**Computing Science 204 (B1)**

**Notes on the Extended Euclidean Algorithm**

Thursday, July 13

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

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

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 */
```