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
Кузьмич В.Н.
Проверила:
Хацкевич А.С.

Брест 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
9	774	4

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:

Код Хэминга

```
###########____1 задание_____
import random

def to_binary(n, bits):
    return f"{n:0{bits}b}"

def generate_hamming_code(data_bits, r):
    m = len(data_bits)
    total_length = m + r
    hamming_code = [0] * total_length

    j = 0
    for i in range(1, total_length + 1):
        if (i & (i - 1)) != 0:
            hamming_code[i - 1] = int(data_bits[j])
            j += 1

    for i in range(r):
        pos = 2 ** i
        parity = 0
        for j in range(1, total_length + 1):
            if j & pos != 0:
                parity ^= hamming_code[j - 1]
        hamming_code[pos - 1] = parity

    return hamming_code

def introduce_error(hamming_code):

    error_pos = random.randint(0, len(hamming_code) - 1)
    hamming_code[error_pos] ^= 1
    return hamming_code, error_pos

def find_error(hamming_code, r):
    error_pos = 0
    for i in range(r):
        pos = 2 ** i
        parity = 0
        for j in range(1, len(hamming_code) + 1):
            if j & pos != 0:
                parity ^= hamming_code[j - 1]
        error_pos += pos if parity != 0 else 0
    return error_pos

M = 748
```

```
M_binary = to_binary(M, 11)
r = 5

hamming_code = generate_hamming_code(M_binary, r)
print("Код Хемминга без ошибок:", hamming_code)

hamming_code_with_error, error_pos = introduce_error(hamming_code.copy())
print("Код Хемминга с ошибкой:", hamming_code_with_error)
print("Ошибка введена в позицию:", error_pos + 1)


found_error_pos = find_error(hamming_code_with_error, r)
print("Найденная позиция ошибки:", found_error_pos
```

Расширенный код хэминга

```
###########_____2 задание_____
def generate_extended_hamming_code(data_bits, r):

    m = len(data_bits)
    total_length = m + r
    hamming_code = [0] * total_length

    j = 0
    for i in range(1, total_length + 1):
        if (i & (i - 1)) != 0:
            hamming_code[i - 1] = int(data_bits[j])
            j += 1

    for i in range(r):
        pos = 2 ** i
        parity = 0
        for j in range(1, total_length + 1):
            if j & pos != 0:
                parity ^= hamming_code[j - 1]
        hamming_code[pos - 1] = parity

    overall_parity = sum(hamming_code) % 2
    hamming_code.append(overall_parity)

    return hamming_code

def introduce_double_error(hamming_code):
    import random
    pos1 = random.randint(0, len(hamming_code) - 2)
    pos2 = pos1
    while pos2 == pos1:
        pos2 = random.randint(0, len(hamming_code) - 2)

    hamming_code[pos1] ^= 1
    hamming_code[pos2] ^= 1

    return hamming_code, (pos1, pos2)

def find_errors_with_extended_code(hamming_code, r):
    error_pos = 0
    for i in range(r):
        pos = 2 ** i
        parity = 0
        for j in range(1, len(hamming_code)):
            if j & pos != 0:
                parity ^= hamming_code[j - 1]
        error_pos += pos if parity != 0 else 0

    overall_parity = sum(hamming_code[:-1]) % 2
    if overall_parity != hamming_code[-1]:
        if error_pos == 0:
            print("Обнаружены две ошибки.")
        else:
            print("Обнаружена ошибка на позиции:", error_pos)
    else:
        if error_pos == 0:
            print("Ошибок нет.")
        else:
            print("Обнаружена одиночная ошибка на позиции:", error_pos)

M_binary = to_binary(M, 11)[:7]
r = 3

extended_hamming_code = generate_extended_hamming_code(M_binary, r)
print("Расширенный код Хемминга без ошибок:", extended_hamming_code)

extended_hamming_code_with_errors, error_positions = introduce_double_error(extended_hamming_code.copy())
print("Расширенный код Хемминга с двумя ошибками:", extended_hamming_code_with_errors)
```

```
print("Ошибки введены на позиции:", error_positions[0] + 1, "и", error_positions[1] + 1)

find_errors_with_extended_code(extended_hamming_code_with_errors, r)
```

Результат работы:

```
Код Хемминга без ошибок: [0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0]
Код Хемминга с ошибкой: [0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0]
Ошибка введена в позицию: 14
Найденная позиция ошибки: 14
Расширенный код Хемминга без ошибок: [1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1]
Расширенный код Хемминга с двумя ошибками: [1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1]
Ошибки введены на позиции: 10 и 8
Обнаружена одиночная ошибка на позиции: 2
```

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