Оглавление

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

Задачи работы:

- Изучить ρ-метод Полларда разложения целых чисел на множители и привести его программную реализацию;
- Изучить $(p-1)$-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 ρ-метод Полларда

Это вероятностный алгоритм факторизации целых чисел, с помощью которого разложено число F_8. Используя случайное сжимающее отображение $f: Z_n \to Z_n$, строится рекуррентная последовательность $x_{i+1} = f(x_i)\ mod\ n$ со случайным начальным условием $x_0 \in Z_n$ и проверяется $1 <$ НОД $(x_i - x_k, n) < n$, где n – составное число, имеющее простой делитель $p < \sqrt{n}$. Тогда последовательность $\{x_i\}$ имеет период $\leq n$ и последовательность $\{x_i\ mod\ p\}$ имеет период $\leq p$. Значит, найдутся такие значения последовательности: $x_i,\ x_k$, для которых $x_i \equiv x_k\ (mod\ p),\ x_i \equiv x_k\ (mod\ n)$, и, значит, НОД $(x_i, x_k) < n$.

Графически члены последовательности $\{x_i\}$ изображаются так, что сначала образуется конечный «хвост», а затем — цикл конечной длины $\leq p$.

1.1.1 Алгоритм ρ-метода Полларда

Вход: составное число n и значение $0 < \varepsilon < 1$.

Выход: нетривиальный делитель p числа n, вероятность не менее $1 - \varepsilon$.

Шаг 1. Вычислить $T = \left[\sqrt{2\sqrt{n}\, ln\frac{1}{\varepsilon}}\right] + 1$ и выбрать случайный многочлен $f \in Z_n$.

Шаг 2. Случайно выбрать $x_0 \in Z_n$ и, последовательно вычисляя значения $x_{i+1} = f(x_i)\ mod\ n, 0 \leq i \leq T$, проверять тест на шаге 3.

Шаг 3. Для каждого $0 \leq k \leq i$ вычислить $d_k =$ НОД$(x - x_k, n)$ и проверить условие $1 < d_k < n$. Если это выполняется, то найден нетривиальный делитель d_k числа n. Если же $d_k = 1$ для всех $0 \leq k \leq i$, то перейти к выбору следующего значения последовательности на шаге 2. Если найдется $d_k = n$ для некоторого $0 \leq k \leq i$, то перейти к выбору нового значения $x_0 \in Z_n$ на шаге 2.

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

Для ускорения алгоритма можно модифицировать шаг 3. Для $2^h \le i < 2^{h+1}$ вычислять $d_k = \text{НОД}(x_{i+1} - x_k, n)$ для $k = 2^{h-1}$.

1.1.2 Псевдокод ρ-метода Полларда

```
Процедура Метод_Полларда_1(n, f, e):
    T = взять_целое(sqrt(2 * sqrt(n) * log10(1/e))) + 1
    a = взять_случайное_число_в_интервале(1, n)
    Взять x как пустой массив
    Взять i = 0
    Пока i < T:
        temp = 0
        Для всех элементов f_i из f:
            temp+= f_i[:c] * a ^f_i[:n]
        Конец Для
        a = temp (mod n)
        x ≪ a
        Для всех k ϵ [0, i):
            d = алгоритм_Евклида ((x_i − x_k) mod n, n)
            Если d == 1:
                Продолжить,
            ИначеЕсли 1 < d < n:
                Вернуть d
            ИначеЕсли d == n:
                Вернуть пустое_значение
            Конец Если
        i+= 1
    Конец Пока
Конец Процедуры
```

1.2 $(p-1)$-метод Полларда

Пусть n – нечетное составное число и p – его нетривиальный делитель, тогда $n = pq$ и каноническое разложение числа $p-1$ имеет вид $p-1 = {p_1}^{\alpha_1}{p_2}^{\alpha_2} \dots {p_s}^{\alpha_s}$. Найдём максимальные показатели $l_1, l_2, \dots, l_s$, для которых ${p_i}^{l_i} \le n$. Прологарифмируем обе части этого неравенства:

$l_i \ln p_i \le \ln n$, откуда $l_i \le \left\lceil \frac{\ln n}{\ln p_i} \right\rceil$

Вычислим:

$$M = {p_1}^{\left\lceil \frac{\ln n}{\ln p_1} \right\rceil}{p_2}^{\left\lceil \frac{\ln n}{\ln p_2} \right\rceil} \dots {p_s}^{\left\lceil \frac{\ln n}{\ln p_s} \right\rceil},$$

тогда $M = (p-1) \cdot z$ для некоторого целого числа z.

По малой теореме Ферма:

$\alpha^{p-1} \equiv 1 \ (mod\ p)$ для любого целого α, взаимно простого с p.

Возведя обе части этого сравнения в степень z, получаем $\alpha^M \equiv 1\ (mod\ p)$.

Обозначим $d = \text{НОД}(\alpha^M - 1, n)$. Если $\alpha^M \equiv 1\ (mod\ n)$, то число d должно делиться на p, поскольку разность $\alpha^M - 1$ делится на p, а число n делится на p.

1.2.1 Алгоритм $(p-1)$-метода Полларда

Пусть n – составное число. Фиксируется параметр метода – положительное B. Рассматривается множество простых чисел $\{q_1, \dots, q_{\pi(B)}\}$ – факторная база и значения $k_i = \left[\frac{ln\, n}{\ln q_i}\right]$, $T = \prod_{i=1}^{\pi(B)} q_i^{k_i}$.

Вход: составное число n, число B > 0.

Выход: разложение числа n на нетривиальные делители.

Шаг 1. Случайно выбрать $a \in Z_n$ и вычислить $d = \text{НОД}(a, n)$. Если $1 < d < n$, то найден нетривиальный делитель d числа n. Если $d = 1$, то вычислить $b = a^T - 1\ (mod\ n)$.

Шаг 2. Вычислить $b_1 = \text{НОД}(b, n)$. Если $n_1 = 1$, то увеличить B. Если $n_1 = n$, то перейти к шагу 1 и выбрать новое значение $a \in Z_n$. Если для нескольких значений $a \in Z_n$ выполняется $n_1 = n$, то уменьшить B. Если $1 < n_1 < n$, то найден нетривиальный делитель n_1 числа n.

Сложность алгоритма: $O(\pi(B) \log^3 n)$.

1.2.2 Псевдокод $(p-1)$-метода Полларда

```
Процедура Метод_Полларда_2(n):
        p = пустой_массив
        x = 2
        Пока длина(p) < 3 И x < n:
                Если x < 4 ИЛИ (x ⋮ 2 И алгоритм_Миллера_Рабина(x, 5)):
                        p ≪ x
                Конец Если
                x+= 1
        Конец Пока
        a = взять_случайное_целое_число_в_диапазоне(2, n − 2)
        d = алгоритм_Евклида(a, n)
        Если d > 3:
```

```
            Вернуть d
        Конец Если
        Для всех i в диапазоне [0, длина(p)):
            l = взять_целое(log2 n / log2 p_i)
            a = a^(p_i^l) mod n
        Конец Для
        d = алгоритм_Евклида(a − 1, n)
        Если d == 1 ИЛИ d == n:
            Вернуть пустое_значение
        Конец Если
        Вернуть d
Конец Процедуры
```

1.3 Метод цепных дробей

В этом методе в качестве чисел s_i выбираются числители подходящих дробей к обыкновенной цепной дроби, выражающей число $\sqrt{n}$.

Теорема 1. Если $\frac{P_k}{Q_k}$, где $k = 1, 2, \ldots$ – подходящие дроби к числу $\alpha > 1$, которое задано обыкновенной непрерывной дробью, то для всех k справедлива оценка $|P_k^2 - \alpha^2 Q_k^2| < 2\alpha$.

Следствие. Пусть положительное целое число n не является полным квадратом и $\frac{P_k}{Q_k}$, где $k = 1, \ldots, n$ – подходящие дроби к числу $\sqrt{n}$. Тогда для абсолютно наименьшего вычета $P_k^2 (mod\ n)$ справедлива оценка $|P_k^2 (mod\ n)| < 2\sqrt{n}$, причем $P_k^2 (mod\ n) = P_k^2 - nQ_k^2$.

Обыкновенная цепная дробь имеет вид:

$$r = \frac{a_0}{a_1} = q_1 + \frac{a_2}{a_1} = q_1 + \frac{1}{\frac{a_1}{a_2}} = q_1 + \frac{1}{q_2 + \frac{a_3}{a_2}} = \cdots$$

$$= q_1 + \cfrac{1}{q_2 + \cfrac{1}{\ldots + \cfrac{1}{q_{k-1} + \cfrac{1}{q_k}}}},$$

где q_1 – целое число и $q_2, \ldots, q_k$ – целые положительные числа (неполные частные). Цепная дробь обозначается: $(q_1; q_2, \ldots, q_k)$.

Для цепной дроби $\frac{a_0}{a_1} = (q_1; q_2, \ldots, q_k)$ выражения

$$\delta_1 = q_1,\ \delta_2 = q_1 + \frac{1}{q_2}, \dots,\ \delta_k = q_1 + \cfrac{1}{q_2 + \cfrac{1}{\dots + \cfrac{1}{q_{k-1} + \cfrac{1}{q_k}}}}$$

называются *подходящими дробями конечной цепной дроби*.

Каждая подходящая дробь $\delta_i,\ i = \overline{1,k}$ является несократимой рациональной дробью $\delta_i = \frac{P_i}{Q_i}$ с числителем P_i и знаменателем Q_i, которые вычисляются по следующим рекуррентным формулам:

$$P_i = q_i P_{i-1} + P_{i-2},\ Q_i = q_i Q_{i-1} + Q_{i-2}$$

с начальными условиями:

$$P_{-1} = 0,$$

$$P_0 = 1,$$

$$Q_{-1} = 1,$$

$Q_0 = 0$.

Теорема 2.

Пусть α – квадратичная иррациональность вида $\alpha = \frac{\sqrt{D}-u}{v}$, где $D \in N$, $\sqrt{D} \notin$ N, $v \in N$, u $\in N$, v|D^2 – u. Тогда для любого k ≥ 0 справедливо разложение в бесконечную цепную дробь $\alpha = [a_0, a_1, \dots, a_k, a_{k+1}, \dots]$, где $a_0 \in Z$, $a_1, \dots, a_k \in N$, a_{k+1} – (k+1)-й остаток. При этом справедливы соотношения $a_0 = [\alpha]$, v_0 = v, u_0 = u + a_0 v и при k ≥ 0 $a_{k+1} = [a_{k+1}]$, где $v_{k+1} = \frac{D-u_k^2}{v_k} \in Z$, $v_{k+1} \neq 0$, $\alpha_{k+1} = \frac{\sqrt{D}+u_k}{v_{k+1}} > 1$ и числа u_k получаются с помощью рекуррентной формулы $u_{k+1} = a_{k+1} v_{k+1} - u_k$.

1.3.1 Алгоритм метода цепных дробей

Обозначения:

$L_n[\gamma, c] = \exp((c + o(1)) \log^{\gamma} n\, (\log\log n)^{1-\gamma})$,

где $o(1)$ — бесконечно малая при $n \to \infty$ и $0 < \gamma < 1$.

Для фиксированного $\gamma = \frac{1}{2}$ положим

$$L_n[c] = L_n\left[\frac{1}{2}, c\right] = \exp((c + o(1))(\log n \log\log n)^{\frac{1}{2}}) = L^{c+o(1)},$$

где $L = \exp((\log n \log\log n)^{\frac{1}{2}})$.

Пусть n – 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 $x^2 \equiv y^2 (mod\ n)$, удовлетворяющие условию $x \not\equiv \pm y\ (mod\ n)$, и, значит,

$$(x - y)(x + y) \equiv 0\ (mod\ n)$$

влечет, что один делитель p числа n делит $x - y$ и другой делитель q числа n делит $x + y$. Для этого проверяются два условия $1 < \text{НОД}(x - y, n) < n$, $1 < \text{НОД}(x + y, n) < n$.

Общая схема субэкспоненциальных алгоритмов факторизации:

1. Создаются наборы сравнений $u \equiv v(mod\ n)$ с небольшими u, v.

2. Факторизуются числа u, v.

3. Перемножаются сравнения из набора с целью получения сравнения $x^2 \equiv y^2 (mod\ n)$ с условием $x \not\equiv \pm y\ (mod\ n)$.

4. Вычисляются $\text{НОД}(x - y, n)$, $\text{НОД}(x + y, n)$.

Известно, что для случайной пары $x, y \in \boldsymbol{Z}_n^*$, удовлетворяющей условию $x^2 \equiv y^2 (mod\ n)$, вероятность

$$P_0 = P[1 < \text{НОД}(x \pm y, n) < n] \geq \frac{1}{2}.$$

Алгоритм Диксона:

Пусть $0 < a < 1$ – некоторый параметр и B – факторная база всех простых чисел, не превосходящих $L^a, k = \pi(L^a)$.

$Q(m) \equiv m^2 (mod\ n)$ – наименьший неотрицательный вычет числа m^2.

Шаг 1. Случайным выбором ищем $k + 1$ чисел $m_1, .., m_{k+1}$, для которых $Q(m_i) = p_1^{\alpha_{i1}} \dots p_k^{\alpha_{ik}}$, обозначаем $\overline{v_i} = (\alpha_{i1}, .., \alpha_{ik})$.

Шаг 2. Найти ненулевое решение $(x_1, .., x_{k+1}) \in \{0,1\}^{k+1}$ системы k линейных уравнений с $k + 1$ неизвестными

$x_1\overline{v_1}+..+x_{k+1}\overline{v_{k+1}} = \bar{0}\ (mod\ 2)$.

Шаг 3. Положить

$X \equiv m_1^{x_1}..m_{k+1}^{x_{k+1}}\ (mod\ n), Y \equiv \prod_{j=1}^{k} p_j^{\frac{\sum x_i\alpha_{ij}}{2}}\ (mod\ n)$,

для которых

$X^2 \equiv p_1^{\sum_{i=1}^{k+1} x_i\alpha_{i1}}..p_k^{\sum_{i=1}^{k+1} x_i\alpha_{ik}} \equiv Y^2\ (mod\ n)$.

Проверить условие $1 <$ НОД$(X \pm Y, n) < n$. Если выполняется, то получаем собственный делитель числа n (с вероятностью успеха $P_0 \geq \frac{1}{2}$). В противном случае возвращаемся на шаг 1 и выбираем другие значения $m_1, .., m_{k+1}$.

Сложность алгоритма минимальна при $a = \frac{1}{2}$ и равна

$L_n\left[\frac{1}{2}, 2\right] = L^{2+o(1)}$ для $L = exp((lognloglogn)^{\frac{1}{2}})$.

Алгоритм Бриллхарта-Моррисона отличается от алгоритма Диксона только способом выбора значений $m_1, .., m_{k+1}$ на шаге 1: случайный выбор заменяется детерминированным определением этих значений с помощью подходящих дробей для представления числа $\sqrt{n}$ цепной дробью.

Теорема. Пусть $n \in N, n > 16, \sqrt{n} \notin N$ и $\frac{P_i}{Q_i}$ — подходящая дробь для представления числа $\sqrt{n}$ цепной дробью. Тогда абсолютно наименьший вычет $P_i^2\ (mod\ n)$ равен значению $P_i^2 - nQ_i^2$ и выполняется $\left|P_i^2 - nQ_i^2\right| < 2\sqrt{n}$.

Разложение числа $\sqrt{n}$ в цепную дробь с помощью только операции с целыми числами и нахождения целой части чисел вида $\frac{\sqrt{D}-u}{v}$ может быть найдено по следующей теореме.

Теорема. Пусть α — квадратичная иррациональность вида $\alpha = \frac{\sqrt{D}-u}{v}$, где $D \in N,\ \sqrt{D} \notin N, v \in N, u \in N, v|D^2 - u$. Тогда для любого $k \geq 0$ справедливо разложение в бесконечную цепную дробь $\alpha = [a_0, a_1, .., a_k, a_{k+1}, ..]$, где $a_0 \in Z, a_1, .., a_k \in N, a_{k+1}$ — $(k+1)$-й остаток.

При этом справедливы соотношения $a_0 = [\alpha], v_0 = v, u_0 = u + a_0 v$ и при $k \geq 0$ $a_{k+1} = [\alpha_{k+1}]$, где $v_{k+1} = \frac{D - u_k^2}{v_k} \in Z, v_{k+1} \neq 0, \alpha_{k+1} = \frac{\sqrt{D} + u_k}{v_{k+1}} > 1$ и числа u_k получаются с помощью рекуррентной формулы $u_{k+1} = a_{k+1} v_{k+1} - u_k$.

Таким образом, в алгоритме Диксона возможен выбор $m_i = P_i, Q(m_i) \equiv m_i^2 = P_i^2 \equiv P_i^2 - nQ_i^2 \ (mod\ n), Q(m_i) = P_i^2 - nQ_i^2$ и факторная база сужается $B = \{p_0 = -1\} \cup \{p - \text{простое число}: p \leq L^a \text{ и } n \in QR_p\}$.

Сложность алгоритма минимальна при $a = \frac{1}{\sqrt{2}}$ и равна $L_n[\frac{1}{2}, \sqrt{2}]$.

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 x 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 ρ-метода Полларда.

Рисунок 1 – Тестирование ρ-метода Полларда

На рисунке 2 представлено тестирование работы программы реализации $p - 1$-метода Полларда.

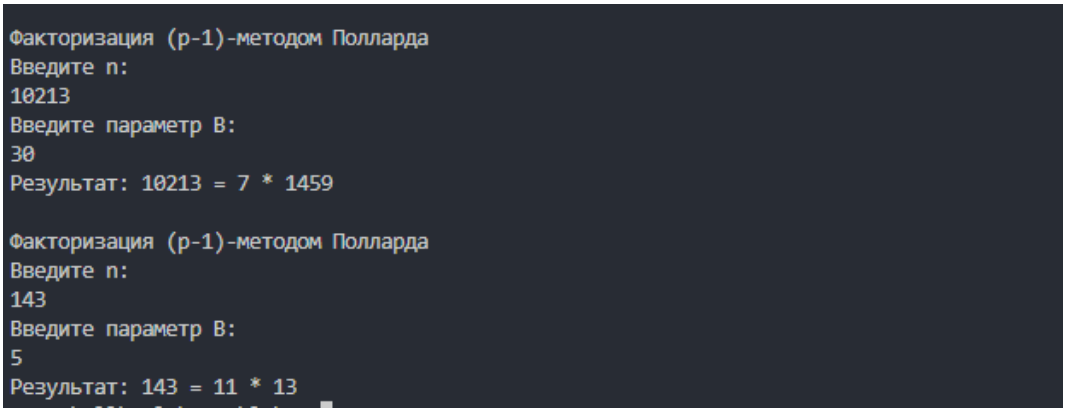

Рисунок 2 - Тестирование ρ-1-метода Полларда

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))
```

```ruby
          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
```

```ruby
#     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
```

```ruby
  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
```