Оглавление

[1 Теоретическая часть 3](#_Toc152437139)

[1.1 -метод Полларда 3](#_Toc152437140)

[1.1.1 Алгоритм -метода Полларда 3](#_Toc152437141)

[1.1.2 Псевдокод -метода Полларда 4](#_Toc152437142)

[1.2 ()-метод Полларда 4](#_Toc152437143)

[1.2.1 Алгоритм ()-метода Полларда 5](#_Toc152437144)

[1.2.2 Псевдокод ()-метода Полларда 5](#_Toc152437145)

[1.3 Метод цепных дробей 6](#_Toc152437146)

[1.3.1 Алгоритм метода цепных дробей 8](#_Toc152437147)

**Цель работы** – изучить основные методы факторизации целых чисел и их программную реализацию.

**Задачи работы**:

* Изучить -метод Полларда разложения целых чисел на множители и привести его программную реализацию;
* Изучить -метод Полларда разложения целых чисел на множители и привести его программную реализацию;
* Изучить метод цепных дробей разложения целых чисел на множители и привести его программную реализацию.

1 Теоретическая часть

1.1 -метод Полларда

Это вероятностный алгоритм факторизации целых чисел, с помощью которого разложено число F8. Используя случайное сжимающее отображение , строится рекуррентная последовательность со случайным начальным условием и проверяется , где – составное число, имеющее простой делитель. Тогда последовательность имеет период и последовательность имеет период . Значит, найдутся такие значения последовательности: , для которых и, значит,

Графически члены последовательности изображаются так, что сначала образуется конечный «хвост», а затем — цикл конечной длины .

1.1.1 Алгоритм -метода Полларда

Вход: составное число и значение .

Выход: нетривиальный делитель числа , вероятность не менее 1 – .

Шаг 1. Вычислить и выбрать случайный многочлен .

Шаг 2. Случайно выбрать и, последовательно вычисляя значения , проверять тест на шаге 3.

Шаг 3. Для каждого вычислить и проверить условие . Если это выполняется, то найден нетривиальный делитель числа *n*. Если же для всех , то перейти к выбору следующего значения последовательности на шаге 2. Если найдется для некоторого , то перейти к выбору нового значения  на шаге 2.

Шаг 4. Если вычислено *T* членов последовательности {}, а делитель числа не найден, то остановить алгоритм.

Для ускорения алгоритма можно модифицировать шаг 3. Для вычислять для .

1.1.2 Псевдокод -метода Полларда

Процедура Метод\_Полларда\_1():

= взять\_целое() + 1

= взять\_случайное\_число\_в\_интервале()

Взять как пустой массив

Взять

Пока :

temp = 0

Для всех элементов из :

Конец Для

Для всех :

Если :

Продолжить,

ИначеЕсли :

Вернуть

ИначеЕсли :

Вернуть пустое\_значение

Конец Если

Конец Пока

Конец Процедуры

1.2 ()-метод Полларда

Пусть – нечетное составное число и – его нетривиальный делитель, тогда и каноническое разложение числа имеет вид . Найдём максимальные показатели , для которых . Прологарифмируем обе части этого неравенства:

, откуда

Вычислим:

тогда для некоторого целого числа .

По малой теореме Ферма:

для любого целого , взаимно простого с .

Возведя обе части этого сравнения в степень , получаем .

Обозначим . Если , то число должно делиться на , поскольку разность делится на , а число делится на .

1.2.1 Алгоритм ()-метода Полларда

Пусть *n* – составное число. Фиксируется параметр метода – положительное B. Рассматривается множество простых чисел {– факторная база и значения , .

Вход: составное число n, число B > 0.

Выход: разложение числа n на нетривиальные делители.

Шаг 1. Случайно выбрать и вычислить . Если , то найден нетривиальный делитель d числа n. Если , то вычислить .

Шаг 2. Вычислить . Если , то увеличить B. Если , то перейти к шагу 1 и выбрать новое значение . Если для нескольких значений выполняется , то уменьшить B. Если , то найден нетривиальный делительчисла .

Сложность алгоритма: .

1.2.2 Псевдокод ()-метода Полларда

Процедура Метод\_Полларда\_2():

Пока :

Если :

Конец Если

Конец Пока

Если :

Вернуть

Конец Если

Для всех в диапазоне :

Конец Для

Если :

Вернуть пустое\_значение

Конец Если

Вернуть

Конец Процедуры

1.3 Метод цепных дробей

В этом методе в качестве чисел выбираются числители подходящих дробей к обыкновенной цепной дроби, выражающей число

Теорема 1. Если , где подходящие дроби к числу которое задано обыкновенной непрерывной дробью, то для всех справедлива оценка

Следствие. Пусть положительное целое число не является полным квадратом и , где подходящие дроби к числу Тогда для абсолютно наименьшего вычета справедлива оценка , причем .

Обыкновенная цепная дробь имеет вид:

где – целое число и – целые положительные числа (неполные частные). Цепная дробь обозначается: .

Для цепной дроби выражения

называются *подходящими дробями конечной цепной дроби*.

Каждая подходящая дробь является несократимой рациональной дробью с числителем и знаменателем *,* которые вычисляются по следующим рекуррентным формулам:

с начальными условиями:

.

Теорема 2.

Пусть – квадратичная иррациональность вида , где , N, , u, v|D2 – u. Тогда для любого k справедливо разложение в бесконечную цепную дробь = [,…], где , , , (k+1)-й остаток. При этом справедливы соотношения = [], = v, = u + и при k , где = , , = и числа получаются с помощью рекуррентной формулы .

1.3.1 Алгоритм метода цепных дробей

Обозначения:

,

где — бесконечно малая при и .

Для фиксированного положим

,

где .

Пусть – составное число (что установлено с помощью вероятностных алгоритмов простоты), которое не имеет небольших простых делителей (что проверяется пробными делениями).

Общая идея Лагранжа: найти решения сравнения , удовлетворяющие условию , и, значит,

влечет, что один делитель числа делит и другой делитель числа делит . Для этого проверяются два условия , .

Общая схема субэкспоненциальных алгоритмов факторизации:

1. Создаются наборы сравнений с небольшими .

2. Факторизуются числа .

3. Перемножаются сравнения из набора с целью получения сравнения с условием .

4. Вычисляются , .

Известно, что для случайной пары , удовлетворяющей условию , вероятность

.

Алгоритм Диксона:

Пусть – некоторый параметр и – факторная база всех простых чисел, не превосходящих .

– наименьший неотрицательный вычет числа .

Шаг 1. Случайным выбором ищем чисел , для которых , обозначаем .

Шаг 2. Найти ненулевое решение системы линейных уравнений с неизвестными

.

Шаг 3. Положить

,

для которых

.

Проверить условие . Если выполняется, то получаем собственный делитель числа (с вероятностью успеха ). В противном случае возвращаемся на шаг 1 и выбираем другие значения .

Сложность алгоритма минимальна при и равна

для

Алгоритм Бриллхарта-Моррисона отличается от алгоритма Диксона только способом выбора значений на шаге 1: случайный выбор заменяется детерминированным определением этих значений с помощью подходящих дробей для представления числа цепной дробью.

Теорема. Пусть и — подходящая дробь для представления числа цепной дробью. Тогда абсолютно наименьший вычет равен значению и выполняется .

Разложение числа в цепную дробь с помощью только операции с целыми числами и нахождения целой части чисел вида может быть найдено по следующей теореме.

Теорема. Пусть — квадратичная иррациональность вида , где . Тогда для любого справедливо разложение в бесконечную цепную дробь , где — -й остаток. При этом справедливы соотношения и при , где и числа получаются с помощью рекуррентной формулы .

Таким образом, в алгоритме Диксона возможен выбор и факторная база сужается .

Сложность алгоритма минимальна при и равна .

1.3.2 Псевдокод алгоритма Бриллхарта-Моррисона

Функция Бриллхар\_Моррисон(n, a)

L=e^((logn\*log(logn))^a)

Сгенерировать факторную базу, первый элемент это -1, затем все простые числа pi <= L такие, что Якоби(pi, n) != -1;

k = размер базы

вычислить числители и знаменатели подходящих дробей корня из n

Бесконечный цикл:

Qmi = Pi^2 – nQi^2

Вычислить k+1 массивов:

vi = (ai0, …, aik)

ei = (ai0 % 2, …, aik % 2)

x = решение СЛУ (k уравнений, k+1 неизвестных) x1v1 + … + xk+1\*vk+1= 0 (mod 2);

Если пусто:

Увеличить базу;

Продолжить цикл;

X = 1;

Y = 1;

Для i от 0 до k

X = X\*P[i]^x[i] mod n;

Для j от 0 до k-1

step = 0;

Для i от 0 до размера решения x

step += x[i] \* vStep[i][j];

step /= 2;

Y = Y\*p[j]^step mod n;

Если X^2 mod n != Y^2 mod n

Продолжить цикл

gcd1 = НОД(X+Y, n);

gcd2 = НОД(X-Y, n)

Если gcd1 (0, n) ИЛИ gcd2 (0, n):

d[0] = gcd1 или gcd2;

d[1] = n / d[0]

Вернуть d[0], d[1]

Иначе:

Продолжить цикл

Конец Функции

2 Тестирование программ

На рисунке 1 представлено тестирование работы программы реализации -метода Полларда.

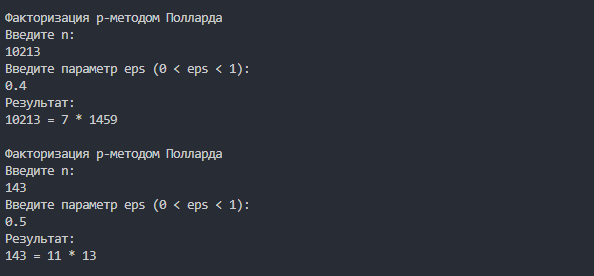


Рисунок 1 – Тестирование -метода Полларда

На рисунке 2 представлено тестирование работы программы реализации -метода Полларда.

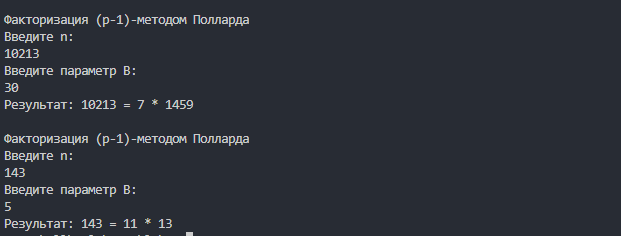


Рисунок 2 - Тестирование ρ-1-метода Полларда

Приложение А  
Код программы

require 'prime'

require 'matrix'

class PollardMethod

  def initialize(n, eps)

    raise ArgumentError, 'eps должен быть в пределах от 0 до 1' unless eps > 0 && eps < 1

    @n = n

    @eps = eps

  end

  def factorize

    return "#{@n} = 1 \* #{@n}" if @n == 1

    return "#{@n} = #{@n}" if Prime.prime?(@n)

    x = 2

    y = 2

    d = 1

    f = proc { |z| (z\*\*2 + 1) % @n }

    while d == 1

      x = f.call(x)

      y = f.call(f.call(y))

      d = (x - y).gcd(@n)

    end

    if d == @n

      return "Не удалось разложить число #{@n}."

    else

      q = @n / d

      return "#{@n} = #{d} \* #{q}"

    end

  end

end

class PollardMethodPMinusOne

  def initialize(n, b)

    @n = n

    @b = b

    @t = 1

  end

  def generate\_base

    base = []

    Prime.each(@b) do |p|

      base << p

      k = (Math.log(@n) / Math.log(p)).to\_i

      @t \*= p\*\*k

    end

    base

  end

  def factorize\_number(number)

    factors = []

    d = 2

    while d <= number

      if (number % d).zero?

        factors << d

        number /= d

      else

        d += 1

      end

    end

    factors

  end

  def factorize

    generate\_base

    factors = []

    loop do

      a = rand(1..@n)

      d = a.gcd(@n)

      if d > 1 && d < @n

        factors.concat(factorize\_number(d))

        factors.concat(factorize\_number(@n / d))

        return factors.uniq.sort

      end

      if d == 1

        b = a.pow(@t, @n) - 1

        n1 = b.gcd(@n)

        if n1 == 1

*# Increase factor base*

          generate\_base

          next

        elsif n1 == @n

*# Decrease factor base*

          generate\_base

          next

        else

          factors.concat(factorize\_number(n1))

          factors.concat(factorize\_number(@n / n1))

          return factors.uniq.sort

        end

      end

    end

  end

end

*# class BrillhartMorrisonFactorization*

*#   attr\_accessor :n, :a*

*#   def initialize(n, a)*

*#     @n = n*

*#     @a = a*

*#   end*

*#   def factorize*

*#     puts "Исходное число: #{n}"*

*#     # Шаг 1: Вычисление L*

*#     l = Math.exp((Math.log(n) \* Math.log(Math.log(n)))\*\*a).to\_i*

*#     puts "L = #{l}"*

*#     # Шаг 2: Генерация факторной базы*

*#     factor\_base = [-1] + Prime.each(l).select { |pi| jacobi(pi, n) != -1 }*

*#     puts "Факторная база: #{factor\_base}"*

*#     loop do*

*#       x, y = solve\_linear\_system(factor\_base)*

*#       break if x.empty?*

*#       gcd1 = gcd(x + y, n)*

*#       gcd2 = gcd(x - y, n)*

*#       if gcd1 > 1 && gcd1 < n*

*#         puts "Найден нетривиальный делитель: #{gcd1}"*

*#         puts "Факторизация: #{gcd1} \* #{n / gcd1}"*

*#         break*

*#       elsif gcd2 > 1 && gcd2 < n*

*#         puts "Найден нетривиальный делитель: #{gcd2}"*

*#         puts "Факторизация: #{gcd2} \* #{n / gcd2}"*

*#         break*

*#       else*

*#         factor\_base += Prime.each(l).select { |pi| jacobi(pi, n) != -1 && !factor\_base.include?(pi) }*

*#         puts "Увеличение факторной базы: #{factor\_base}"*

*#       end*

*#     end*

*#   end*

*#   private*

*#   def legendre\_symbol(a, p)*

*#     return 1 if a == 1*

*#     return -1 if a % 2 == 0 && (p % 8 == 3 || p % 8 == 5)*

*#     return legendre\_symbol(p % a, a) \* -1 if a % 4 == 3 && p % 4 == 3*

*#     return legendre\_symbol(p, a) if a % 2 == 1*

*#     return 0 if a == 0 # Условие выхода из рекурсии*

*#   end*

*#   def legendre\_symbol(a, p)*

*#     return 1 if a == 1*

*#     return -1 if a % 2 == 0 && (p % 8 == 3 || p % 8 == 5)*

*#     return legendre\_symbol(p % a, a) \* -1 if a % 4 == 3 && p % 4 == 3*

*#     return legendre\_symbol(p, a) if a % 2 == 1*

*#     return 0 if a == 0 # Условие выхода из рекурсии*

*#   end*

*#   def solve\_linear\_system(factor\_base)*

*#     k = factor\_base.size*

*#     matrix\_a = Matrix.build(k, k + 1) { rand(2) }*

*#     k.times do |i|*

*#       max\_row = (i...k).max\_by { |row| matrix\_a[row, i].abs }*

*#       tmp = matrix\_a[i, 0..k].to\_a*

*#       matrix\_a[i, 0..k] = matrix\_a[max\_row, 0..k]*

*#       matrix\_a[max\_row, 0..k] = tmp*

*#       pivot = matrix\_a[i, i]*

*#       (i + 1...k).each do |j|*

*#         puts matrix\_a[j, i]*

*#         factor = matrix\_a[j, i] / pivot*

*#         matrix\_a[j, 0..k] -= factor \* matrix\_a[i, 0..k]*

*#       end*

*#     end*

*#     x = Array.new(k, 0)*

*#     y = Array.new(k, 0)*

*#     (k - 1).downto(0) do |i|*

*#       y[i] = matrix\_a[i, k]*

*#       (i + 1...k).each { |j| y[i] -= matrix\_a[i, j] \* y[j] }*

*#       x[i] = y[i] % 2*

*#     end*

*#     return x, y*

*#   end*

*#   def gcd(a, b)*

*#     while b != 0*

*#       a, b = b, a % b*

*#     end*

*#     return a*

*#   end*

*#   def jacobi(a, n)*

*#     return 0 if a.gcd(n) != 1*

*#     t = 1*

*#     a %= n*

*#     while a != 0*

*#       while a % 2 == 0*

*#         a /= 2*

*#         if n % 8 == 3 || n % 8 == 5*

*#           t = -t*

*#         end*

*#       end*

*#       n, a = a, n*

*#       if a % 4 == 3 && n % 4 == 3*

*#         t = -t*

*#       end*

*#       a %= n*

*#     end*

*#     t*

*#   end*

*# end*

*# Пример использования:*

def get\_pollard\_p

  puts "\nФакторизация p-методом Полларда"

  puts "Введите n:"

  n = gets.strip.to\_i

  puts "Введите параметр eps (0 < eps < 1):"

  eps = gets.strip.to\_f

  pollard = PollardMethod.new(n, eps)

  puts "Результат:\n#{pollard.factorize}"

end

def get\_pollard\_p\_2

  puts "\nФакторизация (p-1)-методом Полларда"

  puts "Введите n:"

  n = gets.strip.to\_i

  puts "Введите параметр B:"

  b = gets.strip.to\_i

  pollard = PollardMethodPMinusOne.new(n, b)

  result = pollard.factorize

  puts "Результат: #{n} = #{result.map(&:to\_s).join(" \* ")}"

end

def get\_brimor

  puts "\nФакторизация методом цепных дробей"

  puts "Введите n:"

  n = gets.strip.to\_i

  puts "Введите параметр a:"

  a = gets.strip.to\_f

  factorizer = BrillhartMorrisonFactorization.new(n, a)

  factorizer.factorize

end

get\_pollard\_p

get\_pollard\_p

get\_pollard\_p\_2

get\_pollard\_p\_2

*# get\_brimor*