Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Теоретической и прикладной математики

Расчетно-графическое задание
по дисциплине «ОСНОВЫ ТЕОРИИ ИНФОРМАЦИИИ КРИПТОГРАФИИ»

ФАКТОРИЗАЦИЯ СОСТАВНОГО ЧИСЛА

Факультет:	ПМИ
Группа:	ПМИ-02
Вариант:	7
Студент:	Сидоров Даниил,
	Дюков Богдан
Преподаватель:	Авдеенко Татьяна Владимировна,
	Сивак Мария Алексеевна.

Новосибирск

2026

1. Цель работы

Освоить простые алгоритмы факторизации составного числа.

2. Задача

Часть 1

I. Реализовать приложение, удовлетворяющее следующим требованиям.
1. Во входном файле хранятся входные данные, необходимые для работы программы (например, подлежащее факторизации число).
2. Программа проверяет заданное число на простоту с помощью теста, реализованного в лабораторной работе № 6. Если оно является простым, то процедура факторизации не выполняется.
3. Программа находит разложение заданного числа на произведение простых множителей.
4. Программа выдает список простых делителей заданного числа с указанием степени, с которой они входят в разложение числа, время и количество итераций основного цикла, потребовавшихся для разложения.

II. С помощью реализованного приложения выполнить следующие задания.
1. Протестировать правильность работы приложения на числах разной длины.
2. Сделать выводы о проделанной работе

Вариант	Метод
7	Метод ρ-Полларда

3. Метод решения задачи

В лабораторной работе реализуется программа разложение заданного числа на произведение простых множителей, используя метод ρ-Полларда. Данный алгоритм строит числовую последовательность, элементы которой образуют цикл, начиная с некоторого номера n, что может быть проиллюстрировано, расположением чисел в виде греческой буквы ρ.

Алгоритм метода:

1) Случайным образом выбирается x_1 из множества {0, 1, ... , n - 1}, $y = x_1$, k = 2, i = 1.
   Для следующих шагов запускается бесконечный цикл
2) i = i + 1. Вычисляется следующий элемент последовательности $x_i = f(x_{i-1})$ mod n, где $f(x) = x^2 + c$, где c – рандомное число из множества {0, 1, ... , n - 1}.

3) Вычисляем d = НОД($|y - x_i|$, n). Если 1 < d < n, то d является делителем n и работа алгоритма завершается и возвращается составное число, иначе выполняется переход на шаг 4.
4) Если $y = x_i$, завершаем работу алгоритма и возвращаем n.
5) Если i = k, то $y = x_i$, k = 2 · k

Если в 4 шаге $y = x_i$, то это означает, что числовая последовательность зациклилась. Мы предотвращаем данную зацикленность, завершая работу алгоритма с заведомо неверным результатом, чтобы повторить его снова.

Функция f(x) должна быть не слишком сложной для вычисления, но в то же время не должна быть линейным многочленом, а также не должна порождать взаимнооднозначное отображение.

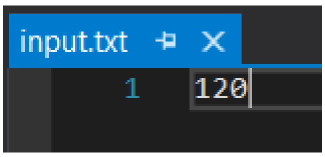
Метод имеет эвристическую оценку сложности $O(n^{\frac{1}{4}})$ арифметических операций и обычно используется для отделения небольших простых делителей факторизуемого числа n. Однако в этой оценке не учитываются накладные расходы по вычислению наибольшего общего делителя.

Метод ищет только один делитель n. Поэтому необходимо примерять метод несколько раз, пока не получится полное разложение числа на простые множители. Для этого используется рекурсивная функция. Сначала находим любой простой делитель числа n (пусть будет p) и сохраняем его. Далее выясняем, является ли число n / p составным, если да, то вызываем рекурсивную функцию для числа n / p, иначе сохраняем это число и завершаем поиск делителей.

1. Разработанное программное средство

Разработанное программное средство представляет собой консольное приложение на языке C#.

У пользователя есть возможность ввести во входном файле подлежащее факторизации число:



Найдем разложение заданного числа на произведение простых множителей, а также вычислим время работы всего алгоритма и число итераций основного цикла:

```
Полученное число из файла: 120

Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры факторизации

Факторизация ро-Полларда:
120 = 2 ^ 3 * 3 ^ 1 * 5 ^ 1

Время работы алгоритма: 0.0517
Число итераций основного алгоритма: 14
```

Программа выполнила факторизацию составного числа за 0.0517 секунд. Так как алгоритм рандомизированный, нельзя точно посчитать число итераций основного алгоритма, но в данном случае программа выполнила факторизацию за 14 итераций.

2. Исследования

Исследуем составные числа разной разрядности. Следует отметить, что все разложения были проверены вручную, корректность приведенных ниже примеров гарантирована.

Разрядность	Число	Результат
1	6	Консоль отладки Microsoft Visual Studio Полученное число из файла: 6 Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры факторизации Факторизация ро-Полларда: 6 = 2 ^ 1 * 3 ^ 1 Время работы алгоритма: 0.0327 Число итераций основного алгоритма: 2
2	95	Консоль отладки Microsoft Visual Studio Полученное число из файла: 95 Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры факторизации Факторизация ро-Полларда: 95 = 5 ^ 1 * 19 ^ 1 Время работы алгоритма: 0.0651 Число итераций основного алгоритма: 4
3	969	Консоль отладки Microsoft Visual Studio Полученное число из файла: 969 Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры факторизации Факторизация ро-Полларда: 969 = 3 ^ 1 * 17 ^ 1 * 19 ^ 1 Время работы алгоритма: 0.059 Число итераций основного алгоритма: 5

4	3567	Консоль отладки Microsoft Visual Studio Полученное число из файла: 3567 Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры факторизации Факторизация ро-Полларда: 3567 = 3 ^ 1 * 29 ^ 1 * 41 ^ 1 Время работы алгоритма: 0.06 Число итераций основного алгоритма: 15
5	65747	Консоль отладки Microsoft Visual Studio Полученное число из файла: 65747 Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры факторизации Факторизация ро-Полларда: 65747 = 11 ^ 1 * 43 ^ 1 * 139 ^ 1 Время работы алгоритма: 0.108 Число итераций основного алгоритма: 13
6	749737	Консоль отладки Microsoft Visual Studio Полученное число из файла: 749737 Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры факторизации Факторизация ро-Полларда: 749737 = 29 ^ 1 * 103 ^ 1 * 251 ^ 1 Время работы алгоритма: 0.1262 Число итераций основного алгоритма: 12
7	9463286	Консоль отладки Microsoft Visual Studio Полученное число из файла: 9463286 Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры факторизации Факторизация ро-Полларда: 9463286 = 2 ^ 1 * 7 ^ 1 * 191 ^ 1 * 3539 ^ 1 Время работы алгоритма: 0.1759 Число итераций основного алгоритма: 23
8	64536783	Консоль отладки Microsoft Visual Studio Полученное число из файла: 64536783 Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры факторизации Факторизация ро-Полларда: 64536783 = 3 ^ 1 * 593 ^ 1 * 36277 ^ 1 Время работы алгоритма: 0.1267 Число итераций основного алгоритма: 15
9	486759487	Выбрать Консоль отладки Microsoft Visual Studio Полученное число из файла: 486759487 Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры факторизации Факторизация ро-Полларда: 486759487 = 17 ^ 1 * 211 ^ 1 * 135701 ^ 1 Время работы алгоритма: 0.1916 Число итераций основного алгоритма: 11
10	1234567890	Консоль отладки Microsoft Visual Studio Полученное число из файла: 1234567890 Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры факторизации Факторизация ро-Полларда: 1234567890 = 2 ^ 1 * 3 ^ 2 * 5 ^ 1 * 3607 ^ 1 * 3803 ^ 1 Время работы алгоритма: 0.181 Число итераций основного алгоритма: 126

Сложность нахождения нетривиального делителя в методе ρ-Полларда зависит только от размера этого делителя, а не от размера числа n.

3. Код программы

Файл RhoPollard.h

```
#pragma once
#ifndef _RhoPollard_H
#define _RhoPollard_H

#include<iostream>
#include<ctime>
#include<algorithm>
#include<map>
#include <cassert>
#include <fstream>
#include <chrono>

#define ll long long

using namespace std;

// Модульное возведение в степень
ll ModularExponentiation(ll a, ll exponent, ll n);

// Модульное умножение
ll ModularMultiplication(ll a, ll b, ll mod);

// Алгоритм Евклида для вычисления НОД
ll Nod(ll a, ll b);

// Проверка числа на четность
bool Odd(int i);

// Рекурсивное вычисление символа Якоби
int GetJacobiSymbol(int a, int b);

// Тест Соловея-Штрассена
bool CheckingSoloveyStrassen(ll n, int iterations);

// Метод ρ-Полларда
ll RhoPollardFactorization(ll n, ll c, int& allIter);

// Факторизация числа, где составляющие вычисляются в методе ρ-Полларда
void Factorization(ll n, map<ll, int>& m, int& allIter);

// Получить число из файла
ll GetInfoFromFile(std::string fileName);

#endif
```

Файл RhoPollard.cpp

```cpp
#include "RhoPollard.h"

// Модульное умножение
ll ModularMultiplication(ll a, ll b, ll mod)
{
    ll x = 0;
    ll y = a % mod;

    while (b > 0)
    {
        if (b % 2 == 1)
        {
            x = (x + y) % mod;
        }
        y = (y * 2) % mod;
        b /= 2;
    }

    return x % mod;
}

// Модульное возведение в степень
ll ModularExponentiation(ll a, ll exponent, ll n)
{
    ll x = 1;
    ll y = a;

    while (exponent > 0)
    {

        if (exponent % 2 == 1)
        {
            x = (x * y) % n;
        }

        y = (y * y) % n;
        exponent = exponent / 2;
    }

    return x % n;
}

// Алгоритм Евклида для вычисления НОД
ll Nod(ll a, ll b)
{
    return b == 0 ? a : Nod(b, a % b);
}

// Проверка числа на четность
bool Odd(int i)
{

    if (i % 2 == 1)
    {
        return true;
    }

    return false;
}

// Рекурсивное вычисление символа Якоби
int GetJacobiSymbol(int a, int b)
{
    int g;

    if (b != 2)
        assert(Odd(b));

    if (a >= b)
```

```
        a %= b;

    if (a == 0)
        return 0;

    if (a == 1)
        return 1;

    if (a < 0)
        if ((b - 1) / 2 % 2 == 0)
            return GetJacobiSymbol(-a, b);
        else
            return -GetJacobiSymbol(-a, b);

    if (a % 2 == 0)
        if (((b * b - 1) / 8) % 2 == 0)
            return GetJacobiSymbol(a / 2, b);
        else
            return -GetJacobiSymbol(a / 2, b);

    g = Nod(a, b);
    assert(Odd(a));

    if (g == a)
        return 0;
    else
        if (g != 1)
            return GetJacobiSymbol(g, b) * GetJacobiSymbol(a / g, b);
        else
            if (((a - 1) * (b - 1) / 4) % 2 == 0)
                return GetJacobiSymbol(b, a);
            else
                return -GetJacobiSymbol(b, a);

}

// Тест Соловея-Штрассена
bool CheckingSoloveyStrassen(ll n, int iterations)
{

    if (n < 2 || (n != 2 && n % 2 == 0))
    {
        return false;
    }

    for (int i = 0; i < iterations; i++)
    {
        ll a = rand() % (n - 1) + 1;

        if (Nod(a, n) != 1)
        {
            return false;
        }

        ll jacobian = (n + GetJacobiSymbol(a, n)) % n;
        ll mod = ModularExponentiation(a, (n - 1) / 2, n);

        if (!jacobian || mod != jacobian)
        {
            return false;
        }

    }

    return true;
}

// Метод ρ-Полларда
ll RhoPollardFactorization(ll n, ll c, int& allIter)
```

```
{
    // Случайным образом выбирается x из множества {0, 1, ... , n - 1}
    ll x = rand() % n;
    ll y = x;
    ll k = 2;
    ll i = 1;

    while (1)
    {
        allIter++;

        i++;

        // Вычисляется следующий элемент последовательности x = f(x) = (x^2 + c) (mod n)
        x = (x * x + c) % n;

        // Вычисляем НОД
        ll d = Nod(abs(y - x), n);

        // Если 1 < d < n, то d является делителем n
        if (1 < d && d < n)
        {
            return d;
        }

        // Не допускаем зацикленности и выходим из алгоритма, чтобы повторить его снова
        if (y == x)
        {
            return n;
        }

        // Если i = k, то y = x, k = 2 * k
        if (i == k)
        {
            y = x;
            k *= 2;
        }

    }

}

// Факторизация числа, где составляющие вычисляются в методе ρ-Полларда
void Factorization(ll n, map<ll, int>& m, int& alliter)
{
    ll p = n;

    // Ищем простой множитель числа n
    while (p >= n || !CheckingSoloveyStrassen(p, 50))
    {
        p = RhoPollardFactorization(p, rand() % n, alliter);
    }

    // Добавляем простой множитель в результат
    m[p]++;

    // Выполняем поиск множителей числа n / p, если оно составное,
    // иначе записываем простое число в результат и завершаем факторизацию
    if (!CheckingSoloveyStrassen(n / p, 50))
    {
        Factorization(n / p, m, alliter);
    }
    else
    {
        m[n / p]++;
    }

}
```

```cpp
// Получить число из файла
ll GetInfoFromFile(std::string fileName)
{
    ifstream file;
    file.open(fileName);

    // Если файл не открыт, генерируем исключение
    if (!file.is_open())
    {
        throw std::invalid_argument("The file is not open");
    }

    ll input;
    file >> input;
    file.close();

    return input;
}
```

Файл Main.cpp

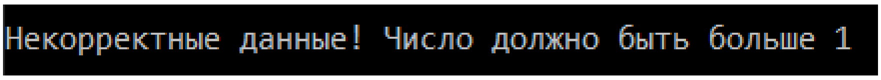
```cpp
#include "RhoPollard.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));

    map<ll, int> m;

    // Берем входные данные из файла
    ll n = GetInfoFromFile("input.txt");

    cout << "Полученное число из файла: " << n << endl << endl;

    // Проверяем полученные данные на корректность
    if(n < 2)
    {
        cout << "Некорректные данные! Число должно быть больше 1";
        return 0;
    }

    // Перед факторизацией проверяем, чтобы число было составным, иначе выход
    if (CheckingSoloveyStrassen(n, 50))
    {
        cout << "Тест Соловея-Штрассена: число простое, процедура факторизации не выполняется"
<< endl << endl;
        return 0;
    }
    else
    {
        cout << "Тест Соловея-Штрассена: число составное и подлежит выполнению процедуры
факторизации" << endl << endl;
    }

    cout << "Факторизация ро-Полларда:" << endl;

    int allIter = 0;

    // Замеряем время работы алгоритма факторизации
    auto start = std::chrono::steady_clock::now();
```

```
    Factorization(n, m, allIter);

    auto end = std::chrono::steady_clock::now();
    std::chrono::duration<double, std::milli> duration = end - start;

    // Выводим результат факторизации, а после - время работы всего алгоритма и число итераций
основного цикла метода ро-Полларда
    cout << n << " = ";

    for (auto it = m.begin(); it != m.end();)
    {
        cout << it->first << " ^ " << it->second;

        if ((++it) != m.end())
        {
            cout << " * ";
        }

    }

    cout << endl << endl << "Время работы алгоритма: " << duration.count() << endl;
    cout << "Число итераций основного алгоритма: " << allIter << endl << endl;

    return 0;
}
```

4. Тесты

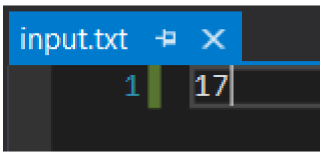
В исследованиях мы уже рассмотрели факторизацию чисел различных разрядностей.

1) Рассмотрим ситуации с ошибочным вводом. При вводе числа n < 2, мы получим следующее сообщение:

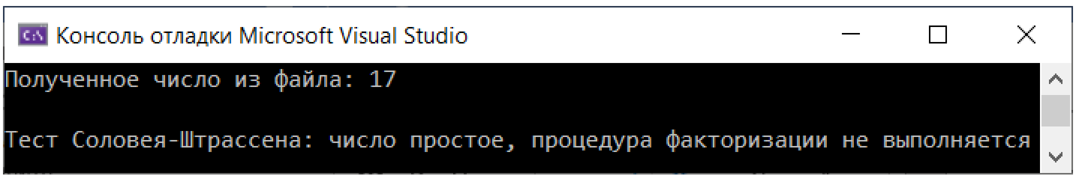


Число 1 также считается ошибочным вводом, потому что оно не является как простым числом, так и составным.

2) Попробуем ввести простое число:



В результате работы программы мы получаем следующее сообщение:



5. Вывод

В ходе выполнения лабораторной работы мы освоили простые алгоритмы факторизации составного числа.

Реализованный метод ρ-Полларда является вероятностным методом, который позволяет найти нетривиальный делитель q числа n за $O(q^{\frac{1}{2}}) \leq O(n^{\frac{1}{4}})$ итераций.

Как было сказано ранее, сложность нахождения нетривиального делителя в методе ρ-Полларда зависит только от размера этого делителя, а не от размера числа n. Поэтому данный метод применяется в тех случаях, когда иные методы факторизации, зависящие от размера n, являются низкоэффективными.