**Experiment No. – 2.2**

**Student Name: Deepak Saini UID: 20BCS4066**

**Branch: 20BCC1 Section/Group: A**

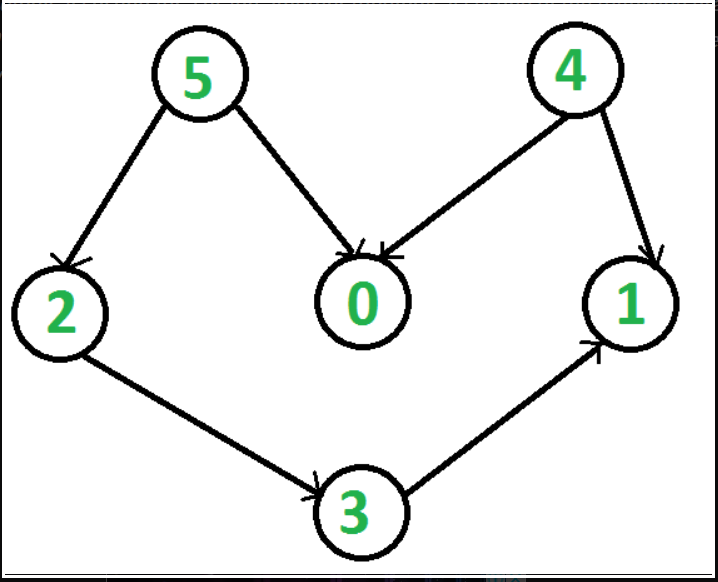
**Semester:**  **5th Date of Performance: 24/09/2022**

**Subject Name: ADVANCED PROGRAMMING LAB**

**Subject Code: 20CSP-334**

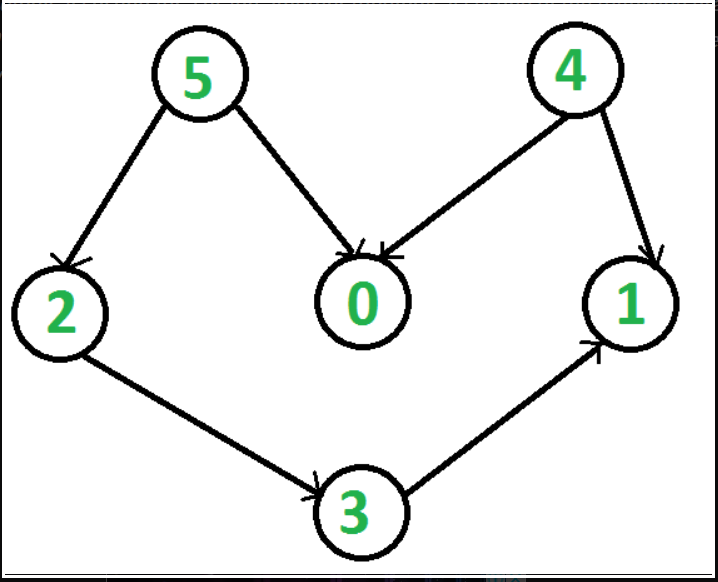
1. **Aim/Overview of the practical:**

Obtain the Topological ordering of vertices in a given digraph.



1. **Task to be done:**

Obtain the Topological ordering of vertices in a given digraph



1. **Steps for practical**:

**Approach:**

* Create a stack to store the nodes.
* Initialize visited array of size N to keep the record of visited nodes.
* Run a loop from 0 till N
* if the node is not marked True in visited array
* Call the recursive function for topological sort and perform the following steps.
* Mark the current node as True in the visited array.
* Run a loop on all the nodes which has a directed edge to the current node
* if the node is not marked True in the visited array:
* Recursively call the topological sort function on the node
* Push the current node in the stack.
* Print all the elements in the stack.

**4 . Code:**

// A C++ program to print topological

// sorting of a DAG

#include <bits/stdc++.h>

using namespace std;

// Class to represent a graph

class Graph {

// No. of vertices'

int V;

// Pointer to an array containing adjacency listsList

list<int>\* adj;

// A function used by topologicalSort

void topologicalSortUtil(int v, bool visited[],

stack<int>& Stack);

public:

// Constructor

Graph(int V);

// function to add an edge to graph

void addEdge(int v, int w);

// prints a Topological Sort of

// the complete graph

void topologicalSort();

};

Graph::Graph(int V)

{

this->V = V;

adj = new list<int>[V];

}

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

{

// Add w to v’s list.

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

}

// A recursive function used by topologicalSort

void Graph::topologicalSortUtil(int v, bool visited[],

stack<int>& Stack)

{

// Mark the current node as visited.

visited[v] = true;

// Recur for all the vertices

// adjacent to this vertex

list<int>::iterator i;

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

if (!visited[\*i])

topologicalSortUtil(\*i, visited, Stack);

// Push current vertex to stack

// which stores result

Stack.push(v);

}

// The function to do Topological Sort.

// It uses recursive topologicalSortUtil()

void Graph::topologicalSort()

{

stack<int> Stack;

// Mark all the vertices as not visited

bool\* visited = new bool[V];

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

visited[i] = false;

// Call the recursive helper function

// to store Topological

// Sort starting from all

// vertices one by one

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

if (visited[i] == false)

topologicalSortUtil(i, visited, Stack);

// Print contents of stack

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

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

Stack.pop();

}

}

// Driver Code

int main()

{

// Create a graph given in the above diagram

Graph g(6);

g.addEdge(5, 2);

g.addEdge(5, 0);

g.addEdge(4, 0);

g.addEdge(4, 1);

g.addEdge(2, 3);

g.addEdge(3, 1);

cout << "Following is a Topological Sort of the given "

"graph \n";

// Function Call

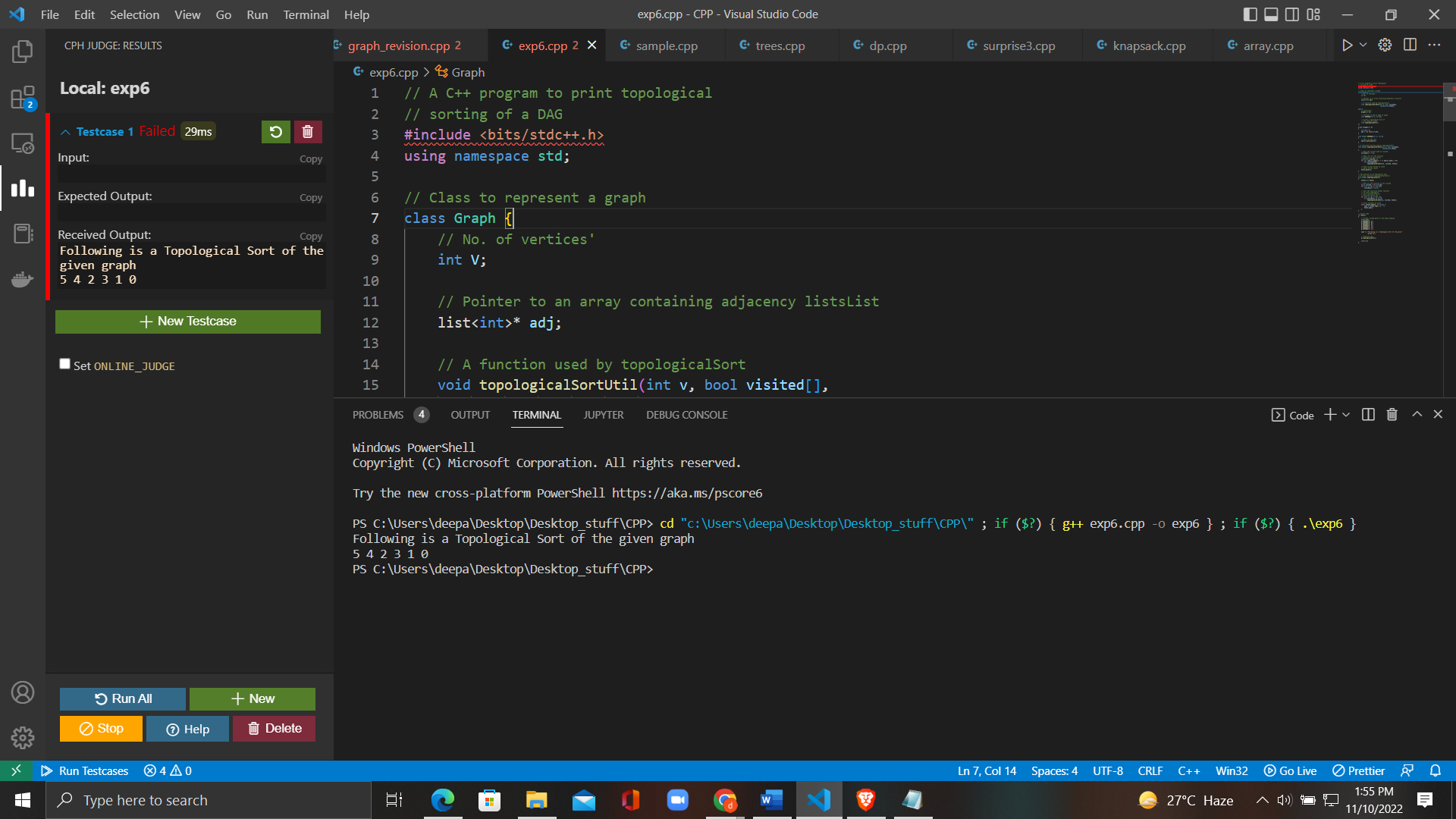
g.topologicalSort();

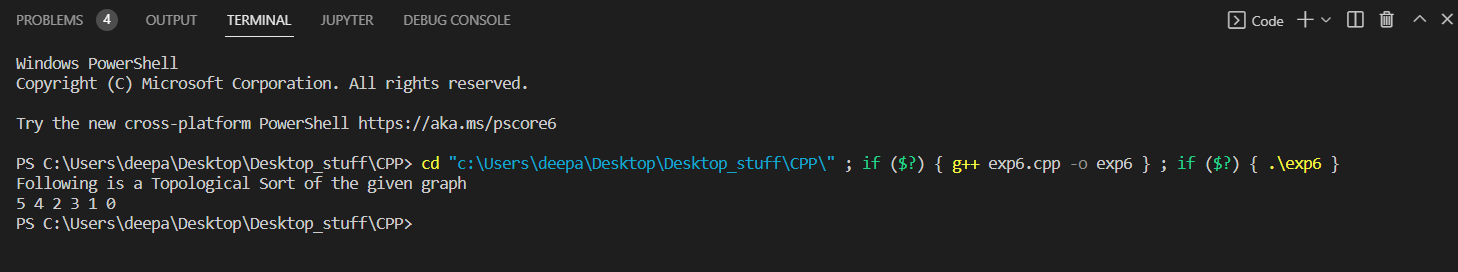
return 0;

}

**5.** **Output:**

**a)**





**6. Learning Outcomes:**

* To learn the basics of Graph to how to take inputs.
* To learn the approach to how to solve problems related to graph.
* To learn about how to use stack data structure.

**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. |  |  |  |