Análisis de algoritmos, complejidad en espacio y ejecución, y eficiencia de algoritmos.

Análisis de algoritmos

La característica básica que debe tener un algoritmo es que sea correcto, es decir, que produzca el resultado deseado en tiempo finito. Adicionalmente puede interesarnos que sea claro, que esté bien estructurado, que sea fácil de usar, que sea fácil de implementar y que sea eficiente.

Eficiencia de algoritmos

Entendemos por eficiencia de un algoritmo la cantidad de recursos de computo que requiere; es decir, cuál es su tiempo de ejecución y que cantidad de memoria utiliza. A la cantidad de tiempo que requiere la ejecución de un cierto algoritmo se le suele llamar coste en tiempo mientras que a la cantidad de memoria que requiere se le suele llamar coste en espacio. Es evidente que conviene buscar algoritmos correctos que mantengan tan bajo como sea posible el consumo de recursos que hacen del sistema, es decir, que sean lo más eficientes posible. Cabe hacer notar que el concepto de eficiencia de un algoritmo es un concepto relativo, esto quiere decir que ante dos algoritmos correctos que resuelven el mismo problema, uno es más eficiente que otro si consume menos recursos. Por tanto, podemos observar que el concepto de eficiencia y en consecuencia el concepto de coste nos permitirá comparar distintos algoritmos entre ellos.

Coste de los algoritmos iterativos

Las siguientes reglas facilitan el cálculo o el análisis del coste de los algoritmos iterativos en el caso peor.

1. El coste de una operación elemental es Θ.

2. Si el coste de un fragmento $F1$ es $f1$ y el de un fragmento $F2$ es $f2$ entonces el coste en caso peor del fragmento:

$F1;\ F2$

es:

$f1\ +\ f2$

3. Si el coste de $F1$ es $f1$ y el de $F2$ es $f2$ y el de evaluar B es g entonces el coste en caso peor del fragmento:

$if(B)\ F1;\ else\ F2$

es:

$g\ +\ máx\{f1, f2\}$

4. Si el coste de F durante la $i-ésima$ iteración es f_i y el coste de evaluar B es g_i y el número de iteraciones es h, entonces el coste en el caso peor del fragmento:

$$while\ (B)\ F;$$

es:

$T(n) = \sum_{i=0}^{h(n)} f_i(n) + g_i(n)$

Si $f\ =\ máx\{f_i + g_i\}\ entonces\ T\ =\ O\ (hf)$.

Complejidad en tiempo de ejecución y espacio

El tiempo que requiere un algoritmo para dar una respuesta, se divide generalmente en 3 casos.

- Peor Caso: caso más extremo, donde se considera el tiempo máximo para solucionar un problema.
- Caso promedio: caso en el cual, bajo ciertas restricciones, se realiza un análisis del algoritmo.
- Mejor caso: caso ideal en el cual el algoritmo tomará el menor tiempo para dar una respuesta.

El tiempo de ejecución de un algoritmo va a depender de diversos factores como son: los datos de entrada que le suministremos, la calidad del código

generado por el compilador para crear el programa objeto, la naturaleza y rapidez de las instrucciones máquina del procesador concreto que ejecute el programa, y la complejidad intrínseca del algoritmo. Hay dos estudios posibles sobre el tiempo:

1. Uno que proporciona una medida teórica (a priori), que consiste en obtener una función que acote (por arriba o por abajo) el tiempo de ejecución del algoritmo para unos valores de entrada dados.

2. Y otro que ofrece una medida real (a posteriori), consistente en medir el tiempo de ejecución del algoritmo para unos valores de entrada dados y en un ordenador concreto.

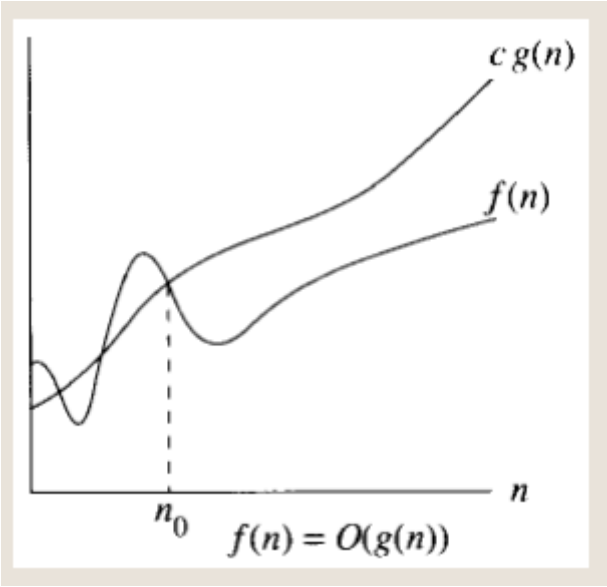
Ambas medidas son importantes puesto que, si bien la primera nos ofrece estimaciones del comportamiento de los algoritmos de forma independiente del ordenador en donde serán implementados y sin necesidad de ejecutarlos, la segunda representa las medidas reales del comportamiento del algoritmo. Estas medidas son funciones temporales de los datos de entrada.

La unidad de tiempo a la que debe hacer referencia estas medidas de eficiencia no puede ser expresada en segundos o en otra unidad de tiempo concreta, pues no existe un ordenador estándar al que puedan hacer referencia todas las medidas. Denotaremos por $T(n)$ el tiempo de ejecución de un algoritmo para una entrada de tamaño n.

Teóricamente $T(n)$ debe indicar el número de instrucciones ejecutadas por un ordenador idealizado. Debemos buscar por tanto medidas simples y abstractas, independientes del ordenador a utilizar. Para ello es necesario acotar de alguna forma la diferencia que se puede producir entre distintas implementaciones de un mismo algoritmo, ya sea del mismo código ejecutado por dos máquinas de distinta velocidad, como de dos códigos que implementen el mismo método.

Notación O

$f(n) = O(g(n)), g(n)$ es una cota superior de $f(n)$ Dada una función $g(n)$, denotamos como $O(g(n))$ al conjunto de funciones tales que: $O(g(n)) = \{f: N \to R + \mid \exists\ c$ constante positiva y N_o∈ N : f(n) ≤ cg(n), ∀ n ≥N_o }


c g(n)
f(n)
n
n0
f(n) = O(g(n))

Referencias

1. Estructura de datos: Algoritmos, abstracción y objetos, Luis Joyanes, McGraw-Hill, España, 1998, Pag. 555.
2. Estructura de datos: Algoritmos, abstracción y objetos, Luis Joyanes, McGraw-Hill, España, 1998, Pag. 554.
3. Estructura de datos: Algoritmos, abstracción y objetos, Luis Joyanes, McGraw-Hill, España, 1998, Pag. 553.
4. Brassard, G., y Bratley, P.: Fundamentos de algoritmia Madrid. Prentice-Hall, 1997.
5. Heileman, Gregory L.: Estructuras de datos, algoritmos, y programación orientada a objetos, Madrid, McGraw-Hill, 1997.