

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

Дубина Н.С.

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'	r
4	799	5

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:

```
class HammingCode:
    def __init__(self, message, r):
        self.message = message
        self.r = r
        self.n = len(message) + r # Полная длина закодированного слова
        self.code = [0] * self.n

    def encode(self):
        # Расставляем информационные биты и проверочные биты на позиции, кратные 2^k
        j = 0
        for i in range(1, self.n + 1):
            # Проверочные биты на позициях 1, 2, 4, 8 и т.д.
            if (i & (i - 1)) == 0:
                continue # Эти позиции оставляем для проверочных битов
            else:
                # Расставляем информационные биты
                if j < len(self.message): # Проверяем, чтобы не выйти за пределы
индексов
                    self.code[i - 1] = int(self.message[j])
                    j += 1

        # Вычисляем значения проверочных битов
        for i in range(self.r):
            pos = 1 << i # Позиция проверочного бита (1, 2, 4, 8, ...)
            if pos - 1 >= self.n: # Проверяем, чтобы не выйти за пределы code
                continue
            sum_bits = 0
            for j in range(1, self.n + 1):
                if j & pos:
                    sum_bits ^= self.code[j - 1]
            self.code[pos - 1] = sum_bits
        print("Сформированный код:", ''.join(map(str, self.code)))
        return self.code

    def introduce_error(self, position):
        # Инvertируем бит на указанной позиции для имитации ошибки
        self.code[position - 1] ^= 1
        print("Код с ошибкой в позиции", position, ":", ''.join(map(str, self.code)))

    def detect_and_correct(self):
        # Обнаружение позиции ошибки
        error_position = 0
        for i in range(self.r):
            pos = 1 << i
            sum_bits = 0
            for j in range(1, self.n + 1):
                if j & pos:
```

```

        sum_bits ^= self.code[j - 1]
    if sum_bits != 0:
        error_position += pos

    if error_position == 0:
        print("Ошибок не обнаружено.")
    else:
        print("Обнаружена ошибка в позиции:", error_position)
        self.code[error_position - 1] ^= 1
        print("Исправленный код:", ''.join(map(str, self.code)))
    return error_position

def decode(self):
    # Извлекаем только информационные биты
    decoded_message = []
    for i in range(1, self.n + 1):
        if (i & (i - 1)) != 0:
            decoded_message.append(str(self.code[i - 1]))
    decoded_message = ''.join(decoded_message[:len(self.message)])
    print("Расшифрованное сообщение:", decoded_message)
    return decoded_message

# Пример использования
message = "1100011111" # Исходное сообщение длиной 10 бит
r = 4 # Количество проверочных битов

hamming = HammingCode(message, r)
encoded_code = hamming.encode() # Кодирование
hamming.introduce_error(3) # Допустим ошибку в третьем бите
hamming.detect_and_correct() # Обнаружение и исправление ошибки
decoded_message = hamming.decode() # Расшифровка исходного сообщения

```

## Результат работы:

```

Сформированный код: 00101001011111
Код с ошибкой в позиции 3 : 00001001011111
Обнаружена ошибка в позиции: 3
Исправленный код: 00101001011111
Расшифрованное сообщение: 1100011111

```

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