

International University of Information Technology

Department of Computer Engineering

**Laborotoy Work №11**

Complete by a student of the group: Ospan Ramazan it2-2310

Checked by senior lector of the Department of Computer Engineering:

Daurenbaeva Nurkamilya

Almaty 2024

During the completion of this lab work on block coding, I focused on understanding the principles behind linear block codes and their application in ensuring reliable data transmission. Here's how I interpreted and approached the task:

**Block Coding Basics**

Block coding is a technique that adds redundancy to a message, enabling the detection and correction of errors during data transmission. The process uses two critical matrices:

* **Generator Matrix (*GG*G)**: Responsible for encoding the original message into a longer codeword by adding parity bits.
* **Parity-Check Matrix (*HH*H)**: Used to verify the integrity of the codeword at the receiver’s side and identify any errors.

The redundancy introduced by block coding is crucial for ensuring data reliability, especially in noisy channels.

**Key Steps in the Lab Work**

1. **Encoding the Message**

I used the generator matrix *GG*G to encode a 4-bit message into a 7-bit codeword. The generator matrix combines the information bits with parity bits calculated based on linear algebra. The encoded codeword contains both the original data and the additional bits for error checking.

1. **Simulating Transmission Errors**

To simulate real-world conditions, I introduced a deliberate error into the transmitted codeword by flipping one bit. This allowed me to test how the parity-check matrix *HH*H could detect and correct the error.

1. **Error Detection Using the Parity-Check Matrix**

The received codeword was multiplied by the transpose of *HH*H to calculate the **syndrome**. This step revealed whether an error occurred and pointed to the bit's exact position. The concept of the syndrome felt intuitive because it directly linked the error to a specific location.

1. **Correcting the Error**

Using the syndrome, I identified the incorrect bit in the received codeword and flipped it back to its original value. This step successfully restored the original, error-free codeword.

**Practical Example from the Lab**

* **Message (input):** [1, 0, 1, 1]
* **Encoded Codeword:** [1, 0, 1, 1, 1, 0, 1]
* **Received Codeword (with an error):** [1, 1, 1, 1, 1, 0, 1] (the second bit was flipped).
* **Syndrome Calculation:** Pointed to the second bit as erroneous.
* **Corrected Codeword:** [1, 0, 1, 1, 1, 0, 1] (matched the original encoded codeword).

**What I Learned**

This lab gave me hands-on experience with how block coding works in practice. I now understand the following:

1. **The Role of *GG*G and *HH*H:**
   1. *GG*G ensures the message is properly encoded with redundancy.
   2. *HH*H detects and helps correct errors using the syndrome.
2. **Error Handling:**
   1. Even a simple linear block code like the (7,4) Hamming code can detect and correct single-bit errors effectively.
3. **Applications in Real Life:**
   1. Block coding techniques are widely used in Wi-Fi, mobile networks, and storage systems to ensure reliable data transmission.

Algorithm:

**i**mport numpy as np

# Создание матрицы генератора (G) для кода (7,4)

def create\_generator\_matrix():

I = np.identity(4, dtype=int) # Единичная матрица 4x4

P = np.array([[1, 0, 1], # Матрица P (проверочные биты)

[1, 1, 0],

[0, 1, 1],

[1, 1, 1]])

G = np.hstack((I, P)) # Генераторная матрица (объединение I и P)

return G

# Создание проверочной матрицы (H) для кода (7,4)

def create\_parity\_check\_matrix():

P = np.array([[1, 0, 1], # Матрица P

[1, 1, 0],

[0, 1, 1],

[1, 1, 1]])

H = np.hstack((P.T, np.identity(3, dtype=int))) # Объединение P^T и I

return H

# Кодирование сообщения с помощью G

def encode\_message(message, G):

message\_vector = np.array(message, dtype=int)

codeword = np.mod(np.dot(message\_vector, G), 2) # Умножение и mod 2

return codeword

# Обнаружение ошибок с помощью H

def detect\_error(received, H):

syndrome = np.mod(np.dot(received, H.T), 2) # Умножение на H^T и mod 2

return syndrome

# Исправление ошибок

def correct\_error(received, syndrome, H):

for i in range(H.shape[1]): # Поиск позиции ошибки

if np.array\_equal(syndrome, H[:, i]):

received[i] = 1 - received[i] # Инвертировать бит

break

return received

# Пример использования

G = create\_generator\_matrix()

H = create\_parity\_check\_matrix()

# Кодирование сообщения

message = [1, 0, 1, 1] # Пример информационного вектора

codeword = encode\_message(message, G)

print("Исходное кодовое слово:", codeword)

# Симуляция ошибки

received = codeword.copy()

received[2] = 1 - received[2] # Инверсия одного бита

print("Принятое кодовое слово (с ошибкой):", received)

# Обнаружение ошибки

syndrome = detect\_error(received, H)

print("Синдром ошибки:", syndrome)

# Исправление ошибки

if np.any(syndrome):

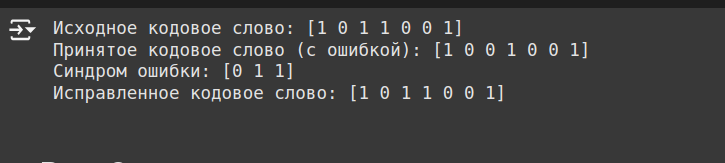
corrected = correct\_error(received, syndrome, H)

print("Исправленное кодовое слово:", corrected)

else:

print("Ошибок нет")

**Test:**



**Final Thoughts**

Overall, this lab demonstrated the practical importance of adding redundancy to data to combat real-world noise and interference. By using simple mathematical tools like matrix multiplication and modulo-2 arithmetic, I could see how coding theory is applied to maintain data integrity.