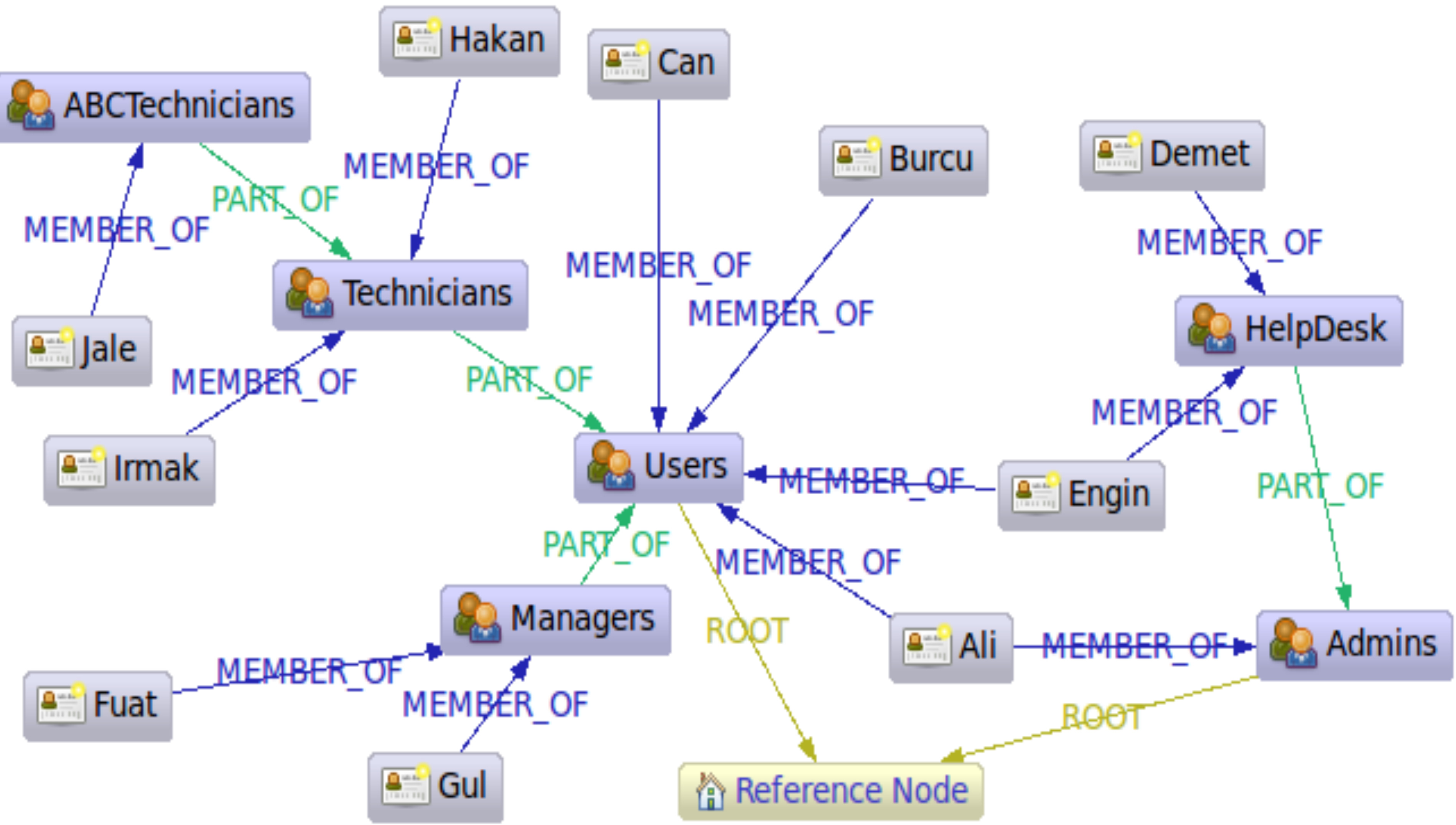
Directed Acyclic Graphs
Topological Sorts



Directed Graphs

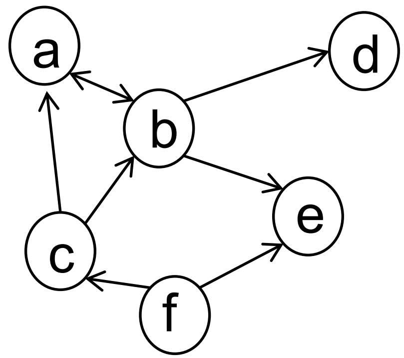
Earlier lecture

The ADT for Directed graphs is essentially the same as for undirected, except we treat the order of the vertices in an Edge as significant,

Edge

get_start()

get_end()



and we require the following methods for the Graph:

in_degree(x):	return the in-degree of vertex x
out_degree(x):	return the out-degree of vertex x
get_in_edges(x):	return a list of all edges pointing in to x
get_out_edges(x):	return a list of all edges pointing away from x

Paths in a Directed Graph

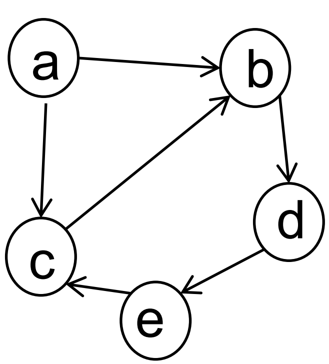
A *path* from v_0 to v_k in a directed graph G is a sequence of vertices

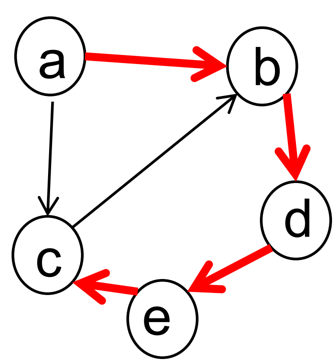
$$\langle v_0, v_1, v_2, \ldots, v_k \rangle$$

such that (v_i, v_{i+1}) is a directed edge in G for each i from 0 to k-1.

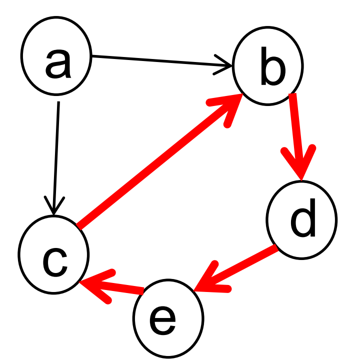
The length of the path is the number of vertices - 1.

A *cycle* in a directed graph is a path of length ≥ 1 which starts and finishes at the same vertex (so at least one edge).



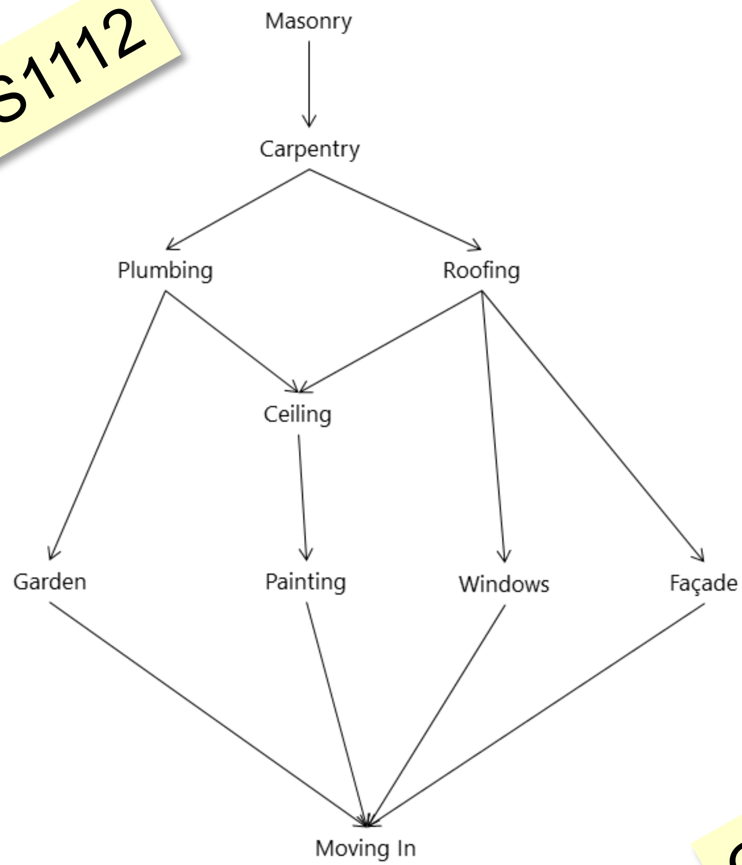


path



cycle

CS1112



A scheduling graph should not have a cycle

A set of tasks to be completed when building a house:

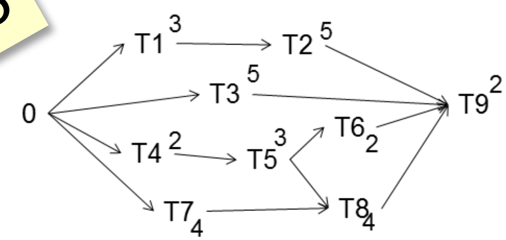
{Masonry, Carpentry, Roofing, Plumbing, Ceiling, Windows, Facade, Garden, Painting, Moving In}

Some tasks must be completed before others are finished.

Project Planning

A set of tasks to complete. Graph shows precedence order.

CS1113



Numbers show time to complete the task

Note: nodes now have a label

How quickly can I finish all my tasks?
How many resources would I need?

Note: links now have a direction

If I have 2 resources, how quickly can I finish?

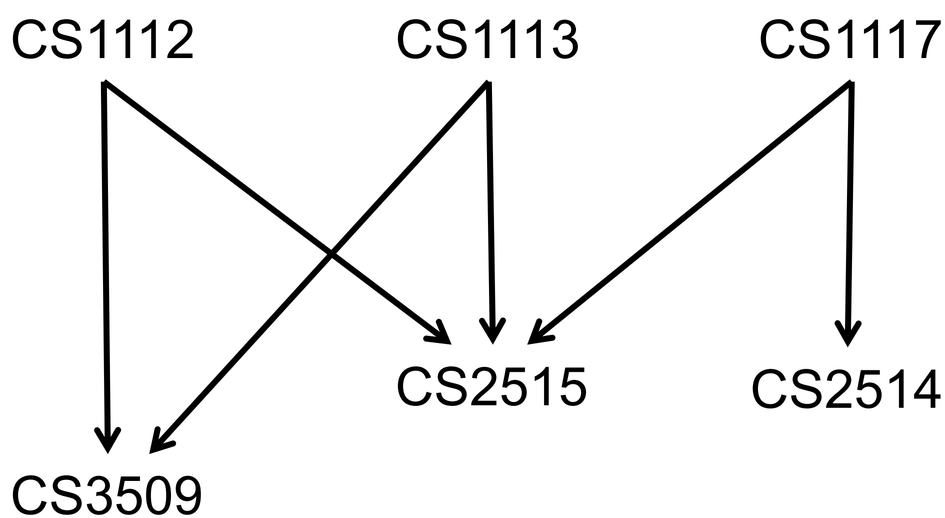
UCC Book of Modules

CS3509 Theory of Computation
Pre-requisite(s): CS1112, CS1113

CS2515 Algorithms and Data Structures I
Pre-requisite(s): CS1112, CS1113, CS1117

CS2514 Introduction to Java
Pre-requisite(s): CS1117

The prerequisite graph implied by the Book of Modules must not have a cycle



```
class Shape:
...
```

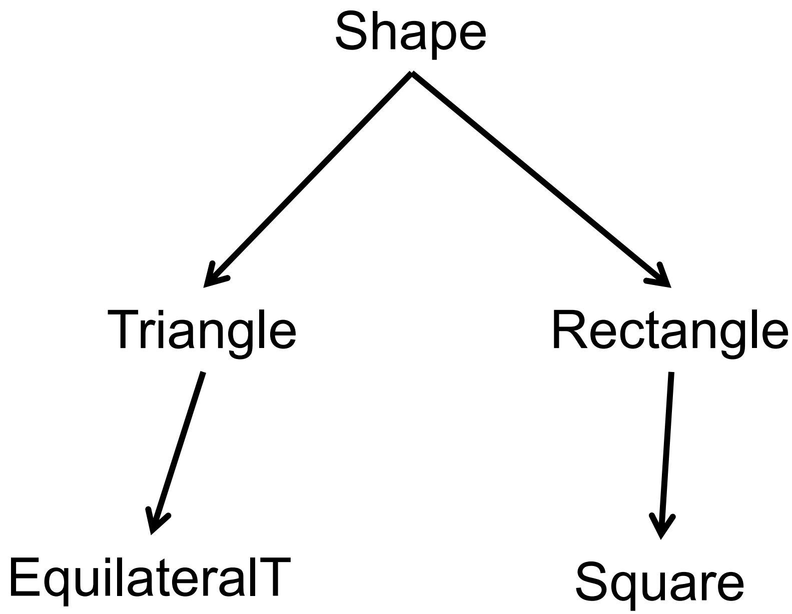
```
class Triangle(Shape):
...
```

```
class EquilateralT(Triangle):
...
```

```
class Rectangle(Shape):
...
```

```
class Square(Rectangle):
...
```

The inheritance graph implied by your programs in Python must not contain a cycle



A partial order is a binary relation defined over a single set.
The relation R must be:

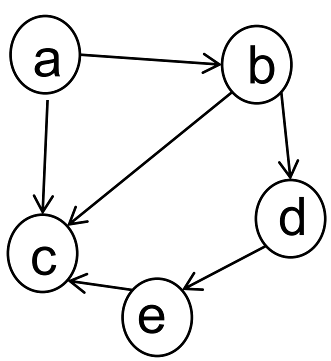
- anti-symmetric
  for any pair of vertices x and y, if x != y and xRy, then y ¬R x
- transitive
  for any three vertices x, y and z, if xRy and yRz, then xRz

We can draw any binary relation over a single set as a directed graph.
If the relation is an order, then it cannot contain a cycle.

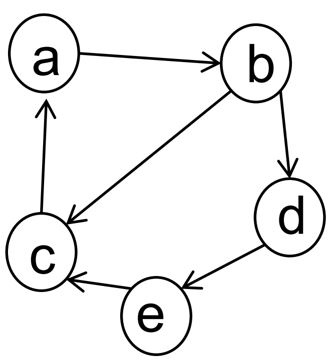
Directed Acyclic Graphs

The property of 'not containing a cycle' in a directed graph is important for many applications, data structures and algorithms.

We define the class of graphs which are directed but do not contain a cycle as *directed acyclic graphs*, or DAGs



DAG



not a DAG ...

Any Directed Acyclic Graph must have at least one vertex with in-degree == 0

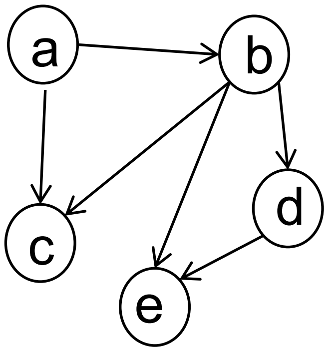
Proof by construction:

From any vertex, travel backwards over the in-edges until a vertex with 0 in-degree is reached.

Suppose no such vertex is reached. There must then be an infinite length path. But there are only finitely many vertices in the graph, so at least one vertex must appear twice. That means there is a cycle in the graph, starting and finishing at that vertex. Contradiction.

Topological Sort

Given a DAG, a *topological sort* is an ordered sequence of all vertices in the graph such that if two vertices *x* and *y* in the graph have a directed edge (*x*,*y*), then *x* appears before *y* in the sequence.



a,b,c,d,e
a,b,d,c,e
a,b,d,e,c

are all topological sorts of the graph

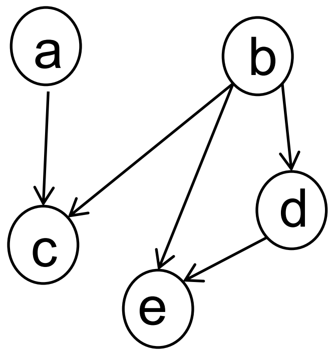
a,b,e,d,c

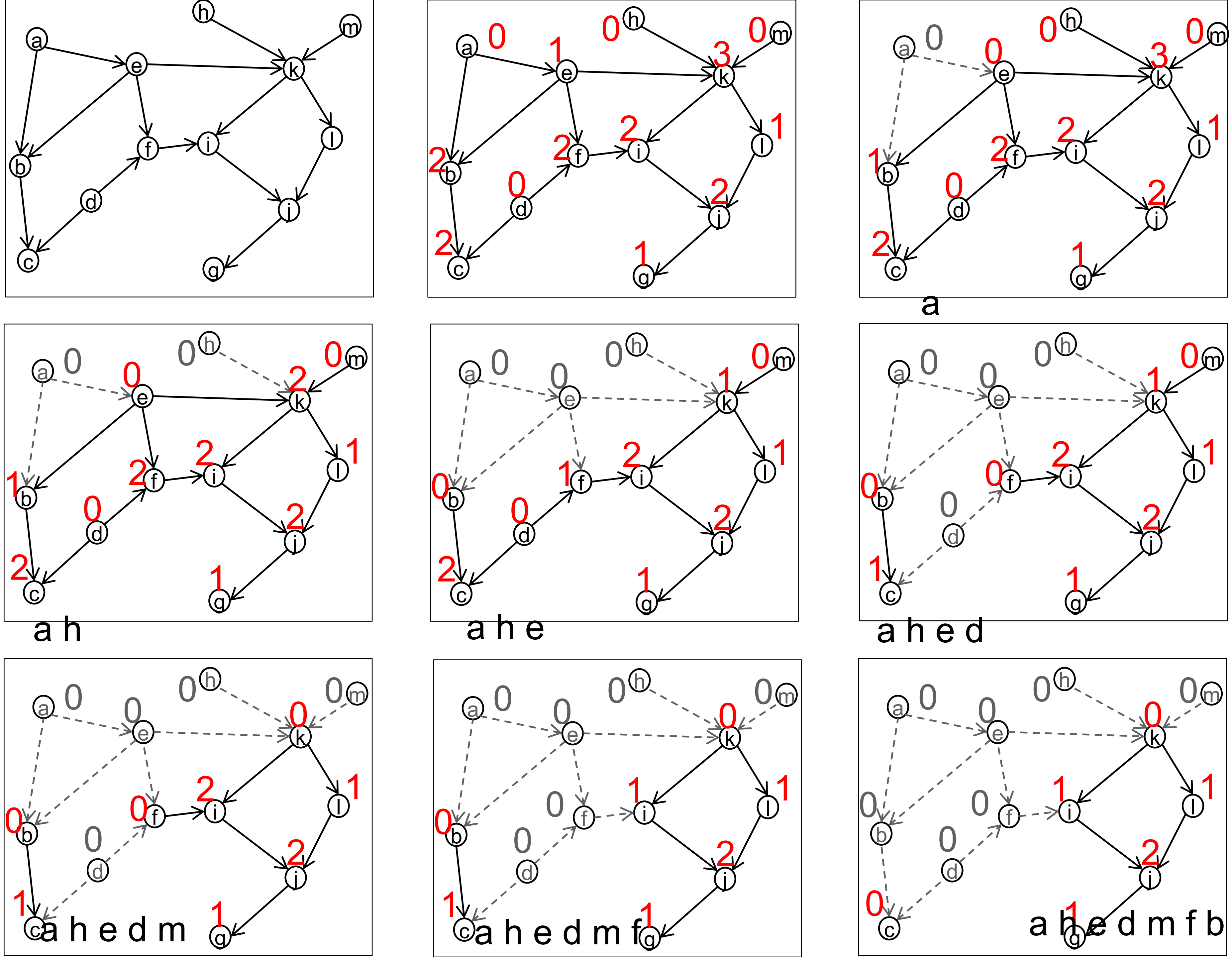
is not a topological sort of the graph. Why not?

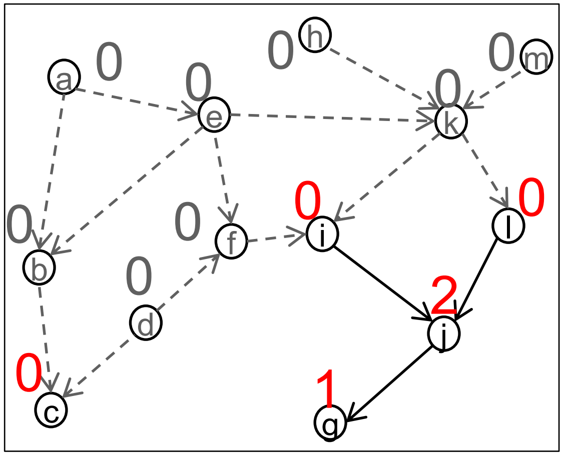
d must be before e

Generating Topological Sorts

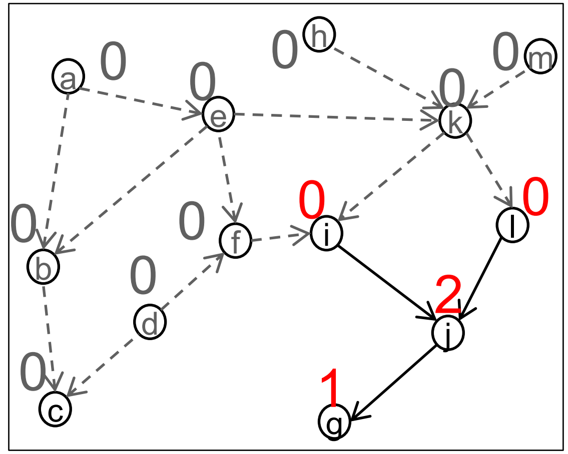
Write an algorithm to generate a topological sort for a DAG.



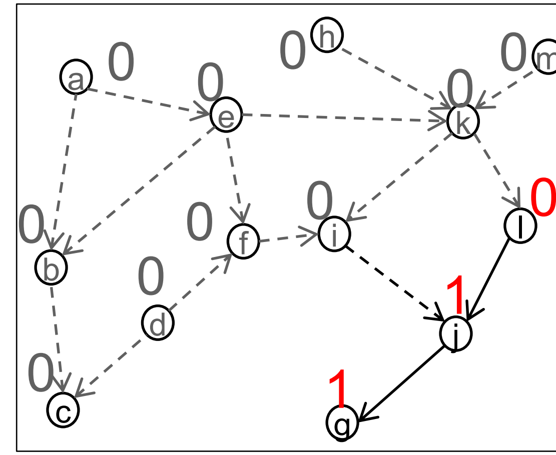

a
a h
a h e
a h e d
a h e d m
a h e d m f
a h e d m f b



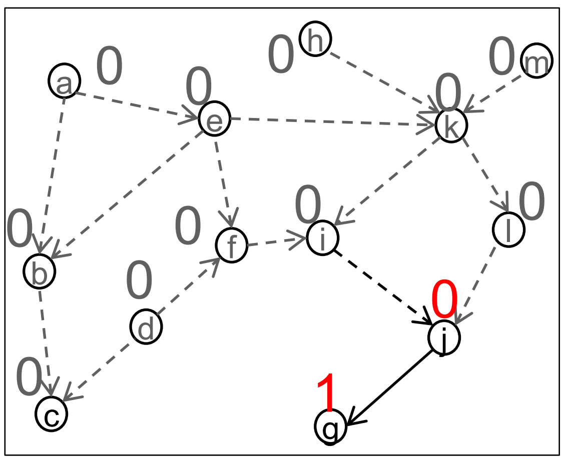
a h e d m f b k



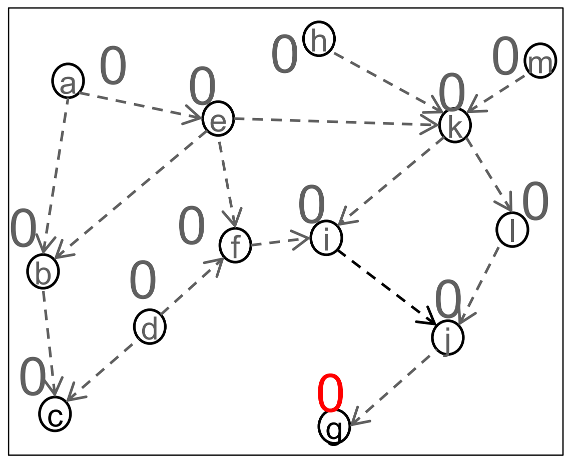
a h e d m f b k c



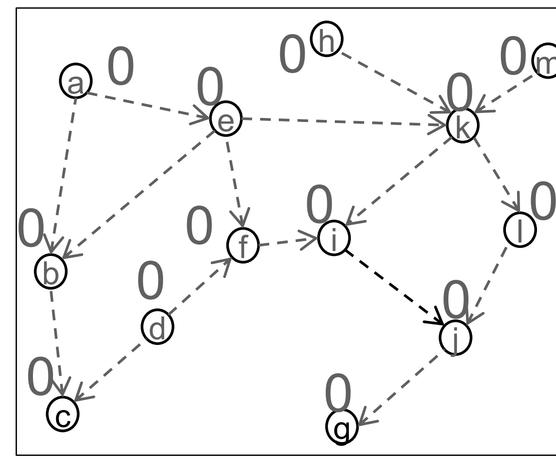
a h e d m f b k c i



a h e d m f b k c i l



a h e d m f b k c i l j



a h e d m f b k c i l j g

```python
def topological_sort(self):
    #assumed to be operating on a DAG
    inedgecount = {} #map of (vertex:in_degree) pairs
    tsort = []       #list of vertices in sort order
    available = []   #vertices with no active in-edges

    for v in self._structure:
        v_incount = self.in_degree(v)
        inedgecount[v] = v_incount
        if v_incount == 0:
           available.append(v)

    while len(available) > 0:
        w = available.pop()
        tsort.append(w)
        for e in self.get_edges(w):
            u = e.opposite(w)
            inedgecount[u] -= 1
            if inedgecount[u] == 0:
                available.append(u)

    return tsort
```

A directed graph G is a DAG if and only if G has a topological sort.

Proof?

Suppose G is a DAG. Then it must have a vertex with in-degree 0, and so in the topological sort algorithm, it enters the while loop. Each time round the loop, updating the in-degree count simulates removing the vertex and its out-edges from the graph. This would leave a smaller DAG, since we didn't add any edges, and that smaller DAG must have a vertex with in-degree == 0, and so we continue around the loop. The process terminates when all vertices have been added. By design, we never violated the ordering of an edge, and so the output is a topological sort.

Suppose G has a topological sort.
Assume G is not a DAG. Then there is a cycle vi ->x1 ->x2 -> ... ->xn -> vi. xn must be before vi in the tsort. But x(n-1) must be before xn in the tsort, and so on, by transitivity, and so x1 is before vi. But the first edge in the cycle means there is an edge vi->x1 in the DAG. *Contradiction*, since the tsort never goes against a directed edge. So G must be a DAG.

Exercise:

Write an algorithm which takes an arbitrary directed graph, and correctly returns True if it is a DAG, and False if it is not.

Exercise:

What is the complexity of the topological sort algorithm?

Next lecture

Easter Vacation ...

Revision