**Batch: B3 Roll. No.: 121**

**Experiment: 8**

**Grade: AA / AB / BB / BC / CC / CD /DD**

|  |
| --- |
| **Title:**  Implementation of Graph traversal menu driven program (DFS and BFS). |

**Objective:** To understand graph as data structure and methods of traversing Graph

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO2** | Apply linear and non-linear data structure in application development |

**Websites/books referred:**

1. Data Structures Using C - Aaron M. Tenenbaum, Yedidyah Langsam, and Moshe J. Augenstein.
2. <https://www.javatpoint.com/ds-graph>
3. <https://www.naukri.com/learning/articles/graphs-in-data-structure-types-representation-operations/>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Abstract**: - (Definition of Graph, types of graphs, and difference and similarity between graph & tree)

**Graph:**

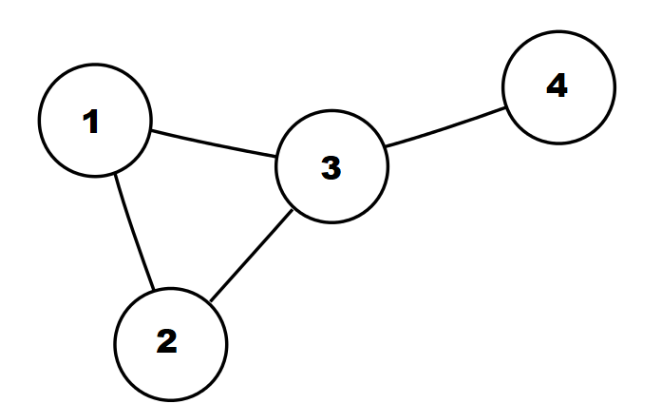
A graph can be defined as a group of vertices and edges that are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (nodes) maintain any complex relationship among them instead of having only parent-child relationship.

A graph G can be defined as an ordered set G(V, E) where V(G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices.

**Types of Graph:**

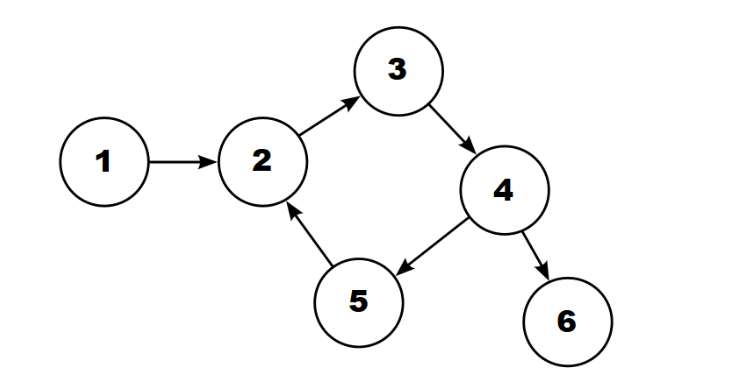
1. Undirected Graph

A graph in which all edges are bi-directional. The edges do not point in a specific direction.



1. Directed Graph

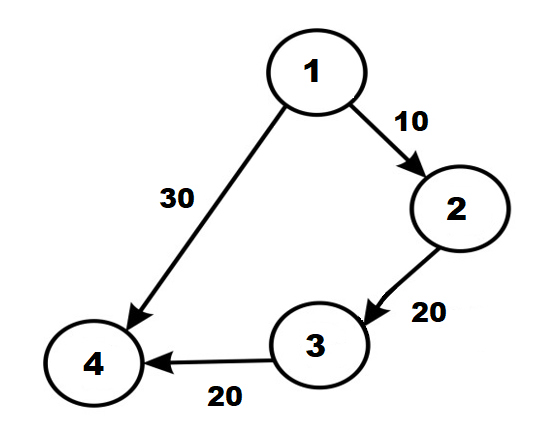
A graph in which all the edges are uni-directional. The edges point in a single direction.



1. Weighted Graph

A graph that has a value associated with every edge. The values corresponding to the edges are called weights. A value in a weighted graph can represent quantities such as cost, distance and/or time, depending on the graph. Weighted graphs are typically used in modelling computer networks.

An edge in a weighted graph is represented as (u, v, w), where u is the source vertex, v is the destination vertex and w represents the weight associated to go from u to v.



1. Unweighted Graph

A graph in which there is no value or weight associated with the edge. All the graphs are unweighted by default unless there is a value associated. An edge of an unweighted graph is represented as (u, w), where u is the source vertex and v is the destination vertex.

**Difference between graph and tree:**

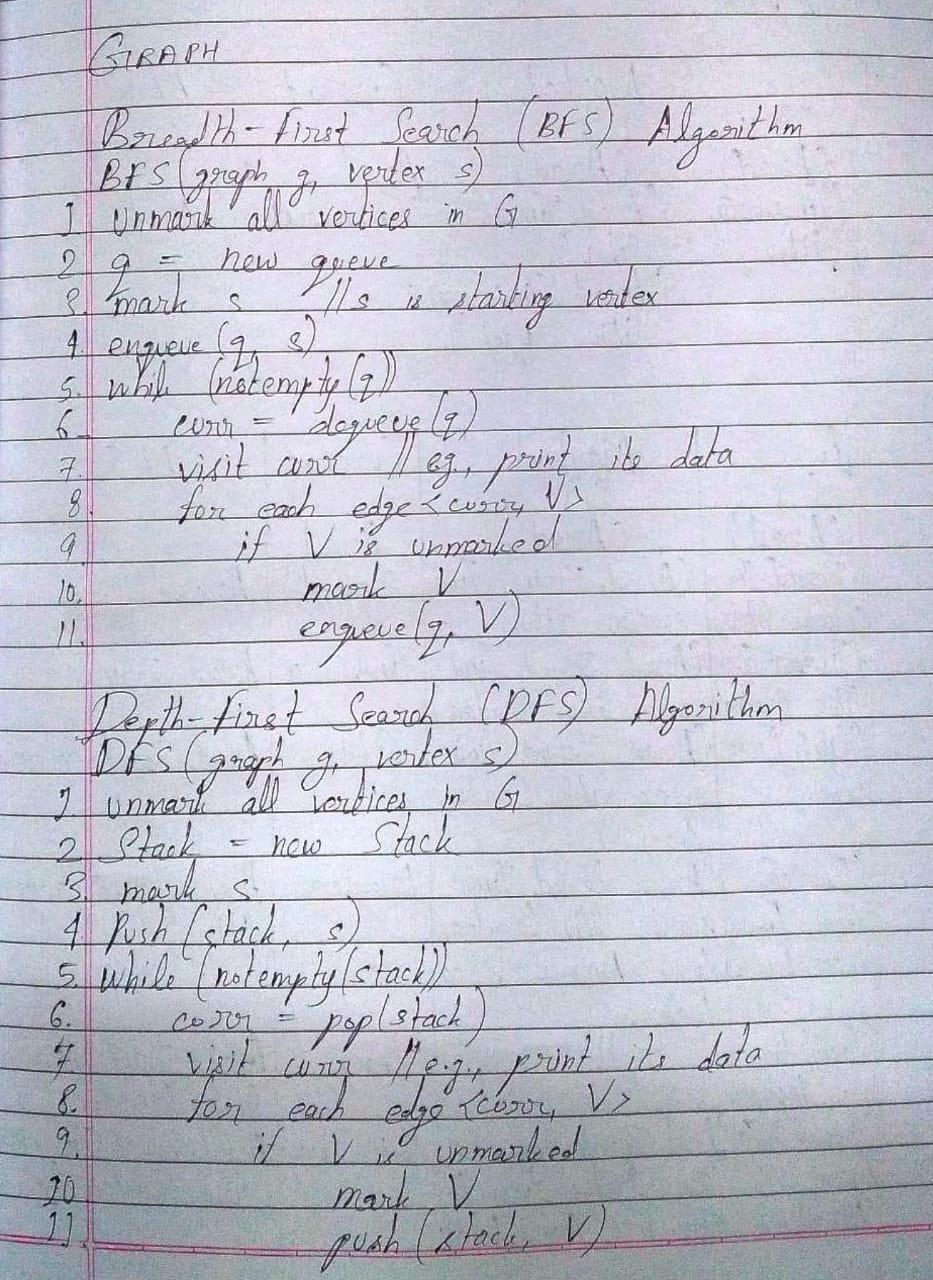
|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Graph** | **Tree** |
| **1.** | Each node in a Graph can have any number of edges. | The nth node in a Tree can have a maximum of (n-1) number of edges. |
| **2.** | The edges can be directed or undirected. | The edges are always directed away from root node and towards leaf node. |
| **3.** | There is no unique node called “root” in a Graph. | Every Tree has a parent node called Root. |
| **4.** | For Graph traversal, Breadth-First Search (BFS) and Depth-First Search (DFS) is used. | For Tree traversal, inorder, preorder and postorder is used. |

**Similarity between graph and tree:**

.

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Graph** | **Tree** |
| **1.** | Graph is a non-linear data structure. | Tree is also a non-linear data structure. |
| **2.** | It is a collection of nodes and edges. | It is also a collection of nodes and edges. |

**Algorithm for DFS/BFS:**



**Code and output screenshots:**

#include<stdio.h>

#include<stdlib.h>

int visit[20] = {0}, v[20] = {0};

typedef struct Node

{

int data;

struct Node \*prev;

struct Node \*next;

}Node;

typedef struct Queue

{

struct Node \*rr;

struct Node \*fr;

}Q;

int dequeue(Q \*q);

void enqueue(int ch, Q \*q);

void display(Q \*q);

void dfs(int t, int a[20][20], int n);

void bfs(int t, int a[20][20], int n, Q \*q);

void main()

{

int num\_vert, i, j, num\_edge, t, p, q, start;

int a[20][20];

printf("\nEnter the number of vertices: ");

scanf("%d", &num\_vert);

for(i = 0; i < num\_vert; i++)

{

visit[i] = 0;

for(j = 0; j < num\_vert; j++)

{

a[i][j] = 0;

}

}

printf("\nEnter the number of edges: ");

scanf("%d", &num\_edge);

printf("\nEnter '1' if the graph is undirected and '0' if the graph is directed: ");

scanf("%d", &t);

for(i = 0; i < num\_edge; i++)

{

printf("\nEnter edge vertex (p,q): ");

scanf("%d%d", &p, &q);

a[p-1][q-1] = 1;

if(t == 1)

a[q-1][p-1] = 1;

}

for(i = 0; i < num\_vert; i++)

{

for(j = 0; j < num\_vert; j++)

{

printf("%d", a[i][j]);

}

printf("\n");

}

printf("\nEnter the element from which you want to start the BFS and DFS: ");

scanf("%d", &start);

Q q1;

q1.fr = q1.rr = NULL;

printf("\nBFS:\n");

bfs(start, a, num\_vert, &q1);

printf("\nDFS:\n");

dfs(start, a, num\_vert);

}

void enqueue(int ch, Q \*q)

{

Node \*newnode;

newnode = (Node\*)malloc(sizeof(Node));

newnode->data = ch;

newnode->prev = NULL;

newnode->next = NULL;

if(q->fr == NULL)

{

q->fr = newnode;

q->rr = newnode;

}

else

{

newnode->next = q->fr;

q->fr->prev = newnode;

q->fr = newnode;

}

}

int dequeue(Q \*q)

{

Node \*temp;

if(q->rr != NULL)

{

temp = q->rr;

int d = temp->data;

q->rr = temp->prev;

if(q->rr != NULL)

q->rr->next = NULL;

else

q->fr = NULL;

return d;

}

return 0;

}

void display(Q \*q)

{

Node \*temp;

temp = q->fr;

while(temp != NULL)

{

printf(" %c", temp->data);

temp = temp->next;

}

}

void bfs(int t, int a[20][20], int num\_vert, Q \*q)

{

int i, j;

printf("%d->", t);

int temp;

enqueue(t, q);

v[t-1] = 1;

while(q->fr != NULL){

temp = dequeue(q);

for(i = 0; i < num\_vert; i++){

if(a[temp-1][i] == 1 && v[i] == 0){

enqueue(i+1, q);

printf("%d->", i+1);

v[i] = 1;

}

}

}

}

void dfs(int t, int a[20][20], int num\_vert)

{

int i, j;

printf("%d->", t);

visit[t-1] = 1;

for(i = 0; i < num\_vert; i++){

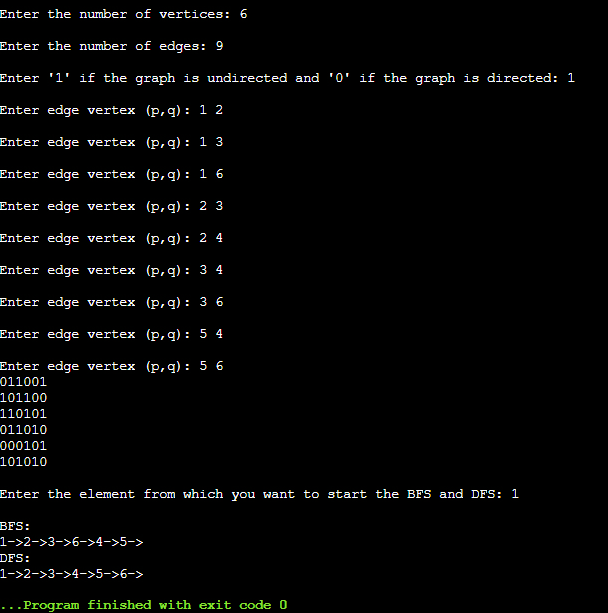
if(a[t-1][i] == 1 && visit[i] == 0){

dfs(i+1, a, num\_vert);

}

}

}



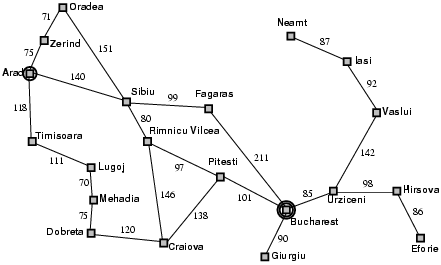
**Post lab questions-**

1. **Differentiate between BFS and DFS.**

**Ans.**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **BFS** | **DFS** |
| **1.** | BFS uses Queue data structure to find the shortest path. | DFS uses Stack data structure. |
| **2.** | BFS is a traversal approach in which all nodes on the same level are first visited before moving to the next level. | DFS is a traversal approach in which the traversal begins at the root node and proceeds through the nodes as far as possible until that node is reached which has no unvisited nearby nodes. |
| **3.** | BFS builds the tree level-by-level. | DFS builds the tree subtree-by-subtree. |
| **4.** | It works on the concept of FIFO (First-In-First-Out) | It works on the concept of LIFO (Last-In-First-Out). |

1. **Give sequence of the nodes visited as per BFS and DFS strategy for following example. Source- Arad, Destination- Bucharest (Traversal would stop after destination is reached)**

****

Ans. BFS: Arad 🡪 Zerind 🡪 Timisoara 🡪 Sibiu 🡪 Oradea 🡪 Lugoj 🡪 Rimnicu Vilcea 🡪 Fagaras 🡪 Mehadia 🡪 Pitesti 🡪 Craiova 🡪 Bucharest

**Conclusion: -**

Thus, in this experiment, the concept of Graphs and their traversal techniques has been learnt. Graphs are used in a wide variety of applications like navigation (to find the path between two points with the least distance or time), computer networking, artificial intelligence, etc. Graphs proves to be a superset of trees – all trees are graphs but all graphs are not trees.