**DAA [Day - 4]**

UID: 24MCI10204

Name: Rahul Saxena

Branch: 24MCA – AI & ML

**Question 1:** Imagine a building or street network represented as a graph:

• Nodes (vertices): Intersections, rooms, or key areas.

• Edges: Paths, corridors, or streets connecting the nodes.

Install the minimum number of surveillance cameras such that every connection (edge) is monitored — meaning at least one of its endpoints (nodes) has a camera.

**Answer:**

import java.util.\*;

public class MinimumVertexCover {

static Map<String, List<String>> graph = new HashMap<>();

static Set<String> visitedEdges = new HashSet<>();

static Set<String> vertexCover = new HashSet<>();

public static void addEdge(String u, String v) {

graph.computeIfAbsent(u, k -> new ArrayList<>()).add(v);

graph.computeIfAbsent(v, k -> new ArrayList<>()).add(u);

}

public static void findVertexCover() {

Set<String> covered = new HashSet<>();

for (String u : graph.keySet()) {

for (String v : graph.get(u)) {

String edge = u + "-" + v;

String reverseEdge = v + "-" + u;

if (!visitedEdges.contains(edge) && !visitedEdges.contains(reverseEdge)) {

vertexCover.add(u);

vertexCover.add(v);

for (String adj : graph.get(u)) {

visitedEdges.add(u + "-" + adj);

}

for (String adj : graph.get(v)) {

visitedEdges.add(v + "-" + adj);

}

}

}

}

}

public static void main(String[] args) {

addEdge("A", "B");

addEdge("A", "C");

addEdge("B", "D");

addEdge("C", "D");

addEdge("D", "E");

findVertexCover();

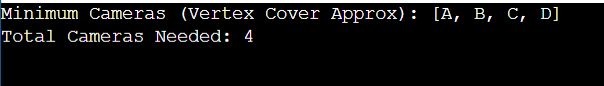
System.out.println("Minimum Cameras (Vertex Cover Approx): " + vertexCover);

System.out.println("Total Cameras Needed: " + vertexCover.size());

}

}

**Output:**

****

**Question 2**: Verify whether a feasible exam timetable exists such that no student has overlapping exams.

**Answer:**

import java.util.\*;

public class ExamTimetable {

static boolean isSafe(int v, int[][] graph, int[] color, int c) {

for (int i = 0; i < graph.length; i++)

if (graph[v][i] == 1 && color[i] == c)

return false;

return true;

}

static boolean graphColoring(int[][] graph, int m, int[] color, int v) {

if (v == graph.length)

return true;

for (int c = 1; c <= m; c++) {

if (isSafe(v, graph, color, c)) {

color[v] = c;

if (graphColoring(graph, m, color, v + 1))

return true;

color[v] = 0;

}

}

return false;

}

public static void main(String[] args) {

// Graph based on conflict: A-B, B-C, A-C

int[][] conflictGraph = {

{0, 1, 1}, // A

{1, 0, 1}, // B

{1, 1, 0} // C

};

int numberOfSlots = 3; // try with 2 and then 3

int[] color = new int[conflictGraph.length];

if (graphColoring(conflictGraph, numberOfSlots, color, 0)) {

System.out.println("Feasible timetable exists using " + numberOfSlots + " slots.");

System.out.println("Exam Time Assignments:");

for (int i = 0; i < color.length; i++)

System.out.println("Exam " + (char)('A' + i) + " → Slot " + color[i]);

} else {

System.out.println(" No feasible timetable with " + numberOfSlots + " slots.");

}

}

}

**Output:**

