**PROJECT REPORT**



**GRAPH STRUCTURE**

**SUBMITTED BY:**

**GROUP NUMBER:** 01

**NAMES:**

ADYASHA JHA - UCSE22003

ATREYEE DUTTA - UCSE22010

AYUSHMAN PANDA - UCSE22012

DEEPSHIKHA MOHANTY - UCSE22016

RENESA DAS - UCSE22028

SHREYA MATHUR- UCSE22036

**SCHOOL:** SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

**UNIVERSITY:** XIM UNIVERSITY

**DATE:** 29/11/2022

**COURSE NAME**: INTRODUCTION TO PROGRAMMING IN C

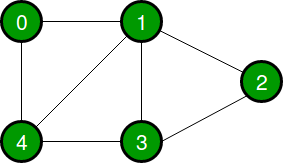
**INTRODUCTION :**

**A graph is a data structure that consists of the following two components:**

**1. A finite set of vertices also called as nodes.**

**2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not the same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight.**

**Following is an example of an undirected graph with 5 vertices.**



**The following two are the most commonly used representations of a graph.**

**1. Adjacency Matrix**

**2. Adjacency List**

**Operations that can be performed on graph structures are:**

**1. Find the adjacency matrix**

**2. Breadth-First Search and Depth-First Search**

**3. To find path from one vertex to another**

**4. Check if a graph contains a given value**

**5. Add or remove edge**

**6. Find indegree and outdegree of each node/vertex**

**DETAILED EXPLANATION OF THE FUNCTION AND PROBLEM EXPLANATION OF THE SOLUTION:**

**1. Find the adjacency matrix-**

To find the adjacency matrix of a graph , a directed and weighted graph is taken as input and in place of the actual values of edges 1 is saved and the resulting matrix is called adjacency matrix.

**2. Breadth -First Search and Depth-First Search-**

Breadth First Search is an algorithm used to search the Graph.**BFS search** starts from root node then traversal into next level of graph and continues, if item found it stops otherwise it continues. Here, in the pseudo code a separate function is created to find the traversal of a directed and weighted graph. In this method of search the searching is done level by level. The search starts at the root node and then it visits all the adjacent nodes on that level, once done it moves to the next level. If it is used for searching ends when the node or edge is found else if it is used for traversing the function will only end when it has visited each node at least once.

**DFS** is an algorithm for traversing or searching graph structure. The algorithm starts at the root node , selecting some arbitrary node as the root node in the case of a graph and explores as far as possible along each branch before backtracking. Here, in the pseudo code a function is created to find the traversal of a directed and weighted graph. In this search the search starts at any node and it continues in that particular branch until it reaches the end node in that branch. If the node is found then it terminates else it moves to the next arbitrary node. But in case of traversal the function will continue to execute until all the nodes are visited once.

1. **To find path from one vertex to another –**

A path in a graph is a sequence of vertices and edges that connects two distinct vertices without repeating any vertex or edge. For example, if you have a graph with vertices A, B, C, D, and E, and edges AB, BC, CD, and DE, then ABCDE is a path from A to E. To find the path from one vertex to another the concept of visiting matrix is used. It will help to find all the nodes and edges that are connecting both the vertex or nodes.

1. **Check if a graph contains a given value-**

To check whether a given graph contains a particular value we can use breadth – first search (BFS) or depth-first search (DFS). But here in this pseudo code we have used for loop to check each node and edge one by one.

1. **Add or remove edge-**

Adding an edge: Adding an edge is done by inserting both of the vertices connected by that edge in each others list. For example, if an edge between (u, v) has to be added, then u is stored in v’s vector list and v is stored in u’s vector list.

Deleting an edge: To delete edge between (u, v), u’s adjacency list is traversed until v is found and it is removed from it. The same operation is performed .

Here , in this pseudo code we have removed an edge from the graph by changing the value of an edge in the adjacency matrix to 0. And an edge is added at a point by adding the particular value.

1. **Find indegree and outdegree of each node/vertex –**

The first step in this program is to build an adjacency matrix (by linking all direct vertices) of the graph. This is done by a function called int out, it is a looping process which calculates the indegree and outdegree of nodes by calling their respective functions. The indegree function does the job of counting the number 1’s which are present in a column in the adjacency matrix by making use of the nodes number having a degree which can be calculated as a column index which can be calculated as a column index.

The outdegree function does the job of counting the numbers of 1’s present in the adjacency matrix by making use of the nodes number having a degree which can be calculated as a row index.

**PSEUDO CODE :**

1) START

2) READ vertices, edge and N <- 100

3) READ array Graph of 100 rows and 100 columns

4) READ i, j

5) PRINT enter the value of vertices and edge

6) STORE the value of vertices and edges from the user

7) i<-0, j<-0

8) for <- i from 0 to vertices-1 do

If i < vertices

for <- j from 0 to vertices-1 do

If j < vertices

array Graph element at i row and j column <- 0

Close If

i <- i+1

Close j for loop

Close If

j <- j+1

Close i for loop

8) READ l, l<-0

9) for <- l from 0 to edge-1 do

If l < edge

PRINT enter the nodes and value of edge

READ a, b, c

STORE array Graph elements at a row and b column <- c

close If

l <- l+1

Close l for loop

10) PRINT the directed and weighted graph:

11) READ k <- 0, m <- 0

12) for <- k from 0 to vertices-1 do

If k < vertices

for <- m from 0 to vertices-1 do

If m < vertices

PRINT array Graph element at k row and m column

Close If

m <- m+1

Close m for loop

Close If

k <- k+1

Close k for loop

13) Take the cursor to the next line

14) READ g <-0, h <-0

15) for <- h from 0 to vertices-1 do

If h < vertices

for <- g from 0 to vertices-1 do

If g < vertices

array Graph element at h row and g column <- 0

Close If

g <- g+1

Close g for loop

Close If

h <- h+1

Close h for loop

16) READ r <- 0

17) for <- r from 0 to edge-1 do

If r < edge

PRINT enter the nodes

READ a2, b2

STORE array Graph elements at a2 row and b2 column <- 1

close If

r <- r+1

Close r for loop

18) PRINT the directed and unweighted graph

19) READ k2 <-0, m2 <-0

20) for <- k2 from 0 to vertices-1 do

If k2 < vertices

for <- m2 from 0 to vertices-1 do

If m2 < vertices

PRINT array Graph element at k2 row and m2 column

Close if

m2 <- m2+1

Close m2 for loop

Close If

k2 <- k2+1

Close k2 for loop

21) Take the cursor to the next line

22) READ i2 <- 0, j2 <- 0

23) for <- i2 from 0 to vertices-1 do

If i2 < vertices

for <- j2 from 0 to vertices-1 do

If j2 < vertices

array Graph element at i2 row and j2 column <- 0

Close if

j2 <- j2+1

Close j2 for loop

Close If

i2 <- i2+1

Close i2 for loop

24) READ l2

25) for <- l2 from 0 to edge-1 do

If l2 < edge

PRINT enter the nodes and the value of edge

READ a3, b3, c3

STORE array Graph elements at a3 row and b3 column <- STORE array Graph elements at b3 row and a3 column <- c3

close If

l2 <- l2+1

26) PRINT the undirected and weighted graph

27) READ k3 <- 0, m3 <- 0

27) for <- k3 from 0 to vertices-1 do

If k3 < vertices

for <- m3 from 0 to vertices-1 do

If m3 < vertices

PRINT array Graph element at k3 row and m3 column

Close if

m3 <- m3+1

Close m3 for loop

Close If

k3 <- k3+1

Close k3 for loop

28) Take the cursor to the next line

29) READ i4 <- 0, j4 <- 0

30) for <- 14 from 0 to vertices-1 do

If i4 < vertices

for <- j4 from 0 to vertices-1 do

If j4 < vertices

array Graph element at i4 row and j4 column <- 0

Close If

j4 <- j4+1

Close j4 for loop

Close If

i4 <- i4+1

Close i4 for loop

31) READ l3 <- 0

32) for <- l3 from 0 to edge-1 do

If l3 < edge

PRINT enter the nodes

READ a4, b4

STORE array Graph elements at a4 row and b4 column <- STORE array Graph elements at b4 row and a4 column <- 1

close If

l2 <- l2+1

close for loop

33) PRINT the undirected and unweighted graph:

34) READ k5 <-0, m5 <-0

35) for <- k5 from 0 to vertices-1 do

If k5 < vertices

for <- m5 from 0 to vertices-1 do

If m5 < vertices

PRINT array Graph element at k5 row and m5 column

Close If

m5 <- m5+1

Close m5 for loop

Close If

k5 <- k5+1

Close k5 for loop

READ an array of 100 rows and 100 columns

READ vertices , edge ,w<-0,z<-0

for<-w from 0 to vertices-1 do

if w < vertices

for<-z from 0 to vertices-1 do

if z < vertices

array element at w row and z column <- 0

close if

z<-z+1

close z for loop

close if

w<-w+1

close w for loop

t1<-0

for<-t1 from 0 to egde-1 do

if t1<edge

READ s1,s2,s3 from the user for the nodes and edges

array element at s1 row and s2 column <- s3

close if

t1<-t1+1

close t1 for loop

o<-0,p<-0

for<-o from 0 to vertices-1 do

if o<vertices

for<-p from 0 to vertices-1 do

if p<vertices

if array element at o row and p column==0

continue

else

array element at o row and p column<-1

close if

p<-p+1

close p for loop

close if

o<-o+1

close o for loop

PRINT the adjacency matrix:

k6<-0,m6<-0

for<-k6 from 0 to vertices-1 do

if k6<vertices

for<-m6 from 0 to vertices-1 do

if m6<vertices

PRINT array element at k6 row and m6 column

close if

m6<-m6+1

close m6 for loop

close if

take the cursor to the next line

k6<-k6+1

close k6 for loop

READ search value from the user

1.READ max <- 100

2.FUNCTION out

READ array element at row and max column, READ n

READ t <- 0, s <-0

for <- t from 0 to n-1 do

if t < n

for <- s from 0 to n-1

if s < n

PRINT if there is an edge from i to j enter 1,else enter 0

take the cursor to the next line

STORE array adj element at i row and j column

close if

s <- s+1

close s for loop

close if

t <- t+1

close t for loop

3.FUNCTION outdeg

READ array element at row and max column, READ X5, READ n

READ f, count <- 0

for <- f from 0 to n-1

if f < n

if array adj element at x5 row and f column <- 1

count <- count + 1

close if

close if

f <- f+1

close f for loop

return count

4.FUNCTION indeg

READ array element at row and column max,READ x6, READ n

READ e, count <- 0

for <- e from 0 to n-1

if e < n

if array adj element at e row and x6 column <- 1

count <- count + 1

close if

close if

e <- e+1

close e for loop

return count

5.FUNCTION void main

READ array element at row max column max, READ node, READ n, READ i

PRINT the number of nodes in the graph maximum = max

Take the cursor to the next line

READ the value of n from the user

FUNCTION out

READ adj, READ n

for <- i0 from 0 to n-1

if i0 < n

PRINT The indegree of node i0 : FUNCTION indeg

PRINT The outdegree of node i0 : FUNCTION outdegclose if

close if

i0 <- i0+1

close i0 for loop

Declare one array of 2D as ‘a’ as an adjacency matrix and one array of 1D as ‘visited’ array

Call the function DFS

For i in range where i=0 and i<n and i=i+1 to check all the vertices in the graph

If a of v and i is not equal to 0 and visited value of i is equal to 0 to check whether it is adjacent to v and not visited

Then visited value of I is equal to 1 to mark the vertex visited

Print i

Then call the function again up to n iteration

Call the main function

Read the variables v, i, j

Take the no. of vertices as an input

Read n

For where I = 0 and I < n and I = I +1

Then visited value of i is equal to 0

Take the graph data in matrix form as an input

For j in range where i=0 and i<n and i=i+1

For j in range where j=0 and j<n and j=j+1

Read the value of a of i and j

Take the starting vertex as an input

Read the value of v

Print the DFS traversals

Then visited value of v = 1 when the starting vertex is visited

Print v

Call the function DFS of v again

Declare one array in 2D as ‘a’ as an adjacency matrix and two arrays of 1D as ‘q’ as queue array and ‘v’ as a visited array.

Take value of f=-1 and r=-1

Create the function BFS

Read i

For i in range where i=0 and i<n and i=i+1 to check all the vertices in the graph

If a of v and i is not equal to 0 and visited value of i is equal to 0 to check whether it is adjacent to v and not visited

Then r=r+1

The implementation of value of q as the value of r to insert them into queue

Visited value of I is equal to v for marking the visited vertex

Print i

Then to remove the vertex at front of the queue f=f+1

If f <=r as long as there are elements in the queue

Call the BFS function again

Take the no. of vertices as an input

Read n

For where I = 0 and I < n and I = I +1

Then visited value of i is equal to 0

Take the graph data in matrix form as an input

For j in range where i=0 and i<n and i=i+1

For j in range where j=0 and j<n and j=j+1

Read the value of a of i and j

Take the starting vertex as an input

Then f=r=0

The implementation of value of q as the value of v to insert them into queue

Print the BFS traversals

Then visited value of v = 1 when the starting vertex is visited

Print v

Call the function BFS of v again

If r is not equal to n – 1 then BFS is not possible

READ search1 from the user

o3<-0,p3<-0

for<-o3 from 0 to vertices-1 do

if o3<vertices

for<-p3 from 0 to vertices-1 do

if p3<vertices

if array element at o3 row and p3 column<-search1

array element at o3 row and p3 column<-0

else

continue

close if

p3<-p3+1

close p3 for loop

close if

o3<-o3+1

close o3 for loop

PRINT the matrix after removing the edge:

k7<-0,m7<-0

for<-k7 from 0 to vertices-1 do

if k7<vertices

for<-m7 from 0 to vertices-1 do

if m7<vertices

PRINT array element at k7 row and m7 column

close if

m7<-m7+1

close m7 for loop

close if

k7<-k7+1

close k7 for loop

READ search2,r2,c2 from the user for adding element ,row and column

x<-0,y<-0

array at r2 row and c2 column<-search2

PRINT the matrix after adding the edge:

for<-x from 0 to vertices-1 do

if x<vertices

for<-y from 0 to vertices-1 do

if y<vertices

PRINT array element at x row and y column

close if

y<-y+1

close y for loop

close if

x<-x+1

close x for loop

STOP

**MATHEMATICAL EXPLANATION :**

The adjacency matrix is a matrix which stores the graph structure. The row and column number of the matrix represent the nodes and the elements of the matrix represent the edge values. The elements of the adjacency matrix are whole numbers. The values that we have search are the elements of the matrix . The edges which we have to remove or add are whole numbers . So, if we remove an edge we are replacing an element of the matrix by 0 and if we are adding an edge value we are replacing a zero by a given whole number. When we are searching a value in the graph we are searching for a value in the matrix and the value is an element of the matrix.

MATHEMATICALLY REPRESENTATION

OF DFS & BFS

ADJACENCY MATRIX (if no. of vertices=5)

0 1 2 3 4

0 0 1 0 0 1

1 1 0 1 1 1

2 0 1 0 1 0

3 0 1 1 0 1

4 1 1 0 1 0

VISITED ARRAY

0 1 2 3 4

0 0 0 0 0 -> Since visited [i]=0

GRAPH REPRESENTATION

3

4

**GITHUB LINK OF CODE: -**

[ucse22016/GROUP-1-PROJECT-CODE (github.com)](https://github.com/ucse22016/GROUP-1-PROJECT-CODE)

**DEMO INPUT AND OUTPUT:**

enter the value of vertices and edge2,4

enter the nodes and value of edge0,0,6

enter the nodes and value of edge0,1,8

enter the nodes and value of edge1,0,9

enter the nodes and value of edge1,1,4

the directed and weighted graph:6 8

9 4

enter the nodes0,0

enter the nodes0,1

enter the nodes1,0

enter the nodes1,1

the directed and unweighted graph:1 1

1 1

enter the nodes and value of the edge0,0,6

enter the nodes and value of the edge0,1,8

enter the nodes and value of the edge1,0,9

enter the nodes and value of the edge1,1,4

the undirected and weighted graph:69

94

enter the nodes 0,0

enter the nodes0,1

enter the nodes1,0

enter the nodes1,1

the undirected and unweighted graph:1 1

1 1

enter the number of vertices and edge 4,10

enter the nodes and value of edge1,2,3

enter the nodes and value of edge3,1,6

enter the nodes and value of edge2,2,8

enter the nodes and value of edge1,3,2

enter the nodes and value of edge0,3,7

enter the nodes and value of edge0,1,9

enter the nodes and value of edge0,2,3

enter the nodes and value of edge2,1,18

enter the nodes and value of edge3,2,1

enter the nodes and value of edge3,3,0

The adjacency matrix:

0 1 1 1

0 0 1 1

0 1 1 0

0 1 1 0

enter the number of vertices and edge 2,4

enter the nodes and value of edge0,0,4

enter the nodes and value of edge0,1,7

enter the nodes and value of edge1,0,19

enter the nodes and value of edge1,1,24

enter the value to search for 19

the search value 19 was found at 1,0

The number of nodes in the graph maximum = 100

5

if there is an edge from 0 to 0 enter 1, else enter 0

1

if there is an edge from 0 to 1 enter 1, else enter 0

0

if there is an edge from 0 to 2 enter 1, else enter 0

1

if there is an edge from 0 to 3 enter 1, else enter 0

0

if there is an edge from 0 to 4 enter 1, else enter 0

1

if there is an edge from 1 to 0 enter 1, else enter 0

1

if there is an edge from 1 to 1 enter 1, else enter 0

0

if there is an edge from 1 to 2 enter 1, else enter 0

0

if there is an edge from 1 to 3 enter 1, else enter 0

1

if there is an edge from 1 to 4 enter 1, else enter 0

0

if there is an edge from 2 to 0 enter 1, else enter 0

1

if there is an edge from 2 to 1 enter 1, else enter 0

0

if there is an edge from 2 to 2 enter 1, else enter 0

1

if there is an edge from 2 to 3 enter 1, else enter 0

1

if there is an edge from 2 to 4 enter 1, else enter 0

0

if there is an edge from 3 to 0 enter 1, else enter 0

0

if there is an edge from3 to 1 enter 1, else enter 0

0

if there is an edge from 3 to 2 enter 1, else enter 0

1

if there is an edge from 3 to 3 enter 1, else enter 0

1

if there is an edge from 3 to 4 enter 1, else enter 0

1

if there is an edge from 4 to 0 enter 1, else enter 0

0

if there is an edge from 4 to 1 enter 1, else enter 0

1

if there is an edge from 4 to 2 enter 1, else enter 0

0

if there is an edge from 4 to 3 enter 1, else enter 0

1

if there is an edge from 4 to 4 enter 1, else enter 0

0

The indegree of node0: 3

The outdegree of node 0 : 3

The indegree of node1: 1

The outdegree of node 1 : 2

The indegree of node2: 3

The outdegree of node 2 : 3

The indegree of node3: 4

The outdegree of node 3 : 3

The indegree of node4: 2

The outdegree of node 4 : 2

Enter the number of vertices:5

Enter graph data in matrix form:

0 1 0 0 1

1 0 1 1 1

0 1 0 1 0

0 1 1 0 1

1 1 0 1 0

Enter the starting vertex:0

BFS traversal is:

0 1 4 2 3

Enter the number of vertices;5

Enter graph data in matrix form:

0 1 0 0 1

1 0 1 1 1

0 1 0 1 0

0 1 1 0 1

1 1 0 1 0

1

1 1 0 1 0

enter the number of vertices and edge 2,3

enter the nodes and value of edge0,0,4

enter the nodes and value of edge0,1,7

enter the nodes and value of edge1,1,8

enter the edge to be removed 7

the graph after removal:

4 0

0 8

enter the value of edge to add and the nodes 3,1,0

the graph after addition of edge:

4 7

3 8



