Write C program which uses Binary search tree library and displays nodes
Create
Inorder
Preorder
postorder

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

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

NODE *create_bst(NODE *root) {
   int i, n, num;
   NODE *newnode, *temp, *parent;
   printf("Enter how many nodes you want to
create\n");
   scanf("%d", &n);
   printf("Enter data in node\n");
   for (i = 0; i < n; i++) {
      newnode = (NODE
*)malloc(sizeof(NODE));
      scanf("%d", &num);
      newnode->data = num;
      newnode->left = newnode->right = NULL;
      if (root == NULL) {
         root = newnode;
         continue;
      }
      temp = root;
      while (temp != NULL) {
         parent = temp;
         if (num < temp->data)
            temp = temp->left;
         else
            temp = temp->right;
      }
      if (num < parent->data)
         parent->left = newnode;
      else
         parent->right = newnode;
   }
   return root;
}

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

void inorder(NODE *root) {
   NODE *temp = root;
   if (temp != NULL) {
      inorder(temp->left);
      printf("%d\t", temp->data);
      inorder(temp->right);
   }
}

void postorder(NODE *root) {
   NODE *temp = root;
   if (temp != NULL) {
      postorder(temp->left);
      postorder(temp->right);
      printf("%d\t", temp->data);
   }
}

NODE *search(NODE *root, int key) {
   NODE *temp = root;
   while (temp != NULL) {
      if (temp->data == key)
         return temp;
      else if (key < temp->data)
         temp = temp->left;
      else
         temp = temp->right;
   }
   return NULL;
}

int main() {
   NODE *root = NULL;
   NODE *t;
   int ch, k;
   do {
      printf("1. Create\n2. Search\n3. Inorder \n4.
Preorder\n5.  display Postorder\n6. Exit\n");
      printf("Enter your choice\n");
      scanf("%d", &ch);
      switch (ch) {
         case 1:
            root = create_bst(root);
            break;
         case 2:
            printf("Enter a node you want to
search\n");
            scanf("%d", &k);
            t = search(root, k);
            if (t == NULL)
               printf("Not found\n");
            else
               printf("Found\n");
            break;
         case 3:
            inorder(root);
            break;
         case 4:
            preorder(root);
            break;
         case 5:
            postorder(root);
            break;
```

```
            case 6:
               exit(0);
         }
      } while (ch != 6);
      return 0;
}
```

Write C program which uses Binary search tree library and displays nodes
a) at each level count of nodes. b) total levels in the tree.

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

NODE *create_bst(NODE *root) {
   int i, n, num;
   NODE *newnode, *temp, *parent;

   printf("Enter how many nodes you want to
create\n");
   scanf("%d", &n);
   printf("Enter data in node\n");

   for (i = 0; i < n; i++) {
      newnode = (NODE
*)malloc(sizeof(NODE));
      scanf("%d", &num);
      newnode->data = num;
      newnode->left = newnode->right = NULL;

      if (root == NULL) {
         root = newnode;
         continue;
      }

      temp = root;
      while (temp != NULL) {
         parent = temp;
         if (num < temp->data)
            temp = temp->left;
         else
            temp = temp->right;
      }

      if (num < parent->data)
         parent->left = newnode;
      else
         parent->right = newnode;
   }
   return root;
}

// Function to count total nodes in the BST
int count(NODE *root) {
   static int cnt = 0;
   NODE *temp = root;
   if (temp != NULL) {
      cnt++;
      count(temp->left);
      count(temp->right);
   }
   return cnt;
}

int countLeaf(NODE *root) {
   static int leaf = 0;
   NODE *temp = root;
   if (temp != NULL) {
      if ((temp->left == NULL) && (temp->right
== NULL))
         leaf++;
      countLeaf(temp->left);
      countLeaf(temp->right);
   }
   return leaf;
}

void main() {
   NODE *root = NULL;
   int ch, n, ln;

   do {
      printf("1. Create\n2. Total Nodes\n3. Total
Leaf Nodes\n4. Exit\n");
      printf("Enter your choice\n");
      scanf("%d", &ch);
      switch (ch) {
         case 1:
            root = create_bst(root);
            break;
         case 2:
            n = count(root);
            printf("Total Nodes = %d\n", n);
            break;
         case 3:
            ln = countLeaf(root);
            printf("Total Leaf Nodes = %d\n", ln);
            break;
         case 4:
            exit(0);
      }
   } while (ch != 4);
}
```

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 n 4

void printMatrix(int matrix[n][n])
{
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         if (matrix[i][j] == 999)
```

```
            printf("%4s", "INF");
         else
            printf("%4d", matrix[i][j]);
      }
      printf("\n");
   }
}

void floydWarshall(int matrix[][n])
{
   int i, j, k;
   for (k = 0; k < n; k++)
   {
      for (i = 0; i < n; i++)
      {
         for (j = 0; j < n; j++)
         {
            if (matrix[i][k] + matrix[k][j] <
matrix[i][j])
               matrix[i][j] = matrix[i][k] +
matrix[k][j];
         }
      }
   }
   printMatrix(matrix);
}

int main()
{
   int matrix[n][n];

   printf("Enter the adjacency matrix (999 for
infinity):\n");
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         scanf("%d", &matrix[i][j]);
      }
   }

   printf("\nOriginal Matrix:\n");
   printMatrix(matrix);

   printf("\nShortest Paths Matrix\n");
   floydWarshall(matrix);

   return 0;
}
```

---

20 marks

Write a C program that accepts the 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 the graph.

```
#include <stdio.h>
#include <stdlib.h> // Include necessary header
for malloc

typedef struct node {
   int vertex;
   struct node *next;
} NODE;

NODE *list[10]; // Declare list array

void createmat(int m[10][10], int n) {
   int i, j;
   char ans;
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         m[i][j] = 0;
         if (i != j) {
            printf("\nIs there an edge between %d
and %d (1/0): ", i + 1, j + 1);
            scanf("%d", &m[i][j]);
         }
      }
   }
}

void dispmat(int m[10][10], int n) {
   int i, j;
   char ans;
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         printf("%5d", m[i][j]);
      }
      printf("\n");
   }
}
void degree(int m[10][10], int n) {
   int v, in, out, total, i;
   for (v = 0; v < n; v++) {
      in = out = 0; // Initialize in and out degree
      for (i = 0; i < n; i++) {
         in += m[i][v]; // Increment in-degree if
there's an edge to v
         out += m[v][i]; // Increment out-degree if
there's an edge from v
      }
      printf("Vertex %d: In-degree = %d, Out-
degree = %d\n", v + 1, in, out);
   }
}

int main() {
   int m[10][10];
   int n;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   createmat(m, n);
   dispmat(m, n);
   createlist(m, n);
   displist(n);
   degree(m, n);
   return 0;
}
```

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>

typedef struct node {
   int vertex;
   struct node *next;
} NODE;

void createmat(int m[10][10], int n) {
   int i, j;
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         m[i][j] = 0;
         if (i != j) {
            printf("\n Is there any edge between %d and %d (1/0): ", i + 1, j + 1);
            scanf("%d", &m[i][j]);
         }
      }
   }
}

void displaymat(int m[10][10], int n) {
   int i, j;
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         printf("%d ", m[i][j]);
      }
      printf("\n");
   }
}

int main() {
   int m[10][10], n;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   createmat(m, n);
   printf("The adjacency matrix is:\n");
   displaymat(m, n);
   return 0;
}
```

Write a program to sort n randomly generated elements using Heap Sort method

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

void display(int arr[], int n) {
   for (int i = 0; i < n; i++)
      printf("%d\t", arr[i]);
}

void Heapify(int A[], int top, int last) {
   int j, temp, key;
   key = A[top];
   j = 2 * top + 1;
   if ((j < last) && (A[j] < A[j + 1]))
      j = j + 1;
   if ((j <= last) && (key < A[j])) {
      temp = A[top];
      A[top] = A[j];
      A[j] = temp;
      Heapify(A, j, last);
   }
}

void BuildHeap(int A[], int n) {
   for (int i = n / 2 - 1; i >= 0; i--)
      Heapify(A, i, n - 1);
}

void Heapsort(int A[], int n) {
   int temp, top = 0, last;
   BuildHeap(A, n);
   printf("Initial heap=");
   display(A, n);
   for (last = n - 1; last >= 1; last--) {
      temp = A[top];
      A[top] = A[last];
      A[last] = temp;
      printf("\nAfter Iteration %d:", n - last);
      display(A, n);
      Heapify(A, top, last - 1);
   }
}

int main()
{
   // Seed for random number generation
   int A[8];
   printf("Randomly generated elements:\n");
   for (int i = 0; i < 8; i++) {
      A[i] = rand() % 100; // Generates random numbers between 0 and 99
      printf("%d ", A[i]);
   }
   printf("\n");
   Heapsort(A, 8);
   printf("\nThe sorted elements are:");
   display(A, 8);
   return 0;
}
```

Write a C program for the Implementation of Prim's Minimum spanning tree algorithm.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10

int cost[MAX][MAX];
int n;

void prim()
{
   int a, b, u, v, i, j, e;
   int visited[MAX] = {0}, min, mincost = 0;
   visited[0] = 1;

   printf("\nMinimum Spanning Tree Edges:\n");
```

```
   for (e = 0; e < n - 1; e++)
   {
      min = 999;
      for (i = 0; i < n; i++)
      {
         if (visited[i] != 0)
         {
            for (j = 0; j < n; j++) {
               if (cost[i][j] < min && visited[j] ==
0)
               {
                  min = cost[i][j];
                  a = u = i;
                  b = v = j;
               }
            }
         }
      }

      visited[v] = 1;
      printf("Edge %d: (%d, %d) cost: %d\n", e +
1, a + 1, b + 1, min);
      mincost += min;
   }

   printf("\nMinimum cost = %d\n", mincost);
}


int main()
{
   int i, j;

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

   printf("Enter the cost matrix (999 for
infinity):\n");
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         scanf("%d", &cost[i][j]);
      }
   }

   prim();

   return 0;
}
```

Write a C program for the implementation of Topological sorting.

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

int stack[20];
int visited[20]={0};
int top=-1;

void creatmat(int m[10][10],int n)
{
   int i,j;
   char ans;
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         m[i][j]=0;
         if(i!=j)
         {
            printf("Is ther an edge between %d
and %d(1/0):",i+1,j+1);
            scanf("%d",&m[i][j]);
         }
      }
   }
}

void dispmat(int m[10][10],int n)
{
   int i,j;
   char ans;
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         printf("%d \t",m[i][j]);
      }
      printf("\n");
   }
}

void dfs(int m[10][10],int i,int n)
{
   int j;
   visited[i]=1;
   for(j=0;j<n;j++)
   {
      if(m[i][j]==1 && !visited[j])
      {
         dfs(m,j,n);
      }
   }
   stack[++top]=i;
}

void topologicalSort(int m[10][10],int n)
{
   int i;
   for(i=0;i<n;i++)
   {
      if(!visited[i])
      {
         dfs(m,i,n);
      }
   }
   printf("\nTopological Sort is: \n");
   while(top!=-1)
   {
      printf("%d",stack[top--]);
```

```
    }
}

void main()
{
    int m[10][10];
    int n;
    printf("Enter the no. of vertices \n");
    scanf("%d",&n);
    createmat(m,n);
    dispmat(m,n);
    topologicalSort(m,n);
}
```

Write a C program for the Implementation of Kruskal's Minimum spanning tree algorithm

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

typedef struct {
    int src, dest, weight;
} edge;

edge *graph, *mst;
int *MSTvertices;

void sort(edge graph[], int nE) {
    for (int pass = 1; pass <= nE - 1; pass++)
        for (int i = 0; i < nE - pass; i++)
            if (graph[i].weight > graph[i + 1].weight)
{
                edge temp = graph[i];
                graph[i] = graph[i + 1];
                graph[i + 1] = temp;
            }
}

int find(int V, int nV) {
    for (int k = 0; k < nV; k++)
        if (MSTvertices[k] == V)
            return 1;
    return 0;
}

void kruskalMST(int nV, int nE) {
    int i = 0, j = 0, k = 0, count = 0, mincost = 0,
first = 0, second = 0;
    MSTvertices = (int *)malloc(nV * sizeof(int));
    for (int v = 0; v < nV; v++) MSTvertices[v] =
-1;
    sort(graph, nE);
    while (count < nV - 1) {
        first = find(graph[i].src, nV);
        second = find(graph[i].dest, nV);
        if (!(first && second)) {
            mst[k++] = graph[i];
            count++;
            mincost += graph[i].weight;
            if (!first) MSTvertices[j++] = graph[i].src;
            if (!second) MSTvertices[j++] =
graph[i].dest;
        }
        i++;
    }
    printf("Edges in MST:\n");
    for (i = 0; i < nV - 1; i++)
        printf("%d -- %d == %d\n", mst[i].src,
mst[i].dest, mst[i].weight);
    printf("MST Cost: %d\n", mincost);
}

int main() {
    int nV, nE;
    printf("Vertices: ");
    scanf("%d", &nV);
    printf("Edges: ");
    scanf("%d", &nE);
    graph = (edge *)malloc(nE * sizeof(edge));
    mst = (edge *)malloc((nV - 1) * sizeof(edge));
    printf("Edges (source dest weight):\n");
    for (int i = 0; i < nE; i++)
        scanf("%d %d %d", &graph[i].src,
&graph[i].dest, &graph[i].weight);
    kruskalMST(nV, nE);
    free(graph);
    free(mst);
    free(MSTvertices);
    return 0;
}
```

Write a C program for the implementation of Dijkstra's shortest path algorithm for finding shortest path from a given source vertex using adjacency cost matrix

```
#include<stdio.h>

void dijkstra(int v, int n, int cost[10][10])
{
    int i, j, u, w, count, min;
    int dist[10], visited[10] = {0};

    visited[v] = 1;

    for (i = 0; i < n; i++)
        dist[i] = cost[v][i];

    count = 1;

    while (count < n)
    {
        min = 999;

        for (i = 0; i < n; i++)
            if (visited[i] == 0 && dist[i] < min)
            {
                min = dist[i];
                u = i;
            }

        visited[u] = 1;
```

```
      for (w = 0; w < n; w++)
         if (dist[u] + cost[u][w] < dist[w])
            dist[w] = dist[u] + cost[u][w];

      count++;
   }

   printf("\nShortest distances from vertex %d
are: \n", v);

   for (i = 0; i < n; i++)
      printf("%d\t", dist[i]);
}

int main()
{
   int v, n,i,j;

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

   int cost[10][10];

   printf("Enter the cost matrix:\n");
   for (i = 0; i < n; i++)
      for ( j = 0; j < n; j++)
         scanf("%d", &cost[i][j]);

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

   dijkstra(v, n, cost);

   return 0;
}
```

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>

typedef struct node {
   int vertex;
   struct node *next;
} NODE;

NODE *list[10]; // Declare list array

void creatmat(int m[10][10], int n) {
   int i, j;
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         m[i][j] = 0;
         if (i != j) {
            printf("\nIs there an edge between %d
and %d (1/0): ", i + 1, j + 1);
            scanf("%d", &m[i][j]);
         }
      }
   }
}

void dispmat(int m[10][10], int n) {
   int i, j;
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         printf("%5d", m[i][j]);
      }
      printf("\n");
   }
}

void createlist(int m[10][10], int n) {
   int i, j;
   NODE *temp, *newnode;
   for (i = 0; i < n; i++) {
      list[i] = NULL;
      for (j = 0; j < n; j++) {
         if (m[i][j] == 1) {
            newnode=(NODE
*)malloc(sizeof(NODE));
            newnode->vertex = j + 1;
            newnode->next = NULL;
            if (list[i] == NULL)
               list[i] = temp = newnode;
            else {
               temp->next = newnode;
               temp = newnode;
            }
         }
      }
   }
}

void displist(int n) {
   NODE *temp;
   int i;
   printf("The Adjacency List is :\n");
   for (i = 0; i < n; i++) {
      printf("V%d->", i + 1);
      temp = list[i];
      while (temp) {
         printf("V%d->", temp->vertex);
         temp = temp->next;
      }
      printf("NULL\n");
   }
}

int main() {
   int m[10][10];
   int n;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   creatmat(m, n);
   dispmat(m, n);
   createlist(m, n);
   displist(n);
```

```
    return 0;
}
```

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>

typedef struct node {
    int vertex;
    struct node *next;
} NODE;

void createmat(int m[10][10], int n) {
    int i, j;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            m[i][j] = 0;
            if (i != j) {
                printf("\nIs there an edge between %d and %d (1/0) : ", i + 1, j + 1);
                scanf("%d", &m[i][j]);
            }
        }
    }
}

void dispmat(int m[10][10], int n) {
    int i, j;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("%5d", m[i][j]);
        }
        printf("\n");
    }
}

void dfs(int m[10][10], int n, int v) {
    int w;
    static int visited[10] = {0};
    visited[v] = 1;
    printf("v%d ", v + 1);
    for (w = 0; w < n; w++) {
        if (m[v][w] == 1 && visited[w] == 0) {
            dfs(m, n, w);
        }
    }
}

int main() {
    int n, m[10][10];
    printf("\nEnter the number of vertices: ");
    scanf("%d", &n);
    createmat(m, n);
    dispmat(m, n);
    printf("\nDFS Traversal: ");
    dfs(m, n, 0); // Start DFS from vertex 0
    printf("\n");
    return 0;
}
```

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

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

int n, i, j, visited[20], queue[10], front = -1, rear = -1;
int adj[10][10];
typedef struct node {
    int vertex;
    struct node* next;
} NODE;
NODE* list[10];

void createmat(int m[10][10], int n) {
    int i, j;
    char ans;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            m[i][j] = 0;
            if (i != j) {
                printf("\nIs there an edge between %d and %d (1/0): ", i + 1, j + 1);
                scanf("%d", &m[i][j]);
            }
        }
    }
}

void dispmat(int m[10][10], int n) {
    int i, j;
    char ans;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("%5d", m[i][j]);
        }
        printf("\n");
    }
}

void createlist(int m[10][10], int n) {
    int i, j;
    NODE* temp, * newnode;
    for (i = 0; i < n; i++) {
        list[i] = NULL;
        for (j = 0; j < n; j++) {
            if (m[i][j] == 1) {
                newnode= (NODE*)malloc(sizeof(NODE));
                newnode->vertex = j + 1;
                newnode->next = NULL;
                if (list[i] == NULL)
                    list[i] = temp = newnode;
                else {
                    temp->next = newnode;
                    temp = newnode;
                }
```

```
        }
      }
   }
}

void displist(int n) {
   NODE* temp;
   int i;
   printf("The adjacency list is\n");
   for (i = 0; i < n; i++) {
      printf("v%d->", i + 1);
      temp = list[i];
      while (temp) {
         printf("v%d->", temp->vertex);
         temp = temp->next;
      }
      printf("NULL\n");
   }
}

void bfs(int v) {
   int i;
   for (i = 1; i <= n; i++) {
      if (adj[v][i] && !visited[i]) {
         queue[++rear] = i;
         visited[i] = 1;
      }
   }
   if (front <= rear) {
      bfs(queue[++front]);
   }
}

int main() {
   int v;
   printf("\nEnter the number of vertices: ");
   scanf("%d", &n);
   createmat(adj, n);
   dispmat(adj, n);
   createlist(adj, n);
   displist(n);
   printf("\nThe BFS traversal of the graph is:\n");
   bfs(0);
   return 0;
}
```

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>

int n, i, j, visited[10], queue[10], front = -1, rear = -1;
int adj[10][10];

void bfs(int v) {
   queue[++rear] = v;
   visited[v] = 1;
   while (front < rear) {
      v = queue[++front];
      printf("%d ", v); // Print the vertex being visited
      for (i = 1; i <= n; i++) {
         if (adj[v][i] && !visited[i]) {
            queue[++rear] = i;
            visited[i] = 1;
         }
      }
   }
}

int main() {
   int v;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   for (i = 1; i <= n; i++) {
      visited[i] = 0;
   }
   printf("Enter graph data in matrix form: \n");
   for (i = 1; i <= n; i++)
      for (j = 1; j <= n; j++)
         scanf("%d", &adj[i][j]);
   printf("Enter the starting vertex: ");
   scanf("%d", &v);
   bfs(v);
   printf("\nThe nodes which are reachable are: \n");
   int reachable = 1; // Assume BFS is possible by default
   for (i = 1; i <= n; i++) {
      if (!visited[i]) {
         printf("%d is unreachable.\n", i);
         reachable = 0; // Update flag if any node is unreachable
      }
   }
   if (!reachable) {
      printf("BFS is not possible. Not all nodes are reachable.\n");
      return 1; // indicate an error
   }
   return 0;
}
```

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

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

typedef struct node {
   int vertex;
   struct node* next;
} NODE;

NODE* list[10];
```

```
void createmat(int m[10][10], int n) {
   int i, j;
   char ans;
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         m[i][j] = 0;
         if (i != j) {
            printf("\nIs there an edge between %d
and %d (1/0): ", i + 1, j + 1);
            scanf("%d", &m[i][j]);
            getchar();  // consume newline
character
         }
      }
   }
}

void dispmat(int m[10][10], int n) {
   int i, j;
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         printf("%5d", m[i][j]);
      }
      printf("\n");
   }
}

void createlist(int m[10][10], int n) {
   int i, j;
   NODE *temp, *newnode;
   for (i = 0; i < n; i++) {
      list[i] = NULL;
      for (j = 0; j < n; j++) {
         if (m[i][j] == 1) {
            newnode=
(NODE*)malloc(sizeof(NODE));
            newnode->vertex = j + 1;
            newnode->next = NULL;
            if (list[i] == NULL)
               list[i] = temp = newnode;
            else {
               temp->next = newnode;
               temp = newnode;
            }
         }
      }
   }
}

void displist(int n) {
   NODE *temp;
   int i;
   printf("The adjacency list is:\n");
   for (i = 0; i < n; i++) {
      printf("v%d -> ", i + 1);
      temp = list[i];
      while (temp) {
         printf("v%d -> ", temp->vertex);
         temp = temp->next;
      }
      printf("NULL\n");
   }
}

void dfs(int m[10][10], int n, int v) {
   int w;
   static int visited[20] = {0};
   visited[v] = 1;
   printf("v%d\t", v + 1);
   for (w = 0; w < n; w++) {
      if ((m[v][w] == 1) && (visited[w] == 0))
         dfs(m, n, w);
   }
   printf("\n");
}

int main() {
   int m[10][10], n;
   printf("\nEnter the number of vertices: ");
   scanf("%d", &n);
   createmat(m, n);
   dispmat(m, n);
   createlist(m, n);
   displist(n);
   printf("\nThe DFS traversal of the graph
is:\n");
   dfs(m, n, 0);
   return 0;
}
```