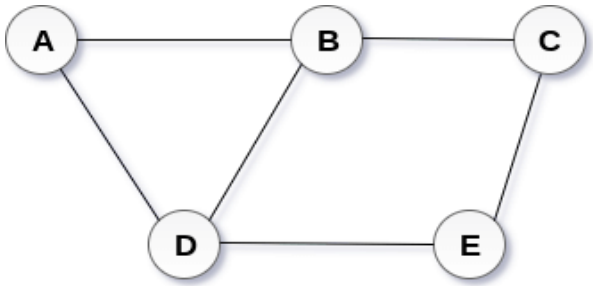
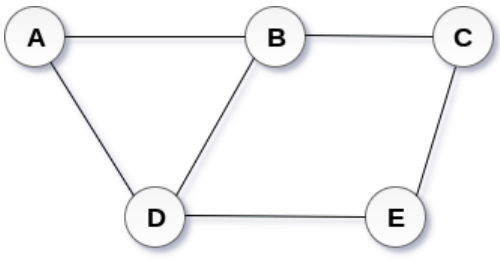
Graph

- A graph can be defined as group of vertices and edges that are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having parent child relationship.

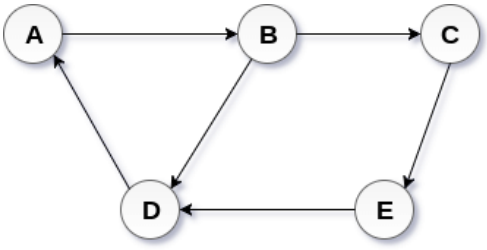
- **Definition:**
  - A graph G can be defined as an ordered set G(V, E) where V(G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices.
  - A Graph G(V, E) with 5 vertices (A, B, C, D, E) and six edges ((A,B), (B,C), (C,E), (E,D), (D,B), (D,A)) is shown in the following figure.



- **Directed and Undirected Graph**
  - A graph can be directed or undirected.
  - In an *undirected graph*, edges are not associated with the directions with them. In an undirected graph edges are not attached with any of the directions. If an edge exists between vertex A and B then the vertices can be traversed from B to A as well as A to B.
  - In a *directed graph*, edges form an ordered pair. Edges represent a specific path from some vertex A to another vertex B. Node A is called initial node while node B is called terminal node.



Undirected Graph

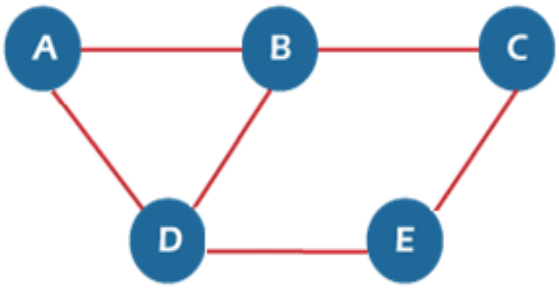


Directed Graph

- **Terminology:**
  - **Path:** A path can be defined as the sequence of nodes that are followed in order to reach some terminal node V from the initial node U.
  - **Closed Path:** A path will be called as closed path if the initial node is same as terminal node. A path will be closed path if $V_0=V_N$.
  - **Simple Path:** If all the nodes of the graph are distinct with an exception $V_0=V_N$, then such path P is called as closed simple path.
  - **Cycle:** A cycle can be defined as the path which has no repeated edges or vertices except the first and last vertices.
  - **Connected Graph:** A connected graph is the one in which some path exists between every two vertices (u, v) in V. There are no isolated nodes in connected graph.
  - **Complete Graph:** A complete graph is the one in which every node is connected with all other nodes. A complete graph contain n(n-1)/2 edges where n is the number of nodes in the graph.
  - **Weighted Graph:** In a weighted graph, each edge is assigned with some data such as length or weight. The weight of an edge e can be given as w(e) which must be a positive (+) value indicating the cost of traversing the edge.
  - **Digraph:** A digraph is a directed graph in which each edge of the graph is associated with some direction and the traversing can be done only in the specified direction.
  - **Loop:** An edge that is associated with the similar end points can be called as Loop.
  - **Adjacent Nodes:** If two nodes u and v are connected via an edge e, then the nodes u and v are called as neighbours or adjacent nodes.
  - **Degree of the Node:** A degree of a node is the number of edges that are connected with that node. A node with degree 0 is called as isolated node.

- **Graph Representation:**
  - A graph is a data structure that consist a sets of vertices (called nodes) and edges. There are two ways to store Graphs into the computer's memory:
    - **Sequential representation Or Adjacency matrix representation**
    - **Linked list representation Or Adjacency list representation**
  - In sequential representation, an adjacency matrix is used to store the graph. Whereas in linked list representation, there is a use of an adjacency list to store the graph.

- **Sequential Representation:**
  - In sequential representation, there is a use of an adjacency matrix to represent the mapping between vertices and edges of the graph. We can use an adjacency matrix to represent the undirected graph, directed graph, weighted directed graph, and weighted undirected graph.
  - If adj[i][j] = w, it means that there is an edge exists from vertex i to vertex j with weight w.
  - An entry A_{ij} in the adjacency matrix representation of an undirected graph G will be 1 if an edge exists between V_i and V_j.
  - If Graph G consists of n vertices, then the adjacency matrix for that graph is n x n, and the matrix A = [aij] can be defined as –
    a_{ij} = 1 {if there is a path exists from V_i to V_j}
    a_{ij} = 0 {Otherwise}
    It means that, in an adjacency matrix, 0 represents that there is no association exists between the nodes, whereas 1 represents the existence of a path between two edges.
  - If there is no self-loop present in the graph, it means that the diagonal entries of the adjacency matrix will be 0.
  - **Adjacency matrix for an undirected graph:**



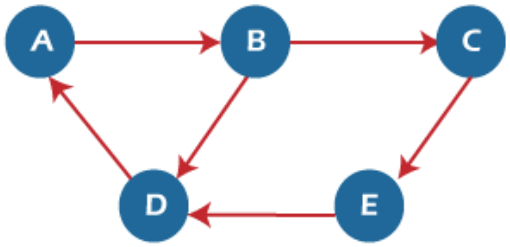
Undirected Graph

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

Adjacency Matrix

  - In the above figure, an image shows the mapping among the vertices (A, B, C, D, E), and this mapping is represented by using the adjacency matrix.
  - There exist different adjacency matrices for the directed and undirected graph. In a directed graph, an entry A_{ij} will be 1 only when there is an edge directed from V_i to V_j.

- **Adjacency matrix for a directed graph:**
  - In a directed graph, edges represent a specific path from one vertex to another vertex. Suppose a path exists from vertex A to another vertex B; it means that node A is the initial node, while node B is the terminal node.



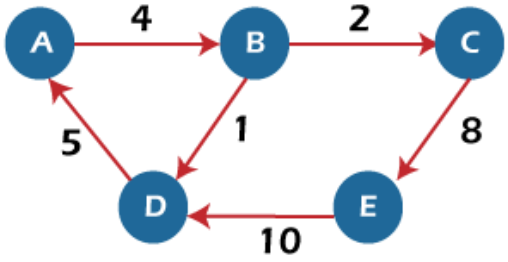
Directed Graph

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

Adjacency Matrix

  - In the above graph, we can see there is no self-loop, so the diagonal entries of the adjacent matrix are 0.

- **Adjacency matrix for a weighted directed graph:**
  - It is similar to an adjacency matrix representation of a directed graph except that instead of using the '1' for the existence of a path, here we have to use the weight associated with the edge. The weights on the graph edges will be represented as the entries of the adjacency matrix.
  - Consider the below graph and its adjacency matrix representation. In the representation, we can see that the weight associated with the edges is represented as the entries in the adjacency matrix.



weighted Directed Graph

	A	B	C	D	E
A	0	4	0	0	0
B	0	0	2	1	0
C	0	0	0	0	8
D	5	0	0	0	0
E	0	0	0	10	0

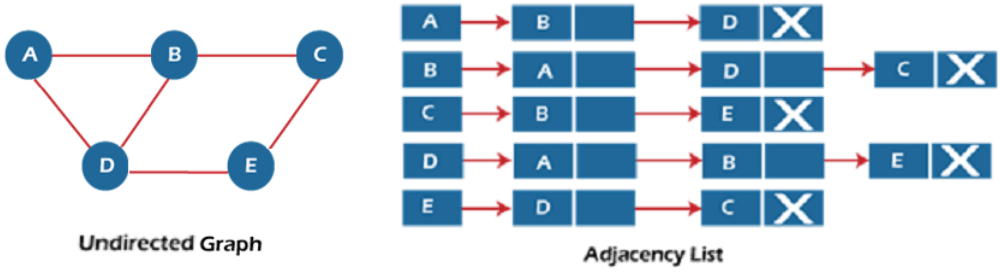
Adjacency Matrix

  - In the above image, we can see that the adjacency matrix representation of the weighted directed graph is different from other representations. It is because, in this representation, the non-zero values are replaced by the actual weight assigned to the edges.

- Adjacency matrix is easier to implement and follow. An adjacency matrix can be used when the graph is dense and a number of edges are large.
- Though, it is advantageous to use an adjacency matrix, but it consumes more space. Even if the graph is sparse, the matrix still consumes the same space.

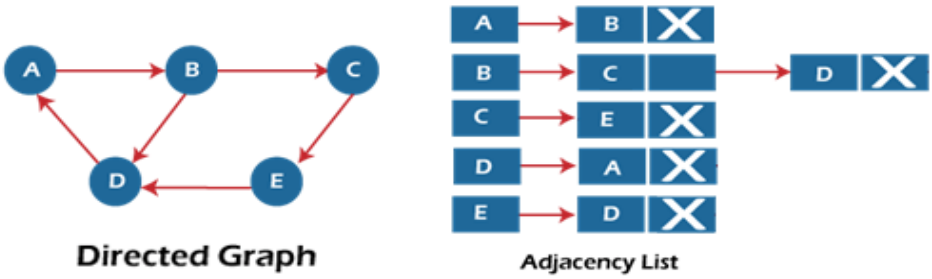
- **Linked List Representation:**
  - An adjacency list is used in the linked representation to store the Graph in the computer's memory. It is efficient in terms of storage as we only have to store the values for edges.

  - **Adjacency list representation of an undirected graph:**



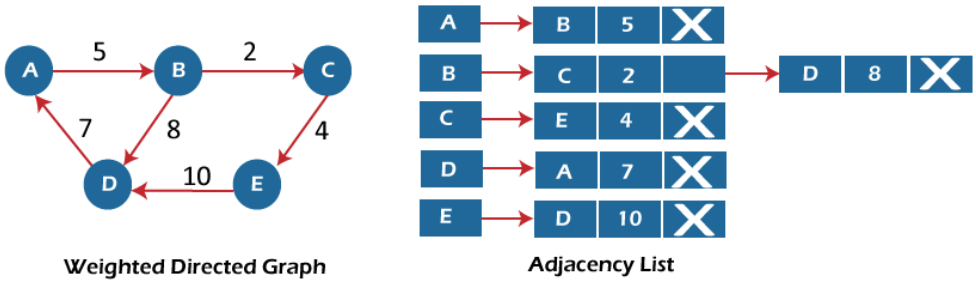
- In the above figure, we can see that there is a linked list or adjacency list for every node of the graph. From vertex A, there are paths to vertex B and vertex D. These nodes are linked to nodes A in the given adjacency list.
- An adjacency list is maintained for each node present in the graph, which stores the node value and a pointer to the next adjacent node to the respective node. If all the adjacent nodes are traversed, then store the NULL in the pointer field of the last node of the list.
- The sum of the lengths of adjacency lists is equal to twice the number of edges present in an undirected graph.

  - **Adjacency list representation of a directed graph:**



- For a directed graph, the sum of the lengths of adjacency lists is equal to the number of edges present in the graph.

- **Adjacency list representation of weighted directed graph:**



  - In the case of a weighted directed graph, each node contains an extra field that is called the weight of the node.
  - In an adjacency list, it is easy to add a vertex. Because of using the linked list, it also saves space.

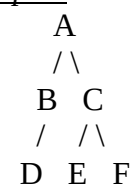
• Breadth First Search:

- Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.
- There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.
- **Applications:**
  - BFS can be used to find the neighboring locations from a given source location.
  - In a peer-to-peer network, BFS algorithm can be used as a traversal method to find all the neighboring nodes. Most torrent clients, such as BitTorrent, uTorrent, etc. employ this process to find "seeds" and "peers" in the network.
  - BFS can be used in web crawlers to create web page indexes. It is one of the main algorithms that can be used to index web pages. It starts traversing from the source page and follows the links associated with the page. Here, every web page is considered as a node in the graph.
  - BFS is used to determine the shortest path and minimum spanning tree.
  - BFS is also used in Cheney's technique to duplicate the garbage collection.
  - It can be used in ford-Fulkerson method to compute the maximum flow in a flow network.

- **Depth First Search:**
  - Depth-first search is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.
  - Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.
  - **Applications:**
    - DFS algorithm can be used to implement the topological sorting.
    - It can be used to find the paths between two vertices.
    - It can also be used to detect cycles in the graph.
    - DFS algorithm is also used for one solution puzzles.
    - DFS is used to determine if a graph is bipartite or not.

- **Difference between BFS and DFS**
  - **Breadth First Search:**
    - **BFS** stands for **Breadth First Search** is a *vertex-based technique* for finding the shortest path in the graph. It uses a *Queue data structure* that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue. It is slower than DFS.
    - **Example**:

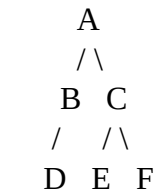
      Input:

      Output:

      A, B, C, D, E, F
  - **Depth First Search:**
    - **DFS** stands for **Depth First Search** is an *edge-based technique*. It uses the *Stack data structure* and performs two stages, first visited vertices are pushed into the stack, and second if there are no vertices then visited vertices are popped.
    - **Example:**

      Input:

      Output:

      A, B, D, C, E, F

Breadth First Search vs Depth First Search		
Sr. No.	**BFS**	**DFS**
1	BFS uses Queue data structure for finding the shortest path.	DFS uses Stack data structure.
2	BFS can be used to find single source shortest path in an unweighted graph, because in BFS, we reach a vertex with minimum number of edges from a source vertex.	DFS, we might traverse through more edges to reach a destination vertex from a source.
3	BFS is more suitable for searching vertices which are closer to the given source.	DFS is more suitable when there are solutions away from source.
4	BFS considers all neighbors first and therefore not suitable for decision making trees used in games or puzzles.	DFS is more suitable for game or puzzle problems. We make a decision, then explore all paths through this decision. And if this decision leads to win situation, we stop.
5	Here, siblings are visited before the children.	Here, children are visited before the siblings.
6	In BFS there is no concept of backtracking.	DFS algorithm is a recursive algorithm that uses the idea of backtracking
7	BFS is used in various application such as bipartite graph, and shortest path etc.	DFS is used in various application such as acyclic graph and topological order etc.
8	BFS requires more memory.	DFS requires less memory.