Análisis Big O

Braian Felipe Ramírez Ortiz
Fundación Universitaria Konrad Lorenz

***Resumen*—Se proporcionaron 15 algoritmos diferentes para ser analizados con la notación Big O. Este taller se llevó a cabo de forma escrita, teniendo como parámetro escribir de color rojo los condicionales y ciclos, de color azul las variables y los demás elementos en color negro. Para completar este taller, se analizó línea por línea para encontrar cuál sería el peor escenario en el cual el algoritmo sería ejecutado.**

I. Introducción

Se proporcionaron 15 algoritmos diferentes para ser analizados con la notación Big O. Este taller se llevó a cabo de forma escrita, teniendo como parámetro escribir de color rojo los condicionales y ciclos, de color azul las variables y los demás elementos en color negro. Para completar este taller, se analizó línea por línea para encontrar cuál sería el peor escenario en el cual el algoritmo sería ejecutado.

II. Resultados

Al analizar los códigos uno por uno, se logró evidenciar que su comportamiento de ejecución varía según la cantidad de ciclos, condicionales y funciones que estos contengan, determinando así cuál sería el escenario que consumiría más tiempo y memoria en su ejecución al ejecutarse.

Con ela análisis del Big O se puede obterner uno de los siguentes resultados aca descritos:

1. O(1) (tiempo constante): Significa que el algoritmo ejecuta en tiempo constante, independientemente del tamaño de la entrada. Es el caso ideal y óptimo.
2. O(log n) (logarítmico): Indica que la complejidad crece logarítmicamente con el tamaño de la entrada. Suelen ser características de algoritmos eficientes en árboles balanceados o búsquedas binarias.
3. O(n) (lineal): El tiempo de ejecución o uso de espacio del algoritmo es proporcional al tamaño de la entrada. Es común en algoritmos de búsqueda lineal o recorridos de estructuras de datos.
4. O(n log n) (linealítmico): Es común en algoritmos de ordenamiento eficientes como merge sort, quicksort, y algoritmos de árboles balanceados.
5. O(nˆ2) (cuadrático): El tiempo de ejecución o uso de espacio es proporcional al cuadrado del tamaño de la entrada. Pueden ser características de algoritmos de comparación ineficientes o problemas de búsqueda en matrices bidimensionales.
6. O(2ˆn) (exponencial): La complejidad crece de manera exponencial con respecto al tamaño de la entrada. Suelen asociarse con algoritmos de fuerza bruta.
7. O(n!) (factorial): Representa una complejidad factorial, donde el tiempo de ejecución o uso de espacio crece de manera factorial con respecto al tamaño de la entrada. Es común en algoritmos con permutaciones o combinaciones.
8. O(m + n): Representa una complejidad que depende de dos variables distintas, m y n, que pueden ser el tamaño de diferentes conjuntos de datos o el tamaño de diferentes operaciones en un algoritmo.
9. O(m * n): Representa una complejidad en la que el tiempo de ejecución o el uso de espacio del algoritmo crece de manera proporcional al producto de dos variables, m y n, en función de su tamaño. Estas variables, m y n, suelen representar diferentes parámetros o tamaños de entrada relevantes para el algoritmo

III. Conclusión

Al finalizar el análisis de los 15 códigos, se logra reconocer la complejidad de cada una de las líneas que componen el algoritmo, haciendo más fácil su estudio. Así, se determina en cuáles de ellas se encuentra la mayor complejidad, obteniendo el caso esxtremo en la ejecución del mismo. Ademas podemos determinar predicción del desempeño del algoritmo al igual que detrminar una optimización y escalabilidad del mismo.

IV. Anexos

En el repositorio en el cual se encuentra anexo este documento, encontrará a su disposición un archivo PDF que muestra el análisis realizado a cada uno de los 15 algoritmos propuestos en el taller 1.