# Networks & Graphs part 3

CS 579 Online Social Network Analysis

Dr. Cindy Hood
9/4/25

# Homework Assignments

- HW #1 Assigned last week
  - Due by midnight Sept 3
- HW #2 assigned today
  - Due by midnight Monday 9/15

# Exams and Final Project Poster Presentation

- Exam 1 – Oct 9 in class
- Exam 2 – Dec 2 in class

- Final Project Poster Session – Dec 4 in class

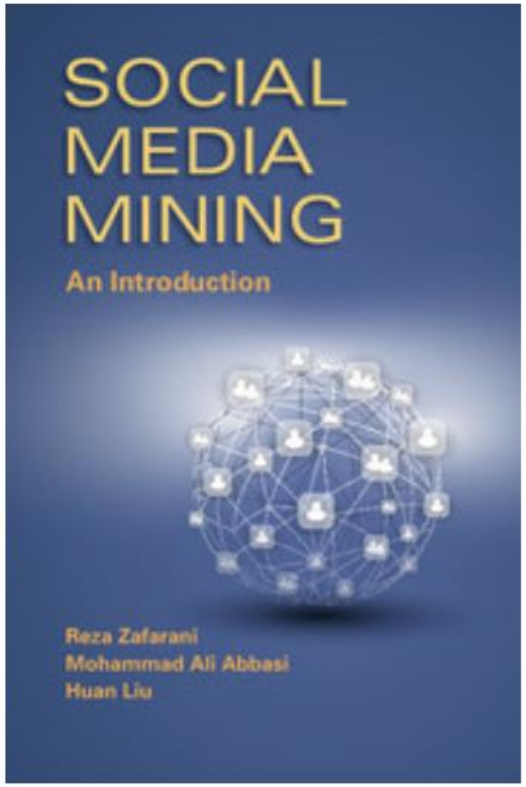
- Online students (sections 2 and 3) will have remote options

# Teaching Assistants

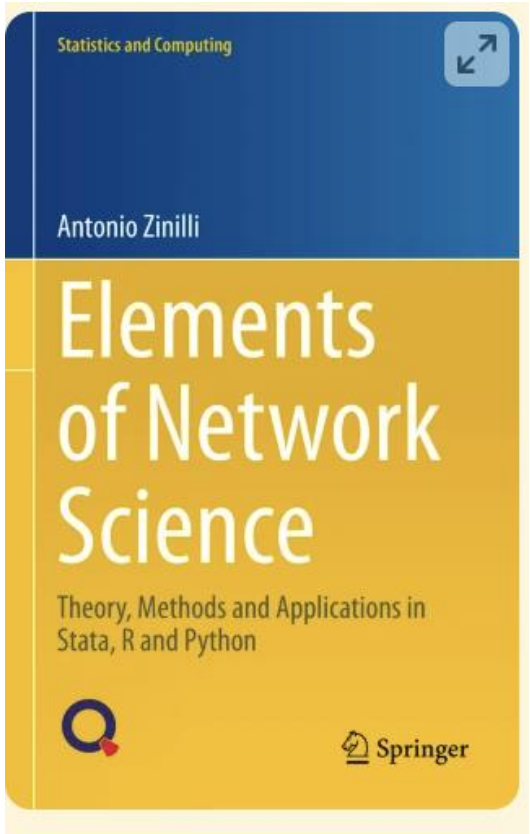
- Siva Krishna Golla
  - sgolla2@hawk.illinoistech.edu
  - Mondays 2-3pm on zoom
- Khush Dhiren Patel
  - kpatel210@hawk.illinoistech.edu
  - Wednesdays 11-12 online
- Aswith Sama
  - asama@hawk.illinoistech.edu
  - Thursdays 3-4pm on zoom

- **Not yet officially working, waiting for authorization (US govt)**

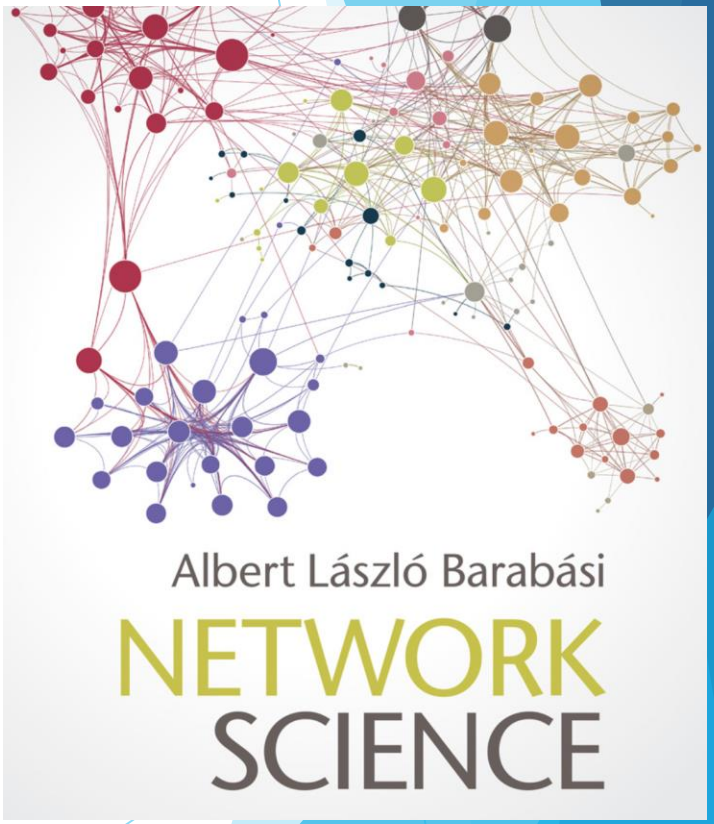
# References




http://www.socialmediamining.info




https://link.springer.com/book/10.1007/978-3-031-84712-7



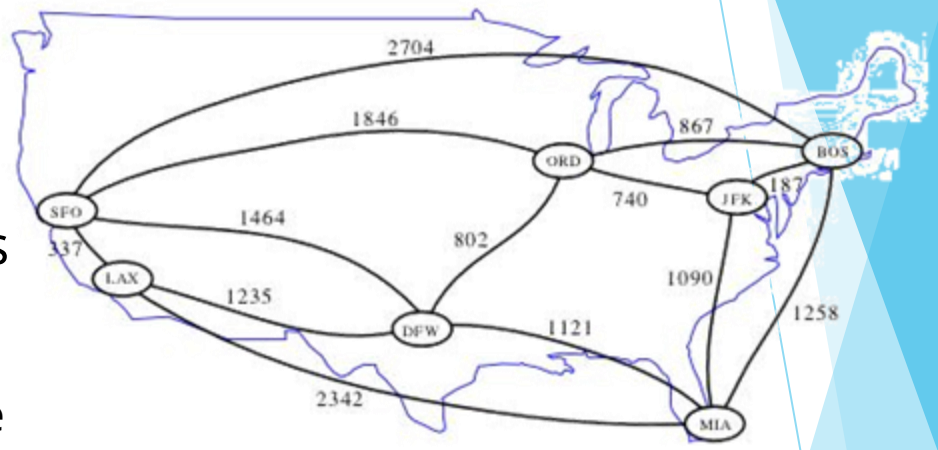

http://networksciencebook.com

# Types of Graphs

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

# Weighted Graph

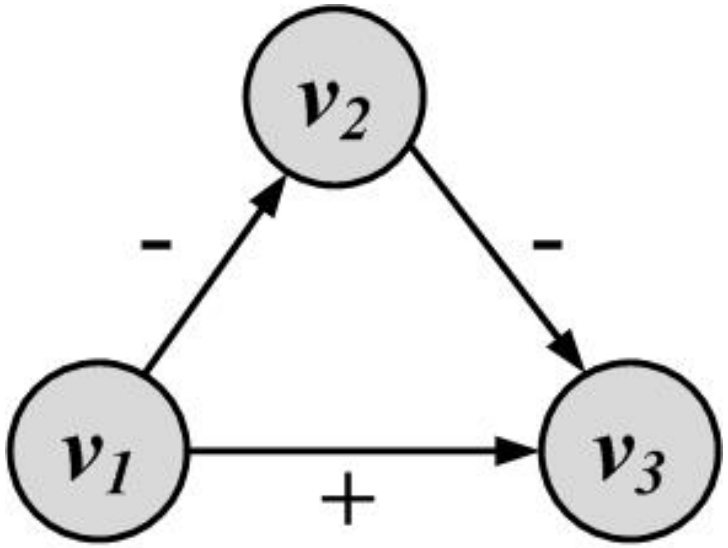
- A weighted graph $\boldsymbol{G(V, E, W)}$ is one where edges are associated with weights
  - For example, a graph could represent a map where nodes are airports and edges are routes between them
    - The weight associated with each edge could represent the distance between the corresponding cities




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

# Signed Graph

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

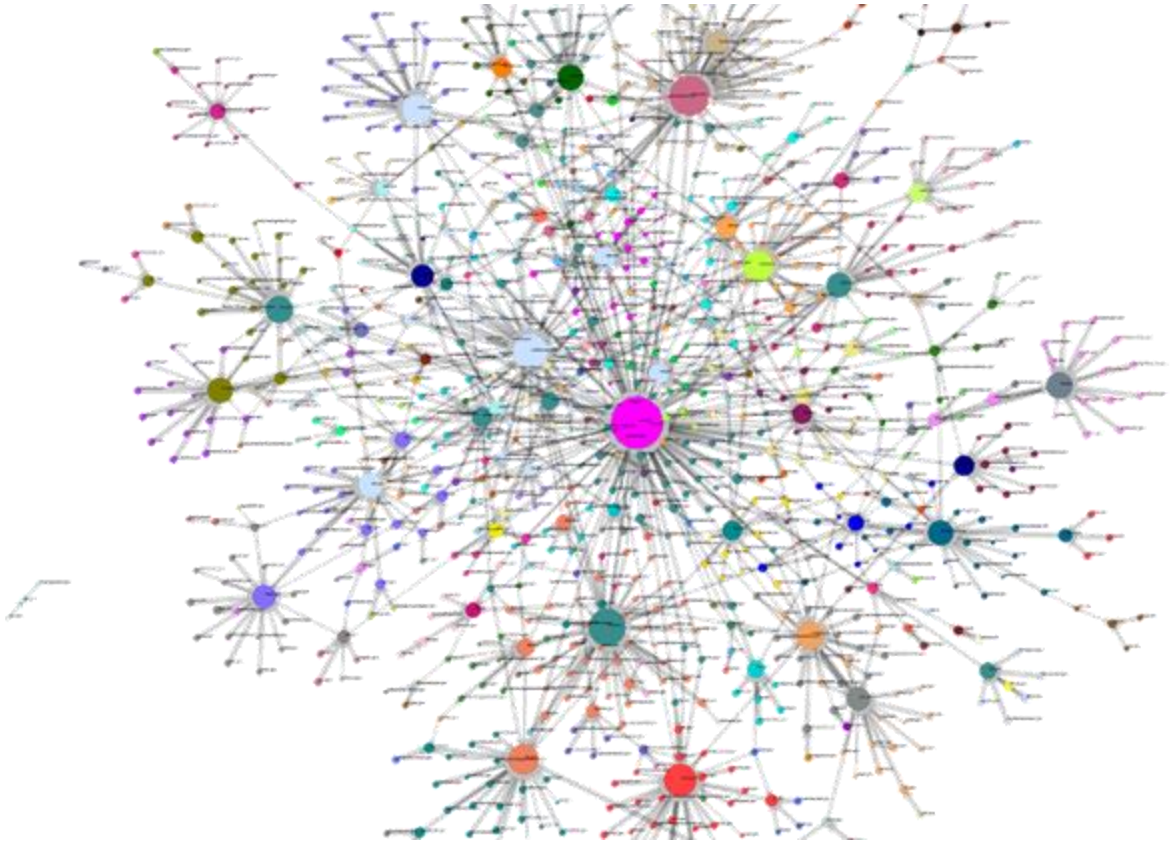



- It is used to represent friends or foes
- It is also used to represent social status

# Webgraph

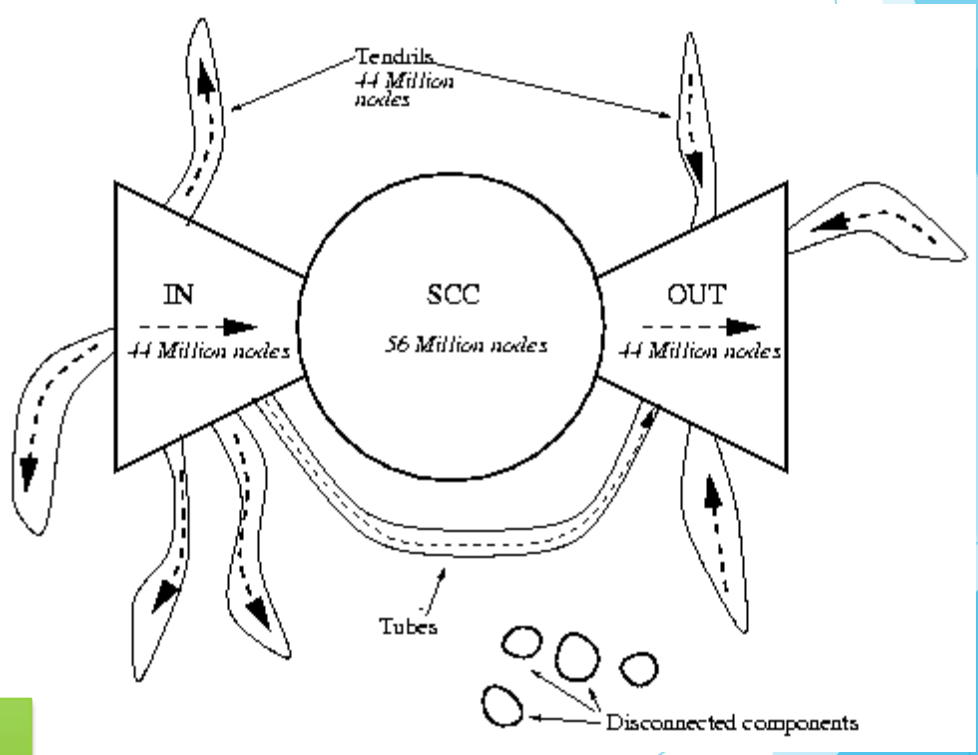
- A webgraph is a way of representing how internet sites are connected on the web
- In general, a web graph is a directed multigraph
- Nodes represent sites and edges represent links between sites.
- Two sites can have multiple links pointing to each other and can have loops (links pointing to themselves)

# Webgraph



**Government Agencies**

**Bow-tie structure**



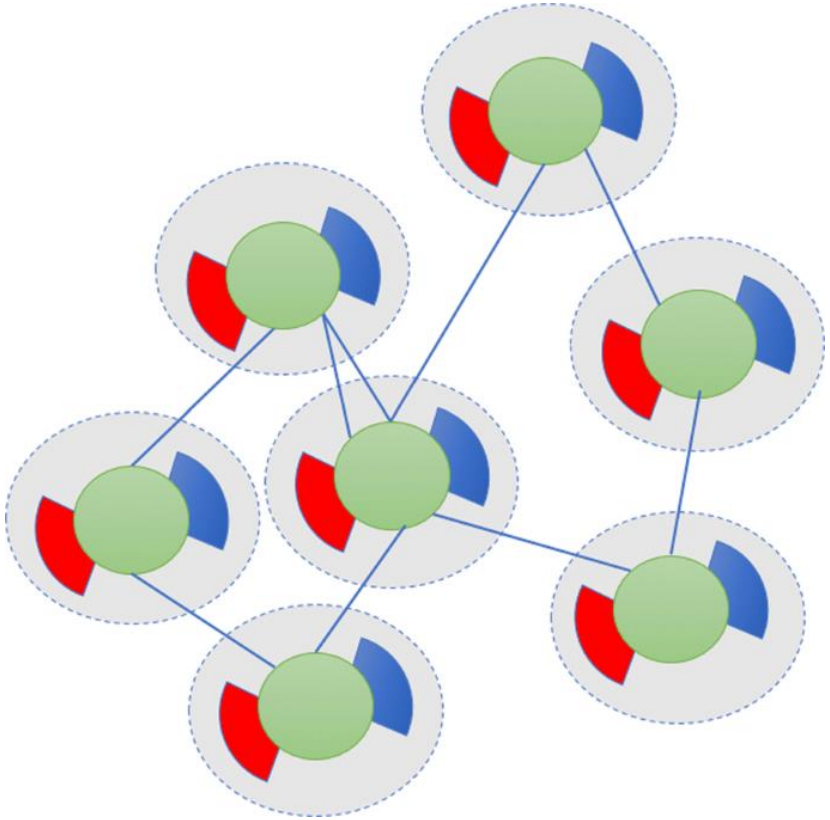

SCC – Strongly Connected Component

Broder et al (2000) –
200 million pages, 1.5 billion links

# Does the Bow-Tie Structure still hold for the web?

- What has changed since late 1990's?

# Web is now a set of weakly connected bow-ties



https://blogs.cornell.edu/info2040/2021/11/05/is-the-web-structure-still-bow-tie/

"Local Bow-tie Structure of the Web." Fujita et al. *Applied Network Science*. 2019.

"The Mobile Web Is Structurally Different." Jindal et al. *IEEE INFOCOM Workshops*. 2008

# 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

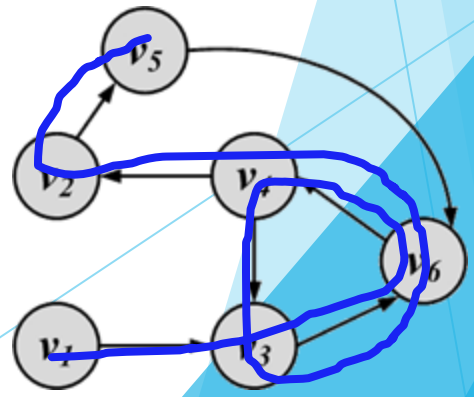
- **Open walk**: A walk does not end where it starts
- **Closed walk**: A walk returns to where it starts

- 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:

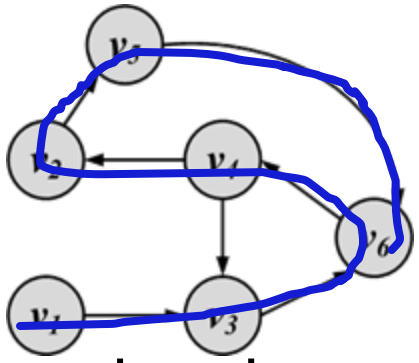
  the number of visited edges

Length of walk= 8



# Trail

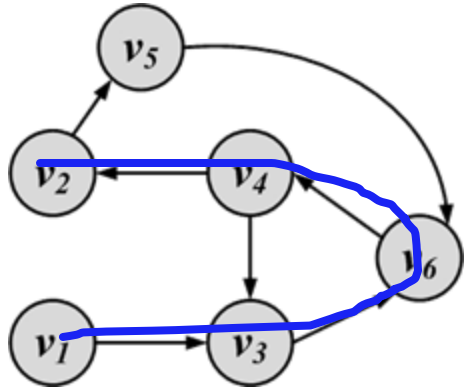
- A trail is a walk where **no edge** **is visited more than once** and all walk edges are distinct



- 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

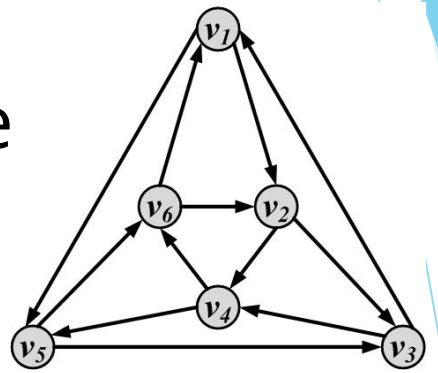



Length of path= 4

# Examples

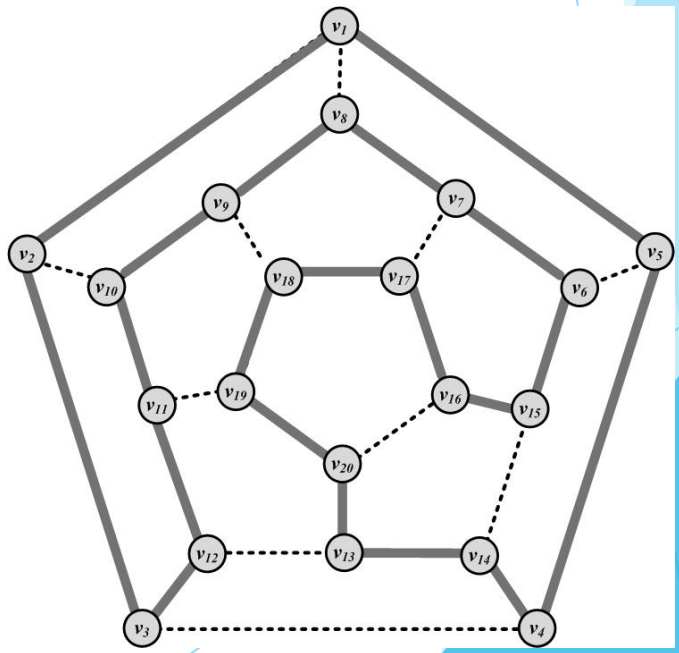
**Eulerian** Tour

- 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

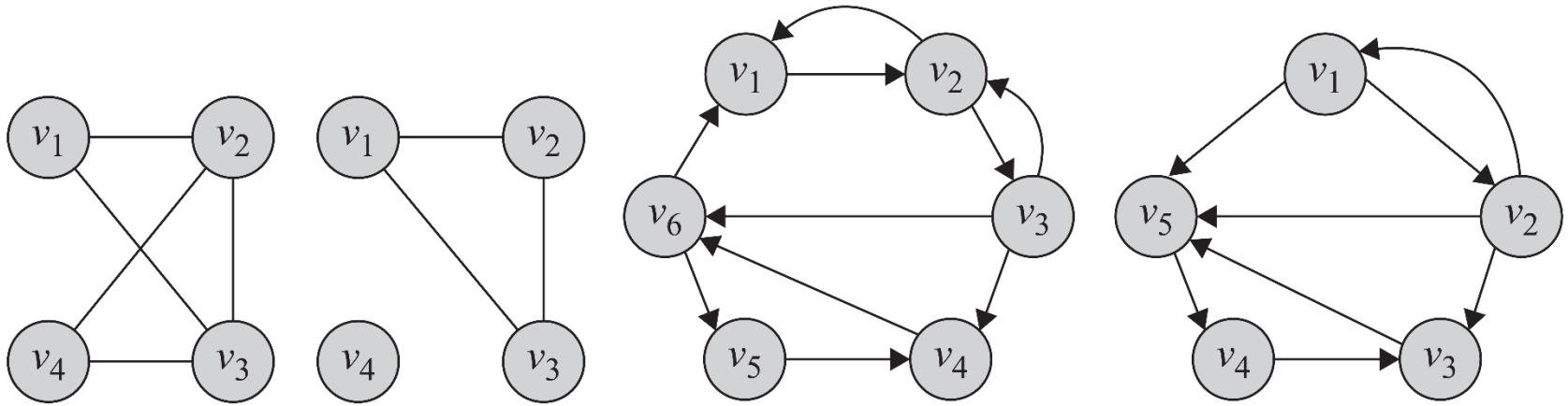
# Connectivity

- **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

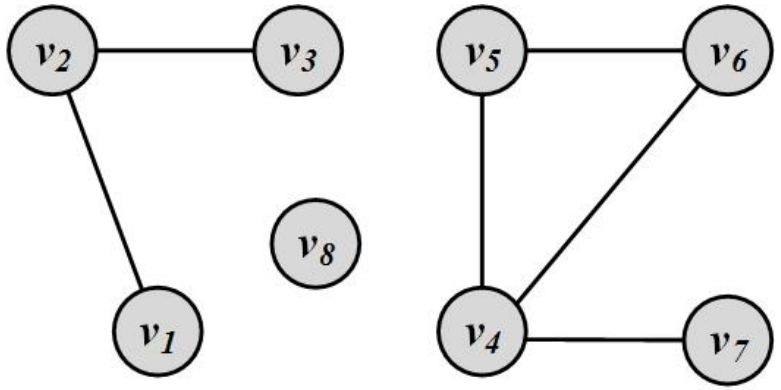
# Component

- A **component** in an undirected graph is a **connected subgraph**, i.e., there is a path between every pair of nodes inside the component

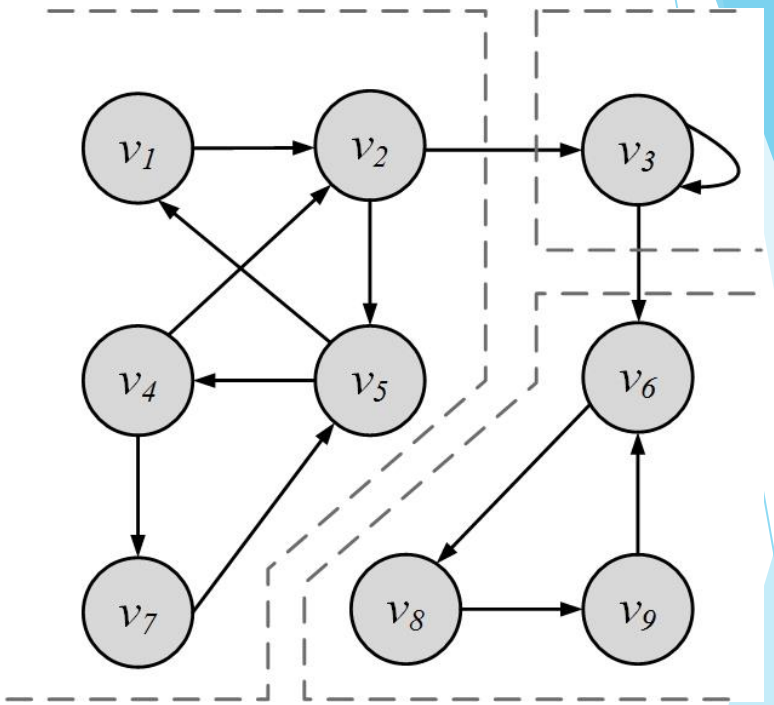
- In directed graphs, we have a **strongly connected** components when there is a path from $u$ to $v$ and one from $v$ to $u$ for every pair of nodes $u$ and $v$.

- 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 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?

# What is the diameter of the web

- 19 (according to a Nature paper published in 1999)
  - https://www.researchgate.net/publication/260282976_Diameter_of_the_World-Wide_Web
    - Estimate at least 800,000,000 web pages
- Now?

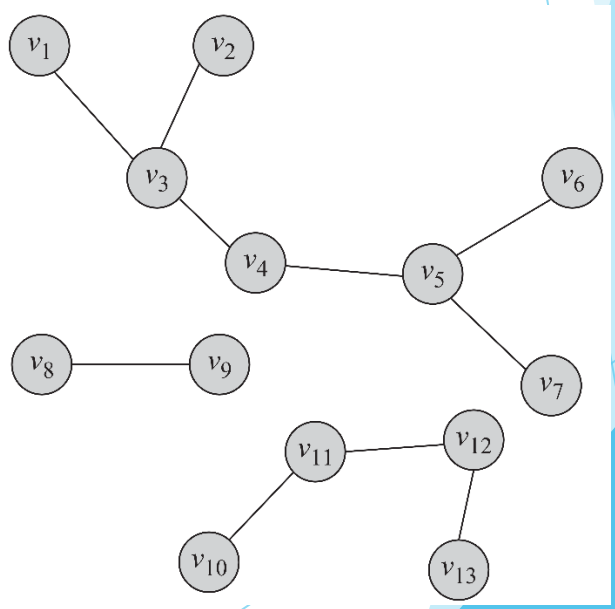
# What is the size of the web?

- Estimate https://www.worldwidewebsize.com
  - Methodology for estimate: https://www.dekunder.nl/Media/10.1007_s11192-016-1863-z.pdf

# 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**



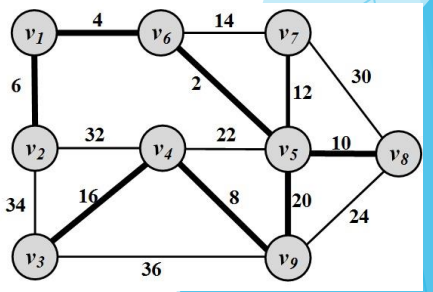

A forest containing 3 trees

# 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

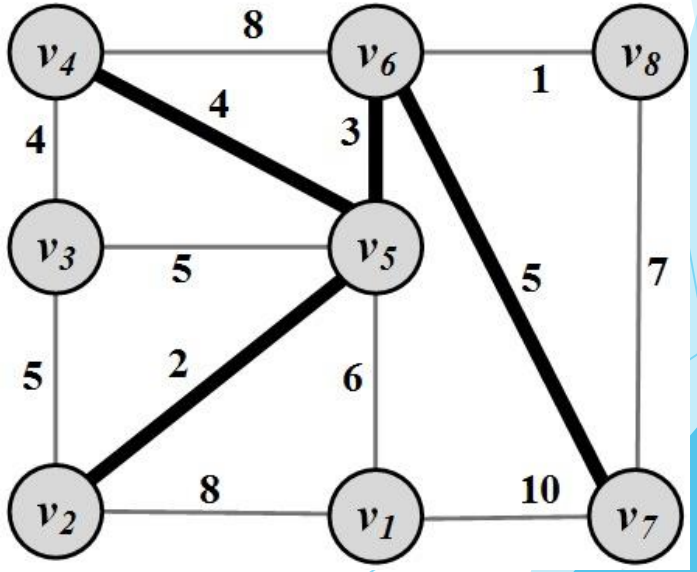
  **minimum spanning tree (MST)**



# 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

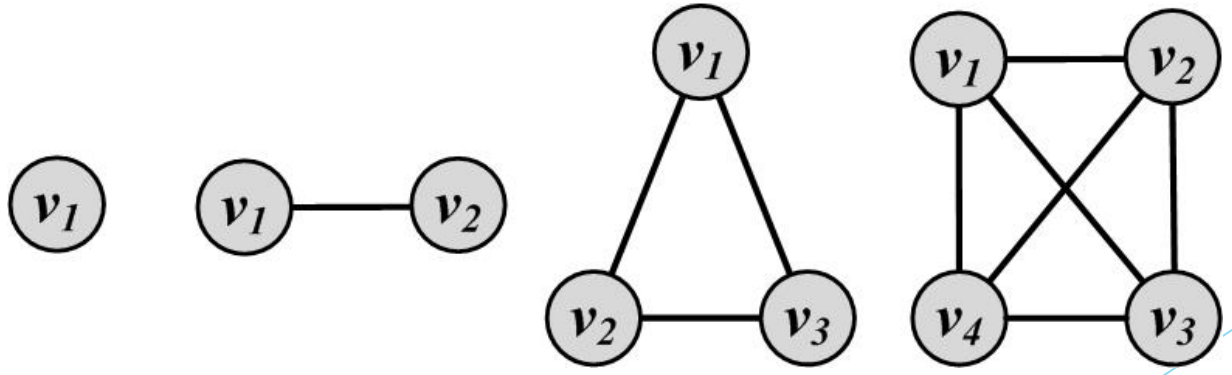
**What can be the terminal set here?**



# Complete Graphs

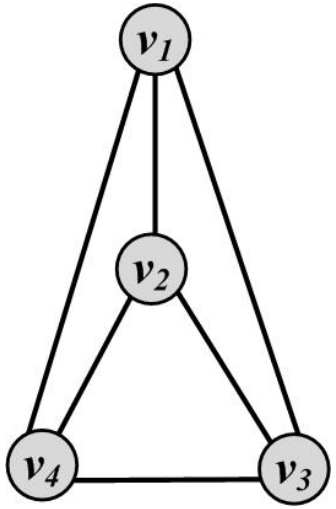
- A complete graph is a graph where for a set of nodes $V$, all possible edges exist in the graph
- In a complete graph, any pair of nodes are connected via an edge

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

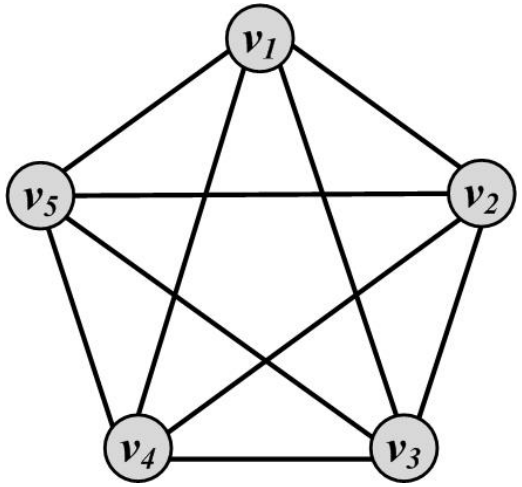


# Planar Graphs

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

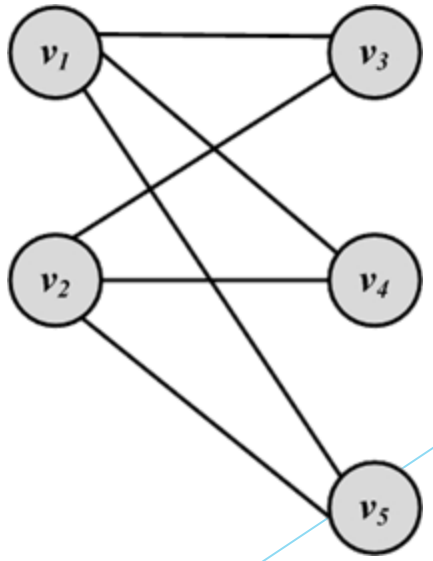



Non-planar Graph

# Bipartite Graphs

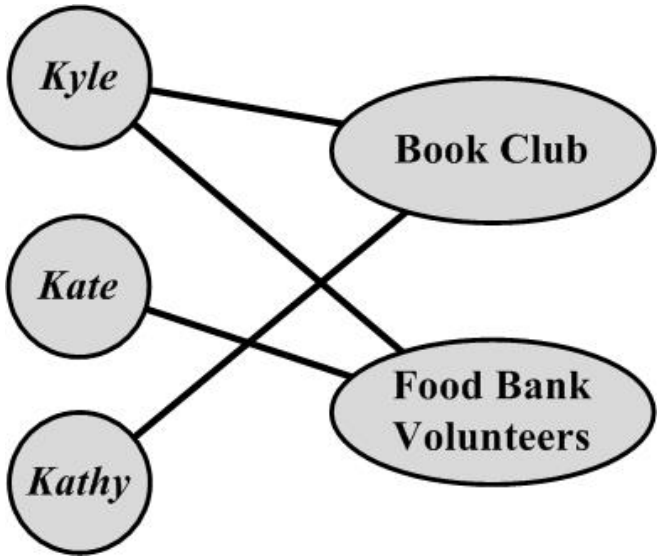
A bipartite graph $G(V, E)$ is a graph where the node set can be partitioned into two sets such that, for all edges, one end-point is in one set and the other end-point is in the other set.

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



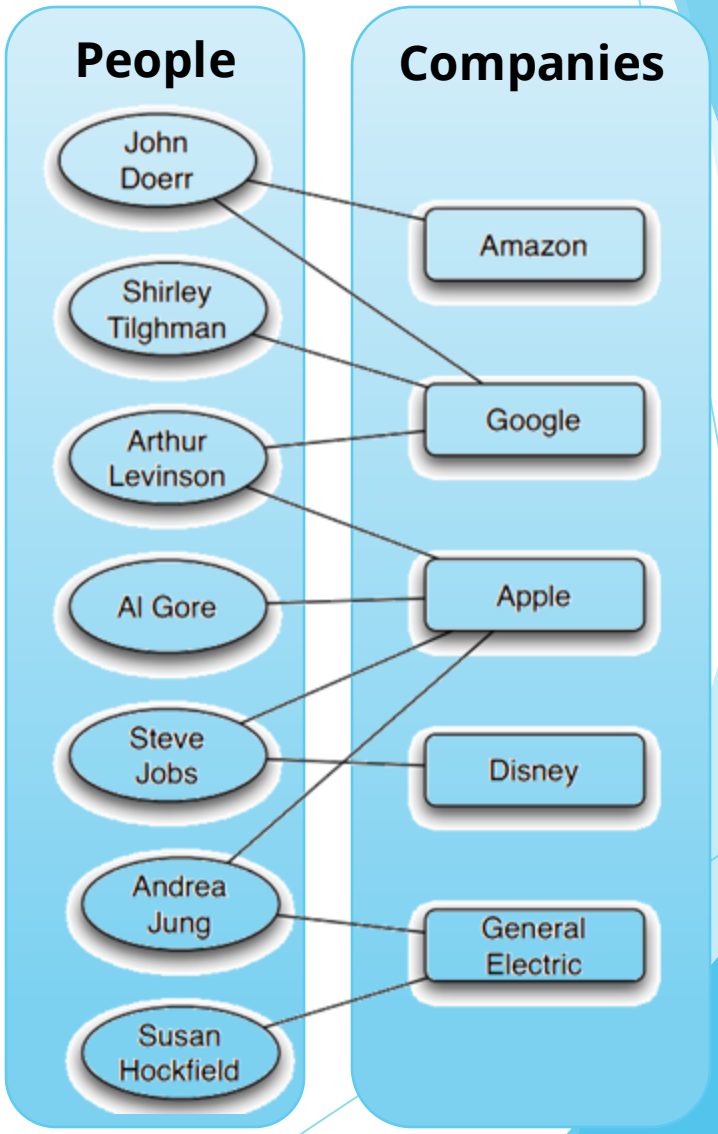
# Affiliation Networks

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



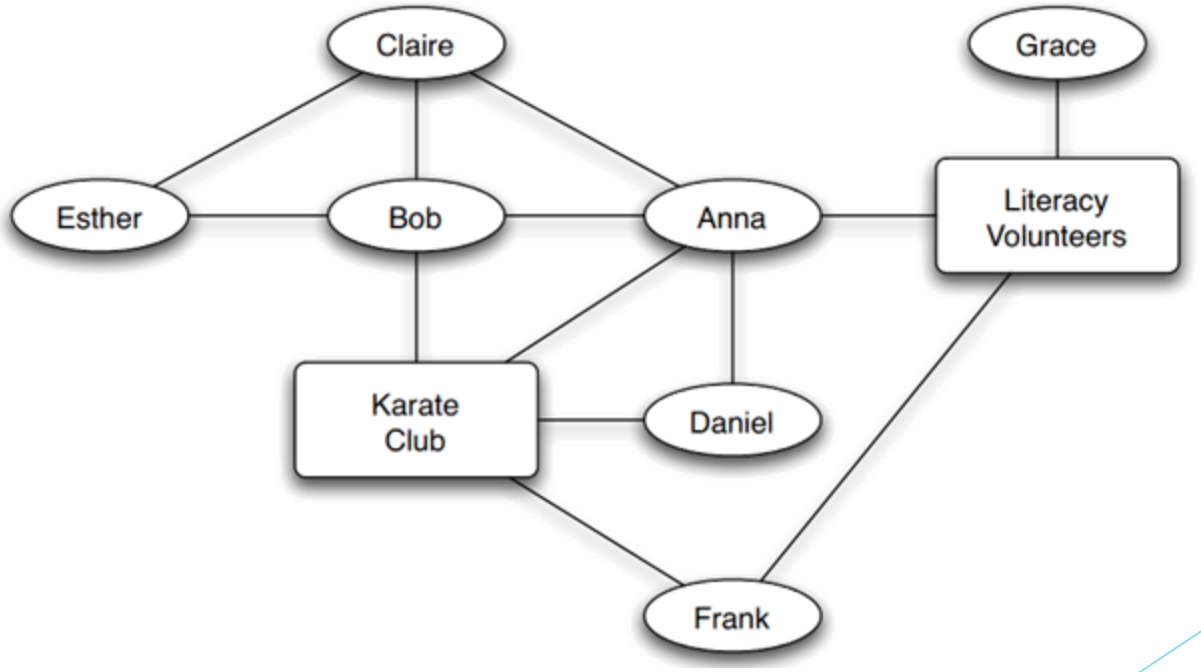
# Affiliation Networks: Membership

Affiliation of people on corporate boards of directors



# Social-Affiliation Network

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



# Projection of Bipartite to Unimodal Network

- Choose one of the vertex sets and connect two vertices from that set if they are linked to the same vertex of the other kind

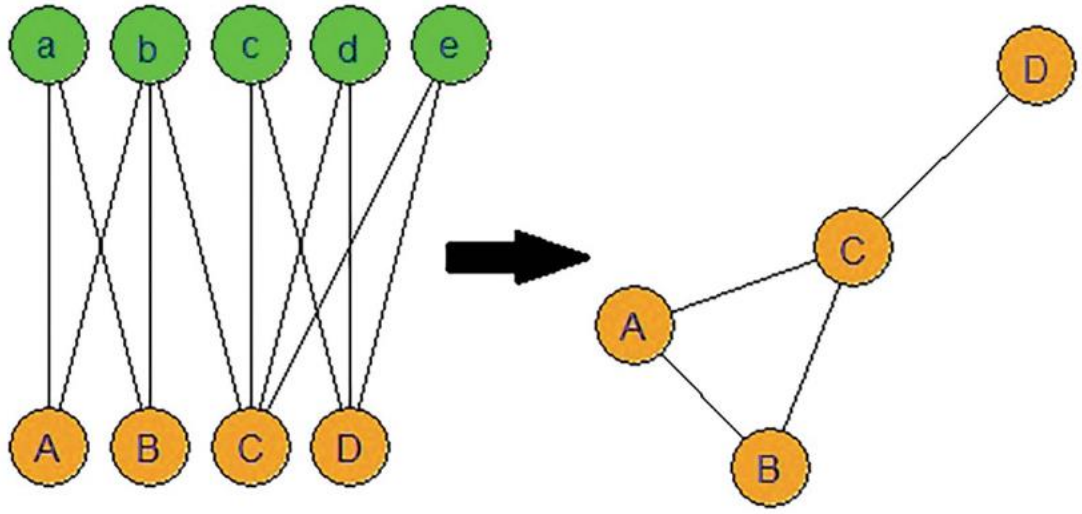



**Fig. 2.11** An example of projection

From Elements of Network Science

# Weighted Unimodal Network

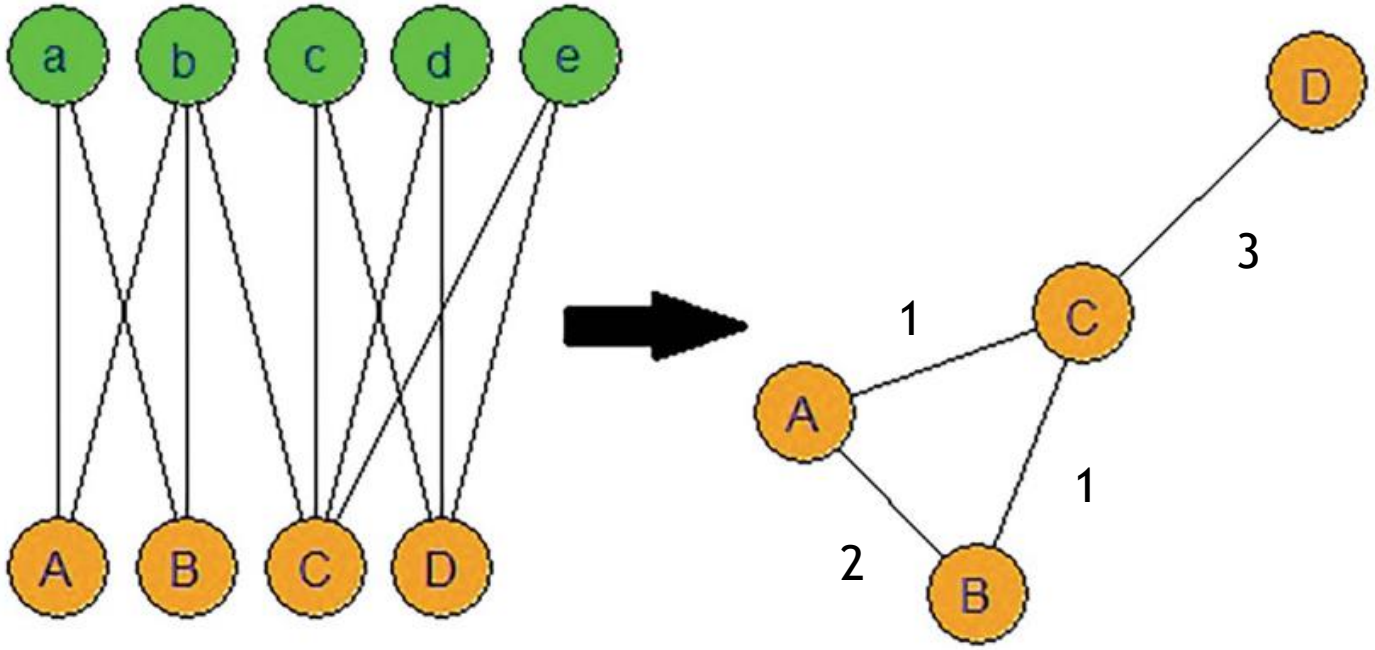
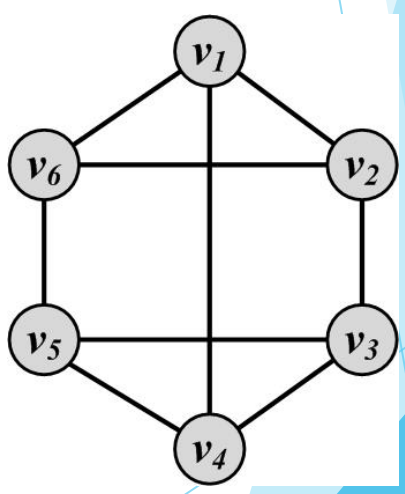



**Fig. 2.11** An example of projection

# Regular Graphs

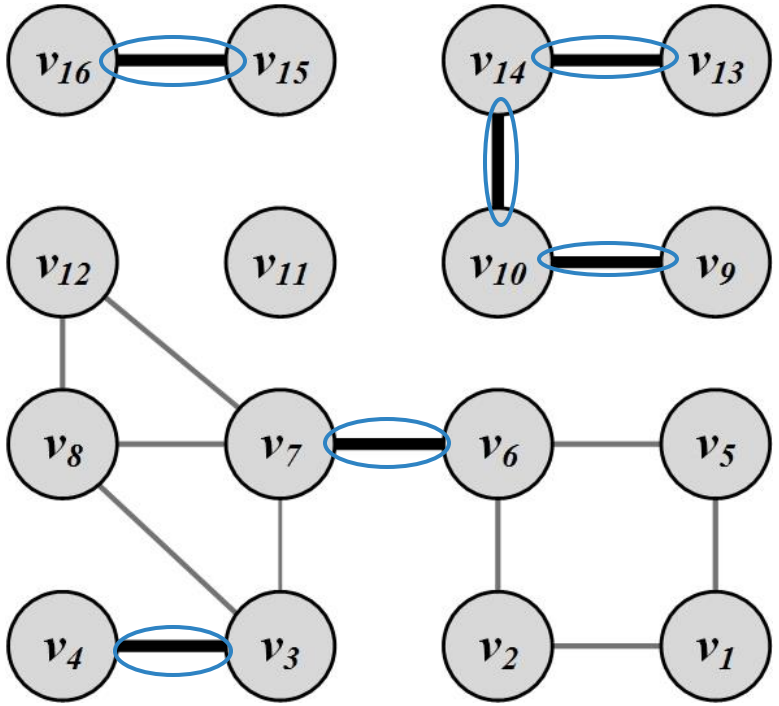
- A regular graph is one in which all nodes have the same degree
- Regular graphs can be connected or disconnected
- In a $k$-regular graph, all nodes have degree $k$
- Complete graphs are examples of regular graphs




Regular graph
With $k = 3$

# 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

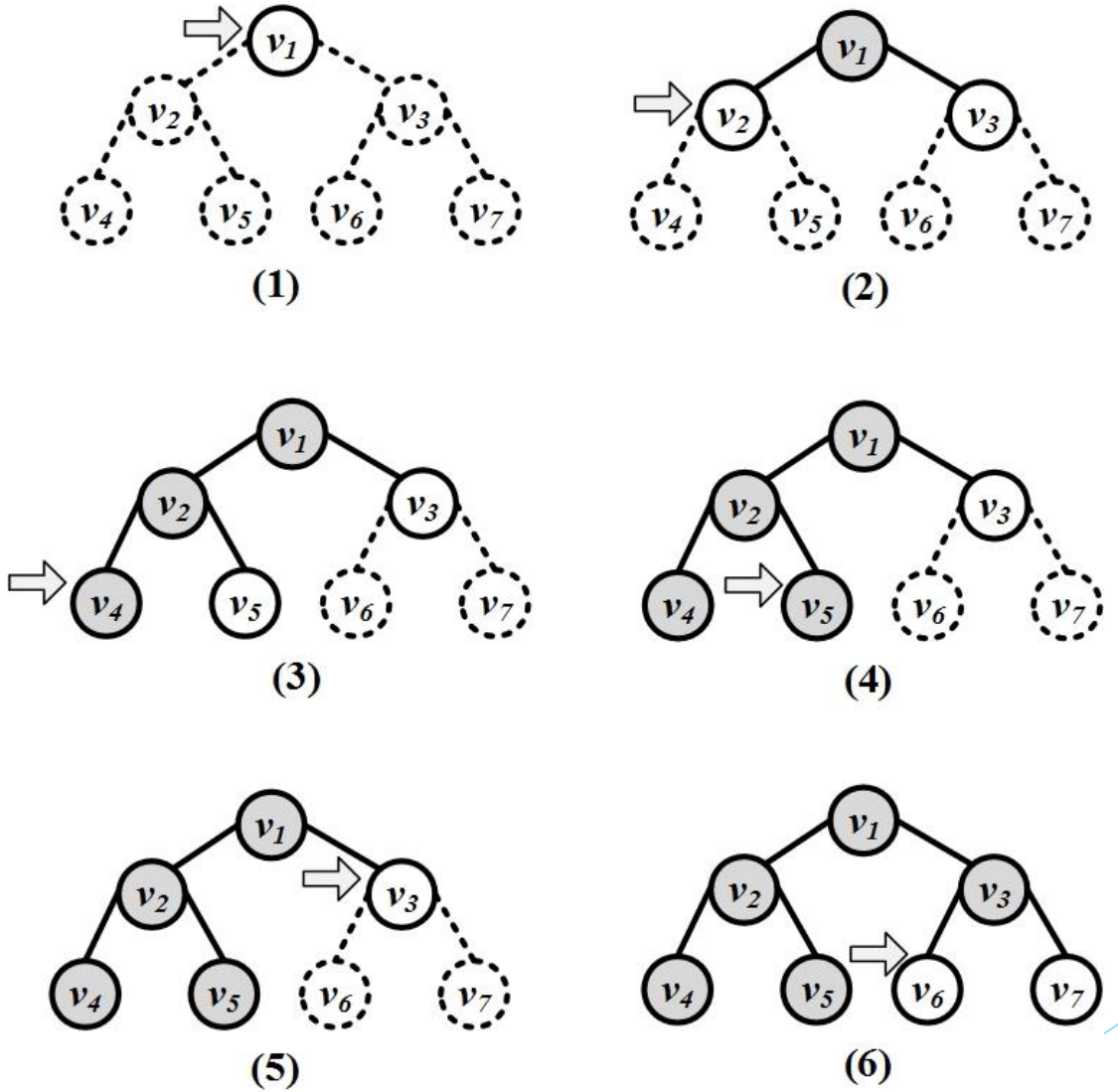
# DFS Algorithm

**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

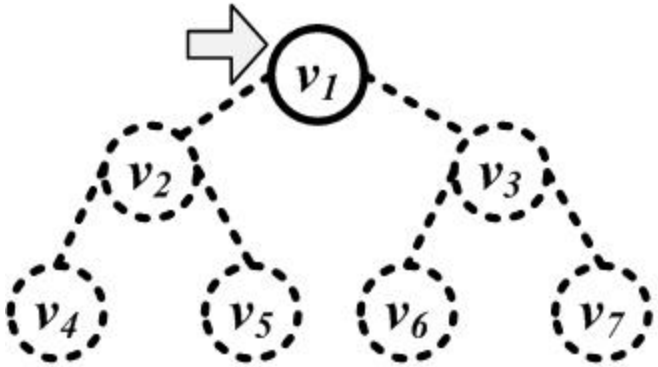
# BFS Algorithm

**Algorithm 2.3** Breadth-First Search (BFS)

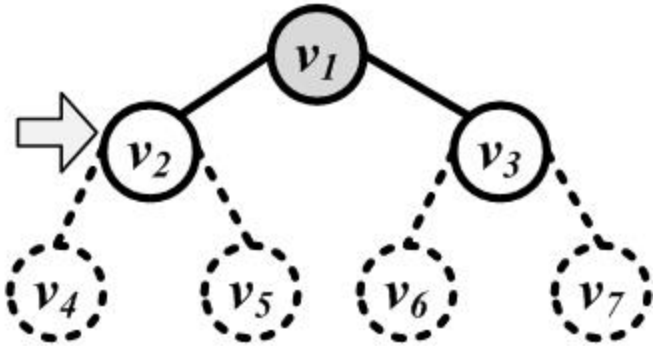
**Require:** Initial node $v$, graph/tree $G(V, E)$, queue $Q$

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**

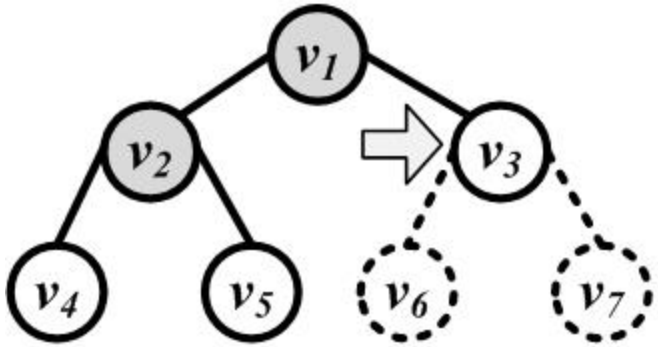
# Breadth-First Search (BFS)



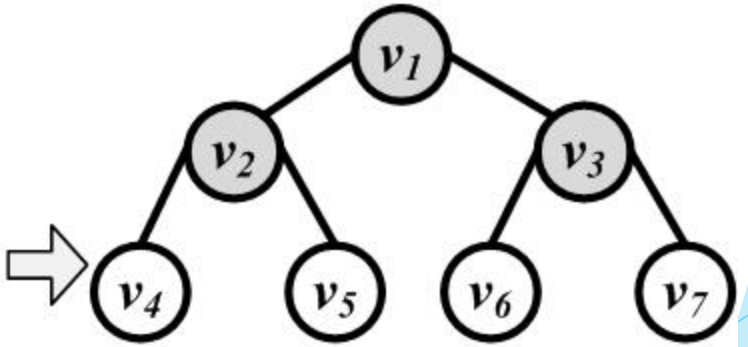

(1)




(2)




(3)




(4)

# 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

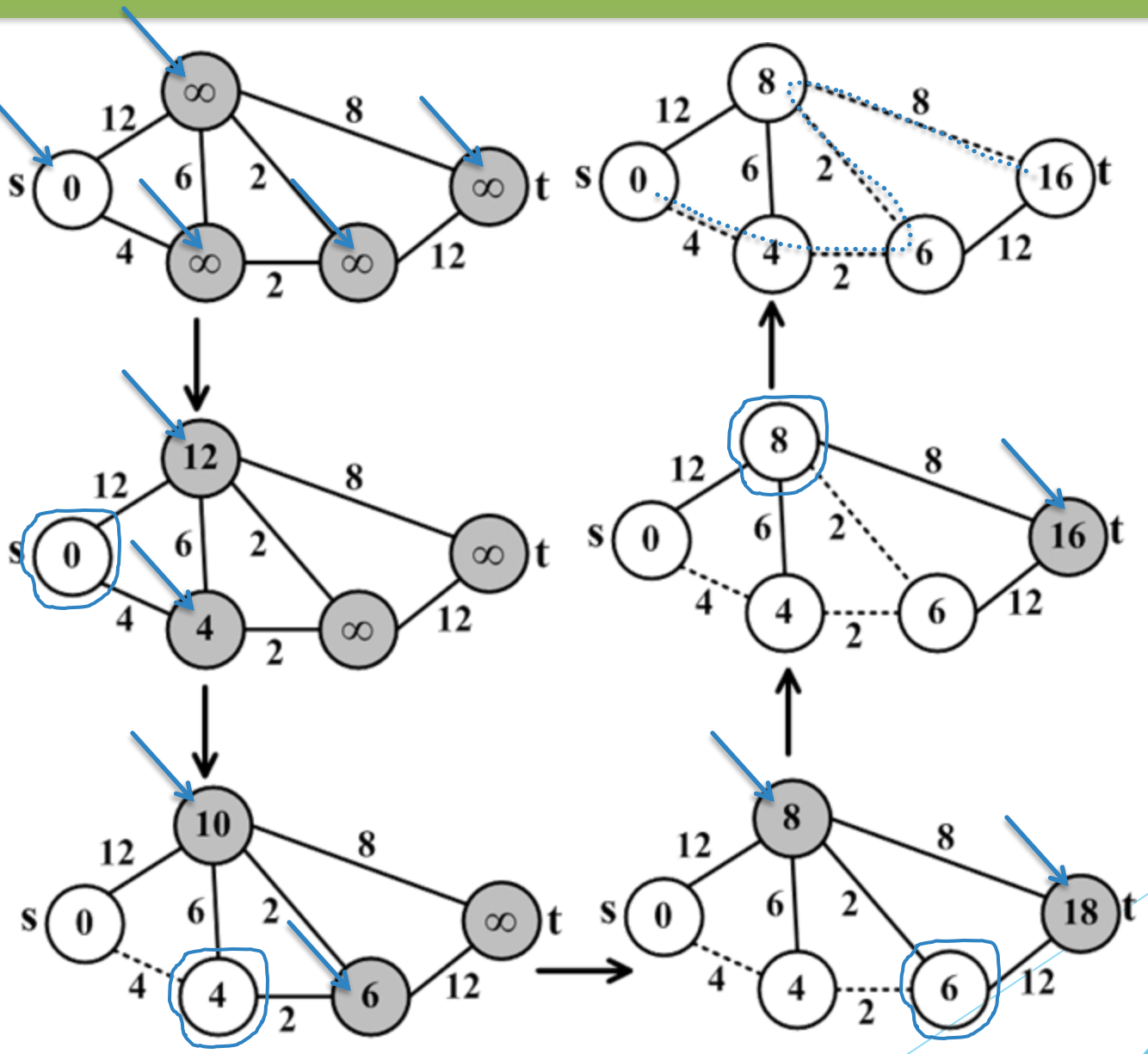
# Dijkstra's Algorithm: Finding the shortest path

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

Tentative distance = current distance + edge weight

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

# 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

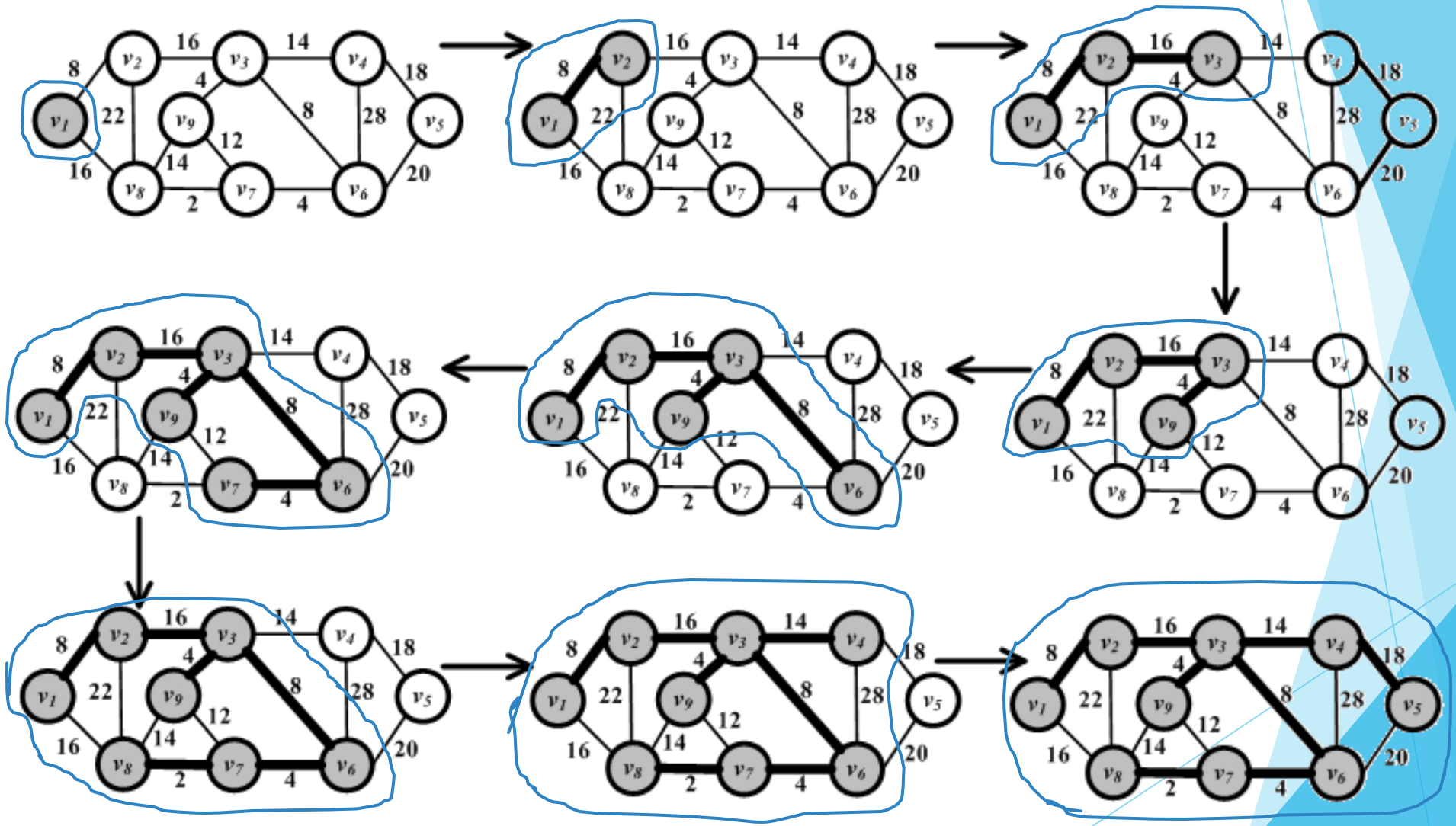
# Finding
# Minimum Spanning Tree

# 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



# HW #2 – Design of Networks

- In this homework you will create networks/graphs from 2 different datasets
  - Dataset 1: Chicago Community Areas https://en.wikipedia.org/wiki/Community_areas_in_Chicago
    - Nodes = Community areas
    - Edges = Shared boundary (i.e. adjacency) with other community area
  - Dataset 2: CS 579 Class Participant data Social Network Data collection.xlsx
    - Nodes = Class Participants, Entities in common
    - Edges = Shared entity

    You will create a bipartite graph. Some data cleaning will be necessary. State and justify any assumptions you make during the data cleaning.

    You will then create a unimodal graph that is a projection of the bipartite graph

  You will create labeled visualizations of all 3 of these graphs

  Any use of AI tools (ChatGPT etc.) must be recorded and submitted

- Due by midnight on Sept 15, submit to Canvas


Chicago
Central
Far North Side
Far Southeast Side
Far Southwest Side
North Side
Northwest Side
South Side
Southwest Side
West Side
A. Burnside
B. Oakland
C. Montclare
O'Hare
Edison Park
Norwood Park
Jefferson Park
Forest Glen
North Park
Rogers Park
West Ridge
Edge-water
Lincoln Square
Albany Park
Uptown
Portage Park
Irving Park
Dunning
North Center
Lakeview
Avondale
(C)
Belmont Cragin
Hermosa
Logan Square
Lincoln Park
Austin
Humboldt Park
West Town
Near North
West Garf. Park
East Garfield Park
Near West Side
Loop
North Lawndale
Near South
Lower West Side
South Lawndale
Bridge-Port
Armor Square
Douglas
McKinley Park
(B)
Archer Heights
Brigton Park
New City
Fuller Park
Grand Blvd
Ken-Wood
Garfield Ridge
West Elsdon
Gage Park
Wash-ington Park
Hyde Park
Clearing
West Lawn
Chicago Lawn
West Engle-Wood
Engle-Wood
Woodlawn
Greater Grand Crossing
South Shore
Ashburn
Auburn Gresham
Chatham
Avalon Park
South Chicago
(A)
Calumet Heights
Wash-ington Heights
Beverly
Roseland
Pullman
East Side
South Deering
Mount Greenwood
Morgan Park
West Pullman
Riverdale
Hegewisch