Modular multiplicative inverse

Given two integers ‘a’ and ‘m’, find modular multiplicative inverse of ‘a’ under modulo ‘m’.

The modular multiplicative inverse is an integer ‘x’ such that.

```
a x ≡ 1 (mod m)
```

The value of x should be in {0, 1, 2, … m-1}, i.e., in the ring of integer modulo m.

The multiplicative inverse of “a modulo m” exists if and only if a and m are relatively prime (i.e., if gcd(a, m) = 1).

Examples:

```
Input:  a = 3, m = 11
Output: 4
Since (4*3) mod 11 = 1, 4 is modulo inverse of 3
One might think, 15 also as a valid output as "(15*3) mod 11"
is also 1, but 15 is not in ring {0, 1, 2, ... 10}, so not
valid.

Input:  a = 10, m = 17
Output: 12
Since (10*12) mod 17 = 1, 12 is modulo inverse of 3
```

We strongly recommend you to minimize your browser and try this yourself first.

Method 1 (Naive)

A Naive method is to try all numbers from 1 to m. For every number x, check if (a*x)%m is 1. Below is C++ implementation of this method.

```
// C++ program to find modular inverse of a under modulo m
#include<iostream>
using namespace std;

// A naive method to find modulor multiplicative inverse of
// 'a' under modulo 'm'
int modInverse(int a, int m)
{
    a = a%m;
    for (int x=1; x<m; x++)
       if ((a*x) % m == 1)
          return x;
```

```
}

// Driver Program
int main()
{
    int a = 3, m = 11;
    cout << modInverse(a, m);
    return 0;
}
```

Run on IDE

Output:

```
4
```

Time Complexity of this method is O(m).

Method 2 (Works when m and a are coprime)

The idea is to use **Extended Euclidean algorithms** that takes two integers 'a' and 'b', finds their gcd and also find 'x' and 'y' such that

```
ax + by = gcd(a, b)
```

To find multiplicative inverse of 'a' under 'm', we put b = m in above formula. Since we know that a and m are relatively prime, we can put value of gcd as 1.

```
ax + my = 1
```

If we take modulo m on both sides, we get

```
ax + my ≡ 1 (mod m)
```

We can remove the second term on left side as 'my (mod m)' would always be 0 for an integer y.

```
ax  ≡ 1 (mod m)
```

So the 'x' that we can find using Extended Euclid Algorithm is multiplicative inverse of 'a'

Below is C++ implementation of above algorithm.

```
// C++ program to find multiplicative modulo inverse using
// Extended Euclid algorithm.
#include<iostream>
using namespace std;

// C function for extended Euclidean Algorithm
int gcdExtended(int a, int b, int *x, int *y);

// Function to find modulo inverse of a
void modInverse(int a, int m)
{
```

```
    int x, y;
    int g = gcdExtended(a, m, &x, &y);
    if (g != 1)
        cout << "Inverse doesn't exist";
    else
    {
        // m is added to handle negative x
        int res = (x%m + m) % m;
        cout << "Modular multiplicative inverse is " << res;
    }
}

// C function for extended Euclidean Algorithm
int gcdExtended(int a, int b, int *x, int *y)
{
    // Base Case
    if (a == 0)
    {
        *x = 0, *y = 1;
        return b;
    }

    int x1, y1; // To store results of recursive call
    int gcd = gcdExtended(b%a, a, &x1, &y1);

    // Update x and y using results of recursive
    // call
    *x = y1 - (b/a) * x1;
    *y = x1;

    return gcd;
}

// Driver Program
int main()
{
    int a = 3, m = 11;
    modInverse(a, m);
    return 0;
}
```

Run on IDE

Output:

```
Modular multiplicative inverse is 4
```

Iterative Implementation:

```
// Iterative C++ program to find modular inverse using
// extended Euclid algorithm
#include <stdio.h>

// Returns modulo inverse of a with respect to m using
// extended Euclid Algorithm
// Assumption: a and m are coprimes, i.e., gcd(a, m) = 1
int modInverse(int a, int m)
{
    int m0 = m, t, q;
    int x0 = 0, x1 = 1;

    if (m == 1)
```

```
      return 0;

    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;
}

// Driver program to test above function
int main()
{
    int a = 3, m = 11;

    printf("Modular multiplicative inverse is %d\n",
          modInverse(a, m));
    return 0;
}
```

Run on IDE

Output:

```
Modular multiplicative inverse is 4
```

Time Complexity of this method is O(Log m)

Method 3 (Works when m is prime)

If we know m is prime, then we can also use **Fermats's little theorem** to find the inverse.

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

If we multiply both sides with a^{-1}, we get

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

Below is C++ implementation of above idea.

```
// C++ program to find modular inverse of a under modulo m
// This program works only if m is prime.
#include<iostream>
using namespace std;

// To find GCD of a and b
int gcd(int a, int b);

// To compute x raised to power y under modulo m
int power(int x, unsigned int y, unsigned int m);

// Function to find modular inverse of a under modulo m
// Assumption: m is prime
void modInverse(int a, int m)
{
    int g = gcd(a, m);
    if (g != 1)
        cout << "Inverse doesn't exist";
    else
    {
        // If a and m are relatively prime, then modulo inverse
        // is a^(m-2) mode m
        cout << "Modular multiplicative inverse is "
             << power(a, m-2, m);
    }
}

// To compute x^y under modulo m
int power(int x, unsigned int y, unsigned int m)
{
    if (y == 0)
        return 1;
    int p = power(x, y/2, m) % m;
    p = (p * p) % m;

    return (y%2 == 0)? p : (x * p) % m;
}

// Function to return gcd of a and b
int gcd(int a, int b)
{
    if (a == 0)
        return b;
    return gcd(b%a, a);
}

// Driver Program
int main()
{
    int a = 3, m = 11;
    modInverse(a, m);
    return 0;
}
```

Run on IDE

Output:

```
Modular multiplicative inverse is 4
```

Time Complexity of this method is O(Log m)

We have discussed three methods to find multiplicative inverse modulo m.

1) Naive Method, O(m)

2) Extended Euler's GCD algorithm, O(Log m) [Works when a and m are coprime]

3) Fermat's Little theorem, O(Log m) [Works when 'm' is prime]

Applications:

Computation of the modular multiplicative inverse is an essential step in RSA public-key encryption method.

References:

https://en.wikipedia.org/wiki/Modular_multiplicative_inverse

http://e-maxx.ru/algo/reverse_element

This article is contributed by **Ankur**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

2 Comments Category: Mathematical Tags: MathematicalAlgo

Related Posts:

- Wilson's Theorem
- Program for Rank of Matrix
- Primality Test | Set 3 (Miller–Rabin)
- Chinese Remainder Theorem | Set 2 (Inverse Modulo based Implementation)
- Euclid's lemma
- Chinese Remainder Theorem | Set 1 (Introduction)
- Compute nCr % p | Set 2 (Lucas Theorem)
- Compute nCr % p | Set 1 (Introduction and Dynamic Programming Solution)

2 Comments GeeksforGeeks 1 Login

Recommend 1 Share Sort by Newest

Join the discussion…

Umair Khan • 5 months ago

+1

• Reply • Share ›

lucy • 6 months ago

iterative method for extended euclidean algorithm..please check it
http://ideone.com/ODiFQL

• Reply • Share ›

Subscribe Add Disqus to your site Add Disqus Add Privacy

 Contact Us! About Us! Advertise with us!