Complejidad - Problemas NP-completos

Algoritmos y Estructuras de Datos III

Teoría de Complejidad

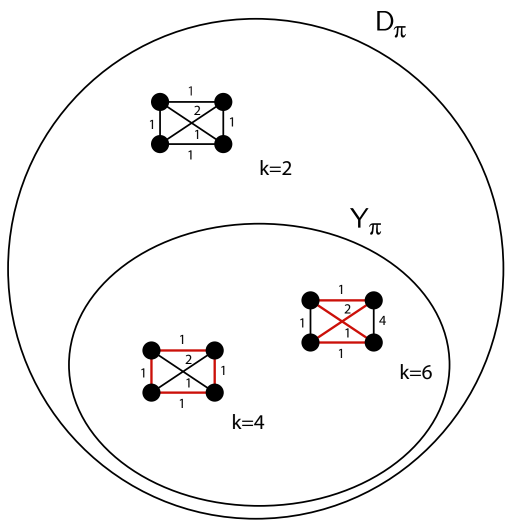
- Un **algoritmo eficiente** es un algoritmo de complejidad polinomial.
- Un problema está **bien resuelto** si se conocen algoritmos eficientes para resolverlo.
- El objetivo es clasificar los problemas según su complejidad.
- Un **problemas de decisión** es un problema cuya respuesta es **SI** o **NO**.
- La clasificación y estudio de teoría de complejidad se aplica a problemas de decisión.

Instancia de un problema Π

- 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?



Distintas versiones de un problema de optimización Π

Dada una instancia I del problema Π:

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

Ejemplo: Problema del viajante de comercio

Dado un grafo G con longitudes asignadas a sus aristas:

- 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 **optimización**: Determinar un circuito hamiltoniano de G de longitud mínima.
- Versión de **decisión**: Dado un número k, ¿existe un circuito hamiltoniano de G de longitud menor o igual a k?
- Versión de **localización**: Dado un número k, determinar un circuito hamiltoniano de G de longitud menor o igual a k.

Distintas versiones de un problema de optimización Π

¿Qué relación hay en la dificultad de resolver las distintas versiones de un mismo problema?

Si resolvemos el problema de decisión, podemos:

- Resolver el problema de evaluación usando búsqueda binaria sobre el parámetro k.
- Resolver el problema de localización resolviendo el problema de decisión para el parámetro k para una versión reducida de la instancia.
- Resolver el problema de optimización resolviendo el problema de decisión para el valor óptimo para una versión reducida de la instancia.

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 repuesta de longitud exponencial (ejemplo: pedir todos los circuitos hamiltonianos de longitud a lo sumo k).
- El problema es **indecidible** (ejemplo: problema de la parada).
- El problema es decidible pero no se conocen algoritmos polinomiales que lo resuelvan (no se sabe si es intratable o no).

Modelos de Computadoras

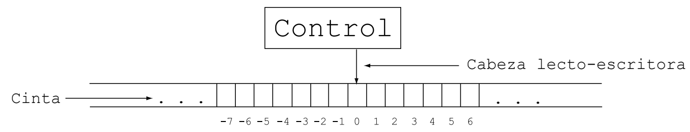
Modelos formales para expresar cualquier algoritmo:

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

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

Máquina de Turing Determinística

- Consiste de un conjunto finito de estados, una cabeza lecto-escritora y una cinta infinita en ambas direcciones con el siguiente esquema.



- Σ finito, el alfabeto; $\Gamma = \Sigma \cup \{*\}$;
- Q finito, el conjunto de estados;
- $q_0 \in Q$, estado inicial; $Q_f \subseteq Q$, estados finales (q_{si} y q_{no} para problemas de decisión)

Máquina de Turing Determinística

- Sobre la cinta está escrito la entrada, que es un string de símbolos de Σ y el resto de las celdas tiene $*$ (blancos).
- Se define un programa S como un conjunto de quíntuplas $S \subseteq Q \times \Gamma \times Q \times \Gamma \times M$, donde $M = \{+1, -1\}$ son los movimientos de la cabeza a derecha o izquierda (tabla finita de instrucciones).
- Para todo par (q_i, s_j), existe a lo sumo una quíntupla que comienza con ese par (máquina determinística).

Máquina de Turing Determinística

- Arranque:
  - máquina posicionada en el estado distinguido q_0, estado inicial
  - cabeza lecto-escritora ubicada en la celda inicial (0) de la cinta
- Terminación:
  - cuando no se puede inferir nuevas acciones para seguir
  - 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

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

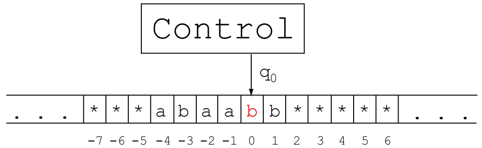
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



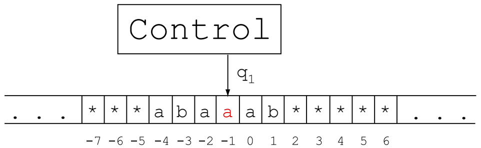
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



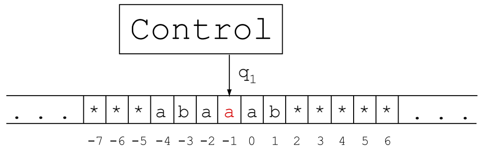
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



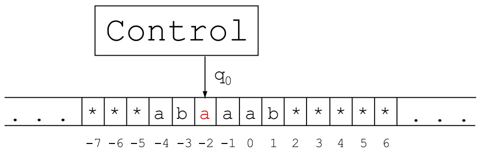
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



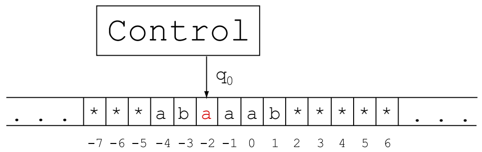
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



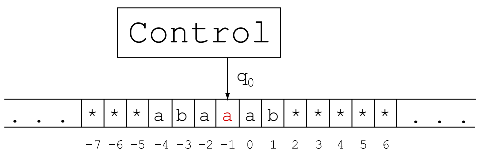
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



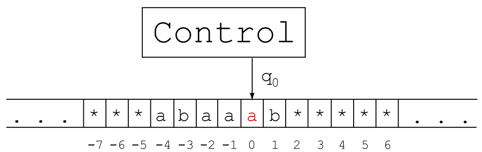
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



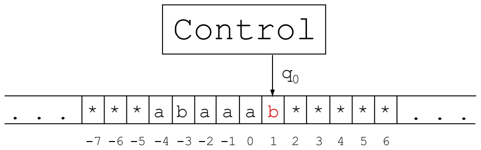
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



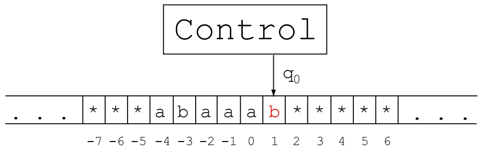
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



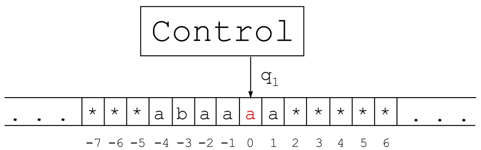
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



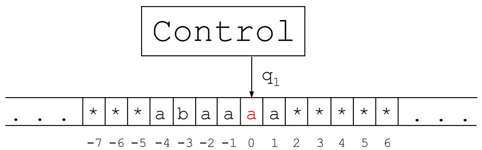
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



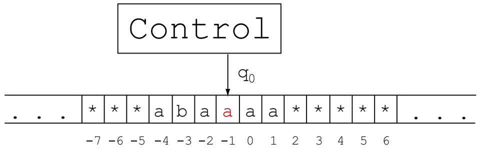
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



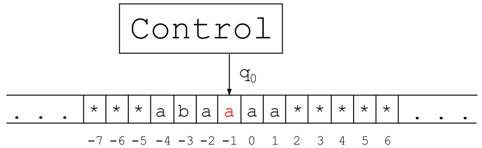
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



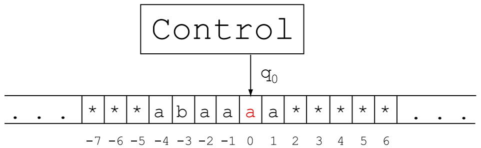
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



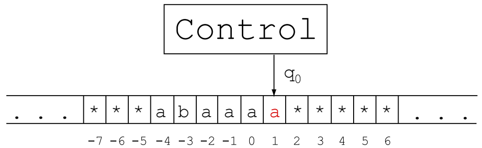
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



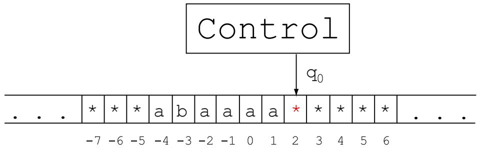
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



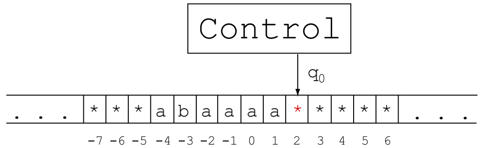
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

Ejemplo:

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



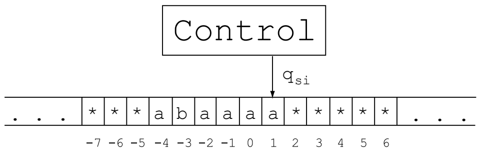
Máquina de Turing Determinística

La quíntupla $(q_i, s_h, q_j, s_k, +1)$ 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.

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).

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.

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

Máquina de Turing Determinística

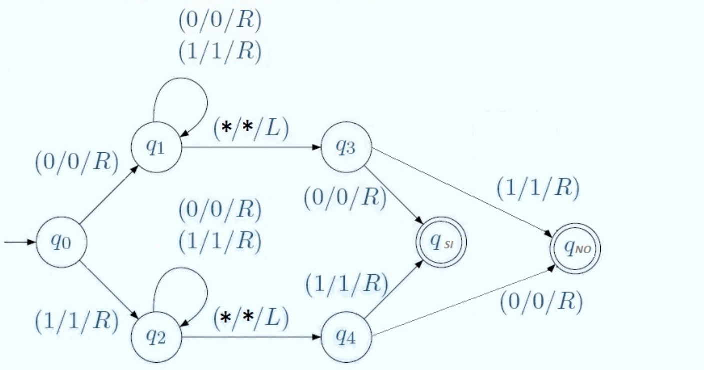
- Una máquina M resuelve el problema Π si para toda instancia alcanza un estado final y responde de forma correcta (o sea, termina en un estado final correcto).
- La complejidad de una MTD está dada por la cantidad de movimientos de la cabeza, desde el estado inicial hasta alcanzar un estado final, en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

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

- Existen otros modelos de computadoras determinísticas (máquina de Turing con varias cintas, Random Access Machines, etc.) pero puede probarse que son equivalentes en términos de la polinomialidad de los problemas a la MTD.

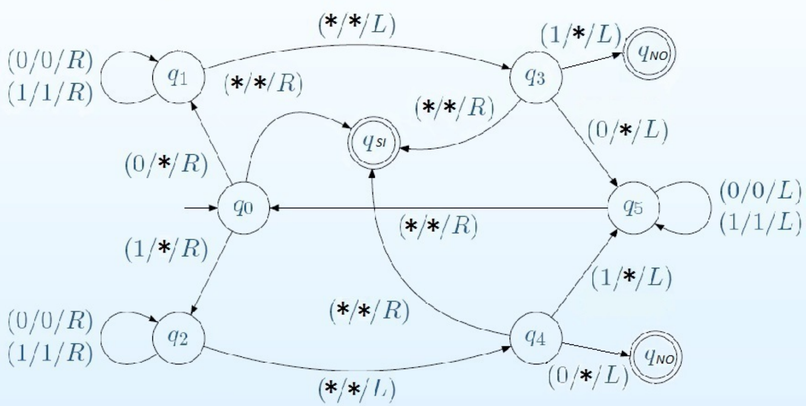
Máquina de Turing Determinística

Ejemplo 1: Máquina que acepta el lenguaje de palabras sobre $\{0,1\}$ que comienzan y acaban con el mismo símbolo



Máquina de Turing Determinística

Ejemplo 2: Máquina que acepta el lenguaje de palíndromos sobre $\{0,1\}$



La clase P

Un problema Π está en **P** si:

- Existe una MTD de complejidad polinomial que lo resuelve.

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

Que es equivalente a:

- Existe un algoritmo polinomial que lo resuelve.

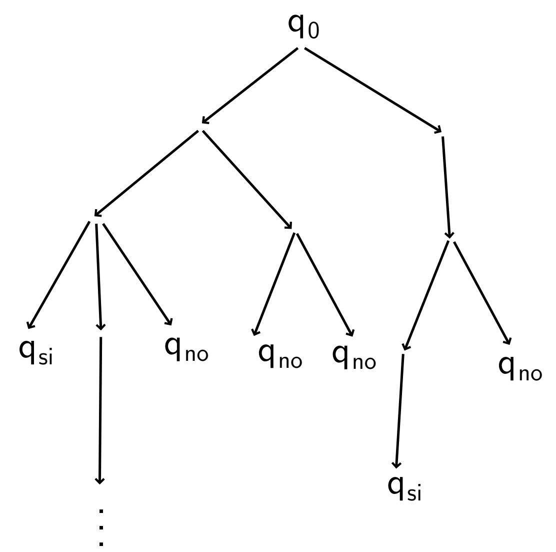
Máquinas de Turing No Determinísticas (MTND)

- No se pide unicidad de la quíntupla que comienza con cualquier par (q_i, s_j).
- Un programa correspondiente en una MTND es una tabla que mapea un par (q_i, t_i) a un **conjunto** de ternas $(q_f, t_f, \{+1, -1\})$.
- Esto admite dos interpretaciones equivalentes:
  - En cada paso se selecciona una de las alternativas posibles.
  - En cada paso se continúa la ejecución en paralelo de las distintas alternativas, generando una copia de la MTND por cada alternativa.

Máquinas de Turing No Determinísticas (MTND)

- Una MTND resuelve un 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}.
- La complejidad temporal de una MTND se define como el máximo número de pasos que toma como mínimo reconocer una instancia de Y_Π (instancia con respuesta SI) en función de su tamaño.

Máquinas de Turing No Determinísticas (MTND)



Máquinas de Turing No Determinísticas (MTND)

- Una MTND es **polinomial** para Π cuando existe una función polinomial $T(n)$ de manera que para toda instancia de Y_Π de tamaño n, alguna de las ramas termina en estado q_{si} en a lo sumo $T(n)$ pasos.

La clase NP

Un problema $\Pi \in$ **NP** (polinimial 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 que garantiza que la respuesta es **SI**, y esta garantía puede ser verificada en tiempo polinomial.
- La clase NP se puede definir como el conjunto de problemas de decisión que se pueden resolver por un algoritmo polinomial no-determinístico.

La clase NP

Lema

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

Ejemplo: Conjunto independiente máximo

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

guess(S): función multivaluada que retorna un nuevo elemento de S.

$I := \emptyset$
mientras $S \neq \emptyset$ **hacer**
    $v := guess(S)$
    $S := S \setminus \{v\}$
    si $\Gamma(v) \cap I = \emptyset$ **entonces** $I := I \cup \{v\}$
    si $|I| \geq k$ **entonces retornar** *SI*
fin mientras
retornar *NO*

Ejemplo: Conjunto independiente máximo

O equivalentemente:

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

Cetificado $S \subseteq V$: conjunto de vértices

verificar que $|S| \geq k$
verificar que S es conjunto independiente

Es posible verificar en tiempo polinomial que S garantiza que la instancia tiene respuesta **SI**.

Las clases P y NP

- P $\subseteq$ NP.

Las clases P y NP

- P $\subseteq$ NP.
- **Problema abierto: ¿Es P = NP?**

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.

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.

Ejemplos de problemas en NP

- Suma de enteros.
- Multipliación de enteros.
- Árbol generador mínimo.
- Clique máxima.
- Camino mínimo entre un par de nodos.
- Problema del viajante de comercio.
- Conjunto independiente de cardinal máximo.
- Problema de satisfabilidad (SAT): Dado un conjunto de claúsulas $C_1, \dots, C_m$ formadas por literales basados en las variables booleanas $X = \{x_1, \dots, x_n\}$, determinar si hay una asignación de valores de verdad a las variables de X tal que la expresión $C_1 \wedge C_2 \wedge \dots \wedge C_m$ sea verdadera.

Transformaciones polinomiales

- Una **transformación o reducción polinomial** de un problema de decisión Π_1 a uno Π_2 es una función polinomial $f : D_{\Pi_1} \to D_{\Pi_2}$ que transforma una instancia de I_1 de Π_1 en una instancia $f(I_1) = I_2$ de Π_2 tal que $I_1 \in Y_{\Pi_1} \Leftrightarrow I_2 \in Y_{\Pi_2}$.
- El problema de decisión Π_1 se reduce polinomialmente a otro problema de decisión Π_2, $\Pi_1 \leq_p \Pi_2$, si existe una transformación polinomial de Π_1 a Π_2.
- Las reducciones polinomiales son transitivas:

$$\text{si } \Pi_1 \leq_p \Pi_2 \text{ y } \Pi_2 \leq_p \Pi_3 \text{ entonces } \Pi_1 \leq_p \Pi_3.$$

La clase NP-completo

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

1. $\Pi \in NP$
2. $\forall \Pi' \in NP$, $\Pi' \leq_p \Pi$

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

- El problema SAT consiste en decidir si, dada una fórmula lógica φ expresada como conjunción de disyunciones (ej: $\varphi = x_1 \wedge (x_2 \vee \neg x_1) \wedge (x_3 \vee \neg x_4 \vee x_1))$, existe una valuación de sus variables que haga verdadera φ.
- Teorema (Cook, 1971 - Levin, 1973): SAT es NP-completo.

  La demostración de Cook es directa: considera un problema genérico $\pi \in$ NP y una instancia genérica $d \in D_\pi$. A partir de la hipotética NDTM 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 $\Pi \in$ NP-completo encontrando otro problema Π_1 que ya sabemos que es NP-completo y demostrando que:
  1. $\Pi \in$ NP
  2. $\Pi_1 \leq_p \Pi$

¿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 $\Pi \in$ NP-completo encontrando otro problema Π_1 que ya sabemos que es NP-completo y demostrando que:
  1. $\Pi \in$ 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?) es NP-completo.
- Conjunto independiente (dado un grafo G y un entero positivo k, ¿G tiene un conjunto independiente de tamaño mayor o igual a k?) es NP-completo.
- Recubrimiento de aristas (dado un grafo G y un entero positivo k, ¿G tiene un recubrimiento de aristas de tamaño menor o igual a k?) es NP-completo

Reducción de SAT a 3-SAT

El problema 3-SAT es una variante del problema SAT, en el cual cada cláusula tiene exactamente tres literales. Como es una restricción del dominio de SAT, está en NP, y en principio es "no más difícil" que SAT.

Reducción de SAT a 3-SAT

El problema 3-SAT es una variante del problema SAT, en el cual cada cláusula tiene exactamente tres literales. Como es una restricción del dominio de SAT, está en NP, y en principio es "no más difícil" que SAT.

Para probar que 3-SAT es NP-completo, vamos entonces a reducir SAT a 3-SAT.

Reducción de SAT a 3-SAT

El problema 3-SAT es una variante del problema SAT, en el cual cada cláusula tiene exactamente tres literales. Como es una restricción del dominio de SAT, está en NP, y en principio es "no más difícil" que SAT.

Para probar que 3-SAT es NP-completo, vamos entonces a reducir SAT a 3-SAT.

Tomemos una instancia genérica de SAT $\varphi = C_1 \wedge \cdots \wedge C_m$. Vamos a reemplazar cada C_i por una conjunción de disyunciones φ'_i, donde cada disyunción tenga tres literales, y de manera que φ sea satisfactible si y sólo si $\varphi'_1 \wedge \cdots \wedge \varphi'_m$ lo es.

Reducción de SAT a 3-SAT

- Si C_i tiene tres literales:

$$\varphi_i' = C_i.$$

Reducción de SAT a 3-SAT

- Si C_i tiene tres literales:

$$\varphi_i' = C_i.$$

- C_i tiene dos literales, x_1 y x_2, agregamos una variable nueva y y definimos:

$$C_i = (x_1 \vee x_2) \to \varphi_i' = (x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y).$$

Reducción de SAT a 3-SAT

- Si C_i tiene tres literales:

$$\varphi_i' = C_i.$$

- C_i tiene dos literales, x_1 y x_2, agregamos una variable nueva y y definimos:

$$C_i = (x_1 \vee x_2) \to \varphi_i' = (x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y).$$

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

$$C_i = (x_1 \vee x_2 \vee \ldots \vee x_r \vee \ldots \vee x_{k-1} \vee x_k) \to$$

$$\varphi_i' = (x_1 \vee x_2 \vee y_1) \wedge \ldots \wedge (\neg y_{r-2} \vee x_r \vee y_{r-1}) \wedge \ldots \wedge (\neg y_{k-3} \vee x_{k-1} \vee x_k)$$

Reducción de SAT a 3-SAT

- Si C_i tiene tres literales:

$$\varphi_i' = C_i.$$

- C_i tiene dos literales, x_1 y x_2, agregamos una variable nueva y y definimos:

$$C_i = (x_1 \vee x_2) \to \varphi_i' = (x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y).$$

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

$$C_i = (x_1 \vee x_2 \vee \ldots \vee x_r \vee \ldots \vee x_{k-1} \vee x_k) \to$$

$$\varphi_i' = (x_1 \vee x_2 \vee y_1) \wedge \ldots \wedge (\neg y_{r-2} \vee x_r \vee y_{r-1}) \wedge \ldots \wedge (\neg y_{k-3} \vee x_{k-1} \vee x_k)$$

Ej: $C_i = (x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \to \varphi_i' = (x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee x_5)$

Coloreo es NP-completo

- Probar que coloreo es NP.

Coloreo es NP-completo

- Probar que coloreo es NP.
- Para probar que coloreo es NP-completo, vamos entonces a reducir SAT a coloreo.

Coloreo es NP-completo

- Probar que coloreo es NP.
- Para probar que coloreo es NP-completo, vamos entonces a reducir SAT a coloreo.

  Tomemos una instancia genérica de SAT $\varphi = C_1 \wedge \cdots \wedge C_m$. 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.

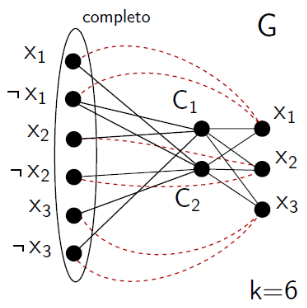
Reducción de SAT a coloreo

G tiene:

- V_1: un vértice por cada variable negada y afirmada, todos adyacentes entre si.
- V_2: un vértice por cada cláusula, adyacente a los literales de V_1 que no aparecen en la cláusula.
- V_3: otro vértice por cada variable, adyacente a todo V_2 y a los literales de V_1 correspondientes a otras variables.

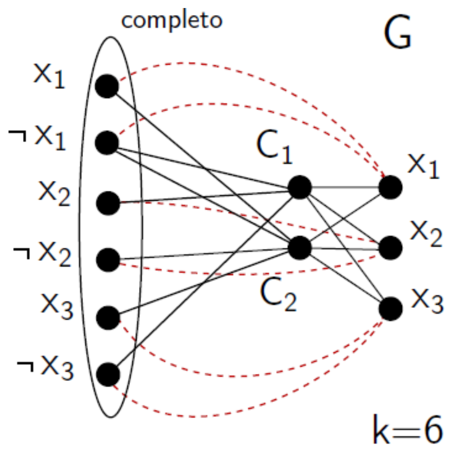
k = dos veces la cantidad de variables.

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



Reducción de SAT a coloreo

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



Conjunto independiente es NP-completo

- Probar que conjunto independiente es NP.

Conjunto independiente es NP-completo

- Probar que conjunto independiente es NP.
- Para probar que conjunto independiente es NP-completo, vamos a reducir 3-SAT a conjunto independiente.

Conjunto independiente es NP-completo

- Probar que conjunto independiente es NP.
- Para probar que conjunto independiente es NP-completo, vamos a reducir 3-SAT a conjunto independiente.

  Tomemos una instancia genérica de 3-SAT $\varphi = C_1 \wedge \ldots \wedge C_m$, con C_j la disyunción de 3 literales. Vamos a construir un grafo G y determinar un número k de manera que φ sea satisfactible si y sólo si G tiene un conjunto independiente de tamaño mayor o igual a k.

Reducción de 3-SAT a conjunto independiente

- Si el literal x_i figura en la clausula C_j agregamos el nodo v_{ij}.
- Si el literal $\neg x_i$ figura en la clausula C_j agregamos el nodo $v_{\neg ij}$.
- Agregamos una arista entre todos los nodos provenientes de la misma clausula (formando un K_3 para cada clausula).
- Agregamos aristas entre nodos representando a un literal y su negación.
- Definimos $k = m$.

¿¿P $\neq$ NP??

- Si existe un problema en NP-c $\cap$ P, entonces P=NP.

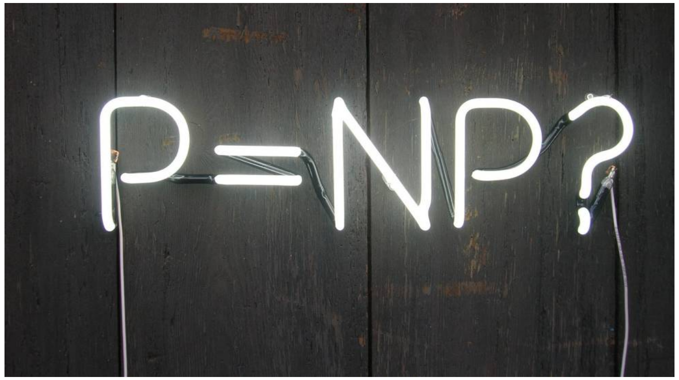
¿¿P $\neq$ NP??

- Si existe un problema en NP-c $\cap$ P, entonces P=NP.
  - Si $\Pi \in$ NP-c $\cap$ P, existe un algoritmo polinomial que resuelve Π, por estar Π en P. Por otro lado, como Π es NP-completo, para todo $\Pi' \in$ NP, $\Pi' \leq_p \Pi$.

¿¿P $\neq$ NP??

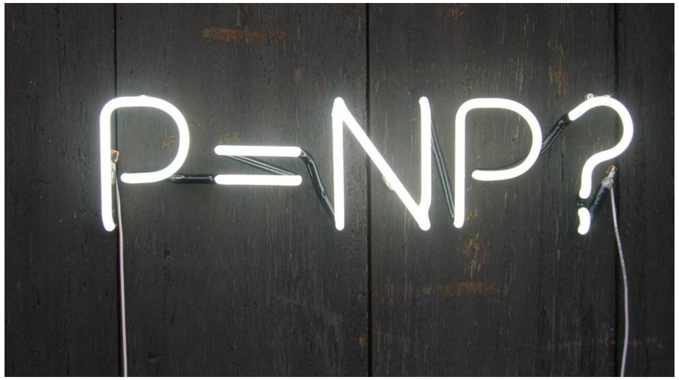
- Si existe un problema en NP-c $\cap$ P, entonces P=NP.
  - Si $\Pi \in$ NP-c $\cap$ P, existe un algoritmo polinomial que resuelve Π, por estar Π en P. Por otro lado, como Π es NP-completo, para todo $\Pi' \in$ NP, $\Pi' \leq_p \Pi$.
  - Sea $\Pi' \in$ NP. Aplicando la reducción polinomial que transforma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

¿¿P $\neq$ NP??



- Hasta el momento no se conoce ningún problema en NP-c$\cap$ P.

¿¿P $\neq$ NP??



- Hasta el momento no se conoce ningún problema en NP-c$\cap$ P.
- Tampoco se ha demostrado que un problema esté en NP$\setminus$P. En ese caso se probaría que P $\neq$ NP.

La clase Co-NP

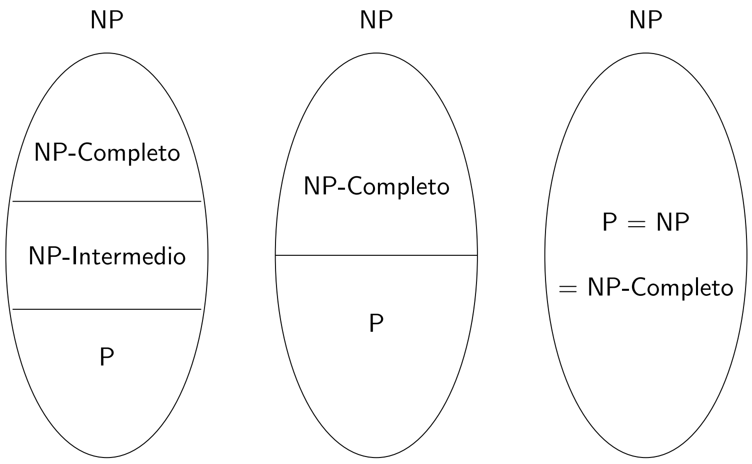
- Un problema de decisión pertenece a la **clase Co-NP** si dada una instancia de **NO** y evidencia de la misma, puede ser verificada en tiempo polinomial.
- El **problema complemento** de un problema de decisión Π, Π^c, es el problema de decisión que responde al complemento de la decisión de Π.
  Ejemplo: problema de primalidad y problema de número compuesto.
- El problema Π^c tiene respuesta NO si y sólo si Π tiene respuesta SI.
- La clase CO-NP es la clase de los problemas complemento de los problemas de la clase NP.
- La clase de los problemas polinomiales (P), está contenida también en Co-NP.

Problemas abiertos de Teoría de Complejidad

Con estas nuevas definiciones tenemos los siguientes problemas abiertos:

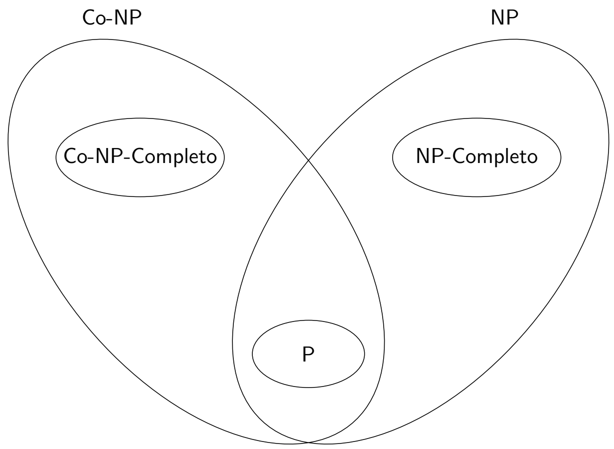
- ¿Es P=NP?
- ¿Es Co-NP=NP?
- ¿Es P=Co-NP $\cap$ NP?

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 Π' si el dominio de Π está incluído en el de Π'.

- Se dice que Π' es una **extensión** de Π.
- Si $\Pi \in$ NP-completo, entonces $\Pi' \in$ NP-difícil.

Ejemplos:

- Viajante de comercio es una extensión de Circuito Hamiltoniano.
- 3-SAT es una restricción de SAT. Sabiendo que SAT es NP-completo, ¿podemos sacar de esto una conclusión sobre la complejidad de 3-SAT?

Algoritmos Pseudopolinomiales

Un algoritmo para resolver un problema Π es **pseudopolinomial** si la complejidad del mismo es polinomial en función del **valor** (no del tamaño!) de la entrada.

Ejemplo:

- 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.

Teoría de Complejidad

- ¿Qué hacer ante un problema del que no sabemos en que clase está?
- ¿Qué importancia tiene saber si un problema está en P o no, 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?
- ¿Qué hacemos si el problema que tenemos en la práctica sabemos que es NP-completo?