4

Algorithms

Definition of an Algorithm

An algorithm is the abstract version of a computer program. In essence a computer program is the codification of an algorithm, and every algorithm can be coded as a computer program. An algorithm can be defined as follows:

- 1. An algorithm has precise initialization criteria
- 2. At every point in an algorithm there are precise unambiguous instructions as to what to do next.
- 3. The termination criteria are precise and unambiguous.
- 4. The algorithm must terminate.

It is only rule four that is at all controversial. Consider for example a traffic light; it turns consecutively green, yellow, then red, but never terminates. Is it following an algorithm? There are two solutions: one is to waive rule four in certain situations. The other solution is to declare that the traffic light algorithm only goes through the green—yellow—red sequence once. We then run that sequence over and over without end. The problem with the second solution is that we seem to be in effect running a proper finite algorithm without end, that is within an unending algorithm. However, in general, we want algorithms to terminate; usually when a program has an endless loop, we regard that loop as a *bug*.

The Collatz Graph

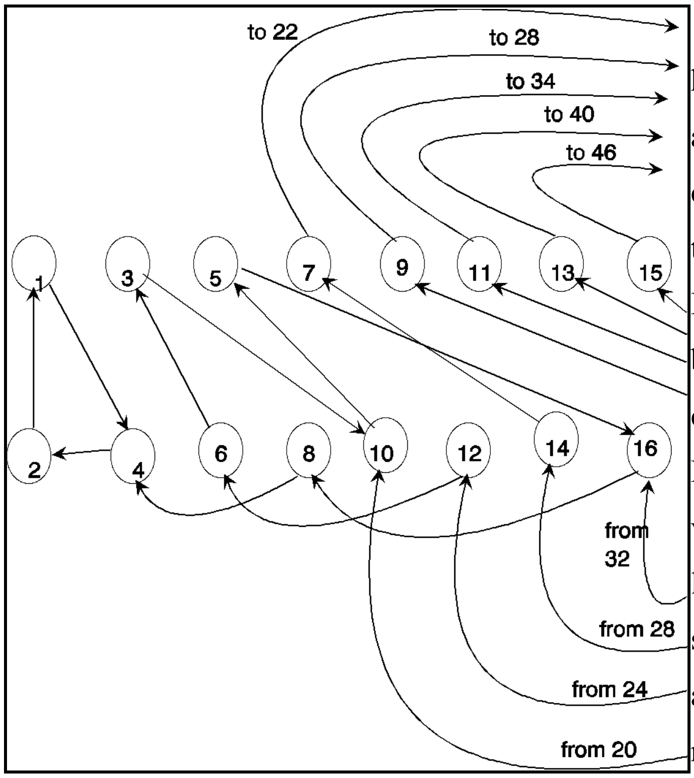

Figure 1 The Collatz Graph

There is no law that a graph must have either a finite number of vertices or a finite number of arcs.[1] Whereas we cannot completely draw such a graph, they are often satisfying to contemplate. Let us consider a graph (that happens to be a function) in which there is a node corresponding to each positive integer. Hence we have an infinite number of vertices, but there will be only one arc leaving each vertex. If the integer corresponding to a node is even, we have an arc leaving that node and going to the node whose integer is one-half the first integer. If the integer corresponding to a node is odd, we have an arc leaving that node and going to the node whose integer is three times the first integer plus one. This graph is the *Collatz graph*.

As an example, suppose we start on vertex 53. Since 53 is odd, we go to $3 \cdot 53 + 1 = 160$. Since 160 is even we proceed to $160/2 = 80$. Again 80 is even, so we proceed to 40 then to 20 to 10 to 5. 5 is odd so we go up to 16, then to 8, to 4, to 2 and to 1. From 1 we repeat the sequence 1—4—2—1—4—2—1....

[1]When I say that there is no law against infinite graphs, or graphs with loops, or permutation graphs or whatever, I am speaking of *federal* laws. Here in Alabama these things are misdemeanors.

The Collatz Algorithm

The Collatz graph is a function because there is only one arc leaving each node. If we choose a node (i.e. a positive integer) at random, we have a strictly determined path that goes on without ending. However, if the path ever reaches 1, then it stays on a circuit: 1—4—2—1—4—2—1.... Therefore we could ask the question: starting at a given integer, n, how long does it take to reach 1? Having asked the question, lets write an algorithm to answer it. Such an algorithm might be written:

The Collatz Algorithm

```
1.    Input a positive integer n.  Set counter ← 1.
2.    Print n, counter.
3.    If n = 1 then stop.  Algorithm is finished.
4.    If n is even then n ← n/2 else n ← 3·n + 1.
5.    counter ← counter + 1.
6.    Goto 2.
```

The symbol ← is the *assignment* operator. For example, $i \leftarrow i + 1$ means *replace i by i + 1.* In Basic this is written i = i + 1. In Pascal it is written i := i + 1. The arrow is used instead of the equal sign because the assignment i = i + 1 is not and cannot be an equality.

The term *counter* keeps track of the number of arcs transversed starting at the node (integer) n. Step 2 prints the current vertex and the number of arcs transversed. The last printed line should have n = 1 and *counter* equal to the total number of arcs transversed. A more proper expression of this algorithm (that is without the *Goto* statement) might be:

The Collatz Algorithm Again

```
1.    Input a positive integer n.  Set counter ← 1.
2.    Repeat
          Print n, counter.
          If n is even then n ← n/2 else n ← 3·n + 1.
          counter ← counter + 1.
3.    Until n = 1.
4.    Stop.
```

Is the Collatz Algorithm Really an Algorithm?

The Collatz procedure clearly satisfies the first three criteria of an algorithm.[1] The input is clearly defined. The termination criteria is clearly defined. At every step there is no ambiguity about what to do next. The only condition that remains to be verified is that the procedure always terminates. We know from the above example that the procedure terminates if we start with n = 53. However, the procedure states that any positive integer is valid input. Therefore, to be an algorithm it must terminate for any positive integer input.

The Collatz algorithm has been tested for billions of input. It has been shown that there can be at most a few exceptional inputs that do not terminate. But it has not been shown that any such inputs do or don't exist. Hence, whether the Collatz algorithm is truly an algorithm is an open question.

☐ **Exercise 1** Suppose that you found an input N such that the Collatz algorithm did not terminate, how would you know? (This is a thought experiment close to theoretical computer science. Don't lose any sleep over it.)

☐ **Exercise 2** Write the Collatz algorithm in computer code. Use a long integer format so that you can deal with large integers. Use the mod function to format

[1]A *procedure* is much like an algorithm. I use the word *procedure* now because the question is raised whether the Collatz algorithm is actually an algorithm.

your output and so that you do not put a single number per line. It is not necessary to write the iteration number as in the algorithm above (where the iteration is given by *counter*) but simply to keep track of the iterations and to output the final value.

1. You would **not** necessarily know by running the algorithm. Since, it would never terminate, you would not know whether it was a non-terminating case or if you simply had not reached the termination point yet. You would know you had a counter-example if the sequence repeated an integer (without hitting one). However, monitoring the algorithm for a repeated integer might be difficult.

2.
```
n:  long integer
i:  short integer
Input n
If n ≤ 0 Then "Error":    Quit.
i ← 0
Repeat
        Begin
        Print n
        If n = 1 Then stop
        Move to next column           {This is pseudo-code!}
        If (n mod 2 = 0) Then n ← n/2          {test whether n is even}
                  Else n ← 3·n + 1
        i ← (i + 1) mod 8                      {if we want 8 columns}
        If i = 0 Then new line
        End
```