ICPC Assiut Community

Newcomers Training

Math

ICPC Assiut
community

Topics

- Modular Arithmetic
- Factorization
- Prime Factorization
- GCD
- LCM

Modular Arithmetic

- What is the **Modular Arithmetic** ?

- The Modular operation is (**%**)
- if we say : ***X % Y = Z*** (*X* and *Y* are positive)

  Z is **the smallest non negative number** such that ***X - Z*** is multiple of ***Y***

  it also mean that : ***Q * Y + Z = X*** (Q is any number)

  so that ***X - Z = Q * Y***

- **Example** : 17 % 4 = 1

  it mean 17 - 1 = 16, and 16 is divisible by 4

  also 4 * 4 + 1 = 17, 17 - 1 = 4 * 4 = 16

- **Try some examples in your paper to get more understand the operation**

Modular Arithmetic

- **Some Cases X % Y :**

1) **X > Y**
   Ex : 19 % 4 = 3

2) **X < Y**
   Ex : 4 % 6 = 4

3) **X = Y**
   Ex : 8 % 8 = 0

4) **X = 0**
   Ex : 0 % 5 = 0

5) **Y = 0**
   Ex : 5 % 0 = **RuntimeError**

Modular Arithmetic Properties

- **Cycling Pattern :**

Take %5 to every number in the given list :

List :	0	1	2	3	4	5	6	7	8	9
Mod:	0	1	2	3	4	0	1	2	3	4

- **Note what is the largest and the smallest number in the Mod**. (0, 4)
  - Another word, your range of numbers is [0, n - 1] (n = 5)

- **Note the Cycle of the numbers**

Modular Arithmetic Properties

- **Some Operations :**

- **(a + b) % c = ((a % c) + (b % c)) % c**

- **(a * b) % c = ((a % c) * (b % c)) % c**

  Ex:

  (4 * 8) % 3 = 32 % 3 = 2
  ((4 % 3) * (8 % 3)) % 3 = (1 * 2) % 3 = 2 % 3 = 2

- if (a) **is negative number**

- We add (c) until (a) become a positive number, then take it %c
- **a % c = ((a % c) + c) % c** it also work with the positive numbers

- **(a - b) % c = ((a % c) - (b % c) + c) % c**

Modular Arithmetic Code

```
// Cycling Pattern
#include<iostream>
using namespace std;
int main () {
    int a, b;
    a = 13;
    b = 5;
    for(int i=0;i<=a;i++){
        cout << i << " >> " << i % b << endl;
    }
    return 0;
}
```

```
Output:
0 >> 0
1 >> 1
2 >> 2
3 >> 3
4 >> 4
5 >> 0
6 >> 1
7 >> 2
8 >> 3
9 >> 4
10 >> 0
11 >> 1
12 >> 2
13 >> 3
```

Factorization

- What is the **Factor** ?

- The **Factor** is a positive number that is **<u>divisible</u>** by another number without remaining.

  Ex :

  if (***X % Y = 0)*** , so ***Y*** is a factor of ***X***

- Suppose the ***X = 36*** , What is the **Factors** of ***X*** ?

- The **Factors** of ***36*** : **{1, 2, 3, 4, 6, 9, 12, 18, 36}**

- In programming, it's simple to make a loop from ***1*** to ***X*** and print all the numbers that ***(X % i == 0)***

  But, Is that the best way !?

  What if ***X*** is a **<u>Large Number</u>** !?

Factorization Optimization

- **The simple way to optimize the Factors :**

- Suppose that ***N = 12***

  The marked numbers is the factors : **1 2 3 4 5 6 7 8 9 10 11 12**

- **More Tricky way to optimize the Factors :**

- we will loop from ***1 to sqrt(N)***, **Why !?**

  if (i == 1), the factors is **{ i, N / i }** = **{1, 12}**
  if (i == 2), the factors is **{ i, N / i }** = **{2, 6}**
  if (i == 3), the factors is **{ i, N / i }** = **{3, 4}**
  if (i == 4), the factors is {4, 3} **But we already have this factors**, so **we don`t need to continue the remaining numbers, because it will repeat the factors**

- Try on your paper if ***N = 36***

Factorization Code

```
// Factors of N
#include<iostream>
using namespace std;
int main () {
    int N = 36;
    // i <= sqrt(N) >> Square both sides >> i * i <= N
    // sqrt(N) take O(log(N)) , i * i take O(1), so it is faster
    for(int i = 1; i * i <= N; i++){
        if(N % i == 0){
            cout << i << " ";
            // if (i * i == N) you just need to print (i) not (N / i)
            if(i * i != N){
                cout << N / i << " ";
            }
        }
    }
    return 0;
}
```

Output:

1 36 2 18 3 12 4 9 6

Prime Factorization

- What is the **Prime Number** ?

- A number ***N*** is a **Prime** if it has **<u>only two factors</u>** {1, N}

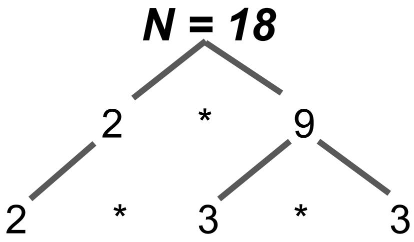
- Let's see what is the prime numbers from 1 to 7

(n = 1) >> **not prime** >> factors {1}
(n = 2) >> **prime** >> factors **{1, 2}**
(n = 3) >> **prime** >> factors **{1, 3}**
(n = 4) >> **not prime** >> factors {1, 2, 4}
(n = 5) >> **prime** >> factors **{1, 5}**
(n = 6) >> **not prime** >> factors {1, 2, 3, 6}
(n = 7) >> **prime** >> factors **{1, 7}**

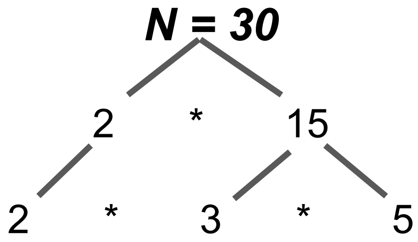
- **The negative numbers is not a prime**

Prime Factorization

- **Note : you can make any number by multiple any prime numbers.**
- Let's take some numbers and get their prime factors



The Prime factors is {2, 3}



The Prime factors is {2, 3, 5}

- **Steps :**

- Loop **i** from ***2 to sqrt(N)***
- while ***N % i == 0***, we divide ***N / i***
- **if (N == 1)**, so we have all the prime factors
- **if (N != 1)**, so **N is a prime number**, so that **it's a prime factor**

Prime Factorization Code

```
// Prime Factorization of N
#include<iostream>
using namespace std;
int main () {
    int N;
    cin >> N;
    // i <= sqrt(N) >> Square both sides >> i * i <= N
    // sqrt(N) take O(log(N)) , i * i take O(1), so it is faster
    for(int i = 2; i * i <= N; i++){
        while(N % i == 0){
            cout << i << " ";
            N /= i;
        }
    }
    if(N > 1){ // To Handle if N is a prime number
        cout << N << endl;
    }
    return 0;
}
```

Output:
2 2 3 3

Problems

- I will give you an array of numbers and for each a_i print YES if the number has an odd number of divisors , NO otherwise

Greatest Common Divisor (GCD)

- What is the **GCD** ?

- It is the ***greatest common factor*** of two numbers, by another word it is the ***largest number that divides them both***

Ex:

gcd(20, 15) = 5
gcd(8, 8) = 8
gcd(6, 3) = 3
gcd(11, 7) = 1
gcd(5, 0) = 5

- In programming, a simple way to get the **GCD** is to loop ***from 1 to min(a, b)***

Greatest Common Divisor (GCD)

- **Let's notice how we get the GCD mathematica :**

  Ex:

  the gcd(30, 45) = 15

  the prime factors of ***(30)*** : **2 * 3 * 5**

  the prime factors of ***(45)*** : **3 * 3 * 5**

- what is the common prime factors between (30) and (45) ?

  it is :

  3 : 1 times

  5 : 1 times

so the gcd(30, 45) = **the multiple of the common prime factors between them** = 3 * 5 = 15

Greatest Common Divisor (GCD)

- ## Another Ex :

  the gcd(75, 450) = 75

  the prime factors of ***(75)*** : **3 * 5 * 5**

  the prime factors of ***(450)*** : **2 * 3 * 3 * 5 * 5**

- what is the common prime factors between ***(75)*** and ***(450)*** ?

  it is :

  3 : 1 times

  5 : 2 times

so the gcd(75, 450) = **<u>the multiple of the common prime factors between them</u>** = 3 * 5 * 5 = 75

- **if there is no common prime factors so the GCD = 1**

GCD Code

```
#include<iostream>
using namespace std;
int gcd(int a, int b){
    // gcd(a, b) = gcd(b, a % b)
    // the condition stop if b = 0
    while(b != 0){
        int x = a;
        a = b;
        b = x % b;
    }
    return a;
}
int main () {
    int a = 20, b = 15;
    //cin >> a >> b;
    cout << gcd(a, b);
    return 0;
}
// In CodeBlocks there is a function __gcd() that take two integers and return the GCD
```

Practice Time !

You are given a river with a series of consecutive logs starting from the first log at the beginning of the river and extending to the opposite bank. The distances between the logs are provided. Your task is to jump from the first log to the last log.

Game Rules:

1. Before starting the game, you choose an initial jump power k.
2. In each second, you can either jump using your current power k, or you can wait to accumulate additional power by waiting for one second, which increases your jump power by k.
   - After waiting for t seconds, your total jump power becomes k*t.
3. You cannot jump on water directly; you must land on a log.
4. You cannot skip any log; you must jump to each log in order.

Goal:

- Choose the smallest possible value for k such that you can finish the game.
- Calculate the minimum number of seconds required to reach the last log while following all the rules.

Least Common Multiple (LCM)

- What is the **LCM** ?

\- it's the **least number** that is a multiple of ***A*** and ***B***, by another word it's ***divisible by A and B***

Ex :

- **lcm(3, 8) = 24** **(there is no number smaller than 24 that is divisible by 3 and 8)**
- **lcm(6, 18) = 18**
- **lcm(4, 4) = 4**
- **lcm(8, 0) = undefined**
- **lcm(0, 0)** = **undefined**

- In programming, a simple way to get the **LCM** is to loop ***from 1 until find the first number that is divisible by A and B***

Least Common Multiple (LCM)

- **Let's notice how we get the LCM mathematica :**

- The LCM is the multiple of the most frequency of the prime factors in both

  Ex :

  lcm(45, 30) = 90

  the prime factors of 45 : **3 * 3** * 5

  the prime factors of 30 : **2** * 3 *** 5**

- the most frequency prime factors are :

  2 : **1 times in 30** **>** 0 times in 45 **(we take 2 with 1 times)**

  3 : 1 times in 30 **<** **2 times in 45** **(we take 3 with 2 times)**

  5 : **1 times in 30** **=** **1 times in 45** **(you can take any of them)**

- So the **lcm(45, 30) = 2 * 3 * 3 * 5 = 90**

Least Common Multiple (LCM)

- **Another Ex :**

  the lcm(8, 3) = 24

  the prime factors of ***(3)*** : **3**

  the prime factors of ***(8)*** : **2 * 2 * 2**

- what is the most frequency prime factors between ***(3)*** and ***(8)*** ?

  it is :

  3 : 1 times

  2 : 3 times

- so the **lcm(3, 8) = 2 * 2 * 2 * 3 = 24**

GCD & LCM Relations

- **Suppose that you have two integer A, B**

- the prime factors of ***A*** = **2 * 2 * 3 * 3 * 5 * 7**
- the prime factors of ***B*** = **2 * 3 * 5 * 5 * 7 * 7**

the **gcd(A, B)** = <u>The common prime factors</u> = **2 * 3 * 5 * 7**
the **lcm(A, B)** = <u>The most frequency prime factors</u> = **2 * 2 * 3 * 3 * 5 * 5 * 7 * 7**

- **<u>NOTE :</u>**
  1) gcd(A, B) * lcm(A, B) = **2 * 2 * 2 * 3 * 3 * 3 * 5 * 5 * 5 * 7 * 7 * 7**
  2) A * B = **2 * 2 * 2 * 3 * 3 * 3 * 5 * 5 * 5 * 7 * 7 * 7**

- **From (1) and (2)**

gcd(A, B) * lcm(A, B) = A * B >>> **lcm(A, B) = (A * B) / gcd(A, B)**

LCM Code

```
#include<iostream>
using namespace std;
int gcd(int a, int b){
    // gcd(a, b) = gcd(b, a % b)
    // the condition stop if b = 0
    while(b != 0){
        int x = a;
        a = b;
        b = x % b;
    }
    return a;
}

int lcm(int a, int b){
    // gcd(a, b) * lcm(a, b) = a * b
    return (a * b) / gcd(a, b);
}

int main () {
    int a = 45, b = 30;
    //cin >> a >> b;
    cout << lcm(a, b) << endl;
    return 0;
}
// In CodeBlocks there is a function __gcd() that take two integers and return the
GCD
```

Output :
90

For more information about **Math Algorithms** visit this Link

Now it's time to practise and solve the problems of Arrays

Math - Geometry Sheet

Good luck <3