РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

Тема: Дискретное логарифмирование в конечном поле

дисциплина: Математические основы защиты информации и информационной безопасности

Студент: Койфман Кирилл Дмитриевич

Группа: НФИмд-01-25

Введение

Цель работы

Получение практических навыков реализации алгоритмов, решающих задачи дискретного логарифмирования.

Задачи

1. Реализовать алгоритм для задач дискретного логарифмирования.
2. Проверить работоспособность реализованного алгоритма.

Ход работы

1 задание

Для решения поставленной задачи реализуем описанный в тексте лабораторной работы алгоритм для задач дискретного логарифмирования на языке программирования C++ (Листинг-1):

```
#include <iostream>
#include <cmath>

namespace functions
{
    long long function_with_compression_behavior_1(long long x, long long
n)
    {
        return static_cast<long long>(std::pow(x, 2) + 5) % n;
    }

    //c is function arg
    long long function_with_compression_behavior_2(long long a, long long
p, long long c)
    {
        return (a * c) % p;
    }

    //c is function arg
    long long function_with_compression_behavior_3(long long b, long long
p, long long c)
    {
        return (b * c) % p;
    }
}

using compressing_function_t = long long(*)(long long, long long);
using compressing_function_t2 = long long(*)(long long, long long, long
long);

long long algorithmEuclid(long long a, long long b)
{
    if (b > a)
        std::swap(a, b);

    if (a <= 0 || b <= 0)
        return 1;

    long long d = 0;
    long long gamma_prev = a;
    long long gamma_curr = b;
    long long gamma_next = 0;

    while (true)
    {
        long long remainder = gamma_prev % gamma_curr;
        gamma_next = remainder;

        if (gamma_next == 0)
```

```
            {
                d = gamma_curr;
                break;
            }
            else
            {
                gamma_prev = gamma_curr;
                gamma_curr = gamma_next;
            }
        }

        return d;
    }

    int algorithmPollard(long long n, long long c = 1, compressing_function_t
    function = functions::function_with_compression_behavior_1)
    {
        long long a = c;
        long long b = c;
        long long d = 0;

        std::cout << "|a\tb\td|\n-------------------------------------\n";

        while (true)
        {
            a = function(a, n);
            b = function(b, n);
            b = function(b, n);

            d = algorithmEuclid(std::abs(a - b), n);

            std::cout << a << '\t' << b << '\t' << d << '\n';

            if (d > 1 && d < n)
            {
                return d;
            }
            else if (d == n)
            {
                return -1;
            }
            else if (d == 1)
            {
                continue;
            }
        }
    }

    //логарифм a по основанию b
    double logarithm(double a, double b)
    {
        return std::log(a) / std::log(b);
    }
```

```
long long algorithmPollardDiscreteLogarifmation(long long p, long long a,
long long gamma, long long b, long long u = 2, long long v = 2,
    compressing_function_t2 function1 =
functions::function_with_compression_behavior_2, compressing_function_t2
function2 = functions::function_with_compression_behavior_3)
{
    long long c = static_cast<long long>(std::pow(a, u) * std::pow(b, v));
    long long d = c;

    double log_c = 0;
    double log_d = 0;

    long long A1 = u, B1 = v;
    long long A2 = u, B2 = v;

    int counter = 10000;
    while (counter >= 0)
    {
        if (c < gamma)
        {
            c = function1(b, p, c) % p;
            d = function1(b, p, d) % p;
            d = function1(b, p, d) % p;
            ++A1;
            A2 += 2;
        }
        else if (c >= gamma)
        {
            c = function2(b, p, c) % p;
            d = function2(b, p, d) % p;
            d = function2(b, p, d) % p;
            ++B1;
            B2 += 2;
        }

        if (c % p == d)
            break;


        --counter;
    }
    return c;
}

int main()
{
    //prime value 'p'
    long long p = 107;
    long long a = 10;
    long long gamma = 53;
    long long b = 64;//(1 + p) / 2;
    long long u = 2, v = 2;

    algorithmPollardDiscreteLogarifmation(p, a, gamma, b, u, v);
```

```
    std::cout << std::endl;

    return 0;
}
```

Листинг-1(реализация алгоритма, реализующего p-метод Полларда)

Заключение

В ходе проделанной лабораторной работы мной были получены навыки по реализации алгоритмов, решающих задачи дискретного логарифмирования.