Complejidad - Problemas NP-Completos

Algoritmos y Estructuras de Datos III

La teoría de NP-completitud

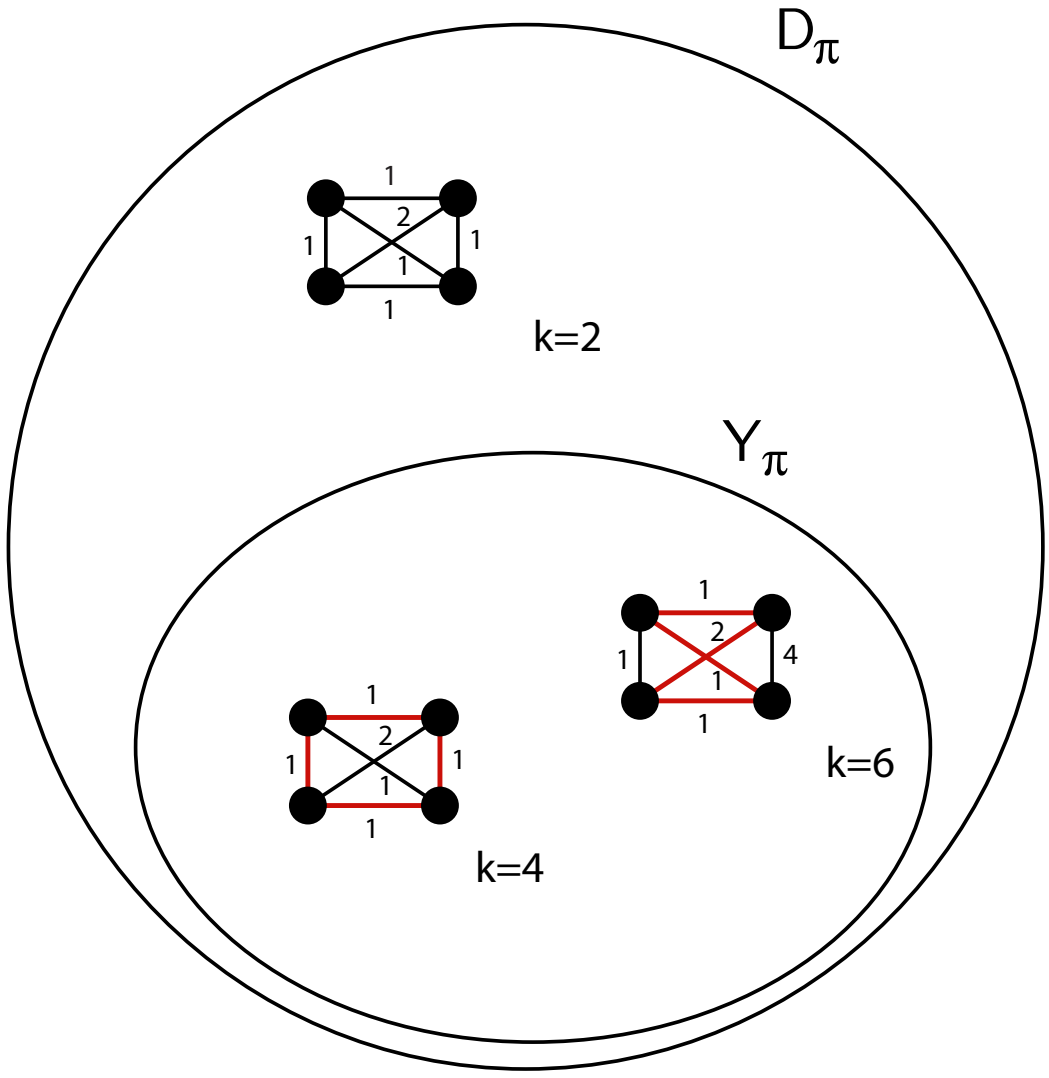
- Un algoritmo eficiente es un algoritmo de complejidad polinomial.
- Un problema está bien resuelto si se conocen algoritmos eficientes para resolverlo.
- El objetivo es clasificar los problemas según su complejidad.
- Se aplica a problemas de decisión, o sea problemas que toman ciertos parámetros y tienen como respuesta SI o NO (aunque es sencillo ver que sus implicancias pueden extenderse a problemas de optimización).

La teoría de NP-completitud

- 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 $c(S) \leq k$.

Distintas versiones de un problema de optimización π

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

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.

Ejemplo: TSP

Dado un grafo completo 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.

Problemas intratables

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

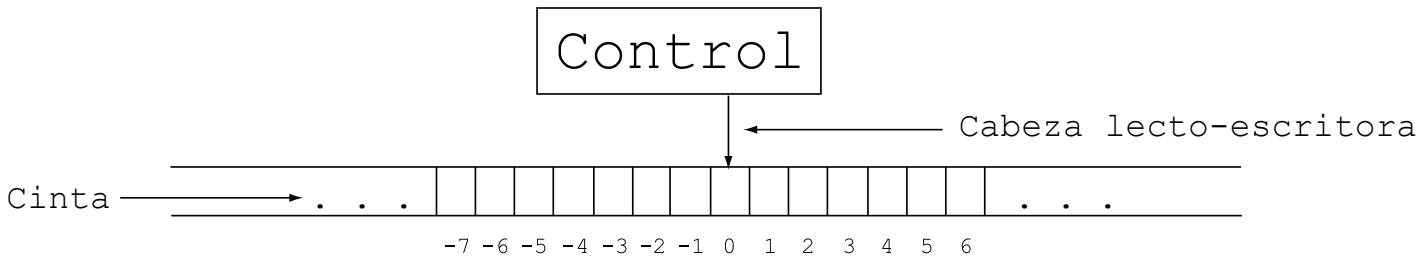
Un problema puede ser intratable por distintos motivos:

- 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.

Modelos de Computadoras: DTM

Máquina de Turing Determinística (DTM)

- Consiste de un control finito, una cabeza lecto-escritora y una cinta 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)

Modelos de Computadoras: DTM

- Sobre la cinta tengo escrito el input que es un string de símbolos de Σ a partir de la celda 1, y el resto de las celdas tiene $*$ (blancos).
- Definimos 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.
- **Para todo par (q_i, s_j), existe exactamente una quíntupla que comienza con ese par (máquina determinística).**

Modelos de Computadoras: DTM

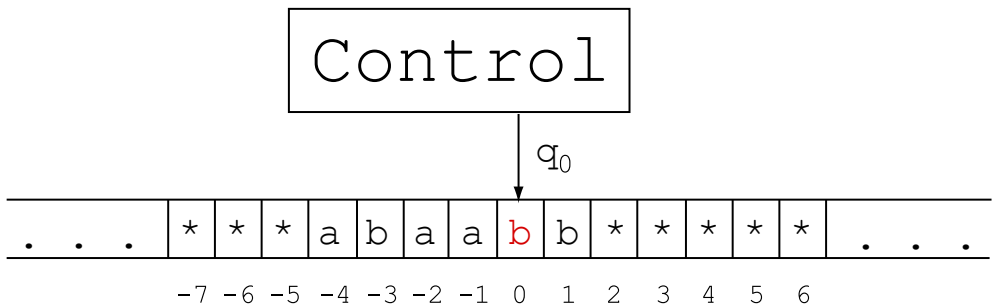
¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

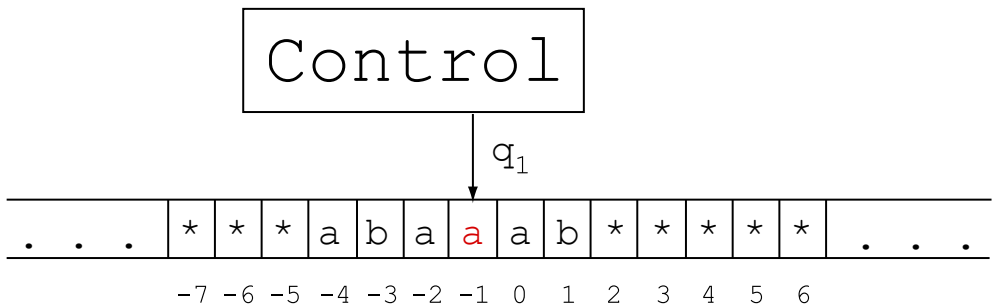


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

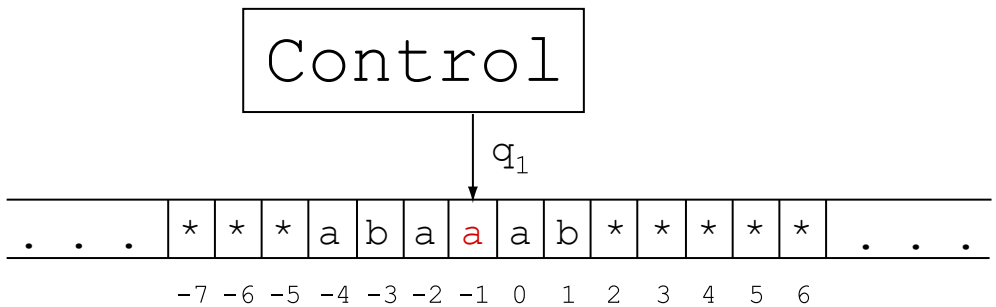


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

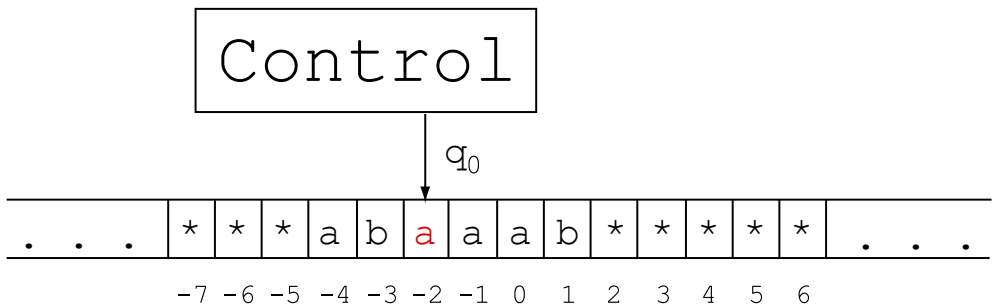


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

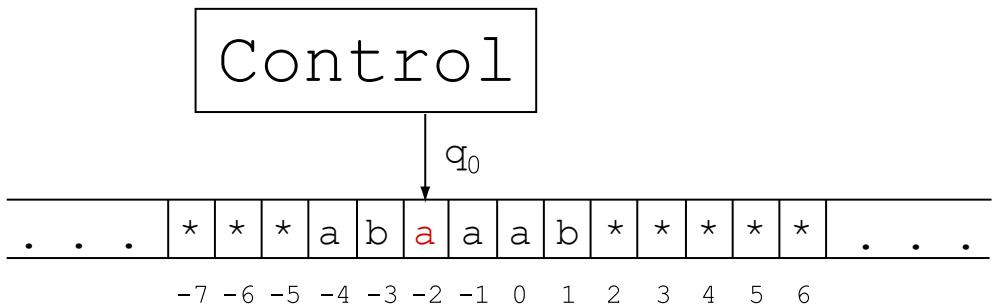


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

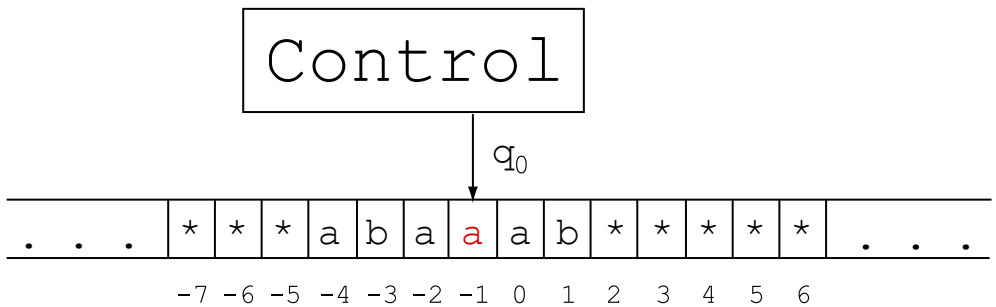


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

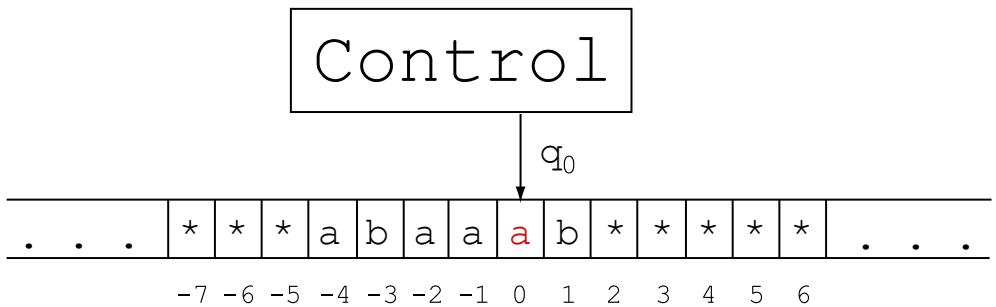


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

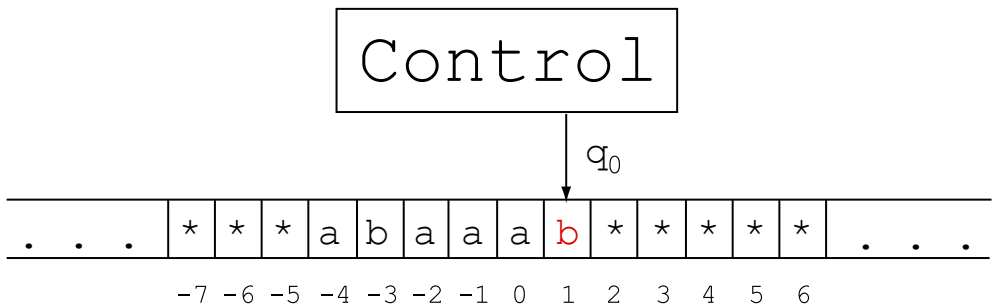


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

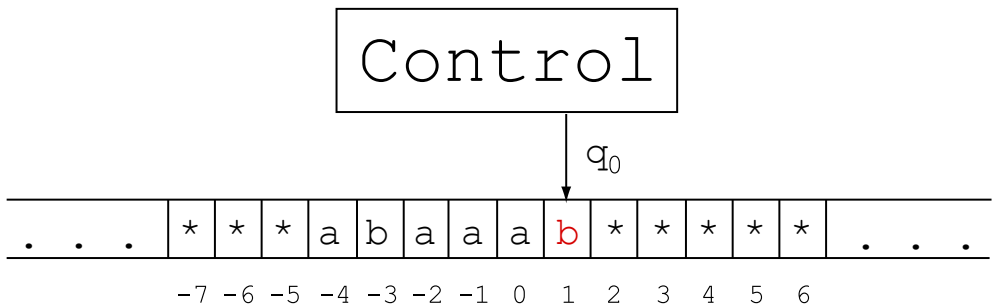


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

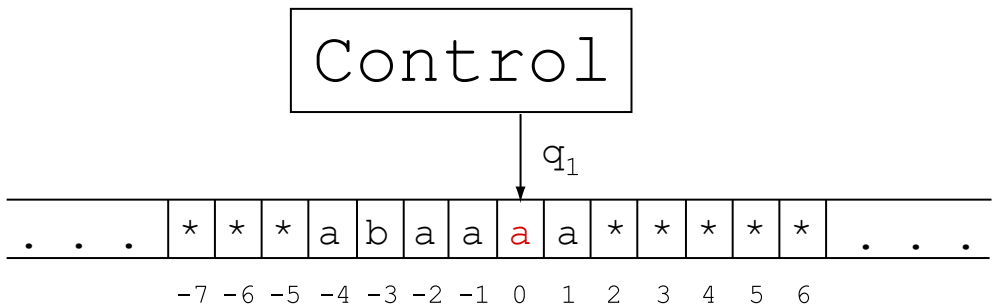


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

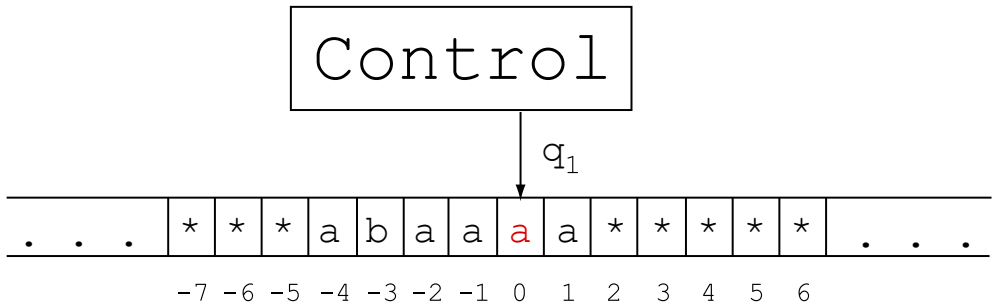


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

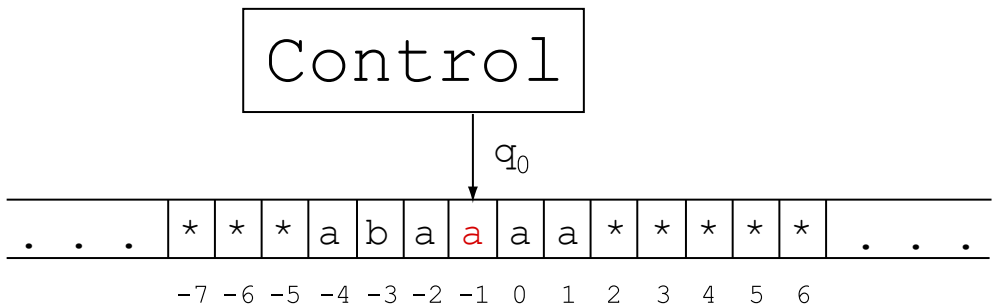


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

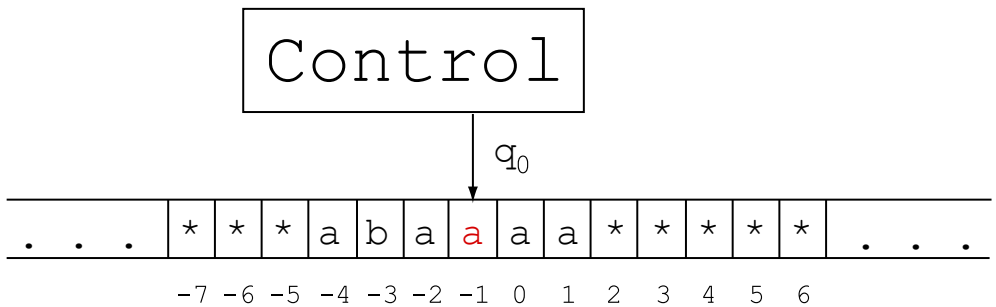


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

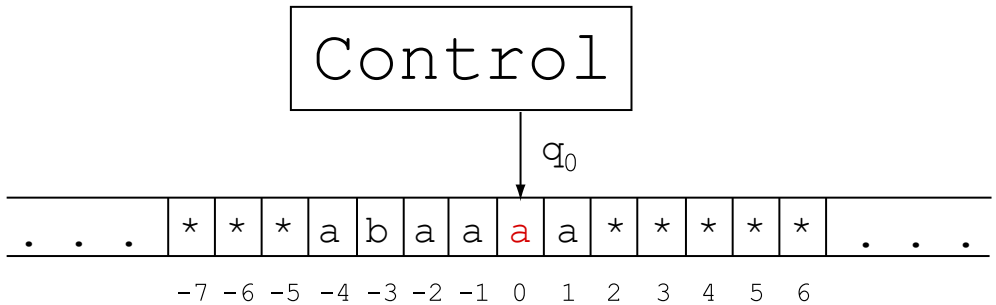


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

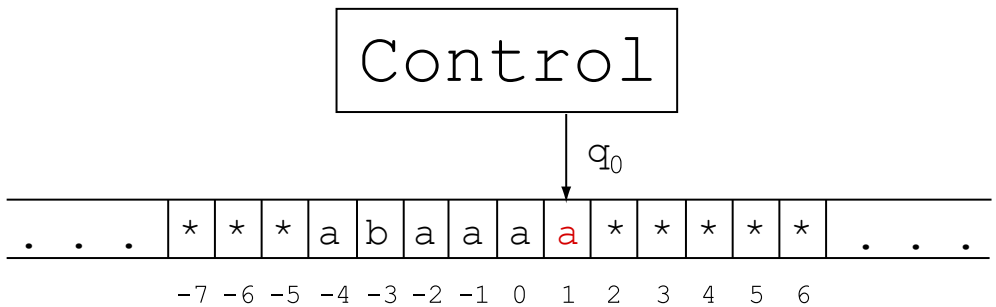


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

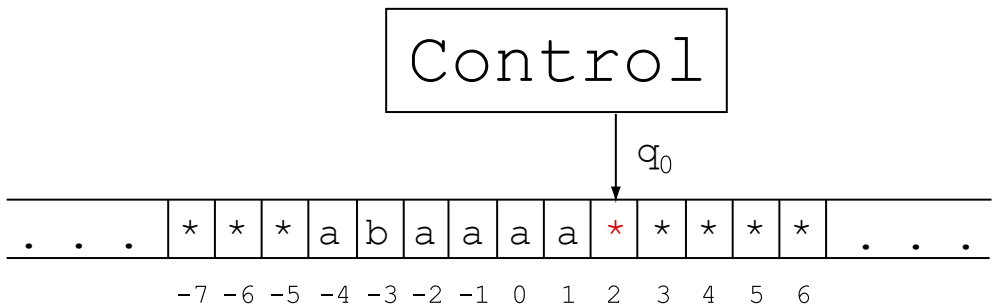


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

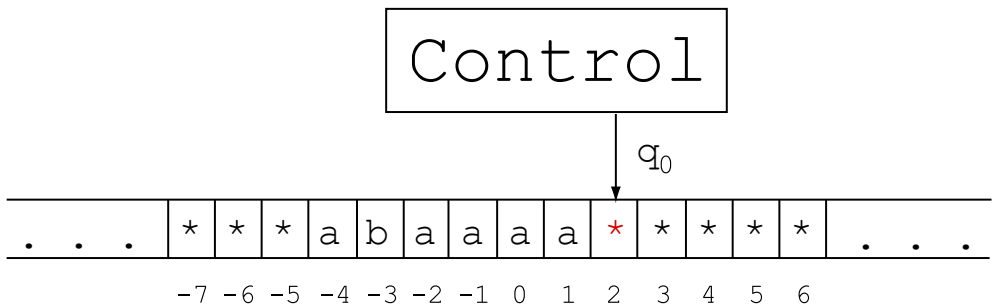


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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

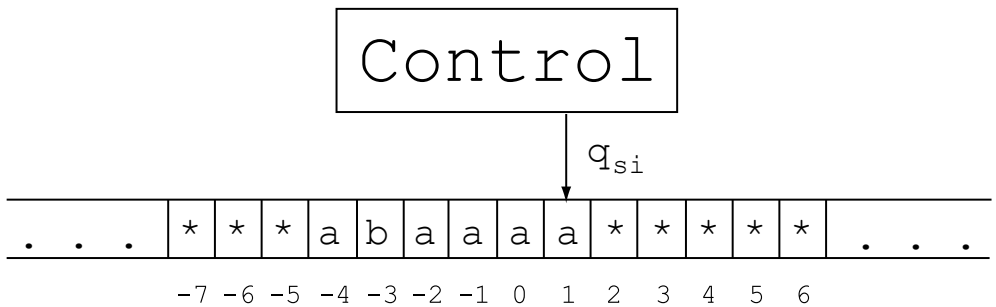


Modelos de Computadoras: DTM

¿Qué significa la quíntupla $(q_i, s_h, q_j, s_k, +1)$? Significa que si estando en el estado q_i la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

Ej:

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



Modelos de Computadoras: DTM

- Una máquina M resuelve el problema π si para toda instancia empieza, termina y contesta bien (o sea, termina en el estado final correcto).

Modelos de Computadoras: DTM

- Una máquina M resuelve el problema π si para toda instancia empieza, termina y contesta bien (o sea, termina en el estado final correcto).
- La complejidad de una DTM 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.
  $T_M(n) = \text{máx}\{m \text{ tq } \exists x \in D_\pi, |x| = n \text{ y } M \text{ con input } x \text{ tarda } m\}$

La clase P

- **Un problema está en P si existe una DTM de complejidad polinomial que lo resuelve.**

  $\mathrm{P} = \{\pi \text{ tq } \exists M \text{ DTM tq } M \text{ resuelve } \pi \text{ y } T_M(n) \in O(p(n)) \text{ para algún polinomio } p\}$

La clase P

- **Un problema está en P si existe una DTM de complejidad polinomial que lo resuelve.**

  P $= \{\pi$ tq $\exists M$ DTM tq M resuelve π y $T_M(n) \in O(p(n))$ para algún polinomio $p\}$

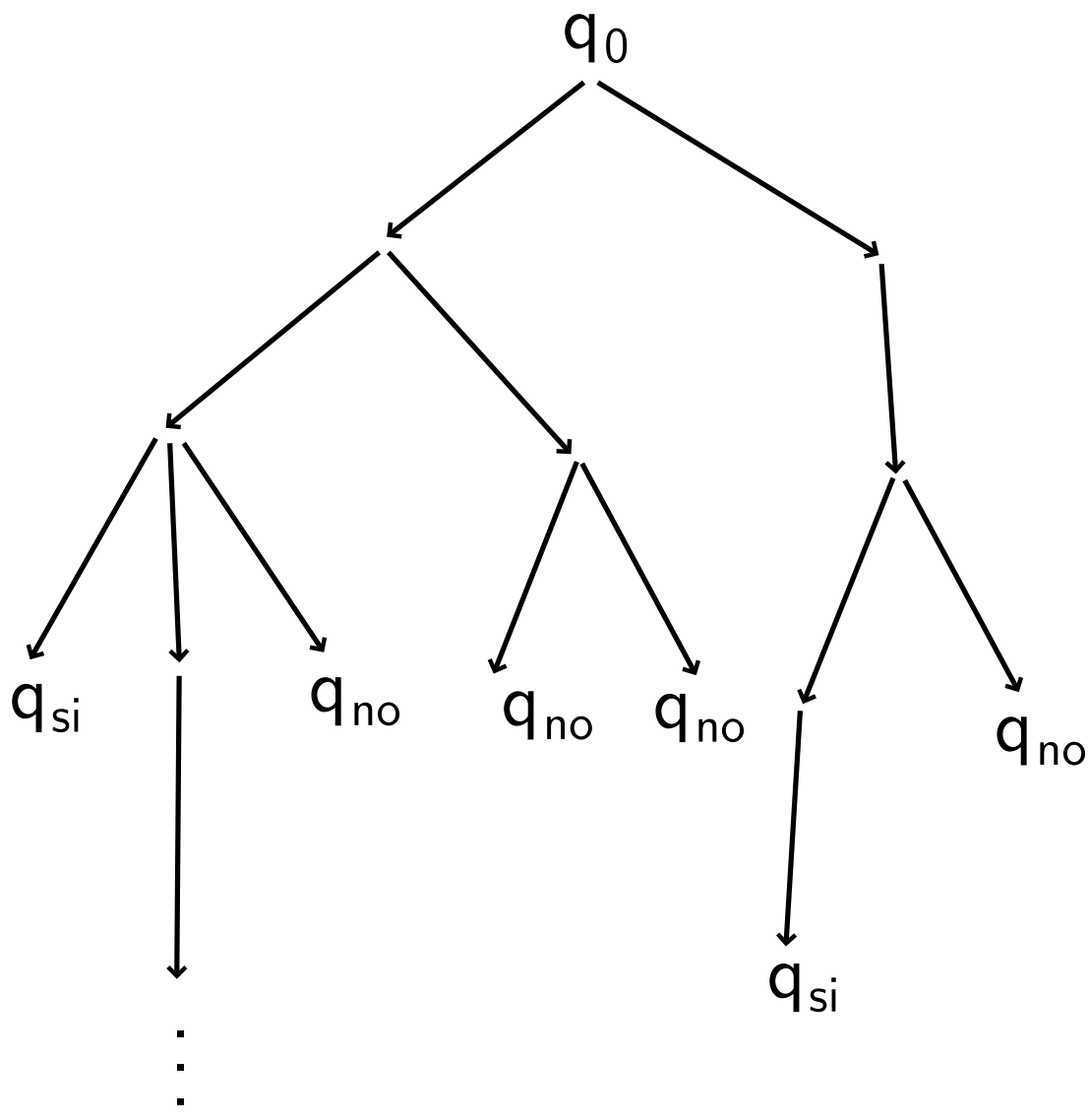
- Existen otros modelos de computadoras determinísticas (máquina de Turing con varias cintas, Random Access Machines, etc.) pero puede probarse que son equivalentes en términos de la polinomialidad de los problemas a la DTM.

Modelos de Computadoras: NDTM

Máquinas de Turing No Determinísticas (NDTM)

- **No se pide unicidad de la quíntupla que comienza con cualquier par (q_i, s_j).**
- En caso de que hubiera más de una quíntupla, la máquina se replica continuando cada una por una rama distinta.
- Decimos que una NDTM resuelve el problema π si 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}.

Modelos de Computadoras: NDTM



Modelos de Computadoras: NDTM

- Una NDTM 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.

Modelos de Computadoras: NDTM

- Una NDTM 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.
- **Un problema $\pi \in$ NP si existe una NDTM polinomial que resuelve π.**

Modelos de Computadoras: NDTM

- Una NDTM 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.
- **Un problema $\pi \in$ NP si existe una NDTM polinomial que resuelve π.**
- **NP es "No-determinístico Polinomial", NO ES "No Polinomial"!!!**

NP-completitud

Equivalentemente, un problema de decisión pertenece a la clase **NP** si dada una instancia de **SI** y evidencia de la misma, puede ser verificada en tiempo polinomial. Esa evidencia a veces se llama certificado, y tiene que tener tamaño polinomial en el tamaño de la entrada.

¿Cuál sería un certificado para TSP?

NP-completitud

Equivalentemente, un problema de decisión pertenece a la clase **NP** si dada una instancia de **SI** y evidencia de la misma, puede ser verificada en tiempo polinomial. Esa evidencia a veces se llama certificado, y tiene que tener tamaño polinomial en el tamaño de la entrada.

¿Cuál sería un certificado para TSP?

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.

NP-completitud

- Claramente, $\mathrm{P} \subseteq \mathrm{NP}$.

NP-completitud

- Claramente, $\mathrm{P} \subseteq \mathrm{NP}$.
- **Conjetura: $\mathbf{P} \neq \mathbf{NP}$.**

  Todavía no se demostró que exista un problema en $\mathrm{NP} \setminus \mathrm{P}$. Mientras tanto, se estudian clases de complejidad "relativa", es decir, que establecen orden de dificultad entre problemas.

NP-completitud

- **Reducción polinomial:** Sean π y π' dos problemas de decisión. Decimos que $f : D_{\pi'} \to D_{\pi}$ es una reducción polinomial de π' en π si f se computa en tiempo polinomial y para todo $d \in D_{\pi'}$, $d \in Y_{\pi'} \Leftrightarrow f(d) \in Y_{\pi}$. Notación: $\pi' \preccurlyeq \pi$.

NP-completitud

- **Reducción polinomial:** Sean π y π' dos problemas de decisión. Decimos que $f : D_{\pi'} \to D_{\pi}$ es una reducción polinomial de π' en π si f se computa en tiempo polinomial y para todo $d \in D_{\pi'}$, $d \in Y_{\pi'} \Leftrightarrow f(d) \in Y_{\pi}$. Notación: $\pi' \preccurlyeq \pi$.
- Notemos que si $\pi'' \preccurlyeq \pi'$ y $\pi' \preccurlyeq \pi$ entonces $\pi'' \preccurlyeq \pi$, ya que la composición de dos reducciones polinomiales es una reducción polinomial.

NP-completitud

- **Reducción polinomial:** Sean π y π' dos problemas de decisión. Decimos que $f : D_{\pi'} \to D_{\pi}$ es una reducción polinomial de π' en π si f se computa en tiempo polinomial y para todo $d \in D_{\pi'}$, $d \in Y_{\pi'} \Leftrightarrow f(d) \in Y_{\pi}$. Notación: $\pi' \preccurlyeq \pi$.
- Notemos que si $\pi'' \preccurlyeq \pi'$ y $\pi' \preccurlyeq \pi$ entonces $\pi'' \preccurlyeq \pi$, ya que la composición de dos reducciones polinomiales es una reducción polinomial.
- Un problema π es **NP-completo** si:

NP-completitud

- **Reducción polinomial:** Sean π y π' dos problemas de decisión. Decimos que $f : D_{\pi'} \to D_{\pi}$ es una reducción polinomial de π' en π si f se computa en tiempo polinomial y para todo $d \in D_{\pi'}$, $d \in Y_{\pi'} \Leftrightarrow f(d) \in Y_{\pi}$. Notación: $\pi' \preccurlyeq \pi$.
- Notemos que si $\pi'' \preccurlyeq \pi'$ y $\pi' \preccurlyeq \pi$ entonces $\pi'' \preccurlyeq \pi$, ya que la composición de dos reducciones polinomiales es una reducción polinomial.
- Un problema π es **NP-completo** si:
  1. $\pi \in$ NP.

NP-completitud

- **Reducción polinomial:** Sean π y π' dos problemas de decisión. Decimos que $f : D_{\pi'} \to D_{\pi}$ es una reducción polinomial de π' en π si f se computa en tiempo polinomial y para todo $d \in D_{\pi'}$, $d \in Y_{\pi'} \Leftrightarrow f(d) \in Y_{\pi}$. Notación: $\pi' \preccurlyeq \pi$.
- Notemos que si $\pi'' \preccurlyeq \pi'$ y $\pi' \preccurlyeq \pi$ entonces $\pi'' \preccurlyeq \pi$, ya que la composición de dos reducciones polinomiales es una reducción polinomial.
- Un problema π es **NP-completo** si:
  1. $\pi \in$ NP.
  2. Para todo $\pi' \in$ NP, $\pi' \preccurlyeq \pi$.

NP-completitud

- **Reducción polinomial:** Sean π y π' dos problemas de decisión. Decimos que $f : D_{\pi'} \to D_{\pi}$ es una reducción polinomial de π' en π si f se computa en tiempo polinomial y para todo $d \in D_{\pi'}$, $d \in Y_{\pi'} \Leftrightarrow f(d) \in Y_{\pi}$. Notación: $\pi' \preccurlyeq \pi$.
- Notemos que si $\pi'' \preccurlyeq \pi'$ y $\pi' \preccurlyeq \pi$ entonces $\pi'' \preccurlyeq \pi$, ya que la composición de dos reducciones polinomiales es una reducción polinomial.
- Un problema π es **NP-completo** si:
  1. $\pi \in$ NP.
  2. Para todo $\pi' \in$ NP, $\pi' \preccurlyeq \pi$.
- Si un problema π verifica la condición 2., π es NP-Hard (es al menos tan "difícil" como todos los problemas de NP).

¿P $\neq$ NP? La pregunta del millón...

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

¿P $\neq$ NP? La pregunta del millón...

- **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, $\pi' \preccurlyeq \pi$.

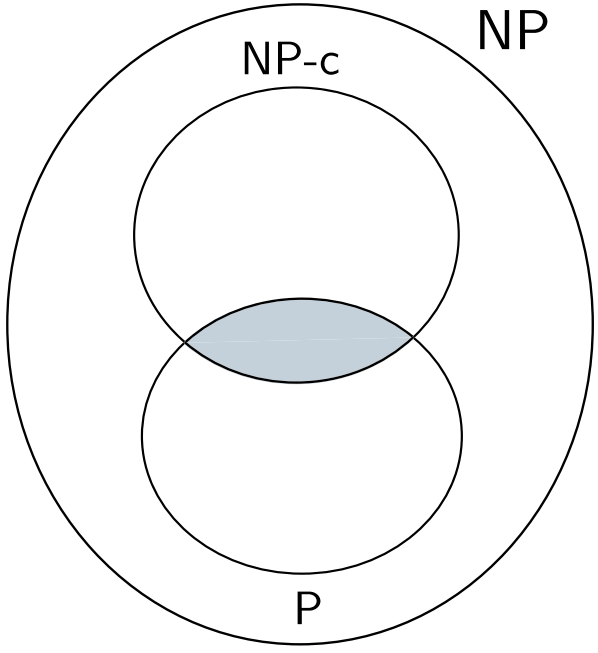
¿P $\neq$ NP? La pregunta del millón...

- **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, $\pi' \preccurlyeq \pi$.
  - Sea $\pi' \in$ NP. Apliquemos 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, es fácil ver que lo que se obtiene es un algoritmo polinomial que resuelve π'.

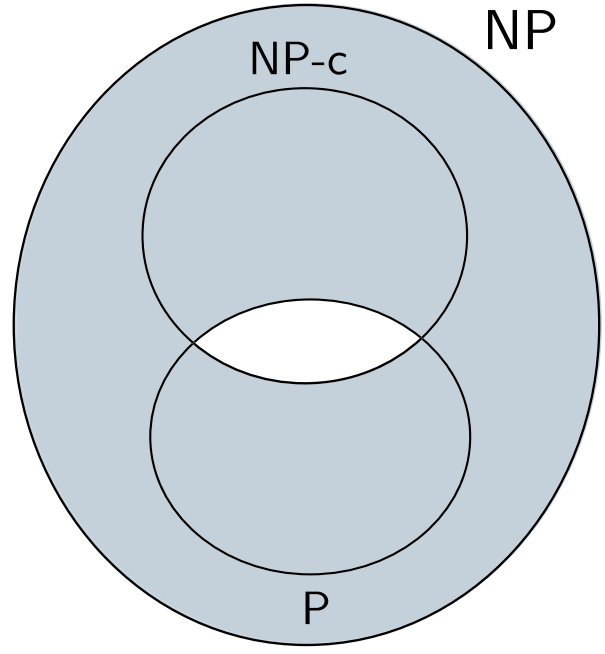
¿P $\neq$ NP? La pregunta del millón...

- **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, $\pi' \preccurlyeq \pi$.
  - Sea $\pi' \in$ NP. Apliquemos 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, es fácil ver que lo que se obtiene es un algoritmo polinomial que resuelve π'.
- Hasta el momento no se conoce ningún problema en NP-c $\cap$ P, así como tampoco se ha demostrado que un problema esté en NP $\setminus$ P. En ese caso, obviamente, se probaría que P $\neq$ NP.

Esquema de clases



si P=NP...



si P≠NP...

¿Cómo se prueba que un problema es 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 de Cook (1971): SAT es NP-completo.

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 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?

A partir del Teorema de Cook, la técnica standard para probar que un problema π es NP-completo aprovecha la transitividad de $\preccurlyeq$, y consiste en lo siguiente:

1. Mostrar que π está en NP.
2. Elegir un problema π' apropiado que se sepa que es NP-completo.
3. Construir una reducción polinomial f de π' en π.

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

A partir del Teorema de Cook, la técnica standard para probar que un problema π es NP-completo aprovecha la transitividad de $\preccurlyeq$, y consiste en lo siguiente:

1. Mostrar que π está en NP.
2. Elegir un problema π' apropiado que se sepa que es NP-completo.
3. Construir una reducción polinomial f de π' en π.

La segunda condición en la definición de problema NP-completo sale usando la transitividad: sea π'' un problema cualquiera de NP. Como π' es NP-completo, $\pi'' \preccurlyeq \pi'$. Como probamos que $\pi' \preccurlyeq \pi$, resulta $\pi'' \preccurlyeq \pi$.

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, queda como está.

Reducción de SAT a 3-SAT

- Si C_i tiene tres literales, queda como está.
- C_i tiene menos de tres literales, agregamos nuevas variables como en el ejemplo:

$$(x_1 \vee \neg x_2) \rightarrow (x_1 \vee \neg x_2 \vee y) \wedge (x_1 \vee \neg x_2 \vee \neg y)$$

Reducción de SAT a 3-SAT

- Si C_i tiene tres literales, queda como está.
- C_i tiene menos de tres literales, agregamos nuevas variables como en el ejemplo:

$$(x_1 \vee \neg x_2) \rightarrow (x_1 \vee \neg x_2 \vee y) \wedge (x_1 \vee \neg x_2 \vee \neg y)$$

- Si C_i tiene cuatro o más literales, agregamos nuevas variables como en el ejemplo:

$$(x_1 \vee \neg x_2 \vee x_3 \vee x_4 \vee \neg x_5) \rightarrow$$

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

Reducción de SAT a 3-SAT

- Si C_i tiene tres literales, queda como está.
- C_i tiene menos de tres literales, agregamos nuevas variables como en el ejemplo:

$$(x_1 \vee \neg x_2) \rightarrow (x_1 \vee \neg x_2 \vee y) \wedge (x_1 \vee \neg x_2 \vee \neg y)$$

- Si C_i tiene cuatro o más literales, agregamos nuevas variables como en el ejemplo:

$$(x_1 \vee \neg x_2 \vee x_3 \vee x_4 \vee \neg x_5) \rightarrow$$

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

Queda como ejercicio escribir formalmente la reducción y demostrar que es una reducción polinomial de SAT a 3-SAT.

La clase NP-Difícil (NP-Hard)

Un problema de decisión π es NP-difícil si todo otro problema de NP se puede transformar polinomialmente a π.

(En la práctica esta definición a veces se usa por un abuso de lenguaje también para problemas que no son de decisión y cuya versión de decisión es NP-completa.)

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.

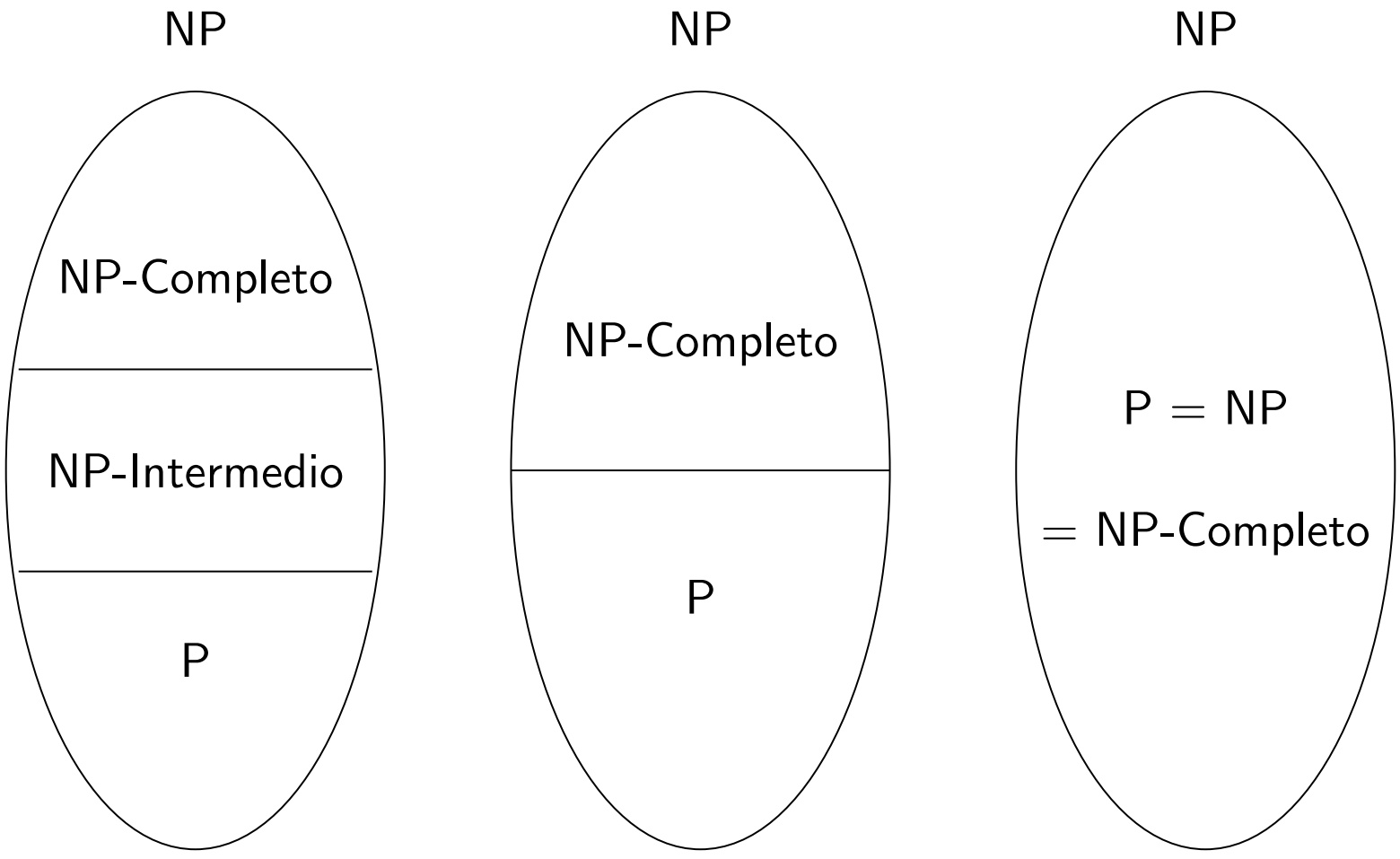
- La clase de los problemas polinomiales (P), está contenida también en Co-NP.

Problemas abiertos de Teoría de Complejidad

Con estas nuevas definiciones tenemos los siguientes problemas abiertos:

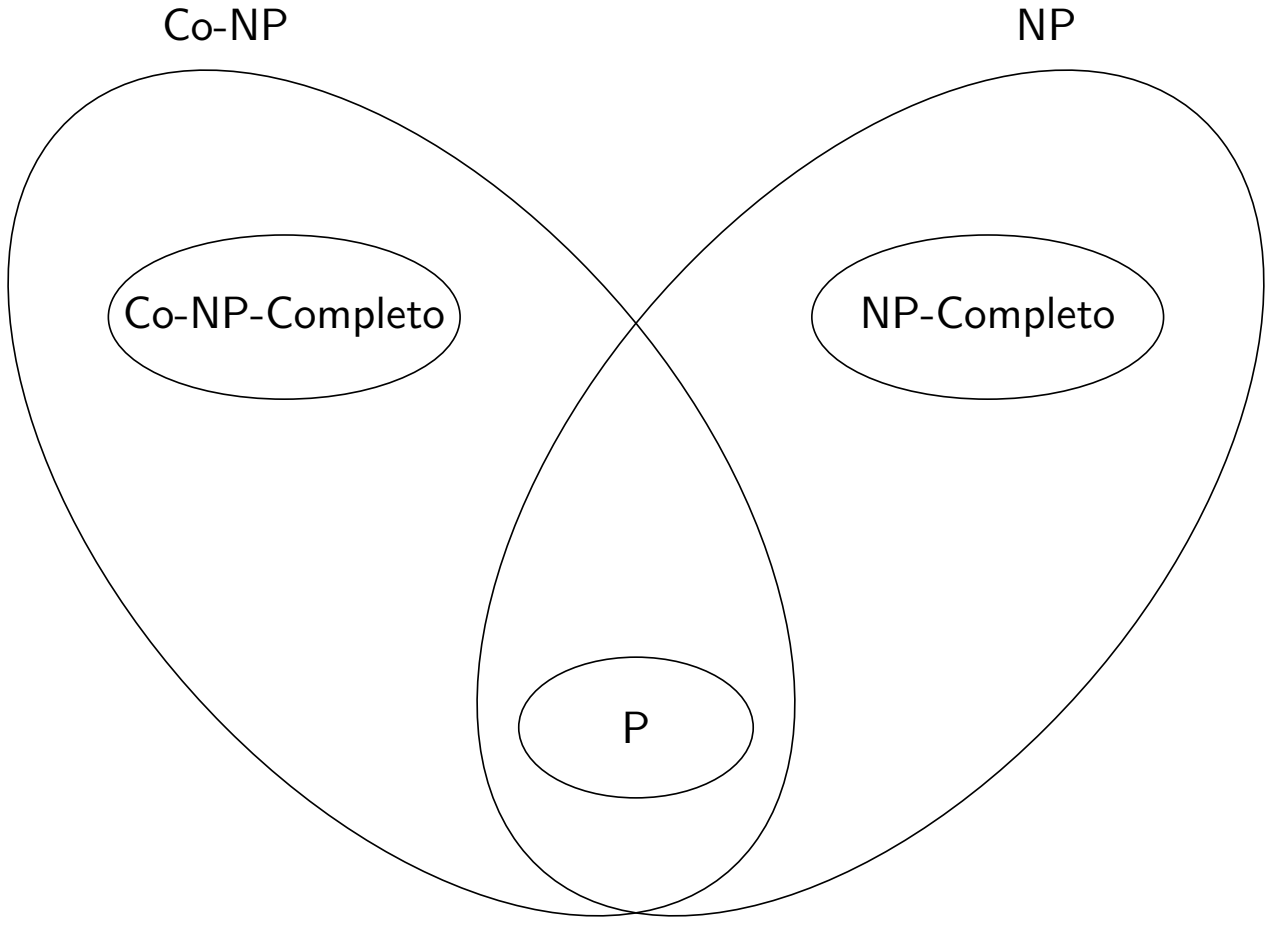
- ¿Es P=NP?
- ¿Es Co-NP=NP?
- ¿Es P=Co-NP $\cap$ NP?

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 el tamaño!) de la entrada.

Ejemplos:

- Primalidad.
- 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?