

# COMP20007 Design of Algorithms

## Graphs and Graph Concepts

---

Lars Kulik

Lecture 6

Semester 1, 2020

# Graphs Again

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.

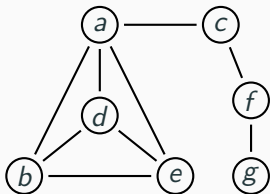
Graph algorithms are useful for a large number of practical problems: network design, flow design, planning, scheduling, route finding, and other logistics applications.

# Graphs, Mathematically

$$G = \langle V, E \rangle$$

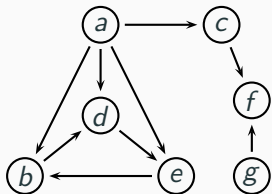
$V$ : Set of **nodes** or **vertices**

$E$ : Set of **edges** (a binary relation on  $V$ )



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

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

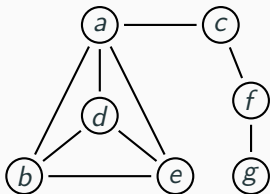


$$V = \{a, b, c, d, e, f, g\}$$

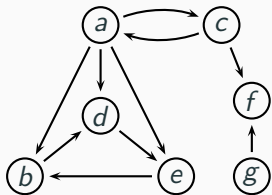
$$E = \{(a, b), (a, c), (a, d), \\ (a, e), (b, d), (c, f), \\ (d, e), (e, b), (g, f)\}$$

# Graph Concepts

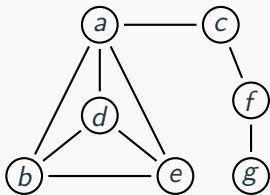
Undirected:



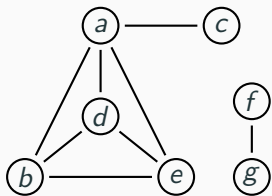
Directed:



Connected:



Not connected, two components:



## More Graph Concepts: Degrees of Nodes

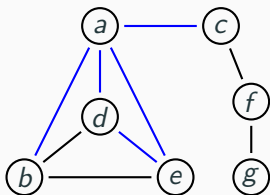
If  $(u, v) \in E$  then  $u$  and  $v$  are **adjacent**, or **neighbours**.

$(u, v)$  **connects**  $u$  and  $v$ , and are  $u$  and  $v$  **incident** to  $(u, v)$ .

The **degree** of node  $v$  is the number of edges incident to  $v$ .

For directed graphs, we talk about  $v$ 's **in-degree** (number of edges going **to**  $v$ ) and its **out-degree** (number of edges going **from**  $v$ ).

## More Graph Concepts: Paths and Cycles



Path  $b, a, d, e, a, c$  shown in blue

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

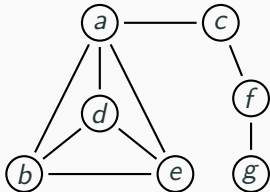
The path  $v_0, v_1, \dots, v_k$  has **length**  $k$ .

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

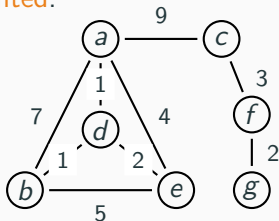
A **cycle** is a simple path, except that  $v_0 = v_k$ , that is, the last node is the same as the first node.

# More Graph Concepts

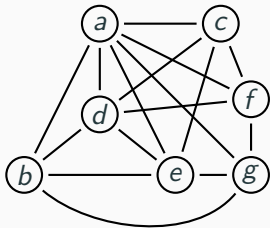
Unweighted:



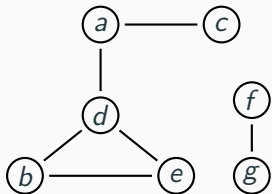
Weighted:



Dense:

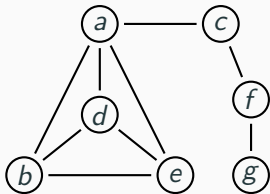


Sparse:

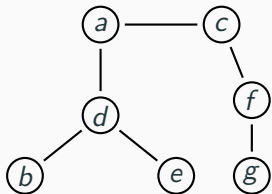


## More Graph Concepts

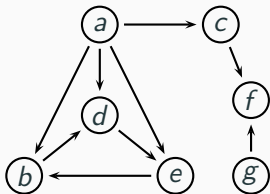
Cyclic:



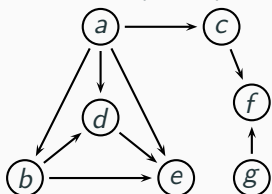
Acyclic (actually, a tree):



Directed cyclic:



Directed acyclic (a dag):

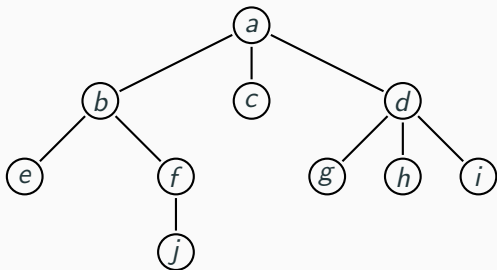




# Rooted Trees

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

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



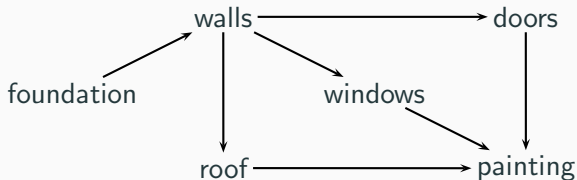
When the root is removed, a set of rooted (sub-)trees remain.

We should draw the rooted tree as a directed graph, but usually we instead rely on the layout: “parents” sit higher than “children”.

# 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 are central in scheduling problems:



# Modelling with Graphs

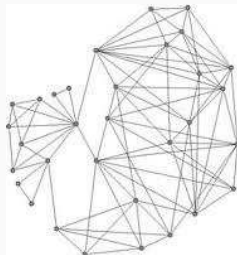
Graphs find use in all sorts of modelling.

Assume you want to invite friends to dinner and you have  $k$  tables available.

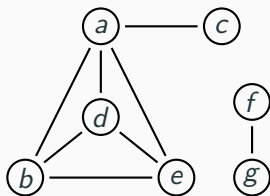
Some guests dislike some of the others; thus we need a seating plan that avoids placing foes at the same table.

The natural model is an undirected graph, with a node for each guest, and an edge between any two guests that don't get along.

This **reduces** your problem to the “graph  $k$ -colouring problem”: find, if possible, a colouring of nodes so that all connected nodes have a different colour.



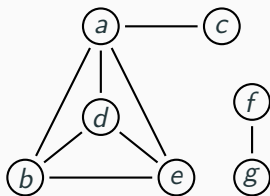
## Graph Representations, Undirected Graphs



	<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

The **adjacency matrix** for the graph.

## Graph Representations, Undirected Graphs

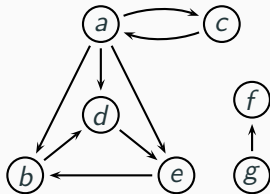


<i>a</i>	$\rightarrow b \rightarrow c \rightarrow d \rightarrow e$
<i>b</i>	$\rightarrow a \rightarrow d \rightarrow e$
<i>c</i>	$\rightarrow a$
<i>d</i>	$\rightarrow a \rightarrow b \rightarrow e$
<i>e</i>	$\rightarrow a \rightarrow b \rightarrow d$
<i>f</i>	$\rightarrow g$
<i>g</i>	$\rightarrow f$

The **adjacency list** representation.

(Assuming lists are kept in sorted order.)

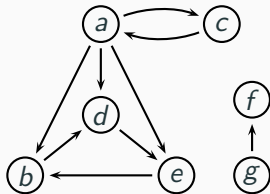
# Graph Representations, Directed Graphs



	<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

The **adjacency matrix** for the graph.

# Graph Representations, Directed Graphs



<i>a</i>	$\rightarrow b \rightarrow c \rightarrow d \rightarrow e$
<i>b</i>	$\rightarrow d$
<i>c</i>	$\rightarrow a$
<i>d</i>	$\rightarrow e$
<i>e</i>	$\rightarrow b$
<i>f</i>	
<i>g</i>	$\rightarrow f$

The **adjacency list** representation.

# Graph Representations

Each representation has advantages and disadvantages.

Think of some!





Graph traversal, in which we get down to the nitty-gritty details of graph algorithms.