

Data Structures

Graph Fundamentals

CS 225

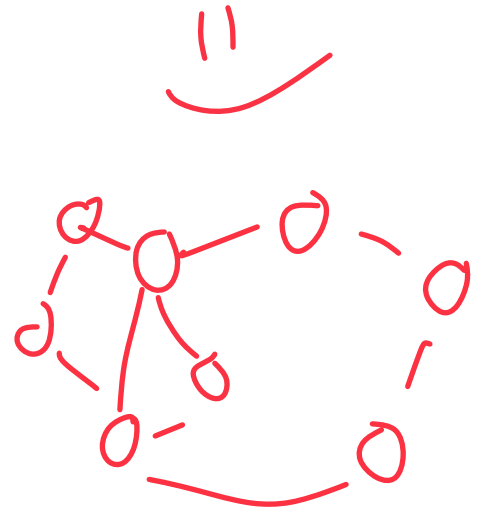
October 23, 2026

Brad Solomon



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science



Announcements

Reminder: MP_Traversals due next week!

~~Pretest 7~~

MP_Mosier's is next MP

Informal Early Feedback Form out now!

The largest of the three course surveys!

Worth 4 EC points if >60% complete!

Learning Objectives

Motivate graph as data structure

Define graph vocabulary ↘ Key Properties

Discuss graph implementation and storage strategies

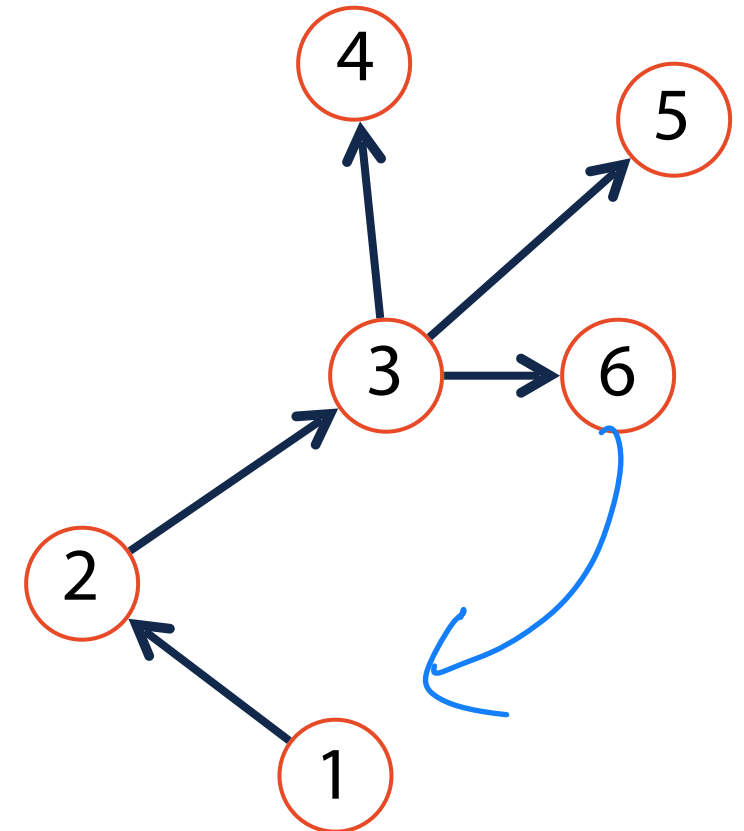
Whats next?

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: → graph

1) Acyclic — contains no cycles

2) Rooted — root node connected to all nodes



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.

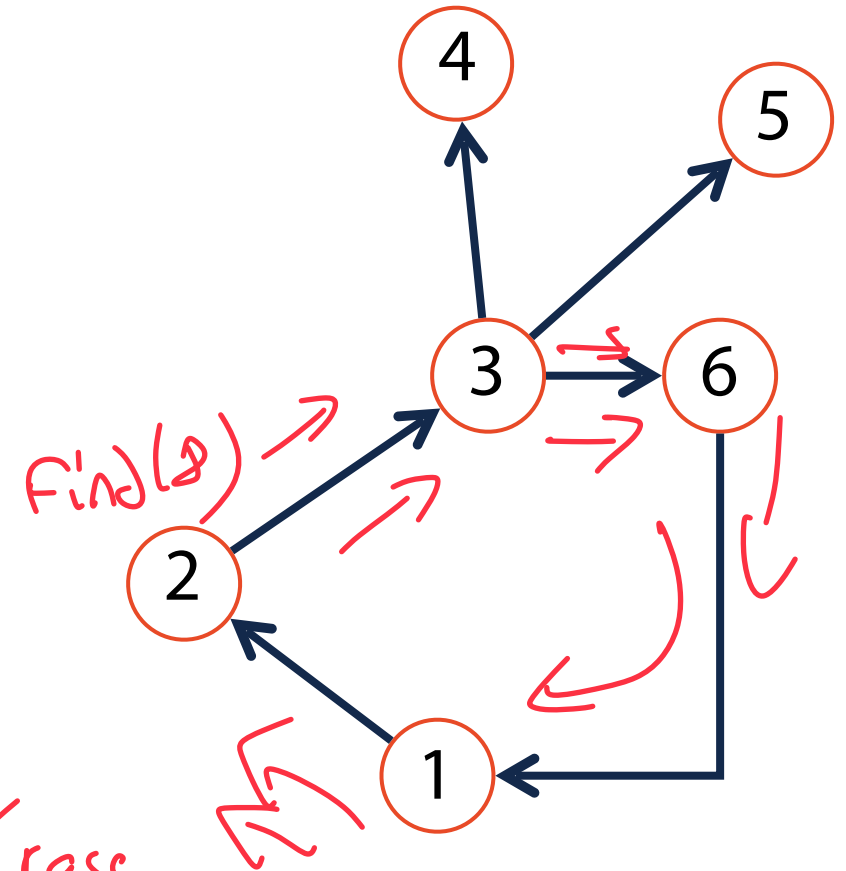
(In CS 225) a graph:

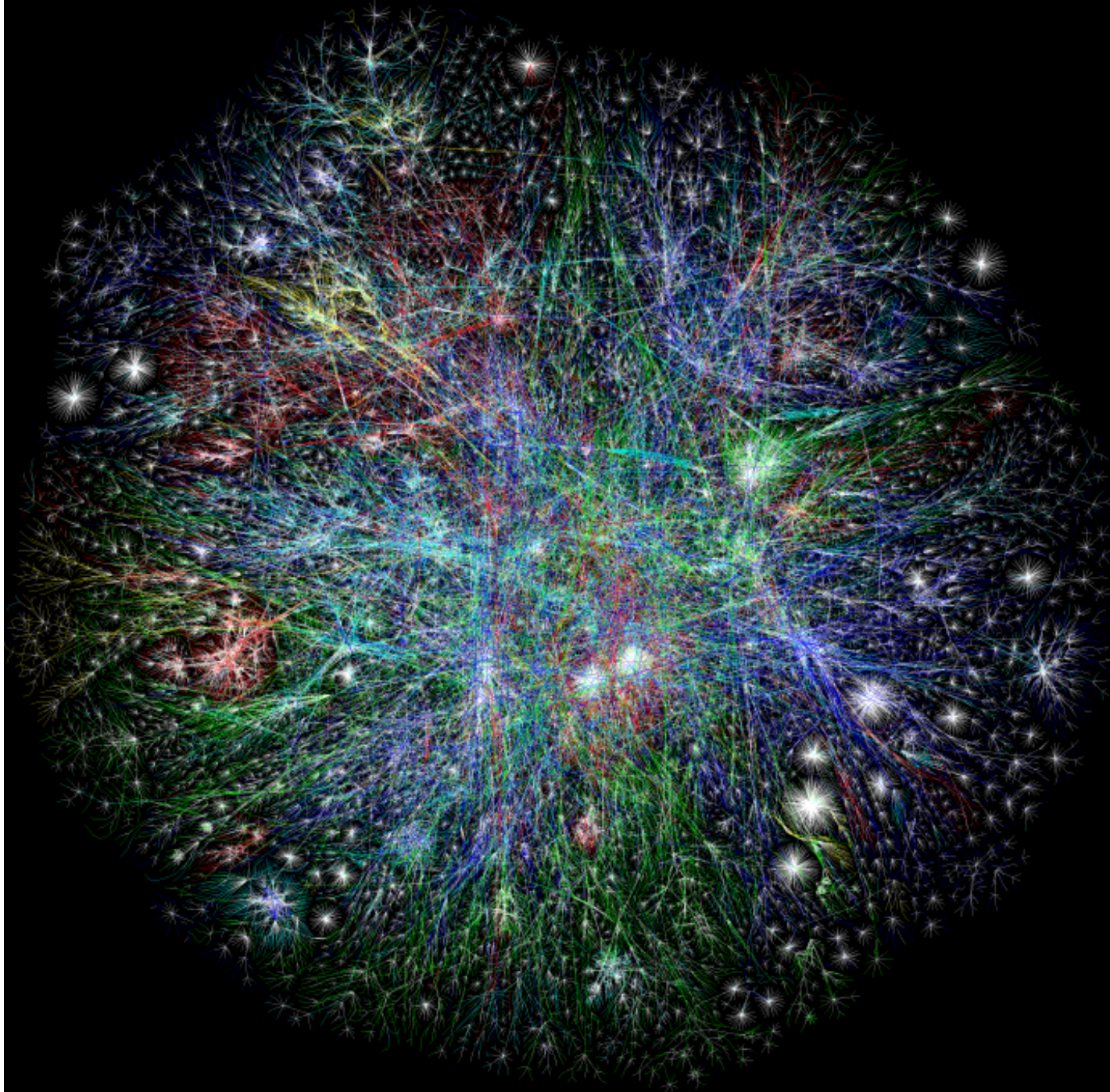
1) Can contain Cycles

2) Has no clear root

In trees: Clear base case!

Graphs: Immediately more difficult - No clear end case





Nodes: Routers and servers

Edges: Connections

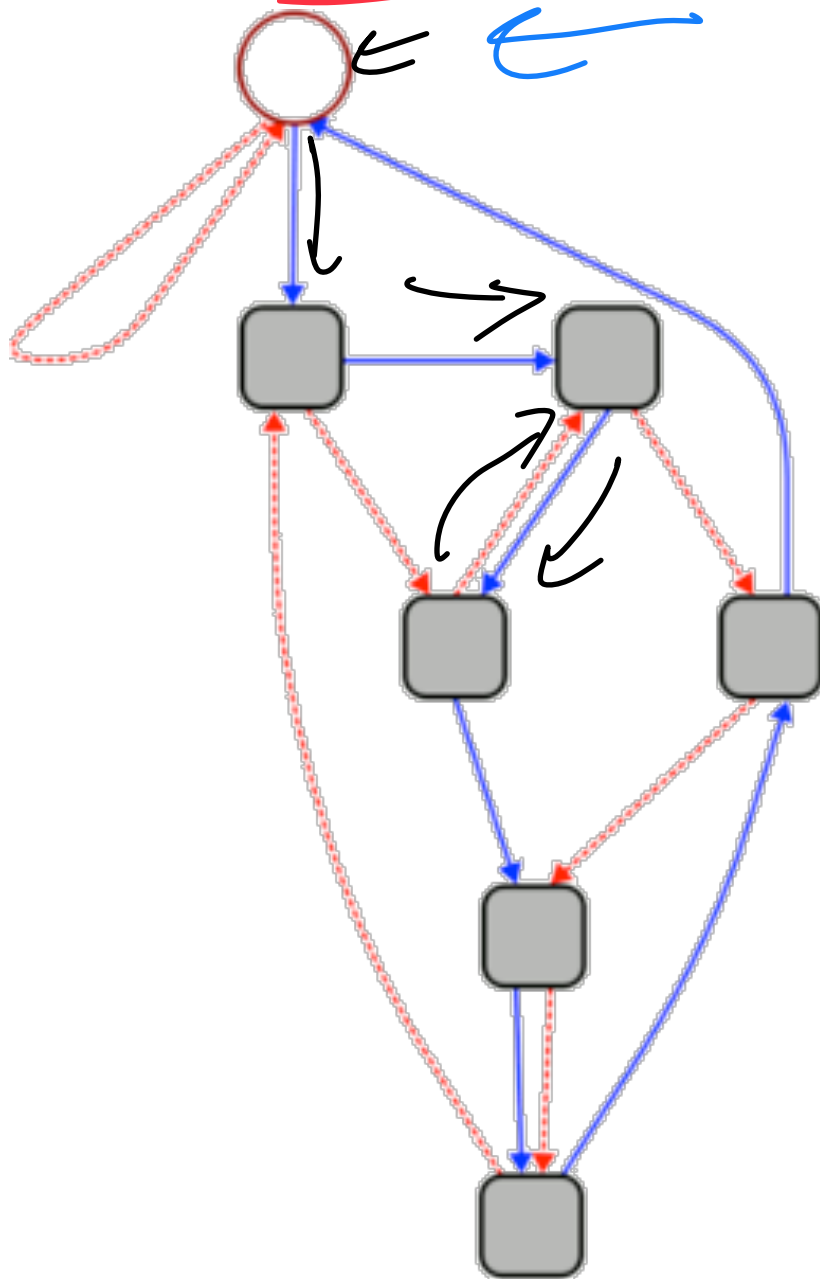
Data Structure

Big data structure

The Internet 2003

[The OPTE Project](#) (2003)

Graph as algorithm



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

Is divisible by 7

“Rule of 7”

Unknown Source

Presented by Cinda Heeren, 2016

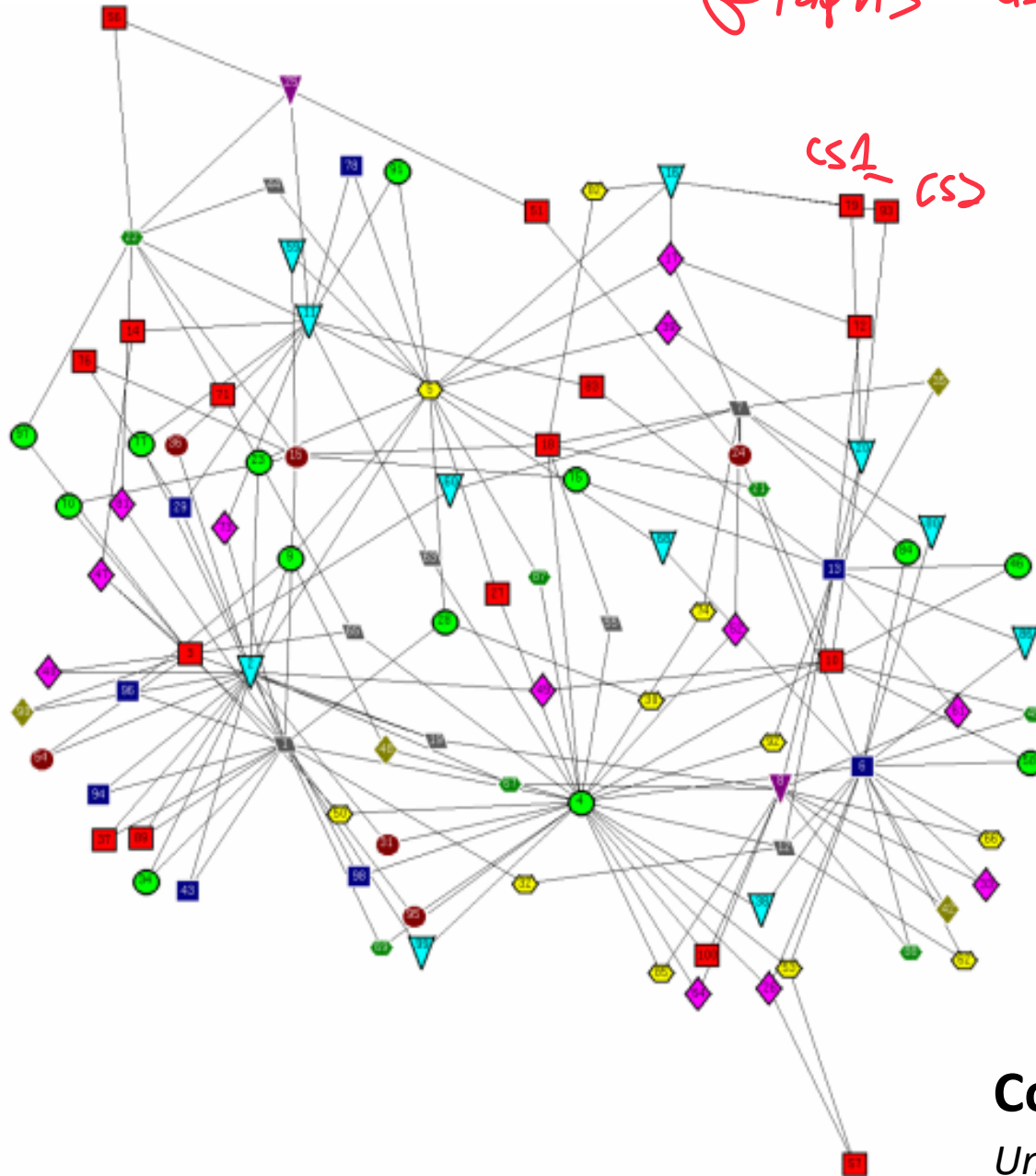
Graphs as heuristics

Graph coloring

→ is exam
scheduled
time!

Every class is a node

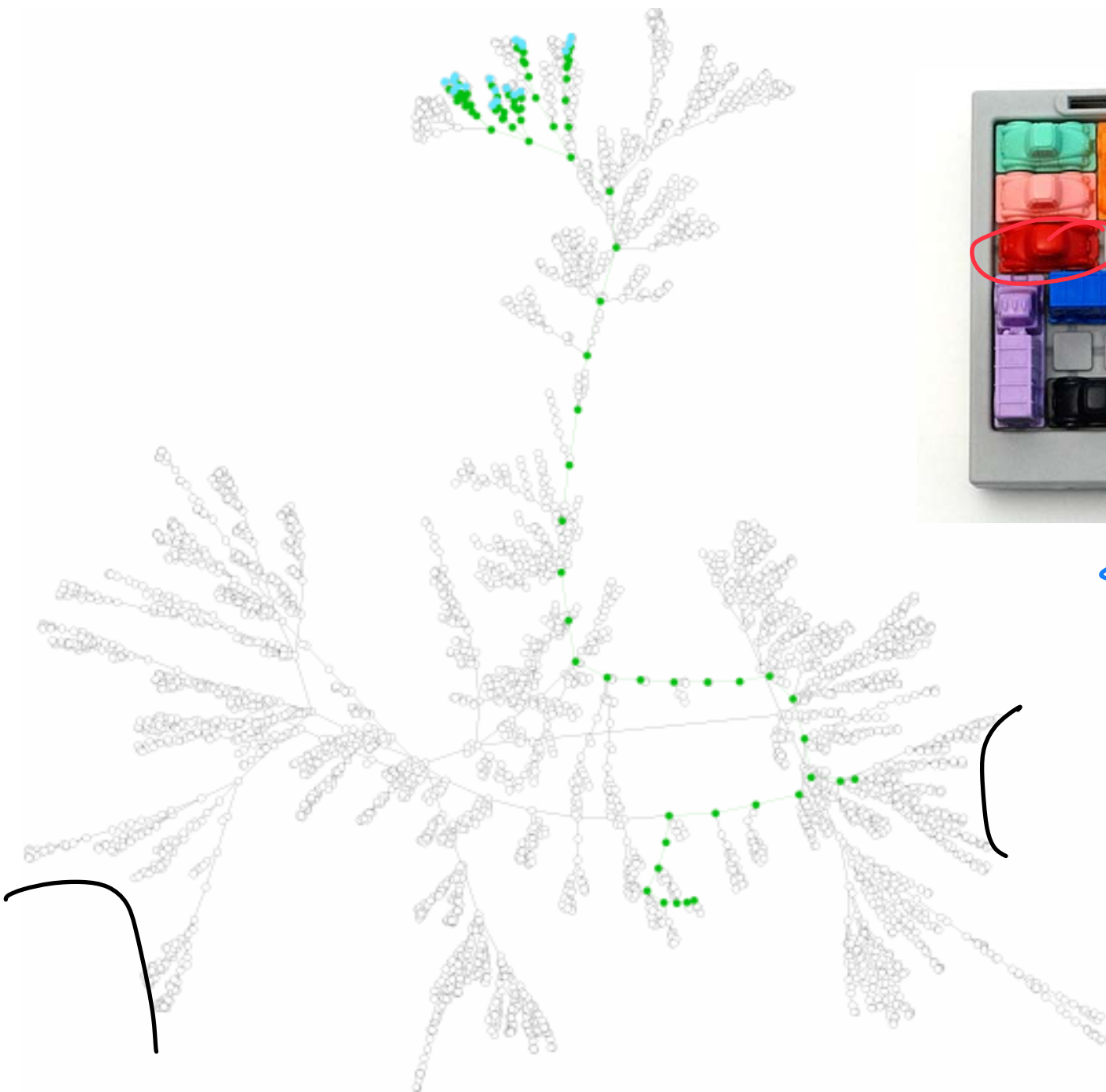
→ Edge if one student
is in two classes



Conflict-Free Final Exam Scheduling Graph

Unknown Source

Presented by Cinda Heeren, 2016



MP-Puzzle

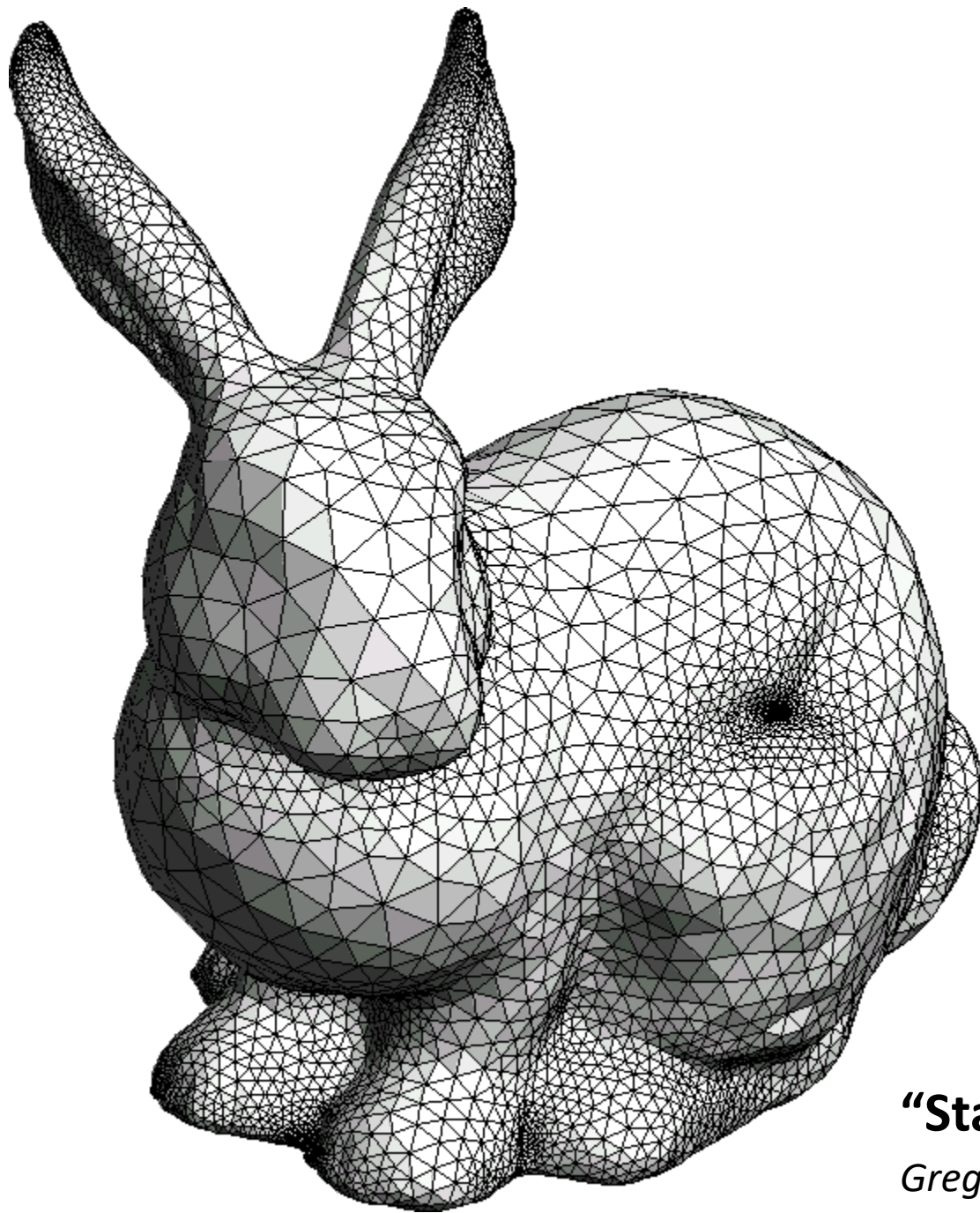
State Graph

↳ edge is a state change

“Rush Hour” Solution

Unknown Source

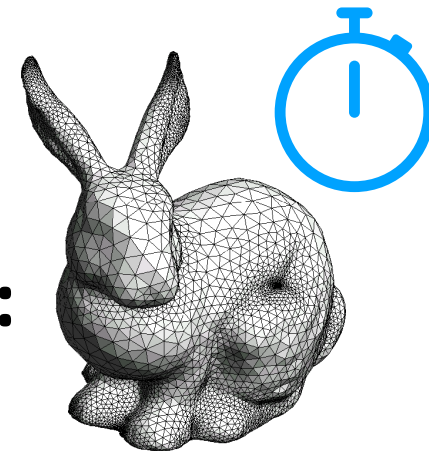
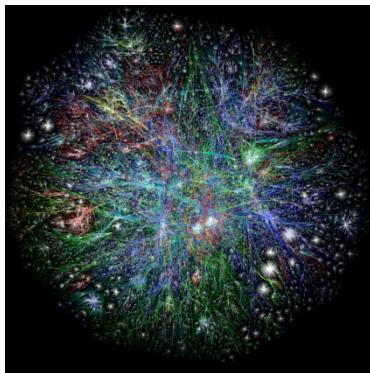
Presented by Cinda Heeren, 2016



“Stanford Bunny”

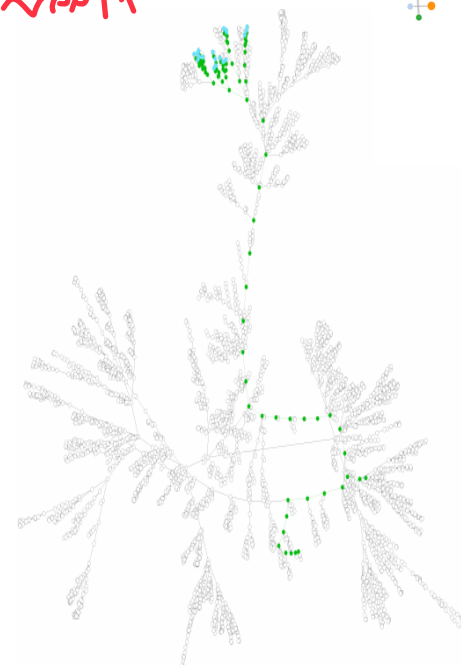
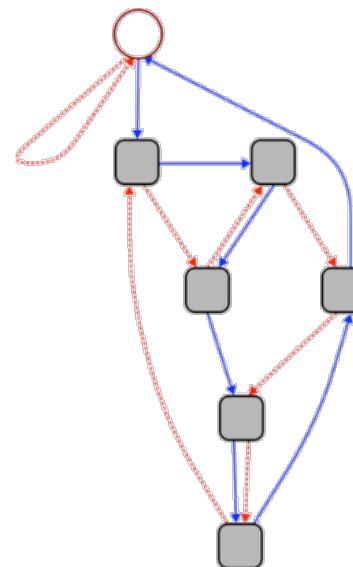
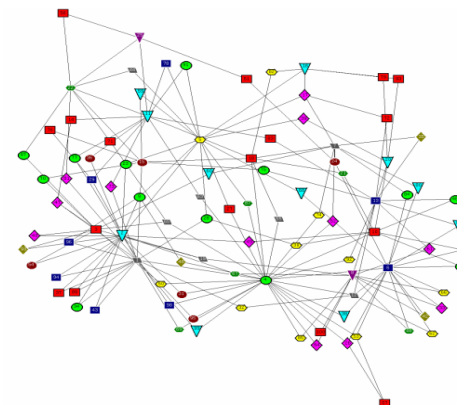
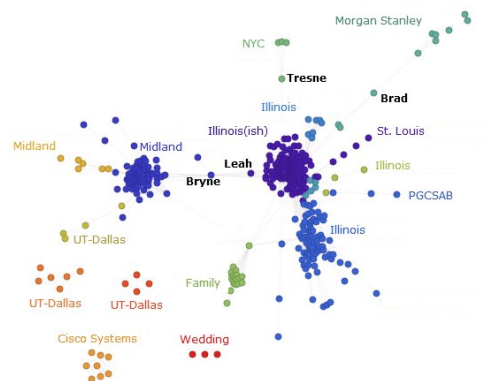
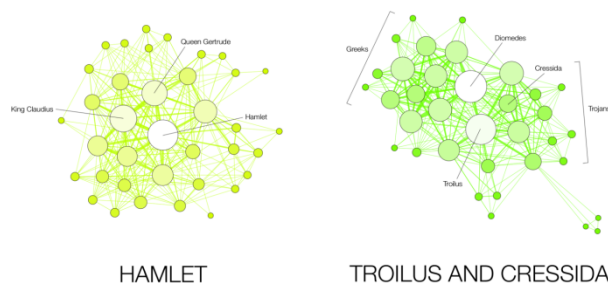
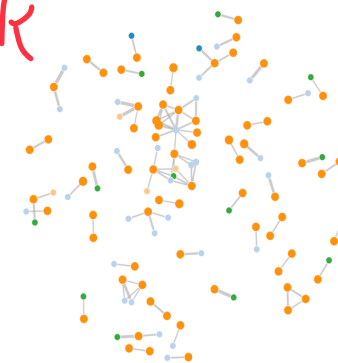
Greg Turk and Mark Levoy (1994)

Graphs



To study all of these structures:

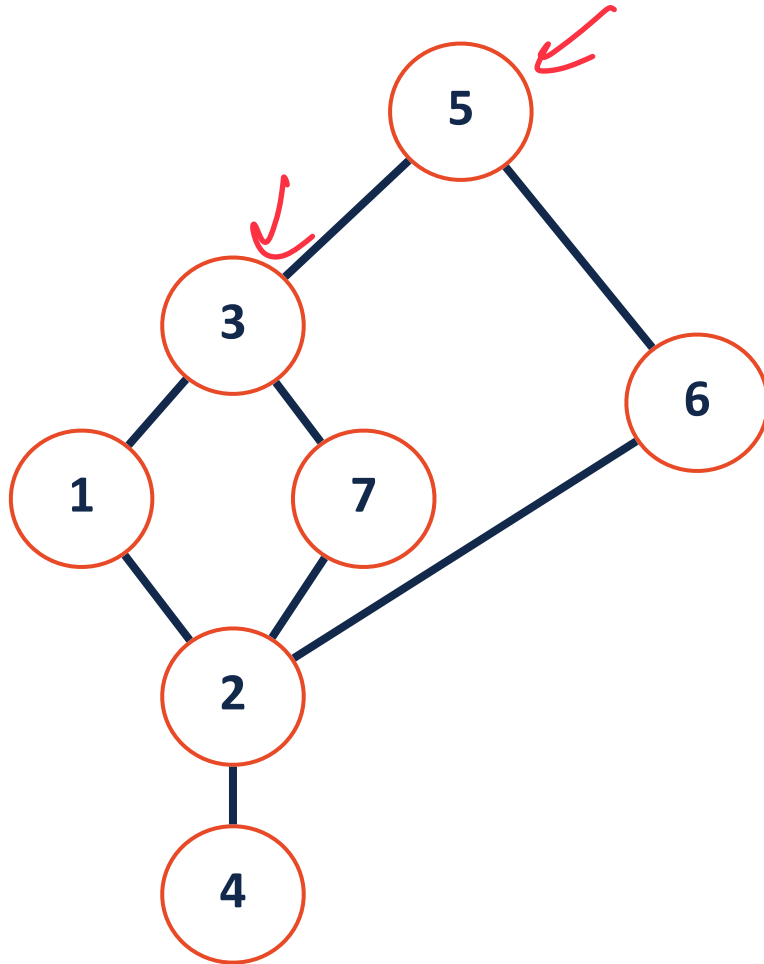
1. A common vocabulary
2. Graph implementations *Next week*
3. Graph traversals *Next next week*
4. Graph algorithms *wrap up*



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

↳ usually storing information!

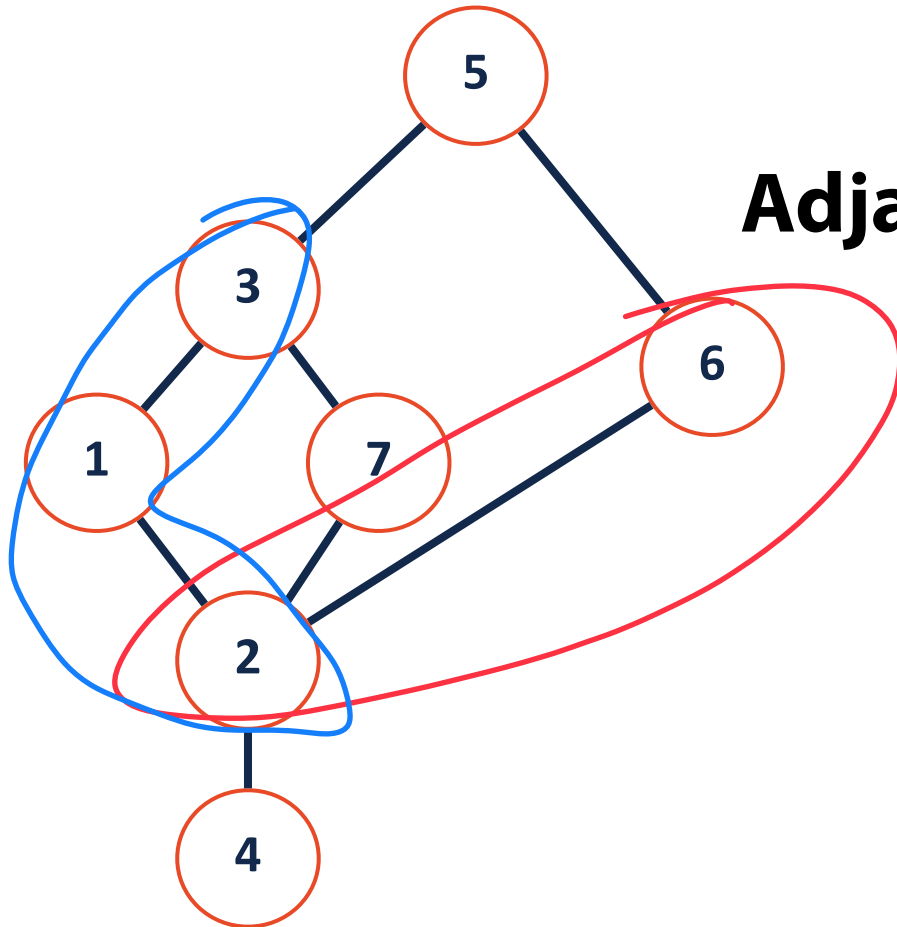
Edges: The connections between nodes

Defined by two endpoints

Graph Vocabulary

Degree: # of edges touching a vertex

↳ can be in-degree (incoming) or out-degree (outgoing)



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



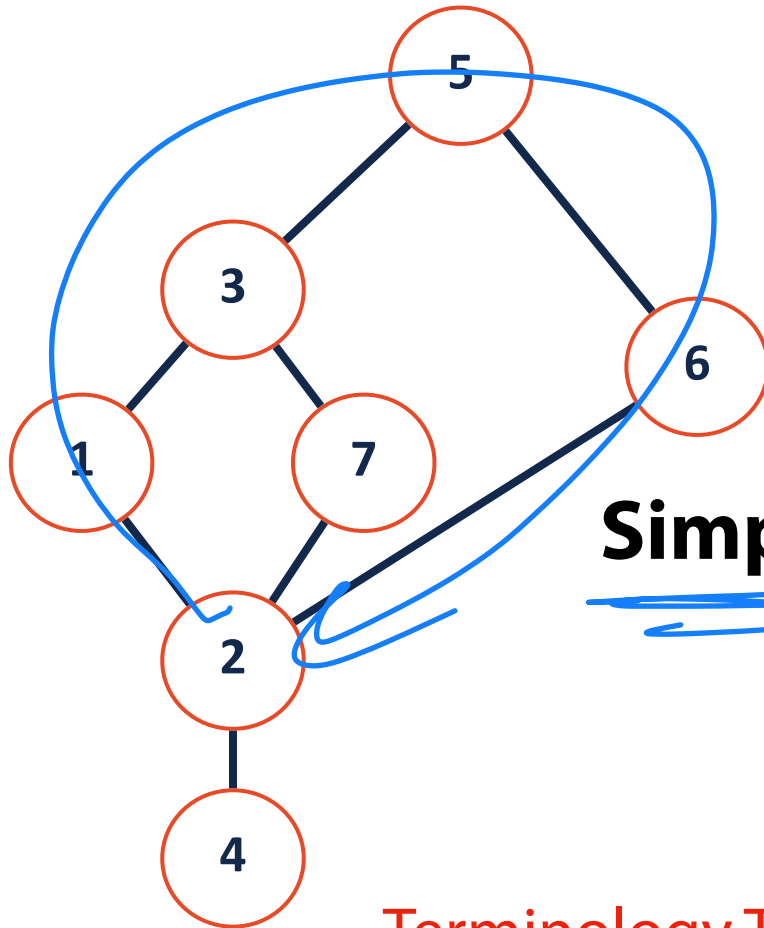
Path: A sequence of vertices (or edges) between two nodes

2 - 1 - 3

[2-1], [1-3]

Graph Vocabulary

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



Cycle: A path from a node to itself

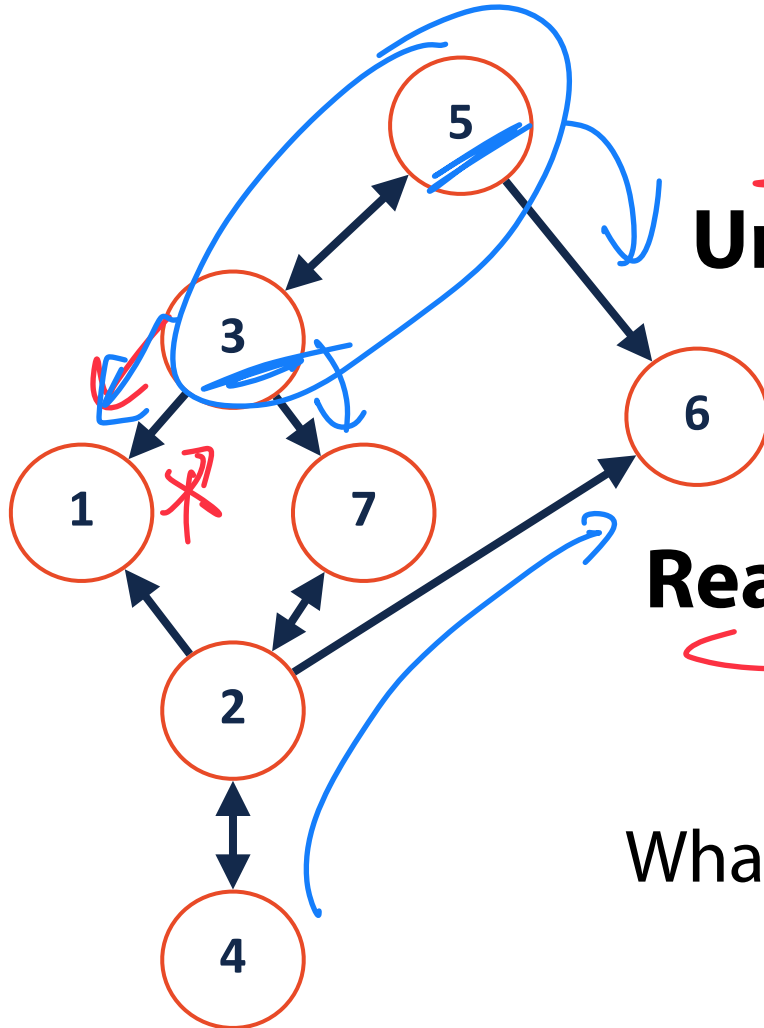
Simple Graph: No self-loops or multi-edges



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

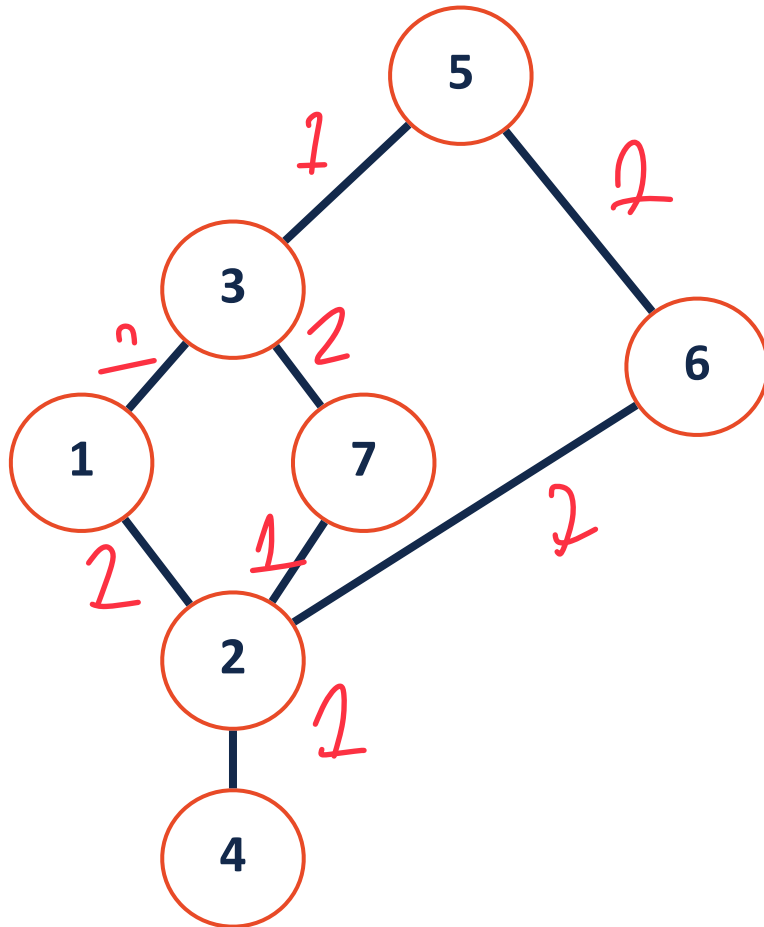
Reachability: v_2 is reachable from v_1 if there is a path from v_1 to v_2

What nodes are not reachable from 4? $\{ 5 \}$

Graph Vocabulary

All weights = 1

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



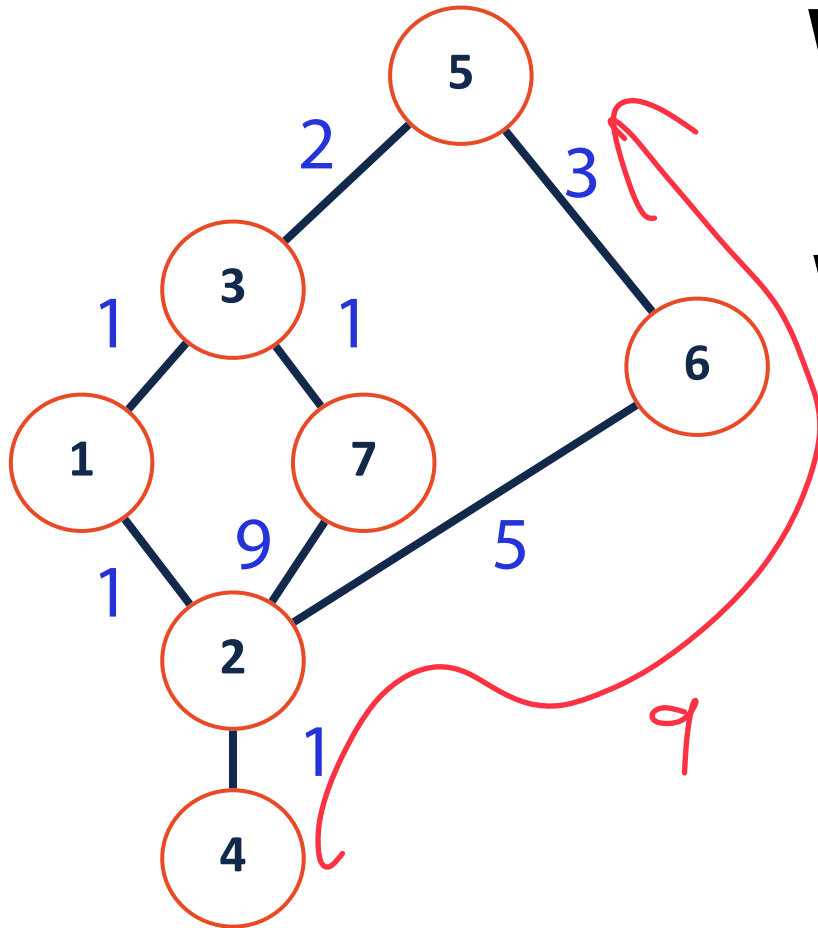
Weights: A value associated with an edge

What is the shortest path from 4 to 5?

↪ fewest edges from start to end

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?

4 - 2 - 1 - 3 - 5

Path length 5

Graph Vocabulary

$$G = (V, E)$$

$$|V| = n$$

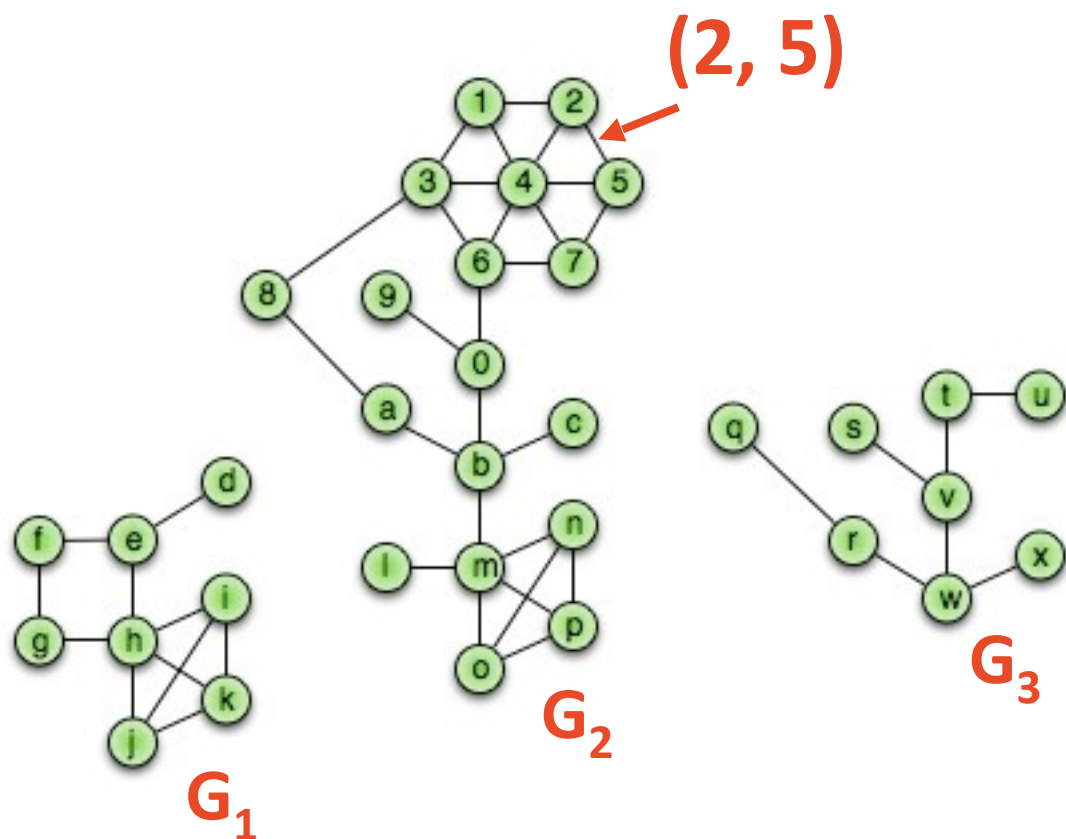
$$|E| = m$$

Subgraph(G): 

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

$$V' \subseteq V, E' \subseteq E, \text{ and}$$

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



Graph Vocabulary

$$G = (V, E)$$

$$|V| = n$$

$$|E| = m$$

Subgraph(G):

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

$$V' \subseteq V, E' \subseteq E, \text{ and}$$

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

Join Code: 225

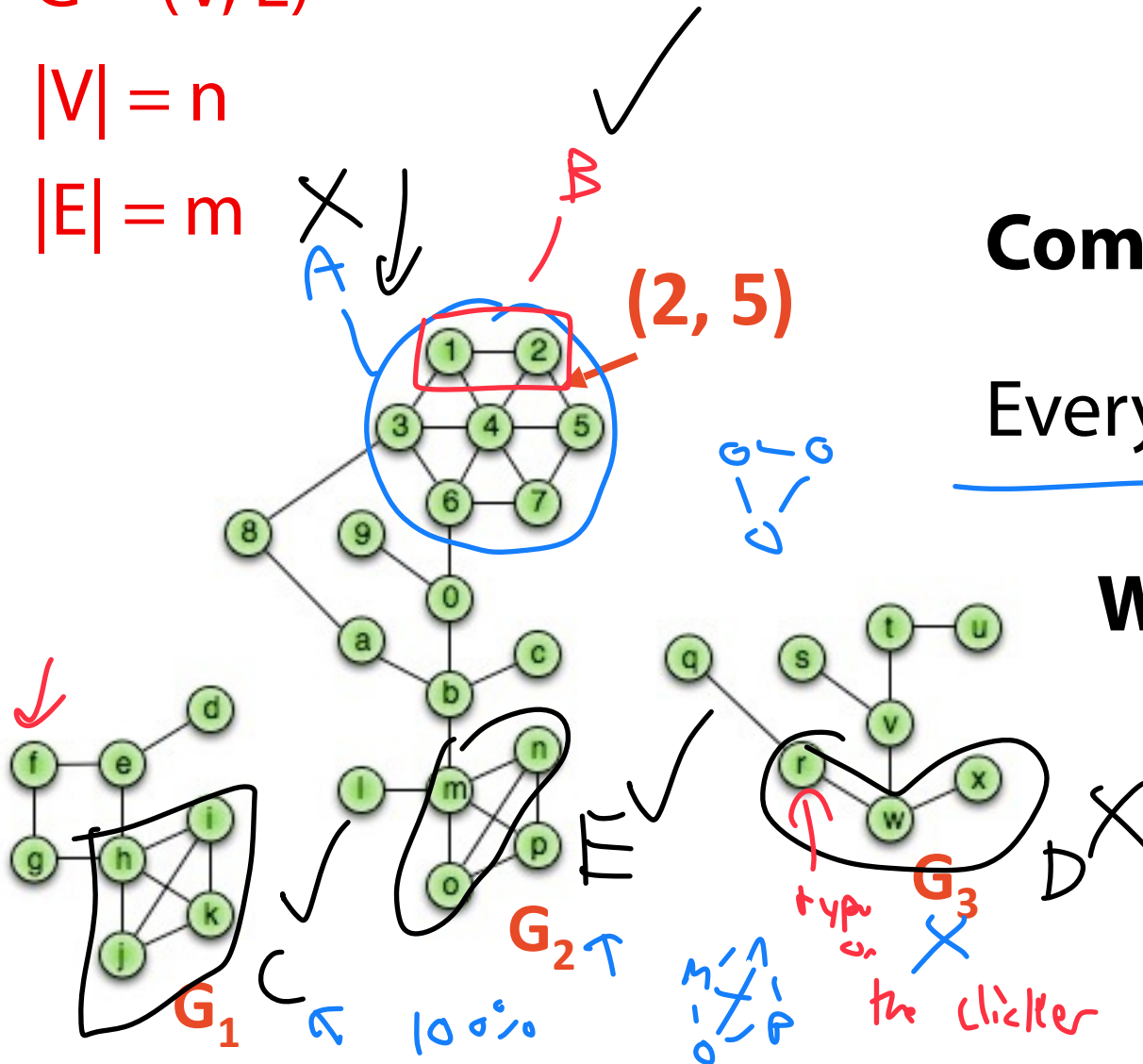


Complete Subgraph:

Every pair of vertices are adjacent

Which are complete subgraphs?

All 2 vertices 1 edge subgraphs
are complete



Graph Vocabulary

$$G = (V, E)$$

$$|V| = n$$

$$|E| = m$$

Subgraph(G):

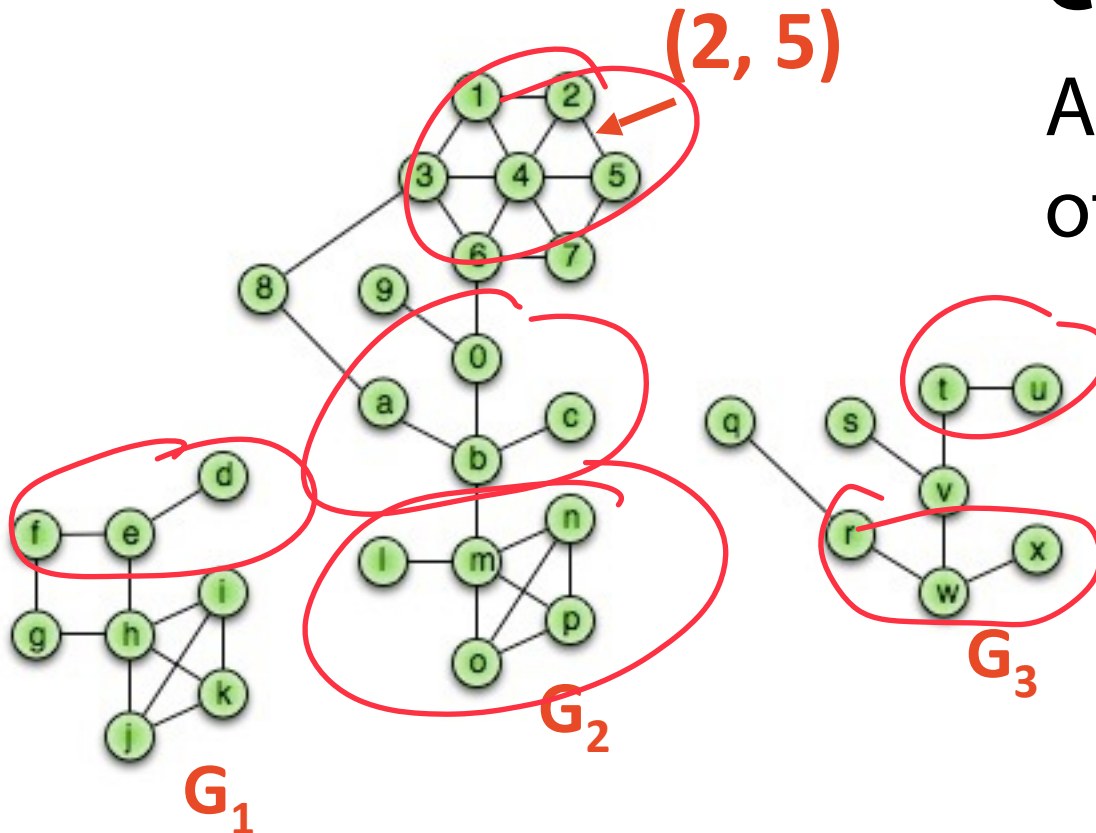
$$G' = (V', E'):$$

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

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

Connected Subgraph:

A path exists between every pair of vertices



$r-w-x$ is (connected) subgraph

Graph Vocabulary

$$G = (V, E)$$

$$|V| = n$$

$$|E| = m$$

Subgraph(G):

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

$$V' \in V, E' \in E, \text{ and}$$

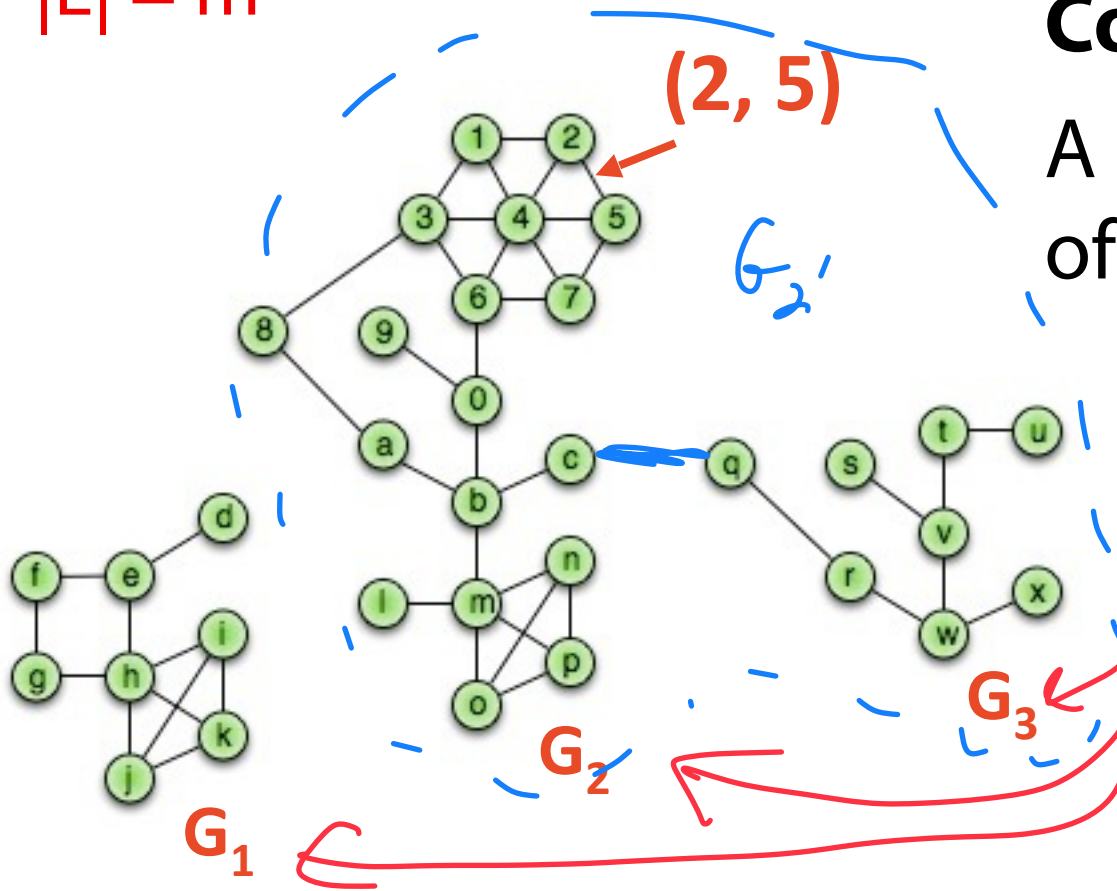
$$(u, v) \in E' \rightarrow u \in V', v \in V'$$

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):

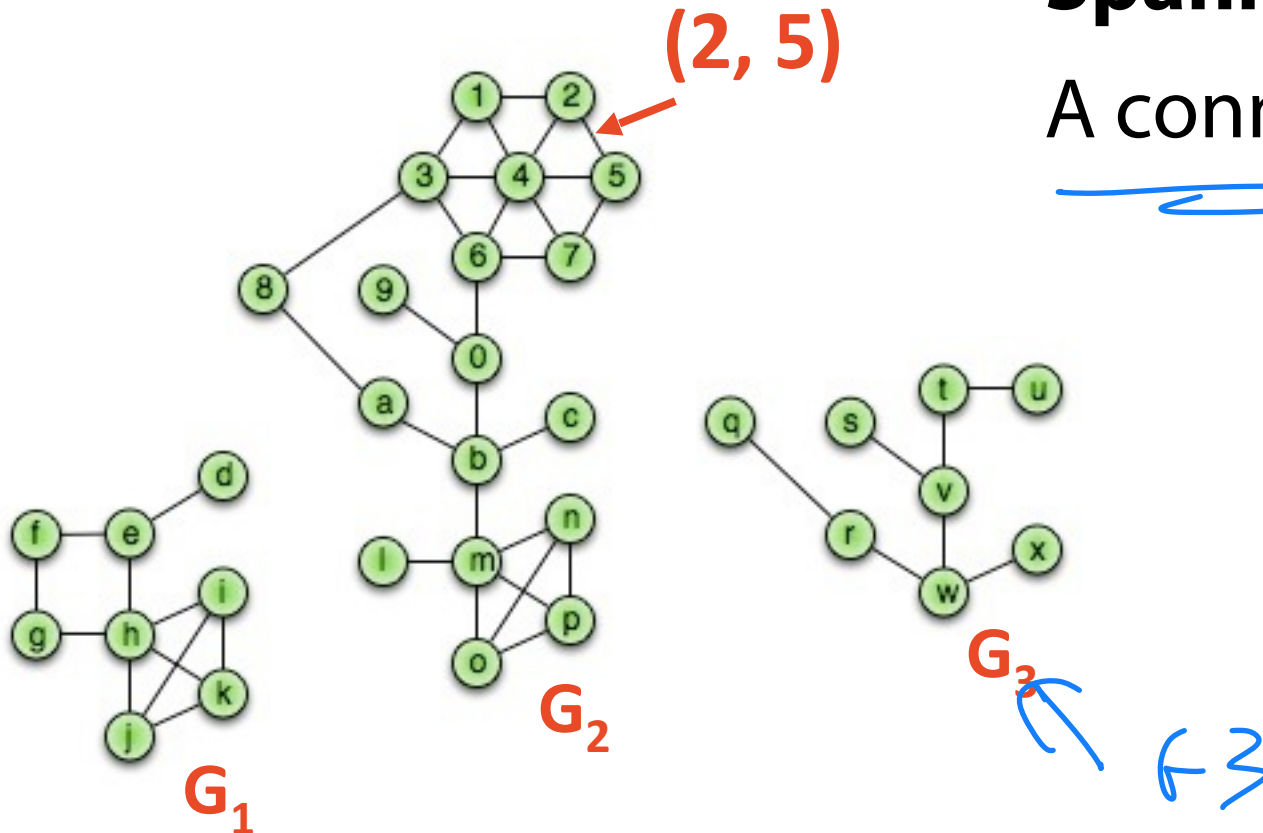
$$G' = (V', E'):$$

$$V' \in V, E' \in E, \text{ and}$$

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

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

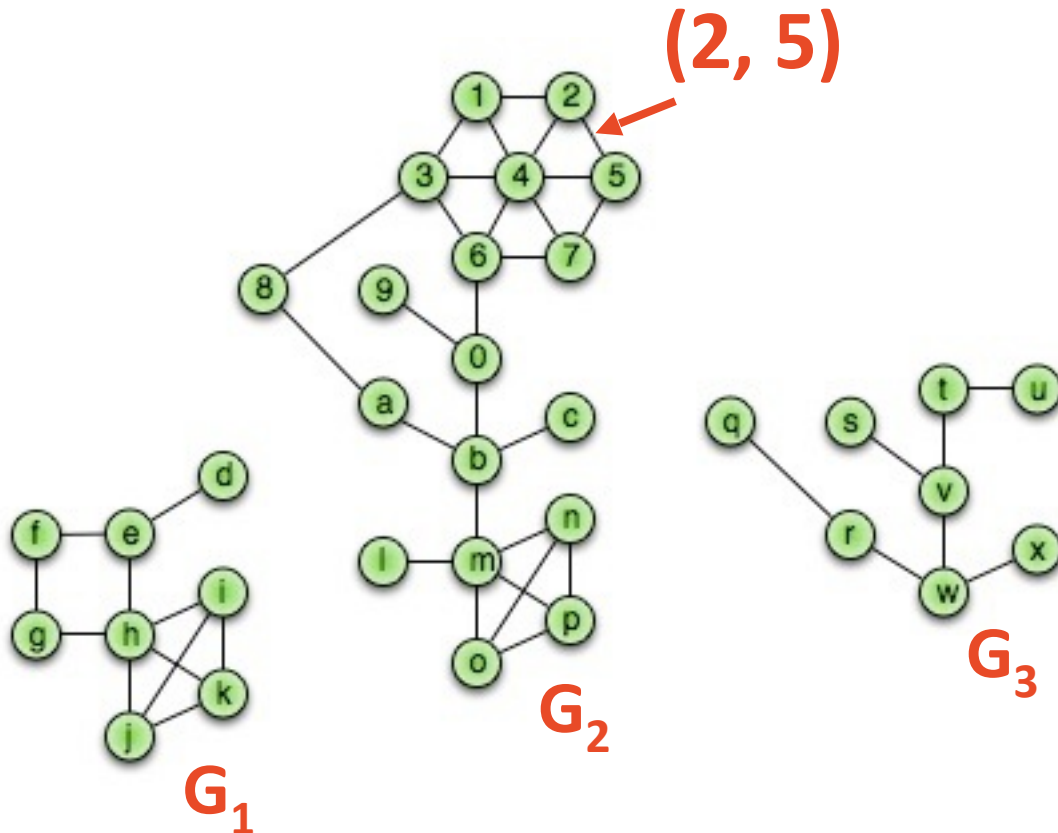
Complete

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.

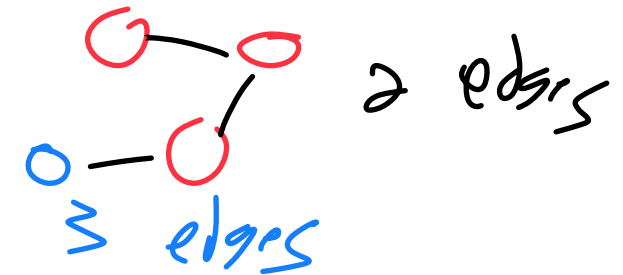


Join Code: 225

Whats the relationship between n and m ?

Minimum Edges:

Unconnected Graph: Zero



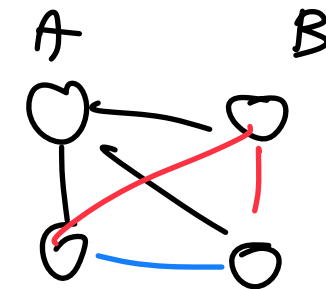
Connected (Simple) Graph: Given n :

$$m \geq n - 1$$

Maximum Edges:

Connected (Simple) Graph:

~~no~~ No more than one edge between pairs



$$\sum_{v \in V} \deg(v) = (N-1) + (N-2) + (N-3) + \dots = n \frac{(n-1)}{2} \approx O(n^2)$$

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?

↳ Individuals

What is an edge?

↳ A DM

Are the edges directed or undirected?

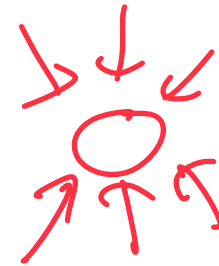
55%

45%

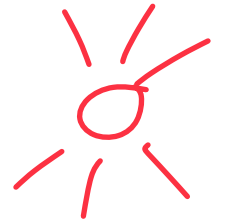
Are the edges weighted or unweighted?

50%

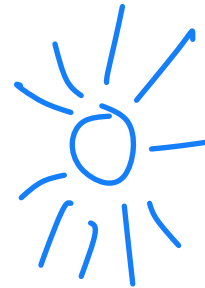
50%



vs

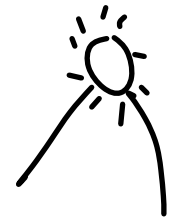


unweighted



weighted

weighted



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**

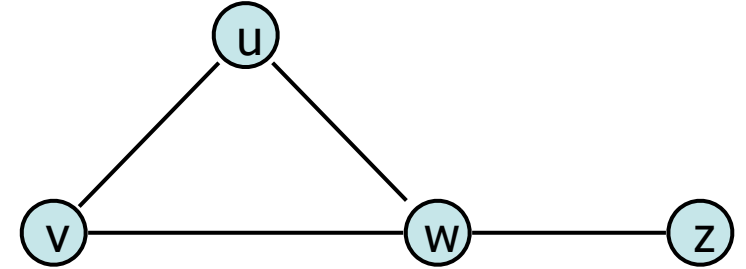
Graph Implementation

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

Vertex:

Skip

Edge:



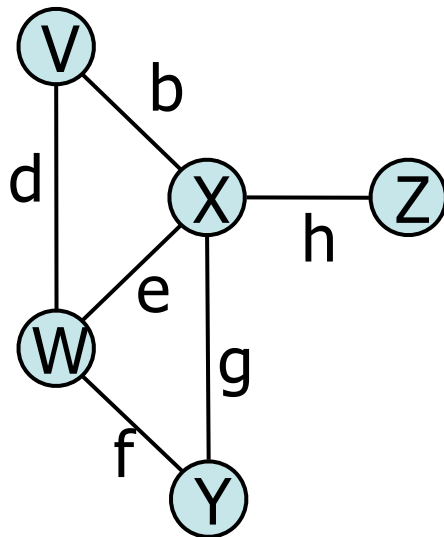
What information do we want to be able to find out quickly?

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);



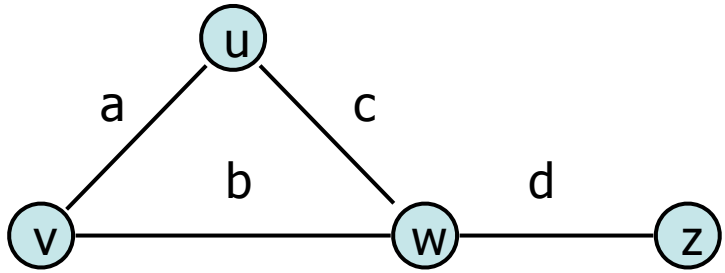
Find()

find vertex (?)



Graph Implementation Idea

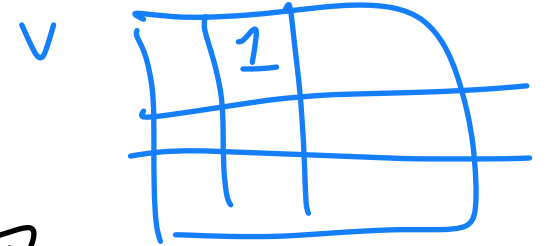
How to store Set of V & Set of E ?



Vertex Array

Edge Array

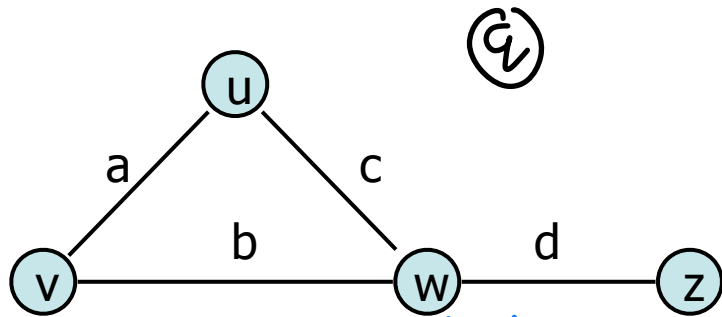
Adjacency Matrix



Map is kind of this

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

The equivalent of an 'unordered' data structure ** if we are connected!*



unordered edges

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

Vertex Storage:

Optional * b/c all vertices in
Not optional if not connected

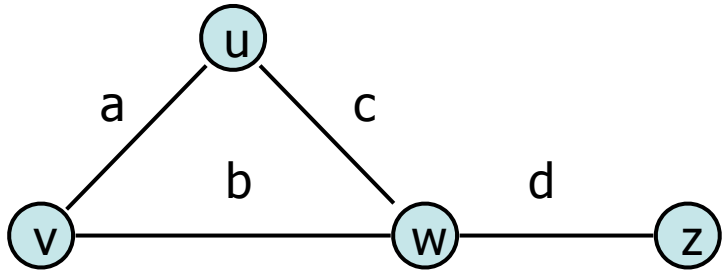
Edge Storage:

create list of vertices from list of edges
List of tuples (start, stop, weight)

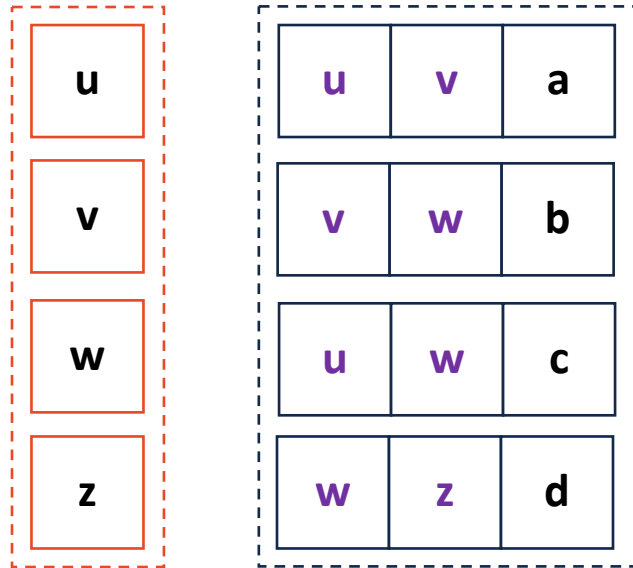
Optional!
↓

This strategy is easy to store (minimal space) & good at insert
→ Memory

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

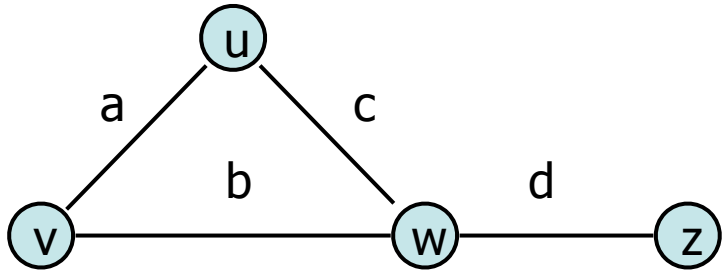


getEdges(Vertex v)

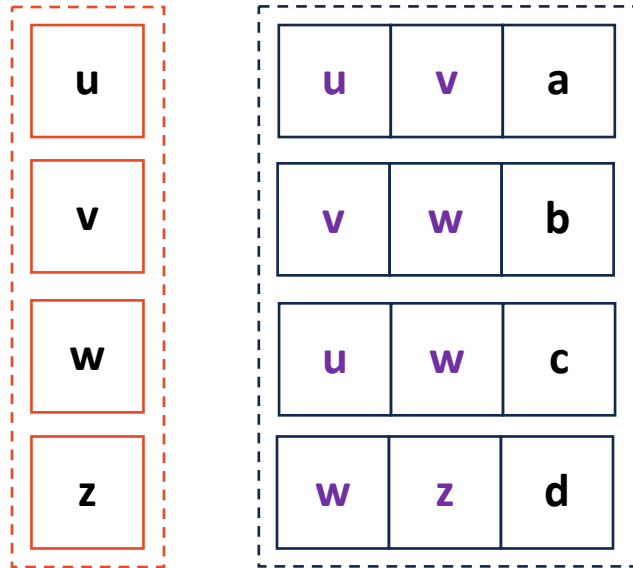


areAdjacent(Vertex v1, Vertex v2)

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

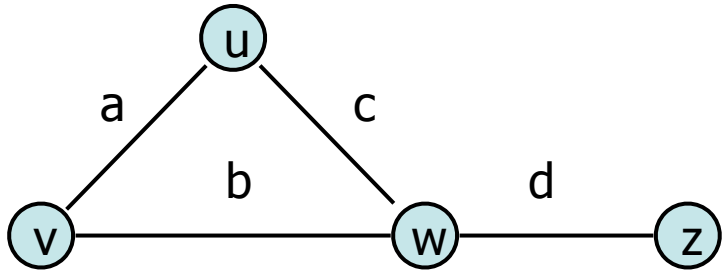


insertVertex(K key)

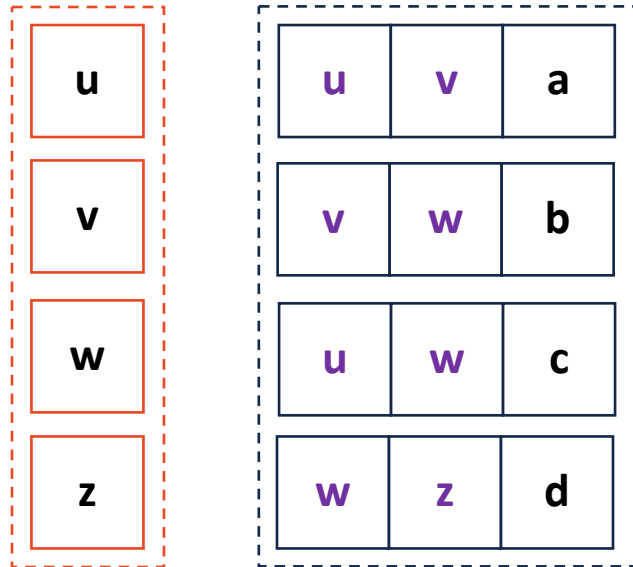


removeVertex(Vertex v)

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



insertEdge(Vertex v1, Vertex v2, K key)



removeEdge(Vertex v1, Vertex v2)

Graph Implementation: Edge List



Pros:

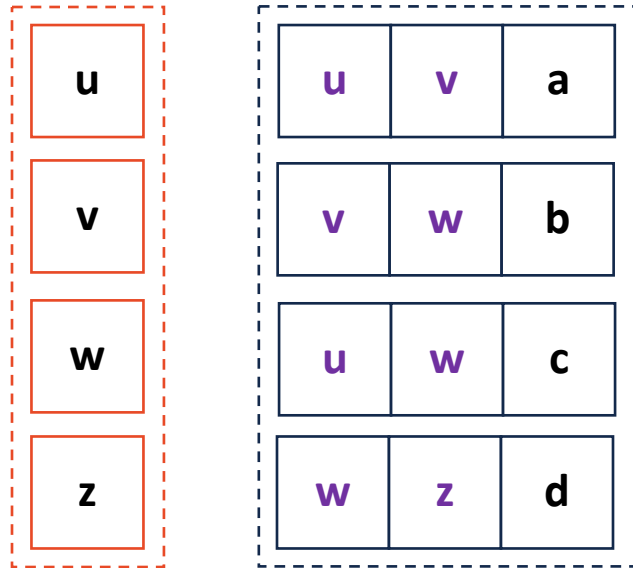
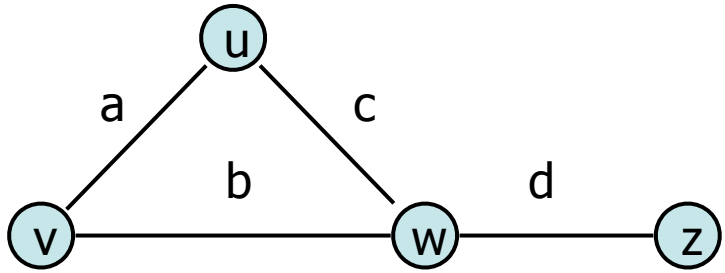
Cons:

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				