МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«Крымский федеральный университет имени В.И. Вернадского»

Физико-технический институт

Кафедра компьютерной инженерии и моделирования

Лабораторная работа № 5
«Код Хемминга»
по дисциплине
«Теория информации и кодирование»

Выполнил:
студент 3 курса
группа ИВТ-222
Гоголев В. Г

Проверил:
Таран Е.П.
«____» _____20___г.
Подпись: _____

Симферополь, 2024

Цель: построить помехоустойчивый код (код Хэмминга), который позволяет обнаруживать и обнаруживать и исправлять ошибки в кодовых комбинациях заданной кратности.

Техническое задание: источник информации вырабатывает сообщения, содержащие k информационных разрядов. Значения разрядов генерируются в двоичной системе счисления счетчиком случайных чисел. Необходимо: 1. разработать программное обеспечение для передатчика, которое будет строить код Хэмминга с заданной исправляющей способностью; 2. разработать программное обеспечение на приемной стороне, позволяющее обрабатывать принятый код Хэмминга; 3. провести комплекс численных экспериментов, в ходе которых на передающей стороне построить код Хэмминга с заданной исправляющей способностью, сгенерировать случайным образом кратность ошибки и ошибочную кодовую комбинацию, на приемной стороне по принятому коду Хэмминга определить кратность ошибки и скорректировать принятую кодовую комбинацию.

Ход работы:

Вариант № 4

Задание I. С использованием разработанного программного обеспечения для передатчика и для приемника провести комплекс численных экспериментов (не менее 6), входе которого необходимо:

а) сгенерировать случайным образом информационную кодовую комбинацию, состоящую из k разрядов, на передающей стороне;

б) построить код Хэмминга, позволяющий обнаруживать и исправлять все однократные ошибки;

в) модифицировать код Хэмминга и построить код, позволяющий обнаруживать двукратные ошибки;

г) передать сформированный код Хэмминга со входа на выход, сгенерировав при этом случайным образом кратность ошибки (от 0 до 2), и сгенерировав случайным образом позицию ошибки в принятом коде Хэмминга с учетом кратности;

д) с использованием программного обеспечения на приемной стороне обработать принятый код Хэмминга, определить синдром ошибки, рассчитать кратность ошибки и определить позицию ошибки для однократных ошибок.

```python
def generate_random_bits(n):
    return [random.randint(0, 1) for _ in range(n)]

def calculate_hamming_code(data_bits):
    # Считаем кол-во проверочных битов
    n = len(data_bits)
    r = 0
    while (2**r) < (n + r + 1):
        r += 1
    total_bits = n + r
    hamming_code = [0] * total_bits

    # Располагаем информационные биты в коде Хэмминга(выбираем место под них)
    j = 0
    for i in range(1, total_bits + 1):
        if i & (i - 1) != 0: #проверяем эллемент является ли он степенью двойки
            hamming_code[i - 1] = data_bits[j]
            j += 1

    # Рассчитываем значения контрольных битов
    for i in range(r):
        control_bit_position = 2**i #позиция проверочного бита(степень двойки)
        control_sum = 0
        for j in range(1, total_bits + 1):
            if j & control_bit_position != 0:
                control_sum ^= hamming_code[j - 1]
        hamming_code[control_bit_position - 1] = control_sum

    return hamming_code,r
```

Рисунок 1 – функции generate_random_bits и calculate_hamming_code

Функция generate_random_bits принимает на вход число k(кол-во разрядов), и создает массив случайных чисел длинны k на основе генератора случайных чисел.

Вторая функция calculate_hamming_code принимает на вход массив из первой функции, изначально рассчитывается число r – это кол-во проверочных битов для комбинации, далее выбираются места в массиве для контрольных бит, путем проверки на степень двойки, потом идет вычисление значений контрольных бит.

```python
# вводим ошибки в код Хэмминга
def introduce_errors(hamming_code, num_errors=1):
    corrupted_code = hamming_code[:]
    error_indices = random.sample(range(len(hamming_code)), num_errors) # позиции ошибок
    for idx in error_indices:
        corrupted_code[idx] ^= 1  # Инвертируем бит
    return corrupted_code, error_indices

#вычисление синдрома ошибки
def calculate_syndrome(corrupted_code, r):
    syndrome = 0
    for i in range(r):
        control_bit_position = 2**i #позиция проверочного бита(степень двойки)
        control_sum = 0
        for j in range(1, len(corrupted_code) + 1):
            if j & control_bit_position != 0: # проверка входит ли текущая позиция(j) в область проверки бита
                control_sum ^= corrupted_code[j - 1]
        syndrome |= (control_sum << i)
        print(syndrome)
    return syndrome
```

Рисунок 2 – функция introduce_errors и calculate_syndrome

Функция introduce_error – принимает на вход массив кода хемминга, и кол-во ошибок, далее создается копия массива, в нем выбирается случайно 1 или 2 индекса в которых будет ошибка, далее в выбранных битах инвертируем значение.

Функция calculate_syndrome принимает на вход код Хемминга с ошибкой и кол-во проверочных битов.

Далее в цикле вычисляется синдром для последовательности, как сумма контрольных сумм для каждого проверочного бита, на выходе функция возвращает int значение синдрома.

Синдром позволяет определить есть ли ошибка и её позицию, по факту синдром = N + 1,где N – индекс ошибки, и благодаря синдрому можно далее идентифицировать где была произведена ошибка и исправить её.

```python
#исправление ошибки в коде Хеминга
def correct_single_error(corrupted_code, syndrome):
    if syndrome > 0:
        error_position = syndrome - 1
        corrupted_code[error_position] ^= 1  # Исправляем ошибку
    return corrupted_code

# Извлечение исходной информационной комбинации из исправленного кода Хэмминга
def extract_information_bits(hamming_code):
    data = []
    n = len(hamming_code)
    for i in range(1, n + 1):
        if (i & (i - 1)) != 0:  # Пропускаем позиции степеней двойки
            data.append(hamming_code[i - 1])
    return data
```

Рисунок 3 – функции correct_single_error и extract_information_bits

Первая функция исправляет ошибку в коде Хемминга, она принимает код с ошибкой и её синдром. Далее мы обращаемся к элементу с индексом [syndrome – 1] и инвертируем значение в данной ячейке.

Вторая функция служит для того, чтобы удалить из исправленного кода Хемминга проверочные биты, чтобы на приемной стороне получать исходное сообщение и была возможность сравнить его с изначально сгенерированной комбинацией.

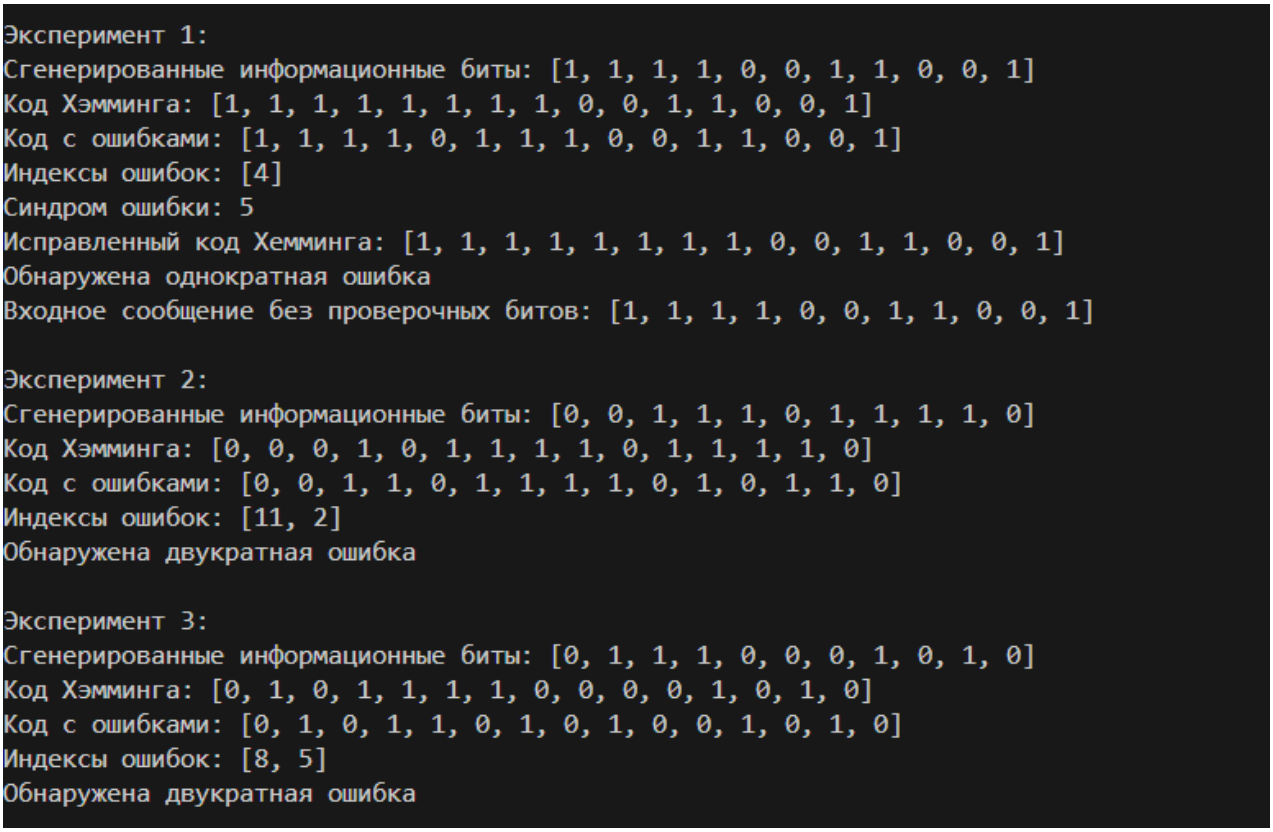

Рисунок 4 – результат работы программы

Программа выполняет 6 экспериментов, выводит исходную кодовую комбинацию, код Хэмминга, код Хэмминга с ошибкой, если это однократная ошибка, то исправляет её, считает для неё индекс, синдром и восстанавливает исходное сообщение, если ошибка кратна 2, то просто показывает индексы ошибочных элементов.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы были получены навыки по формированию кода Хемминга, по созданию однократных и двукратных ошибок в процессе передачи сообщения, расчету контрольной суммы для контролирующего бита и синдрома ошибки, так же освоены навыки по исправлению ошибки в коде Хемминга на приемной стороне и извлечения входной комбинации из кода Хемминга.

ПРИЛОЖЕНИЕ

```
import random

#генерация входного сообщения
def generate_random_bits(n):
    return [random.randint(0, 1) for _ in range(n)]

def calculate_hamming_code(data_bits):
    # Считаем кол-во проверочных битов
    n = len(data_bits)
    r = 0
    while (2**r) < (n + r + 1):
        r += 1
    total_bits = n + r
    hamming_code = [0] * total_bits

    # Располагаем информационные биты в коде Хэмминга(выбираем место под них)
    j = 0
    for i in range(1, total_bits + 1):
        if i & (i - 1) != 0: #проверяем эллемент является ли он степенью двойки
            hamming_code[i - 1] = data_bits[j]
            j += 1

    # Рассчитываем значения контрольных битов
    for i in range(r):
        control_bit_position = 2**i #позиция проверочного бита(степень двойки)
        control_sum = 0
        for j in range(1, total_bits + 1):
            if j & control_bit_position != 0:
                control_sum ^= hamming_code[j - 1]
        hamming_code[control_bit_position - 1] = control_sum

    return hamming_code,r

# вводим ошибки в код Хэмминга
def introduce_errors(hamming_code, num_errors=1):
    corrupted_code = hamming_code[:]
    error_indices = random.sample(range(len(hamming_code)), num_errors) # позиции
ошибок
    for idx in error_indices:
        corrupted_code[idx] ^= 1  # Инвертируем бит
    return corrupted_code, error_indices

#вычисление синдрома ошибки
def calculate_syndrome(corrupted_code, r):
    syndrome = 0
    for i in range(r):
        control_bit_position = 2**i #позиция проверочного бита(степень двойки)
        control_sum = 0
        for j in range(1, len(corrupted_code) + 1):
```

```
            if j & control_bit_position != 0: # проверка входит ли текущая
позиция(j) в область проверки бита
                control_sum ^= corrupted_code[j - 1]
        syndrome |= (control_sum << i)
        print(syndrome)
    return syndrome

#исправление ошибки в коде Хеминга
def correct_single_error(corrupted_code, syndrome):
    if syndrome > 0:
        error_position = syndrome - 1
        corrupted_code[error_position] ^= 1  # Исправляем ошибку
    return corrupted_code

# Извлечение исходной информационной комбинации из исправленного кода Хэмминга
def extract_information_bits(hamming_code):
    data = []
    n = len(hamming_code)
    for i in range(1, n + 1):
        if (i & (i - 1)) != 0:  # Пропускаем позиции степеней двойки
            data.append(hamming_code[i - 1])
    return data


def main():
    n = 11  # Количество информационных битов

    for attempt in range(6):  # 6 попыток
        print(f"\nЭксперимент {attempt + 1}:")

        # 1. Генерация информационных битов
        data_bits = generate_random_bits(n)
        print("Сгенерированные информационные биты:", data_bits)

        # 2. Код Хэмминга
        hamming_code,r = calculate_hamming_code(data_bits)
        print("Код Хэмминга:", hamming_code)

        # 3. Введение ошибок (одной или двух)
        num_errors = random.choice([1, 2])
        corrupted_code, error_indices = introduce_errors(hamming_code, num_errors)
        print("Код с ошибками:", corrupted_code)
        print("Индексы ошибок:", error_indices)

        # 4. Вычисление синдрома
        if num_errors == 1:
            syndrome = calculate_syndrome(corrupted_code, r)
            print("Синдром ошибки:", syndrome)

        # 5. Исправление ошибки, если она однократная
        if  num_errors == 1:
```

```
            corrected_haming_code = correct_single_error(corrupted_code[:],
syndrome)
            corrected_input_code = extract_information_bits(corrected_haming_code)
            print("Исправленный код Хемминга:", corrected_haming_code)
            print("Обнаружена однократная ошибка")
            print("Входное сообщение без проверочных битов:",
corrected_input_code)
        elif num_errors == 2:
            print("Обнаружена двукратная ошибка")

main()
```

Приложение 1 – листинг программного кода