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**

**ПО КУРСУ «КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К.В.Стасюк

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

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

**Задание:**

№ 11. Проверить на примитивность неприводимый многочлен 1011011

**Ход работы:**

**Класс Polynom:**

Для решения данной задачи был реализован класс Polynom.

* **Инициализация** (метод \_\_init\_\_):

Класс Polynom представляет полином в поле GF(2) (бинарные коэффициенты).

Имеет атрибут body, представляющий список коэффициентов полинома.

* **Степень полинома** (метод degree):

Возвращает степень полинома, которая равна наивысшему значащему разряду.

* **Создание полинома по степени** (метод by\_degree):

Создает полином, у которого все коэффициенты равны нулю, кроме указанной степени, которая устанавливается в 1.

* **Создание полинома по числу** (метод by\_number):

Переводит число в бинарную систему и создает полином с коэффициентами, соответствующими битам числа.

* **Вывод полинома в виде строки** (метод print):

Представляет полином в виде строки, где каждый ненулевой коэффициент соответствует слагаемому в виде .

* **Сложение полиномов** (метод add):

Складывает два полинома, применяя операцию XOR к соответствующим коэффициентам.

* **Умножение полиномов** (метод multiplications):

Умножает два полинома методом поразрядного умножения и применения операции XOR.

* **Сравнение полиномов** (метод equals):

Сравнивает два полинома по степеням и соответствующим коэффициентам. Выдает True,если первый полином больше второго.

* **Деление полиномов** (метод division):

Осуществляет деление полинома на другой полином, возвращая частное.

**Проверка на примитивность**:

Полином *poly* считается **примитивным**, если для каждого полинома +1 из рассматриваемого множества справедливо следующее условие: деление полинома +1 (где *i* принимает значения от 1 до max\_degree −1 ) на *poly* дает остаток, и этот остаток является ненулевым. max\_degree −1 равно 2^(размерность поля)

Данная функция проверяет, является ли полином примитивным в заданном поле GF(n). В программе реализован цикл, который проверяет полином в полях в размерностях поля от 2 до , m-максимальная степень, задается в программе, по умолчанию m=10.

Т.к. в условии сказано о том, что полином является неприводимым, проверка на непроводимость не проводится. Далее выбирается максимальная степень для размерности поля, после чего все полиномы вида +1 делятся на рассматриваемый полином. Если остатки не равны нулю, то полином является примитивным, программа вернет True.

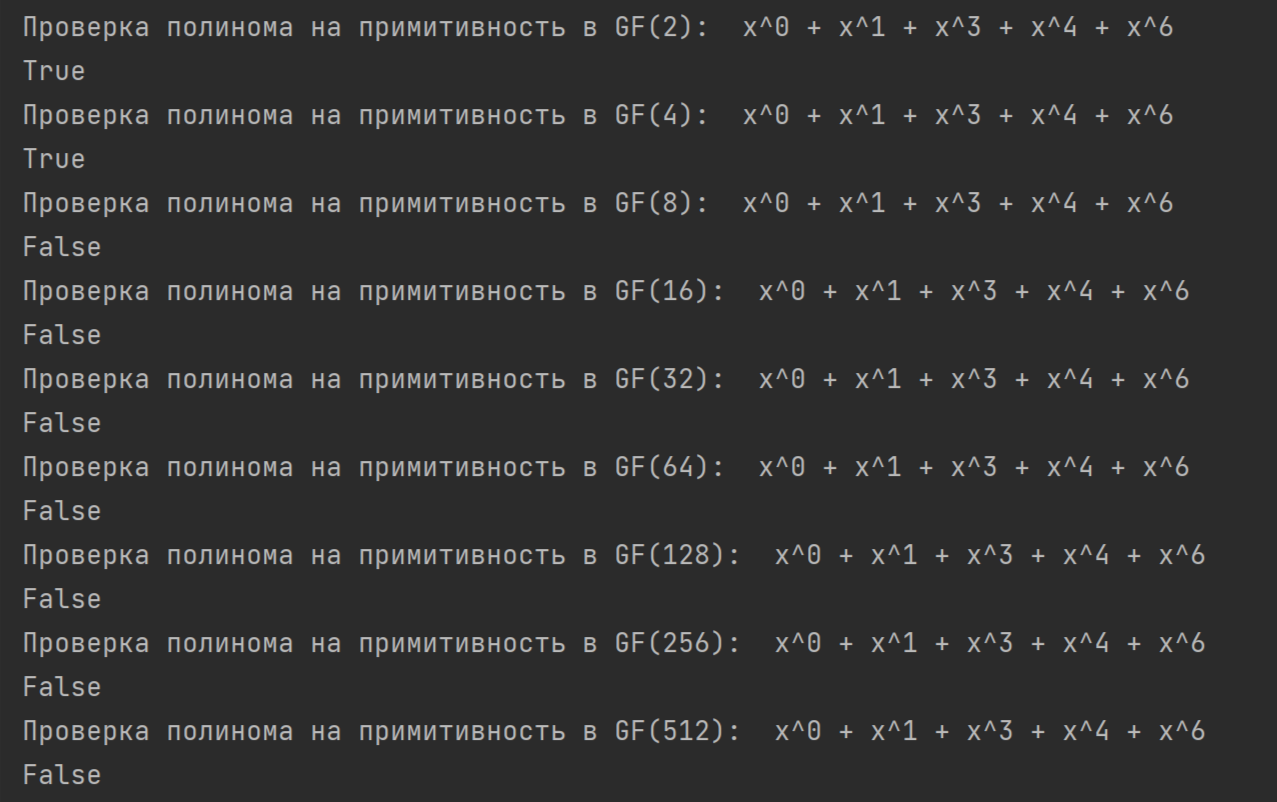


Рисунок 1 – пример работы программы.

Код 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 мтепеней  
 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 примитивным?  
#примитивный=неприводимый+делит многочлены поля вида x^(2^i)+1 с остатком  
def polynom\_is\_primitive\_in\_gp(poly, gp\_degree):  
 #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  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 max\_degree=10  
 #1011011  
 poly\_true = Polynom.by\_number(64+16+8+2+1)  
 for i in range(1,max\_degree):  
 current\_degree = 2 \*\* i  
 print(f"Проверка полинома на примитивность в GF({current\_degree}): ", poly\_true.print())  
 print(polynom\_is\_primitive\_in\_gp(poly\_true, current\_degree))