Отчет по заданию №6

«ЦИКЛИЧЕСКИЙ КОД»

по курсу «Теория информации и кодирования»

студента 3 курса группы ИВТ

Направления подготовки 09.03.01«Информатика и вычислительная техника»

, 2022

Вариант 4.

Техническое задание:

Источник информации вырабатывает сообщения, содержащие k информационных разрядов. Значения разрядов генерируются в двоичной системе счисления счетчиком случайных чисел. Необходимо:

1. разработать программное обеспечение для передатчика, которое будет строить циклический код, позволяющий обнаруживать и исправлять все однократные ошибки;
2. разработать программное обеспечение на приемной стороне, позволяющее обрабатывать принятый циклический код и определять позицию ошибки;
3. провести комплекс численных экспериментов, в ходе которых продемонстрировать работу системы «передатчик-приемник» с использованием циклического кода.

Описание работы программы:

Циклические коды получили довольно широкое применение благодаря их эффективности при обнаружении и исправлении ошибок. Название кодов произошло от их свойства, заключающегося в том, что каждая кодовая комбинация может быть получена путем циклической перестановки символов комбинации, принадлежащей к этому же коду. Если комбинация a0a1a2…an-1 является разрешенной комбинацией циклического кода, то комбинация an1a0a1a2…an-2 также принадлежит этому коду. Циклические коды удобно рассматривать, представляя комбинацию двоичного кода не в виде последовательностей нулей и единиц, а в виде полинома от фиктивной переменной x:

$$G(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$

где ai - цифры данной системы счисления (в двоичной системе 0 и 1).

При построении циклического кода вначале определяется число информационных разрядов k по заданному объему кода. Затем находится наименьшая длина кодовых комбинаций n, обеспечивающая обнаружение или исправление ошибок заданной кратности. Значность (длина) кодовой комбинации (n), которая позволят обнаруживать и исправлять все однократные ошибки, определяется следующим выражением:

$$2^k \le \frac{2^n}{1+n}$$

Рис. 1 Условие, определяющее длину комбинации
и количество проверочных бит

Количество проверочных разрядов p=n-k. Для построения циклического кода используется образующий полином P(x). Образующий полином P(x) должен входить в качестве сомножителя в разложение двучлена (xn+1), являться неприводимым полиномом степени p.

Для получения циклического кода используется следующий алгоритм. Исходная кодовая комбинация k-значного кода, которой соответствует полином G(x), умножается на одночлен x^p. Полученный многочлен делится на образующий полином P(x) степени p. В результате умножения G(x) на x^p степень каждого одночлена, входящего в G(x), повысится на p. При делении произведения $x^pG(x)$ на образующий полином P(x) получится частное Q(x) такой же степени, как и G(x).

Таким образом, алгоритм построения циклического кода сводится к действию над полиномами (многочленами). При этом сложение двоичных многочленов сводится к сложению по модулю два коэффициентов при равных степенях переменной x; умножение производится по обычному правилу перемножения степенных функций, однако полученные при этом коэффициенты при равных степенях переменной x складываются по модулю два; деление осуществляется по правилам деления степенных функций, при этом операции вычитания заменяются операциями суммирования по модулю два. Результат умножения и деления можно представить в следующем виде:

$$\frac{x^pG(x)}{P(x)} = Q(x) + \frac{R(x)}{P(x)}$$

где R(x) - остаток от деления $x^pG(x)$ на P(x).

Умножая обе части равенства на P(x) и произведя некоторые перестановки, получим полином, соответствующий циклическому коду:

$$F(x) = Q(x)P(x) = x^pG(x) + R(x)$$

В правой части знак минус перед R(x) заменен знаком плюс, так как вычитание по модулю два сводится к сложению.

Программа разработана на языке программирования Python 3 и позволяет проводить эксперименты с данными любой длины (по умолчанию 58 бит как по заданию в Варианте 4). Программа выполнена в виде библиотеки (без графического интерфейса) с тестирующим модулем. Может подключаться и использоваться в других проектах.

Длина кодовой комбинации вычисляется при помощи функции на Рис. 2. Фунция может вычислять как общую длину кодового сообщения (по длине информационного сообщения k), так и наоборот при необходимости.

```
28  # Возвращает параметры сообщения [k, n, p] по одному из параметров k или n
29  def getKN(x, xIsN=True):
30      #   - xIsN=False, то передали k=>   k=x, идет подбор n от x+1, шаг 1
31      #   - xIsN=True,  то передали n=>   n=x, идет подбор k от x-1, шаг -1
32      x = ([x - 1, x, -1] if xIsN else [x, x + 1, 1]) + [int(not xIsN)]
33      while 2**x[0] > 2**x[1] / ( x[1] + 1 ): x[x[3]] += x[2]
34      return x[:2] + [ x[1] - x[0] ] # =>[k, n, p]
```

Рис. 2 Вычисление параметров кодовой комбинации

```
gen_arr  = lambda k:    [random.randint(0,1) for i in range(k)] # сгнерировать
```

Рис. 3 Генерация кодовой комбинации

Операция деления произведения $x^pG(x)$ на образующий полином P(x) представлена на Рис.4.

```
# Деление полинома полного (n) сообщения на неприводимый полином степени p
def div2(a):                    # возвращает синдром ошибки
    p = getKN(len(a))[2]        # порядок полинома p вычисляется по его длине
    pL, b = len(poli[p]), cut(a[:]) # длина неприводимого полинома, убираем нули
    while len(b) >= pL:
        b = cut([ int(b[i] != poli[p][i]) for i in range(pL) ] + b[pL:])
    return b
```

Рис. 4 деления произведения $x^pG(x)$ на образующий полином P(x)

Массив с неприводимыми полиномами степени p представлен на Рис.5.

```
poli = [[], #8 7 6 5 4 3 2 1 0     p    # Неприводимые полиномы степени p
            [             1,1], # 1    x + 1
            [           1,1,1], # 2    x^2 + x + 1
            [         1,0,1,1], # 3    x^3 + x + 1
            [       1,0,0,1,1], # 4    x^4 + x + 1
            [     1,0,0,1,0,1], # 5    x^5 + x^2 + 1
            [   1,0,0,0,0,1,1], # 6    x^6 + x + 1
            [ 1,0,0,0,0,0,1,1], # 7    x^7 + x + 1
            [1,0,0,0,1,1,0,1,1]] # 8   x^8 + x^4 + x^3 + x + 1
```

Рис. 5 Неприводимые полиномы степени p

Полиномы в массиве хранятся в бинарном виде, поэтому для вывода на экран используется функция их преобразования в текстовый вид.

```
def poliToStr(p):           # Отображение бинарного вида полинома в str формате
    pL = len(poli[p])
    return " + ".join(reversed(
    [( ("1" if i == pL-1 else "x") if i >= pL-2 else ("x^" + str(pL - 1 -i)) )
    for i in range(pL-1, -1, -1)   if poli[p][i] ]))
```

Рис. 6 Функция отображения бинарной записи полинома в текстовом виде

Для установления соответствия синдрома ошибки номеру бита, в котором она произошла, формируется проверочная матрица, где для каждого бита определяется соответствующий ему синдром ошибки.

```
# Формирование проверочной матрицы соответствия синдромов ошибок для разных бит
# для длины сообщения n и неприводимого полинома степени p (вычисляется по n)
getCheckSindromes = lambda n: [div2( [0]*i+[1]+[0]*(n-i-1) ) for i in range(n)]
```

Рис. 7 Формирование проверочной матрицы соответствия синдромов ошибок для разных бит

```
def Line_With_Errors(pass_block, Er_prob = 0.2, t = 1):
    # длина блока, и индексы битов с ошибками
    block_len, error_bit_list = len(pass_block), []
    # коррекция, если вероятность ошибки > 100%
    if Er_prob > 1.0 or Er_prob < 0.0: Er_prob = 1.0
    print("\n", "-"*70, "\nГенерация "+ ("1 ошибки" if t==1 else "2 ошибок") +\
      " в линии с помехами (", Er_prob*100,\
      "%). Получено для передачи:\n", dump_arr(pass_block), sep="")
    # коррекция, если ошибочных бит больше, чем передаваемых
    if t > block_len: t = block_len
    # Формируем список индексов битов, где может по вероятности Er_prob произойти
    # ошибка (т.е. бит инвертируется)
    err_bits_idxs = random.sample(list(range(block_len)), t)
    # Проходим по списку выбранных для ошибки бит, применяем вероятность,
    # инвертируем если надо, добавляем в итоговый список ошибочно переданных бит
    for i in err_bits_idxs:
        if random.uniform(0.0, 1.0) <= Er_prob:
            pass_block[i] = 0 if pass_block[i] else 1
            error_bit_list.append(i+1)
    print("\nПозиции (!не индексы!) ошибочно переданных бит: ", error_bit_list)
    print("Измененные данные:\n", dump_arr(pass_block), sep="")
    print(pointers(error_bit_list))
    return sorted(error_bit_list)
```

Рис. 8 Функция имитирующая работу линии с помехой (генерирует одну ошибку в сообщении)

Программа генерирует случайную битовую комбинацию, определяет [k, n, p], кодирует сообщение циклическим кодом, пропускае полученный результат через модуль генерации ошибок, находит позицию ошибки и исправляет её.

```
k, n, p = getKN(k, xIsN=False)
Dmin = 2 * 1 + 1 # 2*t+1 t=1
print("Биты:: Информационные k=", k, ", проверочные p=", p, ", всего n=", \
  n, ". Dmin=", Dmin, sep="")

arr = gen_arr(k)
print("\n", "-"*70, "\nПередатчик:\n\nСгенерированное случайным образом"+\
  " информационное сообщение:\n", dump_arr(arr), sep="")

pCode = div2(arr + [0]*p)
arr = arr + [0]*(p-len(pCode)) + pCode

print("Закодированное циклическим кодом (полином: ", poliToStr(p), \
") сообщение:\n", dump_arr(arr), "\n", "-"*70, "\nПриемник:", sep="")
errorBits = Line_With_Errors(arr, Er_prob = 1.0, t = 1)

errorSindrome = div2(arr)
if errorSindrome:
    errorBitIndex = getCheckSindromes(n).index(errorSindrome)
    arr[errorBitIndex] = 0 if arr[errorBitIndex] else 1
    print("\nПроизошла ошибка в бите №",errorBitIndex + 1,". Исправленные "\
    + "данные:\n", dump_arr(arr),"\n", pointers([errorBitIndex + 1]),sep="")
else: print("Передача прошла без ошибок")

print("Раскодированное информационное сообщение:\n",dump_arr(arr[:k]),sep="")
```

Рис. 9 Фрагмент модуля с описанной выше последовательностью действий

В Приложении 2 приведена серия результатов вычисления полной цепочки действий по использованию циклического кода (генерация, кодирование, добавление ошибки, декодирование, исправление ошибки) для информационных сообщений 58, 90 и 100 бит.

Вывод:

В ходе выполнения лабораторной работы мы познакомились на практике с алгоритмом создания и применения циклического кода для передачи информации по помехонеустойчивым линиям, научились исправлять одиночную ошибку, рассчитывать количество дополнительных контрольных бит и их позиции в информационном сообщении, D min.

На языке программирования Python 3 был выполнен библиотечный программный модуль, генерирующий случайную последовательность бит, кодирующий её при помощи циклического кода.

Были проведены тесты при исходном количестве информационных бит – 58 и более (90, 100). Все тесты прошли успешно, что позволяет говорить, об эффективности данного алгоритма защиты от помех при передаче сообщений. При небольшом увеличении длины передаваемого сообщения существенно повышается помехоустойчивость передачи. Циклические сдвиги не нарушают работоспособность алгоритма.

Чем больше длина информационного сообщения в битах, тем меньший процент дополнительного трафика (в сравнении с длиной сообщения) приходится передавать дополнительно в виде проверочного кода. При малых длинах сообщений такая технология сильно теряет свой смысл, так как количество добавляемых бит проверочного кода сопоставимо по длине с передаваемым информационным сообщением и проще передать сообщение повторно, однако циклический код просто незаменим в ситуации, когда повторная передача информации невозможна.

Приложение 1.

Результаты работы программы.

```
Лабораторная работа №6 ("ЦИКЛИЧЕСКИЙ КОД"). Подготовил: [redacted]

Введите количество информационных бит k[2; +∞):
 Установлено значение по умолчанию для Варианта №4: k=58 бит
Биты:: Информационные k=58, проверочные p=7, всего n=65. Dmin=3

------------------------------------------------------------------
Передатчик:

Сгенерированное случайным образом информационное сообщение:
00000101111001000010001000110101100111111011100100111111011
Закодированное циклическим кодом (полином: x^7 + x + 1) сообщение:
000001011110010000100010001101011001111101110010011111101111110111
------------------------------------------------------------------
Приемник:

------------------------------------------------------------------
Генерация 1 ошибки в линии с помехами (100.0%). Получено для передачи:
000001011110010000100010001101011001111101110010011111101111110111

Позиции (!не индексы!) ошибочно переданных бит:  [19]
Измененные данные:
000001011110010000000010001101011001111101110010011111101111110111
                  ^

Произошла ошибка в бите №19. Исправленные данные:
000001011110010000100010001101011001111101110010011111101111110111
                  ^
Раскодированное информационное сообщение:
00000101111001000010001000110101100111111011100100111111011
```

Рис. 10 Тест 1: 58 информационных бит

```
Лабораторная работа №6 ("ЦИКЛИЧЕСКИЙ КОД"). Подготовил:

Введите количество информационных бит k[2; +∞): 58
 Установлено значение по умолчанию для Варианта №4: k=58 бит
Биты:: Информационные k=58, проверочные p=7, всего n=65. Dmin=3

---------------------------------------------------------------------
Передатчик:

Сгенерированное случайным образом информационное сообщение:
1111101001001100011000111111101010001011001011010010000001
Закодированное циклическим кодом (полином: x^7 + x + 1) сообщение:
11111010010011000110001111111010100010110010110100100000010000110
---------------------------------------------------------------------
Приемник:

---------------------------------------------------------------------
Генерация 1 ошибки в линии с помехами (100.0%). Получено для передачи:
11111010010011000110001111111010100010110010110100100000010000110

Позиции (!не индексы!) ошибочно переданных бит:  [61]
Измененные данные:
11111010010011000110001111111010100010110010110100100000010010110
                                                             ^

Произошла ошибка в бите №61. Исправленные данные:
11111010010011000110001111111010100010110010110100100000010000110
                                                             ^
Раскодированное информационное сообщение:
1111101001001100011000111111101010001011001011010010000001
```

Рис. 11 Тест 2: 58 информационных бит

```
Лабораторная работа №6 ("ЦИКЛИЧЕСКИЙ КОД"). Подготовил:

Введите количество информационных бит k[2; +∞): 58
 Установлено значение по умолчанию для Варианта №4: k=58 бит
Биты:: Информационные k=58, проверочные p=7, всего n=65. Dmin=3

---------------------------------------------------------------------
Передатчик:

Сгенерированное случайным образом информационное сообщение:
1111100110110011110000111010000111101100110111110010001011
Закодированное циклическим кодом (полином: x^7 + x + 1) сообщение:
11111001101100111100001110100001111011001101111100100010110100110
---------------------------------------------------------------------
Приемник:

---------------------------------------------------------------------
Генерация 1 ошибки в линии с помехами (100.0%). Получено для передачи:
11111001101100111100001110100001111011001101111100100010110100110

Позиции (!не индексы!) ошибочно переданных бит:  [31]
Измененные данные:
11111001101100111100001110100011111011001101111100100010110100110
                               ^

Произошла ошибка в бите №31. Исправленные данные:
11111001101100111100001110100001111011001101111100100010110100110
                               ^
Раскодированное информационное сообщение:
1111100110110011110000111010000111101100110111110010001011
```

Рис. 12 Тест 3: 58 информационных бит

```
Лабораторная работа №6 ("ЦИКЛИЧЕСКИЙ КОД"). Подготовил:[redacted]

Введите количество информационных бит k[2; +∞): 100
Биты:: Информационные k=100, проверочные p=7, всего n=107. Dmin=3

----------------------------------------------------------------------
Передатчик:

Сгенерированное случайным образом информационное сообщение:
00001111100011000100110100100011100101011000110000111110011111111011110011010000011111110011110010111
Закодированное циклическим кодом (полином: x^7 + x + 1) сообщение:
00001111100011000100110100100011100101011000110000111110011111111011110011010000011111110011110010110100000
----------------------------------------------------------------------
Приемник:

----------------------------------------------------------------------
Генерация 1 ошибки в линии с помехами (100.0%). Получено для передачи:
00001111100011000100110100100011100101011000110000111110011111111011110011010000011111110011110010110100000

Позиции (!не индексы!) ошибочно переданных бит:  [64]
Измененные данные:
00001111100011000100110100100011100101011000110000111110011111101011110011010000011111110011110010110100000
                                                                ^

Произошла ошибка в бите №64. Исправленные данные:
00001111100011000100110100100011100101011000110000111110011111111011110011010000011111110011110010110100000
                                                                ^
Раскодированное информационное сообщение:
00001111100011000100110100100011100101011000110000111110011111111011110011010000011111110011110010111
```

Рис. 13 Тест 1: 100 информационных бит

```
Лабораторная работа №6 ("ЦИКЛИЧЕСКИЙ КОД"). Подготовил:[redacted]

Введите количество информационных бит k[2; +∞): 90
Биты:: Информационные k=90, проверочные p=7, всего n=97. Dmin=3

----------------------------------------------------------------------
Передатчик:

Сгенерированное случайным образом информационное сообщение:
111001000000000011000010101101101000010100010011010111011010001111001010110000110001101111
Закодированное циклическим кодом (полином: x^7 + x + 1) сообщение:
111001000000000011000010101101101000010100010011010111011010001111001010110000110001101111110010
----------------------------------------------------------------------
Приемник:

----------------------------------------------------------------------
Генерация 1 ошибки в линии с помехами (100.0%). Получено для передачи:
111001000000000011000010101101101000010100010011010111011010001111001010110000110001101111110010

Позиции (!не индексы!) ошибочно переданных бит:  [53]
Измененные данные:
111001000000000011000010101101101000010100010011010101011010001111001010110000110001101111110010
                                                    ^

Произошла ошибка в бите №53. Исправленные данные:
111001000000000011000010101101101000010100010011010111011010001111001010110000110001101111110010
                                                    ^
Раскодированное информационное сообщение:
111001000000000011000010101101101000010100010011010111011010001111001010110000110001101111
```

Рис. 14 Тест 1: 90 информационных бит

Приложение 2.
Полный листинг программы

```
#!/usr/bin/python
import random
poli = [[], #8 7 6 5 4 3 2 1 0     p    # Неприводимые полиномы степени p
          [                1,1], # 1    x + 1
          [              1,1,1], # 2    x^2 + x + 1
          [            1,0,1,1], # 3    x^3 + x + 1
          [          1,0,0,1,1], # 4    x^4 + x + 1
          [        1,0,0,1,0,1], # 5    x^5 + x^2 + 1
          [      1,0,0,0,0,1,1], # 6    x^6 + x + 1
          [    1,0,0,0,0,0,1,1], # 7    x^7 + x + 1
          [1,0,0,0,1,1,0,1,1]] # 8    x^8 + x^4 + x^3 + x + 1

def poliToStr(p):          # Отображение бинарного вида полинома в str формате
    pL = len(poli[p])
    return " + ".join(reversed(
    [( ("1" if i == pL-1 else "x") if i >= pL-2 else ("x^" + str(pL - 1 -i)) )
    for i in range(pL-1, -1, -1)   if poli[p][i] ]))

# Формирование проверочной матрицы соответствия синдромов ошибок для разных
бит
# для длины сообщения n и неприводимого полинома степени p (вычисляется по n)
getCheckSindromes = lambda n: [div2( [0]*i+[1]+[0]*(n-i-1) ) for i in range(n)]

# Убирает в полиноме незначащие порядки (нули в бинарном виде) слева
cut = lambda a: a[a.index(1):] if 1 in a else []

# Возвращает параметры сообщения [k, n, p] по одному из параметров k или n
def getKN(x, xIsN=True):
    #   - xIsN=False, то передали k=>   k=x, идет подбор n от x+1, шаг 1
    #   - xIsN=True,  то передали n=>   n=x, идет подбор k от x-1, шаг -1
    x = ([x - 1, x, -1] if xIsN else [x, x + 1, 1]) + [int(not xIsN)]
    while 2**x[0] > 2**x[1] / ( x[1] + 1 ): x[x[3]] += x[2]
    return x[:2] + [ x[1] - x[0] ] # =>[k, n, p]

# Деление полинома полного (n) сообщения на неприводимый полином степени p
def div2(a):                  # возвращает синдром ошибки
    p = getKN(len(a))[2]        # порядок полинома p вычисляется по его длине
    pL, b = len(poli[p]), cut(a[:]) # длина неприводимого полинома, убираем нули
    while len(b) >= pL:
        b = cut([ int(b[i] != poli[p][i]) for i in range(pL) ] + b[pL:])
    return b
```

```
dump_arr = lambda arr : "".join(list(map(str, arr))) # вывести массив
pointers = lambda arr: "".join("^" if i in arr else " " for i in range(1,arr[-1]+1))
gen_arr  = lambda k:    [random.randint(0,1) for i in range(k)] # сгнерировать

def cleanInput(s):
    for char in [",;[]().']: s = s.replace(char, " ")
    return s

# Индексы бит, в которых была допущена ошибка
def Line_With_Errors(pass_block, Er_prob = 0.2, t = 1):
    # длина блока, и индексы битов с ошибками
    block_len, error_bit_list = len(pass_block), []
    # коррекция, если вероятность ошибки > 100%
    if Er_prob > 1.0 or Er_prob < 0.0: Er_prob = 1.0
    print("\n", "-"*70, "\nГенерация "+ ("1 ошибки" if t==1 else "2 ошибок") +\
      " в линии с помехами (", Er_prob*100,\
      "%). Получено для передачи:\n", dump_arr(pass_block), sep="")
    # коррекция, если ошибочных бит больше, чем передаваемых
    if t > block_len: t = block_len
    # Формируем список индексов битов, где может по вероятности Er_prob произойти ошибка (т.е. бит инвертируется)
    err_bits_idxs = random.sample(list(range(block_len)), t)
    # Проходим по списку выбранных для ошибки бит, применяем вероятность,
    # инвертируем если надо, добавляем в итоговый список ошибочно переданных бит
    for i in err_bits_idxs:
        if random.uniform(0.0, 1.0) <= Er_prob:
            pass_block[i] = 0 if pass_block[i] else 1
            error_bit_list.append(i+1)
    print("\nПозиции (!не индексы!) ошибочно переданных бит: ", error_bit_list)
    print("Измененные данные:\n", dump_arr(pass_block), sep="")
    print(pointers(error_bit_list))
    return sorted(error_bit_list)

def lab6():     # Циклический код
    print('\n\nЛабораторная работа №6 ("ЦИКЛИЧЕСКИЙ КОД"). '+\
        'Подготовил: Мазлов Иван, ИВТ-201\n')
    k = input("Введите количество информационных бит k[2; +∞): ")
    try:
        k = int(cleanInput(k))
        if k<2: raise Exception("Неправильный ввод")
    except:
        print(" Установлено значение по умолчанию для Варианта №4: k=58 бит ")
        k = 58
```

```
    k, n, p = getKN(k, xIsN=False)
    Dmin = 2 * 1 + 1 # 2*t+1 t=1
    print("Биты:: Информационные k=", k, ", проверочные p=", p, ", всего n=", \
      n, ". Dmin=", Dmin, sep="")

    arr = gen_arr(k)
    print("\n", "-"*70, "\nПередатчик:\n\nСгенерированное случайным образом"+\
      " информационное сообщение:\n", dump_arr(arr), sep="")
    pCode = div2(arr + [0]*p)
    arr = arr + [0]*(p-len(pCode)) + pCode

    print("Закодированное циклическим кодом (полином: ", poliToStr(p), \
    ") сообщение:\n", dump_arr(arr), "\n", "-"*70, "\nПриемник:", sep="")
    errorBits = Line_With_Errors(arr, Er_prob = 1.0, t = 1)

    errorSindrome = div2(arr)
    if errorSindrome:
        errorBitIndex = getCheckSindromes(n).index(errorSindrome)
        arr[errorBitIndex] = 0 if arr[errorBitIndex] else 1
        print("\nПроизошла ошибка в бите №",errorBitIndex + 1,". Исправленные "\
        + "данные:\n", dump_arr(arr),"\n", pointers([errorBitIndex + 1]),sep="")
    else: print("Передача прошла без ошибок")
    print("Раскодированное информационное сообщение:\n",dump_arr(arr[:k]),sep="")

def main():
    lab6()

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