Cycle detection in undirected graph using Breadth First Search in C++

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

class Solution {
  private:
  bool detect(int src, vector<int> adj[], int
vis[]) {
      vis[src] = 1;
      // store <source node, parent node>
      queue<pair<int,int>> q;
      q.push({src, -1});
      // traverse until queue is not empty
      while(!q.empty()) {
          int node = q.front().first;
          int parent = q.front().second;
          q.pop();

          // go to all adjacent nodes
          for(auto adjacentNode: adj[node]) {
              // if adjacent node is unvisited
              if(!vis[adjacentNode]) {
                  vis[adjacentNode] = 1;
                  q.push({adjacentNode, node});
              }
              // if adjacent node is visited and is
not it's own parent node
              else if(parent != adjacentNode) {
                  // yes it is a cycle
                  return true;
              }
          }
      }
      // there's no cycle
      return false;
  }
  public:
    // Function to detect cycle in an
undirected graph.
    bool isCycle(int V, vector<int> adj[]) {
        // initialise them as unvisited
        int vis[V] = {0};
        for(int i = 0;i<V;i++) {
            if(!vis[i]) {
                if(detect(i, adj, vis)) 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 looks like:-
1 -- 2 -- 3
0 (disconnected)

Adjacency list looks like:-
adj[0] = {} // Node 0 has no connections
adj[1] = {2} // Node 1 is connected to Node 2
adj[2] = {1, 3} // Node 2 is connected to Nodes 1 & 3
adj[3] = {2} // Node 3 is connected to Node 2

Step 1: Initialization

- vis[] = {0, 0, 0, 0} (all nodes initially unvisited).

Step 2: Iteration over Nodes (in isCycle)

1. **Check Node 0:**
    - vis[0] = 0 → call detect(0, adj, vis):
        - Node 0 has no edges (adj[0] is empty).
        - No cycle can be detected here. Return false.
    - Continue to next node.
2. **Check Node 1:**
    - vis[1] = 0 → call detect(1, adj, vis):
        - vis[1] = 1 → mark Node 1 as visited.
        - Initialize queue: q = {{1, -1}} (Node 1 with parent -1).
        - **Process Queue:**
            - Dequeue q.front() → node = 1, parent = -1.
            - Adjacent to Node 1 → Node 2.
                - vis[2] = 0 → mark Node 2 as visited, push {2, 1} to q.
                - Queue: q = {{2, 1}}.
            - Dequeue q.front() → node = 2, parent = 1.
                - Adjacent to Node 2 → Nodes 1 and 3.
                - **Node 1:** Already visited, but parent == 1 → No cycle detected here.
                - **Node 3:** vis[3] = 0 → mark Node 3 as visited, push {3, 2} to q.
                - Queue: q = {{3, 2}}.
            - Dequeue q.front() → node = 3, parent = 2.
                - Adjacent to Node 3 → Node 2.

<table>
<tr>
<td></td>
<td>

Node 2: Already visited, but parent == 2 → No cycle detected here.
Queue is empty, no cycle found. Return false.

3. Check Nodes 2 and 3:

Both are already visited (vis[2] = 1, vis[3] = 1).
Skip further checks.

</td>
</tr>
<tr>
<td colspan="2">Output:-
0
No cycle was found in any component of the graph</td>
</tr>
</table>