MATH 324 Summer 2006
Elementary Number Theory

Notes on the Euclidean Algorithm

Department of Mathematical and Statistical Sciences
University of Alberta

The Euclidean Algorithm

Given two positive integers a and b such that $b \nmid a$, we know from the division algorithm that there exist unique integers q_1 and r_1, such that

$$a = b \cdot q_1 + r_1 \tag{1}$$

with $0 < r_1 < b$. From (1), the integers a and b have the same common divisors as the integers b and r_1, and therefore $\gcd(a, b) = \gcd(b, r_1)$.

Continuing, we find unique integers q_2 and r_2, such that

$$b = r_1 \cdot q_2 + r_2 \tag{2}$$

with $0 < r_2 < r_1$ if $r_1 \nmid b$,

$$\vdots$$

and continuing in this manner, we find unique integers q_k and r_k, such that

$$r_{k-2} = r_{k-1} \cdot q_k + r_k \tag{k}$$

with $0 < r_k < r_{k-1}$ if $r_{k-1} \nmid r_{k-2}$.

This process must terminate, since the remainders are all nonnegative and are strictly decreasing. If r_n is the last nonzero remainder, then the last two equations are

$$\begin{aligned} r_{n-2} &= r_{n-1} \cdot q_n + r_n \\ r_{n-1} &= r_n \cdot q_{n+1}. \end{aligned}$$

It is clear that

$$\gcd(a, b) = \gcd(b, r_1) = \gcd(r_1, r_2) = \cdots = \gcd(r_{n-1}, r_n) = \gcd(r_n, 0) = r_n,$$

since at any stage the integers r_{k-2} and r_{k-1} have the same common divisors as the integers r_{k-1} and r_k, and hence the same greatest common divisor.

This is Euclid's algorithm for computing the greatest common divisor of two positive integers a and b. The extended Euclidean algorithm allows us to write $\gcd(a, b) = s \cdot a + t \cdot b$ for some integers s and t. This can be done by working from the bottom up in the equations in the Euclidean algorithm. However, we have to know all the quotients q_k and remainders r_k in order to do this.

On the other hand, we can work from the top down by first writing r_1 as a linear combination of a and b, and then using the second equation to write r_2 as a linear combination of b and r_1, and hence of a and b, and then using the third equation to write r_3 as a linear combination of r_2 and r_1, and hence of a and b. Once a quotient q_k is used in a calculation, it is no longer needed and so can be forgotten, in fact, the calculation of the s's and t's can be done as the equations in the Euclidean algorithm are derived.

The equations in the Euclidean algorithm have the form

$$\begin{aligned}
a &= b \cdot q_1 + r_1 \\
b &= r_1 \cdot q_2 + r_2 \\
r_1 &= r_2 \cdot q_3 + r_3 \\
&\vdots \\
r_{k-2} &= r_{k-1} \cdot q_k + r_k \\
&\vdots
\end{aligned}$$

and working from the top down, we want to find integers s_k and t_k such that

$$r_k = s_k \cdot a + t_k \cdot b$$

for each k, with $1 \leq k \leq n$.

From the first equation, we have $r_1 = 1 \cdot a + (-q_1) \cdot b$, so that we can take

$$\begin{aligned}
s_1 &= 1 \\
t_1 &= -q_1.
\end{aligned}$$

Similarly, from the second equation, $r_2 = b - r_1 \cdot q_2 = (-q_2) \cdot a + (1 + q_1 q_2) \cdot b$, so that we can take

$$\begin{aligned}
s_2 &= -q_2 \\
t_2 &= 1 + q_1 q_2.
\end{aligned}$$

Assuming that we have found s_i and t_i so that $r_i = s_i \cdot a + t_i \cdot b$, for $i = 1, 2, \ldots, k-1$, then from the equation for the remainder r_k, we need

$$\begin{aligned}
r_k &= r_{k-2} - q_k \cdot r_{k-1} \\
&= s_{k-2} \cdot a + t_{k-2} \cdot b - q_k \cdot (s_{k-1} \cdot a + t_{k-1} \cdot b) \\
&= (s_{k-2} - q_k \cdot s_{k-1}) \cdot a + (t_{k-2} - q_k \cdot t_{k-1}) \cdot b.
\end{aligned}$$

Therefore, we need

$$\begin{aligned}
s_k &= s_{k-2} - q_k \cdot s_{k-1} \\
t_k &= t_{k-2} - q_k \cdot t_{k-1}, \qquad (*)
\end{aligned}$$

for $k > 2$.

Now, if we define s_1, t_1, s_2, t_2 as above, then we can use $(*)$ to define s_k and t_k for larger values of k. In particular, if we define

$$\begin{aligned}
s_{-1} = 1 \quad &\text{and} \quad s_0 = 0, \\
t_{-1} = 0 \quad &\text{and} \quad t_0 = 1,
\end{aligned}$$

then the equations in $(*)$ are true for all $k \geq 1$, and

$$r_k = s_k \cdot a + t_k \cdot b$$

for all $k \geq 1$. If r_n is the last nonzero remainder, then

$$\gcd(a, b) = r_n = s_n \cdot a + t_n \cdot b.$$

Note that if we define $r_{-1} = a$ and $r_0 = b$, then the three sequences $\{r_k\}_{k\geq 0}$, $\{s_k\}_{k\geq 0}$, $\{t_k\}_{k\geq 0}$, all satisfy the *same* difference equation, but with *different* initial conditions. Therefore, as the Euclidean algorithm computes the remainders r_k in order to find the greatest common divisor of a and b, the same calculations can be used to simultaneously calculate the s_k's and t_k's.

Now it is easy to write the program which uses the extended Euclidean algorithm to compute the r_k's using subtractions instead of the division algorithm, and the same operations will calculate the s_k's and t_k's.

A working implementation of this algorithm, written in *C*, can be found below.

```
/* THE EXTENDED EUCLIDEAN ALGORITHM
   This program calculates the g.c.d. of two non-negative integers
   a and b and then writes  (a,b) = s*a + t*b   for some integers
   s and t.
   To exit the program, enter a = 0 and b = 0 */

#include<stdio.h>

main( ) {
   int x,y,a,b,s,t,sprime,tprime,r,temp;
   do {
         printf("\nFind the GCD of the positive integers a and b \n");
         printf("\nEnter a:  \n");
         scanf("%d",&a);
         printf("\nEnter b:  \n");
         scanf("%d",&b);
         if (a != 0 && b == 0) {
             temp = a; a = b; b = temp;
         }
         if ( b != 0) {
             x = a; y = b; s = 1; t = 0; sprime = 0; tprime = 1;
             if (x != 0) {
                 while (x != y) {
                   if (x > y) {
                       x = x - y; s = s - sprime; t = t - tprime;
                    }
                    if (x < y) {
                        y = y - x; sprime = sprime - s; tprime = tprime - t;
                    }
                 }
                 r = x;
             }
             else
                 r = y;
             printf("\nThe GCD of %d and %d is %d \n", a,b,r);
             printf("\n%d = (%d,%d) = (%d)*%d + (%d)*%d \n",r,a,b,s,a,t,b);
         }/* if */
         else
             printf("\nYOU CAN'T DIVIDE BY ZERO!\n");
   }while (a != 0 || b != 0);
   printf("\nBYE!\n");
}/* main */
```

Example. Let F_n be the n^{th} term in the *Fibonacci sequence*

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ \cdots$$

for $n \geq 0$.

Use the *Euclidean Algorithm* to compute the greatest common divisor of F_{12} and F_{18}, and then express (F_{12}, F_{18}) as a linear combination of F_{12} and F_{18}.

Solution. Note first that

$$F_{11} = 89,\ F_{12} = 144,\ F_{13} = 233,\ F_{14} = 377,\ F_{15} = 610,\ F_{16} = 987,\ F_{17} = 1597,\ F_{18} = 2584$$

Using the Euclidean Algorithm, with $F_{12} = 144$ and $F_{18} = 2584$, we have

$$\begin{aligned} 2584 &= 17 \cdot 144 + 136 \\ 144 &= \ \ 1 \cdot 136 + 8 \\ 136 &= 17 \cdot 8 + 0. \end{aligned}$$

Therefore,

$$(F_{12}, F_{18}) = (144, 2584) = 8.$$

Working backwards, we have

$$(F_{12}, F_{18}) = 8 = 144 - 136 = 144 - (2584 - 17 \cdot 144)$$

and therefore

$$(F_{12}, F_{18}) = 8 = 18 \cdot 144 + (-1) \cdot 2584.$$

As a check, note that we can factor these integers as:

$$\begin{aligned} 144 &= 12^2 = 3^2 \cdot 4^2 = 3^2 \cdot 2^4 \\ 2584 &= 8 \cdot 323 = 2^3 \cdot 17 \cdot 19, \end{aligned}$$

from which it is obvious that $(F_{12}, F_{18}) = 2^3 = 8$.

$\square$

Note that we used only 2 divisions in the Euclidean algorithm, so the algorithm can find the greatest common divisor fairly rapidly in most cases.

However, in the case of *consecutive* Fibonacci numbers, in each division, the quotient is 1, since

$$F_{n+2} = 1 \cdot F_{n+1} + F_n,$$

and the remainder is F_n, so the process is much slower.

Now we show that it takes exactly n divisions to compute the greatest common divisor of the consecutive Fibonacci numbers F_{n+1} and F_{n+2}, and that this is, in fact, the worst case scenario for the Euclidean algorithm.

Theorem.

If F_{n+1} and F_{n+2} are successive terms in the Fibonacci sequence, with $n > 1$, then F_{n+1} and F_{n+2} are relatively prime and the Euclidean algorithm takes exactly n divisions to verify that $\gcd(F_{n+1}, F_{n+2}) = 1$.

More generally, the number of divisions needed by the Euclidean algorithm to find the greatest common divisor of two positive integers does not exceed five times the number of decimal digits in the smaller of the two integers.

proof. Applying the Euclidean algorithm to F_{n+1} and F_{n+2}, and using the defining relation for the Fibonacci numbers at each step, we have

$$\begin{aligned}
F_{n+2} &= 1 \cdot F_{n+1} + F_n \\
F_{n+1} &= 1 \cdot F_n + F_{n-1} \\
F_n &= 1 \cdot F_{n-1} + F_{n-2} \\
&\vdots \\
F_4 &= 1 \cdot F_3 + F_2 \\
F_3 &= 2 \cdot F_2
\end{aligned}$$

and the last nonzero remainder is F_2, so that the greatest common divisor $\gcd(F_{n+2}, F_{n+1}) = F_2 = 1$, and F_{n+2} and F_{n+1} are relatively prime. It is clear from the calculations above that the Euclidean algorithm takes exactly n divisions to show that $\gcd(F_{n+2}, F_{n+1}) = F_2 = 1$.

Now let a and b be positive integers with $0 < b < a$. We will apply the Euclidean algorithm to find the greatest common divisor $\gcd(a, b)$ of a and b. Let $r_0 = a$ and $r_1 = b$, then we get the following sequence of equations:

$$\begin{aligned}
r_0 &= q_1 r_1 + r_2, && 0 \le r_2 < r_1, \\
r_1 &= q_2 r_2 + r_3, && 0 \le r_3 < r_2, \\
r_2 &= q_3 r_3 + r_4, && 0 \le r_4 < r_3, \\
&\vdots \\
r_{n-3} &= q_{n-2} r_{n-2} + r_{n-1}, && 0 \le r_{n-1} < r_{n-2}, \\
r_{n-2} &= q_{n-1} r_{n-1} + r_n, && 0 \le r_n < r_{n-1}, \\
r_{n-1} &= q_n r_n.
\end{aligned}$$

Here we have used n divisions. Note that each of the quotients $q_1, q_2, \ldots, q_{n-1}$ is greater than or equal to 1, and $q_n \ge 2$, since $r_n < r_{n-1}$. Therefore,

$$\begin{aligned}
r_n &\ge 1 = F_2 \\
r_{n-1} &\ge 2r_n \ge 2F_2 = F_3 \\
r_{n-2} &\ge r_{n-1} + r_n \ge F_3 + F_2 = F_4 \\
r_{n-3} &\ge r_{n-2} + r_{n-1} \ge F_4 + F_3 = F_5 \\
&\vdots \\
r_2 &\ge r_3 + r_4 \ge F_{n-1} + F_{n-2} = F_n \\
b = r_1 &\ge r_2 + r_3 \ge F_n + F_{n-1} = F_{n+1}.
\end{aligned}$$

Thus, if there are n divisions used in the Euclidean algorithm, then we must have $b \ge F_{n+1}$.

Now we show by induction that $F_{n+1} > \alpha^{n-1}$ for $n \geq 2$, where $\alpha = \dfrac{1+\sqrt{5}}{2}$.

The basis step consists of verifying this inequality for $n = 2$ and $n = 3$. Since $\alpha < 2 = F_3$ then the result is true for $n = 2$. Also, since $\alpha^2 = \dfrac{3+\sqrt{5}}{2} < 3 = F_4$, the result is true for $n = 3$.

The inductive hypothesis consists of assuming that $\alpha^{k-1} < F_{k+1}$ for all integers k with $2 \leq k < n$. Since $\alpha = \dfrac{1+\sqrt{5}}{2}$ is a solution of the quadratic equation $x^2 - x - 1 = 0$, then $\alpha^2 = \alpha + 1$, so that

$$\alpha^{n-1} = \alpha^2 \cdot \alpha^{n-3} = (\alpha + 1) \cdot \alpha^{n-3} = \alpha^{n-2} + \alpha^{n-3}.$$

From the induction hypothesis, we have

$$\alpha^{n-2} < F_n \quad \text{and} \quad \alpha^{n-3} < F_{n-1},$$

and adding these inequalities, we get

$$\alpha^{n-1} = \alpha^{n-2} + \alpha^{n-3} < F_n + F_{n-1} = F_{n+1},$$

so the result is also true for $k = n$. By the principle of mathematical induction, the result is true for all $n \geq 2$.

Therefore, $b \geq F_{n+1} > \alpha^{n-1}$ for $n \geq 2$, and since $\log_{10} \alpha > \dfrac{1}{5}$, then

$$\log_{10} b > (n-1) \log_{10} \alpha > \frac{(n-1)}{5},$$

and therefore

$$n - 1 < 5 \cdot \log_{10} b.$$

Now suppose that b has k decimal digits, so that $b < 10^k$, then $\log_{10} b < k$, so that $n - 1 < 5k$. Since k is an integer, this implies that $n \leq 5k$.

$\square$