Politechnika Poznańska
Wydział Informatyki i Telekomunikacji
Elektronika i Telekomunikacja

Projekt z przedmiotu

Teoria Informacji i Kodowania

Maciej Niedźwiecki
nr albumu: 147973

Prowadzący:
dr inż. Krzysztof Cichoń

Poznań, 23.06.2024 r.

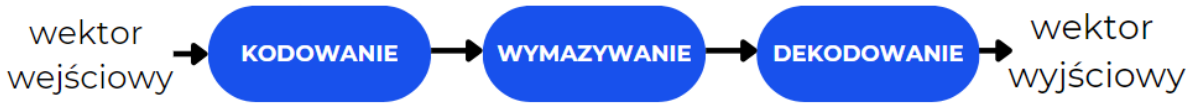
SPIS TREŚCI

1. CZYM JEST KODER POLARYZACYJNY

Koder polaryzacyjny jest technologią stosowaną w telekomunikacji, szczególnie w kontekście kodowania kanałowego w celu poprawy niezawodności transmisji danych. Został on zaproponowany przez Erdala Arikana w 2009 roku i jest uznawany za przełomowy w dziedzinie kodowania kanałowego. Wykorzystuje zjawisko polaryzacji kanałów, które pozwala na efektywne kodowanie i dekodowanie informacji. Proces ten polega na operacjach macierzowych, w tym na operacjach z użyciem macierzy Kroneckera.

2. ZAŁOŻENIA PROJEKTOWE

Zasymuluj działanie kodera i dekodera polaryzacyjnego. Użytkownik wprowadza bity wejściowe wektora. Kodowanie odbywa się za pomocą macierzy Kroneckera o wymiarach F4, F8, F16, F32. Ponadto występuje wymazywanie stałe dla kodu o stałej wielkości. Mamy pewność kiedy wystąpiło zero. Ostatecznie znajdź wymazane bity przeprowadzając dekodowanie.



3. WPROWADZANIE DANYCH PRZEZ UŻYTKOWNIKA

Użytkownik wybiera długość wektora wejściowego spośród dostępnych opcji. Następnie może samodzielnie wprowadzić wartości wektora lub wygenerować je automatycznie. Po wykonaniu tych operacji, program wyświetla wyniki wprowadzonych danych. Program jest również przygotowany na drobne błędy i sprawdza, czy wprowadzone wartości są zgodne z wymaganiami.

```
Podaj długość wektora wejściowego n^2 (4, 8, 16, 32): 7
Nieprawidłowa długość. Spróbuj ponownie.
Podaj długość wektora wejściowego n^2 (4, 8, 16, 32): 4
Czy chcesz wprowadzić wektor? (TAK - wprowadź, NIE - wygeneruj losowy): tAk
Wprowadź wektor wejściowy o długości 4 (format: 1 0 1 1): 1 0 2 1 4
Nieprawidłowe dane, wektor powinien zawierać 4 bitów i składać się z 0 i 1. Spróbuj ponownie.
Wprowadź wektor wejściowy o długości 4 (format: 1 0 1 1): 1 1 1 0
```

```
Podaj długość wektora wejściowego n^2 (4, 8, 16, 32): 16
Czy chcesz wprowadzić wektor? (TAK - wprowadź, NIE - wygeneruj losowy): nie

Wygenerowany losowy wektor wejściowy o długości 16: [1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0]
```

4. KODOWANIE

Na podstawie podstawowej macierzy Arikana (F2) generowane są kolejne macierze w sposób iteracyjny, wykorzystując iloczyn Kroneckera i operacje modulo 2. W programie utworzono słownik, który zgodnie z założeniami zadania przechowuje wykorzystywane macierze Kroneckera (F4, F8, F16, F32). Po utworzeniu odpowiedniej macierzy, jest ona wyświetlana w programie.

$$F_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Macierz Arikana

$$F_4 = F_2 \otimes F_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$F_8 = F_2 \otimes F_4 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Sposób tworzenia macierzy Kroneckera

```
Wykorzystywana macierz Kroneckera F4:
[[1 0 0 0]
 [1 1 0 0]
 [1 0 1 0]
 [1 1 1 1]]
```

```
Wykorzystywana macierz Kroneckera F8:
[[1 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0]
 [1 1 1 1 0 0 0 0]
 [1 0 0 0 1 0 0 0]
 [1 1 0 0 1 1 0 0]
 [1 0 1 0 1 0 1 0]
 [1 1 1 1 1 1 1 1]]
```

```
Wykorzystywana macierz Kroneckera F16:
[[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0]
 [1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0]
 [1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0]
 [1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0]
 [1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]]
```

```
Wykorzystywana macierz Kroneckera F32:
[[1 0 0 ... 0 0 0]
 [1 1 0 ... 0 0 0]
 [1 0 1 ... 0 0 0]
 ...
 [1 1 0 ... 1 0 0]
 [1 0 1 ... 0 1 0]
 [1 1 1 ... 1 1 1]]
```

Następnie kodujemy wprowadzony wektor wejściowy, mnożąc go przez utworzoną macierz Kroneckera i wykonując operację modulo 2, która odpowiada działaniu bramek XOR.

Schemat działania został zaprezentowany w przypadku kodowania wektora **[1, 1, 1, 0]**.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 \\ 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 \\ 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 \\ 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 2 \\ 2 \\ 3 \end{bmatrix} \mod 2 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Wektor **[1, 1, 1, 0]** został zakodowany w wektor **[1, 0, 0, 1]**, co jest tożsame z wynikiem symulacji.

```
Wprowadzony wektor wejściowy u: [1, 1, 1, 0]
Wykorzystywana macierz Kroneckera F4:
[[1 0 0 0]
 [1 1 0 0]
 [1 0 1 0]
 [1 1 1 1]]

Zakodowany wektor u: [1 0 0 1]
```

5. WYMAZYWANIE

Zgodnie z przyjętym wymazywaniem stałym ustalono wymazywanie co drugiego bitu w zakodowanym wektorze. Wymazany bit jest oznaczany jako “*x{i}*”, gdzie “*i*” oznacza jego indeks na liście. Ponadto zgodnie z założeniami zadania, jeśli w miejscu bitu znajduje się 0, to pozostaje ono bez zmian, ponieważ mamy pewność co do jego występowania.

Reprezentacja wymazywania dla wprowadzonego wektora **[1, 1, 1, 0]**.

```
Zakodowany wektor u: [1 0 0 1]
Odebrany wektor: [1, 0, 0, 'x3']
```

6. DEKODOWANIE

Proces dekodowania rozpoczynamy od zdefiniowania macierzy odwrotnej do macierzy Kroneckera. Następnie, ze względu na operacje możliwe jedynie na bitach 0 lub 1, wykonujemy operację modulo 2. Co ciekawe, macierz odwrotna do macierzy Kroneckera jest zawsze taka sama.

```
Odwrotna macierz Kroneckera F4:
[[ 1  0  0  0]
 [-1  1  0  0]
 [-1  0  1  0]
 [ 1 -1 -1  1]]
Wykorzystywana odwrotna macierz Kroneckera (F4 mod 2):
[[1 0 0 0]
 [1 1 0 0]
 [1 0 1 0]
 [1 1 1 1]]
```

```
Odwrotna macierz Kroneckera F8:
[[ 1  0  0  0  0  0  0  0]
 [-1  1  0  0  0  0  0  0]
 [-1  0  1  0  0  0  0  0]
 [ 1 -1 -1  1  0  0  0  0]
 [-1  0  0  0  1  0  0  0]
 [ 1 -1  0  0 -1  1  0  0]
 [ 1  0 -1  0 -1  0  1  0]
 [-1  1  1 -1  1 -1 -1  1]]
Wykorzystywana odwrotna macierz Kroneckera (F8 mod 2):
[[1 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0]
 [1 1 1 1 0 0 0 0]
 [1 0 0 0 1 0 0 0]
 [1 1 0 0 1 1 0 0]
 [1 0 1 0 1 0 1 0]
 [1 1 1 1 1 1 1 1]]
```

```
Odwrotna macierz Kroneckera F16:
[[ 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [-1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [-1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 1 -1 -1  1  0  0  0  0  0  0  0  0  0  0  0  0]
 [-1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 1 -1  0  0 -1  1  0  0  0  0  0  0  0  0  0  0]
 [ 1  0 -1  0 -1  0  1  0  0  0  0  0  0  0  0  0]
 [-1  1  1 -1  1 -1 -1  1  0  0  0  0  0  0  0  0]
 [-1  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0]
 [ 1 -1  0  0  0  0  0  0 -1  1  0  0  0  0  0  0]
 [ 1  0 -1  0  0  0  0  0 -1  0  1  0  0  0  0  0]
 [-1  1  1 -1  0  0  0  0  1 -1 -1  1  0  0  0  0]
 [ 1  0  0  0 -1  0  0  0 -1  0  0  0  1  0  0  0]
 [-1  1  0  0  1 -1  0  0  1 -1  0  0 -1  1  0  0]
 [-1  0  1  0  1  0 -1  0  1  0 -1  0 -1  0  1  0]
 [ 1 -1 -1  1 -1  1  1 -1 -1  1  1 -1  1 -1 -1  1]]
```

```
Wykorzystywana odwrotna macierz Kroneckera (F16 mod 2):
[[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0]
 [1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0]
 [1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0]
 [1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0]
 [1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0]
 [1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]]
```

```
Odwrotna macierz Kroneckera F32:
[[ 1  0  0 ...  0  0  0]
 [-1  1  0 ...  0  0  0]
 [-1  0  1 ...  0  0  0]
 ...
 [ 1 -1  0 ...  1  0  0]
 [ 1  0 -1 ...  0  1  0]
 [-1  1  1 ... -1 -1  1]]
Wykorzystywana odwrotna macierz Kroneckera (F32 mod 2):
[[1 0 0 ... 0 0 0]
 [1 1 0 ... 0 0 0]
 [1 0 1 ... 0 0 0]
 ...
 [1 1 0 ... 1 0 0]
 [1 0 1 ... 0 1 0]
 [1 1 1 ... 1 1 1]]
```

Następnie mnożymy odebrany, wymazany wektor przez odpowiednio wygenerowaną macierz odwrotną. W rezultacie otrzymujemy poprawną kolejność bitów 0 i 1. W celu uproszczenia procesu dekodowania zakładamy, że wymazywany symbol wynosił 1. Na koniec wykonujemy ewentualne przekształcenia i operację modulo 2, symulując działanie bramki XOR.

Reprezentacja dekodowania dla wprowadzonego wektora **[1, 1, 1, 0]**. Po zakodowaniu otrzymaliśmy wektor **[1, 0, 0, 1]**, natomiast po operacji wymazywania **[1, 0, 0 x3]**

```
Oczekiwany zdekodowany wektor: [1, 1, 1, 0]
Zdekodowany wektor (przed operacją mod 2): [[1], [1], [1], [x3 + 1]]
Zdekodowany wektor: [1, 1, 1, 0]
```

7. MOŻLIWOŚCI ROZWOJU

- wprowadzenie większej ilości przypadków testowych wartości wprowadzanych przez użytkownika,
- możliwość wprowadzenia dowolnej długość wektora wejściowego, w tym generowanie macierzy Kroneckera o odpowiedniej wielkości,
- wprowadzenie różnych rodzajów kodowania, takich jak:
    - zamrażanie bitów
    - mapowanie bitów informacji
    - iteracyjne kodowanie
- wprowadzenie różnych rodzajów wymazywania, takich jak:
    - stałe – możliwość wyboru, co który bit ma być wymazywany
    - adaptacyjne
    - w oparciu o polaryzację
    - z priorytetami
    - losowe,
- zaawansowane formy dekodowania nie zakładające wiadomości o obecności 0 lub 1, oraz:
    - metoda SC
    - metoda List Deocding
    - metoda Belief Propagation
- GUI - graficzny interfejs użytkownika umożliwiający wybranie odpowiednich opcji,
- rozdzielenie kodu na kilka plików *.py* w celu ułatwienia analizy kodu.

8. BIBLIOGRAFIA

Wykonanie projektu nie byłoby możliwe bez skorzystania z dodatkowych źródeł. Poniżej znajduje się lista z pomocnymi wykładami i literaturą, które towarzyszyły przy realizacji projektu.

- Ćwiczenia z przedmiotu “Teoria Informacji i Kodowania”
  dr inż. Krzysztof Cichoń
- Wykłady “Teoria Informacji i Kodowania”
  prof. dr hab. inż. Hanna Bogucka
- Chat GPT
  zrozumienie działania kodera/dekodera, poszczególna pomoc w pisaniu kodu, tworzenie raportu.

9. KOD W JĘZYKU PYTHON

```
import random
import numpy as np
from sympy import Matrix, Symbol, Mod


### WPROWADZANIE DANYCH PRZEZ UŻYTKOWNIKA --> n^2, u  ###

# Wprowadzenie długości wektora wejściowego, konwersja na int,
testowanie wprowadzonej wartości
while True:
    n2 = int(input("Podaj długość wektora wejściowego n^2 (4, 8, 16,
32): "))
    if n2 in [4, 8, 16, 32]:
      break
    print("Nieprawidłowa długość. Spróbuj ponownie.")

decision = input("Czy chcesz wprowadzić wektor? (TAK - wprowadź, NIE -
wygeneruj losowy): ").lower()

if decision == "tak":
    while True:
      # Dzielenie ciągu na części (spacja) i tworzenie listy elementów
      u = input(f"Wprowadź wektor wejściowy o długości {n2} (format: 1
0 1 1): ").split()
      if len(u) == n2 and all(bit in ['0', '1'] for bit in u):
        # Konwersja każdego elementu listy z łańcucha znaków na liczbę
całkowitą
        u = [int(bit) for bit in u]
        print(f"\nWprowadzony wektor wejściowy u: {u}")
        break
      print(f"Nieprawidłowe dane, wektor powinien zawierać {n2} bitów i
składać się z 0 i 1. Spróbuj ponownie.")
else:
    u = [random.randint(0, 1) for i in range(n2)]
    print(f"\nWygenerowany losowy wektor wejściowy o długości {n2}:
{u}")


### MACIERZE KRONECKERA ###

# Macierz Arikana
F2 = np.array([[1, 0], [1, 1]])
```

```
# Obliczanie produktu Kroneckera dwóch macierzy
def kronecker_product(A, B):
    return np.kron(A, B)

# Generowanie macierzy Kroneckera
def generate_kronecker_matrices(F2):
    F4 = kronecker_product(F2, F2)
    F8 = kronecker_product(F4, F2)
    F16 = kronecker_product(F8, F2)
    F32 = kronecker_product(F16, F2)
    return F4, F8, F16, F32

# Generowanie macierzy F2, F4, F8, F16, F32
F4, F8, F16, F32 = generate_kronecker_matrices(F2)

kronecker_matrices = {
    4: F4,
    8: F8,
    16: F16,
    32: F32,
}

print(f"Wykorzystywana macierz Kroneckera
F{n2}:\n{kronecker_matrices[n2]}")
#print("F4:\n", F4) #F4, F8, F16, F32


### KODOWANIE ###

# Mnożenie macierzy
u_coded = kronecker_matrices[n2] @ u

# Wykoannie operacji mod 2
for i in range(0, len(u_coded)):
    u_coded[i] = u_coded[i]%2

print(f"\nZakodowany wektor u: {u_coded}")


### WYMAZYWANIE ###

u_coded_erased = []
```

```
# Wymazywanie stałe
for i in range(0, len(u_coded)):
    if i % 2 == 0:
        u_coded_erased.append(int(u_coded[i]))
    # Wymazujemy co drugi bit, ale mamy pewność co do występowanego 0
    else:
        if u_coded[i] == 0:
            u_coded_erased.append(0)
        else:
            u_coded_erased.append(f"x{i}")

print(f"Odebrany wektor: {u_coded_erased}")


### DEKODOWANIE ###

inverse_kronecker_matrix = np.linalg.inv(kronecker_matrices[n2])
inverse_kronecker_matrix = inverse_kronecker_matrix.astype(int)
print(f"\nOdwrotna macierz Kroneckera
F{n2}:\n{inverse_kronecker_matrix}")

inverse_kronecker_matrix = (inverse_kronecker_matrix % 2 + 2) % 2
print(f"Wykorzystywana odwrotna macierz Kroneckera (F{n2} mod
2):\n{inverse_kronecker_matrix}")

# Zamiana odebranego wektora na macierz symboli
u_coded_erased_matrix = Matrix(u_coded_erased)

u_decoded = inverse_kronecker_matrix @ u_coded_erased_matrix

# Czytelne wypisywanie zdekodowanego wektora
u_decoded_str = str(u_decoded).replace('Matrix(', '').replace(')', '')
print(f"\nOczekiwany zdekodowany wektor: {u}")
print(f"Zdekodowany wektor (przed operacją mod 2): {u_decoded_str}")

# Zakładamy, że wymazany symbol wynosił 1
u_decoded = u_decoded.applyfunc(
    lambda x: x.subs({s: 1 for s in x.free_symbols}) if x.has(Symbol)
else x
)

u_decoded = u_decoded.applyfunc(lambda x: Mod(x, 2))
```

```
u_decoded_list = [u_decoded[i, 0] for i in range(u_decoded.shape[0])]
print(f"Zdekodowany wektor: {u_decoded_list}")


### PODSUMOWANIE KODERA I DEKODERA ###

print(f"\nPODSUMOWANIE:"
      f"\nWektor wejściowy: {u}"
      f"\nZakodowany wektor: {u_coded}"
      f"\nOdebrany wektor: {u_coded_erased}"
      f"\nOczekiwany zdekodowany wektor: {u}"
      f"\nZdekodowany wektor: {u_decoded_list}")
```