

## Cycle detection in undirected graph using Depth First Search in C++

```
#include <bits/stdc++.h>
using namespace std;

class Solution {
private:
    bool dfs(int node, int parent, int vis[], vector<int>
adj[]) {
        vis[node] = 1;
        // visit adjacent nodes
        for(auto adjacentNode: adj[node]) {
            // unvisited adjacent node
            if(!vis[adjacentNode]) {
                if(dfs(adjacentNode, node, vis, adj) == true)
                    return true;
            }
            // visited node but not a parent node
            else if(adjacentNode != parent) return true;
        }
        return false;
    }
public:
    // Function to detect cycle in an undirected graph.
    bool isCycle(int V, vector<int> adj[]) {
        int vis[V] = {0};
        // for graph with connected components
        for(int i = 0; i < V; i++) {
            if(!vis[i]) {
                if(dfs(i, -1, vis, adj) == true) return true;
            }
        }
        return false;
    }
};

int main() {
    // V = 4, E = 2
    vector<int> adj[4] = {{}, {2}, {1, 3}, {2}};
    Solution obj;
    bool ans = obj.isCycle(4, adj);
    if (ans)
        cout << "1\n";
    else
        cout << "0\n";
    return 0;
}
```

### Input Graph (Adjacency List)

```
vector<int> adj[4] = {
    {},          // 0 → no connections
    {2},         // 1 → connected to 2
    {1, 3},      // 2 → connected to 1 and 3
    {2}          // 3 → connected to 2
};
```

Graph in visual form:

1 -- 2 -- 3

(0 is isolated and not connected to any node.)

### DFS Function Signature

```
bool dfs(int node, int parent, int vis[],
vector<int> adj[]);
```

- node: current node being explored
- parent: node from which we came
- vis[]: visited array
- adj[]: adjacency list

### Dry Run Table

**Initial:**

- vis[4] = {0, 0, 0, 0}

### DFS Call Stack Trace

Call	Node	Parent	Visited Array	Action
1	0	-1	[1, 0, 0, 0]	No neighbors → return false
2	1	-1	[1, 1, 0, 0]	Visit 2 from 1
3	2	1	[1, 1, 1, 0]	1 is parent → skip; visit 3
4	3	2	[1, 1, 1, 1]	2 is parent → skip; DFS returns false
3↑	2	1	[1, 1, 1, 1]	DFS from 3 returned false → continue → DFS returns false
2↑	1	-1	[1, 1, 1, 1]	DFS from 2 returned false → continue → DFS

					returns false
	<div>✓ <b>Final State</b></div> <div><ul style="list-style-type: none"><li>• All nodes visited: vis = [1, 1, 1, 1]</li><li>• No back-edge found (no adjacent visited node that's not the parent)</li></ul></div> <div>📄 <b>Output:</b></div> <div>0</div>				
<div>Output:-</div> <div>0</div> <div>No cycle</div>					