Data Structure and Algorithms (JAVA)



**Lab**

**Semester: Spring 2025**

**Software Engineering**

**Faculty of Information Technology, UCP Lahore, Pakistan**

# Graph Representations and Traversal Algorithms

## 1. Graph Representation

### a. Adjacency Matrix

int[][] adjMatrix = new int[n][n]; // where n is number of vertices  
adjMatrix[0][1] = 1; // edge from node 0 to node 1

Time Complexity:

- Space: O(V^2)  
- Add edge: O(1)  
- Check if edge exists: O(1)  
- Traverse neighbors: O(V)

### b. Adjacency List

List<List<Integer>> adjList = new ArrayList<>();  
for (int i = 0; i < V; i++) {  
 adjList.add(new ArrayList<>());  
}  
adjList.get(0).add(1); // edge from 0 to 1

Time Complexity:

- Space: O(V + E)  
- Add edge: O(1)  
- Check if edge exists: O(V)  
- Traverse neighbors: O(degree of node)

## 2. Graph Traversal Algorithms

### a. Breadth-First Search (BFS) – Queue Based

void bfs(List<List<Integer>> graph, int start) {  
 boolean[] visited = new boolean[graph.size()];  
 Queue<Integer> q = new LinkedList<>();  
 visited[start] = true;  
 q.add(start);  
  
 while (!q.isEmpty()) {  
 int node = q.poll();  
 System.out.print(node + " ");  
 for (int neighbor : graph.get(node)) {  
 if (!visited[neighbor]) {  
 visited[neighbor] = true;  
 q.add(neighbor);  
 }  
 }  
 }  
}

Time Complexity: O(V + E)

### b. Depth-First Search (DFS) – Recursive

void dfs(List<List<Integer>> graph, int node, boolean[] visited) {  
 visited[node] = true;  
 System.out.print(node + " ");  
 for (int neighbor : graph.get(node)) {  
 if (!visited[neighbor]) {  
 dfs(graph, neighbor, visited);  
 }  
 }  
}

Time Complexity: O(V + E)

## 3. Practice Tasks

**Task 1: Create Adjacency Matrix**  
- Create a 4-node undirected graph.  
- Connect nodes (0,1), (1,2), (2,3), (3,0).  
- Print matrix.  
Time: O(E)

**Task 2: Create Adjacency List**  
- Directed graph: 0→1, 0→2, 1→2, 2→3, 3→4  
- Print list.  
Time: O(V + E)

### Scenario:

You are creating a social network graph, where:

* Each person is a node
* A friendship is an undirected edge

**You will represent this using an adjacency list**

Perform:

* BFS to show all direct and indirect friends (like a social feed)
* DFS to find mutual connections (common friends between two users)

import java.util.\*;

public class SocialNetworkGraph {

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

// Add a person to the graph

public static void addPerson(String name) {

graph.putIfAbsent(name, new ArrayList<>());

}

// Add a friendship between two people (undirected)

public static void addFriendship(String a, String b) {

addPerson(a);

addPerson(b);

graph.get(a).add(b);

graph.get(b).add(a);

}

// TODO Task 1: Complete this method to perform BFS traversal

// It should print all reachable friends from the given person

public static void showAllFriendsBFS(String person) {

// Students will implement this using a Queue and visited set

}

// TODO Task 2: Complete this method to find mutual friends

// It should find and print common friends of person1 and person2

public static void findMutualFriends(String person1, String person2) {

// Students will implement this using Sets and intersection

}

public static void main(String[] args) {

// Graph setup (predefined friendships)

addFriendship("Ali", "Zara");

addFriendship("Ali", "Ahmed");

addFriendship("Zara", "Sara");

addFriendship("Ahmed", "Bilal");

addFriendship("Sara", "Bilal");

addFriendship("Sara", "Tania");

// TODO: Call and test your BFS method

// Example: showAllFriendsBFS("Ali");

// TODO: Call and test your mutual friends method

// Example: findMutualFriends("Sara", "Ahmed");

}

}

**Task 1:**

Complete the showAllFriendsBFS(String person) function  
Use a Queue and HashSet to implement BFS and print all reachable friends.

**Task 2:**

Complete the findMutualFriends(String person1, String person2) function  
Use two sets, take the intersection, and print mutual friends.

**Task 3:**

Call both methods from main() using different names.

**Task 4:**

Also give the complexity analysis of each function implemented in the code.

## Summary Table

|  |  |  |
| --- | --- | --- |
| Representation / Algorithm | Space Complexity | Time Complexity |
| Adjacency Matrix | O(V²) | O(1) add/check |
| Adjacency List | O(V + E) | O(degree) |
| BFS | O(V + E) | O(V + E) |
| DFS | O(V + E) | O(V + E) |
| Shortest Path (BFS) | O(V + E) | O(V + E) |