Social Media Mining

Graph Essentials

Dear instructors/users of these slides:

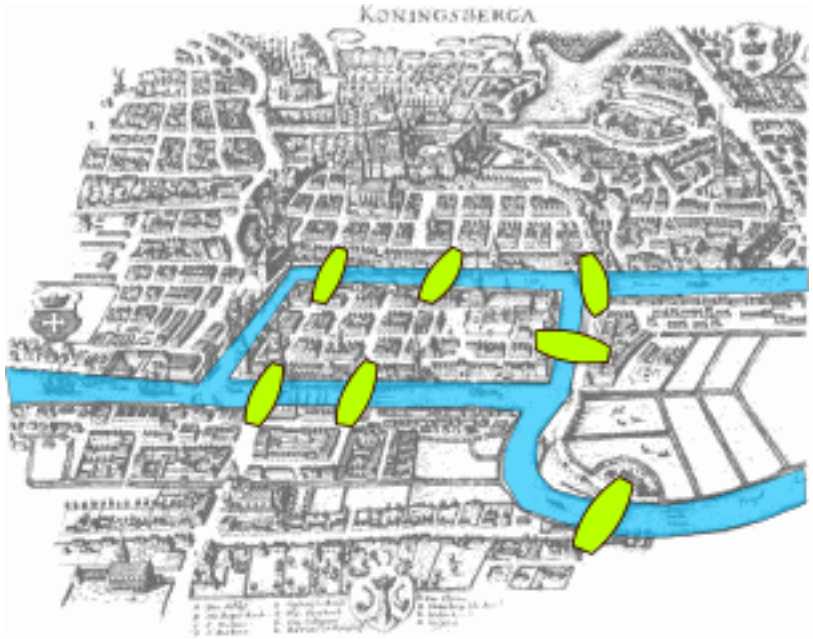
Please feel free to include these slides in your own material, or modify them as you see fit. If you decide to incorporate these slides into your presentations, please include the following note:

R. Zafarani, M. A. Abbasi, and H. Liu, *Social Media Mining: An Introduction*, Cambridge University Press, 2014.
Free book and slides at **http://socialmediamining.info/**

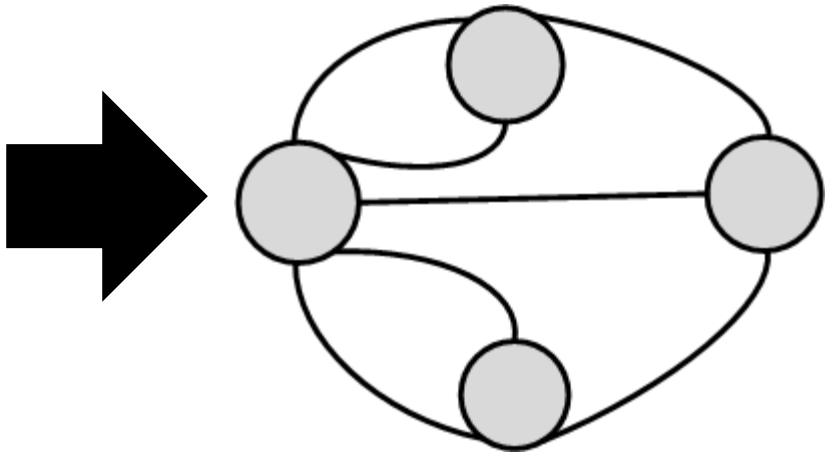
or include a link to the website:
http://socialmediamining.info/

Bridges of Konigsberg

- There are **2 islands** and **7 bridges** that connect the islands and the mainland
- Find a path that crosses each bridge exactly once



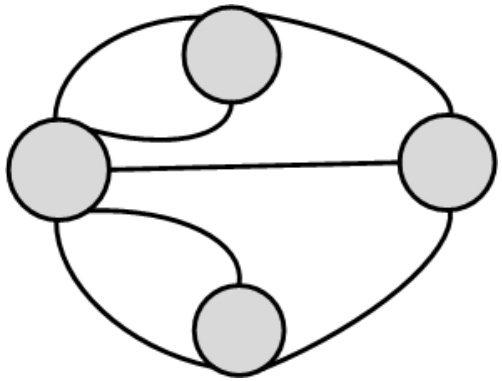
City Map (From Wikipedia)



Graph Representation

Modeling the Problem by Graph Theory

- The key to solve this problem is an ingenious graph representation
- Euler proved that since except for the starting and ending point of a walk, one has to enter and leave all other nodes, thus these nodes should have an even number of bridges connected to them
- This property does not hold in this problem

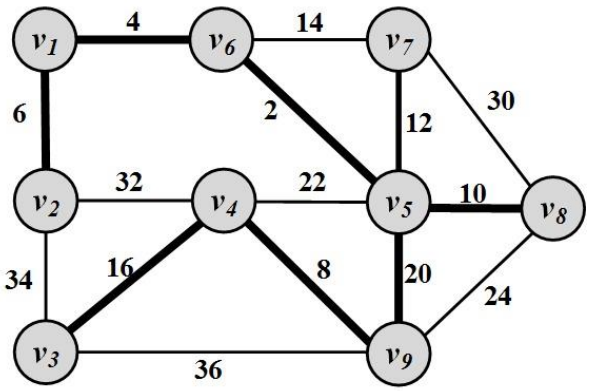


Networks

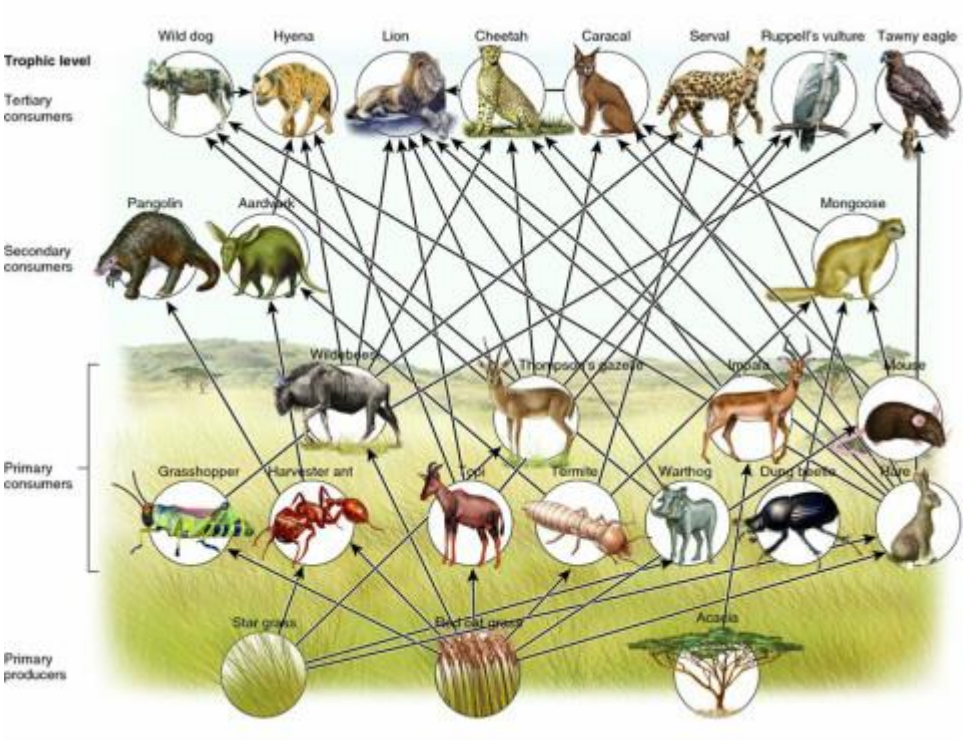
- A network is a graph.
  - Elements of the network have meanings
- Network problems can usually be represented in terms of graph theory

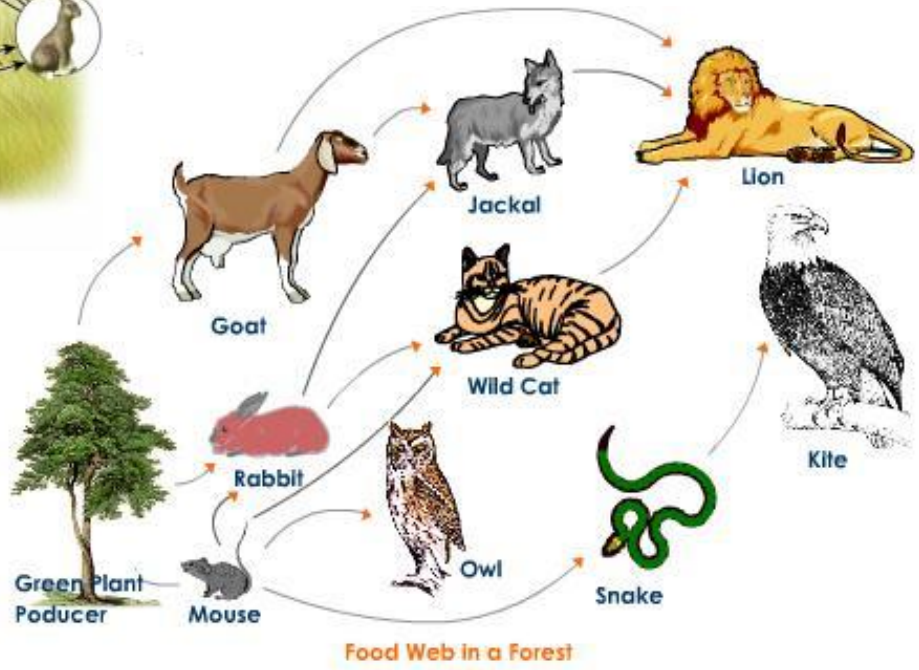
Twitter example:

- Given a piece of information, a network of individuals, and the cost to propagate information among any connected pair, find the minimum cost to disseminate the information to all individuals.



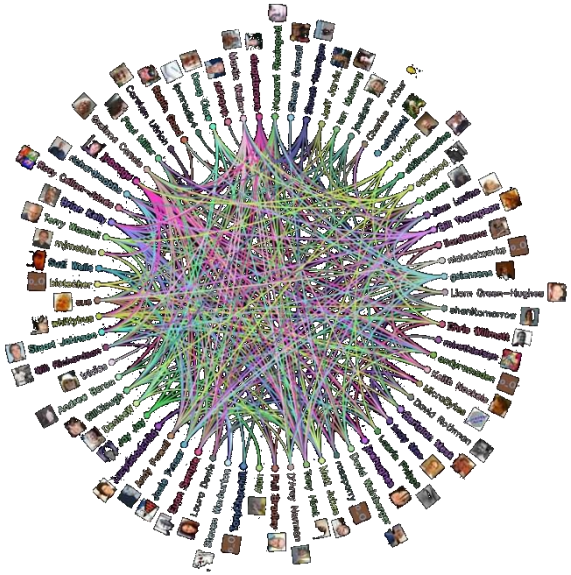
Food Web

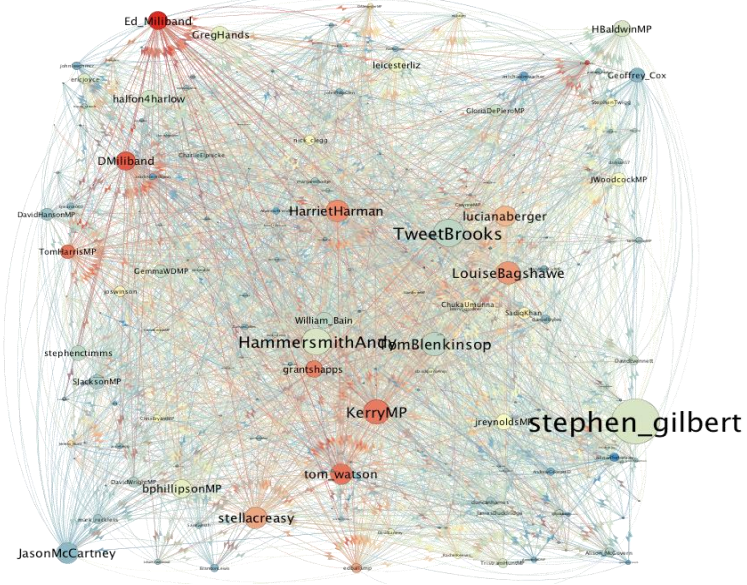

Trophic level
Tertiary consumers
Secondary consumers
Primary consumers
Primary producers
Wild dog
Hyena
Lion
Cheetah
Caracal
Serval
Ruppell's vulture
Tawny eagle
Pangolin
Aardvark
Mongoose
Wildebeest
Thomson's gazelle
Impala
Mouse
Grasshopper
Harvester ant
Topi
Termite
Warthog
Dung beetle
Hare
Star grass
Acacia

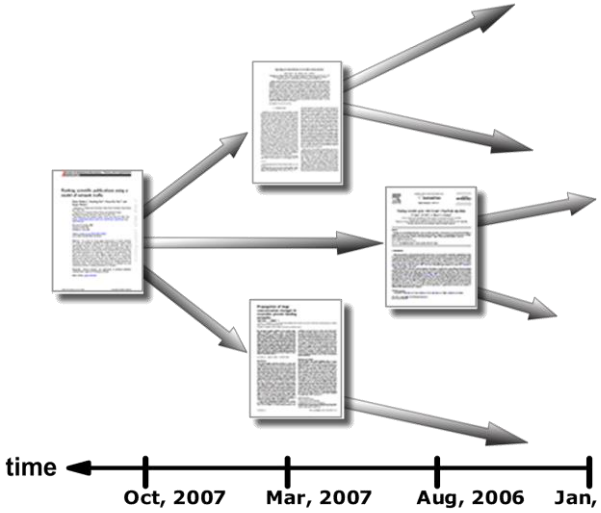

Goat
Jackal
Lion
Wild Cat
Kite
Rabbit
Owl
Snake
Green Plant
Poducer
Mouse
Food Web in a Forest

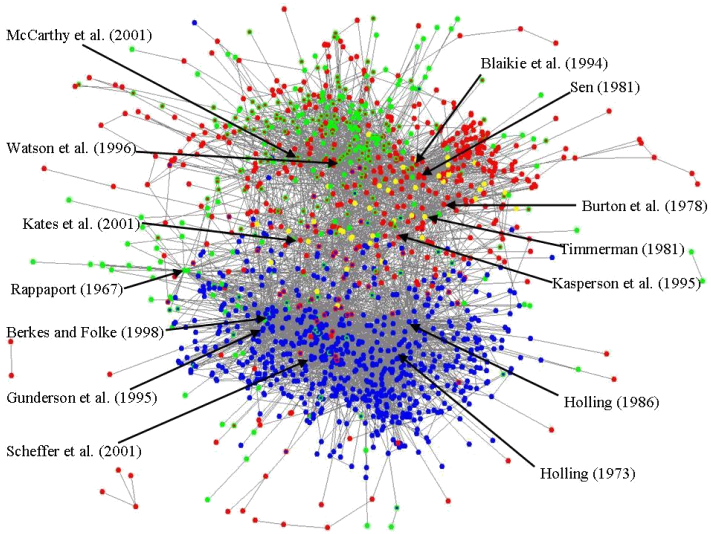
Network are Pervasive

Twitter Networks



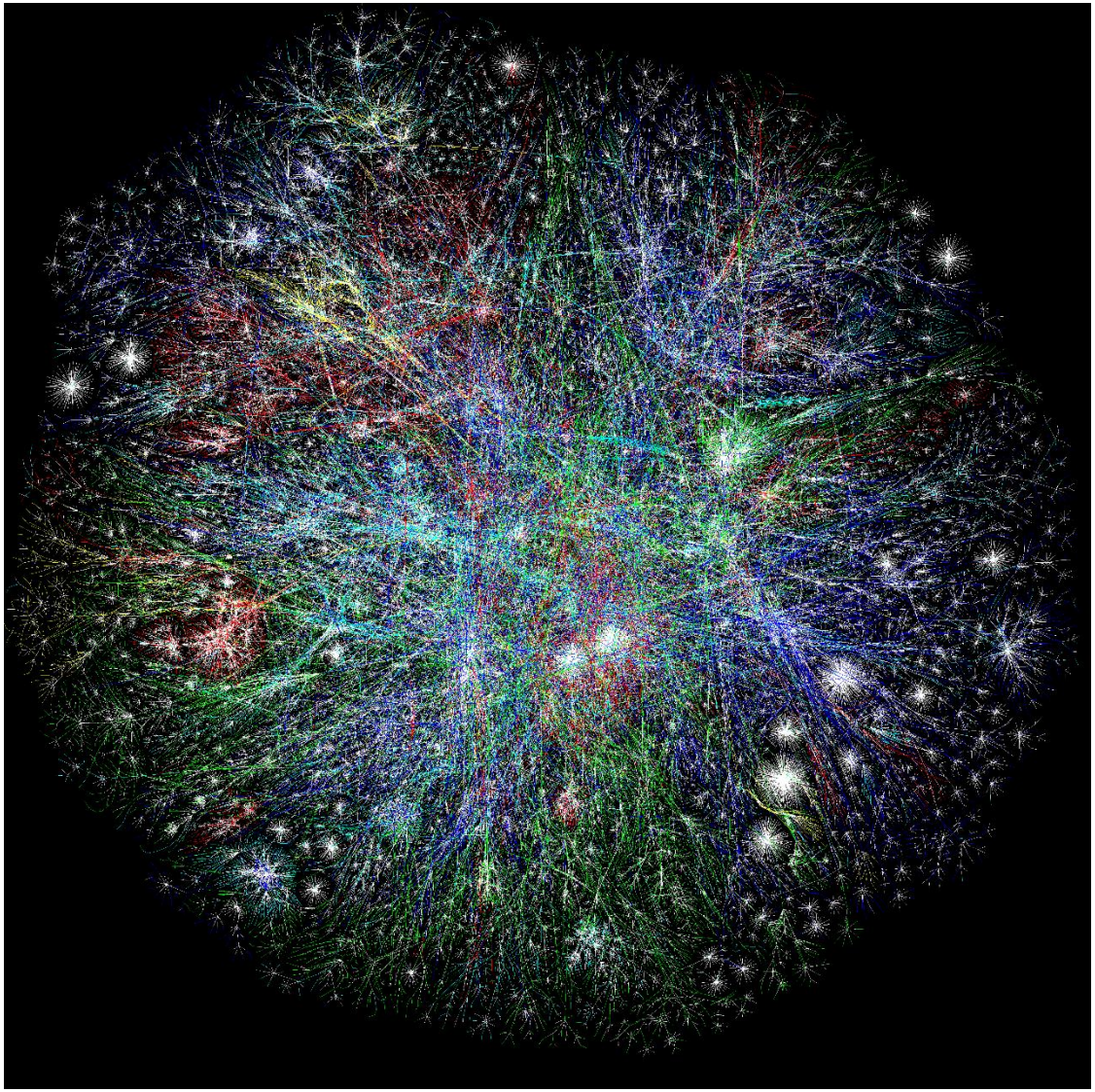






Citation Networks

Internet



Network of the US Interstate Highways

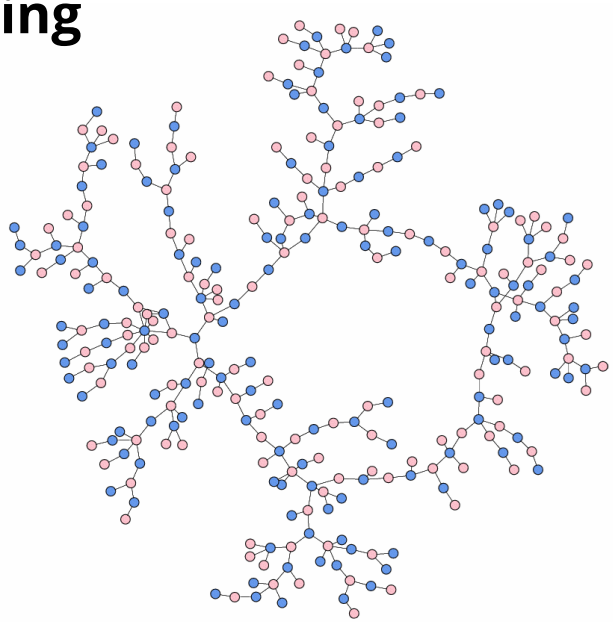
NY State Road Network

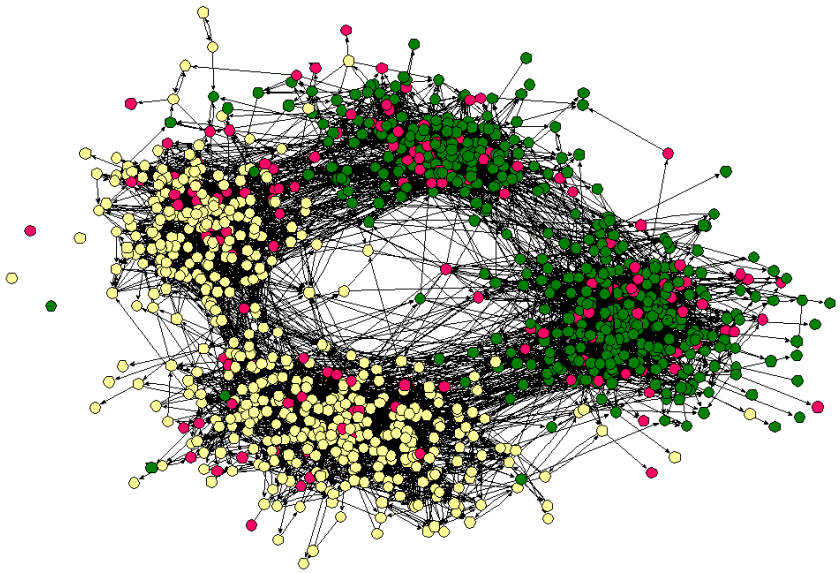
Social Networks and Social Network Analysis

- A social network
  - A network where elements have a social structure
    - A set of actors (such as individuals or organizations)
    - A set of ties (connections between individuals)
- Social networks examples:
  - your family network, your friend network, your colleagues ,etc.
- To analyze these networks we can use Social Network Analysis (SNA)
- Social Network Analysis is an interdisciplinary field from social sciences, statistics, graph theory, complex networks, and now computer science

Social Networks: Examples

High school dating





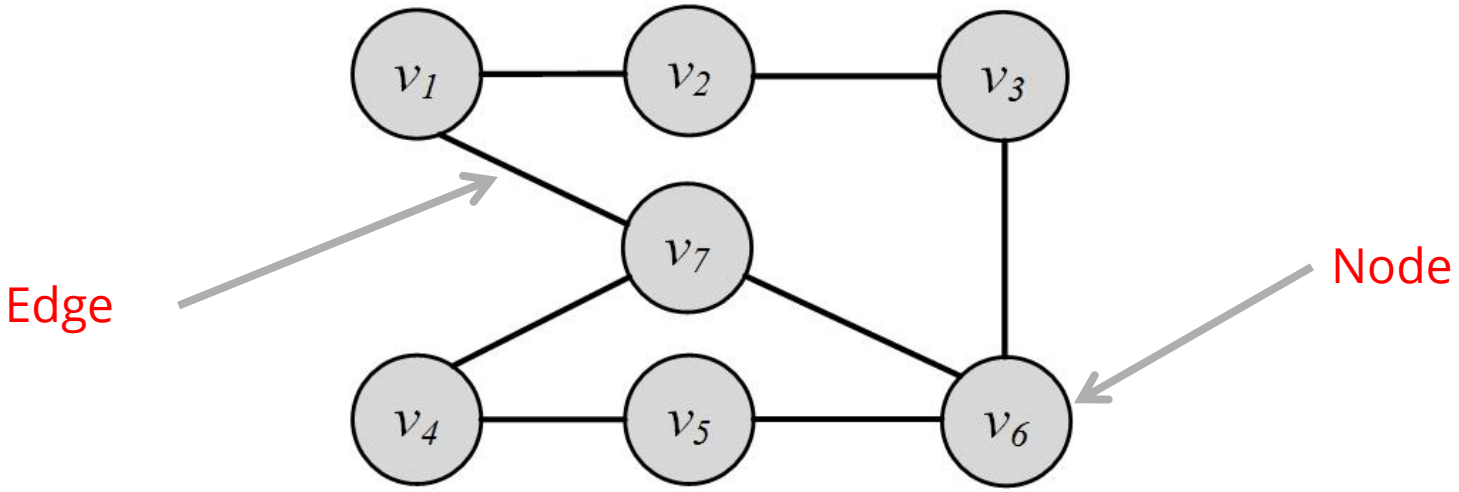
High school friendship

Graph Basics

Nodes and Edges

A network is a graph, or a collection of points connected by lines

- Points are referred to as **nodes, actors,** or **vertices** (plural of **vertex**)
- Connections are referred to as **edges** or **ties**



Nodes or Actors

- In a friendship social graph, nodes are people and any pair of people connected denotes the friendship between them
- Depending on the context, these nodes are called nodes, or actors
  - In a web graph, "*nodes*" represent sites and the connection between nodes indicates web-links between them
  - In a social setting, these nodes are called actors

$$V = \{v_1, v_2, \ldots, v_n\}$$

  - The size of the graph is $|V| = \mathbf{n}$

Edges

- Edges connect nodes and are also known as **ties** or **relationships**

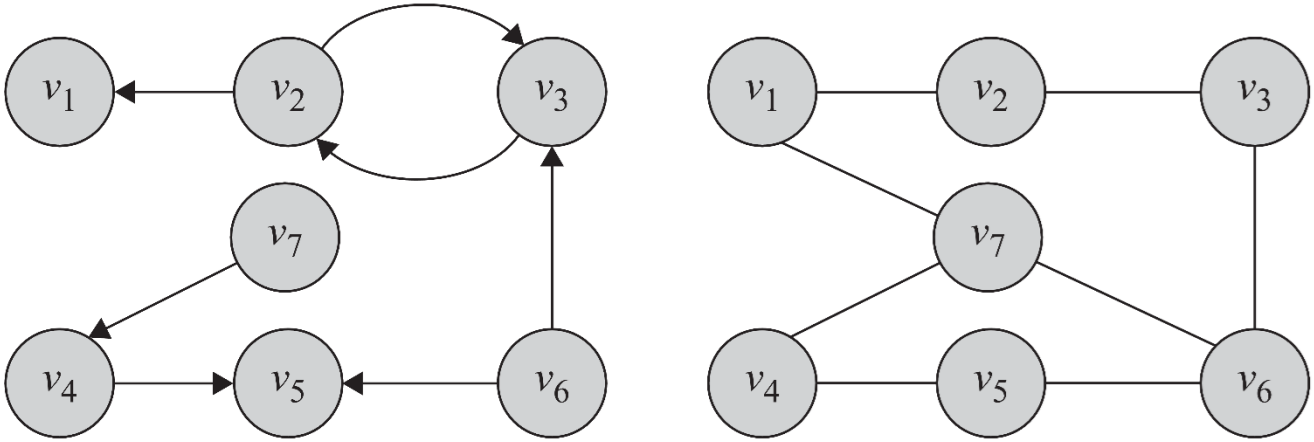
- In a social setting, where nodes represent social entities such as people, edges indicate internode relationships and are therefore known as relationships or (social) ties

$$E = \{e_1, e_2, \ldots, e_m\}$$

- Number is edges (size of the edge-set) is denoted as $|E| = \mathbf{m}$

Directed Edges and Directed Graphs

- Edges can have directions. A directed edge is sometimes called an arc



(a) Directed Graph (b) Undirected Graph

- Edges are represented using their end-points $e(v_2, v_1)$.
- In undirected graphs both representations are the same

Neighborhood and Degree (In-degree, out-degree)

For any node v, in an undirected graph, the set of nodes it is connected to via an edge is called its neighborhood and is represented as $N(v)$

- *In directed graphs we have incoming neighbors $N_{in}(v)$ (nodes that connect to v) and outgoing neighbors $N_{out}(v)$.*

The number of edges connected to one node is the degree of that node (the size of its neighborhood)

- Degree of a node i is usually presented using notation d_i

In Directed graphs:

d_i^{in} – In-degrees is the number of edges pointing towards a node

d_i^{out} – Out-degree is the number of edges pointing away from a node

Degree and Degree Distribution

- **Theorem 1.** The summation of degrees in an undirected graph is twice the number of edges

$$\sum_i d_i = 2|E|$$

- **Lemma 1.** The number of nodes with odd degree is even
- **Lemma 2.** In any directed graph, the summation of in-degrees is equal to the summation of out-degrees,

$$\sum_i d_i^{out} = \sum_j d_j^{in}$$

Degree Distribution

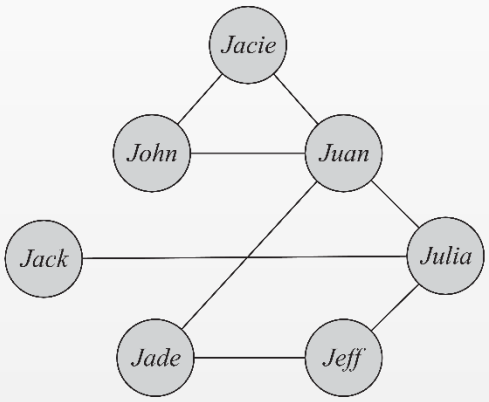
When dealing with very large graphs, how nodes' degrees are distributed is an important concept to analyze and is called ***Degree Distribution***

$$\pi(d) = \{d_1, d_2, \ldots, d_n\}$$ (Degree sequence)

$$p_d = \frac{n_d}{n}$$

n_d is the number of nodes with degree d

$$\sum_{d=0}^{\infty} p_d = 1$$



$$p_1 = \frac{1}{7}, p_2 = \frac{4}{7}, p_3 = \frac{1}{7}, p_4 = \frac{1}{7}$$

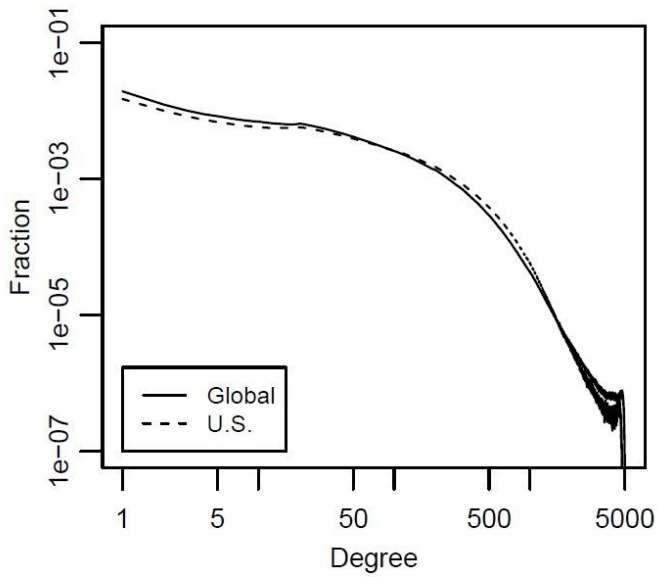
Degree Distribution Plot

The x-axis represents the degree and the y-axis represents the fraction of nodes having that degree

– On social networking sites

There exist many users with few connections and there exist a handful of users with very large numbers of friends.

(**Power-law degree distribution)**



Facebook Degree Distribution

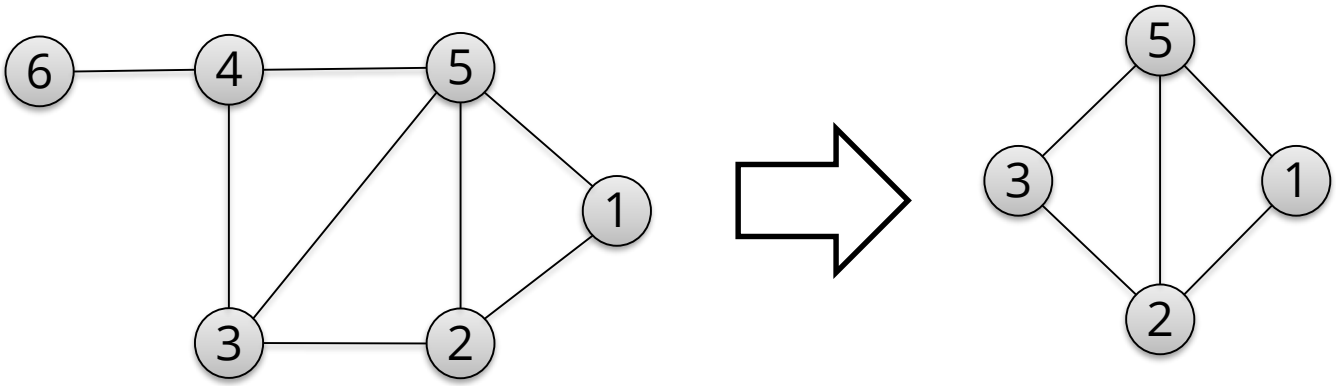
Subgraph

- Graph G can be represented as a pair $G(V, E)$

where V is the node set and E is the edge set

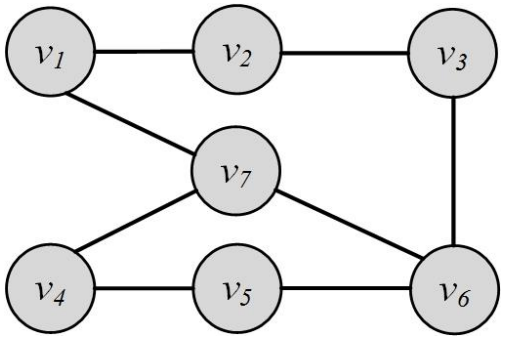
- $G'(V', E')$ is a subgraph of $G(V, E)$

$$V' \subseteq V$$
$$E' \subseteq (V' \times V') \cap E$$



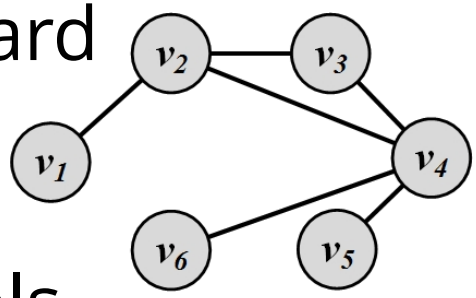
Graph Representation

- **Adjacency Matrix**
- **Adjacency List**
- **Edge List**



Graph Representation

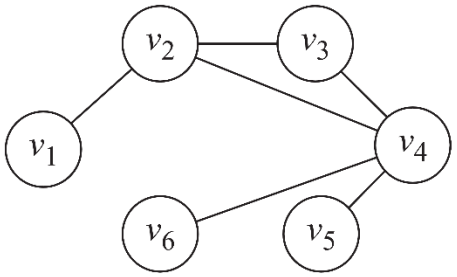
- Graph representation is straightforward and intuitive, but it cannot be effectively manipulated using mathematical and computational tools



- We are seeking representations that can store these two sets in a way such that
  - Does not lose information
  - Can be manipulated easily by computers
  - Can have mathematical methods applied easily

Adjacency Matrix (a.k.a. sociomatrix)

$$A_{ij} = \begin{cases} 1, \text{ if there is an edge between nodes } v_i \text{ and } v_j \\ 0, \text{ otherwise} \end{cases}$$



(a) Graph

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	1	0	0	0	0
v_2	1	0	1	1	0	0
v_3	0	1	0	1	0	0
v_4	0	1	1	0	1	1
v_5	0	0	0	1	0	0
v_6	0	0	0	1	0	0

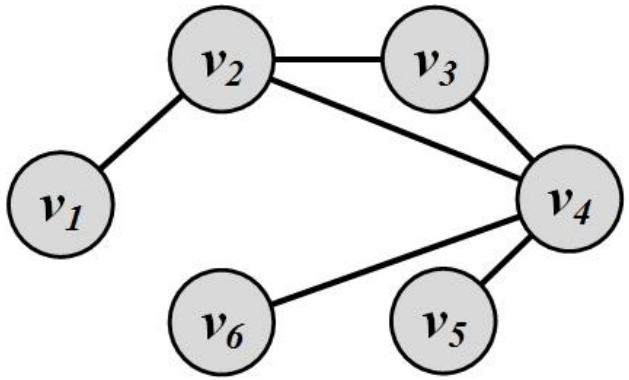
(b) Adjacency Matrix

Diagonal Entries are self-links or loops

Social media networks have very sparse Adjacency matrices

Adjacency List

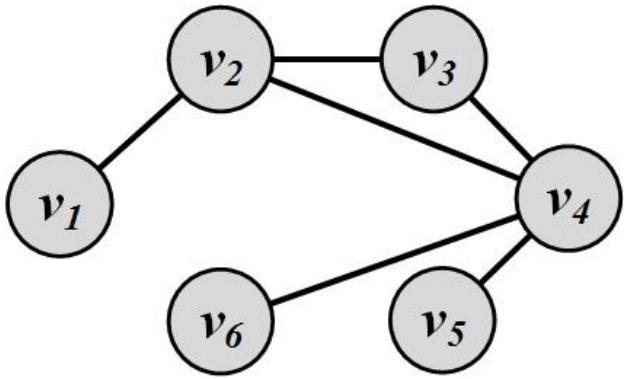
- In an adjacency list for every node, we maintain a list of all the nodes that it is connected to
- The list is usually sorted based on the node order or other preferences



Node	Connected To
v_1	v_2
v_2	v_1, v_3, v_4
v_3	v_2, v_4
v_4	v_2, v_3, v_5, v_6
v_5	v_4
v_6	v_4

Edge List

- In this representation, each element is an edge and is usually represented as (u, v), denoting that node u is connected to node v via an edge



(v_1, v_2)
(v_2, v_3)
(v_2, v_4)
(v_3, v_4)
(v_4, v_5)
(v_4, v_6)

Types of Graphs

- **Null, Empty, Directed/Undirected/Mixed, Simple/Multigraph, Weighted, Signed Graph, Webgraph**

Null Graph and Empty Graph

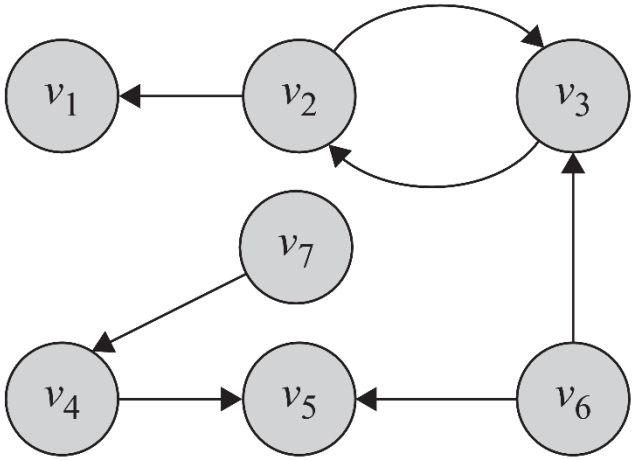
- A **null graph** is one where the node set is empty (there are no nodes)
  - Since there are no nodes, there are also no edges

$$G(V, E), V = E = \emptyset$$

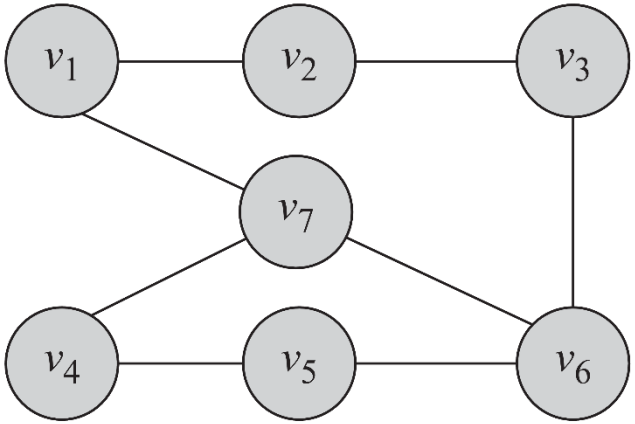
- An **empty graph** or **edge-less graph** is one where the edge set is empty, $E = \emptyset$

- The node set can be non-empty.
  - A null-graph is an empty graph.

Directed/Undirected/Mixed Graphs



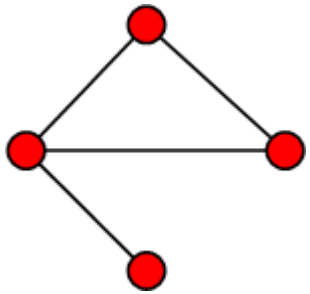
- The adjacency matrix for directed graphs is often not symmetric ($\boldsymbol{A \neq A^T}$)
  - $\boldsymbol{A_{ij} \neq Aji}$
  - We can have equality though



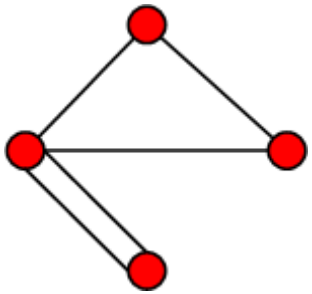
The adjacency matrix for undirected graphs is symmetric ($\boldsymbol{A = A^T}$)

Simple Graphs and Multigraphs

- Simple graphs are graphs where only a single edge can be between any pair of nodes
- Multigraphs are graphs where you can have multiple edges between two nodes and loops



Simple graph

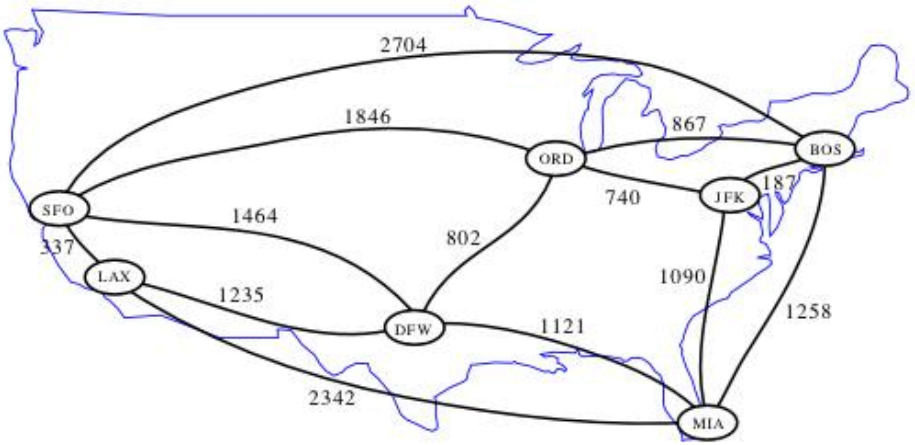


Multigraph

- The adjacency matrix for multigraphs can include numbers larger than one, indicating multiple edges between nodes

Weighted Graph

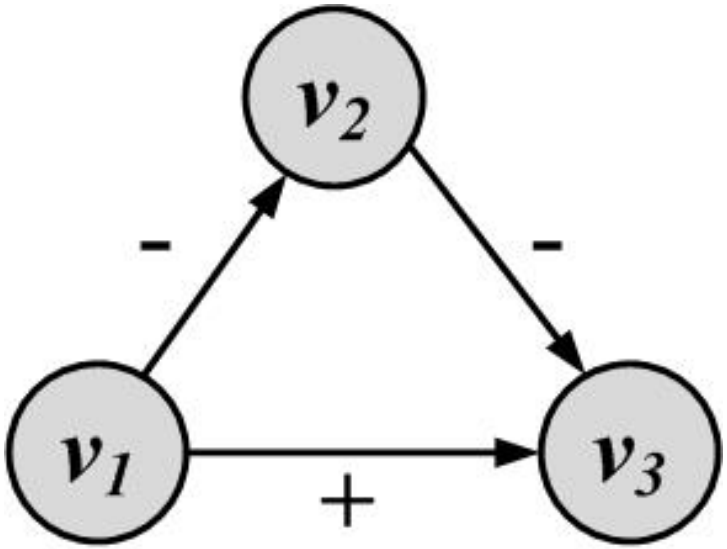
- A weighted graph $\boldsymbol{G}(\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{W})$ is one where edges are associated with weights
  - For example, a graph could represent a map where nodes are airports and edges are routes between them
    - The weight associated with each edge could represent the distance between the corresponding cities



$$A_{ij} = \begin{cases} w_{ij} \text{ or } w(i, j), w \in R \\ 0, \text{There is no edge between } v_i \text{ and } v_j \end{cases}$$

Signed Graph

- When weights are binary (0/1, -1/1, +/-) we have a signed graph

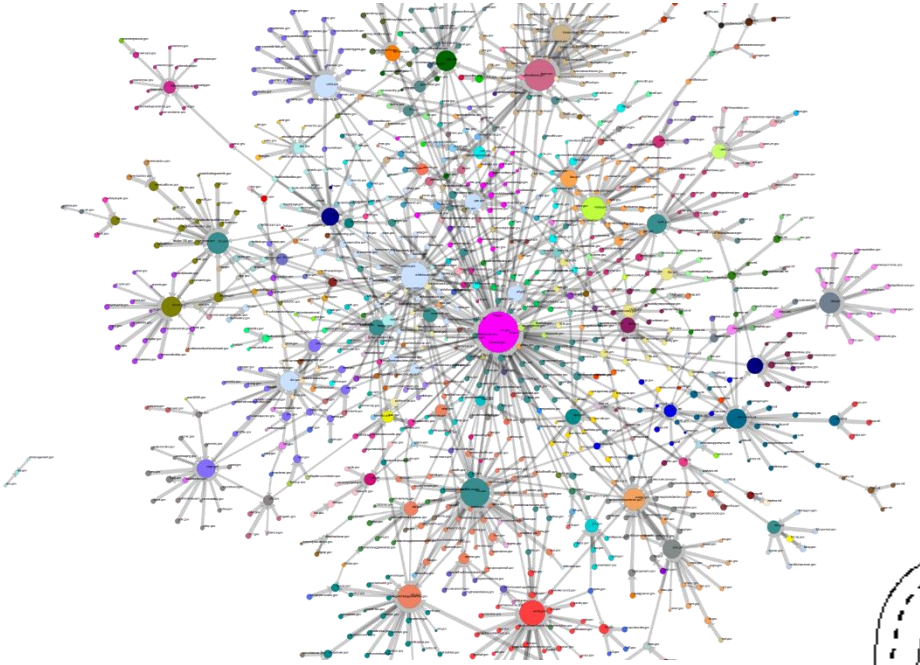


- It is used to represent friends or foes
- It is also used to represent social status

Webgraph

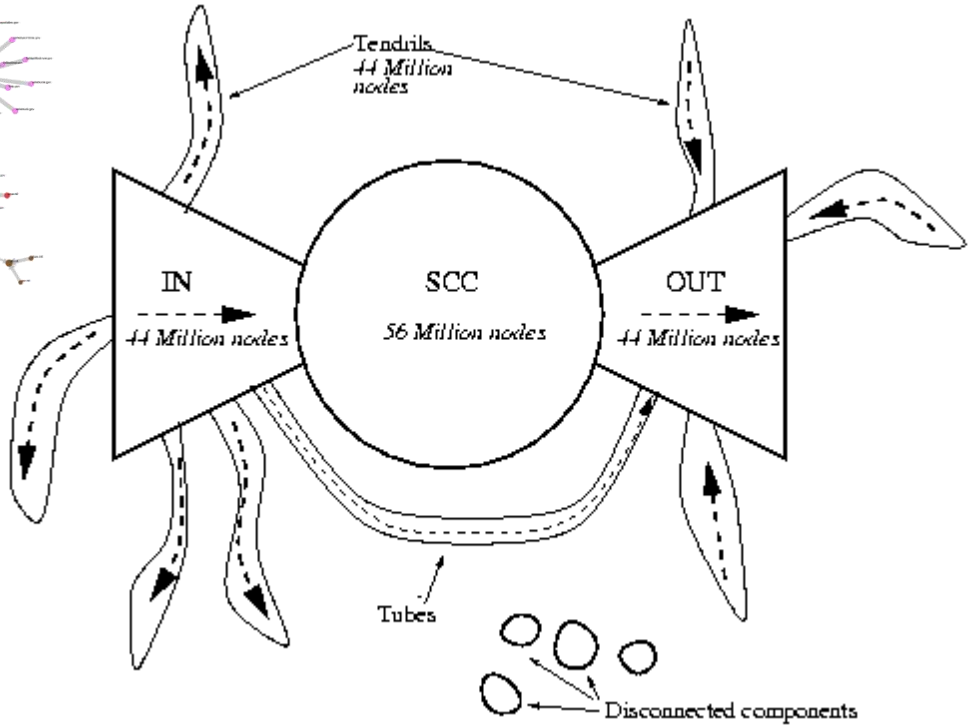
- A webgraph is a way of representing how internet sites are connected on the web
- In general, a web graph is a directed multigraph
- Nodes represent sites and edges represent links between sites.
- Two sites can have multiple links pointing to each other and can have loops (links pointing to themselves)

Webgraph



Government Agencies

Bow-tie structure



Broder et al –
200 million pages, 1.5 billion links

Connectivity in Graphs

- **Adjacent nodes/Edges, Walk/Path/Trail/Tour/Cycle**

Adjacent nodes and Incident Edges

Two nodes are adjacent if they are connected via an edge.

Two edges are incident, if they share on end-point

When the graph is directed, edge directions must match for edges to be incident

An edge in a graph can be traversed when one starts at one of its end-nodes, moves along the edge, and stops at its other end-node.

Walk, Path, Trail, Tour, and Cycle

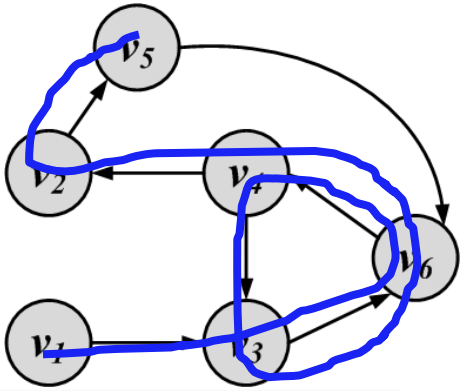
Walk: A walk is a sequence of incident edges visited one after another

- **Open walk**: A walk does not end where it starts
- **Closed walk**: A walk returns to where it starts

- Representing a walk:
  - A sequence of edges: $e_1, e_2, \ldots, en$
  - A sequence of nodes: $v_1, v_2, \ldots, vn$

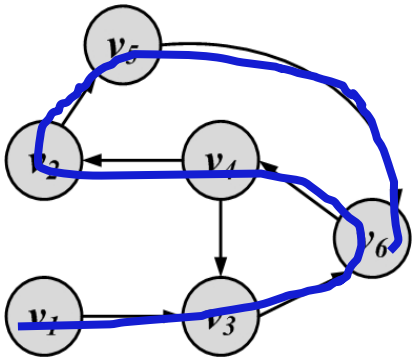
- Length of walk:
  the number of visited edges

Length of walk= 8



Trail

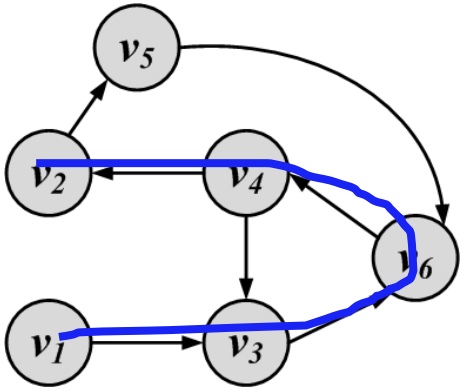
- A trail is a walk where **no edge is visited more than once** and all walk edges are distinct



- A closed trail (one that ends where it starts) is called a **tour** or **circuit**

Path

- A walk where **nodes and edges are distinct** is called a **path** and a closed path is called a **cycle**
- The length of a path or cycle is the number of edges visited in the path or cycle

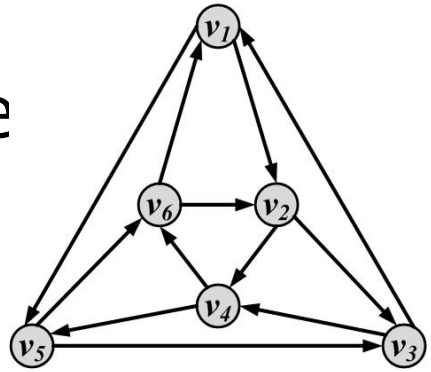


Length of path= 4

Examples

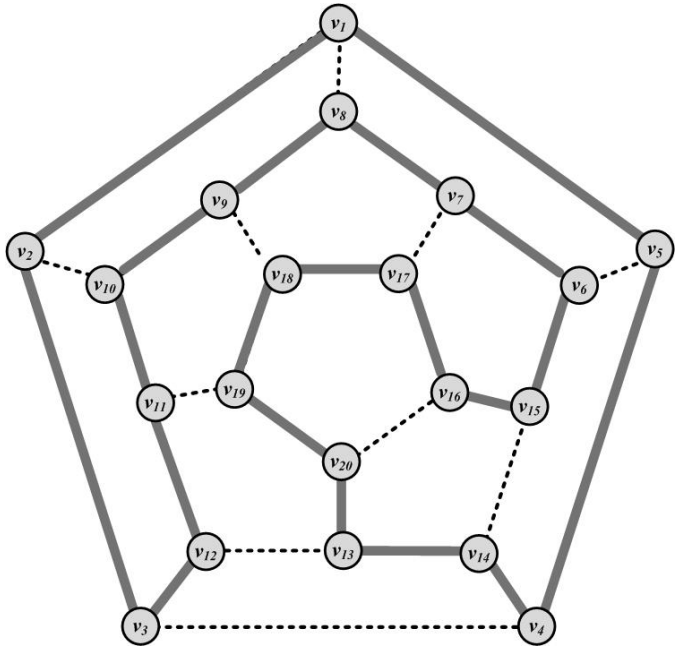
Eulerian Tour

- All edges are traversed only once
  - Konigsberg bridges



Hamiltonian Cycle

- A cycle that visits all nodes



Random walk

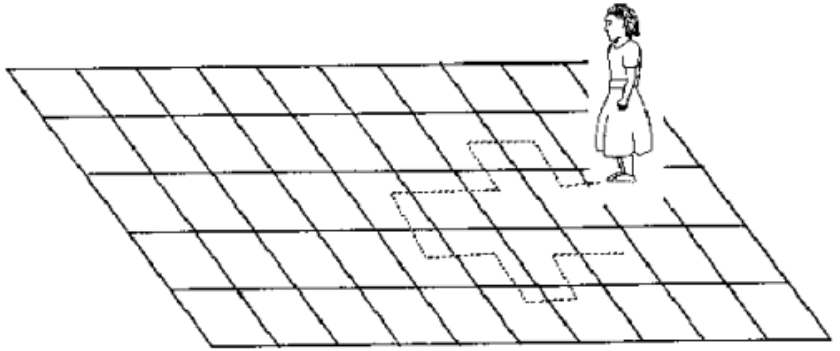
- A walk that in each step the next node is selected randomly among the neighbors
  - The weight of an edge can be used to define the probability of visiting it
  - For all edges that start at v_i the following equation holds

$$\sum_x w_{i,x} = 1, \forall i, j \quad w_{i,j} \geq 0$$

Random Walk: Example

Mark a spot on the ground

- Stand on the spot and flip the coin (or more than one coin depending on the number of choices such as left, right, forward, and backward)
- If the coin comes up heads, turn to the right and take a step
- If the coin comes up tails, turn to the left and take a step
- Keep doing this many times and see where you end up



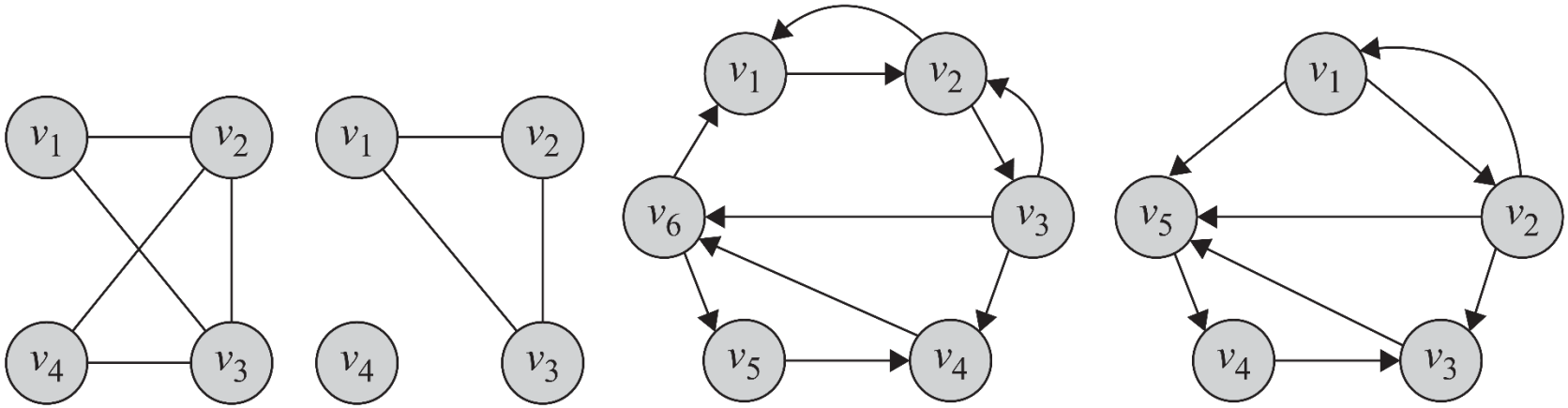
Connectivity

- **A node v_i is connected to node v_j** (or reachable from v_j) if it is adjacent to it or there exists a path from v_i to v_j.

- **A graph is connected**, if there exists a path between any pair of nodes in it
  - In a directed graph, **a graph is strongly connected** if there exists a directed path between any pair of nodes
  - In a directed graph, **a graph is weakly connected** if there exists a path between any pair of nodes, without following the edge directions

- A graph is **disconnected,** if it not connected.

Connectivity: Example

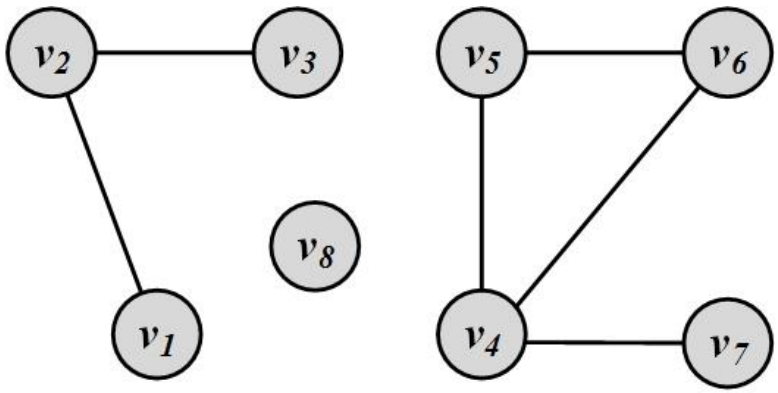


(a) Connected (b) Disconnected (c) Strongly connected (d) Weakly connected

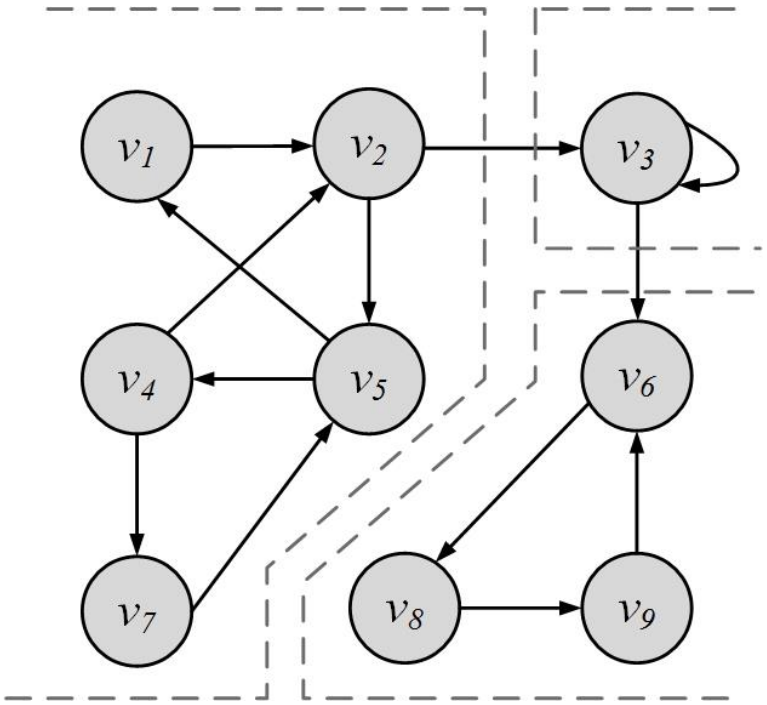
Component

- A **component** in an undirected graph is a connected **subgraph**, i.e., there is a path between every pair of nodes inside the component
- In directed graphs, we have a **strongly connected** components when there is a path from u to v and one from v to u for every pair of nodes u and v.
- The component is **weakly connected** if replacing directed edges with undirected edges results in a connected component

Component Examples:



3 components



3 Strongly-connected components

Shortest Path

- **Shortest Path** is the path between two nodes that has the shortest length.
  - We denote the length of the shortest path between nodes v_i and v_j as $l_{i,j}$

- The concept of the neighborhood of a node can be generalized using shortest paths. An **n-hop neighborhood** of a node is the set of nodes that are within n hops distance from the node.

Diameter

The diameter of a graph is the length of the longest shortest path between any pair of nodes between any pairs of nodes in the graph

$$diameter_G = \max_{(v_i,v_j)\in V\times V} l_{i,j}$$

- How big is the diameter of the web?

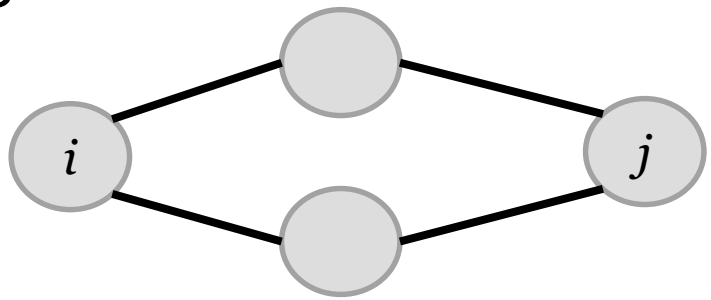
Adjacency Matrix and Connectivity

- Consider the following adjacency matrix

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1n} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ A_{d1} & A_{d2} & A_{d3} & \dots & A_{dn} \end{bmatrix}$$

- Number of Common neighbors between node i and node j

$$\sum_k A_{ik} A_{jk} = A_i \cdot A_j$$



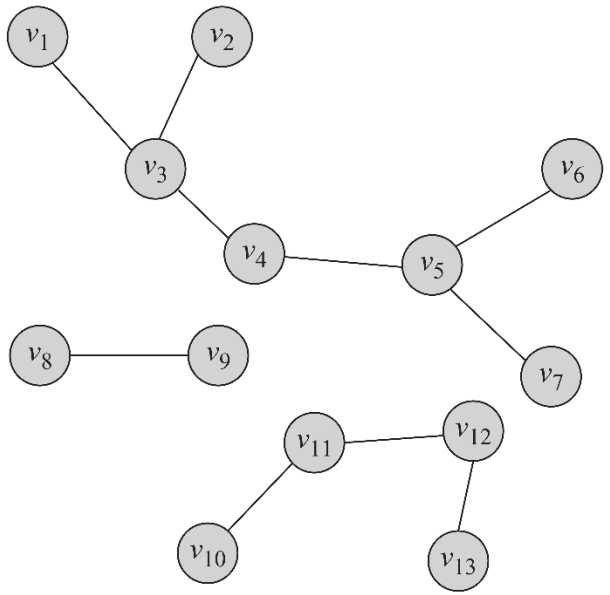
- That's element of [*ij*] of matrix $A \times A^T = A^2$
- Common neighbors are paths of length 2
- Similarly, what is A^3?

Special Graphs

Trees and Forests

- **Trees** are special cases of undirected graphs
- A tree is a graph structure that has no cycle in it
- In a tree, there is exactly one path between any pair of nodes
- In a tree: $|V| = |E| + 1$

- A set of disconnected trees is called a **forest**



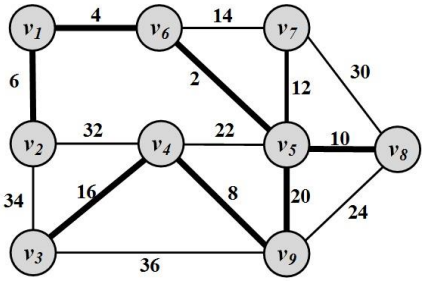
A forest containing 3 trees

Special Subgraphs

Spanning Trees

- For any connected graph, the spanning tree is a subgraph and a tree that includes all the nodes of the graph
- There may exist multiple spanning trees for a graph.
- In a weighted graph, the weight of a spanning tree is the summation of the edge weights in the tree.
- Among the many spanning trees found for a weighted graph, the one with the minimum weight is called the

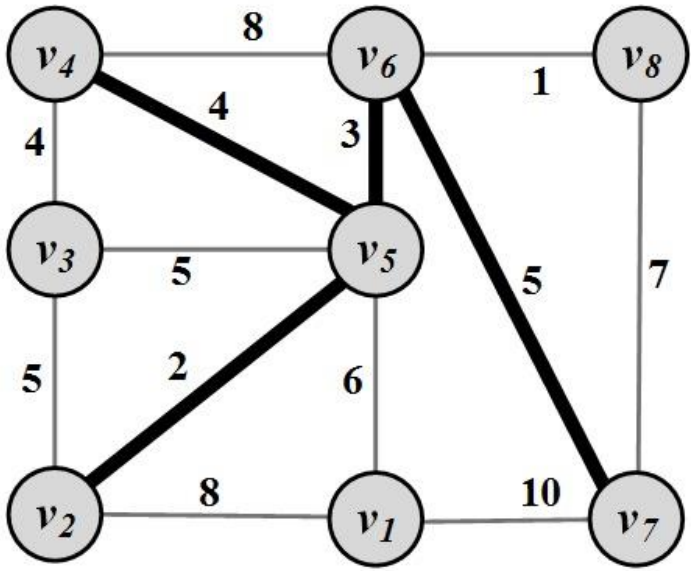
minimum spanning tree (MST)



Steiner Trees

Given a weighted graph G(V, E, W) and a *subset* of nodes $V' \subseteq V$ (terminal nodes), the Steiner tree problem aims to find a tree such that it spans all the V' nodes and the weight of this tree is minimized

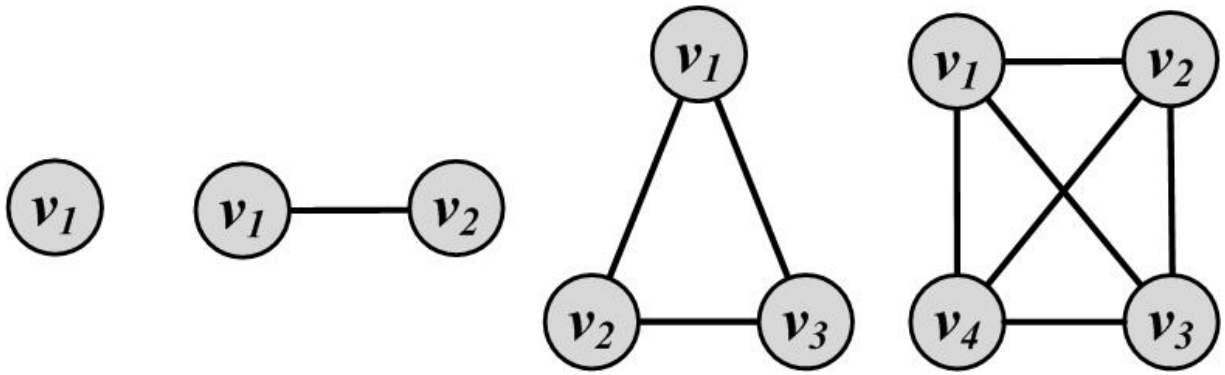
What can be the terminal set here?



Complete Graphs

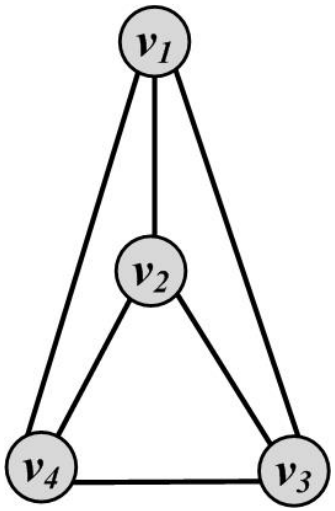
- A complete graph is a graph where for a set of nodes V, all possible edges exist in the graph
- In a complete graph, any pair of nodes are connected via an edge

$$|E| = \binom{|V|}{2}$$

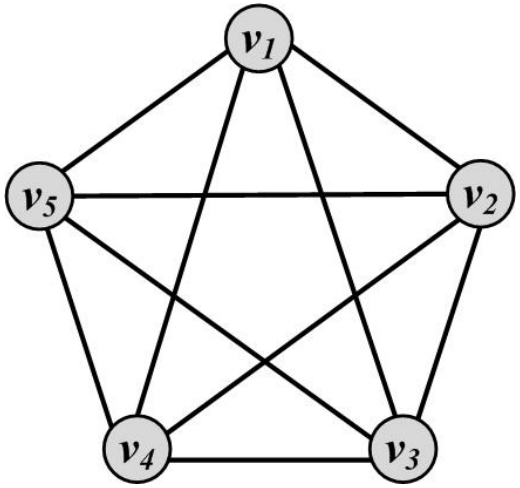


Planar Graphs

A graph that can be drawn in such a way that no two edges cross each other (other than the endpoints) is called planar



Planar Graph

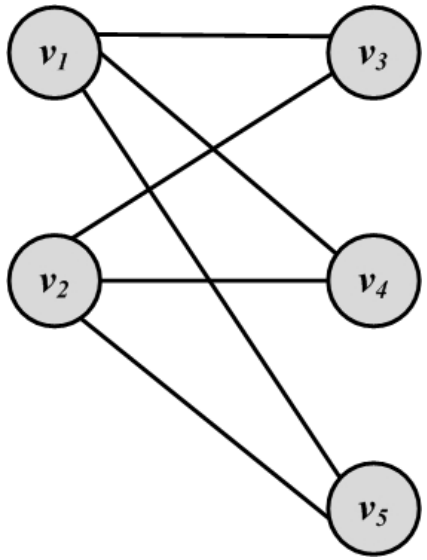


Non-planar Graph

Bipartite Graphs

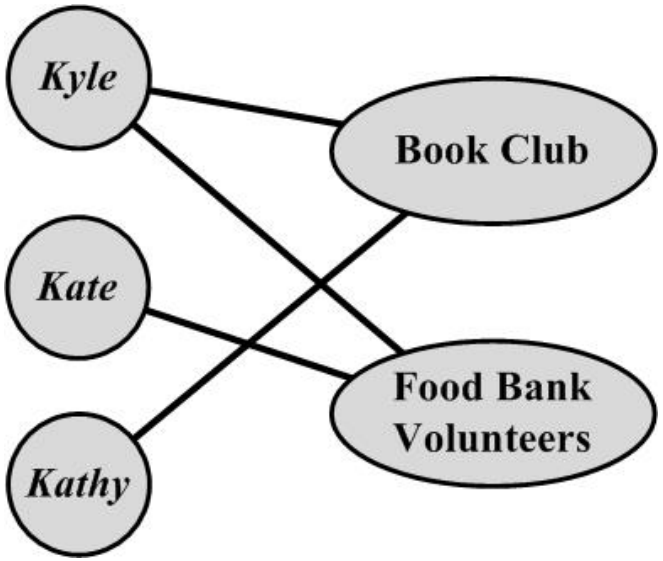
A bipartite graph $G(V, E)$ is a graph where the node set can be partitioned into two sets such that, for all edges, one end-point is in one set and the other end-point is in the other set.

$$\begin{cases} V = V_L \cup V_R, \\ V_L \cap V_R = \emptyset, \\ E \subset V_L \times V_R \end{cases}$$



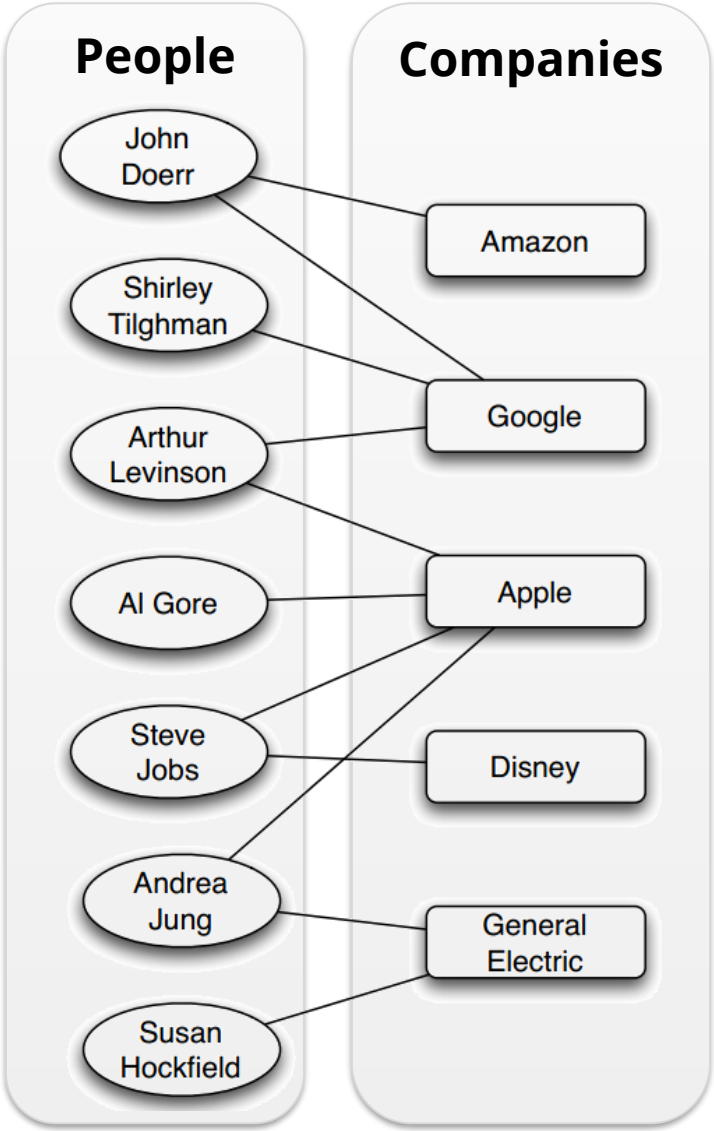
Affiliation Networks

An affiliation network is a bipartite graph. If an individual is associated with an affiliation, an edge connects the corresponding nodes.



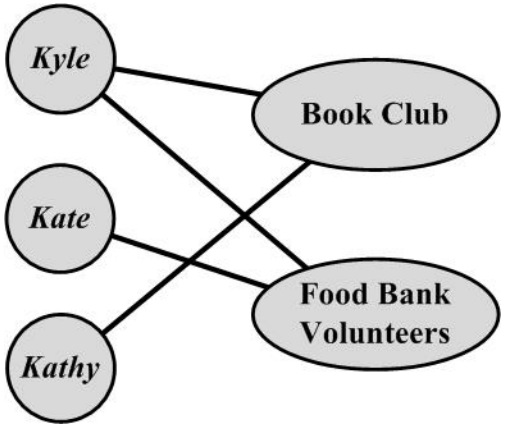
Affiliation Networks: Membership

Affiliation of people on corporate boards of directors



Bipartite Representation / one-mode Projections

- We can save some space by keeping membership matrix *X*



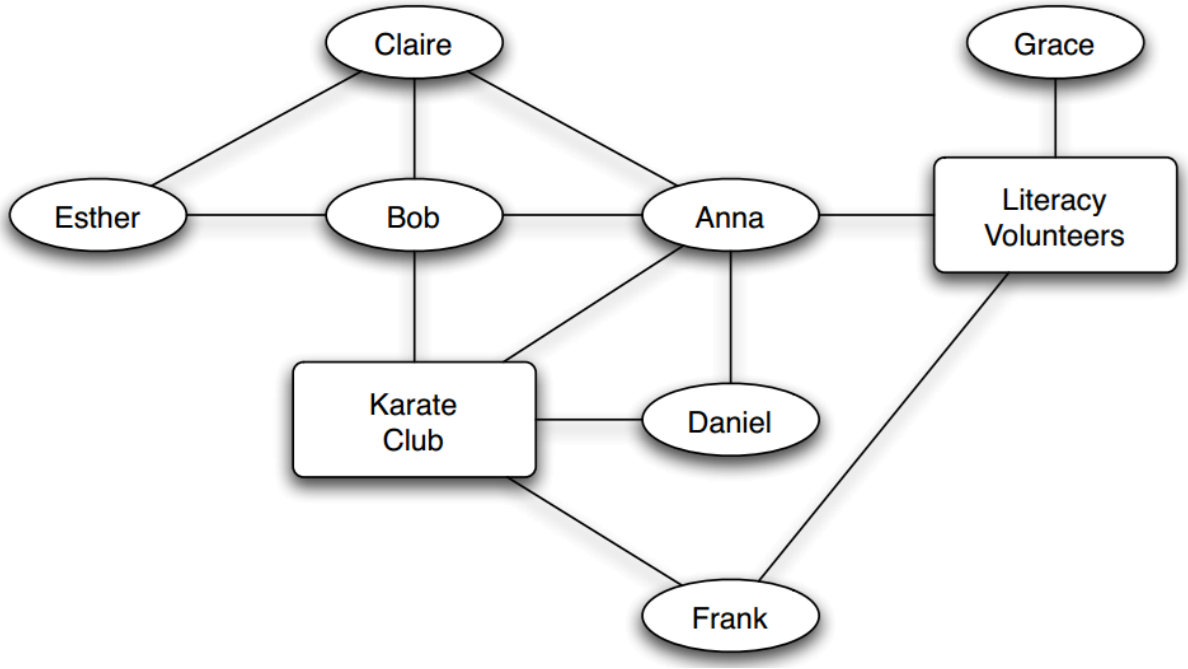
$$X = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- What is XX^T? *Similarity between users - [Bibliographic Coupling]*
- What is X^TX? *Similarity between groups - [Co-citation]*

Elements on the diagonal are number of groups
the user is a member of
OR
number of users in the group

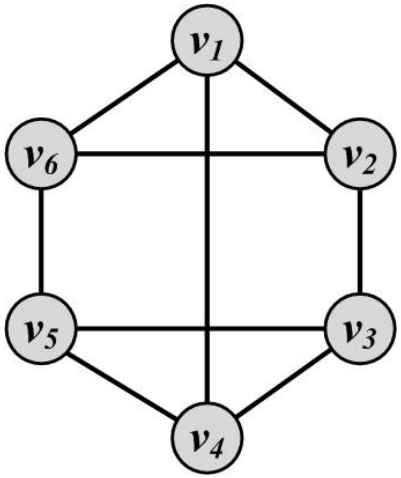
Social-Affiliation Network

Social-Affiliation network is a combination of a social network and an affiliation network



Regular Graphs

- A regular graph is one in which all nodes have the same degree
- Regular graphs can be connected or disconnected
- In a k-regular graph, all nodes have degree k
- Complete graphs are examples of regular graphs



Regular graph
With $k = 3$

Egocentric Networks

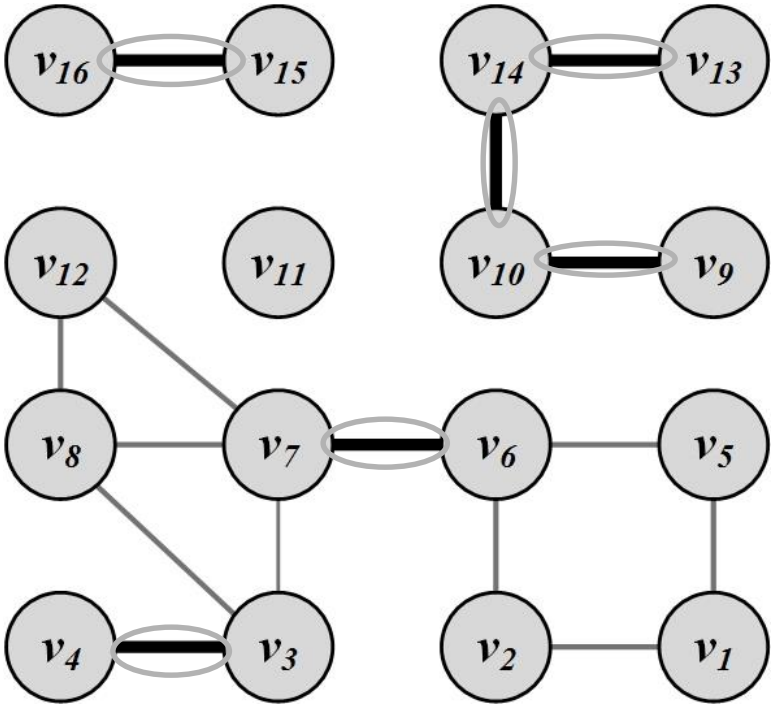
- **Egocentric** network: A focal actor (**ego**) and a set of **alters** who have ties with the ego

- Usually there are limitations for nodes to connect to other nodes or have relation with other nodes
  - **Example:** In a network of mothers and their children:
    - Each mother only holds mother-children relations with her own children

- Additional examples of egocentric networks are Teacher-Student or Husband-Wife

Bridges (cut-edges)

- Bridges are edges whose removal will increase the number of connected components



Graph Algorithms

Graph/Network Traversal Algorithms

Graph/Tree Traversal

- We are interested in surveying a social media site to computing the average age of its users
  - Start from one user;
  - Employ some traversal technique to reach her friends and then friends' friends, …
- The traversal technique guarantees that
  1. All users are visited; and
  2. No user is visited more than once.
- There are two main techniques:
  - **Depth-First Search (DFS)**
  - **Breadth-First Search (BFS)**

Depth-First Search (DFS)

- Depth-First Search (DFS) starts from a node v_i, selects one of its neighbors v_j from $N(v_i)$ and performs Depth-First Search on v_j before visiting other neighbors in $N(v_i)$

- The algorithm can be used both for trees and graphs
  - The algorithm can be implemented using a stack structure

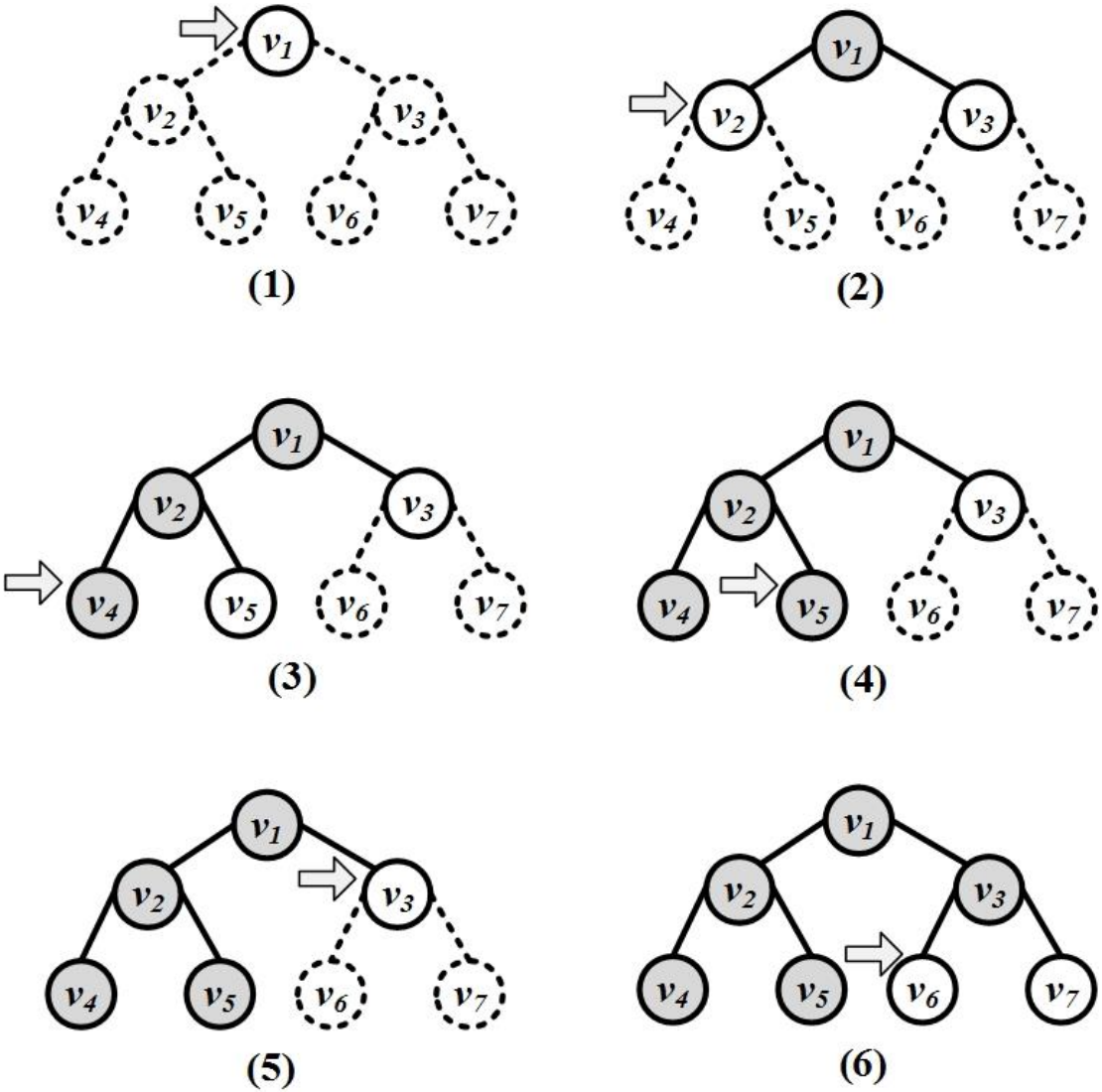
DFS Algorithm

Algorithm 2.2 Depth-First Search (DFS)

Require: Initial node v, graph/tree $G(V, E)$, stack S

```
1:  return An ordering on how nodes in G are visited
2:  Push v into S;
3:  visitOrder = 0;
4:  while S not empty do
5:      node = pop from S;
6:      if node not visited then
7:          visitOrder = visitOrder +1;
8:          Mark node as visited with order visitOrder; //or print node
9:          Push all neighbors/children of node into S;
10:     end if
11: end while
12: Return all nodes with their visit order.
```

Depth-First Search (DFS): An Example



Breadth-First Search (BFS)

- BFS starts from a node and visits all its immediate neighbors first, and then moves to the second level by traversing their neighbors.

- The algorithm can be used both for trees and graphs
  - The algorithm can be implemented using a queue structure

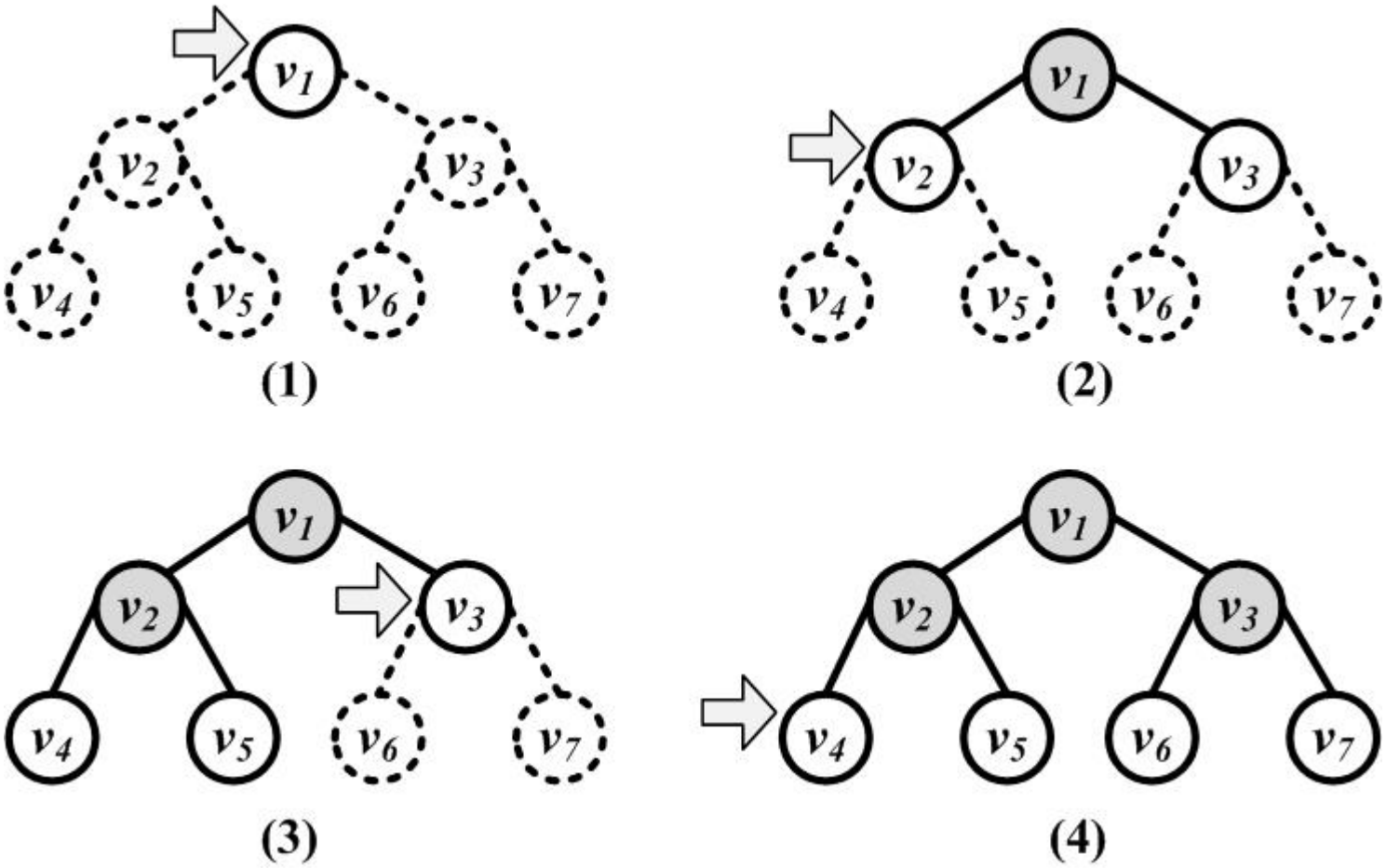
BFS Algorithm

Algorithm 2.3 Breadth-First Search (BFS)

Require: Initial node v, graph/tree $G(V, E)$, queue Q

```
1: return An ordering on how nodes are visited
2: Enqueue v into queue Q;
3: visitOrder = 0;
4: while Q not empty do
5:     node = dequeue from Q;
6:     if node not visited then
7:         visitOrder = visitOrder +1;
8:         Mark node as visited with order visitOrder; //or print node
9:         Enqueue all neighbors/children of node into Q;
10:    end if
11: end while
```

Breadth-First Search (BFS)



Finding Shortest Paths

Shortest Path

When a graph is connected, there is a chance that multiple paths exist between any pair of nodes

- In many scenarios, we want the shortest path between two nodes in a graph
  - How fast can I disseminate information on social media?

Dijkstra's Algorithm

- Designed for weighted graphs with non-negative edges
- It finds shortest paths that start from a provided node s to all other nodes
- It finds both shortest paths and their respective lengths

Dijkstra's Algorithm: Finding the shortest path

1. Initiation:
   - Assign zero to the source node and infinity to all other nodes
   - Mark all nodes as unvisited
   - Set the source node as current

2. For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances
   - If tentative distance is smaller than neighbor's distance, then Neighbor's distance = tentative distance

Tentative distance = current distance + edge weight

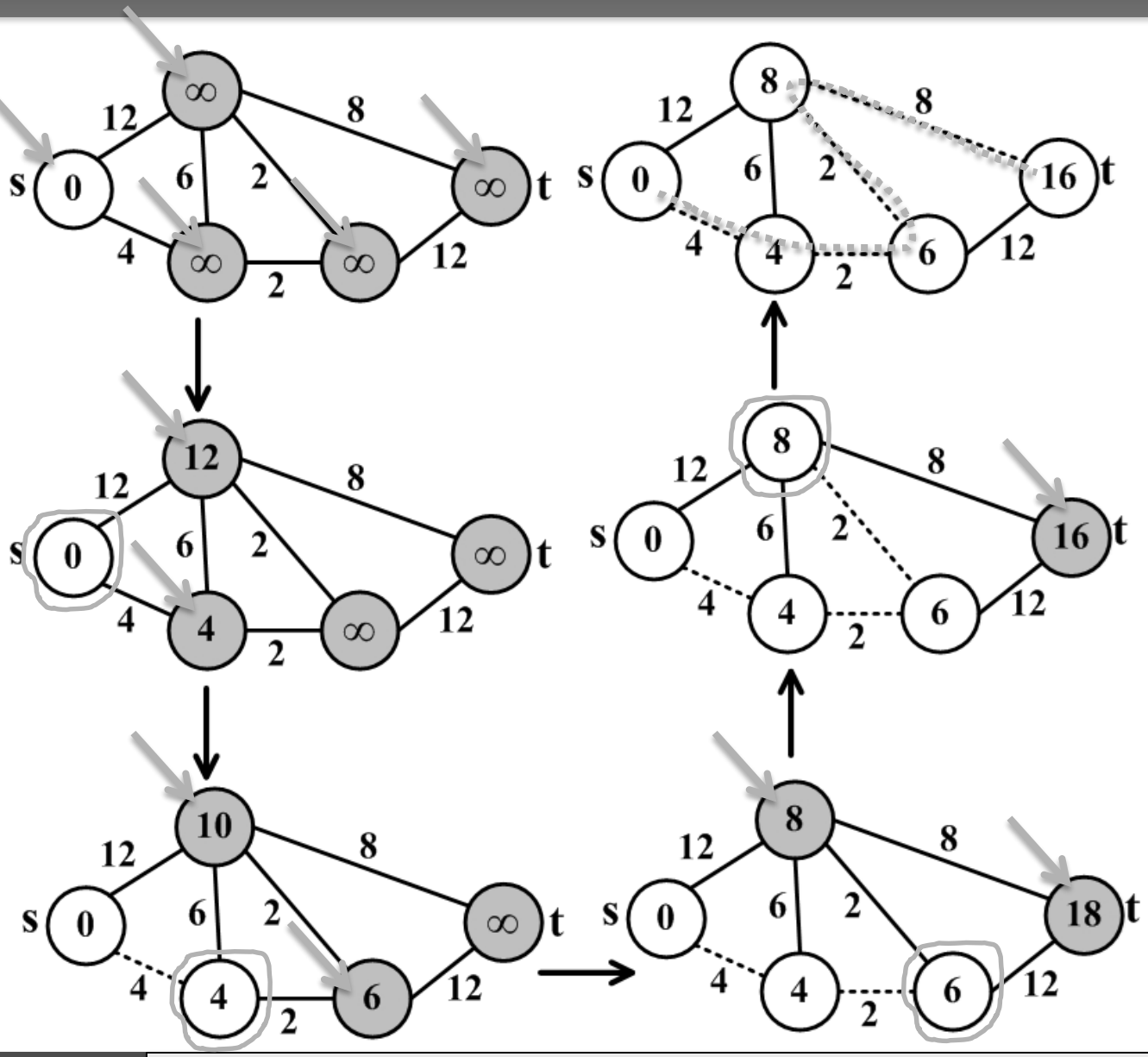
3. After considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*

4. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the *unvisited set* is infinity, then stop

A visited node will never be checked again and its distance recorded now is final and minimal

5. Set the unvisited node marked with the smallest tentative distance as the next "current node" and go to step 2

Dijkstra's Algorithm: Execution Example



Dijkstra's Algorithm: Notes

- Dijkstra's algorithm is source-dependent
  - Finds the shortest paths between the source node and all other nodes.
- To generate all-pair shortest paths,
  - We can run Dijsktra's algorithm n times, or
  - Use other algorithms such as Floyd-Warshall algorithm.
- If we want to compute the shortest path from source v to destination d,
  - we can stop the algorithm once the shortest path to the destination node has been determined

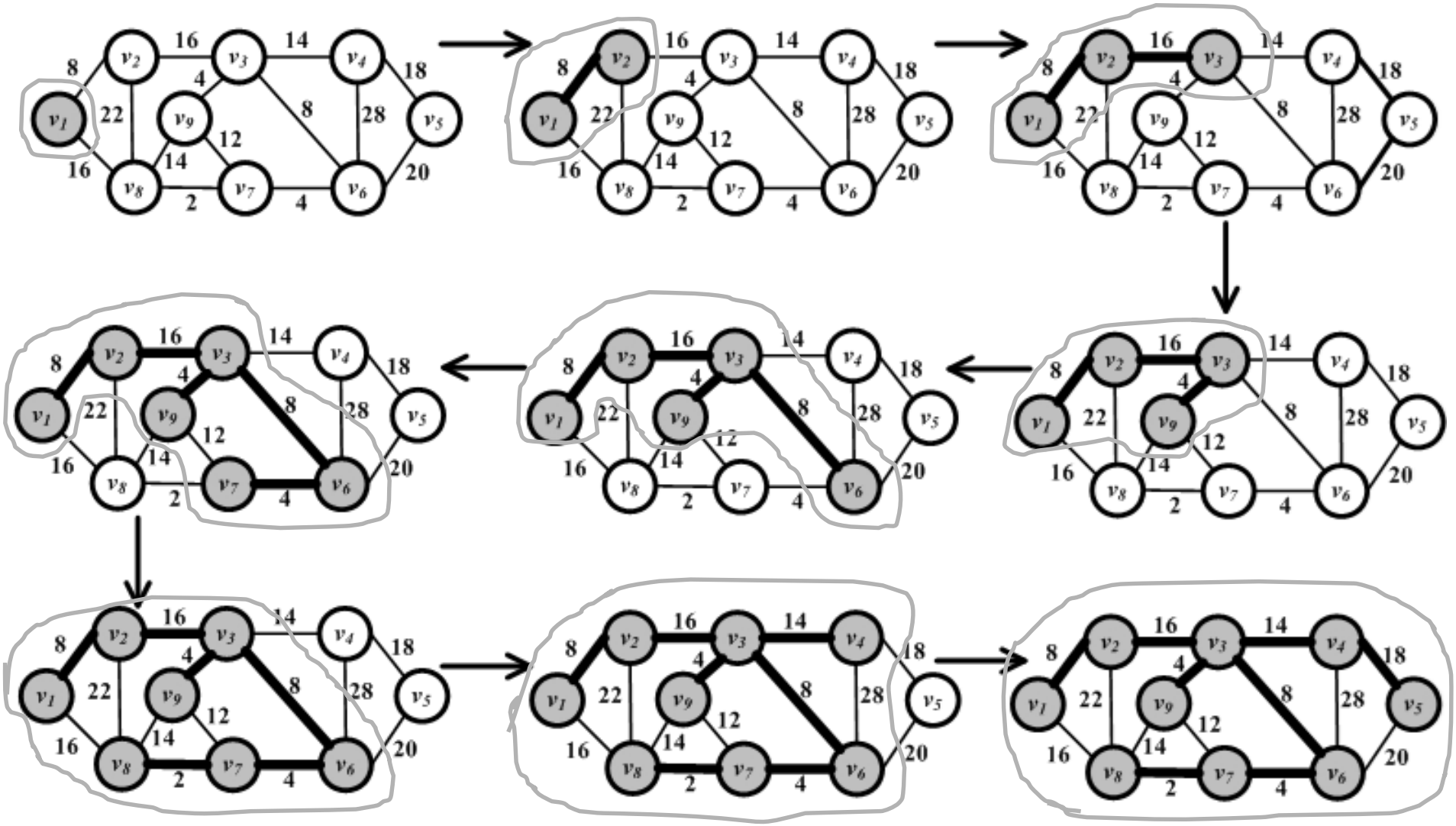
Finding Minimum Spanning Tree

Prim's Algorithm: Finding Minimum Spanning Tree

Finds MST in a weighted graph

1. Selecting a random node and add it to the MST
2. Grows the spanning tree by selecting edges which have one endpoint in the existing spanning tree and one endpoint among the nodes that are not selected yet. Among the possible edges, the one with the minimum weight is added to the set (along with its end-point).
3. This process is iterated until the graph is fully spanned

Prim's Algorithm Execution Example



Network Flow

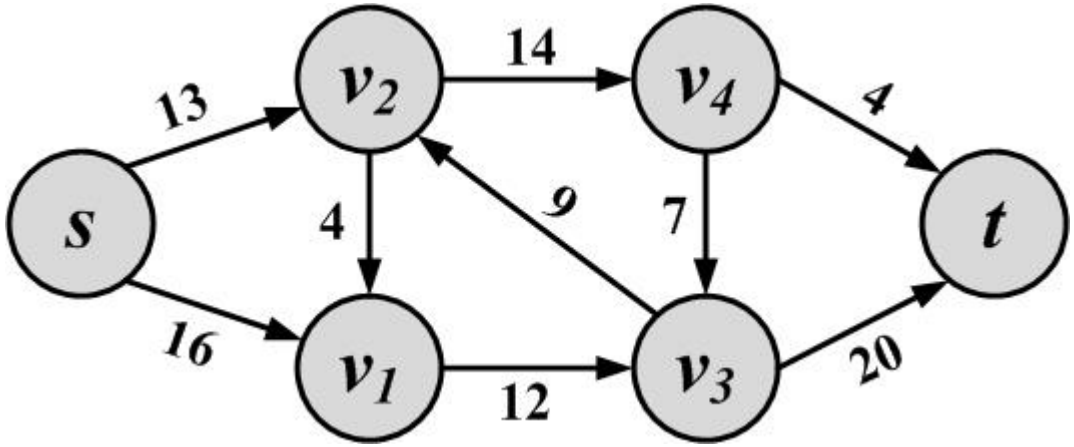
Network Flow

- Consider a network of pipes that connects an infinite water source to a water sink.
  - Given the capacity of these pipes, what is the maximum flow that can be sent from the source to the sink?

- Parallel in Social Media:
  - Users have daily cognitive/time limits (the capacity, here) of sending messages (the flow) to others,
  - What is the maximum number of messages the network should be prepared to handle at any time?

Flow Network

- A Flow network G(V,E,C) is a directed weighted graph, where we have the following:
  - $\forall\ (u,v) \in E, c(u,v) \geq 0$ defines the edge capacity.
  - When $(u,v) \in E, (v,u) \notin E$ (opposite flow is impossible)
  - s defines the source node and t defines the sink node. An infinite supply of flow is connected to the source.



Flow

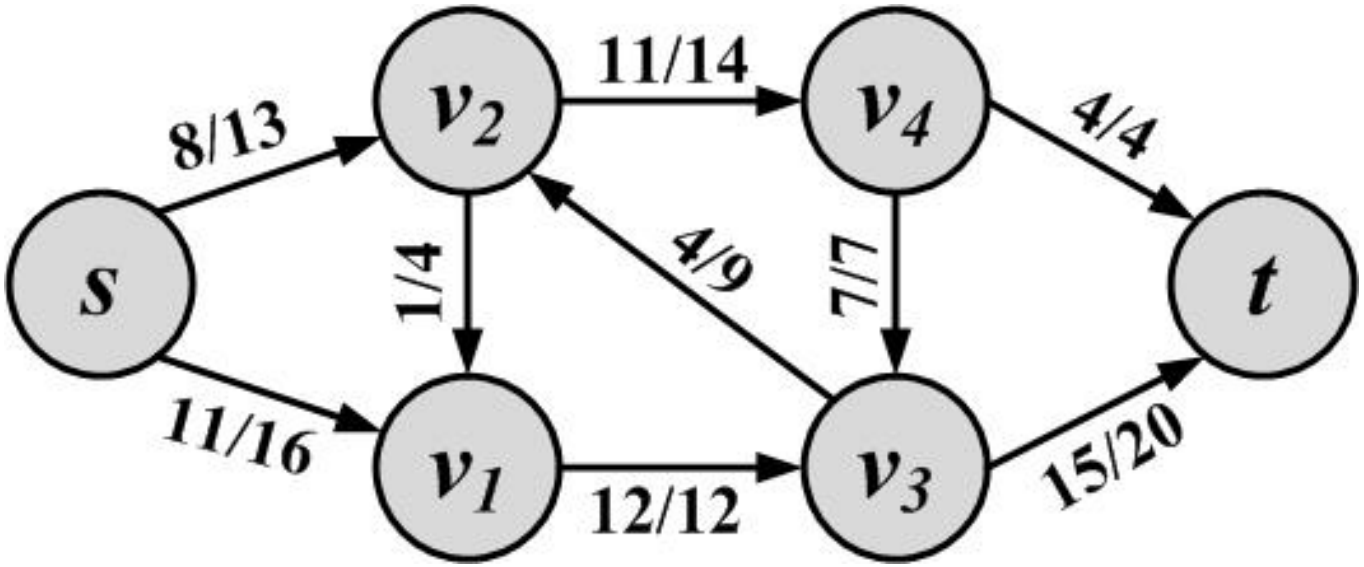
- Given edges with certain capacities, we can fill these edges with the flow up to their capacities (*capacity constraint*)
- The flow that enters any node other than source s and sink t is equal to the flow that exits it so that no flow is lost *(flow conservation constraint*)

- $\forall (u,v) \in E, f(u,v) \geq 0$ defines the flow passing through the edge.
- $\forall (u,v) \in E,\ 0 \leq f(u,v) \leq c(u,v)$ ***(capacity constraint)***
- $\forall v \in V - \{s,t\}, \sum_{k:(k,v)\in E} f(k,v) = \sum_{l:(v,l)\in E} f(v,l)$

(flow conservation constraint)

A Sample Flow Network

- Commonly, to visualize an edge with capacity c and flow f , we use the notation f/c.

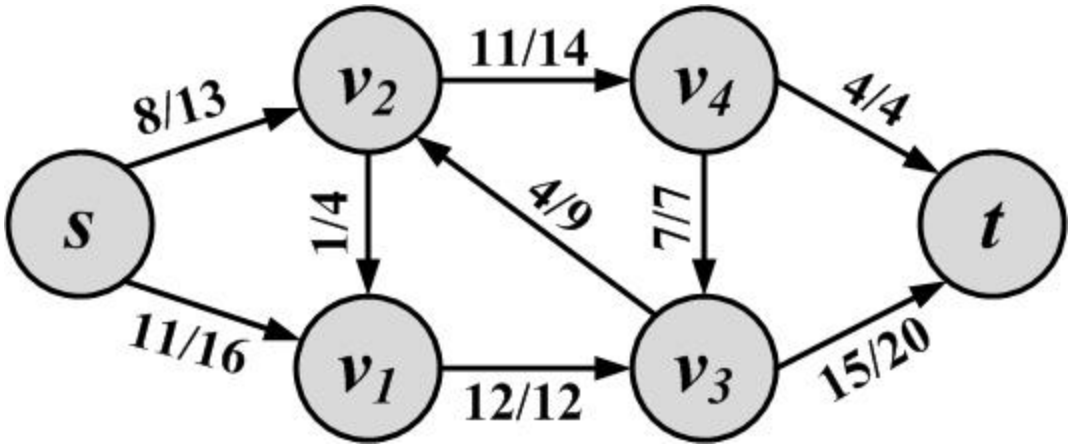


Flow Quantity

- The flow quantity (or value of the flow) in any network is the amount of
  - Outgoing flow from the source minus the incoming flow to the source.
  - Alternatively, one can compute this value by subtracting the outgoing flow from the sink from its incoming value

$$flow = \sum_v f(s,v) - \sum_v f(v,s) = \sum_v f(v,t) - \sum_v f(t,v)$$

What is the flow value?



- **19**
  - **11+8** from **s**, or
  - **4+15** to **t**

Ford-Fulkerson Algorithm

- Find a path from source to sink such that there is unused capacity for all edges in the path.
- Use that capacity (the minimum capacity unused among all edges on the path) to increase the flow.

- Iterate until no other path is available.

Residual Network

- Given a flow network $G(V, E, C)$, we define another network $G(V, E_R, C_R)$
- This network defines how much capacity remains in the original network.
- The residual network has an edge between nodes u and v if and only if either (u, v) or (v, u) exists in the original graph.
  - If one of these two exists in the original network, we would have **two** edges in the residual network: one from (u, v) and one from (v, u).

Intuition

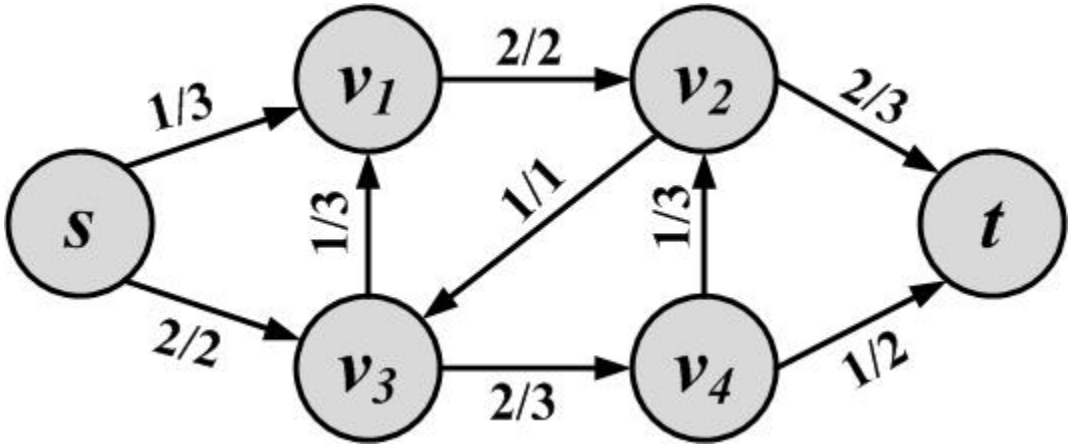
- When there is no flow going through an edge in the original network, a flow of as much as the capacity of the edge remains in the residual.

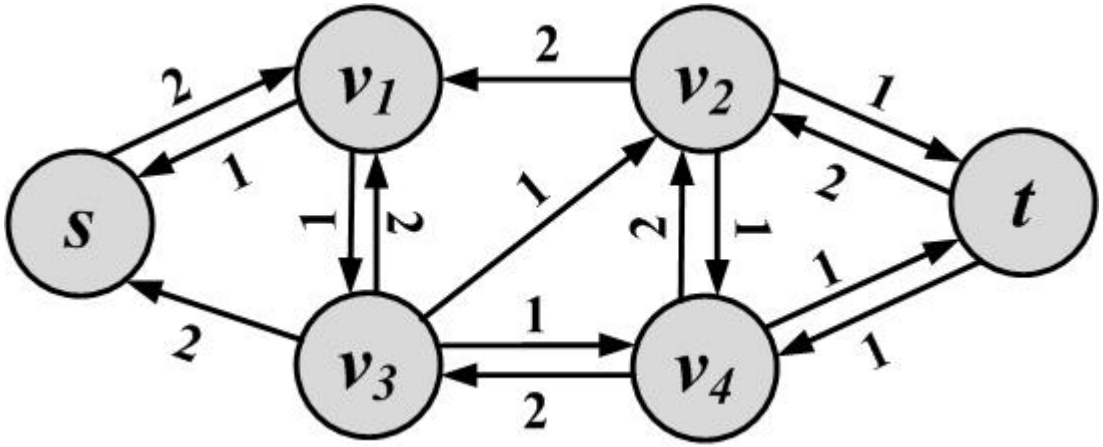
- In the residual network, one has the ability to send flow in the opposite direction to cancel some amount of flow in the original network.

$$c_R(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (u,v) \notin E \end{cases}$$

Residual Network (Example)

- Edges that have zero capacity in the residual are not shown





Augmentation / Augmenting Paths

1. In the residual graph, when edges are in the same direction as the original graph,
   - Their capacity shows how much **more** flow can be pushed along that edge in the **original** graph.

2. When edges are in the opposite direction,
   - their capacities show how much flow can be **pushed back** on the **original graph edge**.

- By finding a flow in the residual, we can **augment** the flow in the original graph.

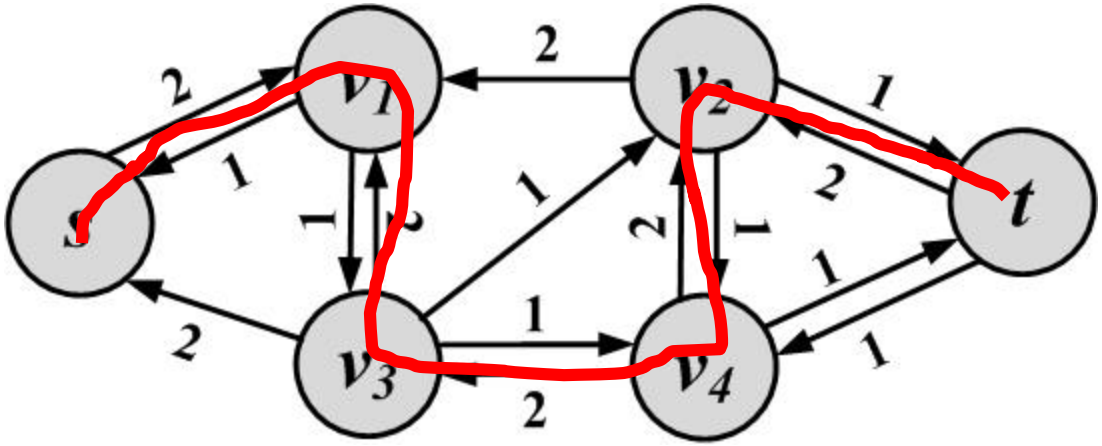
Augmentation / Augmenting Paths

- Any simple path from s to t in the residual graph is an *augmenting path*.
  - **All capacities in the residual are positive,**
    - These paths can augment flows in the original, thus increasing the flow.
  - The amount of flow that can be pushed along this path is equal to the **minimum capacity** along the path
    - The edge with the minimum capacity limits the amount of flow being pushed
    - We call the edge the ***Weak link***

How do we augment?

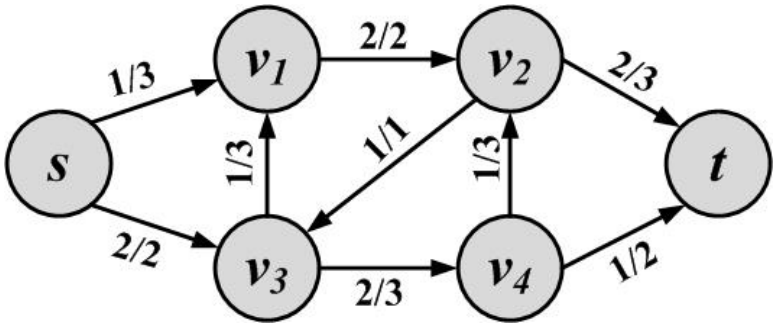
- Given flow f (u, v) in the original graph and flow $f_R(u, v)$ and $f_R(v, u)$ in the residual graph, we can augment the flow as follows:

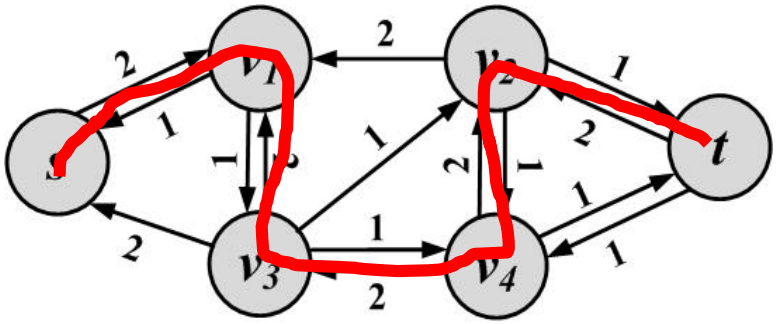
$$f_{augmented}(u, v) = f(u, v) + f_R(u, v) - f_R(v, u)$$

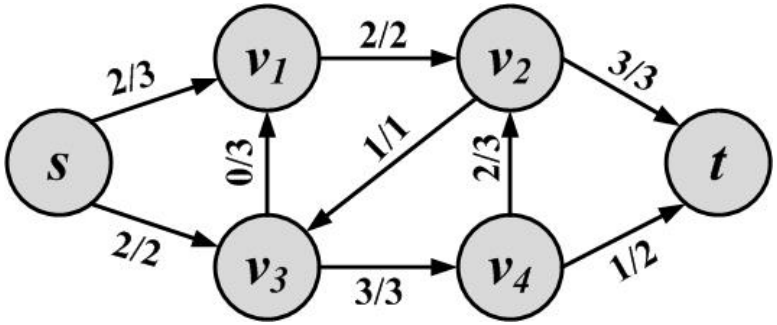


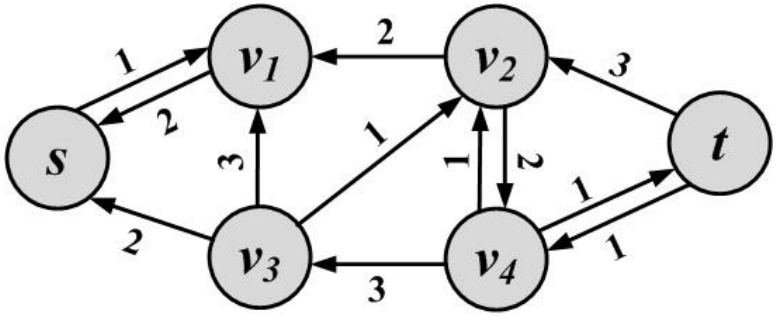
Flow Quantity: 1

Augmenting


s
v_1
v_2
v_3
v_4
t
1/3
2/2
2/2
2/3
1/3
1/1
1/3
2/3
1/2


s
v_1
v_2
v_3
v_4
t


s
v_1
v_2
v_3
v_4
t
2/3
2/2
2/2
3/3
0/3
1/1
2/3
2/2
3/3
1/2


s
v_1
v_2
v_3
v_4
t

The Ford-Fulkerson Algorithm

Algorithm 2.6 Ford-Fulkerson Algorithm

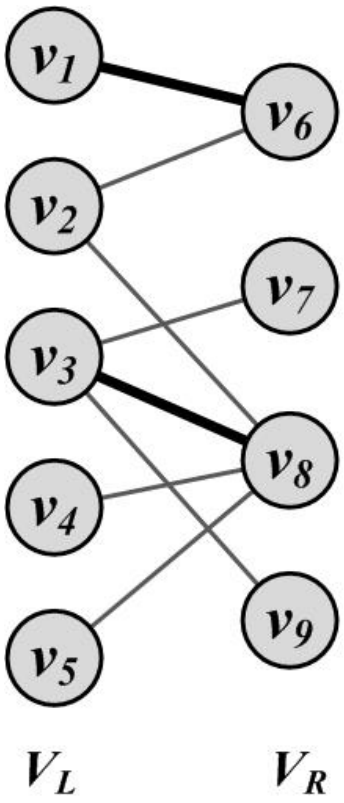
Require: Connected weighted graph $G(V, E, W)$, Source s, Sink t

1: **return** A Maximum flow graph
2: $\forall (u, v) \in E$, $f(u, v) = 0$
3: **while** there exists an augmenting path p in the residual graph G_R **do**
4: Augment flows by p
5: **end while**
6: Return flow value and flow graph;

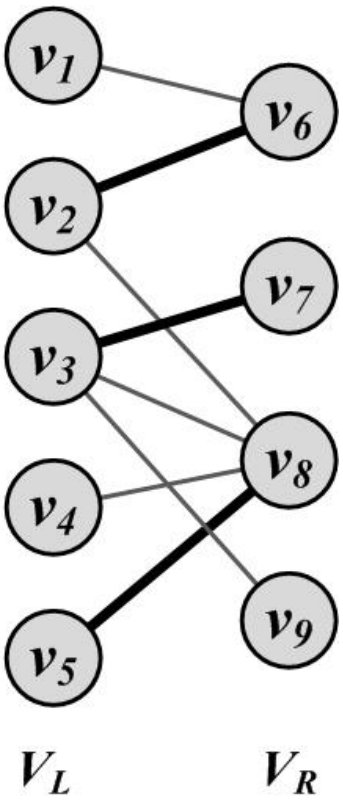
Maximum Bipartite Matching

Example

- Given n products and m users
  - Some users are only interested in certain products
  - We have only one copy of each product.
  - Can be represented as a bipartite graph
  - Find the maximum number of products that can be bought by users
    - No two edges selected share a node



Matching

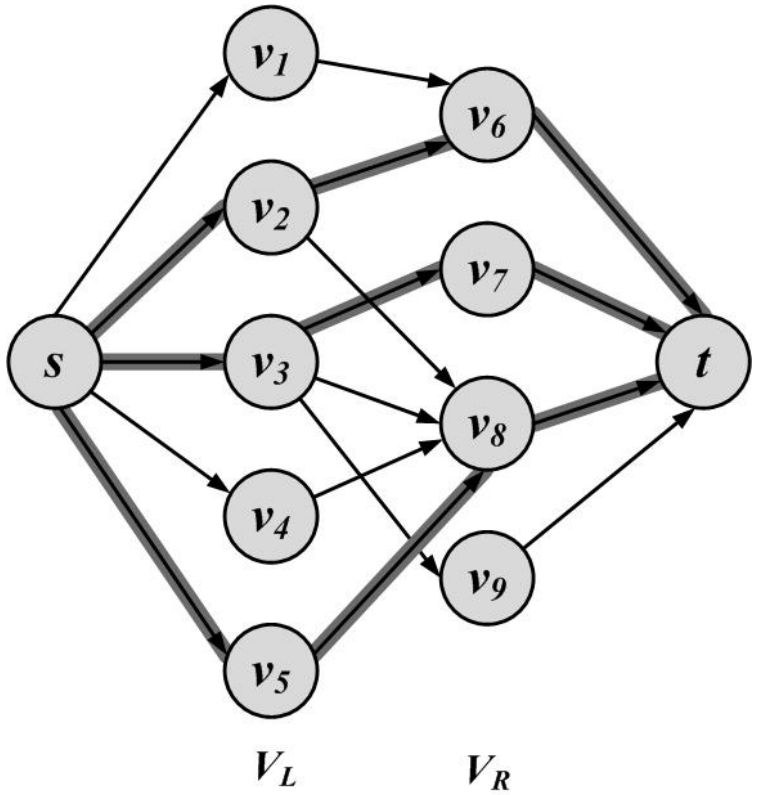


Maximum Matching

Matching Solved with Max-Flow

- Create a flow graph $G(V', E', C)$ from our bipartite graph $G(V, E)$

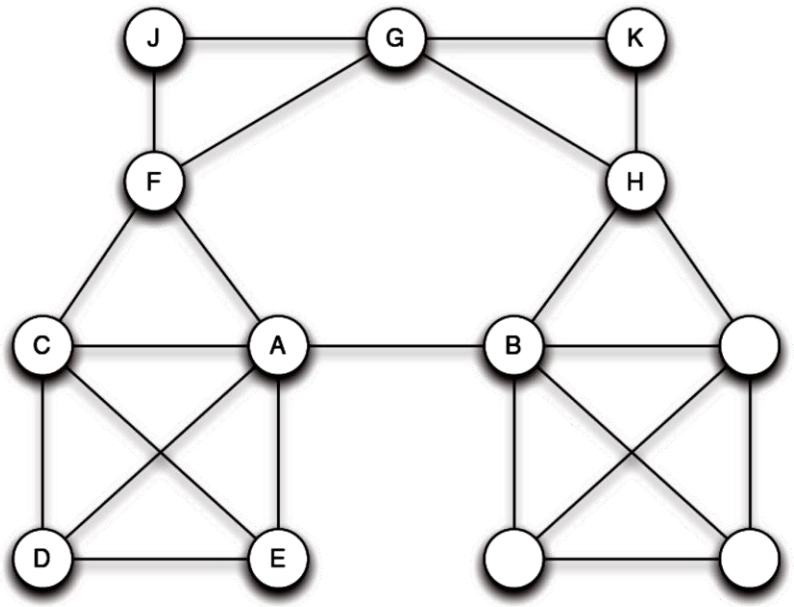
1. Set $V' = V \cup \{s\} \cup \{t\}$
2. Connect all nodes in V_L to s and all nodes in V_R to t
3. Set $c(u, v) = 1$, for all edges in E'



Bridges, Weak Ties, and Bridge Detection

Bridge and a Local Bridge

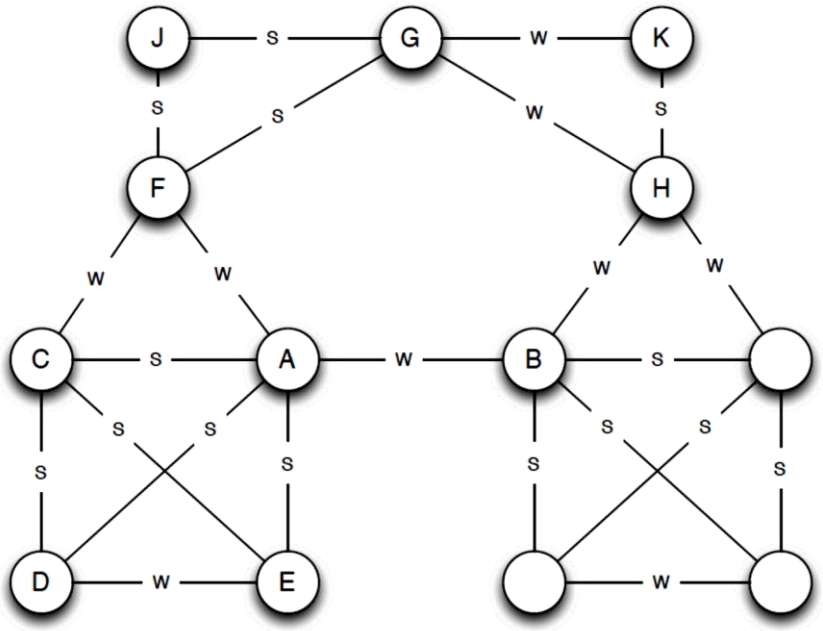
- **Bridge:** Bridges are edges whose removal will increase the number of connected components
  - Bridges are extremely rare in real-world social networks.
- **Local Bridge:** when the endpoints have no friend in common
  - the removal increases the length of shortest path to more than 2
  - **Span of the local bridge:** How much the distance between the endpoints would become if the edge is removed
    - Large span is desirable to find communities



Source: Easley and Kleinberg – Networks, Crowds, and Markets

Strength of Ties

- **Assume** that you can divide connections into two categories:
  - **Strong tie (S)**:
    - friends
  - **Weak ties (W)**:
    - acquaintances

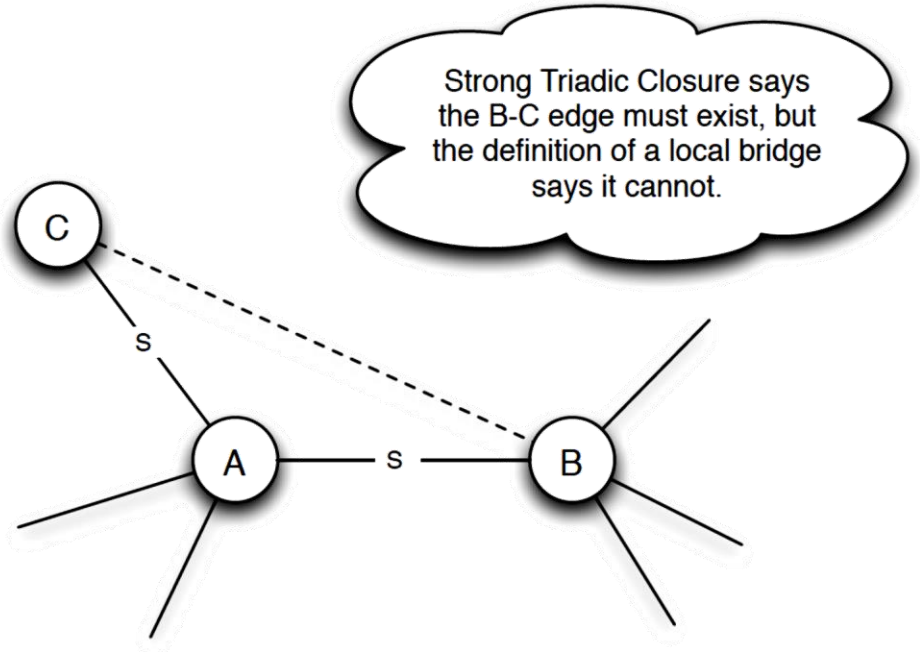


- **Strong Triadic Closure:**
  - Consider a node u that has two strong ties to nodes v and w
  - If there is no edge between v and w (weak or strong tie) then u does not exhibit a strong triadic closure

Connection between Bridges and Tie Strength

If a node exhibits Strong Triadic Closure and has at least two strong ties, then if it part of a local bridge, that bridge must be a weak tie

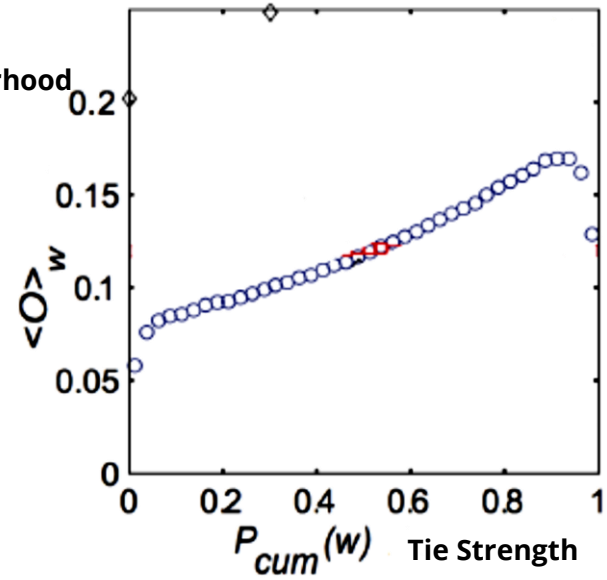
Why?



Source: Easley and Kleinberg – Networks, Crowds, and Markets

Generalizing to Real-World Networks

- Consider a cell-phone network
  - We have an edge if both end points call each other
  - **Tie Strength**: it does not have to be weak/strong
    - For (u, v), the number of minutes spent u and v spent talking to each other on the phone
  - **Local Bridge**: can be generalized using *neighborhood overlap:*



The numerator is called embeddedness of an edge

$$\frac{\text{number of nodes who are neighbors of both } u \text{ and } v}{\text{number of nodes who are neighbors of } \textit{at least one} \text{ of } u \text{ or } v}$$

When numerator is zero we have a local bridge

Bridge Detection

Algorithm 2.7 Bridge Detection Algorithm

Require: Connected graph $G(V, E)$

```
1: return Bridge Edges
2: bridgeSet = {}
3: for e(u, v) ∈ E do
4:     G' = Remove e from G
5:     Disconnected = False;
6:     if BFS in G' starting at u does not visit v then
7:         Disconnected = True;
8:     end if
9:     if Disconnected then
10:        bridgeSet = bridgeSet ∪ {e}
11:    end if
12: end for
13: Return bridgeSet
```