Experiment 11

CODE:

1) BFS

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

#define MAX 100

int adjMatrix[MAX][MAX];
int visited[MAX];

typedef struct {
int items[MAX];
   int front, rear;
} Queue;

void initQueue(Queue* q) {     q-
>front = 0;
   q->rear = 0;
}

int isQueueEmpty(Queue* q) {
   return q->front == q->rear;
}

void enqueue(Queue* q, int value) {
   if (q->rear < MAX) {
      q->items[q->rear++] = value;
   }
}

int dequeue(Queue* q) {
return q->items[q->front++]; }
```

```
void performBFS(int startVertex, int vertexCount) {
Queue queue;     initQueue(&queue);
visited[startVertex] = 1;
   enqueue(&queue, startVertex);

   printf("BFS Order: ");     while
(!isQueueEmpty(&queue)) {        int
currentVertex = dequeue(&queue);
printf("%d ", currentVertex);

       for (int i = 0; i < vertexCount; i++) {
          if (adjMatrix[currentVertex][i] == 1 && !visited[i]) {
visited[i] = 1;
             enqueue(&queue, i);
          }
       }
   }
   printf("\n");
}

void initializeGraph(int* vertexCount, int* edgeCount) {
   printf("Enter the number of vertices: ");
scanf("%d", vertexCount);

   for (int i = 0; i < *vertexCount; i++) {
for (int j = 0; j < *vertexCount; j++) {
          adjMatrix[i][j] = 0;
       }
       visited[i] = 0;
   }
```

```
  printf("Enter the number of edges: ");
scanf("%d", edgeCount);

   for (int i = 0; i < *edgeCount; i++) {
int u, v;
      printf("Enter edge (u, v): ");
scanf("%d %d", &u, &v);
      if (u < *vertexCount && v < *vertexCount) {
         adjMatrix[u][v] = 1;
adjMatrix[v][u] = 1;
      } else {
         printf("Invalid edge: (%d, %d)\n", u, v);
         i--; // Retry the input for the same edge index
      }
   }
}
int main() {
   int vertexCount, edgeCount, startVertex;

   initializeGraph(&vertexCount, &edgeCount);

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

   if (startVertex >= 0 && startVertex < vertexCount) {
printf("Breadth-First Search starting from vertex %d:\n",
startVertex);
      performBFS(startVertex, vertexCount);
   } else {
      printf("Invalid starting vertex.\n");
   }
   return 0;
}
```

Output:

```
Enter the number of vertices: 5
Enter the number of edges: 5
Enter edge (u, v): 1 0
Enter edge (u, v): 3 0
Enter edge (u, v): 2 1
Enter edge (u, v): 4 1
Enter edge (u, v): 4 2
Enter the starting vertex: 4
Breadth-First Search starting from vertex 4:
BFS Order: 4 1 2 0 3

Enter the number of vertices: 5
Enter the number of edges: 5
Enter edge (u, v): 0 1
Enter edge (u, v): 0 3
Enter edge (u, v): 1 2
Enter edge (u, v): 1 4
Enter edge (u, v): 2 4
Enter the starting vertex: 1
Breadth-First Search starting from vertex 1:
BFS Order: 1 0 2 4 3

=== Code Execution Successful ===
```

2) DFS

```
#include <stdio.h>

#define MAX 100

int adjMatrix[MAX][MAX];
int visited[MAX];

void depthFirstSearch(int vertex, int vertexCount) {
   visited[vertex] = 1;
   printf("%d ", vertex);

   for (int i = 0; i < vertexCount; i++) {        if
(adjMatrix[vertex][i] == 1 && !visited[i]) {
          depthFirstSearch(i, vertexCount);
       }
   }
}

void initializeGraph(int* vertexCount, int* edgeCount) {
   printf("Enter the number of vertices: ");
scanf("%d", vertexCount);

   for (int i = 0; i < *vertexCount; i++) {
for (int j = 0; j < *vertexCount; j++) {
          adjMatrix[i][j] = 0;
       }
       visited[i] = 0;
   }

   printf("Enter the number of edges: ");
scanf("%d", edgeCount);
```

```
    for (int i = 0; i < *edgeCount; i++) {
int u, v;
        printf("Enter edge (u, v): ");
scanf("%d %d", &u, &v);
        if (u < *vertexCount && v < *vertexCount) {
adjMatrix[u][v] = 1;
            adjMatrix[v][u] = 1;
        } else {
            printf("Invalid edge: (%d, %d)\n", u, v);
i--;
        }
    }
}

int main() {
    int vertexCount, edgeCount, startVertex;

    initializeGraph(&vertexCount, &edgeCount);

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

    if (startVertex >= 0 && startVertex < vertexCount) {
printf("Depth-First Search starting from vertex %d:\n",
startVertex);
        depthFirstSearch(startVertex, vertexCount);
    } else {
        printf("Invalid starting vertex.\n");
    }

    return 0;
}
```

Output:

```
Enter the number of vertices: 5
Enter the number of edges: 5
Enter edge (u, v): 0 1
Enter edge (u, v): 0 3
Enter edge (u, v): 1 2
Enter edge (u, v): 1 4
Enter edge (u, v): 2 4
Enter the starting vertex: 1
Depth-First Search starting from vertex 1:
1 0 3 2 4

Enter the number of vertices: 5
Enter the number of edges: 5
Enter edge (u, v): 1 0
Enter edge (u, v): 3 0
Enter edge (u, v): 2 1
Enter edge (u, v): 4 1
Enter edge (u, v): 4 2
Enter the starting vertex: 4
Depth-First Search starting from vertex 4:
4 1 0 3 2

=== Code Execution Successful ===
```