QA9.58.L57

ALGORITHM CREATION FROM THE VIEWPOINTS OF BOTH THE MATHEMATICIAN AND THE COMPUTER SCIENTIST

A Thesis
Presented to the Graduate Faculty
of Western Connecticut State College

by

Louis Andrew Lisko Jr.

In Partial Fulfillment
of the Requirements for the Degree
Master of Arts

[illegible] May 20, 1976
Thesis Advisor

[illegible]
Graduate Studies

CHAPTER I

INTRODUCTION

If a mathematician were asked to describe briefly what he considered the primary duty of his profession, he would address himself to those mathematical objects with which he deals--objects such as integers, continuous functions, or projections. He would surely admit that one of his major efforts involves establishing the validity, or proving the falseness, of various theorems concerning these mathematical objects. Another task he probably would mention is problem-solving through the formulating of algorithms or effective computational procedures.

In part, the mathematician is attempting to provide a means of obtaining answers to entire classes of problems by literally listing instructions which serve as mechanical procedures for solving problems. When executing these instructions set forth by the mathematician, there is no need for innovative thought. A machine can be constructed to carry them out, or a digital computer can execute them by means of translating the instructions into a computer program.

Suppose a problem were proposed as follows: Find the product of x and y, where x and y are members of the set of Real numbers. Any textbook on elementary arithmetic will give instructions that can be used. The procedure found is mechanical; it can be executed by someone who is merely following elementary steps or by someone operating an elementary computing device. Since 1975 an entire new electronics market has opened with the introduction of inexpensive electronic calculators which solve this and similar problems. Most problems are not as straightforward in solution as this, however. The majority of problems to be solved on computers are much more intricate. One successful approach to finding solutions for these is to separate them into a number of more simple problems. Then, assuming there is an algorithm to solve each of these sub-problems, an algorithm is sought to solve the original problem. In other words, a solution to a complex problem has been accomplished by obtaining solutions to each of its elementary sub-problems.

The computer scientist deals with assignments in a manner very much like those described above. He utilizes computers, and studies areas of application for these computing devices and how to actually produce solutions for problems on them. This paper is concerned with the algorithmic solutions to problems facing both computer scientists and mathematicians. The picture of computer science presented in this paper is not intended to imply this field is narrow in scope. The realm of computer science is far-reaching, and encompasses a vast number of non-mathematical applications, such as industrial process control, computer aided instruction, cybernetic theory, and message switching.

Whether speaking of a mathematical application or a message switching application, or planning the solution of a simple or complex problem, one thing is certain--a computer program must be developed in order to execute it on a computer. There are many objectives computer programming serves. Most basically, it is the bridge between man, whose goal is the construction of compound objects from simpler elements by combining elements according to the rules of some "algebra", and the computer. Just as the earlier objective in a solution to a complex problem was to solve each sub-problem, it can be said that programming is the discovery of methods for combining elements into compound objects representing useful processes. (13, p. 29) This objective is clear when considering the computer language compiler which translates the original human program statements (elementary objects) into the machine executable code (useful processes). Conversely, programming also serves to find "hidden algebras" in terms of which compound objects representing useful processes may be built. That is, programming is simplification, and, like mathematics, is a search for lucky simplifications. It is worth emphasizing that the discovery of those simplifications is the essential goal of experimental, as distinct from applied, programming. (13, p. 29)

A programmer might then briefly state his task as that of making a defined algorithm suitable for processing on the computer. Yet a mathematician might say, and rightly so, that he has already laid out the same algorithm in terms which are very often much more concise, even though his precision in the statement might vary considerably from a colleague's. It appears then that mathematics, although being more polished or refined in stating algorithms, lacks the exactness necessary for computer programming. Programming makes the specified algorithm as effective or functional as possible, while mathematics allows more leeway, and thus a lesser degree of optimization. If programming applied these same tolerances, one might be led to believe that computers with unlimited resources would be necessary in order to execute such programs. However, the necessity for such careful coding is often cited as the main obstacle to the development of applications on the computer. Mathematicians deal on the one hand with searches of very large sets--perhaps even infinite in scope; but programmers must be concerned with devising equivalent procedures that reduce the margin of choices. The conflict between mathematics and computer science is apparent. After having compared some aspects of the two, it may seem that computer programming is difficult. The fact is, it is not difficult, but rather, programming requires a high degree of discipline in its users.

In the chapters that follow, algorithms will be considered from the conceptual stage (where the mathematician desires to create a valid statement of instructions), to the implementation stage (where the computer scientist wishes to take the mathematician's result and translate it into a form that can be "understood" by a computer). A look at the non-computational area of theorem proving will be taken; here, the emphasis will be on the necessary aspects needed to create the algorithm, and also the features embodied within the field of recursive function theory. This will lead to a discussion of the languages that offer the mathematician a direct tool in which to create an algorithm, and that afford the computer scientist a greater flexibility so he can communicate with mathematicians who wish to use the computer as an expedient discovery tool as well as a problem solver, and not as an end in itself.