
Mochila (decisión)	Lista de objetos	Peso $\leq W$; valor $\geq V$
¿N es compuesto?	Un divisor d	$1 < d < N$ y d divide a N

Más certificados

Problema	Certificado	Qué se verifica
Cobertura de vértices	Lista de nodos	$\leq k$ nodos; cada arista toca a alguno
Clique	Lista de nodos	$\geq k$ nodos; todo par es vecino
Coloreo con k colores	Color de cada nodo	Ninguna arista con dos iguales
SAT	Valor de cada variable	Cada cláusula tiene un literal verdadero
Ciclo hamiltoniano	Orden de los nodos	Cada nodo una vez; pares seguidos son aristas
Mochila (decisión)	Lista de objetos	Peso $\leq W$; valor $\geq V$
¿N es compuesto?	Un divisor d	$1 < d < N$ y d divide a N

En la mochila, verificar es sumar: no aparece el problema de $O(nW)$.

Más certificados

Problema	Certificado	Qué se verifica
Cobertura de vértices	Lista de nodos	$\leq k$ nodos; cada arista toca a alguno
Clique	Lista de nodos	$\geq k$ nodos; todo par es vecino
Coloreo con k colores	Color de cada nodo	Ninguna arista con dos iguales
SAT	Valor de cada variable	Cada cláusula tiene un literal verdadero
Ciclo hamiltoniano	Orden de los nodos	Cada nodo una vez; pares seguidos son aristas
Mochila (decisión)	Lista de objetos	Peso $\leq W$; valor $\geq V$
¿N es compuesto?	Un divisor d	$1 < d < N$ y d divide a N

En la mochila, verificar es sumar: no aparece el problema de $O(nW)$.

¿Y la prueba de que **no** hay conjunto independiente de tamaño k? Nadie conoce un certificado corto. La definición es asimétrica.

P está contenido en NP

Teorema

Todo problema de P está en NP.

Demostración directa.

P está contenido en NP

Teorema

Todo problema de P está en NP.

Demostración directa.

1. Sea A en P, con un algoritmo polinomial.

P está contenido en NP

Teorema

Todo problema de P está en NP.

Demostración directa.

1. Sea A en P, con un algoritmo polinomial.
2. El verificador ignora el certificado, corre ese algoritmo con x, y acepta si dice sí.

P está contenido en NP

Teorema

Todo problema de P está en NP.

Demostración directa.

1. Sea A en P, con un algoritmo polinomial.
2. El verificador ignora el certificado, corre ese algoritmo con x, y acepta si dice sí.
3. Es polinomial.

P está contenido en NP

Teorema

Todo problema de P está en NP.

Demostración directa.

1. Sea A en P, con un algoritmo polinomial.
2. El verificador ignora el certificado, corre ese algoritmo con x, y acepta si dice sí.
3. Es polinomial.
4. Si x es “sí”, acepta con cualquier certificado. Si es “no”, los rechaza todos. ■

P está contenido en NP

Teorema

Todo problema de P está en NP.

Demostración directa.

1. Sea A en P, con un algoritmo polinomial.
2. El verificador ignora el certificado, corre ese algoritmo con x, y acepta si dice sí.
3. Es polinomial.
4. Si x es "sí", acepta con cualquier certificado. Si es "no", los rechaza todos. ■

Error común

Leer NP como "no polinomial". Viene de *nondeterministic polynomial*. Ordenar está en NP.

SECCIÓN 04

P contra NP

04



La pregunta

¿P = NP?

Si una solución se puede **comprobar** rápido, ¿se puede también **encontrar** rápido?

La pregunta

¿P = NP?

Si una solución se puede **comprobar** rápido, ¿se puede también **encontrar** rápido?

- Sabemos que $P \subseteq NP$. No sabemos si la inclusión es estricta.

La pregunta

¿P = NP?

Si una solución se puede **comprobar** rápido, ¿se puede también **encontrar** rápido?

- Sabemos que $P \subseteq NP$. No sabemos si la inclusión es estricta.
- Es un problema abierto, con premio del Instituto Clay.

La pregunta

¿P = NP?

Si una solución se puede **comprobar** rápido, ¿se puede también **encontrar** rápido?

- Sabemos que $P \subseteq NP$. No sabemos si la inclusión es estricta.
- Es un problema abierto, con premio del Instituto Clay.
- La mayoría de los investigadores **cree** que $P \neq NP$: es una creencia, no un teorema.

La pregunta

¿P = NP?

Si una solución se puede **comprobar** rápido, ¿se puede también **encontrar** rápido?

- Sabemos que $P \subseteq NP$. No sabemos si la inclusión es estricta.
- Es un problema abierto, con premio del Instituto Clay.
- La mayoría de los investigadores **cree** que $P \neq NP$: es una creencia, no un teorema.
- Todo problema de NP se resuelve por fuerza bruta, probando certificados. Lo que falta es hacerlo rápido.

La pregunta

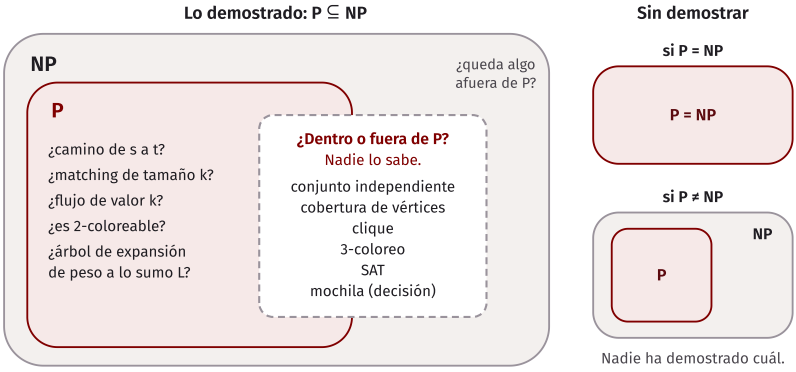
¿P = NP?

Si una solución se puede **comprobar** rápido, ¿se puede también **encontrar** rápido?

- Sabemos que $P \subseteq NP$. No sabemos si la inclusión es estricta.
- Es un problema abierto, con premio del Instituto Clay.
- La mayoría de los investigadores **cree** que $P \neq NP$: es una creencia, no un teorema.
- Todo problema de NP se resuelve por fuerza bruta, probando certificados. Lo que falta es hacerlo rápido.

Una frase “no tiene algoritmo polinomial” debe llevar delante: “si $P \neq NP$”.

Lo que se sabe y lo que no



Es un esquema: las formas y los tamaños no miden nada, y el espacio entre P y NP podría estar vacío.

SECCIÓN 05

Reducciones polinomiales

05

Reducción polinomial

$A \leq_p B$ (polynomial-time reduction)

Una función f tal que:

1. f se calcula en tiempo polinomial en el tamaño de x.
2. Para toda instancia x de A: x es "sí" en A **si y solo si** $f(x)$ es "sí" en B.

Reducción polinomial

$A \leq_p B$ (polynomial-time reduction)

Una función f tal que:

1. f se calcula en tiempo polinomial en el tamaño de x.
2. Para toda instancia x de A: x es "sí" en A **si y solo si** $f(x)$ es "sí" en B.

Comparen con la semana 12: una sola transformación, y la respuesta de B es la de A, sin nada que leer. Las dos direcciones siguen siendo obligatorias.

Si B está en P, A también

Teorema 1

Si $A \leq_p B$ y B está en P, entonces A está en P.

Demostración directa, componiendo algoritmos.

Si B está en P, A también

Teorema 1

Si $A \leq_p B$ y B está en P, entonces A está en P.

Demostración directa, componiendo algoritmos.

1. f corre en $p(b)$. M resuelve B en $q(m)$. Para resolver x: se calcula $f(x)$, y se corre M.

Si B está en P, A también

Teorema 1

Si $A \leq_p B$ y B está en P, entonces A está en P.

Demostración directa, componiendo algoritmos.

1. f corre en $p(b)$. M resuelve B en $q(m)$. Para resolver x: se calcula $f(x)$, y se corre M.
2. **Correcto:** M dice sí $\iff$ $f(x)$ es "sí" en B $\iff$ x es "sí" en A.

Si B está en P, A también

Teorema 1

Si $A \leq_p B$ y B está en P, entonces A está en P.

Demostración directa, componiendo algoritmos.

1. f corre en $p(b)$. M resuelve B en $q(m)$. Para resolver x: se calcula $f(x)$, y se corre M.
2. **Correcto:** M dice sí $\iff$ $f(x)$ es "sí" en B $\iff$ x es "sí" en A.
3. **Salida corta:** f tarda $p(b)$, así que $|f(x)| \leq p(b)$.

Si B está en P, A también

Teorema 1

Si $A \leq_p B$ y B está en P, entonces A está en P.

Demostración directa, componiendo algoritmos.

1. f corre en $p(b)$. M resuelve B en $q(m)$. Para resolver x: se calcula $f(x)$, y se corre M.
2. **Correcto:** M dice sí $\iff$ $f(x)$ es "sí" en B $\iff$ x es "sí" en A.
3. **Salida corta:** f tarda $p(b)$, así que $|f(x)| \leq p(b)$.
4. **Polinomial:** el costo es $p(b) + q(p(b))$, un polinomio en b. ■

Si B está en P, A también

Teorema 1

Si $A \leq_p B$ y B está en P, entonces A está en P.

Demostración directa, componiendo algoritmos.

1. f corre en $p(b)$. M resuelve B en $q(m)$. Para resolver x: se calcula $f(x)$, y se corre M.
2. **Correcto:** M dice sí $\iff$ $f(x)$ es "sí" en B $\iff$ x es "sí" en A.
3. **Salida corta:** f tarda $p(b)$, así que $|f(x)| \leq p(b)$.
4. **Polinomial:** el costo es $p(b) + q(p(b))$, un polinomio en b. ■

La contrarrecíproca

Si $A \leq_p B$ y A no está en P, entonces B no está en P.

Transitividad

Teorema 2

Si $A \leq_p B$ y $B \leq_p C$, entonces $A \leq_p C$.

Demostración directa. Sea $h(x) = g(f(x))$.

Transitividad

Teorema 2

Si $A \leq_p B$ y $B \leq_p C$, entonces $A \leq_p C$.

Demostración directa. Sea $h(x) = g(f(x))$.

- **Correcta:** x "sí" en $A \iff f(x)$ "sí" en $B \iff g(f(x))$ "sí" en C.

Transitividad

Teorema 2

Si $A \leq_p B$ y $B \leq_p C$, entonces $A \leq_p C$.

Demostración directa. Sea $h(x) = g(f(x))$.

- **Correcta:** x "sí" en $A \iff f(x)$ "sí" en $B \iff g(f(x))$ "sí" en C.
- **Polinomial:** f cuesta $p(b)$ y escribe a lo sumo $p(b)$ bits. Sobre eso, g cuesta $q(p(b))$.

■

Transitividad

Teorema 2
Si $A \leq_p B$ y $B \leq_p C$, entonces $A \leq_p C$.

Demostración directa. Sea $h(x) = g(f(x))$.

- **Correcta:** x "sí" en $A \iff f(x)$ "sí" en $B \iff g(f(x))$ "sí" en C.
- **Polinomial:** f cuesta $p(b)$ y escribe a lo sumo $p(b)$ bits. Sobre eso, g cuesta $q(p(b))$.

■

Por esto cada reducción se puede pegar a las anteriores, formando una cadena.

Qué dice y qué no dice

Si $A \leq_p B$:

Si se sabe que...	Se concluye	Por qué

Qué dice y qué no dice

Si $A \leq_p B$:

Si se sabe que...	**Se concluye**	**Por qué**
B está en P	A está en P	Teorema 1

Qué dice y qué no dice

Si $A \leq_p B$:

Si se sabe que...	**Se concluye**	**Por qué**
B está en P	A está en P	Teorema 1
A no está en P	B no está en P	Contrarrecíproca

Qué dice y qué no dice

Si $A \leq_p B$:

Si se sabe que...	**Se concluye**	**Por qué**
B está en P	A está en P	Teorema 1
A no está en P	B no está en P	Contrarrecíproca
A es difícil	B es **al menos tan** difícil	Quien resuelve B resuelve A

Qué dice y qué no dice

Si $A \leq_p B$:

Si se sabe que...	**Se concluye**	**Por qué**
B está en P	A está en P	Teorema 1
A no está en P	B no está en P	Contrarrecíproca
A es difícil	B es **al menos tan** difícil	Quien resuelve B resuelve A
A está en P	**Nada** sobre B	B puede ser cualquier cosa

Qué dice y qué no dice

Si $A \leq_p B$:

Si se sabe que...	**Se concluye**	**Por qué**
B está en P	A está en P	Teorema 1
A no está en P	B no está en P	Contrarrecíproca
A es difícil	B es **al menos tan** difícil	Quien resuelve B resuelve A
A está en P	**Nada** sobre B	B puede ser cualquier cosa
B es difícil	**Nada** sobre A	A puede ser fácil

Qué dice y qué no dice

Si $A \leq_p B$:

Si se sabe que...	**Se concluye**	**Por qué**
B está en P	A está en P	Teorema 1
A no está en P	B no está en P	Contrarrecíproca
A es difícil	B es **al menos tan** difícil	Quien resuelve B resuelve A
A está en P	**Nada** sobre B	B puede ser cualquier cosa
B es difícil	**Nada** sobre A	A puede ser fácil

"Al menos tan difícil" no dice que B esté en NP ni que sean iguales. Para eso hace falta también $B \leq_p A$.

SECCIÓN 06

La dirección correcta

06

El error de esta parte del curso

Quieren demostrar que un problema nuevo N es difícil. Conocen un difícil D.

El error de esta parte del curso

Quieren demostrar que un problema nuevo N es difícil. Conocen un difícil D.

Bien: $D \leq_p N$

Del **conocido** al **nuevo**. Si alguien resolviera N rápido, usaría f para resolver D rápido.

El error de esta parte del curso

Quieren demostrar que un problema nuevo N es difícil. Conocen un difícil D.

Bien: $D \leq_p N$

Del **conocido** al **nuevo**. Si alguien resolviera N rápido, usaría f para resolver D rápido.

Mal: $N \leq_p D$

Del nuevo al conocido. Solo dice que N no es más difícil que D. Podría ser muy fácil.

El error de esta parte del curso

Quieren demostrar que un problema nuevo N es difícil. Conocen un difícil D.

Bien: $D \leq_p N$

Del **conocido** al **nuevo**. Si alguien resolviera N rápido, usaría f para resolver D rápido.

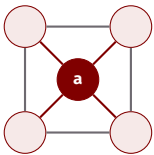
Mal: $N \leq_p D$

Del nuevo al conocido. Solo dice que N no es más difícil que D. Podría ser muy fácil.

La dificultad fluye en el sentido de $\leq_p$, de izquierda a derecha.

Un ejemplo: 2-coloreo $\leq_p$ 3-coloreo

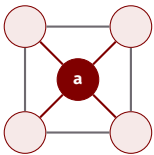
Reducción. Se agrega un nodo nuevo *a*, vecino de todos.



Un ejemplo: 2-coloreo $\leq_p$ 3-coloreo

Reducción. Se agrega un nodo nuevo a, vecino de todos.

Si G usa 2 colores: se dan esos a G, y el tercero a a.

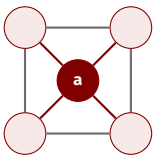


Un ejemplo: 2-coloreo $\leq_p$ 3-coloreo

Reducción. Se agrega un nodo nuevo a, vecino de todos.

Si G usa 2 colores: se dan esos a G, y el tercero a a.

Si G' usa 3 colores: sea c el color de a. Ningún nodo de G tiene c, así que G usa los otros dos. ■



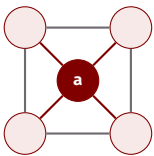
Un ejemplo: 2-coloreo $\leq_p$ 3-coloreo

Reducción. Se agrega un nodo nuevo a, vecino de todos.

Si G usa 2 colores: se dan esos a G, y el tercero a a.

Si G' usa 3 colores: sea c el color de a. Ningún nodo de G tiene c, así que G usa los otros dos. ■

Es correcta, el 3-coloreo parece difícil, y el 2-coloreo se resuelve con un BFS. La flecha al revés no prueba dificultad.



Comprobado con el código

```
build/reduction_demo, extracto

2-coloreo a 3-coloreo: agregar un nodo vecino de todos.
  ciclo de 4: 2 colores con BFS: sí; 3 colores en el grafo con el nodo nuevo: sí
  ciclo de 5: 2 colores con BFS: no; 3 colores en el grafo con el nodo nuevo: no
```

Las pruebas lo revisan en todos los grafos de hasta 5 nodos.

Quiero...	**La flecha**	**Concluyo**
Resolver N con un solucionador de S	$N \leq_p S$	Si S es fácil, N es fácil
Demostrar que N es difícil	$D \leq_p N$	N no es más fácil que D

Comprobado con el código

build/reduction_demo, extracto

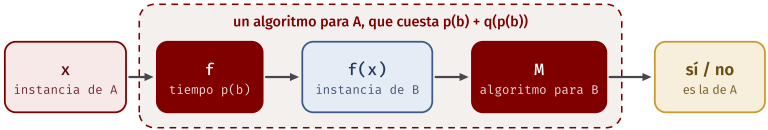
```
2-coloreo a 3-coloreo: agregar un nodo vecino de todos.
  ciclo de 4: 2 colores con BFS: sí; 3 colores en el grafo con el nodo nuevo: sí
  ciclo de 5: 2 colores con BFS: no; 3 colores en el grafo con el nodo nuevo: no
```

Las pruebas lo revisan en todos los grafos de hasta 5 nodos.

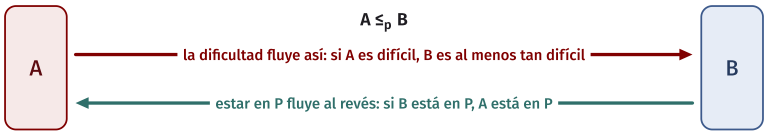
Quiero...	**La flecha**	**Concluyo**
Resolver N con un solucionador de S	$N \leq_p S$	Si S es fácil, N es fácil
Demostrar que N es difícil	$D \leq_p N$	N no es más fácil que D

Codificar el sudoku en SAT sirve para resolverlo. No dice que el sudoku sea difícil.

$A \leq_p B$: transformar, responder y concluir



x es "sí" en A si y solo si f(x) es "sí" en B: la respuesta de B es la de A, sin nada que leer.



Nada se concluye de: A está en P (B puede ser cualquier cosa), ni de: B es difícil (A puede ser fácil).

Para demostrar que N es difícil
$D \leq_p N$
del conocido al nuevo; $N \leq_p D$ no lo demuestra

Para resolver N con un solucionador de S
$N \leq_p S$
si S es fácil, N es fácil; no dice que N sea difícil

La flecha de la reducción siempre va del problema que se transforma al que recibe la instancia.

NP-difícil y NP-completo: un adelanto

NP-difícil *(NP-hard)*

Todo problema de NP se reduce a él.

NP-completo *(NP-complete)*

Es NP-difícil y además está en NP.

NP-difícil y NP-completo: un adelanto

NP-difícil *(NP-hard)*

Todo problema de NP se reduce a él.

NP-completo *(NP-complete)*

Es NP-difícil y además está en NP.

- Si uno solo de estos estuviera en P, entonces P = NP.

NP-difícil y NP-completo: un adelanto

NP-difícil *(NP-hard)*

Todo problema de NP se reduce a él.

NP-completo *(NP-complete)*

Es NP-difícil y además está en NP.

- Si uno solo de estos estuviera en P, entonces P = NP.
- Cook-Levin lo prueba para SAT. Lo demás sale por transitividad: semana 14.

NP-difícil y NP-completo: un adelanto

NP-difícil *(NP-hard)*

Todo problema de NP se reduce a él.

NP-completo *(NP-complete)*

Es NP-difícil y además está en NP.

- Si uno solo de estos estuviera en P, entonces P = NP.
- Cook-Levin lo prueba para SAT. Lo demás sale por transitividad: semana 14.
- **Se usa aquí sin demostrar todavía:** conjunto independiente es NP-completo.

NP-difícil y NP-completo: un adelanto

NP-difícil *(NP-hard)*

Todo problema de NP se reduce a él.

NP-completo *(NP-complete)*

Es NP-difícil y además está en NP.

- Si uno solo de estos estuviera en P, entonces P = NP.
- Cook-Levin lo prueba para SAT. Lo demás sale por transitividad: semana 14.
- **Se usa aquí sin demostrar todavía:** conjunto independiente es NP-completo.
- NP-completo no significa imposible: instancias pequeñas, casos especiales, aproximar, o un solucionador.

Por qué la mochila no demuestra P = NP

Si la mochila se resuelve en $O(nW)$, ¿por qué eso no demuestra P = NP?

Por qué la mochila no demuestra P = NP

Si la mochila se resuelve en $O(nW)$, ¿por qué eso no demuestra P = NP?

1. La versión de decisión es NP-completa. Aquí lo **asumimos**: la semana 14 demuestra suma de subconjuntos, un caso especial.

Por qué la mochila no demuestra P = NP

Si la mochila se resuelve en $O(nW)$, ¿por qué eso no demuestra P = NP?

1. La versión de decisión es NP-completa. Aquí lo **asumimos**: la semana 14 demuestra suma de subconjuntos, un caso especial.
2. Si $O(nW)$ fuera polinomial, tendríamos P = NP.

Por qué la mochila no demuestra P = NP

Si la mochila se resuelve en $O(nW)$, ¿por qué eso no demuestra P = NP?

1. La versión de decisión es NP-completa. Aquí lo **asumimos**: la semana 14 demuestra suma de subconjuntos, un caso especial.
2. Si $O(nW)$ fuera polinomial, tendríamos P = NP.
3. Pero W ocupa $\log W$ bits: es exponencial en esa parte de la entrada.

Por qué la mochila no demuestra P = NP

Si la mochila se resuelve en $O(nW)$, ¿por qué eso no demuestra P = NP?

1. La versión de decisión es NP-completa. Aquí lo **asumimos**: la semana 14 demuestra suma de subconjuntos, un caso especial.
2. Si $O(nW)$ fuera polinomial, tendríamos P = NP.
3. Pero W ocupa $\log W$ bits: es exponencial en esa parte de la entrada.
4. Con un W de 100 bits, la tabla puede tener cerca de 2^{100} columnas.

Por qué la mochila no demuestra P = NP

Si la mochila se resuelve en $O(nW)$, ¿por qué eso no demuestra P = NP?

1. La versión de decisión es NP-completa. Aquí lo **asumimos**: la semana 14 demuestra suma de subconjuntos, un caso especial.
2. Si $O(nW)$ fuera polinomial, tendríamos P = NP.
3. Pero W ocupa $\log W$ bits: es exponencial en esa parte de la entrada.
4. Con un W de 100 bits, la tabla puede tener cerca de 2^{100} columnas.

Es pseudopolinomial. Sirve si W es pequeño, y no dice nada de P contra NP.

SECCIÓN 07

Conjunto independiente y cobertura

07

Dos problemas

Grafos no dirigidos y simples.

Conjunto independiente *(independent set)*
Nodos sin ninguna arista entre ellos. ¿Hay uno de tamaño **al menos** k?

Cobertura de vértices *(vertex cover)*
Nodos que tocan a toda arista. ¿Hay una de tamaño **a lo sumo** k?

El lema

Lema

S es un conjunto independiente si y solo si $V - S$ es una cobertura de vértices.

Demostración directa, en las dos direcciones.

El lema

Lema

S es un conjunto independiente si y solo si $V - S$ es una cobertura de vértices.

Demostración directa, en las dos direcciones.

Si S es independiente: sea $\{u, v\}$ una arista.

El lema

Lema

S es un conjunto independiente si y solo si $V - S$ es una cobertura de vértices.

Demostración directa, en las dos direcciones.

Si S es independiente: sea $\{u, v\}$ una arista. Los dos no pueden estar en S.

El lema

Lema

S es un conjunto independiente si y solo si $V - S$ es una cobertura de vértices.

Demostración directa, en las dos direcciones.

Si S es independiente: sea $\{u, v\}$ una arista. Los dos no pueden estar en S. Entonces uno está en $V - S$, y la arista queda cubierta.

El lema

Lema

S es un conjunto independiente si y solo si $V - S$ es una cobertura de vértices.

Demostración directa, en las dos direcciones.

Si S es independiente: sea $\{u, v\}$ una arista. Los dos no pueden estar en S. Entonces uno está en $V - S$, y la arista queda cubierta.

Si $V - S$ es cobertura: sean u, v en S.

El lema

Lema

S es un conjunto independiente si y solo si $V - S$ es una cobertura de vértices.

Demostración directa, en las dos direcciones.

Si S es independiente: sea $\{u, v\}$ una arista. Los dos no pueden estar en S. Entonces uno está en $V - S$, y la arista queda cubierta.

Si $V - S$ es cobertura: sean u, v en S. Ninguno está en $V - S$.

El lema

Lema

S es un conjunto independiente si y solo si $V - S$ es una cobertura de vértices.

Demostración directa, en las dos direcciones.

Si S es independiente: sea $\{u, v\}$ una arista. Los dos no pueden estar en S. Entonces uno está en $V - S$, y la arista queda cubierta.

Si $V - S$ es cobertura: sean u, v en S. Ninguno está en $V - S$. Si $\{u, v\}$ fuera arista, quedaría sin cubrir: contradicción. ■

La reducción

Teorema

Conjunto independiente $\leq_p$ cobertura de vértices, con $f(G, k) = (G,\ n - k)$.

La reducción

Teorema

Conjunto independiente $\leq_p$ cobertura de vértices, con $f(G, k) = (G,\ n - k)$.

- **Tiempo:** lineal. Copia el grafo y resta.

La reducción

Teorema

Conjunto independiente $\leq_p$ cobertura de vértices, con $f(G, k) = (G,\ n - k)$.

- **Tiempo:** lineal. Copia el grafo y resta.
- **De CI a CV:** $|S| \geq k$. Por el lema $V - S$ es cobertura, y $|V - S| = n - |S| \leq n - k$.

La reducción

Teorema

Conjunto independiente $\leq_p$ cobertura de vértices, con $f(G, k) = (G,\ n - k)$.

- **Tiempo:** lineal. Copia el grafo y resta.
- **De CI a CV:** $|S| \geq k$. Por el lema $V - S$ es cobertura, y $|V - S| = n - |S| \leq n - k$.
- **De CV a CI:** $|C| \leq n - k$. Por el lema $V - C$ es independiente, y $|V - C| = n - |C| \geq k$. ■

La reducción

Teorema

Conjunto independiente $\leq_p$ cobertura de vértices, con $f(G, k) = (G,\ n - k)$.

- **Tiempo:** lineal. Copia el grafo y resta.
- **De CI a CV:** $|S| \geq k$. Por el lema $V - S$ es cobertura, y $|V - S| = n - |S| \leq n - k$.
- **De CV a CI:** $|C| \leq n - k$. Por el lema $V - C$ es independiente, y $|V - C| = n - |C| \geq k$. ■

La misma f sirve en el otro sentido. Los dos son equivalentes:

$$\text{cobertura mínima} = n - \text{conjunto independiente máximo.}$$

La reducción

Teorema

Conjunto independiente $\leq_p$ cobertura de vértices, con $f(G, k) = (G,\ n - k)$.

- **Tiempo:** lineal. Copia el grafo y resta.
- **De CI a CV:** $|S| \geq k$. Por el lema $V - S$ es cobertura, y $|V - S| = n - |S| \leq n - k$.
- **De CV a CI:** $|C| \leq n - k$. Por el lema $V - C$ es independiente, y $|V - C| = n - |C| \geq k$. ■

La misma f sirve en el otro sentido. Los dos son equivalentes:

$$\text{cobertura mínima} = n - \text{conjunto independiente máximo.}$$

Borde: si $k > n$, $n - k$ es negativo, y los dos dicen no.

Corriendo la reducción

build/reduction_demo, extracto

```
G tiene 6 nodos y estas aristas: 0-1 0-2 1-2 2-3 3-4 4-5
conjunto independiente máximo: {0, 3, 5}, tamaño 3
cobertura mínima:              {0, 2, 4}, tamaño 3
V − S = {1, 2, 4}, ¿es cobertura? sí

Conjunto independiente a cobertura, con k → n − k:
  k  CI(G, k)  CV(G, n − k)
  2  sí        sí
  3  sí        sí
  4  no        no
```

Corriendo la reducción

build/reduction_demo, extracto

```
G tiene 6 nodos y estas aristas: 0-1 0-2 1-2 2-3 3-4 4-5
conjunto independiente máximo: {0, 3, 5}, tamaño 3
cobertura mínima:              {0, 2, 4}, tamaño 3
V − S = {1, 2, 4}, ¿es cobertura? sí

Conjunto independiente a cobertura, con k → n − k:
  k  CI(G, k)  CV(G, n − k)
  2  sí        sí
  3  sí        sí
  4  no        no
```

El lema no dice que haya uno solo: van en pares. La fuerza bruta halló otra cobertura mínima.

Corriendo la reducción

build/reduction_demo, extracto

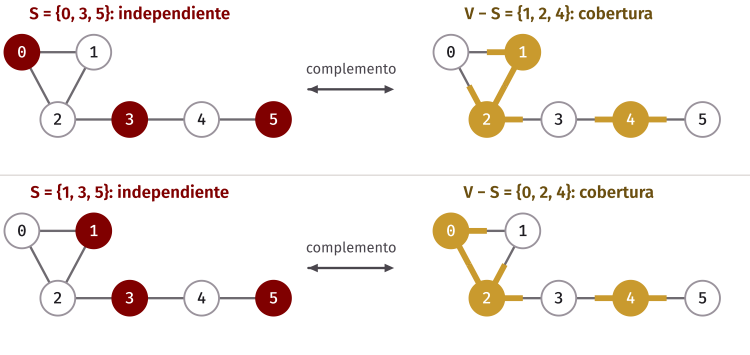
```
G tiene 6 nodos y estas aristas: 0-1 0-2 1-2 2-3 3-4 4-5
conjunto independiente máximo: {0, 3, 5}, tamaño 3
cobertura mínima:              {0, 2, 4}, tamaño 3
V − S = {1, 2, 4}, ¿es cobertura? sí

Conjunto independiente a cobertura, con k → n − k:
  k  CI(G, k)  CV(G, n − k)
  2  sí        sí
  3  sí        sí
  4  no        no
```

El lema no dice que haya uno solo: van en pares. La fuerza bruta halló otra cobertura mínima.

Las pruebas lo revisan en **todos** los grafos de hasta 5 nodos, y en 300 al azar de 6 a 8.

Van en pares



Las 6 aristas de G quedan tocadas en los dos pares, y los tamaños suman n: 3 + 3 = 6.

SECCIÓN 08

Conjunto independiente y clique

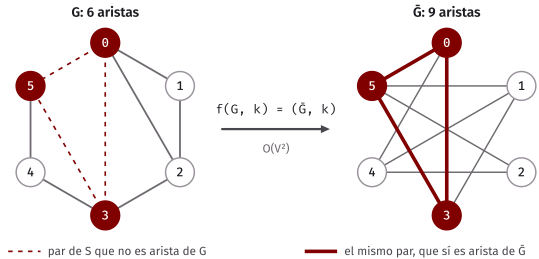
08

El grafo complementario

Clique

Todo par de nodos es vecino. En un independiente, **ningún** par lo es.

$\bar{G}$ tiene los mismos nodos, y $\{u, v\}$ es arista de $\bar{G}$ cuando no lo es de G.



S = {0, 3, 5}: ningún par es arista de G, y todo par es arista de Ḡ.

La reducción

Lema

S es independiente en G si y solo si S es un clique de $\bar{G}$.

La reducción

Lema

S es independiente en G si y solo si S es un clique de $\bar{G}$.

Demostración directa. Para $u \neq v$ en S: $\{u, v\}$ no es arista de $G \iff$ es arista de $\bar{G}$. "Ningún par es arista de G" dice lo mismo que "todo par es arista de $\bar{G}$". ■

La reducción

Lema

S es independiente en G si y solo si S es un clique de $\bar{G}$.

Demostración directa. Para $u \neq v$ en S: $\{u, v\}$ no es arista de $G \iff$ es arista de $\bar{G}$. "Ningún par es arista de G" dice lo mismo que "todo par es arista de $\bar{G}$". ■

Teorema

Conjunto independiente $\leq_p$ clique, con $f(G, k) = (\bar{G}, k)$.

La reducción

Lema

S es independiente en G si y solo si S es un clique de $\bar{G}$.

Demostración directa. Para $u \neq v$ en S: $\{u, v\}$ no es arista de $G \iff$ es arista de $\bar{G}$. "Ningún par es arista de G" dice lo mismo que "todo par es arista de $\bar{G}$". ■

Teorema

Conjunto independiente $\leq_p$ clique, con $f(G, k) = (\bar{G}, k)$.

- **Tiempo:** $O(V^2)$. El resultado puede ser más largo que la entrada.

La reducción

Lema

S es independiente en G si y solo si S es un clique de $\bar{G}$.

Demostración directa. Para $u \neq v$ en S: $\{u, v\}$ no es arista de $G \iff$ es arista de $\bar{G}$. "Ningún par es arista de G" dice lo mismo que "todo par es arista de $\bar{G}$". ■

Teorema

Conjunto independiente $\leq_p$ clique, con $f(G, k) = (\bar{G}, k)$.

- **Tiempo:** $O(V^2)$. El resultado puede ser más largo que la entrada.
- **Dirección 1 y dirección 2:** las dos están en el "si y solo si" del lema.

La cadena en el código

graphs.h

```
inline Instance independent_set_to_vertex_cover(const Instance& in) {
    return {in.graph, in.graph.n - in.k};
}
inline Instance independent_set_to_clique(const Instance& in) {
    return {complement(in.graph), in.k};
}
inline Instance vertex_cover_to_clique(const Instance& in) {
    return independent_set_to_clique(independent_set_to_vertex_cover(in));
}
```

Paso	Instancia	Solución	Qué se comprueba
Cobertura	$(G, 3)$	$\{0, 2, 4\}$	Toca las 6 aristas
Independiente	$(G, 6-3)$	$\{1, 3, 5\}$	Su complemento; sin aristas entre ellos
Clique	$(\bar{G}, 3)$	$\{1, 3, 5\}$	1-3, 1-5 y 3-5 son aristas de $\bar{G}$

SECCIÓN 09

El patrón

09

El patrón de una reducción

Para demostrar $A \leq_p B$:

El patrón de una reducción

Para demostrar $A \leq_p B$:

1. **Escribir f.** No resuelve A: no sabe la respuesta.

El patrón de una reducción

Para demostrar $A \leq_p B$:

1. **Escribir f.** No resuelve A: no sabe la respuesta.
2. **Comprobar el tiempo.** Polinomial en el tamaño de x.

El patrón de una reducción

Para demostrar $A \leq_p B$:

1. **Escribir f.** No resuelve A: no sabe la respuesta.
2. **Comprobar el tiempo.** Polinomial en el tamaño de x.
3. **Si x es "sí", $f(x)$ es "sí".** De una solución de x se construye una de $f(x)$.

El patrón de una reducción

Para demostrar $A \leq_p B$:

1. **Escribir** f**.** No resuelve A: no sabe la respuesta.
2. **Comprobar el tiempo.** Polinomial en el tamaño de x.
3. **Si** x **es "sí",** $f(x)$ **es "sí".** De una solución de x se construye una de $f(x)$.
4. **Si** $f(x)$ **es "sí",** x **es "sí".** De una solución **cualquiera** de $f(x)$ se construye una de x.

El patrón de una reducción

Para demostrar $A \leq_p B$:

1. **Escribir** f**.** No resuelve A: no sabe la respuesta.
2. **Comprobar el tiempo.** Polinomial en el tamaño de x.
3. **Si** x **es "sí",** $f(x)$ **es "sí".** De una solución de x se construye una de $f(x)$.
4. **Si** $f(x)$ **es "sí",** x **es "sí".** De una solución **cualquiera** de $f(x)$ se construye una de x.

Dos cosas que se llaman igual

La **dirección de la reducción** dice qué se transforma en qué. Las **dos direcciones de la demostración** son los pasos 3 y 4, y las dos son obligatorias.

Una reducción que falla

“Son complementarios, así que $f(G, k) = (G, k)$ debe servir.”

build/reduction_demo, extracto

```
Una reducción mal hecha: conservar k, sin cambiarlo por n − k.
  el contraejemplo más chico: 2 nodos, aristas: 0-1
  k = 2: CI dice no, CV dice sí
```

Una reducción que falla

“Son complementarios, así que $f(G, k) = (G, k)$ debe servir.”

build/reduction_demo, extracto

```
Una reducción mal hecha: conservar k, sin cambiarlo por n − k.
  el contraejemplo más chico: 2 nodos, aristas: 0-1
  k = 2: CI dice no, CV dice sí
```

Una instancia distinta basta para que f no sea una reducción. Prueben sus reducciones con fuerza bruta en grafos pequeños.

Una reducción que falla

“Son complementarios, así que $f(G, k) = (G, k)$ debe servir.”

build/reduction_demo, extracto

```
Una reducción mal hecha: conservar k, sin cambiarlo por n − k.
  el contraejemplo más chico: 2 nodos, aristas: 0-1
  k = 2: CI dice no, CV dice sí
```

Una instancia distinta basta para que f no sea una reducción. Prueben sus reducciones con fuerza bruta en grafos pequeños.

Error común

Que f resuelva antes de transformar. Prueba todos los subconjuntos y escribe una instancia trivial: cumple el “si y solo si”, pero no corre en tiempo polinomial.

SECCIÓN 10

SAT

10

SAT y CNF

- **Literal:** una variable x o su negación $\neg x$.
- **Cláusula:** un "o" de literales: $x_1 \vee \neg x_2 \vee x_3$.
- **CNF:** un "y" de cláusulas.
- **Modelo:** una asignación que hace verdadera la fórmula.

SAT

Dada una fórmula, ¿tiene un modelo?

SAT y CNF

- **Literal:** una variable x o su negación $\neg x$.
- **Cláusula:** un "o" de literales: $x_1 \vee \neg x_2 \vee x_3$.
- **CNF:** un "y" de cláusulas.
- **Modelo:** una asignación que hace verdadera la fórmula.

SAT

Dada una fórmula, ¿tiene un modelo?

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$$

SAT y CNF

- **Literal:** una variable x o su negación $\neg x$.
- **Cláusula:** un "o" de literales: $x_1 \vee \neg x_2 \vee x_3$.
- **CNF:** un "y" de cláusulas.
- **Modelo:** una asignación que hace verdadera la fórmula.

SAT

Dada una fórmula, ¿tiene un modelo?

$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$

- Está en NP: el certificado es el modelo.

SAT y CNF

- **Literal:** una variable x o su negación $\neg x$.
- **Cláusula:** un "o" de literales: $x_1 \vee \neg x_2 \vee x_3$.
- **CNF:** un "y" de cláusulas.
- **Modelo:** una asignación que hace verdadera la fórmula.

SAT

Dada una fórmula, ¿tiene un modelo?

$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$

- Está en NP: el certificado es el modelo.
- Fuerza bruta: 2^n asignaciones. Cook-Levin: es NP-completo.

Un solucionador por dentro: DPLL

Backtracking, más la **propagación unitaria**: lo obligatorio no se decide.

pseudocódigo

```
algoritmo dpll(formula, valores)
    mientras haya una cláusula con todos sus literales falsos, salvo uno sin asignar, hacer
        hacer verdadero ese literal en valores
    fin mientras
    si alguna cláusula tiene todos sus literales falsos entonces
        devolver falso
    fin si
```

DPLL: decidir y volver

continúa

```
    si todas las variables tienen valor entonces
        devolver verdadero
    fin si
    x ← una variable sin asignar
    si dpll(formula, valores con x verdadera) entonces
        devolver verdadero
    fin si
    devolver dpll(formula, valores con x falsa)
fin algoritmo
```

Es correcto porque la propagación solo asigna lo obligatorio, y el final prueba los dos valores de x. En el peor caso, exponencial.

DPLL paso a paso

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$$

```
build/cnf_demo

decisión: 1
  unitaria: 3
  conflicto
decisión: -1
  unitaria: 2
  unitaria: 4
  decisión: 3
satisfacible: -1 2 3 4
```

Paso	Por qué

DPLL paso a paso

$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$

```
build/cnf_demo

decisión: 1
  unitaria: 3
  conflicto
decisión: -1
  unitaria: 2
  unitaria: 4
  decisión: 3
satisfacible: -1 2 3 4
```

Paso	Por qué
x_1 verdadera	Ninguna cláusula obliga

DPLL paso a paso

$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$

```
build/cnf_demo

decisión: 1
  unitaria: 3
  conflicto
decisión: -1
  unitaria: 2
  unitaria: 4
  decisión: 3
satisfacible: -1 2 3 4
```

Paso	Por qué
x_1 verdadera	Ninguna cláusula obliga
x_3 verdadera	$\neg x_1$ es falso en $(\neg x_1 \vee x_3)$

DPLL paso a paso

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$$

```
build/cnf_demo

decisión: 1
  unitaria: 3
  conflicto
decisión: -1
  unitaria: 2
  unitaria: 4
  decisión: 3
satisfacible: -1 2 3 4
```

Paso	Por qué
x_1 verdadera	Ninguna cláusula obliga
x_3 verdadera	$\neg x_1$ es falso en $(\neg x_1 \vee x_3)$
conflicto	$(\neg x_1 \vee \neg x_3)$ es falsa

DPLL paso a paso

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$$

```
build/cnf_demo

decisión: 1
  unitaria: 3
  conflicto
decisión: -1
  unitaria: 2
  unitaria: 4
  decisión: 3
satisfacible: -1 2 3 4
```

Paso	Por qué
x_1 verdadera	Ninguna cláusula obliga
x_3 verdadera	$\neg x_1$ es falso en $(\neg x_1 \vee x_3)$
conflicto	$(\neg x_1 \vee \neg x_3)$ es falsa
x_1 falsa	La otra opción

DPLL paso a paso

$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$

```
build/cnf_demo

decisión: 1
  unitaria: 3
  conflicto
decisión: -1
  unitaria: 2
  unitaria: 4
  decisión: 3
satisfacible: -1 2 3 4
```

Paso	Por qué
x_1 verdadera	Ninguna cláusula obliga
x_3 verdadera	$\neg x_1$ es falso en $(\neg x_1 \vee x_3)$
conflicto	$(\neg x_1 \vee \neg x_3)$ es falsa
x_1 falsa	La otra opción
x_2 verdadera	x_1 es falso en $(x_1 \vee x_2)$

DPLL paso a paso

$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$

```
build/cnf_demo

decisión: 1
  unitaria: 3
  conflicto
decisión: -1
  unitaria: 2
  unitaria: 4
  decisión: 3
satisfacible: -1 2 3 4
```

Paso	Por qué
x_1 verdadera	Ninguna cláusula obliga
x_3 verdadera	$\neg x_1$ es falso en $(\neg x_1 \vee x_3)$
conflicto	$(\neg x_1 \vee \neg x_3)$ es falsa
x_1 falsa	La otra opción
x_2 verdadera	x_1 es falso en $(x_1 \vee x_2)$
x_4 verdadera	$\neg x_2$ es falso en $(\neg x_2 \vee x_4)$

DPLL paso a paso

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$$

```
build/cnf_demo

decisión: 1
  unitaria: 3
  conflicto
decisión: -1
  unitaria: 2
  unitaria: 4
  decisión: 3
satisfacible: -1 2 3 4
```

Paso	Por qué
x_1 verdadera	Ninguna cláusula obliga
x_3 verdadera	$\neg x_1$ es falso en $(\neg x_1 \vee x_3)$
conflicto	$(\neg x_1 \vee \neg x_3)$ es falsa
x_1 falsa	La otra opción
x_2 verdadera	x_1 es falso en $(x_1 \vee x_2)$
x_4 verdadera	$\neg x_2$ es falso en $(\neg x_2 \vee x_4)$
x_3 verdadera	Libre: queda el modelo

Práctico y sin garantía

Los de hoy agregan aprendizaje de cláusulas (CDCL). El peor caso no cambia.

build/sat_bench: n + 1 palomas en n huecos

huecos	variables	cláusulas	respuesta	conflictos	CPU (s)
5	30	81	UNSAT	251	0.000532
6	42	133	UNSAT	1206	0.003166
7	56	204	UNSAT	10495	0.05256
8	72	297	UNSAT	76178	0.459086
9	90	415	UNSAT	551941	4.24575
10	se acabó el límite de 20 s de CPU				

Tiempo que informa minisat, en el contenedor, en una máquina compartida. Depende de la máquina.

Práctico y sin garantía

Los de hoy agregan aprendizaje de cláusulas (CDCL). El peor caso no cambia.

build/sat_bench: n + 1 palomas en n huecos

huecos	variables	cláusulas	respuesta	conflictos	CPU (s)
5	30	81	UNSAT	251	0.000532
6	42	133	UNSAT	1206	0.003166
7	56	204	UNSAT	10495	0.05256
8	72	297	UNSAT	76178	0.459086
9	90	415	UNSAT	551941	4.24575
10	se acabó el límite de 20 s de CPU				

Tiempo que informa minisat, en el contenedor, en una máquina compartida. Depende de la máquina.

En cambio, el sudoku de 25 por 25 del laboratorio tiene 15625 variables, y tarda menos de un segundo.

Práctico y sin garantía

Los de hoy agregan aprendizaje de cláusulas (CDCL). El peor caso no cambia.

build/sat_bench: n + 1 palomas en n huecos

huecos	variables	cláusulas	respuesta	conflictos	CPU (s)
5	30	81	UNSAT	251	0.000532
6	42	133	UNSAT	1206	0.003166
7	56	204	UNSAT	10495	0.05256
8	72	297	UNSAT	76178	0.459086
9	90	415	UNSAT	551941	4.24575
10	se acabó el límite de 20 s de CPU				

Tiempo que informa minisat, en el contenedor, en una máquina compartida. Depende de la máquina.

En cambio, el sudoku de 25 por 25 del laboratorio tiene 15625 variables, y tarda menos de un segundo.

El tamaño de la fórmula no dice cuánto va a tardar. Conviene probarlo.

SECCIÓN 11

Codificar

11

Codificar un problema en CNF

1. Escoger **variables** con significado: “el nodo *v* tiene el color *a*”.

Codificar un problema en CNF

1. Escoger **variables** con significado: “el nodo *v* tiene el color *a*”.
2. Escribir **cláusulas por cada regla**.

Codificar un problema en CNF

1. Escoger **variables** con significado: “el nodo *v* tiene el color *a*”.
2. Escribir **cláusulas por cada regla**.
3. **Demostrar** que los modelos corresponden a las soluciones.

Codificar un problema en CNF

1. Escoger **variables** con significado: “el nodo *v* tiene el color *a*”.
2. Escribir **cláusulas por cada regla**.
3. **Demostrar** que los modelos corresponden a las soluciones.
4. Correr el solucionador, y **leer** la respuesta.

Codificar un problema en CNF

1. Escoger **variables** con significado: “el nodo v tiene el color a”.
2. Escribir **cláusulas por cada regla**.
3. **Demostrar** que los modelos corresponden a las soluciones.
4. Correr el solucionador, y **leer** la respuesta.

Regla	**Cláusulas**	**Cuántas**
Al menos uno	$x_1 \vee x_2 \vee \cdots \vee x_m$	1
A lo sumo uno	$\neg x_i \vee \neg x_j$, para cada par $i < j$	$m(m-1)/2$
Exactamente uno	Las dos anteriores	$1 + m(m-1)/2$

Exactamente uno de x, y, z

$(x \vee y \vee z) \wedge (\neg x \vee \neg y)$
$\wedge (\neg x \vee \neg z) \wedge (\neg y \vee \neg z)$

DIMACS

```
p cnf 3 4
1 2 3 0
-1 -2 0
-1 -3 0
-2 -3 0
```

build/cnf_demo, tabla de verdad

x	y	z	fórmula	verdaderas
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	2
1	0	0	1	1
1	0	1	0	2
1	1	0	0	2
1	1	1	0	3

La fórmula vale 1 exactamente en las filas con una sola variable verdadera.

DIMACS y minisat

ejemplos/formula.cnf

```
c (x1 o x2) y ...
p cnf 4 4
1 2 0
-1 3 0
-1 -3 0
-2 4 0
```

terminal

```
$ minisat ejemplos/formula.cnf build/formula.res
...
SATISFIABLE
$ echo $?
10
$ cat build/formula.res
SAT
-1 2 -3 4 0
```

- `c`: comentario.
- `p cnf V C`: encabezado.
- Cada cláusula termina en 0.

DIMACS y minisat

ejemplos/formula.cnf

```
c (x1 o x2) y ...
p cnf 4 4
1 2 0
-1 3 0
-1 -3 0
-2 4 0
```

- `c`: comentario.
- `p cnf V C`: encabezado.
- Cada cláusula termina en 0.

terminal

```
$ minisat ejemplos/formula.cnf build/formula.res
...
SATISFIABLE
$ echo $?
10
$ cat build/formula.res
SAT
-1 2 -3 4 0
```

- **Código de salida:** 10 si es SAT, 20 si es UNSAT.

DIMACS y minisat

ejemplos/formula.cnf

```
c (x1 o x2) y ...
p cnf 4 4
1 2 0
-1 3 0
-1 -3 0
-2 4 0
```

- `c`: comentario.
- `p cnf V C`: encabezado.
- Cada cláusula termina en 0.

terminal

```
$ minisat ejemplos/formula.cnf build/formula.res
...
SATISFIABLE
$ echo $?
10
$ cat build/formula.res
SAT
-1 2 -3 4 0
```

- **Código de salida:** 10 si es SAT, 20 si es UNSAT.
- El modelo va en el **archivo**.

DIMACS y minisat

ejemplos/formula.cnf

```
c (x1 o x2) y ...
p cnf 4 4
1 2 0
-1 3 0
-1 -3 0
-2 4 0
```

- `c`: comentario.
- `p cnf V C`: encabezado.
- Cada cláusula termina en 0.

terminal

```
$ minisat ejemplos/formula.cnf build/formula.res
...
SATISFIABLE
$ echo $?
10
$ cat build/formula.res
SAT
-1 2 -3 4 0
```

- **Código de salida:** 10 si es SAT, 20 si es UNSAT.
- El modelo va en el **archivo**.
- No promete cuál modelo.

Dos trampas de formato

terminal

```
$ cat ejemplos/bad_header.cnf
c el encabezado miente: dice 5 variables y 9 cláusulas
p cnf 5 9
1 -2 0
$ minisat -verb=0 ejemplos/bad_header.cnf build/bad.res
WARNING! DIMACS header mismatch: wrong number of variables.
WARNING! DIMACS header mismatch: wrong number of clauses.
SATISFIABLE
$ cat build/bad.res
SAT
1 -2 0
```

Dos trampas de formato

terminal

```
$ cat ejemplos/bad_header.cnf
c el encabezado miente: dice 5 variables y 9 cláusulas
p cnf 5 9
1 -2 0
$ minisat -verb=0 ejemplos/bad_header.cnf build/bad.res
WARNING! DIMACS header mismatch: wrong number of variables.
WARNING! DIMACS header mismatch: wrong number of clauses.
SATISFIABLE
$ cat build/bad.res
SAT
1 -2 0
```

- Un encabezado equivocado solo da un aviso.

Dos trampas de formato

terminal

```
$ cat ejemplos/bad_header.cnf
c el encabezado miente: dice 5 variables y 9 cláusulas
p cnf 5 9
1 -2 0
$ minisat -verb=0 ejemplos/bad_header.cnf build/bad.res
WARNING! DIMACS header mismatch: wrong number of variables.
WARNING! DIMACS header mismatch: wrong number of clauses.
SATISFIABLE
$ cat build/bad.res
SAT
1 -2 0
```

- Un encabezado equivocado solo da un aviso.
- Una variable que no sale en ninguna cláusula tampoco sale en la respuesta: lean hasta el 0.

Coloreo: exámenes sin choques

Cada curso es un nodo. Dos cursos con un alumno en común son vecinos. Con c turnos, ¿se colorea con c colores?

Variables: $x(v, a)$, "el nodo v tiene el color a". Son $V \cdot c$. En DIMACS, $v \cdot c + a + 1$.

Reglas:

Coloreo: exámenes sin choques

Cada curso es un nodo. Dos cursos con un alumno en común son vecinos. Con c turnos, ¿se colorea con c colores?

Variables: $x(v, a)$, "el nodo v tiene el color a". Son $V \cdot c$. En DIMACS, $v \cdot c + a + 1$.

Reglas:

- Cada nodo, al menos un color: $x(v, 0) \vee \cdots \vee x(v, c-1)$.

Coloreo: exámenes sin choques

Cada curso es un nodo. Dos cursos con un alumno en común son vecinos. Con c turnos, ¿se colorea con c colores?

Variables: $x(v, a)$, "el nodo v tiene el color a". Son $V \cdot c$. En DIMACS, $v \cdot c + a + 1$.

Reglas:

- Cada nodo, al menos un color: $x(v, 0) \vee \cdots \vee x(v, c-1)$.
- Vecinos sin color común: $\neg x(u, a) \vee \neg x(v, a)$, por cada arista y color.

Coloreo: exámenes sin choques

Cada curso es un nodo. Dos cursos con un alumno en común son vecinos. Con c turnos, ¿se colorea con c colores?

Variables: $x(v, a)$, "el nodo v tiene el color a". Son $V \cdot c$. En DIMACS, $v \cdot c + a + 1$.

Reglas:

- Cada nodo, al menos un color: $x(v, 0) \vee \cdots \vee x(v, c-1)$.
- Vecinos sin color común: $\neg x(u, a) \vee \neg x(v, a)$, por cada arista y color.
- Cada nodo, a lo sumo un color: $\neg x(v, a) \vee \neg x(v, b)$, para $a < b$.

Coloreo: exámenes sin choques

Cada curso es un nodo. Dos cursos con un alumno en común son vecinos. Con c turnos, ¿se colorea con c colores?

Variables: $x(v, a)$, "el nodo v tiene el color a". Son $V \cdot c$. En DIMACS, $v \cdot c + a + 1$.

Reglas:

- Cada nodo, al menos un color: $x(v, 0) \vee \cdots \vee x(v, c-1)$.
- Vecinos sin color común: $\neg x(u, a) \vee \neg x(v, a)$, por cada arista y color.
- Cada nodo, a lo sumo un color: $\neg x(v, a) \vee \neg x(v, b)$, para $a < b$.

Cláusulas: $V + E \cdot c + V \cdot c(c-1)/2$. Polinomial.

Coloreo: las dos direcciones

Teorema

G se colorea con *c* colores si y solo si la fórmula es satisfacible.

Coloreo: las dos direcciones

Teorema

G se colorea con c colores si y solo si la fórmula es satisfacible.

De coloreo a modelo. $x(v, a)$ verdadera cuando $col(v) = a$.

Coloreo: las dos direcciones

Teorema

G se colorea con *c* colores si y solo si la fórmula es satisfacible.

De coloreo a modelo. $x(v, a)$ verdadera cuando $col(v) = a$. Cada nodo tiene exactamente un color.

Coloreo: las dos direcciones

Teorema

G se colorea con *c* colores si y solo si la fórmula es satisfacible.

De coloreo a modelo. $x(v, a)$ verdadera cuando $col(v) = a$. Cada nodo tiene exactamente un color. En una arista, $col(u) \neq col(v)$: se cumple su cláusula.

Coloreo: las dos direcciones

Teorema

G se colorea con *c* colores si y solo si la fórmula es satisfacible.

De coloreo a modelo. $x(v, a)$ verdadera cuando $col(v) = a$. Cada nodo tiene exactamente un color. En una arista, $col(u) \neq col(v)$: se cumple su cláusula.

De modelo a coloreo. Cada nodo tiene exactamente un a verdadero. Sea $col(v) = a$.

Coloreo: las dos direcciones

Teorema

G se colorea con *c* colores si y solo si la fórmula es satisfacible.

De coloreo a modelo. $x(v, a)$ verdadera cuando $col(v) = a$. Cada nodo tiene exactamente un color. En una arista, $col(u) \neq col(v)$: se cumple su cláusula.

De modelo a coloreo. Cada nodo tiene exactamente un a verdadero. Sea $col(v) = a$. Si $\{u, v\}$ es arista y $col(u) = col(v) = a$, entonces $x(u, a)$ y $x(v, a)$ son verdaderas,

Coloreo: las dos direcciones

Teorema

G se colorea con *c* colores si y solo si la fórmula es satisfacible.

De coloreo a modelo. $x(v, a)$ verdadera cuando $col(v) = a$. Cada nodo tiene exactamente un color. En una arista, $col(u) \neq col(v)$: se cumple su cláusula.

De modelo a coloreo. Cada nodo tiene exactamente un a verdadero. Sea $col(v) = a$. Si $\{u, v\}$ es arista y $col(u) = col(v) = a$, entonces $x(u, a)$ y $x(v, a)$ son verdaderas, y se viola la cláusula de esa arista y ese color. ■

Coloreo en C++

coloring.h

```cpp
inline int color_variable(int node, int color, int colors) { return node * colors + color + 1; }

inline Cnf encode_coloring(const Graph& g, int colors, bool at_most_one) {
    Cnf formula;
    formula.variables = g.n * colors;
    std::vector<int> vars(colors);
    for (int v = 0; v < g.n; v++) {
        for (int c = 0; c < colors; c++) vars[c] = color_variable(v, c, colors);
        add_at_least_one(formula, vars);
        if (at_most_one) add_at_most_one(formula, vars);
    }
    for (auto [u, v] : g.edges) {
        for (int c = 0; c < colors; c++)
            formula.add({-color_variable(u, c, colors), -color_variable(v, c, colors)});
    }
    return formula;
}
```

Coloreo con minisat

build/coloring_demo, extracto

```
minisat terminó con el código 10. El archivo de respuesta dice:
SAT
-1 -2 3 4 -5 -6 -7 8 -9 0

ciclo de 5 con 2 colores: 10 variables, 20 cláusulas, código 20: no se puede
ciclo de 5 con 3 colores: 15 variables, 35 cláusulas, código 10: 1 2 1 0 2 (propio)
K4 con 3 colores: 12 variables, 34 cláusulas, código 20: no se puede
K4 con 4 colores: 16 variables, 52 cláusulas, código 10: 2 3 0 1 (propio)
Petersen con 2 colores: 20 variables, 50 cláusulas, código 20: no se puede
Petersen con 3 colores: 30 variables, 85 cláusulas, código 10: 1 0 2 1 0 2 1 1 0 2 (propio)
```

Coloreo con minisat

build/coloring_demo, extracto

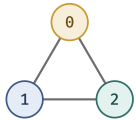
```
minisat terminó con el código 10. El archivo de respuesta dice:
SAT
-1 -2 3 4 -5 -6 -7 8 -9 0

ciclo de 5 con 2 colores: 10 variables, 20 cláusulas, código 20: no se puede
ciclo de 5 con 3 colores: 15 variables, 35 cláusulas, código 10: 1 2 1 0 2 (propio)
K4 con 3 colores: 12 variables, 34 cláusulas, código 20: no se puede
K4 con 4 colores: 16 variables, 52 cláusulas, código 10: 2 3 0 1 (propio)
Petersen con 2 colores: 20 variables, 50 cláusulas, código 20: no se puede
Petersen con 3 colores: 30 variables, 85 cláusulas, código 10: 1 0 2 1 0 2 1 1 0 2 (propio)
```

Triángulo: las verdaderas son 3, 4 y 8. Con $v \cdot 3 + c + 1$: nodo 0 color 2, nodo 1 color 0, nodo 2 color 1.

El triángulo en CNF

el grafo



`x(v, a) = v · 3 + a + 1`

	color 0	color 1	color 2
nodo 0	1	2	3
nodo 1	4	5	6
nodo 2	7	8	9

El modelo de minisat

```
-1 -2 3 4 -5 -6 -7 8 -9
```

verdaderas: 3, 4 y 8

```
3: nodo 0, color 2
4: nodo 1, color 0
8: nodo 2, color 1
```

Cada nodo, al menos un color
3 cláusulas: 1 por nodo



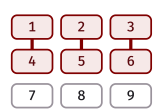

```
0: (1 2 3)
1: (4 5 6)
2: (7 8 9)
```

Cada nodo, a lo sumo un color
9 cláusulas: 3 por nodo




```
0: (-1 -2) (-1 -3) (-2 -3)
1: (-4 -5) (-4 -6) (-5 -6)
2: (-7 -8) (-7 -9) (-8 -9)
```

Dos vecinos no comparten color
9 cláusulas: 3 por arista




```
0-1: (-1 -4) (-2 -5) (-3 -6)
1-2: (-4 -7) (-5 -8) (-6 -9)
0-2: (-1 -7) (-2 -8) (-3 -9)
```

`p cnf 9 21` 9 variables, y 3 + 9 + 9 = 21 cláusulas. En DIMACS, cada paréntesis lleva un 0 al final.

SECCIÓN 12

Al menos uno y a lo sumo uno

12

No siempre es un bug

El error clásico: escribir “al menos uno” y olvidar “a lo sumo uno”.

No siempre es un bug

El error clásico: escribir “al menos uno” y olvidar “a lo sumo uno”.

1. **No hace daño:** en el coloreo.

No siempre es un bug

El error clásico: escribir “al menos uno” y olvidar “a lo sumo uno”.

1. **No hace daño:** en el coloreo.
2. **Ya está implícita:** con el principio del palomar.

No siempre es un bug

El error clásico: escribir “al menos uno” y olvidar “a lo sumo uno”.

1. **No hace daño:** en el coloreo.
2. **Ya está implícita:** con el principio del palomar.
3. **Rompe la respuesta:** si no queda ninguna regla de “a lo sumo”.

No siempre es un bug

El error clásico: escribir “al menos uno” y olvidar “a lo sumo uno”.

1. **No hace daño:** en el coloreo.
2. **Ya está implícita:** con el principio del palomar.
3. **Rompe la respuesta:** si no queda ninguna regla de “a lo sumo”.

Lo decide la **demostración de la dirección 2**: de un modelo a una solución.

Primer caso: en el coloreo no hace daño

Afirmación. Sin la regla, escoger cualquier color verdadero de cada nodo da un coloreo propio.

build/coloring_demo, extracto

```
Sin "a lo sumo un color por nodo":
Petersen con 3 colores: 30 variables, 55 cláusulas, código 10: 2 1 2 1 0 1 0 0 2 2 (propio)
Una arista con 3 colores tiene 6 modelos con la regla y 12 sin ella.
Un modelo con dos colores en un nodo: nodo 0 = { 1 2 } nodo 1 = { 0 }
De 12 modelos sin la regla, en 12 cualquier elección da un coloreo propio.
```

Primer caso: en el coloreo no hace daño

Afirmación. Sin la regla, escoger cualquier color verdadero de cada nodo da un coloreo propio.

Demostración directa. Si $\{u, v\}$ es arista y les tocó el mismo a, entonces $x(u, a)$ y $x(v, a)$ son verdaderas: viola la cláusula de esa arista. ■

build/coloring_demo, extracto

```
Sin "a lo sumo un color por nodo":
Petersen con 3 colores: 30 variables, 55 cláusulas, código 10: 2 1 2 1 0 1 0 0 2 2 (propio)
Una arista con 3 colores tiene 6 modelos con la regla y 12 sin ella.
Un modelo con dos colores en un nodo: nodo 0 = { 1 2 } nodo 1 = { 0 }
De 12 modelos sin la regla, en 12 cualquier elección da un coloreo propio.
```

Segundo caso: el palomar

Cuadrado latino: cada fila y cada columna tiene cada símbolo una vez. Un sudoku sin cajas.

Reglas: cada celda, al menos un símbolo. Cada símbolo, a lo sumo una vez por fila.

build/latin_demo, extracto

```
n = 2: 2 modelos, 0 con dos símbolos en una celda
n = 3: 6 modelos, 0 con dos símbolos en una celda
n = 4: 24 modelos, 0 con dos símbolos en una celda
```

Segundo caso: el palomar

Cuadrado latino: cada fila y cada columna tiene cada símbolo una vez. Un sudoku sin cajas.

Reglas: cada celda, al menos un símbolo. Cada símbolo, a lo sumo una vez por fila.

Afirmación. Cada celda tiene exactamente un símbolo.

build/latin_demo, extracto

```
n = 2: 2 modelos, 0 con dos símbolos en una celda
n = 3: 6 modelos, 0 con dos símbolos en una celda
n = 4: 24 modelos, 0 con dos símbolos en una celda
```

Segundo caso: el palomar

Cuadrado latino: cada fila y cada columna tiene cada símbolo una vez. Un sudoku sin cajas.

Reglas: cada celda, al menos un símbolo. Cada símbolo, a lo sumo una vez por fila.

Afirmación. Cada celda tiene exactamente un símbolo.

- En una fila, las verdaderas son **al menos** *n*: hay *n* celdas, cada una con al menos uno.

build/latin_demo, extracto

```
n = 2: 2 modelos, 0 con dos símbolos en una celda
n = 3: 6 modelos, 0 con dos símbolos en una celda
n = 4: 24 modelos, 0 con dos símbolos en una celda
```

Segundo caso: el palomar

Cuadrado latino: cada fila y cada columna tiene cada símbolo una vez. Un sudoku sin cajas.

Reglas: cada celda, al menos un símbolo. Cada símbolo, a lo sumo una vez por fila.

Afirmación. Cada celda tiene exactamente un símbolo.

- En una fila, las verdaderas son **al menos** n: hay n celdas, cada una con al menos uno.
- Son **a lo sumo** n: hay n símbolos, cada uno a lo sumo una vez.

build/latin_demo, extracto

```
n = 2: 2 modelos, 0 con dos símbolos en una celda
n = 3: 6 modelos, 0 con dos símbolos en una celda
n = 4: 24 modelos, 0 con dos símbolos en una celda
```

Segundo caso: el palomar

Cuadrado latino: cada fila y cada columna tiene cada símbolo una vez. Un sudoku sin cajas.

Reglas: cada celda, al menos un símbolo. Cada símbolo, a lo sumo una vez por fila.

Afirmación. Cada celda tiene exactamente un símbolo.

- En una fila, las verdaderas son **al menos** n: hay n celdas, cada una con al menos uno.
- Son **a lo sumo** n: hay n símbolos, cada uno a lo sumo una vez.
- Dos símbolos en una celda darían $n + 1$ o más. ■

build/latin_demo, extracto

```
n = 2: 2 modelos, 0 con dos símbolos en una celda
n = 3: 6 modelos, 0 con dos símbolos en una celda
n = 4: 24 modelos, 0 con dos símbolos en una celda
```

Tercer caso: se rompe

Sin **ninguna** regla de “a lo sumo”, todas las variables verdaderas son un modelo.

build/latin_demo, extracto

```
El mismo cuadrado con 1 en (0, 0) y 1 en (0, 1), que no tiene solución:
reglas completas: 27 variables, 110 cláusulas, minisat dice UNSAT
celda al menos uno, línea a lo sumo uno: 27 variables, 65 cláusulas, minisat dice UNSAT
solo al menos uno: 27 variables, 29 cláusulas, minisat dice SAT
  1 1 2
  3 3 1
  2 2 1
  NO es un cuadrado latino
```

Tercer caso: se rompe

Sin **ninguna** regla de “a lo sumo”, todas las variables verdaderas son un modelo.

build/latin_demo, extracto

```
El mismo cuadrado con 1 en (0, 0) y 1 en (0, 1), que no tiene solución:
reglas completas: 27 variables, 110 cláusulas, minisat dice UNSAT
celda al menos uno, línea a lo sumo uno: 27 variables, 65 cláusulas, minisat dice UNSAT
solo al menos uno: 27 variables, 29 cláusulas, minisat dice SAT
  1 1 2
  3 3 1
  2 2 1
  NO es un cuadrado latino
```

Error común: declarar satisfacible lo imposible. El criterio no es “escribir todo”: es hacer la demostración de la dirección 2 con las cláusulas escritas.

SECCIÓN 13

Laboratorio

13

Laboratorio: resolver sudokus con SAT

`lab/sudoku_sat`. Un sudoku de orden k tiene $n = k \cdot k$ filas, columnas y dígitos.

- **encode:** una variable por cada (fila, columna, dígito), y la fórmula en DIMACS.
- **decode:** lee la respuesta de minisat y escribe el tablero, o `imposible`.
- `driver.h` las conecta: el juez corre un solo programa, y revisa el encabezado.

Laboratorio: resolver sudokus con SAT

`lab/sudoku_sat`. Un sudoku de orden k tiene $n = k \cdot k$ filas, columnas y dígitos.

- **encode:** una variable por cada (fila, columna, dígito), y la fórmula en DIMACS.
- **decode:** lee la respuesta de minisat y escribe el tablero, o `imposible`.
- `driver.h` las conecta: el juez corre un solo programa, y revisa el encabezado.

Para comparar: variables, cláusulas y tiempo, contra un backtracking. Sirve el suyo de Introducción a la Programación, o `backtrack_bench`.

Laboratorio: resolver sudokus con SAT

`lab/sudoku_sat`. Un sudoku de orden k tiene $n = k \cdot k$ filas, columnas y dígitos.

- **encode:** una variable por cada (fila, columna, dígito), y la fórmula en DIMACS.
- **decode:** lee la respuesta de minisat y escribe el tablero, o `imposible`.
- `driver.h` las conecta: el juez corre un solo programa, y revisa el encabezado.

Para comparar: variables, cláusulas y tiempo, contra un backtracking. Sirve el suyo de Introducción a la Programación, o `backtrack_bench`.

Para probar: quiten cada regla de "a lo sumo uno". ¿Cuáles cambian el veredicto, y por qué?

Un backtracking para comparar

build/backtrack_bench

```
valores dados: 17
resultado: tiene solución
nodos: 69175317
segundos: 3.9239
```

`std::chrono::steady_clock`, con `-O2`, en el contenedor, en una máquina compartida y una sola corrida. Depende de la máquina. Otra corrida dio poco más de 3 segundos. Los nodos no dependen de ella.

Difícil **para este orden de búsqueda**, por filas y con los dígitos en orden. No es una medida de dificultad en general.

Resumen

- **P:** decisión con algoritmo polinomial en los bits. **NP:** cada "sí" tiene un certificado corto, y un verificador polinomial que no se deja engañar. P $\subseteq$ NP.
- **P contra NP** sigue abierto. La mayoría cree que P $\neq$ NP.
- $A \leq_p B$: f polinomial, con "sí" $\iff$ "sí". Si B está en P, A también. Es transitiva.
- **Dirección:** para mostrar que N es difícil, $D \leq_p N$, del conocido al nuevo.
- $O(nW)$ no es polinomial en bits: no dice nada de P = NP.
- S independiente $\iff$ $V - S$ cobertura $\iff$ S clique en $\bar{G}$.
- **Patrón:** f sin saber la respuesta, polinomial, y dos direcciones.
- **SAT:** DIMACS, minisat, 10 y 20. Práctico y sin garantía.
- **Al menos uno sin a lo sumo uno:** inocuo, implicado o bug. Lo decide la demostración.

Para la próxima clase

- Terminar `lab/sudoku_sat`, con la comparación.
- Leer los apuntes de la semana 14: NP-completitud.
- Repaso: ¿qué es un certificado para conjunto independiente? Si $A \leq_p B$ y A es difícil, ¿qué se sabe de B? Escriban en CNF "exactamente uno de x, y, z".
- Pensar: si el conjunto independiente es NP-completo, ¿qué más hace falta para decir lo mismo de la cobertura de vértices y del clique?