Практическое задание. Коды БЧХ.

Голдобина Ольга, 317 гр.

Часть 1

Необходимо было реализовать основные оберащи в поле F_2^n и $F_2^n[x]$, поиск минимального многочлена, расширенный алгоритм евклида, а так же реализовать класс BCH, в котором осуществляется построение кода по заданным параметрам, кодирование и декодирование. Код реализован в файлах gf.py и bch.py.

In [1]:
```
import bch
```

Найдём БЧХ-код, для которого истинное расстояние будет больше, чем величина 2t+ 1, заданная при построении кода

In [935]:
```
print('{0}\t{1}\t{2}\t{3}'.format('n', 't', '2t + 1', 'real distance'))
for t in range(1, 8):
    b = bch.BCH(15, t)
    print('{0:d}\t{1:d}\t{2:d}\t{3:d}'.format(7, t, 2 * t + 1, b.dist())
```

```
n       t       2t + 1  real distance
7       1       3       3
7       2       5       5
7       3       7       7
7       4       9       15
7       5       11      15
7       6       13      15
7       7       15      15
```

замечаем, что для значений t = 4,5,6 истинное расстояние кода больше, чем 2t + 1. при увеличении параметра t какое-то время расстояние остается постоянным.

Часть 2

Необходимо реализовать процедуру оценки доли правильно раскодированных со-общений, доли ошибочно раскодированных сообщений и доли отказов от декодирования для БЧХ-кода

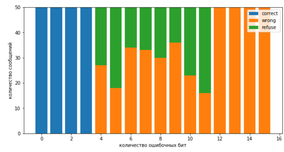
In [936]:
```
import matplotlib.pyplot as plt
from collections import defaultdict
import numpy as np
```

In [1026]:
```
def generate_err(V, num_err):
    W = V.copy()
    for i in range(W.shape[0]):
        idx = np.random.permutation(W.shape[1])[:num_err]
        W[i][idx] ^= 1
    return W
```

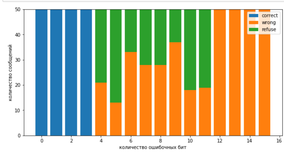
In [1058]:
```
def get_stat(n, b, mode='euclid'):
    correct = np.zeros((b.n + 1), dtype=np.int)
    wrong = np.zeros((b.n + 1), dtype=np.int)
    refuse = np.zeros((b.n + 1), dtype=np.int)

    for num_err in range(b.n + 1):
        U = np.random.randint(0, 2, (n, b.k))
        V = b.encode(U)
        W = generate_err(V, num_err)
        W_decoded = b.decode(W, mode)
        U_decoded = W_decoded[:, :b.k]

        refuse[num_err] = (U_decoded[:,0] < 0).sum()
        check = (U_decoded != U).sum(axis=1)
        correct[num_err] = (check == 0).sum()
        wrong[num_err] = (check != 0).sum() - refuse[num_err]

    return correct, wrong, refuse
```

In [1076]:
```
def show_stat(correct, wrong, refuse, b):
    plt.figure(figsize=(10,5))

    plt.bar(np.arange(0, correct.shape[0]), correct, bottom=0, label='correct')
    plt.bar(np.arange(0, wrong.shape[0]), wrong, bottom=correct, label='wrong')
    plt.bar(np.arange(0, refuse.shape[0]), refuse, bottom=correct + wrong, label='refuse')

    plt.legend()
    plt.xlabel('количество ошибочных бит')
    plt.ylabel('количество сообщений')

    plt.show()
```

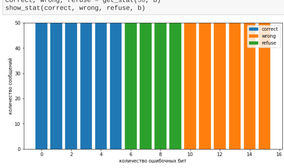
In [1077]:
```
b = bch.BCH(15,3)
correct, wrong, refuse = get_stat(50, b)
show_stat(correct, wrong, refuse, b)
```



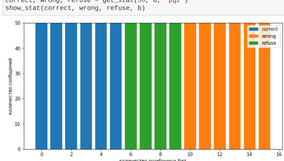
In [1078]:
```
b = bch.BCH(15,3)
correct, wrong, refuse = get_stat(50, b, 'pgz')
show_stat(correct, wrong, refuse, b)
```



In [1079]:
```
b = bch.BCH(15,5)
correct, wrong, refuse = get_stat(50, b)
show_stat(correct, wrong, refuse, b)
```



In [1080]:
```
b = bch.BCH(15,5)
correct, wrong, refuse = get_stat(50, b, 'pgz')
show_stat(correct, wrong, refuse, b)
```



убеждаемся, что бчх-код действительно исправляет до t ошибок

Проведём сравнение двух методов декодирования по времени работы

In [1074]:
```
from time import time
```

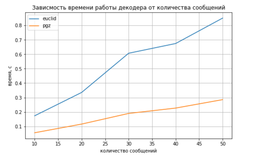
In [1075]:
```
def timing_test(block_size, b):

    U = np.random.randint(0,2, (block_size, b.k))
    V = b.encode(U)
    W = generate_err(V, b.t)

    t_euclid = time()
    b.decode(W)
    t_euclid = time() - t_euclid

    t_pgz = time()
    b.decode(W, 'pgz')
    t_pgz = time() - t_pgz

    return t_euclid, t_pgz
```

In [1087]:
```
code_size = [2** q - 1 for q in range(2,8)]
euclid_time = []
pgz_time = []

for n in code_size:
    b = bch.BCH(n, (n - 1) // 2)
    t_euclid, t_pgz = timing_test(50, b)
    euclid_time.append(t_euclid)
    pgz_time.append(t_pgz)
```

In [1089]:
```
plt.figure(figsize=(8,5))
plt.plot(code_size, euclid_time, label='euclid')
plt.plot(code_size, pgz_time, label='pgz')
plt.grid()
plt.legend()
plt.xlabel('размер кода')
plt.xticks(code_size)
plt.ylabel('время, с')
plt.title('Зависимость времени работы декодера от размера кода')
plt.show()
```

In [1090]:
```
block_size = [i for i in range(10, 51, 10)]
euclid_time = []
pgz_time = []

b = bch.BCH(15, 5)
for bs in block_size:
    t_euclid, t_pgz = timing_test(bs, b)
    euclid_time.append(t_euclid)
    pgz_time.append(t_pgz)
```

In [1091]:
```
plt.figure(figsize=(8,5))
plt.plot(block_size, euclid_time, label='euclid')
plt.plot(block_size, pgz_time, label='pgz')
plt.grid()
plt.legend()
plt.xlabel('количество сообщений')
plt.ylabel('время, с')
plt.title('Зависимость времени работы декодера от количества сообщений')
plt.show()
```



как видим, леторитм PGZ работает значительно быстрее, чем Euclid.

Таким образом, был реализован алгоритм построения кода БЧХ и исследовано время работы различных алгоритмов декодирования.