

# CS1113

## Paths in Graphs

### Lecturer:

Professor Barry O'Sullivan

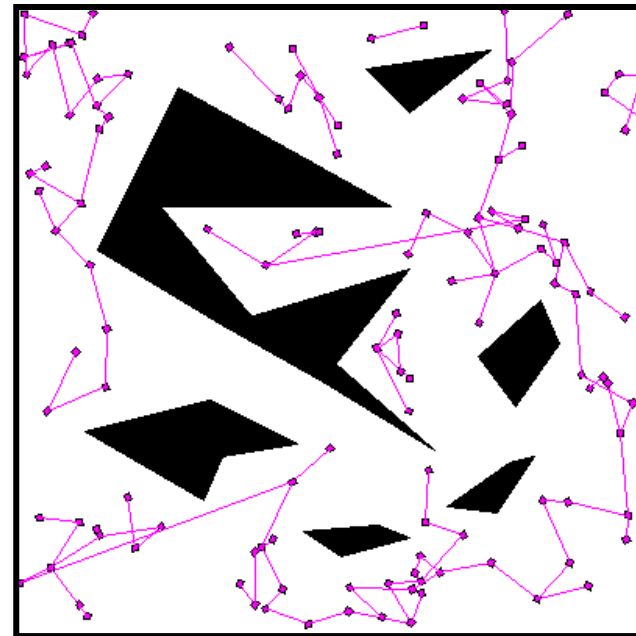
Office: 2.65, Western Gateway Building

email: *b.osullivan@cs.ucc.ie*

<http://osullivan.ucc.ie/teaching/cs1113/>

# Paths

paths  
connected graphs  
shortest paths

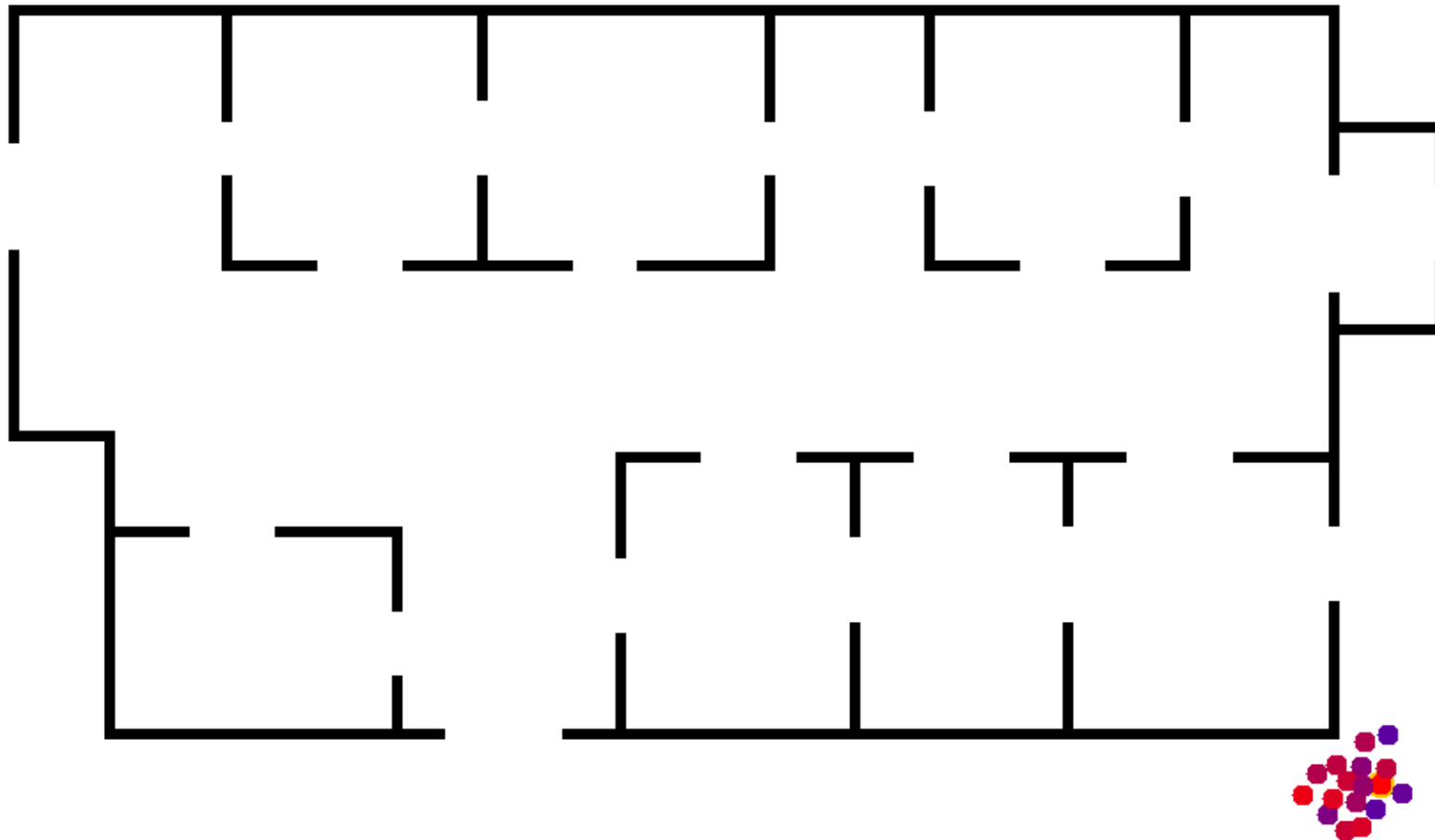


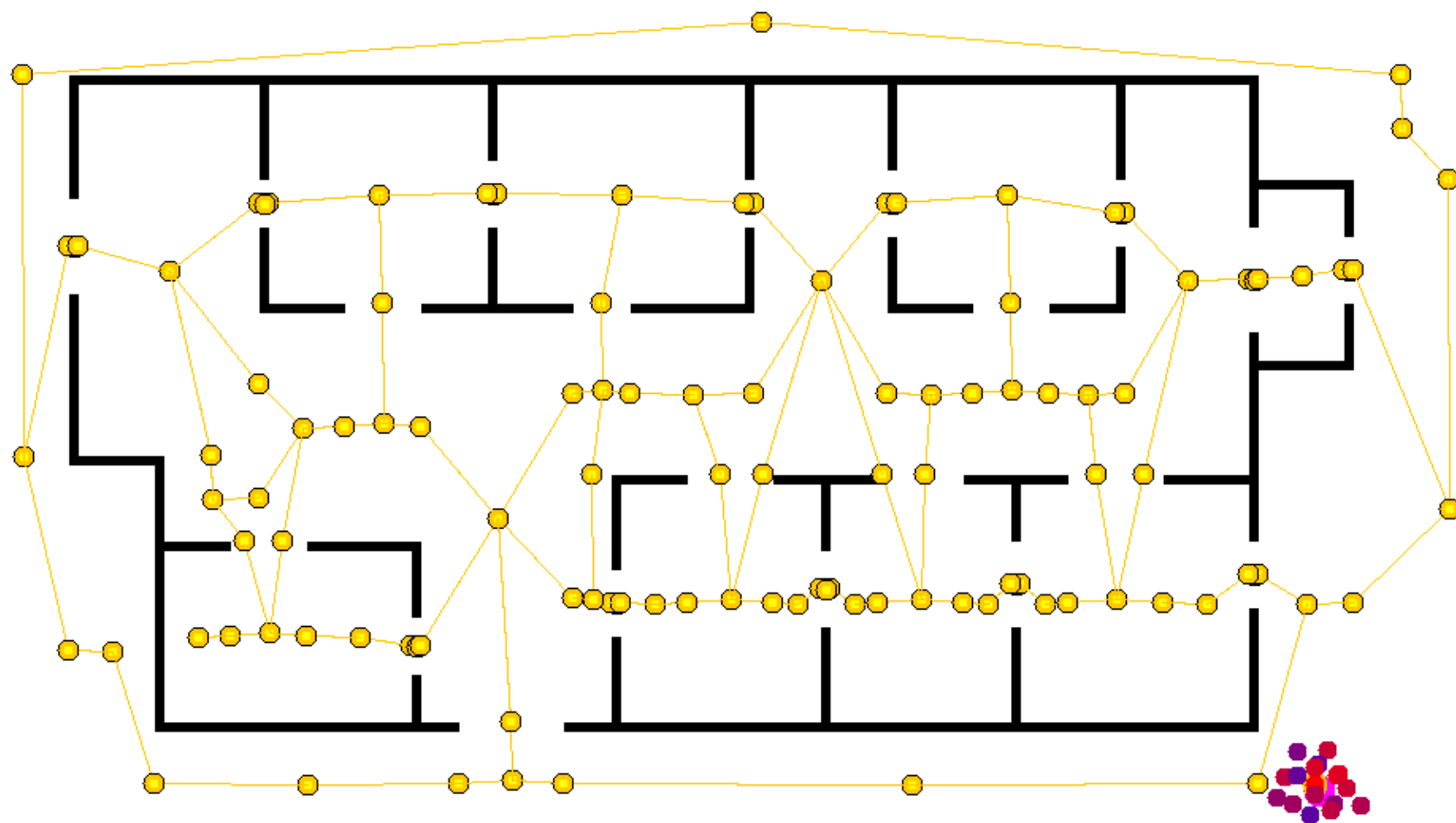
# Reasoning with graphs

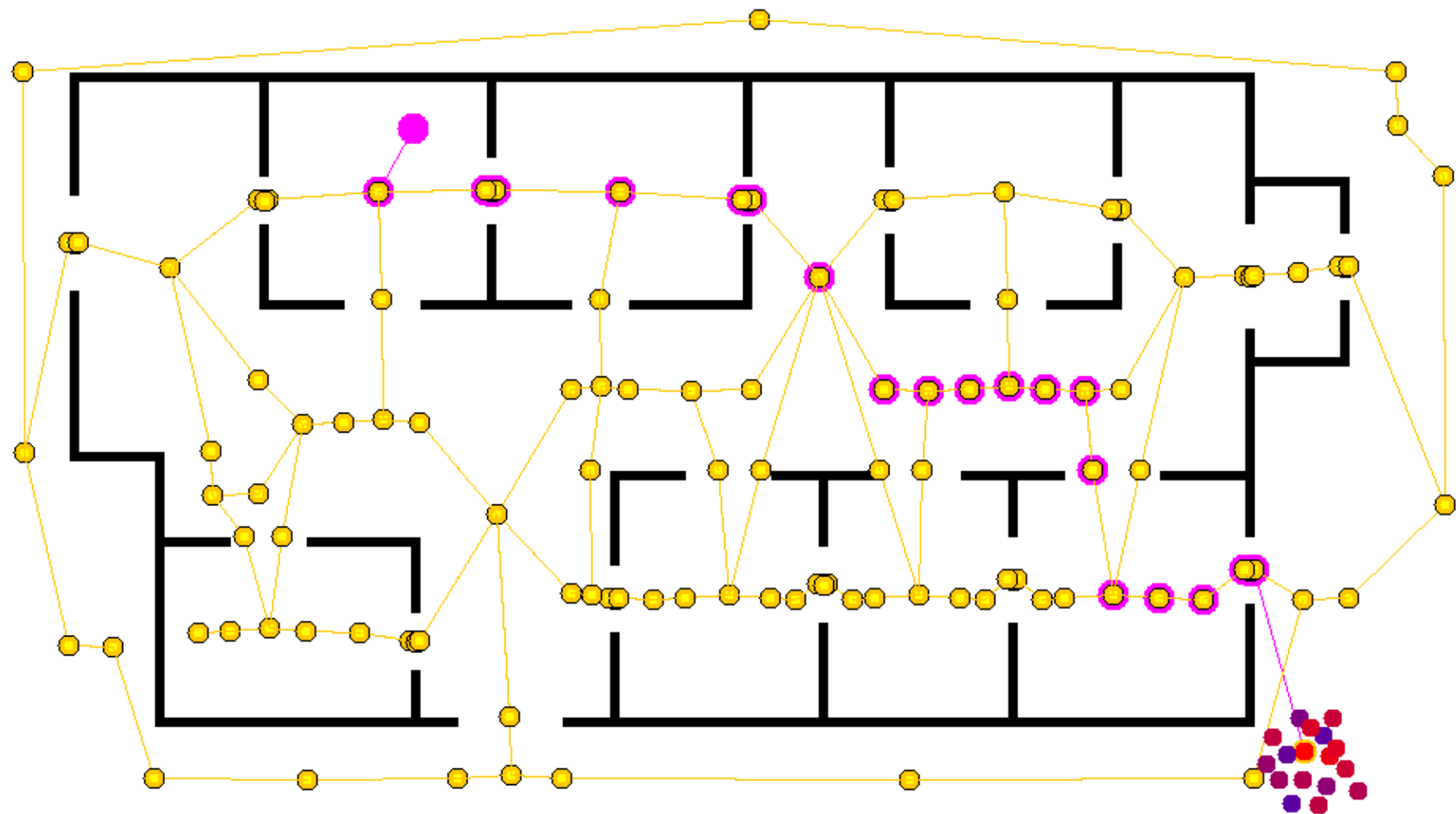
- in telecommunications networks, we will want to identify a route along which we can send data, packets or voice traffic to connect two users
- in route planning, we may want to compute the shortest route from one location to another, by moving across multiple connecting edges
- in social networks, we might want to ask how distant two individuals are (i.e. how many association links are needed to connect one person to another)

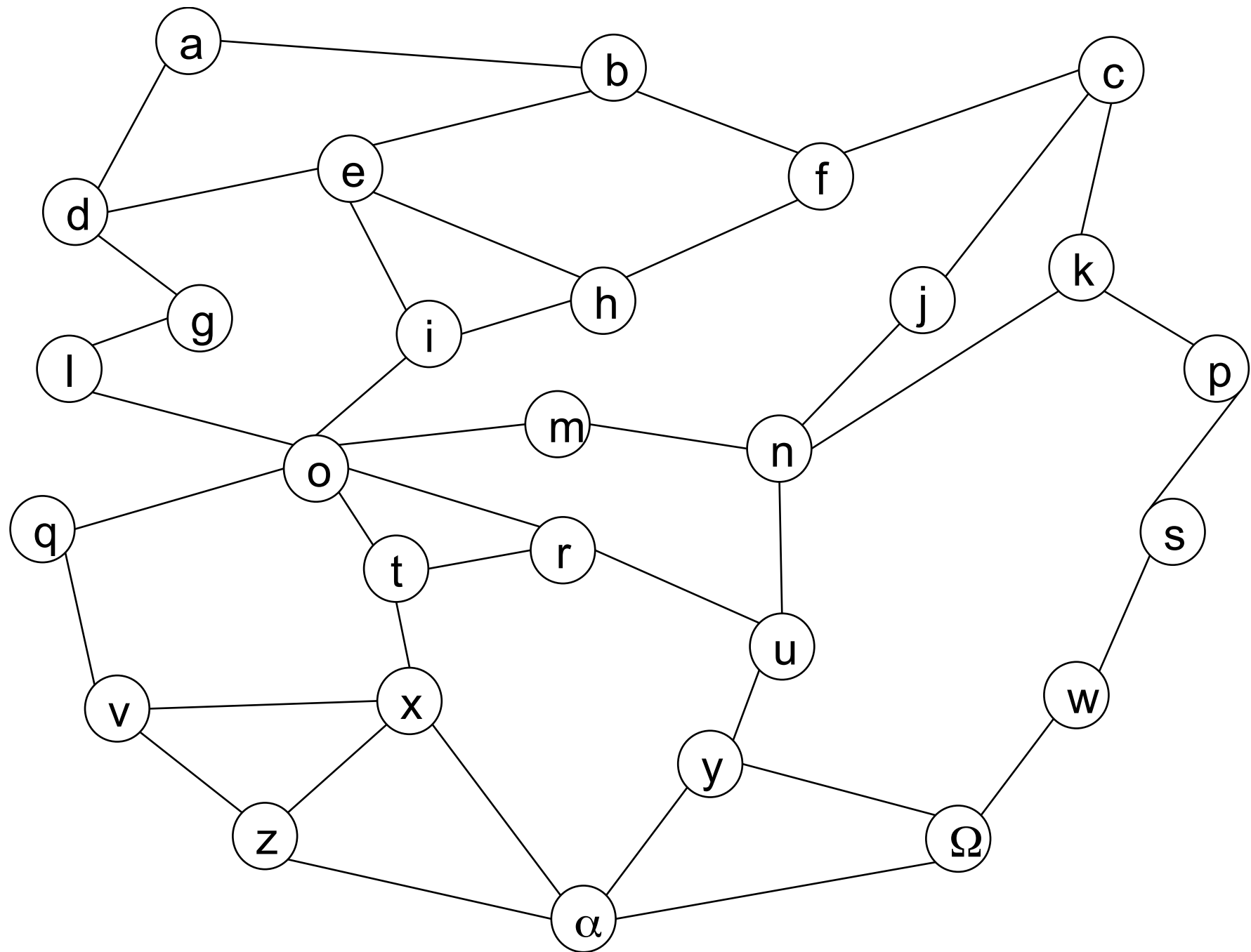
In each of these cases, we want to find a sequence of edges, and move along each edge in a particular direction, in order to connect two vertices

.....





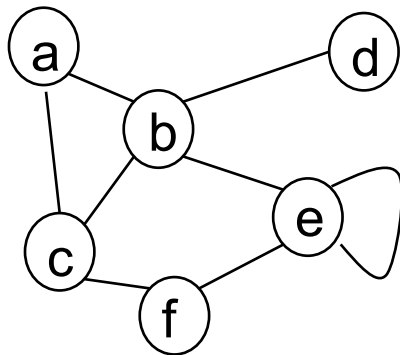




## Graphs (reminder)

A graph is an abstract representation of the relationships between multiple objects.

A simple graph  $G = (V, E)$ , where  $V$  is a set of vertices, and  $E$  is a set of edges, where each edge is a set containing 1 or 2 of the vertices from  $V$ .



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

A simple graph is a symmetric relation on  $V$

A simple graph has at most one edge between any pair of vertices. An edge can be from a vertex to itself, when it is called a **loop**.



$G = (V, E)$ , where

$V = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, \alpha, \Omega\}$

$E = \{ \{a, b\}, \{a, d\}, \{b, e\}, \{b, f\}, \{c, f\}, \{c, k\}, \{d, e\}, \{d, g\}, \{e, h\}, \{e, i\}, \{f, h\},$   
 $\{g, l\}, \{h, i\}, \{i, o\}, \{j, n\}, \{k, n\}, \{k, p\}, \{l, o\}, \{m, n\}, \{m, o\}, \{n, u\}, \{o, q\},$   
 $\{o, r\}, \{o, t\}, \{p, s\}, \{q, v\}, \{r, t\}, \{r, u\}, \{s, w\}, \{t, x\}, \{u, y\}, \{v, x\}, \{v, z\},$   
 $\{w, \Omega\}, \{x, z\}, \{x, \alpha\}, \{y, \alpha\}, \{y, \Omega\}, \{z, \alpha\}, \{\alpha, \Omega\} \}$

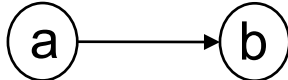
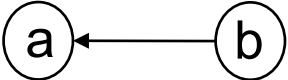
From last lecture:

Two vertices,  $v_1$  and  $v_2$ , are **adjacent** if there is an edge  $\{v_1, v_2\}$  in  $E$ .

# Oriented edges

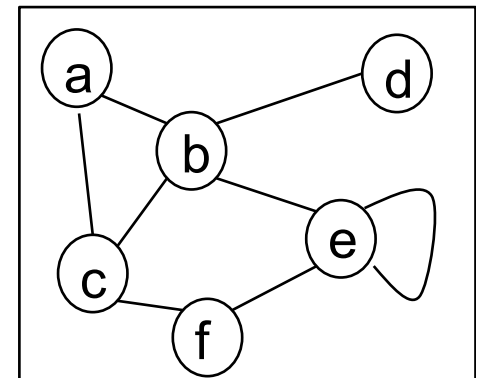
An **orientation** of an edge assigns a direction to that edge, selecting one of the edges as the **startpoint**.

Example: the edge  $\{a,b\}$  has two different orientations:

- $(a,b)$  with  $a$  as the startpoint 
- $(b,a)$  with  $b$  as the startpoint 

Example: the edge  $\{e\}$  has only one orientation:

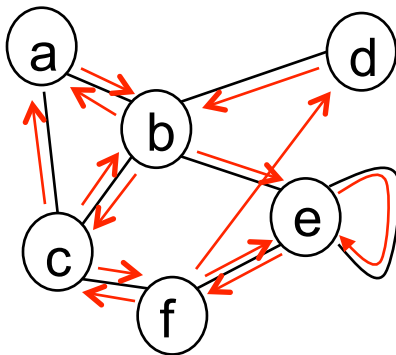
- $(e,e)$  with  $e$  as the startpoint 



# Paths

A **path** in a graph is a sequence of oriented edges, such that the endpoint of one oriented edge is the startpoint of the next oriented edge in the sequence.

Examples:



$\langle (a,b), (b,e), (e,f), (f,c) \rangle$  is a path

$\langle (a,b), (b,c), (c,a) \rangle$  is a path

$\langle (f,e), (e,e), (e,f), (f,c) \rangle$  is a path

$\langle (d,b), (f,c) \rangle$  is not a path

$\langle (c,f), (e,f) \rangle$  is not a path

$\langle (d,b), (c,f), (b,c) \rangle$  is not a path

$\langle (c,b), (b,a), (b,e) \rangle$  is not a path

$\langle (c,f), (f,d) \rangle$  is not a path

What about (i)  $\langle (b,c), (c,f), (f,e), (e,e), (e,b) \rangle$ ?

(ii)  $\langle (f,c), (c,b), (a,b), (b,d) \rangle$ ;

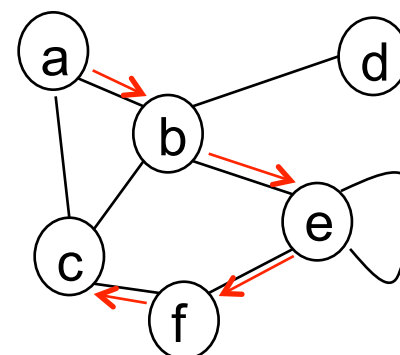
(iii)  $\langle (c,a) \rangle$ ?

For simple graphs, we can describe a path by listing the sequence of vertices it visits, since each pair of vertices is linked by at most one edge.

Example: the path  $\langle (a,b), (b,e), (e,f), (f,c) \rangle$  can be represented by  $\langle a,b,e,f,c \rangle$

For a path represented by a vertex sequence  $\langle v_1, v_2, \dots, v_n \rangle$ ,

- $v_1$  is the **start** of the path
- $v_n$  is the **end** of the path



Example: for the path  $\langle a,b,e,f,c \rangle$ ,  $a$  is the start,  $c$  is the end

The **length** of a path is the number of edges in the sequence (or the number of vertices minus 1)

## Multiple paths in simple graphs

For a given pair of vertices, there may be many different paths which connect them.

Example: to connect  $a$  to  $f$ , we have

$\langle a, b, c, f \rangle$

$\langle a, b, e, f \rangle$

$\langle a, b, e, e, f \rangle$

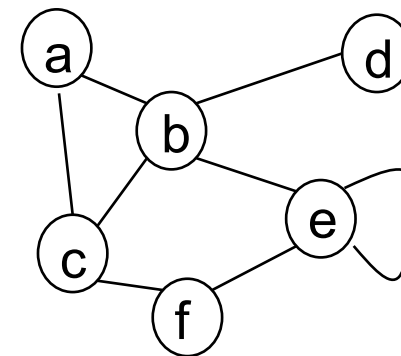
$\langle a, c, b, e, f \rangle$

$\langle a, c, b, e, e, f \rangle$

$\langle a, b, c, a, b, e, f \rangle$

$\langle a, c, f \rangle$

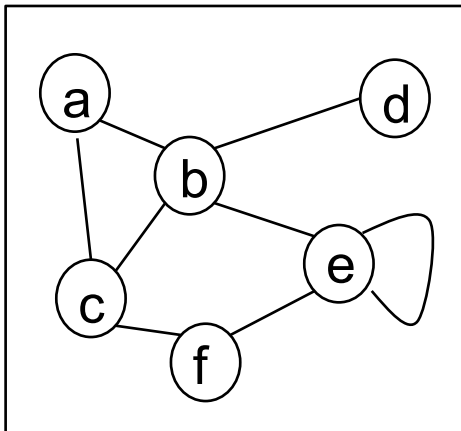
and many more ...



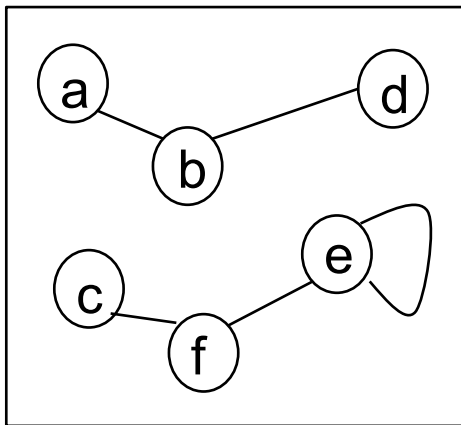
How do we find the shortest path?

# Connected graphs

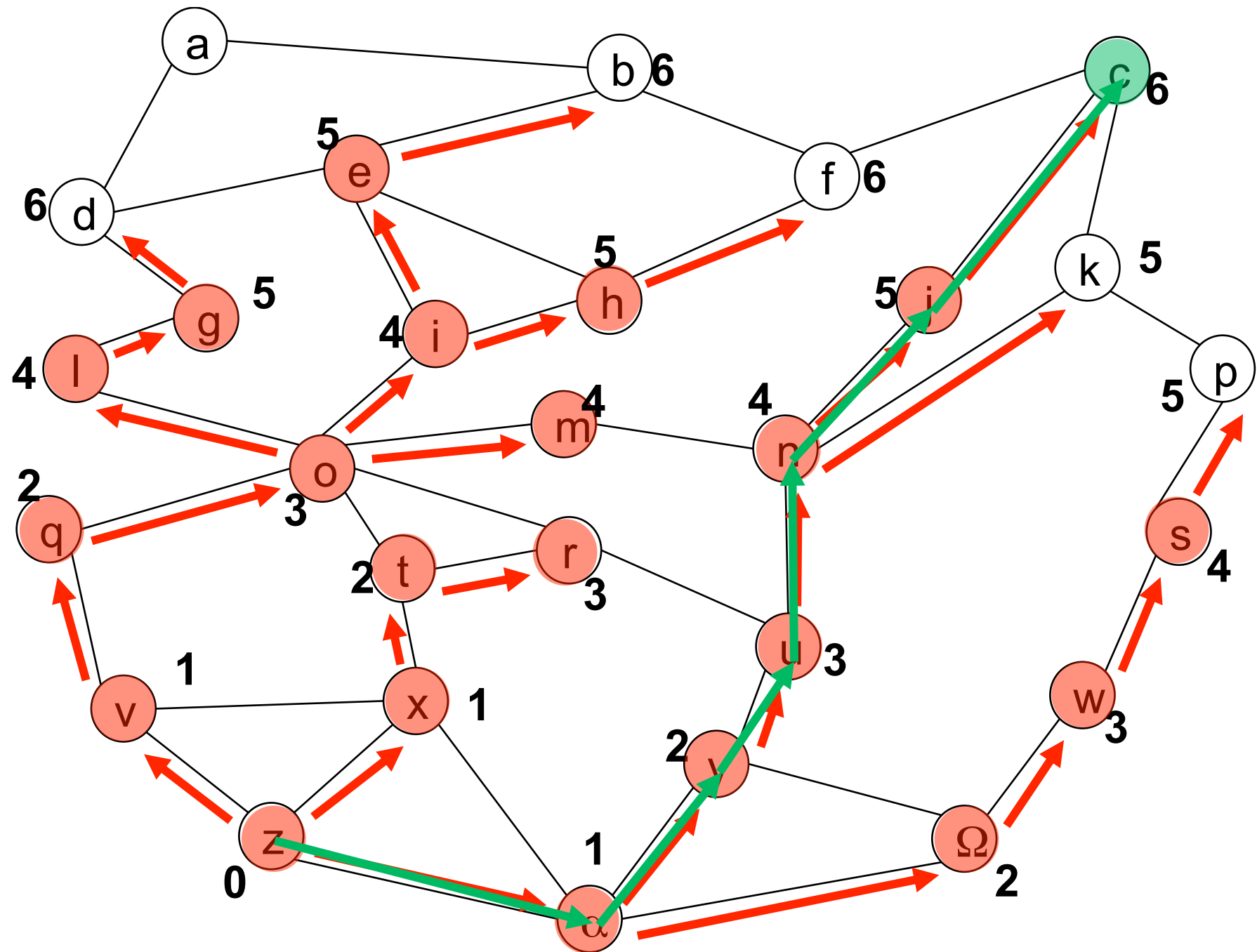
A graph is **connected** if every pair of vertices  $v, w$  can be connected by a path which starts at  $v$  and ends at  $w$ .



is a connected graph



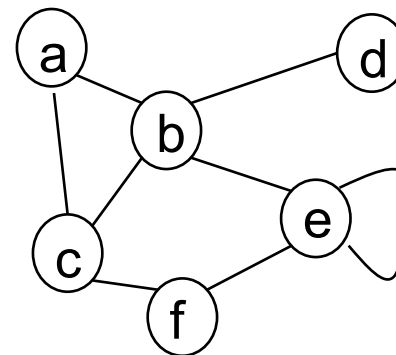
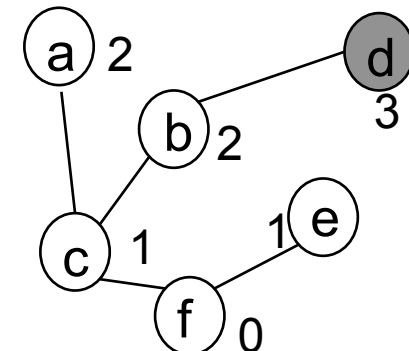
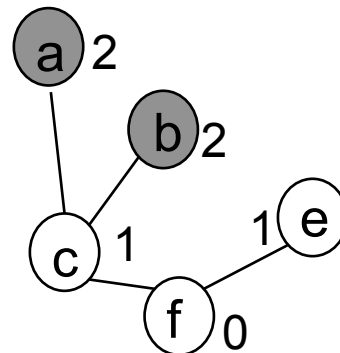
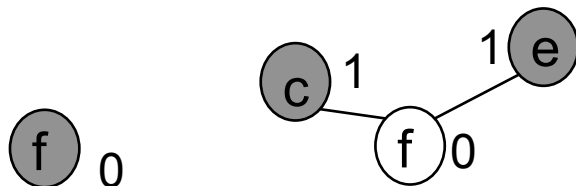
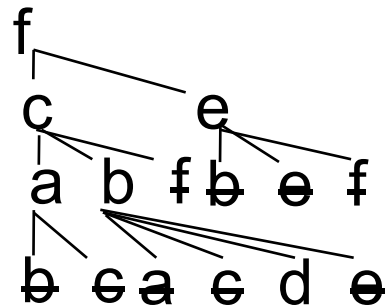
is not a connected graph – there is no path between, for example,  $a$  and  $f$ .



# Finding the shortest path (informal)

We start at the start vertex. We look at all vertices that are adjacent that we haven't seen before. If one of them is the end vertex, we stop; else we mark as being 1 hop away. We then take each 1-hop vertex in turn, and look at all vertices that are adjacent to them (ignoring any we have seen before). If one of them is the end vertex, we stop; else we mark as 2 hops. We then take each 2-hop vertex in turn, ... and so on. Either we find the end vertex via the shortest path, or the start vertex cannot be connected to the end vertex.

Example: find the shortest path from f to d



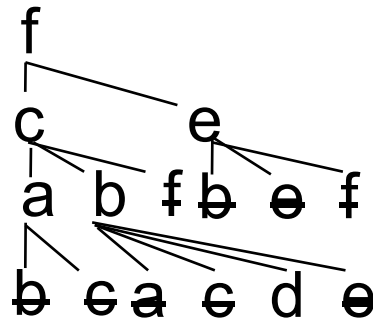
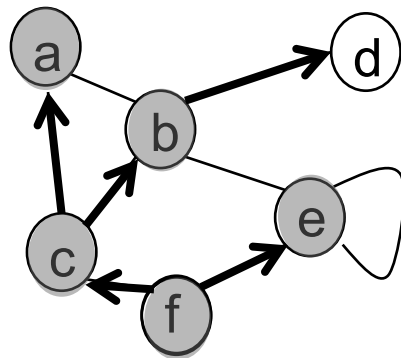
... but  
how do  
we record  
the path?



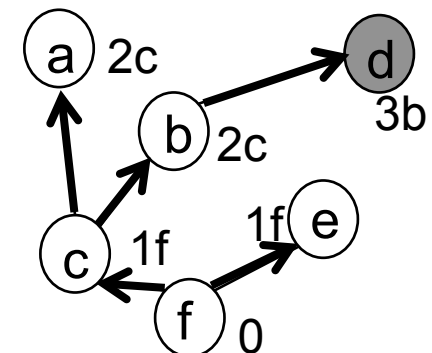
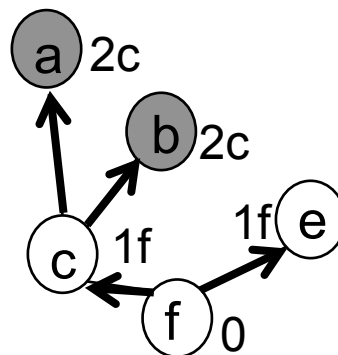
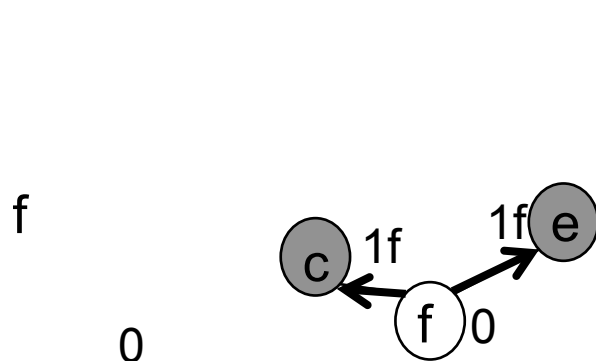
# Recording the shortest path

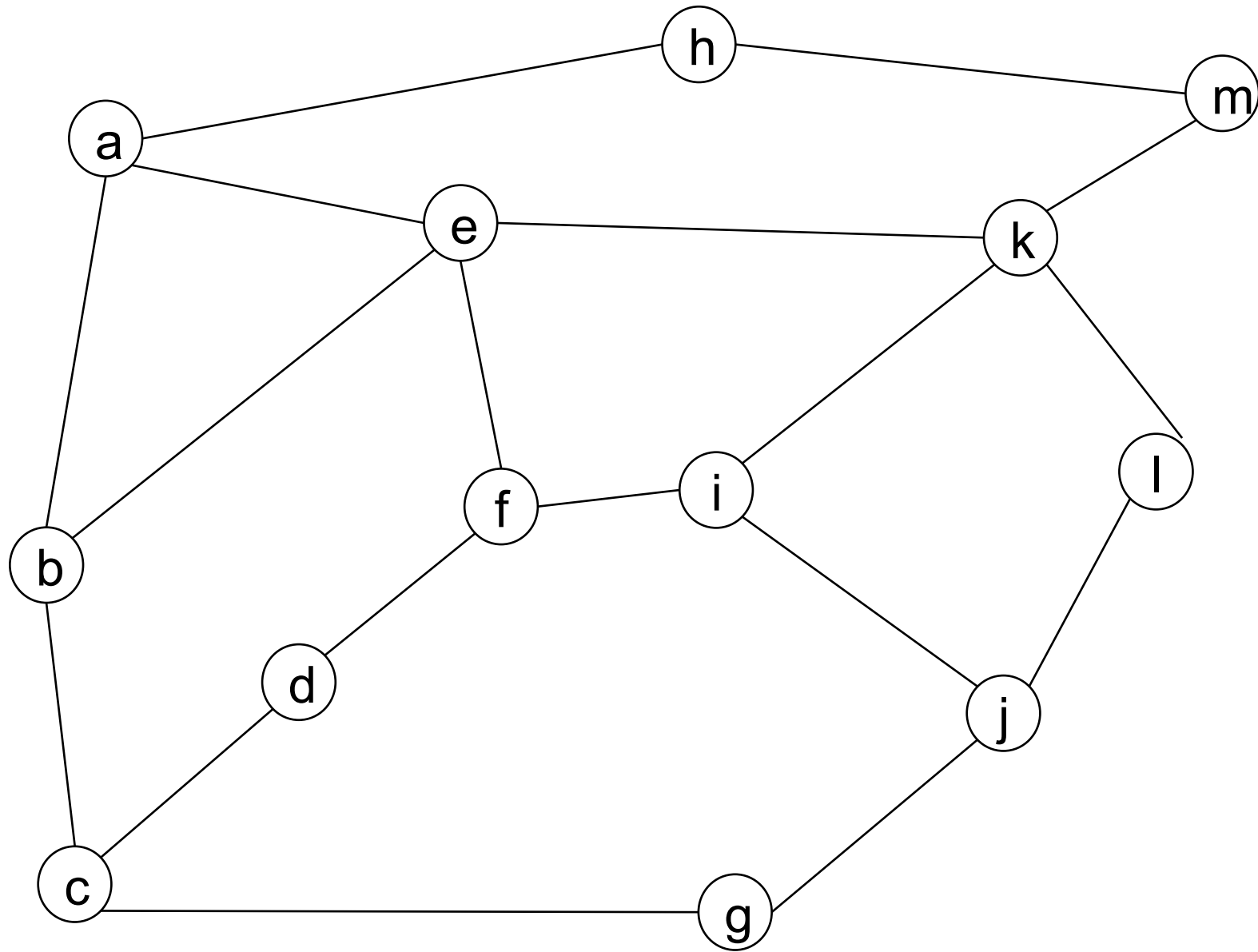
We will maintain a 2D array (i.e. a table), one column for each vertex in  $V$ . The first row says whether the vertex has been expanded; the second says the number of edges to get to it; the third says the vertex that precedes it on the shortest path from the start vertex.

Each time we expand a vertex  $v$  in the search, we look at all its adjacent vertices: if the corresponding cell already has a value, then we have seen it before; else, we assign  $v$  to the cell, to say we reached here from vertex  $v$ .



	a	b	c	d	e	f
done?	1	1	1		1	1
edges?	2	2	1	3	1	0
previous	c	c	f	b	f	0





# Algorithm for finding shortest path

Exercise: prove that the shortest path does not visit any vertex twice (you don't need this algorithm)

Algorithm: shortestPath

Input: a simple graph  $G=(V,E)$ , where  $V = \{1,2,\dots,n\}$

Input: a vertex  $x$  from  $V$ , the start

Input: a vertex  $y$  from  $V$ , the end

Output: an table representation of the path, or null if none

```
1.  L := a table with 3 rows for each of n vertices, all null
2.  v := x
3.  L[v][2] := 0           //the smallest number of edges to v from x
4.  L[v][3] := 0           //the previous vertex in the path to v
5.  while v is not null
6.      edges := L[v][2] + 1           //will grow path by one edge
6.      for each vertex j adjacent to v           //expand paths from v
9.          if L[j][2]== null           //found a shortest path to j
10.         then L[j][2] := edges           //update j's edge count
11.             L[j][3] := v           //say how we got here (v -> j)
12.             if j == y, return L           //stop- found shortest path to y
12.         L[v][1] := 1           //mark v as done
13.         v := vertex with smallest L[v][2] and with L[v][1]== null
14.             or null if there isn't one           //find shortest path to grow
16. return null           //we didn't reach the target by any path
```

Next lecture ...

Paths with travel times

Dijkstra's Algorithms for finding shortest paths