**Worksheet-3.1**

**Student Name:-** Pushpraj Roy **UID:-** 20BCS9866

**Branch:-** BE- CSE **Section/Group:-** WM\_617 “A”

**Subjetct Code:-** 20CSP-312 **Semester:-** 5th

**Subject Name:-** DAA Lab

1. **Aim/Overview of the practical: -**

# Code and analyze to do a depth-first search (DFS) on an undirected

# graph. Implementing an application of DFS such as

# (i) to find the topological sort of a directed acyclic graph, OR

# (ii) to find a path from source to goal in a maze.

# Objective :-

Objective of this experiment is to present the one of the application of DFS by implementing topological sort on a directed acyclic graph or by finding a path from source to goal in a maze.

1. **Algorithm/Flowchart :-**

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 neighbours of N that are in the ready state (whose

STATUS = 1) and set their

STATUS = 2 (waiting state)

[END OF LOOP]

Step 6: EXIT

1. **Steps for experiment/practical/Code :-**

#include <iostream>

#include <list>

#include <stack>

using namespace std;

class Graph {

int V;

list<int>\* adj;

void topologicalSortUtil(int v, bool visited[],

stack<int>& Stack);

public:

Graph(int V);

void addEdge(int v, int w);

void topologicalSort();

};

Graph::Graph(int V) {

this->V = V;

adj = new list<int>[V];

}

void Graph::addEdge(int v, int w) {

adj[v].push\_back(w);

}

void Graph::topologicalSortUtil(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])

topologicalSortUtil(\*i, visited, Stack);

Stack.push(v);

}

void Graph::topologicalSort()

{

stack<int> Stack;

bool\* visited = new bool[V];

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

visited[i] = false;

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

if (visited[i] == false)

topologicalSortUtil(i, visited, Stack);

while (Stack.empty() == false) {

cout << Stack.top() << " ";

Stack.pop();

}

}

int main()

{

Graph g(26);

g.addEdge(25, 10);

g.addEdge(25, 0);

g.addEdge(20, 0);

g.addEdge(20, 5);

g.addEdge(10, 15);

g.addEdge(15, 5);

cout<<"Directed Acyclic Graph (DAG) created successfully.\n";

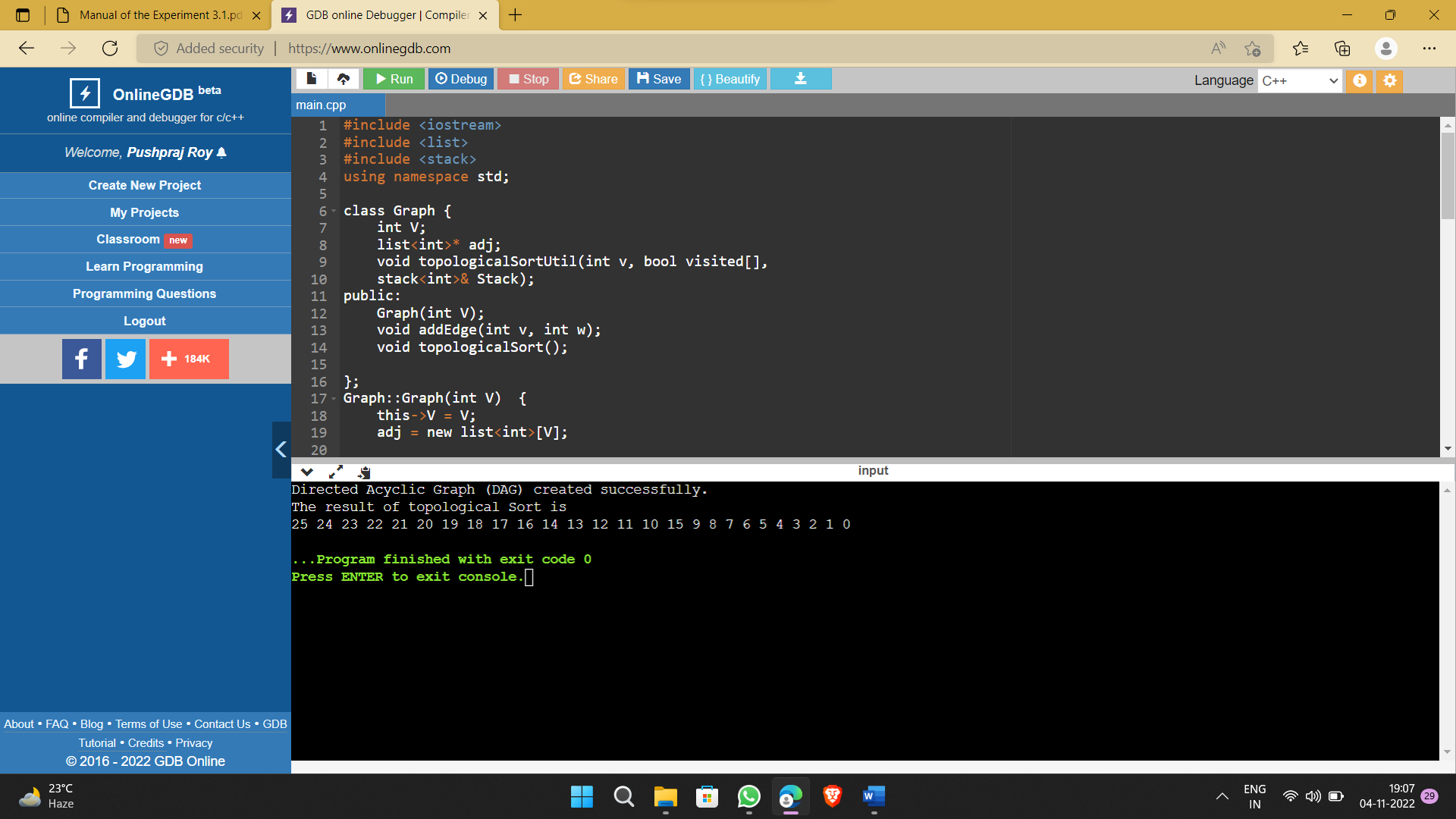
cout << "The result of topological Sort is\n";

g.topologicalSort();

return 0;

}

1. **Result/Output :-**

****