¿Dónde estamos?

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

- Cuanto 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.

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

Notación: A las máquinas de Turing que hemos visto hasta ahora las llamamos deterministas.

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

- Q es un conjunto finito de estados.
- A es un alfabeto.
- q_0 es el estado inicial ($q_0 \in Q$).
- F es un conjunto de estados finales ($F \neq \emptyset$).
- $\delta \subseteq Q \times A \times Q \times A \times \{I, D\}$: Relación de transición.

  Asumimos 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

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

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

Ejercicio: 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.

Ejercicio: ¿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?

Teorema: Para cada MT no determinista M, existe una MT determinista M' tal que M y M' se detienen en las mismas palabras y $L(M) = L(M')$.

Ejercicio: Demuestre el teorema.

Complejidad de un algoritmo

Dado: 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.

Dado un número natural n:

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

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

Complejidad de un problema

Un lenguaje L puede ser aceptado en tiempo t si es que 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

Dado: alfabeto A.

DTIME(t): 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.**

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?

¿Como podemos atacar el segundo problema?

La noción de completitud: Reducción polinomial

Dado: Lenguajes L_1 y L_2 con alfabeto A.

Definición: L_1 es reducible en tiempo polinomial a L_2 si existe una función f computable en tiempo polinomial tal que para todo $w \in A^*$:

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

Corolario: Asuma 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 corolario.

La noción de completitud: Hardness

Dado: Clase de complejidad $\mathcal{C}$ que contiene PTIME.

Definición: 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.

Para probar que $L \notin$ PTIME: encuentre una clase $\mathcal{C}$ tal que PTIME $\subsetneq \mathcal{C}$ y demuestre 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 $\mathcal{C}$.

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

¿Se sabe de 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.

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

- Nadie sabe como 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?