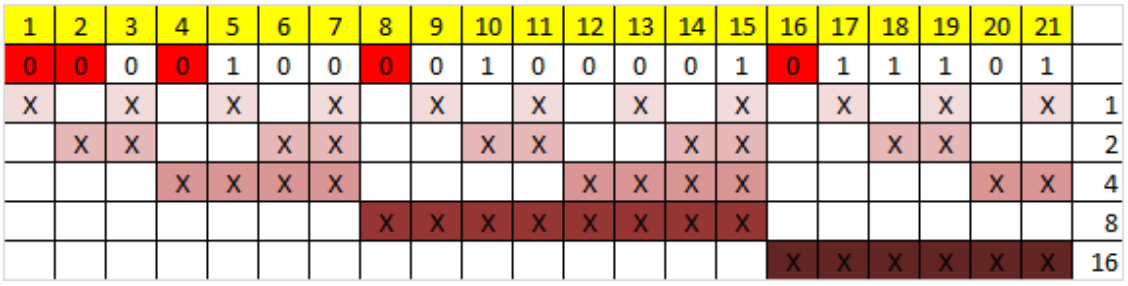
Алгоритм Хемминга

1. Тонкости реализации.
    1. контрольный бит с номером N контролирует все последующие N бит через каждые N бит, начиная с позиции N.
    2. 

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	
0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	1	1	0	1	
X		X		X		X		X		X		X		X		X		X		X	1
	X	X			X	X			X	X			X	X			X	X			2
			X	X	X	X					X	X	X	X					X	X	4
							X	X	X	X	X	X	X	X							8
															X	X	X	X	X	X	16

    3. ```python
def set_control_bits(self):
    """
    Подсчет значений контрольных битов
    """
    for i in range(len(self.initial_code)):
        binary_code = str(bin(i + 1))[2:]
        binary_code = '0' * (len(self.bits) - len(binary_code)) + binary_code
        if self.initial_code[i] != 0:
            for j in range(len(binary_code)):
                self.bits[-j - 1] += 1 if binary_code[j] == '1' else 0
                self.bits[-j - 1] %= 2
```
        a. В данном методе вычисляются значения контрольных битов. Мы просто проходим по всем единицам и прибавляем на 1 значение для контролируещго эту единицу бита. В итогу и делим по модулу, после чего получается 1 или 0.
```

4. 

```
def check_correctness(self):
    for i in range(len(self.initial_code)):
        binary_code = str(bin(i + 1))[2:]
        binary_code = '0' * (len(self.bits) - len(binary_code)) + binary_code
        if self.initial_code[i] != 0:
            for j in range(len(binary_code)):
                self.bits[-j - 1] += 1 if binary_code[j] == '1' else 0
                self.bits[-j - 1] %= 2

    i = 1
    error_bit = 0
    for bit in self.bits:
        if bit == 1:
            error_bit += i
        i *= 2

    if error_bit != 0:
        print(f"Error found. Bit No. - {error_bit}. Fixing")

        self.initial_code[error_bit - 1] = 1 - self.initial_code[error_bit - 1]
```

    a. С декодировкой тоже самое, только в этот раз, все биты должны быть равны 0. Если такого не происходит, то при приведение всех контролирующих битов к числу, мы получим индекс ошибочного бита и заменем его.

2. Структуры данных были выбраны и с какой целью
    1. Списки

3. Кратко указать описание структуры приложений.

1.
```python
def encode(self, code: str):

    # Разбиваем входное сообщение на список символов
    self.initial_code = list(map(lambda s: int(s), list(code)))

    # Расставляем проверочные биты, пока что со значением 0
    self.insert_check_bits()

    # Устаналиваем для каждого проверочного бита его значение 0 или 1
    self.set_control_bits()

    # Вставляем обновленные значения в сообщение
    self.correct_control_bits()

    # Сообщение с проверочными битами
    self.encoded_code = "".join(str(i) for i in self.initial_code)

    # Запись закодироанного сообщения
    with open('encode_text.txt', 'w') as file_writer:
        file_writer.write(self.encoded_code)
    print(encoder.encoded_code)
```

2.
```python
def decode(self, code):
    """
    Декодирование сообщение закодироанного алгоритмом Хемминга
    """
    # Разбиваем входное сообщение на список символов
    self.initial_code = list(map(lambda s: int(s), list(code)))

    # Создаем массив битов
    self.set_bits()

    # Проверяем корректность контрольных битов, заново высчитываем все контрольные биты.
    self.check_correctness()

    # Убираем контрольные биты из входного сообщения.
    self.set_decoded_code()

    with open('decode_text.txt', 'w') as file_writer:
        file_writer.write(self.decoded_code)
    print(decoder.decoded_code)
```