#include <iostream>

#include <vector>

#include <unordered\_set>

#include <algorithm>

Class Graph {

Public:

Int nodes;

Std::vector<std::vector<int>> adj;

Graph(int n) : nodes(n) {

Adj.resize(n);

}

Void addEdge(int u, int v) {

Adj[u].push\_back(v);

Adj[v].push\_back(u); //

}

Int findLongestCycle() {

Int longestCycleLength = 0;

Std::vector<bool> visited(nodes, false);

For (int i = 0; i < nodes; i++) {

Std::unordered\_set<int> path;

Dfs(i, -1, 0, visited, path, longestCycleLength);

}

Return longestCycleLength;

}

Private:

Void dfs(int node, int parent, int length, std::vector<bool>& visited,

Std::unordered\_set<int>& path, int& longestCycleLength) {

t path, we’ve found a cycle

If (path.find(node) != path.end()) {

longestCycleLength = std::max(longestCycleLength, length);

return;

}

Path.insert(node);

Visited[node] = true;

For (int neighbor : adj[node]) {

If (neighbor != parent) {

Dfs(neighbor, node, length + 1, visited, path, longestCycleLength);

}

}

Path.erase(node);

}

};

Int main() {

Graph g(7);

g.addEdge(0, 1);

g.addEdge(1, 2);

g.addEdge(2, 3);

g.addEdge(3, 4);

g.addEdge(4, 5);

g.addEdge(5, 6);

g.addEdge(6, 0); // Adding a cycle

int longestCycle = g.findLongestCycle();

if (longestCycle > 0) {

std::cout << “The length of the longest cycle is: “ << longestCycle << std::endl;

} else {

Std::cout << “No cycles found.” << std::endl;

}

Return 0;

}