ASSIGNMENT-1 MINI PROJECT

Title of The Project: Path Finding Algorithm

Group No: 17

Name: Rahul(4MW24CS112)
Mahalingappa (4MW24CS074)

Abstract:

Pathfinding is an important problem in computer science and has wide applications in navigation systems, computer networks, robotics, and artificial intelligence. This mini project focuses on implementing pathfinding techniques using Graph data structures in the C programming language. The graph is represented using an adjacency list, and traversal algorithms such as Breadth First Search (BFS) and Depth First Search (DFS) are used to explore paths between nodes.

Application Description:

This project simulates a **pathfinding system** where locations are represented as vertices and paths between them are represented as edges in a graph. Such systems are widely used in real-world applications such as:

- GPS navigation systems to find routes between locations
- Network routing to determine data paths
- Robotics for movement planning
- Game development for character navigation

The application allows the user to interactively build a graph and perform BFS and DFS traversals to understand how paths are explored. BFS explores nodes level by level, making it useful for finding shortest paths in unweighted graphs, while DFS explores nodes deeply, useful for checking connectivity and traversal paths.

Algorithm:

Algorithm for Breadth First Search (BFS)

1. Start from the given source vertex.
2. Mark the source vertex as visited.
3. Insert the source vertex into the queue.
4. Remove a vertex from the queue and display it.
5. Visit all unvisited adjacent vertices, mark them as visited, and insert them into the queue.
6. Repeat the process until the queue becomes empty.

Algorithm for Depth First Search (DFS)

1. Start from the source vertex.
2. Mark the vertex as visited and display it.
3. Recursively visit all unvisited adjacent vertices.
4. Continue the process until all reachable vertices are visited.

Methodology:

1. The problem of pathfinding was analyzed and identified as a graph-based problem.
2. A graph was implemented using adjacency list representation for efficient memory usage.
3. BFS was implemented using a queue to explore nodes in a level-wise manner.
4. DFS was implemented using recursion to explore nodes in a depth-wise manner.
5. A menu-driven interface was developed to allow user interaction.
6. The program was tested with various inputs to handle edge cases such as invalid vertices and empty graphs.

Source Code:

```
#include <stdio.h>

#include <stdlib.h>


#define MAX 20


/* Structure for adjacency list node */

struct Node

{

   int vertex;

   struct Node *next;

};


/* Graph structure */

struct Graph

{

   int vertices;

   struct Node *adjList[MAX];

};


/* Queue for BFS */

int queue[MAX];
```

```
int front = -1, rear = -1;

/* Function to create graph */
struct Graph *createGraph(int vertices)
{
   struct Graph *graph = (struct Graph *)malloc(sizeof(struct Graph));
   graph->vertices = vertices;

   for (int i = 0; i < vertices; i++)
      graph->adjList[i] = NULL;

   return graph;
}

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

/* Add edge to graph (Undirected) */
```

```
void addEdge(struct Graph *graph, int src, int dest)
{
    if (src >= graph->vertices || dest >= graph->vertices || src < 0 || dest < 0)
    {
        printf("Invalid vertex!\n");
        return;
    }

    struct Node *newNode = createNode(dest);
    newNode->next = graph->adjList[src];
    graph->adjList[src] = newNode;

    newNode = createNode(src);
    newNode->next = graph->adjList[dest];
    graph->adjList[dest] = newNode;

    printf("Edge added successfully.\n");
}

/* Display graph */
void displayGraph(struct Graph *graph)
{
    for (int i = 0; i < graph->vertices; i++)
    {
```

```
        struct Node *temp = graph->adjList[i];
        printf("Vertex %d: ", i);
        while (temp)
        {
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

/* Queue operations */
void enqueue(int value)
{
    if (rear == MAX - 1)
        return;

    if (front == -1)
        front = 0;
    queue[++rear] = value;
}

int dequeue()
{
```

```
    if (front == -1)
        return -1;
    return queue[front++];
}

/* BFS traversal */
void BFS(struct Graph *graph, int start)
{
    int visited[MAX] = {0};

    printf("BFS Traversal: ");

    enqueue(start);
    visited[start] = 1;

    while (front <= rear)
    {
        int current = dequeue();
        printf("%d ", current);

        struct Node *temp = graph->adjList[current];
        while (temp)
        {
            int adj = temp->vertex;
```

```
            if (!visited[adj])
            {
                visited[adj] = 1;
                enqueue(adj);
            }
            temp = temp->next;
        }
    }
    printf("\n");

    front = rear = -1;
}

/* DFS traversal */
void DFSUtil(struct Graph *graph, int vertex, int visited[])
{
    visited[vertex] = 1;
    printf("%d ", vertex);

    struct Node *temp = graph->adjList[vertex];
    while (temp)
    {
        int adj = temp->vertex;
        if (!visited[adj])
```

```
            DFSUtil(graph, adj, visited);
        temp = temp->next;
    }
}

void DFS(struct Graph *graph, int start)
{
    int visited[MAX] = {0};
    printf("DFS Traversal: ");
    DFSUtil(graph, start, visited);
    printf("\n");
}

/* Main function */
int main()
{
    struct Graph *graph = NULL;
    int choice, vertices, src, dest, start;

    do
    {
        printf("\n----- PATHFINDING USING GRAPHS -----\n");
        printf("1. Create Graph\n");
        printf("2. Add Edge\n");
```

```
printf("3. Display Graph\n");
printf("4. BFS Traversal\n");
printf("5. DFS Traversal\n");
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice)
{
case 1:
   printf("Enter number of vertices: ");
   scanf("%d", &vertices);
   graph = createGraph(vertices);
   printf("Graph created successfully.\n");
   break;

case 2:
   if (!graph)
   {
      printf("Create graph first!\n");
      break;
   }
   printf("Enter source and destination: ");
   scanf("%d %d", &src, &dest);
```

```
        addEdge(graph, src, dest);
        break;

    case 3:
        if (!graph)
        {
            printf("Graph not created!\n");
            break;
        }
        displayGraph(graph);
        break;

    case 4:
        if (!graph)
        {
            printf("Graph not created!\n");
            break;
        }
        printf("Enter starting vertex for BFS: ");
        scanf("%d", &start);
        BFS(graph, start);
        break;

    case 5:
```

```
            if (!graph)
            {
               printf("Graph not created!\n");
               break;
            }
            printf("Enter starting vertex for DFS: ");
            scanf("%d", &start);
            DFS(graph, start);
            break;

         case 6:
            printf("Exiting program...\n");
            break;

         default:
            printf("Invalid choice!\n");
         }
      } while (choice != 6);

      return 0;
   }
```

Results and Discussion:

1.Number of Vertices

The user first inputs the number of vertices in the graph. Each vertex represents a location or node in the pathfinding system.

```
----- PATHFINDING USING GRAPHS -----
1. Create Graph
2. Add Edge
3. Display Graph
4. BFS Traversal
5. DFS Traversal
6. Exit
Enter your choice: 1
Enter number of vertices (max 20): 5
Graph created successfully.
```

2.Edge Input (Source and Destination)

The user enters pairs of vertices to define edges between them. These edges represent paths connecting different locations. The graph is implemented as an undirected graph using adjacency lists.

```
Enter your choice: 2
Enter source and destination: 2 4
Edge added successfully.
```

```
Enter your choice: 2
Enter source and destination: 1 3
Edge added successfully.
```

```
Enter your choice: 2
Enter source and destination: 0 2
Edge added successfully.
```

```
Enter your choice: 2
Enter source and destination: 0 1
Edge added successfully.
```

3.Starting Vertex for Traversal

The user provides a starting vertex from which BFS or DFS traversal begins.

```
Enter your choice: 3
Vertex 0: 2 -> 1 -> NULL
Vertex 1: 3 -> 0 -> NULL
Vertex 2: 4 -> 0 -> NULL
Vertex 3: 1 -> NULL
Vertex 4: 2 -> NULL
```

```
Enter your choice: 4
Enter starting vertex for BFS: 0
BFS Traversal: 0 2 1 4 3
```

```
Enter your choice: 5
Enter starting vertex for DFS: 0
DFS Traversal: 0 2 4 1 3
```