Assignment 2

Part 1: Theoretical Questions

1. Explain the Concept of Bit Stuffing and Its Necessity

Bit stuffing is a process used in bit-oriented protocols (e.g., HDLC) to ensure that special control sequences (such as flags) do not accidentally appear in the data. It involves inserting an extra `0` after a sequence of five consecutive `1`s in the data stream.

Necessity of Bit Stuffing in Communication Protocols:

✅ **Frame Synchronization** – Ensures that flag sequences are not confused with actual data.
✅ **Error Prevention** – Prevents accidental flag patterns in data.
✅ **Protocol Flexibility** – Allows arbitrary data transmission without constraint.

2. Difference Between Bit Stuffing and Byte Stuffing

Feature	Bit Stuffing	Byte Stuffing
Definition	Inserts extra bits to prevent false flag recognition.	Inserts special escape characters in a byte stream when control characters appear.
Used In	Bit-oriented protocols (HDLC, PPP).	Character-oriented protocols (BISYNC, DDCMP).
Example	Data: `1101111110` → Stuffed: `11011111010` (Extra `0` inserted)	Data: `ESC FLAG DATA` → Stuffed: `ESC ESC FLAG DATA` (Escape character added)

3. Bit Stuffing and Destuffing Example

Given Bit Stream:

`01111110 11011111011111010`
(Flag sequence: `011111`)

Bit Stuffing:

Original Data: `11011111011111010`

Step-by-step stuffing:

1. `11111` → Insert `0` after five `1`s.
2. Next `11111` → Insert another `0`.

Stuffed Bit Stream:
`1101111101011111010`

Bit Destuffing:

- Detect sequences of five `1`s followed by a `0`.
- Remove the extra `0` to recover original data.

Restored Data:
`110111111011111010`

4. Advantages and Disadvantages of Bit Stuffing

Advantages:

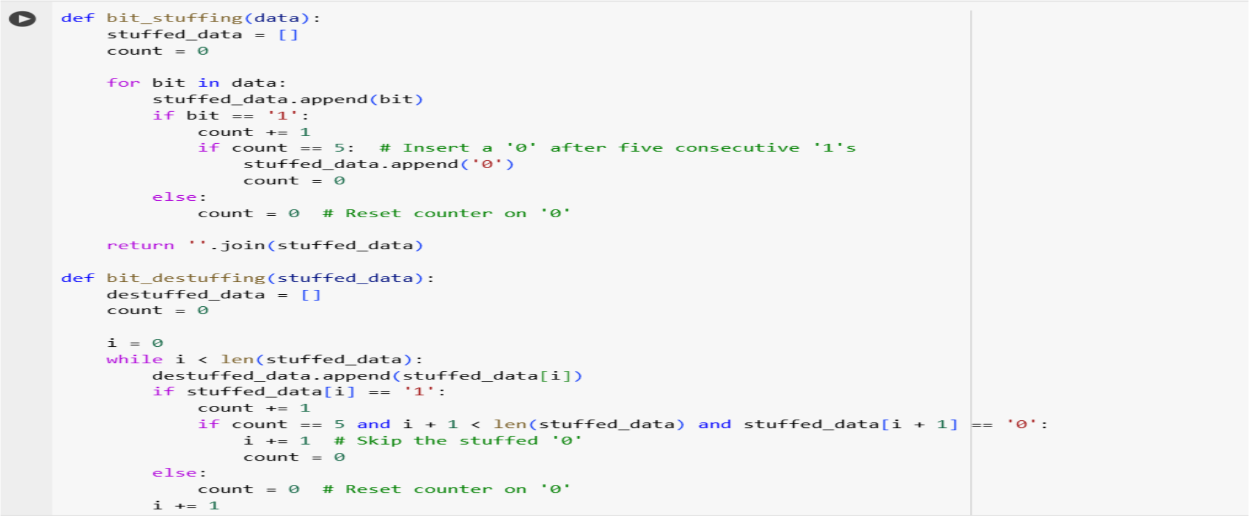
✅ Ensures frame synchronization.
✅ Prevents data from being misinterpreted as control information.
✅ Supports variable-length frames efficiently.

Disadvantages:

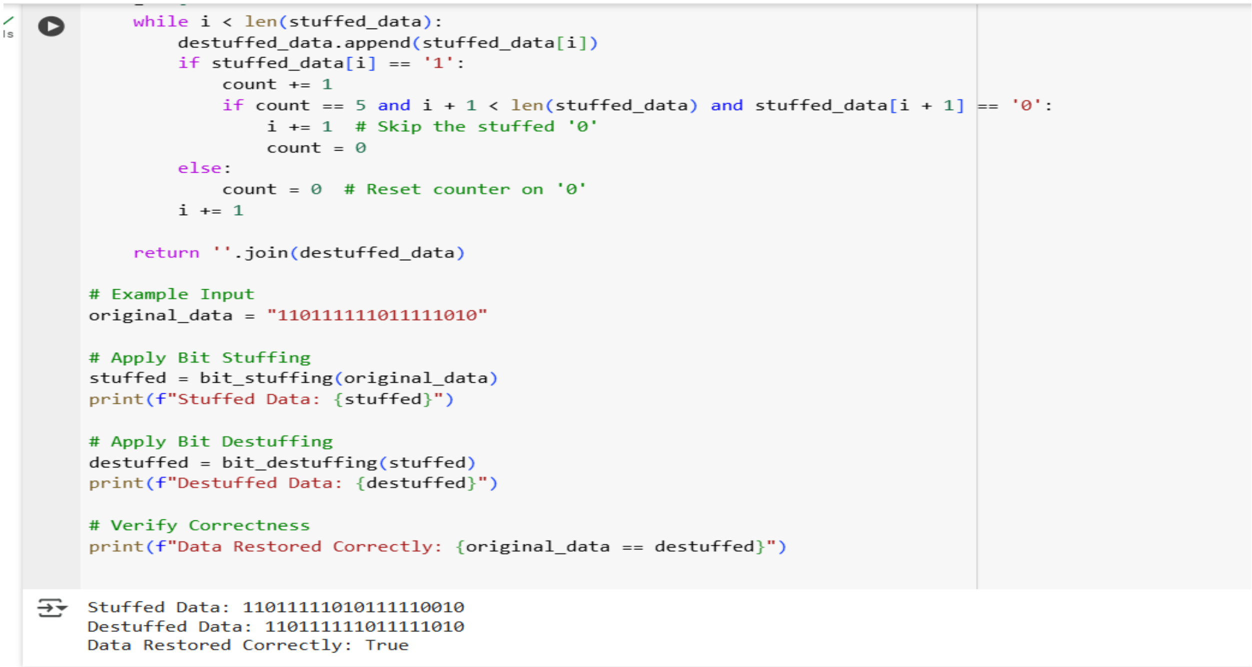
❌ Increases data size due to additional bits.
❌ Requires additional processing for stuffing/destuffing

❌ Can lead to bandwidth inefficiency if excessive stuffing occurs.

code

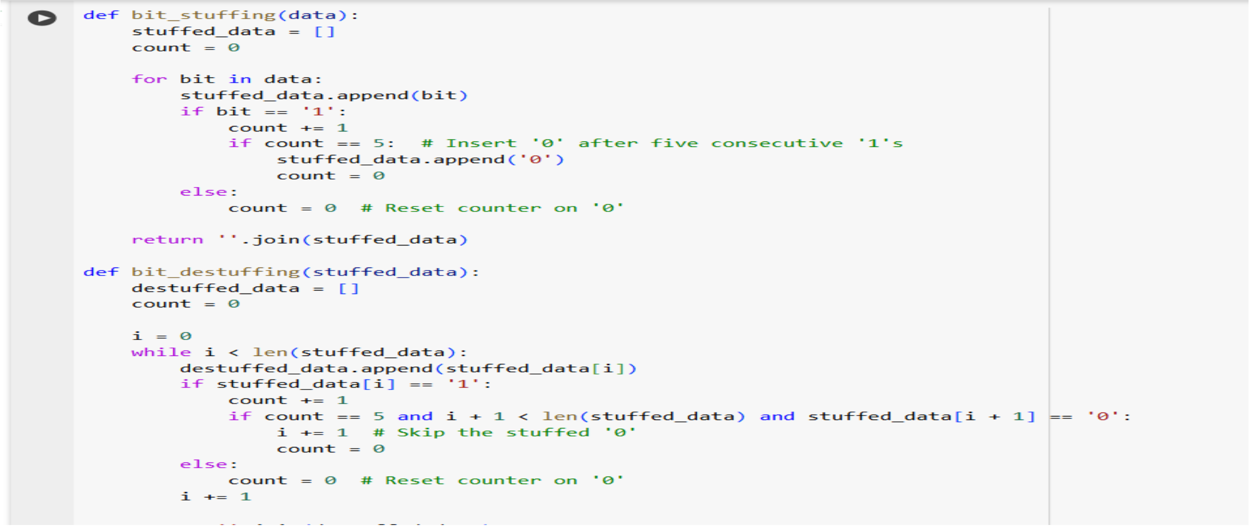


```python
def bit_stuffing(data):
    stuffed_data = []
    count = 0

    for bit in data:
        stuffed_data.append(bit)
        if bit == '1':
            count += 1
            if count == 5:  # Insert a '0' after five consecutive '1's
                stuffed_data.append('0')
                count = 0
        else:
            count = 0  # Reset counter on '0'

    return ''.join(stuffed_data)

def bit_destuffing(stuffed_data):
    destuffed_data = []
    count = 0

    i = 0
    while i < len(stuffed_data):
        destuffed_data.append(stuffed_data[i])
        if stuffed_data[i] == '1':
            count += 1
            if count == 5 and i + 1 < len(stuffed_data) and stuffed_data[i + 1] == '0':
                i += 1  # Skip the stuffed '0'
                count = 0
        else:
            count = 0  # Reset counter on '0'
        i += 1
```

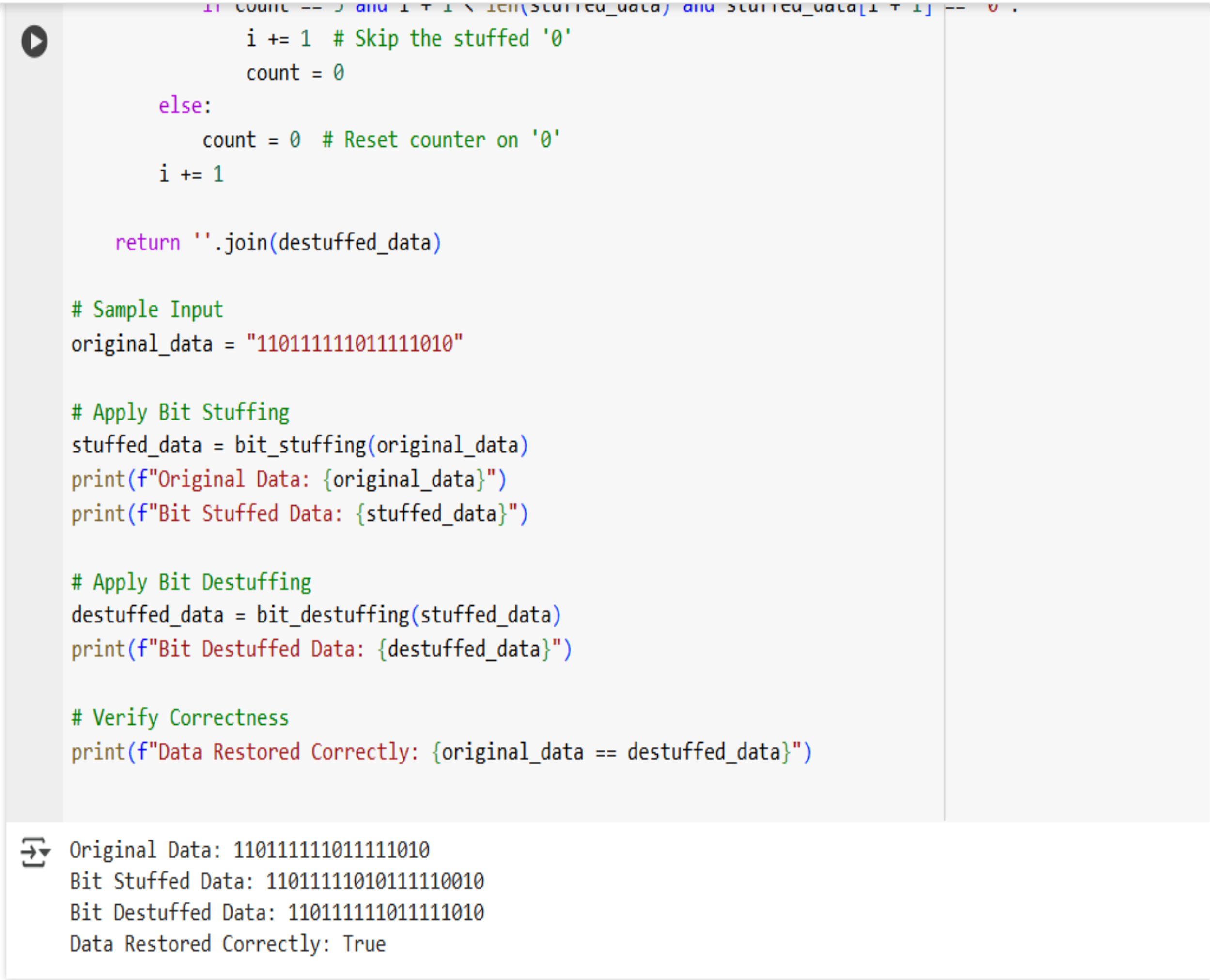


```python
    while i < len(stuffed_data):
        destuffed_data.append(stuffed_data[i])
        if stuffed_data[i] == '1':
            count += 1
            if count == 5 and i + 1 < len(stuffed_data) and stuffed_data[i + 1] == '0':
                i += 1  # Skip the stuffed '0'
                count = 0
        else:
            count = 0  # Reset counter on '0'
        i += 1

    return ''.join(destuffed_data)

# Example Input
original_data = "110111111011111010"

# Apply Bit Stuffing
stuffed = bit_stuffing(original_data)
print(f"Stuffed Data: {stuffed}")

# Apply Bit Destuffing
destuffed = bit_destuffing(stuffed)
print(f"Destuffed Data: {destuffed}")

# Verify Correctness
print(f"Data Restored Correctly: {original_data == destuffed}")
```

```
Stuffed Data: 11011111010111110010
Destuffed Data: 110111111011111010
Data Restored Correctly: True
```



```
def bit_stuffing(data):
    stuffed_data = []
    count = 0

    for bit in data:
        stuffed_data.append(bit)
        if bit == '1':
            count += 1
            if count == 5:  # Insert '0' after five consecutive '1's
                stuffed_data.append('0')
                count = 0
        else:
            count = 0  # Reset counter on '0'

    return ''.join(stuffed_data)

def bit_destuffing(stuffed_data):
    destuffed_data = []
    count = 0

    i = 0
    while i < len(stuffed_data):
        destuffed_data.append(stuffed_data[i])
        if stuffed_data[i] == '1':
            count += 1
            if count == 5 and i + 1 < len(stuffed_data) and stuffed_data[i + 1] == '0':
                i += 1  # Skip the stuffed '0'
                count = 0
        else:
            count = 0  # Reset counter on '0'
        i += 1
```



```
            if count == 5 and i + 1 < len(stuffed_data) and stuffed_data[i + 1] == '0':
                i += 1  # Skip the stuffed '0'
                count = 0
        else:
            count = 0  # Reset counter on '0'
        i += 1

    return ''.join(destuffed_data)

# Sample Input
original_data = "110111111011111010"

# Apply Bit Stuffing
stuffed_data = bit_stuffing(original_data)
print(f"Original Data: {original_data}")
print(f"Bit Stuffed Data: {stuffed_data}")

# Apply Bit Destuffing
destuffed_data = bit_destuffing(stuffed_data)
print(f"Bit Destuffed Data: {destuffed_data}")

# Verify Correctness
print(f"Data Restored Correctly: {original_data == destuffed_data}")
```

```
Original Data: 110111111011111010
Bit Stuffed Data: 11011111010111110010
Bit Destuffed Data: 110111111011111010
Data Restored Correctly: True
```