Introduction to graphs

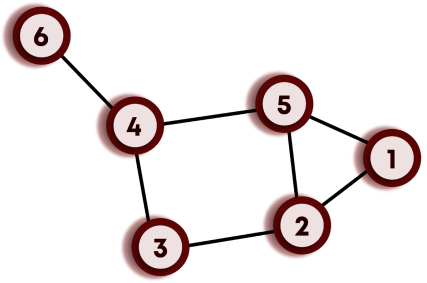
A **graph G** is defined as an ordered set ***(V, E)***, where ***V(G)*** represents the set of ***vertices***, and ***E(G)*** represents the ***edges*** that connect these vertices.

The vertices *x* and *y* of an edge ***{x, y}*** are called the ***endpoints*** of the edge. The edge is said to ***join*** *x* and *y* and to be ***incident*** on ***x*** and ***y***. A vertex may not belong to any edge.

For example: Suppose there is a road network in a country, where we have many cities, and roads are connecting these cities. There could be some cities that are not connected by some other cities like an island. This structure seems to be non-uniform, and hence, we can't use trees to store it. In such cases, we will be using graphs. Refer to the figure below for a better understanding.

V(G) = {1,2,3,4,5,6}

E(G) = {(1,2), (1,5), (2,5), (5,4), (2,3), (3,4), (4,6)}



A graph can be **undirected** or **directed**.

- **Undirected Graphs:** In undirected graphs, edges **do not** have any direction associated with them. In other words, if an edge is drawn between nodes A and B, then the nodes can be traversed from A to B as well as from B to A.
- **Directed Graphs:** In directed graphs, edges form an **ordered pair**. In other words, if there is an edge from A to B, then there is a direct path from A to B but not from B to A. The edge (A, B) is said to initiate from node A (also known as an **initial node**) and terminate at node B (**terminal node**).

A graph can be **weighted** or **unweighted**.

- **Unweighted Graphs:** In unweighted graphs, an edge does not contain any weight.
- **Weighted Graphs:** If edges in a graph have weights associated with them, then the graph is said to be weighted.

Relationship between graphs and trees

- A tree is a special type of graph in which we can reach any node from any other node using some path that is unique, unlike the graphs where this condition may or may not hold.
- A tree is an undirected connected graph with **N** vertices and exactly **N-1** edges.
- A tree does not have any cycles in it, while a graph may have cycles.

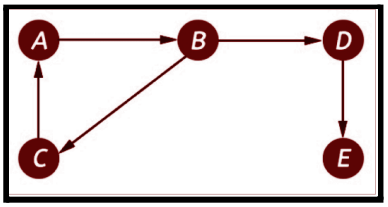
Graph Terminology

- Nodes are called **vertices,** and the connections between them are called **edges**.
- Two vertices are said to be **adjacent** if there exists a direct edge connecting them.
- The **degree** of a node is defined as the number of edges that are incident to it. If the degree of vertex u is 0, it means that u does not belong to any edge and such a node is known as an **isolated vertex.**
- A **loop** is an edge that connects a vertex to itself.
- Distinct edges that connect the same end-points are called **multiple edges**.
- A **path** is a collection of edges through which we can reach from one node to the other node in a graph. A path P is written as P = $\{v_0,v_1,v_2,....,v_n\}$ of length n from a node u to node v, is defined as a sequence of (n+1) nodes. Here u = v_0, v = v_n and v_{i-1} is adjacent to v_i for i = 1,2,3,.....,n.
- A path in which the first and the last vertices are the same forms a **cycle**.
- A graph is said to be **simple**, if it is undirected and unweighted, containing no self-loops and multiple edges.
- A graph with multiple edges and/or self-loops is called a **multi-graph**.
- A graph is said to be **connected** if there is a path between every pair of vertices.
- If the graph is not connected, then all the maximally connected subsets of the graphs are called **connected components**. Each component is connected within itself, but two different components of a graph are never connected.
- The minimum number of edges in a graph can be zero, which means a graph could have no edges as well.
- The minimum number of edges in a connected graph will be **(N-1)**, where **N** is the number of nodes.
- In a **complete graph** (where each node is connected to every other node by a direct edge), there are $^{N}C_{2}$ number of edges means **(N * (N-1)) / 2** edges, where N is the number of nodes, this is the maximum number of edges that a simple graph can have.
- Hence, if an algorithm works on the terms of edges, let's say **O(E)**, where **E** is the number of edges, then in the worst case, the algorithm will take **$O(N^2)$** time, where **N** is the number of nodes.
- **DIRECTED GRAPHS -**
    - The number of edges that originate at a node is the **out-degree** of that node.
    - The number of edges that terminate at a node is the **in-degree** of that node.
    - The **degree** of a node is the sum of the in-degree and out-degree of the node.
    - A digraph or directed graph is said to be **strongly connected** if and only if there exists a path between every pair of nodes in the graph.

- A digraph is said to be a **simple directed graph** if and only if it has no parallel edges. However, a simple directed graph may contain cycles with the exception that it cannot have more than one loop at a given node.
- A digraph is said to be a **directed acyclic graph(DAG)** if the directed graph has no cycles.

Graphs Representation

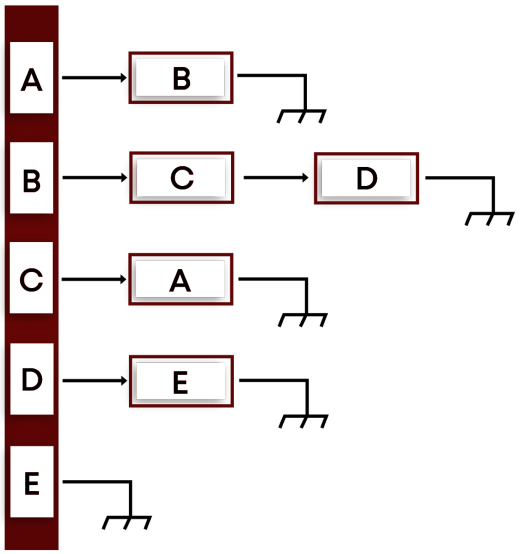
Suppose the graph is as follows:



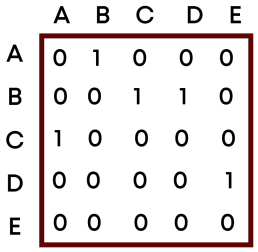
There are the following ways to implement a graph:

- **Using edge list:** We can create a class that could store an array of edges. The array of edges will contain all the pairs that are connected, all put together in one place. It is not preferred to check for a particular edge connecting two nodes; we have to traverse the complete array leading to $O(n^2)$ time complexity in the worst case. Pictorial representation for the above graph using the edge list is given below:

- **Adjacency list:** We will create an array of vertices, but this time, each vertex will have its list of edges connecting this vertex to another vertex. The key advantages of using an adjacency list are:
  - It is easy to follow and clearly shows the adjacent nodes of a particular node
  - It is often used for storing graphs that have a small-to-moderate number of edges. That is, an adjacency list is preferred for representing sparse graphs in the computer's memory; otherwise, an adjacency matrix is a good choice.
  - Adding new nodes in G is easy and straightforward when G is represented using an adjacency list.



- **Adjacency matrix:** Here, we will create a 2D array where the cell **(i, j)** will denote an edge between **node i** and **node j**. The major disadvantage of using the adjacency matrix



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

is vast space consumption

compared to the adjacency list, where each node stores only those nodes that are directly connected to them. For the above graph, the adjacency matrix looks as follows:

  - For a **simple graph** (that has no loops), the adjacency matrix has 0s on the diagonal
  - The adjacency matrix of an **undirected graph** is symmetric.
  - The memory use of an adjacency matrix for **n** nodes is **O(n^2)**, where **n** is the number of nodes in the graph.
  - The adjacency matrix for a **weighted graph** contains the weights of the edges connecting the nodes.