Date: 05-01-2023
Ex no: 7

Name: M.Rohith
3122 21 5001 085

SSN College of Engineering

Department of Computer Science and Engineering

UCS2312 – Data Structures Lab

<u>II Year CSE - B Section (III Semester)</u>

Academic Year 2022-23

<u>Staff Incharge: Dr.H. Shahul Hamead</u>

<u>Exercise-7: Exercises on BFS, DFS, Shortest Path</u>

<u>Aim:</u>

To implement C program in Data structures using the concept of BFS, DFS, Shortest Path (Single source and all-pair)

Date: 05-01-2023 | Name: M.Rohith
Ex no: 7 | 3122 21 5001 085

BFS

Pseudocode:

05.01.2023

Exercises - 7 BFS, DFS, Shortest Path.

BFS Algorithm.

```
enqueue ( )
Input: Queue *queue, int item.
Output: void.
queue->items[queue->rear] = item
queue->rear = (queue->rear +1) % MAX_vertices

dequeue()
Input: Queue *queue.
Output: int.
int item = queue->items[queue->front];
queue->front = (queue->front + 1) % MAX_vertices.
return item

isEmpty()
Input: Queue *queue.
Output: int.
return queue->front == queue->rear.

BFS ()
Input: Graph *graph, int startVertex, int visited[]
Output: void.
Queue queue
queue.front = queue.rear = 0
enqueue(&queue, startVertex)
visited[startVertex] = 1
```

Name: M.Rohith
3122 21 5001 085

```
while(!isEmpty(&queue))
    int currentVertex = dequeue(&queue)
    print currentVertex
    for i=0 to graph->numVertices
        if (graph->adjacencyMatrix[currentVertex][i]
              && !visted[i] =1
        end if
    end for
end loop.

main()
Graph graph;
graph.numVertices=4;
graph.adjacencyMatrix[0][1]=1;
graph.adjacencyMatrix[0][2]=1;
graph.adjacencyMatrix[1][0]=1
graph.adjacencyMatrix[1][3]=1
graph.adjacencyMatrix[2][0]=1
graph.adjacencyMatrix[2][3]=1
graph.adjacencyMatrix[3][1]=1
graph.adjacencyMatrix[3][2]=1
int visited[MAX_VERTICES]
for ( int i=0 to i=graph.numVertices
      visited[i]=0
end for
print BFS
BFS(&graph, 0, visited)
```

DFS
DFS
Inp
Out
visi
pri
for
end
main
Gra
grap
grap
grap
grap
grap
grap
grap
grap
grap
int v
for
end
prin
DFS (

Date: 05-01-2023
Ex no: 7

Name: M.Rohith
3122 21 5001 085

Program code:

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

#define MAX_VERTICES 100

typedef struct Graph {
  int numVertices;
  int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
} Graph;

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

void enqueue(Queue* queue, int item) {
  queue->items[queue->rear] = item;
  queue->rear = (queue->rear + 1) % MAX_VERTICES;
}

int dequeue(Queue* queue) {
  int item = queue->items[queue->front];
  queue->front = (queue->front + 1) % MAX_VERTICES;
  return item;
}

int isEmpty(Queue* queue) {
  return queue->front == queue->rear;
}

void BFS(Graph* graph, int startVertex, int visited[]) {
  Queue queue;
  queue.front = queue.rear = 0;

  enqueue(&queue, startVertex);
  visited[startVertex] = 1;

  while (!isEmpty(&queue)) {
    int currentVertex = dequeue(&queue);
    printf("%d ", currentVertex);
```

```
    for (int i = 0; i < graph->numVertices; i++) {
      if (graph->adjacencyMatrix[currentVertex][i] && !visited[i]) {
        enqueue(&queue, i);
        visited[i] = 1;
      }
    }
  }
}

int main(void) {
  Graph graph;
  graph.numVertices = 4;

  // Assume that the graph is an adjacency matrix representation
  // and the graph is as follows:
  // 0 1 2 3
  // ----------
  // 0 1 1 0
  // 1 0 0 1
  // 1 0 0 1
  // 0 1 1 0
  graph.adjacencyMatrix[0][1] = 1;
  graph.adjacencyMatrix[0][2] = 1;
  graph.adjacencyMatrix[1][0] = 1;
  graph.adjacencyMatrix[1][3] = 1;
  graph.adjacencyMatrix[2][0] = 1;
  graph.adjacencyMatrix[2][3] = 1;
  graph.adjacencyMatrix[3][1] = 1;
  graph.adjacencyMatrix[3][2] = 1;

  // Initialize the visited array
  int visited[MAX_VERTICES];
  for (int i = 0; i < graph.numVertices; i++) {
    visited[i] = 0;
  }

  printf("BFS: ");
  BFS(&graph, 0, visited);

  return 0;
}
```

Date: 05-01-2023

Ex no: 7

Name: M.Rohith

3122 21 5001 085

Output:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7> gcc BFS.c -o run
PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7> ./run
BFS: 0 1 2 3
PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7>
```

Date: 05-01-2023
Ex no: 7

Name: M.Rohith
3122 21 5001 085

DFS

Pseudocode:

DFS Algorithm

```
DFS()
Input: Graph *graph, int startVertex, int visited[]
Output: void
visited[startVertex]=1
print(startVertex)
for i=0 to graph->numVertices
    if (graph->adjacencyMatrix[startVertex][i] && !visited[i])
        DFS(graph, i, visited)
    end if
end loop.

main()

Graph graph
graph numVertices=4
graph.adjacencyMatrix[0][1]=1
graph.adjacencyMatrix[0][2]=1
graph.adjacencyMatrix[1][0]=1
graph.adjacencyMatrix[1][3]=1
graph.adjacencyMatrix[2][0]=1
graph.adjacencyMatrix[2][3]=1
graph.adjacencyMatrix[3][1]=1
graph.adjacencyMatrix[3][2]=1
int visited[MAX_VERTICES]
for i=0 to graph.numVertices
    visited[i]=0
end loop
print DFS
DFS(&graph, 0, visited)
```

Name: M.Rohith
3122 21 5001 085

Program code:

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

#define MAX_VERTICES 100

typedef struct Graph {
  int numVertices;
  int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
} Graph;

void DFS(Graph* graph, int startVertex, int visited[]) {
  visited[startVertex] = 1;
  printf("%d ", startVertex);

  for (int i = 0; i < graph->numVertices; i++) {
    if (graph->adjacencyMatrix[startVertex][i] && !visited[i]) {
      DFS(graph, i, visited);
    }
  }
}

int main(void) {
  Graph graph;
  graph.numVertices = 4;

  // Assume that the graph is an adjacency matrix representation
  // and the graph is as follows:
  // 0 1 2 3
  // ----------
  // 0 1 1 0
  // 1 0 0 1
  // 1 0 0 1
  // 0 1 1 0
  graph.adjacencyMatrix[0][1] = 1;
  graph.adjacencyMatrix[0][2] = 1;
  graph.adjacencyMatrix[1][0] = 1;
  graph.adjacencyMatrix[1][3] = 1;
  graph.adjacencyMatrix[2][0] = 1;
  graph.adjacencyMatrix[2][3] = 1;
  graph.adjacencyMatrix[3][1] = 1;
```

```
  graph.adjacencyMatrix[3][2] = 1;

  // Initialize the visited array
  int visited[MAX_VERTICES];
  for (int i = 0; i < graph.numVertices; i++) {
    visited[i] = 0;
  }

  printf("DFS: ");
  DFS(&graph, 0, visited);

  return 0;
}
```

Output:

```
PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7> gcc DFS.c -o run
PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7> ./run
DFS: 0 1 3 2
PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7>
```

Date: 05-01-2023 Name: M.Rohith
Ex no: 7 3122 21 5001 085

Shortest Path

Dijkstra algorithm (Single source shortest path)

Pseudocode:

Dijkstra Algorithm (Single source shortest path)

```
minDistance()
int min = INT_MAX, min_index;
Input : int dist[ ], bool sptset[ ]
Output: int
for v=0 to V
    if (sptset[v] == false && dist[v] <= min)
        min = dist[v]
        min_index = v
    end if
end loop
return min_index

printSolution()
Input: int dist[ ]
Output: void.
for i=0 to V
    print i, dist[i]
end loop

dijkstra()
Input: int graph[V][V], int src
Output: void.
int dist[V]
bool sptSet[V]
for i=0 to V
    dist[i] = INT_MAX
    sptSet[i] = false
end loop
dist[src] = 0
```

Date: 05-01-2023
Ex no: 7

Name: M.Rohith
3122 21 5001 085

```
for (count=0 to count<V-1
    u = minDistance (dist, sptSet)
    sptSet[u] = true
    for v=0 to v<V
        if (! sptSet[v] && graph[u][v] &&
            dist[u] != INT_MAX && dist[u] +
            graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v]
        end if
    end loop
end loop
printSolution(dist).

main()

int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},
                   {4, 0, 8, 0, 0, 0, 0, 11, 0},
                   {0, 8, 0, 7, 0, 4, 0, 0, 2}
                   {0, 0, 0, 7, 0, 9, 14, 0, 0, 0}
                   {0, 0, 0, 9, 0, 10, 0, 0, 0}
                   {0, 0, 4, 14, 10, 0, 2, 0, 0}
                   {0, 0, 0, 0, 0, 2, 0, 1, 6},
                   {8, 11, 0, 0, 0, 0, 1, 0, 7}
                   {0, 0, 2, 0, 0, 0, 6, 7, 0}}
dijkstra(graph, 0)
```

Date: 05-01-2023
Ex no: 7

Name: M.Rohith
3122 21 5001 085

Program code:

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

// Number of vertices in the graph
#define V 9

// A utility function to find the vertex with minimum
// distance value, from the set of vertices not yet included
// in shortest path tree
int minDistance(int dist[], bool sptSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// A utility function to print the constructed distance
// array
void printSolution(int dist[])
{
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t\t\t %d\n", i, dist[i]);
}

// Function that implements Dijkstra's single source
// shortest path algorithm for a graph represented using
// adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // The output array.  dist[i] will hold the
                 // shortest
    // distance from src to i

```

Name: M.Rohith
3122 21 5001 085

```
    bool sptSet[V]; // sptSet[i] will be true if vertex i is
                    // included in shortest
    // path tree or shortest distance from src to i is
    // finalized

    // Initialize all distances as INFINITE and stpSet[] as
    // false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of
        // vertices not yet processed. u is always equal to
        // src in the first iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the
        // picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if is not in sptSet,
            // there is an edge from u to v, and total
            // weight of path from src to  v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v]
                && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist);
}

// driver's code
```

Name: M.Rohith
3122 21 5001 085

```c
int main()
{
    /* Let us create the example graph discussed above */
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    // Function call
    dijkstra(graph, 0);

    return 0;
}
```

Output:

```
PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7> gcc Dijkstra.c -o run
PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7> ./run
Vertex           Distance from Source
0                                0
1                                4
2                                12
3                                19
4                                21
5                                11
6                                9
7                                8
8                                14
PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7>
```

Date: 05-01-2023 Name: M.Rohith
Ex no: 7 3122 21 5001 085

Floyd Warshell Algorithm (All-pair shortest path)

Pseudocode:

Floyd Warshall Algorithm (All-pair shortest path)

```
floydWarshall()

Input: int dist[][V]
Variables: int i,j,k
for k=0 to k<V
    for i=0 to i<V
        for j=0 to j<V
            if (dist[i][k]+ dist[k][j]< dist[i][j])
                dist[i][j]= dist[i][k]+dist[k][j]
            end if
        end loop
    end loop
end loop
printSolution(dist)

printSolution()

Input: int dist[][V]
Output: void
for i=0 to i<V
    for j=0 to j<V
        if (dist[i][j]==INF)
            print INF
        end if
        else
            print dist[i][j]
        end else
    end loop
end loop
```

Name: M.Rohith
3122 21 5001 085

th)

main()

graph[V][V] = { {0, 5, INF, 10},
{INF, 0, 3, INF},
{INF, INF, 0, 1},
{INF, INF, INF, 0} };

floydWarshall(graph)

[i][j])

] [j]

Date: 05-01-2023

Ex no: 7

Name: M.Rohith

3122 21 5001 085

Program code:

```
#include <stdio.h>

// Number of vertices in the graph
#define V 4

/* Define Infinite as a large enough
  value. This value will be used
  for vertices not connected to each other */
#define INF 99999

// A function to print the solution matrix
void printSolution(int dist[][V]);

// Solves the all-pairs shortest path
// problem using Floyd Warshall algorithm
void floydWarshall(int dist[][V])
{
  int i, j, k;

    /* Add all vertices one by one to
      the set of intermediate vertices.
      ---> Before start of an iteration, we
      have shortest distances between all
      pairs of vertices such that the shortest
      distances consider only the
      vertices in set {0, 1, 2, .. k-1} as
      intermediate vertices.
      ----> After the end of an iteration,
      vertex no. k is added to the set of
      intermediate vertices and the set
      becomes {0, 1, 2, .. k} */
    for (k = 0; k < V; k++) {
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++) {
            // Pick all vertices as destination for the
            // above picked source
            for (j = 0; j < V; j++) {
                // If vertex k is on the shortest path from
                // i to j, then update the value of
                // dist[i][j]
```

Name: M.Rohith
3122 21 5001 085

```
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    // Print the shortest distance matrix
    printSolution(dist);
}

/* A utility function to print solution */
void printSolution(int dist[][V])
{
    printf(
        "The following matrix shows the shortest distances"
        " between every pair of vertices \n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INF)
                printf("%7s", "INF");
            else
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

// driver's code
int main()
{
    /* Let us create the following weighted graph
            10
       (0)------->(3)
        |         /|\
      5 |          |
        |          | 1
       \|/         |
       (1)------->(2)
            3           */
    int graph[V][V] = { { 0, 5, INF, 10 },
                        { INF, 0, 3, INF },
                        { INF, INF, 0, 1 },
```

```
                        { INF, INF, INF, 0 } };

    // Function call
    floydWarshall(graph);
    return 0;
}
```

Output:

```
PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7> gcc Floyd_Warshall.c -o run
PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7> ./run
The following matrix shows the shortest distances between every pair of vertices
      0      5      8      9
    INF      0      3      4
    INF    INF      0      1
    INF    INF    INF      0
PS C:\Rohith\Backup\Desktop\SEM 3\Data Structures in C\Assignment-7>
```

Result:

Hence C program using BFS, DFS and Shortest Path in data structure has been implemented to perform various operations.