Algorithms and Data Structures for Data Science
Graphs

CS 277
Brad Solomon

April 10, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

This week only: Lab room and OH Changes

Friday April 14th: AE3’s Celebration of Teaching in 1306 Everitt

Our lab will be in <u>2101 Everitt</u> instead!

Office Hour Changes: My OH will be Friday between 3:15 and 4:15

There will not be OH on Thursday April 13th!

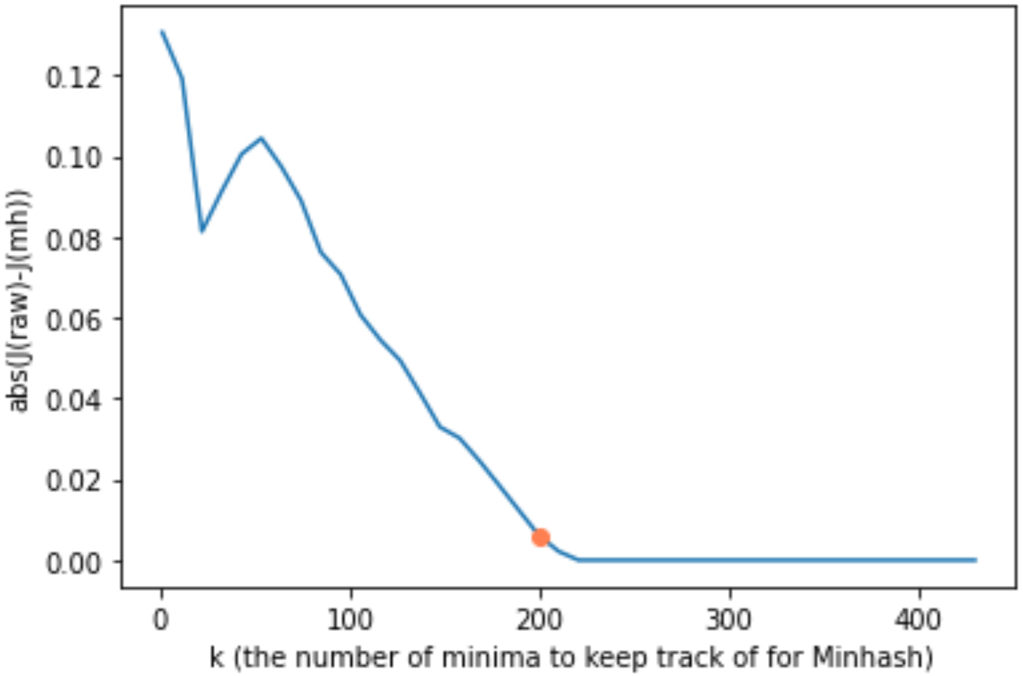
Learning Objectives

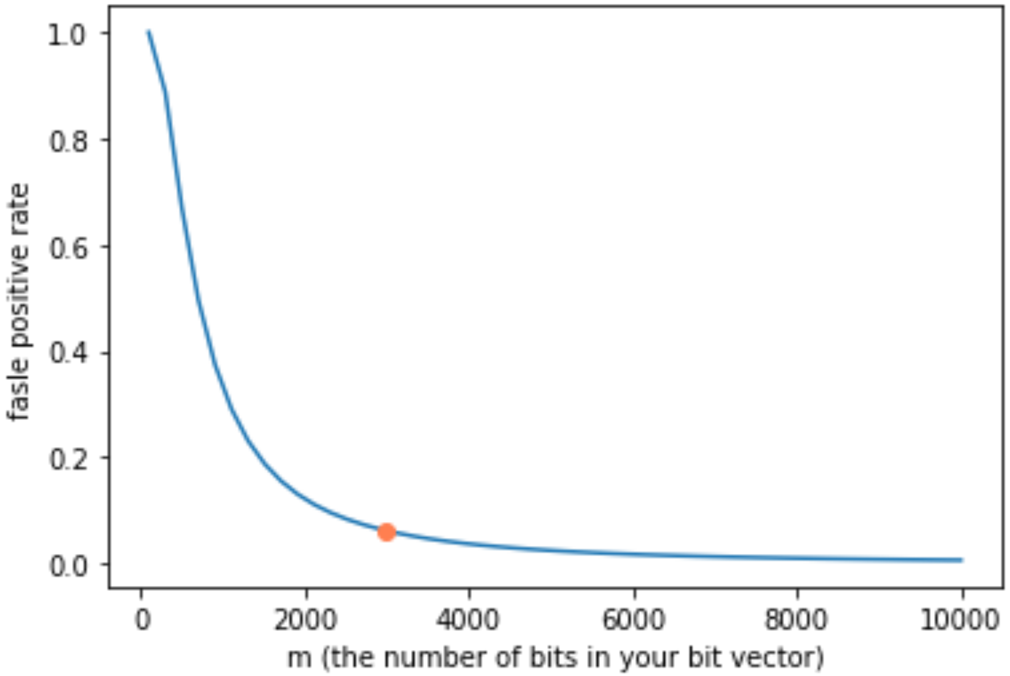
Define graph vocabulary

Discuss graph implementation / storage strategies

Define key graph functions and discuss implementation details

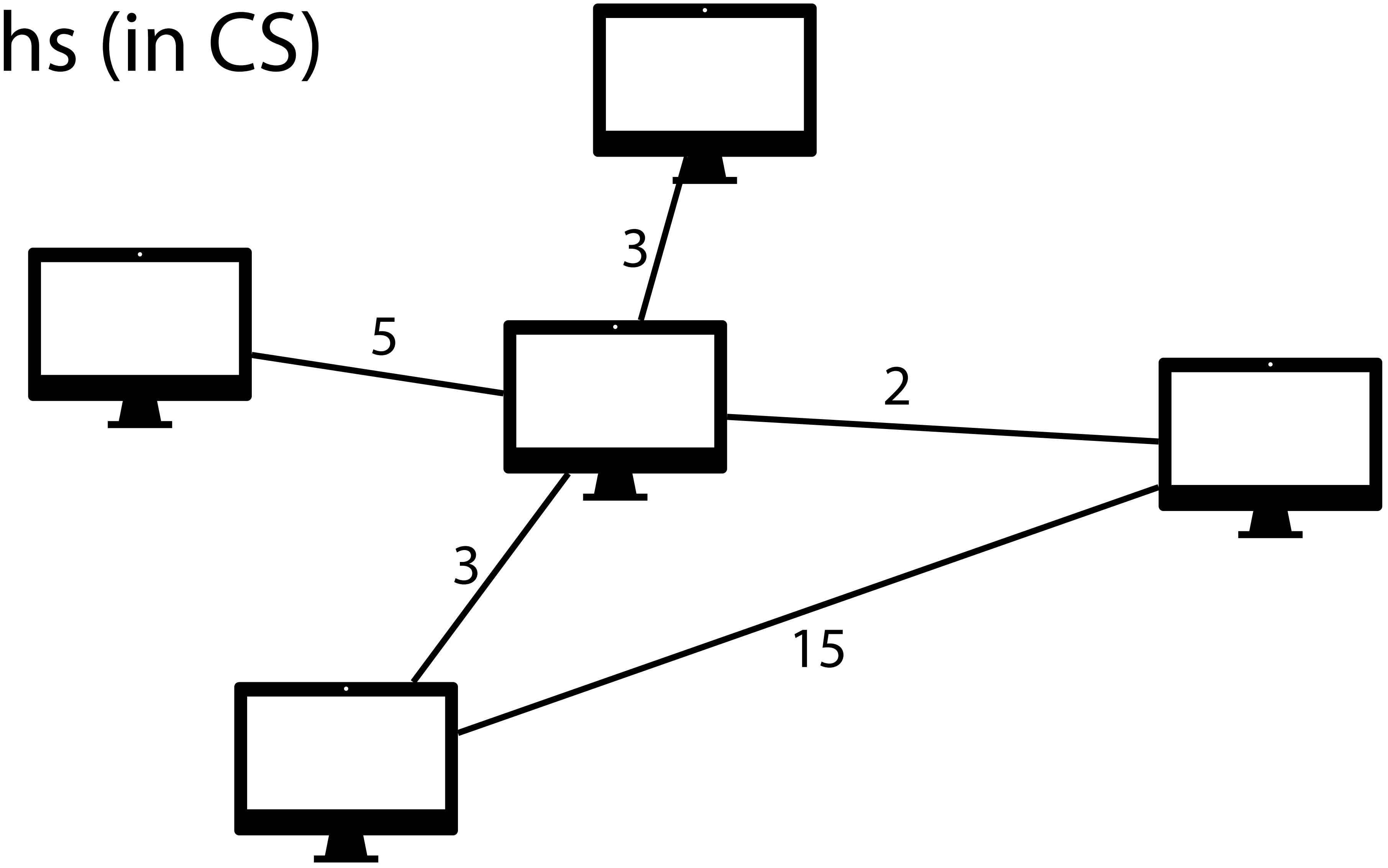
Graphs (for most people)

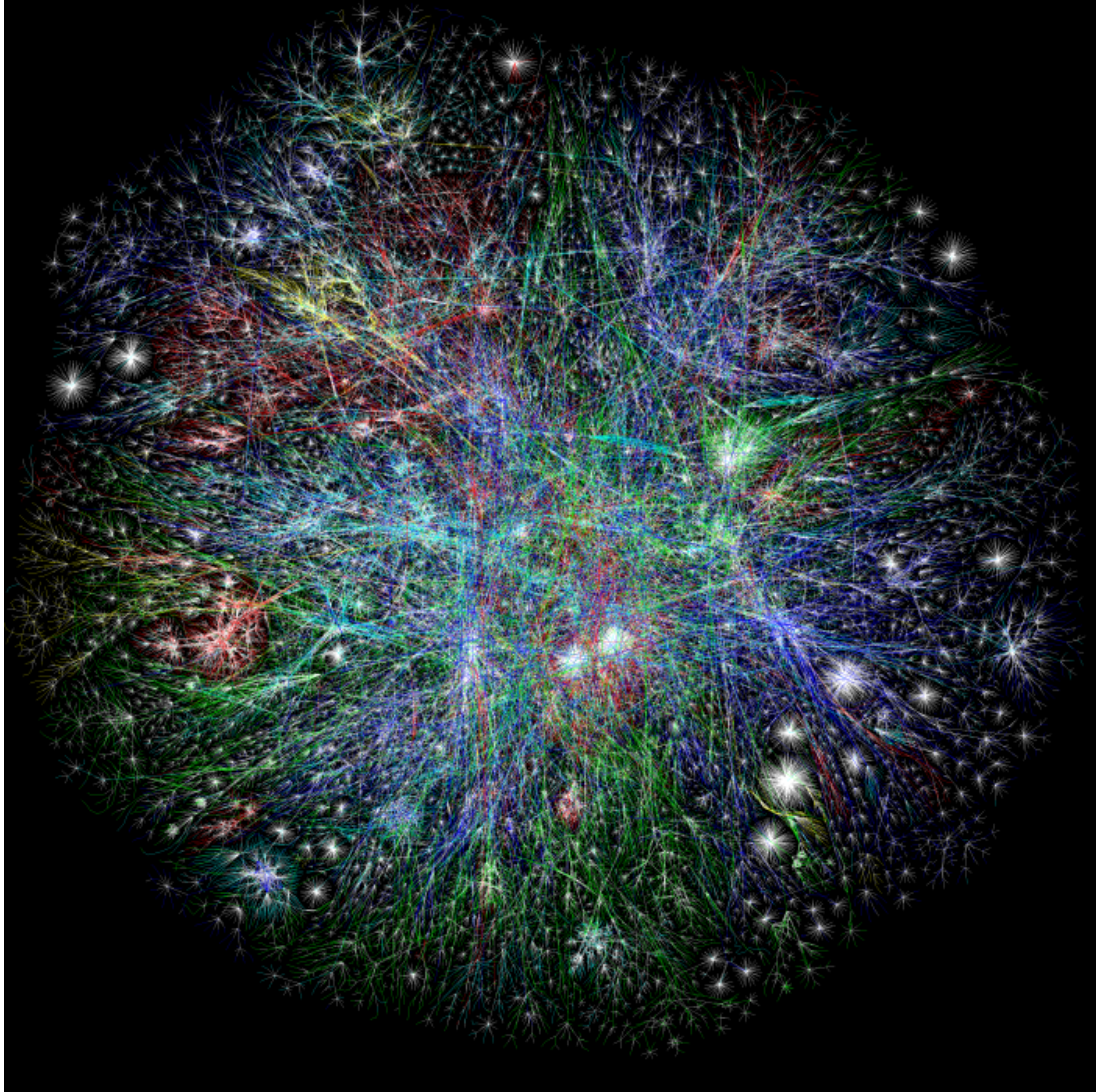




This graph was given as part of justification for MP 2: Sketching!

Graphs (in CS)



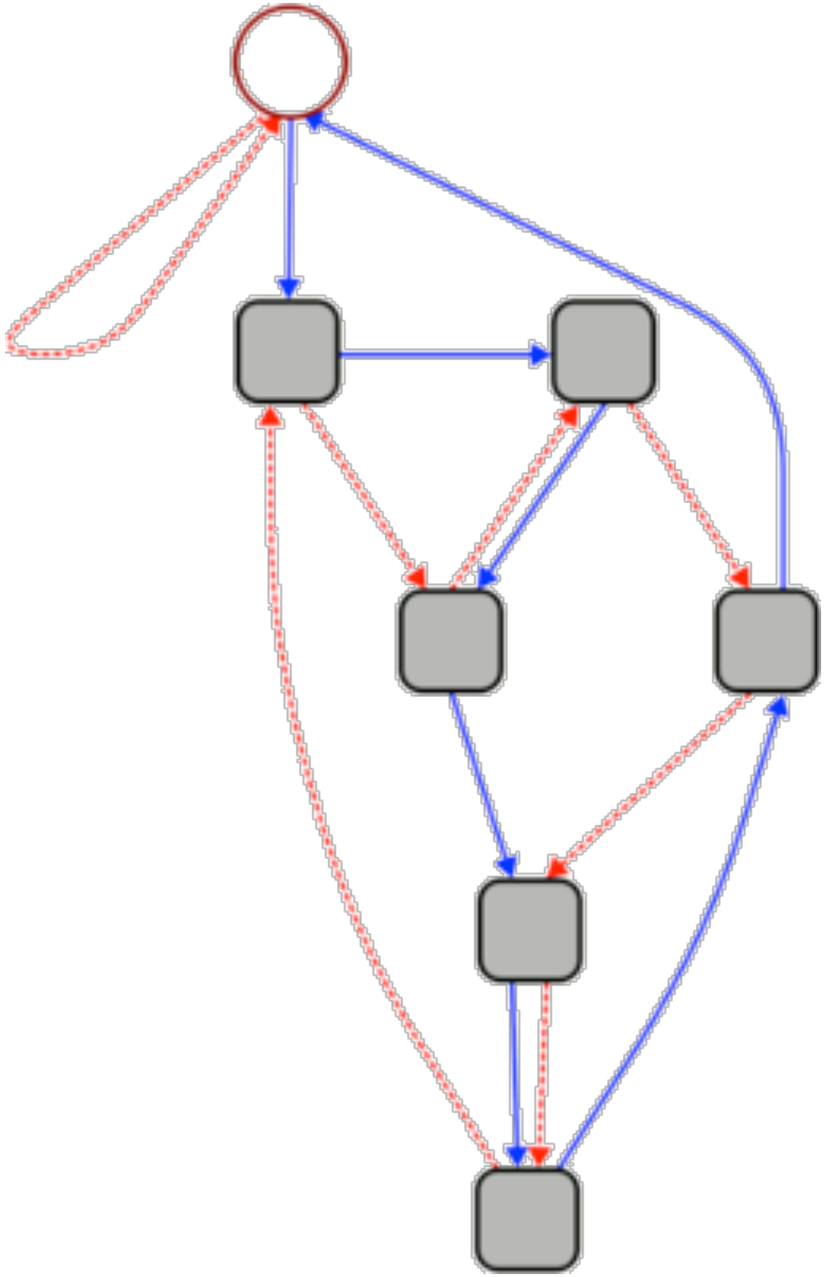


Nodes: Routers and servers

Edges: Connections

The Internet 2003

The OPTE Project (2003)



This graph can be used to quickly calculate whether a given number is divisible by 7.

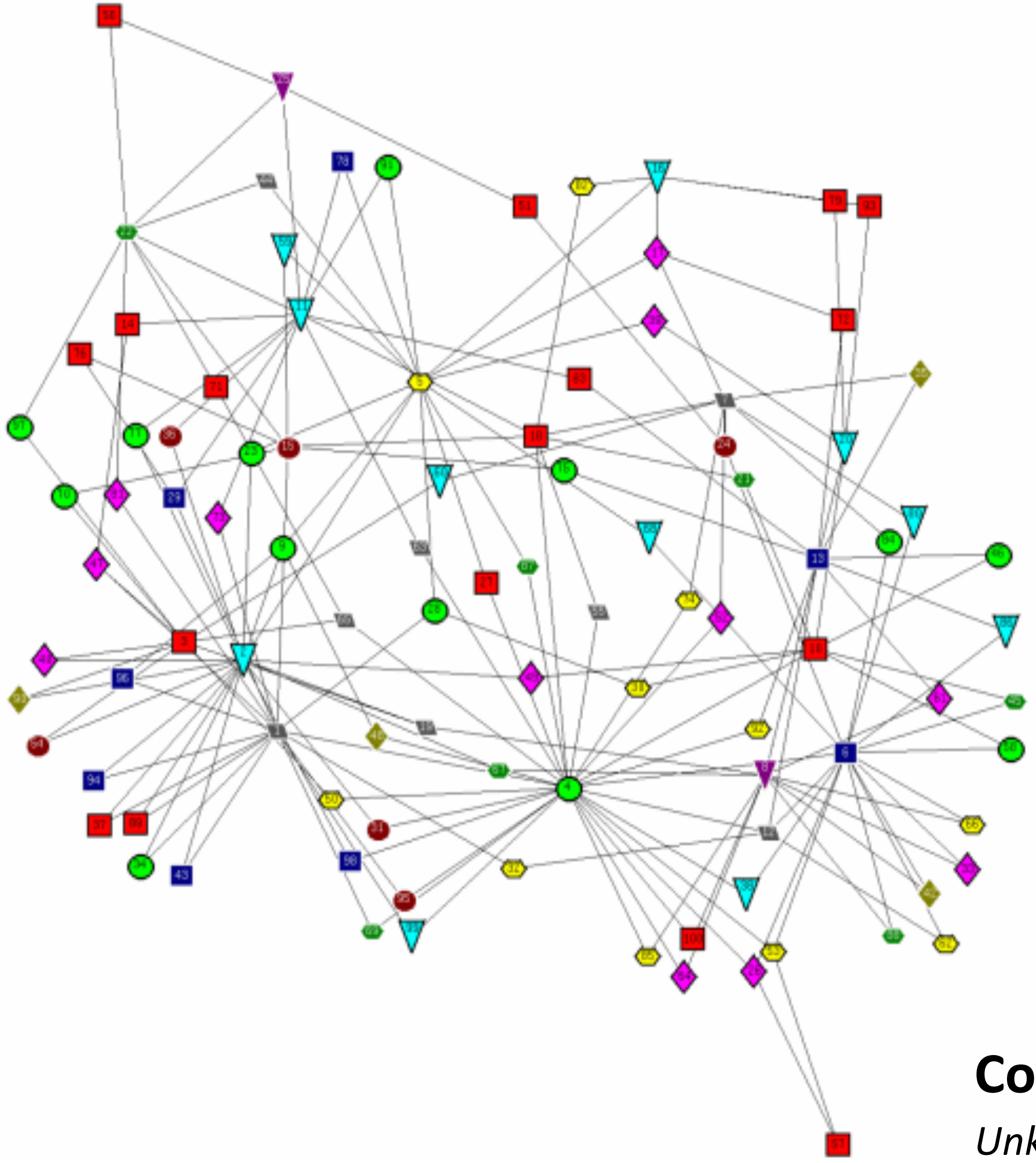
1. Start at the circle node at the top.
2. For each digit **d** in the given number, follow **d** blue (solid) edges in succession. As you move from one digit to the next, follow **1** red (dashed) edge.
3. If you end up back at the circle node, your number is divisible by 7.

3703

“Rule of 7”

Unknown Source

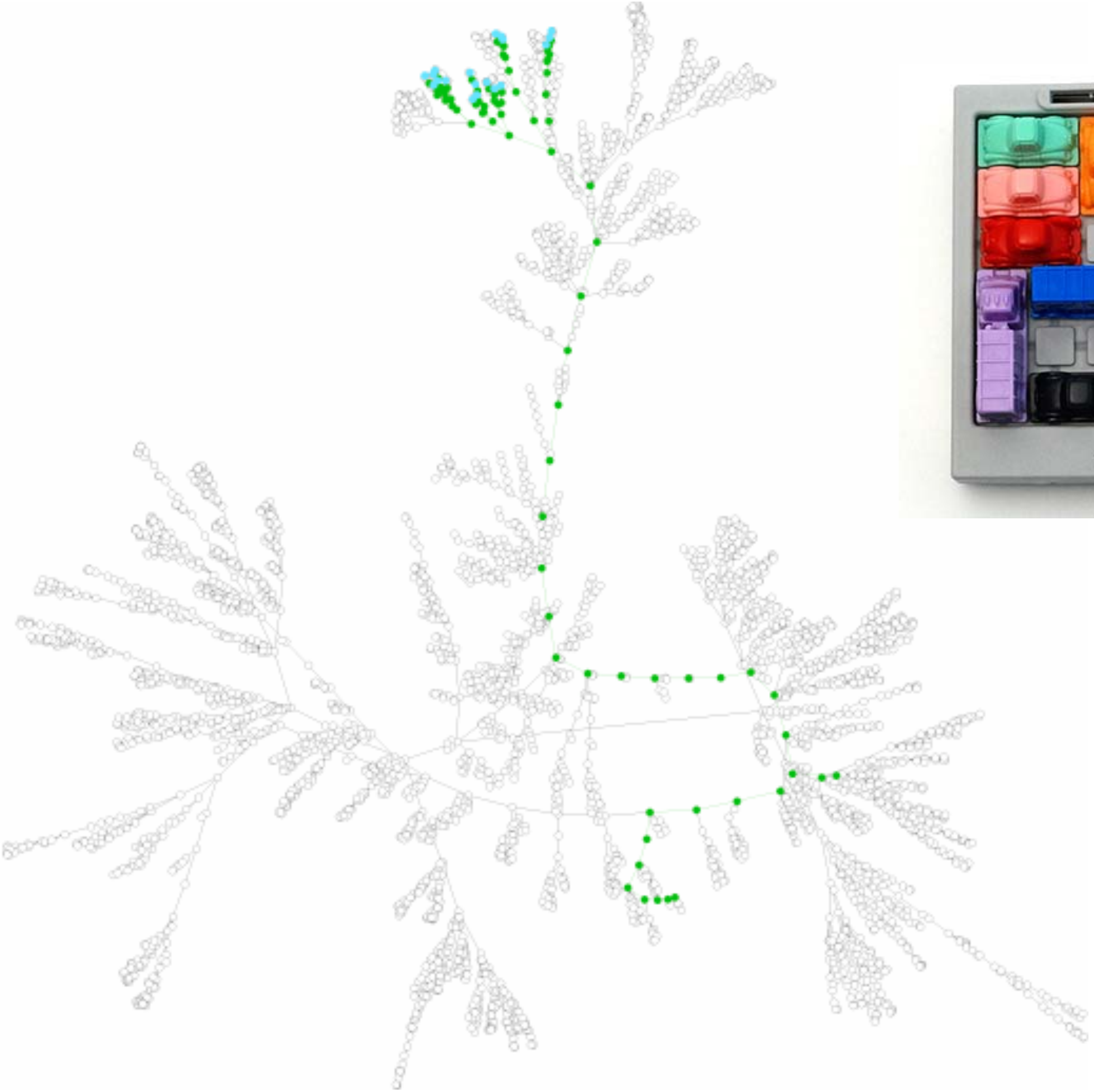
Presented by Cinda Heeren, 2016



Conflict-Free Final Exam Scheduling Graph

Unknown Source

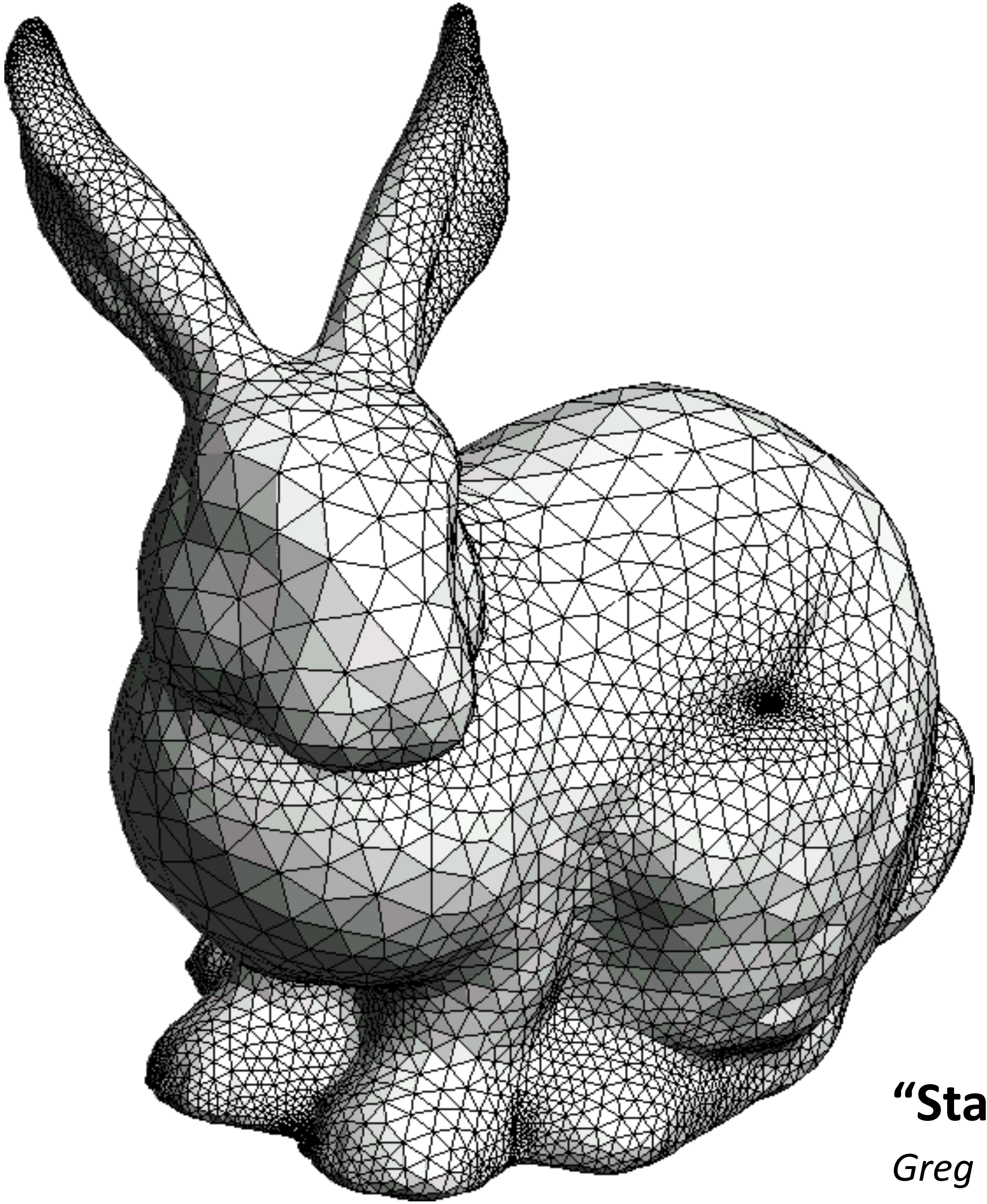
Presented by Cinda Heeren, 2016



“Rush Hour” Solution

Unknown Source

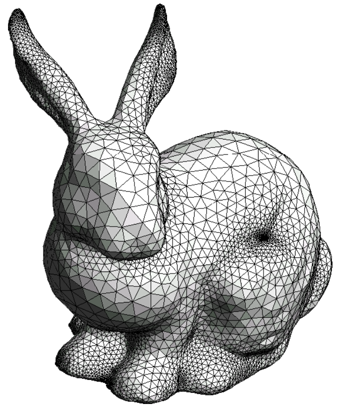
Presented by Cinda Heeren, 2016



“Stanford Bunny”

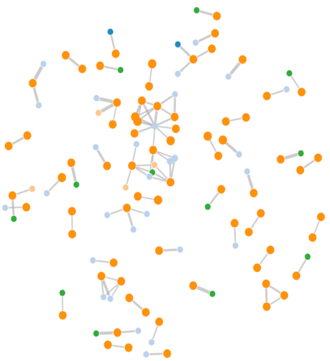
Greg Turk and Mark Levoy (1994)

Graphs



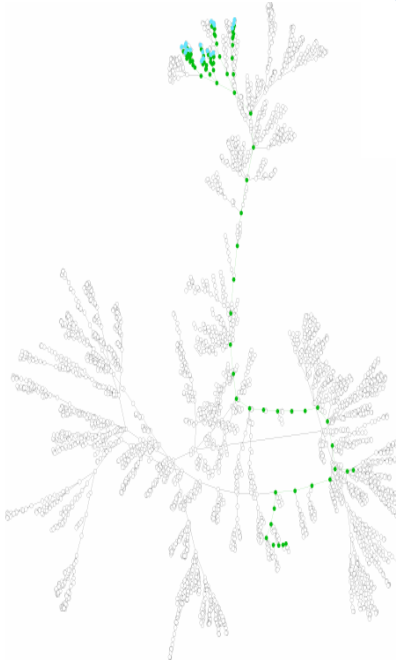
To study all of these structures:

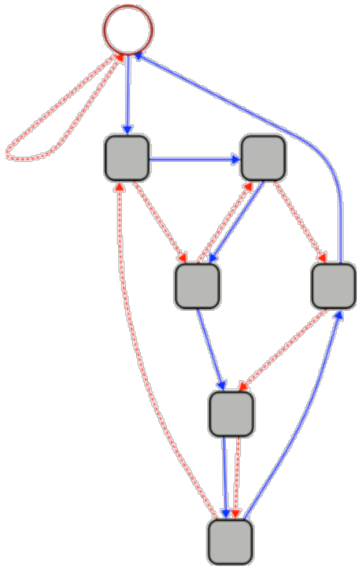
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms

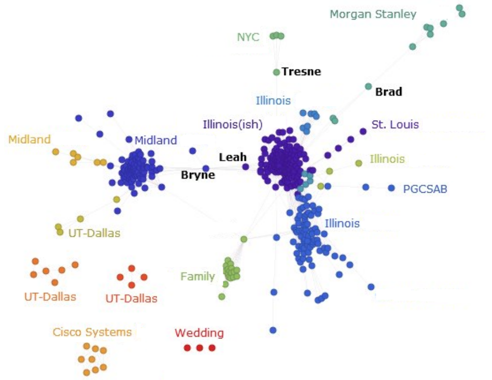


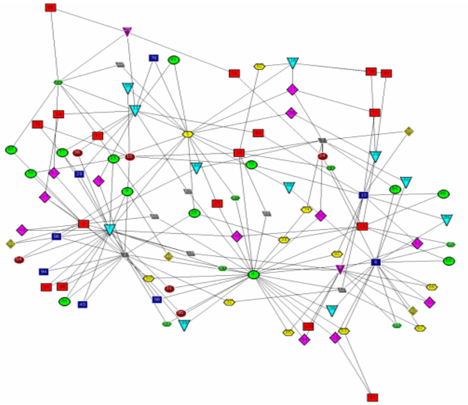
Queen Gertrude
King Claudius
Hamlet
Greeks
Diomedes
Cressida
Trojans
Troilus

HAMLET TROILUS AND CRESSIDA





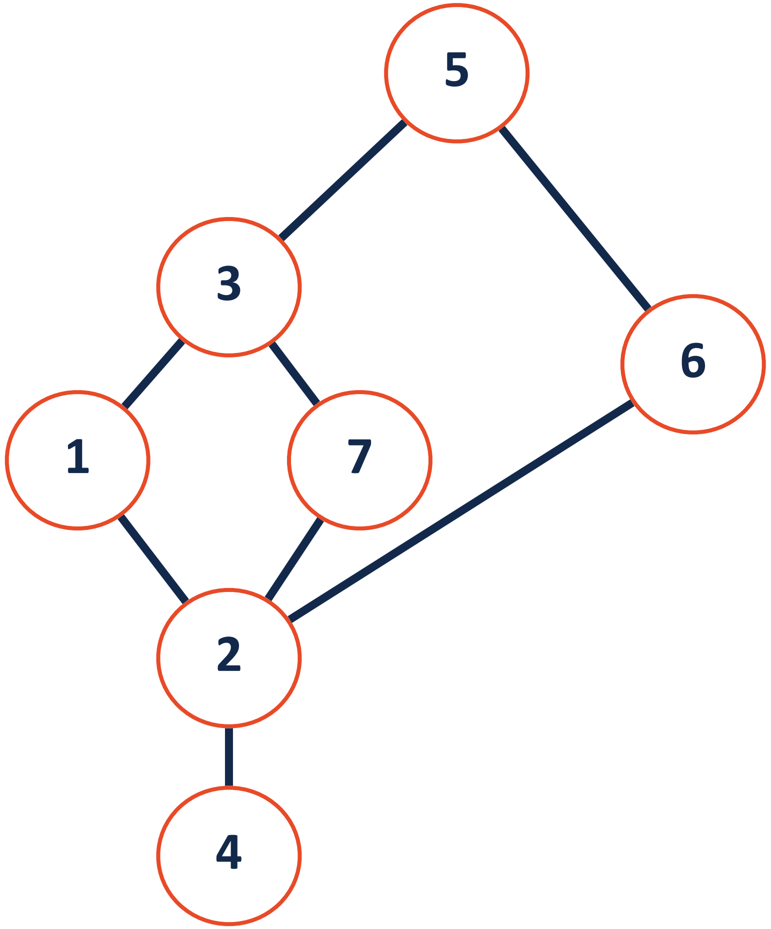




Graph Vocabulary

$G = (V, E)$

A **graph** is a data structure containing a set of vertices and a set of edges

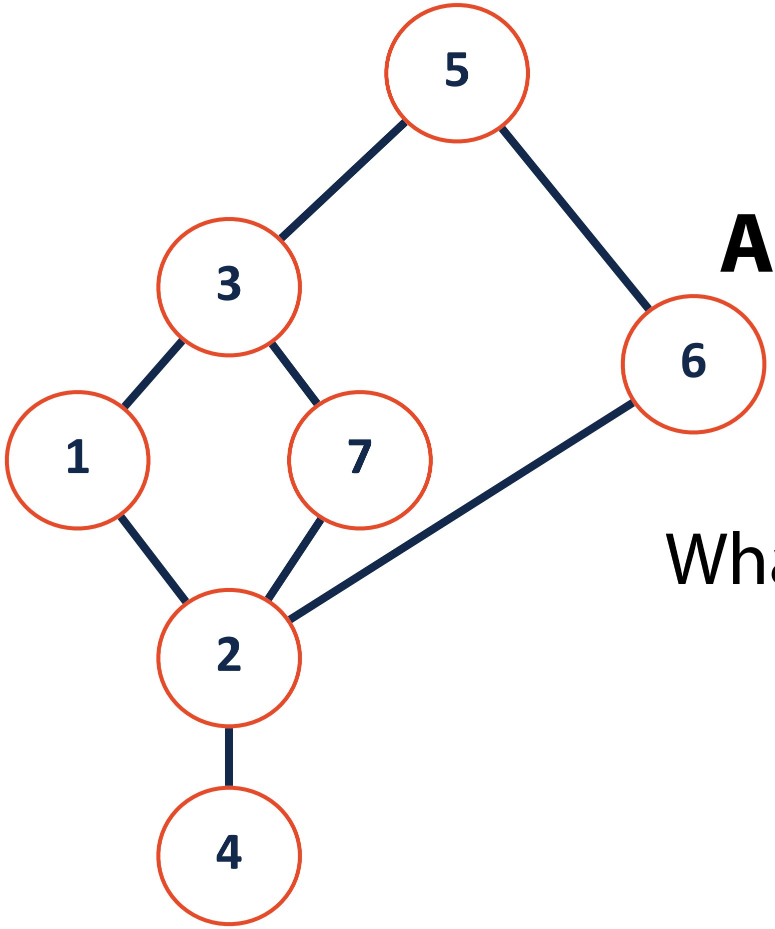


Vertex:

Edges:

Graph Vocabulary

Each vertex can have many edges



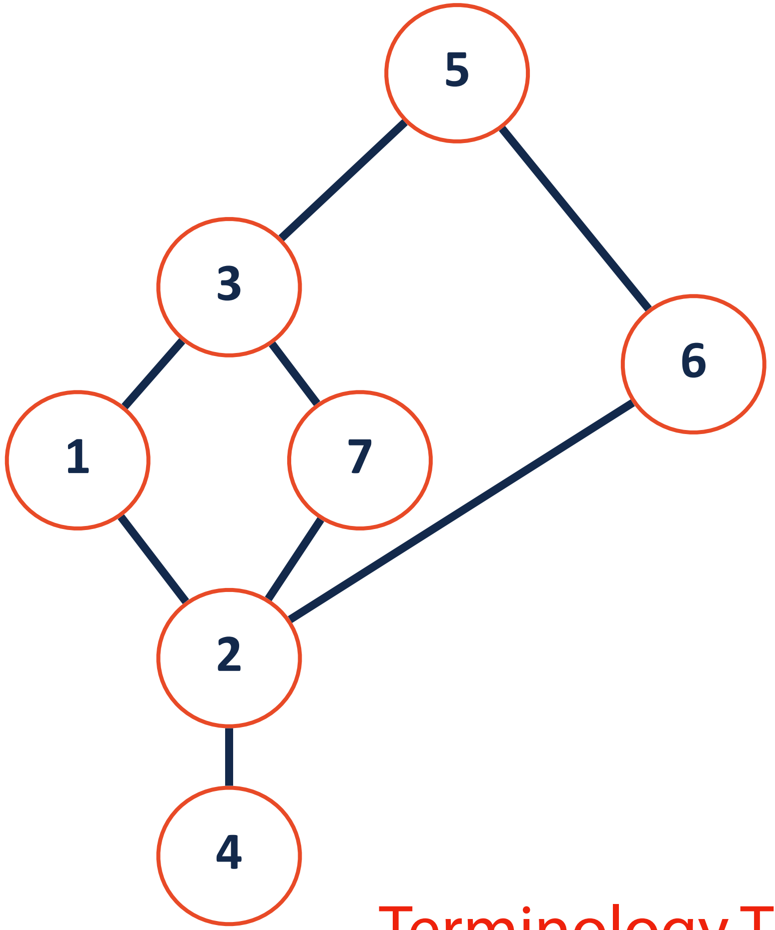
Degree: # of edges touching a vertex

Adjacency: Two vertices are adjacent if they are connected by an edge

What edges are adjacent to 4? To 2?

Graph Vocabulary

A graph has **no root** and **may contain cycles**



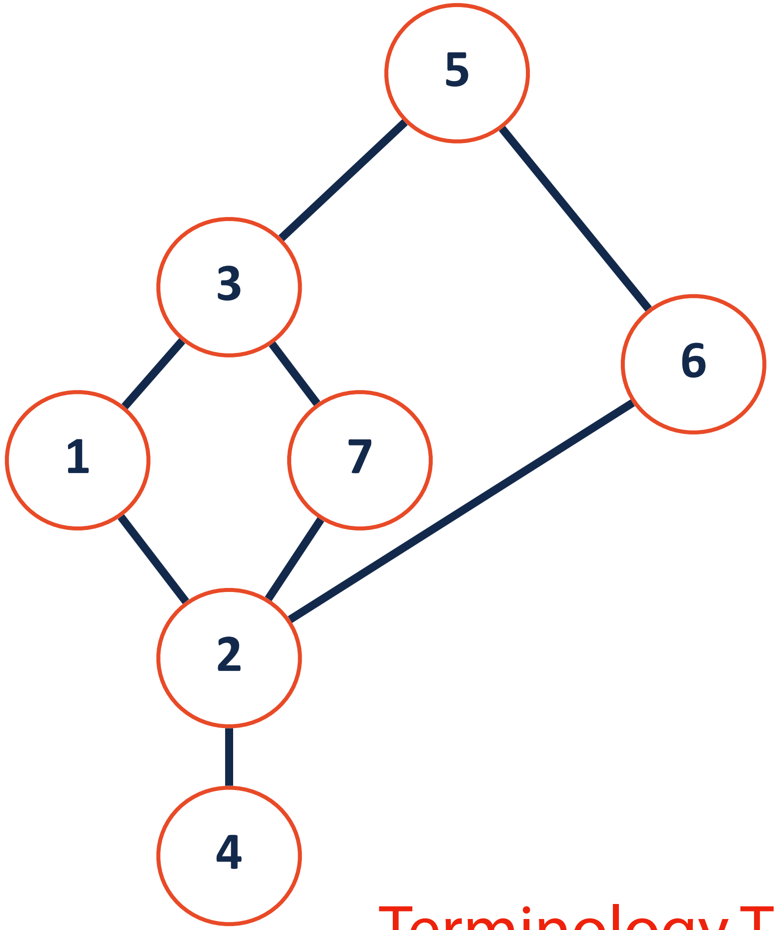
Path: A sequence of vertices (or edges) between two nodes

What is a path between 4 and 3?

Terminology Trivia: Every tree is a graph but not every graph is a tree

Graph Vocabulary

A graph has **no root** and **may contain cycles**



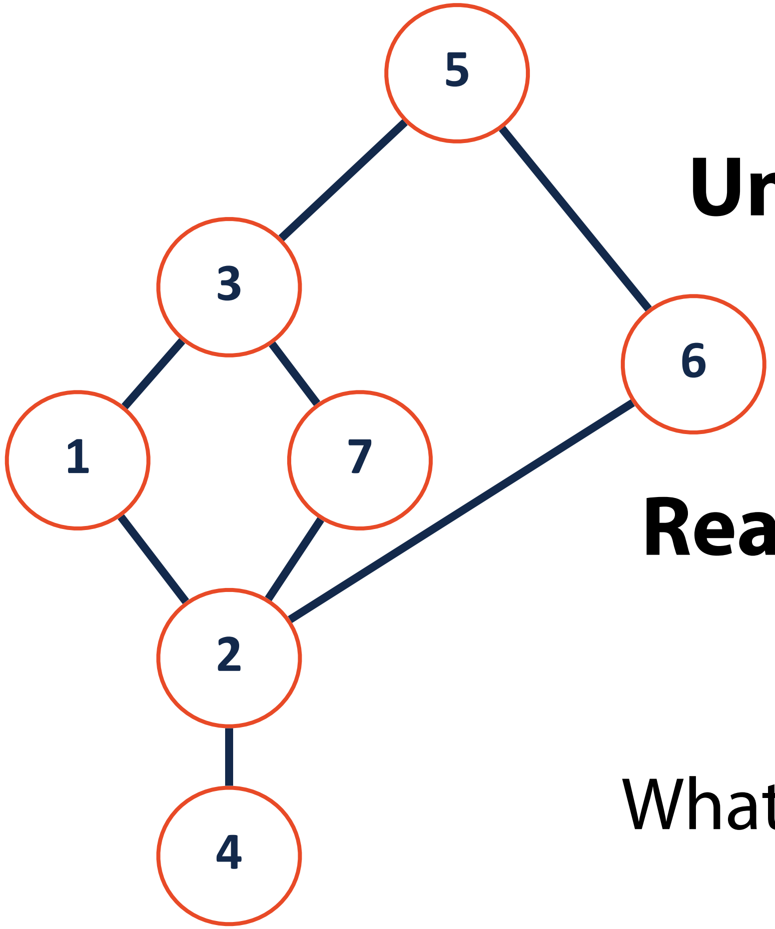
Cycle: A path from a node back to itself

What are some of the cycles in this graph?

Terminology Trivia: Every tree is a graph but not every graph is a tree

Graph Vocabulary

A graph may be **directed** or **undirected**



Directed: Edges are one way connections

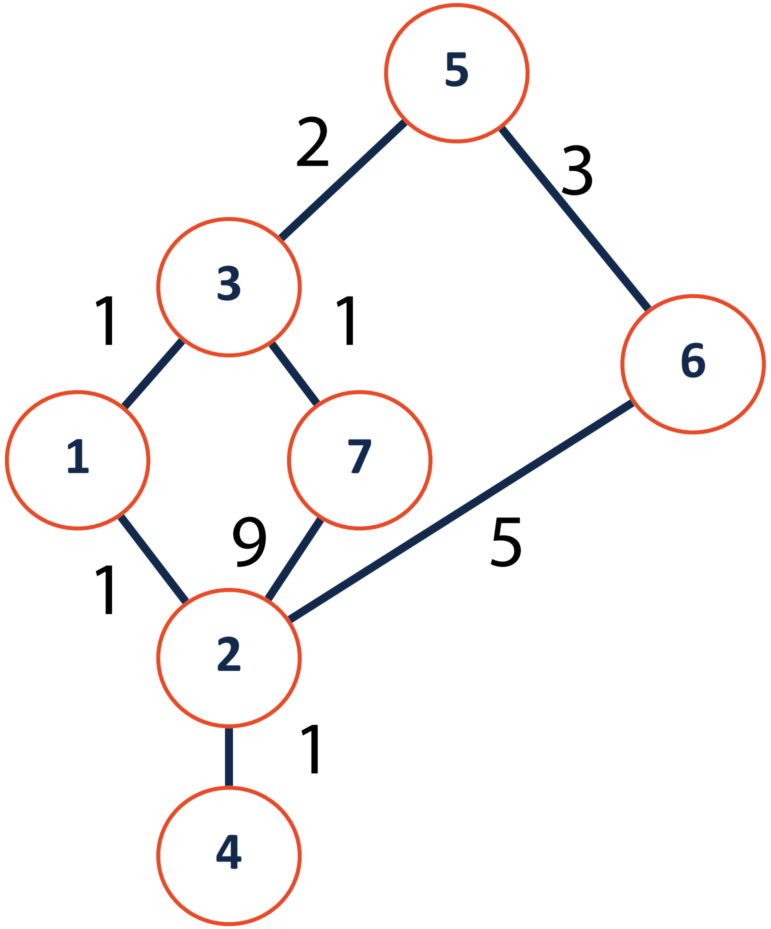
Undirected: Traversable in either direction

Reachability: v_2 is reachable from v_1 if there is a path from v_1 to v_2

What nodes are not reachable from 4?

Graph Vocabulary

A graph may be **weighted** or **unweighted**



Weights: A value associated with an edge

What is the shortest path from 4 to 5?

Graphs

Given a collection of individual DMs between individuals, you want to build a graph of connections in a social network.

What is a vertex?

What is an edge?

Are the edges directed or undirected?

Are the edges weighted or unweighted?

Graphs

Given a collection of roads between cities in Illinois, you want to build a graph of the transportation infrastructure in the state.

What is a vertex?

What is an edge?

Are the edges directed or undirected?

Are the edges weighted or unweighted?

Graphs

It is important to be able to describe the structure of a graph given input.

Some other common questions:

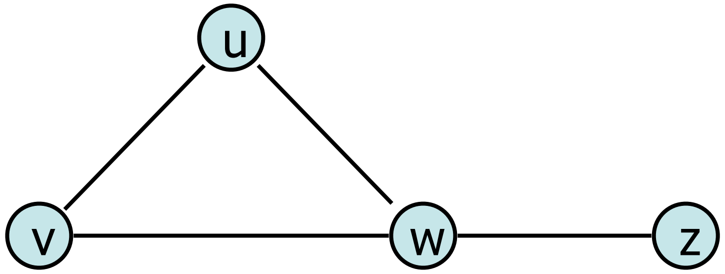
- Does your graph have cycles?
- What is the largest / smallest / average degree in your graph?
- What is the total number of edges?
- …

Of course, we also have to understand the graph as a **data structure**

Graph Implementation

What information do we need to store to fully define a graph?

Vertex:



Edge:

What information do we want to be able to find out quickly?

What operations do we want to prioritize?

Graph ADT

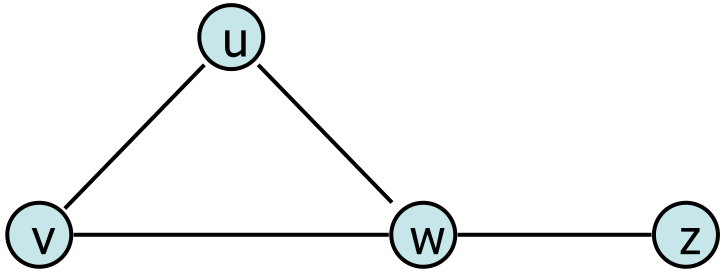
Find

Insert

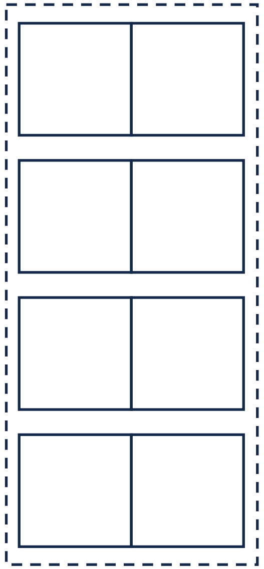
Remove

Graph Implementation: Edge List $|V| = n, |E| = m$

The equivalent of an 'unordered' data structure



Vertex Storage:



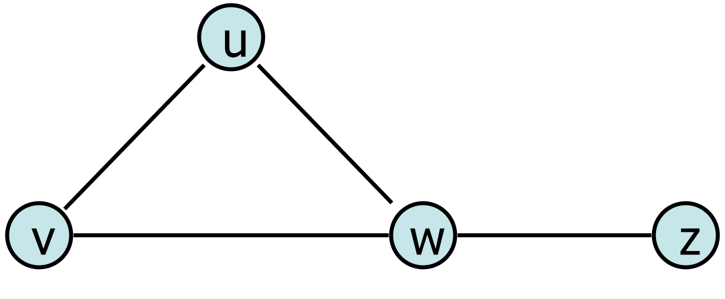
Edge Storage:

Graph Implementation: Edge List

$|V| = n, |E| = m$

The equivalent of an 'unordered' data structure

```
class edgeList:
    def __init__(self, inList):
        self.edges=[]
        for e in inList:
            v1 = e.split(" ")[0]
            v2 = e.split(" ")[1]
            self.edges.append( (v1, v2) )
```



Most graph inputs are line separated lists of edges:

u v
u w
v w
w z

Graph Implementation: Edge List $|V|= n, |E|= m$

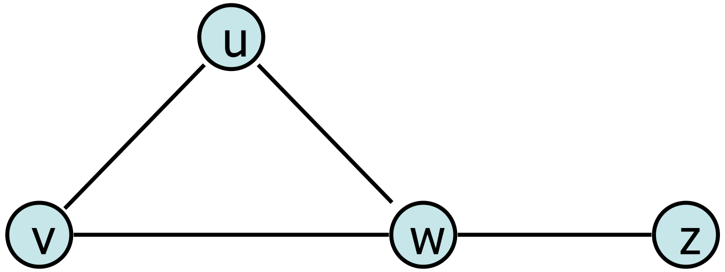
The equivalent of an 'unordered' data structure

```
class edgeList:
    def __init__(self, inList):
        self.edges=[]
        for e in inList:
            v1 = e.split(" ")[0]
            v2 = e.split(" ")[1]
            self.edges.append( (v1, v2) )
```

```
BA,1355,SIN,3316,LHR,507,,0,744 777
BA,1355,SIN,3316,MEL,3339,Y,0,744
TOM,5013,ACE,1055,BFS,465,,0,320
```

Airline
2-letter (IATA) or 3-letter (ICAO) code of the airline.
Airline ID
Unique OpenFlights identifier for airline (see Airline).
Source airport
3-letter (IATA) or 4-letter (ICAO) code of the source airport.
Source airport ID
Unique OpenFlights identifier for source airport (see Airport)
Destination airport
3-letter (IATA) or 4-letter (ICAO) code of the destination airport.
Destination airport ID
Unique OpenFlights identifier for destination airport (see Airport)
Codeshare
"Y" if this flight is a codeshare (that is, not operated by Airline, but another carrier), empty otherwise.
Stops
Number of stops on this flight ("0" for direct)
Equipment
3-letter codes for plane type(s) generally used on this flight, separated by spaces

Graph Implementation: Edge List



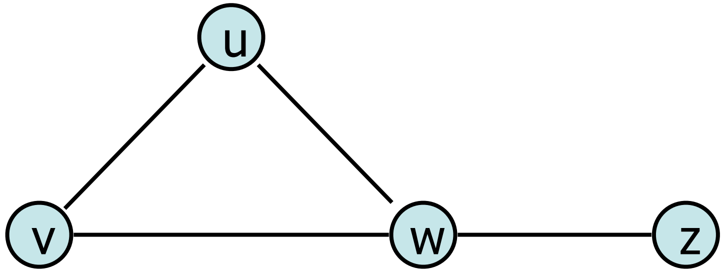
U	V
U	W
V	W
W	Z

getVertices()

getEdge(v)

areAdjacent(u, v)

Graph Implementation: Edge List



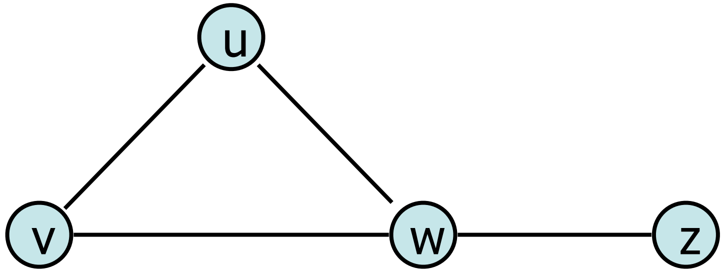
u
v
w
z

U	V
U	W
V	W
W	Z

insertVertex(v)

insertEdge(u, v)

Graph Implementation: Edge List

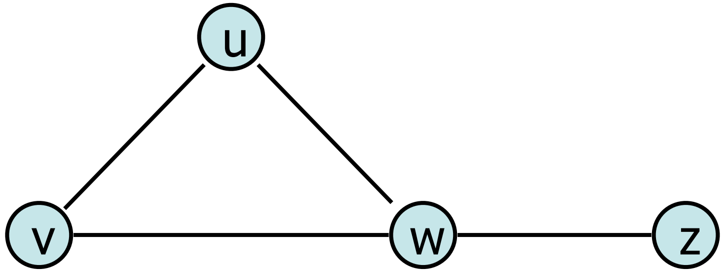


U	V
U	W
V	W
W	Z

Pros:

Cons:

Graph Implementation: Adjacency Matrix



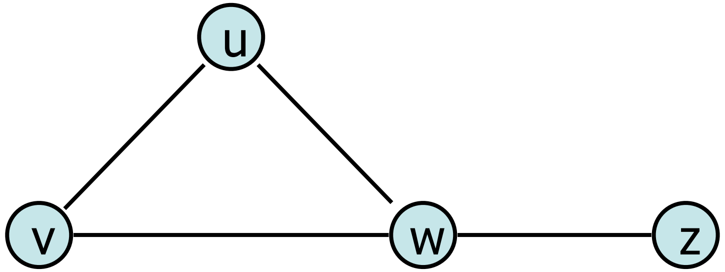
Vertex Storage:

Edge Storage:

u	
v	
w	
z	

	u	v	w	z
u				
v				
w				
z				

Graph Implementation: Adjacency Matrix

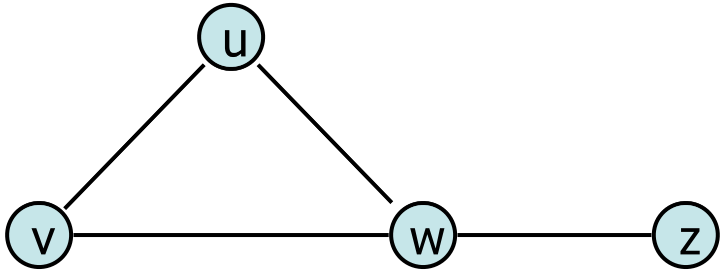


getVertices():

U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Graph Implementation: Adjacency Matrix

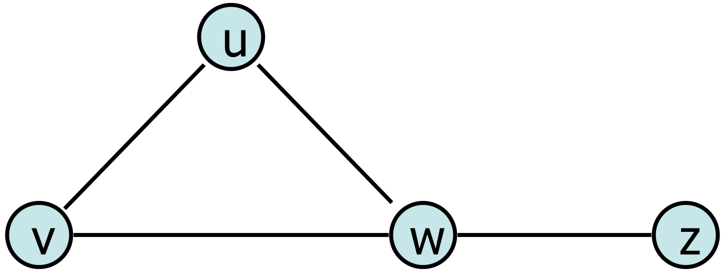


getEdges(v):

U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Graph Implementation: Adjacency Matrix

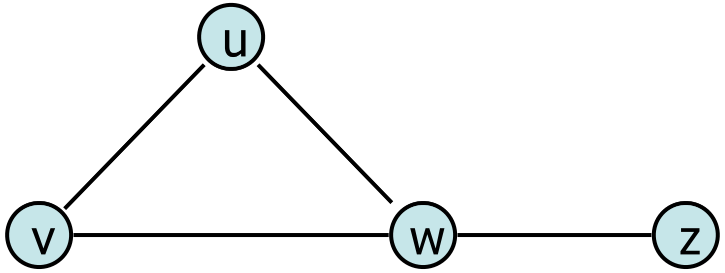


areAdjacent(u, v):

U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Graph Implementation: Adjacency Matrix

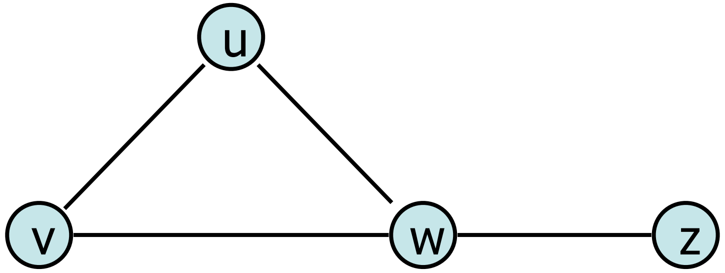


insertVertex(v):

U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Graph Implementation: Adjacency Matrix

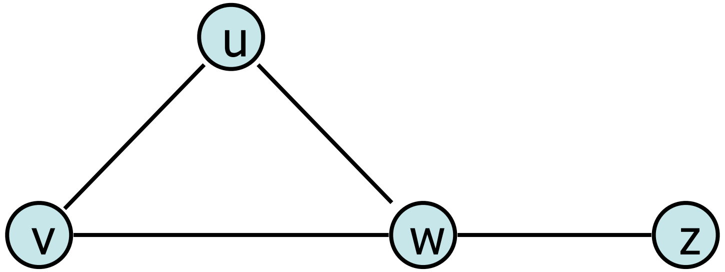


insertEdge(u, v):

U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Graph Implementation: Adjacency Matrix



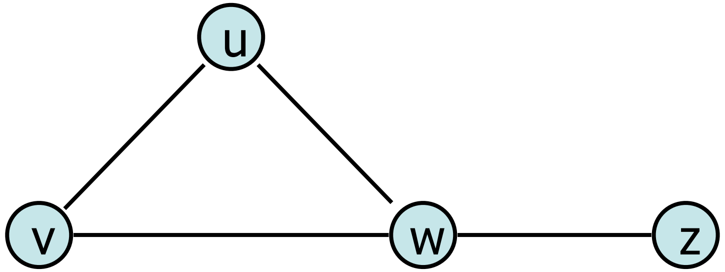
U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

removeVertex(v):

removeEdge(u, v):

Graph Implementation: Adjacency Matrix



U	0
V	1
W	2
Z	3

	u	v	w	z
u	0	1	1	0
v	1	0	1	0
w	1	1	0	1
z	0	0	1	0

Pros:

Cons: