Ссылка на статью:
// Ученые записки УлГУ. Сер. Математика и информационные технологии. 2022, № 2, с. 1-14.

Поступила: 27.11.2022
Окончательный вариант: 01.12.2022

УДК 519.7

Реализация алгоритмов построения конечных полей для помехоустойчивых кодов

Булдаковский П. А.

pbuldakovskii@mail.ru
УлГУ, Ульяновск, Россия

В работе проводится исследование различных методов построения конечных полей в поле Галуа GF (2^8), а также представлен сравнительный анализ скорости работы соответствующих алгоритмов. Реализация данных методов позволит в дальнейшем реализовать эффективный алгоритм декодирования Гао, позволяющий находить и исправлять ошибки, возникающие при передаче сообщений.

Ключевые слова: *коды Рида-Соломона, поле Галуа GF (2^8).*

Введение

В настоящее время обмен информацией происходит практически во всех областях человеческой деятельности при помощи различных каналов передачи информации. Однако какими бы совершенными не были технологии, информация подвергается искажению в процессе передачи. Решением данном проблемы стали помехоустойчивые коды. Самыми популярными кодами среди всех остальных являются коды РС.

Коды РС были открыты в 1960 году американскими учёными Ридом и Соломоном как частный случай недвоичных кодов БЧХ. В настоящее время коды РС являются одним из наиболее востребованных видов помехоустойчивого кодирования. Это связано с тем, что данные коды эффективно справляются с исправлением группирующихся ошибок, которые возникают при воздействии на канал связи импульсных помех. Во многих системах передачи и хранения информации ошибки группируются, поэтому коды РС над большими алфавитами могут оказаться весьма практичными. Подробнее с РС кодами можно познакомиться в [2, 3].

В кодах РС часто используется 2 вида арифметических операций: сложение и умножение. Однако данные действия выполняются над конечными полями, что не позволяет производить их по обычным правилам сложения и умножения. Также возникает потреб-

ность в скорости, т.к. обычно размер передаваемой информации очень большой. Именно поэтому необходимо использовать эффективный алгоритм сложения и умножения, чтобы максимально минимизировать время, которое затрачивается на выполнение данных арифметических операций.

1. Теоретические аспекты конечных полей

1.1. Понятие поля

Для начала введём понятие кольца [7].

Кольцом называется множество $\mathbb{R}$ с двумя бинарными операциями + и *, если:

1. $(R, +)$ является аддитивной абелевой группой с нулевым элементом 0;
2. Операции + и * связаны законом дистрибутивности:

$$(a + b) * c = a * c + b * c, a * (b + c) = a * b + a * c,\ a, b, c \in \mathbb{R}.$$

Кольцо $\mathbb{R}$ называется **ассоциативным**, если операция * обладает свойством ассоциативности: $(a * b) * c = a * (b * c),\ a, b, c \in \mathbb{R}$.

Кольцо $\mathbb{R}$ называется **коммутативным**, если операция * обладает свойством коммутативности: $a * b = b * a,\ a, b \in \mathbb{R}$.

Кольцо $\mathbb{R}$ называется **с единицей 1**, если 1 является единичным элементом относительно операции умножения: $a * 1 = 1 * a = a\ \forall\ a \in \mathbb{R}$.

Полем называется ассоциативное коммутативное кольцо с единицей $1 \neq 0$, в котором каждый ненулевой элемент обратим (по умножению). Другими словами, поле — непустое множество F с двумя бинарными операциями сложения и умножения, относительно которых выполнены следующие условия:

1. $(F, +)$ — аддитивная абелева группа с нулевым элементом 0;
2. $(F \setminus \{0\},\ *)$ — мультипликативная абелева группа с единичным элементом $1 \neq 0$;
3. $\forall\ a, b, c \in F$ выполнен закон дистрибутивности $(a + b) * c = a * c + b * c$, который связывает операции сложения и умножения.

Таким образом, поле F представляет собой гибрид двух абелевых групп (аддитивной $(F, +)$ и мультипликативной $(F \setminus \{0\},\ *)$, связанных законом дистрибутивности. Отметим следующие свойства поля:

1. В поле нет делителей нуля, так как если $a * b = 0$ и $a \neq 0$, то $a^{-1} * (a * b) = b = 0$;
2. В поле F множество всех ненулевых элементов образует мультипликативную абелеву группу: $F^* = F \setminus \{0\}$ (мультипликативная группа поля). Из этого следует, что в поле существует единственная единица и для любого ненулевого элемента существует единственный обратный по умножению элемент.

1.2. Поле Галуа GF (2^8)

Поле Галуа (далее GF) – поле, содержащее конечное множество элементов, сгенерированных с помощью примитивного элемента α. Данные элементы выглядят следующим

образом: $0,\ \alpha^1,\ \alpha^2, \dots, \alpha^{N-1}$, где $N = \alpha^m - 1$. В данном случае $m = 8$. Таким образом, мы сможем получить $GF(2^8)$.

Примитивный элемент – элемент g конечного поля $GF(q)$ порядка $q = p^n$, если порядок элемента g равен $q - 1$, т.е. $GF(q) = \{g,\ g^2, \dots,\ g^{q-1} = 1\}$.

Примитивный многочлен – минимальный многочлен примитивного элемента. Данный многочлен должен быть неприводимым.

Предложение 1. *Для любого* $\alpha \in GF(q^n)$ *существует минимальный многочлен.*

Доказательство следует из того, что расширение $GF(q) \subseteq GF(q^n)$ конечно, поэтому данное расширение алгебраично.

Для поля $GF(2^8)$ существует 30 неприводимых многочленов [9] (см. рис. 1).

Irreducible polynomial	Poly (dec)	Poly (hex)	Min primitive element	Elem (dec)
$x^8+x^4+x^3+x+1$	283	0x11B	$x+1$	3
$x^8+x^4+x^3+x^2+1$	285	0x11D	x	2
$x^8+x^5+x^3+x+1$	299	0x12B	x	2
$x^8+x^5+x^3+x^2+1$	301	0x12D	x	2
$x^8+x^5+x^4+x^3+1$	313	0x139	$x+1$	3
$x^8+x^5+x^4+x^3+x^2+x+1$	319	0x13F	$x+1$	3
$x^8+x^6+x^3+x^2+1$	333	0x14D	x	2
$x^8+x^6+x^4+x^3+x^2+x+1$	351	0x15F	x	2
$x^8+x^6+x^5+x+1$	355	0x163	x	2
$x^8+x^6+x^5+x^2+1$	357	0x165	x	2
$x^8+x^6+x^5+x^3+1$	361	0x169	x	2
$x^8+x^6+x^5+x^4+1$	369	0x171	x	2
$x^8+x^6+x^5+x^4+x^2+x+1$	375	0x177	$x+1$	3
$x^8+x^6+x^5+x^4+x^3+x+1$	379	0x17B	x^3+1	9
$x^8+x^7+x^2+x+1$	391	0x187	x	2
$x^8+x^7+x^3+x+1$	395	0x18B	x^2+x	6
$x^8+x^7+x^3+x^2+1$	397	0x18D	x	2
$x^8+x^7+x^4+x^3+x^2+x+1$	415	0x19F	$x+1$	3
$x^8+x^7+x^5+x+1$	419	0x1A3	$x+1$	3
$x^8+x^7+x^5+x^3+1$	425	0x1A9	x	2
$x^8+x^7+x^5+x^4+1$	433	0x1B1	x^2+x	6
$x^8+x^7+x^5+x^4+x^3+x^2+1$	445	0x1BD	x^2+x+1	7
$x^8+x^7+x^6+x+1$	451	0x1C3	x	2
$x^8+x^7+x^6+x^3+x^2+x+1$	463	0x1CF	x	2
$x^8+x^7+x^6+x^4+x^2+x+1$	471	0x1D7	x^2+x+1	7
$x^8+x^7+x^6+x^4+x^3+x^2+1$	477	0x1DD	x^2+x	6
$x^8+x^7+x^6+x^5+x^2+x+1$	487	0x1E7	x	2
$x^8+x^7+x^6+x^5+x^4+x+1$	499	0x1F3	x^2+x	6
$x^8+x^7+x^6+x^5+x^4+x^2+1$	501	0x1F5	x	2
$x^8+x^7+x^6+x^5+x^4+x^3+1$	505	0x1F9	$x+1$	3

Рис. 1. Таблица неприводимых многочленов над полем $GF(2^8)$

Для программной реализации мы возьмём неприводимый полином $x^8 + x^4 + x^3 + x^2 + 1$ (100011101_2 или 285_{10}). Примитивным элементом в данном случае будет выступать x (10_2 или 2_{10}).

Подробнее с конечными полями можно ознакомиться в [1, 4, 5, 6, 8].

2. Программная реализация построения таблиц сложения и умножения в поле Галуа GF (2^8)

2.1. Алгоритм сложения

Чтобы описать сложение в GF, будет удобно представить элемент данного поля в полиномиальном виде, который выглядит следующим образом:

$\alpha_{m-1} * x^{m-1} + \cdots + \alpha_2 * x^2 + \alpha_1 * x + \alpha_0$, где α_{m-1} ... α_0 принимают значение либо 0, либо 1.

Проще говоря, α_{m-1} ... α_0 – двоичное представление элемента в поле. Таким образом, сложение байт (элементов поля $GF\ (2^8)$) представляет собой поразрядное суммирование по модулю 2 — так называемое XOR-суммирование.

Для данной операции проводить сравнительный анализ не имеет смысла, т.к. XOR-суммирование является битовой операцией, что обуславливает быструю скорость её выполнения.

2.2. Алгоритмы умножения

Умножение в $GF\ (2^8)$ можно выполнить несколькими способами. В данном пункте мы рассмотрим основные алгоритмы умножения [10].

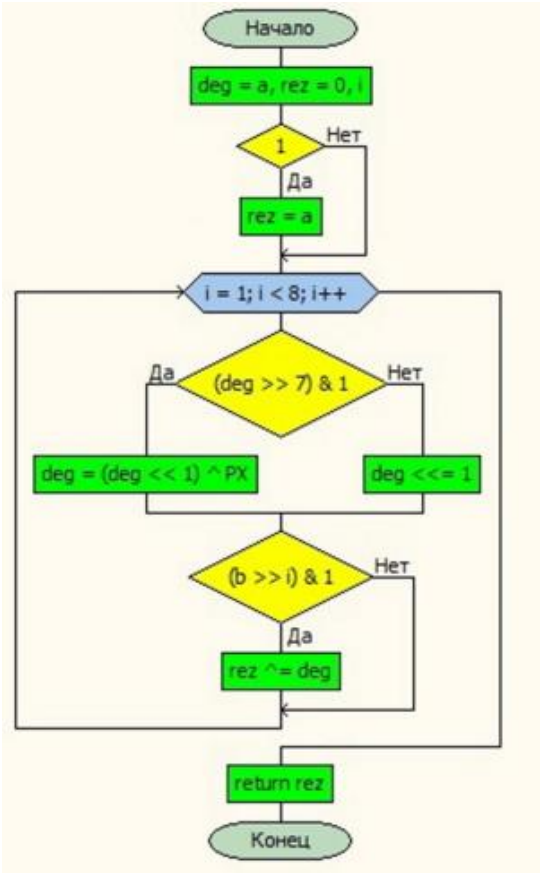

Рис. 2. Блок-схема алгоритма перебора значащих битов

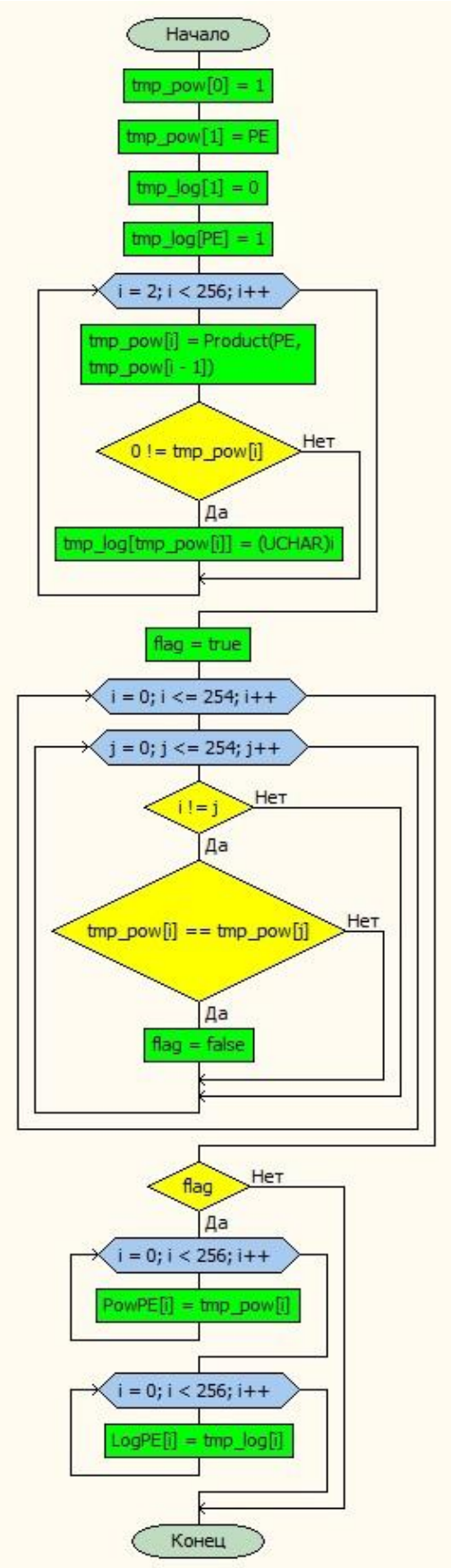

Рис. 3. Блок-схема алгоритма умножения на основе таблиц логарифмов и степеней примитивного члена

2.2.1. Алгоритм перебора значащих битов

В данном алгоритме умножение байт происходит с помощью представления их полиномами от α с коэффициентами из $\mathbb{Z}_2$ и перемножения их по обычным алгебраическим правилам. Полученное произведение необходимо привести по модулю многочлена p(x) (т.е. остаток от деления на p(x)). Блок-схема данного алгоритма представлена на рис. 2. Далее мы будем называть этот алгоритм «Product».

2.2.2. Алгоритм умножения на основе таблиц логарифмов и степеней примитивного члена

Данный способ основан на первоначальном построении таблиц степеней и логарифмов примитивного члена, после чего на основе этих таблиц происходит умножение элементов. Для примера, рассмотрим произведение 17 на 200. Данное умножение будет выглядеть следующим образом:

$$17 * 200 = 2^{\log_2(17)} * 2^{\log_2(200)} = 2^{100} * 2^{196} = 2^{296} = 2^{41} = 212$$

Блок-схема данного алгоритма представлена на рис. 3. Далее для удобства мы будем называть этот алгоритм «Multiplication».

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2)	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
3)	0	3	6	5	12	15	10	9	24	27	30	29	20	23	18	17	48
4)	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64
5)	0	5	10	15	20	17	30	27	40	45	34	39	60	57	54	51	80
6)	0	6	12	10	24	30	20	18	48	54	60	58	40	46	36	34	96
7)	0	7	14	9	28	27	18	21	56	63	54	49	36	35	42	45	112
8)	0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128
9)	0	9	18	27	36	45	54	63	72	65	90	83	108	101	126	119	144
10)	0	10	20	30	40	34	60	54	80	90	68	78	120	114	108	102	160
11)	0	11	22	29	44	39	58	49	88	83	78	69	116	127	98	105	176
12)	0	12	24	20	48	60	40	36	96	108	120	116	80	92	72	68	192
13)	0	13	26	23	52	57	46	35	104	101	114	127	92	81	70	75	208
14)	0	14	28	18	56	54	36	42	112	126	108	98	72	70	84	90	224
15)	0	15	30	17	60	51	34	45	120	119	102	105	68	75	90	85	240
16)	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	29
17)	0	17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	13
18)	0	18	36	54	72	90	108	126	144	130	180	166	216	202	252	238	61
19)	0	19	38	53	76	95	106	121	152	139	190	173	212	199	242	225	45
20)	0	20	40	60	80	68	120	108	160	180	136	156	240	228	216	204	93
21)	0	21	42	63	84	65	126	107	168	189	130	151	252	233	214	195	77
22)	0	22	44	58	88	78	116	98	176	166	156	138	232	254	196	210	125
23)	0	23	46	57	92	75	114	101	184	175	150	129	228	243	202	221	109
24)	0	24	48	40	96	120	80	72	192	216	240	232	160	184	144	136	157
25)	0	25	50	43	100	125	86	79	200	209	250	227	172	181	158	135	141
26)	0	26	52	46	104	114	92	70	208	202	228	254	184	162	140	150	189
27)	0	27	54	45	108	119	90	65	216	195	238	245	180	175	130	153	173
28)	0	28	56	36	112	108	72	84	224	252	216	196	144	140	168	180	221
29)	0	29	58	39	116	105	78	83	232	245	210	207	156	129	166	187	205
30)	0	30	60	34	120	102	68	90	240	238	204	210	136	150	180	170	253
31)	0	31	62	33	124	99	66	93	248	231	198	217	132	155	186	165	237
32)	0	32	64	96	128	160	192	224	29	61	93	125	157	189	221	253	58
33)	0	33	66	99	132	165	198	231	21	52	87	118	145	176	211	242	42
34)	0	34	68	102	136	170	204	238	13	47	73	107	133	167	193	227	26
35)	0	35	70	101	140	175	202	233	5	38	67	96	137	170	207	236	10
36)	0	36	72	108	144	180	216	252	61	25	117	81	173	137	229	193	122
37)	0	37	74	111	148	177	222	251	53	16	127	90	161	132	235	206	106
38)	0	38	76	106	152	190	212	242	45	11	97	71	181	147	249	223	90
39)	0	39	78	105	156	187	210	245	37	2	107	76	185	158	247	208	74
40)	0	40	80	120	160	136	240	216	93	117	13	37	253	213	173	133	186
41)	0	41	82	123	164	141	246	223	85	124	7	46	241	216	163	138	170
42)	0	42	84	126	168	130	252	214	77	103	25	51	229	207	177	155	154
43)	0	43	86	125	172	135	250	209	69	110	19	56	233	194	191	148	138
44)	0	44	88	116	176	156	232	196	125	81	37	9	205	225	149	185	250
45)	0	45	90	119	180	153	238	195	117	88	47	2	193	236	155	182	234
46)	0	46	92	114	184	150	228	202	109	67	49	31	213	251	137	167	218
47)	0	47	94	113	188	147	226	205	101	74	59	20	217	246	135	168	202
48)	0	48	96	80	192	240	160	144	157	173	253	205	93	109	61	13	39
49)	0	49	98	83	196	245	166	151	149	164	247	198	81	96	51	2	55
50)	0	50	100	86	200	250	172	158	141	191	233	219	69	119	33	19	7
51)	0	51	102	85	204	255	170	153	133	182	227	208	73	122	47	28	23
52)	0	52	104	92	208	228	184	140	189	137	213	225	109	89	5	49	103
53)	0	53	106	95	212	225	190	139	181	128	223	234	97	84	11	62	119
54)	0	54	108	90	216	238	180	130	173	155	193	247	117	67	25	47	71
55)	0	55	110	89	220	235	178	133	165	146	203	252	121	78	23	32	87
56)	0	56	112	72	224	216	144	168	221	229	173	149	61	5	77	117	167

Рис. 4. Часть заранее сгенерированной таблицы умножения

2.2.3. Обращение к заранее сгенерированной таблице

Данный метод основывается на том, чтобы построить в самом начале таблицу умножения с помощью одного из двух предыдущих алгоритмов. В дальнейшем можно будет просто обращаться к нужному элементу в матрице. Обращение по адресу в памяти будет гораздо быстрее любых вычислений. Для примера, умножим 49 на 15. Для этого мы просто обращаемся к элементу, который находится на 49 строчке 15 столбца (или на 15 строчке 49 столбца, т.к. умножение – коммутативная операция и данные изменения приведут к одному и тому же результату). На рис. 4 представлена часть таблицы, на которой можно наглядно увидеть результат умножения на нужной позиции.

Блок-схема данного алгоритма представлена на рис. 5. Далее для удобства мы будем называть этот алгоритм «Table».

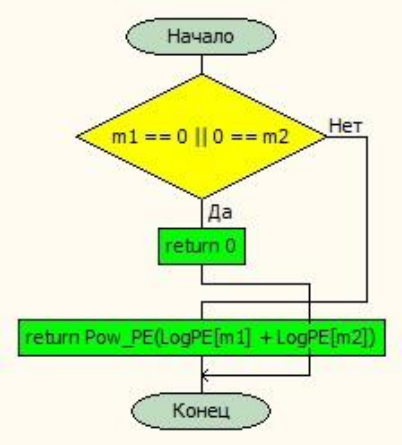

Рис. 5. Блок-схема алгоритма обращения к заранее сгенерированной таблице

2.3. Сравнительный анализ

Для начала следует сказать, какие способы оптимизации были применены к данным алгоритмам. В первую очередь, для обеспечения максимально быстрой скорости работы был выбран язык программирования C++. Далее к алгоритмам были применены следующие способы оптимизации:

- Алгоритмическая оптимизация;
- Особенности языка C++ (битовые операции);
- Оптимизация памяти, используемой для реализации алгоритма.

Скоростные тесты проводились на компьютере со следующими техническими требованиями:

- ОС: Microsoft Windows 10 x64;
- IDE: CLion 2021.3.4;
- CPU: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz;
- RAM: 16 Гб 1333 МГц;
- GPU: NVIDIA GeForce GTX 1050.

Измерение времени выполнения алгоритмов проводилось с помощью встроенной библиотеки языка C++ <chrono>.

Обозначим за N количество итераций цикла, через который прогонялись тестируемые алгоритмы. Скоростные тесты отображены в таблице 1.

Таблица 1. Скоростные тесты, проведённые на тестируемых алгоритмах

Алгоритм	N	Время выполнения, мс
Product	1	3,9406
	10	3,9714
	50	4,035
	100	4,05116
	500	4,099
	1000	4,17
	5000	4,241796
	10000	4,4
	50000	4,47
	100000	4,515609
Multiplication	1	0
	10	0,618848
	50	0,620381
	100	0,63099
	500	0,63286
	1000	0,6354
	5000	0,64
	10000	0,642
	50000	0,6452
	100000	0,655
Table (Product)	1	0
	10	0,11799
	50	0,118609
	100	0,120092
	500	0,12316
	1000	0,128
	5000	0,13
	10000	0,137
	50000	0,1428

	100000	0,1503
Table (Multiplication)	1	0
	10	0,119228
	50	0,119247
	100	0,12093
	500	0,1228
	1000	0,1235
	5000	0,12356
	10000	0,13
	50000	0,136
	100000	0,152

По следующим результатам для наглядности были построены графики. На рис. 6 представлены результаты всех 4 алгоритмов. На рис. 7 представлены все алгоритмы, кроме алгоритма «Product» (для лучшего рассмотрения результатов).

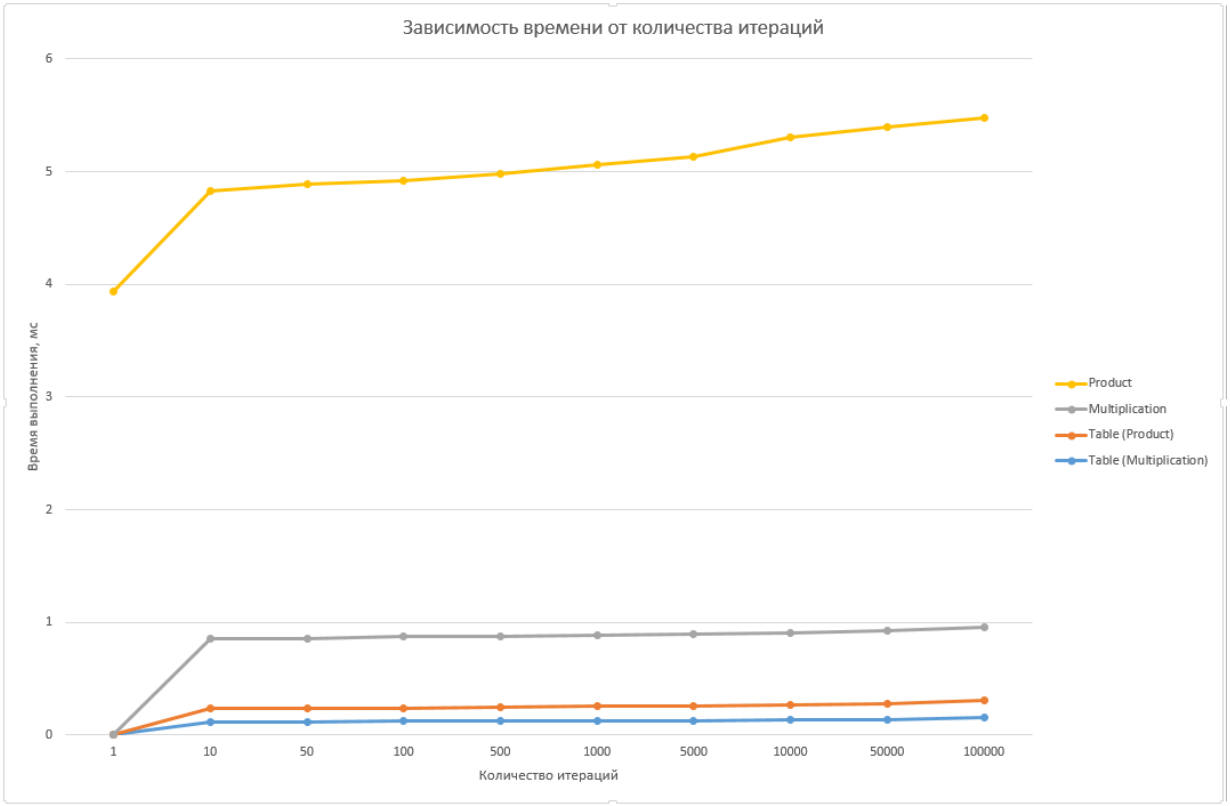

Рис. 6. Зависимость времени работы алгоритмов от количества итераций

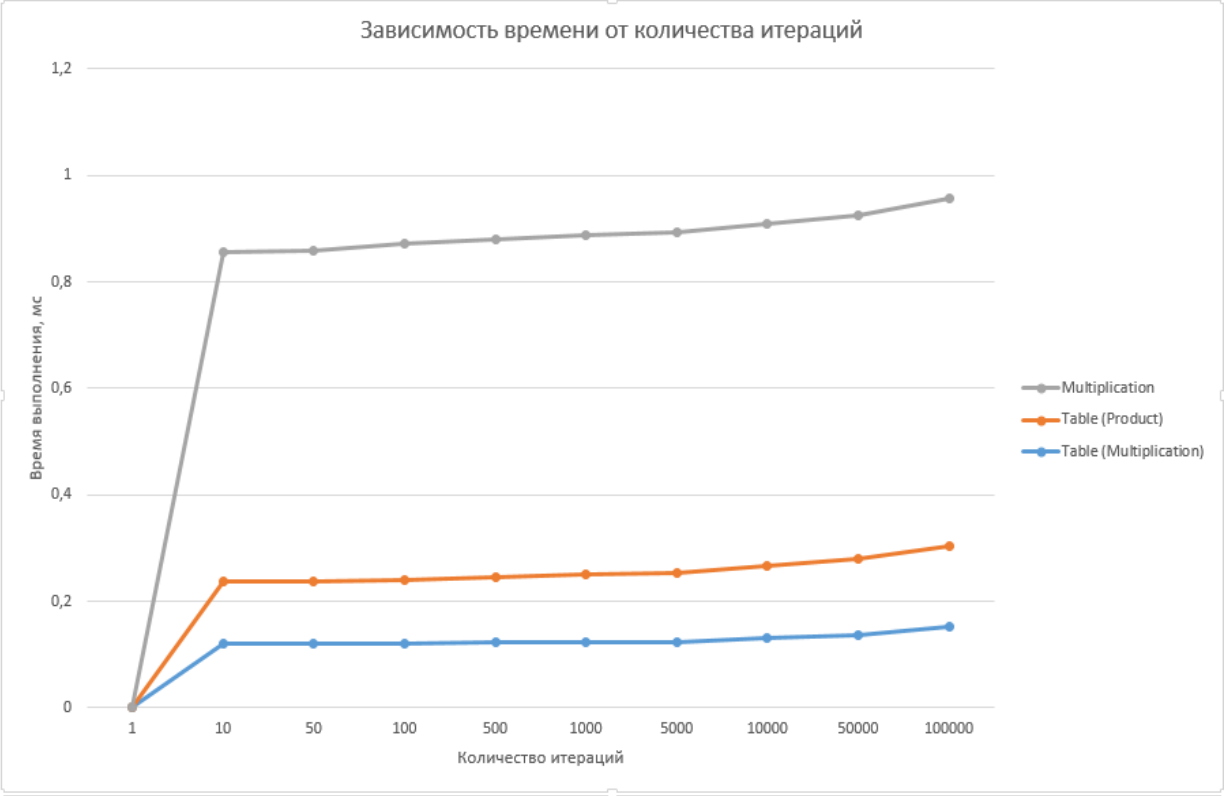

Рис. 7. Зависимость времени работы алгоритмов от количества итераций (без алгоритма «Product»)

Как видно из графиков, алгоритм Product очень сильно уступает прочим алгоритмам. В то же время, доступ к таблицам осуществляется гораздо быстрее, чем происходят вычисления умножения. Это довольно логичный результат, т.к. доступ по нужному адресу в памяти всегда выполняется быстрее любых вычислений.

Далее было решено посмотреть среднюю скорость построения таблиц. Для определённости будем строить 100 таблиц. Обозначим за N номер теста. Скоростные тесты отображены в таблице 2.

Таблица 2. Скоростные тесты, проведённые для алгоритмов Product и Multiplication

Алгоритм	N	Время выполнения, мс
Product	1	3,83
	2	3,88
	3	3,98
	4	4,05
	5	3,95
	6	3,89
	7	4,04
	8	3,93
	9	3,96

	10	4,09
	11	3,89
	12	3,91
	13	3,89
	14	3,95
	15	3,97
	16	4,08
	17	3,99
	18	4,19
	19	3,92
	20	3,84
	21	3,98
	22	4,11
	23	3,93
	24	3,9
	25	3,96
Multiplication	1	0,64
	2	0,68
	3	0,7
	4	0,68
	5	0,64
	6	0,71
	7	0,71
	8	0,72
	9	0,78
	10	0,7
	11	0,68
	12	0,67
	13	0,68
	14	0,67
	15	0,71
	16	0,66
	17	0,69
	18	0,68
	19	0,75

	20	0,74
	21	0,65
	22	0,63
	23	0,7
	24	0,63
	25	0,68

На рис. 8 можно увидеть визуальное представление полученных данных.

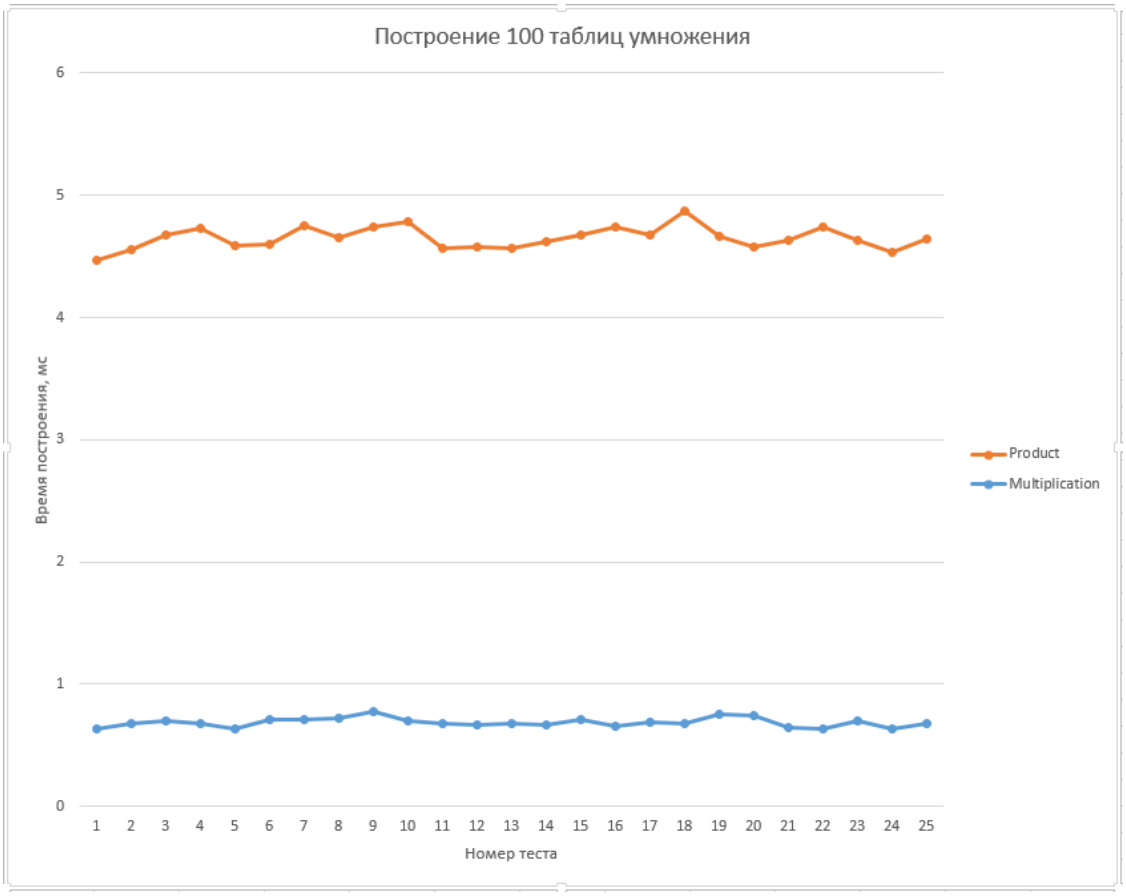

Рис. 8. Средняя скорость построения 100 таблиц умножения

Как видно из графика, алгоритм Product существенно проигрывает по скорости алгоритму Multiplication.

На основании данных результатов можно сделать следующие выводы:

- Алгоритм «Product» не требует никакой дополнительной памяти. Несмотря на то, что по скорости он значительно проиграл другим алгоритмам, для построения 1-ой таблицы умножения данный алгоритм всё равно подойдёт;
- Алгоритм «Multiplication» гораздо быстрее предыдущего алгоритма, но у него появляется потребность в выделении дополнительной памяти, т.к. таблицы степеней и логарифмов примитивного элемента необходимо где-то хранить;
- Алгоритм «Table» является самым быстрым среди двух предыдущих. Однако по

сравнению с алгоритмом «Multiplication» данный способ требует куда больше памяти для хранения таблицы, т.к. она представляет собой матрицу 256x256. Если рассматривать данный способ в перспективе, то он всё равно намного лучше предыдущих 2 алгоритмов, т.к. для многочисленных вычислений гораздо проще 1 раз посчитать таблицу и затем просто обращаться к ней. Поэтому данный способ однозначно выигрывает.

Заключение

В работе было исследовано три алгоритма умножения в поле Галуа $GF\ (2^8)$. Результаты проделанной работы позволили определить наилучший алгоритм для умножения элементов.

Данный алгоритм позволит реализовать в дальнейшем кодирование и декодирование кодов Рида-Соломона. В частности, он позволит в будущем реализовать библиотеку для декодирования кодов Рида-Соломона методом Гао. Данный декодер позволяет исправлять ошибки, возникшие при передаче сообщения, где как раз используется умножение элементов, и полученный в данной работе алгоритм позволит существенно увеличить скорость декодирования за счёт своей оптимизации.

Список литературы

1. Арнольд В.И. *Динамика, статистика и проективная геометрия полей Галуа.* М.: МЦНМО, 2005. 72 с.
2. Бояринов И.М. *Помехоустойчивое кодирование числовой информации.* М.: Наука, 1983. 196 с.
3. Васильев К. К., Глушков В. А., Дормидонтов А. В., Нестеренко А. Г. *Теория электрической связи.* Ульяновск: УлГТУ, 2008. 452 с.
4. Винберг Э. Б. *Курс алгебры (новое изд., перераб. и доп.)* М.: МЦНМО, 2011. 592 с.
5. Лидл Р., Нидеррайтер Г. *Конечные поля: в 2-х томах / пер. с англ.* М.: Мир, 1988. Т. 1. 430 с.
6. Рацеев С.М. *Математические методы защиты информации: учеб. пособие для вузов.* СПб.: Лань, 2022. 544 с.
7. Рацеев С.М. *Элементы высшей алгебры и теории кодирования: учебное пособие для вузов.* Санкт-Петербург: Лань, 2022. 656 с.
8. Журавлёв Ю. И., Флёров Ю. А., Вялый М. Н. *Дискретный анализ. Основы высшей алгебры*. изд. 2, испр. и доп. М.: МЗ Пресс, 2007. 224 с.
9. Planteen C. *Primitive elements and irreducible polynomials of GF (256)*. Режим доступа: https://codyplanteen.com/assets/rs/gf256_prim.pdf (дата обращения 10.05.2022).
10. Kerl J. *Computation in finite fields.* Режим доступа: https://johnkerl.org/doc/ffcomp.pdf (дата обращения 12.05.2022).

Finite field algorithms' implementation for error correcting coding

Buldakovskiy, P. A.

pbuldakovskii@mail.ru

Ulyanovsk State University, Ulyanovsk, Russia

In this work a study of various methods of finite fields' construction in the Galois field GF (2^8) is carried out, and a comparative analysis of the speed of operation of the corresponding algorithms is presented. Implementation of these methods allow to realize further the effective Gao decoding algorithm, allowing to find and correct the errors that occur during the transmission of messages.

Keywords: *Reed-Solomon Codes, Galois Field GF (2^8).*