Experiment No.-8

Student Name: SANSKAR AGRAWAL UID: 20BCS5914

Branch: C.S.E. Section/Group: 20BCS-806B

Semester: 5th Subject Code: 20CSP-312

Subject Name: Design and Analysis of Algorithm Lab

Aim:

1. To create a code to analyze depth-first search (DFS) on an undirected graph.
2. To create a code to find the topological sort of a directed acyclic graph, as the implementation of an application of DFS.

Algorithm:

1. DFS on an undirected graph:

```
DFS(G, u)
u.visited = true for each v ∈
G.Adj[u]   if v.visited == false
DFS(G,v)   init() {
For each u ∈ G
u.visited = false For
each u ∈ G
DFS(G, u) }
```

2. DFS to find the topological sort of a directed acyclic graph:

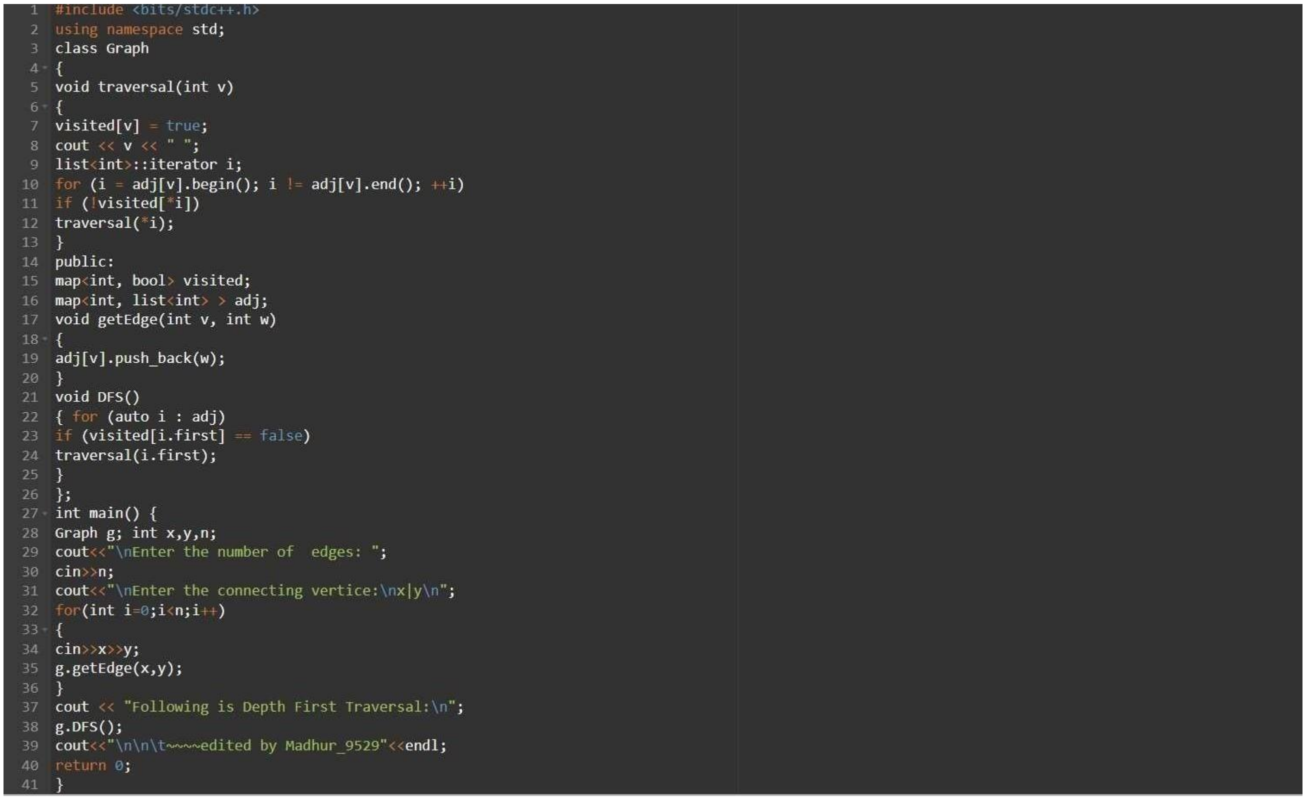
```
Begin function topologicalSort():
a) Mark the current node as visited.
b) Recur for all the vertices adjacent to this vertex.
c) Push current vertex to stack which stores result. End
Begin function topoSort() which uses recursive topological sort() function: a)
Mark all the vertices which are not visited. b)  Call the function topologicalSort().
```

c) Print the content. End

Code:

```
1. DFS on an undirected graph: #include <bits/stdc++.h>
using namespace std;
class Graph {
void traversal(int v)
{ visited[v] = true; cout << v << " "; list<int>::iterator i; for
(i = adj[v].begin(); i
!= adj[v].end(); ++i) if
(!visited[*i])
traversal(*i); } public:
map<int, bool> visited;
map<int, list<int>  > adj;
void getEdge(int v,
int w)
{
adj[v].push_back(w);
}
void DFS()
{ for (auto i : adj) if (visited[i.first]
== false) traversal(i.first);
} }; int main() { Graph g;  int x,y,n;  cout<<"\nEnter the number of
edges: "; cin>>n; cout<<"\nEnter the
connecting vertice:\nx|y\n"; for(int i=0;i<n;i++)
{ cin>>x>>y;
g.getEdge(x,y); } cout << "Following is Depth First Traversal:\n";
        g.DFS();       cout<<"\n\n\t~~~~edited    by
Madhur_9529"<<endl; return
0;
```

}

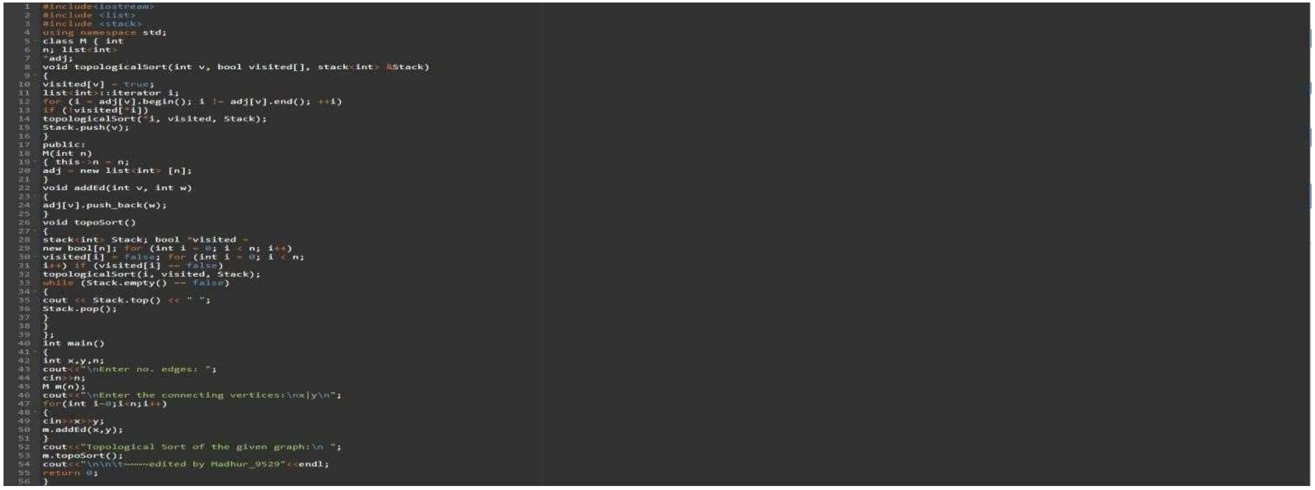


```
#include <bits/stdc++.h>
using namespace std;
class Graph
{
void traversal(int v)
{
visited[v] = true;
cout << v << " ";
list<int>::iterator i;
for (i = adj[v].begin(); i != adj[v].end(); ++i)
if (!visited[*i])
traversal(*i);
}
public:
map<int, bool> visited;
map<int, list<int> > adj;
void getEdge(int v, int w)
{
adj[v].push_back(w);
}
void DFS()
{ for (auto i : adj)
if (visited[i.first] == false)
traversal(i.first);
}
};
int main() {
Graph g; int x,y,n;
cout<<"\nEnter the number of  edges: ";
cin>>n;
cout<<"\nEnter the connecting vertice:\nx|y\n";
for(int i=0;i<n;i++)
{
cin>>x>>y;
g.getEdge(x,y);
}
cout << "Following is Depth First Traversal:\n";
g.DFS();
cout<<"\n\n\t~~~~edited by Madhur_9529"<<endl;
return 0;
}
```

2. DFS to find the topological sort of a directed acyclic graph:

```
#include<iostream>
#include <list> #include
<stack>                    using
namespace std; class M { int n; list<int> *adj; void
topologicalSort(int v, bool visited[], stack<int> &Stack)
{
visited[v] = true; list<int>::iterator i;
for (i = adj[v].begin(); i != adj[v].end(); ++i) if
(!visited[*i]) topologicalSort(*i, visited, Stack);
Stack.push(v);
}
public: M(int n) { this-
```

```
>n = n; adj = new
list<int> [n];
} void addEd(int v, int w)
{  adj[v].push_back(w);
}
void topoSort()
{ stack<int> Stack; bool *visited = new
bool[n]; for (int i = 0; i < n; i++)
visited[i] = false; for (int i = 0; i < n;
i++) if (visited[i] == false)
topologicalSort(i, visited, Stack);  while
(Stack.empty() == false)
{
cout << Stack.top() << " ";
Stack.pop();
}
} }; int main() { int x,y,n;
cout<<"\nEnter       no.
edges: ";  cin>>n; M m(n); cout<<"\nEnter the connecting
vertices:\nx|y\n"; for(int i=0;i<n;i++)
{ cin>>x>>y;
m.addEd(x,y); }
cout<<"Topological Sort of the given graph:\n "; m.topoSort();  return 0; }
```



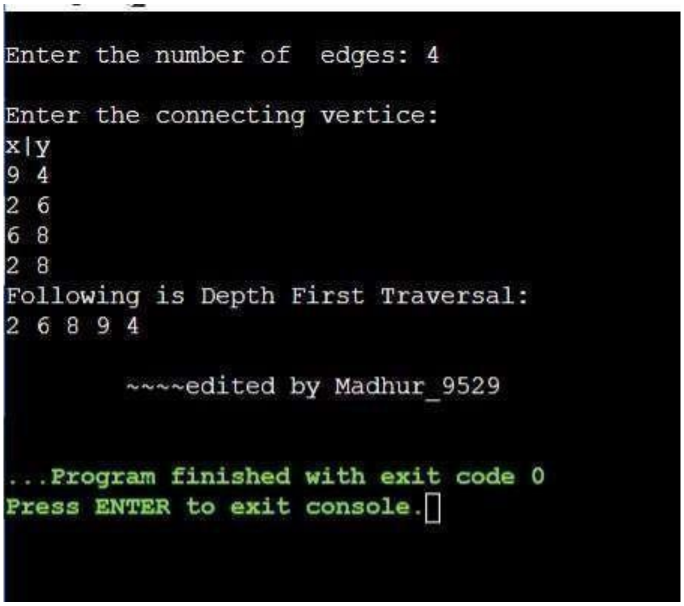
```
#include<iostream>
#include <list>
#include <stack>
using namespace std;
class M { int
n; list<int>
*adj;
void topologicalSort(int v, bool visited[], stack<int> &Stack)
{
visited[v] = true;
list<int>::iterator i;
for (i = adj[v].begin(); i != adj[v].end(); ++i)
if (!visited[*i])
topologicalSort(*i, visited, Stack);
Stack.push(v);
}
public:
M(int n)
{ this->n = n;
adj = new list<int> [n];
}
void addEd(int v, int w)
{
adj[v].push_back(w);
}
void topoSort()
{
stack<int> Stack; bool *visited =
new bool[n]; for (int i = 0; i < n; i++)
visited[i] = false; for (int i = 0; i < n;
i++) if (visited[i] == false)
topologicalSort(i, visited, Stack);
while (Stack.empty() == false)
{
cout << Stack.top() << " ";
Stack.pop();
}
}
};
int main()
{
int x,y,n;
cout<<"\nEnter no. edges: ";
cin>>n;
M m(n);
cout<<"\nEnter the connecting vertices:\nx|y\n";
for(int i=0;i<n;i++)
{
cin>>x>>y;
m.addEd(x,y);
}
cout<<"Topological Sort of the given graph:\n ";
m.topoSort();
cout<<"\n\n\t~~~~~edited by Madhur_9529"<<endl;
return 0;
}
```

Complexity Analysis:

1.DFS on an undirected graph:
Time complexity is O(V + E), where V is the number of vertices and E is the number of edges in the graph. Space complexity is O(V), since an extra visited array of size V is required.

2.DFS to find the topological sort of a directed acyclic graph: Time complexity is O(V+E). The above algorithm is simply DFS with an extra stack. So, time complexity is the same as DFS. Space complexity is O(V), since an extra space is required for the stack. Output:

1. DFS on an undirected graph:



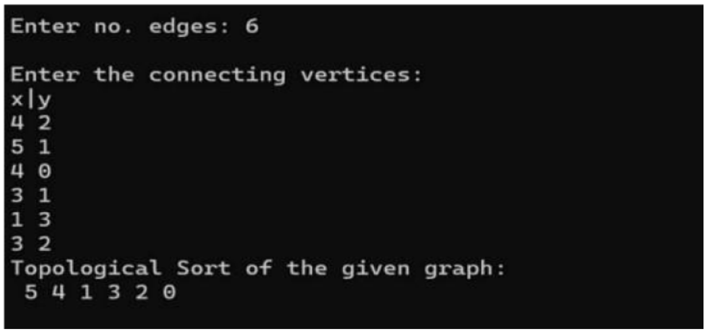
```
Enter the number of  edges: 4

Enter the connecting vertice:
x|y
9 4
2 6
6 8
2 8
Following is Depth First Traversal:
2 6 8 9 4

        ~~~~edited by Madhur_9529


...Program finished with exit code 0
Press ENTER to exit console.
```

2. DFS to find the topological sort of a directed acyclic graph:



```
Enter no. edges: 6

Enter the connecting vertices:
x|y
4 2
5 1
4 0
3 1
1 3
3 2
Topological Sort of the given graph:
 5 4 1 3 2 0
```

Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			