Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Name: Varun Viswanath
SAPID: 60004210105
Batch: B1
Branch: Computer Engineering

Experiment No: 03

Aim: Write a program to implement Framing Techniques:
Character count, Byte stuffing, Bit stuffing

Theory:

Character count

Character count is a measure of the amount of data being transmitted between two devices in a computer network. In computer networks, data is transmitted in the form of packets, and each packet consists of a header and a payload. The payload contains the actual data being transmitted, and the character count is the number of characters in the payload.

For example, let's say we have two devices connected over a network, and one device wants to send a message to the other device. The message is "Hello, World!" and it is being transmitted as a packet. The payload of the packet is the string "Hello, World!", which contains 13 characters. Therefore, the character count for this packet is 13.

Character count is important in network communication because it helps to determine the bandwidth required for transmitting data between devices. It also helps to ensure that data is transmitted accurately and completely, as missing or corrupted characters can cause errors in the transmitted data.

Bit stuffing

Bit stuffing is a technique used in computer networks to ensure that data transmitted over the network is correctly interpreted by the receiving device, even if the data contains certain sequences that may be interpreted as control characters or special codes by the network hardware or software.

In bit stuffing, a special bit sequence is added to the data being transmitted to mark the start and end of the data payload. This special bit sequence is known as a "flag" and is typically composed of a specific bit pattern, such as "01111110". Whenever the transmitting device encounters a sequence of bits that matches the flag pattern within the data payload, it inserts an extra "0" bit immediately after the fifth consecutive "1" bit, so that the flag pattern is never encountered within the data payload.

For example, suppose a transmitting device wants to send the following data over a network:

`11001111011110111101001101001011010110 10`

To ensure that this data is transmitted correctly without any errors, the transmitting device can use bit stuffing by adding a flag pattern "01111110" at the beginning and end of the data payload, resulting in the following bit stream:

`01111110110011110111101111010011010010110101101001111111 0`

Byte stuffing

Byte stuffing, also known as character stuffing, is another technique used in computer networks to ensure that data transmitted over the network is correctly interpreted by the receiving device. Unlike bit stuffing, which operates at the bit level, byte stuffing operates at the byte level.

In byte stuffing, a special byte value is added to the data being transmitted to mark the start and end of the data payload. This special byte value is known as a "flag" and is typically composed of a specific byte pattern, such as "01111110". Whenever the transmitting device encounters a byte within the data payload that matches the flag pattern, it adds an extra byte known as the "escape character" immediately before the flag byte to differentiate it from the actual data.

At the receiving end, the receiving device detects the flag bytes and removes them from the data payload, along with any escape characters that were added by the transmitting device. This ensures that the original data is received correctly without any errors or misinterpretations caused by the presence of flag bytes or other control characters.

Implementation:

Code:

```
import PyPDF2
import random
pdfFileObj = open('Lorem_ipsum.pdf', 'rb')
pdfReader = PyPDF2.PdfReader(pdfFileObj)
pageObj = pdfReader.pages[0]
file_pdf = pageObj.extract_text()
pdfFileObj.close()
ascii_string = ""
binary_string = ""
for ch in file_pdf:
    ascii_string += str(ord(ch))
    binary_string += str(bin(ord(ch)))[2:]

with open('ascii.txt','w') as f:
    f.write(ascii_string)
with open('binary.txt','w') as f:
    f.write(binary_string)

#1. Character Count
start = 0
end = 0
packet_string = ""
while end<len(file_pdf):
    power = random.randint(1,8)
    end += 2**power
    packet = str(2**power)+file_pdf[start:end]
```

```
    start = end
    packet_string = packet_string + packet + "\n\n"


with open('char_count.txt','w') as f:
    f.write(packet_string)

#2. Bit Stuffing
with open('binary.txt','r') as f:
    binary = f.read()
    rem = len(binary)%128
    if(rem != 0):
        binary = binary + '1' + '0'*(128 - rem-1)
    bit_stuff = ""
    start = 0
    for bin in binary:
        if(len(bit_stuff) <= len(binary)):
            bit_stuff = bit_stuff + binary[start:start+8] + 'X'
            start = start + 9

with open('bit_stuff.txt','w') as f:
    f.write(bit_stuff)

#2. Byte Stuffing
with open('binary.txt','r') as f:
    binary = f.read()
    byte_stuff = ""
    start = 0
    end = 0
    while end<len(binary):
        power = random.randint(1,8)
        end += 2**power
        packet = binary[start:end]
        start = end
        byte_stuff = byte_stuff + packet + "\n"

with open('byte_stuff.txt','w') as f:
    f.write(byte_stuff)
```

Output:

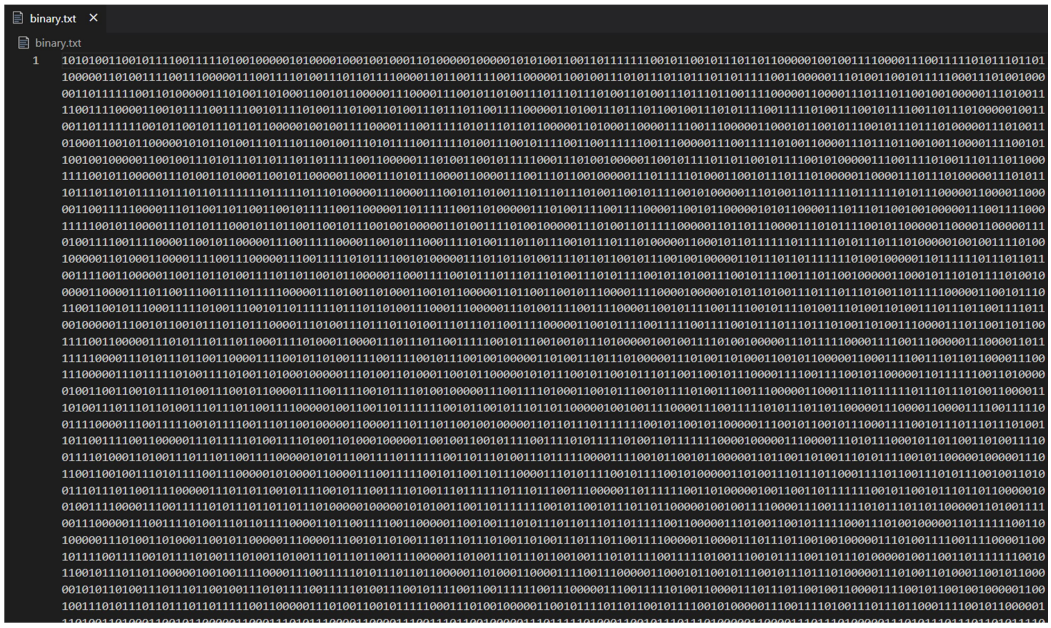
Character Count:

Ascii.txt

ascii.txt

```
841011151163280687032321076111114101109327311211517109321051153211510510911210812132100117109109121321161011201163211110232116104101321121141051101
```

Binary.txt



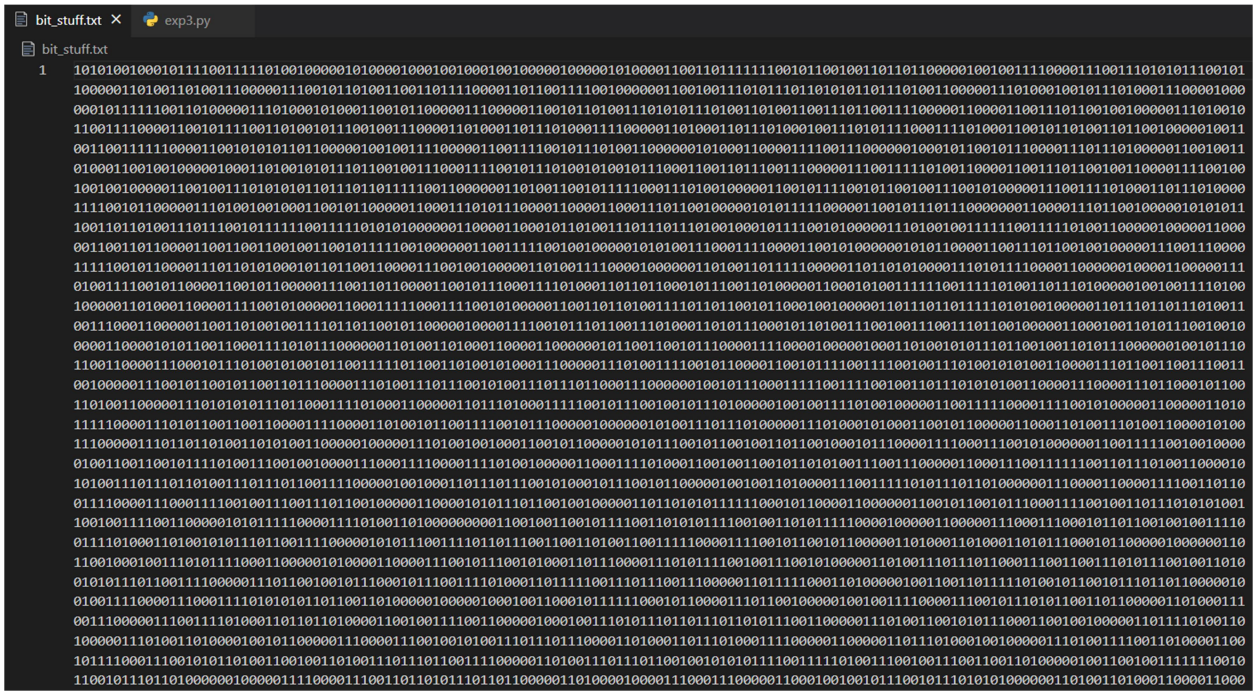
Char_count.txt

```
8Test PDF

8
Lorem

256 Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the
industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type
and scrambled it to make a type specimen book. It has sur vi

16ved not only fiv

32e centuries, but also the leap


128into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the
release of Letraset shee

16ts containing Lo

64rem Ipsum passages, and more recently with desktop publishing
s

4oftw

8are like

256 Aldus PageMaker including versions of Lorem Ipsum.
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the
industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type
a
```

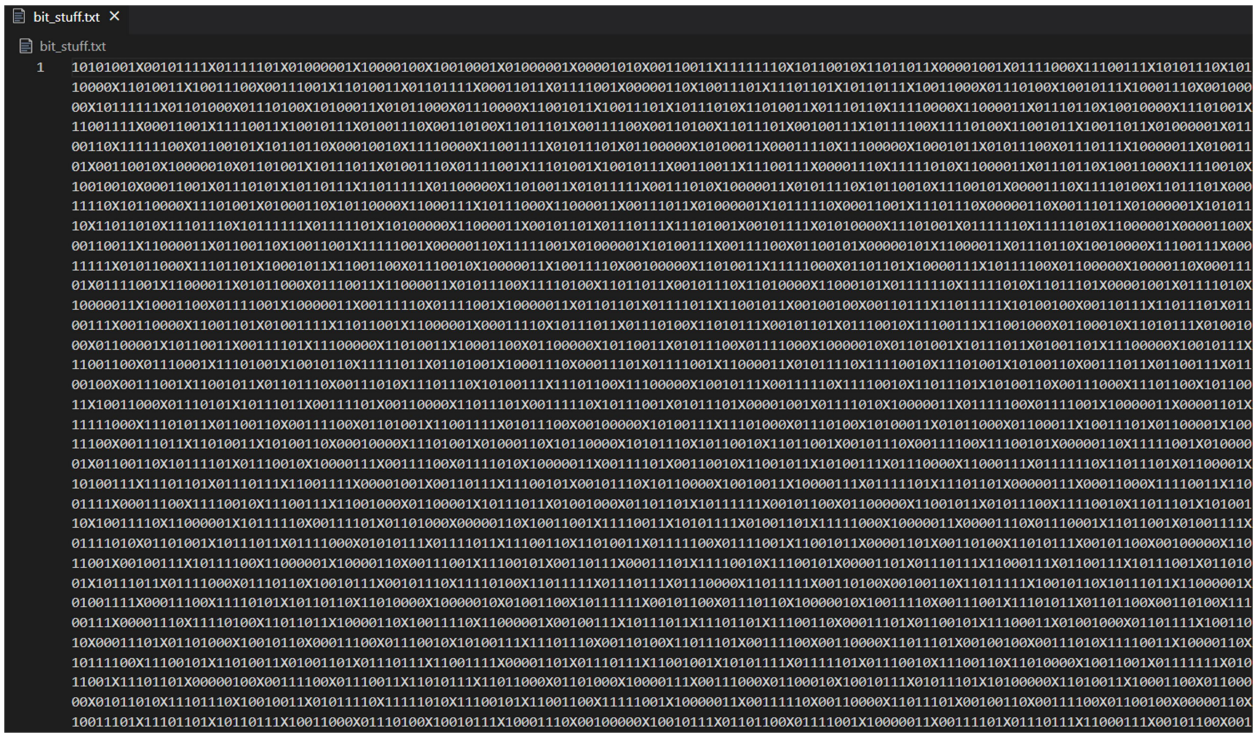
Bit Stuffing:

Bit_stuff.txt

Stuffing bit '0'



Stuffing bit 'X'



Byte Stuffing:

Byte_stuff.txt

byte_stuff.txt

```
1010100110010111
1001111101001000001010000100100
1000110100000100000101010011001101111111001011001011101101100000100100111100001110011111010111011011000001101001111001110000011100111101001110110111
1001101000001110100110100001100 10
1100
0001110000111001011010011101110111010011010011101110110011101000
01100001
1101110110010010000011101001111001110000110010111100111100101111010011010011101110111001111000001101001110110110010011101011110011111010011100
1000110000111100111000001100010 1
10
0101110010111011101000001110100110100011001011000001010110100111011101100100111010111001111101001110010111100110011111100111000
0011100111110100110000111011101100100110000111100101100100100000110010011101011101110111011101111001100000111010011001011111000111010010000011001011
00
0110000111001110110010000011101111101000110010111011101000001100
00
11
1011
1010
0000111010111011101101011110111011011111110111110111010000011100001110010110100111011101110100110010111100101000001110100110111
11011111
1010
1110000011000011000001100111110000111011001101100110010111100110000011011111100110100000111010011110011110000110010110000010101
10
0001
1101110110010010000011100111100011111001011000011101101110001011011001011100100100000110100111101001000001110100110111100000110110111000011101
0
1011
1011101000001100010110111111011111101011101110100000100100111101001000001101000110000111100111000001110011111010111100101000001110110110100111101101
10
0000
1100
0111100101110111
0111
0100111010111100101101001110010111100111011001000001100010111010111010010000011000011101100111001111011111000001110100110100011001011000001101100
1111110111011010011100011100000111010011110011110000110010111100111100101111010011101001101001110111011001111011001000011100101
```

Conclusion:

Different framing methods are implemented.