**Вариант 1**

**#1**

Размер кода не ясен, но должен быть больше 20...

**#2**

Двоичный код (3, 6) с образующим многочленом

**Не является циклическим:**

, так как не делится без остатка.

Значит он просто полиномиальный.

Кодирующая матрица (3х6) строится как сдвиг образующего многочлена [0, 0, 1, 0, 1, 1] влево, начиная с нижней строки и до верхней:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |

Построим множество кодовых слов:

|  |  |  |
| --- | --- | --- |
|  |  | [0,0,0,0,0,0] |
|  |  | [0,0,1,0,1,1] |
|  |  | [0,1,0,1,1,0] |
|  |  | [0,1,1,1,0,1] |
|  |  | [1,0,1,1,0,0] |
|  |  | [1,0,0,1,1,1] |
|  |  | [1,1,1,0,1,0] |
|  |  | [1,1,0,0,0,1] |

Найдем остатки от деления на :

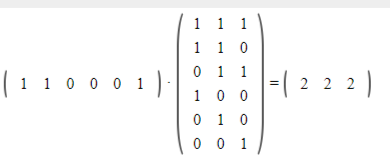
|  |  |  |
| --- | --- | --- |
|  |  | [0,0,1] |
|  |  | [0,1,0] |
|  |  | [1,0,0] |
|  |  | [0,1,1] |
|  |  | [1,1,0] |
|  |  | [1,1,1] |

Запишем проверочную матрицу, используя эти остатки (вертикально):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 |

Транспонированная проверочная матрица должна выдавать нулевой вектор при умножении на правильное кодовое слово:

Например на .



В результате получается нулевой вектор по модулю 2.

Кодовые слова - векторы размера 6 с элементами из множества {0,1}. Их общее количество составляет 64, но при кодировании используются только 8 из них, которые образуют подгруппу. Построим смежные с ней классы, поэтапно прибавляя вектор из общей группы ко всем векторам данной подгруппы:

Итого имеется 8 смежных классов:

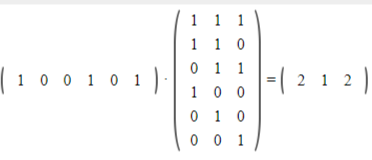
|  |  |  |
| --- | --- | --- |
| Номер класса | Что прибавляли | Состав класса |
| 1 | [0, 0, 0, 0, 0, 0]  [0, 0, 1, 0, 1, 1]  [0, 1, 0, 1, 1, 0]  [0, 1, 1, 1, 0, 1]  [1, 0, 0, 1, 1, 1]  [1, 0, 1, 1, 0, 0]  [1, 1, 0, 0, 0, 1]  [1, 1, 1, 0, 1, 0] | [0, 0, 0, 0, 0, 0]  [0, 0, 1, 0, 1, 1]  [0, 1, 0, 1, 1, 0]  [0, 1, 1, 1, 0, 1]  [1, 0, 1, 1, 0, 0]  [1, 0, 0, 1, 1, 1]  [1, 1, 1, 0, 1, 0]  [1, 1, 0, 0, 0, 1] |
| 2 | [0, 0, 0, 0, 0, 1]  [0, 0, 1, 0, 1, 0]  [0, 1, 0, 1, 1, 1]  [0, 1, 1, 1, 0, 0]  [1, 0, 0, 1, 1, 0]  [1, 0, 1, 1, 0, 1]  [1, 1, 0, 0, 0, 0]  [1, 1, 1, 0, 1, 1] | [0, 0, 0, 0, 0, 1]  [0, 0, 1, 0, 1, 0]  [0, 1, 0, 1, 1, 1]  [0, 1, 1, 1, 0, 0]  [1, 0, 1, 1, 0, 1]  [1, 0, 0, 1, 1, 0]  [1, 1, 1, 0, 1, 1]  [1, 1, 0, 0, 0, 0] |
| 3 | [0, 0, 0, 0, 1, 0]  [0, 0, 1, 0, 0, 1]  [0, 1, 0, 1, 0, 0]  [0, 1, 1, 1, 1, 1]  [1, 0, 0, 1, 0, 1]  [1, 0, 1, 1, 1, 0]  [1, 1, 0, 0, 1, 1]  [1, 1, 1, 0, 0, 0] | [0, 0, 0, 0, 1, 0]  [0, 0, 1, 0, 0, 1]  [0, 1, 0, 1, 0, 0]  [0, 1, 1, 1, 1, 1]  [1, 0, 1, 1, 1, 0]  [1, 0, 0, 1, 0, 1]  [1, 1, 1, 0, 0, 0]  [1, 1, 0, 0, 1, 1] |
| 4 | [0, 0, 0, 0, 1, 1]  [0, 0, 1, 0, 0, 0]  [0, 1, 0, 1, 0, 1]  [0, 1, 1, 1, 1, 0]  [1, 0, 0, 1, 0, 0]  [1, 0, 1, 1, 1, 1]  [1, 1, 0, 0, 1, 0]  [1, 1, 1, 0, 0, 1] | [0, 0, 0, 0, 1, 1]  [0, 0, 1, 0, 0, 0]  [0, 1, 0, 1, 0, 1]  [0, 1, 1, 1, 1, 0]  [1, 0, 1, 1, 1, 1]  [1, 0, 0, 1, 0, 0]  [1, 1, 1, 0, 0, 1]  [1, 1, 0, 0, 1, 0] |
| 5 | [0, 0, 0, 1, 0, 0]  [0, 0, 1, 1, 1, 1]  [0, 1, 0, 0, 1, 0]  [0, 1, 1, 0, 0, 1]  [1, 0, 0, 0, 1, 1]  [1, 0, 1, 0, 0, 0]  [1, 1, 0, 1, 0, 1]  [1, 1, 1, 1, 1, 0] | [0, 0, 0, 1, 0, 0]  [0, 0, 1, 1, 1, 1]  [0, 1, 0, 0, 1, 0]  [0, 1, 1, 0, 0, 1]  [1, 0, 1, 0, 0, 0]  [1, 0, 0, 0, 1, 1]  [1, 1, 1, 1, 1, 0]  [1, 1, 0, 1, 0, 1] |
| 6 | [0, 0, 0, 1, 0, 1]  [0, 0, 1, 1, 1, 0]  [0, 1, 0, 0, 1, 1]  [0, 1, 1, 0, 0, 0]  [1, 0, 0, 0, 1, 0]  [1, 0, 1, 0, 0, 1]  [1, 1, 0, 1, 0, 0]  [1, 1, 1, 1, 1, 1] | [0, 0, 0, 1, 0, 1]  [0, 0, 1, 1, 1, 0]  [0, 1, 0, 0, 1, 1]  [0, 1, 1, 0, 0, 0]  [1, 0, 1, 0, 0, 1]  [1, 0, 0, 0, 1, 0]  [1, 1, 1, 1, 1, 1]  [1, 1, 0, 1, 0, 0] |
| 7 | [0, 0, 0, 1, 1, 0]  [0, 0, 1, 1, 0, 1]  [0, 1, 0, 0, 0, 0]  [0, 1, 1, 0, 1, 1]  [1, 0, 0, 0, 0, 1]  [1, 0, 1, 0, 1, 0]  [1, 1, 0, 1, 1, 1]  [1, 1, 1, 1, 0, 0] | [0, 0, 0, 1, 1, 0]  [0, 0, 1, 1, 0, 1]  [0, 1, 0, 0, 0, 0]  [0, 1, 1, 0, 1, 1]  [1, 0, 1, 0, 1, 0]  [1, 0, 0, 0, 0, 1]  [1, 1, 1, 1, 0, 0]  [1, 1, 0, 1, 1, 1] |
| 8 | [0, 0, 0, 1, 1, 1]  [0, 0, 1, 1, 0, 0]  [0, 1, 0, 0, 0, 1]  [0, 1, 1, 0, 1, 0]  [1, 0, 0, 0, 0, 0]  [1, 0, 1, 0, 1, 1]  [1, 1, 0, 1, 1, 0]  [1, 1, 1, 1, 0, 1] | [0, 0, 0, 1, 1, 1]  [0, 0, 1, 1, 0, 0]  [0, 1, 0, 0, 0, 1]  [0, 1, 1, 0, 1, 0]  [1, 0, 1, 0, 1, 1]  [1, 0, 0, 0, 0, 0]  [1, 1, 1, 1, 0, 1]  [1, 1, 0, 1, 1, 0] |

Наименьшее расстояние может быть найдено либо, сравнивая вектора из таблицы напрямую и считая число разных элементов, либо как наименьшее число ненулевых элементов в векторе среди всех векторов, кроме нулевого. **В данном случае это 3.**

Чтобы код мог обнаружить k ошибок, минимальное расстояние должно составлять k+1, в данном случае **код обнаруживает 2 ошибки**.

Чтобы код мог исправить k ошибок, минимальное расстояние должно составлять 2k + 1, в данном случае **код исправляет 1 ошибку**.

**В многочлене имеется ошибка**, во-первых, имеется не нулевой остаток () от деления на , во-вторых, это вектора нету среди множества кодовых слов, в-третьих, при умножении на проверочную матрицу появляется ненулевой элемент по модулю 2:



Данную ошибку **можно исправить.**

Первый способ — это найти ближайшие правильные коды, минимально отличающиеся от полученного вектора, если такой правильный код один, то использовать его, в данном случае получим этот вектор:

Второй способ заключается в подсчете количества ненулевых элементов в векторе остатка (). В данном случае он 1 и не превышает число ошибок, проверяемых кодом, поэтому чтобы исправить можно сложить этот вектор с полученным сообщением ( и ), тогда получим аналогичный результат.

**#3**

Группа должна состоять из чисел, взаимно простых с 27, т. е. не кратных 3:

Найдем порождающий элемент напрямую, перебирая все числа:

1 - не подходит

2 - **подходит**:

4 - не подходит:

5 - **подходит**:

7 – не подходит

8 - не подходит:

10 - не подходит:

11 - **подходит**:

13 - не подходит:

14 - **подходит:**

16 - не подходит:

17 - не подходит:

19 - не подходит:

20 - **подходит**:

22 - не подходит:

23 - **подходит**:

25 - не подходит:

26 - не подходит

Итого:

**#4**

(не совпадает формула в 4-й части алгоритма: формула по лекции дает дробную степень, брал формулу из интернета)

Посчитать символ Якоби:

**По определению:**

Символ Лежандра:

Разложим 539:

Получаем, что

**По алгоритму:**

1.1)

1.2)

1.3)

1.4)

2.1)

3.1)

3.2)

3.3)

3.4)

4.1)

5.1)

5.2)

5.3)

5.4)

6.1)

7.1)

7.2)

8.1)

8.2)

Т. к. при рекурсивных вызовах знак не менялся

**#5**

Алгоритм Соловея-Штрассена работает с числами:

В данной задаче это

Число является составным по алгоритму в двух случаях:

1. Если , тогда подойдут:
2. Если

Квадратичные вычеты 3:

Квадратичные вычеты 7:

a = 2:

a = 3:

a = 4:

a = 5:

a = 6:

a = 7:

a = 8:

a = 9:

a = 10:

a = 11:

a = 12:

a = 13:

a = 14:

a = 15:

a = 16:

a = 17:

a = 18:

a = 19:

**a = 20:**

Таким образом, по второму критерию подходит только ***20***

**#6**

Построим вообще все точки их никак не больше 25:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
|  | 0 | 1 | 4 | 4 | 1 |
|  | 4 | 4 | 0 | 3 | 4 |

Найдем такие пары (x, y), где выполняется равенство :

Проверим, что это множество является абелевой группой (по доказанному на лекции операция сложения коммутативна и ассоциативна, нейтральный элемент - бесконечно удаленная точка, остается проверить наличие противоположно элемента и замкнутости множества относительно операции сложения):

1) **Проверим наличие противоположного элемента и добавим нейтральный элемент - бесконечно удаленную точку :**

Таким образом, для каждого элемента имеется противоположный элемент, также есть нейтральный.

2) **Проверим замыкание множества, относительно операции сложения:**

Попарно просуммируем множество точек (различных) по правилу:

При разных точках (не учитывая противоположные):

При одинаковых (не учитывая противоположные):

Обозначим , как null

(0, 2) + (0, 2) = (1, 2)

(0, 2) + (0, 3) = null

(0, 2) + (1, 2) = (4, 3)

(0, 2) + (1, 3) = (0, 3)

(0, 2) + (2, 0) = (4, 2)

(0, 2) + (4, 2) = (1, 3)

(0, 2) + (4, 3) = (2, 0)

(0, 3) + (0, 2) = null

(0, 3) + (0, 3) = (1, 3)

(0, 3) + (1, 2) = (0, 2)

(0, 3) + (1, 3) = (4, 2)

(0, 3) + (2, 0) = (4, 3)

(0, 3) + (4, 2) = (2, 0)

(0, 3) + (4, 3) = (1, 2)

(1, 2) + (0, 2) = (4, 3)

(1, 2) + (0, 3) = (0, 2)

(1, 2) + (1, 2) = (2, 0)

(1, 2) + (1, 3) = null

(1, 2) + (2, 0) = (1, 3)

(1, 2) + (4, 2) = (0, 3)

(1, 2) + (4, 3) = (4, 2)

(1, 3) + (0, 2) = (0, 3)

(1, 3) + (0, 3) = (4, 2)

(1, 3) + (1, 2) = null

(1, 3) + (1, 3) = (2, 0)

(1, 3) + (2, 0) = (1, 2)

(1, 3) + (4, 2) = (4, 3)

(1, 3) + (4, 3) = (0, 2)

(2, 0) + (0, 2) = (4, 2)

(2, 0) + (0, 3) = (4, 3)

(2, 0) + (1, 2) = (1, 3)

(2, 0) + (1, 3) = (1, 2)

(2, 0) + (2, 0) = null

(2, 0) + (4, 2) = (0, 2)

(2, 0) + (4, 3) = (0, 3)

(4, 2) + (0, 2) = (1, 3)

(4, 2) + (0, 3) = (2, 0)

(4, 2) + (1, 2) = (0, 3)

(4, 2) + (1, 3) = (4, 3)

(4, 2) + (2, 0) = (0, 2)

(4, 2) + (4, 2) = (1, 2)

(4, 2) + (4, 3) = null

(4, 3) + (0, 2) = (2, 0)

(4, 3) + (0, 3) = (1, 2)

(4, 3) + (1, 2) = (4, 2)

(4, 3) + (1, 3) = (0, 2)

(4, 3) + (2, 0) = (0, 3)

(4, 3) + (4, 2) = null

(4, 3) + (4, 3) = (1, 3)

**Новых точек получено не было, следовательно множество является замкнутым, значит была построена группа:**

3) Определим является ли она циклической:

Проверим какая точка является порождающей:

(0, 2) - **порождающая**

(0, 3) - **порождающая**

(1, 2) - не порождающая

(1, 3) - не порождающая

(2, 0) - не порождающая

(4, 2) - **порождающая**

(4, 3) - **порождающая**

- не порождающая

- **порождающие точки,** они есть, значит группа **циклическая**