УДК КОД

КОДЫ ХЭММИНГА НАД КОНЕЧНЫМИ ПОЛЯМИ GF(q): ТЕОРИЯ, ПОСТРОЕНИЕ И ПРОГРАММНОЕ ПРИЛОЖЕНИЕ

Ведерникова Элоиза Олеговна, студент, направление подготовки 02.03.02 Фундаментальная информатика и информационные технологии, Оренбургский государственный университет, Оренбург
e-mail: eloiza160604@gmail.com

Носов Виталий Валерьевич, кандидат физико-математических наук, доцент, доцент кафедры математики и цифровых технологий, Оренбургский государственный университет, Оренбург
e-mail: puncker1978@mail.ru

Аннотация *В работе исследуются q-арные коды Хэмминга: обобщение классических бинарных кодов Хэмминга на конечные поля GF(q). Приведена единая алгебраическая схема построения таких кодов, основанная на проективном пространстве PG(m−1, q) и выборе системы представителей одномерных подпространств поля. Показано, что параметры q-арных кодов Хэмминга имеют вид* $n = (q^m - 1)/(q - 1)$, $k = n - m$, $d = 3$.
Описан процесс синдромного декодирования, позволяющий определять как позицию, так и величину ошибки для произвольного q. Реализовано учебное программное приложение, выполняющее построение матриц, кодирование, внесение ошибок и декодирование. Представленный алгоритм и программная реализация обеспечивают наглядное изучение q-арных кодов Хэмминга в рамках дисциплины «Теория кодирования».

Ключевые слова: *теория кодирования, линейные коды, коды Хэмминга, конечные поля, GF(q), синдромное декодирование, проективное пространство.*
Для цитирования: *Ведерникова Э. О., Носов В. В. Коды Хэмминга над конечными полями GF(q): теория, построение и программное приложение // Оренбург – 2025 г.*

HAMMING CODES OVER FINITE FIELDS GF(q): THEORY, CONSTRUCTION, AND SOFTWARE APPLICATION

Vedernikova Eloisa Olegovna, Undergraduate Student, Program 02.03.02 *Fundamental Informatics and Information Technologies*, Orenburg State University, Orenburg
e-mail: eloiza160604@gmail.com

Nosov Vitaly Valerievich, Candidate in Physics and Mathematics, Associate Professor, Department of Mathematics and Digital Technologies, Orenburg State University, Orenburg
e-mail: puncker1978@mail.ru

***Abstract**: The paper examines q-ary Hamming codes: a generalization of classical binary Hamming codes to finite fields GF(q). A unified algebraic scheme for constructing such codes is given, based on the projective space PG(m−1, q) and the choice of a system of representatives of one-dimensional subspaces of the field. It is shown that the parameters of q-ary Hamming codes have the form* $n = (q^m - 1)/(q - 1)$, $k=n - m$, $d=3$.

The process of syndrome decoding is described, which makes it possible to determine both the position and the error value for an arbitrary q. An educational software application has been implemented that performs matrix construction, encoding, error correction, and decoding. The presented algorithm and software implementation provide a visual study of q-ary Hamming codes within the framework of the Coding Theory discipline.

***Keywords**: coding theory, linear codes, Hamming codes, finite fields, GF(q), syndrome decoding, projective space.*

***For citation:** Vedernikova E. O., Nosov V. V. Hamming Codes over Finite Fields GF(q): Theory, Construction, and Software Application // Orenburg – 2025.*

Введение

Теория кодирования является одним из ключевых направлений современной информатики и телекоммуникаций, обеспечивающих надёжную передачу данных в условиях шумовых и искажённых каналов связи. В практических системах – от Wi-Fi и мобильной связи до оптоволоконных линий — неизбежно возникают ошибки, и исправление этих ошибок возможно только при использовании специально построенных кодов, обладающих определёнными алгебраическими свойствами.

Одним из наиболее известных и исторически значимых классов являются коды Хэмминга, разработанные Ричардом Хэммингом в 1950-х годах для автоматического обнаружения и исправления одиночной ошибки. Классическая форма этих кодов построена над бинарным полем GF(2) и широко применяется в памяти компьютеров, хранилищах данных и коммуникационных протоколах.

Однако бинарные коды Хэмминга являются лишь частным случаем более общей конструкции. В рамках общей теории линейных кодов возможно построение q-арных кодов Хэмминга, работающих над произвольными конечными полями GF(q), где q – степень простого числа. Такие коды обладают теми же фундаментальными свойствами (минимальное расстояние 3, исправление одной ошибки), но позволяют гибко выбирать длину, алфавит и избыточность. При увеличении q возрастает длина кода при фиксированном числе проверочных символов, улучшается скорость кодирования, а структура кодов становится богаче. Несмотря на широкую известность бинарных кодов Хэмминга, их q-арные обобщения редко рассматриваются в учебной литературе в наглядной, пригодной для практического освоения форме.

Цель данной работы – изучить общую конструкцию q-арных кодов Хэмминга, построить конкретные примеры кодов над различными полями GF(q), сравнить их параметры, а также реализовать программное приложение, демонстрирующее процессы кодирования, моделирования ошибок и синдромного декодирования.

Теоретические основы q-арных кодов Хэмминга

1. Конечные поля GF(q)

Основой q-арных кодов Хэмминга является конечное поле GF(q).

Конечное поле — это алгебраическая структура, содержащая конечное число элементов и допускающая операции сложения, вычитания, умножения и деления (кроме деления на ноль).

Конечное поле GF(q) существует, когда $q = p^m$ – степень простого числа, а m – натуральное. В простейшем случае q = p и GF(p) представляет собой арифметику по модулю p.

Для линейных кодов важно:

– сложение и умножение определены над GF(q);

– все операции выполняются в этом поле;

– векторы из $GF(q)^n$ образуют линейное пространство размерности n.

Для построения кодов Хэмминга в рамках данной работы используются простые поля GF(p), т.е. поля вида GF(2), GF(3), GF(5), где арифметика выполняется по модулю p.

2. Линейные (n, k)-коды

Линейный код над GF(q) — это k-мерное подпространство пространства $GF(q)^n$.

Пусть q – множество символов, m – число проверочных уравнений. Тогда параметры кода Хэмминга имеют вид:

– $n = \frac{q^m - 1}{q - 1}$ – длина кодового слова;

– $k = n - m$ – размерность (длина информационного слова);

– $d = 3$ –минимальное расстояние Хэмминга между разными кодовыми словами.

Минимальное расстояние определяет корректирующую способность кода: код исправляет t ошибок, если выполняется $d \geq 2t + 1$. Для кодов Хэмминга d = 3, поэтому они исправляют одну ошибку.

3. Построение проверочной матрицы H

Столбцы H — это система представителей одномерных подпространств $GF(q)^m$

Алгоритм:

Генерируются все ненулевые векторы $GF(q)^m$.

Группируются по пропорциональности.

Из каждой группы выбирается один вектор-представитель.

Эти векторы образуют столбцы H.

Количество таких столбцов равно n.

Итоговое число столбцов совпадает с количеством одномерных подпространств: $n = \frac{q^m - 1}{q - 1}$

Для q = 2 этот алгоритм превращается в классическое построение, где перебираются все двоичные столбцы от 1 до $2^m - 1$

Например, для $GF(3)^2$ одномерные подпространства задаются векторами: (1,0), (0,1), (1,1), (1,2).

4. Порождающая матрица

Порождающая матрица G получается, как базис ядра пространства решений $Hc^T = 0$.

После приведения H к систематическому виду: $H = [A \mid I_m]$,

порождающая матрица имеет вид: $G = [I_k \mid -A^T]$.

5. Классические бинарные коды Хэмминга GF(2)

Бинарные коды Хэмминга построены над GF(2).

При m проверочных уравнениях параметры:

$n = 2^m - 1$

$k = n - m$

$d = 3$

Проверочная матрица состоит из всех ненулевых двоичных m-мерных векторов. Например, при m = 3, n = 7 получим:

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Это код (7,4,3).

6. Синдромное декодирование q-арных кодов Хэмминга

Перед тем как перейти к алгоритму декодирования, важно отметить, что q-арные коды Хэмминга, так же как и бинарные, основаны на проверочной матрице H. Именно она задаёт линейные зависимости между символами кодового слова. Любое отклонение принятого слова от допустимого кодового пространства фиксируется через синдром – результат умножения H на транспонированный принятый вектор. В q-арном случае синдром несёт более детальную информацию: он позволяет определить не только позицию ошибки, но и её величину в поле GF(q). Благодаря этому синдромное декодирование становится универсальным и полностью алгебраическим способом исправления одиночной ошибки.

Пусть передано кодовое слово c, принято r = c + e, где e – вектор ошибки.

Синдром: $s = Hr^T = He^T = e_j h_j$

Так как e содержит только один ненулевой элемент e_j, синдром равен:

$s = e_j\, h_j$, где h_j — j-й столбец H.

Декодирование сводится к:

1. Нахождению столбца h_j, пропорционального синдрому.
2. Определению $e_j \in GF(q)$, такого что $s = e_j\, h_j$.
3. Исправлению ошибки: $r_j \leftarrow r_j - e_j$.

В отличие от бинарного случая, ошибка может иметь любое ненулевое значение поля, что делает алгоритм более общим.

7. Примеры для различных полей

- GF(2), $m = 3$, $n = 7$, $k = 4$.

Проверочная матрица H состоит из всех ненулевых бинарных троек.

- GF(3), $m = 2$, $n = (9 - 1)/2 = 4$, $k = 2$

Столбцы H: (1,0), (0,1), (1,1), (1,2).

- GF(5), $m = 2$, $n = (25 - 1)/4 = 6$, $k = 4$

Столбцы H: (1,0), (0,1), (1,1), (1,2), (1,3), (1,4).

Эти примеры хорошо демонстрируют связь между размером поля и длиной кода.

8. Сравнительный анализ q-арных и бинарных кодов Хэмминга

Параметр	**q = 2 (бинарный)**	**q > 2 (q-арный)**
Длина n	$(2^m - 1)$	$((q^m - 1)/(q - 1))$
Алфавит	{0,1}	GF(q)
Ошибка	Всегда 1	Любой ненулевой элемент поля
Избыточность	Высокая	Уменьшается при росте q
Сложность вычислений	минимальная	выше (больше арифметика)

Вывод: рост q делает коды длиннее, гибче и более эффективными, хотя декодирование становится более насыщенным арифметикой поля.

Программная реализация q-арных кодов Хэмминга

В рамках работы была создана программная реализация [hhttps://github.com/Eloiza-V/HammingCode], выполненная на языке Python с использованием библиотеки galois, обеспечивающей поддержку арифметики в конечных полях GF(q). Реализация оформлена в виде класса HammingCode, который инкапсулирует все основные этапы работы с кодом: построение проверочной и порождающей матриц, кодирование, синдромное декодирование и восстановление исходного сообщения.

Класс параметризуется значениями q и m, где q определяет порядок конечного поля, а m – число проверочных символов. На основе этих параметров автоматически вычисляются основные характеристики кода: длина кодового слова $n = q^m - 1/q - 1$ и размерность информационного пространства $k = n - m$

Построение проверочной матрицы

Проверочная матрица H формируется как матрица размера m×n, столбцы которой соответствуют представителям одномерных подпространств пространства $GF(q)^m$. Для её построения перебираются все ненулевые векторы длины m над полем GF(q). Каждый такой вектор нормируется путём умножения на обратный элемент так, чтобы его первый ненулевой компонент стал равен единице.

Данная процедура позволяет исключить пропорциональные векторы и получить канонический набор столбцов проверочной матрицы, соответствующий классическому определению q-арного кода Хэмминга. В результате формируется проверочная матрица, корректно задающая линейный код с минимальным расстоянием d = 3.

Формирование порождающей матрицы

Порождающая матрица G строится как базис ядра линейного отображения, заданного проверочной матрицей H. Иными словами, матрица G формируется как базис пространства решений системы линейных уравнений Hx^T = 0.

Для этого проверочная матрица приводится к ступенчатому виду над полем GF(q) с использованием алгоритма Гаусса. По свободным столбцам полученной матрицы строятся базисные векторы пространства решений, которые затем объединяются в строки порождающей матрицы. Такой подход не требует приведения проверочной матрицы к систематическому виду и является универсальным для произвольных конечных полей.

Процедура кодирования

Кодирование информационного слова осуществляется путём умножения вектора сообщения длины k на порождающую матрицу G. В результате формируется кодовое слово длины n, принадлежащее линейному коду Хэмминга. Все операции выполняются в поле GF(q), что обеспечивает корректность кодирования для q-арных кодов.

Синдромное декодирование

Для декодирования принятых кодовых слов используется синдромный метод. В программной реализации применяется таблица синдромов, которая формируется на этапе инициализации кода. Для всех возможных одноошибочных векторов заранее вычисляются соответствующие синдромы вида $s = He^T$, после чего формируется отображение «синдром — вектор ошибки».

При декодировании по вычисленному синдрому мгновенно определяется позиция и величина одиночной ошибки, что позволяет исправить её без перебора столбцов проверочной матрицы. Такой подход обеспечивает высокую эффективность декодирования и полностью соответствует теории кодов Хэмминга.

Восстановление исходного сообщения

Поскольку порождающая матрица G в реализации не предполагается систематической, восстановление исходного информационного слова выполняется путём решения системы линейных уравнений $G^T m^T = c^T$ над полем GF(q), где c – исправленное кодовое слово. Решение данной системы позволяет корректно извлечь исходное сообщение независимо от структуры порождающей матрицы.

Разработанная программная реализация обеспечивает полный цикл работы с q-арными кодами Хэмминга: автоматическое построение матриц H и G, кодирование, моделирование одиночных ошибок, синдромное декодирование и восстановление исходного сообщения. Реализация может использоваться как в учебных целях для демонстрации принципов работы кодов Хэмминга, так и для экспериментальной проверки теоретических свойств линейных кодов над конечными полями.

Ниже приведен укрупненный фрагмент реализации класса HammingCode, отражающий ключевые идеи построения и декодирования q-арных кодов Хэмминга:

```
import numpy as np
import galois
from itertools import product

class HammingCode:
  def __init__(self, q, m):
    """
    Конструктор класса.

    Параметры:
    q — порядок конечного поля GF(q)
    m — число проверочных символов

    Инициализируются параметры кода (n, k), строятся матрицы H и G,
    формируется таблица синдромов для быстрого декодирования.
    """
    self.q = q
    self.m = m
```

```
        self.GF = galois.GF(q)

        self.n = (q**m - 1) // (q - 1)
        self.k = self.n - m

        self.H = self._build_H()
        self.G = self._build_G_from_kernel(self.H)
        self._build_syndrome_table()

    # ------------------------------------------------------------
    def _build_H(self):
        """
        Построение проверочной матрицы H размера m × n.

        - генерирует все ненулевые векторы GF(q)^m,
        - нормирует каждый вектор так, чтобы первый ненулевой элемент стал равен 1,
        - исключает пропорциональные векторы,
        - формирует столбцы H из представителей одномерных подпространств.

        Выход: H — матрица в поле GF(q)
        """
        GF = self.GF
        m = self.m

        reps = []
        seen = set()

        for vec in product(range(self.q), repeat=m):
            if all(v == 0 for v in vec):
                continue

            v = GF(list(vec))

            for i, a in enumerate(v):
                if a != 0:
                    v_norm = v * (GF(1) / a)
                    key = tuple(int(x) for x in v_norm)
                    break

            if key not in seen:
                seen.add(key)
                reps.append(key)

        reps.sort()
        return GF(np.array(reps).T)

    # ------------------------------------------------------------
    def _build_G_from_kernel(self, H):
        """
        Построение порождающей матрицы G как базиса ядра H.

        - выполняет приведение H к ступенчатому виду над GF(q),
        - определяет свободные столбцы,
        - формирует базисные векторы пространства решений Hx = 0,
```

```
    - собирает из них матрицу G размера k × n.

    Вход: H — проверочная матрица

    Выход: G — порождающая матрица
    """
    GF = self.GF
    H = GF(H.copy())

    m, n = H.shape
    A = H.copy()

    pivot_rows = []
    pivot_cols = []

    row = 0
    for col in range(n):
        pivot = None
        for r in range(row, m):
            if A[r, col] != 0:
                pivot = r
                break
        if pivot is None:
            continue

        if pivot != row:
            A[[row, pivot]] = A[[pivot, row]]

        inv = GF(1) / A[row, col]
        A[row] *= inv

        for r in range(m):
            if r != row and A[r, col] != 0:
                A[r] -= A[r, col] * A[row]

        pivot_rows.append(row)
        pivot_cols.append(col)
        row += 1
        if row == m:
            break

    free_cols = [j for j in range(n) if j not in pivot_cols]

    basis = []
    for free in free_cols:
        v = GF.Zeros(n)
        v[free] = GF(1)
        for pr, pc in zip(pivot_rows, pivot_cols):
            v[pc] = GF(0) - A[pr, free]
        basis.append(v)

    return GF(np.vstack(basis))

# ------------------------------------------------------------
```

```
def _build_syndrome_table(self):
    """
    Построение таблицы синдромов для быстрого декодирования.

      - перебирает все одноошибочные векторы e: ошибка в одной позиции, значение ∈ GF(q)\{0},
    - вычисляет их синдромы s = H * eᵀ,
    - сохраняет соответствие: синдром → вектор ошибки.

    Назначение: ускоряет декодирование: поиск ошибки становится операцией словаря.

    Выход: self.syndrome_table — словарь {синдром → ошибка}
    """
    GF = self.GF
    m, n = self.H.shape

    self.syndrome_table = {}

    zero = tuple(int(x) for x in GF.Zeros(m))
    self.syndrome_table[zero] = GF.Zeros(n)

    for pos in range(n):
        for a in range(1, self.q):
            e = GF.Zeros(n)
            e[pos] = GF(a)
            S = (self.H @ e.reshape(-1, 1)).reshape(-1)
            key = tuple(int(x) for x in S)
            if key not in self.syndrome_table:
                self.syndrome_table[key] = e

# ------------------------------------------------------------
def encode(self, msg):
    """
    Кодирование информационного слова.

    - преобразует информационный вектор u длины k в кодовое слово c длины n,
      используя матрицу G:  c = uG.

    Вход: msg — список или массив длины k над GF(q)

    Выход: кодовое слово длины n
    """
    GF = self.GF
    msg = GF(msg).reshape(-1)
    if msg.size != self.k:
        raise ValueError(f"Неверная длина сообщения, ожидается {self.k}")
    return msg @ self.G

# ------------------------------------------------------------
def decode(self, received):
    """
    Синдромное декодирование с исправлением одной ошибки.

    - вычисляет синдром s = H rᵀ,
```

```
    - по таблице синдромов определяет позицию и величину ошибки,
    - исправляет ошибку: r - e.

    Вход: received — принятый вектор длины n

    Выход: исправленный вектор длины n
    """
    GF = self.GF
    r = GF(received).reshape(-1)
    S = (self.H @ r.reshape(-1, 1)).reshape(-1)
    key = tuple(int(x) for x in S)

    e = self.syndrome_table.get(key)
    return r if e is None else r - e

# ------------------------------------------------------------
def decode_to_message(self, received):
    """
    Восстановление исходного сообщения.

    - сначала исправляет ошибки (decode),
    - решает систему линейных уравнений Gᵀ mᵀ = cᵀ,
    - выделяет информационную часть вектора.

    Вход: received — принятый вектор длины n

    Выход: информационное слово длины k
    """
    GF = self.GF
    c = self.decode(received).reshape(-1)

    G = self.G
    k, n = G.shape

    A = GF(G.T.copy())
    b = GF(c).reshape(-1, 1)

    Ab = GF(np.hstack([A, b]))

    rows, cols = Ab.shape
    num_vars = k

    r = 0
    pivot_cols = []

    for col in range(num_vars):
        pivot = None
        for i in range(r, rows):
            if Ab[i, col] != 0:
                pivot = i
                break
        if pivot is None:
            continue
```

```
    if pivot != r:
        Ab[[r, pivot]] = Ab[[pivot, r]]

    inv = GF(1) / Ab[r, col]
    Ab[r] *= inv

    for i in range(rows):
        if i != r and Ab[i, col] != 0:
            Ab[i] -= Ab[i, col] * Ab[r]

    pivot_cols.append(col)
    r += 1
    if r == rows:
        break

x = GF.Zeros(num_vars)
for row_idx, col_idx in enumerate(pivot_cols):
    x[col_idx] = Ab[row_idx, -1]

return x
```

Для проверки корректности реализации был проведен ряд экспериментов над различными полями GF(q)

1. Бинарный код Хэмминга [7,4,3] над GF(2)

При параметрах: q = 2, m = 3:

- длина кода n = 7
- размерность k = 4

Эксперимент:

1. Формируется случайное информационное слово длины 4.
2. Выполняется кодирование encode.
3. В одну из позиций кодового слова искусственно вносится ошибка (инверсия бита).
4. Вектор подается в decode.

```
GF(2), m=3
Сообщение:                [1, 0, 1, 1]
Кодовое слово:            [0 1 1 0 0 1 1]
Принято с ошибкой:        [0 1 1 1 0 1 1]
Исправленное слово:       [0 1 1 0 0 1 1]
Восстановленное сообщение: [1 0 1 1]
```

Рисунок 1 – Пример исправления одиночной ошибки в бинарном коде Хэмминга (7,4,3)

Результат: декодер корректно восстанавливает исходное кодовое слово и информационную часть. Это подтверждает правильность реализации классического бинарного кода Хэмминга.

2. Троичный код Хэмминга над GF(3)

При параметрах q = 3, m = 2:

- $n = (3^2 - 1) / (3 - 1) = 4$
- $k = 2$

Эксперимент аналогичен, но:

- элементы векторов принадлежат GF(3)
- ошибка вносится путем прибавления ненулевого элемента поля (например, +2) в одну позицию

```
GF(3), m=2
Сообщение:                   [1, 2]
Кодовое слово:               [1 0 1 2]
Принято с ошибкой:           [1 2 1 2]
Исправленное слово:          [1 0 1 2]
Восстановленное сообщение: [1 2]
```

Рисунок 2 – Работа синдромного декодирования в троичном коде Хэмминга (4,2,3)

Результат: алгоритм синдромного декодирования корректно определяет и позицию ошибки, и ее значение. Восстановленное слово совпадает с исходным.

3. Код над GF(5)

Аналогичные эксперименты проводятся для q = 5, m = 2:

- $n = (5^2 - 1) / (5 - 1) = 6$
- $k = 4$

```
GF(5), m=2
Сообщение:                   [3, 4, 1, 0]
Кодовое слово:               [1 2 3 4 1 0]
Принято с ошибкой:           [1 2 1 4 1 0]
Исправленное слово:          [1 2 3 4 1 0]
Восстановленное сообщение: [3 4 1 0]
```

Рисунок 3 – Исправление ошибок в q-арном коде Хэмминга над GF(5)

Результаты демонстрируют, что одна одиночная ошибка в любом символе GF(5) также полностью исправляется.

Проведенные вычислительные эксперименты подтверждают теоретические свойства q-арных кодов Хэмминга:

- для всех рассмотренных полей GF(q) реализованный алгоритм успешно исправляет одну ошибку
- структура проверочной матрицы H, построенной через систему представителей одномерных подпространств, обеспечивает минимальное расстояние $d = 3$
- при увеличении q растет длина кода и число возможных кодовых слов, что соответствует теоретическим выражениям $n = (q^m - 1)/(q - 1)$, $k = n - m$

Таким образом, разработанное приложение моделирует полный цикл работы q-арных кодов Хэмминга: построение проверочной и порождающей матриц, кодирование информационного слова, внесение одиночной ошибки и её исправление посредством синдромного декодирования. Полученные результаты позволяют всесторонне оценить корректность теории, а также сравнить свойства кодов над различными полями GF(q). Ниже приводится аналитический разбор вычислительных экспериментов, выполненных для полей GF(2), GF(3) и GF(5).

Программа строит проверочную матрицу H на основе системы представителей одномерных подпространств пространства $GF(q)^m$. В каждом эксперименте было подтверждено:

– Матрица H имеет размер $m \times n$, где $n = (q^m - 1)/(q - 1)$.

– Все столбцы H попарно не являются пропорциональными, что гарантирует минимальное расстояние $d = 3$.

– Выведенная матрица G корректно удовлетворяет условию $H \cdot G^T = 0$, что означает корректность построения порождающей матрицы как базиса ядра проверочной матрицы H.

Для всех рассмотренных параметров (GF(2), GF(3), GF(5)) матрицы G и H совпадают с теоретически ожидаемыми структурами, приведёнными в разделе теории, что подтверждает корректность алгоритмического блока построения кода.

Для классического кода Хэмминга с параметрами (7,4,3) программа продемонстрировала полный цикл исправления одиночной ошибки. Во всех запусков:

- синдром s однозначно соответствовал одному из одноошибочных синдромов, заранее занесённых в таблицу синдромов,
- найденная позиция ошибки соответствовала внесённой позиции,
- восстановленное кодовое слово совпадало с исходным.

Это подтверждает, что бинарный случай реализован корректно и что алгоритм синдромного декодирования работает полностью в соответствии с классической схемой Хэмминга.

Троичный код с параметрами (4,2,3) демонстрирует важное отличие от бинарного: ошибка может иметь произвольную ненулевую величину $e \in GF(3)$.

В результате программа должна определить не только позицию ошибки, но и её значение. В ходе экспериментов для различных информационных слов и разных значений ошибки (например, +1 или +2):

- синдром всегда принимал вид $s = e\, h_j$,
- программа корректно находила j (позицию ошибки) и e (её величину),

Это подтверждает, что предварительно сформированная таблица синдромов корректно учитывает как позицию ошибки, так и её величину $e \in GF(q)$, а сама библиотека galois обеспечивает правильную арифметику поля GF(3).

Для поля GF(5) код Хэмминга имеет параметры (6,4,3). Этот случай особенно интересен, поскольку поле содержит пять значений, и величины ошибки могут быть 1, 2, 3 или 4.

Программа корректно обрабатывает:

- сложение ошибок различной величины,
- вычисление синдрома в GF(5),
- поиск скаляра e, для которого выполняется $s = e\, h_j$,
- восстановление изначального кодового слова.

Было показано, что даже при случайном многократном внесении ошибки в различных позициях и со случайными ненулевыми значениями, декодирование всегда завершалось корректным восстановлением исходного слова. Это свидетельствует о корректной генерализации двоичного алгоритма на произвольные q.

На основании всех экспериментов можно сформулировать следующие наблюдения:

- **При увеличении q растёт длина кода n и его пространство кодовых слов**, что соответствует формуле $n = (q^m-1)/(q-1)$.
- **Чем больше q, тем выше информационная ёмкость кода**: доля информационных символов k/n растёт.
- **Структура матрицы H становится богаче**, поскольку количество одномерных подпространств возрастает.
- **Процесс декодирования становится более насыщенным**, так как необходимо определить не только позицию ошибки, но и её величину.
- **Тем не менее, алгоритм остаётся линейным и однозначным**, что подтверждает универсальность конструкции Хэмминга среди совершенных кодов.

Эти наблюдения демонстрируют, что q-арные коды Хэмминга действительно являются естественным и мощным обобщением бинарного случая.