Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 01

Course Title: ALGORITHMS LAB

Course Code: CSE 206 Section: 231 (D2)

Lab Experiment Name: Implement Bread-First Search Traversal

Student Details

	Name	ID
1.	Promod Chandra Das	231002005

Lab Date :

Submission Date :

Course Teacher's Name : Farjana Akter Jui

Lab Report Status

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

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

❖ TITLE OF THE LAB REPORT EXPERIMENT

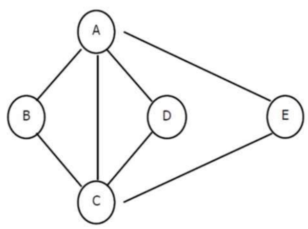
Implement Bread-First Search Traversal

❖ OBJECTIVES/AIM

To understand how to represent a graph using an adjacency matrix.
To understand how Bread-First Search (BFS) works.

❖ IMPLEMENTATION

Every graph is a set of points referred to as vertices or nodes which are connected using lines called edges.The vertices represent entities in a graph. Edges, on the other hand, express relationships between entities.Hence, while nodes model entities, edges model relationships in a network graph. A graph G with a set of V vertices together with a set of E edges is represented as G= (V, E). Both vertices and edges can have additional attributes that are used to describe the entities and relationships. Figure 1 depicts a simple graph with five nodes and seven edges.



Adjacency Matrix:

Vertices are labeled (or re-labeled) with integers from 0 to V (G) - 1. A two-dimensional array "matrix" with dimensions V (G) * V (G) contains a 1 at matrix [j] [k] if there is an edge from the vertex labeled j to the
vertex labeled k, and a 0 otherwise.Table:1 represents the graph of figure:1;

	A	B	C	D	E
A	0	1	1	1	1
B	1	0	1	0	0
C	1	1	0	1	1
D	1	0	1	0	0
E	1	0	1	0	0

Table: 1

Algorithm (Adjacency Matrix)

Step 1. Set i=0, e = Number of edges.

Step 2. e (number of edge) < i (Decision). • if no - continue with the step 7.

Step 3. Take the values of edge by giving the adjacency nodes [j], [k] (A, B, C, D, E=0,1,2,3,4).

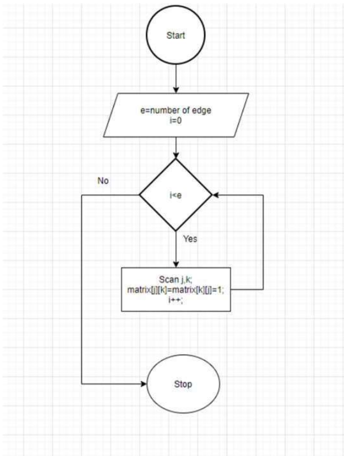
Step 4. matrix[j][k] =matrix[k][j] =1.

Step 5. Increment i (i++).

Step 6. continue with the step 2.

Step 7. Stop.

❖ Flowchart

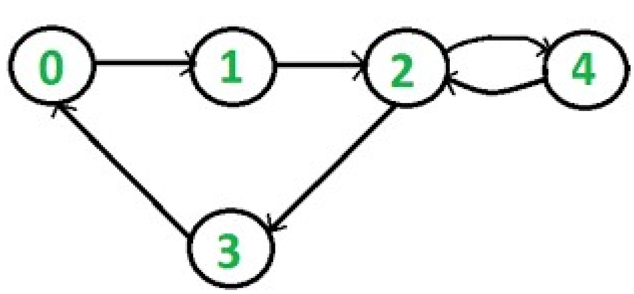


- ❖ **Lab Exercise (Submit as a report)**

<u>Write a program to detect the cycle in a graph using BFS.</u>

```
import java.util.*;

public class Graph {
 private int V;
 private LinkedList<Integer> adj[];

 Graph(int v) {
  V = v;
  adj = new LinkedList[v];
  for (int i = 0; i < v; ++i)
   adj[i] = new LinkedList();
 }


 void addEdge(int v, int w) {
  adj[v].add(w);
 }

 void BFS(int s) {

  boolean visited[] = new boolean[V];

  LinkedList<Integer> queue = new LinkedList();

  visited[s] = true;
```

```java
    queue.add(s);

    while (queue.size() != 0) {
      s = queue.poll();
      System.out.print(s + " ");

      Iterator<Integer> i = adj[s].listIterator();
      while (i.hasNext()) {
        int n = i.next();
        if (!visited[n]) {
          visited[n] = true;
          queue.add(n);
        }
      }
    }
  }

  public static void main(String args[]) {
    Graph g = new Graph(4);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    System.out.println("Following is Breadth First Traversal " + "(starting from vertex 2)");

    g.BFS(2);
  }
}
```

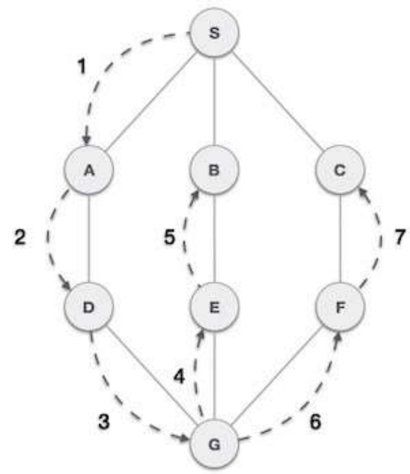


❖ Objective(s)

- To understand how to represent a graph using adjacency list.
- To understand how Depth-First Search (DFS) works.

❖ **Problem analysis**

Two of the most popular tree traversal algorithms are breadth-frst search (BFS) and depth-frst search (DFS).
Both methods visit all vertices and edges of a graph; however, they are diferent in the way in which they
perform the traversal. This diference determines which of the two algorithms is better suited for a specifc
purpose.



Adjacency List:

Vertices are labelled (or re-labelled) from 0 to V(G)- 1. Corresponding to each vertex is a list (either an array
or linked list) of its neighbours. Table: 1 represents the adjacency list of fgure1.

A to	D, S
B to	E, S
C to	F, S
D to	A, G
E to	B, G
F to	C, G
G to	D, E, F
S to	A, B, C

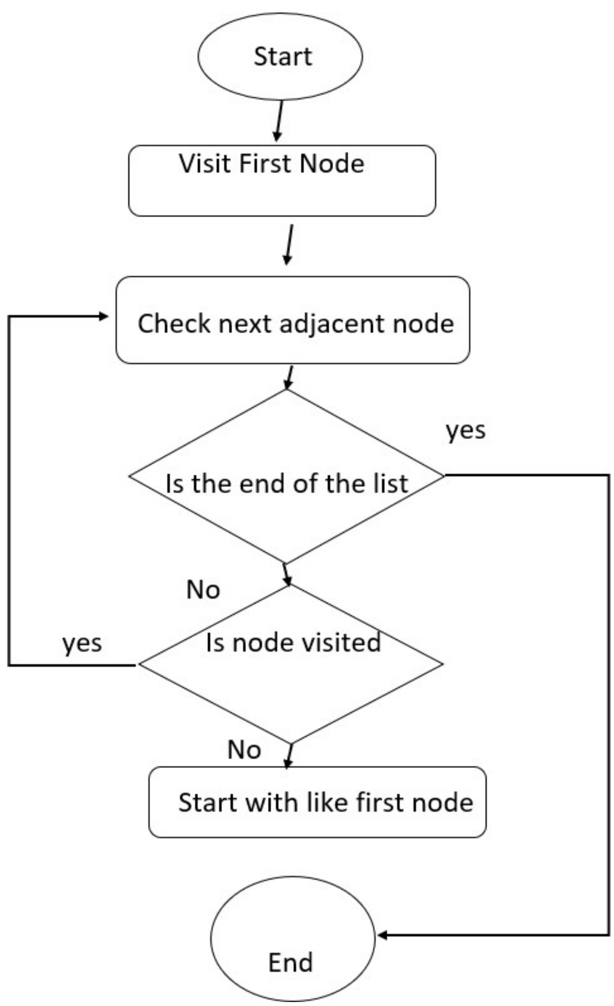
Table:1

DFS:

Depth-frst Search or Depth-frst traversal is a recursive algorithm for searching all the vertices of a graph or tree
data structure. Traversal means visiting all the nodes of a graph. Figure 1 shows the DFS graph traversal.As

in the example given above, the DFS algorithm traverses from S to A to D to G to E to B frst, then to F, and
lastly to C. It employs the following rules.
3 Algorithm (DFS)
A standard DFS implementation puts each vertex of the graph into one of two categories:
1. Visited 2. Not Visited
The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
The DFS algorithm works as follows:
Step 1. Start by putting any one of the graph's vertices on top of a stack.
Step 2. Take the top item of the stack and add it to the visited list.
Step 3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of
the stack.
Step 4. Keep repeating steps 2 and 3 until the stack is empty.

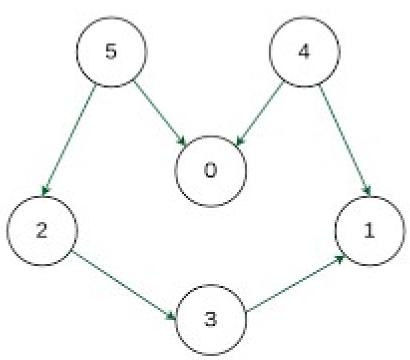
1 Flowchart



- **Lab Exercise (Submit as a report)**
  <u>Write a program to perform a topological search using BFS.</u>

```
import java.util.*;

public class TopologicalSortBFS {

    public static List<String> topologicalSortBFS(List<String> vertices, List<int[]> edges) {

        Map<String, List<String>> graph = new HashMap<>();
        Map<String, Integer> inDegree = new HashMap<>();

        for (String vertex : vertices) {
            graph.put(vertex, new ArrayList<>());
            inDegree.put(vertex, 0);
        }


        for (int[] edge : edges) {
            String u = vertices.get(edge[0]);
            String v = vertices.get(edge[1]);
            graph.get(u).add(v);
            inDegree.put(v, inDegree.get(v) + 1);
        }


        Queue<String> queue = new LinkedList<>();
        for (String vertex : vertices) {
            if (inDegree.get(vertex) == 0) {
                queue.add(vertex);
            }
        }

        List<String> topOrder = new ArrayList<>();


        while (!queue.isEmpty()) {
            String node = queue.poll();
            topOrder.add(node);

            for (String neighbor : graph.get(node)) {
                inDegree.put(neighbor, inDegree.get(neighbor) - 1);
                if (inDegree.get(neighbor) == 0) {
                    queue.add(neighbor);
                }
            }
        }
```

```
        if (topOrder.size() == vertices.size()) {
            return topOrder;
        } else {
            throw new RuntimeException("The graph is not a DAG (contains a cycle)");
        }
    }

    public static void main(String[] args) {
        List<String> vertices = Arrays.asList("A", "B", "C", "D", "E", "F");
        List<int[]> edges = Arrays.asList(
            new int[]{0, 2}, // A -> C
            new int[]{1, 2}, // B -> C
            new int[]{2, 3}, // C -> D
            new int[]{3, 4}, // D -> E
            new int[]{4, 5}  // E -> F
        );

        try {
            List<String> result = topologicalSortBFS(vertices, edges);
            System.out.println("Topological Sort Order: " + result);
        } catch (RuntimeException e) {
            System.out.println(e.getMessage());
        }
    }
}
```



- **ANALYSIS AND DISCUSSION**

  Breadth-First Search (BFS) explores graph or tree structures level by level, using a queue to track nodes. Starting from a source node, it visits all its neighbors before moving to the next level. BFS and DFS efficiently finds the shortest path in unweighted graphs and supports level-order traversal in trees