# GCD

/// return gcd(a, b)

template<typename T>

inline T gcd(T a, T b)

{

if(b == 0) return a;

return gcd(b, a%b);

}

# LCM

/// return lcm(a, b)

/// requirement gcd(a, b)

template<typename T>

inline T lcm(T a, T b)

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

# Sieve

#define maxL (mx >> 5) + 1

#define GET(x) (mark[x>>5] >> (x&31) & 1)

#define SET(x) (mark[x>>5] |= 1 << (x&31))

/// prime numbers are stored

/// in primes upto sqrt(mx)

const long long mx = 1e6 + 5;

unsigned int mark[maxL];

vector<int> primes;

void sieve()

{

primes.push\_back(2); SET(1);

const long long lim = sqrt(mx);

register long long i, j, k;

for(i=3; i<=lim; i+=2)

if(!GET(i)){

primes.push\_back(i);

for(j=i\*i, k=i+i; j<=lim; j+=k)

SET(j);

}

}

# Number of Divisors

/// return number of divisor

/// requirement prime numbers upto sqrt(x)

long long numDiv(long long x)

{

ll i = 0, p = primes[i], ans = 1;

while(p\*p <= x){

ll cnt = 0;

while(x%p == 0) x /= p, cnt++;

ans \*= (cnt + 1);

p = primes[++i];

}

if(x != 1) ans <<= 1;

return ans;

}

# Power (val ^ p) mod m

/// return val ^ p (% mod) O(log(p))

template <typename T>

T power(T val, T p, T mod){

val %= mod;

T result = 1;

while (p > 0) {

if (p & 1) result = (result \* val) % mod;

val = (val \* val) % mod;

p >>= 1;

}

return result;

}

# Power (val ^ p)

/// return val ^ p O(log(p))

template <typename T>

T power(T val, T p){

T result = 1;

while (p > 0) {

if (p & 1) result = (result \* val);

val = (val \* val);

p >>= 1;

}

return result;

}

# Sum of divisors

/// return sum of divisors

/// requirement prime numbers

/// upto sqrt(x)

long long sumDiv(long long x)

{

ll i = 0, p = primes[i], ans = 1;

while(p\*p <= x){

ll cnt = 1;

while(x%p == 0) x /= p, cnt++;

ans \*= ((power(p, cnt) - 1) / (p - 1));

p = primes[++i];

}

if(x != 1) ans \*= ((x\*x - 1) / (x - 1));

return ans;

}

# Euler Phi for X

/// return the number of coprime

/// with x (coprime = gcd(x, i) == 1 i<x;

/// requirement prime numbers upto sqrt(x)

long long eulerPhiX(long long x)

{

long long i = 0, p = primes[i], ans = x;

while(p \* p <= x){

if(x%p == 0) ans -= ans/p;

while(x%p == 0) x /= p;

p = primes[++i];

}

if(x != 1) ans -= ans/x;

return ans;

}

# Euler Phi for upto mx

/// store number of coprime numbers

/// for each index upto mx

/// based on sieve algorithm

/// const long long mx = 1e6 + 5;

long long phi[mx];

void eulerPhiSieve()

{

register long long i, j;

for(i=1; i<mx; i++) phi[i] = i;

for(i=2; i<mx; i++)

if(phi[i] == i)

for(j=i; j<mx; j+=i)

phi[j] -= phi[j]/i;

}

# Extended Euclid

/// return pair of integer,

/// value of x and y

/// which satisfy ax + by = gcd(a, b)

pii extendedEuclid(int a, int b){

if(b == 0) return pii(1, 0);

pii d = extendedEuclid(b, a%b);

return pii(d.second, d.first - d.second \* (a / b));

}

# Moduler Inverse

/// return a^(-1) % n

/// requirement extended Euclid theorem

int modInv(int a, int n){

pii ans = extendedEuclid(a, n);

return ((ans.first % n) + n) % n;

}