УДК 519.688

ВЫЧИСЛЕНИЕ КРАТНЫХ СОМНОЖИТЕЛЕЙ ПОЛИНОМА МНОГИХ ПЕРЕМЕННЫХ

© Д.С. Ивашов

Ключевые слова: вычисление кратных сомножителей; алгоритм Бейкса, алгоритм Муссера; алгоритм Юна.
Исследуются известные алгоритмы вычисления кратных сомножителей многочлена. В таблицах представлены результаты экспериментов, в которых сравнивалось быстродействие процедур реализующих алгоритмы вычисления кратных сомножителей: стандартный алгоритм, алгоритм Бейкса, алгоритм Муссера и алгоритм Юна.

1 Введение

Факторизация многочленов многих переменных является одной из важных математических задач, имеющих применения в различных областях науки и техники.

Задача факторизации многочленов многих переменных была решена более 130 лет назад в работах Кронекера. Данный метод является обобщением методов Ньютона и метода, предложенного Фридрихом Шубертом (1793). Для многочленов с целыми коэффициентами появились новые более быстрые алгоритмы.

Первым алгоритмом факторизации *многочлена одной переменной* за полиномиальное время был алгоритм Берликемпа (1970). Алгоритм Берликемпа требует выполнения $O(n^w + n^{1+o(1)} log\ q)$ арифметических операций в F_q [1]. В 1981 г. Кантор и Цассенхауз предложили метод факторизации многочленов одной переменной требующий выполнения $O(n^{2+o(1)} log\ q)$ операций в F_q. Этот метод включает в себя использование трех алгоритмов: вначале выполняется алгоритм выделения кратных сомножителей (square-free factorization), затем к каждому из полученных сомножителей применяется алгоритм выделения сомножителей с различными степенями при старшей переменной (distinct-degree factorization), в завершение к каждому сомножителю, полученному на предыдущем этапе, применяется третий алгоритм, который позволяет выделить сомножетели с одинаковыми степенями при старшей переменной (equal-degree factorization) [2].

Нидеррайтер разработал новый подход к факторизации полиномов одной переменной над конечными полями со сложностью, близкой к сложности алгоритма Берликемпа [3–5]. Алгоритм Калтофена и Шоупа находит множители многочлена одной переменной в конечном поле за субквадратическое время. Этот метод опирается на быструю матричную арифметику и является одним из самых быстрых алгоритмов для больших конечных полей [6].

Что касается алгоритмов факторизации *многочленов многих переменных*, отметим следующие работы.

Калтофен и Трагер разработали алгоритм факторизации многочленов многих переменных, использующий black box представление [7].

Бернардин разработал схему факторизации многочленов многих переменных над конечными полями, состоящую из двух этапов. На первом этапе он представил новый алгоритм выделения кратных сомножителей многочлена многих переменных, при этом он опирался на алгоритм Юна. На втором этапе происходит вычисление сомножителей свободных от квадратов [8]. Эта схема факторизации была применена в системе компьютерной алгебры Maple.

Рассмотрим схему факторизации полиномов многих переменных $F(x_1, \ldots, x_n)$ над рациональными числами, состоящую из трех этапов, где в дополнение к схеме, предложенной Бернардином, мы добавили первый этап. Введение нового этапа позволяет ускорить вычисление сомножителей полинома в случае, если исходный полином содержал сомножители имеющие различные наборы переменных.

На первом этапе вычислим сомножители полинома, имеющие различные наборы переменных.

У полинома $F(x_1, \ldots, x_n)$ может быть не более 2^n сомножителей h_k, которые имеют различные наборы переменных. Для этого этапа можно воспользоваться алгоритмом описанном в [9].

В результате получим $F = \prod_{k=1}^{m} h_k$, где m — число сомножителей, имеющих различные наборы переменных.

На втором этапе вычислим сомножители, имеющие различную кратность.

Найдем множители, которые входят в полином в разных степенях: $h_k = \prod_{j=1}^{r_k} g_{kj}^{s_{kj}}$, где $s_{ki} \neq s_{kj}$ при $i \neq j$. Множители g_{kj} имеют такой же набор переменных, как и полином h_k. Процесс отделения таких сомножителей описан в параграфе 2. При этом, если g_{kj} имеет сомножители, то каждый из них входит в g_{kj} в первой степени.

На третьем этапе вычислим сомножители полинома, свободного от квадратов.

После первых двух этапов мы получим полиномы g_{kj}, которые не имеют кратных сомножителей и которые могут раскладываться только на взаимно простые множители. Для каждого из этих полиномов найдем множители в разложении $g_{kj} = \prod_{i=1}^{p_{kj}} f_{i_{kj}}$. На этом этапе найдем взаимно простые сомножители, используя алгоритм, описанный в [10].

В итоге получим $F(x_1, \ldots, x_n) = \prod_{k=1}^{m} h_k = \prod_{k=1}^{m} \prod_{j=1}^{r_k} g_{kj}^{s_{kj}} = \prod_{k=1}^{m} \prod_{j=1}^{r_k} \prod_{i=1}^{p_{kj}} f_{i_{kj}}^{s_{kj}}$.

2 Выделение сомножителей, имеющих различную кратность

Цель данной работы состоит в том, чтобы сравнить быстродействие четырех известных алгоритмов вычисления кратных сомножителей и дать рекомендации по их применению. Рассмотрим четыре алгоритма выделения кратных сомножителей полинома $F(x_1, \ldots, x_n)$, а именно, стандартный алгоритм [11], алгоритм Бейкса [11], алгоритм Муссера [12], алгоритм Юна [13].

2.1 Стандартный алгоритм выделения кратных сомножителей

Рассмотрим стандартный алгоритм выделения кратных сомножителей, который широко используется на практике [11]. Данный алгоритм хорошо подходит для разложения полиномов с большим числом множителей, которые находятся в относительно малых степенях.

На практике этот алгоритм предпочтительней, т. к. вероятность того, что произвольно выбранный полином имеет множители высокой кратности, приближается к нулю.

В данном алгоритме используются следующие функции.

GCD(f,g) — НОД полиномов f и g.

Number_Variable(f) — номер переменной, по которой идет дифференцирование.

D(f,i) — производная полинома f по переменной x_i.

Рассмотрим алгоритм выделения кратных сомножителей в полиноме многих переменных.

Алгоритм

Algorithm Standart_Square_Free (f, R)
Input: $f \in R, R = \mathbb{Z}[x_1, \ldots, x_n]$
Output: $result[\,] = f_i^i$
$index :=$ **Number_Variable** (f);
$number := 0$; $u :=$ **D** $(f, index)$; $a :=$ **GCD** (f, u);
If $(a = 1)\{$
Comment: Если НОД$(f, u) = 1$, то полином f не имеет кратных сомножителей.
$result[number] := f$;
$\}else\{$ $a_1 := \dfrac{f}{a}$;
While $(\, a_1 \neq 1\,)\{$
$g :=$ **GCD** (a_1, a); $b := \dfrac{a_1}{g}$; $result[number] := b$;
$number := number + 1$; $a_1 := g$; $a := \dfrac{a}{g}$; $\}\,\}$
return result;

$result[\,]$ будет содержать множители f_i кратности i, где i — номер элемента в массиве.

Пример.

Пусть дан полином $f(x_1, \cdots, x_n) = f_1(x_1, \cdots, x_n)^3 \cdot f_2(x_1, \cdots, x_n) \cdot f_3(x_1, \cdots, x_n)^2$, f_1, f_2, f_3 —неприводимые сомножители. В качестве старшей переменной возьмем x_1 и будем вычислять производные полиномов по этой переменной.

$a = \text{НОД}\,(f, f') = f_1^2 \cdot f_3\,; \qquad a_1 = \dfrac{f_1^3 \cdot f_2 \cdot f_3^2}{f_1^2 \cdot f_3} = f_1 \cdot f_2 \cdot f_3,$

$g = \text{НОД}\,(f_1^2 \cdot f_3, f_1 \cdot f_2 \cdot f_3) = f_1 \cdot f_3.$

Найдем первый сомножитель b_1 полинома f кратности 1:

$b_1 = \dfrac{f_1 \cdot f_2 \cdot f_3}{f_1 \cdot f_3} = f_2\,; \qquad a = \dfrac{f_1^2 \cdot f_3}{f_1 \cdot f_3} = f_1\,; \qquad g = \text{НОД}\,(f_1 \cdot f_3, f_1) = f_1.$

Найдем второй сомножитель b_2 полинома f кратности 2:

$b_2 = \dfrac{f_1 \cdot f_3}{f_1} = f_3\,; \qquad a = \dfrac{f_1}{f_1} = 1\,; \qquad g = \text{НОД}\,(1, f_1) = 1.$

Найдем третий сомножитель b_3 полинома f кратности 3:

$b_3 = \dfrac{f_1}{1} = f_1.$

Получили множители b_1, b_2, b_3 кратности $1, 2, 3$. Таким образом, полином f можно представить в виде $f = f_1^3 \cdot f_2 \cdot f_3^2$.

2.2 Алгоритм Бейкса выделения кратных сомножителей

Рассмотрим алгоритм выделения кратных сомножителей полинома многих переменных, основанный на алгоритме Бейкса [11]. Данный алгоритм позволяет значительно ускорить описанный выше стандартный алгоритм в случае, когда полином имеет небольшое число сомножителей, каждый из которых входит в произведение в большой кратности. Идея заключается в последовательном дифференцировании следующим образом.

Алгоритм

Algorithm New_Square_Free (f, R)
Input: $f \in R, R = \mathbb{Z}[x_1, \ldots, x_n]$
Output: $factors\_f[\,] = f_i^i$
$index :=$ **Number_Variable** (f);
$number := 0$; $g := f$; $g\_derivative :=$ **D** $(g, index)$;
While $(\, g\_derivative \neq 0\,)\{$ $f := g$;
$g :=$ **GCD** $(g\_derivative, g)$; $factors\_f[number] := \dfrac{f}{g}$;
$number := number + 1$; $g\_derivative :=$ **D** $(g, index)$;$\}$
While $(k \neq 0)$ **do** $\{$
if $(factors\_f[k] \neq 1)\{$
for $(i = 1\,; i < k - 1\,; i = i + 1)$ **do** $factors\_f[i]; = \dfrac{factors\_f[i]}{factors\_f[k]};$ $\}$
$k := k - 1;\}$
$factors\_f[\,]$;

Массив $factors\_f[\,]$ содержит множители $f_1, \cdots, f_k$ кратностей $1, \cdots, k$ полинома f.

Пример.

Пусть дан полином $f(x_1, \cdots, x_n) = f_1(x_1, \cdots, x_n)^3 \cdot f_2(x_1, \cdots, x_n)^5$ такой, что f_1, f_2 — неприводимые сомножители полинома f. В качестве старшей переменной возьмем x_1 и будем вычислять производные полиномов по этой переменной.

$f = f_1^3 \cdot f_2^5\,;$ $\quad g = f\,;$ $\quad g = \text{НОД}\,(g, g') = f_1^2 \cdot f_2^4.$

Найдем сомножитель полинома f, имеющий 1-ю степень.

$out[1] = \dfrac{f_1^3 \cdot f_2^5}{f_1^2 \cdot f_2^4} = f_1 \cdot f_2\,;$ $\quad f = f_1^2 \cdot f_2^4\,;$ $\quad g = \text{НОД}\,(g, g') = f_1 \cdot f_2^3.$

Найдем сомножитель полинома f, имеющий 2-ю степень.

$out[2] = \dfrac{f_1^2 \cdot f_2^4}{f_1 \cdot f_2^3} = f_1 \cdot f_2\,;$ $\quad f = f_1 \cdot f_2^3\;;$ $\quad g = \text{НОД}\,(g, g') = f_2^2.$

Найдем сомножитель полинома f, имеющий 3-ю степень.

$out[3] = \dfrac{f_1^1 \cdot f_2^3}{f_2^2} = f_1 \cdot f_2\,;$ $\quad f = f_2^2\;;\; g \quad = \text{НОД}\,(g, g') = f_2.$

Найдем сомножитель полинома f, имеющий 4-ю степень.

$out[4] = \dfrac{f_2^2}{f_2} = f_2\,;$ $\quad f = f_2\,;$ $\quad g = \text{НОД}\,(g, g') = 1.$

Найдем сомножитель полинома f, имеющий 5-ю степень.

$out[5] = \dfrac{f_2}{1} = f_2.$

Среди полиномов массива out выделим сомножители полинома f.

Множитель $out[5] = f_2$ входит в полином f с кратностью 5. Поделим остальные полиномы массива out на $out[5]$. Тогда массив out примет вид:

$out[1] = f_1, out[2] = f_1, out[3] = f_1, out[4] = 1, out[5] = f_2$.

Множитель $out[3] = f_1$ входит в полином f с кратностью 3. Поделим остальные полиномы массива out на $out[3]$. Тогда массив out примет вид:

$out[1] = 1, out[2] = 1, out[3] = f_1, out[4] = 1, out[5] = f_2$.

В итоге получим, что полином $f = out[5]^5 \cdot out[3]^3 = f_1^3 \cdot f_2^5$.

2.3 Алгоритм Муссера

Рассмотрим алгоритм Муссера, позволяющий вычислить кратные сомножители полинома $F(x_1, ..., x_n)$ [12].

Алгоритм

Algorithm Musser (f, R)
Input: $f \in R, R = \mathbb{Z}[x_1, ..., x_n]$
Output: $result\_factors[\,] = f_j, result\_powers[\,] = s_j$
$index :=$ **Number_Variable** (f);
$a := D(f, index)$; $c :=$ **GCD** (f, a); $w := \dfrac{f}{c}$; $k := 1$;
while $(c \neq 1)$ **do**{ $g :=$ **GCD** (w, c); $f_k := \dfrac{w}{g}$; $k := k + 1$; $w := g$; $c := \dfrac{c}{g}$; }
$f_k := w$;

Пример.

Пусть дан полином $f(x_1, \cdots, x_n) = f_1(x_1, \cdots, x_n)^3 \cdot f_2(x_1, \cdots, x_n) \cdot f_3(x_1, \cdots, x_n)^2$, f_1, f_2, f_3 —неприводимые сомножители. В качестве старшей переменной возьмем x_1 и будем вычислять производные полиномов по этой переменной.

$c = \text{НОД}(f, f') = f_1^2 \cdot f_3$; $\quad w = \dfrac{f}{c} = \dfrac{f_1^3 \cdot f_2 \cdot f_3^2}{f_1^2 \cdot f_3}$.

Найдем множитель кратности 1.

$g = \text{НОД}(w, c) = f_1 \cdot f_2$; $out\_[1] = \dfrac{w}{c} = \dfrac{f_1 \cdot f_2 \cdot f_3}{f_1 \cdot f_3} = f_2$; $w = f_1 \cdot f_3$; $c = \dfrac{c}{g} = \dfrac{f_1^2 \cdot f_3}{f_1 \cdot f_3} = f_1$.

Найдем множитель кратности 2.

$g = \text{НОД}(w, c) = f_1$; $\quad out\_[2] = \dfrac{w}{c} = \dfrac{f_1 \cdot f_3}{f_1} = f_3$; $\quad w = f_1$; $\quad c = \dfrac{c}{g} = \dfrac{f_1}{f_1} = 1$.

Получим множитель $out\_[3] = f_1$ кратности 3.

Получили множители $out\_[1], out\_[2], out\_[3]$ кратности $1, 2, 3$. Таким образом, полином f можно представить в виде $f = f_1^3 \cdot f_2 \cdot f_3^2$.

2.4 Алгоритм Юна

Рассмотрим алгоритм Юна выделения кратных сомножителей полинома [13].

Алгоритм

Algorithm Yun (f, R)
Input: $f \in R, R = \mathbb{Z}[x_1, ..., x_n]$
Output: $result\_factors[\,] = f_i, result\_powers[\,] = s_i$
$index :=$ **Number_Variable** (f);
$a :=$ **D** $(f, index)$; $c :=$ **GCD** (f, a); $w := \frac{f}{c}$; $v := \frac{a}{c}$;
$u := v -$ **D** $(w, index)$; $k := 1$; $j := 0$;
while $(u \neq 0)$ do{
$g :=$ **GCD** (w, u);
if $(g \neq 1)$ { $result\_factors[j] := g$; $result\_powers[j] := k$; $j := j + 1$; }
$k := k + 1$; $w := \frac{w}{g}$; $v := \frac{u}{g}$; $u := v -$ **D** $(w, index)$; }
$result\_factors[j] := w$; $result\_powers[j] := k$;
return result_factors, result_powers.

3 Вычислительные эксперименты

В системе компьютерной алгебры Mathpar имеются процедуры, реализующие алгоритмы вычисления кратных множителей: стандартный алгоритм, алгоритм Бейкса, алгоритм Муссера и алгоритм Юна. Эти процедуры реализованы на языке Java.

Были проведены серии экспериментов, в которых сравнивалось быстродействие этих процедур. В экспериментах использовался компьютер Intel(R) Core(TM)2 Duo T4200, 2.00GHz, 2 Гбайта ОЗУ, ОС Linux Ubuntu 11.04.

Ниже в табл. 1–5 мы используем следующие обозначения. Обозначим через S — стандартный алгоритм, BS — алгоритм Бейкса, M — алгоритм Муссера, Y — алгоритм Юна. В табл. 1–5 приведено время выполнения процедур выделения кратных сомножителей в секундах.

Эксперимент 1. В эксперименте использовался полином $F(x) = f(x)^{n_i}$, где $f(x)$ — линейный полином с 5-разрядными коэффициентами. В процессе эксперимента изменялись следующие параметры: n_i — кратность полинома $f(x)$, использовались значения 100, 200, 500, 1000, 1500, 2000.

Таблица 1

Время вычисления кратных сомножителей с использованием стандартного алгоритма, алгоритма Бейкса, алгоритма Муссера и алгоритма Юна

$F(x) = f(x)^{n_i}$				
n_i	S	BS	M	Y
100	0.5	0.48	0.5	0.115
200	1.3	1.26	1.29	0.17
500	2.8	1.8	2.95	0.38
1000	12.29	3.67	13.95	1.05
1500	38.6	8.39	43.8	2.15
2000	94.26	18.09	102	5.57

Алгоритм Юна быстрее остальных алгоритмов, причем при возрастании кратности n_i полинома $f(x)$ возрастает и разница. Например, при $n_i = 100$ алгоритм Юна быстрее

стандартного алгоритма в 4.3 раза, а при $n_i = 2000$ в 16.9 раза. Если сравнивать стандартный алгоритм и алгоритм Бейкса, то из табл. 1 видно, что при увеличении кратности алгоритм Бейкса быстрее при $n_i = 500$ в 1.5 раза, а при $n_i = 2000$ — в 5.2 раза.

Эксперимент 2. В эксперименте использовался полином $F(x) = f_1(x)^{n_i} f_2(x)^{n_j}$, где $f_1(x), f_2(x)$ — линейные полиномы с 5-разрядными коэффициентами. В процессе эксперимента изменялись следующие параметры: n_i, n_j — кратности сомножителей $f_1(x), f_2(x)$ соответственно, использовались значения: $n_i = \{100, 150, 200, 500, 1000\}$, $n_j = \{150, 200, 250, 700, 1500\}$.

Таблица 2

Время вычисления кратных сомножителей с использованием стандартного алгоритма, улучшенного стандартного алгоритма, алгоритма Муссера и алгоритма Юна

$F(x) = f_1(x)^{n_i} f_2(x)^{n_j}$				
n_i, n_j	S	BS	M	Y
100, 150	1.4	1.39	1.4	0.25
150, 200	1.9	1.66	1.8	0.34
200, 250	2.4	1.95	2.35	0.45
500, 700	21	7.6	20.8	1.97
1000, 1500	183	51	183	16

Из табл. 2 видно, что алгоритм Юна выигрывает у остальных алгоритмов, причем при $n_i = 100, n_j = 150$ выигрыш составляет 5.6 раза. При $n_i = 1000, n_j = 1500$ разница между алгоритмом Юна и алгоритмом Бейкса составляет 3.1 раза. Таким образом при увеличении кратности сомножителей постепенно разница между алгоритмом Бейкса и алгоритмом Юна уменьшается.

Эксперимент 3. В эксперименте использовался полином $F(x) = f_1(x)^{n_i} f_2(x)^{n_j} f_3(x)^{n_s}$, где $f_1(x), f_2(x), f_3(x)$ — линейные полиномы с 5-разрядными коэффициентами. В процессе эксперимента изменялись следующие параметры: n_i, n_j, n_s — кратности сомножителей $f_1(x), f_2(x), f_3(x)$ соответственно, использовались значения: $n_i = \{100, 400, 700\}$, $n_j = \{200, 500, 800\}$, $n_s = \{300, 600, 900\}$.

Таблица 3

Время вычисления кратных сомножителей с использованием стандартного алгоритма, алгоритма Бейкса, алгоритма Муссера и алгоритма Юна

$F(x) = f_1(x)^{n_i} f_2(x)^{n_j} f_3(x)^{n_s}$				
n_i, n_j, n_s	S	BS	M	Y
100, 200, 300	3.16	2.1	3.15	0.59
400, 500, 600	27.9	9	28.1	3.24
700, 800, 900	113	35.4	117	13.6

Из табл. 3 видно, что алгоритм Юна выигрывает у остальных алгоритмов. Например, при $n_i = 100, n_j = 200, n_s = 300$ алгоритм Юна быстрее улучшенного стандартного алгоритма в 3.6 раза, а при $n_i = 700, n_j = 800, n_s = 900$ — в 2.6 раза. Таким образом, разница между этими алгоритмами уменьшается при увеличении числа сомножителей и увеличения их кратностей.

Эксперимент 4. В эксперименте использовался полином $F(x) = \prod_{i=1}^{10} f_i(x)^{n_j}$, где $f_i(x)$ — линейные полиномы с 5-разрядными коэффициентами. В процессе эксперимента изменялись следующие параметры: n_j — кратность сомножителей $f_1(x), f_2(x), \cdots, f_{10}(x)$ соответственно, использовались значения: $n_j = \{20, 30, 50, 100\}$.

Таблица 4

Время вычисления кратных сомножителей с использованием стандартного алгоритма, алгоритма Бейкса, алгоритма Муссера и алгоритма Юна

$F(x) = \prod_{i=1}^{10} f_i(x)^{n_j}$				
n_i	S	BS	M	Y
20	0.67	0.89	0.65	0.25
30	1.03	1.2	1	0.4
50	1.35	1.9	1.35	0.45
100	3.78	9.6	3.71	0.65

Из табл. 4 видно, что алгоритм Юна выигрывает у остальных алгоритмов, причем при $n_j = 20$ выигрыш составляет 2.6 раза, а при $n_j = 100$ выигрыш составляет 5.7 раза. Таким образом, при увеличении кратности сомножителей разрыв по быстродействию между алгоритмом Юна и остальными алгоритмами увеличивается.

Эксперимент 5. В эксперименте использовался полином $F(x) = \prod_{i=1}^{N} f_i(x)^{s_i}$, где $f_i(x)$ — полином с 5-разрядными коэффициентами. В процессе эксперимента изменялись следующие параметры: $deg_x F(x) = 5$ — максимальная степень переменной x; N — число сомножителей многочлена; $bf = (s_1, \cdots, s_k)$ — кратности сомножителей, т.е. s_i — кратность i-ого сомножителя; $p1 = 100\ \%, p2 = 50\ \%$ — плотности многочлена.

Таблица 5

Время вычисления кратных сомножителей с использованием стандартного алгоритма, алгоритма Бейкса, алгоритма Муссера и алгоритма Юна

N	bf	Плотность 50 %				Плотность 100 %			
		S	BS	M	Y	S	BS	M	Y
1	16-32	0,25	0,32	0,22	0,055	0,3	0,55	0,32	0,09
	64-128	1,67	2,3	1,61	0,33	1,8	4,6	1,8	0,62
	512-1024	250	700	240	10,5	260	1380,5	225	19
3	16-32	1,5	3,3	1,4	1,37	1,9	5,65	1,8	1,4
3	64-128	82	1113	81	67	167	2188	188	126
5	16-32	23,2	110	23	22,5	101,2	381	97,2	96

Из табл. 5 видно, что алгоритм Юна выигрывает у остальных алгоритмов. Можно отметить, что стандартный алгоритм быстрее алгоритма Бейкса в случае, если полином не состоит из линейных сомножителей, причем чем больше $deg_x f_i$, тем больше разница во времени вычислений у этих двух алгоритмов.

4 Заключение

Эксперименты проводились с полиномами от одной переменной, при этом использовались сомножители большой кратности.

Проведенные эксперименты позволяют утверждать, что алгоритм Юна более эффективен, чем остальные исследуемые алгоритмы, причем он лучше остальных алгоритмов во всех рассмотренных случаях. Разница в быстродействии между алгоритмом Юна и остальными алгоритмами увеличивается с увеличением кратности сомножителей исходного полинома.

ЛИТЕРАТУРА

1. *Berlekamp E.R.* Factoring polynomials over large finite fields // Math. Comp. 1970. V. 24. P. 713-735.

2. *Cantor D.G., Zassenhaus H.* A new algorithm for factoring polynomials over finite fields // Mathematics of Computation. 1981. V. 36. P. 587-592.

3. *Niederreiter H.* A new efficient factorization algorithm for polynomials over small finite fields // Applic. Algebra Engin. Commun. Comput. 1993. V. 4. P. 81-87.

4. *Niederreiter H., Gottfert R.* Factorization of polynomials over finite fields and characteristic sequences // Journal of Symbolic Computation 16. 1993. P. 401-412.

5. *Niederreiter H.* Factorization of polynomialas and some linear-algebra problems over finite fields // Linear Algebra and its Applications. 1993. V. 192. P. 301-328.

6. *Kaltofen E., Shoup V.* Subquadratic-time factoring of polynomials over finite fields // In Proceedings of the 27th Annual ACM Symposium on the Theory of Computing. 1995. P. 398-406.

7. *Kaltofen E., Trager B.M.* Computing with polynomials given by black box for their evaluations: Greatest common divisors, factorization, separation of numerations and denominators. // Journal of Symbolic Computation 9. 1990. P. 300-320.

8. *Bernardin L.* Factoring multivariate polynomials over a finite field // Diss. 1999.

9. *Ивашов Д.С.* Факторизация полиномов многих переменных // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2011. Т. 16. Вып. 1. С. 133-137.

10. *Малашонок Г.И., Ивашов Д.С.* Об алгоритме факторизации полиномов многих переменных // Вестник Тамбовского университета. Серия Естественные и технические науки. Тамбов, 2010. Т. 15. Вып. 1. С. 331-334.

11. *Backes M.* Factorization of Univariate Polynomials // Dissertation. Universitat des Saarlandes, 2002.

12. *Musser D.R.* Algorithms for Polynomial Factorization // PhD thesis, University of Wisconsin. 1971.

13. *Yun D.Y.Y.* On square-free decomposition algorithms // In SYMSAC '76:Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation. 1976. P. 26-35.

БЛАГОДАРНОСТИ: Работа выполнена при поддержке гранта РФФИ № 12-07-00755-а.

Поступила в редакцию 20 декабря 2012 г.

Ivashov D.S. SQUARE-FREE FACTORIZATION POLINOMIALS OF MANY VARIABLES.

We discuss known algorithms for square-free factorization of polynomials of many variables.

Key words: square-free factorization, standart square-free, Backes square-free, Musser, Yun.