Zadanie 1

Aby pokazać, że odległość Hamminga jest metryka sprawdze wszystkie warunki.
1. $D_{(x,y)} = 0 \Leftrightarrow x = y$
2. $D_{(x,y)} = D_{(y,x)}$
3. $D_{(x,z)} \leq D_{(x,y)} + D_{(y,z)}$

1.1 $\Leftarrow$
$x = (x_1, x_2, ..., x_n) = (y_1, y_2, ...y_n) = y$
$D_{(x,y)} = D_{(x_1,y_1)} + D_{(x_2,y_2)} + ... + D_{(x_n,y_n)}$
$x_i = y_i \Rightarrow D_{(x_i,y_i)} = 0 \Rightarrow D_{(x,y)} = 0$

1.2 $\Rightarrow$
$D_{(x,y)} = D_{(x_1,y_1)} + D_{(x_2,y_2)} + ... + D_{(x_n,y_n)} = 0$
Zatem:
$D_{(x_1,y_1)} = 0,\ D_{(x_2,y_2)} = 0, ...\ D_{(x_n,y_n)} = 0 \Rightarrow x_i = y_i \Rightarrow x = y$

2.$D_{(x,y)} = D_{(x,y)}$
Porównujemy zawsze dwa elementy z wektorów x i y o tym samym indeksie. Nie ma znaczenia, czy porównujemy x do y, czy y do x, ponieważ indeksacja w wektorach sie nie zmienia.

3.D$_{(x,z)} \leq D_{(x,y)} + D_{(y,z)}$
$zal : a, b, c \in N$

3.1 Niech x=z $\neq y$

$Wtedy : D_{(x,z)} = 0, D_{(x,y)} \geq 0, D_{(y,z)} \geq 0 \Rightarrow$
$D_{(x,z)} \leq D_{(x,y)} + D_{(y,z)}$ Prawda
3.2 x$\neq z$

$a)\ Niech : x = y, D_{(x,z)} = a$
D$_{(x,z)} \leq D_{(x,y)} + D_{(y,z)}$
$a \leq 0 + a\ \ Prawda$

$b)Niech\ x \neq y$
$D_{(x,z)} = a$
$D_{(x,y)} = b$
$D_{(y,z)} = c$
Jeżeli x i z różnia sie na "a" pozycjach, x i y na "b" pozycjach, to:
• minimalnie z i y różnia sie na $|a - b|$ pozycjach

$$|a - b| \leq c \Rightarrow a \leq b + c)$$

• maksymalnie z i y różnia sie na a+b pozycjach
$a + b \leq c \Rightarrow a + 2b \leq b + c \Rightarrow a \leq b + c$

Zatem $D_{(x,z)} \leq D_{(x,y)} + D_{(y,z)}$ jest spełnione. Odległość Hamminga jest metryka.

Zadanie 2

C - dowolny (n, k) kod liniowy, $B = (e_1, e_2, ..., e_k)$ - baza przestrzeni C. G - macierz generujaca kod C. Wtedy:

$$G = \begin{bmatrix} e_1^T \\ e_2^T \\ ... \\ e_k^T \end{bmatrix}$$

Kodowanie wyraża sie wzorem $w = (v^T G)^T \Leftrightarrow w^T = G^T v$, gdzie G składa sie ze skonkatynowanych kolumnowo wektorów bazy B, v $\in$ V, w - wynik kodowania.
Chcemy pokazać, że w $\in$ C. Iloraz $w^T = G^T v$ równoważny jest z obliczeniem kombinacji liniowej wektorów z B o współczynnikach odpowiadajacych kolejnym wartościom z wektora v. w jest komibnacja wektorów z bazy C $\Leftarrow$ należy do C, tzn. w jest słowem kodowym C.

Zadanie 3

(n,k) - kod liniowy C, v - słowo kodowe kodu C
Podczas odkodowywania w należy zauważyć, że:
$w \in C$, wiec stosujac algorytm MinimizeHammingDistance otrzymujemy $min\{d(v,w) : w \in C\} = 0$, zatem L={w}.
r - wektor ten bedzie miał wymiar k, posiada tyle współrzednych ile wektorów bazie kodu C, tym samym ile wierszy w macierzy generujacej G. Jego kolejne wspołrzedne mówia "ile" jakich wektorów potrzebujemy z bazy C.
Przy ponownym kodowaniu r mnożymy przez G. W pierwszej kolumnie G znajduja sie pierwsze współrzedne z kolejnych wektorów bazy C. Zatem na mnożenie $r \cdot G$ można spojrzeć, że pierwsza współrzedna z r mówi jaka cześć wektora pierwszego z bazy należy wziać do sumy. Druga współrzedna jaka cześć drugiego wektora należy wziać, itd. do k. Gdy zsumujemy te wektory to otrzymamy poczatkowy wektor w. Widać, że jest to proces odwrotny do wyznaczania r.

Zadanie 4

d - odledłośc Hamminga $d(u, v) = i[n]u_i v_i$, tzn liczba miejsc, na których wektory u i v sa różne. Niech a, b, c $\in C$a = [a$_1, a_2, ..., a_n]b = [b_1, b_2, ..., b_n]c = [c_1, c_2, ..., c_n]$

Chcemy pokazać, że $d(a, b) = d(a + x, b + x)$, czyli jeśli a i b maja różne wartości na x miejscach, to a+c i b+c też. Oczywiście tak jest, ponieważ $a_i = b_i \Leftrightarrow a_i + x_i b_i + x_i$, dla i $\in [n]$.

Zadania 5-8 w jezyku python.

Zadanie 5

Obliczamy odległość Hamminga dla wektorów $(1, 2, 0, 1)^T$ i $(0, 0, 0, 1)^T$:

```
v_1=(1,2,0,1)
v_2=(0,0,0,1)
licznik_1=0
for indeks in range(4):
    if v_1[indeks]!=v_2[indeks]: licznik_1+=1
print(licznik_1)
```

Dostajemy: 2.

Obliczamy odległości Hamminga każdego wektora z każdym, znajdujemy te, które maja wzgledem siebi najmniejsze odległości:

```
V=((1,2,1,2,0),(1,1,1,1,1),(0,0,2,1,1),(2,2,2,1,0))
licznik_2=0
min=5
wynik=[]
for i in range(len(V)):
    for j in range(i+1,len(V)):
        licznik_2=0
        for k in range(5):
            if V[i][k]!=V[j][k]:
                licznik_2+=1
        if licznik_2 <min:
            min=licznik_2
            wynik.clear
        if licznik_2 == min:
            wynik.append((i,j))

print(wynik)
```

Dostajemy: (0, 1), (0, 3), (1, 2), (2, 3), gdzie 0 - pierwszy wektor, 3 - wektor czwarty ze zbioru.

Zadanie 6

Aby otrzymać wszystkie możliwe słowa generujemy wszystkie kombinacje liniowe wektorów z bazy (kod jest nad ciałem Z_7, wiec skalary beda liczbami całkowitymi 0-6).

```
import numpy as np
import random as rand

all_coded_words = []

e1 = np.array([1, 0, 0, 2, 4])
e2 = np.array([0, 1, 0, 1, 0])
e3 = np.array([0, 0, 1, 5, 6])


for a in range(7):
    for b in range(7):
        for c in range(7):
            arr = a*e1 + b*e2 + c*e3
            arr = arr%7
            all_coded_words.append(arr.tolist())

for x in all_coded_words:
  print(x)

print(len(all_coded_words))
```

Wychodza 343 słowa kodowe. Oto one:

[0, 0, 0, 0, 0]	[0, 0, 1, 5, 6]	[0, 0, 2, 3, 5]	[0, 0, 3, 1, 4]	[0, 0, 4, 6, 3]	[0, 0, 5, 4, 2]	[0, 0, 6, 2, 1]
[0, 1, 0, 1, 0]	[0, 1, 1, 6, 6]	[0, 1, 2, 4, 5]	[0, 1, 3, 2, 4]	[0, 1, 4, 0, 3]	[0, 1, 5, 5, 2]	[0, 1, 6, 3, 1]
[0, 2, 0, 2, 0]	[0, 2, 1, 0, 6]	[0, 2, 2, 5, 5]	[0, 2, 3, 3, 4]	[0, 2, 4, 1, 3]	[0, 2, 5, 6, 2]	[0, 2, 6, 4, 1]
[0, 3, 0, 3, 0]	[0, 3, 1, 1, 6]	[0, 3, 2, 6, 5]	[0, 3, 3, 4, 4]	[0, 3, 4, 2, 3]	[0, 3, 5, 0, 2]	[0, 3, 6, 5, 1]
[0, 4, 0, 4, 0]	[0, 4, 1, 2, 6]	[0, 4, 2, 0, 5]	[0, 4, 3, 5, 4]	[0, 4, 4, 3, 3]	[0, 4, 5, 1, 2]	[0, 4, 6, 6, 1]
[0, 5, 0, 5, 0]	[0, 5, 1, 3, 6]	[0, 5, 2, 1, 5]	[0, 5, 3, 6, 4]	[0, 5, 4, 4, 3]	[0, 5, 5, 2, 2]	[0, 5, 6, 0, 1]
[0, 6, 0, 6, 0]	[0, 6, 1, 4, 6]	[0, 6, 2, 2, 5]	[0, 6, 3, 0, 4]	[0, 6, 4, 5, 3]	[0, 6, 5, 3, 2]	[0, 6, 6, 1, 1]
[1, 0, 0, 2, 4]	[1, 0, 1, 0, 3]	[1, 0, 2, 5, 2]	[1, 0, 3, 3, 1]	[1, 0, 4, 1, 0]	[1, 0, 5, 6, 6]	[1, 0, 6, 4, 5]
[1, 1, 0, 3, 4]	[1, 1, 1, 1, 3]	[1, 1, 2, 6, 2]	[1, 1, 3, 4, 1]	[1, 1, 4, 2, 0]	[1, 1, 5, 0, 6]	[1, 1, 6, 5, 5]
[1, 2, 0, 4, 4]	[1, 2, 1, 2, 3]	[1, 2, 2, 0, 2]	[1, 2, 3, 5, 1]	[1, 2, 4, 3, 0]	[1, 2, 5, 1, 6]	[1, 2, 6, 6, 5]
[1, 3, 0, 5, 4]	[1, 3, 1, 3, 3]	[1, 3, 2, 1, 2]	[1, 3, 3, 6, 1]	[1, 3, 4, 4, 0]	[1, 3, 5, 2, 6]	[1, 3, 6, 0, 5]
[1, 4, 0, 6, 4]	[1, 4, 1, 4, 3]	[1, 4, 2, 2, 2]	[1, 4, 3, 0, 1]	[1, 4, 4, 5, 0]	[1, 4, 5, 3, 6]	[1, 4, 6, 1, 5]
[1, 5, 0, 0, 4]	[1, 5, 1, 5, 3]	[1, 5, 2, 3, 2]	[1, 5, 3, 1, 1]	[1, 5, 4, 6, 0]	[1, 5, 5, 4, 6]	[1, 5, 6, 2, 5]
[1, 6, 0, 1, 4]	[1, 6, 1, 6, 3]	[1, 6, 2, 4, 2]	[1, 6, 3, 2, 1]	[1, 6, 4, 0, 0]	[1, 6, 5, 5, 6]	[1, 6, 6, 3, 5]
[2, 0, 0, 4, 1]	[2, 0, 1, 2, 0]	[2, 0, 2, 0, 6]	[2, 0, 3, 5, 5]	[2, 0, 4, 3, 4]	[2, 0, 5, 1, 3]	[2, 0, 6, 6, 2]
[2, 1, 0, 5, 1]	[2, 1, 1, 3, 0]	[2, 1, 2, 1, 6]	[2, 1, 3, 6, 5]	[2, 1, 4, 4, 4]	[2, 1, 5, 2, 3]	[2, 1, 6, 0, 2]
[2, 2, 0, 6, 1]	[2, 2, 1, 4, 0]	[2, 2, 2, 2, 6]	[2, 2, 3, 0, 5]	[2, 2, 4, 5, 4]	[2, 2, 5, 3, 3]	[2, 2, 6, 1, 2]
[2, 3, 0, 0, 1]	[2, 3, 1, 5, 0]	[2, 3, 2, 3, 6]	[2, 3, 3, 1, 5]	[2, 3, 4, 6, 4]	[2, 3, 5, 4, 3]	[2, 3, 6, 2, 2]
[2, 4, 0, 1, 1]	[2, 4, 1, 6, 0]	[2, 4, 2, 4, 6]	[2, 4, 3, 2, 5]	[2, 4, 4, 0, 4]	[2, 4, 5, 5, 3]	[2, 4, 6, 3, 2]
[2, 5, 0, 2, 1]	[2, 5, 1, 0, 0]	[2, 5, 2, 5, 6]	[2, 5, 3, 3, 5]	[2, 5, 4, 1, 4]	[2, 5, 5, 6, 3]	[2, 5, 6, 4, 2]
[2, 6, 0, 3, 1]	[2, 6, 1, 1, 0]	[2, 6, 2, 6, 6]	[2, 6, 3, 4, 5]	[2, 6, 4, 2, 4]	[2, 6, 5, 0, 3]	[2, 6, 6, 5, 2]
[3, 0, 0, 6, 5]	[3, 0, 1, 4, 4]	[3, 0, 2, 2, 3]	[3, 0, 3, 0, 2]	[3, 0, 4, 5, 1]	[3, 0, 5, 3, 0]	[3, 0, 6, 1, 6]
[3, 1, 0, 0, 5]	[3, 1, 1, 5, 4]	[3, 1, 2, 3, 3]	[3, 1, 3, 1, 2]	[3, 1, 4, 6, 1]	[3, 1, 5, 4, 0]	[3, 1, 6, 2, 6]
[3, 2, 0, 1, 5]	[3, 2, 1, 6, 4]	[3, 2, 2, 4, 3]	[3, 2, 3, 2, 2]	[3, 2, 4, 0, 1]	[3, 2, 5, 5, 0]	[3, 2, 6, 3, 6]
[3, 3, 0, 2, 5]	[3, 3, 1, 0, 4]	[3, 3, 2, 5, 3]	[3, 3, 3, 3, 2]	[3, 3, 4, 1, 1]	[3, 3, 5, 6, 0]	[3, 3, 6, 4, 6]

[3, 4, 0, 3, 5] [3, 4, 1, 1, 4] [3, 4, 2, 6, 3] [3, 4, 3, 4, 2] [3, 4, 4, 2, 1] [3, 4, 5, 0, 0] [3, 4, 6, 5, 6]
[3, 5, 0, 4, 5] [3, 5, 1, 2, 4] [3, 5, 2, 0, 3] [3, 5, 3, 5, 2] [3, 5, 4, 3, 1] [3, 5, 5, 1, 0] [3, 5, 6, 6, 6]
[3, 6, 0, 5, 5] [3, 6, 1, 3, 4] [3, 6, 2, 1, 3] [3, 6, 3, 6, 2] [3, 6, 4, 4, 1] [3, 6, 5, 2, 0] [3, 6, 6, 0, 6]
[4, 0, 0, 1, 2] [4, 0, 1, 6, 1] [4, 0, 2, 4, 0] [4, 0, 3, 2, 6] [4, 0, 4, 0, 5] [4, 0, 5, 5, 4] [4, 0, 6, 3, 3]
[4, 1, 0, 2, 2] [4, 1, 1, 0, 1] [4, 1, 2, 5, 0] [4, 1, 3, 3, 6] [4, 1, 4, 1, 5] [4, 1, 5, 6, 4] [4, 1, 6, 4, 3]
[4, 2, 0, 3, 2] [4, 2, 1, 1, 1] [4, 2, 2, 6, 0] [4, 2, 3, 4, 6] [4, 2, 4, 2, 5] [4, 2, 5, 0, 4] [4, 2, 6, 5, 3]
[4, 3, 0, 4, 2] [4, 3, 1, 2, 1] [4, 3, 2, 0, 0] [4, 3, 3, 5, 6] [4, 3, 4, 3, 5] [4, 3, 5, 1, 4] [4, 3, 6, 6, 3]
[4, 4, 0, 5, 2] [4, 4, 1, 3, 1] [4, 4, 2, 1, 0] [4, 4, 3, 6, 6] [4, 4, 4, 4, 5] [4, 4, 5, 2, 4] [4, 4, 6, 0, 3]
[4, 5, 0, 6, 2] [4, 5, 1, 4, 1] [4, 5, 2, 2, 0] [4, 5, 3, 0, 6] [4, 5, 4, 5, 5] [4, 5, 5, 3, 4] [4, 5, 6, 1, 3]
[4, 6, 0, 0, 2] [4, 6, 1, 5, 1] [4, 6, 2, 3, 0] [4, 6, 3, 1, 6] [4, 6, 4, 6, 5] [4, 6, 5, 4, 4] [4, 6, 6, 2, 3]
[5, 0, 0, 3, 6] [5, 0, 1, 1, 5] [5, 0, 2, 6, 4] [5, 0, 3, 4, 3] [5, 0, 4, 2, 2] [5, 0, 5, 0, 1] [5, 0, 6, 5, 0]
[5, 1, 0, 4, 6] [5, 1, 1, 2, 5] [5, 1, 2, 0, 4] [5, 1, 3, 5, 3] [5, 1, 4, 3, 2] [5, 1, 5, 1, 1] [5, 1, 6, 6, 0]
[5, 2, 0, 5, 6] [5, 2, 1, 3, 5] [5, 2, 2, 1, 4] [5, 2, 3, 6, 3] [5, 2, 4, 4, 2] [5, 2, 5, 2, 1] [5, 2, 6, 0, 0]
[5, 3, 0, 6, 6] [5, 3, 1, 4, 5] [5, 3, 2, 2, 4] [5, 3, 3, 0, 3] [5, 3, 4, 5, 2] [5, 3, 5, 3, 1] [5, 3, 6, 1, 0]
[5, 4, 0, 0, 6] [5, 4, 1, 5, 5] [5, 4, 2, 3, 4] [5, 4, 3, 1, 3] [5, 4, 4, 6, 2] [5, 4, 5, 4, 1] [5, 4, 6, 2, 0]
[5, 5, 0, 1, 6] [5, 5, 1, 6, 5] [5, 5, 2, 4, 4] [5, 5, 3, 2, 3] [5, 5, 4, 0, 2] [5, 5, 5, 5, 1] [5, 5, 6, 3, 0]
[5, 6, 0, 2, 6] [5, 6, 1, 0, 5] [5, 6, 2, 5, 4] [5, 6, 3, 3, 3] [5, 6, 4, 1, 2] [5, 6, 5, 6, 1] [5, 6, 6, 4, 0]
[6, 0, 0, 5, 3] [6, 0, 1, 3, 2] [6, 0, 2, 1, 1] [6, 0, 3, 6, 0] [6, 0, 4, 4, 6] [6, 0, 5, 2, 5] [6, 0, 6, 0, 4]
[6, 1, 0, 6, 3] [6, 1, 1, 4, 2] [6, 1, 2, 2, 1] [6, 1, 3, 0, 0] [6, 1, 4, 5, 6] [6, 1, 5, 3, 5] [6, 1, 6, 1, 4]
[6, 2, 0, 0, 3] [6, 2, 1, 5, 2] [6, 2, 2, 3, 1] [6, 2, 3, 1, 0] [6, 2, 4, 6, 6] [6, 2, 5, 4, 5] [6, 2, 6, 2, 4]
[6, 3, 0, 1, 3] [6, 3, 1, 6, 2] [6, 3, 2, 4, 1] [6, 3, 3, 2, 0] [6, 3, 4, 0, 6] [6, 3, 5, 5, 5] [6, 3, 6, 4, 4]
[6, 4, 0, 2, 3] [6, 4, 1, 0, 2] [6, 4, 2, 5, 1] [6, 4, 3, 3, 0] [6, 4, 4, 1, 6] [6, 4, 5, 6, 5] [6, 4, 6, 4, 4]
[6, 5, 0, 3, 3] [6, 5, 1, 1, 2] [6, 5, 2, 6, 1] [6, 5, 3, 4, 0] [6, 5, 4, 2, 6] [6, 5, 5, 0, 5] [6, 5, 6, 5, 4]
[6, 6, 0, 4, 3] [6, 6, 1, 2, 2] [6, 6, 2, 0, 1] [6, 6, 3, 5, 0] [6, 6, 4, 3, 6] [6, 6, 5, 1, 5] [6, 6, 6, 6, 4]

Zadanie 7

Zgodnie z algorytmem, najpierw znajdujemy najbliższe kodowanemu wektorowi słowo z wcześniej otrzymanej listy wszystkich możliwych. Nastepnie znajdujemy współrzedne najbliższego słowa w bazie B. Wektor, który dekodujemy to [6,6,0,4,3].

```
G = np.array([[1, 0, 0, 2, 4],
              [0, 1, 0, 1, 0],
              [0, 0, 1, 5, 6]])

v = [6, 6, 0, 4, 3]

hamming_distances = []

for x in all_coded_words:
  count = 0
  for i in range(5):
    if x[i] != v[i]:
      count+=1
  hamming_distances.append(count)

#w -- najbli̦zsze s␣lowo
w_indexes = [hamming_distances.index(x) for x in hamming_distances if x == min(hamming_distances)]
w = all_coded_words[rand.choice(w_indexes)]

for a in range(7):
    for b in range(7):
        for c in range(7):
            arr = a*e1 + b*e2 + c*e3
            arr = arr%7
            if arr.tolist() == w:
              r = [a, b, c]
              break
#r -- wspó␣lrzedne w bazie B, czyli odkodowany wektor
print(r)
```

Odkodowany wektor wyglada nastepujaco: [6,6,0].

Zadanie 8

a)
Kod generujacy macierz A o 10 kolumnach i 4 wierszach.

```
import random
random.seed(2137)
A =[[0]*10 for _ in range(4)]
    for i in range(4):
        for j in range(10):
            A[i][j]=random.randint(0,4)
```

$$A = \begin{bmatrix} 4 & 2 & 2 & 4 & 2 & 0 & 1 & 4 & 3 & 2 \\ 3 & 1 & 0 & 1 & 1 & 1 & 4 & 2 & 2 & 1 \\ 0 & 4 & 4 & 3 & 0 & 2 & 1 & 0 & 0 & 3 \\ 0 & 0 & 4 & 2 & 1 & 4 & 2 & 4 & 3 & 0 \end{bmatrix}$$

b)
Unormowanie jest robione w jezyku Python, natomiast generowanie obrazu w Mathematica

```
for i in range(10):
    for j in range(4):
        A[i][j]=A[i][j]/4
```

$$A = \begin{bmatrix} 1 & 0.5 & 0.5 & 1 & 0.5 & 0 & 0.25 & 1 & 0.75 & 0.5 \\ 0.75 & 0.25 & 0 & 0.25 & 0.25 & 0.25 & 1 & 0.5 & 0.5 & 0.25 \\ 0 & 1 & 1 & 0.75 & 0 & 0.5 & 0.25 & 0 & 0 & 0.75 \\ 0 & 0 & 1 & 0.5 & 0.25 & 1 & 0.5 & 1 & 0.75 & 0 \end{bmatrix}$$

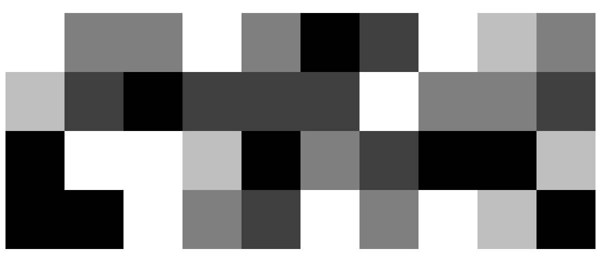

Figure 1: Przedstawienie macierzy z punktu a)

c)
Aby pokazać, że istnieje (11,4)-kod liniowy nad ciałem Z_5, taki, że macierz G jest macierza generujaca kodu C, należy pokazać, że wektory z tej macierzy e_1, e_2, e_3, e_4 tworza baze kodu C, czyli sa liniowo niezależne. Tym samym należy pokazać, że wiersze macierzy G sa liniowo niezależne.

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 4 & 4 & 2 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 3 & 0 & 2 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 4 & 3 & 0 \end{bmatrix}$$

Widać, że w tej macierzy pierwsze 4 kolumny tworza macierz jednostkowa, wiec, wektory e_1, e_2, e_3, e_4 sa liniowo niezależne, co trzeba było pokazać.
d)
Od razy można zakodować cała macierz, bedzie to potrzebne w kolejnym punkcie.

```
G=[[1,0,0,0,0,4,4,2,0,1,1],[0,1,0,0,0,3,0,2,2,1,0,],[0,0,1,0,0,2,0,1,1,1,1],[0,0,0,1,1,0,0,0,4,3,0]]
A_transponowane = np.transpose(A)

wynik_transpozycja = np.dot(A_transponowane,G)

for i in range(wynik_transpozycja.shape[0]):
    for j in range(wynik_transpozycja.shape[1]):
        wynik_transpozycja[i][j]=wynik_transpozycja[i][j]%5

wynik=np.transpose(wynik_transpozycja)
```

$$wynik = \begin{bmatrix} 4 & 2 & 2 & 4 & 2 & 0 & 1 & 4 & 3 & 2 \\ 3 & 1 & 0 & 1 & 1 & 1 & 4 & 2 & 2 & 1 \\ 0 & 4 & 4 & 3 & 0 & 2 & 1 & 0 & 0 & 3 \\ 0 & 0 & 4 & 2 & 1 & 4 & 2 & 4 & 3 & 0 \\ 0 & 0 & 4 & 2 & 1 & 4 & 2 & 4 & 3 & 0 \\ 0 & 4 & 1 & 0 & 1 & 2 & 3 & 2 & 3 & 2 \\ 1 & 3 & 3 & 1 & 3 & 0 & 4 & 1 & 2 & 3 \\ 4 & 0 & 3 & 3 & 1 & 4 & 1 & 2 & 0 & 4 \\ 1 & 1 & 0 & 3 & 1 & 0 & 2 & 0 & 1 & 0 \\ 2 & 2 & 3 & 4 & 1 & 0 & 2 & 3 & 4 & 1 \\ 4 & 1 & 1 & 2 & 2 & 2 & 2 & 4 & 3 & 0 \end{bmatrix}$$

Kolumny tej macierzy to wszystkie zakodowane wektory z macierzy A.
e)
Bierzemy tu wynik transponowany, aby ładnie były wektory wierszowo, co potem ułatwi odkodowywanie.

```
for i in range(wynik_transpozycja.shape[0]):
    for j in range(wynik_transpozycja.shape[1]):
        rand=random.random()
        if rand>=0.95:
            wynik_transpozycja[i][j] += 3
            wynik_transpozycja[i][j]=wynik_transpozycja[i][j]%5
```

$$wynik = \begin{bmatrix} 4 & 2 & 2 & 4 & 2 & 0 & 1 & 2 & 3 & 2 \\ 3 & 1 & 3 & 1 & 1 & 1 & 4 & 2 & 2 & 1 \\ 0 & 4 & 2 & 3 & 0 & 2 & 1 & 0 & 0 & 3 \\ 0 & 0 & 4 & 2 & 4 & 4 & 2 & 4 & 3 & 0 \\ 0 & 0 & 4 & 2 & 1 & 4 & 2 & 4 & 3 & 0 \\ 3 & 4 & 1 & 0 & 1 & 2 & 1 & 2 & 3 & 2 \\ 1 & 3 & 3 & 1 & 3 & 0 & 4 & 1 & 2 & 3 \\ 4 & 0 & 3 & 3 & 1 & 4 & 1 & 2 & 0 & 4 \\ 1 & 1 & 0 & 3 & 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 2 & 3 & 4 & 4 & 0 & 2 & 3 & 4 & 1 \\ 4 & 1 & 1 & 2 & 0 & 2 & 2 & 4 & 3 & 0 \end{bmatrix}$$

f)

```
#znajdujemy wszystkie możliwe zakodowane s␣lowa kodu
e1 = np.array(G[0])
e2 = np.array(G[1])
e3 = np.array(G[2])
e4 = np.array(G[3])

all_coded_words = []

for a in range(5):
  for b in range(5):
    for c in range(5):
      for d in range(5):
        arr = a*e1 + b*e2 + c*e3 + d*e4
        arr = arr%5
        all_coded_words.append(arr.tolist())

#definiujemy funkcje pomocnicza do odkodowywania
def decode(v, e1, e2, e3, e4, all_coded_words):

  hamming_distances = []

  for x in all_coded_words:
    count = 0
    for i in range(11):
      if x[i] != v[i]:
        count+=1
    hamming_distances.append(count)

  w_indexes = [hamming_distances.index(x) for x in hamming_distances if x == min(hamming_distances)]
  w = all_coded_words[random.choice(w_indexes)]

  r = []

  for a in range(5):
    for b in range(5):
        for c in range(5):
          for d in range(5):
            arr = a*e1 + b*e2 + c*e3 +d*e4
            arr = arr%5
            if arr.tolist() == w:
              r = [a, b, c, d]
              break
  return r

result = []
for v in wynik_transpozycja.tolist():
  result.append(decode(v, e1, e2, e3, e4, all_coded_words))
```

g)
Algorytm z punktu f odkodował od razu cała macierz. Po transpozycji wyglada nastepujaco:

$$g = \begin{bmatrix} 1 & 3 & 3 & 1 & 0 & 0 & 4 & 1 & 4 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 3 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 4 & 0 \end{bmatrix}$$

h)
Porównujac macierze z punktu a i z punktu g można zauważyć, że wszystkie kolumny sie różnia, tym samym żaden wektor nie został prawidłowo odkodowany i)

$$g = \begin{bmatrix} 0.25 & 0.75 & 0.75 & 0.25 & 0 & 0 & 1 & 0.25 & 1 & 0.75 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.75 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0 & 0 & 0.25 & 0 & 0 & 0.75 & 0 \\ 0 & 0 & 0.25 & 0 & 0 & 0.25 & 0.25 & 0.25 & 1 & 0 \end{bmatrix}$$

Figure 2: Odkodowana macierz