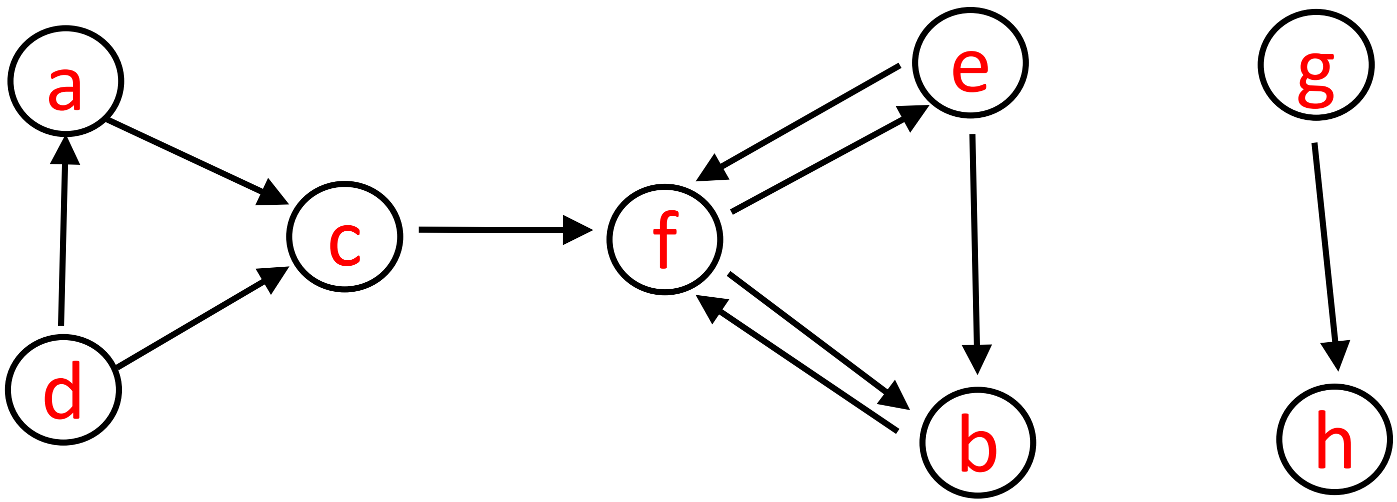
COMP 250

Lecture 28

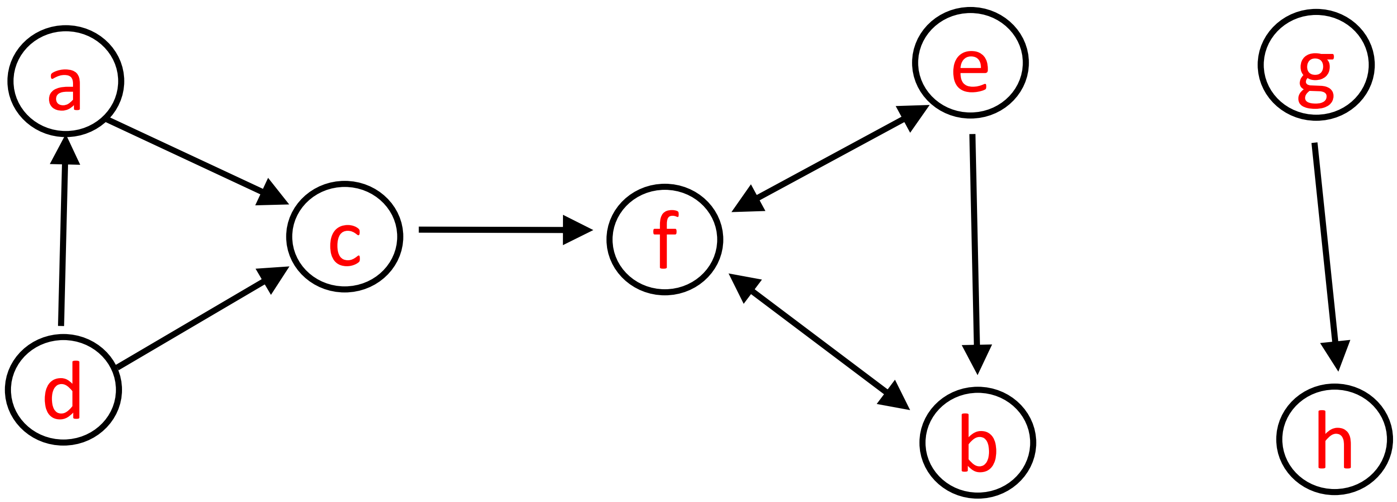
graphs

Nov. 13, 2017

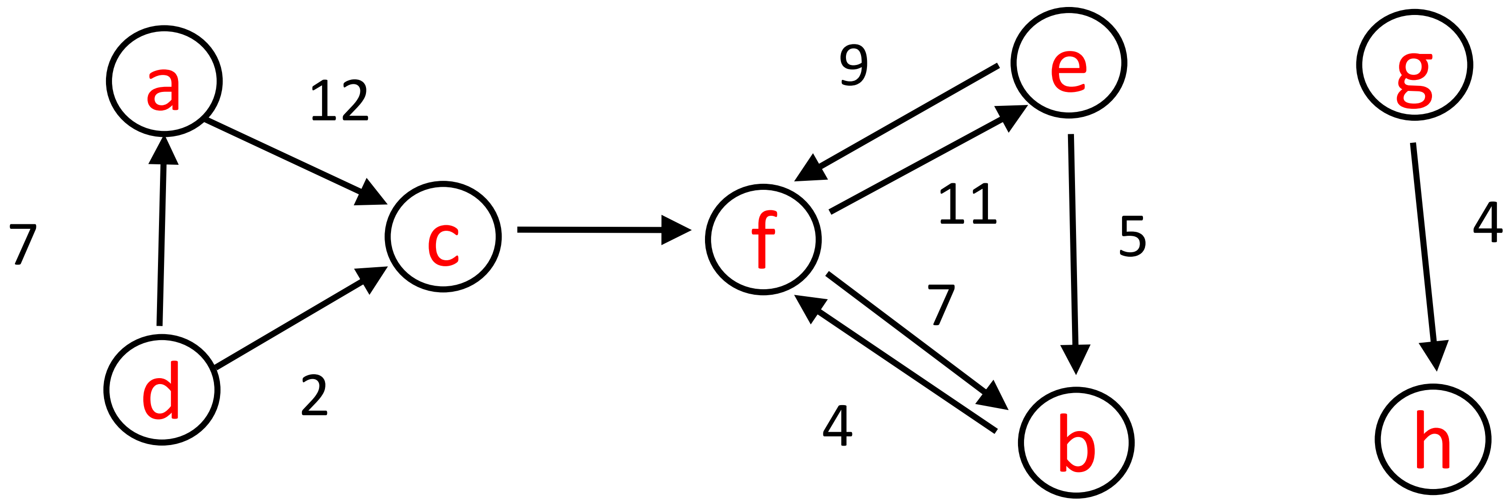
Example



Same Example – different notation



Weighted Graph



Definition

A *directed graph* is a set of *vertices*

$$V = \{v_i \ : \ i \in 1, \ldots, n \}$$

and set of ordered pairs of these vertices called *edges*.

$$E = \{ \left(v_i, v_j\right) \ : \ i, j \in 1, \ldots, n \}$$

In an *undirected* graph, the edges are *unordered* pairs.

$$E = \{ \ \{v_i, v_j\} \ : \ i, j \in 1, \ldots, n \}$$

Examples

Vertices	Edges
airports	
web pages	
Java objects	

Examples

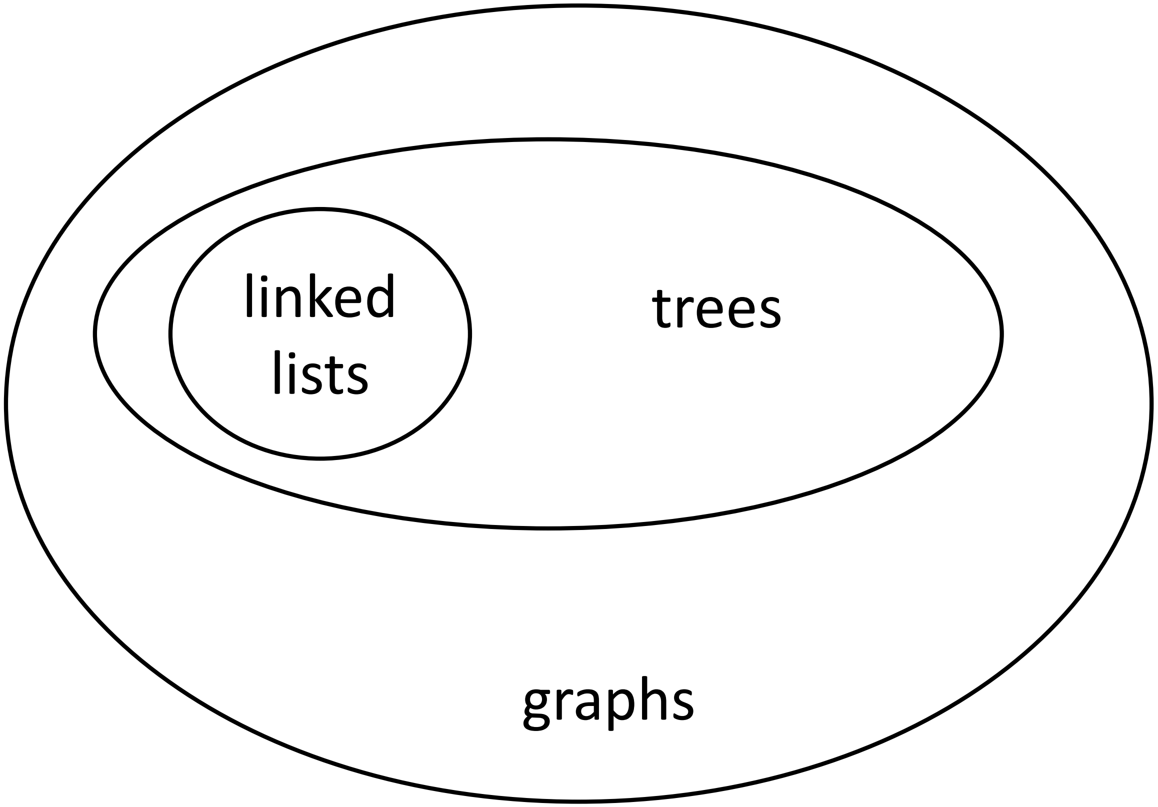
Vertices	Edges
airports	flights
web pages	
Java objects	

Examples

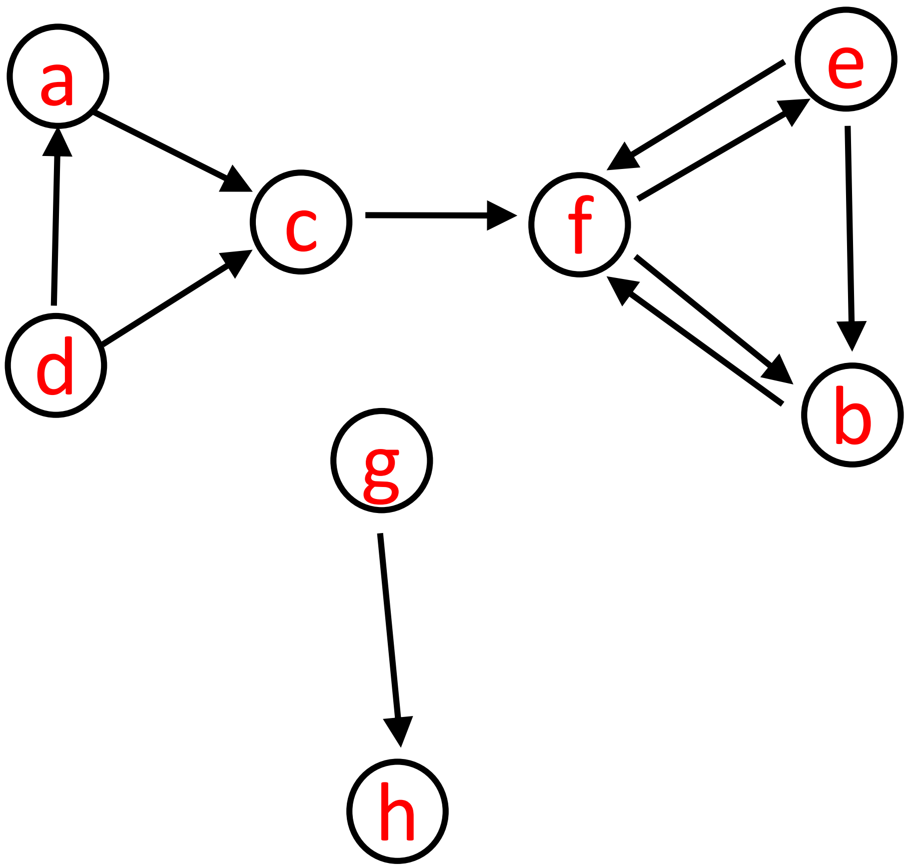
<u>Vertices</u>	<u>Edges</u>
airports	flights
web pages	links (URLs)
Java objects	

Examples

<u>Vertices</u>	<u>Edges</u>
airports	flights
web pages	links (URLs)
Java objects	references

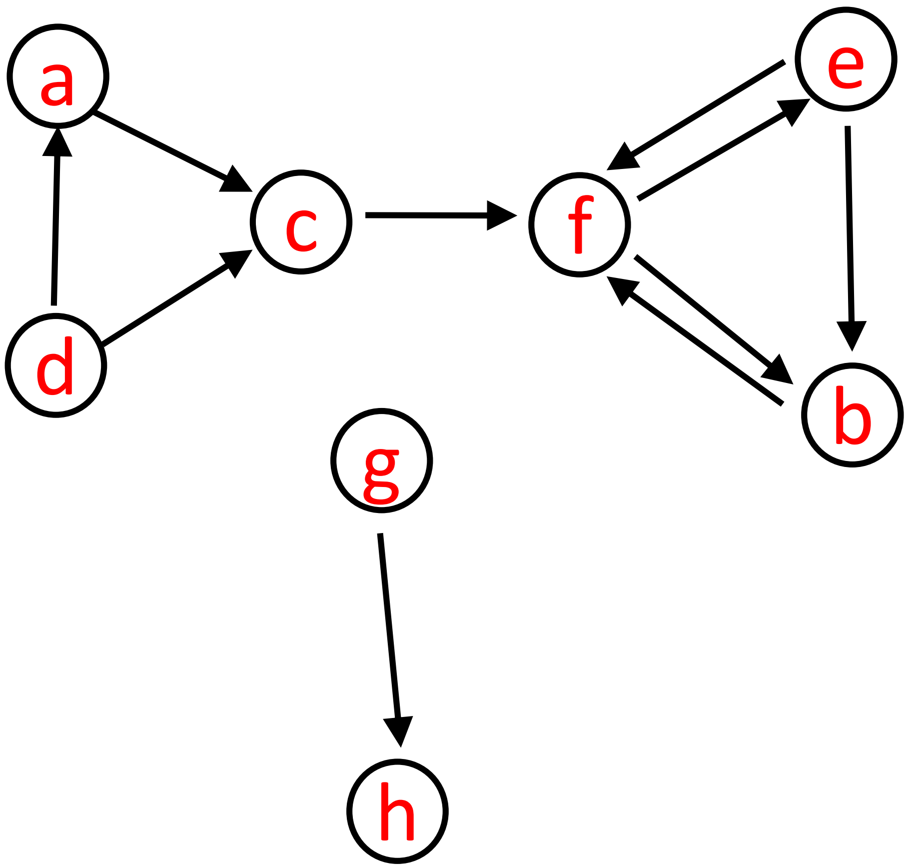

linked lists
trees
graphs

Terminology: “in degree”



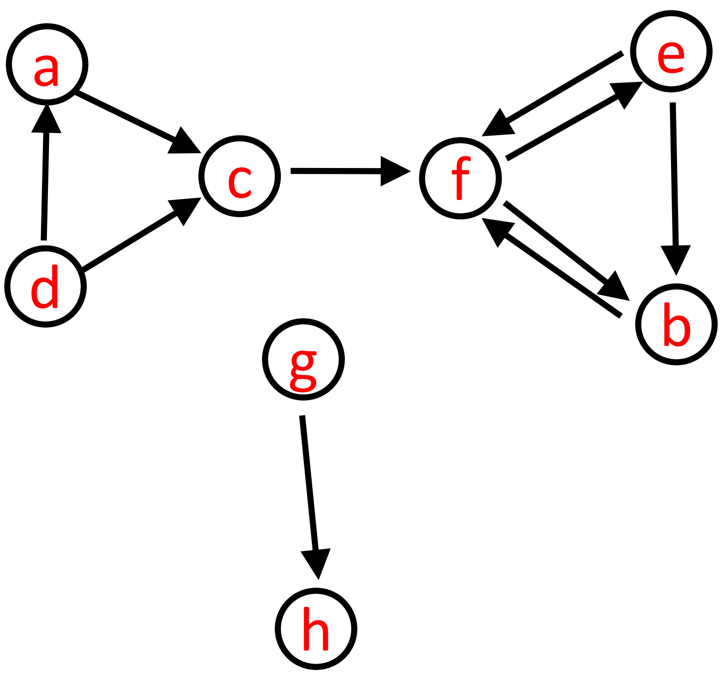
v	in degree
a	1
b	2
c	2
d	0
e	1
f	3
g	0
h	1

Terminology: “out degree”



v	out degree
a	1
b	1
c	1
d	2
e	2
f	2
g	1
h	0

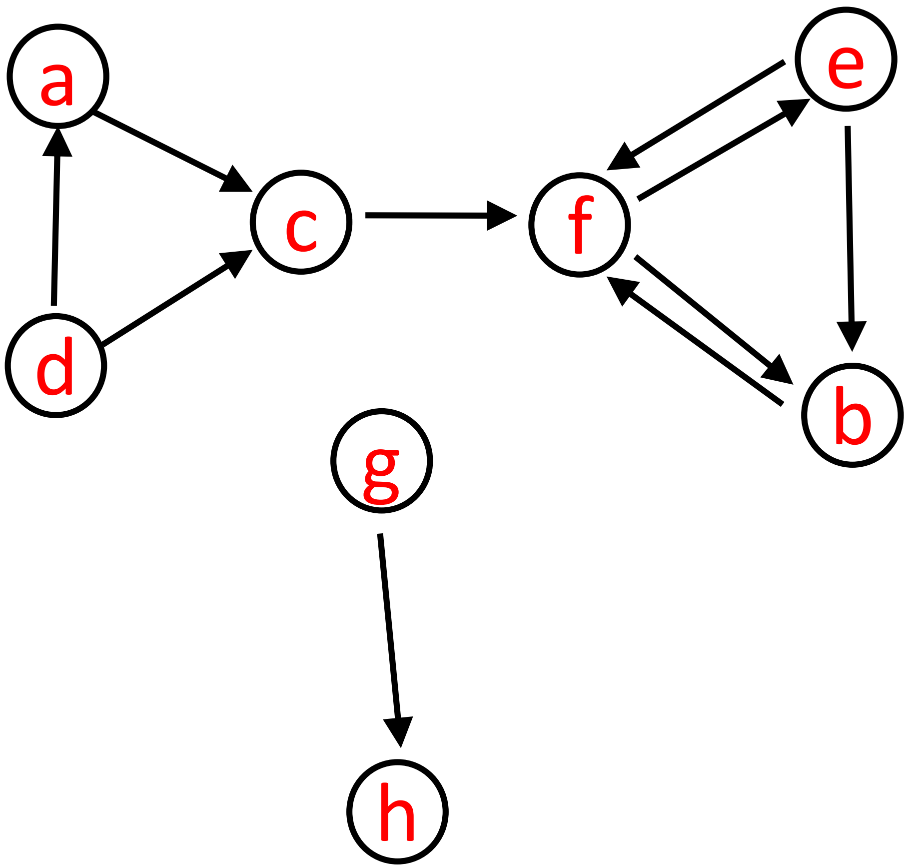
Example: web pages



In degree: How many web pages link to some web page (e.g. f) ?

Out degree: How many web pages does some web page (e.g. f) link to ?

Terminology: path



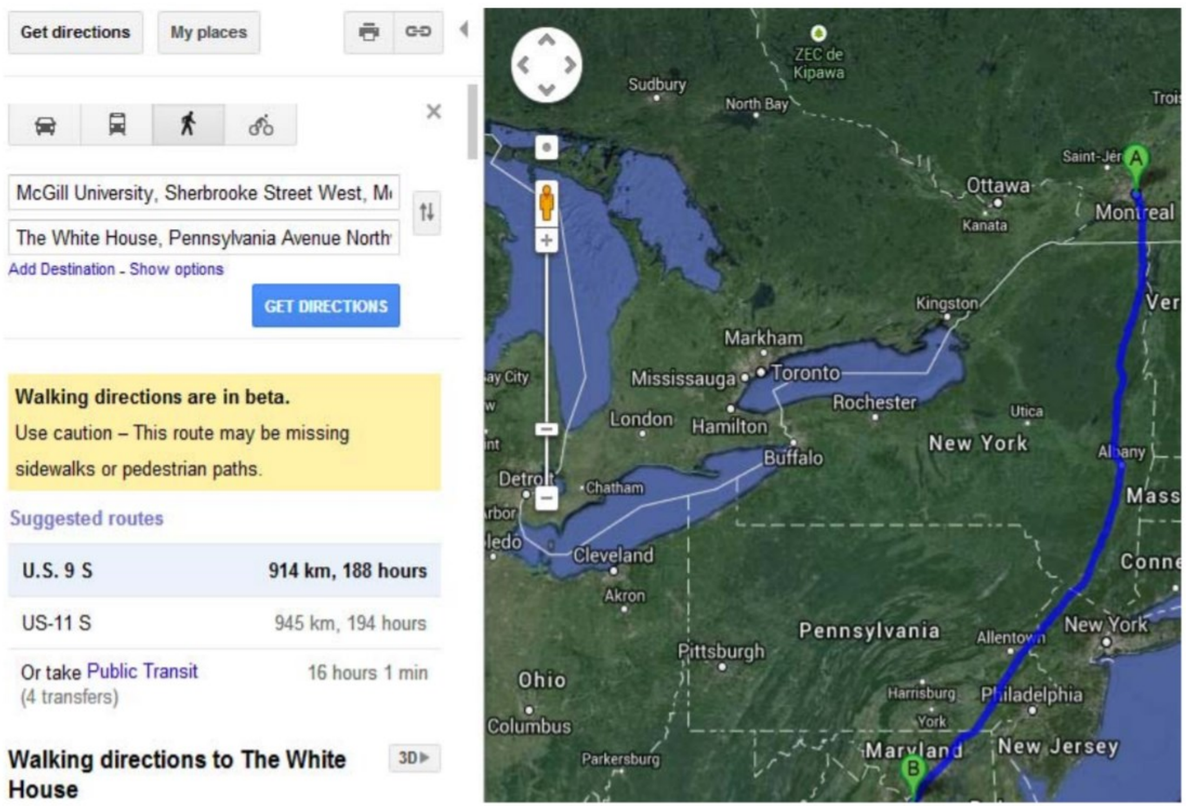
A *path* is a sequence of edges such that end vertex of one edge is the start vertex of the next edge and no vertex repeated except maybe first and last.

Examples

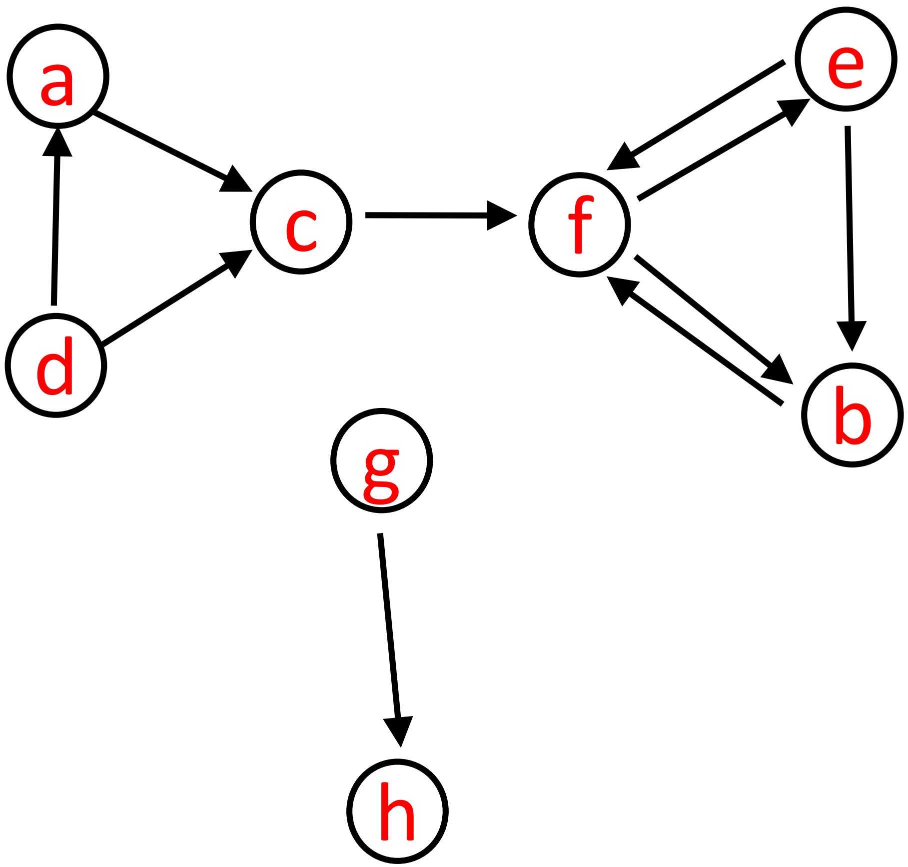
- acfeb
- dac
- febf
- .....

Graph algorithms in COMP 251

Given a graph,   what is the shortest (weighted) path between two vertices?



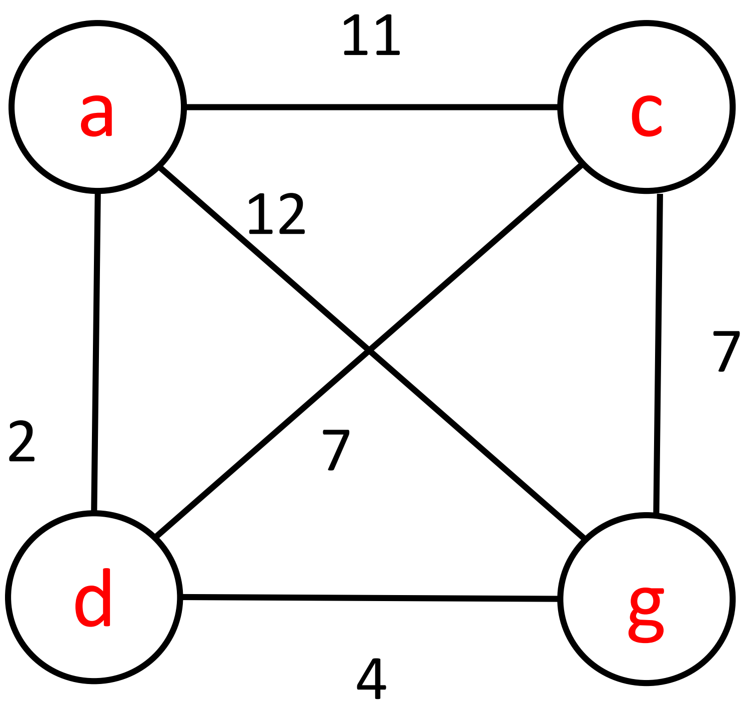
Terminology: cycle



A *cycle* is a path such that the last vertex is the same as the first vertex.

Examples
- febf
- efe
- fbf
- …

“Travelling Salesman” COMP 360
(Hamiltonian circuit)



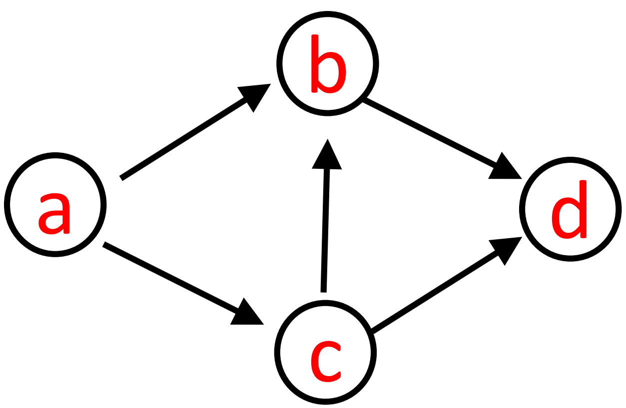
Find the shortest cycle that visits all vertices once.

How many potential cycles are there in a graph of n vertices ?

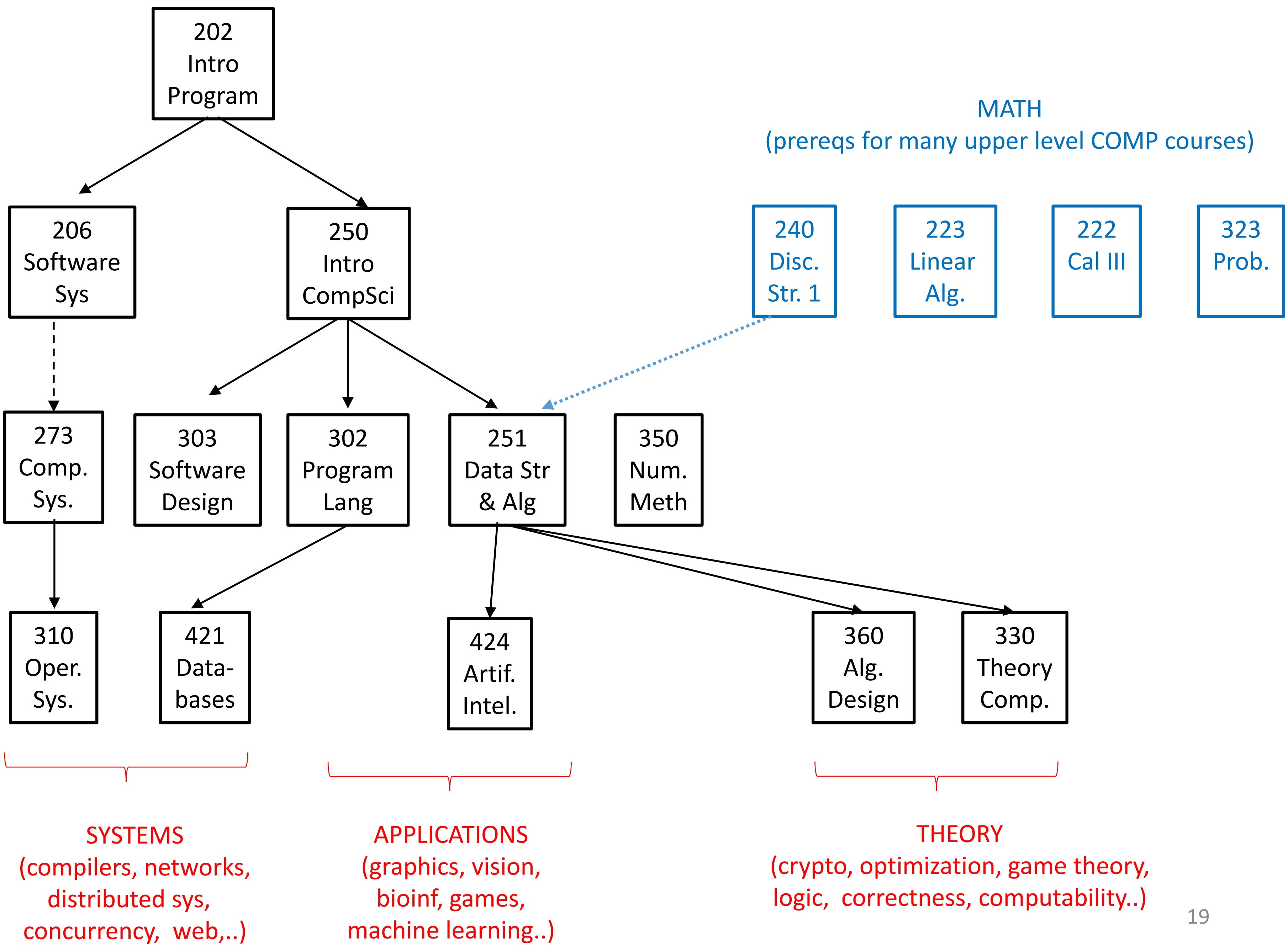
Directed *Acyclic* Graph

no cycles

Used to capture dependencies.



There are three paths from a to d.



202
Intro
Program
MATH
(prereqs for many upper level COMP courses)
206
Software
Sys
250
Intro
CompSci
240
Disc.
Str. 1
223
Linear
Alg.
222
Cal III
323
Prob.
273
Comp.
Sys.
303
Software
Design
302
Program
Lang
251
Data Str
& Alg
350
Num.
Meth
310
Oper.
Sys.
421
Data-
bases
424
Artif.
Intel.
360
Alg.
Design
330
Theory
Comp.
SYSTEMS
(compilers, networks,
distributed sys,
concurrency, web,..)
APPLICATIONS
(graphics, vision,
bioinf, games,
machine learning..)
THEORY
(crypto, optimization, game theory,
logic, correctness, computability..)

Graph ADT

- addVertex(…), addEdge(…)
- containsVertex(…), containsEdge(…)
- getVertex(…), getEdge(…)
- removeVertex(…), removeEdge(…)
- numVertices( ), numEdges( )
- …

How to implement a Graph class? A graph is a generalization of a tree, so …

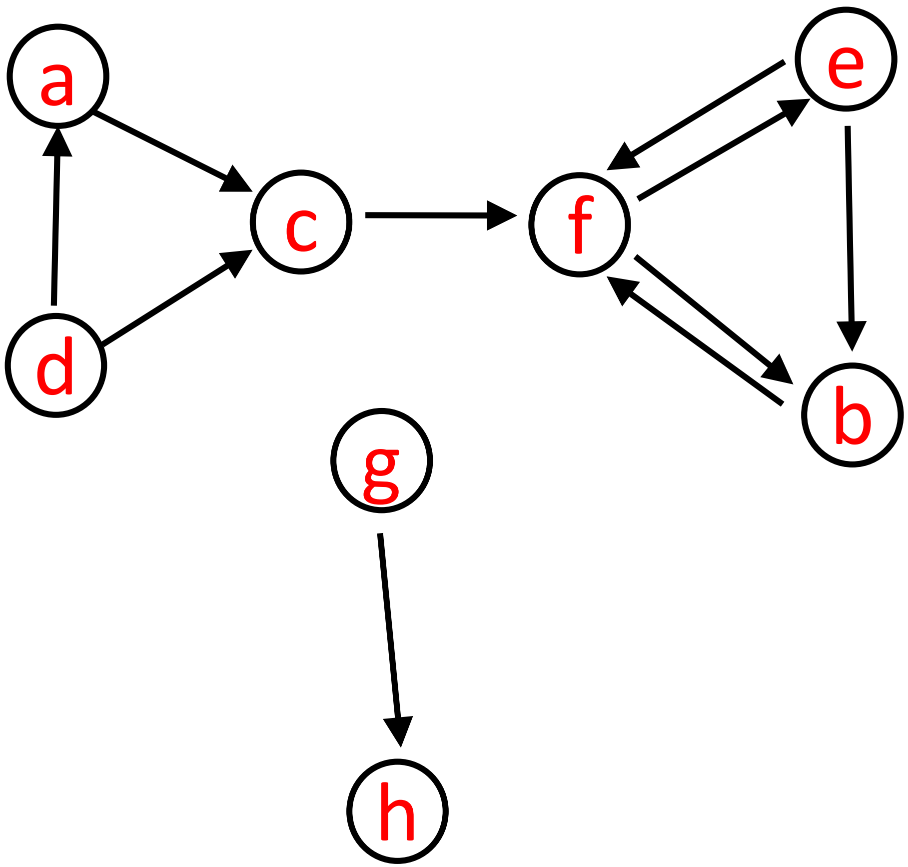
Recall: How to implement a rooted tree in Java ?

```
class  Tree<T>{
   TreeNode<T>   root;
      :

   // inner class

  class  TreeNode<T>{
     T   element;
     ArrayList< TreeNode<T> >     children;
     TreeNode<T>                  parent;
  }

}
```

```
//    alternatively….

class  TreeNode<T>{
    T   element;
    TreeNode<T>   firstChild;
    TreeNode<T>   nextSibling;
       :
  }
```

Adjacency List

(generalization of children for graphs)



v	**v.adjList**
a	c
b	f
c	f
d	a, c
e	b, f
f	b, e
g	h
h	

Here each adjacency list is sorted, but that is not always possible (or necessary).

How to implement a Graph class in Java?

```
class Graph<T>  {

    class Vertex<T>                    //   Could have called it GNode
     {
        ArrayList<Vertex>      adjList;
        T                      element;
     }

}
```

This is a very basic Graph class.

How to implement a Graph class in Java?

```
class Graph<T> {

    class Vertex<T>   {
        ArrayList<Edge>      adjList;
        T                    element;
        boolean              visited;
    }

    class Edge {
       Vertex             endVertex;
       double             weight;
              :
    }
}
```

Unlike a rooted tree, there is no notion of a root vertex in a graph.

How to reference vertices?

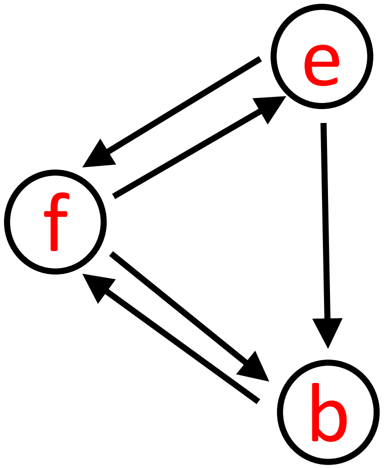
Suppose we have a string name (key) for each vertex.

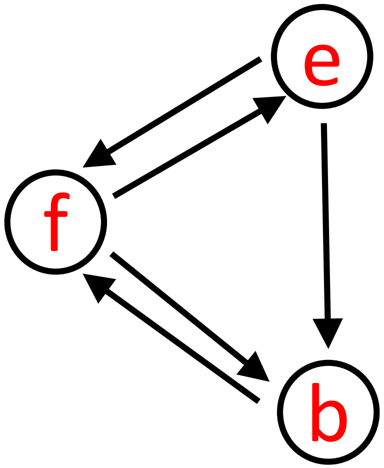
e.g. YUL for Trudeau airport, LAX for Los Angeles, …

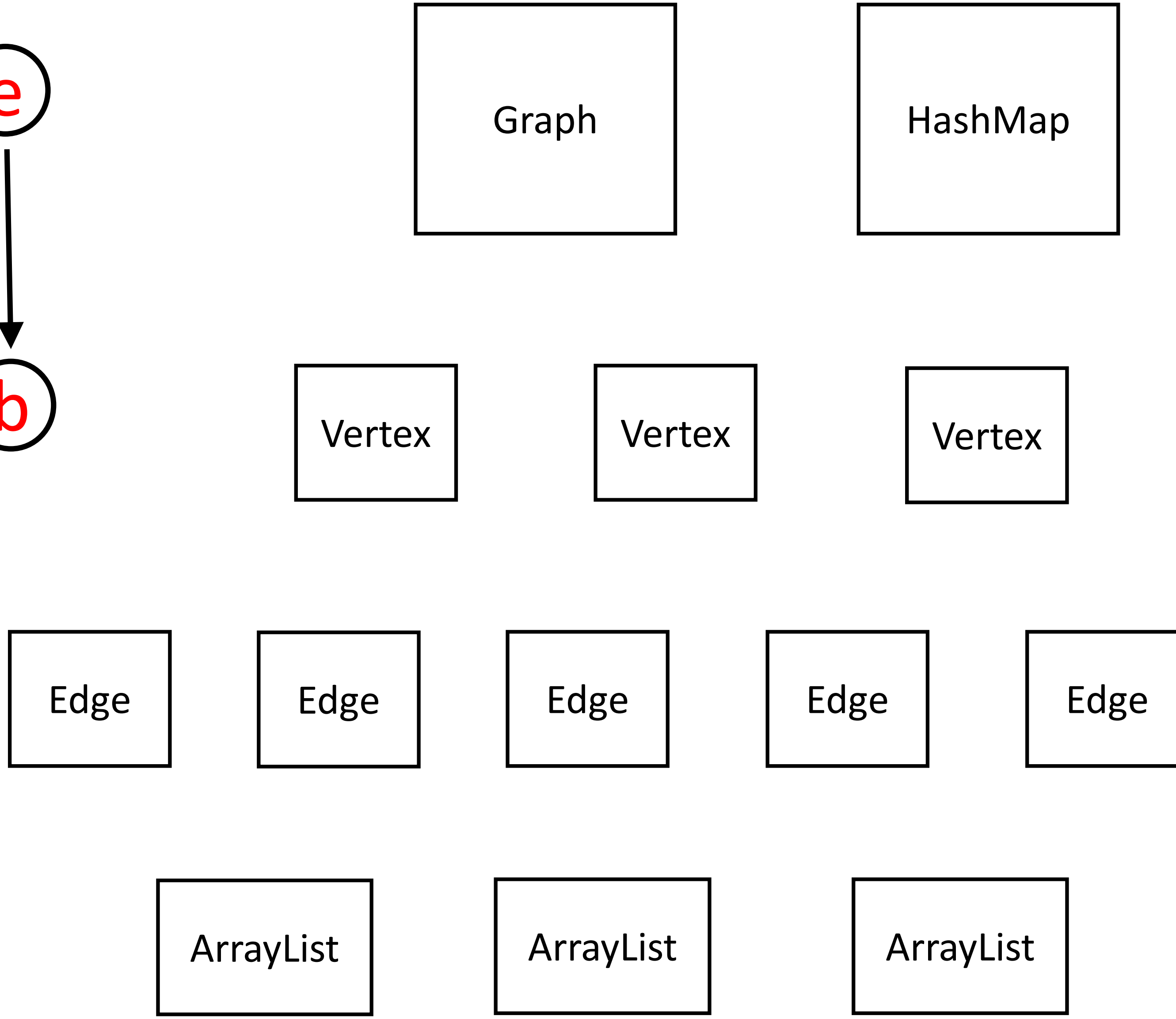
```
class  Graph<T> {
        HashMap< String, Vertex<T> >    vertexMap;
          :
       class Vertex<T> { …}
       class  Edge<T>  { …}
}
```

We could also just enumerate vertices.

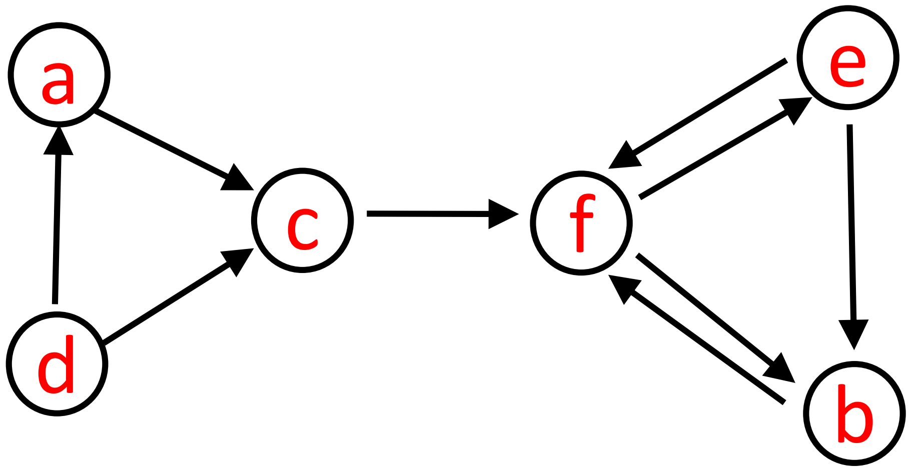
How many objects ?




e
f
b


Graph
HashMap
Vertex
Vertex
Vertex
Edge
Edge
Edge
Edge
Edge
ArrayList
ArrayList
ArrayList

Adjacency Matrix

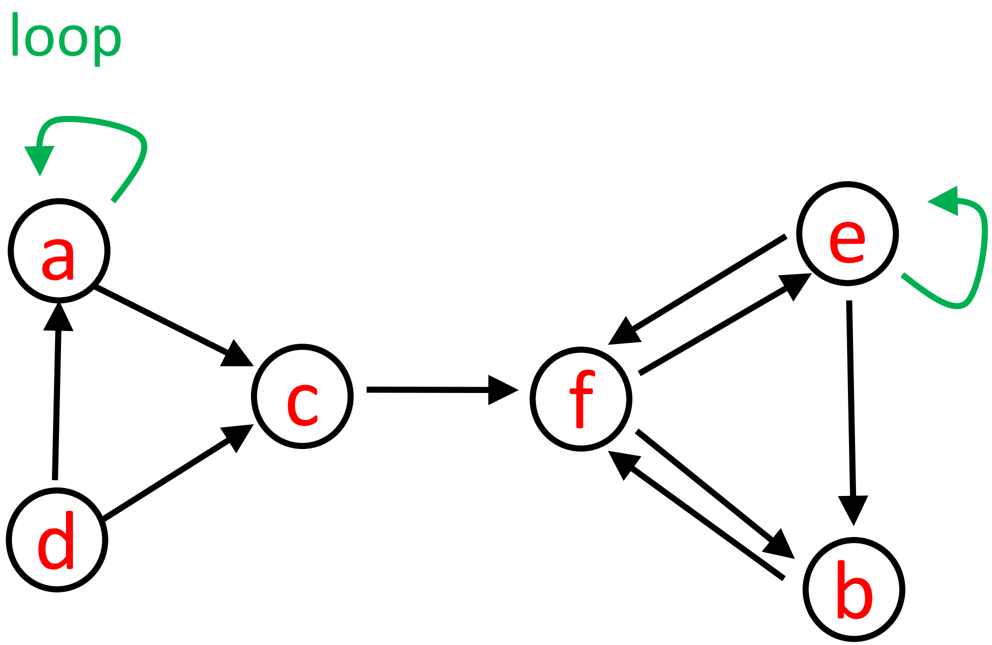


$$
\begin{array}{c|cccccc|}
 & a & b & c & d & e & f \\
\hline
a & 0 & 0 & 1 & 0 & 0 & 0 \\
b & 0 & 0 & 0 & 0 & 0 & 1 \\
c & 0 & 0 & 0 & 0 & 0 & 1 \\
d & 1 & 0 & 1 & 0 & 0 & 0 \\
e & 0 & 1 & 0 & 0 & 0 & 1 \\
f & 0 & 1 & 0 & 0 & 1 & 0 \\
\hline
\end{array}
$$

Assume we have a mapping from vertex names to 0, 1, …. , n-1.

boolean adjMatrix[6][6]

Adjacency Matrix



	a	b	c	d	e	f
a	1	0	1	0	0	0
b	0	0	0	0	0	1
c	0	0	0	0	0	1
d	1	0	1	0	0	0
e	0	1	0	0	1	1
f	0	1	0	0	1	0

boolean   adjMatrix[6][6]

Suppose a graph has n vertices.

The graph is *dense* if number of edges is close to n^2.

The graph is *sparse* if number of edges is close to n.

(These are not formal definitions.)

Exercise

Would you use an *adjacency list* or *adjacency matrix* for each of the following?

- The graph is sparse e.g. 10,000 vertices and 20,000 edges and we want to use as little space as possible.

Exercise

Would you use an *adjacency list* or *adjacency matrix* for each of the following?

- The graph is sparse e.g. 10,000 vertices and 20,000 edges and we want to use as little space as possible.
- The graph is dense e.g. 10,000 vertices and 20,000,000 edges, and we want to use as little space as possible.

Exercise

Would you use an *adjacency list* or *adjacency matrix* for each of the following?

- The graph is sparse e.g. 10,000 vertices and 20,000 edges and we want to use as little space as possible.
- The graph is dense e.g. 10,000 vertices and 20,000,000 edges, and we want to use as little space as possible. .
- Answer the query areAdjacent() as quickly as possible, no matter how much space you use.

Exercise

Would you use an *adjacency list* or *adjacency matrix* for each of the following?

- The graph is sparse e.g. 10,000 vertices and 20,000 edges and we want to use as little space as possible.
- The graph is dense e.g. 10,000 vertices and 20,000,000 edges, and we want to use as little space as possible. .
- Answer the query areAdjacent() as quickly as possible, no matter how much space you use.
- Perform operation insertVertex(v).

Exercise

Would you use an *adjacency list* or *adjacency matrix* for each of the following?

- The graph is sparse e.g. 10,000 vertices and 20,000 edges and we want to use as little space as possible.
- The graph is dense e.g. 10,000 vertices and 20,000,000 edges, and we want to use as little space as possible. .
- Answer the query areAdjacent() as quickly as possible, no matter how much space you use.
- Perform operation insertVertex(v).
- Perform operation removeVertex(v).

Next lecture

- Recursive graph traversal
  - depth first
- Non-recursive graph traversal
  - depth first
  - breadth first