Programación estructurada

Ejecución de un programa: Correctez

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

26 de octubre de 2020

Gráficas de flujo

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

Temas

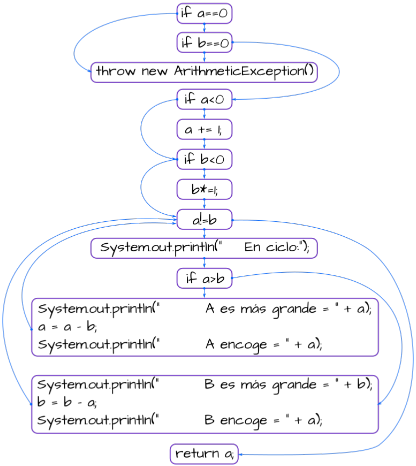
1. Gráficas de flujo
   - Gráficas de flujo
   - Pruebas unitarias

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




```java
/** Calcula el máximo común divisor. */
public static int mcd(int a, int b) {
  if (a == 0 || b == 0) throw new ArithmeticException();
  if (a < 0) a *= -1;   // Lo hacemos positivo
  if (b < 0) b *= -1;

  // Algoritmo de Euclides versión iterativa.
  while(a != b) {
    System.out.println("␣␣␣␣En␣ciclo:");
    if (a > b) {
      System.out.println("␣␣␣␣␣␣␣␣A␣es␣más␣grande␣=␣" + a);
      a = a - b;
      System.out.println("␣␣␣␣␣␣␣␣A␣encoge␣=␣" + a);
    } else {
      System.out.println("␣␣␣␣␣␣␣␣B␣es␣más␣grande␣=␣" + b);
      b = b - a;
      System.out.println("␣␣␣␣␣␣␣␣B␣encoge␣=␣" + a);
    }
  }
  return a;
}
```

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

```java
public static void main(String[] args) {
  // int r1 = mcd(0, 45);    // Provocará una excepción
  int r2 = mcd(12, -16);
  System.out.println("El máximo común divisor entre 12 y -16 es " + r2);

  int r3 = mcd(-12, 16);
  System.out.println("El máximo común divisor entre -12 y 16 es " + r3);

  int iguales = mcd(20, -20);
  System.out.println("El máximo común divisor entre 20 y -20 es " + iguales);

  int max = mcd(9, 6);
  System.out.println("El máximo común divisor entre 9 y 6 es " + max);

  int primosRelativos = mcd(9, 8);
  System.out.println("El máximo común divisor entre 9 y 8 es " +
      ->primosRelativos);
}
```

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:

1. **Tiempo:** ¿cuánto tarda en ejecutarse un algoritmo?
2. **Espacio:** ¿cuánta memoria utiliza para su ejecución?
3. **Tamaño:** número de instrucciones.
4. **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 poseé 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

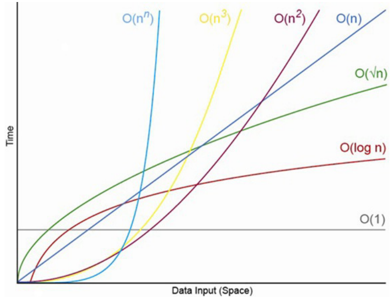
- Para un mismo tamaño de E, es posible tener diferentes desempeños.
- Se distinguen:
  - El mejor caso.
  - El caso promedio.
  - El peor caso.
- Por ejemplo: al calcular el máximo común divisor.
  - Mejor caso: los números son iguales.
  - Caso promedio: los números tienen un divisor común.
  - Peor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



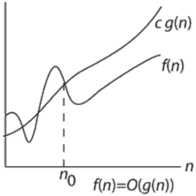
Notación asintótica

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

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
- $\forall n | n \in \mathbb{Z}, n \geqslant n_0 \Rightarrow f(n) \leqslant cg(n)$



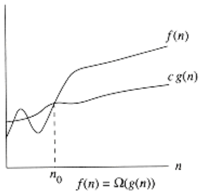
Notación asintótica

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

Definición ($\Omega(g(n))$)

$f(n)$ es "omega" de $g(n)$ ($f(n) = \Omega(g(n))$) si

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

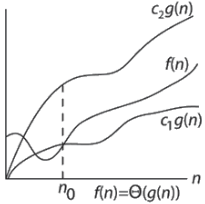


Notación asintótica

Definición ($\Theta(g(n))$)

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

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



Notación asintótica

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

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
- $\forall n | n \in \mathbb{Z}, n \geqslant n_0 \Rightarrow f(n) \leqslant cg(n)$

Consecuencias:

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

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:

$$\begin{aligned} 2n^2 + 1 &\leqslant 3n^2 \\ 1 &\leqslant n^2 \\ n &\geqslant \sqrt{1} = 1 \Rightarrow n_0 = 1 \quad \square \end{aligned}$$

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

$$\begin{aligned} 2n^2 + 1 &\leqslant n^2 \\ n^2 + 1 &\leqslant 0 \\ n^2 &\leqslant -1, \text{ pero } n_0 \in \mathbb{Z}^+! \quad \square \end{aligned}$$

Categorías de orden o(g(n))

- Es posible clasificar categorías de orden con funciones características:

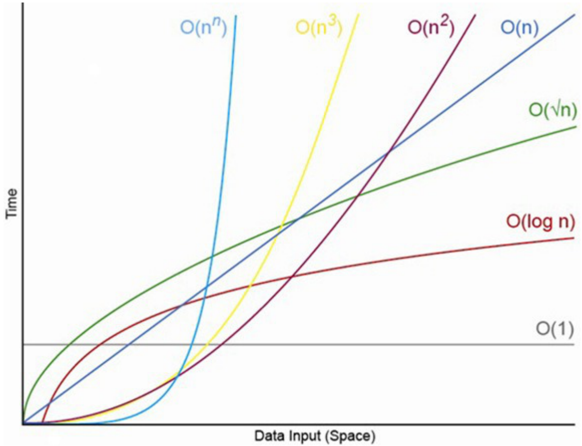
$$\Theta(\log n)\Theta(n)\Theta(n\log n)\Theta(n^2)\underbrace{\Theta(n^j)\Theta(n^k)}_{2<j<k}\underbrace{\Theta(a^n)\Theta(b^n)}_{0<a<b}\Theta(n!)$$

$$\xrightarrow[\text{cota superior}]{}$$

$$\color{red}{\xleftarrow[\text{en esta dirección no}]{}}$$

- 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 $o(g(n))$.

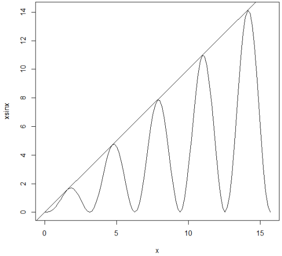
Categorías de orden o(g(n))



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 $o(x)$?

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

Bibliografía

Bibliografía I

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

Licencia