Cryptography and Network Security Lab

Assignment 8
Implementation and Understanding of Euclid and Extended Euclid's Algorithm

2019BTECS00058
Devang K
Batch: B2

Title: Implementation and Understanding of Euclid and Extended Euclid's Algorithm

Aim: To Study, Implement and Demonstrate the:

- Euclid's Algorithm
- Extended Euclid's Algorithm

Euclid's Algorithm

Theory:

The Euclidean algorithm is a way to find the greatest common divisor of two positive integers. GCD of two numbers is the largest number that divides both of them. A simple way to find GCD is to factorize both numbers and multiply common prime factors.

Basic Euclidean Algorithm for GCD:
The algorithm is based on the below facts.

- If we subtract a smaller number from a larger one (we reduce a larger number), GCD doesn't change. So if we keep subtracting repeatedly the larger of two, we end up with GCD.

- Now instead of subtraction, if we divide the smaller number, the algorithm stops when we find the remainder 0.

Let’s illustrate with an example:

We wish to calculate the GCD of 1220 and 516.

1220 and 516

We first compute – 1220 % 516 = 188

Then, 516 % 188 = 140

Then 188 % 140 = 48

Then 140 % 48 = 44

Then 48 % 44 = 4

Then 44 % 4 = 0

Therefore, GCD is 4

This is computed in a tabular form. The operation of GCD gives other useful results, which we shall see in the Extended Euclid Theorem.

Code:

```
from prettytable import PrettyTable

def gcd(a, b):
    # ensure that a is always the larger number
    if b > a:
        temp = a
        a = b
        b = temp
    theGCDData = []
    q = a//b
    r =  a%b
    theGCDData.append([q, a, b, r])
    while(b>0):
        a = b
```

```
        b = r
        if b == 0:
            theGCDData.append(["-", a, b, "-"])
            break
        q = a//b
        r = a%b
        theGCDData.append([q, a, b, r])
    return theGCDData


print("Enter the 2 numbers:\n")
print("First Number: ", end='')
a = int(input())
print("Second Number: ", end='')
b = int(input())

# Initialise the table
x = PrettyTable()
x.field_names = ["q", "a", "b", "r"]

theGCDData = gcd(a,b)
for i in theGCDData:
    x.add_row(i)

print("\nThe GCD Table:")
print(x)
print("\nThe GCD of "+str(a)+" "+str(b)+" is:
"+str(theGCDData[len(theGCDData)-1][1]))
```

We now demonstrate with an example

Let’s compute GCD of 1220 and 516 as in the illustration.

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Euclid\GCD> py .\script.
py
Enter the 2 numbers:

First Number: 1220
Second Number: 516

The GCD Table:
+----+------+-----+-----+
| q  |  a   |  b  |  r  |
+----+------+-----+-----+
| 2  | 1220 | 516 | 188 |
| 2  | 516  | 188 | 140 |
| 1  | 188  | 140 |  48 |
| 2  | 140  |  48 |  44 |
| 1  |  48  |  44 |  4  |
| 11 |  44  |  4  |  0  |
| -  |  4   |  0  |  -  |
+----+------+-----+-----+
```


```
The GCD Table:
+----+------+-----+-----+
| q  |  a   |  b  |  r  |
+----+------+-----+-----+
| 2  | 1220 | 516 | 188 |
| 2  | 516  | 188 | 140 |
| 1  | 188  | 140 |  48 |
| 2  | 140  |  48 |  44 |
| 1  |  48  |  44 |  4  |
| 11 |  44  |  4  |  0  |
| -  |  4   |  0  |  -  |
+----+------+-----+-----+
```


```
The GCD of 1220 516 is: 4
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Euclid\GCD>
```

Therefore, the GCD is 4.

Let’s take another example:

We compute GCD of 9973 and 1009

```
First Number: 9973
Second Number: 1009

The GCD Table:
+----+------+-----+-----+
| q  |  a   |  b  |  r  |
+----+------+-----+-----+
| 9  | 9973 | 1009 | 892 |
| 1  | 1009 | 892  | 117 |
| 7  | 892  | 117  |  73 |
| 1  | 117  |  73  |  44 |
| 1  |  73  |  44  |  29 |
| 1  |  44  |  29  |  15 |
| 1  |  29  |  15  |  14 |
| 1  |  15  |  14  | 1   |
| 14 |  14  | 1    | 0   |
| -  |  1   | 0    | -   |
+----+------+-----+-----+

The GCD of 9973 1009 is: 1
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Euclid\GCD>
```

The GCD is 1, thus, the numbers are co-prime.

Hence, we demonstrated Euclid's Algorithm.

Extended Eulid's Algorithm

Theory

In arithmetic and computer programming, the extended Euclidean algorithm is an extension to the Euclidean algorithm, and computes, in addition to the greatest common divisor (gcd) of integers a and b, also the coefficients of Bézout's identity, which are integers x and y such that:

$$ax + by = gcd(a,b)$$

This is a certifying algorithm, because the gcd is the only number that can simultaneously satisfy this equation and divide the inputs. It allows one to compute also, with almost no extra cost, the quotients of a and b by their greatest common divisor.

Extended Euclidean algorithm also refers to a very similar algorithm for computing the polynomial greatest common divisor and the coefficients of Bézout's identity of two univariate polynomials.

The extended Euclidean algorithm is particularly useful when a and b are coprime. With that provision, x is the modular multiplicative inverse of a modulo b, and y is the modular multiplicative inverse of b modulo a. Similarly, the polynomial extended Euclidean algorithm allows one to compute the multiplicative inverse in algebraic field extensions and, in particular in finite fields of non prime order. It follows that both extended Euclidean algorithms are widely used in cryptography. In particular, the computation of the modular multiplicative inverse is an essential step in the derivation of key-pairs in the RSA public-key encryption method.

Following our Euclid's algorithm, we introduce 3 columns to our table – t1, t2 and t, t1 and t2 initialized as 0 and 1. Then we perform computation to obtain t. Following this, we get the values to obtain the multiplicative inverse of a number.

Code:

```
from prettytable import PrettyTable

def gcd(a, b):
    # ensure that a is always the larger number
    if b > a:
        temp = a
        a = b
        b = temp
    theGCDData = []
    q = a//b
    r =  a%b
    t1 = 0
    t2 = 1
    t = t1 - t2*q
    theGCDData.append([q, a, b, r, t1, t2, t])
    while(b>0):
        a = b
        b = r
        t1 = t2
        t2 = t
        if b == 0:
            theGCDData.append(["-", a, b, "-", t1, t2, "-"])
```

```python
            break
        q = a//b
        r = a%b
        t = t1 - q*t2
        theGCDData.append([q, a, b, r, t1, t2, t])
    return theGCDData

print("\n-- Multiplicative Inverse using Extended Euclid Algo --\n")
print("Enter Number a: ", end='')
a = int(input())
print("Enter Number b: ", end='')
b = int(input())

# Initialise the table
x = PrettyTable()
x.field_names = ["q", "a", "b", "r", "t1", "t2", "t"]

theGCDData = gcd(a,b)
for i in theGCDData:
    x.add_row(i)

print("\nThe Extended Euclid Table:")
print(x)
print("\nThe M.I. of "+str(a)+"%"+str(b)+" is:
"+str(theGCDData[len(theGCDData)-1][4]))
```

Now, let’s see demonstrate with example.

We wish to compute multiplicative inverse of 37 mod 50

Then,

```
-- Multiplicative Inverse using Extended Euclid Algo --

Enter Number a: 37
Enter Number b: 50

The Extended Euclid Table:
+---+----+----+----+----+-----+-----+
| q | a  | b  | r  | t1 |  t2 |  t  |
+---+----+----+----+----+-----+-----+
| 1 | 50 | 37 | 13 | 0  |  1  |  -1 |
| 2 | 37 | 13 | 11 | 1  |  -1 |  3  |
| 1 | 13 | 11 | 2  | -1 |  3  |  -4 |
| 5 | 11 | 2  | 1  | 3  |  -4 |  23 |
| 2 | 2  | 1  | 0  | -4 |  23 | -50 |
| - | 1  | 0  | -  | 23 | -50 |  -  |
+---+----+----+----+----+-----+-----+

The M.I. of 37%50 is: 23
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Euclid\Extended>
```

We find that the Modular Inverse is 23.

We can verify it as – a * a(-1) mod b must equal 1

37 * 23 = 851

851 mod 50 = 1

Hence verified.

Now, we take another example:

Say we wish to compute 34 mod 67

```
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Euclid\Extended> ^C
ript.py

-- Multiplicative Inverse using Extended Euclid Algo --

Enter Number a: 34
Enter Number b: 67

The Extended Euclid Table:
+----+----+----+----+----+-----+-----+
| q  | a  | b  | r  | t1 |  t2 |  t  |
+----+----+----+----+----+-----+-----+
| 1  | 67 | 34 | 33 | 0  |  1  |  -1 |
| 1  | 34 | 33 | 1  | 1  |  -1 |  2  |
| 33 | 33 | 1  | 0  | -1 |  2  | -67 |
| -  | 1  | 0  | -  | 2  | -67 |  -  |
+----+----+----+----+----+-----+-----+

The M.I. of 34%67 is: 2
PS C:\Users\marcus\Desktop\College\CNS-Lab-Archives\Euclid\Extended>
```

We get modular inverse as 2.

We can verify as:

34 * 2 = 68

68 mod 67 = 1

We thus illustrated Extended Euclids Algorithm with example.

Conclusion:

Thus, the Euclid's algorithm and Extended Euclid's algorithm was studied and demonstrated with the code.