ADA–TEMA 6.2
Introducción a la NP–completitud

Amalia Duch Brown

Universitat Politècnica de Catalunya

27 de abril de 2006

Hasta ahora en ADA:

- Hemos estudiado algoritmos para distintos problemas.
- Estos algoritmos han sido lo más eficientes posibles.
- Ejemplo: ordenación utilizando mergesort o heapsort.

Hasta ahora en ADA:

La mayoría de las veces hemos encontrado costes:

1. sublineales
2. lineales
3. quasi lineales

En algunas ocasiones hemos encontrado costes más altos pero aún así polinómicos.

Ejemplo: ordenación por inserción o por selección.

Pero...

- En este tema hemos hablado de algoritmos no polinómicos.
- La fuerza bruta (búsqueda exhaustiva) nos obliga a explorar un espacio muy grande.
- De hecho los árboles que representan a las soluciones crecen exponencialmente.

Pregunta...

La pregunta obvia para todos estos problemas es:

¿Se pueden resolver mejor?

Es decir,

¿Se pueden resolver en tiempo polinómico?

Respuesta...

Nadie lo sabe

Se trata de la famosa pregunta:

¿P=NP?

Esta pregunta es el principal problema de investigación en informática y también uno de los 7 problemas matemáticos que se consideran más importantes de resolver.

www.claymath.org/millennium

Lo que haremos...

En este tema intentaremos entender de qué se trata este problema y cuál es su relevancia en relación con algunos problemas que hemos visto cómo solucionar con búsqueda exhaustiva.

Problema Computacional

Un problema computacional se especifica dando:

- Un conjunto de posibles entradas E.
- Un conjunto de posibles salidas S.
- Una relación de entradas y salidas $R \subseteq E \times S$.
- Una función de talla $|\cdot| : E \longrightarrow N$

El problema consiste en, dada una entrada $x \in E$, encontrar una salida $y \in S$ tal que $(x, y) \in R$.

Ejemplos

HAM: Grafo Hamiltoniano.

- E = conjunto de todos los grafos conexos.
- S = conjunto de todos los posibles caminos.
- $(x, y) \in R \iff y$ es un ciclo simple de tamaño n.
- $|x|$ = tamaño del grafo x.

Ejemplos

TSP: El viajante de comercio (travelling salesman problem).

- E = conjunto de todos los grafos conexos con pesos.
- S = conjunto de todas las posibles secuencias de aristas.
- $(x, y) \in R \iff y$ es un ciclo de coste mínimo que pasa por todos los vértices de x
- $|x|$ = tamaño del grafo x.

Ejemplos

ORD: Ordenar todos los elementos de un vector.

REC: Recorrer todos los vértices de un grafo.

- $E =?$
- $S =?$
- $(x, y) \in R \iff y?$
- $|x| =?$

Problema Decisional

Un problema decisional se especifica dando:

- Un conjunto de entradas E.
- Un subconjunto de entradas positivas $L \subseteq E$.
- Una función de talla $|\cdot| : E \longrightarrow N$

El problema consiste en, dada una entrada $x \in E$, determinar si x está en L o no.

Ejemplos

- HAM: ¿Es Hamiltoniano un grafo dado?
- CIRCUIT-SAT: ¿Es satisfactible un circuito booleano?

Ejemplos

- Ordenación.
- N reinas.
- Sudoku.
- Buscaminas.

Observaciones

- Casi todos los problemas se pueden asociar a un problema decisional.
- Si se trata de problemas de optimización hay que introducir un nuevo parámetro.

Ejemplos

- TSP-DEC: Dado un grafo G y un entero k, ¿tiene G un ciclo hamiltoniano de coste $\leq k$?

Mas observaciones

- Las versiones decisionales son más fáciles que las versiones de optimización.
- Si sabemos resolver un problema de optimización es muy fácil resolver el problema decisional.
- Si las versiones decisionales son difíciles, las versiones de optimización lo son aún más.

Algoritmos Polinómicos

Recordemos que el coste en el caso peor de un algoritmo A se define en función de la talla de la entrada:

$$T_A(n) = max\{t_A(x) : x \in E, |x| = n\}$$

donde $t_A(x)$ es el número de pasos que realiza A sobre x.

Definición

Un algoritmo A con coste $T_A(n)$ en el caso peor es un algoritmo polinómico si $T_A(n) = O(n^c)$ para algún $c \geq 0$.

Ejemplo:

- El algoritmo de ordenación por inserción.
- El algoritmo de ordenación por selección.
- Quicksort.

Decidibilidad en Tiempo Polinómico

Un problema decisional L es decidible en tiempo polinómico si existe algún algoritmo polinómico que lo resuelva.

En este caso decimos que L está en la clase P.

P es la clase de todos los problemas decisionales decidibles en tiempo polinómico.

Ejemplo

Decidir si un grafo no dirigido contiene ciclos o no es un problema que pertenece a la clase P .

¿Porqué?

Encontrar y Verificar

Para poder entender la case NP es necesario entender la diferencia entre *encontrar* y *verificar*.

Ejemplo sencillo: Es más fácil corregir un examen (verificar que una solución es correcta) que hacerlo (encontrar una solución).

Ejemplo HAM:

- Encontrar un ciclo HAM en un grafo es difícil (backtracking)
- Dada una secuencia de vértices es fácil comprobar que esta forma un ciclo hamiltoniano (algoritmo polinómico).

Para poder verificar hace falta un *testimonio* (un examen, una secuencia de vértices).

Verificadores

Un algoritmo $B : E \times T \rightarrow \{0, 1\}$ es un *verificador* del problema decisional $L \subseteq E$ con testimonios T si:

$$\forall x \in E, \forall y \in T, B(x, y) = 1 \Rightarrow x \in L$$

La Clase NP

Un problema decisional $L \subseteq E$ es decidible en tiempo polinómico indeterminista si:

- Existe un polinomio $p(n)$
- Existe un algoritmo polinómico $B : E \times T \rightarrow \{0, 1\}$

tal que para todo $x \in E$, $x \in L \iff$ existe $y \in T$ tal que $|y| \leq p(|x|)$ y $B(x, y) = 1$.

En este caso se dice L está en la clse NP

NP es la clase de todos los problemas decidibles en tiempo polinómico indeterminista.

Observación

De manera informal podemos decir que un problema está en *NP* si su conjunto de entradas positivas tiene testimonios *cortos* (de tamaño polinómico) que se pueden verificar (con un verificador) en tiempo polinómico.

Ejemplos

Por lo general es fácil ver que hay muchos problemas en NP.

- Buscar el elemento mínimo en una secuencia $\in NP$
- HAM $\in NP$
- TSP $\in NP$
- Colorabilitat $\in NP$
- Quadrat llatí $\in NP$
- Sudoku $\in NP$

Theorem

$$P \subseteq NP$$

Demostración: ejercicio.

Reducciones

Sean $L_1 \subseteq E_1$ y $L_2 \subseteq E_2$ dos problemas decisionales.
Decimos que L_1 se *reduce* a L_2 en tiempo polinómico si existe un algoritmo polinómico $A : E_1 \to E_2$ tal que:

$$\forall x \in E_1 : x \in L_1 \iff A(x) \in L_2$$

En este caso decimos que A es una reducción polinómica de L_1 a L_2, lo que suele escribirse $L_1 \leq L_2$.

HAM se reduce a TSP

- Si un grafo G es hamiltoniano tiene un ciclo hamiltoniano de n aristas cada una de coste 1 y por tanto tiene un ciclo hamiltoniano de coste $\leq n$.
- Si G no es hamiltoniano cada ciclo tendrá un coste $> n$.

Theorem

Si $L_1 \leq L_2$ y $L_2 \in P$ entonces $L_1 \in P$

Dem:?

Theorem

Si $L_1 \leq L_2$ y $L_2 \in P$ entonces $L_1 \in P$

Dem: Si $L_2 \in P$ entonces existe un algoritmo polinómico B para decidir L_2. Si $L_1 \geq L_2$ entonces existe un algoritmo A que transforma las entradas de L_1 en entradas de L_2 en tiempo polinómico. Consideremos un algoritmo C que dada una entrada de L_1 utiliza A para tranformarla en una entrada de L_2 y utiliza B para decidirla. Es claro que C decide una entrada de L_1 en tiempo polinómico y por tanto que L_1 está en P.

Problemas NP–hards

Un problema decisional L es NP–hard si todo problema en NP puede reducirse a él en tiempo polinómico:

$$L \in NP\text{–hard} \iff \forall L' \in NP, L' \leq L$$

Por tanto, por el teorema anterior, si $L \in NP$–hard se pudiese resolver en tiempo polinómico, entonces cualquier otro problema en la clase NP también podría resolverse en tiempo polinómico.

Theorem

Si $L \in NP$-hard y $L \in P$ entonces $P = NP$

Problemas NP–completos

Se dice que un problema es *NP*–completo si:

- está en la clase *NP* y
- es *NP*–hard.

Por tanto si un problema es *NP*–completo es tan difícil de resolver o más que cualquier otro problema en la clase *NP*.

Theorem

Si L_1 es un problema decisional NP–completo, L_2 es un problema decisional en la clase NP y $L_1 \leq L_2$ entonces L_2 es NP-completo.

Lo que dice este teorema es que para demostrar que un problema es *NP*–completo es necesario reducir a él, en tiempo polinómico, uno que ya sea *NP*–completo.

Teorema de Cook

La pregunta que nos queda por resolver ahora es:

¿Existen problemas *NP*–completos?

Teorema de Cook

El teorema de Cook demuestra que sí.

Theorem

CIRC-SAT es NP–completo

Por tanto ya tenemos manera de ir demostrando que determinados problemas son *NP*–completos.

Observaciones

- La importancia de poder demostrar que un problema es *NP*–completo es que si algún problema *NP*–completo tuviese un algoritmo que lo resolviese en tiempo polinómico, entonces todos los problemas de la clase *NP* también lo tendrían.
- Por desgracia no se ha podido encontrar hasta ahora tal algoritmo para ningún problema *NP*–completo.
- Aún peor, nadie ha podido demostrar hasta ahora que tal algoritmo no pueda existir.

Especulaciones

- La inmensa mayoría de los expertos piensan que las clases P y NP son diferentes.
- Cook: dudoso.
- Levin: P y NP son iguales.

Fin.