Escuela Politécnica Nacional

[Actividad extracurricular 3b] Complejidad computacional y el problema PvsNP

Asignatura: Métodos Numéricos
Estudiante: Joel Stalin Tinitana Carrión
Fecha: 15/07/2025

Objetivo

- Conocer y entender sobre la complejidad computacional.
- Familiarizarse con el problema de P vs NP.

Indicaciones

- Investigue sobre la notación big O. Esta medida es ¿para el peor escenario, mejor escenario, o el promedio?
- ¿Cuál es la diferencia con little o, y las otras funciones big Omega Ω, big Theta Θ?
- Investigue sobre el problema P vs NP, describa ejemplos de algoritmos en cada caso.

Introducción

La complejidad computacional es una rama de la informática teórica que estudia los recursos necesarios (tiempo y espacio) para resolver problemas computacionales. En este contexto, se han desarrollado herramientas matemáticas para describir la eficiencia de los algoritmos. Una de las más conocidas es la notación Big O, la cual permite clasificar algoritmos según su crecimiento en función del tamaño de entrada. Por otro lado, el problema P vs NP es uno de los grandes enigmas sin resolver, con profundas consecuencias teóricas y prácticas.

Desarrollo

1. ¿Qué es la notación Big-O?

La notación **Big-O** describe el **comportamiento del algoritmo en el peor de los casos**, es decir, cuán rápido crece el número de operaciones necesarias a medida que aumenta el tamaño de la entrada

$$n$$

. No se enfoca en detalles precisos como tiempos concretos de ejecución, sino en cómo escalan los algoritmos.

Por ejemplo:

- Un algoritmo con complejidad

$$O(n)$$

crece linealmente.

- Uno con

$$O(n^2)$$

crece cuadráticamente.

2. ¿Qué significa Big-O, little-o, Omega y Theta?

Notación	Descripción
$O(g(n))$	**Cota superior**: el algoritmo **no crece más rápido que** $g(n)$.
$o(g(n))$	**Cota superior estricta**: crece **estrictamente más lento** que $g(n)$.
$\Omega(g(n))$	**Cota inferior**: el algoritmo crece **al menos tan rápido** como $g(n)$.
$\Theta(g(n))$	**Cota ajustada**: el algoritmo crece **exactamente al mismo ritmo** que $g(n)$.

3. Tabla comparativa: notaciones de menor a mayor crecimiento

Notación	Nombre común	Ejemplo típico
$O(1)$	Constante	Acceder a un elemento de una lista por índice
$O(\log n)$	Logarítmica	Búsqueda binaria
$O(n)$	Lineal	Recorrer una lista
$O(n \log n)$	Lineal logarítmica	Merge Sort, Quick Sort (mejor caso)
$O(n^2)$	Cuadrática	Burbujas, selección, inserción
$O(n^3)$	Cúbica	Multiplicación de matrices
$O(2^n)$	Exponencial	Problema de la mochila (backtracking)
$O(n!)$	Factorial	Generación de permutaciones

4. ¿Cómo se compara la velocidad entre algoritmos?

Cuando decimos que un **algoritmo es más rápido o más lento**, nos referimos a cómo **crece su tiempo de ejecución** con respecto al tamaño de entrada (n). Este crecimiento se expresa con notaciones como (O(n)), (O(n^2)), (O(n^3)), etc. En general, un algoritmo es **más eficiente** si su exponente en (n) es menor, ya que su crecimiento es más lento. Por ejemplo, un algoritmo con (O(n^{3/2})) (equivalente a (O(n^{1.5}))) crece más lentamente que uno con (O(n^2)), y por lo tanto es **más rápido** en entradas grandes.

Los **constantes y divisores** como (O(n^2 / 2)) no afectan la clasificación en notación Big-O, porque esta se centra únicamente en el crecimiento asintótico. Es decir, (O(n^2 / 2)) y (O(n^2)) son equivalentes desde el punto de vista del análisis de algoritmos, porque ambos crecen con el cuadrado de (n). En cambio, (O(n^{3/2})) sí representa una mejora notable respecto a (O(n^2)), ya que su exponente es más bajo.

Por lo tanto, **el criterio más importante para comparar algoritmos es el exponente de (n)** en su notación Big-O. A menor exponente, menor crecimiento, y por tanto, mejor rendimiento para entradas grandes.

5. ¿Qué es el problema P vs NP?

El problema **P vs NP** es uno de los **siete problemas del milenio** propuestos por el Clay Mathematics Institute.

- **P**: Conjunto de problemas que pueden resolverse **rápidamente (en tiempo polinómico)** por un algoritmo determinista.

  Ejemplos:

  - Ordenamiento de listas (Merge Sort)
  - Búsqueda de un elemento en un arreglo (Búsqueda binaria)
  - Multiplicación de matrices
- **NP**: Conjunto de problemas cuya **solución puede verificarse rápidamente**, aunque no se sepa cómo hallarla rápidamente.

  Ejemplos:

  - Problema del viajante (TSP)
  - Problema de la mochila
  - Sudoku

Pregunta central del problema P vs NP:

> ¿Todo problema cuya solución puede ser verificada rápidamente también puede ser resuelto rápidamente?

En otras palabras, ¿P = NP?

Hasta hoy, **no se ha demostrado si P = NP o P ≠ NP**, por lo tanto, sigue siendo un problema abierto.

Conclusiones

- La notación Big-O es una herramienta fundamental para evaluar la **eficiencia de los algoritmos**, especialmente en el **peor de los casos**.
- Otras notaciones como **little-o**, **Ω (Omega)** y **Θ (Theta)** permiten describir límites inferiores y crecimientos exactos, lo que aporta mayor precisión al análisis.
- Un algoritmo se considera **más rápido** cuando su función de crecimiento (exponente de (n)) es menor, sin importar constantes o divisores.

- El problema **P vs NP** sigue siendo un desafío abierto en la ciencia de la computación. Entenderlo permite diferenciar entre problemas fáciles de resolver y aquellos cuya solución solo puede ser verificada, no necesariamente encontrada en tiempo razonable.
- Clasificar algoritmos por su complejidad nos permite **elegir la mejor solución** según el tamaño de entrada y los recursos disponibles.

Referencias

- Documentación oficial de Python sobre `time complexity`
- Big-O Cheat Sheet – Complexity chart
- Problema P vs NP explicado por Clay Mathematics Institute