Tiempo de ejecución de un programa

Cuando resolvemos un problema frecuentemente nos enfrentamos con la selección de un algoritmo entre varios. En qué debemos basar nuestra selección?. Hay, por lo general, dos objetivos contradictorios:

1. Deseamos un algoritmo fácil de entender, codificar y depurar.
2. Deseamos un algoritmo que haga uso eficiente de los recursos del ordenador, especialmente, uno que sea lo más rápido posible.

La rapidez de ejecución de un programa dependerá de las necesidades del sistema que se está construyendo. Usualmente se empieza por un prototipo simple sobre el cual pueden efectuarse mediciones y simulaciones.

En general, el tiempo de ejecución de un programa depende de factores tales como:

1. Los datos de entrada.
2. La calidad del código generado por el compilador usado para crear el programa objeto.
3. La naturaleza y velocidad de las instrucciones en la máquina usada para ejecutar el programa.
4. La complejidad temporal del algoritmo usado en el programa.

El hecho que el tiempo de ejecución dependa de la entrada indica que el tiempo de ejecución de un programa debe estar definido como una función de los datos de entrada, que puede especificarse con el "tamaño" de la entrada. Así, por ejemplo, en un programa de ordenamiento el tamaño natural de medida para la entrada es el número de elementos a ordenar.

Es usual indicar con $T(n)$ el tiempo de ejecución de un programa para una entrada de tamaño n. Las unidades de $T(n)$ pueden dejarse sin especificar, pero puede pensarse que es el número de instrucciones ejecutadas en un ordenador ideal.

Los factores 2 y 3 implican que $T(n)$ no puede expresarse en unidades estándar de tiempo real tal como segundos. En su lugar, se hará referencia a que "un algoritmo es proporcional a n^2", por ejemplo. La constante de proporcionalidad se deja sin especificar.

Notación O-grande y Ω-grande

Para hacer referencia a la tasa de crecimiento de funciones usa una notación especial.

Decimos que $T(n)$ es $O(f(n))$ si hay constantes c y n_0 tales que $T(n) \leq c\ f(n)$ para todo $n \geq n_0$. Un programa cuyo tiempo de ejecución es $O(f(n))$ se dice que tiene una tasa de crecimiento de $f(n)$. En esta notación $f(n)$ es una cota superior de la tasa de crecimiento de $T(n)$.

Ejemplo: La función $T(n) = 3\ n^3 + 2\ n^2$ es $O(n^3)$, ya que para $n_0=0$ y $c=5$ se obtiene: $3\ n^3 + 2\ n^2 \leq 5\ n^3$ para todo $n \geq 0$.

Para especificar una cota inferior en la tasa de crecimiento de T(n) se usa la notación Ω(g(n)). En este caso, T(n) es Ω(g(n)) si existe una constante c (positiva) tal que T(n) ≥ c g(n).

Ejemplo: La función $T(n) = n^3 + 2n^2$ es $\Omega(n^3)$, ya que para c=1 se obtiene: $n^3 + 2n^2 \geq n^3$ para todo $n \geq 0$.

Para comparar programas pueden usarse sus funciones de crecimiento despreciando las constante de proporcionalidad. Por tanto, un programa $O(n^2)$ es mejor que uno con complejidad $O(n^3)$.

Cálculo del tiempo de ejecución de un programa

Es un problema matemático complejo.Para simplificarlo usamos algunas reglas:

Si $T_1(n)$ y $T_2(n)$ son los tiempos de ejecución de dos fragmentos P_1 y P_2 de un programa, y si $T_1(n)$ es O(f(n)) y $T_2(n)$ es O(g(n)), entonces $T_1(n) + T_2(n)$ es O(max(f(n),g(n))).

Si $T_1(n)$ y $T_2(n)$ son O(f(n)) y O(g(n)) respectivamente, entonces $T_1(n)\ T_2(n)$ es O(f(n)g(n)).

Ejemplo: Se desea analizar la complejidad temporal del algoritmo de ordenación según el método de la burbuja.

```
        void burbuja (int a[], int n)
        {
          int i, j, temp;

(1)       for (i=0; i<n-1; i++)
(2)         for (j=n-1; j > i; j--)
(3)         if (a[j-1] > a[j]) {
(4)           temp=a[j-1];
(5)           a[j-1]=a[j];
(6)           a[j]=temp;
            }
        }
```

La medida del tamaño de la entrada está determinado por el número de elementos n a ordenar.

Cada instrucción de asifnación toma un tiempo constante, independiente del tamaño de los datos. Esto es, las instrucciones (4), (5) y (6) toman un tiempo O(1) cada una. Por la regla de la suma, el tiempo total de este grupo de instrucciones es O(1).

La sentencia if requiere un tiempo O(1). No sabemos si el cuerpo de la instrucción if (4-6) se llevará a cabo. En el peor caso se ejecutará, por lo cual las instrucciones (3)-(6) tienen una complejidad O(1).

Para un ciclo (for) la regla general es que el tiempo es la suma del tiempo necesario para ejecutar el cuerpo del ciclo para cada iteración del mismo. En el ciclo interno se cuenta como mínimo O(1) por cada iteración. El número de iteraciones es n-i, así que

por la regla del producto, el tiempo gastado en el ciclo (2)-(6) es O((*n*-i)x1) lo cual es O(*n*-i).

El ciclo exterior se realiza *n*-1 veces, así que el tiempo total de ejecución del programa es:

$$\sum_{i=1}^{n-1}(n-i) = n(n-1)/2 = n^2/2 - n/2$$

que es O(n^2).

Complejidad de los algoritmos de ordenación

Haciendo un análisis comparativo de los diferentes métodos de ordenación encontramos que:

Método de ordenación	Complejidad
Burbuja	O(n^2)
Selección	O(n^2)
Inserción	O(n^2)
Shell	O($n^{1.2}$)
Mezcla	O($n \log n$)
Quicksort	O($n \log n$)