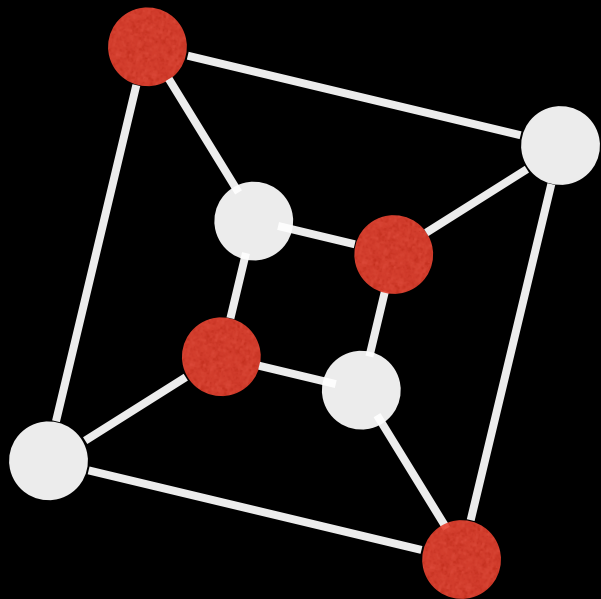


# Graph Theory

## Intro & Overview



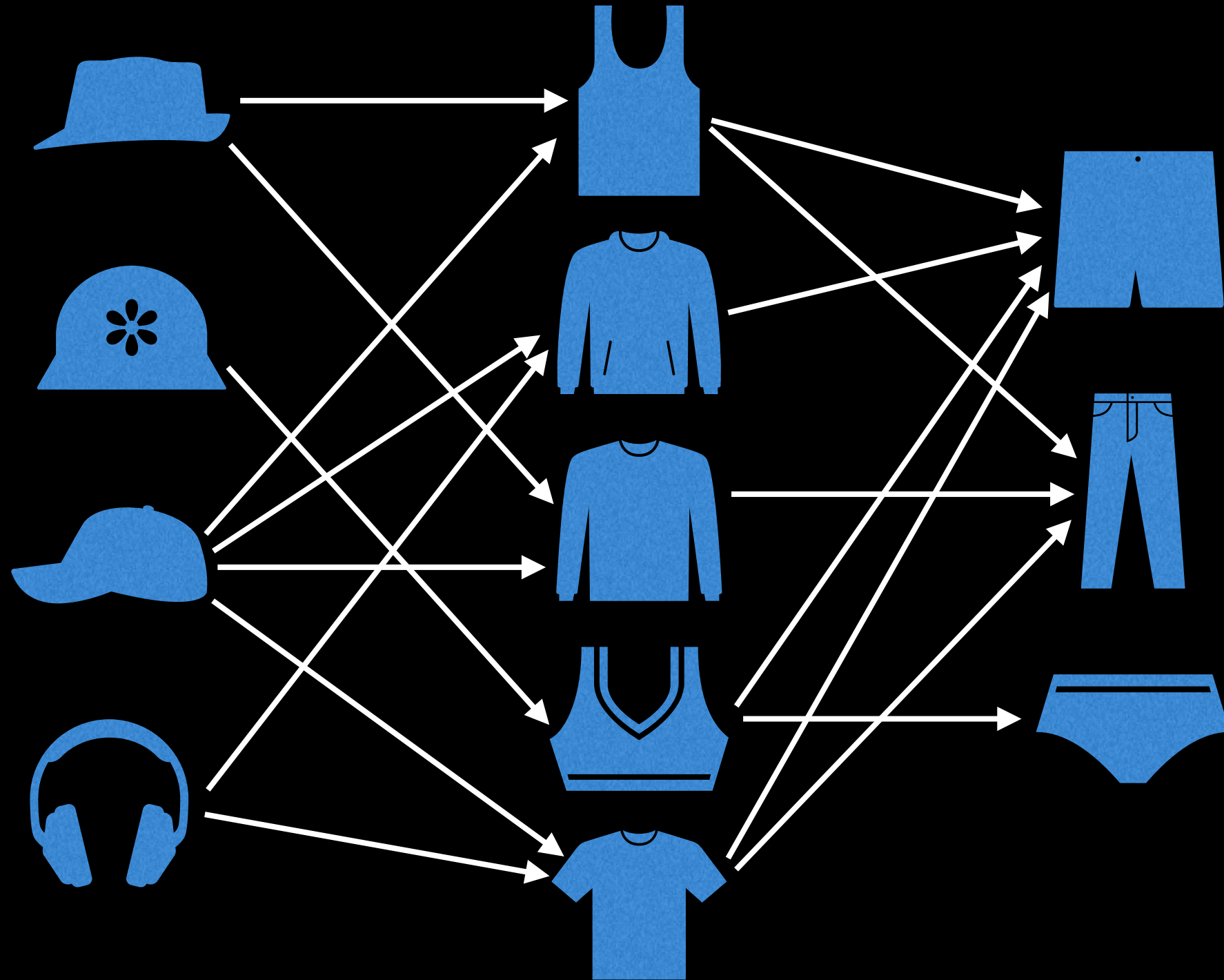
William Fiset

# Brief introduction

**Graph theory** is the mathematical theory of the properties and applications of graphs (networks).

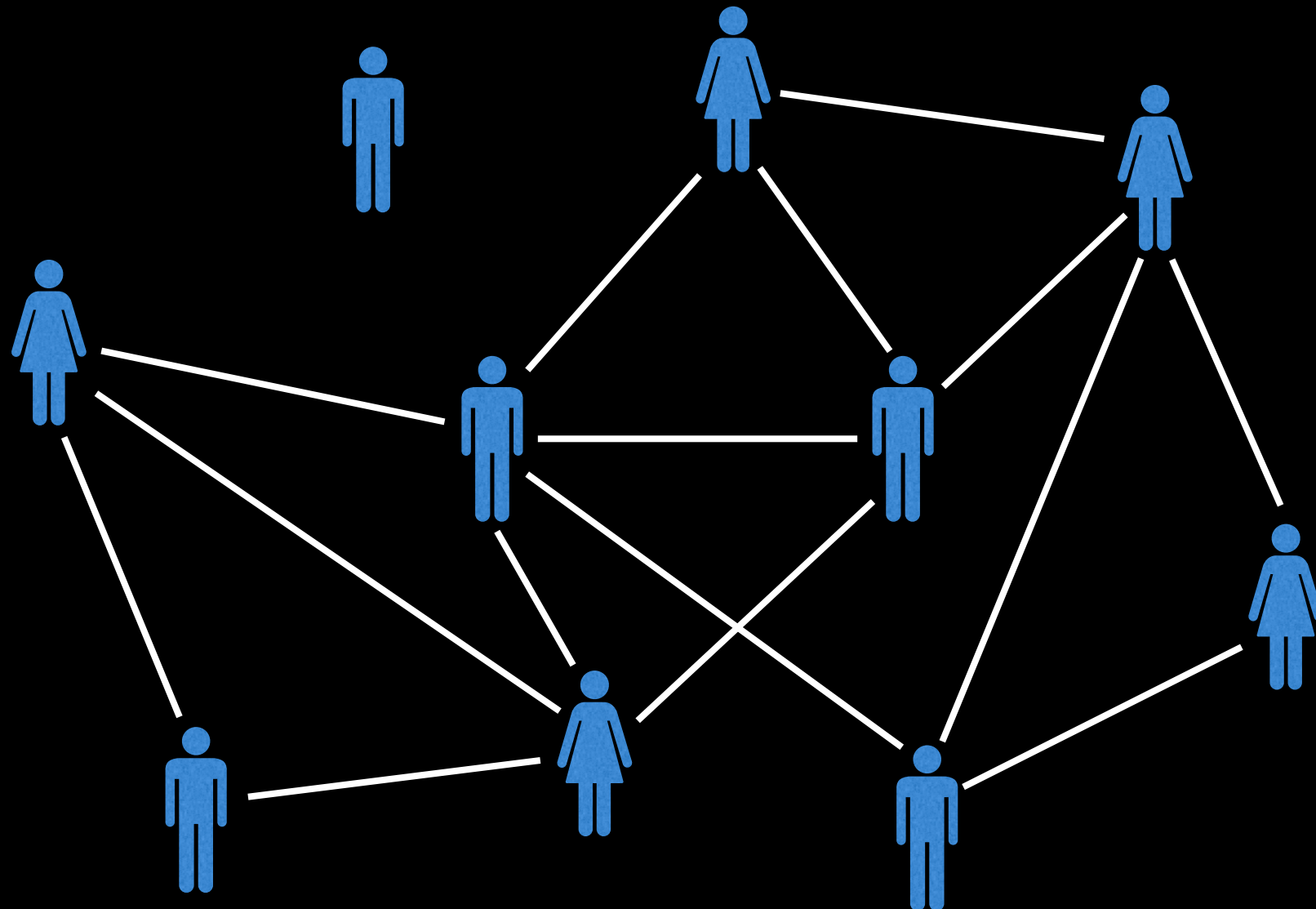
The goal of this series is to gain an understanding of how to apply graph theory to real world applications.

# Brief introduction



A graph theory problem might be:  
Given the constraints above, how many different  
sets of clothing can I make by choosing an article  
from each category?

# Brief introduction



The canonical graph theory example is  
a social network of friends.

This enables interesting questions such as: how many friends does person X have? Or how many degrees of separation are there between person X and person Y?

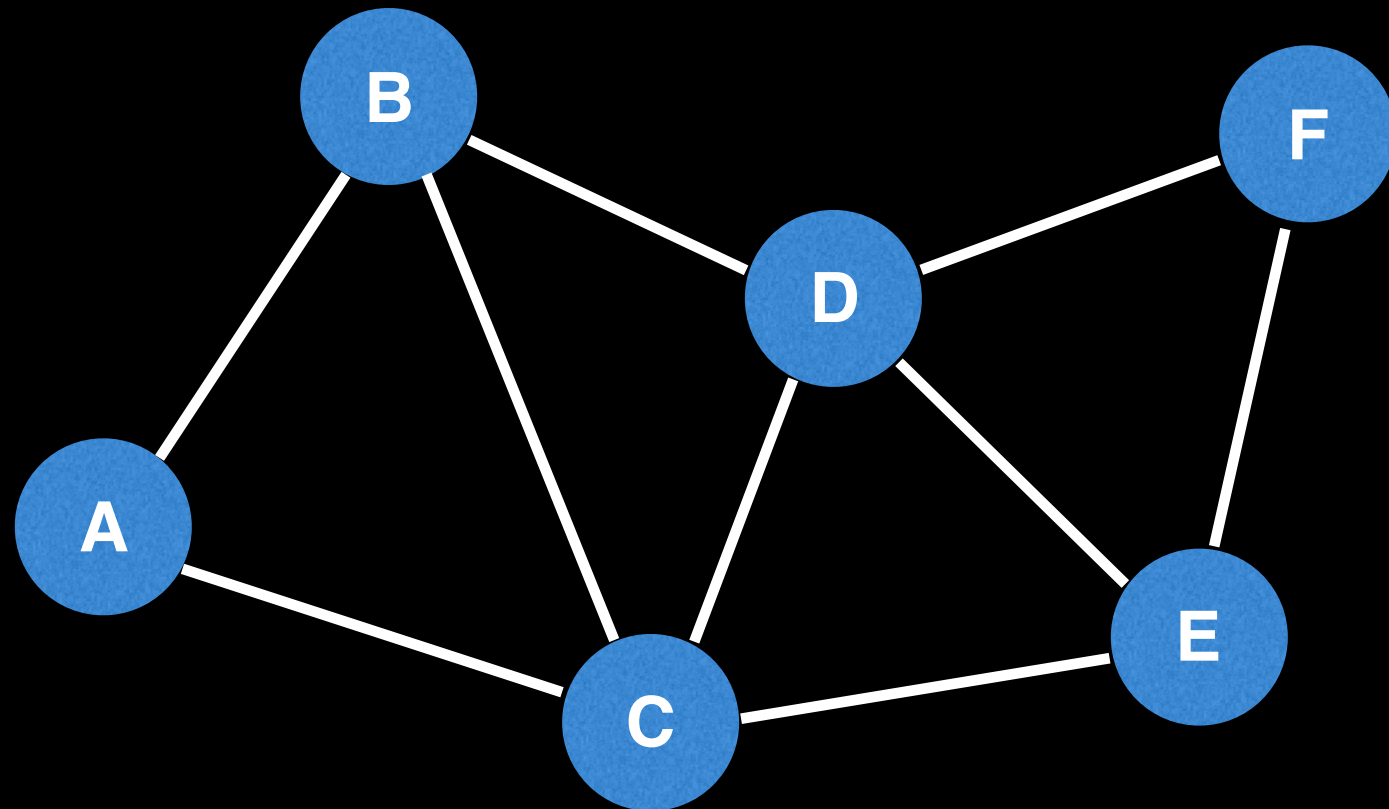
# Types of Graphs

# Undirected Graph

An **undirected graph** is a graph in which edges have no orientation. The edge  $(u, v)$  is identical to the edge  $(v, u)$ . – Wiki

# Undirected Graph

An **undirected graph** is a graph in which edges have no orientation. The edge  $(u, v)$  is identical to the edge  $(v, u)$ . – Wiki



In the graph above, the nodes could represent cities and an edge could represent a bidirectional road.

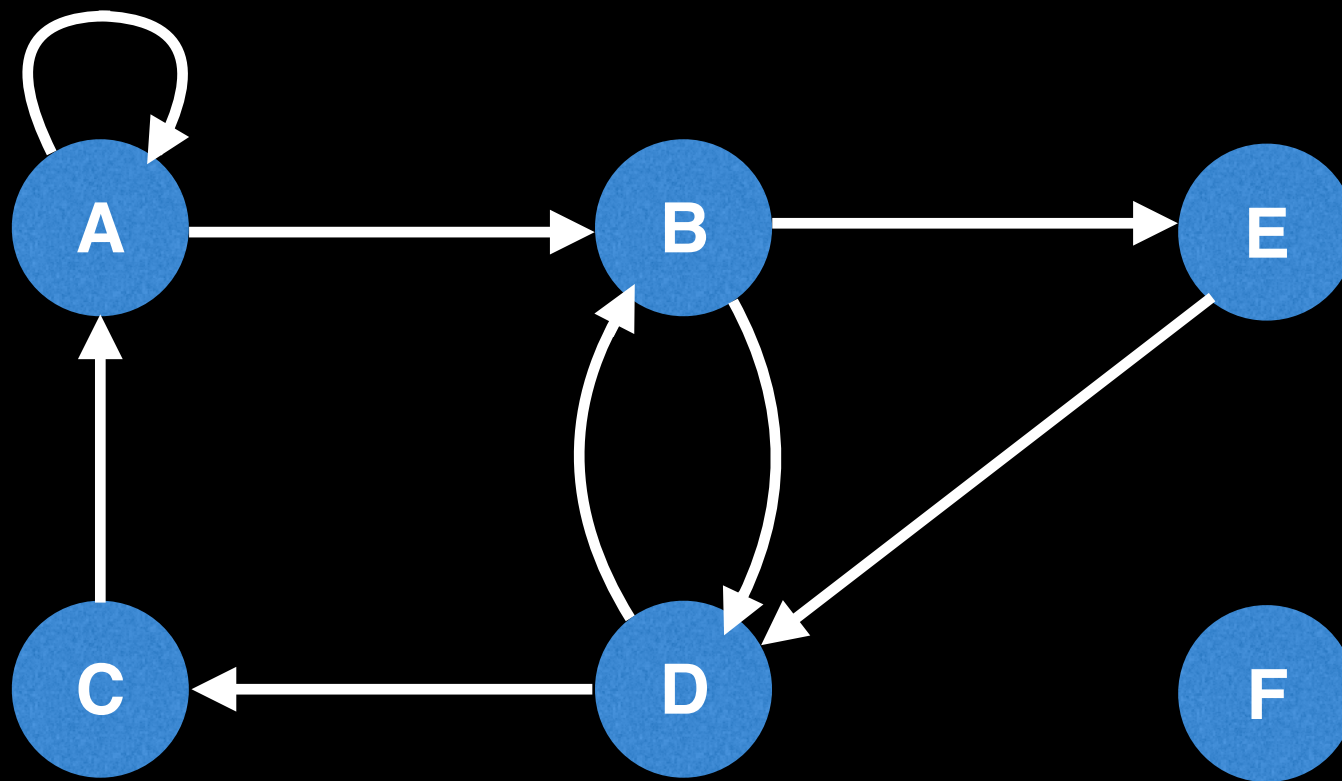
# Directed Graph (Digraph)

A *directed graph* or *digraph* is a graph in which edges have orientations. For example, the edge  $(u, v)$  is the edge *from* node  $u$  *to* node  $v$ .



# Directed Graph (Digraph)

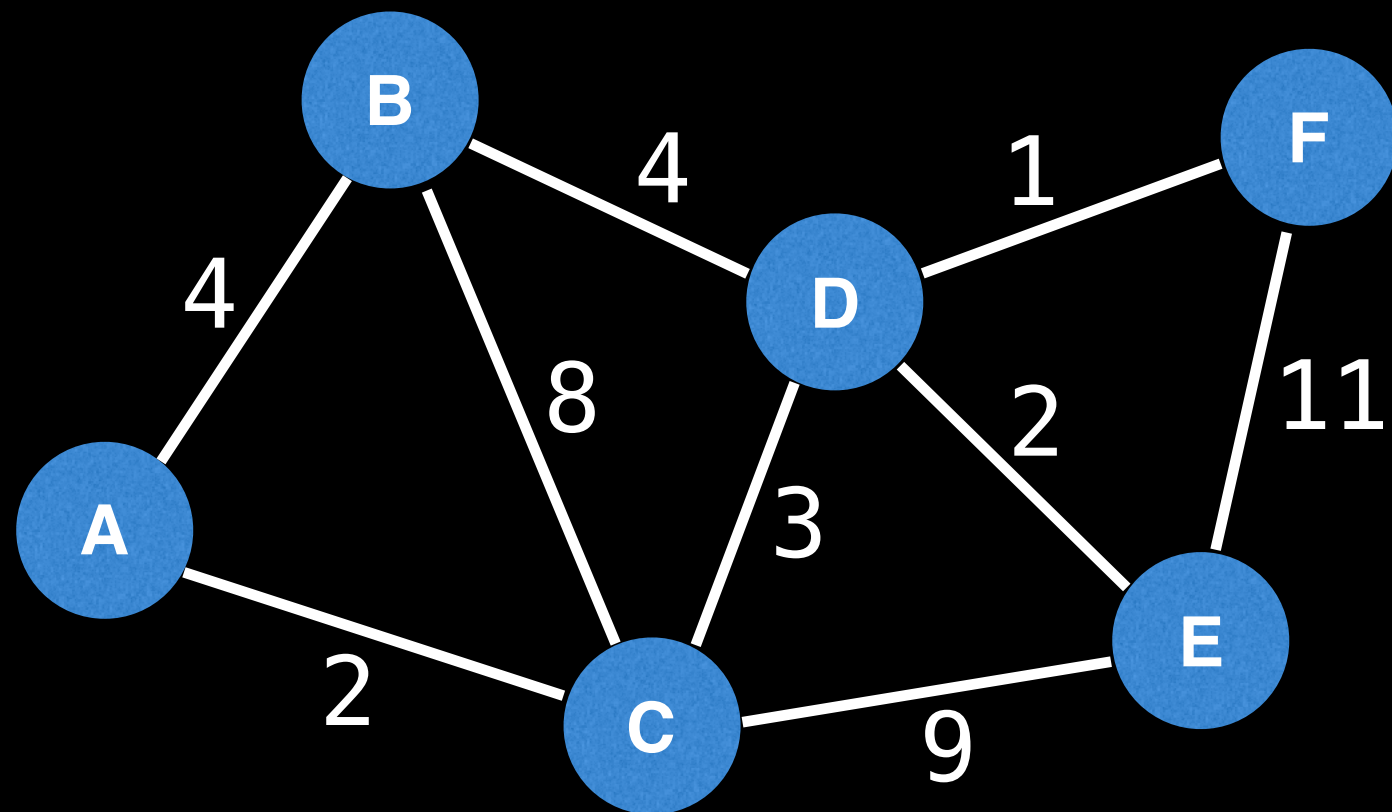
A **directed graph** or **digraph** is a graph in which edges have orientations. For example, the edge  $(u, v)$  is the edge *from* node  $u$  *to* node  $v$ .



In the graph above, the nodes could represent people and an edge  $(u, v)$  could represent that person  $u$  bought person  $v$  a gift.

# Weighted Graphs

Many graphs can have edges that contain a certain weight to represent an arbitrary value such as cost, distance, quantity, etc...

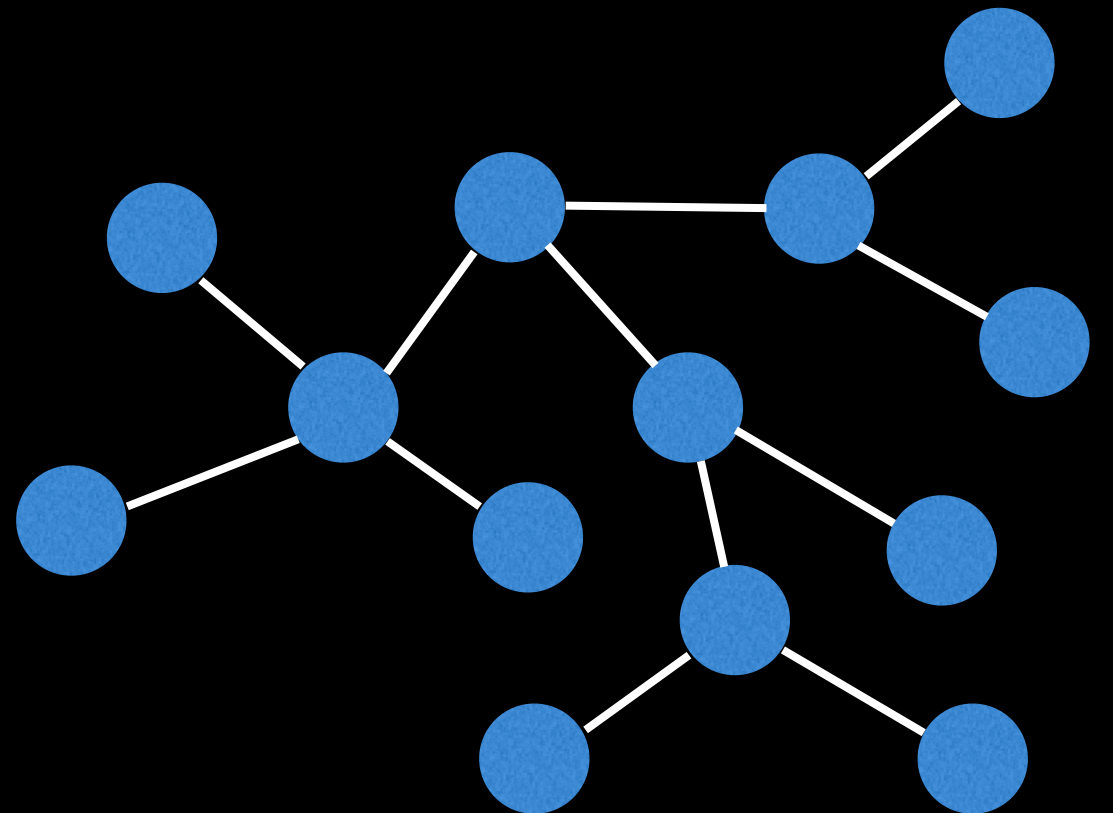
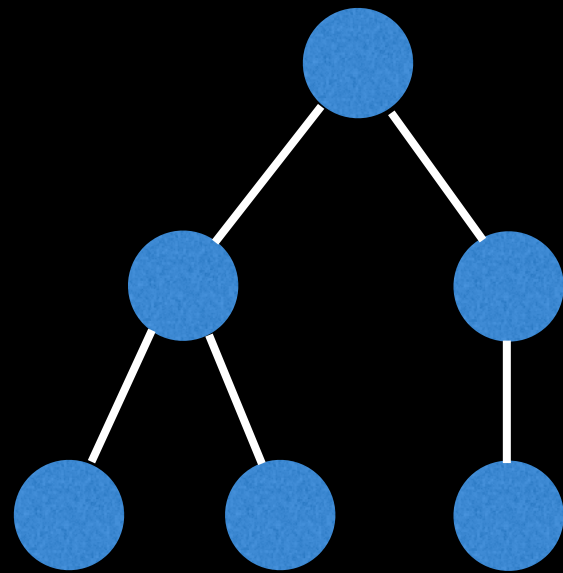
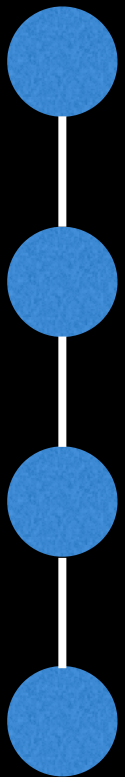


**NOTE:** I will usually denote an edge of such a graph as a triplet  $(u, v, w)$  and specify whether the graph is directed or undirected.

# Special Graphs

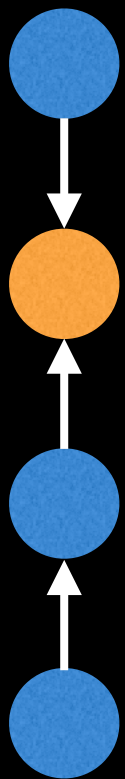
# Trees!

A **tree** is an **undirected graph with no cycles**.  
Equivalently, it is a connected graph with  $N$  nodes and  $N-1$  edges.

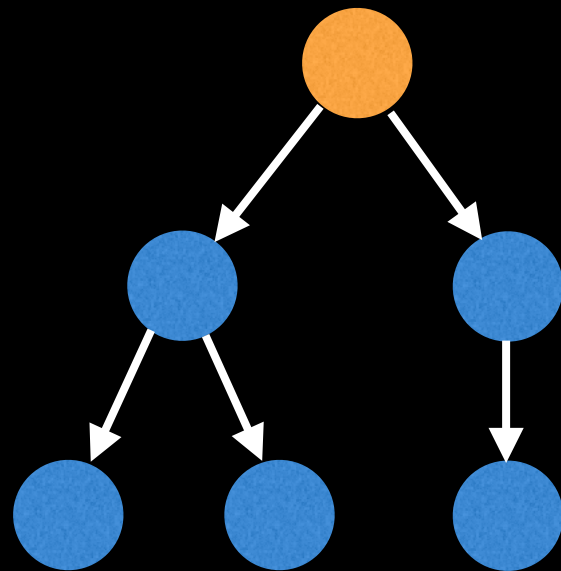


# Rooted Trees!

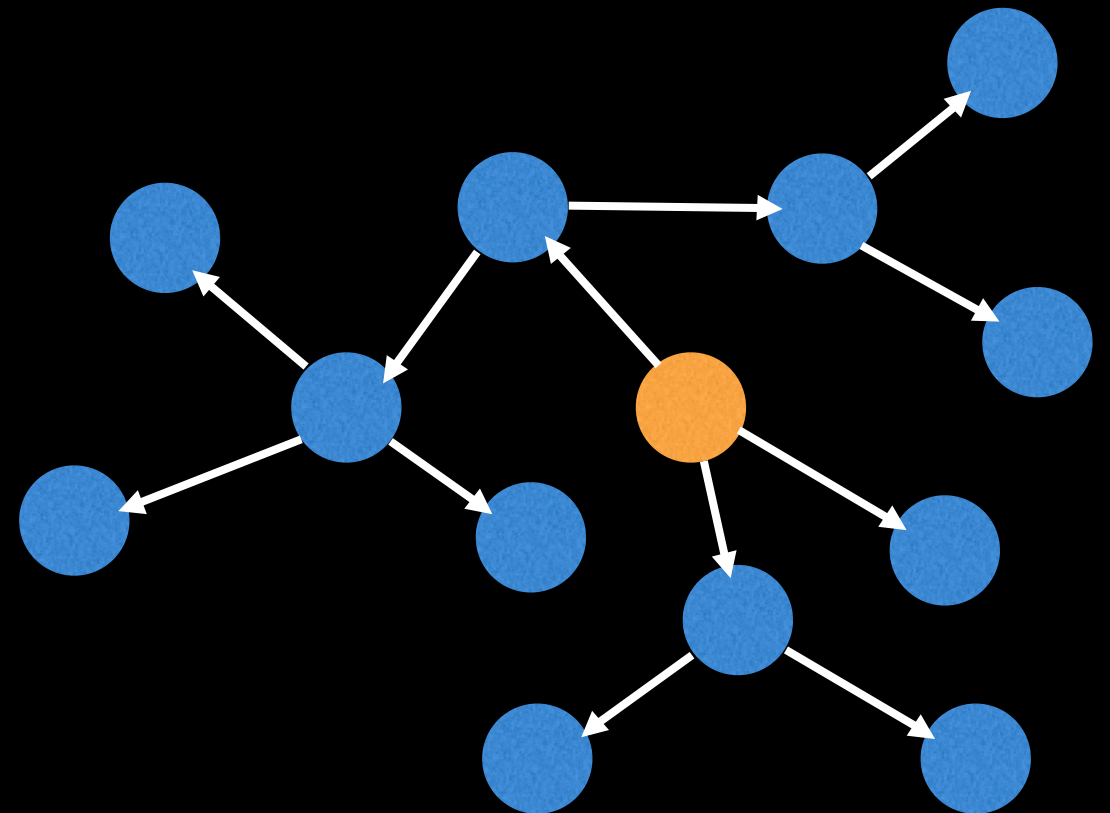
A **rooted tree** is a tree with a **designated root node** where every edge either points away from or towards the root node. When edges point away from the root the graph is called an **arborescence (out-tree)** and anti-arborescence (in-tree) otherwise.



In-tree



Out-tree



Out-tree

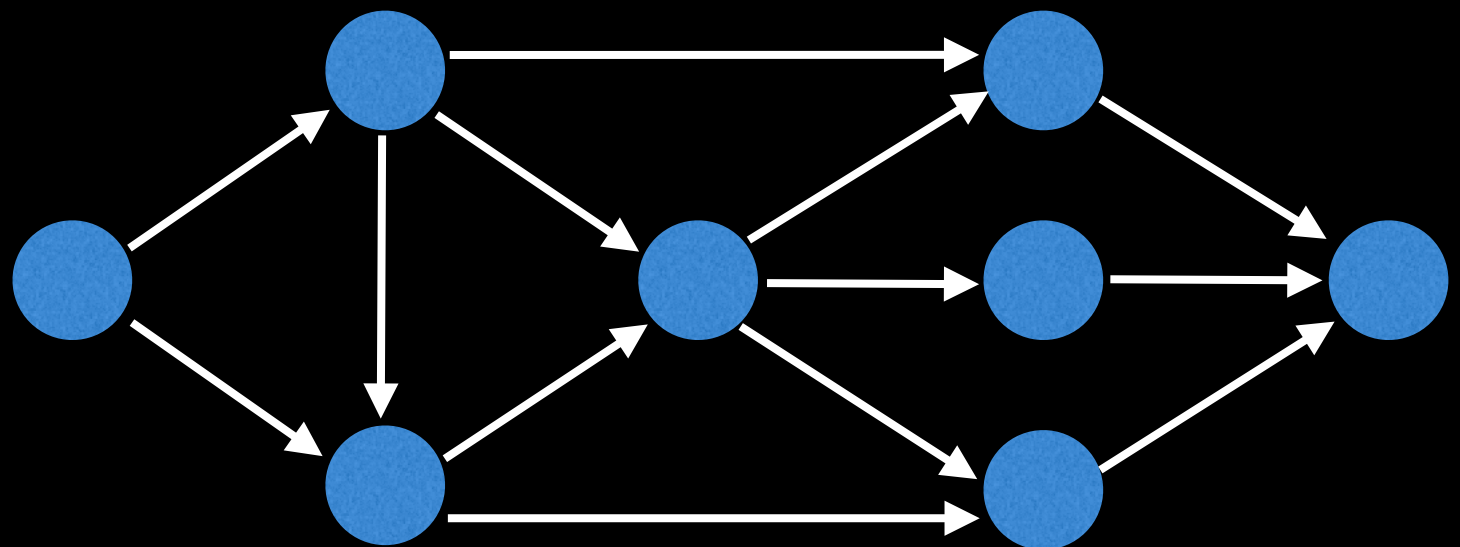
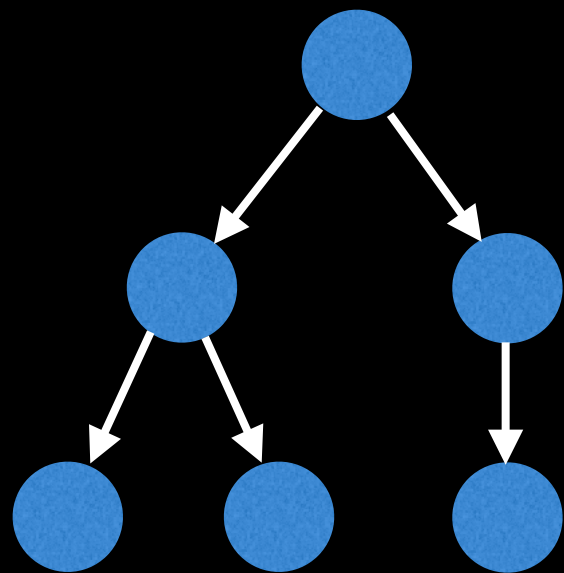
# Directed Acyclic Graphs (DAGs)

DAGs are **directed graphs with no cycles**.

These graphs play an important role in representing structures with dependencies.

Several efficient algorithms exist to operate on DAGs.

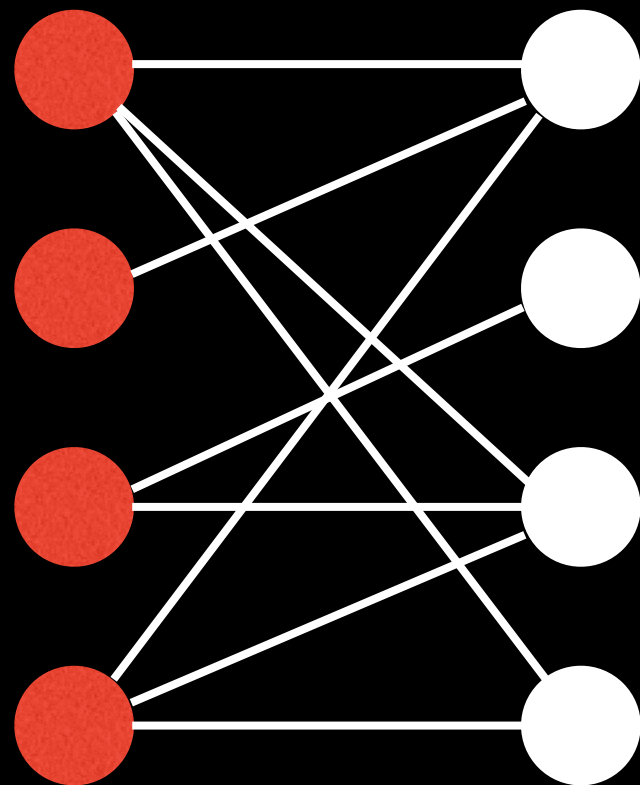
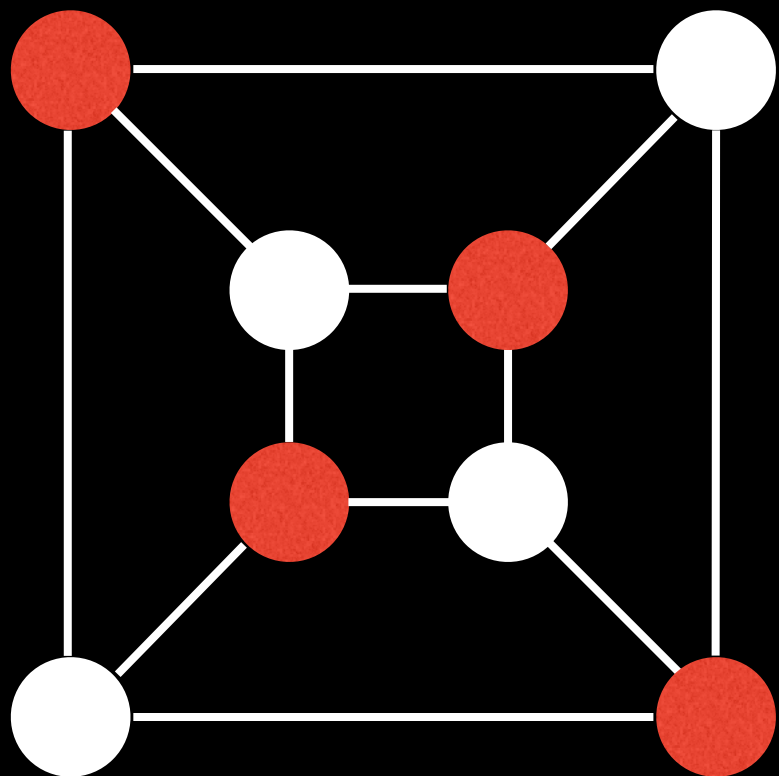
Cool fact: **All out-trees are DAGs** but **not all DAGs are out-trees**.



# Bipartite Graph

A **bipartite graph** is one whose *vertices* can be split into two independent groups  $U$ ,  $V$  such that every edge connects between  $U$  and  $V$ .

Other definitions exist such as: The graph is two colourable or there is no odd length cycle.



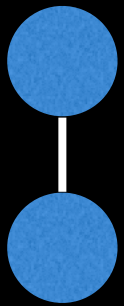
# Complete Graphs

A **complete graph** is one where there is a unique edge between every pair of nodes.

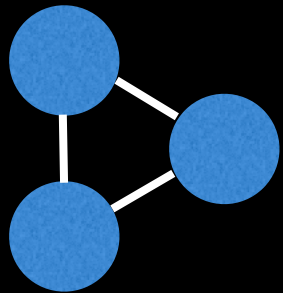
A complete graph with  $n$  vertices is denoted as the graph  $K_n$ .



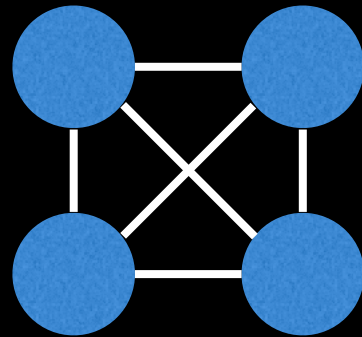
$K_1$



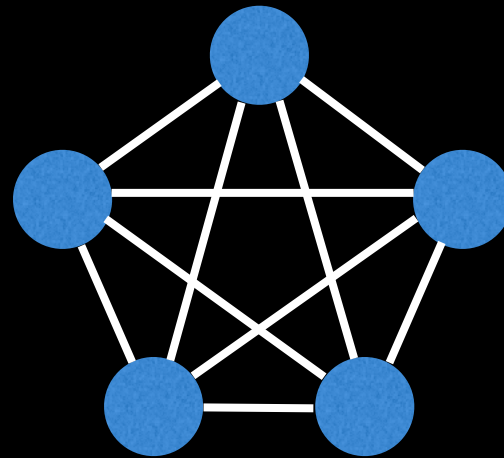
$K_2$



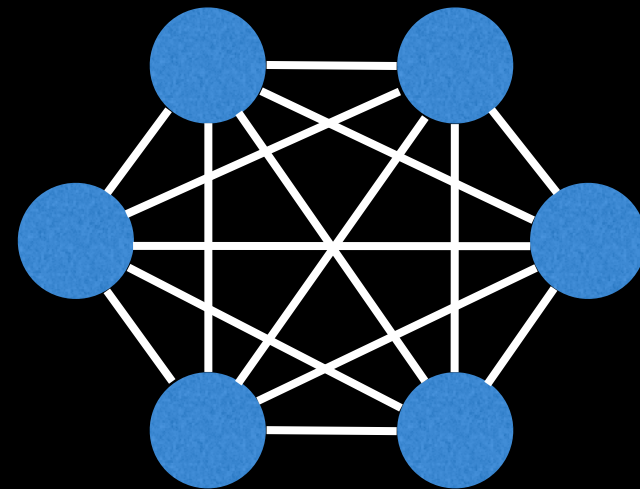
$K_3$



$K_4$



$K_5$



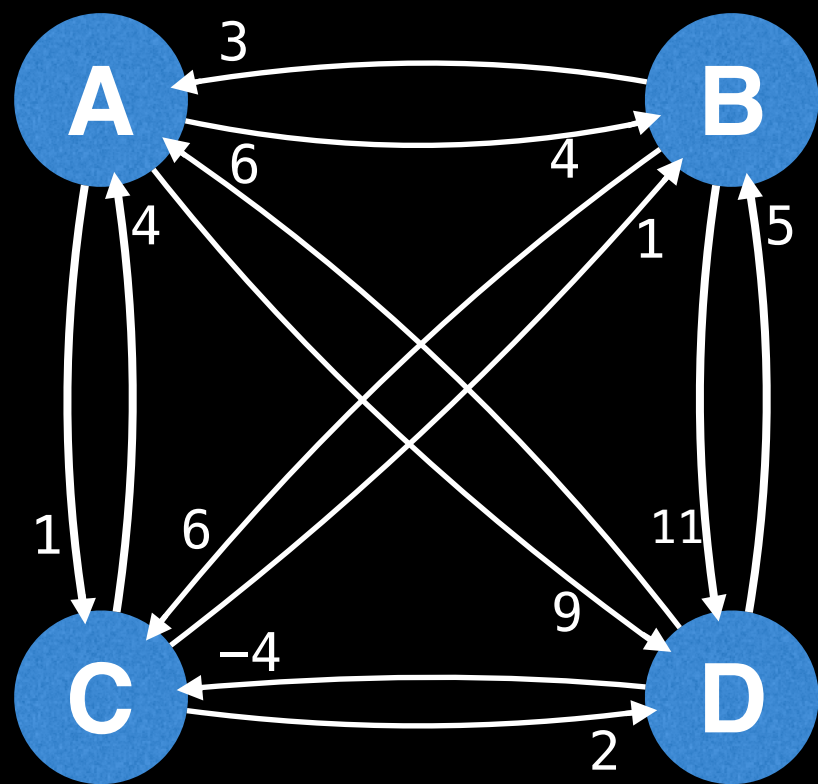
$K_6$



# Representing Graphs

# Adjacency Matrix

A **adjacency matrix**  $m$  is a very simple way to represent a graph. The idea is that the cell  $m[i][j]$  represents the edge weight of going from node  $i$  to node  $j$ .



	A	B	C	D
A	0	4	1	9
B	3	0	6	11
C	4	1	0	2
D	6	5	-4	0

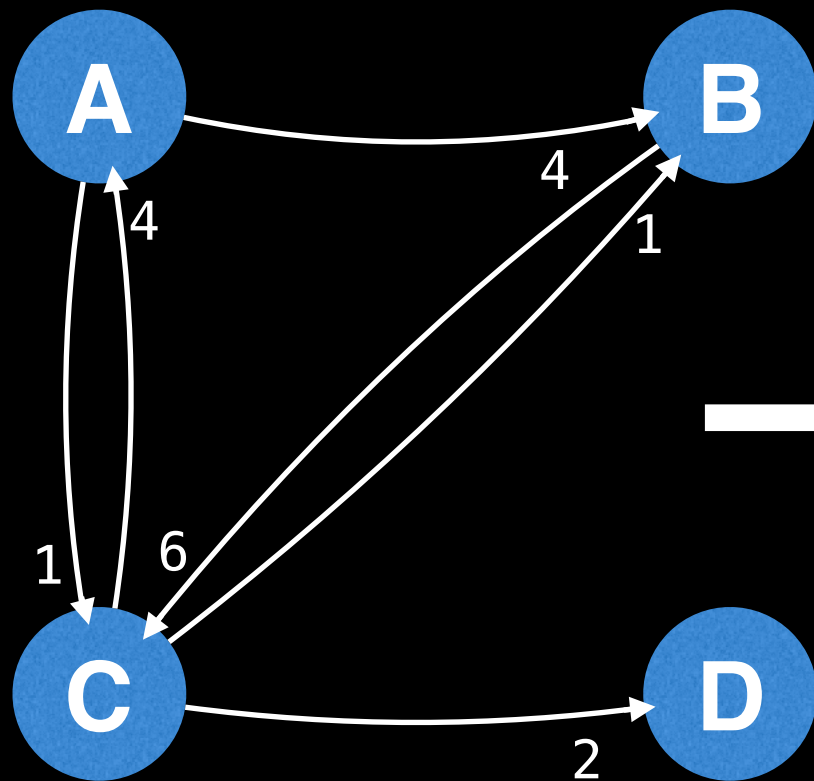
**NOTE:** It is often assumed that the edge of going from a node to itself has a cost of zero.

# Adjacency Matrix

Pros	Cons
Space efficient for representing dense graphs	Requires $\theta(V^2)$ space
Edge weight lookup is $\theta(1)$	Iterating over all edges takes $\theta(V^2)$ time
Simplest graph representation	

# Adjacency List

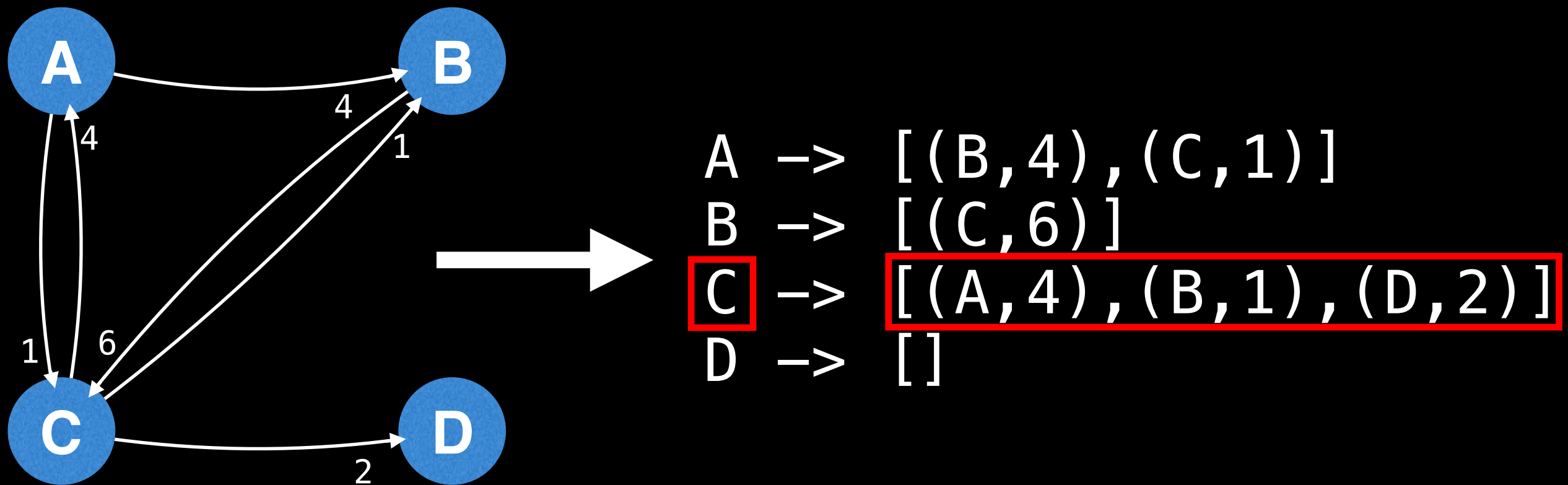
An **adjacency list** is a way to represent a graph as a map from nodes to lists of edges.



A → [(B, 4), (C, 1)]  
B → [(C, 6)]  
C → [(A, 4), (B, 1), (D, 2)]  
D → []

# Adjacency List

An **adjacency list** is a way to represent a graph as a map from nodes to lists of edges.



Node C can reach

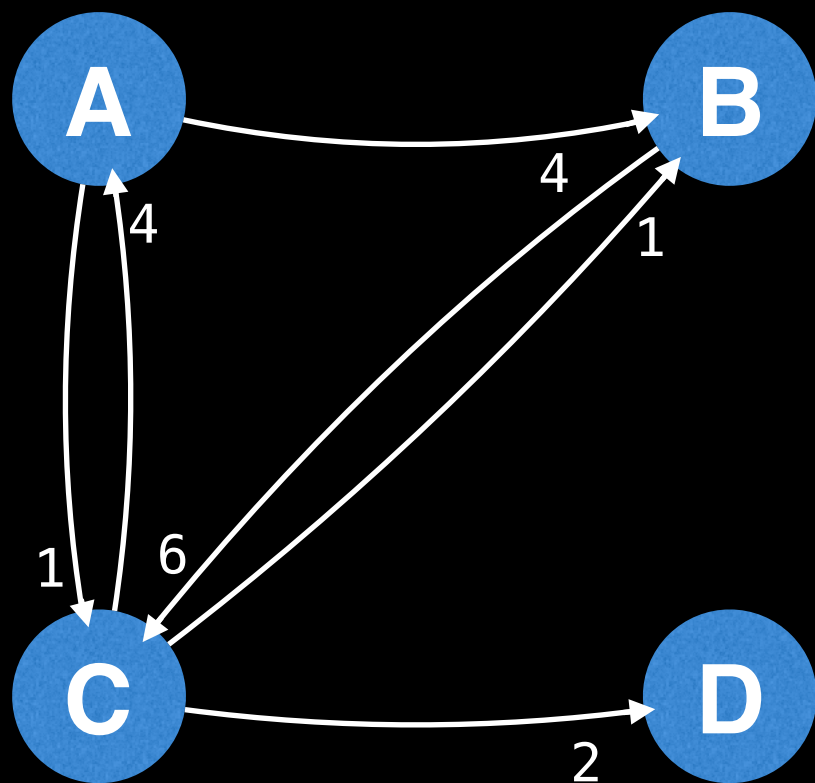
- Node A with cost 4
- Node B with cost 1
- Node D with cost 2

# Adjacency List

Pros	Cons
Space efficient for representing sparse graphs	Less space efficient for denser graphs.
Iterating over all edges is efficient	Edge weight lookup is $O(E)$
	Slightly more complex graph representation

# Edge List

An **edge list** is a way to represent a graph simply as an unordered list of edges. Assume the notation for any triplet  $(u,v,w)$  means: “the cost from node  $u$  to node  $v$  is  $w$ ”



→  $[(C,A,4), (A,C,1), (B,C,6), (A,B,4), (C,B,1), (C,D,2)]$

This representation is seldomly used because of its lack of structure. However, it is conceptually simple and practical in a handful of algorithms.

# Edge List

Pros	Cons
Space efficient for representing sparse graphs	Less space efficient for denser graphs.
Iterating over all edges is efficient	Edge weight lookup is $O(E)$
Very simple structure	



**Next Video: Graph Theory Problems**