

Programación estructurada

Ejecución de un programa: Correctez

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

1 de octubre de 2025



Gráficas de flujo

- 1 Gráficas de flujo
- 2 Complejidad.
- 3 Bibliografía

Temas

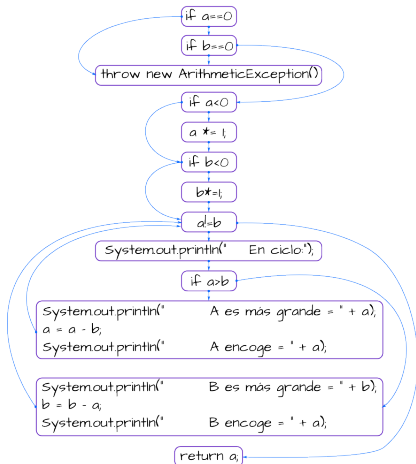
- 1 Gráficas de flujo
 - Gráficas de flujo
 - Pruebas unitarias

Gráficas de flujo simplificadas

- Las gráficas de flujo son estructuras utilizadas por los compiladores para analizar y optimizar código, antes de generar el código objetivo.
- Aquí utilizaremos este concepto para analizar nuestro código de alto nivel.
- En una *gráfica de flujo* se ilustran las posibles rutas que puede seguir un código, dependiendo de sus datos de entrada.

Gráfica de flujo

Gráfica de flujo



```

1 public class Algoritmos {
2     /** Calcula el máximo común divisor.
3      * @param a Entero distinto de cero
4      * @param b Entero distinto de cero
5      * @return mcd(|a|, |b|) */
6     public static int mcd(int a, int b) {
7         if (a == 0 || b == 0) throw new ArithmeticException();
8         if (a < 0) a *= -1; // Lo hacemos positivo
9         if (b < 0) b *= -1;
10
11         // Algoritmo de Euclides versión iterativa.
12         while(a != b) {
13             System.out.println("░░░░░Enciclo:");
14             if (a > b) {
15                 System.out.println("░░░░░░░░░A░es░más░grande░=" + a);
16                 a = a - b;
17                 System.out.println("░░░░░░░░░A░encoge░=" + a);
18             } else {
19                 System.out.println("░░░░░░░░░B░es░más░grande░=" + b);
20                 b = b - a;
21                 System.out.println("░░░░░░░░░B░encoge░=" + a);
22             }
23         }
24         return a;
25     }
26 }

```

Temas

- 1 Gráficas de flujo
 - Gráficas de flujo
 - Pruebas unitarias

Pruebas unitarias

- A partir de la gráfica de flujo es posible **determinar todas las posibles rutas** que podría seguir el código durante su ejecución, dependiendo de los valores actuales.
- Se deben crear pruebas del código que sigan cada una de las rutas y muestren que el código realmente ejecutó lo que se esperaba.
- En estas pruebas también se debe verificar que el código sea **robusto**, es decir, que se defienda bien de datos erróneos.

Ejemplo

```
1 public class Algoritmos {
2
3     // ...
4
5     public static void main(String[] args) {
6         // int r1 = mcd(0, 45); // Provocará una excepción
7         int r2 = mcd(12, -16);
8         System.out.println("El máximo común divisor entre 12 y -16 es " + r2);
9
10        int r3 = mcd(-12, 16);
11        System.out.println("El máximo común divisor entre -12 y 16 es " + r3);
12
13        int iguales = mcd(20, -20);
14        System.out.println("El máximo común divisor entre 20 y -20 es " + iguales);
15
16        int max = mcd(9, 6);
17        System.out.println("El máximo común divisor entre 9 y 6 es " + max);
18
19        int primosRelativos = mcd(9, 8);
20        System.out.println("El máximo común divisor entre 9 y 8 es " + primosRelativos);
21    }
22 }
```


Ejemplo con pruebas refactorizadas

```

1  public class Algoritmos {
2      // ...
3      /* Verifica si mcd(a,b) devuelve r para los enteros a y b. */
4      private static boolean test(int a, int b, int r) {
5          int r2 = mcd(a, b);
6          out.println("El máximo común divisor entre " + a + " y " + b + " es " + r2 + " se esperaba " + r);
7          return r == r2;
8      }
9
10     /* Bateria de pruebas unitarias para mcd(a,b) */
11     public static void main(String[] args) {
12         // int r1 = mcd(0, 45); // Provocará una excepción
13         out.println("Pasó " + test(12, -16, 4));
14         out.println("Pasó " + test(-12, 16, 4));
15
16         out.print("[Iguales]");
17         out.println("Pasó " + test(20, -20, 20));
18
19         out.print("[Max]");
20         out.println("Pasó " + test(9, 6, 3));
21
22         out.print("[Primos relativos]");
23         out.println("Pasó " + test(9, 8, 1));
24     }
25 }

```

Complejidad.

- 1 Gráficas de flujo
- 2 Complejidad.
- 3 Bibliografía

Temas

- 2 Complejidad.
 - Definiciones
 - Notación asintótica

Complejidad

Por *complejidad* nos referimos en forma genérica a las diversas características que impactarán el desempeño de un algoritmo:

- ❶ **Tiempo:** ¿cuánto tarda en ejecutarse un algoritmo?
- ❷ **Espacio:** ¿cuánta memoria utiliza para su ejecución?
- ❸ **Tamaño:** número de instrucciones.
- ❹ **Dificultad:** ¿qué tan complicado es de leer, entender, modificar y extender?

Complejidad algorítmica

Definición (Complejidad algorítmica)

El tiempo de ejecución $T_A(E)$ de un algoritmo A sobre un ejemplar E , de un problema P , es el número de operaciones elementales que requiere A para resolver E .

- Al número de operaciones elementales calculadas se le llama *tiempo de ejecución* o *desempeño computacional*.

Tamaño de un ejemplar

Definición (Tamaño de E)

El tamaño $n = |E|$ de un ejemplar E , es una medida natural del número de elementos que posee E .

Ejemplos:

- El valor del entero que se pasa como parámetro.
- El número de bits requeridos para almacenar los datos.
- El número de caracteres en una cadena.

Dependencia del tamaño

- Los criterios empleados para evaluar la complejidad algorítmica proporcionan medidas **relativas** al tamaño del problema.
- El desempeño computacional de un algoritmo T , es una función que depende del tamaño del ejemplar n .

$$T = T(n)$$

Dependencia de la estructura del problema

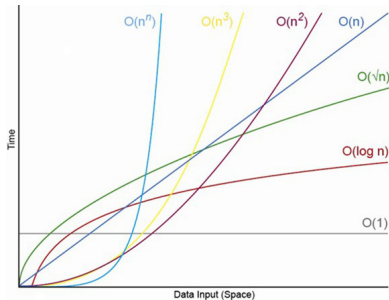
- Para un mismo tamaño de E , es posible tener diferentes desempeños.
- Se distinguen:
 - El mejor caso.
 - El caso promedio.
 - El peor caso.
- Por ejemplo: al calcular el máximo común divisor.
 - Mejor caso: los números son iguales.
 - Caso promedio: los números tienen un divisor común.
 - Peor caso: los números son primos relativos.

Temas

- 2 Complejidad.
 - Definiciones
 - Notación asintótica

Notación asintótica

- Lo que más nos interesa es la **magnitud** del desempeño computacional:
¿Cómo se comporta $T(n)$ cuando n es “grande”?



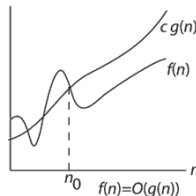
Notación asintótica

Sea $f(n) : f(n) \geq 0, \forall n \geq 0, n \in \mathbb{Z}$.

Definición ($O(n)$)

$f(n)$ es “O grande” de $g(n)$ ($f(n) = O(g(n))$) si

- $\exists n_0 \in \mathbb{Z}$ y una constante $c > 0$ tal que
- $\forall n | n \in \mathbb{Z}, n \geq n_0 \Rightarrow f(n) \leq cg(n)$

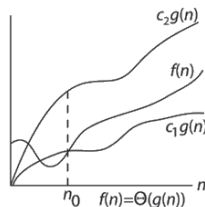


Notación asintótica

Definición ($\Theta(g(n))$)

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2 \text{ constantes con } 0 \leq c_1, 0 \leq c_2, \\ \text{y } n_0 \text{ tal que } , \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

Es decir, $f(n)$ es $O(g(n))$ y $\Omega(g(n))$.



Notación asintótica

Sea $f(n) : f(n) \geq 0, \forall n \geq 0, n \in \mathbb{Z}$.

Definición ($o(n)$)

$f(n)$ es “o pequeña” de $g(n)$ ($f(n) = o(g(n))$) si

- $\forall c > 0, \exists n_0(c) \in \mathbb{Z}$ tal que
- $\forall n | n \in \mathbb{Z}, n \geq n_0 \Rightarrow f(n) \leq cg(n)$

Consecuencias:

- $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

Notación asintótica $o(g(n))$ cont.

Consecuencias (cont.):

- Si $g(n)$ es una **cota justa** de $f(n)$, se cumple $f(n) = O(g(n))$ pero no $f(n) = o(g(n))$.

Ejemplo: Sea $f(n) = 2n^2 + 1$ y $g(n) = n^2$.

- $f(n)$ es $O(n^2)$, sea $c = 3 \Rightarrow$ se calcula n_0 :

$$2n^2 + 1 \leq 3n^2$$

$$1 \leq n^2$$

$$n \geq \sqrt{1} = 1 \Rightarrow n_0 = 1 \quad \square$$

- $f(n)$ no es $o(n^2)$, sea $c = 1 \Rightarrow$

$$2n^2 + 1 \leq n^2$$

$$n^2 + 1 \leq 0$$

$$n^2 \leq -1, \text{ pero } n_0 \in \mathbb{Z}^+! \quad \square$$

Categorías de orden $o(g(n))$

- Es posible clasificar categorías de orden con funciones características:

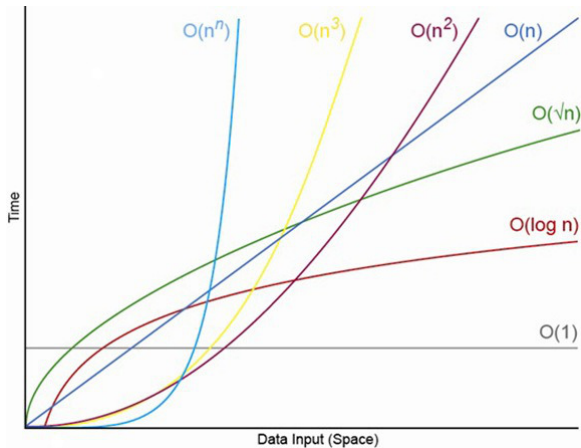
$$\Theta(\log n) \Theta(n) \Theta(n \log n) \Theta(n^2) \underbrace{\Theta(n^j) \Theta(n^k)}_{2 < j < k} \underbrace{\Theta(a^n) \Theta(b^n)}_{0 < a < b} \Theta(n!)$$

$\xrightarrow{\hspace{1.5cm}}$
 cota superior

←
en esta dirección no

- Si una función de complejidad $f(n)$ está en una categoría a la izquierda de la categoría que contiene a $g(n)$ entonces $f(n)$ es $o(g(n))$.

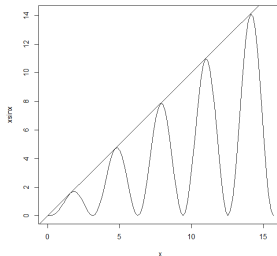
Categorías de orden $o(g(n))$



<http://apelbaum.wordpress.com/2011/05/05/big-o/>

¿Definición alternativa?

- $f(n)$ es $o(g(n))$ si $f(n)$ es $O(g(n))$ pero no $\Omega(g(n))$





$x \sin^2(x)$ es $O(x)$ pero ¿ $x \sin^2(x)$ es $o(x)$?

Haciendo coincidir ambas definiciones: $x \sin^2(x)$ es $o(x^2)$, pero esta no es una cota justa.

Bibliografía

- 1 Gráficas de flujo
- 2 Complejidad.
- 3 Bibliografía**

Bibliografía I

-  Aho, Alfred V. et al. (2007). *Compilers, Principles, Techniques and Tools*. Addison Wesley.
-  Cormen, Thomas H. et al. (2009). *Introduction to Algorithms*. 3rd. The MIT Press.

Licencia

Creative Commons
Atribución-No Comercial-Compartir Igual

