Raport realizacji Projektu 2

Kody liniowe

Aleksander Luckner, Piotr Wysocki

30 kwietnia 2024

Spis treści

1 Zadanko 1

W celu udowodnienia, że odległość Hamminga dana wzorem

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

jest metryką należy pokazać trzy własności:

1.1 Identyczność

Chcemy pokazać, że: $D(x, y) \Leftrightarrow x = y$
"$\Rightarrow$"
Niech $D(x, y) = 0$. Wówczas z definicji odległości Hamminga: $|\{i \in [n] : u_i \neq v_i\}| = 0$ Oznacza to, że na żadnej współrzędnej wektory u i v się nie różnią, zatem u = v.
" $\Leftarrow$ " Niech x = y. Wówczas:

$$D(x, y) = |\{i \in [n] : x_i \neq y_i\}|$$

Skoro x = y, to $D(x, y) = |\emptyset| = 0$ Zatem odległość Hamminga spełnia warunek identyczności.

1.2 Symetryczność

Chcemy pokazać, że: $D(x, y) = D(y, x)$

$$D(x, y) = |\{i \in [n] : x_i \neq y_i\}| = |\{i \in [n] : y_i \neq x_i\}| = D(y, x)$$

Zatem odległość Hamminga spełnia warunek symetryczności.

1.3 Przechodniość

Chcemy pokazać, że: $D(x, z) \leqslant D(x, y) + D(y, z)$ Ustalmy dowolne $i \in [n]$.
Wówczas mamy 2 przypadki:
(1) $x_i = z_i$
Wówczas $D(x_i, z_i) \leqslant D(x_i, y_i) + D(y_i, z_i)$ dla dowolnego $y \in V$
(2) $x_i \neq z_i$
Wówczas $D(x_i, z_i) = 1$. Zarazem $D(x_i, y_i) + D(y_i, z_i) \geqslant 1$. Gdyby tak nie było, to $x_i = y_i = z_i$ co daje sprzeczność z założeniem $x_i \neq z_i$.
Z dowolności i otrzymujemy, że przechodniość zachodzi dla dowolnych wektorów z V w odległości Hamminga.

Zatem skoro spełnione są wszystkie trzy warunki, to odległość Hamminga jest metryką.

2 Zadanko 2

Niech (n,k) - kod liniowy $\mathcal{C}$ nad skończonym ciałem $\mathbb{K}$. Zarazem, niech G będzie jego macierzą generującą powstałą z bazy kodu B. Należy pokazać, że wówczas wynikiem kodowania dowolnego wektora $v \in \mathbb{K}^k$ jest słowo kodowe kodu $\mathcal{C}$.
Kodowaniem wektora v jest wektor $w = (v^T G)^T$. Zauważmy, że macierz G generująca kod $\mathcal{C}$ dana jest wzorem:

$$G = \begin{pmatrix} e_1^T \\ e_2^T \\ \vdots \\ e_k^T \end{pmatrix}$$

Gdzie $(e_1, e_2, \dots e_k)$ jest bazą kodu $\mathcal{C}$. Oznacza to, że dla wektora $v^T = (a_1, a_2, \dots a_k)$ i dowolnego $i \in [n]$:

$$w_i^T = \sum_{j=1}^{k} a_j e_{ji}$$

Gdzie e_{ji} jest i-tą współrzędną j-tego wiersza bazy G. Zatem wektor w jest kombinacją liniową elementów z bazy G. Ponieważ $\mathcal{C}$ jest podprzestrzenią linową to $w \in \mathcal{C}$. $\square$

3 Zadanko 3

Przyjrzyjmy się algorytmowi:

`MinimizeHammingDistance(`$\mathcal{C}$`, `B`, `v`)`

IN: $\mathcal{C}$ – (n,k)–kod liniowy nad ciałem $\mathbb{K}$, B – baza kodu $\mathcal{C}$, v– dekodowany wektor

$m = \min\{d(v,w) : w \in \mathcal{C}\}$ # d to odległość Hamminga

$L = \{w \in \mathcal{C} : d(v,w) = m\}$

w = losowo wybrany wektor należący do L

r = wektor współczynników wektora w w bazie B

OUT: Wektor $r \in \mathbb{K}^k$

Niech wektor $V \in \mathcal{C}$. Wówczas $m = 0$, bo najbliższy wektor z $\mathcal{C}$ do v to v. Zarazem jest on jedynym wektorem odległym o 0 od v. Zatem r będzie wektorem współczynników wektora v w bazie B. Zastanówmy się teraz czym jest kodowanie wektora w bazie B. Wektor otrzymany z kodowania wektora będzie kombinacją liniową elementów z bazy B, gdzie n-ty element bazy przemnożony jest przez n-tą współrzędną wektora kodowanego. Zatem mnożąc wektor współczynników wektora v w bazie B przez bazę otrzymamy wektor v. Stąd dekodując wektor algorytmem `MinimizeHammingDistance` a następnie kodując otrzymany wektor otrzymamy ten sam wektor co na początku.

4 Zadanko 4

Niech $u, v, x \in \mathbb{K}^n$ i niech $m = D(u, v)$. Chcemy pokazać, że $m = D(x + u, x + v)$.
Z definicji odległości Hamminga:

$$m = D(u,v) = |\{i \in [n] : u_i \neq v_i\}| = |\{i \in [n] : u_i + (x_i - x_i) \neq v_i\}| = \\ |\{i \in [n] : u_i + x_i \neq v_i + x_i\}| = D(u + x, v + x) \quad (1)$$

Zatem dla dowolnych $x, u, v \in \mathbb{K}^n$

$$D(u, v) = m = D(u + x, v + x)$$

5 Zadanko 5

Do rozwiązania zadania 5 skorzystano z programu Mathemathica.
Najpierw obliczono odległość wektorów $(1, 2, 0, 1)^T i (0, 0, 0, 1)^T$ stosując poniższy kod.

```
a = {1,2,0,1}
b = {0,0,0,1}
d = HammingDistance[a,b]
```

Otrzymana odległość Hamminga tych wektorów to: 2.

Następnie zaimplementowane zostały kolejne wektory z zadania.

```
v1 = {1,2,1,2,0}
v2 = {1,1,1,1,1}
v3 = {0,0,2,1,1}
v4 = {2,2,2,1,0}
```

Oraz sprawdzono odległość pomiędzy nimi.

```
HammingDistance[v1,v2]
HammingDistance[v1,v3]
HammingDistance[v1,v4]
HammingDistance[v2,v3]
HammingDistance[v2,v4]
HammingDistance[v3,v4]
```

Otrzymując kolejno odległości: 3,5,3,3,4,3 Oznacza to, że $D(v1, v2) = D(v1, v4) = D(v2, v3) = D(v3, v4) = 3$ są najmniejszymi odległościami względem wektorów.
Wektory, które są najbliżej położone to wektory v2 i v4, których suma odległości względem pozostałych wektorów wynosi 10.

6 Zadanko 6

Do wyznaczania słów kodowych (5,3) - kodu liniowego $\mathcal{C}$ stworzono w Pythonie następujący program:

```
x = [1,0,0,2,4]
y = [0,1,0,1,0]
z = [0,0,1,5,6]
G = [x,y,z]
v = [3,6,1,3,4] # dowolny wektor
tab=[]
for a in range(7):
    for b in range(7):
        for c in range(7):
            t = [0, 0, 0, 0, 0]
            for i in range(5):
                t[i] += a * x[i] + b * y[i] + c * z[i] #
kombincacja liniowa wektorow
            for i in range(3,5):
                if t[i] >= 7:
                    t[i] = t[i]%7 # normowanie
            tab.append(t)
print(tab)
```

W ten sposób otrzymano wszystkie słowa kodowe:

```
[[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]]
```

7 Zadanko 7

Do rozwiązania zadanka 7 stworzono kolejny skrypt w Pythonie. W celu sprawdzenia działania programu zdefiniowano wektor $v = (3, 6, 1, 3, 4)$.

```
import random
m=5
wektor=[]
for u in tab: # szukam minimalnego m
    d_uv = 0
    for i in range(5):
        if u[i] != v[i]:
            d_uv += 1
    if d_uv < m:
        m= d_uv
print(m)
for u in tab: # tworze tablice wektor z wektorami
spelniajacymi warunki z algorytmu
    d_uv = 0
    for i in range(5):
        if u[i] != v[i]:
            d_uv += 1
    if d_uv == m:
        wektor.append(u)
w = wektor[random.randint(0,len(wektor)-1)] #losowy
print(w)
for a in range(7):
    for b in range(7):
        for c in range(7):
            nowy = [0, 0, 0, 0, 0]
            for i in range(5):
                nowy[i]=a*x[i] + b*y[i] + c*z[i]
                if nowy[i] >= 7:
                    nowy[i] = nowy[i]%7
            if nowy == w: # robie petle po wszystkich
wektorach z bazy i patrze dla jakich wspolczynnikow jest
on rowny w
                r=[a,b,c]
print(r)
# sprawdzenie
wynik = [0 for x in range(5)]
for i in range(5):
    wynik[i]+=r[0]*x[i] + r[1] * y[i] + r[2]*z[i]
    if wynik[i] >=7:
        wynik[i] = wynik[i] % 7
print(wynik)
```

Wyjście otrzymane dla tego wektora wynosi:

```
    # dekodowanie v
    w = (3,6,1)
    # sprawdzenie czy v jest faktycznie kodowaniem w
    v = (3,6,1,3,4)
```

Rozważmy inny wektor $v = (3, 3, 5, 2, 1)$. Wówczas program zwróci:

```
    w = (5,2,5)
    v = (5,2,5,2,1)
```

Jest to przykład, że kodowanie i dekodowanie wektorów jest niejednoznaczne.

8 Zadanko 8

Wszystkie skrypty przedstawione w zadanku 8 zostały stworzone w programie Python.

8.1 Generowanie macierzy w $\mathbb{Z}_5$

```
macierz = [[0 for x in range(10)]for y in range(4)]
import random, copy
for x in range(4):
    for y in range(10):
        macierz[x][y] = random.randint(0,4) # macierz z losowymi wartosciami
print(macierz)
v=[[]for x in range(10)]
for i in range(10):
    for x in range(4):
        v[i].append(macierz[x][i])
print(v) # odwrotnie wiersze z kolumnami
```

Wywołując program otrzymaliśmy macierz:

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

8.2 Unormowanie macierzy

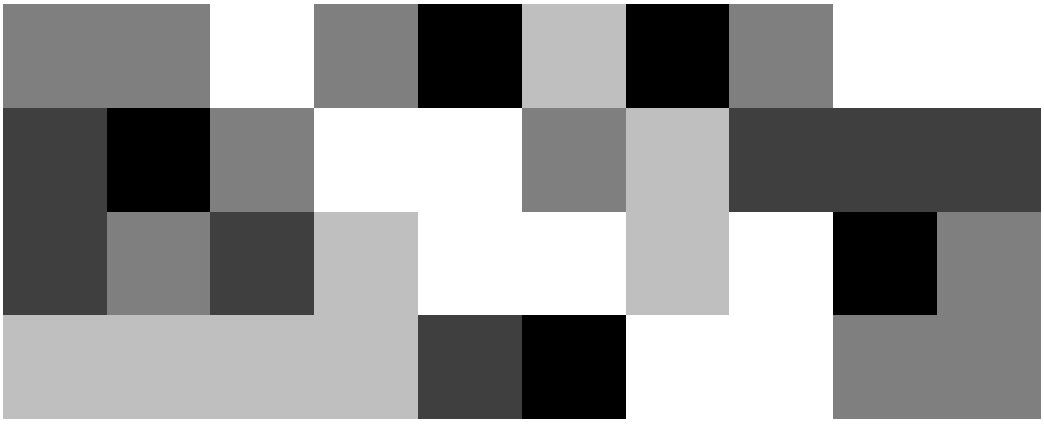
Macierz z poprzedniego podpunktu została unormowana przez kolejny skrypt:

```
new_w = copy.deepcopy(w)
for x in range(len(w)):
    for y in range(len(w[0])):
        p = random.randint(1,20)
        if p == 1: #prawdopodobienstwa 5%
            new_w[x][y] += 3 #dodajemy 3
            new_w[x][y] %= 5
print(new_w)
```

Program zwraca unormowaną macierz z poprzedniego podpunktu:

$$\begin{pmatrix} 0.5 & 0.5 & 1.0 & 0.5 & 0.0 & 0.75 & 0.0 & 0.5 & 1.0 & 1.0 \\ 0.25 & 0.0 & 0.5 & 1.0 & 1.0 & 0.5 & 0.75 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.5 & 0.25 & 0.75 & 1.0 & 1.0 & 0.75 & 1.0 & 0.0 & 0.5 \\ 0.75 & 0.75 & 0.75 & 0.75 & 0.25 & 0.0 & 1.0 & 1.0 & 0.5 & 0.5 \end{pmatrix}$$

W programie Mathematica wygenerowano obraz macierzy:



Rysunek 1: Obraz macierzy wygenerowany funkcją `Image[]`

8.3 (11,4) - kod liniowy $\mathcal{C}$ nad $\mathbb{Z}_5$

Zauważmy najpierw, że liczba kolumn(11) i wierszy (4) zgadzają się z wymiarami potrzebnymi do tego aby G było macierzą generującą (11,4) - kodu liniowego $\mathcal{C}$. Zarazem wszystkie elementy G należą do ciała $\mathbb{Z}_5$. W takim razie wystarczy nam pokazać, że wiersze G generują przestrzeń wymiaru 4(są liniowo niezależne).

$$\alpha_1 r_1 + \alpha_2 r_2 + \alpha_3 r_3 + \alpha_4 r_4 = 0$$

Gdzie r_i dla $i \in [4]$ są kolejnymi wierszami G. Mamy zatem układ równań, któremu odpowiada $G^T = 0$. Spójrzmy na pierwsze 4 wiersze G^T

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Stąd wiemy, liniową niezależność kolumn. Zatem $\dim G = 4$ i musi istnieć (11,4) - kod linowy $\mathcal{C}$ taki, że G jest jego macierzą generującą. $\square$

8.4 Kodowanie wektora

W celu zakodowania macierzy wygenerowanej w podpunkcie a) za pomocą macierzy generującej G stworzono skrypt:

```
#---------kodowanie----------
w= [[0 for y in range(11)]for x in range(10)]
```

```
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]]
for j in range(10):
    for x in range(11):
        for y in range(4):
            w[j][x]+= G[y][x] * v[j][y] #mnozymy
        w[j][x] = w[j][x] % 5 #cialo z5
print(w)
```

Zatem jak ustalimy wektor $v = (2, 1, 1, 3)$ będący pierwszą kolumną macierzy z pierwszego podpunktu, to jego kodowaniem będzie pierwszy wiersz macierzy otrzymanej ze skryptu. Zatem w będący kodowaniem v wynosi:

$$w = (2, 1, 1, 3, 3, 3, 3, 2, 0, 3, 3)$$

8.5 Przesyłanie przez kanał

Symulujemy przesłanie wektora przez kanał do użytkownika poprzez danie $0,05\%$ szans na dodanie do wektora 3, wpp. nie dodajemy nic.

```
new_w = copy.deepcopy(w)
for x in range(len(w)):
    for y in range(len(w[0])):
        p = random.randint(1,20)
        if p == 1: #prawdopodobienstwa 5%
            new_w[x][y] += 3 #dodajemy 3
            new_w[x][y] %= 5
print(new_w)
```

Przykładowo dla wektora w otrzymanego w poprzednim podpunkcie nowy wektor w wynosi:

$$new\_w = (2, 1, 4, 3, 1, 3, 3, 2, 0, 3, 3)$$

Cała macierz wynosi:

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

8.6 Odkodowywanie wektora przesłanego przez kanał

Najpierw stworzony został skrypt do stworzenia tablicy w wektorami z przestrzeni $\mathcal{C}$:

```
C=[]
for a in range(5):
    for b in range(5):
        for c in range(5):
            for d in range(5):
                t = [0 for x in range(11)]
                for i in range(11):
                    t[i] += a * G[0][i] + b * G[1][i] + c
 * G[2][i] + d* G[3][i]
                for i in range(11):
                    if t[i] >=5:
                        t[i] = t[i]%5
                C.append(t)
```

Następnie w sposób analogiczny do sposobu z **Zadanka 7** zaimplementowano algorytm `MinimizeHammingDistance`.

```
for v in new_w: # algorytm Hamminga z 7 dla innych
wartosci (analogiczny)
m=11
wektor = []
for u in C:
    d_uv=0
    for i in range(11):
        if u[i] != v[i]:
            d_uv += 1
    if d_uv < m:
        m = d_uv
print("m= ",m)
for u in C:
    d_uv = 0
    for i in range(11):
        if u[i] != v[i]:
            d_uv += 1
    if d_uv == m:
        wektor.append(u)
w = wektor[random.randint(0, len(wektor) - 1)]
print("w=",w)
for a in range(5):
    for b in range(5):
        for c in range(5):
            for d in range(5):
                nowy = [0 for x in range(11)]
                for i in range(11):
                    nowy[i] = a * G[0][i] + b * G[1][i] +
```

```
    c * G[2][i] + d*G[3][i]
                        if nowy[i] >= 5:
                            nowy[i] = nowy[i] % 5
                    if nowy == w:
                        r = [a, b, c, d]
    wynik.append(r)
print(wynik)
```

Przykładowe początkowe wyjście(dla wektora wcześniej omawianego) wynosi:

$$\mathrm{m} = 2, \quad \mathrm{w} = [2, 1, 1, 3, 3, 3, 3, 2, 0, 3, 3]$$

Gdzie m to odległość wektora od najbliższego wektora w należącego do $\mathcal{C}$. Na koniec wyjścia zwracana jest macierz zdekodowanych wektorów:

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

8.7 Ponowne kodowanie

Mając już odkodowaną macierz stwórzmy macierz odpowiadającą macierzy z **8.1**. W tym celu transponujemy macierz z **8.6**.

```
    odpowiadajaca_kodowanej = [[0 for x in range(10)]for y in range(4)]
    for x in range(4):
        for y in range(10):
            odpowiadajaca_kodowanej[x][y] = wynik[y][x]
    print(odpowiadajaca_kodowanej)
```

Otrzymujemy macierz:

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

8.8 Porównanie macierzy z 8.1 i 8.6

Do porównania macierzy stworzono kod:

```
kolumny = 0
for y in range(10): # sprawdzam czy kolumny sa rowne
    czy_rowne= True
    for x in range(4):
        if odpowiadajaca_kodowanej[x][y] != macierz[x][y]:
            czy_rowne = False
    if czy_rowne:
        kolumny +=1
print(kolumny)
```

Dostajemy, że 10 kolumn jest sobie równych zatem macierze w obydwu podpunktach są sobie równe.
W naszym przypadku przesyłanie przez kanał nie wpłynęło znacząco na wektory. Może jednak dojść do sytuacji, gdzie przesyłanie wektorów przez kanał zmieni wartości w taki sposób, że otrzymalibyśmy inną macierz. Oznacza to też, że w kolejnym podpunkcie powinniśmy dostać macierz unormowaną jak i obraz takie same jak w podpunkcie **8.2**.

8.9 Normowanie macierzy

Macierz jest normowana skryptem:

```
    for x in range(4):
        for y in range(10):
            odpowiadajaca_kodowanej[x][y] /= 4
```

Ponieważ macierze są sobie równe to macierz unormowana powinna być równa macierzy z **8.2**.

$$\begin{pmatrix} 0.5 & 0.5 & 1.0 & 0.5 & 0.0 & 0.75 & 0.0 & 0.5 & 1.0 & 1.0 \\ 0.25 & 0.0 & 0.5 & 1.0 & 1.0 & 0.5 & 0.75 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.5 & 0.25 & 0.75 & 1.0 & 1.0 & 0.75 & 1.0 & 0.0 & 0.5 \\ 0.75 & 0.75 & 0.75 & 0.75 & 0.25 & 0.0 & 1.0 & 1.0 & 0.5 & 0.5 \end{pmatrix}$$

Analogicznie jak w **8.2** generujemy w Mathematice obraz tej macierzy. On również powinien być taki sam jak w **8.2**.