**Graphs:**

A graph data structure consists of a finite (and possibly mutable) set of vertices (also called nodes or points), together with a set of unordered pairs of these vertices for an undirected graph or a set of ordered pairs for a directed graph.

Graphs are mathematical structures that reflect the pairwise relationship between things. A graph is a type of flow structure that displays the interactions of several objects. It may be represented by utilizing the two fundamental components, nodes and edges.

* **Nodes**: These are the most crucial elements of every graph. Edges are used to represent node connections. For example, a graph with two nodes connected using an undirected edge shows a bi-directional connection between those two nodes.
* **Edges**: Edges are part of a graph showing the connections between nodes. An edge represents the connection between two nodes.

**An application in real life**

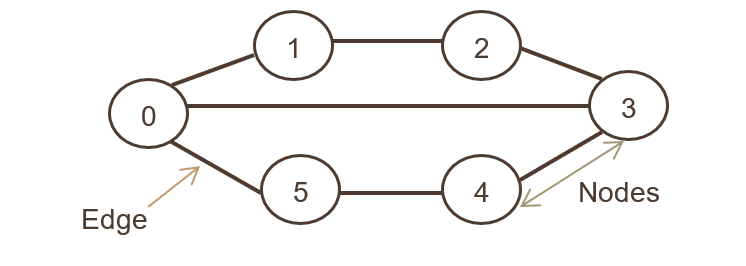
* **Google Maps**: link your journey from the start to the end.
* **Social Networks**: friends are connected with each other using an edge where each user represents a vertex.
* **Recommendation System**: relationship data between user’s recommendations uses graphs for connection.

A data structure is an efficient way of organising data in a database so that that data can be accessed easily and used effectively. There are many types of databases, but why graphs play a vital role in data management is discussed in this article.

Spoiler alert: you use **Graphs in data structure** every day to fetch the best route to your office, to get suggestions for your lunch, movie and to optimise your next flight route. Sounds interesting! Let us see about the graph’s properties and its application.

First, let’s see **what a Graph is?**It is a representation of data in a non-linear structure consisting of nodes (or vertices) and edges (or paths).

A**Graph in the data structure** can be termed as a data structure consisting of data that is stored among many groups of edges(paths) and vertices (nodes), which are interconnected. Graph data structure (N, E) is structured with a collection of Nodes and Edges. Both nodes and vertices need to be finite.



In the above graph representation, Set of Nodes are N={0,1,2,3,4,5,6}and set of edges are

G={01,12,23,34,45,05,03}

Now let’s study the types of graphs.

## Types of Graphs

### Weighted Graph

Graphs whose edges or paths have values. All the values seen associated with the edges are called weights. Edges value can represent weight/cost/length.

Values or weights may also represent:

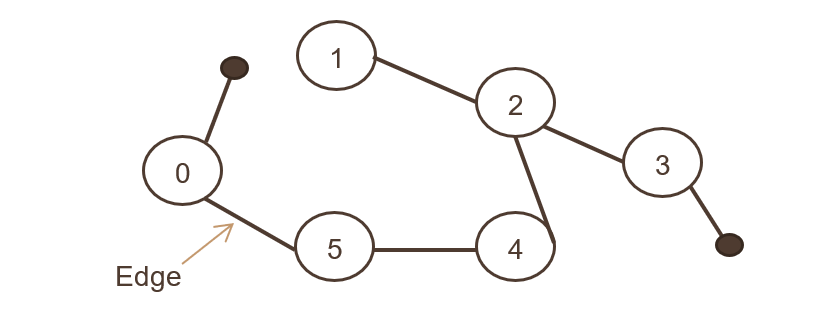
Distance covered between two points- Ex: To look for that shortest path to the office, the distance between two workstations in an office network.

Speed of the data packet in a network or bandwidth.

**2. Unweighted Graph**

Where there is no value or weight associated with the edge. By default, all the graphs are unweighted unless there is a value associated.

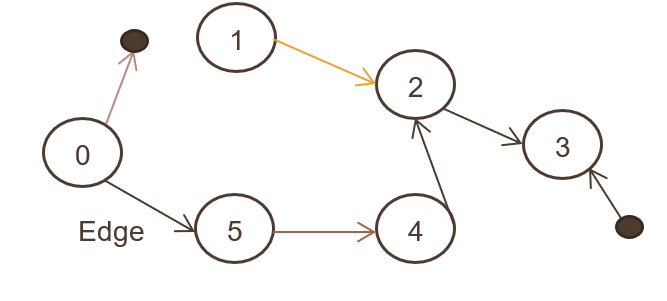
**3. Undirected Graph**



Where a set of objects are connected, and all the edges are bidirectional. The below image showcases the undirected graph,

It’s like the associativity of two Facebook users after connecting as a friend. Both users can refer and share photos, comment among each other.

**4. Directed Graph**

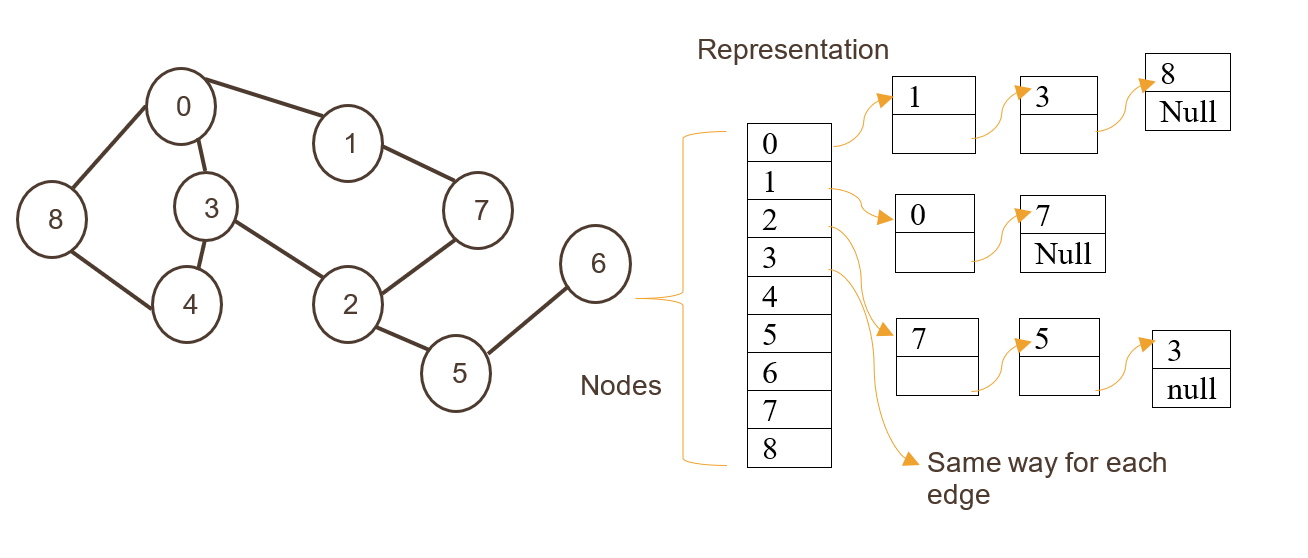


Also called a digraph, where a set of objects (N, E) are connected, and all the edges are directed from one node to another. The above image showcases the directed graph.

## Storing of Graph

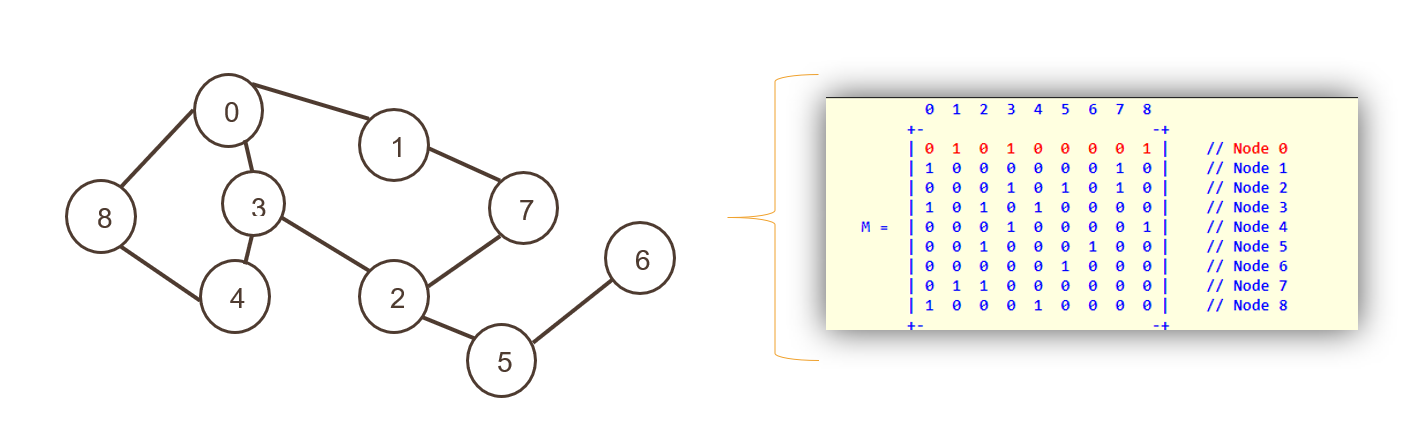
Every storage method has its pros and cons, and the right storage method is chosen based on the complexity. The two most commonly used data structures to store graphs are:

### 1. Adjacency list



Here nodes are stored as an index of the one-dimension array followed by edges being stored as a list.

**2. Adjacency matrix**



Here nodes are represented as the index of a two-dimensional array, followed by edges represented as non-zero values of an adjacent matrix.

Both rows and columns showcase Nodes; the entire matrix is filled with either “0” or “1”, representing true or false. Zero represents that there is no path, and 1 represents a path.

## Graph Traversal

Graph traversal is a method used to search nodes in a graph. The graph traversal is used to decide the order used for node arrangement. It also searches for edges without making a loop, which means all the nodes and edges can be searched without creating a loop.

There are two graph traversal structures.

There are two graph traversal structures.

**DFS (Depth First Search): In-depth search method**

The DFS search begins starting from the first node and goes deeper and deeper, exploring down until the targeted node is found. If the targeted key is not found, the search path is changed to the path that was stopped exploring during the initial search, and the same procedure is repeated for that branch.

The spanning tree is produced from the result of this search. This tree method is without the loops. The total number of nodes in the stack data structure is used to implement DFS traversal.

Steps followed to implement DFS search:

Step 1 – Stack size needs to be defined depending on the total number of nodes.

Step 2 – Select the initial node for transversal; it needs to be pushed to the stack by visiting that node.

Step 3 – Now, visit the adjacent node that is not visited before and push that to the stack.

Step 4 – Repeat Step 3 until there is no adjacent node that is not visited.

Step 5 – Use backtracking and one node when there are no other nodes to be visited.

Step 6 – Empty the stack by repeating steps 3,4, and 5.

Step 7 – When the stack is empty, a final spanning tree is formed by eliminating unused edges.

**Applications of DFS are:**

Solving puzzles with only one solution.

To test if a graph is bipartite.

Topological Sorting for scheduling the job and many others.

**2. BFS (Breadth-First Search): Search is implemented using a queuing method**

Breadth-First Search navigates a graph in a breadth motion and utilises based on the Queue to jump from one node to another, after encountering an end in the path.

Steps followed to implement BFS search,

Step 1 – Based on the number of nodes, the Queue is defined.

Step 2 – Start from any node of the traversal. Visit that node and add it to the Queue.

Step 3 – Now check the non-visited adjacent node, which is in front of the Queue, and add that into the Queue, not to the start.

Step 4 – Now start deleting the node that doesn’t have any edges that need to be visited and is not in the Queue.

Step 5 – Empty the Queue by repeating steps 4 and 5.

Step 6 – Remove the unused edges and form the spanning tree only after the Queue is empty.

**Applications of BFS are:**

* Peer to Peer Networks- Like in Bittorrent, it is used to find all adjacent nodes.
* Crawlers in Search Engine.
* Social Networking Websites and many more.