Definizione di algoritmo e sue proprietà

Adattato da Donald E. Knuth. Fundamental Algorithms, volume 1 di The Art of Computer Programming. Addison-Wesley, 1968 e 1973.

Definition

The modern meaning for algorithm is quite similar to that of *recipe, process, method, technique, procedure, routine, rigmarole,* except that the word “algorithm” connotes something just a little different. Besides merely being a **finite set of rules that gives a sequence of operations for solving a specific type of problem**, an algorithm has five important features:

Properties

1) *Finiteness.* An algorithm must always terminate after a finite number of steps.

(A procedure that has all of the characteristics of an algorithm except that it possibly lacks finiteness may be called a *computational method*. An example of a nonterminating computational method is a *reactive process*, which continually interacts with its environment.)

2) *Definiteness.* Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case. The algorithms of this book will hopefully meet this criterion, but they are specified in the English language, so there is a possibility that the reader might not understand exactly what the author intended. To get around this difficulty, formally defined *programming languages* or *computer languages* are designed for specifying algorithms, in which every statement has a very definite meaning. An expression of a computational method in a computer language is called a *program*.

3) *Input.* An algorithm has zero or more *inputs*: quantities that are given to it initially before the algorithm begins, or dynamically as the algorithm runs. These inputs are taken from specified sets of objects.

4) *Output.* An algorithm has one or more *outputs*: quantities that have a specified relation to the inputs.

5) *Effectiveness.* An algorithm is also generally expected to be *effective*, in the sense that its operations must all be sufficiently basic that they can in principle be done exactly and in a finite length of time by someone using pencil and paper.

Remarks

Let us try to compare the concept of an algorithm with that of a cookbook recipe. A recipe presumably has the qualities of finiteness (although it is said that a watched pot never boils), input (eggs, flour, etc.), and output (TV dinner, etc.), but it notoriously lacks definiteness. There are frequent cases in which a cook’s instructions are indefinite: “Add a dash of salt.” A “dash” is defined to be “less than ⅛ teaspoon,” and salt is perhaps well enough defined; but where should the salt be added—on top? on the side? Instructions like

“toss lightly until mixture is crumbly” or “warm cognac in small saucepan” are quite adequate as explanations to a trained chef, but an algorithm must be specified to such a degree that even a computer can follow the directions.

We should remark that the finiteness restriction is not really strong enough for practical use. A useful algorithm should require not only a finite number of steps, but a *very* finite number, a reasonable number. In practice we not only want algorithms, we want algorithms that are *good* in some loosely defined aesthetic sense. One criterion of goodness is the length of time taken to perform the algorithm; this can be expressed in terms of the number of times each step is executed. Other criteria are the adaptability of the algorithm to different kinds of computers, its simplicity and elegance, etc.

We often are faced with several algorithms for the same problem, and we must decide which is best. This leads us to the extremely interesting and all-important field of *algorithmic analysis* : Given an algorithm, we want to determine its performance characteristics.