Algorithms

1. Introduction to Algorithms

R K Shyamasundar

R K Shyamasundar is a Professor of Computer Science at TIFR, Bombay who has done extensive research in various foundation areas of computer science.

In this introductory article the concept of algorithm which forms the foundation of computer science is defined. A diagrammatic form of describing algorithms known as flowcharts is introduced and used to express some elementary algorithms.

What is an Algorithm?

The concept of an *algorithm* constitutes the foundation for information processing. It is not only a familiar concept to mathematicians but also forms the foundation of computer science. Two striking features of algorithms are: 1) an algorithm is meant to be *executed* and 2) objects underlying an algorithm have an *associated meaning*. In the first few articles, our exploration is confined to the first feature.

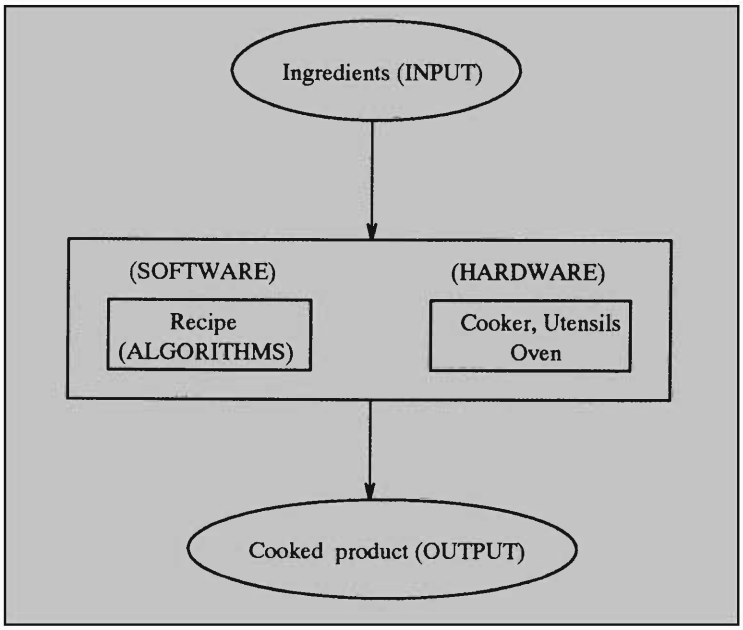

Figure 1 A computational process is similar to the process of cooking.

About the series on "Algorithms"

Algorithms are central to the applications of computers and were proposed well before the advent of computers. A well-known algorithm by Euclid gave a step by step method to find the greatest common divisor of two numbers. The study of algorithms as a subject became important with the advent of computers. The science of "algorithms" deals not only with methods of developing algorithms for solving a variety of problems but also with fundamental questions such as: "are there problems for which algorithms do not exist?", "even if algorithms exist will they lead to solutions within a reasonable time?", "are there systematic methods of proving whether an algorithm, to solve a problem, is correct?".

In this series of articles a variety of topics will be discussed. After discussing some well-known algorithms in depth, algorithmic methods which can be systematically applied to solve problems, methods of proving correctness of algorithms and finding out the complexity of algorithms will be discussed. Some interesting theoretical questions regarding computability which students find difficult to understand will be explained.

Computer scientists have largely devoted their attention to non-numeric algorithms. When it comes to applications of computers in science and engineering, numerical algorithms, namely, those related to numerical calculus are as important. We will thus discuss some interesting issues in evolving numerical algorithms. As the topics discussed are diverse many individuals with expertise in the topics they write about will author this series.

V Rajaraman

Consider the process of cooking rice. Cooking is the process carried out by a cook using the ingredients with the help of utensils, cookers, oven and a *recipe*. The ingredients are the *inputs* to the process, the cooked rice is its output, and the *recipe* is its *algorithm*. That is, an algorithm prescribes the activities that constitute a process. The recipes (algorithms) are grouped under the term *software*, whereas the utensils, cookers, and oven are grouped under *hardware*. The process of cooking is depicted in *Figure 1*.

Let us look at a recipe for cooking rice. It consists of the following simple activities:

1 Put 1 cup of rice in the vessel of an electric rice-cooker.
2 Put 2 cups of water in the cooker vessel.
3 Close the lid of the cooker and switch it on.
4 Wait until the indicator light of the rice-cooker turns off.

Two striking features of algorithms are: 1) an algorithm is meant to be *executed* and 2) objects underlying an algorithm have an *associated meaning*.

History

The word 'algorithm' itself is interesting. At first glance, it appears to have some relation to the familiar 'logarithm' (by some permutation of the first four characters). However, it stems from the name of the author of a famous Arabic textbook (its original Arabic text is lost; a Russian translation of a Latin manuscript exists), Abu Ja'far Mohammed ibn Mûsâ-al-Khowârizmî (A D 825) who first suggested a mechanical method for adding two numbers represented in the Hindu positional number system. The name transcribed in Latin became *algorismus* from which *algorithm* was but a simple transformation. Of course, the first non-trivial algorithm ever was devised by the great Greek mathematician Euclid (between 400 and 300 B C) for finding the greatest common divisor (gcd) of two positive numbers. The word 'algorithm' was most often associated with this algorithm till 1950. It may however be pointed out that several non-trivial algorithms such as synthetic (polynomial) division have been found in *Vedic Mathematics* which are dated much before Euclid's algorithm.

From activities 1-3, we can observe that:

- Each activity is a command.
- Each activity is finite and unambiguous (assuming that a cup and an electric cooker are given).
- Activities 1 to 3 are done in sequence. That is, activity 1 is followed by activity 2; it is only then that activity 3 is performed.

It can be easily seen that activity 4 is a "test" rather than a command. Further, we cannot conclude that the light indeed turns off - that is, whether the activity terminates. However, assuming the cooker to be non-faulty and does cook (even if it is slow), we know that activity 4 eventually terminates. In the same way, one of the most important aspects of an algorithm is that it should always terminate after a finite number of steps. In short, we can say that an algorithm is a finite set of rules that prescribes a sequence of operations for solving a specific problem. Though this is not a formal definition, it captures the concepts underlying algorithms.

A *programming language* is used to describe an algorithm for execution on a computer. An algorithm expressed using a programming language is called a *program.*

As already emphasized, we would like our algorithms to *execute* on a processor. Hence, it can be easily seen that the notation used for

describing the algorithm has to be precise and unambiguous. From the above discussions regarding the recipe for cooking rice, we can see that the notation (essentially the language) used for describing algorithms should have the following properties:

1 All the basic operations used to describe the algorithmic steps must be capable of being performed mechanically (without using any intuition) in a finite amount of time.
2 Each step must be defined unambiguously.
3 The language must be general enough for describing different kinds of algorithms that can operate on a variety of inputs.
4 It should be capable of describing steps which provide input and extract outputs.

To reiterate, the algorithms must be written in an unambiguous and a formal way. A *programming language* is used to describe an algorithm for execution on a computer. An algorithm expressed using a programming language is called a *program.*

A Flowchart Language

Let us now look at a visual and diagrammatic form of describing algorithms. The essential components of a visual diagrammatic representation referred to as *flowcharts* will be described now:

Basic Commands

The basic commands are denoted by a rectangular box with an inscription of the operations to be performed. For example, *Figures 2(a)-(c)* correspond to setting the value of i to 1, setting the value of c to the sum of values of a and b, and setting the value of x to *sin(theta)* respectively. *Figures 2(d) and 2(e)* denote commands to read in values into A,B,C and printing the values of x, y, and z respectively. Observe that *parallelograms* are used to represent Read and Print commands. Further, it is useful to denote a command which does nothing by an empty box or a box with an inscription *nothing.*

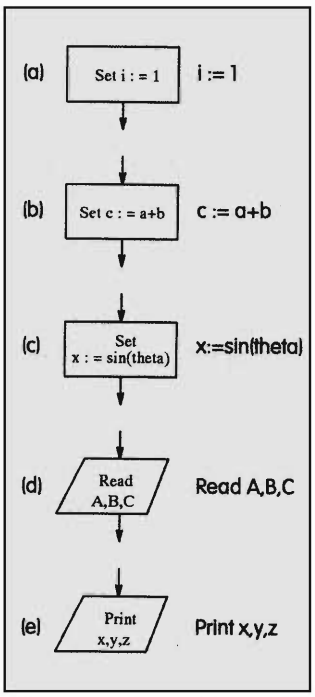

Figure 2 Symbols used in flowchart language to represent Assignment, Read and Print commands.

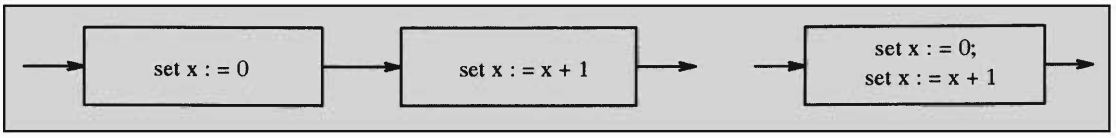

Figure 3 A sequential composition allows building a larger component using smaller components.

Another point to be observed is that one arrow leads into the box and another arrows goes out of the box. The arrow at the top of the box can be visualized as a way of asking the processor to do the prescribed action, and the arrow at the bottom is a way of signalling the environment that the prescribed action is done.

These are referred to as the command actions. In textual form the commands would merely be those inscribed within the box; these are shown on the right half of Figure 2. The operator ":=" is referred to as the "assignment" operator and is interpreted as:

> "Assign the value of the *expression* (operand) on the right-side of the operator, to the *variable* (operand) on the left-side of the operator".

Sequential Composition

The operation of 'sequencing' permits concatenation of boxes; that is, the operation provides a way of 'composing' or 'building' a program (system) from smaller components.

For example, the left-part of *Figure 3* corresponds to first setting the 'variable' x to zero and then assigning to x the value obtained by adding 1 to the old value of x (i.e. increment x by 1). For brevity, we denote the same by the box shown on the right side of Figure 3. Here, the order is implicitly indicated by the top-down order. Sometimes, a semicolon is used to separate the commands in order to avoid confusion.

It is important to note that $x := 0;\ x := x + 1$ is not the same as $x := x + 1;\ x := 0$. In the former, after the execution is complete, x will have the value 1 and in the latter, x will have the value 0. Thus, the order of the commands is important.

Test

'Test' is denoted by a *diamond box* having one input arrow and two output arrows as shown in *Figure 4*. Note that if $x > 100$ then the control goes to the branch labelled "YES", and if $x \leq 100$ (i.e., negation of the condition $x > 100$), then the control goes to the branch labelled "NO".

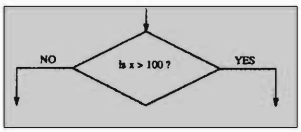

Figure 4 A diamond shaped box is used to represent a test.

A construct called "if-then-else" is shown in *Figure 5*. Observe that if the 'cond' test is "true" the "YES" path is taken and if it is "false" the "NO" path is taken. In the "YES" path statement 1 is executed and in the "NO" path statement 2 is executed.The flowchart can be thought of as a 'block' as it occurs often in algorithms. It can be concisely expressed using an algorithmic language as shown below:

if cond *then* Statement 1
else Statement 2
endif

This is a single statement.

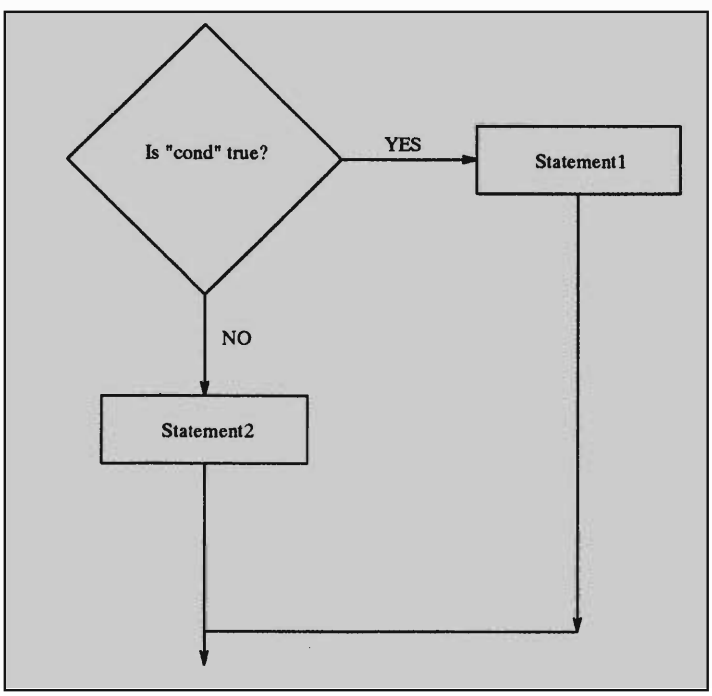

Figure 5 A flowchart illustrating a commonly used algorithmic construct known as if-then-else.

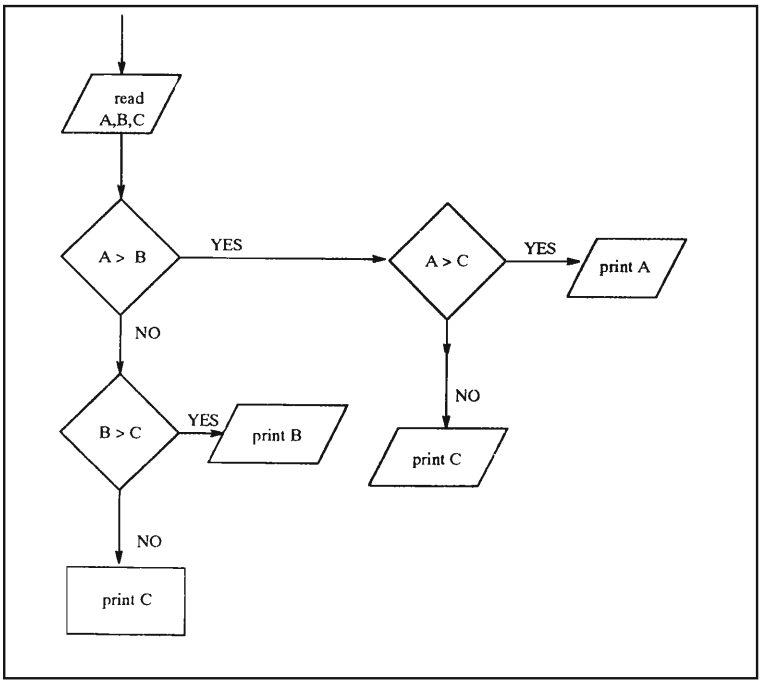

Figure 6 A flowchart which describes an algorithm to find the largest of three numbers.

Suggested Reading

D E Knuth. Art of Computer Programming — Volume 1. Addison-Wesley Publishing Co. 1972.
This volume, which discusses fundamental algorithms, is an advanced and authoritative book on the subject.

E W Dijkstra. A Short Introduction to the Art of Programming. Computer Society of India. 1977.
This is a lucid article written by a pioneer whose research work formed the basis for transforming programming from an art to science.

G Polya. How to Solve It. Princeton University Press. 1973.
Even though this book does not deal with computational algorithms it is recommended reading as it provides insights into the process of problem solving.

R G Dromey. How to Solve it by Computer. Prentice-Hall of India, New Delhi. 1990.
Following Polya's philosophy, this book explains approaches to problem solving algorithmically.

Note that *Statement 1* or *Statement 2* can themselves be composed of many basic commands. Whenever the command is not primitive (made of at least two basic commands), we often refer to the statement as a "block".

Note also that apart from the basic commands, we use a beginning-keyword (such as "*if*") and corresponding matching end-keyword (eg., "*endif*"). This would aid readability of the program (similar to open and close parenthesis) and also enables us to use blocks of statements between the beginning-and end-keywords. For instance, in the "*if-then-else*" construct, the keyword, "*endif*" corresponds to the end for the whole construct; further the keyword "*then*" closes with "*else*", the "*else*" closes with "*endif*". In the programs shown, we have also used "indentation" to highlight the blocks (this aids in reading programs).

Example 1: Given three distinct numbers A,B,C find the largest among them.

In the flowchart shown in *Figure 6*, one can trace the paths from the start (from the beginning of the read-block) to the end (print-

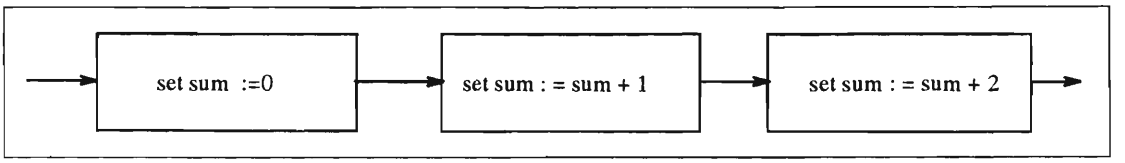

Figure 7 A flowchart to sum the first two natural numbers.

block) and can observe that the path is finite. The path passing through the edges "YES, YES" satisfies the condition $A > B$ and $A > C$ and hence we can conclude that A is the largest. The second path passing through the edges "YES, NO" satisfies the conditions $A > B$, and $A \leq C$ respectively and we can conclude that C is the largest. Similar assertions follow along the same lines on other branches.

The following observations can be made on this algorithm:

1 The algorithm terminates as each path is finite and each block can take only a finite amount of time.
2 The number of boxes that get executed in each path can be fixed before accepting (reading) the values of A, B and C. In other words, the number of boxes in each path is fixed.

However, it is not always possible to satisfy property (2) as illustrated by the following example.

Example 2: Summing N Numbers: The problem is to find the sum of the first N natural numbers (i.e., the set {1,2,3,...N}) for any given N.

Let us first write a flowchart for the case $N = 2$. Next the required algorithm can be derived from this flowchart as shown in *Figure 7*.

Thus, if we have to sum the first 100 numbers, we need 100 + 1 boxes. This is not a very elegant way to solve the problem. Then the natural question is: Is it possible to obtain a flow chart which has the same number of boxes, irrespective of the value of N? The answer is "yes", and the solution lies in the construct wherein the number of times a set of commands gets executed depends on the values of certain variables. We will discuss this in the next article in this series.

Address for correspondence
R K Shyamasundar
Computer Science Group,
Tata Institute of
Fundamental Research,
Homi Bhabha Road,
Bombay 400 005, India.

The material in this series of articles is based on the works of a large number of researchers - too numerous to cite individually. I thank N Raja and Basant Rajan for a critical reading of the manuscript.

R K Shyamasundar