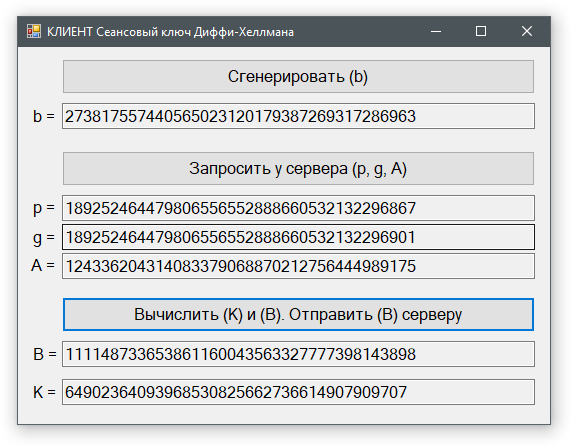
**­­Лабораторная работа №3. Сеансовый ключ. Алгоритм Диффи-Хеллмана.**

**Десятов Александр,**

**09-641**



using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Numerics;

using MySql.Data.MySqlClient;

using System.Threading;

namespace Сеансовый\_ключ\_Диффи\_Хеллмана.Клиент

{

public partial class FormGetKey : Form

{

public FormGetKey()

{

InitializeComponent();

}

string client\_login = "Саня"; // default

int numeric\_len\_b = 128; // default

MySqlConnection con;

Random rnd;

private void FormGetKey\_Shown(object sender, EventArgs e)

{

// Подключаемся к БД при открытии

con = new MySqlConnection("server=localhost;port=3306;database=usersandpasswords;user=root;password=Lhtlltcznjd32!;charset=utf8");

con.Open();

rnd = new Random((int)DateTime.Now.Ticks);

}

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

{

// Все метки изменить на (0, 0, 0) и сообщить серверу, что не нужно ждать.

set\_Marks\_All(false);

// Отключение от БД при закрытии формы

con.Close();

}

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

{

tB\_b.Text = "";

tB\_big\_B.Text = "";

tB\_K.Text = "";

// Получаем b

BigInteger b = SgenerirovatAny(numeric\_len\_b);

tB\_b.Text = b.ToString();

}

private BigInteger SgenerirovatAny(int numeric\_len)

{// Получаем рандомное бинарное число длины numeric\_len и переводим сразу его в десятичное

BigInteger chislo = 1; // Первая цифра 1. Иначе для перевода двоичного в десятичное здесь был бы 0

while (numeric\_len-- > 1) // сначала сравнивает, потом вычитает

{

chislo <<= 1; //умножаем на 2

chislo += rnd.Next(2);

}

return chislo;

}

private void btn\_get\_p\_g\_A\_Click(object sender, EventArgs e)

{

// Обнуляем

tB\_p.Text = "";

tB\_g.Text = "";

tB\_big\_A.Text = "";

tB\_big\_B.Text = "";

tB\_K.Text = "";

// Пемечаем в БД, что клиент ждет g, p, A

set\_Mark\_Wait\_p\_g\_A(true);

// Проверять каждую секунду не более 10 раз, записано p\_g\_A или нет

for (int waiting\_time = 10; !Check\_writen\_p\_g\_A();)

{

Thread.Sleep(1000);

// Если сервер отменил ожидание, выйти

if (!Check\_Wait\_p\_g\_A())

{

MessageBox.Show("Сервер отменил ваш запрос! Попробуйте еще раз!");

return;

}

waiting\_time--;

// Если произведено 10 попыток, выйти

if (waiting\_time == 0)

{

MessageBox.Show("Произведено много попыток, но сервер не ответил! Попробуйте еще раз!");

// Клаиент больше не ждет

set\_Mark\_Wait\_p\_g\_A(false);

return;

}

}

// Если дошли до сюда, значит p\_g\_A записались в БД

// Считать p, g, A

if (TryGet\_p\_g\_A(out BigInteger p, out BigInteger g, out BigInteger A))

{

// Вывести в textBox-ы

tB\_p.Text = p.ToString();

tB\_g.Text = g.ToString();

tB\_big\_A.Text = A.ToString();

}

else

{

MessageBox.Show("Не удалось считать данные! (p,g,A) из БД");

}

// Пемечаем в БД, что клиент не ждет g, p, A

set\_Mark\_Wait\_p\_g\_A(false);

}

// Возвращает, успешно ли считано p\_g\_A из БД

private bool TryGet\_p\_g\_A(out BigInteger p, out BigInteger g, out BigInteger A)

{

MySqlCommand command = new MySqlCommand("SELECT num\_p AS 'p', num\_g AS 'g', num\_A AS 'A' FROM users WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.ExecuteScalar();

MySqlDataReader dr = command.ExecuteReader();

dr.Read();

bool b1 = BigInteger.TryParse(dr["p"].ToString(), out p);

bool b2 = BigInteger.TryParse(dr["g"].ToString(), out g);

bool b3 = BigInteger.TryParse(dr["A"].ToString(), out A);

dr.Close();

return b1 && b2 && b3;

}

// Метки в БД (1 и 0), клиент или сервер что-то ждет

private void set\_Marks\_All(bool val)

{

MySqlCommand command = new MySqlCommand("UPDATE users SET wait\_p\_g\_A = @value, writen\_p\_g\_A = @value, writen\_B = @value, wait\_B = @value WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.Parameters.AddWithValue("@value", val);

command.ExecuteScalar();

}

private void set\_Mark\_Wait\_p\_g\_A(bool val)

{

MySqlCommand command = new MySqlCommand("UPDATE users SET wait\_p\_g\_A = @value WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.Parameters.AddWithValue("@value", val);

command.ExecuteScalar();

}

// Проверяются метки

private bool Check\_writen\_p\_g\_A()

{

MySqlCommand command = new MySqlCommand("SELECT writen\_p\_g\_A as answer FROM users WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.ExecuteScalar();

MySqlDataReader dr = command.ExecuteReader();

dr.Read();

bool answer = (bool)dr["answer"];

dr.Close();

return answer;

}

private bool Check\_Wait\_p\_g\_A()

{

MySqlCommand command = new MySqlCommand("SELECT wait\_p\_g\_A as answer FROM users WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.ExecuteScalar();

MySqlDataReader dr = command.ExecuteReader();

dr.Read();

bool answer = (bool)dr["answer"];

dr.Close();

return answer;

}

private void btn\_calc\_B\_K\_and\_send\_B\_Click(object sender, EventArgs e)

{

// Если для этого есть все необходимое

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

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

BigInteger.TryParse(tB\_big\_A.Text, out BigInteger A) &&

BigInteger.TryParse(tB\_b.Text, out BigInteger b))

{

BigInteger B = Vozvedenie\_v\_step\_po\_mod(g, b, p);

// Запись B в БД и пометить в БД, что B записано

Write\_B\_in\_DB(B);

BigInteger K = Vozvedenie\_v\_step\_po\_mod(A, b, p);

// Запись в textBox-ы

tB\_K.Text = K.ToString();

tB\_big\_B.Text = B.ToString();

}

else

{

MessageBox.Show("Не удалось считать данные! (p,g,A,b) из tB");

}

}

// Запись B в БД и пометить в БД, что B записано

private void Write\_B\_in\_DB(BigInteger B)

{

MySqlCommand command = new MySqlCommand("UPDATE users SET num\_B = @val\_B, writen\_B = @val\_writen\_B WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.Parameters.AddWithValue("@val\_B", B.ToString());

command.Parameters.AddWithValue("@val\_writen\_B", true);

command.ExecuteScalar();

}

private BigInteger Vozvedenie\_v\_step\_po\_mod(BigInteger A, BigInteger B, BigInteger N)

{

#region Обработка нештабных ситуаций

if (N == 0)

return 0;

if (B == 0)

return 1;

if (A == 0)

return 0;

if (B < 0)

{

Find\_NOD\_X\_Y\_Euclid(N, A, out BigInteger nod, out BigInteger x, out BigInteger y);

A = y; // Так возвели A в (-1) степень, теперь надо в степень abs(B)

B = BigInteger.Abs(B);

}

#endregion

Stack<bool> bin = new Stack<bool>();

while (B > 1)

{

bin.Push(!B.IsEven); // B.IsEven == true, если четное, иначе false

B >>= 1; //Делим на 2 побитовым сдвигом

}

BigInteger answer = A;

while (bin.Count > 0)

{

if (bin.Pop())

answer = (answer \* answer \* A) % N;

else

answer = (answer \* answer) % N;

}

return answer;

#region 2 метод из методчки

/\*BigInteger c = 1;

while(B != 0)

{

if(B%2 == 0)

{

B /= 2;

A = (A\*A)%N;

}

else

{

B--;

c=(c\*A)%N;

}

}

return c;\*/

#endregion

}

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

{

// Доллжно быть до любого return, так как out

x = y = nod = 0;

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;

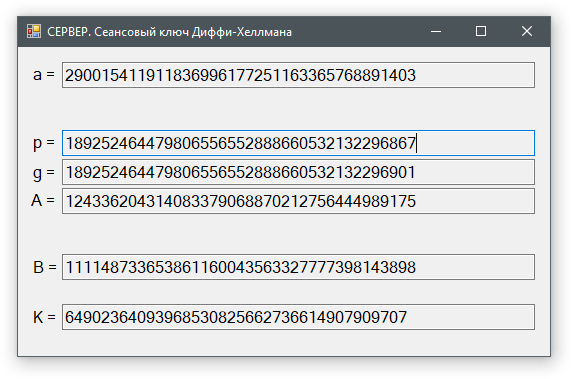
y = (temp - y \* whole\_stack.Pop());

}

}

}

}



using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Numerics;

using MySql.Data.MySqlClient;

namespace Сеансовый\_ключ\_Диффи\_Хеллмана.Сервер

{

public partial class FormGetKey : Form

{

public FormGetKey()

{

InitializeComponent();

}

string client\_login = "Саня"; // default

int numeric\_len\_a = 128; // default

int numeric\_len\_p = 128; // default

int numeric\_len\_g = 128; // default

MySqlConnection con;

Random rnd;

System.Timers.Timer timer;

#region События формы

private void FormGetKey\_Shown(object sender, EventArgs e)

{// Подключаемся к БД при открытии

con = new MySqlConnection("server=localhost;port=3306;database=usersandpasswords;user=root;password=Lhtlltcznjd32!;charset=utf8");

con.Open();

rnd = new Random((int)DateTime.Now.Ticks);

// Запуск таймера

timer = new System.Timers.Timer();

timer.Interval = 1000;

timer.Elapsed += Timer\_Tick;

// Запускаем таймер

timer.Start();

}

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

{

// Выключаем фоновый поток, если он включен

if (bW.IsBusy)

{

bW.WorkerSupportsCancellation = true;

bW.CancelAsync();

}

// Останавливаем таймер

timer.Stop();

timer.Dispose();

// Все метки изменить на (0, 0, 0) и сообщить серверу, что не нужно ждать.

set\_Marks\_All(false);

// Отключение от БД при закрытии формы

con.Close();

}

//Тик таймера

private void Timer\_Tick(object sender, System.Timers.ElapsedEventArgs e)

{

// Запуск фонового потока, если не запущен старый

if (!bW.IsBusy)

bW.RunWorkerAsync();

}

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

{

// Проверим метки

GetMarksFromDB(out bool wait\_p\_g\_A, out bool writen\_p\_g\_A, out bool writen\_B, out bool wait\_B);

// Если клиент ждет p\_g\_A, а сервер еще не записал их

if(wait\_p\_g\_A && !writen\_p\_g\_A)

{

tB\_a.Text = "";

tB\_p.Text = "";

tB\_g.Text = "";

tB\_big\_A.Text = "";

tB\_big\_B.Text = "";

tB\_K.Text = "";

// генерирация любого a, запись его в tB

BigInteger a = SgenerirovatAny(numeric\_len\_a);

tB\_a.Text = a.ToString();

// генерация g и p

BigInteger p = Sgenerirovat(numeric\_len\_p, 0);

tB\_p.Text = p.ToString();

// g отличное от p

BigInteger g = Sgenerirovat(numeric\_len\_g, p, true);

tB\_g.Text = g.ToString();

// подсчет A

BigInteger A = Vozvedenie\_v\_step\_po\_mod(g, a, p);

tB\_big\_A.Text = A.ToString();

// запись p, g, A в БД

Write\_p\_g\_A\_in\_DB(p, g, A);

// поставить метку, что p\_g\_A записано и что сервер ждет когда будет записано B

set\_Mark\_writen\_p\_g\_A(true);

set\_Mark\_wait\_B(true);

}

// Если сервер ждет B и B записано

//else if(wait\_B && writen\_B)

else if(writen\_B)

{

// Если удалось считать B из БД

if (TryGet\_B(out BigInteger B))

{

// Записать B в textBox

tB\_big\_B.Text = B.ToString();

// Все метки приравнять к нулю, потому что никто ничего не ждет

set\_Marks\_All(false);

// Получить g, p, a из textBox-ов

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

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

{

// Посчитать K

BigInteger K = Vozvedenie\_v\_step\_po\_mod(B, a, p);

// Вывести K в textBox

tB\_K.Text = K.ToString();

}

else

{

MessageBox.Show("Не удалось считать данные! (p,a) из tB");

}

}

else

{

MessageBox.Show("Не удалось считать данные! (B) из БД");

}

}

}

#endregion

#region Генерация чисел

// Генерация любого числа

private BigInteger SgenerirovatAny(int numeric\_len)

{// Получаем рандомное бинарное число длины numeric\_len и переводим сразу его в десятичное

BigInteger chislo = 1; // Первая цифра 1. Иначе для перевода двоичного в десятичное здесь был бы 0

while (numeric\_len-- > 1) // сначала сравнивает, потом вычитает

{

chislo <<= 1; //умножаем на 2

chislo += rnd.Next(2);

}

return chislo;

}

// Генерация вероятно простого числа и, если нужно, отличного от другого

private BigInteger Sgenerirovat(int numeric\_len, BigInteger first\_number, bool compare\_with = false)

{// Получаем рандомное бинарное число длины numeric\_len и сразу переделываем его в десятичное число

// И если compare\_with = true, то возвращается число отличное от first\_number

Random rnd = new Random((int)DateTime.Now.Ticks);

BigInteger chislo = 1; // Первая цифра 1. Иначе для перевода двоичного в десятичное здесь был бы 0

while (numeric\_len-- > 1) // сначала сравнивает, потом вычитает

{

chislo <<= 1; //умножаем на 2

chislo += rnd.Next(2);

}

if (chislo.IsEven)// Если четное

chislo += 1;

// Пока составное

// И пока не отличается от first\_number

while (Test\_Soloveya\_Shtrassena(chislo) == 0 || (compare\_with && chislo == first\_number))

chislo += 2;

return chislo; // Возвращаем вероятно простое число

}

// Получить рандомное BigInteger в диапазоне

private BigInteger GetRandomBigInteger(BigInteger min, BigInteger max)

{

if (min > max)

{

var buff = min;

min = max;

max = buff;

}

max -= min;

int deg = (int) BigInteger.Log(max, 2);

// Рандомная длина

deg = rnd.Next(deg+1);

BigInteger tmp = 1;

while (deg-- > 0)

{

tmp <<= 1; //умножаем на 2

tmp += rnd.Next(2);

}

// Если вдруг больше max получилось

BigInteger.DivRem(tmp, max, out BigInteger result);

return result + min;

}

#endregion

#region БД

// Запись p, g, A в БД

private void Write\_p\_g\_A\_in\_DB(BigInteger p, BigInteger g, BigInteger A)

{

MySqlCommand command = new MySqlCommand("UPDATE users SET num\_p = @val\_p, num\_g = @val\_g, num\_A = @val\_A WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.Parameters.AddWithValue("@val\_p", p.ToString());

command.Parameters.AddWithValue("@val\_g", g.ToString());

command.Parameters.AddWithValue("@val\_A", A.ToString());

command.ExecuteScalar();

}

// Получить все метки состояний

private void GetMarksFromDB(out bool wait\_p\_g\_A, out bool writen\_p\_g\_A, out bool writen\_B, out bool wait\_B)

{

MySqlCommand command = new MySqlCommand("SELECT wait\_p\_g\_A AS 'wait\_p\_g\_A', writen\_p\_g\_A AS 'writen\_p\_g\_A', writen\_B AS 'writen\_B', wait\_B AS 'wait\_B' FROM users WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.ExecuteScalar();

MySqlDataReader dr = command.ExecuteReader();

dr.Read();

wait\_p\_g\_A = (bool) dr["wait\_p\_g\_A"];

writen\_p\_g\_A = (bool) dr["writen\_p\_g\_A"];

writen\_B = (bool) dr["writen\_B"];

wait\_B = (bool) dr["wait\_B"];

dr.Close();

}

// Метки в БД (1 и 0), клиент или сервер что-то ждет

private void set\_Marks\_All(bool val)

{

MySqlCommand command = new MySqlCommand("UPDATE users SET wait\_p\_g\_A = @value, writen\_p\_g\_A = @value, writen\_B = @value, wait\_B = @value WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.Parameters.AddWithValue("@value", val);

command.ExecuteScalar();

}

// Метка, записаны ли p\_g\_A

private void set\_Mark\_writen\_p\_g\_A(bool val)

{

MySqlCommand command = new MySqlCommand("UPDATE users SET writen\_p\_g\_A = @val\_writen\_p\_g\_A WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.Parameters.AddWithValue("@val\_writen\_p\_g\_A", val);

command.ExecuteScalar();

}

// Метка, сервер ждет B

private void set\_Mark\_wait\_B(bool val)

{

MySqlCommand command = new MySqlCommand("UPDATE users SET wait\_B = @val\_wait\_B WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.Parameters.AddWithValue("@val\_wait\_B", val);

command.ExecuteScalar();

}

// Возвращает, успешно ли считано B из БД

private bool TryGet\_B(out BigInteger B)

{

MySqlCommand command = new MySqlCommand("SELECT num\_B AS 'B' FROM users WHERE login = @parLogin", con);

command.Parameters.AddWithValue("@parLogin", client\_login);

command.ExecuteScalar();

MySqlDataReader dr = command.ExecuteReader();

dr.Read();

bool b1 = BigInteger.TryParse(dr["B"].ToString(), out B);

dr.Close();

return b1;

}

#endregion

#region Тесты на простоту

private int Test\_Millera\_Rabina(BigInteger n)

{

if (n < 2)

return -1; // не простое и не составное

else

{

int s = 0;

BigInteger d = n - 1;

while (d.IsEven)

{

d /= 2;

s++;

}

BigInteger[] prime\_numbers\_50 = {

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\*/};

// Сколько раз проверять?

BigInteger r = 50; // log2(n)

// Если число n-2 меньше, чем r, то нужно проверять не r раз, а (n-2) раз

foreach (BigInteger a in prime\_numbers\_50)

//r = n - 2 < r ? n - 2 : r;

//for (BigInteger a = 2; a <= r + 1; a++) // a должно по простым ходить...

{

if (a > n - 1)

break;

BigInteger x\_current = Vozvedenie\_v\_step\_po\_mod(a, d, n);

if (x\_current == 1 || x\_current == n - 1)

continue; // a - свидетель простоты

else

{

bool svidetel\_neprostoti = true;

for (int i = 1; i < s; i++)

{

x\_current = Vozvedenie\_v\_step\_po\_mod(x\_current, 2, n);

if (x\_current == n - 1)

{

svidetel\_neprostoti = false;

break; // a - свидетель простоты

}

}

if (svidetel\_neprostoti)

return 0; // составное

}

}

return 1; // вероятно простое

}

}

private int Test\_Soloveya\_Shtrassena(BigInteger n)

{

if (n < 2)

return -1; // не простое и не составное

else if (n == 2)

return 1; // простое

else if (n.IsEven)

return 0; // составное

int k = 10;

while(k-- > 0)

{

// Случайным образом выбирается число a<n

BigInteger a = GetRandomBigInteger(2, n);

if (NOD\_Euclid(a, n) > 1)

{

return 0; // Составное

}

else

{

BigInteger res1 = Vozvedenie\_v\_step\_po\_mod(a, (n - 1) / 2, n);

BigInteger res2 = Yakobi\_symbol(a, n);

while (res2 < 0)

res2 += n;

if (res1 != res2)

return 0; // Составное

}

}

return 1; // Вероятно простое

}

#endregion

private int Yakobi\_symbol(BigInteger a, BigInteger b)

{

if(NOD\_Euclid(a, b) != 1)

return 0;

int r = 1;

if(a<0)

{

a = -a;

if (b % 4 == 3)

r = -r;

}

while (a != 0)

{

BigInteger t = 0;

while (a.IsEven)

{

t++;

a >>= 1; //деление на 2

}

if (!t.IsEven)

{

BigInteger.DivRem(b, 8, out BigInteger ostatok\_b\_na\_8);

if (ostatok\_b\_na\_8 == 3 || ostatok\_b\_na\_8 == 5)

r = -r;

}

BigInteger.DivRem(a, 4, out BigInteger ostatok\_a\_na\_4);

BigInteger.DivRem(b, 4, out BigInteger ostatok\_b\_na\_4);

if (ostatok\_a\_na\_4 == ostatok\_b\_na\_4 && ostatok\_a\_na\_4 == 3)

r = -r;

BigInteger c = a;

BigInteger.DivRem(b, c, out a);

b = c;

}

return r;

}

private BigInteger Vozvedenie\_v\_step\_po\_mod(BigInteger A, BigInteger B, BigInteger N)

{

#region Обработка нештатных ситуаций

if (N == 0)

return 0;

if (B == 0)

return 1;

if (A == 0)

return 0;

if (B < 0)

{

Find\_NOD\_X\_Y\_Euclid(N, A, out BigInteger nod, out BigInteger x, out BigInteger y);

A = y; // Так возвели A в (-1) степень, теперь надо в степень abs(B)

B = BigInteger.Abs(B);

}

#endregion

Stack<bool> bin = new Stack<bool>();

while (B > 1)

{

bin.Push(!B.IsEven); // B.IsEven == true, если четное, иначе false

B >>= 1; //Делим на 2 побитовым сдвигом

}

BigInteger answer = A;

while (bin.Count > 0)

{

if (bin.Pop())

answer = (answer \* answer \* A) % N;

else

answer = (answer \* answer) % N;

}

while (answer < 0)

answer += N;

return answer;

#region 2 метод из методчки

/\*BigInteger c = 1;

while(B != 0)

{

if(B%2 == 0)

{

B /= 2;

A = (A\*A)%N;

}

else

{

B--;

c=(c\*A)%N;

}

}

return c;\*/

#endregion

}

#region НОД

// НОД, x, y

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

{

// Доллжно быть до любого return, так как out

x = y = nod = 0;

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());

}

}

// НОД

private BigInteger NOD\_Euclid(BigInteger A, BigInteger B)

{

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

return 0;

// 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);

}

return BigInteger.Abs(B);

}

#endregion

}

}