

## Лабораторная работа 1.

### 1.1. Реализовать функцию REF(), приводящую матрицу к ступенчатому виду.

Ступенчатой матрицей, или матрицей ступенчатого вида по строкам, называется матрица, такая что

- все ненулевые строки (имеющие по крайней мере один ненулевой элемент) располагаются над всеми чисто нулевыми строками
- ведущий элемент (первый, считая слева направо, ненулевой элемент строки) каждой ненулевой строки располагается строго правее ведущего элемента в строке, расположенной выше данной.

[	1	0	1	1	0	0	1	0	0	1]
[	0	0	0	1	1	1	0	1	0	0]
[	0	0	0	0	1	0	0	1	0	0]
[	0	0	0	0	0	0	1	0	0	1]
[	0	0	0	0	0	0	0	1	1	1]

Бирюзовым цветом указаны ведущие элементы, жёлтым – элементы под ведущими, зелёным – над ведущими. Обратите внимание, что все элементы, отмеченные жёлтым – равны нулю, но часть зелёных – нет.

Допустимо пользоваться питру для базовых операций с матрицами. Значения элементов матрицы должны быть булевскими либо целочисленными. В первом случае сложение выполняется как исключающее ИЛИ (XOR). Во второй операции над строками должны выполняться по модулю 2. Полностью нулевые строки можно удалить.

Обратите внимание, что в зависимости от разработанного вами алгоритма конкретные значения строк и их порядок в полученной матрице может меняться.

### 1.2. Реализовать функцию RREF(), приводящую матрицу к приведённому ступенчатому виду.

Ступенчатая матрица называется приведенной, если матрица, составленная из всех ее основных столбцов, является единичной матрицей (столбец матрицы называется основным, если он содержит ведущий элемент какой-либо строки матрицы).

То есть, приведенная ступенчатая матрица не имеет нулевых строк, и все ведущие элементы ее строк равны единице. При этом все элементы основных столбцов, помимо ведущих элементов, являются нулями.

[	1	0	1	0	0	1	0	1	0	0]
[	0	0	0	1	0	1	0	0	0	1]
[	0	0	0	0	1	0	0	1	0	0]
[	0	0	0	0	0	0	1	0	0	1]
[	0	0	0	0	0	0	0	1	1	1]

Бирюзовым цветом указаны ведущие элементы, жёлтым – элементы под ведущими, зелёным – над ведущими. Обратите внимание, что все элементы, отмеченные жёлтым и зелёным – равны нулю.

См. уточнение к заданию 1.

### 1.3. Создать класс линейных кодов LinearCode.

Для инициализации линейного кода используется множество векторов одной длины (можно представить в форме матрицы).

#### 1.3.1 На основе входной матрицы сформировать порождающую матрицу в ступенчатом виде.

*Тестовый пример:*

```
Input:
S =
[[1 0 1 1 0 0 0 1 0 0 1]
 [0 0 0 1 1 1 0 1 0 1 0]
 [0 0 0 0 1 0 0 1 0 0 1]
 [1 0 1 0 1 1 1 0 0 0 1]
 [0 0 0 0 1 0 0 1 1 1 0]
 [1 0 1 1 1 0 0 0 0 0 0]]

Result:
S_REF =
[[1 0 1 1 0 0 0 1 0 0 1]
 [0 0 0 1 1 1 0 1 0 1 0]
 [0 0 0 0 1 0 0 1 0 0 1]
 [0 0 0 0 0 0 1 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0]]
```

#### 1.3.2 Задать $n$ равное числу столбцов и $k$ равное числу строк полученной матрицы (без учёта полностью нулевых строк).

*Тестовый пример:*

```
Input:
G =
[[1 0 1 1 0 0 0 1 0 0 1]
 [0 0 0 1 1 1 0 1 0 1 0]
 [0 0 0 0 1 0 0 1 0 0 1]
 [0 0 0 0 0 0 1 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 1 1]]

Result:
n = 11
k = 5
```

**1.3.3 Сформировать проверочную матрицу на основе порождающей.**  
Для этого выполнить следующие шаги.

**Шаг 1. Сформировать матрицу  $G^*$  в приведённом ступенчатом виде на основе порождающей.**

*Тестовый пример (Шаг 1):*

```
Input:
G =
[[1 0 1 1 0 0 0 1 0 0 1]
 [0 0 0 1 1 1 0 1 0 1 0]
 [0 0 0 0 1 0 0 1 0 0 1]
 [0 0 0 0 0 0 1 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 1 1]]

Result:
G* =
[[1 0 1 0 0 1 0 1 0 1 0]
 [0 0 0 1 0 1 0 0 0 1 1]
 [0 0 0 0 1 0 0 1 0 0 1]
 [0 0 0 0 0 0 1 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 1 1]]
```

**Шаг 2. Зафиксировать ведущие столбцы *lead* матрицы  $G^*$ .**

*Тестовый пример (Шаг 2):*

```
Input:
G* =
[[1 0 1 0 0 1 0 1 0 1 0]
 [0 0 0 1 0 1 0 0 0 1 1]
 [0 0 0 0 1 0 0 1 0 0 1]
 [0 0 0 0 0 0 1 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 1 1]]

Result:
lead = [0, 3, 4, 6, 8]
```

**Шаг 3. Сформировать сокращённую матрицу  $X$ , удалив ведущие столбцы матрицы  $G^*$ .**

*Тестовый пример (Шаг 3):*

```
Input:
[[1 0 1 0 0 1 0 1 0 1 0]
 [0 0 0 1 0 1 0 0 0 1 1]
 [0 0 0 0 1 0 0 1 0 0 1]
 [0 0 0 0 0 0 1 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 1 1]]

Result:
X =
[[0 1 1 1 1 0]
 [0 0 1 0 1 1]
 [0 0 0 1 0 1]
 [0 0 0 0 1 0]
 [0 0 0 0 1 1]]
```

**Шаг 4. Сформировать матрицу  $H$ , поместив в строки, соответствующие позициям ведущих столбцов строки из  $X$ , а в остальные – строки единичной матрицы.**

*Тестовый пример (Шаг 4):*

Input:

$X =$

```
[
[0 1 1 1 1 0]
[0 0 1 0 1 1]
[0 0 0 1 0 1]
[0 0 0 0 1 0]
[0 0 0 0 1 1]]
```

$I =$

```
[
[1 0 0 0 0 0]
[0 1 0 0 0 0]
[0 0 1 0 0 0]
[0 0 0 1 0 0]
[0 0 0 0 1 0]
[0 0 0 0 0 1]]
```

lead = [0, 3, 4, 6, 8]

Result:

[0 1 1 1 1 0]	[0] from X
[1 0 0 0 0 0]	[1] from I
[0 1 0 0 0 0]	[2] from I
[0 0 1 0 1 1]	[3] from X
[0 0 0 1 0 1]	[4] from X
[0 0 1 0 0 0]	[5] from I
[0 0 0 0 1 0]	[6] from X
[0 0 0 1 0 0]	[7] from I
[0 0 0 0 1 1]	[8] from X
[0 0 0 0 1 0]	[9] from I
[0 0 0 0 0 1]	[10] from I

**1.4. Сформировать все кодовые слова длины  $n$  двумя способами.**

**1.4.1 Сложить все слова из порождающего множества, оставить неповторяющиеся.**

**1.4.2 Взять все двоичные слова длины  $k$ , умножить каждое на  $G$ .**

Убедиться, что полученные множества кодовых слов совпадают. Убедиться, что умножение кодовых слов на проверочную матрицу в результате даёт нулевые вектора

*Тестовый пример:*

Input:

$G =$

```
[
[1 0 1 1 0 0 0 1 0 0 1]
[0 0 0 1 1 1 0 1 0 1 0]
[0 0 0 0 1 0 0 1 0 0 1]
[0 0 0 0 0 0 1 0 0 1 0]
[0 0 0 0 0 0 0 0 1 1 1]]
```

$H =$

```
[
[0 1 1 1 1 0]
[1 0 0 0 0 0]
[0 1 0 0 0 0]
[0 0 1 0 1 1]
[0 0 0 1 0 1]
[0 0 1 0 0 0]
[0 0 0 0 1 0]
[0 0 0 1 0 0]
[0 0 0 0 1 1]
[0 0 0 0 1 0]
[0 0 0 0 0 1]]
```

```

Result:
u = [1 0 1 1 0]
v = u@G = [1 0 1 1 1 0 1 0 0 1 0]
v@H = [0 0 0 0 0 0]

```

#### 1.4 Вычислить кодовое расстояние получившегося кода.

Сделать вывод о кратности обнаруживаемой ошибки  $t$ .

*Тестовый пример:*

```

Input:
G =
[[1 0 1 1 0 0 0 1 0 0 1]
 [0 0 0 1 1 1 0 1 0 1 0]
 [0 0 0 0 1 0 0 1 0 0 1]
 [0 0 0 0 0 0 1 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 1 1]]
n = 11
k = 5

Result:
d = 2
t = 1

```

**1.4.1** Внести в кодовое слово ошибку кратности не более  $t$ , умножить полученное слово на  $H$ , убедиться в обнаружении ошибки.

**1.4.2** Найти для некоторого кодового слова ошибку кратности  $t+1$  такую, что при умножении на  $H$  ошибка не может быть обнаружена.

*Тестовый пример:*

```

Input:
G =
[[1 0 1 1 0 0 0 1 0 0 1]
 [0 0 0 1 1 1 0 1 0 1 0]
 [0 0 0 0 1 0 0 1 0 0 1]
 [0 0 0 0 0 0 1 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 1 1]]

H =
[[0 1 1 1 1 0]
 [1 0 0 0 0 0]
 [0 1 0 0 0 0]
 [0 0 1 0 1 1]
 [0 0 0 1 0 1]
 [0 0 1 0 0 0]
 [0 0 0 0 1 0]
 [0 0 0 1 0 0]
 [0 0 0 0 1 1]
 [0 0 0 0 1 0]
 [0 0 0 0 0 1]]

v = [1 0 1 1 1 0 1 0 0 1 0]

Result:

e1 = [0 0 1 0 0 0 0 0 0 0 0]
v + e1 = [1 0 0 1 1 0 1 0 0 1 0]
(v + e1)@H = [0 1 0 0 0 0] - error

```

```
e2 = [[0 0 0 0 1 0 0 1 0 0]]  
v + e2 = [[1 0 1 1 0 0 1 1 0 1]]  
(v + e2)*H = [0 0 0 0 0]      - no error
```