Number Theory 4

Counting Coprime Pairs:

Given a list of n positive integers, your task is to count the number of pairs of integers that are coprime (i.e., their greatest common divisor is one). n < 1e5, a[i] < 1e6.

2 6 12 9
2 6 12
3 6 9
6 - (6 12)
2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 1 0 1 -1 1 0 0 -1 1 -1 1 0 0

```
p[i] = f1 f2 f3
p[i] = 1 - p[f1] -p[f2] -p[f3]
```

If j is multiple of i:

```
p[j] = 1 - p[f1] -p[f2] -p[f3]-p[i]
     = 1 - p[f1] -p[f2] -p[f3] -1+ p[f1] +p[f2] + p[f3]
```

This problem can be solved by using Mobius Function based on the inclusion-exclusion principle, by selecting one of three operations of (-1,0,1) for each number.
This selection will be dependent on the number of unique prime divisors of each number.
The time complexity will be O(NlogN).

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

using namespace std;

#define int long long
int32_t main() {

    int n;
    cin >> n;
    vector<int> a(n);
    int mx = 0;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
        mx = max(mx, a[i]);
    }
    mx++;
    vector<int> m(mx + 1);
    for (auto el : a) {
        m[el]++;
    }
    int total = n * 1LL * (n - 1) / 2;
    int cnt = 0;
    vector<int> p(mx, 1);
    for (int i = 2; i < mx; i++) {
        int x = 0;
        for (int j = i; j < mx; j += i) {
            x += m[j];
            if (j != i) {
                p[j] -= p[i];
            }
        }
        cnt += p[i] * (x * 1LL * (x - 1)) / 2;
    }
    cout << total - cnt << "\n";
    return 0;
}
```

Extended Euclidean Algorithm: Ref: CP-algorithm

`gcd(a, b) = gcd(b, a%b) { gcd(a, 0) = a }`
`lcm (a, b) = a*b/gcd(a,b)`

To find a way to represent GCD in terms of a and b:

`ax + by = gcd(a, b)` (Bezout's Identity)
`ax + by = d`
d needs to be divisible by `gcd(a, b)` for **x and y integers** to satisfy this equation
`a = gk1`
`b = gk2`
`ax + by = g (xk1+ yk2)`

gcd(a, b) -> gcd(b, a%b)
(a, b) => (b, a%b)
ax+by = g -- (1)
Consider we are calculating for (x, y) at some step in recursion and we have called our function ex_gcd(b, a%b, x1, y1).
Then our values (x1, y1) should satisfy this invariant shown below:
=>bx1+ (a%b) y1 = g
=> bx1 + (a- floor(a/b)*b) y1 = g [As a%b = a - floor(a/b) *b]
=> ay1 + b*(x1- floor(a/b)y1) = g
Using this equation and eq(1)
x = y1
y = x1- floor(a/b)* y1
For Base condition : (g, 0) g*1 + 0*0 = g x= 1 ,y=0

```
int ex_gcd(int a, int b, int &x, int &y){
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = ex_gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return d;
}
```

```
ex_gcd(a, b, x0, y0)

ax0 + by0                       = g
ax0 + by0 + (k-k)ab/g           = g
a(x0 +kb/g) + b(y0 - ka/g)      = g
```

If `(x0 , y0)` satisfy the equation then `(x0+k(b/g) , y0 - k(a/g))` would also satisfy for any integer k. k belongs to integer.

Approximate Time Complexity Analysis

(a, b) => (b, a%b)
0 <= a%b < b
Now a%b can be any values between 0, 1 … , b-1
Let's Take a %b = b-1
Then in next step we will call gcd(b , b-1) = gcd(b-1, b%(b-1)) = gcd(b-1, 1) = gcd(1, 0)
So if we choose a%b = b-1 , then the process ends pretty soon.
Similarly if you take values like 0, 1 Then the process will end soon. For the worst case complexity, we can consider a%b to be around some value b/2. Then it would be
gcd(a, b) = gcd(b, b/2)
Then again this value b would be halved to become b/4 and so on.
In most of the cases it would end earlier only but for worst case analysis you can assume it to an upper bound of log(min(a, b))
Hence Time complexity : O(log (min(a, b)))

Question:

https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=4628

You all know Dr. Sheldon Cooper. Everything Sheldon gets, he must give back the same. He doesn't want to be in someone's debt.

Another friend of Sheldon, Penny has given him a very precious gift for his birthday. What penny didn't know, now Sheldon is going over his head to find out some gifts that will exactly match the price of Penny's gift. So my first task was to find out the price of the gift. With my super powerful sixth sense I found out that the price was P.

Now comes the hardest part. Sheldon needs to buy a gift which will cost exactly P. He went to the nearby shop for this. It is a crazy shop. Because it sells only three types of things: Rubik's cubes, mouth organs and chocolates. The prices of them are A, B, C accordingly. Now Sheldon faces a dilemma. What to buy and how many to buy? Because it turns out he can buy things costing P in various ways. For example if, $A = 202$, $B = 203$, $C = 200$ and $P = 606$, then are two different way to buy gifts costing exactly P. The first way is buying no Rubik's cubes, two mouth organs and one chocolate. The second way is buying three Rubik's cubes, no mouth organs and no chocolates.

Now Sheldon is asking for your help (because I am retired from contests). Given A, B, C and P, you need to find out in how many ways he can buy gifts costing exactly P.

Input

The first line will contain T ($T \leq 100$), the number of test cases. Next T lines each will have four integers, A, B, C and P ($0 < A, B, C, P \leq 100000000$) and $C / \gcd(A, B, C) \geq 200$, where $\gcd(A, B, C)$ means the greatest common divisor of A, B and C.

Output

For each case print '`Case` C`:` W' (without the quotes), where C is the case number and W is the number of ways Sheldon can buy gifts.

Sample Input

```
1
202 203 200 606
```

Sample Output

```
Case 1: 2
```

```
ax + by                 = d   (d is multiple of gcd(a,b) = g) - (1)
ax0+ by0                = g
ax0* d/g + by0 * d/g        = g*d/g
a (x0 * d/g) + b(y0*d/g)    = d
```

For eq (1) you have solution (x0 * d/g , y0 * d/g)

General Sol: `(x0 * d/g + kb/g , y0 * d/g - k*a/g)`

```
First Notice c >= 200 gcd(a, b, c) >= 200 as minimum value of gcd(a,
b, c) can be 1.
This implies that z can have at most 10^8/200 values for P-cz to
remain positive. So we can iterate through all z such that P-cz >0
So for any fixed z we have,
ax + by = P - cz= P1
ax + by = P1
The general form of solution for this equation would be:
(x0 * d/g + kb/g , y0 * d/g - k*a/g)
(xf +kb/g , yf - k*a/g)              ---- F1

x >=0 y>=0 x <= P y <= P
k >= -xf/(b/g)           and k <= yf/(a/g)
k <= (P-xf)/(b/g)         and k >= (yf-p)/(a/g)
```

So range wrt constraint on x is k belongs to [ceil(-xf/(b/g) , floor((P-xf) /(b/g))] and wrt constraint on y is k belongs to [ceil((yf -p)/(a/g)), floor(yf/(a/g))]

But we need to satisfy both equations so we will take the intersection of these two ranges and for all those values of k in the range when plugged in the form F1 would satisfy all constraints.

[2 , 4]

[3 , 5]

[3 , 4]

4 - 3 +1= 2

k>= -2.5

k >= -2

(int) -2.5 = -2

floor(-2.5) = -3

GCDMOD: https://www.codechef.com/problems/GCDMOD

GCD(a^n + b^n, |a-b|) = GCD(|a-b|, (a^n+ b^n) %|a-b|)

Now 1 <= a, b<= 10^12

|a-b| <= 10^12

So a%|a-b| would also be in range 10^12

Now in the binary exponentiation algorithm calculating a * a%m would result in an overflow.
So you need to write a function for multiplication similar to binary exponentiation. Now the values, in addition, will never overflow as 1e12 + 1e12 will be in range 1e12.
So you can use process like this;
a* b %m
(a+ a+ a .....+ a) %m (b times)
((a + a + a.. +a) %m+ (a+ a + a .....) %m) %m (b/2 times both if b is even)
and similarly repeating.
Max depth of recursion will be log (b)

Binary Mul:
4 * 13 (1 0 1 1) = 52

1st - > + 4 a = 4 b = 13
2nd - > + 0 a = 8 b = 6
3rd - > + 16 a = 16 b = 3
4th - > + 32 a = 32 b = 1

res = 52

```
int mul(int a, int b, int MOD) { // a = 1e12, b = 1e12, mod = 1e12
        int res = 0;
        a %= MOD;

        if (a == 0) return 0;
```

```
        while (b > 0)
        {
        if (b & 1)
        res = (res + a) % MOD;
        b = b / 2ll;
        a = (a + a) % MOD;
        }

        return res;
}
```

```
long long multiply_rec(long long a, long long b, long long MOD)
{
   if(b==0)
      return 0;
   else
   {
      long long z = multiply_rec(a,b/2,MOD);
      long long ans = (z+z)%MOD;
      if(b%2==1)
         ans = (ans+a)%MOD;
      return ans;
   }
}
```

Solution:

```
#include <bits/stdc++.h>
#define int long long

using namespace std;
const int MOD = 1e9 + 7;

int mul(int a, int b, int mod) { // a = 1e12, b = 1e12,
mod = 1e12
    int res = 0;
    a %= mod;

    if (a == 0)
        return 0;

    while (b > 0) {
        if (b & 1)
            res = (res + a) % mod;

        b = b / 2ll;
        a = (a + a) % mod;
    }

    return res;
}
```

```
int power(int x, unsigned int y, int mod) {
    int res = 1;
    x = x % mod;
    if (x == 0)
        return 0;

    while (y > 0) {
        if (y & 1)
            res = mul(res, x, mod);
        y = y >> 1;
        x = mul(x, x, mod);
    }
    return res;
}

void solve() {
    int a, b, n;
    cin >> a >> b >> n;

    if (a == b) { // gcd(an + bn, 0) = an + bn
        cout << (power(a, n, MOD) + power(b, n, MOD)) %
MOD << '\n';
        return;
    }

    // gcd ( an + bn, b - a);
    // gcd (an + bn % (b-a), b-a);

    if (b < a)
        swap(a, b);
```

```
    int first = 0;

    first += power(a, n, b - a);
    first += power(b, n, b - a);
    first %= (b - a);

    cout << __gcd(first, b - a) % MOD << '\n';
}

int32_t main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    int t = 1;
    cin >> t;
    while (t--) {
        solve();
    }
    return 0;
}
```

LCMSUM:

https://www.spoj.com/problems/LCMSUM/