**Practical 12**

**Aim :** Implementation of Graph Traversal (Depth First Search).

**Theory :**

**DFS (Depth First Search)** algorithm is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.

Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

**Algorithm**

**Step 1:** SET STATUS = 1 (ready state) for each node in G

**Step 2:** Push the starting node A on the stack and set its STATUS = 2 (waiting state)

**Step 3:** Repeat Steps 4 and 5 until STACK is empty

**Step 4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)

**Step 5:** Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

[END OF LOOP]

**Step 6:** EXIT

**Applications of DFS algorithm**

The applications of using the DFS algorithm are given as follows -

o DFS algorithm can be used to implement the topological sorting.

o It can be used to find the paths between two vertices.

o It can also be used to detect cycles in the graph.

o DFS algorithm is also used for one solution puzzles.

o DFS is used to determine if a graph is bipartite or not.

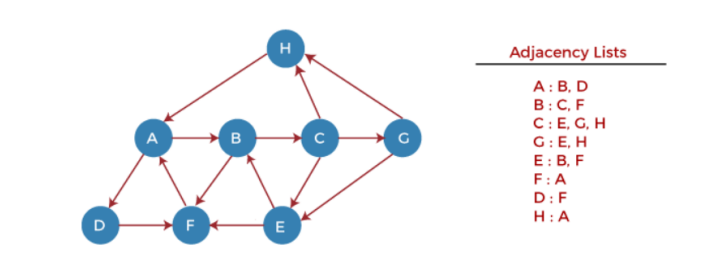
**Complexity of Depth-first search algorithm**

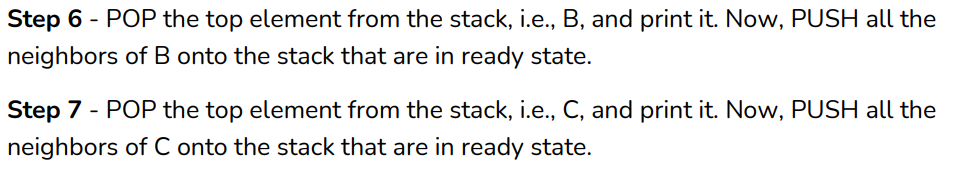
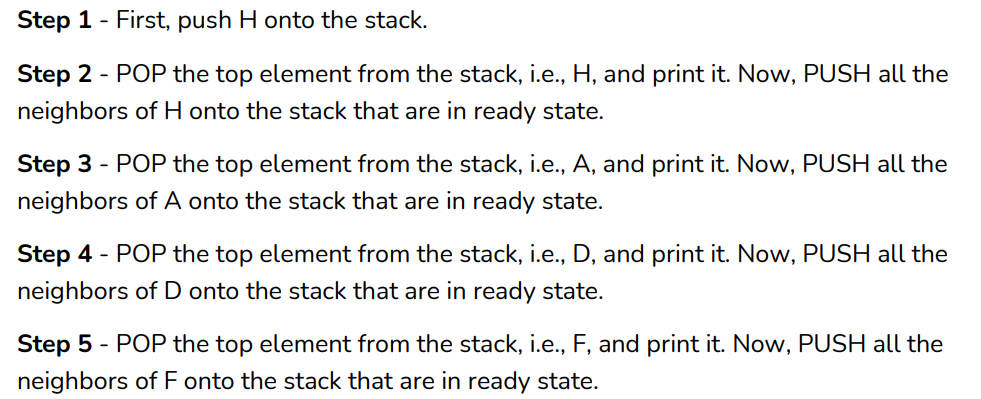
The time complexity of the DFS algorithm is O(V+E), where V is the number of vertices and E is the number of edges in the graph.

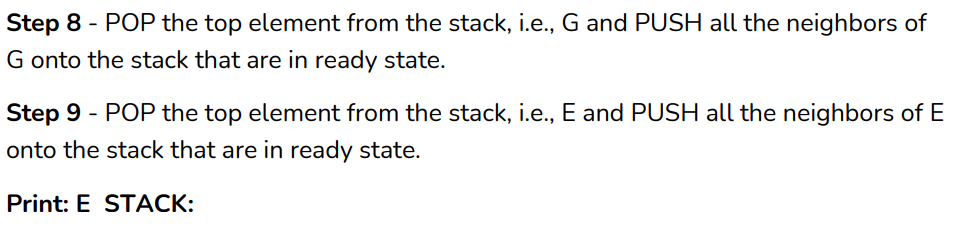
The space complexity of the DFS algorithm is O(V).

**Example of DFS algorithm**

Now, let's understand the working of the DFS algorithm by using an example. In the example given below, there is a directed graph having 7 vertices.







**Code :**

#include<iostream>

#include<vector>

#include<stack>

using namespace std;

void addEdge(int u,int v,vector<int>\* V){

V[u].push\_back(v);

}

void DFS(int s,vector<int>\* adj,int n){

int visited[n+1]={0};

stack<int> stack;

stack.push(s);

vector<int>::iterator i;

while (!stack.empty()){

s = stack.top();

stack.pop();

if (!visited[s])

{

cout << s << " ";

visited[s] = true;

}

for (i = adj[s].begin(); i != adj[s].end(); ++i)

if (!visited[\*i])

stack.push(\*i);

}

}

int main()

{

int n,e,u,v,start;

cout<<"Enter no of vertices : ";

cin>>n;

cout<<"\nEnter no of Edges : ";

cin>>e;

int copy=n;

vector<int> V[n+1];

for(int i=0;i<e;i++)

{ cout<<"\nEnter from : ";

cin>>u;

cout<<"\nEnter To : ";

cin>>v;

addEdge(u,v,V);

}

cout<<"\nGraph Representation using Adjacency List is : "<<endl;

vector<int>::iterator it;

for(int i=1;i<=n;i++)

{

cout<<i<<"->";

for(it=V[i].begin();it!=V[i].end();it++)

{

cout<<\*it<<" ";

}

cout<<endl;

}

cout<<"Enter start vertex : ";

cin>>start;

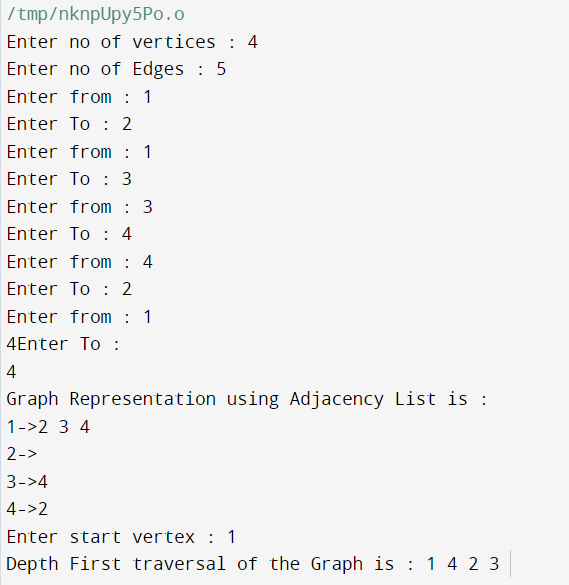
cout<<"\nDepth First traversal of the Graph is : ";

DFS(start,V,n);

return 0;

}

**Output :**



**Conclusion : learned depth first search Algorithm.**