

Clases de complejidad de lenguajes decidibles

Semana (5)₂ = 101

Lógica para Ciencia de la
Computación - IIC2213

Prof. Sebastián Buggedo

Programa

Obertura

Primer acto

Complejidad de algoritmos
Clases P y EXP

Intermedio

Segundo acto

Reducciones polinomiales
Problemas completos

Epílogo

Programa

Obertura

Primer acto

Complejidad de algoritmos
Clases P y EXP

Intermedio

Segundo acto

Reducciones polinomiales
Problemas completos

Epílogo

¿Cómo están?





La noción de reducción

Definición en chileno, 2.0

Una **reducción** transforma un **lenguaje** L_A en un **lenguaje** L_B tal que si encontramos un algoritmo para **B** entonces tenemos un algoritmo para **A**

L_A se reduce a L_B

Si tengo un algoritmo para resolver L_B entonces tengo un algoritmo para resolver L_A .

Reducciones de mapeo

Definición (reducciones de mapeo)

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

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

En tal caso, llamamos a f una **reducción de mapeo** o **many-one** de L_1 a L_2 y denotamos

$$L_1 \leq_m L_2$$

Intuición: L_2 es al menos tan difícil como L_1

Relaciones entre lenguajes

La herramienta de reducción nos permite estudiar los lenguajes como objetos

Para formalizar esta idea, nos centraremos en colecciones de lenguajes

Llamaremos **clase de complejidad** a un conjunto de lenguajes que comparten algún criterio

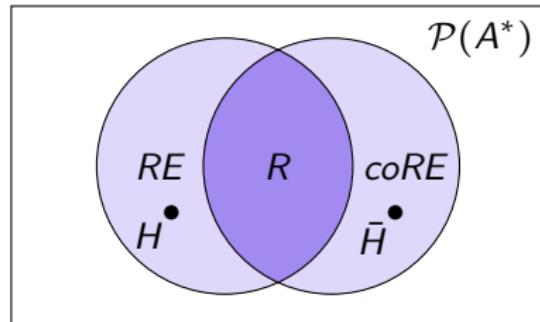
Clasificaremos clases según su complejidad relativa

El panorama hasta ahora

Lenguajes **RE**: aceptados por alguna máquina

Lenguajes **coRE**: su complemento es aceptado por alguna máquina

Lenguajes **R** o **decidibles**: aceptados por alguna máquina que siempre se detiene



Desde hoy nos adentraremos en la clase R

Vamos adentrándonos en R

Ejercicio

Demuestre que para lenguajes L_1, L_2 decidibles se tiene

1. $L_1 \cap L_2$ es decidable
2. $L_1 \cup L_2$ es decidable
3. $\overline{L_1}$ es decidable



¿Por qué nos concentramos en la clase R?

Acordamos que

Resolver un problema \equiv Tener una MT \mathcal{M} que siempre para

Los problemas decidibles son aquellos que
efectivamente podemos resolver

Complejidad

¿Qué es la complejidad de un problema?

- ¿Cómo medimos complejidad?
- ¿Qué diferencia la complejidad de un problema y de un algoritmo?
- ¿Cómo abstraernos de una máquina en particular?
- ¿Conviene usar $\mathcal{O}(\dots)$ para hablar de complejidad de un problema?

Hoy abordaremos estas preguntas

Playlist Unidad II y Orquesta



Playlist: LogiWawos #2

Además sigan en instagram:

@orquesta_tamen

Objetivos de la clase

- Definir la complejidad de un algoritmo
- Definir la complejidad de un lenguaje/problema
- Comprender las primeras clases de complejidad en R
- Demostrar que SAT está en la clase EXP
- Definir las reducciones polinomiales
- Comprender el concepto de hardness y completitud

Programa

Obertura

Primer acto

Complejidad de algoritmos
Clases P y EXP

Intermedio

Segundo acto

Reducciones polinomiales
Problemas completos

Epílogo

Complejidad de algoritmos

Nuestro primer desafío será formalizar **qué tan complejo** es un algoritmo

- Por la hipótesis Church-Turing, esto equivale a analizar máquinas
- Nos centraremos en **cuánto demoran** en ejecutarse
- Naturalmente, asumiremos que **siempre se detienen**

A este enfoque le llamamos **complejidad de tiempo**

En este curso: solo veremos complejidad de tiempo

Recordatorio: Máquinas de Turing

Definición

Un **máquina de Turing determinista** (TM) es una estructura

$$\mathcal{M} = (Q, A, q_0, F, \delta)$$

- Q es un conjunto finito de **estados**
- A es el alfabeto de **input**, tal que $\sqcup \notin A$ está reservado
- q_0 es el estado **inicial**
- $F \subseteq Q$ es un conjunto finito de **estados finales** ($F \neq \emptyset$)
- $\delta : Q \times (A \cup \{\sqcup\}) \rightarrow Q \times (A \cup \{\sqcup\}) \times \{\leftarrow, \rightarrow\}$ es la **función parcial de transición**

Recordatorio: configuraciones

Definición (configuraciones)

Decimos que la configuración

$$u \cdot q \cdot v \in A_{\sqcup}^* \cdot Q \cdot A_{\sqcup}^*$$

- es de **detención** si $\delta(q, a)$ no está definido, con $v = a \cdot v'$
- es de **aceptación** si es de detención y $q \in F$
- es de **rechazo** si es de detención y $q \notin F$

Una configuración resume el estado **actual** de la máquina

Recordatorio: pasos

Definición (pasos)

Sean $a, b, c \in A$ y $u, v \in A^*$. Se define la relación de **siguiente paso**

$$\xrightarrow{\mathcal{M}} \subseteq (A^* \cdot Q \cdot A^*) \times (A^* \cdot Q \cdot A^*)$$

- Si $\delta(p, b) = (q, c, \triangleleft)$, entonces

$$u \cdot a \cdot \mathbf{p} \cdot b \cdot v \xrightarrow{\mathcal{M}} u \cdot \mathbf{q} \cdot a \cdot c \cdot v$$

- Si $\delta(p, b) = (q, c, \triangleright)$, entonces

$$u \cdot \mathbf{p} \cdot b \cdot v \xrightarrow{\mathcal{M}} u \cdot c \cdot \mathbf{q} \cdot v$$

Un paso es un cambio de configuración **válido** según δ

Recordatorio: ejecuciones

Definición (ejecuciones)

Sea $\mathcal{M} = (Q, A, q_0, F, \delta)$ una MT y $w \in A^*$

- Una **ejecución** ρ de \mathcal{M} es una secuencia de configuraciones

$\rho : C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow \dots$ (no necesariamente finita)

tal que $C_i \xrightarrow{\mathcal{M}} C_{i+1}$ para todo C_i y C_{i+1} en ρ .

En el análisis de algoritmos, supondremos que toda ejecución es finita

Recordatorio: ejecuciones

Definición (ejecuciones)

Sea $\mathcal{M} = (Q, A, q_0, F, \delta)$ una MT y $w \in A^*$

- Decimos que $\rho : C_0 \rightarrow \dots \rightarrow C_m$ es **la ejecución** de \mathcal{M} sobre w si:
 - ρ es una ejecución de \mathcal{M}
 - $C_0 = q_0 \cdot w$
 - Su última conf. C_m es de **detención**
- Si C_m es de aceptación, entonces ρ es una **ejecución de aceptación** para w . Decimos que \mathcal{M} **acepta** w

Todo input w tiene una única ejecución en \mathcal{M}

El concepto de paso

Notación

Dada una MT determinista \mathcal{M} con alfabeto A que **se detiene en toda entrada**, llamamos

- **Paso de \mathcal{M} en w** a un par (C_i, C_{i+1}) en una ejecución de \mathcal{M} sobre el input w
- **Tiempo de \mathcal{M} en w** al número de pasos ejecutados por \mathcal{M} con entrada w . Lo denotamos por

$$\text{tiempo}_{\mathcal{M}}(w) = |\{(C_i, C_{i+1}) \mid (C_i, C_{i+1}) \text{ es un paso de } \mathcal{M} \text{ en } w\}|$$

¿Por qué es importante que \mathcal{M} se detenga siempre?

El concepto de paso

Para todo par \mathcal{M} y w tal que \mathcal{M} se detiene,

- $\text{tiempo}_{\mathcal{M}}(w) \geq 1$ (configuración inicial y detención)
- $\text{tiempo}_{\mathcal{M}}(w)$ es finito (\mathcal{M} se detiene en w)

Ahora nos abstraeremos del input particular

Tiempo de ejecución de una máquina

Definición (tiempo de ejecución)

Sea \mathcal{M} una MT determinista con alfabeto A que se detiene en todo input. Dado un natural n , definimos el **tiempo de ejecución de \mathcal{M} en el peor caso** como

$$t_{\mathcal{M}}(n) = \max\{\text{tiempo}_{\mathcal{M}}(w) \mid w \in A^* \text{ y } |w| = n\}$$

Caracterizamos el desempeño de \mathcal{M} según el tamaño de sus inputs

Complejidad de un problema

Definición (tiempo de aceptación)

Decimos que un lenguaje L **puede ser aceptado en tiempo g** , si existe una MT determinista \mathcal{M} tal que

- \mathcal{M} se detiene en todas las entradas
- $L = L(\mathcal{M})$
- $t_{\mathcal{M}} \in \mathcal{O}(g)$

Recordemos que

$$t_{\mathcal{M}} \in \mathcal{O}(g) \Leftrightarrow \exists c \in \mathbb{R}^+. \exists n_0 \in \mathbb{N}. \forall n \geq n_0. t_{\mathcal{M}}(n) \leq c \cdot g(n)$$

Esto define la complejidad de un problema,
más allá de algoritmos particulares que lo resuelven

Complejidad de una función

Definición (tiempo de computación)

Decimos que una función f **puede ser computada en tiempo g** , si existe una MT determinista \mathcal{M} que calcula f y $t_{\mathcal{M}} \in \mathcal{O}(g)$

Esto será clave para usar reducciones dentro de la clase R

Clases de complejidad de tiempo

Definición (clases de tiempo)

Dado un alfabeto A , se define

$$\text{DTIME}(g) = \{L \subseteq A^* \mid L \text{ puede ser aceptado en tiempo } g\}$$

Dos clases fundamentales son

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

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

¿Conocemos algún problema en estas clases?

Programa

Obertura

Primer acto

Complejidad de algoritmos
Clases P y EXP

Intermedio

Segundo acto
Reducciones polinomiales
Problemas completos

Epílogo

P y la noción de eficiencia

Consideraremos a la clase

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

como la clase de problemas que pueden ser **solucionados eficientemente**

¿Significa que todo algoritmo polinomial es rápido? **NO!**

- Recordemos que $\mathcal{O}(n^k)$ es una medida de crecimiento asintótico
- En aplicaciones prácticas, polinomios muy grandes son muy costosos

Pueden haber algoritmos exponenciales que en la práctica sean más rápidos que algoritmos polinomiales para el mismo input

Nuestra misión

El problema fundamental de los computines

Dado un problema, encontrar un algoritmo eficiente para resolverlo **O**
demostrar que tal algoritmo no existe

Llamaremos **tratables** a los problemas que se pueden resolver en un
tiempo *razonable*

Consideraremos a la clase P como **tratable**

P y la noción de eficiencia

El problema fundamental de los computines

Dado un problema, encontrar un algoritmo eficiente para resolverlo O
demostrar que tal algoritmo no existe

¿Cómo determinamos si $L \in \text{DTIME}(g)$?

- Probar que $L \in \text{DTIME}(g)$ puede ser difícil
- Probar que $L \notin \text{DTIME}(g)$ **es difícil**

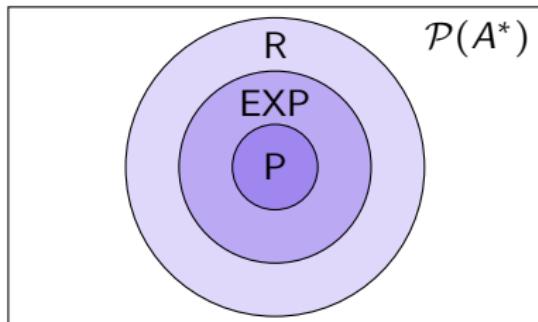
¿SAT $\in P$?

P y EXP

Antes de analizar el caso de SAT, pensemos en la relación de P y EXP

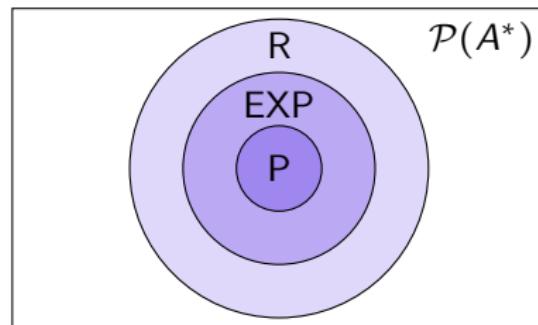
Teorema

Las clases P y EXP cumplen que $P \subseteq EXP$



La contención es **estricta**:
tomen el curso Complejidad Computacional para más info jj

P y EXP



¿Conocemos algún problema que esté en EXP?

¿Dónde vive SAT?

Proposición

$SAT \in EXP$

Para demostrar esto necesitamos un resultado intermedio

Proposición

El siguiente lenguaje pertenece a la clase P

$$LP-EVAL = \{(\sigma, \varphi) \mid \varphi \in \mathcal{L}(P) \text{ y } \sigma(\varphi) = 1\}$$

¿Dónde vive SAT?

Proposición

El siguiente lenguaje pertenece a la clase P

$$\text{LP-EVAL} = \{(\sigma, \varphi) \mid \varphi \in \mathcal{L}(P) \text{ y } \sigma(\varphi) = 1\}$$

Demostración



Pato profe

¿Dónde vive SAT?

Demostración

Sean un conjunto $P = \{p_1, \dots, p_n\}$, una fórmula $\varphi \in \mathcal{L}(P)$ dada como string $|\varphi| = m$ y una valuación σ

En la clase 02 vimos que la evaluación se puede hacer mediante

- Reemplazo de cada variable p_i por $\sigma(p_i)$
- $\mathcal{O}(m)$ pasos para simplificar los conectivos que tienen constantes
- $\mathcal{O}(m)$ repeticiones de dicho proceso hasta obtener $\sigma(\varphi)$

Esto nos daba un algoritmo $\mathcal{O}(m^2)$ para evaluar una fórmula en una valuación dada

¿Dónde vive SAT?

Demostración

El algoritmo de evaluación era a alto nivel: su representación en máquina de Turing involucra

- Convertir operaciones en una serie de pasos en MT
- Cada conversión involucra una cantidad $\mathcal{O}(1)$ de pasos

La máquina de Turing calculadora toma $\mathcal{O}(m^2)$ pasos en ejecutarse

Esta máquina permite decidir el lenguaje LP-EVAL en tiempo polinomial

- Dada una fórmula y valuación, ejecutar la máquina
- Si el resultado es 1, aceptar
- Si no, rechazar



¿Dónde vive SAT?

Proposición

$SAT \in EXP$

Demostración



Pato profe

¿Dónde vive SAT?

Demostración

El análisis de complejidad se hace respecto al largo de la fórmula.

Notemos que dada φ con largo $|\varphi| = m$ en su codificación, se tienen en el peor caso una cantidad $\mathcal{O}(m)$ de variables proposicionales distintas.

Ahora, disponemos de máquinas

- \mathcal{M} para generar las 2^m posibles valuaciones
- \mathcal{N} para evaluar φ dada en una valuación σ

¿Dónde vive SAT?

Demostración

Con esto, la operación de una máquina \mathcal{D} que decida SAT es

1. Generar valuación σ con \mathcal{M}
2. Evaluar φ en σ con \mathcal{N}
3. Si la evaluación da 1, aceptar
4. Si no, y quedan valuaciones, volver a 1.
5. Si no quedan, rechazar

¿Dónde vive SAT?

Demostración

La máquina \mathcal{D} ejecuta $\mathcal{O}(2^m)$ veces la evaluación

- La evaluación es polinomial, $\mathcal{O}(m^2)$
- Total: $\mathcal{O}(m^2 2^m)$

Simplificando, se puede encontrar un k adecuado tal que el algoritmo tiene complejidad $\mathcal{O}(2^{m^k})$. Con esto, la máquina \mathcal{D} toma tiempo exponencial.

Concluimos que $SAT \in EXP$.



¿Esto significa que $SAT \notin P$?

Programa

Obertura

Primer acto

Complejidad de algoritmos
Clases P y EXP

Intermedio

Segundo acto
Reducciones polinomiales
Problemas completos

Epílogo

Intermedio: Repartición de huevitos



Programa

Obertura

Primer acto

Complejidad de algoritmos
Clases P y EXP

Intermedio

Segundo acto

Reducciones polinomiales
Problemas completos

Epílogo

Reducciones polinomiales

Definición (reducciones de mapeo polinomial)

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

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

En tal caso, llamamos a f una **reducción polinomial** de L_1 a L_2 y denotamos

$$L_1 \leq_p L_2$$

Este sabor de reducción nos permite analizar mejor la clase P

Reducciones polinomiales

Teorema

Sean L_1 y L_2 lenguajes tales que $L_1 \leq_p L_2$.

1. Si $L_2 \in P$, entonces $L_1 \in P$
2. Si $L_1 \notin P$, entonces $L_2 \notin P$

El teorema nos da una forma de demostrar que L_2 no está en P

Demostración (1.)



Reducciones polinomiales

Demostración (1.)

Sean L_1, L_2 tales que $L_1 \leq_p L_2$, es decir, existe una reducción polinomial f . Sea \mathcal{F} la máquina que computa f en tiempo polinomial.

Suponemos además que $L_2 \in P$, es decir, existe \mathcal{M} tal que

- se detiene siempre
- acepta L_2
- funciona en tiempo polinomial para todo input

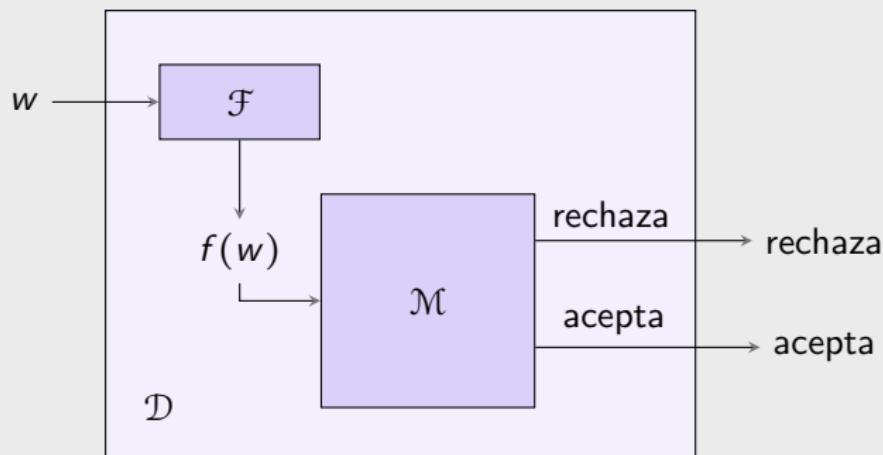
Proponemos una máquina para decidir L_1 en tiempo polinomial.

Reducciones polinomiales

Demostración (1.)

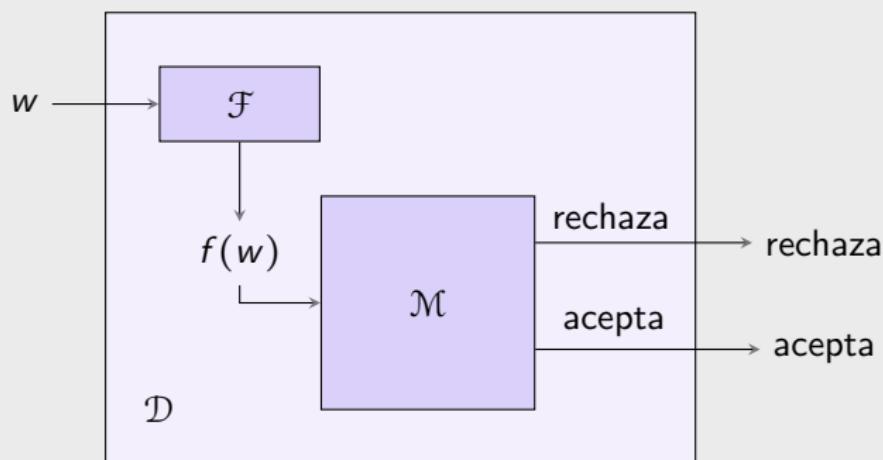
Sea \mathcal{D} una máquina tal que para todo input w

1. Ejecuta \mathcal{F} con entrada w para obtener $f(w)$
2. Simula \mathcal{M} con input $f(w)$
3. Responde igual que \mathcal{M}



Reducciones polinomiales

Demostración (1.)



Probaremos que la máquina \mathcal{D} cumple

1. $L_1 = L(\mathcal{D})$
2. se detiene en todo input
3. opera en tiempo polinomial

Reducciones polinomiales

Demostración

1. $L_1 = L(\mathcal{D})$

$$\begin{aligned} w \in L_1 &\Leftrightarrow f(w) \in L_2 \quad (\text{prop. reducción}) \\ &\Leftrightarrow \mathcal{M} \text{ acepta} \quad (\mathcal{M} \text{ acepta } L_2) \\ &\Leftrightarrow \mathcal{D} \text{ acepta} \quad (\text{def. } \mathcal{D}) \\ &\Leftrightarrow w \in L(\mathcal{D}) \quad (\text{def. de aceptación}) \end{aligned}$$

2. La máquina \mathcal{F} se detiene en todo input pues es máquina que calcula. Además, por hipótesis \mathcal{M} se detiene en todo input. Por lo tanto, \mathcal{D} también lo hace.
3. Dado que \mathcal{F} computa f en tiempo polinomial, y \mathcal{M} decide L_2 en tiempo polinomial, obtenemos lo pedido para \mathcal{D} .

Concluimos que $L_1 \in \mathsf{P}$.



Construyendo reducciones polinomiales

En el curso ya hemos construido reducciones polinomiales

$$\begin{aligned}3\text{COL} &= \{G \mid G \text{ grafo no dirigido 3-coloreable}\} \\ \text{SAT} &= \{\varphi \mid \varphi \text{ es satisfacible}\}\end{aligned}$$

En la clase 02 propusimos cómo construir una fórmula φ_G a partir del grafo G tal que

$$G \in 3\text{COL} \iff \varphi_G \in \text{SAT}$$

y que se construye en tiempo polinomial (¿por qué??)

Esta construcción atestigua $3\text{COL} \leq_p \text{SAT}$

¿Podemos hacer una reducción en el sentido contrario?

Construyendo reducciones polinomiales

Consideraremos un problema relacionado

$$3\text{SAT} = \{\varphi \mid \varphi \text{ está en 3-CNF y es satisfacible}\}$$

donde 3-CNF significa que la fórmula tiene a lo más tres literales por cláusula

Ejercicio

Demuestre que $3\text{SAT} \leq_p 3\text{COL}$



¿Para qué las reducciones polinomiales?

Tenemos una herramienta ad hoc para relacionar problemas dentro de la clase P y alrededores

Nos preguntamos...

*¿Qué problema **representa** a una clase?*

*¿Qué problema **es el más difícil** de una clase?*

Las reducciones polinomiales eran el ingrediente faltante para formalizar esto!

Programa

Obertura

Primer acto

Complejidad de algoritmos
Clases P y EXP

Intermedio

Segundo acto

Reducciones polinomiales
Problemas completos

Epílogo

Dificultad relativa

No olvidemos una de nuestras preguntas difíciles

¿Un lenguaje L está en P?

Dificultad relativa

Comenzamos definiendo una noción de dificultad relativa en una clase

Definición (hardness)

Dada una clase de complejidad \mathcal{C} que contiene a P, decimos que un lenguaje L es \mathcal{C} -hard si

$$\text{para todo } L' \in \mathcal{C}, \quad L' \leq_p L$$

Un lenguaje \mathcal{C} -hard es al menos tan difícil como todo problema de \mathcal{C}

Dificultad exacta

Pero también nos interesa saber la complejidad **exacta** de un problema

Definición (completitud)

Dada una clase de complejidad \mathcal{C} que contiene a P, decimos que un lenguaje L es **\mathcal{C} -completo** si

- $L \in \mathcal{C}$
- L es \mathcal{C} -hard



Los problemas completos son **representantes** de su clase

Dificultad exacta

Teorema

Si $P \not\subseteq \mathcal{C}$ y L es \mathcal{C} -hard, entonces $L \notin P$

Tenemos una forma de determinar que un lenguaje no es polinomial!

Demostración

Sea \mathcal{C} tal que $P \subseteq \mathcal{C}$ y L que es \mathcal{C} -hard. Supongamos que $L \in P$.

Por hipótesis, para cualquier $L' \in \mathcal{C}$, $L' \leq_p L$. Notemos que como $L' \in \mathcal{C}$, en particular $L' \notin P$.

Por teorema anterior de reducciones, tenemos que como $L' \leq_p L$ y $L \in P$, entonces $L' \in P$. ¡Contradicción! □

Dificultad exacta

Corolario

Si $P \not\subseteq \mathcal{C}$ y L es \mathcal{C} -completo, entonces $L \notin P$

¿Conocemos una clase que contenga de forma estricta a P ?

Como $P \not\subseteq EXP$ y probamos que $SAT \in EXP\dots$

... basta probar que SAT es EXP -hard y concluimos que $SAT \notin P$

Dificultad exacta

**NADIE SABE CÓMO HACER
ESTO AYUDAA**



PUCHA LA CUSTIÓN

memegenerator.es

Hacia dónde vamos

Nos daremos un rodeo...

... en lugar de comparar SAT con P, busquemos su clase

¿Qué clase representa la complejidad de SAT?

¿Para qué clase SAT es completo?

¿Esa supuesta clase es distinta de P?

Próxima clase: no determinismo y la clase NP

Programa

Obertura

Primer acto

Complejidad de algoritmos
Clases P y EXP

Intermedio

Segundo acto

Reducciones polinomiales
Problemas completos

Epílogo

¿Qué aprendí hoy? ¿Comentarios?

Ve a

www.menti.com

Introduce el código

8619 4617



O usa el código QR