Escuela Técnica Superior de Ingeniería **Informática**

etsinf

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Computabilidad y Complejidad

Tema 6: Introducción a la Teoría de la Complejidad Computacional

Tema 6: Introducción a la Teoría de la Complejidad Computacional

Índice

1. Introducción. Lenguajes y Problemas
2. Medidas de complejidad abstractas: La axiomática de Blum.
3. Complejidad espacial y temporal en máquinas de Turing.
4. Algunas clases de complejidad.
5. Complejidad en máquinas de registros. El criterio uniforme vs. el criterio logarítmico

Bibliografía básica recomendada

- Introduction to automata theory, languages and computation. J.E. Hopcroft, J.D. Ullman, R. Motwani. Ed. Addison-Wesley. 2001.
- Theory of Computational Complexity. S. Du, K. Ko. John Wiley & Sons. 2000
- Computers and intractability : A guide to the theory of NP-completeness. M. Garey, D. Johnson. Ed. W.H. Freeman. 1979.

La Teoría de la Complejidad Computacional es una disciplina que establece y analiza el consumo de recursos computacionales necesarios para resolver un problema basado en los algoritmos diseñados para esa tarea (nunca en función de sus implementaciones ni de las plataformas informáticas en las que se ejecutan).

Los principales aspectos a considerar para definir correctamente la complejidad computacional son:

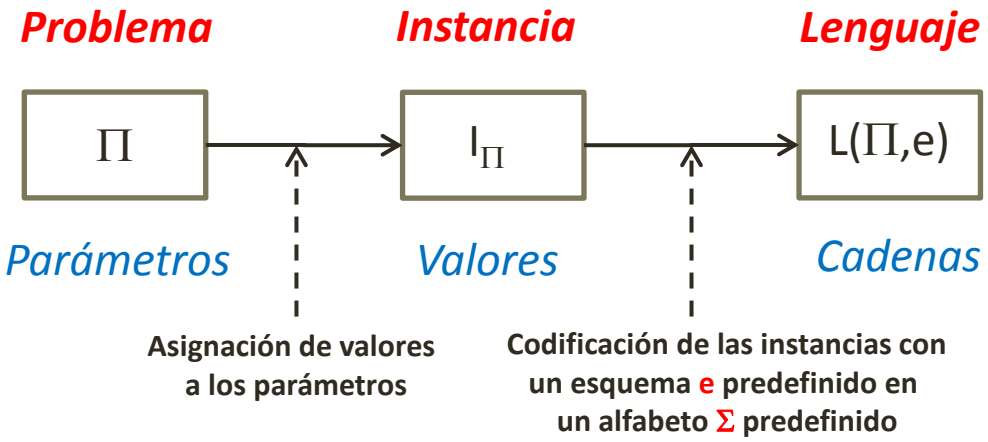
- ¿ Qué es un recurso computacional ?
  Ejemplos: tiempo necesario para encontrar la solución,
  espacio (memoria) necesario para encontrar la solución,
  tiempo de acceso a un entorno distribuido,

- ¿ Qué es un problema ?
  Cualquier cuestión que: 1) se pueda definir matemáticamente y 2) el número de instancias de la misma sea infinito.
  Tipos de problemas: de decisión, de generación, de optimización, de búsqueda

- ¿ Qué solución tomamos en cuenta para realizar una medida de complejidad ?
  Soluciones deterministas vs. no deterministas.
  Soluciones basadas en un modelo de computación vs. soluciones abstractas.
  Soluciones convencionales vs. soluciones no convencionales.

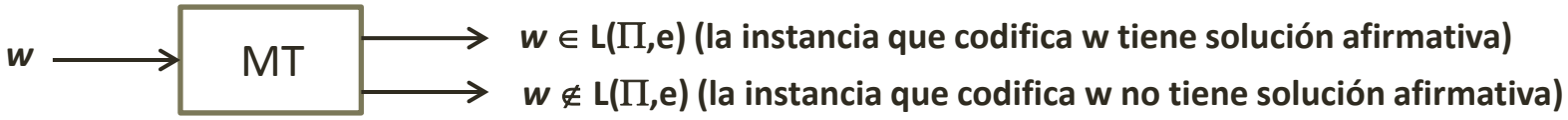
Problemas y lenguajes

Podemos establecer una correspondencia entre los problemas de decisión y los lenguajes formales



L(Π,e): Es el lenguaje sobre Σ^* formado por aquellas cadenas que codifican instancias del problema Π con solución afirmativa que han sido codificadas con el esquema e.

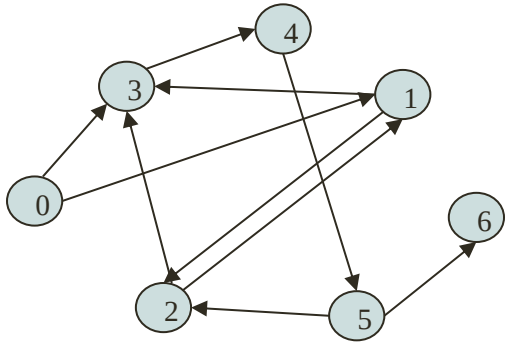
Resolución mediante máquinas de Turing



Si **L(Π,e) no es recursivo ó es finito entonces la complejidad del problema Π no se puede calcular ó se resuelve de forma trivial.**

Problemas y lenguajes

Ejemplo: El problema del camino hamiltoniano (versión dirigida)



¿ Existe un camino que recorra todos los vértices del grafo una sóla vez ?

Solución: 0-1-2-3-4-5-6

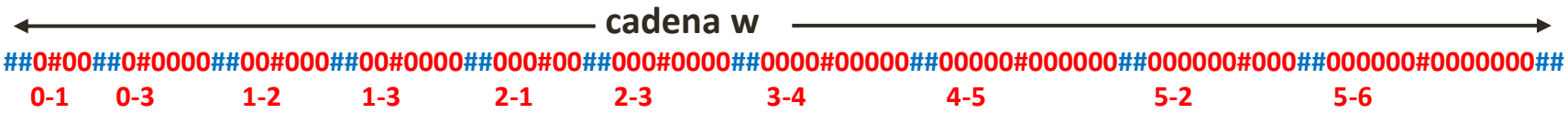
Parámetros del problema: Un grafo dirigido G
Instancias: Todos los posibles grafos dirigidos (existen infinitos)
Esquema de codificación: Proporcionar la lista de las aristas codificada

Una arista: <núm_vértice_de_origen+1>#<núm_vértice_de_destino+1>

0#00

Lista de aristas: ##<arista_1>##<arista_2>##...##<arista_n>##

Lista de aristas:



Algunos protagonistas de la Teoría de la Complejidad Computacional

J. Hartmanis and R.E. Stearns "On the computational complexity of algorithms." Trans. Amer. Math. Soc. 117 (1965), 285--306.

Introdujeron el concepto de complejidad computacional y el de clase de complejidad

M. Blum "A Machine-Independent Theory of the Complexity of Recursive Functions," Journal of the ACM, Vol. XIV, No. 2, April 1967, pp. 322-336.

Desarrolló la teoría de la complejidad independiente del modelo (ó máquina) que subyace a todos los posibles estudios de la complejidad

Stephen A. Cook, "The complexity of theorem-proving procedures," *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing* (STOC '71), ACM, New York, NY, USA, 1971, pp. 151-158.

Introdujo el concepto de problema NP-completo y estableció el primer problema que tiene esa propiedad: el problema de la SATISFACIBILIDAD (SAT)

R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations: Proceedings of a Symposium on the Complexity of Computer Computations,* R. E. Miller and J. W. Thatcher, Eds., The IBM Research Symposia Series, New York, NY: Plenum Press, 1972, pp. 85-103.

Proporcionó la primera lista de 21 problemas NP-completos. Definió el concepto de reducibilidad polinómica y propuso una serie de técnicas de demostración de NP-completitud.

Medidas de Complejidad Abstracta
(la axiomática de Blum)

La axiomática de Blum establece las condiciones necesarias y suficientes que permiten definir una medida de complejidad de los problemas **independiente** de cualquier máquina abstracta o real en las que se puedan ejecutar sus algoritmos de resolución.

Se basa en la definición de funciones recursivas que permiten capturar la esencia de cuándo y cómo se pueden definir medidas de complejidad.

La *axiomática* de Blum

Consideremos en primer lugar el conjunto de todas las funciones computables $\{\varphi_i : i \in \mathbb{N}\}$ (estas funciones se caracterizan a partir de la enumeración de todas las máquinas de Turing)

Una medida de complejidad γ se define como un conjunto de funciones computables $\{\gamma_i : i \in \mathbb{N}\}$ que cumplen los dos siguientes axiomas

1. $(\forall i \in \mathbb{N})\ (\text{dominio}(\gamma_i) = \text{dominio}(\varphi_i)) \wedge (\text{rango}(\gamma_i) \in \mathbb{N})$

2. Existe una función computable ***coste***: $\mathbb{N}\times\mathbb{N}\times\mathbb{N} \rightarrow \{1,0\}$ definida como

$$\textit{coste}(i,n,k) = \begin{cases} 1 & \text{si } \gamma_i(n)=k \\ 0 & \text{en cualquier otro caso} \end{cases}$$

Algunos ejemplos de Medidas de Complejidad Abstracta (… y algunas que no lo son)

Ejemplo 1: Considere que se define el conjunto de funciones computables $\{\gamma_i : i \in \mathbb{N}\}$ de forma que **$(\forall\, i,n \in \mathbb{N})\ \gamma_i(n)=k$ siendo k una constante**
Este conjunto no cumple el primer axioma de Blum por lo que no se puede considerar una medida de complejidad.

Ejemplo 2: Considere que se define el conjunto de funciones computables $\{\gamma_i : i \in \mathbb{N}\}$ de forma que **$(\forall\, i,n \in \mathbb{N})\ \gamma_i(n)= \varphi_i(n)$**
Este conjunto no cumple el segundo axioma de Blum (ya que si lo fuera se podrían resolver problemas indecidibles). Por lo tanto, no se puede considerar una medida de complejidad.

Ejemplo 3: Considere que se define el conjunto de funciones computables $\{TIME_i : i \in \mathbb{N}\}$ de forma que **$(\forall\, i,n \in \mathbb{N})$ $TIME_i(n)$ es el tiempo necesario para calcular $\varphi_i(n)$ en la máquina de Turing M_i**
Este conjunto cumple la axiomática de Blum y, por lo tanto, es una medida de complejidad.

Ejemplo 4: Considere que se define el conjunto de funciones computables $\{SPACE_i : i \in \mathbb{N}\}$ de forma que **$(\forall\, i,n \in \mathbb{N})$ $SPACE_i(n)$ es el número de celdas en blanco que la máquina de Turing M_i utiliza para calcular $\varphi_i(n)$**
Este conjunto cumple la axiomática de Blum y, por lo tanto, es una medida de complejidad.

Complejidad espacial y temporal (I)

La definición pragmática de la complejidad espacial y temporal se basa, en primer lugar, en la adopción de un modelo de cálculo sobre el que se establecen las medidas del consumo de recursos (en este caso espacio y tiempo). Nosotros tomaremos como referencia los modelos de cálculo de la máquina de Turing determinista y no determinista (en ambos casos multicinta). Consideraremos que la cinta de entrada es sólo de lectura y el contenido de sus celdas no se puede modificar.

A continuación se debe definir cómo se va a establecer la medida sobre el modelo de cálculo

Definición (**complejidad espacial determinista**)
"Dada una máquina de Turing multicinta determinista M, diremos que M está acotada en espacio por S(n) *si para cada cadena de entrada de longitud* n*, M explora como máximo S*(n) *celdas en cada cinta. Diremos que la complejidad espacial de M (o del lenguaje L(M)) es S*(n)*."*

Definición (**complejidad temporal determinista**)
"Dada una máquina de Turing multicinta determinista M, diremos que M está acotada en tiempo por T(n) *si para cada cadena de entrada de longitud* n*, M realiza un máximo de T*(n) *movimientos antes de parar. Diremos que la complejidad temporal de M (o del lenguaje L(M)) es T*(n)*."*

Complejidad espacial y temporal (II)

Definición (**complejidad espacial no determinista**)

"Dada una máquina de Turing multicinta no determinista M, diremos que M está acotada en espacio por S(n) *si para cada cadena de entrada de longitud* n*, existe una secuencia mínima de movimientos de forma que M acepta la cadena de entrada y explora un máximo de S*(n) *celdas en cada cinta. Diremos que la complejidad espacial de M (o del lenguaje L(M)) es S*(n)*."*

Definición (**complejidad temporal no determinista**)

"Dada una máquina de Turing multicinta no determinista M, diremos que M está acotada en tiempo por T(n) *si para cada cadena de entrada de longitud* n*, existe una secuencia mínima de T*(n) *movimientos de forma que M acepta la cadena de entrada. Diremos que la complejidad temporal de M (o del lenguaje L(M)) es T*(n)*."*

DSPACE(S(n)) clase de lenguajes aceptados por máquinas de Turing deterministas con complejidad espacial S(n)

NSPACE(S(n)) clase de lenguajes aceptados por máquinas de Turing no deterministas con complejidad espacial S(n)

DTIME(T(n)) clase de lenguajes aceptados por máquinas de Turing deterministas con complejidad temporal T(n)

NTIME(T(n)) clase de lenguajes aceptados por máquinas de Turing no deterministas con complejidad temporal T(n)

Complejidad espacial y temporal (III)

¿ Qué condiciones deben cumplir las funciones T y S empleadas en la definición de complejidad ?

T debe ser una **función constructiva en tiempo** (debe ser una función recursiva total de forma que para cada valor n, existe una máquina de Turing determinista que para cada entrada de longitud n la máquina para exactamente en T(n) pasos)

S debe ser una **función constructiva en espacio** (debe ser una función recursiva total de forma que para cada valor n, existe una máquina de Turing determinista que para cada entrada de longitud n la máquina para en una configuración en la que existen exactamente S(n) celdas de la cinta con contenido no blanco y ninguna otra celda se ha explorado en el transcurso de la computación)

Algunos ejemplos de funciones constructivas en tiempo y espacio

n^k (siendo $k \geq 1$ una constante)
$n!$
$2^{c \cdot n}$ (siendo c una constante mayor que cero)
$n \cdot round(\log n)^k$ (siendo k una constante mayor que cero)
etc,etc …

Complejidad espacial y temporal (IV)

¿ Qué condiciones deben cumplir las funciones T y S empleadas en la definición de complejidad ?

Además, utilizaremos la *notación asintótica* en la definición de funciones que actúen como cotas superiores e inferiores

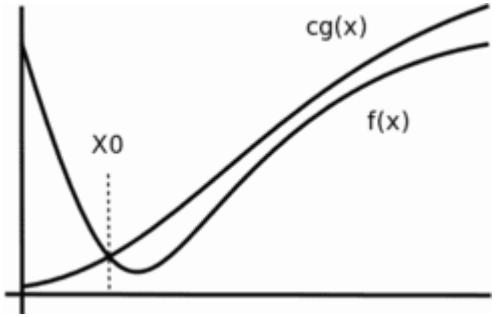
Sean f y g dos funciones definidas como **f,g: $\mathbb{N}$ - {0} → $\mathbb{N}$ - {0}** . Definimos los dos siguientes conjuntos de funciones cota

- O(g(n)) es el conjunto de funciones f que cumplen el enunciado

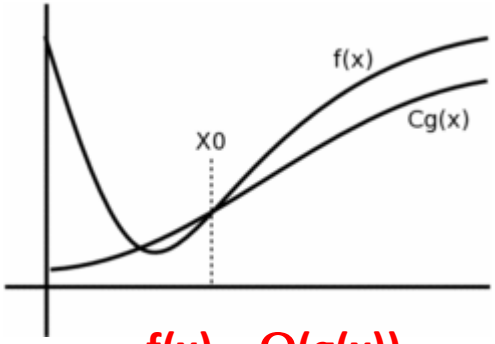
  $\exists$ r > 0 : $\forall$ x > x_0 f(x) < r·g(x)

- Ω(g(n)) es el conjunto de funciones f que cumplen el enunciado

  $\exists$ r > 0 : $\forall$ x > x_0 f(x) > r·g(x)



f(x) = O(g(x))



f(x) = Ω(g(x))

Complejidad espacial y temporal (IV)

Clases de complejidad

Una clase de complejidad se define como un conjunto de lenguajes (por lo tanto como un conjunto de problemas) que comparten una complejidad temporal o espacial, determinista o no determinista.

Algunos ejemplos

Nombre de la clase	Definición
DLOG	DSPACE(log n)
NLOG	NSPACE(log n)
PLOG	DSPACE($\log^{O(1)} n$)
P	DTIME($n^{O(1)}$)
NP	NTIME($n^{O(1)}$)
PSPACE	DSPACE($n^{O(1)}$)
NPSPACE	NSPACE($n^{O(1)}$)
DEXPT	DTIME($O(1)^n$)
NEXPT	NTIME($O(1)^n$)
DEXPTIME	DTIME($2^{n^{O(1)}}$)
NEXPTIME	NTIME($2^{n^{O(1)}}$)
EXPS	DSPACE($O(1)^n$)
EXPSPACE	DSPACE($2^{n^{O(1)}}$)

¿ Qué relaciones existen entre estas clases ?

Cómo establecer medidas de complejidad en otros modelos de computación

"*Cualquier intento razonable de modelar matemáticamente los algoritmos y sus prestaciones en tiempo, está acotada por un modelo de computación y su coste temporal (espacial) asociado, que es equivalente polinómicamente con la máquina de Turing*".
(C. Papadimitriou, 1994)

La Tesis de Church-Turing Extendida

Cualquier función computable en tiempo polinómico en un modelo de computación *"razonable"* utilizando una medida de complejidad temporal "*razonable*" es computable por una máquina de Turing determinista en tiempo polinómico.

Relaciones polinómicas

Sean f_1 y f_2 dos funciones definidas sobre el conjunto de enteros positivos. Decimos que f_1 y f_2 están polinómicamente relacionadas si existen dos polinomios $p_1(x)$ y $p_2(x)$ de forma que se cumplan las siguiente relaciones

$$(\forall\, n \geq 0)\ \ f_1(n) \leq p_1(f_2(n))$$
$$(\forall\, n \geq 0)\ \ f_2(n) \leq p_2(f_1(n))$$

Complejidad en máquinas de registros (ó en máquinas RAM)

Partimos de un conjunto de instrucciones predefinido y les adjudicamos un coste temporal/espacial. Existen dos posibles criterios para asignar los costes: *uniforme* y *logarítmico*.

Criterio uniforme: Cada instrucción requiere una unidad de tiempo para ejecutarse y cada entero (independientemente de su magnitud) ocupa un registro.

Criterio logarítmico: Cada instrucción requiere una cantidad de unidades de tiempo en ejecutarse que es una función logarítmica de las magnitudes de los enteros que participan en la instrucción y cada entero ocupa un número de registros que es función logarítmica de su magnitud.

Complejidad temporal: Es la suma de las unidades de tiempo empleadas por las instrucciones ejecutadas desde el inicio de la computación hasta que la máquina para.

Complejidad espacial: Es el número de registros distintos utilizados desde el inicio de la computación hasta que la máquina para.

No se pueden establecer relaciones polinómicas entre las complejidades de las máquinas RAM con criterio uniforme respecto a las de las máquinas de Turing

Complejidad en máquinas de registros (ó en máquinas RAM)

Tomemos las siguientes instrucciones de una máquina RAM

READ i almacena en el registro i un valor entero leído de la entrada estándar
STORE i almacena en el registro i el contenido del registro 0
MULT i almacena en el registro 0 el producto del contenido del registro 0 por el del registro i
LOAD i almacena en el registro 0 el contenido del registro i
SUB =1 resta uno al contenido del registro 0
JZERO <et> sigue la ejecución en la instrucción etiquetada por <et> si el contenido del registro 0 es cero. En caso contrario sigue la ejecución secuencial.
JUMP <et> sigue la ejecución en la instrucción etiquetada por <et>
DIV i almacena en el registro 0 la división del contenido del registro 0 por el del registro i
WRITE i muestra en la salida estándar el contenido del registro i
HALT finaliza la ejecución del programa

Complejidad en máquinas de registros (ó en máquinas RAM)

Ejemplo:

Tomemos el siguiente programa RAM que calcula la función $f(n) = n^n$ y y que obedece al siguiente algoritmo

```
        READ 0
        STORE 1
        STORE 2
<loop>  MULT 1
        STORE 3
        LOAD 2
        SUB =1
        JZERO <fin>
        STORE 2
        LOAD 3
        JUMP <loop>
<fin>   LOAD 3
        DIV 1
        WRITE 0
        HALT
```

```
leer(n)
var=n
Para k=2 hasta n hacer
        var=var*n
```

Complejidad de la función

(a) Criterio uniforme

Complejidad espacial $O(1)$

Complejidad temporal $O(n)$

(b) Criterio logarítmico

tomando $n \cong b^k$

Complejidad espacial $O(k \cdot b^k)$

Complejidad temporal $O(k \cdot b^k)$

No se puede establecer una relación razonable entre ambos criterios.

Conclusiones

- La complejidad computacional es una medida de los recursos necesarios para resolver un problema en un modelo de computación.
- La complejidad computacional se define para los problemas no para los algoritmos de resolución. No tiene sentido definir la complejidad para problemas irresolubles (indecidibles) o con un número de instancias finito.
- El modelo de computación referencia para medir la complejidad es la máquina de Turing. Los demás modelos de computación, para establecer medidas razonables de complejidad, deben establecer relaciones polinómicas con la máquina de Turing.
- Los problemas que comparten una misma medida de complejidad definen lo que se denomina clase de complejidad. Podemos establecer una identificación entre una clase de problemas y una clase de lenguajes.