# Глава 4

In [176]:

```
import numpy as np
import math
import matplotlib.pyplot as plt
import scipy
import scipy.stats
import time

from tqdm import tqdm
from collections import defaultdict

from typing import List, Set
```

In [2]:

```
BitWord = List[int]
```

In [3]:

```
# generate all sequences of length l
def generate(l):
    res = []

    for i in range(0, 2**l):
        b = bin(i)[2:]
        b = '0' * (l - len(b)) + b
        b = np.array(list(map(lambda x: int(x), b)))
        res.append(b)

    return np.array(res)
```

# Задание 2

In [169]:

```
n = 10
k = 6
r = 4
H = np.array([
    [0, 0, 0, 1, 1, 1, 1, 0, 1, 0],
    [1, 0, 0, 0, 0, 0, 1, 1, 1, 1],
    [1, 1, 1, 0, 1, 1, 1, 0, 0, 1],
    [1, 0, 1, 1, 0, 1, 1, 1, 0, 0]
])

G = np.array([
        [0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
        [1, 1, 0, 0, 0, 0, 0, 1, 0, 0],
        [0, 0, 1, 0, 0, 0, 1, 0, 1, 0],
        [0, 0, 0, 1, 0, 0, 0, 1, 1, 0],
        [0, 0, 0, 0, 1, 0, 1, 1, 0, 0],
        [1, 0, 0, 0, 0, 1, 0, 0, 1, 0]
])

# Минимальная спэновая форма
G = np.array([
        [1, 0, 0, 1, 1, 1, 1, 0, 0, 0],
        [0, 1, 0, 1, 0, 1, 0, 0, 0, 0],
        [0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 1, 1, 0],
        [0, 0, 0, 0, 1, 0, 1, 1, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
])


p0 = 1e-5

all_words = generate(n)

code_words = []
for word in all_words:
    if np.all(np.dot(word, H.T) % 2 == 0):
        code_words.append(word)
```

In [170]:

```
np.dot(G, H.T) % 2
```

Out[170]:

```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

In [6]:

```
def L(y):
    if y == 1:
        return math.log2((1 - p0) / p0)
    else:
        return math.log2(p0/(1 - p0))

L = np.vectorize(L)
```

In [7]:

```
# Множитель 4 * sqrt(E) / N0 не влияет, поэтому просто возвращаем y
def L_noisy(y):
    return y

L_noisy = np.vectorize(L_noisy)
```

In [8]:
```
def mp_decode(L, y: BitWord) -> BitWord:
    y = L(y)
    mx = np.sum(-L(np.ones(n)))
    c = np.zeros(n)

    for cur in code_words:
        if np.dot(cur, y) > mx:
            mx = np.dot(cur, y)
            c = cur

    return c
```

In [9]:
```
def mav_decode(L, y: BitWord) -> BitWord:
    mx = np.sum(-L(np.ones(n)))
    c = np.zeros(n)

    for cur in code_words:
        if np.dot(y, L(cur)) > mx:
            mx = np.dot(y, L(cur))
            c = cur

    return c
```

In [10]:
```
print(mp_decode(L, np.array([1,1,1,1,1,0,0,0,0,0])))
print(mp_decode(L_noisy, np.array([1,1,1,1,1,0,0,0,0,0])))

print(mav_decode(L, np.array([1,1,1,1,1,0,0,0,0,0])))
print(mav_decode(L_noisy, np.array([1,1,1,1,1,0,0,0,0,0])))
```

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

In [11]:
```
def get_prob_of_error(decode_algo, p0):
    prob_of_error = 0
    for c in code_words:
        for e in all_words:
            current_prob = (p0 ** np.sum(e)) * ((1 - p0) ** (n - np.sum(e)))
            if (prob_of_error / current_prob >= (1/p0)): # will not affect erro
r
                continue

            if not np.all(np.equal(decode_algo((c + e) % 2), c)):
                prob_of_error += current_prob

    return prob_of_error / len(code_words)
```

In [15]:
```
no_decode_l = lambda y : y

mp_decode_l = lambda y : mp_decode(L, y)
mp_decode_l_noisy = lambda y : mp_decode(L_noisy, y)

mav_decode_l = lambda y : mav_decode(L, y)
mav_decode_l_noisy = lambda y : mav_decode(L_noisy, y)
```

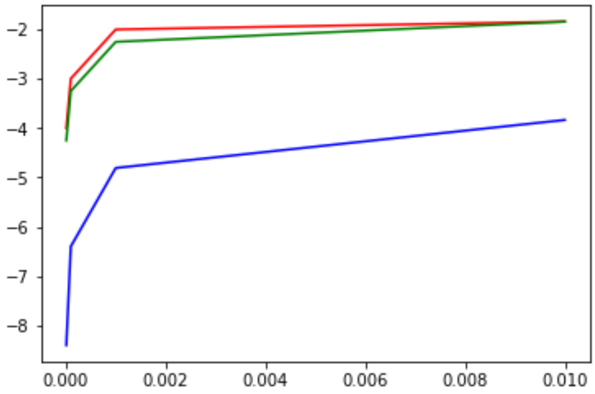
In [33]:
```
p0s = 1 / np.power(10, np.arange(2,6))

def get_errors(decode):
    errors = []
    for p in tqdm(p0s):
        errors.append(get_prob_of_error(decode, p))
    return errors
```

## Зависимость log вероятности ошибки от p0 для различных методов (ДСК)

```
In [34]: plt.plot(p0s, np.log10(get_errors(no_decode_l)), color='r')
         plt.plot(p0s, np.log10(get_errors(mp_decode_l)), color='b')
         plt.plot(p0s, np.log10(get_errors(mav_decode_l)), color='g')
         plt.show()
```

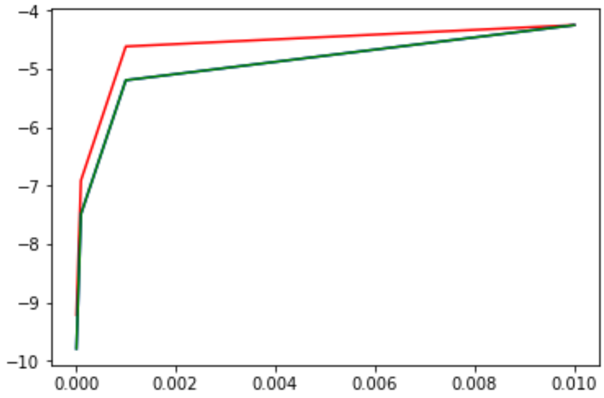
```
100%|██████████| 4/4 [00:02<00:00,  1.51it/s]
100%|██████████| 4/4 [00:03<00:00,  1.05it/s]
100%|██████████| 4/4 [00:07<00:00,  1.68s/it]
```



## Зависимость log вероятности ошибки от p0 для различных методов (АБГШ)

```
In [35]: plt.plot(p0s, np.log(get_errors(no_decode_l)), color='r')
         plt.plot(p0s, np.log(get_errors(mp_decode_l_noisy)), color='b')
         plt.plot(p0s, np.log(get_errors(mav_decode_l_noisy)), color='g')
         plt.show()
```

```
100%|██████████| 4/4 [00:02<00:00,  1.70it/s]
100%|██████████| 4/4 [00:02<00:00,  1.43it/s]
100%|██████████| 4/4 [00:06<00:00,  1.44s/it]
```



## Энергетический выигрыш кодирования

Рассмотрим для примера выигрыш кодирования при декодировании по максимуму правдоподбия и $p_0 = 10^{-5}$:

```
In [ ]: get_prob_of_error(no_decode_l, 1e-5)
```

Тоесть при ошибке на бит $p_0 = 10^{-5}$ вероятность ошибки декодера составит $10^{-4}$. Посчитаем, какая $p_0$ нужна, чтобы добится такой ошибки с помощью алгоритма декодирования по максимуму правдоподбия:

```
In [ ]: for p in range(1, 4):
            print(get_prob_of_error(mp_decode_l, 10**(-p)))
```

Видим, что достаточно $p_0 \approx 10^{-2}$

Тогда выигрыш кодирования составит:

Без кодирования: $p = 1 * 10^{-5} \frac{E_b}{N_0} \approx 9.5\text{дБ}$

С кодированием: $p = 1 * 10^{-2} \frac{E_b}{N_0} = 4.3\text{дБ}$

Выигрыш кодирования = $9.5 - \frac{4.3}{R} = 9.5 - 7.2 = 2.3\text{дБ}$

# Задание 4

Граф для решетки по порождающей матрице нарисовал вручную и приложил в отдельном файле. Граф для решетки по проверочной матрице сгенерировал кодом и попытался красиво отобразить.

In [198]:

```
nodes = {() : [0,0,0,0]}

def is_code_word_prefix(code_words, check_word):
    size = len(check_word)
    for word in code_words:
        if (np.all(np.array(word[:size]) == np.array(check_word))):
            return True
    return False

def process_word(word_p, word_c, next_line):
    pref = np.dot(word_c, H[:, :i+1].T) % 2
    if is_code_word_prefix(code_words, word_c):
        next_line.append(word_c)
        nodes[tuple(word_c)] = pref
        print('Node {} -> Node {}. Current sequence: {}'.format(nodes[tuple(word_p)], pref, word_c))

prev_line = [[]]
for i in range(0, n):
    print('Level ', i)
    next_line = []
    for word in prev_line:
        process_word(word, [*word, 0], next_line)
        process_word(word, [*word, 1], next_line)
    prev_line = next_line
```

```
Level  0
Node [0, 0, 0, 0] -> Node [0 0 0 0]. Current sequence: [0]
Node [0, 0, 0, 0] -> Node [0 1 1 1]. Current sequence: [1]
Level  1
Node [0 0 0 0] -> Node [0 0 0 0]. Current sequence: [0, 0]
Node [0 0 0 0] -> Node [0 0 1 0]. Current sequence: [0, 1]
Node [0 1 1 1] -> Node [0 1 1 1]. Current sequence: [1, 0]
Node [0 1 1 1] -> Node [0 1 0 1]. Current sequence: [1, 1]
Level  2
Node [0 0 0 0] -> Node [0 0 0 0]. Current sequence: [0, 0, 0]
Node [0 0 0 0] -> Node [0 0 1 1]. Current sequence: [0, 0, 1]
Node [0 0 1 0] -> Node [0 0 1 0]. Current sequence: [0, 1, 0]
Node [0 0 1 0] -> Node [0 0 0 1]. Current sequence: [0, 1, 1]
Node [0 1 1 1] -> Node [0 1 1 1]. Current sequence: [1, 0, 0]
Node [0 1 1 1] -> Node [0 1 0 0]. Current sequence: [1, 0, 1]
Node [0 1 0 1] -> Node [0 1 0 1]. Current sequence: [1, 1, 0]
Node [0 1 0 1] -> Node [0 1 1 0]. Current sequence: [1, 1, 1]
Level  3
Node [0 0 0 0] -> Node [0 0 0 0]. Current sequence: [0, 0, 0, 0]
Node [0 0 0 0] -> Node [1 0 0 1]. Current sequence: [0, 0, 0, 1]
Node [0 0 1 1] -> Node [0 0 1 1]. Current sequence: [0, 0, 1, 0]
Node [0 0 1 1] -> Node [1 0 1 0]. Current sequence: [0, 0, 1, 1]
Node [0 0 1 0] -> Node [0 0 1 0]. Current sequence: [0, 1, 0, 0]
Node [0 0 1 0] -> Node [1 0 1 1]. Current sequence: [0, 1, 0, 1]
Node [0 0 0 1] -> Node [0 0 0 1]. Current sequence: [0, 1, 1, 0]
Node [0 0 0 1] -> Node [1 0 0 0]. Current sequence: [0, 1, 1, 1]
Node [0 1 1 1] -> Node [0 1 1 1]. Current sequence: [1, 0, 0, 0]
Node [0 1 1 1] -> Node [1 1 1 0]. Current sequence: [1, 0, 0, 1]
Node [0 1 0 0] -> Node [0 1 0 0]. Current sequence: [1, 0, 1, 0]
Node [0 1 0 0] -> Node [1 1 0 1]. Current sequence: [1, 0, 1, 1]
Node [0 1 0 1] -> Node [0 1 0 1]. Current sequence: [1, 1, 0, 0]
Node [0 1 0 1] -> Node [1 1 0 0]. Current sequence: [1, 1, 0, 1]
Node [0 1 1 0] -> Node [0 1 1 0]. Current sequence: [1, 1, 1, 0]
Node [0 1 1 0] -> Node [1 1 1 1]. Current sequence: [1, 1, 1, 1]
Level  4
Node [0 0 0 0] -> Node [0 0 0 0]. Current sequence: [0, 0, 0, 0, 0]
Node [0 0 0 0] -> Node [1 0 1 0]. Current sequence: [0, 0, 0, 0, 1]
Node [1 0 0 1] -> Node [1 0 0 1]. Current sequence: [0, 0, 0, 1, 0]
Node [1 0 0 1] -> Node [0 0 1 1]. Current sequence: [0, 0, 0, 1, 1]
Node [0 0 1 1] -> Node [0 0 1 1]. Current sequence: [0, 0, 1, 0, 0]
Node [0 0 1 1] -> Node [1 0 0 1]. Current sequence: [0, 0, 1, 0, 1]
Node [1 0 1 0] -> Node [1 0 1 0]. Current sequence: [0, 0, 1, 1, 0]
Node [1 0 1 0] -> Node [0 0 0 0]. Current sequence: [0, 0, 1, 1, 1]
Node [0 0 1 0] -> Node [0 0 1 0]. Current sequence: [0, 1, 0, 0, 0]
Node [0 0 1 0] -> Node [1 0 0 0]. Current sequence: [0, 1, 0, 0, 1]
Node [1 0 1 1] -> Node [1 0 1 1]. Current sequence: [0, 1, 0, 1, 0]
Node [1 0 1 1] -> Node [0 0 0 1]. Current sequence: [0, 1, 0, 1, 1]
Node [0 0 0 1] -> Node [0 0 0 1]. Current sequence: [0, 1, 1, 0, 0]
Node [0 0 0 1] -> Node [1 0 1 1]. Current sequence: [0, 1, 1, 0, 1]
Node [1 0 0 0] -> Node [1 0 0 0]. Current sequence: [0, 1, 1, 1, 0]
Node [1 0 0 0] -> Node [0 0 1 0]. Current sequence: [0, 1, 1, 1, 1]
Node [0 1 1 1] -> Node [0 1 1 1]. Current sequence: [1, 0, 0, 0, 0]
Node [0 1 1 1] -> Node [1 1 0 1]. Current sequence: [1, 0, 0, 0, 1]
Node [1 1 1 0] -> Node [1 1 1 0]. Current sequence: [1, 0, 0, 1, 0]
Node [1 1 1 0] -> Node [0 1 0 0]. Current sequence: [1, 0, 0, 1, 1]
Node [0 1 0 0] -> Node [0 1 0 0]. Current sequence: [1, 0, 1, 0, 0]
Node [0 1 0 0] -> Node [1 1 1 0]. Current sequence: [1, 0, 1, 0, 1]
Node [1 1 0 1] -> Node [1 1 0 1]. Current sequence: [1, 0, 1, 1, 0]
Node [1 1 0 1] -> Node [0 1 1 1]. Current sequence: [1, 0, 1, 1, 1]
Node [0 1 0 1] -> Node [0 1 0 1]. Current sequence: [1, 1, 0, 0, 0]
Node [0 1 0 1] -> Node [1 1 1 1]. Current sequence: [1, 1, 0, 0, 1]
Node [1 1 0 0] -> Node [1 1 0 0]. Current sequence: [1, 1, 0, 1, 0]
Node [1 1 0 0] -> Node [0 1 1 0]. Current sequence: [1, 1, 0, 1, 1]
Node [0 1 1 0] -> Node [0 1 1 0]. Current sequence: [1, 1, 1, 0, 0]
Node [0 1 1 0] -> Node [1 1 0 0]. Current sequence: [1, 1, 1, 0, 1]
Node [1 1 1 1] -> Node [1 1 1 1]. Current sequence: [1, 1, 1, 1, 0]
Node [1 1 1 1] -> Node [0 1 0 1]. Current sequence: [1, 1, 1, 1, 1]
Level  5
Node [0 0 0 0] -> Node [0 0 0 0]. Current sequence: [0, 0, 0, 0, 0, 0]
Node [1 0 1 0] -> Node [1 0 1 0]. Current sequence: [0, 0, 0, 0, 1, 0]
Node [1 0 0 1] -> Node [1 0 0 1]. Current sequence: [0, 0, 0, 1, 0, 0]
```

```
In [ ]:
```