Graph Algorithms

Graph is a non-linear data structure like tree data structure. A Graph is composed of a set of vertices(V) and a set of edges(E). The vertices are connected with each other through edges.

- The limitation of tree is, it can only represent hierarchical data. For situations where nodes or vertices are randomly connected with each other other, we use Graph.
- Example situations where we use graph data structure are, a social network, a computer network, a network of locations used in GPS and many more examples where different nodes or vertices are connected without any hierarchic or constraint on structure.

Representation of Graph

A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices(V) and a set of edges(E). The graph is denoted by G(V, E).

Here are the two most common ways to represent a graph:

1. Adjacency Matrix
2. Adjacency List

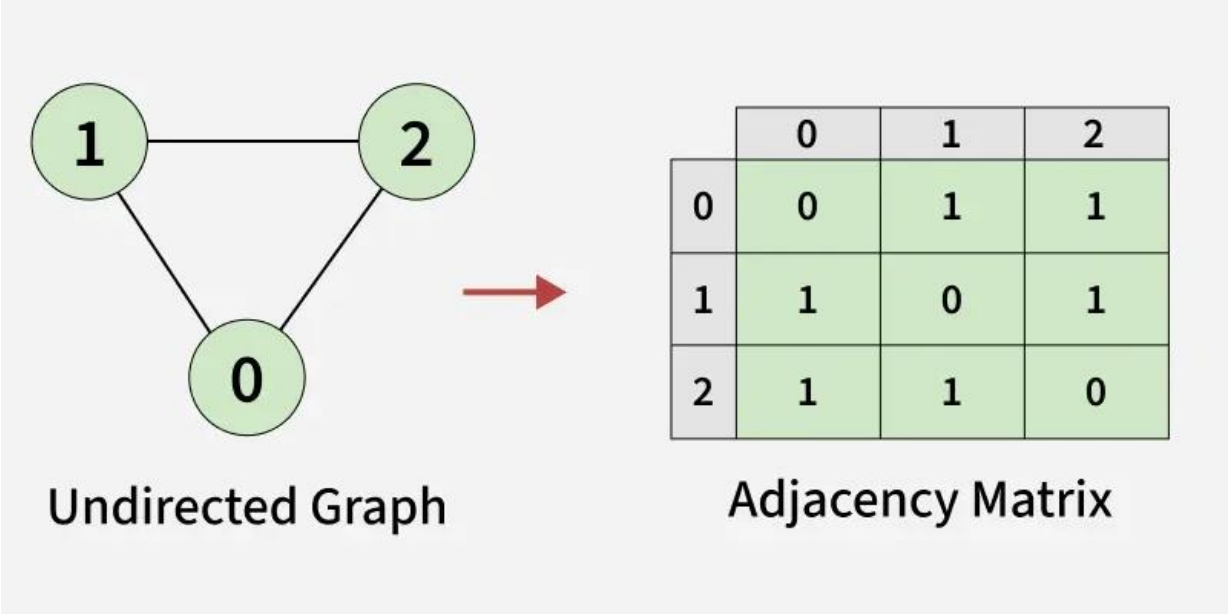
Adjacency Matrix Representation

An adjacency matrix is a way of representing a graph as a boolean matrix of (0's and 1's).

Let's assume there are n vertices in the graph So, create a 2D matrix adjMat[n][n] having dimension n x n.

- If there is an edge from vertex i to j, mark adjMat[i][j] as 1.
- If there is no edge from vertex i to j, mark adjMat[i][j] as 0.

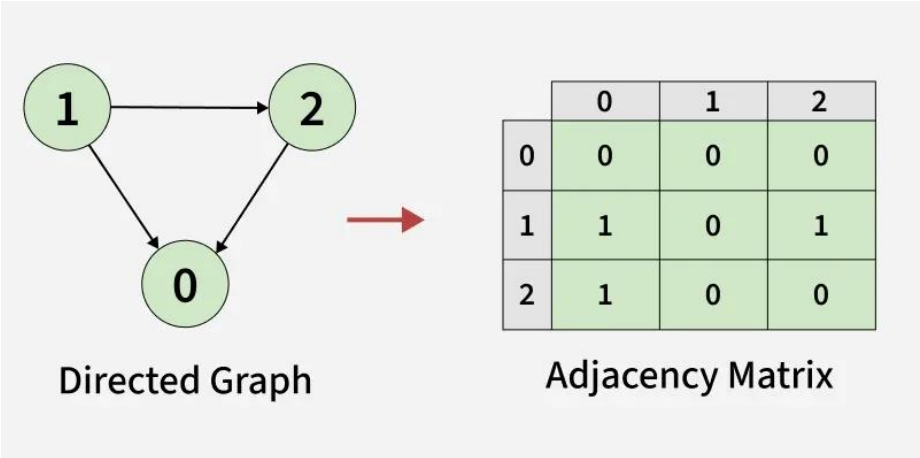
Undirected Graph:



	0	1	2
0	0	1	1
1	1	0	1
2	1	1	0

- We use an adjacency matrix to represent connections between vertices.
- Initially, the entire matrix is filled with 0s, meaning no edges exist.
- There is an edge between vertex 0 and vertex 1,so we set mat[0][1] = 1 and mat[1][0] = 1.
- There is an edge between vertex 0 and vertex 2,so we set mat[0][2] = 1 and mat[2][0] = 1.
- There is an edge between vertex 1 and vertex 2,so we set mat[1][2] = 1 and mat[2][1] = 1.

Directed Graph:



	0	1	2
0	0	0	0
1	1	0	1
2	1	0	0

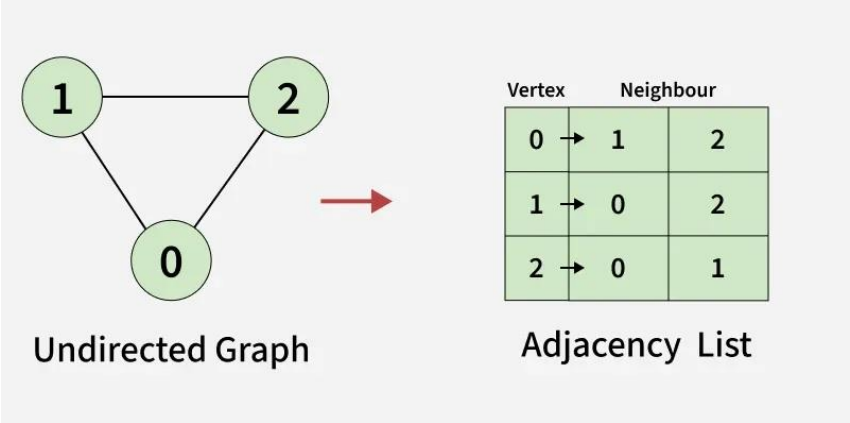
- Initially, the entire matrix is filled with 0s, meaning no edges exist.
- Unlike an undirected graph, we do not set mat[destination][source] because the edge goes in only one direction.
- There is an edge between vertex 1 and vertex 0,so we set mat[1][0] = 1.
- There is an edge between vertex 2 and vertex 0,so we set mat[2][0] = 1.
- There is an edge between vertex 1 and vertex 2,so we set mat[1][2] = 1.

Adjacency List Representation

An array of Lists is used to store edges between two vertices. The size of array is equal to the number of vertices (i.e, n). Each index in this array represents a specific vertex in the graph. The entry at the index i of the array contains a linked list containing the vertices that are adjacent to vertex i. Let's assume there are n vertices in the graph So, create an array of list of size n as adjList[n].

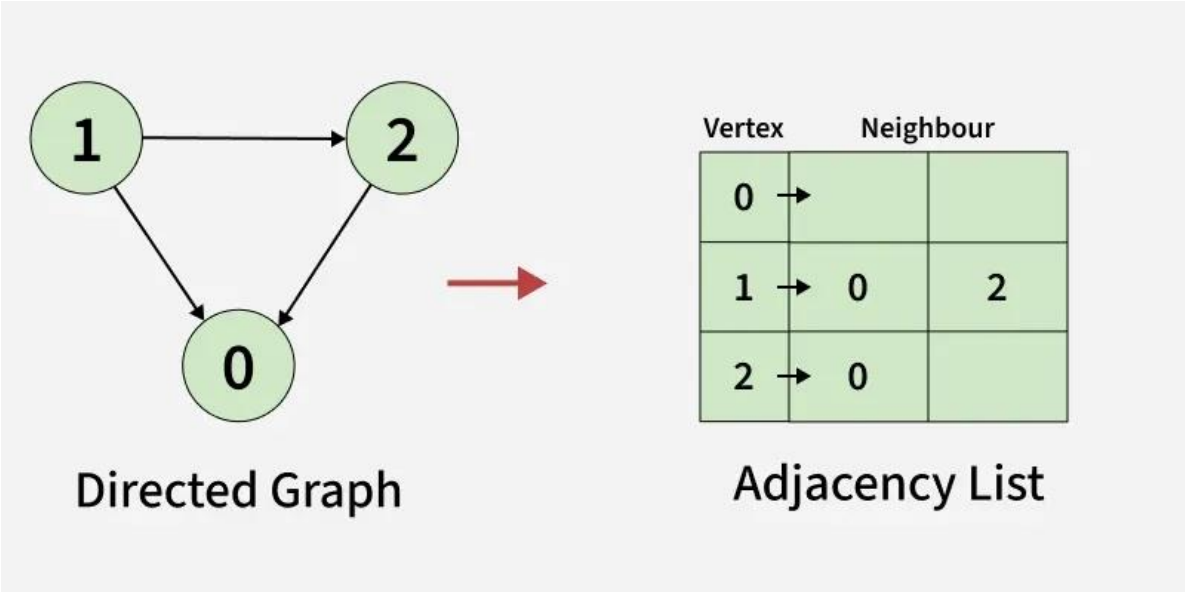
- adjList[0] will have all the nodes which are connected (neighbour) to vertex 0.
- adjList[1] will have all the nodes which are connected (neighbour) to vertex 1 and so on.

Undirected Graph:



- We use an array of lists (or vector of lists) to represent the graph.
- The size of the array is equal to the number of vertices (here, 3).
- Each index in the array represents a vertex.
- Vertex 0 has two neighbours (1 and 2).
- Vertex 1 has two neighbours (0 and 2).
- Vertex 2 has two neighbours (0 and 1).

Directed Graph:



- We use an array of lists (or vector of lists) to represent the graph.
- The size of the array is equal to the number of vertices (here, 3).
- Each index in the array represents a vertex.
- Vertex 0 has no neighbours
- Vertex 1 has two neighbours (0 and 2).
- Vertex 2 has 1 neighbours (0).

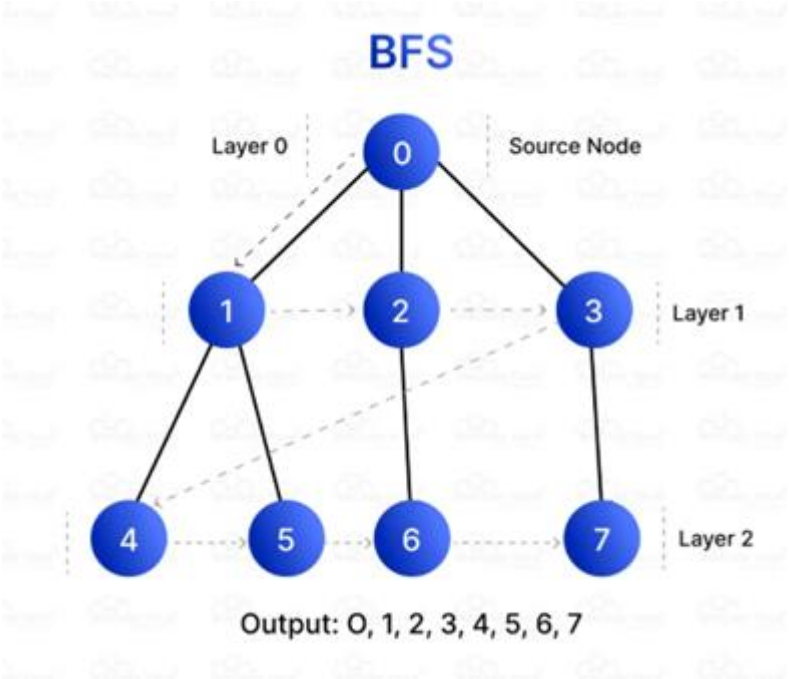
Breadth-First Search

BFS (Breadth-First Search) is a graph traversal algorithm that explores nodes level by level, starting from a given source node. It visits all neighboring nodes at the current depth (or level) before moving on to nodes at the next depth level. BFS is commonly used for traversing or searching tree or graph data structures.

BFS

Approach: BFS explores the graph level by level, starting from the source node. It visits all the nodes at one level before moving to the next. BFS uses a queue to keep track of nodes that need to be explored.

How it works: From the source node, BFS moves to all its direct neighbors (Layer 1), then moves to the next layer (Layer 2), continuing until all nodes are visited.



Traversal Order:

- Starting from node 0:
- Layer 0: Visit node 0 (source node).
- Layer 1: Visit nodes 1, 2, 3 (neighbors of node 0).

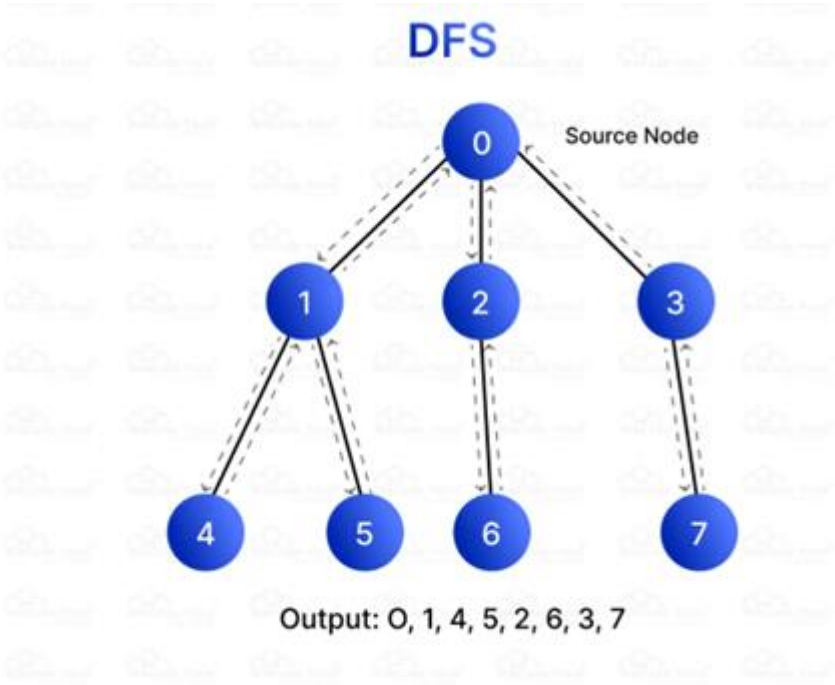
- Layer 2: Visit nodes 4, 5, 6, 7 (neighbors of nodes 1, 2, 3).
- Output: 0, 1, 2, 3, 4, 5, 6, 7.

Depth-First Search

- Depth-First Search (DFS) is a popular algorithm used for traversing or searching through graph or tree data structures. It starts at a source node (or vertex) and explores as far along each branch or path as possible before backtracking.
- This means it dives deep into the graph before moving to other branches, hence the name "depth-first." DFS can be implemented using either recursion or a stack.

Approach: DFS explores as deep as possible along a branch before backtracking. It uses a stack (or recursion) to keep track of nodes and backtracks when it reaches a dead end.

How it works: Starting from the source node, DFS goes deeper into the graph, visiting one branch of the graph first before backtracking to explore others.



Traversal Order:

- Starting from node 0:
- Visit node 0 → Visit node 1 → Visit node 4 (deepest) → Backtrack to node 1 → Visit node 5 → Backtrack to node 0 → Visit node 2 → Visit node 6 → Backtrack to node 2 → Visit node 3 → Visit node 7.
- Output: 0, 1, 4, 5, 2, 6, 3, 7