Last update: August 15, 2024 Translated From: e-maxx.ru

Extended Euclidean Algorithm

While the Euclidean algorithm calculates only the greatest common divisor (GCD) of two integers a and b, the extended version also finds a way to represent GCD in terms of a and b, i.e. coefficients x and y for which:

$$a \cdot x + b \cdot y = \gcd(a, b)$$

It's important to note that by Bézout's identity we can always find such a representation. For instance, $\gcd(55, 80) = 5$, therefore we can represent 5 as a linear combination with the terms 55 and 80: $55 \cdot 3 + 80 \cdot (-2) = 5$

A more general form of that problem is discussed in the article about Linear Diophantine Equations. It will build upon this algorithm.

Algorithm

We will denote the GCD of a and b with g in this section.

The changes to the original algorithm are very simple. If we recall the algorithm, we can see that the algorithm ends with $b = 0$ and $a = g$. For these parameters we can easily find coefficients, namely $g \cdot 1 + 0 \cdot 0 = g$.

Starting from these coefficients $(x, y) = (1, 0)$, we can go backwards up the recursive calls. All we need to do is to figure out how the coefficients x and y change during the transition from (a, b) to $(b, a \bmod b)$.

Let us assume we found the coefficients (x_1, y_1) for $(b, a \bmod b)$:

$$b \cdot x_1 + (a \bmod b) \cdot y_1 = g$$

and we want to find the pair (x, y) for (a, b):

$$a \cdot x + b \cdot y = g$$

We can represent $a \bmod b$ as:

$$a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b$$

Substituting this expression in the coefficient equation of (x_1, y_1) gives:

$$g = b \cdot x_1 + (a \bmod b) \cdot y_1 = b \cdot x_1 + \left(a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b\right) \cdot y_1$$

and after rearranging the terms:

$$g = a \cdot y_1 + b \cdot \left(x_1 - y_1 \cdot \left\lfloor \frac{a}{b} \right\rfloor\right)$$

We found the values of x and y:

$$\begin{cases} x = y_1 \\ y = x_1 - y_1 \cdot \left\lfloor \frac{a}{b} \right\rfloor \end{cases}$$

Implementation

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

The recursive function above returns the GCD and the values of coefficients to `x` and `y` (which are passed by reference to the function).

This implementation of extended Euclidean algorithm produces correct results for negative integers as well.

Iterative version

It's also possible to write the Extended Euclidean algorithm in an iterative way. Because it avoids recursion, the code will run a little bit faster than the recursive one.

```
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}
```

If you look closely at the variables `a1` and `b1`, you can notice that they take exactly the same values as in the iterative version of the normal Euclidean algorithm. So the algorithm will at least compute the correct GCD.

To see why the algorithm computes the correct coefficients, consider that the following invariants hold at any given time (before the while loop begins and at the end of each iteration):

$$x \cdot a + y \cdot b = a_1$$

$$x_1 \cdot a + y_1 \cdot b = b_1$$

Let the values at the end of an iteration be denoted by a prime ($'$), and assume $q = \frac{a_1}{b_1}$. From the Euclidean algorithm, we have:

$$a_1' = b_1$$

$$b_1' = a_1 - q \cdot b_1$$

For the first invariant to hold, the following should be true:

$$x' \cdot a + y' \cdot b = a_1' = b_1$$

$$x' \cdot a + y' \cdot b = x_1 \cdot a + y_1 \cdot b$$

Similarly for the second invariant, the following should hold:

$$x_1' \cdot a + y_1' \cdot b = a_1 - q \cdot b_1$$

$$x_1' \cdot a + y_1' \cdot b = (x - q \cdot x_1) \cdot a + (y - q \cdot y_1) \cdot b$$

By comparing the coefficients of a and b, the update equations for each variable can be derived, ensuring that the invariants are maintained throughout the algorithm.

At the end we know that a_1 contains the GCD, so $x \cdot a + y \cdot b = g$. Which means that we have found the required coefficients.

You can even optimize the code more, and remove the variable a_1 and b_1 from the code, and just reuse a and b. However if you do so, you lose the ability to argue about the invariants.

Practice Problems

- UVA - 10104 - Euclid Problem
- GYM - (J) Once Upon A Time
- UVA - 12775 - Gift Dilemma