

Algorithm

An *algorithm* is a recipe, method, or technique for doing something. The essential feature of an algorithm is that it is made up of a *finite* set of rules or operations that are unambiguous and simple to follow (computer scientists use technical terms for these two properties: *definite* and *effective*, respectively). It is obvious from this definition that the notion of an algorithm is somewhat imprecise, a feature it shares with all foundational mathematical ideas—for instance, the idea of a set. This imprecision arises because being unambiguous and simple are relative, context-dependent terms. However, usually algorithms are thought of as recipes, methods, or techniques for getting *computers* to do something, and when restricted to computers, the term “algorithm” becomes more precise, because then “unambiguous and simple to follow” means “a computer can do it.” The connection with computers is not necessary, however. If a person equipped only with pencil and paper can complete the operations, then the operations constitute an algorithm.

A famous example of an algorithm (dating back at least to Euclid) is finding the greatest common divisor (GCD) of two numbers, m and n .

Step 1. Given two positive integers, set m to be the larger of the two; set n to be the smaller of the two.

Step 2. Divide m by n . Save the remainder as r .

Step 3. If $r = 0$, then halt; the GCD is n .

Step 4. Otherwise, set m to the old value of n , and set n to the value of r . Then go to step 2.

A tax form is also a relatively good example of an algorithm because it is finite and the instructions for completing it are mechanically and finitely completable (at least that is the intention). The recipe for cowboy chocolate cake (with two mugs of coffee) is not really an algorithm because its description is not definite enough (how much is a mug of coffee?). Of course, all computer programs are algorithms.

It should also be noted that algorithms are by no means restricted to numbers. For example, alphabetizing a list of words is also an algorithm. And, one interpretation of the computational hypothesis of the mind is that thinking itself is an algorithm—or perhaps better, the result of many algorithms working simultaneously.

Now to flesh out the definition. An algorithm is an unambiguous, precise, list of simple operations applied mechanically and systematically to a set of tokens or objects (e.g., configurations of chess pieces, numbers, cake ingredients, etc.). The initial state of the tokens is the input; the final state is the output. The operations correspond to *state transitions* where the states are the configuration of the tokens, which changes as operations are applied to them. Almost everything in sight is assumed to be finite: the list of operations itself is finite (there might be a larger but still finite set from which the operations are drawn) and each token is itself finite (or, more generally, a finitely determinable element in the set). Usually, the input, output, and intermediate sets of tokens are also finite, but this does not have to be the

case, at least in theory; indeed these sets can be continuous (see Blum, Shub, and Smale 1989). An algorithm describes a process that the tokens participate in. This process (a computation) is either in a certain state or it is not, it may either go to a certain next state from its current state or it may not, and any transition taken is finite. (One way to relax this definition is to allow the state transitions to be probabilistic, but that doesn't affect their finiteness.)

And finally, in more technical parlance, an algorithm is an *intensional* definition of a special kind of function—namely a *computable* function. The intensional definition contrasts with the *extensional* definition of a computable function, which is just the set of the function's inputs and outputs. Hence, an algorithm describes *how* the function is computed, rather than merely *what* the function is. The connection with computable functions is crucial. A function F is computable if and only if it is describable as an algorithm. The relationship between the extensional definition of F and an intensional definition of it is interesting. There is one extensional definition of F , but there are an infinite number of intensional definitions of it, hence there are an infinite number of algorithms for every extensionally-described computable function F . (Proof: you can always construct a new, longer algorithm by adding instructions that essentially do nothing. Of course, usually we seek the *canonical* algorithm—the shortest and most efficient one.)

A computable function is a function whose inputs and outputs can be produced by a Turing machine. Church's thesis states that *all* the computable functions can be computed by a Turing machine (see CHURCH-TURING THESIS). The best way to understand Church's thesis is to say that Turing computability exhausts the notion of computability. Importantly, not all functions are computable, so not all functions are algorithmically describable (this was a profound discovery, first proved by TURING 1936; Church 1936a; and Kleene 1936; it and related results are among the greatest achievements of twentieth-century mathematics).

Sometimes algorithms are simply equated with Turing machines. The definition given here is logically prior to the notion of a Turing machine. This latter notion is intended to formally capture the former. Gödel (1931; 1934), among his other achievements, was the first to do this, to link a formal definition with an intuitive one: he identified a formally defined class of functions, the recursive functions, with the functions that are computable, that is, with the functions for which algorithms can be written.

This completes the definition of “algorithm.” There are a few loose ends to tie up, and connections to be made. First, mathematicians and computer scientists sometimes sharply restrict the definition of “algorithm.” They take the definition of “algorithm” given here, and use it to define the notion of an *effective procedure*. Then they define an algorithm as an effective procedure that always halts or terminates (not all procedures or computer programs do terminate—sometimes on purpose, sometimes accidentally).

Second, in common parlance, an algorithm is a recipe for *telling* a computer what to do. But this definition does more harm than good because it obliterates the crucial notion of a virtual machine, while it subtly reinforces the

idea that a homunculus of some sort is doing all the work, and this in turn can reinforce the idea of a “ghost in the machine” in the COMPUTATIONAL THEORY OF MIND. So this folk definition should be avoided when precision is needed. Another problem with the folk definition is that it does not do justice the profundity of the notion of an algorithm as a description of a *process*. It is fair to regard algorithms as being as crucial to mathematics as sets. A set is a collection of objects. An intentional definition of a set describes all and only the objects in the set. An algorithm describes a collection of objects that does something. It would be impossible to overstate the importance of this move from statics to dynamics.

Third, the connection between algorithms and COMPUTATION is quite tight. Indeed, some mathematicians regard algorithms as abstract descriptions of computing devices. When implemented on a standard computer, such descriptions cease to be abstract and become real computing devices known as *virtual machines* (virtual does not mean not real, here). A virtual machine is the machine that does what the algorithm specifies. A virtual machine exists at some level higher than the machine on which the algorithm is implemented. For example: a word processor is a virtual machine that exists on top of the hardware machine on which it is implemented. The notion of a virtual machine is very important to cognitive science because it allows us to partition the study of the mind into levels, with neurochemistry at the bottom (or near the bottom) and cognitive psychology near the top. At each level, different methods and technical vocabularies are used. One of the crucial facts about virtual machines is that no one machine is more important than the rest. So it is with the brain and the rest of the nervous system that make up thinking things. Theories at all levels are going to be needed if we are to completely and truly understand the mind.

See also COMPUTATION AND THE BRAIN; FORMAL SYSTEMS, PROPERTIES OF

—Eric Dietrich

References

- Blum, L., M. Shub, and S. Smale. (1989). On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions, and universal machines. *Bulletin of the American Mathematical Society* 21 (1): 1–46.
- Church, A. (1936a). A note on the entscheidungsproblem. *Journal of Symbolic Logic* 1: 40–41, 101–102.
- Gödel, K. (1931). On formally undecidable propositions of Principia Mathematica and related systems I. *Monatshefte für Mathematik und Physik* 38: 173–198. (Reprinted in J. Heijenoort, Ed., (1967), *From Frege to Gödel*. Cambridge, MA: Harvard University Press, pp. 592–617.)
- Gödel, K. (1934/1965). On undecidable propositions of formal mathematical systems. In M. Davis, Ed., *The Undecidable*. New York: Raven Press, pp. 41–71.
- Kleene, S. C. (1936). General recursive functions of natural numbers. *Mathematische Annalen* 112: 727–742.
- Turing, A. (1936). On computable numbers with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society* series 2, 42: 230–265 and 43: 544–546.

Further Readings

- Church, A. (1936b). An unsolvable problem of elementary number theory. *American Journal of Mathematics* 58: 345–363.
- Dietrich, E. (1994). Thinking computers and the problem of intentionality. In E. Dietrich, Ed., *Thinking Computers and Virtual Persons: Essays on the Intentionality of Machines*. San Diego: Academic Press, pp. 3–34.
- Fields, C. (1989). Consequences of nonclassical measurement for the algorithmic description of continuous dynamical systems. *Journal of Experimental and Theoretical Artificial Intelligence* 1: 171–189.
- Knuth, D. (1973). *The Art of Computer Programming*, vol. 1: *Fundamental Algorithms*. Reading, MA: Addison-Wesley.
- Rogers, H. (1967). *The Theory of Recursive Functions and Effective Computability*. New York: McGraw-Hill.

A-Life

See ARTIFICIAL LIFE; EVOLUTIONARY COMPUTATION

Altruism

In biology, *altruism* has a purely descriptive economic meaning: the active donation of resources to one or more individuals at cost to the donor. Moral values or conscious motivations are not implied, and the ideas are as applicable to plants as to animals. Four evolutionary causes of altruism will be considered here: kin selection, reciprocation, manipulation, and group selection. Each implies demonstrably different patterns for what individuals donate what resources to whom and under what circumstances and may suggest different motivational and emotional experiences by both donor and recipient.

It may seem that Darwinian EVOLUTION, directed by natural selection, could never favor altruism. Any avoidable activity that imposes a cost, always measured as reduced reproductive fitness, would be eliminated in evolution. This view is too simple. Natural selection should minimize costs whenever possible, but successful reproduction always requires donation of resources to offspring, at least by females putting nutrients into eggs. A closer look shows that offspring are important to natural selection only because they bear their parents' genes, but this is true of all relatives. From the perspective of genetics and natural selection, the survival and reproduction of any relative are partly equivalent to one's own survival and reproduction. So there must be an evolutionary force of *kin selection* that favors altruism between associated relatives.

Kin selection, first clearly formulated by Hamilton (1964), can be defined as selection among individuals for the adaptive use of cues indicative of kinship. The products of mitotic cell division are exactly similar genetically, and their special physical contact is reliable evidence of full kinship. This accounts for the subservience of somatic cells in a multicellular organism to the reproductive interests of the germ cells. Kin selection also accounts for the generally benign relations among young animals in the same nest. Such early proximity is often a cue indicative of close kinship. Nestmates are often full sibs, with a genetic relation-