МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ
Кафедра интеллектуальных информационных технологий

Отчет
по дисциплине
«Современные методы защиты компьютерных систем»
по лабораторной работе № 1
«Избыточное кодирование данных в информационных системах. Код Хемминга»

Выполнила:
студентка 4
курса группы
ИИ-22
Сокол С.М.
Проверила:
Хацкевич А.С.

Брест 2024

Цель: приобретение практических навыков кодирования/декодирования двоичных данных при использовании кода Хэмминга.

Постановка задачи:

- Разработать приложение для кодирования/декодирования двоичной информации кодом Хемминга с минимальным кодовым расстоянием 3 или 4
- Результаты выполнения лабораторной работы оформить в виде отчета с листингом разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

Вариант	M	r
4	799	5

Ход работы: Была написана программа на языке Python, для реализации необходимых требований

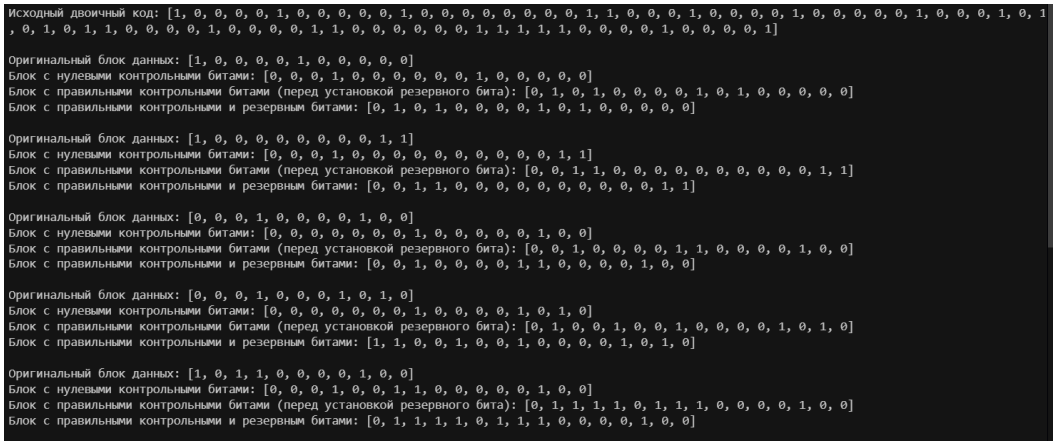
Код:

```
import numpy as np
from functools import reduce
from operator import xor

def is_power_of_two(n):
    return (n & (n - 1)) == 0 and n != 0

def calculate_r(m):
    r = 0
    while m != 1:
        m = m // 2
        r += 1
    return r

def insert_parity_bits(data_bits, r):#вставляют контрольные биты в кодовые слова
    m = len(data_bits)
    total_len = m + r + 1
    result = [0]
    j = 0
    for i in range(1, total_len):
        if (i & (i - 1)) == 0:
            result.append(0)
        else:
            result.append(data_bits[j])
            j += 1
    return result

def hamming_syndrom(bits):
    return reduce(xor, [i for i, bit in enumerate(bits) if bit])

def fix_error(bits, error):
    if error != 0:
        bits[error] = 1 - bits[error]
    return bits
```

```
    def calculate_parity_bits(bits, r):
        n = len(bits)
        for i in range(r):
            parity_pos = (1 << i)
            parity = 0
            for j in range(parity_pos, n, 1 << (i + 1)):
                parity ^= reduce(xor, bits[j:j + (1 << i)], 0)
            bits[parity_pos] = parity
        return bits


    def create_one_error(bits):
        error_bit = np.random.randint(1, len(bits))
        bits[error_bit] = 1 - bits[error_bit]
        return bits

def set_reserved_bit(bits):
        total_ones = sum(bits[1:])
        bits[0] = total_ones % 2
        return bits


    def extract_original_data(bits, r):
        original_data = []
        for i in range(1, len(bits)):
            if not is_power_of_two(i):
                original_data.append(bits[i])
        return original_data

    def text_to_binary(text):#функция преобразует текст в последовательность двоичных цифр
        binary_data = ''.join(format(ord(char), '08b') for char in text)
        return [int(bit) for bit in binary_data]

    def binary_to_text(binary_data):
        chars = [chr(int(''.join(map(str, binary_data[i:i + 8])), 2)) for i in range(0, len(binary_data), 8)]
        return ''.join(chars)

    def hamming_process_file(full_block_length, file_path):
        if not is_power_of_two(full_block_length):
            raise ValueError("Длина блока должна быть степенью двойки.")



        with open(file_path, 'r') as f:
            text = f.read()
        binary_data = text_to_binary(text)
        print(f"Исходный двоичный код: {binary_data}")

        r = calculate_r(full_block_length)#контрольные биты
        block_data_size = full_block_length - r - 1
        blocks = [binary_data[i:i + block_data_size] for i in range(0, len(binary_data), block_data_size)]
```

```
    processed_blocks = []
    for block in blocks:
        print(f"\nОригинальный блок данных: {block}")
        bits_with_parity = insert_parity_bits(block, r)
        print(f"Блок с нулевыми контрольными битами: {bits_with_parity}")
        updated_bits = calculate_parity_bits(bits_with_parity.copy(), r)
        print(f"Блок с правильными контрольными битами (перед установкой резервного бита): {updated_bits}")
        updated_bits = set_reserved_bit(updated_bits)
        print(f"Блок с правильными контрольными и резервным битами: {updated_bits}")
        processed_blocks.append(updated_bits)
    error_block_index = np.random.randint(0, len(processed_blocks))
    print(f"\nДобавляем ошибку в блок {error_block_index}")
    processed_blocks[error_block_index] = create_one_error(processed_blocks[error_block_index])

    #добавление второй ошибки
    processed_blocks[error_block_index] = create_one_error(processed_blocks[error_block_index])

    corrected_blocks = []
    is_two_errors = False
    for i, block in enumerate(processed_blocks):
        print(f"\nПроверка блока {i}: {block}")
        reserved_bit = block[0]
        find_error = hamming_syndrom(block)#находит положение ошибки (если она есть)
        if find_error > 0:
            print(f"Найдена ошибка в позиции: {find_error}")



        total_ones = sum(block[1:])
        if find_error == 0:
            print("Ошибок не найдено.")
            corrected_block = block
        else:
            if reserved_bit != (total_ones % 2):
                print("Одна ошибка найдена.")
                corrected_block = fix_error(block.copy(), find_error)
            else:
                is_two_errors = True
                print("Две ошибки найдены, исправление не выполняется.")
                corrected_block = block
        corrected_blocks.append(extract_original_data(corrected_block, r))

    restored_binary_data = [bit for block in corrected_blocks for bit in block]
    restored_text = binary_to_text(restored_binary_data)
    print(f"\nВосстановленный текст: {restored_text}")
    if is_two_errors:
        print("Текст не исправлен полностью из-за наличия двух ошибок.")
    else:
        print("Текст исправлен полностью.")

def text_to_binary(text):
```

```
    # Обрезаем текст до первых 7 символов
    text = text[:7]
    binary_data = ''.join(format(ord(char), '08b') for char in text)
    return [int(bit) for bit in binary_data]
file_path = 'C:\\Universitet\\No_KMZI\\message.txt'
full_block_length = 16
hamming_process_file(full_block_length, file_path)
```



Вывод: приобрели практических навыков кодирования/декодирования двоичных данных при использовании кода Хэмминга.