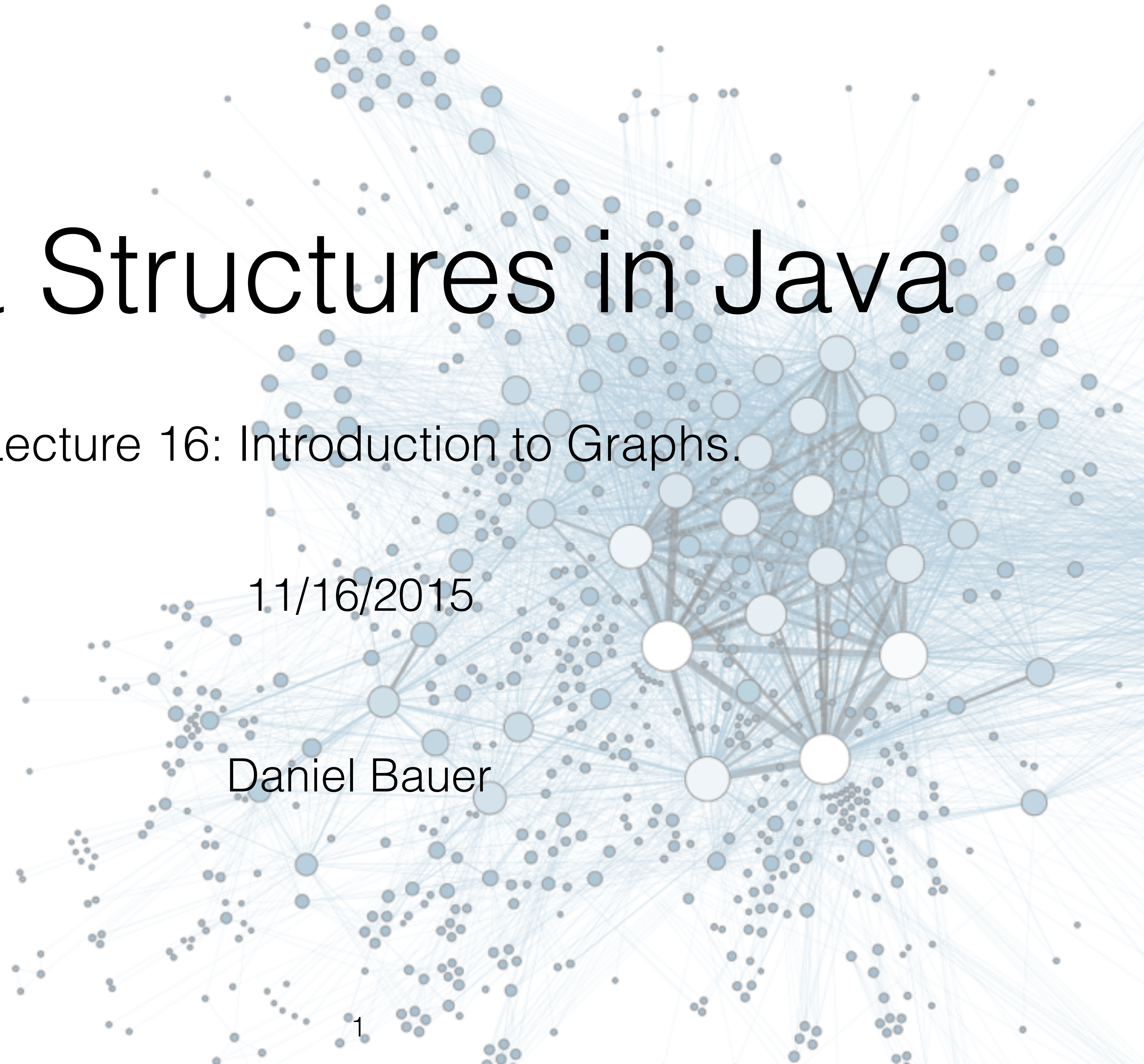
Data Structures in Java

Lecture 16: Introduction to Graphs.

11/16/2015

Daniel Bauer

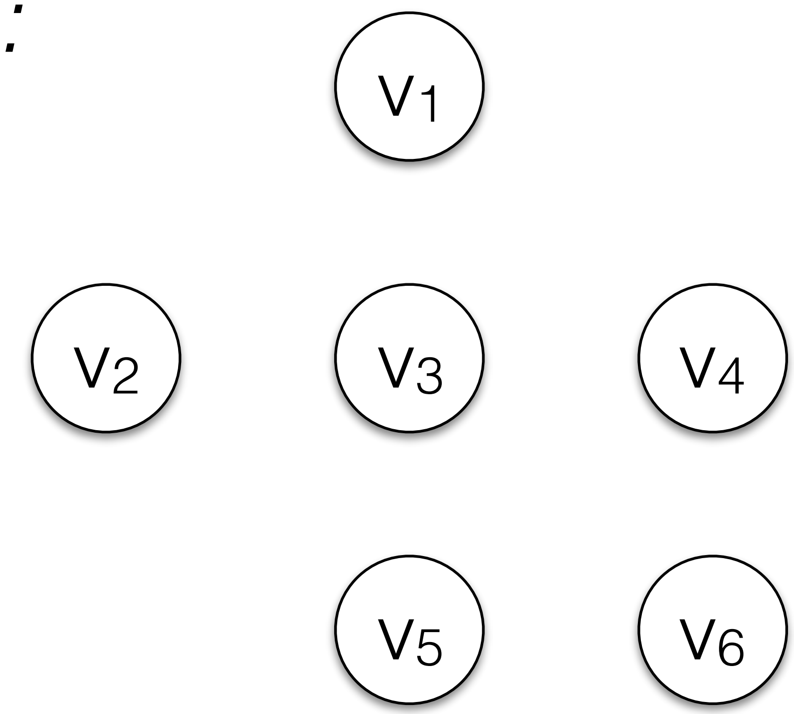


Graphs

- A **Graph** is a pair of two sets *G=(V,E):*
  - V: the set of **vertices** (or **nodes**)
  - E: the set of **edges**.
    - each edge is a pair (v,w) where $v,w \in V$

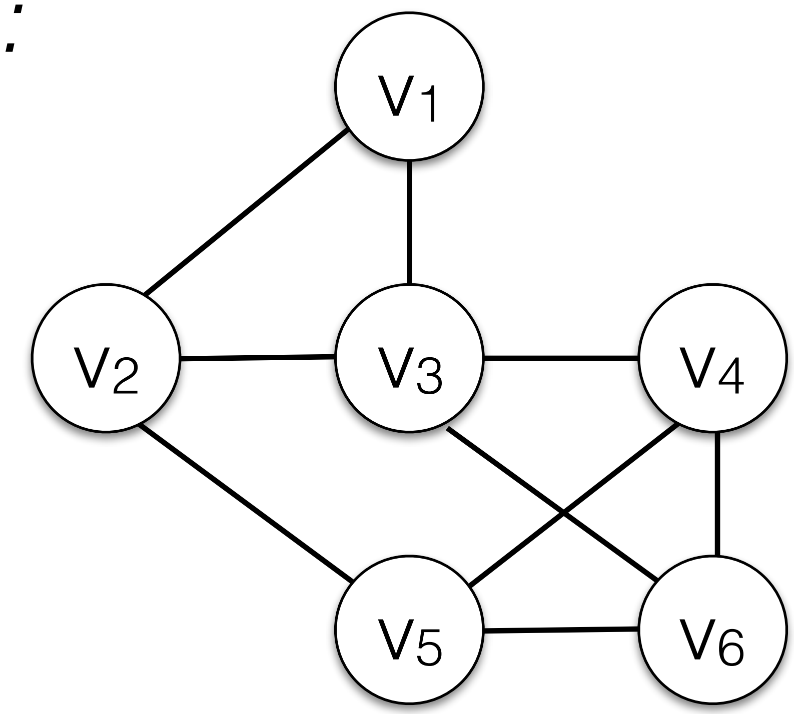
Graphs

- A **Graph** is a pair of two sets *G=(V,E):*
  - V: the set of **vertices** (or **nodes**)
  - E: the set of **edges**.
    - each edge is a pair (v,w) where v,w ∈ V



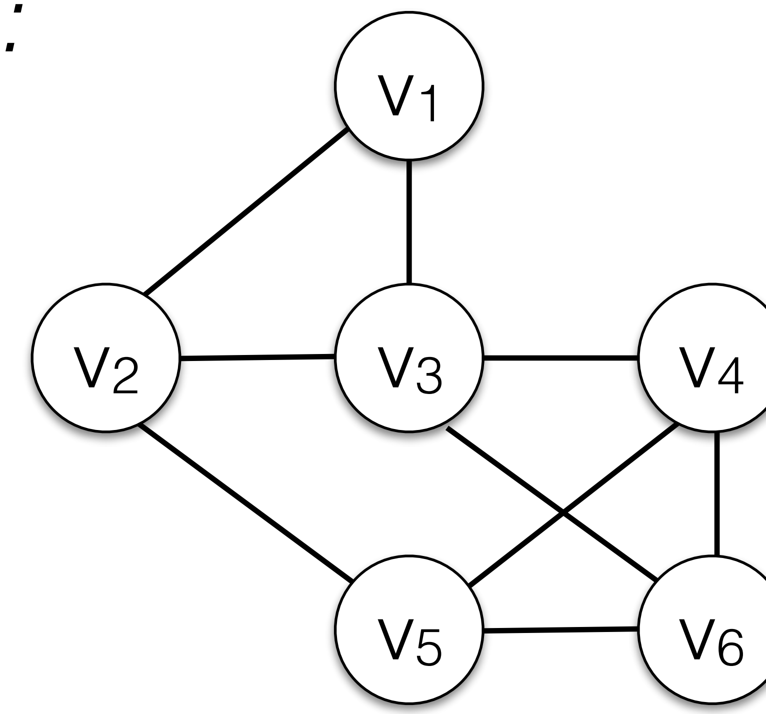
Graphs

- A **Graph** is a pair of two sets *G=(V,E):*
  - V: the set of **vertices** (or **nodes**)
  - E: the set of **edges**.
    - each edge is a pair (v,w) where v,w ∈ V



Graphs

- A **Graph** is a pair of two sets *G=(V,E)*:
  - V: the set of **vertices** (or **nodes**)
  - E: the set of **edges**.
    - each edge is a pair (v,w) where v,w $\in$ V



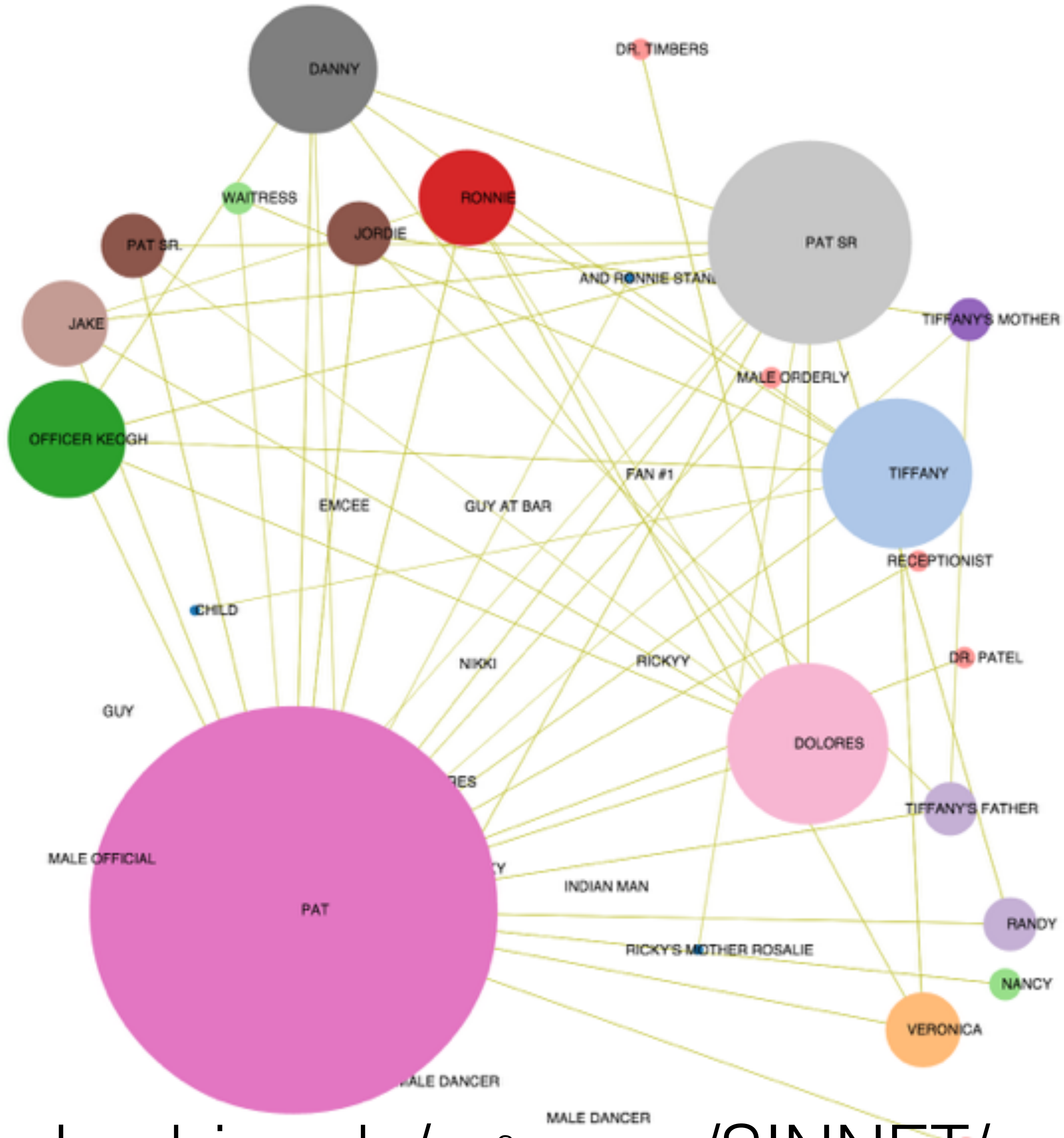
$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$
$$E = \{(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_2, v_5), (v_3, v_4), (v_3, v_6), (v_4, v_5), (v_4, v_6), (v_5, v_6)\}$$

Graphs in Computer Science

- Graphs are used to model all kinds of relational data.
- General purpose algorithms make it possible to solve problems on these models.
  - Shortest Paths, Spanning Tree, Finding Cliques, Strongly Connected Components, Network Flow, Graph Coloring, Minimum Edge/Vertex Cover, Graph Partitioning, …

Social Networks

Interaction Networks Extracted from Text

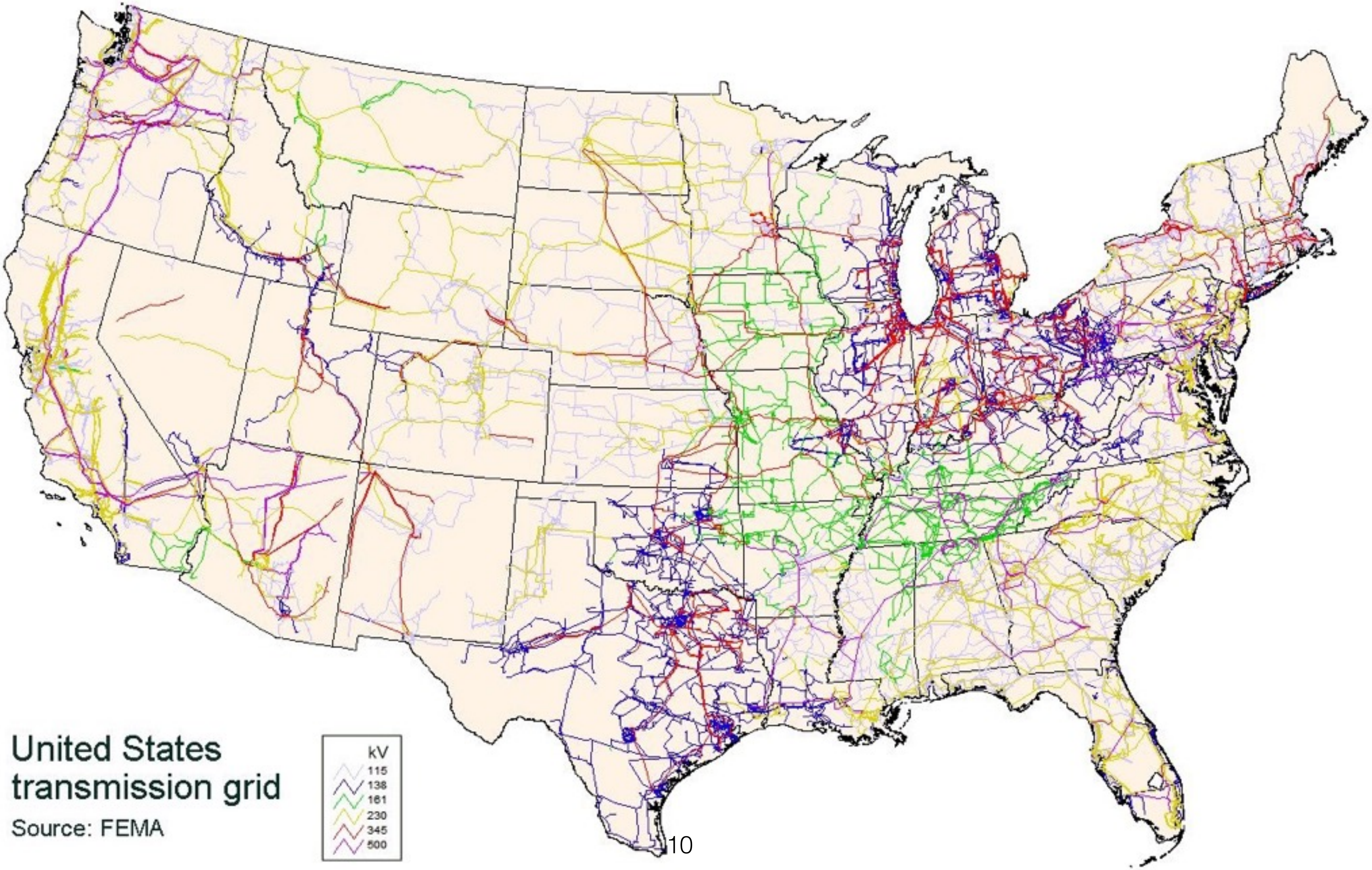


http://www.cs.columbia.edu/~apoorv/SINNET/

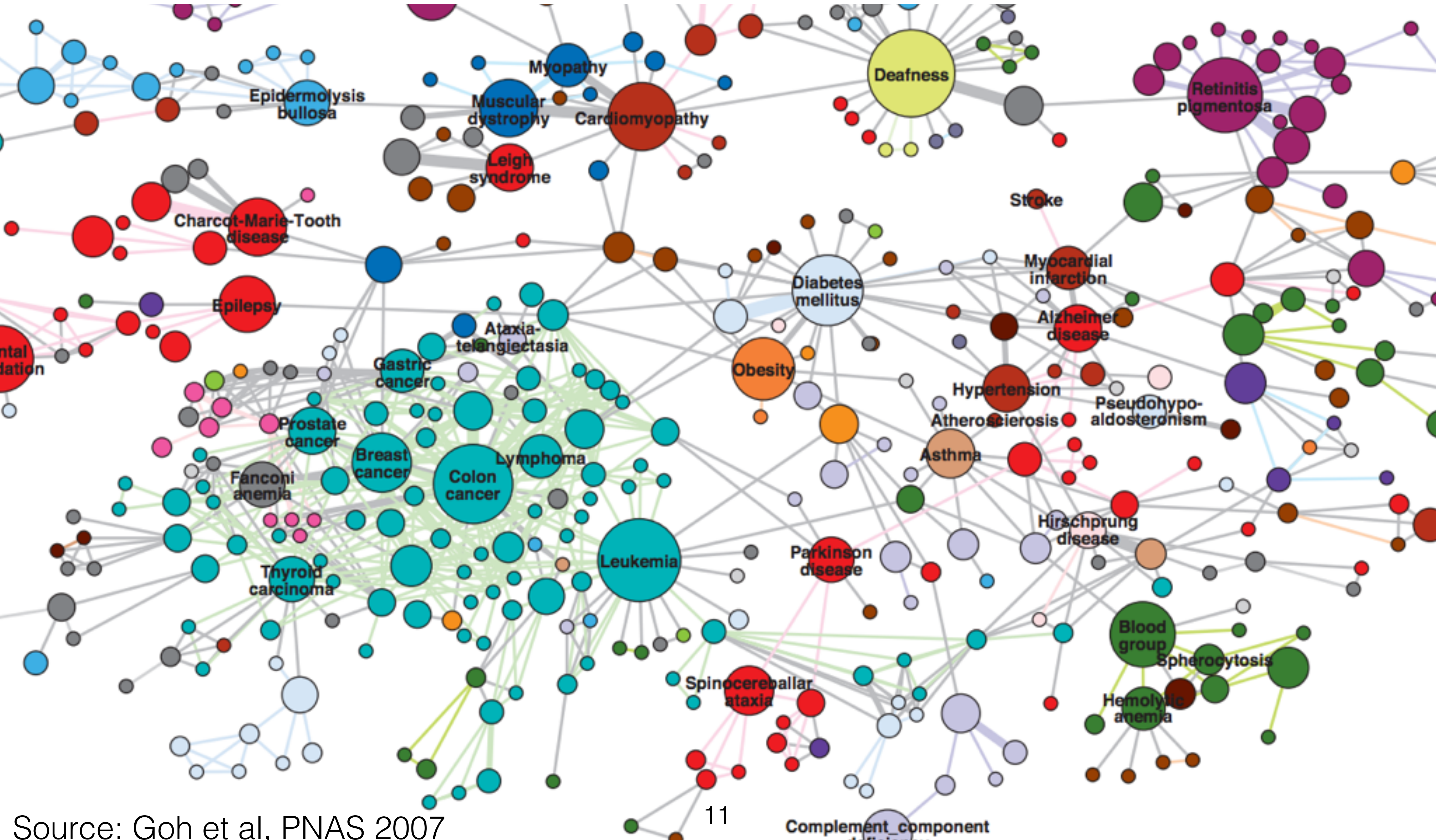
Rail Network

Source: Days of WonderVideo Games

US Power Grid



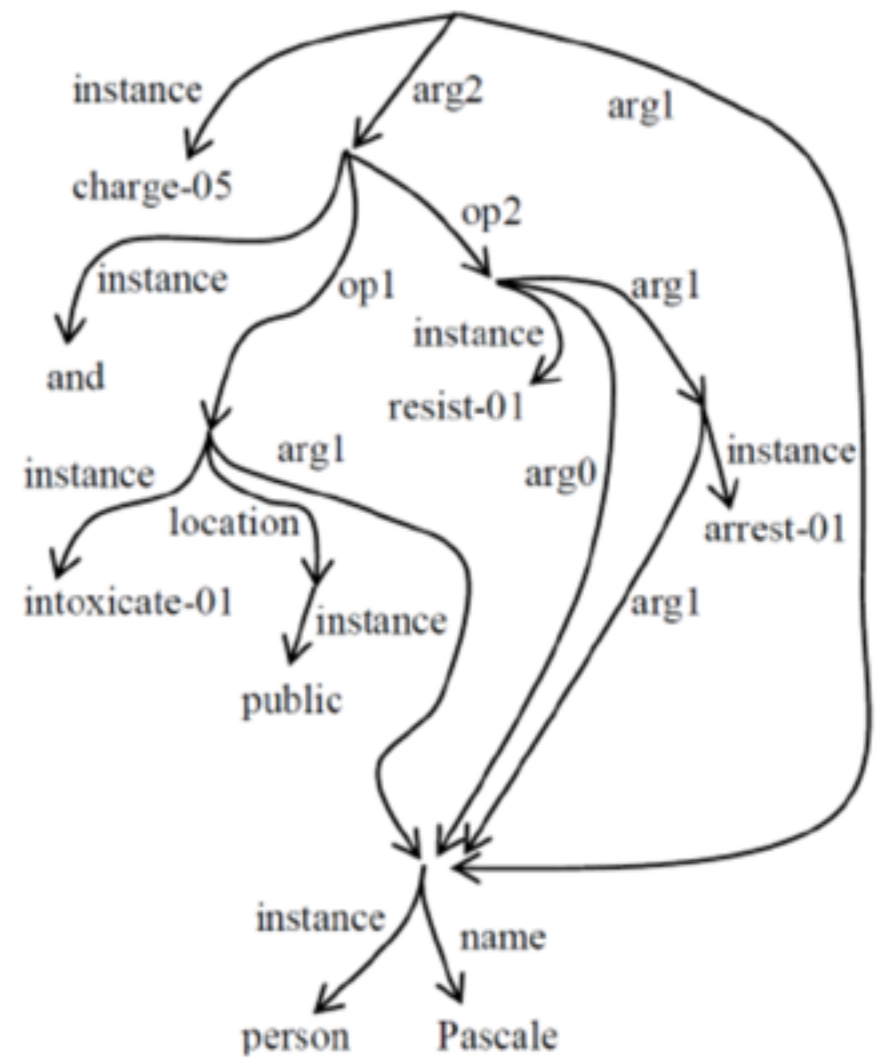
Human Disease Network



Source: Goh et al, PNAS 2007

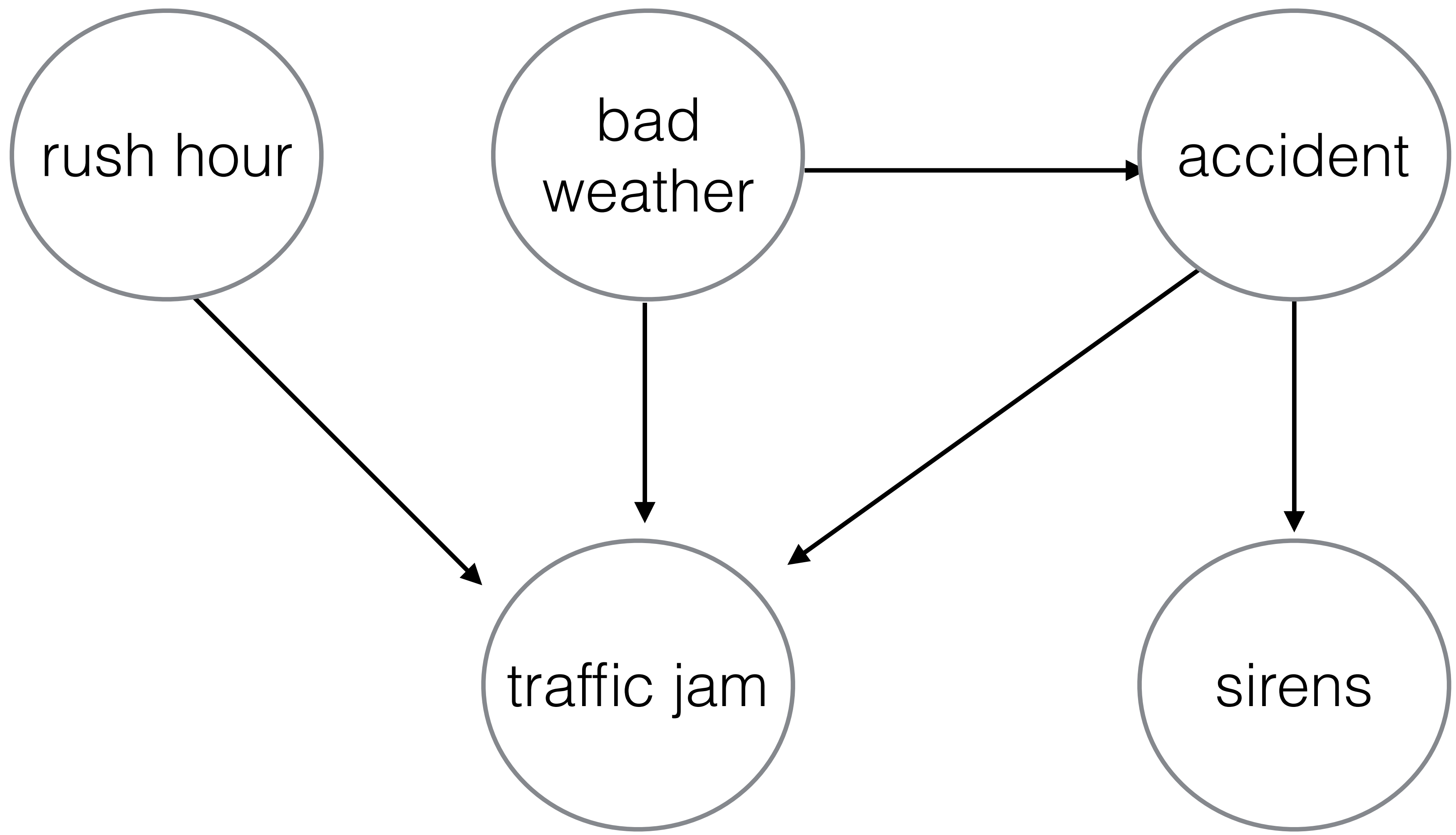
Graph-Based Representation of Sentence Meaning

"Pascale was charged with public intoxication and resisting arrest."



Source: Kevin Knight

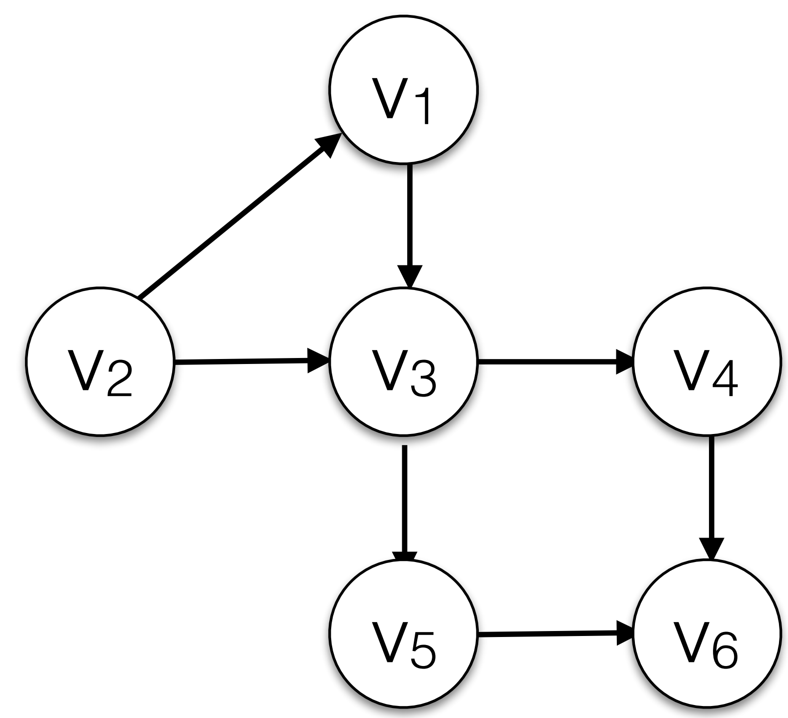
Graphical Models



Edges

- Graphs may be **directed** or **undirected**.
  - In directed graphs, the edge pairs are ordered.

- Edges often have some weight or cost associated with them (**weighted** graphs).



$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$

$E = \{(v_1, v_3), (v_2, v_1), (v_2, v_3), (v_3, v_4), (v_3, v_5), (v_4, v_6), (v_5, v_6)\}$

directed graph

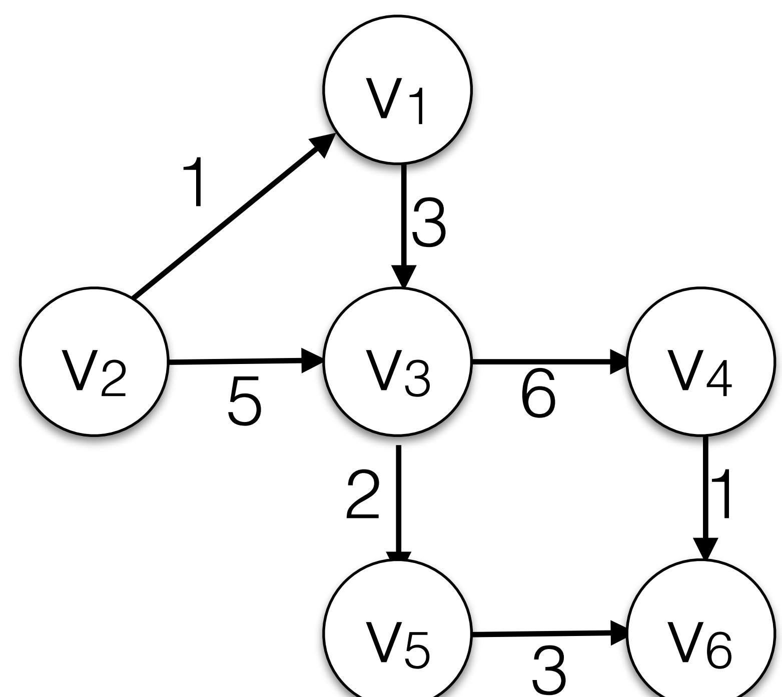
Edges

- Graphs may be **directed** or **undirected**.
  - In directed graphs, the edge pairs are ordered.

- Edges often have some weight or cost associated with them (**weighted** graphs).

$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$

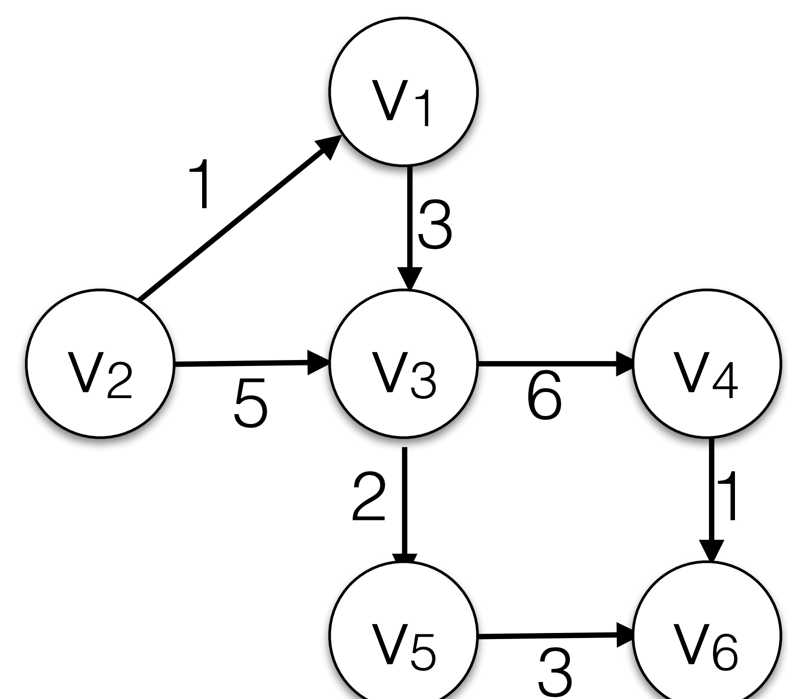
$E = \{(v_1, v_3), (v_2, v_1), (v_2, v_3), (v_3, v_4), (v_3, v_5), (v_4, v_6), (v_5, v_6)\}$



directed and weighted graph

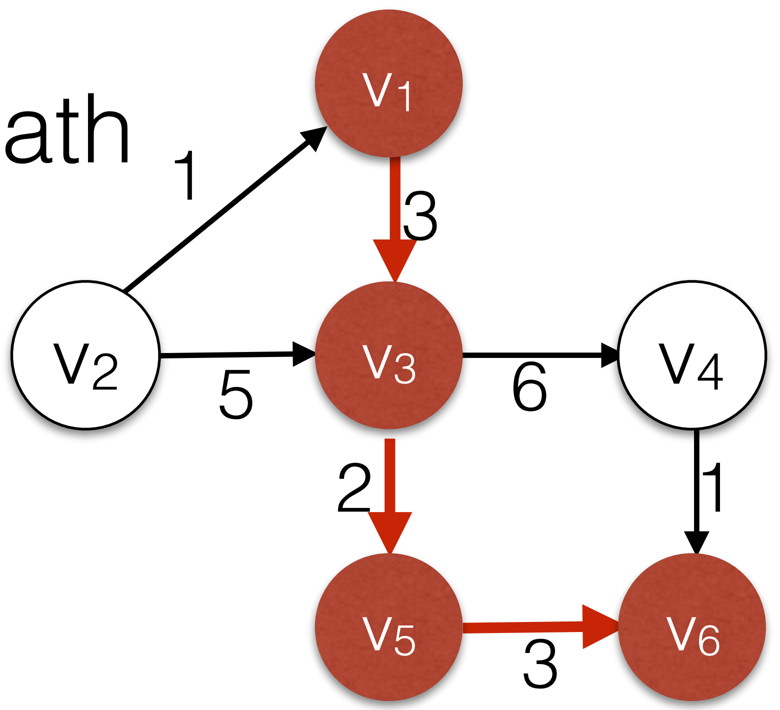
Paths

- Vertex w is **adjacent** to vertex v iff $(w,v) \in E$.
- A **path** is a sequence of vertices $w_1, w_2, \ldots, w_k$ such that $(w_i, w_{i+1}) \in E$.



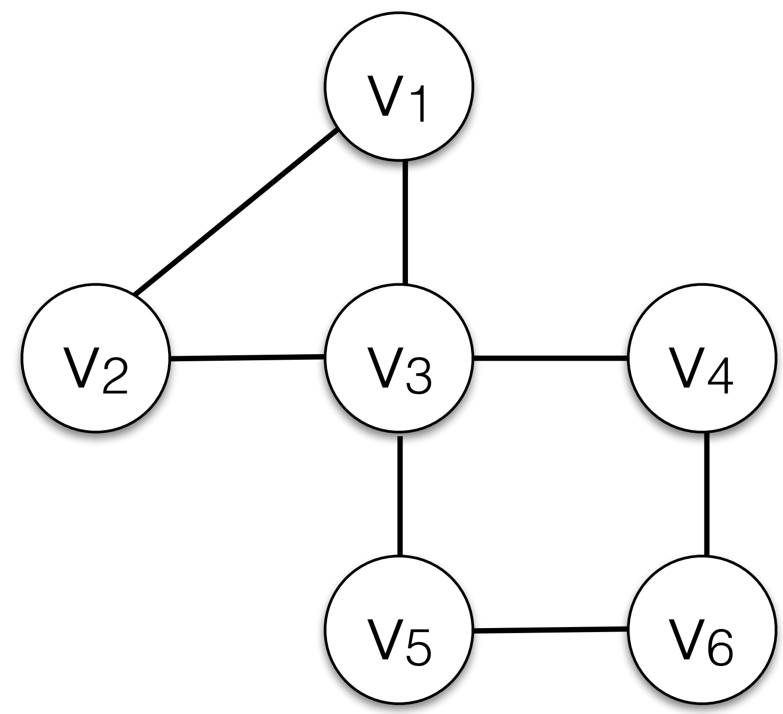
Paths

- Vertex w is **adjacent** to vertex v iff $(w,v) \in E$.
- A **path** is a sequence of vertices $w_1, w_2, \ldots, w_k$ such that $(w_i, w_{i+1}) \in E$.
- **length** of a path:
  k-1 = number of edges on path
- **cost** of a path:
  Sum of all edge costs.



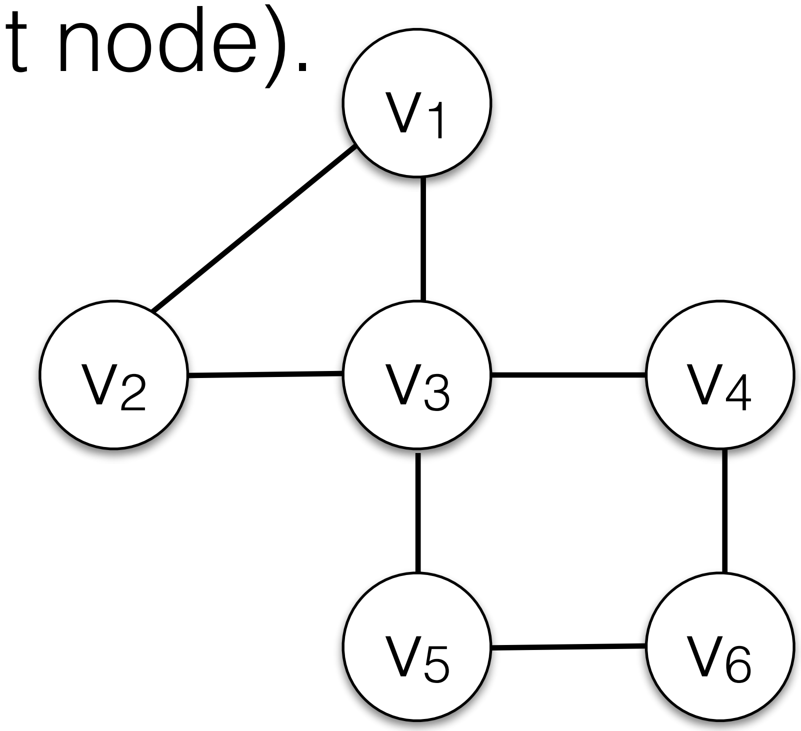
Path from v_1 to v_6, length 3, cost 8
$(v_1, v_3), (v_3, v_5), (v_5, v_6)$

Simple Paths



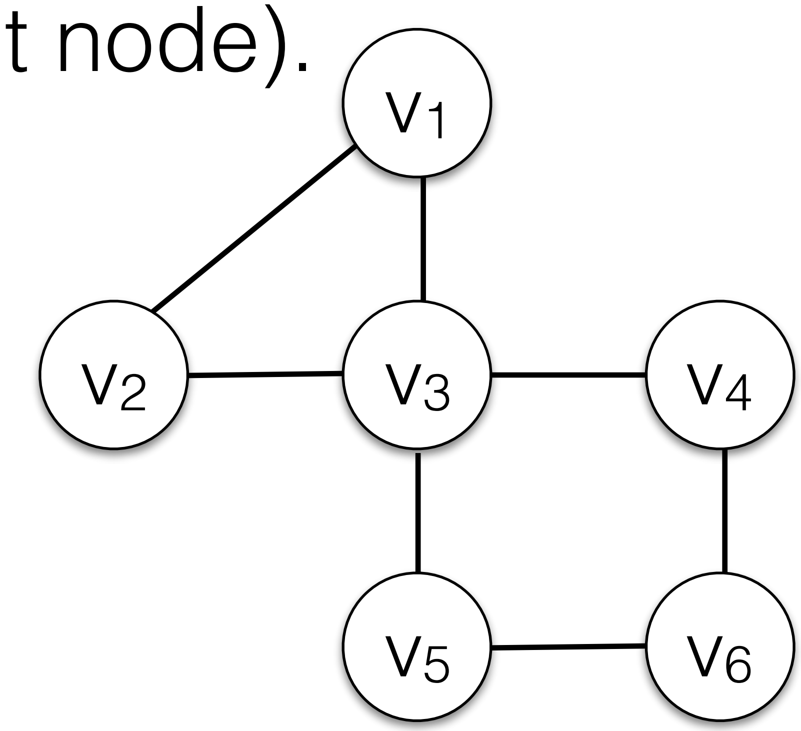
Simple Paths

- A **simple path** is a path that contains every node only once (except possibly the first and last node).



Simple Paths

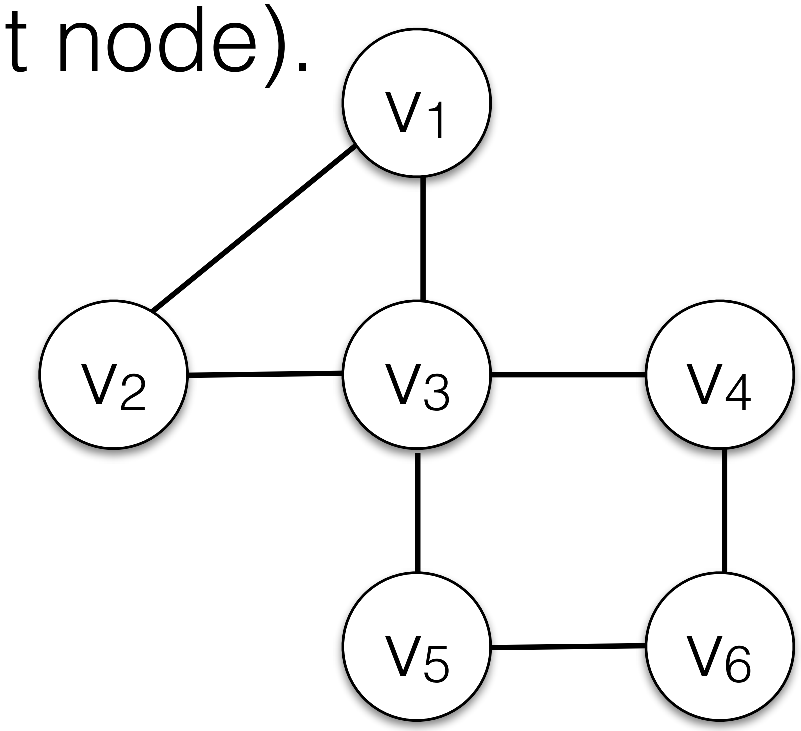
- A **simple path** is a path that contains every node only once (except possibly the first and last node).



- (v_2, v_3, v_4, v_6, v_5,v_3, v_1) is a path but not a simple path.

Simple Paths

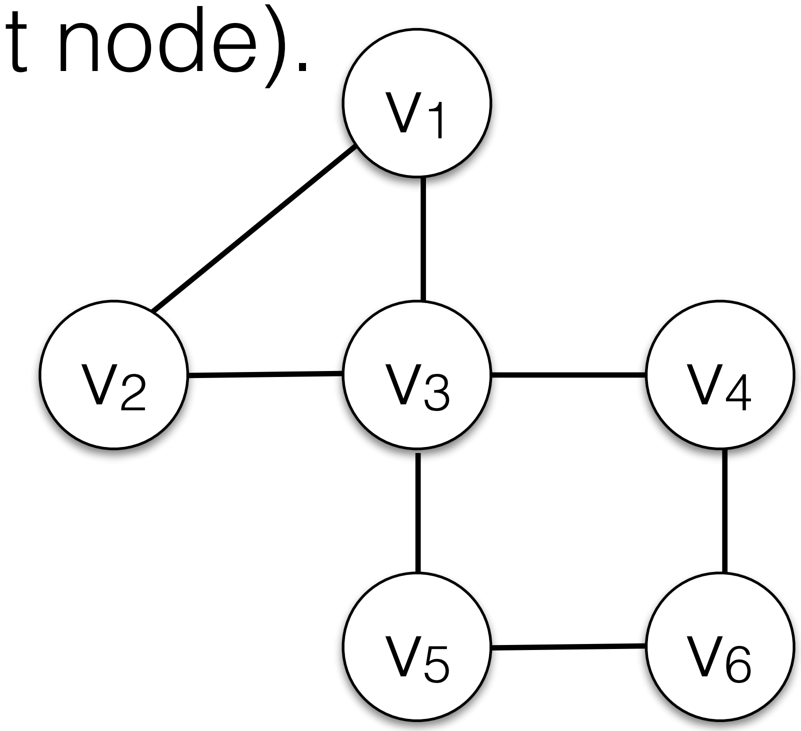
- A **simple path** is a path that contains every node only once (except possibly the first and last node).



- (v_2, v_3, v_4, v_6, v_5,v_3, v_1) is a path but not a simple path.
- There are only two simple paths between v_2 and v_1: (v_2, v_1) and (v_2, v_3, v_1)

Simple Paths

- A **simple path** is a path that contains every node only once (except possibly the first and last node).



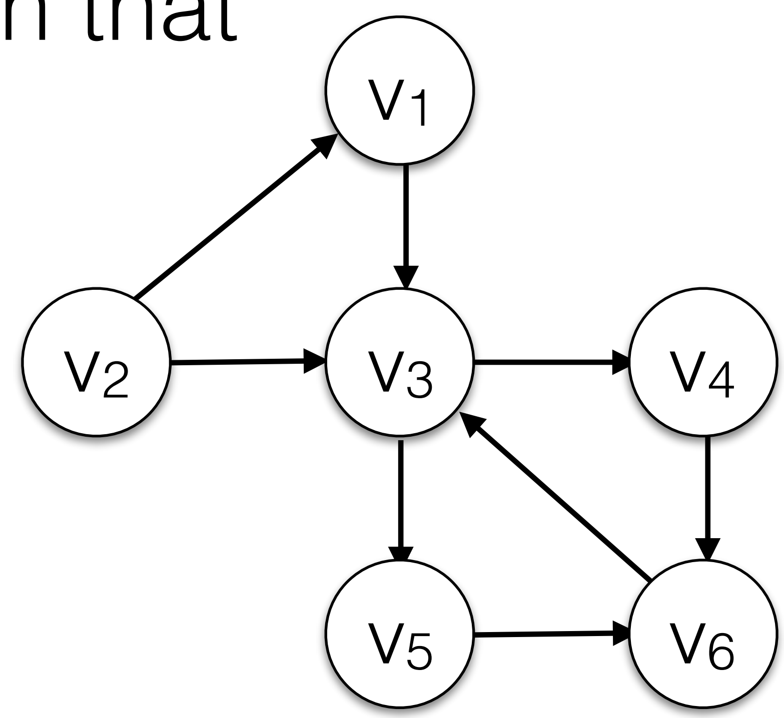
- (v_2, v_3, v_4, v_6, v_5,v_3, v_1) is a path but not a simple path.

- There are only two simple paths between v_2 and v_1: (v_2, v_1) and (v_2, v_3, v_1)

- (v_1, v_3, v_2, v_1) is a simple path.

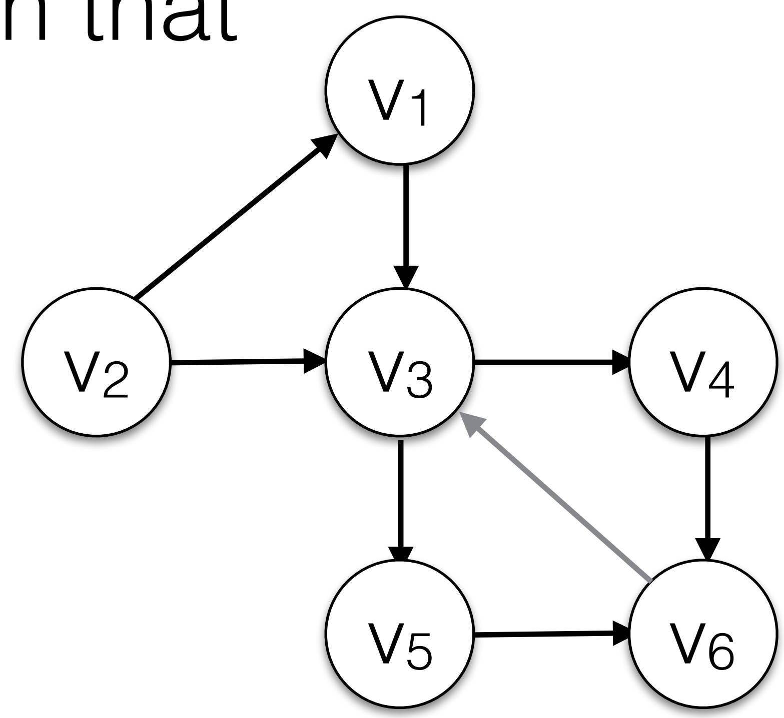
Cycles in Directed Graphs

- A **cycle** is a path (of length > 1) such that $w_1 = w_k$
- (v_3, v_4, v_6, v_3) is a cycle.



Cycles in Directed Graphs

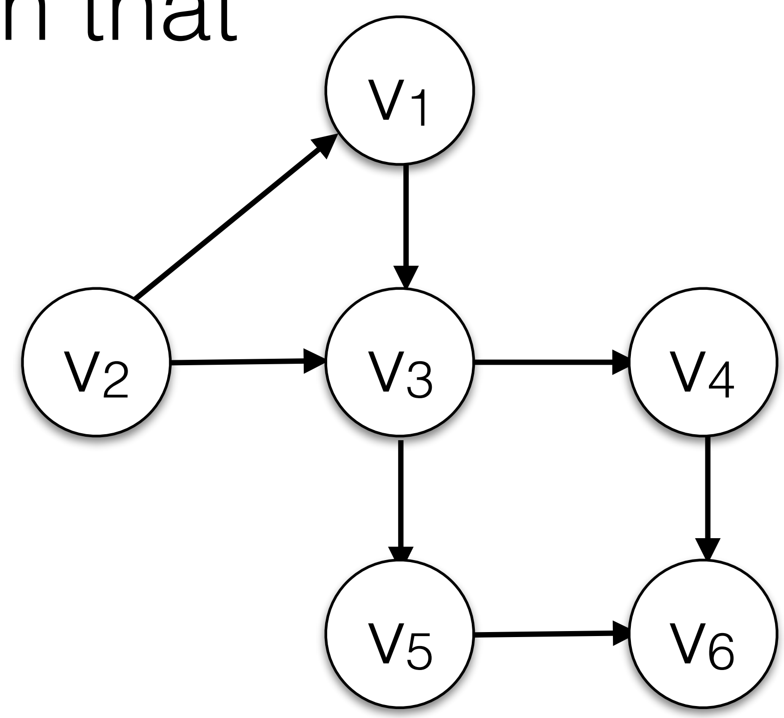
- A **cycle** is a path (of length > 1) such that $w_1 = w_k$
- (v_3, v_4, v_6, v_3) is a cycle.



- A **Directed Acyclic Graph (DAG)** is a directed graph that contains no cycles.

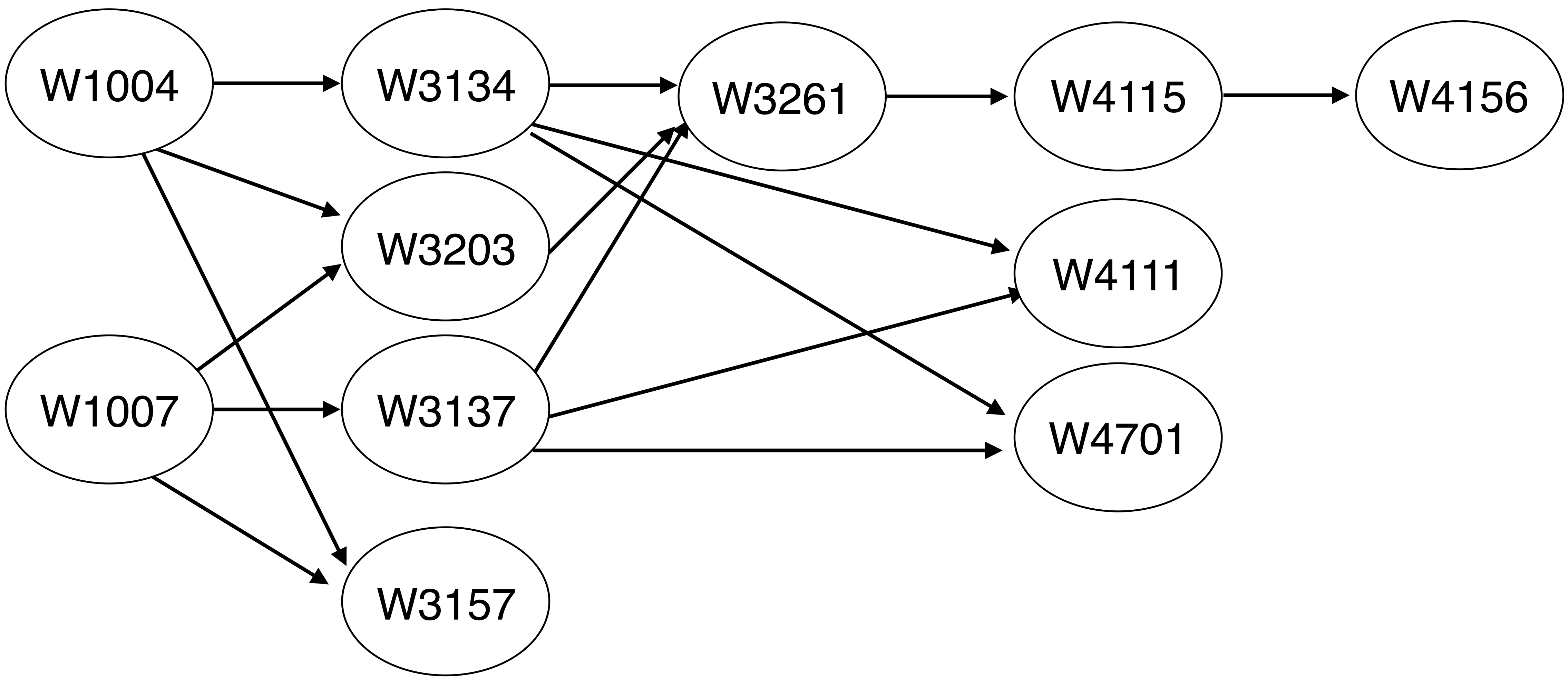
Cycles in Directed Graphs

- A **cycle** is a path (of length > 1) such that $w_1 = w_k$
- (v_3, v_4, v_6, v_3) is a cycle.



- A **Directed Acyclic Graph (DAG)** is a directed graph that contains no cycles.

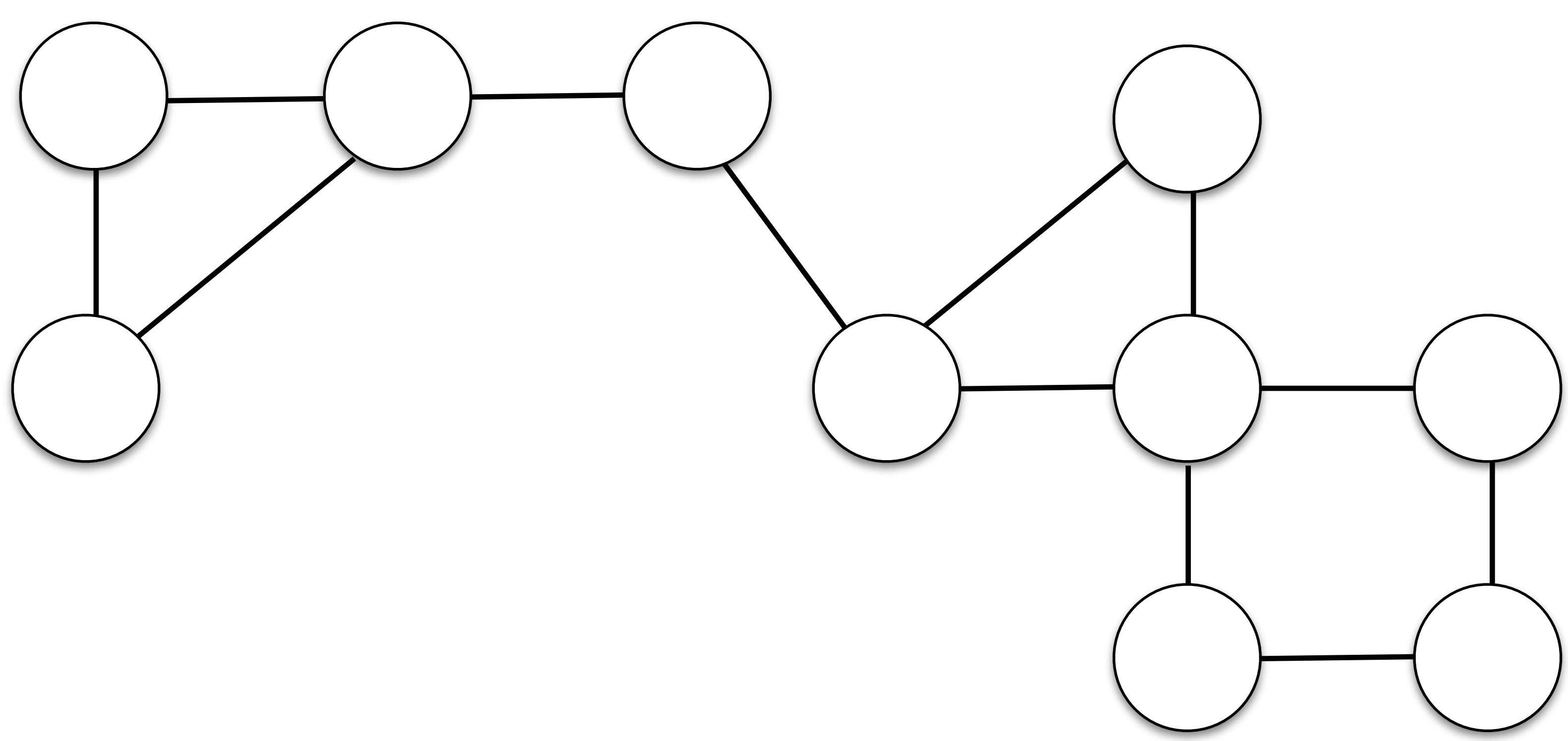
Columbia CS Course Prerequisites as a DAG



Please do not use this figure for program planning! No guarantee for accuracy.

Connectivity

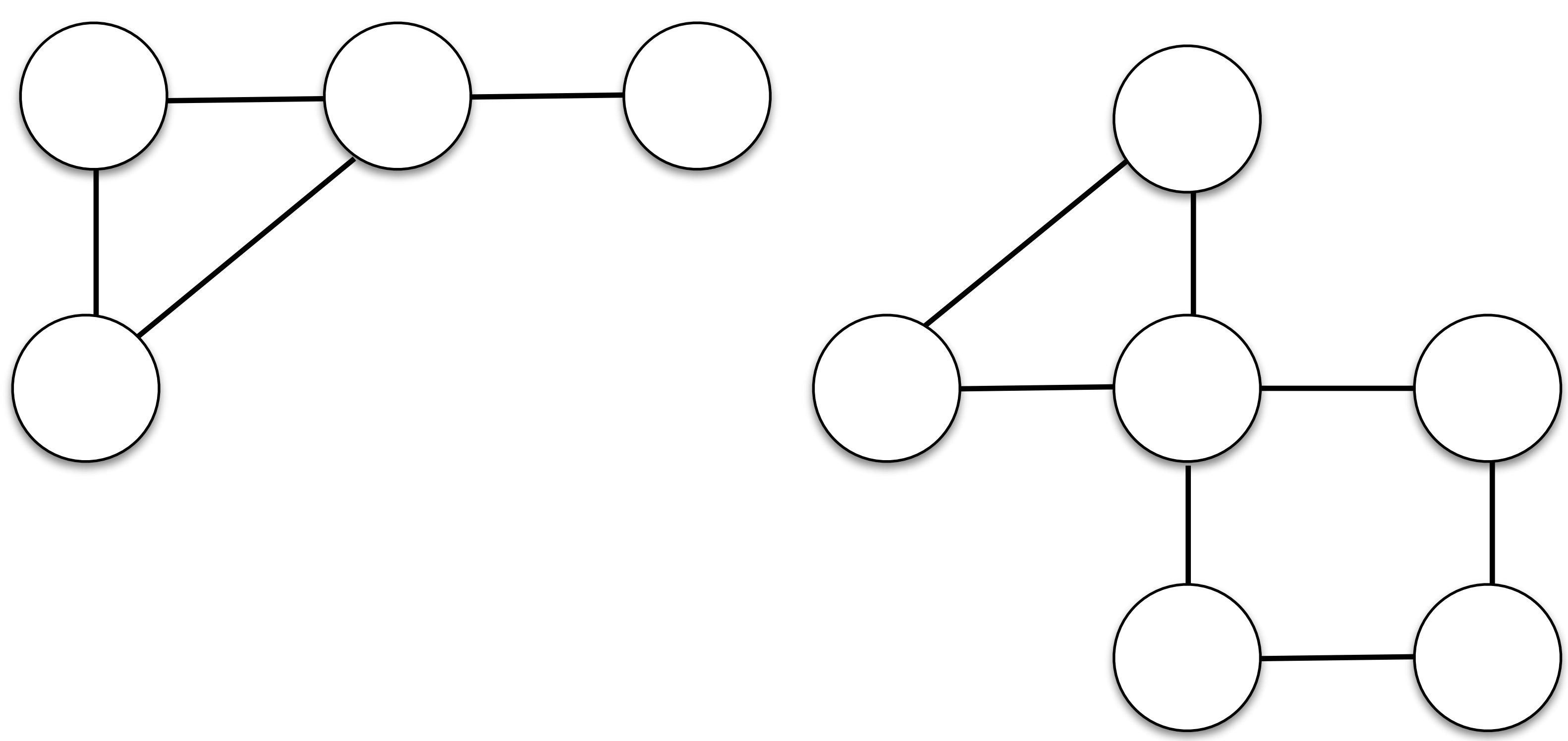
- An undirected graph is **connected** if there is a path from every vertex to every other vertex.



connected graph

Connectivity

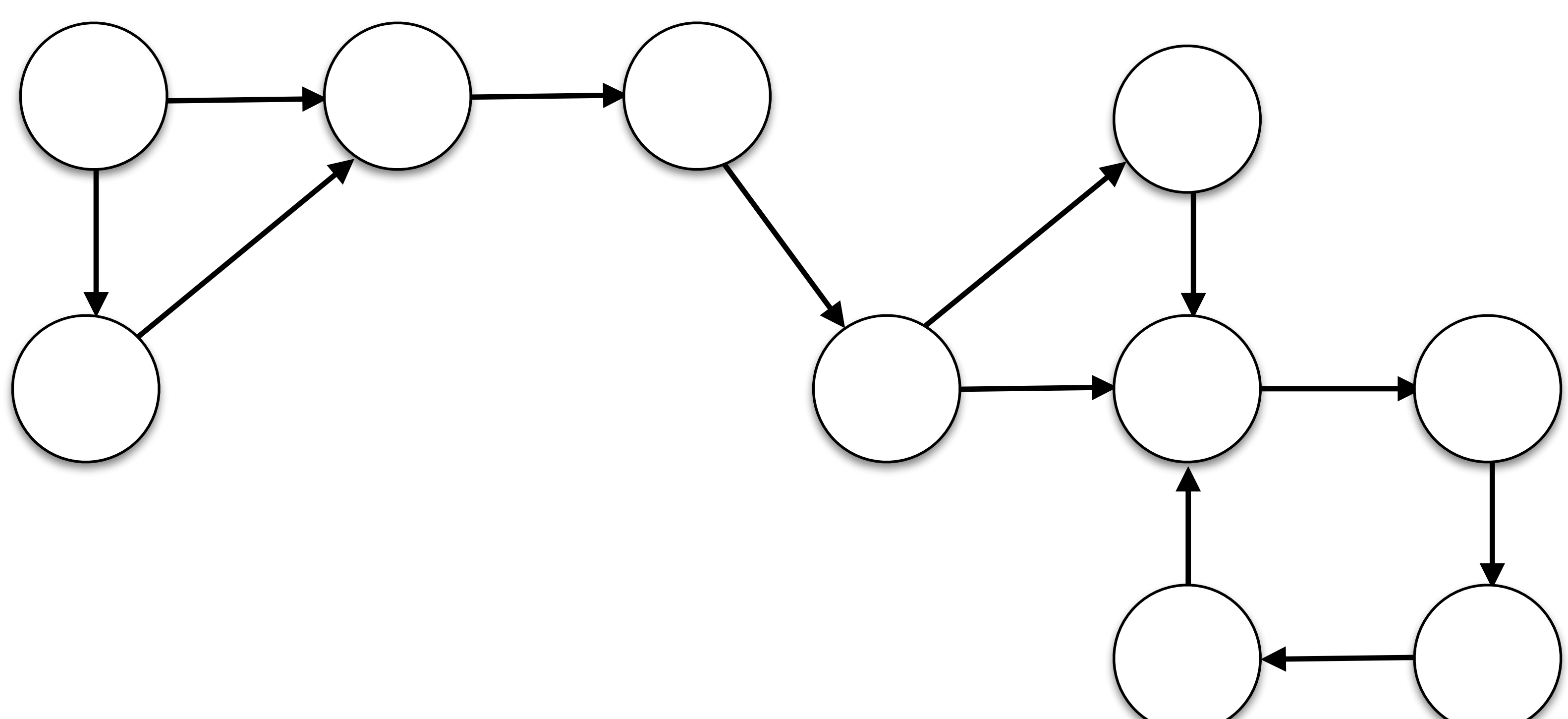
- An undirected graph is **connected** if there is a path from every vertex to every other vertex.



unconnected graph

Connectivity in Directed Graphs

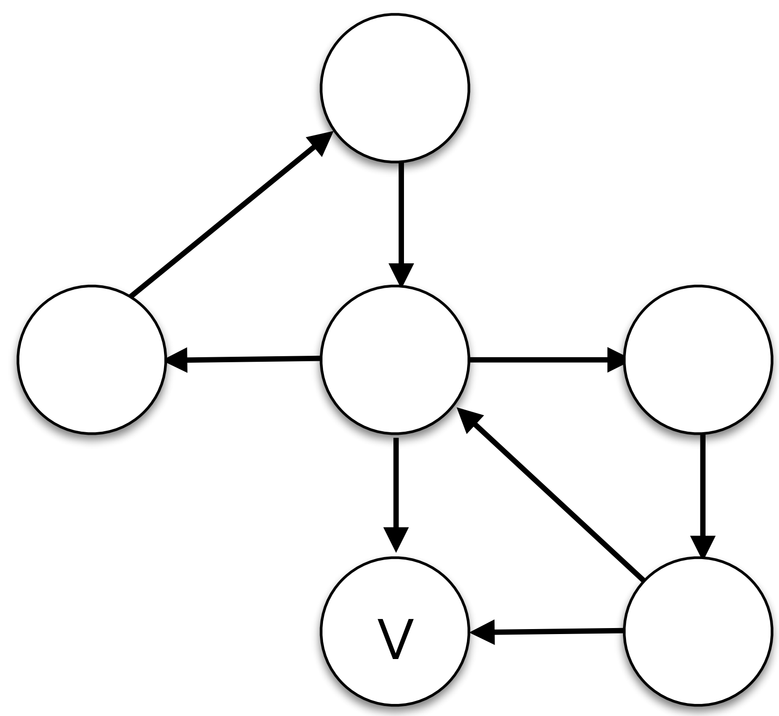
- A directed graph is **weakly connected** if there is an *undirected* path from every vertex to every other vertex.



weakly connected graph

Strongly Connected Graphs

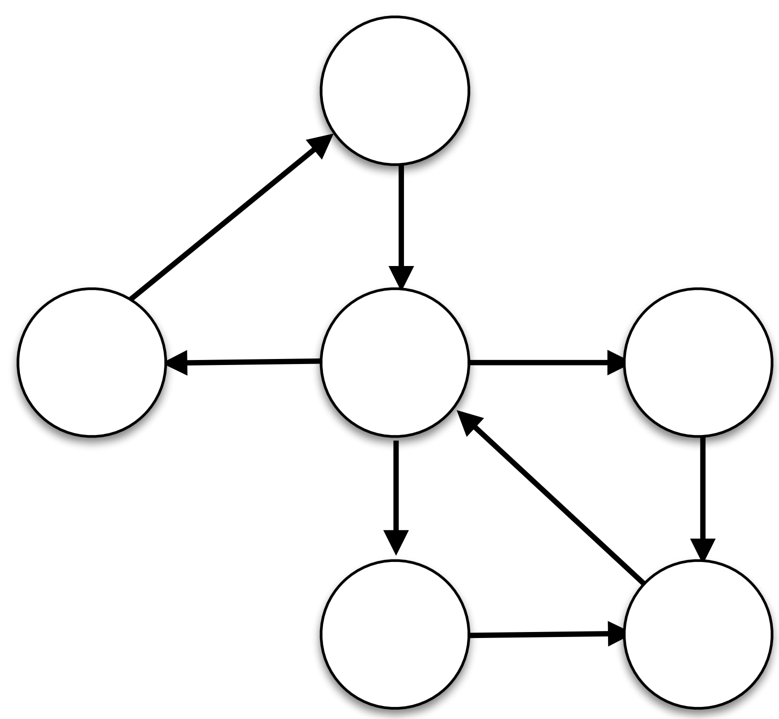
- A directed graph is **strongly connected** if there is a path from every vertex to every other vertex.



Weakly connected, but not strongly connected (no other vertex can be reached from v).

Strongly Connected Graphs

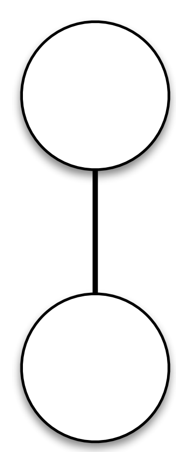
- A directed graph is **strongly connected** if there is a path from every vertex to every other vertex.



strongly connected

Complete Graphs

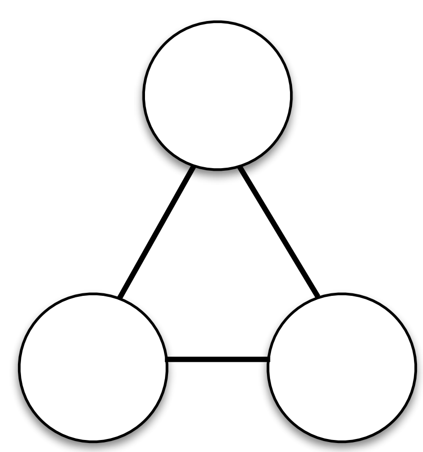
- A **complete graph** has edges between every pair of vertices.



N=2

Complete Graphs

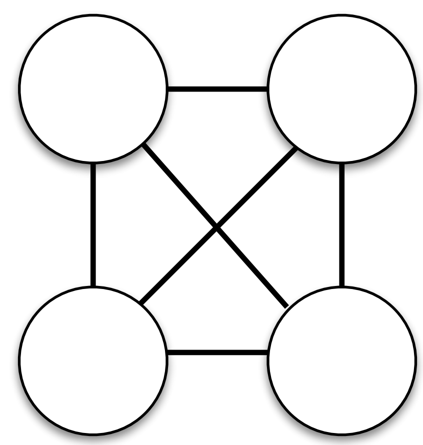
- A **complete graph** has edges between every pair of vertices.



N=3

Complete Graphs

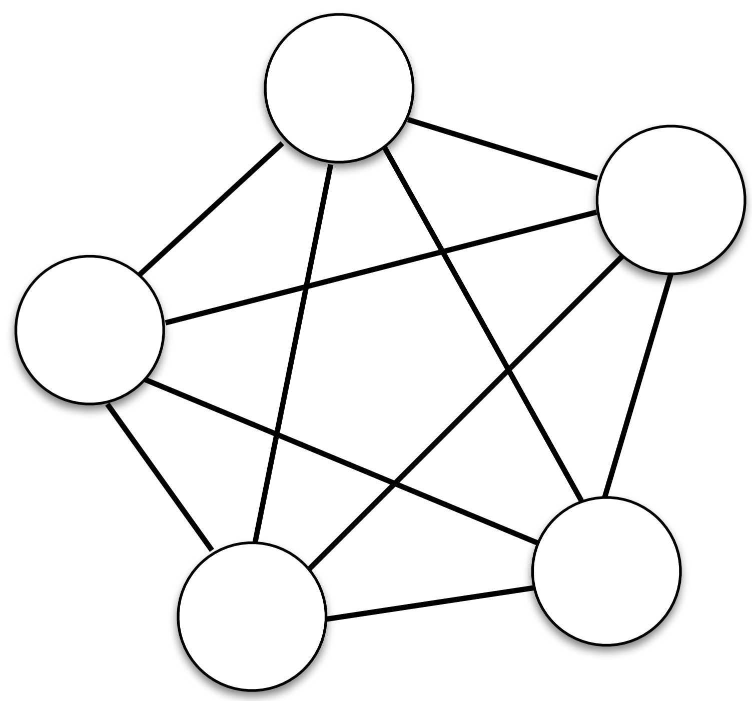
- A **complete graph** has edges between every pair of vertices.



N=4

Complete Graphs

- A **complete graph** has edges between every pair of vertices.

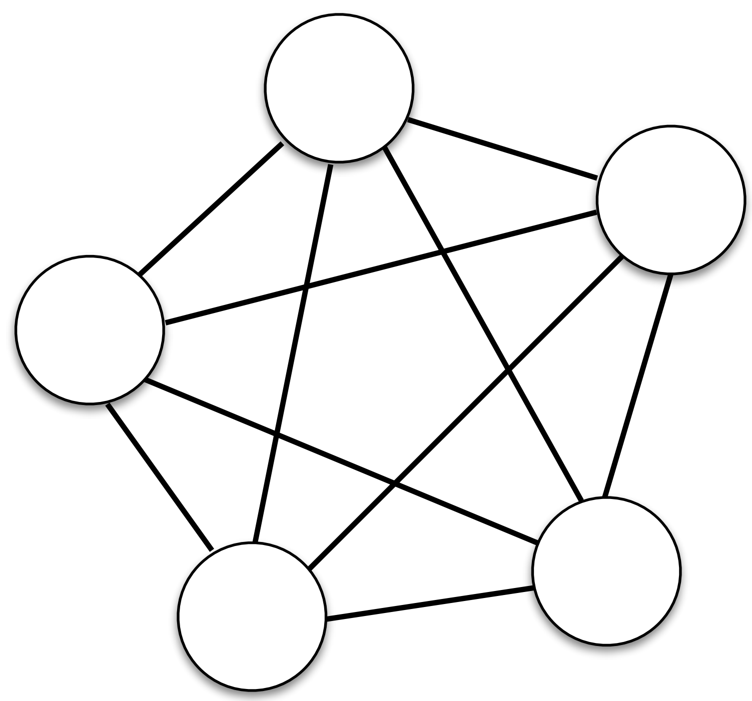


N=5

How many edges are there in a complete graph of size N?

Complete Graphs

- A **complete graph** has edges between every pair of vertices.



N=5

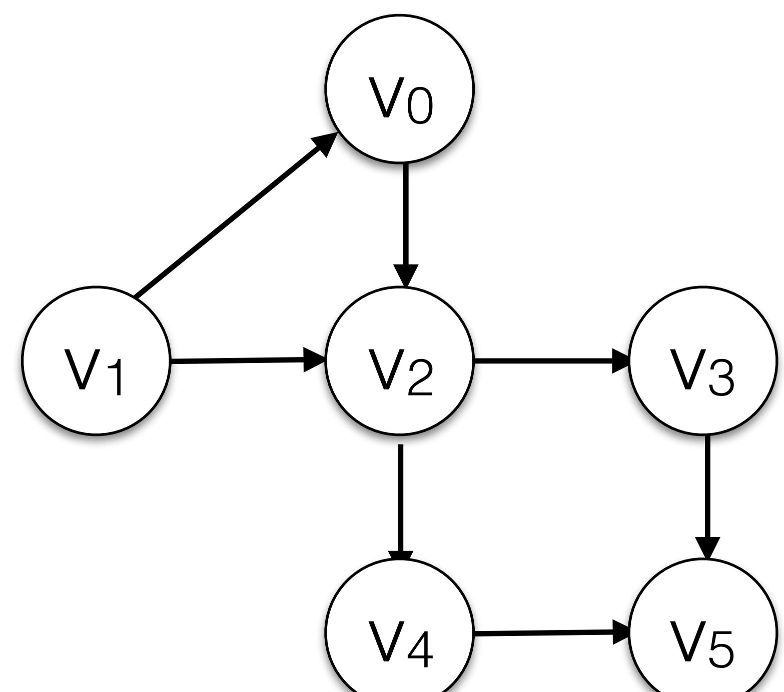
How many edges are there in a complete graph of size N?

$$\sum_{i=1}^{N-1} i = \frac{N \cdot (N-1)}{2}$$

Representing Graphs

- Represent graph G = (E,V), option 1:
  - N x N **Adjacency Matrix** represented as 2-dimensional `Boolean[][]`.
  - A[u][v] = `true` if (u,v) ∈ E, else false

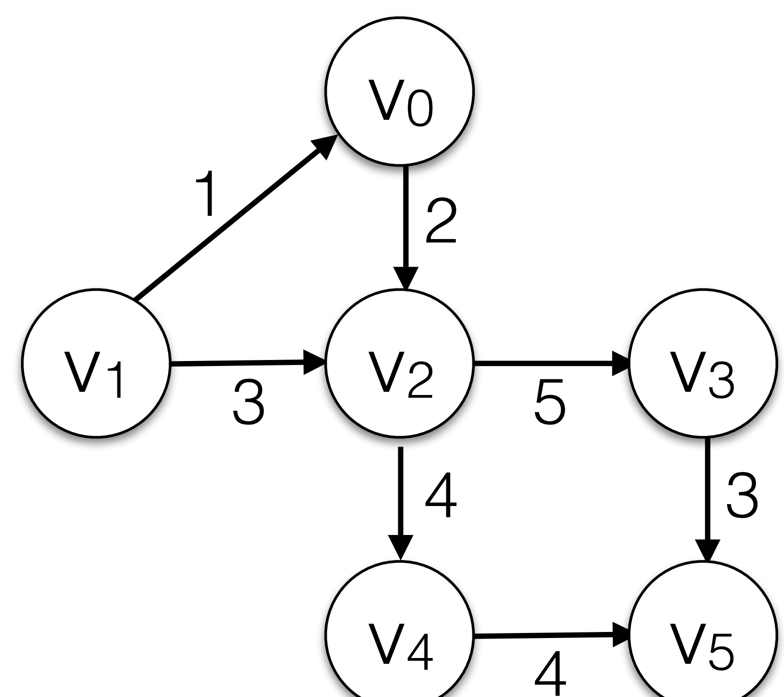
	0	1	2	3	4	5
0	f	f	t	f	f	f
1	t	f	t	f	f	f
2	f	f	f	t	t	f
3	f	f	f	f	f	t
4	f	f	f	f	f	t
5	f	f	f	f	f	f



Representing Graphs

- Represent graph G = (E,V), option 1:
  - N x N **Adjacency Matrix** represented as 2-dimensional `Integer[][]`.
  - A[u][v] = cost(u,v) if (u,v) ∈ E, else ∞

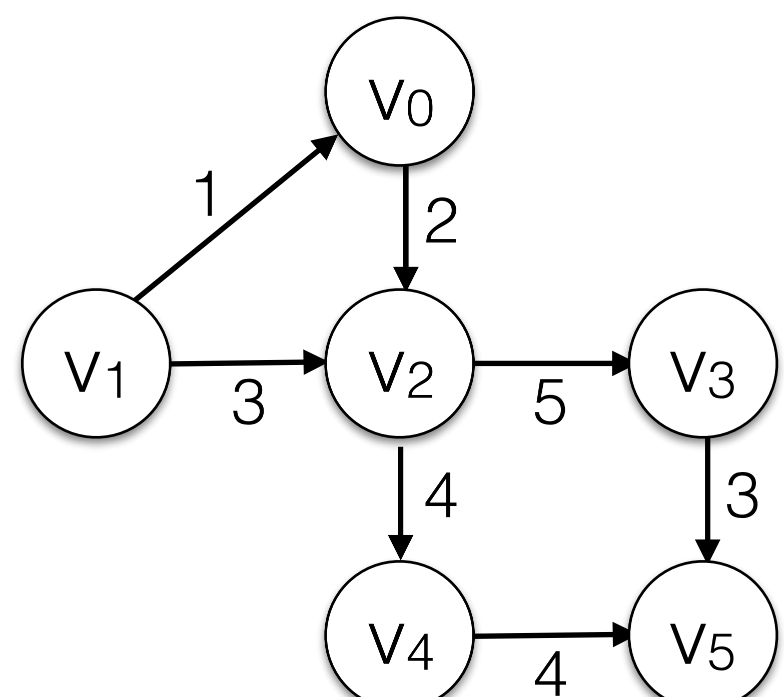
	0	1	2	3	4	5
0	∞	∞	2	∞	∞	∞
1	1	∞	3	∞	∞	∞
2	∞	∞	∞	5	4	∞
3	∞	∞	∞	∞	∞	3
4	∞	∞	∞	∞	∞	4
5	∞	∞	∞	∞	∞	∞



Representing Graphs

- Problem of Adjacency Matrix representation:
  - For **sparse** graphs (that contain much less than $|V|^2$ edges), a lot of array space is wasted.

	0	1	2	3	4	5
0	∞	∞	2	∞	∞	∞
1	1	∞	3	∞	∞	∞
2	∞	∞	∞	5	4	∞
3	∞	∞	∞	∞	∞	3
4	∞	∞	∞	∞	∞	4
5	∞	∞	∞	∞	∞	∞



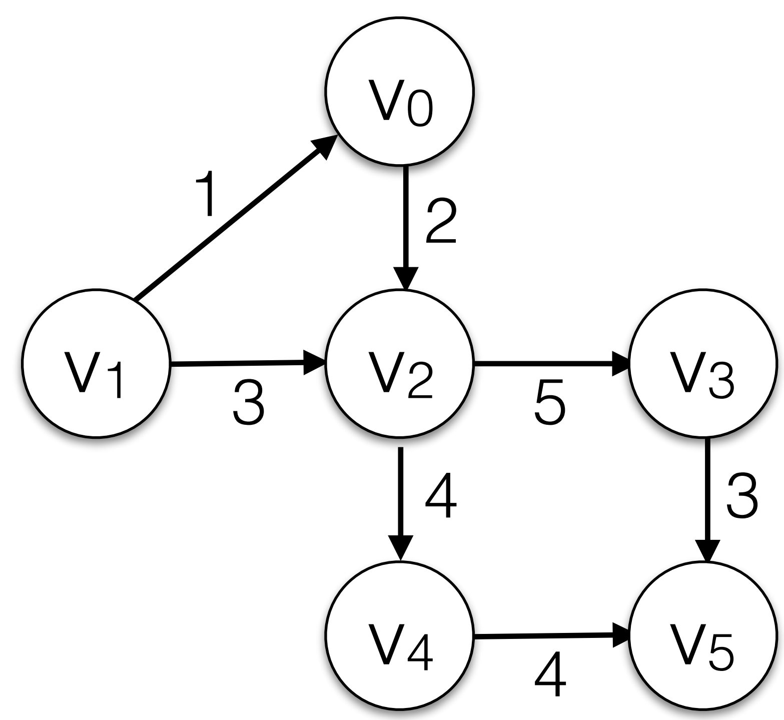
Representing Graphs

- Problem of Adjacency Matrix representation:

Space requirement: $\Theta(|V|^2)$

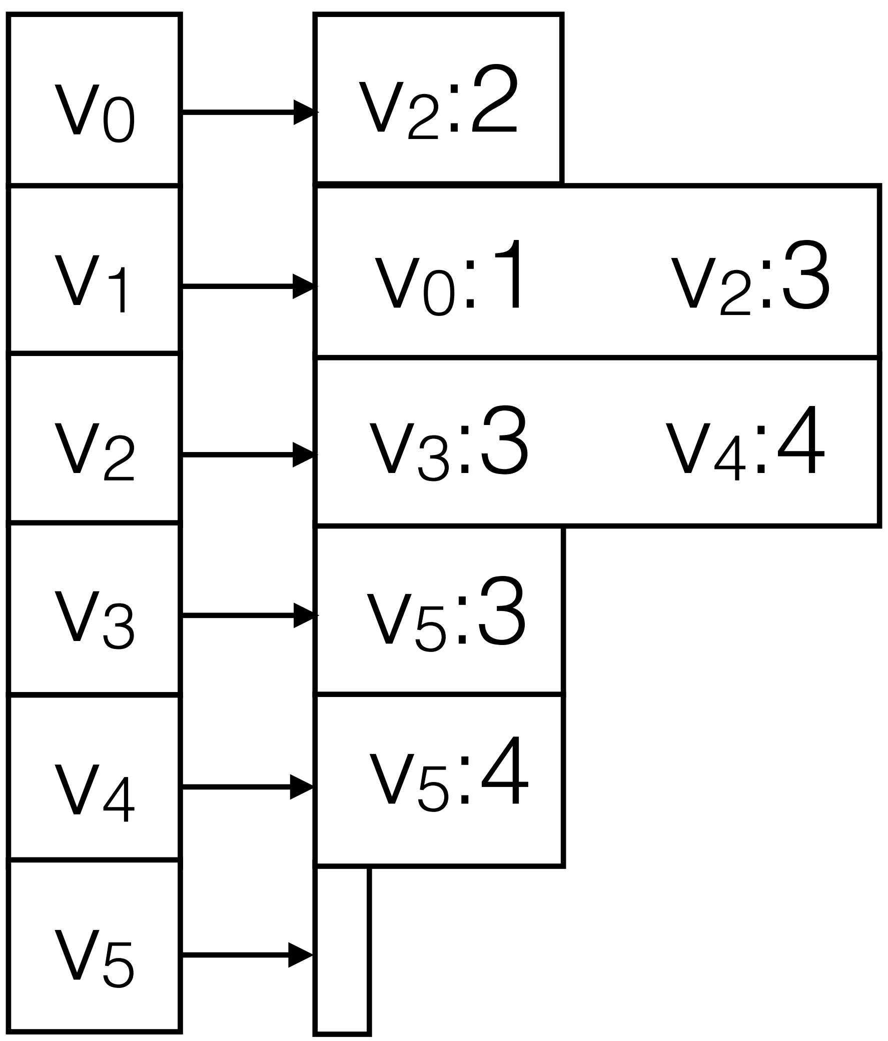
  - For **sparse** graphs (that contain much less than $|V|^2$ edges), a lot of array space is wasted.

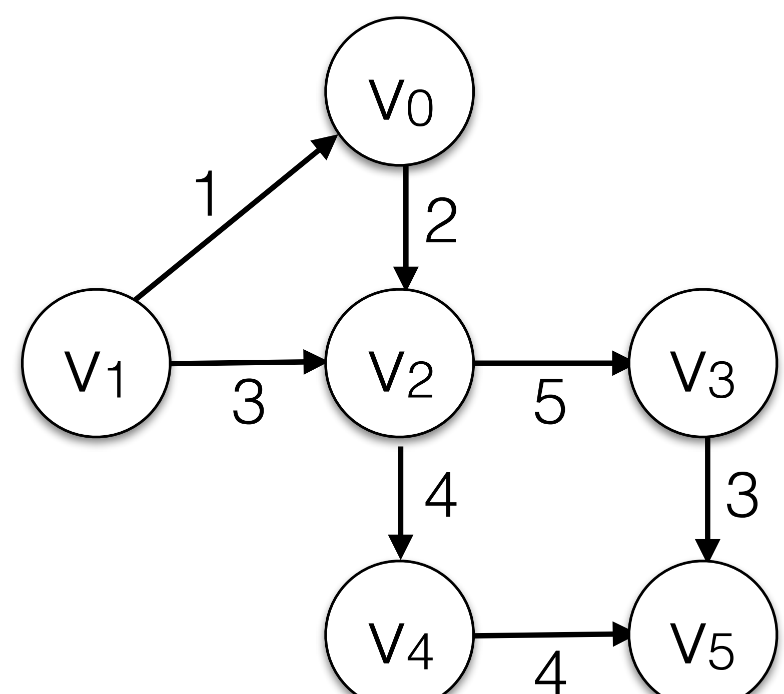
	0	1	2	3	4	5
0	∞	∞	2	∞	∞	∞
1	1	∞	3	∞	∞	∞
2	∞	∞	∞	5	4	∞
3	∞	∞	∞	∞	∞	3
4	∞	∞	∞	∞	∞	4
5	∞	∞	∞	∞	∞	∞



Representing Graphs

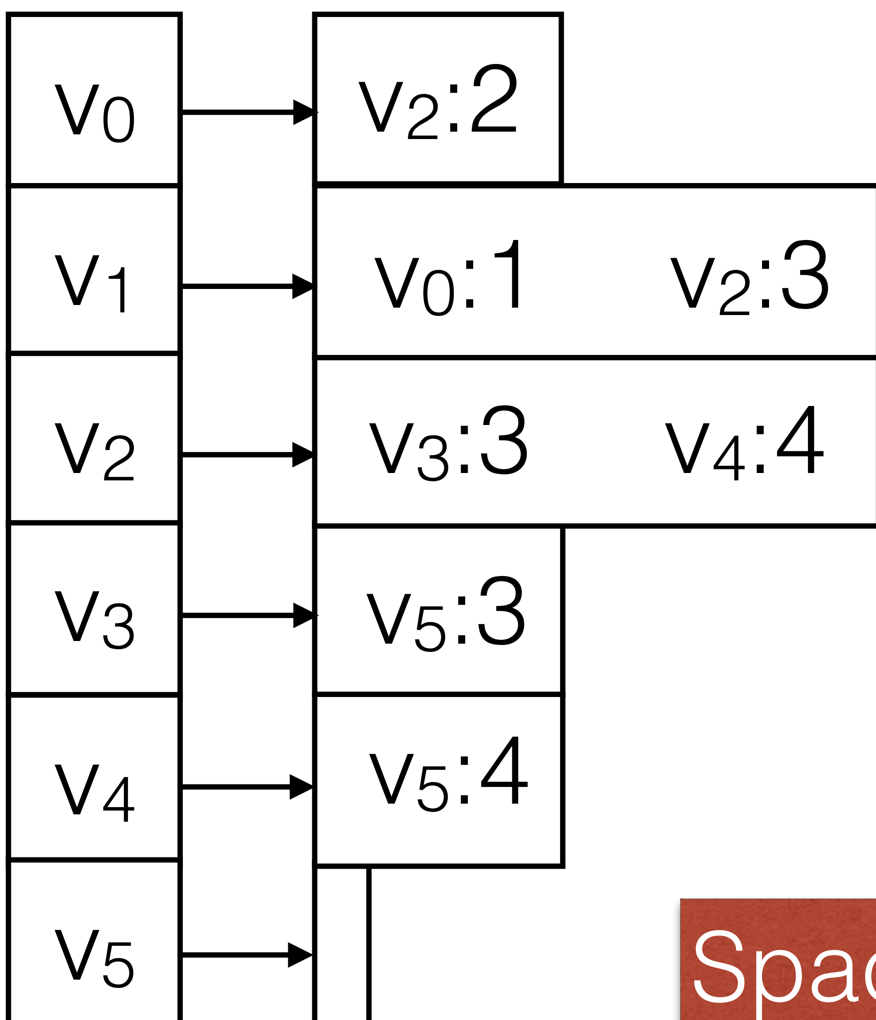
- Represent graph G = (E,V), option 2: **Adjacency Lists**
  - For each vertex, keep a list of all adjacent vertices.

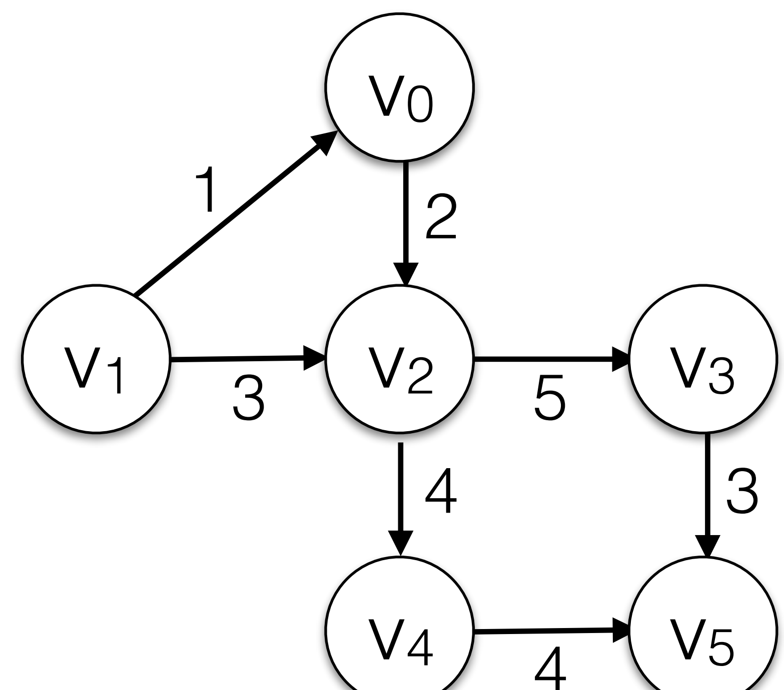




Representing Graphs

- Represent graph G = (E,V), option 2: **Adjacency Lists**
  - For each vertex, keep a list of all adjacent vertices.





Space requirement: $\Theta(|V| + |E|)$

Storing Adjacency Lists

- If we construct a graph (or read it in from some specification), a LinkedList is better than an ArrayList because we don't know how many adjacent vertices there are for each vertex.
- Create an instance of a Vertex class for each vertex and keep adjacency list in this object.
- Can also keep an index to quickly access vertices by name.

http://www.cs.columbia.edu/~bauer/cs3134/code/week11/BasicGraph.java