Lab 6

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

The basic idea is as follows:
Pick a starting node and push all its adjacent nodes into a stack.
Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack. Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

1. Objective of the Experiment

After completing this lab, the student should be able to:

- Clearly understand the difference between Stack and Queue.
- Implementation of DFS in python

2. Concept Map

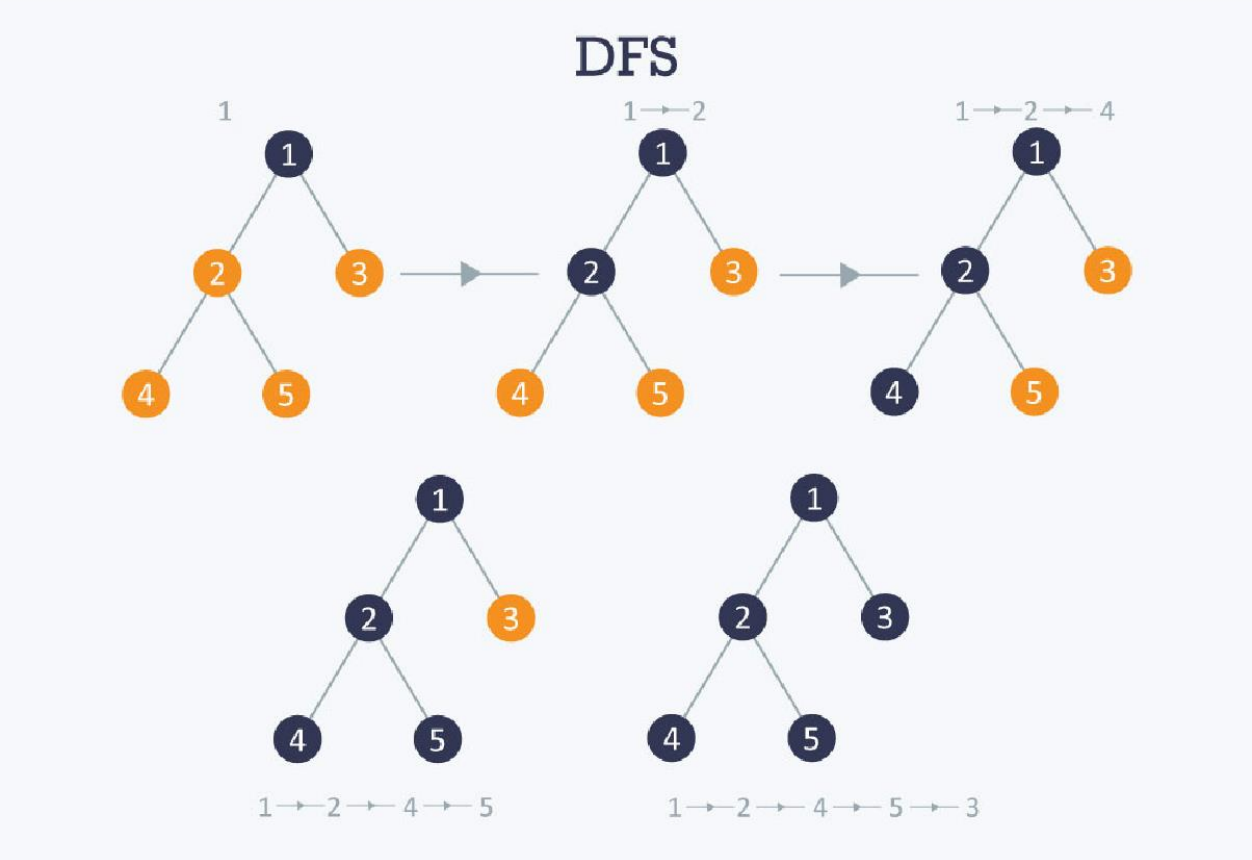
3.1 Depth First Search

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.

This recursive nature of DFS can be implemented using stacks. The basic idea is as follows:
Pick a starting node and push all its adjacent nodes into a stack.
Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack. Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked.
This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

Pseudocode

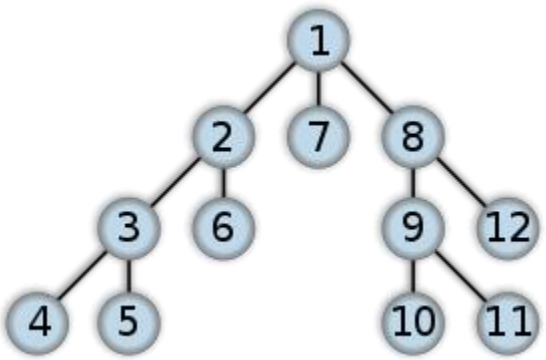
```
DFS-iterative (G, s):                                          //Where G is graph and s is source vertex
  let S be stack
  S.push( s )              //Inserting s in stack
  mark s as visited.
  while ( S is not empty):
      //Pop a vertex from stack to visit next
      v  =  S.top( )
     S.pop( )
     //Push all the neighbours of v in stack that are not visited
    for all neighbours w of v in Graph G:
        if w is not visited :
                 S.push( w )
                mark w as visited


DFS-recursive(G, s):
    mark s as visited
    for all neighbours w of s in Graph G:
        if w is not visited:
            DFS-recursive(G, w)
```



4.

4.1 Problem description

Write a program to implement the graph in python.



5. Procedure & Tools

5.1 Tools

Anaconda, python language

5.2 Walkthrough Task [Expected time = 40 mins]

5.2.1 Implementation of DFS in C++

```
#include<iostream>
#include<list>
using namespace std;

// Graph class represents a directed graph
// using adjacency list representation
class Graph
{
    int V;    // No. of vertices

    // Pointer to an array containing
    // adjacency lists
    list<int> *adj;

    // A recursive function used by DFS
    void DFSUtil(int v, bool visited[]);
public:
    Graph(int V);   // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w);

    // DFS traversal of the vertices
    // reachable from v
    void DFS(int v);
};
```

```
Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Graph::DFSUtil(int v, bool visited[])
{
    // Mark the current node as visited and
    // print it
    visited[v] = true;
    cout << v << " ";

    // 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])
            DFSUtil(*i, visited);
}
void Graph::DFS(int v)
{
    // 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 print DFS traversal
    DFSUtil(v, visited);
}
```

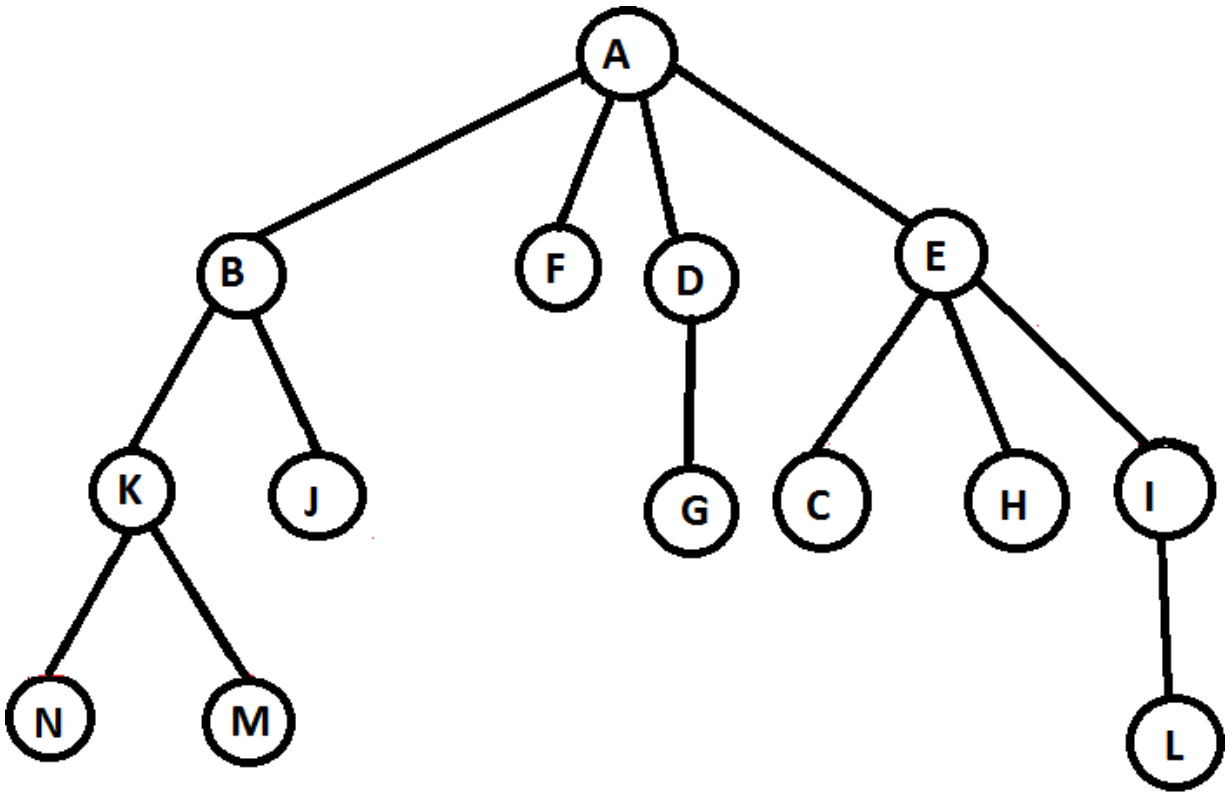
```
int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Depth First Traversal"
            " (starting from vertex 2) \n";
    g.DFS(2);

    return 0;
}
```

6. Practice Tasks

This section will provide more practice exercise which you need to finish during the lab. You need to finish the task in the required time.

6.1 Practice Task 1 [Expected time = 40 mins]

You are given the following tree whose starting node is **A** and Goal node is **G.**



You are required implement the following graph in python and then apply the following search

a) Depth First Search using Stack.

6.2 Outcomes

After completing this lab, students will know the implementation of DFS in python. Students will be able to solve problems using DFS in python.