Complejidad computacional

Sebastian Mestre

complejidad $\neq$ costo

Costo vs complejidad

Definición: complejidad

La complejidad computacional **de un problema** es el mínimo costo asintótico posible que tiene **un programa** que resuelve ese problema.

- Ejemplo: para ordenar un arreglo de elementos que solo podemos comparar de a pares, es sencillo diseñar un algoritmo de costo $O(N^2)$.
- Existen algoritmos más sofisticados (por ejemplo, *merge sort*, *quick sort*) con costo $O(N \log N)$.
- Está demostrado que **no existen** algoritmos asintóticamente más rápidos.
- Entonces, el problema de ordenar un arreglo tiene complejidad $O(N \log N)$.

La clase P

Definición: P

P es el conjunto de todos los problemas cuya complejidad computacional es **polinomial**.
Es decir, problemas que se pueden resolver con costo:

$$O(N),\ O(N^2),\ O(N^3),\ O(1),\ O(N \log N), \ldots$$

Decimos que estos problemas tienen **complejidad polinomial**.

Problemas fuera de P

- Si un problema no es P, tiene complejidad **mayor que cualquier polinomio**.
- Ejemplos típicos de costos:

$$O(2^N), \quad O(N!), \quad \text{o peores.}$$

- Una forma de pensarlos: **"problemas donde no te queda más opción que probar todas las posibilidades"**.

La clase NP

Definición: NP

NP es el conjunto de todos los problemas para los cuales verificar si una solución dada es correcta tiene **complejidad polinomial**.

- En la práctica, verificar una solución suele ser más fácil que **encontrarla**.
- La teoría respalda parcialmente esta idea: $P \subseteq NP$.
- En otras palabras: si un problema se puede **resolver** en tiempo polinomial, entonces también se puede **verificar** una solución en tiempo polinomial.

Reducciones

- En programación competitiva usamos todo el tiempo algoritmos conocidos para resolver problemas nuevos.
- En realidad, esos algoritmos son soluciones a problemas conocidos, y nosotros **reducimos** nuestro problema a uno de esos problemas conocidos.

Definición: reducción

Una reducción es una funcion que convierte un caso de prueba de un problema a un caso de prueba de otro problema que tiene la misma respuesta.

Problemas NP-completos

Definición: NP-completo

Un problema es NP-completo si es NP y cualquier otro problema de NP se puede reducir a él con costo de reducción polinomial.
Es decir, podemos tomar cualquier caso de prueba de cualquier problema de *NP* y, en tiempo polinomial, convertirlo en un caso de prueba de uno de estos problemas especiales.

- **SAT** (satisfacibilidad booleana).
- **TSP** (problema del viajante).
- **Max-clique**.
- **Subset-sum**.
- **Max-independent set**.
- **Set-cover**.

La gran pregunta: P vs NP

- Sabemos que $P \subseteq NP$.
- Pero nadie sabe si existe algún problema en NP que **no** esté en P.
- En particular, se cree que los problemas **NP-completos** no pertenecen a P.

Si un NP-completo estuviera en P...

- Si encontráramos una **solución polinomial** para algún problema NP-completo:
  - Podríamos reducir **cualquier** problema de NP a ese problema NP-completo.
  - Resolveríamos también esos problemas en tiempo polinomial.
- Por lo tanto, se tendría que:

$$P = NP.$$

NP-completo en programación competitiva

- Un NP-completo **no se puede resolver** rapido para N grande. Sin embargo, muchos **casos especiales** salen con técnicas estándar.
- Clave: si el problema se parece a uno NP-completo famoso pero tiene una **pequeña diferencia o restricción**, normalmente:
  - Esa diferencia es lo que hay que explotar para resolverlo eficientemente.
- A veces se toma un problema NP-completo con N chico. No olvidemos practicar **búsquedas exhaustivas**:
  - DP con bitmasks
  - `next_permutation`
  - backtracking optimizado

Para leer más

- Material:
  - Wiki de complejidad (OIA Politécnico)
  - NP-complete (wikipedia)
  - SAT (wikipedia)
  - Travelling Salesman Problem (wikipedia)

Preguntas

Q&A