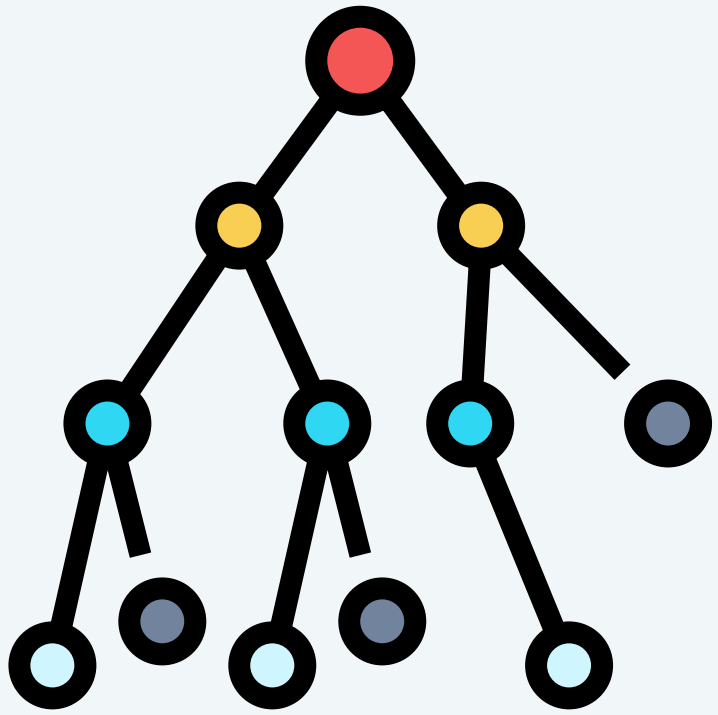
Cómo diseñar algoritmos más eficientes

Domina el arte de resolver problemas con velocidad y precisión



PABLO DEL ÁLAMO

Introducción

La eficiencia de tus algoritmos puede ser la diferencia entre un software rápido y funcional, o uno lento y poco eficiente.

En este carrusel aprenderás los principios clave para diseñar algoritmos más eficientes, junto con ejemplos prácticos.

PABLO DEL ÁLAMO

¿Qué significa que un algoritmo sea eficiente?

Un algoritmo eficiente minimiza los recursos que utiliza (tiempo y memoria) para resolver un problema.

- Tiempo: ¿Qué tan rápido se ejecuta?
- Espacio: ¿Cuánta memoria necesita?

PABLO DEL ÁLAMO

La importancia de la eficiencia

- Velocidad: Los usuarios esperan respuestas rápidas, especialmente en aplicaciones en tiempo real.

- Escalabilidad: Un buen algoritmo debe funcionar bien con millones de datos, no solo con pocos.

- Coste: Más eficiencia equivale a menos gasto.

PABLO DEL ÁLAMO

Big-O Notation: La clave para medir eficiencia

La notación Big-O mide cómo se comporta un algoritmo cuando el tamaño de los datos crece.

- O(1): Tiempo constante.
- O(n): Tiempo lineal.
- O(n^2): Tiempo cuadrático.

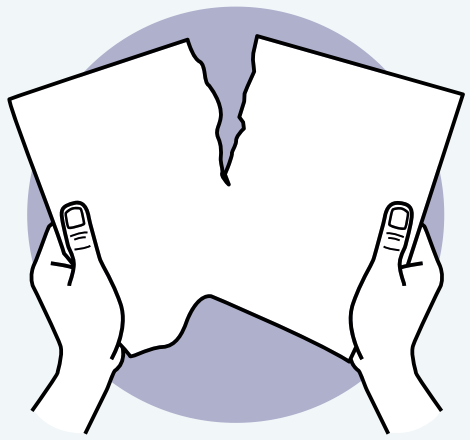
- Ejemplo: Buscar un elemento en una lista ordenada es más rápido con una búsqueda binaria O(log n) que con una búsqueda lineal O(n).

PABLO DEL ÁLAMO

Divide y vencerás

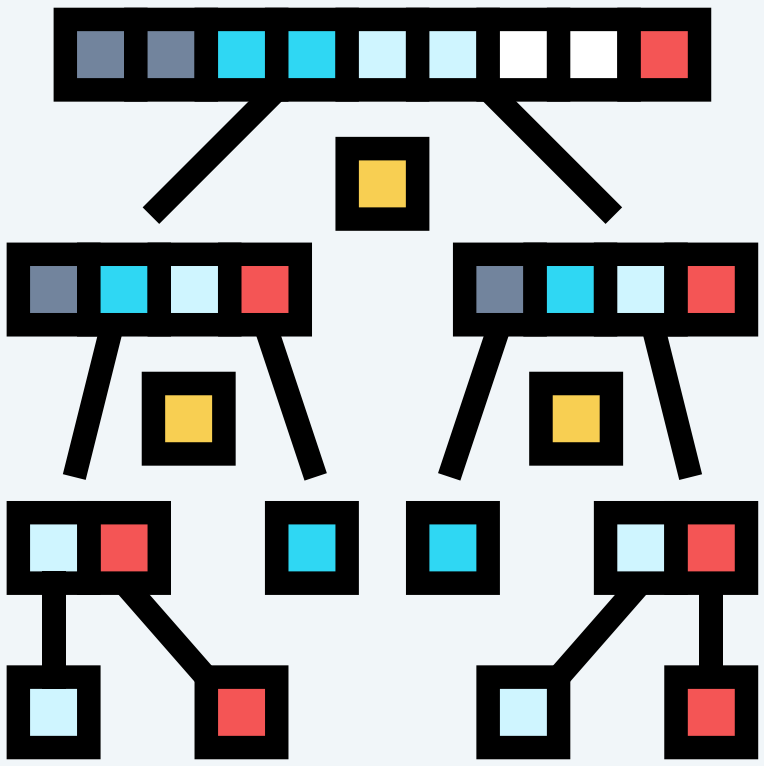
Un enfoque fundamental para mejorar la eficiencia:

- Divide el problema en subproblemas más pequeños.
- Resuelve cada uno de forma eficiente.
- Combina las soluciones.



PABLO DEL ÁLAMO

Ejemplo: El algoritmo de ordenamiento rápido (QuickSort) utiliza esta estrategia para alcanzar un rendimiento promedio de O(n log n).

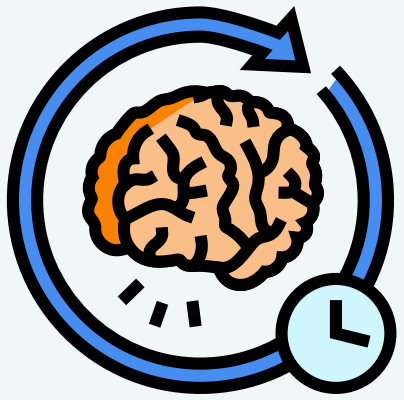


PABLO DEL ÁLAMO

Evita cálculos repetidos: Usa memoization

La memoización almacena resultados de cálculos previos para evitar repeticiones.

- Ejemplo: En problemas de programación dinámica como el cálculo de Fibonacci, puedes reducir $O(2^n)$ a $O(n)$ al guardar los resultados de cada subproblema.



PABLO DEL ÁLAMO

Escoge las estructuras de datos correctas

El rendimiento de tu algoritmo depende de las estructuras que utilices:

- Listas enlazadas para inserciones rápidas.
- HashMaps para búsquedas en tiempo constante O(1).
- Árboles balanceados para búsquedas rápidas con ordenación.

PABLO DEL ÁLAMO

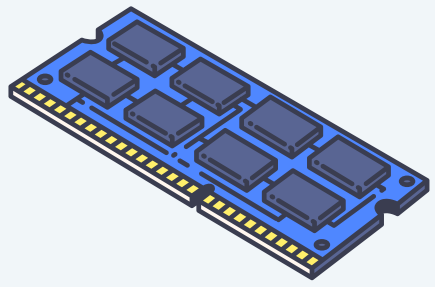
Ejemplo: Usar un HashMap para contar elementos es más eficiente que recorrer una lista varias veces.

PABLO DEL ÁLAMO

Reduce el espacio: Usa algoritmos in-place

Siempre que sea posible, diseña algoritmos que operen directamente sobre los datos de entrada, para minimizar el uso de memoria.

Ejemplo: El algoritmo de ordenamiento QuickSort in-place utiliza solo una pequeña cantidad de memoria adicional, a diferencia de MergeSort, que requiere espacio adicional.



PABLO DEL ÁLAMO

Iteración vs Recursión

Aunque la recursión siempre se ve elegante, a veces es menos eficiente debido al consumo de memoria de las llamadas anidadas.

Ejemplo: Calcular el factorial de un número con un bucle iterativo consume menos memoria que una solución recursiva.

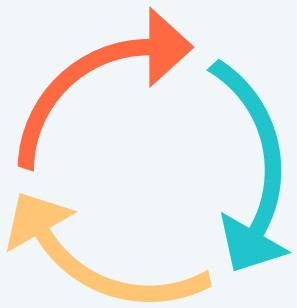
PABLO DEL ÁLAMO

Optimiza bucles anidados

Los bucles anidados pueden disparar la complejidad a $O(n^2)$ o peor. Ejemplo:

- Usar un HashMap para eliminar un bucle interno en lugar de comparar todos los elementos.

- Cambiar un bucle por un cálculo matemático directo.



PABLO DEL ÁLAMO

Algoritmos greedy

Los algoritmos greedy toman decisiones óptimas en cada paso.

Ejemplo: Para el problema de cambio de monedas, elegir siempre la moneda más grande disponible puede ser una solución óptima.

PABLO DEL ÁLAMO

Usa algoritmos probados

No siempre tienes que reinventar la rueda.

Aprovecha algoritmos optimizados ya existentes:

- Búsqueda binaria: Para encontrar elementos en listas ordenadas.

- Dijkstra: Para encontrar el camino más corto en un grafo.

PABLO DEL ÁLAMO

Complejidad amortizada

Algunos algoritmos tienen un coste elevado ocasional, pero son eficientes en promedio. Ejemplo:

- Insertar elementos en un array dinámico, como en ArrayList. El coste de redimensionar es alto, pero ocurre con poca frecuencia.

PABLO DEL ÁLAMO

Pruebas y análisis de rendimiento

Prueba siempre tus algoritmos con diferentes tamaños de datos y usa herramientas de profiling como:

- PyCharm Profiler para Python.
- JProfiler para Java.

Estas herramientas te ayudarán a identificar cuellos de botella en tiempo o espacio.

PABLO DEL ÁLAMO

Ejemplo práctico: Contar palabras únicas

Enfoque ineficiente:

- Usar dos bucles para comparar todas las palabras.

Enfoque eficiente:

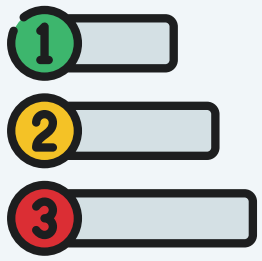
- Usar un HashSet para almacenar palabras únicas con una complejidad de O(n).

PABLO DEL ÁLAMO

Mejorando un algoritmo: Ordenación

Los típicos algoritmos de ordenación son:

- Burbuja ($O(n^2)$): Compara cada par.
- MergeSort ($O(n \log n)$): Divide y conquista.
- QuickSort ($O(n \log n)$): Opta por un pivote para dividir más eficientemente.



PABLO DEL ÁLAMO

Principios clave para recordar

- Entiende el problema antes de picar código.
- Analiza la complejidad antes de implementar.
- Escoge estructuras de datos adecuadas.
- Usa algoritmos existentes si es posible.

PABLO DEL ÁLAMO

Conclusión

Diseñar algoritmos eficientes es clave para resolver problemas complejos con rapidez y optimizar recursos.

Elegir la estructura de datos adecuada, analizar la complejidad y aplicar técnicas como memoización o programación dinámica, te permitirá mejorar el rendimiento de tus soluciones.

No siempre necesitas reinventar la rueda: aprovecha algoritmos probados y herramientas de profiling.

La eficiencia no solo es un reto técnico, sino una habilidad que te diferencia como desarrollador/a.

PABLO DEL ÁLAMO

¿Te ha resultado útil?

- Comparte esta guía con tu equipo o amigos desarrolladores.
- Guárdala para tenerla siempre a mano.
- ¡Dale un like o comenta si tienes preguntas!

PABLO DEL ÁLAMO