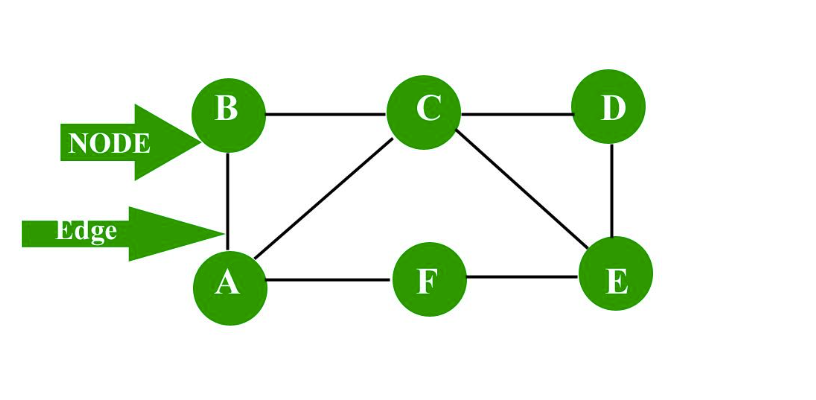
**Graph Theory**

A Graph is just a way to show connections between things. It is set of edges and vertices where each edge is associated with unordered pair of vertices. Graph is a data structure that is defined by two components:

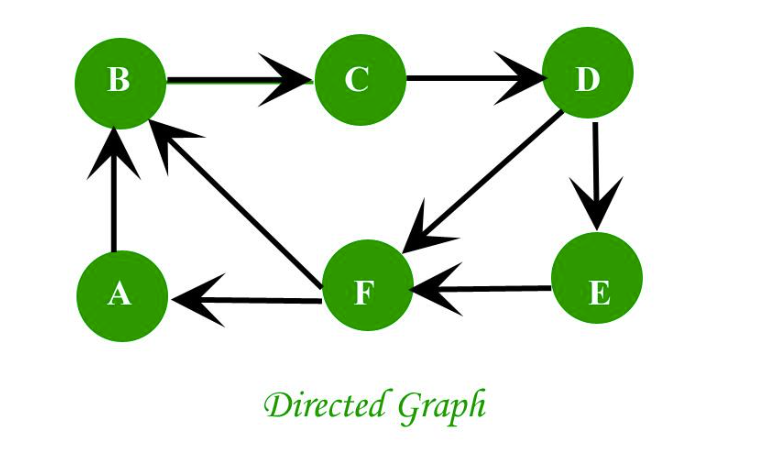
1. **Node or Vertex:** It is a point or joint between two lines like people, cities, or websites. In below diagram the nodes are A, B, C, D, E, F.
2. **Edge:** It is line or connection between two nodes like connections between them (friendships, roads, links). In the below diagram edges are the connecting lines in between them.



**Ordered Pair:**Ordered pair is a connection between two nodes u and v which is identified by unique pair (u, v). The pair (u, v) is ordered because (u, v) is not same as (v, u). It is used in case of directed graph to show which vertex is directing to which vertex.

**Directed Graph**

A graph in which the direction of the edge is defined to a particular node is a directed graph.

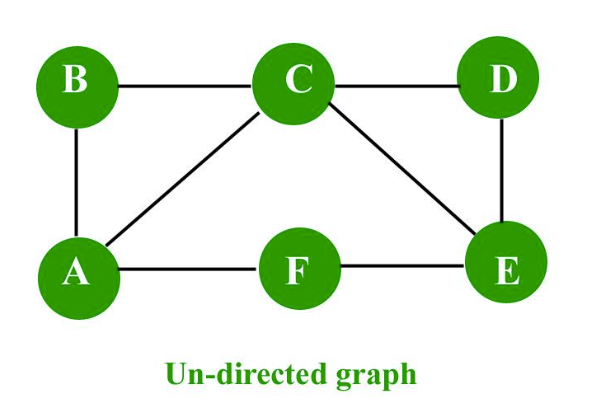


1. **Directed Acyclic graph:**It is a directed graph with no cycle. For a vertex ‘v’ in DAG there is no directed edge starting and ending with vertex ‘v’. The arrows go in one direction only (Directed) and You cannot go in a circle or loop (Acyclic).
2. **Tree:**A tree is just a restricted form of graph. That is, it is a DAG with a restriction that a child can have only one parent.

**Unordered Pair:**In this(u, v) that is identified by unique pair (u, v) can be identified as (v, u). In this the order does not matter in which they come, they are treated same. Undirected graphs are its common example.

**Undirected Graph**

A graph in which the direction of the edge is not defined. So, if an edge exists between node ‘u’ and ‘v’, then there is a path from node ‘u’ to ‘v’ and vice-versa.



**Weighted Graph:**It is a graph (directed or undirected) in which each edge is assigned some numerical value. This value is called a weight. These weights often represent costs, distances, capacities, or other quantifiable relationships between vertices.

**Unweighted Graph:**It is a graph in which edges do not have any weight assigned. In this graph all the edges are treated equally or given equal priority. There are only two possibilities for edges, either an edge exists or it does not.

**Applications**

Graph is a data structure which is used extensively in our real-life like examples below:

1. **Social Network:** Each user is represented as a node and all their activities, suggestion and friend list are represented as an edge between the nodes.
2. **Google Maps:** Various locations are represented as vertices or nodes and the roads are represented as edges and graph theory is used to find shortest path between two nodes.
3. **Recommendations on e-commerce websites:**The “Recommendations for you” section on various e-commerce websites uses graph theory to recommend items of similar type to user’s choice.

**BASIC TERMINOLOGY**

1. **Adjacent Node:**A node ‘v’ is said to be adjacent node of node ‘u’ if and only if there exists an edge between ‘u’ and ‘v’.
2. **Degree of a Node:**In an undirected graph the number of edges incident on a node is the degree of the node.
3. In case of directed graph:
   * Indegree of the node is the number of arriving edges to a node.
   * Outdegree of the node is the number of departing edges to a node.
4. **Self-Loop:**When an edge in graph connects a vertex to itself it is called self-loop. This edge starts and ends at same vertex. A self-loop is counted twice in case of degree of a node.
5. **Isolated Node:** A node with degree 0 is known as isolated node. Isolated node can be found by Breadth first search (BFS). It finds its application in LAN network in finding whether a system is connected or not.
6. **Connected graph:** A graph is connected when there is a path between every pair of vertices. In a connected graph there is no unreachable node.

***Travel Use Case:* Optimal Flight Route Planning for a Travel Booking Platform**

***Business Context: A travel company like Expedia wants to help users find the cheapest and/or fastest route between any two cities using flights, even when there are layovers or connecting flights.***

**Model as a Graph:**

**Cities = Nodes (Vertices)**

**Flights between cities = Edges**

**Edge weight can represent:**

**Cost of the flight 💵**

**Travel time ⏱**

**Number of stops**

**Help the user find:**

**Shortest path (minimum travel time)**

**Cheapest path (minimum cost)**

**Path with fewest layovers**

**Or even multi-city travel plans**

**Suppose a user wants to travel from New York to Bangkok.**

**There are multiple paths:**

1. **NYC → LON → BKK**
2. **NYC → DXB → SIN → BKK**
3. **NYC → SIN → BKK**

**You can model this using a weighted graph, then apply:**

* **Dijkstra's Algorithm to find the cheapest or shortest path**
* **Depth-First Search (DFS) for exploring all possible itineraries**

**Benefits:**

Travel platforms **optimize user routes**

Customers get **time-saving or budget-friendly options**

Airlines can **analyze their route networks** for weak spots or improvements