Assignment 7

Implementation in decoding code:

SecondApproach\_2 = open('outputtxt.txt','r',encoding='utf-8').read()

#Read our output from the previous one and the main goal here

#Is goging to readit and then divivded into the initial\_blockss

BitStringg = []

CodeBllcoks = []

SecondStringgBlocks=[]

#We need to set down the related bit string and the code blocks here

for i in SecondApproach\_2:

    CodeBllcoks.append(i)

#Started the iteration

    if len(CodeBllcoks) == 16:

        ThesString = ''.join(CodeBllcoks)

        BitStringg.append(ThesString)

        CodeBllcoks = []

#Second Iteration considered

for i in SecondApproach\_CodeBlocks:

    CodeBlocks.append(i)

#Thefunction decoded

def DECODEDING1115(self,P):

#Define the function DeCODE

    PARITYBITDATAGG = []

    PARITYBITVALUEGG = 0

    if ParityBitGull == 1:

      #consider the each situation if pbit is euqal to 1 then

      PARITYBITDATAGG.extend(self.bits[::P+1])

      PARITYBITVALUEGG = PARITYBITDATAGG[0]

      PARITYBITDATAGG.pop(0)

      self.parityAnalysis(PARITYBITDATAGG,P,PARITYBITVALUEGG)

    elif ParityBitGull in [2,4,8,16]:

#Els

      for i in range( (ParityBitGull-1), len(self.bits), (ParityBitGull\*2) ):

        for j in range(0, ParityBitGull):

          try:

            PARITYBITDATAGG.append(self.bits[i+j])

          except IndexError:

            self.ErrorLog.append("During parity bit" + str(ParityBitGull) +"has been checked" + str(i+j))

      PARITYBITVALUEGG = PARITYBITDATAGG[0]

      #Pop the first\_bit

      PARITYBITDATAGG.pop(0)

      self.parityAnalysis(PARITYBITDATAGG,P,PARITYBITVALUEGG)

TheGulblocks ={}

TheBits = 1

blockGuk = 'The blocks with '

for i in BitStringg:

    TheGulblocks[blockGuk +str(TheBits)] = i

    TheBits+=1

ThesString = 1

print('Previous\_OrgFile is :',SecondApproach\_2)

print('')

itrrBlocks = Gu1l\_HammingEncodee\_15\_Impp(GgFunc[i])

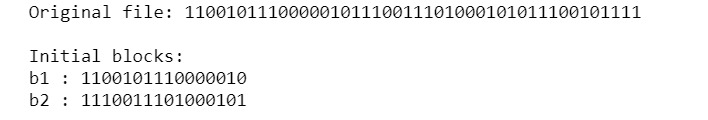
The\_oOutPut = The\_oOutPut + itrrBlocksw

print('The Initial\_Blocks is :')

for i in blocks.values():

    print('CodeBllcoks : '+str(ThesString),i)

ThesString=ThesString+1



import random

#Import the things that we need fromt the initial

def Gul\_ErrorGen(percent,bitstring):

    BitStringg = []

    CodeBllcoks = []

    for i in bitstring:

        CodeBllcoks.append(i)

        if len(CodeBllcoks) == 16:

            ThesString = ''.join(CodeBllcoks)

            BitStringg.append(a)

            CodeBllcoks = []

Gassignment\_pri = open('LastVersion.txt','r',encoding='utf-8').read()

#Consist the previous

Gbitss\_data = 11

#Similar with prs

GFunctionB = [Gassignment\_pri[i:i+Gbitss\_data] for i in range(0, len(Gassignment\_pri), Gbitss\_data)]

#Statr\_the\_iteration

print(GFunctionB)

print('Output as following:')#Get the second itr

for i in range(len(GFunctionB))

    TheGulblocks ={}

    TheBits = 1

    blockGuk = 'block '

    for i in BitStringg:

        TheGulblocks[blockGuk +str(TheBits)] = i

        TheBits+=1

    ThesString = 1

    Gul\_Bblocks\_Innm = int(len(TheGulblocks)\*percent\*0.01)

    BitStringg = list(TheGulblocks.values())

    TheBitNumber = random.sample(range(1, len(TheGulblocks)), Gul\_Bblocks\_Innm)

    Gul\_The\_Error\_check\_bit = []

    for i in TheBitNumber:

        TheBits = random.randint(1,15)

        if int(BitStringg[i-1][TheBits])==0:

            BitStringg[i-1]=BitStringg[i-1][:TheBits]+str(1)+BitStringg[i-1][TheBits+1:]

            TheGulblocks['block '+str(i)] = BitStringg[i-1]

        elif int(BitStringg[i-1][TheBits])==1:

            BitStringg[i-1]=BitStringg[i-1][:TheBits]+str(0)+BitStringg[i-1][TheBits+1:]

            TheGulblocks['block '+str(i)] = BitStringg[i-1]

        Gul\_The\_Error\_check\_bit.append(BitStringg[i-1])

    TheBits = 1

    print('Blocks with error:')

    for i in Gul\_The\_Error\_check\_bit:

        print('block : '+str(TheBits)+,i)

        TheBits+=1

    corrupted\_string = ''

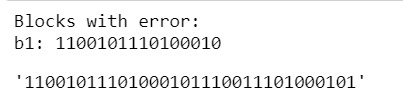
    for i in blocks.values():

        corrupted\_string+=i

    return corrupted\_string

Theerrorrr = Gul\_ErrorGen(50,encoded16[0])

Theerrorrr



our\_file = open('outputtxt.txt','r',encoding='utf-8').read()

def btob(enhypen):

L = []

bts = []

for i in enhypen:

bts.append(i)

if len(bts) == 8:

ailee = ''.join(bts)

L.append(ailee)

bts = []

blockb ={}

jin = 1

btob = 'block '

for i in L:

blockb[btob +str(jin)] = i

jin+=1

return blockb

Bangchan = btob(our\_file)

jin = 1

print('Original\_file:',our\_file)

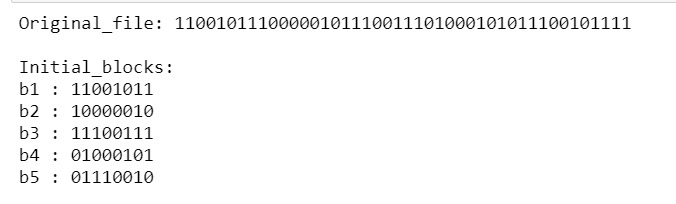
print()

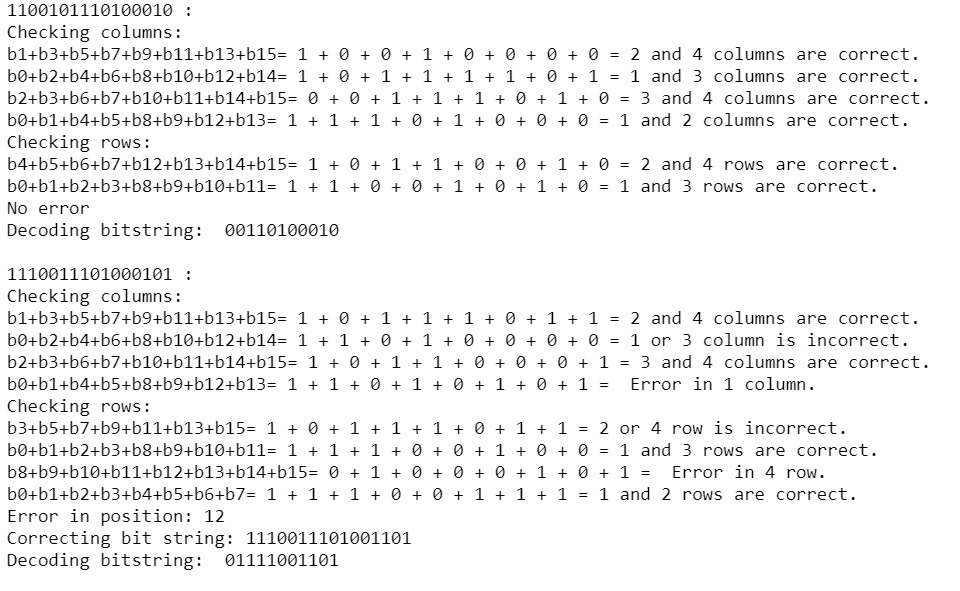
print('Initial\_blocks:')

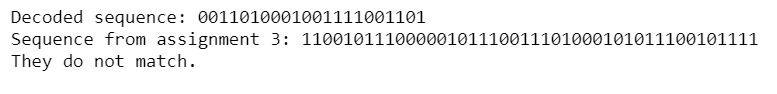
for i in Bangchan.values():

print('b'+str(jin),':',i)

jin+=1







from random import randint

def Exid(pace,booster):

L = []

bts = []

for i in booster:

bts.append(i)

if len(bts) == 8:

ailee = ''.join(bts)

L.append(ailee)

bts = []

blockb ={}

jin = 1

btob = 'block '

for i in L:

blockb[btob +str(jin)] = i

jin+=1

Nice = int(len(blockb)\*pace\*0.01)

L = list(blockb.values())

import random

rainbow = random.sample(range(1, len(blockb)), Nice)

error = []

for i in rainbow:

jin = random.randint(1,7)

if int(L[i-1][jin])==0:

L[i-1]=L[i-1][:jin]+str(1)+L[i-1][jin+1:]

blockb['block '+str(i)] = L[i-1]

elif int(L[i-1][jin])==1:

L[i-1]=L[i-1][:jin]+str(0)+L[i-1][jin+1:]

blockb['block '+str(i)] = L[i-1]

error.append(L[i-1])

jin = 1

print('Detecting errors in this block(s):')

for i in error:

print('block'+str(jin)+':',i)

jin+=1

stay = ''

for i in blockb.values():

stay+=i

return stay

exo = Exid(30,our\_file)



def Haven(booster):

L = []

bts = []

for i in booster:

bts.append(i)

if len(bts) == 8:

ailee = ''.join(bts)

L.append(ailee)

bts = []

blockb ={}

jin = 1

btob = 'block '

for i in L:

blockb[btob +str(jin)] = i

jin+=1

kkami = 0

dori = ''

for i in blockb.values():

kkami+=1

paint0 = i[0]

paint1 = i[1]

paint2 = i[2]

paint3 = i[4]

print(i,':')

print('Checking\_bits:')

ailee = int(i[3])+int(i[5])+int(i[7])

ex = 0

if ailee % 2 == 0:

if int(paint1) == 0:

print(f'p\_1:b\_3+b\_5+b\_7 = {i[3]} + {i[5]} + {i[7]} = {paint1} correct' )

else:

print(f'p\_1:b\_3+b\_5+b\_7 = {i[3]} + {i[5]} + {i[7]} = {paint1} incorrect' )

ex+=1

else:

if int(paint1) == 1:

print(f'p\_1:b\_3+b\_5+b\_7 = {i[3]} + {i[5]} + {i[7]} = {paint1} correct' )

else:

print(f'p\_1:b\_3+b\_5+b\_7 = {i[3]} + {i[5]} + {i[7]} = {paint1} incorrect' )

ex+=1

ailee = int(i[3])+int(i[6])+int(i[7])

if ailee % 2 == 0:

if int(paint2) ==0:

print(f'p\_2:b\_3+b\_6+b\_7 = {i[3]} + {i[6]} + {i[7]} = {paint2} correct' )

else:

print(f'p\_2:b\_3+b\_6+b\_7 = {i[3]} + {i[6]} + {i[7]} = {paint2} incorrect' )

ex+=2

else:

if int(paint2) ==1:

print(f'p\_2:b\_3+b\_6+b\_7 = {i[3]} + {i[6]} + {i[7]} = {paint2} correct' )

else:

print(f'p\_2:b\_3+b\_6+b\_7 = {i[3]} + {i[6]} + {i[7]} = {paint2} incorrect' )

ex+=2

ailee = int(i[5])+int(i[6])+int(i[7])

if ailee % 2 == 0:

if int(paint3) == 0:

print(f'p\_1:b5+b\_6+b\_7 = {i[3]} + {i[6]} + {i[7]} = {paint3} correct' )

else:

print(f'p\_1:b5+b\_6+b\_7 = {i[3]} + {i[6]} + {i[7]} = {paint3} incorrect' )

ex+=4

else:

if int(paint3) == 1:

print(f'p\_1:b\_3+b\_5+b\_7 = {i[3]} + {i[6]} + {i[7]} = {paint3} correct' )

else:

print(f'p\_1:b\_3+b\_5+b\_7 = {i[3]} + {i[6]} + {i[7]} = {paint3} incorrect' )

ex+=4

summa = int(paint1)+int(paint2)+int(paint3)+int(i[3])+int(i[5])+int(i[6])+int(i[7])

if summa % 2 == 0:

if int(paint0)==0:

print(f'p\_0:b1+b2+b\_3+b\_4+b\_5+b\_6+b\_7 = {paint1} + {paint2} + {i[3]} + {paint3} + {i[5]} + {i[6]} + {i[7]} = {paint0} correct' )

else:

print(f'p\_0:b1+b2+b\_3+b\_4+b\_5+b\_6+b\_7 = {paint1} + {paint2} + {i[3]} + {paint3} + {i[5]} + {i[6]} + {i[7]} = {paint0} incorrect' )

else:

if int(paint0)==1:

print(f'p\_0:b1+b2+b\_3+b\_4+b\_5+b\_6+b\_7 = {paint1} + {paint2} + {i[3]} + {paint3} + {i[5]} + {i[6]} + {i[7]} = {paint0} correct' )

else:

print(f'p\_0:b1+b2+b\_3+b\_4+b\_5+b\_6+b\_7 = {paint1} + {paint2} + {i[3]} + {paint3} + {i[5]} + {i[6]} + {i[7]} = {paint0} incorrect' )

L = list(blockb.values())

if ex==0:

print('No\_error')

else:

print('Error:',ex)

if L[kkami-1][ex] =='0':

bts = L[kkami-1][:ex]+str(1)+L[kkami-1][ex+1:]

blockb['blocks '+str(kkami)] = bts

else:

bts = L[kkami-1][:ex]+str(0)+L[kkami-1][ex+1:]

blockb['blocks '+str(kkami)] = bts

print(f'Correcting:',blockb['block '+str(kkami)])

back\_door =''

for i in range(8):

if i==3 or i>=5:

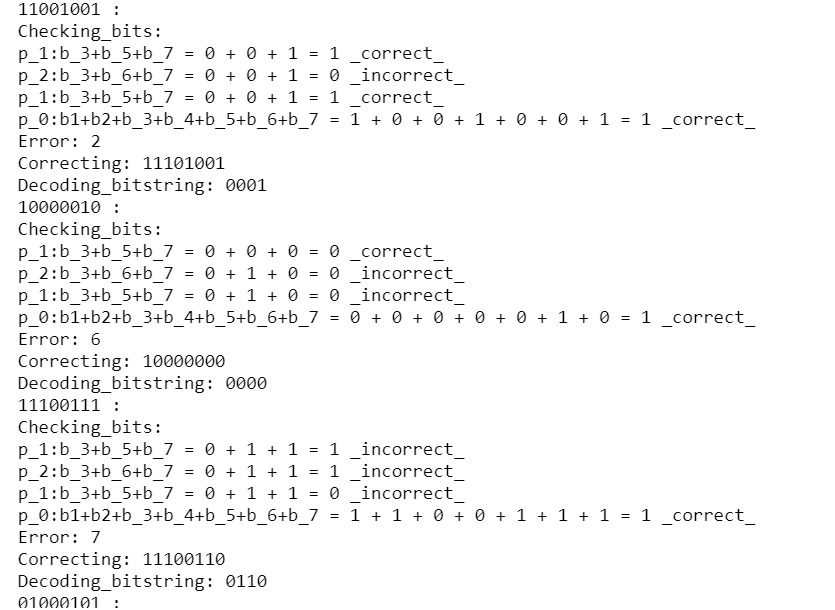
back\_door+=blockb['block '+str(kkami)][i]

dori+=back\_door

print(f'Decoding\_bitstring: {back\_door}')

return dori

dori = Haven(exo)



print('Decoded\_sequence:',dori)

print('assignment 3:',encoded8)

if dori ==encoded8:

print('They match.')

else:

print('They do not match.')

