TRƯỜNG ĐIỆN - ĐIỆN TỬ

BÀI TẬP LỚN LÝ THUYẾT THÔNG TIN

Mã hóa văn bản theo mã Huffman
Mã Hamming cho truyền tín hiệu âm thanh

Giảng viên hướng dẫn : TS. Nguyễn Hữu Phát

Sinh viên	*MSSV*
Phạm Quang Sáng	20193076
Nguyễn Khắc Thành	20193116

Tháng 1, năm 2021

Mục lục

Danh sách hình vẽ

1 Giới thiệu

Trong bài tập lớn này, chúng em ứng dụng 2 thuật toán là mã Huffman sử dụng cho việc mã hoá một đoạn văn bản và mã Hamming cho việc mã hoá tín hiệu âm thanh.
Chúng em lựa chọn ngôn ngữ lập trình Python3 để xây dựng phần mềm vì tính đơn giản và có sẵn thư viện hỗ trợ trong tính toán và xử lý các file âm thanh. Thư viện chúng em sử dụng là scipy và matplotlib. Toàn bộ mã nguồn và hướng dẫn sử dụng được gửi trong file đính kèm và trên trang github cá nhân https://github.com/saltmurai/Information_Theory_Project

2 Cơ sở lý thuyết

2.1 Mã Huffman

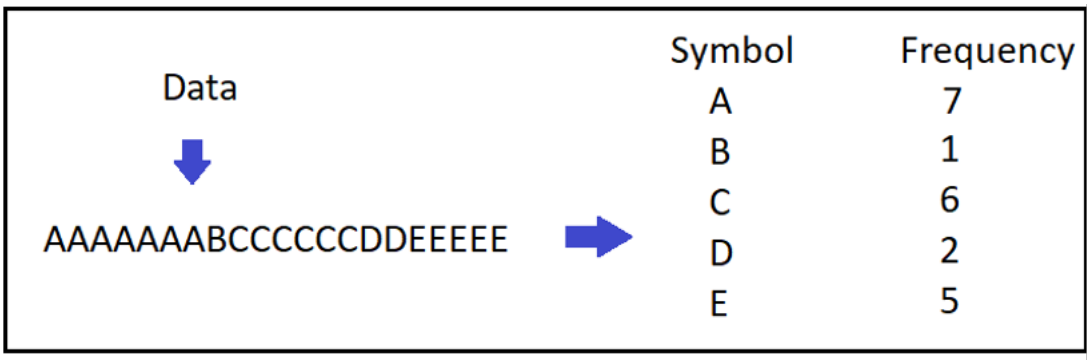
Mã hóa Huffman là một thuật toán sử dụng các tính năng tần số (hoặc xác suất) của các ký hiệu và cấu trúc cây nhị phân. Nó bao gồm 3 bước sau:

1. Tính toán xác suất và sắp xếp các ký hiệu
2. Chuyển đổi cây nhị phân
3. Gán mã cho các biểu tượng

Bước 1. Tính toán xác suất và sắp xếp các ký hiệu

Đếm số lượng của mỗi ký hiệu trong toàn bộ dữ liệu, sau đó tính "xác suất" của mỗi ký hiệu bằng cách chia số lượng đó cho tổng số ký tự trong dữ liệu. Vì đó là một thuật toán sử dụng xác suất, các ký hiệu phổ biến hơn - các ký hiệu có xác suất cao hơn - thường được biểu diễn bằng cách sử dụng ít bit hơn các ký hiệu ít phổ biến hơn. Đây là một trong những mặt thuận lợi của Mã hóa Huffman.

Ví dụ, đối với dữ liệu sau có 5 ký hiệu khác nhau là A B C D E, có các xác suất như hình 1.



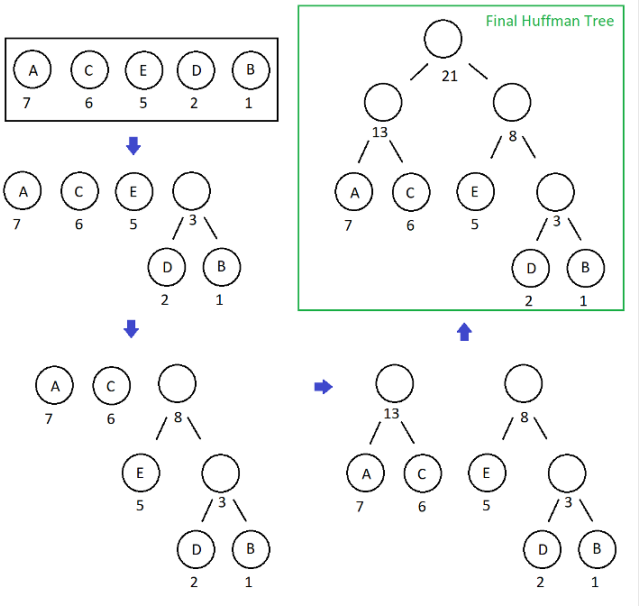
Hình 1: Tính toán sác xuất và sắp xếp các ký hiệu

Sau đó, dễ dàng sắp xếp các ký hiệu theo xác suất của chúng đại diện cho mỗi ký hiệu là một nút và gọi đó là "bộ từ điển".

Bước 2. Chuyển đổi cây nhị phân

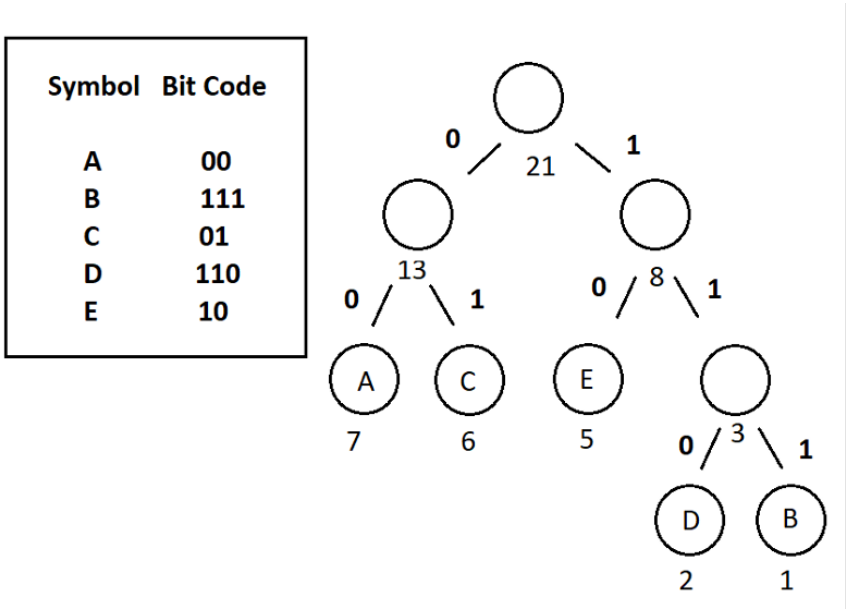
Từ từ điển, chọn ra hai nút có tổng xác suất nhỏ nhất và kết hợp chúng thành một nút mới có gốc có xác suất bằng tổng đó. Thêm nút mới vào bộ từ điển. Lặp lại quá trình này cho đến khi một nút bao gồm tất cả các xác suất đầu vào đã được xây dựng

Bước 3. Gán mã cho các biểu tượng



Hình 2: Chuyển đổi cây nhị phân

Sau khi có được cây nhị phân này – Cây Huffman, gán 1 cho mỗi lần đi qua nút phải và 0 cho mỗi lần đi qua nút trái. Cuối cùng, các ký hiệu và mã của chúng theo Huffman.



Hình 3: Gán mã cho các biểu tượng

2.1.1 Mã hoá kênh

Khi truyền tin qua một kênh bất kỳ luôn có khả năng có lỗi xảy ra. Đối với mã nhị phân điều này xảy ra thường xuyên khi truyền giữa các kênh. Bit $0 \rightarrow 1$ và

ngược lại. Việc mã hoá kênh với mã sửa lỗi giúp giảm thiểu số lỗi và sửa lỗi sinh ra bởi kênh truyền.

2.1.2 Mã Hamming(7-4)

Mã Hamming(7-4) là mã sửa lỗi tuyến tính, dữ liệu 4 bit được mã hoá thành 7 bits bằng việc thêm 3 bits parity. Mã Hamming(7-4) có thể sửa lỗi nếu có 1 lỗi trong quá trình truyền và có thể phát hiện mã nhận được có 1 hay 2 lỗi.Mã Hamming(7,4) có hiệu quả phát hiện lỗi và sửa lỗi cao đối với kênh truyền không có lỗi chùm xuất hiện (burst error).
Với dữ liệu 4 bit là $d_3\ d_2\ d_1\ d_0$ 3 bit $p_1\ p_2\ p_3$ được thêm vào tạo thành mã Hamming 7 bit như sau:

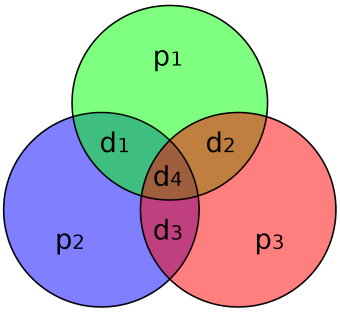
1	2	3	4	5	6	7
p_1	p_2	d_1	p_3	d_2	d_3	d_4

Trong đó p_1 kiểm tra d_1, d_2, d_4, p_2 kiểm tra d_1, d_3, d_4, p_3 kiểm tra d_2, d_3, d_4. Trước khi được truyền đi p_1, p_2 và p_3 được tính theo dựa vào các bit mà chúng kiểm tra theo công thức:

$$p_1 = d_1 \oplus d_2 \oplus d_4 \tag{1}$$

$$p_2 = d_1 \oplus d_3 \oplus d_4 \tag{2}$$

$$p_3 = d_2 \oplus d_3 \oplus d_4 \tag{3}$$



Hình 4: Mối liên hệ giữa bit kiểm tra và bit dữ liệu [1].

Mã Hamming sau khi qua kênh truyền có thể gặp hiện tượng lỗi lúc này để phát hiện vị trí bit sai ta tính các bit kiểm tra c_1, c_2 và c_3. $c_1c_2c_3$ sẽ cho ta biết mã nhận được có sai hay không và vị trí của lỗi sai. Ví dụ $c_1c_2c_3 = 010$ tức là mã nhận được sai ở vị trí số 2, để sửa lỗi ta chỉ cần lật lại bit số 2. Nếu $c_1c_2c_3 = 000$ tức là không có lỗi(đối với kênh truyền chỉ xảy ra 1 lỗi). c_1, c_2 và c_3 được tính như sau:

$$c_1 = p_1 \oplus d_1 \oplus d_2 \oplus d_4 \tag{4}$$

$$c_2 = p_2 \oplus d_1 \oplus d_3 \oplus d_4 \tag{5}$$

$$c_3 = p_3 \oplus d_2 \oplus d_3 \oplus d_4 \tag{6}$$

Đây là một cách phổ biến để sử dụng mã Hamming(7,4) trong phần mềm, ngoài cách này ra còn có thể sử dụng ma trận sinh, ma trận kiểm tra. Trong bài tập lớn này chúng em sẽ sử dụng phương pháp này để mã hoá cho âm thanh 8-bit.

3 Ứng dụng

3.1 Mã hoá văn bản theo mã Huffman

3.1.1 Code Python

Để thực hiện Mã hóa Huffman, bắt đầu với một lớp Node, tham chiếu đến các nút của Cây Huffman. Về bản chất, mỗi nút có một ký hiệu và biến xác suất liên quan, nút trái và phải, và mã hoá. Mã hoá sẽ là 0 hoặc 1 khi đi qua Cây Huffman theo phía đã chọn (trái 0, phải 1)

```
class Node:
    def __init__(self, prob, symbol, left=None, right=None):
        # Xac suat cua ki tu
        self.prob = prob
        # Ky tu
        self.symbol = symbol
        # Nut trai
        self.left = left
        # Nut phai
        self.right = right
        # Huong cua cay (0|1)
        self.code = ""
```

Cần có 3 hàm trợ giúp, hàm thứ nhất để tính xác suất của các ký tự trong dữ liệu đã cho, hàm thứ hai để lấy mã hóa các ký tự sẽ được sử dụng sau khi có Cây Huffman và hàm cuối cùng để lấy đầu ra (dữ liệu được mã hóa).

```
codes = dict()

def Calculate_Codes(node, val=""):
    """Ham in ma ky tu khi di qua Huffman Tree"""
    # Ma Huffman cho nut hien tai
    newVal = val + str(node.code)

    if node.left:
        Calculate_Codes(node.left, newVal)
    if node.right:
        Calculate_Codes(node.right, newVal)
    if not node.left and not node.right:
        codes[node.symbol] = newVal

    return codes

def Calculate_Probability(data):
    """Ham tinh toan xac suat cua cac ky tu trong du lieu nhat dinh
                                        """
    symbols = dict()
    for element in data:
        if symbols.get(element) == None:
            symbols[element] = 1
        else:
            symbols[element] += 1
    for element in symbols:
        symbols[element] = symbols[element] / len(data)
```

```
    return symbols

def Output_Encoded(data, coding):
    """Ham tro giup de lay dau ra duoc ma hoa"""
    encoding_output = []
    for c in data:
        encoding_output.append(coding[c])

    string = "".join([str(item) for item in encoding_output])
    return string
```

Cuối cùng, hàm Huffman_Encoing thực hiện mã Huffman, và in ra dữ liệu là thông số như Entropy, độ dài trung bình, hệ số nén của đầu vào và trả về bộ mã được mã hoá.

```
def Huffman_Encoding(data):
    symbol_with_probs = Calculate_Probability(data)
    symbols = symbol_with_probs.keys()
    probabilities = symbol_with_probs.values()
    etrpy = 0
    print("Ky tu: ", symbols)
    print("Xac suat: ", probabilities)
    # Tinh Entropy cua van ban
    for symbol in symbols:
        etrpy = etrpy - symbol_with_probs[symbol] * math.log2(
                                            symbol_with_probs[symbol]
                                            )
    print("Entropy: ", etrpy)

    nodes = []

    # Chuyen doi cac ky hieu va tan so thanh cac nut cay Huffman
    for symbol in symbols:
        nodes.append(Node(symbol_with_probs.get(symbol), symbol))

    while len(nodes) > 1:
        # Sap xep tat ca cac nut theo thu tu tang dan dua tren tan
                                            so
        nodes = sorted(nodes, key=lambda x: x.prob)

        # Chon 2 nut nho nhat
        right = nodes[0]
        left = nodes[1]

        left.code = 0
        right.code = 1

        # Ket hop 2 nut nho nhat de tao nut moi
        newNode = Node(left.prob + right.prob, left.symbol + right.
                                            symbol, left, right)

        nodes.remove(left)
        nodes.remove(right)
        nodes.append(newNode)
```

```
    huffman_encoding = Calculate_Codes(nodes[0])
    print("Ky tu voi ma: ", huffman_encoding)
    # Tinh do dai trung binh cua van ban
    l = 0
    for symbol in symbols:
        L = len(huffman_encoding[symbol])
        l = l + symbol_with_probs[symbol] * L
    print("do dai trung binh cua van ban: ", l)
    print("He so nen: {}/{} = {}".format(etrpy, l, etrpy / l))
    encoded_output = Output_Encoded(data, huffman_encoding)
    return encoded_output, nodes[0]
```

3.1.2 Kết quả và đánh giá

Thử nghiệm chương trình với đoạn văn bản "AAAAAAABCCCCCCDDEEEEE" được kết quả như sau.

```
AAAAAAABCCCCCCDDEEEEE
Ky tu: dict_keys(['A', 'B', 'C', 'D', 'E'])
Xac suat: dict_values([0.3333333333333333, 0.047619047619047616,
0.2857142857142857, 0.09523809523809523, 0.23809523809523808])
Entropy: 2.0698936164374757
Ky tu voi ma: 'A': '00', 'C': '01', 'E': '10', 'D': '110', 'B':
'111'
Do dai trung binh cua van ban: 2.142857142857143
He so nen: 2.0698936164374757/2.142857142857143 = 0.9659503543374887
Dau ra duoc ma hoa: 00000000000000111010101010101111011010101010
Dau ra duoc giai ma: AAAAAAABCCCCCCDDEEEEE
```

Với đoạn văn bản dài hơn như sau.

```
"Lorem ipsum dolor sit amet, consectetur adipiscing elit. In-
teger sollicitudin magna vel ligula cursus, in lobortis risus
rutrum. Curabitur ut sollicitudin velit, eu gravida odio. Fusce
eu urna ac erat sollicitudin egestas. Etiam blandit sagittis
nisl, vel maximus nunc dapibus at. Nunc bibendum odio venenatis
lacus porttitor, vestibulum condimentum metus convallis. Orci
varius natoque penatibus et magnis dis parturient montes, nasce-
tur ridiculus mus. Proin vel tempor odio. Praesent eu mauris
id dui vestibulum pretium dignissim ac purus.
Nam pharetra molestie sollicitudin. Etiam ante ex, blandit a
elit sed, efficitur volutpat magna. Nunc semper, lacus quis eges-
tas fringilla, mi ligula rhoncus eros, id sodales augue nisi
sed felis. Phasellus viverra viverra vehicula. Nulla vel lobor-
tis massa. Cras non orci nibh. Phasellus sit amet ante sagit-
tis, dignissim massa ut, interdum nibh. Vestibulum vel scelerisque
felis."
```

Kết qủa thu được:

```
Ky tu: dict_keys(['L', 'o', 'r', 'e', 'm', ' ', 'i', 'p', 's',
'u', 'd', 'l', 't', 'a', ',', 'c', 'n', 'g', '.', 'I', 'v', 'b',
'C', 'F', 'E', 'x', 'N', 'O', 'q', 'P', '', 'h', 'f', 'V'])
Xac suat: dict_values([0.0010834236186348862,
0.035752979414951244, 0.047670639219934995, 0.07258938244853738,
0.035752979414951244, 0.1419284940411701, 0.09750812567713976,
0.013001083423618635, 0.0790899241603467, 0.06825568797399784,
0.028169014084507043, 0.05308775731310943, 0.06392199349945829,
0.06500541711809317, 0.014084507042253521, 0.028169014084507043,
0.047670639219934995, 0.01733477789815818, 0.018418201516793065,
0.0010834236186348862, 0.01950162513542795, 0.015167930660888408,
0.0021668472372697724, 0.0010834236186348862, 0.0021668472372697724,
0.0021668472372697724, 0.004333694474539545, 0.0010834236186348862,
0.0032502708559046588, 0.004333694474539545, 0.0010834236186348862,
0.007583965330444204, 0.005417118093174431, 0.0010834236186348862])
Entropy: 4.231680034510799
Ky tu voi ma: 's': '0000', 'N': '000100000', 'q': '000100001',
'f': '00010001', 'V': '0001001000', '': '0001001001', 'O': '0001001010',
'F': '0001001011', 'I': '0001001100', 'L': '0001001101', 'x':
'000100111', 'v': '000101', '.': '000110', 'g': '000111', 'e':
'0010', 'm': '00110', 'o': '00111', ' ': '010', 'u': '0110',
'a': '0111', 't': '1000', 'E': '100100000', 'C': '100100001',
'P': '10010001', 'h': '1001001', 'b': '100101', 'c': '10011',
'd': '10100', ',': '101010', 'p': '101011', 'l': '1011', 'i':
'110', 'n': '1110', 'r': '1111'
Do dai trung binh cua van ban: 4.254604550379198
He so nen: 4.231680034510799/4.254604550379198 = 0.9946118339326377
```

3.2 Mã Hamming cho truyền tín hiệu âm thanh

3.2.1 Code Python

Hàm hamming_74 lấy đầu vào là dự liệu 4 bit và thực hiện tính toán các parity bit theo (1) (2) (3). Hàm trả về mã Hamming trước khi qua kênh truyền

```
def hamming_74(codeWord):
    """
    This function take 4 bit string in binary format and encode it
                                        following
    hamming(7,4) algorithm
    1   2   3   4   5   6   7
    p1  p2  d4  p3  d3  d2  d1
    """
```

```
    if len(codeWord) != 4:
        print("Error, must be a 4 bit value")
        return -1
    x = [int(x) for x in list(codeWord)]
    p1 = x[0] ^ x[1] ^ x[3]
    p2 = x[0] ^ x[2] ^ x[3]
    p3 = x[1] ^ x[2] ^ x[3]
    hamming = [p1, p2, x[0], p3, x[1], x[2], x[3]]
    codedWord = [str(bit) for bit in hamming]
    codedWord = "".join(codedWord)
    return codedWord
```

Hàm decodeHamming7_4 lấy đầu vào là mã nhận, tính toán các bit kiểm tra c1, c2, c3 và sửa sai nếu phát hiện lỗi. Hàm trả về mã Hamming ban đầu

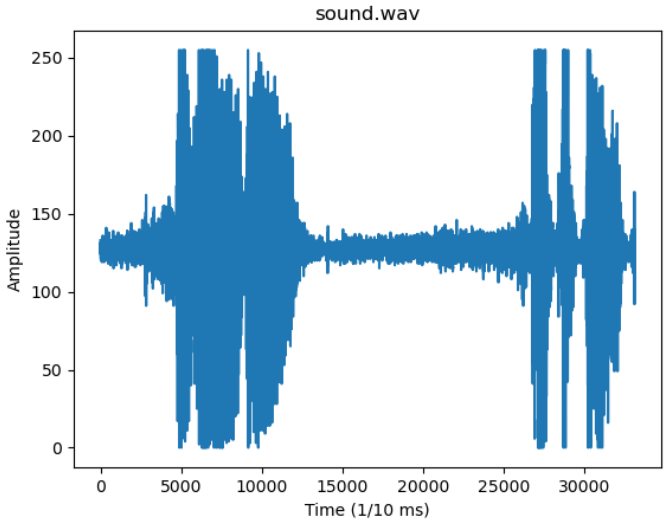
```
def decodeHamming7_4(hammingCode):
    """
    This function take 7 bits Hamming Code then
    return 4 bits original code word
    Example: 1100110 -> 0110
    """
    if len(hammingCode) != 7:
        print("Error: Must be a 7 bit value")
        return -1
    hammingCode_int = [int(bit) for bit in hammingCode]
    # Getting the parity bits and data bits
    p1 = hammingCode_int[0]
    p2 = hammingCode_int[1]
    p3 = hammingCode_int[3]
    d4 = hammingCode_int[2]
    d3 = hammingCode_int[4]
    d2 = hammingCode_int[5]
    d1 = hammingCode_int[6]
    # Compute the check bits
    c1 = p1 ^ d4 ^ d3 ^ d1
    c2 = p2 ^ d4 ^ d2 ^ d1
    c3 = p3 ^ d3 ^ d2 ^ d1
    # Compute the position of 1 error bit
    pos = c1 * 1 + c2 * 2 + c3 * 4
    # If error flip the error bit
    if pos:
        if hammingCode_int[pos - 1] == 0:
            hammingCode_int[pos - 1] = 1
        else:
            hammingCode_int[pos - 1] = 0
    decoded = (
        str(hammingCode_int[2])
        + str(hammingCode_int[4])
        + str(hammingCode_int[5])
        + str(hammingCode_int[6])
    )
    return decoded
```

Tạo một class giúp mô phỏng sự nhiễu của kênh truyền

```
class Channel:
    """
```

```
    msg: list of hamming code
    prob: Probablity for a bit in the message to be flip
    """
    def __init__(self, msg, prob):
        self.message = msg
        self.corruptedMessage = []
        for word in self.message:
            corruptedWord = ""
            for bit in word:
                error = False
                rand = random.randint(0, 100)
                # Simulation of a random error
                if rand < prob:
                    error = True
                # depending on the error factor, bits are corrupted
                                                    or not
                if bit == "0":
                    if error:
                        corruptedWord += "1"
                    else:
                        corruptedWord += "0"

                elif bit == "1":
                    if error:
                        corruptedWord += "0"
                    else:
                        corruptedWord += "1"

            self.corruptedMessage.append(corruptedWord)

    def getMessage(self):
        return self.message

    def getCorruptedMessage(self):
        return self.corruptedMessage
```

Âm thanh 8 bit có biên độ tín hiệu tối đa là 255, sử dụng thư viện scipy và matplotlib vẽ được đồ thị của âm thanh như Hình 5.



Hình 5: Âm thanh gốc

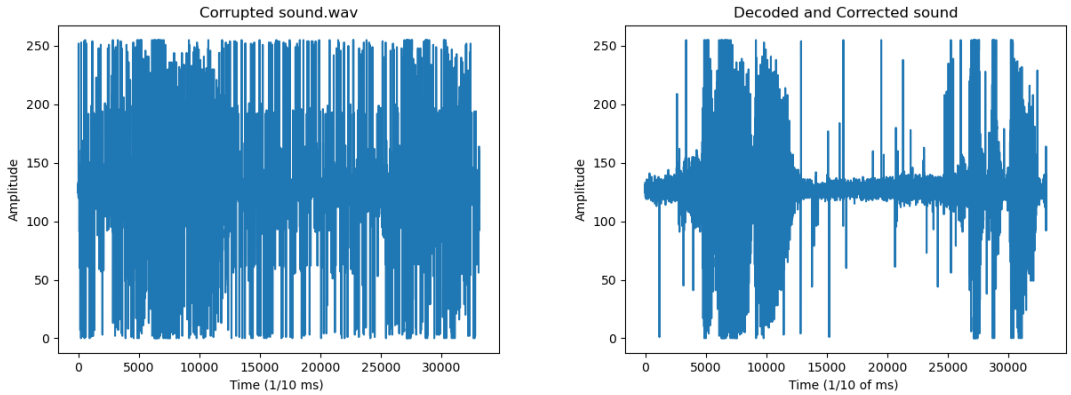
Để sử dụng mã Hamming(7,4) tín hiệu được đưa về dạng số nguyên là tách ra làm 2 nửa để mã hoá và truyền qua kênh. Sau đó để thu được âm thanh sau khi qua kênh truyền ta giải mã và gộp 2 nửa tín hiệu lại.

```
    for codeWord in audioCodedValue:
        halfCodedWord1 = codeWord[0:4]
        halfCodedWord2 = codeWord[4:]
        coded1 = hamming.hamming_74(halfCodedWord1)
        coded2 = hamming.hamming_74(halfCodedWord2)
        codedAudio1.append(coded1)
        codedAudio2.append(coded2)
    errorRate = 1
    channelAudio1 = Channel(codedAudio1, errorRate)
    channelAudio2 = Channel(codedAudio2, errorRate)
    decodedHammingAudioBinary = []
    # Get the 7 bits corrupted values from channelAudio1,2
    corruptedHamMsg1 = channelAudio1.getCorruptedMessage()
    corruptedHamMsg2 = channelAudio2.getCorruptedMessage()
    # Decoded the message
    for i in range(len(corruptedHamMsg1)):
        half1 = hamming.decodeHamming7_4(corruptedHamMsg1[i])
        half2 = hamming.decodeHamming7_4(corruptedHamMsg2[i])
        decodedWord = half1 + half2
        decodedHammingAudioBinary.append(decodedWord)
    # Translation from bit to integers
    for i in range(len(decodedHammingAudioBinary)):
        decodedHammingAudioBinary[i] = int(
                                            decodedHammingAudioBinary
                                            [i], 2)
    # Plot of the corrected audio
```

```
plt.plot(decodedHammingAudioBinary[0:33100])
plt.ylabel("Amplitude")
plt.xlabel("Time (in 10th of ms) ")
plt.title("decoded and corrected sound ")
plt.show()
dt = numpy.dtype(numpy.uint8)
data = numpy.array(decodedHammingAudioBinary, dtype=dt)
write("correctedSound.wav", 11025, data)
```

3.2.2 Kết quả và đánh giá

Âm thanh gốc là âm thanh 8-bit, với xác xuất nhiễu là 1%, khi đi qua kênh tín hiệu thu được có thể nghe thấy rõ có sự nhiễu. Đồ thị âm thanh khi đi qua kênh nhiễu (chưa sửa lỗi) thu được như Hình 6(a)



(a) Âm thanh bị nhiễu (xác suất 1%)

(b) Âm thanh được sửa lỗi (xác suất 1 %)

Hình 6: Âm thanh bị nhiễu và Âm thanh sau khi được sửa lỗi

Với xác xuất nhiễu từ 1 % đến 7 % tín hiệu nhận được tuy bị nhiễu rõ rệt nhưng vẫn có thể nghe được thông điệp của âm thanh gốc, tuy nhiên nếu $>$ 7 % tín hiệu thu được không còn nghe rõ.

Với xác suất 1 %, sử dụng mã Hamming(7,4) để mã hoá và sửa sai âm thanh tín hiệu thu được giảm nhiễu rõ rệt như Hình 6(b). Âm thanh thu được với mức nhiễu 10 % vẫn có thể nghe rõ thông điệp.

4 Kết luận

Trong bài tập lớn này nhóm chúng em đã thực hiện 2 bước quan trọng của việc mã hoá thông tin là mã hoá nguồn và mã hoá kênh. Đối với phần ứng dụng Mã Huffman cho mã hoá văn bản, hệ số nén không thay đổi nhiều so với lượng dữ liệu ngày càng tăng. Mã Huffman giúp nén dữ liệu một cách hiệu quả.

Phần mã hoá kênh, có thể nhận thấy rõ việc sử dụng mã phát hiện sai và sửa sai giúp giảm sự nhiễu đáng kể trong việc truyền thông tin. Trên thực tế mã Hamming(7,4) không còn phổ biến như các thuật toán hiện đại hơn như Reed-Solomon, Turbor, Shor và còn tồn tại nhược điểm như chỉ sửa được 1 lỗi sai và hiệu quả kém với kênh có lỗi chùm (burst error). Nhưng qua bài tập lớn này chúng em vẫn có thể nhận thấy rõ sự hiệu quả và tính quan trọng của việc mã hoá nguồn và mã hoá kênh trong việc truyền thông tin

Tài liệu

[1] Wikipedia. Hamming(7,4). https://en.wikipedia.org/wiki/Hamming(7,4).