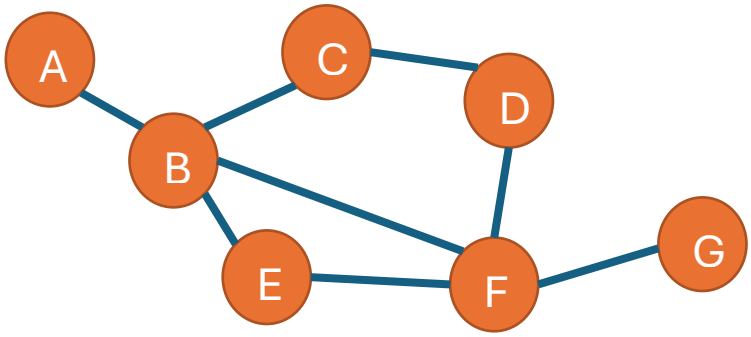
4.2.4 Graphs & Trees

Graphs

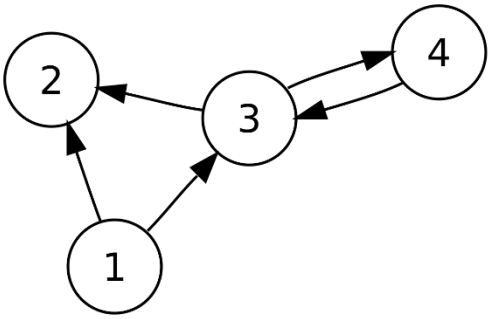
A graph is a data structure that contains:

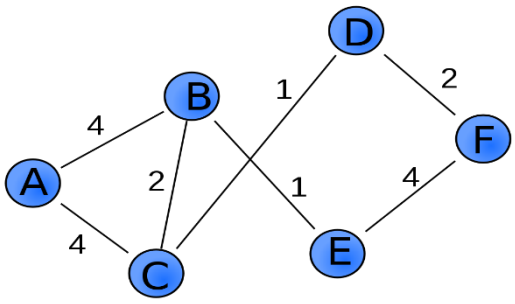
- **Nodes** which hold data
- **Edges** which show relationships between the nodes



Graphs can be:

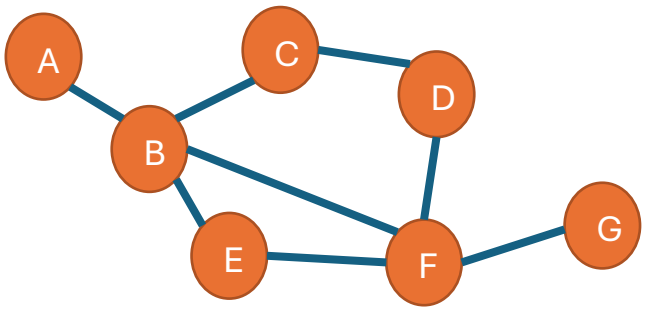
- **Directed** where edges can only be traversed in one direction
- **Weighted** where edges have a cost associated with them





Implementation 1 – Adjacency Matrix

One way of implementing a graph is using an adjacency matrix. The matrix stores a value if an edge exists between two nodes or 0 if one does not exist.



	A	B	C	D	E	F	G
A	**0**	**1**	**0**	**0**	**0**	**0**	**0**
B		**0**	**1**	**0**	**1**	**1**	**0**
C			**0**	**1**	**0**	**0**	**0**
D				**0**	**0**	**1**	**0**
E					**0**	**1**	**0**
F						**0**	**1**
G							**0**

For an undirected graph, the matrix is symmetrical, so only one half needs to be shown.

Pros

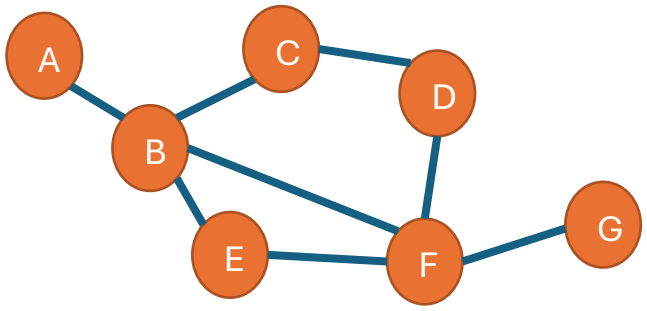
- Fast at checking if there is an edge between two nodes.
- Fast at adding new edges.

Cons

- Slow at deleting nodes.
- Uses up memory recording the absence of edges. Especially in sparse graphs (graphs with minimal edges).

Implementation 2 – Adjacency List

Another way of implementing a graph is using an adjacency list. A list is made for each node that stores the nodes that are directly connected to it.



Vertex	Adjacent vertices
A	[B]
B	[A, C, E]
C	[B, D]
D	[C, F]
E	[B, F]
F	[B, D, E, G]
G	[F]

Pros

- Only stores data where there is an edge so uses less memory. Therefore, better for sparse graphs.
- Fast at adding new nodes.

Cons

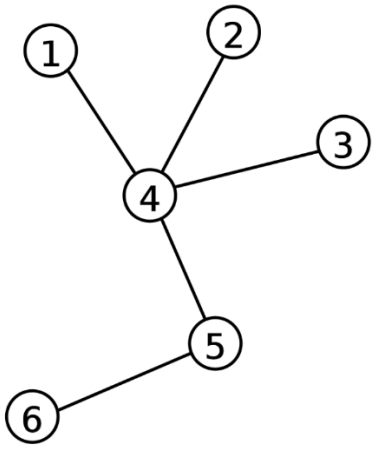
- Slow at checking if there is an edge between two nodes.
- Slow at removing edges

Trees

A tree is a graph that:

- Is **connected**
- Has **no cycles**

As a graph it is made up of nodes and edges.



Binary Trees

A binary tree is a tree:

- That is **rooted**
- where every node has **at most two child nodes**

Typical operations on a binary tree are **adding** a node, **deleting** a node and performing a **binary search**.