МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №6 по дисциплине

“Коды, исправляющие ошибки”

Выполнил студент группы ПМИ-3301-02-00     / А.Л. Вохмянин /

Проверил преподаватель кафедры ПМИ     / И.А. Пушкарёв /

Киров 2019

**Цель:** реализовать программу, реализующую деление многочленов с остатком и представление в линейном виде в поле Zp.

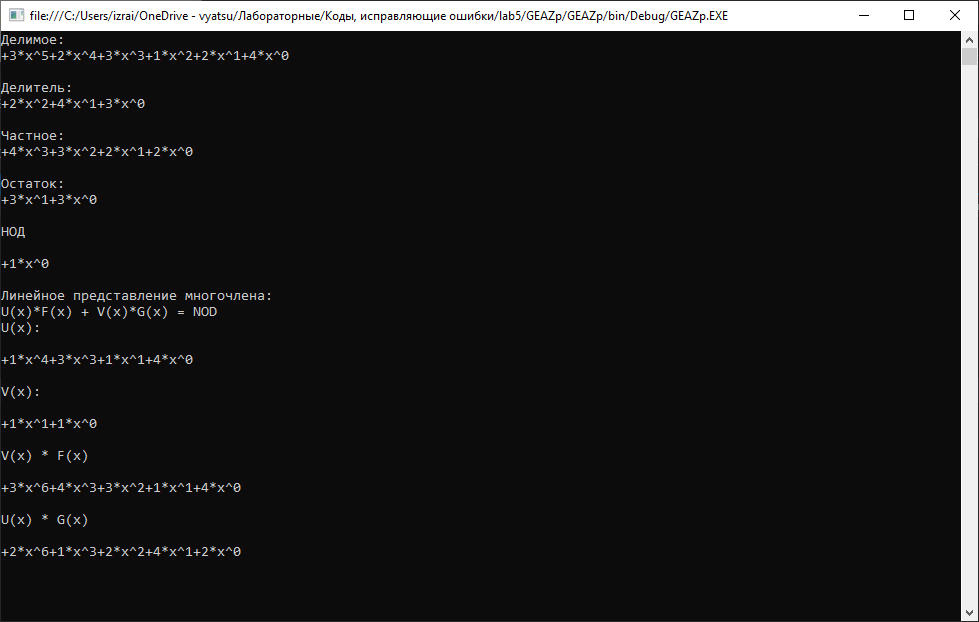
Пусть a и b — целые числа, не равные одновременно нулю, и последовательность чисел

определена тем, что каждое — это остаток от деления предпредыдущего числа на предыдущее, а предпоследнее делится на последнее нацело.

Тогда НОД(a, b), наибольший общий делитель a и b, равен , последнему ненулевому члену этой последовательности

Алгоритм Евклида и расширенный алгоритм Евклида естественным образом обобщается на [кольцо многочленов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D1%8C%D1%86%D0%BE_%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD%D0%BE%D0%B2) k[x] от одной переменной над произвольным [полем](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_(%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0)) k, поскольку для таких многочленов определена операция деления с остатком. При выполнении алгоритма Евклида для многочленов аналогично алгоритму Евклида для целых чисел получается последовательность полиномиальных остатков

**Результаты выполнения программы:**



**Вывод:** былиизучены и получены практические навыки использования алгоритма Евклида для многочленов в поле Zp.

**Листинг:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GEA

{

class Program

{

const double eps = 0.0001;

static List<int> NOD;

const int p = 5;

public static class MyMath

{

public static int Multiply(int a, int b)

{

int result;

result = a \* b;

result = result % p;

return result;

}

public static int Add(int a, int b)

{

int result;

result = a + b;

result = result % p;

return result;

}

public static int Minus(int a, int b)

{

int result;

if (a - b >= 0 && (a - b) <= 4)

result = a - b;

else

{

a += p;

if (a - b >= 0 && (a - b) <= 4)

return a - b;

b = Reversible(b);

//b = a \* b;

b = b % p;

result = a + b;

result = result % p;

}

return result;

}

static int Reversible(int b)

{

// Вычисляется число d, мультипликативно обратное к числу e по модулю φ(n), то есть число, удовлетворяющее сравнению:

// d ⋅ l ≡ 1 (mod φ(n))

int d;

int k = 1;

while (true)

{

k = k + p;

if (k % b == 0)

{

d = (k / b);

return d;

}

}

}

public static int Div(int a, int b)

{

int result = b;

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

{

result \*= i;

if (result % p == a)

return i;

else

result = b;

}

//a \*= p;

//result = a / b;

//result = result % p;

return result;

}

public static List<int> Minus(List<int> polynom1, List<int> polynom2)

{

int itemsCount = Math.Max(polynom1.Count, polynom2.Count);

var result = new List<int>(new int[itemsCount]);

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

{

int a = 0;

int b = 0;

if (i < polynom1.Count)

{

a = polynom1[i];

}

if (i < polynom2.Count)

{

b = polynom2[i];

}

result[i] = Minus(a, b);

}

return result;

}

public static List<int> Multiplication(List<int> polynom1, List<int> polynom2)

{

List<int> result = new List<int>(new int[polynom1.Count + polynom2.Count - 1]);

for (int i = 0; i < polynom1.Count; i++)

for (int j = 0; j < polynom2.Count; j++)

{

if (result[i + j] != 0)

{

result[i + j] = Add(result[i + j], Multiply(polynom1[i], polynom2[j]));

}

else

result[i + j] = Multiply(polynom1[i], polynom2[j]);

}

return result;

}

public static List<int> Add(List<int> polynom1, List<int> polynom2)

{

int count = Math.Max(polynom1.Count, polynom2.Count);

List<int> result = new List<int>(new int[count]);

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

{

int a = 0;

int b = 0;

if (i < polynom1.Count)

{

a = polynom1[i];

}

if (i < polynom2.Count)

{

b = polynom2[i];

}

result[i] = Add(a, b);

}

return result;

}

public static void Deconv(List<int> dividend, List<int> divisor, out List<int> quotient, out List<int> remainder)

{

if (dividend.Count == 0)

{

remainder = new List<int>(dividend);

quotient = new List<int>(new int[remainder.Count - divisor.Count + 1]);

return;

}

if (divisor.Count == 0)

{

remainder = new List<int>(dividend);

quotient = new List<int>(new int[remainder.Count - divisor.Count + 1]);

return;

}

while (dividend.Last() == 0)

{

dividend.RemoveAt(dividend.Count - 1);

if (dividend.Count == 0)

{

remainder = new List<int>(dividend);

quotient = new List<int>(new int[remainder.Count - divisor.Count + 1]);

return;

}

}

while (divisor.Last() == 0)

{

divisor.RemoveAt(divisor.Count - 1);

if (divisor.Count == 0)

{

remainder = new List<int>(dividend);

quotient = new List<int>(new int[remainder.Count - divisor.Count + 1]);

return;

}

}

remainder = new List<int>(dividend);

int temp;

if ((remainder.Count - divisor.Count + 1) < 0)

temp = 0;

else

temp = remainder.Count - divisor.Count + 1;

quotient = new List<int>(new int[temp]);

for (int i = 0; i < quotient.Count; i++)

{

int coeff = Div(remainder[remainder.Count - i - 1], divisor.Last());

quotient[quotient.Count - i - 1] = coeff;

for (int j = 0; j < divisor.Count; j++)

{

remainder[remainder.Count - i - j - 1] = Minus(remainder[remainder.Count - i - j - 1], Multiply(coeff,divisor[divisor.Count - j - 1]));

}

}

}

public static List<int> gcd(List<int> a, List<int> b, out List<int> x, out List<int> y)

{

if (a.Count == 0)

{

x = new List<int> { 0 };

y = new List<int> { 1 };

return b;

}

List<int> x1, y1;

List<int> quotient;

List<int> remainder;

MyMath.Deconv(b, a, out quotient, out remainder);

List<int> d = gcd(remainder, a, out x1, out y1);

x = Minus(y1, (Multiplication(quotient, x1)));

y = new List<int>(x1);

return d;

}

}

public static void Print(List<int> pol)

{

Console.WriteLine();

for (int i = 0; i < pol.Count; i++)

{

if (pol[pol.Count - i - 1] != 0)

{

Console.Write("{0}{1}\*x^{2}", pol[pol.Count - i - 1] >= 0 ? "+" : "", pol[pol.Count - i - 1], pol.Count - i - 1);

}

}

Console.WriteLine("\n");

}

static void Main(string[] args)

{

List<int> dividend = new List<int> {4, 2, 1, 3, 2, 3};

List<int> divisor = new List<int> {3, 4, 2};

List<int> quotient;

List<int> remainder;

#region Вывод dividend = divisor \* quotient + remainder

MyMath.Deconv(dividend, divisor, out quotient, out remainder);

Console.Write("Делимое:");

Print(dividend);

Console.Write("Делитель:");

Print(divisor);

Console.Write("Частное:");

Print(quotient);

Console.Write("Остаток:");

Print(remainder);

#endregion

// List<int> tempDividend = new List<int> (dividend);

// List<int> tempDivisor = new List<int> (divisor);

// List<int> temp = new List<int>(remainder);

// bool flag = false;

//while (true)

//{

// while (remainder.Last() == 0 || Math.Abs(remainder.Last()) < eps)

// {

// remainder.RemoveAt(remainder.Count - 1);

// if (remainder.Count == 0)

// {

// remainder = new List<int>(temp);

// flag = true;

// break;

// }

// }

// if (flag == true)

// break;

// tempDividend = new List<int>(tempDivisor);

// tempDivisor = new List<int>(remainder);

// temp = new List<int>(remainder);

// MyMath.Deconv(dividend, divisor, out quotient, out remainder);

//}

List<int> V = new List<int>();

List<int> U = new List<int>();

NOD = MyMath.gcd(dividend, divisor, out V, out U);

Console.WriteLine("НОД");

Print(NOD);

Console.WriteLine("Линейное представление многочлена: \nU(x)\*F(x) + V(x)\*G(x) = NOD");

//Console.WriteLine("F(x):");

//Print(dividend);

//Console.WriteLine("G(x):");

//Print(divisor);

Console.WriteLine("U(x):");

Print(U);

Console.WriteLine("V(x):");

Print(V);

Console.WriteLine("V(x) \* F(x)");

Print(MyMath.Multiplication(V, dividend));

Console.WriteLine("U(x) \* G(x)");

Print(MyMath.Multiplication(U, divisor));

Console.ReadKey();

}

}

}