Algorítmica y Lenguajes de Programación

Complejidad computacional

Complejidad computacional. Introducción

- La complejidad computacional estudia la "dificultad" inherente de problemas de importancia teórica y/o práctica.
- El esfuerzo necesario para resolver un problema de forma eficiente puede variar enormemente.
- Un problema muy complejo se denomina "NP-completo", lo cual básicamente significa que es imposible encontrar un algoritmo eficiente para encontrar una solución óptima.
- Probar que un problema es "NP-completo" es muy importante puesto que permite abandonar un callejón sin salida (encontrar un algoritmo para la solución óptima) para centrarse en objetivos realizables (encontrar algoritmos para obtener soluciones aproximadas).
- El objetivo fundamental de la teoría de la complejidad computacional es facilitar el avance en aquellas áreas en las que es posible.

2

Complejidad computacional. Optimización y decisión

- Dentro de la evolución de la teoría de complejidad, se han encontrado problemas con características similares, que pueden ser agrupados en categorías para su estudio.
- Los **problemas de optimización** son aquellos en los cuales se busca minimizar o maximizar (es decir, optimizar) el valor de una solución en un grupo de soluciones generadas para una entrada específica (instancia). La forma general de exponer un problema de optimización es:

  optimizar c(s) sobre s en F(n)
- Los **problemas de decisión** son aquellos donde se busca una respuesta de "si" o "no".
- **Cualquier problema de optimización puede ser manejado como un problema de decisión** incluyendo un valor objetivo *K* para la instancia *n* y preguntando si existe o no existe una solución factible en el conjunto de soluciones *F(n)*, con el valor de la solución *c(s)* optimizado sobre *K* (para el caso de minimización sería $c(s) \leq K$ y $c(s) \geq K$ para el caso de maximización).
- De esta forma, para propósitos teóricos, es mas conveniente tratar con problemas de decisión que con problemas de optimización.

3

Complejidad computacional. Problema de satisfactibilidad

- El problema de satisfactibilidad (**SAT**) es un problema central en la lógica matemática y la teoría de la computación.
- La **satisfactibilidad proposicional** es el problema de decidir si existe una asignación de valores de verdad a los átomos de una fórmula proposicional que la hacen verdadera.
- **Ejemplo:** La asignación de valores de verdad que satisfacen la fórmula
  - $(P \vee \neg Q) \wedge (Q \vee R) \wedge (\neg R \vee \neg P)$
  - es P = Q = V y R = F, por lo que la fórmula es **satisfactible**.

4

Complejidad computacional. Algoritmos no deterministas (i)

- Un algoritmo no determinista es una herramienta teórica imposible en la realidad.
- Se trata de algoritmos similares a los habituales pero que permiten la siguiente instrucción:

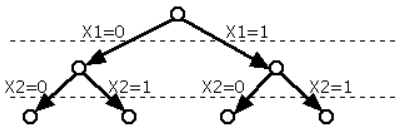
  `ir simultaneamente a etiqueta1 y etiqueta2`
- Esto divide el proceso computacional en dos procesos paralelos ejecutados simultaneamente por el mismo procesador.
- Para hacernos una idea de la potencia de esta herramienta consideremos el siguiente problema:
- Dada una matriz entera *A* de dimensiones *m x n* y un vector de *m* componentes, *b*, ¿existe un vector de *n* componentes, *x*, constituido sólo por los elementos 0 y 1, tal que *Ax=b*?

5

Complejidad computacional. Algoritmos no deterministas (ii)

- Esta instrucción permitiría resolver el problema en un tiempo polinomial al explorar simultaneamente todas las posibilidades:

```
inicio
  desde j ¬ 1 hasta n hacer
    ir simultaneamente a A y B
    A: x(j) ¬ 0
    B: x(j) ¬ 1
  fin desde
  si x satisface Ax=b entonces
    escribir "sí"
  si no
    escribir no
  fin si
fin
```



- Si se dispusiera de ordenadores capaces de ejecutar semejante instrucción no sería necesaria la Teoría de la Complejidad Computacional. Se especula con la posibilidad de que los ordenadores cuánticos pudieran disponer de semejante capacidad.

6

Complejidad computacional. Problemas P y NP

- Cualquier problema de optimización puede ser transformado en un problema de decisión. En el análisis de complejidad se manejan, por lo tanto, los problemas de decisión, que incluyen a los de optimización.
- Si se encuentra solución a un problema de decisión, entonces se encuentra también solución a un problema de optimización. Al estudiar los problemas de decisión, se pueden encontrar varias clases:
  - La clase de **problemas P** está formada por todos aquellos problemas de decisión para los cuales se tiene un algoritmo de solución que se ejecuta en **tiempo polinomial en una máquina determinista**.
  - La clase de **problemas NP** está formado por todos aquellos problemas de decisión para los cuales existe un algoritmo de solución que se ejecuta en **tiempo polinomial** en una (hipotética) **máquina no determinista**. Dicho de otro modo, **no se ha encontrado un algoritmo determinista que lo resuelva en tiempo polinomial**.

7

Complejidad computacional. NP completitud (i)

- La relación entre la clase P y la clase NP es estrecha: **P Í NP**. Cualquier problema de decisión resuelto por un algoritmo determinístico en tiempo polinomial también es resuelto por un algoritmo no determinístico en tiempo polinomial.
- **La relación anterior es fundamental en la teoría de NP-completitud.**
- Un famoso problema es determinar si P = NP o P ≠ NP.
- **¿Es posible que para todos los problemas en NP existan algoritmos deterministas que los resuelvan en tiempo polinomial y que aún no hayan sido descubiertos?**
- Esto parece improbable ya que mucha gente ha trabajado sobre estos problemas. Por otro lado, el hecho de que P ≠ NP tampoco ha sido probado.
- **La teoría de NP-completitud se basa en el concepto de transformación polinomial.**

8

Complejidad computacional. NP completitud (ii)

- Una **transformación polinomial** es una función que permite cambiar la representación de problema D_1 a otro problema D_2 aplicando un algoritmo determinista de tiempo polinomial.
- Lo anterior se puede representar como *"D_1 se transforma a D_2"* o $\boldsymbol{D_1 \mu D_2}$.
- Las transformaciones polinomiales son importantes porque sirven para determinar la pertenencia de los problemas a las clases P y NP, y permiten definir la clase **NP-completo**.
- Si D_1 se transforma a D_2 y D_2 pertenece a la clase P, entonces D_1 también pertenece a la clase P, porque, si para cambiar de D_1 a D_2 se utiliza un algoritmo de tiempo polinomial y si D_2 (ya que pertenece a la clase P) tiene un algoritmo de solución que se ejecuta en tiempo polinomial, D_1 debe tener asociado un algoritmo que lo resuelva también en tiempo polinomial, y D_1 debe pertenecer a la clase P.
- Se puede decir, en general, que un problema D_1 está en P si cualquier problema que se sabe está en P, se puede transformar a D_1 . ($\boldsymbol{D_p \mu D_p}$).

9

Complejidad computacional. NP completitud (iii)

- También se puede decir que un problema D_1 está en NP si cualquier problema que se sabe está en NP, se puede transformar a D_1.
- Un problema D_1 es NP-completo si pertenece a la clase NP, y otro problema D_2 que también pertenece a NP, se puede transformar a D_1.
- Los problemas NP-completos son los problemas más difíciles en la clase NP.
- La palabra completitud significa que la solución de un problema de decisión NP, contiene, de alguna forma, la solución a todos los problemas de decisión de la clase NP.
- Si D_1 se transforma a D_2 y D_2 es un problema NP-completo, entonces D_1 también un problema NP-completo. Esto es importante, ya que establece que se puede probar que un problema de decisión D_1 es NP-completo, si algún problema que se sabe es NP-completo puede transformarse al primero.
- El punto importante para demostrar que un problema es NP-completo es el tener un problema que sea NP-completo.
- **En 1971, Cook demostró que el problema de satisfactibilidad (SAT) es NP-completo.**

10

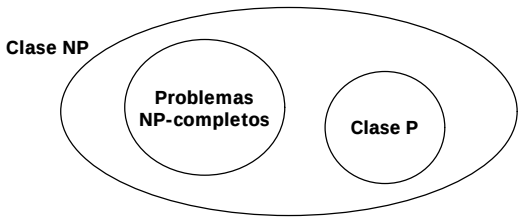
Complejidad computacional. NP completitud (iv)

- Algunas veces se puede demostrar que todos los problemas en NP se pueden transformar a algún problema *A* (D_{np} μ *A*), pero no se puede decir que A sea NP o NP-completo.
- Sin embargo, es indudable que A es tan difícil (duro) como cualquier problema en NP, y será entonces intratable. Para este tipo de problemas se asocia el término de **NP-duros** (o NP-hard).
- En la literatura también se usa el término NP-duro para describir a los problemas de optimización (que, no siendo problemas de decisión, no son NP) cuya versión de decisión está en NP-completo.

11

Complejidad computacional. Conclusión

- A la vista de lo anterior parece que el panorama es poco alentador puesto que hay problemas que no se pueden resolver de forma eficiente.
- Sin embargo, aún hay esperanza: En muchas ocasiones se generaliza de forma innecesaria; incluso cuando un problema se pueda transformar en un problema NP-completo eso no significa que no se trate de un caso especial que admite una solución.
- Un posible "mapa" de los problemas NP se muestra a continuación:



12