Assignment-11

A) Write a program to implement DFS.

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

struct Stack {
    int *arr;
    int top;
    int capacity;
};

struct Stack* createStack(int capacity) {
    struct Stack *stack = (struct Stack*) malloc(sizeof(struct Stack));
    stack->arr = (int*) malloc(capacity * sizeof(int));
    stack->top = -1;
    stack->capacity = capacity;
    return stack;
}

int isEmpty(struct Stack *stack) {
    return stack->top == -1;
}

void push(struct Stack *stack, int value) {
    if (stack->top == stack->capacity - 1) return;
    stack->arr[++stack->top] = value;
}

int pop(struct Stack *stack) {
    if (isEmpty(stack)) return -1;
    return stack->arr[stack->top--];
}

int main() {
```

```c
int n, e, start;
printf("Enter the number of vertices: ");
scanf("%d", &n);
printf("Enter the number of edges: ");
scanf("%d", &e);

int **graph = (int **)malloc(n * sizeof(int *));
for (int i = 0; i < n; i++) {
    graph[i] = (int *)malloc(n * sizeof(int));
    for (int j = 0; j < n; j++) graph[i][j] = 0;
}

printf("Enter edges in format: source destination\n");
for (int i = 0; i < e; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    graph[u][v] = 1;
    graph[v][u] = 1;
}

printf("Enter the starting vertex: ");
scanf("%d", &start);
if (start < 0 || start >= n) {
    printf("Invalid starting vertex.\n");
    for (int i = 0; i < n; i++) free(graph[i]);
    free(graph);
    return 1;
}

int *visited = (int *)malloc(n * sizeof(int));
for (int i = 0; i < n; i++) visited[i] = 0;

struct Stack *stack = createStack(n);
push(stack, start);
```

```
    printf("DFS traversal starting from vertex %d:\n", start);
    while (!isEmpty(stack)) {
        int u = pop(stack);
        if (!visited[u]) {
            printf("%d ", u);
            visited[u] = 1;
            for (int v = n - 1; v >= 0; v--) {
                if (graph[u][v] && !visited[v]) {
                    push(stack, v);
                }
            }
        }
    }
    printf("\n");

    for (int i = 0; i < n; i++) free(graph[i]);
    free(graph);
    free(visited);
    free(stack->arr);
    free(stack);

    return 0;
}
```

B) Write a program to implement BFS.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int vertex;
    struct Node* next;
};
struct Graph {
    int numVertices;
    struct Node** adjLists;
    int* visited;
};
struct Queue {
    int* items;
    int front;
    int rear;
    int capacity;
};
struct Node* createNode(int v) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}
struct Graph* createGraph(int vertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->adjLists = (struct Node**)malloc(vertices * sizeof(struct Node*));
    graph->visited = (int*)malloc(vertices * sizeof(int));
    int i;
    for (i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
```

```c
}
void addEdge(struct Graph* graph, int src, int dest) {
    struct Node* newNode = createNode(dest);
    if (graph->adjLists[src] == NULL) {
        graph->adjLists[src] = newNode;
    } else {
        struct Node* temp = graph->adjLists[src];
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
    newNode = createNode(src);
    if (graph->adjLists[dest] == NULL) {
        graph->adjLists[dest] = newNode;
    } else {
        struct Node* temp = graph->adjLists[dest];
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
}
struct Queue* createQueue(int capacity) {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->items = (int*)malloc(capacity * sizeof(int));
    q->front = 0;
    q->rear = -1;
    q->capacity = capacity;
    return q;
}
int isEmpty(struct Queue* q) {
    return q->front > q->rear;
}
void enqueue(struct Queue* q, int value) {
    q->rear++;
    q->items[q->rear] = value;
```

```
}
int dequeue(struct Queue* q) {
    int val = q->items[q->front];
    q->front++;
    return val;
}
void BFS(struct Graph* graph, int startVertex) {
    struct Queue* q = createQueue(graph->numVertices);
    graph->visited[startVertex] = 1;
    enqueue(q, startVertex);
    while (!isEmpty(q)) {
        int currentVertex = dequeue(q);
        printf("%d ", currentVertex);
        struct Node* temp = graph->adjLists[currentVertex];
        while (temp) {
            int adjVertex = temp->vertex;
            if (graph->visited[adjVertex] == 0) {
                graph->visited[adjVertex] = 1;
                enqueue(q, adjVertex);
            }
            temp = temp->next;
        }
    }
    free(q->items);
    free(q);
}
int main() {
    int vertices, edges, i;
    printf("Enter number of vertices: ");
    scanf("%d", &vertices);
    struct Graph* graph = createGraph(vertices);
    printf("Enter number of edges: ");
    scanf("%d", &edges);
    printf("Enter edges:\n");
    for (i = 0; i < edges; i++) {
        int src, dest;
        scanf("%d %d", &src, &dest);
        if (src >= 0 && src < vertices && dest >= 0 && dest < vertices)
            addEdge(graph, src, dest);
        else {
            printf("Invalid edge\n");
            i--;
        }
    }
    int start;
    printf("Enter starting vertex for BFS: ");
    scanf("%d", &start);
    if (start < 0 || start >= vertices) {
        printf("Invalid starting vertex\n");
        return 1;
    }
    printf("BFS Traversal: ");
    BFS(graph, start);
    printf("\n");
    free(graph->visited);
    for (i = 0; i < vertices; i++) {
        struct Node* temp = graph->adjLists[i];
        while (temp) {
            struct Node* next = temp->next;
            free(temp);
            temp = next;
        }
    }
    free(graph->adjLists);
    free(graph);
    return 0;
}
```