| **Name** | **Ajinkya Sanatan Shelke** |
| --- | --- |
| **Roll No** | **331059** |
| **GR NO** | **21910680** |
| **Class** | **TY-A** |
| **Batch** | **A3** |
| **Subject** | **INS** |

**ASSIGNMENT NO. 2**

**Aim:** Develop and program in C++ or Java based on number theory such as Chinese remainder or Extended Euclidean algorithm. ( Or any other to illustrate number theory for security)

**Theory:**

The Chinese remainder theorem is a theorem which gives a unique solution to simultaneous linear congruences with coprime moduli. In its basic form, the Chinese remainder theorem will determine a number p bar that, when divided by some given divisors, leaves given remainders.

Process to solve systems of congruences with the Chinese remainder theorem:

For a system of congruences with co-prime moduli, the process is as follows:

* Begin with the congruence with the largest modulus, x ≡ ak(mod mk).
* Rewrite this modulus as an equation, x = mk.jk+ak, for some positive integer jk.
* Substitute the expression for x into the congruence with the next largest modulus, x ≡ ak(mod mk) ⟹ mk.jk+ak ≡ ak-1(mod mk-1).
* Solve this congruence for jk.
* Write the solved congruence as an equation, and then substitute this expression for jk into the equation for x.
* Continue substituting and solving congruences until the equation for x implies the solution to the system of congruences.

**Source Code :**

#include <bits/stdc++.h>

using namespace std;

int GCD(int A,int B) {

int R;

GCD:

if(A>=B) {

while(B!=0) {

R = A%B;

A = B;

B = R;

}

return A;

}

else {

int temp=A;

A = B;

B = temp;

goto GCD;

}

return 0;

}

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;

}

int findMinX(int m[], int a[], int k) {

// Compute product of all numbers

int M = 1;

for (int i = 0; i < k; i++)

M \*= m[i];

// Initialize result

int result = 0;

// Apply above formula

for (int i = 0; i < k; i++) {

int pp = M / m[i];

result += a[i] \* pp \* inv(pp, m[i]);

}

return result%M;

}

// Driver method

int main() {

int num, n, m[10], a[10];

int i = 0;

cout<<endl;

cin>>num;

while(i<num){

cout<<"\n\nEnter the number of congruence relations: ";

cin>>n;

cout<<"Enter the values of a: ";

for(int i=0;i<n;i++)

cin>>a[i];

cout<<"Enter the values of m: ";

for(int i=0;i<n;i++)

cin>>m[i];

cout<<"\nGiven congruence relations: "<<endl;

for(int i=0;i<n;i++){

cout<<"X = "<<a[i]<<"(mod "<<m[i]<<")"<<endl;

}

//check whether numbers are coprime with each other

for(int i=1;i<n;i++){

if(GCD(m[i],m[i-1]) == GCD(m[i],m[i+1])){

continue;

}

else{

cout << "elements in m[] are not pairwise coprime";

return 0;

}

}

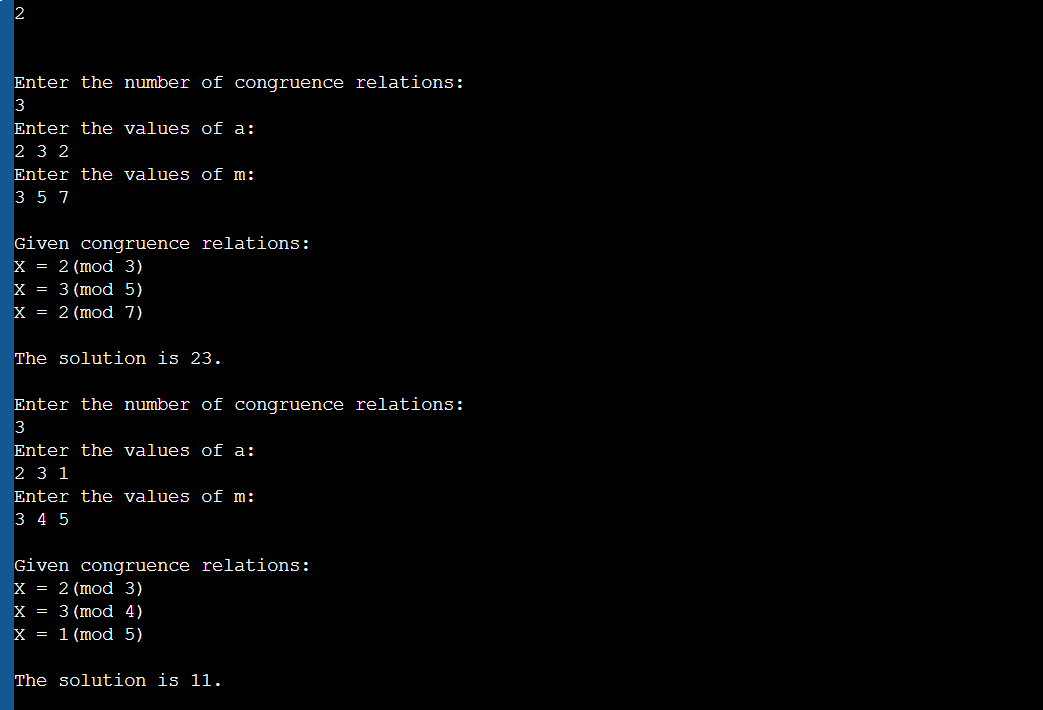
cout<<"\nThe solution is " << findMinX(m, a, n) <<".";

}

return 0;

}

**Output :**

****