МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

Кафедра ИИТ

ОТЧЁТ

По лабораторной работе №1

«Избыточное кодирование данных в информационных системах. Код Хемминга»

Выполнил:
Студент группы ИИ-22
Нестерчук Д.Н.
Проверил: Хацкевич
А.С.

2024

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

Задание.

1. Закрепить теоретические знания по использованию методов помехоустойчивого кодирования для повышения надежности передачи и хранения в памяти компьютера двоичных данных.
2. Разработать приложение для кодирования/декодирования двоичной информации кодом Хемминга с минимальным кодовым расстоянием 3 или 4.
3. Результаты выполнения лабораторной работы оформить в виде отчета с листингом разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.
4. Ответить на контрольные вопросы **Ход работы**

Вариант	M[1]	r
15	636	3

1. Составить код Хемминга (классический алгоритм) (M+r, M), допустить ошибку в одном из разрядов и отыскать её по алгоритму.

2. Составить код Хемминга (расширенный алгоритм) (первые 7 битов M + 3 проверочных, первые 7 битов M), допустить 2 или более ошибок в разрядах и отыскать их по алгоритму

Код программы:

```
import math

r = 0

def encode_hamming_code(number):
    binary_number = bin(number)[2:]
    data_length = len(binary_number)

    global r
    r = 0
    while 2 ** r < (data_length + r + 1):
        r += 1

    code_length = data_length + r
    code = [0] * code_length
    XOR = 0

    j = 0
    for i in range(code_length):
        if math.log(i + 1, 2).is_integer():
            code[i] = 0
        else:
            code[i] = int(binary_number[j])
            j += 1
            if code[i] == 1:
                XOR ^= i + 1

    str_XOR = bin(XOR)[2:].zfill(r)
```

```
    for i in range(r):
        position = 2 ** i
        code[position - 1] = int(str_XOR[i])

    print("\nGenerated Hamming Code:")
    print("".join(map(str, code)))
    return code

def check_hamming_code(code):
    XOR = 0
    for i in range(len(code)):
        if code[i] == 1:
            XOR ^= i + 1

    if XOR != 0:
        print(f"Error detected at position: {XOR}")
        code[XOR - 1] ^= 1
        print("Corrected Hamming Code: ", end="")
        print("".join(map(str, code)))
    else:
        print("No errors detected.")

def decode_hamming_code(code):
    data_bits = "".join(str(code[i]) for i in range(len(code)) if
not math.log(i + 1, 2).is_integer())
    return int(data_bits, 2)

# Пример использования
number = 25  # Исходное число
encoded_code = encode_hamming_code(number)
check_hamming_code(encoded_code)

# Допустим, ошибка была внесена в закодированное сообщение
encoded_code[4] ^= 1  # Симулируем ошибку
check_hamming_code(encoded_code)

decoded_number = decode_hamming_code(encoded_code)
print(f"Decoded number: {decoded_number}")
```

Результат работы:

```
Entered number: 636

Generated Hamming Code:
00100010111100
Incorrected Hamming Code: 10100010111100
Error detected at position: 1
Corrected Hamming Code: 00100010111100

Incorrected Hamming Code: 01100010111100
Error detected at position: 2
Corrected Hamming Code: 00100010111100

Incorrected Hamming Code: 00000010111100
Error detected at position: 3
Corrected Hamming Code: 00100010111100
```

Вывод: приобрёл практические навыки кодирования/декодирования двоичных данных при использовании кода Хемминга.