Introducción a la Complejidad Computacional

IIC2213

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.

Lenguaje L: Conjunto de palabras.

- Ejemplo: $L = \{0^n 1^n \mid n \in \mathbb{N}\}$.

Problemas de decisión

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

Ejemplo

Podemos ver SAT como un problema de decisión. Suponga que $P = \{p, q\}$:

- $A = \{p, q, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, (,)\}$
  Algunas palabras de A^* representan fórmulas de $L(P)$, mientras que otras tales como $\neg\neg$ y $p\neg q \wedge \wedge \vee q$ no representan fórmulas.
- SAT $= \{w \in A^* \mid w$ representa una fórmula de $L(P)$ y w es satisfacible$\}$.

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.

Ejercicio

Muestre que $L = \{w \in \{0,1\}^* \mid$ número de 0s y 1s en w es el mismo$\}$ puede ser resuelto eficientemente.

¿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.

¿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?

- 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: 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 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 que 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 a la derecha o a la izquierda.

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

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* $(F \neq \emptyset)$.
- δ *es una función parcial:*

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

δ *es llamada función de transición. Suponemos que su dominio no es vacío.*

Máquinas de Turing: Ejemplo

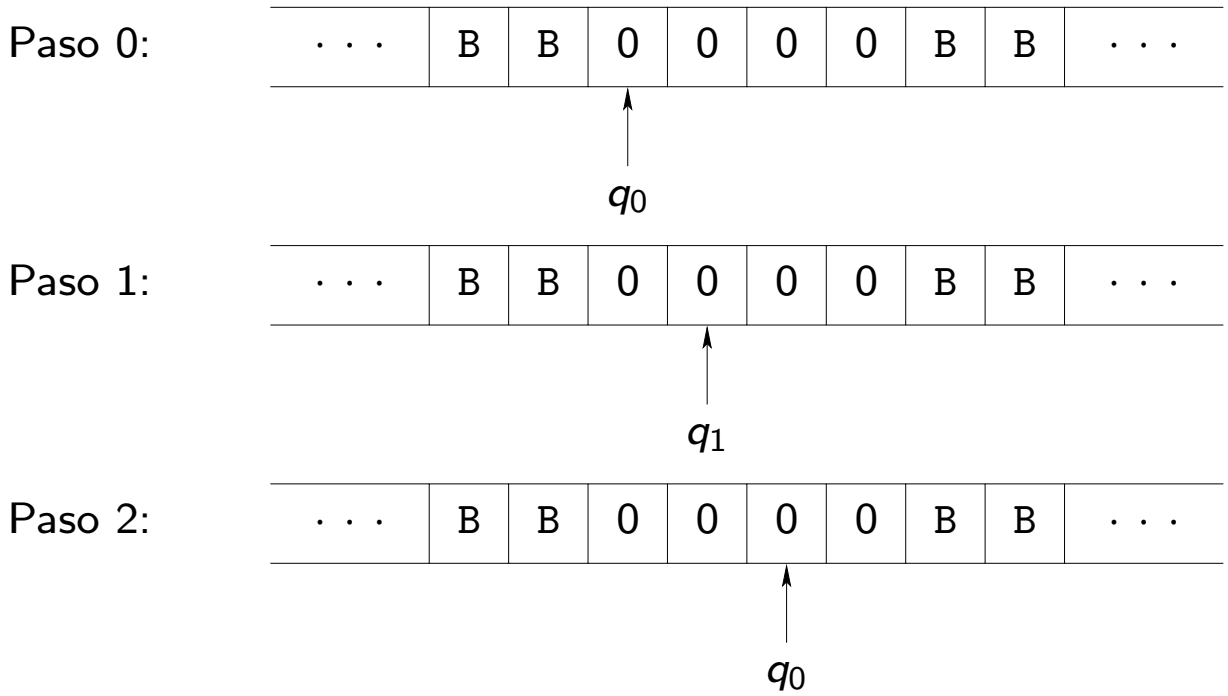
Queremos construir una máquina que verifique si el largo de una palabra de 0s es par: $M = (Q, A, q_0, \delta, F)$

- $A = \{0\}$.
- $Q = \{q_0, q_1, q_S, q_N\}$.
- $F = \{q_S\}$.
- δ es definida como:

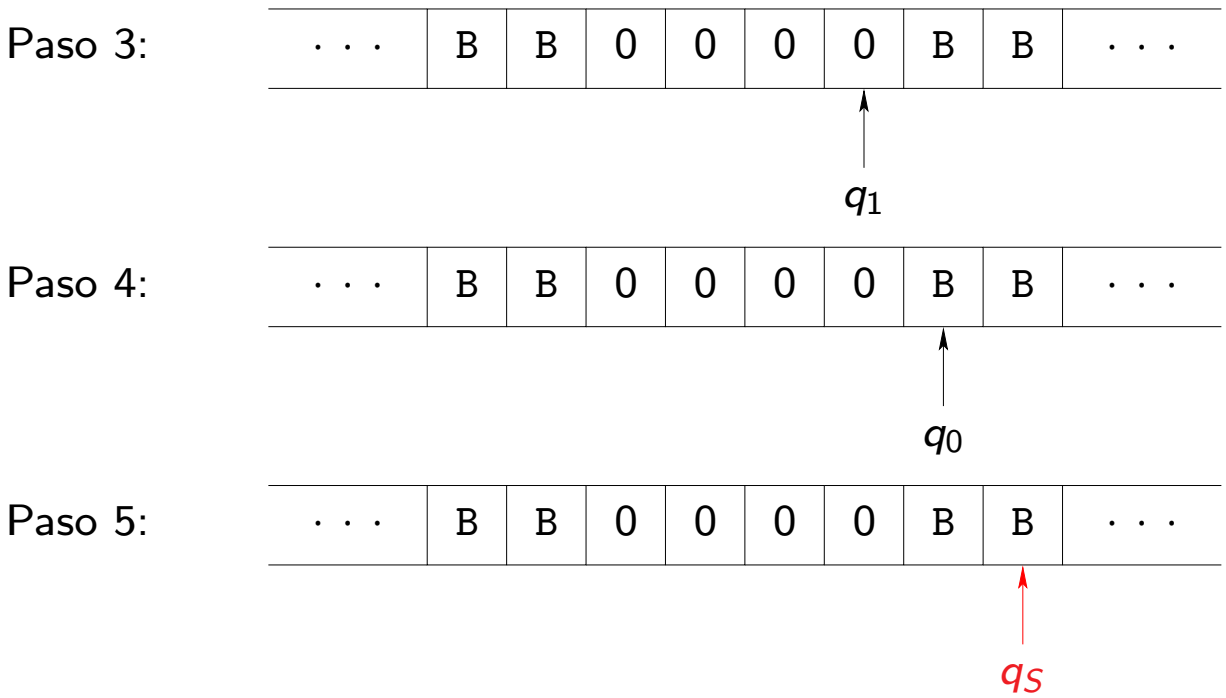
$$\begin{aligned}
\delta(q_0, 0) &= (q_1, 0, \rightarrow) \\
\delta(q_0, \mathtt{B}) &= (q_S, \mathtt{B}, \rightarrow) \\
\delta(q_1, 0) &= (q_0, 0, \rightarrow) \\
\delta(q_1, \mathtt{B}) &= (q_N, \mathtt{B}, \rightarrow)
\end{aligned}$$

Máquinas de Turing: Ejecución

Supongamos que $w = 0000$:

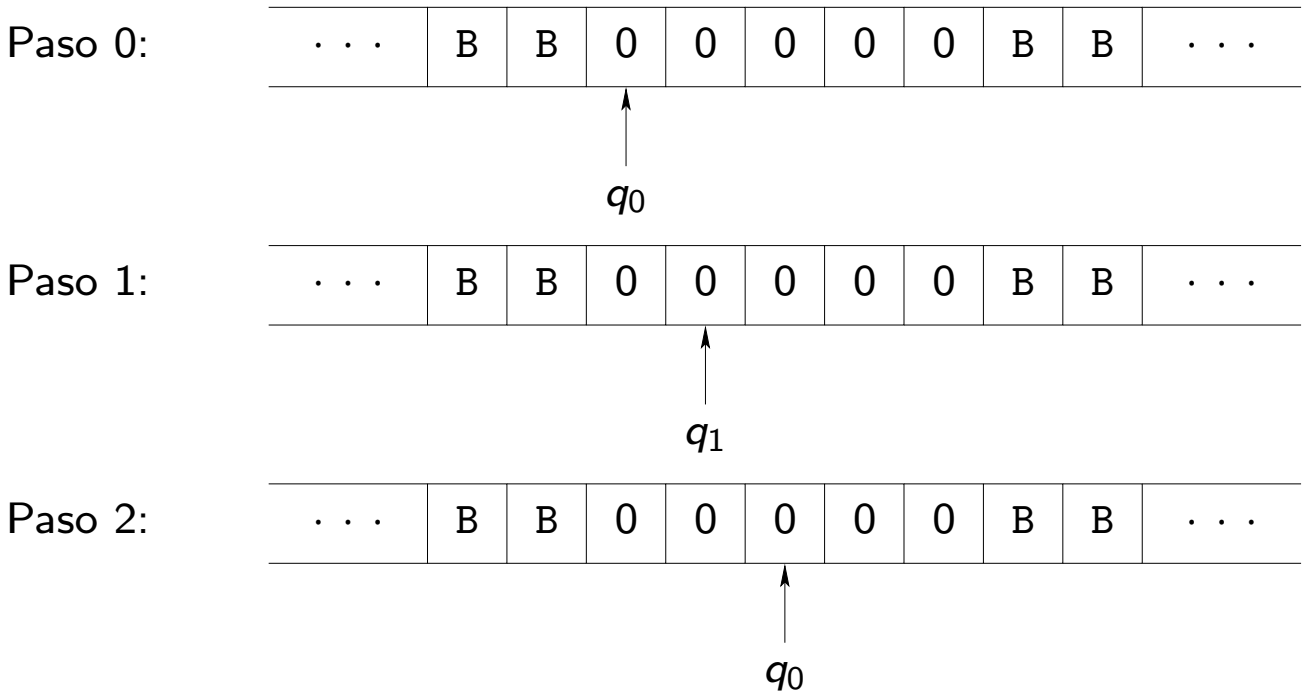


Máquinas de Turing: Ejecución

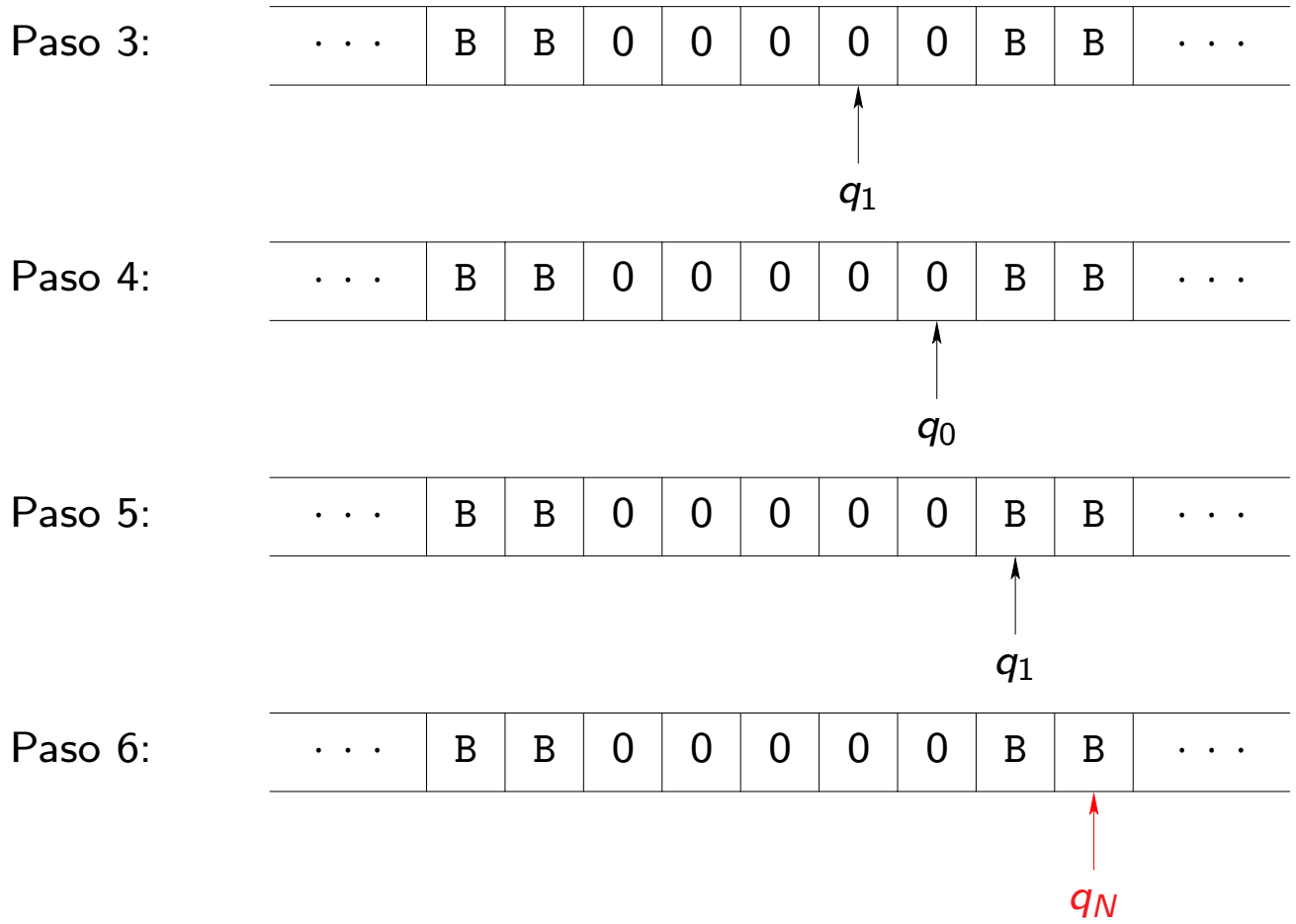


Máquinas de Turing: Otra ejecución

Ahora supongamos que $w = 00000$:

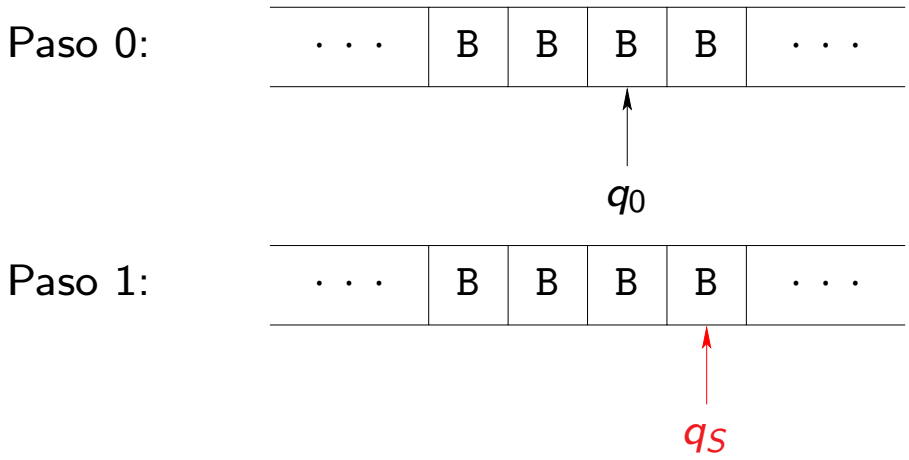


Máquinas de Turing: Otra ejecución



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 $M = (Q, A, q_0, \delta, F)$*:*

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

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

En el ejemplo anterior: $L(M) = \{w \in \{0\}^* \mid \text{largo de } w \text{ es par}\}$.

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.

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 C o Java 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

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

- ¿Cuántos lenguajes con alfabeto A existen?
- ¿Cuántas máquinas de Turing con alfabeto A existen?
- ¿Existen lenguajes no decidibles?
  - Si lanzamos un dardo al azar sobre el espacio de todos los lenguajes con alfabeto A, ¿cuál es la probabilidad de que el dardo caiga en un problema decidible?

Existencia de problemas no decidibles

La demostración anterior no es constructiva.

- Sabemos que existe un problema indecidible, pero no sabemos como es este lenguaje.

Vamos a construir un lenguaje indecidible.

- Usamos diagonalización.

Codificación de una máquina de Turing

Primero tenemos que introducir algo de terminología.

Supongamos que $M = (Q, A, q_1, \delta, F)$, donde

- $Q = \{q_1, \ldots, q_n\}$.
- $A = \{0, 1\}$.
- $F = \{q_{i_1}, q_{i_2}, \ldots, q_{i_m}\}$, $1 \leq i_1 < i_2 < \cdots < i_m \leq n$.

Definimos una palabra $C(M) = w_Q 00 w_\delta 00 w_F$ que representa a M.

Codificación de una máquina de Turing

- w_Q:

$$001001100 \cdots 00 \underbrace{1 \cdots 1}_{n \text{ veces}}$$

- w_δ: Usamos 1 para representar símbolo 0, 11 para símbolo 1 y 111 para símbolo B. También usamos 1 para representar $\leftarrow$ y 11 para $\rightarrow$.

  w_δ es de la forma $w_1 \cdots w_k$, donde cada w_i representa una transición. Por ejemplo, si $\delta(q_i, 0) = (q_j, \mathtt{B}, \leftarrow)$, entonces la siguiente palabra representa esta transición:

$$00 \underbrace{1 \cdots 1}_{i \text{ veces}} 00100 \underbrace{1 \cdots 1}_{j \text{ veces}} 00111001$$

Codificación de una máquina de Turing

En cambio, si $\delta(q_i, 1) = (q_j, 0, \rightarrow)$, entonces la siguiente palabra representa la transición:

$$00\underbrace{1\cdots1}_{i \text{ veces}}001100\underbrace{1\cdots1}_{j \text{ veces}}0010011$$

- w_F:

$$00\underbrace{1\cdots1}_{i_1 \text{ veces}}00\underbrace{1\cdots1}_{i_2 \text{ veces}}00\cdots00\underbrace{1\cdots1}_{i_m \text{ veces}}$$

Codificación de una máquina de Turing: Ejemplo

Sea $M = (Q, A, q_1, \delta, F)$, donde $Q = \{q_1, q_2, q_3, q_4\}$, $A = \{0, 1\}$, $F = \{q_3\}$ y δ es definido de la siguiente forma:

$$\begin{array}{lcl} \delta(q_1, 0) & = & (q_1, 0, \rightarrow) \\ \delta(q_1, 1) & = & (q_2, 1, \rightarrow) \\ \delta(q_2, 1) & = & (q_4, 1, \leftarrow) \end{array} \qquad \begin{array}{lcl} \delta(q_2, \mathtt{B}) & = & (q_3, \mathtt{B}, \leftarrow) \\ \delta(q_2, 0) & = & (q_4, 0, \leftarrow) \end{array}$$

Entonces: $C(M) = w_Q 00 w_\delta 00 w_F$, donde:

$$\begin{array}{rcl} w_Q & = & 00100110011100 1111 \\ w_F & = & 00111 \\ w_\delta & = & 00100100100100110010011001100110011 \\ & & 00110011100111001110010011001001111001001 \\ & & 0011001100111110011001 \end{array}$$

Un problema indecidible: Problema de la parada

Problema natural

Dada una máquina de Turing M y una palabra w, ¿se detiene M con entrada w?

Como problema de decisión: Consideramos máquinas de Turing con alfabeto $\{0, 1\}$. Entonces,

$$U = \{w' \in \{0,1\}^* \mid \text{existe una MT } M \text{ y } w \in A^* \text{ tal que } w' = C(M)0000w \text{ y } M \text{ se detiene con entrada } w\}.$$

¿Es este problema decidible?

- Vamos a demostrar que no, pero para eso vamos a considerar primero un problema más simple.

Un problema de decisión más simple

Otro problema de decisión

Dada una máquina de Turing M, ¿es cierto que M se detiene con entrada $C(M)$?

Le damos como entrada a M su propio código, ¿es válido hacer esto?

- Sí, $C(M)$ es una palabra como cualquier otra.

Como problema de decisión:

$$H = \{w \in \{0,1\}^* \mid \text{existe una MT } M \text{ tal que } w = C(M) \text{ y } M \text{ se detiene con entrada } C(M)\}.$$

Un problema de decisión más simple: Demostración

Teorema

H es indecidible, vale decir, no existe una máquina de Turing M que se detiene en todas las entradas y tal que $H = L(M)$.

Demostración: Por contradicción, asuma que existe una MT $M^\star$ que se detiene en todas las entradas y tal que $H = L(M^\star)$.

Vale decir: para cada MT M, si M se detiene con su propio código, entonces $M^\star$ con entrada $C(M)$ se detiene en un estado final. En caso contrario, $M^\star$ se detiene en un estado que no es final.

Un problema de decisión más simple: Demostración

Sea M_0 una MT que con entrada w funciona de la siguiente forma:

hace funcionar la máquina $M^\star$ con entrada w
si $M^\star$ se detiene en un estado final
    entonces M_0 no se detiene
    en caso contrario M_0 se detiene en un estado final

¿Es posible construir M_0? ¿Cómo?

Un problema de decisión más simple: Demostración

La pregunta ahora es si $M^\star$ acepta M_0:

$$
\begin{gathered}
M^\star \text{ acepta } M_0 \\
\text{si y sólo si} \\
M_0 \text{ se detiene con entrada } C(M_0) \\
\text{si y sólo si} \\
M^\star \text{ se detiene en un estado no final con entrada } C(M_0) \\
\text{si y sólo si} \\
M^\star \text{ no acepta } M_0
\end{gathered}
$$

¡Tenemos una contradicción!

Un problema indecidible: Problema de la parada

Podemos concluir que nuestro problema original U es indecidible.

- ¿Por qué?

Pero, ¿cómo demostramos formalmente que U es indecidible?

- Tenemos que formalizar el concepto de reducción.

La noción de reducción

Dado: Lenguajes L_1 y L_2 con alfabeto A.

Intuición: Decimos que un lenguaje L_1 es reducible a un lenguaje L_2 si existe un algoritmo tal que dado $w_1 \in A^*$, calcula $w_2 \in A^*$ tal que $w_1 \in L_1$ si y sólo si $w_2 \in L_2$.

Para formalizar esta idea necesitamos introducir la noción de algoritmo que calcula.

- Tenemos que introducir máquinas de Turing que calculan.

Máquinas de Turing que calculan

Definición

Una MT M calcula una función $f : A^* \longrightarrow A^*$ si:

> *con entrada $w \in A^*$, la máquina se detiene en un estado final teniendo en la memoria una cadena infinita de símbolos* `B` *seguida de $f(w)$ seguida de otra cadena infinita de símbolos* `B`*.*

Definición

Una función f es computable si es que existe una máquina de Turing que calcula f.

Máquinas de Turing que calculan

Ejemplo

Sea $g : \{0,1\}^* \to \{0,1\}^*$ una función definida como $g(w) = w0000$. Queremos demostrar que g es computable.

Sea $M = (Q, A, q_1, \delta, F)$, donde

- $Q = \{q_1, q_2, q_3, q_4, q_F\}$.
- $A = \{0,1\}$.
- $F = \{q_F\}$.
- δ es definida como:

$$\begin{array}{lcl} \delta(q_1, 0) & = & (q_1, 0, \to) \\ \delta(q_1, 1) & = & (q_1, 1, \to) \\ \delta(q_1, \mathrm{B}) & = & (q_2, 0, \to) \end{array} \qquad \begin{array}{l} \delta(q_2, \mathrm{B}) = (q_3, 0, \to) \\ \delta(q_3, \mathrm{B}) = (q_4, 0, \to) \\ \delta(q_4, \mathrm{B}) = (q_F, 0, \to) \end{array}$$

La noción de reducción: Formalización

Definición

Dados lenguajes L_1 y L_2 con alfabeto A, L_1 es reducible a L_2 si existe una función computable f tal que para todo $w \in A^*$:
$w \in L_1$ si y sólo si $f(w) \in L_2$.

Teorema

Suponga que L_1 es reducible a L_2.

- *Si L_2 es decidible entonces L_1 es decidible.*
- *Si L_1 es indecidible entonces L_2 es indecidible.*

Ejercicio

Demuestre el teorema.

El problema de la parada: Demostración

Por fin estamos listos para demostrar que el problema de la parada U es indecidible.

Demostración: Sea $g : \{0,1\}^* \longrightarrow \{0,1\}^*$ definida como $g(w) = w0000w$.

Es fácil ver que g es computable y es una reducción de H a U.

Dado que H es indecidible, concluimos que U es indecidible.

Paréntesis: Otro problema indecidible

¿Puede una máquina de Turing funcionar como un computador?

- Un computador es de uso general, no es usado para resolver un solo problema.

Definición

Máquina de Turing universal M_A con alfabeto $\{0,1\}$:

- *Recibe como entrada $C(M)0000w$.*
- *Simula M con entrada w.*
- *Se detiene en un estado final si y sólo si M acepta w. Además, si M con entrada w retorna x, entonces M_A también retorna x.*

Paréntesis: La noción de aleatoriedad

Suponga que cada vez que lanzamos una moneda anotamos un 0 si sale cara y un 1 si sale sello.

- Decimos que estamos generando una palabra "aleatoria".

¿Cómo podemos definir la noción de "ser aleatorio"?

- ¿Es la siguiente palabra aleatoria? ¿Por qué?
  0000000000000000000000000000000000000000
- ¿Qué sucede con el siguiente caso? ¿Es aleatoria?
  0101010101010101010101010101010101010101
- ¿Y en el siguiente caso diría que la palabra es aleatoria?
  0001101101111100010101000100111001100110

Paréntesis: Complejidad de Kolmogorov

Cuan aleatoria es una palabra depende de cuanto nos cuesta describirla.

- Para formalizar esta idea tenemos que definir que significa "describir una palabra".

Definición

Dado una palabra x, definimos la complejidad de Kolmogorov de x como:

$$K_A(x) \quad = \quad min\{\ |y| \ \mid \ M_A \text{ con entrada } y \text{ retorna } x\},$$

donde $|y|$ es el largo de y. En particular, $|\varepsilon| = 0$.

Ejercicio

De una cota superior para $K_A(x)$ en función de $|x|$.

Paréntesis: Complejidad de Kolmogorov

La complejidad de Kolmogorov depende de la MT universal que elijamos. ¿Qué pasa si elegimos otra MT universal?

Teorema

Si M_B es otra MT universal, entonces
$K_A(x) \leq K_B(x) + |C(M_B)| + 4$

Vale decir: Las dos medidas sólo difieren en una constante.

Ejercicio

Demuestre el teorema.

Paréntesis: Complejidad de Kolmogorov y aleatoriedad

Noción de aleatoriedad: una palabra x es aleatoria si su estructura es compleja.

- x es "difícil de describir" (no puede ser "comprimida" mucho).

Definición

Una palabra x es aleatoria si:

$$K_A(x) > \frac{|x|}{2}.$$

Ejercicio

1. Demuestre que existen infinitas palabras aleatorias.
2. ¿Existen más palabra aleatorias que no aleatorias?

Paréntesis: Aleatoriedad es indecidible

Problema de decisión: Dado $x \in \{0,1\}^*$, ¿es x aleatorio?

Vamos a demostrar que este problema es indecidible.

- Nuevamente usamos diagonalización.

Teorema

El lenguaje $L_A = \{x \in \{0,1\}^* \mid K_A(x) > \frac{|x|}{2}\}$ *es indecidible.*

Demostración: Suponga que L_A es decidible. Entonces existe una MT $M^\star$ que se detiene en todas las entradas y tal que $L(M^\star) = L_A$.

Paréntesis: Aleatoriedad es indecidible

Defina una MT M que con entrada $y \in \{0,1\}^*$ funciona de la siguiente forma:

asigna 0 a x
mientras que $M^\star$ no acepta x o $|x| < 2 \cdot y$
    incrementa x en 1
retorna x

Con entrada y: M genera una palabra x tal que $K_A(x) > \frac{|x|}{2} \geq y$.

Paréntesis: Aleatoriedad es indecidible

Dado que $|y| = \lceil \log(y+1) \rceil$: Existe y_0 tal que $|C(M)| + 4 + |y_0| \leq y_0$.

Sea x_0 el valor retornado por M con entrada y_0.

Entonces : $K_A(x_0) > \frac{|x_0|}{2} \geq y_0$.

Pero M con entrada y_0 describe a x_0: MT universal M_A con entrada $C(M)0000y_0$ retorna x_0. Entonces:

Entonces : $K_A(x_0) \leq |C(M)| + 4 + |y_0| \leq y_0$.

¡Tenemos una contradicción!

¿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

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:

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

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.
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?

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

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 $n \geq n_0$.*

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.

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?
- ¿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 $w \in A^*$:

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

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.

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 $\mathcal{C}$.

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

$PTIME \subsetneq EXPTIME$.

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?

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.

Una clase fundamental:

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

¿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.

¿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.

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

Esto significa que existe una función $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ 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 $L \in \text{NP}$, 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.

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 $a \in \{0, 1, \text{B}\}$, existe al menos una transición en δ para (q, a).
- $w = a_0 \cdots a_{n-1}$, donde cada $a_i \in \{0, 1\}$.

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 $\varphi_I \wedge \varphi_C \wedge \varphi_A \wedge \varphi_\delta$.

Teorema de Cook: Demostración

φ_I: Estado inicial.

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

Teorema de Cook: Demostración

φ_C: La máquina funciona correctamente.

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

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

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

Teorema de Cook: Demostración

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:

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

Teorema de Cook: Demostración

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

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

Teorema de Cook: Demostración

φ_A: La máquina acepta w.

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

Teorema de Cook: Demostración

φ_δ: relación δ define como funciona la máquina.

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

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

Otros problemas NP-completos

Notación

Si L es completo para $\mathcal{C}$, decimos que L es $\mathcal{C}$-completo.

¿Cómo podemos demostrar que un problema L es NP-completo?

- Nos basta con demostrar que $L \in \text{NP}$ y que existe una reducción polinomial de SAT a L.

Antes de utilizar esta técnica tenemos que pensar que *versión* de SAT usar.

- En general es más fácil utilizar fórmulas en alguna forma normal.

Otros problemas NP-completos

Una versión de SAT muy útil:

$$\text{CNF-SAT} = \{\varphi \mid \varphi \text{ es una fórmula en CNF y } \varphi \text{ es satisfacible}\}.$$

¿Es CNF-SAT NP-completo?

- Podemos modificar la demostración del Teorema de Cook para mostrar esto.

Teorema

CNF-SAT es NP-completo.

Pero algunas versiones de SAT son más simples . . .

Otra versión de SAT:

$$\text{DNF-SAT} = \{\varphi \mid \varphi \text{ es una fórmula en DNF y } \varphi \text{ es satisfacible}\}.$$

¿Es DNF-SAT NP-completo?

- ¿Podemos modificar la demostración del Teorema de Cook para mostrar esto?

Teorema

DNF-SAT está en PTIME.

Otros problemas NP-completos

Ahora podemos usar la idea inicial para mostrar que otros problemas son NP-completos.

Algunos ejemplos:

- 3-CNF-SAT $= \{\varphi \mid \varphi$ es una fórmula en CNF que contiene tres literales por cláusula y es satisfacible$\}$.
- 3-COL $= \{G \mid G$ es una grafo 3-coloreable$\}$.
- PRO-ENT $= \{(A, \vec{b}) \mid A$ es una matriz de enteros, $\vec{b}$ es un vector de enteros y existe un vector de enteros $\vec{x}$ tal que $A\vec{x} \leq \vec{b}\}$.

Conjetura de Cook

Conjetura de Cook

PTIME $\subsetneq$ NP

Si la conjetura es cierta:

- No existe un algoritmo eficiente para resolver SAT.
- Tampoco existen algoritmos eficientes para resolver ninguno de los problemas NP-completos.

¿Cuán fundamental es este problema?

Conjetura de Cook

Desde la página del Clay Mathematics Institute:

> "In order to celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) has named seven Prize Problems. The Scientific Advisory Board of CMI selected these problems, focusing on important classic questions that have resisted solution over the years. The Board of Directors of CMI designated a $7 million prize fund for the solution to these problems, with $1 million allocated to each."

Conjetura de Cook

Uno de los siete problemas es PTIME versus NP:

> "Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory ..., however the task of generating such a list from scratch seems to be so hard as to be completely impractical ..."

Ejercicio

Muestre que el problema mencionado en este párrafo es NP-completo.

Que hacer con un problema NP-completo

¿Qué hacemos si tenemos que solucionar un problema NP-completo?

- Existen casos que no vamos a ser capaces de solucionar.
- Pero, ¿Qué pasa en la práctica?

Dado un problema NP-completo:

- Nos gustaría encontrar casos que puedan ser solucionados eficientemente y que sean interesantes en la práctica.

Ejemplo

3-COL es NP-completo pero 2-COL puede ser solucionado eficientemente.

Vamos a ver dos casos en los que SAT puede ser resuelto eficientemente: 2-CNF-SAT y Cláusulas de Horn.

La complejidad de 2-CNF-SAT

Sea 2-CNF-SAT $= \{\varphi \mid \varphi$ es una fórmula en CNF que contiene dos literales por cláusula y es satisfacible$\}$.

Teorema

2-CNF-SAT puede ser solucionado eficientemente.

Demostración: Sea φ una fórmula en CNF que contiene dos literales por cláusula. Suponga que φ menciona las variables proposicionales $x_1, \ldots, x_n$.

Sea $G_\varphi = (N, A)$ un grafo construido de la siguiente forma:

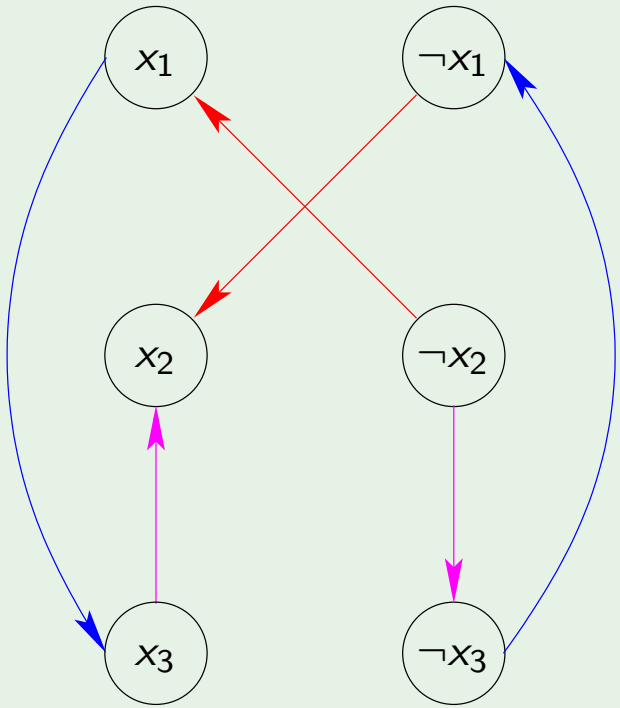
- $N = \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$.
- Para cada cláusula $C = l_1 \vee l_2$ en φ, A contiene los arcos $(\bar{l_1}, l_2)$ y $(\bar{l_2}, l_1)$.

Nótese que este grafo es dirigido.

La complejidad de 2-CNF-SAT

Ejemplo

Si $\varphi = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3)$, entonces G_φ es el siguiente grafo:



La complejidad de 2-CNF-SAT

¿Por qué construimos G_φ? ¿Cuál es la relación entre φ y G_φ?

Vamos a demostrar que φ es satisfacible si y sólo si no existe un nodo x_i tal que $\neg x_i$ es alcanzable desde x_i y x_i es alcanzable desde $\neg x_i$ en G_φ.

¿Por qué nos sirve esta propiedad?

- Porque se puede verificar en tiempo polinomial (¿cómo?).

La complejidad de 2-CNF-SAT

($\Rightarrow$) Por contradicción: Suponga que φ es satisfacible y existe x_i tal que $\neg x_i$ es alcanzable desde x_i y x_i es alcanzable desde $\neg x_i$ en G_φ.

Sea σ una valuación tal que $\sigma(\varphi) = 1$.

Consideramos $\sigma(x_i) = 1$. El otro caso se resuelve de la misma forma.

La complejidad de 2-CNF-SAT

Como $\neg x_i$ es alcanzable desde x_i en G_φ: Existe un camino $l_1, \ldots, l_k$ en G_φ tal que $k \geq 2$, $l_1 = x_i$ y $l_k = \neg x_i$.

Como $\sigma(\neg x_i) = 0$: Existe $j \in [1, k-1]$ tal que $\sigma(l_j) = 1$ y $\sigma(l_{j+1}) = 0$.

Como $(l_j, l_{j+1}) \in A$: $(\overline{l_j} \vee l_{j+1})$ o $(l_{j+1} \vee \overline{l_j})$ es una cláusula en φ.

Pero: σ no satisface a ninguna de estas cláusulas, por lo que $\sigma(\varphi) = 0$, una contradicción.

La complejidad de 2-CNF-SAT

($\Leftarrow$) Suponga que para todo $i \in [1, n]$: x_i no es alcanzable desde $\neg x_i$ o $\neg x_i$ no es alcanzable desde x_i en G_φ.

Usamos G_φ para construir una valuación σ tal que $\sigma(\varphi) = 1$:

Sea $V = \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$
Mientras $V \neq \emptyset$
    elija $l_i \in V$ tal que $\bar{l}_i$ no es alcanzable desde l_i en G_φ
    $V := V \setminus \{l_i, \bar{l}_i\}$
    $\sigma(l_i) := 1$
    Para todo l_j tal que l_j es alcanzable desde l_i en G_φ
        $V := V \setminus \{l_j, \bar{l}_j\}$
        $\sigma(l_j) := 1$

La complejidad de 2-CNF-SAT

Primero tenemos que demostrar que σ está bien definida:

- σ asigna un valor de verdad a cada variable x_i.
- No existe una variable x_i tal que el procedimiento asigna 1 tanto a x_i como a $\neg x_i$.

Es claro que el primer punto se cumple (¿por qué?).

La complejidad de 2-CNF-SAT

Segundo punto: Por contradicción, asuma que existe x_i tal que el procedimiento asigna 1 tanto a x_i como a $\neg x_i$.

Entonces existen literales l y l' tales que:

- l y l' fueron elegidos en la tercera linea del procedimiento.
- Existe un camino $l_1, \ldots, l_j$ en G_φ tal que $j \geq 1$, $l_1 = l$ y $l_j = x_i$.
- Existe un camino $l'_1, \ldots, l'_k$ en G_φ tal que $k \geq 1$, $l'_1 = l'$ y $l'_k = \neg x_i$.

La complejidad de 2-CNF-SAT

Supongamos que l' fue elegido primero.

Como $\bar{l}_j, \ldots, \bar{l}_1$ es un camino en G_φ desde $\neg x_i$ a $\bar{l}$: Existe un camino en G_φ desde l' a $\bar{l}$.

Como l' fue elegido primero: el procedimiento asigna 1 a $\sigma(\bar{l})$ y remueve tanto l como $\bar{l}$ desde V.

Por lo tanto: l no es elegido y $\sigma(l) = 0$, una contradicción.

La complejidad de 2-CNF-SAT

Para terminar la demostración, tenemos que mostrar que $\sigma(\varphi) = 1$.

- Sea $C = l_1 \vee l_2$ una cláusula en φ. Tenemos que demostrar que $\sigma(C) = 1$.

Como $(l_1 \vee l_2)$ es una cláusula en φ: $(\bar{l}_1, l_2)$ es un arco en G_φ.

Para cada arco (l, l') en G_φ, el procedimiento:

- asigna 0 a $\sigma(l)$ y $\sigma(l')$ o
- asigna 0 a $\sigma(l)$ y 1 a $\sigma(l')$ o
- asigna 1 a $\sigma(l)$ y $\sigma(l')$.

La complejidad de 2-CNF-SAT

Por lo tanto: En los dos primeros casos $\sigma(l_1) = 1$, y en el último caso $\sigma(l_2) = 1$.

En cualquiera de los tres casos, $\sigma(l_1 \vee l_2) = 1$, lo cual concluye la demostración.

Cláusulas de Horn

Notación

p es un literal positivo y $\neg p$ es un literal negativo.

Una cláusula C es de Horn si C contiene a lo más un literal positivo.

Ejemplo

$$p \vee \neg q \vee \neg r, \qquad p,$$
$$\neg q \vee \neg r, \qquad \neg q.$$

¿Por qué son interesantes las cláusulas de Horn?

Conocimiento definitivo

Una fórmula $(p \wedge q) \rightarrow r$ expresa conocimiento *definitivo*:

- Si sabemos que p y q son verdaderos, entonces también sabemos que r es verdadero.

En cambio, una fórmula $(p \wedge q) \rightarrow (r \vee s)$ no expresa conocimiento definitivo:

- Si sabemos que p y q son verdaderos, no podemos concluir que r es verdadero ni tampoco podemos concluir que s es verdadero, sólo podemos concluir que alguno de los dos es verdadero.
- Esta fórmula expresa conocimiento *disyuntivo*.

Conocimiento definitivo

Las cláusulas de Horn corresponden al siguiente tipo de reglas que expresan conocimiento definitivo:

- q.
- $\neg q$.
- $(p_1 \wedge \cdots \wedge p_n) \rightarrow q$.
- $(p_1 \wedge \cdots \wedge p_n) \rightarrow \neg q$.

En muchos casos es posible modelar el dominio de aplicación usando sólo este tipo de reglas.

La complejidad de HORN-SAT

Sea HORN-SAT $= \{\varphi \mid \varphi$ es una conjunción de cláusulas de Horn y φ es satisfacible$\}$.

Teorema

HORN-SAT puede ser solucionado eficientemente.

Ejercicio

Demuestre el teorema.

La complejidad de NEG-HORN-SAT

C es una cláusula de Horn negativa si C contiene a lo más un literal negativo.

Sea NEG-HORN-SAT $= \{\varphi \mid \varphi$ es una conjunción de cláusulas de Horn negativas y φ es satisfacible$\}$.

Ejercicio

Demuestre que NEG-HORN-SAT puede ser solucionado eficientemente.

Clases de complejidad: Espacio

Para recordar:

Fácil de demostrar	:	$\text{PTIME} \subseteq \text{NP} \subseteq \text{EXPTIME}$.
No tan fácil de demostrar	:	$\text{PTIME} \subsetneq \text{EXPTIME}$.
Aún sin resolver	:	¿$\text{PTIME} \subsetneq \text{NP}$?

¿Nos basta con estas clases?

- No, en algunos casos también es necesario utilizar clases definidas en término del espacio utilizado.

Espacio utilizado en una Máquina de Turing

Para introducir la noción de espacio utilizado en una MT tenemos que distinguir entre:

- el espacio ocupado por la entrada, y
- el espacio necesario para procesar la entrada.

Para hacer esta distinción utilizamos MT con dos cintas:

- Cinta para la entrada: Sólo de lectura.
- Cinta de trabajo: Como en una MT usual.

El espacio utilizado se mide en términos de las celdas visitadas en la cinta de trabajo.

Máquinas de Turing con cinta de trabajo

Definición

Máquina de Turing con cinta de trabajo: (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* $(F \neq \emptyset)$.
- δ *es una función parcial:*

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

Suponemos que el dominio de δ *no es vacío.*

Espacio utilizado en una MT

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

- $espacio_M(w)$: Número de celdas distintas visitadas en la cinta de trabajo de M al procesar w.

Definición

Dado un número natural n:

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

s_M: Espacio utilizado en la ejecución de M en el peor caso.

Espacio utilizado para resolver un problema

Definición

Un lenguaje L puede ser aceptado en espacio s si es que existe una MT determinista M tal que:

- *M para en todas las entradas.*
- *$L = L(M)$.*
- *s_M es $O(s)$.*

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

- En este caso, usualmente se supone que la máquina tiene una tercera cinta donde escribe el resultado. El espacio utilizado se mide sólo considerando las celdas en la cinta de trabajo.

Clases de complejidad: Espacio

Definición

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

Tres clases fundamentales:

$$\begin{aligned} \text{LOGSPACE} &= \text{DSPACE}(\log n) \\ \text{PSPACE} &= \bigcup_{k \in \mathbb{N}} \text{DSPACE}(n^k) \\ \text{EXPSPACE} &= \bigcup_{k \in \mathbb{N}} \text{DSPACE}(2^{n^k}) \end{aligned}$$

Relación entre clases de complejidad

¿Cuál es la relación entre las clases de complejidad que acabamos de definir?

La relación es más compleja:

$$\text{LOGSPACE} \subseteq \text{PTIME} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{EXPSPACE}.$$

También se sabe que: $\text{LOGSPACE} \subsetneq \text{PSPACE} \subsetneq \text{EXPSPACE}$.

Y todavía nos faltan las clases no deterministas ...

Clases de complejidad no deterministas: Espacio

Notación

Dada una MT no determinista M con alfabeto Σ:

- $espacio_M(w)$: *Número mínimo de celdas distintas visitadas en la cinta de trabajo de M, entre todas las ejecuciones de M que aceptan w.*

  Sólo esta definido para palabras aceptadas por M.

Clases de complejidad no deterministas: Espacio

Definición

Dado un número natural n:

$$s_M(n) = \text{máx}\{1\} \cup \{ \textit{espacio}_M(w) \mid w \in \Sigma^*, |w| = n \text{ y } M \textit{ acepta } w\}.$$

¿Por que incluimos $\{1\}$ en la definición?

Clases de complejidad no deterministas: Espacio

Definición

Un lenguaje L es aceptado en espacio s por una MT M no determinista si:

- $L = L(M)$.
- s_M *es* $O(s)$.

Clases de Complejidad no deterministas: Espacio

Definición

NSPACE(s) se define como el conjunto de todos los lenguajes que pueden ser aceptados en espacio s por alguna MT no determinista.

Dos clases fundamentales:

$$\begin{aligned} \text{NLOGSPACE} &= \text{NSPACE}(\log n) \\ \text{NPSPACE} &= \bigcup_{k \in \mathbb{N}} \text{NSPACE}(n^k) \end{aligned}$$

Relación entre clases de complejidad

¿Cuál es la relación entre las clases de complejidad que definimos?

Se sabe que: $\text{PSPACE} = \text{NPSPACE}$.

La relación final:

$$\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{PTIME} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{EXPSPACE}$$

Para terminar: Un poco de intuición

Ejercicio

Determine en qué clases de complejidad viven los siguientes problemas.

1. Dado un grafo dirigido G y nodos a, b, determinar si existe un camino de a a b.
2. El mismo problema que en 1. pero ahora suponiendo que G es una relación de sucesor.
3. Determinar si un conjunto de cláusulas de Horn es satisfacible.