Practical # 14

Objective:

Discuss Graph data structure. ***Design*** *a C++ program that implement traversal technique for a searching vertex in a graph.*

Theory:

In this Lab, we discuss the Graph data structure. Graph is a non-linear data structure. It contains a set of points known as nodes (or vertices) and a set of links known as edges (or Arcs). Here edges are used to connect the vertices. A graph is defined as a collection of vertices and arcs in which vertices are connected with arcs. Graph traversal is a technique used for a searching vertex in a graph. There are two graph traversal techniques and they are as follows:

1. **DFS (Depth First Search)**
2. **BFS (Breadth First Search)**

DFS (Depth First Search)

DFS traversal of a graph produces a **spanning tree** as final result. **Spanning Tree** is a graph without loops. We use **Stack data structure** with maximum size of total number of vertices in the graph to implement DFS traversal. Following steps are used to implement DFS traversal.

Step 1 - Define a Stack of size total number of vertices in the graph.
Step 2 - Select any vertex as **starting point** for traversal. Visit that vertex and push it on to the Stack.
Step 3 - Visit any one of the non-visited **adjacent** vertices of a vertex which is at the top of stack and push it on to the stack.
Step 4 - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
Step 5 - When there is no new vertex to visit then use **back tracking** and pop one vertex from the stack.
Step 6 - Repeat steps 3, 4 and 5 until stack becomes Empty.
Step 7 - When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

Lab Objectives:

- To be able to write C++ program for performing traversal technique on Graph data structure.

C++ program: *Write C++ program to traverse Graph data structure using DFS traversal technique.*

```
#include <iostream>
#include <list>
using namespace std;
//graph class for DFS traversal
class DFSGraph
{
int V;    // No. of vertices
list<int> *adjList;    // adjacency list
void DFS_util(int v, bool visited[]);  // A function used by DFS
public:
    // class Constructor
DFSGraph(int V)
    {
 this->V = V;
 adjList = new list<int>[V];
    }
    // function to add an edge to graph
void addEdge(int v, int w){
adjList[v].push_back(w); // Add w to v's list.
    }

void DFS();    // DFS traversal function
};
void DFSGraph::DFS_util(int v, bool visited[])
{
    // current node v is visited
visited[v] = true;
cout << v << " ";

    // recursively process all the adjacent vertices of the node
list<int>::iterator i;
```

```
for(i = adjList[v].begin(); i != adjList[v].end(); ++i)
if(!visited[*i])
DFS_util(*i, visited);
}
// DFS traversal
void DFSGraph::DFS()
{
    // initially none of the vertices are visited
bool *visited = new bool[V];
for (int i = 0; i < V; i++)
visited[i] = false;
    // explore the vertices one by one by recursively calling  DFS_util
for (int i = 0; i < V; i++)
if (visited[i] == false)
DFS_util(i, visited);
}
int main()
{
    // Create a graph
DFSGraph gdfs(5);
gdfs.addEdge(0, 1);
gdfs.addEdge(0, 2);
gdfs.addEdge(0, 3);
gdfs.addEdge(1, 2);
gdfs.addEdge(2, 4);
gdfs.addEdge(3, 3);
gdfs.addEdge(4, 4);
cout << "Depth-first traversal for the given graph:"<<endl;
gdfs.DFS();
return 0;
}
```

OUTPUT

```
Depth-first traversal for the given graph:
0 1 2 4 3
```

Review Questions/ Exercise:

1. Explain Breadth first search traversal technique.

   _____

   _____

   _____

2. Write a C++ program traverse the Graph using Breadth first search traversal technique.

   _____

   _____

   _____

Name: _____

Roll #: _____

Date: _____

Subject Teacher

Remarks: