

Laboratory 1

The Euclidean Algorithm and Applications

Math 3530

1 What to do before the lab

Read through the Linux/GAP/emacs Reference Guide. A little familiarity with the system that will be used should significantly reduce any frustrations you might have. You should also read this laboratory description and do the written exercises.

1.1 File needed for this lab

GCD.gap

2 Greatest Common Divisor

Suppose a and b are integers, with $b > 0$. Recall that we say b **divides** a , denoted $b|a$, if and only if there is an integer q such that $a = bq$. More generally, we have the

Division Algorithm: Suppose a and b are integers, with $b > 0$. Then there exist unique integers q and r such that

$$a = bq + r \text{ and } 0 \leq r < b.$$

The number q of the division algorithm, called the **quotient upon dividing a by b** , is equal to $\lfloor a/b \rfloor$, while r , the **remainder upon dividing a by b** , is equal to $a \bmod b$. In this notation we have

$$b|a \iff a \bmod b = 0$$

and

$$a = b \cdot \lfloor a/b \rfloor + a \bmod b$$

This last formula is particularly useful when it is necessary to compute $a \bmod b$ using a calculator for large values of a and b .

Definition: If a and b are integers, at least one of which is nonzero, then the **greatest common divisor of a and b** , denoted $\gcd(a, b)$, is the largest integer which divides both a and b . In the case that $\gcd(a, b) = 1$, we say a and b are **relatively prime**.

It is easy to see that if a is not zero, then $\gcd(a, 0) = |a|$. In general, $\gcd(a, b) = \gcd(|a|, |b|)$. Thus, the problem of computing greatest common divisors is reduced to finding a method which finds the greatest common divisor of positive integers. The Euclidean Algorithm is such a method.

Euclidean Algorithm

Input: Positive integers a and b

$$r_1 \leftarrow a; r_2 \leftarrow b$$

while $r_2 \neq 0$

$$r_0 \leftarrow r_1; r_1 \leftarrow r_2$$

$$r_2 \leftarrow r_0 \bmod r_1$$

endwhile

Output: r_1 , the greatest common divisor of a and b .

Mathematical induction can be used to *prove* this algorithm provides the appropriate output from a given input.

Proof of the Euclidean Algorithm: For $k = 0, 1, \dots$, let $P(k)$ be the statement:

After the while loop has been executed k times, $\gcd(r_1, r_2) = \gcd(a, b)$. If $k \geq 1$ and the loop is executed k times, the value of r_2 at the end of k -th loop is less than its value at the beginning of the k -th loop.

Initially, $r_1 = a$ and $r_2 = b$. Hence, $P(0)$ is clearly true.

Assume, by way of induction, that $P(k)$ is true from some $k \geq 0$. The statement $P(k+1)$ is trivially true if the loop is not executed $k+1$ times. Hence, assume that it has been executed $k+1$ times. By abuse of notation, r_1 and r_2 , respectively, will be used to represent both the variables **and** the values of those variables at the end of the $k+1$ -st loop. Let R_1 and R_2 , respectively, be the values of those variables at the beginning of the $k+1$ -st loop.

By the inductive hypothesis, $\gcd(R_1, R_2) = \gcd(a, b)$. According to the algorithm, $r_1 = R_2$. Also, $r_2 = R_1 \bmod R_2$. Hence, $r_2 < R_2$, i.e. the value of the variable r_2 at the end of the loop is less than its value at the beginning of the loop. Further, if $q = \lfloor R_1/R_2 \rfloor$, then $r_2 = R_1 - qR_2$. Hence, using the inductive hypothesis, it follows that

$$\gcd(r_1, r_2) = \gcd(R_2, R_1 - qR_2) = \gcd(R_2, R_1) = \gcd(a, b).$$

3 Extended Euclidean Algorithm

One of the most important properties of the greatest common divisor of two integers a and b is that it can be represented as a linear combination of a and b .

Theorem: If a and b are integers, at least one of which is nonzero, and if $d = \gcd(a, b)$, then there exists integers x and y such that

$$ax + by = d$$

In fact, d is the smallest positive number in the set $\{ax + by : x, y \in \mathbb{Z}\}$.

The Euclidean Algorithm can be easily extended to return the greatest common divisor, d , as well as two integers x and y such that $ax + by = d$. It should be noted that these two integers are definitely not unique.

Extended Euclidean Algorithm

Input: Positive integers a and b

```

 $r_1 \leftarrow a; r_2 \leftarrow b$ 
 $x_1 \leftarrow 1; x_2 \leftarrow 0$ 
 $y_1 \leftarrow 0; y_2 \leftarrow 1$ 
while  $r_2 \neq 0$ 
     $r_0 \leftarrow r_1; r_1 \leftarrow r_2$ 
     $x_0 \leftarrow x_1; x_1 \leftarrow x_2$ 
     $y_0 \leftarrow y_1; y_1 \leftarrow y_2$ 
     $q \leftarrow \lfloor r_0/r_1 \rfloor$ 
     $r_2 \leftarrow r_0 - qr_1$  {or equivalently,  $r_2 \leftarrow r_0 \bmod r_1$  }
     $x_2 \leftarrow x_0 - qx_1; y_2 \leftarrow y_0 - qy_1$ 
endwhile

```

Output: r_1, x_1, y_1 where $r_1 = \gcd(a, b) = ax_1 + by_1$.

The following is a trace of the extended Euclidean Algorithm which shows the values of the variables for the input values $a = 49$ and $b = 38$.

k	values after the loop is executed k times									
	r_0	r_1	r_2	x_0	x_1	x_2	y_0	y_1	y_2	q
		49	38		1	0		0	1	
1	49	38	11	1	0	1	0	1	-1	1
2	38	11	5	0	1	-3	1	-1	4	3
3	11	5	1	1	-3	7	-1	4	-9	2
4	5	1	0	-3	7	38	4	-9	49	5

Note that $\gcd(r_1, r_2)$ is always the same and the algorithm terminates when $\gcd(r_1, r_2) = r_1$. Also, it is always true that $49x_1 + 38y_1 = r_1$. Hence, when the algorithm terminates, the greatest common divisor of 49 and 38 has been found as well as two numbers, 7 and -9 , such that

$$7 \cdot 49 + (-9) \cdot 38 = \gcd(49, 38).$$

4 Using GAP

Sign on to a computer in OM304, obtain the file `GCD.gap` for this lab, start a GAP session reading `GCD.gap` into it, and log the GAP session. For more details, see the `Linux/GAP/emacs Reference Guide`.

For a little practice, enter the following commands into your session.

```
gap> 2^10;
gap> Factorial(4);
gap> Combinations([1,2,3,4,5], 2);
gap> Size(last);
```

The first command computes 2^{10} , while the second computes $4!$. The third gives all subsets of size 2 of the set $\{1, 2, 3, 4, 5\}$. In the fourth command, the `Size` function is used to find the number of elements in the previous output (`last` means “previous output”).

The following GAP session uses the implementation of the extended Euclidean Algorithm given in `GCD.gap`. The lines beginning with `#` are comments and are not executed by GAP. Enter the lines without a comment mark into your session. You will probably get an error message if the file needed for this lab is not in the appropriate directory.

```
gap> # Compute the greatest common divisor of 49 and 38
gap> # and store that in ANS. Storing the result is not
gap> # essential but does make it possible to recall the values later.
gap> ANS := GCD(49,38);
rec( gcd := 1, coeff1 := 7, coeff2 := -9 )
gap> # This means the output ANS is a record with three
gap> # components --- gcd which is 1, coeff1 which is 7
gap> # and coeff2 which is -9. Thus, 1 is the greatest
gap> # common divisor of 49 and 38 and, further, 1 = 7 * 49 + (-9) * 38.
gap> # The components of ANS can be extracted individually.
gap> ANS.gcd;
1
gap> ANS.coeff1;
7
```

5 Solution of Equations and Multiplicative Inverses

The output of the extended Euclidean Algorithm can be used to solve equations involving modular arithmetic. Consider the equation

$$38x \equiv 1 \pmod{49}.$$

We want to find numbers x such that $38x$ is one more than a multiple of 49. From the equation $7 \cdot 49 + (-9) \cdot 38 = 1$ it is easy to see that $38 \cdot (-9) = 1 + 49 \cdot (-7)$ and that $38 \cdot 40 = 1 + 49 \cdot 31$. In fact, $x \equiv 40 \pmod{49}$ if and only if $38x \equiv 1 \pmod{49}$.

Definition: If a and b are integers with $b > 0$, we say c is the **multiplicative inverse of a modulo b** provided $0 < c < b$ and $ac \bmod b = 1$.

Several multiplicative inverses can be found by examining the equation $7 \cdot 49 + (-9) \cdot 38 = 1$. This equation implies that 7 is the multiplicative inverse of 49 modulo 38, 38 is the multiplicative inverse of -9 modulo 49, and 40 is the multiplicative inverse of 38 modulo 49.

6 Functions

Functions are simple to write in GAP and will be found to be very useful. The following GAP session defines a function `ModSum` which finds the value of $\sum_{k=1}^{n-1} k \bmod n$.

```
gap> ModSum := function( n )
> # Purpose: To find the sum, modulo n, of the first n-1
> #           positive integers.
> # Input: n - a positive integer
> # Output: S - the sum
> local k, S;
> S := 0;
> for k in [1..n-1] do
>   S := (S + k) mod n;
> od;
> return S;
> end;
```

Notes:

- The first line gives the name of the function, `ModSum`, and indicates that it has a single variable, `n`. There is no semi-colon at the end of this line.
- The comment lines do not have to be entered.
- The next line declares the local variables for the function, `k` and `S`.
- The initial value of `S` is 0.
- Each member of the set $\{1, 2, \dots, n-1\}$ is added to `S` and before the result is stored in `S` it is reduced modulo n .
- The final value of `S` is returned to the main program and the function definition terminates with `end`;
- To invoke this function to find the value of $\sum_{k=1}^{17} k \bmod 18$, enter `ModSum(18)`.

In the exercises, below, you will use functions to formulate hypotheses and to check conjectures.

7 Exercises

Some of these exercises require GAP. However, some of them only require a written solution. Those that require GAP are marked with an asterisk.

7.1 Written exercises

1. Show that, for x an integer, $x \bmod 49 = 40$ if and only if $38x \bmod 49 = 1$. (Part of this has been done above!)
2. Let a, b, c be positive integers. Prove or disprove: If $d = \gcd(a, b) = \gcd(b, c)$, then $d = \gcd(a, c)$.
3. Trace the extended Euclidean Algorithm for $a = 37, b = 14$.

7.2 Computer Exercises

1. * Enter all of the sample sections, above, into GAP. You do not need to enter the comment lines.
2. * Either find the smallest positive integer x which satisfies the equation $43x \bmod 152 = 1$ or show no such solution exists.
3. * Either find the smallest positive integer x which satisfies the equation $153x \bmod 219 = 1$ or show no such solution exists.
4. * Use GAP to find the smallest positive integer x which satisfies the equation $1345x \bmod 4321 = 5$. Hint: First solve the equation $1345x \bmod 4321 = 1$.
5. * By using different values in the function `ModSum` defined above, make a conjecture concerning the value of

$$\sum_{k=1}^{n-1} k \bmod n.$$

(The correct conjecture depends upon n .)

6. Try to prove your conjecture from the previous problem (by “hand”).
7. * Write a function which will compute $(n - 1)! \bmod n$. This function should differ only slightly from the function of the previous problem. By using different values, make a conjecture concerning the value of

$$(n - 1)! \bmod n.$$

(The correct conjecture depends upon n .) At first glance, a user-defined function does not seem necessary. To find $(12345)! \bmod 123456$ we could just find $(12345)!$ using the `Factorial` command and then record the remainder when this result is divided by 123456. But what is `Factorial(12345)`?

8. Try to prove your conjecture. (Note: Finding the correct conjecture should be relatively simple. Proving part of this conjecture should also be relatively simple. However, part of this conjecture is quite difficult to prove.)

8 What to Hand In

Your solutions are due at the beginning of class of the due date.

Your submission should include solutions to the written exercises as well as an edited file of your GAP session. To produce such a file, log your session and then use `emacs` to add comment lines and delete erroneous output.

9 Appendix

Contents of GCD.gap

```
GCD := function(a, b)
  # Purpose: To find the greatest common divisor of a and b
  #           and to express it as a linear combination of a and b.
  # Input : a and b - non-zero integers.
  # Output: result - a record containing
  #         gcd - the greatest common divisor
  #         coeff1 and coeff2 - integers satisfying
  #         gcd = coeff1 * a + coeff2 * b
  local x0,x1,x2,y0,y1,y2,r0,r1,r2,q, result,
         gcd, coeff1, coeff2;
  r1 := a; r2 := b;
  x1 := 1; y1 := 0;
  x2 := 0; y2 := 1;
  while (r2 <> 0) do
    r0 := r1; r1 := r2;
    x0 := x1; x1 := x2;
    y0 := y1; y1 := y2;
    q := Int(r0/r1);
    r2 := r0 - q*r1;
    x2 := x0 - q*x1;
    y2 := y0 - q*y1;
  od;
  result := rec(
    gcd := r1,
    coeff1 := x1,
    coeff2 := y1
  );
  return result;
end;
```