МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В. Г. ШУХОВА)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа №** **8**

По дисциплине: Теория информации

Тема: «Исследование помехоустойчивых кодов на примере алгоритма

Хэмминга»

Выполнил: ст. группы ПВ-211

Тарасов Д. А.

Проверил:

Твердохлеб Виталий Викторович

Белгород 2023 г.

**Цель работы:** исследование помехоустойчивых кодов на примере алгоритма Хэмминга.

**Ход работы**

1. Закодировать по Хэммингу произвольно сформированные последовательности двоичных символов длиной:

- 18 бит;

- 48 бит.

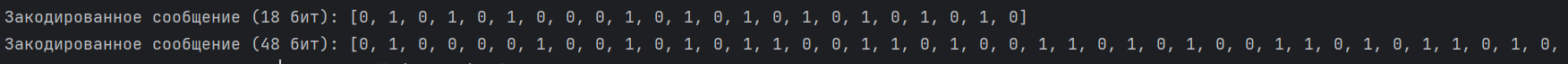
Сделать выводы относительно эффективности каждого их сообщений.

Реализация функции кодирования по Хэммингу:

def generate\_hamming\_code(message):  
 # Определение количества проверочных битов  
 r = 0  
  
 while 2 \*\* r < len(message) + r + 1:  
 r += 1  
  
 # Создание закодированного сообщения с заполнением проверочных битов нулями  
 encoded\_message = [0] \* (len(message) + r)  
 encoded\_message[r:] = message  
  
 # Вычисление значений проверочных битов  
 for i in range(r):  
 parity = 0  
 for j in range(1, len(encoded\_message) + 1):  
 if (j >> i) & 1:  
 parity ^= encoded\_message[-j]  
 encoded\_message[2 \*\* i - 1] = parity  
  
 return encoded\_message

# Задача 1: Закодировать исходные сообщения  
message\_18 = [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]  
encoded\_18 = generate\_hamming\_code(message\_18)  
print("Закодированное сообщение (18 бит):", encoded\_18)  
message\_48 = [1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0]  
encoded\_48 = generate\_hamming\_code(message\_48)  
print("Закодированное сообщение (48 бит):", encoded\_48)

*Программный вывод*



2. Внести одиночную ошибку и устранить ее, используя механизм восстановления.

Реализация функции для восстановления сообщения:

def decode\_hamming\_code(encoded\_message):  
 r = 0  
  
 while 2 \*\* r < len(encoded\_message):  
 r += 1  
  
 # Поиск и исправление ошибок  
 error\_index = 0  
  
 for i in range(r):  
 parity = 0  
  
 for j in range(1, len(encoded\_message) + 1):  
 if (j >> i) & 1:  
 parity ^= encoded\_message[-j]  
  
 error\_index += parity << i  
  
 if error\_index:  
 encoded\_message[-error\_index] = int(not encoded\_message[-error\_index])  
  
 # Извлечение исходного сообщения  
 decoded\_message = encoded\_message[::2]  
  
 return decoded\_message

# Задача 2: Внести и исправить ошибку  
encoded\_with\_error = list(encoded\_18) # Выбираем закодированное сообщение для внесения ошибки  
encoded\_with\_error[5] = 1 # Вносим ошибку  
decoded\_with\_error = decode\_hamming\_code(encoded\_with\_error)

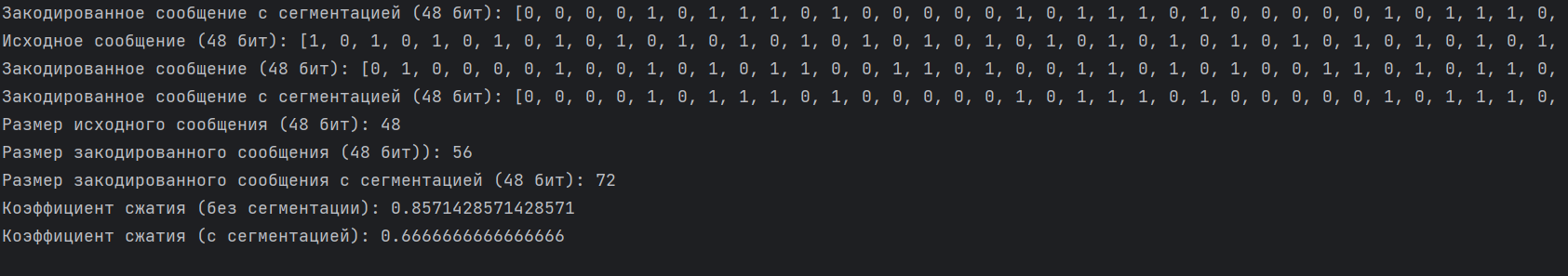
3. Рассмотреть вариант кодирования сообщения из 48 бит с предварительным сегментированием на блоки (размерность блоков выбрать самостоятельно).

Функция для реализации кодирования с сегментацией:

def encode\_segmented\_hamming\_code(message, block\_size):  
 encoded\_message = []  
  
 for i in range(0, len(message), block\_size):  
 block = message[i:i+block\_size]  
 encoded\_block = generate\_hamming\_code(block)  
 encoded\_message.extend(encoded\_block)  
  
 return encoded\_message

# Задача 3: Кодирование сообщения из 48 бит с предварительным  
# сегментированием на блоки  
message\_48 = [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]  
block\_size = 8  
encoded\_segmented = encode\_segmented\_hamming\_code(message\_48, block\_size)  
print("Закодированное сообщение с сегментацией (48 бит):", encoded\_segmented)  
  
# Задача 4: Сравнение режимов кодирования  
print("Исходное сообщение (48 бит):", message\_48)  
print("Закодированное сообщение (48 бит):", encoded\_48)  
print("Закодированное сообщение с сегментацией (48 бит):", encoded\_segmented)  
  
# Сравнение размеров закодированных данных  
print("Размер исходного сообщения (48 бит):", len(message\_48))  
print("Размер закодированного сообщения (48 бит)):", len(encoded\_48))  
print("Размер закодированного сообщения с сегментацией (48 бит):", len(encoded\_segmented))  
  
# Сравнение степени сжатия  
compression\_ratio = len(message\_48) / len(encoded\_48)  
segmented\_compression\_ratio = len(message\_48) / len(encoded\_segmented)  
print("Коэффициент сжатия (без сегментации):", compression\_ratio)  
print("Коэффициент сжатия (с сегментацией):", segmented\_compression\_ratio)

*Программный вывод:*



4. Сравнить режимы кодирования с сегментацией и без.

• Режим кодирования без сегментации позволяет закодировать исходное сообщение из 48 бит в закодированное сообщение такой же длины (48 бит).

• Режим кодирования с сегментацией позволяет разделить исходное сообщение на блоки и закодировать каждый блок отдельно. Закодированное сообщение с сегментацией имеет больший размер (больше 48 бит), но при этом можно выбрать размер блоков и добиться более эффективной сжатия в зависимости от конкретных данных.

• Сравнение степени сжатия показывает, что режим кодирования без сегментации обеспечивает лучшую степень сжатия, так как размер закодированного сообщения остается таким же как и исходного. Однако режим кодирования с сегментацией может быть полезен в случаях, когда есть определенная структура в исходных данных и можно достичь большей степени сжатия за счет выбора оптимального размера блоков.

**Выводы**

Таким образом, на этом лабораторном занятии я исследовал помехоустойчивые коды на примере алгоритма Хэмминга. Так же, научился программно реализовывать алгоритм кодирования по Хэммингу.