LA EFICIENCIA DE LOS ALGORITMOS

Gloria Inés Giraldo Echeverri
Ingeniera Industrial Universidad
Tecnológica de Pereira
Magister en Ingeniería de Sistemas
Universidad Nacional
Profesora Asociada Depto. de Sistemas
Universidad Nacional

INTRODUCCION

Al enfrentarse a la tarea de desarrollo de software es común encontrarse en la situación de tener que resolver un problema dado a través del algoritmo correcto. Sin embargo, algunos algoritmos correctos son mejores o más eficientes que otros. Qué significa la eficiencia de un algoritmo ? . El presente artículo, pretende de una forma sencilla responder esta pregunta.

Eficiencia se refiere al consumo de algún recurso y normalmente es una medida de como el recurso es consumido. Los dos recursos normalmente asociados con la eficiencia de los algoritmos son tiempo y memoria requeridos durante el proceso de ejecución.

En general, el tiempo y el espacio requeridos para la ejecución de un algoritmo dependen del tamaño de la entrada. Supongamos que n representa el tamaño de la entrada. A fin de analizar la eficiencia de un algoritmo debemos resolver el siguiente problema: Dado un algoritmo y una entrada de tamaño n, estimar el tiempo y el monto de almacenamiento requeridos para su ejecución.

El almacenamiento requerido puede medirse por la cantidad de memoria usada para almacenar objetos tales como: variables, constantes, arreglos. El tiempo que toma un algoritmo en ser ejecutado se mide contando el número de instrucciones a ser ejecutadas con una entrada de tamaño n. Lo que se pretende es tener instrumentos de análisis que nos permitan evaluar la eficiencia de los algoritmos y tomar las medidas correctivas antes de llevarlos al computador.

Básicamente se tratará la medida de tiempo, que es la que hace más crítico el desarrollo de software.

Es natural esperar que para cualquier algoritmo de búsqueda toma mas tiempo, una búsqueda en una estructura de cien registros que la búsqueda en una estructura de diez. Si se selecciona un algoritmo de búsqueda específico y una clave, el tamaño de la estructura de búsqueda determinará el número de registros que el algoritmo examinará. Por lo tanto, decimos que el número de registros examinados es una función del tamaño del problema. Es común en el campo de las Ciencias de la Computación denotar el tamaño del problema por n.

TABLA N.1

Crecimiento de f(n) con n

f(n)	n=3	n=10	n=30	n=100	n=300	n=1000
*lg n	1.6	3.3	4.9	6.6	8.2	10
n	3	10	30	100	300	1000
n lg n	4.8	33	147	664	2469	9966
n^2	9	100	900	10000	90000	10^6
n^3	27	1000	27000	10^6	$2.7*10^7$	10^9
2^n	8	10^{24}	10^9	10^{30}	10^{90}	10^{300}
10^n	1000	10^{10}	10^{30}	10^{100}	10^{300}	10^{1000}

*lg: logaritmo en base 2

TABLA N.2

Límite del tamaño del problema determinado por la rata de crecimiento de la función

Algoritmo	f(n) Complejidad de tiempo	Máximo tamaño del problema 1 seg.	1min.	1 hora
A1	n	1000	$6*10^4$	$3.6*10^6$
A2	n lg n	140	4893	$2.0*10^5$
A3	n^2	31	244	1897
A4	n^3	10	39	153
A5	2^n	9	15	21

FAMILIAS DE FUNCIONES

Medir la eficiencia de un algoritmo en forma exacta es prácticamente imposible, puesto que existen diversos factores que son difíciles de cuantificar, como por ejemplo la eficiencia del compilador en que se corre el programa.

La eficiencia de un algoritmo se mide tratando de encontrar una expresión que dependa del parámetro n y que pueda usarse para aproximar el desarrollo del mismo. Se determina a que familia de funciones pertenece la función; esto nos permite ignorar todos los coeficientes de una función, así como, todos los términos excepto el término dominante de la función. Por ejemplo, si se tiene un algoritmo que requiere en promedio $88n^2 + 7n + 11$ pasos en promedio para ser ejecutado con una entrada de tamaño n. Cuando la entrada llega a ser muy grande el número de pasos $(88n^2 + 7n + 11)$ requeridos en la ejecución del algoritmo es dominada por el primer término, $88n^2$. Decimos entonces que la función es orden de n^2 lo cual se denota como $O(n^2)$.

La estimación de la eficiencia del algoritmo se conoce técnicamente como orden del algoritmo y se escribe mediante la notación-O.

Las diferencias en el crecimiento entre las familias de funciones más representativas en el desempeño de algoritmos se pueden apreciar en la tabla N.1 (Smith [5]). ***Ver tabla 1***

Se puede observar que la función lineal (n) se incrementa en proporción al incremento en n, mientras que la función cuadrática (n^2) se incrementa mucho mas rápidamente. En la última fila de la tabla se observa que la función 10^n crece mucho mas rápidamente que las otras funciones.

Según Aho[1]: "La complejidad de tiempo es el número de unidades requeridas para procesar una entrada de tamaño n". La tabla N.2 (Aho [1]) muestra el tamaño del problema que puede resolverse en un segundo, un minuto y una hora para cinco algoritmos con diferentes clases de complejidad. ***Ver tabla 2***

Puede observarse que en una unidad de tiempo igual a un segundo el algoritmo A1 puede procesar una entrada de tamaño 1000, mientras que el algoritmo A5 puede procesar en un segundo una entrada de tamaño 9.

CONDICIONES COMPARATIVAS

Las condiciones bajo las cuales los algoritmos son más comunmente comparados y evaluados son el mejor caso y el peor caso. Para un algoritmo de búsqueda secuencial, un registro es examinado en el mejor de los casos y n registros son examinados en el peor de los casos.

Existe otra condición bajo la cual un algoritmo es evaluado y comparado: el caso promedio; éste es raramente tan fácil de cuantificar como el mejor caso o el peor caso; requiere un poco más de sofisticación matemática.

Consideremos el desempeño de un algoritmo de búsqueda secuencial para el caso promedio: la llave de búsqueda tiene la misma probabilidad de ser encontrada en cualquier parte de la estructura. Se requiere examinar un registro para encontrar el primero, dos registros para encontrar el segundo; y así examinar los n registros para localizar el último.

El caso promedio requiere el promedio de los números de 1 hasta n, el cual es la suma de los números de 1 hasta n dividido por n. La suma de los números de 1 hasta n es: n*(n+1)/2. Por lo tanto el promedio de 1 hasta n es (n+1)/2.

El caso promedio cae entre el mejor de los casos (1) y el peor de los casos (n).

Es posible expresar el número de registros examinado por la búsqueda secuencial como una función de n en los tres casos. Se dice entonces que es O(n).

CALCULO DEL TIEMPO DE EJECUCION DE UN ALGORITMO

Antes de ilustrar con un ejemplo para el cálculo del tiempo en la ejecución de un algoritmo se presentarán las dos propiedades básicas de la notación-O (Smith[5]):

-Propiedad aditiva de la notación-O . Si se tiene un algoritmo Pi seguido por otro Pj, tal que: Pi se ejecuta en un tiempo Ti = O(f(m)) y Pj se ejecuta en un tiempo Tj = O(g(n)). Entonces la complejidad para ejecutar Pi seguido de Pj es:

$$T = O(\text{máx}(f(m),g(n))).$$

-Propiedad multiplicativa de la notación-O. Si se tiene un algoritmo P que contiene partes Pi y subpartes Pj tal que: los pasos externos Pi se ejecutan en un tiempo Ti = O(f(m)) y los pasos internos Pij se ejecutan en un tiempo Tij = O(g(n)) para cada paso externo Pi. Entonces la complejidad total de ejecutar P está determinada por el producto:

$$T = O((f(m)*g(n))$$

Se ilustrará el cálculo de tiempo con el algoritmo de clasificación de burbuja (Aho [2]) que ordena un arreglo de enteros de menor a mayor.

```
Procedure Burbuja(var A: array[1..n] of
integer);
   var
     i,j,temp: integer;
   begin
     for i:=1 to n-1 do
       for j:=n downto i+1 do
         if A[j-1]> A[j] then begin
             temp   := A[j-1];
             A[j-1] := A[j];
             A[j]   := temp
         end
   end;
```

Como las proposiciones están anidadas se comienza de adentro hacia afuera.

El caso promedio requiere el promedio de los números de 1 hasta n, el cual es la suma de los números de 1 hasta n dividido por n.

En la estructura **if,** probar la condición requiere O(1), el cuerpo de esta estructura se ejecutará en el peor de los casos y requiere un tiempo de O(1).

Continuando hacia afuera de la estructura **if,** se encuentra la estructura **for** controlada por j. Para el ciclo de esta estructura el número de iteraciones es n-i. Por la propiedad multiplicativa el tiempo total de esta estructura es O((n-i)*1), que es igual a O(n-i).

Continuando hacia afuera se encuentra la estructura **for** controlada por i. El número de iteraciones para esta estructura es n-1.

Por lo tanto, el tiempo total de ejecución del programa será la sumatoria desde i=1 hasta n-1 de (n-i) que es igual a n(n-1)/2, que es $O(n^2)$. Esto significa que para ejecutar este algoritmo se requiere de un tiempo proporcional al cuadrado del número de elementos que se van a clasificar. Es de anotar que existen varios algoritmos de clasificación con un tiempo de ejecución menor que n^2.

Para finalizar, se enfatizará la importancia de este tema con un aparte del prefacio del libro de Aho[2]: "...a medida que las grandes máquinas aumenten su velocidad, los programadores tendrán que tratar con problemas progresivamente mayores y, en consecuencia, la importancia de la complejidad de tiempo aumentará en vez de disminuír, con la aparición de nuevas generaciones de equipos".

CONCLUSIONES

La eficiencia de los algoritmos se refiere al tiempo y el espacio requeridos para su ejecución, dada una entrada de tamaño n.

Se necesita tener elementos de análisis que nos permitan detectar las deficiencias para tomar las medidas correctivas antes de llevar los algoritmos al computador.

La eficiencia de un algoritmo no se puede medir en forma exacta, se trata de encontrar una expresión que dependa del parámetro n (tamaño de la entrada) y que pueda usarse para aproximar el desarrollo del mismo. Esto se logra con la notación-O (notación de orden).

Con la aparición de nuevas generaciones de equipos no se resuelve el problema de la eficiencia, por lo tanto, el cálculo de complejidad para estimar la eficiencia de los algoritmos seguirá siendo un tema de gran importancia.

BIBLIOGRAFIA

1- AHO,A.V., HOPCROFT,J.E. Y ULLMAN,J.D. The Design and Analysis of Computer Algorithms. Addison Wesley Publishing Company, E.U.A., 1974, pp. 2-5.

2- AHO,A.V., HOPCROFT,J.E. Y ULLMAN,J.D. Estructura de Datos y Algoritmos. Addison Wesley Iberoamericana, S.A. Wilmington Delawre, E.U.A., 1988, pp. 21-27.

3- KNUTH,D.E. Algoritmos Fundamentales, Vol.I. Editorial Reverté S.A. Barcelona , España, 1980, pp. 112-115.

4- MILLER,P.L. Y MILLER, L.W. Programming by Design. Wadsworth Publishing Company, Belmont, California, E.U.A., 1987. Special Edition, pp. 478-483.

5- SMITH,H.F. Data Structures. Jarcourt Brace Jovanovich, Publishers. Orlando, E.U.A., 1987, pp. 14-21.