# Practical No 9

## Aim: Write A Program To Implement Graph

```
from collections import deque, defaultdict

class Graph:
    def __init__(self, vertices):
        self.V = vertices
        # Adjacency Matrix
        self.adj_matrix = [[0] * vertices for _ in range(vertices)]
        # Adjacency List
        self.adj_list = defaultdict(list)

    def add_edge(self, u, v):
        """Add an undirected edge between u and v"""
        # For adjacency matrix
        self.adj_matrix[u][v] = 1
        self.adj_matrix[v][u] = 1
        # For adjacency list
        self.adj_list[u].append(v)
        self.adj_list[v].append(u)

    def display_matrix(self):
        print("\nAdjacency Matrix:")
        for row in self.adj_matrix:
            print(row)

    def display_list(self):
        print("\nAdjacency List:")
        for node in self.adj_list:
            print(node, "->", self.adj_list[node])

    def bfs(self, start):
        visited = [False] * self.V
        queue = deque([start])
        visited[start] = True
        result = []

        while queue:
            node = queue.popleft()
            result.append(node)
            for neighbor in self.adj_list[node]:
                if not visited[neighbor]:
                    visited[neighbor] = True
                    queue.append(neighbor)
        return result

    def dfs(self, start):
        visited = [False] * self.V
        result = []
        def dfs_recursive(v):
            visited[v] = True
            result.append(v)
            for neighbor in self.adj_list[v]:
                if not visited[neighbor]:
                    dfs_recursive(neighbor)

        dfs_recursive(start)
        return result

# -------------------- Example Usage --------------------
if __name__ == "__main__":
    # Example: Social Network (0=Alice, 1=Bob, 2=Charlie, 3=David, 4=Eve)
    g = Graph(5)

    # Connections
    g.add_edge(0, 1)  # Alice - Bob
    g.add_edge(0, 2)  # Alice - Charlie
    g.add_edge(1, 3)  # Bob - David
    g.add_edge(2, 4)  # Charlie - Eve
    g.add_edge(3, 4)  # David - Eve

    # Display representations
    g.display_matrix()
    g.display_list()

    # Traversals
    print("\nBFS Traversal (from Alice/0):", g.bfs(0))
    print("DFS Traversal (from Alice/0):", g.dfs(0))
```

```
Adjacency Matrix:
[0, 1, 1, 0, 0]
[1, 0, 0, 1, 0]
[1, 0, 0, 0, 1]
[0, 1, 0, 0, 1]
[0, 0, 1, 1, 0]

Adjacency List:
0 -> [1, 2]
1 -> [0, 3]
2 -> [0, 4]
3 -> [1, 4]
4 -> [2, 3]

BFS Traversal (from Alice/0): [0, 1, 2, 3, 4]
DFS Traversal (from Alice/0): [0, 1, 3, 4, 2]
```