Matematyka Algebra 3
Mały Projekt 5
Tymon Zadara IIR2
Kod Hamminga

Zadanie 1: Skonstruować macierz kontroli parzystości dla kodu H3. Zakodować wiadomość u = 1010. Zakładając, że w trakcie transmisji został popełniony jeden błąd odkodować słowo f = 0011111.

Dane:
Kod: H3
Wiadomość pocz.: 1010
Wiadomość końc.: 0011111
Licz. Błąd.: 1
m = 3 - liczba bitów kontrolnych
n = 7 (✓) - liczba bitów zakodowanej wiadomości
k = 4(✓) - liczba bitów informacji
n = k+m

Układ bitów wiadomości:

[C1,C2,1,C3,0,1,0] gdzie C - check (weryfikacja poprawności); 1,0 to bity przesyłanej wiadomości

Bit kontrolny C1 (pozycja 1): sprawdza pozycje, których bity binarne mają 1. bit = 1
➤ sprawdza: 1(1), 3(11), 5(101), 7(111)

C2 (pozycja 2): sprawdza pozycje z 2. bitem binarnym = 1
➤ sprawdza: 2(10), 3(11), 6(110), 7(111)

C3 (pozycja 4): sprawdza pozycje z 3. bitem binarnym = 1
➤ sprawdza: 4(100), 5(101), 6(110), 7(111)

Macierz kontroli H3:
H3 = [1 0 1 0 1 0 1]
[0 1 1 0 0 1 1]
[0 0 0 1 1 1 1]

Metoda 1(kod):

In[291]:=

```
H3 = Transpose[IntegerDigits[Range[1, 7], 2, 3]]
H3 = H3[[{3, 2, 1}]]
MatrixForm[H3]
```

Out[291]=

```
{{0, 0, 0, 1, 1, 1, 1}, {0, 1, 1, 0, 0, 1, 1}, {1, 0, 1, 0, 1, 0, 1}}
```

Out[292]=

```
{{1, 0, 1, 0, 1, 0, 1}, {0, 1, 1, 0, 0, 1, 1}, {0, 0, 0, 1, 1, 1, 1}}
```

Out[293]//MatrixForm=

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

In[294]:=

```
u = {1, 0, 1, 0};



(* Obliczenie bitów parzystości *)
p1 = Mod[u[[1]] + u[[2]] + u[[4]], 2];
p2 = Mod[u[[1]] + u[[3]] + u[[4]], 2];
p3 = Mod[u[[2]] + u[[3]] + u[[4]], 2];

(* Złożenie zakodowanej wiadomości (Hamming(7,4)) *)
hammingCode = {p1, p2, u[[1]], p3, u[[2]], u[[3]], u[[4]]};

hammingCode
```

Out[299]=

```
{1, 0, 1, 1, 0, 1, 0}
```

In[300]:=

Metoda 2(kod):

In[301]:=

```
dataPositions = Complement[Range[7], {1, 2, 4}];



code = ConstantArray[0, 7];



(*uzupełnienie wartości z u*)
Do[
 code[[dataPositions[[i]]]] = u[[i]],
 {i, Length[u]}
];
(*funkcja uzupełniająca bity parzystości*)
Do[
 positions = Select[Range[7], BitAnd[#, 2^(p - 1)] > 0 &];
 code[[2^(p - 1)]] = Mod[Total[code[[positions]]], 2],
 {p, 1, 3}
];

code
```

Out[305]=

```
{1, 0, 1, 1, 0, 1, 0}
```

Sprawdzenie:

Bity parzystości: C1: 1+0+0 mod 2 = 1
C2: 1+1+0 mod 2 = 0
C3: 0+1+0 mod 2 = 1

Zakodowane slowo u = {1,0,1,1,0,1,0}

In[306]:=

Odkodowanie Teoria:

f = {0,0,1,1,1,1,1}
f = {C1,C2,1,C3,1,1,1}
C1 = 0 = 1+1+1 mod 2 = 1 ---> Błąd ---> może być w bit 3/5/7
C2 = 0 = 1+1+1 mod 2 = 1 ---> Błąd ---> może być w bit 3/6/7
C3 = 1 = 1+1+1 mod 2 = 1 ---> Poprawnie ---> nie może być w 5/6/7

Z tego wynika, ze błąd jest w bicie 3

Odkodowane słowo: 0111

Kod:

In[307]:=
```
f = {0, 0, 1, 1, 1, 1, 1};
C1 = f[[1]];
C2 = f[[2]];
C3 = f[[4]];
popr = {};
blad = {};
```

In[313]:=
```
If[C1 == Mod[f[[3]] + f[[5]] + f[[7]], 2], {AppendTo[popr, 3], AppendTo[popr, 5], AppendTo[popr, 7]},
  {AppendTo[blad, 3], AppendTo[blad, 5], AppendTo[blad, 7]}];
```

In[314]:=
```
If[C2 == Mod[f[[3]] + f[[6]] + f[[7]], 2], {AppendTo[popr, 3], AppendTo[popr, 6], AppendTo[popr, 7]},
  {AppendTo[blad, 3], AppendTo[blad, 6], AppendTo[blad, 7]}];
If[C3 == Mod[f[[5]] + f[[6]] + f[[7]], 2], {AppendTo[popr, 5], AppendTo[popr, 6], AppendTo[popr, 7]},
  {AppendTo[blad, 5], AppendTo[blad, 6 ], AppendTo[blad, 7]}];
```

In[316]:=
```
wynik = DeleteDuplicates[DeleteCases[blad, Alternatives @@ popr]]
```
Out[316]=
```
{3}
```

Potwierdza się z obliczonym teorytycznie

In[317]:=
```
(Mod[H3.f, 2])
```
Out[317]=
```
{1, 1, 0}
```

Również potwierdza występowanie błędu na pozycji 3.

Zadanie 2: Skonstruować macierz kontroli parzystości dla kodu H4. Zakodować wiadomość u = 11101110000. Zakładając, że w trakcie transmisji został popełniony jeden błąd odkodować słowo f = 11101010000000.

Analogicznie jak w zadaniu 1 przeprowadzone są te same działania tylko dla macierzy H4

Metoda 1(kod):

In[318]:=
```
ClearAll
H4 = Transpose[IntegerDigits[Range[1, 15], 2, 4]];
H4 = H4[[{4, 3, 2, 1}]];
MatrixForm[H4]
```
Out[318]=
```
ClearAll
```

Out[321]//MatrixForm=

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

In[322]:=

```
u = {1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0};



(* Obliczenie bitów parzystości *)
p1 = Mod[u[[1]] + u[[2]] + u[[4]] + u[[5]] + u[[7]] + u[[9]] + u[[11]], 2];
p2 = Mod[u[[1]] + u[[3]] + u[[4]] + u[[6]] + u[[7]] + u[[10]] + u[[11]], 2];
p3 = Mod[u[[2]] + u[[3]] + u[[4]] + u[[8]] + u[[9]] + u[[10]] + u[[11]], 2];
p4 = Mod[u[[5]] + u[[6]] + u[[7]] + u[[8]] + u[[9]] + u[[10]] + u[[11]], 2];

(* Złożenie zakodowanej wiadomości (Hamming(7,4)) *)
hammingCode =
  {p1, p2, u[[1]], p3, u[[2]], u[[3]], u[[4]], p4, u[[5]], u[[6]], u[[7]], u[[8]], u[[9]], u[[10]], u[[11]]};

hammingCode
```

Out[328]=

```
{0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0}
```

In[329]:=

Metoda 2(kod):

In[330]:=

```
dataPositions = Complement[Range[15], {1, 2, 4, 8}];



code = ConstantArray[0, 15];



(*uzupełnienie wartości z u*)
Do[
 code[[dataPositions[[i]]]] = u[[i]],
 {i, Length[u]}
];
(*funkcja uzupełniająca bity parzystości*)
Do[
 positions = Select[Range[15], BitAnd[#, 2^(p - 1)] > 0 &];
 code[[2^(p - 1)]] = Mod[Total[code[[positions]]], 2],
 {p, 1, 4}
];

code
```

Out[334]=

```
{0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0}
```

Odkodowanie 2:

In[335]:=

```
f = {1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0};
C1 = f[[1]];
C2 = f[[2]];
C3 = f[[4]];
C4 = f[[8]]
popr = {};
blad = {};
```

Out[339]=

```
0
```

In[342]:=

```
If[C1 == Mod[f[[3]] + f[[5]] + f[[7]] + f[[9]] + f[[11]] + f[[13]] + f[[15]], 2],
  {AppendTo[popr, 3], AppendTo[popr, 5], AppendTo[popr, 7], AppendTo[popr, 9],
   AppendTo[popr, 11], AppendTo[popr, 13], AppendTo[popr, 15]},
  {AppendTo[blad, 3], AppendTo[blad, 5], AppendTo[blad, 7], AppendTo[blad, 9],
   AppendTo[blad, 11], AppendTo[blad, 13], AppendTo[blad, 15]}];
If[C2 == Mod[f[[3]] + f[[6]] + f[[7]] + f[[10]] + f[[11]] + f[[14]] + f[[15]], 2],
  {AppendTo[popr, 3], AppendTo[popr, 6], AppendTo[popr, 7], AppendTo[popr, 10],
   AppendTo[popr, 11], AppendTo[popr, 14], AppendTo[popr, 15]},
  {AppendTo[blad, 3], AppendTo[blad, 6], AppendTo[blad, 7], AppendTo[blad, 10],
   AppendTo[blad, 11], AppendTo[blad, 14], AppendTo[blad, 15]}];
If[C3 == Mod[f[[5]] + f[[6]] + f[[7]] + f[[12]] + f[[13]] + f[[14]] + f[[15]], 2],
  {AppendTo[popr, 5], AppendTo[popr, 6], AppendTo[popr, 7], AppendTo[popr, 12],
   AppendTo[popr, 13], AppendTo[popr, 14], AppendTo[popr, 15]},
  {AppendTo[blad, 5], AppendTo[blad, 6], AppendTo[blad, 7], AppendTo[blad, 12],
   AppendTo[blad, 13], AppendTo[blad, 14], AppendTo[blad, 15]}];
If[C4 == Mod[f[[9]] + f[[10]] + f[[11]] + f[[12]] + f[[13]] + f[[14]] + f[[15]], 2],
  {AppendTo[popr, 9], AppendTo[popr, 10], AppendTo[popr, 11], AppendTo[popr, 12],
   AppendTo[popr, 13], AppendTo[popr, 14], AppendTo[popr, 15]},
  {AppendTo[blad, 9], AppendTo[blad, 10], AppendTo[blad, 11], AppendTo[blad, 12],
   AppendTo[blad, 13], AppendTo[blad, 14], AppendTo[blad, 15]}];
```

In[350]:=

```
wynik = DeleteDuplicates[DeleteCases[blad, Alternatives @@ popr]]
```

Out[350]=

```
{}
```

Ten prosty algorytm pozwala na sprawdzenie poprawności działania - metoda weryfikacji

In[347]:=

```
(Mod[H4.f, 2])
```

Out[347]=

```
{0, 1, 0, 0}
```

Błędnym bitem jest bit 2

Poprawnie odkodowana wiadomość:

{1,1,0,1,0,0,0,0,0,0}

W przypadku odkodowania w obu zadaniach zostały zastosowane dwie metody:

1. Mod[H4.f,2]) służąca do znalezienia bitu błędnego
2. dodawanie do list popr oraz blad - ta metoda służy do weryfikacji czy poprawnie został napisany kod i czy nie popełniony został gdzieś błąd (nie jest to potrzebne ale nie szkodzi programowi)