Assignment Sheet – 2

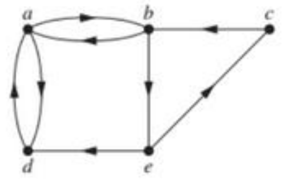
Name – Sayak Sen

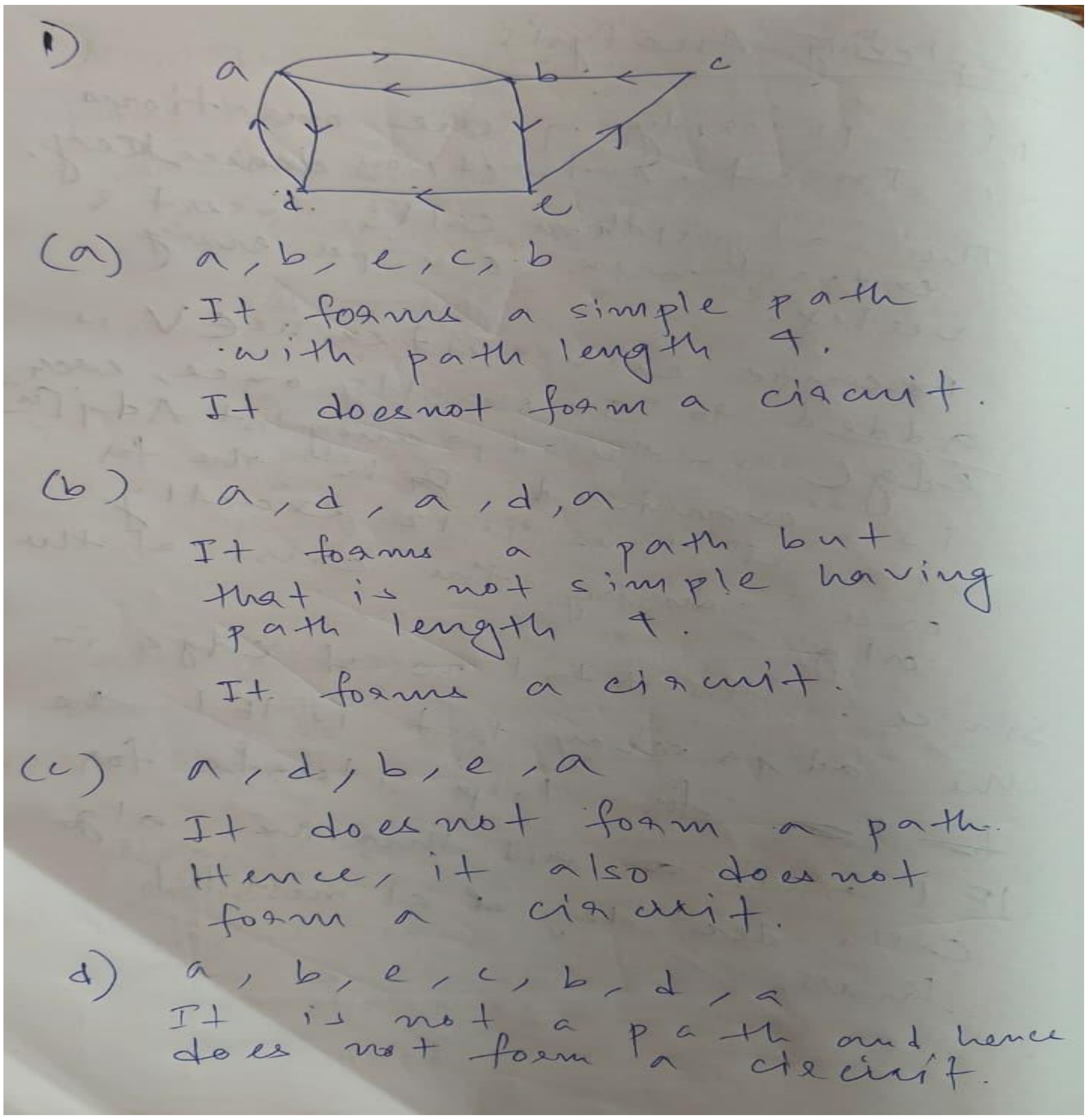
Enrollment No – 2023CSB047

1. Does each of these lists of vertices form a path in the following graph? Which paths are simple? Which are circuits? What are the lengths of those that are paths?

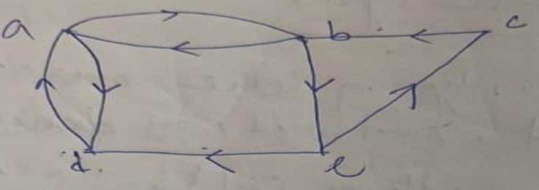
(a) a, b, e, c, b (b) a, d, a, d, a (c) a, d, b, e, a

(d) a, b, e, c, b, d, a





1)



(a) a, b, e, c, b

It forms a simple path with path length 4.

It doesnot form a circuit.

(b) a, d, a, d, a

It forms a path but that is not simple having path length 4.

It forms a circuit.

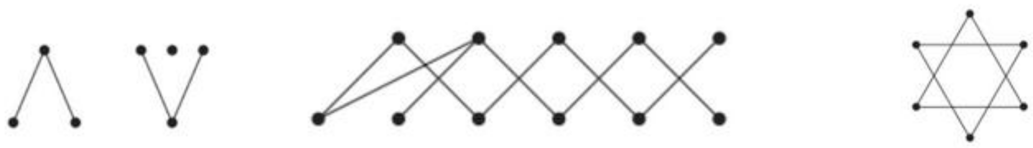
(c) a, d, b, e, a

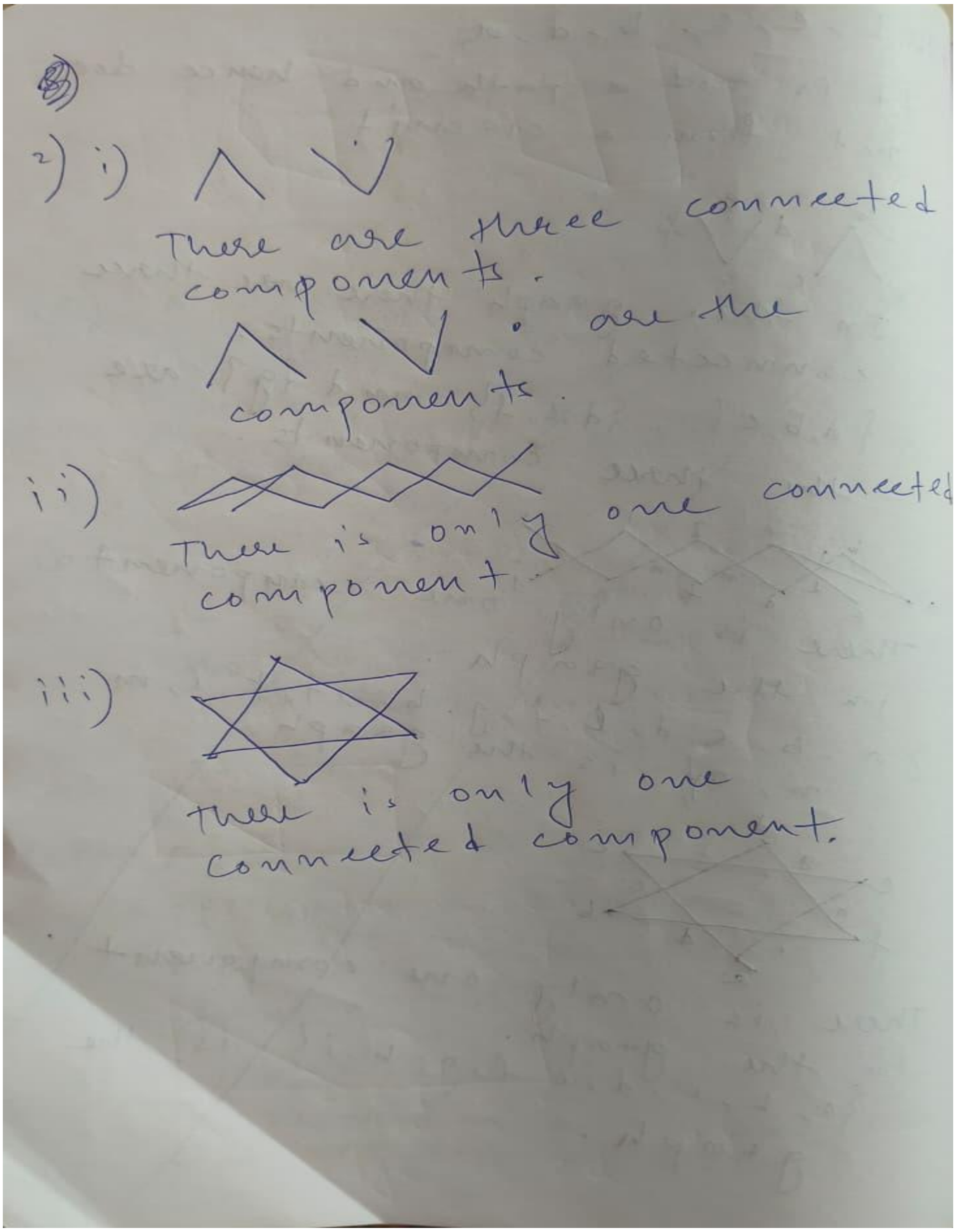
It does not form a path. Hence, it also does not form a circuit.

d) a, b, e, c, b, d, a

It is not a path and hence does not form a circuit.

2. How many connected components does each of the following graphs have? For each graph find each of its connected components.





8)

2) i)

There are three connected components.

are the components.

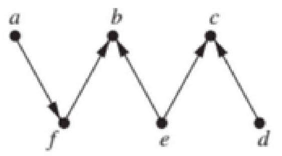
ii)

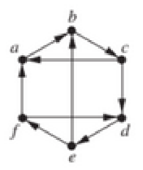
There is only one connected component.

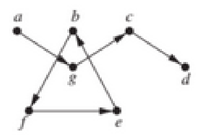
iii)

There is only one connected component.

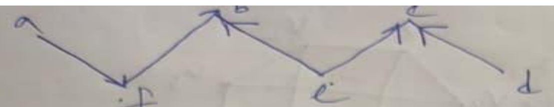
3. Determine whether each of these graphs is strongly connected and if not, whether it is weakly connected.



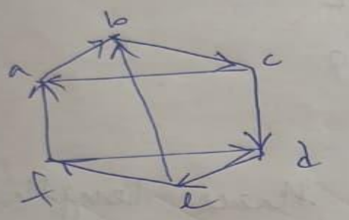




3)

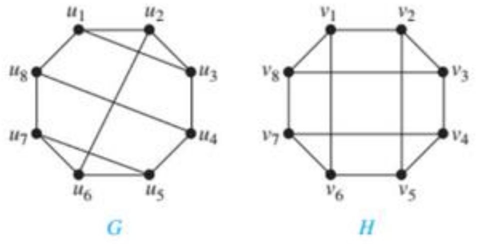


It is not a strongly connected graph but it is weakly connected

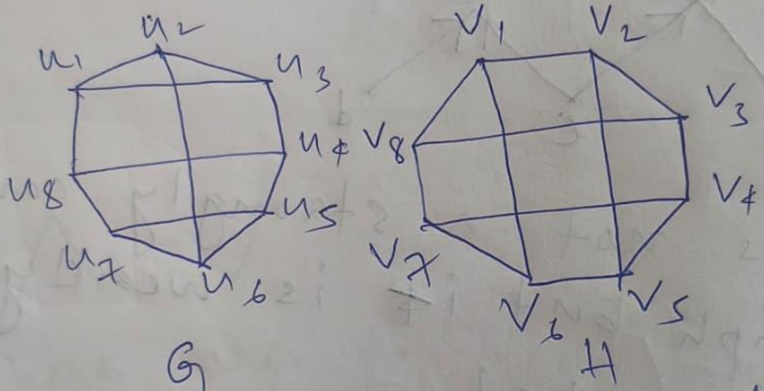


It is strongly connected as for any pair of vertex u and v there-is a path from u to v and v to u. Hence it is also weakly connected.

4. Use paths either to show that these graphs are not isomorphic or to find an isomorphism between them.



4)

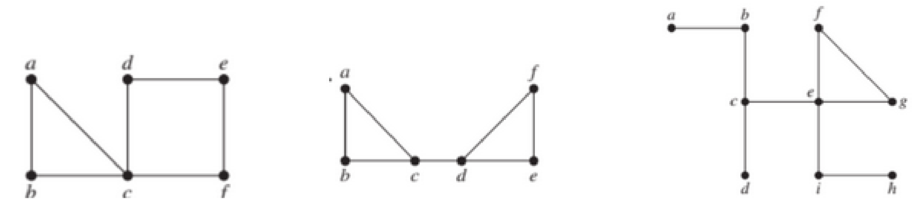


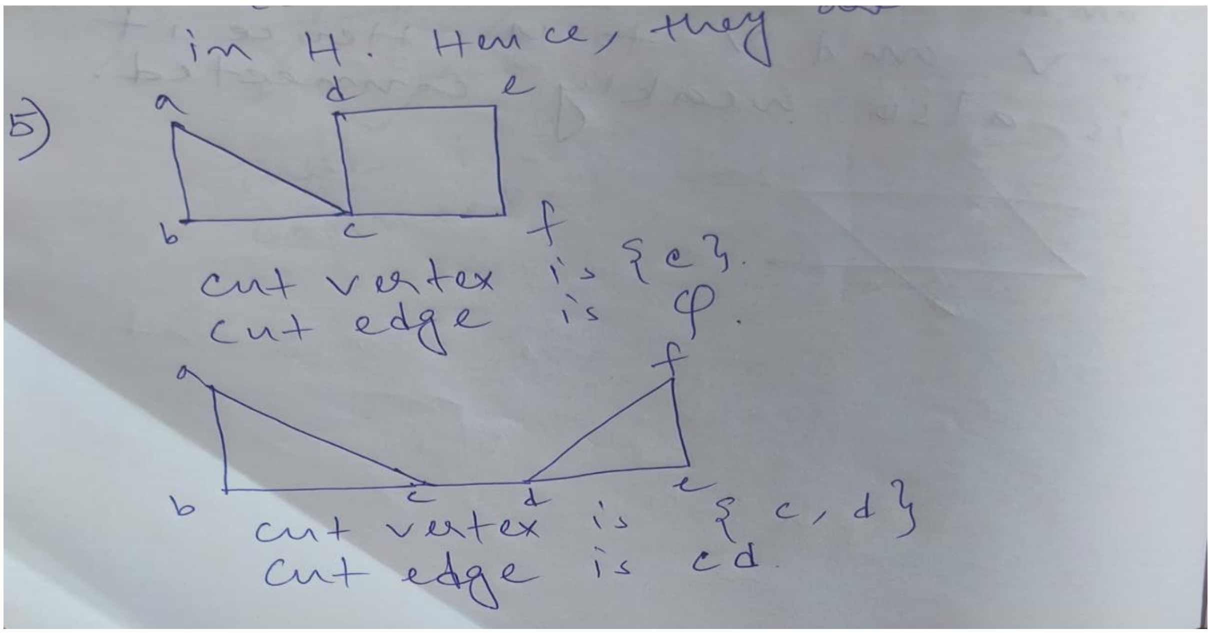
	G	H
No. of vertices	8	8
No of edges	12	12
Vertices of degree (3)	8	8

In G we have two three length cycles $\{u_1, u_2, u_3\}$ and $\{u_5, u_6, u_7\}$ but there are no three length cycles in H. Hence, they are not isomorphic

5) a d e

5. Find all the cut vertices and cut edges of the following graphs.



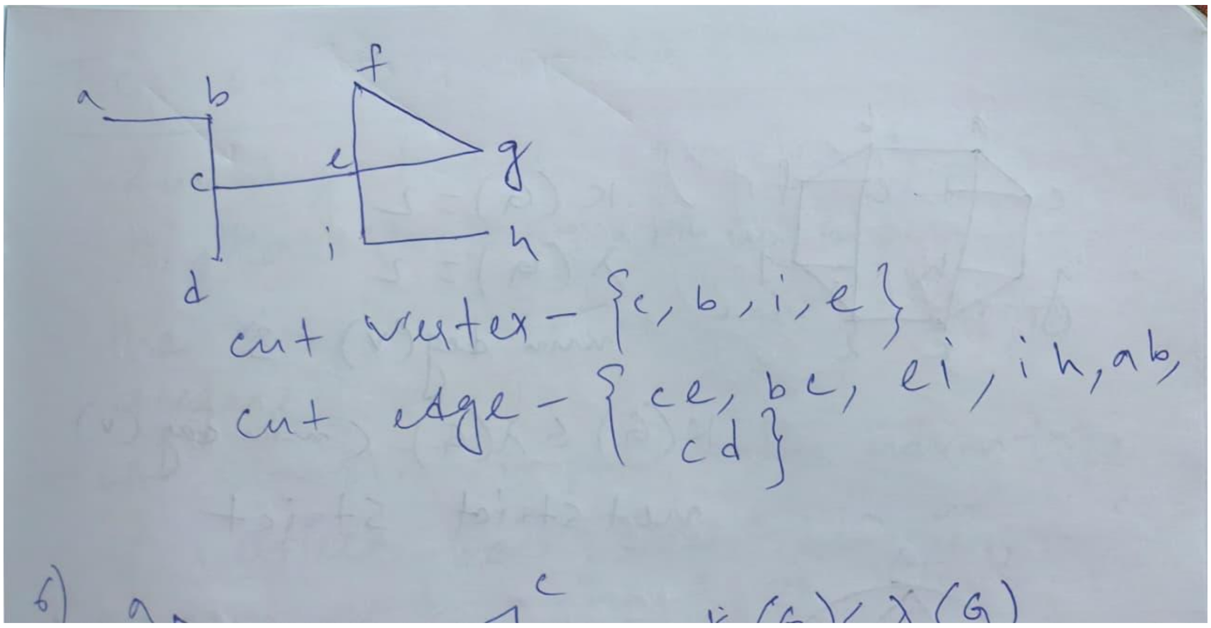

a
d
e
b
c
f
a
f
b
c
d
e

in H. Hence, they

5)

cut vertex is {c}.
cut edge is φ.

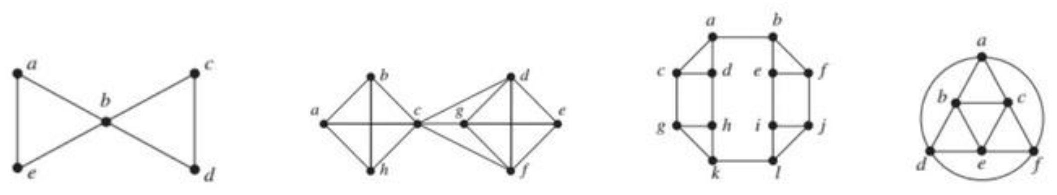
cut vertex is {c, d}
cut edge is cd.



cut vertex - {c, b, i, e}
cut edge - {ce, bc, ei, ih, ab, cd}

6) a c $\kappa(G) < \lambda(G)$

6. For each of the following graphs, find $\kappa(G)$, $\lambda(G)$, and $\min_{v \in V} deg(v)$, and determine which of the two inequalities in $\kappa(G) \le \lambda(G) \le \min_{v \in V} deg(v)$ are strict.



6)

$\kappa(G) < \lambda(G)$

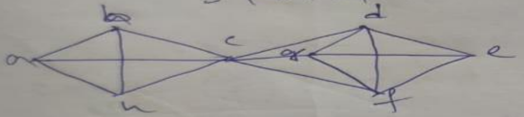
$\lambda(G) \le \min_{v \in V} deg(v)$

$\kappa(G) = 1$

$\lambda(G) = 2$

$\min deg(v) = 2$

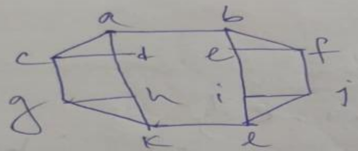
$\kappa(G) < \lambda(G)$ is strict but $\lambda(G) \le \min deg(v)$ is not strict.



$\kappa(G) = 1$

$\lambda(G) = 3$

$\min deg(v) = 3$



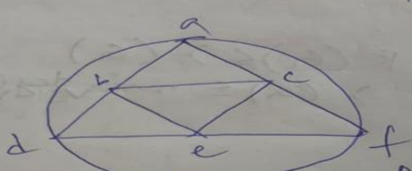
$\kappa(G) = 2$

$\lambda(G) = 2$

$\min deg(v) = 3$

$\kappa(G) \le \lambda(G) < \min deg(v)$

not strict strict



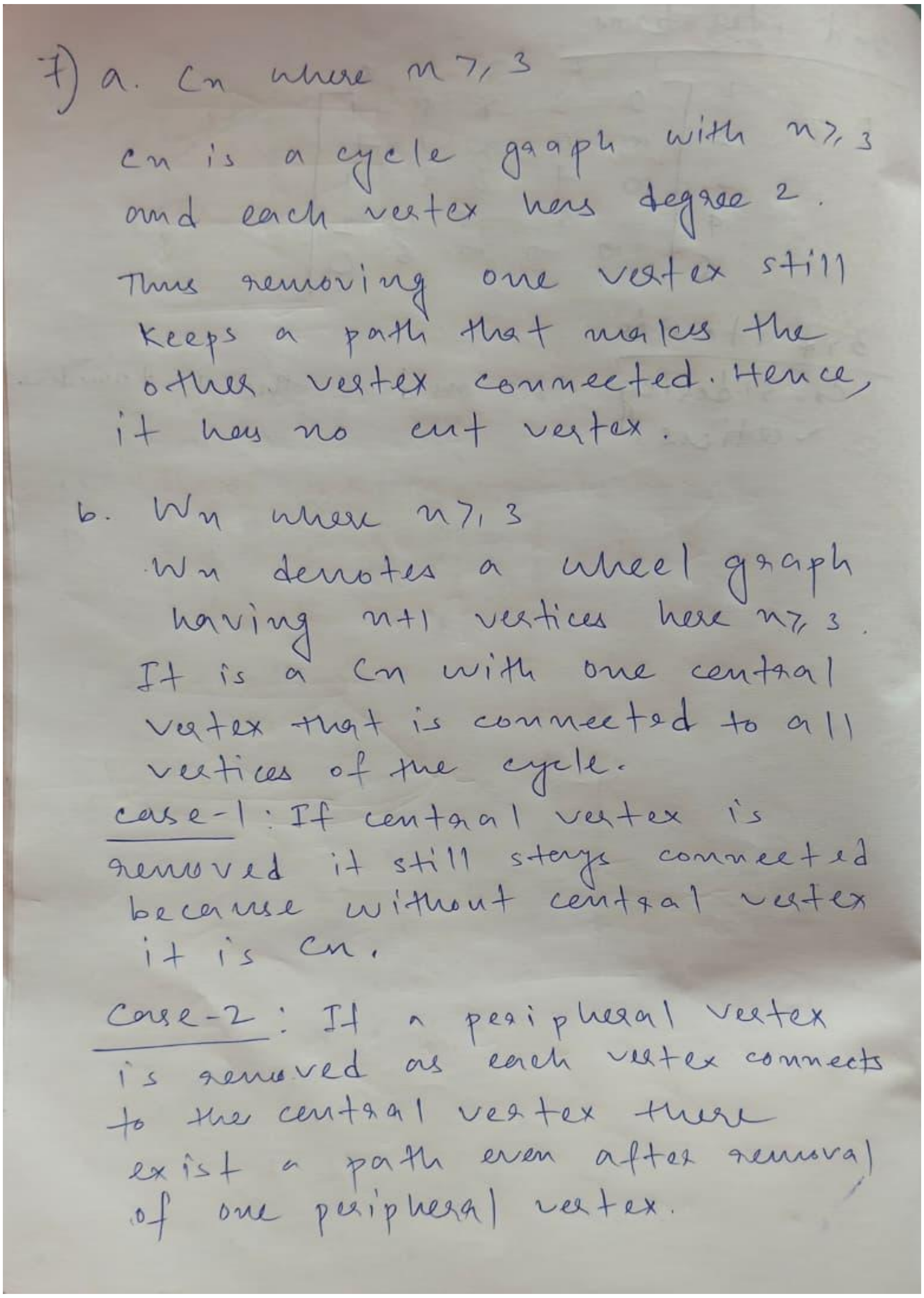
$\kappa(G) = 4$ $\min deg(v) = 4$

$\lambda(G) = 4$

None of the inequalities are strict

$\kappa(G) \le \lambda(G) \le \min deg(v)$

7. Show that each of the following graphs has no cut vertices.
   a. C_n where $n \geq 3$
   b. W_n where $n \geq 3$
   c. $K_{m,n}$ where $m \geq 2$ and $n \geq 2$



7) a. C_n where $n \geq 3$

C_n is a cycle graph with $n \geq 3$ and each vertex has degree 2.

Thus removing one vertex still keeps a path that makes the other vertex connected. Hence, it has no cut vertex.

b. W_n where $n \geq 3$

W_n denotes a wheel graph having $n+1$ vertices here $n \geq 3$.

It is a C_n with one central vertex that is connected to all vertices of the cycle.

Case-1: If central vertex is removed it still stays connected because without central vertex it is C_n.

Case-2: If a peripheral vertex is removed as each vertex connects to the central vertex there exist a path even after removal of one peripheral vertex.

c) $K_{m,n}$ where $m \geq 2$ and $n \geq 2$

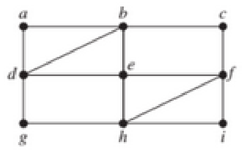
$K_{m,n}$ is a complete bipartite graph. Assume V_1 (size m) and V_2 (size n). Each vertex in V_1 is connected to each vertex in V_2 and vice-versa.

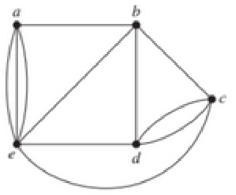
i) If we remove a vertex from V_1 The graph becomes $K_{m-1,n}$ graph where $m-1$ vertices are connected to n vertices and vice-versa.

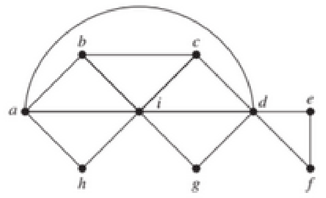
ii) If we remove a vertex from V_2. The graph becomes $K_{m,n-1}$ graph where m vertices are connected to $n-1$ vertices and vice-versa.

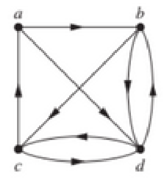
Hence, there are no cut vertices.

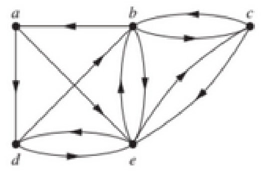
8. Determine whether the given graphs have a Euler circuit. Construct such a circuit when one exists. If no Euler circuit exists, determine whether the graph has a Euler path and construct such a path if one exists.



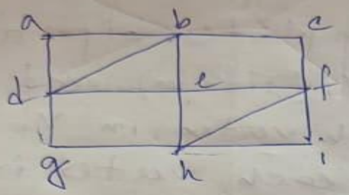








8)

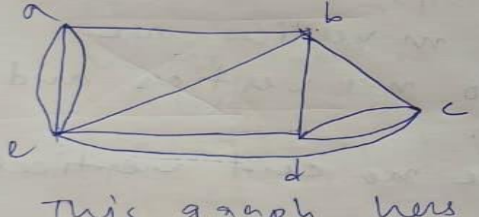


~~All the v~~

This graph has all the vertices even degree and hence has a euler circuit.

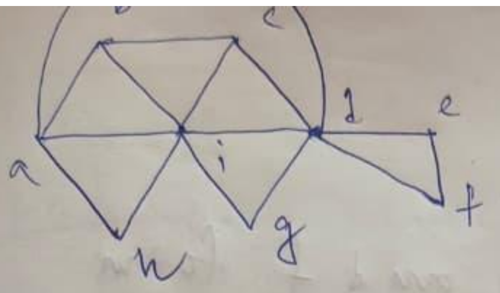
~~{a, b, c, f, i, h, f, e, b~~

{a, b, c, f, i, h, f, e h, g, d, e, b, d, a}



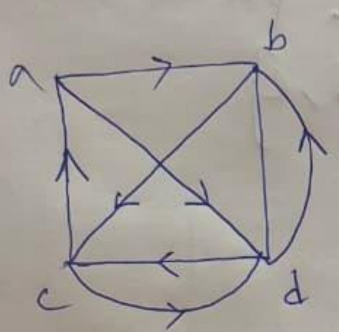
This graph has all vertices with even degree hence has a euler circuit. ~~Euler p~~

{a, b, [illegible], c, d, c, e, d, b, e, a, e, a}



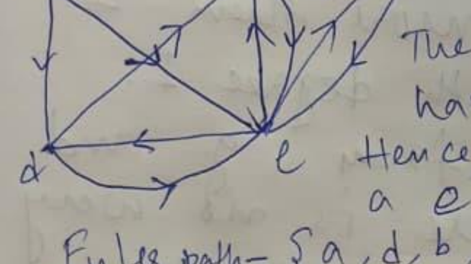
The vertices b and c have odd edges. It doesn't have euler circuit. Euler path exist.

{b, c, d, e, f, d, g, i, h, a, d, i, a, b, i, c}



The vertices a, d have odd degree. Hence, it has no euler circuit. but euler path exist.

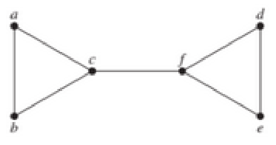
{a, b, d, b, c, d, c, a, d}

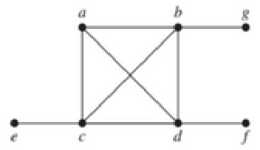


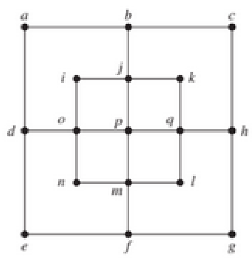
The vertices a and c have odd degree. Hence, it doesnot have a euler circuit.

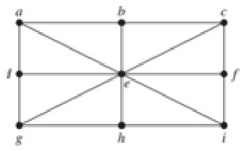
Euler path- {a, d, b, a, e, d, e, c, b, c, e, b, e}

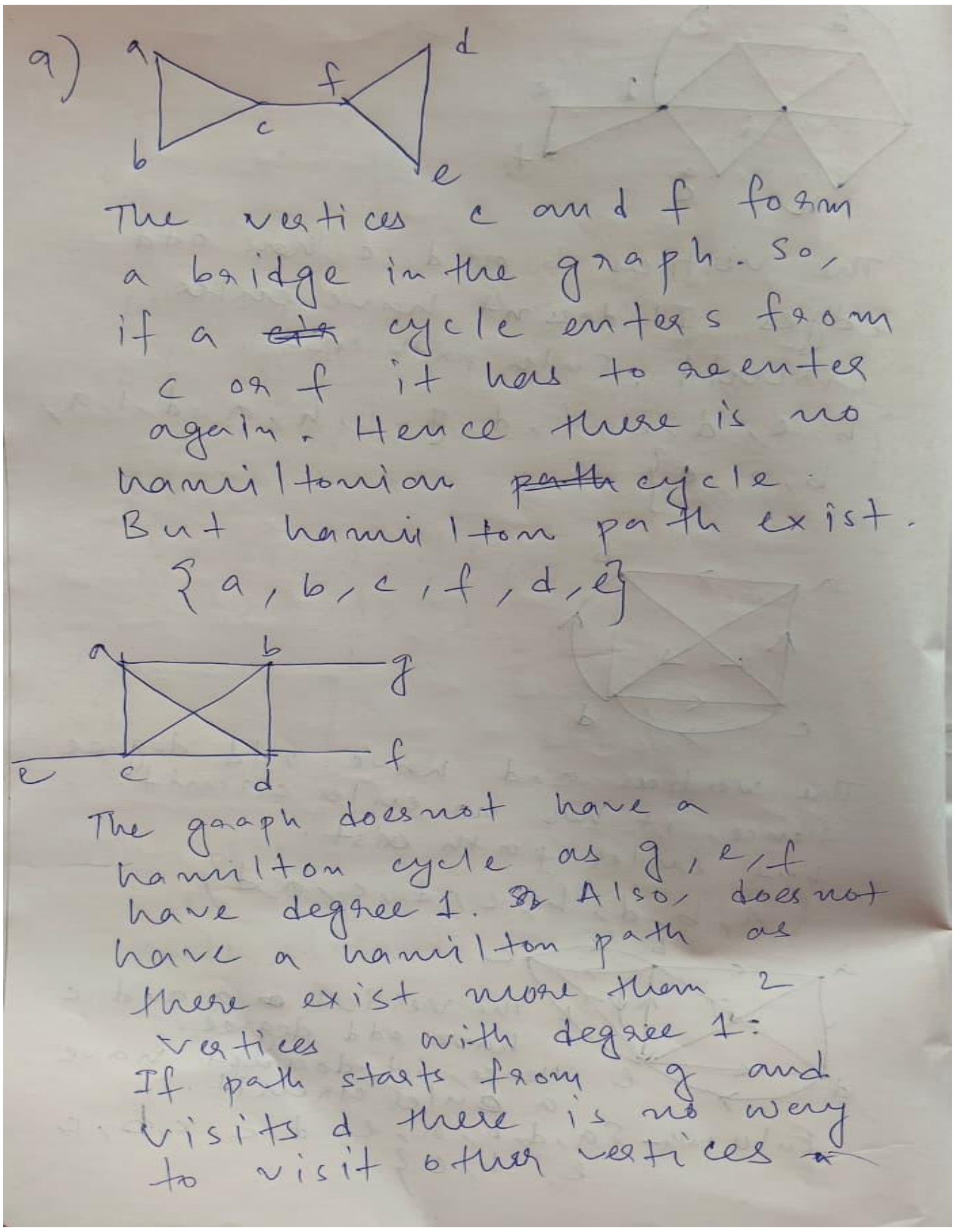
9. Determine whether the given graphs have a Hamilton circuit. If it does, find such a circuit. If it does not, give an argument to show why no such circuit exists. Do the graphs have a Hamilton path? If so, find such a path. If it does not, give an argument to show why no such path exists.







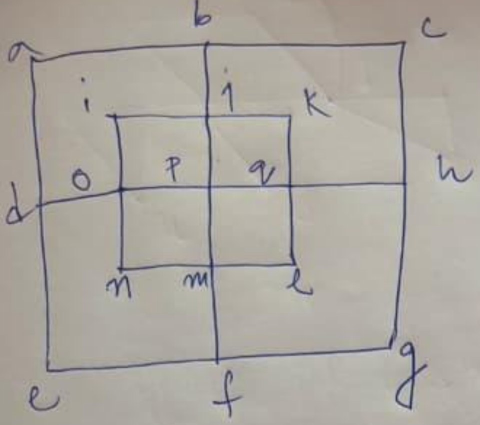




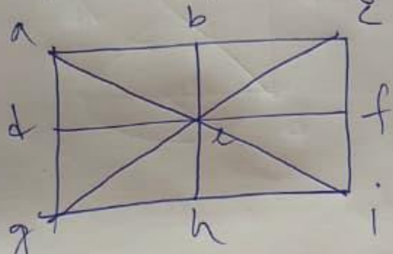
a)
a
b
c
f
d
e
a
b
g
c
d
e
f

The vertices c and f form a bridge in the graph. So, if a ~~ck~~ cycle enters from c or f it has to reenter again. Hence there is no hamiltonian ~~path~~ cycle.
But hamiltom path exist.
{a, b, c, f, d, e}

The graph does not have a hamilton cycle as g, e, f have degree 1. ~~So~~ Also, does not have a hamilton path as there exist more than 2 vertices with degree 1.
If path starts from g and visits d there is no way to visit other vertices

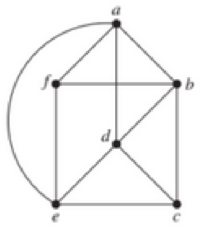


The vertex p cannot be visited with out revisiting {o, i, j, k, q, l, m, n} any one of them. Therefore, we cannot form a hamilton path and as hamilton path is not possible Hamilton cycle is also not possible.

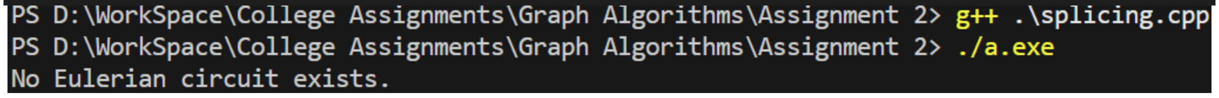


The following hamilton cycle is possible. {a, b, c, f, i, h, g, d, e, a}.

10. Write a C/C++/Java program to implement the Euler circuit finding algorithm (using the splicing technique) and test the same on the given input graph.

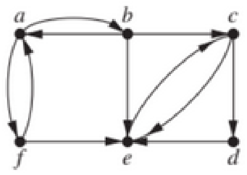


```cpp
#include <bits/stdc++.h>
using namespace std;

void eraseEdge(vector<vector<int>>& adj, int u, int v) {
    adj[u].erase(find(adj[u].begin(), adj[u].end(), v));
    adj[v].erase(find(adj[v].begin(), adj[v].end(), u));
}

vector<int> buildCycle(vector<vector<int>>& adj, int start) {
    vector<int> cycle;
    stack<int> st;
    st.push(start);

    while (!st.empty()) {
        int u = st.top();
        if (!adj[u].empty()) {
            int v = adj[u].back();
            eraseEdge(adj, u, v);
            st.push(v);
        } else {
            cycle.push_back(u);
            st.pop();
        }
    }
    return cycle;
}

vector<int> getEulerianCircuit(vector<vector<int>>& adj) {
    for (int i = 0; i < adj.size(); i++) {
        if (adj[i].size() % 2 != 0) {
            return {};
        }
    }
    int start = -1;
    for (int i = 0; i < adj.size(); i++) {
        if (!adj[i].empty()) {
            start = i;
            break;
        }
    }
    if (start == -1) return {};

    vector<int> circuit = buildCycle(adj, start);

    bool moreEdges = true;
    while (moreEdges) {
        moreEdges = false;
        for (int i = 0; i < circuit.size(); i++) {
            int u = circuit[i];
            if (!adj[u].empty()) {
                vector<int> extraCycle = buildCycle(adj, u);
                circuit.insert(circuit.begin() + i + 1,
extraCycle.begin(), extraCycle.end() - 1);
                moreEdges = true;
                break;
            }
        }
    }
    return circuit;
}

int main() {
    vector<vector<int>> adj = {
        {1,3,4,5},
        {0,2,3,5},
        {1,3,4},
        {0,1,2,4},
        {0,5,3,2},
        {0,1,4}
    };

    vector<int> circuit = getEulerianCircuit(adj);

    if (circuit.empty()) {
        cout << "No Eulerian circuit exists." << endl;
    } else {
        cout << "Eulerian circuit: ";
        for (int v : circuit) {
            cout << v << " ";
        }
        cout << endl;
    }
    return 0;
}
```



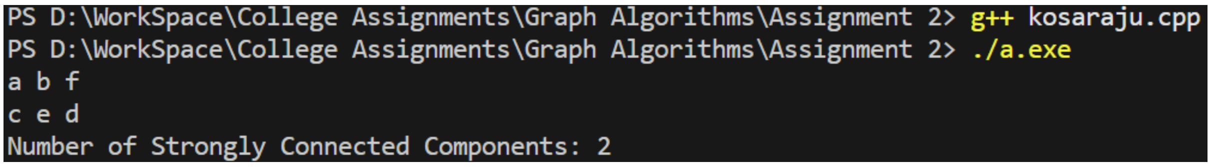
```
PS D:\WorkSpace\College Assignments\Graph Algorithms\Assignment 2> g++ .\splicing.cpp
PS D:\WorkSpace\College Assignments\Graph Algorithms\Assignment 2> ./a.exe
No Eulerian circuit exists.
```

11. Write a C/C++/Java program to implement Kosaraju's algorithm for finding the strongly connected components in the following directed graph.



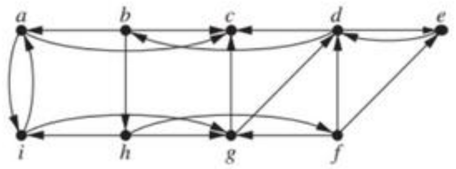
```
# include <bits/stdc++.h>
using namespace std;
void dfs(int node,vector<int>&vis,vector<int>adj[],stack<int>&st){
    vis[node] = 1;
    for(auto it : adj[node]){
        if(!vis[it]){
            dfs(it,vis,adj,st);
        }
    }
    st.push(node);
}
void dfsOnTranspose(int node,vector<int>&vis,vector<int>
adjT[],vector<char>&components){
    vis[node] = 1;
    components.push_back('a' + node);
    for(auto it : adjT[node]){
        if(!vis[it]){
            dfsOnTranspose(it,vis,adjT,components);
        }
    }
}
int main(){
```

```cpp
    // 0 -> a, 1 -> b, 2 -> c, 3 -> d, 4 -> e, 5 -> f
    vector<int>adj[6];
    adj[0] = {1,5};
    adj[1] = {0,2,4};
    adj[2] = {4,3};
    adj[3] = {4};
    adj[4] = {2};
    adj[5] = {4, 0};
    vector<int>vis(6,0);
    stack<int>st;
    //Step 1 : Ordering based on finishing time
    dfs(0,vis,adj,st);
    //Step 2 : Transpose the graph
    vector<int>adjT[6];
    for(int i=0;i<6;i++){
        vis[i] = 0;
        for(auto it : adj[i]){
            adjT[it].push_back(i);
        }
    }
    //Step 3 : Do DFS based on finishing time
    int sccs = 0;
    vector<string> components;
    while(!st.empty()){
        int node = st.top();
        st.pop();
        if(!vis[node]){
            vector<char>components;
            dfsOnTranspose(node,vis,adjT,components);
            sccs++;
            for(auto it : components){
                cout << it << " ";
            }
            cout << endl;
        }
    }
    cout << "Number of Strongly Connected Components: " << sccs <<
endl;
    return 0;
}
```

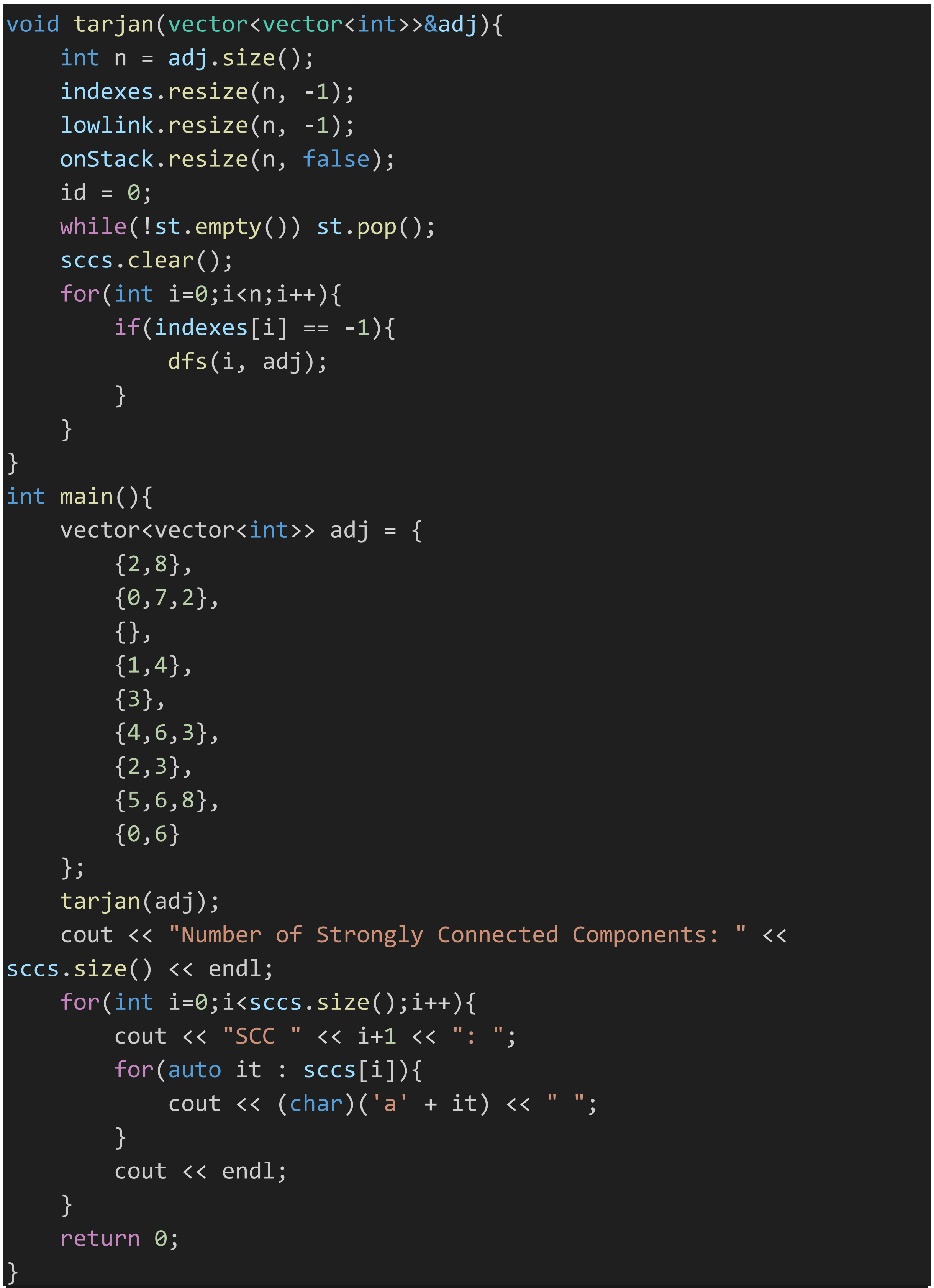


```
PS D:\WorkSpace\College Assignments\Graph Algorithms\Assignment 2> g++ kosaraju.cpp
PS D:\WorkSpace\College Assignments\Graph Algorithms\Assignment 2> ./a.exe
a b f
c e d
Number of Strongly Connected Components: 2
```

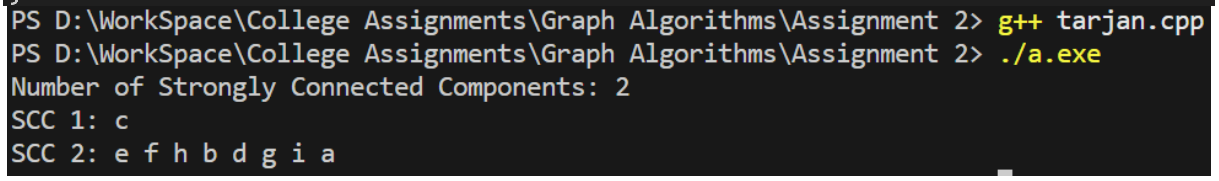
12. Write a C/C++/Java program to implement Tarjan's algorithm for finding the strongly connected components in the following directed graph.



```cpp
#include <bits/stdc++.h>
using namespace std;
int id = 0;
vector<int> indexes, lowlink;
stack<int> st;
vector<bool> onStack;
vector<vector<int>> sccs;
void dfs(int node, vector<vector<int>>& adj){
    indexes[node] = lowlink[node] = id++;
    st.push(node);
    onStack[node] = true;
    for(auto it : adj[node]){
        if(indexes[it] == -1){
            dfs(it, adj);
            lowlink[node] = min(lowlink[node], lowlink[it]);
        } else if(onStack[it]){
            lowlink[node] = min(lowlink[node], indexes[it]);
        }
    }
    if(lowlink[node] == indexes[node]){
        vector<int> component;
        while(true){
            int curr = st.top();
            st.pop();
            onStack[curr] = false;
            component.push_back(curr);
            if(curr == node) break;
        }
        sccs.push_back(component);
    }
}
```



```cpp
void tarjan(vector<vector<int>>&adj){
    int n = adj.size();
    indexes.resize(n, -1);
    lowlink.resize(n, -1);
    onStack.resize(n, false);
    id = 0;
    while(!st.empty()) st.pop();
    sccs.clear();
    for(int i=0;i<n;i++){
        if(indexes[i] == -1){
            dfs(i, adj);
        }
    }
}
int main(){
    vector<vector<int>> adj = {
        {2,8},
        {0,7,2},
        {},
        {1,4},
        {3},
        {4,6,3},
        {2,3},
        {5,6,8},
        {0,6}
    };
    tarjan(adj);
    cout << "Number of Strongly Connected Components: " <<
sccs.size() << endl;
    for(int i=0;i<sccs.size();i++){
        cout << "SCC " << i+1 << ": ";
        for(auto it : sccs[i]){
            cout << (char)('a' + it) << " ";
        }
        cout << endl;
    }
    return 0;
}
```



```
PS D:\WorkSpace\College Assignments\Graph Algorithms\Assignment 2> g++ tarjan.cpp
PS D:\WorkSpace\College Assignments\Graph Algorithms\Assignment 2> ./a.exe
Number of Strongly Connected Components: 2
SCC 1: c
SCC 2: e f h b d g i a
```