Tema 2:

Introducción a los algoritmos

***Objetivos**: este tema pretende mostrar al alumno cómo, a partir de unas especificaciones de un problema del mundo real, diseñar una solución para dicho problema (algoritmo) susceptible de ser codificada en un lenguaje de programación. Con este objetivo se describirán las propiedades básicas de cualquier algoritmo, un conjunto de bloques básicos que permiten la construcción de algoritmos y diversas formas de representación de los algoritmos. En el tema también se mostrarán las distintas fases que se deben de seguir para buscar una solución a un problema del mundo real.*

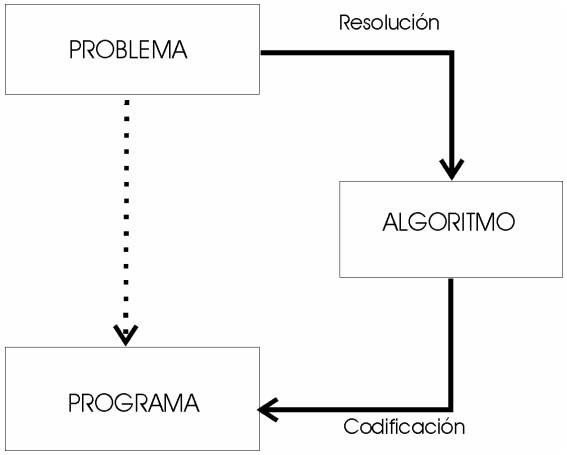
En este tema no se aborda ningún lenguaje de programación particular; sino que se muestra cómo diseñar soluciones a problemas que sean fáciles de implementar en cualquier lenguaje de programación.

Índice

1 Algoritmos

La resolución de un problema mediante un ordenador consiste en, partiendo de una especificación del problema, construir un programa que lo resuelva. Los procesos necesarios para la creación de un programa son:

1. Especificación y análisis del problema en cuestión.
2. Diseño de un algoritmo que resuelva el problema.
3. Codificación del algoritmo en un lenguaje de programación.
4. Validación del programa.



Un **algoritmo** es una secuencia ordenada de operaciones tal que su ejecución resuelve determinado problema. La palabra algoritmo viene de **Al-Khwarizmi**, sobrenombre del matemático árabe del siglo IX Mohámed ben Musa, que alcanzó gran reputación al enunciar paso a paso las reglas para sumar, restar, multiplicar y dividir números con decimales. Las características fundamentales que debe tener todo algoritmo son:

- Debe ser preciso, es decir, indicar el orden de realización de cada paso.
- Debe estar definido, esto es, si se ejecuta varias veces partiendo de las mismas condiciones iniciales debe obtenerse siempre el mismo resultado.
- Debe ser finito (debe tener un número finito de pasos).
- Debe ser independiente del lenguaje de programación que se emplee para implementarlo.

En cualquier algoritmo se pueden distinguir tres partes: la entrada de datos (la información sobre la cual se va a efectuar operaciones), procesamiento y salida del resultado (la información que debe proporcionar).

Ejemplos clásicos de algoritmos son el algoritmo de Euclides, que sirve para encontrar el máximo común divisor de 2 números enteros positivos A y B, o el de Newton-Raphson, para hallar una raíz de una función. Veamos como ejemplo el algoritmo de Euclides:

- **Paso 1**. Tomar el número mayor como dividendo y el menor como divisor.
- **Paso 2**. Calcular el resto de la división entera.
- **Paso 3**. Si el resto es igual a cero entonces ir al Paso 4. En caso contrario, tomar el divisor como nuevo dividendo y el resto como divisor y volver al Paso 2.
- **Paso 4**. El m.c.d. es el divisor da última división.

El algoritmo de Euclides para calcular el m.c.d. es definido, prevé todas las situaciones posibles para el resto la división (ser cero o diferente de cero); no es ambiguo: las acciones de dividir y comparar el resto con cero son claras y precisas, al igual que la obtención del número mayor; es finito, contando con cuatro pasos diferentes; y acaba en un tiempo finito cuando se cumple la condición del Paso 3 que hace avanzar al Paso 4, siendo este el punto de fin. Como ejemplo podemos considerar en el caso en el que A=9 y B=24;

- **Paso 1** D:=24, d:=9
- **Paso 2** R:= 6 (24 % 9)
- **Paso 3** D:=9, d:=6
- **Paso 2** R := 3
- **Paso 3** D:=6, d:=3
- **Paso 2** R := 0
- **Paso 4** m.c.d. := 3

1.1 Cómo escribir un algoritmo

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 texto, pero numerando los pasos, utilizando variables, etc. Es la descripción que se ha empleado para el algoritmo de Euclides.
- **Diagramas 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 del algoritmo, como alternativas, repeticiones, etc.

Las más utilizadas son las dos últimas.

1.2 Estructuras básicas en un algoritmo

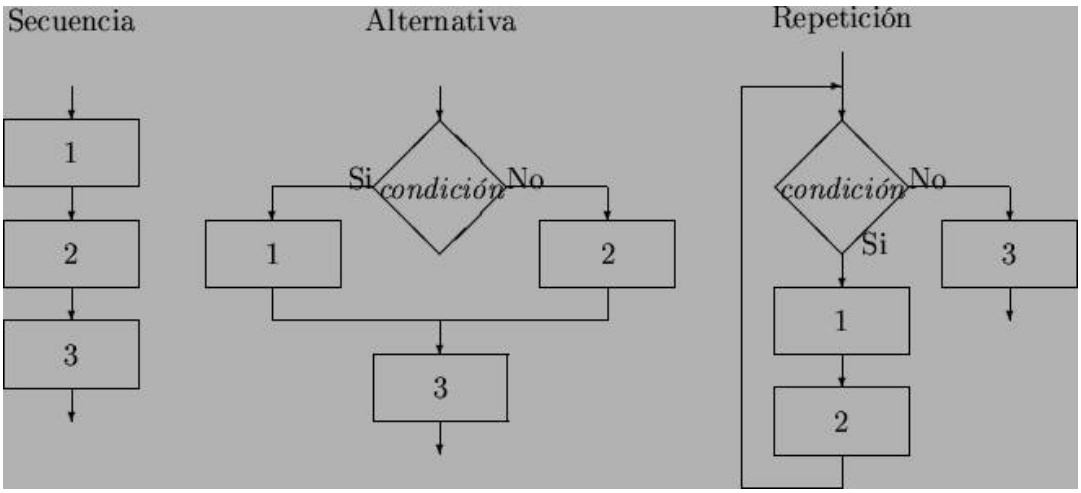
Un programa se puede definir como una secuencia ordenada de instrucciones cuyo propósito es realizar una determinada tarea. Por tanto, aparece el concepto de **flujo de ejecución** de un programa: este será el orden que siguen dichas instrucciones durante la ejecución del programa.

En principio, el flujo de ejecución de un programa será el orden en el cual se escriban las instrucciones (ejecución secuencial). Sin embargo, este esquema de ejecución impone serias limitaciones: a menudo hay ciertas porciones de código que sólo se deben ejecutar si se cumplen ciertas condiciones y tareas repetitivas que hay que ejecutar un gran número de veces. Por ello, se han definido una serie de estructuras de programación, independientes del lenguaje concreto que se está utilizando, que permiten crear programas cuyo flujo de ejecución sea más versátil que una ejecución secuencial. Los tipos de estructuras básicas que se pueden emplear en un algoritmo son:

- **Secuencia**: constituido por 0, 1 o N instrucciones que se ejecutan según el orden en el que han sido escritas. Es la estructura más simple y la pieza más básica a la hora de componer estructuras.
- **Selección, bifurcación o alternativa**: consta de una instrucción especial de decisión y de una o dos secuencias de instrucciones. La sentencia de decisión genera un resultado delimitado dentro de un rango preseleccionado (generalmente verdadero o falso) y,

dependiendo del resultado obtenido, se ejecuta o no la secuencia de instrucciones (si la estructura sólo cuenta con una secuencia) o se ejecuta una de las una secuencias de instrucciones (si la estructura cuenta con dos secuencias).

- **Iteración, bucle o repetición**: consta de una instrucción especial de decisión y de una secuencia. La instrucción de decisión sólo genera dos tipos de resultado (verdadero o falso) y la secuencia de instrucciones se ejecutará de modo reiterativo mientras que la instrucción de decisión genere el resultado verdadero; en caso contrario finalizará la ejecución de la secuencia. Los bucles pueden tener la instrucción de decisión al principio o al final. Si la condición está al final, el bucle siempre se ejecuta al menos una vez.



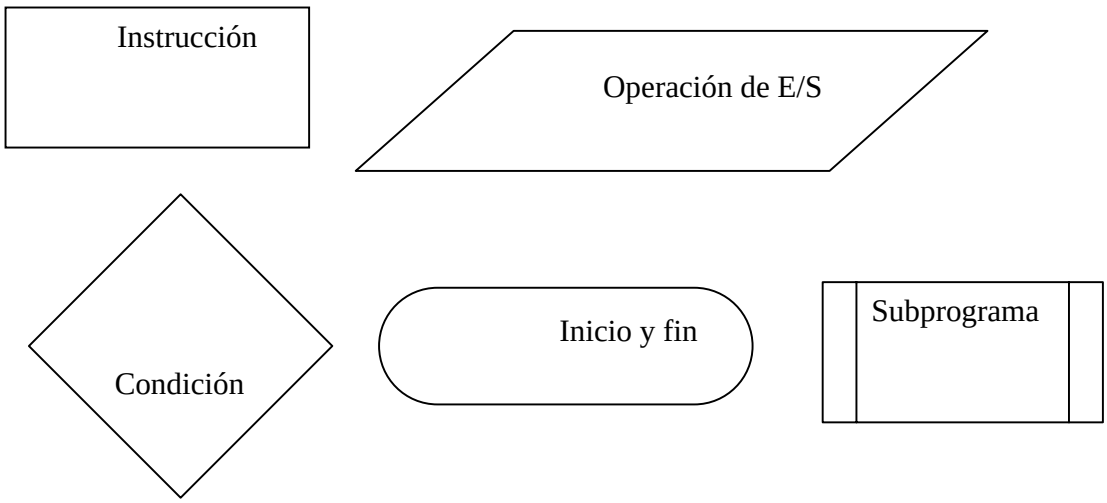
1.3 Diagramas de flujo y pseudocódigo

La notación de lista de operaciones utilizada para expresar el algoritmo de Euclides tiene varios inconvenientes:

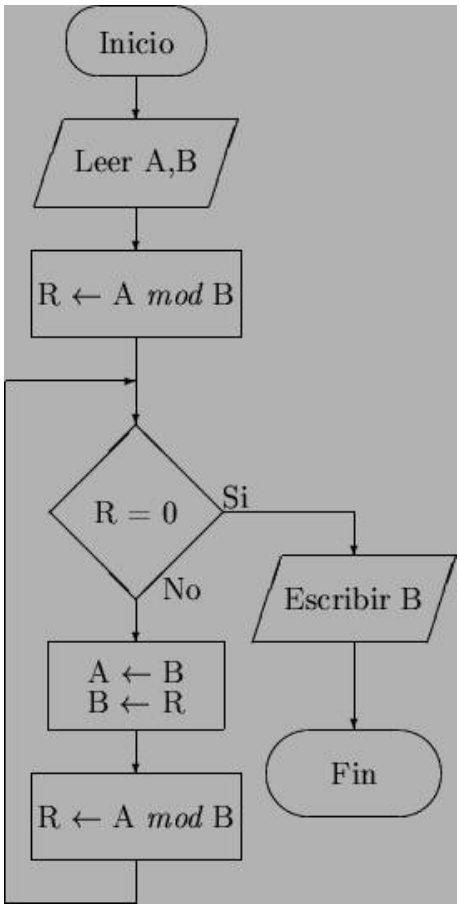
- Si se desea introducir un paso intermedio, es necesario cambiar las referencias de unos pasos a otros. (ir al paso **n**).
- La estructura del algoritmo (alternativas, repeticiones) no salta a la vista, hay que leer detenidamente la lista de pasos para enterarse.

- La descripción de cada paso, al ser en lenguaje común, puede ser poco clara para una persona distinta de la que lo creó.

Para solucionar estos problemas se utilizan los diagramas de flujo y los pseudocódigos. Los **diagramas de flujo** son representaciones gráficas de la lógica de un algoritmo mediante un conjunto de símbolos y flechas, utilizando texto abreviado para describir las tareas. Existen conjuntos de símbolos normalizados para representar los distintos pasos del programa, pero habitualmente se utilizan solamente cinco: elipses para el inicio y fin de programa, rectángulos para las acciones, rectángulos con dos barras verticales cerca de los extremos que van desde la parte superior a la inferior para los subprogramas, romboides para las operaciones de entrada y salida y rombos para las decisiones.



En un diagrama de flujo, el **flujo del programa**, es decir, el orden en el que se ejecuta la lista de operaciones, se obtiene *siguiendo las flechas*. Las estructuras pueden anidarse cuantas veces se quiera, es decir, una de las tareas en una rama de una bifurcación puede ser un bucle o viceversa. El diagrama de flujo correspondiente a algoritmo de Euclides es el siguiente:



Donde por A<--B se denota la asignación del valor B a A.

El **pseudocódigo** es un lenguaje creado a medida por el programador. Está formado por:

- Un conjunto finito de palabras del lenguaje natural (español, inglés, etc.), que se utilizan para expresar la estructura del programa.
- Descripciones en lenguaje natural, sin estructurar, junto con fórmulas, para expresar las tareas.

El conjunto de palabras lo construye el programador, generalmente tomando como base uno o varios lenguajes de programación estructurados, como C o Pascal. El pseudocódigo correspondiente a las estructuras algorítmicas básicas es:

Secuencia	Alternativa	Repetición o Bucle
operación1	**si** condición	**mientras** condición
operación2	operación1	operación1
operación3	**sino**	operación2
	operación2	**fin mientras**
	fin si	operación 3
	operación 3	

El pseudocódigo correspondiente al algoritmo de Euclides puede ser el siguiente:

```
PROGRAMA: ALGORITMO_EUCLIDES
INICIO
   VARIABLES
   A número entero
   B número entero
   R número entero

   ALGORITMO
       Leer A,B
       R ←A%B
       mientras R!=0
          A ← B
          B ← R
          R ← A%B
       fin mientras
       escribir b
FIN
```

Un diagrama de flujo es más visual y es menos estructurado que el pseudocódigo correspondiente. El pseudocódigo es más parecido a un lenguaje de programación (más fácil de traducir) y, por tanto, es más complicado de escribir. El pseudocódigo es más compacto que un diagrama de flujo.

1.3.1 Operaciones válidas en un algoritmo:

En la definición de un algoritmo se pueden emplear las siguientes operaciones:

- Aritméticas: suma, resta, multiplicación, división y módulo (resto de la división de enteros).
- Relacionales: <,>,<=,>=,etc
- Lógicas: Y, O, NO.

La instrucción básica es la asignación, que se denota como V <-- E y consiste en guardar el valor de una expresión E en la variable V. Las expresiones pueden ser constantes (ej. 5.36), variables (ej. A), dos expresiones operadas (ej. 5*A), una función aplicada a una expresión (ej. cos(A)), etc. Se pueden emplear las funciones matemáticas comunes: trigonométricas, exponencial, logaritmo, etc. No se pueden emplear las funciones sumatorio, productorio, integración, derivación, cálculo matricial, etc. ni cualquier otra función cuya implementación en el ordenador no sea trivial.

2 Fases del desarrollo de un programa

El proceso que va desde el planteamiento de un problema hasta obtener un programa que lo resuelve se divide en varias etapas, cada una con distintos objetivos y métodos de trabajo. Empezar a programar una vez planteado el problema, sin hacer ningún análisis ni recabar más información, suele producir un desperdicio de tiempo y recursos. Por ello el proceso de creación de un programa se suele dividir en varios pasos genéricos:

- **Especificación**: en esta fase se determina los límites y restricciones generales del problema: que tendrá que hacer, que no tendrá que hacer, bajo qué condiciones operará.... aquí intervienen los usuarios finales y, típicamente, la especificación surge de varias reuniones entre éstos y los desarrolladores.
- **Análisis**: el problema se analiza teniendo presente la especificación de los requisitos dados. En esta fase se determinan con la mayor precisión posible las tareas necesarias

para la resolución del problema y, si fuera necesario, estas tareas se descomponen en subtareas. La descripción de las tareas debe hacerse de forma independiente de lenguaje y del ordenador que se utilizará. En esta fase también debe detallarse cómo se supone que funcionará todo una vez finalizado el programa y qué pruebas habrá que hacer para saber si el funcionamiento es correcto.

- **Diseño**: tras analizar el problema se diseña la solución que permita crear un algoritmo para resolver el problema. En este nivel se tiene en cuenta consideraciones como el espacio de almacenamiento y el tiempo de ejecución que requerirán los programas. El nivel de detalle la descripción de cada uno de los programas depende de la complejidad de las tareas a desarrollar. Este es el momento de emplear el pseudocódigo en los diagramas de flujo. En esta fase suele escogerse el lenguaje o lenguajes de programación que se van emplear.
- **Codificación o implementación**: la solución se escribe en la sintaxis de un lenguaje de programación, obteniendo así nuestro programa.
- **Compilación, ejecución y verificación**: el programa se compila, lo cual, las primeras veces, suele dar lugar a errores de compilación (errores sintácticos en la escritura del código). Tras solucionar estos errores el programa se ejecuta y se le hace una serie de pruebas que tratan de ser representativas de todos los posibles modos de operación del software (se prueba cualquier tipo de entrada posible para el programa y se verifica si la salida es la esperada). Habitualmente esto lleva a descubrir nuevos fallos, esta vez de tipo semántico (el programa no hace lo que debe).
- **Documentación**: se documenta las diferentes fases del ciclo de vida del software, esencialmente el análisis, diseño y codificación. Se crean los manuales de usuario y de referencia y guías para el mantenimiento del software.
- **Explotación**: el programa comienza a ser usado por los usuarios finales en su entorno real de ejecución. Habitualmente, se descubren nuevos fallos y/o los usuarios plantean nuevos requerimientos para el software, lo cual requiere depurar el software (para corregir los fallos) y realizar mantenimiento (para cumplir los nuevos requerimientos de los usuarios).

3 Diseño de programas

En esta sección presentamos diversas estrategias para abordar la fase de diseño en la construcción de un programa.

3.1 Programación modular

La programación modular es un método de diseño flexible y potente que permite reducir la complejidad de un problema. En la programación modular el programa se divide en **módulos**, esto es, partes independientes, cada uno de los cuales ejecuta una única actividad o tarea y se codifica y prueba de modo independiente de los demás módulos. Cada programa consta de un módulo denominado **programa principal** que controla todo lo que sucede; éste va delegando las diversas tareas a realizar en **submódulos** (subprogramas) cada uno de los cuales tras terminar su tarea devuelve el control al módulo principal. Cada submódulo puede constar también de un conjunto de módulos si la tarea que debe realizar es suficiente compleja como para beneficiarse de una descomposición en módulos más simples. Un módulo puede transferir temporalmente el control a otro módulo; sin embargo, cada módulo debe eventualmente devolver control al módulo del cual recibió el control.

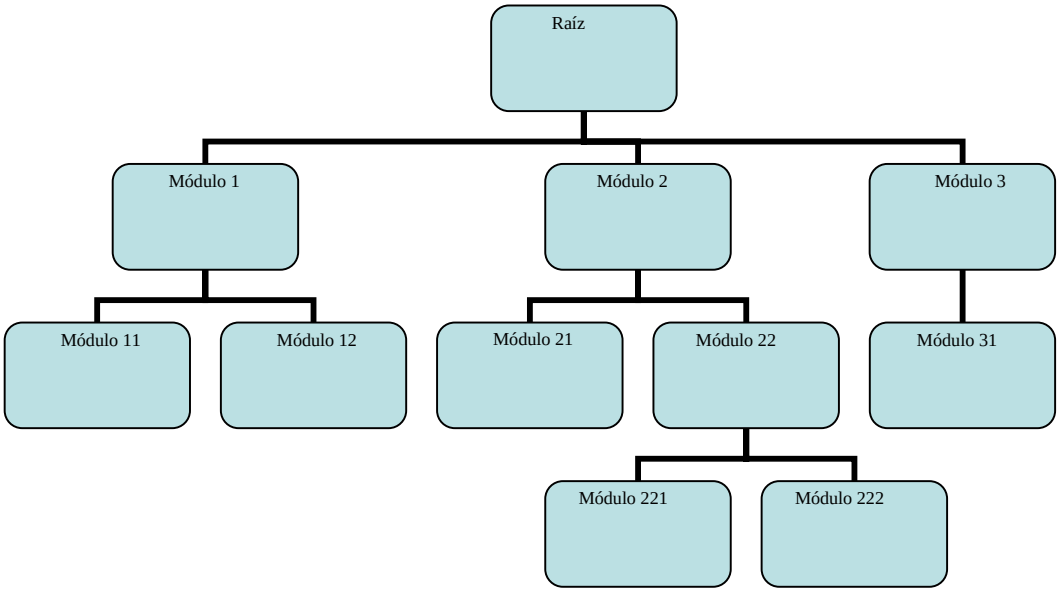
Dado que los módulos son independientes, diferentes programadores pueden trabajar simultáneamente en diferentes partes del mismo programa, tanto en la codificación como en su depuración. Esto reduce el tiempo de codificación del programa. Por otro lado, como cada uno de los módulos es muy simple, es fácil abordar el diseño del algoritmo correspondiente.

El proceso de puesta a punto de un módulo es el siguiente:

- Diseñar el algoritmo que realizará la tarea del módulo.
- Programar el módulo.
- Comprobar el correcto funcionamiento del módulo.
- Si fuera necesario, depurar el módulo.

- Combina el módulo con los módulos anteriores.

La descomposición de un programa en módulos independientes se conoce también como el método "divide y decenas". Cada módulo es diseñado con independencia de los demás y se sigue un método ascendente (se empiece diseñando los módulos más simples y se van agrupando para construir módulos más complejos hasta llegar a formar el programa principal) o descendente (se empieza diseñando el módulo correspondiente con el programa principal y cada tarea que debe realizar se va descomponiendo sucesivamente en otros módulos). En cualquiera de los casos se llegará a una descomposición final del programa en módulos organizados de forma jerárquica.



3.2 Programación estructurada

La programación estructurada requiere escribir un programa siguiendo las siguientes tres reglas:

- El programa se diseña siguiendo la aproximación modular.
- Cada uno de los módulos es diseñado siguiendo un método descendente (top-down); esto es, el módulo se va descomponiendo sucesivamente en tareas más simples, cada

una de las cuales será efectuada por un submódulo. Este proceso de descomposición jerárquico puede considerarse guiado por dos preguntas diferentes: primero debemos de responder a la pregunta "¿qué hace el módulo?". Una vez que su funcionalidad se ha sido determinada debemos de pasar a la siguiente etapa: "¿cómo lo hace?".

- Cada módulo se codifica utilizando tres estructuras de control básicas: secuencia, selección o alternativa, y repetición o bucle.

Para los que estén familiarizados con lenguajes de programación como Pascal, visual Basic o Fortran, la programación estructurada significa también programar sin la instrucción GOTO. La programación estructurada hace que los programas sean más fáciles de escribir, verificar, leer y mantener ya que al utilizar un número limitado de estructuras de control minimiza la complejidad de los problemas.

3.2.1 Teorema de la programación estructurada

En 1969 Bohm y Jcopini demostraron que un **programa propio** puede ser codificado utilizando solamente estructuras de control secuenciales, selectivas y repetitivas. Un programa propio es un programa que:

- Posee un solo punto de entrada y un solo punto de salida.
- Existen caminos de la entrada hasta la salida que se pueden seguir y pasan por todas parte del programa.
- Todas las instrucciones son ejecutables y no existen lazos o bucles infinitos.

4 Ejercicios:

Realizar pseudocódigo y diagrama de flujo de....

1. Dados dos números enteros calcular el cociente y el resto de su división entera.
2. Dados tres números A, B y C, calcular las soluciones de la ecuación Ax2+Bx+C=0.
3. Dado un número entero determine si el número es primo.
4. Dado un número entero determine el factorial de ese número.
5. Escribir el algoritmo para calcular la suma de los n primeros términos de la serie 1/x2 , para x=1,2,...,n.
6. Escribir el algoritmo de Newton-Raphson para encontrar una raíz de una función concreta. Este método parte de una estimación inicial de la raíz, x0, y va calculado aproximaciones sucesivas al valor de la misma utilizando la fórmula:

$$x_{i+1}=x_i-f(x_i)/f'(x_i)$$

Al ser un método iterativo, es necesario tener un criterio para que termine. Puede establecerse como criterio de terminación que la diferencia entre f(x) y 0 sea menor que un valor e pequeño. Además, por seguridad, se debe establecer un número máximo de iteraciones para que el algoritmo termine aunque no converja a una solución (de lo contrario, al ejecutarlo en el ordenador éste podría quedarse bloqueado).