# Optimización III (525551)

## Complejidad temporal

Pr. Julio Aracena.

Departamento de Ingeniería Matemática
Facultad de Ciencias Físicas y Matemáticas
Universidad de Concepción

Primer semestre, 2021

# Problemas de decisión

**Definición:** Un problema combinatorial abstracto $Q$ puede ser visto como una relación entre objetos matemáticos (grafos, matrices, números, etc.) $O_1 \times \cdots \times O_k$ y un conjunto de soluciones $S$. Es decir,

$$Q \subseteq O_1 \times \cdots \times O_k \times S.$$

dependiendo de $S$, $Q$ puede ser de conteo, existencia, optimización, etc. Diremos que $Q \subseteq O_1 \times \cdots \times O_k \times S$ es un problema de decisión si $S = \{s, n\}$ y $Q : Dom(Q) \subseteq (O_1 \times \cdots \times O_k) \to S$ es una función.

# Problemas de decisión

**Definición:** Un problema combinatorial abstracto $Q$ puede ser visto como una relación entre objetos matemáticos (grafos, matrices, números, etc.) $O_1 \times \cdots \times O_k$ y un conjunto de soluciones $S$. Es decir,

$$Q \subseteq O_1 \times \cdots \times O_k \times S.$$

dependiendo de $S$, $Q$ puede ser de conteo, existencia, optimización, etc. Diremos que $Q \subseteq O_1 \times \cdots \times O_k \times S$ es un problema de decisión si $S = \{s, n\}$ y $Q : Dom(Q) \subseteq (O_1 \times \cdots \times O_k) \to S$ es una función.

**Ejemplo:**

- Se define **PATH**:=$\{(G, u, v, x)$ : G=(V,E) un grafo (dirigido o no) $u, v \in V, \; x \in \{s, n\}, \; x = s \iff G$ tiene un camino de $u$ a $v\}$.

# Problemas de decisión

**Definición:** Un problema combinatorial abstracto $Q$ puede ser visto como una relación entre objetos matemáticos (grafos, matrices, números, etc.) $O_1 \times \cdots \times O_k$ y un conjunto de soluciones $S$. Es decir,

$$Q \subseteq O_1 \times \cdots \times O_k \times S.$$

dependiendo de $S$, $Q$ puede ser de conteo, existencia, optimización, etc. Diremos que $Q \subseteq O_1 \times \cdots \times O_k \times S$ es un problema de decisión si $S = \{s, n\}$ y $Q : Dom(Q) \subseteq (O_1 \times \cdots \times O_k) \to S$ es una función.

**Ejemplo:**

- Se define **PATH**:=$\{(G, u, v, x)$ : G=(V,E) un grafo (dirigido o no) $u, v \in V,\ x \in \{s, n\},\ x = s \iff G$ tiene un camino de $u$ a $v\}$.
- Otra forma más coloquial, se define:
  PATH: Dado $G = (V, E)$ un grafo (dirigido o no) y $u, v \in V$. ¿Existe un camino de $u$ a $v$ en $G$?

# Problemas de decisión

**Definición:** Un problema combinatorial abstracto $Q$ puede ser visto como una relación entre objetos matemáticos (grafos, matrices, números, etc.) $O_1 \times \cdots \times O_k$ y un conjunto de soluciones $S$. Es decir,

$$Q \subseteq O_1 \times \cdots \times O_k \times S.$$

dependiendo de $S$, $Q$ puede ser de conteo, existencia, optimización, etc. Diremos que $Q \subseteq O_1 \times \cdots \times O_k \times S$ es un problema de decisión si $S = \{s, n\}$ y $Q : Dom(Q) \subseteq (O_1 \times \cdots \times O_k) \to S$ es una función.

**Ejemplo:**

- Se define **PATH**:=$\{(G, u, v, x)$ : G=(V,E) un grafo (dirigido o no) $u, v \in V$, $x \in \{s, n\}$, $x = s \iff G$ tiene un camino de $u$ a $v\}$.
- Otra forma más coloquial, se define:
  PATH: Dado $G = (V, E)$ un grafo (dirigido o no) y $u, v \in V$. ¿Existe un camino de $u$ a $v$ en $G$?
- **PRIME**: Dado $n \in \mathbb{N}$, ¿Es $n$ un número primo?

# Problemas de decisión polinomial

**Definición:** Sea $Q$ un problema de decisión. Cualquier elemento $x \in Dom(Q)$ se denomina una instancia de $Q$. $x \in I_Q$ se dice instancia positiva si $Q(x) = s$, e instancia negativa en caso contrario. El conjunto de instancias de $Q$ (ie $Dom(Q)$) se denotará $I_Q$. Diremos que $Q$ es un **problema polinomial** si existe un algoritmo polinomial $A$ con entrada $x \in I_Q$ y salida $S = \{s, n\}$ tal que

$$\forall x \in I_Q,\ A(x) = s \Longleftrightarrow Q(x) = s.$$

# Problemas de decisión polinomial

**Definición:** Sea $Q$ un problema de decisión. Cualquier elemento $x \in Dom(Q)$ se denomina una instancia de $Q$. $x \in I_Q$ se dice instancia positiva si $Q(x) = s$, e instancia negativa en caso contrario. El conjunto de instancias de $Q$ (ie $Dom(Q)$) se denotará $I_Q$. Diremos que $Q$ es un **problema polinomial** si existe un algoritmo polinomial $A$ con entrada $x \in I_Q$ y salida $S = \{s, n\}$ tal que

$$\forall x \in I_Q,\ A(x) = s \Longleftrightarrow Q(x) = s.$$

**Ejemplo:** PATH es polinomial. Sea el siguiente algoritmo:

**Algorithm** PATH(G,u,v)

**Input:** $G = (V, E)$ un grafo (dirigido o no) y $u, v \in V$

```
Ejecutar Exploración Grafo (G,u)
if v fue marcado then
    return s
end if
return n
```

Como Exploración Grafo es un algoritmo lineal, el algoritmo PATH es polinomial y obviamente resuelve el problema PATH.

# Problemas de decisión polinomial

**Ejemplo:** Otros ejemplos de problemas de decisión polinomial (ejercicio):

- **PRIME**: Dado $n \in \mathbb{N}$, ¿Es $n$ un número primo?

# Problemas de decisión polinomial

**Ejemplo:** Otros ejemplos de problemas de decisión polinomial (ejercicio):

- **PRIME**: Dado $n \in \mathbb{N}$, ¿Es $n$ un número primo?
- **RELATIVE PRIMES**: Dado $m, n \in \mathbb{N}$, ¿Son $m$ y $n$ primos relativos?

# Problemas de decisión polinomial

**Ejemplo:** Otros ejemplos de problemas de decisión polinomial (ejercicio):

- **PRIME**: Dado $n \in \mathbb{N}$, ¿Es $n$ un número primo?
- **RELATIVE PRIMES**: Dado $m, n \in \mathbb{N}$, ¿Son $m$ y $n$ primos relativos?
- **CONNECTED**: Dado $G$ un grafo (dirigido o no), ¿Es $G$ un grafo conexo?

# Problemas de decisión polinomial

**Ejemplo:** Otros ejemplos de problemas de decisión polinomial (ejercicio):

- **PRIME**: Dado $n \in \mathbb{N}$, ¿Es $n$ un número primo?
- **RELATIVE PRIMES**: Dado $m, n \in \mathbb{N}$, ¿Son $m$ y $n$ primos relativos?
- **CONNECTED**: Dado $G$ un grafo (dirigido o no), ¿Es $G$ un grafo conexo?
- **STRONGLY CONNECTED**: Dado $G$ un grafo dirigido, ¿Es $G$ fuertemente conexo?

# Problemas de decisión polinomial

**Ejemplo:** Otros ejemplos de problemas de decisión polinomial (ejercicio):

- **PRIME**: Dado $n \in \mathbb{N}$, ¿Es $n$ un número primo?
- **RELATIVE PRIMES**: Dado $m, n \in \mathbb{N}$, ¿Son $m$ y $n$ primos relativos?
- **CONNECTED**: Dado $G$ un grafo (dirigido o no), ¿Es $G$ un grafo conexo?
- **STRONGLY CONNECTED**: Dado $G$ un grafo dirigido, ¿Es $G$ fuertemente conexo?
- **ACYCLIC GRAPH**: Dado $G$ un grafo (dirigido o no), ¿Es $G$ un grafo acíclico (sin ciclos)?

# Problemas NP

Existen problemas de decisión para los que no se sabe si son polinomiales y más aún se conjetura que no lo son.

# Problemas NP

Existen problemas de decisión para los que no se sabe si son polinomiales y más aún se conjetura que no lo son.

**Ejemplo:**
**HAMILTONIAN**: Dado un grafo (dirigido o no) $G = (V, E)$, ¿Es $G$ Hamiltoniano?

# Problemas NP

Existen problemas de decisión para los que no se sabe si son polinomiales y más aún se conjetura que no lo son.

**Ejemplo:**
**HAMILTONIAN**: Dado un grafo (dirigido o no) $G = (V, E)$, ¿Es $G$ Hamiltoniano?

**Definición:** Un problema de decisión $Q$ se dice **NP** (non determinista polynomial) si existe $k \in \mathbb{N}$ y un algoritmo polinomial $A(x, y)$ que recibe como entrada $x \in I_Q$ y un certificado (información adicional) $y \in \{0, 1\}^*$ con $size(y) = O(size(x)^k)$ tal que $\forall x \in I_Q$ :

$$Q(x) = s \Longleftrightarrow \exists y \in \{0,1\}^*,\ size(y) = O(size(x)^k),\ A(x, y) = s.$$

# Problemas NP

Existen problemas de decisión para los que no se sabe si son polinomiales y más aún se conjetura que no lo son.

**Ejemplo:**
**HAMILTONIAN**: Dado un grafo (dirigido o no) $G = (V, E)$, ¿Es $G$ Hamiltoniano?

**Definición:** Un problema de decisión $Q$ se dice **NP** (non determinista polynomial) si existe $k \in \mathbb{N}$ y un algoritmo polinomial $A(x, y)$ que recibe como entrada $x \in I_Q$ y un certificado (información adicional) $y \in \{0, 1\}^*$ con $size(y) = O(size(x)^k)$ tal que $\forall x \in I_Q$ :

$$Q(x) = s \iff \exists y \in \{0, 1\}^*,\ size(y) = O(size(x)^k),\ A(x, y) = s.$$

El algoritmo $A$ es llamado **verificador polinomial** de instancias afirmativas de $Q$.

# Problemas NP

Existen problemas de decisión para los que no se sabe si son polinomiales y más aún se conjetura que no lo son.

**Ejemplo:**
**HAMILTONIAN**: Dado un grafo (dirigido o no) $G = (V, E)$, ¿Es $G$ Hamiltoniano?

**Definición:** Un problema de decisión $Q$ se dice **NP** (non determinista polynomial) si existe $k \in \mathbb{N}$ y un algoritmo polinomial $A(x, y)$ que recibe como entrada $x \in I_Q$ y un certificado (información adicional) $y \in \{0, 1\}^*$ con $size(y) = O(size(x)^k)$ tal que $\forall x \in I_Q$ :

$$Q(x) = s \iff \exists y \in \{0,1\}^*,\ size(y) = O(size(x)^k),\ A(x, y) = s.$$

El algoritmo $A$ es llamado **verificador polinomial** de instancias afirmativas de $Q$.

**Observación:** Como $A$ es algoritmo polinomial y $size(y)$ es polinomial en $size(x)$, entonces $A$ es polinomial en $size(x)$. Es decir, $A(x, y)$ retorna $s$ o $n$ en tiempo polinomial con respecto a $size(x)$ independiente de $y$.

# Problemas NP

**Ejemplo:** Para el problema HAMILTONIAN se tiene que dado $G = (V, E)$ un grafo e $y = v_1, v_2, \ldots, v_n, v_1$ una secuencia de vértices de $G$ con $size(y) = O(|V|)$, ie $size(y) = O(size(G))$, se define el siguiente algoritmo $A$:

---

**Algorithm** A(G,y)

---

**Input:** $G = (V, E)$ un grafo (dirigido o no),
$y = v_1, \ldots, v_n, v_1$ secuencia de vértices de $G$

1: **if** $y$ es un ciclo en $G \ \wedge \ \{v_1, \ldots, v_n\} = V$ **then**
2:     **return** $s$
3: **end if**
4: **return** $n$

---

Luego, $A$ es obviamente polinomial, pues ambas condiciones de la línea 1 se pueden verificar en tiempo lineal con respecto al $size(G)$, y $G$ es Hamiltoniano si y sólo si $\exists y, \ A(G, y) = s$. Por lo tanto, HAMILTONIAN es NP.

# Problemas NP

**Ejemplo:**

**CLIQUE**: Dado un grafo no dirigido $G = (V, E)$ y $k \in \mathbb{N}$, ¿Tiene $G$ un clique (subgrafo completo) de tamaño $k$?

# Problemas NP

**Ejemplo:**

**CLIQUE**: Dado un grafo no dirigido $G = (V, E)$ y $k \in \mathbb{N}$, ¿Tiene $G$ un clique (subgrafo completo) de tamaño $k$?

Dado $G = (V, E)$ un grafo no dirigido, $k \in \mathbb{N}$ e $y = \{v_1, v_2, \ldots, v_k\}$ un subconjunto de $k$ vértices de $G$ con $size(y) = O(size(G, k))$, se define el siguiente algoritmo $A$:

# Problemas NP

**Ejemplo:**

**CLIQUE**: Dado un grafo no dirigido $G = (V, E)$ y $k \in \mathbb{N}$, ¿Tiene $G$ un clique (subgrafo completo) de tamaño $k$?

Dado $G = (V, E)$ un grafo no dirigido, $k \in \mathbb{N}$ e $y = \{v_1, v_2, \ldots, v_k\}$ un subconjunto de $k$ vértices de $G$ con $size(y) = O(size(G, k))$, se define el siguiente algoritmo $A$:

**Algorithm** A(G,k,y)

**Input:** $G = (V, E)$ un grafo no dirigido, $k \in \mathbb{N}$,
$y = \{v_1, \ldots, v_k\}$ un subconjunto de vértices de $G$

```
if y es un conjunto de k vértices de G then
    if ∀i ≠ j ∈ {1, ..., k}, {v_i, v_j} ∈ E then
        return s
    end if
end if
return n
```

# Problemas NP

**Ejemplo:**
**CLIQUE**: Dado un grafo no dirigido $G = (V, E)$ y $k \in \mathbb{N}$, ¿Tiene $G$ un clique (subgrafo completo) de tamaño $k$?

Dado $G = (V, E)$ un grafo no dirigido, $k \in \mathbb{N}$ e $y = \{v_1, v_2, \dots, v_k\}$ un subconjunto de $k$ vértices de $G$ con $size(y) = O(size(G, k))$, se define el siguiente algoritmo $A$:

---

**Algorithm** A(G,k,y)

---

**Input:** $G = (V, E)$ un grafo no dirigido, $k \in \mathbb{N}$,
$y = \{v_1, \dots, v_k\}$ un subconjunto de vértices de $G$

1: **if** $y$ es un conjunto de $k$ vértices de $G$ **then**
2:     **if** $\forall i \neq j \in \{1, \dots, k\}$, $\{v_i, v_j\} \in E$ **then**
3:         **return** s
4:     **end if**
5: **end if**
6: **return** $n$

---

Como verificar las condiciones de las líneas 1 y 2 se puede hacer en tiempo polinomial (cuadrático) con respecto al tamaño de la entrada, entonces $A$ es polinomial. Además, obviamente $G$ tiene un k-clique si y sólo si $\exists y, A(G, k, y) = s$. Luego, CLIQUE es NP.

# Problemas NP

**Observaciones:**

- No confundir el problema CLIQUE con el problema k-CLIQUE definido por:

# Problemas NP

**Observaciones:**

- No confundir el problema CLIQUE con el problema k-CLIQUE definido por:
  **k-CLIQUE**: Dado un grafo no dirigido $G = (V, E)$ ¿Tiene $G$ un clique de tamaño $k$?

# Problemas NP

**Observaciones:**

- No confundir el problema CLIQUE con el problema k-CLIQUE definido por:
  **k-CLIQUE**: Dado un grafo no dirigido $G = (V, E)$ ¿Tiene $G$ un clique de tamaño $k$?
  En el segundo caso $k$ está fijo, luego no es un parámetro del problema, ie no forma parte de sus instancias.

# Problemas NP

**Observaciones:**

- No confundir el problema CLIQUE con el problema k-CLIQUE definido por:
  **k-CLIQUE**: Dado un grafo no dirigido $G = (V, E)$ ¿Tiene $G$ un clique de tamaño $k$?
  En el segundo caso $k$ está fijo, luego no es un parámetro del problema, ie no forma parte de sus instancias.
- Obviamente k-CLIQUE es un problema polinomial. Basta chequear todos los subconjuntos de $k$ vértices de $G$, que son $\binom{n}{k} = O(n^k)$ conjuntos, para determinar si hay un clique en $G$ de tamaño $k$.

# Problemas NP

**Observaciones:**

- No confundir el problema CLIQUE con el problema k-CLIQUE definido por:
  **k-CLIQUE**: Dado un grafo no dirigido $G = (V, E)$ ¿Tiene $G$ un clique de tamaño $k$?
  En el segundo caso $k$ está fijo, luego no es un parámetro del problema, ie no forma parte de sus instancias.
- Obviamente k-CLIQUE es un problema polinomial. Basta chequear todos los subconjuntos de $k$ vértices de $G$, que son $\binom{n}{k} = O(n^k)$ conjuntos, para determinar si hay un clique en $G$ de tamaño $k$.
- Es un problema abierto saber si CLIQUE es polinomial. Se conjetura que no lo es.

# Problemas co-NP

**Definición:** Sea $Q$ un problema combinatorial de decisión. Se define el problema $\overline{Q}$ con $I_{\overline{Q}} = I_Q$ por:

$$\forall x \in I_{\overline{Q}}, \quad \overline{Q}(x) = s \Longleftrightarrow Q(x) = n.$$

# Problemas co-NP

**Definición:** Sea $Q$ un problema combinatorial de decisión. Se define el problema $\overline{Q}$ con $I_{\overline{Q}} = I_Q$ por:

$$\forall x \in I_{\overline{Q}}, \quad \overline{Q}(x) = s \Longleftrightarrow Q(x) = n.$$

Luego, $Q$ se dice que es un problema **Co-NP** si $\overline{Q}$ es NP.

# Problemas co-NP

**Definición:** Sea $Q$ un problema combinatorial de decisión. Se define el problema $\overline{Q}$ con $I_{\overline{Q}} = I_Q$ por:

$$\forall x \in I_{\overline{Q}}, \quad \overline{Q}(x) = s \Longleftrightarrow Q(x) = n.$$

Luego, $Q$ se dice que es un problema **Co-NP** si $\overline{Q}$ es NP.

**Observación:** De la definición se tiene directamente: $\overline{\overline{Q}} = Q$.

# Problemas co-NP

**Definición:** Sea $Q$ un problema combinatorial de decisión. Se define el problema $\overline{Q}$ con $I_{\overline{Q}} = I_Q$ por:

$$\forall x \in I_{\overline{Q}}, \quad \overline{Q}(x) = s \Longleftrightarrow Q(x) = n.$$

Luego, $Q$ se dice que es un problema **Co-NP** si $\overline{Q}$ es NP.
**Observación:** De la definición se tiene directamente: $\overline{\overline{Q}} = Q$.

**Ejemplo:**

1. **COMPOSITE**: Dado $n \in \mathbb{N}$ ¿Es $n$ un número compuesto?

# Problemas co-NP

**Definición:** Sea $Q$ un problema combinatorial de decisión. Se define el problema $\overline{Q}$ con $I_{\overline{Q}} = I_Q$ por:

$$\forall x \in I_{\overline{Q}}, \quad \overline{Q}(x) = s \Longleftrightarrow Q(x) = n.$$

Luego, $Q$ se dice que es un problema **Co-NP** si $\overline{Q}$ es NP.

**Observación:** De la definición se tiene directamente: $\overline{\overline{Q}} = Q$.

**Ejemplo:**

1. **COMPOSITE**: Dado $n \in \mathbb{N}$ ¿Es $n$ un número compuesto?
   Luego, $\overline{\text{PRIME}} = \text{COMPOSITE}$.
   Además, COMPOSITE es obviamente NP

# Problemas co-NP

**Definición:** Sea $Q$ un problema combinatorial de decisión. Se define el problema $\overline{Q}$ con $I_{\overline{Q}} = I_Q$ por:

$$\forall x \in I_{\overline{Q}}, \quad \overline{Q}(x) = s \Longleftrightarrow Q(x) = n.$$

Luego, $Q$ se dice que es un problema **Co-NP** si $\overline{Q}$ es NP.
**Observación:** De la definición se tiene directamente: $\overline{\overline{Q}} = Q$.

**Ejemplo:**

1. **COMPOSITE**: Dado $n \in \mathbb{N}$ ¿Es $n$ un número compuesto?
   Luego, $\overline{\text{PRIME}} = \text{COMPOSITE}$.
   Además, COMPOSITE es obviamente NP (elegir como certificado un divisor no trivial de $n$). De aquí, PRIME is Co-NP. ¿Es PRIME NP?

# Problemas co-NP

**Definición:** Sea $Q$ un problema combinatorial de decisión. Se define el problema $\overline{Q}$ con $I_{\overline{Q}} = I_Q$ por:

$$\forall x \in I_{\overline{Q}}, \quad \overline{Q}(x) = s \Longleftrightarrow Q(x) = n.$$

Luego, $Q$ se dice que es un problema **Co-NP** si $\overline{Q}$ es NP.
**Observación:** De la definición se tiene directamente: $\overline{\overline{Q}} = Q$.

**Ejemplo:**

1. **COMPOSITE**: Dado $n \in \mathbb{N}$ ¿Es $n$ un número compuesto?
   Luego, $\overline{\text{PRIME}} = \text{COMPOSITE}$.
   Además, COMPOSITE es obviamente NP (elegir como certificado un divisor no trivial de $n$). De aquí, PRIME is Co-NP. ¿Es PRIME NP?
2. $\overline{\textbf{HAMILTONIAN}}$: Dado $G$ un grafo (dirigido o no). ¿$G$ no es Hamiltoniano?

# Problemas co-NP

**Definición:** Sea $Q$ un problema combinatorial de decisión. Se define el problema $\overline{Q}$ con $I_{\overline{Q}} = I_Q$ por:

$$\forall x \in I_{\overline{Q}}, \quad \overline{Q}(x) = s \Longleftrightarrow Q(x) = n.$$

Luego, $Q$ se dice que es un problema **Co-NP** si $\overline{Q}$ es NP.
**Observación:** De la definición se tiene directamente: $\overline{\overline{Q}} = Q$.

**Ejemplo:**

1. **COMPOSITE**: Dado $n \in \mathbb{N}$ ¿Es $n$ un número compuesto?
   Luego, $\overline{\text{PRIME}} = \text{COMPOSITE}$.
   Además, COMPOSITE es obviamente NP (elegir como certificado un divisor no trivial de $n$). De aquí, PRIME is Co-NP. ¿Es PRIME NP?
2. $\overline{\textbf{HAMILTONIAN}}$: Dado $G$ un grafo (dirigido o no). ¿$G$ no es Hamiltoniano?
   HAMILTONIAN es co-NP?

# Problemas co-NP

Usualmente, la familia de problemas polinomiales, NP o Co-NP se denota por P, NP y co-NP respectivamente. De esta forma podemos decir que un problema $Q$ es polinomial o simplemente $Q \in P$. Lo mismo si $Q$ es NP o co-NP.

# Problemas co-NP

Usualmente, la familia de problemas polinomiales, NP o Co-NP se denota por P, NP y co-NP respectivamente. De esta forma podemos decir que un problema $Q$ es polinomial o simplemente $Q \in P$. Lo mismo si $Q$ es NP o co-NP.

**Teorema:** *$P \subseteq NP \cap$ co-$NP$.*

# Problemas co-NP

Usualmente, la familia de problemas polinomiales, NP o Co-NP se denota por P, NP y co-NP respectivamente. De esta forma podemos decir que un problema $Q$ es polinomial o simplemente $Q \in P$. Lo mismo si $Q$ es NP o co-NP.

**Teorema:** *$P \subseteq NP \cap co\text{-}NP$.*
**Demo:** Sea $Q \in$ P y $A$ un algoritmo polinomial que recibe $x \in I_Q$ tal que $A(x) = s \Longleftrightarrow Q(x) = s$. Luego, se define el algoritmo $A'$ por:

**Algorithm** A'(x,y)

**Input:** $x \in I_Q$, $y \in \{0, 1\}$ (un bit)

```
Ejecutar A(x)
if A(x) = s then
    return s
else
    return n
end if
```

# Problemas co-NP

Usualmente, la familia de problemas polinomiales, NP o Co-NP se denota por P, NP y co-NP respectivamente. De esta forma podemos decir que un problema $Q$ es polinomial o simplemente $Q \in P$. Lo mismo si $Q$ es NP o co-NP.

**Teorema:** *$P \subseteq NP \cap co\text{-}NP$.*
**Demo:** Sea $Q \in$ P y $A$ un algoritmo polinomial que recibe $x \in I_Q$ tal que $A(x) = s \iff Q(x) = s$. Luego, se define el algoritmo $A'$ por:

**Algorithm** A'(x,y)

**Input:** $x \in I_Q$, $y \in \{0, 1\}$ (un bit)

```
1: Ejecutar A(x)
2: if A(x) = s then
3:     return s
4: else
5:     return n
6: end if
```

Es claro que $A'$ es polinomial y $\forall x \in I_Q,\ Q(x) = s \iff \exists y, A(x, y) = s$, lo que implica que $Q \in NP$. Análogamente se prueba que $Q \in$ co-NP, cambiando $s$ por $n$ en la línea 2.

# Problemas co-NP

**Observaciones:**

- Del teorema anterior se tiene que: PRIME es NP.

# Problemas co-NP

**Observaciones:**

- Del teorema anterior se tiene que: PRIME es NP.
- Todos los problemas polinomiales vistos hasta ahora son NP y co-NP.

# Problemas co-NP

**Observaciones:**

- Del teorema anterior se tiene que: PRIME es NP.
- Todos los problemas polinomiales vistos hasta ahora son NP y co-NP.
- Son problemas abiertos: ¿P=NP?, ¿P=co-NP? y ¿NP=co-NP?

# Problemas co-NP

**Observaciones:**

- Del teorema anterior se tiene que: PRIME es NP.
- Todos los problemas polinomiales vistos hasta ahora son NP y co-NP.
- Son problemas abiertos: ¿P=NP?, ¿P=co-NP? y ¿NP=co-NP?
- Se conjetura que los tres conjuntos: P, NP y co-NP son distintos entre sí.

# Reducción polinomial

**Definición:** Sea $Q$ y $R$ dos problemas de decisión. Diremos que $R$ se reduce polinomialmente a $Q$, denotado por $R \leq_p Q$, si existe una función $f : I_R \to I_Q$, que puede ser calculada por un algoritmo polinomial, tal que:

$$\forall x \in I_R : \quad R(x) = s \Longleftrightarrow Q(f(x)) = s.$$

En este caso decimos que $f$ es una reducción polinomial.

# Reducción polinomial

**Definición:** Sea $Q$ y $R$ dos problemas de decisión. Diremos que $R$ se reduce polinomialmente a $Q$, denotado por $R \leq_p Q$, si existe una función $f : I_R \to I_Q$, que puede ser calculada por un algoritmo polinomial, tal que:

$$\forall x \in I_R : \quad R(x) = s \Longleftrightarrow Q(f(x)) = s.$$

En este caso decimos que $f$ es una reducción polinomial.

**Observación:** : Una reducción polinomial $f$ no necesariamente es inyectiva, ni sobreyectiva, ni menos biyectiva.

# Reducción polinomial

**Definición:** Sea $Q$ y $R$ dos problemas de decisión. Diremos que $R$ se reduce polinomialmente a $Q$, denotado por $R \leq_p Q$, si existe una función $f : I_R \to I_Q$, que puede ser calculada por un algoritmo polinomial, tal que:

$$\forall x \in I_R : \quad R(x) = s \Longleftrightarrow Q(f(x)) = s.$$

En este caso decimos que $f$ es una reducción polinomial.

**Observación:** : Una reducción polinomial $f$ no necesariamente es inyectiva, ni sobreyectiva, ni menos biyectiva.

**Ejemplo:** Se define el siguiente problema:
**VERTEX (NODE) COVER**: Dado $G = (V, E)$ un grafo no dirigido y $k \in \mathbb{N}$, ¿Existe $U \subseteq V$ un vertex cover de $G$, i.e. $\forall \{a, b\} \in E,\ \{a, b\} \cap U \neq \emptyset$, tal que $|U| \leq k$?

# Reducción polinomial

**Definición:** Sea $Q$ y $R$ dos problemas de decisión. Diremos que $R$ se reduce polinomialmente a $Q$, denotado por $R \leq_p Q$, si existe una función $f : I_R \to I_Q$, que puede ser calculada por un algoritmo polinomial, tal que:

$$\forall x \in I_R : \quad R(x) = s \Longleftrightarrow Q(f(x)) = s.$$

En este caso decimos que $f$ es una reducción polinomial.

**Observación:** : Una reducción polinomial $f$ no necesariamente es inyectiva, ni sobreyectiva, ni menos biyectiva.

**Ejemplo:** Se define el siguiente problema:
**VERTEX (NODE) COVER**: Dado $G = (V, E)$ un grafo no dirigido y $k \in \mathbb{N}$, ¿Existe $U \subseteq V$ un vertex cover de $G$, i.e. $\forall \{a, b\} \in E, \ \{a, b\} \cap U \neq \emptyset$, tal que $|U| \leq k$?

Afirmación: CLIQUE $\leq_p$ VERTEX COVER.

# Reducción polinomial

**Definición:** Sea $Q$ y $R$ dos problemas de decisión. Diremos que $R$ se reduce polinomialmente a $Q$, denotado por $R \leq_p Q$, si existe una función $f : I_R \to I_Q$, que puede ser calculada por un algoritmo polinomial, tal que:

$$\forall x \in I_R : \quad R(x) = s \Longleftrightarrow Q(f(x)) = s.$$

En este caso decimos que $f$ es una reducción polinomial.

**Observación:** : Una reducción polinomial $f$ no necesariamente es inyectiva, ni sobreyectiva, ni menos biyectiva.

**Ejemplo:** Se define el siguiente problema:
**VERTEX (NODE) COVER**: Dado $G = (V, E)$ un grafo no dirigido y $k \in \mathbb{N}$, ¿Existe $U \subseteq V$ un vertex cover de $G$, i.e. $\forall \{a, b\} \in E, \ \{a, b\} \cap U \neq \emptyset$, tal que $|U| \leq k$?

Afirmación: CLIQUE $\leq_p$ VERTEX COVER.
Dado $(G = (V, E), k)$ una instancia de CLIQUE, se define una instancia de VERTEX COVER $f(G, k) = (G', k')$ donde $G' = (V', E')$ es un grafo no dirigido con $V' = V$ y $E' = \{\{a, b\} : \{a, b\} \notin E, \ a \neq b\}$ y $k' = |V| - k$. Es fácil ver que $G$ tiene un clique $U$ con $|U| = k$ si y sólo si $V' \setminus U$ es un vertex cover de $G'$ de tamaño $|V' \setminus U| = |V| - k = k'$. Además, $f$ puede ser calculado por un algoritmo polinomial. Luego $f$ es una reducción polinomial.

# Reducción polinomial

**Proposición:** *Si* $R \leq_p Q$ *y* $Q \in P$*, entonces* $R \in P$.

# Reducción polinomial

**Proposición:** *Si $R \leq_p Q$ y $Q \in P$, entonces $R \in P$.*

**Demo:** Como $R \leq_p Q$, existe $f : I_R \to I_Q$ una reducción polinomial calculada por un algoritmo polinomial $A_f$. Además, si $Q \in$ P, entonces existe un algoritmo polinomial $A_Q$ que resuelve $Q$, ie $\forall x \in I_Q,\ Q(x) = s \iff A_Q(x) = s$. Luego, se construye un algoritmo $A$ de la siguiente forma:

**Algorithm** A(x)

**Input:** $x \in I_R$

$y \leftarrow A_f(x)$
**if** $A_Q(y) = s$ **then**
    **return** $s$
**else**
    **return** $n$
**end if**

# Reducción polinomial

**Proposición:** *Si $R \leq_p Q$ y $Q \in P$, entonces $R \in P$.*

**Demo:** Como $R \leq_p Q$, existe $f : I_R \to I_Q$ una reducción polinomial calculada por un algoritmo polinomial $A_f$. Además, si $Q \in \mathrm{P}$, entonces existe un algoritmo polinomial $A_Q$ que resuelve $Q$, ie $\forall x \in I_Q,\ Q(x) = s \iff A_Q(x) = s$. Luego, se construye un algoritmo $A$ de la siguiente forma:

**Algorithm** A(x)

```
Input: x ∈ I_R
y ← A_f(x)
if A_Q(y) = s then
    return s
else
    return n
end if
```

Luego, $A$ es polinomial, pues $A_f$ y $A_Q$ son polinomiales. Además, $\forall x \in I_R$,

$$A(x) = s \Leftrightarrow A_Q(f(x)) = s \Leftrightarrow Q(f(x)) = s \Leftrightarrow R(x) = s.$$

Por lo tanto, $R \in \mathrm{P}$.

# NP-completitud

**Definición:** Un problema de decisión $Q$ se dice NP-hard (o NP-difícil) si $\forall R \in \mathrm{NP}, R \leq_p Q$. $Q$ se dice NP-completo si es NP-hard y NP.

# NP-completitud

**Definición:** Un problema de decisión $Q$ se dice NP-hard (o NP-difícil) si $\forall R \in \mathrm{NP}, R \leq_p Q$. $Q$ se dice NP-completo si es NP-hard y NP.

**Teorema:** *Si $Q$ es NP-hard y $Q \in P$, entonces $P{=}NP$.*

# NP-completitud

**Definición:** Un problema de decisión $Q$ se dice NP-hard (o NP-difícil) si $\forall R \in \mathrm{NP}, R \leq_p Q$. $Q$ se dice NP-completo si es NP-hard y NP.

**Teorema:** *Si $Q$ es NP-hard y $Q \in P$, entonces P=NP.*

**Demo:** Probemos que NP $\subseteq$ P. Como $Q$ es NP-hard, $\forall R \in \mathrm{NP}, R \leq_p Q$. De aquí, si $Q \in \mathrm{P}$, por proposición anterior se tiene que $\forall R \in \mathrm{NP}$, $R \in P$. Así, NP $\subseteq$ P.

**Ejemplo:** Algunos ejemplos de problemas NP-Completo son: CLIQUE, HAMILTONIAN, VERTEX COVER, etc.

**Observación:** El famoso paper de Karp titulado: "Reducibility Among Combinatorial Problems"de 1972, contiene una lista de 21 problemas NP-completo y las reducciones entre ellos.

# NP-completitud

**Teorema:** *Si $R$ es NP-hard y $R \leq_p Q$, entonces $Q$ es NP-hard.*

# NP-completitud

**Teorema:** *Si $R$ es NP-hard y $R \leq_p Q$, entonces $Q$ es NP-hard.*

**Demo:** Como $R \leq_p Q$, existe $f : I_R \rightarrow I_Q$ una reducción polinomial calculada por un algoritmo polinomial $A_f$. Además, como $R$ es NP-hard, entonces $\forall T \in NP,\ T \leq_p R$. Sea $T \in$ NP, luego existe $g : I_T \rightarrow I_R$ una reducción polinomial calculada por un algoritmo polinomial $A_g$. Mostremos que $f \circ g : I_T \rightarrow I_Q$ es reducción polinomial. Para ello se define el siguiente algoritmo $A$:

# NP-completitud

**Teorema:** *Si $R$ es NP-hard y $R \leq_p Q$, entonces $Q$ es NP-hard.*

**Demo:** Como $R \leq_p Q$, existe $f : I_R \to I_Q$ una reducción polinomial calculada por un algoritmo polinomial $A_f$. Además, como $R$ es NP-hard, entonces $\forall T \in NP,\ T \leq_p R$. Sea $T \in$ NP, luego existe $g : I_T \to I_R$ una reducción polinomial calculada por un algoritmo polinomial $A_g$. Mostremos que $f \circ g : I_T \to I_Q$ es reducción polinomial. Para ello se define el siguiente algoritmo $A$:

**Algorithm** A(x)

**Input:** $x \in I_T$

1: $y \leftarrow A_g(x)$
2: $z \leftarrow A_f(y)$
3: **return** $z$

# NP-completitud

**Teorema:** *Si $R$ es NP-hard y $R \leq_p Q$, entonces $Q$ es NP-hard.*

**Demo:** Como $R \leq_p Q$, existe $f : I_R \to I_Q$ una reducción polinomial calculada por un algoritmo polinomial $A_f$. Además, como $R$ es NP-hard, entonces $\forall T \in NP,\ T \leq_p R$. Sea $T \in$ NP, luego existe $g : I_T \to I_R$ una reducción polinomial calculada por un algoritmo polinomial $A_g$. Mostremos que $f \circ g : I_T \to I_Q$ es reducción polinomial. Para ello se define el siguiente algoritmo $A$:

**Algorithm** A(x)

**Input:** $x \in I_T$

1: $y \leftarrow A_g(x)$
2: $z \leftarrow A_f(y)$
3: **return** $z$

Luego, $A$ es polinomial, pues $A_f$ y $A_g$ son polinomiales. Además, $\forall x \in I_T$,

$$A(x) = s \Leftrightarrow A_f(y) = s \Leftrightarrow Q(f(y)) = s \Leftrightarrow R(y) = s$$
$$\Leftrightarrow A_g(x) = s \Leftrightarrow R(g(x)) = s \Leftrightarrow T(x) = s.$$

Así, $T \leq_p Q$. Por lo tanto, $Q$ es NP-hard.

# Problema SAT

**Definición:** Una **fórmula Booleana** es una función Booleana (o función lógica) $\phi : \{V, F\}^n \to \{V, F\}$ cuya expresión está formada por variables Booleanas y conectores Booleanos $(\wedge,\ \vee, \neg)$.

# Problema SAT

**Definición:** Una **fórmula Booleana** es una función Booleana (o función lógica) $\phi : \{V, F\}^n \to \{V, F\}$ cuya expresión está formada por variables Booleanas y conectores Booleanos $(\wedge, \ \vee, \neg)$.

**Ejemplo:** $\phi(x_1, x_2, x_3) = \neg x_2 \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \wedge \neg x_3)$ es una fórmula Booleana tal que $\phi(V, F, F) = V$ and $\phi(F, F, V) = F$.

# Problema SAT

**Definición:** Una **fórmula Booleana** es una función Booleana (o función lógica) $\phi : \{V, F\}^n \to \{V, F\}$ cuya expresión está formada por variables Booleanas y conectores Booleanos $(\wedge, \ \vee, \neg)$.

**Ejemplo:** $\phi(x_1, x_2, x_3) = \neg x_2 \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \wedge \neg x_3)$ es una fórmula Booleana tal que $\phi(V, F, F) = V$ and $\phi(F, F, V) = F$.

**Definición:** Sea $\phi(x)$ una fórmula Booleana. Se define:

- **Literal** de $\phi$: cualquier variable o variable negada que aparezca en la expresión de $\phi$.

# Problema SAT

**Definición:** Una **fórmula Booleana** es una función Booleana (o función lógica) $\phi : \{V, F\}^n \rightarrow \{V, F\}$ cuya expresión está formada por variables Booleanas y conectores Booleanos ($\wedge,\ \vee, \neg$).

**Ejemplo:** $\phi(x_1, x_2, x_3) = \neg x_2 \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \wedge \neg x_3)$ es una fórmula Booleana tal que $\phi(V, F, F) = V$ and $\phi(F, F, V) = F$.

**Definición:** Sea $\phi(x)$ una fórmula Booleana. Se define:

- **Literal** de $\phi$: cualquier variable o variable negada que aparezca en la expresión de $\phi$.
- **Cláusula** de $\phi$: conjunto de literales conectados por un mismo conector: $\vee$ o $\wedge$. Si el conector es $\vee$ se dice cláusula disyuntiva, en caso contrario se dice cláusula conjuntiva.

# Problema SAT

**Definición:** Una **fórmula Booleana** es una función Booleana (o función lógica) $\phi : \{V, F\}^n \to \{V, F\}$ cuya expresión está formada por variables Booleanas y conectores Booleanos $(\wedge, \vee, \neg)$.

**Ejemplo:** $\phi(x_1, x_2, x_3) = \neg x_2 \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \wedge \neg x_3)$ es una fórmula Booleana tal que $\phi(V, F, F) = V$ and $\phi(F, F, V) = F$.

**Definición:** Sea $\phi(x)$ una fórmula Booleana. Se define:

- **Literal** de $\phi$: cualquier variable o variable negada que aparezca en la expresión de $\phi$.
- **Cláusula** de $\phi$: conjunto de literales conectados por un mismo conector: $\vee$ o $\wedge$. Si el conector es $\vee$ se dice cláusula disyuntiva, en caso contrario se dice cláusula conjuntiva.
- **Forma normal conjuntiva (cnf-fórmula)**: $\phi$ se dice una cnf-fórmula si su expresión es un conjunto de cláusulas disyuntivas unidas por el conector $\wedge$.

# Problema SAT

**Definición:** Una **fórmula Booleana** es una función Booleana (o función lógica) $\phi : \{V, F\}^n \to \{V, F\}$ cuya expresión está formada por variables Booleanas y conectores Booleanos ($\wedge$, $\vee$, $\neg$).

**Ejemplo:** $\phi(x_1, x_2, x_3) = \neg x_2 \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \wedge \neg x_3)$ es una fórmula Booleana tal que $\phi(V, F, F) = V$ and $\phi(F, F, V) = F$.

**Definición:** Sea $\phi(x)$ una fórmula Booleana. Se define:

- **Literal** de $\phi$: cualquier variable o variable negada que aparezca en la expresión de $\phi$.
- **Cláusula** de $\phi$: conjunto de literales conectados por un mismo conector: $\vee$ o $\wedge$. Si el conector es $\vee$ se dice cláusula disyuntiva, en caso contrario se dice cláusula conjuntiva.
- **Forma normal conjuntiva (cnf-fórmula)**: $\phi$ se dice una cnf-fórmula si su expresión es un conjunto de cláusulas disyuntivas unidas por el conector $\wedge$. Análogamente, una **forma normal disyuntiva** (dnf-fórmula) es cuando las cláusulas son conjuntivas y están unidas por el conector $\vee$.

# Problema SAT

**Definición:** Una **fórmula Booleana** es una función Booleana (o función lógica) $\phi : \{V, F\}^n \rightarrow \{V, F\}$ cuya expresión está formada por variables Booleanas y conectores Booleanos ($\wedge, \vee, \neg$).

**Ejemplo:** $\phi(x_1, x_2, x_3) = \neg x_2 \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \wedge \neg x_3)$ es una fórmula Booleana tal que $\phi(V, F, F) = V$ and $\phi(F, F, V) = F$.

**Definición:** Sea $\phi(x)$ una fórmula Booleana. Se define:

- **Literal** de $\phi$: cualquier variable o variable negada que aparezca en la expresión de $\phi$.
- **Cláusula** de $\phi$: conjunto de literales conectados por un mismo conector: $\vee$ o $\wedge$. Si el conector es $\vee$ se dice cláusula disyuntiva, en caso contrario se dice cláusula conjuntiva.
- **Forma normal conjuntiva (cnf-fórmula)**: $\phi$ se dice una cnf-fórmula si su expresión es un conjunto de cláusulas disyuntivas unidas por el conector $\wedge$. Análogamente, una **forma normal disyuntiva** (dnf-fórmula) es cuando las cláusulas son conjuntivas y están unidas por el conector $\vee$.

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$ es una cnf-fórmula.

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$ es una cnf-fórmula.
2. $\phi(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge \neg x_3)$

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$ es una cnf-fórmula.
2. $\phi(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge \neg x_3)$ es una dnf-fórmula.

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$ es una cnf-fórmula.
2. $\phi(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge \neg x_3)$ es una dnf-fórmula.
3. $\phi(x_1, x_2) = x_1 \wedge \neg x_2$

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$ es una cnf-fórmula.
2. $\phi(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge \neg x_3)$ es una dnf-fórmula.
3. $\phi(x_1, x_2) = x_1 \wedge \neg x_2$ es una cnf-fórmula y una dnf-fórmula.

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$ es una cnf-fórmula.
2. $\phi(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge \neg x_3)$ es una dnf-fórmula.
3. $\phi(x_1, x_2) = x_1 \wedge \neg x_2$ es una cnf-fórmula y una dnf-fórmula.
4. $\phi(x_1, x_2, x_3) = V \iff x_1 = x_2 = V$ o $x_2 = x_3 = F$

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$ es una cnf-fórmula.
2. $\phi(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge \neg x_3)$ es una dnf-fórmula.
3. $\phi(x_1, x_2) = x_1 \wedge \neg x_2$ es una cnf-fórmula y una dnf-fórmula.
4. $\phi(x_1, x_2, x_3) = V \iff x_1 = x_2 = V$ o $x_2 = x_3 = F$ no es una fórmula Booleana.

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$ es una cnf-fórmula.
2. $\phi(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge \neg x_3)$ es una dnf-fórmula.
3. $\phi(x_1, x_2) = x_1 \wedge \neg x_2$ es una cnf-fórmula y una dnf-fórmula.
4. $\phi(x_1, x_2, x_3) = V \iff x_1 = x_2 = V$ o $x_2 = x_3 = F$ no es una fórmula Booleana.

**Teorema:** *Toda fórmula Booleana puede ser escrita como una cnf-fórmula o una dnf-fórmula.*

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$ es una cnf-fórmula.
2. $\phi(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge \neg x_3)$ es una dnf-fórmula.
3. $\phi(x_1, x_2) = x_1 \wedge \neg x_2$ es una cnf-fórmula y una dnf-fórmula.
4. $\phi(x_1, x_2, x_3) = V \iff x_1 = x_2 = V$ o $x_2 = x_3 = F$ no es una fórmula Booleana.

**Teorema:** *Toda fórmula Booleana puede ser escrita como una cnf-fórmula o una dnf-fórmula.*

**Ejemplo:** Sea $\phi(x_1, x_2, x_3)$ la fórmula Booleana dada por:

| $x_1$ | $x_2$ | $x_3$ | $\phi(x)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$ es una cnf-fórmula.
2. $\phi(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge \neg x_3)$ es una dnf-fórmula.
3. $\phi(x_1, x_2) = x_1 \wedge \neg x_2$ es una cnf-fórmula y una dnf-fórmula.
4. $\phi(x_1, x_2, x_3) = V \iff x_1 = x_2 = V$ o $x_2 = x_3 = F$ no es una fórmula Booleana.

**Teorema:** *Toda fórmula Booleana puede ser escrita como una cnf-fórmula o una dnf-fórmula.*

**Ejemplo:** Sea $\phi(x_1, x_2, x_3)$ la fórmula Booleana dada por:

| $x_1$ | $x_2$ | $x_3$ | $\phi(x)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

$$\begin{aligned}\phi(x_1, x_2, x_3) =& (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \\ & \vee (x_1 \wedge \neg x_2 \wedge x_3)\end{aligned}$$

es una dnf-fómula.

# Problema SAT

**Ejemplos:**

1. $\phi(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_1 \vee \neg x_4) \wedge \neg x_1 \wedge x_3$ es una cnf-fórmula.
2. $\phi(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge \neg x_3)$ es una dnf-fórmula.
3. $\phi(x_1, x_2) = x_1 \wedge \neg x_2$ es una cnf-fórmula y una dnf-fórmula.
4. $\phi(x_1, x_2, x_3) = V \iff x_1 = x_2 = V$ o $x_2 = x_3 = F$ no es una fórmula Booleana.

**Teorema:** *Toda fórmula Booleana puede ser escrita como una cnf-fórmula o una dnf-fórmula.*

**Ejemplo:** Sea $\phi(x_1, x_2, x_3)$ la fórmula Booleana dada por:

| $x_1$ | $x_2$ | $x_3$ | $\phi(x)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

$$\begin{aligned}\phi(x_1, x_2, x_3) =& (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \\ & \vee (x_1 \wedge \neg x_2 \wedge x_3)\end{aligned}$$

es una dnf-fómula. Usando la propiedad distributiva, la de doble negación y las Leyes de Morgan se obtiene una cnf-fórmula.

# Problema SAT

**Definición:** Una fórmula Booleana $\phi(x_1, \ldots, x_n)$ se dice **satisfacible** si $\exists x \in \{V, F\}^n, \ \phi(x) = V$.

# Problema SAT

**Definición:** Una fórmula Booleana $\phi(x_1, \ldots, x_n)$ se dice **satisfacible** si $\exists x \in \{V, F\}^n,\ \phi(x) = V$.

**Ejemplos:**

- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$

# Problema SAT

**Definición:** Una fórmula Booleana $\phi(x_1, \ldots, x_n)$ se dice **satisfacible** si $\exists x \in \{V, F\}^n,\ \phi(x) = V$.

**Ejemplos:**

- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ es satisfacible.

# Problema SAT

**Definición:** Una fórmula Booleana $\phi(x_1, \ldots, x_n)$ se dice **satisfacible** si $\exists x \in \{V, F\}^n,\ \phi(x) = V$.

**Ejemplos:**

- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ es satisfacible.
- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge x_1 \wedge \neg x_1$

# Problema SAT

**Definición:** Una fórmula Booleana $\phi(x_1, \ldots, x_n)$ se dice **satisfacible** si $\exists x \in \{V, F\}^n,\ \phi(x) = V$.

**Ejemplos:**

- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ es satisfacible.
- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge x_1 \wedge \neg x_1$ no es satisfacible.

# Problema SAT

**Definición:** Una fórmula Booleana $\phi(x_1, \ldots, x_n)$ se dice **satisfacible** si $\exists x \in \{V, F\}^n,\ \phi(x) = V$.

**Ejemplos:**

- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ es satisfacible.
- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge x_1 \wedge \neg x_1$ no es satisfacible.

De esta forma, se define el problema SAT como:

# Problema SAT

**Definición:** Una fórmula Booleana $\phi(x_1, \ldots, x_n)$ se dice **satisfacible** si $\exists x \in \{V, F\}^n,\ \phi(x) = V$.

**Ejemplos:**

- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ es satisfacible.
- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge x_1 \wedge \neg x_1$ no es satisfacible.

De esta forma, se define el problema SAT como:

**SAT**: Dada una cnf-fórmula $\phi$. ¿Es $\phi$ satisfacible?

# Problema SAT

**Definición:** Una fórmula Booleana $\phi(x_1, \ldots, x_n)$ se dice **satisfacible** si $\exists x \in \{V, F\}^n,\ \phi(x) = V$.

**Ejemplos:**

- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ es satisfacible.
- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge x_1 \wedge \neg x_1$ no es satisfacible.

De esta forma, se define el problema SAT como:

**SAT**: Dada una cnf-fórmula $\phi$. ¿Es $\phi$ satisfacible?

**Teorema Cook-Levin (1971):** *SAT es NP-completo.*

# Problema SAT

**Definición:** Una fórmula Booleana $\phi(x_1, \ldots, x_n)$ se dice **satisfacible** si $\exists x \in \{V, F\}^n,\ \phi(x) = V$.

**Ejemplos:**

- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ es satisfacible.
- $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge x_1 \wedge \neg x_1$ no es satisfacible.

De esta forma, se define el problema SAT como:

**SAT**: Dada una cnf-fórmula $\phi$. ¿Es $\phi$ satisfacible?

**Teorema Cook-Levin (1971):** *SAT es NP-completo.*
**Demo:** (ver libro de Sipser).

# Problema 3-SAT

**Definición:** Una fórmula Booleana $\phi$ se dice una **3-cnf-fórmula** si está escrita como una cnf-fórmula donde cada cláusula tiene exactamente tres literales. De esta forma, se define el problema 3-SAT como:

# Problema 3-SAT

**Definición:** Una fórmula Booleana $\phi$ se dice una **3-cnf-fórmula** si está escrita como una cnf-fórmula donde cada cláusula tiene exactamente tres literales. De esta forma, se define el problema 3-SAT como:

**3-SAT**: Dada una 3-cnf-fórmula $\phi$. ¿Es $\phi$ satisfacible?

# Problema 3-SAT

**Definición:** Una fórmula Booleana $\phi$ se dice una **3-cnf-fórmula** si está escrita como una cnf-fórmula donde cada cláusula tiene exactamente tres literales. De esta forma, se define el problema 3-SAT como:

**3-SAT**: Dada una 3-cnf-fórmula $\phi$. ¿Es $\phi$ satisfacible?

**Teorema:** *3-SAT es NP-completo*

# Problema 3-SAT

**Definición:** Una fórmula Booleana $\phi$ se dice una **3-cnf-fórmula** si está escrita como una cnf-fórmula donde cada cláusula tiene exactamente tres literales. De esta forma, se define el problema 3-SAT como:

**3-SAT**: Dada una 3-cnf-fórmula $\phi$. ¿Es $\phi$ satisfacible?

**Teorema:** *3-SAT es NP-completo*
**Demo:** Sea $\phi$ una 3-cnf-fórmula con $k$ cláusulas disyuntivas y $n$ variables. Como SAT es NP y 3-SAT es un caso particular de SAT, entonces 3-SAT es NP. En este caso, el certificado es $y \in \{V, F\}^n$ con $size(y) = n$ y $size(\phi) = O(n + 3k) \Longrightarrow size(y) = O(size(\phi))$. Para verificar que $\phi(y) = V$ se requiere chequear que en cada cláusula alguno de los literales es V, lo que se puede hacer en tiempo lineal en el número de las cláusulas.

# Problema 3-SAT

### Demo: (continuación)

Defina la función $f : I_{\text{SAT}} \to I_{\text{3-SAT}}$ con $f(\phi) = \phi'$, donde $\phi'$ es construído a partir de $\phi$ como sigue:

# Problema 3-SAT

**Demo: (continuación)**

Defina la función $f : I_{\text{SAT}} \to I_{\text{3-SAT}}$ con $f(\phi) = \phi'$, donde $\phi'$ es construído a partir de $\phi$ como sigue:

1. Si $C_i$ es una cáusula de $\phi$ con menos de 3 literales, entonces se repite un literal hasta tener 3 literales en total. Ejemplo: $C_i = (x_1 \vee x_2) \to C_i' = (x_1 \vee x_2 \vee x_2)$.

# Problema 3-SAT

**Demo: (continuación)**

Defina la función $f : I_{\text{SAT}} \to I_{\text{3-SAT}}$ con $f(\phi) = \phi'$, donde $\phi'$ es construído a partir de $\phi$ como sigue:

1. Si $C_i$ es una cáusula de $\phi$ con menos de 3 literales, entonces se repite un literal hasta tener 3 literales en total. Ejemplo: $C_i = (x_1 \vee x_2) \to C_i' = (x_1 \vee x_2 \vee x_2)$.
2. Si $C_i$ tiene más de 3 literales, entonces se crea una cnf-fómula $C_i'$ formado por dos cláusulas más pequeñas, agregando literales nuevos $y, \neg y$ como muestra el siguiente ejemplo:

# Problema 3-SAT

**Demo: (continuación)**

Defina la función $f : I_{\text{SAT}} \to I_{\text{3-SAT}}$ con $f(\phi) = \phi'$, donde $\phi'$ es construído a partir de $\phi$ como sigue:

1. Si $C_i$ es una cáusula de $\phi$ con menos de 3 literales, entonces se repite un literal hasta tener 3 literales en total. Ejemplo: $C_i = (x_1 \vee x_2) \to C_i' = (x_1 \vee x_2 \vee x_2)$.
2. Si $C_i$ tiene más de 3 literales, entonces se crea una cnf-fómula $C_i'$ formado por dos cláusulas más pequeñas, agregando literales nuevos $y, \neg y$ como muestra el siguiente ejemplo:

$$C_i = (x_1 \vee x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5) \to C_i' = (x_1 \vee x_2 \vee y) \wedge (\neg y \vee \neg x_3 \vee x_4 \vee \neg x_5).$$

# Problema 3-SAT

### Demo: (continuación)

Defina la función $f : I_{\text{SAT}} \to I_{\text{3-SAT}}$ con $f(\phi) = \phi'$, donde $\phi'$ es construído a partir de $\phi$ como sigue:

1. Si $C_i$ es una cáusula de $\phi$ con menos de 3 literales, entonces se repite un literal hasta tener 3 literales en total. Ejemplo: $C_i = (x_1 \vee x_2) \to C_i' = (x_1 \vee x_2 \vee x_2)$.
2. Si $C_i$ tiene más de 3 literales, entonces se crea una cnf-fómula $C_i'$ formado por dos cláusulas más pequeñas, agregando literales nuevos $y, \neg y$ como muestra el siguiente ejemplo:

   $$C_i = (x_1 \vee x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5) \to C_i' = (x_1 \vee x_2 \vee y) \wedge (\neg y \vee \neg x_3 \vee x_4 \vee \neg x_5).$$

   Este procedimiento se repite hasta que todas las cláusulas son de tamaño 3. Ejemplo:

   $$(x_1 \vee x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5) \to (x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee \neg x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee \neg x_5).$$

# Problema 3-SAT

**Demo: (continuación)**

Defina la función $f : I_{\text{SAT}} \to I_{\text{3-SAT}}$ con $f(\phi) = \phi'$, donde $\phi'$ es construído a partir de $\phi$ como sigue:

1. Si $C_i$ es una cáusula de $\phi$ con menos de 3 literales, entonces se repite un literal hasta tener 3 literales en total. Ejemplo: $C_i = (x_1 \vee x_2) \to C_i' = (x_1 \vee x_2 \vee x_2)$.
2. Si $C_i$ tiene más de 3 literales, entonces se crea una cnf-fómula $C_i'$ formado por dos cláusulas más pequeñas, agregando literales nuevos $y, \neg y$ como muestra el siguiente ejemplo:

   $$C_i = (x_1 \vee x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5) \to C_i' = (x_1 \vee x_2 \vee y) \wedge (\neg y \vee \neg x_3 \vee x_4 \vee \neg x_5).$$

   Este procedimiento se repite hasta que todas las cláusulas son de tamaño 3. Ejemplo:

   $$(x_1 \vee x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5) \to (x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee \neg x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee \neg x_5).$$

Es fácil ver que $\phi'$ es una 3-cnf fórmula tal que $\phi$ es satisfacible si y sólo si $\phi'$ es satisfacible y que $f$ puede ser calculada en tiempo polinomial. Por lo tanto, SAT $\leq_p$ 3-SAT $\Longrightarrow$ 3-SAT es NP-hard y así 3-SAT es NP-completo.

# NP-completitud

**Definición:** Análogamente, un problema de decisión $Q$ se dice **co-NP-hard** (o **co-NP-difícil**) si $\forall R \in$ co-NP, $R \leq_p Q$. $Q$ se dice **co-NP-completo** si es co-NP-hard y co-NP.

# NP-completitud

**Definición:** Análogamente, un problema de decisión $Q$ se dice **co-NP-hard** (o **co-NP-difícil**) si $\forall R \in$ co-NP, $R \leq_p Q$. $Q$ se dice **co-NP-completo** si es co-NP-hard y co-NP.

En resumen, tenemos las siguientes clases de complejidad temporal.

# NP-completitud

**Definición:** Análogamente, un problema de decisión $Q$ se dice **co-NP-hard** (o **co-NP-difícil**) si $\forall R \in$ co-NP, $R \leq_p Q$. $Q$ se dice **co-NP-completo** si es co-NP-hard y co-NP.

En resumen, tenemos las siguientes clases de complejidad temporal.

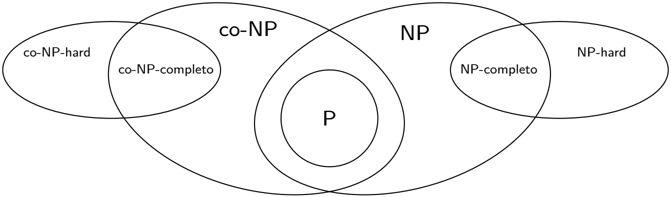


Figura: Relación entre clases de complejidad temporal. La igualdad de algunos de estos conjuntos es un problema abierto.