МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**Арифметические операции в числовых полях**

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ»

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Серебрякова Алексея Владимировича

|  |  |  |
| --- | --- | --- |
| Преподаватель  профессор, д.ф.-м.н. | \_\_\_\_\_\_\_\_\_\_ | В. А. Молчанов |
|  | подпись, дата |  |

Саратов 2023

**1 Постановка задачи**

Цель работы:

* изучение основных операции в числовых полях и их программная реализация.

Задачи работы:

* изучить алгоритмы Евклида (обычный, бинарный и расширенный) вычисления НОД целых чисел и привести их программную реализацию;
* изучить алгоритм решения систем сравнений и привести его программную реализацию;
* изучить метод Гаусса решения систем линейных уравнений над конечными полями и привести его программную реализацию.

# 2 Теоретические сведения

**2. 1 Алгоритм Евклида**

Алгоритм Евклида вычисления наибольшего общего делителя целых чисел и состоит из следующих этапов. Положим и выполним последовательно деления с остатком на :

Так как остатки выполняемых делений образуют строго убывающую последовательность , то этот процесс обязательно остановится в результате получения нулевого остатка деления. Легко видеть, что . Значит, последний ненулевой остаток .

Сложность алгоритма Евклида: .

Описание алгоритма Евклида:

Вход: целые числа

Выход:

1. Положить .
2. Найти остаток от деления на .
3. Если , то положить . В противном случае положить и вернуться на шаг 2.
4. Результат:

**2.2 Расширенный алгоритм Евклида**

Расширенный алгоритм Евклида позволяет не только вычислять наибольший общий делитель целых чисел и , но и представлять его в виде для некоторых . Значения находятся в результате обратного прохода этапов алгоритма Евклида, в каждом из которых уравнение разрешается относительно остатка , который представляется в форме для некоторых . В результате получается следующая последовательность вычислений:

В правом столбце все элементы представляются в виде . Очевидно, что и выполняются равенства: . Отсюда последовательно получаются искомые представления всех элементов и, в частности, представление .

Сложность расширенного алгоритма Евклида: .

Описание расширенного алгоритма Евклида:

Вход: целые числа

Выход: .

1. Положить
2. Найти остаток от деления на и частное .
3. Положить и
4. Если , то положить . В противном случае положить и вернуться на шаг 2.
5. Результат:

**2.3 Бинарный алгоритм Евклида**

Бинарный алгоритм Евклида – это ускоренный алгоритм для поиска наибольшего общего делителя двух чисел. Он основан на следующих свойствах:

Сложность бинарного алгоритма Евклида: .

Описание бинарного алгоритма Евклида:

Вход: целые числа

Выход:

1. Положить
2. Если , то положить и перейти к шагу 9.
3. Если , то положить и перейти к шагу 9.
4. Если , то положить и перейти к шагу 9.
5. Если и четные, то положить и перейти к шагу 2.
6. Если четное и нечетное, то положить и перейти к шагу 2.
7. Если нечетное и четное, то положить и перейти к шагу 2.
8. Если и нечетные:
9. Если , то положить , и перейти к шагу 2.
10. Если , то положить , и перейти к шагу 2.
11. Положить
12. Результат: .

**2.4 Решение систем сравнений при помощи греко-китайской теоремы об остатках**

Греко-китайская теорема об остатках. Пусть попарно взаимно простые целые числа и . Тогда система линейных сравнений

имеет единственное неотрицательное решение по модулю

Пусть и . Тогда решение системы сравнений находится по формуле:

Алгоритм:

Вход: целые попарно взаимно простые числа и коэффициенты системы сравнений .

Выход: решение системы .

1. Положим
2. Для всех вычисляем .
3. Для всех вычисляем при помощи расширенного алгоритма Евклида .
4. Вычисляем решение системы сравнений
5. Результат:

Сложность вычислений равна , где .

**2.5 Решение системы сравнений при помощи алгоритма Гарнера**

Существует и другой способ решения системы, носящий название алгоритма Гарнера.

Пусть , числа попарно взаимно просты, и , . Тогда решение системы может быть представлено в виде

где , и числа вычисляется по формулам

Алгоритм:

Вход: целые попарно взаимно простые числа и коэффициенты системы сравнений .

Выход: решение системы .

1. Положим
2. Для всех и вычисляем при помощи расширенного алгоритма Евклида .
3. Положить и .
4. Для всех вычисляем .
5. Вычисляем решение системы сравнений .
6. Результат: .

Сложность вычислений равна , где .

**2.6 Решение систем линейных уравнений при помощи метода Гаусса**

Пусть – произвольное поле.

Системой линейных уравнений с неизвестными называется выражение вида

, где – линейные уравнения с неизвестными коэффициентами (первый индекс указывает номер уравнения, второй индекс — номер неизвестного) и свободными членами (индекс — номер уравнения). При этом числа называются также коэффициентами системы и – свободными членами системы.

Система называется однородной, если .

*Решением системы* (s) называется такой упорядоченный набор элементов , что при подстановке в уравнения значений получается верные равенства .

Лемма. Следующие элементарные преобразования сохраняют множество решений любой системы линейных уравнений, т.е. являются равносильными:

* удаление из системы тривиальных уравнений;
* умножение обеих частей какого-либо уравнения на одно и тот же ненулевой элемент поля;
* прибавление к обеим частям какого-либо уравнения системы соответствующих частей другого уравнения системы.

Метод решения системы заключается в равносильном преобразовании ее в систему линейных уравнений с противоречивым уравнением или в *разрешенную* систему линейных уравнений вида:

, где , так как в процессе элементарных преобразований исходной системы удаляются тривиальные уравнения. В этом случае неизвестные называются *разрешенными* (или *базисными*) и – свободными.

Система равносильна системе

которая *называется общим* решением исходной системы уравнений . Преобразование системы в равносильную ей разрешенную систему осуществляется по методу Гаусса с помощью последовательного выполнения *Жордановых преобразований*.

Матрицей системы (s) называется матрица

Алгоритм:

Вход: система линейный уравнений (𝑠) c 𝑛 неизвестными и m уравнениями.

Выход: общее решение системы (𝑠′′).

1. Строим матрицу
2. Положим .
3. В 𝑖-ой строке выбирается ненулевой коэффициент , 𝑖-ая строка умножается на элемент .
4. Прибавляем к остальных 𝑘-ым строкам новую 𝑖-ую строку, умноженную на коэффициент .
5. Если , то переход к следующему шагу, иначе и переход к шагу 3.
6. Удаляем из системы нулевые строки.
7. Если получили матрицу со строкой вида со значением , то алгоритм закончен, система не имеет решений.
8. Результат: строим общее решение системы по полученной матрице.

Сложность вычислений равна .

**3 Результаты работы**

**3.1 Псевдокод алгоритма Евклида**

Функция НОД (a, b)

Пока b ≠ 0

temp = b

b = a % b

a = temp

Конец Пока

Вернуть a

Конец Функции

**3.2 Псевдокод бинарного алгоритма Евклида**

Функция Бинарный\_НОД(a, b)

Если a == b

Вернуть a

Если a == 0

Вернуть b

Если b == 0

Вернуть a

Если a является четным числом и b является четным числом

Вернуть 2 \* Бинарный\_НОД(a / 2, b / 2)

Если a является четным числом и b является нечетным числом

Вернуть Бинарный\_НОД(a / 2, b)

Если a является нечетным числом и b является четным числом

Вернуть Бинарный\_НОД(a, b / 2)

Если a является нечетным числом и b является нечетным числом и a > b

Вернуть Бинарный\_НОД((a - b) / 2, b)

Если a является нечетным числом и b является нечетным числом и a < b

Вернуть Бинарный\_НОД((b - a) / 2, a)

Конец Функции

**3.3 Псевдокод расширенного алгоритма Евклида**

Функция Расширенный\_НОД(a, b)

u1 = 1;

v1 = 0;

u2 = 0;

v2 = 1;

Пока b != 0

d = a / b;

tmp = a % b;

a = b;

b = tmp;

tmp = u1 - u2 \* d;

u1 = u2;

u2 = tmp;

tmp = v1 - v2 \* d;

v1 = v2;

v2 = tmp;

Конец Пока

Вернуть (a, u1, v1)

Конец Функции

**3.4 Псевдокод решения системы сравнений по греко-китайской теореме об остатках**

Функция китайская\_теорема(коэффициенты, модули)

n = Длина(коэффициенты)

M = 1

x = 0

Для i от 0 до n – 1

M = M \* модули[i]

Для i от 0 до n – 1

Mᵢ = M / модули[i]

yᵢ = Расширенный\_НОД(Mᵢ, модули[i])[1]

x = x + коэффициенты[i] \* Mᵢ \* yᵢ

Вернуть x % M

Конец Функции

**3.5 Псевдокод решения системы сравнений по алгоритму Гарнера**

Функция Гарнера(коэффициенты, модули)

Для i от 0 до Длина(модули)

Для j от 0 до Длина(модули)

Если (i == j)

cij = 0

Иначе

cij = Расширенный\_НОД(модули[i], модули[j])

q = коэффициенты[0] mod модули[0]

Для k от 0 до Длина(модули)

q[i] = (коэффициенты[k] - q[0]) \* c0k mod модули[k]

Для i от 1 до k q[i] -= q[i]

q[i]\*= cik

q[i]= q[i] mod m[k]

mult = 1

U = q[0]

Для i от 0 до Длина(q)

mult \*= m[i – 1]

U = U + q.get(i) \* mult

U %= M

Вернуть U

Конец функции

**3.6 Псевдокод решения системы линейных уравнений по методу Гаусса**

Функция СЛУ(матрица A)

вектор b = последний столбец A

n = Размер(матрица A)[0] // Количество уравнений

m = Размер(матрица A)[1] // Количество переменных

Для i от 0 до n – 1

max\_row = i

Для k от i + 1 до n

Если Abs(A[k][i]) > Abs(A[max\_row][i])

max\_row = k

Конец Если

Конец Для

Если max\_row ≠ i

Обменять(матрица A[i], матрица A[max\_row])

Обменять(b[i], b[max\_row])

Конец Если

pivot = A[i][i]

Для j от i до m

A[i][j] = A[i][j] / pivot

Конец Для

b[i] = b[i] / pivot

Для k от 0 до n

Если k ≠ i

factor = A[k][i]

Для j от i до m

A[k][j] = A[k][j] - factor \* A[i][j]

Конец Для

b[k] = b[k] - factor \* b[i]

Конец Если

Конец Для

Конец Для

x = Создать\_Вектор(m)

Для i от n - 1 до 0 с шагом -1

x[i] = b[i]

Для j от i + 1 до m

x[i] = x[i] - A[i][j] \* x[j]

Конец Для

Конец Для

Вернуть x

Конец Функции

**4 Тестирование программы**

На рисунках 1-2 представлено тестирования работы программы для нахождения НОД

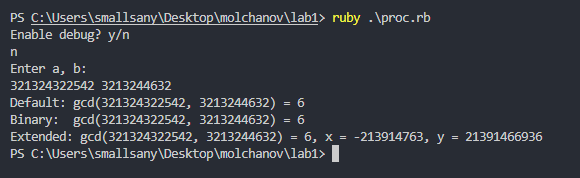


Рисунок 1 - Нахождение НОД

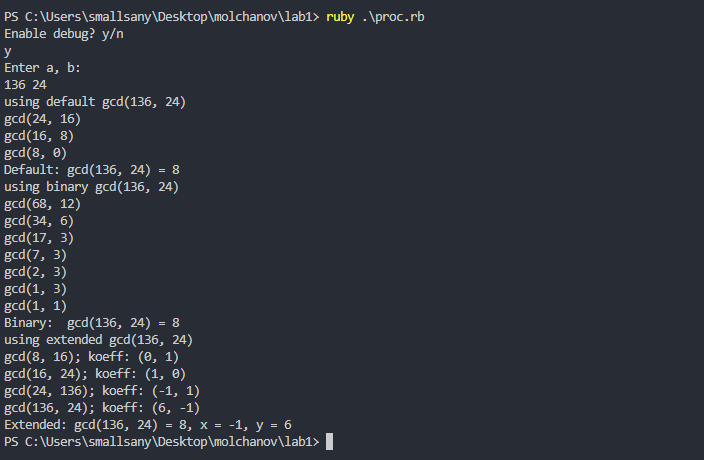


Рисунок 2 - Нахождение НОД (с отображением промежуточных шагов)

На рисунках 3 и 4 представлено тестирование работы программы для нахождения систем сравнений.

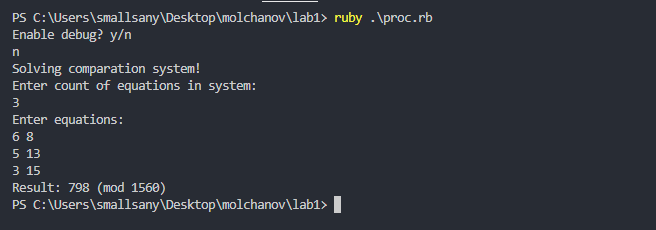


Рисунок 3 - Решение системы сравнений

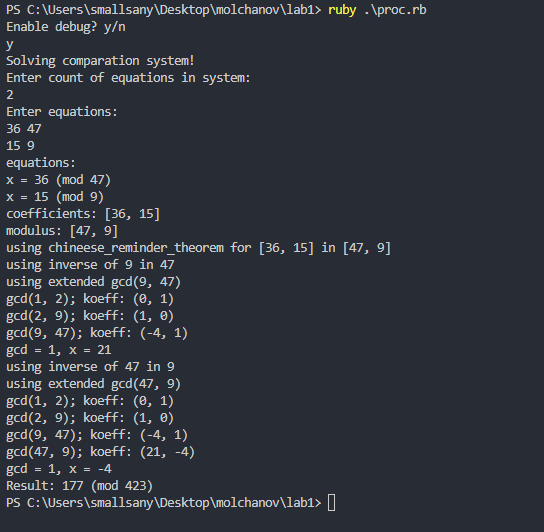


Рисунок 4 - Решение системы сравнений (с отображением промежуточных шагов)

На рисунке 5 представлено тестирование работы программы для решения системы линейных уравнений над конечным полем.

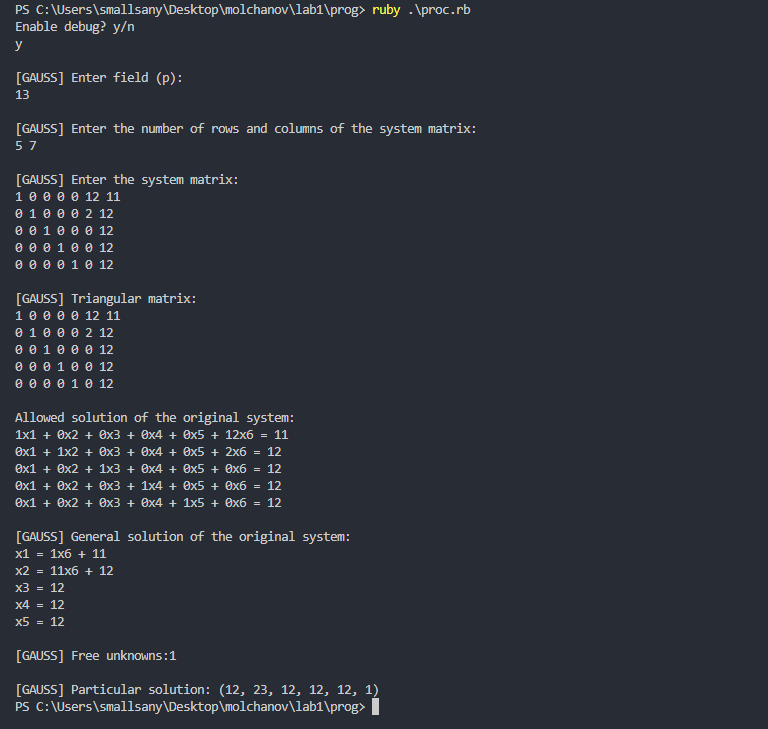


Рисунок 5 - Решение системы уравнений

ПРИЛОЖЕНИЕ А

Листинг программы methods.rb

class Methods

  def initialize(params = {})

    @debug\_mode = params.dig(:debug\_mode)

  end

  def gcd(a,b)

    puts "using default gcd(#{a}, #{b})" if @debug\_mode

    while b != 0

      remainder = a % b

      a = b

      b = remainder

*#debug*

      sleep 0.5 if @debug\_mode

      puts "gcd(#{a}, #{b})" if @debug\_mode

    end

    return a

  end

  def gcd\_bin(a,b)

    puts "using binary gcd(#{a}, #{b})" if @debug\_mode

    shift = 0

    while a != b

      if a % 2 == 0 && b % 2 == 0

        a = a / 2

        b = b / 2

        shift += 1

      elsif a % 2 == 0

        a = a / 2

      elsif b % 2 == 0

        b = b / 2

      elsif a > b

        a = (a-b)/2

      else

        b = (b-a)/2

      end

*#debug*

      sleep 0.5 if @debug\_mode

      puts "gcd(#{a}, #{b})" if @debug\_mode

    end

    return a \* (2 \*\* shift)

  end

  def gcd\_ext(a, b, first = true)

    puts "using extended gcd(#{a}, #{b})" if @debug\_mode && first

    if a == 0

      return b, 0, 1

    else

      res, x, y = gcd\_ext(b%a, a, false)

*#debug*

      sleep 0.5 if @debug\_mode

      puts "gcd(#{a}, #{b}); koeff: (#{x}, #{y})" if @debug\_mode

      return res, y - (b / a) \* x, x

    end

  end

  def inverse(a, md)

    puts %{using inverse of #{a} in #{md}} if @debug\_mode

    gcd, x, \_ = gcd\_ext(a, md)

    puts %{gcd = #{gcd}, x = #{x}} if @debug\_mode

    if gcd != 1

      raise "\nNo inverse element exists\n"

    else

      return x % md

    end

  end

  def chineese\_reminder\_theorem(coefficients = [], modulus = [])

    if coefficients.empty? || modulus.empty?

      raise "Not enough data!"

    end

    puts %{using chineese\_reminder\_theorem for #{coefficients} in #{modulus}} if @debug\_mode

    x = 0

    fact = modulus.reduce(:\*)

    coefficients.zip(modulus).each do |a, m|

      ci = fact / m

      ci\_inv = inverse(ci, m)

      x += a \* ci \* ci\_inv

    end

    x %= fact

    return {x:x, md: fact}

  end

  def exea(a, b)

    return [0, 1] if a % b == 0

    x, y = exea(b, a % b)

    [y, x - y \* (a / b)]

  end

  def mult\_row\_to\_num(row, num, field)

    row.map { |el| (el \* num) % field }

  end

  def add\_rows(row1, row2, field)

    row1.each\_with\_index.map { |el, i| (el + row2[i]) % field }

  end

  def del\_zero\_rows(matrix)

    matrix.reject! { |row| row.all?(&:zero?) }

  end

  def swap\_columns(matrix, col)

    (col + 1...matrix[col].size - 1).each do |i|

      if matrix[col][i] != 0

        matrix.each { |row| row[col], row[i] = row[i], row[col] }

        return

      end

    end

  end

  def gauss(matrix, field)

    matrix.each\_with\_index do |row, i|

      swap\_columns(matrix, i) if row[i] == 0

      rev\_el = exea(row[i], field)[0]

      matrix[i] = mult\_row\_to\_num(row, rev\_el, field)

      matrix.each\_with\_index do |row2, j|

        next if i == j

        matrix[j] = add\_rows(row2, mult\_row\_to\_num(matrix[i], -row2[i], field), field)

      end

    end

    del\_zero\_rows(matrix)

    have\_solution = !matrix.any? { |row| row.last != 0 && row.take(matrix.size).all?(&:zero?) }

    return matrix, have\_solution

  end

  def get\_matrix(matrix)

    matrix.each { |row| puts row.join(' ') }

  end

  def get\_ans(input, field)

    matrix = input[0]

    boo = input[1]

    if !boo

      puts "\n[GAUSS] There are no solutions!"

      return

    end

    puts "\nAllowed solution of the original system:"

    matrix.each do |row|

      row.each\_with\_index do |el, j|

        if j == row.size - 2

          print "#{el}x#{j + 1} = "

        elsif j == row.size - 1

          puts el

        else

          print "#{el}x#{j + 1} + "

        end

      end

    end

    puts "\n[GAUSS] General solution of the original system:"

    matrix.each\_with\_index do |row, i|

      print "x#{i + 1} = "

      (i + 1...row.size - 1).each do |j|

        if matrix[i][j] != 0

          matrix[i][j] = -matrix[i][j] + field

          print "#{matrix[i][j]}x#{j + 1} + "

        end

      end

      puts matrix[i].last

    end

    print "\n[GAUSS] Free unknowns:"

    vals = gets.chomp.split(',').map(&:to\_i)

    res = []

    matrix.each do |row|

      ans = 0

      (matrix.size...row.size - 1).each { |j| ans += vals[j - matrix.size] \* row[j] }

      ans += row.last

      res << ans

    end

    res.concat(vals)

    print "\n[GAUSS] Particular solution: ("

    res.each\_with\_index do |el, i|

      if i == res.size - 1

        print "#{el})"

      else

        print "#{el}, "

      end

    end

  end

end

ПРИЛОЖЕНИЕ Б

Листинг программы proc.rb

require './methods.rb'

puts %{Enable debug? y/n}

@debug\_mode = gets.strip == 'y'

@methods = Methods.new({debug\_mode: @debug\_mode})

def many\_gcds

  puts %{Enter a, b:}

  a, b = gets.strip.split.map(&:to\_i)

  puts %{Default: gcd(#{a}, #{b}) = *#{@methods.gcd(a, b)}}*

  puts %{Binary:  gcd(#{a}, #{b}) = *#{@methods.gcd\_bin(a, b)}}*

  res = @methods.gcd\_ext(a, b)

  puts %{Extended: gcd(#{a}, #{b}) = *#{res[0]}, x = #{res[1]}, y = #{res[2]}}*

end

def solve\_comparation\_system

  puts %{Solving comparation system!}

  equations = []

  coefficients = []

  modulus = []

  puts %{Enter count of equations in system:}

  n = gets.strip.to\_i

  puts %{Enter equations:}

  n.times do

    equation = gets.strip.split.map(&:to\_i)

    a, b = equation[0], equation[1]

    equations.append(%{x = #{a} (mod #{b})})

    coefficients.append(a)

    modulus.append(b)

  end

*#debug*

  if @debug\_mode

    puts %{equations:}

    equations.map {|eq| puts eq}

    puts %{coefficients: #{coefficients}}

    puts %{modulus: #{modulus}}

  end

  res = @methods.chineese\_reminder\_theorem(coefficients, modulus)

  puts %{Result: #{res[:x]} (mod #{res[:md]})}

end

def solve\_gauss\_system

  puts "\n[GAUSS] Enter field (p):"

  field = gets.chomp.to\_i

  puts "\n[GAUSS] Enter the number of rows and columns of the system matrix:"

  rows, cols = gets.chomp.split.map(&:to\_i)

  puts "\n[GAUSS] Enter the system matrix:"

  matrix = []

  rows.times do

    row = gets.chomp.split.map(&:to\_i)

    matrix << row

  end

  triangular, status = @methods.gauss(matrix, field)

  puts "\n[GAUSS] Triangular matrix: "

  @methods.get\_matrix(matrix)

  @methods.get\_ans([matrix, status], field)

end

*#1*

*# many\_gcds*

*#2*

*# solve\_comparation\_system*

*#3*

solve\_gauss\_system