Manuel Arturo Chinchilla Sánchez – 2013009344
Resumen #7

Lectura: Big O

Notación O grande

La notación O grande (con mayúscula O, no un cero), también llamada símbolo de Landau, es un simbolismo utilizado en la teoría de la complejidad, la informática y las matemáticas para describir el comportamiento asintótico de las funciones. Básicamente, te dice qué tan rápido una función crece o disminuye.

Por ejemplo, al analizar algún algoritmo, uno puede encontrar que el tiempo (o el número de pasos) que se necesita para completar un problema de tamaño n viene dada por $T(n) = 4n^2 - 2n + 2$. Si ignoramos constantes (lo cual tiene sentido porque depende del hardware en específico sobre el cual el programa está corriendo) y las condiciones de crecimiento más lento, podríamos decir "T(n) crece en el orden de n^2" y escribir: $T(n) = O(n^2)$.

Para la definición formal, supongamos que f(x) y g(x) son dos funciones definidas en un subconjunto de los números reales. Escribimos f(x) = O (g(x)).

Esta definición es la utilizada en ciencias de la computación (donde normalmente sólo se consideran funciones positivas con un número natural n como argumento; los valores absolutos pueden ser ignorados).

Aquí está una lista de clases de funciones que se encuentran comúnmente en el análisis de algoritmos. Las funciones de crecimiento más lentas se muestran primero. c es una constante arbitraria.

Notación	**Nombre**
$O(1)$	Constante
$O(\log(n))$	Logarítmica
$O((\log(n))^c)$	Polilogarítmica
$O(n)$	Lineal
$O(n^2)$	Cuadrática
$O(n^c)$	Polinomial
$O(c^n)$	Exponencial

Tenga en cuenta que $O(n^c)$ y $O(c^n)$ son muy diferentes. Este último crece mucho, mucho más rápido, no importa cuán grande sea la constante c. Una función que crece más rápido que cualquier potencia de n se llama superpolinomial. Uno que crece más lento que una función exponencial de la forma c^n se llama subexponencial.

Comprendiendo la O grande.

Eficiencia cubre una gran cantidad de recursos, incluyendo:

- Uso (tiempo) de CPU
- Uso de memoria
- Uso del disco
- Uso de la red

Tenga cuidado de diferenciar entre:

1. Rendimiento: cuánto tiempo / memoria / disco /... realmente se utiliza cuando se ejecuta un programa. Esto depende de la máquina, compilador, etc., así como el código.

2. Complejidad: ¿cómo está resolviendo los requerimientos de recursos de un programa o escala de algoritmo, es decir, qué ocurre cuando el tamaño del problema que se solucionó se hace más grande?

La complejidad afecta el rendimiento, pero no a la inversa.

Las funciones/procedimientos pueden realizar diferentes números de operaciones, en función del valor de un parámetro. Estos parámetros (número de elementos) se denomina el tamaño del problema/tamaño de la entrada.

Cuando estamos tratando de encontrar la complejidad de la función / procedimiento / algoritmo / programa, no estamos interesados en el número exacto de las operaciones que se están realizando. En su lugar, nos interesa la relación entre el número de operaciones para el tamaño del problema.

Por lo general, estamos usualmente interesados en el peor de los casos: ¿cuál es el número máximo de operaciones que pueden llevarse a cabo para un tamaño determinado problema?

Cómo determinar Complejidades

Secuencia de los estados

Si cada declaración es "simple" (sólo implica operaciones básicas), entonces el tiempo para cada declaración es constante y el tiempo total es también constante: O (1).

If-Then-Else

Aquí, ya sea el bloque 1 se ejecutará, o el bloque 2 ejecutará. Por lo tanto, el tiempo del peor caso es la más lenta de las dos posibilidades:

max(tiempo(bloque 1), tiempo (bloque 2)).

Loops

El bucle se ejecuta N veces, por lo que la secuencia de sentencias también ejecuta N veces.

Declaraciones con llamadas de función / procedimiento

Cuando una instrucción implica una llamada de función / procedimiento, la complejidad de la declaración incluye la complejidad de la función / procedimiento.