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

Факторизация целых чисел

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

Арбузова Матвея Александровича

Преподаватель

профессор, д.ф.-м.н. ___________ В. А. Молчанов

подпись, дата

Саратов 2023

1 Постановка задачи

Целью данной лабораторной работы является изучение основных методов факторизации целых чисел и их программная реализация.

Порядок выполнения работы:

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 по рассмотренным темам, алгоритмы и их сложности

Разложение целых чисел на множители

Существуют следующие алгоритмы факторизации чисел:

1) экспоненциально зависящие от длины позиционной записи числа n;

2) субэкспоненциальные алгоритмы, имеющие оценку сложности вида

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

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 $logn$.

Все современные алгоритмы факторизации субэкспоненциальны.

Экспоненциальные алгоритмы факторизации

$\boldsymbol{\rho}$-метод Полларда

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

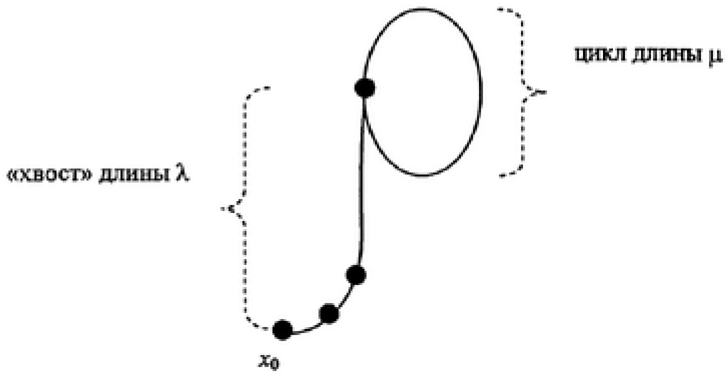
$$1 < \text{НОД}(x_i - x_k, n) < n.$$

Так как составное число n имеет простой делитель $p < \sqrt{n}$, то последовательность $\{x_i\}$ имеет период $\leq n$ и последовательность $\{x_i(mod\ p)\}$ имеет период $\leq p$. 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 $\leq p$. Из-за такой фигуры метод называется ρ-методом.



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 $P_{n,k} < e^{-\lambda}$.

Алгоритм факторизации целых чисел ρ-методом Полларда

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 $T = \left\lceil\sqrt{2\sqrt{n}\ln\frac{1}{\varepsilon}}\right\rceil + 1$ 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 $x_{i+1} = f(x_i)(mod\ n), 0 \leq i \leq T$, проверять тест на шаге 3.

Шаг 3. Для каждого $0 \leq k \leq i$ 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 найдется $d_k = n$ для некоторого $0 \leq k \leq i$, то перейти к выбору нового значения $x_0 \in \boldsymbol{Z}_n$ на шаге 2. Если же $d_k = 1$ для всех $0 \leq k \leq i$, то перейти к выбору следующего значения последовательности на шаге 2.

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

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

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 $O(\sqrt[4]{n}\ln\frac{1}{\varepsilon}\log^2 n)$.

(p-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

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

Алгоритм факторизации целых чисел ($p - 1$)-методом Полларда

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 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.

Субэкспоненциальные алгоритмы факторизации

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 $1 <$ НОД$(x - y, n) < n$, $1 <$ 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 НОД$(x - y, n)$, 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 $Q(m_i) = p_1^{\alpha_{i1}} ... p_k^{\alpha_{ik}}$, обозначаем $\overline{v_\iota} = (\alpha_{i1}, .., \alpha_{ik})$.

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 $1 < \text{НОД}(X \pm Y, n) < n$. 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 $n \in \boldsymbol{N}, n > 16, \sqrt{n} \notin \boldsymbol{N}$ 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 $D \in \boldsymbol{N},\ \sqrt{D} \notin \boldsymbol{N}, v \in \boldsymbol{N}, u \in \boldsymbol{N}, v|(D^2 - u)$. Тогда для любого $k \geq 0$ справедливо разложение в бесконечную цепную дробь $\alpha = [a_0, a_1, \dots, a_k, a_{k+1}, \dots]$, где $a_0 \in\ \boldsymbol{Z}, a_1, \dots, a_k \dots \in \boldsymbol{N}$. При этом справедливы соотношения

$$a_0 = [\alpha], v_0 = v, u_0 = u + a_0 v$$

и при $k \geq 0$

$$a_{k+1} = [\alpha_{k+1}], \text{где } v_{k+1} = \frac{D-u_k^2}{v_k} \in \boldsymbol{Z}, v_{k+1} \neq 0,$$

$$\alpha_{k+1} = \frac{\sqrt{D}+u_k}{v_{k+1}} > 1$$

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

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

так как $p|Q(m_i) = P_i^2 - nQ_i^2$ влечёт

$$P_i^2 - nQ_i^2 \equiv 0 (mod\ p), P_i^2 \equiv nQ_i^2 (mod\ p)$$

и в силу НОД$(P_i, Q_i) = 1$ выполняется:

$p \nmid \mathrm{P}_i, p \nmid Q_i$, НОД$(p, Q_i) = 1$, существует Q_i^{-1} в группе $\boldsymbol{Z}_p^*$ и $n \equiv (P_iQ_i^{-1})^2 (mod\ p)$, $(\frac{n}{p}) = 1$, т. е. $n \in QR_p$.

При этом $|Q(m_i)| = |P_i^2 - nQ_i^2| < 2\sqrt{n}$ – повышает вероятность B-гладкости значения $Q(m_i)$.

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

3 Практическая реализация

3.1 Псевдокоды рассмотренных алгоритмов

Псевдокод алгоритма факторизации целых чисел ρ-методом Полларда

```
Ввод n,ε, где n – число, которое нужно факторизовать
Если (n < 1)
      вывести «Число n должно быть больше 0»
Если (n простое)
      вывести «Число n, вероятно, простое»
Если (не выполняется 0 < ε < 1)
      вывести «ε должно удовлетворять условию: 0 < ε < 1»
Вывести результат функции roPollard(n,ε)


Функция roPollard(n,ε):
      T = ⌊√(2√n ln(1/ε))⌋ + 1
      В бесконечном цикле, пока не будет условия выхода, выполнять:
            x_0 = случайному числу от 0 до n−1
            В цикле по i от 0 до T:
                  x_{i+1} = x_i^2 (mod n)
                  В цикле по k от 0 до i:
                        d_k = НОД(x_{i+1} − x_k, n)
                        Если (1 < d_k < n)
                              Вывести в качестве результата d_k и n/d_k
                              Закончить все циклы
                        Если (d_k = n)
                              Перейти к следующей итерации бесконечного
                              Цикла
                        Если (все d_k = 1)
                              Перейти к следующей итерации цикла по i
```

Псевдокод алгоритма факторизации целых чисел $(p-1)$-методом Полларда

```
Ввод n,B, где n – число, которое нужно факторизовать, B – верхняя граница
факторной базы
Если (n < 1)
      вывести «Число n должно быть больше 0»
Если (n простое)
```

вывести «Число n, вероятно, простое»
Если $(B < 1)$
вывести «B должно быть больше 0»
Вывести результат функции $pm1Pollard(n, B)$

Функция $Base(n, B)$:
В список q кладутся все простые числа $q_i \le B$ (их количество обозначается через $\pi(B)$)
Вывести $T = \prod_{i=1}^{\pi(B)} q_i^{\left[\frac{\ln n}{\ln q_i}\right]}$

Функция $pm1Pollard(n, B)$:
$checkA = 0$ (счётчик a, при которых $n_1 = n$)
$T =$ результату функции $Base(n, B)$
В бесконечном цикле, пока не будет условия выхода, выполнять:
$a =$ случайному числу от 0 до $n - 1$
$checkA = checkA + 1$
$d = \text{НОД}(a, n)$
Если $(d = 1)$
$b = a^T - 1 \ (mod\ n)$
$n_1 = \text{НОД}(b, n)$
Если $(1 < n_1 < n)$
Вывести в качестве результата n_1 и $\frac{n}{n_1}$
Закончить все циклы
Если $(n_1 = n)$
Если $(checkA \neq 2)$
Перейти к следующей итерации цикла
Иначе
$B = B - 1$
$T =$ результату функции $Base(n, B)$
$checkA = 0$
Если $(n_1 = 1)$
$B = B + 1$
$T =$ результату функции $Base(n, B)$

Псевдокод алгоритма факторизации целых методом цепных дробей

Ввод n, a, где n – число, которое нужно факторизовать, a – параметр, определяющий верхнюю границу L^a факторной базы B
Если $(n < 1)$
вывести «Число n должно быть больше 0»
Если (n простое)
вывести «Число n, вероятно, простое»
Если (n делится на один из маленьких простых делителей $(2, 3, 5, 7, 11, 13, 17)$)
вывести «Число n имеет небольшой простой делитель»
Если (n является квадратом некоторого числа)
вывести «Число n является квадратом числа»
Если (a не удовлетворяет условию $0 < a < 1$)
вывести «Число a должно удовлетворять условию: $0 < a < 1$»
Вывести результат функции $BrillhartMorrison(n, a)$

Функция $BrillhartMorrison(n,a)$:

$L = \exp((lognloglogn)^a)$

В факторную базу B кладётся -1, после чего кладутся все простые числа $p_i \leq L$ такие, что символ Якоби $\left(\frac{p_i}{n}\right) \neq -1$

$k = \pi(B)$ – число элементов в B

В бесконечном цикле, пока не встретиться выход выполнять:

Вычислить числитель P_i и знаменатель Q_i подходящей дроби $Q(m_i) = P_i^2 - nQ_i^2$.

Вычисляются $\overline{v_i} = (\alpha_{i1},..,\alpha_{ik})$ и $\overline{e_i} = (\alpha_{i1}(mod\ 2),..,\alpha_{ik}(mod\ 2))$, их должно быть $k+1$ штук

Вычисляется ненулевое решение $(x_1,..,x_{k+1}) \in \{0,1\}^{k+1}$ системы k линейных уравнений с $k+1$ неизвестными $x_1\overline{v_1} + .. + x_{k+1}\overline{v_{k+1}} = \bar{0}\ (mod\ 2)$

Если (ни одного решения x не найдено)

Увеличить факторную базу и перейти к следующей итерации бесконечного цикла

Если (решение x найдено)

$X \equiv P_1^{x_1}..P_{k+1}^{x_{k+1}}\ (mod\ n)$

$Y \equiv \prod_{j=1}^{k} p_j^{\frac{\sum x_i\alpha_{ij}}{2}}\ (mod\ n)$

Если $(X^2\ (mod\ n) = Y^2 (mod\ n))$

Если (НОД$(X+Y,n)$ или НОД$(X-Y,n)$ в промежутке от 0 до n)

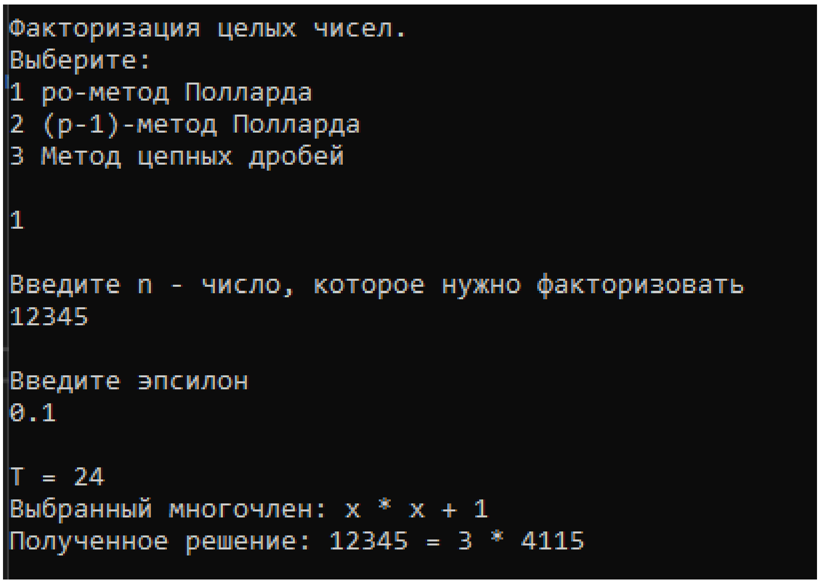
Вывести решение НОД$(X+Y,n)$ и $\frac{n}{\text{НОД}(X+Y,n)}$ или НОД$(X-Y,n)$ *и* $\frac{n}{\text{НОД}(X-Y,n)}$

Иначе найти другое решение x

Иначе найти другое решение x

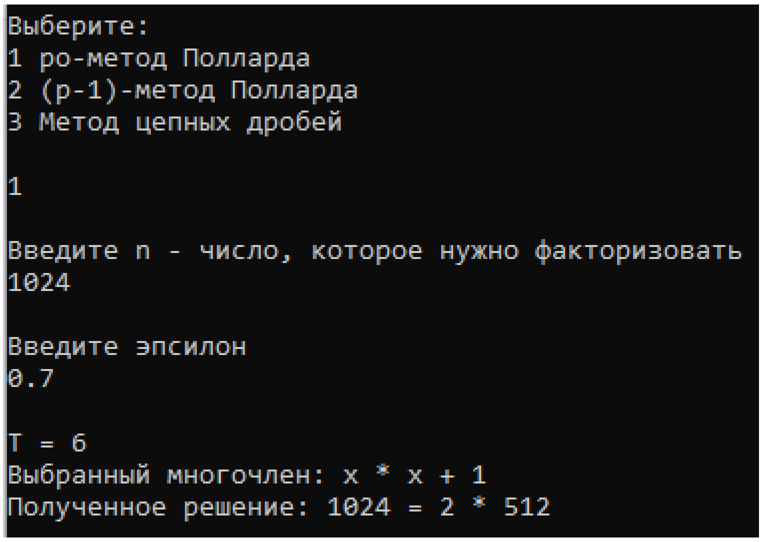
3.2 Результаты тестирования программы

Тестирование алгоритма факторизации целых чисел ρ-методом Полларда (рисунки 1-2).



```
Факторизация целых чисел.
Выберите:
1 ро-метод Полларда
2 (p-1)-метод Полларда
3 Метод цепных дробей

1

Введите n - число, которое нужно факторизовать
12345

Введите эпсилон
0.1

T = 24
Выбранный многочлен: x * x + 1
Полученное решение: 12345 = 3 * 4115
```

Рисунок 1 – Результат первого теста ρ-метода Полларда



```
Выберите:
1 ро-метод Полларда
2 (p-1)-метод Полларда
3 Метод цепных дробей

1

Введите n - число, которое нужно факторизовать
1024

Введите эпсилон
0.7

T = 6
Выбранный многочлен: x * x + 1
Полученное решение: 1024 = 2 * 512
```

Рисунок 2 – Результат второго теста ρ-метода Полларда

Тестирование алгоритма факторизации целых чисел $(p-1)$-методом Полларда (рисунки 3-4).

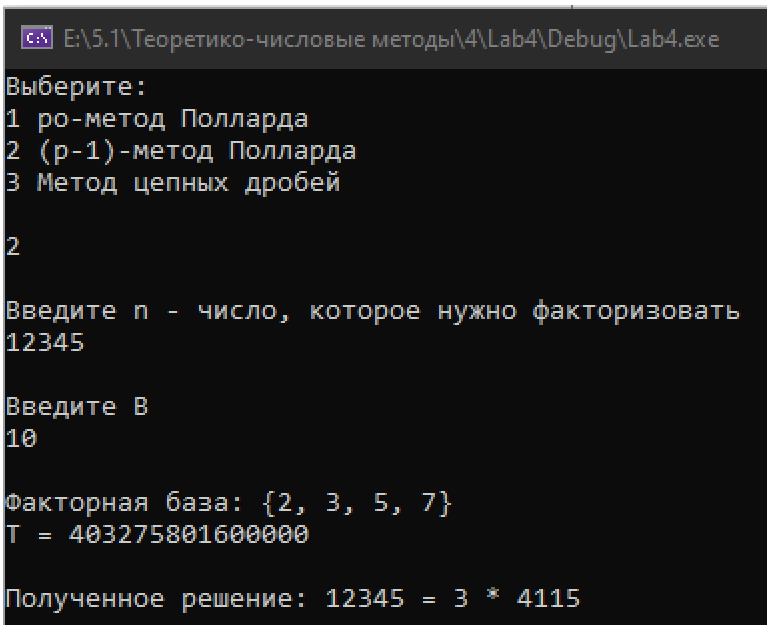


Рисунок 3 – Результат первого теста $(\boldsymbol{p-1})$-метода Полларда

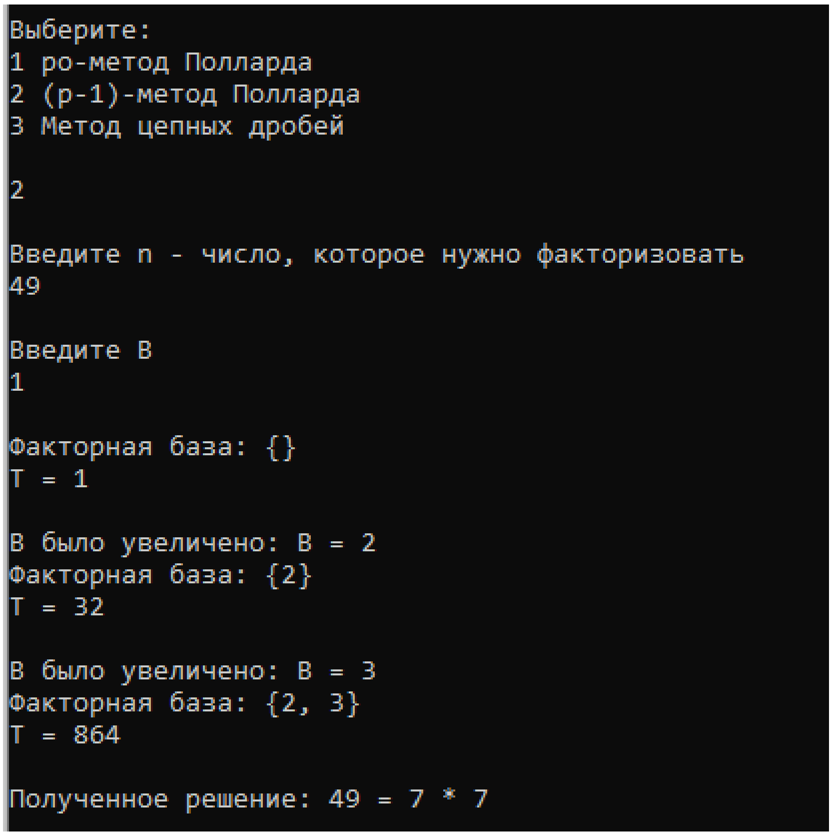


Рисунок 4 – Результат второго теста $(\boldsymbol{p-1})$-метода Полларда

Тестирование алгоритма факторизации целых чисел методом цепных дробей (рисунки 5-6).

```
3 Метод цепных дробей

3

Введите n - число, которое нужно факторизовать
21299881

Введите a
0.1

L = 117
B = {-1, 2, 3, 5, 7, 11, 19, 23, 53, 89, 101, 107, 109}
-4235: 1 0 0 1 1 0 0 0 0 0 0 0 0
2688: 0 1 1 0 1 0 0 0 0 0 0 0 0
-7920: 1 0 0 1 0 1 0 0 0 0 0 0 0
321: 0 0 1 0 0 0 0 0 0 0 0 1 0
385: 0 0 0 1 1 1 0 0 0 0 0 0 0
-3800: 1 1 0 0 0 0 1 0 0 0 0 0 0
-1331: 1 0 0 0 0 1 0 0 0 0 0 0 0
5415: 0 0 1 1 0 0 0 0 0 0 0 0 0
-3424: 1 1 0 0 0 0 0 0 0 0 0 1 0
5313: 0 0 1 0 1 1 0 1 0 0 0 0 0
7521: 0 0 1 0 0 0 0 1 0 0 0 0 1
-112: 1 0 0 0 1 0 0 0 0 0 0 0 0
5415: 0 0 1 1 0 0 0 0 0 0 0 0 0
-3795: 1 0 1 1 0 1 0 1 0 0 0 0 0
x: 0 0 0 0 0 0 0 1 0 0 0 0 1 0

Полученное решение: 21299881 = 3851 * 5531
```

Рисунок 5 – Результат первого теста метода цепных дробей

```
Введите n - число, которое нужно факторизовать
1457

Введите a
0.2

L = 18
B = {-1, 2, 7, 11, 13}
56: 0 1 1 0 0
-11: 1 0 0 1 0
8: 0 1 0 0 0
32: 0 1 0 0 0
7: 0 0 1 0 0
7: 0 0 1 0 0
x: 0 0 0 0 1 1

Полученное решение: 1457 = 47 * 31
```

Рисунок 6 – Результат первого теста метода цепных дробей

4 Выводы по работе

В ходе выполнения лабораторной работы были рассмотрены как экспоненциальные, так и субэкспоненциальные алгоритмы факторизации целых чисел. А именно – ρ-метод Полларда, $(p-1)$-метод Полларда и метод цепных дробей, в основе которого лежит алгоритм Бриллхарта-Моррисона.

В практической части лабораторной работы была написана программа на языке C++, содержащая в себе реализацию всех описанных в теоретической части алгоритмов, работоспособность которых продемонстрирована на рисунках 1-6.

5 Код программы

```
#include <iostream>
#include <vector>
#include <random>
#include <boost/multiprecision/cpp_int.hpp>
#include <boost/math/constants/constants.hpp>
#include <boost/multiprecision/cpp_dec_float.hpp>
#include <boost/random/uniform_int.hpp>
#include <boost/random/variate_generator.hpp>

using namespace std;
using namespace boost::multiprecision;
namespace mp = boost::multiprecision;

cpp_int NegativeMod(cpp_int a, cpp_int p) {
    if (a < 0)
        a = a + p * (((-1 * a) / p) + 1);
    return a % p;
}

cpp_int StandartEuclid(cpp_int a, cpp_int b) {
    if (b == 0)
        return a;
    else
        return StandartEuclid(b, a % b);
}

cpp_int Exponentiation(cpp_int x, cpp_int n, cpp_int m) {
    cpp_int N = n, Y = 1, Z = x % m;
    while (N != 0) {
        cpp_int lastN = N % 2;
        N = N / 2;
        if (lastN == 0) {
            Z = (Z * Z) % m;
            continue;
        }
        Y = (Y * Z) % m;
        if (N == 0)
            break;
        Z = (Z * Z) % m;
    }
    Y = Y % m;
    return Y;
}

cpp_int Exponentiation1(cpp_int x, cpp_int n) {
    cpp_int N = n, Y = 1, Z = x;
    while (N != 0) {
        cpp_int lastN = N % 2;
        N = N / 2;
        if (lastN == 0) {
            Z = (Z * Z);
            continue;
        }
        Y = (Y * Z);
```

```
            if (N == 0)
                break;
            Z = (Z * Z);
        }
        Y = Y;
        return Y;
    }

    cpp_int Jac(cpp_int a, cpp_int b) {
        if (StandartEuclid(a, b) != 1)
            return 0;
        else {
            cpp_int r = 1;
            while (a != 0) {
                cpp_int t = 0;
                while (a % 2 == 0) {
                    t = t + 1;
                    a = a / 2;
                }
                if (t % 2 != 0)
                    if (Exponentiation(b, 1, 8) == 3 || Exponentiation(b, 1,
8) == 5)
                        r = r * (-1);
                if (Exponentiation(a, 1, 4) == 3 && Exponentiation(b, 1, 4)
== 3)
                    r = r * (-1);
                cpp_int c = a;
                if (c != 0)
                    a = Exponentiation(b, 1, c);
                b = c;
            }
            return r;
        }
    }

    bool MilRab(cpp_int p, cpp_int k) {
        if (p == 1 || p == 2 || p == 3)
            return true;
        if (p % 2 == 0)
            return false;
        cpp_int t = p - 1;
        cpp_int s = 0;
        while (t % 2 == 0) {
            t = t / 2;
            s++;
        }
        for (cpp_int i = 0; i < k; i++) {
            cpp_int a = rand() % (p - 3) + 2;
            if (StandartEuclid(p, a) > 1)
                return false;
            cpp_int x = Exponentiation(a, t, p);
            if (x == 1 || x == p - 1)
                continue;
            for (cpp_int g = 1; g < s; g++) {
                x = x * x % p;
                if (x == 1)
```

```
                return false;
            if (x == p - 1)
                break;
        }
        if (x != p - 1)
            return false;
    }
    return true;
}

cpp_int Random(cpp_int minim, cpp_int maxim) {
    random_device gen;
    boost::random::uniform_int_distribution<cpp_int> ui(minim, maxim);
    return ui(gen);
}


pair <cpp_int, cpp_int> roPollard(cpp_int n, cpp_dec_float_50 eps) {
    cpp_dec_float_50  t  =  mp::sqrt(2  *  mp::sqrt(cpp_dec_float_50(n))  *
log(1 / eps));
    cpp_int T(t + 2);
    cpp_int x0;
    cout << "\nT = " << T << "\n";
    cout << "Выбранный многочлен: x * x + 1\n";
    vector <cpp_int> xs, ds;
    for (;;) {
        xs.clear();
        xs.push_back(Random(0, n - 1));
        for (int i = 0; i <= T; i++) {
            xs.push_back((xs[i] * xs[i] + 1) % n);
            bool GoNewX0 = false;
            ds.clear();
            for (int k = 0; k <= i; k++) {
                ds.push_back(StandartEuclid(xs[i+1]-xs[k], n));
                if (ds[k] > 1 && ds[k] < n)
                    return { ds[k], n / ds[k] };
                if (ds[k] == n) {
                    GoNewX0 = true;
                    break;
                }
            }
            if (GoNewX0)
                break;
        }
    }
}


void roPolInit() {
    cpp_int n;
    cpp_dec_float_50 eps;
    pair <cpp_int, cpp_int> resultat;
    cout << "\nВведите n - число, которое нужно факторизовать\n";
    cin >> n;
    if (n < 1) {
        cerr << "\nn должно быть больше 0\n";
```

```
            return;
        }
        srand(time(0));
        if (MilRab(n, 10)) {
            cout << "\nВведённое n, вероятно, простое\n";
            return;
        }
        cout << "\nВведите эпсилон\n";
        cin >> eps;
        if (eps > 0 && eps < 1) {
        }
        else {
            cout << "\nЭпсилон должно удовлетворять условию: 0 < эпсилон < 1\n";
            return;
        }
        resultat = roPollard(n, eps);
        cout << "Полученное решение: "<<  n << " = " << resultat.first << " * " << resultat.second << "\n";
        return;
    }

    vector <cpp_int> allPrime = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
            71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
            179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281,
            283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
            419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541,
            547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659,
            661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809,
            811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941,
            947, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061,
            1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187,
            1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297, 1301,
            1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447,
            1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543, 1549, 1553, 1559,
            1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667, 1669,
            1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811,
            1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951,
            1973, 1979, 1987, 1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083,
```

2087, 2089, 2099, 2111, 2113, 2129, 2131, 2137, 2141, 2143, 2153,
2161, 2179, 2203, 2207, 2213, 2221, 2237, 2239, 2243, 2251, 2267,
2269, 2273, 2281, 2287, 2293, 2297, 2309, 2311, 2333, 2339, 2341, 2347,
2351, 2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423,
2437, 2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543,
2549, 2551, 2557, 2579, 2591, 2593, 2609, 2617, 2621, 2633, 2647, 2657,
2659, 2663, 2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713,
2719, 2729, 2731, 2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801,
2803, 2819, 2833, 2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903,
2909, 2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001, 3011,
3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083, 3089, 3109, 3119,
3121, 3137, 3163, 3167, 3169, 3181, 3187, 3191, 3203, 3209, 3217, 3221,
3229, 3251, 3253, 3257, 3259, 3271, 3299, 3301, 3307, 3313, 3319, 3323,
3329, 3331, 3343, 3347, 3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413,
3433, 3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517, 3527,
3529, 3533, 3539, 3541, 3547, 3557, 3559, 3571, 3581, 3583, 3593, 3607,
3613, 3617, 3623, 3631, 3637, 3643, 3659, 3671, 3673, 3677, 3691, 3697,
3701, 3709, 3719, 3727, 3733, 3739, 3761, 3767, 3769, 3779, 3793, 3797,
3803, 3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907,
3911, 3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001, 4003,
4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079, 4091, 4093,
4099, 4111, 4127, 4129, 4133, 4139, 4153, 4157, 4159, 4177, 4201, 4211,
4217, 4219, 4229, 4231, 4241, 4243, 4253, 4259, 4261, 4271, 4273, 4283,
4289, 4297, 4327, 4337, 4339, 4349, 4357, 4363, 4373, 4391, 4397, 4409,
4421, 4423, 4441, 4447, 4451, 4457, 4463, 4481, 4483, 4493, 4507, 4513,
4517, 4519, 4523, 4547, 4549, 4561, 4567, 4583, 4591, 4597, 4603, 4621,
4637, 4639, 4643, 4649, 4651, 4657, 4663, 4673, 4679, 4691, 4703, 4721,
4723, 4729, 4733, 4751, 4759, 4783, 4787, 4789, 4793, 4799, 4801, 4813,
4817, 4831, 4861, 4871, 4877, 4889, 4903, 4909, 4919, 4931, 4933, 4937,
4943, 4951, 4957, 4967, 4969, 4973, 4987, 4993, 4999, 5003, 5009, 5011,
5021, 5023, 5039, 5051, 5059, 5077, 5081, 5087, 5099, 5101, 5107, 5113,
5119, 5147, 5153, 5167, 5171, 5179, 5189, 5197, 5209, 5227, 5231, 5233,
5237, 5261, 5273, 5279, 5281, 5297, 5303, 5309, 5323, 5333, 5347, 5351,
5381, 5387, 5393, 5399, 5407, 5413, 5417, 5419, 5431, 5437, 5441, 5443,
5449, 5471, 5477, 5479, 5483, 5501, 5503, 5507, 5519, 5521, 5527, 5531,
5557, 5563, 5569, 5573, 5581, 5591, 5623, 5639, 5641, 5647, 5651, 5653,
5657, 5659, 5669, 5683, 5689, 5693, 5701, 5711, 5717, 5737, 5741, 5743,
5749, 5779, 5783, 5791, 5801, 5807, 5813, 5821, 5827, 5839, 5843, 5849,
5851, 5857, 5861, 5867, 5869, 5879, 5881, 5897, 5903, 5923, 5927, 5939,
5953, 5981, 5987, 6007, 6011, 6029, 6037, 6043, 6047, 6053, 6067, 6073,
6079, 6089, 6091, 6101, 6113, 6121, 6131, 6133, 6143, 6151, 6163, 6173,
6197, 6199, 6203, 6211, 6217, 6221, 6229, 6247, 6257, 6263, 6269, 6271,
6277, 6287, 6299, 6301, 6311, 6317, 6323, 6329, 6337, 6343, 6353, 6359,
6361, 6367, 6373, 6379, 6389, 6397, 6421, 6427, 6449, 6451, 6469, 6473,
6481, 6491, 6521, 6529, 6547, 6551, 6553, 6563, 6569, 6571, 6577, 6581,
6599, 6607, 6619, 6637, 6653, 6659, 6661, 6673, 6679, 6689, 6691, 6701,
6703, 6709, 6719, 6733, 6737, 6761, 6763, 6779, 6781, 6791, 6793, 6803,
6823, 6827, 6829, 6833, 6841, 6857, 6863, 6869, 6871, 6883, 6899, 6907,
6911, 6917, 6947, 6949, 6959, 6961, 6967, 6971, 6977, 6983, 6991, 6997,
7001, 7013, 7019, 7027, 7039, 7043, 7057, 7069, 7079, 7103, 7109, 7121,
7127, 7129, 7151, 7159, 7177, 7187, 7193, 7207, 7211, 7213, 7219, 7229,
7237, 7243, 7247, 7253, 7283, 7297, 7307, 7309, 7321, 7331, 7333, 7349,
7351, 7369, 7393, 7411, 7417, 7433, 7451, 7457, 7459, 7477, 7481, 7487,
7489, 7499, 7507, 7517, 7523, 7529, 7537, 7541, 7547, 7549, 7559, 7561,
7573, 7577, 7583, 7589, 7591, 7603, 7607, 7621, 7639, 7643, 7649, 7669,
7673, 7681, 7687, 7691, 7699, 7703, 7717, 7723, 7727, 7741, 7753, 7757,

```
7759, 7789, 7793, 7817, 7823, 7829, 7841, 7853, 7867, 7873, 7877, 7879,
7883, 7901, 7907, 7919, 7927, 7933, 7937, 7949, 7951, 7963, 7993, 8009,
8011, 8017, 8039, 8053, 8059, 8069, 8081, 8087, 8089, 8093, 8101, 8111,
8117, 8123, 8147, 8161, 8167, 8171, 8179, 8191, 8209, 8219, 8221, 8231,
8233, 8237, 8243, 8263, 8269, 8273, 8287, 8291, 8293, 8297, 8311, 8317,
8329, 8353, 8363, 8369, 8377, 8387, 8389, 8419, 8423, 8429, 8431, 8443,
8447, 8461, 8467, 8501, 8513, 8521, 8527, 8537, 8539, 8543, 8563, 8573,
8581, 8597, 8599, 8609, 8623, 8627, 8629, 8641, 8647, 8663, 8669, 8677,
8681, 8689, 8693, 8699, 8707, 8713, 8719, 8731, 8737, 8741, 8747, 8753,
8761, 8779, 8783, 8803, 8807, 8819, 8821, 8831, 8837, 8839, 8849, 8861,
8863, 8867, 8887, 8893, 8923, 8929, 8933, 8941, 8951, 8963, 8969, 8971,
8999, 9001, 9007, 9011, 9013, 9029, 9041, 9043, 9049, 9059, 9067, 9091,
9103, 9109, 9127, 9133, 9137, 9151, 9157, 9161, 9173, 9181, 9187, 9199,
9203, 9209, 9221, 9227, 9239, 9241, 9257, 9277, 9281, 9283, 9293, 9311,
9319, 9323, 9337, 9341, 9343, 9349, 9371, 9377, 9391, 9397, 9403, 9413,
9419, 9421, 9431, 9433, 9437, 9439, 9461, 9463, 9467, 9473, 9479, 9491,
9497, 9511, 9521, 9533, 9539, 9547, 9551, 9587, 9601, 9613, 9619, 9623,
9629, 9631, 9643, 9649, 9661, 9677, 9679, 9689, 9697, 9719, 9721, 9733,
9739, 9743, 9749, 9767, 9769, 9781, 9787, 9791, 9803, 9811, 9817, 9829,
9833, 9839, 9851, 9857, 9859, 9871, 9883, 9887, 9901, 9907, 9923, 9929,
9931, 9941, 9949, 9967, 9973 };

cpp_int Base(cpp_int n, cpp_int B) {

    vector <cpp_int> q;
    cout << "\nФакторная база: {";
    for (int i = 0; i < allPrime.size() - 1; i++) {
        if (allPrime[i] <= B) {
            q.push_back(allPrime[i]);
            cout << allPrime[i];
        }
        if (allPrime[i + 1] <= B)
            cout << ", ";
        else {
            cout << "}\n";
            break;
        }
    }
    cpp_int T = 1;
    for (int i = 0; i < q.size(); i++) {
        cpp_int                k(log(cpp_dec_float_50(n))                 /
log(cpp_dec_float_50(q[i])));
        T = T * Exponentiation1(q[i], k);
    }
    cout << "T = " << T << "\n";
    return T;
}


pair <cpp_int, cpp_int> pm1Pollard(cpp_int n, cpp_int B) {
    int checkA = 0;
    cpp_int T = Base(n, B);
    for (;;) {
        cpp_int a = Random(0, n - 1);
        checkA++;
        cpp_int d = StandartEuclid(a, n);
```

```
            if (d > 1 && d < n)
                return { d, n / d };
            if (d == 1) {
                cpp_int b = NegativeMod(Exponentiation(a, T, n) - 1, n);
                cpp_int n1 = StandartEuclid(b, n);
                if (n1 > 1 && n1 < n)
                    return { n1, n / n1 };
                if (n1 == n) {
                    if (checkA != 2)
                        continue;
                    B--;
                    cout << "\nB было уменьшено: B = " << B;
                    T = Base(n, B);
                    checkA = 0;
                    continue;
                }
                if (n1 == 1) {
                    B++;
                    cout << "\nB было увеличено: B = " << B;
                    T = Base(n, B);
                    continue;
                }
            }
        }
    }


    void pm1PolInit() {
        cpp_int n, B;
        pair <cpp_int, cpp_int> resultat;
        cout << "\nВведите n - число, которое нужно факторизовать\n";
        cin >> n;
        if (n < 1) {
            cerr << "\nn должно быть больше 0\n";
            return;
        }
        srand(time(0));
        if (MilRab(n, 10)) {
            cout << "\nВведённое n, вероятно, простое\n";
            return;
        }
        cout << "\nВведите B\n";
        cin >> B;
        if (B < 1) {
            cout << "\nB должно быть > 0\n";
            return;
        }
        resultat = pm1Pollard(n, B);
        cout << "\nПолученное решение: " << n << " = " << resultat.first << "
* " << resultat.second << "\n";
        return;
    }

    vector <mp::cpp_int> P, Q;
    mp::cpp_int vi, ui;
```

```
vector <cpp_int> CFStep(cpp_int n) {
    cpp_int sqrtN(sqrt(n));
    vector <cpp_int> res = { P[P.size() - 1], Q[Q.size() - 1] };
    vi = (n - (ui * ui)) / vi;
    cpp_int q = (sqrtN + ui) / vi;
    P.push_back(q * P[P.size() - 1] + P[P.size() - 2]);
    Q.push_back(q * Q[Q.size() - 1] + Q[Q.size() - 2]);
    ui = q * vi - ui;
    P.erase(P.begin());
    Q.erase(Q.begin());
    return res;
}

vector <cpp_int> CFStart(cpp_int n) {
    P = { 0, 1 };
    Q = { 1, 0 };
    cpp_int sqrtN(sqrt(n));
    P.push_back(sqrtN * P[P.size()- 1] + P[P.size() - 2]);
    Q.push_back(sqrtN * Q[Q.size() - 1] + Q[Q.size() - 2]);
    vi = 1;
    ui = sqrtN;
    P.erase(P.begin());
    Q.erase(Q.begin());
    return CFStep(n);
}

pair <vector <cpp_int>, vector <cpp_int>> FB(cpp_int Qmi, vector <cpp_int> vStep, vector <cpp_int> B) {
    vector <cpp_int> vi;
    if (Qmi < 0) {
        vi.push_back(1);
        vStep.push_back(1);
        Qmi = -1 * Qmi;
    }
    else {
        vi.push_back(0);
        vStep.push_back(0);
    }
    bool checker = false;
    for (int i = 1; i < B.size(); i++) {
        int a = 0;
        while(Qmi % B[i] == 0) {
            a++;
            Qmi = Qmi / B[i];
        }
        if (a % 2 != 0)
            checker = true;
        vStep.push_back(a);
        a %= 2;
        vi.push_back(a);
    }
    if (!checker || Qmi != 1)
        return {};
    return { vi, vStep };
}
```

```
vector <cpp_int> Solution(vector<vector <cpp_int>> v){
    vector <cpp_int> x(v.size() - 1, 0);
    x.push_back(1);
    for (;;) {
        int j;
        for (j = 0; j < v[0].size(); j++) {
            cpp_int res = 0;
            for (int i = 0; i < v.size(); i++)
                res = (res + v[i][j] * x[i]) % 2;
            if (res != 0)
                break;
        }
        if (j == v[0].size())
            break;
        int l;
        for (l = x.size() - 1; l >= 0 && x[l] == 1; l--)
            x[l] = 0;
        if (l == -1)
            return {};
        x[l] = 1;
    }
    return x;
}


pair<cpp_int, cpp_int> BrillhartMorrison(cpp_int n, cpp_dec_float_50 a) {
    cpp_dec_float_50    l    =    pow(mp::exp(log(cpp_dec_float_50(n))    *
log(log(cpp_dec_float_50(n)))), a);
    cpp_int L(l);
    cout << "\nL = " << L << "\n";
    vector <cpp_int> B = { -1 };
    int g = 0;
    while (g < allPrime.size() && allPrime[g] <= L) {
        if (Jac(n, allPrime[g]) != -1)
            B.push_back(allPrime[g]);
        g++;
    }
    cout << "B = {";
    for (int i = 0; i < B.size(); i++) {
        if (i != B.size() - 1)
            cout << B[i] << ", ";
        else
            cout << B[i] << "}\n";
    }
    int k = B.size();

    vector <cpp_int> ch = CFStart(n);
    for (;;) {
        vector <vector <cpp_int>> v, vStep;
        vector <cpp_int> newP, Qm;
        int counter = 0;
        while (v.size() != k + 1) {
            vector <cpp_int> PQ = CFStep(n);
            cpp_int Qmi = PQ[0] * PQ[0] - (n * (PQ[1] * PQ[1]));
            vector <cpp_int> viStep;
```

```
                    pair  <vector  <cpp_int>,  vector  <cpp_int>>  resFB  =  FB(Qmi,
viStep, B);
                    counter++;
                    if (counter > 5000)
                        return { 0, 0 };
                    if (resFB.first.size() == 0)
                        continue;
                    Qm.push_back(Qmi);
                    newP.push_back(PQ[0]);
                    v.push_back(resFB.first);
                    vStep.push_back(resFB.second);

                }
                vector <cpp_int> x;
                cpp_int fakt = 1;
                for (int sad = 1; sad < 5; sad++) {
                    x = Solution(v);
                    if (x.size() != 0) {
                        cpp_int X = 1;
                        cpp_int Y = 1;
                        for (int i = 0; i <= k; i++)
                            X = (X * (Exponentiation(newP[i], x[i], n))) % n;
                        for (int j = 0; j < k; j++) {
                            cpp_int step = 0;
                            for (int i = 0; i < x.size(); i++)
                                step = step + x[i] * vStep[i][j];
                            step = step / 2;
                            Y = Y * (Exponentiation(B[j], step, n)) % n;
                        }
                        if (X * X % n == Y * Y % n) {
                            vector <cpp_int> prov = { X + Y, X - Y };
                            vector <cpp_int> d(2);
                            for (int i = 0; i < prov.size(); i++) {
                                cpp_int gcd = StandartEuclid(prov[i], n);
                                if (gcd > 1 && gcd < n) {
                                    d[0] = gcd;
                                    d[1] = n / gcd;
                                    string str = "";
                                    for (int j = 0; j < v.size(); j++) {
                                        str = str + to_string(Qm[j]) + ": ";
                                        for (int l = 0; l < v[j].size(); l++)
                                            str = str + to_string(v[j][l]) + " ";
                                        str += "\n";
                                    }
                                    str = str + "x: ";
                                    for (cpp_int j : x)
                                        str = str + to_string(j) + " ";
                                    cout << str << "\n";
                                    return { d[0], d[1] };
                                }
                            }
                        }

                    }
                }
                g++;
```

```
            for (int sadsadas = 1; sadsadas < 5; sadsadas++) {
                while (g < allPrime.size() && Jac(n, allPrime[g]) == -1) {
                    g++;
                }
                if (g >= allPrime.size())
                    return { 0, 0 };

                B.push_back(allPrime[g]);
            }
            cout << "B = {";
            for (int i = 0; i < B.size(); i++) {
                if (i != B.size() - 1)
                    cout << B[i] << ", ";
                else
                    cout << B[i] << "}\n";
            }
            k = B.size();
        }
    }


    void BMInit() {
        cpp_int n;
        cpp_dec_float_50 a;
        pair <cpp_int, cpp_int> resultat;
        cout << "\nВведите n - число, которое нужно факторизовать\n";
        cin >> n;
        if (n < 1) {
            cerr << "\nn должно быть больше 0\n";
            return;
        }
        srand(time(0));
        if (MilRab(n, 10)) {
            cout << "\nВведённое n, вероятно, простое\n";
            return;
        }
        vector <cpp_int> smallPrime{ 2, 3, 5, 7, 11 };
        for (int i = 0; i < smallPrime.size(); i++) {
            if (n % smallPrime[i] == 0) {
                cout << "\nЧисло n имеет небольшой простой делитель, равный "
<< smallPrime[i] << "\n";
                cout << "n = " << smallPrime[i] << " * " << n / smallPrime[i]
<< "\n";
                return;
            }
        }
        cpp_int sqrtN = sqrt(n);
        if (n == sqrtN * sqrtN){
            cout << "\nЧисло n является квадратом числа " << sqrtN << "\n";
            cout << "n = " << sqrtN << " * " << sqrtN << "\n";
            return;
        }
        cout << "\nВведите a\n";
        cin >> a;
        if (a < 0 || a >= 1) {
            cout << "\nЧисло a должно удовлетворять условию: 0 < a < 1\n";
```

```
            return;
        }
        resultat = BrillhartMorrison(n, a);
        if (resultat.first == 0)
            cout << "\nРешение не найдено\n";
        else
            cout << "\nПолученное решение: " << n << " = " << resultat.first
<< " * " << resultat.second << "\n";
        return;
    }


    int main() {
        setlocale(LC_ALL, "Russian");
        cpp_int c = 100;
        cout << "\nФакторизация целых чисел.";
        while (c != 0) {
            cout << "\nВыберите:\n";
            cout << "1 ро-метод Полларда\n";
            cout << "2 (p-1)-метод Полларда\n";
            cout << "3 Метод цепных дробей\n\n";
            cin >> c;
            if (c == 1) {
                roPolInit();
                continue;
            }
            if (c == 2) {
                pm1PolInit();
                continue;
            }
            if (c == 3) {
                BMInit();
                continue;
            }
            cerr << "Введённый вариант ответа отсутствует\n";
            c = 100;
        }
        return 0;
    }
```