Университет ИТМО

Отчет

Реализация алгоритма Кантора-Зассенхауза разложения на множители многочленов от одной переменной над конечным полем

Выполнила: студентка гр. M33381
Гречишкина Дарья

Содержание

Введение 1
1 Сведение к общему случаю 2
2 Алгоритм Кантора-Зассенхауза 3
3 Реализация алгоритма 4
4 Тестирование и анализ производительности 6
Список использованных источников 8

Введение

Многочлен $f \in F[x_1, \dots, x_n]$ называется **неприводимым**, если $f \notin F$ и для любых $g, h \in F[x_1, \dots, x_n]$ верно, что $g \in F$ или $h \in F$.

Для любого поля F многочлены в $F[x_1, \dots, x_n]$ могут быть единственным (с точностью до порядка множителей и произведения на константу) разложены (факторизованы) на произведение неприводимых многочленов(факторов).

Фундаментальная вычислительная задача состоит в том, чтобы найти разложение на неприводимые множители унитарного многочлена $f(x) \in F[x]$, то есть в том, чтобы найти такие $f_1, \dots, f_r \in F[x]$ и $e_1, \dots, e_r \in \mathbf{N}$ что

$$f(x) = f_1^{e_1} f_2^{e_2} \dots f_r^{e_r},$$

где F - конечное поле с характеристикой p с $q = p^d$ элементами.

1 Сведение к общему случаю

Прежде всего покажем, что определенную в предыдущей главе задачу можно свести к проблеме поиска разложения многочлена без кратных неприводимых сомножителей, т.е. такого разложения, в котором $e_1 = e_2 = \cdots = e_r = 1$.

Заметим, что если a – кратный корень многочлена $f(x)$, то a так же является корнем $f'(x)$. Удалим из $f(x)$ кратные факторы с помощью следующего рекурсивного алгоритма:

$d(x) = gcd(f(x), f'(x))$

а) Если $d(x) = 1$, то многочлен $f(x)$ не содержит кратных корней.

б) Если $d(x) = f(x)$, то $f'(x) = 0$. А значит $f(x) = g^p(x)$, где p – характеристика поля F_q. Применим указанный алгоритм к $g(x)$ рекурсивно, получим $f(x) = h^{p^s}(x)$, $h'(x) \neq 0$

в) Иначе многочлен $d(x)$ является нетривиальным делителем $f(x)$, $\dfrac{f(x)}{d(x)}$ не имеет кратных неприводимых сомножителей, а к $d(x)$ можно рекурсивно применить наш алгоритм. Зная разложения $\dfrac{f(x)}{d(x)}$ и $d(x)$ нетрудно получить разложение $f(x)$.

Далее будем считать, что $f(x)$ - многочлен без кратных факторов.

Упростим задачу еще больше, сведя поиск разложения $f(x)$ к поиску разложения многочлена, все факторы которого имеют одинаковую степень. Для этого представим $f(x) = u_1(x) \ldots u_k(x)$, где $u_j(x)$ - многочлен, являющийся произведением неприводимых многочленов степени j.

Пусть $f_1(x) = f(x)$ и для $j = 1, 2, \ldots n$ определим $u_j(x) = gcd(f_j(x), x^{q^j} - x)$ и $f_{j+1}(x) = \dfrac{f_j(x)}{u_j(x)}$. n – номер итерации, на которой $f_{j+1}(x) = const$.

Таким образом задача факторизации произвольного унитарного многочлена может быть сведена к задаче факторизации унитарного многочлена без кратных факторов, все факторы которого имеют одинаковую степень.

2 Алгоритм Кантора-Зассенхауза

Пусть $f(x) \in F[x]$ - многочлен степени n, $u_1(x)$,$u_2(x), \ldots, u_r(x) \in F[x]$ – попарно различные неприводимые многочлены, $f(x) = u_1(x)u_2(x)\ldots u_r(x)$, $deg(u_i) = \frac{n}{r} = s$.

Рассмотрим кольцо $F[x]/\langle f(x)\rangle$. Поскольку $u_i(x)$ попарно взаимнопросты, тогда по китайской теореме об остатках:

$$F_q[x]/(f(x)) \cong F_q[x]/(u_1(x)) \times F_q[x]/(u_2(x)) \times \cdots \times F_q[x]/(u_r(x))$$

Поскольку $u_i(x)$ – неприводимый многочлен степени s, $F_q[x]/(u_i(x))$ изоморфно F_{q^s}.

Рассмотрим u – делитель нуля в $F_q[x]/(f(x))$: u изоморфен некоторому $a = (a_1, \ldots, a_r)$ по китайской теореме об остатках. Будем рассматривать только такие u, для которых $\exists i, j : a_i = 0, a_j \neq 0$ (таким образом мы убираем из рассмотрения тривиальные факторы $u = 1$ и $u = f$). Таким образом, мы сводим задачу поиска факторизации к задаче поиска делителей нуля в $F_q[x]/(f(x))$.

Рассмотрим следующий рандомизированный алгоритм поиска делителя нуля:

а) Выберем случайно полином $a(x)$, $0 < deg(a(x)) < n$. Если $gcd(a(x), f(x)) \neq 1$ – нам повезло, возвращаем $a(x)$. Иначе $a \in (F_q[x]/f(x))*$, a – братимый элемент.

б) $a_i \in F_{q^s}$, F_{p^s} – абелева группа порядка $q^s - 1$, поэтому для любого элемента $b \in F_{p^s}$, $b^{q^s-1} = 1$. То есть, $b^{\frac{q^s-1}{2}} \in \{-1, 1\}$. Т.е. $a' = b^{\frac{q^s-1}{2}} + 1$ содержит нули хотя бы на одном из индексов с вероятностью $1 - \frac{1}{2^{r-1}}$. Если $a' \neq 0$ и $gcd(a', f) \neq 1$, то возвращаем $gcd(a', f)$, иначе повторим выбор a еще раз.

Алгоритм успешно завершается на очередной итерации с вероятностью $1 - \frac{1}{2^{r-1}} \geqslant \frac{1}{2}$. Время работы каждой итерации – $O(n^2 \log q)$.

Описанный выше алгоритм впервые предложен Кантором и Зассенхаузом в 1981.

3 Реализация алгоритма

3.1 Выбор языка и используемые библиотеки

Для реализации алгоритма был выбран язык программирования C++ из-за скорости работы.

Для работы с многочленами над конечным полем была использована библиотека Givaro, поддерживающая необходимые тривиальные операции над многочленами: арифметические операции, дифференцирование, поиск наибольшего общего делителя.

3.2 Архитектура программы

Программа представляет собой реализацию класса CantorZassenhaus с единственным публичным методом run(PolynomialRing Fx, Polynomial f), который принимает кольцо полиномов над некоторым конечным полем и полином в этом поле, а возвращает искомую факторизацию.

Для реализации алгоритма Кантора-Зассенхауза были так же реализованы приватные методы:

а) rooting(PolynomialRing, Polynomial) — возведение многочлена с нулевой производной в степень $\frac{1}{p}$, где p — характеристика поля.

б) divide_by_lc(PolynomialRing, Polynomial) — деление многочлена на его старший коэффициент.

в) square_free(PolynomialRing, Polynomial) — удаление из многочлена кратных неприводимых сомножителей.

г) compute_distinct_degree(PolynomialRing, Polynomial) — разложение многочлена без кратных неприводимых сомножителей на произведение многочленов, все неприводимые сомножители которых имеют одинаковую степень.

д) factorize_equal_degree(PolynomialRing, Polynomial) — факторизация многочлена, все неприводимые множители которого имеют одинаковую степень.

```
class CantorZassenhaus {
 public:
    typedef Givaro::GFqDom<int> GaloisField;
    typedef Givaro::Poly1Dom<GaloisField, Givaro::Dense>::Element Polynomial;
    typedef Givaro::Poly1Dom<GaloisField, Givaro::Dense> PolynomialRing;

    static std::vector<std::pair<Polynomial, int>> run(
            const PolynomialRing &Fx,
            const Polynomial &f);

 private:
    CantorZassenhaus() = default;

    static Polynomial rooting(
            const PolynomialRing &Fx,
            const Polynomial &f);

    static Polynomial divide_by_lc(
            const PolynomialRing &Fx,
            const Polynomial &f);

    static Polynomial square_free(
            const PolynomialRing &Fx,
            const Polynomial &f);

    static std::vector<std::pair<Polynomial, int>>
    compute_distinct_degree(
            const PolynomialRing &Fx,
            const Polynomial &f);

    static std::vector<Polynomial> factorize_equal_degree(
            const PolynomialRing &Fx,
            const Polynomial &f, int degree);
};
```

Рисунок 3.1 — Архитектура программы

4 Тестирование и анализ производительности

Для тестирования программы был написан класс `Test`, запускающий тесты над случайными полиномами в данном кольце полиномов с помощью метода `run_random_test`.

```
class Test {
public:
    static void run_random_tests(const CantorZassenhaus::PolynomialRing &Fx);

    static long measure(const CantorZassenhaus::PolynomialRing &Fx,
                        int degree);

private:
    Test() = default;

    static bool run_random_test(const CantorZassenhaus::PolynomialRing &Fx,
                                int degree);

    static std::vector<std::pair<CantorZassenhaus::Polynomial, int>>
    run_random(const CantorZassenhaus::PolynomialRing &Fx, int degree);

    static const std::vector<int> kDegrees;

    static bool test(const CantorZassenhaus::PolynomialRing &Fx,
                     CantorZassenhaus::Polynomial &f);
};
```

Рисунок 4.1 — Класс тестов

Метод `measure` возвращает время(в наносекундах) факторизации одного случайного многочлена в данном кольце полиномов `Fx` и данной степени `d`.

Для оценки производительности метод `measure` вызывался 10 раз для каждых фиксированных порядка конечного поля и степени многочлена, итоговое время усреднялось. Из графиков(4.2,4.3) видно, что время работы растет достаточно быстро.

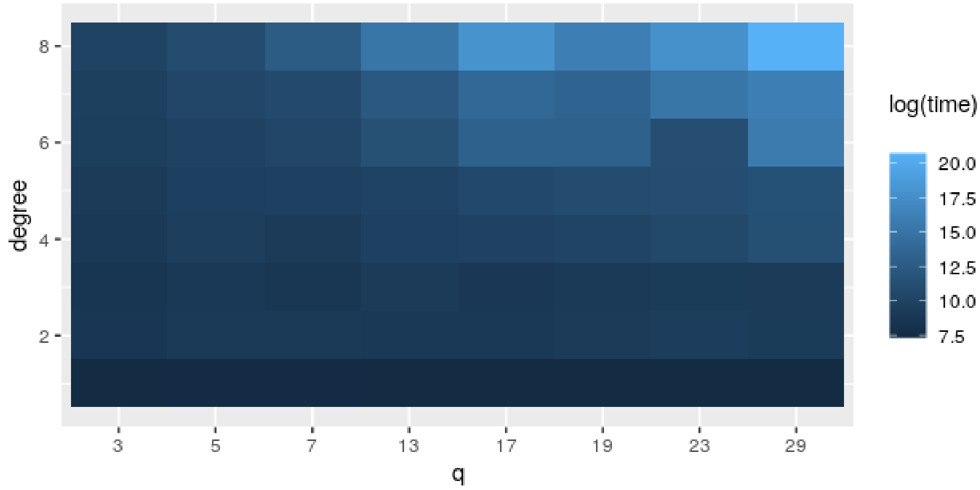

Рисунок 4.2

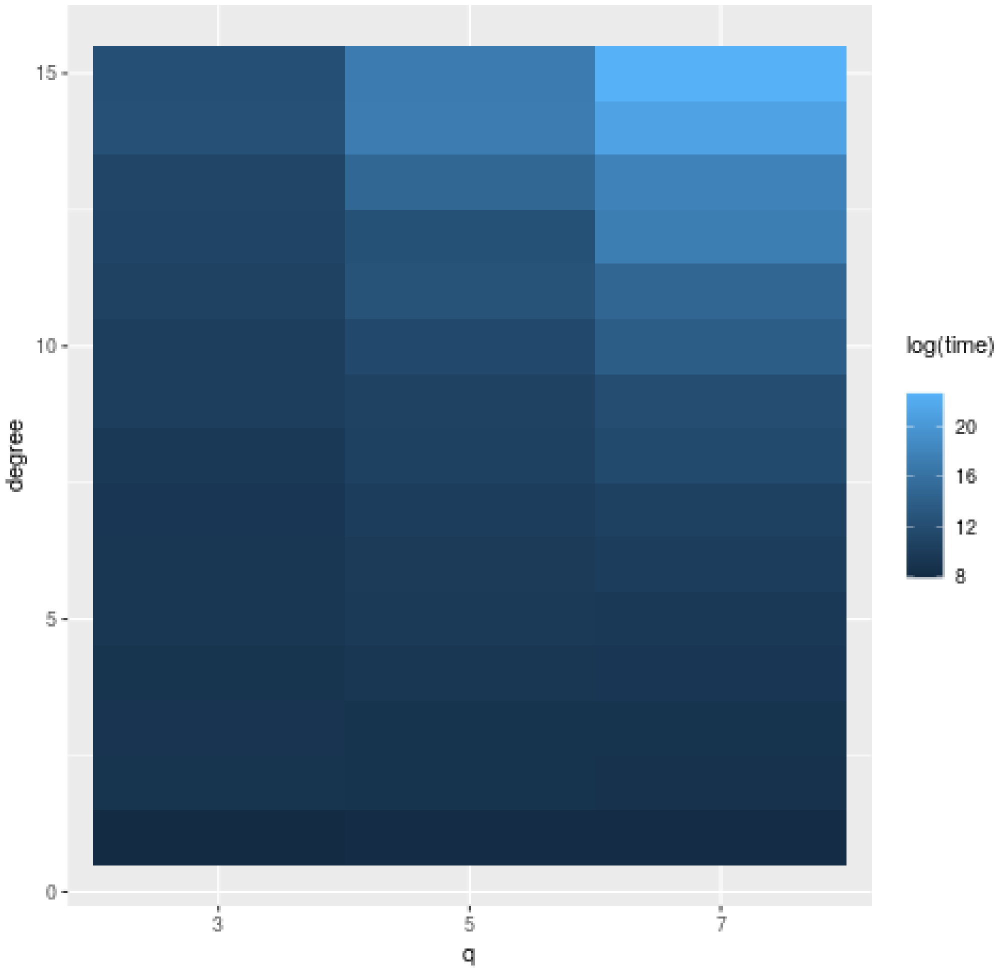

Рисунок 4.3

Список использованных источников

1. Лидл Р., Нидеррайтер Г., *Конечные поля: том 1*, Мир, 1988: 188

2. Cantor, David G., Zassenhaus, Hans, *A new algorithm for factoring polynomials over finite fields*, Mathematics of Computation, 36 (154): 587–592

3. Joachim von zur Gathen, Jürgen Gerhard, *Modern Computer Algebra*, Cambridge University Press, 2013: 377-392