Slip 1

Q 1. Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix.

```
=> #include <stdio.h>

#define MAX_VERTICES 100

int adj_matrix[MAX_VERTICES][MAX_VERTICES];

int main() {
    int num_vertices, num_edges;

    printf("Enter the number of vertices: ");
    scanf("%d", &num_vertices);

    printf("Enter the number of edges: ");
    scanf("%d", &num_edges);

    // Initialize adjacency matrix to all zeros
    for (int i = 0; i < num_vertices; i++) {
        for (int j = 0; j < num_vertices; j++) {
            adj_matrix[i][j] = 0;
        }
    }

    // Read edges and update adjacency matrix
    printf("Enter the edges as pairs of vertices (e.g. 0 1): \n");
    for (int k = 0; k < num_edges; k++) {
        int i, j;
        scanf("%d %d", &i, &j);
        adj_matrix[i][j] = 1;
        adj_matrix[j][i] = 1; // assuming undirected graph
    }

    // Display adjacency matrix
    printf("Adjacency matrix:\n");
    for (int i = 0; i < num_vertices; i++) {
        for (int j = 0; j < num_vertices; j++) {
            printf("%d ", adj_matrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Q 2. Write a C program for the Implementation of Prim’s Minimum spanning tree algorithm.
=>

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

#define MAX_VERTICES 1000

typedef struct {
    int v;
    int weight;
} edge_t;

int n, m;
int visited[MAX_VERTICES];
int parent[MAX_VERTICES];
edge_t graph[MAX_VERTICES][MAX_VERTICES];

int find_min_vertex(int* dist) {
    int min_vertex = -1;
    int min_distance = INT_MAX;
    for (int i = 1; i <= n; i++) {
        if (!visited[i] && dist[i] < min_distance) {
            min_vertex = i;
            min_distance = dist[i];
        }
    }
    return min_vertex;
}

void prim(int start_vertex) {
    int dist[MAX_VERTICES];
    for (int i = 1; i <= n; i++) {
        dist[i] = INT_MAX;
        visited[i] = 0;
    }
    dist[start_vertex] = 0;
    parent[start_vertex] = -1;
    for (int i = 1; i <= n; i++) {
        int u = find_min_vertex(dist);
        visited[u] = 1;
        for (int v = 1; v <= n; v++) {
            if (graph[u][v].weight != 0 && !visited[v] && graph[u][v].weight < dist[v]) {
                dist[v] = graph[u][v].weight;
                parent[v] = u;
            }
        }
    }
}

void print_mst() {
    int total_weight = 0;
    for (int i = 2; i <= n; i++) {
        int u = parent[i];
        int v = i;
        int weight = graph[u][v].weight;
        printf("%d - %d : %d\n", u, v, weight);
        total_weight += weight;
    }
    printf("Total weight: %d\n", total_weight);
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            graph[i][j].weight = 0;
        }
    }
    for (int i = 0; i < m; i++) {
        int u, v, w;
        scanf("%d %d %d", &u, &v, &w);
        graph[u][v].v = v;
        graph[u][v].weight = w;
        graph[v][u].v = u;
        graph[v][u].weight = w;
    }
    prim(1);
    print_mst();
    return 0;
}
```

Slip2

Q1. Write a C program for the implementation of Topological sorting

=>

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

#define MAX_VERTICES 1000

typedef struct {
    int v;
    int weight;
} edge_t;
```

```
int n, m;
int indegree[MAX_VERTICES];
edge_t graph[MAX_VERTICES][MAX_VERTICES];

void add_edge(int u, int v, int w) {
    graph[u][v].v = v;
    graph[u][v].weight = w;
    indegree[v]++;
}

void topological_sort() {
    int queue[MAX_VERTICES], front = -1, rear = -1;
    for (int i = 1; i <= n; i++) {
        if (indegree[i] == 0) {
            queue[++rear] = i;
        }
    }
    while (front != rear) {
        int u = queue[++front];
        printf("%d ", u);
        for (int v = 1; v <= n; v++) {
            if (graph[u][v].weight != 0) {
                indegree[v]--;
                if (indegree[v] == 0) {
                    queue[++rear] = v;
                }
            }
        }
    }
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            graph[i][j].weight = 0;
        }
        indegree[i] = 0;
    }
    for (int i = 0; i < m; i++) {
        int u, v, w;
        scanf("%d %d %d", &u, &v, &w);
        add_edge(u, v, w);
    }
    topological_sort();
    return 0;
}
```

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as anadjacency matrix. Implement function to traverse the graph using Depth First Search (DFS)traversal.

=>

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

#define MAX_VERTICES 1000

int n, m;
int graph[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES];

void add_edge(int u, int v) {
    graph[u][v] = 1;
    graph[v][u] = 1;
}

void dfs(int u) {
    visited[u] = 1;
    printf("%d ", u);
    for (int v = 1; v <= n; v++) {
        if (graph[u][v] && !visited[v]) {
            dfs(v);
        }
    }
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            graph[i][j] = 0;
        }
        visited[i] = 0;
    }
    for (int i = 0; i < m; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        add_edge(u, v);
    }
    dfs(1);
    return 0;
}
```

Slip3

Q 1. Write a C program for the Implementation of Prim’s Minimum spanning tree algorithm

=>

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define MAX_VERTICES 1000

int n;
int graph[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES];
int parent[MAX_VERTICES];
int key[MAX_VERTICES];

int get_min_key_vertex() {
    int min_key = INT_MAX, min_index;
    for (int v = 1; v <= n; v++) {
        if (!visited[v] && key[v] < min_key) {
            min_key = key[v];
            min_index = v;
        }
    }
    return min_index;
}

void prim() {
    for (int v = 1; v <= n; v++) {
        key[v] = INT_MAX;
        visited[v] = 0;
    }
    key[1] = 0;
    parent[1] = -1;
    for (int i = 1; i < n; i++) {
        int u = get_min_key_vertex();
        visited[u] = 1;
        for (int v = 1; v <= n; v++) {
            if (graph[u][v] && !visited[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    prim();
    printf("Minimum Spanning Tree:\n");
    for (int i = 2; i <= n; i++) {
        printf("%d - %d\n", parent[i], i);
    }
    return 0;
}
```

Q 2. Write a C program for the implementation of Floyd Warshall's algorithm for finding all pairs shortest path using adjacency costmatrix

=>

```
#include <stdio.h>
#include <limits.h>

#define MAX_VERTICES 1000

int n;
int graph[MAX_VERTICES][MAX_VERTICES];

void floyd_warshall() {
    int dist[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            dist[i][j] = graph[i][j];
        }
    }
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX && dist[i][j] > dist[i][k] + dist[k][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
    printf("All Pairs Shortest Paths:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dist[i][j] == INT_MAX) {
                printf("INF ");
            } else {
                printf("%d ", dist[i][j]);
            }
        }
        printf("\n");
    }
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
            if (graph[i][j] == -1) {
                graph[i][j] = INT_MAX;
            }
        }
    }
    floyd_warshall();
    return 0;
}
```

Slip4

Q 1. Write a C program that accepts the vertices and edges of a graph. Create adjacency list.

=>

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

struct Node {
    int vertex;
    struct Node* next;
};

struct Graph {
    int numVertices;
    struct Node** adjLists;
};

struct Node* createNode(int v) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int vertices) {
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->adjLists = malloc(vertices * sizeof(struct Node*));
    int i;
    for (i = 0; i < vertices; i++)
        graph->adjLists[i] = NULL;
    return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

void printGraph(struct Graph* graph) {
    int v;
    for (v = 0; v < graph->numVertices; v++) {
        struct Node* temp = graph->adjLists[v];
        printf("\n Adjacency list of vertex %d\n ", v);
        while (temp) {
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

int main() {
    int vertices, edges, i, src, dest;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &vertices);
    struct Graph* graph = createGraph(vertices);
    printf("Enter the number of edges in the graph: ");
    scanf("%d", &edges);
    for (i = 0; i < edges; i++) {
        printf("Enter the source and destination of edge %d: ", i + 1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }
    printGraph(graph);
    return 0;
}
```

Q 2. Write a C program for the implementation of Topological sorting

=>

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

```
#include <stdlib.h>

#define MAX_VERTICES 100

struct Node {
    int vertex;
    struct Node* next;
};

struct Graph {
    int numVertices;
    struct Node** adjLists;
    int* visited;
};

struct Node* createNode(int v) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int vertices) {
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->adjLists = malloc(vertices * sizeof(struct Node*));
    graph->visited = malloc(vertices * sizeof(int));
    int i;
    for (i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;
}

void printGraph(struct Graph* graph) {
    int v;
    for (v = 0; v < graph->numVertices; v++) {
        struct Node* temp = graph->adjLists[v];
        printf("\n Adjacency list of vertex %d\n ", v);
        while (temp) {
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

void DFS(struct Graph* graph, int vertex, int* stack) {
    struct Node* adjList = graph->adjLists[vertex];
    graph->visited[vertex] = 1;
    while (adjList != NULL) {
        int adjVertex = adjList->vertex;
        if (graph->visited[adjVertex] == 0)
            DFS(graph, adjVertex, stack);
        adjList = adjList->next;
    }
    stack[--(*stack)] = vertex;
}

void topologicalSort(struct Graph* graph) {
    int stack[MAX_VERTICES];
    int i;
    for (i = 0; i < graph->numVertices; i++) {
        if (graph->visited[i] == 0)
            DFS(graph, i, &stack[MAX_VERTICES]);
    }
    printf("\nTopological Sort: ");
    for (i = 0; i < graph->numVertices; i++) {
        printf("%d ", stack[i]);
    }
}

int main() {
    int vertices, edges, i, src, dest;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &vertices);
    struct Graph* graph = createGraph(vertices);
    printf("Enter the number of edges in the graph: ");
    scanf("%d", &edges);
    for (i = 0; i < edges; i++) {
        printf("Enter the source and destination of edge %d: ", i + 1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }
    printGraph(graph);
    topologicalSort(graph);
    return 0;
}
```

Slip6

Q 1. Write a C program for the Implementation of Prim’s Minimum spanning tree algorithm.

=>

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

#define MAX_VERTICES 100

int adjMatrix[MAX_VERTICES][MAX_VERTICES];
int numVertices;
int visited[MAX_VERTICES];
int distance[MAX_VERTICES];
int parent[MAX_VERTICES];

int getMinVertex() {
    int minVertex = -1;
    for (int i = 0; i < numVertices; i++) {
        if (visited[i] == 0 && (minVertex == -1 || distance[i] < distance[minVertex])) {
            minVertex = i;
        }
    }
    return minVertex;
}

void prim() {
    for (int i = 0; i < numVertices; i++) {
        distance[i] = INT_MAX;
    }

    distance[0] = 0;
    parent[0] = -1;

    for (int i = 0; i < numVertices - 1; i++) {
        int minVertex = getMinVertex();
        visited[minVertex] = 1;

        for (int j = 0; j < numVertices; j++) {
            if (adjMatrix[minVertex][j] != 0 && visited[j] == 0 && adjMatrix[minVertex][j] <
distance[j]) {
                distance[j] = adjMatrix[minVertex][j];
                parent[j] = minVertex;
            }
        }
    }
}
```

```
void printMST() {
    printf("Edge \tWeight\n");
    for (int i = 1; i < numVertices; i++) {
        printf("%d - %d \t%d \n", parent[i], i, adjMatrix[parent[i]][i]);
    }
}

int main() {
    int numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for (int i = 0; i < numEdges; i++) {
        int start, end, weight;
        printf("Enter the start, end, and weight of edge %d: ", i + 1);
        scanf("%d %d %d", &start, &end, &weight);
        adjMatrix[start][end] = weight;
        adjMatrix[end][start] = weight;
    }

    prim();

    printMST();

    return 0;
}
```

Q 2. Write aCprogram that acceptsthe vertices and edges of a graph and storesit as an adjacency matrix. Display the adjacency matrix

=>

```
#include <stdio.h>
#define MAX_VERTICES 100

int adjMatrix[MAX_VERTICES][MAX_VERTICES];
int numVertices;

void displayAdjMatrix() {
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            printf("%d ", adjMatrix[i][j]);
        }
        printf("\n");
    }
}
```

```
int main() {
    int numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for (int i = 0; i < numEdges; i++) {
        int start, end;
        printf("Enter the start and end vertices of edge %d: ", i + 1);
        scanf("%d %d", &start, &end);
        adjMatrix[start][end] = 1;
        adjMatrix[end][start] = 1;  // for undirected graph
    }

    displayAdjMatrix();

    return 0;
}
```

Slip7
Q 1. Write a C program for the implementation of Floyd Warshall’s algorithm for finding all pairs shortest path using adjacency cost matrix.
=>

```
#include <stdio.h>
#define INF 99999
#define MAX_VERTICES 100

int adjMatrix[MAX_VERTICES][MAX_VERTICES];
int numVertices;

void floydWarshall() {
    int dist[numVertices][numVertices];

    // Initialize dist matrix with the values of adjMatrix
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            dist[i][j] = adjMatrix[i][j];
        }
    }

    // Update dist matrix with intermediate vertex k
    for (int k = 0; k < numVertices; k++) {
        for (int i = 0; i < numVertices; i++) {
            for (int j = 0; j < numVertices; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    // Print the shortest distance matrix
    printf("Shortest Distance Matrix:\n");
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            if (dist[i][j] == INF) {
                printf("INF ");
            } else {
                printf("%d ", dist[i][j]);
            }
        }
        printf("\n");
    }
}

int main() {
    int numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    // Initialize adjMatrix with INF
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            if (i == j) {
                adjMatrix[i][j] = 0;
            } else {
                adjMatrix[i][j] = INF;
            }
        }
    }

    // Populate adjMatrix with edge weights
    for (int i = 0; i < numEdges; i++) {
        int start, end, weight;
        printf("Enter the start and end vertices and weight of edge %d: ", i + 1);
        scanf("%d %d %d", &start, &end, &weight);
        adjMatrix[start][end] = weight;
    }

    floydWarshall();
    return 0;
}
```

Q 2. Write a C program which uses Binary search tree library and displays nodes at each level,and total levels in the tree

```
=>#include <stdio.h>
#include <stdlib.h>
#include "bst.h"

// Function to calculate the maximum of two integers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Function to calculate the height of a binary search tree
int height(Node* root) {
    if (root == NULL) {
        return 0;
    } else {
        int left_height = height(root->left);
        int right_height = height(root->right);

        return 1 + max(left_height, right_height);
    }
}

// Function to print nodes at a given level of a binary search tree
void print_level(Node* root, int level) {
    if (root == NULL) {
        return;
    }
    if (level == 1) {
        printf("%d ", root->data);
    } else if (level > 1) {
        print_level(root->left, level - 1);
        print_level(root->right, level - 1);
    }
}

// Function to print nodes at all levels of a binary search tree
void print_levels(Node* root) {
    int h = height(root);
    int i;
    for (i = 1; i <= h; i++) {
        printf("Level %d: ", i);
        print_level(root, i);
        printf("\n");
    }
}

// Main function
int main() {
    Node* root = NULL;

    // Inserting nodes into the binary search tree
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);

    // Printing the nodes at each level of the binary search tree
    printf("Nodes at each level of the binary search tree:\n");
    print_levels(root);

    // Calculating the total number of levels in the binary search tree
    int levels = height(root);
    printf("\nTotal levels in the binary search tree: %d", levels);

    return 0;
}
```

Slip9

Q 1. Write a C program that accepts the vertices and edges of a graph. Create adjacency list anddisplay the adjacency list.

=>

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

#define MAX_VERTICES 100

struct Node {
    int vertex;
    struct Node* next;
};

struct Node* adjacency_list[MAX_VERTICES]; // array of linked lists to store adjacency list

void add_edge(int u, int v) {
    // Add edge from vertex u to vertex v in the adjacency list
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->vertex = v;
    new_node->next = adjacency_list[u];
    adjacency_list[u] = new_node;
}

void display_adj_list(int num_vertices) {
    // Display the adjacency list for each vertex
    printf("Adjacency List:\n");
    for (int i = 0; i < num_vertices; i++) {
        printf("Vertex %d: ", i);
        struct Node* current_node = adjacency_list[i];
        while (current_node != NULL) {
            printf("%d ", current_node->vertex);
            current_node = current_node->next;
        }
        printf("\n");
    }
}

int main() {
    int num_vertices, num_edges;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &num_vertices);
    printf("Enter the number of edges in the graph: ");
    scanf("%d", &num_edges);

    // Initialize the adjacency list to NULL
    for (int i = 0; i < num_vertices; i++) {
        adjacency_list[i] = NULL;
    }

    // Take input for each edge and store it in the adjacency list
    printf("Enter the edges of the graph in the format u v, where u and v are vertices: \n");
    int u, v;
    for (int i = 0; i < num_edges; i++) {
        scanf("%d %d", &u, &v);
        add_edge(u, v);
    }

    // Display the adjacency list
    display_adj_list(num_vertices);

    return 0;
}
```

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacencymatrix. Implement function to traverse the graph using Depth First Search (BFS) traversal.

=>

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

#define MAX_VERTICES 100

int adjacency_matrix[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES];

void dfs(int vertex, int num_vertices) {
    visited[vertex] = 1;
    printf("%d ", vertex);
    for (int i = 0; i < num_vertices; i++) {
        if (adjacency_matrix[vertex][i] == 1 && visited[i] == 0) {
            dfs(i, num_vertices);
        }
    }
}

int main() {
    int num_vertices, num_edges;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &num_vertices);
    printf("Enter the number of edges in the graph: ");
    scanf("%d", &num_edges);

    // Initialize the adjacency matrix to all 0's
    for (int i = 0; i < num_vertices; i++) {
        for (int j = 0; j < num_vertices; j++) {
            adjacency_matrix[i][j] = 0;
        }
    }

    // Take input for each edge and store it in the adjacency matrix
    printf("Enter the edges of the graph in the format u v, where u and v are vertices: \n");
    int u, v;
    for (int i = 0; i < num_edges; i++) {
        scanf("%d %d", &u, &v);
        adjacency_matrix[u][v] = 1;
        adjacency_matrix[v][u] = 1; // assuming undirected graph
    }

    // Display the adjacency matrix
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < num_vertices; i++) {
        for (int j = 0; j < num_vertices; j++) {
            printf("%d ", adjacency_matrix[i][j]);
        }
        printf("\n");
    }

    // Perform DFS traversal
    printf("DFS Traversal: ");
    for (int i = 0; i < num_vertices; i++) {
        visited[i] = 0;
    }
    for (int i = 0; i < num_vertices; i++) {
        if (visited[i] == 0) {
            dfs(i, num_vertices);
        }
    }
    printf("\n");

    return 0;
}
```

Slip10

Q 1. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, inorder. Write a menu driven program that performs the above operations.

=>

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* create_node(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        root = create_node(data);
        return root;
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void menu() {
    printf("Binary Search Tree Operations\n");
    printf("1. Create Tree\n");
    printf("2. Insert Node\n");
    printf("3. Inorder Traversal\n");
    printf("4. Exit\n");
}

int main() {
    int choice, data;
    struct Node* root = NULL;
    do {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the root node data: ");
                scanf("%d", &data);
                root = insert(root, data);
                break;
            case 2:
                printf("Enter the data to be inserted: ");
                scanf("%d", &data);
                insert(root, data);
                break;
            case 3:
                printf("Inorder Traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 4:
                printf("Exiting program...");
                break;
            default:
                printf("Invalid choice. Try again.\n");
        }
        printf("\n");
    } while (choice != 4);
    return 0;
}
```

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Breadth First Search (BFS) traversal.

=>

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_VERTICES 100

int adj_matrix[MAX_VERTICES][MAX_VERTICES];
bool visited[MAX_VERTICES];
int queue[MAX_VERTICES];
int front = -1, rear = -1;

void breadth_first_search(int start_vertex, int num_vertices) {
    int i, vertex;

    // Mark all vertices as not visited
    for(i = 0; i < num_vertices; i++) {
        visited[i] = false;
    }

    // Mark the starting vertex as visited and enqueue it
    visited[start_vertex] = true;
    rear++;
    queue[rear] = start_vertex;

    while(front != rear) {
        // Dequeue a vertex from the queue and print it
        front++;
        vertex = queue[front];
        printf("%d ", vertex);

        // Get all adjacent vertices of the dequeued vertex
        // If an adjacent vertex has not been visited, mark it as visited and enqueue it
        for(i = 0; i < num_vertices; i++) {
            if(adj_matrix[vertex][i] == 1 && !visited[i]) {
                visited[i] = true;
                rear++;
                queue[rear] = i;
            }
        }
    }
}

int main() {
    int i, j, num_vertices, num_edges, v1, v2;

    // Read in the number of vertices and edges
    printf("Enter the number of vertices: ");
    scanf("%d", &num_vertices);

    printf("Enter the number of edges: ");
    scanf("%d", &num_edges);

    // Initialize the adjacency matrix
    for(i = 0; i < num_vertices; i++) {
        for(j = 0; j < num_vertices; j++) {
            adj_matrix[i][j] = 0;
        }
    }

    // Read in the edges and store them in the adjacency matrix
    for(i = 0; i < num_edges; i++) {
        printf("Enter edge %d: ", i + 1);
        scanf("%d %d", &v1, &v2);
        adj_matrix[v1][v2] = 1;
        adj_matrix[v2][v1] = 1;
    }

    // Print the adjacency matrix
    printf("Adjacency matrix:\n");
    for(i = 0; i < num_vertices; i++) {
        for(j = 0; j < num_vertices; j++) {
            printf("%d ", adj_matrix[i][j]);
        }
        printf("\n");
    }

    // Perform BFS traversal starting from vertex 0
    printf("BFS traversal starting from vertex 0: ");
    breadth_first_search(0, num_vertices);

    return 0;
}
```

Slip11

Q1 Write a C program for the implementation of Floyd Warshall's algorithm for finding all pairs
shortest path using adjacency cost matrix.
=>

```
#include <stdio.h>
#include <limits.h>

#define V 4  // number of vertices in the graph

// function to print the solution matrix
void printSolution(int dist[][V]) {
  printf("The following matrix shows the shortest distances"
      " between every pair of vertices:\n");
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
      if (dist[i][j] == INT_MAX)
        printf("INF\t");
      else
        printf("%d\t", dist[i][j]);
    }
    printf("\n");
  }
}

// implementation of Floyd Warshall algorithm
void floydWarshall(int graph[][V]) {
  int dist[V][V];

  // initialize the solution matrix
  for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
      dist[i][j] = graph[i][j];

  // find the shortest path for all pairs of vertices
  for (int k = 0; k < V; k++) {
    for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
        if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX
          && dist[i][k] + dist[k][j] < dist[i][j])
          dist[i][j] = dist[i][k] + dist[k][j];
      }
    }
  }

  // print the solution matrix
  printSolution(dist);
}
```

```
// main function
int main() {
    int graph[V][V] = { {0, 5, INT_MAX, 10},
                        {INT_MAX, 0, 3, INT_MAX},
                        {INT_MAX, INT_MAX, 0, 1},
                        {INT_MAX, INT_MAX, INT_MAX, 0} };

    // call the Floyd Warshall function
    floydWarshall(graph);

    return 0;
}
```

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacency matrix. Implement function to traverse the graph using Depth First Search (DFS) traversal.

=>

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

#define MAX_VERTICES 100

// adjacency matrix to represent the graph
int adj[MAX_VERTICES][MAX_VERTICES];
// array to keep track of visited vertices
int visited[MAX_VERTICES];

// function to add an edge to the graph
void add_edge(int u, int v) {
    adj[u][v] = 1;
    adj[v][u] = 1;
}

// DFS traversal function
void DFS(int v, int n) {
    printf("%d ", v);
    visited[v] = 1;
    for (int i = 0; i < n; i++) {
        if (adj[v][i] == 1 && !visited[i]) {
            DFS(i, n);
        }
    }
}

int main() {
    int n, e, u, v;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the number of edges: ");
    scanf("%d", &e);
    // initialize the adjacency matrix
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            adj[i][j] = 0;
        }
        visited[i] = 0;
    }
    // add edges to the graph
    printf("Enter the edges (u v): ");
    for (int i = 0; i < e; i++) {
        scanf("%d %d", &u, &v);
        add_edge(u, v);
    }
    // perform DFS traversal
    printf("DFS traversal: ");
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            DFS(i, n);
        }
    }
    return 0;
}
```

Slip12

Q 1. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, preorder. Write a menu driven program that performs the above operations

=>

```
#ifndef BTREE_H
#define BTREE_H

typedef struct node {
    int data;
    struct node* left;
    struct node* right;
} Node;

Node* create_node(int data);
Node* insert(Node* root, int data);
void preorder(Node* root);

#endif

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

```
#include "btree.h"

Node* create_node(int data) {
    Node* node = (Node*) malloc(sizeof(Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}

Node* insert(Node* root, int data) {
    if (root == NULL) {
        return create_node(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

void preorder(Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

#include <stdio.h>
#include <stdlib.h>
#include "btree.h"

int main() {
    Node* root = NULL;
    int choice, data;

    while (1) {
        printf("\nBinary Search Tree Operations\n");
        printf("------------------------------\n");
        printf("1. Insert\n");
        printf("2. Preorder Traversal\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter data to be inserted: ");
                scanf("%d", &data);
                root = insert(root, data);
                break;
            case 2:
                printf("Preorder traversal: ");
                preorder(root);
                printf("\n");
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }

    return 0;
}
```

Q 2. Write a C program that acceptsthe vertices and edges of a graph and store it as an adjacency matrix. Implement functions to print indegree, outdegree and total degree of all vertices of graph.

=>

```
#include <stdio.h>

#define MAX_VERTICES 100

int adjMatrix[MAX_VERTICES][MAX_VERTICES];

int main() {
    int numVertices, numEdges, i, j;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    // Initialize the adjacency matrix
    for (i = 0; i < numVertices; i++) {
        for (j = 0; j < numVertices; j++) {
            adjMatrix[i][j] = 0;
        }
    }

    // Get the edges
    for (i = 0; i < numEdges; i++) {
        int u, v;
        printf("Enter edge %d (u v): ", i+1);
        scanf("%d %d", &u, &v);
        adjMatrix[u][v] = 1;
    }

    // Print the adjacency matrix
    printf("Adjacency matrix:\n");

    for (i = 0; i < numVertices; i++) {
        for (j = 0; j < numVertices; j++) {
            printf("%d ", adjMatrix[i][j]);
        }
        printf("\n");
    }

    // Print the indegree, outdegree, and total degree of each vertex
    for (i = 0; i < numVertices; i++) {
        int indegree = 0, outdegree = 0;
        for (j = 0; j < numVertices; j++) {
            if (adjMatrix[i][j] == 1) {
                outdegree++;
            }
            if (adjMatrix[j][i] == 1) {
                indegree++;
            }
        }
        printf("Vertex %d: indegree = %d, outdegree = %d, total degree = %d\n",
               i, indegree, outdegree, indegree + outdegree);
    }

    return 0;
}
```

Slip13
Q 1. Write a C program for the Implementation of Kruskal’s Minimum spanning tree algorithm.
=>

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

#define MAX_EDGES 100
#define MAX_VERTICES 100

struct Edge {
    int u, v, weight;
};
```

```
int numVertices, numEdges;
int parent[MAX_VERTICES];
struct Edge edges[MAX_EDGES];

int find(int u) {
    if (parent[u] == u) {
        return u;
    }
    return find(parent[u]);
}

void unionSet(int u, int v) {
    parent[find(u)] = find(v);
}

int compare(const void* a, const void* b) {
    struct Edge* edgeA = (struct Edge*) a;
    struct Edge* edgeB = (struct Edge*) b;
    return edgeA->weight - edgeB->weight;
}

void kruskal() {
    qsort(edges, numEdges, sizeof(struct Edge), compare);

    for (int i = 0; i < numVertices; i++) {
        parent[i] = i;
    }

    int numSelected = 0;
    struct Edge selected[MAX_EDGES];

    for (int i = 0; i < numEdges && numSelected < numVertices - 1; i++) {
        int u = edges[i].u;
        int v = edges[i].v;
        if (find(u) != find(v)) {
            unionSet(u, v);
            selected[numSelected] = edges[i];
            numSelected++;
        }
    }

    printf("Minimum spanning tree:\n");
    int totalWeight = 0;
    for (int i = 0; i < numSelected; i++) {
        printf("(%d, %d) - %d\n", selected[i].u, selected[i].v, selected[i].weight);
        totalWeight += selected[i].weight;
    }
    printf("Total weight: %d\n", totalWeight);
}

int main() {
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for (int i = 0; i < numEdges; i++) {
        printf("Enter edge %d (u v weight): ", i+1);
        scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].weight);
    }

    kruskal();

    return 0;
}
```

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacencymatrix. Implement function to traverse the graph using Breadth First Search (BFS) traversal

=>

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

#define MAX_VERTICES 100

int adjMatrix[MAX_VERTICES][MAX_VERTICES];
bool visited[MAX_VERTICES];
int numVertices, numEdges;

void bfs(int startVertex) {
    int queue[MAX_VERTICES], front = 0, rear = 0;
    visited[startVertex] = true;
    queue[rear++] = startVertex;

    while (front != rear) {
        int currVertex = queue[front++];
        printf("%d ", currVertex);

        for (int i = 0; i < numVertices; i++) {
            if (adjMatrix[currVertex][i] == 1 && !visited[i]) {
                visited[i] = true;
                queue[rear++] = i;
            }
        }
    }
}

int main() {
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for (int i = 0; i < numEdges; i++) {
        printf("Enter edge %d (u v): ", i+1);
        int u, v;
        scanf("%d %d", &u, &v);
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1;
    }

    printf("Enter the starting vertex: ");
    int startVertex;
    scanf("%d", &startVertex);

    printf("Breadth First Search (BFS) traversal: ");
    bfs(startVertex);

    return 0;
}
```

Slip14

Q 1. Write a C program for the implementation of Floyd Warshall’s algorithm for finding all pairs shortest path using adjacency cost matrix.

=>

```
#include <stdio.h>

#define INF 99999
#define MAX_VERTICES 100

int adjMatrix[MAX_VERTICES][MAX_VERTICES];
int numVertices;

void floydWarshall() {
    int dist[numVertices][numVertices];

    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            dist[i][j] = adjMatrix[i][j];
        }
    }

    for (int k = 0; k < numVertices; k++) {
        for (int i = 0; i < numVertices; i++) {
            for (int j = 0; j < numVertices; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    printf("All pairs shortest path:\n");

    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            if (dist[i][j] == INF) {
                printf("INF ");
            } else {
                printf("%d ", dist[i][j]);
            }
        }
        printf("\n");
    }
}

int main() {
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    printf("Enter the adjacency matrix:\n");

    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            scanf("%d", &adjMatrix[i][j]);
            if (adjMatrix[i][j] == 0 && i != j) {
                adjMatrix[i][j] = INF;
            }
        }
    }

    floydWarshall();

    return 0;
}
```

Q 2. Write a C program which uses Binary search tree library and displays nodes at each level,and total levels in the tree

=>

```
#include <stdio.h>
#include <stdlib.h>
#include "btree.h"

void displayLevelOrder(BTreeNode* root) {
   if (root == NULL) {
      return;
   }

   Queue* queue = createQueue();

   enqueue(queue, root);

   int currentLevelCount = 1;
   int nextLevelCount = 0;
   int level = 1;

   printf("Level %d: ", level);

   while (!isEmpty(queue)) {
      BTreeNode* node = dequeue(queue);
      printf("%d ", node->data);

      currentLevelCount--;

      if (node->left != NULL) {
         enqueue(queue, node->left);
         nextLevelCount++;
      }

      if (node->right != NULL) {
         enqueue(queue, node->right);
         nextLevelCount++;
      }

      if (currentLevelCount == 0) {
         currentLevelCount = nextLevelCount;
         nextLevelCount = 0;
         level++;
         printf("\nLevel %d: ", level);
      }
   }

   printf("\nTotal levels: %d\n", level - 1);
}
```

```
int main() {
    BTreeNode* root = NULL;

    int option;
    int data;

    do {
        printf("\nBinary Search Tree Menu\n");
        printf("1. Insert a node\n");
        printf("2. Display nodes at each level and total levels\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &option);

        switch (option) {
            case 1:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                root = insertNode(root, data);
                printf("Node inserted successfully.\n");
                break;
            case 2:
                printf("Nodes at each level:\n");
                displayLevelOrder(root);
                break;
            case 3:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid option. Please try again.\n");
                break;
        }
    } while (option != 3);

    return 0;
}
```

Slip16

Q 1. Write a C program for the implementation of Floyd Warshall's algorithm for finding all pairs shortest path using adjacency cost matrix

=>

```
#include <stdio.h>

#define INF 99999
#define V 4
```

```
void floydWarshall(int graph[][V]) {
    int dist[V][V], i, j, k;

    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            dist[i][j] = graph[i][j];
        }
    }

    for (k = 0; k < V; k++) {
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    printf("Shortest distances between every pair of vertices:\n");
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            if (dist[i][j] == INF) {
                printf("%7s", "INF");
            } else {
                printf("%7d", dist[i][j]);
            }
        }
        printf("\n");
    }
}

int main() {
    int graph[V][V] = {
        {0, 5, INF, 10},
        {INF, 0, 3, INF},
        {INF, INF, 0, 1},
        {INF, INF, INF, 0}
    };

    floydWarshall(graph);

    return 0;
}
```

Q 2. Write a C program which uses Binary search tree library and displays nodes at each level,and total levels in the tree.

=>

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }

    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }

    return root;
}

int maxDepth(struct Node* node) {
    if (node == NULL) {
        return 0;
    } else {
        int leftDepth = maxDepth(node->left);
        int rightDepth = maxDepth(node->right);

        if (leftDepth > rightDepth) {
            return (leftDepth + 1);
        } else {
            return (rightDepth + 1);
        }
    }
}

void printGivenLevel(struct Node* root, int level) {
    if (root == NULL) {
        return;
    }

    if (level == 1) {
        printf("%d ", root->data);
    } else if (level > 1) {
        printGivenLevel(root->left, level - 1);
        printGivenLevel(root->right, level - 1);
    }
}

void printLevelOrder(struct Node* root) {
    int i;
    int depth = maxDepth(root);

    printf("Total levels in the tree: %d\n", depth);

    for (i = 1; i <= depth; i++) {
        printf("Nodes at level %d: ", i);
        printGivenLevel(root, i);
        printf("\n");
    }
}

int main() {
    struct Node* root = NULL;

    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    printLevelOrder(root);

    return 0;
}
```

Slip17

Q1. Write a menu driven program to implement hash table using array (insert, delete,display).

Use any of the above-mentioned hash functions. In case of collision apply linear probing.

=>

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

```
#include <stdlib.h>

#define SIZE 10

int hash(int key) {
    return key % SIZE;
}

void insert(int ht[], int key) {
    int index = hash(key);
    int i = 0;
    while (ht[(index + i) % SIZE] != -1) {
        i++;
    }
    ht[(index + i) % SIZE] = key;
}

void delete(int ht[], int key) {
    int index = hash(key);
    int i = 0;
    while (ht[(index + i) % SIZE] != key) {
        if (ht[(index + i) % SIZE] == -1) {
            printf("Key not found\n");
            return;
        }
        i++;
    }
    ht[(index + i) % SIZE] = -1;
    printf("Key %d deleted\n", key);
}

void display(int ht[]) {
    printf("HashTable: ");
    for (int i = 0; i < SIZE; i++) {
        if (ht[i] != -1) {
            printf("%d ", ht[i]);
        } else {
            printf("_ ");
        }
    }
    printf("\n");
}

int main() {
    int ht[SIZE];
    for (int i = 0; i < SIZE; i++) {
        ht[i] = -1;
    }
    int choice, key;
    while (1) {
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the key to be inserted: ");
                scanf("%d", &key);
                insert(ht, key);
                break;
            case 2:
                printf("Enter the key to be deleted: ");
                scanf("%d", &key);
                delete(ht, key);
                break;
            case 3:
                display(ht);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

Q 2. Write a C program which uses Binary search tree library and displays nodes at each level,and total levels in the tree.

=>

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
    int data;
    struct node *left;
    struct node *right;
} Node;

Node *createNode(int data) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
```

```
}

Node *insert(Node *root, int data) {
    if (root == NULL) {
        return createNode(data);
    } else if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

int getHeight(Node *root) {
    if (root == NULL) {
        return 0;
    }
    int leftHeight = getHeight(root->left);
    int rightHeight = getHeight(root->right);
    return (leftHeight > rightHeight) ? (leftHeight + 1) : (rightHeight + 1);
}

void printLevel(Node *root, int level) {
    if (root == NULL) {
        return;
    }
    if (level == 1) {
        printf("%d ", root->data);
    } else if (level > 1) {
        printLevel(root->left, level - 1);
        printLevel(root->right, level - 1);
    }
}

void printTree(Node *root) {
    int height = getHeight(root);
    printf("Total levels in the tree: %d\n", height);
    for (int i = 1; i <= height; i++) {
        printf("Level %d: ", i);
        printLevel(root, i);
        printf("\n");
    }
}

int main() {
    Node *root = NULL;
    int choice, data;
    while (1) {
        printf("\n1. Insert\n2. Print levels\n3. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data: ");
                scanf("%d", &data);
                root = insert(root, data);
                break;
            case 2:
                printTree(root);
                break;
            case 3:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

Slip18

Q 1. Write aCprogramthat acceptsthe vertices and edges of a graph and storesit as an adjacency matrix. Display the adjacency matrix.

=>

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

#define MAX_VERTICES 10

int adjMatrix[MAX_VERTICES][MAX_VERTICES];

int main() {
    int numVertices, numEdges;

    // Accept the number of vertices and edges
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    // Initialize the adjacency matrix to all zeros
    for(int i = 0; i < numVertices; i++) {
        for(int j = 0; j < numVertices; j++) {
            adjMatrix[i][j] = 0;
        }
    }

    // Accept the edges and store them in the adjacency matrix
    printf("Enter the edges (in the format: vertex1 vertex2):\n");
    int v1, v2;
    for(int i = 0; i < numEdges; i++) {
        scanf("%d %d", &v1, &v2);
        adjMatrix[v1][v2] = 1;
        adjMatrix[v2][v1] = 1; // Uncomment this line for an undirected graph
    }

    // Display the adjacency matrix
    printf("The adjacency matrix is:\n");
    for(int i = 0; i < numVertices; i++) {
        for(int j = 0; j < numVertices; j++) {
            printf("%d ", adjMatrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Q 2. Write a C program for the Implementation of Prim’s Minimum spanning tree algorithm
=>

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

#define V 5  // Maximum number of vertices in the graph

int minKey(int key[], int mstSet[]) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++) {
        if (mstSet[v] == 0 && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    }

    return min_index;
}

void printMST(int parent[], int graph[V][V]) {
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++) {
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
    }
}

void primMST(int graph[V][V]) {
    int parent[V];  // Array to store the constructed MST
    int key[V];  // Key values used to pick minimum weight edge in cut
    int mstSet[V];  // To represent set of vertices not yet included in MST

    // Initialize all keys as infinite and mstSet[] as
    for (int i = 0; i < V; i++) {
        key[i] = INT_MAX;
        mstSet[i] = 0;
    }

    // Always include first  vertex in MST.
    // Make
```

Slip19

Q 1. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, in order. Write a menu driven program that performs the above operations.

=>

```
#include <stdio.h>
#include <stdlib.h>
#include "btree.h"  // include header file for the BST library

int main() {
    int option, key;
    BSTNode *root = NULL;  // initialize the root node to null

    do {
        // display menu options
        printf("\nMenu:\n");
        printf("1. Create a new BST\n");
        printf("2. Insert a node into the BST\n");
        printf("3. Perform an inorder traversal of the BST\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &option);

        switch(option) {
            case 1:
                // create a new BST
                root = createBST();
                printf("BST created successfully.\n");
                break;
            case 2:
                // insert a node into the BST
                printf("Enter the key to insert: ");
                scanf("%d", &key);
                root = insertNode(root, key);
                printf("Node with key %d inserted successfully.\n", key);
                break;
            case 3:
                // perform an inorder traversal of the BST
                printf("Inorder traversal of the BST:\n");
                inorderTraversal(root);
                printf("\n");
                break;
            case 4:
                // exit the program
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while(option != 4);

    return 0;
}
```

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacencymatrix. Implement function to traverse the graph using Depth First Search (DFS) traversal.

=>

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

#define MAX_VERTICES 10

// function prototypes
void initializeGraph(int graph[][MAX_VERTICES], int numVertices);
void addEdge(int graph[][MAX_VERTICES], int u, int v);
void DFS(int graph[][MAX_VERTICES], int startVertex, int visited[]);

int main() {
    int numVertices, numEdges;
    int graph[MAX_VERTICES][MAX_VERTICES];
    int visited[MAX_VERTICES];

    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    initializeGraph(graph, numVertices); // initialize graph with zeros

    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for(int i=0; i<numEdges; i++) {
        int u, v;
        printf("Enter edge (u, v): ");
        scanf("%d %d", &u, &v);
        addEdge(graph, u, v);
    }

    // perform DFS traversal
    printf("DFS traversal of the graph: ");
    for(int i=0; i<numVertices; i++) {
        visited[i] = 0;
    }

    for(int i=0; i<numVertices; i++) {
        if(visited[i] == 0) {
            DFS(graph, i, visited);
        }
    }

    printf("\n");

    return 0;
}

// function to initialize graph with zeros
void initializeGraph(int graph[][MAX_VERTICES], int numVertices) {
    for(int i=0; i<numVertices; i++) {
        for(int j=0; j<numVertices; j++) {
            graph[i][j] = 0;
        }
    }
}

// function to add edge to graph
void addEdge(int graph[][MAX_VERTICES], int u, int v) {
    graph[u][v] = 1;
    graph[v][u] = 1;
}

// function to perform DFS traversal
void DFS(int graph[][MAX_VERTICES], int startVertex, int visited[]) {
    visited[startVertex] = 1;
    printf("%d ", startVertex);
    for(int i=0; i<MAX_VERTICES; i++) {
        if(graph[startVertex][i] == 1 && visited[i] == 0) {
            DFS(graph, i, visited);
        }
    }
}
```

Slip20

Q 1. Write a C program for the Implementation of Kruskal’s Minimum spanning tree algorithm.

=>

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

#define MAX_EDGES 100

// structure to represent an edge
struct Edge {
    int source, destination, weight;
};

// structure to represent a subset for Union-Find algorithm
struct Subset {
    int parent, rank;
};

// function prototypes
int find(struct Subset subsets[], int i);
void Union(struct Subset subsets[], int x, int y);
int compareEdges(const void* a, const void* b);
void KruskalMST(struct Edge edges[], int numVertices, int numEdges);

int main() {
    int numVertices, numEdges;
    struct Edge edges[MAX_EDGES];

    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for(int i=0; i<numEdges; i++) {
        printf("Enter edge %d (source, destination, weight): ", i+1);
        scanf("%d %d %d", &edges[i].source, &edges[i].destination, &edges[i].weight);
    }
    KruskalMST(edges, numVertices, numEdges);

    return 0;
}

// function to find the subset of an element using Union-Find algorithm
int find(struct Subset subsets[], int i) {
    if(subsets[i].parent != i) {
        subsets[i].parent = find(subsets, subsets[i].parent);
    }

    return subsets[i].parent;
}

// function to perform Union of two subsets using Union-Find algorithm
void Union(struct Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if(subsets[xroot].rank < subsets[yroot].rank) {
        subsets[xroot].parent = yroot;
    } else if(subsets[xroot].rank > subsets[yroot].rank) {
        subsets[yroot].parent = xroot;
    } else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

// function to compare two edges for sorting
int compareEdges(const void* a, const void* b) {
    struct Edge* edge1 = (struct Edge*) a;
    struct Edge* edge2 = (struct Edge*) b;

    return edge1->weight - edge2->weight;
}

// function to perform Kruskal's minimum spanning tree algorithm
void KruskalMST(struct Edge edges[], int numVertices, int numEdges) {
    struct Edge result[numVertices]; // array to store the minimum spanning tree
    struct Subset subsets[numVertices];
    int i = 0, e = 0;

    qsort(edges, numEdges, sizeof(edges[0]), compareEdges);

    // initialize subsets
    for(int i=0; i<numVertices; i++) {
        subsets[i].parent = i;
        subsets[i].rank = 0;
    }

    while(e < numVertices - 1 && i < numEdges) {
        struct Edge nextEdge = edges[i++];

        int x = find(subsets, nextEdge.source);
        int y = find(subsets, nextEdge.destination);

        if(x != y) {
            result[e++] = nextEdge;
            Union(subsets, x, y);
        }
    }

    printf("Minimum Spanning Tree:\n");
    for(int i=0; i<e; i++) {
        printf("%d - %d : %d\n", result[i].source, result[i].
```

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacencymatrix. Implement function to traverse the graph using Breadth First Search (BFS) traversal

=>

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

#define MAX_VERTICES 100

// function prototypes
void BFS(int adjacencyMatrix[][MAX_VERTICES], int numVertices, int startVertex);

// main function
int main() {
    int numVertices, numEdges;
    int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES] = {0};

    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for(int i=0; i<numEdges; i++) {
        int source, destination;
        printf("Enter edge %d (source, destination): ", i+1);
        scanf("%d %d", &source, &destination);
        adjacencyMatrix[source][destination] = 1;
        adjacencyMatrix[destination][source] = 1;
    }

    int startVertex;
    printf("Enter the starting vertex: ");
    scanf("%d", &startVertex);

    BFS(adjacencyMatrix, numVertices, startVertex);

    return 0;
}

// function to perform BFS traversal on the graph represented as an adjacency matrix
void BFS(int adjacencyMatrix[][MAX_VERTICES], int numVertices, int startVertex) {
    bool visited[MAX_VERTICES] = {false};
    int queue[MAX_VERTICES];
    int front = 0, rear = 0;

    visited[startVertex] = true;
    queue[rear++] = startVertex;

    while(front != rear) {
        int currentVertex = queue[front++];
        printf("%d ", currentVertex);

        for(int i=0; i<numVertices; i++) {
            if(adjacencyMatrix[currentVertex][i] && !visited[i]) {
                visited[i] = true;
                queue[rear++] = i;
            }
        }
    }
}
```

Slip21

Q 1. Write a C program for the implen of Dijkstra’s shortest path algoritforfindingshortest path from a given source vertex using adjacency cost matrix

=>

```
#include <stdio.h>
#include <limits.h>

#define MAX_VERTICES 100

// function prototypes
```

```
void dijkstra(int adjacencyMatrix[][MAX_VERTICES], int numVertices, int startVertex, int
distance[]);

// main function
int main() {
    int numVertices, numEdges;
    int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES] = {0};

    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for(int i=0; i<numEdges; i++) {
        int source, destination, weight;
        printf("Enter edge %d (source, destination, weight): ", i+1);
        scanf("%d %d %d", &source, &destination, &weight);
        adjacencyMatrix[source][destination] = weight;
    }

    int startVertex;
    printf("Enter the starting vertex: ");
    scanf("%d", &startVertex);

    int distance[MAX_VERTICES];
    dijkstra(adjacencyMatrix, numVertices, startVertex, distance);

    printf("Shortest distances from vertex %d:\n", startVertex);
    for(int i=0; i<numVertices; i++) {
        printf("%d to %d: %d\n", startVertex, i, distance[i]);
    }

    return 0;
}

// function to perform Dijkstra's shortest path algorithm on a graph represented as an
adjacency cost matrix
void dijkstra(int adjacencyMatrix[][MAX_VERTICES], int numVertices, int startVertex, int
distance[]) {
    bool visited[MAX_VERTICES] = {false};

    for(int i=0; i<numVertices; i++) {
        distance[i] = INT_MAX;
    }

    distance[startVertex] = 0;
    for(int i=0; i<numVertices-1; i++) {
        int minDistance = INT_MAX, minVertex;
        for(int j=0; j<numVertices; j++) {
            if(!visited[j] && distance[j] < minDistance) {
                minDistance = distance[j];
                minVertex = j;
            }
        }

        visited[minVertex] = true;

        for(int j=0; j<numVertices; j++) {
            if(adjacencyMatrix[minVertex][j] && !visited[j]) {
                int newDistance = distance[minVertex] + adjacencyMatrix[minVertex][j];
                if(newDistance < distance[j]) {
                    distance[j] = newDistance;
                }
            }
        }
    }
}
```

Slip22

Q 1. Write a C program that accepts the vertices and edges of a graph. Create adjacency list and display the adjacency list.

=>

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

struct Node {
    int vertex;
    struct Node* next;
};

struct Graph {
    int numVertices;
    struct Node** adjLists;
};

// function prototypes
struct Node* createNode(int v);
struct Graph* createGraph(int vertices);
void addEdge(struct Graph* graph, int src, int dest);
void printGraph(struct Graph* graph);

// main function
int main() {
    int numVertices, numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    struct Graph* graph = createGraph(numVertices);

    for(int i=0; i<numEdges; i++) {
        int src, dest;
        printf("Enter edge %d (source, destination): ", i+1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    printf("Adjacency list:\n");
    printGraph(graph);

    return 0;
}

// function to create a new node with a given vertex number
struct Node* createNode(int v) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

// function to create a graph with a given number of vertices
struct Graph* createGraph(int vertices) {
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->adjLists = malloc(vertices * sizeof(struct Node*));

    for(int i=0; i<vertices; i++) {
        graph->adjLists[i] = NULL;
    }

    return graph;
}

// function to add an edge to an undirected graph
void addEdge(struct Graph* graph, int src, int dest) {
    // add edge from src to dest
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;
    // add edge from dest to src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// function to print the adjacency list of a graph
void printGraph(struct Graph* graph) {
    for(int i=0; i<graph->numVertices; i++) {
        struct Node* temp = graph->adjLists[i];
        printf("%d: ", i);
        while(temp) {
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}
```

Q 2. Write a C program that accepts the vertices and edges of a graph and store it as an adjacencymatrix. Implement function to traverse the graph using Depth First Search (DFS) traversal.

=>

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

#define MAX_VERTICES 100

int adjMatrix[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES];

// function prototypes
void init();
void addEdge(int src, int dest);
void dfs(int vertex, int numVertices);

// main function
int main() {
    int numVertices, numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    init();
    for(int i=0; i<numEdges; i++) {
        int src, dest;
        printf("Enter edge %d (source, destination): ", i+1);
        scanf("%d %d", &src, &dest);
        addEdge(src, dest);
    }

    printf("Depth First Search (DFS) Traversal:\n");
    for(int i=0; i<numVertices; i++) {
        if(!visited[i]) {
            dfs(i, numVertices);
        }
    }

    return 0;
}

// function to initialize the adjacency matrix and visited array
void init() {
    for(int i=0; i<MAX_VERTICES; i++) {
        for(int j=0; j<MAX_VERTICES; j++) {
            adjMatrix[i][j] = 0;
        }
        visited[i] = 0;
    }
}

// function to add an edge to the adjacency matrix
void addEdge(int src, int dest) {
    adjMatrix[src][dest] = 1;
    adjMatrix[dest][src] = 1;
}

// function to perform DFS traversal of the graph
void dfs(int vertex, int numVertices) {
    visited[vertex] = 1;
    printf("%d ", vertex);

    for(int i=0; i<numVertices; i++) {
        if(adjMatrix[vertex][i] && !visited[i]) {
            dfs(i, numVertices);
        }
    }
}
```

Slip23

Q 1. Write a C program for the Implementation of Prim’s Minimum spanning tree algorithm

=>

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

#define MAX_VERTICES 100

int adjMatrix[MAX_VERTICES][MAX_VERTICES];
int numVertices;

// function prototypes
void init();
void addEdge(int src, int dest, int weight);
void prim();

// main function
int main() {
    int numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    init();

    for(int i=0; i<numEdges; i++) {
        int src, dest, weight;
        printf("Enter edge %d (source, destination, weight): ", i+1);
        scanf("%d %d %d", &src, &dest, &weight);
        addEdge(src, dest, weight);
    }

    prim();

    return 0;
}

// function to initialize the adjacency matrix
void init() {
    for(int i=0; i<MAX_VERTICES; i++) {
        for(int j=0; j<MAX_VERTICES; j++) {
            adjMatrix[i][j] = 0;
        }
    }
}

// function to add an edge to the adjacency matrix
void addEdge(int src, int dest, int weight) {
    adjMatrix[src][dest] = weight;
    adjMatrix[dest][src] = weight;
}

// function to find the minimum weight edge from the set of vertices
int findMin(int key[], int mstSet[]) {
    int min = INT_MAX, minIndex;
    for(int i=0; i<numVertices; i++) {
        if(mstSet[i] == 0 && key[i] < min) {
            min = key[i];
            minIndex = i;
        }
    }
    return minIndex;
}

// function to perform Prim's Minimum Spanning Tree Algorithm
void prim() {
    int parent[MAX_VERTICES], key[MAX_VERTICES], mstSet[MAX_VERTICES];

    for(int i=0; i<numVertices; i++) {
        key[i] = INT_MAX;
        mstSet[i] = 0;
    }

    key[0] = 0;
    parent[0] = -1;

    for(int count=0; count<numVertices-1; count++) {
        int u = findMin(key, mstSet);
        mstSet[u] = 1;

        for(int v=0; v<numVertices; v++) {
            if(adjMatrix[u][v] && mstSet[v] == 0 && adjMatrix[u][v] < key[v]) {
                parent[v] = u;
                key[v] = adjMatrix[u][v];
            }
        }
    }

    printf("Edges of Minimum Spanning Tree:\n");
    for(int i=1; i<numVertices; i++) {
        printf("%d - %d\n", parent[i], i);
    }
}
```

Q 2. Write a C program for the implementation of Floyd Warshall’s algorithm for finding all pairs shortest path using adjacency cost matrix.

=>

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

#define MAX_VERTICES 100

int adjMatrix[MAX_VERTICES][MAX_VERTICES];
int numVertices;

// function prototypes
void init();
void addEdge(int src, int dest, int weight);
void floydWarshall();

// main function
int main() {
    int numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    init();

    for(int i=0; i<numEdges; i++) {
        int src, dest, weight;
        printf("Enter edge %d (source, destination, weight): ", i+1);
        scanf("%d %d %d", &src, &dest, &weight);
        addEdge(src, dest, weight);
    }

    floydWarshall();

    return 0;
}

// function to initialize the adjacency matrix
void init() {
    for(int i=0; i<MAX_VERTICES; i++) {
        for(int j=0; j<MAX_VERTICES; j++) {
            if(i == j) {
                adjMatrix[i][j] = 0;
            } else {
                adjMatrix[i][j] = INT_MAX;
            }
        }
    }
}

// function to add an edge to the adjacency matrix
void addEdge(int src, int dest, int weight) {
    adjMatrix[src][dest] = weight;
}

// function to perform Floyd Warshall's Algorithm
void floydWarshall() {
    int dist[MAX_VERTICES][MAX_VERTICES];

    // initialize the distance matrix
    for(int i=0; i<numVertices; i++) {
        for(int j=0; j<numVertices; j++) {
            dist[i][j] = adjMatrix[i][j];
        }
    }

    // compute shortest path for all pairs
    for(int k=0; k<numVertices; k++) {
        for(int i=0; i<numVertices; i++) {
            for(int j=0; j<numVertices; j++) {
                if(dist[i][k] != INT_MAX && dist[k][j] != INT_MAX &&
                   dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    printf("All Pairs Shortest Path:\n");
    for(int i=0; i<numVertices; i++) {
        for(int j=0; j<numVertices; j++) {
            if(dist[i][j] == INT_MAX) {
                printf("INF ");
            } else {
                printf("%d ", dist[i][j]);
            }
        }
        printf("\n");
    }
}
```

Slip25

Q 1. Write a C program for the implementation of Floyd Warshall’s algorithm for finding all pairs shortest path using adjacency cost matrix.

=>

```
#include <stdio.h>
#include <limits.h>

#define MAX_VERTICES 100

int adjMatrix[MAX_VERTICES][MAX_VERTICES];
int numVertices;

// Function prototypes
void init();
void addEdge(int src, int dest, int weight);
void floydWarshall();

int main() {
    int numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    init();

    for (int i = 0; i < numEdges; i++) {
        int src, dest, weight;
        printf("Enter edge %d (source, destination, weight): ", i + 1);
        scanf("%d %d %d", &src, &dest, &weight);
        addEdge(src, dest, weight);
    }

    floydWarshall();

    return 0;
}

void init() {
    for (int i = 0; i < MAX_VERTICES; i++) {
        for (int j = 0; j < MAX_VERTICES; j++) {
            if (i == j) {
                adjMatrix[i][j] = 0;
            } else {
                adjMatrix[i][j] = INT_MAX;
            }
        }
    }
}
```

```
void addEdge(int src, int dest, int weight) {
    adjMatrix[src][dest] = weight;
}

void floydWarshall() {
    int dist[MAX_VERTICES][MAX_VERTICES];

    // Initialize the distance matrix
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            dist[i][j] = adjMatrix[i][j];
        }
    }

    // Compute shortest path for all pairs
    for (int k = 0; k < numVertices; k++) {
        for (int i = 0; i < numVertices; i++) {
            for (int j = 0; j < numVertices; j++) {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX &&
                   dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    // Display the all pairs shortest path
    printf("All Pairs Shortest Path:\n");
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            if (dist[i][j] == INT_MAX) {
                printf("INF ");
            } else {
                printf("%d ", dist[i][j]);
            }
        }
        printf("\n");
    }
}
```

Q 2. Write a C program which uses Binary search tree library and displays nodes at each level,and total levels in the tree.

=>

```
#include <stdio.h>
#include "btree.h" // assuming btree.h library has already been implemented

int main() {
    int n, i, data;
    struct node* root = NULL;

    // create a binary search tree
    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    printf("Enter the nodes:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &data);
        root = insert(root, data);
    }

    // display nodes at each level
    printf("\nNodes at each level:\n");
    int level;
    for (level = 1; level <= height(root); level++) {
        printf("Level %d: ", level);
        printLevel(root, level);
        printf("\n");
    }

    // display the total number of levels in the tree
    printf("\nTotal levels: %d\n", height(root));

    return 0;
}
```