Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Дисциплина: Криптографические протоколы**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Д.Е.Гиренко

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Крамаренко

**Цель работы:** реализовать программный продукт для решения задач над некоторым полем Галуа для ускорения решения задач из лабораторной работы.

**Ход работы:**

1) Воспользуемся классом Polynom и функциями проверки многочлена на неприводимость и примитивность из прошлой лабораторной работы:

Polynom operator+(Polynom& b) {

ll newDegree = max(this->degree(), b.degree());

vector<bool> newPoly(newDegree + 1, false);

for (ll i = newDegree; i >= 0; i--) {

if (i > this->degree()) {

newPoly[i] = b.body[i];

}

else if (i > b.degree()) {

newPoly[i] = this->body[i];

}

else {

newPoly[i] = this->body[i] ^ b.body[i];

}

}

ll zeroPadLeft = 0;

for (ll i = newDegree; i >= 0; i--) {

if (newPoly[i] == 0) {

zeroPadLeft++;

}

else {

break;

}

}

newPoly.resize(newPoly.size() - zeroPadLeft);

return Polynom(newPoly);

}

bool polynomIsPrimitiveInGP(Polynom& poly, ll gpDegree) {

if (polynomIsIrreducible(poly)) {

ll maxDegree = pow(2, gpDegree) - 1;

cout << endl;

cout << "primitive check: " << endl;

for (ll i = 1; i <= maxDegree; i++) {

ll polyNumber = pow(2, i);

Polynom checkPoly = Polynom::byNumber(polyNumber + 1);

cout << "check: "; checkPoly.print();

Polynom remainder = checkPoly % poly;

cout << "remainder is: "; remainder.print();

if (remainder.degree() == -1) {

return false;

}

}

return true;

}

return false;

}

2) Создадим класс GP2 и реализуем функцию получения его элементов по произвольному образующему полиному:

vector<Polynom> getAllRemainsInGF() {

int degree = m();

int remainsCount = 1 << degree;

vector<Polynom> result;

result.push\_back(Polynom::byDegree(0));

Polynom x = Polynom::byDegree(1);

for (int i = 1; i < remainsCount - 1; i++) {

Polynom tmp = (result[i - 1] \* x);

tmp = tmp % mainPoly;

result.push\_back(tmp);

}

return result;

}

3) Реализуем получение всех цикломатических классов и минимальных полиномов, соответствующих им. Все промежуточные вычисления записываются в класс для дальнейшей работы:

vector<set<int>> getAllcClasses() {

vector<set<int>> result;

vector<bool> visited(allRemains.size(), false);

for (int i = 0; i < allRemains.size(); i++) {

set<int> c;

for (int j = 0; j < m(); j++) {

int a = (i \* pow(p(), j));

a %= allRemains.size();

if (visited[a] && c.find(a) == c.end()) {

goto exit;

}

c.insert(a);

visited[a] = true;

}

result.push\_back(c);

exit:

;

}

return result;

}

vector<Polynom> getMinPolynoms() {

vector<Polynom> result;

for (int i = 0; i < cClasses.size(); i++) {

vector<int> c;

for (auto j = cClasses[i].begin(); j != cClasses[i].end(); j++) {

c.push\_back(\*j);

}

Polynom minPoly = Polynom();

int sumCount = 1 << c.size();

for (int j = 0; j < sumCount; j++) {

int xCount = 0;

int aCount = 0;

int tmp = j;

for (int k = 0; k < c.size(); k++) {

if (tmp % 2 == 1) {

aCount += c[k];

}

else {

xCount += 1;

}

tmp /= 2;

}

aCount %= allRemains.size();

Polynom xPoly = Polynom::byDegree(xCount);

xPoly = allRemains[aCount] \* xPoly;

minPoly = minPoly + xPoly;

}

result.push\_back(minPoly);

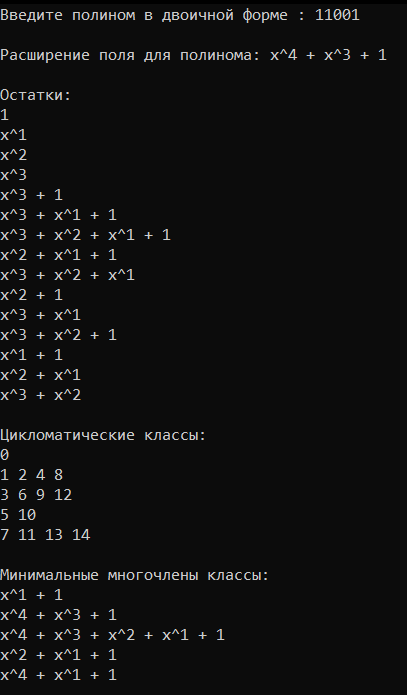
}

return result;

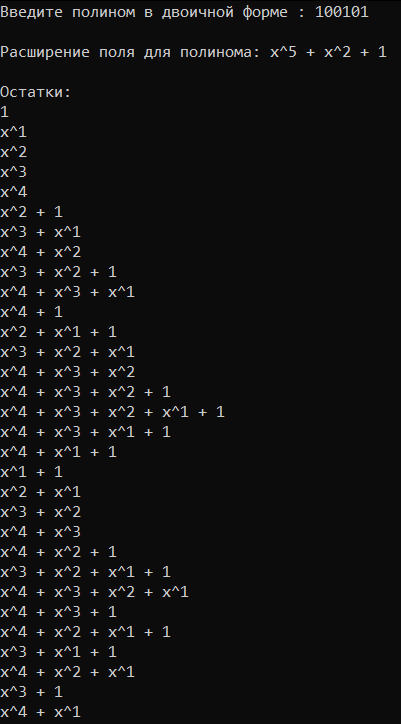
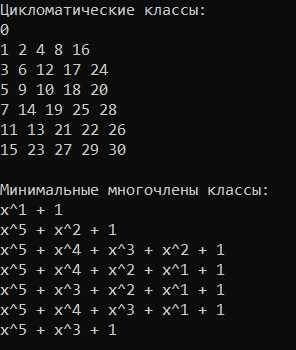
}

4) Применим полученный программный продукт для решения задач из лабораторной работы:

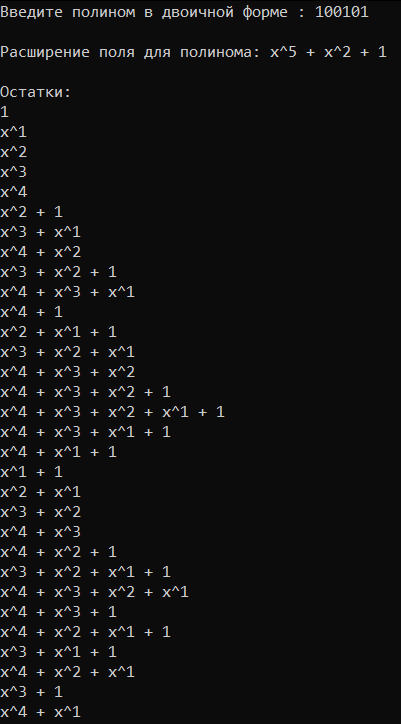
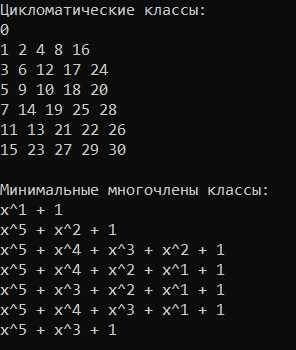
1. Выделить циклотомические классы и найти соответствующие им минимальные многочлены для поля GF(16) для образующего многочлена 11001:



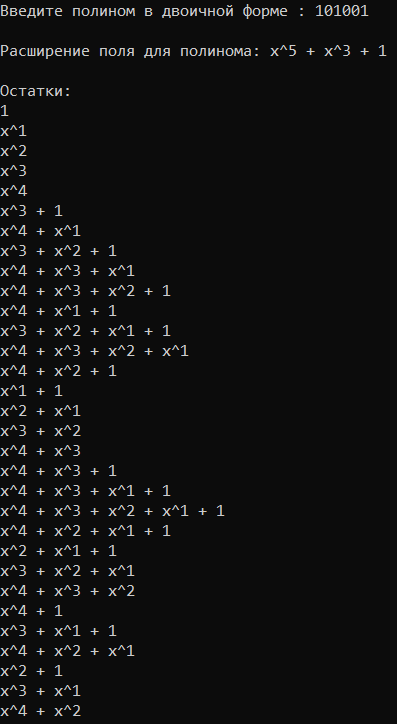
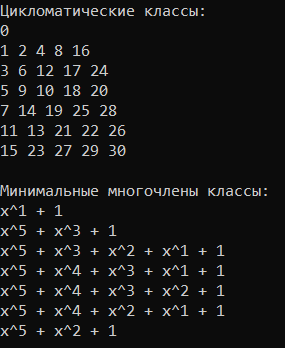
2. Для многочлена х над полем с образующим многочленом найти циклотомический класс и минимальный многочлен:

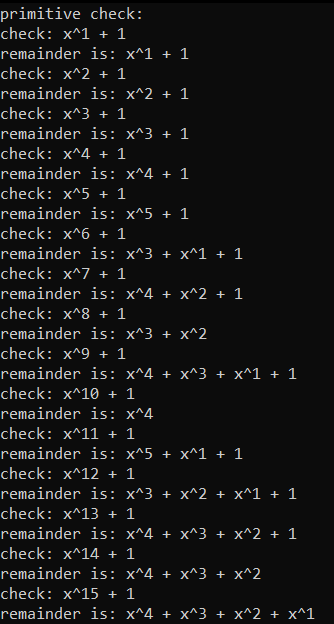
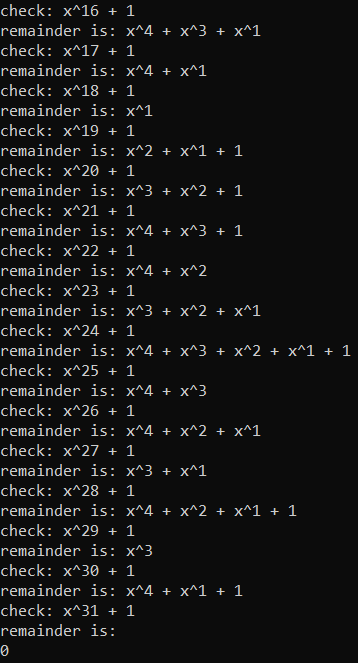
3. Для многочлена над полем с образующим многочленом найти циклотомический класс и минимальный многочлен:

4. Построить циклотомические классы и минимальные многочлены в поле Галуа :

 ь

5. Проверить на примитивность неприводимый многочлен 100101:

В результате имеем программный продукт с удобным интерфейсом, способный вычислять минимальные полиномы произвольного расширения поля Галуа GF2 и решённую 4-ю лабораторную работу.

**Листинг программы**

#include <iostream>

#include <chrono>

#include <random>

#include <vector>

#include <string>

#include <set>

#include <map>

#include <algorithm>

#include "windows.h"

#include <numeric>

using namespace std;

typedef long long ll;

class Polynom {

public:

vector<bool> body;

ll degree() {

return body.size() - 1;

}

Polynom() {

this->body = {};

}

Polynom(vector<bool> body) {

this->body = body;

}

static Polynom byDegree(ll degree)

{

vector<bool> newPoly(degree + 1, 0);

newPoly[degree] = true;

return Polynom(newPoly);

}

static Polynom byNumber(ll number)

{

vector<bool> newPoly;

while (number) {

newPoly.push\_back(number % 2);

number /= 2;

}

return Polynom(newPoly);

}

static Polynom inputPoly() {

string s;

cout << "Введите полином в двоичной форме : ";

getline(cin, s);

cout << endl;

vector<bool> poly(s.size());

for (int i = 0; i < s.size(); i++) {

poly[s.size() - 1 - i] = s[i] == '1';

}

return Polynom(poly);

}

void print() {

bool lastPrinted = false;

for (ll i = this->degree(); i >= 0; i--) {

if (this->body[i]) {

if (lastPrinted) {

cout << " + ";

}

if (i == 0) {

cout << "1";

}

else {

cout << "x^" << i;

}

lastPrinted = true;

}

}

cout << endl;

}

Polynom operator+(Polynom& b) {

ll newDegree = max(this->degree(), b.degree());

vector<bool> newPoly(newDegree + 1, false);

for (ll i = newDegree; i >= 0; i--) {

if (i > this->degree()) {

newPoly[i] = b.body[i];

}

else if (i > b.degree()) {

newPoly[i] = this->body[i];

}

else {

newPoly[i] = this->body[i] ^ b.body[i];

}

}

ll zeroPadLeft = 0;

for (ll i = newDegree; i >= 0; i--) {

if (newPoly[i] == 0) {

zeroPadLeft++;

}

else {

break;

}

}

newPoly.resize(newPoly.size() - zeroPadLeft);

return Polynom(newPoly);

}

Polynom operator\*(Polynom& b) {

vector<bool> newPoly(this->degree() + b.degree() + 1, 0);

for (ll i = this->degree(); i >= 0; i--) {

if (!this->body[i]) {

continue;

}

for (ll j = b.degree(); j >= 0; j--) {

newPoly[i + j] = newPoly[i + j] ^ (this->body[i] \* b.body[j]);

}

}

return Polynom(newPoly);

}

bool operator>=(Polynom& b) {

if (this->degree() == b.degree()) {

for (ll i = b.degree(); i >= 0; i--) {

if (this->body[i] != b.body[i]) {

return this->body[i] > b.body[i];

}

}

return true;

}

else

{

return this->degree() >= b.degree();

}

}

Polynom operator%(Polynom& b) {

Polynom dividedPoly = Polynom(this->body);

while (dividedPoly.degree() > b.degree()) {

ll divisionPartDegree = dividedPoly.degree() - b.degree();

Polynom divisionPartPoly = Polynom::byDegree(divisionPartDegree);

Polynom subPartPoly = b \* divisionPartPoly;

dividedPoly = dividedPoly + subPartPoly;

}

if (dividedPoly.degree() == b.degree()) {

dividedPoly = dividedPoly + b;

}

return Polynom(dividedPoly);

}

};

bool polynomIsIrreducible(Polynom& poly) {

ll maxSearchDegree = poly.degree() / 2;

ll maxSearchNumber = pow(2, maxSearchDegree + 1) - 1;

for (ll i = 2; i <= maxSearchNumber; i++) {

Polynom checkPoly = Polynom::byNumber(i);

cout << "check: "; checkPoly.print();

Polynom remainder = poly % checkPoly;

cout << "remainder is: "; remainder.print();

if (remainder.degree() == -1) {

return false;

}

}

return true;

}

bool polynomIsPrimitiveInGP(Polynom& poly, ll gpDegree) {

if (polynomIsIrreducible(poly)) {

ll maxDegree = pow(2, gpDegree) - 1;

cout << endl;

cout << "primitive check: " << endl;

for (ll i = 1; i <= maxDegree; i++) {

ll polyNumber = pow(2, i);

Polynom checkPoly = Polynom::byNumber(polyNumber + 1);

cout << "check: "; checkPoly.print();

Polynom remainder = checkPoly % poly;

cout << "remainder is: "; remainder.print();

if (remainder.degree() == -1) {

return false;

}

}

return true;

}

return false;

}

class GP2 {

public:

Polynom mainPoly;

vector<Polynom> allRemains;

vector<set<int>> cClasses;

vector<Polynom> minPolynoms;

int p() {

return 2;

}

int m() {

return mainPoly.degree();

}

vector<Polynom> getAllRemainsInGF() {

int degree = m();

int remainsCount = 1 << degree;

vector<Polynom> result;

result.push\_back(Polynom::byDegree(0));

Polynom x = Polynom::byDegree(1);

for (int i = 1; i < remainsCount - 1; i++) {

Polynom tmp = (result[i - 1] \* x);

tmp = tmp % mainPoly;

result.push\_back(tmp);

}

return result;

}

vector<set<int>> getAllcClasses() {

vector<set<int>> result;

vector<bool> visited(allRemains.size(), false);

for (int i = 0; i < allRemains.size(); i++) {

set<int> c;

for (int j = 0; j < m(); j++) {

int a = (i \* pow(p(), j));

a %= allRemains.size();

if (visited[a] && c.find(a) == c.end()) {

goto exit;

}

c.insert(a);

visited[a] = true;

}

result.push\_back(c);

exit:

;

}

return result;

}

vector<Polynom> getMinPolynoms() {

vector<Polynom> result;

for (int i = 0; i < cClasses.size(); i++) {

vector<int> c;

for (auto j = cClasses[i].begin(); j != cClasses[i].end(); j++) {

c.push\_back(\*j);

}

Polynom minPoly = Polynom();

int sumCount = 1 << c.size();

for (int j = 0; j < sumCount; j++) {

int xCount = 0;

int aCount = 0;

int tmp = j;

for (int k = 0; k < c.size(); k++) {

if (tmp % 2 == 1) {

aCount += c[k];

}

else {

xCount += 1;

}

tmp /= 2;

}

aCount %= allRemains.size();

Polynom xPoly = Polynom::byDegree(xCount);

xPoly = allRemains[aCount] \* xPoly;

minPoly = minPoly + xPoly;

}

result.push\_back(minPoly);

}

return result;

}

GP2(Polynom mainPoly) {

this->mainPoly = mainPoly;

this->allRemains = getAllRemainsInGF();

this->cClasses = getAllcClasses();

this->minPolynoms = getMinPolynoms();

}

void print() {

cout << "Расширение поля для полинома: ";

mainPoly.print();

cout << endl;

cout << "Остатки: " << endl;

for (int i = 0; i < allRemains.size(); i++) {

allRemains[i].print();

}

cout << endl;

cout << "Цикломатические классы: " << endl;

for (int i = 0; i < cClasses.size(); i++) {

for (auto j = cClasses[i].begin(); j != cClasses[i].end(); j++) {

cout << \*j << " ";

}

cout << endl;

}

cout << endl;

cout << "Минимальные многочлены классы: " << endl;

for (int i = 0; i < minPolynoms.size(); i++) {

minPolynoms[i].print();

}

}

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Polynom polyTrue = Polynom::inputPoly();

GP2 gf = GP2(polyTrue);

gf.print();

}