COMPUTABILIDAD Y COMPLEJIDAD

Belén Pérez Lancho
Dpto. Informática y Automática
12-Enero-2006

Indice

Introducción

Conceptos Elementales

- *Alfabeto*: Conjunto finito de símbolos $\Sigma=\{a_1, \ldots, a_n\}$
- *Cadena o palabra:* Secuencia de símbolos $w \in \Sigma^*$
- *Lenguaje*: Conjunto de cadenas sobre un alfabeto $L \subseteq \Sigma^*$, $L=\{w_1, \ldots, w_n, \ldots\}$
- *Problema*: Pregunta precisa (*predicado lógico*) sobre un conjunto de elementos (*dominio*) $p: A \rightarrow B$
- *Problema de decisión*: Pregunta de respuesta binaria $d: A \rightarrow \{0, 1\}$

Introducción

Estudio formal de un problema

- *Representación*: asignación de una cadena a cada elemento del dominio $R: A \rightarrow \Sigma^*$
- *Resolución*: secuencia de acciones de manipulación de las cadenas $f: \Sigma^* \rightarrow \Sigma^*$
- *Interpretación* de los resultados o cadenas finales $I: \Sigma^* \rightarrow B$

Computación ≈ Resolución de problemas

Lenguaje formal ≈ Representación de un problema de decisión

Modelos de computación

Modelos de computación

- Determinista
  - Máquina de Turing
  - Funciones Recursivas
- No determinista
  - Máquina de Turing no determinista
- Otros
  - Paralelos
  - Cuánticos
  - …

Máquinas de Turing

La Máquina de Turing (MT)

Modelo de proceso mental algorítmico (Alan Turing, 1936)

Propuesto antes de la aparición de los computadores

Proceso mental	***Modelo (Máquina de Turing)***
Conjunto de símbolos (finito)	Alfabeto
Papel (ilimitado) y lápiz	Cinta (ilimitada) y cabeza
Estados mentales (finitos)	Estados
Acciones: → observar → sustituir → transferir atención	Transiciones: → leer de la cinta → escribir/cambiar estado → mover cabeza (R/L)

Máquinas de Turing

Definición Formal $M = (\Sigma, \Gamma, \#, Q, q_0, f, F)$

Σ	Alfabeto de la máquina
Γ	Alfabeto de cinta
$\# \in \Gamma$	Espacio blanco (casilla vacía, $\#\notin\Sigma$)
Q	Conjunto finito de estados
$q_0 \in Q$	Estado inicial
f	Función de transición $f: Q \times \Gamma \to Q \times \Gamma \times \{R, L\}$
$F \subseteq Q$	Conjunto de estados finales o de **parada**

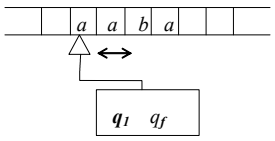
*El comportamiento es **determinista***

Máquinas de Turing

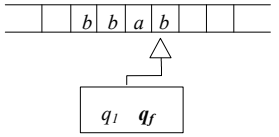
Ejemplo $M = (\{a, b\}, \{a, b, \#\}, \#, \{q_1, q_f\}, q_1, f, \{q_f\})$

f	a	b	$\#$
$\to q_1$	$q_1\ b\ R$	$q_1\ a\ R$	$q_f \# L$

Inicio (dato de entrada)



Parada (dato de salida)



Máquinas de Turing

Más definiciones

- *Computación:* Secuencia de movimientos que llevan a configuración de parada
  $(q_0, w) \vdash^* (q_f, w')$ con $qf \in F$
- *Bucle infinito:* Situación de la MT que nunca para
  $(q_0, w) \vdash \infty$
- *Máquina de Turing no determinista:* Su operación viene definida por una *relación* de transición $R \subseteq Q \times \Gamma \to Q \times \Gamma \times \{R, L\}$
- *Función Turing computable:* Cualquier $f: \Sigma^* \to \Sigma^*$ para la que existe una MT capaz de computar todas las cadenas $w \in \Sigma^*$ para las que $f(w)$ está definida, transformándolas en $w' = f(w)$
  $(q_0, w) \vdash^* (q_f, f(w))$

Funciones Recursivas

Funciones Recursivas

Enfoque funcional de los problemas

Problema $\Rightarrow$ *Función* $f: N^n \to N^m$

Dominio y rango $\Rightarrow$ *Números naturales*

Construcción $\Rightarrow$ ***funciones iniciales + operaciones***

Funciones iniciales

- Cero $\zeta: N^0 \to N$ $\zeta() = 0$
- Sucesor $\sigma: N \to N$ $\sigma(x) = x + 1$
- Proyecciones $\pi^n_i: N^n \to N$ $\pi^n_i(x_1, x_2, \ldots x_n) = x_i$

Son totales y computables

Funciones Recursivas

Operaciones básicas

- Combinación $f \times g : N^n \to N^{m+k}$
  Se define a partir de $f: N^n \to N^m$ y $g: N^n \to N^k$ como:
  $f \times g\ (x_1, x_2, \ldots x_n) = (f(x_1, x_2, \ldots x_n), g(x_1, x_2, \ldots x_n))$
- Composición $g \cdot f : N^n \to N^k$
  Se define a partir de $f: N^n \to N^m$ y $g: N^m \to N^k$ como:
  $g \cdot f\ (x_1, x_2, \ldots x_n) = g\ (f\ (x_1, x_2, \ldots x_n))$

Funciones Recursivas

- Recursividad primitiva $h: N^{n+1} \to N^m$
  Se define a partir de $f: N^n \to N^m$ y $g: N^{n+m+1} \to N^m$ como:
  $h(x_1, x_2, \ldots x_n, 0) = f(x_1, x_2, \ldots x_n)$
  $h(x_1, x_2, \ldots x_n, y+1) = g(x_1, x_2, \ldots x_n, y, h(x_1, x_2, \ldots x_n, y))$
- Minimalización $\mu: N^n \to N$
  A partir de $f: N^{n+1} \to N$ es $\mu(x_1, x_2, \ldots x_n) = y$ si se cumple que:
  - $f(x_1, x_2, \ldots x_n, y) = 0$
  - $\forall x_{n+1} < y \ \exists f(x_1, x_2, \ldots x_n, x_{n+1}) \ \&\ f(x_1, x_2, \ldots x_n, x_{n+1}) \neq 0$

  (y es el mínimo valor que hace $f(x1, x2, \ldots xn, y) = 0$, estando f definida para todos los valores anteriores)

Funciones Recursivas

Definiciones

Funciones recursivas primitivas

- Conjunto de funciones $f: \mathbb{N}^n \rightarrow \mathbb{N}^m$ que pueden obtenerse a partir de las *iniciales* usando un número finito de operaciones de *composición*, *combinación* y/o *recursividad primitiva*.
- Son funciones *totales*

Funciones recursivas parciales

- Conjunto de funciones $f: \mathbb{N}^n \rightarrow \mathbb{N}^m$ que pueden obtenerse de las *iniciales* usando un número finito de operaciones de *composición*, *combinación, recursividad primitiva* y/o *minimalización*
- La *minimalización* permite definir funciones *estrictamente parciales*

Conclusiones interesantes

Teoremas

- Toda función *recursiva parcial* es *Turing-computable*
- Todo proceso computacional realizado por una máquina de Turing es el cálculo de una función *recursiva parcial*

$\Rightarrow$ *{fs. rec. parciales} = {fs. Turing computables}*

Son MODELOS EQUIVALENTES

Conclusiones interesantes

Tesis de Church-Turing

- Cualquier *función computable* es una función *recursiva parcial*
- Cualquier *proceso computacional* puede ser realizado por una *Máquina de Turing*

$\Rightarrow$ *NO son teoremas,*
son afirmaciones NO DEMOSTRADAS NI REFUTADAS

Conclusiones interesantes

Límites de la Computabilidad

- Las máquinas de Turing se pueden codificar como cadenas $\Rightarrow$ Son un conjunto infinito numerable

$$|\{MT\}| = |\aleph|$$

- Los problemas de decisión $d: \Sigma^* \rightarrow \{0, 1\}$ (conjunto de las partes de Σ^*) forman un conjunto infinito no numerable

$$|\{d\}| = 2^{|\aleph|} > |\{MT\}|$$

$$|\{\text{problemas de decisión}\}| > |\{\text{algoritmos}\}|$$

$\Rightarrow$ *Hay infinidad de problemas NO COMPUTABLES*

Conclusiones interesantes

Lenguaje Esencial o Mínimo

Es el mínimo lenguaje de programación suficiente para expresar cualquier función recursiva parcial $f: \mathcal{N}^n \rightarrow \mathcal{N}^m$

Tipo de datos	`enteros`
Sentencias	`incr var`
	`decr var`
	`while var≠0 do`
...	
	`end`

⇒ *Ningún lenguaje de programación es más potente que éste*

⇒ *mismo LIMITE COMPUTACIONAL para el software*

Ejemplos

Lagarias (1985)

¿El algoritmo de Lagarias se detiene para cualquier número positivo de entrada?

```
Entrar X
Mientras X≠1 hacer:
        Si X es par, X = X/2
        Si no, X = 3X +1
Parar
```

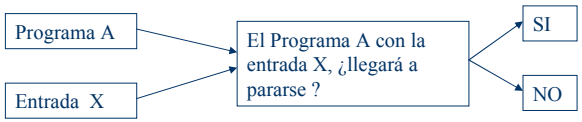
Si X es 7 los valores son: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Se trata de un problema de decisión ***no computable***

Ejemplos

Problema de Parada

Dado un programa (o algoritmo o Máquina de Turing) A y un valor de la entrada X, ¿podemos saber siempre si A se parará o no?



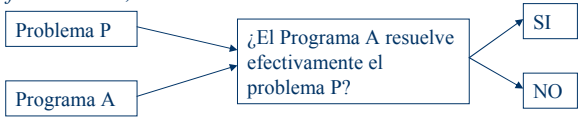
No hay manera de saber, en general y en un tiempo finito, si la ejecución de un programa dado con una entrada dada terminará o no.

Es un problema de decisión ***no computable***

Ejemplos

El problema de la verificación

Dada la descripción formal de un problema, P, y un programa A que supuestamente lo resuelve, ¿es posible construir un programa que verifique si A resuelve, efectivamente, P ?

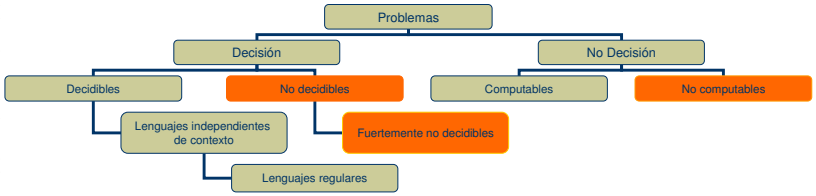


Este problema es mucho más general que el de la parada

Es un problema de decisión ***fuertemente no computable***

Límites de la Computabilidad

Resumiendo …



Indice

Introducción a la Complejidad

Teoría de la complejidad computacional

- Estudio de los recursos necesarios para resolver los problemas teóricamente computables
- Clasificación de los problemas computables y los algoritmos en términos de su "*coste*" o posibilidad de *resolución efectiva*

Introducción a la Complejidad

Definición de complejidad

Cantidad de *recursos* necesarios para la resolución de:

- un problema,
- un algoritmo o
- un cálculo

Se analizan dos recursos principales:

- *Tiempo* (pasos, operaciones elementales, ...) — *Complejidad temporal*
- *Espacio* (celdas, posiciones de memoria, ...) — *Complejidad espacial*

Esta definición *depende* del sistema computacional ⇒ Se considera como sistema de referencia la Máquina de Turing

Introducción a la Complejidad

Complejidad de un cálculo

Se refiere a una MT y una entrada concretas

Son valores numéricos *ct*, *ce*

Complejidad de un algoritmo

Para una cierta MT compute un dato de entrada genérico

Son funciones *t(n)* y s*(n)* de la longitud del dato *n* en el *peor caso*

Complejidad un problema

Se definiría como la complejidad de su *algoritmo más sencillo* (¿?)

Normalmente se establece una cota superior, un valor medio o se delimita una clase de funciones (*tasa u orden de crecimiento*)

Introducción a la Complejidad

Órdenes de complejidad

- constante $f(n) = K$
- logarítmica $f(n) \in O(\log n)$
- lineal $f(n) \in O(n)$
- polinómica $f(n) \in O(n^p)$
- exponencial $f(n) \in O(2^n)$
- factorial $f(n) \in O(n!)$
- …

*los algoritmos se consideran **efectivos** o **tratables** hasta*
Complejidad polinómica

Introducción a la Complejidad

Reducciones en tiempo polinómico

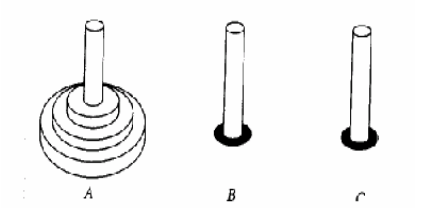
Son algoritmos de transformación entre problemas que requieren tiempo polinómico

- Generales entre sí — operaciones de *división* y *cuadrado*
- Generales a decisión — *factorización y primos*
- Decisión entre sí — *Viajante* y *SAT*

*los problemas que se reducen unos en otros en tiempo polinómico son de **complejidad similar***

Problemas intratables

Ejemplo: Las torres de Hanoi



El juego consiste en mover los *n* anillos de la torre A a la torre B o C, usando la otra como ayuda, pero sin que haya un disco de mayor diámetro sobre otro de menor.

- La cota mínima de complejidad para este problema es 2^n

  un ordenador capaz de hacer 10^6 operaciones por segundo, tardaría 1 ms en resolver el juego con 10 anillos y casi 36 años si colocamos 50 anillos !

Problemas intratables

Número de discos *n*	Tiempo de CPU (sg)[1]	Tiempo de CPU (sg)[2]	Tiempo de CPU (sg)[3]	Tiempo de CPU (sg)[4]
14	0.96	0.15		
15	1.92	0.26		
16	3.86	0.52		
17	7.74	0.99	0.57	0.187
18	15.57	1.98	1.11	0.312
19	31.27	3.89	2.16	0.64
20	62.61	7.82	4.29	1.312
21	124.76	15.62	8.8	2.73

[1] HP 9000/D250 (tejo)

[2] SGI Origin 200 (lisisu02)

[3] Sun Enterprise 250 (encina)

[4] Pentium 4 a 2,8 MHz

•Tiempos calculados con un programa en Fortran 90

Problemas aún más intratables

- Muchos juegos son problemas intratables (ej. el ajedrez 3^n)
- Hay problemas todavía peores:
  - la *aritmética de Presburger* (la que trata de establecer mecanismos de validación lógica sobre proposiciones planteadas en números enteros con la operación suma) es de complejidad ***2exp(2(exp(n)))***
    n, en este caso, es el número de operadores de la proposición $(+, =, \Rightarrow, \neg, \exists ...)$
  - *WS1S* (es decir, *Presburger* incluyendo afirmaciones sobre conjuntos de enteros) es ***fuertemente intratable***
  - la *aritmética de primer orden* (*Presburger* incluyendo la operación multiplicación) es ***no computable***

Clases de complejidad

Clase P

Problemas **de decisión** computables
- por una MT (o algoritmo) **determinista**
- en **tiempo polinómico**

Clase NP

Problemas **de decisión** computables
- por una MT (o algoritmo) **no determinista**
- en **tiempo polinómico**

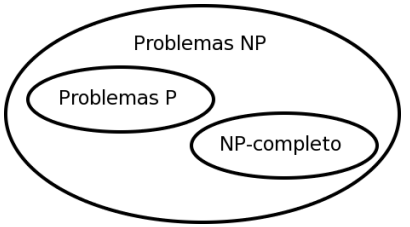
Conjetura ¿P=NP?

¡1 millón de dólares para quien lo demuestre!

Clases de complejidad

NP-difícil (*NP-hard*)
- Problema al que se pueden reducir en tiempo polinómico problemas NP

NP-completo
- es NP-difícil
- está en NP



Ejemplos

Ejemplos de problemas NP-completos

- Satisfacibilidad booleana (SAT)
- Viajante de comercio (TSP)
- Ciclo hamiltoniano
- Puzzle del 15 (generalizado)
- Buscaminas
- Tetris
- Sudoku
- …

Ejemplos

Problema del viajante (TSP)

Se dispone de un mapa de n ciudades, con las distancias entre ellas. ¿Es posible realizar una ruta que pase por todas las ciudades y cuya longitud total sea menor de una cierta cantidad ?

- Este problema es inabordable para *n* mayor de 200.
- Se puede generalizar fácilmente: cómo mover la cabeza de un robot de taladrar agujeros sobre un circuito impreso (cientos de agujeros, y miles de circuitos)

Ejemplos

Primos

Dado un número entero no negativo ¿es primo?

- era considerado NP hasta que, en 2002, Manindra Agrawal con sus estudiantes encontró un algorimto polinómico (*PRIMES in P, 2002*)
- en este problema se basan algunos métodos de encriptación

Decidir si un número es primo es un problema P

Más clases de complejidad

Otras clases

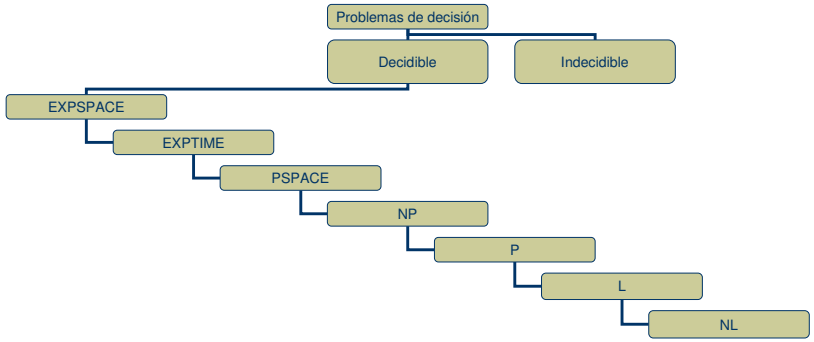
Considerando complejidad temporal

- EXPTIME tiempo exponencial $t(n) \in O(2^{p(n)})$
- NEXPTIME tiempo exponencial en máquina no determinista
- …

Respecto a la complejidad espacial

- L espacio logarítmico y tiempo ilimitado
- NL espacio logarítmico en máquina no determinista
- PSPACE espacio polinómico y tiempo ilimitado
- EXPSPACE espacio exponencial $s(n) \in O(2^{p(n)})$ y tiempo ilimitado
- ….

Jerarquía de Clases



Algunas ideas finales

- Los problemas *no computables* seguirán siéndolo aunque cambiemos el modelo de computación.
- Los problemas *intratables* pueden abordarse con modelos alternativos:
  - Sistemas paralelos
  - Computación cuántica
  - Computación molecular
  - Computación estocástica
- Hay mucho "trabajo pendiente" en este campo
- …

Bibliografía

- Brassard, G. y P. Bratley; *Fundamentos de Algoritmia*; Prentice Hall, 1997.
- Brookshear, J. G.; *Teoría de la computación: Lenguajes formales, autómatas y complejidad*; Addison Wesley, 1993.
- Harel, D.; *Computers Ltd. What they really can't do*; Oxford University Press, 2000.
- Hopcroft, J. E., R. Motwani, J. D. Ullman; *Introducción a la teoría de autómatas, lenguajes y computación*; Addison Wesley, 2002.
- http://es.wikipedia.org/wiki/Clase_de_complejidad
- http://www.ics.uci.edu/~eppstein/cgt/hard.html
- Apuntes UNED
- …