Algorithm and Bound for the Greatest Common Divisor of n Integers

GORDON H. BRADLEY
Yale University, New Haven, Connecticut*

A new version of the Euclidean algorithm for finding the greatest common divisor of n integers a_i and multipliers x_i such that gcd $= x_1 a_1 + \cdots + x_n a_n$ is presented. The number of arithmetic operations and the number of storage locations are linear in n. A theorem of Lamé that gives a bound for the number of iterations of the Euclidean algorithm for two integers is extended to the case of n integers. An algorithm to construct a minimal set of multipliers is presented. A Fortran program for the algorithm appears as Comm. ACM Algorithm 386.

KEY WORDS AND PHRASES: greatest common divisor, Euclidean algorithm, number theory, diophantine equations
CR CATEGORIES: 3.15, 5.10

1. Introduction

The greatest common divisor, gcd, of two positive integers can be determined by the Euclidean algorithm. Euclid noted that the algorithm could be used to find the gcd of n positive integers; this follows since $\gcd(a_1, a_2, a_3) = \gcd(\gcd(a_1, a_2), a_3)$. If the steps of the algorithm are traced back, it is possible to determine integers x_i such that

$$\gcd(a_1, \cdots, a_n) = x_1 a_1 + \cdots + x_n a_n . \quad (1)$$

The x_i will be called "multipliers." Many applications require the multipliers, for instance, the Chinese problem of remainders [1], calculations on polynomial rings [1], solving linear diophantine equations [2], and computing the Hermite or Smith normal form of an integer matrix [2].

Blankinship [1] has proposed a new version of the Euclidean algorithm that eliminates the necessity of backtracking to determine the multipliers. An $(n) \times (n)$ matrix is carried along to keep track of the calculations. The number of arithmetic operations is proportional to n times the number of iterations of the Euclidean algorithm.

An improved algorithm for computing the gcd and multipliers is presented; the number of storage locations required is $2n$, and the number of arithmetic operations is $5n$ plus five times the number of iterations of the Euclidean algorithm. The notion of a minimal set of multipliers is introduced; an algorithm to construct a minimal set is presented.

A theorem of Lamé for bounding the number of iterations of the Euclidean algorithm for two integers is extended to provide a sharp bound on the number of iterations of the algorithm for n integers.

2. Blankinship Algorithm and MBA

A short description of the Blankinship version of the Euclidean algorithm [1] follows; see [7] for an ALGOL program. Let $a^T = (a_1, \cdots, a_n)$, form the matrix $[a, I]$.

Step 1. Call the row with the smallest nonzero first element the "operator" row.

Step 2. Select any other row with nonzero first element and call it the "operand" row. When no operand row exists, the process terminates.

Step 3. Add an integer multiple of the operator row to the operand row so that the first element of the resulting row is nonnegative and strictly less than the first element of the operator row. Return to step 1.

When the process is complete, the first element of the operator row is the gcd, and the other elements of the row are the multipliers.

Make the algorithm concrete by specifying in step 2 that the operand row be the topmost eligible row. If a_1 is initially the smallest a_i, then the algorithm can be seen to involve computing $\gcd(a_1, a_2)$, then $\gcd(\gcd(a_1, a_2), a_3), \cdots, \gcd(\gcd(a_1, a_2, \cdots, a_{n-1}), a_n)$.

Calculations. If $\gcd(\gcd(a_1, a_2, \cdots, a_{i-1}), a_i)$ requires K_i iterations and $K = \sum_{i=2}^{n} K_i$, then the Blankinship algorithm requires $(n + 1)K$ multiplications, K divisions, and $(n + 1)K$ additions. (Note $K \geq n - 1$.)

Storage. The algorithm needs n times $n + 1$ locations for input, working storage, and result (input destroyed).

In the new algorithm presented in Section 3, repeated use will be made of a new modified version of the Blankinship algorithm (MBA) for $n = 2$. To determine $\gcd(c_1, c_2)$ and multipliers y, z such that $\gcd(c_1, c_2) = yc_1 + zc_2$ form the matrix

$$\begin{bmatrix} c_1 & 1 \\ c_2 & 0 \end{bmatrix}$$

and perform the Blankinship algorithm on this matrix. At the completion of the process, the nonzero element in the first column is $\gcd(c_1, c_2)$, and the other element in that

* Administrative Sciences Department. This research was supported in part by funds from the Yale Computer Center. This paper gives the theoretical background of Algorithm 386, Greatest Common Divisor of n Integers and Multipliers, by the same author, appearing on pages 447–448 of this issue.

row is y. The other multiplier is calculated by

$$z = (\gcd(c_1, c_2) - yc_1)/c_2. \tag{2}$$

If the MBA takes L iterations, there are $2L + 1$ multiplications, $L + 1$ divisions, and $2L + 1$ additions.

The MBA is an improvement over existing algorithms for computing the gcd of two integers and multipliers; compare for instance [6, 3 pp. 302] or the Blankinship algorithm for $n = 2$.

3. New Algorithm

Step 1. 1.1 Use MBA to calculate $\gcd(a_1, a_2)$, save multipliers y_2, z_2.

1.2 Use MBA to calculate $\gcd(\gcd(a_1, a_2), a_3)$, save multipliers y_3, z_3.

...

1.$n-1$. Use MBA to calculate $\gcd(\gcd(a_1, a_2, \cdots, a_{n-1}), a_n)$, save multipliers y_n, z_n.

Final result is $\gcd(a_1, a_2, \cdots, a_n)$.

Step 2. 2.1 $x_n = z_n$; $y_n' = y_n$.

2.2 $x_i = z_i y_{i+1}'$, $\quad i = n-1, \cdots, 2$,
$y_i' = y_i y_{i+1}'$.

2.3 $x_1 = y_2'$.

PROOF. Step 1 follows since $\gcd(a_1, a_2, a_3) = \gcd(\gcd(a_1, a_2), a_3)$. For step 2,

$$\begin{aligned}\gcd(a_1, a_2, \cdots, a_n) &= z_n a_n + y_n(z_{n-1}a_{n-1} + y_{n-1}(z_{n-2}a_{n-2} + \cdots \\ &\quad + y_3(z_2a_2 + y_2a_1)\cdots)) \\ &= z_n a_n + y_n z_{n-1}a_{n-1} + y_n y_{n-1}z_{n-2}a_{n-2} + \cdots \\ &\quad + y_n y_{n-1}\cdots y_2 a_1. \end{aligned}$$

Q.E.D.

Calculations. If $\gcd(\gcd(a_1, a_2, \cdots, a_{i-1}), a_i)$ requires K_i iterations and $K = \sum_{i=2}^{n} K_i$, then the algorithm requires $2K + 3(n-1)$ multiplications, $K + n - 1$ divisions, and $2K + n - 1$ additions.

Storage. If overlay y_i on a_i, y_i' on y_i, and x_i on z_i, then $2n + 4$ locations are needed for input, working storage, and result (input destroyed).

The next section develops a sharp bound on K.

4. Bound for the Number of Iterations

Lamé in 1844 established a sharp bound on the number of iterations of the Euclidean algorithm for two integers ("sharp" indicates that the bound can be achieved). A theorem is developed that gives a sharp bound on the number of iterations of the algorithm for n integers.

THEOREM. (See Lamé [3] 1844, also [6, p. 43].) *The number of iterations for the Euclidean algorithm for two integers is never greater than five times the number of digits in the smaller number.*

The algorithm of the following theorem is: assume a_1 is the smallest a_i, compute $\gcd(a_1, a_2)$, then compute $\gcd(\gcd(a_1, a_2), a_3)$, etc. Any bound for this algorithm must be at least as great as the bound from the Lamé theorem for a_1, a_2 plus at least one iteration for each of the remaining $n - 2$ calculations. Surprisingly, this bound can be shown to hold.

THEOREM. *The number of iterations of the Euclidean algorithm for n integers is never greater than $n - 2$ plus five times the number of digits in the smallest number.*

PROOF. The following is a modification of the proof of Lamé, see [6 pp. 43–45]. Let the operation of computing $\gcd(a_1, a_2)$ be described as

$$\begin{aligned} a_2 &= q_1 a_1 + b_1 \\ a_1 &= q_2 b_1 + b_2 \\ &\cdots \\ b_{i-2} &= q_i b_{i-1} + b_i. \end{aligned} \tag{3}$$

The final step is

$$b_{i-1} = q_{i+1} b_i. \tag{4}$$

The quotients $q_1, \cdots, q_i$ are all greater than or equal to 1; q_{i+1} is greater than 1 since $b_{i-1} > b_i$.

At this point, $\gcd(a_1, a_2) = b_i$ is used to begin the calculation of $\gcd(b_i, a_3)$. The first step of the new calculation is

$$a_3 = r_1 b_i + c_1. \tag{5}$$

This initial iteration of the second application of the Euclidean algorithm for the integers will be counted in the $n - 2$ of the bound. If $c_1 = 0$, there are no additional iterations, assume $c_1 > 0$. Equation (4) can be rewritten as

$$b_{i-1} = (q_{i+1} - 1)b_i + c_1 + (b_i - c_1) \tag{6}$$

where $r_1 = (q_{i+1} - 1) \geq 1$ and $(b_i - c_1) > 0$.

The iterations for $\gcd(b_i, a_3)$ continue

$$\begin{aligned} b_i &= r_2 c_1 + c_2 \\ &\cdots \\ c_{j-2} &= r_j c_{j-1} + c_j \\ c_{j-1} &= r_{j+1} c_j. \end{aligned} \tag{7}$$

Note that $r_1, \cdots, r_j \geq 1$ and $r_{j+1} \geq 2$.

As the algorithm proceeds, the first iteration of each application of the Euclidean algorithm is counted in the $n - 2$ of the bound and the last step of the previous application is rewritten like (6). Ignoring the iterations counted in the $n - 2$ of the bound, and relabeling the relations of (3), (6), (7), etc., the following system of inequalities can be written

$$\begin{aligned} a_1 &\geq b_1 + b_2 \\ b_1 &\geq b_2 + b_3 \\ &\cdots \\ b_{k-1} &\geq 2b_k \\ b_k &\geq 1 \end{aligned} \tag{8}$$

where $k + 1$ is the number of iterations we wish to bound.

The Fibonacci series is defined by $u_1 = 1$, $u_2 = 2$, $u_i = u_{i-1} + u_{i-2}$, $i = 3, 4, \cdots$. From (8), $b_k \geq 1 = u_1$, $b_{k-1} \geq 2 = u_2$, $b_{k-2} \geq b_{k-1} + b_k \geq u_1 + u_2 = u_3$, in general

$b_i \geq b_{i+1} + b_{i+2} \geq u_{k-i+1}$. Consequently,

$$a_1 \geq u_{k+1}. \tag{9}$$

This inequality will be used to determine an upper bound on $k + 1$.

The Fibonacci series can be bounded by the powers of $g = (1 + (5^{\frac{1}{2}}))/2$

$$g^i < u_{i+1} \qquad i = 1, 2, \cdots. \tag{10}$$

Combining (10) with (9) gives

$$a_1 > g^k,$$

or

$$k < \log a_1/\log g. \tag{11}$$

If the number of digits of a_1 is p, then $a_1 < 10^p$ or $\log a_1 < p$, also $\log g > 1/5$ which gives the result $k < 5p$ and $k + 1 \leq 5p$. This is added to the $n - 2$ iterations counted separately to give the result. Q.E.D.

The bound of the theorem is sharp because the bound of the Lamé theorem is sharp; for instance the bound is achieved with 8, 13, 15 or 9, 24, 26.

COROLLARY. *The number of iterations of the Euclidean algorithm for n integers is less than n − 1 plus the logarithm of the smallest number to the base 1.6.*

PROOF. From the proof of the theorem (11) can be rewritten

$$k < \log_{1.6} a_1/\log_{1.6} g. \tag{12}$$

Since $\log_{1.6} g > 1$, $k < \log_{1.6} a_1$ and $k + 1 < \log_{1.6} a_1 + 1$. Q.E.D.

Even though the Lamé bound is sharp, computational experience indicates that the average number of iterations is less than the bound would suggest (in [3, pp. 320–333] computational experience and heuristic reasoning is used to assert that the average number of iterations is of the form $1.9405 \log_{10} a_1$). The above proof gives some insight into this phenomenon. The proof shows that at any iteration of the algorithm the remainder b_i can be adjusted (with malice) and although there may be no two integers that could yield an identical sequence of iterations, the bound still holds. The assumption that nature is nonmalicious leads one to expect fewer iterations than the bound would suggest.

There are other possible arrangements of the Euclidean algorithm such as the least remainder algorithm; see [6, pp. 45–51] for a description and a discussion of bounds. The number of iterations for the least remainder version is less than or equal to the number of iterations for the algorithm described above. The possible reduction in the number of iterations must be weighed against the additional number of arithmetic operations per iteration.

5. Minimal Set of Multipliers

Levit [5] introduces the notion of a "definitely least solution" for the linear diophantine equation $yc_1 + zc_2 = c_3$ where c_1 and c_2 are relatively prime. A solution $\bar{y}$, $\bar{z}$ is said to be a "definitely least solution" if $|\bar{y}| \leq \frac{1}{2}|c_2|$ and $|\bar{z}| \leq \frac{1}{2}|c_1|$. The MBA constructs a definitely least solution for $c_3 = \gcd(c_1, c_2)$; the MBA may be viewed as a constructive proof that a definitely least solution exists for this value of c_3.

Although the multipliers y_i, z_i are definitely least solutions for two integers, the method of Section 3 constructs multipliers for the n integers, that are, in general, not small numbers.

Definition. Let $x_1a_1 + \cdots + x_na_n = \gcd(a_1, \cdots, a_n)$. Let the nonzero elements among $x_2, \cdots, x_n$ be relabeled $x_{i(1)}, \cdots, x_{i(k)}$. A set of multipliers is called *minimal* if

$$\prod_{j=1}^{k} |x_{i(j)}| \leq (\tfrac{1}{2})^k |a_1| / \gcd(a_1, \cdots, a_n).$$

For example, for the integers 424; 444; 932; 22,347 the multipliers constructed by the method in Section 3 are 122,914; −117,327; 0; −1. A minimal set of multipliers is 37; 15; 0; −1.

The above method can be modified to construct a minimal set of multipliers. At step 1.i ($i = 1, \cdots, n - 1$) in addition to y_{i+1}, z_{i+1} save

$$v_{i+1} = \gcd(a_1, \cdots, a_i)/\gcd(a_1, \cdots, a_{i+1})$$

and

$$u_{i+1} = -a_{i+1}/\gcd(a_1, \cdots, a_{i+1}).$$

Let $[[x]]$ denote the integer nearest to x. Step 2.2 becomes:

$$x_i = z_iy'_{i+1} - v_i[[z_iy'_{i+1}/v_i]]$$

and

$$y_i' = y_iy'_{i+1} - u_i[[z_iy'_{i+1}/v_i]]$$

for $i = n - 1, \cdots, 2$.

The pair of numbers (u_i, v_i) constructed above is a generator for all the solutions to the equation $\gcd(a_1, \cdots, a_{i-1})y + a_iz = 0$. That is, all integer solutions to this equation are an integer multiple of this solution. A proof that the above method constructs a minimal set of multipliers is easily developed by carrying out the complete Blankinship algorithm for n integers and then adding multiples of the nonoperator rows to the operator row. In addition, note that the absolute value of the determinant of each $n \times n$ submatrix is unchanged by the algorithm.

The method for minimal multipliers requires an additional $2n$ storage locations; the number of additional arithmetic operations is linear in n. A FORTRAN implementation of the method is available from the author.

6. Comments and Conclusions

The algorithm described here also can be used to solve a single linear diophantine equation

$$a_1y_1 + a_2y_2 + \cdots + a_ny_n = b. \tag{13}$$

The eq. (13) has a solution if and only if $\gcd(a_1, a_2, \cdots, a_n)$ divides b. If (13) has a solution and if $c = b, \gcd/a_1, a_2, \cdots, a_n)$ then the multipliers x_i give a solution $y_i = cx_i$, $i = 1, \cdots, n$.

Often there exists a proper subset of the a_i such that the

gcd of the subset equals the gcd of the whole set. If the a_i's are arranged such that a subset of minimal cardinality with this property comes first, then the algorithm will construct a set of multipliers with the minimum number of nonzero elements. In many applications it would be valuable to have such a set of multipliers, but it seems a nontrivial combinatorial problem to determine such a set.

The algorithm can be applied to elements in any Euclidean ring. For an example over polynomial rings see [1].

It has long been known that the Euclidean algorithm for two integers is efficient. The algorithm presented here shows that this efficiency can be extended to the n integer case with multipliers. It is more surprising that a sharp bound for the number of iterations of the n integer case can be established that is virtually the same as the bound for the two integer case.

RECEIVED SEPTEMBER, 1969; REVISED FEBRUARY, 1970

REFERENCES

1. BLANKINSHIP, W. A. A new version of the Euclidean algorithm. *Amer. Math. Mon. 70* (1963), 742–745.
2. BRADLEY, G. H. Algorithms for Hermite and Smith normal matrices and linear diophantine equations. Administrative Sciences Dep., Yale U., New Haven, Conn. (in preparation).
3. KNUTH, D. E. *The Art of Computing Programming, Vol. 2, Seminumerical Algorithms*. Addison-Wesley, Reading, Mass., 1969.
4. LAMÉ, G. Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers. *C.R. Acad. Sci. Paris*, 19 (1844), 867–870.
5. LEVIT, R. J. A minimum solution of a diophantine equation. *Amer. Math. Mon. 63* (1956), 647–651.
6. PECK, J. E. L. Algorithm 139, Solutions of the diophantine equation. *Comm. ACM 5*, 11 (Nov. 1962), 556.
7. PECK, J. E. L. Algorithm 237, Greatest common divisor. *Comm. ACM 7*, 9 (Aug. 1964), 481.
8. USPENSKY, J. V., AND HEASLET, M. A. *Elementary Number Theory*. McGraw-Hill, New York, 1939.

Coffman and Eve—*cont'd from page 432*

trees at the sacrifice in search times (and complexity) brought about by the need to sequence through lists of nodes at the same level (and in the same family). It should also be noted that a key in a prefix tree is stored at the *minimum* level such that its hash code (prefix) is distinct from all others represented in the tree. In the trees described by Sussenguth, a key with c characters (elements) is normally stored at level c. Finally, a potential value in hash coding would be in the randomization of keys so that groups of similar keys are made less frequent.

Appendix. Proof of Equation (11)

The eigenvalues λ_i, $i = 0, 1, 2, \cdots, m$ of the matrix $\mathcal{P}$ are distinct, real, positive, and given by the diagonal elements. That is,

$$\begin{aligned} \lambda_i &= 1 - \xi^{i+1}, \qquad i = 0, 1, 2, \cdots, m-1 \\ \lambda_m &= 1. \end{aligned} \tag{A1}$$

The eigenvectors $\mathbf{x}_j = \text{col}\,[x_{0j}, x_{1j}, \cdots, x_{mj}]$ as computed from

$$\mathcal{P}\mathbf{x}_j = \lambda_j \mathbf{x}_j, \qquad j = 0, 1, 2, \cdots, m \tag{A2}$$

are given by

$$\begin{aligned} x_{ij} &= \frac{1}{(1-\xi)(1-\xi^2)\cdots(1-\xi^{j-i})}, \qquad i < j \\ &= 1, \qquad i = j \\ &= 0, \qquad i > j. \end{aligned} \tag{A3}$$

Let us define the matrix $X = ((x_{ij}))$ with the x_{ij} given as above. To compute $\mathcal{P}^{(m)}$ we make use of the properties of the eigenvalues and apply the result (to be found in any basic text on matrix theory)

$$\mathcal{P}^{(m)} = XD^{(m)}X^{-1} \tag{A4}$$

where D is the diagonal matrix whose diagonal elements are given by $d_{ii} = \lambda_i$, $i = 0, 1, 2, \cdots, m$. It thus remains to calculate the inverse X^{-1}. It is a routine matter to verify that the elements $x_{ij}^{(-1)}$ of X^{-1} are given by

$$\begin{aligned} x_{ij}^{(-1)} &= \frac{(-1)^{i+j}\xi^{(j-i)(j-i-1)/2}}{(1-\xi)(1-\xi^2)\cdots(1-\xi^{j-i})}, \qquad i < j \quad \text{(A5)} \\ &= 1, \qquad i = j \quad \text{(A6)} \\ &= 0, \qquad i > j. \end{aligned}$$

At this point we can calculate $\mathcal{P}^{(m)}$ from eq. (A4). However, from eq. (5) and the form of π_0 we need only the first row computed from eq. (A4). Carrying out the required multiplication yields eq. (11).

RECEIVED FEBRUARY, 1970

REFERENCES

1. HIBBARD, T. N. Some combinatorial properties of certain trees with applications to searching and sorting. *J. ACM 9*, 1 (Jan. 1962), 13–28.
2. KNUTH, D. E. *The Art of Computer Programming, Vol. 1 Fundamental Algorithms*. Addison-Wesley, Reading, Mass., 1968.
3. MORRIS, R. Scatter storage techniques. *Comm. ACM 11*, 1 (Jan. 1968), 38–43.
4. SUSSENGUTH, E. H., JR. Use of tree structures for processing files. *Comm. ACM 6*, 5 (May 1963), 272–279.