CSC110 Lecture 16: Greatest Common Divisor, Revisited

Print this handout

Exercise 1: The Euclidean Algorithm

Here is the code for the *Euclidean Algorithm* for calculating the greatest common divisor of two numbers.

```
def euclidean_gcd(a: int, b: int) -> int:
    """Return the gcd of a and b.

    Preconditions:
    - a >= 0
    - b >= 0
    """
    x, y = a, b

    while y != 0:
        # Loop invariant:
        # assert math.gcd(x, y) == math.gcd(a, b)

        r = x % y

        x, y = y, r

    return x
```

1. Suppose we make the following function call:

   ```
   >>> euclidean_gcd(50, 23)
   ```

   Complete the following table to show the values of the loop variables `x` and `y` at the end of each loop iteration when we call this function. We have completed the first row for you, and have given you more rows than necessary (part of this exercise is determining exactly when the loop will stop).

Iteration	`x`	`y`
0	50	23
1		
2		
3		
4		
5		
6		
7		

2. What does `euclidean_gcd(10, 0)` return? Is this correct (according to the definition of gcd)?

3. What does `euclidean_gcd(0, 0)` return? Is this correct (according to the definition of gcd)?

4. How can we modify this function to allow for negative values for `a` and `b`? (You don't need a formal proof of correctness here, but should briefly justify your response.)

Exercise 2: Completing the Extended Euclidean Algorithm

We left off our discussion of the *Extended Euclidean Algorithm* in lecture with the following code:

```
def extended_euclidean_gcd(a: int, b: int) -> tuple[int, int, int]:
    """Return the gcd of a and b, and integers p and q such that

    gcd(a, b) == p * a + b * q.

    Preconditions:
    - a >= 0
    - b >= 0

    >>> extended_euclidean_gcd(10, 3)
    (1, 1, -3)
    """
    x, y = a, b

    # Initialize px, qx, py, and qy
    px, qx = 1, 0  # Since x == a == 1 * a + 0 * b
    py, qy = 0, 1  # Since y == b == 0 * a + 1 * b

    while y != 0:
        # assert math.gcd(a, b) == math.gcd(x, y)  # Loop Invariant 1
        assert x == px * a + qx * b                # Loop Invariant 2
        assert y == py * a + qy * b                # Loop Invariant 3

        q, r = divmod(x, y)  # quotient and remainder when a is divided by b

        # Update x and y
        x, y = y, r

        # Update px, qx, py, and qy
        px, qx, py, qy = ...,  ..., ..., ...

    return (x, px, qx)
```

Remember that the key new **loop invariants** are:

```
x == px * a + qx * b
y == py * a + py * b
```

Now, let's investigate how to complete this function by filling in the `...` in the loop body.

1. Suppose we call `extended_euclidean_gcd(100, 13)`.

   Complete the **first row** in table below to show the *initial values* of all six loop variables at the very start of the loop (which we label "Iteration 0"). Leave the other rows blank for now; we'll get to them in future questions.

Iteration	`x`	`px`	`qx`	`y`	`py`	`qy`
0						
1						
2						
3						
4						

   Before moving on, check that your values satisfy `x == px * 100 + qx * 13` *and* `y == py * 100 + qy * 13`.

2. Next, suppose we execute one iteration of the loop. Note that `divmod(100, 13) == (7, 9)`, i.e., 100 divided by 13 has quotient 7 and remainder 9.

   a. In the above table, fill in the values for `x` and `y` in the "Iteration 1" row.

   b. Determine what numbers you should use to fill in for `px` and `qx` in "Iteration 1" to preserve the invariant `x == px * 100 + qx * 13`. Then, fill those entries.

      You may use the space below for rough work.

   c. Does what you wrote generalize for all possible values of `x`, `y`, `a`, and `b`? Convince yourself that it does, and then use this information to fill in the first two `...` in the reassignment statement for `px` and `qx`. (Halfway done!)

      You may use the space below for rough work.

   d. Now let's look at `py` and `qy`. From the quotient-remainder theorem, you have the equation:

      $$100 = 7 \cdot 13 + 9$$

      Use this information to fill in the values for `py` and `qy` for Iteration 1, while preserving the invariant `y == py * 100 + qy * 13`.

      You may use the space below for rough work.

   e. Unfortunately, what you did in (e) doesn't generalize just yet. So, let's do another iteration.

      First, using what you've learned fill in the "Iteration 2" row for `x`, `px`, `py`, `y`.

   f. Now, you are given the following equations:

      $$\begin{aligned} 13 &= 0 \cdot 100 + 1 \cdot 13 \\ 9 &= 1 \cdot 100 - 7 \cdot 13 \\ 13 &= 1 \cdot 9 + 4 \end{aligned}$$

      Using these equations, perform a calculation to find coefficients p and q such that $4 = p \cdot 100 + q \cdot 13$. Then, use this to complete the "Iteration 2" row of the table.

   g. Let's generalize what you did in the previous step. Given the following equations:

      $$\begin{aligned} \mathtt{x} &= \mathtt{px} \cdot 100 + \mathtt{qx} \cdot 13 \\ \mathtt{y} &= \mathtt{py} \cdot 100 + \mathtt{qy} \cdot 13 \\ \mathtt{x} &= \mathtt{q} \cdot \mathtt{y} + \mathtt{r} \end{aligned}$$

      Write `r` as a linear combination of 100 and 13.

   h. Use your work in the previous part to complete the Extended Euclidean Algorithm by filling in the last two `...` to update `py` and `qy`.

Congratulations, you've just completed a derivation of the Extended Euclidean Algorithm!