



DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Algoritmos y Estructuras de Datos III

Primer cuatrimestre 2021 (*dictado a distancia*)

Teoría de complejidad computacional

Teoría de complejidad computacional

- ▶ Un *algoritmo eficiente* es un algoritmo de complejidad polinomial.
- ▶ El objetivo es clasificar los *problemas* según su dificultad.
- ▶ Un problema está *bien resuelto* si se conocen algoritmos eficientes para resolverlo.

Distintas versiones de un problema de optimización Π

Dada una instancia I del problema Π con función objetivo f :

- ▶ Versión de *optimización*: Encontrar una *solución óptima* del problema Π para I (de valor mínimo o máximo).
- ▶ Versión de *evaluación*: Determinar el *valor* de una solución óptima de Π para I .
- ▶ Versión de *localización*: Dado un número k , determinar una *solución factible* de Π para I tal que $f(S) \leq k$ si el problema es de minimización (o $f(S) \geq k$ si el problema es de maximización).
- ▶ Versión de *localización*: Dado un número k , determinar una *solución factible* de Π para I tal que $f(S) \leq k$ si el problema es de minimización (o $f(S) \geq k$ si el problema es de maximización).
- ▶ Versión de *decisión*: Dado un número k , ¿existe una solución factible de Π para I tal que $f(S) \leq k$ si el problema es de minimización (o $f(S) \geq k$ si el problema es de maximización)?

Ejemplo: Problema del viajante de comercio

Dado un grafo G con longitudes asignadas a sus aristas:

- ▶ Versión de *optimización*: Determinar un circuito hamiltoniano de G de longitud mínima.
- ▶ Versión de *evaluación*: Determinar el valor de una solución óptima, o sea la longitud de un circuito hamiltoniano de G de longitud mínima.
- ▶ Versión de *localización*: Dado un número k , determinar un circuito hamiltoniano de G de longitud menor o igual a k .
- ▶ Versión de *decisión*: Dado un número k , ¿existe un circuito hamiltoniano de G de longitud menor o igual a k ?

Distintas versiones de un problema de optimización Π

Hay cierta relación entre la dificultad de resolver las distintas versiones de un mismo problema:

- ▶ La versión optimización es computacionalmente la *más difícil*, ya que usualmente es fácil (eficiente) evaluar la función objetivo para una solución factible.
- ▶ La versión de decisión suele ser la *más fácil*. Dada la respuesta de cualquiera de las otras versiones es fácil (eficiente) calcular la respuesta de esta versión.

Distintas versiones de un problema de optimización Π

- ▶ Para los problemas de optimización que cumplen que el valor de una solución óptima es un entero K cuyo logaritmo está acotado por un polinomio en la medida de la entrada, la versión de evaluación puede ser resuelta ejecutando una cantidad de veces polinomial la versión de decisión, realizando búsqueda binaria sobre el parámetro K . Muchos problemas cumplen esta propiedad.
- ▶ Si resolvemos el problema de decisión, podemos encontrar la solución de la versión de localización resolviendo una cantidad polinomial de problemas de decisión para instancias reducidas de la instancia original. La misma relación ocurre entre las versiones de evaluación y optimización.

Ejemplo: Problema del viajante de comercio

Supongamos que tenemos un grafo G y un entero k tales que la respuesta de la versión de decisión es **SI** y queremos resolver la versión de localización.

Para esto seleccionamos una arista del grafo y resolvemos la versión de decisión colocando peso infinito a esa arista:

- ▶ Si la respuesta sigue siendo **SI** significa que G tiene un circuito hamiltoniano de peso menor o igual a k que no utiliza esa arista. Dejamos el peso de esa arista en infinito.
- ▶ Si la respuesta pasa a ser **NO** significa que esa arista pertenece a todo circuito hamiltoniano de peso menor o igual a k . Volvemos a poner el peso original de esa arista.

Realizamos este proceso sobre todas las aristas. Al finalizar, las aristas que quedan con peso distinto de infinito formarán un ciclo hamiltoniano de peso menor o igual a k .

Teoría de complejidad computacional

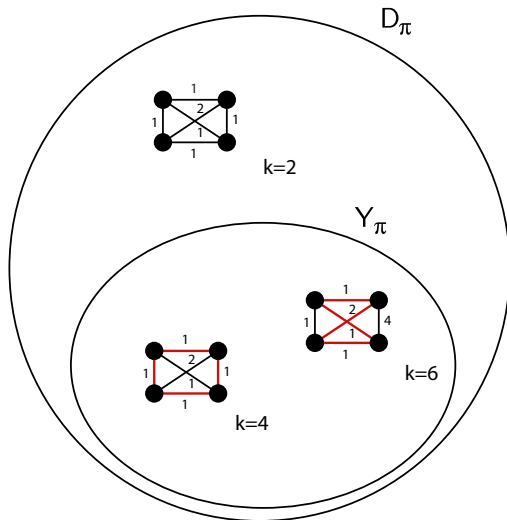
- ▶ Para muchos problemas de optimización combinatoria las cuatro versiones son equivalentes: si existe un algoritmo eficiente para una de ellas, entonces existe para todas.
- ▶ La clasificación y el estudio se realiza sobre problemas de decisión.
- ▶ Un *problemas de decisión* tiene respuesta *SI* o *NO*.
- ▶ Ésto permite uniformar el estudio, ya que hay problemas, como saber si un grafo es hamiltoniano o satisfabilidad, que no tienen versión de optimización.
- ▶ Un resultado negativo para la versión de decisión se traslada a un resultado negativo a la versión de optimización.

Instancias de un problema

- ▶ Una *instancia* de un problema es una especificación de sus parámetros.
- ▶ Un problema de decisión Π tiene asociado
 - ▶ un conjunto D_Π de instancias,
 - ▶ y un subconjunto $Y_\Pi \subseteq D_\Pi$ de instancias cuya respuesta es *SI*.

Ejemplo: TSP

Dado un grafo completo con peso en las aristas y un número k , ¿existe un circuito Hamiltoniano de longitud a lo sumo k ?



Problemas intratables

Un problema es *intratable* si no puede ser resuelto por algún algoritmo eficiente.

Un problema puede ser intratable por distintos motivos:

- ▶ El problema requiere una respuesta de longitud exponencial (ejemplo: pedir todos los circuitos hamiltonianos de longitud a lo sumo s).
- ▶ El problema es *indecidable* (ejemplo: problema de la parada).
- ▶ El problema es *decidable* pero no se conocen algoritmos polinomiales que lo resuelvan (no se sabe si es intratable o no).

Problema de satisfacibilidad (SAT)

- ▶ Una variable booleana o lógica es una variable que puede tomar valor verdadero (V) o falso (F).
- ▶ Un literal es una variable booleana o su negación y una expresión o fórmula booleana está formada por variables, conjunciones, disyunciones y negaciones.
- ▶ Una cláusula es una disyunción de literales y una fórmula está en forma normal conjuntiva (FNC) si es una conjunción de cláusulas.
- ▶ Toda fórmula lógica puede ser expresada en FNC.
- ▶ Una fórmula se dice satisfacible si existe una asignación de valores de verdad a sus variables tal que la haga verdadera.

Problema de satisfacibilidad (SAT) - Ejemplo

Dados $X = \{x_1, x_2, x_3\}$ un conjunto de variables booleanas y las fórmulas en FNC:

- ▶ $\varphi_1 = (x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_3)$ es satisfacible:
 - ▶ asignando los valores de verdad $x_1 = x_2 = V$ y $x_3 = F$, φ_1 es verdadera.
- ▶ $\varphi_2 = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2) \wedge (\neg x_3)$ no es satisfacible:
 - ▶ no existe asignación de valores de verdad a las variables de X que hagan a φ_2 verdadera.

Problema de satisfabilidad (SAT)

Veremos que este problema tiene un rol central en el estudio de la teoría de la complejidad computacional.

Problema de satisfabilidad (SAT):

Dado un conjunto de cláusulas C_1, \dots, C_m formadas por literales basados en las variables booleanas $X = \{x_1, \dots, x_n\}$,

¿la expresión $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_m$ es satisfacible?

Modelos de cómputo

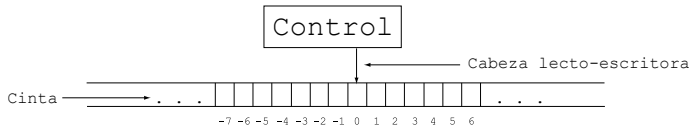
Modelos formales para expresar cualquier algoritmo:

- ▶ Máquina de Turing (1937, Alan Turing)
- ▶ Máquinas de Acceso Random - RAM (1974, Aho, Hopcroft y Ullman).

Los dos modelos son polinomialmente equivalentes. Es decir, se puede simular uno a otro con un costo polinomial.

Máquina de Turing Determinística

- ▶ Una cabeza lecto-escritora.
- ▶ Una cinta infinita con el siguiente esquema:



- ▶ La cabeza lectora se puede mover en ambas direcciones, $M = \{+1, -1\}$, derecha (+1) o izquierda (-1).
- ▶ Hay una celda distinguida, la celda 0, que define la celda inicial.

Máquina de Turing Determinística

- ▶ Un alfabeto finito Σ y un símbolo especial $*$ que indica *blanco*, definiendo $\Gamma = \Sigma \cup \{*\}$.
- ▶ Sobre la cinta está escrita la entrada, que es un string de símbolos de Σ y el resto de las celdas tiene $*$ (blancos).
- ▶ Un conjunto finito de estados, Q .
- ▶ Hay un estado distinguido, $q_0 \in Q$, que define el estado inicial.
- ▶ Y un conjunto de estados finales, $Q_f \subseteq Q$ (q_{si} y q_{no} para problemas de decisión)

Máquina de Turing Determinística

- ▶ Una *instrucción* en una MT es una quintupla $S \subseteq Q \times \Gamma \times Q \times \Gamma \times M$. La quintupla $(q_i, s_h, q_j, s_k, +1)$ se interpreta como:
Si la máquina está en el estado q_i y la cabeza lee s_h , entonces escribe s_k , se mueve a la derecha y pasa al estado q_j .
- ▶ Se define un *programa* en una MT como un conjunto finito de instrucciones.
- ▶ Una MT es *determinística*, MTD, si para todo par (q_i, s_h) existe en el programa a lo sumo una quintupla que comienza con ese par.

Máquina de Turing Determinística

▶ Arranque

- ▶ máquina posicionada en el estado distinguido q_0 , estado inicial
- ▶ cabeza lectora-escritora ubicada en la celda inicial (0) de la cinta.

▶ Transiciones

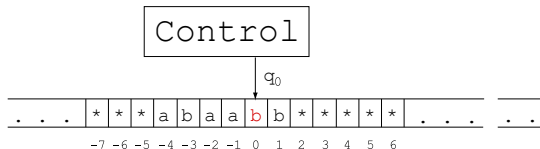
- ▶ si la máquina está en el estado q_i y lee en la cinta s_h , busca en la tabla de instrucciones un quintupla que comience con el par (q_i, s_j) .
- ▶ si existe esta quintupla, (q_i, s_h, q_j, s_k, m) , la máquina pasa al estado q_j , escribe en la cinta el símbolo s_k y se mueve a derecha o izquierda según el valor de m .

▶ Terminación

- ▶ cuando no se puede inferir nuevas acciones para seguir: no existe la quintupla que comienza con (q_i, s_h)
- ▶ cuando se alcanza un estado final: si el estado final es de SI, entonces la respuesta es **SI**, caso contrario la respuesta es **NO**.

Máquina de Turing Determinística - Ejemplo

- ▶ $\Sigma = \{a, b\};$
 $\Gamma = \Sigma \cup \{*\};$
- ▶ $Q = \{q_0, q_1, q_{si}, q_{no}\};$
 $Q_f = \{q_{si}, q_{no}\}$
- ▶ $S = (q_0, a, q_0, a, +1),$
 $(q_0, b, q_1, a, -1),$
 $(q_0, *, q_{si}, *, -1),$
 $(q_1, a, q_0, a, -1),$
 $(q_1, b, q_{no}, a, -1),$
 $(q_1, *, q_0, b, +1)$



Máquina de Turing Determinística

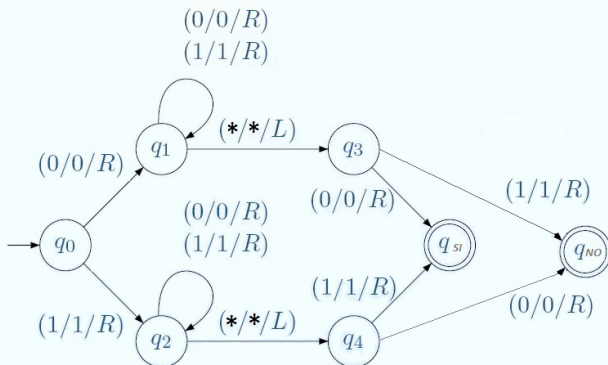
- ▶ Una máquina M resuelve el problema Π si para toda instancia alcanza un estado final y responde de forma correcta (o sea, termina en un estado final correcto).
- ▶ La complejidad de una MTD está dada por la cantidad de movimientos de la cabeza, desde el estado inicial hasta alcanzar un estado final, en función del tamaño de la entrada.

$$T_M(n) = \max\{m \text{ tq } x \in D_\Pi, |x| = n \text{ y } M \text{ con entrada } x \text{ hace } m \text{ movimientos}\}$$

- ▶ Existen otros modelos de computadoras determinísticas, pero puede probarse que son equivalentes en términos de la polinomialidad de los problemas a la MTD.

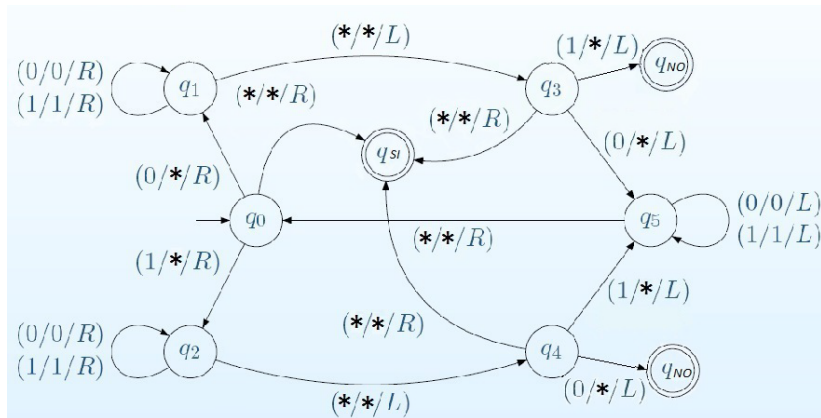
Máquina de Turing Determinística - Ejemplo

Dada una palabra sobre $\{0, 1\}$, ¿comienza y termina con el mismo símbolo?



Máquina de Turing Determinística - Ejemplo

Dada una palabra sobre $\{0, 1\}$, ¿es palíndromo?



La clase P

Un problema de decisión Π pertenece a la clase **P** (polinomial) si

- ▶ existe una MTD de complejidad polinomial que lo resuelve:

$$\mathbf{P} = \{\Pi \text{ tq } \exists M \text{ MTD tq } M \text{ resuelve } \Pi \text{ y } T_M(n) \in O(p(n)) \\ \text{para algún polinomio } p\}$$

Ésto es equivalente a:

- ▶ existe un algoritmo polinomial que lo resuelve.

La clase P - Ejemplos

- ▶ Grafo conexo.
- ▶ Grafo bipartito.
- ▶ Árbol generador mínimo.
- ▶ Camino mínimo entre un par de nodos.
- ▶ Matching máximo.
- ▶ Grafo euleriano.

Máquinas de Turing No Determinísticas (MTND)

- ▶ Generalización de una MTD.
- ▶ No se pide unicidad de la quintupla que comienza con cualquier par (q_i, s_j) .
- ▶ Las instrucciones dejan de ser quintuplas para pasar a ser un mapeo multivaluado.
- ▶ Un programa es una tabla que mapea un par (q_i, t_i) a un *conjunto* de ternas $(q_f, t_f, \{+1, -1\})$.

Máquinas de Turing No Determinísticas (MTND)

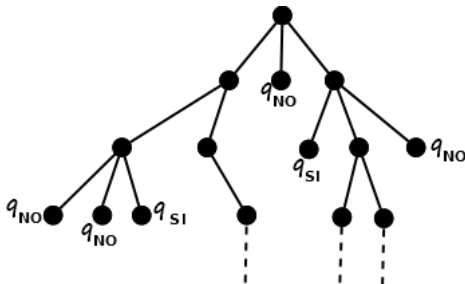
- ▶ Permite la ejecución en paralelo de las distintas alternativas.
- ▶ Cuando llega a un punto en su ejecución donde son posibles múltiples alternativas, examina todas las alternativas en paralelo, en lugar de hacerlo secuencialmente como una MTD.
- ▶ Podemos pensarlo como que se abre en k copias cuando se encuentra con k alternativas.
- ▶ Cada copia continua su ejecución independientemente.
- ▶ Si una copia acepta la entrada (para en un estado final de *SI*), todas las copias paran.
- ▶ Una interpretación equivalente es que cuando se presentan múltiples alternativas, la MTND selecciona la *alternativa correcta*.

Máquinas de Turing No Determinísticas (MTND)

- ▶ Una MTND resuelve el problema de decisión Π si
 - ▶ existe una secuencia de alternativas que lleva a un estado de aceptación si, y sólo si, la respuesta es *SI*, o bien
 - ▶ alguna de las copias se detiene en un estado de aceptación si y sólo si la respuesta es *SI*.
- ▶ Es equivalente a: Para toda instancia de Y_{Π} existe una rama que llega a un estado final q_{si} y para toda instancia en $D_{\Pi} \setminus Y_{\Pi}$ ninguna rama llega a un estado final q_{si} .

Máquinas de Turing No Determinísticas (MTND)

Podemos interpretar la ejecución de una MTND como un árbol de alternativas. Para una instancia de Y_{Π} será:



Máquinas de Turing No Determinísticas (MTND)

- ▶ La complejidad temporal de una MTND M se define como el máximo número de pasos que toma como mínimo reconocer una instancia de Y_{Π} en función de su tamaño:

$$T_M(n) = \max\{m \text{ tq } x \in Y_{\Pi}, |x| = n \text{ y alguna de las ramas de } M \text{ termina en estado } q_{si} \text{ en } m \text{ movimientos cuando la entrada es } x\}$$

- ▶ Una MTND M es *polinomial* para Π cuando $T_M(n)$ es un función polinomial.

La clase NP

Un problema de decisión Π pertenece a la clase **NP** (polinomial no-determinístico) si:

- ▶ las instancias de Π con respuesta *SI* son reconocidas por una MTND polinomial.

Equivalentemente:

- ▶ Dada una instancia de Π con respuesta *SI* se puede dar un certificado de longitud polinomial que garantiza que la respuesta es *SI*, y esta garantía puede ser verificada en tiempo polinomial.

Ejemplos de problemas en NP

- ▶ Todos los problemas de la clase P.
- ▶ Clique máxima.
- ▶ Problema del viajante de comercio.
- ▶ Conjunto independiente de cardinal máximo.
- ▶ Problema de satisfacibilidad (SAT).

La clase NP - Conjunto independiente máximo

Dado un grafo $G = (V, X)$, un *conjunto independiente* de vértices de G , es un conjunto de vértices $I \subseteq V$ tal que para toda arista $e \in X$, e es incidente a lo sumo a un vértice $v \in I$.

Dados un grafo $G = (V, X)$ y $k \in \mathbb{N}$, ¿ G tiene un conjunto independiente de tamaño mayor o igual a k ?

- ▶ Para una instancia con respuesta *SI*, podemos exponer $S \subseteq V$ conjunto independiente de G tal que $|S| \geq k$.
- ▶ Es posible chequear polinomialmente que S cumple estas dos propiedades: ser conjunto independiente de G y tener cardinal mayor o igual a k .
- ▶ Esto demuestra que conjunto independiente pertenece a la clase NP.

La clase NP - SAT

- ▶ Una instancia de *SI* de SAT es un conjunto de cláusulas C_1, \dots, C_m satisfacible.
- ▶ Ésto es, existe una asignación de valores de verdad a las variables booleanas intervinientes en las cláusulas que hacen verdadero el valor de verdad de la expresión $C_1 \wedge C_2 \wedge \dots \wedge C_m$.
- ▶ El certificado que podemos mostrar es una asignación de valores de verdad a las variables que haga verdadera a la expresión $C_1 \wedge C_2 \wedge \dots \wedge C_m$.
- ▶ Como es posible verificar en tiempo polinomial que esta expresión es verdad con esos valores de las variables, demuestra que $\text{SAT} \in \text{NP}$.

La clase NP - Clique

CLIQUE: Dado un grafo $G = (V, X)$ y $k \in \mathbb{N}$, ¿ G tiene una clique de tamaño mayor o igual a k ?

Dada una instancia de **SI**, ésto es un grafo $G = (V, X)$ que tiene una clique de tamaño mayor o igual a k , podemos verificar polinomialmente que un conjunto de vértices C cumple que:

- ▶ $C \subseteq V$
- ▶ C induce una clique en G
- ▶ $|C| \geq k$

La existencia de este algoritmo polinomial demuestra que CLIQUE \in NP.

La clase NP - Circuito hamiltoniano

- ▶ La evidencia que soporta una respuesta positiva es un ciclo hamiltoniano de G .
- ▶ Dado un lista de vértices, se puede chequear polinomialmente si define un ciclo hamiltoniano.

La clase NP - TSP

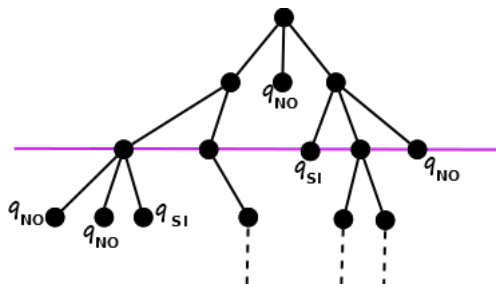
Dada una instancia de TSP, G y $k \in \mathbb{Z}_{\geq 0}$:

- ▶ La evidencia que soporta una respuesta positiva es un ciclo hamiltoniano de G con peso menor o igual a k .
- ▶ Dada una lista de vértices, se puede chequear polinomialmente si define un ciclo hamiltoniano y que la suma de los pesos de las aristas respectivas es menor o igual a k .

La clase NP

Lema:

Si Π es un problema de decisión que pertenece a la clase NP, entonces Π puede ser resuelto por un algoritmo determinístico en tiempo exponencial respecto del tamaño de la entrada.



Las clases P y NP

- ▶ $P \subseteq NP$.
- ▶ *Problema abierto*: ¿Es $P = NP$?
 - ▶ Todavía no se demostró que exista un problema en $NP \setminus P$.
 - ▶ Mientras tanto, se estudian clases de complejidad *relativa*, es decir, que establecen orden de dificultad entre problemas.

Transformaciones polinomiales

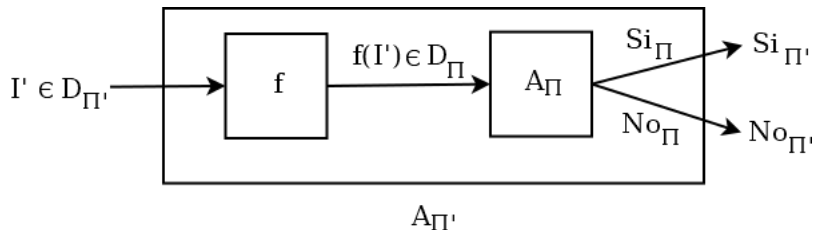
- Imaginemos que tenemos un algoritmo A_{Π} para resolver un problema Π y ahora queremos resolver otro problema Π' .
¿Habr  forma de que podamos aprovechar A_{Π} ?
- Para  sto, deber amos ser capaces de, dada una instancia I' del problema Π' , poder transformarla en una instancia I del problema Π , tal que la respuesta para I sea **SI** si, y s lo si, la respuesta para I' es **SI**.
- Lo que necesitamos es una transformaci n $f : D_{\Pi'} \rightarrow D_{\Pi}$, tal que

$$I' \in Y_{\Pi'} \iff f(I') \in Y_{\Pi}.$$

Transformaciones polinomiales

Entonces, dada una instancia I' de Π' , un algoritmo posible para resolverla podría ser

$$A_{\Pi'}(I') = A_{\Pi}(f(I')).$$



Si la función f y el algoritmo A_{Π} son polinomiales, el algoritmo $A_{\Pi'}$ resultará polinomial.

Transformaciones polinomiales

- Una *transformación o reducción polinomial* de un problema de decisión Π' a uno Π es una función polinomial que transforma una instancia I' de Π' en una instancia I de Π tal que I' tiene respuesta *SI* para Π' si, y sólo si, I tiene respuesta *SI* para Π :

$$I' \in Y_{\Pi'} \iff f(I') \in Y_{\Pi}$$

- El problema de decisión Π' se *reduce polinomialmente* a otro problema de decisión Π , $\Pi' \leq_p \Pi$, si existe una transformación polinomial de Π' a Π .

Proposición: Las reducciones polinomiales son transitivas:

si $\Pi_1 \leq_p \Pi_2$ y $\Pi_2 \leq_p \Pi_3$ entonces $\Pi_1 \leq_p \Pi_3$.

La clase NP-completo

- ▶ Es el centro de la teoría de la complejidad computacional.
- ▶ Intuitivamente, agrupa a los problemas de decisión que son los *más difíciles* de la clase NP.
- ▶ Tienen dos propiedades sumamente importantes:
 - ▶ No se conocen algoritmos polinomiales para resolverlos.
 - ▶ Si existe un algoritmo polinomial para uno de ellos, existen algoritmos polinomiales para todos ellos.
- ▶ Una creencia es que los problemas de esta clase son intratables desde el punto de vista computacional, por lo que no existirían algoritmos eficientes para resolverlos.
- ▶ Pero ésto no está demostrado y podría no ser verdadero.

La clase NP-completo

Un problema de decisión Π es **NP-completo** si:

1. $\Pi \in \text{NP}$
2. $\forall \bar{\Pi} \in \text{NP}, \bar{\Pi} \leq_p \Pi$

Si un problema Π verifica la condición 2., Π es **NP-difícil** (es al menos tan *difícil* como todos los problemas de NP).

La clase NP-completo

Teorema (Cook, 1971 - Levin, 1973): SAT es NP-completo.

- ▶ SAT fue el primer problema que se demostró que permanece a la clase NP-completo, dando origen a esta clase.
- ▶ La demostración de Cook es directa: considera un problema genérico $\Pi \in \text{NP}$ y una instancia genérica $d \in D_\Pi$. A partir de la hipotética MTND que resuelve Π , genera en tiempo polinomial una fórmula lógica $\varphi_{\Pi,d}$ en forma normal (conjunción de disyunciones) tal que $d \in Y_\Pi$ si, y sólo si, $\varphi_{\Pi,d}$ es satisfactible.

¿Cómo se prueba que un problema es NP-completo?

- ▶ Usando la transitividad de las reducciones polinomiales, a partir de este primer resultado podemos probar que otros problemas son NP-completos.
- ▶ Si Π es un problema de decisión, podemos probar que $\Pi \in \text{NP-completo}$ encontrando otro problema Π_1 que ya sabemos que es NP-completo y demostrando que:
 1. $\Pi \in \text{NP}$
 2. $\Pi_1 \leq_p \Pi$

La segunda condición en la definición de problema NP-completo se deriva de la transitividad:

Sea Π' un problema cualquiera de NP. Como Π_1 es NP-completo, $\Pi' \leq_p \Pi_1$. Como probamos que $\Pi_1 \leq_p \Pi$, resulta $\Pi' \leq_p \Pi$.

La clase NP-completo

- ▶ Desde 1971, se ha probado la NP-completitud de muchos problemas usando el método anterior.
- ▶ A partir del Teorema de Cook-Levin, Richard Karp demostró en 1972 que otros 21 problemas son NP-completos.
- ▶ Actualmente se conocen más de 3.000 problemas NP-completos.

Problemas NP-completos - Clique

Dado un grafo $G = (V, X)$ y un entero positivo k , ¿ G tiene una clique de tamaño mayor o igual a k ?

► CLIQUE \in NP: Ya lo hicimos

► SAT \leq_p CLIQUE:

Dada una instancia de SAT, I_S , con k_S cláusulas, definimos $k_C = k_S$ y $G = (V, X)$ de la siguiente manera:

- Para cada literal x en la cláusula i , ponemos un vértice $(x, i) \in V$
- $((x, i), (y, j)) \in X$ si y sólo si $x \neq \neg y$ y $i \neq j$.

Tenemos que ver que I_S es satisficible si, y sólo si, G tiene una clique de tamaño mayor o igual a k .

Problemas NP-completos - Clique

- ▶ Si I_S es satisfacible:
 - ▶ Existe un conjunto de literales que hacen verdaderas las k_S cláusulas simultáneamente.
 - ▶ Tomando para cada cláusula un literal de este conjunto, representarán distintos vértices en G todos adyacentes entre sí (por ser de distintas cláusulas).
 - ▶ Estos vértices forman una clique de tamaño $k_S = k_C$ en G .
- ▶ Si H es una clique de G de tamaño mayor o igual a k_C :
 - ▶ Cada vértice de H representará un literal de una cláusula distinta que pueden ser simultáneamente verdaderos por no ser complementarios.
 - ▶ Asignándole valor verdadero a estos literales, I_S toma valor verdadero.
 - ▶ Por lo tanto, I_S es satisfacible.

Problemas NP-completos - 3-SAT

El problema 3-SAT es un caso particular del problema SAT, en el cual se pide que cada cláusula tenga exáctamente tres literales.

1. Como 3-SAT es un caso particular de SAT y ya sabemos que $\text{SAT} \in \text{NP}$, podemos decir que 3-SAT está en NP.
2. Para probar que 3-SAT es NP-completo, vamos a reducir polinomialmente SAT a 3-SAT:
 - ▶ Tomemos una instancia genérica de SAT, $\varphi = C_1 \wedge \cdots \wedge C_m$ sobre las variables booleanas x_1, \dots, x_n .
 - ▶ Vamos a reemplazar cada C_i por una conjunción de disyunciones φ'_i , donde cada disyunción tenga tres literales,
 - ▶ de manera que φ sea satisfactible si, y sólo si, $\varphi' = \varphi'_1 \wedge \cdots \wedge \varphi'_m$ lo es.

Problemas NP-completos - 3-SAT

- Si C_i tiene tres literales:

$$\varphi'_i = C_i.$$

- C_i tiene dos literales, a_1 y a_2 , agregamos una variable nueva y y definimos:

$$C_i = (a_1 \vee a_2) \rightarrow \varphi'_i = (a_1 \vee a_2 \vee y^i) \wedge (a_1 \vee a_2 \vee \neg y^i).$$

- Si C_i tiene $k \geq 4$ literales, agregamos $k - 3$ variables nuevas:

$$C_i = (a_1 \vee a_2 \vee \dots \vee a_r \vee \dots \vee a_{k-1} \vee a_k) \rightarrow$$
$$\varphi'_i = (a_1 \vee a_2 \vee y_1^i) \wedge \dots \wedge (\neg y_{r-2}^i \vee a_r \vee y_{r-1}^i) \wedge \dots \wedge (\neg y_{k-3}^i \vee a_{k-1} \vee a_k)$$

$$C_i = (a_1 \vee a_2 \vee a_3 \vee a_4 \vee a_5) \rightarrow$$
$$\varphi'_i = (a_1 \vee a_2 \vee y_1^i) \wedge (\neg y_1^i \vee a_3 \vee y_2^i) \wedge (\neg y_2^i \vee a_4 \vee a_5)$$

Problemas NP-completos - 3-SAT

Falta ver que φ es satisfactible si, y sólo si, $\varphi' = \varphi'_1 \wedge \dots \wedge \varphi'_m$ lo es.

- ▶ Supongamos que φ es satisfacible. Sea x_1^*, \dots, x_n^* valores de las variables x_1, \dots, x_n que hacen verdadera φ .
 - ▶ Entonces por lo menos un literal a_j^i de cada cláusula C_i tiene que ser verdadero (si hay más de uno elegimos el primero por ejemplo).
 - ▶ Para ver que φ' es satisfacible, mantenemos el valor de las variables originales $x_j = x_j^*$ para $j = 1, \dots, n$.
 - ▶ Si el literal a_j^i es verdadero, fijamos y_1^i, \dots, y_{j-2}^i en verdadero y $y_{j-1}^i, \dots, y_{k-3}^i$ en falso.
 - ▶ Esto hará a la cláusula φ'_i verdadera, haciendo a φ verdadera.

Problemas NP-completos - 3-SAT

- ▶ Ahora supongamos que $\varphi' = \varphi'_1 \wedge \cdots \wedge \varphi'_m$ es verdadera.
 - ▶ Al menos uno de los literales a_j^i de la conjunción de cláusulas φ'_i debe ser verdadero, de lo contrario y_1^i debe ser verdadera, pero eso impone que y_2^i sea verdadera, y así siguiendo.
 - ▶ Pero ésto hará que la última cláusula φ'_m sea falsa, contradiciendo que φ' es verdadera.
 - ▶ Ésto asegura que tomando esos valores de las variables x_j , la cláusula C_i es verdadera.

Problemas NP-completos - Coloreo

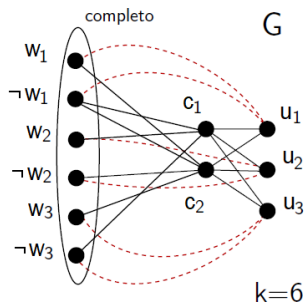
- ▶ Es fácil probar que COLOREO es **NP**.
- ▶ Para probar que COLOREO es **NP-completo**, vamos a reducir polinomialmente SAT a COLOREO.
 - ▶ Tomemos una instancia genérica de SAT, $\varphi = C_1 \wedge \dots \wedge C_m$ sobre las variables x_1, \dots, x_n .
 - ▶ Vamos a construir un grafo G y determinar un número k
 - ▶ de manera que φ sea satisfactible si, y sólo si, G se puede colorear con k -colores.

Problemas NP-completos - Coloreo

Definimos $k =$ dos veces la cantidad de variables de φ y $G = (V_1 \cup V_2 \cup V_3, X)$ como:

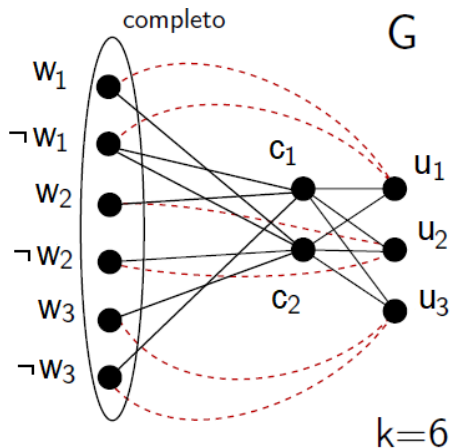
- ▶ V_1 : dos vértice w_i y $\neg w_i$ por cada variable x_i (representando a todos los posibles literales), todos adyacentes entre si, formando un K_k .
- ▶ V_2 : un vértice c_j por cada cláusula C_j , adyacente a los literales de V_1 que no aparecen en la cláusula.
- ▶ V_3 : un vértice u_i por cada variable x_i , adyacente a todos los vértices de V_2 y a los literales de V_1 correspondientes a otras variables (u_i no adyacente a w_i ni a $\neg w_i$).

$$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3)$$



Problemas NP-completos - Coloreo

$$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3)$$



Problemas NP-completos - Coloreo

Falta ver que φ es satisfactible si, y sólo si, G es k -coloreable.

- Supongamos que φ es satisfactible. Sea x_1^*, \dots, x_n^* valores de las variables x_1, \dots, x_n que hacen verdadera φ .

Pintamos a G con k colores de la siguiente forma:

- Para cada vértice de V_1 obviamente utilizamos un color distinto.
- Al vértice $c_j \in V_2$ lo pintamos del color del vértice de V_1 correspondiente a un literal que la hace verdadera.
- Si $x_i^* = V$, pintamos a $u_i \in V_3$ con el mismo color que $\neg w_i$, y si $x_i^* = F$, con el mismo color que w_i .

Obtenemos así un k -coloreo de G .

Problemas NP-completos - Coloreo

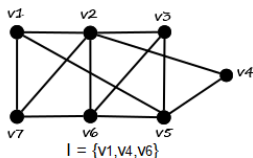
- ▶ Consideremos ahora que G es k -coloreable.
 - ▶ En un k -coloreo de G , cada uno de los k colores se utiliza exactamente una vez en los vértices de V_1 .
 - ▶ Un vértice $u_i \in V_3$ tendrá el mismo color que w_i o de $\neg w_i$, ya que es adyacente al resto de los vértices de V_1 .
 - ▶ El color del vértice c_j será el mismo que tiene algún vértice de V_1 representando a un literal de la cláusula C_j (porque es adyacente al resto de los vértices de V_1) y distinto a todos los colores de los vértices de V_3 .
 - ▶ Si u_i tiene el mismo color que w_i , le asignamos valor F a la variable x_i , mientras que si tiene el mismo color que $\neg w_i$, le asignamos valor V.
 - ▶ Entonces, el vértice c_j comparte color con un literal que es verdadero en esta asignación, haciendo este literal verdadera a la cláusula C_j .
 - ▶ Por lo tanto esta asignación hace verdadera a φ , demostrando que φ es satisfacible.

Problemas NP-completos - Conjunto independiente

1. Es fácil probar que CONJUNTO INDEPENDIENTE es **NP**.
2. Para probar que es **NP-difícil**, vamos a reducir polinomialmente CLIQUE a CONJUNTO INDEPENDIENTE:
 - ▶ Tomemos una instancia genérica de CLIQUE, I_C , esto es un grafo G y $k \in \mathbb{N}$ y definamos la instancia I_{CI} de CONJUNTO INDEPENDIENTE, \bar{G} y k .
 - ▶ Como una clique S en un grafo G es un conjunto independiente en \bar{G} , se cumple que G tiene una clique de tamaño mayor o igual a k si, y sólo si, \bar{G} tiene un conjunto independiente de tamaño mayor o igual a k .
 - ▶ Ésto muestra que CLIQUE responde que SI para I_C si, y sólo si, CONJUNTO INDEPENDIENTE responde SI para I_{CI} .

Problemas NP-completos - Recubrimiento de Aristas

Dado un grafo $G = (V, X)$, un *recubrimiento de las aristas* de G , es un conjunto $R_a \subseteq V$ de vértices tal que para todo $e \in X$, e es incidente al menos a un vértice $v \in R_a$.



$$R_a = \{v2, v3, v5, v7\}$$

El problema de RECUBRIMIENTO DE ARISTAS (en su versión de decisión) es: dados un grafo $G = (V, X)$ y $k \in \mathbb{N}$, ¿ G tiene un recubrimiento de aristas de tamaño menor o igual a k ?

Problemas NP-completos - Recubrimiento de Aristas

1. Es fácil probar que RECUBRIMIENTO DE ARISTAS es **NP**.
2. Para probar que es **NP-completo**, vamos a reducir polinomialmente CONJUNTO INDEPENDIENTE a RECUBRIMIENTO DE ARISTAS.
 - ▶ Tomamos una instancia genérica de CONJUNTO INDEPENDIENTE, $G = (V, X)$ de n vértices y $k \in \mathbb{N}$, $k \leq n$ y la transformamos a la instancia de RECUBRIMIENTO DE ARISTAS, G y $k' = n - k$.
 - ▶ Dado $G = (V, X)$, S es conjunto independiente si, y sólo si, $V \setminus S$ es recubrimiento de aristas.
 - ▶ Esto asegura que G tendrá un conjunto independiente de tamaño mayor o igual a k si, y sólo si, G tiene un recubrimiento de aristas de tamaño menor o igual a k' .

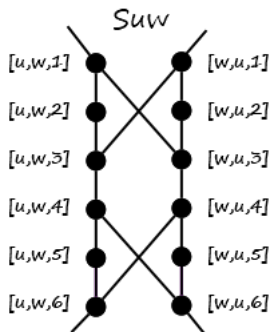
Problemas NP-completos - Circuito hamiltoniano

1. Ya vimos que CIRCUITO HAMILTONIANO es NP.
2. Para probar que es **NP-completo**, vamos a reducir polinomialmente RECUBRIMIENTO DE ARISTAS a CIRCUITO HAMILTONIANO.

Tomamos una instancia genérica de RECUBRIMIENTO DE ARISTAS, $G = (V, X)$ de n vértices y $k \in \mathbb{N}$ y la transformamos a una instancia de CIRCUITO HAMILTONIANO, $G' = (V', X')$, de forma tal que G tenga un recubrimiento de aristas de k o menos vértices si, y sólo si, G' es hamiltoniano.

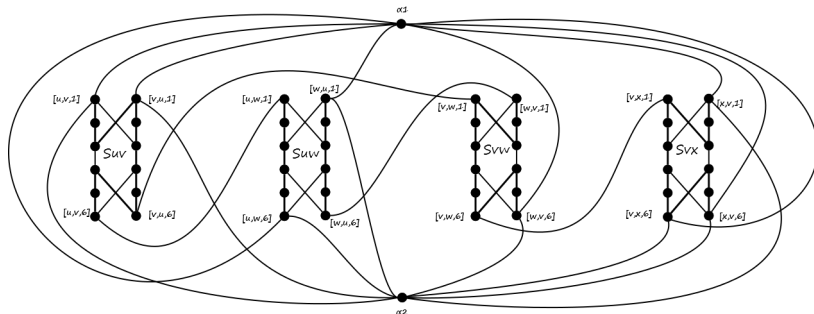
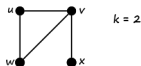
Problemas NP-completos - Circuito hamiltoniano

Por cada arista $(u, w) \in X$, colocamos 12 vértices en G' con la siguiente estructura S_{uw} :



Problemas NP-completos - Circuito hamiltoniano

- ▶ Dado un vértice $u \in V$, sean $w_1, \dots, w_{d(u)}$ sus adyacentes. Entonces hacemos adyacentes en G' los vértices $[u, w_i, 6]$ y $[u, w_{i+1}, 1]$ para $i = 1, \dots, d(u) - 1$.
- ▶ Además, colocamos en V' k nuevos vértices $\alpha_1, \dots, \alpha_k$ y los hacemos adyacentes a $[u, w_1, 1]$ y $[u, w_{d(u)}, 6]$.



Problemas NP-completos - TSP

1. Ya vimos que TSP es NP.
2. Para probar que es NP-completo, vamos a reducir polinomialmente CIRCUITO HAMILTONIANO a TSP.

Dado un grafo G vamos a construir un grafo completo G' con peso en las aristas y definir $k \in \mathbb{Z}_{\geq 0}$, de forma tal que G sea hamiltoniano si, y sólo si, G' tiene un circuito hamiltoniano de peso menor o igual a k .

Problemas NP-completos - TSP

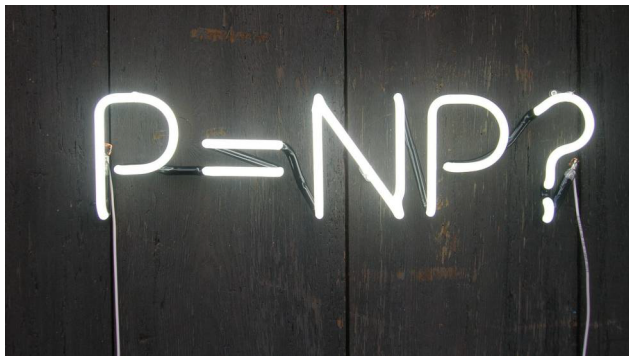
- ▶ Si G tiene n vértices, G' será el grafo completo de n vértices.
- ▶ Fijaremos peso 0 para las aristas de G' que están en G y 1 para las que no están y definiremos $k = 0$.
- ▶ Entonces, claramente, G' tiene un circuito hamiltoniano de peso 0 si, y sólo si, G es hamiltoniano.

$P \neq NP??$ $P = NP??$

Lema: Si existe un problema en NP-completo $\cap P$, entonces $P=NP$.

- ▶ Si $\Pi \in \text{NP-completo} \cap P$, existe un algoritmo polinomial que resuelve Π , por estar Π en P .
- ▶ Por otro lado, como Π es NP-completo, para todo $\Pi' \in \text{NP}$, $\Pi' \leq_p \Pi$.
- ▶ Sea $\Pi' \in \text{NP}$:
 - ▶ Aplicando la reducción polinomial que transforma instancias de Π' en instancias de Π
 - ▶ y luego el algoritmo polinomial que resuelve Π ,se obtiene es un algoritmo polinomial que resuelve Π' .

$P \neq NP??$ $P = NP??$



- ▶ Hasta el momento no se conoce ningún problema en $NP\text{-}completo \cap P$.
- ▶ Tampoco se ha demostrado que exista algún problema en $NP \setminus P$. En ese caso se probaría que $P \neq NP$.

La clase Co-NP

Vimos que circuito hamiltoniano está en la clase **NP**. Pero consideremos ahora su versión *inversa*, *circuito hamiltoniano complemento*:

Dado un grafo G , ¿es G no hamiltoniano?

- ▶ ¿Estará este problema también en **NP**? No sabemos la respuesta.
- ▶ Hasta el momento, la forma de verificar que un grafo general no tiene un circuito hamiltoniano es listar todas las permutaciones de sus vértices y verificar que ninguna define un circuito.
- ▶ Este certificado obviamente no es polinomial, por lo tanto no nos sirve para responder la pregunta.

La clase Co-NP

El *problema complemento* de un problema de decisión Π , Π^c , es el problema de decisión cuyo conjunto de instancias es igual al de Π y responde *SI* para las instancias que Π responde *NO* y viceversa.

Es decir Π^c es el problema de decisión tal que:

- ▶ $D_{\Pi^c} = D_{\Pi}$ y
- ▶ $Y_{\Pi^c} = D_{\Pi} \setminus Y_{\Pi}$

La clase Co-NP

- ▶ El problema de primalidad y el problema de número compuesto son problemas complementarios.
- ▶ El problema complemento de TSP es:
Dado un grafo $G = (V, X)$ con peso en sus aristas y $k \in \mathbb{N}$, ¿es verdad que todos los circuitos hamiltonianos de G tienen peso mayor que k ?
- ▶ Los problemas:
 - ▶ Dado un grafo $G = (V, X)$, ¿es conexo?
 - ▶ Dado un grafo $G = (V, X)$, ¿es desconexo?son problemas complementarios.

La clase Co-NP

Proposición: Si un problema Π pertenece a **P**, entonces Π^c también pertenece a **P**.

Este argumento no aplica para la clase **NP**:

- ▶ Si un problema Π está en **NP** no sirve este argumento para demostrar que Π^c está en **NP**.
- ▶ Es más, no se sabe si ésto es cierto o no.

La clase Co-NP

Un problema de decisión pertenece a la clase **Co-NP** si dada una instancia de **NO** y evidencia polinomial de la misma, puede ser verificada en tiempo polinomial.

La relación que podemos plantear entre estas clases es:

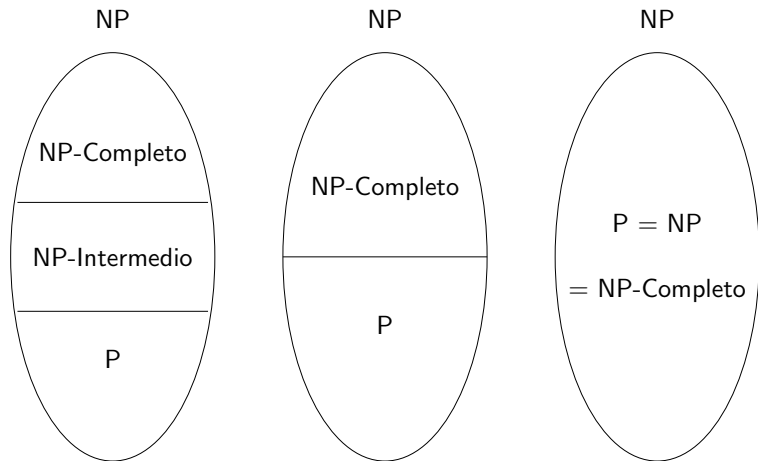
Proposición: Si un problema Π pertenece a **NP**, entonces Π^c pertenece a **Co-NP**.

Problemas abiertos de Teoría de Complejidad

Con estas nuevas definiciones tenemos los siguientes problemas abiertos:

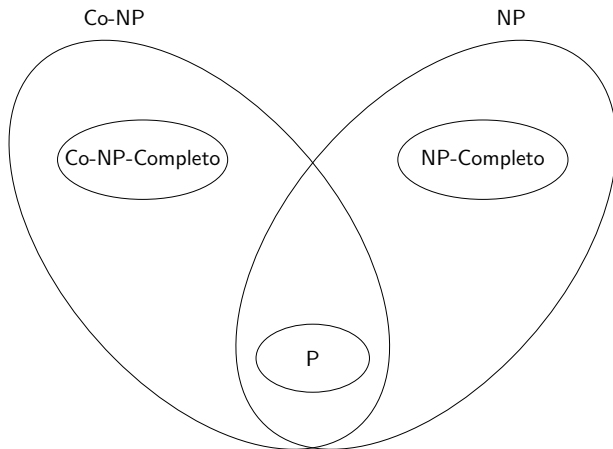
- ▶ ¿Es $P=NP$?
- ▶ ¿Es $Co-NP=NP$?
- ▶ ¿Es $P=Co-NP \cap NP$?

Las incógnitas...



Tres mapas posibles para las clases de complejidad

Las incógnitas...



Situación si se probara que $P \neq NP$, $NP \neq Co - NP$,
 $P \neq Co - NP \cap NP$

Extensión de un problema

- ▶ El problema Π es una *restricción* de un problema $\bar{\Pi}$ si el dominio de Π está incluido en el de $\bar{\Pi}$.
- ▶ Si Π es una restricción de $\bar{\Pi}$, se dice que $\bar{\Pi}$ es una *extensión* o *generalización* de Π .
- ▶ Es intuitivo pensar que cuanto más general es el problema, más difícil es de resolver.
- ▶ Algunas veces, casos particulares (restricciones) de un problema **NP-completo** puede ser **P**.
- ▶ Pero no se puede dar la situación recíproca (ser el caso general más fácil que el caso particular).

Extensión de un problema

- ▶ 3-SAT es una restricción de SAT. Ambos son problemas NP-completos.
- ▶ 2-SAT es una restricción de SAT. 2-SAT es polinomial, mientras que SAT es NP-completo.
- ▶ COLOREO de grafos bipartitos es una restricción de COLOREO. Colorear un grafo bipartito es un problema polinomial, mientras COLOREO es NP-completo.
- ▶ CLIQUE de grafos planares es una restricción de CLIQUE. Encontrar una clique máxima de un grafo planar es un problema polinomial (porque sabemos que no puede tener a K_5 como subgrafo), mientras CLIQUE es NP-completo.
- ▶ CARTERO CHINO en grafos planares mixtos es una restricción de CARTERO CHINO en grafos mixtos. Ambos son problemas NP-completos.

Extensión de un problema

Si Π es una *restricción* de $\bar{\Pi}$, podemos deducir que:

- ▶ Si $\bar{\Pi} \in \mathbf{P}$, entonces $\Pi \in \mathbf{P}$.
- ▶ Si $\bar{\Pi} \in \mathbf{NP}$, entonces $\Pi \in \mathbf{NP}$.
- ▶ Si $\Pi \in \mathbf{NP-Completo}$, entonces $\bar{\Pi} \in \mathbf{NP-Difícil}$.

Algoritmos Pseudopolinomiales

- ▶ Un algoritmo para resolver un problema Π es *pseudopolinomial* si la complejidad del mismo es polinomial en función del valor de la entrada.
- ▶ El problema de la mochila es NP-Completo, sin embargo, existe un algoritmo de complejidad $\mathcal{O}(nB)$ que lo resuelve, donde n es la cantidad de objetos y B el peso máximo que se puede cargar en la mochila.

Una instancia de este problema es c_1, \dots, c_n, B , por lo tanto su tamaño es $\mathcal{O}(n \log B)$. Por lo tanto $\mathcal{O}(nB)$ no es polinomial en el tamaño de la instancia, pero sí en el valor de la instancia.

- ▶ Desde un punto de vista práctico, muchas veces los algoritmos pseudopolinomiales, a pesar de ser exponenciales, son aplicables.

Teoría de complejidad computacional

- ▶ ¿Qué hacer ante un problema del que no sabemos en que clase está?
- ▶ ¿Qué importancia tiene saber si un problema está en NP-completo desde el punto de vista teórico?
- ▶ ¿Qué importancia tiene la misma pregunta desde el punto de vista práctico, o sea ante una aplicación real que se quiere resolver?