Data Structures and Algorithms (CS09203)

Lab Report

Name:	Rehman ullah baig
Registration #:	CSU-F16118
Lab Report #:	09
Dated:	11-06-2018
Submitted To:	Mr. Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 9
The Depth-first search algorithm

Objective
To understand and implement The Depth-first search algorithm.

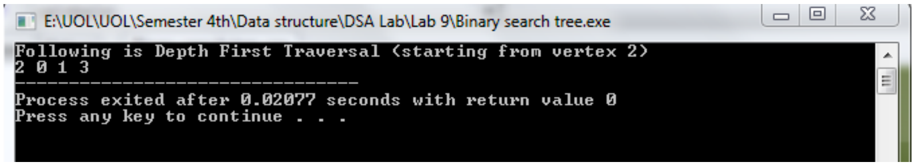
Software Tool
1. Windows 7
2. Dev C++
3. Miktex

1 Theory

A standard DFS implementation puts each vertex of the graph into one of two categories:

Visited Not Visited The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
The DFS algorithm works as follows:
1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of stack.

3. Keep repeating steps 2 and 3 until the stack is empty.



E:\UOL\UOL\Semester 4th\Data structure\DSA Lab\Lab 9\Binary search tree.exe

```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
--------------------------------
Process exited after 0.02077 seconds with return value 0
Press any key to continue . . .
```

Figure 1: Data entering into different locations

2 Task

2.1 Procedure: Task 1

Tin the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we dont mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Depth First Traversal of the following graph is 2, 0, 1, 3

2.2 Procedure: Task 2

```
#include<iostream>
#include<list>
using namespace std;
class Graph
{
    int V;

    list<int> *adj;

    void DFSUtil(int v, bool visited[]);
public:
    Graph(int V);

    void addEdge(int v, int w);
    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);
}

void Graph::DFSUtil(int v, bool visited[])
{
    visited[v] = true;
    cout << v << "␣";
    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)
{
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    DFSUtil(v, visited);
}

int main()
{
    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;
}
```

3 Conclusion

Depth first search is an interesting algorithm, and as you might suspect, it is particularly well suited for inspecting if a graph is connected; if the tree returned by depth first search contains all vertices in the graph, it is connected, otherwise, it is not.