During our foray into number theory we learned about the greatest common divisor of two integers \( a \) and \( b \) (not both 0) and the importance of the existence of integral solutions to the equation

\[ ax + by = (a, b). \]

The one element of this discussion which is missing is a way to compute the greatest common divisor as well as compute solutions to the above equation. The importance of being able to compute the \( x \) and \( y \) manifests itself in modular arithmetic in the following way. If \( (a, n) = 1 \) then any solution to

\[ ax + ny = 1 \]

gives

\[ ax \equiv 1 \pmod{n} \]

so that \( x \) is a multiplicative inverse of \( a \) modulo \( n \). Thus in order to actually compute multiplicative inverses we must know how to solve \( ax + by = (a, b) \).

The key insight into beginning to understand how to compute lies in the following observation.

**Lemma 0.1.** Suppose \( a, b, q, r \in \mathbb{Z} \) with \( a = bq + r \). Then \( (a, b) = (b, r) \).

**Proof.** Let \( d = (a, b) \) and \( e = (b, r) \). Then since \( d \mid a \) and \( d \mid b \) we also get

\[ d \mid a - bq = r. \]

Thus \( d \leq e \) since \( e \) is the greatest common divisor of \( b \) and \( r \). Similarly, since \( e \mid b \) and \( e \mid r \) we also get

\[ e \mid bq + r = a \]

so that \( e \leq d \) and therefore \( d = e \). \( \square \)

Why should Lemma 0.1 be useful to us? Intuitively, one can reason as follows. If \( a > b \) are positive integers then we know we can find \( q, r \in \mathbb{Z} \) with \( 0 \leq r < b \) so that \( a = bq + r \). Then Lemma 0.1 says that to compute \( (a, b) \) we can instead compute \( (b, r) \). We know that \( b < a \) and \( r < b \) so somehow the problem of computing \( (b, r) \) should be easier. We can turn this idea into an iterative and efficient algorithm.

Given \( a > b \) let \( r_{-1} = a \) and \( r_0 = b \). For \( i \geq 0 \), as long as \( r_i > 0 \), we define \( r_{i+1} \) and \( r_{i+1} \) via the division algorithm

\[ r_{i-1} = r_i q_{i+1} + r_{i+1} \]

where \( 0 \leq r_{i+1} < r_i \). We have
Proposition 0.2. There exists an \( n \geq 0 \) so that \( r_n > 0 \) and \( r_{n+1} = 0 \). For this \( n \) we have \( r_n = (a, b) \).

Proof. Let’s first prove that \( r_n \) exists. To do so, let

\[
S = \{ r_i | r_i > 0 \}.
\]

Then \( S \) is a non-empty subset of the positive integers so it has a smallest element \( r_n \). Then

\[
r_{n-1} = r_n q_{n+1} + r_{n+1}
\]

where \( 0 \leq r_{n+1} < r_n \). Since \( r_n \) is the smallest element of \( S \) we must have \( r_{n+1} = 0 \).

By definition of the \( r_i \)’s, Lemma 0.1 tells us that \( (r_i, r_{i+1}) = (r_{i+1}, r_{i+2}) \).

We can thus use induction to prove that

\[
(a, b) = (r_{-1}, r_0) = (r_n, r_{n+1}).
\]

Of course \( r_{n+1} = 0 \) so \( (r_n, r_{n+1}) = r_n \) and the proof is complete. \( \square \)

To not get lost in all the symbols, let’s compute an example.

Example 0.3. Let’s compute \((6540, 1206)\). We have

\[
\begin{align*}
6540 &= 1206 \cdot 5 + 510 \\
1206 &= 510 \cdot 2 + 186 \\
510 &= 186 \cdot 2 + 138 \\
186 &= 138 \cdot 1 + 48 \\
138 &= 48 \cdot 2 + 42 \\
48 &= 42 \cdot 1 + 6 \\
42 &= 6 \cdot 7 + 0
\end{align*}
\]

so that \((6540, 1206) = 6\).

It’s worth pointing out that this algorithm translates into repeated modular arithmetic. We can rephrase Example 0.3 by saying

\[
\begin{align*}
6540 &\equiv 510 \pmod{1206} \\
1206 &\equiv 186 \pmod{510} \\
510 &\equiv 138 \pmod{186} \\
186 &\equiv 48 \pmod{138} \\
138 &\equiv 42 \pmod{48} \\
48 &\equiv 6 \pmod{42} \\
42 &\equiv 0 \pmod{6}.
\end{align*}
\]

Now we’d like to be able to use this idea, the Euclidean Algorithm, to produce solutions to \( ax + by = (a, b) \). We can do this via forward or backward substitution. Let’s look at a simple example.
Example 0.4. Let’s consider 13 and 5. We have

\begin{align*}
13 &= 5 \cdot 2 + 3 \\
5 &= 3 \cdot 1 + 2 \\
3 &= 2 \cdot 1 + 1 \\
2 &= 1 \cdot 2 + 0.
\end{align*}

Suppose now we wanted to solve $13x + 5y = 1$. We can use Example 0.4 to solve this. Forward substitution would start with the first line of the Euclidean Algorithm, which says

$$13 - 5 \cdot 2 = 3$$

and substitute this equation into the second line of the Euclidean Algorithm to get

$$5 = (13 - 5 \cdot 2) \cdot 1 + 2$$

or equivalently

$$5 \cdot 3 + 13 \cdot (-1) = 2.$$  

We can continue to substitute into the third line to get

$$13 - 5 \cdot 2 = (5 \cdot 3 + 13 \cdot (-1)) \cdot 1 + 1$$

which rewrites as

$$13 \cdot 2 + 5 \cdot (-5) = 1.$$

This gives us a solution to $13x + 5y = 1$! Moreover, one can see that this method will work much more generally. It is called forward substitution. We can also use backward substitution which is a similar idea but starts at the second to last step of the algorithm

$$3 - 2 = 1$$

and uses the previous line,

$$5 - 3 = 2$$

to substitute. This gives

$$3 - (5 - 3) = 1$$

or

$$3 \cdot 2 - 5 = 1.$$  

Then again we go to the previous line which says

$$13 - 5 \cdot 2 = 3$$

so substitute:

$$(13 - 5 \cdot 2) \cdot 2 - 5 = 1$$

or

$$13 \cdot 2 + 5 \cdot (-5) = 1.$$  

Again to connect back with modular arithmetic, we see for example that

$$5 \cdot (-5) \equiv 1 \pmod{13}.$$
and since \(-5 \equiv 8 \mod 13\) we get that the multiplicative inverse of 5 modulo 13 is 8. That is, in \(J_{13}\) we have

\[ [5][8] = [1]. \]

There exist faster ways to solve \(ax + by = (a, b)\) but almost all of them rely on first performing the Euclidean Algorithm. My favorite method, which I will include without proof (the proof is not hard and might be worth thinking about) is the following.

Let \(q_1, q_2, \ldots, q_{n+1}\) be the sequence of quotients showing up in the Euclidean Algorithm. In Example 0.3 this sequence would be

\[ 5, 2, 2, 1, 2, 1, 7. \]

Let \(P_{-1} = 0, P_0 = 1, Q_{-1} = 1\) and \(Q_0 = 0\). For \(i \geq 1\) we inductively define

\[ P_i = q_i \cdot P_{i-1} + P_{i-2}, \]

\[ Q_i = q_i \cdot Q_{i-1} + Q_{i-2}. \]

For Example 0.3, after organizing into a table, we get

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>5</td>
<td>11</td>
<td>27</td>
<td>38</td>
<td>103</td>
<td>141</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>19</td>
<td>26</td>
</tr>
</tbody>
</table>

so for example \(P_3 = 27\) and \(Q_5 = 19\). One can check that on this table we always have

\[ P_i \cdot Q_{i+1} - P_{i+1} \cdot Q_i = \pm 1. \]

For example,

\[ 103 \cdot 26 - 141 \cdot 19 = -1. \]

at the very end of the table we have

\[ 141 \cdot 201 - 26 \cdot 1090 = 1. \]

Notice that \(201 \cdot 6 = 1206\) and \(1090 \cdot 6 = 6540\). Thus if we multiply both sides of the equation by 6 we get

\[ 1206 \cdot 141 - 6540 \cdot 26 = 6 \]

a solution to \(1206x + 6540y = 6\).

Remarkably, this method always works. In Example 0.4, the quotients are \(2, 1, 1, 2\). This gives the table

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

and sure enough \(5 \cdot 5 - 13 \cdot 2 = -1\) so that \(13 \cdot 2 - 5 \cdot 5 = 1\). It will take a little bit of work, but from the definition of the \(P_i\) and \(Q_i\), one can figure out and prove why this table always gives the right answer.
I’d like to make a few concluding and interesting remarks regarding computation of the GCD. Herstein 1.3.5 tells us that if
\[ a = p_1^{e_1} \cdots p_k^{e_k} \]
and
\[ b = p_1^{f_1} \cdots p_k^{f_k} \]
are prime factorizations for \( a \) and \( b \) then
\[ (a, b) = p_1^{g_1} \cdots p_k^{g_k} \]
where \( g_i = \min(e_i, f_i) \). This gives us another way of computing GCDs, but in practice it is useless compared to the Euclidean Algorithm. The reason is because factoring integers is HARD. Try factoring 6540 and 1206 by hand. You might quickly realize that both are divisible by 6 and so
\[ 6540 = 6 \cdot 1090, \quad 1206 = 6 \cdot 201 \]
but then how easy is it to factor 1090 and 201? Even worse, suppose \( a \) and \( b \) are huge numbers with hundred of digits. It should be much easier to do repeated division than it is to factor. In fact, it is currently a giant open question in mathematics and computer science as to whether one can factor in a “reasonable” amount of time. Without background in computer science it’s hard to define what a reasonable amount of time means, but intuitively the question centers around whether one can factor a number \( n \) in a more efficient manner than just trial and error of dividing it by integers from 2 up to \( \sqrt{n} \). Most people think the answer is no you cannot, and this is the basis for much of modern cryptography.