Name of the Report: DFS and Strongly Connected Components

Problem Statement

The objective of this lab is to understand and implement Depth First Search (DFS) and to identify Strongly Connected Components in a directed graph using Kosaraju's Algorithm.

Given Graph:

- **Vertices:** `{1, 2, 3, 4, 5, 6}`
- **Directed Edges:** `1 -> 2, 2 -> 3, 3 -> 1, 3 -> 4, 4 -> 5, 5 -> 4, 5 -> 6`

Tasks

1. Perform DFS traversal starting from vertex 1 and determine the traversal order.
2. Find all Strongly Connected Components (SCCs) using DFS-based Kosaraju's Algorithm.

Algorithm

Kosaraju's Algorithm for SCC:

1. Perform DFS on the original graph and store vertices according to their finishing times.
2. Transpose the graph by reversing all edges.
3. Perform DFS on the transposed graph by considering vertices in decreasing order of finishing time.
4. Each DFS tree formed in this step represents one Strongly Connected Component.

Task 1: Implement DFS

```
#include<bits/stdc++.h>
using namespace std;

void travel(vector<vector<int>>&adj,vector<bool>&visited, int st){
    cout<<st+1<<' ';
    visited[st] = true;
    for(auto v:adj[st]){
        if(!visited[v]){
            travel(adj,visited,v);
        }
    }
}

void dfs(vector<vector<int>>&adj, int st){
    vector<bool>visited(adj.size(),false);
    travel(adj,visited,st);
}

int main(){
    int v,e;cin>>v>>e;
    vector<vector<int>>adj(v);
    int x,y;
    for(int i=0;i<e;i++){
        cin>>x>>y;
        adj[x-1].push_back(y-1);
    }
    int st;cin>>st;
    dfs(adj,st-1);
}
```

Task 2:Find Strongly Connected Components using DFS

```
#include<bits/stdc++.h>
using namespace std;

void dfs1(vector<vector<int>>&adj,vector<bool>&visited,stack<int>&st,int
node){
    visited[node] = true;
    for(auto v:adj[node]){
        if(!visited[v]){
            dfs1(adj,visited,st,v);
        }
    }
    st.push(node);
}
void dfs2(vector<vector<int>>&adj,vector<bool>&visited,int node){
    cout<<node+1<<' ';
    visited[node] = true;
    for(auto v:adj[node]){
        if(!visited[v]){
            dfs2(adj,visited,v);
        }
    }
}

void scc(vector<vector<int>>graph,int v){
    vector<bool>visited(v);
    stack<int>st;

    for(int i=0;i<v;i++){
        if(!visited[i]){
           dfs1(graph,visited,st,i);
        }
    }

    vector<vector<int>>revGraph(v);
    fill(visited.begin(),visited.end(),false);

    for(int i=0;i<v;i++){
        for(auto v:graph[i]){
            revGraph[v].push_back(i);
        }
    }

    while(!st.empty()){
        int k = st.top();
        st.pop();

        if(!visited[k]){
            dfs2(revGraph,visited,k);
            cout<<endl;
        }
    }

}
```

```
int main(){
    int v,e;cin>>v>>e;
    vector<vector<int>>adj(v);
    for(int i=0;i<e;i++){
        int x,y;cin>>x>>y;
        adj[x-1].push_back(y-1);
    }
    scc(adj,v);
}
```

Results and Analysis

The DFS traversal successfully visited all vertices starting from node 1 in the order:

```
     1 -> 2 -> 3 -> 4 -> 5 -> 6
```

DFS on transposed graph (using finishing order):

- SCCs found: `{1, 3, 2} ,{4, 5} , {6}`

This confirms that the algorithm correctly identifies components where every vertex is mutually reachable.

Discussion

Kosaraju's Algorithm is an effective method for finding Strongly Connected Components in directed graphs. Its main advantage is simplicity, as it relies on the well-known DFS technique, making it easy to understand and implement. The algorithm runs in O(V+E) time, which makes it efficient even for relatively large graphs. It accurately identifies groups of vertices where each node is mutually reachable, which is useful in applications such as network analysis, web link structure, and deadlock detection.

However, the algorithm also has some limitations. It requires two DFS passes and the construction of a transposed graph, which increases memory usage. Recursive DFS may cause stack overflow for extremely large graphs.