МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Алгоритм Берлекэмпа

ЛАБОРАТОРНАЯ РАБОТА

студента 4 курса 431 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Серебрякова Алексея Владимировича

Научный руководитель
доцент, к. п. н. _____ А. С. Гераськин
подпись, дата

Саратов 2022

ВА. Алгоритм Берлекэмпа (**Berlekamp's Algorithm**)

Вход: Нормированный свободный от квадратов полином $p(x)$ над $GF(p)$, $\deg[p(x)] = n$.

Выход: Неприводимые сомножители полинома $p(x)$ над $GF(p)$.

1. [Построение матрицы **Q**] Построить $n \times n$-матрицу **Q** так, как описано в (B9). Как показано ниже, это можно сделать одним из двух способов в зависимости от того, насколько велико число p.

2. [Триангуляризация $\mathbf{Q}-\mathbf{I}$] Привести матрицу $\mathbf{Q}-\mathbf{I}$ к треугольному виду, вычислив ее ранг $n-r$ и найдя нуль-пространство матрицы $\mathbf{Q}-\mathbf{I}$, т.е. найти r линейно независимых векторов $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_r$, таких, что $\mathbf{b}_j[\mathbf{Q}-\mathbf{I}] = 0$ для $1 \leqslant j \leqslant r$. [Первый вектор всегда может быть выбран в виде $(1, 0, \ldots, 0)$, что представляет тривиальное решение $b_1(x) = 1$ уравнения (B2). Приведение к треугольному виду может быть осуществлено так, как описано в разд. 5.3.3, или с использованием представленного ниже алгоритма **NS**.] *В этой точке r — это число неприводимых сомножителей полинома $p(x)$*, поскольку решениями уравнения (B2) являются p^r полиномов, соответствующих векторам $a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + \cdots + a_r\mathbf{b}_r$ при любом выборе целых чисел $0 \leqslant a_1, \ldots, a_r \leqslant p$. Поэтому, если $r = 1$, то полином $p(x)$ неприводим, и алгоритм заканчивает работу.

3. [Вычисление сомножителей] Пусть $b_2(x)$ — полином, соответствующий вектору $\mathbf{b}_2$. Вычислим $\gcd[p(x), b_2(x) - s]$ для всех $s \in GF(p)$. В результате по теореме 6.2.15 получим нетривиальное разложение полинома $p(x)$. Если с использованием $b_2(x)$ получено менее r сомножителей, вычислим $\gcd[w(x), b_k(x) - s]$ для всех $s \in GF(p)$ и всех сомножителей $w(x)$, найденных к данному времени, для $k = 3, 4, \ldots, r$, пока не найдем r сомножителей. Теорема 6.2.18 гарантирует, что таким образом мы найдем все сомножители полинома $p(x)$. Если p мало, то вычисления на данном шаге весьма эффективны. Однако для больших p (например, $p > 25$) может быть предложен лучший способ, разбираемый ниже.

Пусть

$$\mathbf{Q} = \begin{bmatrix} r_{0,0} & r_{0,1} & \cdots & r_{0,n-1} \\ r_{1,0} & r_{1,1} & \cdots & r_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n-1,0} & r_{n-1,1} & \cdots & r_{n-1,n-1} \end{bmatrix} \qquad \text{(B9)}$$

— матрица, строки которой образуют коэффициенты полиномов-остатков $r_0(x), \ldots, r_{n-1}(x)$. (*Замечание.* Сначала выписываются коэффициенты меньших степеней x.) Тогда имеет место

```
[alse0722@alse0722-ms7a34 coding_teory]$ /bin/python /home/alse0722/Desktop/univer/coding_teory/fin/n18.py
[1]
[1 0 0]
[1 0 0 0 0]
[1 0 0 0 0 0 0]

[[0. 0. 0. 0.]
 [0. 1. 1. 0.]
 [1. 1. 0. 0.]
 [1. 1. 0. 0.]]
2
[alse0722@alse0722-ms7a34 coding_teory]$
```

```
import warnings
from sympy.polys.galoistools import gf_monic
from sympy.polys.galoistools import gf_div
from sympy.polys.galoistools import gf_sub_mul
from sympy.polys.galoistools import gf_mul_ground
from sympy.polys.domains import ZZ
import numpy.core.numeric as NX
from numpy.lib.twodim_base import diag
from numpy.linalg import eigvals
from numpy.core import (hstack)
import numpy as np

def berlekamp():
    from numpy.polynomial import Polynomial as P
    import numpy as np

    from numpy.linalg import matrix_rank
    import fractions

    p = np.array([1, 0, 1, 1, 1])

    p1 = np.array([1])
    p2 = np.array([1, 0, 0])
    p3 = np.array([1, 0, 0, 0, 0])
    p4 = np.array([1, 0, 0, 0, 0, 0, 0])

    print(p1)
    print(p2)
    print(p3)
    print(p4)
    print()

    a1 = np.absolute(np.polydiv(p1, p)[1])[::-1].tolist()
    a2 = np.absolute(np.polydiv(p2, p)[1])[::-1].tolist()
    a3 = np.absolute(np.polydiv(p3, p)[1])[::-1].tolist()
    a4 = np.absolute(np.polydiv(p4, p)[1])[::-1].tolist()

    for i in range(len(a4)-len(a1)):
        a1.append(0)
```

```
    for i in range(len(a4)-len(a2)):
        a2.append(0)

    for i in range(len(a4)-len(a3)):
        a3.append(0)

    for i in range(len(a4)-len(a4)):
        a4.append(0)

    x = np.array([a1, a2, a3, a4])

    y = np.absolute(
        x-np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]))
    print(y)

    r = matrix_rank(y)

    print(r)

berlekamp()
```