Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Fall, Year:2023), B.Sc. in CSE (Day)

Lab Report NO: 01

Course Title : Algorithms Lab
Course Code : CSE 204
Section : 221 D9

Lab Experiment Name: Detecting Cycles in an Undirected Graph using BFS

Student Details

Name		ID
1.	Jahidul Islam	221002504

Lab Date : 02-09-2023
Submission Date : 09-10-2023
Course Teacher's Name : Md. Abu Rumman Refat

Lab Report Status

Marks: …… **Signature:.....**
Comments:..... **Date:.....**

1. Introduction:

The objective of this lab experiment is to implement an algorithm to detect cycles in an undirected graph using Breadth-First Search (BFS). Cycles in a graph are fundamental structures, and detecting them has various applications in computer science, such as in network analysis and route planning.

In this lab report, we will use C++ to implement the BFS-based cycle detection algorithm and test it on sample graphs.

2. OBJECTIVES/AIM:

1. To implement the Breadth-First Search (BFS) algorithm as the primary method for graph traversal and cycle detection.
2. To efficiently detect and report the presence or absence of cycles within the input graph.
3. To provide a clear and concise explanation of the program's functionality and implementation.

3. Equipment and Software Used:

1. C++ Programming Language
2. IDE – Code Blocks
3. Graph Visualization Tools

4. Implementation:

1. **Graph Representation:** The program uses an adjacency list to represent the undirected graph. Each vertex is associated with a list of its neighboring vertices.
2. **Breadth-First Search (BFS):** BFS is implemented to traverse the graph. It starts from an initial vertex and explores all of its neighbors level by level.
3. **Cycle Detection:** During BFS, if the algorithm encounters a neighbor that has already been visited and is not the parent of the current node, it means a cycle has been detected.

4. **Data Structures:** The program uses a queue to implement BFS, a Boolean array to track visited nodes, and a parent array to track the parent of each node in the traversal.
5. **Input:** The user provides the number of vertices and edges in the graph, followed by the edges themselves.
6. **Output:** The program outputs whether a cycle is detected or not.

4. Code in C++

```cpp
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

bool isCyclic(vector<vector<int>>& adjList, int vertices) {
    vector<bool> visited(vertices, false);
    vector<int> parent(vertices, -1);

    for (int i = 0; i < vertices; ++i) {
        if (!visited[i]) {
            queue<int> q;
            q.push(i);
            visited[i] = true;

            while (!q.empty()) {
                int current = q.front();
                q.pop();

                for (int neighbor : adjList[current]) {
                    if (!visited[neighbor]) {
                        visited[neighbor] = true;
                        q.push(neighbor);
                        parent[neighbor] = current;
                    } else if (parent[current] != neighbor) {
                        // If the neighbor is visited and not the parent, a cycle is detected.
                        return true;
                    }
                }
            }
        }
    }
    return false;
}
```

```
36
37  int main() {
38      int vertices, edges;
39      cout << "Enter the number of vertices and edges: ";
40      cin >> vertices >> edges;
41
42      vector<vector<int>> adjList(vertices);
43
44      cout << "Enter the edges (vertex pairs):" << endl;
45      for (int i = 0; i < edges; ++i) {
46          int u, v;
47          cin >> u >> v;
48          adjList[u].push_back(v);
49          adjList[v].push_back(u);
50      }
51
52      if (isCyclic(adjList, vertices)) {
53          cout << "Cycle detected in the graph." << endl;
54      } else {
55          cout << "No cycle detected in the graph." << endl;
56      }
57
58      return 0;
59  }
60
```

5. OUTPUT:

```
Enter the number of vertices and edges: 3 3
Enter the edges (vertex pairs):
0 1
1 2
2 1
0 1

1 2

2 1
Cycle detected in the graph.
```

```
Enter the number of vertices and edges: 5 6
Enter the edges (vertex pairs):
0 1
0 2
1 2
2 3
3 4
4 1
0 1

0 2

1 2

2 3

3 4

4 1
Cycle detected in the graph.
```

6. DISCUSSION:

The BFS-based cycle detection algorithm has been successfully implemented in C++. It correctly identified the presence or absence of cycles in the test cases, producing results consistent with our expectations.

The time complexity of the algorithm is O(V + E), where V is the number of vertices and E is the number of edges. This is because each vertex and edge are visited once during the BFS traversal.

7. CONCLUSION:

Detecting cycles in a graph is a fundamental problem with numerous applications. This C++ program effectively uses Breadth-First Search to detect cycles in an undirected graph. By providing the number of vertices and edges along with the edge connections, the program can determine whether a cycle exists within the graph. This algorithm is essential for various graph-based problems and can be further extended for directed graphs or to find the specific nodes involved in a cycle.