Universidad Andrés Bello®
Conectar • Innovar • Liderar

FUNDAMENTOS DE PROGRAMACIÓN EN JAVA

Algoritmos

Introducción a los algoritmos

A lo largo de la historia, el ser humano ha aprendido a resolver problemas, desde la creación de la rueda hasta la invención de los teléfonos móviles. En la actualidad se siguen resolviendo problemas de forma innata hasta sin darse cuenta; sin embargo, no siempre se utiliza alguna metodología para llevarla a cabo, sino que se realiza basado en la práctica.

Para ejemplificar lo anterior basta con preguntarse, ¿qué hacemos todas las mañanas al despertarnos?. Seguramente la respuesta sería algo como esto:

- Pararse de la cama
- Ir al baño
- Tratar de arreglarse
- Desayunar
- Salir al trabajo/escuela/universidad

Al analizar la acción de “pararse de la cama”, se debe tomar en cuenta ¿cuál sería el planteamiento del problema?. Una respuesta acertada sería: “un día cualquiera por la mañana, una persona debe levantarse de la cama, pero ésta se encuentra tapada por una sábana y dos cobijas”. Con el problema identificado, se pueden identificar los pasos a seguir para lograr levantarse de la cama. Este cuestionamiento se denomina **lógica algorítmica**.

Definición de un algoritmo

Un algoritmo es un conjunto de instrucciones o reglas definidas y no ambiguas, ordenadas y finitas que permite, típicamente, solucionar un problema, realizar un cómputo, procesar datos y llevar a cabo otras tareas o actividades. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución o salida.

Los algoritmos son independientes de los lenguajes de programación. En cada problema el algoritmo puede escribirse y luego ejecutarse en un lenguaje diferente de programación. El algoritmo es la infraestructura de cualquier solución, escrita luego en cualquier lenguaje de programación.

En la vida cotidiana, se emplean algoritmos frecuentemente para resolver problemas. Algunos ejemplos son los manuales de usuario, que muestran algoritmos para usar un aparato, o las instrucciones que recibe un trabajador de su patrón.

En términos de programación, un algoritmo es una secuencia de pasos lógicos que permiten solucionar un problema.

Un algoritmo debe cumplir además con las siguientes características:

- **Finito:** el algoritmo debe acabar tras un número finito de pasos. Es más, es casi fundamental que sea en un número razonable de pasos.
- **Bien definido:** el algoritmo debe definirse de forma precisa para cada paso, es decir, hay que evitar toda ambigüedad al definir cada paso. Puesto que el lenguaje humano es impreciso, los algoritmos se expresan mediante un lenguaje formal, ya sea matemático o de programación para un computador.
- **Efectivo:** se entiende por esto que una persona sea capaz de realizar el algoritmo de modo exacto y sin ayuda de una máquina en un lapso de tiempo finito.

En el diseño de un algoritmo debe considerarse, además, lo siguiente:

- **Entradas:** el algoritmo tendrá cero o más entradas, es decir, cantidades dadas antes de empezar el algoritmo. Estas cantidades pertenecen además a conjuntos especificados de objetos. Por ejemplo, pueden ser cadenas de caracteres, enteros, naturales, fraccionarios, etc. Se trata siempre de cantidades representativas del mundo real expresadas de tal forma que sean aptas para su interpretación por el computador.
- **Salidas:** el algoritmo tiene una o más salidas, en relación con las entradas.

Un algoritmo debe escribirse sin ceñirse a las reglas de un lenguaje. Existen varias formas para describir las operaciones de las que consta un algoritmo:

- **Descripción textual:** consiste en describir los pasos de forma narrativa.
- **Lista de operaciones:** es similar al textual, pero enumerando los pasos, utilizando variables, etc.
- **Diagrama de flujo:** son una representación gráfica en la que se utilizan cajas,rombos, flechas y otros símbolos para indicar los pasos del algoritmo.
- **Pseudocódigo:** se utilizan palabras clave para identificar las estructuras delalgoritmo, como alternativas, repeticiones, etc.

Partes de un algoritmo

Para poder llevar a cabo un algoritmo, éste debe obedecer a la estructura básica de un sistema, lo que permitirá dar una adecuada solución a un problema propuesto. Esta estructura debe contener lo siguiente:

- **Entrada (Input):** corresponde al insumo, a los datos necesarios que requiere elproceso para ofrecer los resultados esperados.
- **Proceso:** pasos necesarios para obtener la solución del problema o la situaciónplanteada.
- **Salida (Output):** Resultados arrojados por el proceso como solución.

Para ejemplificar lo mencionado recientemente, tomemos un caso matemático,donde se nos solicita un algoritmo que de solución al planteamiento siguiente.

Ejemplo 1: “Calcular el área de un triángulo rectángulo”

Solución ejemplo 1:

```
1. INICIO
2. Identificar las medidas de la base y la altura
3. Multiplicar la base por la altura y dividir por 2 el resultado
4. Mostrar resultado
5. FIN
```

Tenemos los datos de entrada: base y altura, los cuales nos permitirán realizar el

proceso de calcular el área de un triángulo rectángulo (base * altura / 2) para obtener como resultado el área solicitada.

Ejemplo 2: “Averiguar si un número es primo o no”.

Solución ejemplo 2:

Suponiendo que razonamos de la siguiente forma:

“Del análisis del hecho de que un número N es primo si sólo puede dividirse por sí mismo y por la unidad, un método que nos puede dar la solución sería dividir sucesivamente el número por 2, 3, 4..., etc. y, según el resultado, podríamos resolver el problema”. Un diseño del mismo sería:

```
1. INICIO
2. Poner X igual a 2 (X = 2, X, variable que representa a los posibles
divisores de N)
3. Dividir N por X (N/X)
4. Si el resultado es entero, entonces N no es primo, y saltar al
punto 9 (en caso contrario continuar el proceso en el siguiente
punto, 5)
5. Incrementar X en una unidad
6. Si X es menor que N saltar al punto 3 (en caso contrario continuar
el proceso en el siguiente punto, 7)
7. Declarar “N es primo”
8. Saltar al fin (punto 10)
9. Declarar “N no es primo”
10. FIN
```

Como parte del diseño de algoritmo está la selección de uno que sea razonablemente aceptable, entre todos los muchos posibles que resuelven el mismo problema (el ejemplo que acabamos de dar es claramente mejorable, pues si N no era divisible por 2 no tiene mucho sentido volver a preguntar si lo es por 4).

Durante el diseño es posible, y aconsejable, realizar comparaciones entre algoritmos que resuelven el mismo problema. La bondad de un algoritmo puede medirse por dos factores:

- El tiempo que se necesita para ejecutarlo. Para tener una idea aproximada de ello, basta con saber el número de instrucciones de cada tipo necesarias para resolver el problema.
- Los recursos que se necesitan para implantarlo.

Así, una vez diseñado un primer algoritmo, conviene realizar una evaluación del mismo, cuestión a veces nada banal y sobre la que volveremos en capítulos

posteriores. Si se decide que éste no es eficiente será necesario o bien diseñar uno nuevo o bien optimizar el original. Optimizar un algoritmo consiste en introducir modificaciones en él, tendentes a disminuir el tiempo que necesita para resolver el problema o a reducir los recursos que utiliza (en el ejemplo 2 el algoritmo se optimiza, si N se declara como primo cuando X supera a N/2).

Variables y tipos de dato

En programación, una variable es un espacio de memoria reservado para almacenar un valor que corresponde a un tipo de dato soportado por el lenguaje de programación. Una variable es representada y usada a través de una etiqueta (un nombre) que le asigna un programador o que ya viene predefinida.

Nombres de variables: Deben ser nombres que tengan relación con el dato que se almacena. Se puede utilizar caracteres y números, pero no símbolos ni tildes. El único carácter especial que se permite es el “guión bajo” (_).

Un ejemplo de lo anterior, es tener una variable llamada ciudad en la que almacenaremos el valor Viña del Mar.

```
Ciudad = “Viña del Mar”
```

Los tipos de datos son una restricción almacenada en un espacio en memoria con restricciones impuesta para la interpretación, manipulación, o representación de datos.

Tipos de datos comunes en lenguajes de programación son los tipos:

- **Tipo numérico:** Son aquellos datos que representan sólo números, ya sean enteros o con decimales.
- **Tipo caracteres:** Un carácter es una unidad de información mínima e indivisible de la escritura de una lengua. Un ejemplo de carácter es una letra, un número o un signo de puntuación. Este tipo de dato también abarca a los caracteres de control, como el salto de línea o el tabulador.

- **Tipo alfanumérico:** Este tipo corresponde a todos los caracteres del código ASCII, que incluye las letras desde la A a la Z, números del cero al nueve, y símbolos (incluye los tildes, la ñ y espacio). Si el dato 77 es almacenado en una

variable de tipo numérico, es posible realizar operaciones matemáticas con él. En este caso 77 + 3 es 80, pues el computador los ha sumado. Por otro lado, si77 es almacenado en una variable del tipo alfanumérico, entonces 77 + 3 es 773, pues el computador lo ha concatenado ya que se unen ambas cadenas de texto.

- **Tipo booleano:** Es aquel que puede representar valores de lógica binaria, dos valores que normalmente representan verdadero o falso.

Otro concepto básico en programación, además de las variables y los tipos de datos, es una constante. Esta corresponde a una zona de memoria que almacena un dato al igual que una variable, con la diferencia de que el dato almacenado no se puede cambiar.

Expresiones aritméticas y operadores

En informática y lenguajes de programación, se entiende por expresión aritmética a aquella donde los operadores que intervienen en ella son numéricos, el resultado es un número y los operadores son aritméticos. Los operadores aritméticos más comúnmente utilizados son: el signo más (+) se emplea para sumar dos valores, el signo menos (-) para restar un valor de otro, el asterisco (*) para multiplicar dos valores, la división (/) para dividir un valor por otro, y el signo % para obtener el resto de una división entera. Estos símbolos se conocen como operadores binarios, pues operan sobre dos valores o variables.

Los operadores aritméticos son aquellos que permiten resolver operaciones matemáticas básicas, incluso se pueden utilizar los paréntesis en las fórmulas matemáticas para problemas de precedencia.

Operador	Descripción	Símbolo	Ejemplo
Igual	Valida que dos datos sean iguales	==	a == b
Menor que	Valida si el primer dato es menor que el segundo	<	a < b
Menor o igual que	Permite comparar si el primer dato es menor o igual	<=	a <= b

	que el segundo		
Mayor que	Valida si el primer dato es mayor que el segundo	>	a > b
Mayor o igual que	Valida si el primer dato es mayor o igual que el segundo	>=	a >= b
Distinto	Valida que dos datos sean distintos	a !=b	a != b

Los operadores relacionales corresponden a los símbolos que se usan para comparar dos valores. Si el resultado de dicha comparación es correcta, la expresión evaluada es verdadera, en caso contrario es falsa.

Operador	Descripción	Símbolo	Ejemplo
Suma	Suma dos datos	+	a + b
Resta	Resta dos datos	-	a - b
Multiplicación	Multiplica dos datos	*	a * b
División	Divide dos datos. Sep puede considerar que una división por un dato igual a cero arroja un error	/	a / b
Módulo	Entrega el resto de una división entera	%	a % b

Por último, los operadores lógicos son aquellos que proporcionan un resultado a partir de que se cumpla o no una cierta condición, produciendo un resultado booleano, y siendo sus operandos también valores lógicos o asimilables a ellos (los valores numéricos son asimilados a cierto o falso según su valor sea cero o distinto de cero).

Esto genera una serie de valores que, en los casos más sencillos, pueden ser parametrizados con los valores numéricos 0 y 1. La combinación de dos o más operadores lógicos conforma una función lógica.

Las expresiones conectadas con los operadores && y || se evalúan de izquierda a derecha, y la evaluación se detiene tan pronto como el resultado verdadero o falso es conocido (muchos programas tienen una lógica que se basa en este propiedad).

- **AND** (el resultado es verdadero si ambas expresiones son verdaderas)
- **OR** (el resultado es verdadero si alguna expresión es verdadera)
- **NOT** (el resultado invierte la condición de la expresión)

Operador lógico AND		
X	**Y**	**Resultado**
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso
Falso	Verdadero	Falso
Falso	Falso	Falso

Operador lógico NOT	
X	**Resultado**
Verdadero	Falso
Falso	Verdadero

Operador lógico OR		
X	**Y**	**Resultado**
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

Estructuras de control condicional

Una estructura de control condicional consiste en la ejecución de una o más

instrucciones dependiendo de la evaluación de una condición.

En el ejemplo siguiente, si la condición es verdadera, se ejecuta el bloque de sentencias 1; de lo contrario, se ejecuta el bloque de sentencias 2.

```
Si (Condición) entonces
    Bloque de sentencias 1
Sino
    Bloque de sentencias 2
Fin Si
```

Entendiendo cómo funciona la estructura **SI .. ENTONCES**.

1. Se evalúa la condición y si ésta se cumple, entra al bloque
    a. Se ejecutan las instrucciones que se encuentra en su interior
    b. Luego de ejecutadas las instrucciones, el programa sigue con su secuencia, ejecutando la instrucción que se encuentre a continuación del fin del SI.
2. Si la condición no se cumple, no entra al bloque y el programa sigue con su secuencia, ejecutando la instrucción que se encuentre a continuación del fin de la sentencia SI

Se pueden plantear múltiples condiciones simultáneamente: Si se cumple la (Condición 1) se ejecuta (Bloque de sentencias 1). En caso contrario se comprueba la (Condición 2); si es cierta se ejecuta (Bloque de sentencias 2), y así sucesivamente hasta n condiciones. Si ninguna de ellas es cumple se ejecuta (Bloque de sentencias else).

```
Si (Condición 1) entonces
      (Bloque de sentencias 1)
Sino si (Condición 2) entonces
      (Bloque de sentencias 2)
Sino si (Condición 3) entonces
      (Bloque de sentencias 3)
Sino
      (Bloque de sentencias else)
Fin Si
```

Un caso práctico sería “ingresar 2 números por teclado, si el segundo número es cero, entonces muestre el primer número en pantalla, en caso contrario, mostrar el segundo número.”

Analizando el enunciado, se determina que el usuario del programa debe ingresar dos números y determinar si el segundo número es cero.

Así tenemos que las entradas son los dos números. Pero en cuanto a la salida, tenemos dos posibles, la primera cuando el segundo número ingresado es cero, y la segunda, cuando el segundo número ingresado no es cero.

Entonces, para entregar una solución se debe hacer uso de las estructuras de control, que permiten dirigir el flujo de acción dentro del programa, en base a la evaluación del cumplimiento de ciertas condiciones lógicas.

```
INICIO
      Leer num1, num2
      Si (num2 == 0) entonces
            Mostrar num1
      Sino
            Mostrar num2
      Fin Si
FIN
```

Otra sentencia de control son las de tipo **En caso de**. En este tipo de sentencias se especifica la variable a comparar y una lista de valores con los que comparar. Aquel que sea el verdadero, se ejecutará. Adicionalmente se puede dejar una respuesta por defecto, que tendría la misma función del “Sino”. La estructura se vería de la siguiente forma:

```
En-caso-de expresión
Caso expresión 1
      Bloque de sentencias 1
Caso expresión 2
      Bloque de sentencias 2
Por-defecto
      Bloque de sentencias 3
Fin Caso
```

Anexo: Referencias

1.- Algoritmos – Definición:

Referencia: http://ing.unne.edu.ar/pub/informatica/Alg_diag.pdf

2.- Algoritmos – Tipos

Referencia: https://www.lifeder.com/tipos-algoritmos/

3.- Estructuras de control

Referencia: http://di002.edv.uniovi.es/~dani/asignaturas/apuntes-leccion4.PDF

4.- Estructuras condicionales

Referencia: https://desarrolloweb.com/articulos/2225.php

5.- PSeInt

Referencia: http://pseint.sourceforge.net/

6.- Desarrollo de algoritmos en pseudocódigo con PSeInt

Referencia: https://www.youtube.com/watch?v=DHIi4dcaMEc