Experiment-1

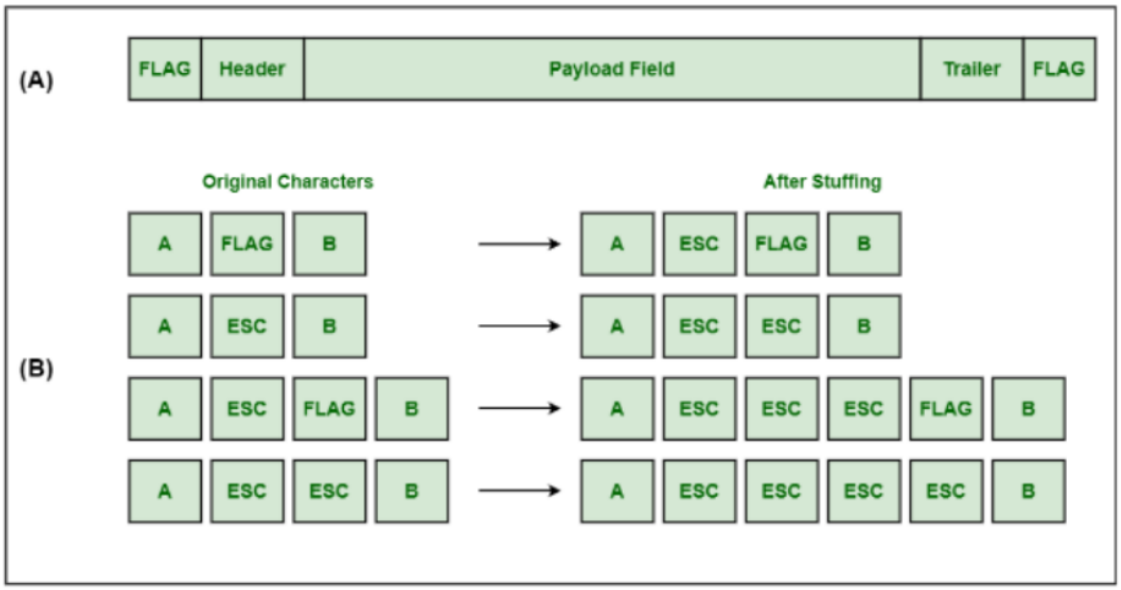
AIM: Write a Program to implement the data link layer farming methods such as

i) Character stuffing ii) bit stuffing.

i)Character Stuffing

Description: Character stuffing is also known as byte stuffing or character-oriented framing and is same as that of bit stuffing but byte stuffing actually operates on bytes whereas bit stuffing operates on bits. In byte stuffing, special byte that is basically known as ESC (Escape Character) that has predefined pattern is generally added to data section of the data stream or frame when there is message or character that has same pattern as that of flag byte.

Example:



Implement the data link layer framing methods such as character, character-stuffing and bit stuffing.

character-stuffing

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{
   int i = 0, j = 0, n, pos;
   char a[20], b[100], ch;
 printf("Enter the string:\n");
   scanf("%s", a);
  n = strlen(a);
printf("Enter position to insert the character:\n");
   scanf("%d", &pos);
 // Validate position
   while(pos < 1 || pos > n + 1)
   {
      printf("Invalid position, enter again: ");
      scanf("%d", &pos);
   }
getchar(); // Clear newline left by previous scanf
   printf("Enter the character to stuff:\n");
   ch = getchar();
```

```
    // Add starting frame delimiter
    b[0] = 'd'; b[1] = 'l'; b[2] = 'e'; b[3] = 's'; b[4] = 't'; b[5] = 'x';
    j = 6;
  for (i = 0; i < n; i++)
    {
        // Insert stuffing character at given position
        if (i == pos - 1)
        {
            b[j++] = 'd'; b[j++] = 'l'; b[j++] = 'e';
            b[j++] = ch;
            b[j++] = 'd'; b[j++] = 'l'; b[j++] = 'e';
        }
  // Escape 'dle' sequence
        if (a[i] == 'd' && a[i + 1] == 'l' && a[i + 2] == 'e')
        {
            b[j++] = 'd'; b[j++] = 'l'; b[j++] = 'e';
        }
b[j++] = a[i];
    }
   // If position is at the end
    if (pos == n + 1)
    {
        b[j++] = 'd'; b[j++] = 'l'; b[j++] = 'e';
        b[j++] = ch;
        b[j++] = 'd'; b[j++] = 'l'; b[j++] = 'e';
```

```
    }

    // Add ending frame delimiter
    b[j++] = 'd'; b[j++] = 'l'; b[j++] = 'e'; b[j++] = 'e'; b[j++] = 't'; b[j++] = 'x';
    b[j] = '\0';
printf("\nFrame after stuffing:\n%s\n", b);
}
```

Output:

```
$ gcc char_stuffing.c
$ ./a.out
Enter the string:
madhu
Enter position to insert the character:
2
Enter the character to stuff:
k
Frame after stuffing:
Dlestxmdlekdleadhudleetx
```

Bit Stuffing:

Program:

```
#include<stdio.h>

void main() {
    int ip_frame[100], op_frame[200];
    int i, j = 0, n;
    int count = 0;

    printf("Enter frame length: ");
    scanf("%d", &n);

    printf("Enter input frame (0's and 1's only):\n");
    for(i = 0; i < n; i++) {
        scanf("%d", &ip_frame[i]);
    }

    // Bit stuffing logic
    for(i = 0; i < n; i++) {
        op_frame[j++] = ip_frame[i];

        if(ip_frame[i] == 1) {
            count++;
            if(count == 5) {
                op_frame[j++] = 0;  // Stuff a 0 after five 1s
                count = 0;          // Reset count after stuffing
            }
        } else {
```

```
            count = 0;  // Reset count if a 0 is found
        }
    }

    printf("\nAfter bit stuffing, the frame is:\n");
    for(i = 0; i < j; i++) {
        printf("%d", op_frame[i]);
    }
    printf("\n");
}
```

Output:

```
$ gcc bit_stuffing.c
$ ./a.out
enter frame length: 9
Enter input frame (0's & 1's only): 010111111
After stuffing, the frame is:
0101111101
```

Experiment-2

AIM:

Write a program to compute CRC code for the polynomials CRC-12, CRC-16 and CRC CCIP

Description: CRC method can detect a single burst of length n, since only one bit per column will be changed, a burst of length n+1 will pass undetected, if the first bit is inverted, the last bit is inverted and all other bits are correct. If the block is badly garbled by a long burst or by multiple shorter burst, the probability that any of the n columns will have the correct parity that is 0.5. so the probability of a bad block being expected when it should not be 2 power(-n). This scheme sometimes is known as Cyclic Redundancy Code.

Program:

```
#include<stdio.h>
int main() {
    int data[100], div[20], temp[100];
    int datalen = 0, divlen = 0, i, j;
 char ch;
// Input data
    printf("Enter the data (binary): ");
    while ((ch = getchar()) != '\n') {
        if (ch == '1' || ch == '0') {
            data[datalen++] = ch - '0';
        }
    }
   // Input divisor
    printf("Enter the divisor (generator polynomial): ");
    while ((ch = getchar()) != '\n') {
```

```
        if (ch == '1' || ch == '0') {
            div[divlen++] = ch - '0';
        }
    }
// Copy data to temp and append zeros for CRC calculation
    for (i = 0; i < datalen; i++) {
        temp[i] = data[i];
    }
    for (i = 0; i < divlen - 1; i++) {
        temp[datalen + i] = 0;
    }
  int totalLen = datalen + divlen - 1;
    // CRC Division
    for (i = 0; i <= totalLen - divlen; i++) {
        if (temp[i] == 1) {
            for (j = 0; j < divlen; j++) {
                temp[i + j] = temp[i + j] ^ div[j];
            }
        }
    }
  // Append CRC (remainder) to data
    for (i = 0; i < datalen; i++) {
        printf("%d", data[i]);
    }
          printf(" (Data) + ");
```

```
        for (i = datalen; i < totalLen; i++) {
        printf("%d", temp[i]);
        data[i] = temp[i]; // Appending CRC bits
    }
         printf(" (CRC)\n");
    // Final transmitted data
    printf("Transmitted Data (Data + CRC): ");
    for (i = 0; i < totalLen; i++) {
        printf("%d", data[i]);
    }
    printf("\n");
return 0;
}
```

Output:

CRC-12

Enter the data (binary): 110100111011

Enter the divisor (generator polynomial): 1100000001111

CRC-16

Enter the data (binary): 11010011101100

Enter the divisor (generator polynomial): 11000000000000101

CRC-ccitt

Enter the data (binary): 1010101010101010

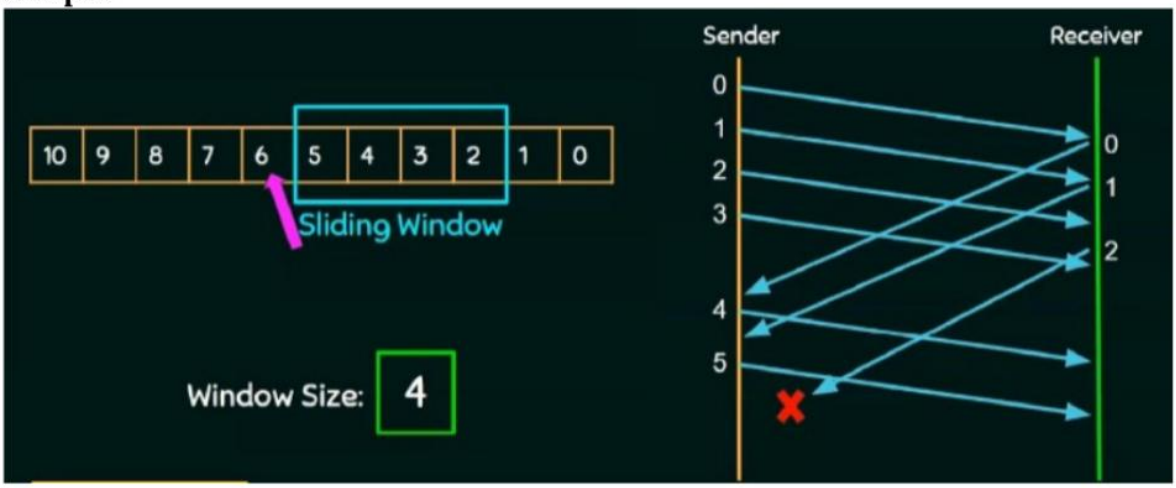
Enter the divisor (generator polynomial): 10001000000100001

Experiment-3

AIM: Develop a simple data link layer that performs the flow control using the sliding window protocol, loss recovery using the Go-Back-N mechanism.

Description: Go-Back-N protocol, also called Go-Back-N Automatic Repeat reQuest, is a data link layer protocol that uses a sliding window method for reliable and sequential delivery of data frames. It is a case of sliding window protocol having to send window size of N and receiving window size of 1. Go – Back – N ARQ provides for sending multiple frames before receiving the acknowledgment for the first frame. The frames are sequentially numbered and a finite number of frames. The maximum number of frames that can be sent depends upon the size ofthe sending window. If the acknowledgment of a frame is not received within an agreed upon time period, all frames starting from that frame are retransmitted.

Example:



Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_FRAMES 10
#define WINDOW_SIZE 4
#define LOSS_PROBABILITY 30  // Percentage (0-100)

int simulate_ack(int frame) {
```

```
    // Randomly decide if ACK is received or lost
    int random_value = rand() % 100;
    if (random_value < LOSS_PROBABILITY) {
        printf("ACK for Frame %d lost!\n", frame);
        return 0; // ACK lost
    }
    printf("ACK for Frame %d received.\n", frame);
    return 1; // ACK received
}

void sender(int total_frames) {
    int base = 0;
    int next_frame = 0;

    while (base < total_frames) {
        // Send frames in the window
        printf("\nSending Window: ");
        for (int i = 0; i < WINDOW_SIZE && next_frame < total_frames; i++) {
            printf("[Frame %d] ", next_frame);
            next_frame++;
        }
        printf("\n");

        // Simulate ACKs for sent frames
        int all_acks_received = 1;
        for (int i = base; i < base + WINDOW_SIZE && i < total_frames; i++) {
            if (!simulate_ack(i)) {
                all_acks_received = 0;
                // Go back to this frame
                printf("Go-Back-N triggered! Retransmitting from Frame %d\n", i);
                next_frame = i;
```

```
                break;
            }
        }

        // Move base forward if all ACKs received
        if (all_acks_received) {
            base += WINDOW_SIZE;
        }
    }

    printf("\nAll frames transmitted successfully.\n");
}

int main() {
    srand(time(0)); // Seed for randomness
    int total_frames;

    printf("Enter total number of frames to send (<= %d): ", MAX_FRAMES);
    scanf("%d", &total_frames);

    if (total_frames <= 0 || total_frames > MAX_FRAMES) {
        printf("Invalid number of frames.\n");
        return 1;
    }

    sender(total_frames);
    return 0;
}
```

Output:

```
Enter total number of frames to send (<= 10): 6
```

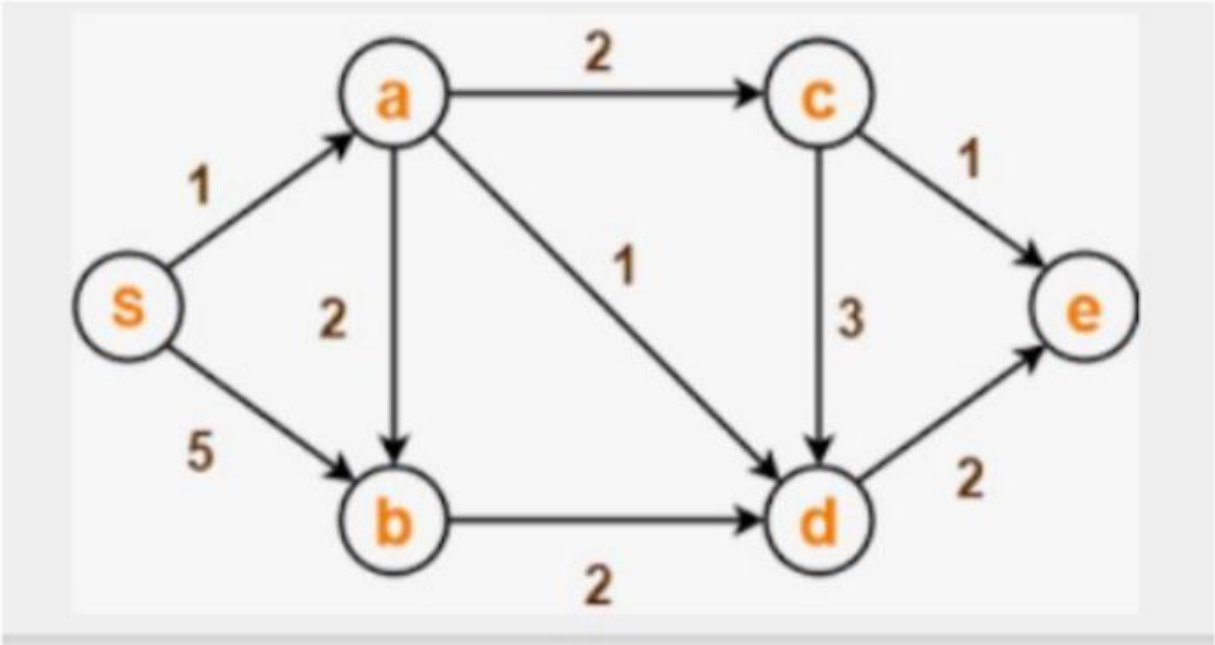
```
Sending Window: [Frame 0] [Frame 1] [Frame 2] [Frame 3]
ACK for Frame 0 received.
ACK for Frame 1 received.
ACK for Frame 2 lost!
Go-Back-N triggered! Retransmitting from Frame 2

Sending Window: [Frame 2] [Frame 3] [Frame 4] [Frame 5]
ACK for Frame 2 received.
ACK for Frame 3 received.
ACK for Frame 4 received.
ACK for Frame 5 received.

All frames transmitted successfully.
```

Result:

Experiment-4

AIM: To implement Dijsktra's algorithm to compute the shortest path through a network

Description: The Dijkstra's algorithm finds the shortest path from a particular node, called the source node to every other node in a connected graph. It produces a shortest path tree with the source node as the root. It is profoundly used in computer networks to generate optimal routes with the aim of minimizing routing costs.
Input − A graph representing the network; and a source node, s
Output − A shortest path tree, spt[], with s as the root node.



Program:

```
#include<stdio.h>
#define INFINITY 9999
#define MAX 10
// Function declaration
void dijkstra(int graph[MAX][MAX], int numVertices, int startNode);
int main()
 {
   int graph[MAX][MAX], numVertices, i, j, startNode;
        // Input number of vertices
   printf("Enter the number of vertices: ");
   scanf("%d", &numVertices);
// Input adjacency matrix
```

```
    printf("\nEnter the adjacency matrix:\n");
    for (i = 0; i < numVertices; i++) {
        for (j = 0; j < numVertices; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    // Input the starting node
    printf("\nEnter the starting node (0 to %d): ", numVertices - 1);
    scanf("%d", &startNode);
// Run Dijkstra's algorithm
    dijkstra(graph, numVertices, startNode);
         return 0;
}


// Dijkstra's algorithm implementation
void dijkstra(int graph[MAX][MAX], int numVertices, int startNode) {
    int cost[MAX][MAX];
    int distance[MAX];       // Shortest distances from startNode
    int predecessor[MAX];    // To store the shortest path tree
    int visited[MAX];        // To mark visited nodes
 int count, minDistance, nextNode, i, j;
    // Create cost matrix (replace 0 with INFINITY, except on diagonal)
    for (i = 0; i < numVertices; i++) {
```

```
        for (j = 0; j < numVertices; j++) {
            if (graph[i][j] == 0 && i != j)ṁ;,,.; ./
                cost[i][j] = INFINITY;
            else
                cost[i][j] = graph[i][j];
        }
    }
// Initialize distances, predecessors, and visited array
    for (i = 0; i < numVertices; i++) {
        distance[i] = cost[startNode][i];
        predecessor[i] = startNode;
        visited[i] = 0;
    }
    distance[startNode] = 0;
    visited[startNode] = 1;
    count = 1;

    // Find shortest path for all vertices
    while (count < numVertices - 1) {
        minDistance = INFINITY;
// Find the next node with the smallest tentative distance
        for (i = 0; i < numVertices; i++) {
            if (!visited[i] && distance[i] < minDistance) {
                minDistance = distance[i];
                nextNode = i;
```

```
            }
        }
 visited[nextNode] = 1;
// Update distances of neighboring unvisited nodes
        for (i = 0; i < numVertices; i++) {
            if (!visited[i] && (minDistance + cost[nextNode][i] < distance[i])) {
                distance[i] = minDistance + cost[nextNode][i];
                predecessor[i] = nextNode;
            }
        }
 count++;
    }
    // Print the shortest distance and path from startNode to each other node
    for (i = 0; i < numVertices; i++) {
        if (i != startNode) {
            printf("\nDistance from node %d to node %d = %d", startNode, i,
distance[i]);
            printf("\nPath: %d", i);
          j = i;
            while (j != startNode) {
                j = predecessor[j];
                printf(" <- %d", j);
            }
            printf("\n");
        }
    }
```

```
}
```

Output:

```
$ gcc dijsktras.c
$ ./a.out
Enter the number of vertices: 5
Enter the adjacency matrix:
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
Enter the starting node (0 to 4): 0
Distance from node 0 to node 1 = 2
Path: 1 <- 0
Distance from node 0 to node 2 = 5
Path: 2 <- 1 <- 0

Distance from node 0 to node 3 = 6
Path: 3 <- 0
Distance from node 0 to node 4 = 7
Path: 4 <- 1 <- 0
```

Result:

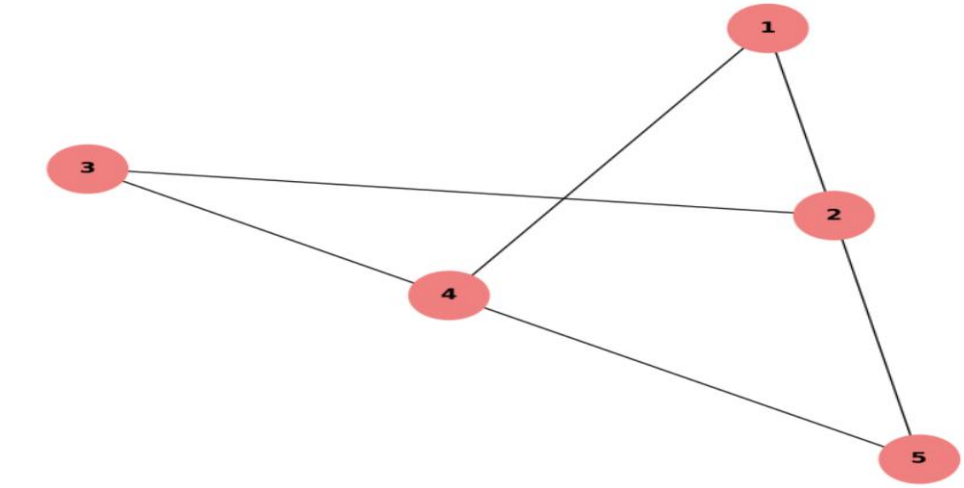
Experiment-5

Take an example subnet of hosts and obtain a broadcast tree for the subnet.

AIM:

Description:

Input Graph:



Program:

```
#include<stdio.h>
#define MAX 10
int adjacencyMatrix[MAX][MAX];
int numNodes;
// Function to display adjacent nodes of a given node
void displayAdjacentNodes(int rootNode) {
```

```
    int i;
printf("\nAdjacent nodes of node %d:\n", rootNode);
    for (i = 1; i <= numNodes; i++) {
        if (adjacencyMatrix[rootNode][i] == 1 || adjacencyMatrix[i][rootNode] == 1) {
            printf("%d\t", i);
        }
    }

    printf("\n");
}

int main() {
    int i, j, rootNode;
 // Input number of nodes
    printf("Enter number of nodes: ");
    scanf("%d", &numNodes);
// Input the adjacency matrix
    printf("Enter adjacency matrix:\n");
    for (i = 1; i <= numNodes; i++) {
        for (j = 1; j <= numNodes; j++) {
            printf("Is there a connection from node %d to node %d (1=yes, 0=no): ", i, j);
            scanf("%d", &adjacencyMatrix[i][j]);
        }
    }
 // Input the root node
    printf("Enter root node (1 to %d): ", numNodes);
    scanf("%d", &rootNode);
    // Display adjacent nodes
```

```
    displayAdjacentNodes(rootNode);

    return 0;
}
```

Output:

```
$ gcc broadcast.c
$ ./a.out
Enter number of nodes: 5
Enter adjacency matrix:
Is there a connection from node 1 to node 1 (1=yes, 0=no): 0
Is there a connection from node 1 to node 2 (1=yes, 0=no): 1
Is there a connection from node 1 to node 3 (1=yes, 0=no): 0
Is there a connection from node 1 to node 4 (1=yes, 0=no): 1
Is there a connection from node 1 to node 5 (1=yes, 0=no): 0
Is there a connection from node 2 to node 1 (1=yes, 0=no): 1
Is there a connection from node 2 to node 2 (1=yes, 0=no): 0
Is there a connection from node 2 to node 3 (1=yes, 0=no): 1
Is there a connection from node 2 to node 4 (1=yes, 0=no): 0
Is there a connection from node 2 to node 5 (1=yes, 0=no): 1
Is there a connection from node 3 to node 1 (1=yes, 0=no): 0
Is there a connection from node 3 to node 2 (1=yes, 0=no): 1
Is there a connection from node 3 to node 3 (1=yes, 0=no): 0
Is there a connection from node 3 to node 4 (1=yes, 0=no): 1
Is there a connection from node 3 to node 5 (1=yes, 0=no): 0
Is there a connection from node 4 to node 1 (1=yes, 0=no): 1
Is there a connection from node 4 to node 2 (1=yes, 0=no): 0
Is there a connection from node 4 to node 3 (1=yes, 0=no): 1
Is there a connection from node 4 to node 4 (1=yes, 0=no): 0
```

Is there a connection from node 4 to node 5 (1=yes, 0=no): 1

Is there a connection from node 5 to node 1 (1=yes, 0=no): 0

Is there a connection from node 5 to node 2 (1=yes, 0=no): 1

Is there a connection from node 5 to node 3 (1=yes, 0=no): 0

Is there a connection from node 5 to node 4 (1=yes, 0=no): 1

Is there a connection from node 5 to node 5 (1=yes, 0=no): 0

Enter root node (1 to 5): 4

Adjacent nodes of node 4:

1 3 5

Result:

Experiment-6

Implement data encryption and data decryption

AIM: To Implement data encryption and data decryption

Description:

Program:

```
#include<stdio.h>
#include <string.h>
#include <math.h>

// Function to perform modular exponentiation: (base^exp) % mod
int modular_exponentiation(unsigned int base, unsigned int exp, unsigned int mod)
{
   unsigned long int result = 1;
   for (int i = 1; i <= exp; i++) {
      result = (result * base) % mod;
   }
   return (unsigned int) result;
}

void main() {
   char message[100];
   unsigned int plaintext[100], ciphertext[100];
   unsigned int n = 253;      // Modulus (product of two primes: p * q)
   unsigned int e = 13;       // Public exponent
```

```
unsigned int d = 17;       // Private exponent
int i;


// Input plaintext
printf("Enter the plain text: ");
fgets(message, sizeof(message), stdin);

// Strip newline
size_t len = strlen(message);
if (len > 0 && message[len - 1] == '\n') {
    message[len - 1] = '\0';
}


// Convert characters to ASCII integers
for (i = 0; i < strlen(message); i++) {
    plaintext[i] = (unsigned int) message[i];
}

// Encrypt each character using RSA encryption: ct = pt^e mod n
printf("\nEncrypted Cipher Text (Numeric): ");
       for (i = 0; i < strlen(message); i++) {
    ciphertext[i] = modular_exponentiation(plaintext[i], e, n);
    printf("%d ", ciphertext[i]);
```

```
    }

    // Print original plain text
    printf("\nOriginal Plain Text: ");
    for (i = 0; i < strlen(message); i++) {
        printf("%c", plaintext[i]);
    }

    // Decrypt each character using RSA decryption: pt = ct^d mod n
    printf("\nDecrypted Plain Text: ");
    for (i = 0; i < strlen(message); i++) {
        plaintext[i] = modular_exponentiation(ciphertext[i], d, n);
        printf("%c", plaintext[i]);
    }


}
```

Output:

$ gcc encrypt_decrypt.c

$ ./a.out

Enter the plain text: StudyGlance

Encrypted Cipher Text (Numeric): 172 139 211 133 220 4 3 113 209 66 173

Original Plain Text: StudyGlance

Decrypted Plain Text: StudyGlance

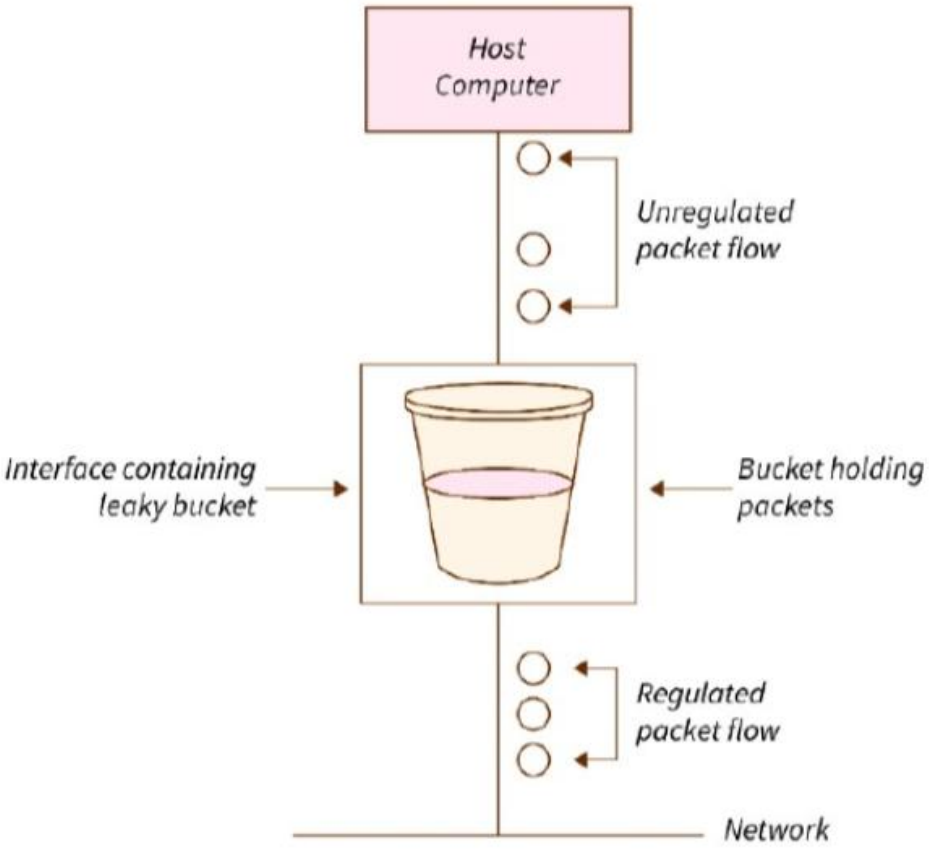
Result:

Experiment-7

Write a program for congestion control using leaky bucket algorithm.

AIM: Algorithm to implement congestion control using leaky bucket algorithm.

Description: The leaky bucket algorithm is a method of congestion control where multiple packets are stored temporarily. These packets are sent to the network at a constant rate that is decided between the sender and the network. This algorithm is used to implement congestion control through traffic shaping in data networks.



Program:

```
#include<stdio.h>

#include <stdlib.h>

#include <unistd.h>
```

```
#define NUM_PACKETS 10

// Custom random function to ensure non-zero result
int get_random(int max_value) {
   int result = (random() % 10) % max_value;
   return result == 0 ? 1 : result;
}

int main() {
   int packet_size[NUM_PACKETS];
   int i, clock_tick, bucket_size, output_rate;
   int remaining_data = 0, data_to_send, transmission_time, transmitted;

   // Generate random packet sizes
   for (i = 0; i < NUM_PACKETS; ++i)
      packet_size[i] = get_random(6) * 10;

   // Display the generated packet sizes
   for (i = 0; i < NUM_PACKETS; ++i)
      printf("\nPacket[%d]: %d bytes", i + 1, packet_size[i]);

   // Get output rate and bucket size from the user
   printf("\n\nEnter the Output Rate (bytes/unit time): ");
   scanf("%d", &output_rate);

   printf("Enter the Bucket Size (in bytes): ");
   scanf("%d", &bucket_size);
```

```
  // Process each incoming packet
  for (i = 0; i < NUM_PACKETS; ++i) {
    // Check if incoming packet + remaining data exceeds bucket size
    if ((packet_size[i] + remaining_data) > bucket_size) {
      if (packet_size[i] > bucket_size) {
        printf("\n\nIncoming Packet[%d] size (%d bytes) exceeds bucket capacity (%d bytes) - PACKET REJECTED!", i + 1, packet_size[i], bucket_size);
      } else {
        printf("\n\nBucket capacity exceeded for Packet[%d] - PACKET REJECTED!", i + 1);
      }
    } else {
      // Accept and add packet to bucket
      remaining_data += packet_size[i];
      printf("\n\nIncoming Packet[%d] size: %d bytes", i + 1, packet_size[i]);
      printf("\nTotal bytes to transmit (in bucket): %d", remaining_data);

      // Simulated transmission time (random)
      transmission_time = get_random(4) * 10;
      printf("\nEstimated transmission time: %d units", transmission_time);

      // Transmit data in units of time
      for (clock_tick = 10; clock_tick <= transmission_time; clock_tick += 10) {
        sleep(1);  // Simulate delay (1 second per 10 units)

        if (remaining_data > 0) {
          if (remaining_data <= output_rate) {
            transmitted = remaining_data;
            remaining_data = 0;
          } else {
```

```
                transmitted = output_rate;
                remaining_data -= output_rate;
            }

            printf("\nTransmitted %d bytes ---- Remaining in bucket: %d bytes", transmitted,
remaining_data);
        } else {
            printf("\nNo data to transmit at time unit %d!", clock_tick);
        }
      }
    }
  }

  return 0;
}
```

Output:

```
$ gcc leakybucket.c
$ ./a.out

Packet[1]: 30 bytes
Packet[2]: 10 bytes
Packet[3]: 10 bytes
Packet[4]: 50 bytes
Packet[5]: 30 bytes
Packet[6]: 50 bytes
```

Packet[7]: 10 bytes

Packet[8]: 20 bytes

Packet[9]: 30 bytes

Packet[10]: 10 bytes

Enter the Output Rate (bytes/unit time): 10

Enter the Bucket Size (in bytes): 15

Incoming Packet[1] size (30 bytes) exceeds bucket capacity (15 bytes) - PACKET REJECTED!

Incoming Packet[2] size: 10 bytes

Total bytes to transmit (in bucket): 10

Estimated transmission time: 20 units

Transmitted 10 bytes ---- Remaining in bucket: 0 bytes

No data to transmit at time unit 20!

Incoming Packet[3] size: 10 bytes

Total bytes to transmit (in bucket): 10

Estimated transmission time: 30 units

Transmitted 10 bytes ---- Remaining in bucket: 0 bytes

No data to transmit at time unit 20!

No data to transmit at time unit 30!

Incoming Packet[4] size (50 bytes) exceeds bucket capacity (15 bytes) - PACKET REJECTED!

Incoming Packet[5] size (30 bytes) exceeds bucket capacity (15 bytes) - PACKET REJECTED!

Incoming Packet[6] size (50 bytes) exceeds bucket capacity (15 bytes) - PACKET REJECTED!

Incoming Packet[7] size: 10 bytes

Total bytes to transmit (in bucket): 10

Estimated transmission time: 10 units

Transmitted 10 bytes ---- Remaining in bucket: 0 bytes

Incoming Packet[8] size (20 bytes) exceeds bucket capacity (15 bytes) - PACKET REJECTED!

Incoming Packet[9] size (30 bytes) exceeds bucket capacity (15 bytes) - PACKET REJECTED!

Incoming Packet[10] size: 10 bytes

Total bytes to transmit (in bucket): 10

Estimated transmission time: 10 units

Transmitted 10 bytes ---- Remaining in bucket: 0 bytes

Result:

Experiment-7

Write a program for frame sorting techniques used in buffers.

AIM: To implement

Description:

Program:

```
#include<stdio.h>
#include <stdlib.h>
#include <string.h>

#define FRAME_TEXT_SIZE 3      // Each frame holds 3 characters
#define MAX_NO_OF_FRAMES 127   // Maximum number of frames supported

char inputMessage[FRAME_TEXT_SIZE * MAX_NO_OF_FRAMES]; // Buffer to store the original message

// Frame structure to hold individual fragments of the message
struct Frame {
   char text[FRAME_TEXT_SIZE];
   int sequenceNumber;
} frames[MAX_NO_OF_FRAMES], shuffledFrames[MAX_NO_OF_FRAMES];

// Function to split the message into frames and assign sequence numbers
int assignSequenceNumbers() {
   int i = 0, j = 0, frameIndex = 0;

   while (i < strlen(inputMessage)) {
      frames[frameIndex].sequenceNumber = frameIndex;
      for (j = 0; j < FRAME_TEXT_SIZE && inputMessage[i] != '\0'; j++)
```

```
            frames[frameIndex].text[j] = inputMessage[i++];
        frameIndex++;
    }

    printf("\nAfter assigning sequence numbers:\n");
    for (i = 0; i < frameIndex; i++)
        printf("%d:%s ", frames[i].sequenceNumber, frames[i].text);

    return frameIndex;  // Return total number of frames created
}

// Utility to generate an array of unique random indices for shuffling
void generateRandomArray(int *randomArray, int limit) {
    int r, i = 0, j;
    while (i < limit) {
        r = rand() % limit;
        for (j = 0; j < i; j++)
            if (randomArray[j] == r)
                break;
        if (i == j)  // r is unique
            randomArray[i++] = r;
    }
}

// Shuffle the frames based on random sequence numbers
void shuffleFrames(int totalFrames) {
    int randomIndices[totalFrames];
    generateRandomArray(randomIndices, totalFrames);
```

```
    for (int i = 0; i < totalFrames; i++)
        shuffledFrames[i] = frames[randomIndices[i]];

    printf("\n\nAfter shuffling:\n");
    for (int i = 0; i < totalFrames; i++)
        printf("%d:%s ", shuffledFrames[i].sequenceNumber, shuffledFrames[i].text);
}

// Sort the shuffled frames based on their original sequence numbers
void sortFrames(int totalFrames) {
    int i, j, swapped;
    struct Frame temp;

    for (i = 0; i < totalFrames - 1; i++) {
        swapped = 0;
        for (j = 0; j < totalFrames - i - 1; j++) {
            if (shuffledFrames[j].sequenceNumber > shuffledFrames[j + 1].sequenceNumber) {
                temp = shuffledFrames[j];
                shuffledFrames[j] = shuffledFrames[j + 1];
                shuffledFrames[j + 1] = temp;
                swapped = 1;
            }
        }
        if (!swapped) break; // Optimization: Break if already sorted
    }
}

int main() {
    int totalFrames;
```

```
    printf("Enter the message: ");
    fgets(inputMessage, sizeof(inputMessage), stdin);

    // Remove newline character if present
    size_t len = strlen(inputMessage);
    if (len > 0 && inputMessage[len - 1] == '\n')
        inputMessage[len - 1] = '\0';

    // Split into frames, shuffle them, then sort to recover original message
    totalFrames = assignSequenceNumbers();
    shuffleFrames(totalFrames);
    sortFrames(totalFrames);

    // Display the final reconstructed message
    printf("\n\nAfter sorting (Reconstructed message):\n");
    for (int i = 0; i < totalFrames; i++)
        printf("%s", shuffledFrames[i].text);
    printf("\n\n");

    return 0;
}
```

Output:

```
$ gcc frame_sorting.c
$ ./a.out
```

Enter the message: Study Glance

After assigning sequence numbers:

0:Stu 1:dy 2:Gla 3:nce

After shuffling:

3:nce 2:Gla 1:dy 0:Stu

After sorting (Reconstructed message):

Study Glance

Result: