

GREEN UNIVERSITY OF BANGLADESH



Department of Computer Science and Engineering (CSE)

Semester: (Fall, Year:2025), B.Sc. in CSE (Day)  
  
Lab Report NO: 03

**Experiment Name:** **Implement Bread-First Search Traversal.**  
**Course Title** : Algorithm lab.

Course Code: CSE 208 Section: D8

Student Details:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Name |  | ID |
| 1. | Majharul Islam | 232002256 |  |

Submission Date. : 26.02.2025   
Course Teacher’s Name: Feroza Naznin

[For Teachers use only: Don’t Write Anything inside this box]

Assignment Report Status

Marks: ………………………………… Signature:.....................

Comments:.............................................. Date:..............................

**Detect Cycle in a Graph using BFS**

**1. TITLE OF THE LAB REPORT EXPERIMENT**

Detecting Cycles in a Graph using BFS

**2. OBJECTIVES**

The objective of this experiment is to implement a method for detecting cycles in an undirected graph using the Breadth-First Search (BFS) algorithm. The goals include:

* Understanding graph traversal techniques.
* Implementing BFS to detect cycles efficiently.
* Analyzing the time complexity of the approach.

**3. PROCEDURE / ANALYSIS / DESIGN**

**Algorithm:**

1. Represent the graph using an adjacency list.
2. Use a queue to perform BFS traversal.
3. Maintain a visited array to track visited nodes.
4. For each node, check if an already visited node is encountered again (except its immediate parent).
5. If a visited node is found that is not the parent, a cycle exists.
6. Continue the process for all connected components of the graph.

**Pseudocode:**

|  |
| --- |
| function isCyclic(graph, V):  visited = [False] \* V  for each node in graph:  if node is not visited:  if BFS detects a cycle:  return True  return False  function BFS(graph, start, visited):  queue = [(start, -1)]  visited[start] = True  while queue is not empty:  node, parent = queue.dequeue()  for neighbor in graph[node]:  if not visited[neighbor]:  visited[neighbor] = True  queue.enqueue((neighbor, node))  else if neighbor != parent:  return True  return False |

**4. IMPLEMENTATION**

|  |
| --- |
| import java.util.\*;  class Graph {  private int V;  private List<List<Integer>> adj;  public Graph(int V) {  this.V = V;  adj = new ArrayList<>(V);  for (int i = 0; i < V; i++) {  adj.add(new ArrayList<>());  }  }  public void addEdge(int u, int v) {  adj.get(u).add(v);  adj.get(v).add(u); // Undirected graph  }  public boolean isCyclic() {  boolean[] visited = new boolean[V];  for (int i = 0; i < V; i++) {  if (!visited[i]) {  if (bfsCycleCheck(i, visited)) {  return true;  }  }  }  return false;  }  private boolean bfsCycleCheck(int start, boolean[] visited) {  Queue<int[]> queue = new LinkedList<>();  queue.add(new int[]{start, -1}); // {node, parent}  visited[start] = true;  while (!queue.isEmpty()) {  int[] nodePair = queue.poll();  int node = nodePair[0], parent = nodePair[1];  for (int neighbor : adj.get(node)) {  if (!visited[neighbor]) {  visited[neighbor] = true;  queue.add(new int[]{neighbor, node});  } else if (neighbor != parent) {  return true; // Cycle detected  }  }  }  return false;  }  }  public class DetectCycleBFS {  public static void main(String[] args) {  Graph graph = new Graph(5);  graph.addEdge(0, 1);  graph.addEdge(1, 2);  graph.addEdge(2, 3);  graph.addEdge(3, 4);  graph.addEdge(4, 1); // Introduces a cycle    if (graph.isCyclic()) {  System.out.println("Cycle detected in the graph.");  } else {  System.out.println("No cycle detected in the graph.");  }  }  } |

**5. TEST RESULT / OUTPUT**

**Test Cases:**

**Input:**

Graph:

0 - 1 - 2 - 3 - 4

|\_\_\_\_\_\_\_\_\_\_|

**Output:**

Cycle detected in the graph.

**Explanation:**

* The BFS traversal detects that node 1 is visited again from node 4, indicating a cycle.

**6. ANALYSIS AND DISCUSSION**

**What went well?**

* Successfully implemented BFS-based cycle detection.
* Used an efficient queue-based approach for BFS traversal.

**Trouble Spots:**

* Ensuring proper parent tracking to avoid false cycle detection.

**Difficult Parts:**

* Maintaining correctness in handling disconnected components.

**Learnings:**

* BFS can be used effectively to detect cycles in an undirected graph.
* The time complexity of BFS cycle detection is **O(V + E)**.

**Mapping of Objectives:**

* Achieved the goal of detecting cycles in a graph using BFS.
* Demonstrated efficient BFS traversal and cycle detection.

**7. SUMMARY**

Cycle detection in an undirected graph can be achieved using BFS. The approach involves tracking visited nodes and ensuring a node is not revisited unless it is not the immediate parent. The experiment reinforced understanding of BFS and its application in graph problems.