algorithm

Canonical name	Algorithm
Date of creation	2013-03-22 17:15:23
Last modified on	2013-03-22 17:15:23
Owner	jk81 (17362)
Last modified by	jk81 (17362)
Numerical id	15
Author	jk81 (17362)
Entry type	Definition
Classification	msc 68W01
Synonym	effective procedure
Synonym	Matiyasevich's theorem
Defines	computable function
Defines	Hilbert's 10th problem
Defines	Matiyasevič's theorem

Intuitive Introduction

An algorithm, generally speaking, is a collection of instructions that provides a step-by-step procedure on how to complete a particular task. Examples of algorithms in real life range from a simple recipe on how to bake a fruit pie, to an owner's manual of a newly purchased television, to the complicated process of producing an automobile in an assembly line.

In mathematics, an algorithm can be thought of, in a narrower sense, as a set of mathematical instructions on how to complete a mathematical task. For example, if we were interested in determining that, given an integral quadratic equation

$$ax^2 + bx + c = 0, \text{ where } a, b, c \in \mathbb{Z} \text{ and } a \neq 0$$

if an integral solution for x exists, a possible algorithm for solving this problem would be

1. first of all, check if $b^2 - 4ac$ is some square d^2 with $d \in \mathbb{Z}$.
2. If not, then no integral solutions exist.
3. If so, check next if $2a$ divides either $-b + d$ or $-b - d$.
4. If not, then no integral solutions exist.
5. If so, then yes, an integral solution exists.

Algorithms abound in mathematics, from the familiar adding and multiplying two integers, to solving a system of linear equations, to processes much more complicated, such as determining if two groups are isomorphic.

To understand algorithms a little better, let us see what all of the algorithms mentioned so far have in common. Three essential features of a typical algorithm that come to mind are

(a) the set of instructions must be *finite*, and

(b) each instruction must be precise with no ambiguity.

(c) the algorithm must be re-usable.

Feature (a) says that, in order to be able to solve a problem using an algorithm, there must not be an infinite list of instructions, lest we will never be able to get to a solution. This is different from saying that, given an input, the *actual computation* of an algorithm is finite. For example, given any positive integer, one can calculate its squared root. An algorithm exists to carry out this computation. However, unless the integer is a square, the actual computation will go on indefinitely, with a more accurate approximation of the actual value after each iteration. Feature (b) simply means that, given an input, if we were to give the corresponding algorithm to someone to carry out the computations to arrive at an output, the person (or machine) carrying out the steps should have no trouble in doing so. For example, if we were to give the algorithm above to a math student, he/she should have no trouble in following each of the instructions. However, if we were to hand the algorithm to someone who has never studied math before, we are in trouble. What does it mean for an integer to be a square? What does it mean for a number to divide another number? We would have to break down the instructions into even simpler ones, until there is no ambiguity in the solver's mind what they are. Finally, feature (c) means that if a particular input is given, then it does not matter when the computations are carried out, and we will always get the same output. For example, if $x^2 - 4x + 3$ has integral solutions, as determined by algorithm A, then we would not expect to find that $x^2 - 4x + 3$ has *no* integral solutions two days later again using A.

Remark. The last feature is sometimes relaxed so that "re-usable" means that the output is *as expected*. This means that an algorithm may include an instruction to, say, toss a coin, or to use a random number generator, in order to move forward with a computation. A common example of an algorithm involving a random device is the algorithm to draw a simple random sample from a given population. One computation results in a sample that is almost guaranteed to be different from the sample produced by a later computation. Nevertheless, if a certain sample size is asked for, that sample size will be constant from one computation to the next.

Algorithms and Computations

Closely associated with the notion of an algorithm is the notion of computation, mentioned earlier but not formally introduced. Given an input x, *computation* is the actual execution of the given x using instructions in some algorithm A. Each step in the computation is a *computation step*. Depending on the input, the computation either terminates or halts, after a finite number of computation steps, or goes on indefinitely. When the computation terminates, an output y is produced, and we say that the computation is *effective*, or that the output y can be *effectively computed* (with respect to the algorithm A). A function f is said to be A-*computable* if, for each input x in the domain of f, the output $f(x)$ can be effectively computed. Furthermore, f is *effectively computable* if there is an algorithm A such that f is A-computable. An *effective procedure* is another way of calling an algorithm. For example, addition of two integer is effectively computable. Given two integers a, b, there is an algorithm A such that, when a computation is carried out using A, the result $a + b$ will be produced.

Are there any functions that are not effectively computable? More generally, are there (mathematical) problems which can not be solved using algorithms? This is basically equivalent to asking: are there functions that are not effectively computable? Intuition tells us that there are. For example, if we want to generalize the procedure of determining the existence of integral solutions to a quadratic equations to a general polynomial equations of degree n, we would most likely not succeed. In fact, this is a special case of a more general problem known as Hilbert's 10th Problem:

> given a diophantine equation, can one determine if it has any integral solutions using an algorithm?

How does one prove this rigorously? Of course, to answer such a question, one needs to treat an "algorithm" itself as a mathematical object, since the informal heuristic description above is inadequate.

Abstractions of Algorithms

During the early and middle part of the 1900's, various proposals of what an algorithm is were put forth. These include include:

T Turing machine (Turing)

R partial recursive functions (Kleene)

L `http://planetmath.org/LambdaCalculus`λ-calculus (Church)

M Markov algorithm (Markov)

P Post system (Post)

S semi-Thue system (Thue)

U unlimited register machine, or URM (Shepherdson and Sturgis)

Some, like **T** and **U**, have more of a resemblance of the intuitive notion of an algorithm described earlier, while some, like **M**, **P** and **S**, are based on the realization that input and instructions are nothing but strings of symbols, and computation is nothing more than transforming one string into another string. Nevertheless, in each of the above proposals, it is possible to rigorously define the notion of "computability" informally described earlier. As it turns out, the class of computable functions according to each of the proposals turns out to be identical! In other words, Turing computability is equivalent to Markov computability, etc...

Remark. Although it was generally suspected that Hilbert's 10th Problem was false, it was not until 1970, that Yuri Matiyasevič proved the 10th Problem in the negative. This fact is now known as Matiyasevič's theorem.