Green University of Bangladesh
Department of Computer Science and Engineering(CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year: 2025), B.Sc. in CSE (Day)

Lab Report NO 02

Course Title: Data Communication Lab

Course Code: CSE 308 Section: 223 D1

Student Details

Name	ID
Anisur Rahaman Maruf	222902078

Submission Date: 02/03/2025

Course Teacher's Name: Md. Samin Hossain Utsho

[For Teachers use only: Don't Write Anything inside this box]

Lab Report Status

Marks: …………………………………	Signature:.....................
Comments:...	Date:.............................

1. Title of the Experiment

Implementation of Bit Stuffing & De-Stuffing Algorithm in Java

2. Objectives/Aim

- To understand the concept of **Bit Stuffing** and **De-Stuffing** in data communication.
- To implement a **Bit Stuffing and De-Stuffing algorithm** in Java.
- To verify the correctness of the implemented algorithm with a sample input.
- To ensure that the transmitted data does not interfere with frame delimiters.

3. Procedure / Analysis / Design

3.1 Bit Stuffing Concept

Bit stuffing is a technique used in data transmission to prevent unintended control sequences within the actual data. It works as follows:

- The frame delimiter (header & trailer) is **01111110**.
- In the data, whenever five consecutive 1s appear, a 0 is inserted immediately after.
- This prevents confusion between actual data and frame delimiters.

3.2 Bit De-Stuffing Concept

Bit de-stuffing is the reverse process where the receiver removes the stuffed 0s to retrieve the original data:

- The receiver scans the data for five consecutive 1s.
- If a 0 appears after five 1s, it is removed to reconstruct the original message.
- The frame delimiter **01111110** is removed before processing the data.

3.3 Java Implementation

Below is the Java implementation of the **Bit Stuffing & De-Stuffing algorithm**:

Step 1: Implementing Bit Stuffing

```
public static String bitStuffing(String data) {
    StringBuilder stuffedData = new StringBuilder();
    int count = 0;

    for (int i = 0; i < data.length(); i++) {
        stuffedData.append(data.charAt(i));
        if (data.charAt(i) == '1') {
            count++;
            if (count == 5) {
                stuffedData.append("0"); // Insert '0' after five consecutive '1's
                count = 0;
            }
        } else {
            count = 0;
        }
    }
    return "01111110 " + stuffedData.toString() + " 01111110"; // Adding header & trailer
}
```

Step 2: Implementing Bit De-Stuffing

```
public static String bitDeStuffing(String stuffedData) {
    String data = stuffedData.substring(9, stuffedData.length() - 9); // Remove header & trailer
    StringBuilder deStuffedData = new StringBuilder();
    int count = 0;

    for (int i = 0; i < data.length(); i++) {
        if (data.charAt(i) == '1') {
            count++;
            deStuffedData.append("1");
            if (count == 5 && (i + 1) < data.length() && data.charAt(i + 1) == '0') {
                i++; // Skip the stuffed '0'
                count = 0;
            }
        } else {
            deStuffedData.append("0");
            count = 0;
        }
    }
    return "01111110 " + deStuffedData.toString() + " 01111110"; // Adding header & trailer
}
```

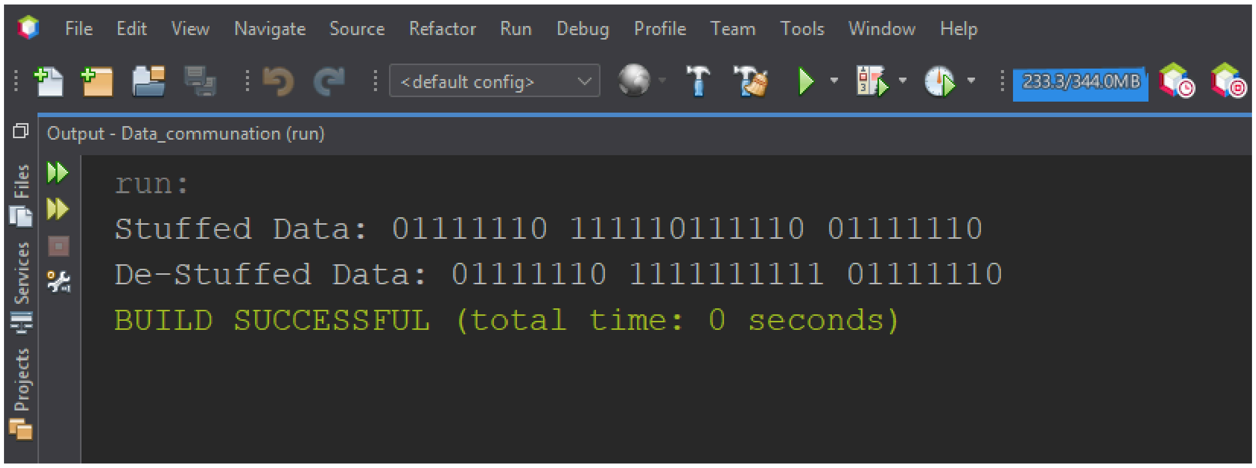
Step 3: Testing the Implementation

```
public static void main(String[] args) {
    String input = "1111111111"; // Sample input data

    String stuffed = bitStuffing(input);
    System.out.println("Stuffed Data: " + stuffed);

    String deStuffed = bitDeStuffing(stuffed);
    System.out.println("De-Stuffed Data: " + deStuffed);}
```

4. Output


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
233.3/344.0MB
Output - Data_communation (run)
Files
Services
Projects

```
run:
Stuffed Data: 01111110 111110111110 01111110
De-Stuffed Data: 01111110 1111111111 01111110
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Discussion

- **Why is bit stuffing needed?**
  - It ensures that frame delimiters do not appear within the actual data.
  - It maintains data integrity in protocols like **HDLC** and **PPP**.
- **What happens if bit stuffing is not used?**
  - If a sequence similar to 01111110 appears in data, the receiver may incorrectly detect it as a frame boundary, leading to transmission errors.
- **How does de-stuffing help?**
  - It reverses the bit stuffing process and restores the original data for correct interpretation.
- **Limitations:**
  - Bit stuffing increases the size of transmitted data.
  - It requires additional processing at both the sender and receiver.

6. Conclusion

In this experiment, we successfully implemented and tested **Bit Stuffing & De-Stuffing** in Java. We demonstrated how extra bits are inserted and removed to ensure error-free data transmission. This method plays a crucial role in data link layer protocols, ensuring smooth and reliable communication.