МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа № 7

по дисциплине: Теория информации

тема: «Исследование помехоустойчивых кодов на примере алгоритма Хэмминга»

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович

Проверил:
Твердохлеб Виталий Викторович

Белгород 2024г.

Лабораторная работа №7

«Исследование помехоустойчивых кодов на примере алгоритма Хэмминга»

Задание 1

Закодировать по Хэммингу произвольно сформированные последовательности двоичных символов длиной:

- 18 бит;

- 48 бит.

Сделать выводы относительно эффективности каждого их сообщений.

```
import random

# Длина блока кодирования
CHUNK_LENGTH = 8

# Проверка длины блока
assert CHUNK_LENGTH > 0, 'CHUNK_LENGTH должен быть больше 0'

# Вычисление контрольных бит
CHECK_BITS = [i for i in range(1, CHUNK_LENGTH + 1) if not i & (i - 1)]

def chars_to_bin(chars):
    """
    Преобразование символов в бинарный формат
    """
    assert not len(chars) * 8 % CHUNK_LENGTH, 'Длина кодируемых данных должна
быть кратна длине блока кодирования'
    return ''.join([bin(ord(c))[2:].zfill(8) for c in chars])

def chunk_iterator(text_bin, chunk_size=CHUNK_LENGTH):
    """
    Поблочный вывод бинарных данных
    """
    for i in range(0, len(text_bin), chunk_size):
        yield text_bin[i:i + chunk_size]

def get_check_bits_data(value_bin):
    """
    Получение информации о контрольных битах из бинарного блока данных
    """
    check_bits_count_map = {k: 0 for k in CHECK_BITS}
    for index, value in enumerate(value_bin, 1):
        if int(value):
            bin_char_list = list(bin(index)[2:].zfill(8))
            bin_char_list.reverse()
            for degree in [2 ** int(i) for i, value in enumer-
ate(bin_char_list) if int(value)]:
                check_bits_count_map[degree] += 1
    check_bits_value_map = {}
    for check_bit, count in check_bits_count_map.items():
        check_bits_value_map[check_bit] = 0 if not count % 2 else 1
    return check_bits_value_map

def set_empty_check_bits(value_bin):
    """
```

```
    Добавить в бинарный блок "пустые" контрольные биты
    """
    for bit in CHECK_BITS:
        value_bin = value_bin[:bit - 1] + '0' + value_bin[bit - 1:]
    return value_bin

def set_check_bits(value_bin):
    """
    Установить значения контрольных бит
    """
    value_bin = set_empty_check_bits(value_bin)
    check_bits_data = get_check_bits_data(value_bin)
    for check_bit, bit_value in check_bits_data.items():
        value_bin = '{0}{1}{2}'.format(value_bin[:check_bit - 1], bit_value,
value_bin[check_bit:])
    return value_bin

def get_check_bits(value_bin):
    """
    Получить информацию о контрольных битах из блока бинарных данных
    """
    check_bits = {}
    for index, value in enumerate(value_bin, 1):
        if index in CHECK_BITS:
            check_bits[index] = int(value)
    return check_bits

def exclude_check_bits(value_bin):
    """
    Исключить информацию о контрольных битах из блока бинарных данных
    """
    clean_value_bin = ''
    for index, char_bin in enumerate(list(value_bin), 1):
        if index not in CHECK_BITS:
            clean_value_bin += char_bin
    return clean_value_bin

def set_errors(encoded):
    """
    Допустить ошибку в блоках бинарных данных
    """
    result = ''
    for chunk in chunk_iterator(encoded, CHUNK_LENGTH + len(CHECK_BITS)):
        num_bit = random.randint(1, len(chunk))
        chunk = '{0}{1}{2}'.format(chunk[:num_bit - 1], int(chunk[num_bit -
1]) ^ 1, chunk[num_bit:])
        result += chunk
    return result

def check_and_fix_error(encoded_chunk):
    """
    Проверка и исправление ошибки в блоке бинарных данных
    """
    check_bits_encoded = get_check_bits(encoded_chunk)
    check_item = exclude_check_bits(encoded_chunk)
    check_item = set_check_bits(check_item)
    check_bits = get_check_bits(check_item)
    if check_bits_encoded != check_bits:
        invalid_bits = []
        for check_bit_encoded, value in check_bits_encoded.items():
            if check_bits[check_bit_encoded] != value:
                invalid_bits.append(check_bit_encoded)
        num_bit = sum(invalid_bits)
        encoded_chunk = '{0}{1}{2}'.format(encoded_chunk[:num_bit - 1],
```

```
int(encoded_chunk[num_bit - 1]) ^ 1, encoded_chunk[num_bit:])
    return encoded_chunk

def get_diff_index_list(value_bin1, value_bin2):
    """
    Получить список индексов различающихся битов
    """
    diff_index_list = []
    for index, char_bin_items in enumerate(zip(list(value_bin1),
list(value_bin2)), 1):
        if char_bin_items[0] != char_bin_items[1]:
            diff_index_list.append(index)
    return diff_index_list

def encode(source):
    """
    Кодирование данных
    """
    text_bin = chars_to_bin(source)
    result = ''
    for chunk_bin in chunk_iterator(text_bin):
        chunk_bin = set_check_bits(chunk_bin)
        result += chunk_bin
    return result

def decode(encoded, fix_errors=True):
    """
    Декодирование данных
    """
    decoded_value = ''
    fixed_encoded_list = []
    for encoded_chunk in chunk_iterator(encoded, CHUNK_LENGTH +
len(CHECK_BITS)):
        if fix_errors:
            encoded_chunk = check_and_fix_error(encoded_chunk)
        fixed_encoded_list.append(encoded_chunk)
    clean_chunk_list = []
    for encoded_chunk in fixed_encoded_list:
        encoded_chunk = exclude_check_bits(encoded_chunk)
        clean_chunk_list.append(encoded_chunk)
    for clean_chunk in clean_chunk_list:
        for clean_char in [clean_chunk[i:i + 8] for i in range(0,
len(clean_chunk), 8)]:
            decoded_value += chr(int(clean_char, 2))
    return decoded_value

if __name__ == '__main__':
    source = input('Укажите текст для кодирования/декодирования:')
    print('Длина блока кодирования: {0}'.format(CHUNK_LENGTH))
    print('Контрольные биты: {0}'.format(CHECK_BITS))
    encoded = encode(source)
    print('Закодированные данные: {0}'.format(encoded))
    decoded = decode(encoded)
    print('Результат декодирования: {0}'.format(decoded))
    encoded_with_error = set_errors(encoded)
    print('Допускаем ошибки в закодированных данных: {0}'.format(en-
coded_with_error))
    diff_index_list = get_diff_index_list(encoded, encoded_with_error)
    print('Допущены ошибки в битах: {0}'.format(diff_index_list))
    decoded = decode(encoded_with_error, fix_errors=False)
    print('Результат декодирования ошибочных данных без исправления ошибок:
{0}'.format(decoded))
    decoded = decode(encoded_with_error)
```

```
    print('Результат декодирования ошибочных данных с исправлением ошибок:
{0}'.format(decoded))
```

Для сообщения длинной 18 бит:

```
Укажите текст для кодирования/декодирования:111000110000100100
Длина блока кодирования: 0
Контрольные биты: [1, 2, 4, 8]
Закодированные данные:
100101110001100101110001100101110001100001100000100001100000100001100000100010
111000110010111000110000110000010000110000010000110000010000110000010010111000
011000011000001000011000001001011100011000011000001000011000001001011100011000
Результат декодирования: 111000110000100100
```

Для сообщения длинной 48 бит:

```
Укажите текст для кодирования/декодирова-
ния:110111000100101010100000111011000001000011100110
Длина блока кодирования: 0
Контрольные биты: [1, 2, 4, 8]
Закодированные данные:
100101110001100101110001100001100000100101110001100101110001100101110001100000
110000010000110000010000110000010010111000110000110000010000110000010010111000
011000011000001001011100011000011000001001011100011000011000001001011100011000
001100000100001100000100001100000100001100000100001100000100101110001100101110
000110010111000110000110000010010111000110010111000110000110000010000110000001
000001100000100001100000100001100000100101110001100001100000100001100000100001
100000100001100000100101110001100101110001100101110001100001100000100001100000
010010111000110010111000110000110000
Результат декодирования: 110111000100101010100000111011000001000011100110
```

Задание 2

Внести одиночную ошибку и устранить ее, используя механизм восстановления.

Для 18 бит:

```
Допускаем ошибки в закодированных данных:
100101100001100101010001100101111001100001000000100001000000100001100001100101
111010110010110000110001110000010000110100010000110010010000111000010011111000
010000011000001000011000100001011100011100011000001100011000000
Допущены ошибки в битах: [8, 19, 33, 43, 55, 72, 82, 92, 101, 117, 130, 140,
149, 157, 179, 181, 194, 206]
Результат декодирования ошибочных данных без исправления ошибок: 1!9
151p840q02100
Результат декодирования ошибочных данных с исправлением ошибок:
111000110000100100
```

Для 48 бит:

```
Допускаем ошибки в закодированных данных:
100100110001100101111001101001100000100100110001101101110001100101110011111000
110000010001110000000000110000010000111000100000110000010000110100000010111000
011000011001001001001100011001011000001001111100011000011100001011011100011000
011100000100001100010100001110000000001100000100001101000100101110011100101110
001110010111010100000110000010010111001100111110001110001100000100000100000100
001100000100001100001100000100000100101110101101001100000100101100000100001100
000100001000000101101110001110101110001101101110001100001110000100101100000100
010010111010110010101000110000111000
Допущены ошибки в битах: [6, 21, 27, 42, 51, 71, 74, 89, 97, 112, 121, 141,
145, 166, 174, 184, 197, 212, 219, 233, 251, 260, 265, 285, 299, 311, 322,
325, 345, 353, 362, 378, 386, 408, 414, 430, 435, 448, 464, 475, 483, 494,
507, 524, 532, 550, 559, 572]
Результат декодирования ошибочных данных без исправления ошибок:
 9° ±30p010814 0q0±p200833509q0 01 5°00 ±1±005!0
Результат декодирования ошибочных данных с исправлением ошибок:
110111000100101010100000111011000001000011100110
```

Задание 3

Рассмотреть вариант кодирования сообщения из 48 бит с предварительным сегментированием на блоки (размерность блоков выбрать самостоятельно).

Длинна блока равна 8:

```
Укажите текст для кодирования/декодирова-
ния:001011111010100100111100111000010101001110101101
Длина блока кодирования: 8
Контрольные биты: [1, 2, 4, 8]
Закодированные данные:
100001100000100001100000100101110001100001100000100101110001100101110001100101
111000110010111000110010111000110000110000010010111000110000110000010010111000
011000011000001000011000001001011100011000011000001000011000001001011100011000
101110001100101110001100101110001100001100000100001100000100101110001100101110
000110010111000110000110000010000110000010000110000010000110000010010111000110
000011000001001011100011000011000001001011100011000011000001000011000001001011
110001100101110001100101110001100001100000100101110001100001100000100101110001
100101110001100001100000100101110001
Результат декодирования: 001011111010100100111100111000010101001110101101
Допускаем ошибки в закодированных данных:
100001100001100001100001100101100001100001100001010010011000110010101000110000
111000110010101000110010111000010001110000010000111000110001110000011010111100
011000011000011000000100000000101110001100001000000000001100000100101111001101
101110001101101110001000101110001100001110000100001101000100101010001100101010
000111010111000110100110000011000110000010000110000110000011010001000011100011
000011000101001011110011000011001001101011100011100011000000000001100000100101
100001110101110001000101110001000001100000100100110001100001100010010010111010
110010101000110000110000111010111000 1
Допущены ошибки в битах: [12, 24, 32, 47, 54, 67, 76, 91, 108, 113, 124, 137,
146, 168, 174, 181, 199, 205, 225, 231, 243, 253, 272, 285, 295, 307, 314,
327, 338, 360, 369, 376, 395, 405, 418, 422, 434, 445, 464, 470, 481, 493,
510, 526, 538, 547, 564, 566]
Результат декодирования ошибочных данных без исправления ошибок:
1112 !1!0p1p11 1 09±±108!!1°01812941001110 45!11
Результат декодирования ошибочных данных с исправлением ошибок:
001011111010100100111100111000010101001110101101
```

Задание 4

Сравнить режимы кодирования с сегментацией и без.

- Режим кодирования Хэмминга без сегментации предоставляет простой и надежный способ обнаружения и исправления ошибок в одном блоке данных.

- Режим кодирования Хэмминга с сегментацией улучшает способность обработки больших объемов данных, обеспечивая более надежную коррекцию ошибок при передаче информации по частям.

- Выбор между этими режимами зависит от потребностей конкретной системы передачи данных и требований к эффективности исправления ошибок.

Вывод: В ходе выполнения лабораторной работы были изучены и сравнены два режима кодирования методом Хэмминга: с сегментацией данных и без неё. Кодирование Хэмминга без сегментации обеспечивает простой и надежный способ обнаружения и исправления ошибок в пределах одного блока данных, но ограничено в способности исправлять ошибки и неэффективно для больших объемов данных. В отличие от этого, кодирование Хэмминга с сегментацией улучшает обработку больших объемов данных и обеспечивает более надежную коррекцию ошибок при передаче информации частями. Выбор между этими методами зависит от потребностей конкретной системы передачи данных и требований к эффективности исправления ошибок.