Euclid's Algorithm

By: Aieshah Nasir (0827CS201018)

GREATEST COMMON DIVISOR

The **Greatest Common Divisor** (GCD) or the **Highest Common Factor** (HCF) of two or more natural numbers is the largest natural number which can divide each of the numbers. As the name suggests, GCD is a factor or a divisor common to the numbers.

Traditionally, GCD is calculated by:

- Prime factorization
- Repeated division

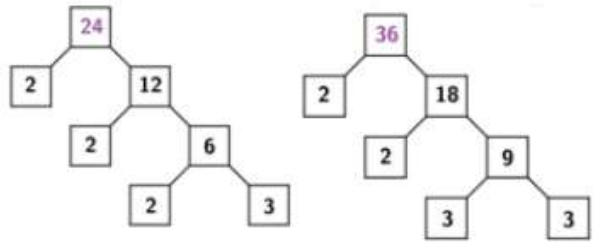

Figure 1: Prime Factorization

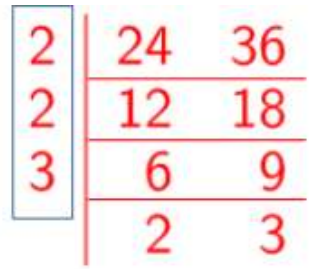

Figure 2: Repeated Division

For example, to find $GCD(24, 36)$, find the prime factors of 24 and 36, list them out and multiply the common factors to get the greatest common factor.

$$24 = 2 \times 2 \times 2 \times 3 \qquad 36 = 2 \times 2 \times 3 \times 3$$

$$GCD(24, 36) = 2 \times 2 \times 3 = 12$$

Performing prime factorization on computers becomes difficult and time – consuming as the magnitude of the numbers increase and thus, less efficient.

We explore the most basic and the earliest-known algorithm for efficient computation of GCD.

ASSERTION

Consider the following statement,

$$For\ any\ two\ positive\ integers\ a\ and\ b, such\ that\ a > b,$$

$$the\ common\ divisors\ of\ a\ and\ b = the\ common\ divisors\ of\ (a-b)\ and\ b.$$

Really? Let's explore this statement to satisfy our curiosity.

In the above example, we found that

$$GCD(24, 36) = 12$$

By the above statement,

$$GCD(24, 36) = GCD(24, 36 - 24) \because 36 > 24$$
$$= GCD(24, 12)$$

Again, apply the above statement to get

$$GCD(24, 12) = GCD(24 - 12, 12) \because 24 > 12$$
$$= GCD(12, 12)$$
$$= 12$$

The result is same as what we found earlier. Amazing!

EUCLID'S ALGORITHM

Euclid, a Greek mathematician, introduced a method based on the above discussed assertion to compute the greatest common divisors.

The following is a function called $gcd()$ implemented in C with the stopping condition $a == b$.

```
int gcd(int a, int b)    // Recursive Function
{
  if(a == b)  return a;
  if (a > b)    return gcd(a - b, b);
```

```
  if (a < b)    return gcd(a, b − a);
}
int gcd(int a, int b)   // Iterative Function
{
  while(a != b){
    if (a > b)
      a = a − b;
    else
      b = b − a;
  }
  return a;
}
```

However, when the absolute difference between the two integers is very large, this method becomes very slow.

EUCLIDEAN ALGORITHM

Euclidean algorithm is a variant of Euclid's algorithm, the change being the use of the remainder instead of the difference. Remember that *division is repeated subtraction*. This modification in the algorithm reduces the total number of iterations required to compute the GCD and proves to be more efficient.

$Algorithm\ to\ find\ the\ GCD\ of\ a_1\ and\ a_2, such\ that\ a_1 > a_2 \geq 0$:

1. $If\ a_2 = 0, then\ gcd = a_1$
2. $If\ a_2 > 0, a_1 = a_2 q_2 + a_3\ with\ a_2 > a_3 \geq 0$
3. $Replace\ a_1\ by\ a_2, a_2\ by\ a_3\ and\ go\ to\ step\ 1$

Consider the following examples:

Example 1: To find the $gcd(662, 414)$:

We have $a_1 = 662$ and $a_2 = 414$.

Divide a_1 by a_2 to get the remainder as a_3.

$$662 = 414 \times 1 + 248 \ \therefore a_3 = 248$$

Swap a_1 with a_2 and a_2 with a_3 and continue the division process until $a_2 = 0$.

$\therefore a_1 = 414, \quad a_2 = 248$

$$414 = 248 \times 1 + 166 \ \therefore a_3 = 166$$

$\because a_1 = 248, \ a_2 = 166$

$$248 = 166 \times 1 + 82 \ \therefore a_3 = 82$$

$\because a_1 = 166, \quad a_2 = 82$

$$166 = 82 \times 2 + 2 \ \therefore a_3 = 2$$

$\because a_1 = 82, \quad a_2 = 2$

$$82 = 2 \times 41 + 0 \ \therefore a_3 = 0$$

After swapping, $a_1 = 2, \ a_2 = 0$.

Hence, we stop and $gcd(662,414) = \ a_1 = 2$.

Example 2: To find the $gcd(36, 24)$:

We have $a_1 = 36$ and $a_2 = 24$.

Divide a_1 by a_2 to get the remainder as a_3.

$$36 \ = \ 24 \times 1 + 12 \ \therefore a_3 = 12$$

Swap a_1 with a_2 and a_2 with a_3 and continue the division process until $a_2 = 0$.

$\therefore a_1 = 24, \quad a_2 = 12$

$$24 = 12 \times 2 + 0 \ \therefore a_3 = 0$$

$\because a_1 = 12, \ a_2 = 0$, we stop and $gcd(36, 24) = \ a_1 = 12$ which is the same as what we found above using prime factorization.

C implementation of this variant of Euclid's algorithm is as follows, assuming that $a \geq b$:

```
int gcd(int a, int b)   // Iterative Function
{
  while(b != 0) {
      // divide and swap until b becomes zero
      int temp = b;
```

```
        b = a % b;      // '%' operator returns the remainder
        a = temp;
    }
    return a;
}
int gcd(int a, int b)    // Recursive Function
{
    if (b == 0)         return a;
    else                return gcd(b, a % b);
}
```

This algorithm has been further modified to work on negative numbers also. However, this has not been discussed here.

APPLICATIONS OF GCD

GCD is useful for simplifying fractions, compute modular inverses and in modular arithmetic. Modular inverses are of extreme importance in encryption schemes such as RSA.