Лабораторная работа 2

Тема:

“Защитное кодирование по методу Хэмминга”

Выполнил:

Студент группы 22207 Гордеев Никита

Задание:

- Составить программу, которая будет выполнять кодирование методом Хэмминга произвольной битовой последовательности заданной вариантом длины (задается в виде последовательности символов 0 и 1, вводимых без пробелов).

Значения параметров разбиения (Вариант 1):

- Длина кодируемой последовательности: 9
- Длина новой последовательности: 13

О методе:

При хранении данных и передаче их по каналам связи, возможно возникновение ошибок, даже одна из которых способна существенно исказить смысл сообщения. Для обнаружения и устранения ошибок применяются методы защитного кодирования, связанные с внесением в исходные данные некоторой избыточной информации.

Метод Хэмминга, позволяет фиксировать и исправлять единичные ошибки в блоке двоичных данных. При использовании этого метода вся последовательность данных разбивается на блоки фиксированной длины n. При кодировании длина фиксированного блока увеличивается до такой длины N, чтобы дополнительные биты позволяли сформировать число от нуля до N – для фиксации места ошибки.

Метод требует формирования матрицы контрольного суммирования, каждый столбец которой представляет собой двоичное значение порядкового номера столбца (число столбцов равно N). Эта матрица будет умножаться на передаваемый вектор.

Битами исходной последовательности заполняется, в порядке их следования, передаваемая последовательность длины N, исключая резервные позиции, соответствующие степеням числа 2. Резервные позиции формируются так, чтобы произведение каждой строки матрицы на этот вектор давало значение 0.

Решение:

1) Решение на бумаге:

Исходная последовательность													101101010											
Преобразованная последовательность													0011011001010											
i	*	*	1	*	2	3	4	*	5	6	7	8	9	s(0)	s(t)	s(r)	к. бит. степени двойки							
j	1	2	3	4	5	6	7	8	9	10	11	12	13				длина посл-и							
a1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	заполн. проп-в. ч-з 1	0101000	2	0101000	2	0111000	3	
a2	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	заполн. проп-в. ч-з 2	011110	4	011110	4	011110	4	
a3	0	0	0	1	1	1	1	0	0	0	0	1	1	1	0	1	заполн. проп-в. ч-з 4	001110	3	101110	4	111110	5	
a4	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	заполн. проп-в. ч-з 8	001010	2	001010	2	001010	2	
b	*	*	1	*	0	1	1	*	0	1	0	1	0				исх. сообщение							
β	0	0		1				0									перед-й вектор							
b(t)	0	0	1	1	0	1	1	0	0	1	0	1	0											
b(r)	0	0	1	1	1	1	1	0	0	1	0	1	0				полн. защ-й с ошиб.	ошибка = 1 + 4 = 5 ячейка						
														век. кс. исх. пос-ти	век. кс. пер. пос-ти	век. кс. ошиб. пос-ти								
														чёт-0 неч-1										

2) Программа:

Формирование матрицы контрольного суммирования, передаваемого вектора, векторов контрольных сумм), сопровождаемый комментариями, с предварительным описанием основных использованных переменных и массивов (тип, размерность, назначение и т.п.). Фрагменты должны быть выполнены в виде одной или нескольких функций (процедур).

```
# ###################### ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ ######################

def main():
    while True:
        user_select = int(input("Выберите режим:\n" + "(1): Кодирование методом Хэмминга\n" +
                                "(2): Проверка методом Хэмминга\n" + "(3): Выход\n" + "Ваш выбор: "))

        # Кодирование методом Хэмминга
        if user_select == 1:
            encode()

        # Проверка методом Хэмминга
        elif user_select == 2:
            decode()

        # Выход
        elif user_select == 3:
            exit(0)

# ###################### ЗАПУСК ПРОГРАММЫ ######################

if __name__ == "__main__":
    main()
```

Фрагмент кода 1, Интерфейс пользователя с выбором работы программы.

2.1) Шифрование

```
# Функция заполнения матрицы контрольного суммирования (сверху вниз, слева направо)
def completed_checksum_matrix(checksum_matrix):
    for i in range(13):
        sequence_length = i + 1
        for j in range(4):
            checksum_matrix[j][i] = sequence_length % 2
            sequence_length //= 2
```

Фрагмент кода 2, Заполнение матрица контрольного суммирования

```python
# ##################### ОБЩАЯ ЧАСТЬ – ШИФРОВАНИЕ И РАСШИФРОВАНИЕ #####################

# Получение данных. Заполнение матрицы
def get_data_fill_matrix(sequence_length):
    # Получение данных. Запись в массив. Преобразование элементов массива к типу int
    try:
        original_sequence = list(input(f"Введите последовательность длинной {sequence_length}: "))
        original_sequence = list(map(int, original_sequence))
    except:
        print("Ошибка в вводе данных!")
        exit(-1)

    # заготовка (для разработки и тестирования)
    if original_sequence[0] == 1 and len(original_sequence) == 1:
        print("Использован заготовленный вариант: 101101010")
        original_sequence = [1, 0, 1, 1, 0, 1, 0, 1, 0]

    if original_sequence[0] == 2 and len(original_sequence) == 1:
        print("Использован заготовленный вариант: 0011111001010")
        original_sequence = [0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0]


    # Проверка на корректность введённой последовательности
    for i in original_sequence:
        if (i != 0 and i != 1) or len(original_sequence) != sequence_length:
            print("Ошибка в вводе данных!\n")
            exit(-1)

    # Создание матрицы контрольного суммирования (4 столбца, 13 строк, заполнение "0")
    checksum_matrix = []
    for i in range(4):
        checksum_matrix.append([])
        for j in range(13):
            checksum_matrix[i].append(0)

    # Функция "заполнить матрицу контрольного суммирования"
    completed_checksum_matrix(checksum_matrix)

    return original_sequence, checksum_matrix
```

Фрагмент кода 3, Получение последовательности от пользователя. Подготовка и заполнение матрицы контрольного суммирования нулями.

```python
# ##################### ШИФРОВАНИЕ #####################

# Функция кодирования бинарной последовательности данных
def encode():
    sequence_length = 9 # Вариант 1

    # ---------------------------------------------------------------
    # Получение данных. Заполнение матрицы
    original_sequence, checksum_matrix = get_data_fill_matrix(sequence_length)
    # ---------------------------------------------------------------

    # Добавляем места, куда потом вставим контрольные битов
    result_vector = []
    pos = 0
    for i in range(13):
        k = math.log2(i+1) % 1  # вычисление степень двойки
        if k == 0:
            result_vector.append(0)
        else:
            result_vector.append(original_sequence[pos]) # вставляем нули на место куда подставятся числа
            pos += 1

    # Нахождение вектора контрольных сумм (St)
    checksum_vector = []
    for i in range(4):
        summ = 0
        for j in range(13):
            summ += result_vector[j] * checksum_matrix[i][j] # (побитовое и) и подсчёт единиц
        checksum_vector.append(summ % 2) # чётность → 0 / 1

    # Получаем кодовое слово, вставляя биты вектора контрольных сумм
    pos = 0
    for i in range(13):
    # подставляем в степени двойки
        if math.log2(i+1) % 1 == 0:
            result_vector[i] = checksum_vector[pos]
            pos += 1

    # Печать результата пользователю
    print_data_encode(original_sequence, checksum_vector, result_vector, checksum_matrix)
```

Фрагмент кода 4, Получение кодового слова.

```python
# Функция вывода данных ШИФРОВАНИЕ
def print_data_encode(original_sequence, checksum_vector, result_vector, checksum_matrix):
    print("Матрица контрольного суммирования:")
    print("  i  [*, *, 1, *, 2, 3, 4, *, 5, 6, 7, 8, 9]")
    print("  j  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3]")

    # Печатаем матрицу контрольного суммирования (0, 1, 0, ...)
    c = 0 # столбцы
    for i in checksum_matrix:
        print(f"a[{c + 1}] {i}")
        c += 1

    # Исходная последовательность
    print(f"  b  [_, _, {original_sequence[0]}, _, {original_sequence[1]}, {original_sequence[2]},\
        {original_sequence[3]}, _, {original_sequence[4]}, {original_sequence[5]}, {original_sequence[6]}, \
            {original_sequence[7]}, {original_sequence[8]}]")

    #  Биты резервных позиций
    print(f"beta [{checksum_vector[0]}, {checksum_vector[1]}, _, {checksum_vector[2]}, _, \
        _, _, {checksum_vector[3]}, _, _, _, _, _]")

    # Получившийся вектор
    print(f"b[t] {result_vector}")
```

Фрагмент кода 5, Вывод результата пользователю

```
КОНСОЛЬ ОТЛАДКИ   ПРОБЛЕМЫ   ТЕРМИНАЛ

(venv) PS C:\Users\nikit\3D Objects\petrsu_practice\information_security\lab_2> python .\hamming.py
Выберите режим:
(1): Кодирование методом Хэмминга
(2): Проверка методом Хэмминга
(3): Выход
Ваш выбор: 1
Введите последовательность длинной 9: 1
Использован заготовленный вариант: 101101010
Матрица контрольного суммирования:
  i  [*, *, 1, *, 2, 3, 4, *, 5, 6, 7, 8, 9]
  j  [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3]
a[1] [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
a[2] [0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0]
a[3] [0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1]
a[4] [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
  b  [_, _, 1, _, 0, 1, 1, _, 0, 1, 0, 1, 0]
beta [0, 0, _, 1, _, _, _, 0, _, _, _, _, _]
b[t] [0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0]
Выберите режим:
(1): Кодирование методом Хэмминга
(2): Проверка методом Хэмминга
(3): Выход
Ваш выбор:
```

Фрагмент кода 6, Демонстрация работы в терминале

2.2) Расшифрование

```python
# Функция заполнения матрицы контрольного суммирования (сверху вниз, слева направо)
def completed_checksum_matrix(checksum_matrix):
    for i in range(13):
        sequence_length = i + 1
        for j in range(4):
            checksum_matrix[j][i] = sequence_length % 2
            sequence_length //= 2
```

Фрагмент кода 7, Заполнение матрица контрольного суммирования

```python
# ##################### ОБЩАЯ ЧАСТЬ – ШИФРОВАНИЕ И РАСШИФРОВАНИЕ #####################

# Получение данных. Заполнение матрицы
def get_data_fill_matrix(sequence_length):
    # Получение данных. Запись в массив. Преобразование элементов массива к типу int
    try:
        original_sequence = list(input(f"Введите последовательность длинной {sequence_length}: "))
        original_sequence = list(map(int, original_sequence))
    except:
        print("Ошибка в вводе данных!")
        exit(-1)

    # заготовка (для разработки и тестирования)
    if original_sequence[0] == 1 and len(original_sequence) == 1:
        print("Использован заготовленный вариант: 101101010")
        original_sequence = [1, 0, 1, 1, 0, 1, 0, 1, 0]

    if original_sequence[0] == 2 and len(original_sequence) == 1:
        print("Использован заготовленный вариант: 0011111001010")
        original_sequence = [0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0]


    # Проверка на корректность введённой последовательности
    for i in original_sequence:
        if (i != 0 and i != 1) or len(original_sequence) != sequence_length:
            print("Ошибка в вводе данных!\n")
            exit(-1)

    # Создание матрицы контрольного суммирования (4 столбца, 13 строк, заполнение "0")
    checksum_matrix = []
    for i in range(4):
        checksum_matrix.append([])
        for j in range(13):
            checksum_matrix[i].append(0)

    # Функция "заполнить матрицу контрольного суммирования"
    completed_checksum_matrix(checksum_matrix)

    return original_sequence, checksum_matrix
```

Фрагмент кода 8, Получение последовательности от пользователя. Подготовка и заполнение матрицы контрольного суммирования нулями.

```python
# ##################### РАСШИФРОВКА #####################

# Функция декодирования бинарной последовательности данных
def decode():
    sequence_length = 13

    # ----------------------------------------------------------------
    # Получение данных. Заполнение матрицы
    original_sequence, checksum_matrix = get_data_fill_matrix(sequence_length)
    # ----------------------------------------------------------------

    # Нахождение вектора контрольных сумм (St)
    checksum_vector = []
    for i in range(4):
        summ = 0
        for j in range(13):
            summ += original_sequence[j] * checksum_matrix[i][j] # (побитовое и) и подсчёт единиц
        checksum_vector.append(summ % 2) # чётность → 0 / 1

    # Сумма битых ячеек
    place_error = 0
    for i in range(4):
        place_error += checksum_vector[i] * 2**i

    # Печать результата пользователю
    print_data_decode(checksum_vector, place_error)
```

Фрагмент кода 9, Поиск позиции ошибки, если она есть

```python
# Функция вывода данных РАСШИФРОВКА
def print_data_decode(checksum_vector, place_error):
    #  Биты резервных позиций
    print(f"s[r] {checksum_vector}")

    #  Вывод ошибки
    if place_error != 0:
        print(f"Ошибка в {place_error} бите")
    else:
        print("Нет ошибок!")
```

Фрагмент кода 10, отчёт об ошибке пользователю

```
(venv) PS C:\Users\nikit\3D Objects\petrsu_practice\information_security\lab_2> python .\hamming.py
Выберите режим:
(1): Кодирование методом Хэмминга
(2): Проверка методом Хэмминга
(3): Выход
Ваш выбор: 2
Введите последовательность длинной 13: 2
Использован заготовленный вариант: 0011111001010
s[r] [1, 0, 1, 0]
Ошибка в 5 бите
Выберите режим:
(1): Кодирование методом Хэмминга
(2): Проверка методом Хэмминга
(3): Выход
Ваш выбор:
```

Фрагмент кода 11, Демонстрация работы в терминале

Материалы:

Код Хэмминга // YouTube Artemy URL: https://www.youtube.com/watch?v=ehuNcmE8S84 (дата обращения: 05.12.2022).

Hamming Error-Correcting Code // dCode URL: https://www.dcode.fr/hamming-error-correction (дата обращения: 05.12.2022).

Проверить, является ли заданное натуральное число степенью двойки // cyberforum URL: https://www.cyberforum.ru/python-beginners/thread2238705.html (дата обращения: 05.12.2022).

vilisov/hamming_code.py // GitHub Gist URL: https://gist.github.com/vilisov/8278ad08700c89afde28 (дата обращения: 05.12.2022).