Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



**Отчет**

**По домашнему заданию №2**

**По курсу «Сети и телекоммуникации»**

**«Разработка программных средств определения обнаруживающей и корректирующей способности кода в линейных протоколах»**

**Вариант 11**

**ИСПОЛНИТЕЛЬ:**

Группа ИУ5-51Б

\_Карпов Д.К.

"7"\_декабря\_2020 г.

**ПРЕПОДАВАТЕЛЬ:**

\_\_Галкин В.А.\_\_

" "\_\_декабря\_\_2020 г.

Москва 2020

Оглавление

[**1)** **Постановка задачи** 3](#_Toc57575573)

[**2)** **Метод решения задачи** 3](#_Toc57575574)

[**3)**  **Реализация на языке Python3** 5](#_Toc57575575)

[**4)**  **Заполненная таблица результатов** 8](#_Toc57575576)

[**5)**  **Выводы** 8](#_Toc57575577)

[6) **Список используемой литературы** 9](#_Toc57575578)

# **Постановка задачи**

Имеется дискретный канал связи, на вход которого подается закодированная в соответствии с вариантом задания кодовая последовательность. В канале возможны ошибки любой кратности. Вектор ошибки может принимать значения от единицы в младшем разряде до единицы во всех разрядах кодового вектора. Для каждого значения вектора ошибки на выходе канала после декодирования определяется факт наличия ошибки или предпринимается попытка ее исправления в зависимости от требуемой способности кода в варианте задания.

Необходимо составить алгоритмы кодирования, декодирования, вычисления обнаруживающей способности кода Co или корректирующей способности кода Ck в зависимости от варианта задания для ошибок всех возможных кратностей.

Вариант задания:

|  |  |  |  |
| --- | --- | --- | --- |
| № варианта | Информационный  вектор | Код | Способность  кода |
| 11 | 1011010001(в файле)  01011010001(использован) | Ц [15,11] | Co |

***Обозначения:***

Ц[15,11] – Циклический код g(x) = х4+ х + 1

Co - обнаруживающая способность кода.

# **Метод решения задачи**

Для решения поставленной задачи были разработаны алгоритмы, изображённые в виде блок-схем (см. пункт 3) и реализованные на языке программирования Python3 (см. пункт 4).

1. **Главный алгоритм** содержит алгоритм решения задачи подсчета обнаруживающей способности кода [n, k] для всех кратностей ошибки от 1 до n включительно при данном информационном векторе m (что определяет количество информационных разрядов k) и порождающем полиноме g (что определяет количество проверочных разрядов n-k, так как максимальная длина остатка Et равна длине g без единицы). Будем считать длину остатка фиксированной и в случае отсутствия значащих цифр на старших разрядах заменять их нулями.

В соответствии с алгоритмом кодирования циклическим кодом сначала выполняется сдвиг влево вектора m на r = n-k разрядов с заменой освободившихся разрядов нулями, для чего используется вспомогательный вектор v. Остаток от деления v на g по модулю 2 (для этого используется алгоритм, описанный в п. 2.2) записывается в проверочный код Et. Далее в конец исходного вектора m добавляется полученный вектор Et.

Для генерации ошибок i-й кратности (i = 1, 2, …, n) и получения общего количества ошибок comb и обнаруженных ошибок count используются функции c[k], где k = 0, …, n-1 – кратность ошибки без единицы (данные функции описаны в алгоритме, описанном в п. 2.3).

После получения count и comb производится подсчет обнаруживающей способности для определённой кратности ошибки (в %) как частное от деления count на comb с умножением на число 100. Результат печатается на экран.

1. В **алгоритме нахождения остатка от деления полиномов по модулю 2** сначала выполняется смещение указателя на начальный коэффициент полинома в делении ind при обнаружении ведущих нулей делимого до первого ненулевого коэффициента. Если после этих действий степень делимого окажется меньшей степени делителя, тогда необходимо записать этот результат в переданный параметр E и выйти из алгоритма.

Следующим пунктом является временное выравнивание длины остатка E и полинома g с целью выполнения поразрядной операции «исключающее ИЛИ».

Далее происходит получение в E первых разрядов из вектора m и выполнение поразрядной операции «исключающее ИЛИ» между E и g. Далее действия выполняются в соответствии с операцией деления в столбик. Отличие состоит в том, что остаток E нужно представить кодом фиксированной длины на 1 меньше, чем у порождающего полинома (например, если остаток получается равным 7 и порождающий полином равен 10011, остаток нужно вернуть как 0011). Для этого выполняется сдвиг вправо на определенное количество разрядов с заполнением освободившихся разрядов нулями [функция shift\_r] и возвращение исходной длины E (то есть удаление последних разрядов, которые были добавлены для выравнивания с g). Максимальная степень остатка ниже степени g на 1, поэтому эта операция не повреждает его. А значит, в случае необходимости определения корректирующей способности можно будет доработать алгоритмы, добавив сравнение полученного остатка (синдрома ошибки) с таблицей синдромов и инвертирование разряда, в котором детектировалась ошибка.

1. **Алгоритмы генерации ошибок в кодовой комбинации и подсчета общего количества ошибок и количества обнаруженных ошибок** выполняют три действия: генерацию ошибок определенной кратности (изменение кодовой комбинации), подсчёт общего количества и обнаруженных ошибок для определенной кратности и возвращение кодовой комбинации в исходное состояние.

Для генерации ошибок в кодовой комбинации и её возвращения в исходное состояние предусмотрены процедуры c[k] – генерация ошибки для ошибки (k+1)-й кратности. В основе алгоритма лежит алгоритм выбора сочетаний из n элементов по k из комбинаторики, где n – длина кодовой комбинации. Поэтому алгоритм является рекурсивным. Например, чтобы выполнить ошибку 5-й кратности, нужно вызвать c[4], которая сгенерирует ошибку в определенном разряде и вызовет c[3], которая также сгенерирует ошибку и вызовет c[2] и т.д. В итоге последним будет вызов функции c[0], которая в данном алгоритме располагается отдельно. Именно она вызывает функцию calc для получения результата обнаружения ошибки и подсчитывает общее и обнаруженное количество ошибок. Ошибка обнаруживается данным кодом, если получаемый остаток от деления кодовой комбинации на порождающий полином не равен 0. Иначе – фиксируется, что ошибка обнаружена не была. Возврат к исходному состоянию осуществляется инвертированием ранее инвертированного бита каждой из вызываемых функций. Для инвертирования используется функция invers.

# **3) Реализация на языке Python3**

1. ***Файл main.py:***

from Algorithms.main\_algo import main\_func  
  
  
if \_\_name\_\_ == **"\_\_main\_\_"**:  
 m = [**0**, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1] *# коэффициенты информационного вектора* g = [1, 0, 0, 1, 1] *# коэффициенты порождающего полинома* main\_func(m, g) *# точка входа в алгоритмы*

1. ***Файл main\_algo.py:***

from Algorithms.ost import ost  
from Algorithms.combinations import c\_1, c\_k  
from typing import List  
  
  
def main\_func(m: List[int], g: List[int]):  
 *"""Выполняет подсчёт обнаруживающей способности циклического кода  
 с образующим полиномом g для ошибок всех кратностей от 1 до  
 длины кодовой комбинации включительно.  
 Здесь m - информационный вектор."""* k = len(m) *# количество информационных разрядов* n = k + len(g) - 1 *# длина кодовой комбинации* r = n - k *# количество разрядов проверочного кода* v = [] *# вспомогательный вектор для вычисления проверочного кода* v.extend(m)  
 e = [] *# проверочный код* for i in range(r):  
 v.append(0)  
 e.append(0)  
 ost(e, v, g) *# получение остатка 'e' при делении 'v' на 'g'* m.extend(e) *# конкатенация информационного вектора 'm' и проверочного кода 'e'* count, comb = c\_1(m, g) *# count - количество обнаруженных ошибок; comb - общее количество ошибок* print(**'Обнаруживающая способность для ошибки кратности 1 = '**, count / comb \* 100)  
 print(**'Ошибок всего: {}. Обнаружено: {}'**.format(comb, count))  
 print(**'--------------------------------------'**)  
 for i in range(1, n):  
 count, comb = c\_k(m, g, i) *# count - количество обнаруженных ошибок; comb - общее количество ошибок* print(**'Обнаруживающая способность для ошибки кратности {} = '**.format(i + 1), count / comb \* 100)  
 print(**'Ошибок всего: {}. Обнаружено: {}'**.format(comb, count))  
 print(**'--------------------------------------'**)

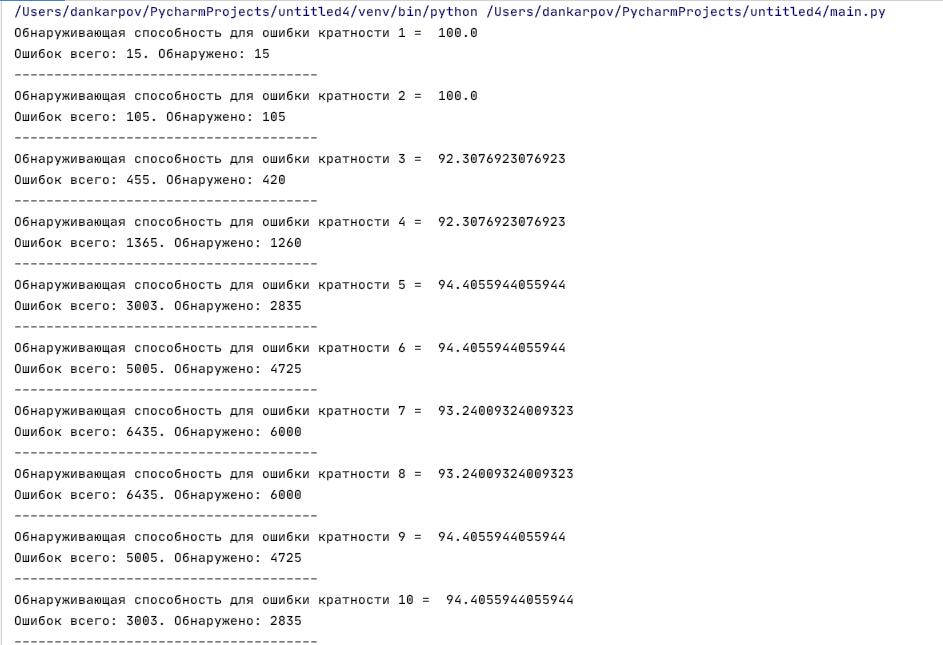
1. ***Файл ost.py:***

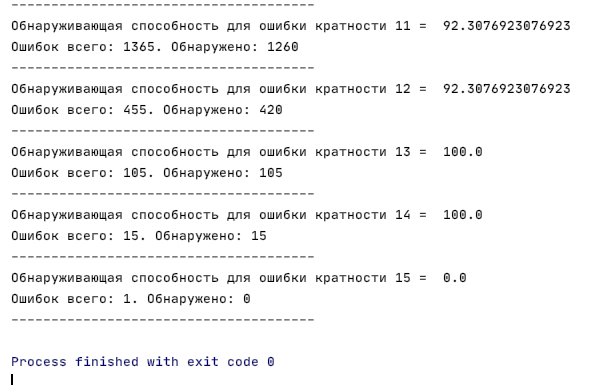
from typing import List  
  
  
def shift\_r(x: List[int]):  
 *"""Выполняет сдвиг вправо списка 'x'  
 с заполнением старшего разряда нулём."""* tmp = []  
 tmp.insert(0, x[0])  
 for i in range(len(x)-1):  
 tmp.insert(i+1, x[i+1])  
 x[i+1] = tmp[i]  
 x[0] = 0  
  
  
def ost(e: List[int], m: List[int], g: List[int]):  
 *"""Выполняет операцию нахождения остатка при делении 'm' на 'g' по модулю 2,  
 присваивая остаток переменной 'e'.  
 Здесь 'e', 'm', 'g' - списки, содержащие коэффициенты соответствующих чисел."""* n = len(m)  
 len\_g = len(g)  
 len\_e = len(e)  
 flag = True  
 ind = 0 *# текущий элемент в m при делении* while m[ind] == 0: *# сброс ведущих нулей* ind += 1  
 if ind > (n-len\_g):  
 for i in range(len\_g-1):  
 e[i] = m[n-(len\_g-1-i)]  
 return  
 while len(e) != len\_g:  
 e.append(0)  
 for i in range(len\_g):  
 e[i] = m[ind + i]  
 while flag:  
 i = 0  
 while (i < len\_g) and ((e[i] ^ g[i]) == 0):  
 i += 1  
 ind += 1  
 tmp = ind  
 j = i  
 while j < len\_g:  
 e[j-i] = e[j] ^ g[j]  
 tmp += 1  
 j += 1  
 k = len\_g-i  
 while k < len\_g:  
 if tmp > n-1:  
 while k < len\_g-1:  
 shift\_r(e)  
 k += 1  
 while len(e) != len\_e:  
 e.pop()  
 return  
 e[k] = m[tmp]  
 tmp += 1  
 k += 1

1. ***Файл combinations.py:***

from Algorithms.ost import ost  
from typing import List  
  
  
def invers(x: int) -> int:  
 *"""Возвращает выполнение операции инверсия к биту x."""* assert 0 <= x <= 1  
 if x == 1:  
 return 0  
 else:  
 return 1  
  
  
def \_calc(m: List[int], g: List[int]) -> int:  
 *"""Возвращает результат обнаружения ошибки.  
 1 - ошибка обнаружена, 0 - ошибка не обнаружена."""* e, reference = [], []  
 for i in range(len(g)-1):  
 e.append(0)  
 reference.append(0)  
 ost(e, m, g)  
 if e != reference:  
 return 1  
 else:  
 return 0  
  
  
def c\_1(m: List[int], g: List[int], count = 0, comb = 0, j = 0) -> (int, int):  
 *"""Выполняет генерацию ошибок 1-й кратности.  
 Принимает параметры: m - проверяемый вектор;  
 g - порождающий полином.  
 Возвращает количество обнаруженных  
 ошибок; общее количество сгенерированных ошибок  
 для кратности 1 соответственно."""* n = len(m)  
 for i in range(j, n):  
 m[i] = invers(m[i])  
 comb += 1  
 count += \_calc(m, g)  
 m[i] = invers(m[i])  
 return count, comb  
  
  
def c\_k(m: List[int], g: List[int], k: int, count=0, comb=0, j=0) -> (int, int):  
 *"""Выполняет генерацию ошибок (k+1)-й кратности.  
 Принимает параметры: m - проверяемый вектор;  
 g - порождающий полином; k - кратность ошибки  
 без единицы.  
 Возвращает количество обнаруженных  
 ошибок; общее количество сгенерированных ошибок  
 для кратности (k+1) соответственно."""* n = len(m)  
 for i in range(j, n-k+1):  
 m[i] = invers(m[i])  
 if k == 1:  
 count, comb = c\_1(m, g, count, comb, i+1)  
 else:  
 count, comb = c\_k(m, g, k-1, count, comb, i+1)  
 m[i] = invers(m[i])  
 return count, comb

# **4) Результат работы программы:**

**

****

# **5) Выводы**

Выполняя данное домашнее задание, я приобрел навыки по разработке и реализации алгоритмов кодирования и декодирования корректирующим кодом. Для циклического кода [15,11] была определена обнаруживающая способность для всех возможных кратностей ошибки в векторе ошибки, который накладывается на передаваемую кодовую комбинацию. Результаты были получены с использованием реализации алгоритмов на языке программирования Python3.

В итоге было определено, что данный код имеет кодовое расстояние, равное трём, так как с его помощью можно обнаружить ошибки всех кратностей подряд до максимальной второй включительно. Также было выяснено, что для ошибок разной кратности, но с одинаковым числом сочетаний, обнаруживающая способность одинаковая.

# **6)** **Список используемой литературы**

1. Галкин В.А., Григорьев Ю.А. Телекоммуникации и сети: Учеб. пособие для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2003. – 608 с.: ил. – (Сер. Информатика в техническом университете)
2. Передача дискретных сообщений: Учебник для вузов/ В. П. Шувалов, Н. В. Захарченко, В. О. Шварцман и др.; Под ред. В. П. Шувалова.–М.: Радио и связь, –1990-464 с.: ил.-