Algorithm

An *algorithm* is a finite sequence of precise instructions for performing a computation or for solving a problem.

ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

procedure $max(a_1, a_2, \ldots, a_n$: integers)
$max := a_1$
for $i := 2$ **to** n
    if $max < a_i$ **then** $max := a_i$
return max {max is the largest element}

PROPERTIES OF ALGORITHMS There are several properties that algorithms generally share. They are useful to keep in mind when algorithms are described. These properties are:

- *Input.* An algorithm has input values from a specified set.
- *Output.* From each set of input values an algorithm produces output values from a specified set. The output values are the solution to the problem.
- *Definiteness.* The steps of an algorithm must be defined precisely.
- *Correctness.* An algorithm should produce the correct output values for each set of input values.
- *Finiteness.* An algorithm should produce the desired output after a finite (but perhaps large) number of steps for any input in the set.
- *Effectiveness.* It must be possible to perform each step of an algorithm exactly and in a finite amount of time.
- *Generality.* The procedure should be applicable for all problems of the desired form, not just for a particular set of input values.

ALGORITHM 2 The Linear Search Algorithm.

procedure *linear search*(x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ **then** *location* $:= i$
else *location* $:= 0$
return *location*{*location* is the subscript of the term that equals x, or is 0 if x is not found}

Pseudocode

Procedure Statements

The pseudocode for an algorithm begins with a **procedure** statement that gives the name of an algorithm, lists the input variables, and describes what kind of variable each input is. For instance, the statement

procedure *maximum*(L: list of integers)

is the first statement in the pseudocode description of the algorithm, which we have named *maximum*, that finds the maximum of a list L of integers.

Assignments and Other Types of Statements

An assignment statement is used to assign values to variables. In an assignment statement the left-hand side is the name of the variable and the right-hand side is an expression that involves constants, variables that have been assigned values, or functions defined by procedures. The right-hand side may contain any of the usual arithmetic operations. However, in the pseudocode in this book it may include any well-defined operation, even if this operation can be carried out only by using a large number of statements in an actual programming language.

The symbol $:=$ is used for assignments. Thus, an assignment statement has the form

variable $:=$ *expression*

For example, the statement

$max := a$

assigns the value of a to the variable max. A statement such as

$x :=$ largest integer in the list L

can also be used. This sets x equal to the largest integer in the list L. To translate this statement into an actual programming language would require more than one statement. Also, the instruction

interchange a and b

can be used to interchange a and b. We could also express this one statement with several assignment statements (see Exercise 2), but for simplicity, we will often prefer this abbreviated form of pseudocode.

Comments

In the pseudocode in this book, statements enclosed in curly braces are not executed. Such statements serve as comments or reminders that help explain how the procedure works. For instance, the statement

{x is the largest element in L}

can be used to remind the reader that at that point in the procedure the variable x equals the largest element in the list L.

Conditional Constructions

The simplest form of the conditional construction that we will use is

if condition **then** statement

or

if condition **then**
    block of statements

Here, the condition is checked, and if it is true, then the statement or block of statements given is carried out. In particular, the pseudocode

```
if condition then
    statement 1
    statement 2
    statement 3
    .
    .
    .
    statement n
```

tells us that the statements in the block are executed sequentially if the condition is true.

For example, in Algorithm 1 in Section 3.1, which finds the maximum of a set of integers, we use a conditional statement to check whether $max < a_i$ for each variable; if it is, we assign the value of a_i to max.

Often, we require the use of a more general type of construction. This is used when we wish to do one thing when the indicated condition is true, but another when it is false. We use the construction

```
if condition then statement 1
else statement 2
```

Note that either one or both of statement 1 and statement 2 can be replaced with a block of statements.

Sometimes, we require the use of an even more general form of a conditional. The general form of the conditional construction that we will use is

```
if condition 1 then statement 1
else if condition 2 then statement 2
else if condition 3 then statement 3
    .
    .
    .
else if condition n then statement n
else statement n + 1
```

When this construction is used, if condition 1 is true, then statement 1 is carried out, and the program exits this construction. In addition, if condition 1 is false, the program checks whether condition 2 is true; if it is, statement 2 is carried out, and so on. Thus, if none of the first $n - 1$ conditions hold, but condition n does, statement n is carried out. Finally, if none of condition 1, condition 2, condition 3, . . . , condition n is true, then statement $n + 1$ is executed. Note that any of the $n + 1$ statements can be replaced by a block of statements.

Loop Constructions

There are two types of loop construction in the pseudocode in this book. The first is the "for" construction, which has the form

for *variable* := *initial value* **to** *final value*
    statement

or

for *variable* := *initial value* **to** *final value*
    block of statements

where *initial value* and *final value* are integers. Here, at the start of the loop, *variable* is assigned *initial value* if *initial value* is less than or equal to *final value*, and the statements at the end of this construction are carried out with this value of *variable*. Then *variable* is increased by one, and the statement, or the statements in the block, are carried out with this new value of *variable*. This is repeated until *variable* reaches *final value*. After the instructions are carried out with *variable* equal to *final value*, the algorithm proceeds to the next statement. When *initial value* exceeds *final value*, none of the statements in the loop is executed.

We can use the "for" loop construction to find the sum of the positive integers from 1 to n with the following pseudocode.

$sum := 0$
for $i := 1$ **to** n
    $sum := sum + i$

Also, the more general "for" statement, of the form

for all elements with a certain property

is used in this text. This means that the statement or block of statements that follow are carried out successively for the elements with the given property.

The second type of loop construction that we will use is the "while" construction. This has the form

while condition
    statement

or

while condition
    block of statements

When this construction is used, the condition given is checked, and if it is true, the statements that follow are carried out, which may change the values of the variables that are part of the condition.

If the condition is still true after these instructions have been carried out, the instructions are carried out again. This is repeated until the condition becomes false. As an example, we can find the sum of the integers from 1 to n using the following block of pseudocode including a "while" construction.

```
sum := 0
while n > 0
    sum := sum + n
    n := n − 1
```

Note that any "for" construction can be turned into a "while" construction (see Exercise 3). However, it is often easier to understand the "for" construction. So, when it makes sense, we will use the "for" construction in preference to the corresponding "while" construction.

Loops within Loops

Loops or conditional statements are often used within other loops or conditional statements. In the pseudocode used in this book, we use successive levels of indentation to indicate nested loops, which are loops within loops, and which blocks of commands correspond to which loops.

Using Procedures in Other Procedures

We can use a procedure from within another procedure (or within itself in a recursive program) simply by writing the name of this procedure followed by the inputs to this procedure. For instance,

```
max(L)
```

will carry out the procedure *max* with the input list L. After all the steps of this procedure have been carried out, execution carries on with the next statement in the procedure.

Return Statements

We use a **return** statement to show where a procedure produces output. A return statement of the form

```
return x
```

produces the current value of x as output. The output x can involve the value of one or more functions, including the same function under evaluation, but at a smaller value. For instance, the statement

```
return f(n − 1)
```

is used to call the algorithm with input of $n - 1$. This means that the algorithm is run again with input equal to $n - 1$.