Team: Bakhtiyar Rakhimzhanov, Nazira Tukeyeva

Group: BD-1903

Github Link to the Repository: <https://github.com/nazirait/Information-Theory-Project>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Github Link of Bakhtiyar: <https://github.com/Godadoreu>

Github Link of Nazira: <https://github.com/nazirait>

**Assignment 6 Report**

|  |  |
| --- | --- |
| Task Part | Responsible team member |
| Hamming (7,4) | Nazira |
| Hamming (15,11) | Bakhtiyar |
| Assignment Report | Both |

**General Description:**

**HAMMING CODE**

Steps 1 & 2: Reading the file, defining separator function to split bits into 11 and 4 bits sequence.

def decoded\_text(texts):

'''The given function reads sequence of bits'''

# Checking if the text is txt format

f = open(texts)

# reading the file

reading = f.read()

# Closing the file

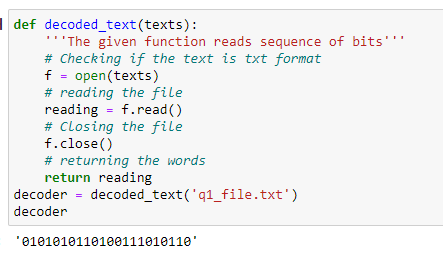
f.close()

# returning the words

return reading

decoder = decoded\_text('q1\_file.txt')

decoder



def separator(string,num):

'''The following function separates the bits into a specified number of bits'''

string = list(string)

count = 0

strings = []

words = ''

for x in string:

words+=x

count+=1

if count%num==0:

strings.append(words)

words = ''

if words!='':

while count%num!=0:

words+='0'

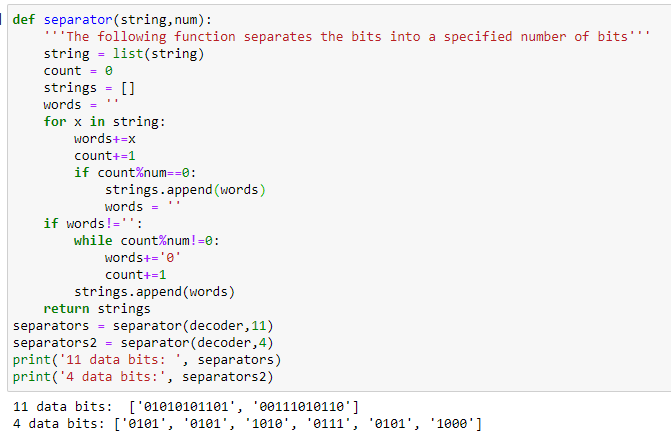
count+=1

strings.append(words)

return strings

separators = separator(decoder,11)

separators2 = separator(decoder,4)



**Hamming Code (15,11)**

def create\_list(strings):

'''Creating the list from the bit strings'''

results = []

for x in strings:

list\_of\_bits = list(x)

length = len(list\_of\_bits)

# turning the list data into integer

for x in range(length):

list\_of\_bits[x] = int(list\_of\_bits[x])

# appending the none values to the bits

list\_of\_bits.insert(0,None)

list\_of\_bits.insert(1,None)

list\_of\_bits.insert(2,None)

list\_of\_bits.insert(4,None)

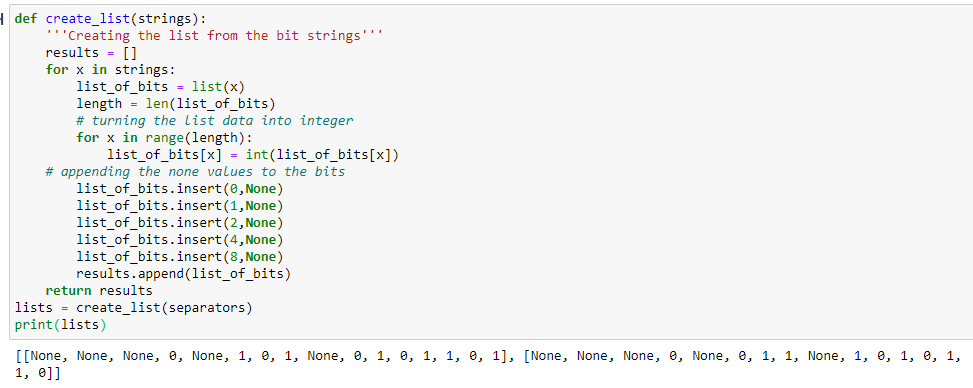
list\_of\_bits.insert(8,None)

results.append(list\_of\_bits)

return results

lists = create\_list(separators)

print(lists)



def HammingEncode(bits):

'''The given function returns extended hamming code'''

for bites in bits:

# adding the values to the list by specific columns and rows

value\_1 = bites[3] + bites[5] + bites[7] + bites[9] + bites[11] + bites[13] + bites[15]

value\_2 = bites[3] + bites[6] + bites[7] + bites[10] + bites[11] + bites[14] + bites[15]

value\_4 = bites[5] + bites[6] + bites[7] + bites[12] + bites[13] + bites[14] + bites[15]

value\_8 = bites[9] + bites[10] + bites[11] + bites[12] + bites[13] + bites[14] + bites[15]

# if the sum is even

if value\_1%2 != 0:

bites[1] = 1

else:

bites[1] = 0

if value\_2%2!=0:

bites[2] = 1

else:

bites[2] = 0

if value\_4%2!=0:

bites[4] = 1

else:

bites[4] = 0

if value\_8%2!=0:

bites[8] = 1

else:

bites[8] = 0

if bits.count(1)%2==0:

bites[0] = 0

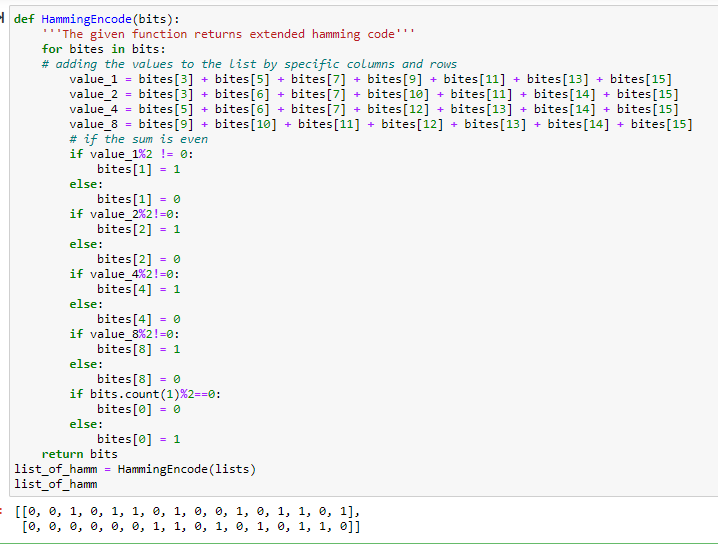
else:

bites[0] = 1

return bits

list\_of\_hamm = HammingEncode(lists)

list\_of\_hamm



def add\_save(list\_of\_bits):

'''The given function returns the bitstrings after Hamming Coding and save the result in new txt file'''

# turning the list data into string type

for x in range(len(list\_of\_bits)):

list\_of\_bits[x] = [str(x) for x in list\_of\_bits[x]]

# joining the results into one string

bitstring = ''

for x in list\_of\_bits:

for values in x:

bitstring+=values

file = open('hamm.txt','w')

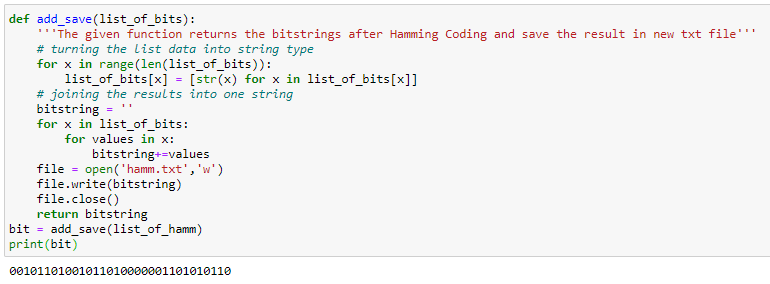
file.write(bitstring)

file.close()

return bitstring

bit = add\_save(list\_of\_hamm)

print(bit)

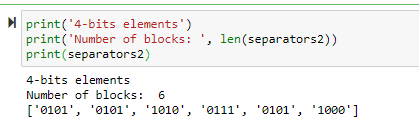


**Hamming Code (7,4)**

print('4-bits elements')

print('Number of blocks: ', len(separators2))

print(separators2)



# 8 bit codeword (adding parity bits)

parityBits = '0'

dataBlocks = []

for i in separators2:

bits1 = 3 \* parityBits + i[0] + parityBits + i[1:4]

dataBlocks.append(bits1)

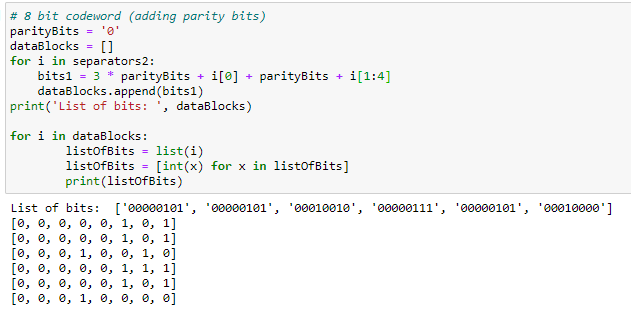
print('List of bits: ', dataBlocks)

for i in dataBlocks:

listOfBits = list(i)

listOfBits = [int(x) for x in listOfBits]

print(listOfBits)



def HammingEncode2(bitString):

'''The following function returns the 8-bit codeword for all data blocks'''

encoded\_list = []

for i in bitString:

blocks = list(i)

blocks = [int(x) for x in blocks]

for x in blocks:

blocks[1] = blocks[3] + blocks[5] + blocks[7]

blocks[2] = blocks[3] + blocks[6] + blocks[7]

blocks[4] = blocks[5] + blocks[6] + blocks[7]

blocks[0] = sum(blocks[1:])

for x in (0,1,2,4):

if blocks[x] % 2 == 0:

blocks[x] = 0

else:

blocks[x] = 1

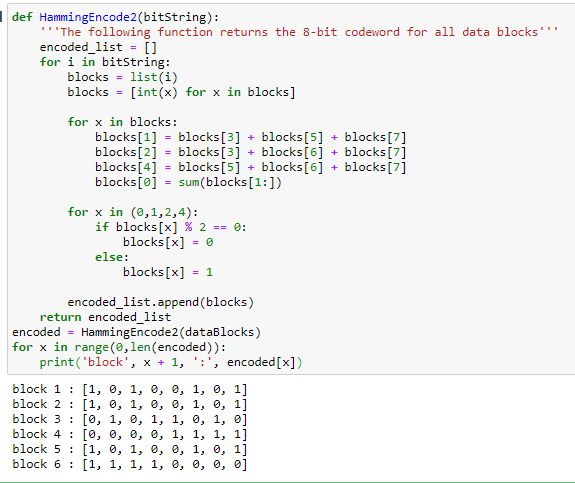
encoded\_list.append(blocks)

return encoded\_list

encoded = HammingEncode2(dataBlocks)

for x in range(0,len(encoded)):

print('block', x + 1, ':', encoded[x])



def bitsSequence(encodedBits):

'''The following function saves the combined sequence to a new txt file'''

for x in range(len(encodedBits)):

encodedBits[x] = [str(x) for x in encodedBits[x]]

allBlocks = ''

for x in encodedBits:

for values in x:

allBlocks+=values

f = open('hamming74.txt','w')

f.write(allBlocks)

f.close()

return allBlocks

bits = bitsSequence(encoded)

bits

