Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования   
**Национальный исследовательский университет «МЭИ»**

**Лабораторная работа №6**

по курсу«**Методы и средства защиты информации**»

**«Кодирование методом Хэмминга»**

Выполнил:

студент группы А-07-18

Востриков Р.В.

Вариант 8

Москва  
2020

**1. Задание**

Задание 1

а) Закодировать по методу Хэмминга число 11. В полученный код ввести одиночную ошибку и определить номер искаженного разряда.

б) Пусть 11010011 и 11001111 искаженные слова расширенного кода Хемминга длины 8. Какое из этих слов содержит одиночную ошибку, а какое – двойную ошибку? В случае одиночной ошибки определить ее местоположение.

Задание 2  
Составить программу для кодирования методом Хэмминга двоичного слова длины M и определения номера искаженного разряда.

**2. Описание результатов**

Задание 1а)  
Представим число 11 в двоичной системе счисления: 1110 = 10112.Т.к. сообщение содержит 4 информационных бита, нужно добавить 3 контрольных бита на позиции 1, 2, 4.  
Получаем сообщение: k1k21k3011, где k1, k2, k3 – контрольные биты.  
k1 = (a3 + a5 + a7) mod 2 = (1 + 0 + 1) mod 2 = 0  
k2 = (a3 + a6 + a7) mod 2 = (1 + 1 + 1) mod 2 = 1  
k3 = (a5 + a6 + a7) mod 2 = (0 + 1 + 1) mod 2 = 0  
Итоговое сообщение, закодированное методом Хэмминга: 0110011.

Пусть ошибка будет в 6-м бите, тогда сообщение примет вид: 0110001.  
Определим местоположение ошибки, проверяя каждый контрольный бит.  
s1 = (a1 + a3 + a5 + a7) mod 2 = (0 + 1 + 0 + 1) mod 2 = 0 +  
s2 = (a2 + a3 + a6 + a7) mod 2 = (1 + 1 + 0 + 1) mod 2 = 1 -  
s3 = (a4 + a5 + a6 + a7) mod 2 = (0 + 0 + 0 + 1) mod 2 = 1 -  
Контрольные биты, стоящие на позициях 2 и 4 нечётны, отсюда бит ошибки - (2 + 4) = 6.

Задание 1б)  
Проверим слово 11010011.  
s = (a1 + a2 + a3 + a4 + a5 + a6 + a7 + a8) mod 2 = (1 + 1 + 0 + 1 + 0 + 0 + 1 + 1) mod 2 = 1  
=> В слове допущена одна ошибка. Найдём её.  
s1 = (a1 + a3 + a5 + a7) mod 2 = (1 + 0 + 0 + 1) mod 2 = 0 +  
s2 = (a2 + a3 + a6 + a7) mod 2 = (1 + 0 + 0 + 1) mod 2 = 0 +  
s3 = (a4 + a5 + a6 + a7) mod 2 = (1 + 0 + 0 + 1) mod 2 = 0 +  
Значит, ошибка произошла в a8.

Проверим слово 11001111.  
s = (a1 + a2 + a3 + a4 + a5 + a6 + a7 + a8) mod 2 = (1 + 1 + 0 + 0 + 1 + 1 + 1 + 1) mod 2 = 0  
s1 = (a1 + a3 + a5 + a7) mod 2 = (1 + 0 + 1 + 1) mod 2 = 1 -  
s2 = (a2 + a3 + a6 + a7) mod 2 = (1 + 0 + 1 + 1) mod 2 = 1 -  
s3 = (a4 + a5 + a6 + a7) mod 2 = (0 + 1 + 1 + 1) mod 2 = 1 -  
=> В слове допущено 2 ошибки, нельзя определить их местоположение.

Задание 2

Была написана программа на языке C#, позволяющая кодировать сообщения произвольной длины методом Хэмминга. Вначале вычисляется количество контрольных битов, исходя из длины передаваемого сообщения. Далее вычисляются позиции, на которых будут стоять контрольные биты. Создаётся новое слово, в которое записывается исходное сообщение, пропуская позиции для контрольных битов. Затем вычисляются сами контрольные биты. Таким образом получается закодированное сообщение.

**3. Код программы (к заданию 2)**

# определение количества контрольных битов для сообщения длины M

def parity\_count(M):

from math import ceil, floor, log2

n = (M << 1) if is\_power\_of\_2(M) else 2 \*\* ceil(log2(M)) # n - наименьшая степень 2, превосходящая M

lower\_bin = floor(log2(n))

upper\_bin = lower\_bin + 1

data\_bit\_boundary = n - lower\_bin - 1

return lower\_bin if M <= data\_bit\_boundary else upper\_bin

# определение битов данных, соответствующих контрольному биту в указанной позиции

def data\_bits(parity, datalen):

if is\_power\_of\_2(parity):

data\_index = 1 # позиция бита в строке данных

code\_index = 3 # позиция бита в кодовом слове

indexes = [] # позиции битов в кодовом слове, которые контроллирует данный бит четности

while data\_index <= datalen:

curr\_bit\_is\_data = not is\_power\_of\_2(code\_index)

if curr\_bit\_is\_data and (code\_index % (parity << 1)) >= parity:

indexes.append(code\_index)

data\_index += curr\_bit\_is\_data

code\_index += 1

return np.array(indexes)

else:

raise ValueError("Контрольные биты могут находится только в позициях, равных степеням двойки")

# функция кодирования простым кодом Хемминга (без бита общей проверки четности)

def encode(msg):

length = len(msg)

parity = parity\_count(length)

length += parity # итоговая длина кода

data = np.array(msg) # информационные биты

powers = get\_powers\_2(parity) # позиции контрольных битов (считая с 1)

code = np.zeros(length, dtype = 'i8')

mask = np.full\_like(code, True, dtype ='bool') # маска битов данных в коде

mask[powers-1] = False

code[mask] = data # устанавливаем биты данных в коде

# устанавливаем контрольные биты

for p in powers:

bits = data\_bits(p, data.size) - 1 # позиции битов данных(считая с 0)

code[p-1] = code[bits].sum() % 2

return stringify(code)

# функция декодирования сообщений, закодированных простым кодом Хемминга (без бита общей проверки четности)

# функция возвращает сообщение и позицию ошибки (считая с 1, если ее не было - 0)

def decode(msg):

from math import floor, log2

length = len(msg)

parity = floor(log2(length)) + 1 # число проверочных битов в кодовом слове

powers = get\_powers\_2(parity) # позиции контрольных битов (считая с 1)

code = np.array(msg)

mask = np.full\_like(code, True, dtype ='bool') # маска битов данных в коде

mask[powers-1] = False

actual = code[np.invert(mask)] # полученные контрольные биты

# вычисляем контролные биты, соответствующие полученному сообщению

expected = np.zeros\_like(actual)

for i in range(actual.size):

bits = data\_bits(powers[i], length-parity) - 1 # позиции битов данных(считая с 0)

expected[i] = code[bits].sum() % 2

errors = np.where(expected != actual)[0] # несовпадающие контрольные биты

if errors.size != 0:

pos = np.sum(2 \*\* errors) # позиция бита ошибки в кодовом слове (считая с 1)

code[pos-1] = 0 if code[pos-1] == 1 else 1

return (stringify(code[mask]), pos)

else:

return (stringify(code[mask]), 0)

**4. Вывод**

В лабораторной работе было показано кодирование сообщений методом Хэмминга. Этот метод удобен тем, что позволяет исправлять единичные ошибки (расширенный код позволяет обнаруживать двойные). Главная особенность этого метода состоит в том, что контрольные биты ставятся на конкретные позиции, а не в конце сообщения, как, например, в прямоугольном коде.