```
//  Advanced Algorithms Lab
//  Devise an algorithm to solve a linear congruent equation that is helpful in finding modulo i
nverse of a number.

/* A number multiplied by its inverse equals 1

   Basic arithmetic : The inverse of a number A is 1/A since A * 1/A = 1

   All real numbers other than 0 have an inverse

   Multiplying a number by the inverse of A is equivalent to dividing by A */

/* Modular arithmetic does not have division operation

   The modular inverse of A (mod m) is A^-1 , ( where A^-1 in not 1/A ),
     such that (A * A^-1) ≡ 1 (mod m)
       or equivalently
     (A * A^-1) mod m = 1    */

/* An equation of the form ax = b (where a and b are real numbers) is called a linear equation
   And its solution x = b/a is obtained by multiplying both sides of the equation by 1/a
   a-1

   Linear congruence is : ax ≡ b ( mod m )

   ( If b = 1 , then Linear congruence is : ax ≡ 1 ( mod m )
     then x would be the modulo inverse of a  ) */

/* Proof :

   ax ≡ b ( mod m )
     Where gcd ( a , m ) = 1, that is a ⊥ m , and we seek the value of x (mod m)
       where , ⊥ means relatively prime

     then :
       ax = b + km
       So, ax - km = b

     Because ax ≡ b ( mod m ) ↔ b ≡ ax ( mod m ), because they are equivalent
       We can write ax - km = b as ax + km = b, with a change in sign for k

     If b = gcd(a,m) = 1 , we have: ax - km = 1

   Euclid's Extended Algorithm, then there are numbers which satisfy x and k :
     ax ≡ 1 ( mod m ) has solutions for x when a and m are relatively prime */

/* Only the numbers coprime to m have a modular inverse (mod m)
   coprime to m (numbers that share no prime factors with m) */

//   An inverse of a mod m exists iff gcd( a, m ) = 1 ; What if gcd ( a, m ) != 1 ?

/* Naive method of finding a modular inverse for A (mod m) is:
     step 1. Calculate A * B mod m for B values 0 through m-1
     step 2. The modular inverse of A mod m is the B value that makes A * B mod m = 1
```

```
   B mod m can only have an integer value 0 through m-1,
     so testing larger values for B is redundant

  Its mod operator : it cycles back : 0 to m-1 is repeated  */

/* Example: A=3 m=7
   Step 1. Calculate A * B mod m for B values 0 through m-1
     3 * 0 ≡ 0 (mod 7)
     3 * 1 ≡ 3 (mod 7)
     3 * 2 ≡ 6 (mod 7)
     3 * 3 ≡ 9 ≡ 2 (mod 7)
     3 * 4 ≡ 12 ≡ 5 (mod 7)
     3 * 5 ≡ 15 (mod 7) ≡ 1 (mod 7)   <------ FOUND INVERSE!
     3 * 6 ≡ 18 (mod 7) ≡ 4 (mod 7)

   Step 2. The modular inverse of A mod m is the B value that makes A * B mod m = 1
     5 is the modular inverse of 3 mod 7 since 5*3 mod 7 = 1 */

/* How about : for A=2 C=6 */

// Run time for Naive method of finding a modular inverse for A (mod m) is ?

// O ( m )

// Faster methods :
//   Extended Euclidean Algorithm - works when A and C are co prime, run time ?
//   Fermats's little theorem - works when C is prime, run time ?
//   Solving diophantine equation

/* Euclid's Extended Algorithm can be used to solve equations of the form:
    ax + by = 1

  Use Extended Euclids Algorithm to solve :
   56x + 93y = 1  , 56 x ≡ 1 (mod 93)  */

            // Extended Euclid's Algorithm

            // ALGORITHM Extended Euclids Algorithm(m, n)
            //   Computes gcd(m, n) , Bézout coefficients x and y such than mx+ny=gcd(m,n)
            //   Input: Two nonnegative, not-both-zero integers m and n
            //   Output: Greatest common divisor of m and n and Bézout coefficients
            //
            //     rPrevious  ← m ; // this is r0
            //     rPresent   ← n ; // this is r1
            //     sPrevious  ← 1 ; // this is s0
            //     sPresent   ← 0 ; // this is s1
            //     tPrevious  ← 0 ; // this is t0
            //     tPresent   ← 1 ; // this is t1
            //
            //     do until present remainder rPresent != 0
            //        rNext  ←  rPrevious - quotient * rPresent  // This provided
            //                              // 0 <= rNext < absolute value of rPresent
            //        sNext  ←  sPrevious - quotient * sPresent
```

```
        //      tNext  ←  tPrevious - quotient * tPresent
        //    done
        //
        //    // gcd  ←  rPrevious
        //    // x  ←  sPrevious
        //    // y  ←  tPrevious
        //   return gcd, x, y

/* Application:
    Computation of the modular multiplicative inverse is an essential step in
     RSA public-key encryption method  */

import java.util.Scanner;

class ModuloInverseOfNumberUsingLinearCongruentEquation
 {
  public static int findInverse( int a, int m )
   {
     int rPrevious = a ; // this is r0
     int rPresent  = m ; // this is r1
     int sPrevious = 1 ; // this is s0
     int sPresent  = 0 ; // this is s1
     int tPrevious = 0 ; // this is t0
     int tPresent  = 1 ; // this is t1

     int rNext =0;
     int sNext =0;
     int tNext =0;

     int quotient;
     int remainder;      // Or Reminder!

     System.out.println("\n Uisng Extended Euclid's Algorithm \n");

     System.out.println("\n What is to be done in loop : \n ");
     System.out.println("\n rNext  =  rPrevious - quotient * remainder;");
     System.out.println("\n sNext  =  sPrevious - quotient * sPresent;");
     System.out.println("\n tNext  =  tPrevious - quotient * tPresent;");

     System.out.println("\n\n Value in While loop\n \nQuotient \t\t Remainder \t\t s \t\t\t t");

     while ( rPresent != 0 )
     {
          quotient  = rPrevious / rPresent ;
          remainder = rPrevious % rPresent ;

          rNext  =  rPrevious - quotient * remainder;  // This provided
          // 0 <= rNext < absolute value of rPresent
          sNext  =  sPrevious - quotient * sPresent;
          tNext  =  tPrevious - quotient * tPresent;

          // print the values
          System.out.print("\n " + rPrevious + " / " + rPresent + " = " + quotient + " \t");
          System.out.print(" " + rPrevious + " mod " + rPresent + " = " + remainder + " \t ");
```

```
            System.out.print(" " + sPrevious + " - ( " + quotient + " * " +  sPresent + " ) = " + s
Next + " \t ");
            System.out.print(" " + tPrevious +" - ( " + quotient + " * " + tPresent + " ) = " + tNe
xt + " \t ");

            // Now update values for next ireartion
            // rNext , rPresent , nPrevious
            //        rPresent assign to rPrevious , again we are preparing for next iteration
            //        rNext assign to remainder , similarly assign s and t
            // Can we change the order of assignment

            rPrevious = rPresent;
            rPresent = remainder;        // Its remainder and not rNext

            sPrevious = sPresent;
            sPresent = sNext;

            tPrevious = tPresent;
            tPresent = tNext;

            // Console.ReadLine ();
        }

      System.out.println("\n sPrevious = " + sPrevious );
      System.out.println(" tPrevious = " + tPrevious );
      System.out.println(" rPrevious = " + rPrevious );

      System.out.println(" Extended Euclid : ( " + a + " * " + sPrevious + " ) + ( " + m + " * " +
tPrevious + " ) = " + ( a*sPrevious + m*tPrevious) );

      return sPrevious;
    }

   public static void main( String args[] )
    {
      int a; // read from user

      int m;

      System.out.print("\n To solve a linear congruent equation , hence finding modulo inverse
of a number using Extended Euclid's Algorithm, then gcd ( a , m ) = 1  ");

      System.out.print("\n Enter value of a in ax mod m , a = ");

      Scanner conin = new Scanner(System.in);
      a = conin.nextInt();

      System.out.print("\n Enter value of m in ax mod m , m = ");

      conin = new Scanner(System.in);
      m = conin.nextInt();

      /* Definition : If a' is a solution of the congruence ax ≡ 1 (mod m)
                  then a' is called the (multiplicative) inverse of a modulo m
```

```
                and we say that a is invertible

        The congruence ax ≡ 1 (mod m) has solutions if, and only if, gcd (a, m) | 1 ,
         i.e. gcd (a, m) = 1
        Thus a has an inverse modulo m iff a and m are coprime. */

      System.out.println("\n a = " + a + "\n m = " + m );

      System.out.println("\n Goal : find x such that : ax mod m  = 1 , that is "
                      + a + "x mod " + m + " = 1 " );

      // Assume entered numbers are relatively prime / co prime
      //   That is gcd ( a, m ) = 1

      int moduloInverse = findInverse( a, m );

      if( moduloInverse < 0 ) // How to prove the following ?
       {
        moduloInverse = moduloInverse * ( ( a * moduloInverse ) % m );
       }

      // Modulo inverse , may not be unique
      // If i is modulo inverse , then i + m , i + 2m , i + 3m . . . are also modular inverses
      System.out.println(" Inverse of a , in ax mod m , x = " + moduloInverse );
      System.out.println( " " + a + " * " + moduloInverse + " mod " + m + " = " + ( a * moduloI
nverse ) % m );

      return;
    }
  }
// References - Text books
// Introduction to the design & analysis of algorithms / Anany Levitin
// Introduction to Algorithms , Thomas H. Cormen , Charles E. Leiserson , Ronald L. Rivest
//                              Clifford Stein[For proofs]
// Data Structures and Algorithm Analysis in Java , Mark Allen Weiss
// Java : The Complete Reference, Herbert Schildt

//   Modular inverses (article) | Khan Academy
//   https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/
a/modular-inverses

//   Modular multiplicative inverse - GeeksforGeeks
//   http://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/

/* Related : Linear congruential method - algorithm for generating (pseudo)random numbers
   called - linear congruential generator, first described by Lehmer in 1951   */

      // Diophantine equation

      // Monte Carlo ? // Generate random numbers using Linear Cong
      //   Test for Modular inverse , like primality testing
```