COMP 250

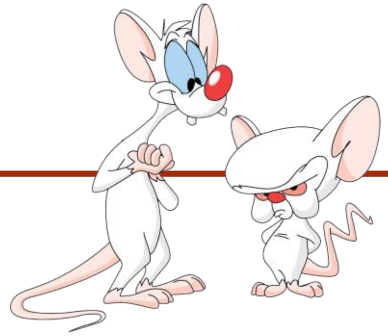
INTRODUCTION TO COMPUTER SCIENCE

Week 13-3 : Graphs

Giulia Alberini, Fall 2020

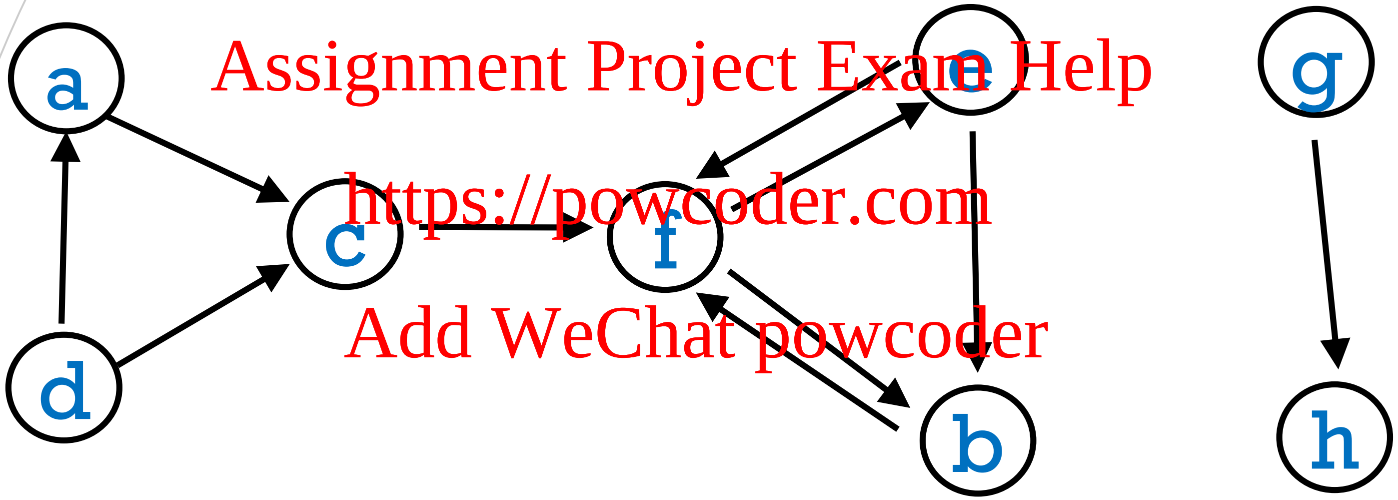
Slides adapted from Michael Langer's

WHAT ARE WE GOING TO DO IN THIS VIDEO?

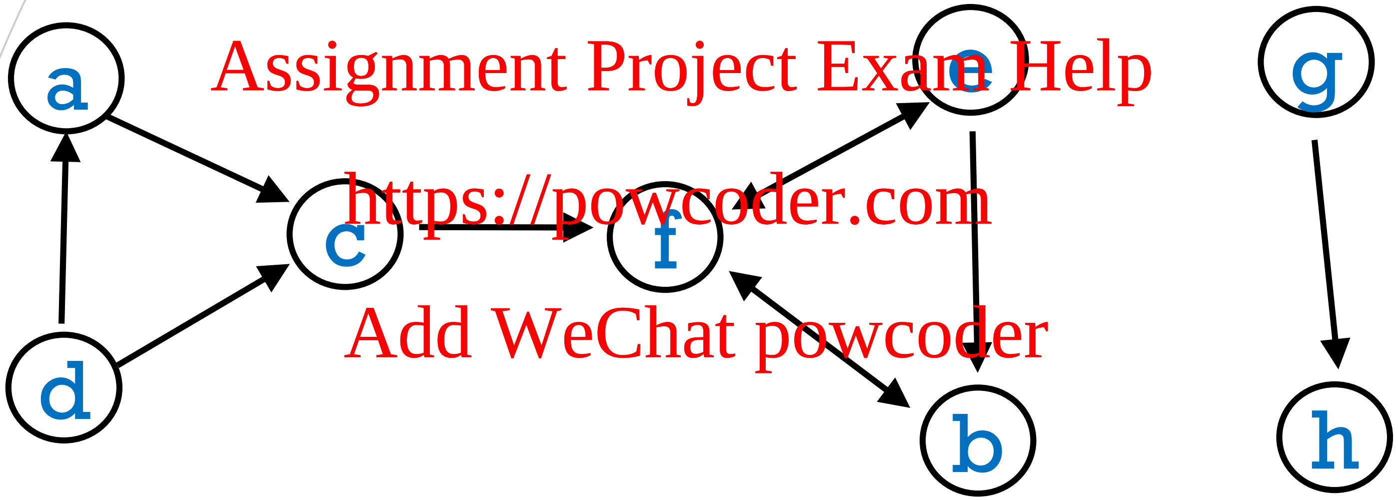


- Graphs
  - Definitions

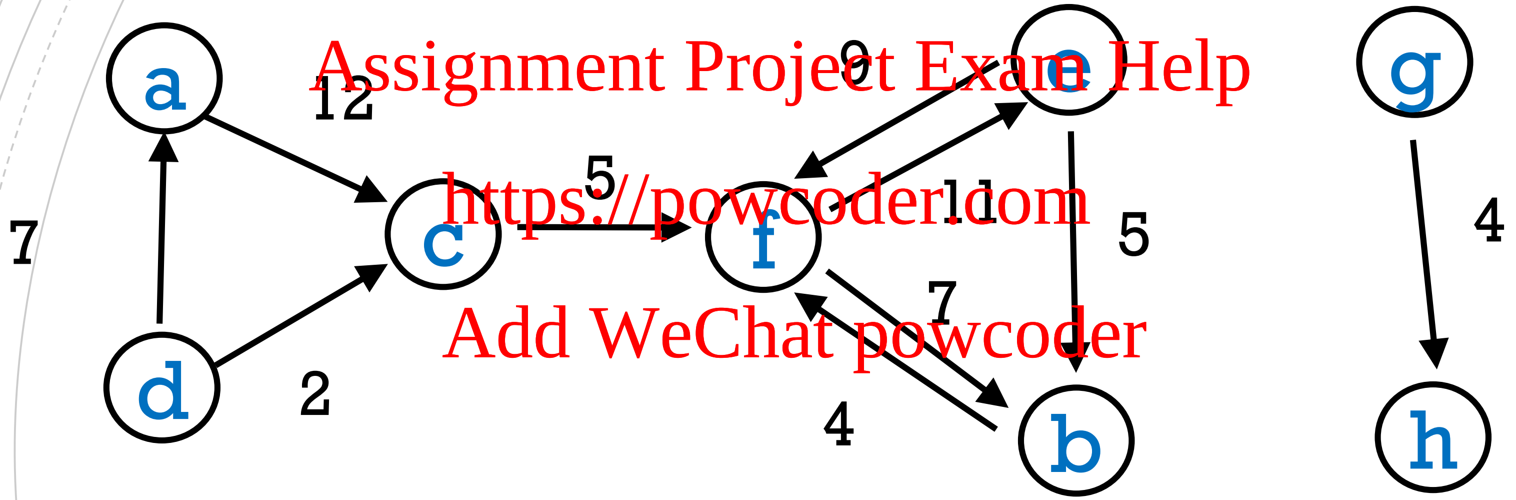
EXAMPLE



SAME EXAMPLE – DIFFERENT NOTATION



WEIGHTED GRAPH



DEFINITION

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

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

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

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

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

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

EXAMPLES

Vertices | Edges

airports

web pages

Java objects

EXAMPLES

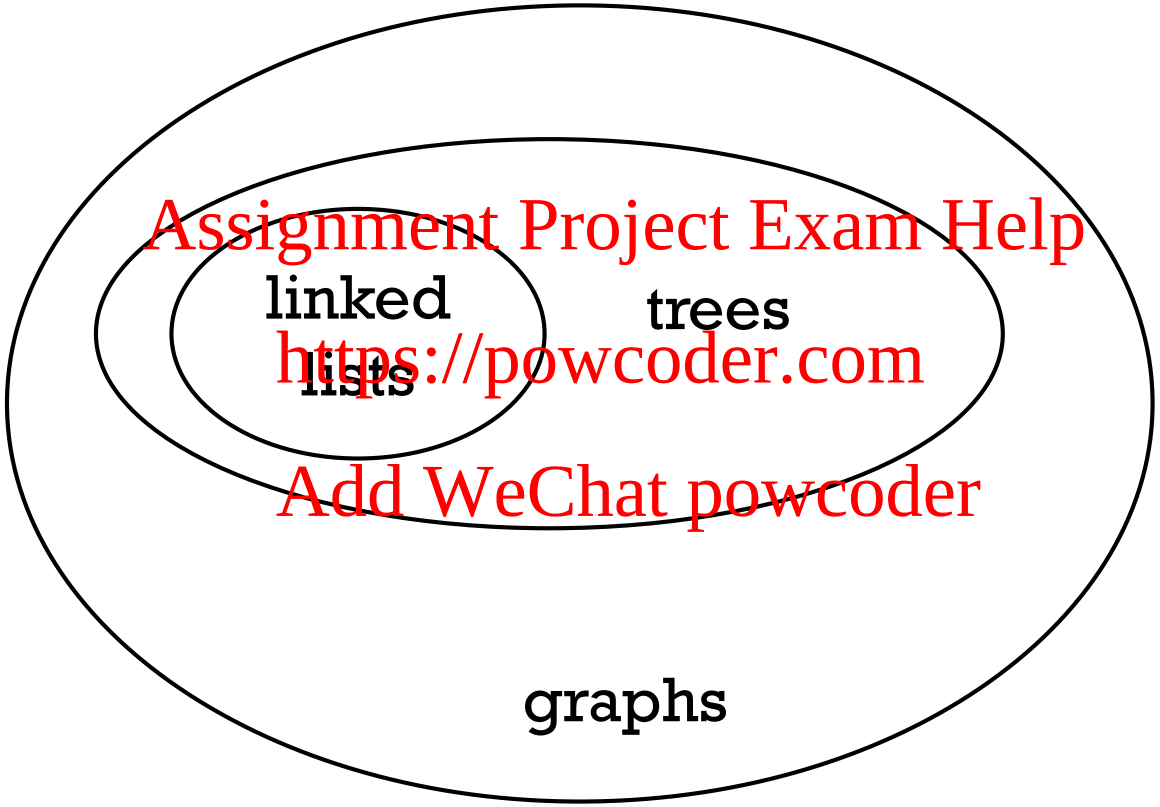
Vertices	Edges
airports	flights
web pages	
Java objects	

EXAMPLES

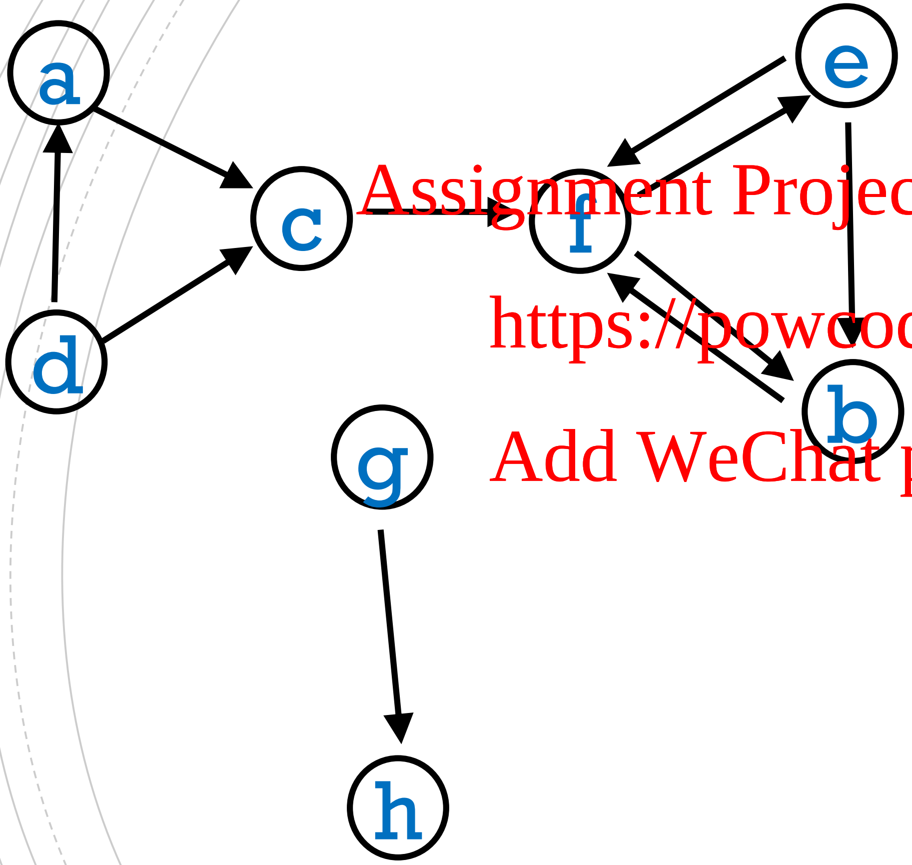
Vertices	Edges
airports	flights
web pages	links (URLs)
Java objects	

EXAMPLES

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

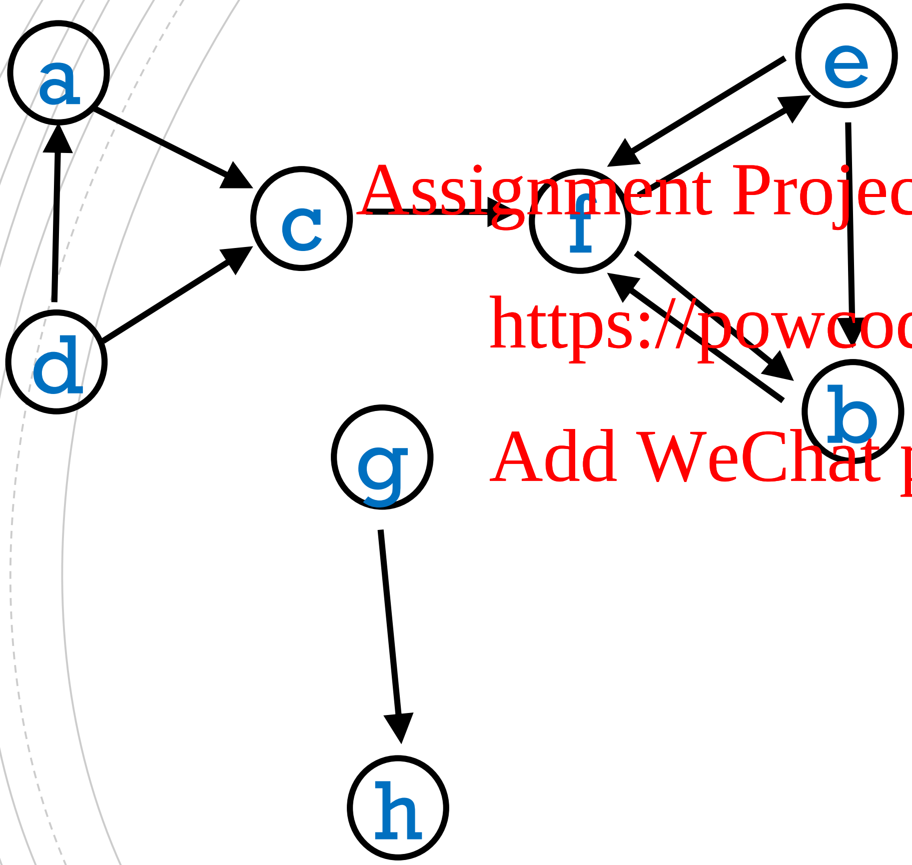

linked
lists
trees
graphs

TERMINOLOGY: "IN DEGREE"



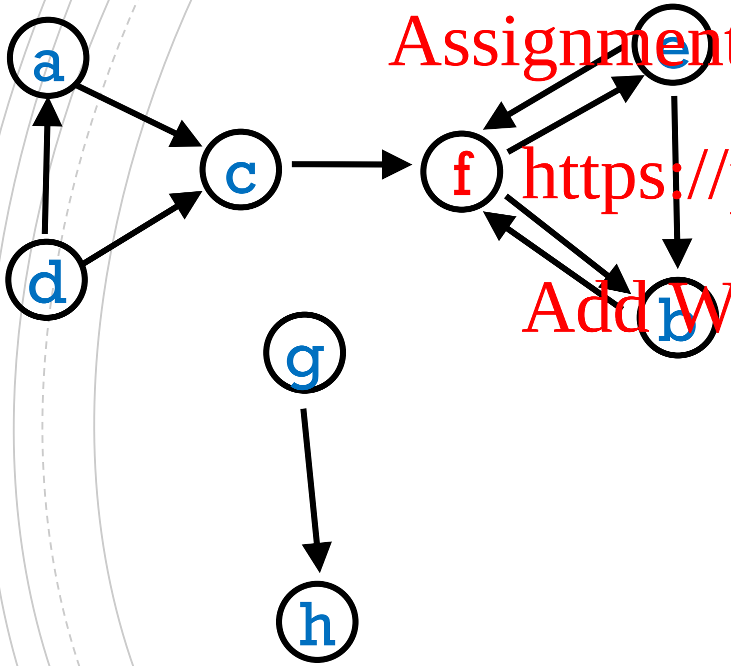
v	in degree
a	1
b	2
c	2
d	0
e	1
f	3
g	0
h	1

TERMINOLOGY: "OUT DEGREE"



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

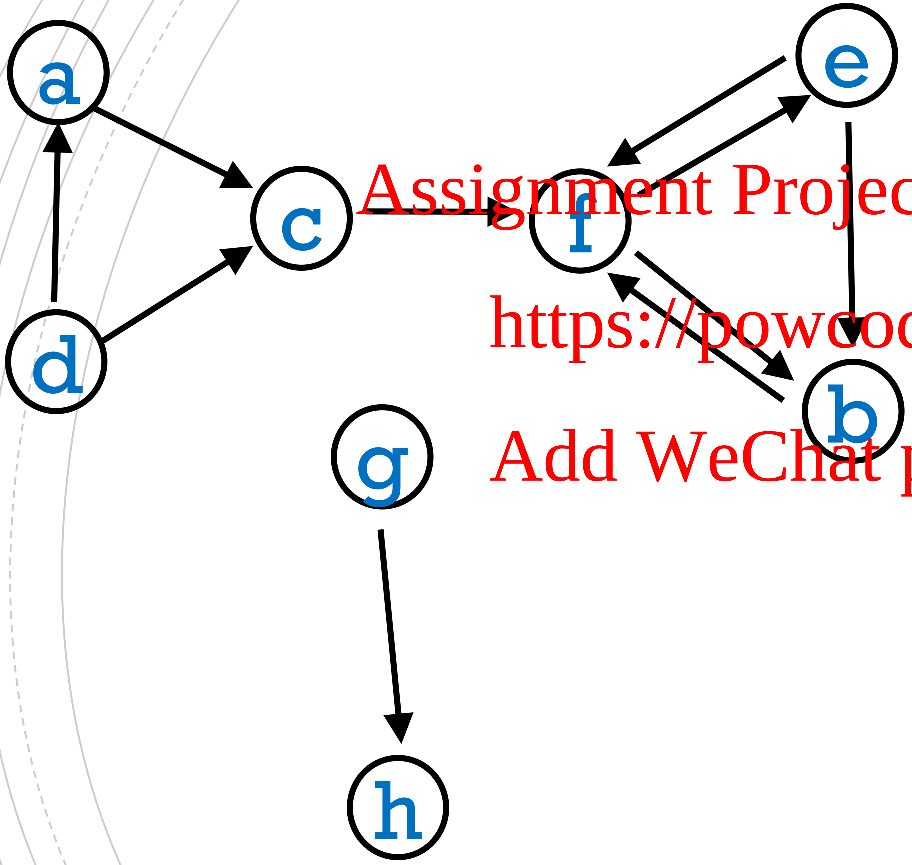
EXAMPLE: WEB PAGES



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

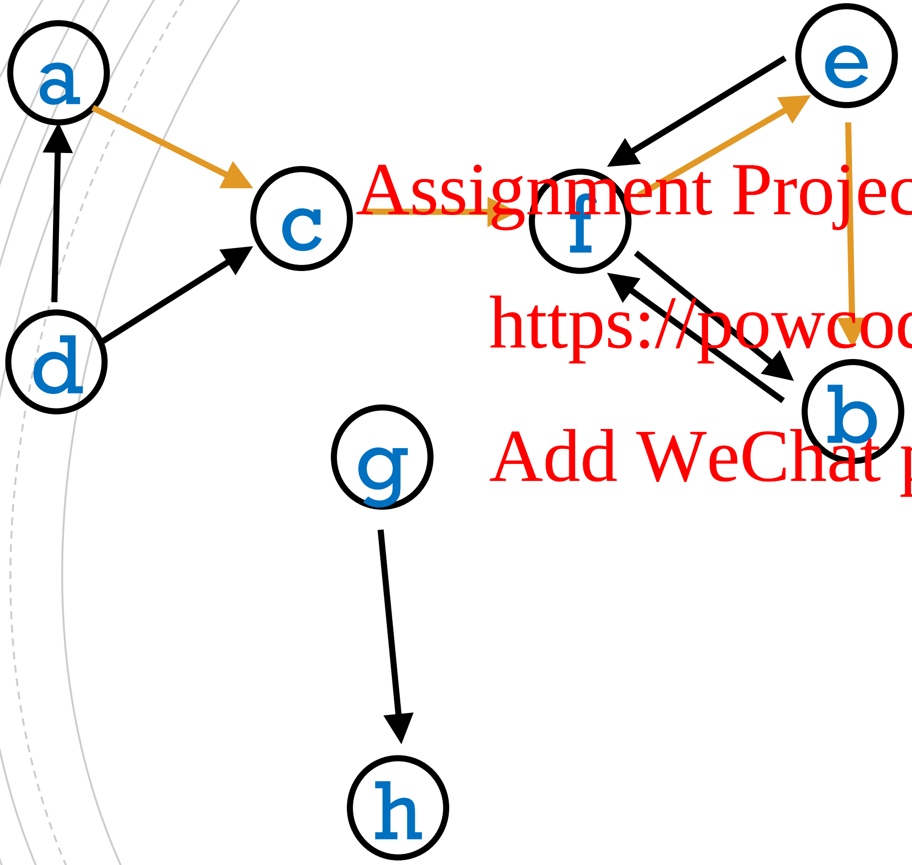
Out degree: How many web pages does some web page (e.g. f) link to ?

TERMINOLOGY: PATH



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

TERMINOLOGY: PATH

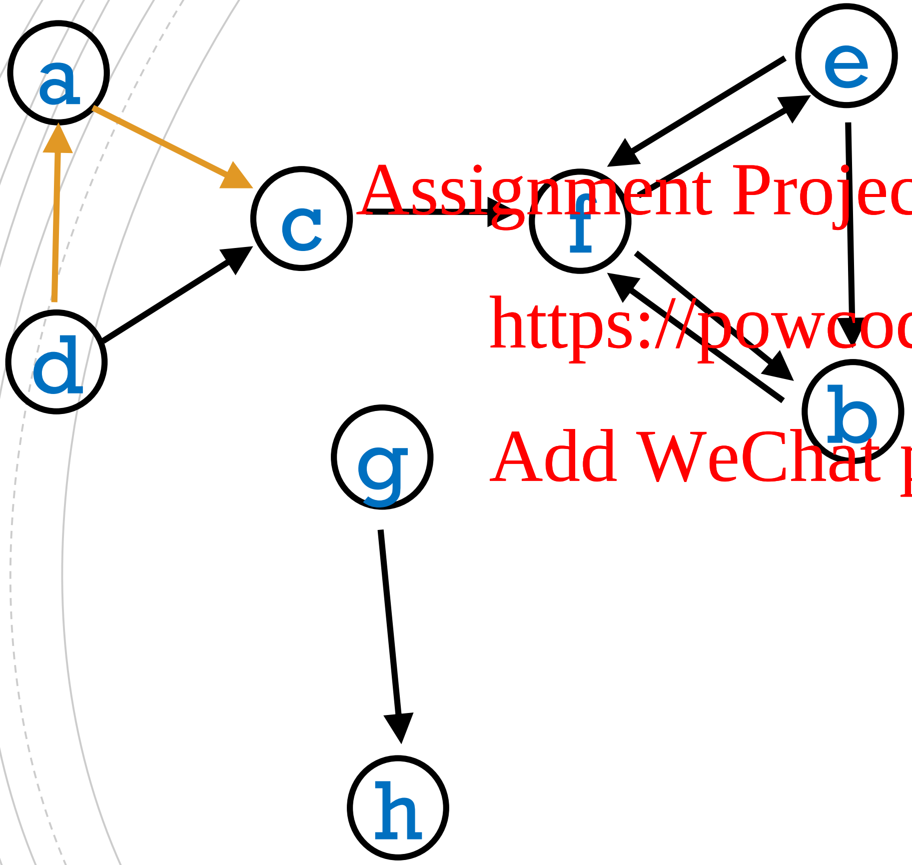


A *path* is a sequence of edges such that the end vertex of one edge is the start vertex of the next edge and no vertex is repeated except maybe first and last.

Examples

- acfeb

TERMINOLOGY: PATH

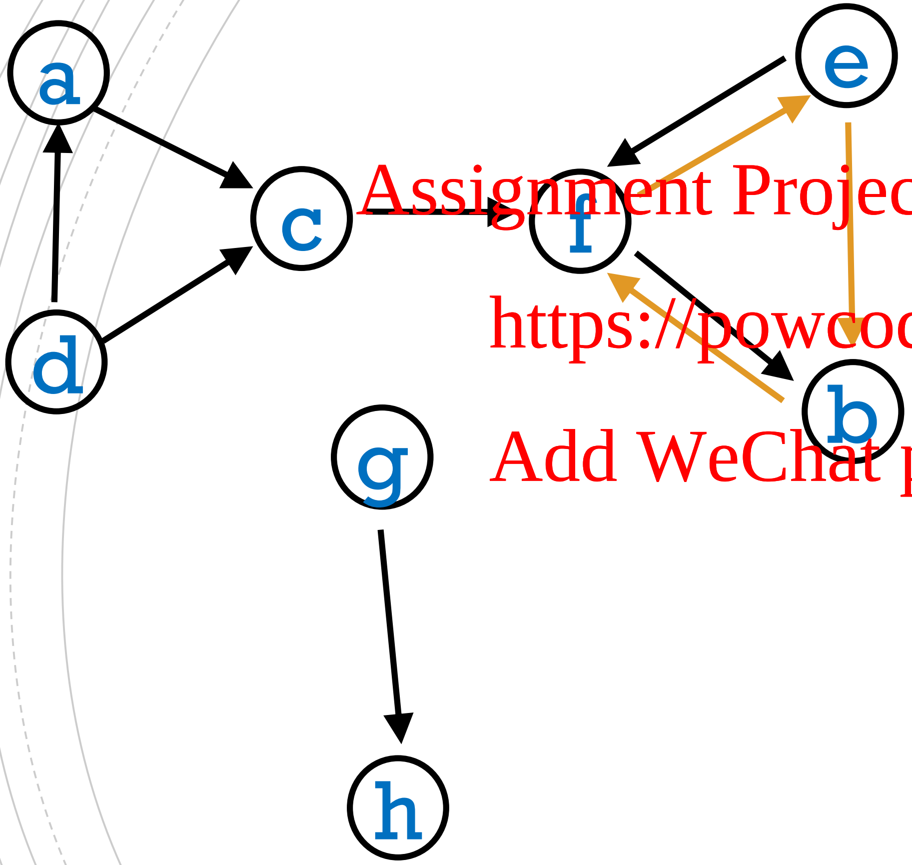


A *path* is a sequence of edges such that the end vertex of one edge is the start vertex of the next edge and no vertex is repeated except maybe first and last.

Examples

- acfeb
- dac

TERMINOLOGY: PATH



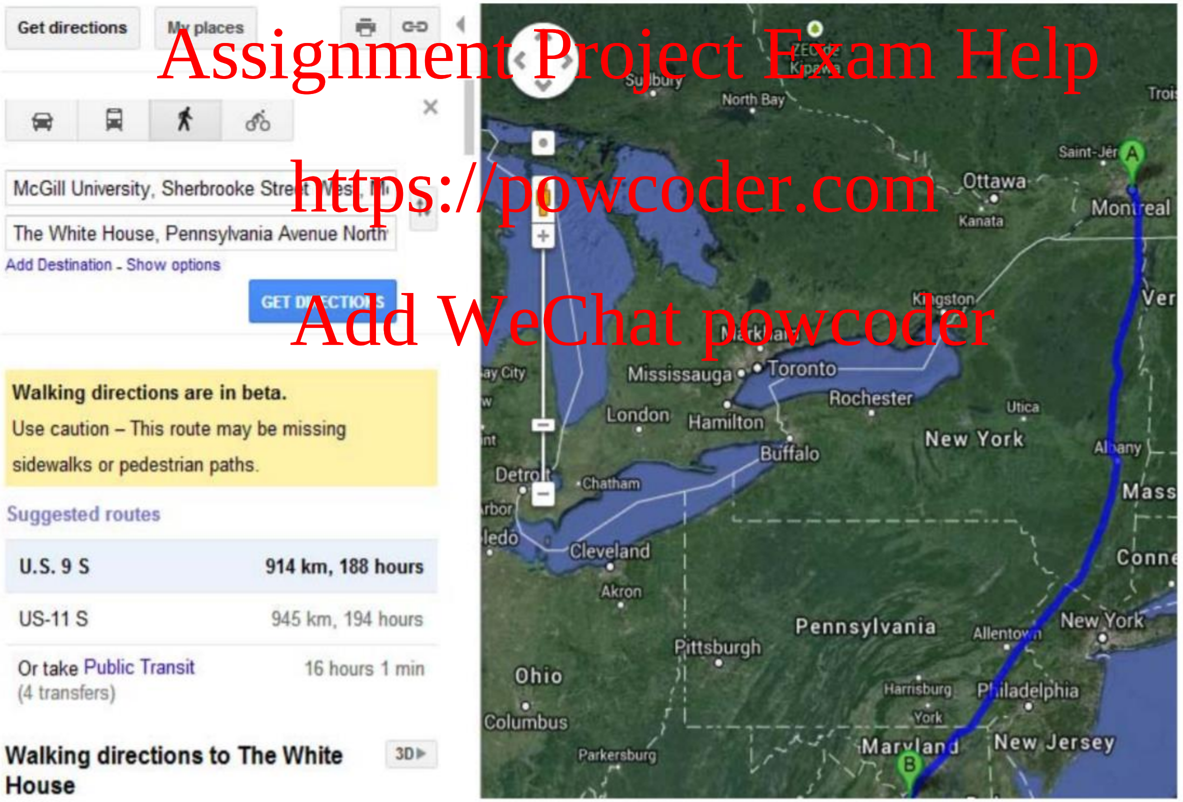
A *path* is a sequence of edges such that the end vertex of one edge is the start vertex of the next edge and no vertex is repeated except maybe first and last.

Examples

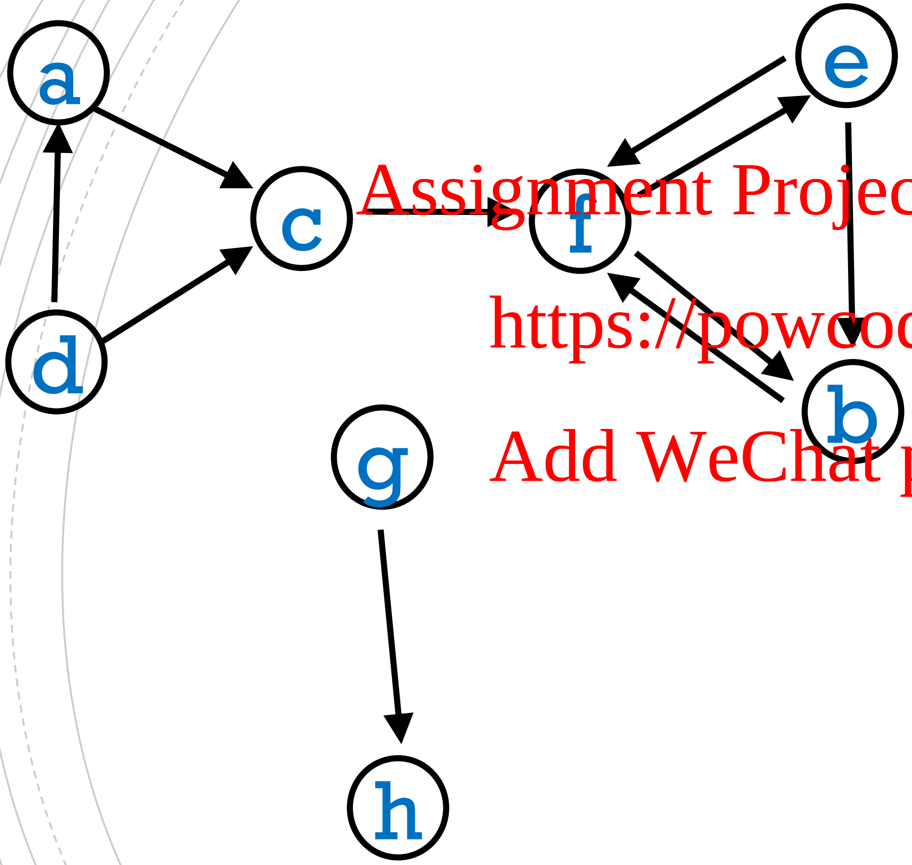
- acfeb
- dac
- febf
- …..

GRAPH ALGORITHMS IN COMP 251 (DIJKSTRA'S ALGORITHM)

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

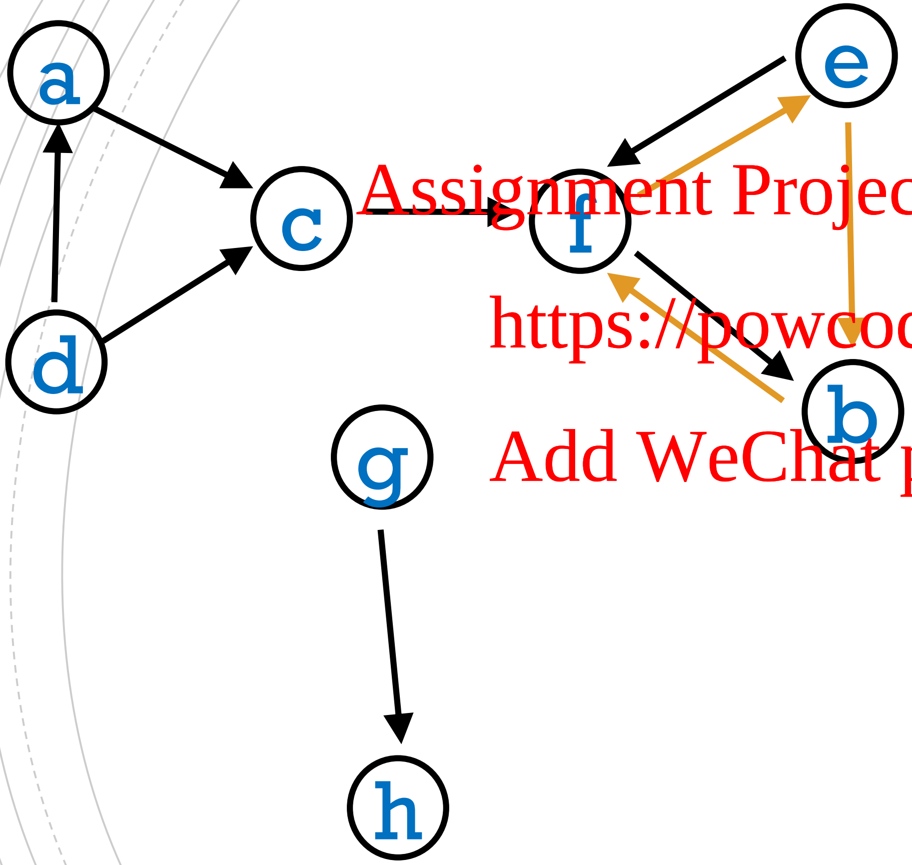


TERMINOLOGY: CYCLE



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

TERMINOLOGY: CYCLE

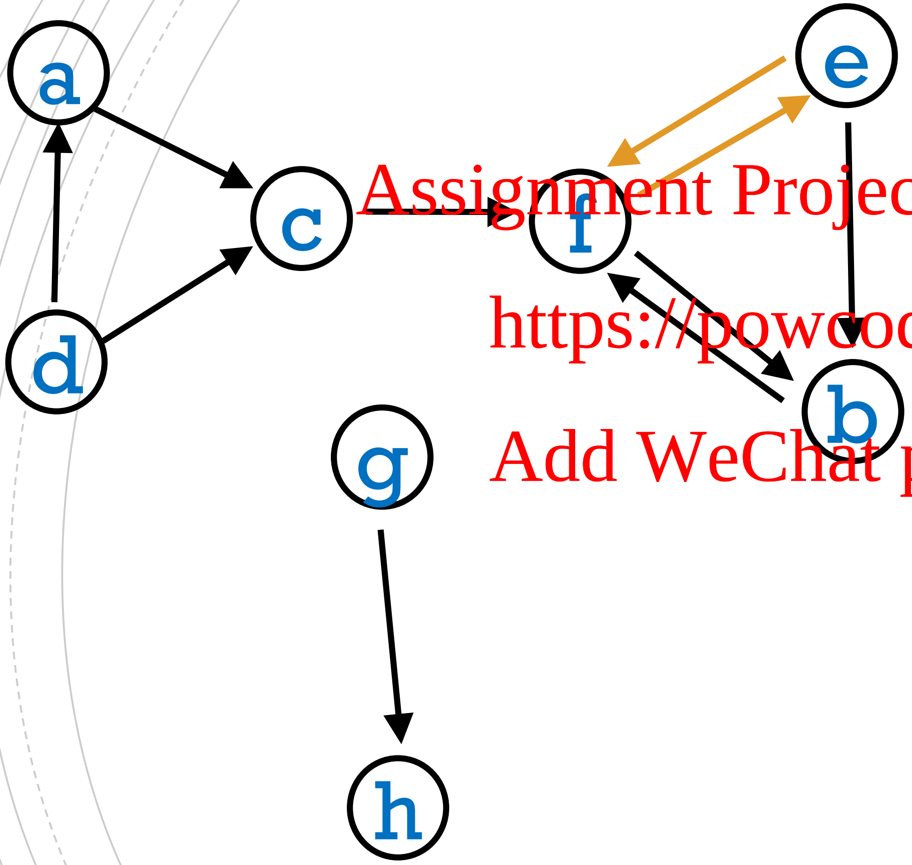


A *cycle* is a path such that the last vertex is the same as the first vertex.

Examples

- febf

TERMINOLOGY: CYCLE

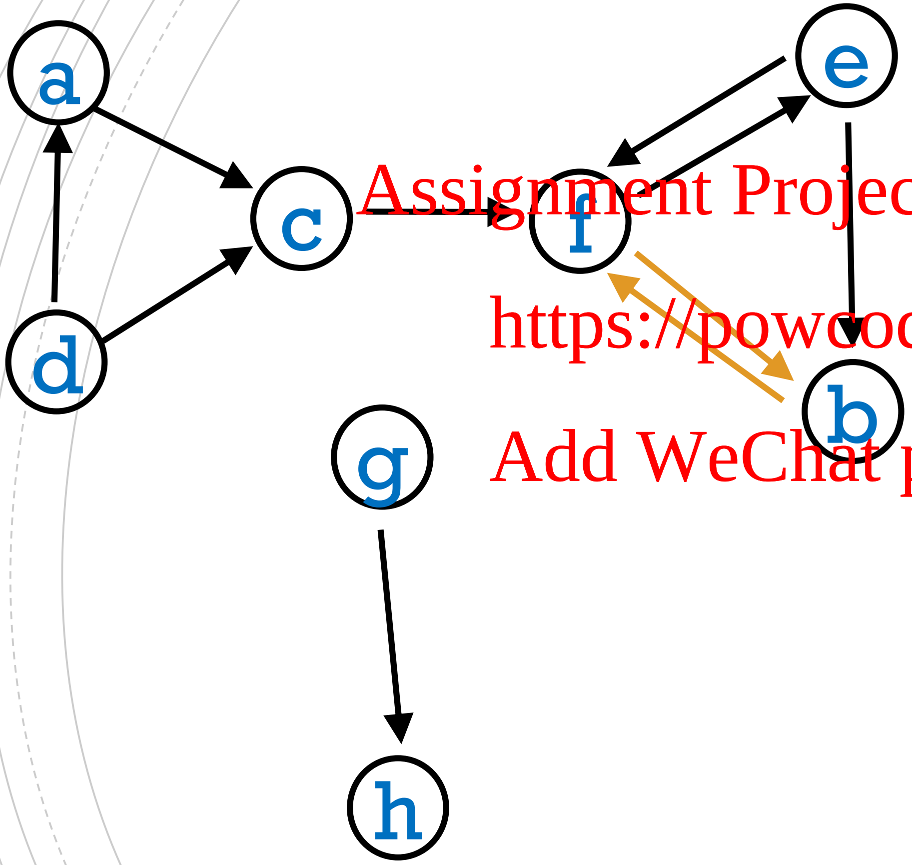


A *cycle* is a path such that the last vertex is the same as the first vertex.

Examples

- febf
- efe

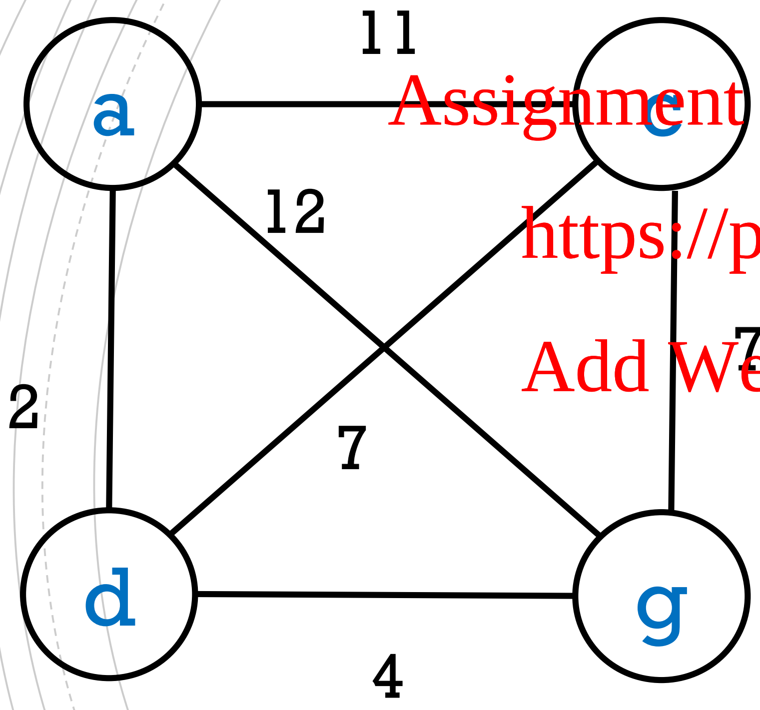
TERMINOLOGY: CYCLE



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)

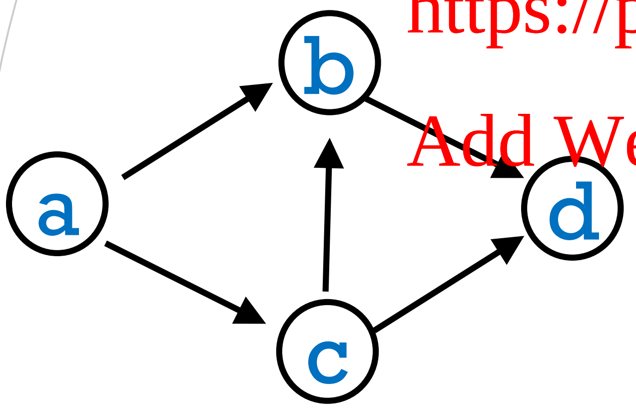


Find the shortest cycle that visits all vertices once.

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

DIRECTED *ACYCLIC* GRAPH

no cycles

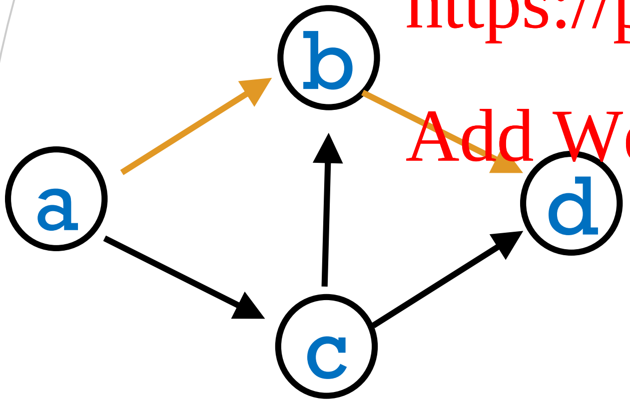


Used to capture dependencies.

There are three paths from a to d.

DIRECTED *ACYCLIC* GRAPH

no cycles

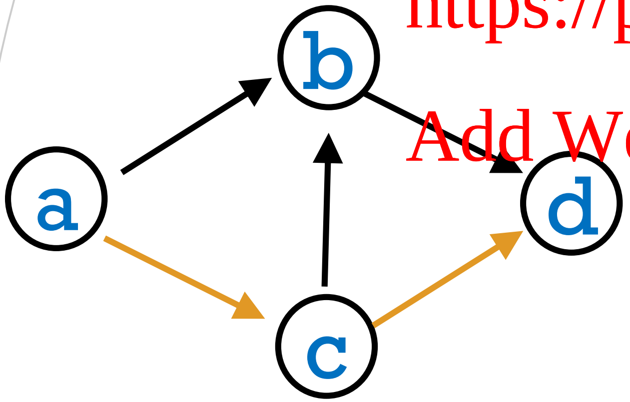


Used to capture dependencies.

There are three paths from a to d.

DIRECTED *ACYCLIC* GRAPH

no cycles

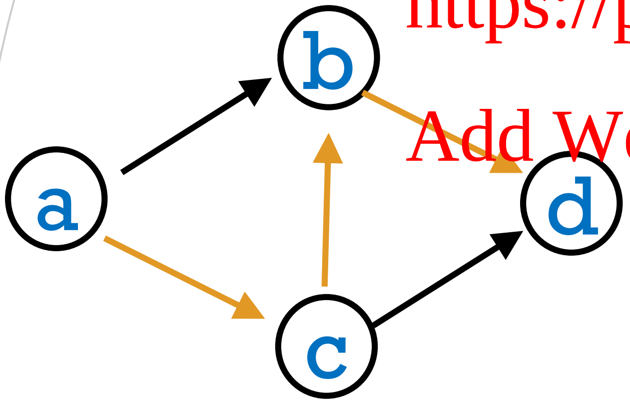


Used to capture dependencies.

There are three paths from a to d.

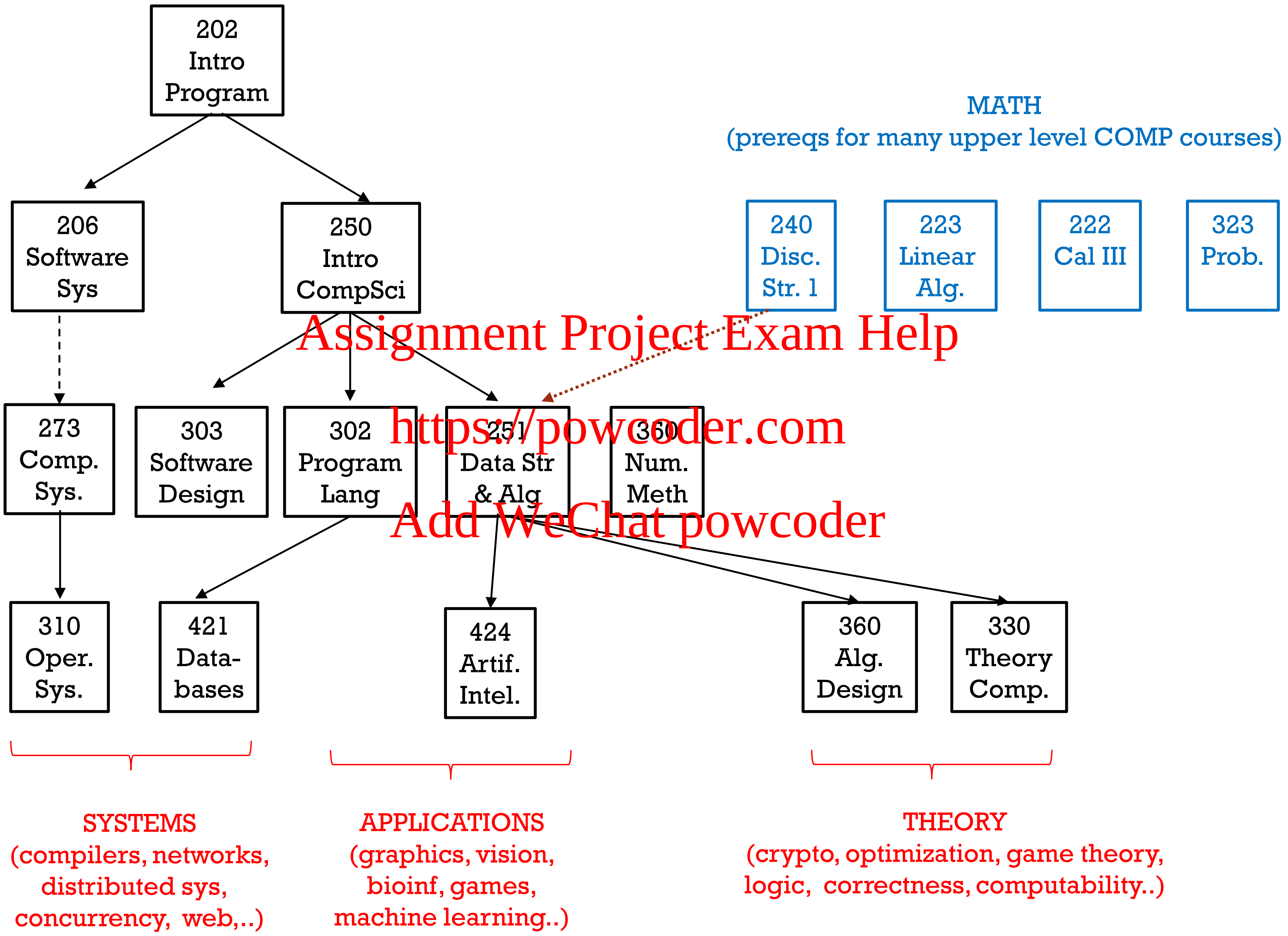
DIRECTED *ACYCLIC* GRAPH

no cycles



Used to capture dependencies.

There are three paths from a to d.


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.
273
Comp.
Sys.
303
Software
Design
302
Program
Lang
251
Data Str
& Alg
360
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 …

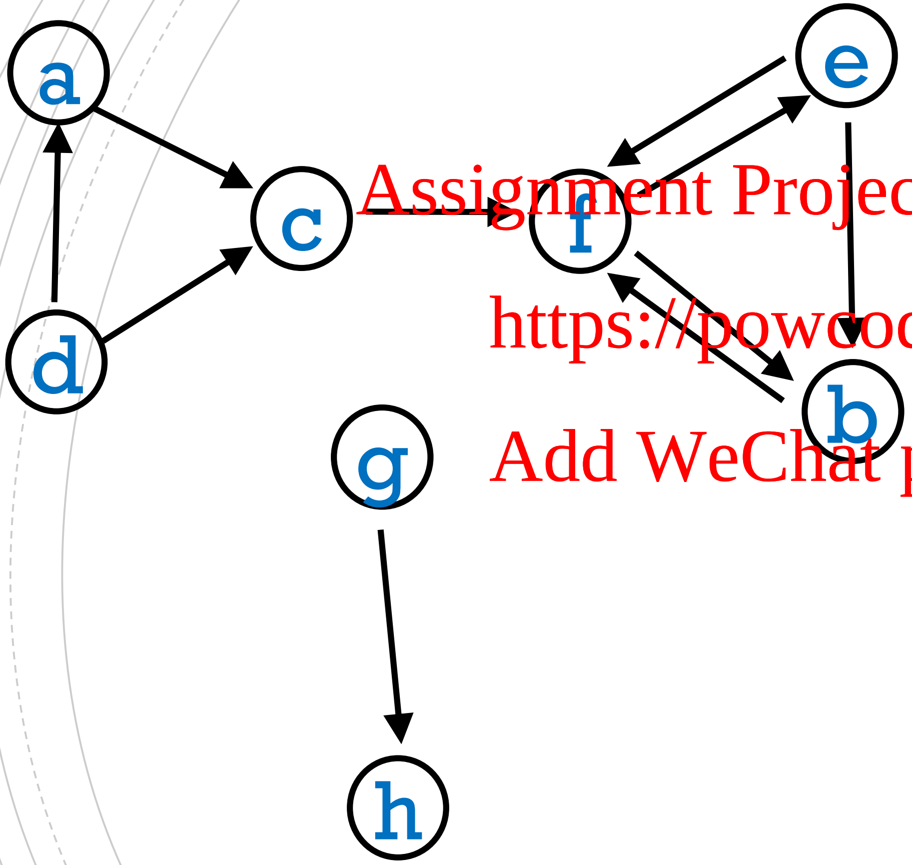
RECALL: HOW TO IMPLEMENT A ROOTED TREE IN JAVA ?

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

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

// alternatively….

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

   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?

A very basic Graph class:


```
class Graph<T> {
   class Vertex<T> { // We could have called it GNode
      ArrayList<Vertex> adjList;
      T element;
   }
}
```

HOW TO IMPLEMENT A GRAPH CLASS IN JAVA?

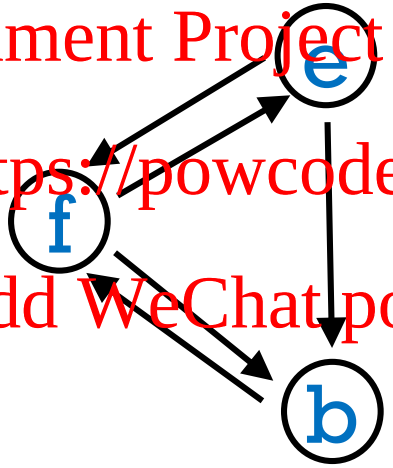
```java
class Graph<T> {
    class Vertex<T> {
        ArrayList<Edge> adjList;
        T element;
        boolean visited;
    }

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


Note that, unlike a rooted tree, there is no notion of a root vertex in a graph.

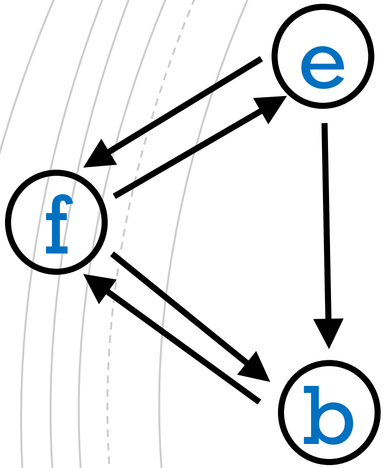
HOW TO REFERENCE VERTICES?

```
class  Graph<T> {
       ArrayList< Vertex<T> > vertexList;
            :
       class Vertex<T> { ... }
       class  Edge<T>  { ... }
}
```

HOW MANY OBJECTS ?



HOW MANY OBJECTS ?



Graph

ArrayList <Vertex>

Vertex

Vertex

Vertex

Edge

Edge

Edge

Edge

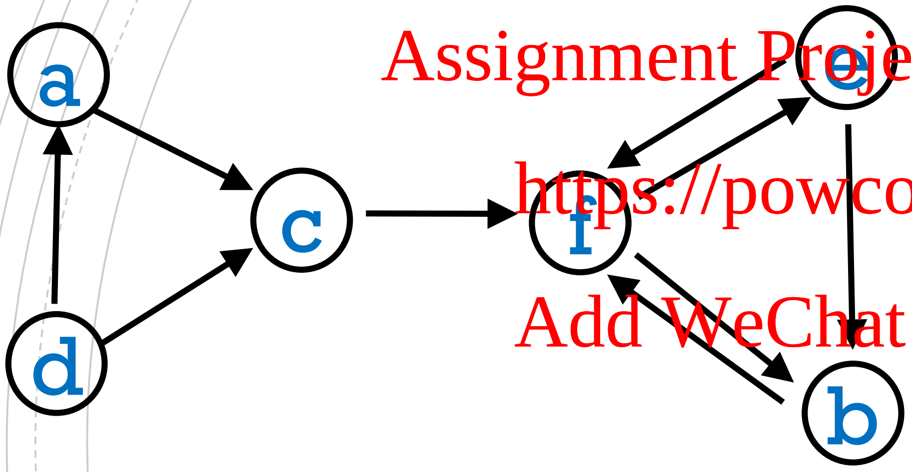
Edge

ArrayList<Edge>

ArrayList<Edge>

ArrayList<Edge>

ADJACENCY MATRIX

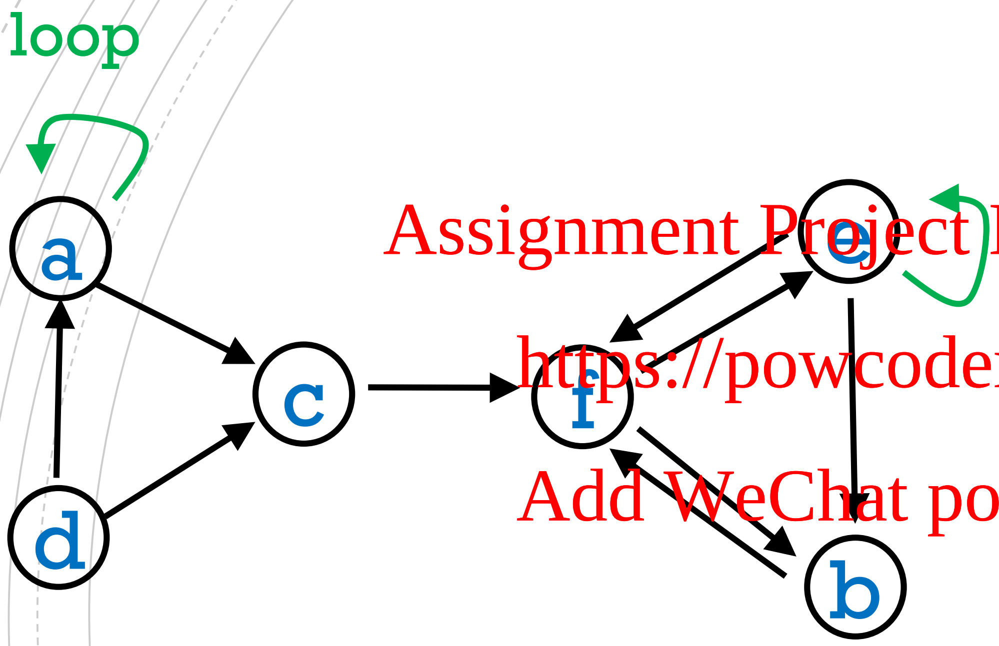


	a	b	c	d	e	f
a	0	0	1	0	0	0
b	0	0	0	0	0	1
c	0	0	0	0	0	1
d	1	0	1	0	0	0
e	0	1	0	0	0	1
f	0	1	0	0	1	0

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

```
boolean[][] adjMatrix = new boolean[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

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

```
boolean[][] adjMatrix = new Boolean[6][6]
```

"DEFINITIONS"

Consider a graph with n vertices.

We say that the graph is *dense* if the number of edges is close to n^2.

We say that the graph is *sparse* if the 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

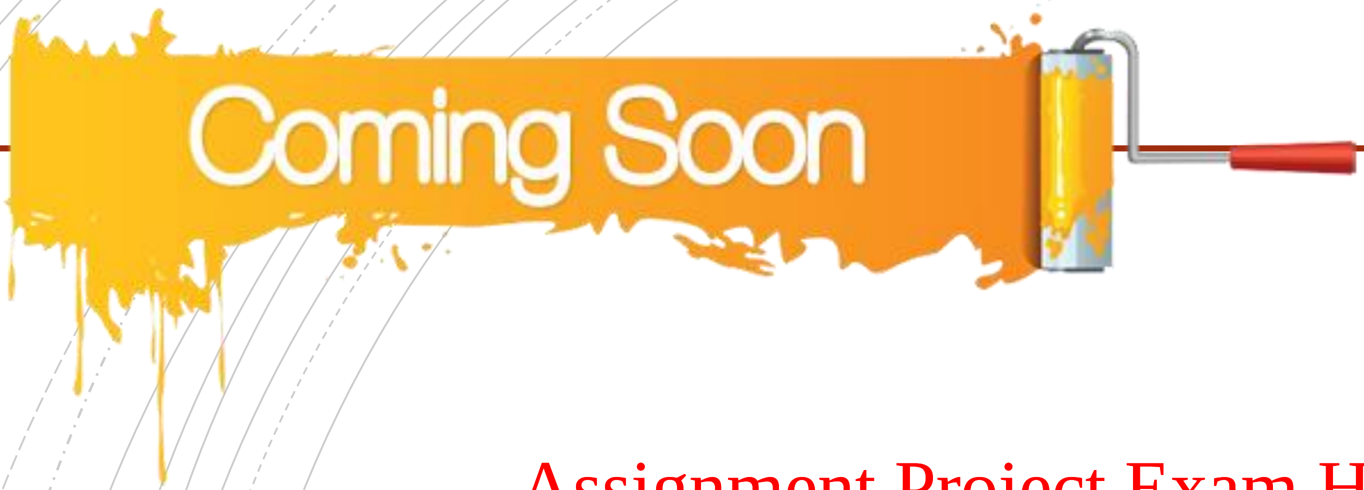
Would you use an *adjacency list* or *adjacency matrix* for each of the following?

- 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).

COMING UP!

- Recursive graph traversal
  - depth first

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



In the next videos:

- More on graphs