Data Structures

Graph Fundamentals

CS 225
Brad Solomon

October 21, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

Define graph vocabulary

Discuss graph implementation and storage strategies

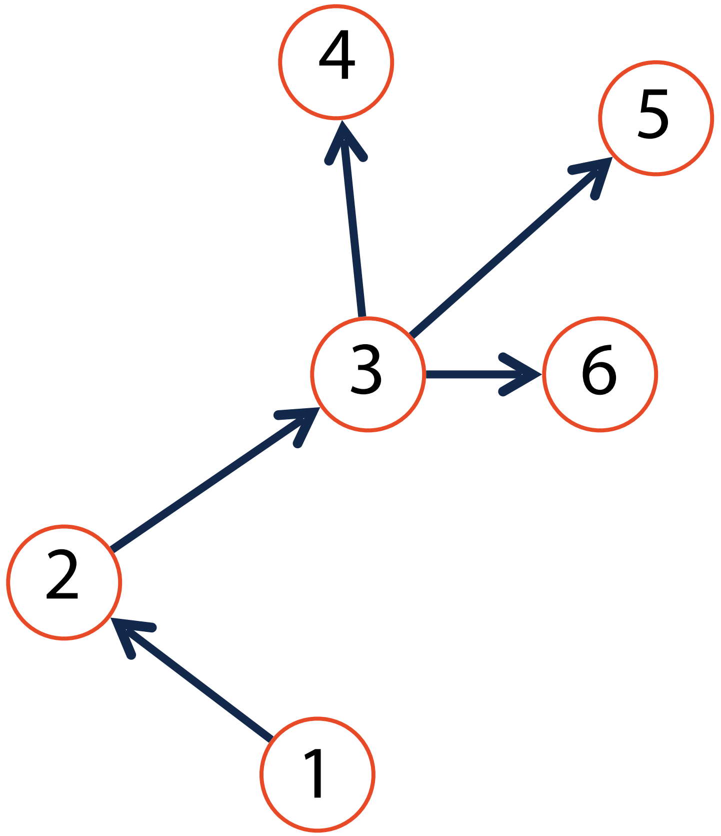
Whats next?

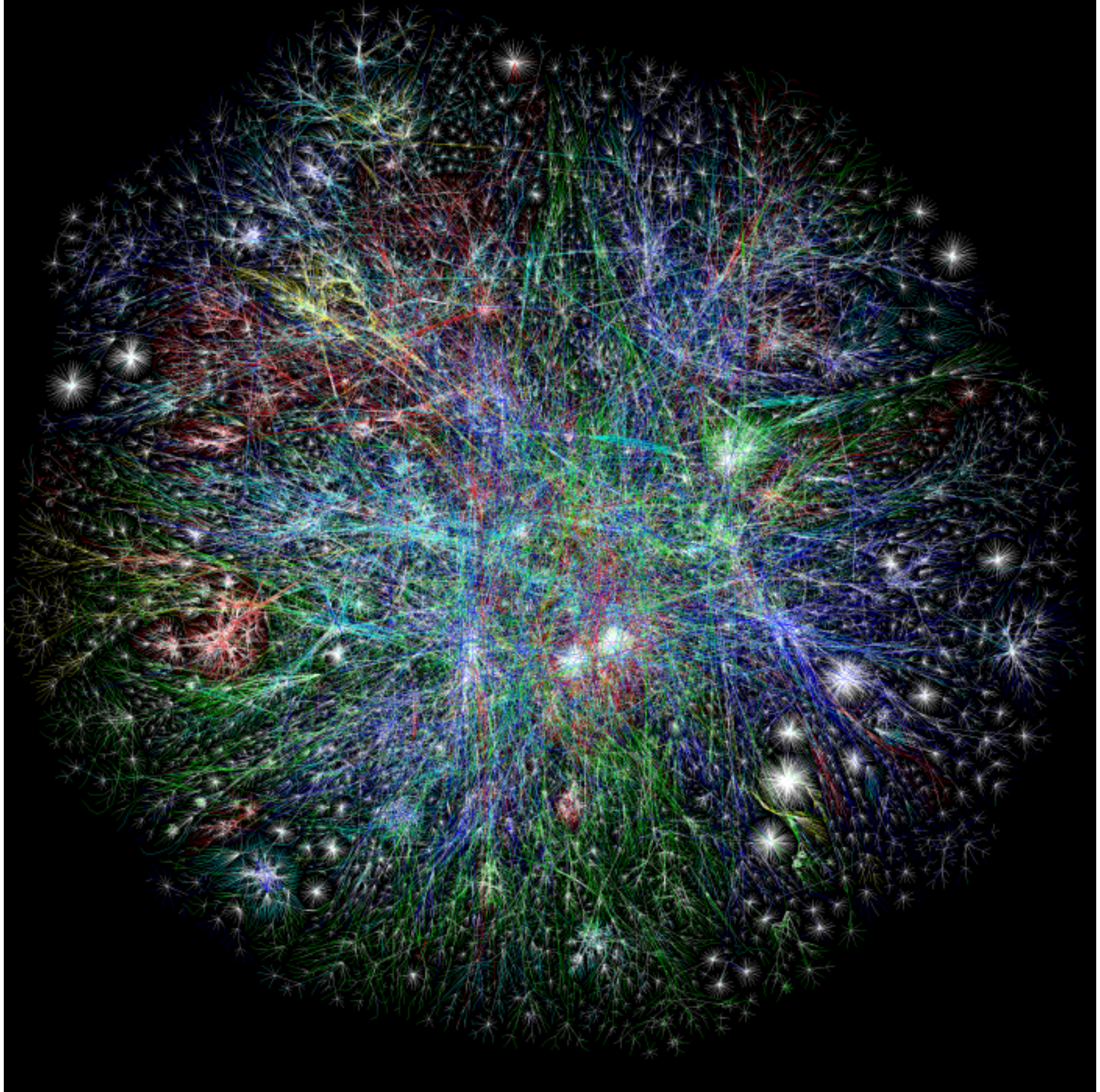
A non-linear data structure defined recursively as a collection of nodes where each node contains a value and zero or more connected nodes.

(In CS 225) a tree is also:

1) Acyclic — contains no cycles

2) Rooted — root node connected to all nodes



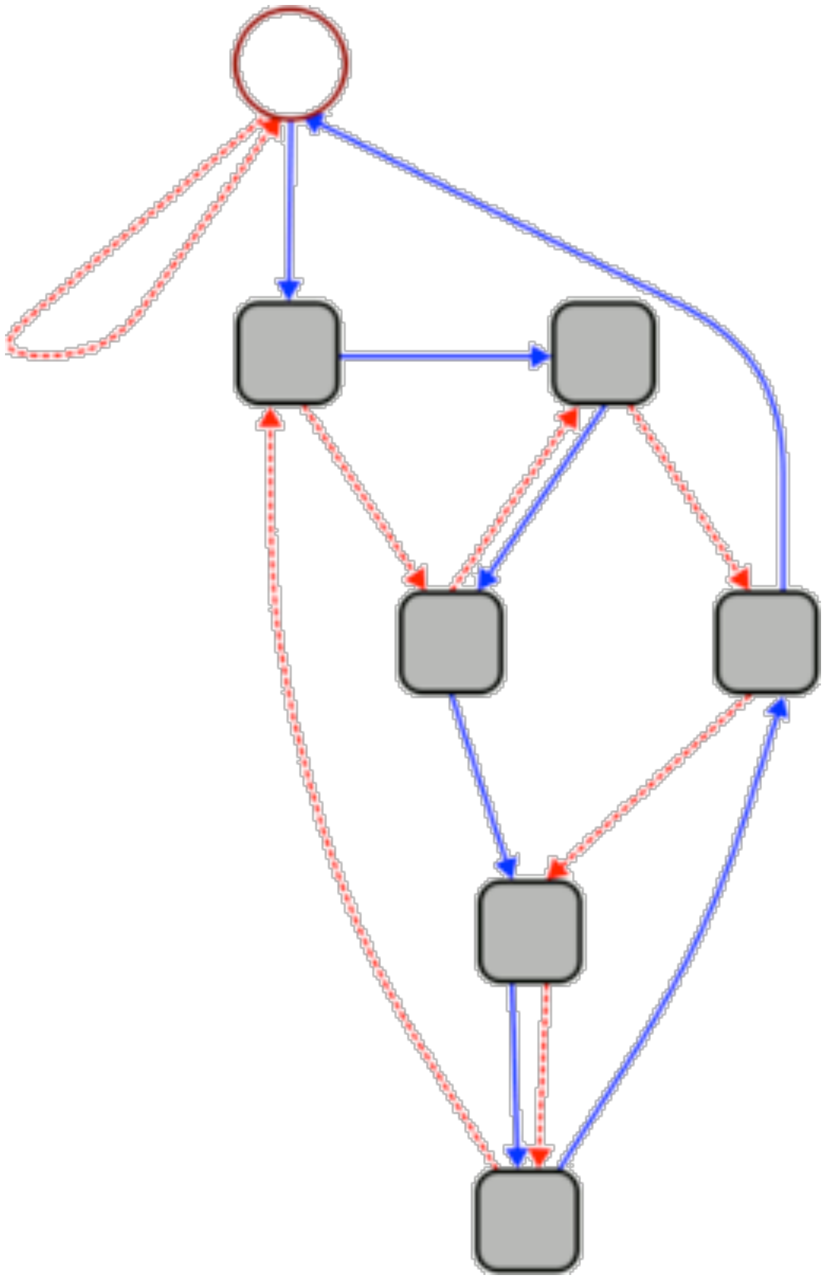


Nodes: Routers and servers

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.

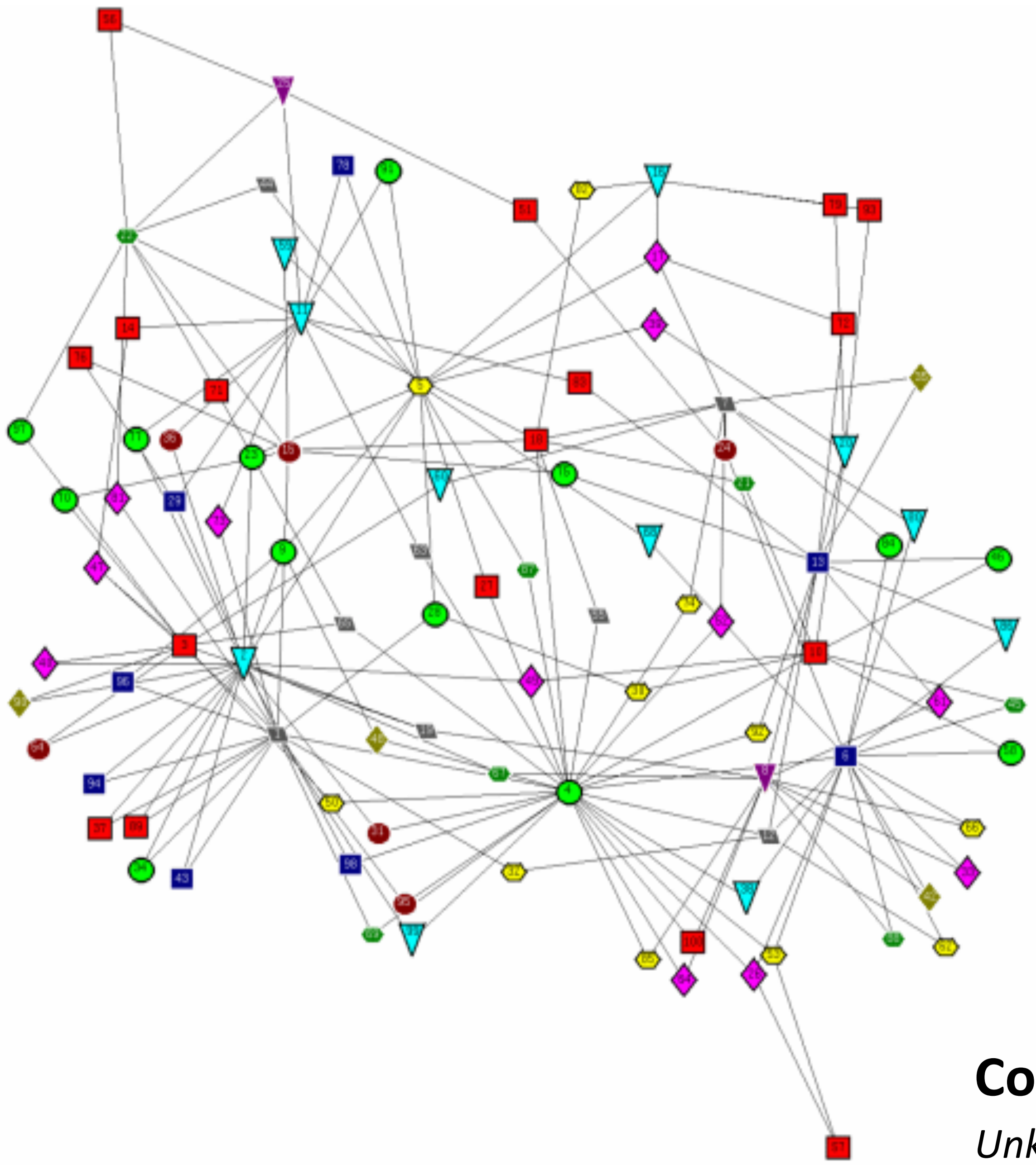
1. Start at the circle node at the top.
2. For each digit **d** in the given number, follow **d** blue (solid) edges in succession. As you move from one digit to the next, follow **1** red (dashed) edge.
3. If you end up back at the circle node, your number is divisible by 7.

3703

"Rule of 7"

Unknown Source

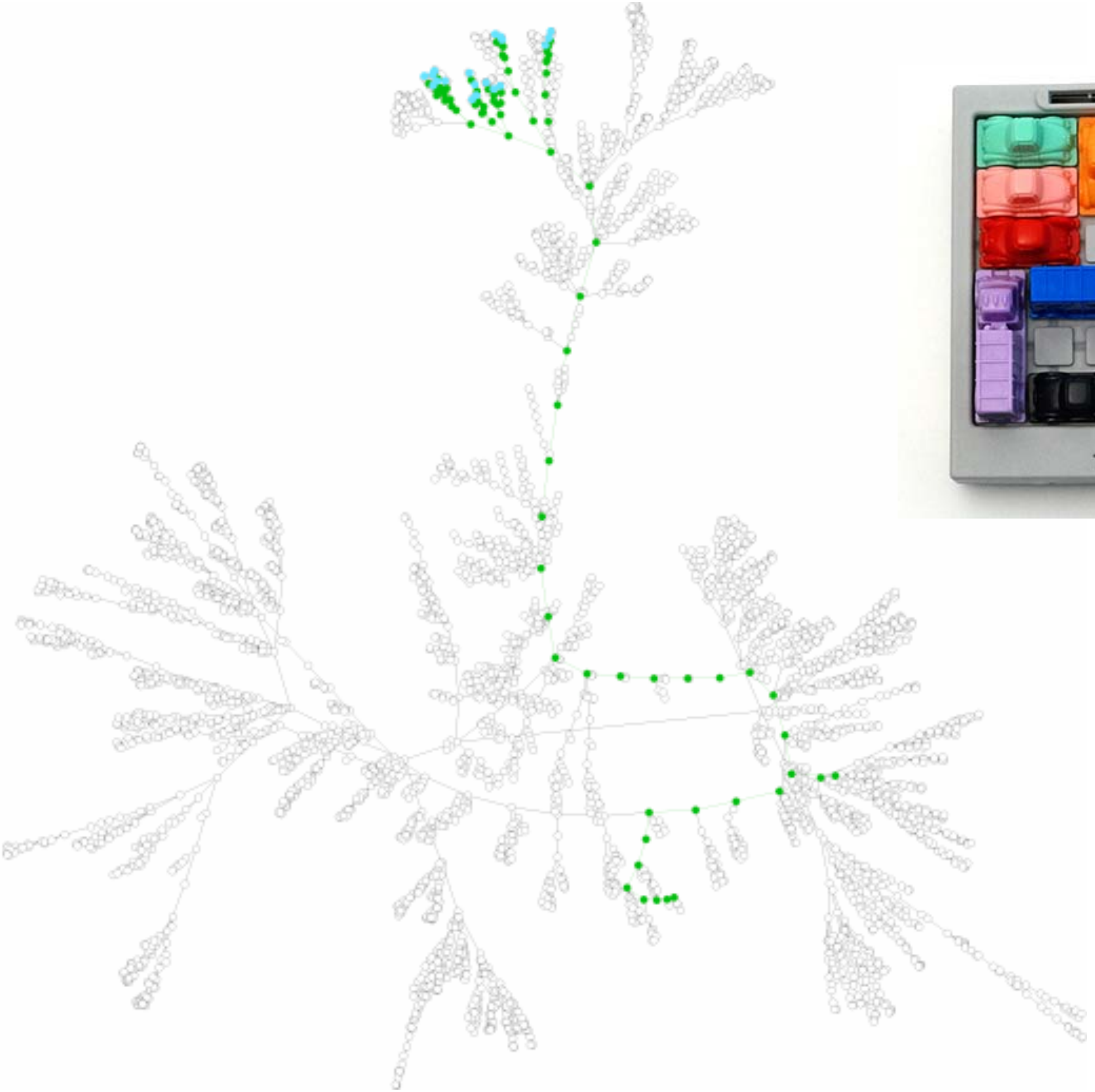
Presented by Cinda Heeren, 2016



Conflict-Free Final Exam Scheduling Graph

Unknown Source

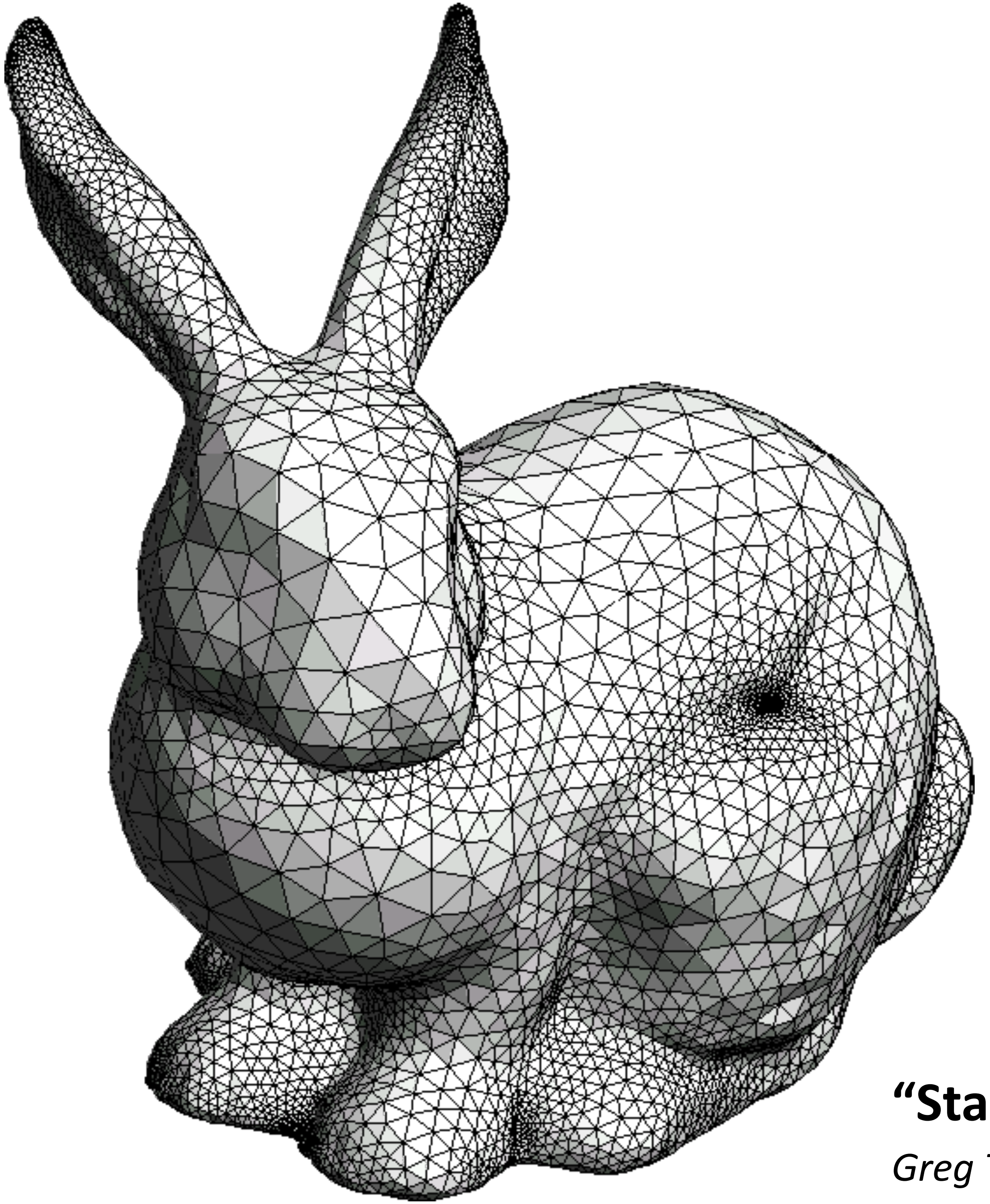
Presented by Cinda Heeren, 2016



“Rush Hour” Solution

Unknown Source

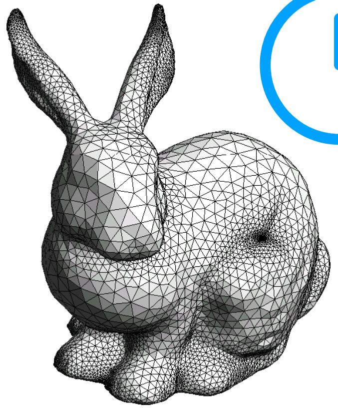
Presented by Cinda Heeren, 2016



“Stanford Bunny”

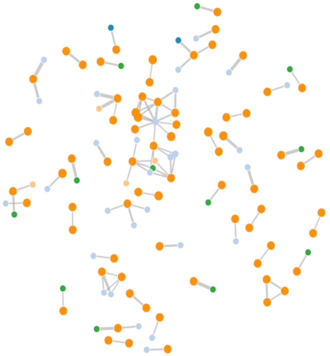
Greg Turk and Mark Levoy (1994)

Graphs

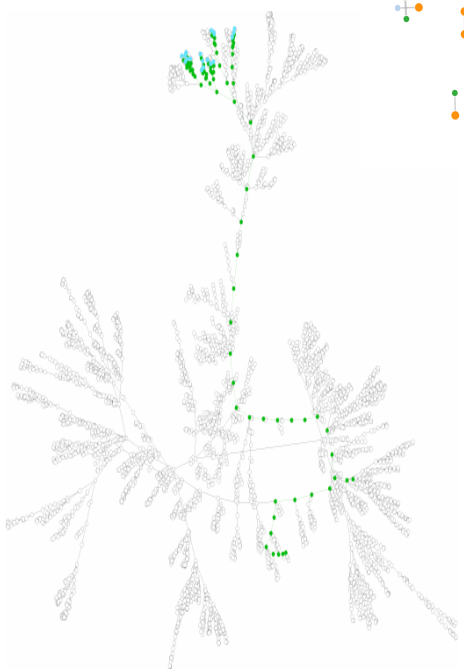


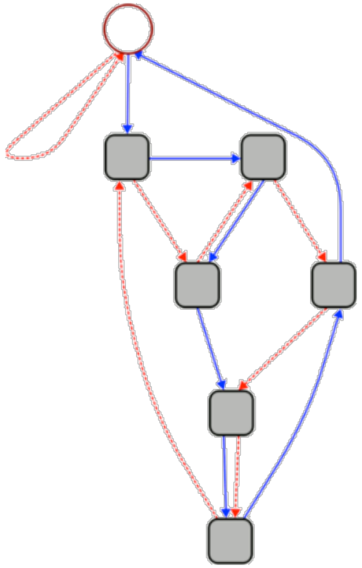
To study all of these structures:

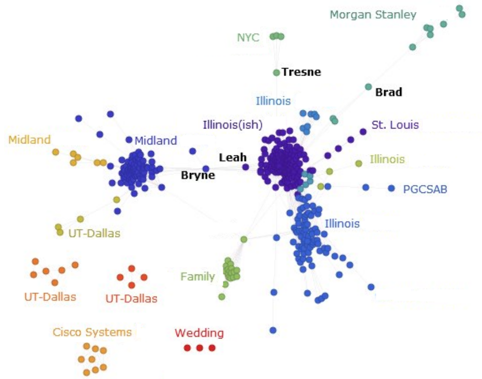
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms

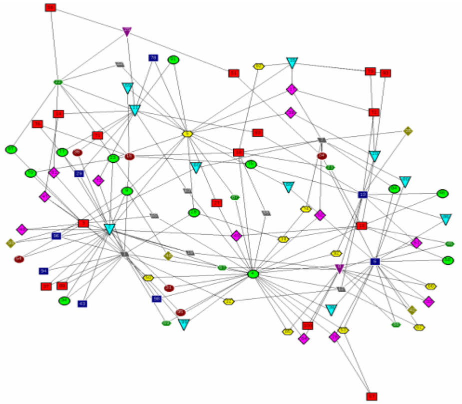


HAMLET TROILUS AND CRESSIDA





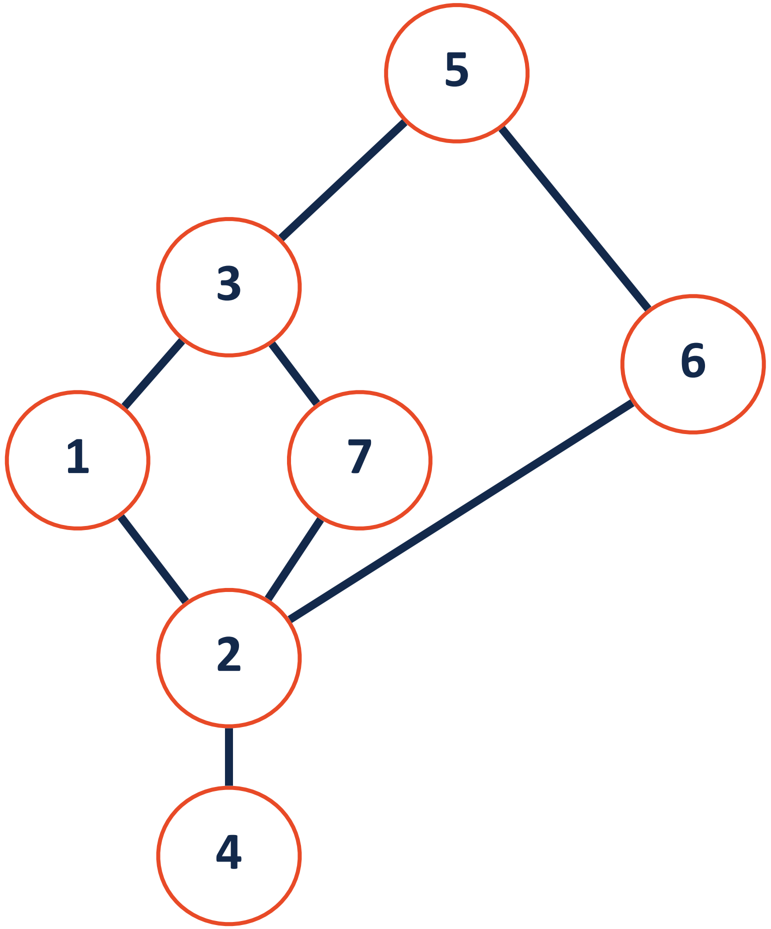




Graph Vocabulary

$G = (V, E)$

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

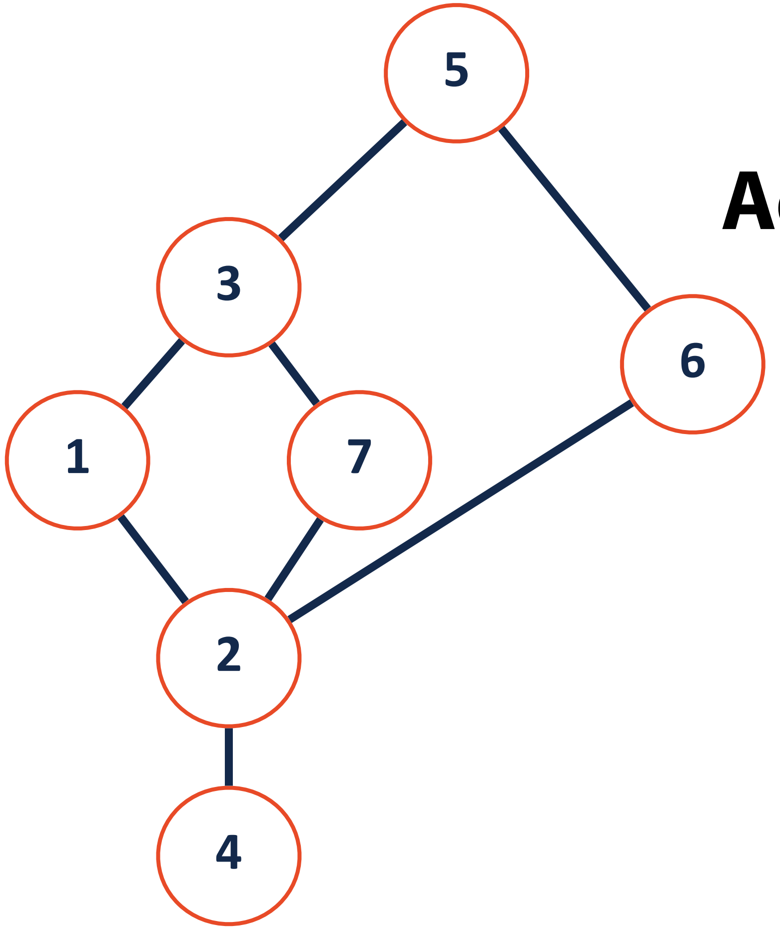


Vertex: Nodes of the graph

Edges: The connections between nodes

Defined by two endpoints

Graph Vocabulary



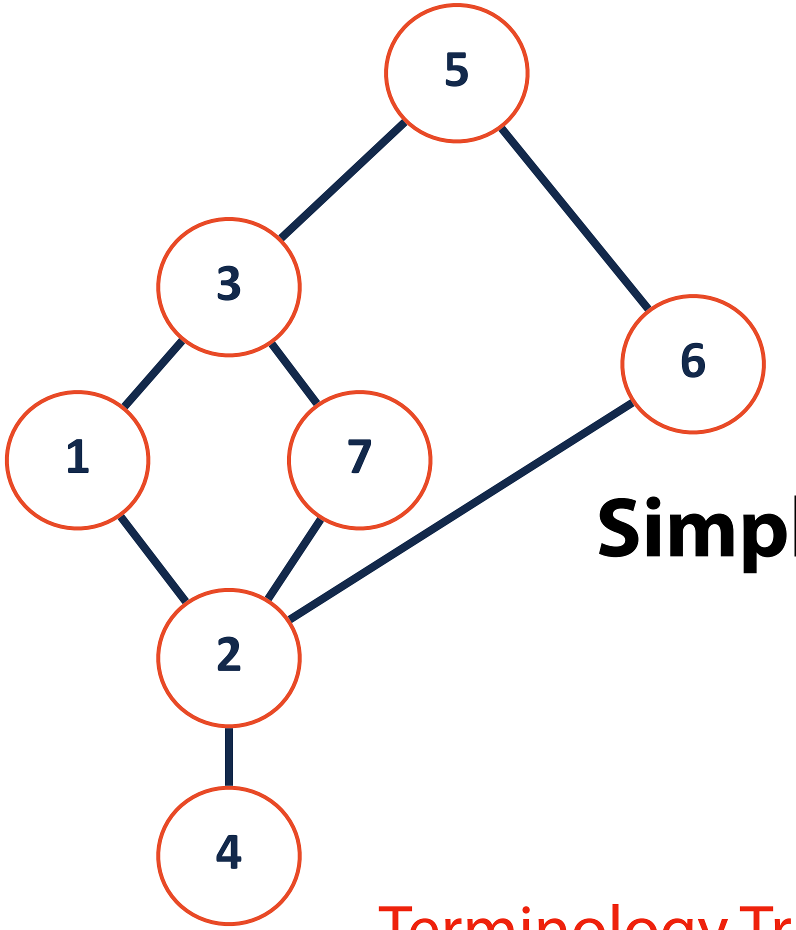
Degree: # of edges touching a vertex

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

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

Graph Vocabulary

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



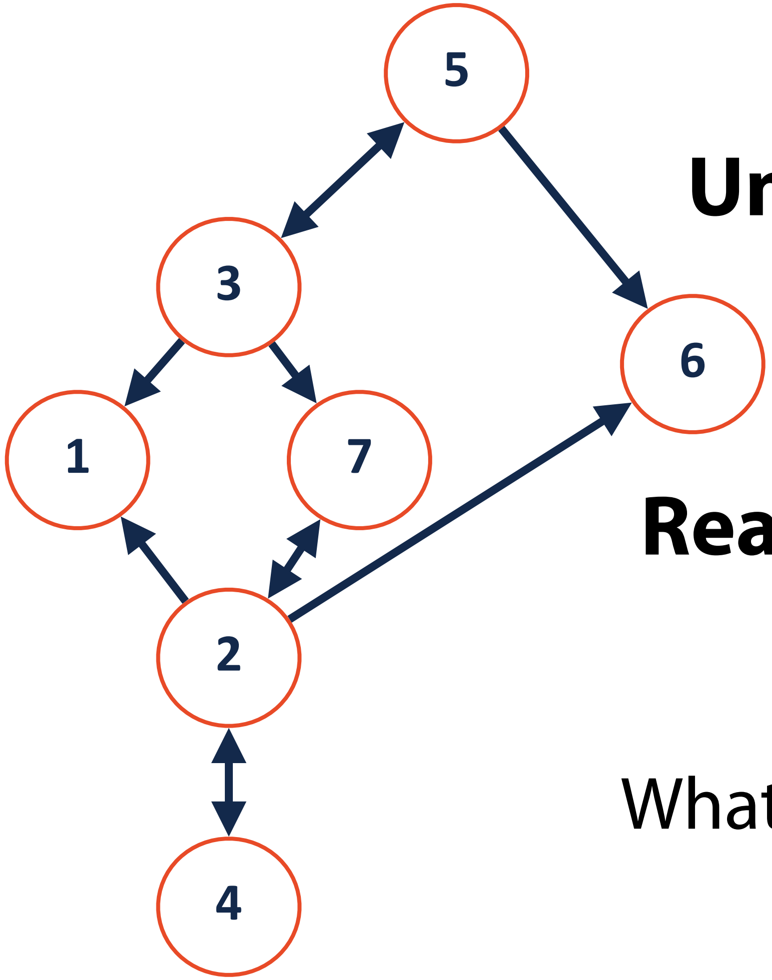
Cycle: A path from a node to itself

Simple Graph: No self-loops or multi-edges

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

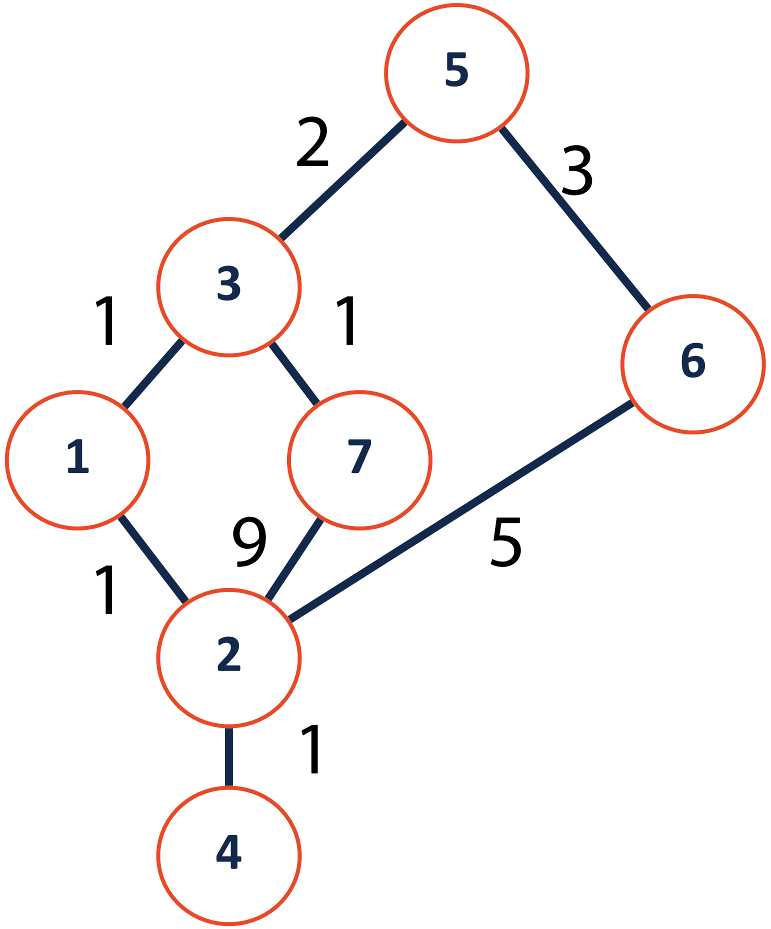
Undirected: Traversable in either direction

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

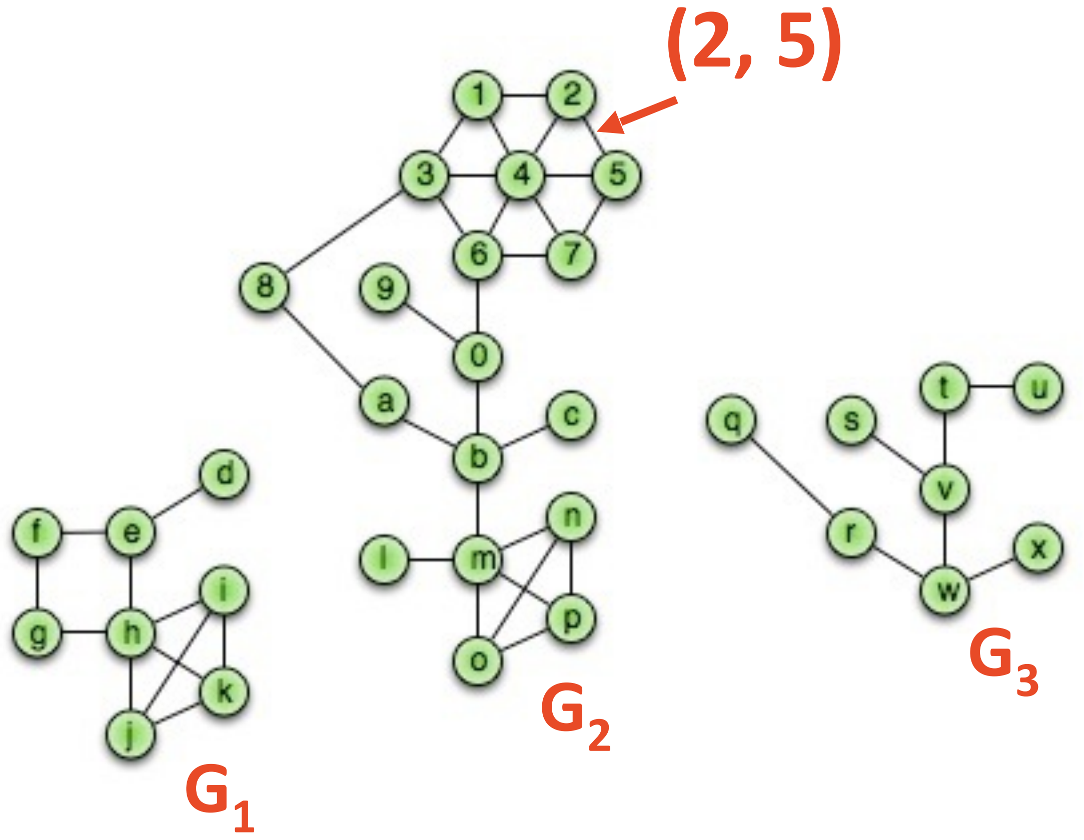
What is the shortest path from 4 to 5?

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Subgraph(G):

$G' = (V', E')$:

$V' \in V$, $E' \in E$, and

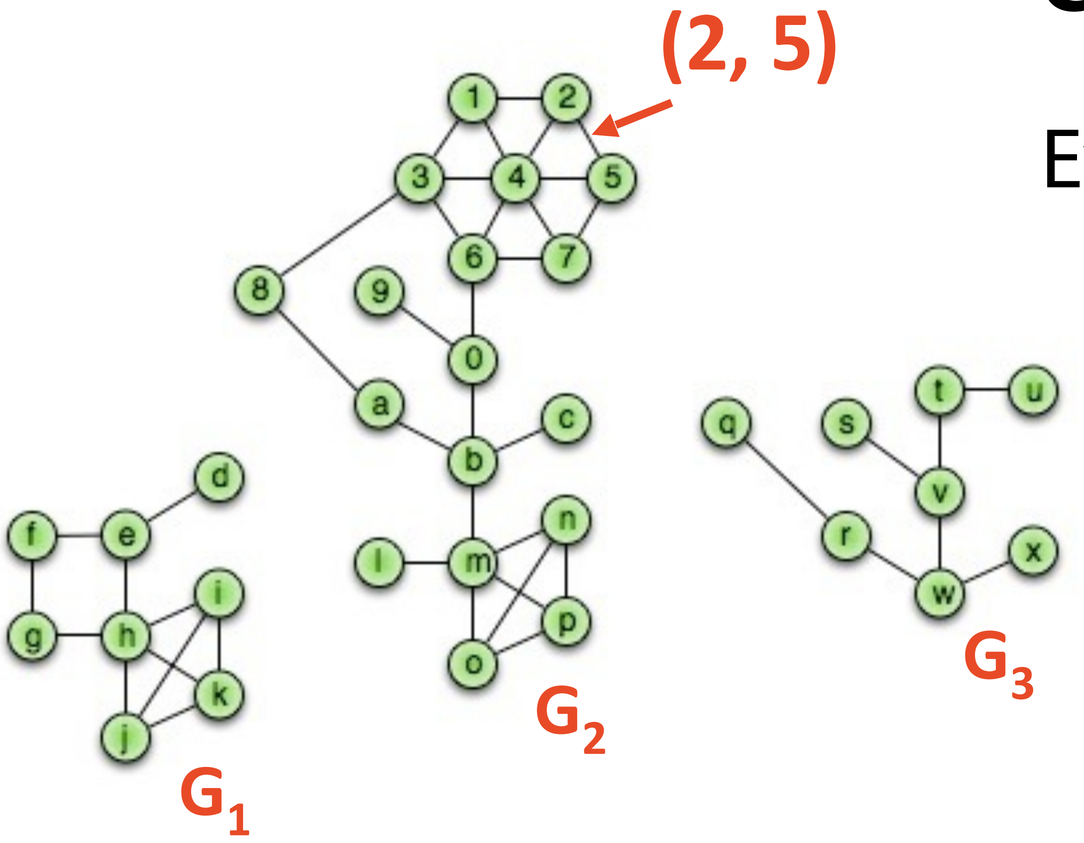
$(u, v) \in E' \rightarrow u \in V', v \in V'$

Graph Vocabulary

G = (V, E)

|V| = n

|E| = m



Subgraph(G):

G' = (V', E'):

V' ∈ V, E' ∈ E, and

(u, v) ∈ E' → u ∈ V', v ∈ V'

Complete Subgraph:

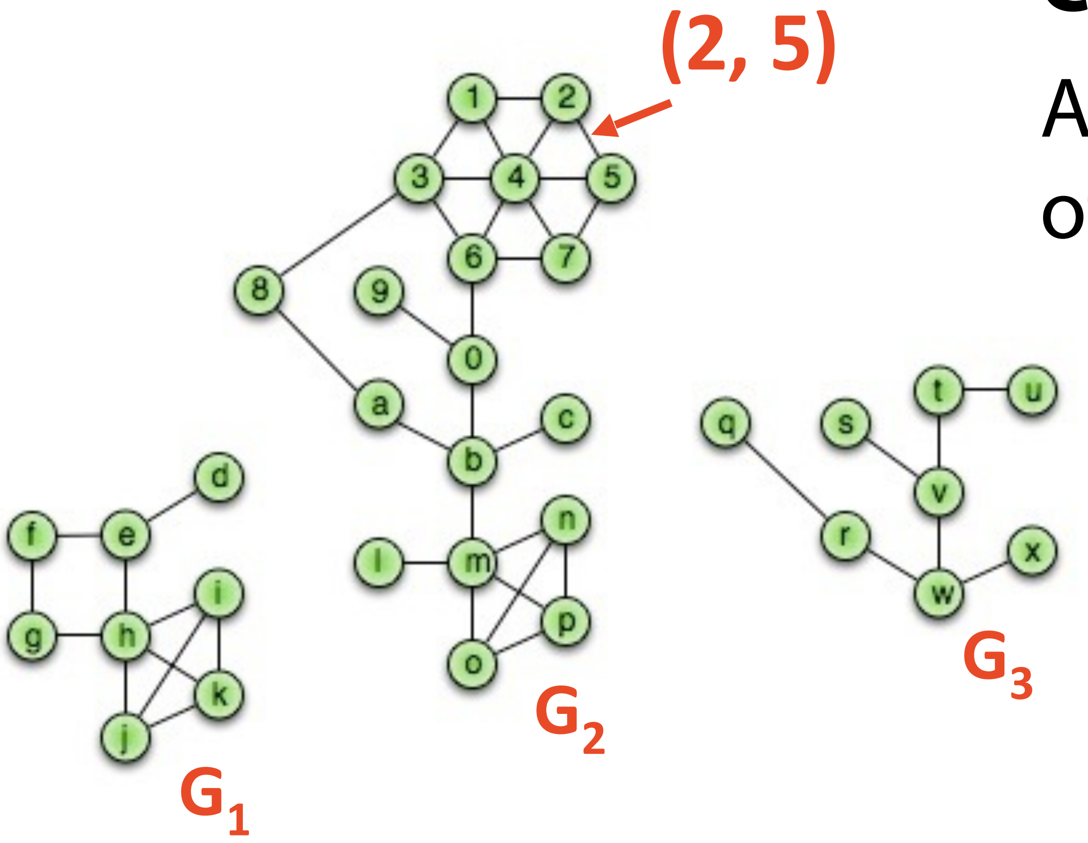
Every pair of vertices are adjacent

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Subgraph(G):

$G' = (V', E')$:

$V' \in V$, $E' \in E$, and

$(u, v) \in E' \rightarrow u \in V', v \in V'$

Connected Subgraph:

A path exists between every pair of vertices

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$

Subgraph(G):

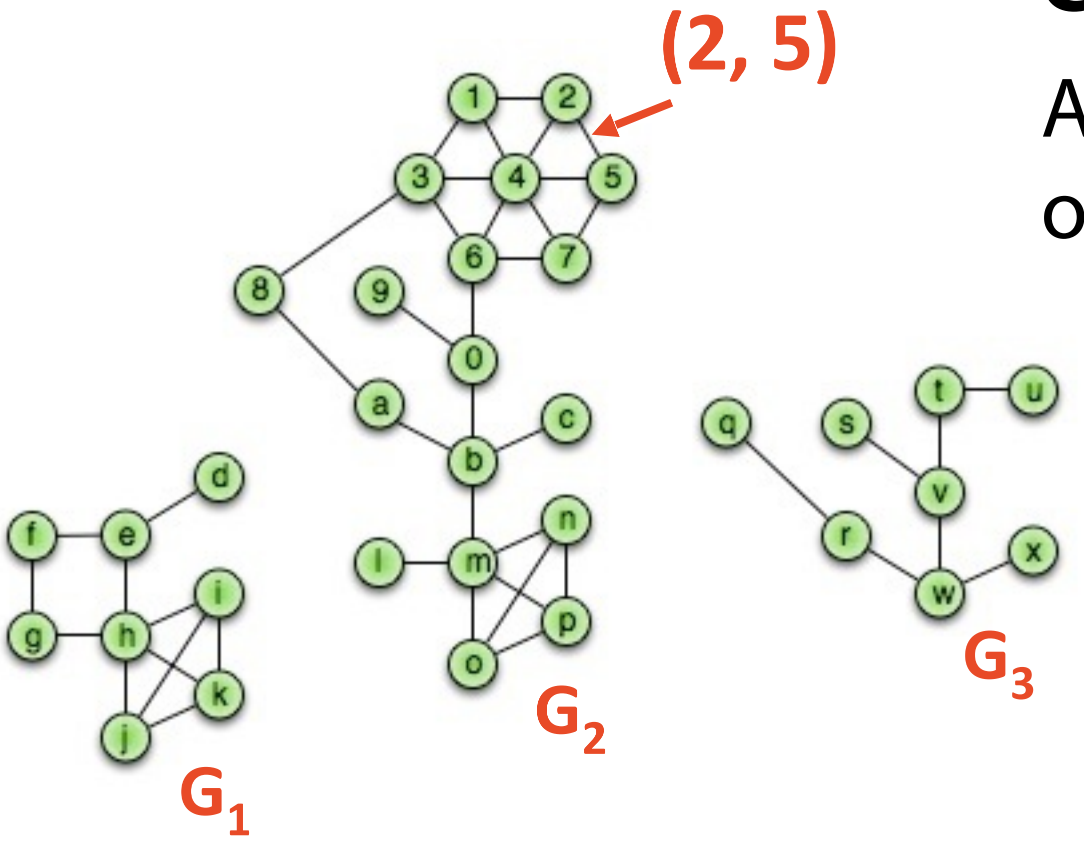
$G' = (V', E')$:

$V' \in V$, $E' \in E$, and

$(u, v) \in E' \rightarrow u \in V', v \in V'$

Connected Subgraph:

A path exists between every pair of vertices



Connected Components:

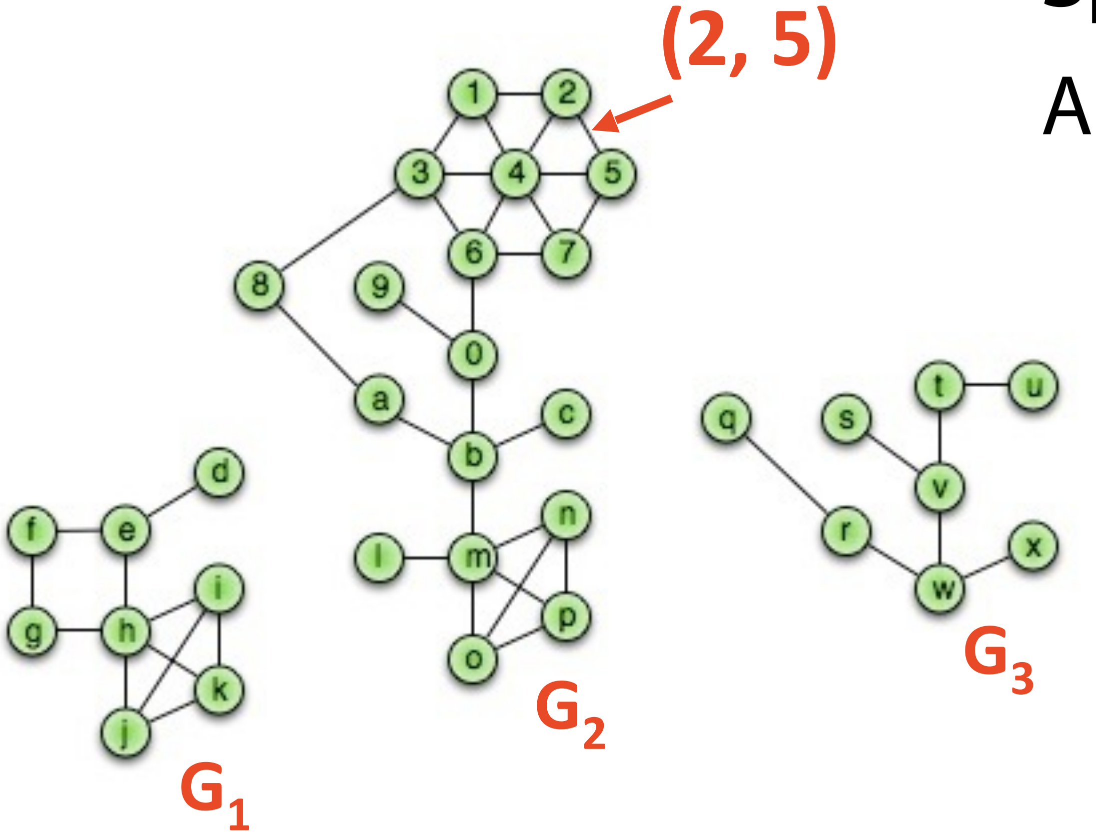
A connected subgraph that is not part of a larger subgraph

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Subgraph(G):

$G' = (V', E')$:

$V' \in V$, $E' \in E$, and

$(u, v) \in E' \rightarrow u \in V', v \in V'$

Spanning Tree:

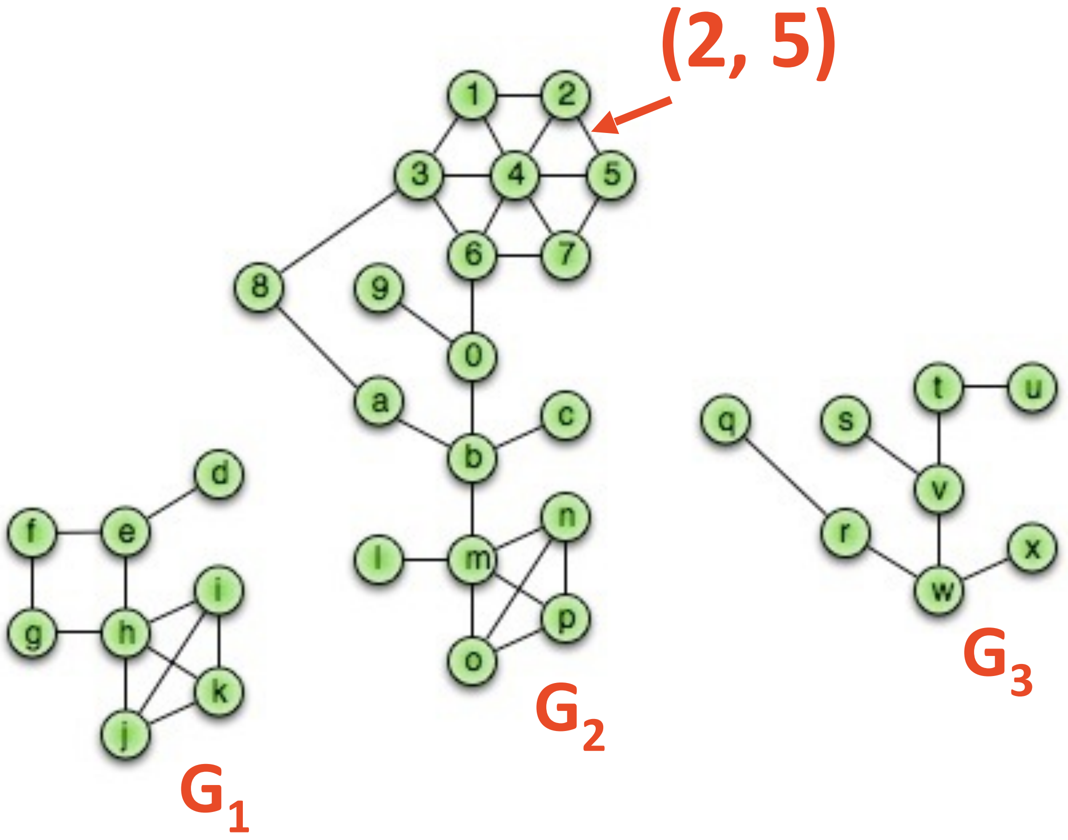
A connected graph with no cycles

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Graph Terminology is very important!

Degree

Weight

Direction

Adjacency

Complete

Connected

Acyclic

Spanning

And more…

Running times are often reported by **n**, the number of vertices, but often depend on **m**, the number of edges.

Whats the relationship between **n** and **m**?

Minimum Edges:

Unconnected Graph:

Connected (Simple) Graph:

Maximum Edges:

Connected (Simple) Graph:

$$\sum_{v \in V} deg(v) =$$

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:

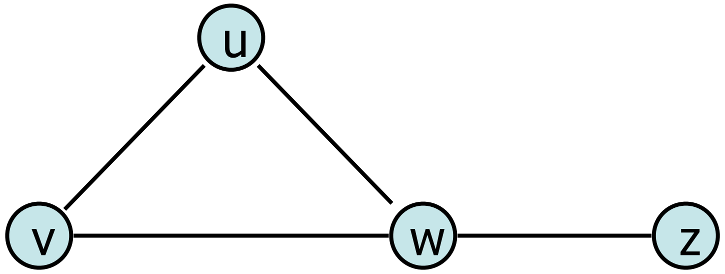
- Does your graph have cycles?
- What is the largest / smallest / average degree in your graph?
- What is the total number of edges?
- …

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:

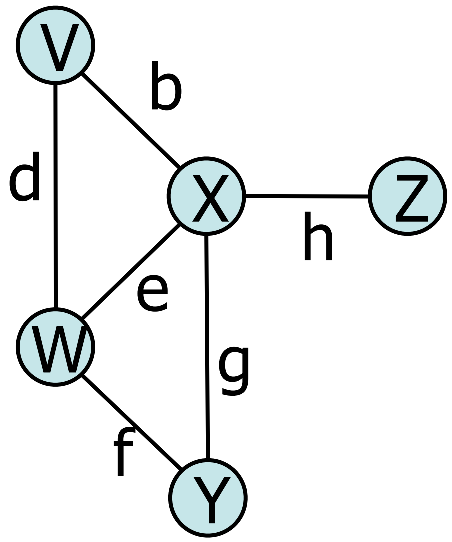
What information do we want to be able to find out quickly?

What operations do we want to prioritize?

Graph ADT

Data:

- **Vertices**
- **Edges**
- **Some data structure maintaining the structure between vertices and edges.**



Functions:

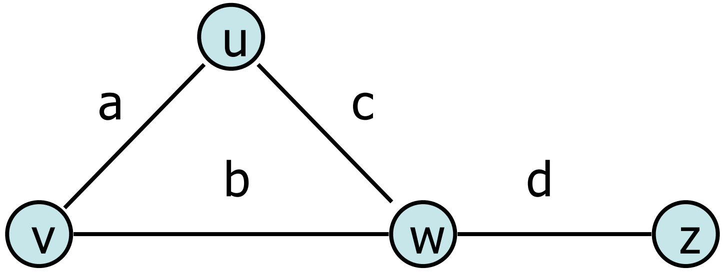
- **insertVertex(K key);**
- **insertEdge(Vertex v1, Vertex v2, K key);**

- **removeVertex(Vertex v);**
- **removeEdge(Vertex v1, Vertex v2);**

- **getEdges(Vertex v);**
- **areAdjacent(Vertex v1, Vertex v2);**

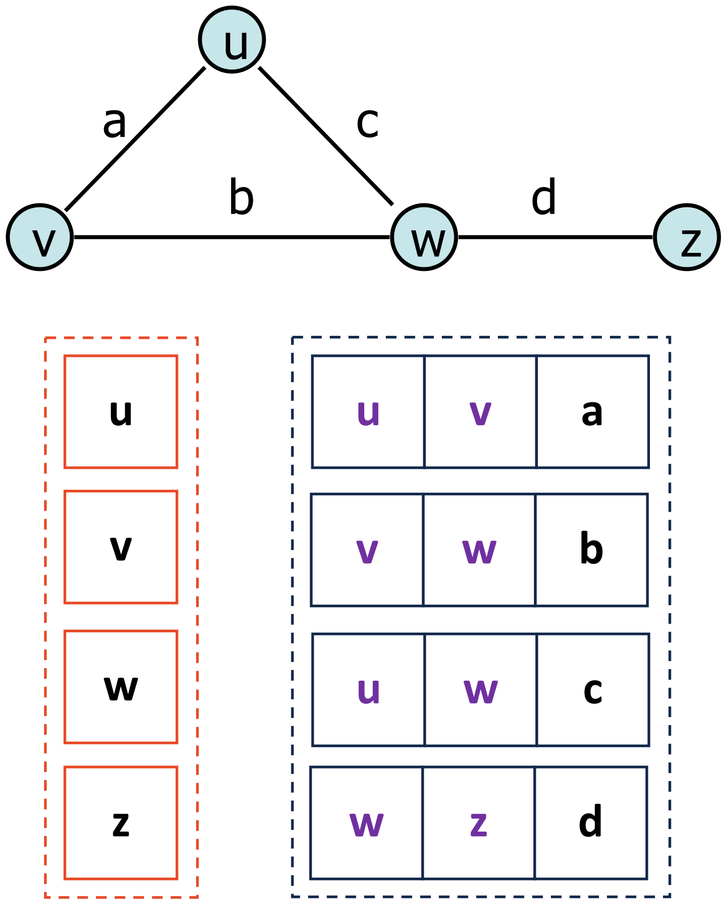
- **origin(Edge e);**
- **destination(Edge e);**

Graph Implementation Idea



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

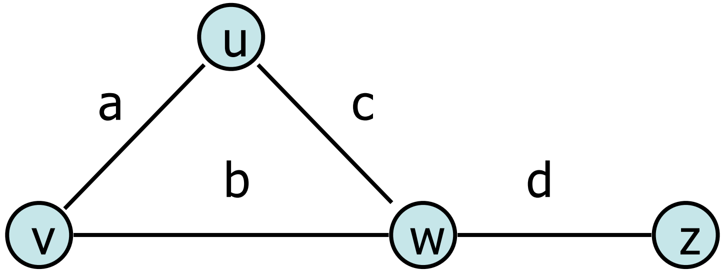
The equivalent of an 'unordered' data structure



Vertex Storage:

Edge Storage:

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



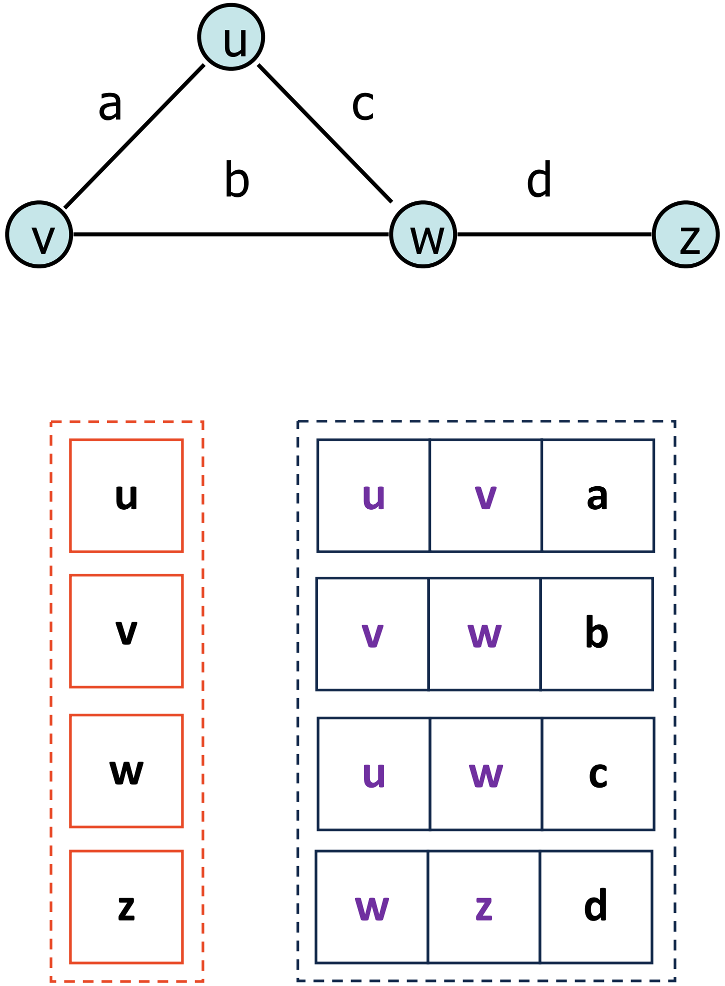
u
v
w
z

u	v	a
v	w	b
u	w	c
w	z	d

getEdges(Vertex v)

areAdjacent(Vertex v1, Vertex v2)

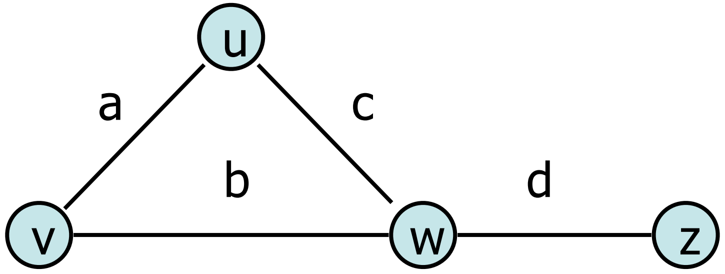
Graph Implementation: Edge List |V|= n, |E|= m



insertVertex(K key)

removeVertex(Vertex v)

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



u
v
w
z

u	v	a
v	w	b
u	w	c
w	z	d

insertEdge(Vertex v1, Vertex v2, K key)

removeEdge(Vertex v1, Vertex v2)

Graph Implementation: Edge List

Pros:

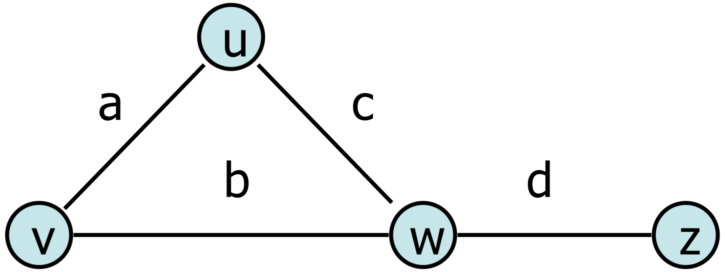
Cons:

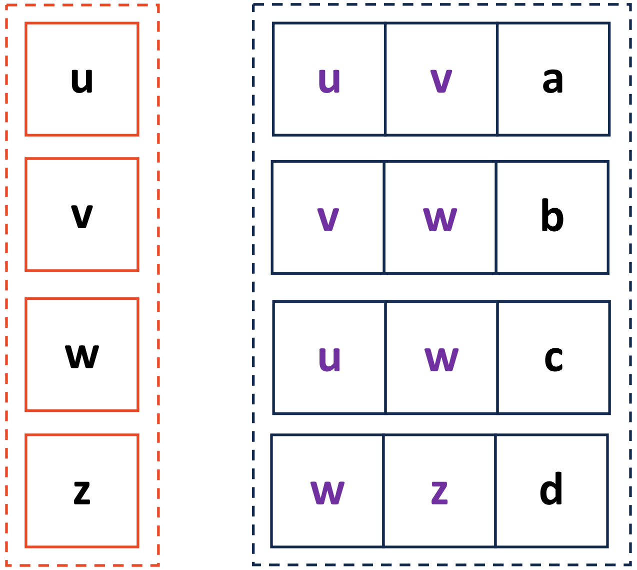
Graph Implementation: Brainstorming better

What operations might I want to do very quickly?

What modifications might allow me to do these things faster?

Graph Implementation: Adjacency Matrix





	u	v	w	z
u				
v				
w				
z				