**ASSIGNMENT 14**

**AIM :-**

Store a graph using adjacency matrix or adjacency list representation and perform Breadth First Traversal. (Recursive / non-recursive)

**OBJECTIVE :-**

This algorithm represents a graph using adjacency matrix. Where the matrix is square and of size VXV. The V is the number of vertices in the graph. For each adjacent vertices , 1 is stored in the respective position ,otherwise 0. Breadth First traversal for a graph is similar to BFT for a tree. The only catch here is , unlike trees graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use boolean visited array.

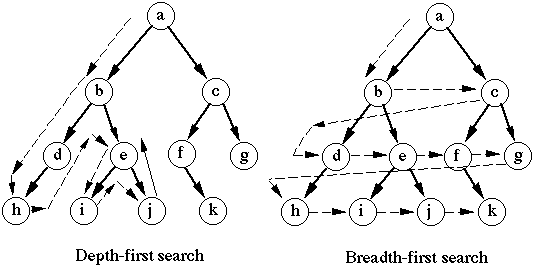
**THEORY:**

**Depth**-**first search**

(DFS) is an algorithm for traversing or searching tree or graph data structures. 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.

**Breadth first search:**

Traversal means visiting all the nodes of a graph. Breadth first traversal or Breadth first Search is a recursive algorithm for searching all the vertices of a graph or tree data structure. In this article, you will learn with the help of examples the BFS algorithm, BFS pseudocode and the code of the breadth first search algorithm with implementation in C++, C, Java and Python programs.



**ALGORITHM:**

Algorithm for **Depth First Traversal** or **DFS()** function :-

1. Start by passing three parameters wiz. Starting vertex, boolean vector visited, and number of edges.
2. First when the boolean vector is initialized in the **main()** , it stores all false values for the respective edges.
3. Print the starting vertex and change the value of visited vector of the respective index to true.
4. Check all the vertices adjacent to the starting vertex and if they are visited or not. If not ,recall the DFS() function by passing the respective vertex.
5. The for loop gets executes till all the edges have been visited according to the Depth First Search Algorithm.
6. This function here is menu driven, as per user ,if they want to check it for different starting vertices.
7. Exit the function.

Algorithm for Breadth First Traversal or **BFS()** function :

1. Start the function by passing three parameters wiz. Starting vertex , visited vector , number of vertices.
2. Initialize an integer queue to store the vertices while traversing.
3. Push the starting element into the queue and mark its respective position in visited vector to true.
4. Print the element from the starting position of the queue.
5. Check for all the adjacent vertices of the starting position whether they are visited or not.
6. If yes, recursively call BFS function by passing the respective parameters.
7. Exit the function.

**CODE:**

#include<iostream>

#include <bits/stdc++.h>

#include<queue>

using namespace std;

#define max 50

int adjmtx[max][max];

bool isEdge(int ,int ); //function to check whether the edge is a valid integer or not

void DFS(int ,vector<bool>& visited,int ); //recursive function for depth first traversal

void BFS(int ,vector<bool>& visited,int ); //recursive function for breadth first traversal

//function to add edges in the adjacency matrix

void addEdge(){

int m,n;

cin>>m>>n;

if(isEdge(m,n)){

if(m==n){

adjmtx[m][n] += 2;

}

else{

adjmtx[m-1][n-1]++;

adjmtx[n-1][m-1]++;

}

}

else cout<<"Enter a valid edge!";

}

//function to display the adjacency matrix

void displaymtx(int v){

for(int i=0;i<v;i++){

cout<<" |";

for(int j=0;j<v;j++){

cout<<" "<<adjmtx[i][j];

}

cout<<" |\n";

}

}

int main(){

int ver,edge,s,choice;

char ch;

cout<<"\n ------------------------- \n";

cout<<" :: GRAPHS :: ";

cout<<"\n ------------------------- \n";

cout<<"\n\n Enter the number of vertices you want in the graph : ";

cin>>ver;

adjmtx[ver][ver]; //changing the size of matrix for every graph

//initialising all elements of the matrix to 0

for(int i=0;i<ver;i++){

for(int j=0;j<ver;j++){

adjmtx[i][j]=0;

}

}

cout<<"\n Enter the number of edges in the graph : ";

cin>>edge;

//taking the edge input (non-directional)

for(int i=1;i<=edge;i++){

cout<<"\n Enter the "<<i<<" edge (v1 v2) : ";

addEdge();

}

cout<<"\n ----------------------------------------------------- \n";

cout<<"\n The adjacency matrix of the given graph is : \n\n";

displaymtx(ver);

cout<<"\n ----------------------------------------------------- \n";

do{

vector<bool> visited(ver, false); //initialization of vector to store visited vertices

cout<<" Which search algorithm you want to use? \n 1.DFS \n 2.BFS \n Choice : ";

cin>>choice;

switch(choice){

case 1 : cout<<"\n ------------------------- \n";

cout<<"\n Depth First Traversal \n\n";

cout<<" Enter the vertex you want to start with : ";

cin>>s;

cout<<"\n The DFS output is : \n";

DFS(s,visited,ver);

cout<<"\n ------------------------- \n";

break;

case 2 : cout<<"\n ------------------------- \n";

cout<<"\n Breadth First Traversal \n\n";

cout<<" Enter the vertex you want to start with : ";

cin>>s;

cout<<"\n The BFS output is : \n";

BFS(s,visited,ver);

cout<<"\n ------------------------- \n";

break;

default : cout<<"\n Enter a valid choice.";

break;

}

cout<<"\n Do you want to continue DFS ? Y-Yes/N-No : ";

cin>>ch;

}while(ch=='Y');

return 0;

}

bool isEdge(int m,int n){

if(m>0 && n>0) return true;

return false;

}

void BFS(int startver,vector<bool>& visited,int ver){

queue <int> Q;

//pushing the starting element into queue

Q.push(startver);

visited[startver-1]=true;

while(!Q.empty()){

int temp = Q.front();

cout<<" "<<temp; //Printing all the elements adjacent to starting vertex

Q.pop();

for(int i=0;i<ver;i++){

// If some node is adjacent to the current node

// and it has not already been visited

if(!visited[i] && adjmtx[startver-1][i]==1){

BFS(i+1,visited,ver);

}

}

}

}

void DFS(int start,vector<bool>& visited,int ver){

cout << " " << start;

visited[start-1]=true;

for (int i = 1; i <= ver; i++) {

// If some node is adjacent to the current node

// and it has not already been visited

if (adjmtx[start-1][i-1] == 1 && (!visited[i-1])) {

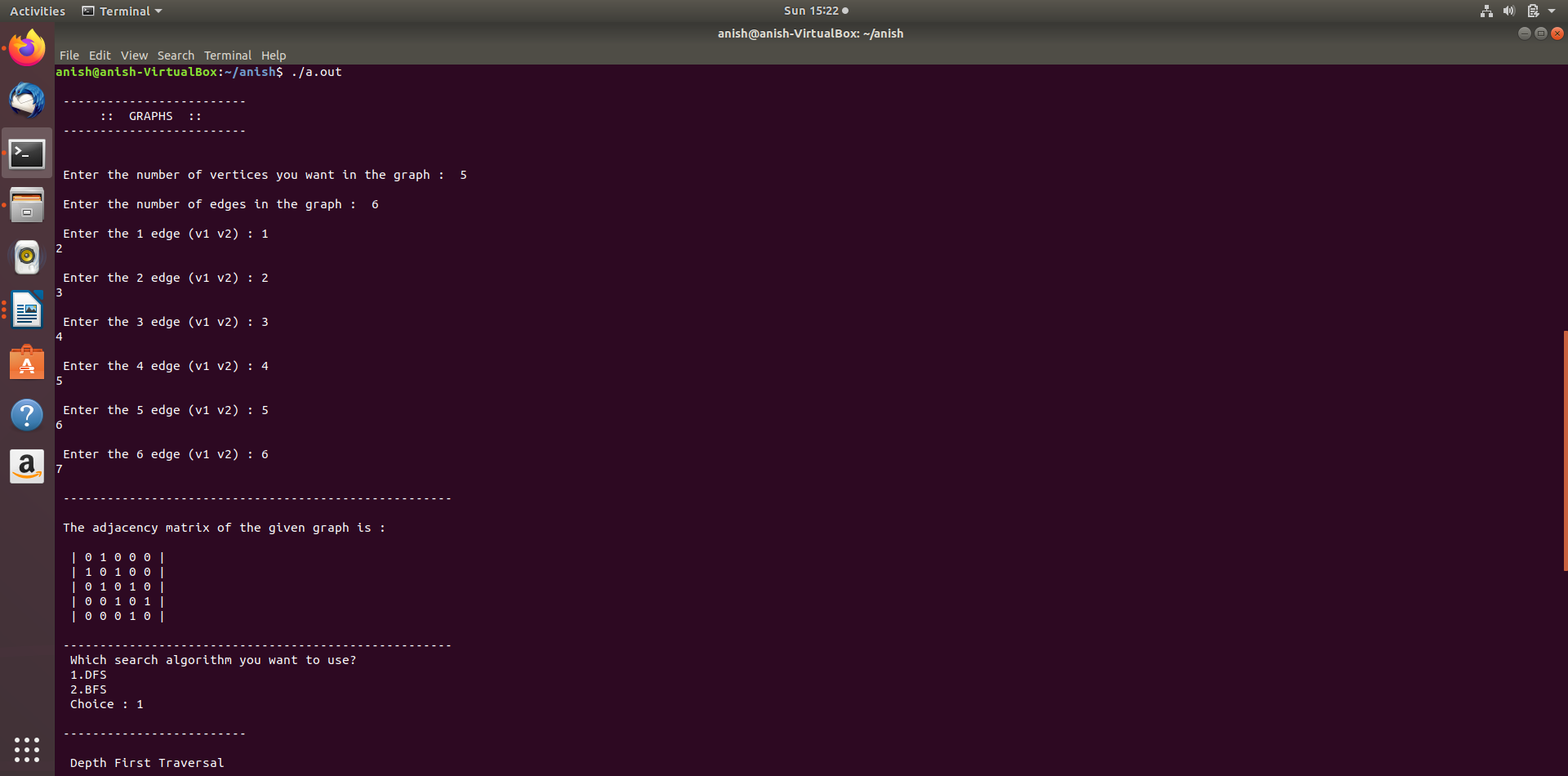
DFS(i, visited, ver);

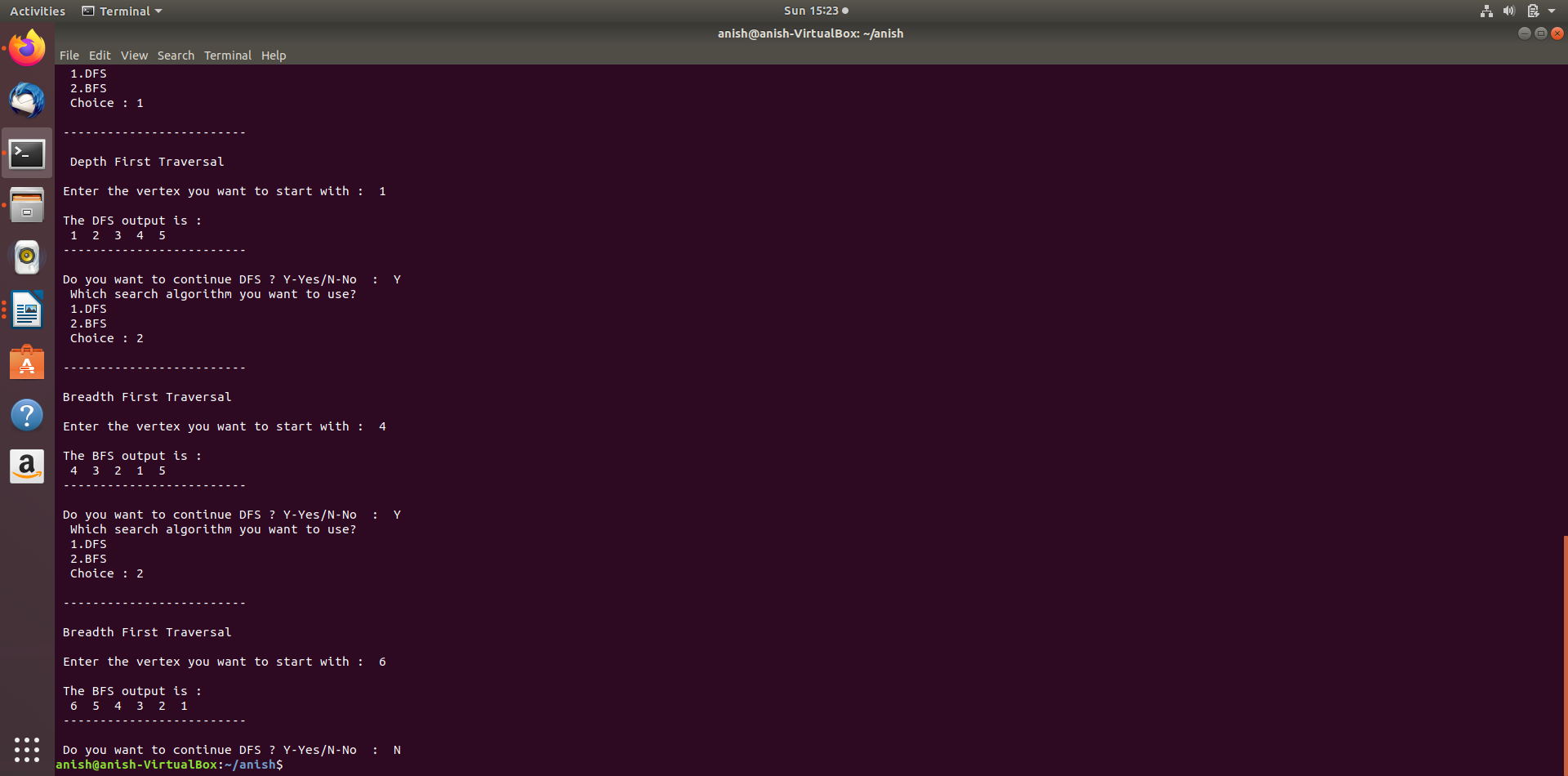
}

}

}

**OUTPUT:**

****



**CONCLUSION:**

The total time complexity of the program is O(n2) where n=total number of vertices.

The actual time complexity of the search algorithm is O(V+E) where V=number of vertices and E=number of edges.