Practical 8: Implementation of Graph

Name: Sutariya Savankumar

Roll no: MA065

1. **Write a program to implement an undirected graph with the following.**
   - **➔ Create an adjacency matrix.**
   - **➔ Create an adjacency List.**
   - **➔ Print the information of the graph such as number of edges, edges list, degree of each vertex. (using both matrix and list)**
   - **➔ implement traversal of graph using DFS (using both matrix and list)**
   - **➔ implement traversal of graph using BFS. (using both matrix and list)**

Code

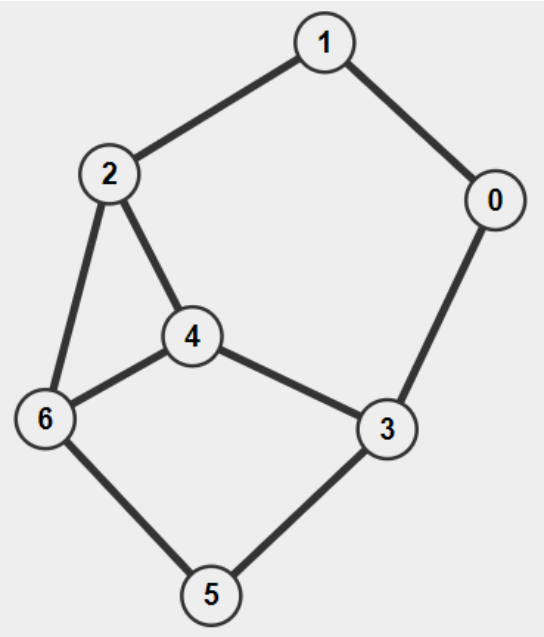
```
#include <stdio.h>
#include <stdlib.h>
#define max_vert 50
int adjmat[max_vert][max_vert];
typedef struct node{
    int vertex;
    struct node *next;
} Node;
Node *adjList[max_vert];
int numVertices = 0, numEdges = 0;
void addEdge(int src, int dest){
    adjmat[src][dest] = 1;
    adjmat[dest][src] = 1;
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->vertex = dest;
    newNode->next = adjList[src];
    adjList[src] = newNode;
    newNode = (Node *)malloc(sizeof(Node));
    newNode->vertex = src;
    newNode->next = adjList[dest];
    adjList[dest] = newNode;
    numEdges++;
}
void dfsMatrix(int vertex, int visited[]){
    visited[vertex] = 1;
    printf("%d ", vertex);
    for (int i = 0; i < numVertices; i++){
        if (adjmat[vertex][i] && !visited[i]){
            dfsMatrix(i, visited);
        }
    }
}
```

```
void dfsList(int vertex, int visited[]){
    visited[vertex] = 1;
    printf("%d ", vertex);
    Node *currNode = adjList[vertex];
    while (currNode != NULL){
        int adjVertex = currNode->vertex;
        if (!visited[adjVertex]){
            dfsList(adjVertex, visited);
        }
        currNode = currNode->next;
    }
}
void bfsMatrix(int startVertex){
    int visited[max_vert] = {0};
    int queue[max_vert];
    int front = -1, rear = -1;
    visited[startVertex] = 1;
    queue[++rear] = startVertex;
    while (front != rear){
        int vertex = queue[++front];
        printf("%d ", vertex);
        for (int i = 0; i < numVertices; i++){
            if (adjmat[vertex][i] && !visited[i]){
                visited[i] = 1;
                queue[++rear] = i;
            }
        }
    }
}
void bfsList(int startVertex){
    int visited[max_vert] = {0};
    int queue[max_vert];
    int front = -1, rear = -1;
    visited[startVertex] = 1;
    queue[++rear] = startVertex;
    while (front != rear){
        int vertex = queue[++front];
        printf("%d ", vertex);
        Node *currNode = adjList[vertex];
        while (currNode != NULL){
            int adjVertex = currNode->vertex;
            if (!visited[adjVertex]){
                visited[adjVertex] = 1;
                queue[++rear] = adjVertex;
            }
            currNode = currNode->next;
        }
    }
}
void printGraph(){
    printf("Number of vertices: %d\n", numVertices);
    printf("Number of edges: %d\n", numEdges);
    printf("Edges list:\n");
    for (int i = 0; i < numVertices; i++){
        Node *currNode = adjList[i];
        while (currNode != NULL){
```

```
            if (i < currNode->vertex){
                printf("%d - %d\n", i, currNode->vertex);
            }
            currNode = currNode->next;
        }
    }

    printf("Adjacency Matrix:\n ");
    for (int i = 0; i < numVertices; i++){
        printf("%d ", i);
    }
    printf("\n");

    for (int i = 0; i < numVertices; i++){
        printf("%d: ", i);
        for (int j = 0; j < numVertices; j++){
            printf("%d ", adjmat[i][j]);
        }
        printf("\n");
    }
}
void createGraph(){
    numVertices = 7;
    for (int i = 0; i < numVertices; i++){
        adjList[i] = NULL;
    }
    addEdge(0, 1);
    addEdge(0, 3);
    addEdge(1, 2);
    addEdge(2, 4);
    addEdge(2, 6);
    addEdge(3, 4);
    addEdge(3, 5);
    addEdge(4, 6);
    addEdge(5, 6);
}
int main(){
    createGraph();
    printf("Undirected Graph:\n");
    printf(" DFS using adjacency matrix:\n");
    int visited[max_vert] = {0};
    dfsMatrix(1, visited);
    printf("\n");
    printf("DFS using adjacency list:\n");
    int visitedList[max_vert] = {0};
    dfsList(1, visitedList);
    printf("\n");
    printf("BFS using adjacency matrix:\n");
    bfsMatrix(1);
    printf("\n");
    printf("BFS using adjacency list:\n");
    bfsList(1);
    printf("\n");
    printGraph();
    return 0;
}
```

Output

```
Savan@Savan MINGW64 /c/Drive/Study/MCA/DSsubmission/MA065_Savan (master)
$ gcc prac08-01.c -o p1

Savan@Savan MINGW64 /c/Drive/Study/MCA/DSsubmission/MA065_Savan (master)
$ ./p1
Undirected Graph:
 DFS using adjacency matrix:
1 0 3 4 2 6 5
DFS using adjacency list:
1 2 6 5 3 4 0
BFS using adjacency matrix:
1 0 2 3 4 6 5
BFS using adjacency list:
1 2 0 6 4 3 5
Number of vertices: 7
Number of edges: 9
Edges list:
0 - 3
0 - 1
1 - 2
2 - 6
2 - 4
3 - 5
3 - 4
4 - 6
5 - 6
Adjacency Matrix:
 0 1 2 3 4 5 6
0: 0 1 0 1 0 0 0
1: 1 0 1 0 0 0 0
2: 0 1 0 0 1 0 1
3: 1 0 0 0 1 1 0
4: 0 0 1 1 0 0 1
5: 0 0 0 1 0 0 1
6: 0 0 1 0 1 1 0
```



2. **Write a program to implement a directed graph with the following.**
   - **➔ Create an adjacency matrix.**
   - **➔ Create an adjacency List.**
   - **➔ Print the information of the graph such as number of edges, edges list, degree of each vertex. (using both matrix and list)**
   - **➔ implement traversal of graph using DFS (using both matrix and list)**
   - **➔ implement traversal of graph using BFS. (using both matrix and list)**Code

```
#include <stdio.h>
#include <stdlib.h>
#define max 50
int adjmat[max][max];
typedef struct node{
    int vertex;
    struct node *next;
} Node;
Node *adjList[max];
int numVertices = 0, numEdges = 0;
void addEdge(int src, int dest){
    adjmat[src][dest] = 1;
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->vertex = dest;
    newNode->next = adjList[src];
    adjList[src] = newNode;
    numEdges++;
}
void dfsMatrix(int vertex, int visited[]){
    visited[vertex] = 1;
    printf("%d ", vertex);
    for (int i = 0; i < numVertices; i++){
        if (adjmat[vertex][i] && !visited[i]){
            dfsMatrix(i, visited);
        }
    }
}
void dfsList(int vertex, int visited[]){
    visited[vertex] = 1;
    printf("%d ", vertex);
    Node *currNode = adjList[vertex];
    while (currNode != NULL){
        int adjVertex = currNode->vertex;
        if (!visited[adjVertex]){
            dfsList(adjVertex, visited);
        }
        currNode = currNode->next;
    }
```

```
}
void bfsMatrix(int startVertex){
    int visited[max] = {0};
    int queue[max];
    int front = -1, rear = -1;
    visited[startVertex] = 1;
    queue[++rear] = startVertex;
    while (front != rear){
        int vertex = queue[++front];
        printf("%d ", vertex);
        for (int i = 0; i < numVertices; i++){
            if (adjmat[vertex][i] && !visited[i]){
                visited[i] = 1;
                queue[++rear] = i;
            }
        }
    }
}
void bfsList(int startVertex){
    int visited[max] = {0};
    int queue[max];
    int front = -1, rear = -1;
    visited[startVertex] = 1;
    queue[++rear] = startVertex;
    while (front != rear){
        int vertex = queue[++front];
        printf("%d ", vertex);
        Node *currNode = adjList[vertex];
        while (currNode != NULL){
            int adjVertex = currNode->vertex;
            if (!visited[adjVertex]){
                visited[adjVertex] = 1;
                queue[++rear] = adjVertex;
            }
            currNode = currNode->next;
        }
    }
}
void printGraph(){
    printf("Number of vertices: %d\n",numVertices);
    printf("Number of edges: %d\n",numEdges);
    printf("Edges list:\n");
    for (int i = 0; i < numVertices; i++){
        Node *currNode = adjList[i];
        if(currNode != NULL){
            printf("\n%d: ", currNode->vertex);
            while (currNode != NULL){
                printf("-> %d", i,currNode->vertex);
                currNode = currNode->next;
            }
        }
```

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

    for (int i = 0; i < numVertices; i++){
        printf("%d: ", i);
        for (int j = 0; j < numVertices; j++){
            printf("%d ", adjmat[i][j]);
        }
        printf("\n");
    }
}
void createGraph(){
    numVertices = 7;
    for (int i = 0; i < numVertices; i++){
        adjList[i] = NULL;
    }
    addEdge(0, 1);
    addEdge(0, 3);
    addEdge(1, 2);
    addEdge(2, 4);
    addEdge(2, 6);
    addEdge(3, 4);
    addEdge(3, 5);
    addEdge(4, 6);
    addEdge(5, 6);
}
int main(){
    createGraph();
    printf(" (DFS using adjacency matrix:\n");
    int visited[max] = {0};
    dfsMatrix(1, visited);
    printf("\n");
    printf("DFS using adjacency list:\n");
    int visitedList[max] = {0};
    dfsList(1, visitedList);
    printf("\n");
    printf("BFS using adjacency matrix:\n");
    bfsMatrix(1);
    printf("\n");
    printf("BFS using adjacency list:\n");
    bfsList(1);
    printf("\n");
    printGraph();
    return 0;
}
```

Output

```
Savan@Savan MINGW64 /c/Drive/Study/MCA/DSsubmission/MA065_Savan (master)
$ ./p2
 (DFS using adjacency matrix:
1 2 4 6
DFS using adjacency list:
1 2 6 4
BFS using adjacency matrix:
1 2 4 6
BFS using adjacency list:
1 2 6 4
Number of vertices: 7
Number of edges: 9
Edges list:

3: -> 0-> 0
2: -> 1
6: -> 2-> 2
5: -> 3-> 3
6: -> 4
6: -> 5Adjacency Matrix:
 0 1 2 3 4 5 6
0: 0 1 0 1 0 0 0
1: 0 0 1 0 0 0 0
2: 0 0 0 0 1 0 1
3: 0 0 0 0 1 1 0
4: 0 0 0 0 0 0 1
5: 0 0 0 0 0 0 1
6: 0 0 0 0 0 0 0
```