МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность Информационные системы и технологии

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6 НА ТЕМУ:**

**Избыточное кодирование данных в информационных системах. Циклические коды**

Выполнила студентка 3 курса 1 группы

Пригодич Вера Валерьевна

Минск 2022

**Задание 1:** Задание выполняется по указанию преподавателя в соответствии с вариантом из табл. 6.2, из которого выбирается порождающий полином ЦК, а по значению соответствующего ему значения *r* – длина *k* информационного слова *Xk*. Полагаем, что каждый полином соответствует коду, обнаруживающему и исправляющему одиночные ошибки в кодовых словах. Определить параметры *(n, k)*-кода для своего варианта. Основой задания является разработка приложения.

Для выполнения задания был использован следующий код на языке *python*:

# Подсчитывает число информационных разрядов

def count\_k(sequence):

    if type(sequence) == int:

        sequence = bin(sequence)[2:]

    return len(sequence)

Листинг 1 – задание 1

**Задание 2:** Составить порождающую матрицу *(n, k)-*кода в соответствии с формулой (6.7), трансформировать ее в каноническую форму и далее – в проверочную матрицу канонической формы.

Для выполнения этого задания код, разработанный в предыдущем задании был дополнен следующим кодом:

# Составляет порождающую матрицу по полиному

def generate\_polynomial(g, n, k):

    g\_x = [int(x) for x in g]

    g\_x = [\*g\_x, \* np.zeros(n-len(g\_x), dtype=int)]

    generating\_matrix = g\_x

    for j in range (k):

        buf = np.roll(g\_x, j, axis=0)

        if j > 0 :

            generating\_matrix = np.vstack((generating\_matrix, buf))

    return generating\_matrix

# Приводит порождающую матрицу к каноническому виду

def generate\_canonical(k, n, g):

    G = np.zeros([k, n], dtype=int)

    g\_x = [int(x) for x in g]

    for i in range(0, k):

        G[i][i] = 1

    for i in range(0, k):

        q, r = np.polydiv(G[i], g\_x)

        for k in range(0, len(r)):

            buff = (r[k] % 2)

            r[k] = buff

        new\_result = np.polyadd(G[i], r)

        G[i] = new\_result

    return G

# Составляет проверочную матрицу канонического вида

def generate\_check(canonical\_matrix, k, n):

    check\_matrix = []

    for i in range(k, n):

        check\_matrix.append(canonical\_matrix[:, i].transpose())

    i\_matrix = np.eye(k-1, dtype=int)

    return np.hstack((check\_matrix, i\_matrix))

Листинг 2 – задание 2

**Задание 3:** Используя порождающую матрицу ЦК, вычислить избыточные символы (слово *Xr*) кодового слова *Xn* и сформировать это кодовое слово.

Дополним разработанный в предыдущем задании код следующим образом:

#  Кодирует принципами циклического кода инф. последовательность

def encrypt\_with\_cyclic\_code(sequence, g, k, n):

    # 1. умножаем полином инф. посл-ти на x^r

    # фактически выполняем сдвиг влево

    shifted = sequence + '0' \* (n - k)

    print('\nx\_k\*x^r =', shifted)

     # 2. вычисляем остаток от деления получившегося полинома на g(x)

    mod = divide\_with\_xor(int(shifted, 2), int(g, 2))[1]

    print('x\_k\*x^r mod (with xor) g(x) =', bin(mod)[2:])

    # 3. конкатенация

    mod\_len = len(bin(mod)[2:])

    return sequence + '0' \* (n - k - mod\_len) + bin(mod)[2:]

Листинг 3 – задание 3

**Задание 4:** Принять кодовое слово *Yn* со следующим числом ошибок: 0; 1; 2. Позиция ошибки определяется (генерируется) случайным образом.

Дополним разработанный в предыдущем задании код следующей функцией:

# Позиция ошибки определяется (генерируется) случайным образом.

def rand\_mistake(encoded, num):

    y = [int(x) for x in encoded]

    for n in range (num):

        i = random.randint(0, len(encoded)-1)

        if (y[i] == 0):

            y[i] = 1

        else:

            y[i] = 0

    y = ''.join(map(str, y))

    return y

Листинг 4 – задание 4

**Задание 5:** Для полученного слова *Yn* вычислить и проанализировать синдром. В случае, если анализ синдрома показал, что информационное сообщение было передано с ошибкой (или 2 ошибками), сгенерировать унарный вектор ошибки *Еn = е1, е2, …, еn* и исправить одиночную ошибку, используя выражение (6.5); проанализировать ситуацию при возникновении ошибки в 2 битах.

Для выполнения данного задания дополним скрипт следующей функцией:

def count\_syndrome(y\_n, g):

    return bin(divide\_with\_xor(int(y\_n, 2), int(g, 2))[1])[2:].rjust(5, '0')

def error\_position(syndrome, check\_matrix, n):

    error\_vector = np.zeros(n, dtype=int)

    for i in range(n):

        if (''.join(map(str, check\_matrix[:, i].transpose())) == syndrome):

            error\_vector[i] = 1

    error\_vector = ''.join(map(str, error\_vector))

    return error\_vector

def xor(a: str, b: str) -> str:

    delta\_length = abs(len(a) - len(b))

    if len(a) > len(b):

        for i in range(delta\_length):

            b = '0' + b

    elif len(b) > len(a):

        for i in range(delta\_length):

            a = '0' + a

    result = ""

    for i in range(len(a)):

        if a[i] == b[i]:

            result += '0'

        else:

            result += '1'

    return result

Листинг 5 – задание 5

Результат работы итогового кода:

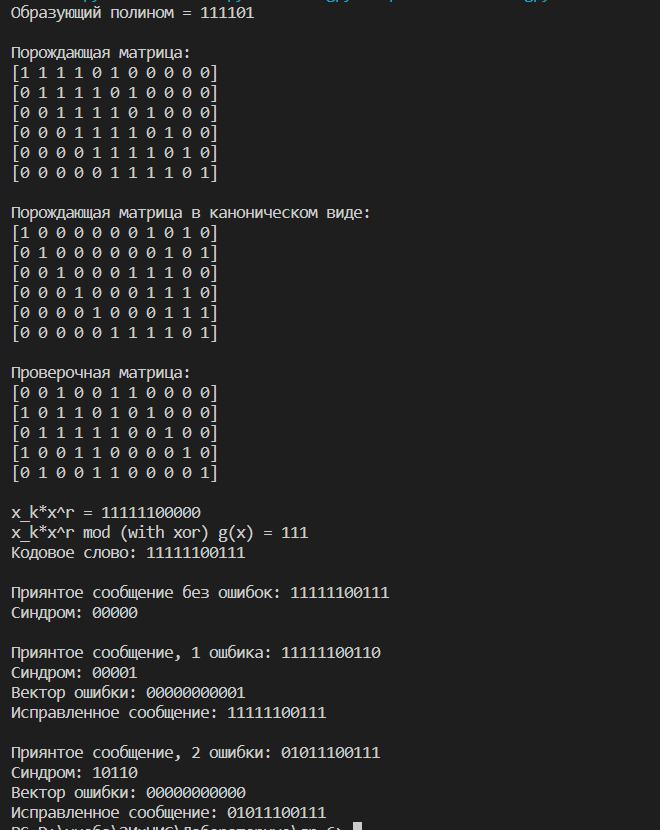


Рисунок 1 ­­­­– Результат работы программы

Данный код гарантирует исправление только 1 ошибки.

**Вывод:** в ходе лабораторной работы были приобретены практические навыки кодирования/декодирования двоичных данных при использовании циклических кодов.