COMPRESSION FOR ENGLISH TEXT

EE539: Principles of Information Theory and Coding

Abstract

A report on the fulfillment of Exercise 3, focusing on the compression and decompression processes of an English text file.

Jayant Som

MS Student in Quantum Science and Nanotechnology

Lempel-Ziv compression algorithm

In Lempel-Ziv compression algorithm, the idea is that if some text is not random, a substring that has already been encountered is more likely to reoccur than those we haven't observed yet. The LZ78 algorithm builds a dictionary of substrings, referred to as "phrases," that have appeared in the text. This algorithm creates its dictionary dynamically, processing the data in a single pass. This feature is particularly beneficial because it allows us to begin encoding without needing to have the full document beforehand.

LZ78 compression works by building a dictionary of substrings as it reads the input text. Each compressed token consists of:

- A dictionary index (binary) referring to a previously stored substring.
- A new character or the last bit.

Rules to apply LZ algorithm:

- The first step in applying LZ algorithm is to divide the given text into segments.
- These segments should be the shortest segments, not encountered previously. All these segments are referred as "Phrase".
- We assign a unique incremental index to each Phrase we encounter while traversing through the text.
- We encode each phrase by prefix index and last bit.
- We convert the assigned code for each Phrase into its binary value.
- The binary conversion consists of 2 parts,
    - the first part is the prefix index code – we convert the numerical index to its binary value.
    - the second part is the last bit – we can convert the last bit in multiple ways:
        - If the number of unique characters is less, we can assume a mapping and share the same mapping dictionary with the decoder.
        - If the frequencies of unique letters are known, we could even use a Huffman code to encode the letters.
        - If the number of unique characters is known, we can use the universal ASCII conversion, which the decoder will also have.

Example of LZ algorithm

COMPRESSION

1. Segmentation:

Suppose, for example, we have the given input string:

AABABBBABAABABBBABBABB

We start with the shortest phrase on the left that we haven't encountered before. This will always be a single letter, in this case A:

A|ABABBBABAABABBBABBABB

We now take the next phrase we haven't come across. We've already seen A, so now we take AB:

A|AB|ABBBABAABABBBABBABB

The next phrase we haven't encountered is ABB, as we've already seen AB. We continue further, we get B after that:

A|AB|ABB|B|ABAABABBBABBABB

and we finally get the segmented text:

A|AB|ABB|B|ABA|ABAB|BB|ABBA|BB

2. Encoding:

Now, we need to encode the segmented text. For each phrase we see, we prepare a dictionary of indexes. The next time we want to send it, we don't send the entire phrase, but just the index of this phrase. Consider the following tabular notation, in which the first row gives the index of the phrase, the second row represents the phrases, and the third row their encodings:

1	2	3	4	5	6	7	8	9
A	AB	ABB	B	ABA	ABAB	BB	ABBA	BB
∅A	1B	2B	∅B	2A	5B	4B	3A	7

For instance, when we're encoding the segment ABAB (the 6th phrase), we encode it as 5B. This maps to ABAB since the 5th phrase was ABA, and we add B to it.

- Here, the empty set ∅ should be considered as the 0'th phrase and encoded by 0.
- The last phrase BB at index 9 is same as at index 7, so we don't need any symbolic representation to denote it. We just consider the previously encountered index value.

3. Binary conversion:

Now, we convert each encoding symbol into its binary notations. In this example, I have mapped A to 0 and B to 1:

0, 0|1, 1|10,1|00, 1|010,0|101, 1|100,1|011,0|0111

Finally, the Binary representation of the encoding looks like:

00111010010100101111001011001111

DECOMPRESSION

The binary string is:

00111010010100101111001011001111

We need to split this into (index, last bit) pairs. To do this, we need to know how many bits are used for the index and how many for the last bit. Since A and B are single bits (0 and 1), the last bit is always 1 bit. The index is represented using a variable number of bits, depending on the size of the dictionary.

1. We will start with an empty dictionary.
2. We will reconstruct the dictionary step by step - we will process the binary string step by step, extracting the prefix and the new symbol, and updating the dictionary.
3. We will use the dictionary references to rebuild the original sequence.
4. While compressing, we assumed A=0 and B=1, so while decompressing, we will need to use the same mapping.

Decoding Process:

First, we convert the binary string into an encoded list of tuples. To get a list of tuples, we first find the index value by increasing the number of bits dynamically (as the entries increase) and get the index and the corresponding character (where 0 is mapped to 'A' and 1 to 'B') from it. After that, the (index, last bit) tuple is added to the resultant list. It involves the procedure of dividing the binary string into the chunks and changing the bit-length of the index to be followed while the string is being processed. We stop when the available bits are not enough to form a full index or pair of characters. In this case, the final list of tuples will be returned.

After that, we need to decompress the data that has been encoded in a LZ like format. We do this by looping through our list of tuples (index, last_bit). We maintain a phrase dictionary to keep track of phrases we've seen before, where the index points to a phrase in the dictionary

and last_bit is just a single character ('A' or 'B'). For each tuple, we rebuild the phrase by either taking last_bit directly (if the index is 0) or by appending last_bit to the phrase indicated by the index. The newly reconstructed phrase gets added to our decompressed text and is also stored in the dictionary for future use. Finally, we output the fully decompressed text.

Decoded Phases:

A, AB, ABB, B, ABA, ABAB, BB, ABBA, BB

Decoded Text:

AABABBBABAABABBBABBABB

Implementation of LZ algorithm

I have implemented the LZ algorithm using Python programming language. My program consists of 3 modules:

- Compression

- Decompression

- Comparison

The Orchestrator is used to run the 3 modules sequentially one after the another.

1. Compressor

This program is used to compress the input text using LZ algorithm.

1. Imports

I have imported the os library to perform file operations.

In [ ]:
```
import os
```

2. Define functions

This function is used to read a text file and return its content.

In [ ]:
```
def read_text_file(file_path):
    with open(file_path, 'r') as file :
        return file.read().strip()
```

This function is used to write binary data to a file.

In [ ]:
```
def write_binary_file(file_path, binary_data):
    with open(file_path, 'wb') as file :
        file.write(binary_data)
```

This function is used to compress the input text using LZ78 algorithm.

It compresses the text and generates encodings for each segment.

In [ ]:
```
def lempel_ziv_compression(input_text):
    phrase_dictionary = {}
    next_index = 1
    current_phrase = ""
```

```
    compressed_text = []

    for char in input_text:
        new_phrase = current_phrase + char
        if new_phrase not in phrase_dictionary :
            if current_phrase:
                compressed_text.append((phrase_dictionary[current_phrase],
                                        char))
            else:
                compressed_text.append((0, char))
            phrase_dictionary[new_phrase] = next_index
            next_index += 1
            current_phrase = ""
        else:
            current_phrase = new_phrase

    if current_phrase:
        compressed_text.append((phrase_dictionary[current_phrase] , ''))

    return compressed_text
```

This function is used to convert the compressed text encodings into binary format.

- 8-bit binary for index
- 8-bit binary for the last bit (ASCII)

In [ ]:
```
def compressed_phrase_to_binary(compressed_data):
    binary_output = ""
    for index, char in compressed_data :
        binary_output += f"{index:08b}"
        if char:
            binary_output += f"{ord(char):08b}"
    return binary_output
```

3. Main function and code execution

The main function is where all the operations are happening. I am calling all the methods defined above in the main function and executing them systematically and producing the output.

In [ ]:
```
def main():
    # I/p and O/p file paths
    input_file = "./Dumps/Input_Text_File.txt"
    compressed_file= "./Dumps/Compressed_version.bin"

    # Step 1: Reading the input text file and finding its size in bits
    input_text = read_text_file(input_file)
    print("\033[93m\nInput text:\033[0m", input_text)
    print("\033[96m\n    Size of the input text:\033[0m",
          len(input_text)*8,"Bits")

    # Step 2: Compressing the input text using the LZ78 algorithm
```

```
    compressed_data = lempel_ziv_compression(input_text)

    # Step 3: Converting the compressed phrases into binary format
    binary_data = compressed_phrase_to_binary(compressed_data)

    # Step 4: Writing the compressed binary data to a binary file
    write_binary_file(compressed_file, binary_data.encode('utf-8'))
    print("\033[92m\n    Size of the compressed binary output:\033[0m",
          len(binary_data),"Bits")

if __name__ == "__main__":
    main()
```

2. Decompressor

This program is used to decompress a decoded text generated using LZ algorithm

1. Imports

I have imported the os library to perform file operations.

In [ ]:
```
import os
```

2. Define functions

This function is used to read a binary file and return its content.

In [ ]:
```
def read_binary_file(file_path):
    try:
        with open(file_path, 'rb') as file:
            return file.read()
    except FileNotFoundError:
        print(f"Error: File not found.")
        return None
    except IOError as e:
        print(f"Error reading file : {e}")
        return None
```

This function is used to read a binary file and return its content.

In [ ]:
```
# Used to write to text file.
def write_decoded_text_file(file_path  , text) :
    with open(file_path , "w" , encoding  ="utf-8") as file:
        file.write(text)
```

This function is used to convert compressed binary data back into phrase encoded format.

In [ ]:
```
def binary_to_encoded(binary_data ):
    encoded_data= []
    i = 0
    while i < len(binary_data) :
        index = int( binary_data[i:i+8], 2)
        i += 8
        if i + 8 <= len( binary_data):
            char = chr(int(binary_data[i:i+8] , 2))
            i += 8
        else:
            char = ''
        encoded_data.append((index, char ))
    return encoded_data
```

This function is used to decompress the phrase encoded data back into text.

In [ ]:
```
def lempel_ziv_decompression(compressed_data ) :
    phrase_dictionary = {}
    next_index = 1
    decompressed_text = ""

    for index , last_bit in compressed_data:
        if index == 0 :
            phrase =last_bit
        else :
            phrase = phrase_dictionary[index] + last_bit

        decompressed_text +=  phrase
        phrase_dictionary[next_index] = phrase
        next_index += 1

    return decompressed_text
```

3. Main function and code execution

The main function is where all the operations are happening. I am calling all the methods defined above in the main function and executing them systematically and producing the output.

In [ ]:
```
def main() :
    # Compressed and decompressed o/p file paths
    compressed_file = "./Dumps/Compressed_version.bin"
    decoded_file  = "./Dumps/Decoded_Text_File.txt"

    # Step 1: Reading the binary file containing the compressed binary data
    compressed_binary_data = read_binary_file(compressed_file)

    # Step 2: Converting compressed binary data back into encoded phrase for
    encoded_data = binary_to_encoded(compressed_binary_data )

    # Step 3: Converting the LZ78 encoded phrase format back into text.
    decoded_text= lempel_ziv_decompression(encoded_data)
```

```
    # Step 4: Saving the decompressed text to a file
    write_decoded_text_file(decoded_file , decoded_text)
    print("\033[91m\nDecoded text:\033[0m", decoded_text,
          "\033[91m\033[0m")

if __name__ == "__main__":
    main()
```

3. Comparator

This program is used to compare the input text with the decoded text and generate the comparison result.

1. Imports

I have imported the os library to perform file operations.

In [ ]:
```
import os
```

2. Define functions

This function is used to read a text file and return its content.

In [ ]:
```
def read_text_file(file_path) :
    with open(file_path , 'r') as file :
        return file.read().strip()
```

This function is used to compare the input text and decoded text.

In [ ]:
```
def compare_ip_op(input_text , decoded_text):
    if input_text == decoded_text :
        return "Good"
    else :
        return "Bad"
```

3. Main function and code execution

The main function is where all the operations are happening. I am calling all the methods defined above in the main function and executing them systematically and producing the output.

In [ ]:
```
def main():
    # I/p and decompressed O/p file paths
    input_file= "./Dumps/Input_Text_File.txt"
    decoded_file = "./Dumps/Decoded_Text_File.txt"

    # Step 1: Reading the decoded text file
    final_decoded_text = read_text_file(decoded_file)

    # Step 2: Reading the input text file
    input_text= read_text_file(input_file)

    # Step 3: Comparing input text with the final decoded text
    comparison_result = compare_ip_op(input_text , final_decoded_text)
    print("\033[95m\nComparison Result:\033[0m", comparison_result,
          "\n\033[95m\033[0m")
```

```
if __name__ ==  "__main__" :
    main()
```

EXAMPLE 1

Input Text:

aabbaabbaabbabbabbaabbaabbaabaabaabba

EXAMPLE 2

Input Text:

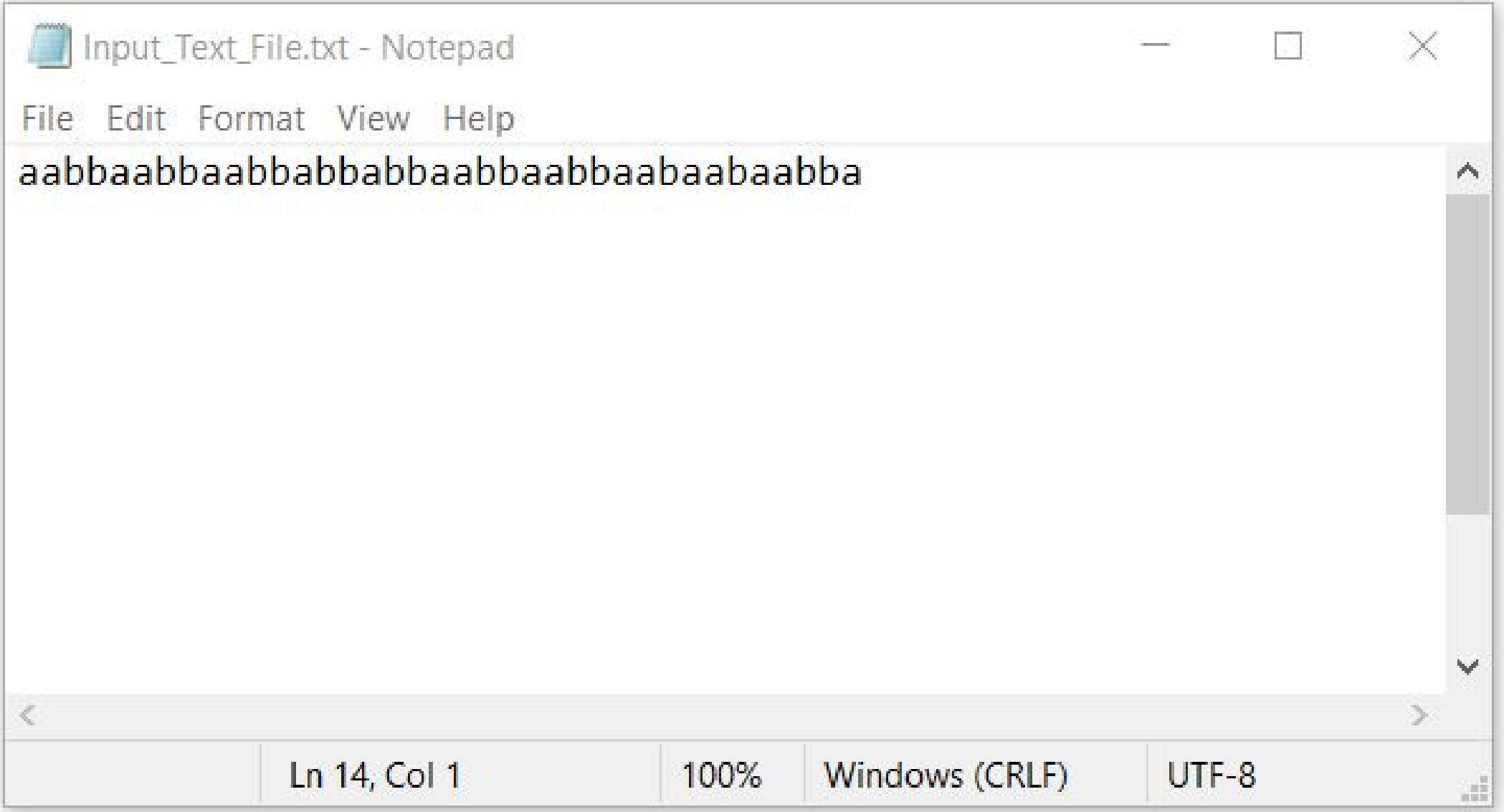
Who Am I? Why Am I? In philosophy, the Upanishads are Sanskrit texts with central concern to discover the relations between cosmic realities, consciousness and the human body or person, postulating Atman and Brahman as the summit of the interconnected universe. Brahman connotes the highest universal principle, the Ultimate Reality of the universe. Atman is the true or eternal Self or the self existent essence within each individual. The Bhagavad Gita summarizes the key philosophies of the Upanishads with a major focus on the philosophy of karma.

Orchestrator : Example 1

This program is used to run the modules sequentially one after the other.

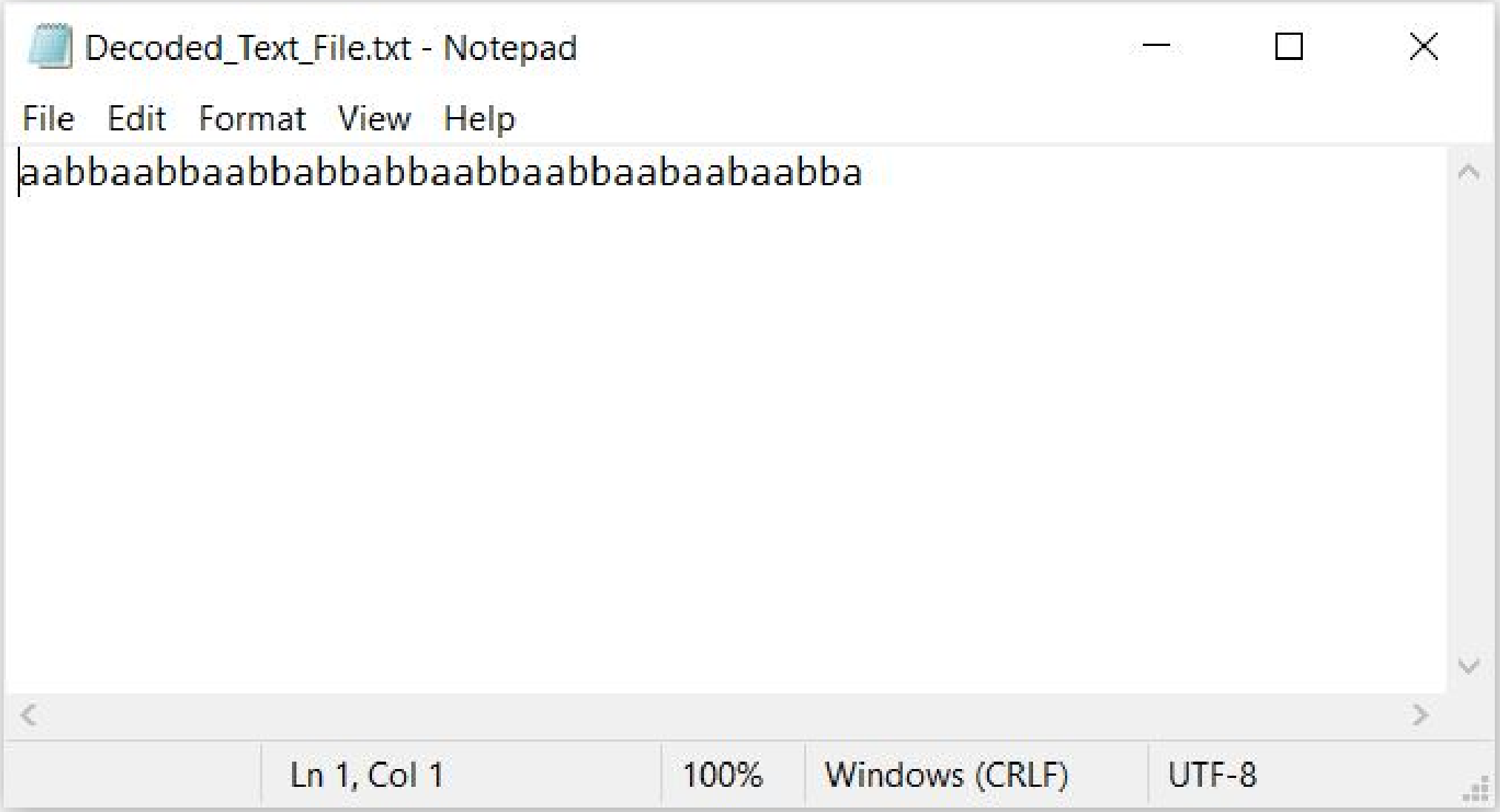
In [1]:
```
%run 1_Compression.ipynb

%run 2_Decompression.ipynb

%run 3_Comparator.ipynb
```

```
Input text: aabbaabbaabbabbabbaabbaabbaabaabaabba

    Size of the input text: 296 Bits

    Size of the compressed binary output: 216 Bits

Decoded text: aabbaabbaabbabbabbaabbaabbaabaabaabba

Comparison Result: Good
```


Input_Text_File.txt - Notepad
File Edit Format View Help
aabbaabbaabbabbabbaabbaabbaabaabaabba
Ln 14, Col 1
100%
Windows (CRLF)
UTF-8

main.cc | new 31 | new 32 | new 45 | new 43 | new 46 | new 47 | Compressed_version.bin

```
1  00000000011000010000000101100010000000000011000100000000101100001000000110110001000000100011000100000001101100001000001010110000100001000011000010000100101100010000001110110000100001011011000100000001100110001000000001
```


Decoded_Text_File.txt - Notepad
File Edit Format View Help
aabbaabbaabbabbabbaabbaabbaabaabaabba
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

Orchestrator : Example 2

This program is used to run the modules sequentially one after the other.

In [1]:
```
%run 1_Compression.ipynb

%run 2_Decompression.ipynb

%run 3_Comparator.ipynb
```

```
Input text: Who Am I? Why Am I? In philosophy, the Upanishads are Sanskrit t
exts with central concern to discover the relations between cosmic realitie
s, consciousness and the human body or person, postulating Atman and Brahman
as the summit of the interconnected universe. Brahman connotes the highest u
niversal principle, the Ultimate Reality of the universe. Atman is the true
or eternal Self or the self existent essence within each individual. The Bha
gavad Gita summarizes the key philosophies of the Upanishads with a major fo
cus on the philosophy of karma.

    Size of the input text: 4408 Bits

    Size of the compressed binary output: 3760 Bits

Decoded text: Who Am I? Why Am I? In philosophy, the Upanishads are Sanskrit
texts with central concern to discover the relations between cosmic realitie
s, consciousness and the human body or person, postulating Atman and Brahman
as the summit of the interconnected universe. Brahman connotes the highest u
niversal principle, the Ultimate Reality of the universe. Atman is the true
or eternal Self or the self existent essence within each individual. The Bha
gavad Gita summarizes the key philosophies of the Upanishads with a major fo
cus on the philosophy of karma.

Comparison Result: Good
```

main.cc | new 31 | new 32 | new 45 | new 43 | new 46 | new 47 | Input_Text_File.txt

```
1  Who Am I? Why Am I? In philosophy, the Upanishads are Sanskrit texts with central concern to discover the relations
   between cosmic realities, consciousness and the human body or person, postulating Atman and Brahman as the summit of
   the interconnected universe. Brahman connotes the highest universal principle, the Ultimate Reality of the universe.
   Atman is the true or eternal Self or the self existent essence within each individual. The Bhagavad Gita summarizes the
   key philosophies of the Upanishads with a major focus on the philosophy of karma.
```

Normal text file | length : 551 lines : 1 | Ln : 1 Col : 552 Pos : 552 | Windows (CR LF) | UTF-8 | INS

new 32 | new 45 | new 43 | new 46 | new 47 | Compressed_version.bin

1 000000000101011100000000001101000000000000001101111000000000001000000000000001000001000000000110110100000100010010010000000000111111000001000101011110000000100111100100000100010000010000011000100000000000000100100100001000001000000000110101101110000001000111000000000001001101001000000000110110000000001101110011000000110111000000000101000101100000000100011101000000000100110010100000100010101010000000000111000000000000011000010000000001101110000000000011010010000000000111001100000010011000010000000001100100000111010010000000001101001110010000000000110010100000100010100110001101001101111000011101011010110000000001110010000111000111010000010110011001010000000001111000000000000111010000100000011101110010011101101000000001000110001100100010011011100010101001110010000110100110110000101101011011110001101101100011001000100111001000011011001000000010101001101111000001000110010000011100011100110000000001100011000000110111011000110011001000000001010100110100000100010001000000001001100110010100010010011000010010101001101001000000110110111000100000011000100010001001110100000000000011101110010001001100101001101000110001100010011011011010001110001100011000001000111001000100010011000010001001001101001001111110110010100011101001011000011000101101110000111010110001100011100011011110000000001110101000111010110111000100010011100110010000001100001000110110110010000010110011010000011110001101000010100000110110100100100001000000000000000110001000000001101100100000000000111100100000100011011110010011000100000000011001011001010010011001110011010000000001011000001000001101111000111010111010001010000011011000000110100111010000011100011011100000000000110011100001011011101000000011001100010011010001100001010101000010000000000000100001000100110011000010000001001101101010110000110000100100000011101000001011100100000000111010111010100000110011011010010011101000000000000110110011001010101011001010000010001101001000110110111010000110011011000110100000001101110001000100110001100101010011001010001111100100000001010000011011100000111000111011000110011011100110010001000101110000000100010000100110110001101000011010000110111001001101011011100000001101111010001010010001000000011101101100101000001000110100000001110001100111000101110111001100101010001000000111110101101001000000000111011001111111101100001000100100010000000011001011100100110010101100011000111000111000000010010011001010000000000101100011101010010000000000000010101010100010010011101000001110001101101011001000110010100000100010100100100100101101100001001110111100101011100011001101001010101110101000110110110100110001101011001010101111110110010100000000000101110011001110110110101011000011010010110111101101000001111000111010000100110011101010011110001101111101011101011001010111101101110010000110110110000110001111010100110010001001101100000000000110011001011100011100101001010101110011101011010110011000000010001100101001010010110100101100010011001010111011100100000010100100111001100101110011000110011110001110111001011000110100100110100011001010001101001100011000000010001000000110010101100100011111100110100100011111011101010011000000101110000000100010101000111000001000010000111100110011100011010011101100001101001100100000000100010001110010011101100001000000100011100110101011101101101001000010110100100000000011111010100000110011101000111000000110101100100010011110010001000001101000000011100011011000000100110101111000110010110100000001110001100101001000000110111110101110001000000100001110010000001001011001110000000100100011010010001110101101000110001010111001100000010001110111001011000010000000011010001000000011010000110101000000001101110010000000100011001100000000110110001101010000011100110101110001101110100101010111000000001000101101100110100010111000000000101000100000011101000010000000000000001101011001000010110110100011010001011100

main.cc | new 31 | new 32 | new 45 | new 43 | new 46 | new 47 | Decoded_Text_File.txt

```
1  Who Am I? Why Am I? In philosophy, the Upanishads are Sanskrit texts with central concern to discover the relations
   between cosmic realities, consciousness and the human body or person, postulating Atman and Brahman as the summit of
   the interconnected universe. Brahman connotes the highest universal principle, the Ultimate Reality of the universe.
   Atman is the true or eternal Self or the self existent essence within each individual. The Bhagavad Gita summarizes the
   key philosophies of the Upanishads with a major focus on the philosophy of karma.
```

Normal text file | length : 551 lines : 1 | Ln : 1 Col : 1 Pos : 1 | Windows (CR LF) | UTF-8 | INS

Orchestrator : Example 3

This program is used to run the modules sequentially one after the other.

In [1]:
```
%run 1_Compression.ipynb

%run 2_Decompression.ipynb

%run 3_Comparator.ipynb
```

```
Input text: The quick brown fox jumps over the lazy dog.

    Size of the input text: 352 Bits

    Size of the compressed binary output: 512 Bits

Decoded text: The quick brown fox jumps over the lazy dog.

Comparison Result: Good
```

In this example, the text is a pangram, which contains all alphabets in English. There are no repeated phrases. So we are not able to compress this type of text using LZ algorithm.