THE UNIVERSITY OF MELBOURNE

COMP90038
Algorithms and Complexity

Lecture 7: Graphs and Graph Concepts
(with thanks to Harald Søndergaard)

Toby Murray

toby.murray@unimelb.edu.au

DMD 8.17 (Level 8, Doug McDonell Bldg)

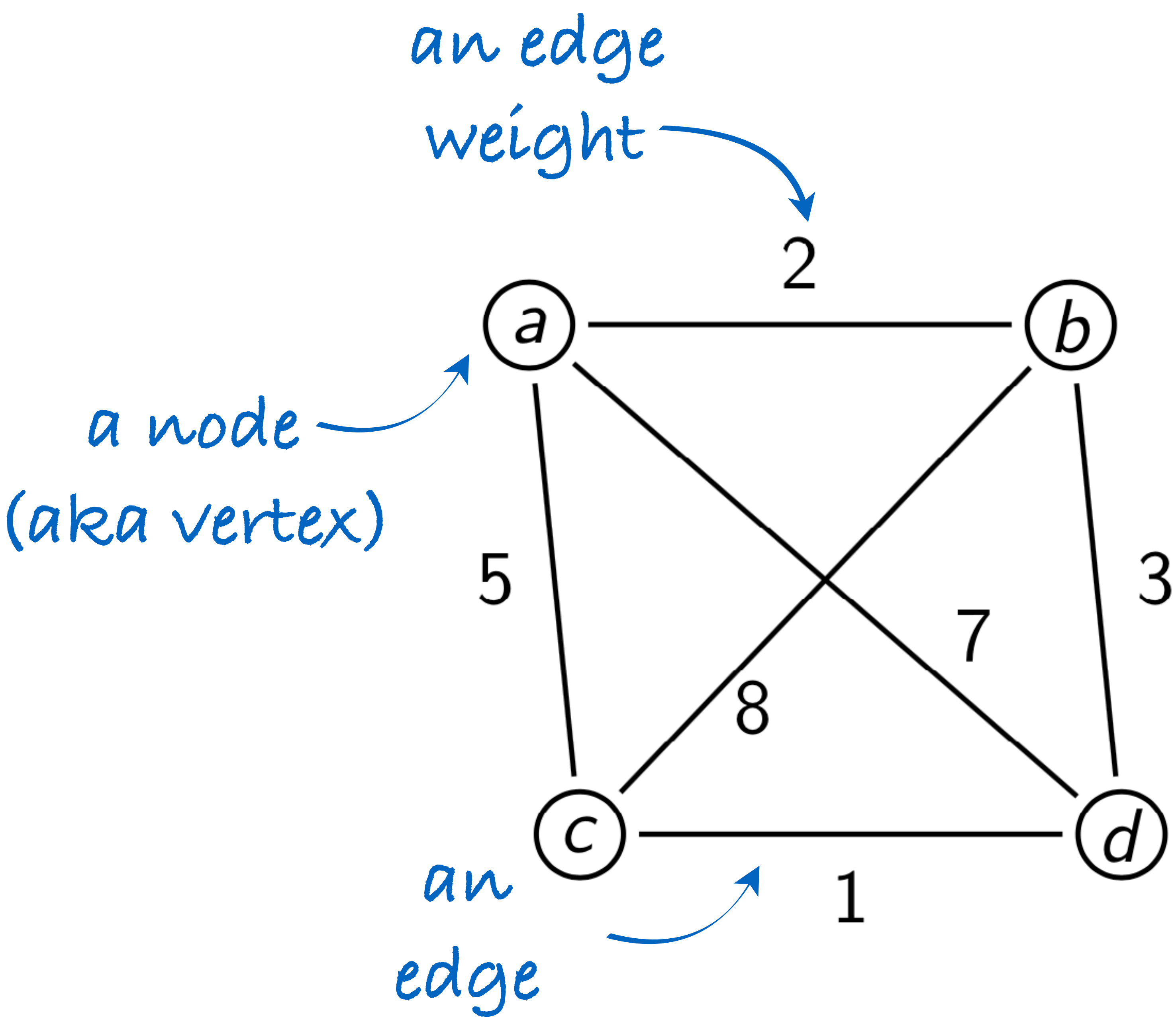
http://people.eng.unimelb.edu.au/tobym

@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

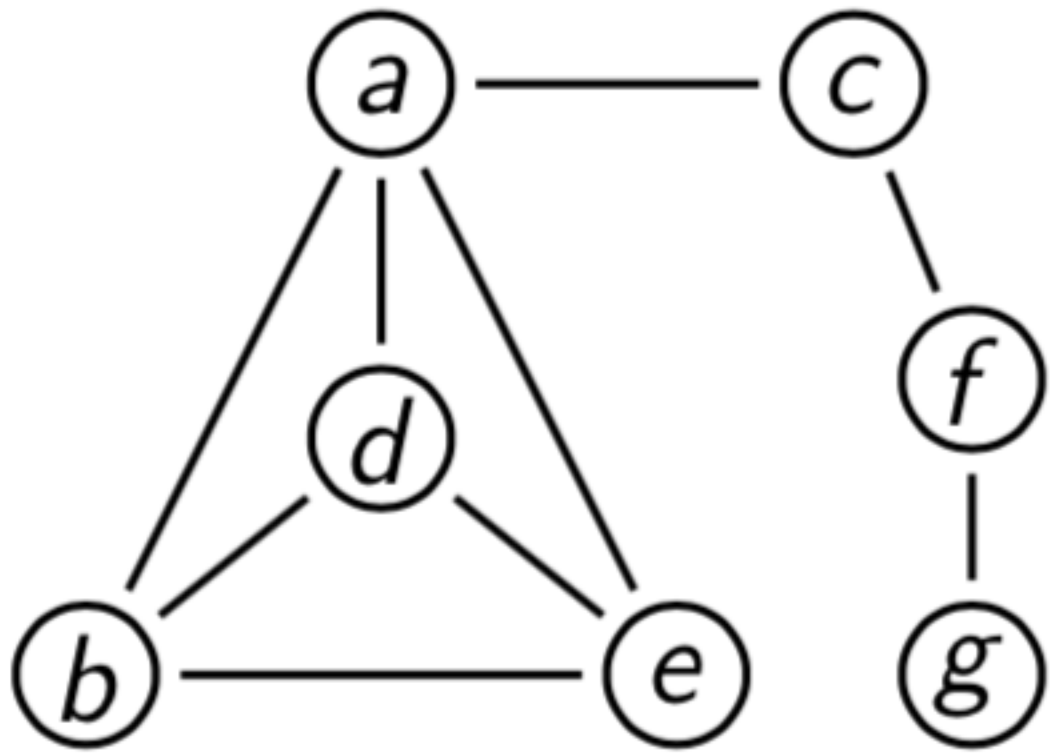
Basic Graph Concepts



This graph is <u>undirected</u> since edges do not have directions associated with them.

Graph Concepts

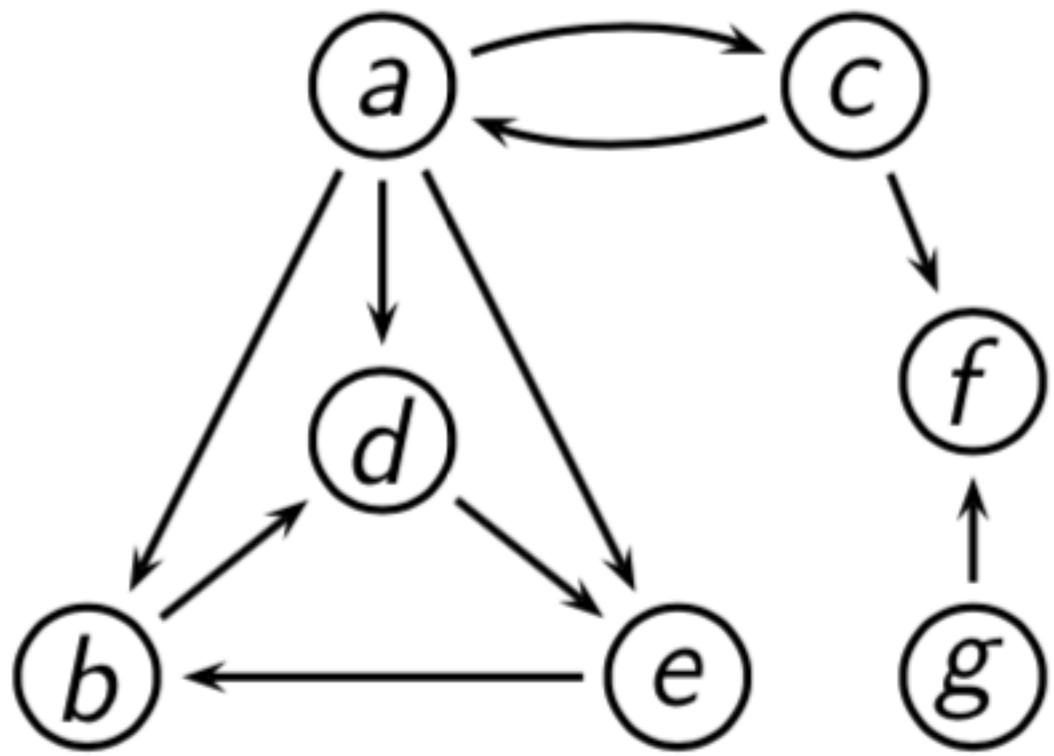
Undirected Graph



(Edges do not have directions associated with them.)

Graph Concepts

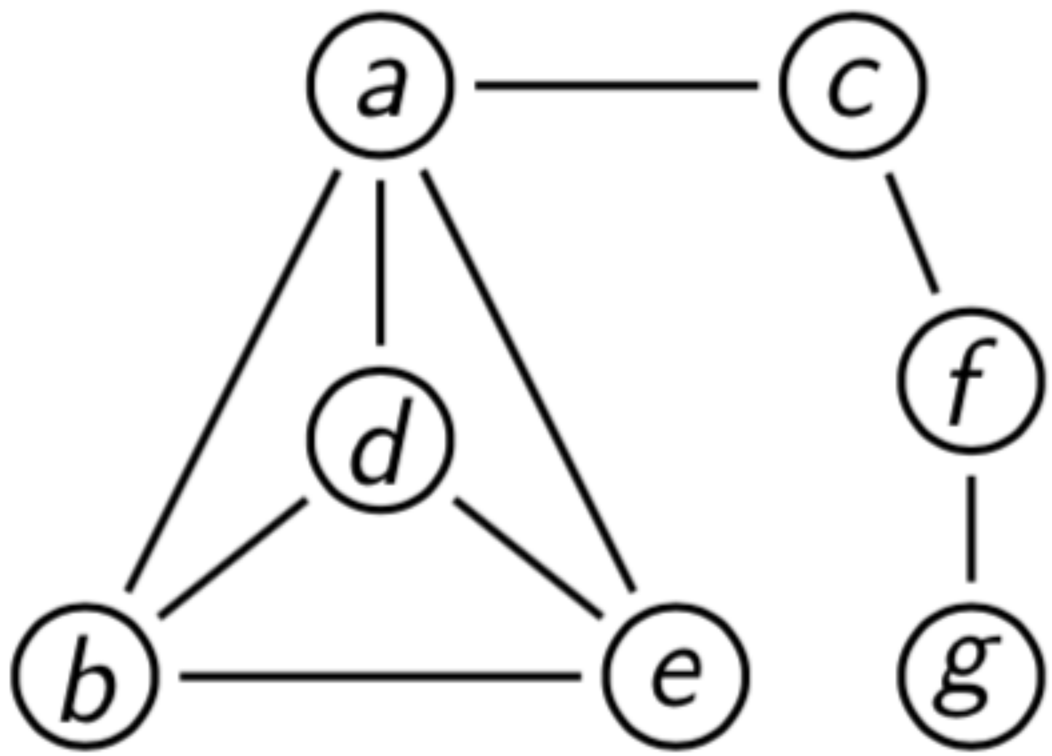
Directed Graph



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

Graph Concepts

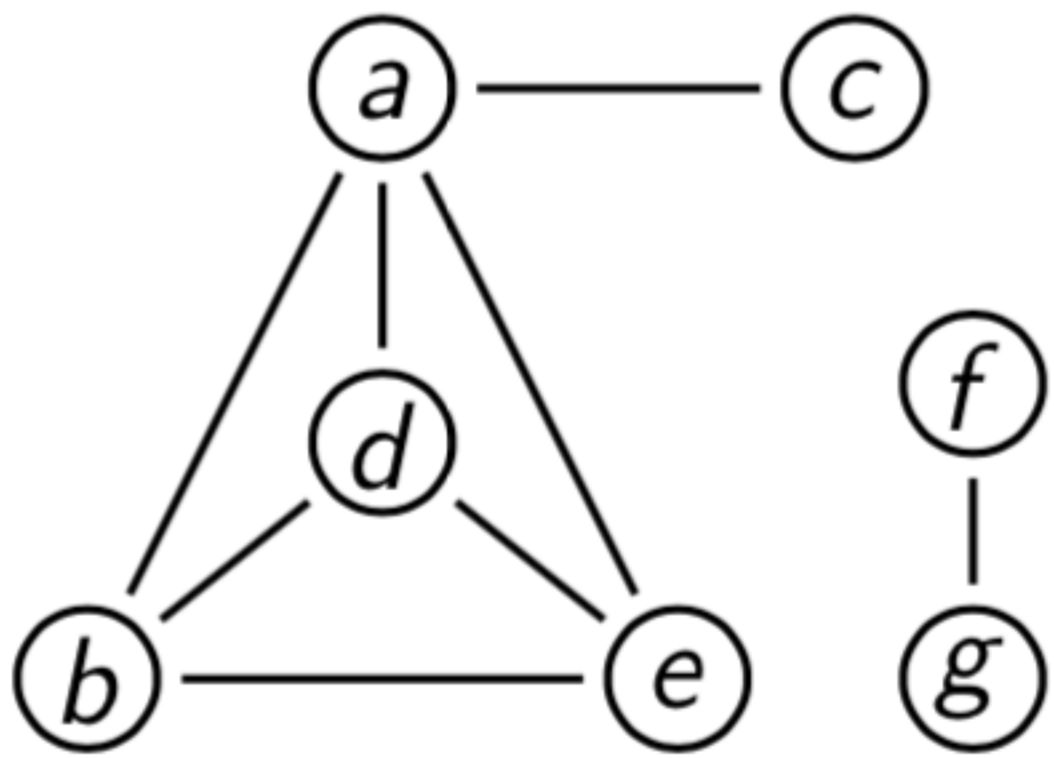
Connected Graph



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

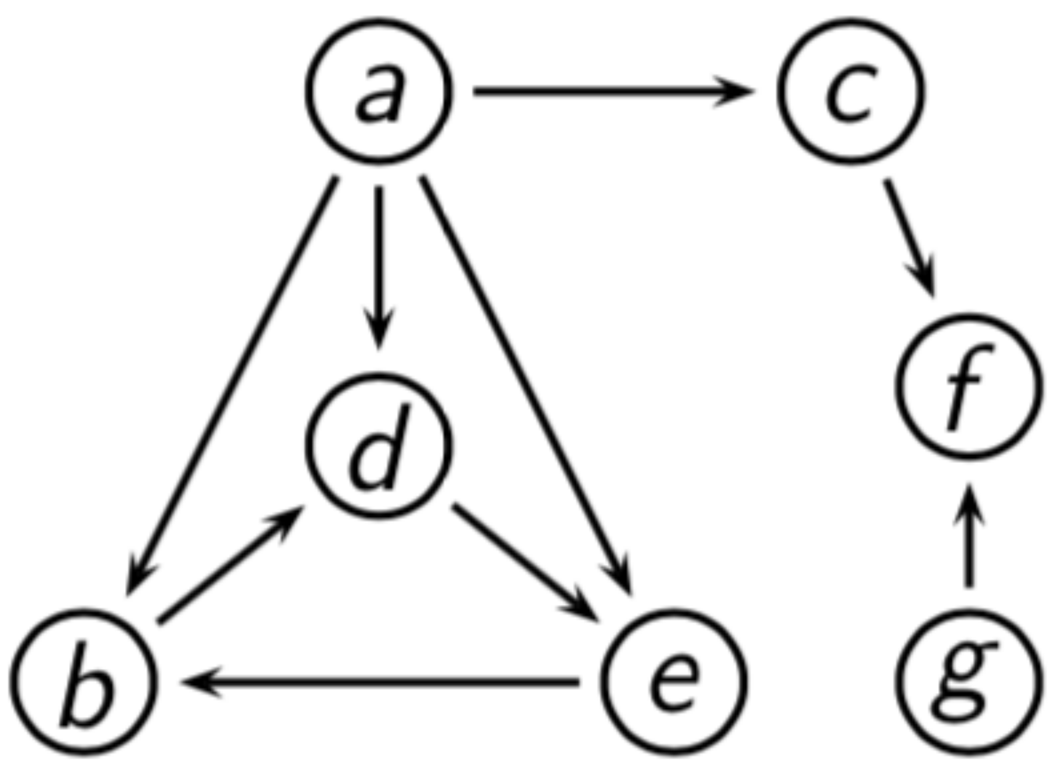
Graph Concepts

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

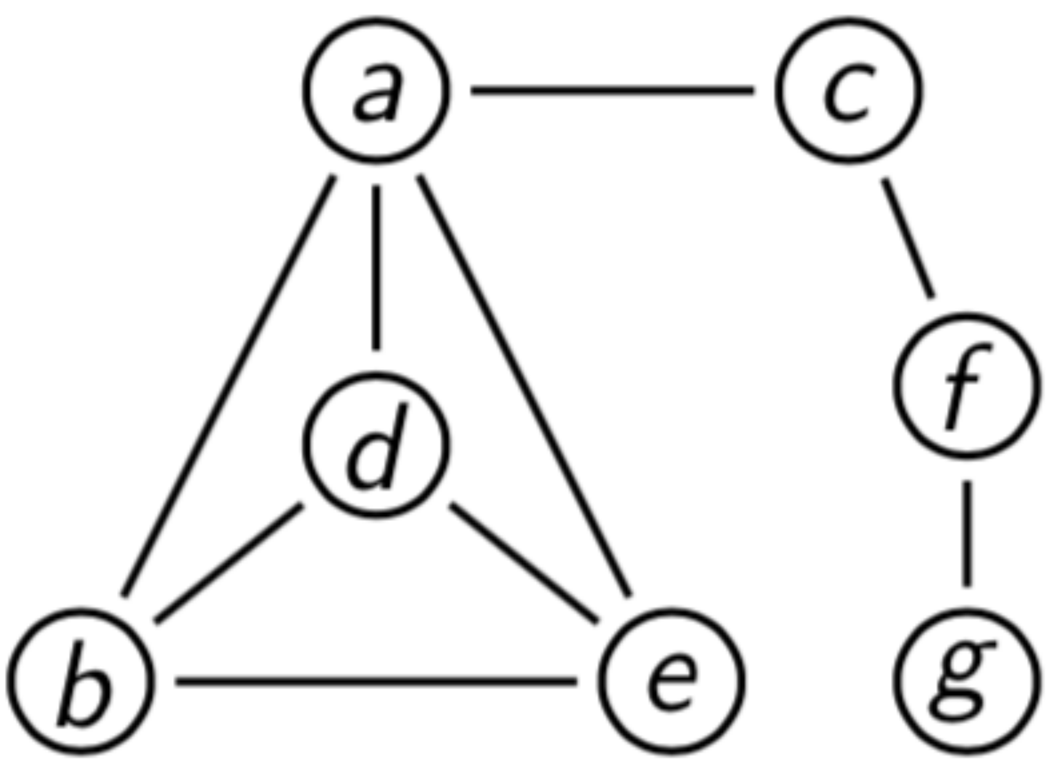


V = {a,b,c,d,e,f,g}

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

Graphs, Mathematically

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

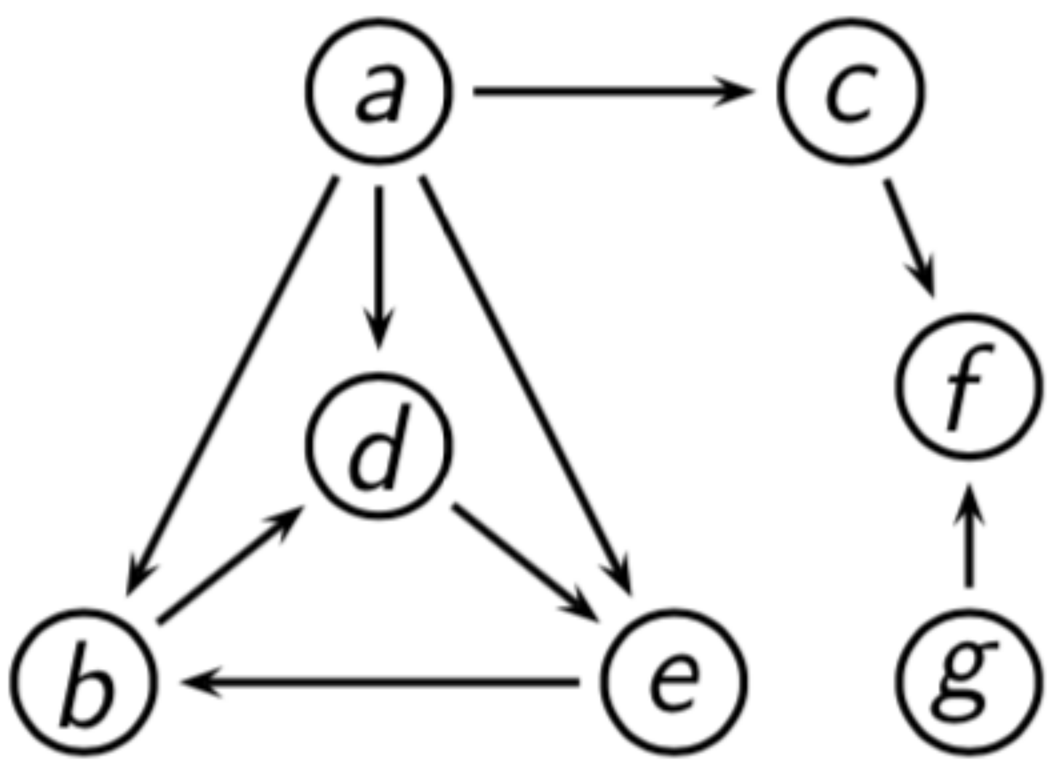


V = {a,b,c,d,e,f,g}

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

Degrees of Nodes

- If (v, u) ∈ 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**



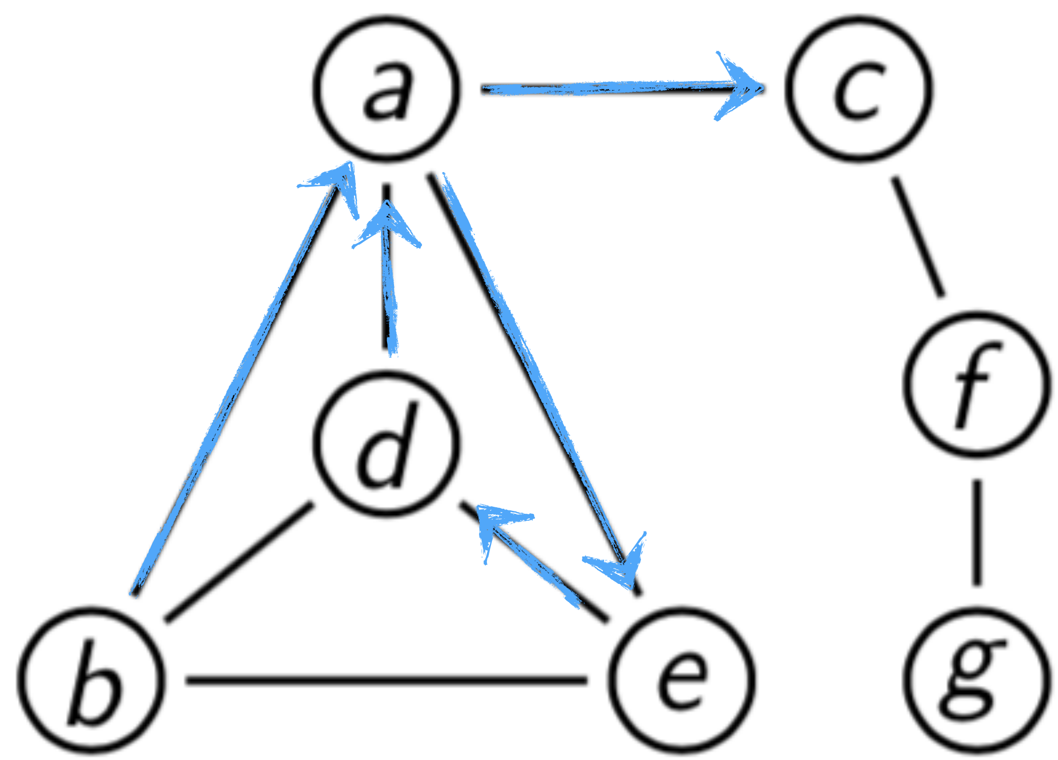
For connected graphs:

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

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

Paths

A path:
b, a, e, d, a, c



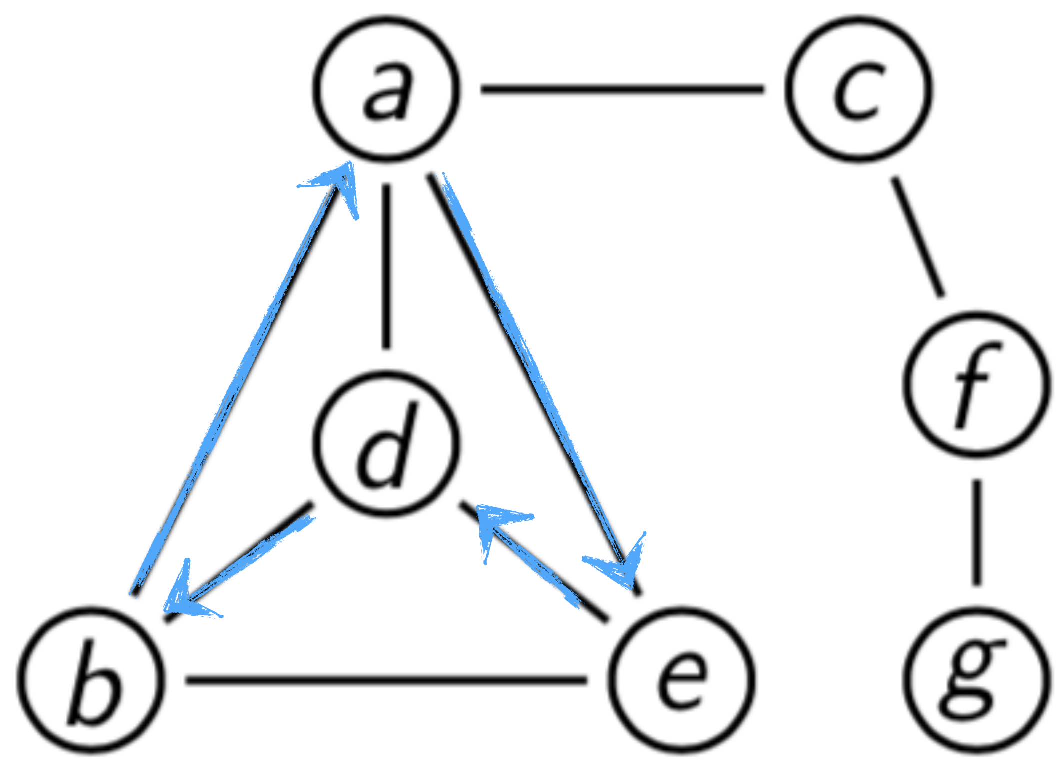
A **path** in ⟨V, E⟩ is a sequence of nodes $v_0, v_1, \ldots, v_k$ from V, so that each $(v_i, v_{i+1}) \in E$.

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

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

Cycles

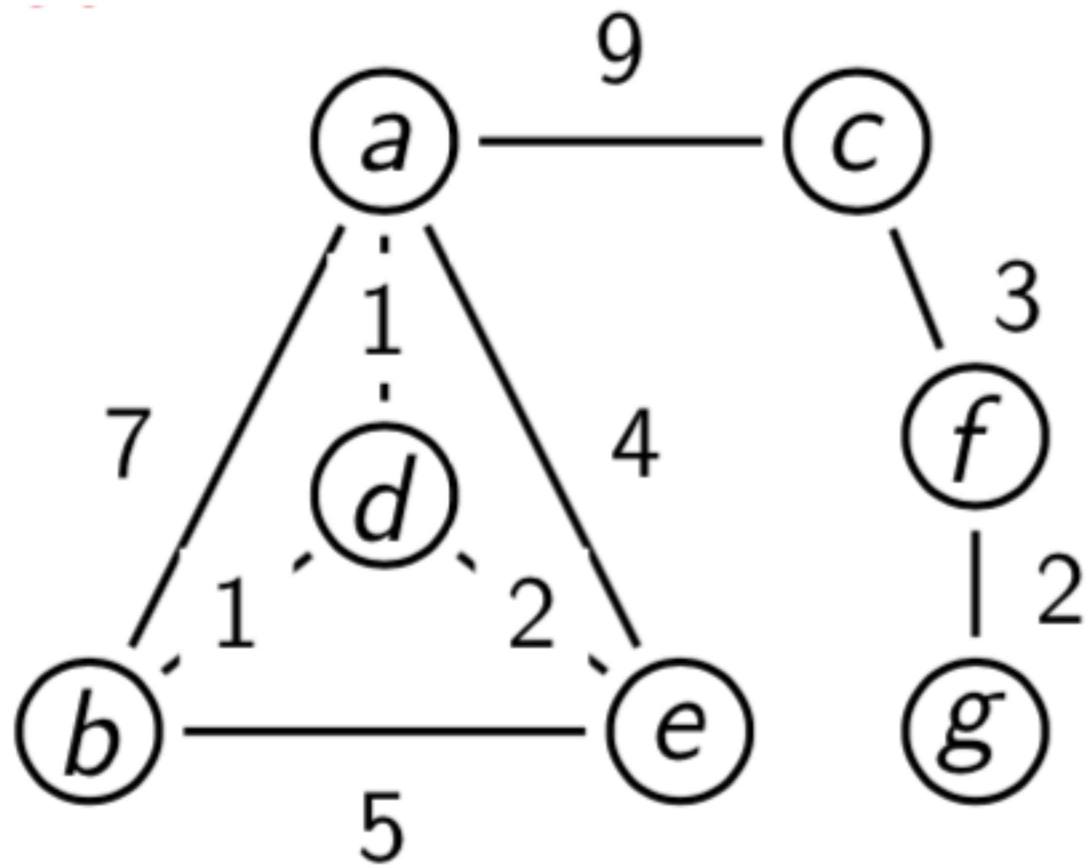
A cycle:
b, a, e, d, b



A **cycle** is a path that starts and finishes at the **same** node and does not traverse the same edge more than once.

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

Weighted Graphs

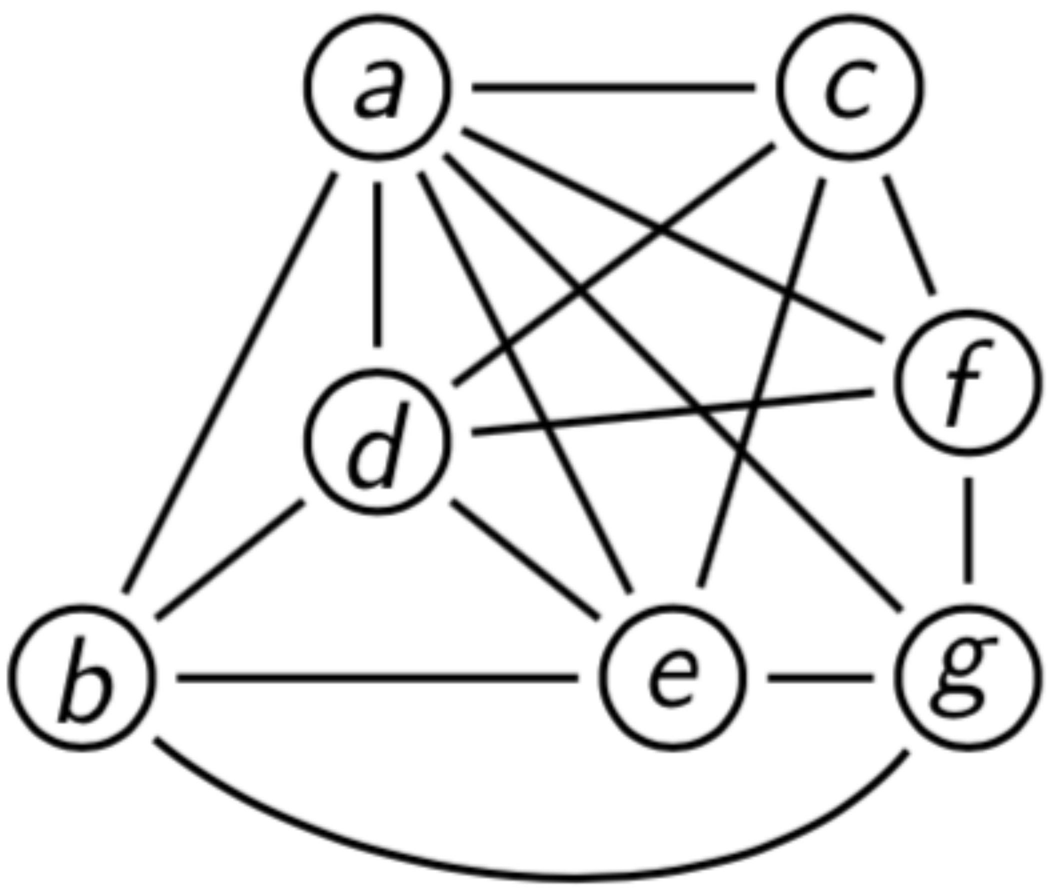


Each edge (v,u) has a **weight** indicating some information about that connection from v to u

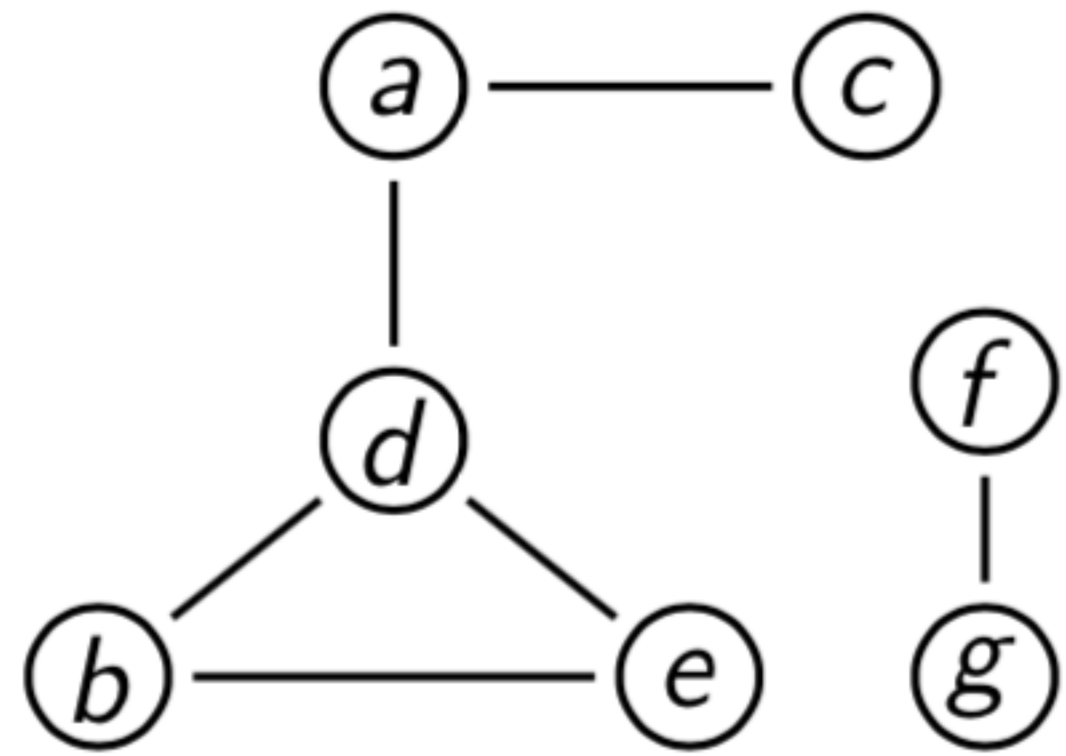
The information depends on what the graph represents: network congestion, physical distance, cost, etc.

Dense vs Sparse Graphs

THE UNIVERSITY OF MELBOURNE



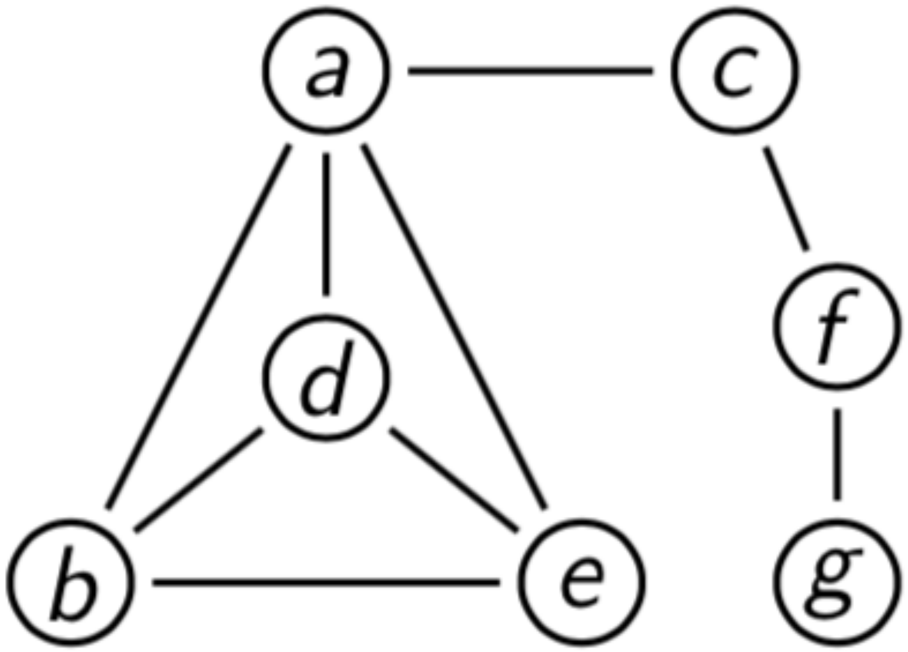
Dense Graph
(lots of edges, relative to number of nodes)



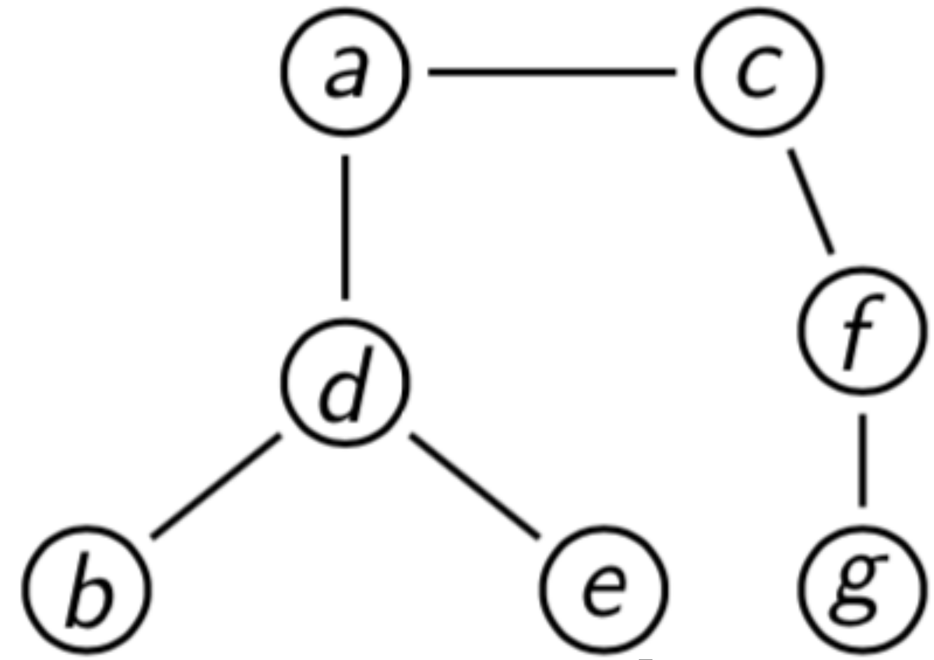
Sparse Graph
(few edges, relative to number of nodes)

Cyclic vs Acyclic

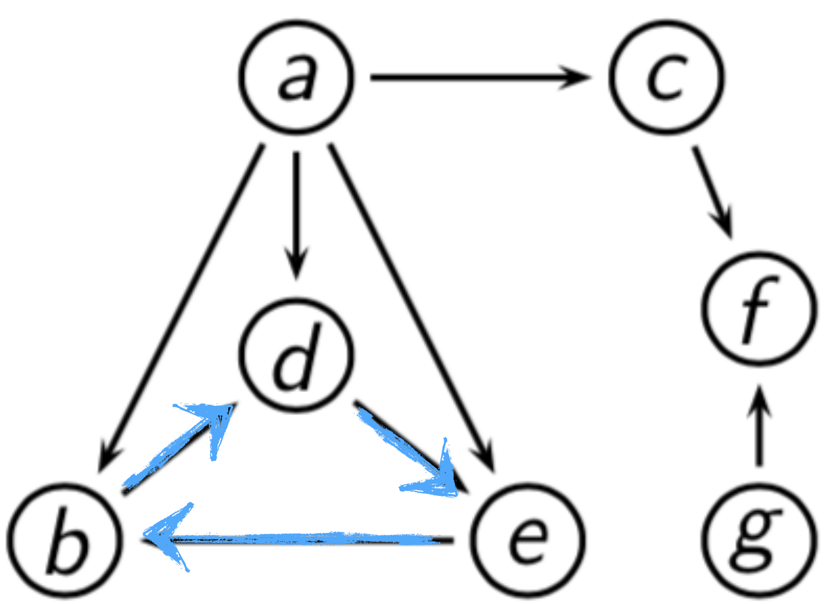
THE UNIVERSITY OF MELBOURNE



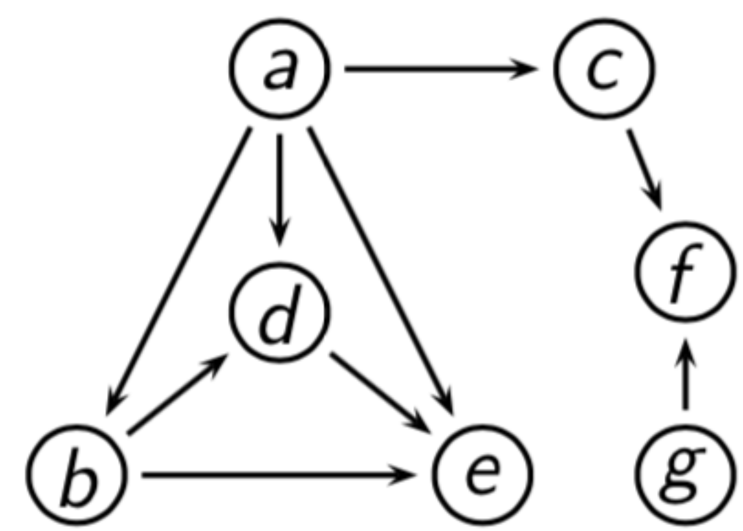
Cyclic



Acyclic



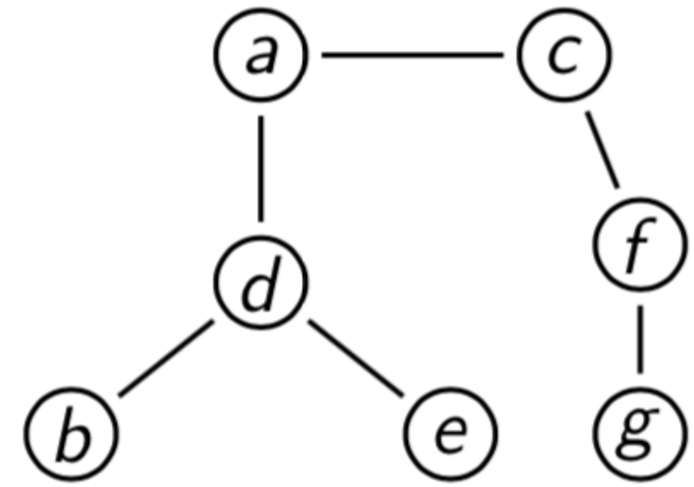
Directed Cyclic



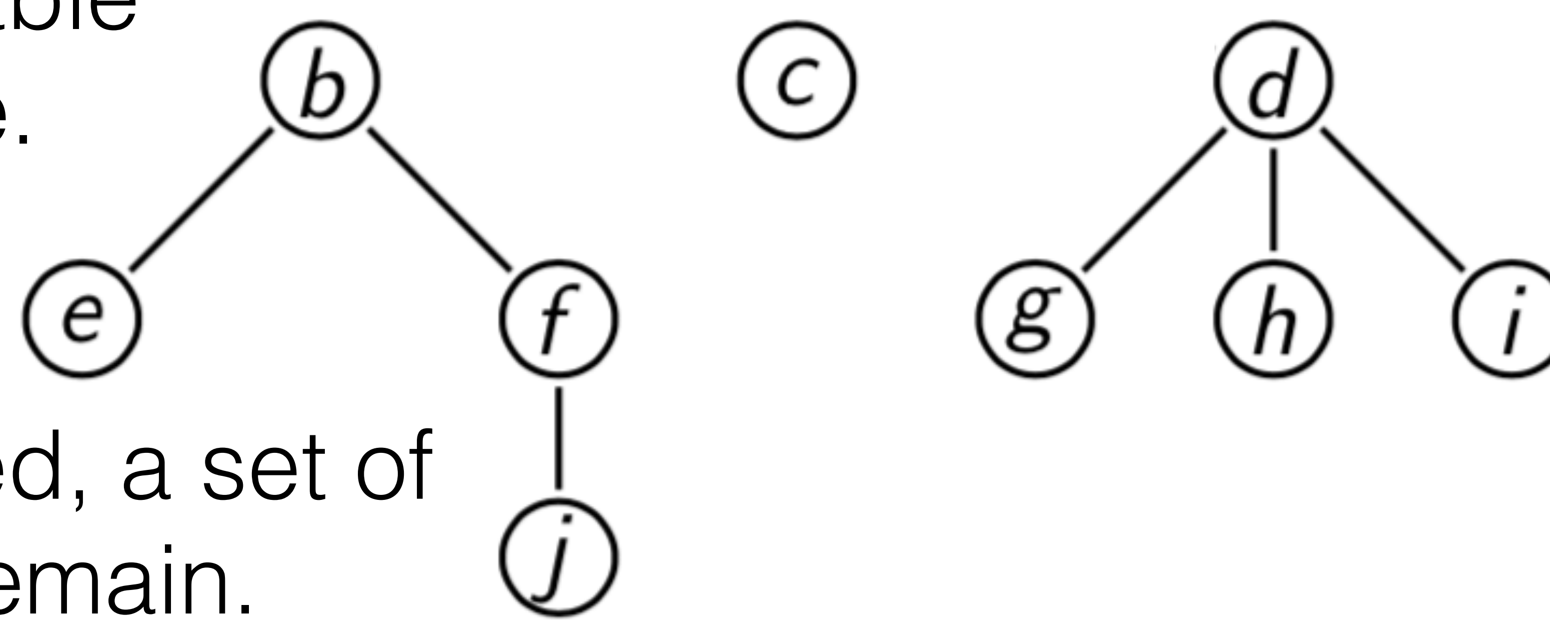
Directed Acyclic Graph (a dag)

Rooted Trees

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.

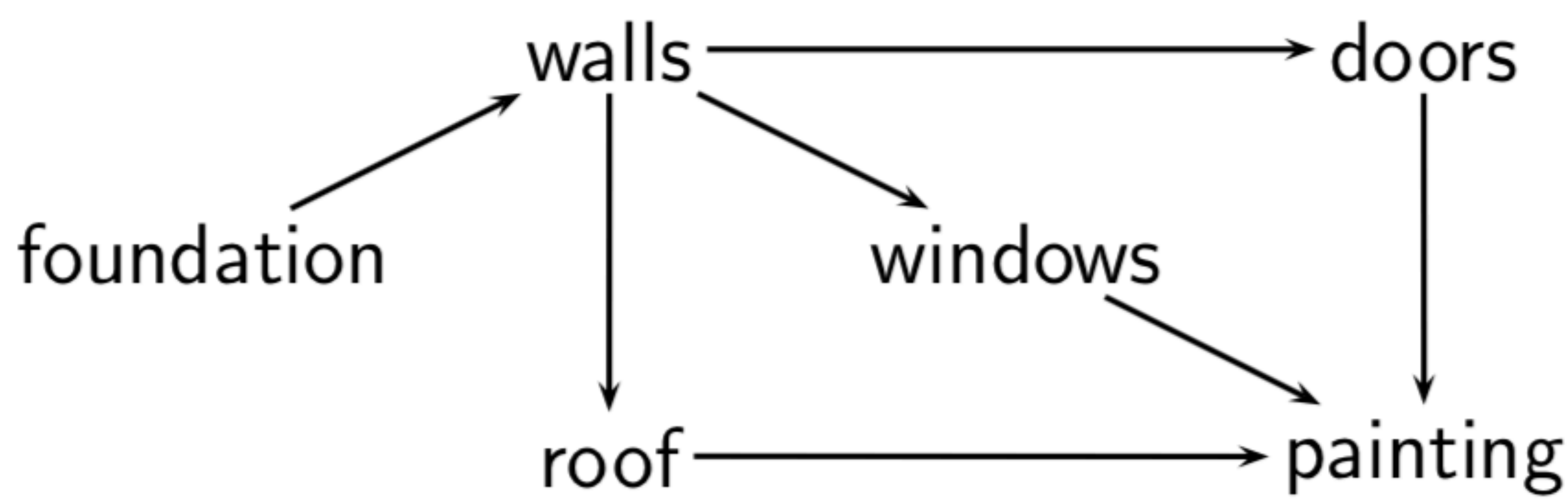


When the root is removed, a set of rooted (sub-)trees remain.

Modelling with Graphs

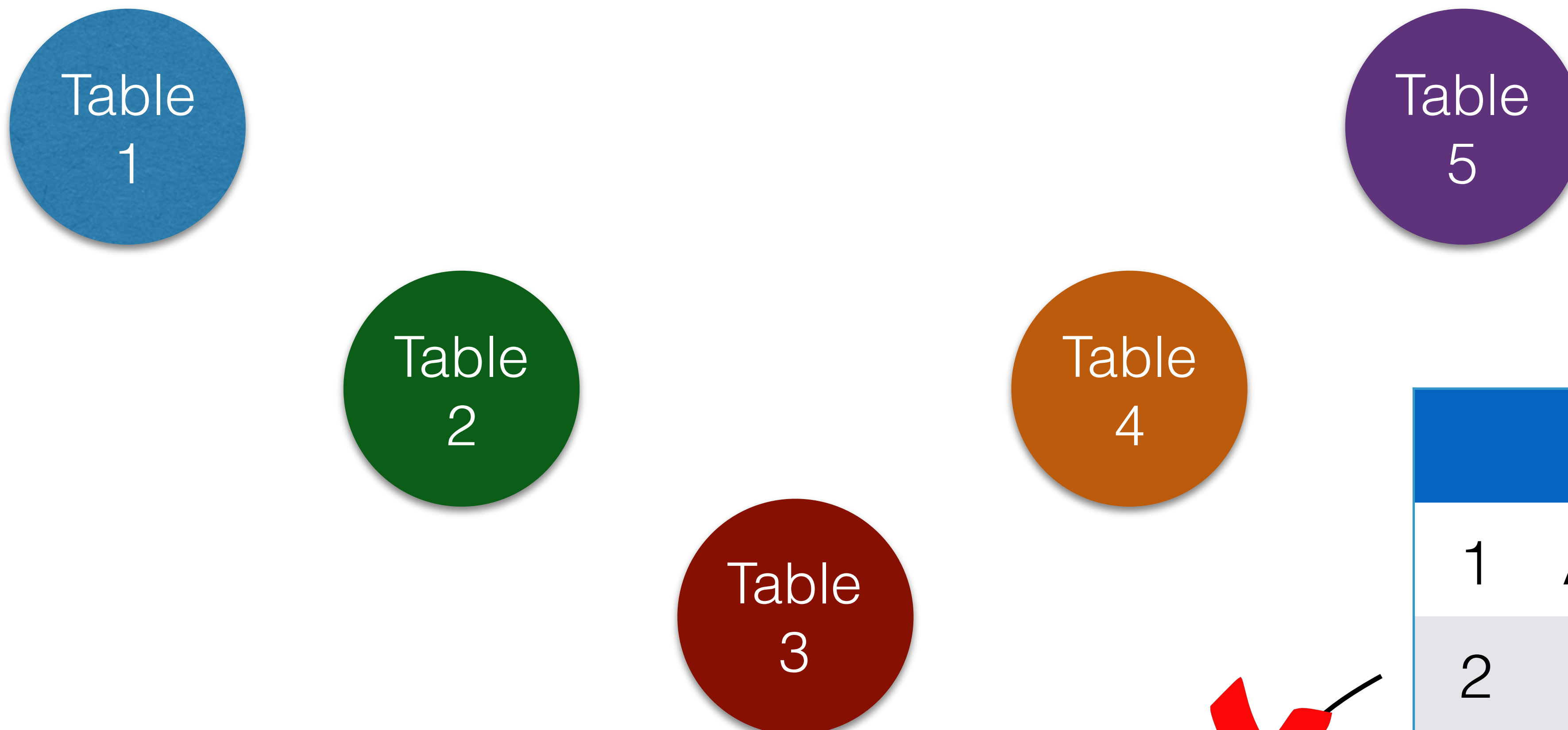
Graph algorithms are of great importance because so many different problem types can be abstracted to graph problems.

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



Modelling with Graphs

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

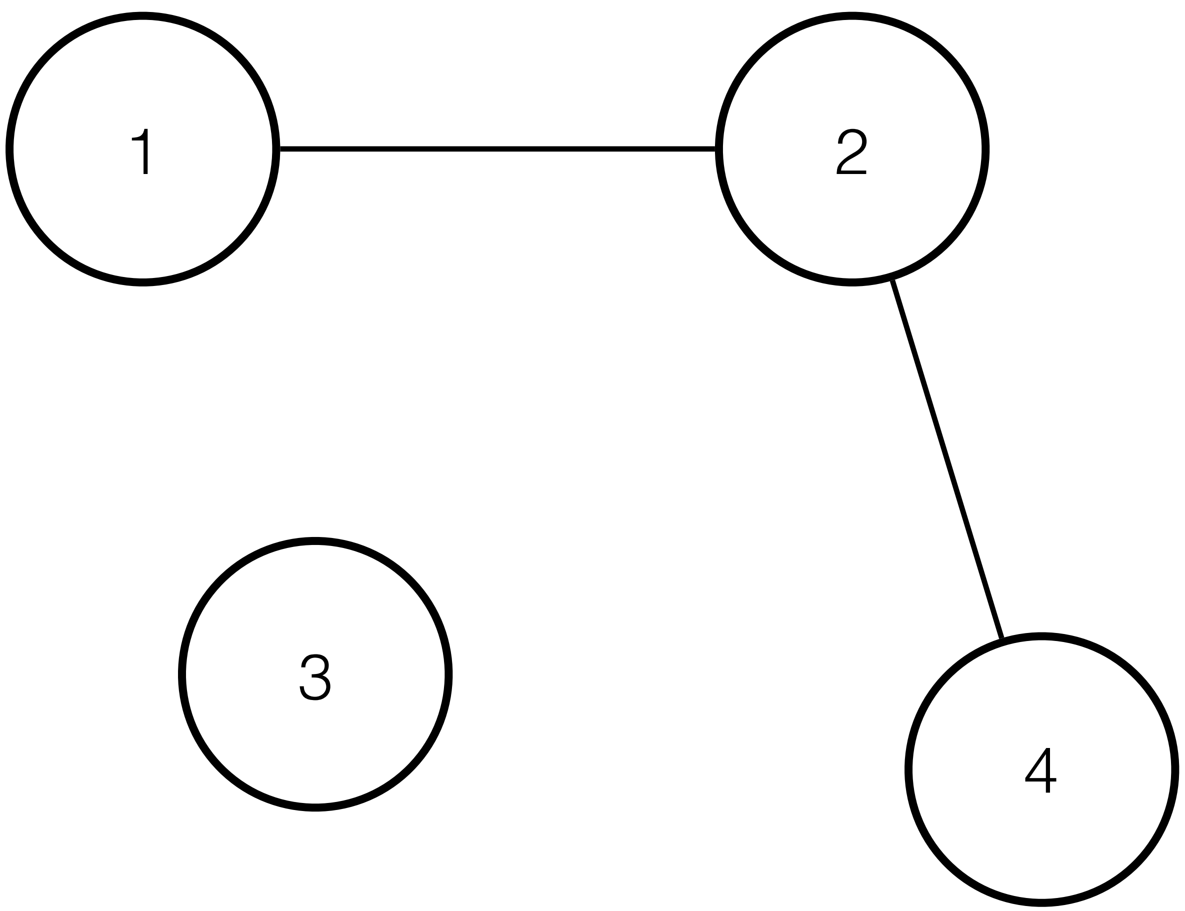


	Guest List
1	Aunt Marth
2	Aunt Sally
3	Uncle Joe
4	Cousin Andy
…	….

Modelling with Graphs

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.

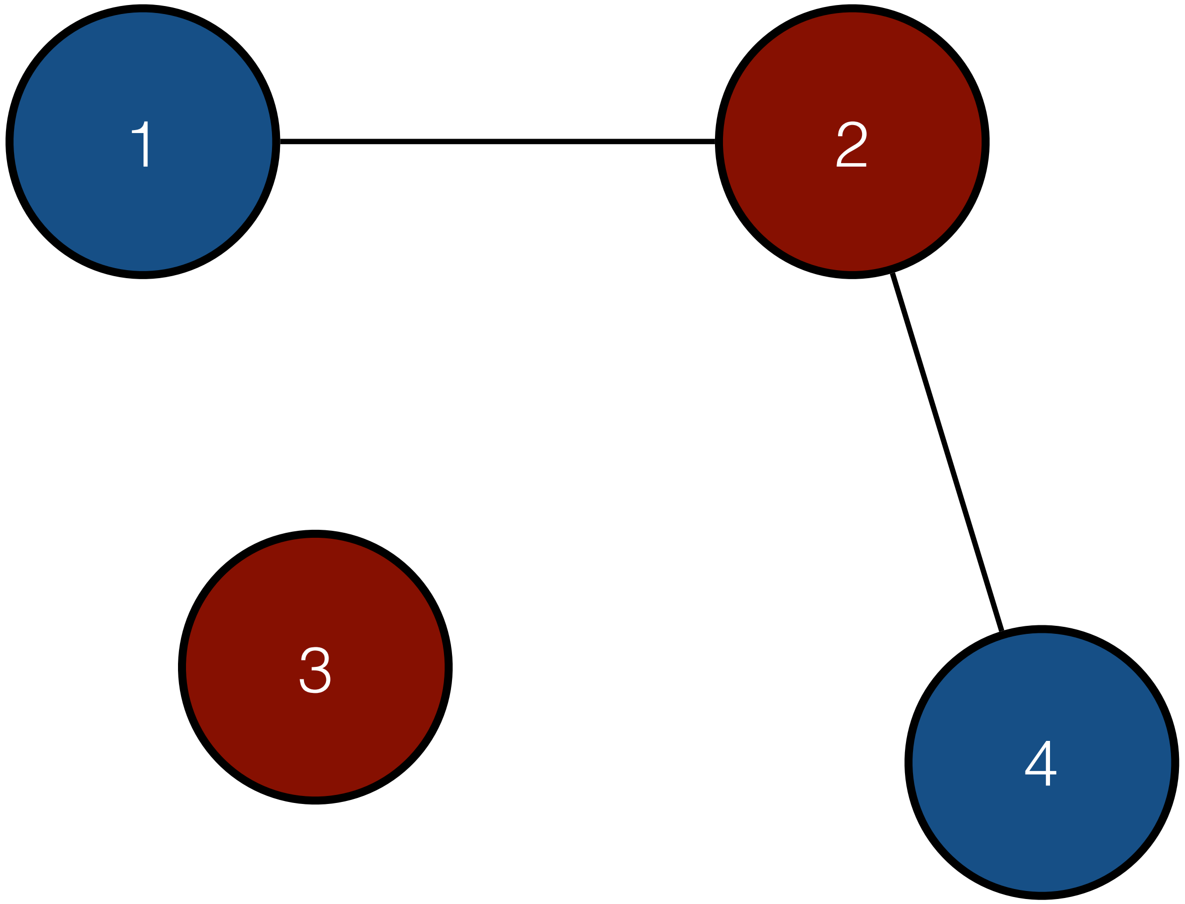


	Guest List
1	Aunt Martha
2	Aunt Sally
3	Uncle Joe
4	Cousin Andy
…	….

Modelling with Graphs

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

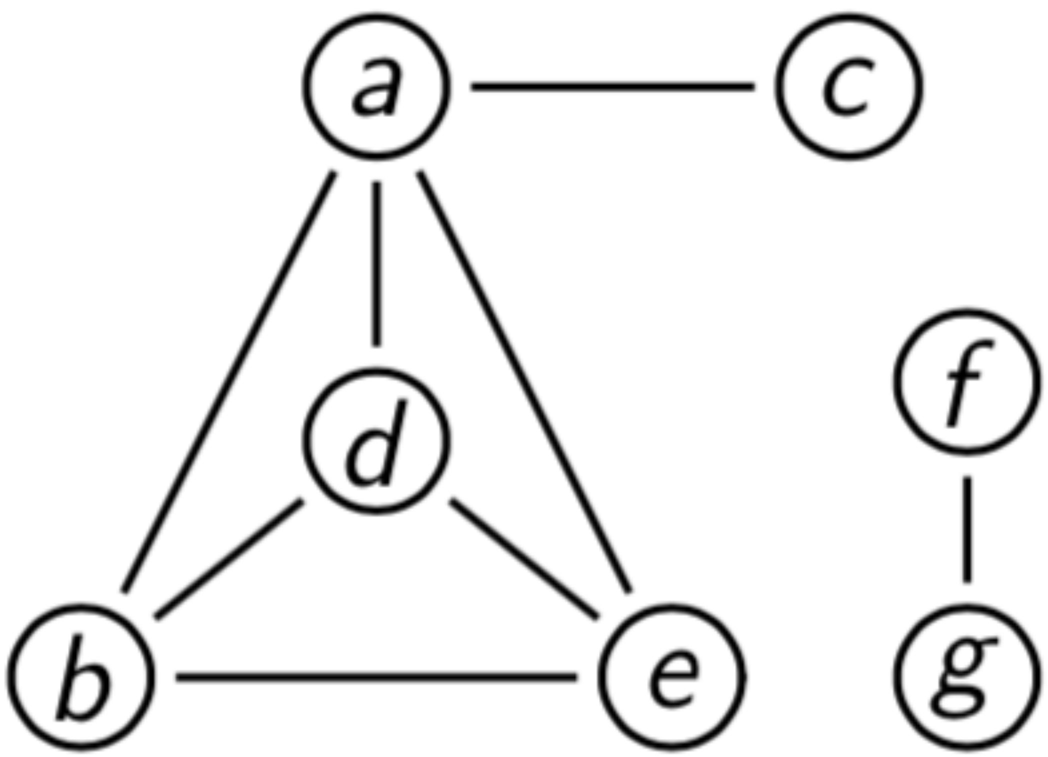
Seating planning with k tables can be **reduced** to the graph k-coloring problem:

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

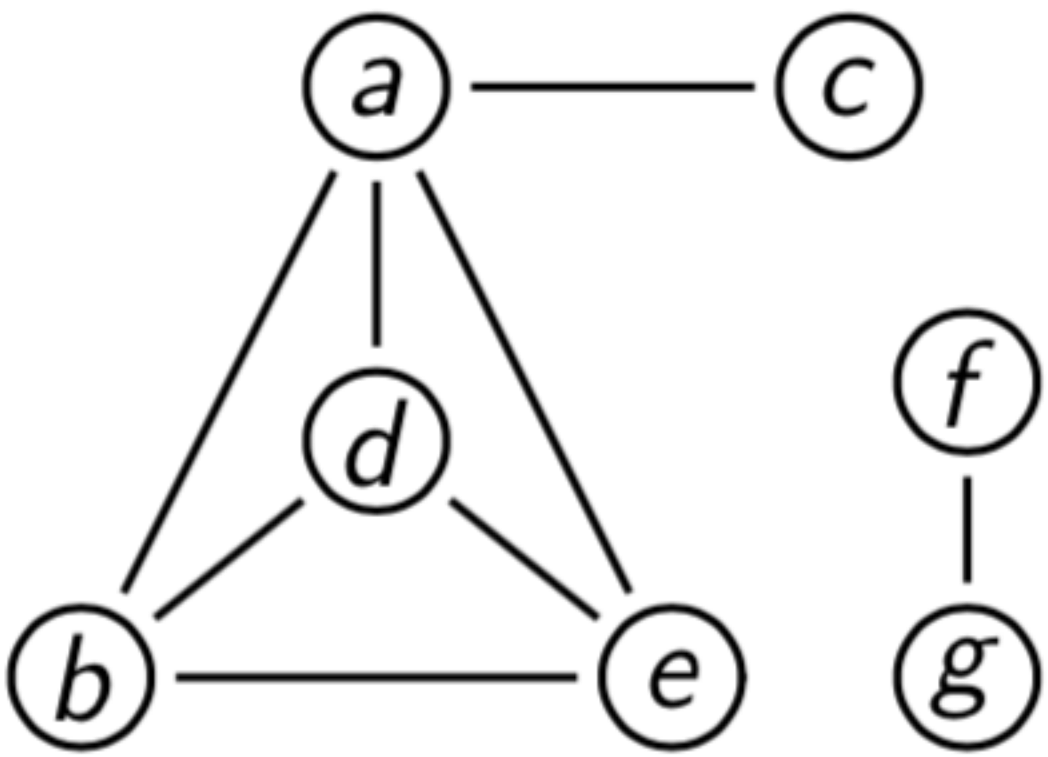


	a	b	c	d	e	f	g
a	0	1	1	1	1	0	0
b	1	0	0	1	1	0	0
c	1	0	0	0	0	0	0
d	1	1	0	0	1	0	0
e	1	1	0	1	0	0	0
f	0	0	0	0	0	0	1
g	0	0	0	0	0	1	0

Adjacency Matrix

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

Graph Representations: Undirected Graphs



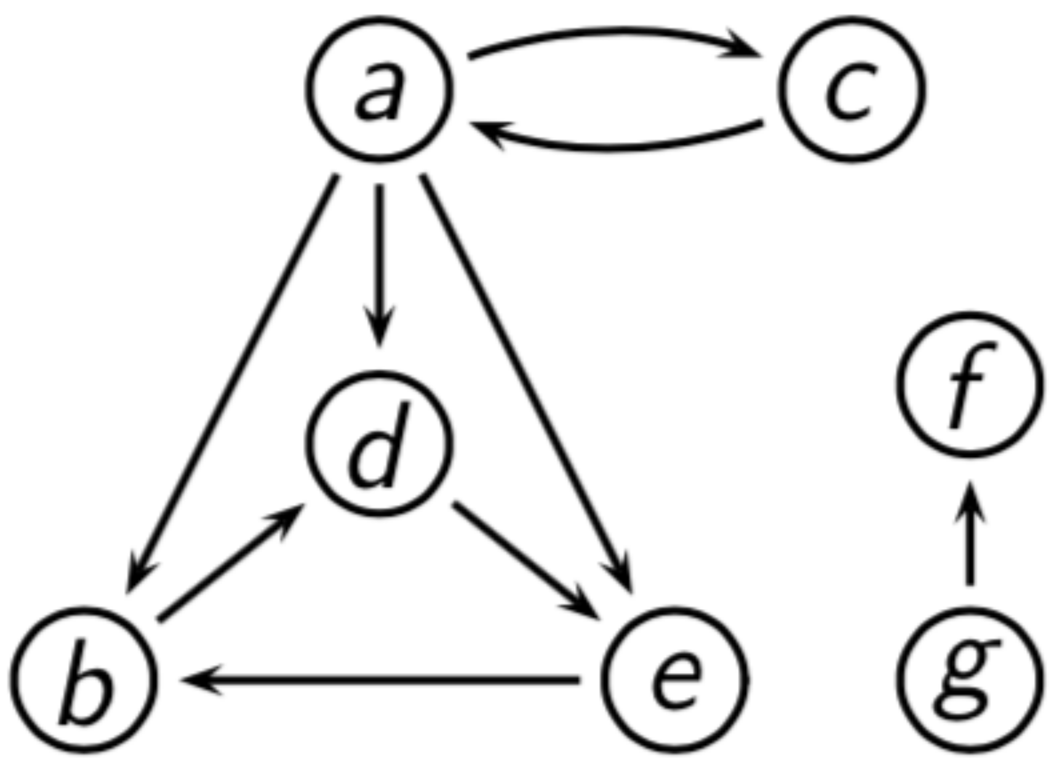
a	$\to b \to c \to d \to e$
b	$\to a \to d \to e$
c	$\to a$
d	$\to a \to b \to e$
e	$\to a \to b \to d$
f	$\to g$
g	$\to f$

Adjacency List

An **array of linked lists**

(Assuming lists are kept in sorted order.)

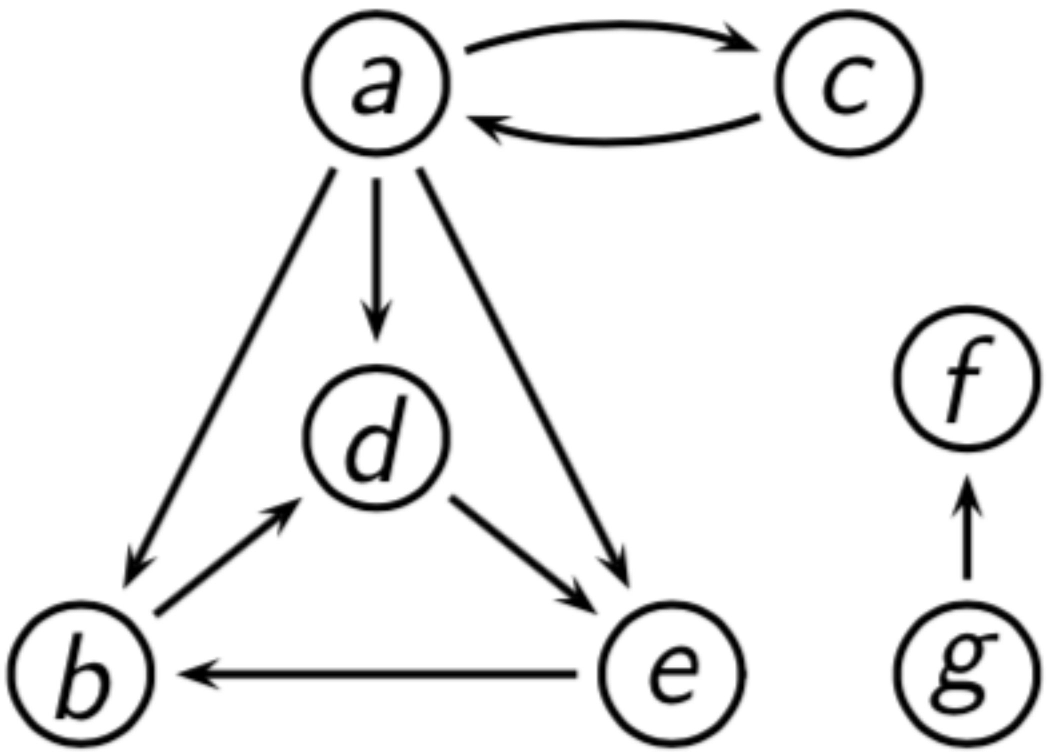
Graph Representations: Directed Graphs



	a	b	c	d	e	f	g
a	0	1	1	1	1	0	0
b	0	0	0	1	0	0	0
c	1	0	0	0	0	0	0
d	0	0	0	0	1	0	0
e	0	1	0	0	0	0	0
f	0	0	0	0	0	0	0
g	0	0	0	0	0	1	0

Adjacency Matrix

Graph Representations: Directed Graphs



$a \to b \to c \to d \to e$

$b \to d$

$c \to a$

$d \to e$

$e \to b$

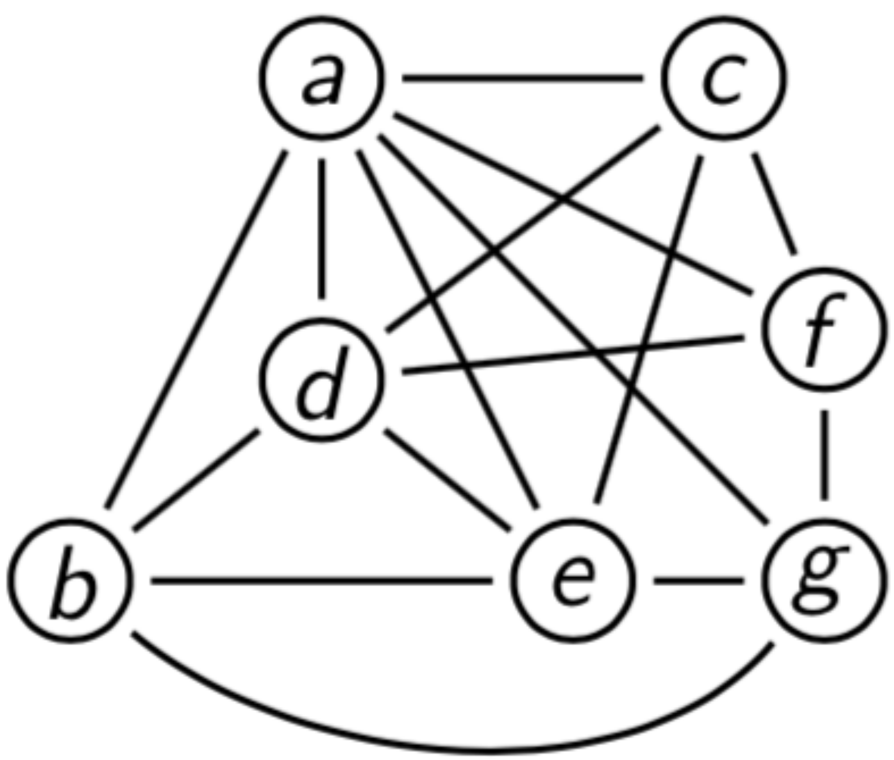
f

$g \to f$

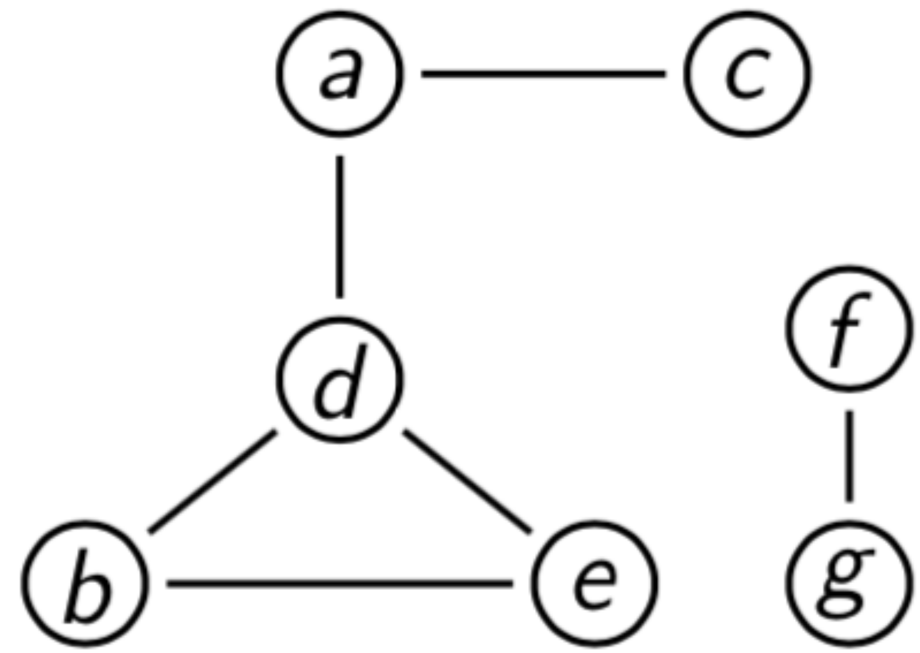
Adjacency List

Graph Representations

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