**Q10: Cycle Detection in Graph**: Implement a graph traversal algorithm (DFS) to detect cycles in a directed and undirected graph, simulating dependencies between software modules.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Cycle Detection in Graph</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

}

.container {

display: flex;

flex-direction: column;

align-items: center;

}

.graph-input {

margin-bottom: 20px;

}

button {

padding: 10px 15px;

margin-top: 10px;

background-color: #4CAF50;

color: white;

border: none;

cursor: pointer;

border-radius: 5px;

}

button:hover {

background-color: #45a049;

}

.result {

margin-top: 20px;

font-size: 18px;

}

</style>

</head>

<body>

<div class="container">

<h1>Cycle Detection in Graph</h1>

<div class="graph-input">

<label for="graph">Enter Graph (as adjacency list, e.g., A:[B,C], B:[C], C:[]): </label>

<input type="text" id="graph" placeholder="Enter graph" size="50">

</div>

<button onclick="detectCycle()">Detect Cycle</button>

<div class="result" id="result"></div>

</div>

<script>

function detectCycle() {

const graphInput = document.getElementById('graph').value;

const graph = parseGraph(graphInput);

if (!graph) {

document.getElementById('result').innerHTML = 'Invalid graph format!';

return;

}

let visited = new Set();

let recStack = new Set();

let hasCycle = false;

// Check for cycle in directed graph

for (let node in graph) {

if (!visited.has(node)) {

if (dfs(graph, node, visited, recStack)) {

hasCycle = true;

break;

}

}

}

if (hasCycle) {

document.getElementById('result').innerHTML = 'Cycle detected in the graph!';

} else {

document.getElementById('result').innerHTML = 'No cycle detected in the graph.';

}

}

function dfs(graph, node, visited, recStack) {

visited.add(node);

recStack.add(node);

// Explore neighbors

for (let neighbor of graph[node]) {

if (!visited.has(neighbor)) {

if (dfs(graph, neighbor, visited, recStack)) {

return true;

}

} else if (recStack.has(neighbor)) {

return true;

}

}

recStack.delete(node);

return false;

}

// Parse the input graph in adjacency list format

function parseGraph(input) {

try {

let graph = {};

let rawGraph = input.split(',').map(pair => pair.trim());

rawGraph.forEach(pair => {

let [node, dependencies] = pair.split(':');

node = node.trim();

dependencies = dependencies.trim();

let neighbors = dependencies ? dependencies.slice(1, -1).split(',').map(dep => dep.trim()) : [];

graph[node] = neighbors;

});

return graph;

} catch (error) {

return null;

}

}

</script>

</body>

</html>