ПОЛЯ ГАЛУА – ВЫЧИСЛЕНИЯ НА ПЛАТФОРМЕ JULIA[1]

GALOIS FIELDS – JULIA PLATFORM COMPUTING

А.В. Рожков, Е.О. Филонцева, Р.Л. Репин

A.V. Rozhkov, E.O. Filontseva, R.L. Repin

Поля Галуа, примитивный элемент, язык программирования Julia, криптография.
В рамках освоения магистерского курса по криптографии производятся вычисления в полях Галуа средствами нового языка программирования Julia.

Galois fields, primitive element, Julia programming language, cryptography.
As part of the master's course in cryptography, calculations are made in the Galois fields using the new Julia programming language.

В рамках реализации проекта, поддержанного грантом фонда Владимира Потанина ГСГК-0072-21, разрабатывается ряд курсов для магистерской программы “Алгебраические методы защиты информации”, открытой в Кубанском государственном университете в 2013 г. В данной работе речь идет о курсе “Теоретико-числовые методы криптографии”. Основой курса – теория полей Галуа.

Julia – высокопроизводительный язык программирования с динамической типизацией, созданный для математических вычислений. Синтаксис языка схож с MATLAB и Python. Julia написан на Си, C++ и Scheme. В стандартный комплект входит JIT-компилятор LLVM, благодаря чему он не уступает в производительности компилируемому языку C/C++.

Язык имеет встроенную поддержку распределенных и параллельных вычислений. Более того, в код Julia можно включать модули и библиотеки, написанные на языках C/C++, Fortran, Python, Java.

Julia включает в себя множество пакетов, с помощью которых можно производить алгебраические вычисления. Язык активно развивается, и уже имеется 6500 официально принятых расширяющих пакетов и более 10 тыс. еще не сертифицированных. Каждый день добавляется 2–3 новых пакета. Обширный функционал позволяет использовать экосистему Julia как систему компьютерной алгебры.

В области алгебры базовые пакеты Nemo v0.27.0, AbstractAlgebra v0.22.1, GaloisFields v1.1.1, Primes v0.5.0, LinearAlgebra, Hecke, SymPy v1.0.52. Для построения графиков хорош мощный пакет Plots.

Julia работает очень устойчиво, месяцами не загружая память более, чем на 400 Mb, чем очень выгодно отличается от GAP (https://www.gap-system.org/), который через 2–3 часа работы приходится перезагружать.

Julia 1.6.3 (https://julialang.org/) имеет и 32 и 64 битные версии под Windows и Linux.

[1] Проект реализуется победителем Конкурса на предоставление грантов преподавателям магистратуры благотворительной программы «Стипендиальная программа Владимира Потанина» Благотворительного фонда Владимира Потанина.

Вычисления в полях Галуа

Запускаем Julia в терминале REPL, можно это сделать и различных редакторах типа VS Code, и подключаем пакет Nemo

```
julia> using Nemo
Welcome to Nemo version 0.27.0
Nemo comes with absolutely no warranty whatsoever
```

У пакета Nemo есть одна методическая особенность. Полями Галуа в нем называются простые поля Галуа GF(p), которые реализуются как кольца вычетов по простому модулю.

```
julia> F=GF(7)
Galois field with characteristic 7
julia> F(3)^3
6
julia> F(3^3)
6
```

Произвольные поля Галуа называются конечными полями.

Конечное поле можно задать двумя способами.

Первый способ как абстрактное поле с неизвестным нам порождающим многочленом:

```
julia> R, x = FiniteField(5, 2, "x")
(Finite field of degree 2 over F_5, x)
```

Но мы можем легко выяснить, какой это многочлен

```
julia> x^2
x+3
```

Значит это многочлен $x^2 = x + 3 \Rightarrow x^2 - x - 3 = x^2 + 4x + 2$.

Второй способ. Вначале создадим кольцо T многочленов над полем вычетов, а потом зададим расширение простого поля Галуа, как поля разложения U нашего многочлена:

```
julia> T, t = PolynomialRing(ResidueRing(ZZ, 5), "t")
(Univariate Polynomial Ring in t over Integers modulo 5, t)
julia> U, z = FiniteField(t^2+t + 1, "z")
(Finite field of degree 2 over F_5, z)
```

Модельные задачи

Задача № 1. *Для многочлена* $x^{17} - x + 3$ *над полем GF(199) найти корни.*

Решим задачу, не используя конечные поля.

```
julia> function f(p)
       for i in 1:p
       if (i^17-i+3)%p == 0
       print(i,",")
       end
       end
       end
```

```
f (generic function with 1 method)
julia> f(199)
25,199,
```

Получилось, что 199, т.е. 0 в поле GF(199), является корнем, что неверно. В чем причина? В разрядности целых чисел

```
julia> 199^17 – это Int64 19-значные числа
5245870108793248583
julia> BigInt(199)^BigInt(17) – допустимы миллионы знаков
120365576140143338953454431235743456339 9
```

Немного изменим нашу программку

```
julia> function f(p)
       for i in 1:p
       if (BigInt(i)^BigInt(17)-i+3)%p == 0
       print(i,",")
       end
       end
       end
f (generic function with 1 method)
julia> f(199)
28,149,
```

Корни получились другие.

Теперь подключим поля Галуа

```
using Nemo
julia> function f(p)
       for i in GF(p)
       if i^17-i+3 == 0
       print(i,",")
       end
       end
       end
f (generic function with 1 method)
julia> f(199)
28,149,
```

Теперь корни совпали.

Задача № 2. *Найти примитивный элемент поля* $GF(5^2)$, *заданного как поле разложения многочлена* $f(x) = x^2 + x + 1, GF(5)$.

Решение. Используем второй способ задания поля Галуа. Вначале создадим кольцо T многочленов над полем вычетов, а потом зададим расширение простого поля Галуа, как поля разложения U нашего многочлена.

```
julia> T, t = PolynomialRing(ResidueRing(ZZ, 5), "t")
(Univariate Polynomial Ring in t over Integers modulo 5, t)
julia> U, z = FiniteField(t^2+t + 1, "z")
(Finite field of degree 2 over F_5, z)
```

Проверим является ли элемент z примитивным. К сожалению, нет:

```
julia> z^8
4*z+4
julia> z^12
1
```

Проверим элемент y = z+1. Тоже не подходит:

```
julia> y=z+1
z+1
julia> y^8
z
julia> y^12
1
```

Проверим y = z+2. “Упорство и труд – все перетрут!” – подходит

```
julia> y=z+2
z+2
julia> y^8
4*z+4
julia> y^12
4
```

Итак, y = *z*+2 – примитивный элемент нашего поля.

Задачи 3. *Составить таблицу степеней примитивного элемента и таблицу логарифма Якоби.*

Решение. Вычислим все 24 степени элемента y:

```
julia> y=z+2
z+2
julia> for i in 1:24
       print(y^i, ",")
       end
z+2,3*z+3,z+3,4*z,4*z+1,3,3*z+1,4*z+4,3*z+4,2*z,2*z+3,4,4*z+3,2*z+2,4*z+2,z,
z+4,2,2*z+4,z+1,2*z+1,3*z,3*z+2,1
```

Составим объединенную таблицу – первая строка – показатели степеней примитивного элемента y, вторая строка – значение его степеней, третья -логарифм Якоби, который строится просто просмотром первых двух строк. Например, как найти L(10), по определению логарифма $1+y^{10}=y^{L(10)}$. Находим $y^{10}=2z \Rightarrow y^{10}+1=2z+1=y^{21}$, таким образом L(10)=21. Отметим, т.к. 0 не является степенью примитивного элемента, то в 12-столбце вместо значения логарифма стоит символ запрета или останова в машинах Тьюринга – знак #.

В итоге получаем:

i	1	2	3	4	5	6	7	8	9	10	11	12
y^i	z+2	3z+3	z+3	4z	4z+1	3	3z+1	4z+4	3z+4	2z	2z+3	4
L(i)	3	9	17	5	15	12	23	4	22	21	19	#

13	14	15	16	17	18	19	20	21	22	23	24
4z+3	2z+2	4z+2	z	z+4	2	2z+4	z+1	2z+1	3z	3z+2	1
8	11	13	20	16	6	10	1	14	7	2	18

Применение языка Julia при изучении разделов алгебры, требующих больших вычислений, очень естественно и продуктивно.

Библиографический список

1. Глухов М.М., Круглов И.А., Пичкур А.Б., Черемушкин А.В. Введение в теоретико-числовые методы криптографии [Электронный ресурс]. СПб.: Лань, 2021. URL: https://e.lanbook.com/reader/book/153680