A Shift-Remainder GCD Algorithm

Graham Norton*
Laboratoire d'Analyse Numérique, Mathématiques
Université Paul Sabatier
118 Route de Narbonne, 31062 Toulouse, France.

Abstract This paper studies an integer greatest common divisor algorithm which uses a "shift-divide" instruction to compute the gcd of two integers \( u, v \). If \( u > v \), the worst case run-time is \( \lceil \log_2 v \rceil + 1 \), and for uniformly distributed integers in the range \([0,u-1]\), the average run-time is experimentally 0.555 ln \( u \).

Introduction We discuss an integer gcd algorithm which uses shifting (division by 2) and a certain type of integer division to form remainders. The worst case analysis is an analogue of Lamé's Theorem. The average analysis is more subtle than Heilbronn's analysis of the Euclidean algorithm and so we give an experimental estimate of the average run-time only.

Since the time needed to shift-divide integers is comparable with the time needed to divide, we deduce that a machine equipped with a shift-divide instruction (in an arithmetic coprocessor, for example) should compute gcd's about 34% faster than the Euclidean algorithm on the average. The worst case improvement is approximately 30%.

Section 1 contains the basic result on the existence and uniqueness of shift-remainders (Theorem 2) and the associated algorithms. We also give a Lagrangian form of the shift-remainder gcd algorithm. The analyses are described in Section 2.

I am indebted to the referees for their valuable comments and especially to the referee who proved a generalization of Theorem 1.2 to all primes. I would also like to thank the Royal Society for financial assistance.

Theorem 1.2 and the shift-remainder gcd algorithm were first announced in Karlsruhe at AAECC-4 and in [N] (Theorem 1.4, algorithm

*On leave from the University of Bristol, U.K.
Section 1 The Algorithms

Regarding division as repeated subtraction, we obtain the left shift \(\text{gcd}\) algorithm of [B] from the Euclidean algorithm. We can reverse this process: by repeating the "shift-subtraction" step of the binary integer \(\text{gcd}\) algorithm (see [K, p. 321]) until the shifted difference is less than the divisor, we can "shift-divide" two odd integers.

Example 1 \(19 - 5 = 7.2\) and \(7 > 5\), so we continue: \(7 - 5 = 1.2\) and \(1 < 5\), so we stop. Thus \(19 = 5 + 7.2 = 5 + (5 + 1.2).2 = 3.5 + 1.4\) and so the odd "shift-remainder" on shift-dividing 19 by 5 is 1.

The following theorem establishes the existence and uniqueness of shift-remainders:

Theorem 2 Let \(u, v \in \mathbb{N}\) be odd with \(u > v\). Then either (i) \(v\) divides \(u\) or (ii) there are \(q, r, s \in \mathbb{N}\) with \(q, r\) odd such that \(u = q \cdot v + r \cdot 2^s\) \(r < v\), and \(q < 2^s\). Further, \(q, r\) and \(s\) are uniquely determined.

We will refer to \(q, r\) and \(s\) of Theorem 2 by \(u \text{ sdiv } v\), \(u \text{ smod } v\) and \(u \text{ shft } v\) respectively, where \(u \text{ smod } v = u \text{ shft } v = 0\) if \(v\) divides \(u\). These quantities can be computed as follows:

Algorithm 3 (Shift-division)

Input Odd integers \(u, v\) with \(u > v > 0\).

Output \(u \text{ sdiv } v\), \(u \text{ smod } v\) and \(u \text{ shft } v\).

procedure shift-division\((u,v: \text{ integer})\);

\(q := 0; r := u; s := 0; t := 1;\)

repeat \(r := r - v;\)

\(q := q + t;\)

if \(r \neq 0\) then repeat \(r := r / 2;\)

\(s := s + 1;\)

\(t := t * 2;\)

until odd\((r)\)
else \( s := 0; \)
unti\( r < v; \)
return \( q, r, s \)

The shift-division algorithm terminates since the successive shift-ed differences form a strictly decreasing sequence of non-negative integers. The loop invariants of the algorithm are "\( u = q \cdot v + r \cdot t \)", "\( t = 2^s \)"", "\( q \) is odd" and either (a) "\( r = s = 0 \)" or (b) "\( r \) is odd", "\( s > 0 \)" and "\( q < t \)".

To compute the \( \text{gcd} \) of two odd integers \( u, v \) using shift-remainders we first write \( u = q \cdot v + r \cdot 2^s \) as in the theorem, where \( r = 0 \) or we have \( 0 < r < v \) and \( r \) is odd. Since \( u \) and \( v \) are odd, \( \text{gcd}(u,v) = \text{gcd}(v,r^2) = \text{gcd}(v,r) \) and so if \( r \) is non-zero, we can continue with \( v \) and \( r \). For example, \( \text{gcd}(19,7) = \text{gcd}(7,3) = \text{gcd}(3,1) = \text{gcd}(1,0) = 1. \)

**Algorithm 4 (Shift-remainder gcd algorithm)**

**Input** Odd integers \( u, v \) with \( u > v > 0 \).

**Output** \( \text{gcd}(u,v) \).

**procedure** shift_remainder_gcd\((u,v : \text{integer});\)

while \( v \neq 0 \) do begin
\( r := u \bmod v; \)
\( u := v; \)
\( v := r; \)
end;

return \( u \)

The algorithm terminates since the shift-remainders form a strictly decreasing sequence of non-negative integers, and the loop invariants are "\( r \) is odd" and "\( \text{gcd}(u,v) = \text{gcd}(v,r) \)". It is easy to see that Algorithm 4 may be used to compute the \( \text{gcd} \) of any two integers (not both zero) since the number of common powers of 2 may be accumulated beforehand.

There is also a Lagrangian form of Algorithm 4:
Algorithm 5

Input Odd integers \( u, v \) with \( u \geq v > 0 \).

Output Integers \( r, s \) and \( t \) such that \( r = \gcd(u,v) \) and \( r = s u + t v \).

procedure L_shift_remainder_gcd(u,v : integer);

\[ u_3 := u; \quad v_3 := v; \]
\[ u_2 := 0; \quad v_2 := 1; \]

while \( v_3 \neq 0 \) do begin

\[ q := u_3 \text{sdiv} v_3; \]
\[ r_3 := u_3 - q * v_3; \quad r_2 := u_2 - q * v_2; \]
if \( r_3 \neq 0 \) then repeat \( r_3 := r_3 / 2; \)

if \( \text{even}(r_2) \) then \( r_2 := r_2 / 2 \)
else \( r_2 := (r_2 - \sigma(r_2) u) / 2 \)

until \( \text{odd}(r_3) \);

\[ u_3 := v_3; \quad u_2 := v_2; \]
\[ v_3 := r_3; \quad v_2 := r_2; \]
end;

\[ r := u_3; \quad t := u_2; \quad s := (r - t * v) / u; \]

return \( r, s, t \).

Here \( \sigma(r_2) \) denotes the sign of \( r_2 \) and is introduced to avoid large values of \( v_2 \). The algorithm terminates since it reduces to Algorithm 4 if we suppress the variables with subscript 2. In this case, the loop invariants are "\( q, r_3 \) are odd", "\( \gcd(u,v) = \gcd(v_3,r_3) \)" and "\( u | (u_3 - v u_2) \)", "\( u | (v_3 - v v_2) \)". Clearly, \( r = s u + t v \).

Example 6 Algorithm 5 applied to 20451 and 6035 yields \(-1438 20451 + 4873 6035 = 17\) as follows:

<table>
<thead>
<tr>
<th>( u_3 )</th>
<th>( v_3 )</th>
<th>( q )</th>
<th>( r_3 )</th>
<th>( u_2 )</th>
<th>( v_2 )</th>
<th>( r_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>20451</td>
<td>6035</td>
<td>1</td>
<td>14416</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7208</td>
<td></td>
<td></td>
<td>10225</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3604</td>
<td></td>
<td></td>
<td>-5113</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1802</td>
<td></td>
<td></td>
<td>7669</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>901</td>
<td></td>
<td></td>
<td>-6391</td>
</tr>
</tbody>
</table>
Algorithm 5 may be modified to deal with (not necessarily) odd integers $u$ and $v$ as follows: without loss of generality, we may assume that there are no common powers of 2 in $u$ and $v$.

(*first ensure that $u$ is odd *)

swapped := false;
if even($u$) then begin
    swap($u,v$);
    swapped := true;
end;

(* now initialize *)

$u_3 := u$; $v_3 := v$;
$u_2 := 0$; $v_2 := 1$;

while even($v_3$) do begin
    $v_3 := v_3 / 2$;
    if even($v_2$) then $v_2 := v_2 / 2$
    else $v_2 := (v_2 - \sigma(v_2) \cdot u) / 2$;
end;

As above, $\sigma(v_2)$ denotes the sign of $v_2$. At this stage, $u_3$ and $v_3$ are odd and $u | (u_3 - vu_2), u | (v_3 - vv_2)$ and so we can apply the main loop of L_Shift_Remainder_GCD to $u_3$ and $v_3$ to calculate $r$, $s$, and $t$. The final modification is to add

"if swapped then swap(s,t);"

before returning $r$, $s$, and $t."
Section 2  The Analyses

We begin with the worst case analysis. The appearance of the following integers is not unexpected - see [K, p. 338, nos. 27,28].

Proposition 1  Let $G_n$ be defined by $G_0 = 0$, $G_1 = 1$ and for $n \geq 2$, $G_n = G_{n-1} + 2G_{n-2}$. Then for $n \geq 1$ (a) $2^{n-2} \leq G_n \leq 2^{n-1}$ and (b) $G_n$ is given by $G_n = (2^n - (-1)^n) / 3$.

The next result is the analogue of Lamé's Theorem:

Theorem 2  If $u$, $v \in \mathbb{N}$ are odd, $u > v > 0$ and gcd$(u,v)$ is computed using $n$ shift-divisions, where $n \geq 1$, then $v \geq (2^{n+1} - (-1)^{n+1}) / 3$ and $n \leq \lceil \log_2 v \rceil + 1$.

The run-time average that we are interested in is defined as follows:

Definition 3  Let $u > 1$ be odd. For odd $v$ with $0 < v < u$, let $m(u,v)$ be the number of shift-divisions required to compute gcd$(u,v)$ and let $m(u) = \frac{2}{u-1} \sum m(u,v)$ where the sum is over all odd $v$ such that $0 < v < u$.

Our approach to estimating $m(u)$ is based on [K, pp. 353,354]: we first consider the average number of shift-divisions for relatively prime inputs.

Proposition 4  If $u > 1$ is odd, the number of odd integers $v$ with $0 < v < u$ and gcd$(u,v) = 1$ is $\phi(u)/2$, where $\phi$ denotes Euler's totient function.

Definition 5  For odd $u > 1$, let $m_1(u) = \frac{2}{\phi(u)} \sum m(u,v)$ where the sum is taken over all odd $v$ such that $0 < v < u$ and $v$ is relatively prime to $u$.

An experimental value of $0.355 \ln u + 0.815$ for $m_1(u)$ was obtained in the following way:

Experiment 6  For $i := 4.0$ to 8.5 by 0.5, let $u_i = \lceil e^i \rceil$, adding 1 to $u_i$ if it is even ( $u_i$ varies from 91 to 8103 ). Compute the actual values of $m_1(u_i)$ and do a least squares regression on the ten points $(i,m_1(u_i))$. 

Incidentally, the same experiment yielded 0.844 \ln u + 0.875 as an estimate of \tau_n, the corresponding average for the Euclidean algorithm. (See [K, p.353, formula (46)] for the definition of \tau_n.) This slope compares well with the true value of 0.843 (to three decimal places).

The averages \( m^1(u) \) and \( m(u) \) are related as follows:

**Proposition 7.** For odd \( u > 1 \),

\[
\frac{1}{u-1} \sum \phi(d) m^1(d)
\]

where the sum is over all divisors \( d \) of \( u \).

The well-known fact that \( \sum \phi(d) \ln d = \ln u - \sum \Lambda(d)/d \), where each sum is over all divisors \( d \) of \( u \) and \( \Lambda \) is van Mangoldt's function, finally yields 0.555 (\( \ln u - \sum \Lambda(d)/d \)) + 0.815 as our estimate of \( m(u) \).

**Bibliography**


Graham Norton
Laboratoire d'Analyse Numérique
Université Paul Sabatier
118 Route de Narbonne
31062 Toulouse, France