Number Theory

Bigmod

int bigmod **(**int a**,** int b**,** int mod**){**

**if** **(**b **==** 0**)** **return** 1**;**

int x **=** bigmod **(**a**,** b **/** 2**,** mod**);**

x **=** **(**x **\*** x**)** **%** mod**;**

**if** **(**b **&** 1**)** x **=** **(**x **\*** a**)** **%** mod**;**

**return** x**;**

**}**

Bitwise Sieve of Eratosthenes

const int N **=** 10000000**;**

int arr**[(**N **>>** 5**)** **+** 8**],** sq **=** sqrt**(**N**);**

void setBit **(**int**&** n**,** int i**){**

n **|=** **(**1 **<<** i**);**

**}**

bool checkBit **(**int**&** n**,** int i**){**

**return** **(**n **&** **(**1 **<<** i**));**

**}**

void sieve **(){**

**for** **(**int i **=** 4**;** i **<=** N**;** i **+=** 2**){**

setBit **(**arr**[**i **>>** 5**],** i **&** 31**);**

**}**

**for** **(**int i **=** 3**;** i **<=** sq**;** i **+=** 2**){**

**if** **(!**checkBit **(**arr**[**i **>>** 5**],** i **&** 31**)){**

**for** **(**int k **=** i **\*** i**;** k **<=** N**;** k **+=** i **+** i**){**

setBit **(**arr**[**k **>>** 5**],** k **&** 31**);**

**}**

**}**

**}**

**}**

Euler Totient Function

int phi **(**int n**){**

int ret **=** n**;**

**for** **(**int i **=** 2**;** i **\*** i **<=** n**;** i**++){**

**if** **(**n **%** i **==** 0**){**

**while** **(**n **%** i **==** 0**){**

n **/=** i**;**

**}**

ret **-=** ret **/** i**;**

**}**

**}**

**if** **(**n **>** 1**){**

ret **-=** ret **/** n**;**

**}**

**return** ret**;**

**}**

const int N **=** 10000000**;**

int phi**[**N **+** 7**],** mark**[**N **+** 7**];**

void sievePhi **(){**

**for** **(**int i **=** 1**;** i **<=** N**;** i**++){**

phi**[**i**]** **=** i**;**

**}**

**for** **(**int i **=** 2**;** i **<=** N**;** i**++){**

**if** **(!**mark**[**i**]){**

**for** **(**int j **=** i**;** j **<=** N**;** j **+=** i**){**

mark**[**j**]** **=** 1**;**

phi**[**j**]** **=** **(**phi**[**j**]** **/** i**)** **\*** **(**i **-** 1**);**

**}**

**}**

**}**

**}**

Greatest Common Divisor (GCD)

//On GNU compiler, use \_\_gcd()

int gcd **(**int a**,** int b**){**

**if** **(**a **%** b **==** 0**)** **return** b**;**

**return** gcd **(**b**,** a **%** b**);**

**}**

Lowest Common Multiple (LCM)

int lcm **(**int a**,** int b**){**

**return** **(**a **/** gcd **(**a**,** b**))** **\*** b**;**

**}**

Mobius Function

const int N **=** 10000000**;**

vector **<**int**>** primes**;**

int mu**[**N **+** 7**],** sq **=** sqrt**(**N**);**

void mobius **(){**

**for** **(**int i **=** 1**;** i **<=** N**;** i**++){**

mu**[**i**]** **=** 1**;**

**}**

**for** **(**int i **=** 0**;** i **<** **(**int**)** primes**.**size**()** **and** primes**[**i**]** **<=** sq**;** i**++){**

int x **=** primes**[**i**]** **\*** primes**[**i**];**

**for** **(**int j **=** x**;** j **<=** N**;** j **+=** x**){**

mu**[**j**]** **=** 0**;**

**}**

**}**

**for** **(**int i **=** 0**;** i **<** **(**int**)** primes**.**size**();** i**++){**

**for** **(**int j **=** primes**[**i**];** j **<=** N**;** j **+=** primes**[**i**]){**

mu**[**j**]** **\*=** **-**1**;**

**}**

**}**

**}**

Modular Inverse

//If mod is a prime, using Fermat's Little Theorem

int modInv **(**int a**,** int mod**){**

**return** bigmod **(**a**,** mod **-** 2**,** mod**);**

**}**

Sieve of Eratosthenes

const int N **=** 10000000**;**

int arr**[**N **+** 7**],** sq **=** sqrt**(**N**);**

void sieve **(){**

**for** **(**int i **=** 4**;** i **<=** N**;** i **+=** 2**){**

arr**[**i**]** **=** 1**;**

**}**

**for** **(**int i **=** 3**;** i **<=** sq**;** i **+=** 2**){**

**if** **(!**arr**[**i**]){**

**for** **(**int k **=** i **\*** i**;** k **<=** N**;** k **+=** i **+** i**){**

arr**[**k**]** **=** 1**;**

**}**

**}**

**}**

**}**