¿Dónde estamos?

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

- Cuánto cuesta resolver un problema

¿Dónde estamos?

El objetivo de esta sección 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
- Demostramos que hay problemas muy difíciles: indecidibles
- Estudiamos alguna propiedades de los problemas indecidibles, y definimos los problemas recursivamente enumerables

¿Dónde estamos?

El objetivo de esta sección 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
- Demostramos que hay problemas muy difíciles: indecidibles
- Estudiamos alguna propiedades de los problemas indecidibles, y definimos los problemas recursivamente enumerables

Lo que viene:

- Definir la complejidad de un algoritmo
- Estudiar la complejidad de un problema

¿Dónde estamos?

El objetivo de esta sección 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
- Demostramos que hay problemas muy difíciles: indecidibles
- Estudiamos alguna propiedades de los problemas indecidibles, y definimos los problemas recursivamente enumerables

Lo que viene:

- Definir la complejidad de un algoritmo
- Estudiar la complejidad de un problema

Antes de esto: Máquinas de Turing no deterministas

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* $(F \neq \emptyset)$
- $\delta \subseteq Q \times (A \cup \{\mathtt{B}\}) \times Q \times (A \cup \{\mathtt{B}\}) \times \{\leftarrow, \rightarrow\}$*: Relación de transición*

  Suponemos que $\delta \neq \emptyset$

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:

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

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 $L = \{w \in \{0\}^* \mid$ el largo de w es divisible por 2 ó 3$\}$. Construya una MT no determinista que acepte L y que sólo mueva su cabeza a la derecha.
2. ¿Puede hacer lo mismo con una MT determinista?

Máquinas de Turing no deterministas: Poder de computación

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

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 $L(M) = L(M')$.

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.

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 $O(t)$,*

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 $n \geq n_0$*

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 $n \geq n_0$*

El tiempo para computar una función f se define de la misma forma.

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.

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.

Un problema fundamental

El problema fundamental de nuestra disciplina:

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

Un 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 (puede ser difícil): Demostrar que $L \in \text{DTIME}(t)$

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

Un 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 (puede ser difícil): Demostrar que $L \in \text{DTIME}(t)$

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

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

- ¿Es cierto que SAT $\notin$ PTIME?
- ¿Cómo podemos atacar este problema?

La noción de completitud: Reducción polinomial

Definición

Dados lenguajes L_1 y L_2 con alfabeto A, decimos que L_1 es reducible en tiempo polinomial a L_2 si existe una función f computable en tiempo polinomial tal que para todo $w \in A^*$:

$$w \in L_1 \text{ si y sólo si } f(w) \in L_2$$

La noción de completitud: Reducción polinomial

Definición

Dados lenguajes L_1 y L_2 con alfabeto A, decimos que L_1 es reducible en tiempo polinomial a L_2 si existe una función f computable en tiempo polinomial tal que para todo $w \in A^*$:

$$w \in L_1 \text{ si y sólo si } f(w) \in L_2$$

Teorema

Suponga que L_1 es reducible en tiempo polinomial a L_2.

- *Si $L_2 \in PTIME$ entonces $L_1 \in PTIME$*
- *Si $L_1 \notin PTIME$ entonces $L_2 \notin PTIME$*

La noción de completitud: Reducción polinomial

Definición

Dados lenguajes L_1 y L_2 con alfabeto A, decimos que L_1 es reducible en tiempo polinomial a L_2 si existe una función f computable en tiempo polinomial tal que para todo $w \in A^*$:

$$w \in L_1 \text{ si y sólo si } f(w) \in L_2$$

Teorema

Suponga que L_1 es reducible en tiempo polinomial a L_2.

- *Si $L_2 \in PTIME$ entonces $L_1 \in PTIME$*
- *Si $L_1 \notin PTIME$ entonces $L_2 \notin PTIME$*

Ejercicio

Demuestre el teorema.

La noción de completitud: Hardness

Definición

Dada una clase de complejidad $\mathcal{C}$ que contiene PTIME, decimos que L es hard para $\mathcal{C}$ si para todo $L' \in \mathcal{C}$ existe una reducción polinomial de L' a L.

La noción de completitud: Hardness

Definición

Dada una clase de complejidad $\mathcal{C}$ que contiene PTIME, decimos que L es hard para $\mathcal{C}$ si para todo $L' \in \mathcal{C}$ existe una reducción polinomial de L' a L.

Teorema

Si PTIME $\subsetneq \mathcal{C}$ y L es hard para $\mathcal{C}$, entonces $L \notin$ PTIME.

La noción de completitud: Hardness

Definición

Dada una clase de complejidad $\mathcal{C}$ que contiene PTIME, decimos que L es hard para $\mathcal{C}$ si para todo $L' \in \mathcal{C}$ existe una reducción polinomial de L' a L.

Teorema

Si PTIME $\subsetneq \mathcal{C}$ y L es hard para $\mathcal{C}$, entonces $L \notin$ PTIME.

Entonces: Para probar que $L \notin$ PTIME, basta encontrar una clase $\mathcal{C}$ tal que PTIME $\subsetneq \mathcal{C}$ y demostrar que L es hard para esta clase.

La noción de completitud: Hardness

Definición

Dada una clase de complejidad $\mathcal{C}$ que contiene PTIME, decimos que L es hard para $\mathcal{C}$ si para todo $L' \in \mathcal{C}$ existe una reducción polinomial de L' a L.

Teorema

Si PTIME $\subsetneq \mathcal{C}$ y L es hard para $\mathcal{C}$, entonces $L \notin$ PTIME.

Entonces: Para probar que $L \notin$ PTIME, basta encontrar una clase $\mathcal{C}$ tal que PTIME $\subsetneq \mathcal{C}$ y demostrar que L es hard para esta clase.

Ejercicio

Demuestre el teorema.

La noción de completitud

Pero: También queremos saber cual es la complejidad exacta de un problema.

Definición

Decimos que L es completo para $\mathcal{C}$ si $L \in \mathcal{C}$ y L es hard para $\mathcal{C}$.

La noción de completitud

Pero: También queremos saber cual es la complejidad exacta de un problema.

Definición

Decimos que L es completo para $\mathcal{C}$ si $L \in \mathcal{C}$ y L es hard para $\mathcal{C}$.

Corolario

Si $PTIME \subsetneq \mathcal{C}$ y L es completo para $\mathcal{C}$, entonces $L \notin PTIME$.

La noción de completitud

Pero: También queremos saber cual es la complejidad exacta de un problema.

Definición

Decimos que L es completo para $\mathcal{C}$ si $L \in \mathcal{C}$ y L es hard para $\mathcal{C}$.

Corolario

Si PTIME $\subsetneq \mathcal{C}$ y L es completo para $\mathcal{C}$, entonces $L \notin$ PTIME.

¿Se conoce alguna clase $\mathcal{C}$ tal que PTIME $\subsetneq \mathcal{C}$? ¿Se puede demostrar que SAT $\notin$ PTIME usando este enfoque?

La existencia de problemas difíciles

Veamos un ejemplo de una clase que contiene en forma estricta a PTIME:

Teorema

$PTIME \subsetneq EXPTIME$

La existencia de problemas difíciles

Veamos un ejemplo de una clase que contiene en forma estricta a PTIME:

Teorema

$PTIME \subsetneq EXPTIME$

Existen problemas *naturales* que son completos para EXPTIME, y que por lo tanto no pueden ser resueltos eficientemente.

La existencia de problemas difíciles

Veamos un ejemplo de una clase que contiene en forma estricta a PTIME:

Teorema

$PTIME \subsetneq EXPTIME$

Existen problemas *naturales* que son completos para EXPTIME, y que por lo tanto no pueden ser resueltos eficientemente.

Ejercicio

Demuestre que SAT $\in$ EXPTIME.

¿Es SAT un problema difícil?

Para demostrar que SAT $\notin$ PTIME, sólo tenemos que demostrar que SAT es hard para EXPTIME

¿Es SAT un problema difícil?

Para demostrar que SAT $\notin$ PTIME, sólo tenemos que demostrar que SAT es hard para EXPTIME

- Nadie sabe cómo hacer esto

¿Es SAT un problema difícil?

Para demostrar que SAT $\notin$ PTIME, sólo tenemos que demostrar que SAT es hard para EXPTIME

- Nadie sabe cómo hacer esto

Preguntas a resolver:

- ¿Qué clase representa la complejidad de SAT? ¿Para qué clase de complejidad SAT es completo?
- ¿Se puede demostrar que esta clase no es igual a PTIME?

La complejidad de SAT

Tenemos que usar máquinas de Turing no deterministas para entender la complejidad exacta de SAT.

La complejidad de SAT

Tenemos que usar máquinas de Turing no deterministas para entender la complejidad exacta de SAT.

Notación

Dada una MT no determinista M con alfabeto A:

- *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 esta definido para palabras aceptadas por M.

La complejidad de SAT

Definición

Dado un número natural n:

$$t_M(n) = \text{máx}\{n\} \cup \{\ tiempo_M(w) \mid w \in A^*, |w| = n \text{ y } M \text{ acepta } w\}$$

¿Por que 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.

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:

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

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? ¿Qué problemas son completos para 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 cual
- 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 cual
- Tampoco se sabe si ambas inclusiones son estrictas

¿Qué problemas están en NP?

Ejercicio

Muestre que SAT y 3-coloración de grafos están 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 cual
- Tampoco se sabe si ambas inclusiones son estrictas

¿Qué problemas están en NP?

Ejercicio

Muestre que SAT y 3-coloración de grafos están en NP.

¿Qué problemas son completos para esta clase?

Teorema de Cook

Teorema (Cook)

SAT es completo para la clase NP.

Teorema de Cook

Teorema (Cook)

SAT es completo para la clase NP.

Demostración: Sea $L \in \text{NP}$. Tenemos que demostrar que L es reducible en tiempo polinomial a SAT.

Esto significa que existe una función $f : \{0,1\}^* \to \{0,1\}^*$ tal que:

- f es computable en tiempo polinomial
- Para cada $w \in \{0,1\}^*$ se tiene que $w \in L$ si y sólo si $f(w) \in \text{SAT}$

Notación: $f(w) = \varphi_w$

Teorema de Cook: Demostración

¿Qué sabemos de L?

- Como $L \in$ NP, existe una MT M no determinista tal que $L = L(M)$ y t_M es $O(n^k)$, donde $k > 0$ es una constante

Vamos a representar el funcionamiento de M con entrada w usando la fórmula φ_w: M acepta w si y sólo si φ_w es satisfacible.

Teorema de Cook: Demostración

Suponemos:

- $M = (Q, \{0,1\}, q_0, \delta, F)$, donde $F = \{q_m\}$ y no existe una transición en δ para q_m
- Para cada $q \in (Q \setminus \{q_m\})$ y $a \in \{0, 1, \mathtt{B}\}$, existe al menos una transición en δ para (q, a)
- $w = a_0 \cdots a_{n-1}$, donde $n \geq 0$ y cada $a_i \in \{0,1\}$
  - Nótese que si $n = 0$, entonces $w = \varepsilon$

Teorema de Cook: Demostración

Usamos las siguientes variables proposicionales:

$$
\begin{array}{lcl}
s_{t,p,a} & : & t \in [0, t_M(n)],\ p \in [-t_M(n), t_M(n)] \text{ y } a \in \{0, 1, \mathtt{B}\}. \\
c_{t,p} & : & t \in [0, t_M(n)] \text{ y } p \in [-t_M(n), t_M(n)]. \\
e_{t,q} & : & t \in [0, t_M(n)] \text{ y } q \in Q.
\end{array}
$$

φ_w es definida como $\varphi_I \wedge \varphi_C \wedge \varphi_A \wedge \varphi_\delta$

φ_I: estado inicial

$$c_{0,0} \wedge e_{0,q_0} \wedge \left(\bigwedge_{p=-t_M(n)}^{-1} s_{0,p,\mathtt{B}} \right) \wedge \left(\bigwedge_{p=0}^{n-1} s_{0,p,a_p} \right) \wedge \left(\bigwedge_{p=n}^{t_M(n)} s_{0,p,\mathtt{B}} \right)$$

φ_I: estado inicial

$$c_{0,0} \wedge e_{0,q_0} \wedge \left(\bigwedge_{p=-t_M(n)}^{-1} s_{0,p,\mathtt{B}} \right) \wedge \left(\bigwedge_{p=0}^{n-1} s_{0,p,a_p} \right) \wedge \left(\bigwedge_{p=n}^{t_M(n)} s_{0,p,\mathtt{B}} \right)$$

Nótese que si $w = \varepsilon$, entonces φ_I es la siguiente fórmula:

$$c_{0,0} \wedge e_{0,q_0} \wedge \left(\bigwedge_{p=-t_M(n)}^{t_M(n)} s_{0,p,\mathtt{B}} \right)$$

φ_C: la máquina funciona correctamente

φ_C se define como la conjunción de cuatro fórmulas. Primero, cada celda siempre contiene un único símbolo:

$$\bigwedge_{t=0}^{t_M(n)} \bigwedge_{p=-t_M(n)}^{t_M(n)} \Big((s_{t,p,0} \vee s_{t,p,1} \vee s_{t,p,\mathrm{B}}) \wedge (s_{t,p,0} \rightarrow (\neg s_{t,p,1} \wedge \neg s_{t,p,\mathrm{B}})) \wedge$$

$$(s_{t,p,1} \rightarrow (\neg s_{t,p,0} \wedge \neg s_{t,p,\mathrm{B}})) \wedge (s_{t,p,\mathrm{B}} \rightarrow (\neg s_{t,p,0} \wedge \neg s_{t,p,1})) \Big)$$

φ_C: la máquina funciona correctamente

Segundo, la máquina siempre está en un único estado:

$$\bigwedge_{t=0}^{t_M(n)} \left(\bigvee_{q \in Q} \left(e_{t,q} \wedge \bigwedge_{q' \in (Q \setminus \{q\})} \neg e_{t,q'} \right) \right)$$

Tercero, la cabeza siempre está en una única posición:

$$\bigwedge_{t=0}^{t_M(n)} \left(\bigvee_{p=-t_M(n)}^{t_M(n)} \left(c_{t,p} \wedge \bigwedge_{p' \in ([-t_M(n), t_M(n)] \setminus \{p\})} \neg c_{t,p'} \right) \right)$$

φ_C: la máquina funciona correctamente

Cuarto, el valor de una celda no cambia si no es apuntada por la cabeza lectora:

$$\bigwedge_{t=0}^{t_M(n)-1} \bigwedge_{p=-t_M(n)}^{t_M(n)} \Big((\neg c_{t,p}) \rightarrow ((s_{t,p,0} \wedge s_{t+1,p,0}) \vee (s_{t,p,1} \wedge s_{t+1,p,1}) \vee (s_{t,p,\mathtt{B}} \wedge s_{t+1,p,\mathtt{B}})) \Big)$$

φ_A: la máquina acepta w

$$\bigvee_{t=0}^{t_M(n)} e_{t,q_m}$$

φ_δ: relación δ define como funciona la máquina

$$\bigwedge_{t=0}^{t_M(n)-1} \bigwedge_{p=-(t_M(n)-1)}^{t_M(n)-1} \left(\bigwedge_{(q,a)\in Q\times\{0,1,\mathtt{B}\}} \left((e_{t,q} \wedge c_{t,p} \wedge s_{t,p,a}) \rightarrow \left(\bigvee_{(q',a',k)\,:\,(q,a,q',a',k)\in\delta} (e_{t+1,q'} \wedge c_{t+1,p+k} \wedge s_{t+1,p,a'}) \right) \right) \right)$$

Representamos $\leftarrow$ como -1 y $\rightarrow$ como 1. $\square$