Evaluación de Algoritmos

Año 2025

Introducción

Antes de empezar con este tema debemos responder *¿qué es un algoritmo?* y para ello consideremos las definiciones que nos dan los diccionarios:

Real Academia Española *Algoritmo:* (Quizá del lat. tardío **algobarismus*, y este abrev. del ár. clás. *ḥisābu lgūbār* 'cálculo mediante cifras arábigas'.[1]

1. Conjunto ordenado y finito de operaciones que permite hallar la solución a un problema.
2. Método y notación en las distintas formas del cálculo.

Enciclopedia Británica *Algoritmo:* Es un procedimiento sistemático que produce la respuesta a una pregunta o la solución a un problema en un número finito de pasos. El nombre proviene de la traducción al Latín, *Algoritmi de numero Indorum*, del tratado de aritmética "al-Khowârizmî acerca del arte hindú del cálculo" del matemático musulmán del siglo IX al-Khowârizmî (sobrenombre del célebre matemático árabe Mohámed ben Musa) [2].

Ahora podemos ver cómo definen qué es un *algoritmo* algunos de los libros que forman parte de la bibliografía de la materia [3]:

Brassard y Bratley: Es un conjunto de reglas para efectuar algún cálculo, ya sea a mano o, en una máquina.

Cormen, Leiserson y Rivest: Es un procedimiento computacional bien definido que toma algún valor, o conjunto de valores, como entrada y produce algún valor, o conjunto de valores, como salida.

Aho, Hopcroft y Ullman: Es una secuencia finita de instrucciones, cada una de las cuales tiene un significado preciso y puede ejecutarse con una cantidad finita de esfuerzo en un tiempo finito.

Sedgewick: Describe un método para resolver un problema, adecuado a ser implementado en una computadora.

Erickson: Un algoritmo es una secuencia explícita, precisa, no ambigua y mecánicamente ejecutable de instrucciones elementales. [4]

A la luz de las definiciones vistas queda claro que un algoritmo es una secuencia de reglas o instrucciones finita y que cada una de ellas debe ser no ambigua. Además, la ejecución de un algoritmo no debe involucrar decisiones subjetivas ni basarse en el uso de la intuición o la creatividad. Por lo tanto, una

[1] `http://dle.rae.es/?id=1nmLTsh`

[2] `https://www.britannica.com/topic/algorithm`

[3] Un completo análisis del origen de la palabra *algoritmo* aparece en la Sección 1.1 *Algorithms* del Capítulo 1 del libro *The Art of Computer Programming: Fundamental Algorithms (Vol. I, 3ª edición)*, de Donald Knuth, Addison Wesley Longman, 1997.

[4] `http://jeffe.cs.illinois.edu/teaching/algorithms/all-algorithms.pdf`

receta de cocina se puede considerar un algoritmo sólo si ésta describe con precisión los pasos, dando cantidades *exactas* de sus ingredientes (por ejemplo, donde diga "una pizca de ..." debería expresar algo como "0.5 gr. de ...") y *detalladas* instrucciones de los tiempos de cocción (donde aparezca, por ejemplo, "hasta que esté a punto" debería decir algo como "20 minutos de cocción").

Entonces, un algoritmo es una herramienta para resolver un problema computacional bien definido, donde la especificación del problema da la relación deseada entre la entrada y la salida. Por lo tanto, el algoritmo describe un procedimiento computacional para lograr esa relación entre entrada y salida. Un programa es la especificación de un algoritmo en un lenguaje de programación para que pueda ser ejecutado en una máquina.

Motivación

En las materias anteriores se preocupaban por resolver un problema dado, sin preocuparse si el algoritmo que resolvía el problema era o no bueno, o eficiente, y si era el mejor algoritmo.

Hemos visto hasta ahora sólo algunos algoritmos que resuelven la pertenencia de un elemento a un conjunto que puede almacenarse de distintas maneras.

Por ejemplo, para un problema tan sencillo de aritmética elemental como el multiplicar dos números de d dígitos existen varios algoritmos disponibles, tales como la forma en la que nos enseñaron en la escuela, el algoritmo de multiplicación "a la rusa", usando el método hindú, o cualquier otro algoritmo que descomponga los números en partes más pequeñas y construya la solución basada en la solución de las partes (es decir, diseñado con una técnica de dividir para conquistar), entre otros. Entonces, ¿cuál elegiríamos?

Estudiando las propiedades formales de los algoritmos podríamos tomar una decisión inteligente acerca del algoritmo a utilizar en cualquier situación dada. Además, no es práctico correr un algoritmo para ver si es demasiado lento o no. Aunque fuera suficientemente rápido para los datos de entrada considerados, aún no sabríamos si será suficientemente rápido para otros datos de entrada.

Entonces, ¿podemos comparar dos programas corriéndolos? La respuesta es no, porque sus tiempos dependerán del programador, lenguaje, compilador, sistema operativo y máquina. Entonces, en lugar de pensar sobre un programa particular deberíamos considerar la idea detrás del programa, o sea el algoritmo (la estrategia del programa independiente del lenguaje y la máquina).

Por lo tanto, necesitamos estudiar en esta materia cómo evaluar algoritmos, para aprender a decidir si un algoritmo es eficiente o no, y además cuándo un algoritmo es mejor que otro.

Pero, ¿por qué analizamos un algoritmo? Puede ser por el uso que tendrá el algoritmo, por la importancia del mismo con relación a otros algoritmos desde los puntos de vista teórico y práctico, y por la dificultad en el análisis y precisión de la respuesta requerida; pero la razón más directa que justifica analizar un algoritmo es descubrir sus características con el fin de evaluar su adecuación para varias aplicaciones o para compararlo con otros algoritmos para la misma aplicación. Además, el análisis de algoritmos nos brinda conocimientos útiles para el diseño de algoritmos eficientes, ya que como en cualquier disciplina el diseño exitoso está fuertemente influenciado por la capacidad de análisis.

En general pensamos en un algoritmo que resuelve un problema $\mathcal{P}$ dado. $\mathcal{P}$ puede ser cualquier problema, por ejemplo: *multiplicar matrices de determinada dimensión*, *buscar un nº telefónico*, etc. Para poder conocer el problema es necesario conocer el universo de los datos a los cuales se aplicará el

algoritmo; o sea, aquellos elementos que intervienen en el problema. En el caso de la multiplicación de matrices de números reales positivos de dimensión 2×2, el universo de datos será: $\mathbb{R}^{+^{(2\times 2)}}$.

A dicho universo de datos lo denotamos con $\mathcal{D}$ y es el conjunto de todos los objetos válidos que se pueden proveer como entrada al algoritmo que resuelve el problema $\mathcal{P}$. El universo de datos puede ser infinito, pero siempre numerable (en general este conjunto será finito pero muy numeroso).

Necesitamos una manera de "predecir" cuán bueno es un algoritmo dado y para ello necesitamos una "vara" para medir el desempeño de un algoritmo para un dato de entrada. Además debemos tener en cuenta que una "vara' ' conveniente debería:

- medir aquello que nos preocupa medir;
- ser cuantitativa; para que podamos comparar sus valores;
- ser fácil de calcular y
- ser una buena herramienta para predecir.

Supongamos tener dos algoritmos A_1 y A_2 que resuelven al problema $\mathcal{P}$. ¿Con cuál algoritmo me quedo? Para poder decidir debería poder medir el "costo" de ejecutar un algoritmo dado con un determinado dato del universo. Pero, para poder obtener dicho "costo" debo analizar qué cantidad de recursos de cierto tipo (aquellos en los que estamos interesados) insumió el algoritmo con ese dato. Entonces aparece una función a la que llamamos *función de costo*.

Función de costo

Entonces, dado un problema $\mathcal{P}$ habrá un conjunto o familia de algoritmos posibles para resolver a $\mathcal{P}$, el cual denotaremos con $\mathcal{A}$.

La función de costo dirá para un algoritmo $A \in \mathcal{A}$ y un dato $d \in \mathcal{D}$ qué costo tuvo ese algoritmo (cantidad de recursos de cierto tipo que insumió [5]). Formalmente definimos a la función de costo c de la siguiente manera:

$$c : \mathcal{A} \times \mathcal{D} \longrightarrow {\mathbb{R}_0}^+$$

Entonces, en general tenemos un problema $\mathcal{P}$ particular, conocemos el universo de datos $\mathcal{D}$, y además tenemos una determinada función de costo c. Si deseamos comparar algoritmos, tendremos al menos dos algoritmos A_1 y A_2 que resuelven a $\mathcal{P}$. Por lo tanto, resumiendo, podremos tener la siguiente información:

	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	...
A_1	$c(A_1, d_1)$	$c(A_1, d_2)$	$c(A_1, d_3)$	...
A_2	$c(A_2, d_1)$	$c(A_2, d_2)$	$c(A_2, d_3)$	...

Matemáticamente, podemos extender el dominio de c para aplicarla ahora al algoritmo y al universo de datos. Obtenemos así, en lugar de un número real positivo ($c \in {\mathbb{R}_0}^+$), un vector de tamaño $\mid \mathcal{D} \mid$ de valores $c_i \in {\mathbb{R}_0}^+$. Verdaderamente debería ser un conjunto de valores pero, por simplicidad y usando la enumerabilidad de $\mathcal{D}$, lo veremos como un vector.

[5] Por ser una "cantidad de recursos insumidos" no es posible que dicho valor sea negativo, y por lo tanto lo más general es suponer que dicho valor pertenece al conjunto ${\mathbb{R}_0}^+$.

Así, los costos de un algoritmo A, para cada uno de los datos de $\mathcal{D}$, pueden ser vistos como una secuencia de números reales positivos; y ya que el universo $\mathcal{D}$ es enumerable, dicha secuencia puede verse como un vector de $|\mathcal{D}|$ elementos de ${\mathbb{R}_0}^+$. En símbolos, denotaremos con $\overline{c_A}$ al *vector de costos de un algoritmo* A [6], que es un vector de números reales positivos, donde la componente $c_i = c(A, d_i)$ para $i = 1, \ldots, |\mathcal{D}|$.

Cuando por contexto quede claro de qué algoritmo estamos hablando omitiremos la mención del algoritmo y hablaremos simplemente del vector de costos $\bar{c}$, en otro caso lo subindicaremos para que quede explícito el algoritmo al que corresponde.

La función de costo puede ser por ejemplo:

- Tiempo que insumió el algoritmo,
- Espacio ocupado,
- Tiempo de transmisión del resultado,
- Cantidad de operaciones de entrada/salida,
- Esfuerzo de programación,
- Cantidad de operaciones de cierto tipo que realizó,
- etc...

En general es importante mantener el análisis relativamente independiente de las características de su implantación particular [7]. Por lo tanto, es fundamental concentrarse en obtener resultados para características esenciales del algoritmo que se podrán usar para derivar estimaciones precisas del requerimiento verdadero de recursos sobre máquinas reales. En la práctica, alcanzar esta independencia puede ser difícil porque la calidad de la implantación y las propiedades de los compiladores, arquitectura de máquina y otros importantes aspectos del medio ambiente de programación utilizado tienen efectos sustanciales sobre el desempeño del algoritmo.

Si por ejemplo tuviéramos los siguientes costos de aplicar A_1 y A_2:

	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$	$\mathbf{d}_5$	...
A_1	11	9	20	15	5	...
A_2	5	7	14	15	4	...

¿Cuál algoritmo conviene utilizar para resolver el problema?

Si analizamos en detalle, vemos que el algoritmo A_2 en todos los datos evaluados parece costar menos que A_1 (suponiendo que se mantiene que para todos los datos el costo de A_2 es menor que el de A_1) y por lo tanto debe elegirse sin duda a A_2 para resolver a $\mathcal{P}$.

En general si el vector de costos de un algoritmo A_1 es menor que el vector de costos de un algoritmo A_2, esto es cada componente del vector de A_1 es menor o igual que la correspondiente componente de A_2, implica que frente a toda posible entrada de datos A_1 es mejor (o al menos no peor) que A_2.

[6] Un vector de costos es finalmente un elemento de $({\mathbb{R}_0}^+)^{|\mathcal{D}|}$.

[7] Por ejemplo: lenguaje de programación, compilador, velocidad de procesador y tamaño de la memoria de la máquina utilizados.

Pero, qué ocurriría si tengo la siguiente situación:

	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$	$\mathbf{d}_5$	$\mathbf{d}_6$	$\mathbf{d}_7$	...
A_1	11	9	20	15	5	8	6	...
A_2	5	7	14	15	4	16	8	...

Claramente, aquí no es cierto que $\overline{c_{A_1}} \leq \overline{c_{A_2}}$ ni que $\overline{c_{A_2}} \leq \overline{c_{A_1}}$. Entonces, ¿cuál algoritmo debemos elegir?

Podemos observar que en la mayoría de los datos presentados los costos del algoritmo A_2 son menores o iguales que los de A_1. Entonces, ¿seguiría eligiendo a A_2?

¿Qué ocurriría si los datos utilizados para ejecutar el algoritmo fueran d_6 y d_7?

Como no estamos seguros de cuál elegir y no conocemos de antemano cuál será la entrada al algoritmo, podríamos pensar en crear un nuevo algoritmo A_3 que haga lo siguiente:

Entrar el dato d.
Si $c(A_1, d) < c(A_2, d)$ **entonces** *Ejecutar* A_1 *con* d.
sino *Ejecutar* A_2 *con* d.

Este nuevo algoritmo tiene el costo adicional que implica averiguar cuál es el caso. Si este costo adicional es muy bajo el mejor será A_3, en otro caso puede ser que no.

Habíamos dicho que dados dos algoritmos buscábamos poder decidir cuál es mejor, pero seguimos sin saber si es mejor A_1 o A_2. Entonces analicemos en detalle la situación:

Sean $\overline{c_1} = c(A_1, \mathcal{D})$ y $\overline{c_2} = c(A_2, \mathcal{D})$. Podemos asegurar que:

Si $\overline{c_1} \leq \overline{c_2}$ entonces A_1 **es mejor que** A_2.

(esto significa que el vector $\overline{c_1}$ es menor o igual componente a componente que el vector $\overline{c_2}$).

Pero, el problema sigue siendo: ¿cómo decidirse si en algunas componentes $\overline{c_1}$ es menor o igual que $\overline{c_2}$, pero en otras no?

Entonces, surge la necesidad de una función que a partir de la información de un vector (un conjunto grande de números reales) nos permita obtener algo más sencillo de comparar. Esta nueva función, por lo tanto, debería aplicarse a vectores de números reales positivos y obtener un número real positivo que resuma la información del vector. A dicha función la llamaremos *función de evaluación*.

Función de evaluación

La función de evaluación se define formalmente de la siguiente manera:

$$e : (\mathbb{R}_0{}^+)^{|\mathcal{D}|} \longrightarrow \mathbb{R}_0{}^+$$

y ella nos permitirá condensar o resumir en un único valor la información contenida en el vector al que se aplica.

Vamos a requerirle a esta función que cumpla ciertas condiciones para que podamos usarla para comparar algoritmos. Por ejemplo, si ya sabíamos que un algoritmo era mejor que otro (usando sus vectores de costo) el valor que devuelva la función debe mantener esta información.

Propiedades generales de las funciones de evaluación:

Sean $\overline{c}$, $\overline{c_1}$ y $\overline{c_2} \in (\mathbb{R}_0{}^+)^{|\mathcal{D}|}$ y $\alpha \in \mathbb{R}^+$

1) $e(\overline{c}) = 0 \Leftrightarrow \overline{c} = \overline{0}$

2) $\overline{c_1} \leq \overline{c_2} \Rightarrow e(\overline{c_1}) \leq e(\overline{c_2})$

3) $e(\alpha \cdot \overline{c}) = \alpha \cdot e(\overline{c})$ con $\alpha > 0$

4) $e(\overline{c_1} + \overline{c_2}) \leq e(\overline{c_1}) + e(\overline{c_2})$

- La propiedad **1)** pide que e no obtenga un valor distinto de 0 cuando el vector posee todas sus componentes nulas, y a la inversa que sólo devuelva el valor 0 cuando el vector tiene todas sus componentes nulas.
- La propiedad **2)** plantea que si de antemano sabemos que " $\overline{c_1} \leq \overline{c_2}$ ", no puede ser que la función de evaluación no respete este conocimiento previo.
- La propiedad **3)** plantea la independencia de unidades, ya que si expreso en otra medida el resultado entonces la conclusión no debe cambiar.
- La propiedad **4)** es la llamada *propiedad triangular* en Matemática, y en nuestro caso pide que al evaluar la ejecución en secuencia de dos algoritmos no puede obtener algo peor que si evalúa la ejecución de uno y luego la ejecución del otro.

Si hacemos memoria, sobre lo visto en materias anteriores, ya conocíamos funciones que satisfacían esas mismas propiedades y eran aquellas que llamábamos *normas vectoriales* (o *matriciales*). Por lo tanto, esas funciones serán las primeras candidatas a ser usadas como funciones de evaluación.

Las funciones de evaluación más usadas son:

- *Máximo*.
- *Esperanza* (hay probabilidades asociadas a cada dato).
- *Promedio* (todos los datos tienen igual probabilidad).

Recordar que:

$$0 \leq P(d_i) \leq 1, \quad \forall d_i \in \mathcal{D} \quad \text{(siendo } P(d_i) \text{ la probabilidad asociada al dato } d_i)$$

$$\sum_{i=1}^{|\mathcal{D}|} P(d_i) = 1$$

Si los datos son isoprobables $\quad P(d_i) = \dfrac{1}{|\mathcal{D}|} \quad \forall d_i \in \mathcal{D}$

La Esperanza es equivalente al Promedio, cuando los datos son isoprobables.

Ahora la pregunta es: ¿cómo decidimos cuál usar? Para decidirse por una en particular se debe tener en cuenta en qué situación es más conveniente cada una:

Máximo: si se rinden cuentas después de cada ejecución del algoritmo para un caso (por ejemplo: tiempo de respuesta de una consulta en un cajero automático, si pasó el tiempo máximo de respuesta tolerable por un cliente éste se irá disconforme, pero si demoró menos no). Es más sencillo de calcular.

Esperanza: no puedo aplicarlo si no conozco algo sobre las probabilidades de los datos. Se usa si uno rinde cuentas luego de procesar lotes (ej.: actividades no interactivas como liquidación de sueldos).

Promedio: es la Esperanza considerando isoprobabilidad.

Ejemplo: (para entender mejor el porqué pedimos que se cumpla la propiedad 4)

Tenemos dos algoritmos A_1 y A_2, evaluamos por *Máximo* y los costos son los siguientes:

	$\mathbf{d}_1$	$\mathbf{d}_2$	$\mathbf{d}_3$	$\mathbf{d}_4$	$\mathbf{d}_5$	...
A_1	11	9	20	15	5	...
A_2	5	7	14	15	4	...

La propiedad **4)** enunciaba que:

$$\mathit{Máx.}(\overline{c_1} + \overline{c_2}) \leq \mathit{Máx.}(\overline{c_1}) + \mathit{Máx.}(\overline{c_2})$$

Esto es así porque es posible que los máximos de cada algoritmo se compensen. En el ejemplo vemos que:

$$\mathit{Máx.}(\overline{c_1} + \overline{c_2}) = 34$$

mientras que:

$$\mathit{Máx.}(\overline{c_1}) + \mathit{Máx.}(\overline{c_2}) = 35$$

□

Puede ocurrir que un algoritmo A_1 sea mejor que un algoritmo A_2 si evaluamos por *máximo* y usamos como función de costo *cantidad de memoria utilizada*, pero si usamos como función de costo *cantidad de productos realizados* A_2 sea mejor que A_1.

Entonces, dados dos algoritmos, sabiendo qué nos interesa medir para compararlos (o sea, definida cuál será la función de costo utilizada) y qué función de evaluación usaremos, tenemos las herramientas necesarias para poder decidir cuál algoritmo es el mejor. Así, para que sea **completa** y **correcta** la afirmación que "un algoritmo A_1 es mejor que otro algoritmo A_2" debemos agregar:

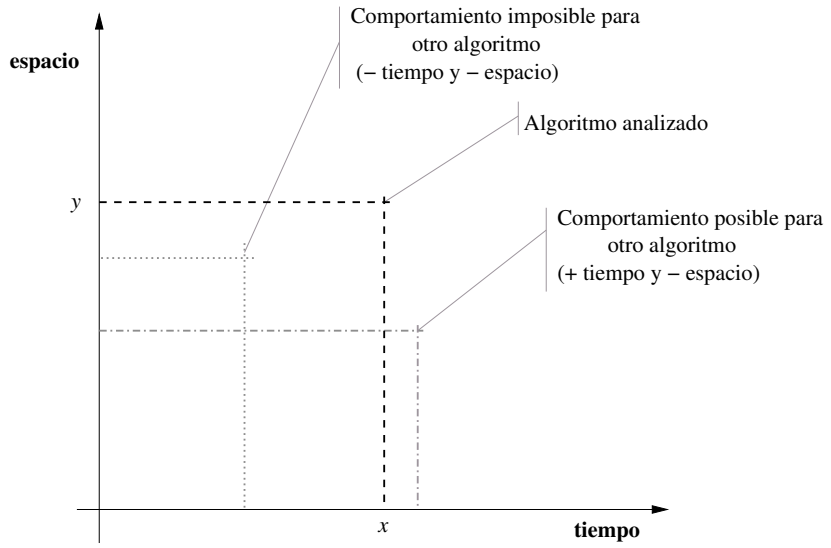
siendo el costo..... y la función de evaluación.....

Ejemplo: siendo el costo *celdas consultadas* y evaluando por *máximo*.

□

En general los algoritmos que ocupan menos memoria auxiliar utilizan más tiempo.

Ejemplo: si graficamos el uso de espacio y tiempo de un algoritmo cualquiera, podríamos tener:



La función de costo puede combinar espacio y tiempo. □

Si conociera realmente el conjunto $\mathcal{A}$ de todos los algoritmos posibles para resolver el problema $\mathcal{P}$, y en ellos pueden evaluarse la función de evaluación e y la función de costo c, entonces podemos definir:

$$\textit{complejidad e-c de } \mathcal{P} = \textit{mín.}_{A \in \mathcal{A}} e(c(A, \mathcal{D}))$$

Si varios algoritmos dan ese mínimo ⇒ se denominan ***optimales***.

Si un único algoritmo realiza ese mínimo ⇒ se denomina ***óptimo***.

El estudio de algoritmos concretos permite obtener cotas superiores y es un análisis más sencillo. Obtener el mínimo es complicado debido a que no conozco todos los algoritmos. Entonces, como en general no se conocen (o no se pretenden conocer) todos los algoritmos para resolver un problema, para encontrar el mínimo se razona en forma abstracta y se analiza lo mínimo que debería hacerse para resolver el problema.

Ejemplo: si el problema $\mathcal{P}$ es encontrar el máximo de 7 elementos, siendo ellos $\{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$, podemos deducir que deben realizarse al menos 6 comparaciones para resolverlo [8].

Un algoritmo sería por ejemplo:

$$\begin{aligned}
&\text{máx} = a_1 \\
&\textbf{for } \; i = 2 \; \textbf{ to } \; 7 \\
&\qquad \textbf{if } \; \text{máx} < a_i \; \textbf{ then } \; \text{máx} = a_i
\end{aligned}$$

Por lo tanto, evaluando por máximo y usando como función de costo la *cantidad de comparaciones*, la complejidad *e-c* de $\mathcal{P}$ es 6, y en este caso podríamos afirmar que este algoritmo es *optimal*. Pero, para afirmar que es el óptimo deberíamos estar seguros que es el único que alcanza este mínimo y en este caso no es así.

□

[8] Esto se puede demostrar formalmente haciendo uso de Teoría de Grafos.

Familia parametrizada de problemas

Para continuar ampliando nuestro análisis podemos observar que muchos problemas pueden parametrizarse de acuerdo al tamaño de los mismos. Por ejemplo, un problema es encontrar el máximo de 7 elementos, otro problema encontrar el máximo de 8 elementos, etc.

Por lo tanto, podemos hablar de una *familia de problemas*, en los cuales aparece como parámetro el "*tamaño*":

$$\mathcal{P} = \{\mathcal{P}_i / i \in \mathbb{N}\}$$

Ahora, no tengo sólo un universo de datos, sino que aparece un universo de datos por cada instancia del problema en el que todos los datos serán de tamaño i, $\mathcal{D}_i$.

A su vez podemos pensar ahora en tener generadores de algoritmos que lean el parámetro del tamaño del problema y en función de su valor devuelvan el algoritmo correspondiente:

$$\mathcal{G} : \mathbb{N} \longrightarrow \mathcal{A}$$

Ejemplo: si queremos encontrar el máximo de n elementos $\{a_1, a_2, a_3, a_4, \ldots, a_n\}$, sólo necesitaríamos cambiar en el ejemplo anterior lo siguiente:

$\text{máx} = a_1$
for $i = 2$ **to** ~~7~~ n
  if $\text{máx} < a_i$ **then** $\text{máx} = a_i$

y para tener el generador en este caso bastaría con agregar como primera línea de código:

read(n)

□

Ahora nos encontramos en la siguiente situación: tenemos una familia parametrizada de problemas y tenemos distintos generadores de algoritmos (dado el valor n, cada uno de ellos genera un algoritmo distinto para resolver $\mathcal{P}_n$), entonces la pregunta es: ¿cuál elegimos como el mejor generador?

En general, si denotamos con A_{ij} al algoritmo generado por el generador $\mathcal{G}_i$ para resolver el problema $\mathcal{P}_j$ (con $n = j$), tendremos la siguiente información:

n	**1**	**2**	**3**	...
	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	...
$\mathcal{G}_1$	$e(c(A_{11}, \mathcal{D}_1))$	$e(c(A_{12}, \mathcal{D}_2))$	$e(c(A_{13}, \mathcal{D}_3))$	...
$\mathcal{G}_2$	$e(c(A_{21}, \mathcal{D}_1))$	$e(c(A_{22}, \mathcal{D}_2))$	$e(c(A_{23}, \mathcal{D}_3))$	...

Volvemos a tener una situación parecida a la ya analizada previamente; pero ahora aparecen secuencias correspondientes a los valores de la función de evaluación para los costos de los algoritmos ya instanciados.

Nuevamente la decisión simple es:

Si en cada posición $\mathcal{G}_1$ ***"es mejor que"*** $\mathcal{G}_2$ entonces elegimos a $\mathcal{G}_1$

en caso contrario:

Si en cada posición $\mathcal{G}_2$ ***"es mejor que"*** $\mathcal{G}_1$ entonces elegimos a $\mathcal{G}_2$.

Otra manera sería intentar nuevamente plantear una función que condense estas secuencias de valores en algo más simple de comparar, pero esto no se hace porque a medida que n crece los valores que aparecen se vuelven muy grandes.

La secuencia de valores para un generador de algoritmos particular $\mathcal{G}$ es *monótona creciente*, porque es imposible que para un tamaño de entrada más grande el costo sea menor que para uno más chico [9].

En este caso necesitamos trabajar de otra manera. Por lo tanto, las comparaciones entre estas secuencias se suelen hacer asintóticamente, es decir comparando sólo para valores de n grandes. Ello significa que si a partir de un determinado valor n_0 de la entrada conviene siempre un generador $\mathcal{G}_1$ en lugar de otro generador $\mathcal{G}_2$, entonces $\mathcal{G}_1$ será el mejor. Entonces, la decisión se basa en que hay una cantidad finita de casos en que puede ganar $\mathcal{G}_2$, pero la cantidad de casos en que gana $\mathcal{G}_1$ es infinita.

Pero, si tenemos un problema $\mathcal{P}_n$ particular a resolver (o sea para un n determinado) dependerá de cuál se comporta mejor para ese n concreto de aplicación cuál elegiremos. Más aún, si conocemos de antemano que en los casos en que tendremos que aplicar el algoritmo el tamaño será menor que n_0, entonces elegimos el que más nos conviene, independientemente de su comportamiento asintótico.

Para poder comparar *asintóticamente* las secuencias mencionadas en general se hace uso de una función matemática denominada *Notación* "O" (*Big Oh- Notation*) [10].

Notación "O"

Como mencionamos ésta es una nueva función y ella se define formalmente de la siguiente manera:

$$\mathcal{F} : \mathbb{N} \longrightarrow \mathbb{R}^+$$

$$O : \mathcal{F} \longrightarrow 2^{\mathcal{F}}$$

$$O(f) = \{g / g \in \mathcal{F} \wedge (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}_0)(\forall n \geq n_0)(g(n) \leq c \cdot f(n))\}$$ [11]

Mediante esta nueva función podemos agrupar funciones en clases que reflejan la velocidad con la que crece la función, y usándola compararemos los generadores de algoritmos.

Aquí, entonces, lo que nos interesará será determinar a qué clase pertenece la función que da la evaluación del vector de costos de un algoritmo para un tamaño de entrada n. O sea, si A_n es el algoritmo generado para el tamaño de entrada n y si llamamos $g(n) = e(c(A_n, \mathcal{D}_n))$; queremos encontrar una función f tal que $g \in O(f)$ [12] (esto se lee "g pertenece al orden de f"). De todas las posibles f's que pudiéramos obtener nos quedaremos con la más simple y además con aquella que crezca más lentamente.

[9] Si fuera así se agrandaría artificialmente el tamaño del problema.

[10] En castellano también se suele nombrar "Notación O grande", para diferenciarla de otras.

[11] Recordar que para definir por comprensión un conjunto hay que dar el género próximo y la diferencia específica.

[12] En la bibliografía también aparece como g es $O(f)$ (se lee: "g es del orden de f"), o que $g = O(f)$ (se lee: "g es un miembro de $O(f)$").

Ejemplo: si $g(n) = 15n + 1$, $f(n)$ podría ser $15n+1$ o n o n^2 o $\frac{3}{8}n$ o $n\sqrt{n}$ o $n^5 + n^2 + 9$, etc.

Pero, en este caso la más simple y de velocidad de crecimiento más lenta de todas es: $f(n) = n$. Entonces, diremos que $g \in O(n)$.

□

Para demostrar que una función pertenece a una determinada clase de funciones, en esta notación, es necesario exhibir al menos un valor para c y un tamaño para n_0 que hagan verdadera la diferencia específica para esa clase o conjunto de funciones. Para demostrar la no pertenencia se debe mostrar que no existen valores para c y n_0 tales que la hagan verdadera.

Ejemplo: si tenemos las funciones n y $2n$, ambas pertenecen a la clase $O(n)$, porque en un caso $c = 1$ y $n_0 = 0$ y en el otro $c = 2$ y $n_0 = 0$.

□

Ejemplo: $n^2 \notin O(n)$ porque no puede existir la constante c, que haga verdadera la desigualdad, debido a que siempre n^2 crece más rápidamente que n.

□

En general si una función pertenece a una determinada clase de funciones pueden existir varios valores para c y n_0; pero, si buscamos un c más chico entonces n_0 será más grande, y a la inversa si buscamos un c más grande entonces n_0 será más chico.

Ejemplo: $n + 1000 \in O(n)$, considerando $c = 1001$ y $n_0 = 1$; o considerando $c = 2$ y $n_0 = 1001$.

□

Otra herramienta útil para analizar la pertenencia de una función g al orden de f ($g \in O(f)$) o viceversa ($f \in O(g)$) es la siguiente:

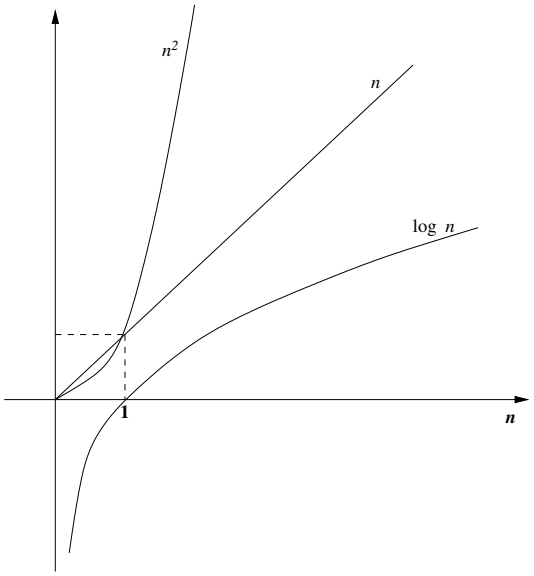
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \begin{cases} \infty & \text{si } g \in O(f) \\ 0 & \text{si } f \in O(g) \\ \text{cte.} & \text{si } O(f) = O(g) \end{cases}$$

Las funciones se pueden ordenar por su velocidad de crecimiento usando su pertenencia a las clases de funciones.

Ejemplo: como $\log n \in O(n)$ y $n \in O(n^2)$ [13], entonces la lista ordenada de estas tres funciones (en orden de velocidad de crecimiento) es: $\log n$, n y n^2.

Gráficamente se ve el porqué; ya que claramente a medida que n crece la función que crece más rápidamente de las tres es n^2 y la que crece más lentamente es $\log n$.

[13] Otra forma de justificar el orden de las funciones es que la primera función debe pertenecer al orden de todas las demás, la segunda al orden de todas las demás menos la primera, y así sucesivamente.



□

Las clases de funciones más comunes son: $O(1)$ que incluye las funciones constantes (que no dependen de n), $O\left(\sqrt{n}\right)$, $O(\log n)$ (logarítmico), $O(n)$ (lineal), $O(n \log n)$, $O\left(n^2\right)$, $O\left(n^3\right)$.

Otras notaciones asintóticas

Cuando se dice que $g \in O(f)$, se sabe que f es una cota superior para la velocidad de crecimiento de g. Para especificar una cota inferior para la velocidad de crecimiento de una función su usa la notación "*Omega*" [14](*Big-Omega Notation*) y se simboliza con Ω.

Formalmente se define como:

$$\Omega(f) = \{g/g \in \mathcal{F} \wedge (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}_0)(\forall n \geq n_0)(g(n) \geq c \cdot f(n))\}$$

Es decir, que en ella se establece que $g \in \Omega(f)$ si existen los valores para las constantes c y n_0 tal que, para todo valor $n \geq n_0$, $g(n) \geq c \cdot f(n)$ (o sea, que para infinitos casos se cumple que $g(n) \geq c \cdot f(n)$).

Razonamientos similares, a los realizados para la notación O, se pueden realizar para la notación Ω.

Ejemplo: la función $g(n) = 25n^3 + 3n^2 + n \in \Omega(n^3)$, ya que para $c = 1$ y $n_0 = 0$, se cumple que para todo $n \geq n_0$ es $g(n) \geq c \cdot f(n)$.

□

Para caracterizar completamente la velocidad de crecimiento de una función se usa la notación "*Thita*" (*Big-Theta Notation*) y se simboliza con la letra griega mayúscula Θ.

Formalmente se define como:

$$\Theta(f) = \{g/g \in \mathcal{F} \wedge (\exists c_1, c_2 \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}_0)(\forall n \geq n_0)(c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n))\}$$

Mediante esta notación se caracteriza la cota inferior y la superior de la velocidad de crecimiento de una función, o sea que g está comprendida entre $c_1 \cdot f$ y $c_2 \cdot f$ para infinitos valores de n.

[14]Es realmente la letra griega Omega mayúscula (Ω).

Por lo tanto, relacionando todas estas notaciones, se tiene el siguiente teorema.

Teorema: Para cualquier par de funciones f y $g \in \mathcal{F}$

$$g \in \Theta(f) \Leftrightarrow g \in O(f) \ \wedge \ g \in \Omega(f).$$

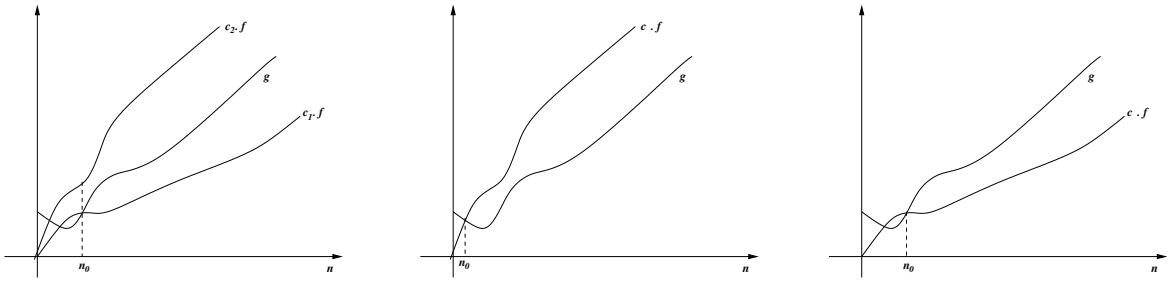
■

Para visualizar más claramente qué significan estas notaciones, analicemos los siguientes ejemplos gráficos de las notaciones Θ, O y Ω. En cada una de las gráficas se muestra el mínimo valor posible de n_0; pero, cualquier valor mayor también sería adecuado.

(a) La notación Θ limita a una función dentro de factores constantes. Decimos que $g \in \Theta(f)$ si existen constantes positivas c_1, c_2 y n_0, tales que a la derecha de n_0, el valor de g siempre cae entre $c_1 \cdot f$ y $c_2 \cdot f$ inclusive.

(b) La notación O da un límite superior para una función dentro de un factor constante. Escribimos $g \in O(f)$ si existen constantes positivas c y n_0, tales que a la derecha de n_0, el valor de g siempre está en o bajo $c \cdot f$.

(c) La notación Ω da un límite inferior para una función dentro de un factor constante. Escribimos $g \in \Omega(f)$ si existen constantes positivas c y n_0, tales que a la derecha de n_0, el valor de g siempre está en o sobre $c \cdot f$.



$g \in \Theta(f)$ **(a)** $g \in O(f)$ **(b)** $g \in \Omega(f)$ **(c)**

En el siguiente ejemplo veremos cómo demostrar la pertenencia de una función a una clase de funciones en notación Θ, porque con este ejemplo se verá también cómo razonar las otras dos notaciones.

Ejemplo: Sea $g(n) = \frac{1}{4}n^2 - 3n$. Demostraremos que $g \in \Theta(n^2)$, para ello debemos encontrar las constantes positivas c_1, c_2 y n_0 tales que:

$$c_1 n^2 \leq \frac{1}{4}n^2 - 3n \leq c_2 n^2 \quad \text{para todo } n \geq n_0$$

Primero dividimos todo por n^2, produciendo así:

$$c_1 \leq \frac{1}{4} - \frac{3}{n} \leq c_2$$

La desigualdad de la derecha se puede mantener para cualquier valor de $n \geq 1$, eligiendo $c_2 \geq \frac{1}{4}$. Del mismo modo podemos razonar la desigualdad del lado izquierdo, ya que ella se puede mantener

para cualquier valor de $n \geq 13$ eligiendo a $c_1 \geq \frac{1}{52}$. Así, eligiendo a $c_1 = \frac{1}{52}$ y a $c_2 = \frac{1}{4}$, y siendo $n_0 = 13$, podemos verificar que $\frac{1}{4}n^2 - 3n \in \Theta(n^2)$. Realmente otras elecciones de las constantes son posibles, pero lo importante es mostrar que existen. Vale la pena notar que estas constantes dependen de la función $\frac{1}{4}n^2 - 3n$; otra función distinta que pertenece a $\Theta(n^2)$ podría requerir de otras constantes.

□

Veamos ahora otro ejemplo para demostrar la no pertenencia de una función a una clase de funciones en notación Θ.

Ejemplo: Sea $g(n) = 5n^4$. Demostraremos que $g \notin \Theta(n^3)$. Se puede usar la definición formal para verificarlo. Supongamos, con el fin de llegar a una contradicción, que existen las constantes positivas c_2 y n_0 tales que $5n^4 \leq c_2 n^3$ para todo $n \geq n_0$. Pero entonces $n \leq \frac{c_2}{5}$, lo cual posiblemente no pueda mantenerse para n grandes dado que c_2 es una constante!!

□

Intuitivamente, los términos de menor orden de una función positiva asintóticamente se pueden ignorar para determinar los límites o cotas inferiores y superiores (debido a que para valores de n grandes ellos son insignificantes). Una fracción pequeña de los términos de mayor orden basta para dominar a los términos de orden menor. Así, darle a c_1 un valor que sea levemente menor que el coeficiente del término de mayor orden y darle a c_2 un valor que sea levemente mayor permite que se satisfagan las desigualdades en la definición de la notación Θ (lo mismo ocurre en las notaciones O y Ω). Por lo tanto, el coeficiente del término de mayor orden puede ser ignorado, dado que sólo cambia los valores de c_1 y c_2 por un factor constante igual al coeficiente.

Para entender mejor lo expuesto anteriormente, veamos el siguiente ejemplo:

Ejemplo: Sea g cualquier función cuadrática de la forma $g(n) = an^2 + bn + c$, donde a, b y c son constantes y $a > 0$. Despreciando los términos de menor orden e ignorando la constante a llegamos a que $g \in \Theta(n^2)$. Formalmente, para mostrar que esto es cierto, tomamos las constantes $c_1 = \frac{a}{4}$ y $c_2 = \frac{7a}{4}$, y $n_0 = 2 \cdot \text{máx}(\frac{|b|}{a}, \sqrt{\frac{|c|}{a}})$. Queda como ejercicio verificar que:

$$c_1 n^2 \leq an^2 + bn + c \leq c_2 n^2 \quad \text{para todo } n \geq n_0.$$

□

En general se puede demostrar que para cualquier polinomio $p(n) = \sum_{i=0}^{k} a_i \cdot n^i$, donde las a_i son constantes y $a_k > 0$, se cumple que $p(n) \in \Theta(n^k)$.

Comparación de funciones

Muchas de las propiedades relacionales que se cumplen en $\mathbb{R}$ se aplican también a las comparaciones asintóticas. Mostraremos aquí algunas de ellas y asumiremos que f, g y h son funciones que pertenecen a $\mathcal{F}$.

Transitividad:

$f \in O(g)$ y $g \in O(h)$ implica que $f \in O(h)$,
$f \in \Omega(g)$ y $g \in \Omega(h)$ implica que $f \in \Omega(h)$,
$f \in \Theta(g)$ y $g \in \Theta(h)$ implica que $f \in \Theta(h)$.

Reflexividad:

$f \in O(f)$,
$f \in \Omega(f)$,
$f \in \Theta(f)$.

Simetría:

$f \in \Theta(g)$ si y sólo si $g \in \Theta(f)$.

Simetría Transpuesta:

$f \in O(g)$ si y sólo si $g \in \Omega(f)$,

o lo que es lo mismo:

$f \in \Omega(g)$ si y sólo si $g \in O(f)$, (por ser una doble implicación)

A pesar de que muchas de las propiedades de los números reales se mantienen al comparar asintóticamente funciones, no es cierto que dado cualquier par de funciones se cumpla que ellas son comparables. Es decir, para dos funciones cualesquiera f y g puede ser que $f \notin O(g)$ y $f \notin \Omega(g)$.

Existen además otras notaciones (o y ω) que en esta materia no analizaremos, dado que no son de interés en el contexto del primer curso en el que se hace análisis de algoritmos.

Notas:

Donald Knuth señala que el origen de la notación O es un texto sobre Teoría de Números de *P. Bachmann* en 1892. Las notaciones Θ y Ω fueron ideadas por Donald Knuth para corregir la popular, pero técnicamente descuidada, práctica de usar la notación O para ambos límites superior e inferior.

No todos los autores definen las notaciones asintóticas de la misma manera, aunque varias de ellas concuerdan en la mayoría de las situaciones comunes.

Reconocimientos

El presente apunte se realizó tomando como base notas de las clases del **Ing. Hugo Ryckeboer** en la Universidad Nacional de San Luis y completando con material obtenido desde la bibliografía de la materia.

Apéndice

Mostraremos acá algunos resultados elementales acerca de límites. En nuestra presentación omitiremos las demostraciones, puesto que las mismas se deben haber analizado en asignaturas de matemáticas correlativas con la nuestra.

Reglas para el cálculo de límites

La mayoría de las funciones se obtienen como suma, producto o composición de funciones más simples, por este motivo se enuncian teoremas que nos permiten calcular límites usando reglas de cálculo.

1. Si f es la función constante $f(x) = c$ para todo x:

$$\lim_{x \to a} f(x) = \lim_{x \to a} c = c$$

◇

Ejemplo: $\lim_{x \to 3} 8 = 8$

2. Si $f(x) = x$ para todo x:

$$\lim_{x \to a} f(x) = \lim_{x \to a} x = a$$

◇

Ejemplo: $\lim_{x \to \sqrt{3}} x = \sqrt{3}$

3. Si el $\lim_{x \to a} f(x) = L$ y $\lim_{x \to a} g(x) = M$, entonces:

   a) $\lim_{x \to a} [f(x) \pm g(x)] = L \pm M$

   b) $\lim_{x \to a} [f(x) \cdot g(x)] = L \cdot M$

   c) $\lim_{x \to a} \frac{f(x)}{g(x)} = \frac{L}{M}$, si $M \neq 0$

   d) $\lim_{x \to a} c \cdot f(x) = c \cdot L$

◇

Ejemplo:

(i) $\lim_{x \to \sqrt{3}} 5 + x = 5 + \sqrt{3}$

(ii) $\lim_{x \to \sqrt{3}} x^2 = \sqrt{3} \cdot \sqrt{3} = 3$

(iii) $\lim_{x \to 3} \frac{5x^2 - 2x + 1}{4x^3 - 7} = \frac{5(3)^2 - 2(3) + 1}{4(3)^3 - 7} = \frac{40}{101}$

4. Un teorema muy importante es el siguiente:

   Teorema 1 *Si f y g son funciones tales que $\lim_{x\to a} g(x) = b$ y f es continua en b, entonces:*

   $$\lim_{x\to a} f(g(x)) = f(b) = f(\lim_{x\to a} g(x))$$

   ◇

   Ejemplo:

   (i) $\lim_{h\to 0} \sqrt{h^2 + 3h - 8} = -2$

   (ii) $\lim_{t\to 1} e^{-16(4+t)} = e^{-80}$

Infinitos

Sea $f(n)$ cualquier función de n. Diremos que $f(n)$ tiende a un límite v cuando n tiende a infinito si $f(n)$ es casi igual a v cuando n es grande. La siguiente definición formal hace más precisa esta noción.

Definición 1 *Se dice que la función $f(n)$ tiende al límite v cuando n tiende a infinito si para todo número real positivo δ, independientemente de lo pequeño que sea, $f(n)$ dista de v en una cantidad menor que δ para todos los valores de n suficientemente grandes.* ◇

Dicho de otra manera, independientemente de lo pequeño que sea δ, podemos hallar un valor $n_0(\delta)$ correspondiente a δ tal que $f(n)$ dista de v en menos que δ para todos los valores de n mayores o iguales que $n_0(\delta)$. Si $f(n)$ tiende al límite v cuando n tiende a infinito, escribimos:

$$\lim_{n\to\infty} f(n) = v$$

Existen numerosas funciones, desde luego, que no tienden a un límite cuando n tiende a infinito. La función n^2, por ejemplo, puede hacerse tan grande como sea preciso seleccionando sólo un valor de n suficientemente grande. Se dice que estas funciones tienden a infinito cuando n tiende a infinito. La definición formal es como sigue.

Definición 2 *Se dice que la función $f(n)$ tiende a $+\infty$ si para todo número Δ, independientemente de lo grande que sea, $f(n)$ es mayor que Δ para todos los valores de n suficientemente grandes.* ◇

Nuevamente esto significa que podemos hallar un umbral $n_0(\Delta)$ correspondiente a Δ, tal que $f(n)$ es mayor que Δ para todos los valores de n mayores o iguales que $n_0(\Delta)$[15]. Escribiremos:

$$\lim_{n\to\infty} f(n) = +\infty$$

[15]Hay una definición parecida para fórmulas tales como $-n^2$, que pueden tomar valores negativos cada vez más grandes a medida que n tiende a infinito. Se dice que esas funciones tienden a $-\infty$. Dado que necesitamos trabajar en $\mathbb{R}_0{}^+$, no analizaremos esto en más detalle.

Cuando una función $f(n)$ no tiende a un límite, ni tampoco a $+\infty$ o $-\infty$, diremos que $f(n)$ *oscila* cuando n tiende a infinito. Si es posible encontrar una constante positiva k tal que $-k < f(n) < k$ para todos los valores de n, diremos que $f(n)$ oscila de *forma finita*; en caso contrario, $f(n)$ oscila de *forma infinita*. Por ejemplo, la función $(-1)^n$ oscila de forma finita; la función $(-1)^n n$ oscila de forma infinita.

Las siguientes proposiciones enuncian algunas de las propiedades generales de los límites.

Proposición 1 *Si dos funciones $f(n)$ y $g(n)$ tienden respectivamente a los límites v y w cuando n tiende a infinito, entonces la función $f(n) + g(n)$ tiende al límite $v + w$.* ◇

Proposición 2 *Si dos funciones $f(n)$ y $g(n)$ tienden respectivamente a los límites v y w cuando n tiende a infinito, entonces la función $f(n) \cdot g(n)$ tiende al límite $v \cdot w$.* ◇

Ambas proposiciones se pueden extender a la suma o al producto de cualquier número finito de funciones de n. Un caso particular importante de la Proposición 2 es aquél en el cual $g(n)$ es constante. La proposición afirma, entonces, que si el límite de $f(n)$ es v, entonces el límite de $c \cdot f(n)$ es $c \cdot v$, en donde c es cualquier constante.[16] Por último, la siguiente proposición afecta la división.

Proposición 3 *Si dos funciones $f(n)$ y $g(n)$ tienden respectivamente a los límites v y w cuando n tiende a infinito, y w no es cero, entonces la función $f(n)/g(n)$ tiende al límite v/w.* ◇

Estas proposiciones, aunque sean sencillas, resultan sorprendentemente potentes. Por ejemplo, supongamos que se desea conocer el comportamiento de la función racional más general de n, a saber:

$$S(n) = \frac{a_0 n^p + a_1 n^{p-1} + \ldots + a_p}{b_0 n^q + b_1 n^{q-1} + \ldots + b_q}$$

en donde ni a_0 ni b_0 son cero. Si escribimos $S(n)$ de la forma:

$$S(n) = n^{p-q} \left\{ \left(a_0 + \frac{a_1}{n} + \ldots + \frac{a_p}{n^p} \right) \Big/ \left(b_0 + \frac{b_1}{n} + \ldots + \frac{b_q}{n^q} \right) \right\}$$

y aplicando las proposiciones anteriores, resulta sencillo ver que la función que va entre llaves tiende al límite $\frac{a_0}{b_0}$ cuando n tiende a infinito. Además, n^{p-q} tiende al límite 0 si $p < q$; $n^{p-q} = 1$ y además n^{p-q} tiende al límite 1 si $p = q$; y n^{p-q} tiende a infinito si $p > q$. Por lo tanto:

$$\lim_{n \to \infty} S(n) = 0 \quad \text{cuando} \quad p < q$$

$$\lim_{n \to \infty} S(n) = \frac{a_0}{b_0} \quad \text{cuando} \quad p = q$$

y $S(n)$ tiende a más o menos infinito cuando $p > q$ dependiendo del signo de $\frac{a_0}{b_0}$.

[16] Es perfectamente posible que $f(n) + g(n)$, o bien $f(n) \cdot g(n)$ tiendan a un límite cuando ni $f(n)$ ni $g(n)$ tengan límite.

Proposición 4 *Si* $\lim_{n\to\infty}(f(n+1)/f(n)) = v$, $-1 < v < 1$, *entonces* $\lim_{n\to\infty} f(n) = 0$. *Si* $f(n)$ *es positivo y* $\lim_{n\to\infty}(f(n+1)/f(n)) = v > 1$, *entonces* $f(n)$ *tiende a infinito.* ◇

Esta proposición se puede utilizar para determinar el comportamiento cuando n tiende a infinito de $f(n) = n^r x^n$, donde r es cualquier entero positivo. Si $x = 0$ entonces $f(n) = 0$ para todos los valores de n. En caso contrario:

$$\frac{f(n+1)}{f(n)} = \left(\frac{n+1}{n}\right)^r x$$

tiende a x cuando n tiende a infinito. (Utilizar las proposiciones anteriores para verificarlo). Por tanto si $-1 < x < 1$ entonces $f(n)$ tiende a 0, y si $x > 1$ entonces $f(n)$ tiende a infinito. Si $x = 1$ entonces $f(n) = n^r$, lo cual tiende claramente a infinito. Por último es fácil ver que si $x \leq -1$ entonces $f(n)$ presenta una oscilación infinita.

La *regla de l'Hôpital* se puede utilizar en algunas ocasiones cuando resulta imposible aplicar la Proposición 3. Una forma sencilla de esta regla es la siguiente:

Proposición 5 *(De l'Hôpital) Supóngase que*

$$\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = 0,$$

o alternativamente que estos dos limites son infinitos. Supóngase además que los dominios de f *y* g *se pueden extender hasta algún intervalo real* $[n_0, +\infty)$ *de tal forma que las nuevas funciones correspondientes* $\hat{f}$ *y* $\hat{g}$ *son diferenciables en este intervalo, y además que* $\hat{g}'(x)$, *la derivada de* $\hat{g}(x)$, *nunca es cero para* $x \in [n_0, +\infty)$, *entonces*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\hat{f}'(x)}{\hat{g}'(x)}$$

◇

Como ejemplo sencillo, supongamos que $f(n) = \log n$ y $g(n) = n^a$, en donde $a > 0$ es una constante arbitraria. Ahora bien, tanto $f(n)$ como $g(n)$ tienden a infinito cuando n tiende a infinito, así que no se puede utilizar la Proposición 3. Sin embargo, si extendemos $f(n)$ a $\hat{f}(x) = \log x$ y $g(n)$ a $\hat{g}(x) = x^a$, entonces la regla de l'Hôpital nos permite concluir que:

$$\lim_{n\to\infty} \log\left(\frac{n}{n^a}\right) = \lim_{x\to\infty} \left(\frac{1/x}{ax^{a-1}}\right) = \lim_{x\to\infty} \left(\frac{1}{ax^a}\right) = 0$$

sea cual fuere el valor positivo de a.

Finalmente, la siguiente proposición es útil en algunas ocasiones aunque es muy sencilla de demostrar.

Proposición 6 *Si las dos funciones* $f(n)$ *y* $g(n)$ *tienden a los límites* v *y* w *respectivamente cuando* n *tiende a infinito, y si* $f(n) \leq g(n)$ *para todo* n *suficientemente grande, entonces* $v \leq w$. ◇