8.1

Graphs

Syllabus

Graph and tree theory;
applied to networks

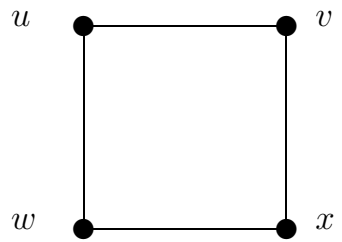
8.2

A *graph* is a kind of diagram.
It consists of *vertices* which are drawn as points or blobs,
and *edges* which are drawn as lines from a vertex to a vertex.

We say an edge that goes from vertex u to vertex v
has *endpoints* u and v.

8.3

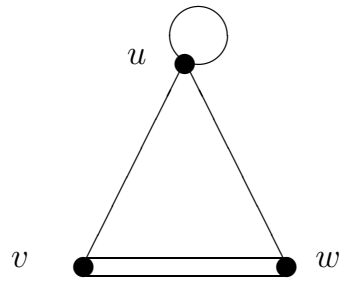
Example:



This graph has four vertices u, v, w, x
and four edges uv, vx, wx and uw.

8.4

Example:



This graph has three vertices u, v, w,
and five edges.
One edge uu is a *loop*,
and two edges together form a *double edge* from v to w.

8.5

The *adjacency matrix* of a graph G is a table:
the vertices are listed along the top and down the left.
The entry in row u and column v is the number of edges
with endpoints u and v.

The adjacency matrix of the graph in 8.4 is

	u	v	w
u	1	1	1
v	1	0	2
w	1	2	0

8.6

A *path* in a graph is a list of vertices and edges,

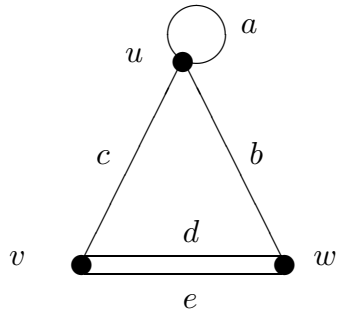
$$v_0, e_0, v_1, e_1, \ldots, e_{n-1}, v_n$$

where each e_i is an edge with endpoints v_i and v_{i+1},
the edges $e_0, \ldots, e_{n-1}$ are all different, and
the vertices $v_0, \ldots, v_n$ are all different except that v_1 and v_n
can be the same.
When $v_1 = v_n$, we say that the path is a *cycle.*

The *length* of a path is the number of edges in it;
so the path above has length n.
The path is *from* v_0 *to* v_n, and these vertices are its *endpoints.*

8.7

Example:



$$v, d, w, b, u, c, v$$

is a cycle of length 3.

$$u, c, v, e, w$$

is a path of length 2 but not a cycle.

8.8

Let G be a graph.

We say that two vertices u, v of G are *linked* to each other
if there is a path with endpoints u and v.

A vertex u is always linked to itself, because u counts
as a path of length 0 with endpoints u and u.

8.9

Writing uRv for

‘u is linked to v’,

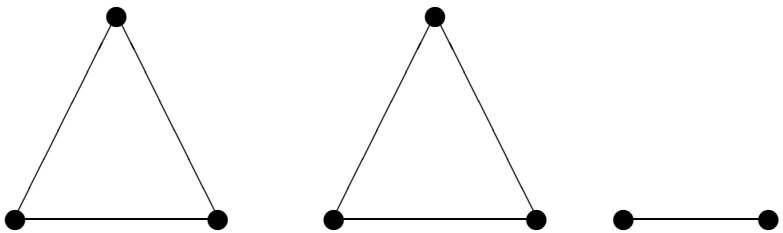
R is an equivalence relation on the set of vertices of G.

Each equivalence class of R, together with the edges between vertices in the class,
is called a *connected component* of G.

A graph is called *connected* if it has just one connected component.

8.10

Example



has three connected components, so it is not connected.

8.11

To find a connected component

When the graph is drawn on a page, we can just see the connected components.

(Important and difficult question: How do our brains do this?
Designers of automatic pilots (for example) would like to know.)

When the graph is described by an adjacency matrix and the answer has to be found by a computer,
we need a precise method.

8.12

Given a vertex v in a graph G:

1. Put $C = \{v\}$.
2. Let C' be the set of all vertices joined by an edge to some vertex in C.
   If C' is a subset of C, return C.
   If C' is not a subset of C, put $C := C \cup C'$ and repeat step 2.

The connected component of G containing v is the set of vertices returned, together with all the edges joining them.

8.13

Example

	s	t	u	v	w	x	y	z
s	0	0	0	0	0	0	1	0
t	0	0	0	0	0	1	1	0
u	0	0	0	1	1	0	0	1
v	0	0	1	1	1	0	0	1
w	0	0	1	1	0	0	0	1
x	0	1	0	0	0	1	1	0
y	1	1	0	0	0	1	0	0
z	0	0	1	1	1	0	0	0

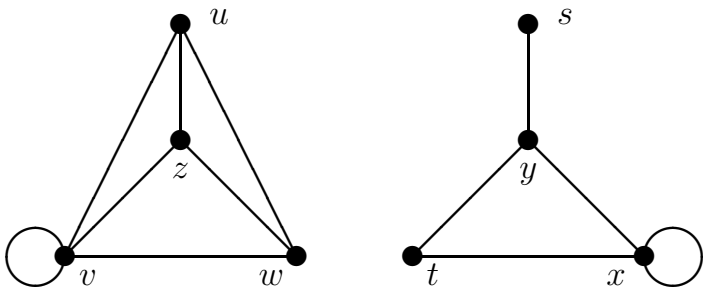
8.14

Steps calculating the connected component containing t:

1. $C = \{t\}$.
2. $C' = \{x, y\}$. We put $C = \{t, x, y\}$.
3. $C' = \{s, t, x, y\}$. We put $C = \{s, t, x, y\}$.
4. $C' = \{s, t, x, y\}$. So we return $\{s, t, x, y\}$.

8.15

We can check this result by drawing the graph from its adjacency matrix 8.13:



8.16

We say that a graph H is a *subgraph* of G if

(a) Every vertex of H is also a vertex of G, and

(b) every edge of H is also an edge of G, with the same endpoints.

For example:

- Every graph is a subgraph of itself.
- Every connected component of G is a subgraph of G.

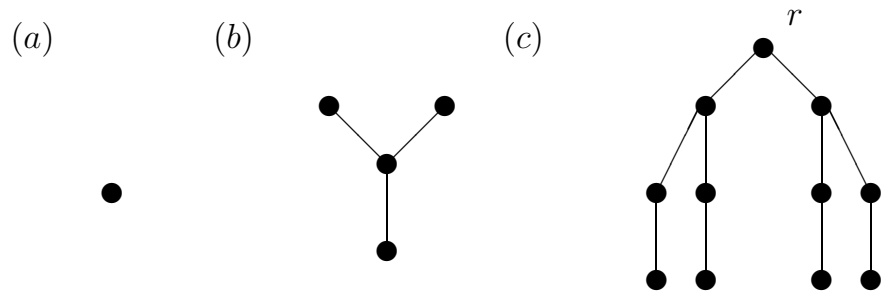
8.17

A *tree* is a connected graph with no cycles of length > 0.
(For brevity we just say 'with no cycles'
and ignore cycles of length 0.)

In a tree, if u and v are distinct vertices
then there is exactly one path from u to v.
Also every graph which has this property and has no cycles
of length 1 is a tree.

8.18

Examples of trees



The tree on the right has one vertex picked out, marked r.
A tree with a 'distinguished vertex' is called a *rooted tree*.

8.19

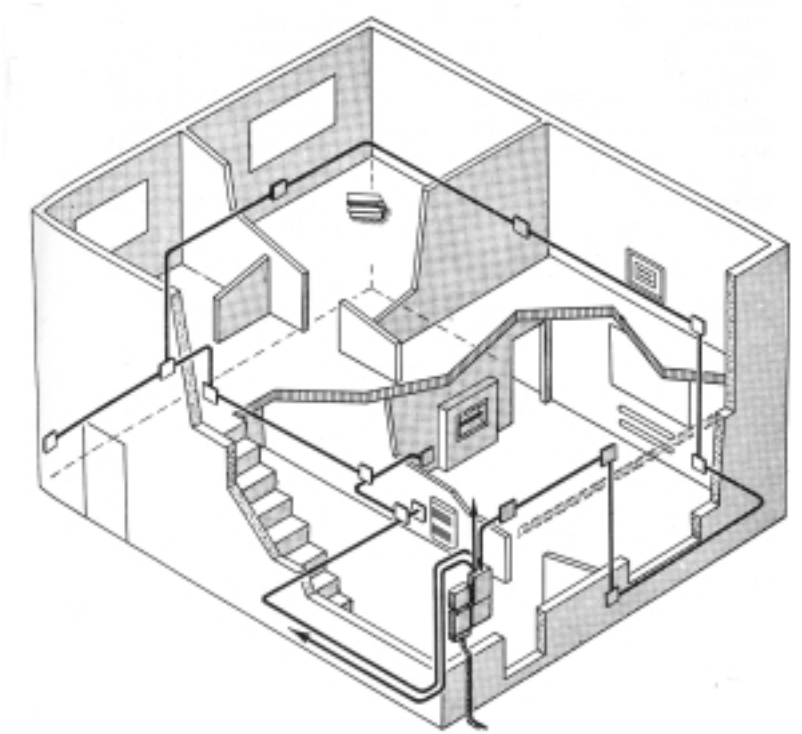
Suppose G is a connected graph and e is an edge of G.
and taking away the edge e makes G not connected.
Then we say that taking away e *disconnects* the graph.

Suppose we keep taking away edges without disconnecting
the graph, for as long as we can. The graph H that is left

- is a subgraph of G,
- has the same vertices as G, and
- is a tree.

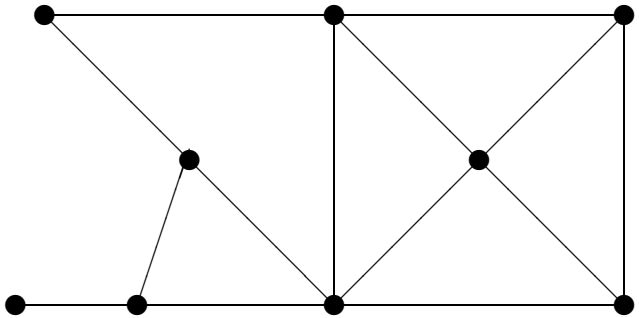
A graph H with these properties is a *spanning tree* of G.

8.20



8.21

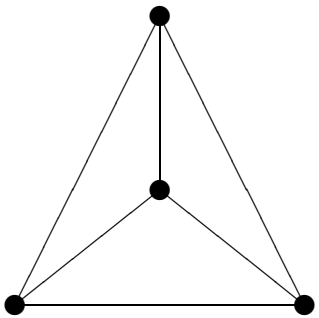
Example: The wiring diagram of the power supply in a house.



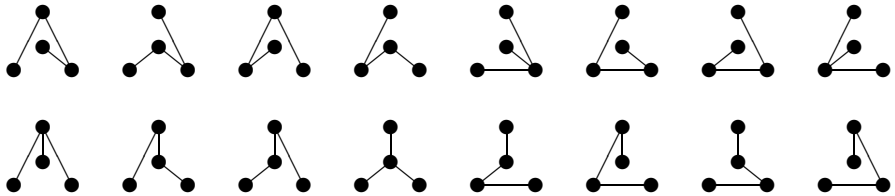
A spanning tree will still bring power to all the same sockets. So why do we use ring mains?

8.22

From Exam 2004: Find all the spanning trees in the following graph.



8.23



There are more than you expected.
It pays to be systematic in looking for them.

8.24

Digraphs

A *directed graph*, or for short a *digraph*, is the same as a graph, except that each edge has a direction marked by an arrow.

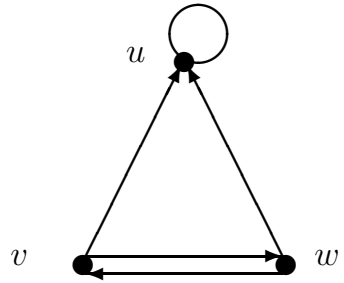
A *directed path* in a digraph is a path

$$v_0, e_0, v_1, e_1, \ldots, e_{n-1}, v_n$$

where each edge e_i runs forwards from v_i to v_{i+1}.

8.25

Example:



There is no need for an arrow on the loop at u.

8.26

The *adjacency matrix* of a digraph G has in row u and column v the number of edges that go *forwards*
from vertex u to vertex v.

The adjacency matrix of the digraph in 8.23:

	u	v	w
u	1	0	0
v	1	0	1
w	1	1	0

8.27

Networks

A *network* is a connected digraph N where each edge e has a number $c(e) \geqslant 0$ associated to it, called the *capacity* of e.

The *sources* of N are the vertices that have no arrows going to them,
and the *sinks* of N are the vertices that have no arrows going from them.

8.28

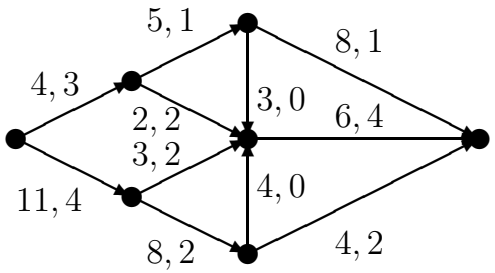
A *flow* on a network is a function f whose domain is the set of edges, such that

- for each edge e, $0 \leqslant f(e) \leqslant c(e)$;
- at each vertex v that is neither a source nor a sink, the sum of $f(e)$ for e going from v is equal to the sum of $f(e)$ for edges e going to v.

Think of oil flowing along the edges. It can't flow backwards; an edge can't carry more than its capacity; and oil can't pile up or appear from nowhere at a vertex that is neither a source nor a sink.

8.29

Example In this network the first number on each edge is the capacity, the second is the flow.



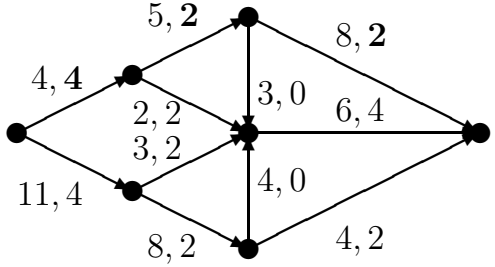
8.30

The *value* of the flow is the sum of the values of flows on edges leading out of the sources.
This is equal to the sum of the values of flows on edges leading into the sinks.

In the example above, there are just one source and just one sink.
The value of the flow at the source is $3 + 4 = 7$.
The value of the flow at the sink is $1 + 4 + 2 = 7$.

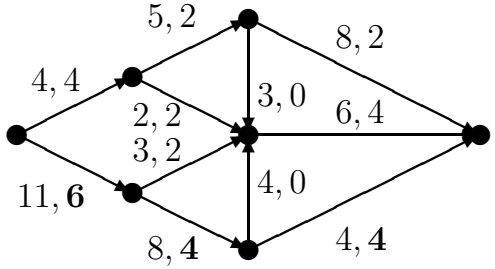
8.31

There is spare capacity all the way along the top of the network above.
We can increase the flow by 1:



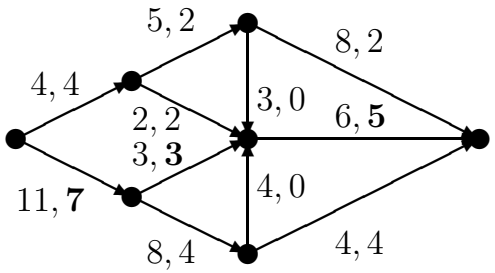
8.32

There is still spare capacity of 2 all the way along the bottom:



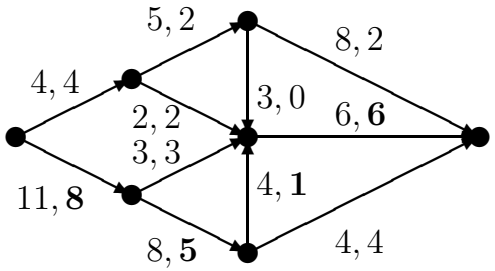
8.33

We can squeeze one more unit along a path through the middle:



8.34

And one more unit by a path that goes down and then up again:



8.35

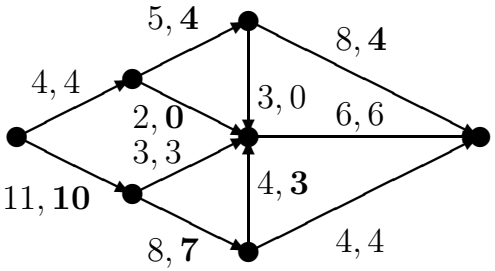
Can we squeeze a larger flow through this network?

Not by pushing larger amounts *forward* along directed edges.

But we can if we allow ourselves to push a *smaller* amount *backwards* along a directed edge:

8.36

And one more unit by a path that goes down and then up again:



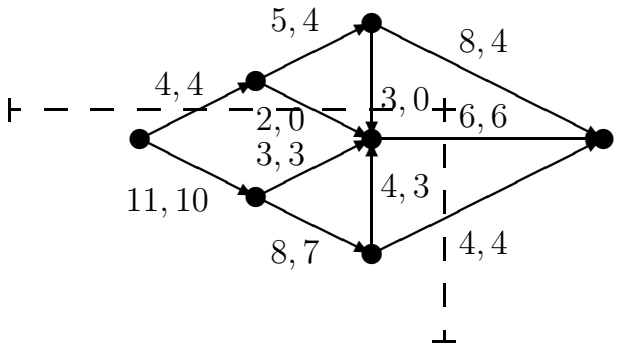
8.37

The flow is now $4 + 10 = 14$ at the source and $4 + 6 + 4 = 13$ at the sink, so its value is 14.

To check that this is the maximum possible, we look for a cut of value 14.
A *cut* is a line cutting some edges of the network, to separate the sources from the sinks,
and its *value* is the sum of the capacities of edges crossing the cut from left to right.

8.38

A cut of value 14 (= 4 + 0 + 0 + 6 + 4)



8.39

The max-flow min-cut theorem says that for every network there is a unique number f such that

(1) there is a flow of value f through the network;

(2) there is a cut through the network with value f.

This number f is the value of the maximum flow that we can get through the network.