Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Факторизация целых чисел

ОТЧЕТ ПО ДИСЦИПЛИНЕ
«ТЕОРЕТКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ»

Лабораторная работа №4

студента 5 курса 531 группы
специальности 10.05.01 «Компьютерная безопасность»
факультета компьютерных наук и информационных технологий
Гельфанова Даниила Руслановича

Преподаватель
профессор ___________ В. А. Молчанов
подпись, дата

Саратов 2023

1 Постановка задачи

Цель работы – изучение основных методов факторизации целых чисел и их программная реализация.

В рамках данной лабораторной работы нужно выполнить следующие задачи:

1. Рассмотреть ρ-метод Полларда разложения целых чисел на множители и привести его программную реализацию.

2. Рассмотреть $(p-1)$-метод Полларда разложения целых чисел на множители и привести его программную реализацию.

3. Рассмотреть метод цепных дробей разложения целых чисел на множители и привести его программную реализацию.

2 Теоретические сведения

Алгоритмы факторизации бывают:

1) экспоненциально зависящие от длины позиционной записи числа n;

2) субэкспоненциальные алгоритмы, имеющие оценку сложности вида

$$L_n(\gamma, c) = \exp(\left(c + o(1)\right) \log^{\gamma} n\, (loglogn)^{1-\gamma})$$

где $o(1)$ – б.м. при $n \to \infty$ и $0 < \gamma < 1$.

При $\gamma = 0$ величина $L_n(0, c) = (logn)^{c+o(1)}$ – степенная функция от $\log n$.

При $\gamma = 1$ величина $L_n(1, c) = n^{c+o(1)}$ – экспоненциальная функция от $\log n$.

Все современные алгоритмы факторизации субэкспоненциальны.

2.1 $\boldsymbol{\rho}$-метод Полларда

Этот алгоритм является экспоненциальным. Это вероятностный алгоритм факторизации целых чисел, с помощью которого разложено число $F_8 = 2^{2^8} + 1$.

С помощью случайного сжимающего отображения $f: \boldsymbol{Z}_n \to \boldsymbol{Z}_n$ (например, многочлена) строится рекуррентная последовательность $x_{i+1} = f(x_i)(mod\ n)$ со случайным начальным условием $x_0 \in \boldsymbol{Z}_n$ и проверяется

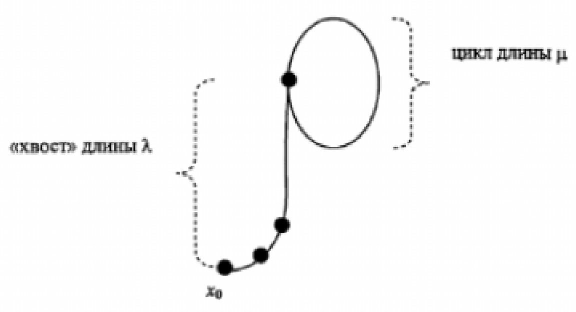
$$1 < \text{НОД}(x_i - x_k, n) < n.$$

Так как составное число n имеет простой делитель $p < \sqrt{n}$, то последовательность $\{x_i\}$ имеет период $\le n$ и последовательность $\{x_i(mod\ p)\}$ имеет период $\le p$. Значит, с большой вероятностью найдутся такие значения последовательности x_i, x_k, для которых

$$x_i \equiv x_k(mod\ p), x_i \not\equiv x_k(mod\ n)$$

и, значит, $1 < \text{НОД}(x_i - x_k, n) < n$.

Графически члены последовательности $\{x_i\}$ изображаются так, что сначала образуется конечный «хвост», а затем — цикл конечной длины $\le p$. Из-за такой фигуры метод называется ρ-методом.



Алгоритм:

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

Выход. Нетривиальный делитель d числа n, $1 < d < n$ с вероятностью не менее $1 - \varepsilon$.

Шаг 1. Вычислить $T = \left[\sqrt{2\sqrt{n}\ln\frac{1}{\varepsilon}}\right] + 1$ и выбрать случайный многочлен $f \in \boldsymbol{Z}_n[x]$ (например, $f(x) = x^2 + 1$).

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

Шаг 3. Для каждого $0 \le k \le i$ вычислить $d_k = \text{НОД}(x_{i+1} - x_k, n)$ 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 $0 \le k \le i$, то перейти к выбору следующего значения последовательности на шаге 2. . Если найдется $d_k = n$ для некоторого $0 \le k \le i$, то перейти к выбору нового значения $x_0 \in \boldsymbol{Z}_n$ на шаге 2.

Число шагов алгоритма можно ограничить значением $T = \left[\sqrt{2\sqrt{n}\ln\frac{1}{\varepsilon}}\right] + 1$ и получаем экспоненциальную общую сложность вычислений

$$O(k^2 \log^2 n) = O(\sqrt{n}\ln\tfrac{1}{\varepsilon}\log^2 n).$$

Теорема («парадокс дней рождения»). Пусть $\lambda > 0$ и $k = \lceil\sqrt{2\lambda n}\rceil$. Для случайной выборки объема $k+1$ из n элементов вероятность $P_{n,k}$ того, что все элементы попарно различны удовлетворяет условию $P_{n,k} < e^{-\lambda}$.

Замечание 1. Емкостная сложность алгоритма значительно упрощается за счёт его модификации (предложенной Флойдом) – параллельно вычисляются пары членов последовательности (x_i, x_{2i}) до тех пор, пока не найдётся такое k, что $x_k = x_{2k}$. Здесь экспоненциальная сложность вычислений $O(\sqrt{n})$.

Замечание 2. Алгоритм значительно ускоряется за счет модификации шага 3: для $2^h \le i < 2^{h+1}$ вычислять $d_k = \text{НОД}(x_{i+1} - x_k, n)$ для $k = 2^{h-1}$. Получаем экспоненциальную общую сложность вычислений $O(\sqrt[4]{n}\ln\frac{1}{\varepsilon}\log^2 n)$.

2.2 $(\boldsymbol{p} - \mathbf{1})$-метод Полларда

Пусть n – составное число. Фиксируется параметр метода – число $B > 0$, (для больших чисел n, как правило, $10^5 < B \le \sqrt{n}$).

Будем называть B – гладкими те числа, у которых все простые множители не превосходят B.

Рассматривается множество простых чисел $\{q_1, .., q_{\pi(B)}\}$ – факторная база и значения

$$k_i = \left[\frac{\ln n}{\ln q_i}\right] (\text{чтобы } q_i^{k_i} \leq n), T = \prod_{i=1}^{\pi(B)} q_i^{k_i}.$$

Алгоритм:

Вход. Составное число n, число $B > 0$ и значение $T = \prod_{i=1}^{\pi(B)} q_i^{k_i}$.

Выход. Разложение числа n на нетривиальные делители.

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

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

Сложность вычисления $a^T \equiv 1\ (mod\ n)$ равна $O(\log T) = O(\pi(B) \log n)$, сложность вычисления $\text{НОД}(b, n)$ равна $O(\log^2 n)$ и общая алгоритма равна $O(\pi(B) \log^3 n)$. Сложность алгоритма при малых B полиномиальная и при $B \approx \sqrt{n}$ экспоненциальная.

2.3 Алгоритм Бриллхарта-Моррисона

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

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

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

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

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

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

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 $B = \{p_0 = -1\} \cup \{p -$ простое число$: p \leq L^a$ и $n \in QR_p\}$.

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

3 Результаты работы

3.1 Псевдокод ρ-метода Полларда

```
Функция ро_Поллард(n, eps)
        T = корень(2*корень(n) * log(1 / eps)) + 1;
        Бесконечный цикл:
            xi = случайное число от 1 до n;
            xk = xi;
            Цикл для i от 0 до T:
                xi = f(xi) mod n;
                dk = НОД(xi - xk, n);
                Если dk = 1 {
                    Продолжить цикл;
                Если dk = n:
                    Выйти из цикла;
                d[0] = dk;
                d[1] n /dk;
                Вернуть d[0], d[1];
            Конец цикла
            Если i <= T
                Продолжить цикл;
            Прекратить цикл;
Конец функции
```

3.2 Псевдокод $(p-1)$-метода Полларда

```
Функция Генерация_базы()
        this.T = BigInteger.ONE;
        для всех простых p от 2 до B:
            В массив базы положить qi = p;
            ki = (целая часть) log(n) / log(qi));
            T = (T * qi)^ki;
Конец функции

Функция p-1_Поллард (n, B)
        Генерация_базы();
        flag = ложь;
        Бесконечный цикл
            a = случайное число от 1 до n;
            d = НОД(a, n);
            Если d > 1 И d < n:
```

```
                    d[0] = d;
                    d[1] = n /d;
                    Вернуть d[0], d[1];
                Если d = 1
                    b = (a^T mod n) -1;
                    n1 = НОД(b, n);
                    Если n1 == 1
                        Увеличить факторную базу
                        Продолжить цикл;
                    Если n1 = n
                        Если flag
                            Уменьшить факторную базу;
                            flag = ложь;
                            Продолжить цикл;
                        flag = истина;
                        Продолжить цикл;
                    d[0] = n1;
                    d[1] = n / n1;
                    Вернуть d[0], d[1]
Конец Функции
```

3.3 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
```

3.4 Тестирование программы

На рисунке 1 представлено тестирование работы программа реализации ρ-метода Полларда.

```
Выберите задание:
1. ρ-метод Полларда разложения целых чисел на множители;
2. (p-1)-метод Полларда разложения целых чисел на множители;
3. Метод цепных дробей разложения целых чисел на множители.
Ваш выбор: 1

Введите составное число n: 143
Введите число 0 < eps < 1: 0.4
143 = 13 * 11
```

Рисунок 1 – Вход: n = 143, eps = 0.4

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

```
Выберите задание:
1. ρ-метод Полларда разложения целых чисел на множители;
2. (p-1)-метод Полларда разложения целых чисел на множители;
3. Метод цепных дробей разложения целых чисел на множители.
Ваш выбор: 2

Введите составное число n: 10213
Введите число B: 30
10213 = 7 * 1459
```

Рисунок 2 – Вход: n = 10213, B = 30

На рисунке 3 представлено тестирование работы программы реализации алгоритма Бриллхарта-Моррисона:

```
Выберите задание:
1. ρ-метод Полларда разложения целых чисел на множители;
2. (p-1)-метод Полларда разложения целых чисел на множители;
3. Метод цепных дробей разложения целых чисел на множители.
Ваш выбор: 3

Введите составное число n: 21299881
Введите параметр a: 0.065
p: -1 2 3 5 7 11 19
-4235: 1 0 0 1 1 0 0
2688: 0 1 1 0 1 0 0
-7920: 1 0 0 1 0 1 0
385: 0 0 0 1 1 1 0
-3800: 1 1 0 0 0 0 1
-1331: 1 0 0 0 0 1 0
5415: 0 0 1 1 0 0 0
-112: 1 0 0 0 1 0 0
x: 1 0 1 0 0 1 0 1
21299881 = 5531 * 3851
```

Рисунок 3 – Вход: n = 21299881, a = 0 065

ПРИЛОЖЕНИЕ А

Листинг программы

```
import java.util.*;

public class Main {
    public static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {
        System.out.print("""
                Выберите задание:
                1. ρ-метод Полларда разложения целых чисел на множители;
                2.  (p-1)-метод  Полларда  разложения  целых  чисел  на
множители;
                3.  Метод  цепных  дробей  разложения  целых  чисел  на
множители.
                Ваш выбор:\s""");
        int task;
        for (; ; ) {
            String strTask = sc.next();
            if (strTask.isEmpty() || !isNumber(strTask)) {
                System.out.print("Вы    ввели    некорректное    число!
Пожалуйста, попробуйте снова: ");
            } else {
                task = Integer.parseInt(strTask);
                if (task < 1 || task > 3) {
                    System.out.print("Некорректный номер задания. Число
должно быть в промежутке [1, 3]. Повторите попытку: ");
                    continue;
                }
                break;
            }
        }
        System.out.println();
        if (task == 1) {
            RhoPollard.runPollard();
        } else if (task == 2) {
            P1Pollard.runPollard();
        } else {
            BrillhartMorrison.runFactorization();
        }
    }

    public static boolean isNumber(String isNum) {
        int check;
        if (isNum.charAt(0) == '-') {
            isNum = isNum.substring(1);
        }
        String[] sArray = isNum.split("");
        for (String digit : sArray) {
            try {
                int dig = Integer.parseInt(digit);
            } catch (Exception e) {
                return false;
            }
        }
```

```
        return true;
    }
}


import java.math.*;
import java.util.*;

public class BrillhartMorrison {
    public BigInteger n;
    private double a;
    private int k;
    private ArrayList<BigInteger> p = new ArrayList<>();
    public BigInteger[] d = new BigInteger[2];

    private static BigDecimal TWO = BigDecimal.valueOf(2);

    private double log(BigInteger c) {
        int t = c.bitLength();
        BigDecimal s = new BigDecimal(c).divide(TWO.pow(t));
        return Math.log(s.doubleValue()) + t * Math.log(2);
    }

    private void generateFactorBase() {
        String s = "p: -1 ";
        this.p.add(BigInteger.valueOf(-1));
        BigInteger         L         =         BigInteger.valueOf((long)
Math.pow(Math.exp(log(this.n) * Math.log(log(n))), this.a));
//         System.out.println("L " + L.toString());
        for (BigInteger p = BigInteger.valueOf(2); p.compareTo(L) <= 0;
p = p.nextProbablePrime()) {
            if (new JacobiSymbol(this.n, p).value != -1) {
                this.p.add(p);
                s += p.toString() + " ";
            }
        }
        this.k = this.p.size();
        System.out.println(s);
    }

    private   ArrayList<Integer>   factorizationBase(BigInteger   Qmi,
ArrayList<Integer> vStep) {
        ArrayList<Integer> vi = new ArrayList<>();
        BigInteger c = Qmi;
        if (c.compareTo(BigInteger.ZERO) < 0) {
            vi.add(1);
            vStep.add(1);
            c = c.negate();
        } else {
            vi.add(0);
            vStep.add(0);
        }
        boolean existNotNull = false;
        for (int i = 1; i < this.p.size(); i++) {
            int a = 0;
            for  (;  c.mod(this.p.get(i)).compareTo(BigInteger.ZERO)  ==
0; ) {
                a++;
```

```java
            c = c.divide(this.p.get(i));
        }
        if (a % 2 != 0) {
            existNotNull = true;
        }
        vStep.add(a);
        a %= 2;
        vi.add(a);
    }
    if (!existNotNull || c.compareTo(BigInteger.ONE) != 0) {
        return null;
    }
    return vi;
}

private                                          ArrayList<Integer>
findSolution(ArrayList<ArrayList<Integer>> v) {
    ArrayList<Integer> x = new ArrayList<>();
    for (int i = 0; i < v.size() - 1; i++) {
        x.add(0);
    }
    x.add(1);
    for (; ; ) {
        int j;
        for (j = 0; j < v.get(0).size(); j++) {
            int res = 0;
            for (int i = 0; i < v.size(); i++) {
                res += v.get(i).get(j) * x.get(i);
                res %= 2;
            }
            if (res != 0) {
                break;
            }
        }
        if (j == v.get(0).size()) {
            break;
        }
        int l;
        for (l = x.size() - 1; l >= 0 && x.get(l) == 1; l--) {
            x.set(l, 0);
        }
        if (l == -1) {
            return new ArrayList<>();
        }
        x.set(l, 1);
    }
    return x;
}

private                                          ArrayList<Integer>
findSolution(ArrayList<ArrayList<Integer>> v, ArrayList<Integer> x) {
    int l;
    for (l = x.size() - 1; l >= 0 && x.get(l) == 1; l--) {
        x.set(l, 0);
    }
    if (l == -1) {
        return new ArrayList<>();
    }
```

```
        x.set(l, 1);
        for (; ; ) {
            int j;
            for (j = 0; j < v.get(0).size(); j++) {
                int res = 0;
                for (int i = 0; i < v.size(); i++) {
                    res += v.get(i).get(j) * x.get(i);
                    res %= 2;
                }
                if (res != 0) {
                    break;
                }
            }
            if (j == v.get(0).size()) {
                break;
            }
            for (l = x.size() - 1; l >= 0 && x.get(l) == 1; l--) {
                x.set(l, 0);
            }
            if (l == -1) {
                return new ArrayList<>();
            }
            x.set(l, 1);
        }
        return x;
    }

    public BrillhartMorrison(BigInteger n, double a) {
        this.n = n;
        this.a = a;
        if (n.mod(BigInteger.TWO).equals(BigInteger.ZERO)) {
            d[0] = BigInteger.TWO;
            d[1] = n.divide(BigInteger.TWO);
            return;
        }
        if (n.mod(new BigInteger("3")).equals(BigInteger.ZERO)) {
            d[0] = new BigInteger("3");
            d[1] = n.divide(d[0]);
            return;
        }
        if (n.mod(new BigInteger("5")).equals(BigInteger.ZERO)) {
            d[0] = new BigInteger("5");
            d[1] = n.divide(d[0]);
            return;
        }
        if (n.mod(new BigInteger("7")).equals(BigInteger.ZERO)) {
            d[0] = new BigInteger("7");
            d[1] = n.divide(d[0]);
            return;
        }
        BigInteger square = n.sqrt();
        if (square.multiply(square).equals(n)){
            d[0] = square;
            d[1] = square;
            return;
        }
        this.generateFactorBase();
        ContinuedFractionSqrt ch = new ContinuedFractionSqrt(n);
```

```
        for (; ; ) {
            ArrayList<ArrayList<Integer>> v = new ArrayList<>();
            ArrayList<ArrayList<Integer>> vStep = new ArrayList<>();
            ArrayList<BigInteger> P = new ArrayList<>();
            ArrayList<BigInteger> Qm = new ArrayList<>();
            for (; v.size() != this.k + 1; ) {
                BigInteger[] PQ = ch.getLastAndGenerateNext();
                BigInteger                    Qmi                    =
PQ[0].pow(2).subtract(this.n.multiply(PQ[1].pow(2)));
                ArrayList<Integer> viStep = new ArrayList<>();
                ArrayList<Integer>  vi  =  this.factorizationBase(Qmi,
viStep);
                if (vi == null) {
                    continue;
                }
                Qm.add(Qmi);
                P.add(PQ[0]);
                v.add(vi);
                vStep.add(viStep);
            }
            ArrayList<Integer> x = this.findSolution(v);
            if (x.isEmpty()) {
//                    System.out.println("Не нашлось ни одного решения
уравнения");
                continue;
            }
            for (; ; ) {
                x = this.findSolution(v, x);
                if (x.isEmpty()) {
//                    System.out.println("Не нашлось ни одного решения
уравнения");
                    break;
                }
                BigInteger X = BigInteger.ONE;
                BigInteger Y = BigInteger.ONE;
                for (int i = 0; i <= this.k; i++) {
                    X = X.multiply(P.get(i).pow(x.get(i))).mod(this.n);
                }
                for (int j = 0; j < this.k; j++) {
                    int step = 0;
                    for (int i = 0; i < x.size(); i++) {
                        step += x.get(i) * vStep.get(i).get(j);
                    }
                    step /= 2;
                    Y                                                  =
Y.multiply(this.p.get(j).pow(step)).mod(this.n);
                }

                if
(X.pow(2).mod(this.n).compareTo(Y.pow(2).mod(this.n)) != 0) {
//                    System.out.println("X и Y не прошли проверку");
                    continue;
                }
                BigInteger[]   prov   =   new   BigInteger[]{X.add(Y),
X.subtract(Y)};
                for (int i = 0; i < prov.length; i++) {
                    BigInteger gcd = prov[i].gcd(this.n);
```

```
                    if    (gcd.compareTo(BigInteger.ONE)    >    0    &&
gcd.compareTo(this.n) < 0) {
                        d[0] = gcd;
                        d[1] = this.n.divide(gcd);
                        String s = "";
                        for (int j = 0; j < v.size(); j++) {
                            s += Qm.get(j) + ": ";
                            for (int l = 0; l < v.get(j).size(); l++) {
                                s += v.get(j).get(l) + " ";
                            }
                            s += "\n";
                        }
                        s += "x: ";
                        for (int j : x) {
                            s += j + " ";
                        }
                        System.out.println(s);
                        return;
                    }
                }
            }
        }
    }

    public String toString() {
        if (this.d[0] == null) {
            return "Разложение не найдено";
        }
        return this.n.toString() + " = " + this.d[0].toString() + " * "
+ this.d[1].toString();
    }

    public static void runFactorization() {
        System.out.print("Введите составное число n: ");
        Scanner in = new Scanner(System.in);
        BigInteger n = in.nextBigInteger();
        if (n.isProbablePrime(100)) {
            System.out.println("Введено простое число n");
            return;
        }
        System.out.print("Введите параметр a: ");
        double a = in.nextDouble();
        System.out.println(new BrillhartMorrison(n, a));
    }
}


import java.util.*;
import java.math.*;

public class ContinuedFractionSqrt {
    public ArrayList<BigInteger> P;
    public ArrayList<BigInteger> Q;
    private BigInteger vi, ui, n, sqrtN;

    private static BigInteger TWO = BigInteger.valueOf(2);

    public ContinuedFractionSqrt(BigInteger n) {
```

```
        this.P = new ArrayList<>();
        this.P.add(BigInteger.ZERO);
        this.P.add(BigInteger.ONE);
        this.Q = new ArrayList<>();
        this.Q.add(BigInteger.ONE);
        this.Q.add(BigInteger.ZERO);
        this.n = n;
        this.sqrtN = sqrt(n);
        this.P.add(this.sqrtN.multiply(this.P.get(this.P.size() -
1)).add(this.P.get(this.P.size() - 2)));
        this.Q.add(this.sqrtN.multiply(this.Q.get(this.Q.size() -
1)).add(this.Q.get(this.Q.size() - 2)));
        this.vi = BigInteger.ONE;
        this.ui = this.sqrtN;
        P.remove(0);
        Q.remove(0);
        this.getLastAndGenerateNext();
    }

    public BigInteger[] getLastAndGenerateNext() {
        BigInteger[] ans = new BigInteger[]{this.P.get(this.P.size() -
1), this.Q.get(this.Q.size() - 1)};
        this.vi = this.n.subtract(this.ui.pow(2)).divide(this.vi);
        BigInteger q = this.sqrtN.add(this.ui).divide(this.vi);
        this.P.add(q.multiply(this.P.get(this.P.size() -
1)).add(this.P.get(this.P.size() - 2)));
        this.Q.add(q.multiply(this.Q.get(this.Q.size() -
1)).add(this.Q.get(this.Q.size() - 2)));
        this.ui = q.multiply(this.vi).subtract(this.ui);
        P.remove(0);
        Q.remove(0);
        return ans;
    }

    public static BigInteger sqrt(BigInteger n) {
        BigInteger low = BigInteger.ZERO;
        BigInteger high = n.add(BigInteger.ONE);
        while (high.subtract(low).compareTo(BigInteger.ONE) > 0) {
            BigInteger mid = low.add(high).divide(TWO);
            if (mid.multiply(mid).compareTo(n) <= 0) {
                low = mid;
            } else {
                high = mid;
            }
        }
        return low;
    }

    private static String formatMasAnswer(String objName,
ArrayList<BigInteger> e) {
        String answer = objName + ": [";
        ArrayList<String> mas = new ArrayList<>();
        for (BigInteger i : e) {
            mas.add(i.toString());
        }
        answer += String.join(", ", mas) + "]\n";
        return answer;
    }
```

```java
    public String toString() {
        String answer = "";
        answer += formatMasAnswer("P", this.P);
        answer += formatMasAnswer("Q", this.Q);
        return answer;
    }
}


import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.Random;
import java.util.Scanner;

public class P1Pollard {
    public BigInteger n, B, T;
    private BigInteger qi;
    private int ki;
    public BigInteger[] d = new BigInteger[2];
    private static BigDecimal TWO = BigDecimal.valueOf(2);
    public static void runPollard() {
        System.out.print("Введите составное число n: ");
        Scanner in = new Scanner(System.in);
        BigInteger n = in.nextBigInteger();
        if (n.isProbablePrime(100)) {
            System.out.println("Введено простое число n");
            return;
        }
        System.out.print("Введите число B: ");
        BigInteger B = in.nextBigInteger();
        System.out.println(new P1Pollard(n, B));
    }

    private double log(BigInteger c) {
        int t = c.bitLength();
        BigDecimal s = new BigDecimal(c).divide(TWO.pow(t));
        return Math.log(s.doubleValue()) + t * Math.log(2);
    }
    private void generateFactorBase() {
        this.T = BigInteger.ONE;
        for (BigInteger p = BigInteger.valueOf(2); p.compareTo(this.B)
<= 0; p = p.nextProbablePrime()) {
            this.qi = p;
            this.ki = (int) (log(this.n) / log(this.qi));
            this.T = this.T.multiply(this.qi.pow(this.ki));
        }
    }

    private void addB() {
        this.B = this.B.nextProbablePrime();
        this.qi = this.qi.nextProbablePrime();
        this.ki = (int) (log(this.n) / log(this.qi));
        this.T = this.T.multiply(this.qi.pow(this.ki));
    }

    private static BigInteger previousProbablePrime(BigInteger n) {
        BigInteger res = n.subtract(BigInteger.ONE);
```

```
        for (; !res.isProbablePrime(100);) {
            res = res.subtract(BigInteger.ONE);
        }
        return res;
    }

    private void subB() {
        this.T = this.T.divide(this.qi.pow(this.ki));
        this.B = previousProbablePrime(this.B);
        this.qi = previousProbablePrime(this.qi);
        this.ki = (int) (log(this.n) / log(this.qi));
    }

    public P1Pollard(BigInteger n, BigInteger B) {
        this.n = n;
        this.B = B;
        this.generateFactorBase();
        Random rand = new Random();
        boolean flag = false;
        for (;;) {
            BigInteger a = new BigInteger(n.bitLength(), rand).mod(n);
            BigInteger d = a.gcd(n);
            if (d.compareTo(BigInteger.ONE) > 0 && d.compareTo(n) < 0)
{
                this.d[0] = d;
                this.d[1] = n.divide(d);
                return;
            }
            if (d.compareTo(BigInteger.ONE) == 0) {
                BigInteger         b         =         a.modPow(this.T,
this.n).subtract(BigInteger.ONE).mod(this.n);
                BigInteger n1 = b.gcd(this.n);
                if (n1.compareTo(BigInteger.ONE) == 0) {
                    this.addB();
                    continue;
                }
                if (n1.compareTo(n) == 0) {
                    if (flag) {
                        this.subB();
                        flag = false;
                        continue;
                    }
                    flag = true;
                    continue;
                }
                this.d[0] = n1;
                this.d[1] = n.divide(n1);
                return;
            }
        }
    }

    public String toString() {
        if (this.d[0] == null) {
            return "Разложение не найдено";
        }
        return this.n.toString() + " = " + this.d[0].toString() + " * "
+ this.d[1].toString();
```

```
    }

}


import java.math.BigInteger;
import java.util.Random;
import java.util.Scanner;

public class RhoPollard {
    public BigInteger n;
    public double eps;
    public BigInteger[] d = new BigInteger[2];

    private static BigInteger TWO = BigInteger.TWO;

    public static void runPollard() {
        System.out.print("Введите составное число n: ");
        Scanner in = new Scanner(System.in);
        BigInteger n = in.nextBigInteger();
        if (n.isProbablePrime(100)) {
            System.out.println("Введено простое число n");
            return;
        }
        System.out.print("Введите число 0 < eps < 1: ");
        double eps = in.nextDouble();
//        System.out.print("Введите число T: ");
//        long T = in.nextLong();
        if (eps <= 0 || eps >= 1) {
            System.out.println("Введено неверное eps");
            return;
        }
//        System.out.println(new RhoPollard(n, T));
        System.out.println(new RhoPollard(n, eps));
    }

    public static BigInteger sqrt(BigInteger n) {
        BigInteger low = BigInteger.ZERO;
        BigInteger high = n.add(BigInteger.ONE);
        while (high.subtract(low).compareTo(BigInteger.ONE) > 0) {
            BigInteger mid = low.add(high).divide(TWO);
            if (mid.multiply(mid).compareTo(n) <= 0) {
                low = mid;
            } else {
                high = mid;
            }
        }
        return low;
    }

    private static BigInteger f(BigInteger x) {
        return x.multiply(x).add(BigInteger.ONE);
    }

//    public RhoPollard(BigInteger n, long T) {
    public RhoPollard(BigInteger n, double eps) {
        this.n = n;
        this.eps = eps;
```

```
        double   T   =   Math.sqrt(TWO.multiply(sqrt(n)).intValue()   *
Math.log(1 / eps)) + 1;
        Random rand = new Random();
        for (; ; ) {
            BigInteger xi = new BigInteger(n.bitLength(), rand).mod(n);
            BigInteger xk = xi;
            long i;
            for (i = 0; i <= T; i++) {
                xi = f(xi).mod(n);
                BigInteger dk = xi.subtract(xk).gcd(n);
                if ((i & (i - 1)) == 0) {
                    xk = xi;
                }
                if (dk.compareTo(BigInteger.ONE) == 0) {
                    continue;
                }
                if (dk.compareTo(n) == 0) {
                    break;
                }
                this.d[0] = dk;
                this.d[1] = n.divide(dk);
                return;
            }
            if (i <= T) {
                continue;
            }
            break;
        }
    }

    public String toString() {
        if (this.d[0] == null) {
            return "Разложение не найдено";
        }
        return this.n.toString() + " = " + this.d[0].toString() + " * "
+ this.d[1].toString();
    }

}


import java.math.*;

public class JacobiSymbol {

    public BigInteger a, p;
    public int value;
    private static BigInteger NEG_ONE = BigInteger.ONE.negate();
    private static BigInteger TWO = BigInteger.TWO;
    private static BigInteger EIGHT = BigInteger.valueOf(8);

    public JacobiSymbol(BigInteger a, BigInteger p) {
        this.a = new BigInteger(a.toString());
        this.p = new BigInteger(p.toString());
        this.value = 1;
        if (this.a.compareTo(BigInteger.ZERO) < 0) {
            this.a = this.a.negate();
```

```
            this.value                                                *=
NEG_ONE.pow(this.p.subtract(BigInteger.ONE).divide(TWO).mod(TWO).intVa
lue()).intValue();
        }
        if (this.a.gcd(this.p).compareTo(BigInteger.ONE) != 0) {
            this.value = 0;
            return;
        }
        this.a = this.a.mod(this.p);
        for (; this.a.compareTo(BigInteger.ONE) != 0; ) {
            if (this.a.mod(TWO).compareTo(BigInteger.ZERO) == 0) {
                long t = 0;
                for (; this.a.mod(TWO).compareTo(BigInteger.ZERO) == 0;
) {
                    this.a = this.a.divide(TWO);
                    t++;
                }
                if (t % 2 != 0) {
                    this.value                                        *=
NEG_ONE.pow(this.p.pow(2).subtract(BigInteger.ONE).divide(EIGHT).mod(T
WO).intValue()).intValue();
                }
                continue;
            }
            int                          tmp                           =
this.a.subtract(BigInteger.ONE).divide(TWO).mod(TWO).intValue()       *
this.p.subtract(BigInteger.ONE).divide(TWO).mod(TWO).intValue();
            this.value *= NEG_ONE.pow(tmp).intValue();
            BigInteger tmpBig = this.p;
            this.p = this.a;
            this.a = tmpBig;
            this.a = this.a.mod(this.p);
        }
        this.a = new BigInteger(a.toString());
        this.p = new BigInteger(p.toString());
    }

    public String toString() {
        return "(" + a + "/" + p + ") = " + value;
    }
}
```