Complejidad de Algoritmos

El análisis de algoritmos provee estimaciones teóricas para los recursos que necesita cualquier algoritmo y se basan en criterios como: eficiencia, portabilidad, eficacia y robustez. El análisis de complejidad está relacionado con la eficiencia del programa. La eficiencia mide el uso de los recursos del computador por un algoritmo tal como el tiempo de cálculo para ejecutar las operaciones (complejidad en tiempo) y el espacio de memoria para contener y manipular el programa más los datos (complejidad en espacio). Así, el objetivo del análisis de complejidad es cuantificar las medidas físicas en tiempo de ejecución y espacio de memoria para comparar distintos algoritmos que resuelven un mismo problema.

En un análisis teórico de algoritmos, es común calcular su complejidad en un sentido asintótico, es decir, para un tamaño de entrada n suficientemente grande. La notación "O grande" (Big O) es una notación para expresar esto. Por ello, estudiar la complejidad del peor caso, mejor caso, y caso promedio resulta importante en las Ciencias de la Computación. La complejidad del algoritmo en el peor caso resulta de tomar el máximo tiempo (complejidad máxima) de ejecución de un algoritmo entre todas las instancias del problema de tamaño n; la complejidad en el caso promedio es la esperanza matemática del tiempo de ejecución del algoritmo para entradas de tamaño n, y la complejidad del mejor caso es el menor tiempo en que se ejecuta el algoritmo para entradas de tamaño n. Por defecto se toma la complejidad del peor caso como medida de complejidad T(n) del algoritmo. Por ejemplo, una búsqueda binaria se dice que se ejecuta en un número de pasos proporcional al logaritmo de la longitud de la estructura de datos de la búsqueda, ó en O(log(n)), dicho coloquialmente como tiempo logarítmico.

Definición Formal

* Sean f y g dos funciones definidas como un subconjunto de los números reales, se dice que:

$f(x) = O(g(x))$ con $x \in \infty$

f es de orden g si y solo si existe una constante positiva $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que $\forall n > n_0$ se cumpla $f(n) < c.g(n)$. La relación O denota una dominancia de funciones en donde la función f está acotada superiormente por un múltiplo de la función g. Entonces, la expresión $f = O(g)$ refleja que el orden de crecimiento asintótico de la función f es inferior o igual al de la función g.

* Sean f y g dos funciones definidas como un subconjunto de los números reales, se dice que f y g tienen igual orden de crecimiento $f = \Theta(g)$ si y solo si existe $c, d \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que $\forall n > n_0$ se cumpla $d.g(n) \leq f(n) \leq c.g(n)$

Análisis de Complejidad

Se considerará la entrada de datos al programa como el factor para el cálculo de su complejidad. Así, la complejidad $T(n)$ de un algoritmo es de $O(f(n))$ si $T, f : \mathbb{N} \to \mathbb{R}^+$ y $\exists\ c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que $\forall n > n_0$ se cumpla $T(n)$

Regla de la Suma: Sean $T_1(n)$ y $T_2(n)$ las funciones de complejidad para ejecutar dos instrucciones P_1 y P_2 de complejidad $T_1(n) = O(f(n))$ y $T_2(n) = O(g(n))$ respectivamente, la secuencia de instrucciones P_1 y P_2 es $T_1(n) + T_2(n) = O(Max f(n), g(n))$

Regla del Producto: Se define como:
$T_1(n) = O(f(n)) \wedge T_2(n) = O(g(n)) = O(g(n)) \Rightarrow T_1(n).T_2(n) = O(f(n).g(n))$
Sean $c, d \in \mathbb{R}^+$ se puede derivar que:

- $T(n) = c \Rightarrow T(n) = O(1)$
- $T(n) = c + f(n) \Rightarrow T(n) = O(f(n))$
- $T_1(n) = c.f(n) \Rightarrow T_1(n) = O(f(n))$
- $T_1(n) = c.f(n) + d \Rightarrow T_1(n) = O(f(n))$
- $T_1(n) = O(n^k) \wedge T_2(n) = O(n^{k+1}) \Rightarrow T_1(n) + T_2(n) = O(n^{k+1})$
- $T(n) = c.n^d \Rightarrow T(n) = O(n^d)$
- $T(n) = P_k(n) \Rightarrow T(n) = O(n^k)$, donde $P_k(n)$ es un polinomio de grado $k \geq 0$
- $T_1(n) = Ln(n) \wedge T_2(n) = n^k \wedge k > 1 \Rightarrow T_1(n) + T_2(n) = O(n^k)$
- $T_1(n) = r^n \wedge T_2(n) = P_k(n) \wedge r > 1 \Rightarrow T_1(n) + T_2(n) = O(r^n)$

La función de complejidad en tiempo de una instrucción de asignación simple es una constante independiente del tamaño de la entrada de datos $\Rightarrow O(1)$. La operación de E/S es $T(n) = c$, por lo tanto es $O(1)$. De la regla de la suma se deriva que la complejidad de una secuencia de k instrucciones con $T_i(n) = O(f_i(n))$ es $\sum_{i=1}^{k} T_i(n) = O(Max f_1(n), f_2(n), ..., f_k(n))$. La complejidad de una selección viene dada por la complejidad de ejecutar la condición y las instrucciones. Del mismo modo, en un ciclo iterativo la complejidad es la sobre del cuerpo de iteración más la evaluación de la condición de terminación del ciclo.

Si un programa tiene procedimientos recursivos, se calcula cada tiempo de ejecución a la vez y aplicando la ecuación de recurrencia correspondiente.

esmitt ramírez/2014