# **LEC-14 Graph/Graph Theory**

## **Graph:**

* A graph is a collection of nodes (vertices) and edges that connect pairs of nodes.
* Graphs are versatile data structures used to model relationships between objects, network connections, and more.
* They provide a powerful way to represent complex relationships and solve various problems.

## **Types of Graphs:**

Undirected Graph:

* Edges have no direction, and they connect two vertices.
* Example: Social network connections.

Directed Graph (Digraph):

* Edges have a direction from one vertex to another.
* Example: Website links or dependency relationships.

Weighted Graph:

* Edges have associated weights, representing costs, distances, etc.

Cyclic and Acyclic Graphs:

* Cyclic graphs contain cycles (loops), while acyclic graphs do not.

# **Graph Representation:**

Adjacency Matrix:

A 2D matrix where each cell (i, j) represents an edge between vertex i and vertex j.

Adjacency List:

Each vertex has a list of its neighboring vertices.

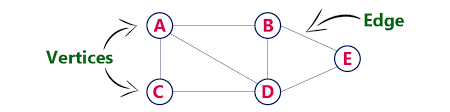
# **Graph Traversal:**

Depth-First Search (DFS):

* Explores as far as possible along each branch before backtracking.
* Uses a stack or recursion.
* Used for tasks like topological sorting and finding connected components.

Breadth-First Search (BFS):

* Explores all neighbor vertices at the current level before moving to the next level.
* Uses a queue.
* Useful for finding shortest paths and spanning trees.



Graph(Vertex,Edge) or G(V,E)

**Vertex or Node:**

V={a,b,c,d,e}

* Order of Graph= | V |

**Edge:**

E={7 lines }

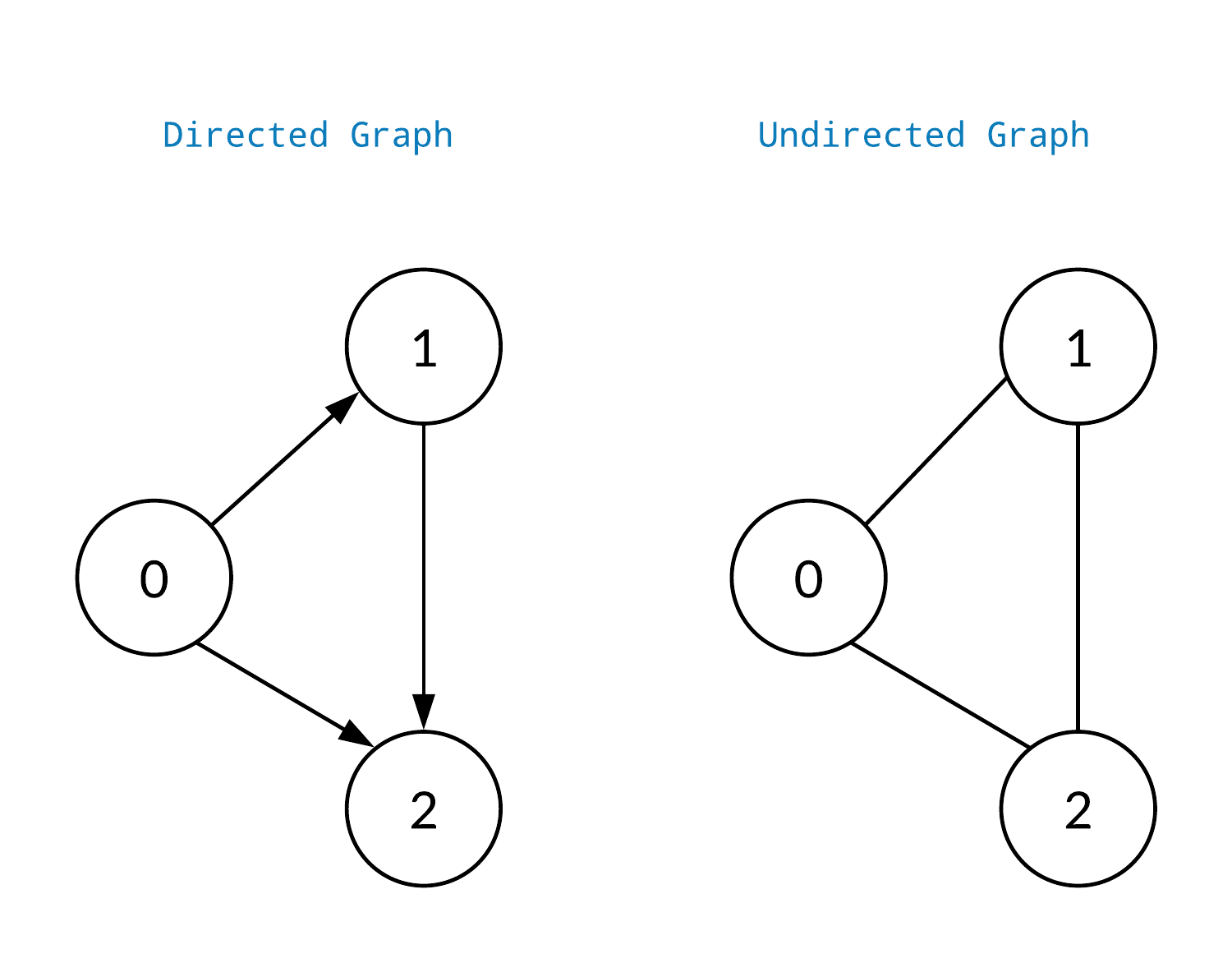
* Size of Graph= | E |

**Types:**

🡪Directed Graph(has specific direction(s))

🡪Undirected Graph

E:g



**Adjacent Values: (vertex with common edge)**

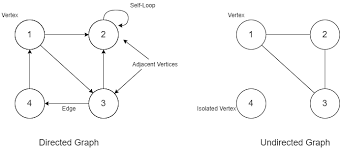
E:g {(1,2),(1,3),……}

(1,4) X

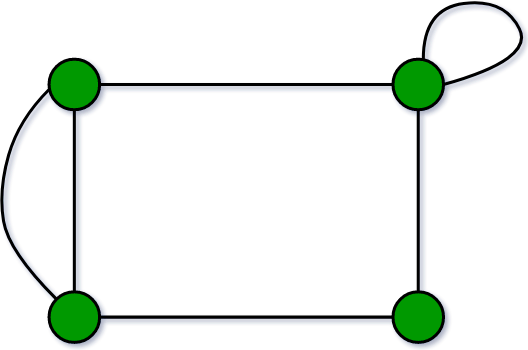
**Adjacent Edge: (edge with common vertex)**

E:g {(a,b),……}

**Graph with self loop:**

****

**Pseudo Graph:**

****

**Multi Graph:**

Having multi-path but no self-loop

**Simple Graph:**

No self-loop and no multiple path

**Traversal of Graph: (Visit and explore)**

i)BFS

ii)DFS

🡪BFS(Breadth-First Search):

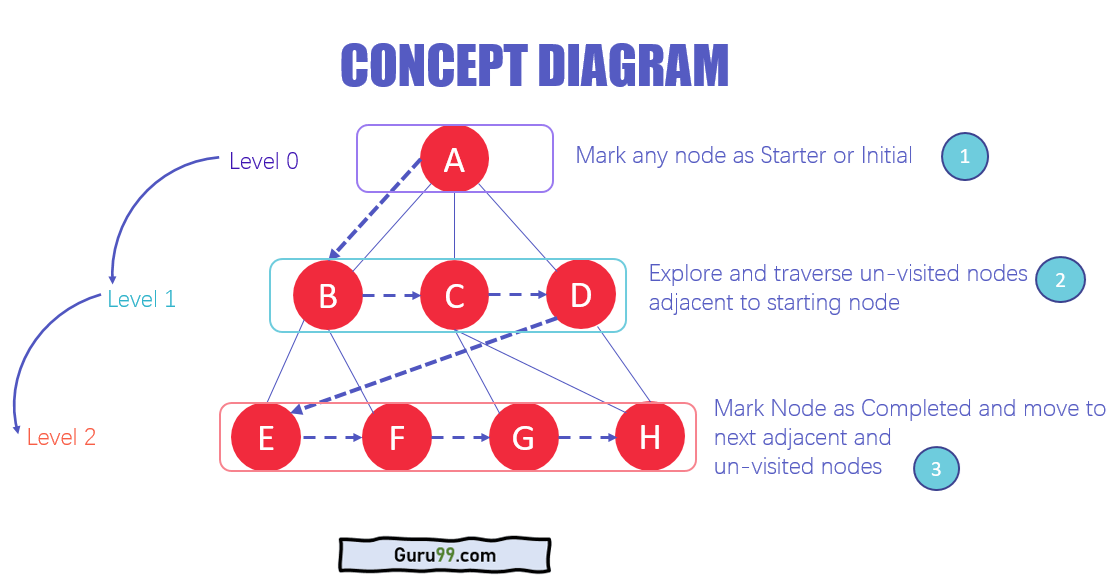
No back tracking or Level wise traversal

🡪DFS(Depth-First Search):

Supports back tracking

E:g

**BFS**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H |

## Rules of BFS Algorithm

* A queue data structure is used by BFS.
* You mark any node in the graph as root.
* BFS visits an adjacent unvisited node,
* Removes the previous vertex from the graph
* BFS algorithm iterates until all vertices in the graph are traversed.
* <https://www.guru99.com/breadth-first-search-bfs-graph-example.html>

**DFS**

* DFS uses a stack (implicitly or explicitly) to traverse through nodes.
* Start by selecting any node as the initial node and mark it as visited.
* Explore as far as possible along each branch before backtracking.
* Visit an adjacent unvisited node and make it the current node.
* If there are no unvisited adjacent nodes, backtrack to the previous node.
* DFS continues until all vertices in the graph are visited.
* DFS can be implemented recursively or using an explicit stack data structure.

**🡪Representation of graph using adjacent list:**

1

0

2

3

4

**🡪Adjacency list:**

We maintain an array of vertex. Every index will have a linked list of adjacent edges.

**🡪Size of array in adjacency:**

Answer:

|  |
| --- |
|  |

🡪Representation:

|  |  |  |
| --- | --- | --- |
| 0 | 1 | 2 |
| 1 | 0 | 3 |
| 2 | 0 | 3 |
| 3 | 1 | 2 | 4 |
| 4 | 3 |