Generate Divisors of N

We want to find the all divisors of N. We begin by writing the number as a product of prime factors:

$$N=p_1^{q1} \times p_2^{q2} \times p_3^{q3} \times \ldots\ldots\ldots\ldots \times p_k^{qk}$$

We store the prime number and their frequency. Then, recursively generate all possible combination and store the divisors.

```
#include <stdio.h>
#include <math.h>
#include <algorithm>

using namespace std;

#define SIZE_N 1000
#define SIZE_P 1000

bool flag[SIZE_N+5];
int primes[SIZE_P+5];

int seive()
{
	int i,j,total=0,val;

	for(i=2;i<=SIZE_N;i++) flag[i]=1;

	val=sqrt(SIZE_N)+1;

	for(i=2;i<val;i++)
		if(flag[i])
			for(j=i;j*i<=SIZE_N;j++)
				flag[i*j]=0;

	for(i=2;i<=SIZE_N;i++)
		if(flag[i])
			primes[total++]=i;

	return total;
}
```

```
int store_primes[100],freq_primes[100], store_divisor[10000],
Total_Prime, ans;
//Very Much Faster Divisor Function……
void divisor(int N)
{
	int i,val,count;

	val=sqrt(N)+1;
	Total_Prime=0;
	for(i=0;primes[i]<val;i++)
	{
		if(N%primes[i]==0)
		{
			count=0;
			while(N%primes[i]==0)
			{
				N/=primes[i];
				count++;
			}
			store_primes[Total_Prime]=primes[i];
			freq_primes[Total_Prime]=count;
			Total_Prime++;
			val=sqrt(N)+1;     // sqrt again for fast compute.
		}
	}
	if(N>1)
	{
		store_primes[Total_Prime]=N;
		freq_primes[Total_Prime]=1;
		Total_Prime++;
	}
}
void Generate(int cur,int num)
{
    int i,val;

    if(cur==Total_Prime)
    {
        store_divisor[ans++]=num;
    }
    else
    {
        val=1;
        for(i=0;i<=freq_primes[cur];i++)
        {
            Generate(cur+1,num*val);
            val=val*store_primes[cur];
        }
    }
}
int main()
```

```
{
	int total=seive();
	int n,i;

	while(scanf("%d",&n)==1)
	{
		divisor(n);
		ans=0;
		Generate(0,1);
		sort(&store_divisor[0],&store_divisor[ans]);

		printf("Total No of Divisors: %d\n",ans);
		for(i=0;i<ans;i++)
			printf("%d ",store_divisor[i]);
		printf("\n");
	}
	return 0;
}

/*
Input:
20
30
15
Output:
Total No of Divisors: 6
1 2 4 5 10 20
Total No of Divisors: 8
1 2 3 5 6 10 15 30
Total No of Divisors: 4
1 3 5 15
*/
```

Generate Number of Divisors [1 to N]

We know : $N=p_1^{q1} \times p_2^{q2} \times p_3^{q3} \times \ldots\ldots\ldots\ldots \times p_k^{qk}$

$=p_1^{q1} \times M$ [where $M= p_2^{q2} \times p_3^{q3} \times \ldots\ldots\ldots\ldots \times p_k^{qk}$]

Number of Divisor N is:

$$D(N)=(q1+1) \times (q2+1) \times (q3+1) \times \ldots\ldots\ldots\ldots \times (qk+1)$$

Number of Divisor M is:

$$D(M)= (q2+1) \times (q3+1) \times \ldots\ldots\ldots\ldots \times (qk+1)$$

Solve the simple math we find the relation:

$$D(N)=(q1+1) \times D(M)$$

Sample C++ Code:

```
int D[1000010];
void DivisorGenerate()
{
    int i,j,val,N,M,count;

    D[1]=1;
    for(i=2;i<=1000000;i++)
    {
        N=M=i;
        val=sqrt(N)+1;
        for(j=0;primes[j]<val;j++)
        {
            if(M%primes[j]==0)
            {
                count=0;
                while(M%primes[j]==0)
                {
                    M/=primes[j];
                    count++;
                }
                D[N]=(count+1)*D[M];
                break;
            }
        }
        if(M==N)            //Special Case if N equal prime
        {
            D[N]=2;
        }
    }
}
```

Generate Sum of Divisors [1 to N]

$$n = {p_1}^{e_1} * {p_2}^{e_2} * \cdots * {p_k}^{e_k}$$

$$= {p_1}^{e_1} * \mathrm{m} \qquad \mathrm{m} = {p_2}^{e_2} * \cdots * {p_k}^{e_k}$$

$$\sigma(n) = \frac{{p_1}^{e_1+1} - 1}{p_1 - 1} * \frac{{p_2}^{e_2+1} - 1}{p_2 - 1} * \cdots * \frac{{p_k}^{e_k+1} - 1}{p_k - 1}$$

$$\sigma(\mathrm{m}) = \frac{{p_2}^{e_2+1} - 1}{p_2 - 1} * \cdots * \frac{{p_k}^{e_k+1} - 1}{p_k - 1}$$

Replace The value:

$$\sigma(n) = \frac{{p_1}^{e_1+1} - 1}{p_1 - 1} * \sigma(\mathrm{m})$$

Generate Relative Prime [1 to N]

$$n = {p_1}^{e_1} * {p_2}^{e_2} * \cdots * {p_k}^{e_k}$$

$$= {p_1}^{e_1} * \mathrm{m} \qquad \mathrm{m} = {p_2}^{e_2} * \cdots * {p_k}^{e_k}$$

$$\phi(n) = ({p_1}^{e_1} - {p_1}^{e_1-1})\ ({p_2}^{e_2} - {p_2}^{e_2-1})\ \ldots\ldots\ ({p_k}^{e_k} - {p_k}^{e_k-1})$$

$$\phi(\mathrm{m}) = ({p_2}^{e_2} - {p_2}^{e_2-1})\ \ldots\ldots\ ({p_k}^{e_k} - {p_k}^{e_k-1})$$

Replace The value:

$$\phi(n) = ({p_1}^{e_1} - {p_1}^{e_1-1}) * \phi(\mathrm{m})$$

Number of Trailing Zeroes of N Factorial Base B

It is a usual exercise in Elementary Number Theory to compute the number of trailing zeroes in the base 10 expansion of the factorial of any integer. In fact, given any base b and any integer n, it is easy to compute the number of trailing zeroes of the base b expansion of $n!$. Namely, if we denote such number by $Z_b(n)$, we have.

Lemma 1.

(1) $Z_p(n) = \sum_{i \geq 1} \left[\frac{n}{p^i}\right] = \frac{n - \sigma_p(n)}{p-1}$, *where* $\sigma_p(n)$ *is the sum of the digits of the base* p *expansion of* n.

(2) $Z_{p^r}(n) = \left[\frac{Z_p(n)}{r}\right]$ *for every* $r \geq 1$.

(3) *If* $b = p_1^{r_1} \cdots p_s^{r_s}$, *then* $Z_b(n) = \min_{1 \leq i \leq s} Z_{p_i^{r_i}}(n)$.

C++ Code:

```
#include <stdio.h>
#include <math.h>
#include <algorithm>

using namespace std;

#define SIZE_N 1000
#define SIZE_P 1000

bool flag[SIZE_N+5];
int primes[SIZE_P+5];

int seive()
{
	int i,j,total=0,val;

	for(i=2;i<=SIZE_N;i++) flag[i]=1;

	val=sqrt(SIZE_N)+1;

	for(i=2;i<val;i++)
		if(flag[i])
			for(j=i;j*i<=SIZE_N;j++)
				flag[i*j]=0;
```

```
        for(i=2;i<=SIZE_N;i++)
                if(flag[i])
                        primes[total++]=i;

        return total;
}
int factors_in_factorial(int N,int p)
{
        int sum=0;

        while(N)
        {
                sum+=N/p;
                N/=p;
        }
        return sum;
}
int Trailingzero_Base_B(int N,int B)
{
        int i,ans,freq,power;

        ans=1000000000;
        for(i=0;primes[i]<=B;i++)
        {
                if(B%primes[i]==0)
                {
                        freq=0;
                        while(B%primes[i]==0)
                        {
                                freq++;
                                B/=primes[i];
                        }
                        power=factors_in_factorial(N,primes[i]);
                        ans=min(ans,power/freq);
                }
        }
        return ans;
}
int main()
{
        int total=seive();
        int i,N,B,zero;

        while(scanf("%d %d",&N,&B)==2)
        {
                zero=Trailingzero_Base_B(N,B);
                printf("%d\n",zero);
        }
        return 0;
}
```

Last Non Zero Digit of Factorial

Approach 1: Slow

We want to last non zero digit of Factorial N. Example: 5! = 120, here last non zero digit 2. How calculate? We know any factorial can be represented by prime factor.

$$N! = 2^{q2} \times 3^{q3} \times 5^{q5} \times 7^{q7} \times 11^{q11} \times 13^{q13} \ldots\ldots\ldots$$

But we know that, pair(5,2) create a trailing zero and power of 5(q5) is less than power of 2(q2). So we discard it for this problem

$$N!'' = 2^{q2-q5} \times 3^{q3} \times 7^{q7} \times 11^{q11} \times 13^{q13} \ldots\ldots\ldots$$

So that last non Zero digit:

$$L(N)=(N!'')\%10$$

$$=((2^{q2-q5}\%10) \times (3^{q3}\%10) \times (7^{q7}\%10) \times (11^{q11}\%10) \times (13^{q13}\%10) \ldots\ldots)\%10$$

Approach 2: Very Faster

L(0)=1
L(1)=1
L(2)=2
L(3)=6
L(4)=4
$L(N)= (2^{N/5} \times L(N/5) \times L(N\%5))\%10$

$= ((2^{N/5}\%10) \times L(N/5) \times L(N\%5))\%10$

```
int PTwo(int N)
{
    int T[]={6,2,4,8};
    if(N==0) return 1;
    return T[N%4];
}

int LastNZDigit(int N)
{
    int A[]={1,1,2,6,4};

    if(N<5) return A[N];

    return (PTwo(N/5)*LastNZDigit(N/5)*LastNZDigit(N%5))%10;
}
```

Greatest Common Divisor

- If d|a and d|b, then d|(ax+by) for any integers x and y GCD(a, b) = GCD(b, a)
- GCD(a, b) = GCD(-a, b)
- GCD(a, b) = GCD(|a|, |b|)
- GCD(a,0) = |a|
- GCD(a, ka) = |a|
- If *a* and *b* are any integers, not both zero, then GCD(a,b) is the smallest positive element of the set {ax + by : x, y in **Z**} of linear combinations of *a* and *b*.
- GCD(a, b) = GCD(b, a mod b)
- LCM(a,b)=(a*b)/GCD(a,b)

If $A= p_1^{x1} \times p_2^{x2} \times p_3^{x3} \times \ldots\ldots\ldots\ldots \times p_k^{xk}$

$B= p_1^{y1} \times p_2^{y2} \times p_3^{y3} \times \ldots\ldots\ldots\ldots \times p_k^{yk}$

$GCD(A,B)= p_1^{\min(x1,y1)} \times p_2^{\min(x2,y2)} \times \ldots\ldots\ldots\ldots \times p_k^{\min(xk,yk)}$

$LCM(A,B) = p_1^{\max(x1,y1)} \times p_2^{\max(x2,y2)} \times \ldots\ldots\ldots\ldots \times p_k^{\max(xk,yk)}$

Extended Euclidean Algorithm:-

Given **a** and **b**. Extended Euclidean Algorithm to find **x**, **y** and **gcd(a,b)**. Form This equation
ax+by=gcd(a,b)

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

Congruence

Definition: Two integers a and b are said to be congruence (or equivalent) module an integer m — written a ≡ b mod m — if m | (a−b).

Example: In class we said that 19≡2 mod 17, that 51 ≡ 0 mod 17, and that 10≡−10mod20.

1. **Reflexive:** for any integer *a* and any modulus *m*, we have $a \equiv a \bmod m$.
2. **Symmetric:** for any integers *a* and *b* and any modulus *m*, if $a \equiv b \bmod m$ then $b \equiv a \bmod m$.
3. **Transitive:** for any integers *a*,*b* and *c*, and any modulus *m*, if $a \equiv b \bmod m$ and $b \equiv c \bmod m$, then $a \equiv c \bmod m$.

Proof:

To prove the reflexive property, note that $a \equiv a \bmod m$ just means that we want to verify $m \mid (a-a) = 0$. We saw a while back, though, that any integer *m* divides 0, so this statement is valid.

To prove symmetry, we need to show that $a \equiv b \bmod m$ implies $b \equiv a \bmod m$. If $a \equiv b \bmod m$, though, the definition of modular congruence tells us that $m \mid (a-b)$, so that $mk=(a-b)$. But then we have $m(-k)=-(a-b)=b-a$, and so $m \mid (b-a)$. By the definition of modular congruence, we therefore have $b \equiv a \bmod m$.

Finally, for transitivity we are supposed to assume that $a \equiv b \bmod m$ and $b \equiv c \bmod m$, and somehow conclude that $a \equiv c \bmod m$. To prove this result, we note that the first two congruence conditions tells us that $m \mid (a-b)$ and $m \mid (b-c)$. Our result on divisibility of integral linear combinations, then, tells us that $m \mid (a-b)+(b-c)=a-c$. Hence the definition of modular congruence tells us that $a \equiv c \bmod m$.

The benefit of showing that modular congruence is an equivalence relation is that this tells us that congruence class partition the integers into distinct sets. For instance, when the modulus is 3, we know that every integer fits into one of the three collections

$$\{x \in \mathbb{Z} : x \equiv 0 \mod 3\} = \{x \in \mathbb{Z} : 3 \mid x-0\} = \{x \in \mathbb{Z} : x = 3k\} = \{\cdots, -6, -3, 0, 3, 6, \cdots\}$$
$$\{x \in \mathbb{Z} : x \equiv 1 \mod 3\} = \{x \in \mathbb{Z} : 3 \mid x-1\} = \{x \in \mathbb{Z} : x = 3k+1\} = \{\cdots, -5, -2, 1, 4, 7, \cdots\}$$
$$\{x \in \mathbb{Z} : x \equiv 2 \mod 3\} = \{x \in \mathbb{Z} : 3 \mid x-2\} = \{x \in \mathbb{Z} : x = 3k+2\} = \{\cdots, -4, -1, 2, 5, 8, \cdots\}$$

Coming up with a collection of integers which represent all these possible classes, then, is an important task. This leads to the following

Definition: A collection of integers is called a complete residue system for modulus m if every integer is congruent modulo m to exactly one element from the collection.

Inverse Modulo (Extended Euclidean Algorithm)

The **modular multiplicative inverse** of an integer *a* modulo *m* is an integer *x* such that

$$a^{-1} \equiv x \pmod{m}.$$

That is, it is the multiplicative inverse in the ring of integers modulo *m*. This is equivalent to

$$ax \equiv aa^{-1} \equiv 1 \pmod{m}.$$

The multiplicative inverse of *a* modulo *m* exists if and only if *a* and *m* are coprime (i.e., if gcd(*a*, *m*) = 1). If the modular multiplicative inverse of*a* modulo *m* exists, the operation of division by *a* modulo *m* can be defined as multiplying by the inverse, which is in essence the same concept as division in the field of reals.

The modular multiplicative inverse of *a* modulo *m* can be found with the extended Euclidean algorithm.

$$ax + by = \gcd(a, b)$$

where *a*, *b* are given and *x*, *y*, and gcd(*a*, *b*) are the integers that the algorithm discovers. So, since the modular multiplicative inverse is the solution to

$$ax \equiv 1 \pmod{m},$$

by the definition of congruence, $m \mid ax - 1$, which means that m is a divisor of $ax - 1$. This, in turn, means that

$$ax - 1 = qm.$$

Rearranging produces

$$ax - qm = 1.$$

with *a* and *m* given, *x* the inverse, and *q* an integer multiple that will be discarded. This is the exact form of equation that the extended Euclidean algorithm solves—the only difference being that gcd(*a*, *m*) = 1 is predetermined instead of discovered. Thus, *a* needs to be co-prime to the modulus, or the inverse won't exist. The inverse is *x*, and *q* is discarded.

C++ Code Inverse Modulo Using Extended Euclidean Algorithm:

```
#include <iostream>
#include <algorithm>
#include <stdio.h>

using namespace std;

int Extended_Euclidean(int a,int b,int &x,int &y)
{
    if(b==0)
    {
        x=1;y=0;
        return a;
    }

    int d;

    d=Extended_Euclidean(b,a%b,y,x);
    y=y-(a/b)*x;

    return d;
}
int Inverse_Modulo(int a,int m)
{
    int x,y,d;

    d=Extended_Euclidean(a,m,x,y);

    if(d==1)    return (x+m)%m; //Solution Exists
    return -1;  //No Solution
}
int main()
{
    int ans,a,m;
    while(scanf("%d %d",&a,&m))
    {
        ans=Inverse_Modulo(a,m);
        printf("%d\n",ans);
    }
    return 0;
}
//Input:
98 101
65 79
Output:
67
62
```

Inverse Modulo (Using Fermat Little Theorem)

The modular multiplicative inverse of an integer **a modulo m** is an integer **x** such that :

$$a^{-1} \equiv x \pmod{m}.$$

That is, it is the multiplicative inverse in the ring of integers modulo m. This is equivalent to

$$ax \equiv aa^{-1} \equiv 1 \pmod{m}.$$

The multiplicative inverse of a modulo m exists if and only if a and m are coprime (i.e., if gcd(a, m) = 1). Let's see, Calculate Modular Multiplicative Inverse using Fermat Little Theorem:

Fermat's little theorem states that if **m** is a prime and **a** is an integer co-prime to m, then:

$$a^{m-1} \equiv 1 \pmod{m}$$
$$a^{-1}a^{m-1} \equiv a^{-1} \pmod{m}$$
$$a^{m-2} \equiv a^{-1} \pmod{m}$$

That means: $(a^{-1} \bmod m) = (a^{m-2} \bmod m)$ **if m is prime**

How Can we calculate the value of $(a^{m-2} \bmod m)$? Using Big Mod Algorithm: Here Is C++ Code:

```
int BigMod(long long B,long long P,long long M)
{
        long long R=1;
        while(P>0)
        {
                if(P%2==1)
                {
                        R=(R*B)%M;
                }
                P/=2;
                B=(B*B)%M;
        }
        return R;
}

int Inverse_Modulo(int a,int m)
{
    return BigMod(a,m-2,m);
}
```

Wilson Theorem and Its Generalization

Wilson's theorem can be generalized to the following statement:

$$\forall n \in \mathbb{N} : (n-1)! \equiv \begin{cases} -1 \pmod{n} & \leftarrow n \text{ is prime} \\ 2 \pmod{n} & \leftarrow n = 4 \\ 0 \pmod{n} & \text{otherwise} \end{cases}$$

Gauss's generalization

The following is a stronger generalization of Wilson's theorem, due to Carl Friedrich Gauss:

$$\prod_{\substack{k=1 \\ gcd(k,m)=1}}^{m} k \equiv \begin{cases} 0 \pmod{m} & \text{if } m = 1 \\ -1 \pmod{m} & \text{if } m = 4,\ p^{\alpha},\ 2p^{\alpha} \\ 1 \pmod{m} & \text{otherwise} \end{cases}$$

Here **p** is an odd prime and **α** is positive integer.

Problem ID: TJU 3618

Divisibility Properties

Definition:. d | n means there is an integer k such that n = dk.

Theorem: (Divisibility Properties). If n, m, and d are integers then

the following statements hold:

1. n | n (everything divides itself)

2. d | n and n | m ⟹ d | m (transitivity)

3. d | n and d | m ⟹ d | an + bm for all a and b (linearity property)

4. d | n ⟹ ad | an (multiplication property)

Divisibility Rules

By Janine Bouyssounouse

The divisibility rules make math easier. Did you ever wonder how people could tell if something was divisible by a number just by looking at it? These rules are how they do it. Memorize a few simple rules and simplifying fractions and prime factorization will be so much easier.

Number	Divisibility Rule	Example
Two (2)	A number is divisible by two if it is **even**. Another way to say a word is even is to say it ends in 0, 2, 4, 6 or 8.	642 is divisible by two because it ends in a two, which makes it an even number
Three (3)	A number is divisible by three if the **sum of the digits adds up to a multiple of three**.	423 is divisible by three because 4 + 2 + 3 = 9. Since nine is a multiple of three (or is divisible by three), then 423 is divisible by three
Four (4)	A number is divisible by four if it is even and **can be divided by two twice**.	128 is divisible by four because half of it is 64 and 64 is still divisible by two
Five (5)	A number is divisible by five if it **ends in a five or a zero**.	435 is divisible by five because it ends in a five
Six (6)	A number is divisible by six if it is **divisible by both two and three**.	222 is divisible by six because it is even, so it is divisible by two and its digits add up to six, which makes it divisible by three
Nine (9)	A number is divisible by nine if the **sum of the digits adds up to a multiple of nine**. This rule is similar to the divisibility rule for three.	9243 is divisible by nine because the sum of the digits adds up to eighteen, which is a multiple of nine
Ten (10)	A number is divisible by ten if it **ends in a zero**. This rule is similar to the divisibility rule for five.	730 is divisible by ten because it ends in zero

Linear Diophantine Equations

Theorem 4.6 *Let a, b and c be integers with a and b not both zero.*

(i) *The linear Diophantine equation*

$$ax + by = c$$

has a solution if and only if $d = \gcd(a, b)$ divides c.

(ii) *If $d \mid c$, then one solution may be found by determining u and v such that $d = ua + vb$ and then setting*

$$x_0 = uc/d \qquad \text{and} \qquad y_0 = vc/d.$$

All other solutions are given by

$$x = x_0 + (b/d)t, \qquad y = y_0 - (a/d)t$$

for $t \in \mathbb{Z}$.

Finding all Solutions

We have (under the condition $d \mid c$) infinitely many solutions to our linear Diophantine equation. But could there be others about which we are currently unaware?

We shall need the following result in the course of our discussion.

Lemma 4.5 *Let r, s and t be integers and assume that r and s are coprime. If $r \mid st$, then $r \mid t$.*

Recall that to say r and s are coprime is to say that their greatest common divisor is 1.

PROOF: $\gcd(r, s) = 1$, so by part (ii) of Theorem 2.6, there exist integers u and v such that

$$ur + vs = 1.$$

Therefore

$$t = t(ur + vs) = utr + vst.$$

Now $r \mid st$ by assumption, while clearly r divides utr. Hence $r \mid (utr + vst)$, so $r \mid t$, as claimed. □

Now let us return to our linear Diophantine equation (4.1). Suppose we have fixed one solution (x_0, y_0) to (4.1). Let (x, y) be any other solution. So we have

$$ax + by = c = ax_0 + by_0.$$

Hence

$$a_1 d(x - x_0) = b_1 d(y - y_0).$$

Dividing by d gives

$$a_1(x - x_0) = b_1(y - y_0).$$

Now $a_1 = a/d$ and $b_1 = b/d$, so we have $\gcd(a_1, b_1) = 1$ (see Question 3 on Tutorial Sheet II). Hence a_1 and b_1 are coprime, while the above equation tells us

$$a_1 \mid b_1(y - y_0).$$

Hence Lemma 4.5 tells us that

$$a_1 \mid (y_0 - y).$$

This means that $y_0 - y = a_1 t$ for some $t \in \mathbb{Z}$. Substituting into the above equation gives

$$a_1(x - x_0) = b_1 a_1 t.$$

Therefore

$$x - x_0 = b_1 t.$$

Hence $x = x_0 + b_1 t$ and $y = y_0 - a_1 t$.

So we have shown that all solutions to (4.1) arise in the form we previously presented.

We summarise our finding as follows:

Example 4.8 *A customer bought some apples and some oranges, 12 pieces of fruit in total, and they cost him £1.32. If an apple costs 3p more than an orange, and if more apples than oranges were purchased, how many pieces of each fruit were bought?*

Solution: Let x be the number of apples bought. Then $12 - x$ is the number of oranges bought. Let y be the cost of an apple. Then $y - 3$ is the cost of an orange. We obtain the following equation

$$xy + (12 - x)(y - 3) = 132.$$

Therefore

$$xy + 12y - 36 - xy + 3x = 132$$

$$3x + 12y = 168$$

$$x + 4y = 56$$

We can solve this equation by inspection:

$$x = 56 - 4t, \quad y = t \qquad (\text{for } t \in \mathbb{Z}).$$

But we have further requirements: $6 < x < 12$, so

$$6 < 56 - 4t < 12.$$

Therefore

$$44 < 4t < 50$$

$$11 < t < 12\tfrac{1}{2}.$$

Hence $t = 12$. We deduce that

$$x = 8, \qquad y = 12.$$

So the customer bought 8 apples at 12p each and 4 oranges at 9p each.
(Finally check our working: $8 \cdot 12 + 4 \cdot 9 = 132$.)

Problem – Marbles (UVa 10090)

I have some (say, *n*) marbles (small glass balls) and I am going to buy some boxes to store them. The boxes are of two types:

Type 1: each box costs c_1 Taka and can hold exactly n_1 marbles
Type 2: each box costs c_2 Taka and can hold exactly n_2 marbles

I want each of the used boxes to be filled to its capacity and also to minimize the total cost of buying them. Since I find it difficult for me to figure out how to distribute my marbles among the boxes, I seek your help. I want your program to be efficient also.

Input

The input file may contain multiple test cases. Each test case begins with a line containing the integer n (1 <= n <= 2,000,000,000). The second line contains c_1 and n_1, and the third line

contains c_2 and n_2. Here, c_1, c_2, $n_1$and n_2 are all positive integers having values smaller than 2,000,000,000.

A test case containing a zero for *n* in the first line terminates the input.

Output

For each test case in the input print a line containing the minimum cost solution (two nonnegative integers m_1 and m_2, where m_i = number of *Type i* boxes required) if one exists, print "failed" otherwise.

If a solution exists, you may assume that it is unique.

Sample Input	**Sample Output**
43 1 3 2 4 40 5 9 5 12 0	13 1 failed

Solution:

$n_1m_1 + n_2m_2 = n$ ---------- (1)

Minimize -> $c_1m_1 + c_2m_2$

$g = gcd(n_1, n_2)$

$n_1m_1' + n_2m_2' = g$ ---------- (2)

Multiplying by (n/g) ->

$n_1m_1'' + n_2m_2'' = n$ ---------- (3)

From (1) and (3) ->

$m_1 = m_1'' + (n_2/g)\ t$

$m_2 = m_2'' - (n_1/g)\ t$

Here, t is an integer parameter.

From the conditions -> $m_1 \geq 0$, $m_2 \geq 0$, $n_1 > 0$, $n_2 > 0$,

$ceil(-m_1''\ g/n_2) \leq t \leq floor(m_2''\ g/n_1)$

$c = c_1m_1 + c_2m_2$

$= c_1m_1'' + c_2m_2'' + (c_1n_2/g - c_2n_1/g)\ t$

$= a + bt$

As this is a linear function, its minimum value will be on either of the boundaries.

Modular Linear Equations

- Find solutions to the equation –

  $ax = b \pmod n$ where $a > 0$ and $n > 0$

- This equation is solvable for the unknown x if and only if $gcd(a, n) \mid b$.
- This equation either has d distinct solutions modulo n, where $d = gcd(a, n)$, or it has no solutions.
- Let $d = gcd(a, n)$ and suppose that $d = ax' + ny'$ for some integers x' and y'. If $d \mid b$, then the equation $ax = b \pmod n$ has one of its solutions the value x_0, where $x_0 = x'(b/d) \bmod n$
- The other solutions of the equation would be –

  $x_i = x_0 + i\,(n/d)$ where $i = 0, 1, \ldots, d-1$

Algorithm:

```
MODULAR-LINEAR-EQUATION-SOLVER(a, b, n)
        (d, x', y') = EXTENDED-EUCLID(a, n)
        if d | b
                x0 = x'(b/d) mod n
                for i = 0 to d-1
                        print (x0 + i(n/d)) mod n
        else
                print "no solutions"
```

Chinese Remainder Theorem

- Given a set of simultaneous congruences

  $x = a_1 \pmod{n_1}$
  $x = a_2 \pmod{n_2}$
  …….
  …….
  $x = a_i \pmod{n_i}$

- For $i = 1, 2, \ldots. k$ and for which the n_i are pairwise relatively prime, the unique solution of the set of congruences is –

  $x = a_1 b_1 (N/n_1) + \ldots. + a_k b_k (N/n_k) \pmod{N}$

- Where $N = n_1 n_2 \ldots. N_k$

- And the b_i are determined from

  $b_i (N/n_i) = 1 \pmod{n_i}$

  $b_i M = 1 \pmod{n_i}$ $[M = (N/n_i)]$

  $b_i = M^{-1} \pmod{n_i}$ [Multiply by M]

  b_i is the Inverse of M.

Practice Problems:

LOJ: 1054, 1067, 1102, 1306, 1007, 1138, 1259, 1278, 1289, 1319, 1306

UVa: 10236, 10329, 10680, 10789, 10924, 10948, 11191, 11347, 11408, 11440, 10090,

TJU: 3618, 1233, 1375, 1476, 1528, 1637, 1698, 1730, 1748, 1991, 2218, 2308, 2502, 2520, 2526, 2601, 2648, 2658, 2859, 2901, 2916, 3043, 3076, 3150, 3232, 3237, 3238, 3259, 3261, 3262, 3288, 3293, 3467, 3483, 3496, 3599,3618

Prepared By:

Forhad Ahmed(Email: forhadsustbd@gmail.com)