WEEK 8 - Lecture 2

INTRODUCTION:

In this lecture we started the topic of number theory and discussed few fundamentals problems related to number theory as follows :

- Euclid's Algorithm to find GCD
- Extended Euclid's Algorithm
- Inverse modulo using GCD algorithm
- Public Key Cryptography
- Fermat's Little Theorem

Euclid's Algorithm :

This algorithm is used to compute the GCD of two numbers say a and b in an efficient manner without calculating the prime factors of the two numbers a and b .

In the school method , we would have to find the prime factors of both numbers a and b and b and then take the common factors among them to calculate GCD but this is not a very efficient algorithm since calculating the prime factors would take $O(sqrt(n))$ time which is very costly.

The euclid's algorithm only takes $O(loga)$ time to find the gcd of two numbers a and b assuming $a >= b$.

The algorithm is as follows :

```
function GCD(a,b):
if b = 0: return a;
return GCD(b,a%b)
```

Proof of Euclid's Algorithm:

We know that if two numbers a and b have gcd g , then g divides both a and b . Now g will also divide $(a-b,b)$.

Proof:

Every integer that divides a and b (say d) will also divide a-b since we cam wrote a and b as some number multiplied by d and hence d will divide a-b . This would hold true for every common factor of a and b and hence the gcd(a-b,b) ≥gcd(a,b) .

If we apply the same argument from the other side , we would get that gcd(a,b) ≥gcd(a-b,b) and hence from both these inequalities the gcd of a,b,a-b have to be equal .

Now we can recursively find $amodb$ by saying

$$gcd(a,b) = gcd(a-b,b) = gcd(a-2b,b) = gcd(a-3b,b)$$

and so on until we reach $amodb$.

Now since we can break down the problem of finding gcd of a and b as finding gcd of b and $amodb$ which is much easier to solve .

Time complexity:

If we look at two consecutive steps of our algorithm we will observe that:

$$gcd(a,b) = gcd(b,amodb) = gcd(amodb,bmod(amodb))$$

Now one important observation is that $amodb < a/2$ if a≥b .

To prove this , we can just naively consider both the cases where b is less than equal to a/2 and where b is greater than a/2 and we would observe that the value of modulo is in both cases less than half of a.

Therefore in every two steps we would be reducing the value of first argument of our GCD function by 2 since after two opeations we get $amodb$.

Hence we would only need to run this algorithm $2*log(a)$ times because after that number a would dilute .

Hence the time complexity of this algorithm would be around $O(loga)$.

Extended Euclid's Algorithm:

We know that :

If d divides a and b and $d = ax + by$ where x and y are integers then d is the gcd of a and b.

Proof:

We know d divides a and b , hence it will be less than equal to the gcd of a and b . Now since we know $(ax + by) mod g$ is zero where g is the gcd of a and b , then as ax+by=d , g also divides d and hence g≤d.

From both these inequalities d==g.

Hence d would be equals to the gcd of a and b .

So we can extend Euclid ALgorithm to also give us the integers x and y and not only the gcd as follows :

```
function Extended-Euclid(a,b):
if b=0: return (1,0,a)
(x',y',d) = Extended-Euclid(b,a%b)
return (y',x' - a/b.y',d)
```

Proof:

The base case is easy to prove when b==0 .

Now when b is not equal to 0 , we know the gcd will remain same and also we can see the following relation between the two corresponding relations we would get :

$$gcd(a, b) = gcd(b, amodb) = bx' + (a - [a/b]b)y' = ay' + b(x' - [a/b]y')$$

So we can find the values of x and y from the values of x' and y' where x=y' and $y = x' - [a/b]y'$.

Time complexity :

The time complexity would be same as that of the original euclid's Algorithm since we are just doing few more arithmetic opeations.

Hence complexity will be $O(loga)$

Modular Inverse :

For a given number N , x is said to be the multiplicative inverse of a if

$(x * a)modN == 1$

We can observe that this can hold true only if a and N are relatively prime since we can write a*x mod N as (ax+kN) for some integer k and hence we can write g=ax+kN where g is the gcd of a and N and so from here we can observe that the gcd will have to be 1 since kN mod N is anyways going to be zero .

We can find this multiplicative inverse using again Euclid's Algorithm in $O(n^3)$ time where n denotes the number of bits in N .

Hence we would just need to solve the Linear Diophantine equation and we would easily get the value of multiplicative inverse.

Public key Cryptography:

This is an algorithm where the keys used for encryption and decryption are different and the decryption key cannot be calculated from the encryption key. This allows us to keep a public/private key pair.

The idea is that there are two representations R1 and R2 of the key where R1 is private to the owner where only he will be able to make changes while R2 would belong to public which by name everyone can access.

The encryption operation E_k should be fast in R2 while the decryption operation that is E_k inverse should be very slow so that no one can decrypt the messsage other than the owner.

RSA algorithm:

This is a cryptography encryption decryption algorithm .

Here we pick two primes p and q and define a large integer N as N=p.q

Now for any e relatively prime to (p-1)(q-1) :

- The mapping $x -> x^e$ mod N should be a bijective mapping.
- The inverse mapping would be given by say if d be the inverse of e modulo (p-1)(q-1) . Then for all x belonging to whole numbers will N-1 , x^(ed) mod N == x mod N.

The value of d is only known o the owner and hence only he would be able to decrypt the message .

We can proof the above results using Fermat's Little Theorem :

Fermat's Little Theorem :

For any integer a and prime p which are coprime ,

$a^{p-1} mod p == 1 mod p$

RSA proof:

Since e is relatively prime to (p-1)(q-1) , e*d will be congruent to 1 modulo (p-1)(q-1) . Hence $ed = 1 + k * (p-1)(q-1)$.

Now it is given to us that N=pq. So $x^{p-1} mod p == 1 mod p$ and

$x^{p-1} mod q == 1 mod q$ from Fermat's Little Theorem and hence now if we apply Chinese Remainder Theorem , $x^{(p-1)(q-1)} mod pq == 1 mod pq$.

Therefore we can say $x^{ed} - x$ will be divisible by M .

Thus we have proved the correctness of RSA algorithm .