Theory of Numbers lecture 6

Raveesh Gupta

March 2019

Chapter 1

Introduction to Number Theory

Sport Programming:= Working with very numbers. In $DP, combinatorics$, etc problems, have a very large answer (¿1E18). For example:- $n!$ of 1E5 can be very very large. In competitive programming we usually find the remainder of the ans with some number (MOD).

Definition of $a|b$ "a divides b" it means if $a \epsilon Z$ and $b \epsilon Z$ then $b = a * c$ for some $c \epsilon$.

Definition of Division Algorithm.: Let $a\epsilon$, $b\epsilon$ There exist only q, r, $r\epsilon[0, b)$ such that $a = b * q + r$.

```
if a>= 0, then
q = a/b
r = a%b
```

But if $(a < 0)$ it will not work

```
if a = b*q + r, then

a/b = -((-a)/b)
a%b = -(-(a)%b)
```

1.1 Arithmetic Modulo Operations

Suppose According to some problem, we have some big number a and some big number b. Find $rem_{MOD}(a + b)$ by some modulo MOD. As discussed earlier, we don't want to store the whole a or the whole b,we want to store only $rem_{MOD}(a)$ and $rem_{MOD}(b)$. So the question is how to find $rem_{MOD}(a+b)$ if we don't know the whole numbers a and b. Its easy to see that..

$$rem_{MOD}(a) = rem_{MOD}(a - MOD),$$
$$rem_{MOD}(a) = rem_{MOD}(a - 2 * MOD),$$
$$rem_{MOD}(a) = rem_{MOD}(a + 2 * MOD)...etc,$$

So when we try to find $rem_{MOD}(a + b)$, we can subtract MOD from a and b enough number of times to replace $rem_{MOD}(a)$ by $rem_{MOD}(a - q * MOD)$ subsequently $rem_{MOD}(rem_{MOD}(a))$ and similarly replace $rem_{MOD}(b)$ by $rem_{MOD}(rem_{MOD}(b))$. Hence a and b are replaced by $rem_{MOD}(a)$ and $rem_{MOD}(b)$. So $rem_{MOD}(a + b)$ becomes

$$rem_{MOD}(A + B) = rem_{MOD}(rem_{MOD}(A) + rem_{MOD}(B))$$

For example:-

```
int deg3 = 1
FOR(i, 1,1E6+1) deg3 = (deg3 * 3)%MOD;
int deg2 = 1
FOR(i, 1,1E6+1) deg2 = (deg2 * 2)%MOD;
int deg38 = 1
FOR(i, 1,1E6+1) deg38 = (deg38 * 38)%MOD;
int ans = deg2 + deg3 - deg38;
if(ans<0) ans += MOD
```

Chapter 2

Fast Exponentiation and Prime Numbers

2.1 FastExpo

As we remember, it's impossible to calculate a^nin a straightforward way — we are to calculate it faster. so its faster to calculate a^{n-i} than a^n where $i > 0$, so its also faster to calculate $a^{n/k}$ where k¿1 than a^n. In fast exponentiation we usually take $k = 2$ and then store it in $a2 = a^{n/2}$. if n is even our $res = a2 * a2$ otherwise odd, $res = a2 * a2 * a$. But you can make it faster by changing k.

```
ll FastExpo (ll a, ll n, ll MOD) {
    if (n == 0)
        return 1%MOD;
    if (n % 2 == 1) {
        ll b = FastExpo(a, n - 1, MOD) %MOD;
        return (b*a)%MOD;
    }
    else {
        ll b = FastExpo (a, n/2, MOD);
        return (b*b)%MOD;
    }
}
```

2.2 Prime Check

Check if a number n is prime or not. Naive method is to go through all integers $i\epsilon[2, n-1]$. and check the condition $i|p$ is true or false this will take $O(n)$. But an important observation is that if $i|p$ is true then $(p/i)|p$ is always true. So we only need

$$i\epsilon[2, \sqrt{n}], (p/iisMINiffp/i == i)$$

2.3 Implementation

```
bool prime_check(ll x)
    for i:= 2...n^(1/2), do
        if(x%i==0), then
            return true
    return false
```

2.4 Sieve of Eratosthenes

Suppose now that we have a positive integer n and what to reveal for each number from 2 to n whether it's prime or not. One can check all the numbers independently, and the complexity will be $O(n * \sqrt(n))$. We will discuss more effective method which is called Sieve of Eratosthenes.

Instead of iterating through all the integers from $[2, n]$ and checking if a number is prime or not we Iterate through all the known primes and mark their multiples less than n as not prime. We generate these multiples in such a way that $pair(a, b)$ does not repeat as $pair(b, a)$ i.e pair(2, 3), pair(3,2) . For this we fix $a <= b$. Hence all the multiples start from $pr * pr...n$. Complexity: $(n * log(log(n)))$ as each composite number from 1 to n is visited at least once.

Uniqueness of pair(a, b) does not imply that a*b product won't repeat. i.e 3*4=12 gets repeated as 2*6= 12.

2.5 Implementation

```
for i:= 2....n, do
    prime[i]:= true

for i:= 2....n, do
    if(prime[i] is true), then
        for j:= i*i ...n, do
            prime[j] := false
```

2.6 Linear Sieve of Eratosthenes

This is an advanced version of the sieve of Eratosthenes which works in linear time but takes up more memory than it. In this algorithm we introduce lp_x, the least prime divisor of x.

$$lp[x]! = 0 <=> xisprime.$$

Next step is to find all products of x with $pr[j]$ such that $pr[j] <= lp[x]$ and $x * pr[j] <= N$. Assign $lp[x * pr[j]]$ as $pr[j]$. Those x for which $lp[x] = 0$ are prime numbers. Since each composite number from 1 to n is visited exactly once. Its running time is linear. We use the fact that $i = lp[i].x$ is an unique representation and the fact that $lp[i] <= lp[x]$. In comparison with the sieve of Eratosthenes $i = a * b$ is not a unique representation like 6*2 = 4*3.

2.7 Implementation

```
const int N = 10000000;
int lp[N+1];
vector<int> pr;

for (int i=2; i<=N; ++i) {
        if (lp[i] == 0) {
                lp[i] = i;
                pr.push_back (i);
        }
        for (int j=0; j<(int)pr.size() && pr[j]<=lp[i] && i*pr[j]<=N; ++j)
                lp[i * pr[j]] = pr[j];
}
```

Chapter 3

Division in Modular Arithmetic

3.1 Division by prime modulo

Suppose we have to divide a by b and $b|a$ is true. But usually in sport programming we need to perform (a'/b') where $a' = a\%MOD$ and $b' = b\%MOD$. Little Fermat's Theorem: if $a\epsilon[1, p-1]$ if p is prime, and p—a is false then $a^{p-1} = 1(MOD)$.So,

$$rem_{MOD}(a * b^{-1}) = rem_{MOD}(a * b^{MOD-2})$$

3.2 Euler's Function

3.3 Greatest Common Divisor

$$G(a, b) = Max(c)|(c|a \wedge c|b)$$

. if $c|a$ and $c|b$ then $c|(a-b)$. also $gcd(a, 0) = a$

$$gcd(a, b) = gcd(a-b, b) = gcd(a-b*q, b) = gcd(a\%b, b), q\epsilon Z.$$

3.4 Fundamental Theorem Of Arithmetic

The fundamental theorem of arithmetic, also called the unique factorization theorem or the unique-prime-factorization theorem, states that every integer greater than 1 either is a prime number itself or can be represented as the product of prime numbers and that, moreover, this representation is unique.

3.5 Diophantine equation

3.6 contractility lemma

we have 3 non-negative a, b, c Our lemma says if $gcd(a, m) = 1 m|(a * b)$ then $m|b$ (reduction of problem). using Bezout's identity

$$ax = my = 1$$

Chapter 4

Dirichlet Convolutions, Mobius Inversions formula

4.1 Problem: Given $n <= 1e5$ positive integers $\mathbf{a}_1, a_2$.

4.2

4.3