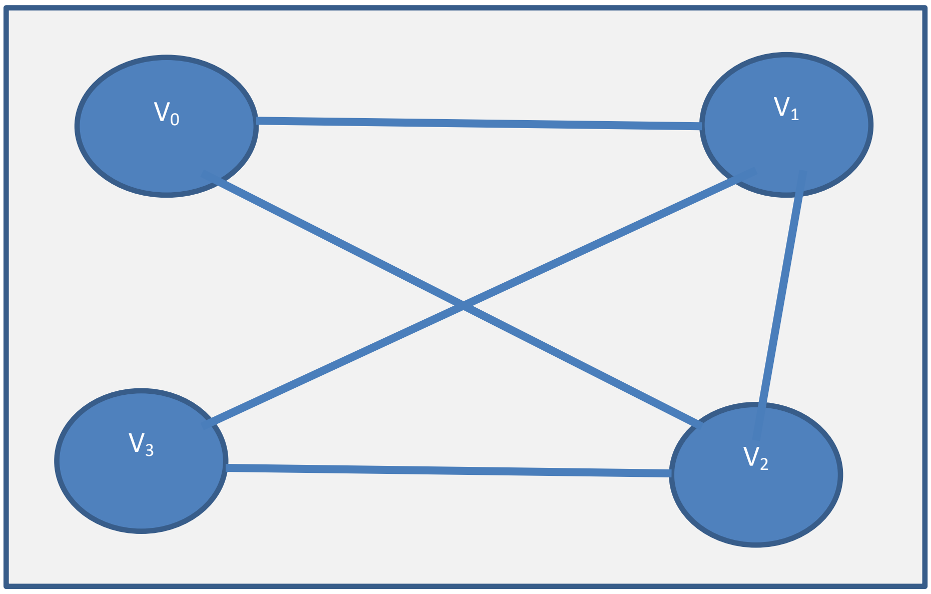
Practical 1 (Class 1) -

1. PROGRAM FOR ADJACENCY MATRIX REPRESENTATION OF AN UNDIRECTED GRAPH USING JAVA.
2. PROGRAM FOR ADJACENCY LIST REPRESENTATION OF A GRAPH USING JAVA.
3. PROGRAM FOR DFS (USING STACK) OF A GRAPH USING JAVA.
4. PROGRAM FOR DFS (RECURSIVE) OF A GRAPH USING JAVA.
5. PROGRAM FOR BFS (USING QUEUE) OF A GRAPH USING JAVA.

Q. PROGRAM FOR ADJACENCY MATRIX REPRESENTATION OF AN UNDIRECTED GRAPH USING JAVA.

```java
import java.util.Scanner;
public class AdjacencyMatrix {
    private int[][] adjacencyMatrix;
    private int numberOfVertices;

    // Constructor to initialize the graph
    public AdjacencyMatrix(int numberOfVertices) {
        this.numberOfVertices = numberOfVertices;
        adjacencyMatrix = new int[numberOfVertices][numberOfVertices];
    }

    // Method to add an edge
    public void addEdge(int source, int destination) {
        if (source < 0 || source >= numberOfVertices ||
                destination < 0 || destination >= numberOfVertices) {
            System.out.println("Invalid edge!");
        } else {
            adjacencyMatrix[source][destination] = 1;
            adjacencyMatrix[destination][source] = 1; //undirected graph
        }
    }

    // Method to display the adjacency matrix
    public void displayMatrix() {
        System.out.println("Adjacency Matrix:");
        for (int i = 0; i < numberOfVertices; i++) {
            for (int j = 0; j < numberOfVertices; j++) {
                System.out.print(adjacencyMatrix[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of vertices:");
        int vertices = scanner.nextInt();

        AdjacencyMatrix graph = new AdjacencyMatrix(vertices);

        System.out.println("Enter the number of edges:");
        int edges = scanner.nextInt();

        System.out.println("Enter the edges (source and destination):");
        for (int i = 0; i < edges; i++) {
            int source = scanner.nextInt();
            int destination = scanner.nextInt();
            graph.addEdge(source, destination);
        }

        graph.displayMatrix();

        scanner.close();
    }
}
```

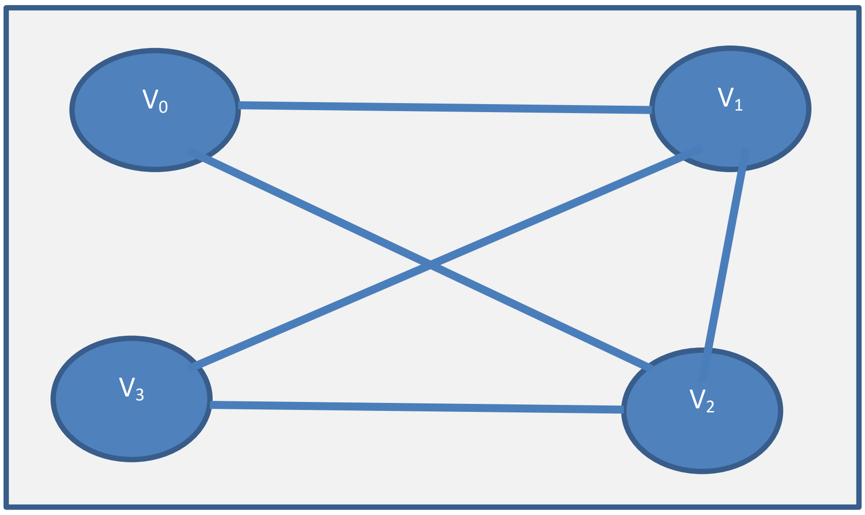
```
Output -
Enter the number of vertices:
4
Enter the number of edges:
5
Enter the edges (source and destination):
0 1
1 2
2 3
0 2
1 3
Adjacency Matrix:
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
```



Q. PROGRAM FOR ADJACENCY LIST REPRESENTATION OF A GRAPH USING JAVA

```java
import java.util.LinkedList;
import java.util.Scanner;

class Graph {
    private int vertices; // Number of vertices
    private LinkedList<Integer>[] adjacencyList; // Adjacency list array

    // Constructor to initialize the graph
    public Graph(int vertices) {
        this.vertices = vertices;
        adjacencyList = new LinkedList[vertices];

        // Initialize each list in the adjacency list array
        for (int i = 0; i < vertices; i++) {
            adjacencyList[i] = new LinkedList<>();
        }
    }

    // Method to add an edge to the graph
    public void addEdge(int source, int destination) {
        adjacencyList[source].add(destination);
        adjacencyList[destination].add(source); // For an undirected
graph
    }

    // Method to display the adjacency list representation of the graph
    public void displayGraph() {
        for (int i = 0; i < vertices; i++) {
            System.out.print("Vertex " + i + ": ");
            for (Integer neighbor : adjacencyList[i]) {
                System.out.print(neighbor + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get the number of vertices and edges
        System.out.print("Enter the number of vertices: ");
        int vertices = scanner.nextInt();
        System.out.print("Enter the number of edges: ");
        int edges = scanner.nextInt();

        Graph graph = new Graph(vertices);
        // Get the edges from the user
        System.out.println("Enter the edges (source and destination): ");
        for (int i = 0; i < edges; i++) {
            int source = scanner.nextInt();
            int destination = scanner.nextInt();
            graph.addEdge(source, destination);
        }

        // Display the adjacency list
        System.out.println("\nAdjacency List:");
        graph.displayGraph();

        scanner.close();
    }
}
```

```
Output -
Enter the number of vertices: 4
Enter the number of edges: 5
Enter the edges (source and destination):
0 1
1 2
2 3
0 2
1 3

Adjacency List:
Vertex 0: 1 2
Vertex 1: 0 2 3
Vertex 2: 1 3 0
Vertex 3: 2 1
```

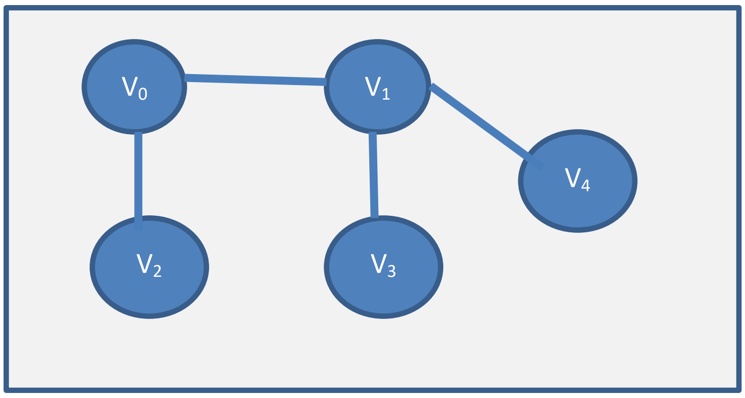


Q. PROGRAM FOR DFS (USING STACK) OF A GRAPH USING JAVA.

```
import java.util.Stack;
class DFS {
    // Perform DFS starting from a given source vertex
    public static void dfs(int startVertex, int[][] adjacencyMatrix) {
         int vertices = adjacencyMatrix.length;
        boolean[] visited = new boolean[vertices]; // Track visited nodes
        Stack<Integer> stack = new Stack<>(); // Use Stack for DFS

        stack.push(startVertex); // Push the starting vertex
        while (!stack.isEmpty()) {
            int currentVertex = stack.pop();

            // If the vertex has not been visited, mark it and process
            if (!visited[currentVertex]) {
                System.out.print(currentVertex + " ");
                visited[currentVertex] = true;

                // Visit all unvisited adjacent vertices
                for (int i = vertices - 1; i >= 0; i--) {
                    if (adjacencyMatrix[currentVertex][i] == 1 &&
                                                    !visited[i])
                   {
                            stack.push(i); // Push adjacent vertex
                    }
                }
            }
        }
    }

    public static void main(String[] args) {
        // Initialize adjacency matrix directly
        int[][] adjacencyMatrix = {
            {0, 1, 1, 0, 0}, // Vertex 0 connected to 1 and 2
            {1, 0, 0, 1, 1}, // Vertex 1 connected to 0, 3, and 4
            {1, 0, 0, 0, 0}, // Vertex 2 connected to 0
            {0, 1, 0, 0, 0}, // Vertex 3 connected to 1
            {0, 1, 0, 0, 0}  // Vertex 4 connected to 1
        };

        System.out.println("DFS starting from vertex 0:");
        dfs(2,adjacencyMatrix);
    }
}
```

Output -

```
DFS starting from
vertex 2:
2 0 1 3 4
```

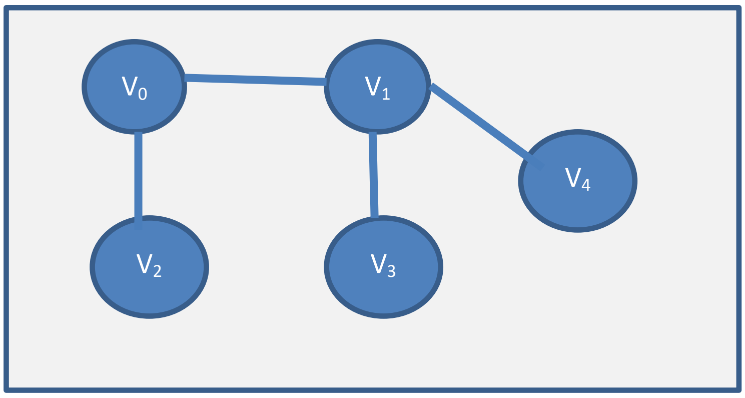


Q. PROGRAM FOR DFS (RECURSIVE) OF A GRAPH USING JAVA.

```java
public class DFSRecursive {

    private static void dfs(int[][] graph, boolean[] visited, int node) {
        // Mark the current node as visited
        visited[node] = true;
        System.out.print(node + " ");

        // Recursively visit all unvisited neighbours
        for (int i = 0; i < graph[node].length; i++) {
            if (graph[node][i] == 1 && !visited[i]) {
                dfs(graph, visited, i);
            }
        }
    }

    public static void main(String[] args) {
        // Hard-coded adjacency matrix
        int[][] graph = {
                {0, 1, 1, 0, 0}, // Vertex 0 connected to 1 and 2
                {1, 0, 0, 1, 1}, // Vertex 1 connected to 0, 3, and 4
                {1, 0, 0, 0, 0}, // Vertex 2 connected to 0
                {0, 1, 0, 0, 0}, // Vertex 3 connected to 1
                {0, 1, 0, 0, 0}  // Vertex 4 connected to 1
        };


        int numberOfNodes = graph.length;
        boolean[] visited = new boolean[numberOfNodes];

        System.out.println("DFS Traversal starting from node 0:");
        dfs(graph, visited, 2); // Start DFS from node 2
    }
}
```

Output -

```
DFS starting from
vertex 2:
2 0 1 3 4
```



Q. PROGRAM FOR BFS (USING QUEUE) OF A GRAPH USING JAVA.

```java
import java.util.LinkedList;
import java.util.Queue;

public class BFS {
    // Method to perform BFS on a graph
    public static void bfs(int[][] adjacencyMatrix, int startNode) {
        int numberOfNodes = adjacencyMatrix.length;
        boolean[] visited = new boolean[numberOfNodes];

        // Queue for BFS
        Queue<Integer> queue = new LinkedList<>();

        // Start with the given node
        visited[startNode] = true;
        queue.add(startNode);
        System.out.println("BFS Traversal: ");
        while (!queue.isEmpty())
        {
            int currentNode = queue.poll();
            System.out.print(currentNode + " ");

            // Traverse neighbors of the current node
            for (int i = 0; i < numberOfNodes; i++) {
             if (adjacencyMatrix[currentNode][i] == 1 && !visited[i])
                {
                    visited[i] = true;
                    queue.add(i);
                }
            }
        }
    }

    public static void main(String[] args) {
        // Example graph as an adjacency matrix
        int[][] adjacencyMatrix = {
            {0, 1, 1, 0},
            {1, 0, 1, 1},
            {1, 1, 0, 1},
            {0, 1, 1, 0}
        };

        // Start BFS from node 0
        bfs(adjacencyMatrix, 3);
    }
}
```

```
Output -
BFS Traversal:
3 1 2 0
```