МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский ядерный университет «МИФИ»**

**Отчёт к проекту**

**на тему «ГПСЧ в поле Галуа »**

**Выполнили:**

Студенты группы Б18-503

Чуркин К.Ю.

Григорьев М.П.

Хисамутдинов М.А.

**Преподаватель:**

Иванов Михаил Александрович

**Москва 2021**

**Оглавление**

[1. Введение в поля Галуа 3](#_Toc69135220)

[1.1. Общее понятие поля Галуа 3](#_Toc69135221)

[1.2. Арифметика в поле Галуа 3](#_Toc69135222)

[1.2.1. Операция сложения 3](#_Toc69135223)

[1.2.2. Операция вычитания 4](#_Toc69135224)

[1.2.3. Операция умножения 4](#_Toc69135225)

[2. Программная реализация поля Галуа на языке С++ 5](#_Toc69135226)

[2.1. Особенности размещения кода 5](#_Toc69135227)

[2.2. Модульная арифметика 5](#_Toc69135228)

[2.2.1. Сложение 5](#_Toc69135229)

[2.2.2. Вычитание 5](#_Toc69135230)

[2.2.3. Умножение 6](#_Toc69135231)

[2.2.4. Инверсия 6](#_Toc69135232)

[2.2.5. Деление 7](#_Toc69135233)

[2.3. Арифметика полиномов 7](#_Toc69135234)

[2.3.1. Представление полиномов 7](#_Toc69135235)

[2.3.2. Немного про “контракты” 8](#_Toc69135236)

[2.3.3. Сложение 8](#_Toc69135237)

[2.3.4. Вычитание 9](#_Toc69135238)

[2.3.5. Умножение 10](#_Toc69135239)

[2.3.6. Определение степени полинома 11](#_Toc69135240)

[2.3.7. Деление 12](#_Toc69135241)

[2.4. Поле Галуа и его элементы 13](#_Toc69135242)

# Введение в поля Галуа

## Общее понятие поля Галуа

Поле Галуа — это поле из конечного числа элементов. Поле Галуа обычно обозначают , где – число элементов поля, это число называется ***порядком*** поля. Так же установлено, что , где – простое число (***характеристика*** поля), а – любое натуральное число (***степень*** поля). Таким образом получаем ещё один вариант записи поля – .

Элементами поля являются полиномы вида:

Так же существует так называемый ***примитивный*** (***порождающий***) полином поля вида:

Все элементы поля являются степенями примитивного полинома:

## Арифметика в поле Галуа

В конечном поле определены операции сложения, вычитания, умножения и деления.

Так как для создания ГПСЧ операция деления элементов поля не используется, то мы не будем её рассматривать.

### Операция сложения

Операция сложения полиномов выполняется с каждым коэффициентом отдельно.

Исходные элементы поля:

По правилу сложения:

Получаем:

### Операция вычитания

Операция вычитания полиномов так же выполняется с каждым коэффициентом отдельно.

Исходные элементы поля:

По правилу вычитания:

Получаем:

### Операция умножения

Исходные элементы поля:

Перемножив их, а так же приведя соответствующие слагаемые и получим новый полином :

В зависимости от коэффициентов степень существует две ситуации:

1. Степень полинома меньше полученный полином принадлежит
2. Степень полинома больше, либо равна полученный полином не принадлежит

# Программная реализация поля Галуа на языке С++

## Особенности размещения кода

В файлах с расширением .h хранятся так называемые определения *типов* и *функций*. В файлах с расширением .cpp хранятся реализации этих *типов* и *функций*.

## Модульная арифметика

Модульная арифметика является основой всех последующих операций с полиномами в поле Галуа, поэтому нужно начать именно с её реализации.

Для начала обозначим целочисленный *тип*, с которым мы будем работать. Это будет целое число, которое занимает 64 бита.

Файл ModularArithmetic.h:

using Int = long long int;

### Сложение

Функция модульного сложения принимает три аргумента: левый операнд, правый операнд и сам модуль соответственно. Возвращает результат модульного сложения.

Файл ModularArithmetic.h:

Int modular\_add(const Int left, const Int right, const Int mod);

Файл ModularArithmetic.cpp:

Int modular\_add(const Int left, const Int right, const Int mod)

{

return (left + right) % mod;

}

### Вычитание

Функция модульного вычитания принимает три аргумента: левый операнд, правый операнд и модуль соответственно. Возвращает результат модульного вычитания.

Файл ModularArithmetic.h:

Int modular\_sub(const Int left, const Int right, const Int mod);

Файл ModularArithmetic.cpp:

Int modular\_sub(const Int left, const Int right, const Int mod)

{

return (left + (mod - right)) % mod;

}

### Умножение

Функция модульного умножения принимает три аргумента: левый операнд, правый операнд и модуль соответственно. Возвращает результат модульного умножения.

Файл ModularArithmetic.h:

Int modular\_mul(const Int left, const Int right, const Int mod);

Файл ModularArithmetic.cpp:

Int modular\_mul(const Int left, const Int right, const Int mod)

{

return (left \* right) % mod;

}

### Инверсия

Функция модульной инверсии принимает два аргумента: операнд и модуль соответственно. Возвращает результат модульной инверсии. Так же функция использует функцию быстрого возведения в степень.

Файл ModularArithmetic.h:

Int fast\_power(Int base, Int power, Int mod);

Int modular\_inv(const Int left, const Int mod);

Файл ModularArithmetic.cpp:

Int fast\_power(Int base, Int power, Int mod) {

Int result = 1;

while (power > 0) {

if ((power & 1) == 1) {

result = (result \* base) % mod;

}

base = (base \* base) % mod;

power = power >> 1;

}

return result;

}

Int modular\_inv(const Int left, const Int mod)

{

return fast\_power(left, mod - 2, mod);

}

### Деление

Функция модульного деления использует мольную инверсию числа и выполняется как модульное умножение левого операнда и инверсии правого.

Принимает три аргумента: левый операнд, правый операнд и модуль соответственно. Возвращает результат модульного деления. Если правый операнд(делитель) равен 0, то функция “кидает” соответствующее *исключение*.

Файл ModularArithmetic.h:

Int modular\_mul(const Int left, const Int right, const Int mod);

Файл ModularArithmetic.cpp:

Int modular\_mul(const Int left, const Int right, const Int mod)

{

if (right == 0) {

throw std::exception("divide by zero");

}

return (left \* modular\_inv(right, mod)) % mod;

}

## Арифметика полиномов

Арифметика полиномов является промежуточной ступенью между арифметикой элементов поля и модульной арифметикой, поэтому следующей стадией станет реализация функций для полиномов.

### Представление полиномов

Так как в полиномах обозначает лишь позицию, то можно хранить только коэффициенты полиномов. При этом, если в полиноме нет какого-то , то будет соответственно указан коэффициент равный 0.

Теперь нужно установить тип, которым будут представлены наборы коэффициентов. Это будет std::vector<Int>. Этот тип является реализацией концепции динамического массива (***вектора***) и находится в Стандартной Библиотеке Шаблонов (STL) языка С++.

Файл PolynomialArithmetic.h:

using CoefficientsVector = std::vector<Int>;

### Немного про “контракты”

Существует практика установления определённых “контрактов” для функций. Контракты обозначаются лишь на словах и устанавливают определённые требования для входных данных

Воспользуемся этим и скажем, что два входных вектора коэффициентов обязаны быть одинакового размера. Это очень важное уточнение, так как оно позволяет существенно упростить функции и отказаться от специфичных реализаций. Естественно, перед вызовом функции нужно убедиться, что данные соответствуют контракту.

### Сложение

Сложение полиномов производится с помощью модульного сложения коэффициентов у соответствующего .

Функция сложения полиномов принимает три аргумента: вектор коэффициентов первого (левого) операнда, вектор коэффициентов первого (правого) операнда и модуль. Возвращает вектор коэффициентов результирующего полинома.

Файл PolynomialArithmetic.h:

CoefficientsVector polynomial\_add(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod);

Файл PolynomialArithmetic.cpp:

CoefficientsVector polynomial\_add(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod

)

{

auto size = left.size();

if (size != right.size()) {

throw std::exception(

"dimensions of the polynomials are not equal");

}

CoefficientsVector result(size);

for (size\_t i = 0; i < size; ++i) {

result[i] = modular\_add(left[i], right[i], mod);

}

return result;

}

### Вычитание

Вычитание полиномов производится с помощью модульного вычитания коэффициентов у соответствующего .

Функция вычитания полиномов принимает три аргумента: вектор коэффициентов первого (левого) операнда, вектор коэффициентов первого (правого) операнда и модуль. Возвращает вектор коэффициентов результирующего полинома.

Файл PolynomialArithmetic.h:

CoefficientsVector polynomial\_sub(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod);

Файл PolynomialArithmetic.cpp:

CoefficientsVector polynomial\_sub(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod

)

{

auto size = left.size();

if (size != right.size()) {

throw std::exception(

"dimensions of the polynomials are not equal");

}

CoefficientsVector result(size);

for (size\_t i = 0; i < size; ++i) {

result[i] = modular\_sub(left[i], right[i], mod);

}

return result;

}

### Умножение

Умножение полиномов производится обычным раскрытием скобок с помощью модульного умножения коэффициентов.

Функция умножения полиномов принимает три аргумента: вектор коэффициентов первого (левого) операнда, вектор коэффициентов первого (правого) операнда и модуль. Возвращает вектор коэффициентов двойной длины результирующего полинома.

Файл PolynomialArithmetic.h:

CoefficientsVector polynomial\_mul(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod);

Файл PolynomialArithmetic.cpp:

CoefficientsVector polynomial\_mul(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod

)

{

CoefficientsVector result(left.size() + right.size(), 0);

for (size\_t i = 0; i < result.size(); ++i) {

for (size\_t j = 0; j <= i; ++j) {

if (i - j < right.size() && j < left.size()) {

decltype(auto) temp =

modular\_mul(

left[j],

right[i - j],

mod

);

result[i] = modular\_add(result[i], temp, mod);

}

}

}

return result;

}

### Определение степени полинома

Определение степени полинома является довольно полезной функцией при установленном ранее представлении полиномов. Эта функция требуется для алгоритмов, в которых нужно знать степень полинома именно на данный момент, например, в делении.

Функция принимает один аргумент – вектор коэффициентов.

Возвращает индекс первого старшего коэффициента не равного 0.   
Если все коэффициенты равны 0, то возвращает -1.

Файл ModularArithmetic.h:

Int polynomial\_degree(

const CoefficientsVector& polynomial\_coefficients

);

Файл ModularArithmetic.cpp:

Int polynomial\_degree(

const CoefficientsVector& polynomial\_coefficients

)

{

for (Int i = polynomial\_coefficients.size() - 1; i >= 0; --i) {

if (polynomial\_coefficients[i]) return i;

}

return -1;

}

### Деление

Деление полиномов производится обычным делением столбиком с использованием модульной арифметики.

Функция деления полиномов принимает три аргумента: вектор коэффициентов первого (левого) операнда, вектор коэффициентов первого (правого) операнда и модуль. Возвращает пару векторов-коэффициентов из частного и остатка соответственно.

Файл PolynomialArithmetic.h:

std::pair<CoefficientsVector, CoefficientsVector> polynomial\_div(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod);

Файл PolynomialArithmetic.cpp:

std::pair<CoefficientsVector, CoefficientsVector> polynomial\_div(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod

)

{

decltype(auto) right\_degree = polynomial\_degree(right);

if (right\_degree == -1) throw std::exception("divide by zero");

CoefficientsVector quotient(left.size(),0);

CoefficientsVector remainder = left;

CoefficientsVector qRight(left.size(), 0);

decltype(auto) remainder\_degree = polynomial\_degree(remainder);

while (remainder\_degree >= right\_degree) {

decltype(auto) q = modular\_div(

remainder[remainder\_degree],

right[right\_degree],

mod);

decltype(auto) diff = remainder\_degree - right\_degree;

quotient[diff] = q;

for (Int i = right\_degree; i >= 0; --i)

qRight[i + diff] = modular\_mul(right[i], q, mod);

for (Int i = 0; i < diff; ++i)

qRight[i] = 0;

for (Int i = 0; i <= remainder\_degree; ++i) {

remainder[i] = modular\_sub(

remainder[i], qRight[i], mod);

remainder\_degree = polynomial\_degree(remainder);

}

return { quotient, remainder };

}

## Поле Галуа и его элементы

Поле описывает *класс* (*тип*) GaloisField, элемент поля – GFElement.

Функции, принадлежащие классу – *методы класса*.

Данные, принадлежащие классу – *поля класса*.

Так как эти два класса имеют сильную внутреннюю связь, то сперва нужно определить оба класса, и только после переходить к их реализации.

Файл GFElement.h:

class GFElement {

private:

CoefficientsVector \_coefficients;

GaloisField& \_field;

private:

void field\_assert(const GFElement& other) const;

public:

GFElement operator+(const GFElement& other) const;

GFElement operator-(const GFElement& other) const;

GFElement operator\*(const GFElement& other) const;

bool operator == (const GFElement& other) const noexcept;

bool operator != (const GFElement& other) const noexcept;

bool operator < (const GFElement& other) const noexcept;

bool operator > (const GFElement& other) const noexcept;

bool operator <= (const GFElement& other) const noexcept;

bool operator >= (const GFElement& other) const noexcept;

public:

GFElement mod\_add(const GFElement& other) const;

public:

GFElement() = delete;

GFElement(const GFElement& other);

GFElement(GFElement&& other) noexcept;

GFElement(const CoefficientsVector& coefficients,

GaloisField& field);

GFElement(CoefficientsVector&& coefficients, GaloisField& field);

GFElement(const GFElement& left, const GFElement& right,

decltype(polynomial\_add) polynomial\_operator);

GFElement& operator=(const GFElement& other);

GFElement& operator=(GFElement&& other) noexcept;

public:

friend std::ostream& operator << (std::ostream& out,

const GFElement& element);

public:

Int to\_int() const;

};

Элемент поля определяют его коэффициенты (*поле класса* \_coefficients) и поле (*поле класса* \_field), к которому оно принадлежит.

Файл GaloisField.h:

class GaloisField {

private:

Int \_characteristic;

Int \_dimension;

CoefficientsVector \_reduction\_polynomial;

public:

GaloisField(Int characteristic, Int dimension,

CoefficientsVector reduction\_polynomial);

public:

Int characteristic() const noexcept;

Int dimension() const noexcept;

const CoefficientsVector& reduction\_polynomial() const noexcept;

GFElement create\_polynomial(

const CoefficientsVector& coefficients);

GFElement create\_polynomial(CoefficientsVector&& coefficients);

std::vector<GFElement> all\_field\_elements();

public:П

bool operator == (const GaloisField& other) const noexcept;

bool operator != (const GaloisField& other) const noexcept;

};

Поле определяет его ***характеристика*** (*поле класса* \_characteristic), ***степень*** поля (*поле класса* \_dimension) и ***порождающий*** полином (*поле класса* \_reduction\_polynomial).

Здесь поле является своеобразной меткой для элементов поля, имеют ли они право производить арифметические операции друг с другом или нет. В то время как основное взаимодействие берут на себя элементы поля.

### Элементы поля

Функция field\_assert(…) проверяет совпадают ли поля. Она используется во всех арифметических и логических операциях.

Перегруженные операторы сложения, вычитания и умножения используют рассмотренные выше арифметические функции для работы с полиномами.

Сложение - polynomial\_add(…)

Вычитание - polynomial\_sub(…)

Умножение – сначала polynomial\_mul(…), после чего берётся остаток от деления на порождающий полином, получаемый через polynomial\_div(…).

Операторы сравнения используют последовательное сравнение, начиная со старших разрядов.

Оператор “<<” используется для вывода информации о элементе поля.

Функция to\_int() используется для перевода поразрядного представления числа в десятичное.

Особого внимания заслуживает функция mod\_add(…), так как она используется для поразрядного модульного сложения с модулем , а не сложения в поле и нужна для ГПСЧ.

### Поле Галуа

Как и говорилось ничего особенного здесь нет.

Функции characteristic(), dimension() и reduction\_polynomial() используются для получения соответствующих параметров поля.

Функция create\_polynomial(…) нужна для создания элементов поля.

Перегруженные операторы сравнения используются для проверки, являются ли два объекта одним и тем же полем.