## LAB 14

## Static Implementation of Graph and graph traversal(DFS with recursion, without recursion and BFS)


## Mark lopes

## SE-Comps-A

## 9913


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

#define V 50  // Maximum number of vertices

// Define a stack data structure
typedef struct {
    int b[V];
    int tos;
} Stack;

// Define a queue data structure
typedef struct {
    int a[V];
    int front, rear;
} Queue;

// Define a graph data structure
typedef struct {
    int adj[V][V];  // Adjacency matrix
    int e, v;       // Number of edges (e) and vertices (v)
} Graph;

// Function to check if the stack is empty
int isEmpty(Stack s) {
    return (s.tos == -1);
}

// Function to dequeue from the queue
int dequeue(Queue *q) {
    int x = q->a[q->front];
    if (q->front == q->rear) {
        q->front = q->rear = -1;
    } else {
        q->front++;
    }
    return x;
}
```

```c
// Function to enqueue into the queue
void enqueue(Queue *q, int e) {
    q->rear++;
    q->a[q->rear] = e;
    if (q->front == -1) {
        q->front = 0;
    }
}

// Function to push onto the stack
void push(Stack *s, int e) {
    s->tos++;
    s->b[s->tos] = e;
}

// Function to pop from the stack
int pop(Stack *s) {
    int x = s->b[s->tos];
    s->tos--;
    return x;
}

// Function to peek at the top of the stack
int peek(Stack *s) {
    return s->b[s->tos];
}

// Function to initialize the graph
void initializeGraph(Graph *g) {
    for (int i = 0; i < g->v; i++) {
        for (int j = 0; j < g->v; j++) {
            g->adj[i][j] = 0;  // Initialize adjacency matrix to all zeros
        }
    }
}

// Function to add an edge to the graph
void add(Graph *g, int src, int dest) {
    g->adj[src][dest] = 1;
    g->adj[dest][src] = 1;  // For an undirected graph, set both directions to
1
}

// Function for DFS traversal using recursion
void dfsRecursion(int root, Graph g, int visited[]) {
    visited[root] = 1;  // Mark the node as visited
    printf("%d ", root);  // Print the visited node
```

```
    for (int j = 0; j < g.v; j++) {
        if (g.adj[root][j] == 1 && visited[j] != 1) {
            // If there is a connection from root to j and j is not visited,
recursively visit j
            dfsRecursion(j, g, visited);
        }
    }
}

// Function for DFS traversal using iteration
void dfsIterative(int root, Graph g, int visited[]) {
    Stack s;
    s.tos = -1;
    push(&s, root);  // Push the root node onto the stack

    while (!isEmpty(s)) {
        int x = pop(&s);

        if (visited[x] != 1) {
            visited[x] = 1;  // Mark the node as visited
            printf("%d\t", x);  // Print the visited node

            for (int i = 0; i < g.v; i++) {
                if (g.adj[x][i] == 1 && visited[i] != 1) {
                    push(&s, i);  // Push unvisited neighbors onto the stack
                }
            }
        }
    }
}

// Function for BFS traversal using iteration
void bfsIterative(int root, Graph g, int visited[]) {
    Queue q;
    q.front = q.rear = -1;
    enqueue(&q, root);  // Enqueue the root node

    visited[root] = 1;

    while (q.front != -1) {
        int x = dequeue(&q);
        printf("%d\t", x);  // Print the visited node

        for (int i = 0; i < g.v; i++) {
            if (g.adj[x][i] == 1 && visited[i] != 1) {
                enqueue(&q, i);  // Enqueue unvisited neighbors
                visited[i] = 1;  // Mark the node as visited
```

```
            }
        }
    }
}

// Function to print the adjacency matrix of the graph
void printAdjMatrix(Graph g) {
    for (int i = 0; i < g.v; i++) {
        for (int j = 0; j < g.v; j++) {
            printf("%d ", g.adj[i][j]);
        }
        printf("\n");
    }
}

// Function to reset the visited array
void resetVisitedArray(int visited[], int size) {
    for (int i = 0; i < size; i++) {
        visited[i] = 0;
    }
}

int main() {
    Graph g;
    int root, src, dest, c;

    printf("Enter the number of vertices for the directed graph: ");
    scanf("%d", &g.v);

    printf("Enter the number of edges for the directed graph: ");
    scanf("%d", &g.e);

    initializeGraph(&g);  // Initialize the graph

    for (int i = 1; i <= g.e; i++) {
        printf("Enter the source node value: ");
        scanf("%d", &src);
        printf("Enter the destination node value: ");
        scanf("%d", &dest);
        add(&g, src, dest);  // Add edges to the graph
    }

    printf("Printing the adjacency matrix:\n");
    printAdjMatrix(g);

    int visited[V] = {0};  // Initialize the visited array

    do {
```

```
        printf("\nChoose your operation:\n");
        printf("1. DFS Recursion\n");
        printf("2. DFS Iterative\n");
        printf("3. BFS Iterative\n");
        printf("4. Quit\n");

        printf("Enter your choice: ");
        scanf("%d", &c);

        resetVisitedArray(visited, g.v);  // Reset the visited array

        switch (c) {
            case 1:
                printf("Enter the root: ");
                scanf("%d", &root);
                printf("DFS Recursion:\n");
                dfsRecursion(root, g, visited);
                break;
            case 2:
                printf("Enter the root: ");
                scanf("%d", &root);
                printf("DFS Iterative:\n");
                dfsIterative(root, g, visited);
                break;
            case 3:
                printf("Enter the root: ");
                scanf("%d", &root);
                printf("BFS Iterative:\n");
                bfsIterative(root, g, visited);
                break;
            case 4:
                exit(0);
        }
    } while (1);

    return 0;
}
```

```
PS C:\Users\Mark Lopes\Desktop\college\ds\lab14>  & 'c:\Users\Mark Lop
uncher.exe' '--stdin=Microsoft-MIEngine-In-a0iq2rcv.hde' '--stdout=Mic
Microsoft-MIEngine-Pid-kkrf0drp.r2r' '--dbgExe=C:\msys64\mingw64\bin\g
Enter the number of vertices for the directed graph: 4
Enter the number of edges for the directed graph: 4
Enter the source node value: 0
Enter the destination node value: 1
Enter the source node value: 1
Enter the destination node value: 2
Enter the source node value: 2
Enter the destination node value: 3
Enter the source node value: 3
Enter the destination node value: 0
Printing the adjacency matrix:
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0

Choose your operation:
1. DFS Recursion
2. DFS Iterative
3. BFS Iterative
4. Quit
Enter your choice: 1
Enter the root: 0
DFS Recursion:
0 1 2 3
Choose your operation:
1. DFS Recursion
2. DFS Iterative
3. BFS Iterative
4. Quit
Enter your choice: 2
Enter the root: 0
DFS Iterative:
0       3       2       1
```

```
Choose your operation:
1. DFS Recursion
2. DFS Iterative
3. BFS Iterative
4. Quit
Enter your choice: 3
Enter the root: 0
BFS Iterative:
0       1       3       2
Choose your operation:
1. DFS Recursion
2. DFS Iterative
3. BFS Iterative
4. Quit
Enter your choice: 4
PS C:\Users\Mark Lopes\Desktop\college\ds\lab14>
```