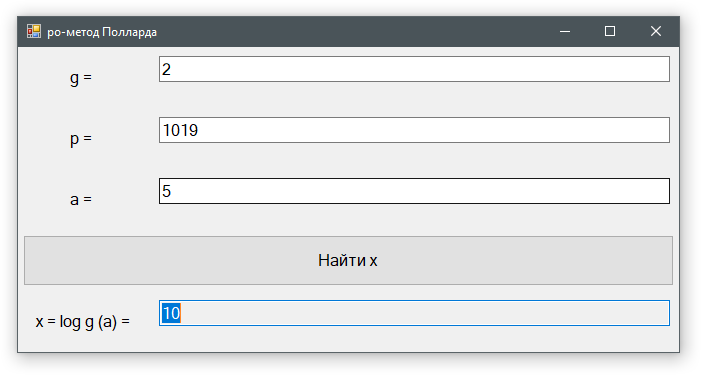
Лабораторная работа № 2. Метод Полларда дискретного логарифмирования.

Десятов Александр, 09-641



using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Numerics;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ро\_метод\_Полларда\_дискретного\_логарифмирования

{

public partial class FormPollard : Form

{

public FormPollard()

{

InitializeComponent();

}

private void btn\_calc\_Click(object sender, EventArgs e)

{

// Очистка полей

tB\_x.Text = "";

// Запрет на редактирование до окончания подсчета

tB\_g.ReadOnly = true;

tB\_p.ReadOnly = true;

tB\_a.ReadOnly = true;

// Запрет нажимать кнопку

btn\_calc.Enabled = false;

// Вызвать функцию подсчета X в отдельном потоке

bW\_calc.RunWorkerAsync();

}

// Функция подсчета X

private void bW\_calc\_DoWork(object sender, DoWorkEventArgs e)

{

// Считывание g

if (BigInteger.TryParse(tB\_g.Text, out BigInteger g))

{

// Проверка на положительность

if (g <= 0)

{

MessageBox.Show("g должно быть положительным!");

return;

}

}

else

{

MessageBox.Show("Не удалось считать g!");

return;

}

// Считывание n.

if (BigInteger.TryParse(tB\_p.Text, out BigInteger p))

{

// Проверка на положительность

if (p <= 0)

{

MessageBox.Show("n должно быть положительным!");

return;

}

}

else

{

MessageBox.Show("Не удалось считать n!");

return;

}

// Считывание a

if (BigInteger.TryParse(tB\_a.Text, out BigInteger a))

{

// Проверка на положительность

if (a < 0)

{

MessageBox.Show("a должно быть неотрицательным!");

return;

}

}

else

{

MessageBox.Show("Не удалось считать a!");

return;

}

// Использование Функции Эйлера дает возможность работать не только для простых p

BigInteger n = EulerPhi(p);

// Алгоритм ро-метод Полларда дискретного логарифмирования

BigInteger x1 = 1, x2 = 1;

BigInteger y1 = 0, y2 = 0;

BigInteger b1 = 0, b2 = 0;

if(a == g)

{

BigInteger x = 1;

tB\_x.Text = x.ToString();

return;

}

do

{

Floid(ref x1, ref y1, ref b1, g, p, a, n);

for (int i = 0; i < 2; i++)

{

Floid(ref x2, ref y2, ref b2, g, p, a, n);

}

} while (x1 != x2);

BigInteger r = (b1 - b2) % n;

while (r < 0)

r += n;

if (r == 0)

{

MessageBox.Show("Отказ");

return;

}

// Поиск d=НОД(r, n)

BigInteger d = BigInteger.GreatestCommonDivisor(r, n);

// Поиск обратного элемента к r/d по модулю n/d

Find\_NOD\_X\_Y\_Euclid(n/d, r/d, out \_, out \_, out BigInteger reverse);

while (reverse < 0)

reverse += n/d;

BigInteger residual\_y = y2-y1;

while (residual\_y < 0)

residual\_y += n;

BigInteger x0 = (residual\_y\*reverse);

for (BigInteger i = 0; i < d; i++)

{

BigInteger z = ((x0 + i\*n) / d) % n;

while (z < 0)

z += n;

if(a == BigInteger.ModPow(g, z, p))

{

tB\_x.Text = z.ToString();

return;

}

}

MessageBox.Show("Отказ");

}

private void Floid(ref BigInteger x, ref BigInteger y, ref BigInteger b, BigInteger g, BigInteger p, BigInteger a, BigInteger n)

{

switch ((int)(x % 3))

{

case 0:

x = (x \* x) % p;

y = (2 \* y) % n;

b = (2 \* b) % n;

break;

case 1: // x=1 придет сюда. Важно, чтобы x=1 не пришел в case 0

x = (g \* x) % p;

y = (y + 1) % n;

//b = b % n;

break;

case 2:

x = (a \* x) % p;

//y = y % n);

b = (b + 1) % n;

break;

default:

break;

}

}

// Событие завершения функции bW\_calc\_DoWork

private void bW\_calc\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)

{

// Разрешение на редактирование входных данных

tB\_g.ReadOnly = false;

tB\_p.ReadOnly = false;

tB\_a.ReadOnly = false;

// Разрешение нажимать кнопку

btn\_calc.Enabled = true;

}

// Событие закрытия формы

private void FormPollard\_FormClosing(object sender, FormClosingEventArgs e)

{

// Принудительно завершить поток, если он не завершен

if (bW\_calc.IsBusy)

{

bW\_calc.WorkerSupportsCancellation = true;

bW\_calc.CancelAsync();

}

}

private void Find\_NOD\_X\_Y\_Euclid(BigInteger A, BigInteger B, out BigInteger nod, out BigInteger x, out BigInteger y)

{

x = y = nod = 0;// Доллжно быть до любого return, так как out

if (A == 0 || B == 0)

return;

// Чтобы запомнить частные от делений и посчитать потом x и y

Stack<BigInteger> whole\_stack = new Stack<BigInteger>();

// DivRem() - возвращает частное и через out предоставляет остаток от деления

BigInteger whole = BigInteger.DivRem(A, B, out BigInteger ostatok);

while (ostatok != 0)

{

whole\_stack.Push(whole);

A = B;

B = ostatok;

whole = BigInteger.DivRem(A, B, out ostatok);

}

nod = B; //BigInteger.Abs(B);

// Считаем x и y

x = 0;

y = 1;

while (whole\_stack.Count > 0)

{

BigInteger temp = x;

x = y;

y = temp - y \* whole\_stack.Pop();

}

}

// Функция Эйлера

private BigInteger EulerPhi(BigInteger n)

{

BigInteger result = n;

for (BigInteger i = 2; i \* i <= n; ++i)

if (n % i == 0)

{

while (n % i == 0)

n /= i;

result -= result / i;

}

if (n > 1)

result -= result / n;

return result;

}

}

}