



THE UNIVERSITY OF
MELBOURNE

COMP90038

Algorithms and Complexity

Lecture 6: Graphs and Graph Concepts
(with thanks to Harald Søndergaard)

Toby Murray



toby.murray@unimelb.edu.au



DMD 8.17 (Level 8, Doug McDonell Bldg)



<http://people.eng.unimelb.edu.au/tobym>



@tobycmurray

Graphs and Trees

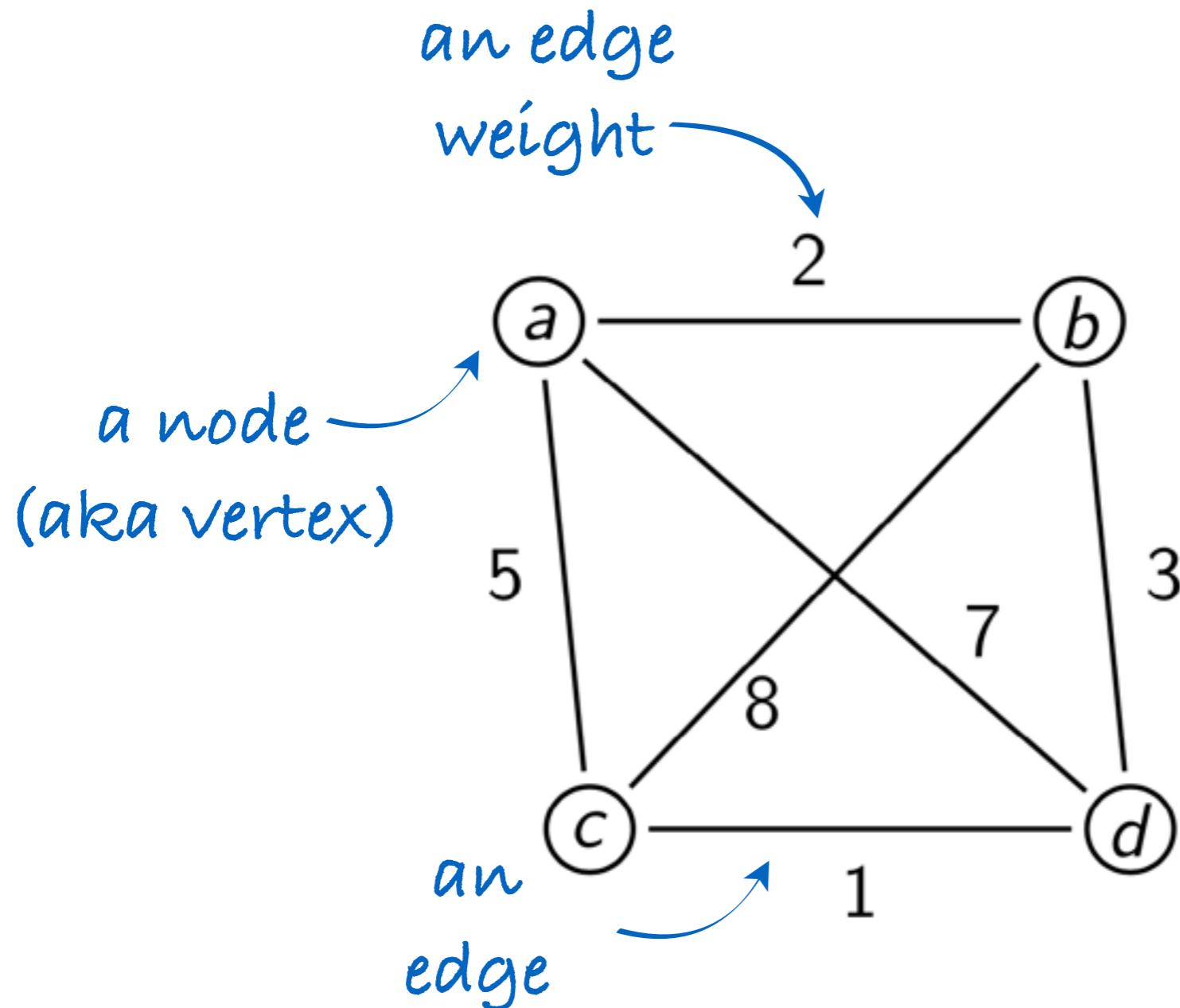


- One instance of the **exhaustive search** paradigm is **graph traversal**.
- After this lecture we shall look at two ways of systematically visiting every node of a graph, namely **depth-first** and **breadth-first search**.
- These two methods of graph traversal form the backbone of a surprisingly large number of useful graph algorithms.
- The graph algorithms are useful because of the large number of practical problems we can model as graph problems, in network design, flow design, planning, scheduling, route finding, and other logistics applications.
- Moreover, important numeric and logic problems can be **reduced** to graph problems—more on this in Week 12

Basic Graph Concepts

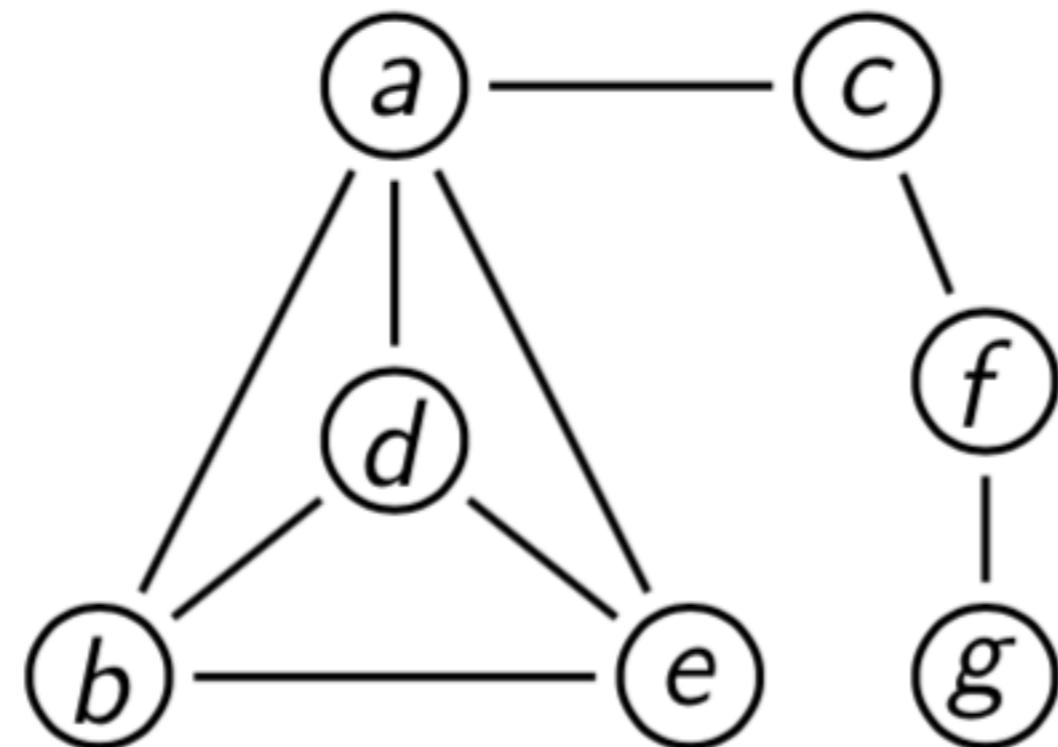


THE UNIVERSITY OF
MELBOURNE



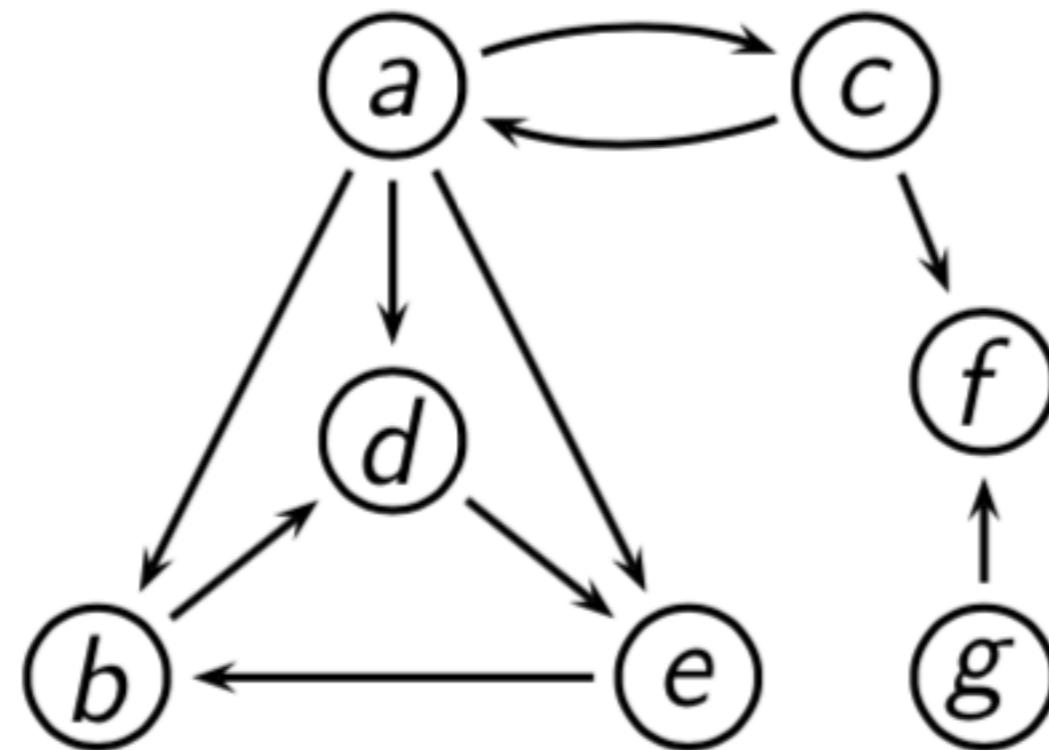
This graph is undirected since edges do not have directions associated with them.

Undirected Graph



(Edges do not have directions associated with them.)

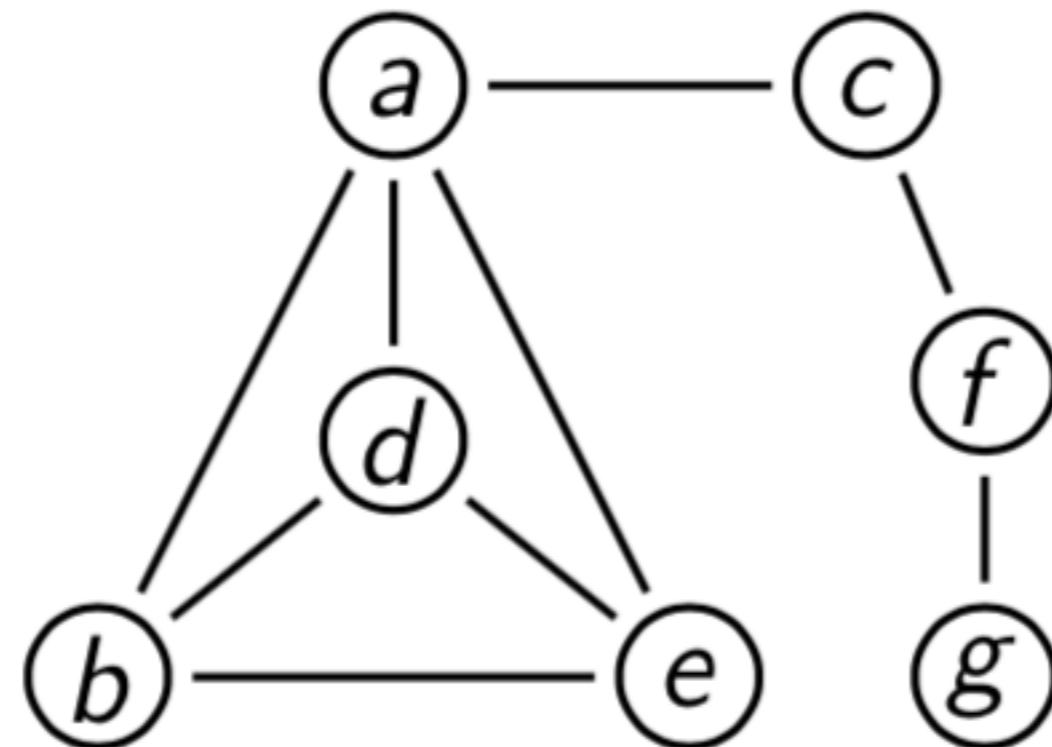
Directed Graph



(Each edge has an associated **direction**.)

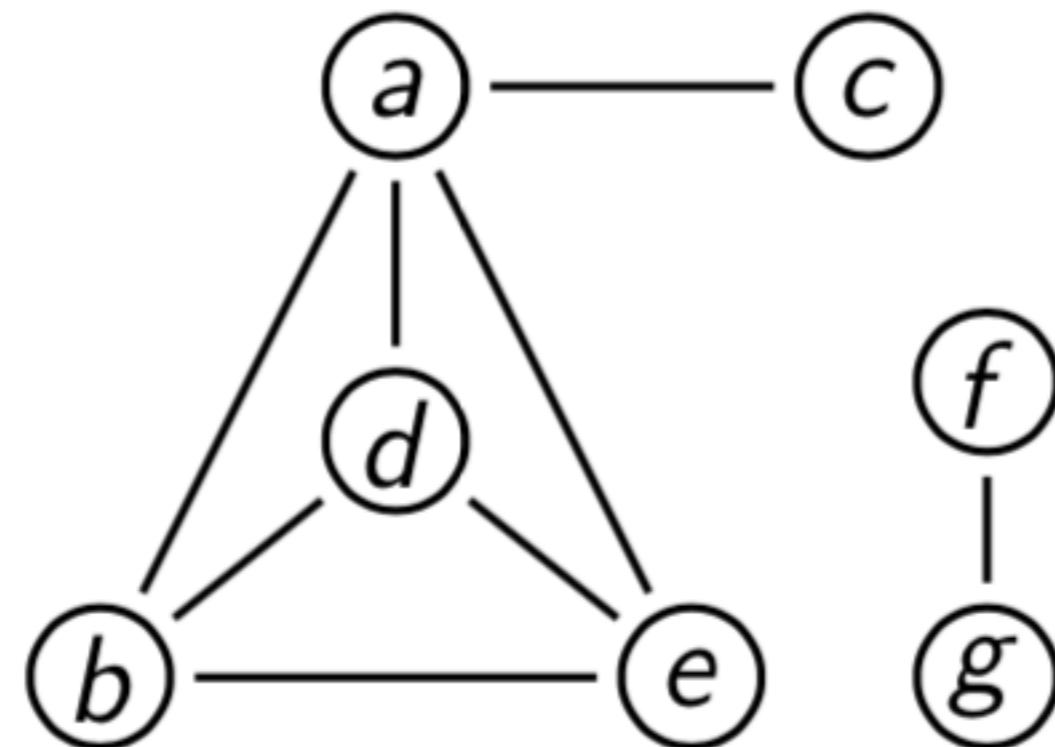


Connected Graph



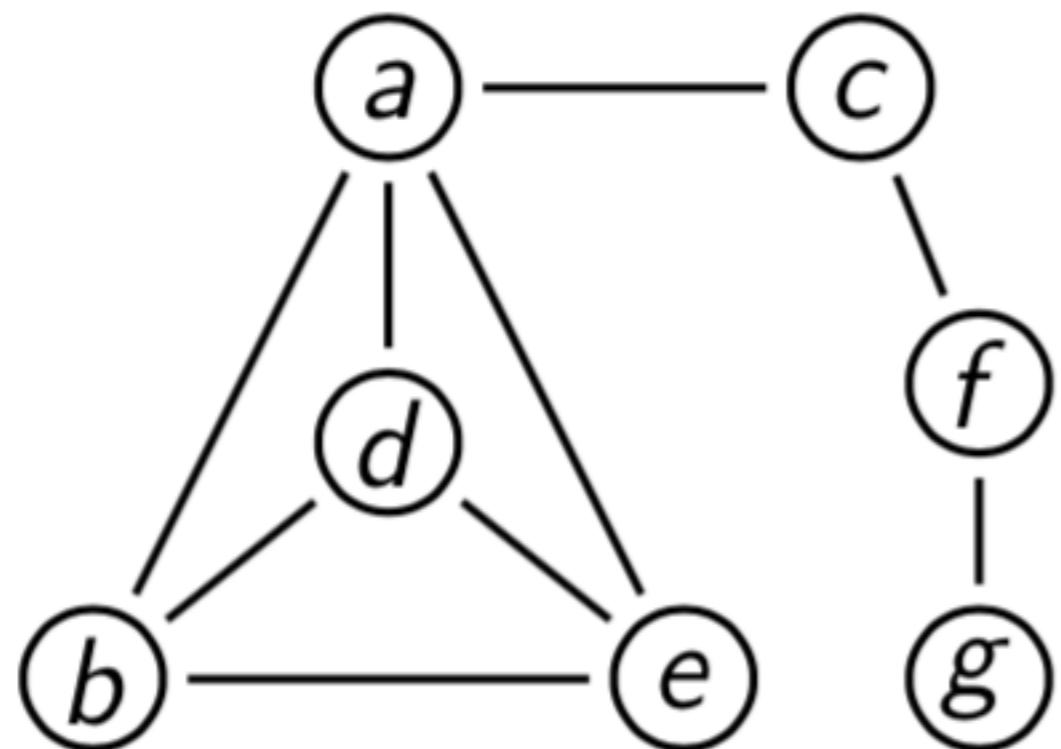
(Each node is reachable from all others
by following edges.)

Not Connected Graph, with 2 Components



Graphs, Mathematically

- A Graph **G** is a pair: $\langle \mathbf{V}, \mathbf{E} \rangle$
 - **V** : set of **nodes** (aka **vertices**)
 - **E** : set of **edges** (a binary relation on **V**)
 - $(u, v) \in \mathbf{E}$ means there is an edge from u to v

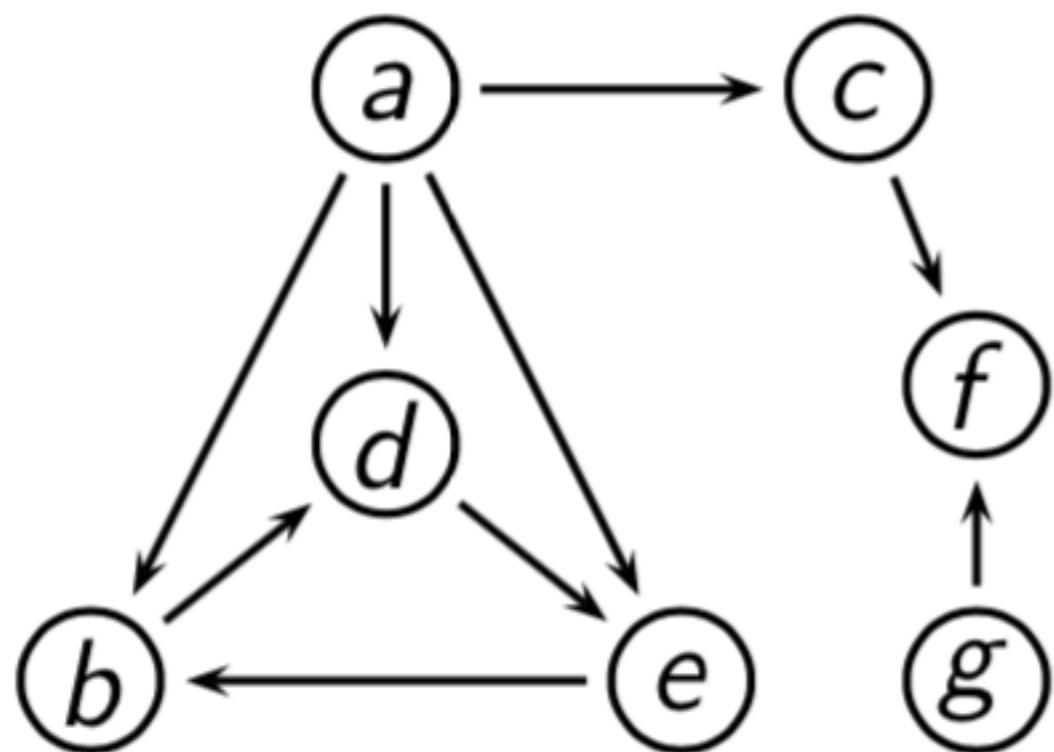


$$\mathbf{V} = \{a, b, c, d, e, f, g\}$$

E = symmetric closure of
 $\{(a, b), (a, c), (a, d),$
 $(a, e), (b, d), (b, e),$
 $(c, f), (d, e), (f, g)\}$

Graphs, Mathematically

- A Graph **G** is a pair: $\langle \mathbf{V}, \mathbf{E} \rangle$
 - **V** : set of **nodes** (aka **vertices**)
 - **E** : set of **edges** (a binary relation on **V**)
 - $(u, v) \in \mathbf{E}$ means there is an edge from u to v

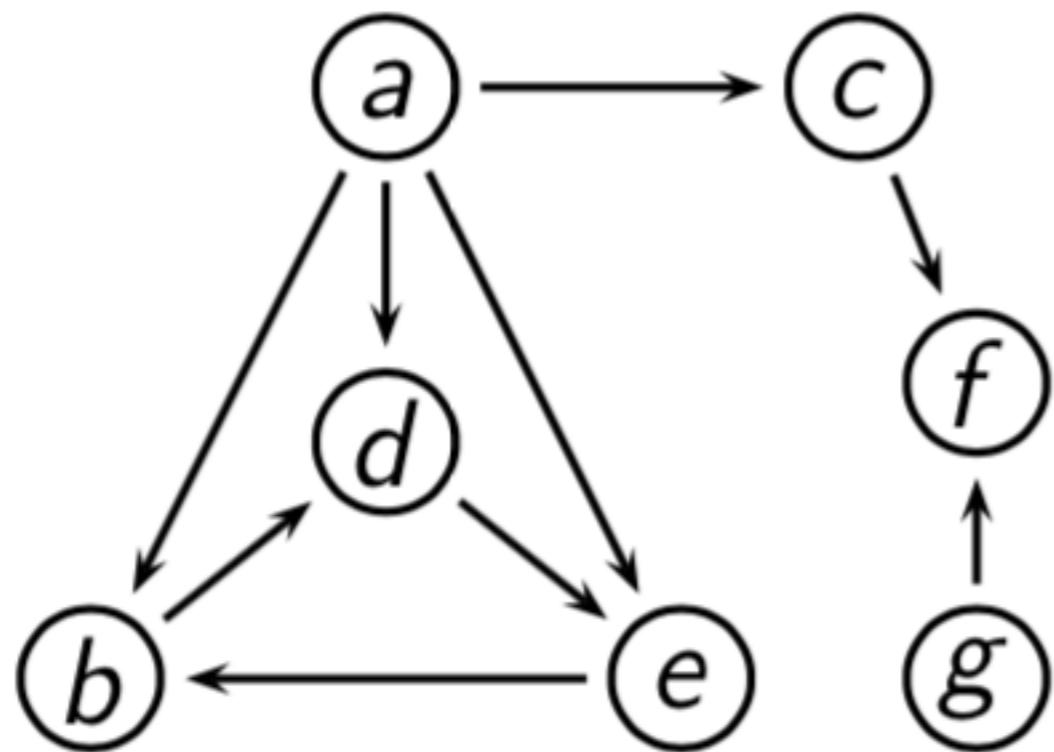


$$\mathbf{V} = \{a, b, c, d, e, f, g\}$$

$$\mathbf{E} = \{(a, b), (a, c), (a, d), (a, e), (b, d), (c, f), (d, e), (e, b), (g, f)\}$$

Degrees of Nodes

- If $(v, u) \in E$ then v and u are **adjacent**, or **neighbours**
- Edge (v, u) is **incident** on, or **connects**, v and u
- **Degree of a node v :** number of edges incident on v



For connected graphs:

In-degree of v : number of edges going to v

Out-degree of v : number of edges leaving from v

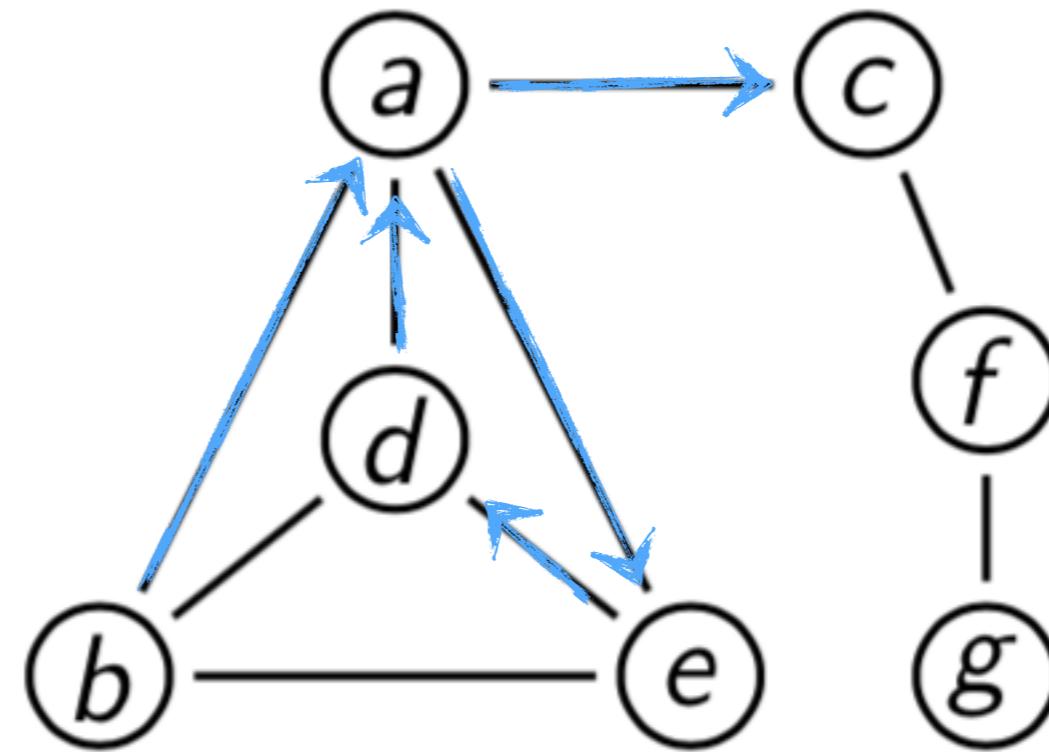
Paths



THE UNIVERSITY OF
MELBOURNE

A path:

b, a, e, d, a, c



A **path** in $\langle V, E \rangle$ is a sequence of nodes v_0, v_1, \dots, v_k from V , so that each $(v_i, v_{i+1}) \in E$.

The path v_0, v_1, \dots, v_k has **length** k .

A **simple** path is one that has no repeated nodes.

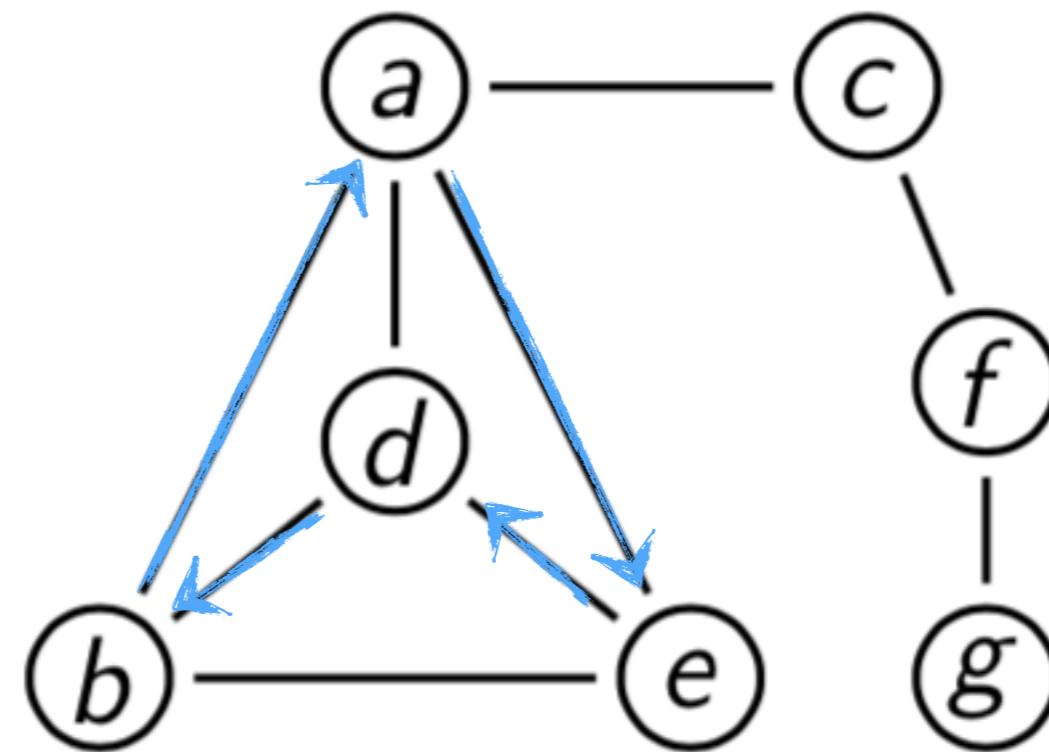
Cycles



THE UNIVERSITY OF
MELBOURNE

A cycle:

b, a, e, d, b



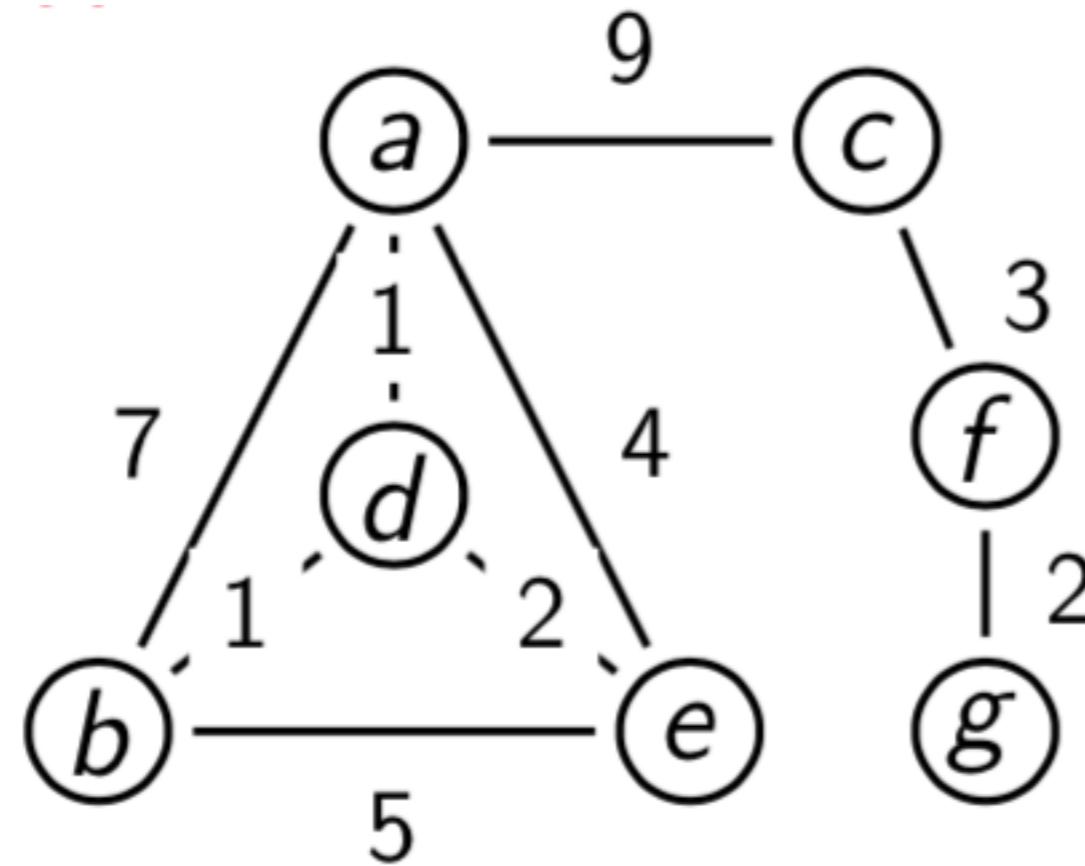
A **cycle** is like a simple path that starts and finishes at the **same** node.

(Cycles turn out to be very important for lots of applications.)

Weighted Graphs



THE UNIVERSITY OF
MELBOURNE



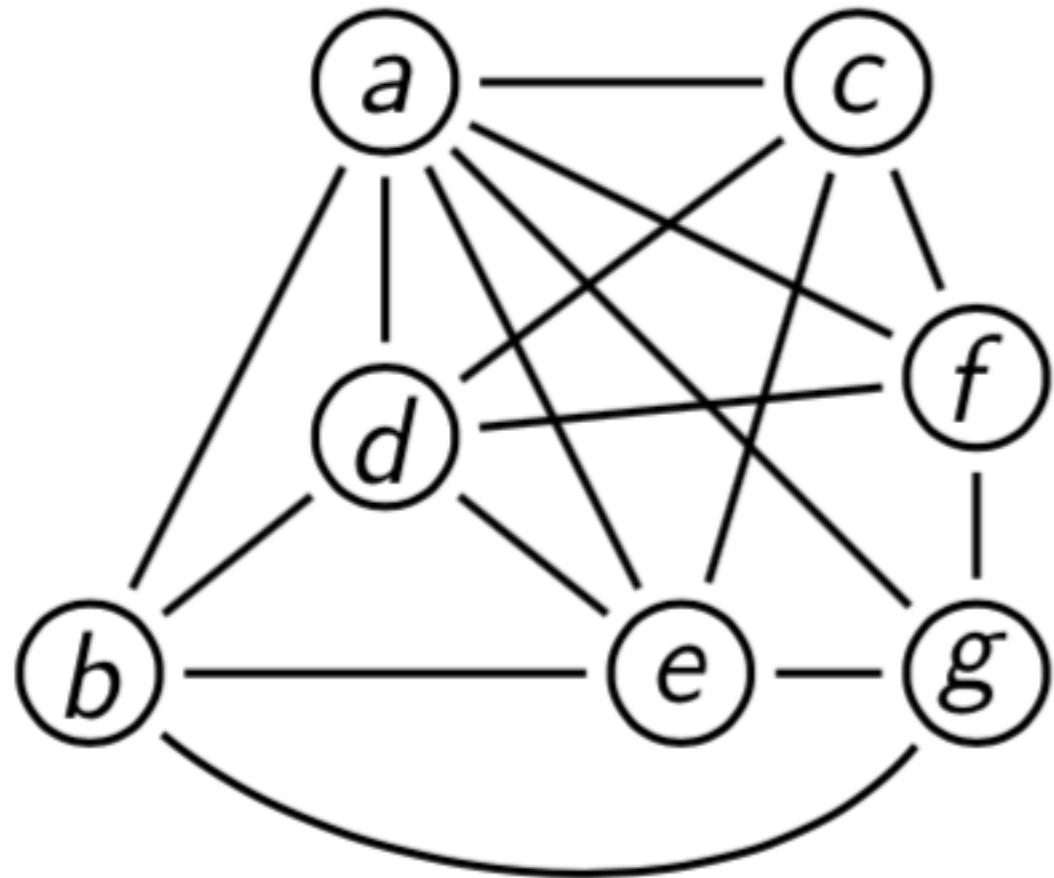
Each edge (v,u) has a **weight** indicating some information about that connection from v to u

The information depends on what the graph represents:
network congestion, physical distance, cost, etc.

Dense vs Sparse Graphs

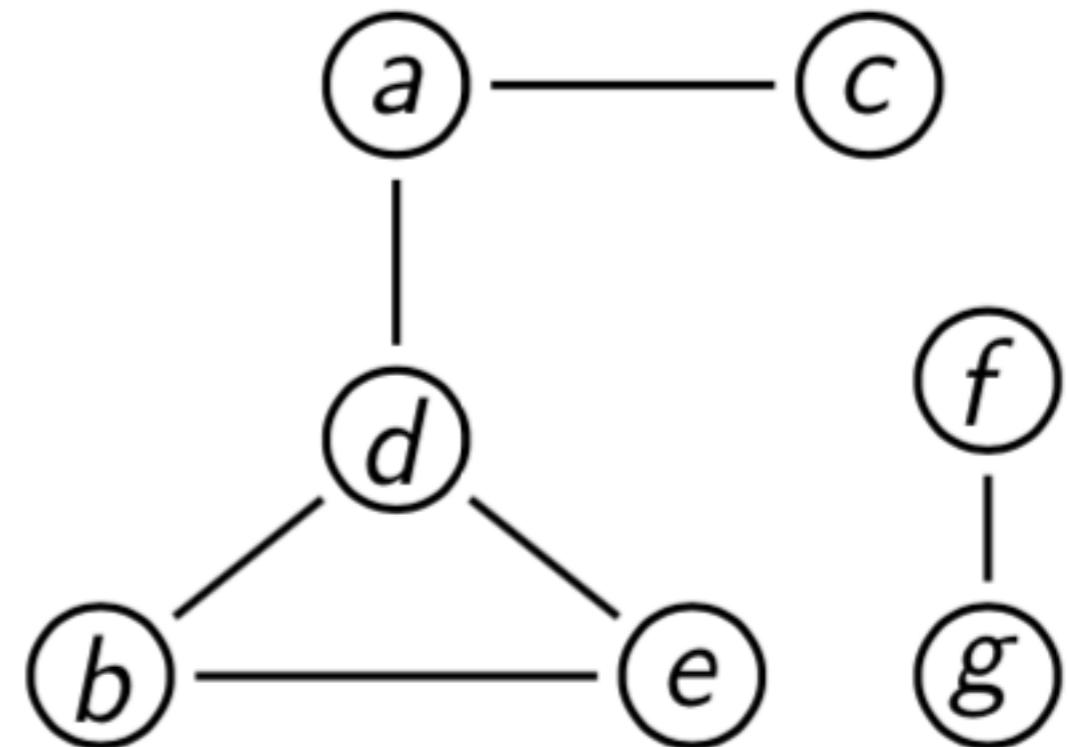


THE UNIVERSITY OF
MELBOURNE



Dense Graph

(lots of edges, relative
to number of nodes)



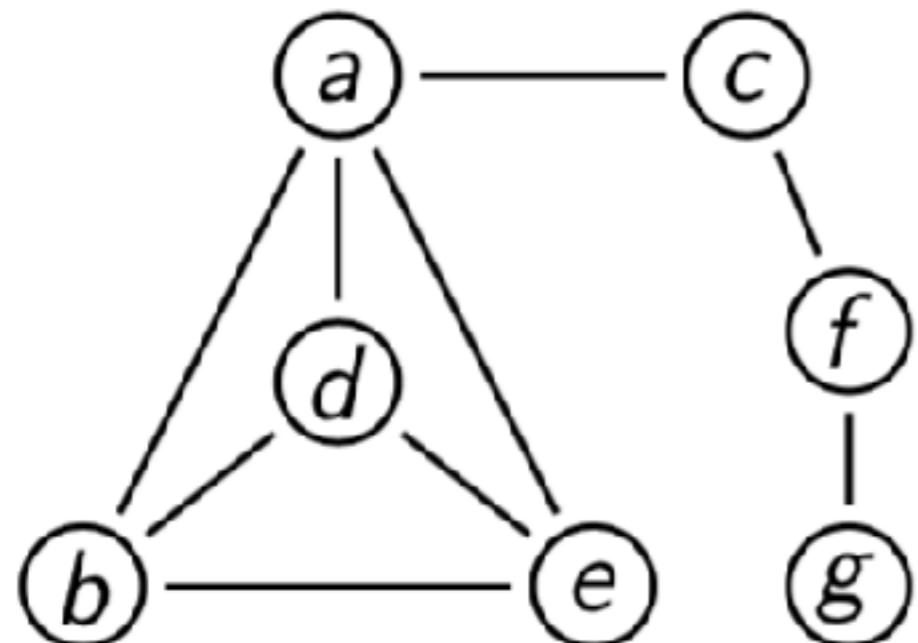
Sparse Graph

(few edges, relative
to number of nodes)

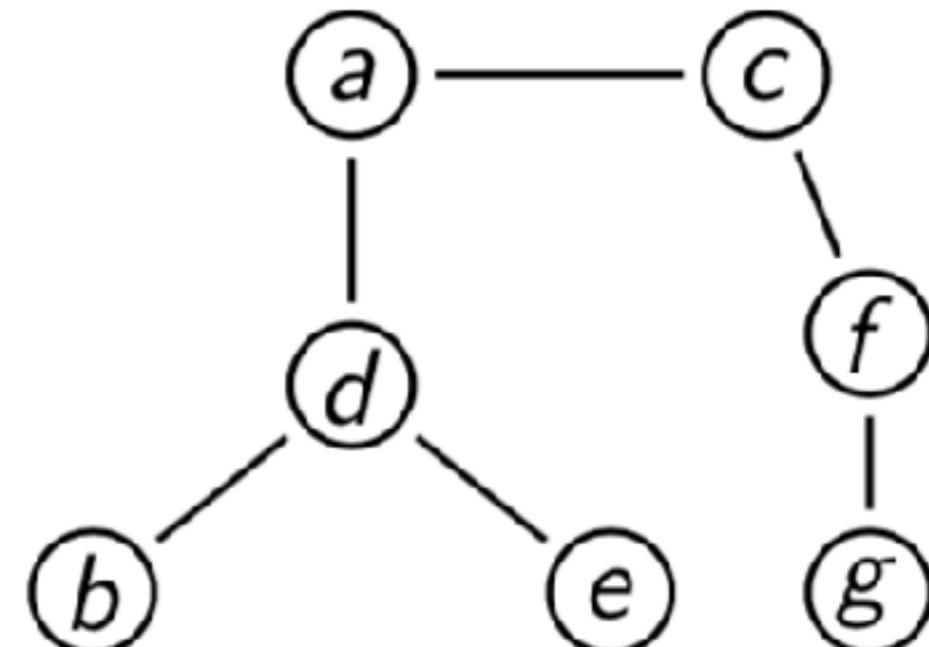
Cyclic vs Acyclic



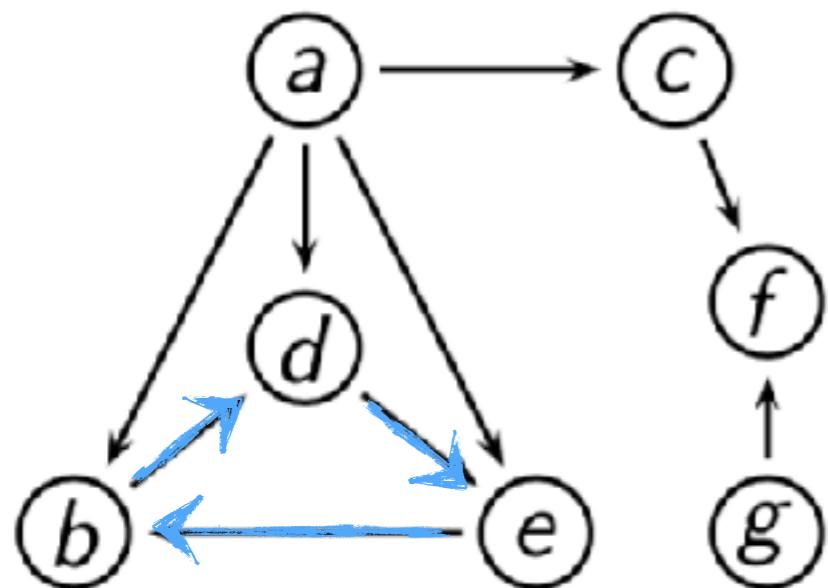
THE UNIVERSITY OF
MELBOURNE



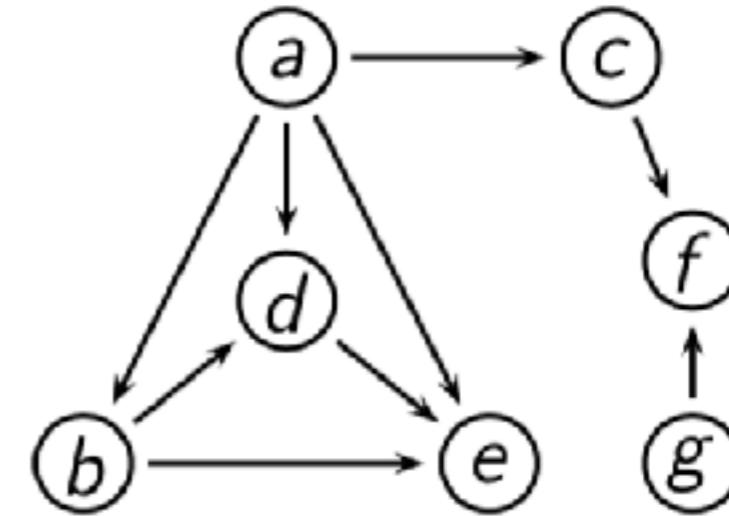
Cyclic



Acyclic



Directed Cyclic



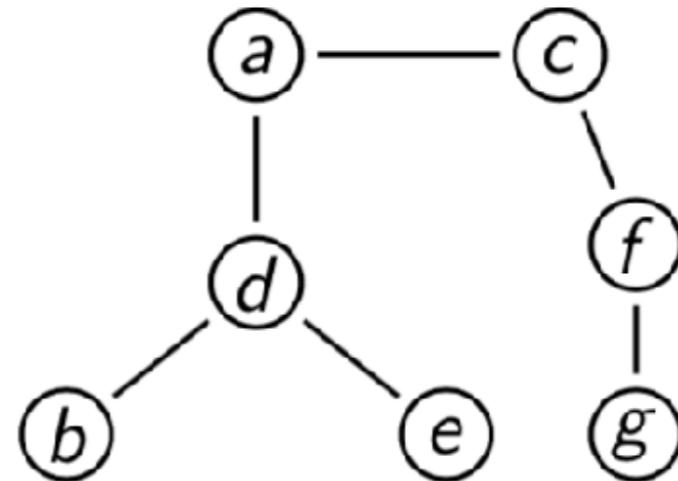
**Directed Acyclic Graph
(a dag)**

Rooted Trees

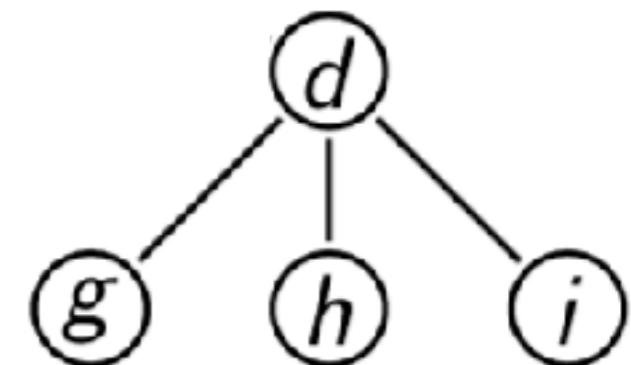
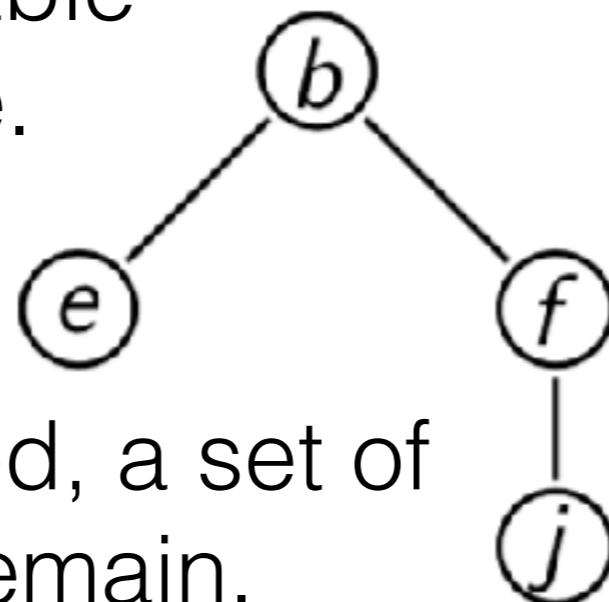


THE UNIVERSITY OF
MELBOURNE

A (free) **tree** is a connected, acyclic graph, e.g.



A **rooted tree** is a tree with one node (the **root**) identified as special. Every other node is reachable from the root node.



When the root is removed, a set of rooted (sub-)trees remain.

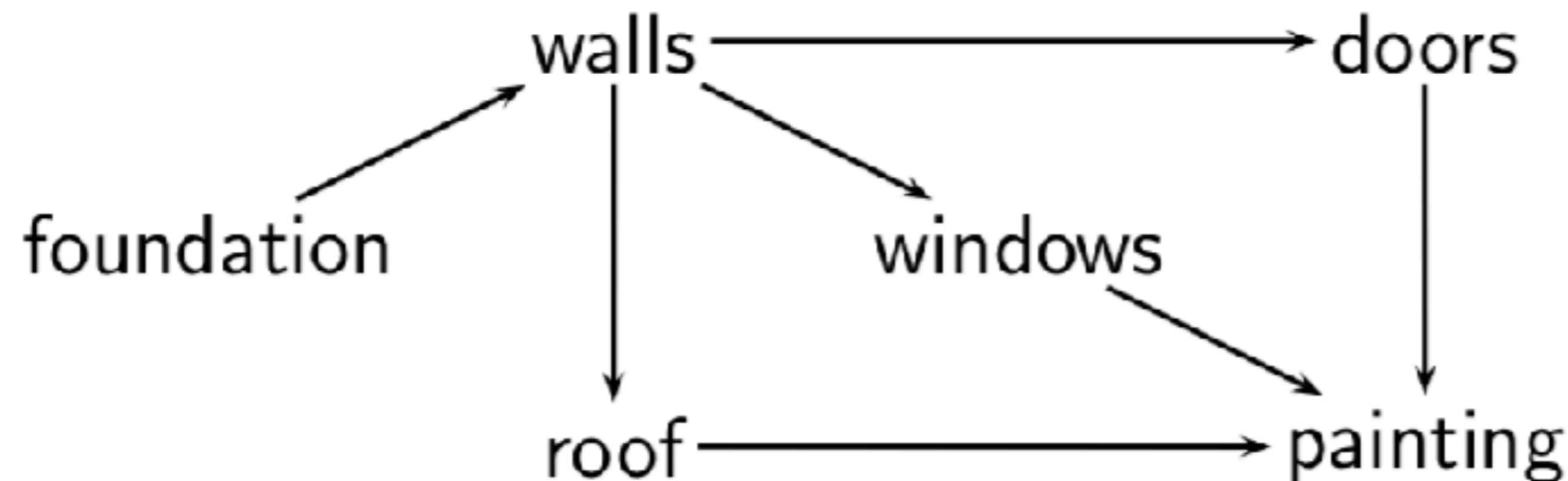
Modelling with Graphs



THE UNIVERSITY OF
MELBOURNE

Graph algorithms are of great importance because so many different problem types can be abstracted to graph problems.

For example, directed graphs (they'd better be dags) are central in scheduling problems:

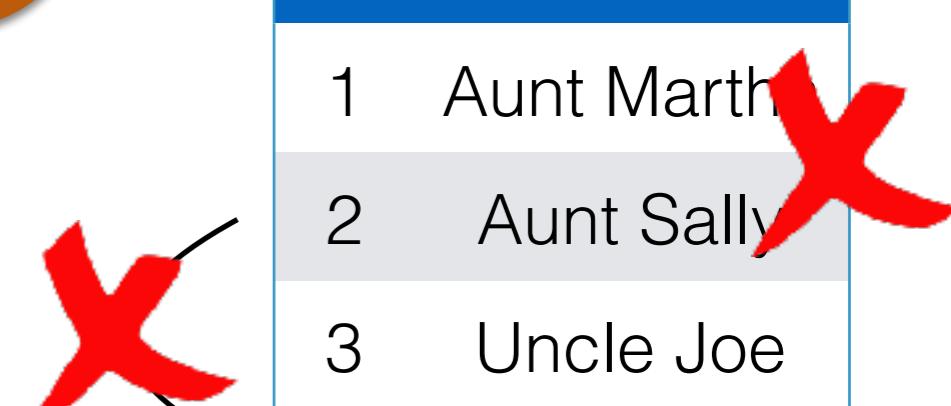


Modelling with Graphs



THE UNIVERSITY OF
MELBOURNE

Imagine I'm doing the seating plan for a wedding.

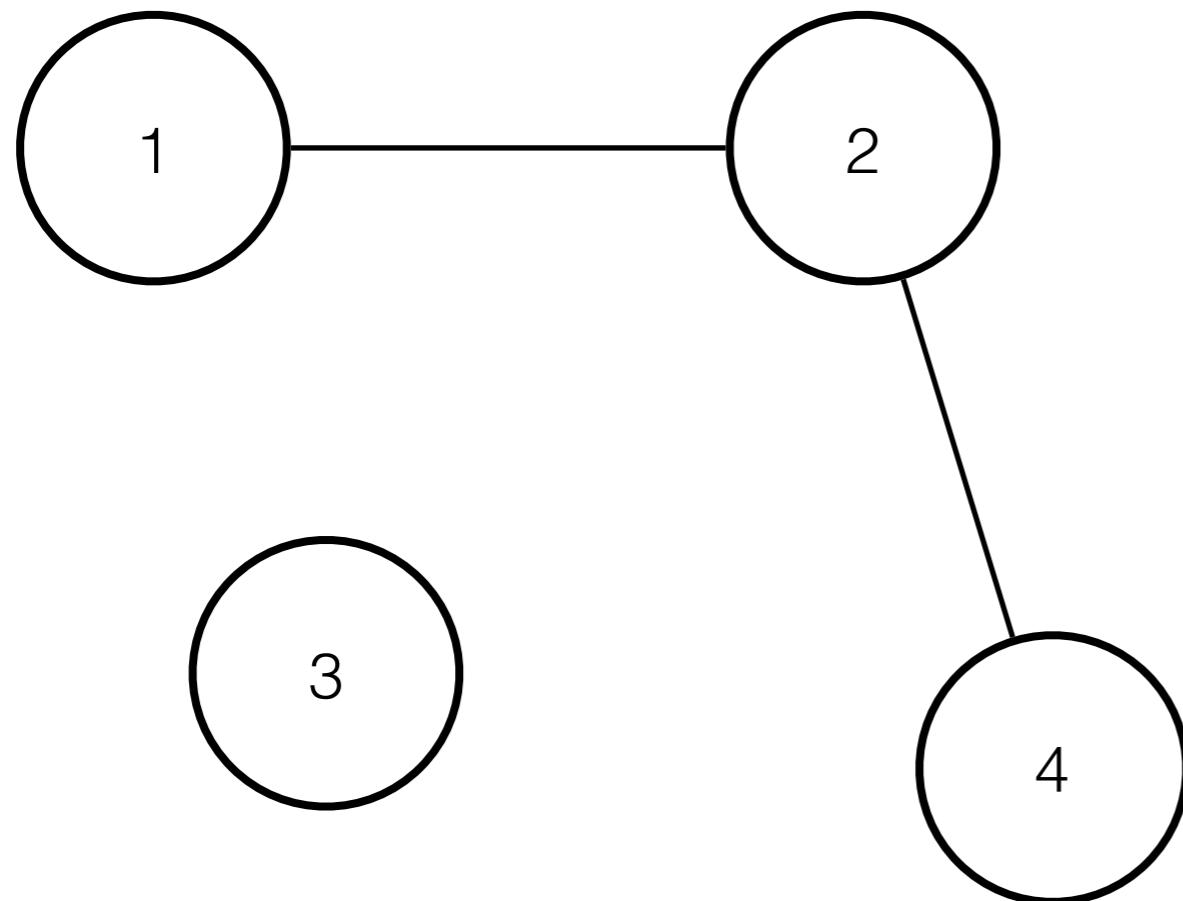


Modelling with Graphs



THE UNIVERSITY OF
MELBOURNE

Each person becomes a node. An edge between v and u means v and u cannot sit together.



Now colour the nodes so that no two adjacent nodes have the same colour.

Guest List	
1	Aunt Martha
2	Aunt Sally
3	Uncle Joe
4	Cousin Andy
...

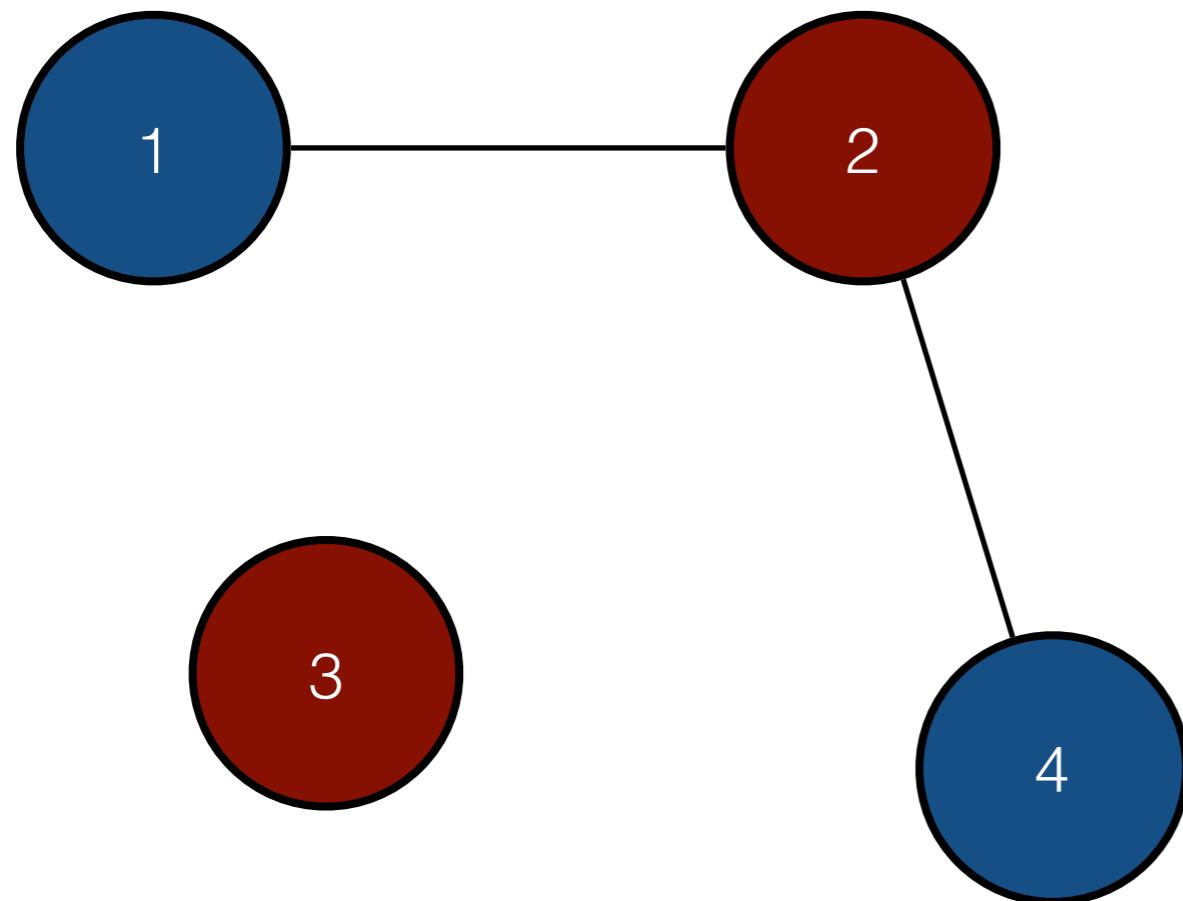
Red X marks are placed over the second row (Aunt Sally) and the fifth row (Cousin Andy) of the guest list table.

Modelling with Graphs



THE UNIVERSITY OF
MELBOURNE

Each person becomes a node. An edge between v and u means v and u cannot sit together.



Now colour the nodes so that no two adjacent nodes have the same colour.

Guest List	
1	Aunt Martha
2	Aunt Sally
3	Uncle Joe
4	Cousin Andy
...

Modelling with Graphs



Seating planning with k tables can be
reduced to the graph k -coloring problem:

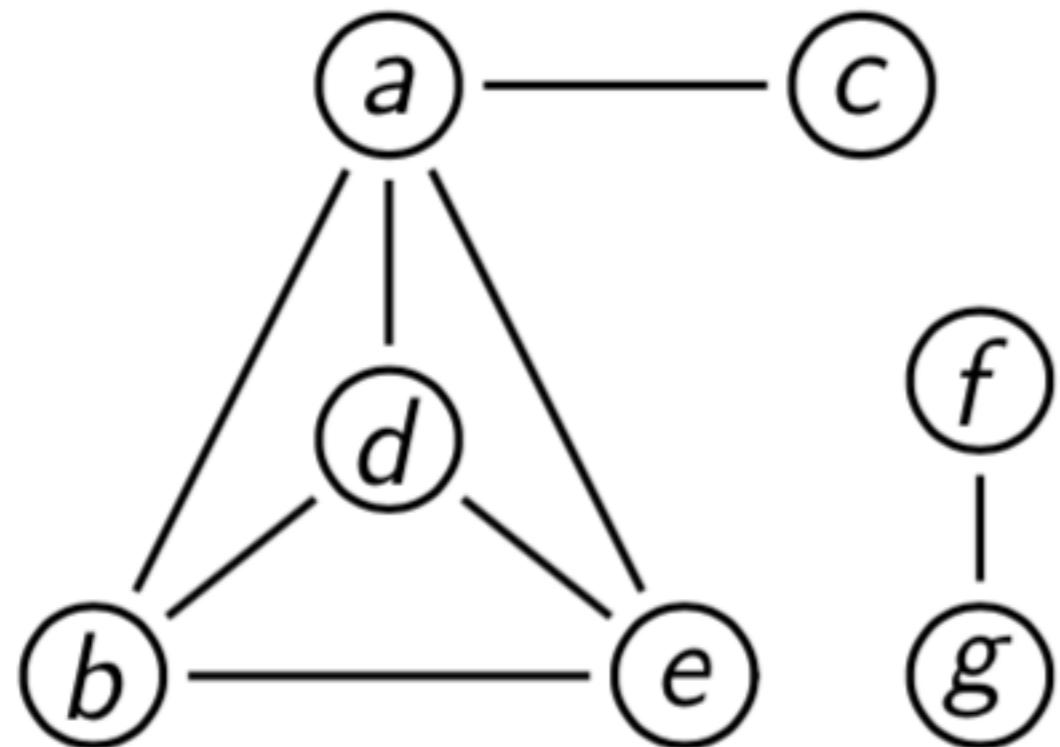
Find, if possible, a colouring of nodes so that no
two connected nodes get the same colour.

Lots of other problems can be
reduced to graph colouring.

Graph Representations: Undirected Graphs



THE UNIVERSITY OF
MELBOURNE



	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>
<i>a</i>	0	1	1	1	1	0	0
<i>b</i>	1	0	0	1	1	0	0
<i>c</i>	1	0	0	0	0	0	0
<i>d</i>	1	1	0	0	1	0	0
<i>e</i>	1	1	0	1	0	0	0
<i>f</i>	0	0	0	0	0	0	1
<i>g</i>	0	0	0	0	0	1	0

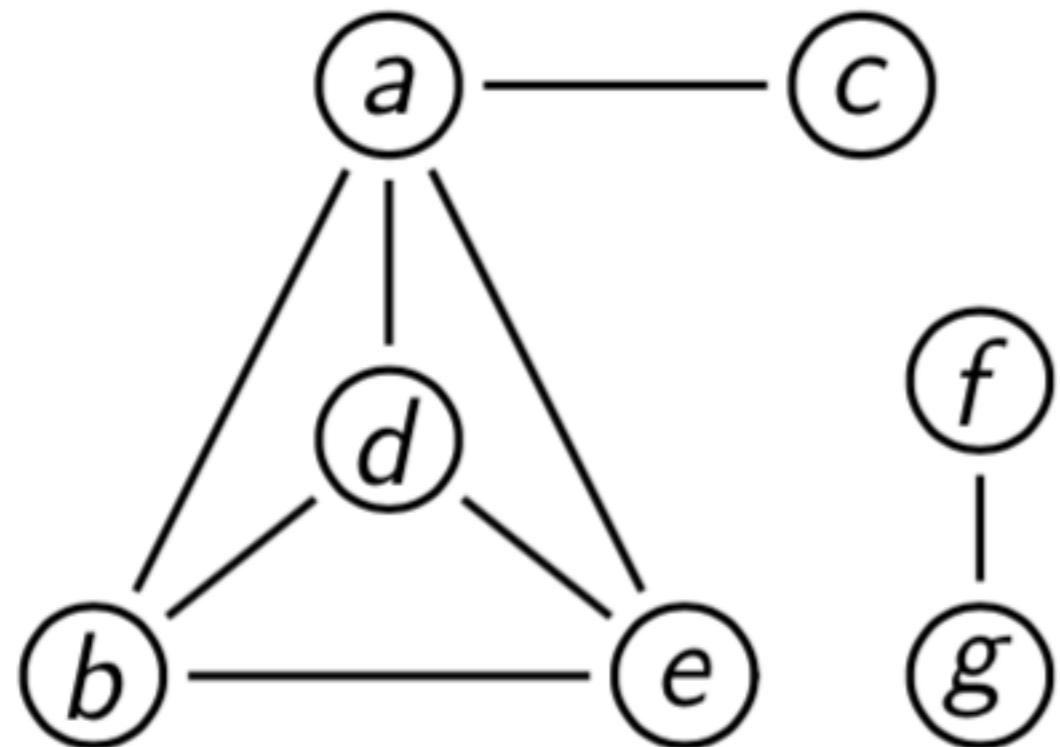
Adjacency Matrix

For an undirected graph, this matrix is **symmetric** about the diagonal.

Graph Representations: Undirected Graphs



THE UNIVERSITY OF
MELBOURNE



a	→ b → c → d → e
b	→ a → d → e
c	→ a
d	→ a → b → e
e	→ a → b → d
f	→ g
g	→ f

Adjacency List

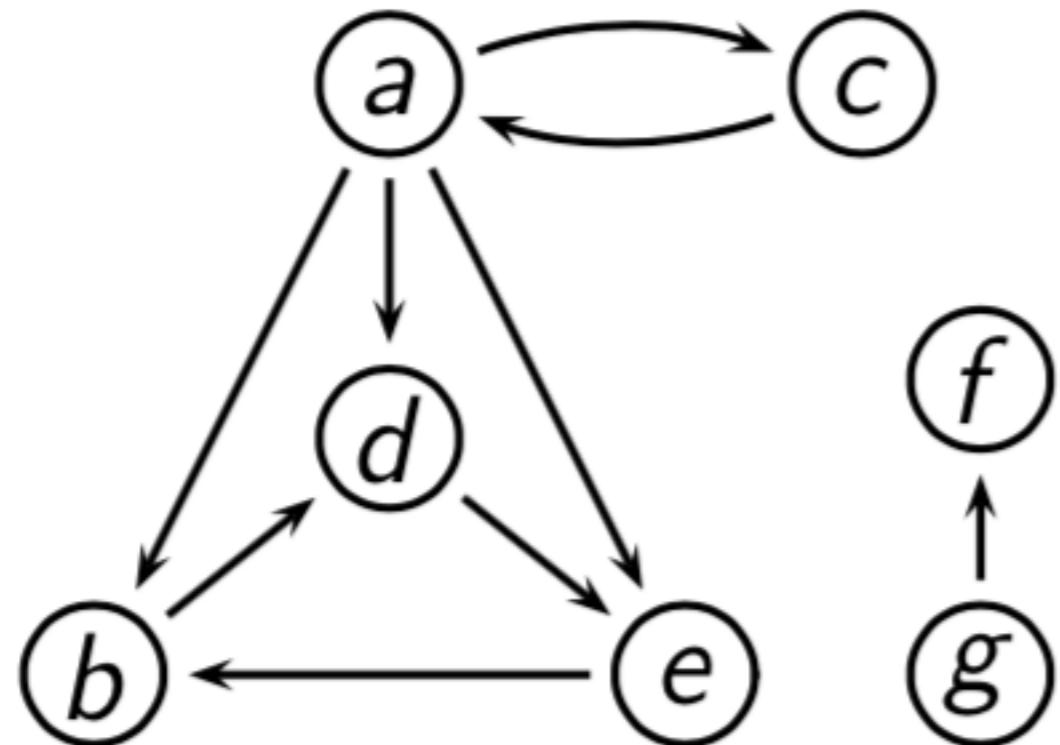
An **array of linked lists**

(Assuming lists are kept in sorted order.)

Graph Representations: Directed Graphs



THE UNIVERSITY OF
MELBOURNE



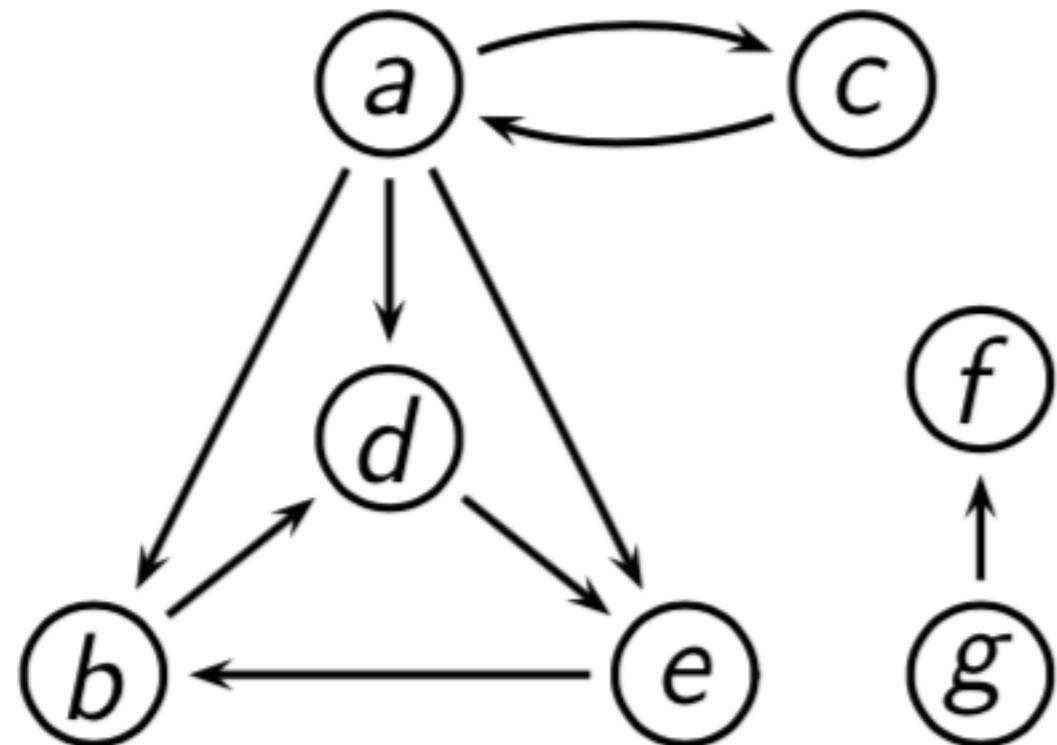
	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>
<i>a</i>	0	1	1	1	1	0	0
<i>b</i>	0	0	0	1	0	0	0
<i>c</i>	1	0	0	0	0	0	0
<i>d</i>	0	0	0	0	1	0	0
<i>e</i>	0	1	0	0	0	0	0
<i>f</i>	0	0	0	0	0	0	0
<i>g</i>	0	0	0	0	0	1	0

Adjacency Matrix

Graph Representations: Directed Graphs



THE UNIVERSITY OF
MELBOURNE



a	→ b → c → d → e
b	→ d
c	→ a
d	→ e
e	→ b
f	
g	→ f

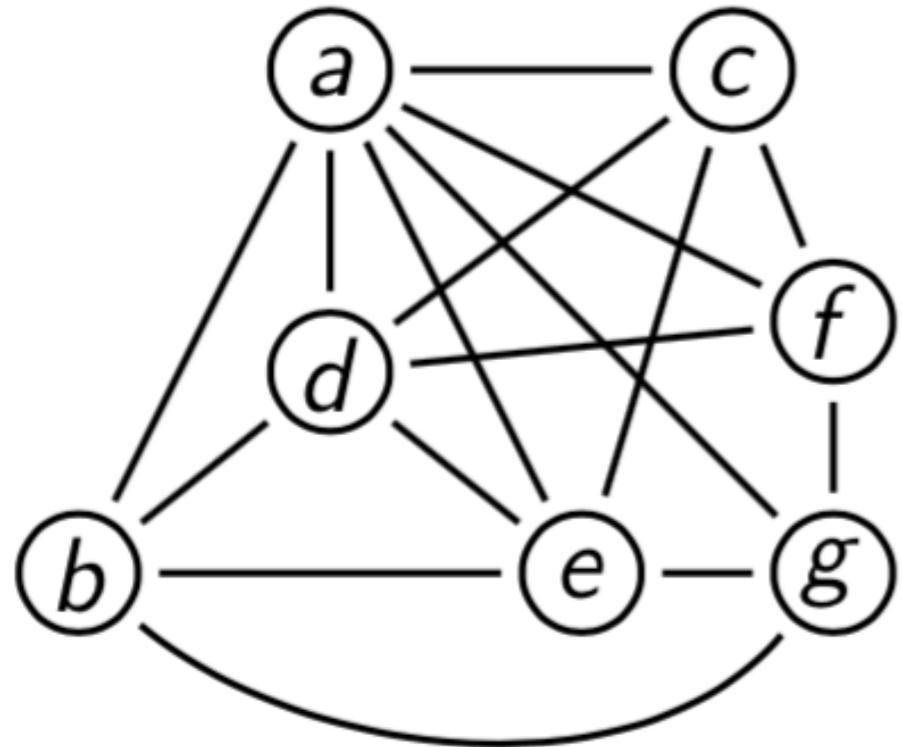
Adjacency List

Graph Representations

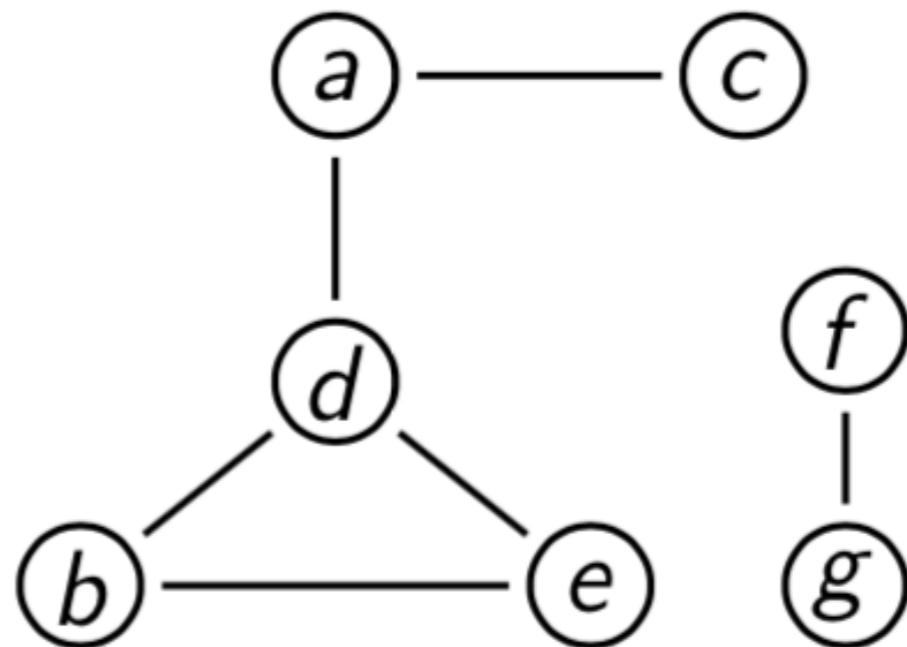


THE UNIVERSITY OF
MELBOURNE

Different kinds of representations are better suited to different kinds of graphs.



For a **dense graph** adjacency matrix might be better



For a **sparse graph** adjacency list might be better

Next time



- Graph traversal, where we get down to the details of graph algorithms