

COMP90038 Algorithms and Complexity

Graphs and Graph Concepts

Michael Kirley

Lecture 7

Semester 1, 2019

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.

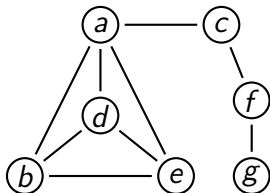
These two methods of graph traversal form the backbone of a surprisingly large number of useful graph algorithms.

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.

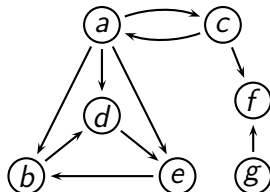
Moreover, important numeric and logic problems can be **reduced** to graph problems—more on this in Week 12.

Graph Concepts

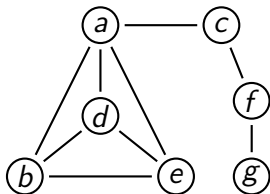
Undirected:



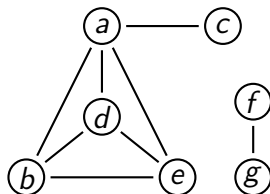
Directed:



Connected:



Not connected, two components:

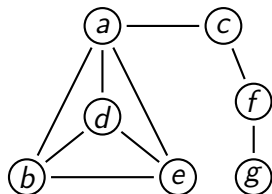


Graphs, Mathematically

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

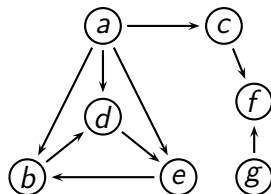
V : Set of **nodes** or **vertices**

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



array

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



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

More Graph Concepts: Degrees of Nodes

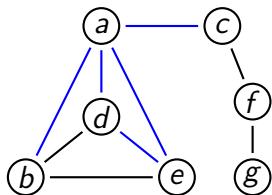
If $(v, u) \in E$ then v and u are ^{比邻}adjacent, or neighbours.

(v, u) is incident on, or connects, v and u .

The degree of node v is the number of edges incident on 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 $(v_i, v_{i+1}) \in E$.

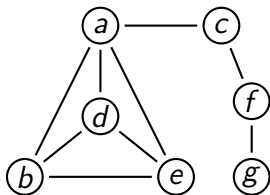
The path v_0, v_1, \dots, v_k has **length** k .

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

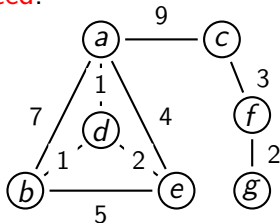
A **cycle** is a simple path, except that $v_0 = v_k$, that is, the last node is the same as the first node.

More Graph Concepts

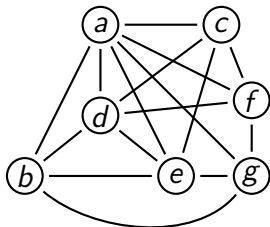
Unweighted:



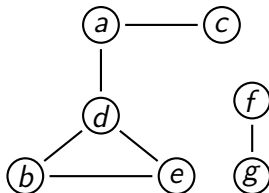
Weighted:



密度大的
Dense:

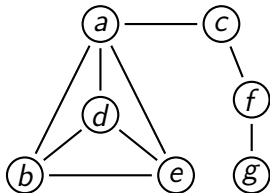


稀少的
Sparse:

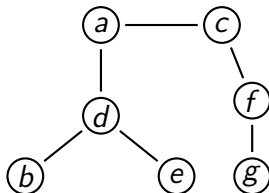


More Graph Concepts

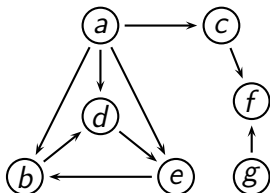
Cyclic:



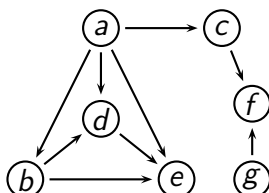
Acyclic (actually, a tree):



Directed cyclic:



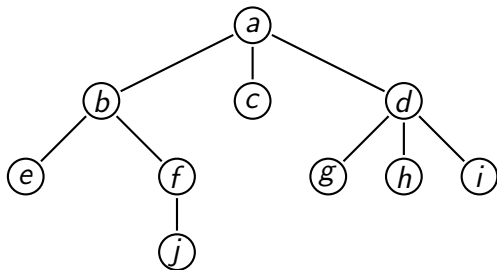
Directed acyclic (a dag):



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 node.



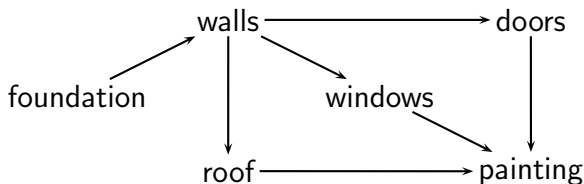
When the root is removed, a set of rooted (sub-)trees remain.

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

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:

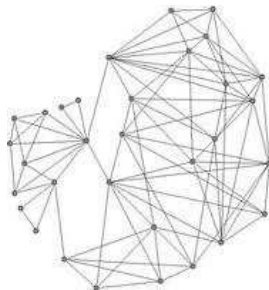


Modelling with Graphs

Graphs find use in all sorts of modelling.

For a simple example, assume you want to invite lots of friends to dinner, and you have k tables available.

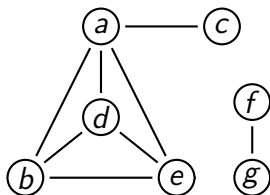
But some of your guests dislike some of the other guests; you want to make a seating plan that avoids placing foes at the same table.



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 no two connected nodes get the same colour.

Graph Representations, Undirected Graphs

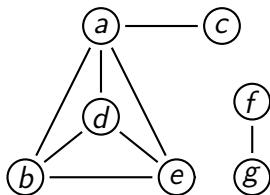


	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>
<i>a</i>	0	1	1	1	1	0	0
<i>b</i>	1	0	0	1	1	0	0
<i>c</i>	1	0	0	0	0	0	0
<i>d</i>	1	1	0	0	1	0	0
<i>e</i>	1	1	0	1	0	0	0
<i>f</i>	0	0	0	0	0	0	1
<i>g</i>	0	0	0	0	0	1	0

The **adjacency matrix** for the graph.

Graph Representations, Undirected Graphs

Sort the node at first when write the program of find the degree of the node

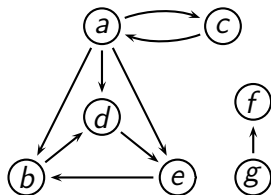


a	$\rightarrow b \rightarrow c \rightarrow d \rightarrow e$
b	$\rightarrow a \rightarrow d \rightarrow e$
c	$\rightarrow a$
d	$\rightarrow a \rightarrow b \rightarrow e$
e	$\rightarrow a \rightarrow b \rightarrow d$
f	$\rightarrow g$
g	$\rightarrow f$

The ^{邻接}adjacency list representation.

(Assuming lists are kept in sorted order.)

Graph Representations, Directed Graphs

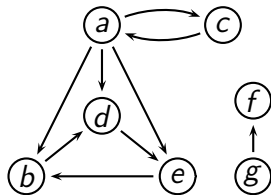


The first row of node
is the start point of
each connection

	<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.

基本事实

correct and correct time complexity 70%