Una (muy) breve introducción a la teoría de la computación

Marcelo Arenas

Ciencia de la computación

¿Cuál es el objeto de estudio de esta ciencia?

- Dijkstra: "Computer science is no more about computers than astronomy is about telescopes"

Ciencia de la computación

¿Cuál es el objeto de estudio de esta ciencia?

- Dijkstra: "Computer science is no more about computers than astronomy is about telescopes"

¿Cuál es su objeto de estudio entonces?

Ciencia de la computación: Objeto de estudio

Pregunta a resolver: ¿Cuál es la complejidad de un problema?

- ¿Puede ser resuelto eficientemente? ¿Cuán rápido?

Ciencia de la computación: Objeto de estudio

Pregunta a resolver: ¿Cuál es la complejidad de un problema?

- ¿Puede ser resuelto eficientemente? ¿Cuán rápido?

En general:

- ¿Existen problemas difíciles?
- ¿Existen problemas que no pueden ser resueltos?

Complejidad computacional

Objetivo: Medir la complejidad computacional de un problema.

Vale decir: Medir la cantidad de recursos computacionales necesarios para solucionar un problema.

- Tiempo
- Espacio
- ...

Complejidad computacional

Objetivo: Medir la complejidad computacional de un problema.

Vale decir: Medir la cantidad de recursos computacionales necesarios para solucionar un problema.

- Tiempo
- Espacio
- ...

Para hacer esto primero tenemos que introducir la noción de problema.

Problemas de decisión

Alfabeto A: Conjunto finito de símbolos.

- Ejemplo: $A = \{0, 1\}$

Palabra w: Secuencia finita de símbolos de A.

- Ejemplo: $w = 01101$

A^*: Conjunto de todas las palabras construidas con símbolos de A.

Problemas de decisión

Lenguaje L: Conjunto de palabras ($L \subseteq A^*$)

- Ejemplo: $L = \{w \in \{0,1\}^* \mid$ número de 0s y 1s en w es el mismo$\}$

Problema de decisión asociado a un lenguaje L: Dado $w \in A^*$, decidir si $w \in L$.

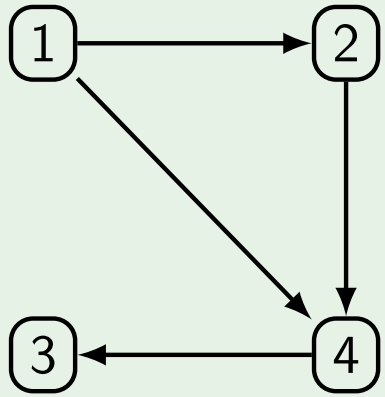
Problemas de decisión: Ejemplo

Grafo: $G = (N, E)$, donde N es un conjunto finito de nodos y $E \subseteq N \times N$.

Ejemplo

$G = (N, E)$ con $N = \{1, 2, 3, 4\}$ y $E = \{(1, 2), (1, 4), (2, 4), (4, 3)\}$

G es representado como:



Problemas de decisión: Ejemplo

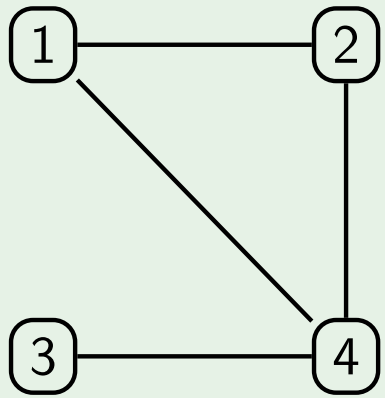
Grafo $G = (N, E)$ es no-dirigido si

- Para cada $(a, b) \in E$, se tiene que $(b, a) \in E$

Ejemplo

$G = (N, E)$ con $N = \{1, 2, 3, 4\}$ y $E = \{(1, 2), (2, 1), (1, 4), (4, 1), (2, 4), (4, 2), (4, 3), (3, 4)\}$

G es representado como:



Problemas de decisión: Ejemplo

Problema: Un grafo no-dirigido $G = (N, E)$ es 3-coloreable si existe $f : N \to \{\text{blanco}, \text{rojo}, \text{verde}\}$ tal que

- Para cada $(a, b) \in E$, se tiene que $f(a) \neq f(b)$

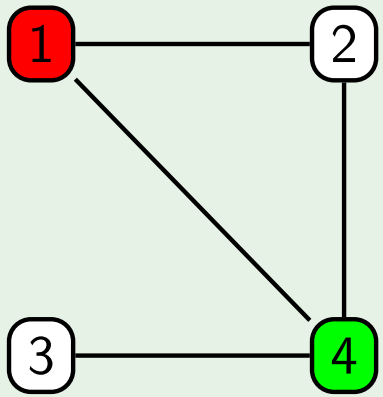
Problemas de decisión: Ejemplo

Problema: Un grafo no-dirigido $G = (N, E)$ es 3-coloreable si existe $f : N \to \{\text{blanco}, \text{rojo}, \text{verde}\}$ tal que

- Para cada $(a, b) \in E$, se tiene que $f(a) \neq f(b)$

Ejemplo

3-coloración de un grafo:



Problemas de decisión: Ejemplo

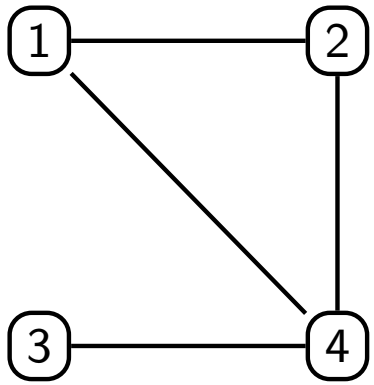
Podemos ver el problema de 3-coloración como un problema de decisión.

Problemas de decisión: Ejemplo

Podemos ver el problema de 3-coloración como un problema de decisión.

Dado $A = \{0, 1\}$, representamos un grafo con n nodos como una palabra con n^2 símbolos.

- La palabra 0101100100011110 representa al grafo

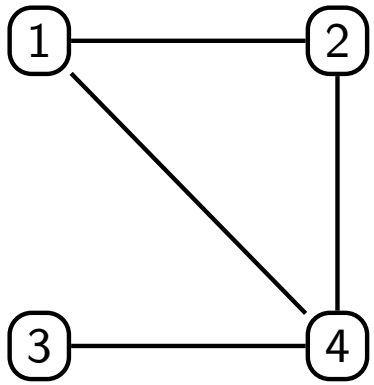


Problemas de decisión: Ejemplo

Podemos ver el problema de 3-coloración como un problema de decisión.

Dado $A = \{0, 1\}$, representamos un grafo con n nodos como una palabra con n^2 símbolos.

- La palabra 0101100100011110 representa al grafo



Problema de 3-coloración

3-COL $= \{w \in \{0,1\}^* \mid w$ representa un grafo G tal que G es 3-coloreable$\}$

Complejidad de un problema de decisión

La complejidad de un lenguaje L es la complejidad del problema de decisión asociado a L.

Complejidad de un problema de decisión

La complejidad de un lenguaje L es la complejidad del problema de decisión asociado a L.

¿Cuándo decimos que L puede ser solucionado eficientemente?

- Cuando existe un algoritmo eficiente que decide L.

Complejidad de un problema de decisión

La complejidad de un lenguaje L es la complejidad del problema de decisión asociado a L.

¿Cuándo decimos que L puede ser solucionado eficientemente?

- Cuando existe un algoritmo eficiente que decide L.

¿Cuándo decimos que L es un problema difícil?

- Cuando **no existe** un algoritmo eficiente que decide L.

Máquinas de Turing

¿Cómo podemos demostrar que un problema es difícil?

- Para hacer esto, primero tenemos que formalizar la noción de algoritmo.

Máquinas de Turing

¿Cómo podemos demostrar que un problema es difícil?

- Para hacer esto, primero tenemos que formalizar la noción de algoritmo.

¿Qué es un algoritmo? ¿Podemos formalizar este concepto?

- Máquinas de Turing: Intento por formalizar este concepto

Máquinas de Turing

¿Cómo podemos demostrar que un problema es difícil?

- Para hacer esto, primero tenemos que formalizar la noción de algoritmo.

¿Qué es un algoritmo? ¿Podemos formalizar este concepto?

- Máquinas de Turing: Intento por formalizar este concepto

¿Podemos demostrar que las máquinas de Turing capturan la noción de algoritmo?

- No, el concepto de algoritmo es intuitivo

Máquinas de Turing

¿Por qué creemos que las máquinas de Turing son una buena formalización del concepto de algoritmo?

Máquinas de Turing

¿Por qué creemos que las máquinas de Turing son una buena formalización del concepto de algoritmo?

- Porque cada programa de una máquina de Turing puede ser implementado

Máquinas de Turing

¿Por qué creemos que las máquinas de Turing son una buena formalización del concepto de algoritmo?

- Porque cada programa de una máquina de Turing puede ser implementado
- Porque todos los algoritmos conocidos han podido ser implementados en máquinas de Turing

Máquinas de Turing

¿Por qué creemos que las máquinas de Turing son una buena formalización del concepto de algoritmo?

- Porque cada programa de una máquina de Turing puede ser implementado
- Porque todos los algoritmos conocidos han podido ser implementados en máquinas de Turing
- Porque todos los otros intentos por formalizar este concepto fueron reducidos a las máquinas de Turing
  - Los mejores intentos resultaron ser equivalentes a las máquinas de Turing
  - Todos los intentos "razonables" fueron reducidos eficientemente

Máquinas de Turing

¿Por qué creemos que las máquinas de Turing son una buena formalización del concepto de algoritmo?

- Porque cada programa de una máquina de Turing puede ser implementado
- Porque todos los algoritmos conocidos han podido ser implementados en máquinas de Turing
- Porque todos los otros intentos por formalizar este concepto fueron reducidos a las máquinas de Turing
  - Los mejores intentos resultaron ser equivalentes a las máquinas de Turing
  - Todos los intentos "razonables" fueron reducidos eficientemente
- Tesis de Church: **Algoritmo = Máquina de Turing**

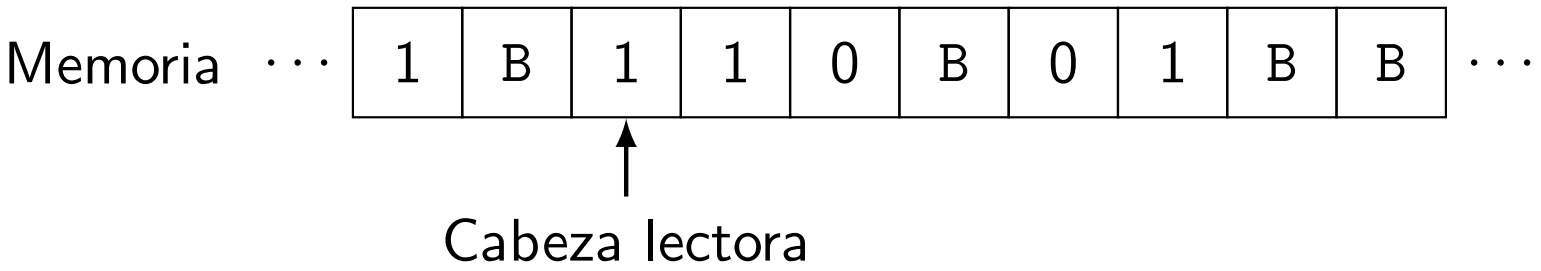
Máquinas de Turing: Componentes

Dado: Alfabeto A que no contiene símbolo especial $\mathtt{B}$.

Componentes:

- Memoria: Arreglo infinito (en ambas direcciones) que en cada posición almacena un símbolo de $A \cup \{\mathtt{B}\}$
- Control: Conjunto finitos de estados, una cabeza lectora, un estado inicial y un conjunto de estados finales
- Programa: Conjunto de instrucciones

Máquinas de Turing: Componentes



Estados q_0 q_1 q_2 q_3 $\cdots$ q_k

Programa $\cdots$

Máquinas de Turing: Funcionamiento

Inicialmente:

- La máquina recibe como entrada una palabra w.
- Coloca esta palabra en alguna parte del arreglo. En las posiciones restantes el arreglo contiene el símbolo blanco `B`.
- La cabeza lectora se coloca en la primera posición de w y la máquina queda en el estado inicial q_0.

Máquinas de Turing: Funcionamiento

En cada paso:

- La máquina lee el símbolo a en la celda apuntada por la cabeza lectora y determina en qué estado q se encuentra.
- Busca en el programa una instrucción para (q, a). Si esta instrucción no existe, entonces la máquina se detiene.
- Si la instrucción existe, entonces la ejecuta: Escribe un símbolo en la posición apuntada por la cabeza lectora, pasa a un nuevo estado y mueve la cabeza lectora según lo que indica la instrucción.

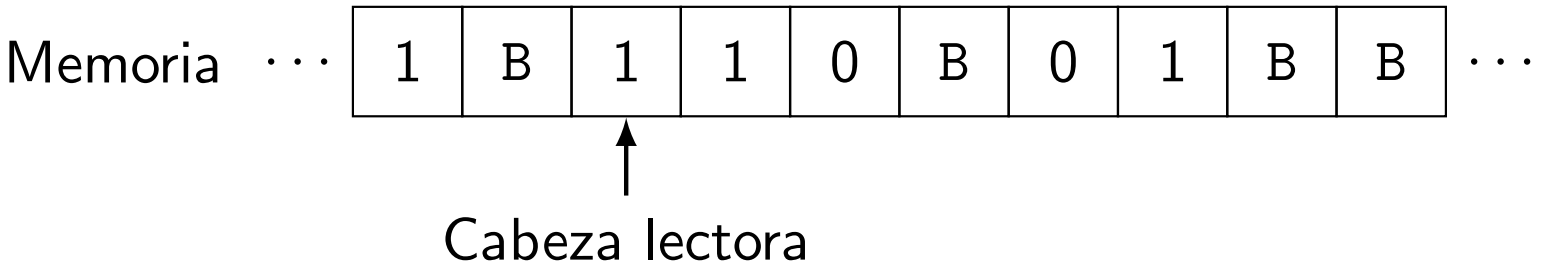
Máquinas de Turing: Funcionamiento

En cada paso:

- La máquina lee el símbolo a en la celda apuntada por la cabeza lectora y determina en qué estado q se encuentra.
- Busca en el programa una instrucción para (q, a). Si esta instrucción no existe, entonces la máquina se detiene.
- Si la instrucción existe, entonces la ejecuta: Escribe un símbolo en la posición apuntada por la cabeza lectora, pasa a un nuevo estado y mueve la cabeza lectora según lo que indica la instrucción.

Si la máquina se detiene en un estado final, entonces acepta w.

Máquinas de Turing: Componentes



Estados $q_0 \quad q_1 \quad q_2 \quad q_3 \quad \cdots \quad q_k$

Programa

$$\begin{array}{llll} (q_3, & \mathtt{B}) & \mapsto & (q_3, \quad \mathtt{B}, \quad \leftarrow) \\ (q_2, & 1) & \mapsto & (q_3, \quad 0, \quad \rightarrow) \\ (q_0, & 0) & \mapsto & (q_4, \quad \mathtt{B}, \quad \square) \\ & \cdots & & \end{array}$$

Máquinas de Turing: Formalización

Definición

Máquina de Turing (determinista): (Q, A, q_0, δ, F)

- Q *es un conjunto finito de estados*
- A *es un alfabeto tal que* $\mathtt{B} \notin A$
- q_0 *es el estado inicial* $(q_0 \in Q)$
- F *es un conjunto de estados finales* $(F \subseteq Q)$
- δ *es una función parcial:*

$$\delta \quad : \quad Q \times (A \cup \{\mathtt{B}\}) \to Q \times (A \cup \{\mathtt{B}\}) \times \{\leftarrow, \square, \rightarrow\}.$$

δ *es llamada función de transición*

Máquinas de Turing: Ejemplo

Sea $L = \{w \in \{0,1\}^* \mid \text{el número de 0s en } w \text{ es par}\}$

- Queremos construir una MT $M = (Q, A, q_0, \delta, F)$ que decida L

Máquinas de Turing: Ejemplo

Sea $L = \{w \in \{0,1\}^* \mid \text{el número de 0s en } w \text{ es par}\}$

- Queremos construir una MT $M = (Q, A, q_0, \delta, F)$ que decida L

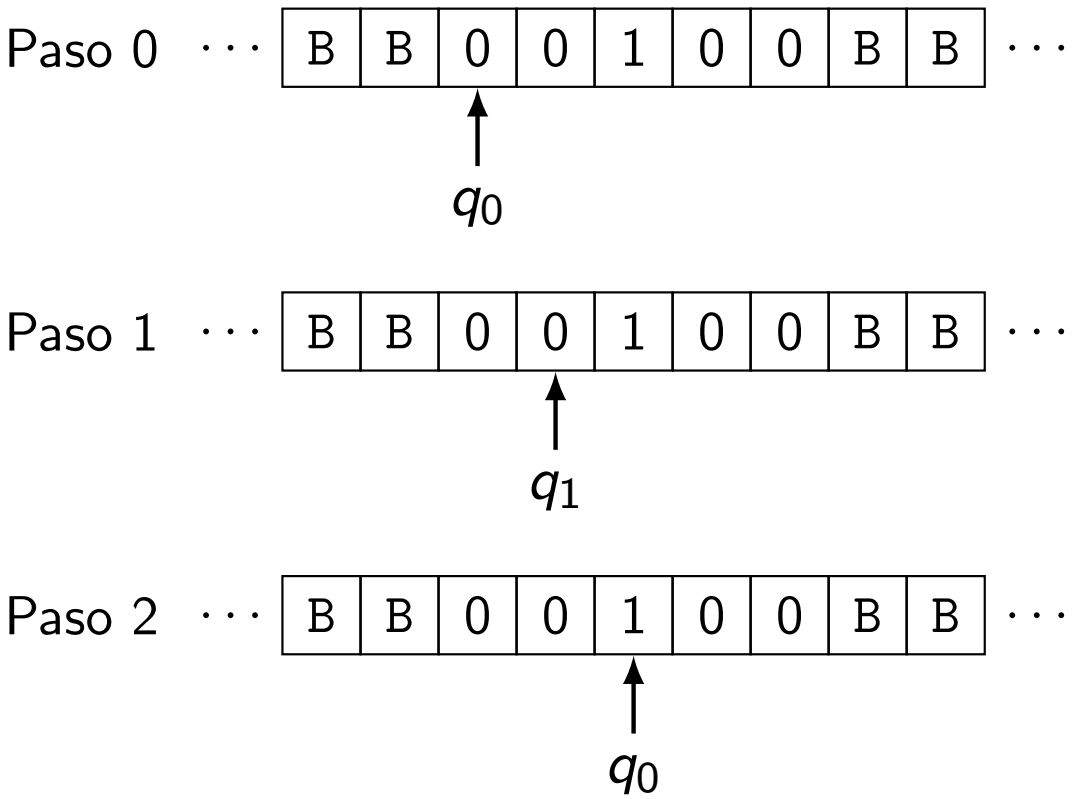
Se tiene que:

- $A = \{0, 1\}$
- $Q = \{q_0, q_1, q_f\}$
- $F = \{q_f\}$
- δ es definida como:

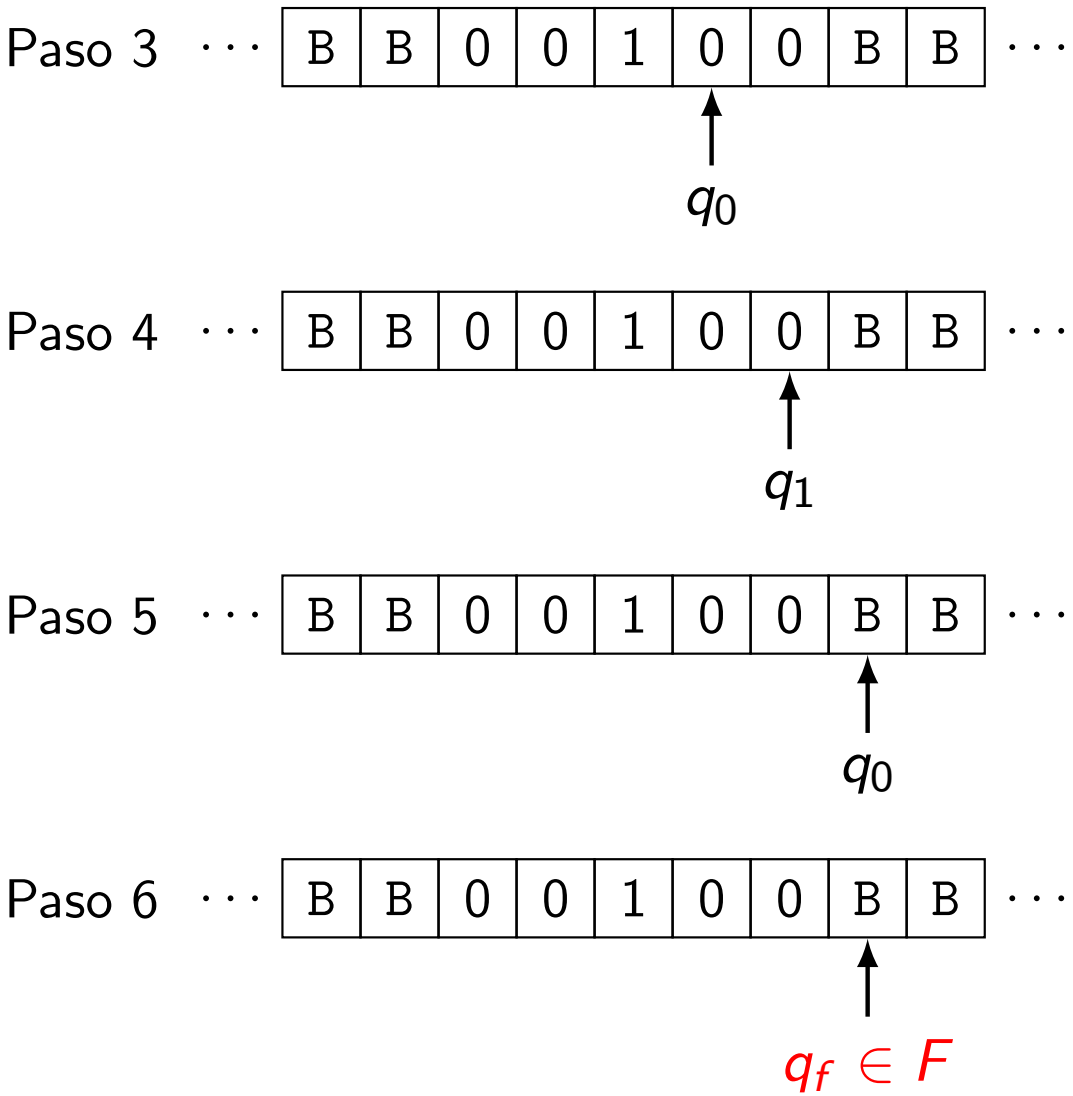
$$\begin{aligned}
\delta(q_0, 0) &= (q_1, 0, \rightarrow) \\
\delta(q_0, 1) &= (q_0, 1, \rightarrow) \\
\delta(q_0, \mathtt{B}) &= (q_f, \mathtt{B}, \square) \\
\delta(q_1, 0) &= (q_0, 0, \rightarrow) \\
\delta(q_1, 1) &= (q_1, 1, \rightarrow)
\end{aligned}$$

Máquinas de Turing: Ejecución

Supongamos que $w = 00100$:

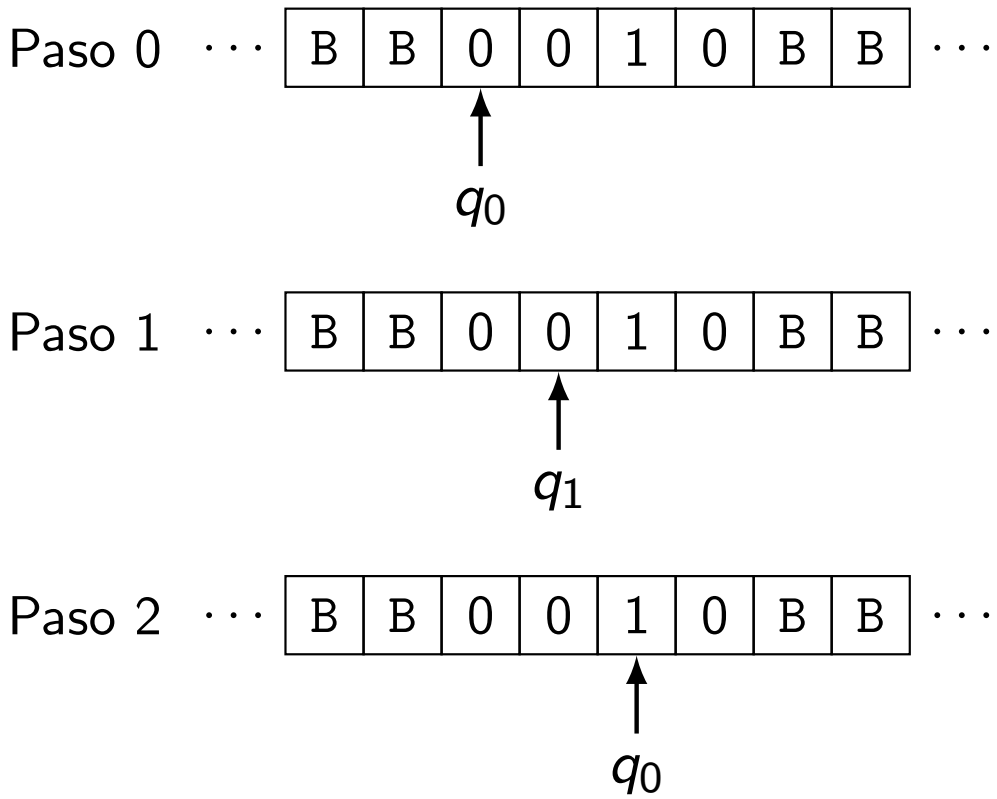


Máquinas de Turing: Ejecución

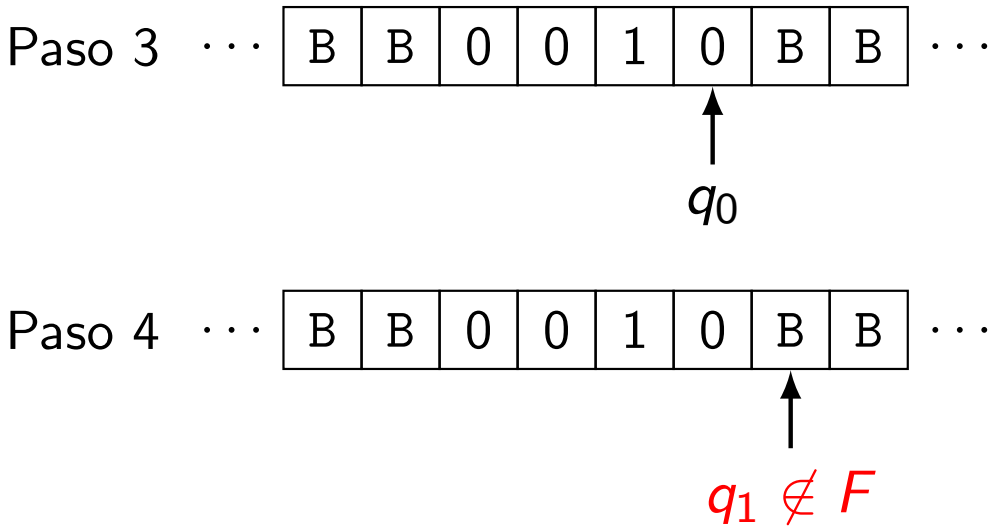


Máquinas de Turing: Ejecución

Ahora supongamos que $w = 0010$:

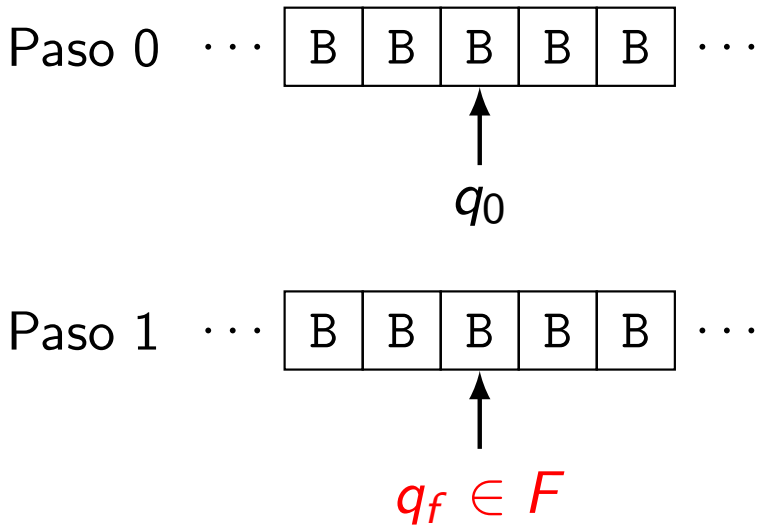


Máquinas de Turing: Ejecución



Máquinas de Turing: Palabra vacía

Finalmente supongamos que $w = \varepsilon$:



Inicialmente: La cabeza lectora se encuentra en una posición cualquiera.

El lenguaje aceptado por una máquina de Turing

Definición

Dada una máquina de Turing $M = (Q, A, q_0, \delta, F)$*:*

$$L(M) = \{w \in A^* \mid M \text{ acepta } w\}$$

$L(M)$ es el lenguaje aceptado por M.

En el ejemplo anterior: $L(M) = \{w \in \{0,1\}^* \mid$ el número de 0s en w es par$\}$.

El lenguaje aceptado por una máquina de Turing

Ejercicio

1. Construya una máquina de Turing que acepta el lenguaje de las palabras de 0s que tienen largo divisible por 3.

2. Construya una máquina de Turing que acepta el lenguaje de las palabras de 0s que tienen largo divisible por 6.

3. Construya una máquina de Turing que acepta el lenguaje de las palabras de 0s y 1s que tienen el mismo número de 0s y 1s.

Puede ejecutar sus programas aquí:

`http://db.ing.puc.cl/turingmachine`

Detención de una máquina de Turing

Una máquina de Turing puede no detenerse en alguna entrada.

Ejemplo

$M = (Q, A, q_0, \delta, F)$, donde $Q = \{q_0, q_1\}$, $A = \{0\}$, $F = \{q_1\}$ y δ es definida de la siguiente forma:

$$\begin{aligned} \delta(q_0, 0) &= (q_1, 0, \rightarrow) \\ \delta(q_0, \mathtt{B}) &= (q_0, \mathtt{B}, \rightarrow) \end{aligned}$$

¿Con qué palabra no se detiene esta máquina?

Problemas decidibles

Si una máquina de Turing no se detiene en alguna entrada, entonces no puede ser utilizada como un algoritmo.

- Un buen programa en Java o Python no se debería quedar en un loop infinito.

Definición

Un lenguaje L es decidible si existe una máquina de Turing M que se detiene en todas las entradas y tal que $L = L(M)$.

¿Existen problemas para los cuales no hay algoritmos? ¿Se puede demostrar esto?

Existencia de problemas no decidibles: Argumento basado en cardinalidad

Sea $A = \{0, 1\}$.

- ¿Cuántos lenguajes con alfabeto A existen?
- ¿Cuántas máquinas de Turing con alfabeto A existen?
- ¿Existen lenguajes no decidibles?

Existencia de problemas no decidibles: Argumento basado en cardinalidad

Sea $A = \{0, 1\}$.

- ¿Cuántos lenguajes con alfabeto A existen?
  - Tantos como $2^{\mathbb{N}}$, o equivalentemente $\mathbb{R}$
- ¿Cuántas máquinas de Turing con alfabeto A existen?
- ¿Existen lenguajes no decidibles?

Existencia de problemas no decidibles: Argumento basado en cardinalidad

Sea $A = \{0, 1\}$.

- ¿Cuántos lenguajes con alfabeto A existen?
  - Tantos como $2^{\mathbb{N}}$, o equivalentemente $\mathbb{R}$
- ¿Cuántas máquinas de Turing con alfabeto A existen?
  - Tantos como $\mathbb{N}$
- ¿Existen lenguajes no decidibles?

Existencia de problemas no decidibles: Argumento basado en cardinalidad

Sea $A = \{0, 1\}$.

- ¿Cuántos lenguajes con alfabeto A existen?
  - Tantos como $2^{\mathbb{N}}$, o equivalentemente $\mathbb{R}$
- ¿Cuántas máquinas de Turing con alfabeto A existen?
  - Tantos como $\mathbb{N}$
- ¿Existen lenguajes no decidibles?
  - Sí, por el teorema de Cantor

Un problema no decidible: Décimo problema de Hilbert

Problema H10: Dado un polinomio $p(x_1, \ldots, x_k)$ con coeficientes en $\mathbb{Z}$, determinar si existe $(a_1, \ldots, a_k) \in \mathbb{Z}^k$ tal que $p(a_1, \ldots, a_k) = 0$.

- ¿Cuál es el alfabeto para H10? ¿Cuál es la entrada de H10?

Un problema no decidible: Décimo problema de Hilbert

Problema H10: Dado un polinomio $p(x_1, \ldots, x_k)$ con coeficientes en $\mathbb{Z}$, determinar si existe $(a_1, \ldots, a_k) \in \mathbb{Z}^k$ tal que $p(a_1, \ldots, a_k) = 0$.

- ¿Cuál es el alfabeto para H10? ¿Cuál es la entrada de H10?

Teorema (Matiyasévich, Davis, Robinson & Putnam)

H10 no es decidible.

¿Dónde estamos?

El objetivo de esta clase es introducir la noción de complejidad computacional.

- Cuánto cuesta resolver un problema

Hasta ahora:

- Formalizamos la noción de algoritmo a través de las Máquinas de Turing
- Mostramos que hay problemas muy difíciles: no son decidibles

Lo que viene: Definir la complejidad de un algoritmo.

- Estudiar la complejidad de un problema

Complejidad de un algoritmo

Notación

Dada una MT determinista M con alfabeto A que para en todas las entradas:

- *Paso de M: Ejecutar una instrucción de la función de transición.*
- *$tiempo_M(w)$: Número de pasos ejecutados por M con entrada w.*

Complejidad de un algoritmo

Notación

Dada una MT determinista M con alfabeto A que para en todas las entradas:

- *Paso de M: Ejecutar una instrucción de la función de transición.*
- *$tiempo_M(w)$: Número de pasos ejecutados por M con entrada w.*

Definición

Dado un número natural n:

$$t_M(n) \quad = \quad \text{máx}\{\ tiempo_M(w) \ \mid \ w \in A^* \ y \ |w| = n\}.$$

t_M: tiempo de ejecución de M en el peor caso.

Complejidad de un problema

Definición

Un lenguaje L puede ser aceptado en tiempo t si existe una MT determinista M tal que:

- *M para en todas las entradas.*
- *$L = L(M)$.*
- *t_M es $O(t)$, vale decir, existe $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que $t_M(n) \leq c \cdot t(n)$ para todo $n \geq n_0$.*

Clases de complejidad

Definición

Dado un alfabeto A, $DTIME(t)$ se define como el conjunto de todos los lenguajes $L \subseteq A^*$ que pueden ser aceptados en tiempo t.

Dos clases fundamentales:

$$\text{PTIME} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k)$$

$$\text{EXPTIME} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(2^{n^k})$$

PTIME: conjunto de todos los problemas que pueden ser solucionados eficientemente.

Ciencia de la computación: Problema fundamental

El problema fundamental de nuestra disciplina:

Dado un problema, encontrar un algoritmo eficiente para solucionarlo o demostrar que no existe tal algoritmo.

Ciencia de la computación: Problema fundamental

El problema fundamental de nuestra disciplina:

> **Dado un problema, encontrar un algoritmo eficiente para solucionarlo o demostrar que no existe tal algoritmo.**

Primera parte: Demostrar que $L \in \text{DTIME}(t)$.

- Si $L = \{w \in \{0,1\}^* \mid \text{el número de 0s en } w \text{ es par}\}$, entonces $L \in \text{DTIME}(\ \ )$

Ciencia de la computación: Problema fundamental

El problema fundamental de nuestra disciplina:

> **Dado un problema, encontrar un algoritmo eficiente para solucionarlo o demostrar que no existe tal algoritmo.**

Primera parte: Demostrar que $L \in \text{DTIME}(t)$.

- Si $L = \{w \in \{0,1\}^* \mid \text{el número de 0s en } w \text{ es par}\}$, entonces $L \in \text{DTIME}(n)$

Ciencia de la computación: Problema fundamental

El problema fundamental de nuestra disciplina:

Dado un problema, encontrar un algoritmo eficiente para solucionarlo o demostrar que no existe tal algoritmo.

Primera parte: Demostrar que $L \in \text{DTIME}(t)$.

- Si $L = \{w \in \{0,1\}^* \mid \text{el número de 0s en } w \text{ es par}\}$, entonces $L \in \text{DTIME}(n)$

Segunda parte: Demostrar que $L \notin \text{DTIME}(t)$.

- ¿Es cierto que 3-COL $\notin$ PTIME?

Comentarios finales

Los computadores que usamos son equivalentes a las máquinas de Turing.

- Todo lo que puede ser implementado de manera eficiente en uno, puede ser implementado de manera eficiente en el otro

Lo que sabemos: $\text{PTIME} \subsetneq \text{EXPTIME}$

- Existen problemas (naturales) que pueden ser resueltos, pero no de manera eficiente

Comentarios finales

Lo que creemos (pero no sabemos cómo demostrar): 3-COL $\notin$ PTIME

- Esta es una formulación (equivalente) del problema "P vs NP"
- Puede cambiar 3-COL por el problema de satisfacción de fórmulas proposicionales, programación lineal entera, el problema del agente viajero, ...

Y existen muchas otras clases de complejidad, y problemas por resolver.

Transparencias adicionales

Máquinas de Turing no deterministas

Definición

Máquina de Turing no determinista: (Q, A, q_0, δ, F)

- Q *es un conjunto finito de estados*
- A *es un alfabeto tal que* $\mathtt{B} \notin A$
- q_0 *es el estado inicial* $(q_0 \in Q)$
- F *es un conjunto de estados finales* $(F \subseteq Q)$
- $\delta \subseteq Q \times (A \cup \{\mathtt{B}\}) \times Q \times (A \cup \{\mathtt{B}\}) \times \{\leftarrow, \square, \rightarrow\}$*: Relación de transición*

Máquinas de Turing no deterministas: Funcionamiento

Inicialmente: Tal como en una MT determinista.

En cada paso:

- La máquina lee el símbolo a en la celda apuntada por la cabeza lectora y determina en que estado q se encuentra.
- Determina el conjunto de todas las instrucciones para (q, a). Si este conjunto es vacío, entonces la máquina se detiene.
- Si el conjunto no es vacío, entonces escoge una instrucción de este conjunto y la ejecuta.

Si la máquina se detiene en un estado final, entonces la máquina acepta w.

Máquinas de Turing no deterministas: Lenguaje aceptado

Definición

Dada una máquina de Turing M no determinista con alfabeto A:

$$L(M) = \{w \in A^* \mid \textit{existe alguna ejecución de } M \textit{ con entrada } w \textit{ que termina en un estado final}\}.$$

Ejercicio

1. Sea $L = \{w \in \{0\}^* \mid \text{el largo de } w \text{ es divisible por 2 ó 3}\}$. Construya una MT no determinista que acepte L y que sólo mueva su cabeza a la derecha.

Máquinas de Turing no deterministas: Poder de computación

¿Son las máquinas de Turing no deterministas más poderosas que las deterministas?

Teorema

Para cada MT no determinista M, existe una MT determinista M' tal que $L(M) = L(M')$.

Ejercicio

Demuestre el teorema.

Máquinas de Turing no deterministas: Complejidad

Para una MT no determinista:

- Paso de M: Ejecutar una instrucción de la relación de transición
- $tiempo_M(w)$: Número de pasos de M con entrada w en la ejecución más corta que acepta a w

  Sólo está definido para palabras aceptadas por M
  - ¿Por qué?

Máquinas de Turing no deterministas: Complejidad

Definición

El tiempo de funcionamiento de una MT no determinista M en el peor caso es definido por la función t_M:

$$t_M(n) = \text{máx}\Big[\{n\} \cup \{tiempo_M(w) \mid w \in \Sigma^*, |w| = n \text{ y } M \text{ acepta a } w\}\Big]$$

Máquinas de Turing no deterministas: Complejidad

Definición

El tiempo de funcionamiento de una MT no determinista M en el peor caso es definido por la función t_M:

$$t_M(n) = \text{máx}\Big[\{n\} \cup \{tiempo_M(w) \mid w \in \Sigma^*, |w| = n \text{ y } M \text{ acepta a } w\}\Big]$$

¿Por qué incluimos $\{n\}$ en la definición?

Clases de complejidad no deterministas

Definición

Un lenguaje L es aceptado en tiempo t por una MT M no determinista si:

- $L = L(M)$
- t_M *es* $O(t)$

Clases de Complejidad no deterministas

Definición

NTIME(t) se define como el conjunto de todos los lenguajes que pueden ser aceptados en tiempo t por alguna MT no determinista.

Una clase fundamental:

$$\text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$$

¿Por qué es tan importante esta clase? ¿Qué problemas están en esta clase?

Algunas propiedades de la clase NP

¿Dónde vive NP? $\text{PTIME} \subseteq \text{NP} \subseteq \text{EXPTIME}$

- Se sabe que al menos una de estas inclusiones es estricta (¿por qué?), pero no se sabe cuál
- Tampoco se sabe si ambas inclusiones son estrictas

Algunas propiedades de la clase NP

¿Dónde vive NP? PTIME $\subseteq$ NP $\subseteq$ EXPTIME

- Se sabe que al menos una de estas inclusiones es estricta (¿por qué?), pero no se sabe cuál
- Tampoco se sabe si ambas inclusiones son estrictas

¿Qué problemas están en NP?

Ejercicio

Muestre que 3-COL está en NP.

Algunas propiedades de la clase NP

¿Dónde vive NP? PTIME $\subseteq$ NP $\subseteq$ EXPTIME

- Se sabe que al menos una de estas inclusiones es estricta (¿por qué?), pero no se sabe cuál
- Tampoco se sabe si ambas inclusiones son estrictas

¿Qué problemas están en NP?

Ejercicio

Muestre que 3-COL está en NP.

Uno de los problemas del milenio (Clay)

Demuestre que PTIME $\neq$ NP