Rajshahi University of Engineering & Technology

CSE 2202: Sessional Based on CSE 2201

Lab Report 05

Date: December 11, 2018

Submitted to

Biprodip Pal

Instructor, CSE 2101 & CSE 2202

Assistant Professor, Dept. of CSE

Submitted by

Fuad Al Abir

Roll: 1603021

Section: A

Dept. of CSE

Sessional – Cycle 5 – Problem A

Use DFS to determine if the root/starting node can be traversed again visiting the other (all or some of) nodes.

Given Graph:

Code:

```
/*------------------------------
    I N T R O D U C T I O N
------------------------------
Author:     Fuad Al Abir
Date:       November 18, 2018
Name:       dfs.cpp
Objective:  This program determines if the root node can be traversed by visiting
             the other (all or some of) nodes.
*/
/*-----------------------------------
    H E A D E R   F U N C T I O N
-----------------------------------
Header: iostream
Reason: Input/Output stream
Header: vector
Reason: For representing graph
Header: stack
Reason: For traversing the graph using DFS
*/
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
/*-------------------------------------------------
    U S E R   D E F I N E D   F U N C T I O N
-------------------------------------------------
Function:   void addEdge(vector <int> adjList[], int u, int v);
Reason:     To add the edges between nodes
Function:   void adjacencyList(vector <int> adjList[], int graph[][2], int node, int edge);
Reason:     To create adjacency list and checking the primary objective of this
             program.
Function:   void dfs(vector <int> adjList[], int start, int node);
Reason:     To traverse the graph using DFS algorithm
*/
void addEdge(vector <int> adjList[], int u, int v)
{
    adjList[u].push_back(v);
}
void adjacencyList(vector <int> adjList[], int graph[][2], int node, int edge)
{
    for(int i = 0; i < edge; i++)
    {
        addEdge(adjList, graph[i][0], graph[i][1]);
    }
    cout << "\nAdjacency list:" << endl;
    for (int i = 0; i < node; i++)
```

```
    {
        cout << "Node: " << i;
        for (int j = 0; j < adjList[i].size(); j++)
        {
           cout << " -> " << adjList[i][j];
        }
        cout << endl;
    }
}
void dfs(vector <int> adjList[], int start, int node)
{
    int visited[node];
    int edgeID[node];
    for(int i = 0; i < node; i++)
    {
        visited[i] = 0;
        edgeID[i] = 0;
    }

    cout << "\nDepth First Search: ";
    stack <int> s;
    s.push(start);
    while(!s.empty())
    {
        int u = s.top();
        if(visited[u] == 0)
        {
            cout << u << " ";
        }

        // Checking if a node can be traversed to the root
        if(u != start)
        {
            for (int i = 0; i < node; i++)
            {
                for (int j = 0; j < adjList[i].size(); j++)
                {
                    cout << adjList[i][j] << " ";
                    if (adjList[i][j] == start)
                    {
                        // Printing "YES" if the root node
                        // can be traversed from
                        // a particular node
                        cout << "\n\nYES\n";
                        return;
                    }
                }
            }
        }

        visited[u] = 1;
        s.pop();

        while(edgeID[u] < adjList[u].size())
        {
            int v = adjList[u][edgeID[u]];
            edgeID[u]++;
            if(visited[v] == 0)
            {
                s.push(u);
                s.push(v);
                break;
            }
        }
    }
    cout << endl;
}
```

```
/*-------------------------------
    M A I N   F U N C T I O N
-------------------------------*/

int main()
{
    int node;
    int edge;
    int start;

    cout << "Number of node(s): ";
    cin >> node;
    cout << "Number of edge(s): ";
    cin >> edge;

    int graph[edge][2];
    int *adjMat[node];
    int i = node;
    while(i--)
    {
        adjMat[i] = new int[node];
    }
    vector <int> adjList[node];

    for(int i = 0; i < edge; i++)
    {
        for(int j = 0; j < 2; j++)
        {
            graph[i][j] = 0;
        }
    }

    for(int i = 0; i < node; i++)
    {
        for(int j = 0; j < node; j++)
        {
            adjMat[i][j] = 0;
        }
    }

    cout << "Enter the connected nodes, pairwise: ";
    for(int c = 0; c < edge; c++)
    {
        for(int r = 0; r < 2; r++)
        {
            cin >> graph[c][r];
        }
    }

    adjacencyList(adjList, graph, node, edge);
    cout << "\nGraph Traversal starting Node: ";
    cin >> start;
    dfs(adjList, start, node);

    return 0;
}
```

Input/Output:

```
Number of node(s): 6
Number of edge(s): 7
Enter the connected nodes, pairwise: 0 1 1 2 2 3 2 4 2 5 5 0 3 0

Adjacency list:
Node: 0 -> 1
Node: 1 -> 2
Node: 2 -> 3 -> 4 -> 5
Node: 3 -> 0
Node: 4
Node: 5 -> 0

Graph Traversal starting Node: 5

Depth First Search: 5 0 1 2 3 4 5

YES
```

```
Number of node(s): 6
Number of edge(s): 7
Enter the connected nodes, pairwise: 0 1 1 2 2 3 2 4 2 5 5 0 3 0

Adjacency list:
Node: 0 -> 1
Node: 1 -> 2
Node: 2 -> 3 -> 4 -> 5
Node: 3 -> 0
Node: 4
Node: 5 -> 0

Graph Traversal starting Node: 3

Depth First Search: 3 0 1 2 3

YES
```

Discussion: The program determines if a root node can be traversed again after visiting other nodes of the graph using Depth First Search Traversing algorithm. It prints "YES" in the console if the root node can be traversed and also prints the path if so whereas prints nothing if it can't.