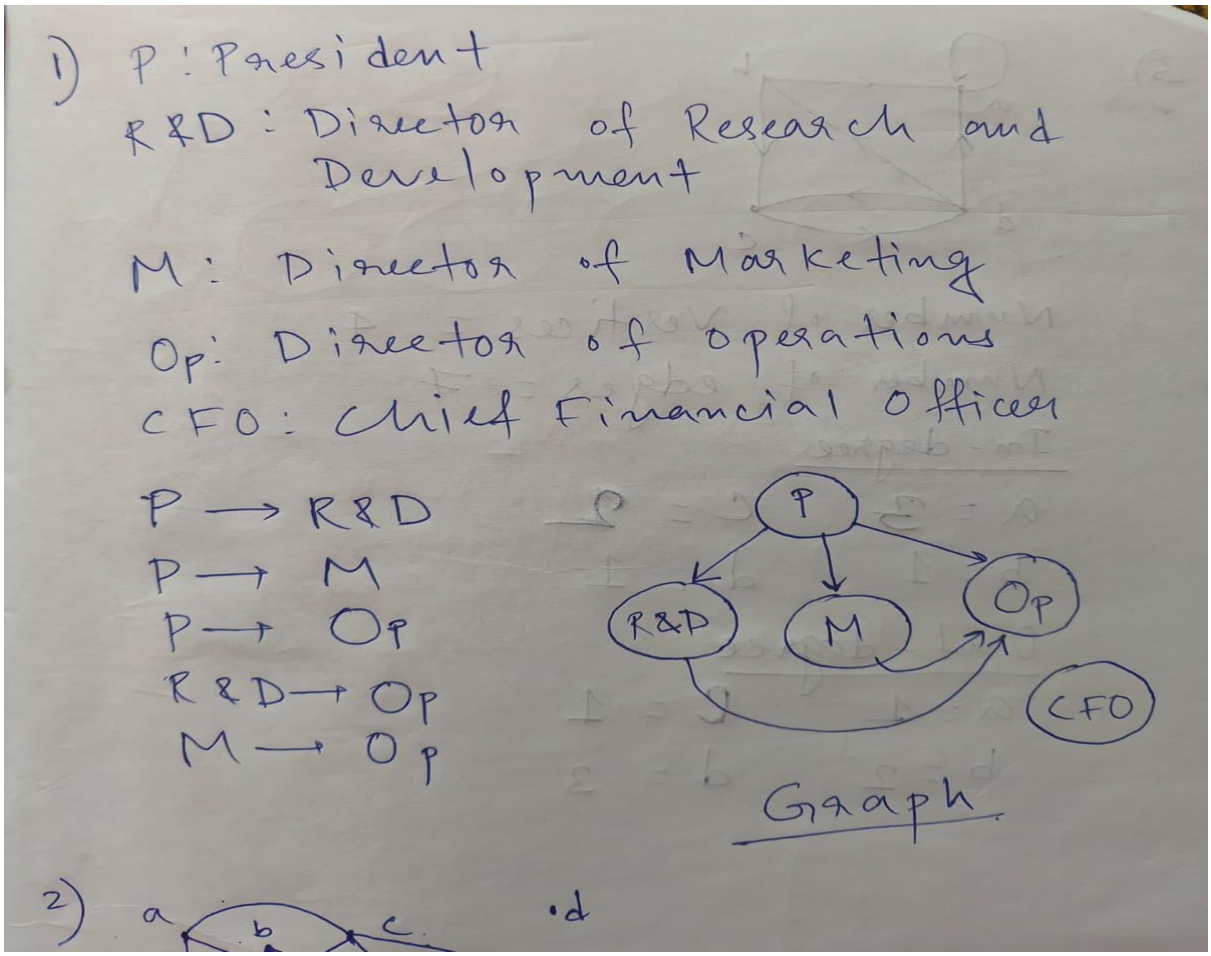
Assignment Sheet – 1

Name – Sayak Sen

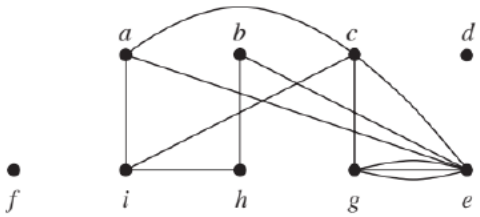
Enrollment Number – 2023CSB047

1. Construct an influence graph for the board members of acompany if the President can influence the Director of Research and Development, the Director of Marketing, andthe Director of Operations; the Director of Research andDevelopment can influence the Director of Operations;the Director of Marketing can influence the Director ofOperations; and no one can influence, or be influencedby, the Chief Financial Officer.

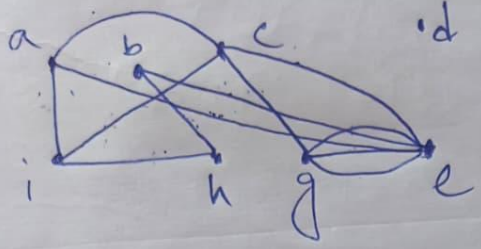
Ans



2. For the given graph G, find the number of vertices, the number ofedges, and the degree of each vertex in the given undirectedgraph. Also, identify all isolated and pendant vertices.



2)



No. of vertices = 9

No. of edges = ~~10~~ 12

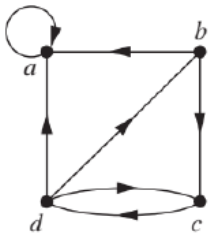
Isolated Vertex = {f, d}

Pendant Vertex = { }

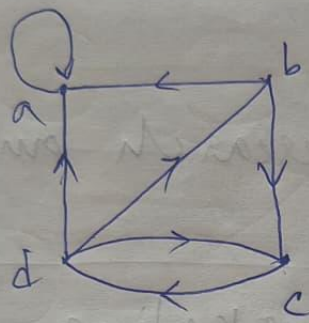
Degree of each vertex:-

a = 3	d = 0	g = 4
b = 2	e = 6	h = 2
c = 4	f = 0	i = 3

3. Determine the number of vertices and edgesand find the in-degree and out-degree of each vertex for thegiven directed multigraph.



3)



Number of Vertices = 4

Number of edges = 7

In-degree

a = 3 c = 2

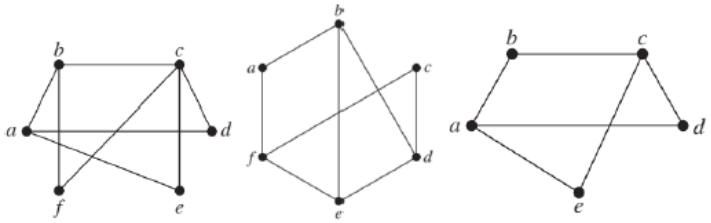
b = 1 d = 1

Out-degree

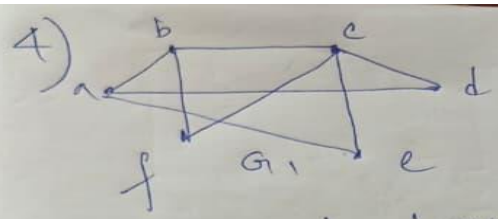
a = 1 c = 1

b = 2 d = 3

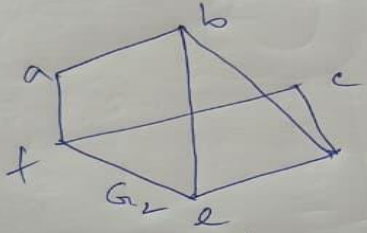
4. Determine whether the following graphs are bipartite. Write a function in C/C++/Java to automate the process.



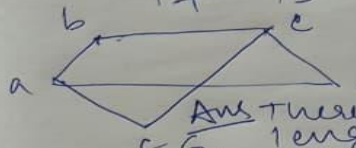
4)



Ans The graph has a odd length cycle which makes the graph coloured using two colours impossible.
∴ The vertex set cannot be divided into two disjoint subsets thus it is not bipartite.

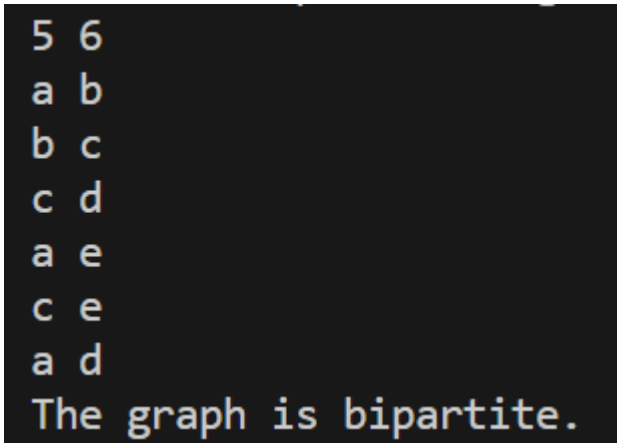


Ans The graph has a odd length cycle which makes the graph coloured using two colours impossible.
∴ The vertex set cannot be divided into ~~two~~ disjoint subsets thus it is not bipartite.

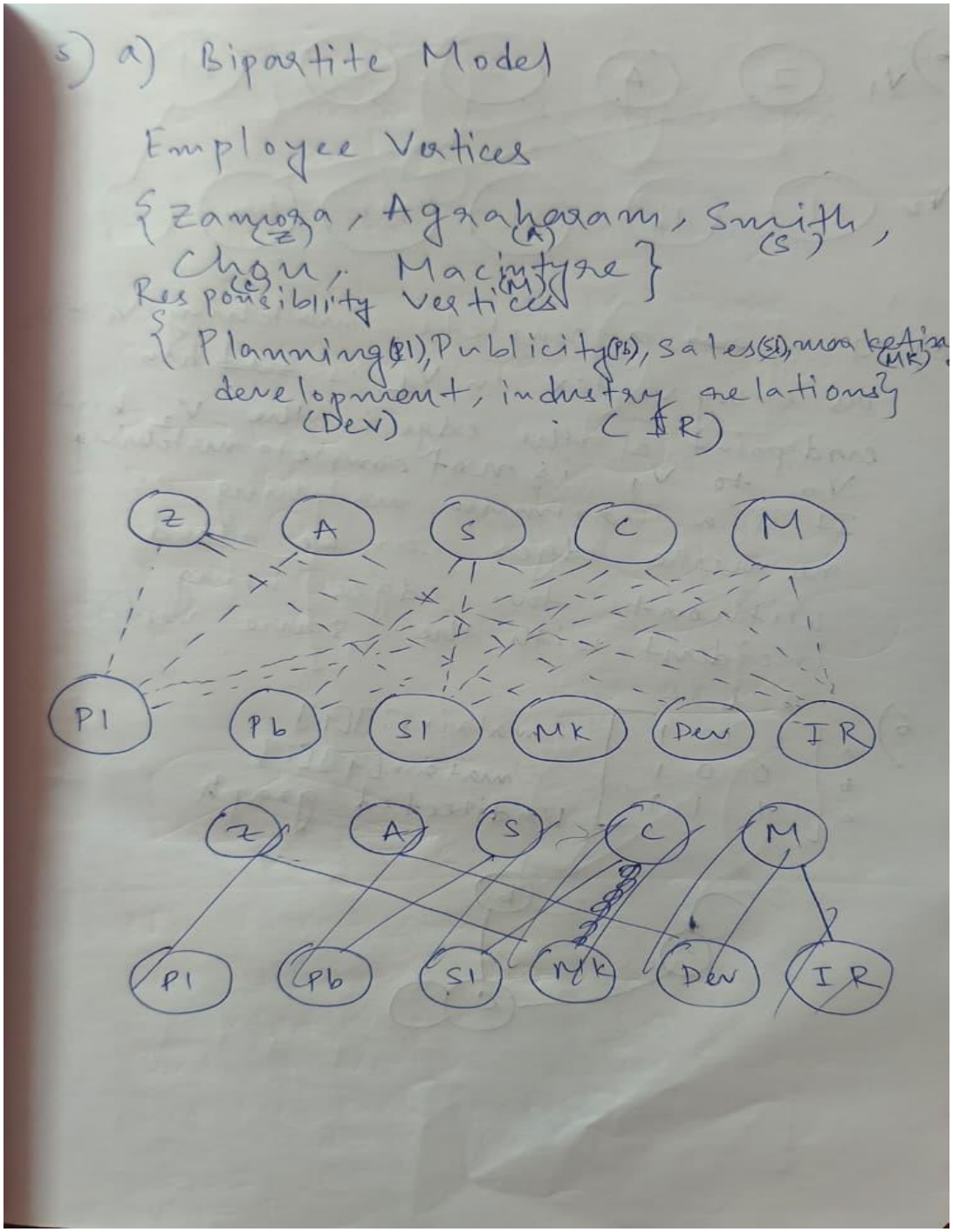


Ans ~~There~~ does not exist odd length cycle.
$V_1 = \{a, c\}$, $V_2 = \{b, d, e\}$
(V_1, V_2) is a bipartition of bipartite graph G_3.

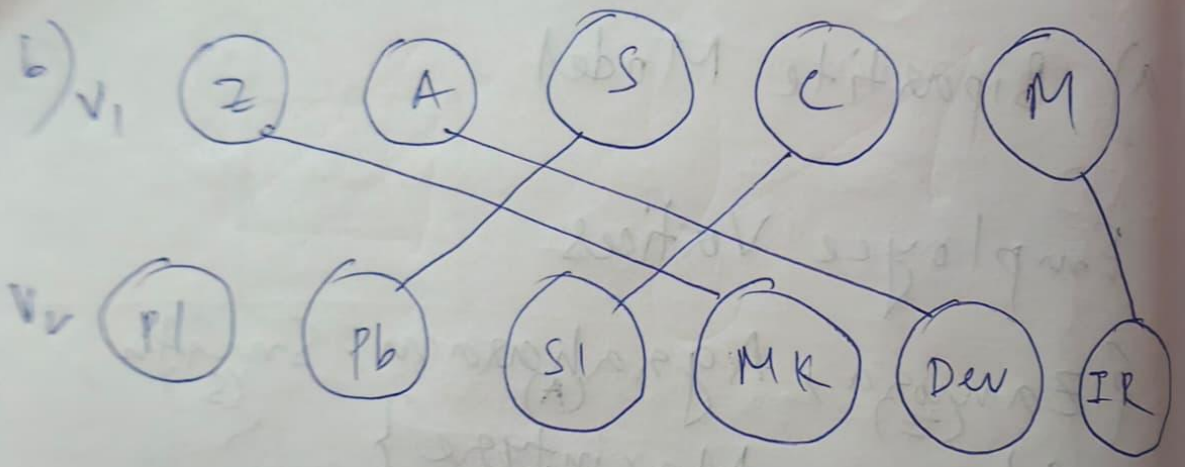
```cpp
#include<bits/stdc++.h>
using namespace std;
bool dfs(int node,int col,vector<int>&color,vector<vector<int>>&
graph){
    color[node] = col;
    for(auto it : graph[node]){
        if(color[it]==-1){
            if(dfs(it,!col,color,graph)==false) return false;

        }
        else if(color[it]==col) return false;
    }
    return true;
}
bool isBipartite(vector<vector<int>>& graph) {
    int n = graph.size();
    vector<int>color(n,-1);
    for(int i=0;i<n;i++){
        if(color[i]==-1){
            if(dfs(i,0,color,graph)==false) return false;
        }
    }
    return true;
}
int main(){
    int n, m;
    cin >> n >> m;
    vector<vector<int>> graph(n);
    for (int i = 0; i < m; i++) {
        char u, v;
        cin >> u >> v;
        graph[u - 'a'].push_back(v - 'a');
        graph[v - 'a'].push_back(u - 'a');
    }
    if (isBipartite(graph)) {
        cout << "The graph is bipartite." << endl;
    } else {
        cout << "The graph is not bipartite." << endl;
    }
    return 0;
}
```


5 6
a b
b c
c d
a e
c e
a d
The graph is bipartite.

5. Suppose that a new company has five employees: Zamora,Agraharam, Smith, Chou, and Macintyre. Each employeewill assume one of six responsibilities: planning, publicity, sales, marketing, development, and industry relations.Each employee is capable of doing one or more of thesejobs: Zamora could do planning, sales, marketing, or industry relations; Agraharam could do planning or development; Smith could do publicity, sales, or industry relations; Chou could do planning, sales, or industry relations; and Macintyre could do planning, publicity, sales,or industry relations.
   a. Model the capabilities of these employees using a bipartite graph.
   b. Find an assignment of responsibilities such that eachemployee is assigned one responsibility.
   c. Is the matching of responsibilities you found in part (b) a complete matching? Is it a maximum matching?



b)

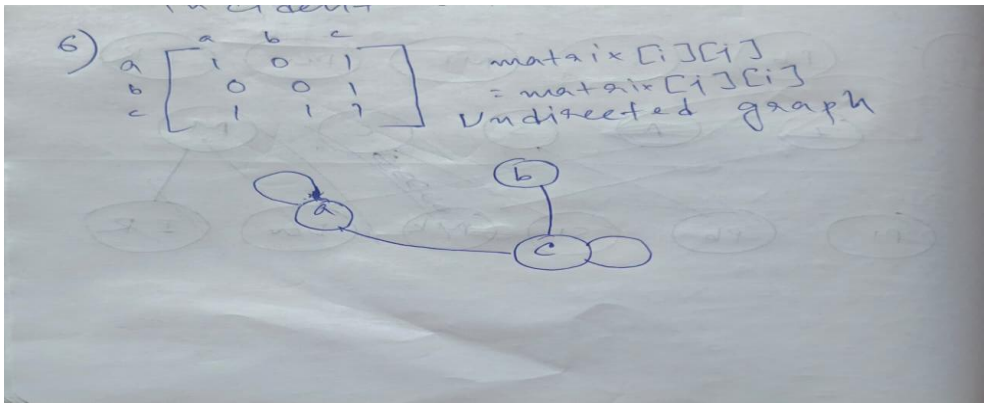


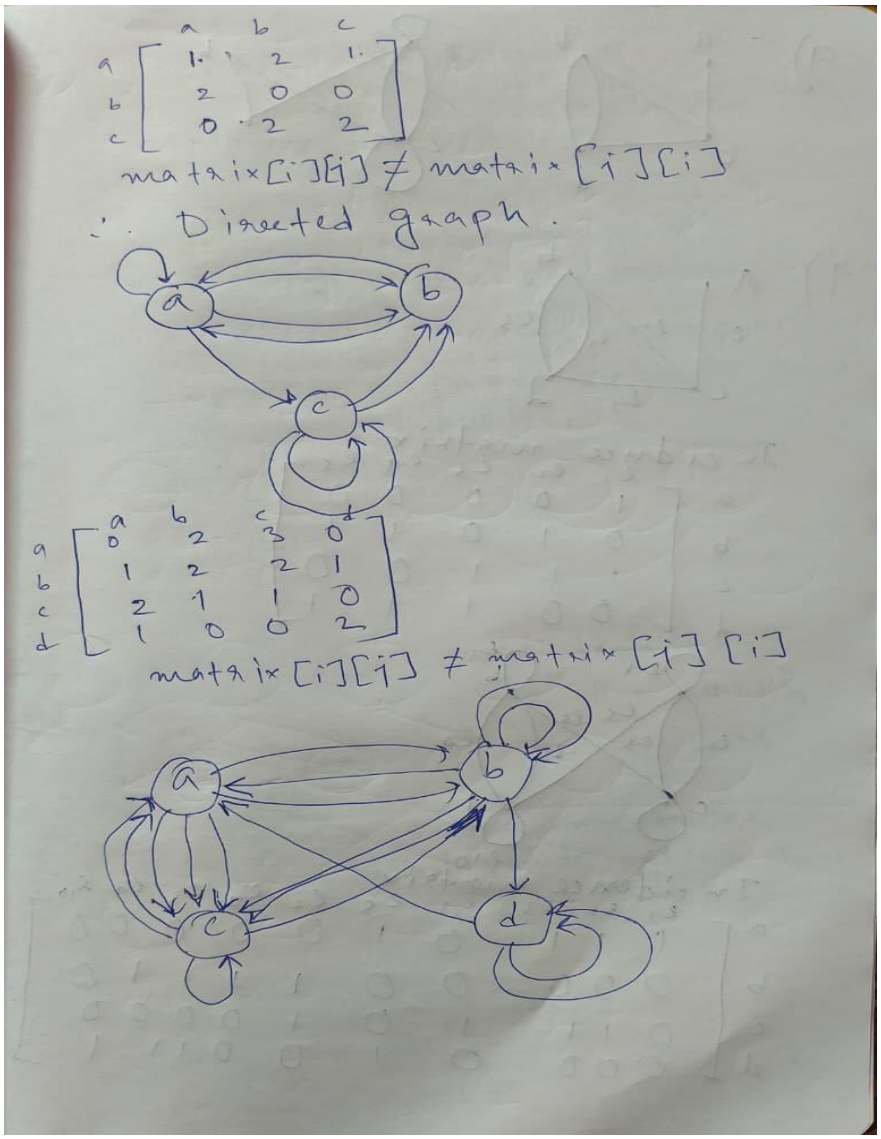
c) It is ~~not~~ a complete matching V_1 t as every vertex in V_1 is ~~not~~ han endpoint of the edge in the V_2 b V_2 to V_1 is not complete matchin It is a maximum matching as no more edges can be added without two edges being incident with the same vertex

6) a b c

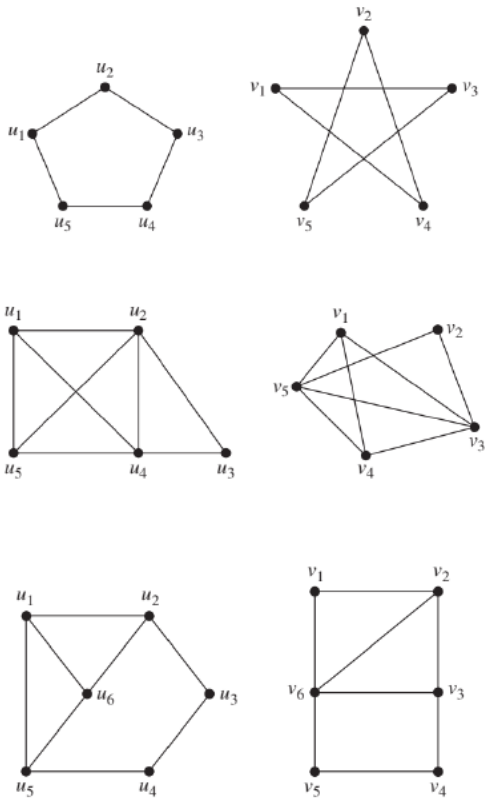
6. Draw the graphs represented by the following adjacency matrices.

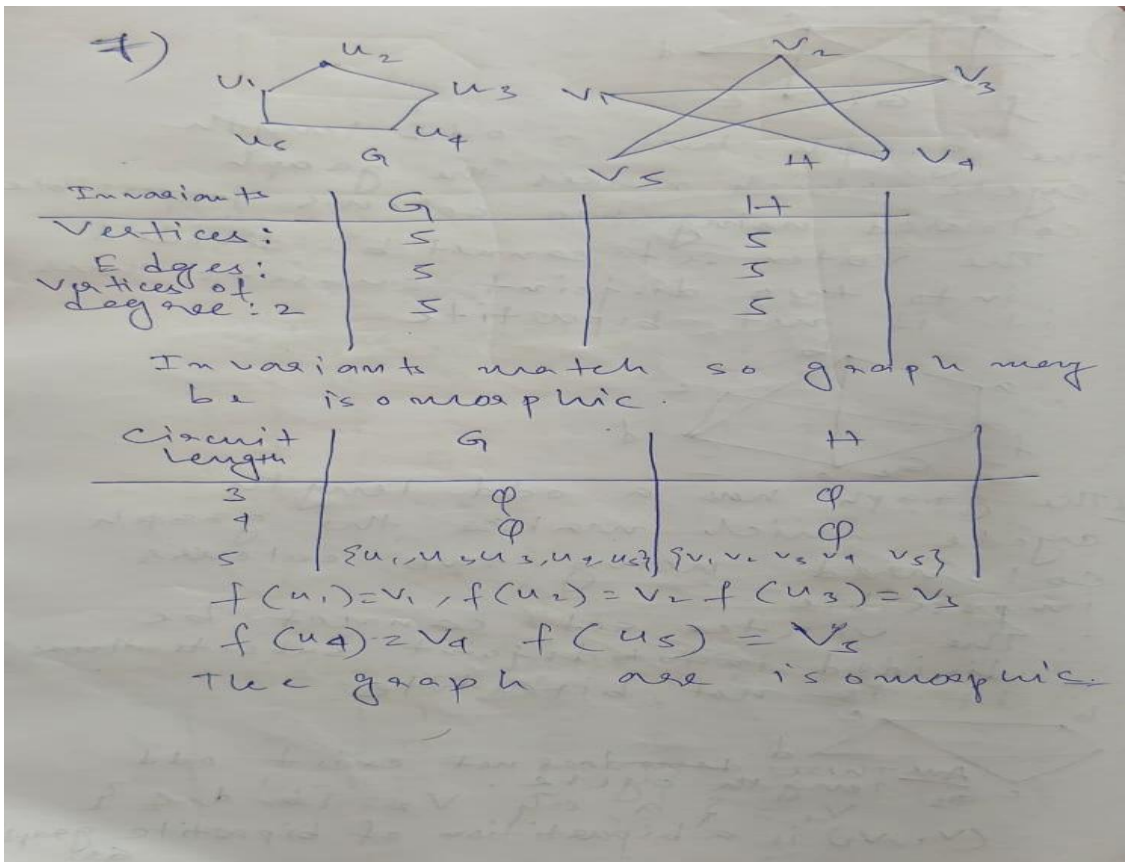
$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & 0 \\ 0 & 2 & 2 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 & 3 & 0 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 0 \\ 1 & 0 & 0 & 2 \end{bmatrix}$$

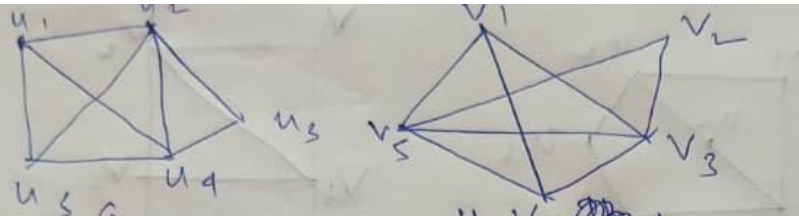




7. Determine whether the given pairs of graphsare isomorphic. Exhibit an isomorphism or provide a rigorousargument that none exists.







Invariants	G	H
Vertices	5	5
edges	8	8
Vertices of degree 2	1	1
3	2	2
4	2	2

Invariants match graphs may be isomorphic.

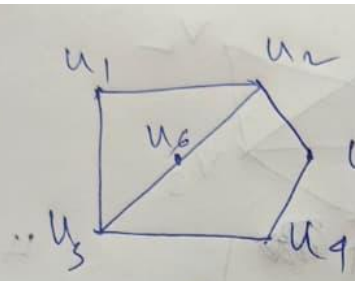
Subgraph matching	G	H
Degree 2	u_3	v_2
Degree 3	u_1, u_5	v_1, v_4
Degree 4	u_2, u_4	v_5, v_3

Subgraphs are isomorphic

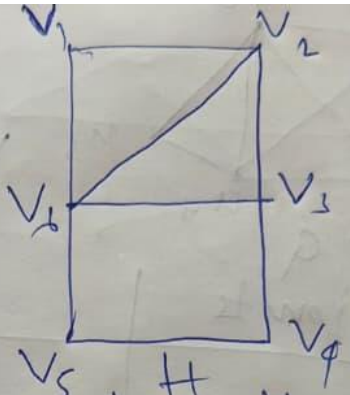
∴ G and H are also isomorphic

$f(u_3) = v_2$, $f(u_1) = v_1$ $f(u_5) = v_4$

$f(u_2) = v_5$ $f(u_4) = v_3$



G



H

Invariant ⇄	G	H →
Vertices	6	6
edges	8	8
No. of vertices degree 2	2	3

The number of vertices with degree 2 are unequal for G and H.

∴ Graphs are not isomorphic.

8. Write a C/C++/Jave function to automate the graph isomorphism test between two input graphs. Your program should check for the three invariants first. If any invariant does not match, return "Input graphs are not isomorphic." Otherwise, follow the subgraph matching (formed by vertices of similar degree) approach to determine if the graphs are isomorphic.

```
#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <algorithm>
using namespace std;

class GraphIsomorphism {
public:
    static bool areIsomorphic(const vector<vector<int>>& g1, const
vector<vector<int>>& g2) {
        int n1 = g1.size();
        int n2 = g2.size();

        if (n1 != n2) {
            return false;
        }

        int e1 = countEdges(g1);
        int e2 = countEdges(g2);
        if (e1 != e2) {
            return false;
        }

        vector<int> deg1 = getDegreeSequence(g1);
        vector<int> deg2 = getDegreeSequence(g2);
        sort(deg1.begin(), deg1.end());
        sort(deg2.begin(), deg2.end());
        if (deg1 != deg2) {
            return false;
        }

        map<int, vector<int>> degreeGroups1 = groupByDegree(g1);
        map<int, vector<int>> degreeGroups2 = groupByDegree(g2);

        map<int, int> mapping;
```

```
        set<int> used;
        return checkMapping(g1, g2, degreeGroups1, degreeGroups2,
mapping, used);
    }

private:
    static int countEdges(const vector<vector<int>>& graph) {
        int sum = 0;
        for (const auto& adj : graph) {
            sum += adj.size();
        }
        return sum / 2; // each edge counted twice
    }

    static vector<int> getDegreeSequence(const vector<vector<int>>&
graph) {
        vector<int> deg(graph.size());
        for (int i = 0; i < graph.size(); i++) {
            deg[i] = graph[i].size();
        }
        return deg;
    }

    static map<int, vector<int>> groupByDegree(const
vector<vector<int>>& graph) {
        map<int, vector<int>> degreeMap;
        for (int i = 0; i < graph.size(); i++) {
            int degree = graph[i].size();
            degreeMap[degree].push_back(i);
        }
        return degreeMap;
    }

    static bool checkMapping(const vector<vector<int>>& g1, const
vector<vector<int>>& g2,
            const map<int, vector<int>>& degGroups1, const map<int,
vector<int>>& degGroups2,
            map<int, int>& mapping, set<int>& used) {
        if (mapping.size() == g1.size()) {
            return checkAdjacency(g1, g2, mapping);
        }
```

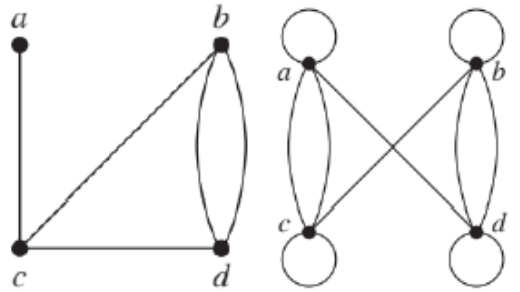
```
        for (const auto& pair : degGroups1) {
            int degree = pair.first;
            const vector<int>& vertices1 = pair.second;
            const vector<int>& vertices2 = degGroups2.at(degree);

            for (int v1 : vertices1) {
                if (mapping.find(v1) != mapping.end()) continue;

                for (int v2 : vertices2) {
                    if (used.find(v2) != used.end()) continue;

                    mapping[v1] = v2;
                    used.insert(v2);

                    if (checkMapping(g1, g2, degGroups1, degGroups2,
mapping, used)) {
                        return true;
                    }

                    mapping.erase(v1);
                    used.erase(v2);
                }
                return false;
            }
        }
        return false;
    }

    static bool checkAdjacency(const vector<vector<int>>& g1,
            const vector<vector<int>>& g2, const map<int, int>&
mapping) {
        for (int u1 = 0; u1 < g1.size(); u1++) {
            int u2 = mapping.at(u1);
            for (int v1 : g1[u1]) {
                int v2 = mapping.at(v1);
                // Check if v2 is in the adjacency list of u2
                if (find(g2[u2].begin(), g2[u2].end(), v2) ==
g2[u2].end()) {
                    return false;
                }
            }
        }
```

```
        return true;
    }
};

int main() {
    // First graph
    cout << "Enter number of vertices in Graph 1: ";
    int V1;
    cin >> V1;
    cout << "Enter number of edges in Graph 1: ";
    int E1;
    cin >> E1;

    vector<vector<int>> g1(V1);

    cout << "Enter edges (u v) for Graph 1:" << endl;
    for (int i = 0; i < E1; i++) {
        int u, v;
        cin >> u >> v;
        u--; v--;
        g1[u].push_back(v);
        g1[v].push_back(u);
    }

    // Second graph
    cout << "Enter number of vertices in Graph 2: ";
    int V2;
    cin >> V2;
    cout << "Enter number of edges in Graph 2: ";
    int E2;
    cin >> E2;

    vector<vector<int>> g2(V2);

    cout << "Enter edges (u v) for Graph 2:" << endl;
    for (int i = 0; i < E2; i++) {
        int u, v;
        cin >> u >> v;
        u--; v--;
        g2[u].push_back(v);
        g2[v].push_back(u);
    }
```

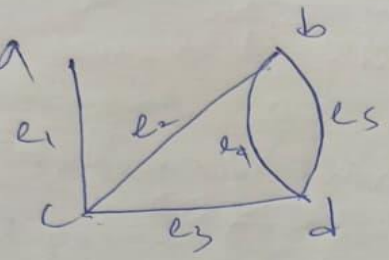
```
    bool result = GraphIsomorphism::areIsomorphic(g1, g2);
    if (result) {
        cout << "Graphs are isomorphic." << endl;
    } else {
        cout << "Graphs are not isomorphic." << endl;
    }

    return 0;
}
```

```
Enter number of vertices in Graph 1: 5
Enter number of edges in Graph 1: 5
Enter edges (u v) for Graph 1:
1 2
2 3
3 4
4 5
5 1
Enter number of vertices in Graph 2: 5 5
Enter number of edges in Graph 2: Enter edges (u v) for Graph 2:
1 3
1 4
2 5
2 4
3 5
Graphs are isomorphic.
```

9. Write incidence matrices to represent the following graphs.

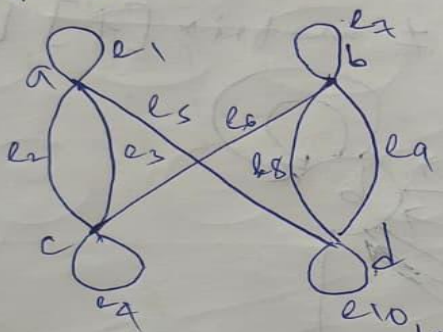


9)



Incidence matrix

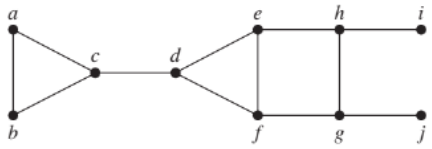
	e_1	e_2	e_3	e_4	e_5
a	1	0	0	0	0
b	0	1	0	1	1
c	1	1	1	0	0
d	0	0	1	1	1



Incidence matrix

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
a	1	1	1	0	1	0	0	0	0	0
b	0	0	0	0	0	1	1	1	1	0
c	0	1	1	1	0	1	0	0	0	0
d	0	0	0	0	1	0	0	1	1	1

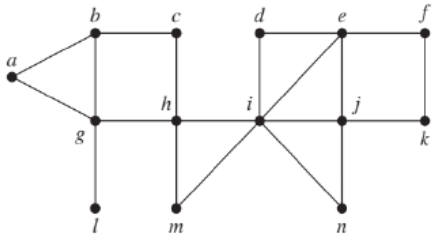
10. Write a C/C++/Java function to implement the depth-first search (DFS) to produce a spanning tree for the given simple graph. Choose *a* as the root ofthis spanning tree and assume that the vertices are orderedalphabetically.




```cpp
#include<bits/stdc++.h>
using namespace std;
void dfs(int node, vector<int>& visited, vector<vector<int>>& graph)
{
    visited[node] = 1;
    for (auto it : graph[node]) {
        if (!visited[it]) {
            cout<<char(it+'a')<<" -- "<<char(node+'a')<<endl;
            dfs(it, visited, graph);
        }
    }
}
int main(){
    int n, m;
    cout << "Enter the number of nodes and edges: ";
    cin >> n >> m;
    vector<vector<int>> graph(n);
    cout << "Enter the edges (u v) :" << endl;
    for (int i = 0; i < m; i++) {
        char u, v;
        cin >> u >> v;
        graph[u - 'a'].push_back(v - 'a');
        graph[v - 'a'].push_back(u - 'a');
    }

    vector<int> visited(n, 0);
    cout << "Edges in the Spanning Tree through DFS :" << endl;
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i, visited, graph);
        }
    }
    return 0;
}
```

```
Enter the number of nodes and edges: 10 12
Enter the edges (u v) :
a b
a c
b c
c d
d e
d f
e f
e h
f g
h g
h i
g j
Edges in the Spanning Tree through DFS :
b -- a
c -- b
d -- c
e -- d
f -- e
g -- f
h -- g
i -- h
j -- g
```

11. Write a C/C++/Java function to implement the breadth-first search (BFS) to produce a spanning tree for the given simple graph. Choose *a* as the root of this spanning tree and assume that the vertices are ordered alphabetically.




```cpp
#include<bits/stdc++.h>
using namespace std;
void bfs(int start, vector<int>& visited, vector<vector<int>>&
graph) {
    queue<int> q;
    q.push(start);
    visited[start] = 1;

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        for (auto it : graph[node]) {
            if (!visited[it]) {
                cout << char(it + 'a') << " -- " << char(node + 'a')
<< endl;
                visited[it] = 1;
                q.push(it);
            }
        }
    }
}
int main(){
    int n, m;
    cout << "Enter the number of nodes and edges: ";
    cin >> n >> m;
    vector<vector<int>> graph(n);
    cout << "Enter the edges (u v) :" << endl;
    for (int i = 0; i < m; i++) {
        char u, v;
        cin >> u >> v;
        graph[u - 'a'].push_back(v - 'a');
        graph[v - 'a'].push_back(u - 'a');
    }
```

```
    vector<int> visited(n, 0);
    cout << "Edges in the Spanning Tree through BFS :" << endl;
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            bfs(i, visited, graph);
        }
    }
    return 0;
}
```

```
Enter the number of nodes and edges: 14 20
Enter the edges (u v) :
a b
a g
b g
b c
c h
g h
g l
h m
h i
m i
d i
i e
d e
i j
e j
e f
f k
j k
i n
j n
```

```
Edges in the Spanning Tree through BFS :
b -- a
g -- a
c -- b
h -- g
l -- g
m -- h
i -- h
d -- i
e -- i
j -- i
n -- i
f -- e
k -- j
```