AN ADDITION ALGORITHM FOR GREATEST COMMON DIVISOR

D. E. DAYKIN
University of Reading, Reading, England

ABSTRACT

An elementary algorithm is presented for finding the greatest common divisor of two numbers. It is trivial to programme and fast, even for large numbers. Only addition is used, and the only storage space needed is enough to hold the two numbers.

About three years ago, I discovered an algorithm which K. Y. Choong, C. R. Rathbone and I used to obtain the first 20,000 partial quotients of the continued fraction of π. I here show how an adaptation of the algorithm may be used to find the greatest common divisor (g.c.d.) of any two positive integers. The complete process when the numbers are 1168 and 2847 is:

N	8832	= 10000 - 1168
Z	2847	= 2847
Z	1679	
Z	0511	
N	9343	
N	9854	
Z	0365	
Z	0219	
Z	0073	= g.c.d. = 73
N	9927	
	0000	

I will now describe the general process. Let P, Q be the two given positive integers, and suppose that k is the number of digits in the larger of P and Q. From a finite sequence of numbers, each of not more than k digits, according to the rules:

(a) The first two numbers are 10^k - P and Q. Of these 10^k - P is an N number and Q is a Z number.

(b) At each subsequent stage, the next number is the last N number plus the last Z number. Carries beyond k digits are ignored. If there is no such carry, this next number is an N number; otherwise, it is a Z number.

(c) Stop when the next number would be zero.

Than the last Z number is the g. c. d. of P and Q.

We start with $10^k - P$ and it might be argued that this requires a subtraction. However, we can obtain $10^k - P$ by applying the transformation $0 \to 9,\ 1 \to 8, \cdots, 9 \to 0$ to the digits of P, and then adding 1 to P. Hence, I am justified in saying that the algorithm only uses addition.

It will be noticed that, as we move down the sequence, the N numbers begin with more and more nines, while the Z numbers begin with more and more zeros. Hence the designations N and Z. It is not necessary to evaluate the next number in order to determine whether it is an N or a Z. Suppose the last N and Z numbers, respectively, are

$$\begin{array}{ll} N & n_k n_{k-1} \cdots n_1 \\ Z & z_k z_{k-1} \cdots z_1 \ . \end{array}$$

Then we look for the largest integer j such that the sum $n_j + z_j$ of the j^{th} digits is not 9. If there is no such j then the g. c. d. is 1. If there is such a j, and $n_j + z_j < 9$, we will get no carry and so, by the definition in rule (b), the next number will be an N number. Moreover, the addition is worked out step-by-step from the right, and so we can over-write the digits of the last N number, step-by-step, with the digits of the new N number. Similarly, if $n_j + n_k > 9$, the next number is written over the last Z number. Hence, the space for the storage of the 2k digits of P and Q is sufficient for the complete calculation. Once we have $n_k = p$ and $z_k = 0$ we can ignore these digits, and similarly for n_{k-1}, z_{k-1}, etc. Thus the amount of work required in the additions steadily diminishes, and this is indicated by the dotted line in the example. The simplest flow diagram is also shown.

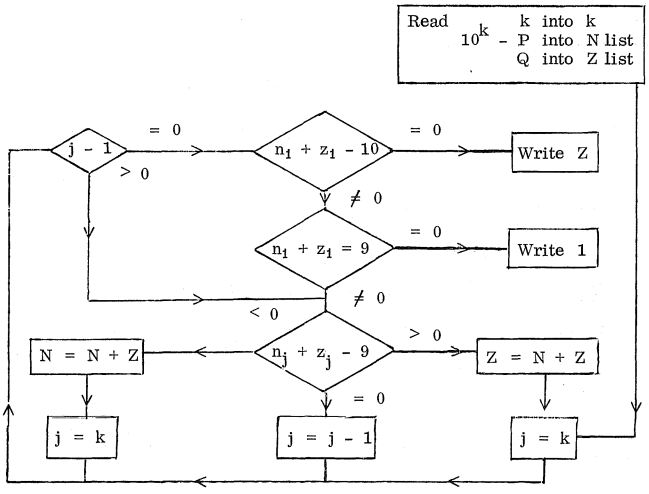
I will now prove that the algorithm does produce the g. c. d. At each stage, we have $N = 10^k - p$ with $0 < p < 10^k$ and $Z = q$ with $0 < q < 10^k$ and we want the g.c.d. (p,q). By rule (a), this is certainly the case initially.

Case 1. Suppose $N + Z$ would give no carry; that is, $10^k - p + q \leq 10^k - 1$ or $q < p$. Then the next number will be $10^k - (p - q)$. It will be an N number bigger than N but less than 10^k. Since g. c. d. $(p - q, q)$ = g. c. d. (p, q) the next stage will be of the correct form.

Case 2. Suppose $N + Z$ would give a carry; that is, $10^k - p + q \geq 10^k$ or $q \geq p$. This carry is ignored, so the next number will be $(10^k - p) + (q) - 10^k = q - p$, and $0 \leq q - p < q$. Again the next stage will be of the correct form. Since the size of Z foes down at each stage, we will reach a stage where the next Z would be zero; then $p = q$ so that the g.c.d. $(p, q) = q$ in Z.

I have described all this in terms of arithmetic to the base 10. Clearly, the algorithm works with any base, and in particular in binary. Sometimes it is convenient to work to the base 10 but with several consecutive digits of the numbers in a computer word. With a little more programming effort, one can speed the process up as follows.

Let r and s be the largest integers such that $n_r \neq 9$ and $z_s \neq 0$ respectively, and suppose that $r > s$. Then, instead of replacing N by $N + Z$, it saves work to replace N by $N + 10^{r-s-1}Z$. For this operation, we again only need addition with the appropriate shift. In most cases, we are in fact able to improve this to replacing N by $N + 10^{r-s}Z$, it is not difficult to distinguish the exceptional cases. As one would expect, the corresponding situation obtains if $r < s$.



Flow Diagram for g. c. d.