Q: Display an Adjacency Matrix for a Graph

```
#include <iostream>
#include <vector>

using namespace std;

// Function to display the adjacency matrix
void displayMatrix(const vector<vector<int>>& matrix) {
    int n = matrix.size();
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}

int main() {
    int vertices, edges;

    // Input number of vertices and edges
    cout << "Enter the number of vertices: ";
    cin >> vertices;
    cout << "Enter the number of edges: ";
    cin >> edges;

    // Initialize adjacency matrix with 0
    vector<vector<int>> adjMatrix(vertices, vector<int>(vertices, 0));

    cout << "Enter edges (format: u v):" << endl;
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;

        // Set matrix values for undirected graph
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1; // Since it's undirected
    }

    // Display the adjacency matrix
    cout << "Adjacency Matrix:" << endl;
    displayMatrix(adjMatrix);

    return 0;
}
```

Q: Display an Adjacency List for a Graph

```
#include <iostream>
#include <vector>

using namespace std;

// Function to display the adjacency list
void displayAdjList(const vector<vector<int>>& adjList) {
    for (int i = 0; i < adjList.size(); ++i) {
        cout << i << " -> ";
        for (int j : adjList[i]) {
            cout << j << " ";
        }
        cout << endl;
    }
}
```

```
int main() {
    int vertices, edges;

    // Input number of vertices and edges
    cout << "Enter the number of vertices: ";
    cin >> vertices;
    cout << "Enter the number of edges: ";
    cin >> edges;

    // Initialize adjacency list
    vector<vector<int>> adjList(vertices);

    cout << "Enter edges (format: u v):" << endl;
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;

        // Add edges to adjacency list for undirected graph
        adjList[u].push_back(v);
        adjList[v].push_back(u); // Since it's undirected
    }

    // Display the adjacency list
    cout << "Adjacency List:" << endl;
    displayAdjList(adjList);

    return 0;
}
```

Q: Detect a Cycle in an Undirected Graph

```
#include <iostream>
#include <vector>
```

```
#include <list>
#include <queue>

using namespace std;

// Function to perform DFS and detect cycle
bool dfs(int node, int parent, vector<bool>& visited, const vector<vector<int>>& adjList) {
    visited[node] = true;
    for (int neighbor : adjList[node]) {
        if (!visited[neighbor]) {
            if (dfs(neighbor, node, visited, adjList)) {
                return true;
            }
        } else if (neighbor != parent) {
            return true; // Cycle detected
        }
    }
    return false;
}

// Function to check for cycle in an undirected graph
bool hasCycle(const vector<vector<int>>& adjList, int vertices) {
    vector<bool> visited(vertices, false);
    for (int i = 0; i < vertices; ++i) {
        if (!visited[i]) {
            if (dfs(i, -1, visited, adjList)) {
                return true;
            }
        }
    }
    return false;
```

```
}

int main() {
    int vertices, edges;

    // Input number of vertices and edges
    cout << "Enter the number of vertices: ";
    cin >> vertices;
    cout << "Enter the number of edges: ";
    cin >> edges;

    // Initialize adjacency list
    vector<vector<int>> adjList(vertices);

    cout << "Enter edges (format: u v):" << endl;
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;

        // Add edges to adjacency list for undirected graph
        adjList[u].push_back(v);
        adjList[v].push_back(u); // Since it's undirected
    }

    // Check for cycle
    if (hasCycle(adjList, vertices)) {
        cout << "Cycle Detected: Yes" << endl;
    } else {
        cout << "Cycle Detected: No" << endl;
    }
    return 0;
}
```

Q: Find the Shortest Path in an Unweighted Graph using BFS

```
#include <iostream>
#include <vector>
#include <queue>
#include <stack>

using namespace std;

// Function to find the shortest path using BFS
void shortestPathBFS(int src, int dest, const vector<vector<int>>& adjList, int vertices) {
    vector<int> dist(vertices, -1);   // Distance array initialized to -1
    vector<int> parent(vertices, -1); // To store the path
    queue<int> q;

    // Start BFS from the source
    q.push(src);
    dist[src] = 0;

    while (!q.empty()) {
        int node = q.front();
        q.pop();

        for (int neighbor : adjList[node]) {
            if (dist[neighbor] == -1) { // If not visited
                dist[neighbor] = dist[node] + 1;
                parent[neighbor] = node;
                q.push(neighbor);

                if (neighbor == dest) { // Stop BFS if destination is reached
```

```
                    break;
                }
            }
        }
    }

    // Output results
    if (dist[dest] == -1) {
        cout << "No path exists between " << src << " and " << dest << endl;
    } else {
        cout << "Shortest path length: " << dist[dest] << endl;
        cout << "Path: ";
        stack<int> path;
        for (int v = dest; v != -1; v = parent[v]) {
            path.push(v);
        }
        while (!path.empty()) {
            cout << path.top();
            path.pop();
            if (!path.empty()) cout << " -> ";
        }
        cout << endl;
    }
}

int main() {
    int vertices, edges;

    // Input number of vertices and edges
    cout << "Enter the number of vertices: ";
    cin >> vertices;
    cout << "Enter the number of edges: ";
    cin >> edges;

    // Initialize adjacency list
    vector<vector<int>> adjList(vertices);

    cout << "Enter edges (format: u v):" << endl;
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;

        // Add edges to adjacency list for undirected graph
        adjList[u].push_back(v);
        adjList[v].push_back(u);
    }

    int src, dest;
    cout << "Enter source vertex: ";
    cin >> src;
    cout << "Enter destination vertex: ";
    cin >> dest;

    // Find shortest path
    shortestPathBFS(src, dest, adjList, vertices);

    return 0;
}
```