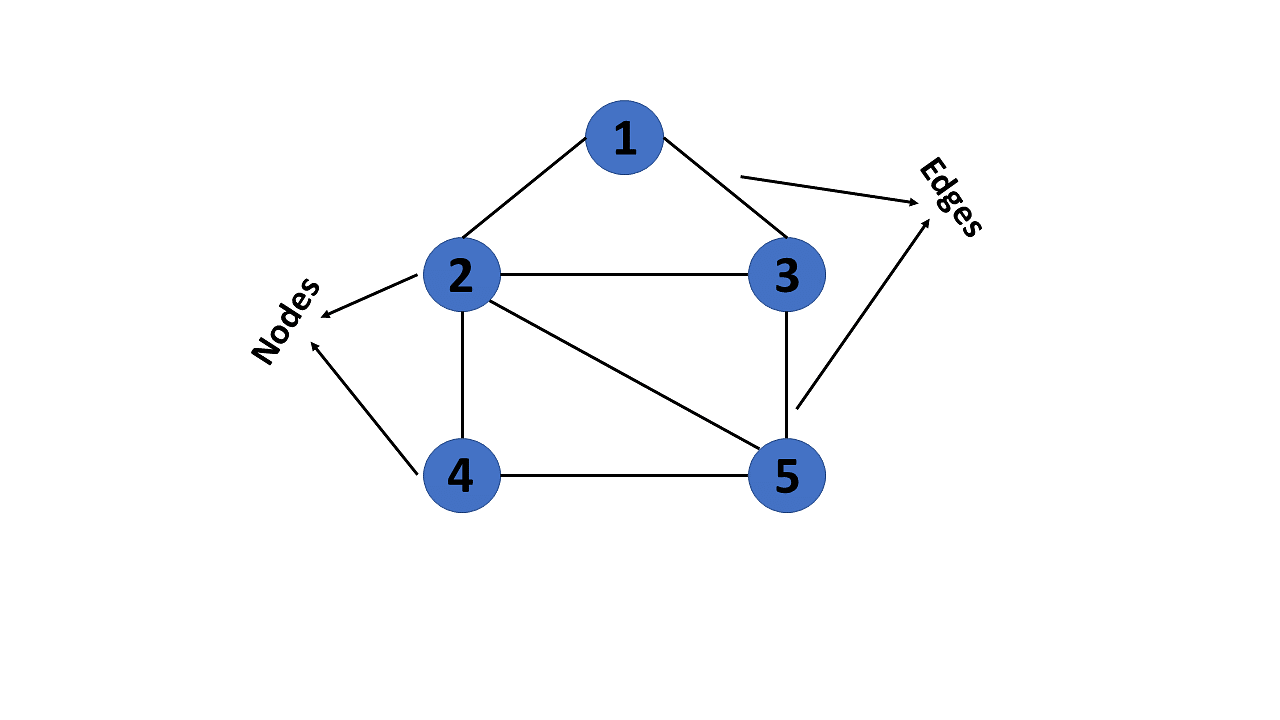
Graphs in data structures are non-linear [data structures](https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-data-structure) made up of a finite number of nodes or vertices and the edges that connect them. Graphs in data structures are used to address real-world problems in which it represents the problem area as a network like telephone networks, circuit networks, and social networks. For example, it can represent a single user as nodes or vertices in a telephone network, while the link between them via telephone represents edges.

What Are Graphs in Data Structure?

A graph is a non-linear kind of data structure made up of nodes or vertices and edges. The edges connect any two nodes in the graph, and the nodes are also known as vertices.



This graph has a set of vertices V= { 1,2,3,4,5} and a set of edges E= { (1,2),(1,3),(2,3),(2,4),(2,5),(3,5),(4,50 }.

Now that you’ve learned about the definition of graphs in data structures, you will learn about their various types.

Types of Graphs in Data Structures

There are different types of graphs in data structures, each of which is detailed below.

1. Finite Graph

The graph G=(V, E) is called a finite graph if the number of vertices and edges in the graph is limited in number



2. Infinite Graph

The graph G=(V, E) is called a finite graph if the number of vertices and edges in the graph is interminable.



3. Trivial Graph

A graph G= (V, E) is trivial if it contains only a single vertex and no edges.



4. Simple Graph

If each pair of nodes or vertices in a graph G=(V, E) has only one edge, it is a simple graph. As a result, there is just one edge linking two vertices, depicting one-to-one interactions between two elements.



5. Multi Graph

If there are numerous edges between a pair of vertices in a graph G= (V, E), the graph is referred to as a multigraph. There are no self-loops in a Multigraph.



6. Null Graph

It's a reworked version of a trivial graph. If several vertices but no edges connect them, a graph G= (V, E) is a null graph.



7. Complete Graph

If a graph G= (V, E) is also a simple graph, it is complete. Using the edges, with n number of vertices must be connected. It's also known as a full graph because each vertex's degree must be n-1.



8. Pseudo Graph

If a graph G= (V, E) contains a self-loop besides other edges, it is a pseudograph.



9. Regular Graph

If a graph G= (V, E) is a simple graph with the same degree at each vertex, it is a regular graph. As a result, every whole graph is a regular graph.



10. Weighted Graph

A graph G= (V, E) is called a labeled or weighted graph because each edge has a value or weight representing the cost of traversing that edge.



11. Directed GraphA directed graph also referred to as a digraph, is a set of nodes connected by edges, each with a direction.

12. Undirected Graph

An undirected graph comprises a set of nodes and links connecting them. The order of the two connected vertices is irrelevant and has no direction. You can form an undirected graph with a finite number of vertices and edges.



13. Connected Graph

If there is a path between one vertex of a graph data structure and any other vertex, the graph is connected.



14. Disconnected Graph

When there is no edge linking the vertices, you refer to the null graph as a disconnected graph.



15. Cyclic Graph

If a graph contains at least one graph cycle, it is considered to be cyclic.



16. Acyclic Graph

When there are no cycles in a graph, it is called an acyclic graph.



17. Directed Acyclic Graph

It's also known as a directed acyclic graph (DAG), and it's a graph with directed edges but no cycle. It represents the edges using an ordered pair of vertices since it directs the vertices and stores some data.



18. Subgraph

The vertices and edges of a graph that are subsets of another graph are known as a subgraph.



After you learn about the many types of graphs in graphs in data structures, you will move on to graph terminologies.

Terminologies of Graphs in Data Structures

Following are the basic terminologies of graphs in data structures:

* An edge is one of the two primary units used to form graphs. Each edge has two ends, which are vertices to which it is attached.
* If two vertices are endpoints of the same edge, they are adjacent.
* A vertex's outgoing edges are directed edges that point to the origin.
* A vertex's incoming edges are directed edges that point to the vertex's destination.
* The total number of edges occurring to a vertex in a graph is its degree.
* The out-degree of a vertex in a directed graph is the total number of outgoing edges, whereas the in-degree is the total number of incoming edges.
* A vertex with an in-degree of zero is referred to as a source vertex, while one with an out-degree of zero is known as sink vertex.
* An isolated vertex is a zero-degree vertex that is not an edge's endpoint.
* A path is a set of alternating vertices and edges, with each vertex connected by an edge.
* The path that starts and finishes at the same vertex is known as a cycle.
* A path with unique vertices is called a simple path.
* For each pair of vertices x, y, a graph is strongly connected if it contains a directed path from x to y and a directed path from y to x.
* A directed graph is weakly connected if all of its directed edges are replaced with undirected edges, resulting in a connected graph. A weakly linked graph's vertices have at least one out-degree or in-degree.
* A tree is a connected forest. The primary form of the tree is called a rooted tree, which is a free tree.
* A spanning subgraph that is also a tree is known as a [spanning tree.](https://www.simplilearn.com/tutorials/data-structure-tutorial/spanning-tree-in-data-structure)
* A connected component is the unconnected graph's most connected subgraph.
* A bridge, which is an edge of removal, would sever the graph.
* Forest is a graph without a cycle.

Following that, you will look at the graph representation in this data structures tutorial.

Representation of Graphs in Data Structures

Graphs in data structures are used to represent the relationships between objects. Every graph consists of a set of points known as vertices or nodes connected by lines known as edges. The vertices in a network represent entities.

The most frequent graph representations are the two that follow:

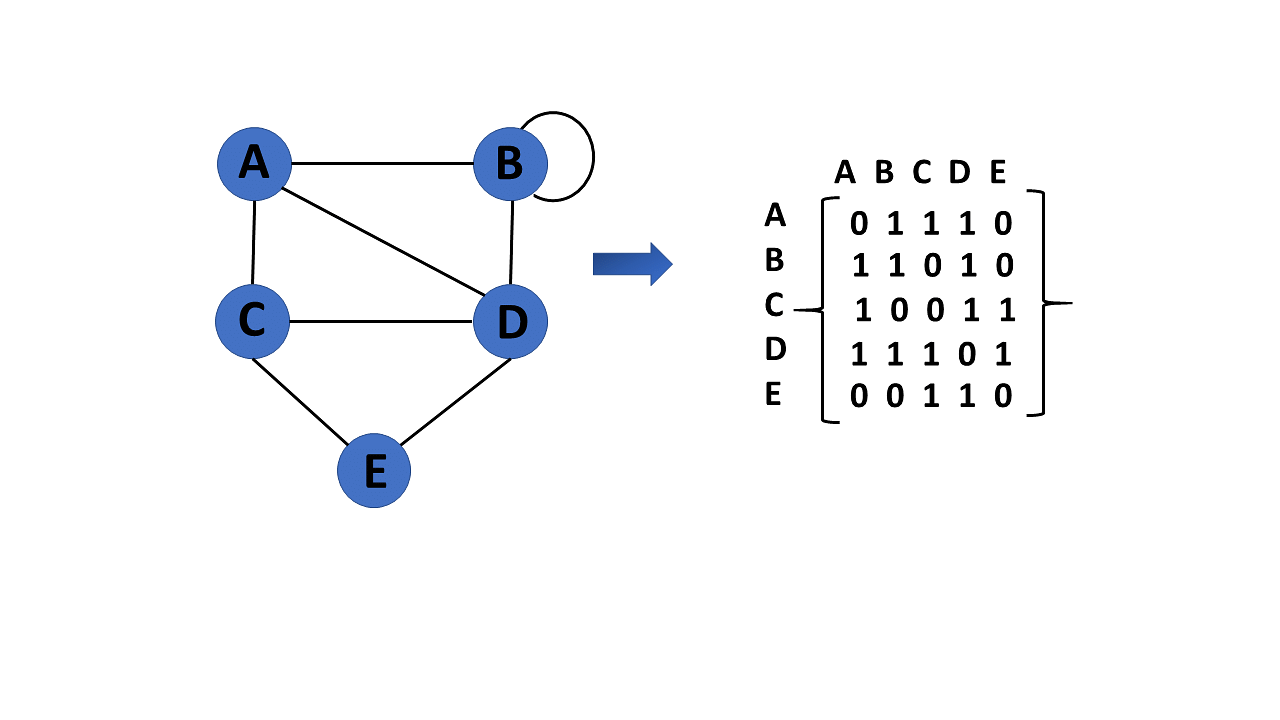
* Adjacency matrix
* Adjacency list

You’ll look at these two representations of graphs in data structures in more detail:

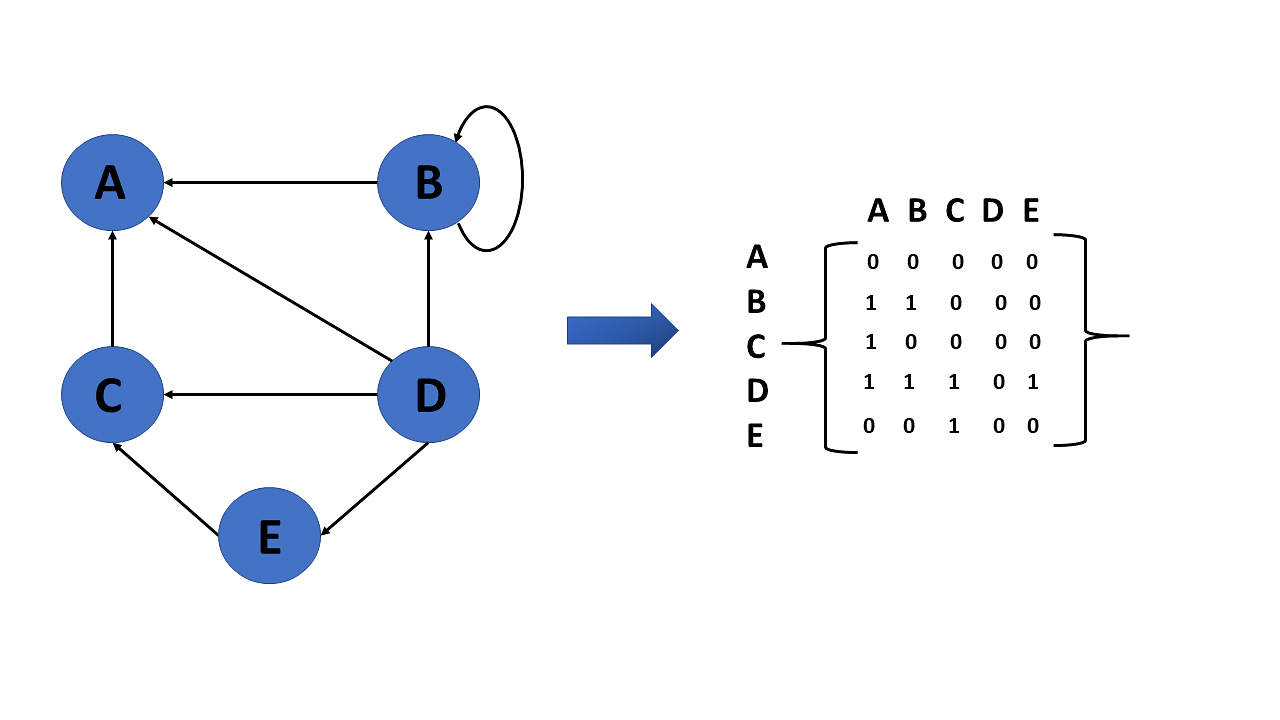
Adjacency Matrix

* A sequential representation is an adjacency matrix.
* It's used to show which nodes are next to one another. I.e., is there any connection between nodes in a graph?
* You create an n \* n matrix G for this representation. If an edge exists between vertex a and vertex b, the corresponding element of G, gi,j = 1, otherwise gi,j = 0.
* If there is a weighted graph, you can record the edge's weight instead of 1s and 0s.

Undirected Graph Representation

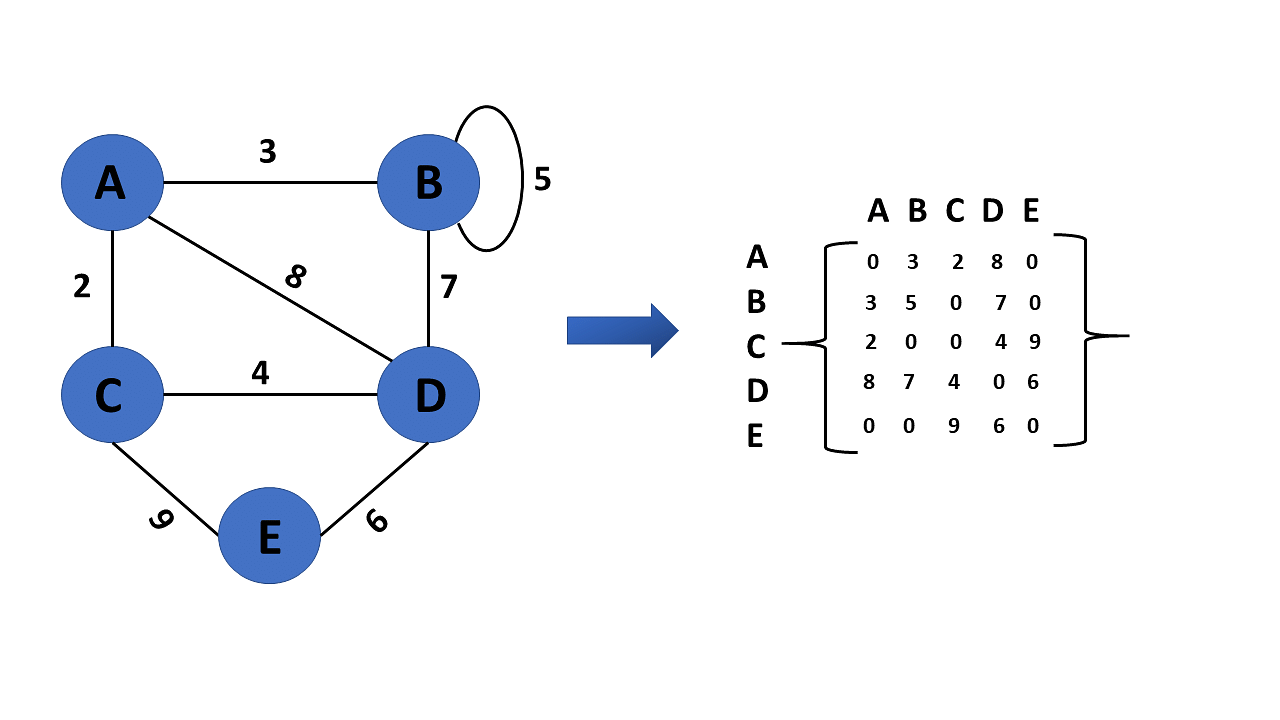


Directed Graph Representation



Weighted Undirected Graph Representation

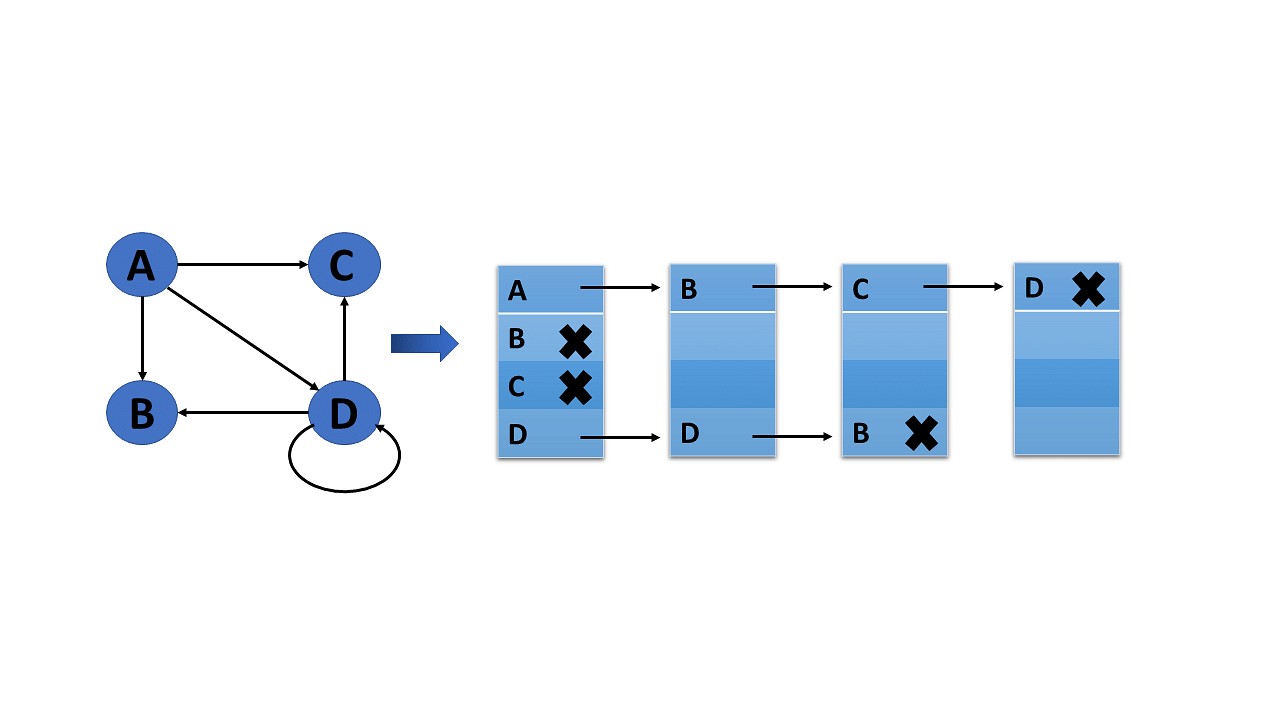
Weight or cost is indicated at the graph's edge, a weighted graph representing these values in the matrix.



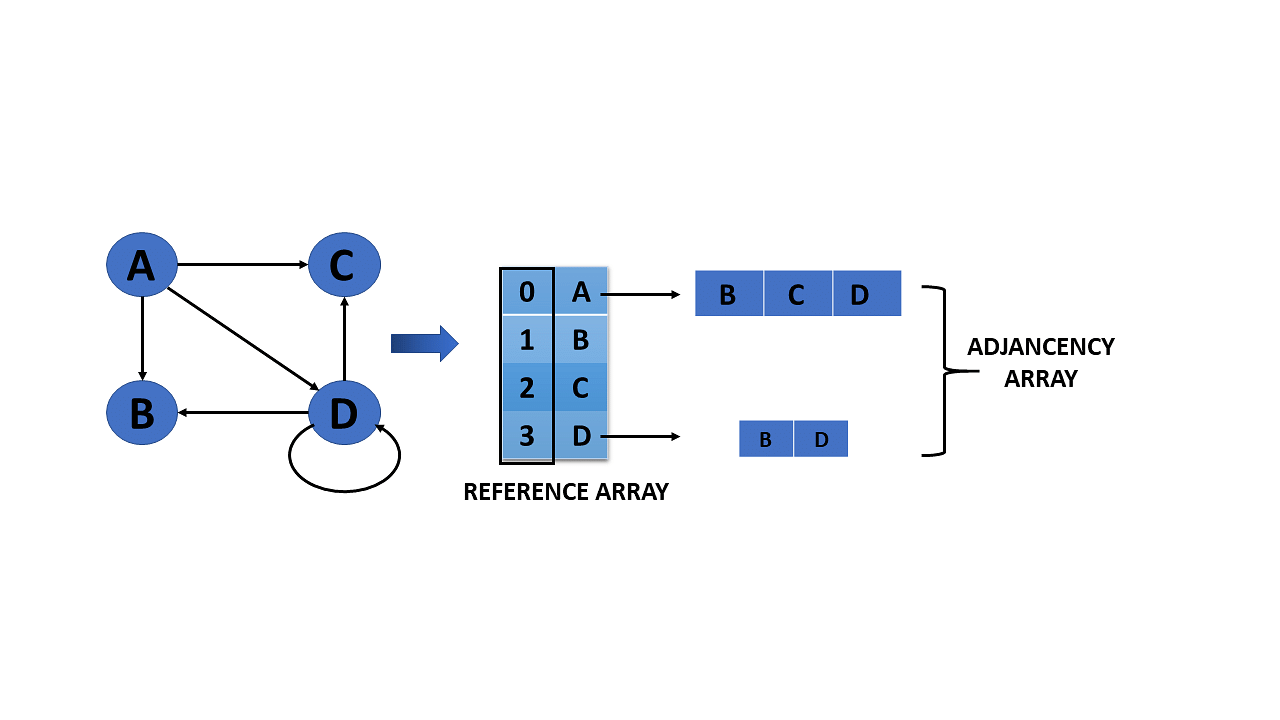
Adjacency List

* A linked representation is an adjacency list.
* You keep a list of neighbors for each vertex in the graph in this representation. It means that each vertex in the graph has a list of its neighboring vertices.
* You have an arra of vertices indexed by the vertex number, and the corresponding [array](https://www.simplilearn.com/tutorials/data-structure-tutorial/arrays-in-data-structure) member for each vertex x points to a [singly linked list](https://www.simplilearn.com/tutorials/data-structure-tutorial/singly-linked-list) of x's neighbors.

Weighted Undirected Graph Representation Using Linked-List



Weighted Undirected Graph Representation Using an Array



Operations on Graphs in Data Structures

The operations you perform on the graphs in data structures are listed below:

* Creating graphs
* Insert vertex
* Delete vertex
* Insert edge
* Delete edge

You will go over each operation in detail one by one:

Creating Graphs

There are two techniques to make a graph:

1. Adjacency Matrix

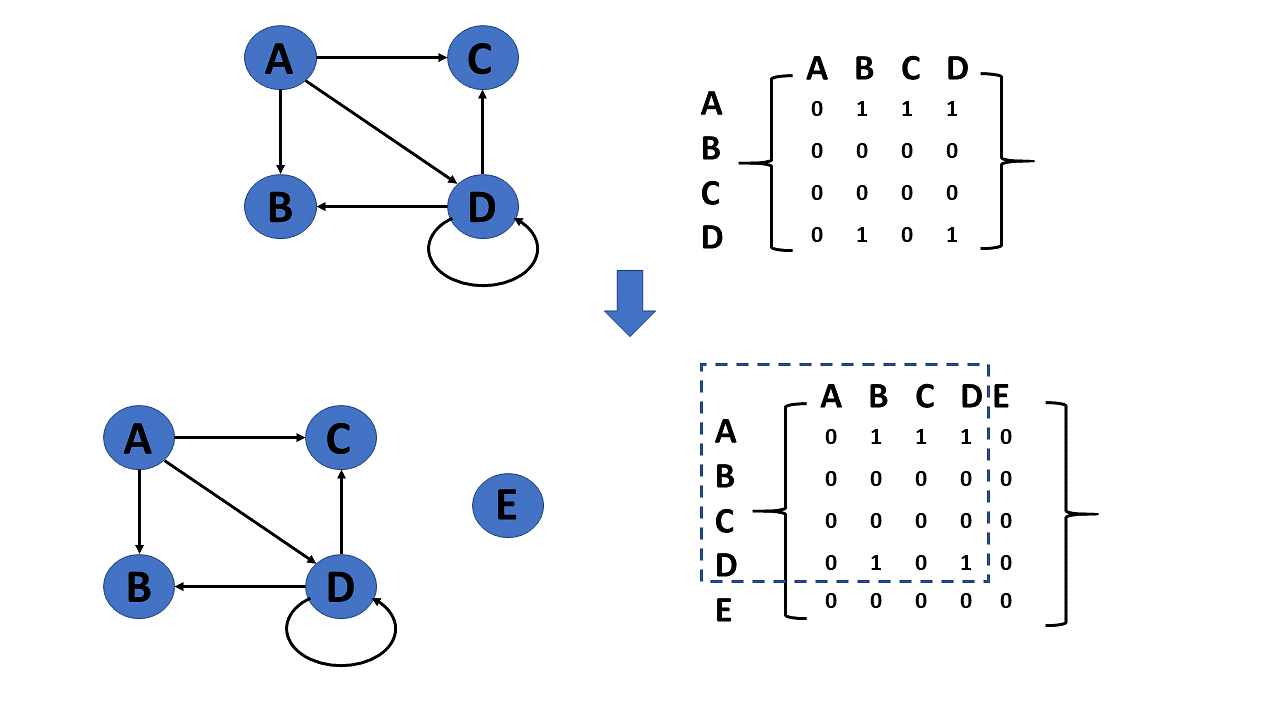
The adjacency matrix of a simple labeled graph, also known as the connection matrix, is a matrix with rows and columns labeled by graph vertices and a 1 or 0 in position depending on whether they are adjacent or not.

2. Adjacency List

A finite graph is represented by an adjacency list, which is a collection of unordered lists. Each unordered list describes the set of neighbors of a particular vertex in the graph within an adjacency list.

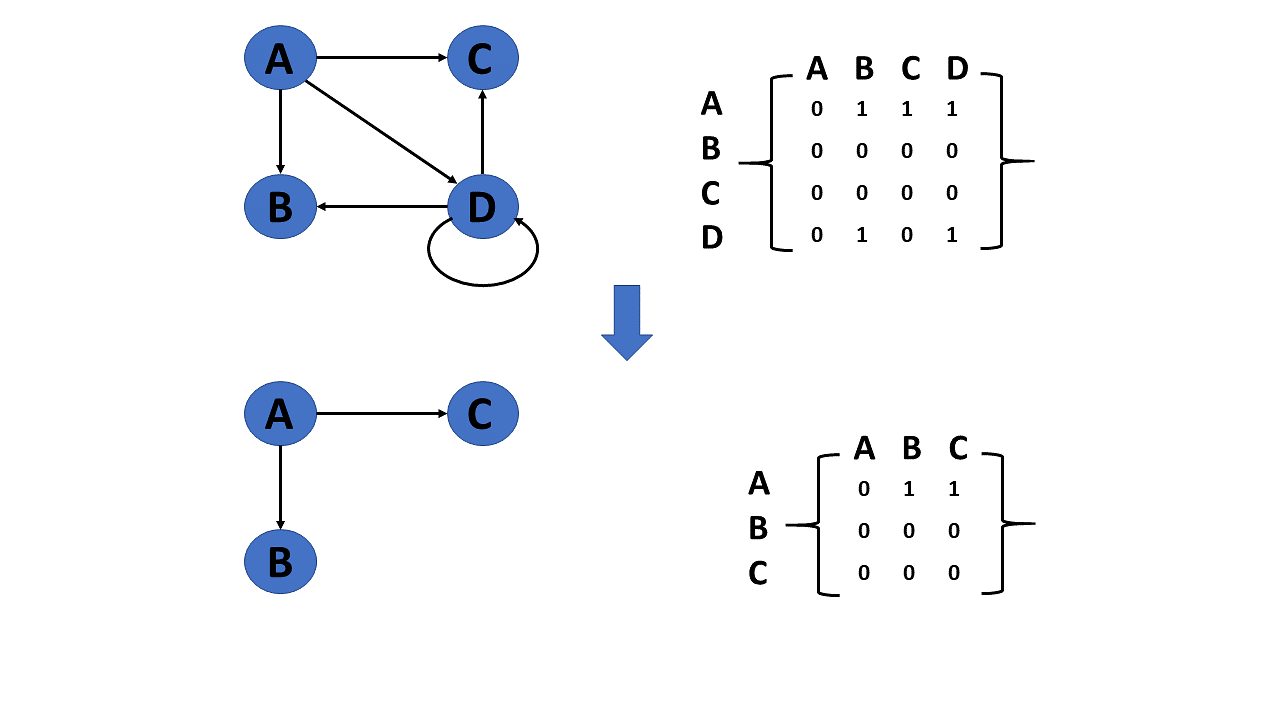
Insert Vertex

When you add a vertex that after introducing one or more vertices or nodes, the graph's size grows by one, increasing the matrix's size by one at the row and column levels.



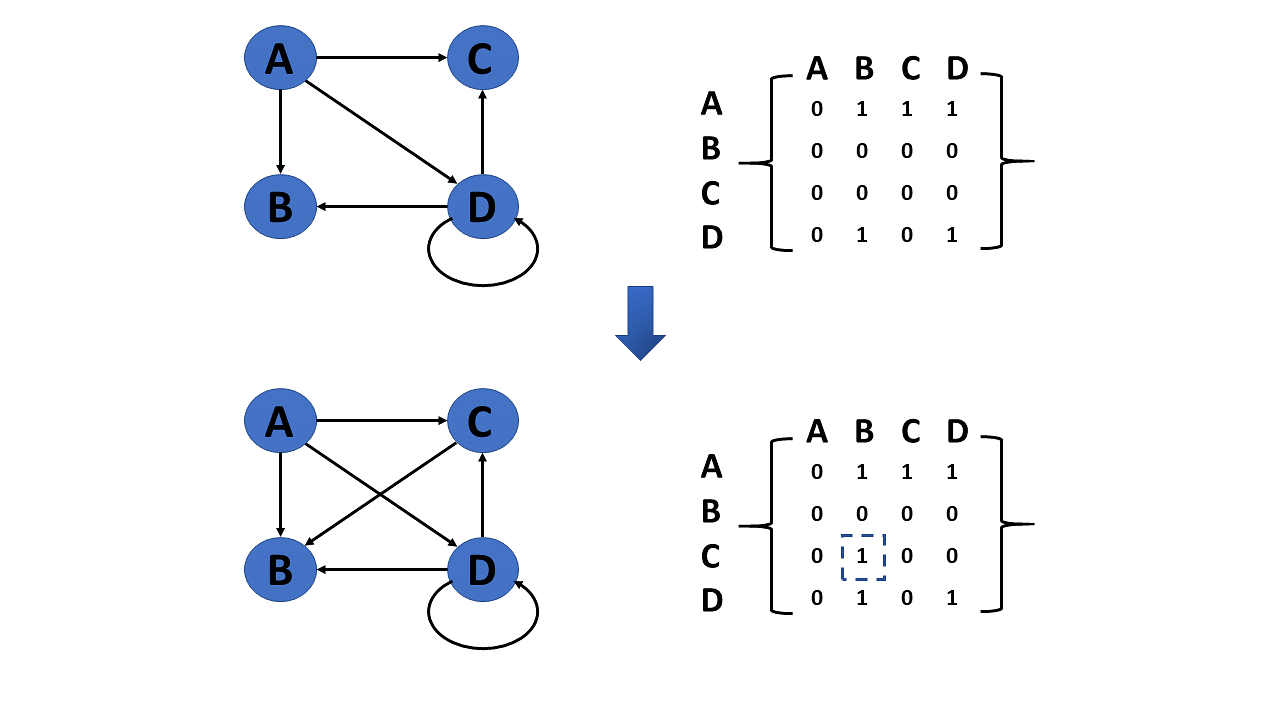
Delete Vertex

* Deleting a vertex refers to removing a specific node or vertex from a graph that has been saved.
* If a removed node appears in the graph, the matrix returns that node. If a deleted node does not appear in the graph, the matrix returns the node not available.



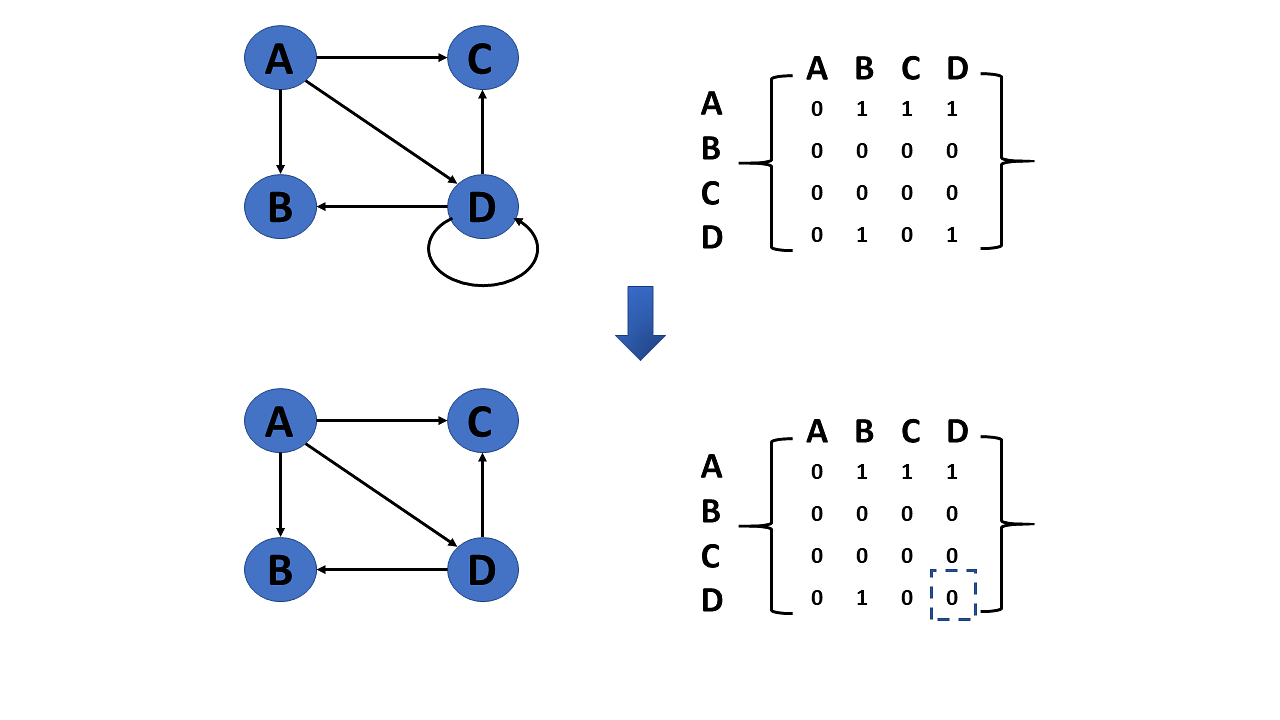
Insert Edge

Connecting two provided vertices can be used to add an edge to a graph.



Delete Edge

The connection between the vertices or nodes can be removed to delete an edge.



The types of graph traversal algorithms will be discussed next in the graphs in this data structures tutorial.

Graph Traversal Algorithm

The process of visiting or updating each vertex in a graph is known as graph traversal. The sequence in which they visit the vertices is used to classify such traversals. Graph traversal is a subset of tree traversal.

There are two techniques to implement a graph traversal algorithm:

* Breadth-first search
* Depth-first search

Breadth-First Search or BFS

[BFS](https://www.simplilearn.com/tutorials/data-structure-tutorial/bfs-algorithm) is a search technique for finding a node in a graph data structure that meets a set of criteria.

* It begins at the root of the graph and investigates all nodes at the current depth level before moving on to nodes at the next depth level.
* To maintain track of the child nodes that have been encountered but not yet inspected, more memory, generally you require a [queue.](https://www.simplilearn.com/tutorials/data-structure-tutorial/queue-in-data-structure)

Algorithm of breadth-first search

Step 1: Consider the graph you want to navigate.

Step 2: Select any vertex in your graph, say v1, from which you want to traverse the graph.

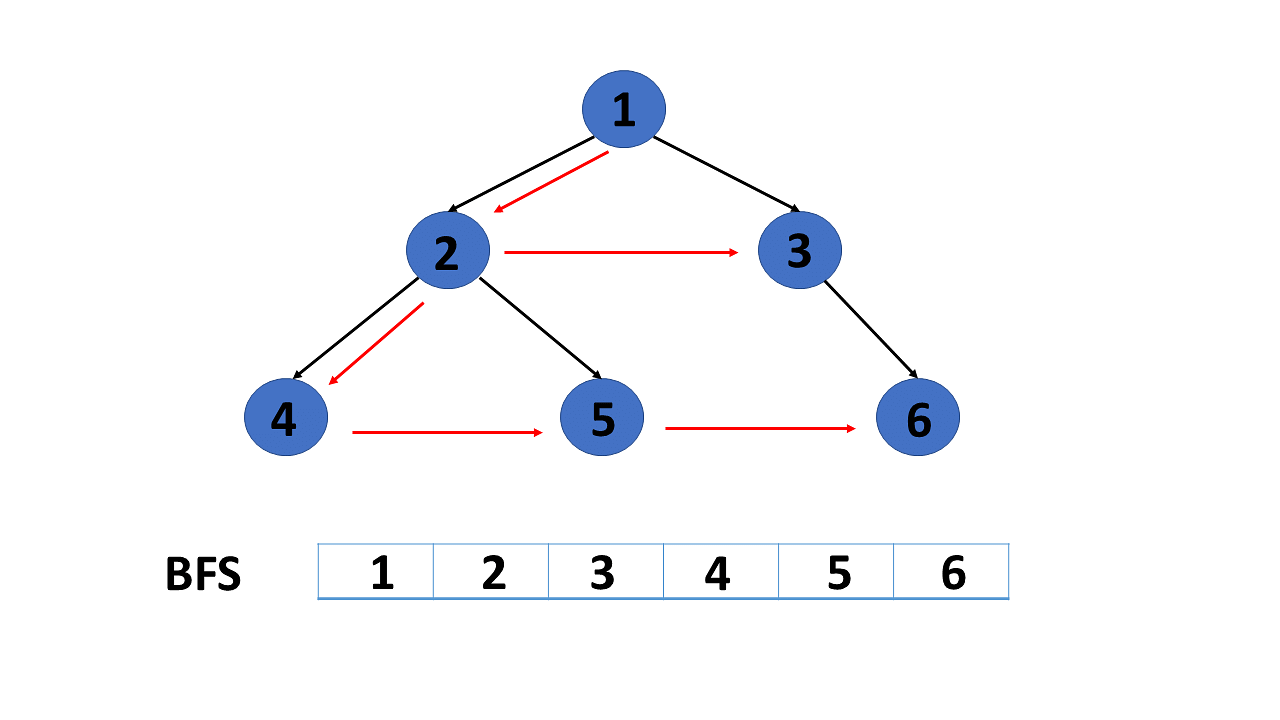
Step 3: Examine any two data structures for traversing the graph.

* Visited array (size of the graph)
* Queue data structure

Step 4: Starting from the vertex, you will add to the visited array, and afterward, you will v1's adjacent vertices to the queue data structure.

Step 5: Now, using the FIFO concept, you must remove the element from the queue, put it into the visited array, and then return to the queue to add the adjacent vertices of the removed element.

Step 6: Repeat step 5 until the queue is not empty and no vertex is left to be visited.



Depth-First Search or DFS

[DFS](https://www.simplilearn.com/tutorials/data-structure-tutorial/dfs-algorithm) is a search technique for finding a node in a graph data structure that meets a set of criteria.

* The depth-first search (DFS) algorithm traverses or explores data structures such as trees and graphs. The DFS algorithm begins at the root node and examines each branch as far as feasible before backtracking.
* To maintain track of the child nodes that have been encountered but not yet inspected, more memory, [generally a stack](https://www.simplilearn.com/tutorials/data-structure-tutorial/stacks-in-data-structures), is required.

Algorithm of depth-first search

Step 1: Consider the graph you want to navigate.

Step 2: Select any vertex in our graph, say v1, from which you want to begin traversing the graph.

Step 3: Examine any two data structures for traversing the graph.

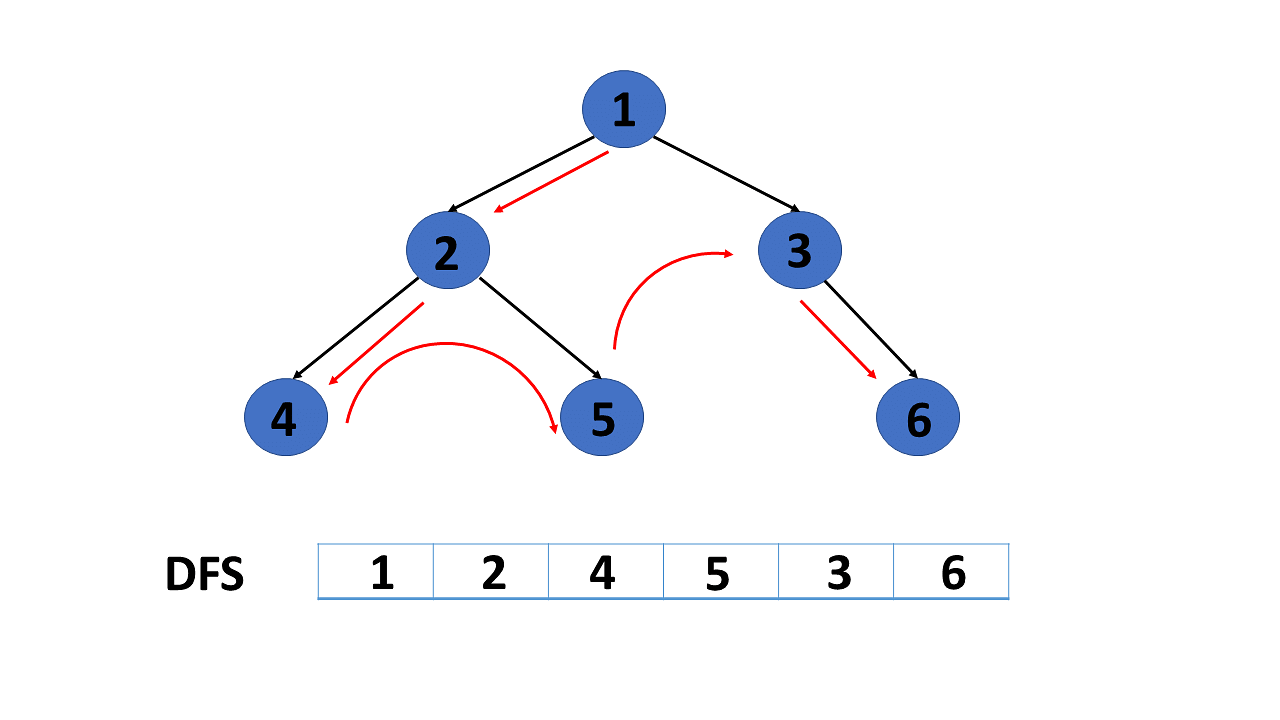
* Visited array (size of the graph)
* Stack data structure

Step 4: Insert v1 into the array's first block and push all the adjacent nodes or vertices of vertex v1 into the stack.

Step 5: Now, using the FIFO principle, pop the topmost element and put it into the visited array, pushing all of the popped element's nearby nodes into it.

Step 6: If the topmost element of the stack is already present in the array, discard it instead of inserting it into the visited array.

Step 7: Repeat step 6 until the stack data structure isn't empty.



Application of Graphs in Data Structures

Following  are some applications of graphs in data structures:

* Graphs are used in computer science to depict the flow of computation.
* Users on Facebook are referred to as vertices, and if they are friends, there is an edge connecting them. The Friend Suggestion system on Facebook is based on graph theory.
* You come across the Resource Allocation Graph in the Operating System, where each process and resource are regarded vertically. Edges are drawn from resources to assigned functions or from the requesting process to the desired resource. A stalemate will develop if this results in the establishment of a cycle.
* Web pages are referred to as vertices on the World Wide Web. Suppose there is a link from page A to page B that can represent an edge. This application is an illustration of a directed graph.
* Graph transformation systems manipulate graphs in memory using rules. Graph databases store and query graph-structured data in a transaction-safe, permanent manner.

1. Where are graph data structures used in real life?

You most likely utilise social networking platforms such as Facebook, LinkedIn, Instagram, and others. A wonderful example of a graph in usage is social media. Graphs are used in social media to hold information about each user. Every user is a node in this case, just like in Graph. Similarly, Google Maps is another application that makes use of graphs. In the case of Google Maps, each place is referred to as a node, and the roads that connect them are referred to as edges.

2. What are the different types of graphs in data structure?

A graph is a non-linear data structure composed of nodes and edges. They come in a variety of forms. Namely, they are Finite Graphs, Infinite Graphs, Trivial Graphs, Simple Graphs, Multi Graphs, Null Graphs, Complete Graphs, Pseudo Graphs, Regular Graphs, Labeled Graphs, Digraph Graphs, Subgraphs, Connected or Disconnected Graphs, and Cyclic Graphs.

3. How many types of graphs are there in data structure?

They are of 14 to 15 types. However, the most commonly used graph is the finite graph.

4. What is a complete graph in data structure?

A graph is considered to be complete if there is an edge between every pair of vertices in the graph. In other words, all of the graph's vertices are connected to the remainder of the graph's vertices. A full graph of 'n' vertices has precisely nC2 edges and is written as Kn.

5. What is a directed acyclic graph?

A directed acyclic graph (DAG) is a graph that is directed and has no cycles linking the other edges in computer science and mathematics. This indicates that traversing the complete graph from one edge is impossible. The edges of the directed graph can only move in one direction. The graph is a topological sorting in which each node has a specific order.

6. What is graph in data structure?

A graph is a type of non-linear data structure made up of vertices and edges. Vertices are also known as nodes, while edges are lines or arcs that link any two nodes in the network. In more technical terms, a graph comprises vertices (V) and edges (E). The graph is represented as G(E, V).

7. What is graph in data structure and its application?

A graph is a non-linear data structure made up of vertices (or nodes) linked by edges (or arcs), which can be directed or undirected. Graphs are used in computer science to depict the flow of computation.

8. How are graphs useful when interpreting data?

Graphs are a popular way to visually depict data connections. A graph's objective is to convey too many or intricate facts to be fully expressed in words and in less space. However, do not use graphs for little quantities of data that may be expressed in a phrase.