HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

**SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING**

**PROJECT REPORT**

**HAMMING CODES**

Name of students: Phan Trinh Dong Phuong -

Bui Xuan Trung

Nguyen Quang Thang

Ho Tuan Tu

Course: Information Theory – ET2072E

Class: 144311

Name of instructor: Ph.D. Assoc. Prof. Nguyen Huu Phat

Name of referee: ……………………………………

Hanoi, 01-2024

1. **Introduction**

In telecommunication, Hamming codes are a family of linear error-correcting codes. Hamming codes can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors. By contrast, the simple parity code cannot correct errors, and can detect only an odd number of bits in error. Hamming codes are perfect codes, that is, they achieve the highest possible rate for codes with their block length and minimum distance of three.

The implemented code consists of:

* A function responsible for encoding the message (emitterConverter)

\* return the encoded message

* A function responsible for decoding the message (receptorConverter)

\* return the decoded message and a ack of data integrity

1. **The way how to use this code**

* The variable of the message/word that must be desired to be encoded (text).
* To be used we must declare how many parity bits (sizePari) you want to include in the message.
* It is desired (for test purposes) to select a bit to be set as an error. This serves to check whether the code is working correctly.

For example,

binaryText = get\_binary\_input()

# number of parity bits

sizePari = get\_integer\_input("Enter the size of parity bit: ")

# location of the bit that will be forced an error

be = get\_integer\_input("Enter the location of the bit that will be forced an error: ")

# Prints the binary of the string

print("Input in binary is '" + binaryText + "'")

# total transmitted bits

totalBits = len(binaryText) + sizePari

print("Size of data is " + str(totalBits))

print("\n --Message exchange--")

print("Data to send ------------> " + binaryText)

dataOut = emitter\_converter(sizePari, binaryText)

print("Data converted ----------> " + "".join(dataOut))

dataReceiv, ack = receptor\_converter(sizePari, dataOut)

print(

    "Data receive ------------> "

    + "".join(dataReceiv)

    + "\t\t -- Data integrity: "

    + str(ack)

)

print("\n --Force error--")

print("Data to send ------------> " + binaryText)

dataOut = emitter\_converter(sizePari, binaryText)

print("Data converted ----------> " + "".join(dataOut))

# forces error

dataOut[-be] = "1" \* (dataOut[-be] == "0") + "0" \* (dataOut[-be] == "1")

print("Data after transmission -> " + "".join(dataOut))

dataReceiv, ack = receptor\_converter(sizePari, dataOut)

print(

    "Data receive ------------> "

    + "".join(dataReceiv)

    + "\t\t -- Data integrity: "

    + str(ack)

)

After we enter three variables into the code, the program will run it automatically and show the user 4 kinds of detail outputs:

* The input in binary
* The size of the data in the transmission
* Message exchange, that has Data to send (Text input in binary), Data converted (Data that are encoded in the process of transmitting) and Data receive (Data that are decoded in the process)
* Force Error contains Data to send (Text input in binary), Data Converted (Data that are encoded in the process of transmitting), Data after transmission (Data are made mistake through transmission) and Data receive (Data will be fixed at the decoder)

***Result for the example above:***

A computer screen shot of a black screen

Description automatically generated

* If the users enter the input that is not satisfied with the condition, the program will ask them to re-enter the input until it is corrected.

1. **Operation of the code**
2. Declaration of variables (sizePari, be, text)

A screen shot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

1. Encodes the message using the rules of hamming encoding.

***Check the size of the parity bit, data and initialize variables:***

A computer screen shot of a program code

Description automatically generated

* With the condition size\_par + length(data) ≤ 2size\_par – (length(data) −1), the size of parity doesn’t match with the size of the data. So, the data will not be encoded in the transmission.
* We utilize the example bits = 00111001 with the size of the parity bit = 4 and the location of the bit that will be forced an error = 3.
* bin\_pos: Array containing binary string of numbers from 1 to size\_par + length(data)

bin\_pos = [0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100]

***Create Template for Data Location and Check Bits by specifying the position of the check bit based on the bit number***A computer screen with text and images

Description automatically generated

For x variables from 1 to 12:

* data\_out\_gab = [P, P, D, P, D, D, D, P, D, D, D, D]
* In the array data\_out\_gab, at position P, it is called “Parity bit” and at position D, it is called “Data bit”.
* data\_ord = [None, None, “0”, None, “0”, “1”, “1”, None, “1”, “0”, “0”, “1”]
* cont\_data = 8
* qtd\_bp = 4

A computer screen shot of a program

Description automatically generated

***#Calculate the parity bit***

Inside the loop bp from 1 to 4, there is another loop with variable x running in the data\_ord.

* With bp = 1, the cont\_bo = 2 → the first parity is 2%2 = 0 → qtd\_bp =1
* With bp = 2, the cont\_bo = 2 → the second parity is 2%2 = 0 → qtd\_bp =2
* With bp = 3, the cont\_bo = 3 → the third parity is 3%2 = 1 → qtd\_bp =3
* With bp = 4, the cont\_bo = 4 → the third parity is 4%2 = 0 → qtd\_bp =4

→ parity = [0, 0, 1, 0]

***#Mount the message:***

The parity bit [0, 0, 1, 0] will match the the data\_ord = [None, None, “0”, None, “0”, “1”, “1”, None, “1”, “0”, “0”, “1”] at the position of element None.

→ Data Converted = 000101101001

1. Decodes the message using the rules of hamming encoding.

A computer screen shot of a program

Description automatically generated

Initialize variables and Create a template for the positions of data bits and parity bits by specifying the position of the check bit based on the bit order.

A screenshot of a computer program

Description automatically generated

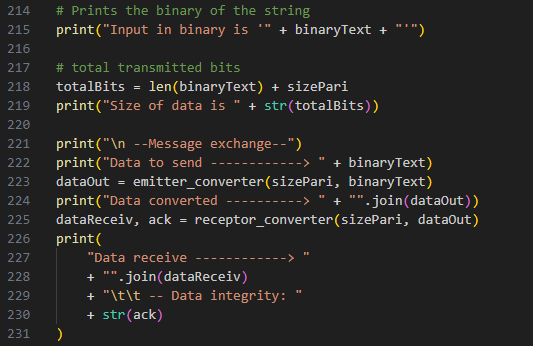
Regenerate a template for the positions of data bits and parity bits after decoding based on the bit order.

A screen shot of a computer program

Description automatically generated

* Recalculate parity bit value based on the decoded data bit.
* Use the recalculated parity bit value to produce the final decoded message.
* Returns both the decoded message and a flag (ack) indicating the correctness of the decoded data. This flag will be true if the parity bit received is the same as the parity bit recalculated from the decoded data.

1. Print the original input, the encoded input, and the decoded input.



1. Forces an error in the coded binary number. Decodes the binary number that was forced the error. Print the original message, the encoded message, the bit changed message and the decoded message.

A computer screen shot of a program code

Description automatically generated