ESCUELA SUPERIOR POLITECNICA DE CHIMBORAZO

FACULTAD DE INFORMATICA Y ELECTRONICA

ESCUELA DE INGENIERIA EN SISTEMAS

Integrantes:

Wilmer Aldas (6035)

Claudio Pilataxi (6042)

Richard Manotoa (6016)

**NOTACIONES ASINTÓTICAS**

**Importancia en Análisis de Algoritmos**

El desarrollo de la computación y el trabajo con algoritmos ha generado varias interrogantes a los ingenieros que se dedican a trabajar en el campo de las ciencias de la computación, por ejemplo

¿Cómo podemos caracterizar un algoritmo que resuelve determinado problema?

¿Cómo podemos comparar dos algoritmos que resuelven el mismo problema?

¿Existirá un algoritmo *ideal* para resolver cualquier problema computacional?

Consideremos un ejemplo práctico. Supongamos que necesitamos clasificar (ordenar) un grupo de personas según su edad, esto puede ser de menor a mayor o viceversa. Existen varios enfoques para ordenar datos, sin embargo algunos son más **complicados** de implementar.

La primera condición que debemos tener en cuenta al elegir nuestro algoritmo es la **Simplicidad** de su diseño.

Cuando usamos un programa para resolver un problema, como el de ordenar datos en este caso, siempre hay que tener en cuenta de que nuestro trabajo puede ser usado por otros ingenieros y programadores más adelante (o incluso por nosotros mismos). No es viable que luego de implementar nuestro ordenador, al revisarlo 6 meses después no podamos entender que hicimos!.

Es por esto que la segunda condición que debemos considerar es la **Claridad** del algoritmo, a modo que sea factible documentarlo de manera eficiente.

Hemos mencionado detalles sobre el diseño del algoritmo pero una de las características más importantes que debemos considerar es la **Eficiencia** del mismo. El desempeño del algoritmo se puede describir de varias maneras lo cual depende del problema en sí, por ejemplo en una base de datos es usual que se busque medir el número de *consultas* que puede atender en un determinado tiempo; en un sistema embebido (como el software de los equipos celulares) resulta indispensable medir:

* espacio en memoria que consume la aplicación.
* tráfico de datos que maneja la aplicación con el exterior.
* ya que estas características están muy relacionadas con la duración de las fuentes de energía del dispositivo.

Sin embargo cuando se trabajan con aplicaciones mayores la característica más usada para describir su desempeño es el llamado **Tiempo de Ejecución** del algoritmo.

Uso y Resultados de Interés

Antes de continuar con el ejemplo del clasificador, es necesario indicar cómo trabajar e interpretar las fórmulas descritas en la tabla. Para ejemplificar vamos a usar la primera de ellas, también llamada notación Big-Oh.

Usando la definición

De la tabla vemos que al considerar dos funciones f(n), \, g(n)podemos relacionarlas usando la notación Big-Oh si cumplen con la definición:

\exist k > 0, \, n_0 \forall n > n_0: |f(n)| \leq k \cdot g(n) 

¿pero qué quiere decir toda esta notación? Vamos por partes, se dice que *n* es un parámetro del cual dependen ambas funciones, por ejemplo podría ser el tamaño de una foto en un programa de procesamiento de imágenes, sobre el cual vamos a realizar una comparación.

Tanto la función *f*(*n*), como *g*(*n*) dependen de este parámetro y usualmente indican el **tiempo** asociado a realizar una tarea. Una interpretación usual es que *f*(*n*) es el tiempo que demora un programa en solucionar un problema dada la entrada del tamaño *n*.

Consideremos ahora los otros parámetros: *k* y *n*0. En este caso *k* se usa para indicar un *múltiplo positivo* de la función *g*(*n*) y el parámetro *n*0 indica un punto de partida para la desigualdad de la definición. Notar que la definición solo busca que existan al menos un valor para cada uno de estas constantes.

Finalmente la definición: Dada estas funciones a comparar, a partir del punto inicial *n*0 decimos que:

f(n) = O(g(n)), \forall n \geq n_0

si, y solo sí la magnitud de *f*(*n*) está acotada superiormente por un múltiplo positivo de la otra función (*g*(*n*)) a partir de dicho punto de partida. Es decir, la notación Big-Oh nos ofrece una cota superior para la función *f*(*n*) conforme nosotros cambiamos el tamaño del problema *n*.

* Llamamos **notación asintótica** a la familia de relaciones conocidas como notación de **Bachmann-Landau**, la cual es usada para describir el tiempo de ejecución de los algoritmos.

|  |  |
| --- | --- |
| relación | definición |
| f(n) \underset{n \rightarrow \infty}{=} O(g(n)) | \exist k > 0, \, n_0 \forall n > n_0: |f(n)| \leq k \cdot g(n) |
| f(n) \underset{n \rightarrow \infty}{=} \Omega (g(n)) | \exist k > 0, \, n_0 \forall n > n_0: |f(n)| \geq k \cdot g(n) |
| f(n) \underset{n \rightarrow \infty}{=} \Theta (g(n)) | \exist k_1 > 0, \, k_2 >0, \, n_0 \forall n > n_0: k_1 \cdot g(n) \leq |f(n)| \leq k_2 \cdot g(n) |
| f(n) \underset{n \rightarrow \infty}{=} o(g(n)) | \forall \epsilon > 0 \, \exist n_0 \forall n > n_0 : |f(n)| \leq \epsilon \cdot g(n) |
| f(n) \underset{n \rightarrow \infty}{=} \omega (g(n)) | \forall \epsilon > 0 \, \exist n_0 \forall n > n_0 : |f(n)| \geq \epsilon \cdot g(n |
| f(n) \underset{n \rightarrow \infty}{\thicksim} g(n) | \lim_{n \rightarrow \infty} f(n)/g(n) = 1 |

**Caso de ejemplo:**

Después de hacer varias mediciones, los ayudantes de fundamentos calcularon que el tiempo en que se demoran en revisar las tareas es igual a la progresión aritmética del número de tareas que reciben, por ejemplo si reciben 3 tareas, se demorarán aproximadamente

1 + 2 + 3 = 6[*minutos*]

en terminar.

Generalizando, sea *t*(*n*) el tiempo en minutos en el cual los ayudantes revisan las tareas, luego:

t(n) = \sum_{1 \leq i \leq n} i = \frac{n(n+1)}{2} = 0.5 n^2 + 0.5 n

Pero notamos que ambos sumandos son menores a *n*2, es decir, podemos acotar la función de la siguiente manera:

t(n) = 0.5 n^2 + 0.5 n \leq n^2 + n^2 = 2n^2

Notar que la desigualdad se cumple en particular a partir de 1, luego:

t(n) \leq 2 n^2 \quad \forall n \geq 1

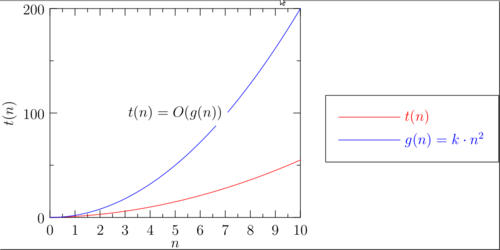
donde

*k* = 2 > 0

*n*0 = 1 > 0

Con estos valores hemos cumplido la definición de la cota superior, por lo que es correcto decir que:

t(n) \underset{n \rightarrow \infty}{=} O(n^2)

[](http://wiki.inf.utfsm.cl/index.php?title=Archivo:BigO1.png)

Para casos de análisis en general, a la constante *k* que multiplica a la función acotante se le llama constante anónima y refleja las propiedades particulares de la máquina donde estamos ejecutando el algoritmo, así como detalles de implementación de los lenguajes sobre los cuales diseñamos la solución. Usualmente calcular con precisión dichas constantes es un trabajo tan difícil que se prefiere trabajar con el orden de la cota (en este ejemplo el orden sería *n*2).

Big-Oh, esquema de demostraciones

Cuando deseamos comprobar relaciones Big-Oh debemos seguir el esquema siguiente que suele ser común para la mayoría de los casos:

**Cómo probar relaciones Big-Oh**:

Lo primero es establecer un *testigo* o valor inicial

a partir del cual la relación es válida, es decir debemos establecer nuestro *n*0. Luego debemos establecer nuestra constante anónima, la cual nos va a indicar que estamos usando un múltiplo de la función que va por arriba. **importante** es recordar que el testigo *n*0 es un natural no cero y la constante es un real positivo.

Lo siguiente es, mediante manipulaciones matemáticas,

que para el problema de tamaño *n* > *n*0 la definición es verdadera, es decir se cumple que c \cdot g(n)es una cota superior de la función *f*(*n*)

**Ejemplo** Considere que luego de una medición se determinó que la máquina ~~para poner ceros~~ correctora de tío Pablo se demora un tiempo igual a (*n* + 5)2, donde *n* es el número de pruebas a corregir. Podemos decir que este tiempo, que vamos a denotar por *T*(*n*) es **cuadrático** porque podemos elegir:

un testigo *n*0 = 1.

una constante anónima *c* = 36.

tal que:


\begin{align}
  T(n) &   =  (n + 5)^2 = n^2 + 10n + 25 \\
       & \leq n^2 + 10n^2 + 25n^2 \\
       & \leq 36n^2
\end{align}


**Cuidado con las demostraciones Big-Oh**

La definición de las notaciones es engañosa ya que requiere elegir el testigo y el múltiplo junto con probar la relación de orden. Es por esto que debemos cuidar nuestra argumentación. Por ejemplo, es común ver argumentaciones en las cuales el testigo es elegido y luego se indica tomar la constante anónima *c* = *n*. Este tipo de argumentos es errado ya que la notación nos exige tomar una constante, pero acá necesitamos conocer el valor de *n*, sin embargo puede variar!.

**BIBLIOGRAFIA**

<http://wiki.inf.utfsm.cl/index.php?title=Notaci%C3%B3n_Asint%C3%B3tica>