DFS Cycle undirected 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;
}
```

Graph Input (V = 4):

```
vector<int> adj[4] = {
    {},     // Node 0: No edges
    {2},    // Node 1: Connected to 2
    {1, 3}, // Node 2: Connected to 1 and 3
    {2}     // Node 3: Connected to 2
};
```

So the actual edges are:

- 1 - 2
- 2 - 3

This graph is a simple path, not a cycle.

🔁 Dry Run Table (DFS traversal):

Step	Current Node	Parent	vis[] Status	Adjacent Nodes	Action	Cycle Detected?
1	0	-1	[1, 0, 0, 0]	{}	No adj nodes	No
2	1	-1	[1, 1, 0, 0]	{2}	DFS to 2	No
3	2	1	[1, 1, 1, 0]	{1, 3}	1 is parent, DFS to 3	No
4	3	2	[1, 1, 1, 1]	{2}	2 is parent, backtrack	No

🔚 No cycle detected

The code correctly determines that no adjacent node points back to a **previously visited node that's not its parent**, so there is **no cycle**.

📄 Output:

0

Output:-
0