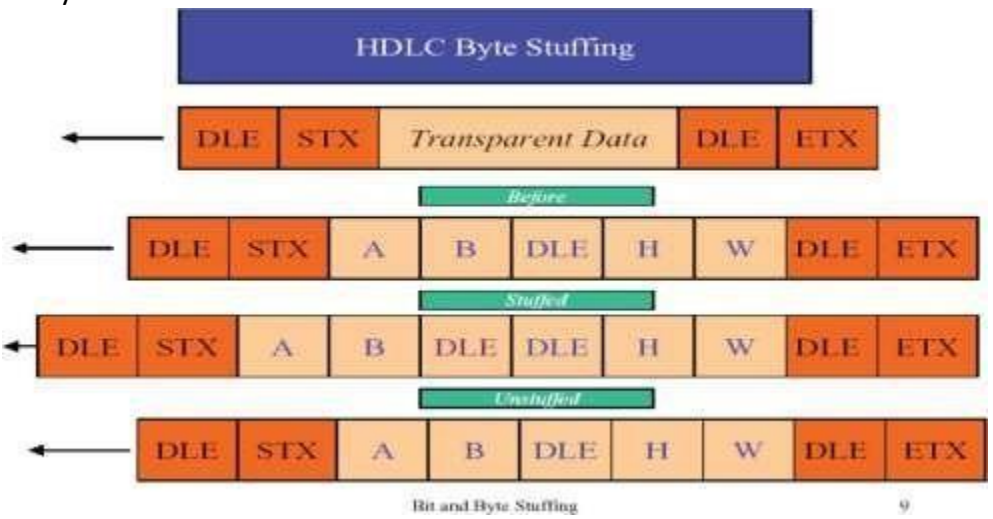
1(A) Program to demonstrate Character Stuffing

Aim: Implement the data link layer farming methods such as Character Stuffing and also De-stuff itusing Java Program.

Description: First framing method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count, it knows how many characters follow and hence where the end of the frame is. First empty box used for the header indicates character count.

Coming to the Character Stuffing, DLESTX and DLEETX are used to denote start and end of character data with some constraints imposed on repetition of characters as shown in the program clearly.



Conclusion:

Finally the following Character stuffing program executed successfully using Java and output generated without any errors.

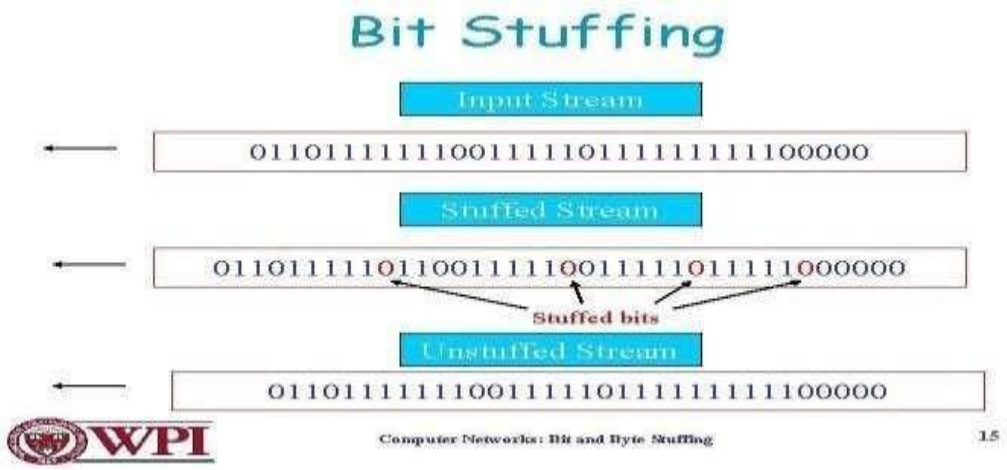
1(B)Program to Demonstrate Bit Stuffing

Aim: Implement the data link layer farming methods such as Bit Stuffing using Java Program.

Description: Security and Error detection are the most prominent features that are to be provided by any application which transfers data from one end to the other end.

One of such a mechanism in tracking errors which may add up to the original data during transfer is known as Stuffing. It is of two types namely Bit Stuffing and the other Character Stuffing.

Coming to the Bit Stuffing, 01111110 is appended within the original data while transfer of it. The program describes how it is stuffed at the sender end and de- stuffed at the receiver end.



Conclusion:

Finally the following bit stuffing program executed successfully using Java and output generated without any errors.

2.Program to implement the CRC polinomials in sender state and receiver state

Aim: TO demonstrate CRC polinomials in sender state and receiver state
Dscription: CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel.
CRC uses **Generator Polynomial** which is available on both sender and receiver side. An example generator polynomial is of the form like $x^3 + x + 1$. This generator polynomial represents key 1011. Another example is $x^2 + 1$ that represents key 101.

n : Number of bits in data to be sent
from sender side.
k : Number of bits in the key obtained
from generator polynomial.

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

1. The binary data is first augmented by adding k-1 zeros in the end of the data
2. Use ***modulo-2 binary division*** to divide binary data by the key and store remainder of division.
3. Append the remainder at the end of the data to form the encoded data and send the same

Receiver Side (Check if there are errors introduced in transmission)
Perform modulo-2 division again and if the remainder is 0, then there are no errors.
In this article we will focus only on finding the remainder i.e. check word and the code word.
Algorthm:
The CRC algorithm operates at both the sender and receiver sides during data transmission. Let's explore each step in detail:

1. **Sender Side (CRC Generator and Modulo Division)**

At the sender side, the CRC algorithm generates the CRC checksum and appends it to the data before transmitting it to the receiver. The steps involved are as follows:
Step 1: Augmentation - Append zeros to the data equal to the degree of the CRC polynomial minus one.
Step 2: Modulo Division - Perform polynomial division (modulo-2 division) on the augmented data with the CRC polynomial.
Step 3: Append the remainder (CRC checksum) to the original data.
Let's take another real-life example to understand this better.
Cyclic redundancy check example 2: CRC Generation
Suppose we have a 16-bit data packet: 1010101011001100. The CRC polynomial used is x^4 + x^3 + 1 (expressed as 11001 in binary).
Step 1: Augment the data with 3 zeros (degree of the polynomial is 4, so we append 4 - 1 = 3 zeros):
1010101011001100000
Step 2: Perform modulo-2 division with the CRC polynomial:

```
11001
----------------
1010101011001100000
11001
----------------
1010101011000001101 (Remainder: CRC Checksum)
```

Step 3: Append the CRC checksum to the original data:
1010101011001100101
Now, the data to be transmitted is 1010101011001100101, where the last 4 bits represent the CRC checksum.

2. **Receiver Side (Checking for Errors in the Received Data)**

On the receiver side, the CRC algorithm verifies the integrity of the received data. The process involves calculating the CRC checksum using the same polynomial and comparing it with the received checksum. If they match, the data is considered error-free; otherwise, errors are detected.
Step 1: Augmentation - Append zeros to the received data equal to the degree of the CRC polynomial minus one.
Step 2: Modulo Division - Perform polynomial division (modulo-2 division) on the augmented received data with the CRC polynomial.
Step 3: The data is error-free if the remainder obtained matches the received CRC checksum. Otherwise, errors are detected.
Let's continue with our previous example to see how error detection works.
3. Cyclic redundancy check error detection
Suppose during transmission, the data packet 1010101011001100101 is received with errors, resulting in 1010101011000100101.
Step 1: Augment the received data with 3 zeros:
101010101100010010100 0
Step 2: Perform modulo-2 division with the CRC polynomial:
11001

101010101100010010100 0
11001

101010101100000110101 0 (Remainder: Calculated CRC Checksum)
Step 3: Compare the calculated CRC checksum (1010) with the received CRC checksum (0101). As they don't match, errors are detected.

3.Program to demonstrate for implementataion of Data Link Layer Framing Method

Aim:To demonstarate of Data Link Layer Framing Method(character stuffing)
Description:
Character stuffing involves inserting special characters into the data stream to distinguish data from control information. This technique is primarily used in protocols like PPP (Point-to-Point Protocol) and HDLC (High-Level Data Link Control).

Process:

1. **Start and End Flags:** Frames are delimited by special flag characters (usually 01111110 in binary, or 0x7E in hexadecimal).
2. **Stuffing:** Whenever the data contains a byte identical to the flag character, an escape character (usually 0x7D) is inserted before it, followed by the original byte XORed with 0x20.
3. **Unstuffing:** At the receiver's end, the escape character is removed, and the subsequent byte is XORed with 0x20 to retrieve the original data.

Example:

- **Original Data:** A B FLAG C D
- **Stuffed Data:** A B ESC FLAG_ESC C D

Algorithm:
Sender Side:

1. **Initialize Frame:** Add the starting flag to the frame.
2. **Iterate through Data:**
    - For each byte in the data:
        - If the byte is the same as the flag character:
            - Add the escape character to the frame.
            - Add the byte XORed with 0x20 to the frame.
        - Else if the byte is the same as the escape character:
            - Add the escape character to the frame.
            - Add the byte XORed with 0x20 to the frame.
        - Else:
            - Add the byte directly to the frame.
3. **Finalize Frame:** Add the ending flag to the frame.
4. **Transmit Frame:** Send the frame over the communication medium.

Receiver Side:

1. **Read Frame:** Read the frame from the communication medium.
2. **Remove Flags:** Remove the starting and ending flags.
3. **Iterate through Data:**
    - For each byte in the frame:
        - If the byte is the escape character:
            - Read the next byte.
            - XOR the byte with 0x20 and add it to the data.
        - Else:
            - Add the byte directly to the data.

4.Program to demonstrate for implementation of the parity check bit

Aim:To demonstrate the Odd Even parity check bit

A **parity bit** is a bit added to a string of binary code to ensure that the total number of 1-bits in the string is even or odd. It is a simple form of error detection used in digital communications and storage.

Description:

1. **Even Parity**: The parity bit is set such that the total number of 1s in the code, including the parity bit, is even.
2. **Odd Parity**: The parity bit is set such that the total number of 1s in the code, including the parity bit, is odd.

Even Parity

Example:

- Data: 1011001
- Count of 1s: 4
- Since 4 is even, the parity bit for even parity is 0.
- Result: 10110010

Odd Parity

Example:

- Data: 1011001
- Count of 1s: 4
- Since 4 is even, the parity bit for odd parity is 1.
- Result: 10110011

Algorithm:

Adding Parity Bit

1. **Initialize Count**: Set a counter to 0.
2. **Count 1s**: Iterate through each bit in the data and count the number of 1s.
3. **Set Parity Bit**:
    - For even parity: If the count of 1s is even, set the parity bit to 0. If odd, set the parity bit to 1.
    - For odd parity: If the count of 1s is even, set the parity bit to 1. If odd, set the parity bit to 0.
4. **Append Parity Bit**: Append the parity bit to the data.
5. **Transmit Data**: Transmit the data along with the parity bit.

Checking Parity Bit

1. **Initialize Count**: Set a counter to 0.
2. **Count 1s**: Iterate through each bit in the received data (including the parity bit) and count the number of 1s.
3. **Check Parity**:
    - For even parity: If the count of 1s is even, the data is correct. If odd, an error is detected.
    - For odd parity: If the count of 1s is odd, the data is correct. If even, an error is detected.

5.Program to demonstrate for implementation of RSA algorithm

Aim:To demonstrate implementation of RSA algorithm

The RSA algorithm can handle strings by converting them into big integers. This allows for secure encryption and decryption of textual data.

Description:

1. Key Generation

- Select two large prime numbers, ppp and qqq.
- Compute n=p×qn = p \times qn=p×q. nnn is the modulus for both the public and private keys.
- Calculate the totient function ϕ(n)=(p−1)×(q−1)\phi(n) = (p-1) \times (q-1)ϕ(n)=(p−1)×(q−1).
- Choose an integer eee such that 1<e<ϕ(n)1 < e < \phi(n)1<e<ϕ(n) and gcd⁡(e,ϕ(n))=1\gcd(e, \phi(n)) = 1gcd(e,ϕ(n))=1. eee is the public exponent.
- Determine ddd as d≡e−1mod ⁡ϕ(n)d \equiv e^{-1} \mod \phi(n)d≡e−1modϕ(n). ddd is the private exponent.
- The public key is (e,n)(e, n)(e,n) and the private key is (d,n)(d, n)(d,n).

2. Encryption

- Convert the plaintext message MMM into an integer mmm such that 0≤m<n0 \leq m < n0≤m<n.
- Compute the ciphertext ccc as c≡memod ⁡nc \equiv m^e \mod nc≡memodn.

3. Decryption

- Compute the plaintext message mmm as m≡cdmod ⁡nm \equiv c^d \mod nm≡cdmodn.
- Convert mmm back into the original plaintext message MMM.

Algorithm

Key Generation Algorithm

1. **Choose Primes**:
    - Select two large prime numbers ppp and qqq.
2. **Compute nnn**:
    - n=p×qn = p \times qn=p×q.
3. **Compute Totient**:
    - ϕ(n)=(p−1)×(q−1)\phi(n) = (p-1) \times (q-1)ϕ(n)=(p−1)×(q−1).
4. **Choose eee**:
    - Select eee such that 1<e<ϕ(n)1 < e < \phi(n)1<e<ϕ(n) and gcd⁡(e,ϕ(n))=1\gcd(e, \phi(n)) = 1gcd(e,ϕ(n))=1.

5. **Compute ddd**:
    - Find ddd such that d×e≡1mod ϕ(n)d \times e \equiv 1 \mod \phi(n)d×e≡1modϕ(n).
6. **Public and Private Keys**:
    - The public key is (e,n)(e, n)(e,n) and the private key is (d,n)(d, n)(d,n).

Encryption Algorithm

1. **Convert Plaintext to Integer**:
    - Convert the plaintext message MMM to an integer mmm such that 0≤m<n0 \leq m < n0≤m<n.
    - This conversion can be done using a character encoding such as UTF-8.
2. **Encrypt**:
    - Compute the ciphertext ccc as c≡memod nc \equiv m^e \mod nc≡memodn.
3. **Result**:
    - The ciphertext ccc.

Decryption Algorithm

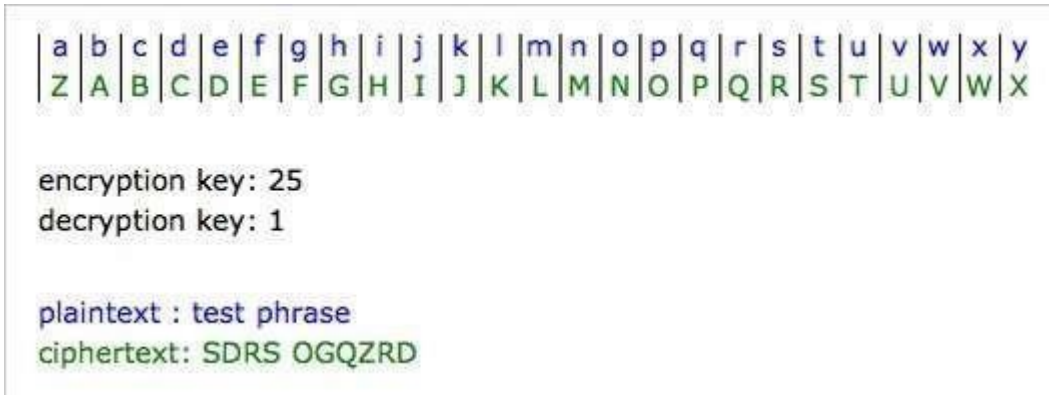
1. **Decrypt**:
    - Compute mmm as m≡cdmod nm \equiv c^d \mod nm≡cdmodn.
2. **Convert Integer to Plaintext**:
    - Convert the integer mmm back to the plaintext message MMM using the same encoding.
3. **Result**:
    - The plaintext message M.

6. Program to Demonstrate Playfair Substitution Cipher

Aim: Implement the data link layer farming methods such as Substitution cipher using Java Program.

Description: In a Substitution cipher, any character of plain text from the given fixed set of characters is substituted by some other character from the same set depending on a key. For example with a shift of 1, A would be replaced by B, B would become C, and so on

In substitution cipher the plain text is replaced by cipher text according to a fixed rule. There are two types of substitution cipher – Mono alphabetic and Poly alphabetic cipher.



Conclusion:

Finally the following Substitution cipher program executed successfully using Java and output generated without any errors.

7.Program to demonstrate Dijikstras algorithm

Aim: To demonstrate Dijikstras algorithm for finding the shortest path

Description:

Dijkstra's algorithm to find the shortest path from a single source vertex to all other vertices in the given graph. Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. We maintain two sets, one set contains vertices included in shortest path tree, and other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

Algorithm:

1. Create cost matrix C[ ][ ] from adjacency matrix adj[ ][ ]. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity.
2. Array visited[ ] is initialized to zero. for(i=0;i<n;i++) visited[i]=0;
3. If the vertex 0 is the source vertex then visited[0] is marked as 1.
4. Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex 0.

```
for(i=1;i<n;i++)
distance[i]=cost[0][i];
```

Initially, distance of source vertex is taken as 0. i.e.
distance[0]=0; 5. for(i=1;i<n;i++)

- Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.
- Recalculate the shortest distance of remaining vertices from the source.
- Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v

```
if(visited[v]==0)

distance[v]=min(distance[v], distance[w]+cost[w][v])
```

8.Program to Demonstrate Prim's algorithm for finding minimal spanning tree

Aim: To demonstrate prims algorithm for finding the minimum spanning tree

Description: Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a *Spanning* Tree. And they must be connected with the minimum weight edge to make it a *Minimum* Spanning Tree.

Algorithm:

1. Step 1: Select a starting vertex
2. Step 2: Repeat Steps 3 and 4 until there are fringe vertices
3. Step 3: Select an edge 'e' connecting the tree vertex and fringe vertex that has m inimum weight
4. Step 4: Add the selected edge and the vertex to the minimum spanning tree T
5. [END OF LOOP]
6. Step 5: EXIT

9. Program to Demonstrate Kruskal's Algorithm for minimal spanning tree

Aim : to demonstrate kruskal's algorithm for finding minimum spanning tree.

Description : Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph. The main target of the algorithm is to find the subset of edges by using which we can traverse every vertex of the graph.

Algorithm:

1. Step 1: Create a forest F in such a way that every vertex of the graph is a separa te tree.
2. Step 2: Create a set E that contains all the edges of the graph.
3. Step 3: Repeat Steps 4 and 5 **while** E is NOT EMPTY and F is not spanning
4. Step 4: Remove an edge from E with minimum weight
5. Step 5: IF the edge obtained in Step 4 connects two different trees, then add it to the forest F
6. (**for** combining two trees into one tree).
7. ELSE
8. Discard the edge
9. Step 6: END

10.Program to Demonstrate Fulkerson algorithm for finding the maximum flow

Aim: To demonstrate ford Fulkerson algorithm for finding the maximum flow.

Description: Given a graph which represents a flow network where every edge has a capacity. Also given two vertices *source* 's' and *sink* 't' in the graph, find the maximum possible flow from s to t with following constraints:

a) Flow on an edge doesn't exceed the given capacity of the edge.

b) Incoming flow is equal to outgoing flow for every vertex except s and t.

Algorithm:

FORD-FULKERSON METHOD (G, s, t)

1. Initialize flow f to 0
2. while there exists an augmenting path p
3. do argument flow f along p
4. Return

FORD-FULKERSON (G, s, t)

1. for each edge (u, v) ∈E [G]
2. do f *u, v+ ← 0
3. f *u, v+ ← 0
4. while there exists a path p from s to t in the residual network Gf.
5. do cf (p)←min?, Cf (u,v):(u,v)is on p}
6. for each edge (u, v) in p
7. do f *u, v+ ← f *u, v+ + cf (p) 8.

f [u, v] ←-f[u,v]

11.North West Corner method using Arrays

Aim: To demonstrate NWC method by using Arrays

Description: The **North-West Corner Rule** is a method adopted to compute the initial feasible solution of the transportation problem. The name North-west corner is given to this method because the basic variables are selected from the extreme left corner.

Algorithm:

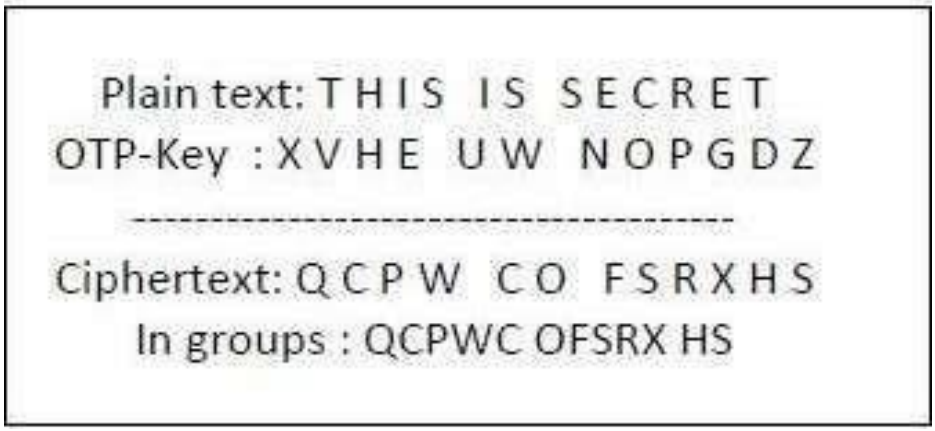
Step-1:	Select the upper left corner cell of the transportation matrix and allocate min(s1, d1).
Step-2:	a. Subtract this value from supply and demand of respective row and column. b. If the supply is 0, then cross (strike) that row and move down to the next cell. c. If the demand is 0, then cross (strike) that column and move right to the next cell. d. If supply and demand both are 0, then cross (strike) both row & column and move diagonally to the next cell.
Step-3:	Repeat this steps until all supply and demand values are 0

12. Program to Demonstrate One Time Pad

Aim: Implement the data link layer farming methods such as One Time Pad using Java Program.

Description: One-time pads are practical in situations where two parties in a secure environment must be able to depart from one another and communicate from two separate secure environments with perfect secrecy. The one-time-pad can be used in super encryption.

The primary use of padding with classical ciphers is to prevent the cryptanalyst from using that predictability to find known plaintext that aids in breaking the encryption. Random length padding also prevents an attacker from knowing the exact length of the plaintext message.



Conclusion:

Finally the following One Time Pad program executed successfully using Java and output generated without any errors.