DS LAB 14: Graph Traversal

Name : Vivian Vijay Ludrick

Branch: SE Comps A Batch C

Roll No: 9914

CODE:

#include <stdio.h>

#include <stdlib.h>

#define **MAX\_VERTICES** 50 *// Maximum number of vertices*

*// Define arr\_b stack data structure*

typedef struct

{

int arr\_a[**MAX\_VERTICES**];

int tos;

} Stack;

*// Define arr\_b queue data structure*

typedef struct

{

int arr\_b[**MAX\_VERTICES**];

int front, rear;

} Queue;

*// Define arr\_b graph data structure*

typedef struct

{

int adj[**MAX\_VERTICES**][**MAX\_VERTICES**]; *// Adjacency matrix*

int ele, vert; *// Number of edges (ele) and vertices (vert)*

} Graph;

*// check whether the stack is empty*

int **isEmpty**(Stack *stk*)

{

return (*stk*.tos == -1);

}

*// dequeue an element from the queue*

int **dequeue**(Queue \**q*)

{

int temp = *q*->arr\_b[*q*->front];

if (*q*->front == *q*->rear)

{

*q*->front = *q*->rear = -1;

}

else

{

*q*->front++;

}

return temp;

}

*// enqueue an element into the queue*

void **enqueue**(Queue \**q*, int *ele*)

{

*q*->rear++;

*q*->arr\_b[*q*->rear] = *ele*;

if (*q*->front == -1)

{

*q*->front = 0;

}

}

*// push an element onto the stack*

void **push**(Stack \**stk*, int *ele*)

{

*stk*->tos++;

*stk*->arr\_a[*stk*->tos] = *ele*;

}

*// pop an element from the stack*

int **pop**(Stack \**stk*)

{

int temp = *stk*->arr\_a[*stk*->tos];

*stk*->tos--;

return temp;

}

*// peek tos*

int **peek**(Stack \**stk*)

{

return *stk*->arr\_a[*stk*->tos]; *// returns the element at tos*

}

*// graph init*

void **initializeGraph**(Graph \**g*)

{

for (int i = 0; i < *g*->vert; i++)

{

for (int j = 0; j < *g*->vert; j++)

{

*g*->adj[i][j] = 0; *// let adjacency matrix have all the elements as 0*

}

}

}

*// add an edge to the graph*

void **add**(Graph \**g*, int *src*, int *dest*)

{

*g*->adj[*src*][*dest*] = 1;

*g*->adj[*dest*][*src*] = 1; *// undirected graph? both directions = 1:return;*

}

*// DFS using recursion*

void **dfsRecursion**(int *root*, Graph *g*, int *visited*[])

{

*visited*[*root*] = 1; *// Mark node as visited*

**printf**("%d ", *root*); *// Print visited node*

for (int j = 0; j < *g*.vert; j++)

{

if (*g*.adj[*root*][j] == 1 && *visited*[j] != 1)

{

*// root connected to j && j visited? return : loop till j visited;*

**dfsRecursion**(j, *g*, *visited*);

}

}

}

*// DFS traversal without recursion*

void **dfsIterative**(int *root*, Graph *g*, int *visited*[])

{

Stack stk;

stk.tos = -1;

**push**(&stk, *root*); *// push node onto stack*

while (!**isEmpty**(stk))

{

int x = **pop**(&stk);

if (*visited*[x] != 1)

{

*visited*[x] = 1; *// Mark node as visited*

**printf**("%d\t", x); *// Print visited node*

for (int i = 0; i < *g*.vert; i++)

{

if (*g*.adj[x][i] == 1 && *visited*[i] != 1)

{

**push**(&stk, i); *// Push not visited neighbors onto the stack*

}

}

}

}

}

*// BFS traversal non recursive*

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 visited node*

for (int i = 0; i < *g*.vert; i++)

{

if (*g*.adj[x][i] == 1 && *visited*[i] != 1)

{

**enqueue**(&q, i); *// Enqueue unvisited neighbors*

*visited*[i] = 1; *// Mark node as visited*

}

}

}

}

*// print the adjacency matrix of the graph*

void **printAdjMatrix**(Graph *g*)

{

for (int i = 0; i < *g*.vert; i++)

{

for (int j = 0; j < *g*.vert; j++)

{

**printf**("%d ", *g*.adj[i][j]);

}

**printf**("\n");

}

}

*// 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, option;

**printf**("Enter the no. of vertices for directed graph:\t");

**scanf**("%d", &g.vert);

**printf**("\nEnter the no. of edges for directed graph:\t");

**scanf**("%d", &g.ele);

**initializeGraph**(&g); *// Initialize the graph*

for (int i = 1; i <= g.ele; i++)

{

**printf**("Enter the source node value:\t");

**scanf**("%d", &src);

**printf**("Enter the destination node value:\t");

**scanf**("%d", &dest);

**add**(&g, src, dest); *// Add edges to the graph*

}

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

**printAdjMatrix**(g);

int visited[**MAX\_VERTICES**] = {0}; *// Initialize the visited array*

do

{

**printf**("\nSelect:\n");

**printf**("\t1. DFS Recursive\n");

**printf**("\t2. DFS Non recursive\n");

**printf**("\t3. BFS Non recursive\n");

**printf**("\t4. Exit\n");

**printf**("Enter your choice:\t");

**scanf**("%d", &option);

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

switch (option)

{

case 1:

**printf**("Enter root:\t");

**scanf**("%d", &*root*);

**printf**("DFS Recursion:\n");

**dfsRecursion**(root, g, visited);

break;

case 2:

**printf**("Enter root:\t");

**scanf**("%d", &*root*);

**printf**("DFS Iterative:\n");

**dfsIterative**(root, g, visited);

break;

case 3:

**printf**("Enter root:\t");

**scanf**("%d", &*root*);

**printf**("BFS Iterative:\n");

**bfsIterative**(root, g, visited);

break;

case 4:

**exit**(0);

}

} while (1);

return 0;

}

OUTPUT:

