CS 225

Data Structures

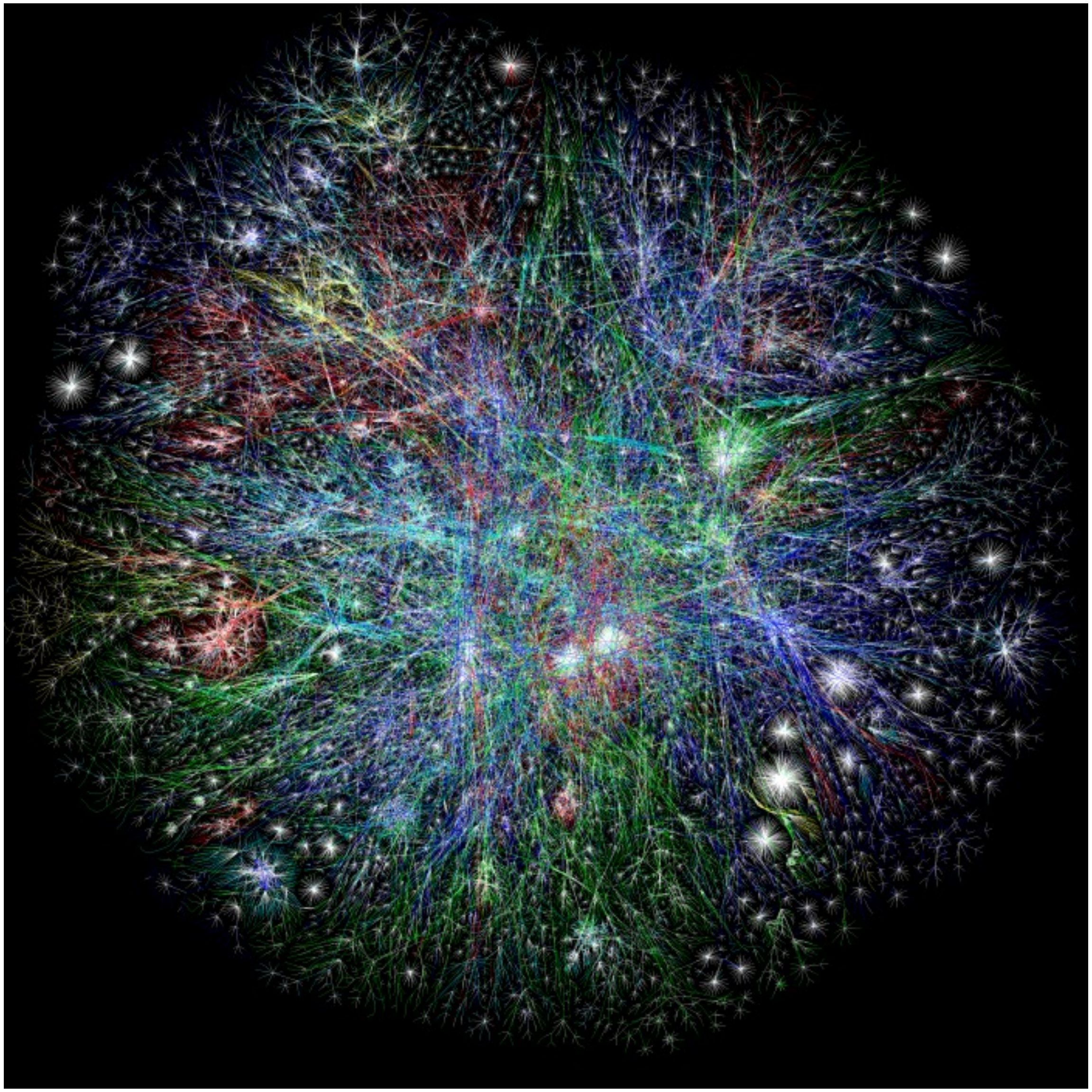
March 18– Graphs

G Carl Evans

In Review: Data Structures

Array
- Sorted Array
- Unsorted Array
  - Stacks
  - Queues
  - Priority Queues
    - Heaps
  - Disjoint Sets
    - UpTrees

Linked
- Doubly Linked List
- Trees
  - BTree
  - Binary Tree
    - Huffman Encoding
    - kd-Tree
    - AVL Tree

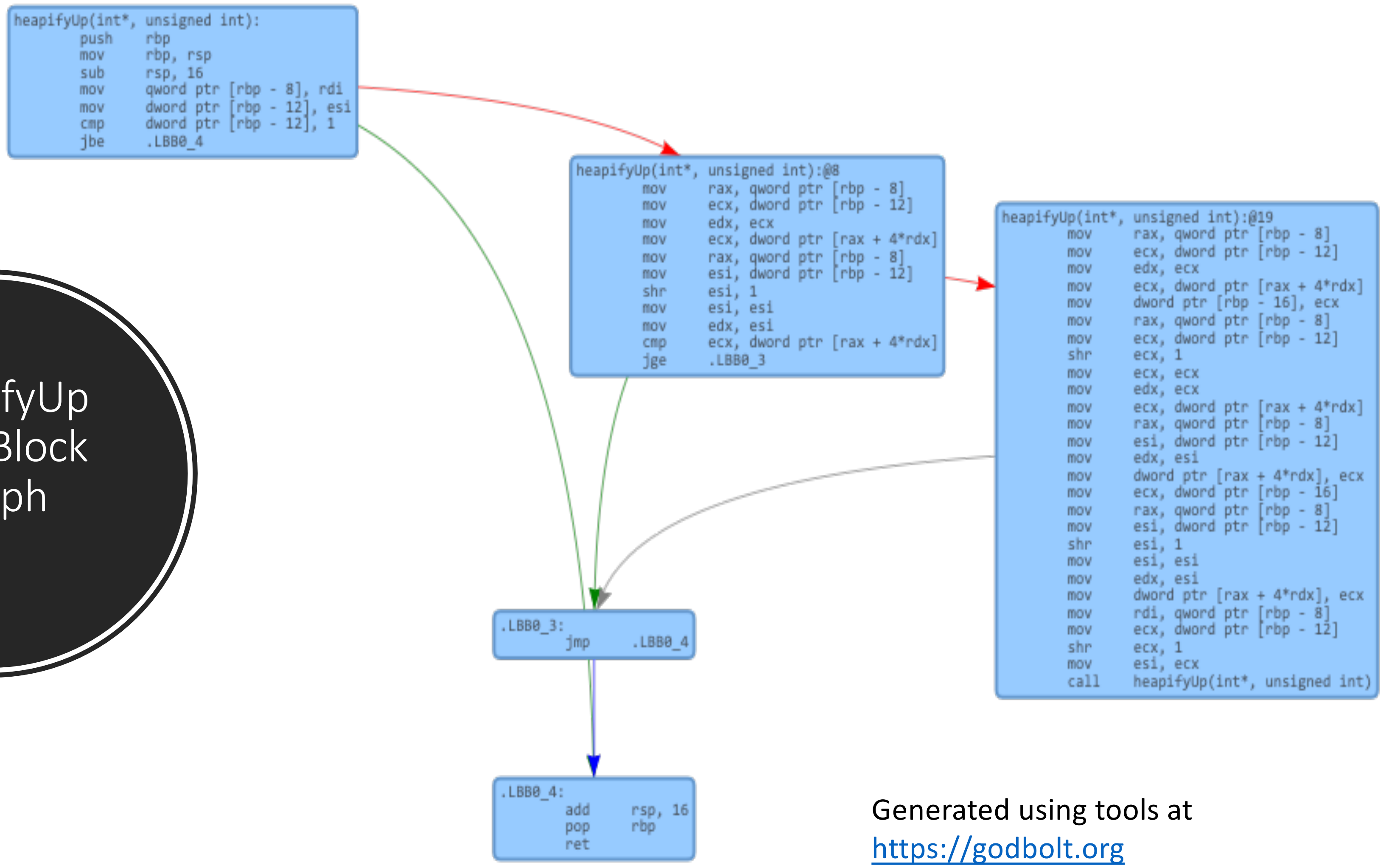


The Internet 2003

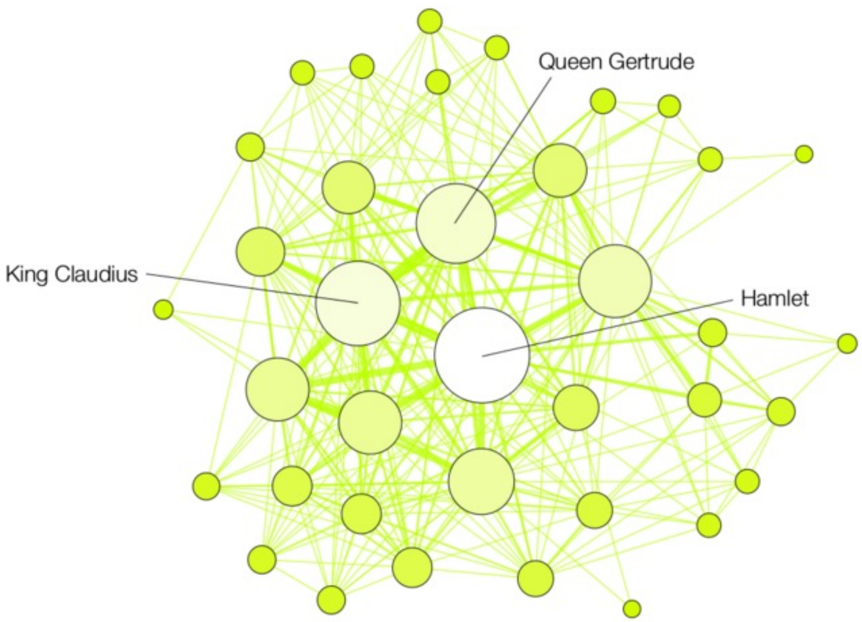
The OPTE Project (2003)

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

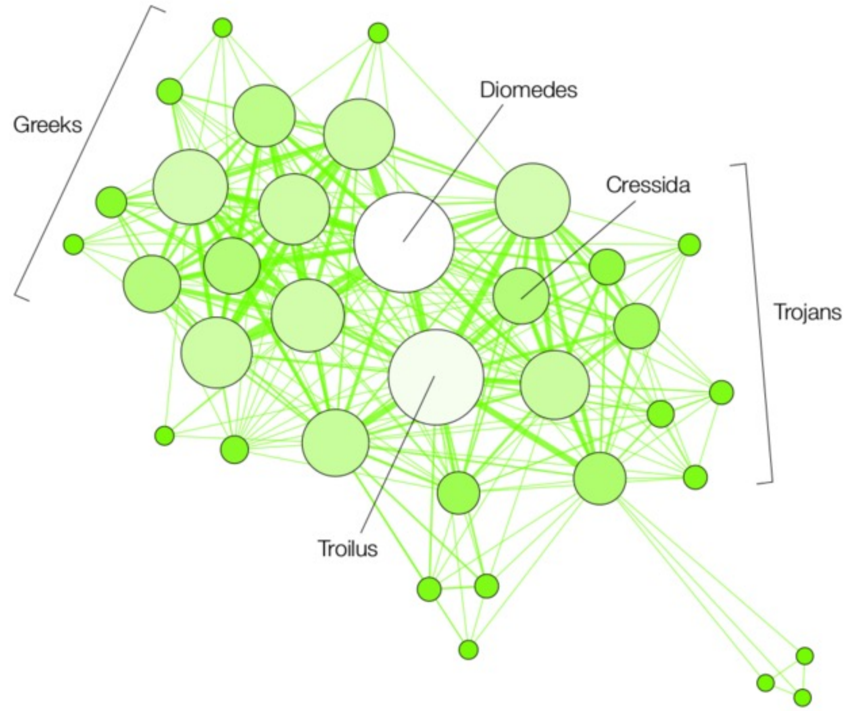
HeapifyUp BasicBlock Graph



Generated using tools at https://godbolt.org



HAMLET

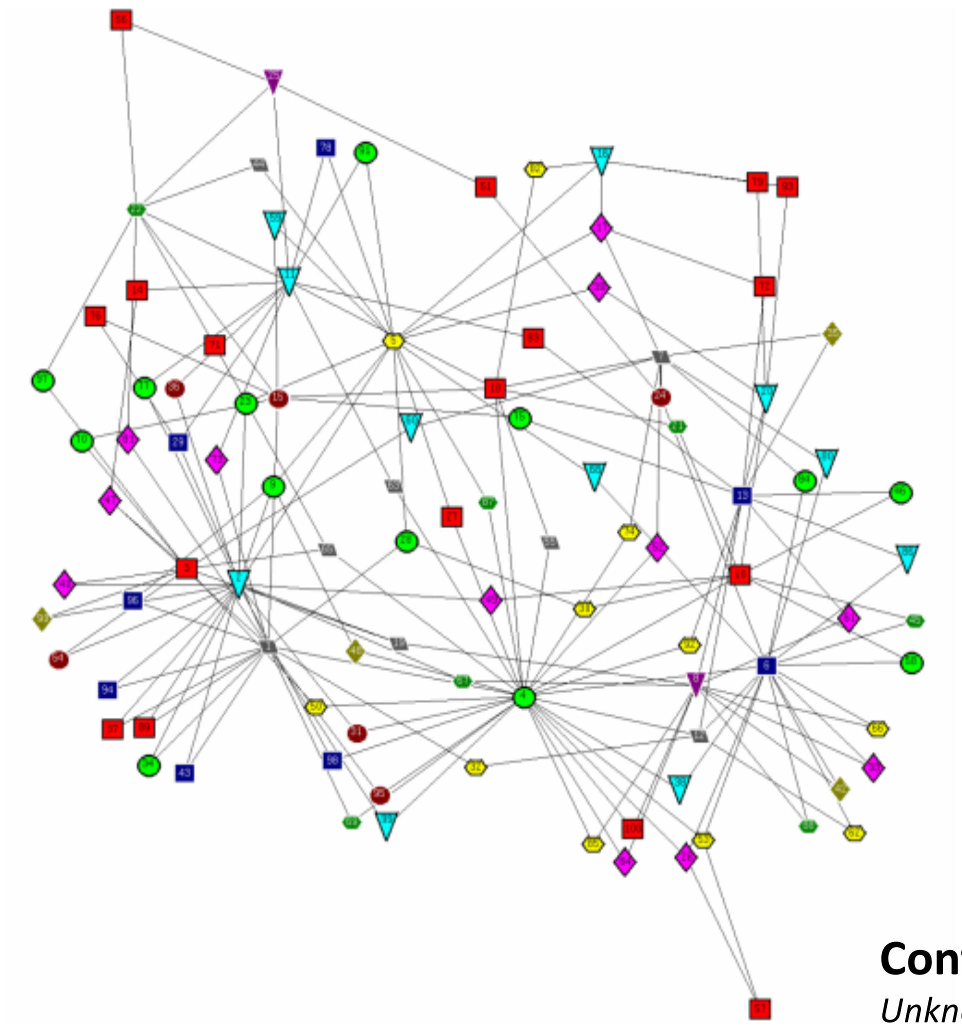


TROILUS AND CRESSIDA

Who's the real main character in Shakespearean tragedies?

Martin Grandjean (2016)

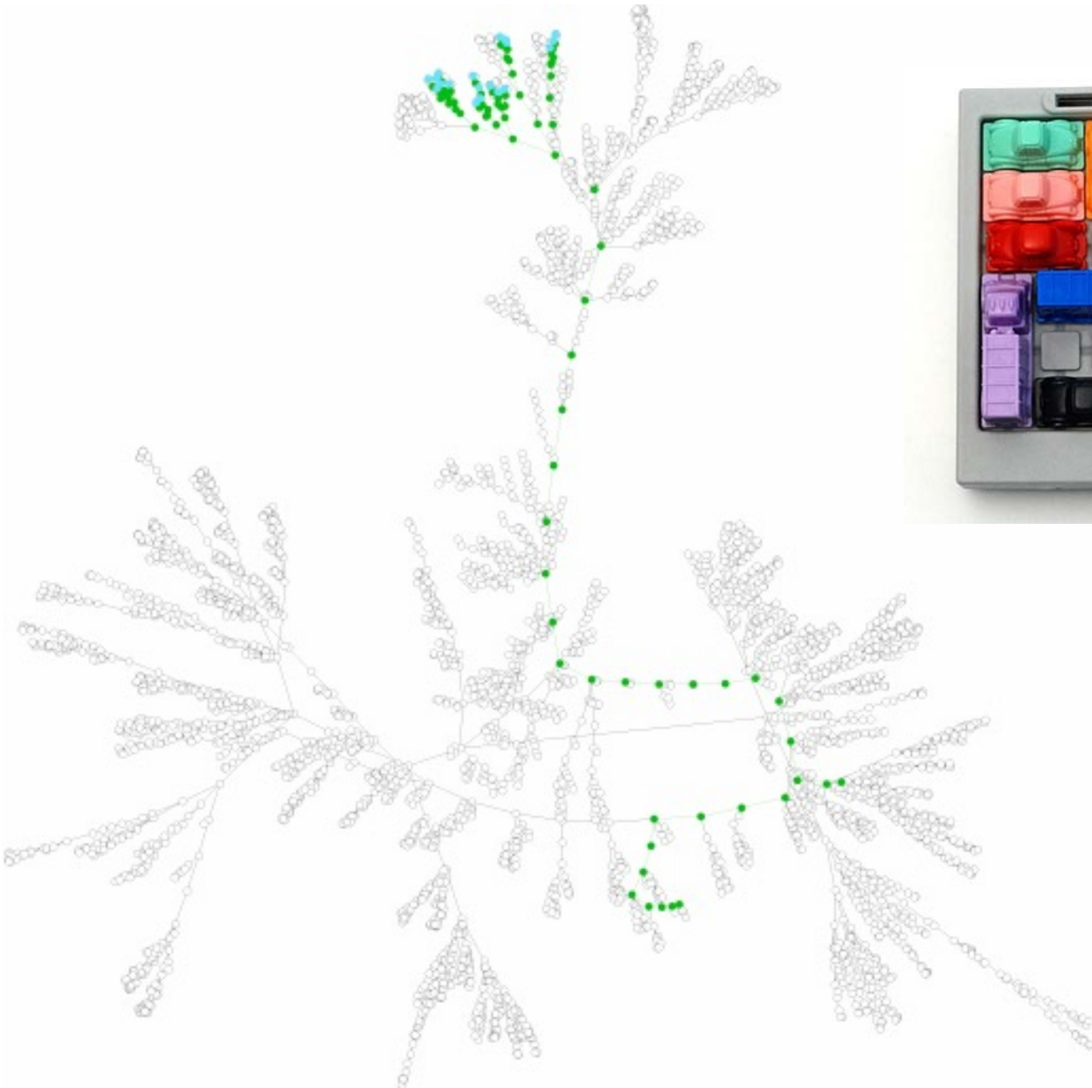
[*https://www.pbs.org/newshour/arts/whos-the-real-main-character-in-shakespearen-tragedies-heres-what-the-data-say*](https://www.pbs.org/newshour/arts/whos-the-real-main-character-in-shakespearen-tragedies-heres-what-the-data-say)



Conflict-Free Final Exam Scheduling Graph

Unknown Source

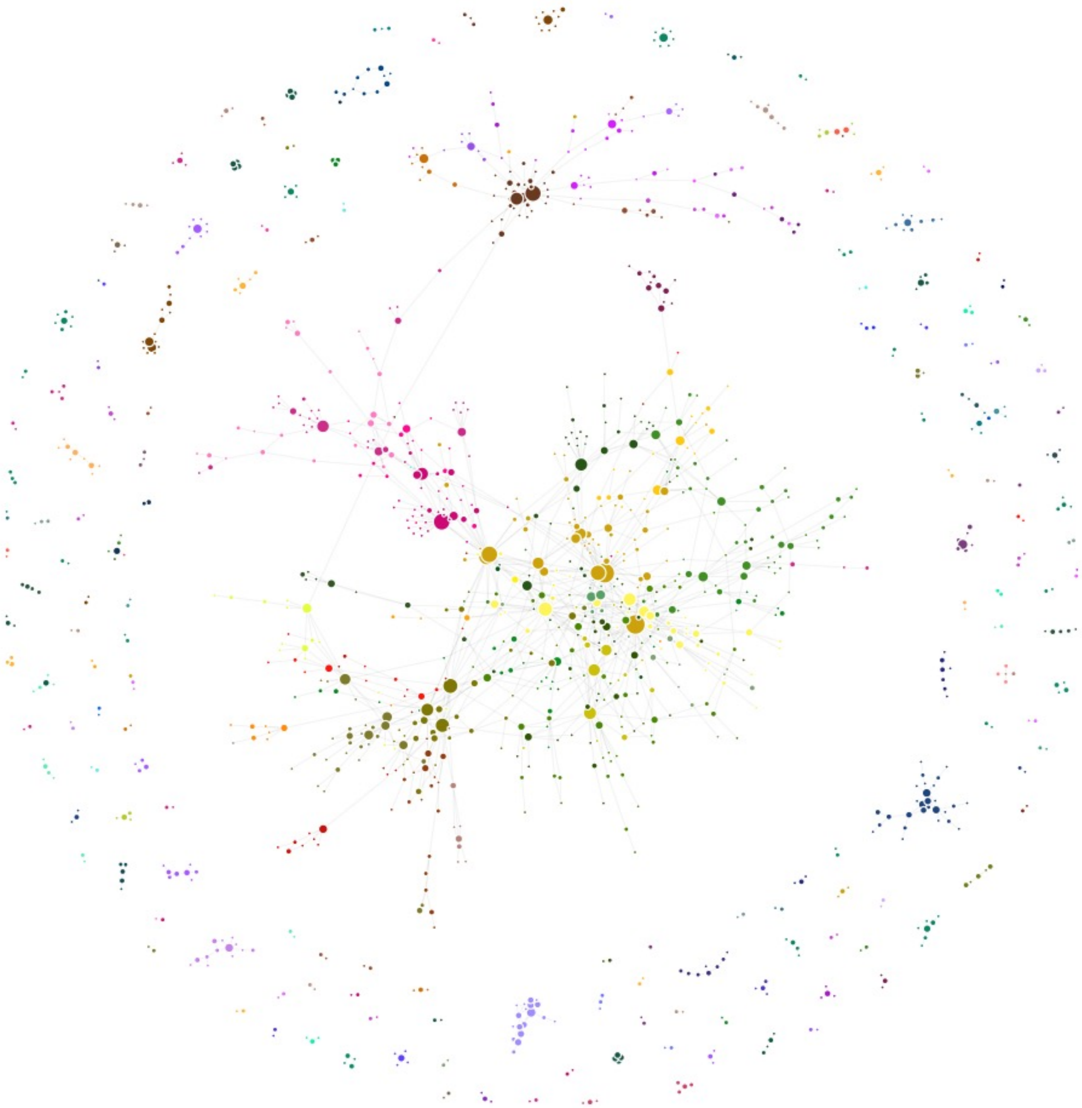
Presented by Cinda Heeren, 2016



“Rush Hour” Solution

Unknown Source

Presented by Cinda Heeren, 2016

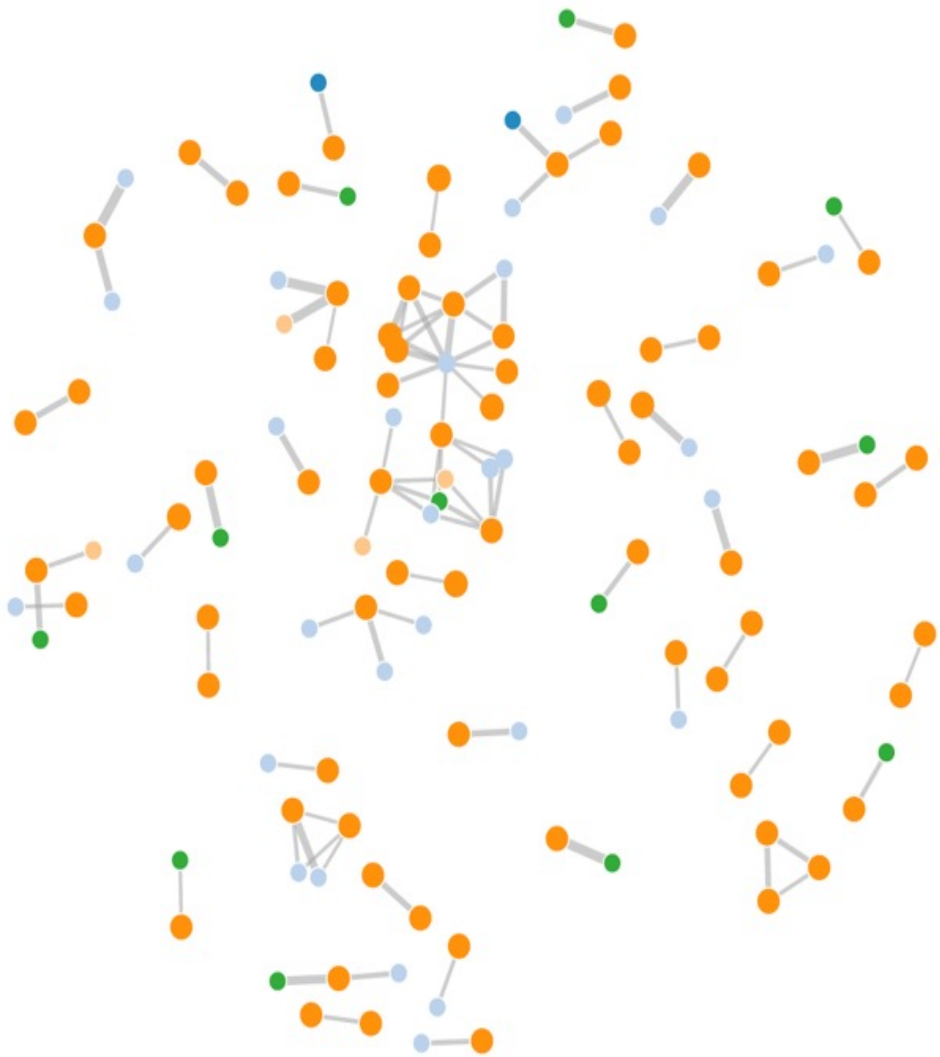


Class Hierarchy At University of Illinois Urbana-Champaign

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

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

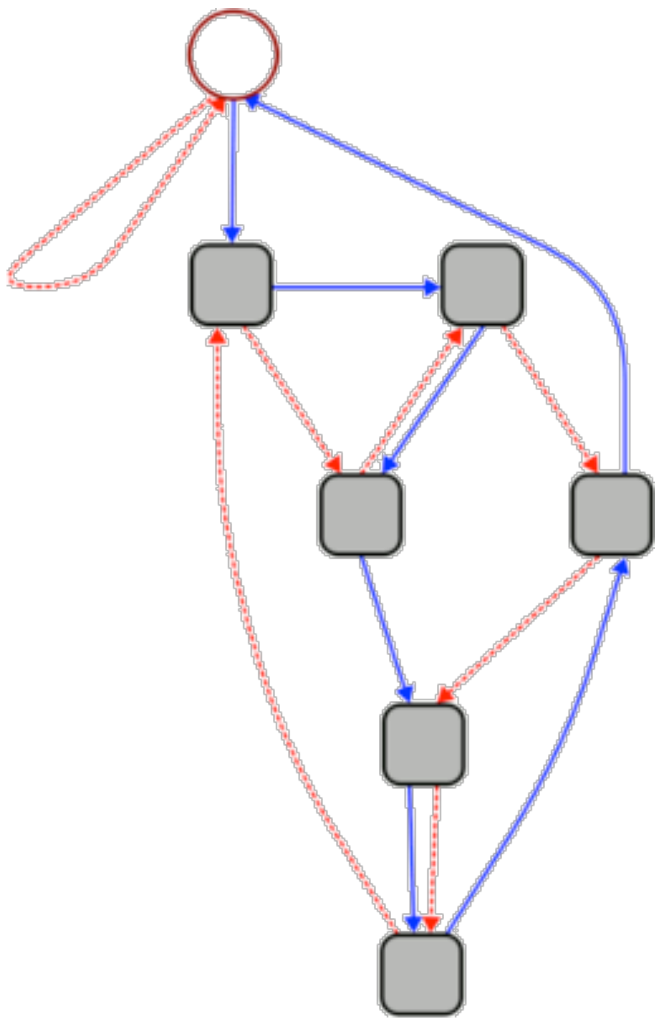
http://waf.cs.illinois.edu/discovery/class_hierarchy_at_illinois/



MP Collaborations in CS 225

Unknown Source

Presented by Cinda Heeren, 2016



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

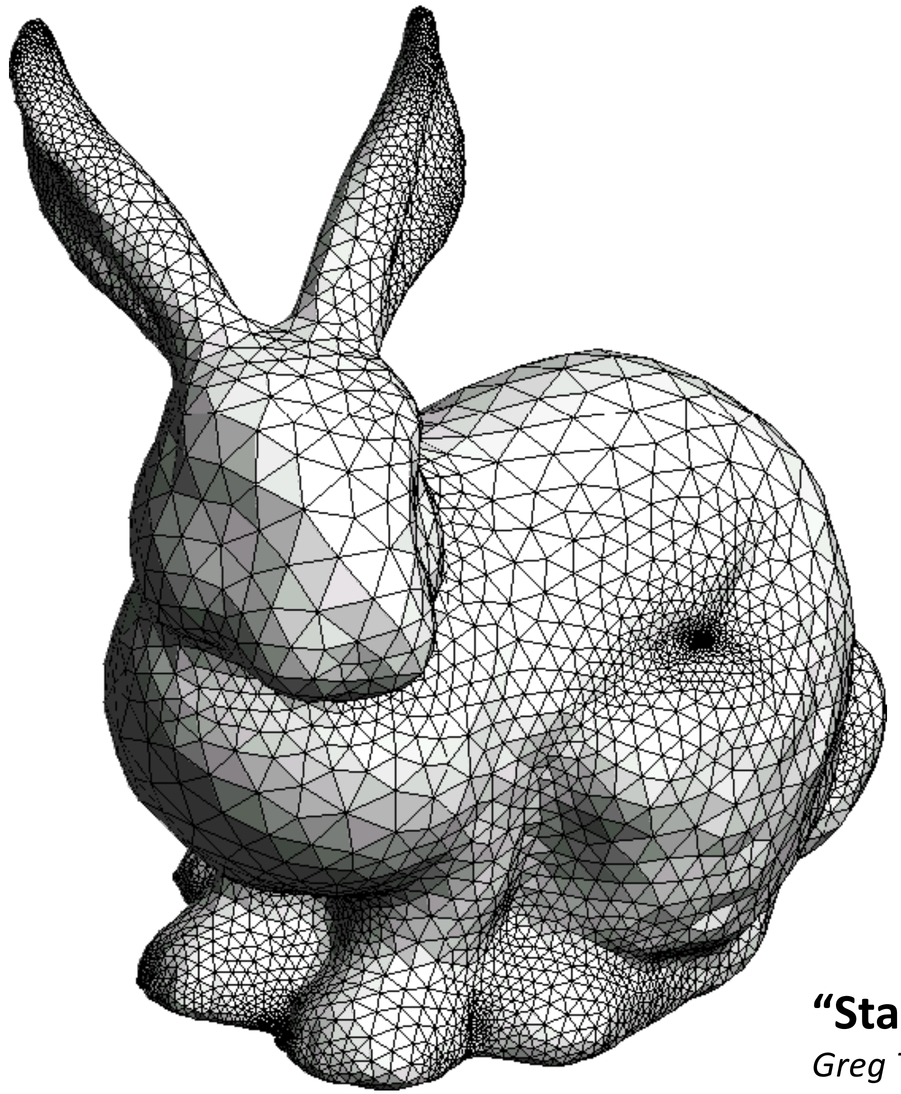
1. Start at the circle node at the top.
2. For each digit **d** in the given number, follow **d** blue (solid) edges in succession. As you move from one digit to the next, follow **1** red (dashed) edge.
3. If you end up back at the circle node, your number is divisible by 7.

3703

"Rule of 7"

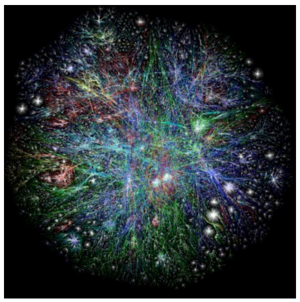
Unknown Source

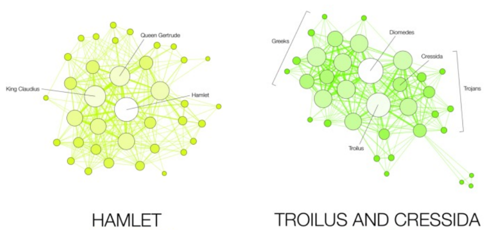
Presented by Cinda Heeren, 2016

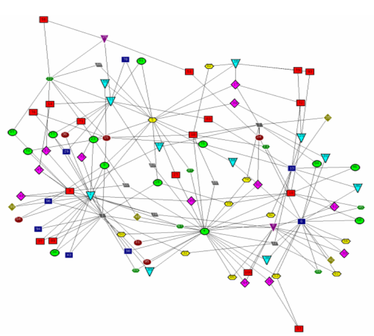


“Stanford Bunny”
Greg Turk and Mark Levoy (1994)

Graphs

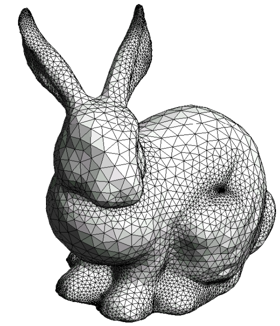


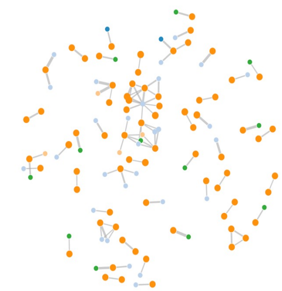


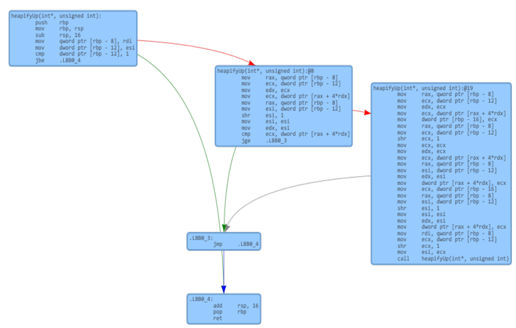


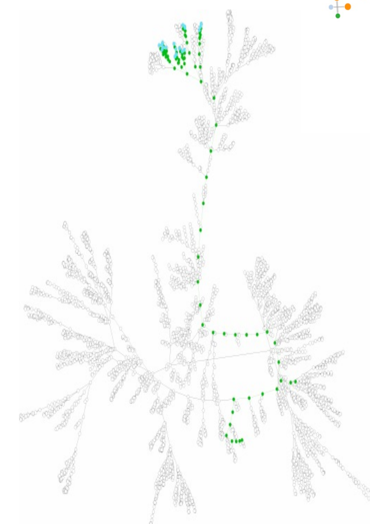
To study all of these structures:

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



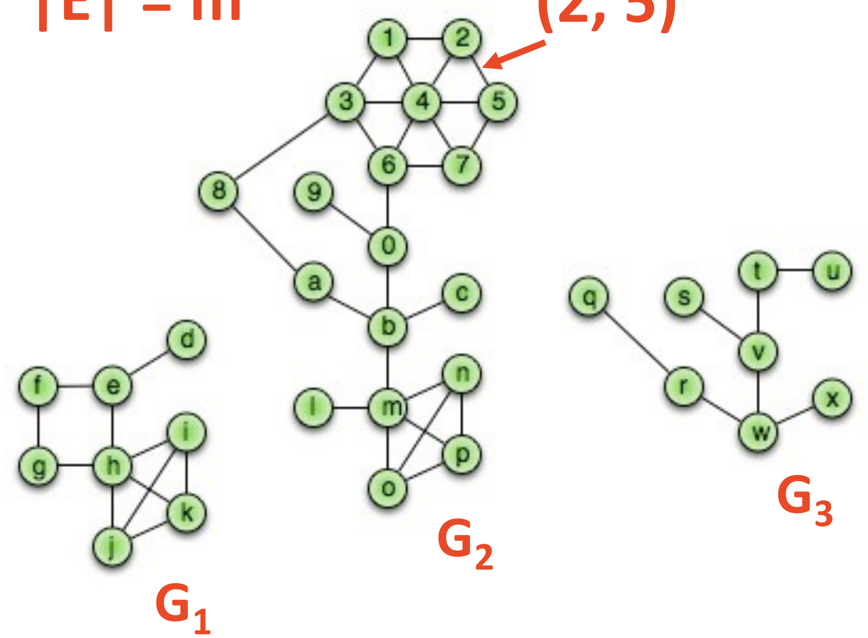






Graph Vocabulary

G = (V, E)
|V| = n
|E| = m



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

Degree(v): |I|

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

Path(G_2): Sequence of vertices connected by edges

Cycle(G_1): Path with a common begin and end vertex with at least 3 vertices.

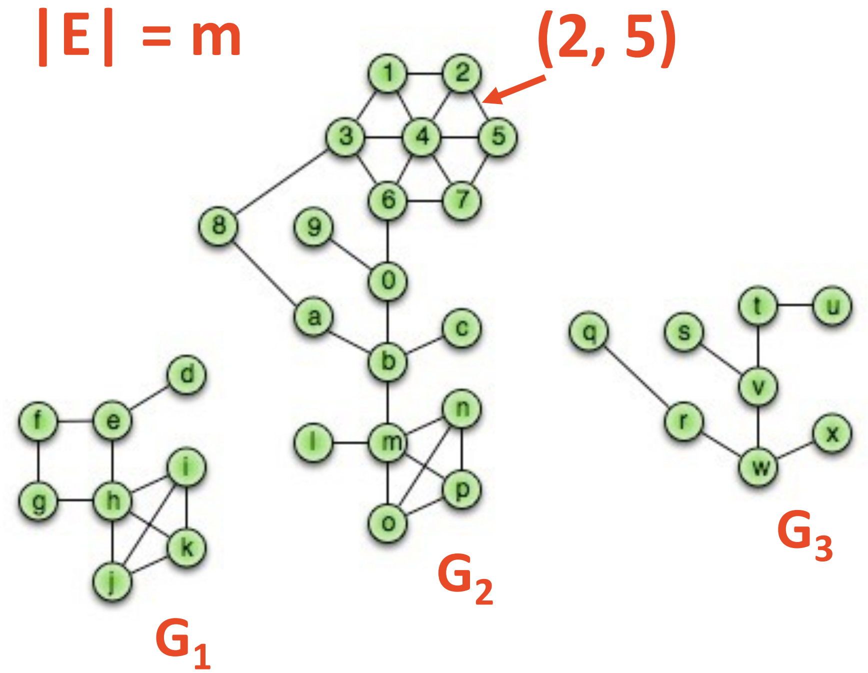
Simple Graph(G): A graph with no self loops or multi-edges.

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Subgraph(G):

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

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

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

Complete subgraph(G)

Connected subgraph(G)

Connected component(G)

Acyclic subgraph(G)

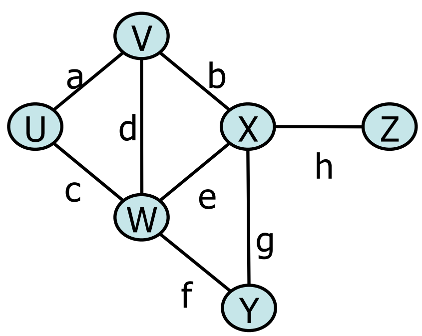
Spanning tree(G)

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:

Connected*:



Maximum edges:

Simple:

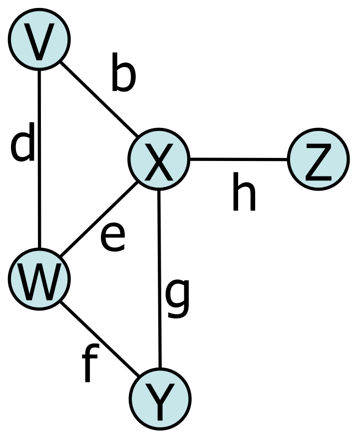
Not simple:

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

Graph ADT

Data:

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



Functions:

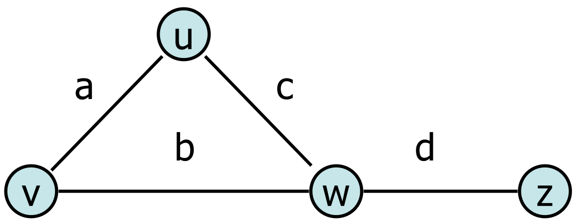
- insertVertex(K key);
- insertEdge(Vertex v1, Vertex v2, K key);

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

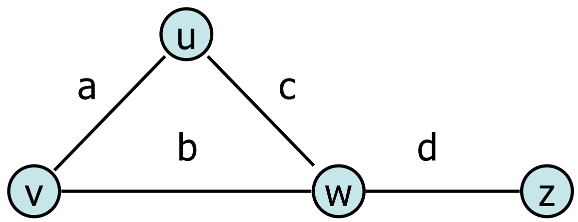
- incidentEdges(Vertex v);
- areAdjacent(Vertex v1, Vertex v2);

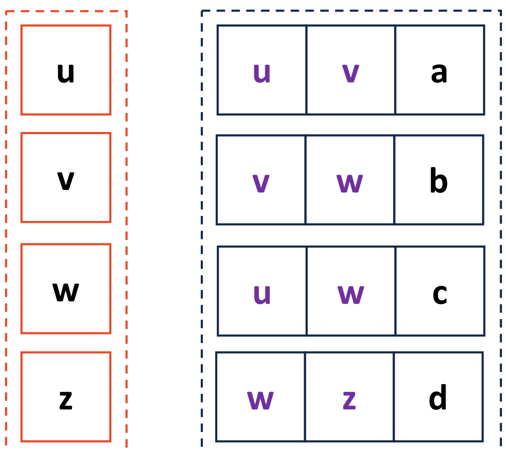
- origin(Edge e);
- destination(Edge e);

Graph Implementation Idea



Graph Implementation: Edge List

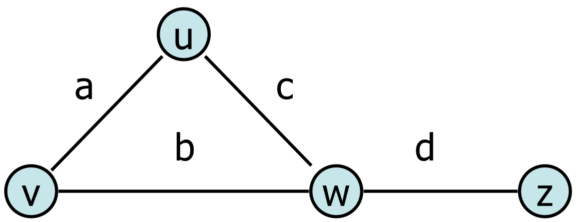


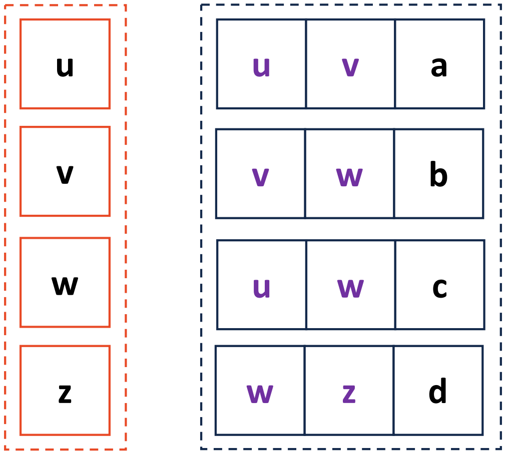


Vertex Collection:

Edge Collection:

Graph Implementation: Edge List

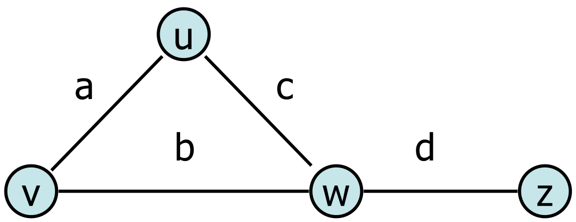




insertVertex(K key):

removeVertex(Vertex v):

Graph Implementation: Edge List



u
v
w
z

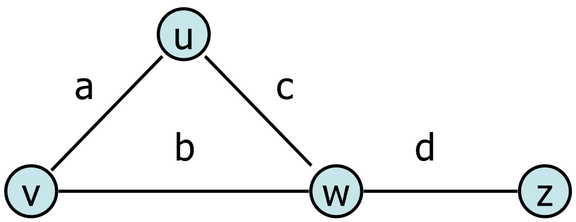
u	v	a
v	w	b
u	w	c
w	z	d

incidentEdges(Vertex v):

areAdjacent(Vertex v1, Vertex v2):

`G.incidentEdges(v1).contains(v2)`

Graph Implementation: Edge List

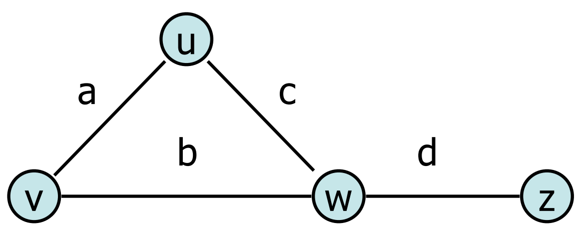


u
v
w
z

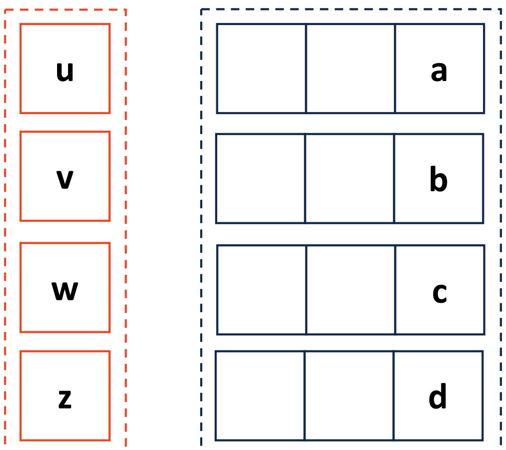
u	**v**	**a**
v	**w**	**b**
u	**w**	**c**
w	**z**	**d**

insertEdge(Vertex v1, Vertex v2, K key):

Graph Implementation: Adjacency Matrix



insertVertex(K key);
removeVertex(Vertex v);
areAdjacent(Vertex v1, Vertex v2);
incidentEdges(Vertex v);



	u	v	w	z
u				
v				
w				
z				