Algoritmo **(De Wikipedia)**

Un **algoritmo** es un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema. De un modo más formal, un algoritmo es una secuencia finita de operaciones realizables, no ambiguas, cuya ejecución da una solución de un problema.

El término algoritmo no está exclusivamente relacionado con las matemáticas, ciencias de la computación o informática. En realidad, en la vida cotidiana empleamos algoritmos en multitud de ocasiones para resolver diversos problemas. Ejemplos son el uso de una lavadora (se siguen las instrucciones), para cocinar (se siguen los pasos de la receta). También, existen ejemplos de índole matemática, como el algoritmo de la división para calcular el cociente de dos números, el algoritmo de Euclides para calcular el máximo común divisor de dos enteros positivos, o incluso el método de Gauss para resolver sistemas de ecuaciones.

Concepto

Sistema por el cual se llega a una solución, teniendo en cuenta que debe de ser: definido, finito y preciso. Por preciso entendemos que cada paso a seguir tiene un orden; finito implica que tiene un determinado número de pasos, o sea que tiene un fin; y definido que si se sigue el mismo proceso más de un vez llegaremos al mismo resultado.

Estructura Básica: 1-inicio 2-datos de entrada (operaciones básicas) 3-procesamiento de los datos 4-datos de salida 5-fin

Historia

La palabra *algoritmo* proviene del nombre del matemático persa llamado Abu Abdullah Muhammad bin Musa al-Khwarizmi que vivió entre los siglos VIII y IX. Su trabajo consistió en preservar y difundir el conocimiento de la antigua Grecia y de la India. Sus libros eran de fácil comprensión, he aquí que su principal valor no fuera el de crear nuevos teoremas o nuevas corrientes de pensamiento, sino el simplificar las matemáticas a un nivel lo suficientemente bajo para que pudiera ser comprendido por un amplio público. Cabe destacar, cómo él señaló las virtudes del sistema decimal indio, (en contra de los sistemas tradicionales árabes) y cómo explicó que mediante una especificación clara y concisa de cómo calcular sistemáticamente, se podrían definir algoritmos que fueran usados en dispositivos mecánicos en vez de las manos (por ejemplo, ábacos). También estudió la manera de reducir las operaciones que formaban el cálculo. Es por esto que aún no siendo él el creador del primer algoritmo, el concepto lleva aunque no su nombre, sí su pseudónimo.

Así, de la palabra *algorismo* que originalmente hacía referencia a las reglas de uso de la aritmética utilizando dígitos arábigos, se evolucionó a la palabra latina, derivación de al-Khwarizmi, algobarismus, y luego más tarde mutó en *algoritmo* en el siglo XVIII. La palabra ha cambiado de forma que en su definición se incluyen a todos los procedimientos finitos para resolver problemas.

Ya en el siglo XIX, se produjo el primer algoritmo escrito para un computador. La autora fue Ada Byron en cuyos escritos se detallaban la máquina analítica en 1842. Es por ello que es considerada por muchos como la primera programadora aunque, desde Charles Babbage nadie completó su máquina, por lo que el algoritmo nunca se implementó.

Implementación

Algunas veces en una red neuronal biológica (por ejemplo, el cerebro humano implementa la aritmética básica o, incluso, una rata sigue un algoritmo para conseguir comida), también en circuitos eléctricos, en instalaciones industriales o maquinaria pesada.

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 éste 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 es 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

Se presenta el algoritmo para encontrar el máximo de un conjunto de enteros positivos. Se basa en recorrer una vez cada uno de los elementos, comparándolo con un valor concreto (el máximo entero encontrado hasta ahora). En el caso de que el elemento actual sea mayor que el máximo, se le asigna su valor al máximo.

El algoritmo escrito de una manera más formal, esto es, en pseudocódigo tendría el siguiente aspecto:

```
ALGORITMO Maximo
  ENTRADAS: Un conjunto no vacío de enteros C.
  SALIDAS: El mayor número en el conjunto C.

  maximo ← -∞
  PARA CADA elemento EN el conjunto C, HAZ
    SI  item > maximo, HAZ
      maximo ←  elem
  DEVUELVE maximo
```

Sobre la notación:

- "←" representa la asignación entre dos elementos. Por ejemplo, con *maximo* ← *elem* significa que el número *maximo* cambia su valor por el de *elem*.
- "**DEVUELVE**" termina el algoritmo y devuelve el valor a su derecha (en este caso maximo).

Como medida de la bondad de un algoritmo, se suelen estudiar los recursos (memoria y tiempo) que consume el algoritmo. Por eso, se ha desarrollado el análisis de algoritmos 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. Por ejemplo, el algoritmo anterior tiene un orden de eficiencia en tiempo de O(n), en la notación O mayúscula n es el tamaño de la entradas; en este caso n es la el número de elementos de C. Además, como el algoritmo necesita recordar un único valor (el máximo) requiere un espacio de O(1) (hay que tener en cuenta que el tamaño de las entradas no se considera como memoria usada por el algoritmo).

Técnicas de diseño de algoritmos

- Algoritmos greedy: Informalmente, podemos decir que este tipo de algoritmos selecciona los elementos del conjunto de candidatos en un determinado orden hasta encontrar una solución; es decir, calcula la solución al problema tomando en cada paso la opción más prometedora. En la mayoría de los casos la solución no es óptima.
- Algoritmos paralelos
- Algoritmos probabilísticos
- Algoritmos determinísticos
- Algoritmos no deterministicos
- Divide y vencerás (*divide & conquer*, en inglés): este tipo de algoritmos dividen el problema en subconjuntos disjuntos obteniendo una solución de cada uno de ellos para después unirlas, logrando así la solución al problema completo.
- Heurísticas: algoritmos que encuentran soluciones a problemas basándose en un conocimiento anterior (a veces llamado experiencia) de los mismos.
- Programación dinámica: intenta resolver problemas disminuyendo su coste espacial aumentando el coste computacional.
- Ramificación y acotación: también conocidos como *ramificación y poda*, *branch and bound*. Este método se basa en la construcción de las soluciones al problema mediante un árbol implícito que se recorre de forma controlada encontrando las mejores soluciones.
- Vuelta Atrás: al igual que el método ramificación y acotación, vuelta atrás (*backtracking*, en inglés) se construye el espacio de soluciones del problema en un árbol que se examina completamente, almacenando las soluciones menos costosas.

Temas relacionados

- Cota superior asintótica
- Cota inferior asintótica
- Cota ajustada asintótica
- Complejidad computacional

Disciplinas relacionadas

- Ciencias de la Computación
- Complejidad computacional
- Informática
- Inteligencia artificial
- Investigación operativa
- Matemáticas
- Programación

Libros sobre Algoritmia

- *The Art of Computer Programming*, Knuth, D. E. [quien fue también, el creador del TeX]
- *Introduction to Algorithms (2nd ed)*, Cormen, T. H., Leiserson, C. E., Rivest, R. L. y Stein, C.
- *Introduction to Algorithms. A Creative Approach*, Mamber, U.
- *Algorithms in C (3r ed)*, Sedgewick, R. [también existen versiones en C++ y Java]

Enlaces externos

- Portal de algoritmia
- Técnicas de Diseño de Algoritmos manual que explica y ejemplifica los distintos paradigmas de diseño de algoritmos. Rosa Guerequeta y Antonio Vallecillo (profesores de la Universidad de Málaga).
- Transparencias de la asignatura "Esquemas Algorítmicos", Campos, J.
- Apuntes y problemas de Algorítmica por Domingo Giménez Cánovas
- Algoritmos básicos