**Algorithm Analysis and Data Structures**

**CS 5343.502(Spring 2020)**

**Assignment 5**

**Divya Birla**

**2021514344**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**QUESTION:**

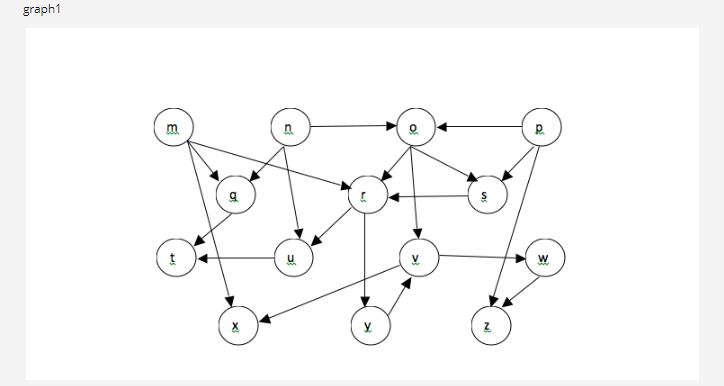
a. Write DFS topological sort program.  Your program must catch cycles.

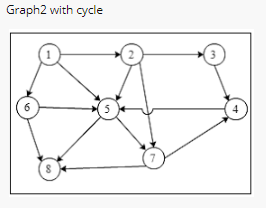
b. Write BFS topological sort program.  Your program must catch cycles.

Run your programs on the attached graphs.

submit screen shots of execution

submit the code files



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SOURCE CODE:**

**PART 1: DFS Topological Sorting**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Course: CS 5343.502 – Spring 2020

\* Assignment <5>

\* Name: Divya Birla

\* E-mail: dxb190021@utdallas.edu

\* Submitted: <04/23/20>

This program performs DFS topological sort on two graphs.

Graph 1, having 14 vertices and 21 edges, is Directed Acyclic Graph.

Graph 2, having 8 vertices and 14 edges, is a Directed Cyclic Graph.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <vector>

#include <queue>

#include<stack>

using namespace std;

// Creating the graph 1 using adjacency list

vector< vector<int> > CreateGraphAdjList()

{

vector< vector<int> > adjList;

// To create adjacency list size

const int n = 14;

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

{

// intialising list with vertices

vector<int> list;

adjList.push\_back(list);

}

//Adding edges using 2D vector

// adding edges directed out of vertex 0

adjList[0].push\_back(4);

adjList[0].push\_back(5);

adjList[0].push\_back(11);

// adding edges directed out of vertex 1

adjList[1].push\_back(2);

adjList[1].push\_back(4);

adjList[1].push\_back(8);

// adding edges directed out of vertex 2

adjList[2].push\_back(5);

adjList[2].push\_back(6);

adjList[2].push\_back(9);

// adding edges directed out of vertex 3

adjList[3].push\_back(2);

adjList[3].push\_back(6);

adjList[3].push\_back(13);

// adding edges directed out of vertex 4

adjList[4].push\_back(7);

// adding edges directed out of vertex 5

adjList[5].push\_back(8);

adjList[5].push\_back(12);

// adding edges directed out of vertex 6

adjList[6].push\_back(5);

// adding edges directed out of vertex 8

adjList[8].push\_back(7);

// adding edges directed out of vertex 9

adjList[9].push\_back(10);

adjList[9].push\_back(11);

// adding edges directed out of vertex 10

adjList[10].push\_back(13);

// adding edges directed out of vertex 12

adjList[12].push\_back(9);

return adjList;

}

// Creating the graph 2 using adjacency list

vector< vector<int> > CreateGraphAdjList2() {

vector< vector<int>> adjList2;

// To create adjacency list size

const int n = 8;

// intialising list with vertices

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

vector<int> list;

adjList2.push\_back(list);

}

// adding edges directed out of vertex 1 of Graph 2

adjList2[0].push\_back(1);

adjList2[0].push\_back(4);

adjList2[0].push\_back(5);

// adding edges directed out of vertex 2 of Graph 2

adjList2[1].push\_back(2);

adjList2[1].push\_back(4);

adjList2[1].push\_back(6);

// adding edges directed out of vertex 3 of Graph 2

adjList2[2].push\_back(3);

// adding edges directed out of vertex 4 of Graph 2

adjList2[3].push\_back(4);

// adding edges directed out of vertex 5 of Graph 2

adjList2[4].push\_back(6);

adjList2[4].push\_back(7);

// adding edges directed out of vertex 6 of Graph 2

adjList2[5].push\_back(4);

adjList2[5].push\_back(7);

// adding edges directed out of vertex 7 of Graph 2

adjList2[6].push\_back(3);

adjList2[6].push\_back(7);

return adjList2;

}

//To carry out DFS topological sorting on Graphs

void DFSTopoSortUtil(vector<vector<int>> adjList, int vertex, bool visited[],bool cyclic[],stack<int> &s, bool &cycle) {

visited[vertex] = true;

cyclic[vertex] = true;

for (int i = 0; i < adjList[vertex].size(); i++) {

int adjNode = adjList[vertex][i];

if (visited[adjNode] == false)

DFSTopoSortUtil(adjList, adjNode, visited,cyclic,s,cycle);

else

if (cyclic[adjNode] == true)

cycle = true;

}

cyclic[vertex] = false;

s.push(vertex+1);

}

bool DFSTopoSort(vector<vector<int>> &adjList, int n) {

bool cycle = false;

stack<int> s;

bool\* visited= new bool[n];

bool\* cyclic = new bool[n];

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

visited[i] = false;

cyclic[i] = false;

}

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

if (visited[i] == false) {

DFSTopoSortUtil(adjList, i, visited,cyclic,s,cycle);

}

}

//To print sorted list if there is no cycle

if (cycle == false) {

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

if (n == 14) {

cout << (char)(s.top() + 108)<<" ";//To print sorted list as alphabets

s.pop();

}

else {

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

s.pop();

}

}

return true;

}

else

return false;

}

int main()

{

cout << "\n--PROGRAM TO PERFORM DFS TOPOLOGICAL SORT--\n";

vector< vector<int> > adjList = CreateGraphAdjList();

int n1 = adjList.size();

cout << "\n Carrying out DFS Topological Sort for the graph 1:" << endl;

cout << "\n ";

if(DFSTopoSort(adjList, n1))

{

cout << "\n ";

}

else {

cout << "\n\n There is a cycle in the graph 1 " << endl;

}

cout << "\n Carrying out DFS Topological Sort for the graph 2:" << endl;

vector< vector<int> > adjList2 = CreateGraphAdjList2();

int n2 = adjList2.size();

if (DFSTopoSort(adjList2, n2))

{

cout << "\n";

}

else {

cout << "\n There is a cycle in the graph 2 \n\n";

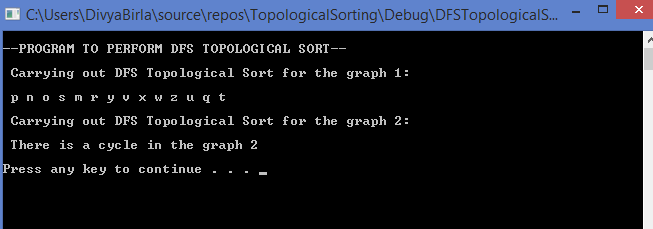
}

system("pause");

return 0;

}

**OUTPUT:**



**PART 2: BFS Topological Sorting**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Course: CS 5343.502 – Spring 2020

\* Assignment <5>

\* Name: Divya Birla

\* E-mail: dxb190021@utdallas.edu

\* Submitted: <04/23/20>

This program performs BFS topological sort on two graphs.

Graph 1, having 14 vertices and 21 edges, is Directed Acyclic Graph.

Graph 2, having 8 vertices and 14 edges, is a Directed Cyclic Graph.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

vector<int> indegree(14, 0);

// Creating the graph 1 using adjacency list

vector< vector<int> > CreateGraphAdjList()

{

vector< vector<int> > adjList;

// To create adjacency list size

const int n = 14;

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

{

// intialising list with vertices

vector<int> list;

adjList.push\_back(list);

}

//Adding edges using 2D vector

// adding edges directed out of vertex 0

adjList[0].push\_back(4);

indegree[4]++;

adjList[0].push\_back(5);

indegree[5]++;

adjList[0].push\_back(11);

indegree[11]++;

// adding edges directed out of vertex 1

adjList[1].push\_back(2);

indegree[2]++;

adjList[1].push\_back(4);

indegree[4]++;

adjList[1].push\_back(8);

indegree[8]++;

// adding edges directed out of vertex 2

adjList[2].push\_back(5);

indegree[5]++;

adjList[2].push\_back(6);

indegree[6]++;

adjList[2].push\_back(9);

indegree[9]++;

// adding edges directed out of vertex 3

adjList[3].push\_back(2);

indegree[2]++;

adjList[3].push\_back(6);

indegree[6]++;

adjList[3].push\_back(13);

indegree[13]++;

// adding edges directed out of vertex 4

adjList[4].push\_back(7);

indegree[7]++;

// adding edges directed out of vertex 5

adjList[5].push\_back(8);

indegree[8]++;

adjList[5].push\_back(12);

indegree[12]++;

// adding edges directed out of vertex 6

adjList[6].push\_back(5);

indegree[5]++;

// adding edges directed out of vertex 8

adjList[8].push\_back(7);

indegree[7]++;

// adding edges directed out of vertex 9

adjList[9].push\_back(10);

indegree[10]++;

adjList[9].push\_back(11);

indegree[11]++;

// adding edges directed out of vertex 10

adjList[10].push\_back(13);

indegree[13]++;

// adding edges directed out of vertex 12

adjList[12].push\_back(9);

indegree[9]++;

return adjList;

}

// Creating the graph 2 using adjacency list

vector< vector<int> > CreateGraphAdjList2() {

vector< vector<int>> adjList2;

// To create adjacency list size

const int n = 8;

// intialising list with vertices

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

vector<int> list;

adjList2.push\_back(list);

}

//To resize indegree vector as per the size of Graph 2 vertices list

indegree.resize(8, 0);

// adding edges directed out of vertex 1 of Graph 2

adjList2[0].push\_back(1);

indegree[1]++;

adjList2[0].push\_back(4);

indegree[4]++;

adjList2[0].push\_back(5);

indegree[5]++;

// adding edges directed out of vertex 2 of Graph 2

adjList2[1].push\_back(2);

indegree[2]++;

adjList2[1].push\_back(4);

indegree[4]++;

adjList2[1].push\_back(6);

indegree[6]++;

// adding edges directed out of vertex 3 of Graph 2

adjList2[2].push\_back(3);

indegree[3]++;

// adding edges directed out of vertex 4 of Graph 2

adjList2[3].push\_back(4);

indegree[4]++;

// adding edges directed out of vertex 5 of Graph 2

adjList2[4].push\_back(6);

indegree[6]++;

adjList2[4].push\_back(7);

indegree[7]++;

// adding edges directed out of vertex 6 of Graph 2

adjList2[5].push\_back(4);

indegree[4]++;

adjList2[5].push\_back(7);

indegree[7]++;

// adding edges directed out of vertex 7 of Graph 2

adjList2[6].push\_back(3);

indegree[3]++;

adjList2[6].push\_back(7);

indegree[7]++;

return adjList2;

}

//To carry out BFS topological sorting on Graphs

bool BFSTopoSort(vector< vector<int> > adjList, int n) {

//Vector which will contain sorted elements

vector<int> sortedList;

int visited = 0;

queue<int> q;

//To push vertices with 0 indegree into the queue for processing

for (int v = 0; v < n; v++)

if (indegree[v] == 0)

q.push(v);

while (!q.empty()) {

int u = q.front();

q.pop();

//Store sorted elements

sortedList.push\_back(u+1);

vector<int>::iterator it;

for (it = adjList[u].begin(); it != adjList[u].end(); it++)

if (--indegree[\*it] == 0)// decrease indgree of vertices connected to the visited element and check if indegree for any changes to 0

q.push(\*it);//Push vertices with changed indegree as 0 into the queue

//To keep track of number of vertices of the graph that have been visited

visited++;

}

//To check if entire graph has been processed or there is a cycle

if (visited == n) {

if (n == 14) {

for (int j = 0; j < sortedList.size(); j++)

cout << " " << (char)(sortedList[j] + 108) << " ";//To print sorted list as alphabets

return true;

}

else {

for (int j = 0; j < sortedList.size(); j++)

cout << " " << sortedList[j] << " ";

return true;

}

}

else

return false;

}

int main()

{

cout << "\n--PROGRAM TO PERFORM BFS TOPOLOGICAL SORT--\n";

vector< vector<int> > adjList = CreateGraphAdjList();

int n1 = adjList.size();

cout << "\n Carrying out BFS Topological Sort for the graph 1:" << endl;

if (BFSTopoSort(adjList, n1))

{

cout << "\n";

}

else {

cout << "\n There is a cycle in the graph 1 " <<endl;

}

cout << "\n Carrying out BFS Topological Sort for the graph 2:" << endl;

vector< vector<int> > adjList2 = CreateGraphAdjList2();

int n2 = adjList2.size();

if (BFSTopoSort(adjList2, n2))

{

cout << "\n";

}

else {

cout << "\n There is a cycle in the graph 2 \n\n";

}

system("pause");

return 0;

}

**OUTPUT:**

