Graph Suggestion Sheet

(Undirected, Unweighted Graphs)

Given:
A graph has 9 nodes labeled from 0 to 8, i.e., $V = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$.
An edge written as $(0 - -2)$ means there is an undirected edge between nodes 0 and 2.

Use the following edges for a **non-cyclic undirected graph**

$$E = \{(0,1), (0,2), (1,3), (1,4), (2,5), (2,6), (6,7), (6,8)\}$$

Adjacency Matrix Representation

Using the given V and E, represent the graph using an **adjacency matrix** A of size 9×9 where:

$$A[i][j] = \begin{cases} 1 & \text{if there is an edge between } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

Adjacency List Representation

Using the same V and E, represent the graph using an **adjacency list**.

Problem 1: Breadth-First Search (BFS)

Write a function `BFS()` that takes:

- a graph representation (**adjacency matrix** *or* **adjacency list**),
- a `startNode`,

and performs **BFS** traversal, printing nodes in the order they are visited(Assume the graph is connected).

Write BFS for adjacency matrix:

```
// Input: A (n x n adjacency matrix), startNode
void BFS_Matrix(int A[][9], int startNode) {
    // your code here
}
```

Write BFS for adjacency list:

```
// Input: adj (vector/list of neighbors), startNode
void BFS_List(vector<vector<int>>& adj, int startNode) {
    // your code here
}
```

Problem 2: Depth-First Search (DFS)

Write a function `DFS()` that takes:

- a graph representation (**adjacency matrix** *or* **adjacency list**),
- a `startNode`,

and performs **DFS** traversal, printing nodes in the order they are visited(Assume the graph is connected).

Write DFS for adjacency matrix:

```
// Input: A (n x n adjacency matrix), startNode
void DFS_Matrix(int A[][9], int startNode) {
    // your code here
}
```

Write DFS for adjacency list:

```
// Input: adj (vector/list of neighbors), startNode
void DFS_List(vector<vector<int>>& adj, int startNode) {
    // your code here
}
```

Problem 3: Bipartite Check (Connected Graph)

A graph is **bipartite** if its vertices can be divided into two disjoint sets U and W such that every edge connects a vertex in U to a vertex in W (no edge connects two vertices in the same set).

Task: Given a **connected** graph, determine whether it is bipartite or not.

Hints:

- Use BFS/DFS coloring with two colors (e.g., 0 and 1).

Write a function:

```
// Return true if bipartite, false otherwise
bool isBipartite(vector<vector<int>>& adj) {
    // your code here
}
```

Problem 4: Connectedness Check

Determine whether a given graph is **connected**.

Write a function:

```
// Return true if connected, false otherwise
bool isConnected(vector<vector<int>>& adj) {
    // your code here
}
```

Problem 5: Shortest Distance in an Unweighted Connected Graph

Suppose a graph is **connected**. Given a `startNode` and an `endNode`, find the **shortest distance** (minimum number of edges) from `startNode` to `endNode`(No need to use parent array).

Write a function:

```
// Return shortest distance from startNode to endNode
int shortestDistance(vector<vector<int>>& adj, int startNode, int
    endNode) {
     // your code here
}
```

Problem 6: Find the Connected Components

Given an **undirected graph**, find all **connected components**.

A **connected component** is a subset of vertices such that there exists a path between every pair of vertices in this subset.

If the graph is disconnected, it may contain multiple connected components. Each connected component should be reported as a separate list of vertices.