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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К.В.Стасюк

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:

def polynom\_is\_primitive\_in\_gp(poly, gp\_degree):  
 #полином не приводим  
 if polynom\_is\_irreducible(poly):  
 #макс степень=степень поля  
 max\_degree = 2\*\*gp\_degree  
 for i in range(1, max\_degree):  
 poly\_number = 2\*\*i  
 check\_poly = Polynom.by\_number(poly\_number + 1)  
 print("Делимый полином:", check\_poly.print())  
 remainder = check\_poly.division(poly)  
 print("Остаток:", remainder.print())  
 #остаток=0  
 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 :

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;

}

При помощи разработанных программ заишем решение задач:

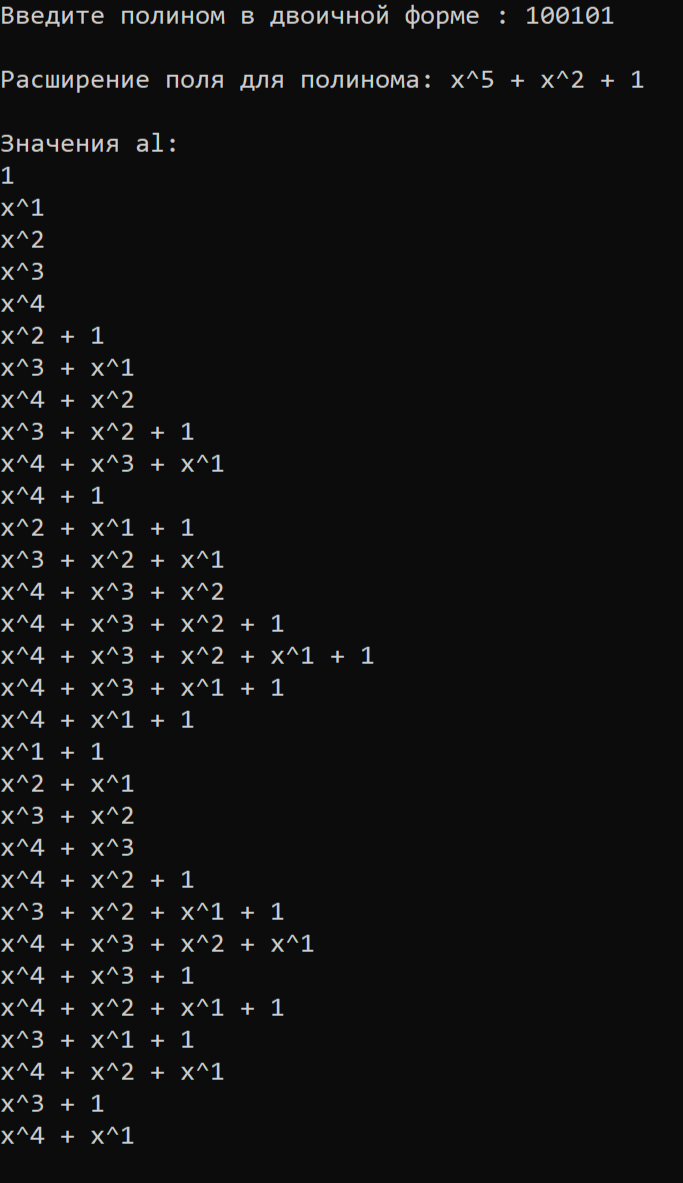
1. Выделить циклотомические классы и найти соответствующие им минимальные многочлены для поля GF(16) для образующего многочлена 11001:

Изображение выглядит как текст, снимок экрана, дизайн

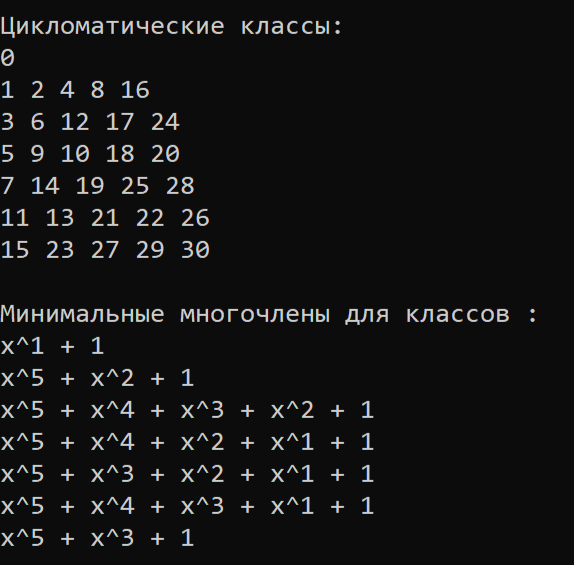
Автоматически созданное описание

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

Построение расширения поля GF(32)

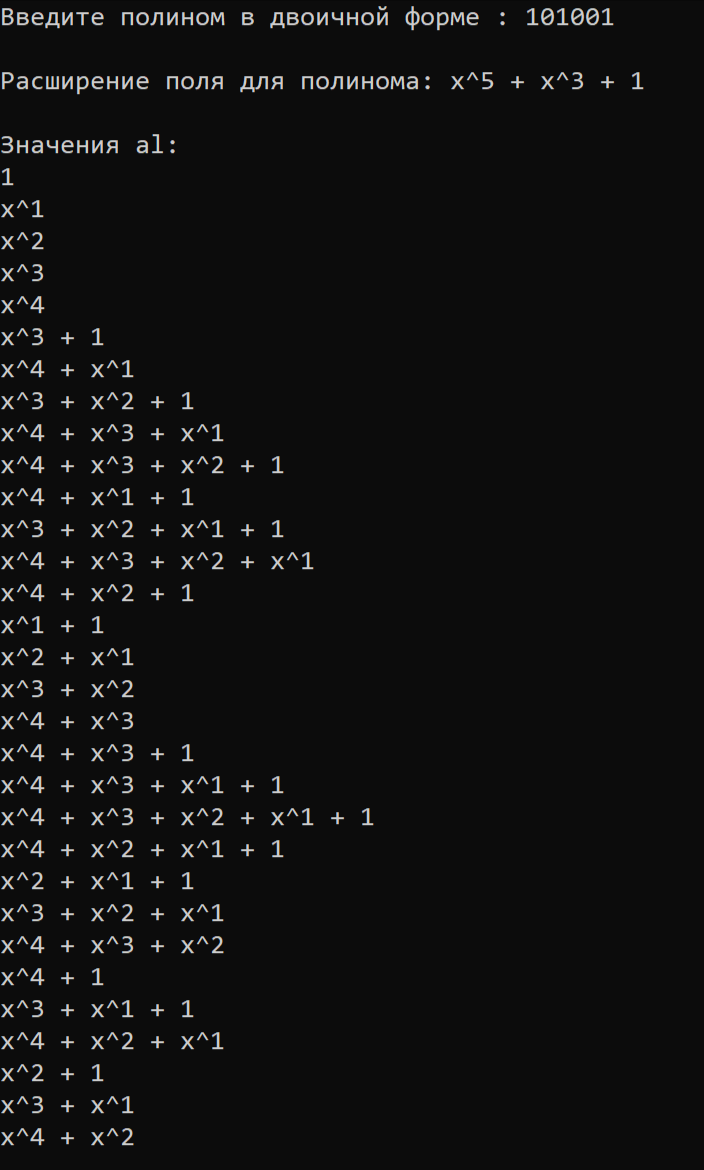


Построение циклотомических классов и минимальных многочленов для них:

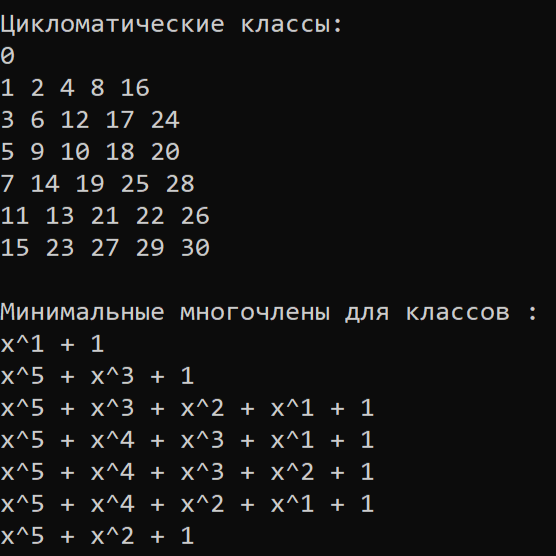


3. Построить циклотомические классы и минимальные многочлены в поле Галуа :

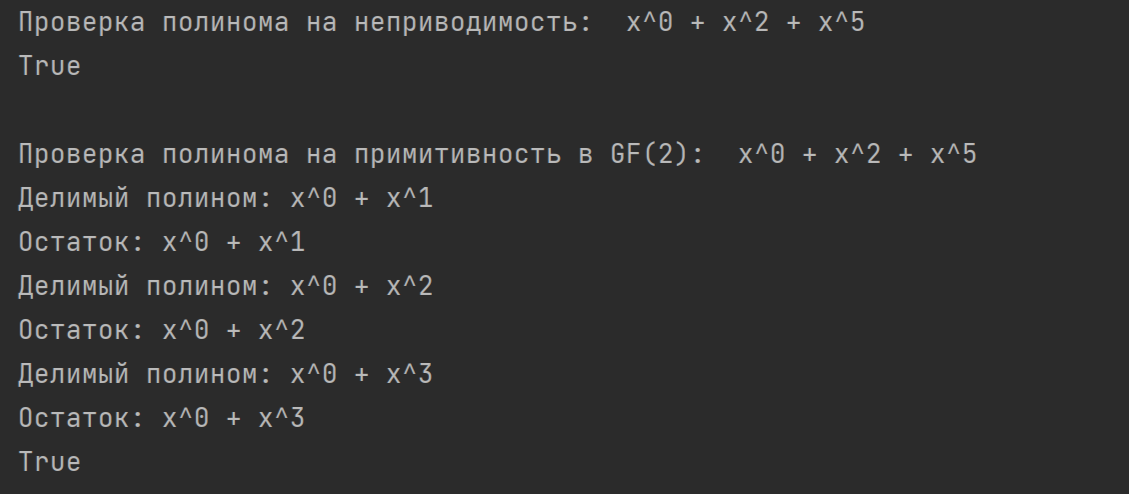
Построим расширение поля:



Построим цикломатические классы и минимальные многочлены:



4. Проверить на примитивность неприводимый многочлен 100101:

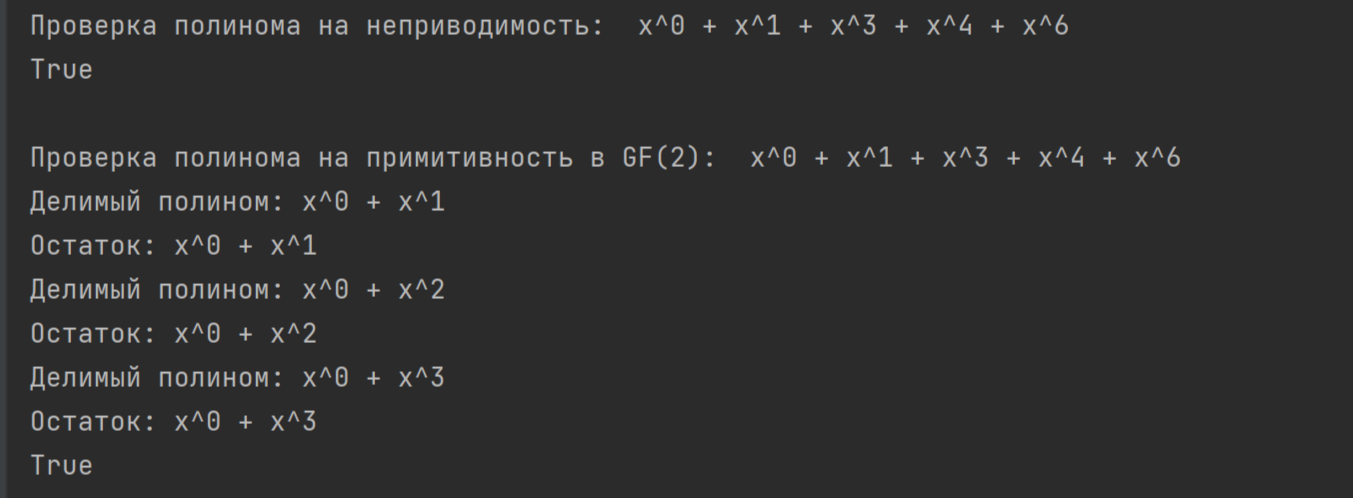


5. Проверить на примитивность неприводимый многочлен 110111:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

6. Проверить на примитивность неприводимый многочлен 1011011:



7. Проверить на примитивность неприводимый многочлен 1100001:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Листинг программы**

class Polynom:  
 #инициализация  
 def \_\_init\_\_(self, body):  
 self.body = body  
  
 #степень полинома  
 def degree(self):  
 return len(self.body) - 1  
  
 #создаем полином по степени(5=x^5)  
 @classmethod  
 def by\_degree(cls, degree):  
 new\_poly = [0] \* (degree + 1)  
 new\_poly[degree] = 1  
 return cls(new\_poly)  
  
 #получить полином по числам(4+2+1=7=111) - метод класса  
 @classmethod  
 def by\_number(cls, number):  
 new\_poly = []  
 while number:  
 #добавляем остаток от деления  
 new\_poly.append(number % 2)  
 #переходим к след разряду  
 number //= 2  
 return cls(new\_poly)  
  
 #вывод в форме полинома  
 def print(self):  
 terms = []  
 # Идем с конца последовательности  
 for i, c in enumerate(self.body):  
 #если эл-т=1  
 if c:  
 term = 'x^{}'.format(i) #форматриуем индекс под вид полинома  
 terms.append(term)  
  
 polynomial\_str = ' + '.join(terms) # Соединяем члены полинома в строку с разделителем ' + '  
  
 return polynomial\_str  
  
 #сложение полиномов  
 def add(self, b):  
 #степень итогового полинома=макс степеней  
 new\_degree = max(self.degree(), b.degree())  
 new\_poly = [0] \* (new\_degree + 1)  
 for i in range(new\_degree + 1):  
 #степень первого полинома меньше чем итоговый разряд  
 if i > self.degree():  
 #берем значение второго полинома  
 new\_poly[i] = b.body[i]  
 # степень второго полинома меньше чем итоговый разряд  
 elif i > b.degree():  
 new\_poly[i] = self.body[i]  
 else:  
 #операция xor  
 new\_poly[i] = self.body[i] ^ b.body[i]  
  
 #убираем ведущие нули  
 zero\_pad\_left = 0  
 for i in range(new\_degree, -1, -1):  
 if new\_poly[i] == 0:  
 zero\_pad\_left += 1  
 else:  
 break  
 new\_poly = new\_poly[:-zero\_pad\_left] if zero\_pad\_left > 0 else new\_poly  
  
 return Polynom(new\_poly)  
  
 #умножение полиномов  
 def multiplications(self, b):  
 #полином степени равной сумме степеней множителей self и b  
 new\_poly = [0] \* (self.degree() + b.degree() + 1)  
 #Цикл проходит по всем степеням полинома self от самой высокой до нулевой  
 for i in range(self.degree(), -1, -1):  
 #Если коэффициент при текущей степени в полиноме self равен нулю, то переходим к следующей итерации цикла.  
 if not self.body[i]:  
 continue  
 for j in range(b.degree(), -1, -1):  
 #Операция XOR между new\_poly и результатом поразрядного умножения коэффициентов при степенях i и j в полиномах self и b  
 new\_poly[i + j] ^= self.body[i] & b.body[j]  
  
 return Polynom(new\_poly)  
  
 #сравнение полиномов  
 def equals(self, b):  
 #если степени равны  
 if self.degree() == b.degree():  
 #сравниваем каждый бит  
 for i in range(b.degree(), -1, -1):  
 if self.body[i] != b.body[i]:  
 return self.body[i] > b.body[i]  
 return True  
 else:  
 #если степени не равны-первый точно больше  
 return self.degree() >= b.degree()  
  
 #деление полиномов  
 def division(self, b):  
 divided\_poly = Polynom(self.body)  
 #пока степень делимого многочлена больше  
 while divided\_poly.degree() > b.degree():  
 division\_part\_degree = divided\_poly.degree() - b.degree()  
 #берем полином степени(делимый многочлен-делитель)  
 division\_part\_poly = Polynom.by\_degree(division\_part\_degree)  
 #умножаем полученный многочлен на подобранный  
 sub\_part\_poly = division\_part\_poly.multiplications(b)  
 #прибавляем делимый многочлен с умножаемым членом(т.к. в поле +=-)  
 divided\_poly = divided\_poly.add(sub\_part\_poly)  
 #если степени равны  
 if divided\_poly.degree() == b.degree():  
 #если они равны  
 if divided\_poly.equals(b):  
 #прибавляем b(получаем ост от деления)  
 divided\_poly = divided\_poly.add(b)  
  
 return Polynom(divided\_poly.body)  
  
#является ли полином неприводимым?  
def polynom\_is\_irreducible(poly):  
 # неприводимые полиномы из таблицы  
 known\_irreducible\_polynomials = [2, 3, 7, 11, 13, 19, 25, 31, 37, 41, 47, 55, 59, 61, 67, 73, 87, 91, 97, 103, 109,  
 115, 117, 131, 137, 143, 145, 157, 167, 171, 185, 191, 193, 203, 211, 213, 229]  
 for poly\_number in known\_irreducible\_polynomials:  
 if(poly.body==Polynom.by\_number(poly\_number).body):  
 return True  
 #максимальная степень полинома для деления  
 max\_search\_degree = poly.degree() // 2  
 max\_search\_number = 2\*\*(max\_search\_degree + 1)  
  
  
 #проверяем неприводимость делением на неприводимые полиномы степени(degree/2)  
 for poly\_number in known\_irreducible\_polynomials:  
 # Проверяем, что степень полинома не превышает max\_search\_degree  
 if poly\_number <= max\_search\_number:  
 check\_poly = Polynom.by\_number(poly\_number)  
 print("Делим на:", check\_poly.print())  
 remainder = poly.division(check\_poly)  
 print("Остаток:", remainder.print())  
 if remainder.degree() == -1:  
 return False  
  
 return True  
  
#является ли полином примитивным?  
#примитивный=неприводимый+делит многочлены поля вида x^(2^i)+1 с остатком  
def polynom\_is\_primitive\_in\_gp(poly, gp\_degree):  
 #полином не приводим  
 if polynom\_is\_irreducible(poly):  
 #макс степень=степень поля  
 max\_degree = 2\*\*gp\_degree  
 for i in range(1, max\_degree):  
 poly\_number = 2\*\*i  
 check\_poly = Polynom.by\_number(poly\_number + 1)  
 print("Делимый полином:", check\_poly.print())  
 remainder = check\_poly.division(poly)  
 print("Остаток:", remainder.print())  
 #остаток=0  
 if remainder.degree() == -1:  
 return False  
 return True  
 return False  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 #неприводимый=True(из таблицы или не делится)  
 #приводимый(делится на тот что из таблицы) 1100001  
  
 poly\_true = Polynom.by\_number(64+32+1)  
 print("Проверка полинома на неприводимость: ", poly\_true.print())  
 print(polynom\_is\_irreducible(poly\_true))  
 print()  
  
 # poly\_false = Polynom.by\_number(8+2)  
 # print("Проверка полинома на неприводимость: ", poly\_false.print())  
 # print(polynom\_is\_irreducible(poly\_false))  
 # print()  
  
 print("Проверка полинома на примитивность в GF(2): ", poly\_true.print())  
 print(polynom\_is\_primitive\_in\_gp(poly\_true, 2))