OCTOBER 20,2020

DESIGN AND ANALYSIS OF ALGORITHMS

(QUESTION: 7, DEPTH FIRST SEARCH)
EXERCISE 7

SUBMITTED BY-
NAME - RAJ KRISHNA
ROLL NO - 1805233
BATCH - CSE-G1
GROUP - B

1. The objective of the Experiment

The objective of the experiment is to traverse all nodes present in tree or graph data structure via **Depth First Search**.

2. Solution Code

```
#include<bits/stdc++.h>
using namespace std;
vector<int> adjnodes[100];
bool isVisited[100];
int main()
{
        int vertices=9,edges=22;
        for(int i = 0; i < vertices; i++)
        {
        isVisited[i + 1] = false;
        }
        for(int i = 0; i < edges; i++)
        {
        int u, v;
        cin >> u >> v;
        adjnodes[u].push_back(v);
        adjnodes[v].push_back(u);
        }
        int start=0;
        stack<int> s;
        s.push(start);
        cout<<"DFS(all trees):| ";
        while(!s.empty())
        {
         int popped = s.top();
    if(isVisited[popped]==true)
    cout<<"";
    else{
         isVisited[popped] = true;
         cout<<popped<<" ";
         }
    s.pop();
         for(int n: adjnodes[popped])
         {
                  if(!isVisited[n])
                           s.push(n);
         }
        }
cout<<"| ";
        for(int i = 0; i <vertices; i++) {
         if(!isVisited[i])
        cout<<i<<" ";
        }
        return 0;
}
```

3. Summary of the program

Depth first traversal or Depth first Search is a recursive algorithm for searching all the vertices of a graph or tree data structure.

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

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The DFS algorithm works as follows:

4. Start by putting any one of the graph's vertices on top of a stack.
5. Take the top item of the stack and add it to the visited list.
6. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
7. Keep repeating steps 2 and 3 until the stack is empty.

The time complexity of the DFS algorithm is represented in the form of O(V + E), where V is the number of nodes and E is the number of edges.

The space complexity of the algorithm is O(V).

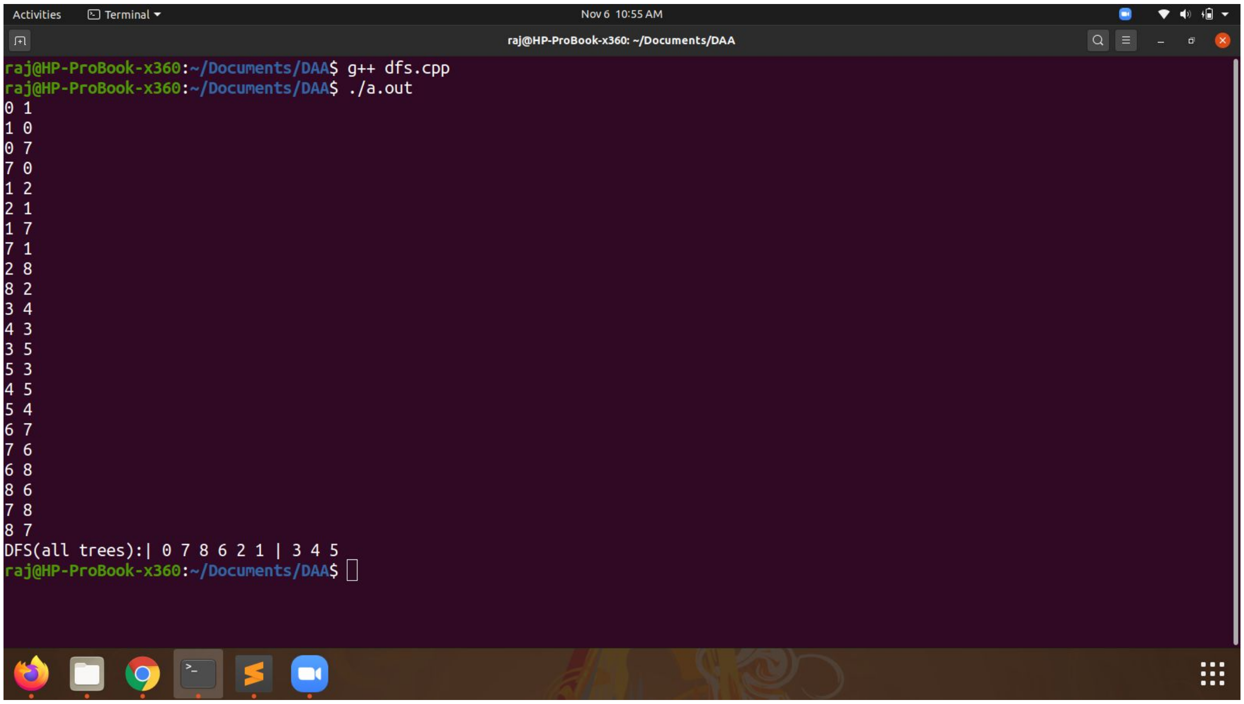
4. Sample Output

Input-

```
0 1
1 0
0 7
7 0
1 2
2 1
1 7
7 1
2 8
8 2
3 4
4 3
3 5
5 3
4 5
5 4
6 7
7 6
6 8
8 6
7 8
8 7
```

Output-

DFS(all trees):| 0 7 8 6 2 1 | 3 4 5



```
raj@HP-ProBook-x360:~/Documents/DAA$ g++ dfs.cpp
raj@HP-ProBook-x360:~/Documents/DAA$ ./a.out
0 1
1 0
0 7
7 0
1 2
2 1
1 7
7 1
2 8
8 2
3 4
4 3
3 5
5 3
4 5
5 4
6 7
7 6
6 8
8 6
7 8
8 7
DFS(all trees):| 0 7 8 6 2 1 | 3 4 5
raj@HP-ProBook-x360:~/Documents/DAA$ 
```