Implementation of BFS and DFS

Breadth First Search (BFS)

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

#define MAX_VERTICES 50

// This struct represents a directed graph using
// adjacency list representation
typedef struct Graph_t {

    // No. of vertices
    int V;
    bool adj[MAX_VERTICES][MAX_VERTICES];
} Graph;

// Constructor
Graph* Graph_create(int V)
{
    Graph* g = malloc(sizeof(Graph));
    g->V = V;

    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            g->adj[i][j] = false;
        }
    }
```

```
    return g;
}

// Destructor
void Graph_destroy(Graph* g) { free(g); }

// Function to add an edge to graph
void Graph_addEdge(Graph* g, int v, int w)
{
    // Add w to v's list.
    g->adj[v][w] = true;
}

// Prints BFS traversal from a given source s
void Graph_BFS(Graph* g, int s)
{
    // Mark all the vertices as not visited
    bool visited[MAX_VERTICES];
    for (int i = 0; i < g->V; i++) {
        visited[i] = false;
    }

    // Create a queue for BFS
    int queue[MAX_VERTICES];
    int front = 0, rear = 0;

    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue[rear++] = s;

    while (front != rear) {
```

```
        // Dequeue a vertex from queue and print it
        s = queue[front++];
        printf("%d ", s);

        // Get all adjacent vertices of the dequeued
        // vertex s.
        // If an adjacent has not been visited,
        // then mark it visited and enqueue it
        for (int adjacent = 0; adjacent < g->V;
             adjacent++) {
            if (g->adj[s][adjacent] && !visited[adjacent]) {
                visited[adjacent] = true;
                queue[rear++] = adjacent;
            }
        }
    }
}

// Driver code
int main()
{
    // Create a graph
    Graph* g = Graph_create(4);
    Graph_addEdge(g, 0, 1);
    Graph_addEdge(g, 0, 2);
    Graph_addEdge(g, 1, 2);
    Graph_addEdge(g, 2, 0);
    Graph_addEdge(g, 2, 3);
    Graph_addEdge(g, 3, 3);
```

```
    printf("Following is Breadth First Traversal "
           "(starting from vertex 2) \n");
    Graph_BFS(g, 2);
    Graph_destroy(g);

    return 0;
}
```

Depth First Search (DFS)

```
// DFS algorithm in C

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

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

struct node* createNode(int v);

struct Graph {
  int numVertices;
```

```
  int* visited;

  // We need int** to store a two dimensional array.
  // Similary, we need struct node** to store an array of Linked lists
  struct node** adjLists;
};

// DFS algo
void DFS(struct Graph* graph, int vertex) {
  struct node* adjList = graph->adjLists[vertex];
  struct node* temp = adjList;

  graph->visited[vertex] = 1;
  printf("Visited %d \n", vertex);

  while (temp != NULL) {
    int connectedVertex = temp->vertex;

    if (graph->visited[connectedVertex] == 0) {
      DFS(graph, connectedVertex);
    }
    temp = temp->next;
  }
```

```
}

// Create a node
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}

// Create graph
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));

  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
```

```
  }
  return graph;
}

// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
  // Add edge from src to dest
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  // Add edge from dest to src
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}

// Print the graph
void printGraph(struct Graph* graph) {
  int v;
  for (v = 0; v < graph->numVertices; v++) {
    struct node* temp = graph->adjLists[v];
    printf("\n Adjacency list of vertex %d\n ", v);
```

```
    while (temp) {
      printf("%d -> ", temp->vertex);
      temp = temp->next;
    }
    printf("\n");
  }
}

int main() {
  struct Graph* graph = createGraph(4);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
  addEdge(graph, 1, 2);
  addEdge(graph, 2, 3);

  printGraph(graph);

  DFS(graph, 2);

  return 0;
}
```