Data Structure & Algorithms

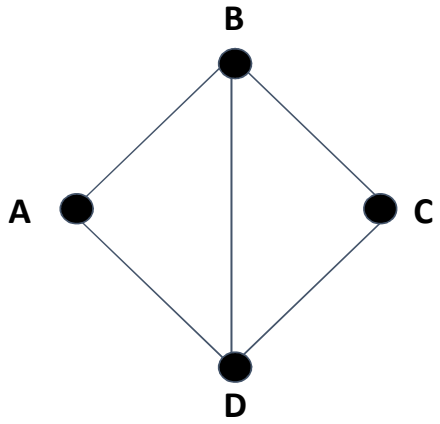
Graph

What is graph?

- A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links.
- The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.
- Formally, a graph is a pair of sets **(V, E)**, where V is the set of vertices and E is the set of edges, connecting the pairs of vertices.
- Take a look at this example:

V = { A, B, C, D}

E = { (A,B) , (B,D) , (A,D) , (B,C) , (C,D) }



Graph Terminology

- **Path:**

  Path represents a sequence of edges between the two vertices. In the previous example, ADBC represents a path from A to C.

- **Cycle:**

  A cycle is a non-empty trail in which the only repeated vertices are the first and last vertices. In the previous example, ABDA represents a cycle.

- **Adjacency:**

  Two vertices are adjacent if they are connected to each other through an edge. In the previous example, B is adjacent to A and C.

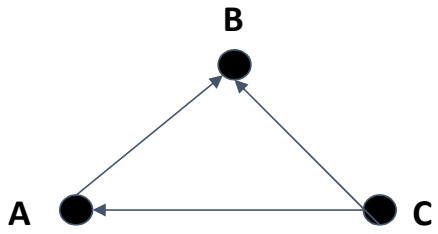
Graph Terminology (cont.)

- **Degree:**

  The degree of a vertex is the number of edges that are connected with that vertex. In the previous example, degree of B is 3 and we write: d(B) = 3

  In a directed graph, each vertex has an **indegree** and an **outdegree**. Indegree of a vertex is the number of edges which are coming into that vertex and Outdegree of a vertex is the number of edges which are going out from that vertex. See the example below:

indegree(B) = 2
outdegree(C) = 2



Types of Graph

- **Directed Graph:**

  In a directed graph, nodes are connected by directed edges – they only go in one direction. For example, if an edge connects node 1 and 2, but the arrow head points towards 2, we can only traverse from node 1 to node 2 – not in the opposite direction.

- **Undirected Graph:**

  In an undirected graph, nodes are connected by edges that are all bidirectional. For example if an edge connects node 1 and 2, we can traverse from node 1 to node 2, and from node 2 to 1.

Types of Graph (cont.)

- **Weighted Graph:**

  A weight is a numerical value attached to each individual edge. In a weighted graph relationships between nodes have a magnitude and this magnitude is important to the relationship we're studying. For example, consider that we want to show the distances between 2 cities: **Isfahan** —(500 km)— **Tehran**

- **Unweighted Graph:**

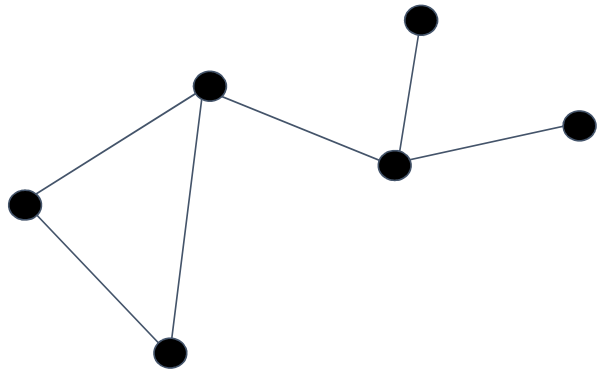
  If the edges do not have weights, the graph is said to be unweighted.

Types of Graph (cont.)

- **Connected Graph:**

  A graph is said to be connected if there is a path between every pair of vertex. In other words, from every vertex to any other vertex, there should be at least one path to traverse.

  Example:



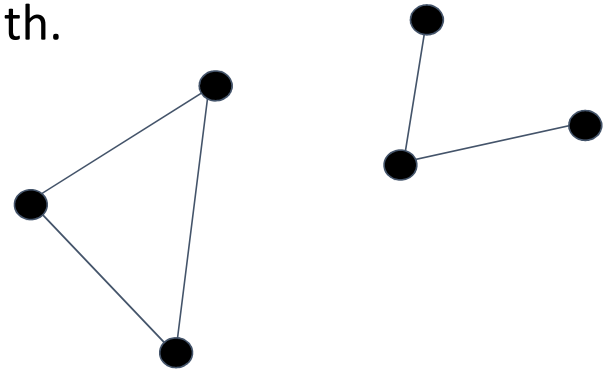
Types of Graph (cont.)

- **Disconnected Graph:**

  A graph where the vertices separate into two or more distinct groups, where you cannot link a vertex in one group to a vertex in another by traversing along a series of edges.

  In simple words, a graph is disconnected if at least two vertices of the graph are not connected by a path.

  Example:

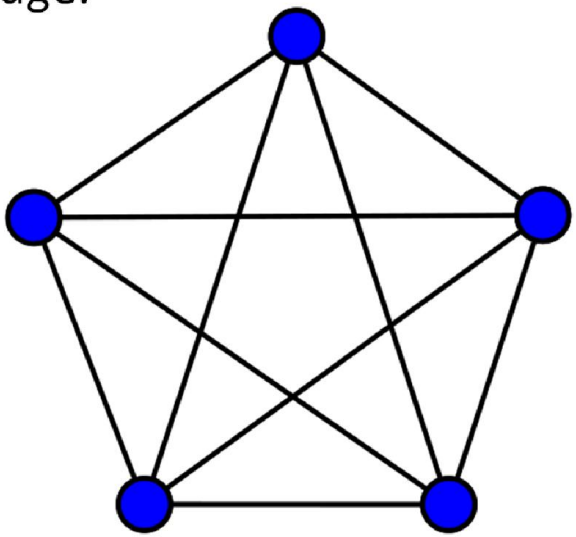


Types of Graph (cont.)

- **Complete Graph:**

  A complete graph is an undirected graph in which every pair of vertices is connected by a unique edge.

  Example:

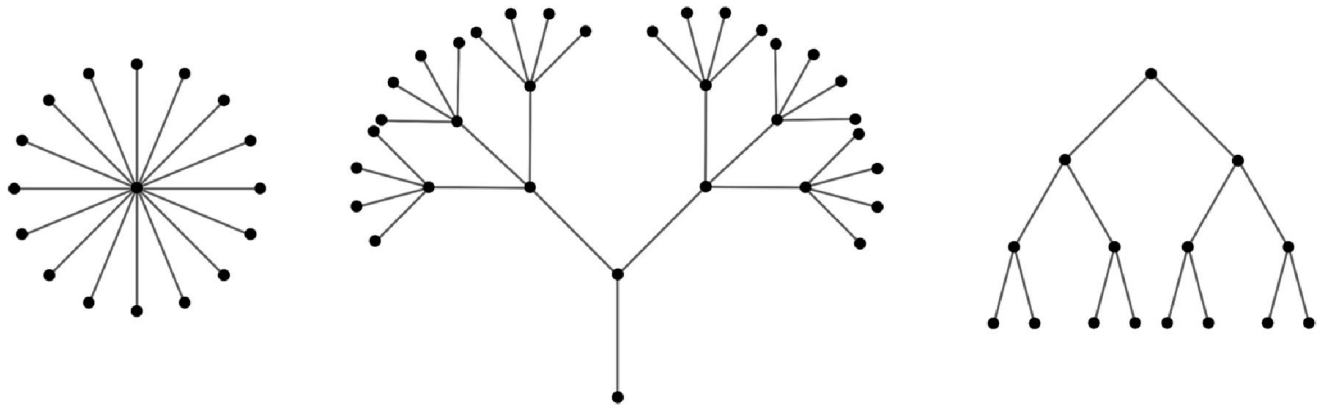


Types of Graph – Tree

- **Tree:**

  A tree is a **connected** graph *without* any cycle, which means any two vertices are connected by *exactly one* path.

  Examples:



Types of Graph – Tree (cont.)

- **Rooted Tree:**

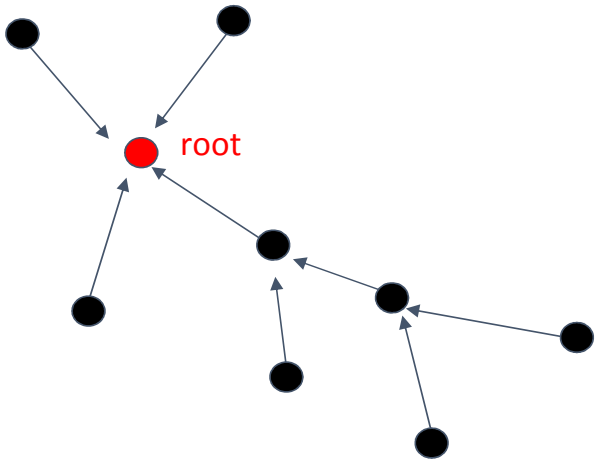
  A rooted tree is a tree in which one vertex has been designated the **root**.

  The edges of a rooted tree can be assigned a natural orientation, either *away from* or *towards* the root, in which case the structure becomes a *directed rooted tree*.

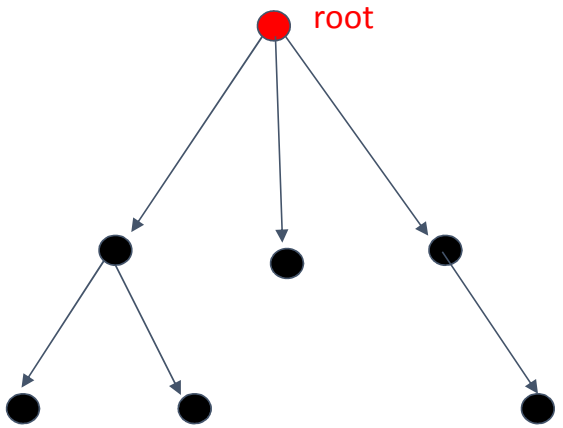
  When a directed rooted tree has an orientation away from the root, it is called an out-tree; when it has an orientation towards the root, it is called an in-tree.

Types of Graph – Tree (cont.)

Examples:



In- tree



out- tree

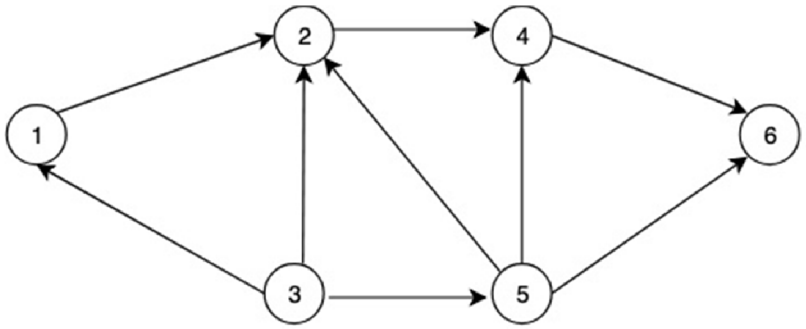
Graph Representation

Graphs can be represented in two specific ways:

1. by using an **adjacency matrix**
2. by using an **adjacency list**

Adjacency Matrix

- Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph.
- Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j.
  Consider the following example:

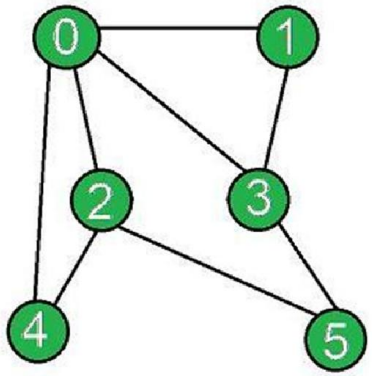


	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	0	1	0	0
3	1	1	0	0	1	0
4	0	0	0	0	0	1
5	0	1	0	1	0	1
6	0	0	0	0	0	0

Adjacency Matrix (cont.)

- Adjacency matrix for undirected graph is always **symmetric**. Because a[i][j] and a[j][i] are both equal to 1.

  Example:

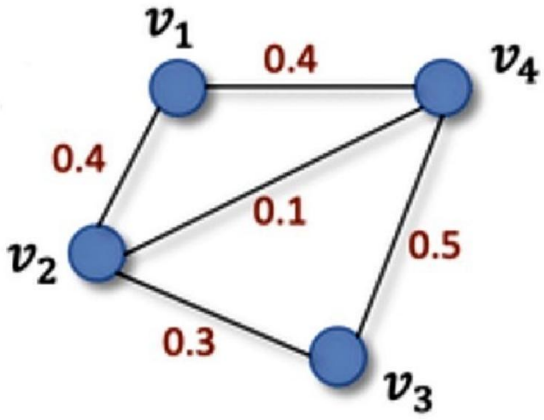


	0	1	2	3	4	5
0	0	1	1	1	1	0
1	1	0	0	1	0	0
2	1	0	0	0	1	1
3	1	1	0	0	0	1
4	1	0	1	0	0	0
5	0	0	1	1	0	0

Adjacency Matrix (cont.)

- Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

  Example:

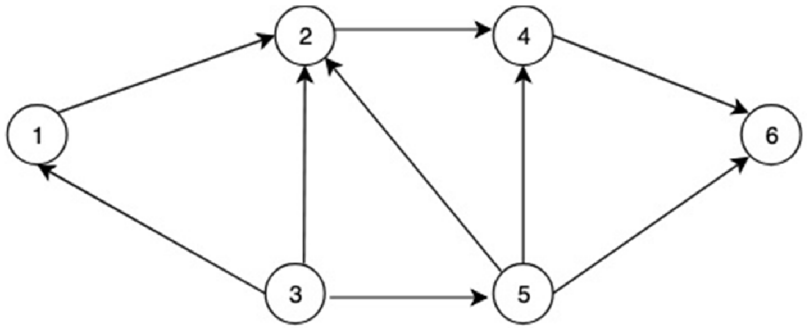


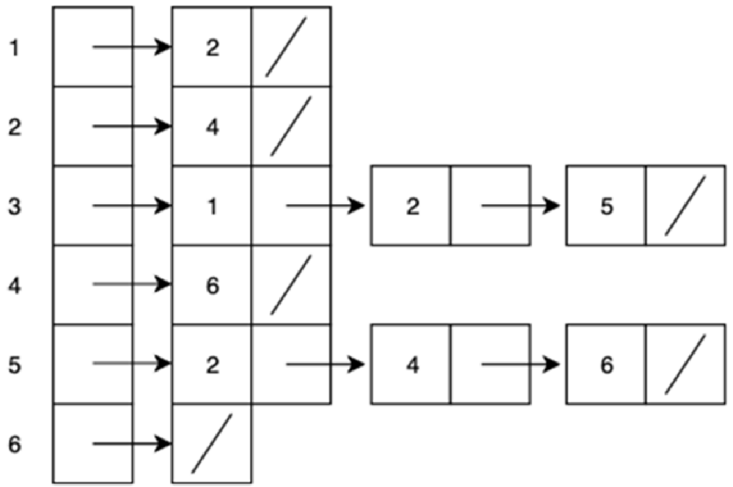
	V1	V2	V3	V4
V1	**0**	**0.4**	**0**	**0.4**
V2	**0.4**	**0**	**0.3**	**0.1**
V3	**0**	**0.3**	**0**	**0.5**
V4	**0.4**	**0.1**	**0.5**	**0**

Adjacency List

- In adjacency list representation we have a list of size V where V is the number of vertices in a graph. Each entry of the list contains another list, which is the set of all the vertices adjacent to the current vertex.

Consider the following example:

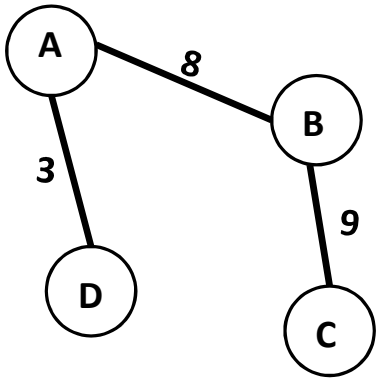


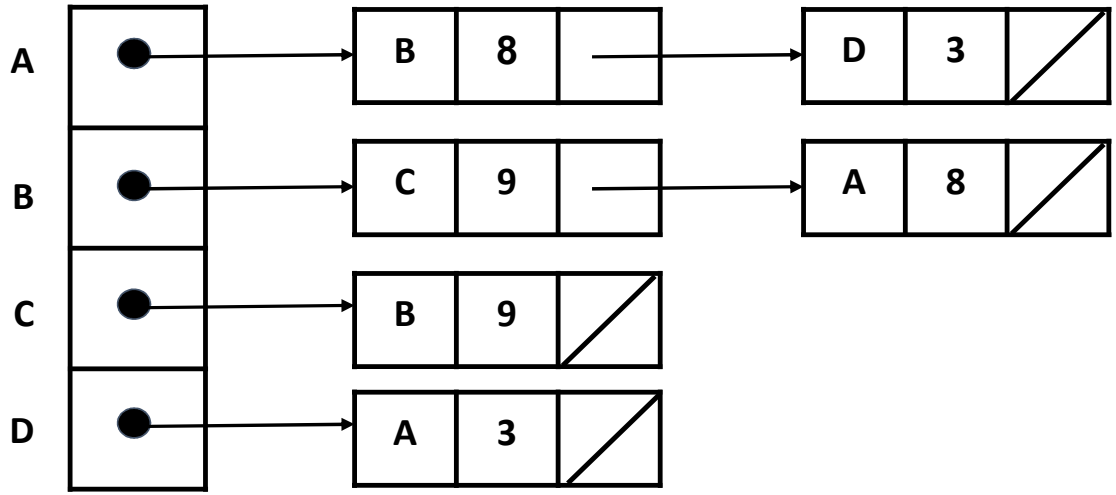


Adjacency List (cont.)

- For a weighted graph, we will add an extra weight parameter to the adjacency list representation.

  Example:





Comparison between adj matrix and adj list

- **Space Complexity:**

  - Adjacency matrix makes use of VxV matrix, so the required space is **$O(|V|^2)$**.

  - In Adjacency list, O(V) is required for storing the vertices and O(E) is required for storing neighbors corresponding to every vertex. Thus, overall space complexity is **$O(|V|+|E|)$**. (Note: in a connected graph, $|E| = |V|*(|V|-1) / 2$. So in worst case the required space for adjacency list is also $O(|V|^2)$).

Comparison between adj matrix and adj list

- **Time Complexity:**

  In order to find an existing edge:

  - In adjacency matrix, given two vertices say i and j matrix[i][j] can be checked in **O(1)** time.

  - In adjacency list, in order to check for an edge we need to check for vertices adjacent to given vertex. A vertex can have at most O(|V|) neighbours and in worst case we would have to check for every adjacent vertex. Therefore, time complexity is **O(|V|)** .