# Modulus Notes

## Euclidean Algorithm: GCD

Considering a > b

GCD(a, b) = GCD(b, (a%b)) but why?

### Proof:

a = b . q1 + r1 eq. 1

b = r1 . q2 + r2 eq. 2

r1 = r­2 . q­3 + r3 eq. 3

…..

rn-3 = rn-2 . qn-1 + rn-1 eq. n-1

rn-2 = rn-1 . qn + 0 eq. n

Algorithm says rn-1 is the gcd of a and b when rn = 0.

### Proof 1: This iteration will end

Since ri = (ri-2 % ri-1), so ri < ri-1. This is true for all i, so we can conclude:

a > b > r1 > r2 > r3 > … > rn-1 > rn > 0

### Proof 2: rn-1 is a common divisor of a and b

From the equ n, we can say rn-1 divides rn-2

rn-1 | rn-2

Now from equ n-1 we can say,

rn-1 | rn-3

Same way we can say rn-1 | a and rn-1 | b

### Proof 3: rn-1 is the greatest common divisor

Let d be another integer such that d | a and d | b

Now, from equation 1 we can say d | r1

From equation 2 we can say d | r2

…

From equation n-1 we can say d | rn-1

Which implies d < rn-1. It says rn-1 is the greatest common divisor.

## Rules:

(a + b) % M = (a % M + b % M) % M

(a \* b) % M = ((a % M) \* (b % M)) % M

(a – b) % M = (a % M – b % M + M) % M

(a / b) % M = (a % M \* (b-1) % M) % M

## Extended Euclid Algorithm:

Let say we have 2 values a & b then the gcd(a, b) -> g.

Then we can say ax + by = g but why??

Remember any equation ax’ + by’ = 1, always has a solution,

Since g | a, g | b

=> g | (ax + by)

So the equation has a solution.

Now we want to find the values of ‘x’ and ‘y’.

* ax + by = gcd(a, b) = k
* bx1 + (a%b) y1 = gcd(b, a%b) = k
* bx1 + (a – b \* (floor(a / b)))y1 = k

From two equation

* ax + by = bx1 + (a – b \* (floor(a / b)))y1
* ax + by = ay1 + b \* (x1 - floor(a / b)\*y1)

Equating coefficient of a & b

x = y1

y = x1 - floor(a / b)\*y1

This is a recusive equation

Program to find the value of x, y given a, b

class TripletEqn {

    public:

    int x;

    int y;

    int gcd;

    TripletEqn(int *x*, int *y*, int *gcd*){

        this->x = *x*;

        this->y = *y*;

        this->gcd = *gcd*;

    }

    TripletEqn(){}

};

TripletEqn getExtendedEuclid(int *a*, int *b*) {

    if(*b* == 0){

        return TripletEqn(1, 0, *a*);

    }

    TripletEqn ret = getExtendedEuclid(*b*, *a*%*b*);

    TripletEqn soln = TripletEqn();

    soln.x = ret.y;

    soln.y = ret.x - (*a*/*b*) \* ret.y;

    soln.gcd = ret.gcd;

    return soln;

};

void solve()

{

    int a = 35, b = 15;

    TripletEqn solution = getExtendedEuclid(a, b);

    printf("(%d) . %d + (%d) . %d = %d\n", solution.x, a, solution.y, b, solution.gcd);

}

Output: (1) . 35 + (-2) . 15 = 5

## Multiplicative Modulo Inverse

### Multiplicative Inverse

The multiplicative inverse of a number ‘a’ is b iff:

a . b = 1

### Modular Congruency

If, a . b ≡ 1 (mod M) Then, b is called Multiplicative Modular Inverse.

Note: x ≡ y (mod z) => x is congruent to y, on mod z => z | (x – y) or (x % z = y)

Problem: Given the value of a and M, find the value of b.

For example a = 3 and M = 5, we can say b = 2. as ab – M = 1

In equation we can write:

* ab – 1 = Mq
* ab – M . q = 1
* a . b + M . Q = 1

We can use extended modulo inverse, to solve the equation and find the value of b and Q. We already have gcd(a, M) = 1. That why in CP we have M as a prime number.

## Fermat Little Theorem

If p is a prime number and p does not divide a, then ap-1≡ 1 (mod p)

For example, Consider p = 5,

24 ≡ 1 (mod 5)

34 ≡ 1 (mod 5)

44 ≡ 1 (mod 5)

64 ≡ 1 (mod 5)

Now, we have ap-1≡ 1 (mod p)

Or, a . ap-2 ≡ 1 (mod p)

So, a-1 ~ ap-2