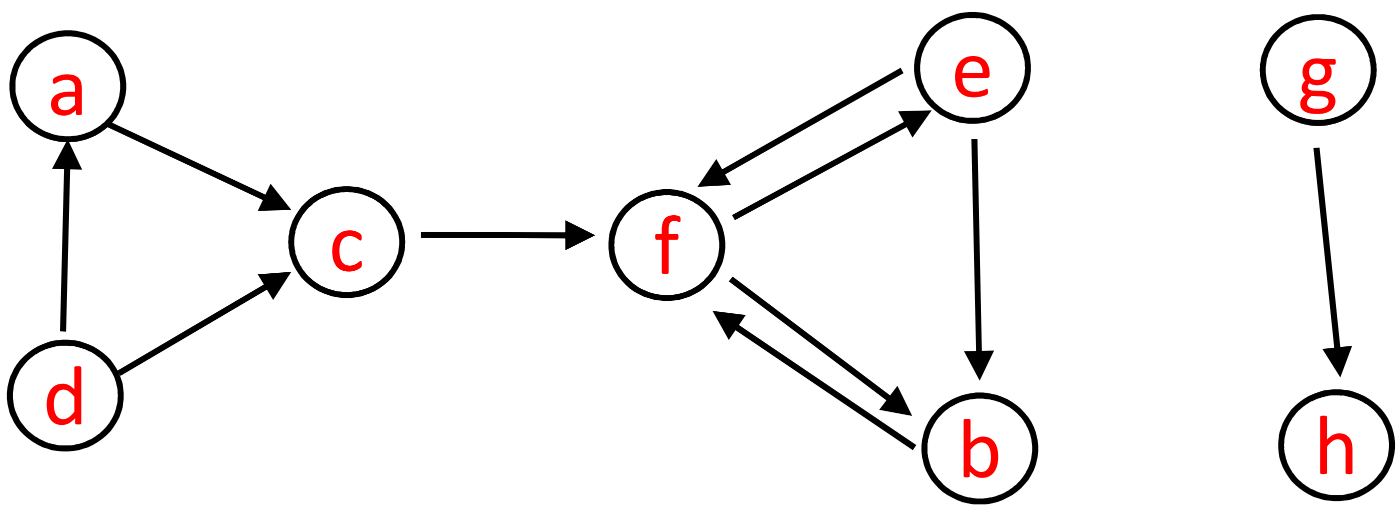
COMP 250

Lecture 31

graphs 1

March 25, 2022

Example



Definition

A *directed graph* is a set of *vertices (or "nodes")*

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

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

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

Examples (Directed)

Vertices	Edges
airports	flights
web pages	links (URLs)
Java objects	references

Definition

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

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

and set of unordered pairs, again called *edges*.

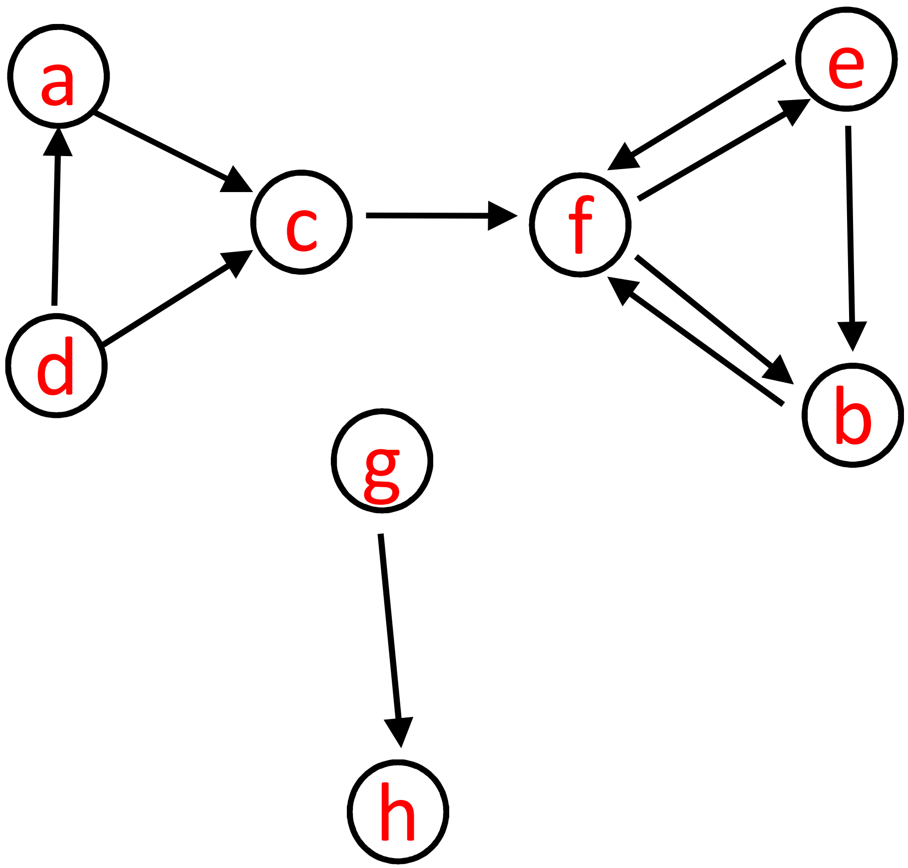
$$E = \{\, \{v_i, v_j\} : \; i, j \in 1, \dots, n\}$$

Examples (Undirected)

<u>Vertices</u>	<u>Edges</u>
people	friends
events	conflicts (edge if two events cannot be at same time)
towns/cities	roads (two way)

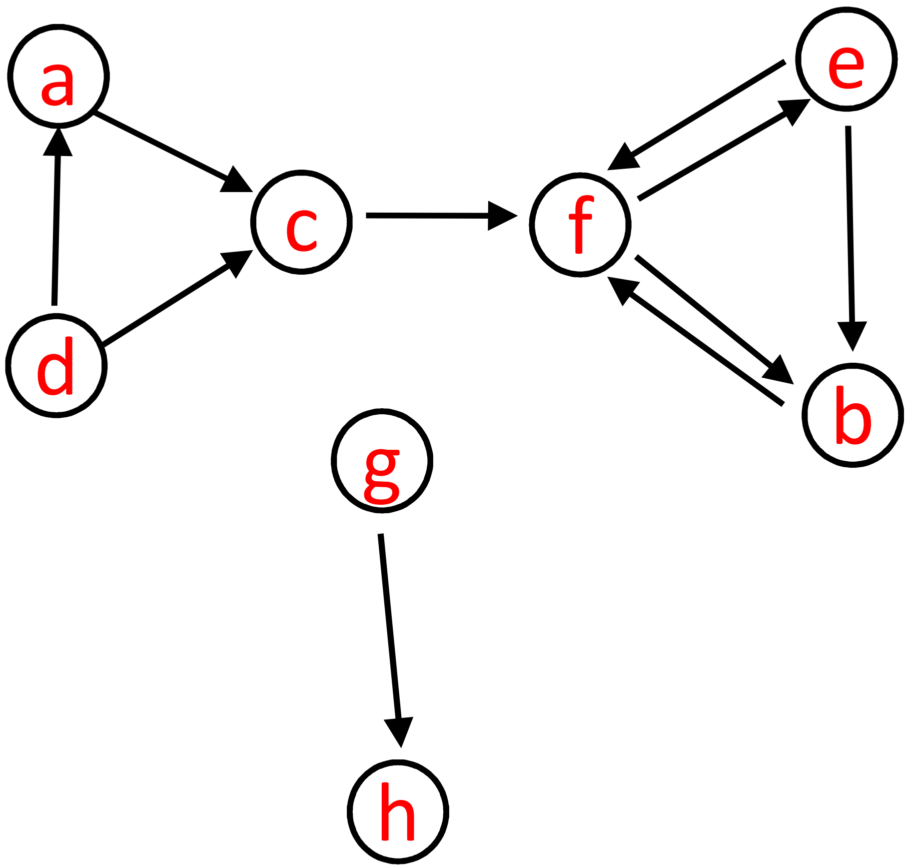
We will mostly discuss directed graphs.

Terminology: “in degree”



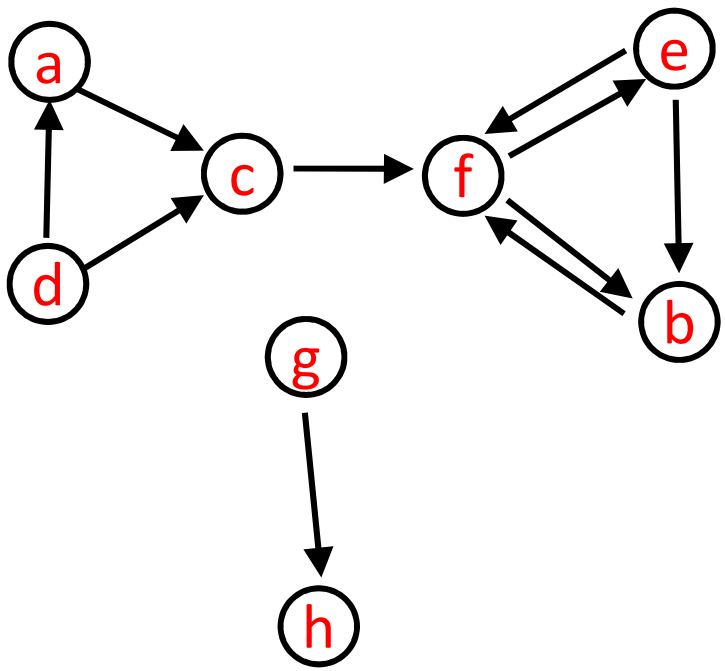
v	**in degree**
a	1
b	2
c	2
d	0
e	1
f	3
g	0
h	1

Terminology: “out degree”



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

Example: www pages



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

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

Google "crawls" the web graph, retrieving and storing as many web pages as it can.

Google updates its web graph:

- the vertices V are the web pages
- the edges E are the hyperlink (references) within the web pages

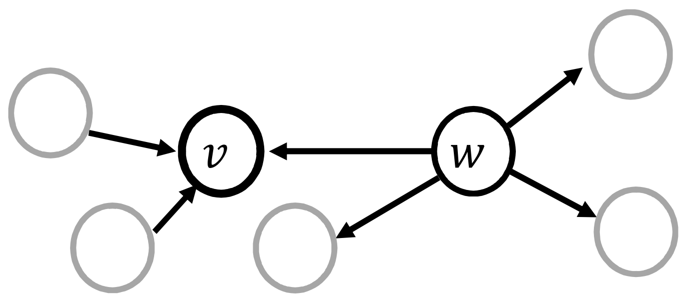
ASIDE: Google also updates a hashmap:

- the keys K are the URL's, and the values are the web pages

ASIDE: Google PageRank

Google tries to find *important* web pages for your search term.

Q: How important is a web page v ?



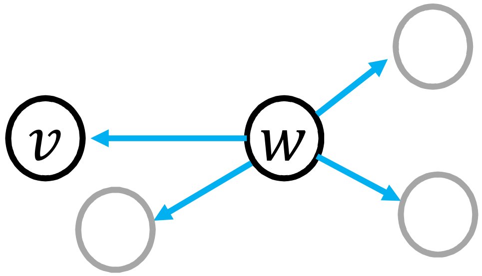
A:

- Which set of pages $\{ w \}$ link to v and how important is each page w (recursive definition!) ?
- How many other pages does each w point to ?

To define the “page rank” of v :

Let w be a vertex such that

(w, v) is an edge.

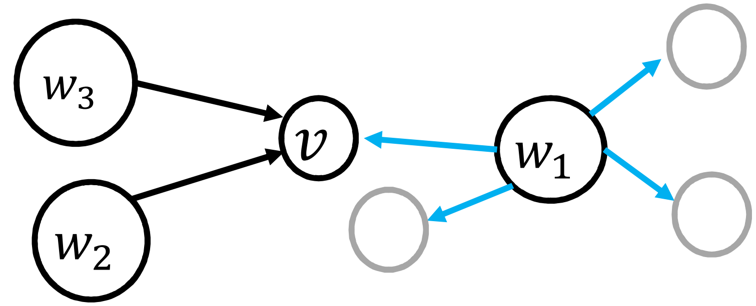


Let $N_{out}(w)$ be the out-degree of *w*.

To define the "page rank" of v :

Let w be a vertex such that

(w, v) is an edge.



Let $N_{out}(w)$ be the out-degree of w.

Define the PageRank of v:

$$R(v) = \sum_{\substack{incoming\ edges \\ (w,v)\ to\ v}} \frac{R(w)}{N_{out}(w)}$$

ASIDE: To calculate this, (1) we need a list of the incoming edges to each vertex, similar to an adjacency list but now we list the incoming instead of outgoing edges, and (2) we compute $R(v)$ for all v, and then plug the result back into the right side, and iterate. We initialize all $R(v)$ to 1.

Not secure | infolab.stanford.edu/~sergey/

Sergey Brin's Home Page

Ph.D. student in Computer Science at Stanford - sergey@cs.stanford.edu

Research

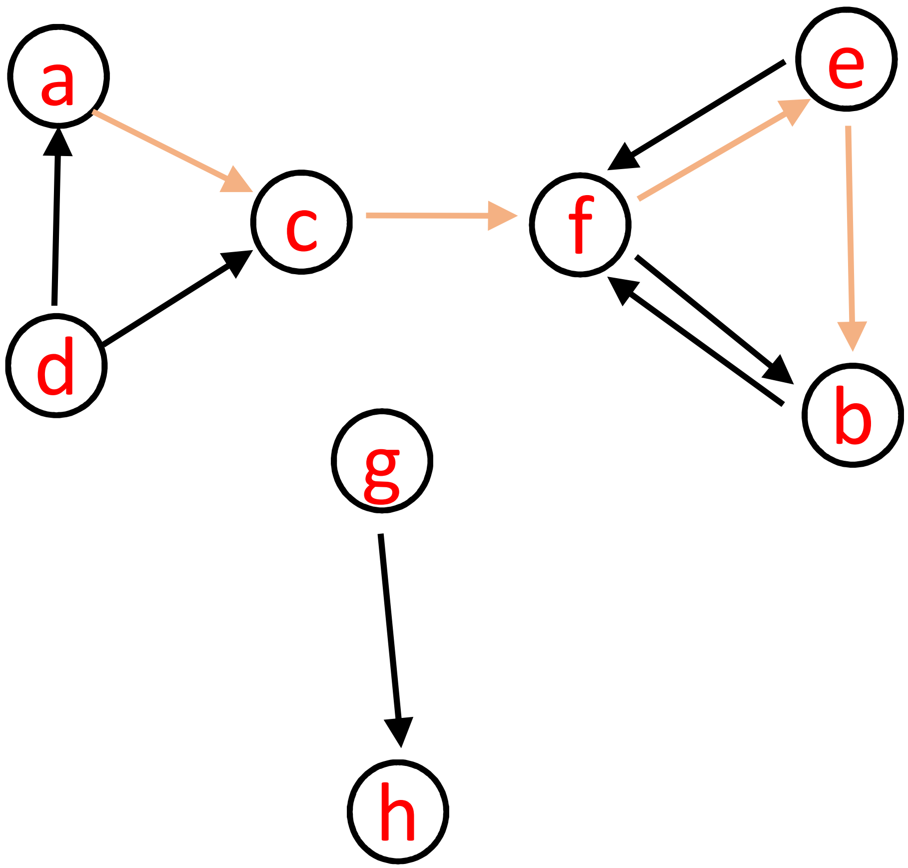
Currently I am at Google.

In fall '98 I taught CS 349.

Data Mining

A major research interest is data mining and I run a meeting group here at Stanford. For more information take a look at the MIDAS home page or see the datamine maling list achive. Here are some recent publications:

Terminology: path

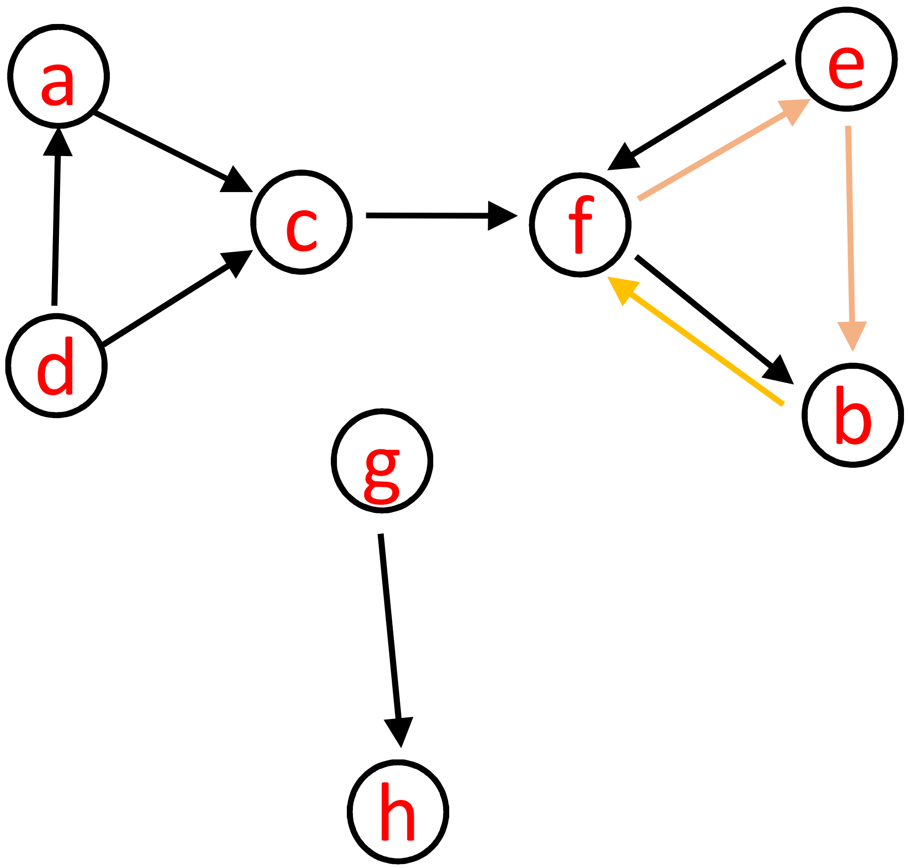


A *path* is a sequence of edges such that the end vertex of one edge is the start vertex of the next edge. No vertex may be repeated except first and last.

Examples

- acfeb
- dac
- dcfeb
- .....

Terminology: cycle



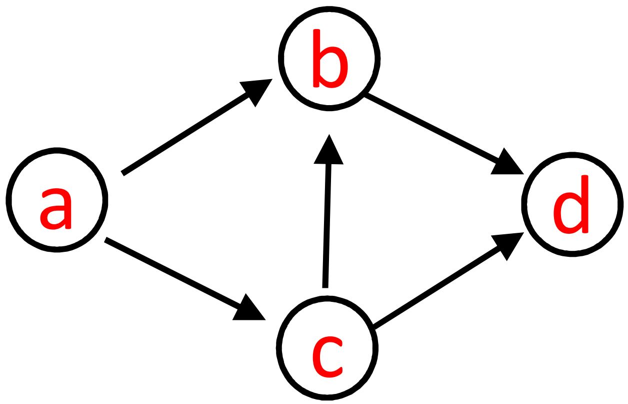
A *cycle* is a path such that the last vertex is the same as the first vertex.

Examples

- febf
- efe
- fbf
- …

Directed *Acyclic* Graph

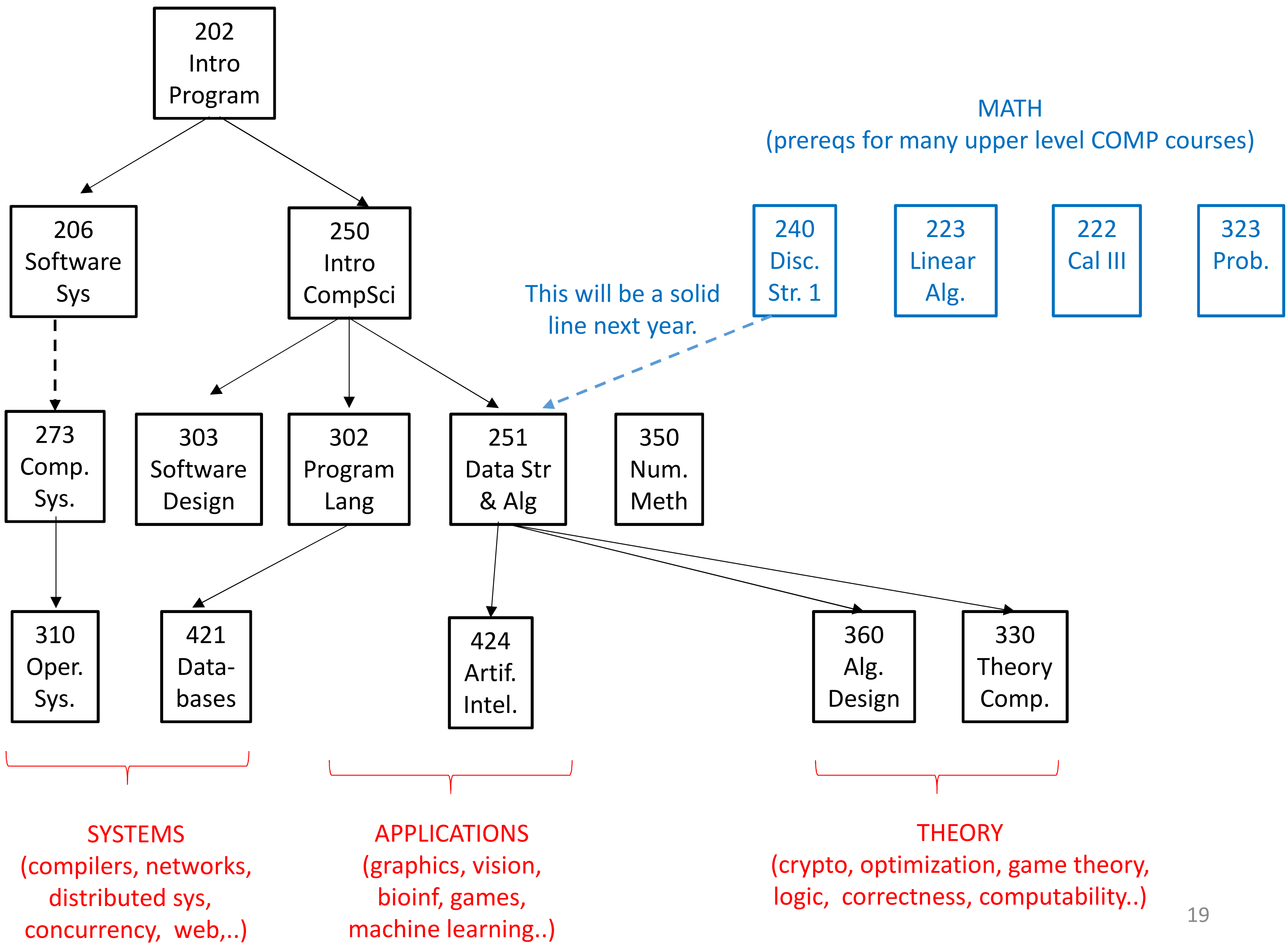
(directed graph with no cycles)



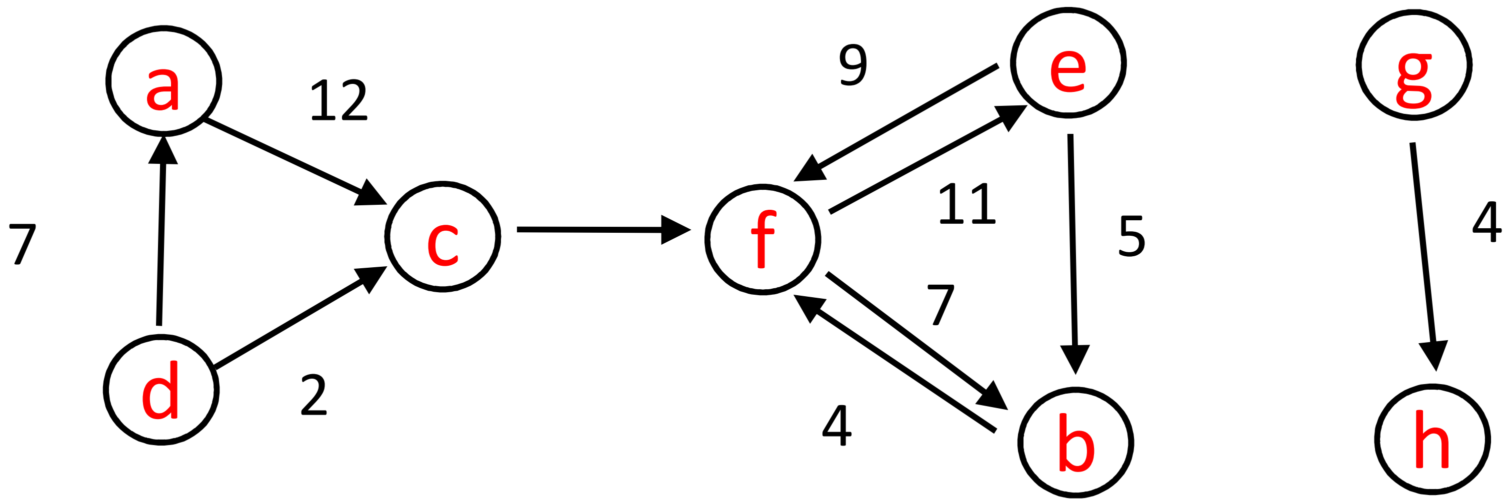
There are three paths from a to d, but no cycles.

DAGs are used to capture dependencies. e.g.

- a *causes or implies* b
- a must happen *before* b can happen (temporally)
- .....

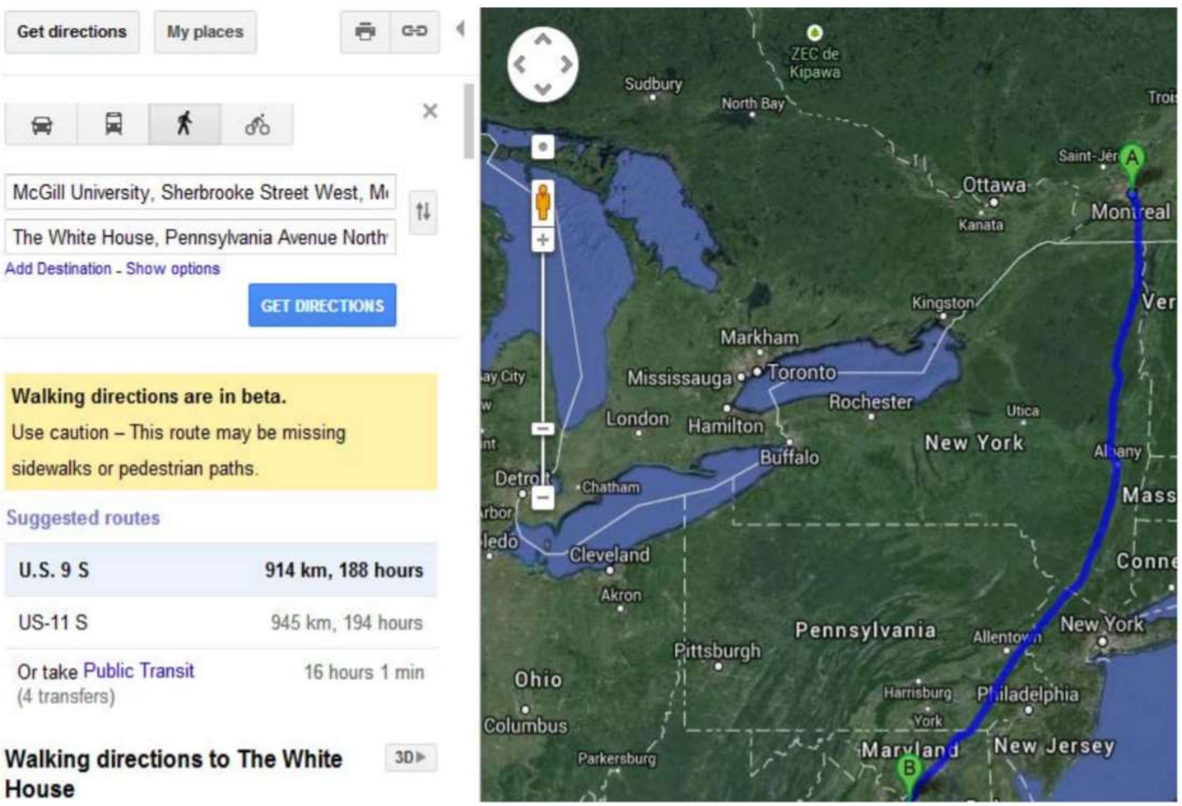

202
Intro
Program
206
Software
Sys
250
Intro
CompSci
MATH
(prereqs for many upper level COMP courses)
240
Disc.
Str. 1
223
Linear
Alg.
222
Cal III
323
Prob.
This will be a solid
line next year.
273
Comp.
Sys.
303
Software
Design
302
Program
Lang
251
Data Str
& Alg
350
Num.
Meth
310
Oper.
Sys.
421
Data-
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..)

Weighted Graph

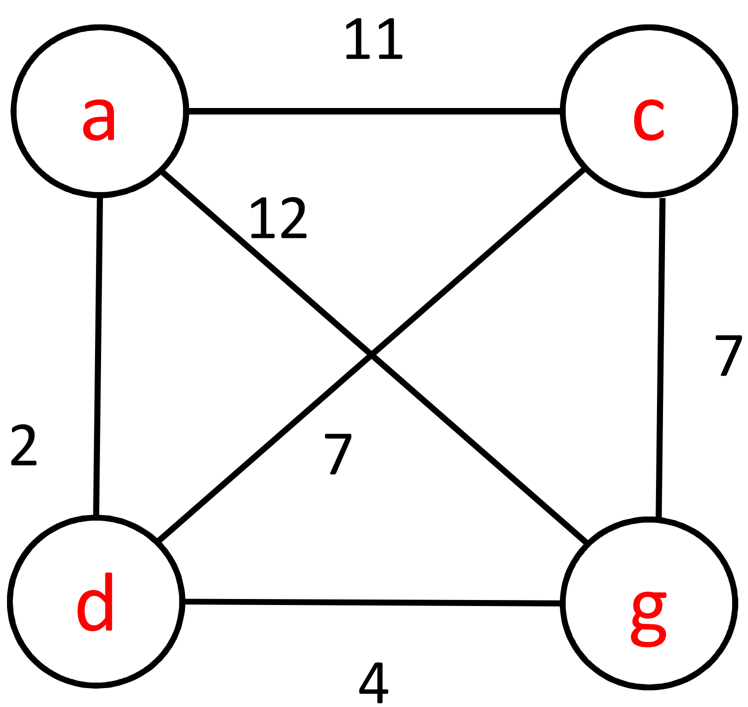


ASIDE: Shortest path algorithms
(COMP 251)

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



ASIDE: "Travelling Salesman"
(COMP 360)



Find the minimum weight cycle *that visits all vertices once.* (except first & last).

This is a hard problem (called "NP complete").

With n vertices and edges between each pair, time complexity is $O(n^2\, 2^n)$ which is very slow.

Graph ADT

- addVertex(), addEdge()
- removeVertex(), removeEdge()
- getVertex(), getEdge()

- containsVertex(), containsEdge()
- numVertices(), numEdges()
- …

How to implement a Graph class?

- Graphs are a generalization of trees, but a graph does not have a root vertex.

- Outgoing edges from a vertex in a graph are like children of a vertex in a tree.

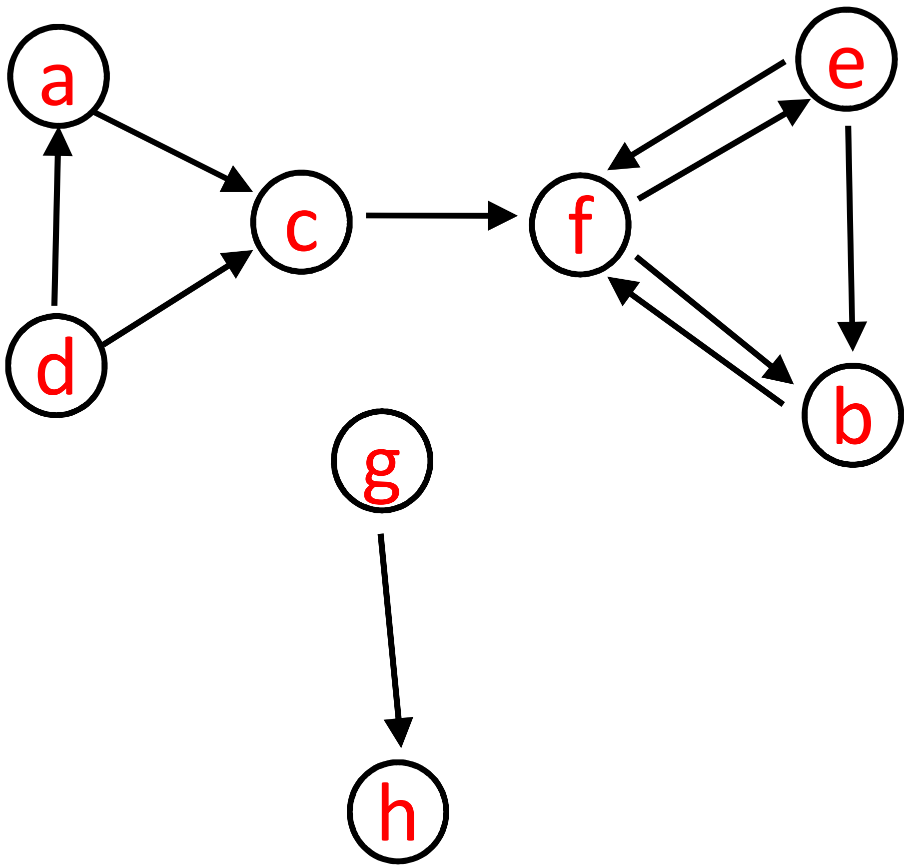
- Incoming edges are like parent(s).

There are two standard ways of representing edges

(next few slides).

1. Adjacency List

(generalization of children in trees)



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/meaningful (or necessary).

How to implement a graph with adjacency lists in Java? (1)

```
class Graph<T>  {
     :
    class Vertex<T>    {                                  //    could have called it GNode
        ArrayList<Vertex>      adjList;        //  end vertex of edge  (start is 'this' vertex)

        T          element;
     }
    :
}
```

Q: What if you want the edges to have weights?

How to implement a graph with adjacency lists in Java? (2)

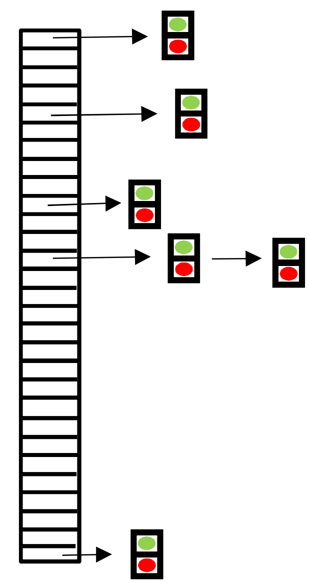
```
class Graph<T>  {                    //   this would be a weighted graph

    class Vertex<T>   {
        ArrayList<Edge>      adjList;    //    end vertex of an edge  (start is 'this' vertex)
        T        element;
    }

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

Q: How to access vertices?

Q: How to access vertices?

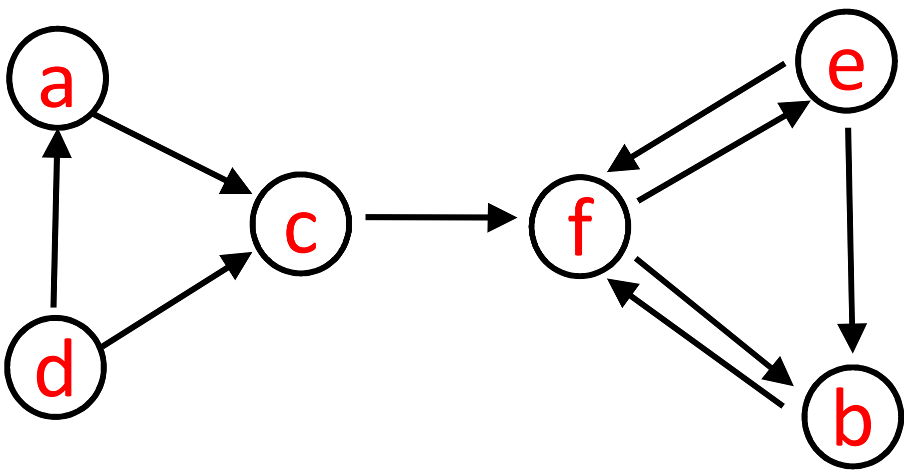
A: Use a HashMap. The key could be a string name for each vertex,
e.g. "YUL" for Trudeau airport, "LAX" for Los Angeles, etc.

```
class Graph<T> {

   HashMap< String, Vertex<T> >   vertexMap;

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

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



HashMap's have methods
like getKeys(), getValues(), ...

2. Adjacency Matrix



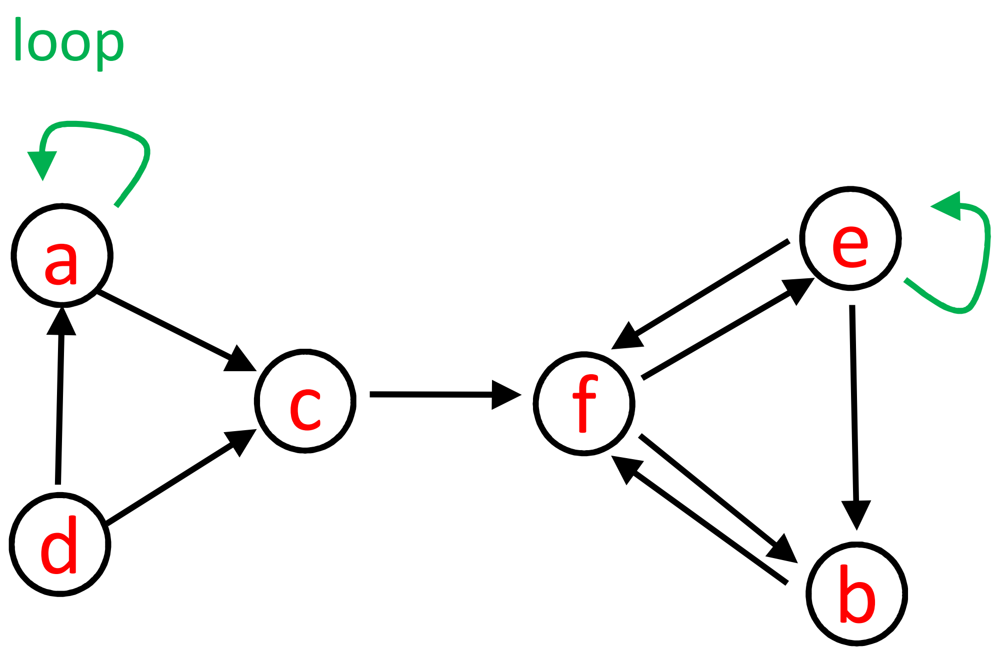
	a	b	c	d	e	f
a	0	0	1	0	0	0
b	0	0	0	0	0	1
c	0	0	0	0	0	1
d	1	0	1	0	0	0
e	0	1	0	0	0	1
f	0	1	0	0	1	0

Note:

- We need a hashmap from vertex names to indices 0, 1, …. , n-1.

boolean adjMatrix[6][6]

2. Adjacency Matrix



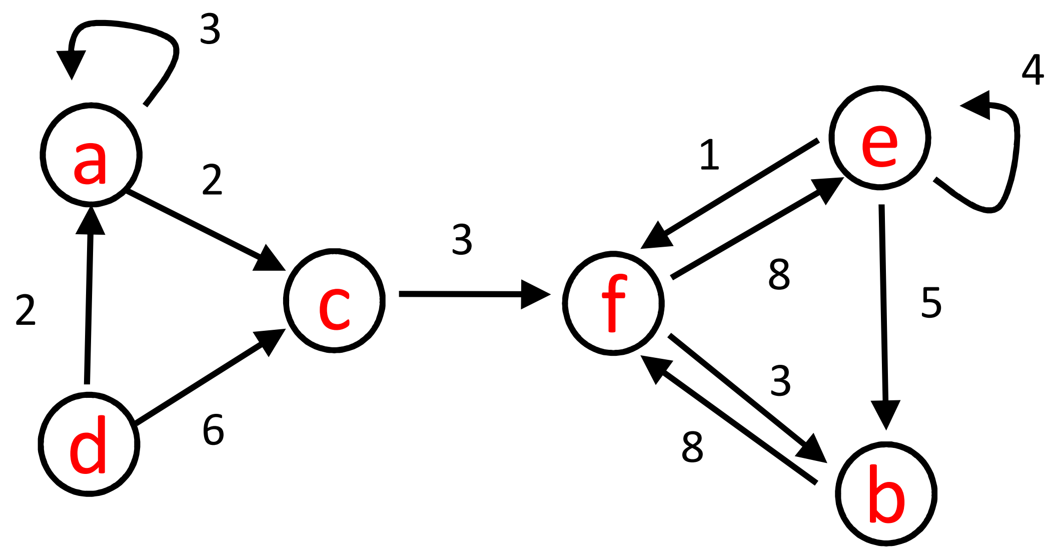
	a	b	c	d	e	f
a	**1**	0	1	0	0	0
b	0	**0**	0	0	0	1
c	0	0	**0**	0	0	1
d	1	0	1	**0**	0	0
e	0	1	0	0	**1**	1
f	0	1	0	0	1	**0**

Note:

- Use the diagonal elements for loop edges.

boolean adjMatrix[6][6]

2. Adjacency Matrix



$$
\begin{array}{c|cccccc|}
 & a & b & c & d & e & f \\
\hline
a & \mathbf{3} & 0 & 2 & 0 & 0 & 0 \\
b & 0 & \mathbf{0} & 0 & 0 & 0 & 8 \\
c & 0 & 0 & \mathbf{0} & 0 & 0 & 3 \\
d & 2 & 0 & 6 & \mathbf{0} & 0 & 0 \\
e & 0 & 5 & 0 & 0 & \mathbf{4} & 1 \\
f & 0 & 3 & 0 & 0 & 8 & \mathbf{0} \\
\hline
\end{array}
$$

int adjMatrix[6][6]

Note:

- For a weighted graph, we could use weights in the matrix instead of booleans.

See Exercises for when you would use
adjacency list versus *adjacency matrix*.

Hint: it depends on how many edges we have relative to number of vertices.

Coming up...

Lectures

Mon. March 28

Graphs traversal

Wed & Fri, March 30 & April 1

recurrences

Mon, Wed, Fri : April 4, 6, 8

big O, ...

Assessments

Quiz 5 is in Mon. April 4

Assignment 4 due Wed. April 6.