

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:

Филиппов Д.М.

«\_\_\_\_» \_\_\_\_\_\_\_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;

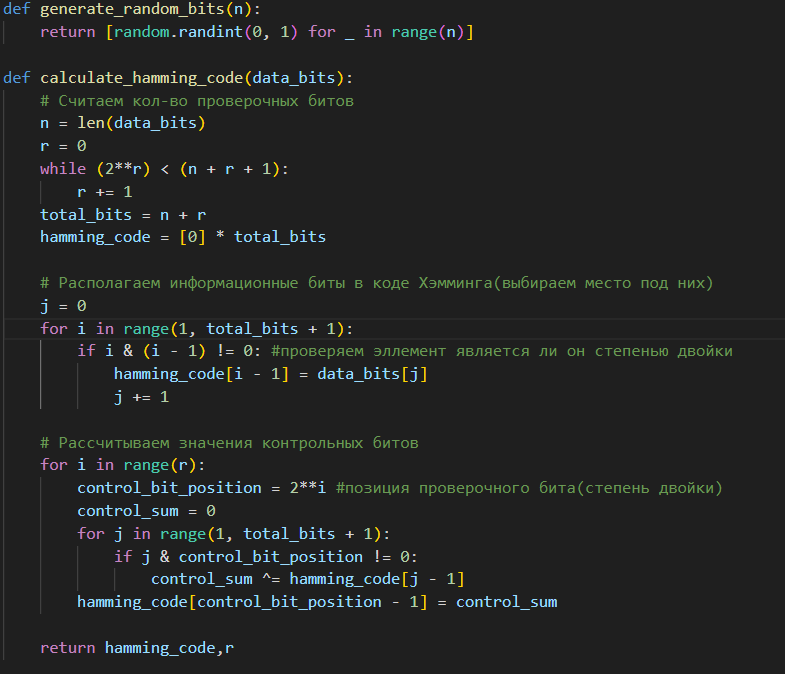
д) с использованием программного обеспечения на приемной стороне обработать принятый код Хэмминга, определить синдром ошибки, рассчитать кратность ошибки и определить позицию ошибки для однократных ошибок.

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.

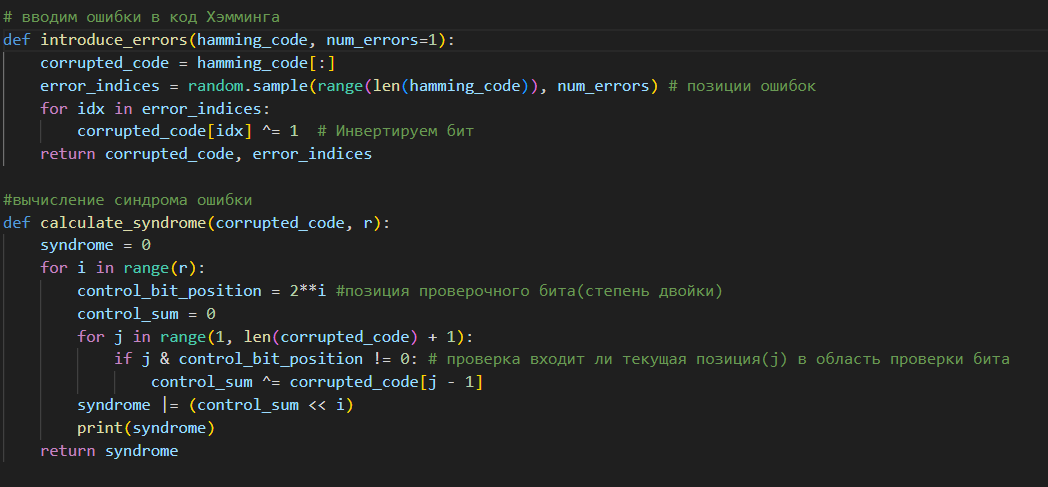
Вторая функция calculate\_hamming\_code принимает на вход массив из первой функции, изначально рассчитывается число r – это кол-во проверочных битов для комбинации, далее выбираются места в массиве для контрольных бит, путем проверки на степень двойки, потом идет вычисление значений контрольных бит.

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.

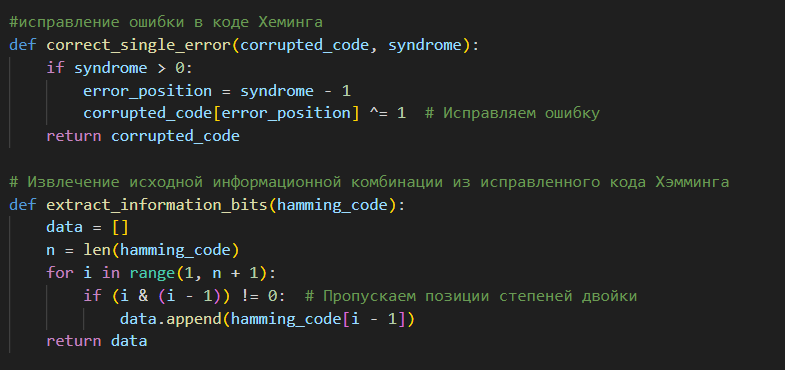
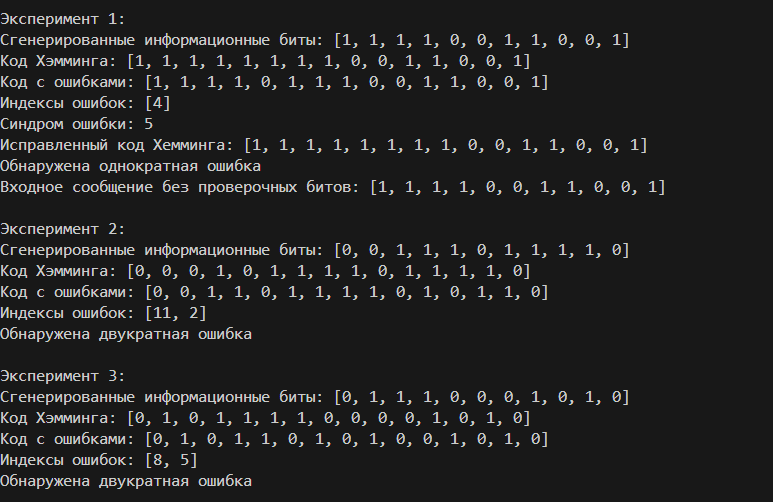


Рисунок 3 – функции correct\_single\_error и extract\_information\_bits

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