Projekt nr 2, Algebra w Analizie Danych   Jakub Półtorak, Igor Rudolf

Zad.1

Zadanie 1.

Chcemy udowodnić, że odległość Hamminga jest metryką.

Wykażmy najpierw, że gdy: $d(u,v)=0 \Leftrightarrow u=v$.

Dowód:

$(\Rightarrow)$. Skoro $d(u,v)=0$, to moc zbioru $|\{i\in[n]: u_i \neq v_i\}|$ jest równa 0. Innymi słowy $\forall_{i\in[n]}\ u_i = v_i$, czyli $u=v$. ■

$(\Leftarrow)$ Niech $u=v$. To mamy, że $\forall_{i\in[n]}\ u_i = v_i$, czyli $\{i\in[n]: u_i \neq v_i\} = \emptyset$. Zatem $d(u,v)=0$. ■

Wykażmy, że ~~skoro~~ $d(u,v) = d(v,u)$

~~$D(x,y) = d($~~ $d(u,v) = |\{i\in[n]: u_i \neq v_i\}|$

$d(v,u) = |\{i\in[n]: v_i \neq u_i\}|$.

Zatem $d(u,v) = d(v,u)$

oczywiste.

Wykażmy, że $d(x,z) \leq d(x,y) + d(y,z)$.

Ustalmy dowolne $n\in\mathbb{N}$. Rozważmy dowolne $x,y,z\in\mathbb{K}^n$.

Dla ustalonego ciała $\mathbb{K}$.

Zapiszmy $x=(x_1, x_2, \ldots x_n)^T$

$y=(y_1, y_2, \ldots y_n)^T$

$z=(z_1, z_2, \ldots z_n)^T$

Zauważmy, że $\forall i\in[n]$ mogą zajść następujące sytuacje:

1. $x_i = y_i \wedge y_i \neq z_i \Rightarrow x_i \neq z_i$
2. $x_i = y_i \wedge y_i = z_i \Rightarrow x_i \neq z_i$
3. $x_i \neq y_i \wedge y_i = z_i \Rightarrow x_i \neq z_i$
4. $x_i \neq y_i \wedge y_i \neq z_i \Rightarrow (x_i = z_1 \vee x_i \neq z_i)$

Rozważmy trzy zbiory $XZ = \{i\in[n]: x_i \neq z_i\}$ $YZ = \{i\in[n]: y_i \neq z_i\}$

$XY = \{i\in[n]: x_i \neq y_i\}$

~~Weźmy teraz dowolne $i \in XZ$~~

Rozważmy sytuację gdy $XZ = \emptyset$, nie ma co dowodzić, bo

$d(x,z) = 0$, zatem $0 \leq d(x,y) + d(y,z)$, oraz

~~str.~~ $\forall u,v \in \mathbb{Z}^n,\ d(u,v) \geq 0$ ∎

Rozważmy zatem gdy $XZ \neq \emptyset$. Weźmy dowolne

$i \in XZ$ gdzie $i \in [n]$. Z (1), (2), (3), (4) wynika, że

wtedy $i \in XY \vee i \in YZ$. Zatem ~~musi być, że~~ zachodzi, że:

$|XZ| \leq |XY| + |YZ|$, czyli mamy, że:

$d(x,z) \leq d(x,y) + d(y,z)$ ∎

Zad.2

Zadanie 2

Aby móc pokazać, że wynikiem kodowania dowolnego wektora $v \in K^k$ jest słowo kodowe kodu C, wykażemy, że zakodowany wektor v (będziemy od teraz oznaczać go jako w) jest kombinacją liniową wierszy z macierzy G. Oczywiście rozważamy kod (n,k). Oznaczmy jako β bazę kodu C, mamy $B = (e_1, e_2, \ldots e_k)$.

Zatem $G = \begin{pmatrix} e_1^T \\ e_2^T \\ \vdots \\ e_k^T \end{pmatrix}$. Dodatkowo $e_i \in K^n$ dla $i \in [n]$.

Chcemy mieć, że: $w = \alpha_1 e_1^T + \alpha_2 e_2^T + \ldots + \alpha_k e_k^T$.

Wiemy jednak, że: $w = (v^T G)^T$, oraz niech $v = (\beta_1, \beta_2, \ldots \beta_k)^T$.

$w = G^T \cdot v = (e_1\ e_2 \ldots e_k) \cdot \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{pmatrix}$.

Ale wiemy też, że dowolny wektor $e_i \in K^n$ gdzie $i \in [n]$

Niech zatem $e_i = (e_{i1}\ e_{i2} \ldots e_{in})^T$ dla pewnych $e_{i1}, e_{i2}, \ldots e_{in} \in K$.

Możemy zatem zapisać: $w = \begin{pmatrix} e_{11} & e_{21} & & e_{k1} \\ e_{12} & e_{22} & \ldots & e_{k2} \\ \vdots & \vdots & & \vdots \\ e_{1n} & e_{2n} & & e_{kn} \end{pmatrix} \cdot \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{pmatrix} =$

$$= \begin{pmatrix} e_{11}\beta_1 + e_{21}\beta_2 + \ldots + e_{k1}\beta_k \\ e_{12}\beta_1 + e_{22}\beta_2 + \ldots + e_{k2}\beta_k \\ \vdots \\ e_{1n}\beta_1 + e_{2n}\beta_2 + \ldots + e_{kn}\beta_k \end{pmatrix} = \beta_1 e_1 + \beta_2 e_2 + \ldots + \beta_k e_k.$$

Wystarczy teraz, że $\alpha_1 = \beta_1$, $\alpha_2 = \beta_2$, ... $\alpha_k = \beta_k$.

Zatem ~~nasze to~~ wektor powstały z kodowania jest kombinacją liniową wierszy z G. $\square$

Zad.3

Zadanie 3

Załóżmy, że mamy bazę $B = (e_1, e_2, e_3, \dots e_k)$. gdzie

wtedy bez $\forall_{i \in [k]}\ e_i \in \mathbb{K}^n$. Definiujemy

$G = \begin{pmatrix} e_1^T \\ e_2^T \\ \vdots \\ e_k^T \end{pmatrix}$. Niech C będzie kodem (n, k) nad skończonym ciałem K powstałym z macierzy

generującej G. Weźmy zatem dowolny wektor $w \in C$, wtedy

możemy zapisać: $w = \alpha_1 e_1 + \alpha_2 e_2 + \dots + \alpha_k e_k$. Zauważmy, że

w wyniku dekodowania ~~dostaniemy wekt.~~ przy użyciu algorytmu

MinimizeHammingDistance dostaniemy wektor $v = (\alpha_1\ \alpha_2 \dots \alpha_k)^T$.

Wykażmy teraz, że kodując wektor v dostaniemy wektor w.

mamy, że jeśli $w' = (v^T \cdot G)^T = G^T \cdot v^T$, my zał., że

to $G^T = (e_1\ e_2 \dots e_k)$ $\quad v^T = (\alpha_1\ \alpha_2 \dots, \alpha_k)$.

Rozważmy zatem:

$$w' = (e_1\ e_2 \dots e_k) \cdot (\alpha_1\ \alpha_2 \dots \alpha_k)$$

Ale skoro $\forall_{i \in [k]}\ e_i \in \mathbb{K}^n$, to możemy zapisać, że

$e_i = (e_{i1}\ e_{i2} \dots e_{in})^T$, czyli mamy, że

$$w' = \begin{pmatrix} e_{11} & e_{21} & \dots & e_{k1} \\ e_{12} & e_{22} & \dots & e_{k2} \\ \vdots & \vdots & & \vdots \\ e_{1n} & e_{2n} & & e_{kn} \end{pmatrix} \cdot (\alpha_1\ \alpha_2 \dots \alpha_k) =$$

$$= \begin{pmatrix} \alpha_1 e_{11} + \alpha_2 e_{21} + \dots + \alpha_k e_{k1} \\ \vdots \\ \alpha_1 e_{1n} + \alpha_2 e_{2n} + \dots + \alpha_k e_{kn} \end{pmatrix} = \alpha_1 \begin{pmatrix} e_{11} \\ \vdots \\ e_{1n} \end{pmatrix} + \dots + \alpha_k \begin{pmatrix} e_{k1} \\ \vdots \\ e_{kn} \end{pmatrix} =$$

$$= \alpha_1 e_1 + \dots + \alpha_k e_k = w$$

Zad.4

Zadanie 4

Chcemy udowodnić, że dla dowolnej przestrzeni liniowej V nad ciałem $\mathbb{K}$ odległość Hamminga jest niezmiennicza ze względu na przesunięcia.

Dowód:

Ustalmy dowolne $u, x, v \in \mathbb{K}^n$. Chcemy wykazać, że $d(u,v) = d(u+x, v+x)$.

$u = (u_1, u_2, \dots u_n)^T$

$v = (v_1, v_2, \dots, v_n)^T$

$x = (x_1, x_2, \dots, x_n)^T$

$u + x = (u_1 + x_1, u_2 + x_2, \dots u_n + x_n)$

$v + x = (v_1 + x_1, v_2 + x_2, \dots v_n + x_n)$.

Rozważmy $d(u+x, v+x) = |\{i \in [n] : u_i + x_i = v_i + x_i\}| =$

$= |\{i \in [n] : u_i = v_i| = d(u, v)$

Zad.5

```python
def hammingDistance(vectorA, vectorB):
    if (type(vectorA) or type(vectorB)) != list:
        raise TypeError("Podane wektory musza byc typu List")
    elif len(vectorA) != len(vectorB):
        raise ValueError("Podane wektory musza byc tej samej podprzestrzeni")
    counter = 0
    for k in range(len(vectorA)):
        if vectorA[k] != vectorB[k]:
            counter += 1
    return counter


def distanceComparator(consideringList):
    if type(consideringList) != list:
        raise TypeError("Przytrzymywane elementy musza znajdowac sie w liscie")
    minDistance = float("inf")
    lookingVectors = []
    for k in range(len(consideringList)):
        for l in range(len(consideringList)):
            if consideringList[k] != consideringList[l]:
                comparingDistance = hammingDistance(consideringList[k], consideringList[l])
                if comparingDistance <= minDistance:
                    minDistance = comparingDistance
                    if ([k, l] not in lookingVectors) and ([l, k] not in lookingVectors):
                        lookingVectors.append([k, l])
    return lookingVectors
```

```python
def main():
    vectorFirst = [1, 2, 0, 1]
    vectorSecond = [0, 0, 0, 1]
    consideringDist = hammingDistance(vectorFirst, vectorSecond)
    print("Odleglosc Hamminga dla rozwazanych dwoch wektorow wynosi: " + str(consideringDist))

    vectorA = [1, 2, 1, 2, 0]
    vectorB = [1, 1, 1, 1, 1]
    vectorC = [0, 0, 2, 1, 1]
    vectorD = [2, 2, 2, 1, 0]
    helpingList = [vectorA, vectorB, vectorC, vectorD]
    listOfIndexes = distanceComparator(helpingList)
    print("Wektory znajdujace sie najblizej siebie to wektory(są one zapisane w formie transponowanej):")
    for k in range(len(listOfIndexes)):
        pair = listOfIndexes[k]
        first= pair[0]
        second= pair[1]
        firstVector= helpingList[first]
        secondVector= helpingList[second]
        print(str(firstVector)+" oraz "+str(secondVector))


if __name__ == '__main__':
    main()
```

```
Odleglosc Hamminga dla rozwazanych dwoch wektorow wynosi: 2
Wektory znajdujace sie najblizej siebie to wektory(są one zapisane w formie transponowanej):
[1, 2, 1, 2, 0] oraz [1, 1, 1, 1, 1]
[1, 2, 1, 2, 0] oraz [2, 2, 2, 1, 0]
[1, 1, 1, 1, 1] oraz [0, 0, 2, 1, 1]
[0, 0, 2, 1, 1] oraz [2, 2, 2, 1, 0]
```

W powyższym programie metoda hammingDistance, wyznacza odległość Hamminga dla danych dwóch wektorów. Metoda distanceComparator wyznacza na podstawie podanej listy w parametrze metody, indeksy dwóch wektorów (indeksowanie jest od zera) rozważanej listy mających najmniejszą odległość w sensie Hamminga.

Zad.6

Zapisujemy wektory z bazy B, pod zmiennymi Vector1, Vector2, Vector3. Tworzymy przy okazji Macierz AllCodedWords, która będzie zawierała wszystkie możliwe słowa kodowe kodu C.

```
In[1]:= Vector1 = {1, 0, 0, 2, 4}

Out[1]= {1, 0, 0, 2, 4}

In[2]:= Vector2 = {0, 1, 0, 1, 0}

Out[2]= {0, 1, 0, 1, 0}

In[3]:= Vector3 = {0, 0, 1, 5, 6}

Out[3]= {0, 0, 1, 5, 6}

In[4]:= AllCodedWords = {}

Out[4]= {}
```

Tworzymy wszystkie możliwe kombinacje liniowe powyższych wektorów w ciele Z7, poprzez potrójną iteracje I operacje modulo:

```
In[10]:= For[i = 0, i < 7, i++, For[j = 0, j < 7, j++, For[k = 0, k < 7, k++, AppendTo[AllCodedWords, Mod[i * Vector1 + j * Vector2 + k * Vector3,
            7]]]]]
```

Tak prezentuje się pełna lista wszystkich wektorów, będących słowami kodowymi kodu liniowego C:

{{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, 3, 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}}

Zad.7

Tworzymy macierz G, która jest macierzą generującą dla rozważanego kodu liniowego. Następnie generujemy pseudolosowo wektor ConsideredVector, który będzie odpowiadał za dowolnie przez nas wybrany wektor.

```
In[13]:= G = Transpose[{Vector1, Vector2, Vector3}]

Out[13]= {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}, {2, 1, 5}, {4, 0, 6}}

In[20]:= ConsideredVector = {}

Out[20]= {}

In[21]:= For[i = 1, i ≤ 5, i++, AppendTo[ConsideredVector, RandomInteger[{0, 6}]]]

In[22]:= ConsideredVector

Out[22]= {5, 0, 1, 5, 1}
```

Tworzymy macierz MatrixOfHammingDistances, przechowującą wartości odległości Hamminga dla poszczególnych słów kodowych i rozważanego wcześniej wektora:

```
In[25]:= MatrixOfHammingDistances = {}

Out[25]= {}

In[26]:= For[i = 1, i ≤ Length[AllCodedWords], i++, AppendTo[MatrixOfHammingDistances, HammingDistance[ConsideredVector,
           AllCodedWords[[i]]]]]

In[27]:= MatrixOfHammingDistances

Out[27]= {4, 2, 4, 4, 4, 4, 3, 5, 4, 5, 5, 5, 4, 4, 5, 4, 4, 5, 5, 5, 4, 5, 4, 5, 5, 5, 5, 3, 5, 4, 5, 4, 5, 5, 4, 4, 4, 5, 5, 5, 5, 4, 5, 4, 5, 5, 4,
          5, 4, 4, 3, 3, 3, 4, 4, 4, 5, 4, 5, 4, 5, 5, 4, 5, 4, 5, 3, 5, 5, 5, 4, 4, 5, 4, 5, 5, 5, 5, 4, 5, 4, 4, 5, 5, 5, 3, 5, 4, 5, 5, 5, 5, 4, 5,
          4, 5, 4, 5, 3, 3, 4, 3, 4, 4, 4, 3, 4, 5, 5, 5, 5, 5, 4, 4, 5, 5, 4, 5, 5, 4, 3, 5, 5, 5, 5, 5, 4, 4, 5, 5, 5, 4, 5, 4, 4, 4, 5, 5, 5, 5, 4,
          4, 5, 5, 5, 5, 4, 4, 3, 4, 4, 2, 4, 4, 5, 3, 5, 5, 4, 5, 5, 5, 4, 5, 5, 4, 4, 5, 5, 4, 4, 5, 4, 5, 5, 5, 4, 5, 5, 4, 5, 4, 5, 4, 5, 4, 4, 5,
          5, 4, 4, 5, 5, 4, 5, 5, 4, 2, 4, 4, 4, 3, 4, 5, 3, 4, 5, 5, 5, 5, 5, 3, 5, 5, 5, 5, 4, 5, 3, 5, 4, 5, 5, 5, 4, 3, 5, 5, 5, 5, 5, 5, 3, 5, 5,
          4, 5, 5, 5, 2, 5, 5, 5, 5, 5, 3, 2, 3, 3, 3, 2, 2, 4, 3, 4, 3, 4, 3, 4, 3, 3, 4, 4, 4, 3, 4, 4, 3, 4, 4, 3, 3, 4, 4, 2, 4, 4, 4, 3, 4, 4, 3,
          4, 4, 4, 2, 4, 4, 3, 3, 4, 4, 3, 4, 3, 3, 3, 4, 4, 4, 4, 5, 4, 4, 5, 4, 5, 5, 5, 3, 4, 5, 5, 5, 5, 5, 4, 4, 5, 5, 4, 5, 5, 4, 3, 5, 5, 5, 5,
          5, 4, 4, 5, 5, 5, 4, 5, 4, 4, 4, 5, 5, 5}
```

Następnie sprawdzamy, jaka jest najmniejsza odległość Hamminga w rozważanej macierzy. Tworzymy macierz VectorsWithMinimalHammingDistance, która przechowuje wszystkie wektory, o minimalnej wartości Hamminga:

```
In[31]:= MinimalHammingDistance = Min[MatrixOfHammingDistances]

Out[31]= 2

In[30]:= VectorsWithMinimalHammingDistance = {}

Out[30]= {}

In[32]:= For[i = 1, i ≤ Length[MatrixOfHammingDistances], i++, If[MatrixOfHammingDistances[[i]] == MinimalHammingDistance,
          AppendTo[VectorsWithMinimalHammingDistance, AllCodedWords[[i]]]]]

In[34]:= VectorsWithMinimalHammingDistance

Out[34]= {{0, 0, 1, 5, 6}, {3, 0, 4, 5, 1}, {4, 0, 1, 6, 1}, {4, 6, 1, 5, 1}, {5, 0, 1, 1, 5}, {5, 0, 5, 0, 1}, {5, 0, 6, 5, 0}, {5, 4, 1, 5, 5}, {5,
          5, 5, 5, 1}}
```

Następnie pseudolosowo wybieramy jeden z rozważanych wektorów w macierzy. Dokonujemy jego dekodowania, poprzez znalezienie jego współrzędnych w bazie B, dzięki czemu otrzymujemy szukany dekodowany wektor:

```
In[40]:= FinalVector = VectorsWithMinimalHammingDistance[[RandomInteger[{1, Length[VectorsWithMinimalHammingDistance]}]]]

Out[40]= {5, 0, 5, 0, 1}

In[42]:= DecodedVector = Transpose[LinearSolve[G, FinalVector, Modulus → 7]]

Out[42]= {5, 0, 5}

In[43]:= MatrixForm[DecodedVector]

Out[43]//MatrixForm=
```

$$\begin{pmatrix} 5 \\ 0 \\ 5 \end{pmatrix}$$

Zad.8

a)

Pseudolosowo generujemy macierz o 10 kolumnach i 4 wierszach:

```
In[6]:= A = ResourceFunction["RandomMatrix"][Integer, {0, 4}, {4, 10}]

Out[6]= {{1, 4, 1, 4, 4, 2, 1, 3, 0, 4}, {2, 1, 0, 3, 1, 4, 3, 3, 4, 2}, {1, 4, 1, 3, 1, 4, 1, 0, 3, 4}, {1, 0, 3, 0, 1, 0, 3, 2, 0, 2}}

In[7]:= MatrixForm[A]
```

Out[7]//MatrixForm=

$$\begin{pmatrix} 1 & 4 & 1 & 4 & 4 & 2 & 1 & 3 & 0 & 4 \\ 2 & 1 & 0 & 3 & 1 & 4 & 3 & 3 & 4 & 2 \\ 1 & 4 & 1 & 3 & 1 & 4 & 1 & 0 & 3 & 4 \\ 1 & 0 & 3 & 0 & 1 & 0 & 3 & 2 & 0 & 2 \end{pmatrix}$$

b)

Dokonujemy normalizacji macierzy A, dzieląc ją przez 4 i na podstawie znormalizowanej macierzy generujemy obrazek:

```
In[5]:= B = A / 4
```

Out[5]= $\left\{\left\{\frac{1}{4}, 1, \frac{1}{4}, 1, 1, \frac{1}{2}, \frac{1}{4}, \frac{3}{4}, 0, 1\right\}, \left\{\frac{1}{2}, \frac{1}{4}, 0, \frac{3}{4}, \frac{1}{4}, 1, \frac{3}{4}, \frac{3}{4}, 1, \frac{1}{2}\right\}, \left\{\frac{1}{4}, 1, \frac{1}{4}, \frac{3}{4}, \frac{1}{4}, 1, \frac{1}{4}, 0, \frac{3}{4}, 1\right\}, \left\{\frac{1}{4}, 0, \frac{3}{4}, 0, \frac{1}{4}, 0, \frac{3}{4}, \frac{1}{2}, 0, \frac{1}{2}\right\}\right\}$

```
In[7]:= Image[B, ImageSize → 300]
```

Out[7]=

c)

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

Aby istniał (11,4) kod liniowy C nad ciałem Z5 taki, że G jest macierzą generującą kodu C, macierz G musi mieć 4 wiersze i 11 kolumn, co gołym okiem widać. Wszystkie wiersze macierzy G powinny być także liniowo niezależne. Łatwo zauważyć, że rzeczywiście tak jest. Wystarczy spojrzeć na pierwsze cztery współrzędne rozważanych wierszy, które po sklejeniu tworzą macierz jednostkową. To pokazuje, że wiersze są liniowo niezależne, więc Macierz G jest macierzą generującą rozważanego kodu.

d)

Wprowadzamy podaną macierz G do programu

```
In[18]:= 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}}

Out[18]= {{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}}

In[19]:= MatrixForm[G]

Out[19]//MatrixForm=
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 4 & 4 & 2 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 3 & 0 & 2 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 4 & 3 & 0 \end{pmatrix}$$

Dokonujemy kodowania wszystkich wektorów z macierzy A, poprzez mnożenie transponowanej macierzy A z G, ponownej transpozycji i wykonanie operacji modulo 5:

```
In[3]:= Coded = Mod[Transpose[Transpose[A].G], 5]

Out[3]= {{1, 4, 1, 4, 4, 2, 1, 3, 0, 4}, {2, 1, 0, 3, 1, 4, 3, 3, 4, 2}, {1, 4, 1, 3, 1, 4, 1, 0, 3, 4}, {1, 0, 3, 0, 1, 0, 3, 2, 0, 2}, {1, 0, 3, 0,
      1, 0, 3, 2, 0, 2}, {2, 2, 1, 1, 1, 3, 0, 1, 3, 0}, {4, 1, 4, 1, 1, 3, 4, 2, 0, 1}, {2, 4, 3, 2, 1, 1, 4, 2, 1, 1}, {4, 1, 3, 4, 2, 2, 4,
      4, 1, 1}, {2, 4, 1, 0, 4, 0, 4, 2, 2, 1}, {2, 3, 2, 2, 0, 1, 2, 3, 3, 3}}

In[4]:= MatrixForm[Coded]

Out[4]//MatrixForm=
```

$$\begin{pmatrix} 1 & 4 & 1 & 4 & 4 & 2 & 1 & 3 & 0 & 4 \\ 2 & 1 & 0 & 3 & 1 & 4 & 3 & 3 & 4 & 2 \\ 1 & 4 & 1 & 3 & 1 & 4 & 1 & 0 & 3 & 4 \\ 1 & 0 & 3 & 0 & 1 & 0 & 3 & 2 & 0 & 2 \\ 1 & 0 & 3 & 0 & 1 & 0 & 3 & 2 & 0 & 2 \\ 2 & 2 & 1 & 1 & 1 & 3 & 0 & 1 & 3 & 0 \\ 4 & 1 & 4 & 1 & 1 & 3 & 4 & 2 & 0 & 1 \\ 2 & 4 & 3 & 2 & 1 & 1 & 4 & 2 & 1 & 1 \\ 4 & 1 & 3 & 4 & 2 & 2 & 4 & 4 & 1 & 1 \\ 2 & 4 & 1 & 0 & 4 & 0 & 4 & 2 & 2 & 1 \\ 2 & 3 & 2 & 2 & 0 & 1 & 2 & 3 & 3 & 3 \end{pmatrix}$$

e)

Pseudolosowo dokonujemy losowania 10 wartosci, a następnie określamy czy przy transmisji zostanie dodane 0 lub 3.

```
In[20]:= R = {{}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}}

Out[20]= {{}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}}

In[21]:= For[i = 1, i ≤ 11, i++, For[j = 0, j ≤ 9, j++, AppendTo[R[[i]], RandomInteger[{0, 99}]]]]

In[22]:= R

Out[22]= {{58, 66, 41, 80, 62, 43, 90, 29, 32, 63}, {60, 70, 99, 18, 28, 98, 58, 53, 72, 92}, {17, 83, 67, 49, 67, 32, 28, 40, 7, 87}, {62, 42, 30, 2, 15, 23, 40, 40, 72, 89}, {41, 57, 40, 65, 14, 94,
      91, 85, 0, 59}, {42, 82, 90, 20, 69, 55, 47, 32, 1, 53}, {57, 16, 30, 2, 52, 99, 77, 87, 27, 43}, {40, 28, 82, 81, 16, 46, 33, 50, 21, 43}, {12, 17, 0, 85, 56, 45, 97, 23, 99, 28}, {49,
      15, 4, 96, 54, 46, 64, 34, 92, 32}, {28, 40, 82, 39, 52, 77, 3, 33, 40, 38}}

In[23]:= For[i = 1, i ≤ 11, i++, For[j = 1, j ≤ 10, j++, If[R[[i]][[j]] ≤ 94, R[[i]][[j]] = 0, R[[i]][[j]] = 3]]]

In[24]:= R

Out[24]= {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 3, 0, 0, 3, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0,
      0, 0, 0, 0, 3, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 3, 0, 3, 0}, {0, 0, 0, 3, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}
```

Dodajemy każdemu wektorowi wcześniej przydzieloną wartość:

```
In[25]:= AfterTransmission = Mod[R + Coded, 5]

Out[25]= {{1, 4, 1, 4, 4, 2, 1, 3, 0, 4}, {2, 1, 3, 3, 1, 2, 3, 3, 4, 2}, {1, 4, 1, 3, 1, 4, 1, 0, 3, 4}, {1, 0, 3, 0, 1, 0, 3, 2, 0, 2}, {1, 0, 3, 0, 1, 0, 3, 2, 0, 2}, {2, 2, 1, 1, 1, 3, 0, 1, 3, 0}, {4,
    1, 4, 1, 1, 1, 4, 2, 0, 1}, {2, 4, 3, 2, 1, 1, 4, 2, 1, 1}, {4, 1, 3, 4, 2, 2, 2, 4, 4, 1}, {2, 4, 1, 3, 4, 0, 4, 2, 2, 1}, {2, 3, 2, 2, 0, 1, 2, 3, 3, 3}}

In[26]:= MatrixForm[AfterTransmission]

Out[26]//MatrixForm=
```

$$\begin{pmatrix} 1 & 4 & 1 & 4 & 4 & 2 & 1 & 3 & 0 & 4 \\ 2 & 1 & 3 & 3 & 1 & 2 & 3 & 3 & 4 & 2 \\ 1 & 4 & 1 & 3 & 1 & 4 & 1 & 0 & 3 & 4 \\ 1 & 0 & 3 & 0 & 1 & 0 & 3 & 2 & 0 & 2 \\ 1 & 0 & 3 & 0 & 1 & 0 & 3 & 2 & 0 & 2 \\ 2 & 2 & 1 & 1 & 1 & 3 & 0 & 1 & 3 & 0 \\ 4 & 1 & 4 & 1 & 1 & 1 & 4 & 2 & 0 & 1 \\ 2 & 4 & 3 & 2 & 1 & 1 & 4 & 2 & 1 & 1 \\ 4 & 1 & 3 & 4 & 2 & 2 & 2 & 4 & 4 & 1 \\ 2 & 4 & 1 & 3 & 4 & 0 & 4 & 2 & 2 & 1 \\ 2 & 3 & 2 & 2 & 0 & 1 & 2 & 3 & 3 & 3 \end{pmatrix}$$

f)

Tworzymy macierz GenerateMatrix, która będzie zawierała wszystkie możliwe kombinacje liniowe wektorów w Z5 długości 4:

```
In[51]:= GenerateMatrix = {{}, {}, {}, {}}

Out[51]= {{}, {}, {}, {}}
```

```
For[i = 0, i ≤ 4, i++, For[j = 0, j ≤ 4, j++, For[k = 0, k ≤ 4, k++, For[l = 0, l ≤ 4, l++, AppendTo[GenerateMatrix[[1]], i];
    AppendTo[GenerateMatrix[[2]], j]; AppendTo[GenerateMatrix[[3]], k]; AppendTo[GenerateMatrix[[4]], l]]]]]

GenerateMatrix
```

Tak się prezentuje cała macierz:

```
{{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4},
 {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2,
  2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0,
  0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3,
  3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1,
  1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4,
  4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
  2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4,
  4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2,
  2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0,
  0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3,
  3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0,
  1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3,
  3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1,
  1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4,
  4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2,
  2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4},
 {0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2,
  3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2,
  3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2,
  3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1,
  2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0,
  1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4,
  0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3,
  4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2,
  3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1,
  2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0,
  1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4,
  0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3,
  4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2,
  3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1,
  2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0,
  1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4}}
```

Wypełniamy macierz AllCodedVectors wszystkimi możliwymi słowami kodowymi wcześniej rozważanego kodu(Pokazana tylko część wektorów):

```
AllCodedVectors = Transpose[Mod[Transpose[GenerateMatrix].G, 5]]
```

```
{{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3,
```

Tworzymy macierz MatrixOfHammingDistances, która będzie przechowywała wszystkie odległości Hamminga dla rozważanych wektorów:

```
In[71]:= MatrixOfHammingDistances = {{}, {}, {}, {}, {}, {}, {}, {}, {}, {}}
Out[71]= {{}, {}, {}, {}, {}, {}, {}, {}, {}, {}}
```

```
In[19]:= For[i = 1, i ≤ 10, i++, For[j = 1, j ≤ 625, j++, AppendTo[MatrixOfHammingDistances[[i]],
           HammingDistance[Transpose[AfterTransmission][[i]], Transpose[AllCodedVectors][[j]]]]]]
```

Tworzymy macierz MatrixOfMinimalHammingDistanceForEachVector, która przechowuje minimalną odległość Hamminga dla rozważanych wektorów:

```
In[26]:= MatrixOfMinimalHammingDistancesForEachVector = {{11}, {11}, {11}, {11}, {11}, {11}, {11}, {11}, {11}, {11}}
Out[26]= {{11}, {11}, {11}, {11}, {11}, {11}, {11}, {11}, {11}, {11}}
```

Iterujemy i aktualizujemy minimalną odległość Hamminga dla rozważanych wektorów:

```
In[38]:= For[i = 1, i ≤ 10, i++, For[j = 1, j ≤ 625, j++, If[MatrixOfMinimalHammingDistancesForEachVector[[i]][[1]] > MatrixOfHammingDistances[[i]][[j]],
          MatrixOfMinimalHammingDistancesForEachVector[[i]][[1]] = MatrixOfHammingDistances[[i]][[j]]]]]

In[39]:= MatrixOfMinimalHammingDistancesForEachVector
Out[39]= {{0}, {0}, {1}, {1}, {0}, {2}, {1}, {0}, {1}, {0}}
```

Tworzymy i wypełniamy macierz MatrixOfAllPossibleVectorsWithConsideredHammingDistances, która przechowuje wszystkie możliwe wektory o wcześniej ustalonych długościach Hamminga:

```
In[44]:= MatrixOfAllPossibleVectorsWithConsideredHammingDistances = {{}, {}, {}, {}, {}, {}, {}, {}, {}, {}}

Out[44]= {{}, {}, {}, {}, {}, {}, {}, {}, {}, {}}

In[45]:= For[i = 1, i ≤ 10, i++, For[j = 1, j ≤ 625, j++, If[MatrixOfHammingDistances[[i]][[j]] == MatrixOfMinimalHammingDistancesForEachVector[[i]][[1]],
           AppendTo[MatrixOfAllPossibleVectorsWithConsideredHammingDistances[[i]], Transpose[AllCodedVectors][[j]]]]]]

In[46]:= MatrixOfAllPossibleVectorsWithConsideredHammingDistances

Out[46]= {{{1, 2, 1, 1, 1, 2, 4, 2, 4, 2, 2}}, {{4, 1, 4, 0, 0, 2, 1, 4, 1, 4, 3}}, {{1, 0, 1, 3, 3, 1, 4, 3, 3, 1, 2}}, {{4, 3, 3, 0, 0, 1, 1, 2, 4, 0, 2}}, {{4, 1, 1, 1, 1, 1, 1, 1, 2, 4, 0}}, {{2, 4, 4,
           0, 0, 3, 3, 1, 2, 0, 1}}, {{1, 3, 1, 3, 3, 0, 4, 4, 4, 4, 2}}, {{3, 3, 0, 2, 2, 1, 2, 2, 4, 2, 3}}, {{0, 4, 3, 0, 0, 3, 0, 1, 1, 2, 3}}, {{4, 2, 4, 2, 2, 0, 1, 1, 1, 1, 3}}}
```

Tworzymy finalną macierz, która będzie składała się z możliwych wylosowanych wektorów, zgodnych z algorytmem MinimizeHammingDistance:

```
In[47]:= FinalCodedMatrix = {}

Out[47]= {}

In[49]:= For[i = 1, i ≤ 10, i++, rand = RandomInteger[{1, Length[MatrixOfAllPossibleVectorsWithConsideredHammingDistances[[i]]]}]; AppendTo[FinalCodedMatrix,
         MatrixOfAllPossibleVectorsWithConsideredHammingDistances[[i]][[rand]]]]

In[50]:= FinalCodedMatrix

Out[50]= {{1, 2, 1, 1, 1, 2, 4, 2, 4, 2, 2}, {4, 1, 4, 0, 0, 2, 1, 4, 1, 4, 3}, {1, 0, 1, 3, 3, 1, 4, 3, 3, 1, 2}, {4, 3, 3, 0, 0, 1, 1, 2, 4, 0, 2}, {4, 1, 1, 1, 1, 1, 1, 1, 2, 4, 0}, {2, 4, 4, 0, 0, 3, 3,
         1, 2, 0, 1}, {1, 3, 1, 3, 3, 0, 4, 4, 4, 4, 2}, {3, 3, 0, 2, 2, 1, 2, 2, 4, 2, 3}, {0, 4, 3, 0, 0, 3, 0, 1, 1, 2, 3}, {4, 2, 4, 2, 2, 0, 1, 1, 1, 1, 3}}

In[51]:= MatrixForm[FinalCodedMatrix]
```

Out[51]//MatrixForm=

$$\begin{pmatrix} 1 & 2 & 1 & 1 & 1 & 2 & 4 & 2 & 4 & 2 & 2 \\ 4 & 1 & 4 & 0 & 0 & 2 & 1 & 4 & 1 & 4 & 3 \\ 1 & 0 & 1 & 3 & 3 & 1 & 4 & 3 & 3 & 1 & 2 \\ 4 & 3 & 3 & 0 & 0 & 1 & 1 & 2 & 4 & 0 & 2 \\ 4 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 4 & 0 \\ 2 & 4 & 4 & 0 & 0 & 3 & 3 & 1 & 2 & 0 & 1 \\ 1 & 3 & 1 & 3 & 3 & 0 & 4 & 4 & 4 & 4 & 2 \\ 3 & 3 & 0 & 2 & 2 & 1 & 2 & 2 & 4 & 2 & 3 \\ 0 & 4 & 3 & 0 & 0 & 3 & 0 & 1 & 1 & 2 & 3 \\ 4 & 2 & 4 & 2 & 2 & 0 & 1 & 1 & 1 & 1 & 3 \end{pmatrix}$$

Dokonujemy dekodowania otrzymanej macierzy, znajdując współrzędne rozważanych wektorów w bazie B:

```
In[96]:= DecodedMatrix = {}

Out[96]= {}

In[97]:= For[i = 1, i ≤ 10, i++, AppendTo[DecodedMatrix, LinearSolve[Transpose[G], FinalCodedMatrix[[i]], Modulus → 5]]]
```

g)

```
In[58]:= MatrixForm[DecodedMatrix]
Out[58]//MatrixForm=
```

$$\begin{pmatrix} 1 & 4 & 1 & 4 & 4 & 2 & 1 & 3 & 0 & 4 \\ 2 & 1 & 0 & 3 & 1 & 4 & 3 & 3 & 4 & 2 \\ 1 & 4 & 1 & 3 & 1 & 4 & 1 & 0 & 3 & 4 \\ 1 & 0 & 3 & 0 & 1 & 0 & 3 & 2 & 0 & 2 \end{pmatrix}$$

h)

Dla porównania macierz A:

```
In[9]:= MatrixForm[A]
Out[9]//MatrixForm=
```

$$\begin{pmatrix} 1 & 4 & 1 & 4 & 4 & 2 & 1 & 3 & 0 & 4 \\ 2 & 1 & 0 & 3 & 1 & 4 & 3 & 3 & 4 & 2 \\ 1 & 4 & 1 & 3 & 1 & 4 & 1 & 0 & 3 & 4 \\ 1 & 0 & 3 & 0 & 1 & 0 & 3 & 2 & 0 & 2 \end{pmatrix}$$

Widzimy, że wszystkie wektory zostały odkodowane poprawnie.

i)

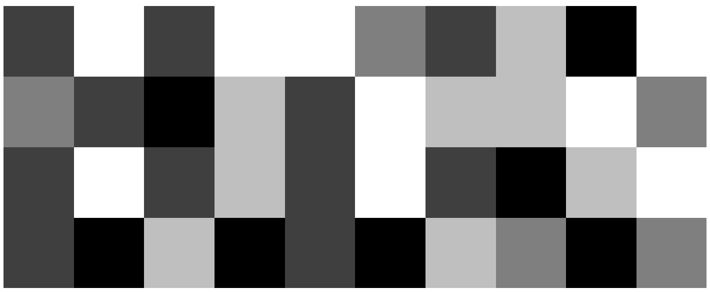
Dokonujemy normalizacji macierzy DecodedMatrix do przedziału [0,1] i generujemy obrazek:

In[59]:= NormalizedMatrix = DecodedMatrix / 4

Out[59]= $\left\{\left\{\frac{1}{4}, 1, \frac{1}{4}, 1, 1, \frac{1}{2}, \frac{1}{4}, \frac{3}{4}, 0, 1\right\}, \left\{\frac{1}{2}, \frac{1}{4}, 0, \frac{3}{4}, \frac{1}{4}, 1, \frac{3}{4}, \frac{3}{4}, 1, \frac{1}{2}\right\}, \left\{\frac{1}{4}, 1, \frac{1}{4}, \frac{3}{4}, \frac{1}{4}, 1, \frac{1}{4}, 0, \frac{3}{4}, 1\right\}, \left\{\frac{1}{4}, 0, \frac{3}{4}, 0, \frac{1}{4}, 0, \frac{3}{4}, \frac{1}{2}, 0, \frac{1}{2}\right\}\right\}$

In[61]:= Image[NormalizedMatrix, ImageSize → 300]

Out[61]=



Dla porównania obrazek z podpunktu b):

In[11]:= Image[B, ImageSize → 300]

Out[11]=