# MET CS 521 TERM PROJECT REPORT

## Table of Contents

# ABOUT THIS PROJECT

This document and code included in this folder/directory serves as the deliverables for term project for MET CS 521. This project shows usage of various python structures and operations and fulfills the requirements as stated in the Project Guideline document. This project implements two types of lossless data compression. Lossless compression is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data. The algorithms used in this application are Huffman algorithm and Run-Length algorithm. The idea behind Huffman algorithm is to assign variable-length codes to input characters where lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code, and the least frequent character gets the largest code. The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream. As for the Run-length encoding (RLE), it is a form of lossless data compression in which runs of data (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run.

# SET UP

This project requires Python 3.10 or above to execute.

# INTERACTING WITH THE APPLICATION

Menu.py serves as the entry point for our project/application. This file displays a variety of options. Option 1 allows user to encode data using Huffman algorithm by obtaining the user input via console. Option 2 allows user to encode data using Huffman algorithm for the data present in file HuffmanFile.txt. Currently, the file includes the nerdy poem which was provided to us by professor for the implementation of string editor. Option 3 allows user to enter the input and encode the data using run-length algorithm. Option 4 terminates the program. After execution of each option, user is prompted to continue with the execution of program to perform any of the options mentioned above. User can enter y or Y to continue. Any other key press will terminate the program.

# PROJECT REQUIREMENTS & FULFILLMENT

The locations of code snippets that implement each project requirement are shown in the screenshots below. Each screenshot has the file name and the location (line number) of occurrence of said implementation.

1. CONTAINER TYPE

TUPLE

```
Menu.py > ...
4  #DEFINE MENU ITEMS
5  mymenu = ("Press 1 to Huffman Encode an input String","Press 2 to Huffman Encode contents of a file",
6            "Press 3 to Run Length Encode an input String", "Press 4 to Exit")
7
```

LIST

```
Huffman.py ×
Huffman.py > Huffman > decode
112     def decode(self):
113         encoded_data = self.__encodedOutput
114         huffman_tree = self.__nodes[0]
115         tree_head = huffman_tree
116         decoded_output = []        You, 21 hours ago • updates to accommodate requirements …
117         for x in encoded_data:
118             if x == '1':
119                 huffman_tree = huffman_tree.right
120             elif x == '0':
121                 huffman_tree = huffman_tree.left
122             try:
123                 if huffman_tree.left.symbol == None and huffman_tree.right.symbol == None:
124                     pass
125             except AttributeError:
126                 decoded_output.append(huffman_tree.symbol)
127                 huffman_tree = tree_head
128
129         string = ''.join([str(item) for item in decoded_output])
130         self.__decodedOutput = string
131         return string
```

DICTIONARY

```
Huffman.py ×
Huffman.py > Huffman > decode
10      __symbols = dict()
11      codes = dict()
```

2. ITERATION TYPE

WHILE LOOP

Menu.py > ...

```python
#LOOP AS LONG AS USER SAYS y OR Y
while continueAnswer == 'y' or continueAnswer == 'Y':
    menuinput = __printMenu()
    #LOOP TILL USER DOESNT PRESS 4
    while menuinput.isnumeric() == False or (int(menuinput) not in range(1,5)):
```

FOR LOOP

RunLength.py > RunLength > __formatOutput

```python
    def decode(self):
        for item in self.__encodedList:
            self.__decodedList.append(item[0] * item[1])
        self.decodedSequence = "".join(self.__decodedList)
        return self.decodedSequence
```

3. CONDITIONAL (IF)

RunLength.py > RunLength > __formatOutput

```python
            if x == 0:
                continue
            elif item == self.__sequence[x - 1]:
                count += 1
            else:
                self.__encodedList.append((self.__sequence[x - 1], count))
                count = 1
```

4. TRY BLOCKS

Huffman.py > Huffman > __repr__

```python
                huffman_tree = huffman_tree.left
            try:
                if huffman_tree.left.symbol == None and huffman_tree.right.symbol == None:
                    pass
            except AttributeError:
                decoded_output.append(huffman_tree.symbol)
                huffman_tree = tree_head
```

5. USER DEFINED FUNCTIONS

Menu.py M ×

Menu.py > ...

```python
#PRINT MENU ON CONSOLE
def __printMenu():
    for i in mymenu:
        print(i)
    menuinput = input()
    return menuinput

#PROMPT USER TO CONTINUE DOING MORE OPERATIONS
def __continuePrompt():
    continueAnswer = input("Do you want to continue?(Y/N)")
    return continueAnswer
```

6. INPUT AND/OR OUTPUT FILE (submit input data)

EXPLORER ...

OPEN EDITORS

× Huffman.py

× HuffmanFile.txt

METCS521PROJECT

- __pycache__
- Huffman.py
- HuffmanFile.txt
- Menu.py M
- README.docx M
- RunLength.py
- testCases.py M

Huffman.py × | HuffmanFile.txt

Huffman.py > ...

```python
    def fileHuffman(self):
        f = open("HuffmanFile.txt", "r")
        data = f.read()
        self.data = data
        encoding = self.encode()
        print("\n============:: Encoded Output ::============",encoding, sep='\n')
        print("\n============:: Decoded Output ::============",self.decode(),"\n", sep='\n')

    def __repr__(self):
        return f'Huffman('+self.data+','+self.__encodedOutput+','+self.__decodedOutput+')'
                You, 21 hours ago • updates to accommodate requirements ...
```

7. USER-DEFINED CLASS. THE CLASS MUST BE IMPORTED BY YOUR MAIN PROGRAM AND HAVE THE FOLLOWING REQUIRED STRUCTURES.

Menu.py

```python
from Huffman import Huffman
from RunLength import RunLength
```

a. AT LEAST 1 PRIVATE AND 2 PUBLIC SELF ATTRIBUTES

RunLength.py

```python
class RunLength:
    encodedSequence = ""
    decodedSequence = ""
    __decodedList = []
    __encodedList = []
    #usage of init method
    def __init__(self, sequence):
        #DO Nothing
        self.__sequence = sequence
```

b. AT LEAST 1 PRIVATE AND 1 PUBLIC METHOD THAT TAKE ARGUMENTS, RETURN VALUES AND ARE USED BY YOUR PROGRAM

RunLength.py

```python
    def decode(self):
        for item in self.__encodedList:
            self.__decodedList.append(item[0] * item[1])
        self.decodedSequence = "".join(self.__decodedList)
        return self.decodedSequence

    def __formatOutput(self, sequence):
        result = []
        for item in sequence:
            if (item[1] == 1):
                result.append(item[0])
            else:
                result.append(item[0] + str(item[1]))
        return "".join(result)
```

c. AN INIT() METHOD THAT TAKES AT LEAST 1 ARGUMENT

RunLength.py M

RunLength.py > RunLength > __init__

```python
class RunLength:
    encodedSequence = ""
    decodedSequence = ""
    __decodedList = []
    __encodedList = []
    #usage of init method
    def __init__(self, sequence):
                    You, now • Uncommitted changes
        self.__sequence = sequence
```

d. A REPR() METHOD

RunLength.py

RunLength.py > ...

```python
    def __repr__(self):
        return f'RunLength('+self.__sequence+','+self.encodedSequence+','+self.decodedSequence+')'
```

8. PROVIDE UNIT TESTS THAT PROVE THAT YOUR CLASS METHODS WORK AS EXPECTED. THE TESTS SHOULD EVALUATE RESULTS USING ASSERT STATEMENTS.

testCases.py M

testCases.py > ...

```python
You, 1 second ago | 1 author (You)
import unittest          You, 5 hours ago • 0507 ...
from RunLength import RunLength
from Huffman import Huffman

You, 1 second ago | 1 author (You)
class TermProjectTestCases(unittest.TestCase):
    __object1 = RunLength("aaaaabb")
    __encodedValue = __object1.encode()
    __decodedValue = __object1.decode()
    __object2 = Huffman("aaan")
    __HFencodedValue = __object2.encode()
    __HFdecodedValue = __object2.decode()

    def test_RL_decode(self):
        self.assertEqual(self.__decodedValue,'aaaaabb')

    def test_RL_encode(self):
        self.assertEqual(self.__encodedValue,'a5b2')

    def test_HF_decode(self):
        self.assertEqual(self.__HFdecodedValue,'aaan')

    def test_HF_encode(self):
        self.assertEqual(self.__HFencodedValue,'0001')

if __name__ == '__main__':
    unittest.main()
```

# SAMPLE I/O SCREENSHOTS

Main menu

```
PS C:\Users\Anukool\OneDrive\Desktop\Boston University\Spring 22 - BU\521\Homework\anukools@bu.edu_final_project> & C:/Python310/python.exe "c:/Users/Anukool/OneDrive/Desktop/Boston University/Spring 22 - BU/521/Homework/anukools@bu.edu_final_project/Menu.py"
Press 1 to Huffman Encode an input String
Press 2 to Huffman Encode contents of a file
Press 3 to Run Length Encode an input String
Press 4 to Exit
```

Selecting 1 and running Huffman Algorithm on an input string

```
1
****** HUFFMAN ENCODE/DECODE AN INPUT STRING   ******

Enter the String to encode using Huffman Algorithm
this is a fun test

============:: List of Symbols in the input ::============
['t', 'h', 'i', 's', ' ', 'a', 'f', 'u', 'n', 'e']

============:: Respective occurence of each symbol ::============
[3, 1, 2, 3, 4, 1, 1, 1, 1, 1]

============:: Symbol and it's Code ::============
{'s': '000', 't': '001', 'e': '0100', 'n': '0101', 'u': '0110', 'f': '0111', 'a': '1000', 'h': '1001', 'i': '101', ' ': '11'}

============:: Space usage before compression (in bits) ::============
144

============:: Space usage after compression (in bits) ::============
56

============:: Encoded Output ::============
00110011010001110100011100011011101100101110010100000001

============:: Decoded Output ::============
this is a fun test
Do you want to continue?(Y/N)
```

Selecting 2 and running Huffman algorithm on contents of a file

```
Press 4 to Exit
2
****** HUFFMAN ENCODE/DECODE CONTENTS OF A FILE  ******


============:: List of Symbols in the input ::============
['t', 'h', 'i', 's', ' ', 'a', 'f', 'u', 'n', 'e', 'B', 'l', 'b', 'r', 'g', 'y', '.', '\n', 'E', 'x', 'p', 'c', 'm', 'S', 'o', 'C', 'd'
]

============:: Respective occurence of each symbol ::============
[19, 5, 14, 7, 20, 7, 2, 4, 5, 14, 1, 8, 4, 4, 1, 1, 4, 3, 1, 3, 6, 5, 5, 1, 3, 1, 1]

============:: Symbol and it's Code ::============
{'s': '010001', 't': '101', 'e': '1100000', 'n': '1100001', 'u': '11101', 'f': '110011', 'a': '1100100', 'h': '1100101', 'i': '0010', '
 ': '011', 'd': '0000011', 'o': '001110', 'x': '001111', '\n': '010000', 'p': '01001', 'C': '0101010', 'S': '0101011', 'm': '01011', 'c
': '10000', 'E': '1001100', 'y': '1001101', 'g': '1001110', 'B': '1001111', '.': '11010', 'r': '11011', 'b': '11100', 'l': '1111'}

============:: Space usage before compression (in bits) ::============
1048

============:: Space usage after compression (in bits) ::============
653

============:: Encoded Output ::============
10011111110000011001001110110100101100111110111111011001001000101111110011000001011011100000110110111011100101110010011000010111111101100111
0111110011011101001000010011000011110100111110010100000001010101100100100010111110011000001011011100000110110111011100101110010011000010
1100100101101001111100101000000101011101001000001010110010010110100111111100000011001001000101111110011000001011011100000110110111011100
1011100100110000101110000000111001011010011111111000000001111110100100000101010001110010110100111111110000000011110110010010001011111100110000
001011011100000011011011101110010111001001100001011100000001110010110100111111001010000011001001011100000000000001111010

============:: Decoded Output ::============
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
```

Selecting 3 and running run length algorithm

```
Do you want to continue?(Y/N)y
Press 1 to Huffman Encode an input String
Press 2 to Huffman Encode contents of a file
Press 3 to Run Length Encode an input String
Press 4 to Exit
3
****** RUN LENGTH ENCODE/DECODE AN INPUT STRING ******

Enter the String to encode using Run-Length Algorithm
aaaaaaaaaaaaaaaaaaabbbyyyyyyyyyyyyyyyyyyyyyy

============:: Encoded Output ::============
a19b3y22

============:: Decoded Output ::============
aaaaaaaaaaaaaaaaaaabbbyyyyyyyyyyyyyyyyyyyyyy
Do you want to continue?(Y/N)
```