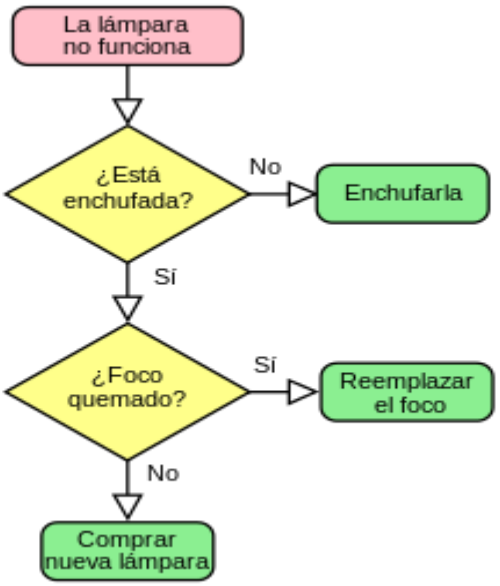
Algoritmo

En matemáticas, lógica, ciencias de la computación y disciplinas relacionadas, un algoritmo (del griego y latín, dixit algorithmus y este a su vez del matemático persa Al-Juarismi)1 es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite llevar a cabo una actividad mediante pasos sucesivos que no generen dudas a quien deba hacer dicha actividad.2 Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución. Los algoritmos son el objeto de estudio de la algoritmia.1

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 por parte de su patrón. Algunos ejemplos en matemática son el algoritmo de multiplicación, para calcular el producto, el algoritmo de la división para calcular el cociente de dos números, el algoritmo de Euclides para obtener el máximo común divisor de dos enteros positivos, o el método de Gauss para resolver un sistema de ecuaciones lineales.



Los diagramas de flujo sirven para representar Algoritmos de manera gráfica.

Definición formal

En general, no existe ningún consenso definitivo en cuanto a la definición formal de algoritmo. Muchos autores los señalan como listas de instrucciones para resolver un cálculo o un problema abstracto, es decir, que un número finito de pasos convierten los datos de un problema (entrada) en una solución (salida).[1][2][3][4][5][6] Sin embargo cabe notar que algunos algoritmos no necesariamente tienen que terminar o resolver un problema en particular. Por ejemplo, una versión modificada de la criba de Eratóstenes que nunca termine de calcular números primos no deja de ser un algoritmo.[7]

A lo largo de la historia varios autores han tratado de definir formalmente a los algoritmos utilizando modelos matemáticos. Esto fue realizado por Alonzo Church en 1936 con el concepto de "calculabilidad efectiva" basada en su cálculo lambda y por Alan Turing basándose en la máquina de Turing. Los dos enfoques son equivalentes, en el

sentido en que se pueden resolver exactamente los mismos problemas con ambos enfoques.[8] [9] Sin embargo, estos modelos están sujetos a un tipo particular de datos como son números, símbolos o gráficas mientras que, en general, los algoritmos funcionan sobre una vasta cantidad de estructuras de datos.[3] [1] En general, la parte común en todas las definiciones se puede resumir en las siguientes tres propiedades siempre y cuando no consideremos algoritmos paralelos:[7]

Tiempo secuencial. Un algoritmo funciona en tiempo discretizado –paso a paso–, definiendo así una secuencia de estados *computacionales* por cada entrada válida (la *entrada* son los datos que se le suministran al algoritmo antes de comenzar).

Estado abstracto. Cada estado computacional puede ser descrito formalmente utilizando una estructura de primer orden y cada algoritmo es independiente de su implementación (los algoritmos son objetos abstractos) de manera que en un algoritmo las estructuras de primer orden son invariantes bajo isomorfismo.

Exploración acotada. La transición de un estado al siguiente queda completamente determinada por una descripción fija y finita; es decir, entre cada estado y el siguiente solamente se puede tomar en cuenta una cantidad fija y limitada de términos del estado actual.

> En resumen, un algoritmo es cualquier cosa que funcione paso a paso, donde cada paso se pueda describir sin ambigüedad y sin hacer referencia a una computadora en particular, y además tiene un límite fijo en cuanto a la cantidad de datos que se pueden leer/escribir en un solo paso. Esta amplia definición abarca tanto a algoritmos prácticos como aquellos que solo funcionan en teoría, por ejemplo el método de Newton y la eliminación de Gauss-Jordan funcionan, al menos en principio, con números de precisión infinita; sin embargo no es posible programar la precisión infinita en una computadora, y no por ello dejan de ser algoritmos.[10] En particular es posible considerar una cuarta propiedad que puede ser usada para validar la tesis de Church-Turing de que toda función calculable se puede programar en una máquina de Turing (o equivalentemente, en un lenguaje de programación suficientemente general):[10]

Aritmetizabilidad. Solamente operaciones innegablemente calculables están disponibles en el paso inicial.

Medios de expresión de un algoritmo[editar]

Los algoritmos pueden ser expresados de muchas maneras, incluyendo al lenguaje natural, pseudocódigo, diagramas de flujo y lenguajes de programación entre otros. Las descripciones en lenguaje natural tienden a ser ambiguas y extensas. El usar pseudocódigo y diagramas de flujo evita muchas ambigüedades del lenguaje natural. Dichas expresiones son formas más estructuradas para representar algoritmos; no obstante, se mantienen independientes de un lenguaje de programación específico.

La descripción de un algoritmo usualmente se hace en tres niveles:

1. **Descripción de alto nivel**. Se establece el problema, se selecciona un modelo matemático y se explica el algoritmo de manera verbal, posiblemente con ilustraciones y omitiendo detalles.
2. **Descripción formal**. Se usa pseudocódigo para describir la secuencia de pasos que encuentran la solución.
3. **Implementación**. Se muestra el algoritmo expresado en un lenguaje de programación específico o algún objeto capaz de llevar a cabo instrucciones.

También es posible incluir un teorema que demuestre que el algoritmo es correcto, un análisis de complejidad o ambos.

Diagrama de flujo

Los diagramas de flujo son descripciones gráficas de algoritmos; usan símbolos conectados con flechas para indicar la secuencia de instrucciones y están regidos por ISO.

Los diagramas de flujo son usados para representar algoritmos pequeños, ya que abarcan mucho espacio y su construcción es laboriosa. Por su facilidad de lectura son usados como introducción a los algoritmos, descripción de un lenguaje y descripción de procesos a personas ajenas a la computación.

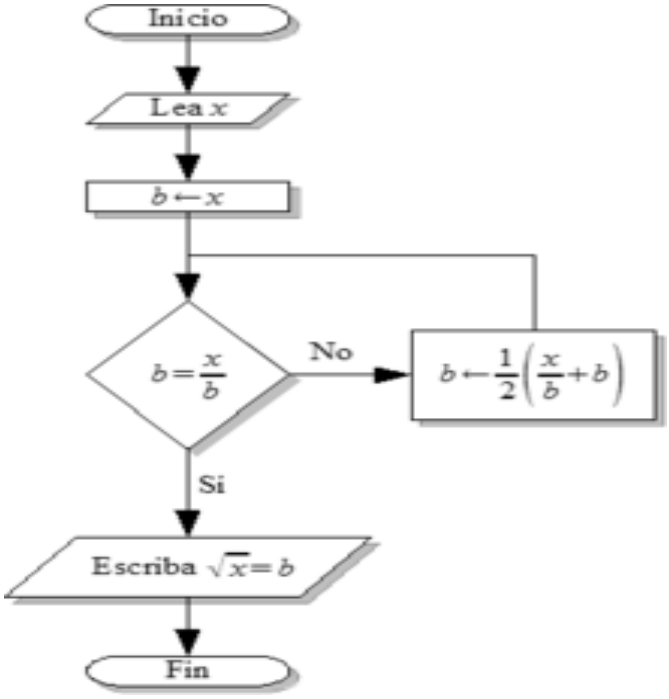


Diagrama de flujo que expresa un algoritmo

Para calcular la raíz cuadrada de un número {\displaystyle x} x

Pseudocódigo

El pseudocódigo (*falso lenguaje*, el prefijo *pseudo* significa *falso*) es una descripción de alto nivel de un algoritmo que emplea una mezcla de lenguaje natural con algunas convenciones sintácticas propias de lenguajes de programación, como asignaciones, ciclos y condicionales, aunque no está regido por ningún estándar. Es utilizado para describir algoritmos en libros y publicaciones científicas, y como producto intermedio durante el desarrollo de un algoritmo, como los diagramas de flujo, aunque presentan una ventaja importante sobre estos, y es que los algoritmos descritos en pseudocódigo requieren menos espacio para representar instrucciones complejas.

El pseudocódigo está pensado para facilitar a las personas el entendimiento de un algoritmo, y por lo tanto puede omitir detalles irrelevantes que son necesarios en una implementación. Programadores diferentes suelen utilizar convenciones distintas, que pueden estar basadas en la sintaxis de lenguajes de programación concretos. Sin embargo, el pseudocódigo, en general, es comprensible sin necesidad de conocer o utilizar un entorno de programación específico, y es a la vez suficientemente estructurado para que su implementación se pueda hacer directamente a partir de él.

Así el pseudocódigo cumple con las funciones antes mencionadas para representar algo abstracto los protocolos son los lenguajes para la programación. Busque fuentes más precisas para tener mayor comprensión del tema.

Sistemas formales

La teoría de autómatas y la teoría de funciones recursivas proveen modelos matemáticos que formalizan el concepto de *algoritmo*. Los modelos más comunes son la máquina de Turing, máquina de registro y funciones μ-recursivas. Estos modelos son tan precisos como un lenguaje máquina, careciendo de expresiones coloquiales o ambigüedad, sin embargo se mantienen independientes de cualquier computadora y de cualquier implementación.

Implementación

Muchos algoritmos son ideados para implementarse en un programa. Sin embargo, los algoritmos pueden ser implementados en otros medios, como una red neuronal, un circuito eléctrico o un aparato mecánico y eléctrico. Algunos algoritmos inclusive se diseñan especialmente para implementarse usando lápiz y papel. El algoritmo de multiplicación tradicional, el algoritmo de Euclides, la criba de Eratóstenes y muchas formas de resolver la raíz cuadrada son solo algunos ejemplos.

Variables

Son elementos que toman valores específicos de un tipo de datos concreto. La declaración de una variable puede realizarse comenzando con **var**. Principalmente, existen dos maneras de otorgar valores iniciales a variables:

1. Mediante una sentencia de asignación.
2. Mediante un procedimiento de entrada de datos (por ejemplo: 'read').

Ejemplo:

```
 ...
i:=1;
read(n);
while i < n do begin
   (* cuerpo del bucle *)
   i := i + 1
end;
 ...
```

Estructuras secuenciales

La estructura secuencial es aquella en la que una acción sigue a otra en secuencia. Las operaciones se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. La asignación de esto consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor. La asignación se puede clasificar de la siguiente forma:

1. **Simples**: Consiste en pasar un valor constante a una variable (a ← 15)
2. **Contador**: Consiste en usarla como un verificador del número de veces que se realiza un proceso (a ← a + 1)
3. **Acumulador**: Consiste en usarla como un sumador en un proceso (a ← a + b)

4. **De trabajo**: Donde puede recibir el resultado de una operación matemática que involucre muchas variables (a ← c + b*1/2).

Un ejemplo de estructura secuencial, como obtener el área de un triángulo:

```
Inicio
...
    float b, h, a;
    printf("Diga la base");
    scanf("%f", &b);
    printf("Diga la altura");
    scanf("%f", &h);
    a = (b*h)/2;
    printf("El área del triángulo es %f", a)
...
Fin
```

Algoritmos como funciones

Un algoritmo se puede concebir como una función que transforma los datos de un problema (entrada) en los datos de una solución (salida). Más aún, los datos se pueden representar a su vez como secuencias de bits, y en general, de símbolos cualesquiera.[1 9 11] Como cada secuencia de bits representa a un número natural (véase Sistema binario), entonces los algoritmos son en esencia funciones de los números naturales en los números naturales que sí se pueden calcular. Es decir que todo algoritmo calcula una función $\mathbf{F: N \rightarrow N}$ donde cada número natural es la codificación de un problema o de una solución.

En ocasiones los algoritmos son susceptibles de nunca terminar, por ejemplo, cuando entran a un bucle infinito. Cuando esto ocurre, el algoritmo nunca devuelve ningún valor de salida, y podemos decir que la función queda indefinida para ese valor de entrada. Por esta razón se considera que los algoritmos son funciones parciales, es decir, no necesariamente definidas en todo su dominio de definición.

Cuando una función puede ser calculada por medios algorítmicos, sin importar la cantidad de memoria que ocupe o el tiempo que se tarde, se dice que dicha función es computable. No todas las funciones entre secuencias datos son computables. El problema de la parada es un ejemplo.

Esquemática de un algoritmo solucionando un problema de ciclo hamiltoniano.

Análisis de algoritmos

Artículo principal: Análisis de algoritmos

Como medida de la eficiencia de un algoritmo, se suelen estudiar los recursos (memoria y tiempo) que consume el algoritmo. El análisis de algoritmos se ha desarrollado para

obtener valores que de alguna forma indiquen (o especifiquen) la evolución del gasto de tiempo y memoria en función del tamaño de los valores de entrada.

El análisis y estudio de los algoritmos es una disciplina de las ciencias de la computación y, en la mayoría de los casos, su estudio es completamente abstracto sin usar ningún tipo de lenguaje de programación ni cualquier otra implementación; por eso, en ese sentido, comparte las características de las disciplinas matemáticas. Así, el análisis de los algoritmos se centra en los principios básicos del algoritmo, no en los de la implementación particular. Una forma de plasmar (o algunas veces "codificar") un algoritmo es escribirlo en pseudocódigo o utilizar un lenguaje muy simple tal como Lexico, cuyos códigos pueden estar en el idioma del programador.

Algunos escritores restringen la definición de algoritmo a procedimientos que deben acabar en algún momento, mientras que otros consideran procedimientos que podrían ejecutarse eternamente sin pararse, suponiendo el caso en el que existiera algún dispositivo físico que fuera capaz de funcionar eternamente. En este último caso, la finalización con éxito del algoritmo no se podría definir como la terminación de este con una salida satisfactoria, sino que el éxito estaría definido en función de las secuencias de salidas dadas durante un periodo de vida de la ejecución del algoritmo. Por ejemplo, un algoritmo que verifica que hay más ceros que unos en una secuencia binaria infinita debe ejecutarse siempre para que pueda devolver un valor útil. Si se implementa correctamente, el valor devuelto por el algoritmo será válido, hasta que evalúe el siguiente dígito binario. De esta forma, mientras evalúa la siguiente secuencia podrán leerse dos tipos de señales: una señal positiva (en el caso de que el número de ceros sea mayor que el de unos) y una negativa en caso contrario. Finalmente, la salida de este algoritmo se define como la devolución de valores exclusivamente positivos si hay más ceros que unos en la secuencia y, en cualquier otro caso, devolverá una mezcla de señales positivas y negativas.

Ejemplo de algoritmo

El problema consiste en encontrar el máximo de un conjunto de números. Para un ejemplo más complejo véase Algoritmo de Euclides.

Descripción de alto nivel

Dado un conjunto finito de números, se tiene el problema de encontrar el número más grande. Sin pérdida de generalidad se puede asumir que dicho conjunto no es vacío y

que sus elementos están numerados como .

Es decir, dado un conjunto se pide encontrar tal que para todo

elemento que pertenece al conjunto .

Para encontrar el elemento máximo, se asume que el primer elemento ( ) es el máximo; luego, se recorre el conjunto y se compara cada valor con el valor del máximo número encontrado hasta ese momento. En el caso que un elemento sea mayor que el máximo, se asigna su valor al máximo. Cuando se termina de recorrer la lista, el máximo número que se ha encontrado es el máximo de todo el conjunto.

Descripción formal

El algoritmo puede ser escrito de una manera más formal en el siguiente pseudocódigo:

Algoritmo Encontrar el máximo de un conjunto

```
función max(        )
        //        es un conjunto no vacío de números//
              ←        //        es el número de elementos de        //
              ←
        para        ←        hasta        hacer
        si        entonces
              ←
        devolver
```

Sobre la notación:

- "←" representa una asignación: ← significa que la variable toma el valor de ;
- "**devolver**" termina el algoritmo y devuelve el valor a su derecha (en este caso, el máximo de).

Implementación

En lenguaje C++:

```
int max(int c[], int n)
{
   int i, m = c[0];
   for (i = 1; i < n; i++)
      if (c[i] > m) m = c[i];
   return m;
}.
```