Graph

As you looked at the **binary tree** data structure, which provides a useful way of storing data for efficient searching, whereas each node can have up to **two child nodes**. More general tree structures can be created in which different numbers of child nodes are allowed for each node.

All tree structures are **hierarchical**, which means that each node can **only have one parent node**, for example a family tree or a computer files system. But some data need to have connections between items which **do not fit into a hierarchy** like this. **Graph** data structures can be useful in these situations. A graph consists of a number of data items, each of which is called a **vertex**. Any vertex may be connected to any other, and these connections are called edges.

Some problems that can be represented by a graph:

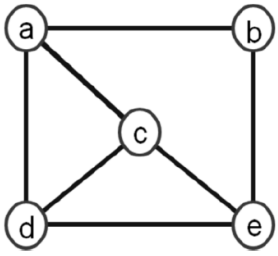
- computer networks
- airline flights
- networks of paths in a city (roads map)
- telephone network or circuit network
- social networks like linkedIn, facebook
- course prerequisite structure
- tasks for completing a job
- flow of control through a program

Graph and its representations

Graph is a data structure G = (V,E) that consists of following two components:

1. V: A finite set of **vertices** also called as **nodes**.
2. E: A finite set of ordered pairs of the form (u, v) called as **edges**. The pair is ordered because (u, v) is not same as (v, u) in case of directed graph(di-graph). The pair of form (u, v) indicates that there is an edge from vertex u to vertex v. **The edges may contain weight/value/cost.**

Example:

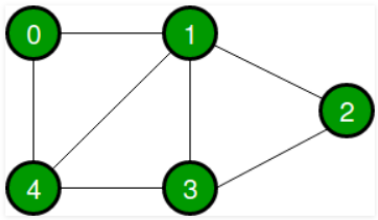


V= {a,b,c,d,e}

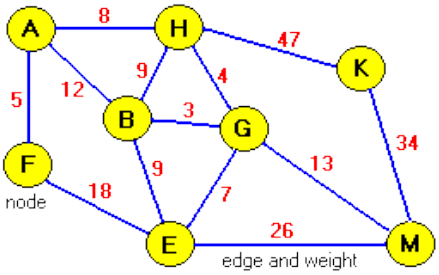
E=
{(a,b),(a,c),(a,d),
(b,e),(c,d),(c,e),
(d,e)}

The following figures are examples of **types of graphs**:

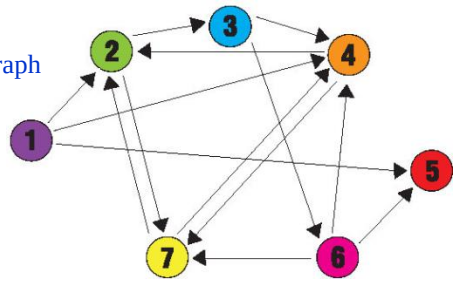
Undirected-Unweighted Graph

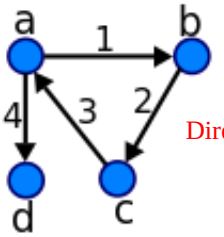


Undirected-weighted Graph



Directed-Unweighted Graph





Directed-Weighted Graph

The two most commonly used representations of graph are:

1. Adjacency Matrix
2. Adjacency List

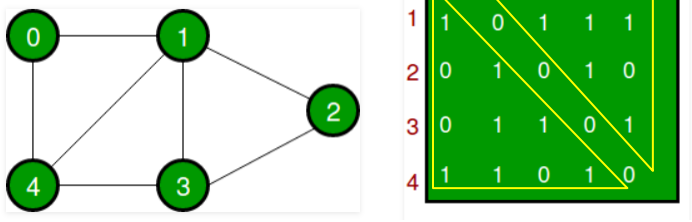
There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

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.

Adjacency matrix for undirected graph is always symmetric. 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.



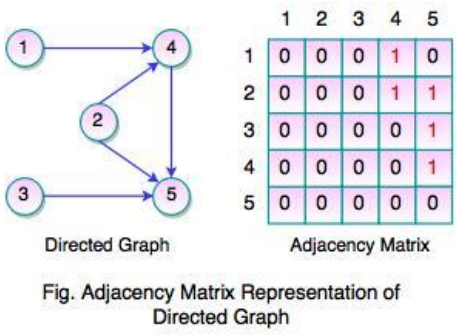


Fig. Adjacency Matrix Representation of Directed Graph

Adjacency List:

An array of linked lists is used, size of the array is equal to number of vertices. An entry array[i] represents the linked list of vertices adjacent to the ***i***th vertex.

This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists.