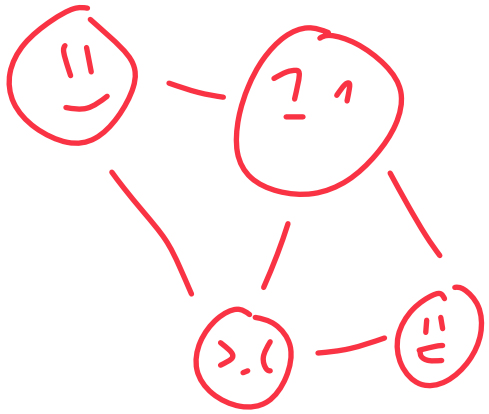
Data Structures

Graph Fundamentals

CS 225

Brad Solomon

October 21, 2024



Department of Computer Science

Learning Objectives

Define graph vocabulary

Discuss graph implementation and storage strategies

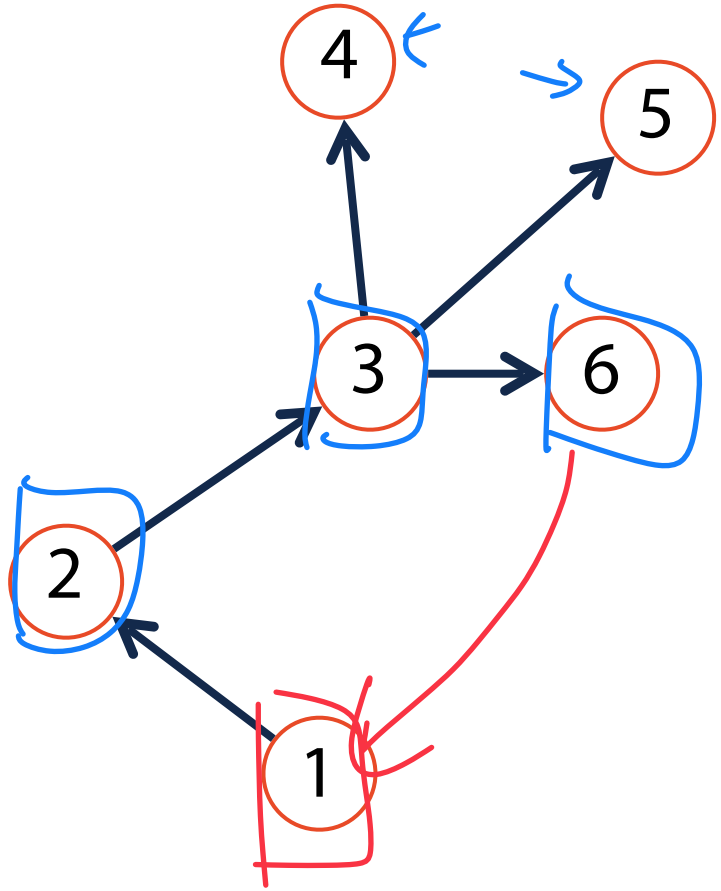
Whats next?

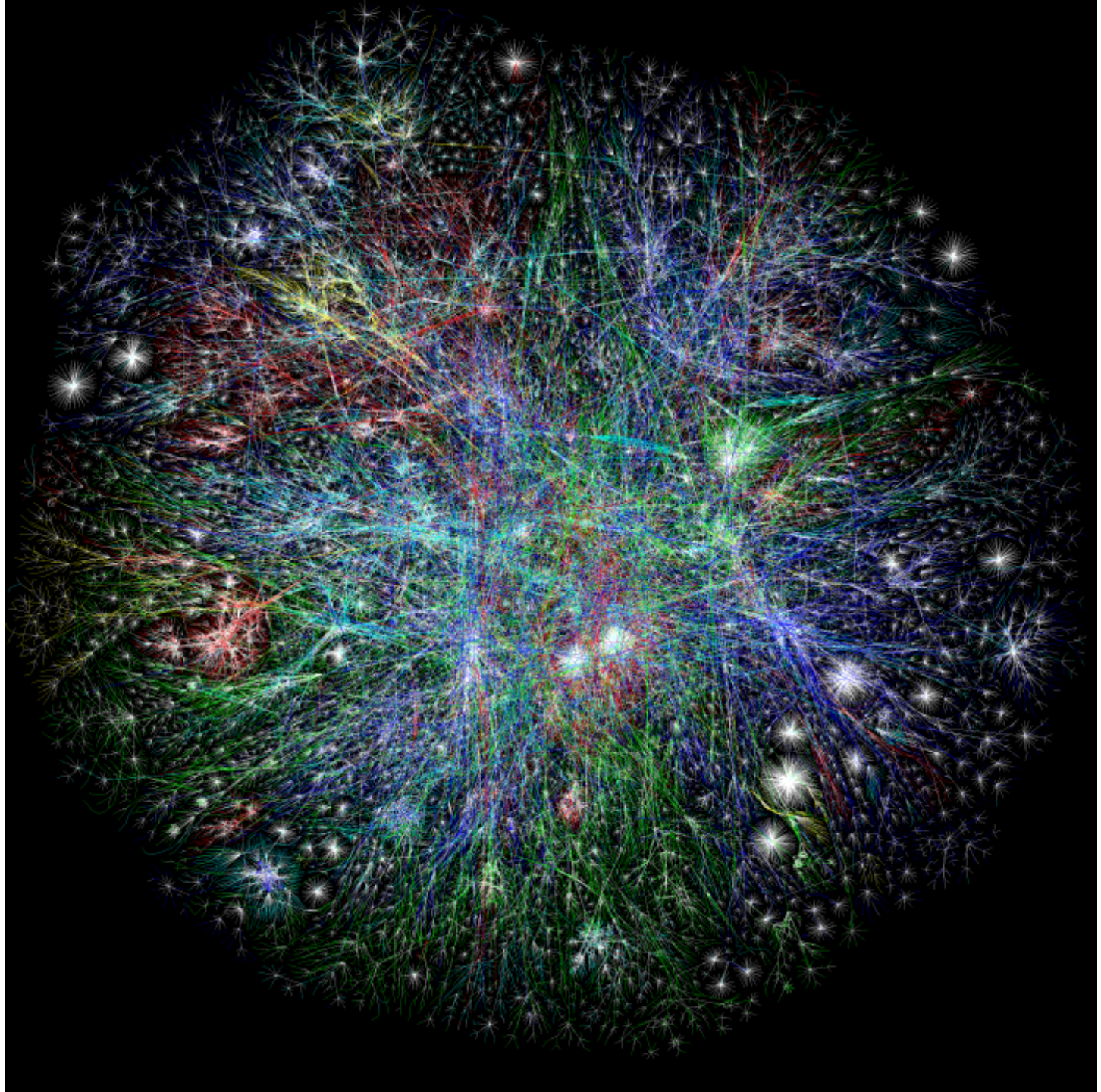
A non-linear data structure defined recursively as a collection of nodes where each node contains a value and zero or more connected nodes.

(In CS 225) a tree is also:

1) Acyclic — contains no cycles

2) Rooted — root node connected to all nodes





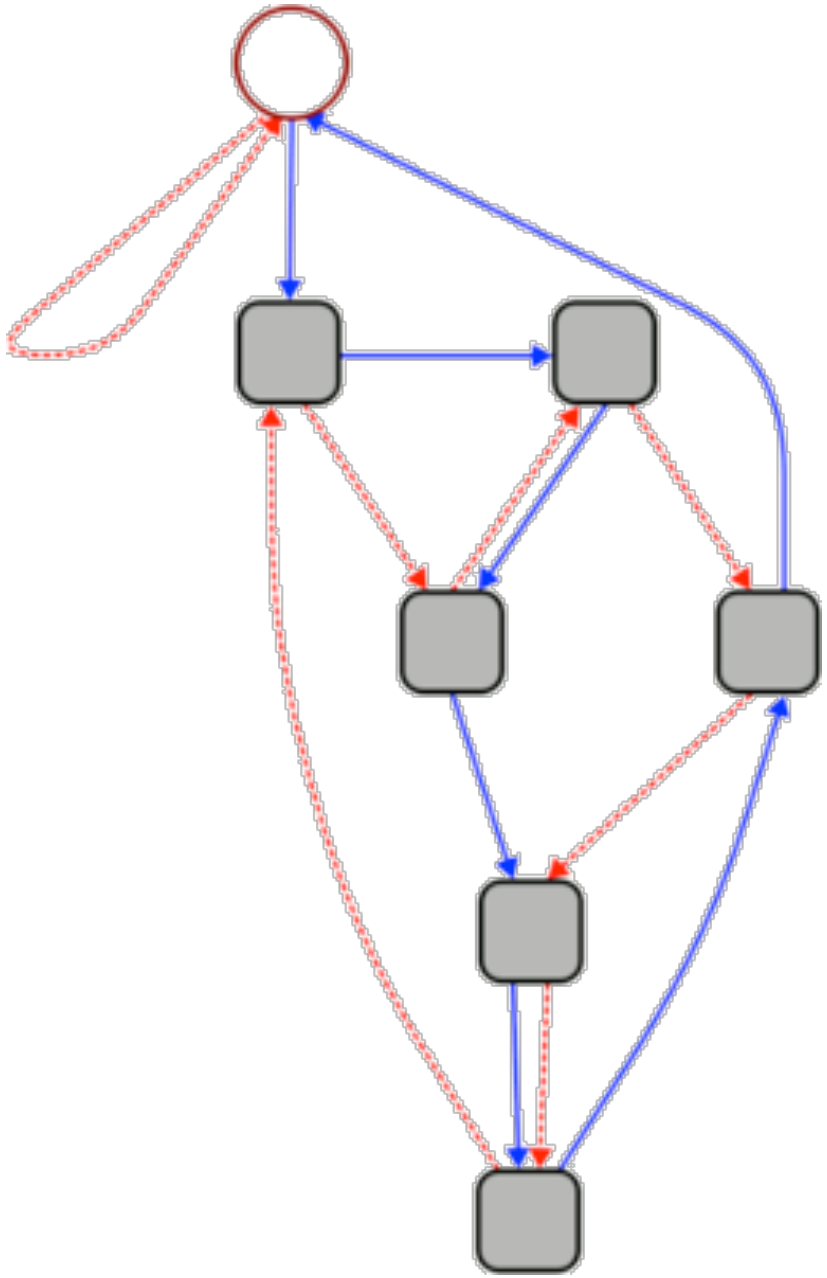
Nodes: Routers and servers

Edges: Connections

Big Data!

The Internet 2003

The OPTE Project (2003)



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

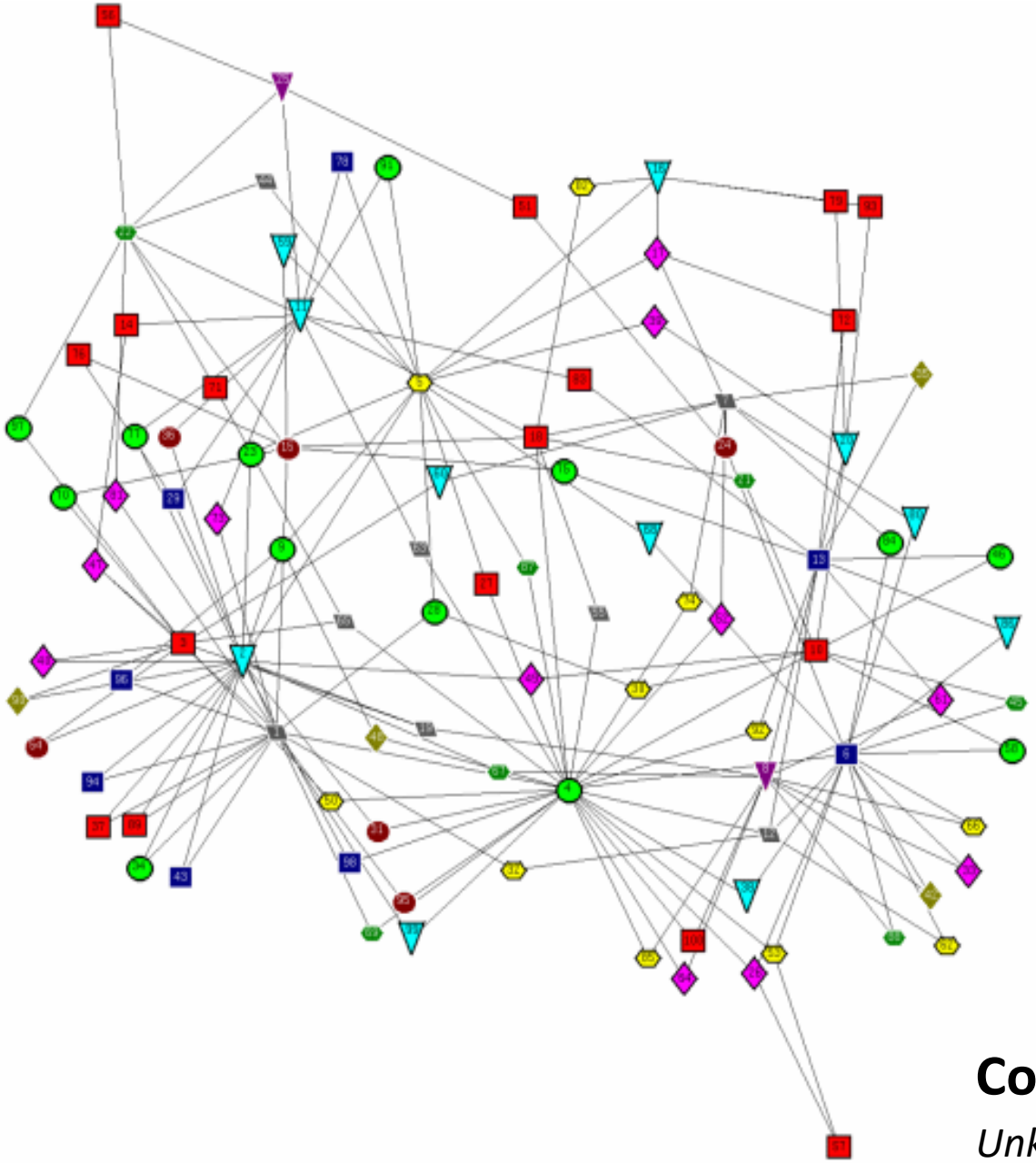
1. Start at the circle node at the top.
2. For each digit **d** in the given number, follow **d** blue (solid) edges in succession. As you move from one digit to the next, follow **1** red (dashed) edge.
3. If you end up back at the circle node, your number is divisible by 7.

3703 – divisible by 7!

"Rule of 7"

Unknown Source

Presented by Cinda Heeren, 2016



Course as node

edge is shared student

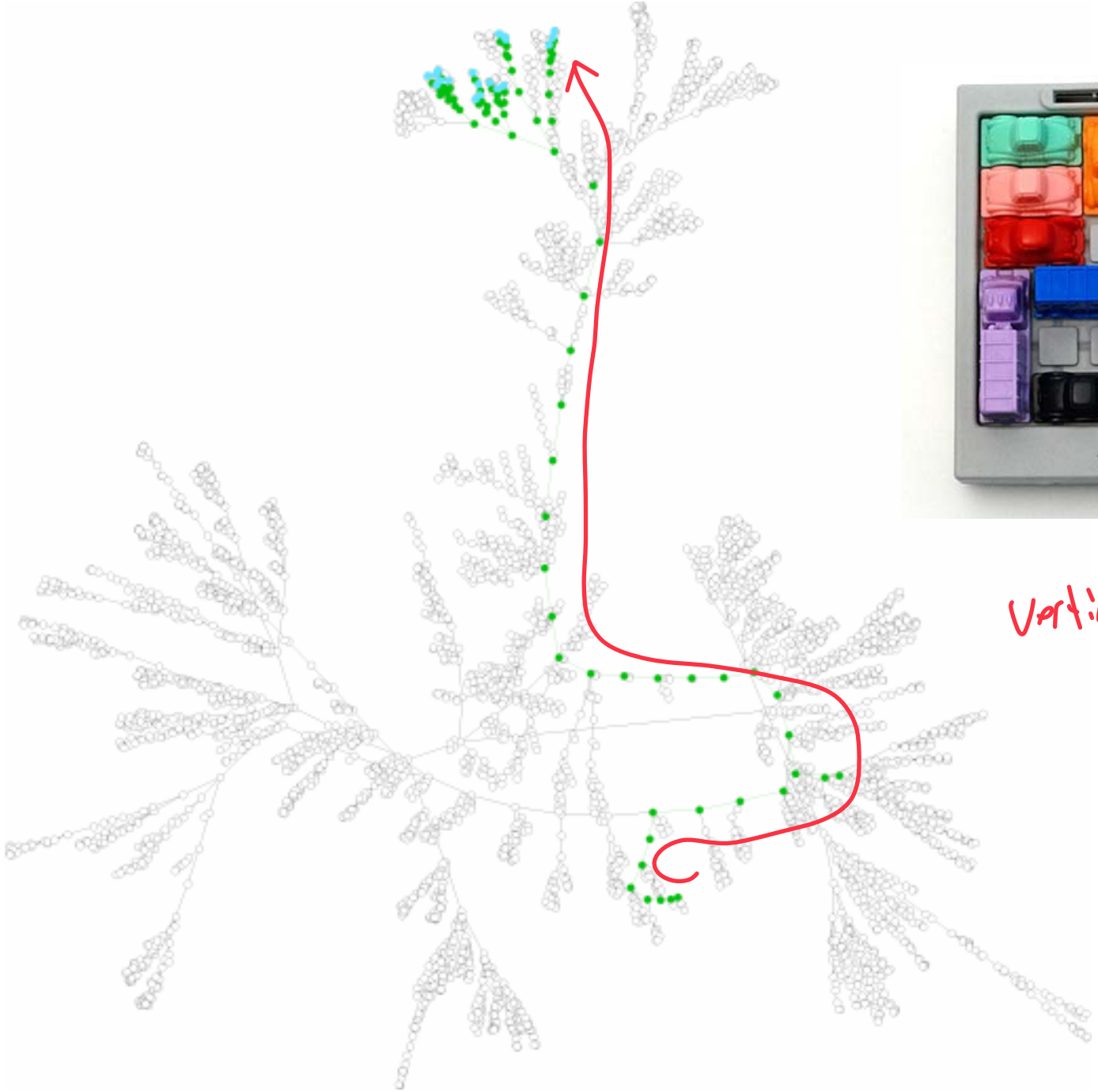
Color = exam time

↳ oops one too many exam times!

Conflict-Free Final Exam Scheduling Graph

Unknown Source

Presented by Cinda Heeren, 2016

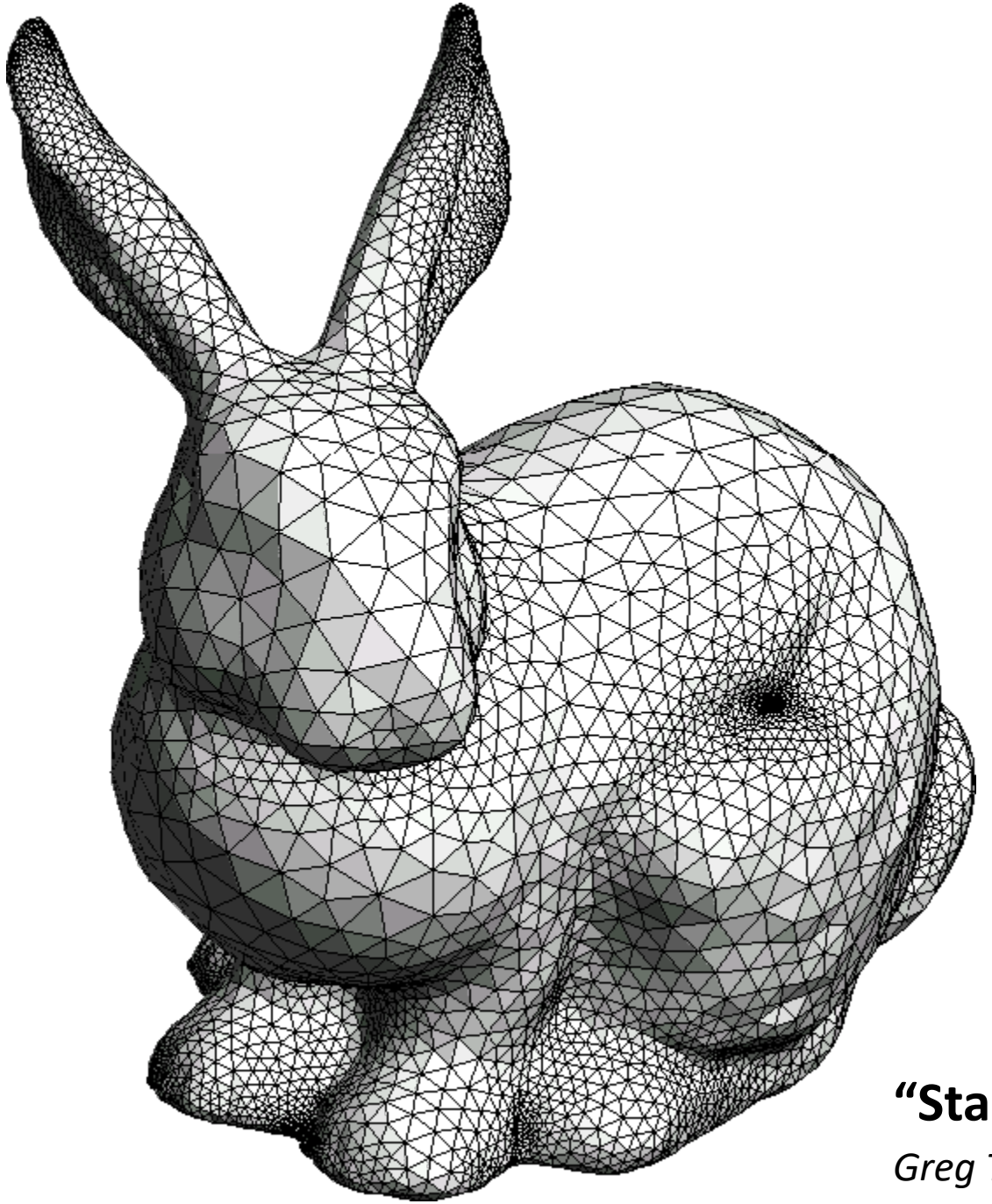


Vertices: States of a System

“Rush Hour” Solution

Unknown Source

Presented by Cinda Heeren, 2016

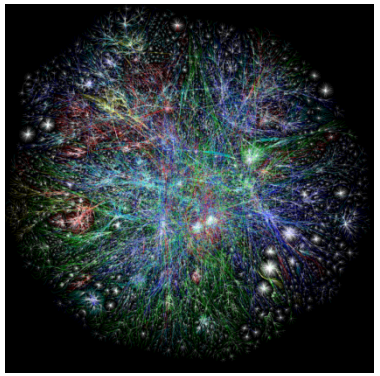


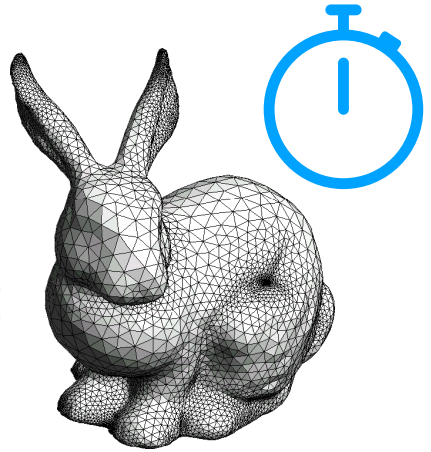
graphics!

“Stanford Bunny”

Greg Turk and Mark Levoy (1994)

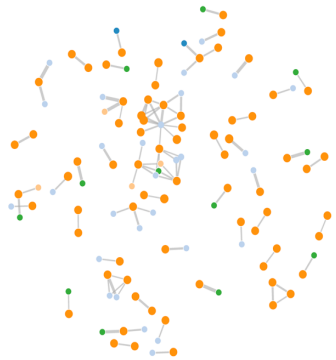
Graphs





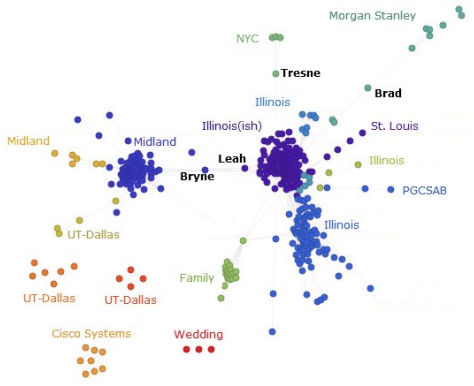
To study all of these structures:

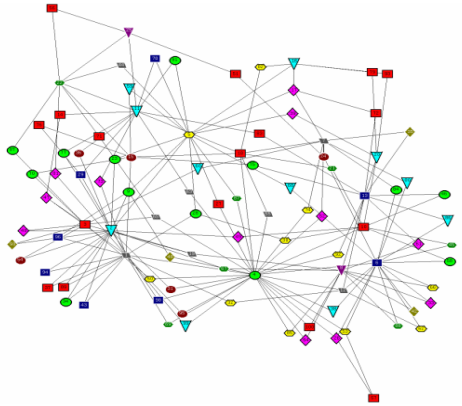
1. A common vocabulary ← Today
2. Graph implementations ← This week
3. Graph traversals ← Next week
4. Graph algorithms

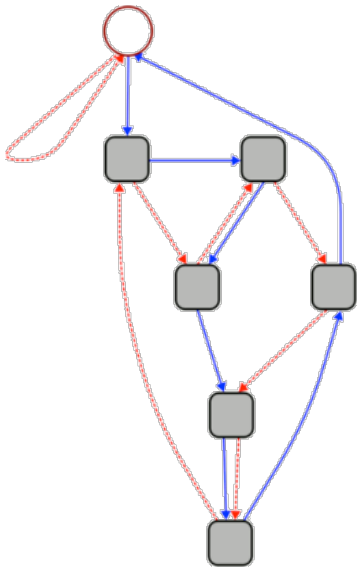


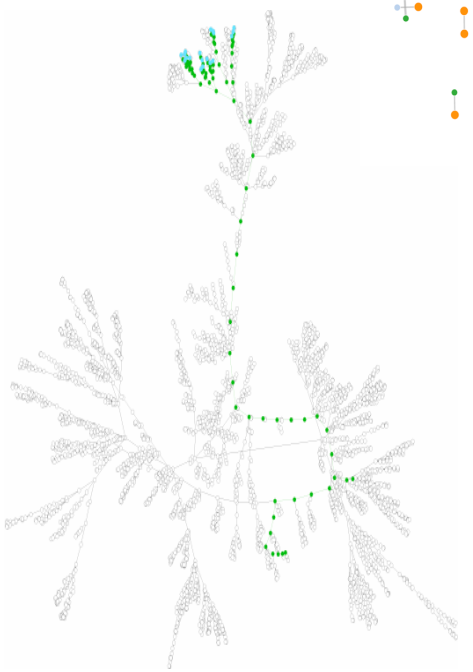
HAMLET

TROILUS AND CRESSIDA





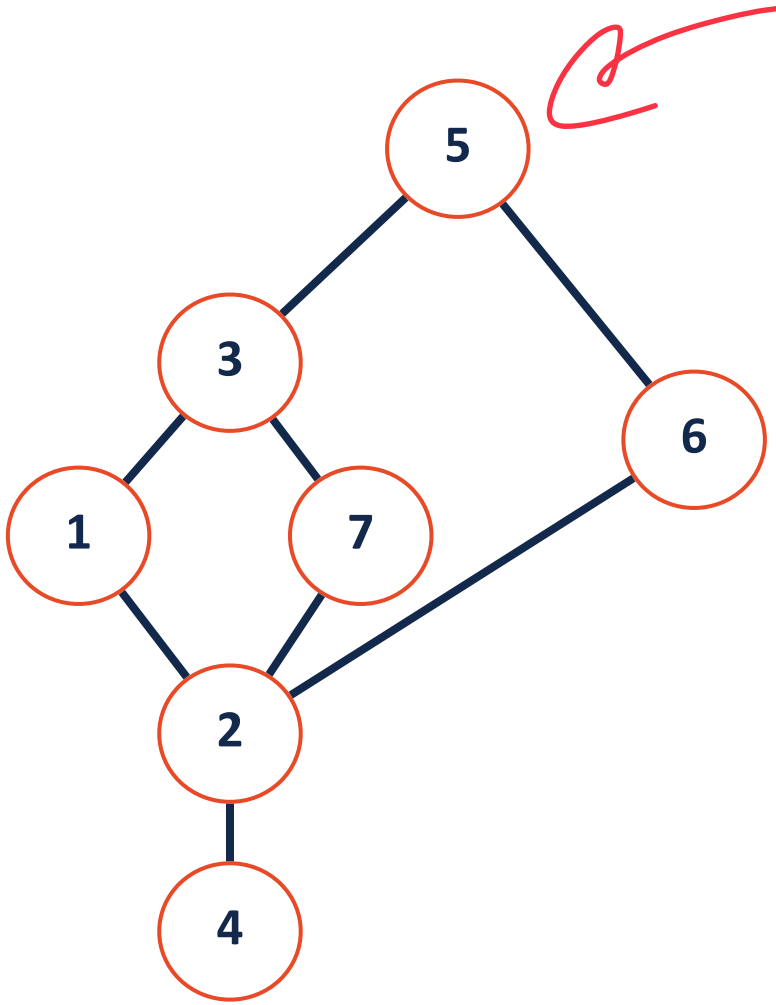




Graph Vocabulary

$G = (V, E)$

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



Vertex: Nodes of the graph

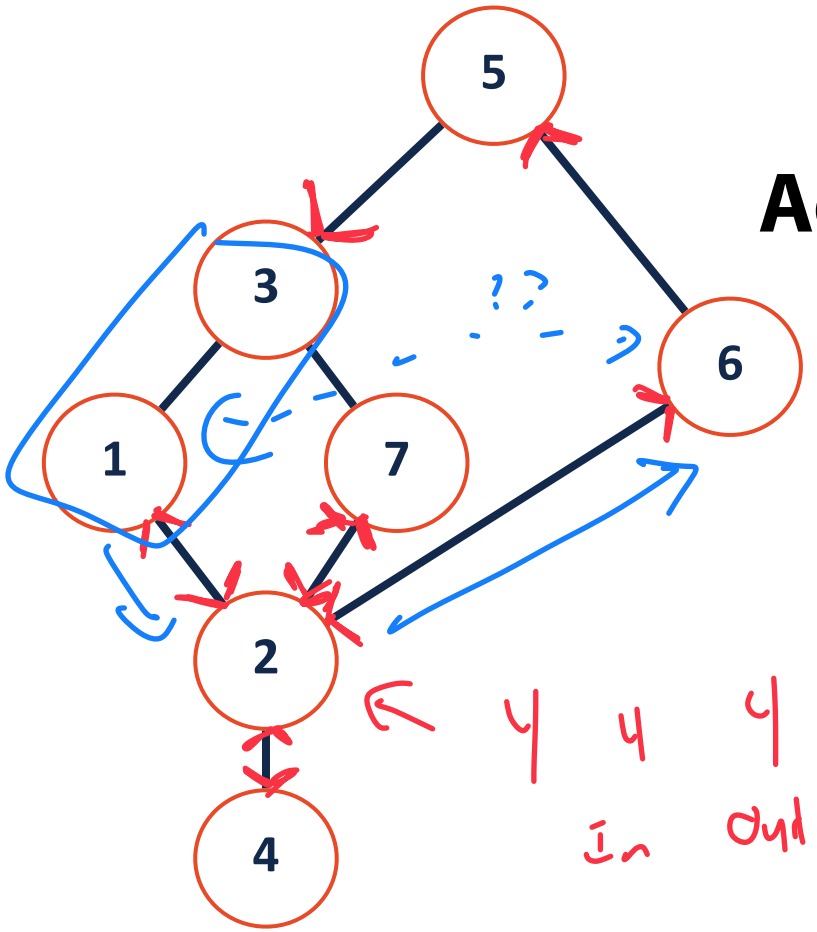
- Data
- Nothing!
- Key, value
- State of system ....

Edges: The connections between nodes

Defined by two endpoints

- Direction
- Weight

Graph Vocabulary



Degree: # of edges touching a vertex

- In-degree: # edges in
- Out-degree: # edges out

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

1 - 3

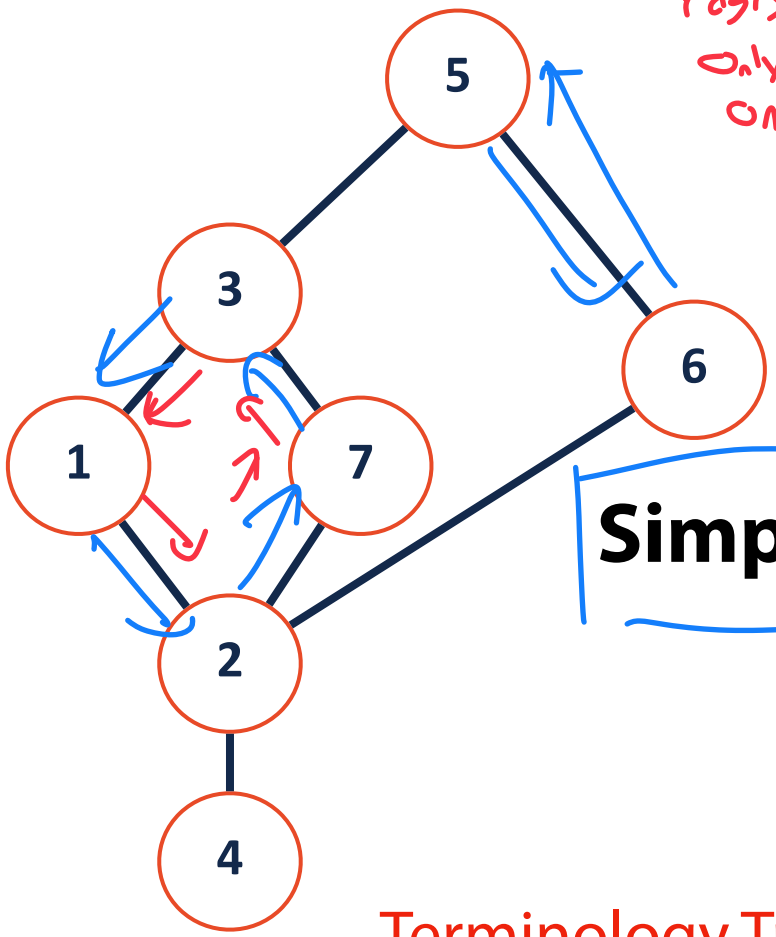
1 - 2

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

1 → 2 → 6

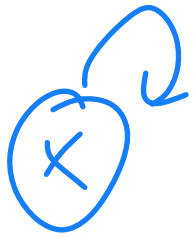
Graph Vocabulary

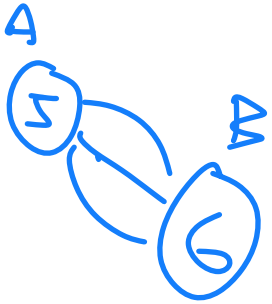
A graph has **no root** and **may contain cycles**



Cycle: A path from a node to itself

Simple Graph: No self-loops or multi-edges

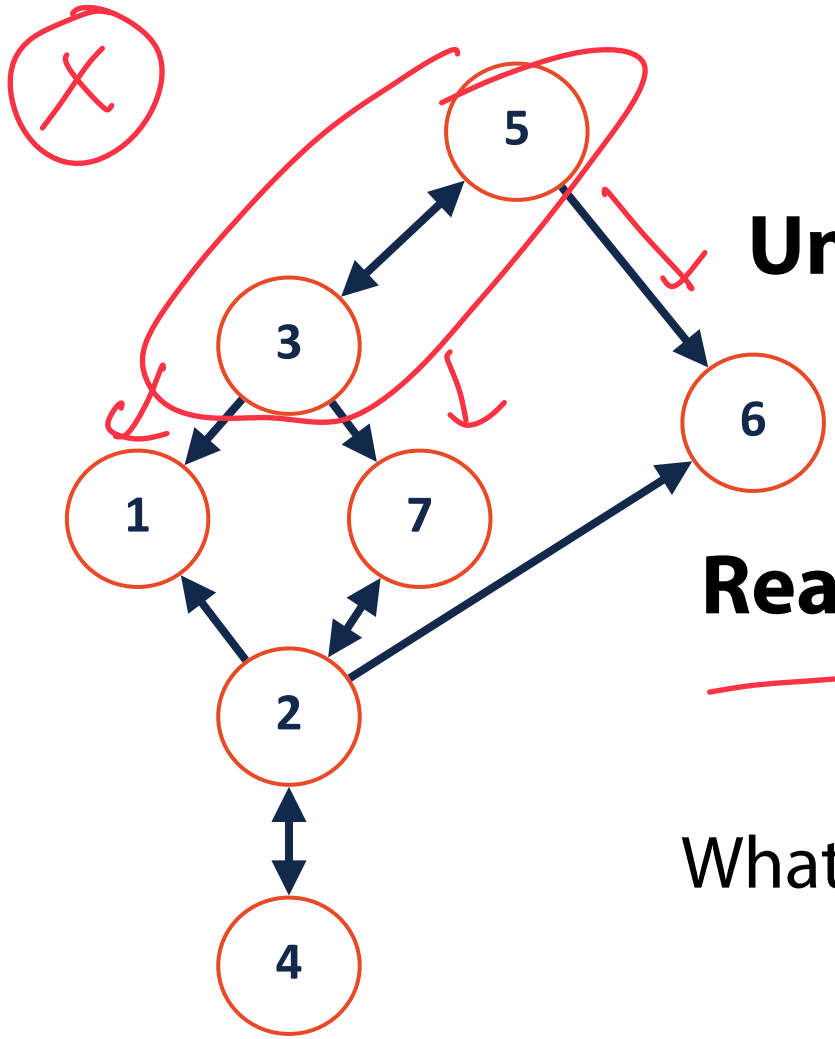




Terminology Trivia: Every tree is a graph but not every graph is a tree

Graph Vocabulary

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



Directed: Edges are one way connections

Undirected: Traversable in either direction

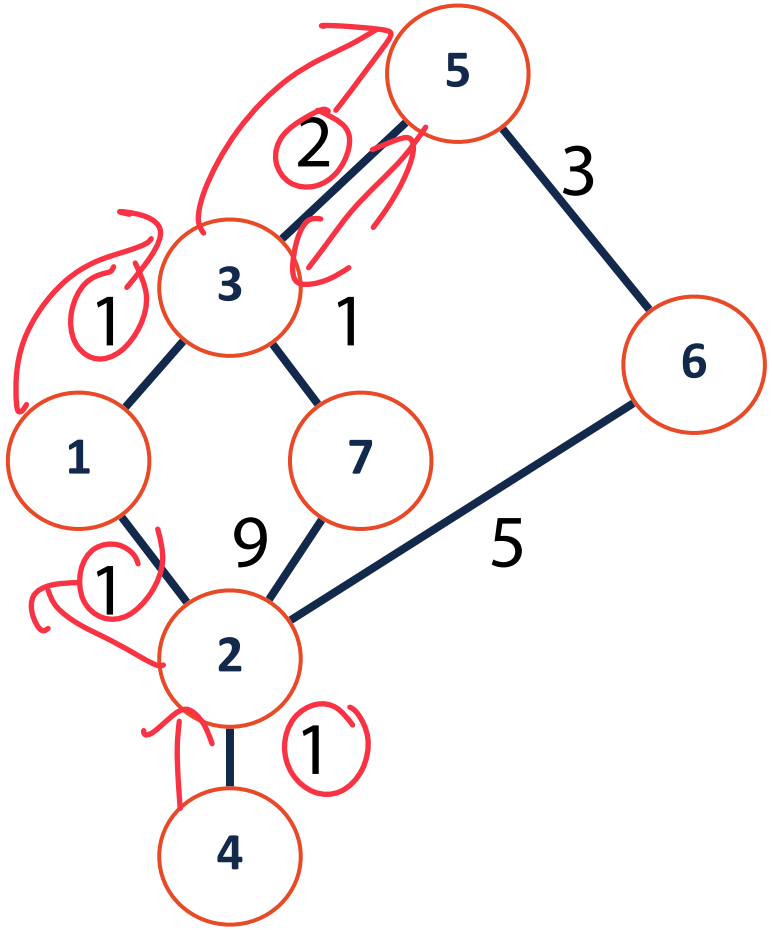
Reachability: v_2 is reachable from v_1 if there is a path from v_1 to v_2

What nodes are not reachable from 4?

Graph Vocabulary

Start End Weight
3 5 2

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



Weights: A value associated with an edge

What is the shortest path from 4 to 5? 5

Sum of weights

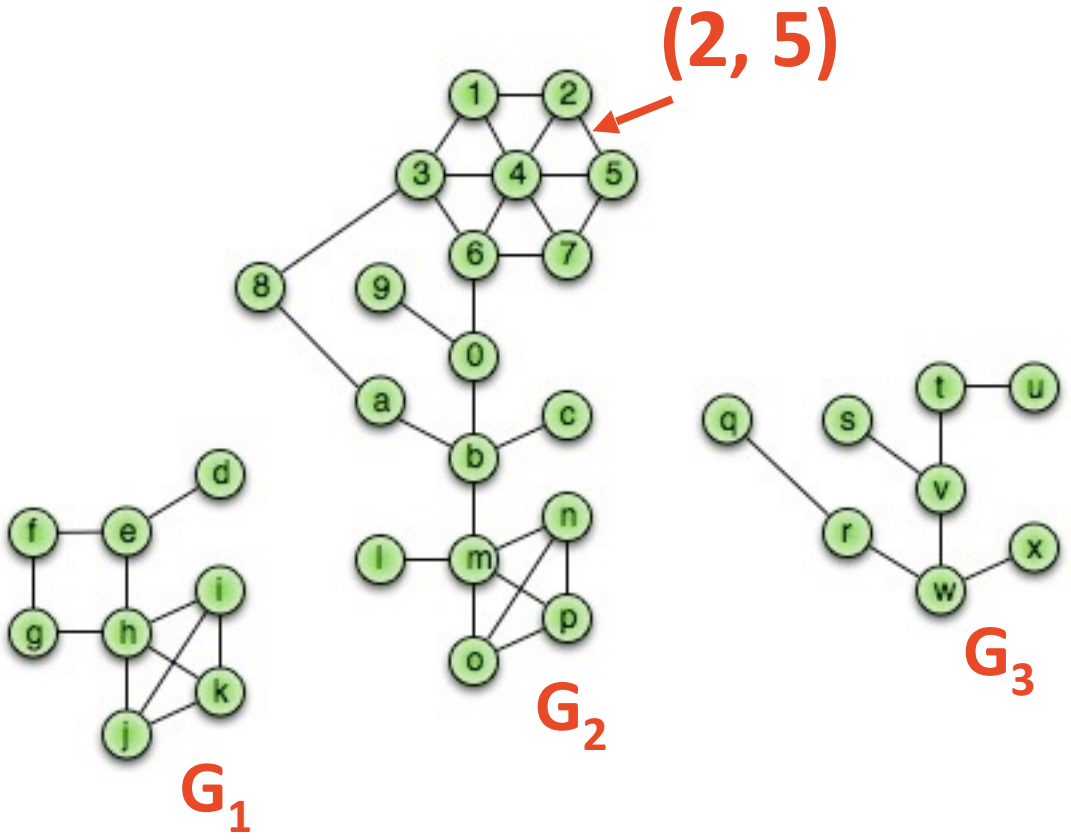
Unweighted shortest Path: # edges
↳ Every edge weight 1

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Subgraph(G):

G' = (V', E'):

V' ∈ V, E' ∈ E, and

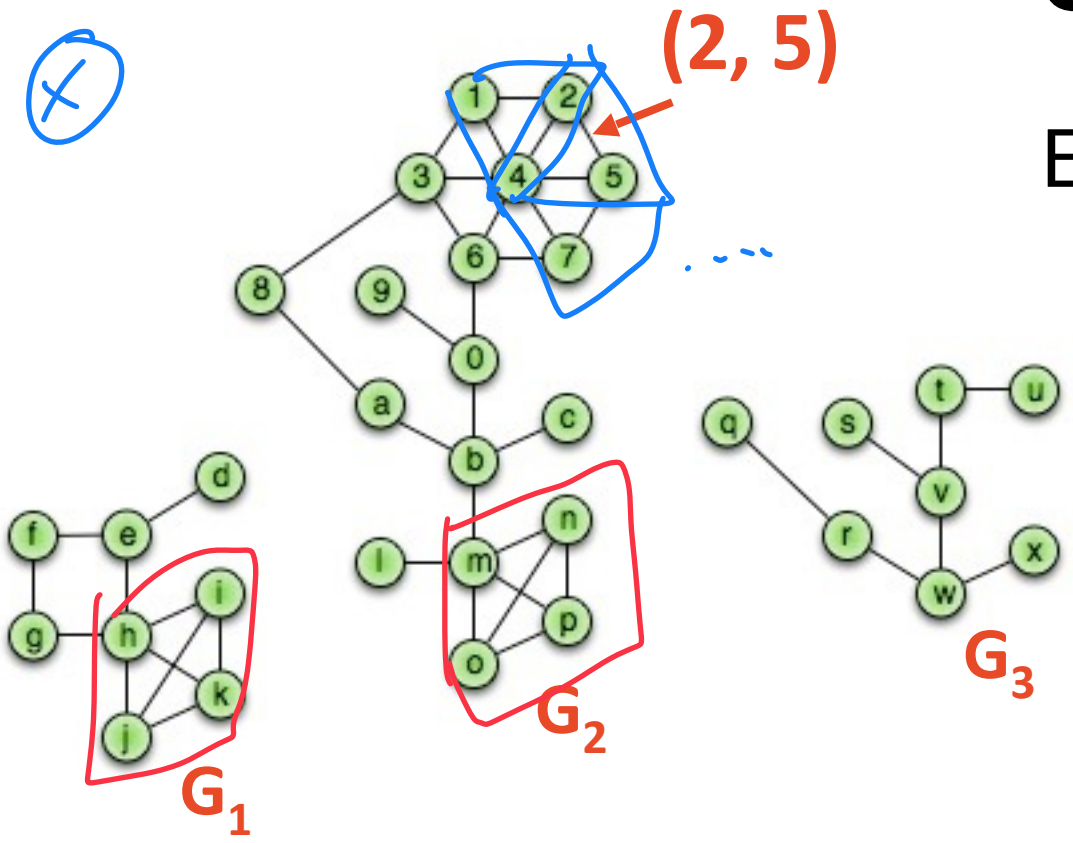
(u, v) ∈ E' → u ∈ V', v ∈ V'

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

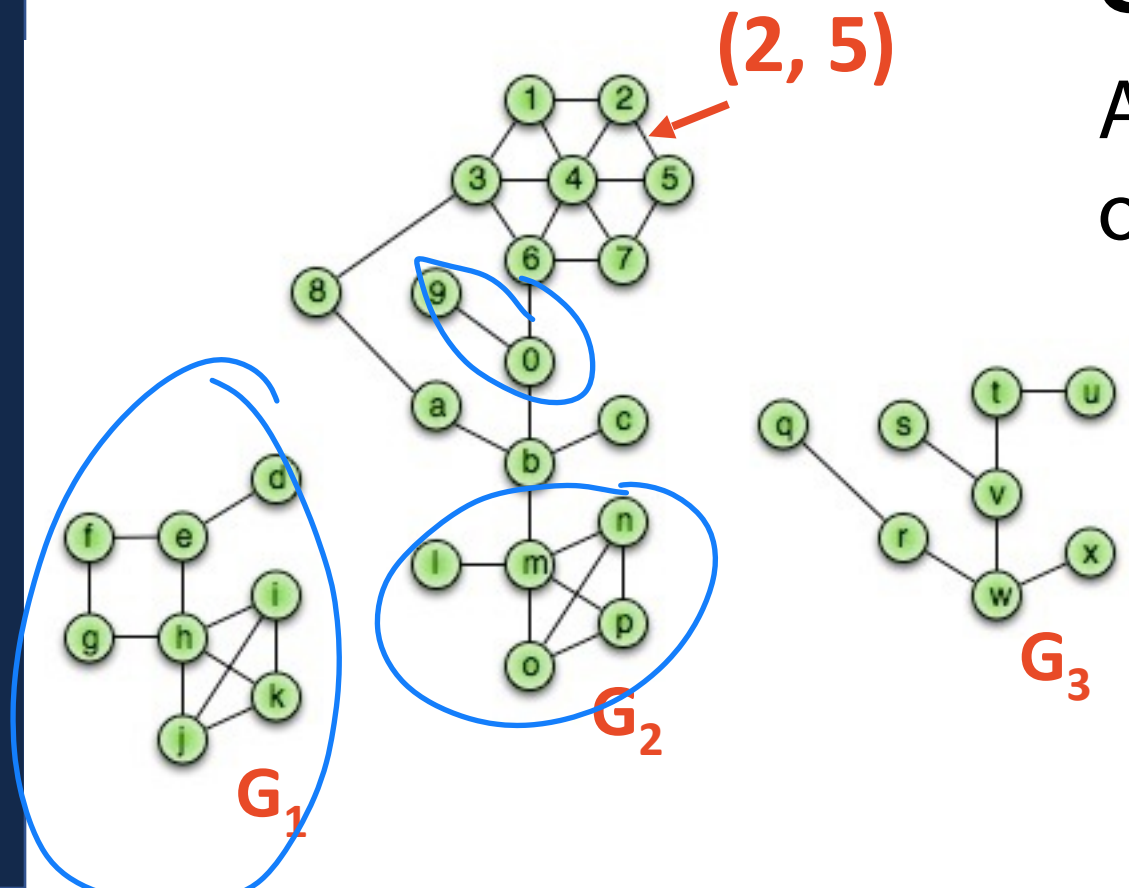
|E| = m

Subgraph(G):

G' = (V', E'):

V' ∈ V, E' ∈ E, and

(u, v) ∈ E' → u ∈ V', v ∈ V'



Connected Subgraph:

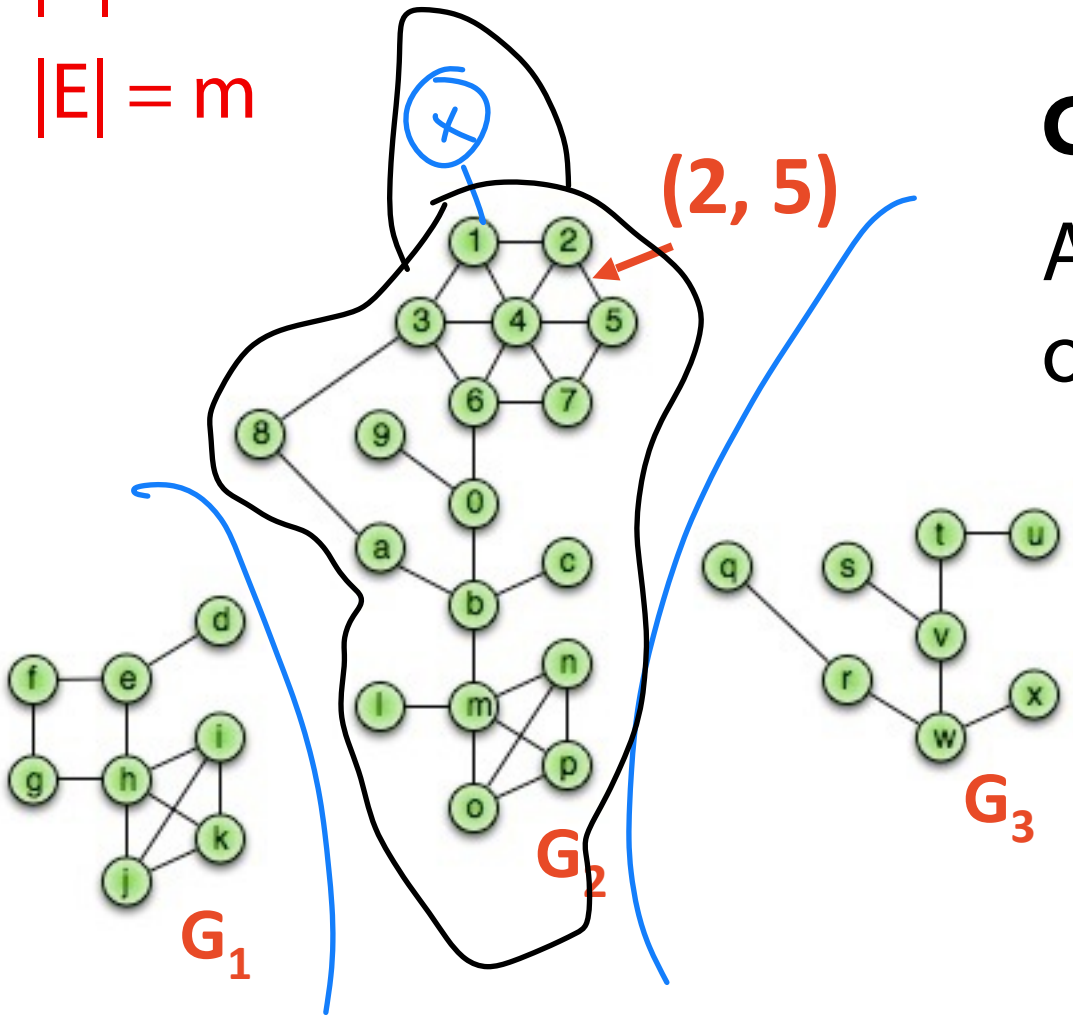
A path exists between every pair of vertices

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Subgraph(G):

G' = (V', E'):

V' ∈ V, E' ∈ E, and

(u, v) ∈ E' → u ∈ V', v ∈ V'

Connected Subgraph:

A path exists between every pair of vertices

Connected Components:

A connected subgraph that is not part of a larger subgraph

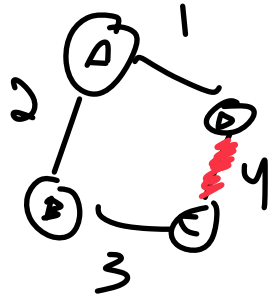
↳ The largest connected subgraph

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$

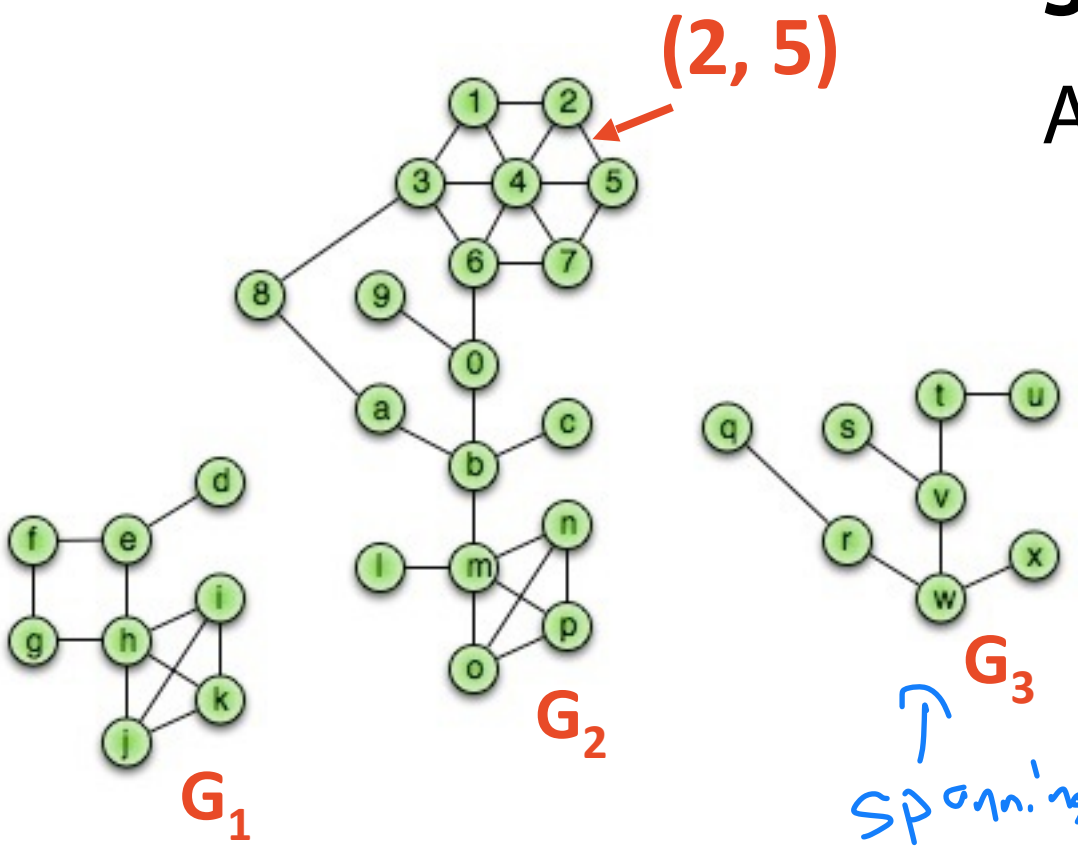


Subgraph(G):

G' = (V', E'):

V' ∈ V, E' ∈ E, and

(u, v) ∈ E' → u ∈ V', v ∈ V'



Spanning Tree:

A connected graph with no cycles

Minimum spanning tree

↳ picks lowest weight

n-1 edges in spanning tree

spanning tree

Graph Vocabulary

$G = (V, E)$

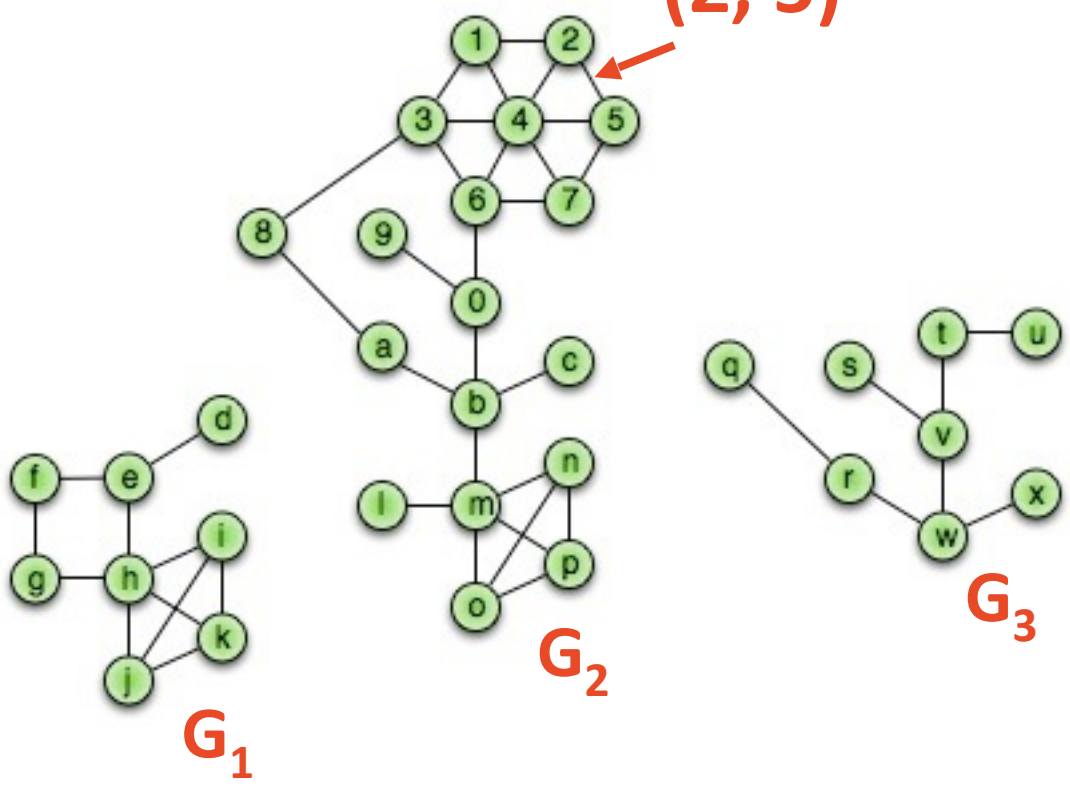
$|V| = n$

$|E| = m$

collection (or set) of vertices & edges

Else

(2, 5)



Graph Terminology is very important!

- Degree
- Weight
- Direction
- Adjacency
- Complete
- Connected
- Acyclic
- Spanning

And more…

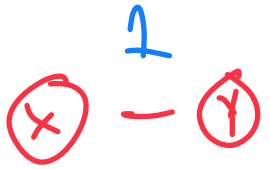
Running times are often reported by **n**, the number of vertices, but often depend on **m**, the number of edges.

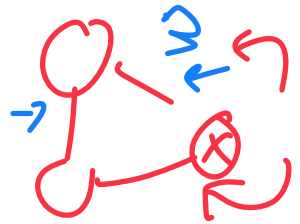
Whats the relationship between **n** and **m**?

Minimum Edges:

Unconnected Graph:

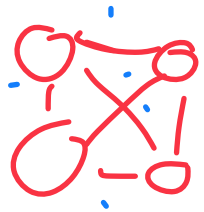
Connected (Simple) Graph: $n - 1$





Maximum Edges:

Connected (Simple) Graph: $(N-1) + (N-2) + (N-3) + \dots$



$$\sum_{v \in V} deg(v) = \frac{n(n-1)}{2} \approx O(n^2)$$

Graphs

A → B

A ← B

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

What is a vertex?

Individual

What is an edge?

A message

Are the edges directed or undirected?

↳ Directed

↳ Depends!

Are the edges weighted or unweighted?

↳ Depends!

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:

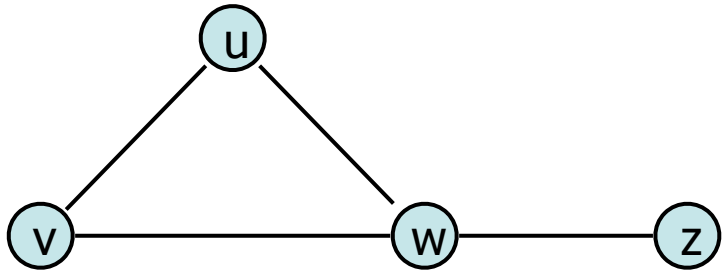
- Does your graph have cycles?
- What is the largest / smallest / average degree in your graph?
- What is the total number of edges?
- …

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:

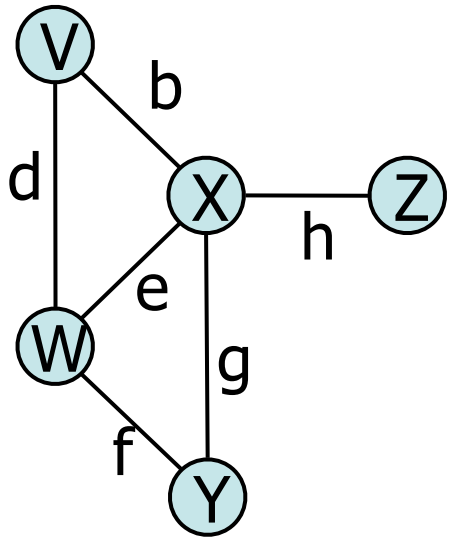
What information do we want to be able to find out quickly?

What operations do we want to prioritize?

Graph ADT

Data:

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



Functions:

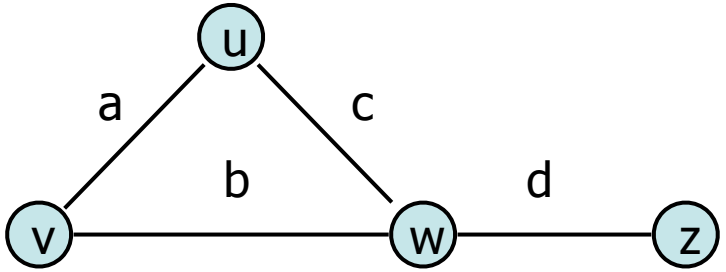
- insertVertex(K key);
- insertEdge(Vertex v1, Vertex v2, K key);

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

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

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

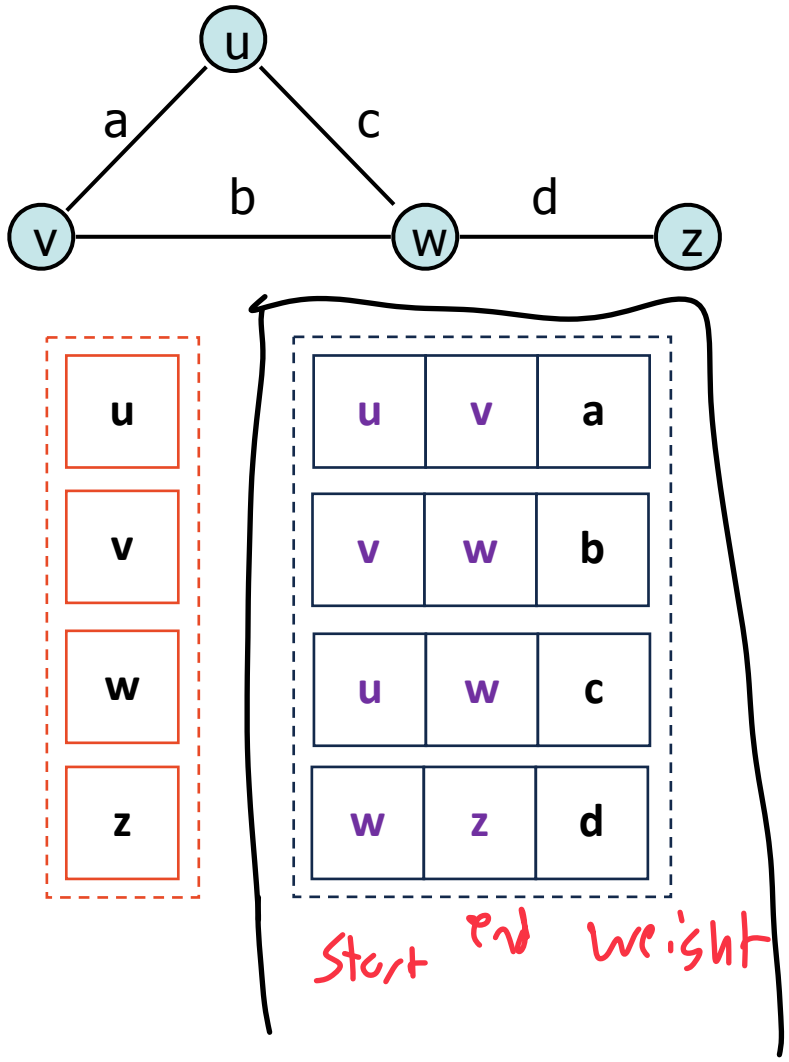
Graph Implementation Idea



↳ A list!

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

The equivalent of an 'unordered' data structure



Vertex Storage:

↳ Optionally as a list of vertex labels

↳ Build this from edge storage

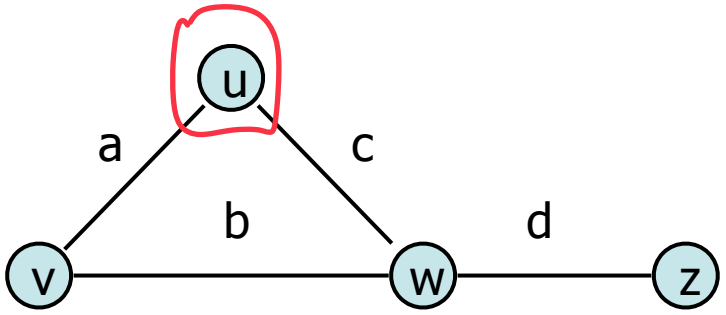
Edge Storage:

↳ A list of three values

↳ Very compact representation

Graph Implementation: Edge List

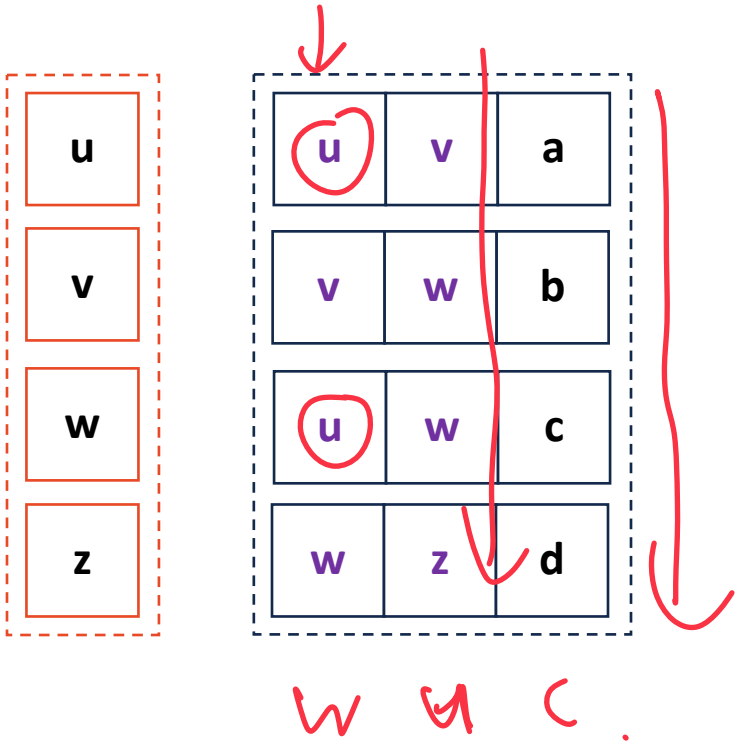
$|V| = n, |E| = m$



getEdges(Vertex v) $O(m)$

↳ Traverse edge storage

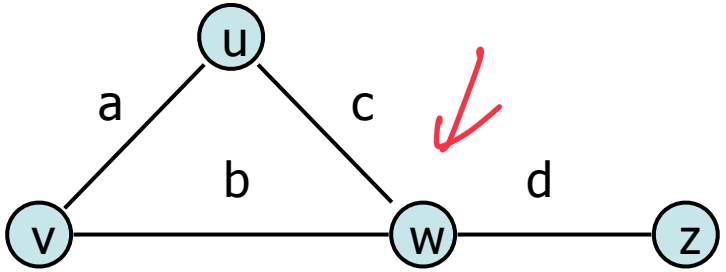
Look for v in start or end



areAdjacent(Vertex v1, Vertex v2) $O(m)$

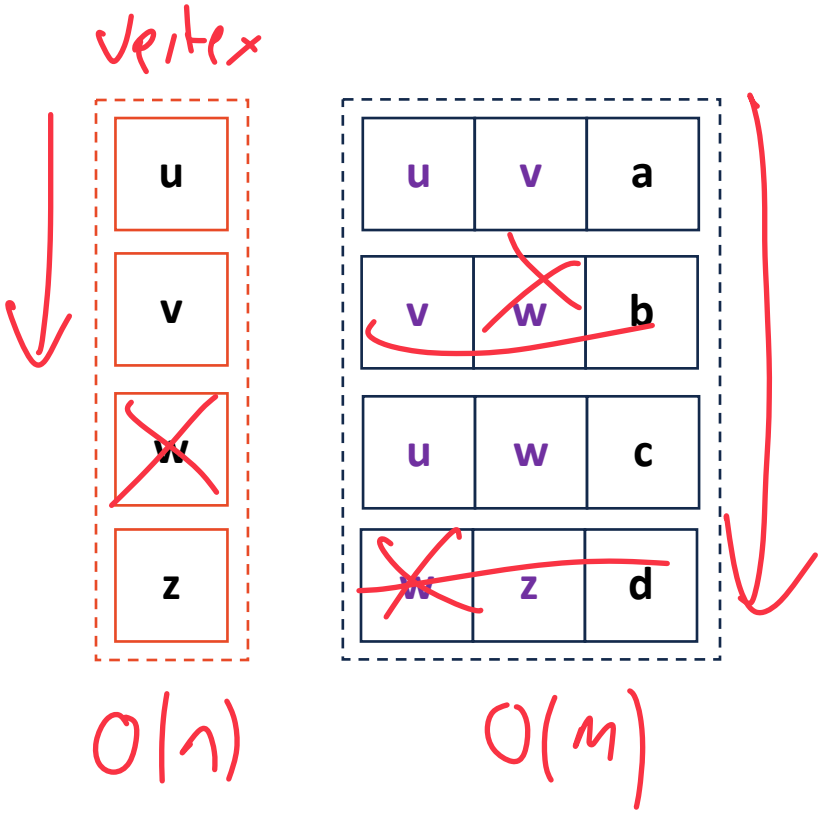
↳ Do same thing!

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



insertVertex(K key) $O(1)^*$

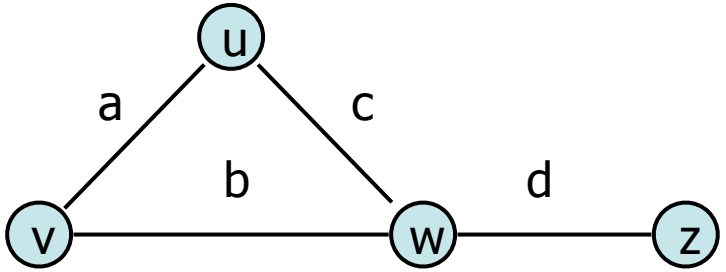
↳ Array insert



removeVertex(Vertex v) $O(n+m)$

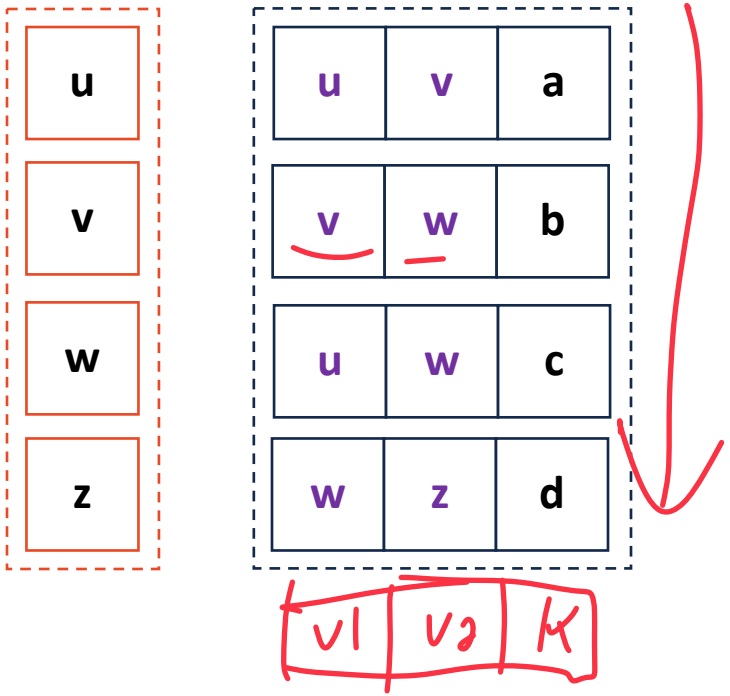
↳ Array removal

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



insertEdge(Vertex v1, Vertex v2, K key)

↳ Array insert $O(1)^*$



removeEdge(Vertex v1, Vertex v2)

↳ $O(m)$

Graph Implementation: Edge List

Pros:

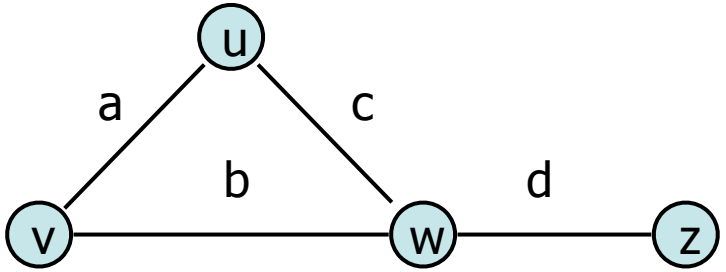
Cons:

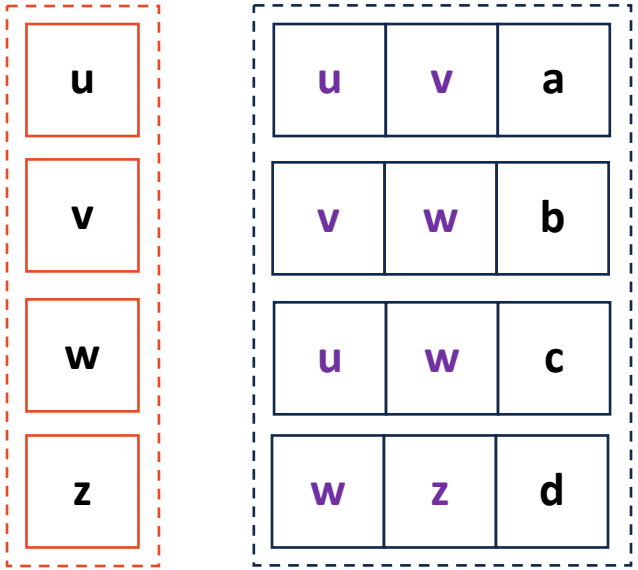
Graph Implementation: Brainstorming better

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				