Number Theory 2

Sachin and Varun Problem

Problem Statement: Given two weights of **a** and **b** units, in how many different ways you can achieve a weight of **d** units using only the given weights? Any of the given weights can be used any number of times (including 0 number of times).

Explanation:

Let's take an example to understand the problem more clearly:

a = 2

b = 3

d = 7

So, one possible way could be **2 * 2 + 3 = 7**

Another could be **2 * 5 - 3 = 7.**

So, basically we have to find the solutions to a*x + b*y = d, where x and y are always greater than zero.

Additional Info: For the maximum value of **x**, we need to keep y as zero i.e, **x <= d/a**.

Similarly, **y <= d/b**.

Approach 1: One approach will be to iterate over all possible x and y values. Time complexity of this approach will be **O (d^2 / a * b).** Since in the question constraints,

we are given that in the worst case **a** would be equal to **b.** Hence the worst case time complexity would be **O (d^2 / b^2).**

Approach 2: Let us understand the equations a bit more.

$$a * x + b * y = d$$

$$a * x = d - b * y$$

We know that for one value of **y** there is at-most one **x** which will satisfy this equation. So, iterate over all values of **y** in the range and check if **d – b * y** is divisible by **a** or not.

Similarly we could also check that is **d – a * x** divisible by **b** or not as

$$b * y = d - a * x$$

Thus, the time complexity of this approach is **O(d/b)** or **O(d/a).** This is better than the previous solution's complexity but still not good enough to pass within the time limit.

Approach 3: Beginning of Optimized Approach

In the above approach, the value of **y** is changing. Hence, if we could find the smallest value for **y** for which we get our conditions satisfied, we can easily find out our next value of **y**. That is, we can add integer multiples of **a** to satisfy the conditions.

Let y_1 be the smallest non-negative number such that **d – b * y_1** is divisible by **a**. We can say that y_1 is the first value of **y** such that **d – b * y** is divisible by a.

If **d – b*y** is divisible by **a**, then **d – b*y + a** is also divisible by **a**.

If we have y_1, the next value of **y** will be $y_1 + a$. Let us prove this.

Proof: Let $y = y_1 + a$. Then,

$$d - b * y = d - b - (y_1 + a)$$

$$= d - b * y_1 - b * a$$

We know that $d - b * y_1$ is divisible by **a.** Additionally, **b * a** is also divisible by **a.** Hence we may conclude that $d - b * (y_1 + a)$ will also be divisible by **a.**

Thus the values of y will be $y_1, y_1 + a, y_1 + 2a, y_1 + 3a, y_1 + 4a, \ldots, y_1 + n * a,$ where

$$y_1 + n * a <= d/b$$

The value of **y** seems to be an Arithmetic Progression. So we can easily calculate the number of terms in the Arithmetic progression using $y_1 + n * a <= d/b$ equation.

$n = (d/b - y_1) / a.$

So the number of terms in the Arithmetic progression is equal to n + 1.

Now the only part that's left is to find the value of y_1. An obvious solution is to iterate over all values of **y** and break as soon as we find the first value such that **d – b * y** is divisible by **a**. Time complexity of this solution is **O(d/b)** in the worst case. But this solution is faster than previous solutions in other cases.

Approach 4: Optimisation over Approach 3

A small optimization is to divide both sides of the equation **a * x + b * y = d** by the GCD(a, b). Why ?? Think about it.

a * x + b * y = d is a linear Diophantine equation in two variables. The solution of this equation exists if and only if d is divisible by the GCD(a, b).

We can compute the first value of y_1 using Modulo Inverse (Extended Euclid).

We have, **a * x + b * y = d.**

y = (d – a*x) / b

Taking modulo **a** on both sides we get

y_1 = y % a = [(d – a*x) / b] % a

Now we need to check if **(d - b * y_1)** is divisible by a.

We can compute y_1 in **O(log(max(a, b)).** The number of terms used can be computed using the formula of the previous solution **(n = (d/b - y_1) / a)** in **O(1)**.

Hence, the overall time complexity of this solution is **O(log(max(a, b))).**

Code:

```
#include<bits/stdc++.h>

using namespace std;
typedef long long ll;

class Triplet{
public:
    ll gcd;
    ll x;
    ll y;
};
Triplet gcdExtendedEuclid(ll a,ll b){
    //Base Case
    if(b==0){
        Triplet myAns;
        myAns.gcd = a;
        myAns.x = 1;
        myAns.y = 0;
```

```
        return myAns;

    }
    Triplet smallAns = gcdExtendedEuclid(b,a%b);
    //Extended euclid says

    Triplet myAns;
    myAns.gcd = smallAns.gcd;
    myAns.x  = smallAns.y;
    myAns.y = (smallAns.x - ((a/b)*(smallAns.y)));
    return myAns;
}

ll modInverse(ll A, ll M)
{
    ll x = gcdExtendedEuclid(A, M).x;
    return (x % M + M) % M;
}

int main(){

    int t;
    cin>>t;
    while(t--){
        ll a,b,d;

        cin >> a >> b >> d;
        ll g = __gcd(a,b);
//Special Cases
        if(d%g){
            cout << 0 << endl;
            continue;
        }

        if(d == 0){
            cout << 1 << endl;
            continue;
        }
```

```
        a/=g;
        b/=g;
        d/=g;


        ll y1  = ((d%a) * modInverse(b,a) ) %a;
        ll firstValue = d/b;

        if(d < y1*b){
            cout << 0 <<endl;
            continue;
        }

        ll n = (firstValue - y1)/a;

        ll ans = n +1 ;
        cout << ans <<endl;

    }

    return 0;
}
```

Advanced GCD

Problem Statement: Write a more efficient way to calculate GCD(a, b).

$0 < b < 10^{250}$ and $0 < a < 40000$

Explanation: Our function to calculate GCD earlier was:

```
gcd (int a, int b) {
        If (b == 0) {
                return a;
        }
        return (b, a%b);
}
```

We know that **gcd(b, a)** is the same as **gcd(a, b%a)**.

Thus, suppose had we needed to calculate **gcd(10^{240}, 40),** we may write this as

gcd(40, 10^{240} % 40) ------ Eq1

Now, since **b** can be very large, it can't be stored in integers. So, we need to think of a different way. We can store **b** in a string. We need to figure out how to take the modulo of a string. Taking modulo with a small number like **a** would enable us to store the resultant in an integer variable, thus satisfying the usage of the above formula **Eq1.**

Important Note: **(a * b) % m** is same as **(a%m * b%m) % m.**

Keeping in mind the above note, we may break our string into effective integers and calculate modulo. Let us understand the process. Suppose we have 104. Let us deconstruct this integer.

104 = 10*10 + 4

If we are supposed to find 104 % m, we may write this like

((10*10) % m + (4 % m)) % m

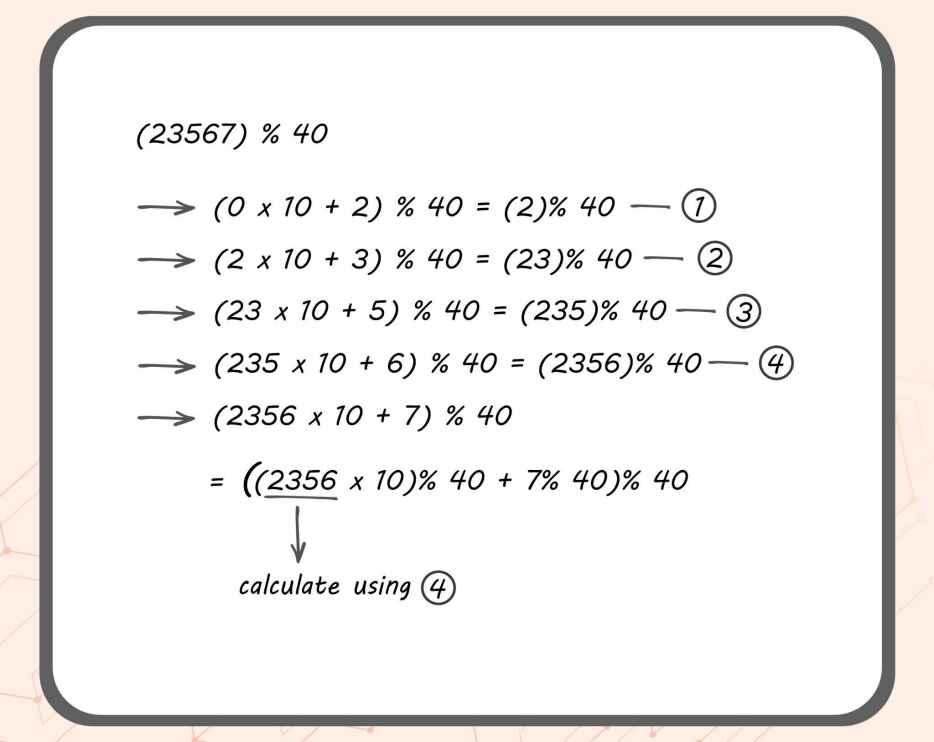
((100) % m + (4 % m)) % m

Similarly, we may deconstruct 100 like

(10 * 10) + 0

This 10 can be deconstructed into **1 * 10 + 0.**

Now, instead of breaking this big number into a small number, let's have a look at it in the reverse direction. Given a string, we shall take each digit separately from the start, and begin forming our big integer, all the while, performing modulo on it to maintain it inside an integer variable. Example for this is explained below in the image.



Extracting each digit from a string can be easily done using a for loop. In fact, you may use the same for loop for the purpose of building your larger integer.

```
for (int i = 0; i < n; i++)
{
    b = (10 * b + s[i] - '0') % a;
}
```

Let us now write the code for the same.

Code:

```
#include <bits/stdc++.h>
using namespace std;

int gcd(int a, int b)
{
    if (b == 0)
        return a;
    else
        return gcd(b, a % b);
}
int main()
{

    int t;
    cin >> t;

    while (t--)
    {
        int a;
        string s;
```

```
        cin >> a >> s;

        if (a == 0)
        {
            cout << s << "\n";
            continue;
        }

        int n = s.size();
        int b = 0;
        for (int i = 0; i < n; i++)
        {
            b = (10 * b + s[i] - '0') % a;
        }
        int ans = gcd(a, b);
        cout << ans << endl;
    }

    return 0;
}
```

Divisors of a Factorial

Problem Statement: We are given **N!** and we have to tell the total number of divisors for it.

Explanation:

Basic Approach: The basic approach to solve this problem would be to calculate the value of **N!** Then we traverse from 1 to N! and count the number of divisors.

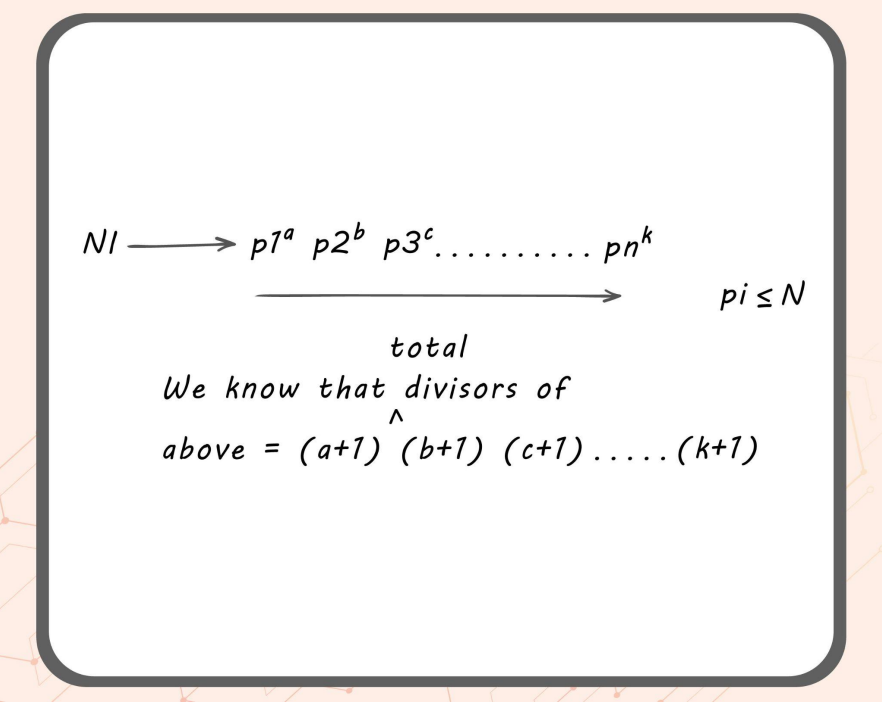
Time complexity of this will be **O(N!)** which will give time limit exceeded.

Optimized Approach:

We know that any number can be expressed in the form of prime numbers, that is,

$N \rightarrow 2^a 3^b 5^c ...$

For example, **$10 \rightarrow 2^1 5^1$**

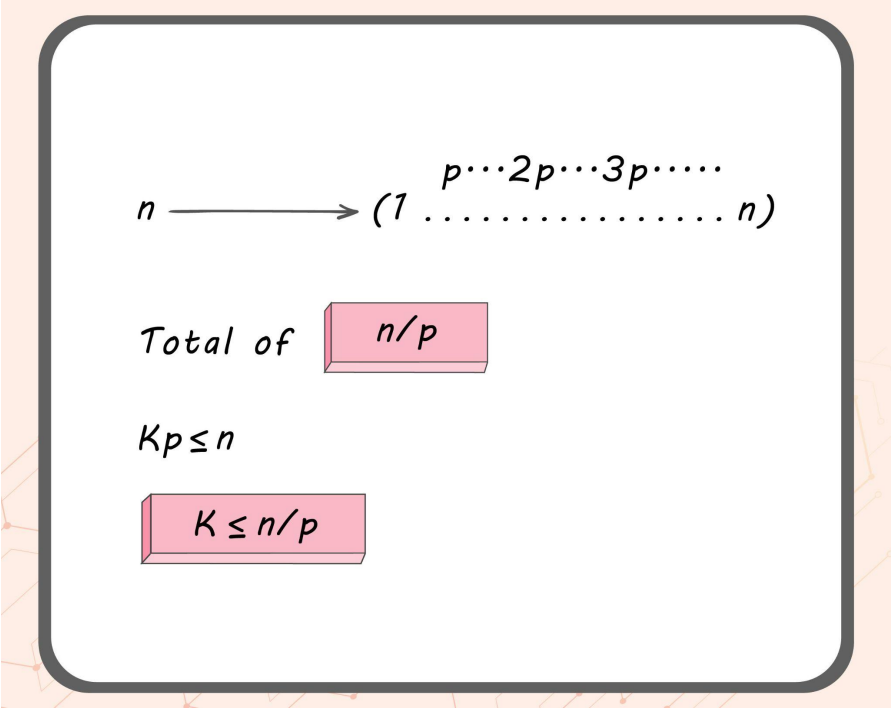


Now, how to express any number in the form of prime numbers ?

Suppose we are given 100! It is obvious that it won't contain any prime number that is greater than 100.

To simplify the problem even more, we can think of it in a way that we just need to calculate the powers of the prime numbers less than 100 that make up 100!

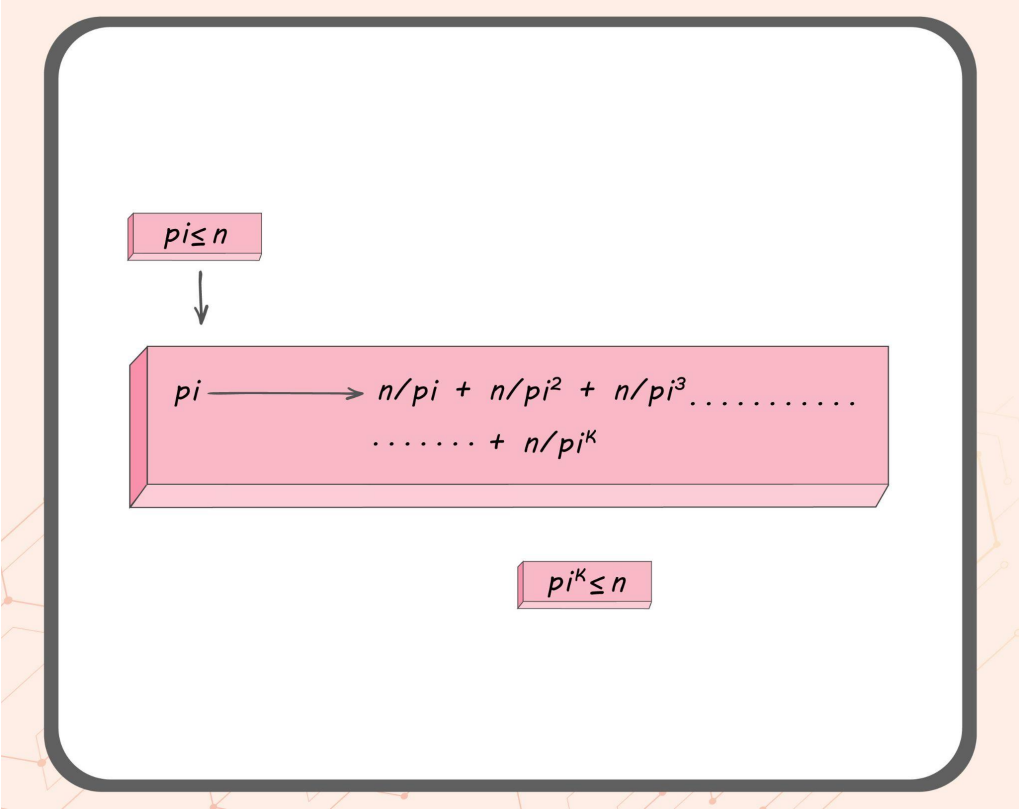
Now, we need to calculate the count of prime number p_i.



So let's say that **2** comes **n/p** times

Therefore $\mathbf{2^2}$ comes $\mathbf{n/p^2}$ times.

$\mathbf{2^3}$ comes $\mathbf{n/p^3}$ times and so on ...



We shall pre compute the values of primes using Sieve to save time. Let us now write the code for the same.

Code :

```
#include<bits/stdc++.h>
using namespace std;

#define MAX 500001
#define MOD 1000000007
#define pb push_back

typedef long long ll;

vector<int>* sieve(){
    bool isPrime[MAX];

    vector<int>* primes = new vector<int>();
```

```
    for(int i=2;i<=MAX;i++){
        isPrime[i] = true;
    }
    for(int i=2;i*i<=MAX;i++){

        if(isPrime[i]){
            for(int j=i*i;j<=MAX;j+=i){
                isPrime[j] = false;
            }
        }
    }

    primes->pb(2);
    for(int i=3;i<=MAX;i+=2){
        if(isPrime[i]){
            primes->pb(i);
        }
    }
    return primes;

}

ll divisors(int n,vector<int>* & primes){


    ll result = 1;
    for(int i=0;primes->at(i)<=n;i++){
        int k = primes->at(i);
        ll count = 0;
        while((n/k)!=0){
            count = (count + (n/k))%MOD;
            k = k*primes->at(i);
        }
        result = (result * ((count+1)%MOD))%MOD;
    }
    return result;
}
int main(){
    vector<int>* primes = sieve();
    int t;
```

```
    cin >> t;
    while(t--){
        int n;
        cin >> n;
        cout << divisors(n,primes) << endl;
    }
    return 0;
}
```