**Agenda:**

- Introduction to Graphs
- Types of Graphs
- Representation of Graphs
    - Adjacency matrix
    - Adjacency list
- Depth First Search (DFS) Traversal
    - Finding the number of islands
- Breadth First Search (BFS) Traversal
    - Shortest path between two nodes (HW)

## Introduction to Graphs

A graph is a collection of nodes (also called vertices) connected via edges.

Comparing Graphs with Trees:

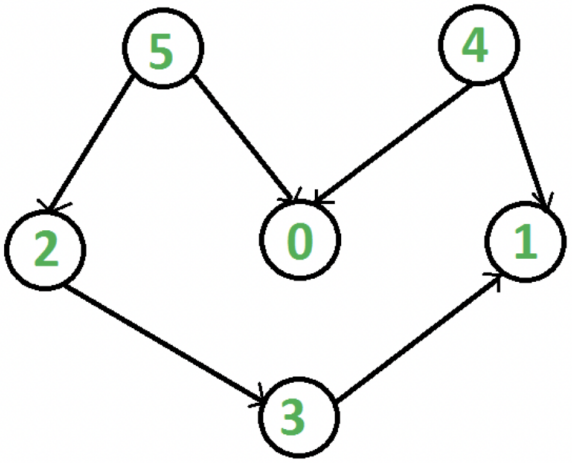
| No. | Graph | Tree |
|---|---|---|
| 1 | Graph is a non-linear data structure. | Tree is a non-linear data structure. |
| 2 | It is a collection of vertices/nodes and edges. | It is a collection of nodes and edges. |
| 3 | Each node can have any number of edges. | General trees consist of the nodes having any number of child nodes. But in case of binary trees every node can have at the most two child nodes. |
| 4 | There is no unique node called root in graph. | There is a unique node called root in trees. |
| 5 | A cycle can be formed. | There will not be any cycle. |

Graphs are used to represent many real-life applications:

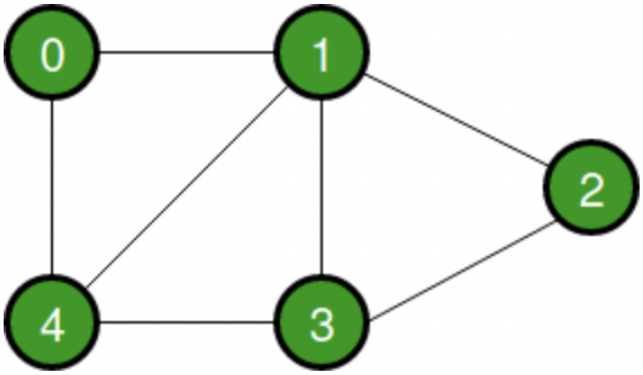
- Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. For example Google GPS
- Graphs are also used in social networks like LinkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, and locale.

**Types of Graphs:**

- **Directed Graphs:** Directed graphs are such graphs in which edges are directed in a single direction. Edge(u, v) is not the same as Edge(v, u).



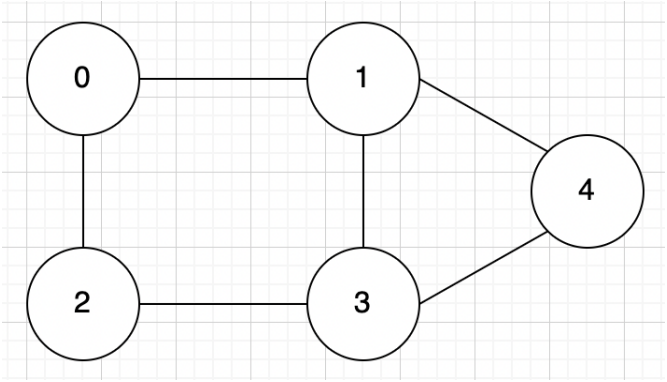

- **Undirected Graphs:** Undirected graphs are such graphs in which the edges are directionless or in other words bi-directional. That is, if there is an edge between vertices u and v then it means we can use the edge to go from both u to v and v to u.



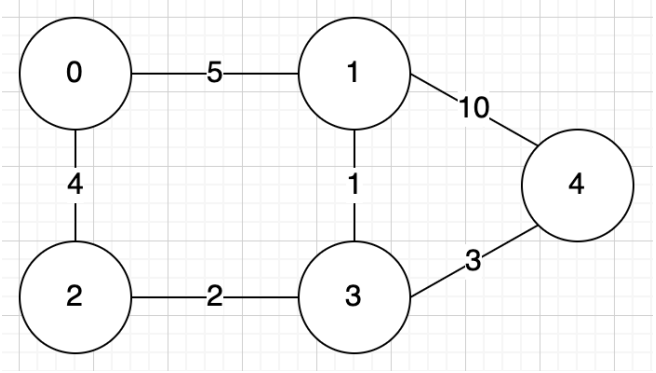
- Weighted and Unweighted graphs: If the edges have weights, then they are called weighted graphs. A weighted graph has a numerical value associated with each edge. On the other hand, if the edges do not have weights, the graph is called unweighted.

Based on the above types, we may have 4 different kinds of graphs:

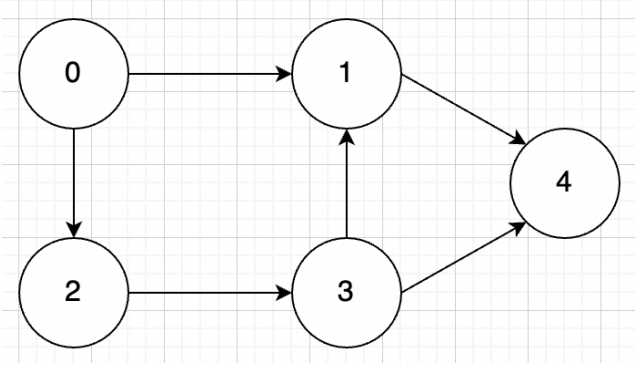
- Undirected + unweighted



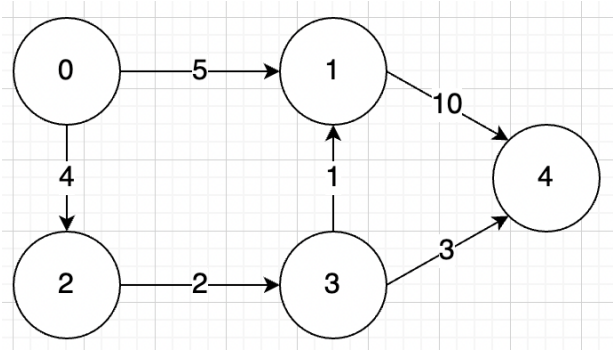

- Undirected + weighted




- Directed + unweighted



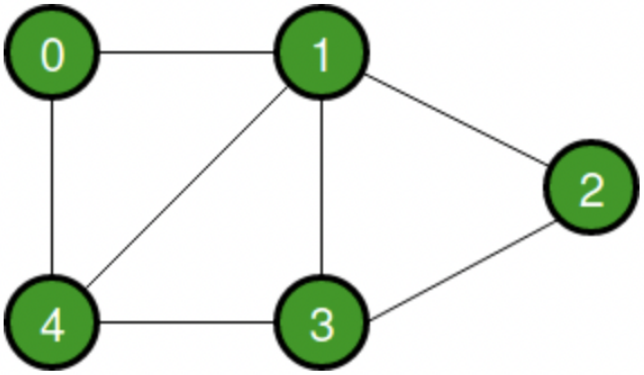
- Directed + weighted



**Representation of Graphs**

**Adjacency Matrix:** The Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph.
Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. The adjacency matrix for the undirected graph is always symmetric.
Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w (where w != 0), then there is an edge from vertex i to vertex j with weight w.

If we have the following undirected graph:




Then its adjacency matrix will look like below:

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

Advantages:

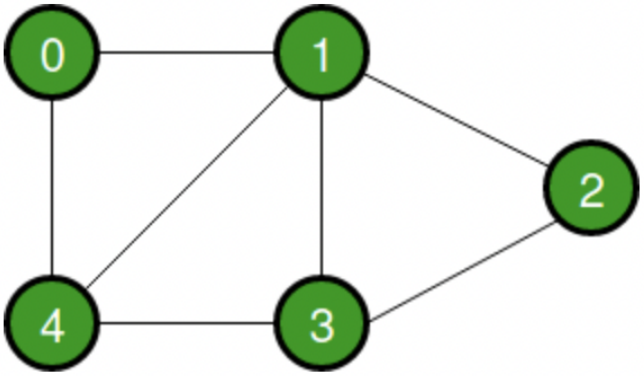
- Adding and removing edges takes O(1) time.
- Queries like checking whether an edge from u to v exists or not, take O(1) time.

Disadvantage:

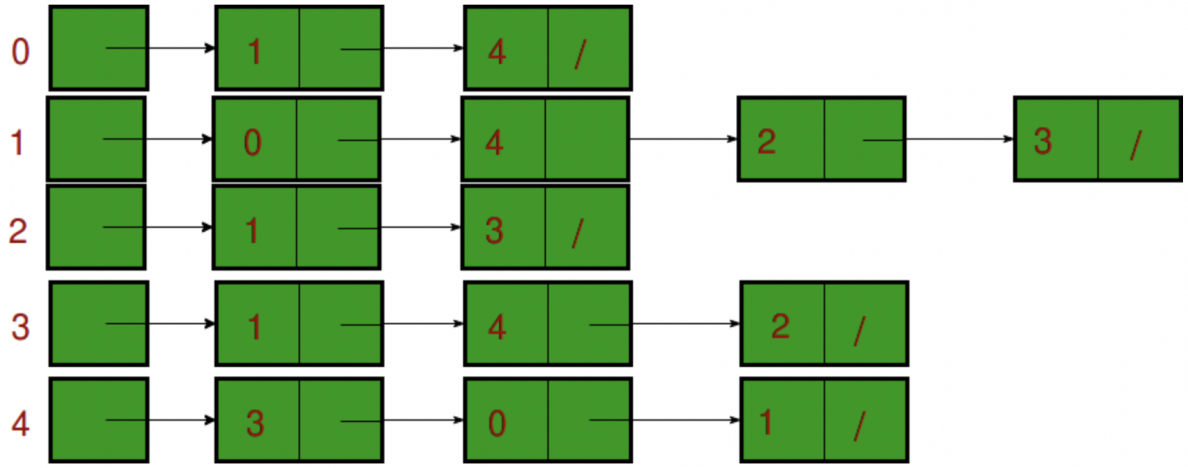
- Very space inefficient, especially in the case of sparse graphs (sparse graph is the one having a very small number of edges).
- The addition of a new vertex in the graph takes $O(V^2)$ time.

**Adjacency List:** Graph can also be implemented using an array of lists. That is every index of the array will contain a complete list. The size of the array is equal to the number of vertices and every index i in the array will store the list of vertices connected to the vertex numbered i. Let the array be adj[]. An entry adj[i] represents the list of vertices adjacent to the ith vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs.

If we have the following undirected graph:




Then its adjacency list will be:



Advantages:

- Saves space, especially in the case of sparse graphs. The overall space occupied is always equal to O(V + E) where V is the number of vertices and E is the number of edges.
- The addition of any new edge takes O(1) time.
- Vertex addition is easy.

Disadvantages:

- Query to check whether an edge from u to v exists or not, take O(V) time.

**DFS Traversal**

The Depth-First Traversal or the DFS traversal of a Graph is used to traverse a graph depth-wise. That is, in this traversal method, we start traversing the graph from a node and keep on going in the same direction as far as possible. When no nodes are left to be traversed along the current path, backtrack to find a new possible path and repeat this process until all of the nodes are visited.

We can implement the DFS traversal algorithm using a recursive approach. While performing the DFS traversal the graph may contain a cycle and the same node can be visited again, so in order to avoid this, we can keep track of the visited array using an auxiliary array. On each step of the recursion mark that the current vertex is visited and call the recursive function again for all the adjacent vertices.

## Finding the number of islands:

Given a grid consisting of '0's(Water) and '1's(Land). Find the number of islands.
**Note:** An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically or diagonally i.e., in all 8 directions.

**Input:**
grid = {{0,1,1,1,0,0,0},{0,0,1,1,0,1,0}}
**Output:**
2
**Expanation:**
The grid is-
0 1 1 1 0 0 0
0 0 1 1 0 1 0
There are two islands one is colored in blue and other in orange.

**BFS Traversal**

The Breadth First Traversal or BFS traversal of a graph is similar to that of the Level Order Traversal of Trees.

The BFS traversal of Graphs also traverses the graph in levels. It starts the traversal with a given vertex, visits all of the vertices adjacent to the initially given vertex, and pushes them all to a queue in order of visiting. Then it pops an element from the front of the queue, visits all of its neighbors and pushes the neighbors which are not already visited into the queue, and repeats the process until the queue is empty or all of the vertices are visited.

The BFS traversal uses an auxiliary boolean array say visited[] which keeps track of the visited vertices. That is if visited[i] = true then it means that the i-th vertex is already visited.