**Name: Kshitija Shamrao Jadhav**

**PRN: 2019BTECS00053**

**BATCH: B6**

**Chinese Remainder Theorem**

We are given two arrays num[0..k-1] and rem[0..k-1]. In num[0..k-1], every pair is coprime (gcd for every pair is 1). We need to find minimum positive number x such that:

x % num[0] = rem[0],

x % num[1] = rem[1],

.......................

x % num[k-1] = rem[k-1]

Basically, we are given k numbers which are pairwise coprime, and given remainders of these numbers when an unknown number x is divided by them. We need to find the minimum possible value of x that produces given remainders.  
**Examples :** 

Input: num[] = {5, 7}, rem[] = {1, 3}

Output: 31

Explanation:

31 is the smallest number such that:

(1) When we divide it by 5, we get remainder 1.

(2) When we divide it by 7, we get remainder 3.

Input: num[] = {3, 4, 5}, rem[] = {2, 3, 1}

Output: 11

Explanation:

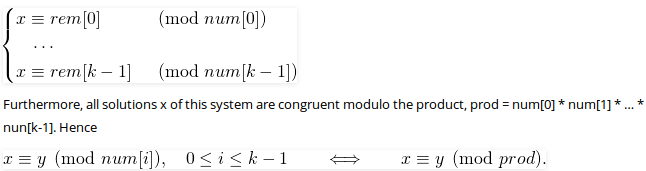
11 is the smallest number such that:

(1) When we divide it by 3, we get remainder 2.

(2) When we divide it by 4, we get remainder 3.

(3) When we divide it by 5, we get remainder 1.

***Chinese Remainder Theorem states that there always exists an x that satisfies given congruences.*** Below is theorem statement adapted from [wikipedia](https://en.wikipedia.org/wiki/Chinese_remainder_theorem" \t "_blank).   
Let num[0], num[1], …num[k-1] be positive integers that are pairwise coprime. Then, for any given sequence of integers rem[0], rem[1], … rem[k-1], there exists an integer x solving the following system of simultaneous congruences.



We are given two arrays num[0..k-1] and rem[0..k-1]. In num[0..k-1], every pair is coprime (gcd for every pair is 1). We need to find minimum positive number x such that:

x % num[0] = rem[0],

x % num[1] = rem[1],

.......................

x % num[k-1] = rem[k-1]

Basically, we are given k numbers which are pairwise coprime, and given remainders of these numbers when an unknown number x is divided by them. We need to find the minimum possible value of x that produces given remainders.  
**Examples :** 

Input: num[] = {5, 7}, rem[] = {1, 3}

Output: 31

Explanation:

31 is the smallest number such that:

(1) When we divide it by 5, we get remainder 1.

(2) When we divide it by 7, we get remainder 3.

Input: num[] = {3, 4, 5}, rem[] = {2, 3, 1}

Output: 11

Explanation:

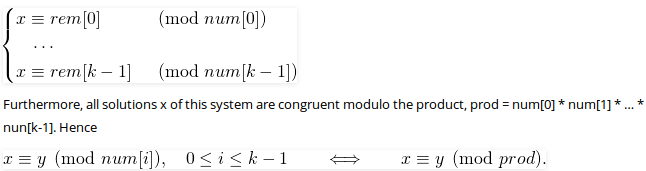
11 is the smallest number such that:

(1) When we divide it by 3, we get remainder 2.

(2) When we divide it by 4, we get remainder 3.

(3) When we divide it by 5, we get remainder 1.

***Chinese Remainder Theorem states that there always exists an x that satisfies given congruences.*** Below is theorem statement adapted from [wikipedia](https://en.wikipedia.org/wiki/Chinese_remainder_theorem" \t "_blank).   
Let num[0], num[1], …num[k-1] be positive integers that are pairwise coprime. Then, for any given sequence of integers rem[0], rem[1], … rem[k-1], there exists an integer x solving the following system of simultaneous congruences.



#include <bits/stdc++.h>

using namespace std;

// Returns modulo inverse of a

// with respect to m using

// extended Euclid Algorithm.

// Refer below post for details:

// https://www.geeksforgeeks.org/

// multiplicative-inverse-under-modulo-m/

int inv(int a, int m)

{

    int m0 = m, t, q;

    int x0 = 0, x1 = 1;

    if (m == 1)

        return 0;

    // Apply extended Euclid Algorithm

    while (a > 1) {

        // q is quotient

        q = a / m;

        t = m;

        // m is remainder now, process same as

        // euclid's algo

        m = a % m, a = t;

        t = x0;

        x0 = x1 - q \* x0;

        x1 = t;

    }

    // Make x1 positive

    if (x1 < 0)

        x1 += m0;

    return x1;

}

// k is size of num[] and rem[]. Returns the smallest

// number x such that:

// x % num[0] = rem[0],

// x % num[1] = rem[1],

// ..................

// x % num[k-2] = rem[k-1]

// Assumption: Numbers in num[] are pairwise coprime

// (gcd for every pair is 1)

int findMinX(int num[], int rem[], int k)

{

    // Compute product of all numbers

    int prod = 1;

    for (int i = 0; i < k; i++)

        prod \*= num[i];

    // Initialize result

    int result = 0;

    // Apply above formula

    for (int i = 0; i < k; i++) {

        int pp = prod / num[i];

        result += rem[i] \* inv(pp, num[i]) \* pp;

    }

    return result % prod;

}

// Driver method

int main(void)

{

    int num[] = { 3, 4, 5 };

    int rem[] = { 2, 3, 1 };

    int k = sizeof(num) / sizeof(num[0]);

    cout << "x is " << findMinX(num, rem, k);

    return 0;

}

OUTPUT:  
