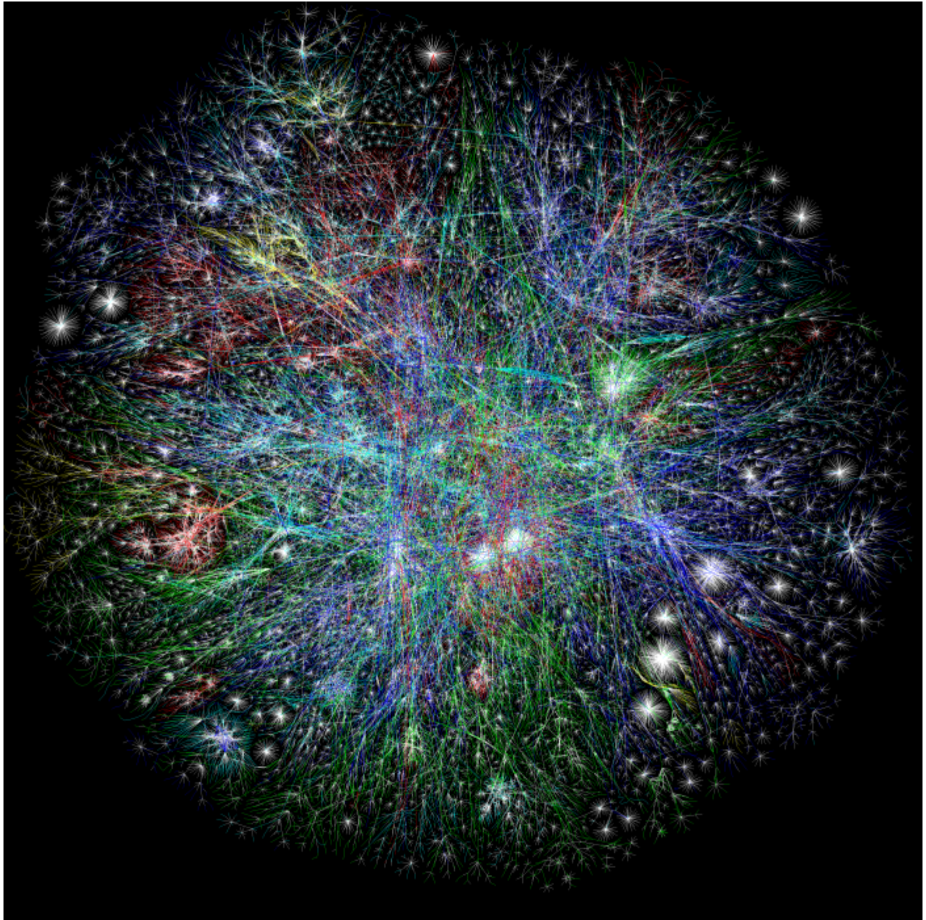
CS 400

Graphs: Intro

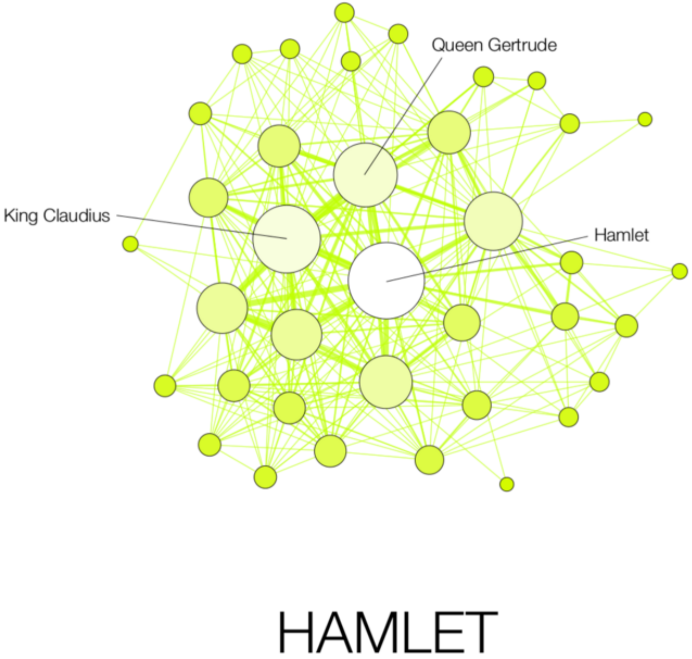
ID: 12-01

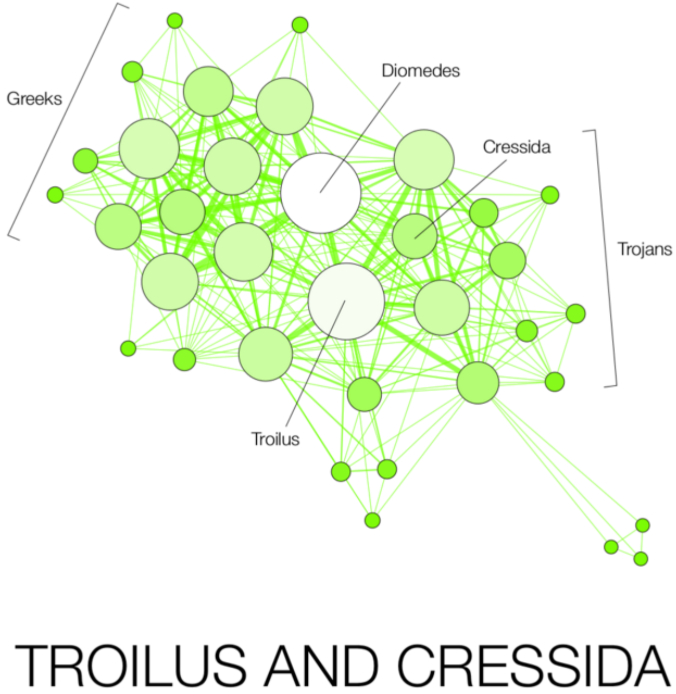


The Internet 2003

The OPTE Project (2003)

Map of the entire internet; nodes are routers; edges are connections.

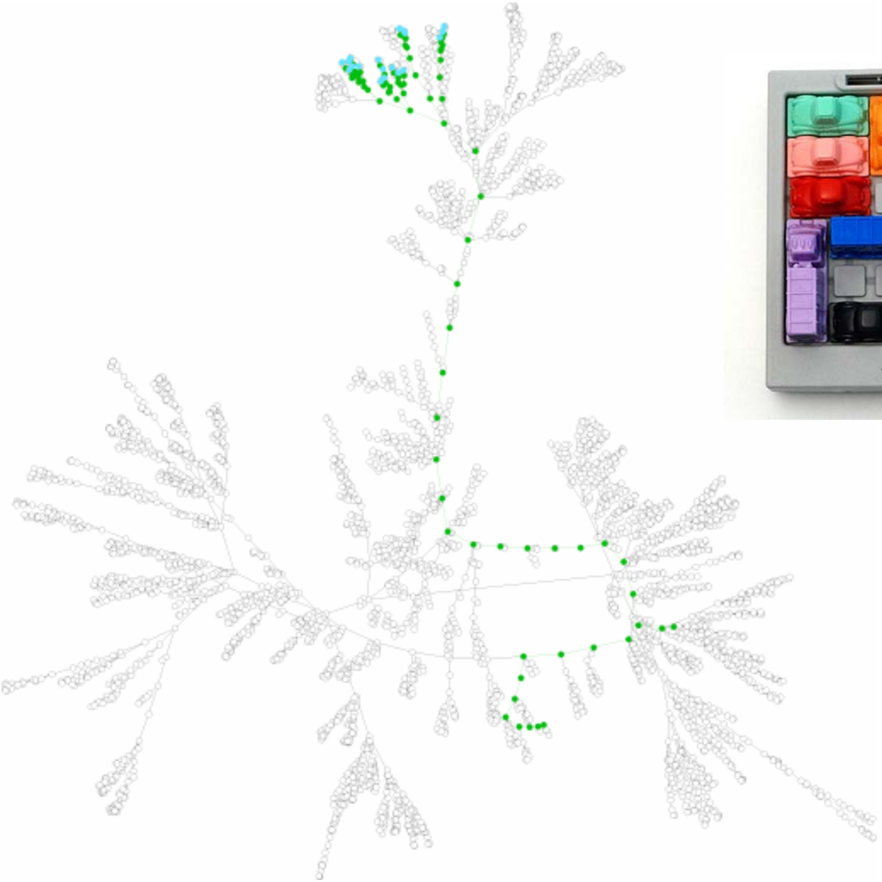




Who's the real main character in Shakespearean tragedies?

Martin Grandjean (2016)

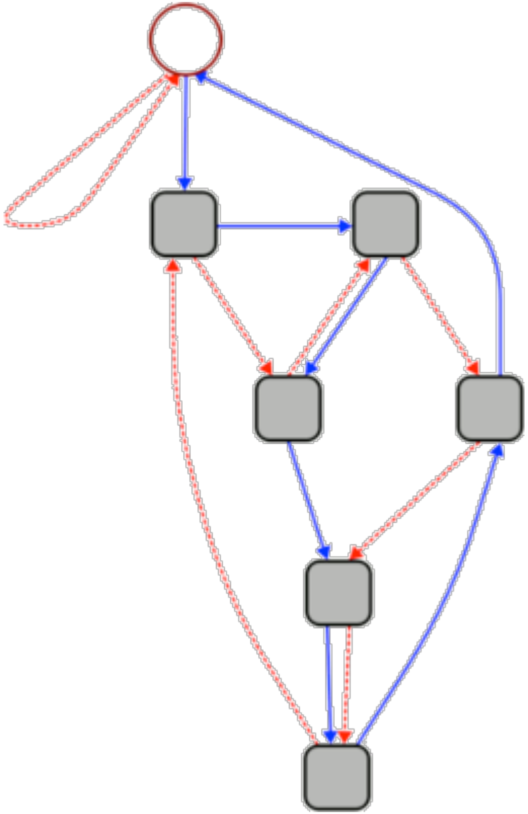
https://www.pbs.org/newshour/arts/whos-the-real-main-character-in-shakespearen-tragedies-heres-what-the-data-say



“Rush Hour” Solution

Unknown Source

Presented by Cinda Heeren, 2016



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

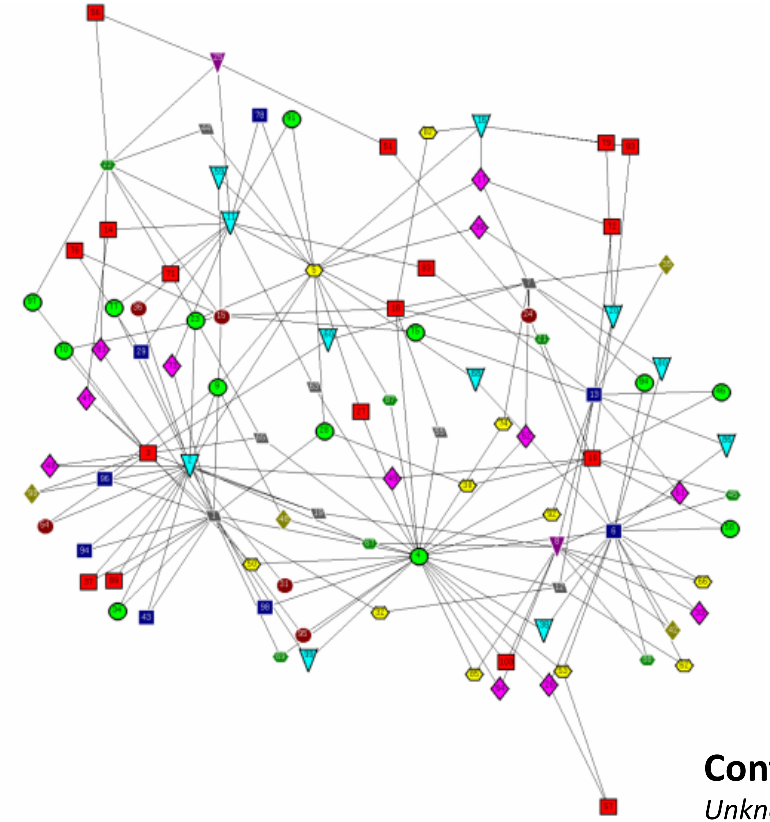
1. Start at the circle node at the top.
2. For each digit **d** in the given number, follow **d** blue (solid) edges in succession. As you move from one digit to the next, follow **1** red (dashed) edge.
3. If you end up back at the circle node, your number is divisible by 7.

3703

“Rule of 7”

Unknown Source

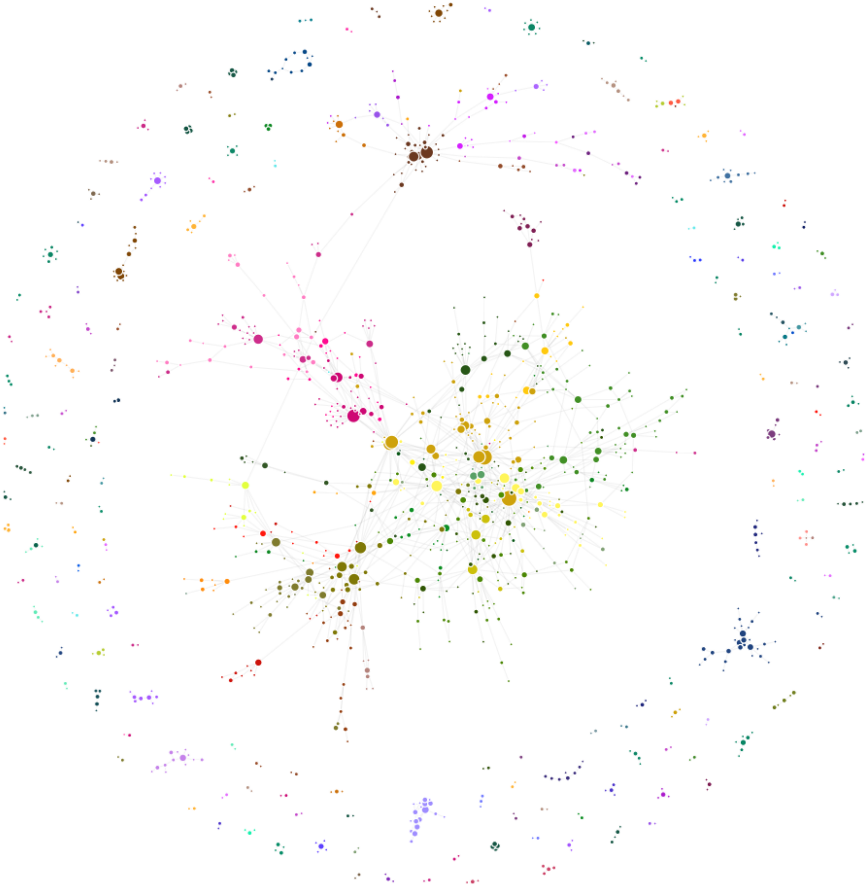
Presented by Cinda Heeren, 2016



Conflict-Free Final Exam Scheduling Graph

Unknown Source

Presented by Cinda Heeren, 2016

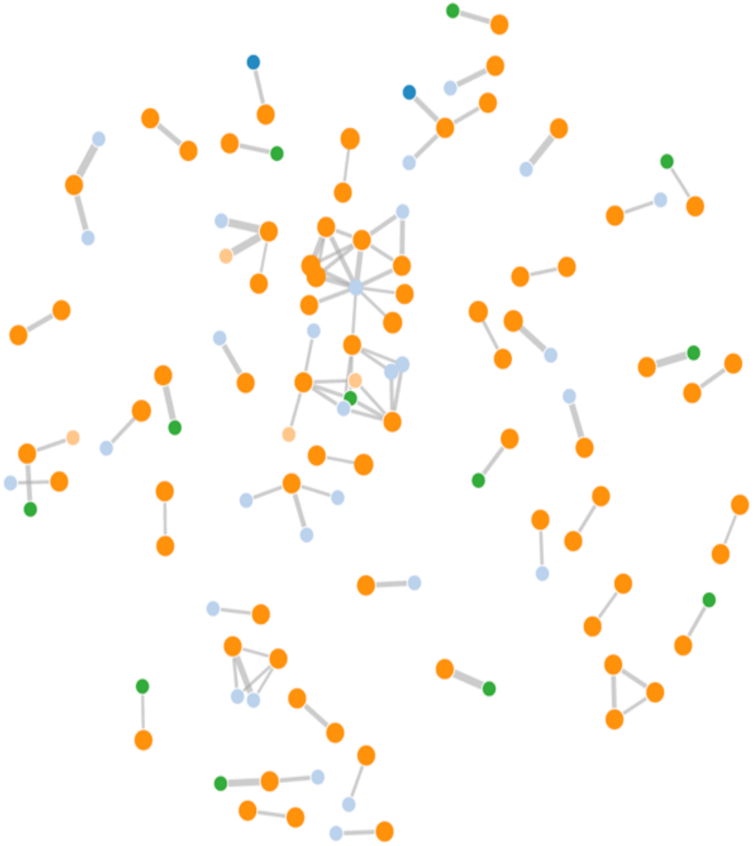


Class Hierarchy At University of Illinois Urbana-Champaign

A. Mori, W. Fagen-Ulmschneider, C. Heeren

Graph of every course at UIUC; nodes are courses, edges are prerequisites

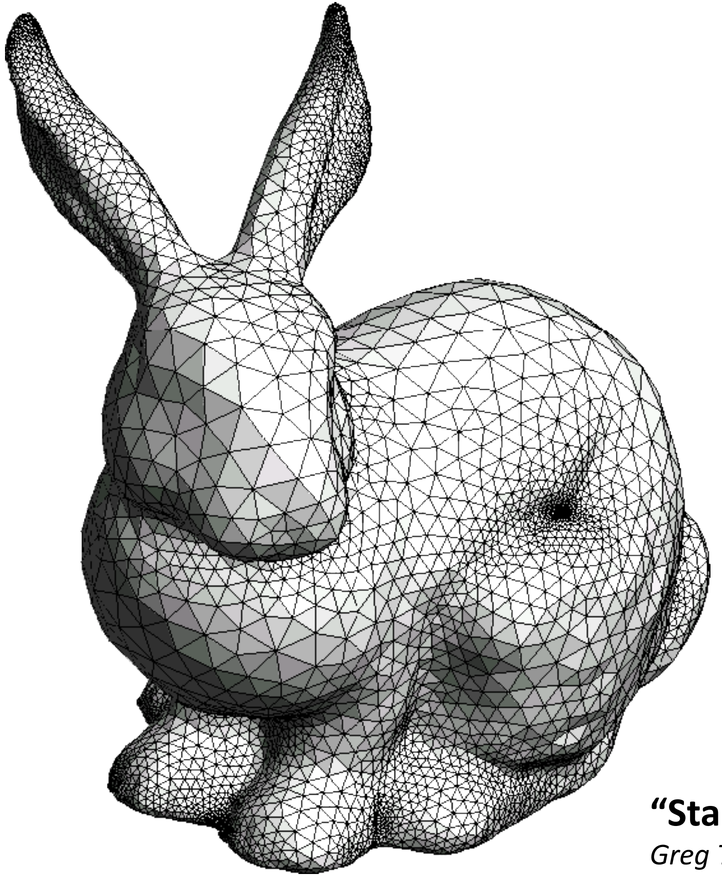
http://waf.cs.illinois.edu/discovery/class_hierarchy_at_illinois/



MP Collaborations in CS 225

Unknown Source

Presented by Cinda Heeren, 2016



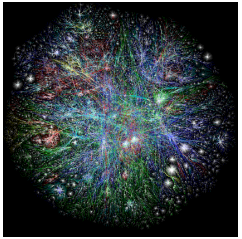
“Stanford Bunny”

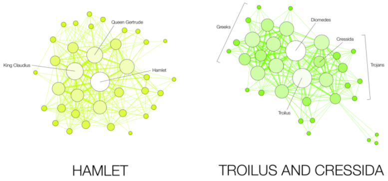
Greg Turk and Mark Levoy (1994)

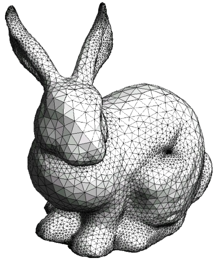
Graphs

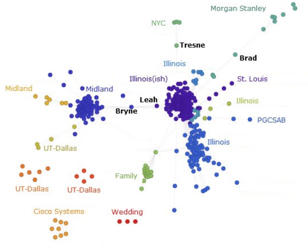
To study all of these structures:

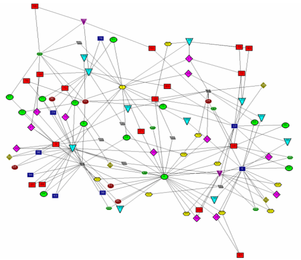
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms

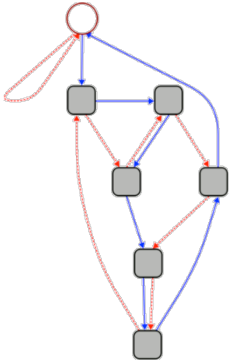


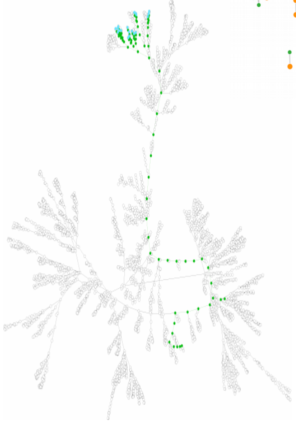


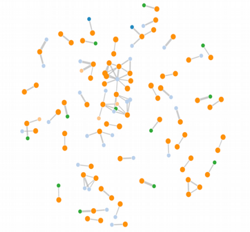












CS 400

Graphs: Vocabulary

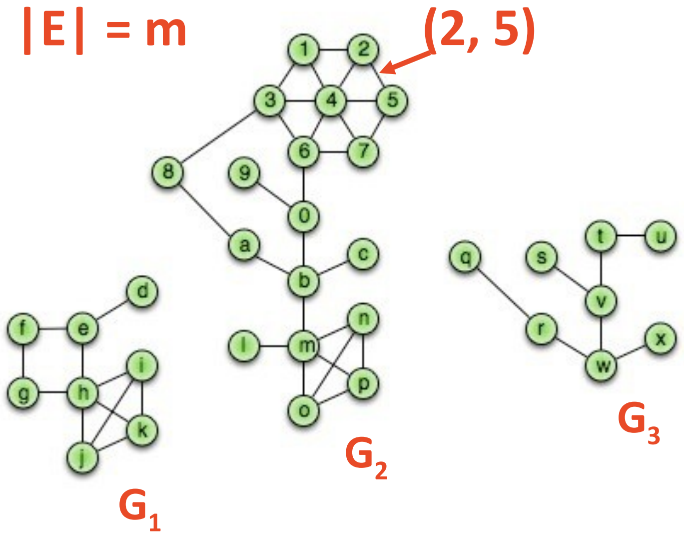
ID: 12-02

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Incident Edges:
$I(v) = \{ (x, v) \text{ in } E \}$

Degree(v): $|I|$

Adjacent Vertices:
$A(v) = \{ x : (x, v) \text{ in } E \}$

Path(G_2): Sequence of vertices connected by edges

Cycle(G_1): Path with a common begin and end vertex.

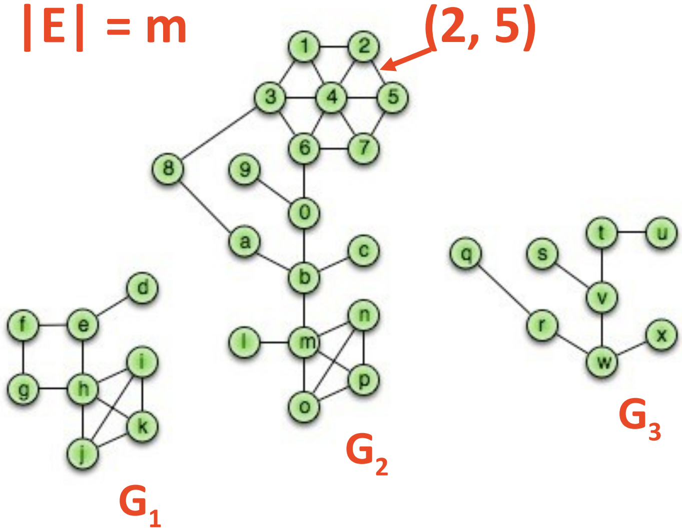
Simple Graph(G): A graph with no self loops or multi-edges.

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Subgraph(G):

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

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

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

Complete subgraph(G)

Connected subgraph(G)

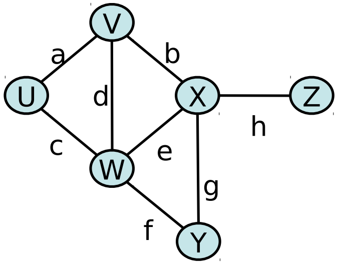
Connected component(G)

Acyclic subgraph(G)

Spanning tree(G)

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

How many edges?



Minimum edges:

Not Connected: ∅ (subgraphs)

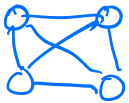
Connected*: $n-1$

Maximum edges:

Simple: → $n(n-1)/2$

Not simple: infinite ∞

e.g.



4 nodes

$4(4-1)/2 = 6$

$\boxed{\text{6 edges}}$

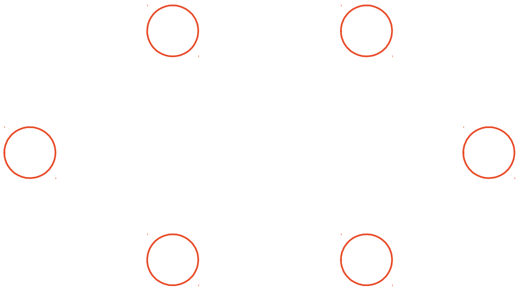
$\sum_{v \in V} \deg(v) =$

CS 400

Graphs: Connected Graphs

ID: 12-03

Connected Graphs



Proving the size of a minimally connected graph

Theorem:
Every minimally connected graph **G=(V, E)** has **|V|-1** edges.

Thm: Every minimally connected graph **G=(V, E)** has **|V|-1** edges.

Proof: Consider an arbitrary, minimally connected graph **G=(V, E)**.

Lemma 1: Every connected subgraph of **G** is minimally connected.
(Easy proof by contradiction left for you.)

Inductive Hypothesis: For any **j < |V|**, any minimally connected graph of **j** vertices has **j-1** edges.

Suppose |V| = 1:

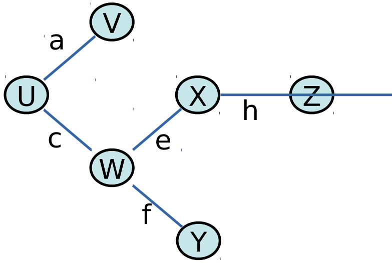
Definition: A minimally connected graph of 1 vertex has 0 edges.

Theorem: |V|-1 edges 1-1 = 0.

Suppose |V| > 1:

Choose any vertex **u** and let **d** denote the degree of **u**.

Remove the incident edges of **u**, partitioning the graph into ___ components: $\mathbf{C_0 = (V_0, E_0)}$, …, $\mathbf{C_d = (V_d, E_d)}$.



By Lemma 1, every component $\mathbf{C_k}$ is a minimally connected subgraph of **G**.

By our ________: ____________________.

Finally, we count edges:

CS 400

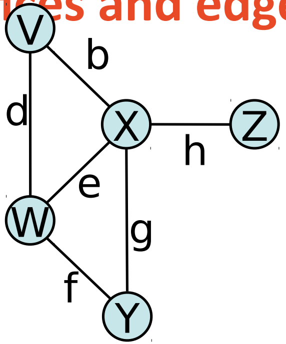
Graphs: Edge List Implementation

ID: 12-04

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);

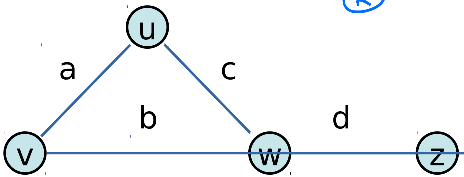
- incidentEdges(Vertex v);
- areAdjacent(Vertex v1, Vertex v2);

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

Graph Implementation: Edge List

(naive implementation to be improved on later)

K



u
v
w
z

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

insertVertex(K key); → O(1)* amortized

removeVertex(Vertex v); → O(1)

areAdjacent(Vertex v1, Vertex v2);

O(m) go through each edge slow

incidentEdges(Vertex v);

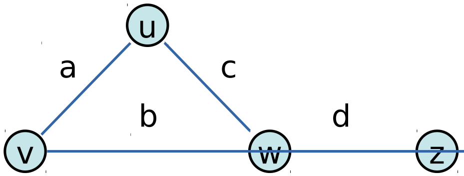
O(m) slow

CS 400

Graphs: Adjacency Matrix Implementation

ID: 12-05

Graph Implementation: Adjacency Matrix



insertVertex(K key); $O(1)^* \rightarrow O(n)$ slower

removeVertex(Vertex v); $O(n)$ slower

areAdjacent(Vertex v1, Vertex v2); $O(1)$ fast!

incidentEdges(Vertex v); $O(n)$ about same

0 = no edge 1 = edge

u
v
w
z

V	U	a
V	W	b
U	W	c
W	Z	d

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

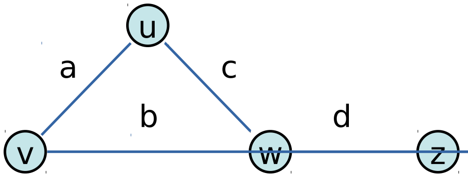
create pointers for each edge

CS 400

Graphs: Adjacency List Implementation

ID: 12-06

Graph Implementation: Adjacency List

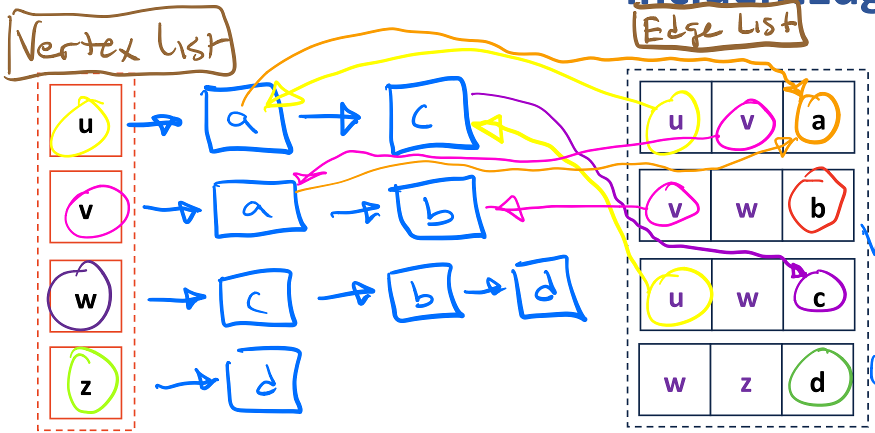


insertVertex(K key); $O(1)$

removeVertex(Vertex v); $O(\text{degrees of vertex})$

areAdjacent(Vertex v1, Vertex v2); $O\left(\min \text{ of } \left(\begin{matrix} deg(v1) \\ deg(v2) \end{matrix}\right)\right)$

incidentEdges(Vertex v); $O(deg(v))$



Vertex list points to Edge list and Edge list points back to Vertex list.

Complex but fast

✓ = best ✗ = worst

Expressed as O(f)	Edge List	Adjacency Matrix	Adjacency List
Space	n+m	n+m	n^2 ✗
insertVertex(v)	1 ✓	n	1 ✓
removeVertex(v)	m	n ✗	deg(v) ✓
insertEdge(v, w, k)	1 ✓	1 ✓	1 ✓
removeEdge(v, w)	1 ✓	1 ✓	1 ✓
incidentEdges(v)	m	n ✗	deg(v) ✓
areAdjacent(v, w)	m ✗	1 ✓	min(deg(v), deg(w))