

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования

«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ

Кафедра компьютерной инженерии и моделирования

Отчет по заданию №5
«КОД ХЭММИНГА»

по курсу «Теория информации и кодирования»
студента 3 курса группы ИВТ-б-о-201(1)

Мазлова Ивана Денисовича

Направления подготовки 09.03.01 «Информатика и вычислительная техника»

Симферополь, 2022

Вариант 4.

Техническое задание:

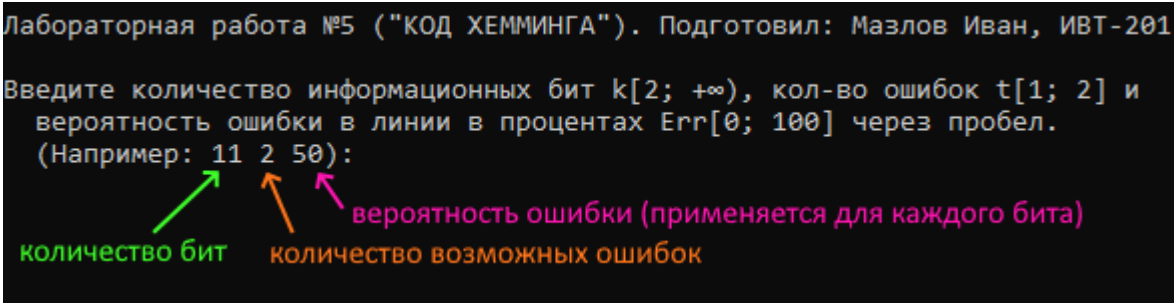
Источник информации вырабатывает сообщения, содержащие k информационных разрядов. Значения разрядов генерируются в двоичной системе счисления счетчиком случайных чисел. Необходимо:

1. разработать программное обеспечение для передатчика, которое будет строить код Хэмминга с заданной исправляющей способностью;
2. разработать программное обеспечение на приемной стороне, позволяющее обрабатывать принятый код Хэмминга;
3. провести комплекс численных экспериментов, в ходе которых на передающей стороне построить код Хэмминга с заданной исправляющей способностью, сгенерировать случайным образом кратность ошибки и ошибочную кодовую комбинацию, на приемной стороне по принятому коду Хэмминга определить кратность ошибки и скорректировать принятую кодовую комбинацию.

Описание работы программы:

Программа разработана на языке программирования Python 3 и позволяет проводить эксперименты с данными любой длины (не только 11 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.

Был выполнен вариант «модифицированного кода Хэмминга», когда при передаче может случиться две ошибки, но и «стандартный вариант с единичной ошибкой» тоже предусмотрен. Кроме того можно вводить вероятность возникновения ошибки при передаче сообщения.



```
Лабораторная работа №5 ("КОД ХЕММИНГА"). Подготовил: Мазлов Иван, ИВТ-201
Введите количество информационных бит k[2; +∞), кол-во ошибок t[1; 2] и
вероятность ошибки в линии в процентах Err[0; 100] через пробел.
(Например: 11 2 50):
количество бит      количество возможных ошибок      вероятность ошибки (применяется для каждого бита)
```

Рис. 1 Запрос параметров в консоли.

Вначале мы рассчитываем основные показатели системы: общее количество бит (n), количество проверочных бит (p), D_{min} и генерируем информационный массив (arr) случайным образом. Далее определяем позиции для проверочных бит и перестраиваем массив, освобождая под них место. Потом кодируем информационное сообщение кодом Хэмминга.

```

83     Err/= 100
84     n = k + 1
85     while (2**k) > ( (2**n) / (n+1) ): n += 1
86     p, Dmin = n-k, 2 * t + 1
87     print("Биты:: Информационные k=", k, ", проверочные p=", p, ", всего n=", \
88           n, ". Dmin=", Dmin, sep="")
89
90     arr = gen_arr(k)
91     print("\n", "-"*70, "\nПередатчик:\n\nСгенерированное случайным образом"+\
92           " информационное сообщение:\n", dump_arr(arr), sep="")
93     insert_check_bits(arr, n)
94     print("Освобождаем позиции для проверочных битов", " и бита четности в "+\
95           "конце" if t==2 else "" ,":\n", dump_arr(arr), "x" if t==2 else "", sep="")
96
97     Haming(arr, False)
98     print("Закодированное кодом Хемминга сообщение:\n", dump_arr(arr), sep="")

```

Рис. 2 Расчет показателей, генерация и кодирование информационного массива.

Для расчета общего количества бит и количества проверочных бит используется формула:

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

Рис. 3 Условие, определяющее длину комбинации и количество проверочных бит

Если задана возможность появления 2-х ошибок при передаче, то необходимо ещё добавить бит четности. Я добавил его в конце информационного массива. Бит рассчитывается как сумма по модулю два всех битов информационного сообщения, т.е. в сумме с битом четности должен получаться 0.

```

100     if t==2: # есть две ошибки, добавляем бит четности в конце кода
101         arr+= [sum(arr)&1]
102         print("Добавлен бит четности в конце:\n", dump_arr(arr), sep="")
103

```

Рис. 4 Расчет Бита четности закодированного сообщения.

Сгенерированное случайным образом информационное сообщение:
00110011000
Освобождаем позиции для проверочных битов и бита четности в конце:
--0-011-0011000x
Закодированное кодом Хемминга сообщение:
010101100011000
Добавлен бит четности в конце:
0101011000110000 **Бит четности**

Рис. 5 Вывод сообщения с битом четности

Далее закодированное сообщение (с битом четности или без него как в простом варианте) отправляется в путешествие по линии связи, где, применяя коэффициент вероятности могут инвертироваться (ошибочно передаваться) один или два бита (или ни одного, если не сработает вероятность).

Приемник распаковывает сообщение и **в простом случае с единичной ошибкой** пересчитывает кодовые биты кода Хэмминга: по формулам рассчитывается значение синдрома ошибки. Если подсчет не равен 0, то значит в «зоне» этого контрольного бита произошла ошибка. Из этих бит составляется синдром ошибки – бинарное число, которое «показывает» бит, в котором произошла эта ошибка (в десятичной системе). При этом порядок счета бит начинается с 1 и идет слева направо. В этом случае мы просто инвертируем этот бит и таким образом исправляем ошибку передачи.

Распределение разрядов в коде Хэмминга имеет следующий вид:

$$\overset{1}{b_0} \overset{2}{b_1} \overset{3}{a_1} \overset{4}{b_2} \overset{5}{a_2} \overset{6}{a_3} \overset{7}{a_4} \overset{8}{b_3} \overset{9}{a_5} \overset{10}{a_6} \overset{11}{a_7} \overset{12}{a_8} \dots$$

Контрольные суммы будут иметь следующий вид

$$S_1 = b_0 + a_1 + a_2 + a_4 + a_5 + a_7 + \dots$$

$$S_2 = b_1 + a_1 + a_3 + a_4 + a_6 + a_7 + \dots$$

$$S_3 = b_2 + a_2 + a_3 + a_4 + a_8 + \dots$$

$$S_4 = b_3 + a_5 + a_6 + a_7 + a_8 + \dots$$

Рис. 6 Формулы расчета кода Хэмминга

Степень двойки	2 ⁰	2 ¹		2 ²				2 ³								2 ⁴					
№ бита	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Шаг	0	0	1	1	0	1	1	0	0	1	0	0	0	1	0	1	0	0	1	1	1
1	1		1		1		1		1		1		1		1		1		1		1
2		2	2			2	2			2	2			2	2			2	2		
4				4	4	4	4					4	4	4	4					4	4
8								8	8	8	8	8	8	8	8						
16																16	16	16	16	16	16

Рис. 7 Графическое представление расположения контрольных битов.

При возможности возникновения двух ошибок (**более сложный вариант**) код Хэмминга может исправить код, если произошла только одна ошибка и сигнализировать, что произошло две ошибки, но исправить код в этом случае не получится.

Бит четности используется как контрольная сумма (CRC) для всего сообщения: закодированного информационного и самого бита четности. После получения сообщения с битом четности нужно пересчитать сумму по модулю 2 всех бит сообщения (0-если нет изменений или изменений было четное количество) и отдельно пересчитать код Хэминга (синдром ошибки равен 0 - код не поврежден, не ноль-поврежден).

В этом случае возможно 4-е варианта:

Таблица 1.

Значения [бита четности и суммы по модулю два битов закодированного сообщения]	Описание и действия
[0, 0]	Сообщение передано без ошибок. Отбросить бит четности и раскодировать кодом Хэминга сообщение.
[0, 1]	Двойная ошибка в коде. Ничего сделать нельзя.
[1, 0]	Была единичная ошибка при передаче, но она попала именно на бит четности – код не пострадал. Просто отбрасываем бит четности и декодируем код Хэминга. Сообщение не пострадало.
[1, 1]	Произошла единичная ошибка в битах кода. Тогда отбрасываем бит четности и рассчитываем синдром ошибки, десятичное значение которого является порядковым номером неправильного бита. Инвертируем этот бит. Код исправлен.

Для обнаружения двукратных ошибок минимальное кодовое расстояние должно быть $D_{\min}=4$.

```

125     # Есть 4 возможных варианта:
126     # 1) ParityBit=0 и Hamming=0 => ошибок не было
127     # 2) ParityBit=0 и Hamming=1 => двойная ошибка!! ничего нельзя сделать
128     # 3) ParityBit=1 и Hamming=0 => 1 ошибка и попала на ParityBit. => 1)
129     # 4) ParityBit=1 и Hamming=1 => 1 ошибка. Исправляем код по Хеммингу.
130     #      0              0
131     if (not res) and not ParityBit:
132         print("\nПередача прошла без ошибок. Код не изменен.")
133
134     # бит четности есть, но код не совпадает = двойная ошибка
135     #      1              0
136     elif res and not ParityBit and t==2:
137         print("\nПри передаче прошла двойная ошибка."+\\
138             " Декодирование невозможно!")
139
140     #      0              1
141     elif (not res) and ParityBit: # поврежден check_bit, информблок нормальный
142         print("\nПри передаче была одна ошибка, но повредился именно бит"+\\
143             " четности.\nПередача информационного сообщения прошла без ошибок."+\\
144             " Принятые данные:\n", dump_arr(arr), sep="")
145     #      1              1
146     else:
147         print("\nПроизошла единичная ошибка в бите на позиции"+\\
148             " (не индексе!):", res)
149         print("Исправляем ошибку:")
150         arr[res-1] = 0 if arr[res-1] else 1
151         print("Исправленное информационное сообщение:\n", dump_arr(arr), sep="")

```

Рис. 8 Часть кода обрабатывающая ошибки

```

12 def del_check_bits(arr):
13     for power in range(int(log2(len(arr))), -1, -1):
14         del arr[(2**power)-1]

```

Рис. 9 Восстановление информационного сообщения из кода Хэмминга.

Вывод:

В ходе выполнения лабораторной работы мы познакомились на практике с алгоритмом создания и применения кода Хэминга для передачи информации по помехонеустойчивым линиям, научился определять кратность ошибки и исправлять её (когда это возможно), 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, кодирующую её при помощи кода Хэминга, рассчитывающую проверочный бит. Программа имитировала ошибку в передаче одного из битов сообщения, а приемная часть программы обнаруживала и исправляла её, используя код Хэминга и

контрольный бит. Контрольный бит использовался для сигнализирования ситуации, когда произошло более двух ошибок при передаче кода и, код не может быть исправлен. По сути этот бит представляет собой контрольную сумму закодированного сообщения.

Были проведены тесты при исходном количестве информационных бит – 11 и более, проверены все варианты результатов передачи: без ошибок, ошибку можно исправить, ошибку нельзя исправить, ошибка попала на бит четности и не повлияла на информационное сообщение (при этом код не пострадал – по сути передача информационного сообщения без ошибок). 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. Однако у кода Хэминга есть свои недостатки и в настоящее время для защиты передаваемой информации от помех используются более сложные коды.

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.

Приложение 1.

Результаты работы программы.

```
Лабораторная работа №5 ("КОД ХЕММИНГА"). Подготовил: Мазлов Иван, ИВТ-201

Введите количество информационных бит k[2; +∞), кол-во ошибок t[1; 2] и
  вероятность ошибки в линии в процентах Err[0; 100] через пробел.
  (Например: 11 2 50): 11 2 50
Биты: Информационные k=11, проверочные p=4, всего n=15. Dmin=5

-----
Передатчик:

Сгенерированное случайным образом информационное сообщение:
10011101010
Освобождаем позиции для проверочных битов и бита четности в конце:
--1-001-1101010x
Закодированное кодом Хемминга сообщение:
101100101101010
Добавлен бит четности в конце:
1011001011010100

-----
Генерация 2 ошибок в линии с помехами (50.0%). Получено для передачи:
1011001011010100

Позиции (!не индексы!) ошибочно переданных бит: [4]
Измененные данные:
1010001011010100

-----
Приемник:

Произошла единичная ошибка в бите на позиции (не индексе!): 4
Исправляем ошибку:
Исправленное информационное сообщение:
101100101101010
```

Рис. 10 Тест 1 (с битом четности).
Произошла только одна ошибка в информ. блоке. Исправлена.

Лабораторная работа №5 ("КОД ХЕММИНГА"). Подготовил: Мазлов Иван, ИВТ-201

Введите количество информационных бит $k[2; +\infty)$, кол-во ошибок $t[1; 2]$ и вероятность ошибки в линии в процентах $Err[0; 100]$ через пробел.

(Например: 11 2 50): 11 2 90

Биты:: Информационные $k=11$, проверочные $p=4$, всего $n=15$. $D_{min}=5$

Передатчик:

Сгенерированное случайным образом информационное сообщение:

10010101111

Освобождаем позиции для проверочных битов и бита четности в конце:

--1-001-0101111x

Закодированное кодом Хемминга сообщение:

011100110101111

Добавлен бит четности в конце:

0111001101011110

Генерация 2 ошибок в линии с помехами (90.0%). Получено для передачи:

0111001101011110

Позиции (!не индексы!) ошибочно переданных бит: [4, 8]

Измененные данные:

0110001001011110

Приемник:

При передаче прошла двойная ошибка. Декодирование невозможно!

Рис. 11 Тест 2 (с битом четности).
Произошло две ошибки. Исправление невозможно.

Лабораторная работа №5 ("КОД ХЕММИНГА"). Подготовил: Мазлов Иван, ИВТ-201

Введите количество информационных бит $k[2; +\infty)$, кол-во ошибок $t[1; 2]$ и вероятность ошибки в линии в процентах $Err[0; 100]$ через пробел.

(Например: 11 2 50): 11 2 1

Биты:: Информационные $k=11$, проверочные $p=4$, всего $n=15$. $Dmin=5$

Передатчик:

Сгенерированное случайным образом информационное сообщение:

11001111011

Освобождаем позиции для проверочных битов и бита четности в конце:

--1-100-1111011x

Закодированное кодом Хемминга сообщение:

111010001111011

Добавлен бит четности в конце:

1110100011110110

Генерация 2 ошибок в линии с помехами (1.0%). Получено для передачи:

1110100011110110

Позиции (!не индексы!) ошибочно переданных бит: []

Измененные данные:

1110100011110110

Приемник:

Передача прошла без ошибок. Код не изменен.

Рис. 12 Тест 3 (с битом четности).
Передача прошла без ошибок.

Лабораторная работа №5 ("КОД ХЕММИНГА"). Подготовил: Мазлов Иван, ИВТ-201

Введите количество информационных бит $k[2; +\infty)$, кол-во ошибок $t[1; 2]$ и вероятность ошибки в линии в процентах $Err[0; 100]$ через пробел.

(Например: 11 2 50): 11 2 50

Биты:: Информационные $k=11$, проверочные $p=4$, всего $n=15$. $D_{min}=5$

Передатчик:

Сгенерированное случайным образом информационное сообщение:

00000111010

Освобождаем позиции для проверочных битов и бита четности в конце:

--0-000-0111010x

Закодированное кодом Хемминга сообщение:

110000000111010

Добавлен бит четности в конце:

1100000001110100

Генерация 2 ошибок в линии с помехами (50.0%). Получено для передачи:

1100000001110100

Позиции (!не индексы!) ошибочно переданных бит: [16]

Измененные данные:

1100000001110101

Приемник:

При передаче была одна ошибка, но повредился именно бит четности.

Передача информационного сообщения прошла без ошибок. Принятые данные:

110000000111010

Рис. 13 Тест 4 (с битом четности).

Передача прошла с единичной ошибкой, но повредился бит четности.

Исправление произведено.

Лабораторная работа №5 ("КОД ХЕММИНГА"). Подготовил: Мазлов Иван, ИВТ-201

Введите количество информационных бит $k[2; +\infty)$, кол-во ошибок $t[1; 2]$ и вероятность ошибки в линии в процентах $Err[0; 100]$ через пробел.

(Например: 11 2 50): 11 1 80

Биты:: Информационные $k=11$, проверочные $p=4$, всего $n=15$. $Dmin=3$

Передатчик:

Сгенерированное случайным образом информационное сообщение:

01111111011

Освобождаем позиции для проверочных битов:

--0-111-1111011

Закодированное кодом Хемминга сообщение:

100011101111011

Генерация 1 ошибки в линии с помехами (80.0%). Получено для передачи:

100011101111011

Позиции (!не индексы!) ошибочно переданных бит: [3]

Измененные данные:

101011101111011

Приемник:

Произошла единичная ошибка в бите на позиции (не индексе!): 3

Исправляем ошибку:

Исправленное информационное сообщение:

100011101111011

Рис. 14 Тест 5 (без бита четности).
Единичная ошибка исправлена.

```

Введите количество информационных бит k[2; +∞), кол-во ошибок t[1; 2] и
вероятность ошибки в линии в процентах Err[0; 100] через пробел.
(Например: 11 2 50): 80 2 30
Биты:: Информационные k=80, проверочные p=7, всего n=87. Dmin=5

-----
Передатчик:

Сгенерированное случайным образом информационное сообщение:
11001101001110010001110010011100010111011111010111100100111101000110101001110010
Освобождаем позиции для проверочных битов и бита четности в конце:
--1-100-1101001-110010001110010-0111000101110111110101111001001-11101000110101001110010x
Закодированное кодом Хемминга сообщение:
101010001101001111001000111001010111000101110111110101111001001011101000110101001110010
Добавлен бит четности в конце:
1010100011010011110010001110010101110001011101111101011110010010111010001101010011100101

-----
Генерация 2 ошибок в линии с помехами (30.0%). Получено для передачи:
1010100011010011110010001110010101110001011101111101011110010010111010001101010011100101

Позиции (!не индексы!) ошибочно переданных бит:  []
Измененные данные:
1010100011010011110010001110010101110001011101111101011110010010111010001101010011100101

-----
Приемник:

Передача прошла без ошибок. Код не изменен.

```

Рис. 15 Тест 6 (без бита четности). 80-битное сообщение.
Передача без ошибок.

```

Лабораторная работа №5 ("КОД ХЕММИНГА"). Подготовил: Мазлов Иван, ИВТ-201

Введите количество информационных бит k[2; +∞), кол-во ошибок t[1; 2] и
  вероятность ошибки в линии в процентах Err[0; 100] через пробел.
(Например: 11 2 50): 100 1 100
Биты:: Информационные k=100, проверочные p=7, всего n=107. Dmin=3

-----
Передатчик:

Сгенерированное случайным образом информационное сообщение:
000010101001101101100110011010100011001101101000000101001011000000011111110001111001110101110111101
Освобождаем позиции для проверочных битов:
--0-000-1010100-110110110011001-1010100011001101101000000101001-0110000000011111110001111001110101110111101
Закодированное кодом Хемминга сообщение:
10010000101010011101101100110010101010001100110110100000010100110110000000011111110001111001110101110111101

-----
Генерация 1 ошибки в линии с помехами (100.0%). Получено для передачи:
10010000101010011101101100110010101010001100110110100000010100110110000000011111110001111001110101110111101

Позиции (!не индексы!) ошибочно переданных бит: [36]
Измененные данные:
10010000101010011101101100110010101110001100110110100000010100110110000000011111110001111001110101110111101

-----
Приемник:

Произошла единичная ошибка в бите на позиции (не индексе!): 36
Исправляем ошибку:
Исправленное информационное сообщение:
10010000101010011101101100110010101010001100110110100000010100110110000000011111110001111001110101110111101

```

Рис. 16 Тест 7 (без бита четности). 100-битное сообщение.
Одна ошибка. Исправлена.

Приложение 2.

Полный листинг программы

```

from functools import reduce as red
from math import log2 as log2, ceil as ceil
import random

dump_arr = lambda arr : "".join(list(map(str, arr)))
gen_arr = lambda k: [random.randint(0,1) for i in range(k)]

def insert_check_bits(arr, n):
    for power in range(ceil(log2(n))):
        arr.insert((2**power)-1, "-")

def del_check_bits(arr):

```

```

for power in range(int(log2(len(arr))), -1, -1):
    del arr[(2**power)-1]

#   x x  x   x
#   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
# a = [8,8,1,8,0,1,0,8,0,1,1,1,0,1,1]

def Get_Haming_bit(power, arr):
    st, L, res = 2**power, len(arr), []
    for start in range(st-1, L, 2*st):
        res+=arr[start : min(start+st, L)]
    return sum(res[1:])&1

def Haming(arr, Check=True):
    res = 0
    for power in range(int(log2(len(arr)))+1):
        if Check: # Check bit
            if Get_Haming_bit(power, arr)!=arr[(2**power)-1]:
                res += 2**power
        else: # Set bit
            HB = Get_Haming_bit(power, arr)
            arr[(2**power)-1] = HB
    return res

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_idx = 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_idx:
    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)

# для теста повреждения бита четности
#pass_block[-1] = 0 if pass_block[-1] else 1
#error_bit_list.append(len(pass_block))

print("\nПозиции (!не индексы!) ошибочно переданных бит: ", error_bit_list)
print("Измененные данные:\n", dump_arr(pass_block), sep="")
return sorted(error_bit_list) # индексы бит, в которых была допущена ошибка

def lab5():
    print("\n\nЛабораторная работа №5 ("КОД ХЕММИНГА"). '+\
        'Подготовил: Мазлов Иван, ИВТ-201\n')
    k = input("Введите количество информационных бит k[2; +∞), "+\

```



```

"кол-во ошибок t[1; 2]"+"\n
" и \n вероятность ошибки в линии в процентах Err[0; 100] через пробел."+"\n
\n (Например: 11 2 50): ")
try:
    k, t, Err = map(int, k.replace(",", " ").replace(";", " ").\
        replace("]", " ").replace("[", " ").replace(")", " ").\
        replace("(", " ").split())
    if t not in [1,2]:
        print(" Установлено значение по умолчанию для количества ошибок. t=1")
        t = 1
    if not 0<=Err<=100:
        Err = 50
        print(" Установлено значение по умолчанию вероятности ошибки. Err=50%")
except:
    print(" Установлено значение по умолчанию для Варианта №4: 11 бит "+"
    "(единичная ошибка, Err=50%)\n")
    k = 11
    t = 1
    Err = 50

Err/= 100
n = k + 1
while (2**k) > ( (2**n) / (n+1) ): n += 1
p, Dmin = n-k, 2 * 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="")

```

```
insert_check_bits(arr, n)
print("Освобождаем позиции для проверочных битов", " и бита четности в "+" \
      "конце" if t==2 else "" , ":\n", dump_arr(arr), "x" if t==2 else "", sep="")
```

```
Haming(arr, False)
```

```
print("Закодированное кодом Хемминга сообщение:\n", dump_arr(arr), sep="")
```

```
if t==2: # есть две ошибки, добавляем бит четности в конце кода
    arr+= [sum(arr)&1]
    print("Добавлен бит четности в конце:\n", dump_arr(arr), sep="")
```

```
Line_With_Errors(arr, Err, t)
```

```
#Line_With_Errors(arr, 1.0, 1) # тест варианта с одной ошибкой
```

```
print("\n", "-"*70, "\nПриемник:", sep="")
```

```
# если стояло, что может быть 2 ошибки, то значит есть
```

```
# ParityBit. Генерируем его из сообщения и сравниваем с тем,
```

```
# что получили по линии связи
```

```
if t==2:
```

```
    # Получаем результирующий бит четности:
```

```
    # полученный (сумма модуля 2) рассчитанный заново по сообщению
```

```
    #ParityBit = (arr[-1] + sum(arr[:-1]))&1
```

```
    ParityBit = sum(arr)&1
```

```
    arr = arr[:-1] # обрезаем бит четности
```

```
else:
```

```
    ParityBit = 0
```

```
res = Hamming(arr)
```

```
# Есть 4 возможных варианта:
```

```
# 1) ParityBit=0 и Hamming=0 => ошибок не было
```

```
# 2) ParityBit=0 и Hamming=1 => двойная ошибка!! ничего нельзя сделать
```

```
# 3) ParityBit=1 и Hamming=0 => 1 ошибка и попала на ParityBit. => 1)
```

```
# 4) ParityBit=1 и Hamming=1 => 1 ошибка. Исправляем код по Хаммингу.
```

```
# 0 0
```

```
if (not res) and not ParityBit:
```

```
    print("\nПередача прошла без ошибок. Код не изменен.")
```

```
    del_check_bits(arr)
```

```
    print(dump_arr(arr))
```

```
# бит четности есть, но код не совпадает = двойная ошибка
```

```
# 1 0
```

```
elif res and not ParityBit and t==2:
```

```
    print("\nПри передаче прошла двойная ошибка."+\\
```

```
    " Декодирование невозможно!")
```

```
# 0 1
```

```
elif (not res) and ParityBit: # поврежден check_bit, информблок нормальный
```

```
    print("\nПри передаче была одна ошибка, но повредился именно бит"+\\
```

```
    " четности.\nПередача информационного сообщения прошла без ошибок."+\\
```

```
    " Принятые данные:", sep="")
```

```
    del_check_bits(arr)
```

```
    print(dump_arr(arr))
```

```
# 1 1
```

```
else:
```

```
    print("\nПроизошла единичная ошибка в бите на позиции"+\\
```

```
    " (не индексе!):", res)
```

```
print("Исправляем ошибку:")  
arr[res-1] = 0 if arr[res-1] else 1  
print("Исправленное информационное сообщение:", sep="")  
del_check_bits(arr)  
print(dump_arr(arr))
```

```
def main():  
    #a = [8,8,1,8,0,1,0,8,0,1,1,1,0,1,1]  
    lab5()
```

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