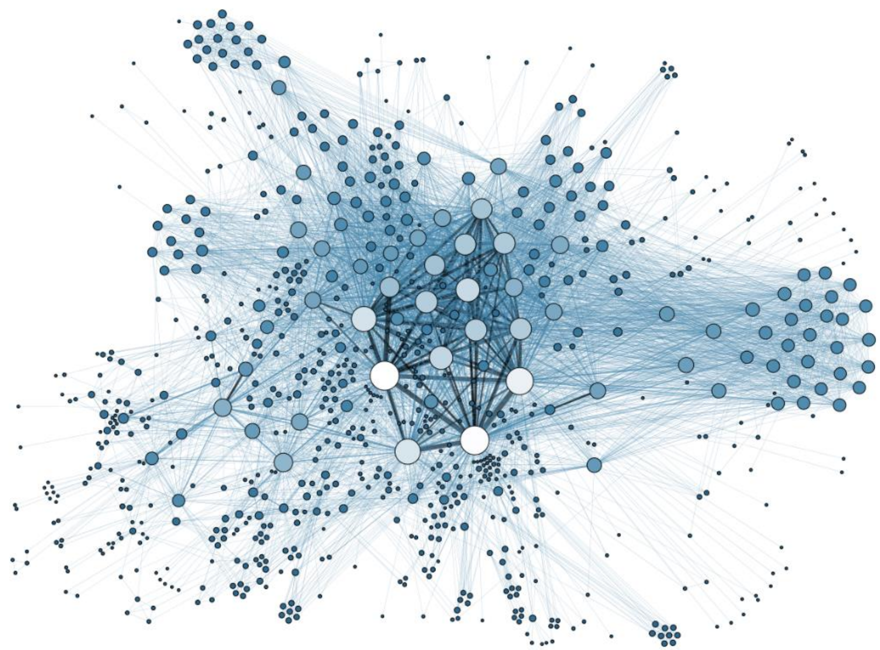
Social Media & Network Analytics



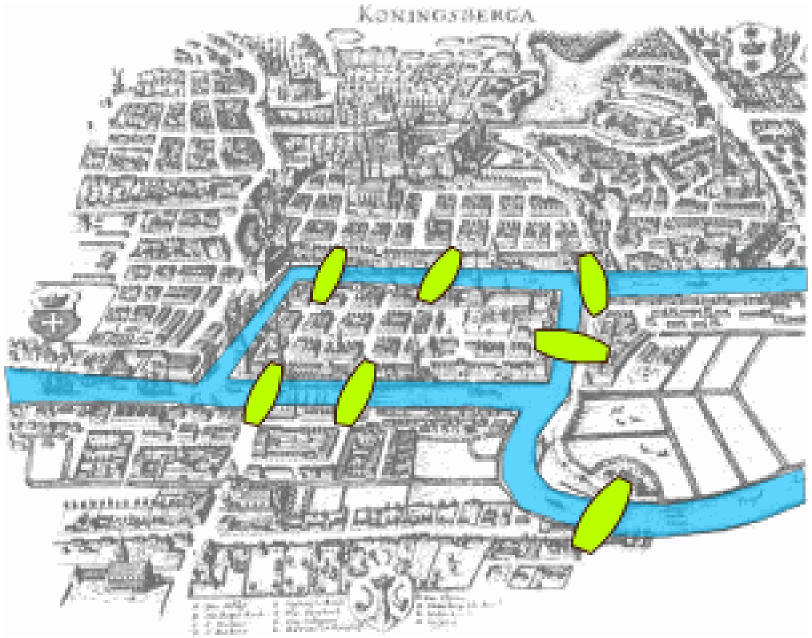
Graph Introduction

Outline

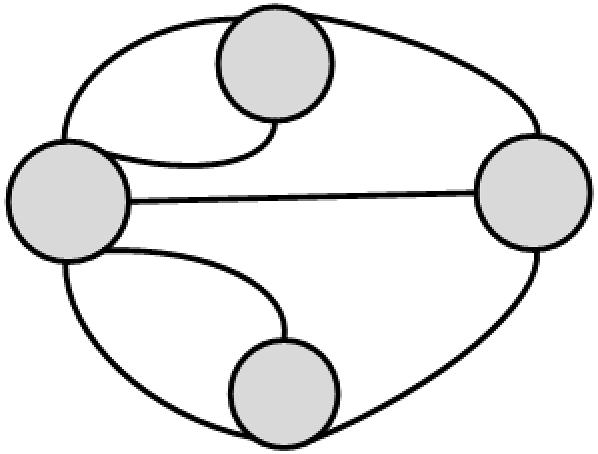
- Motivation for studying graphs and networks
- Graph basics
- Graph representation
- Types of graphs
- Connectivity
- Subgraphs
- Graph Algorithms

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)



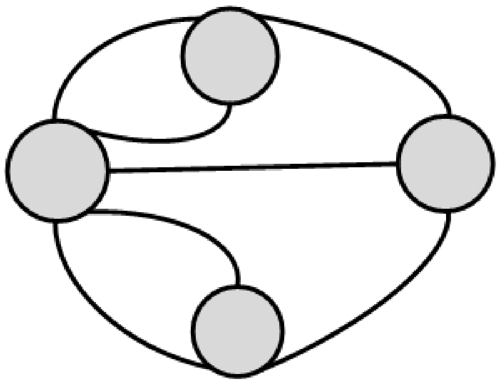
Graph Representation

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

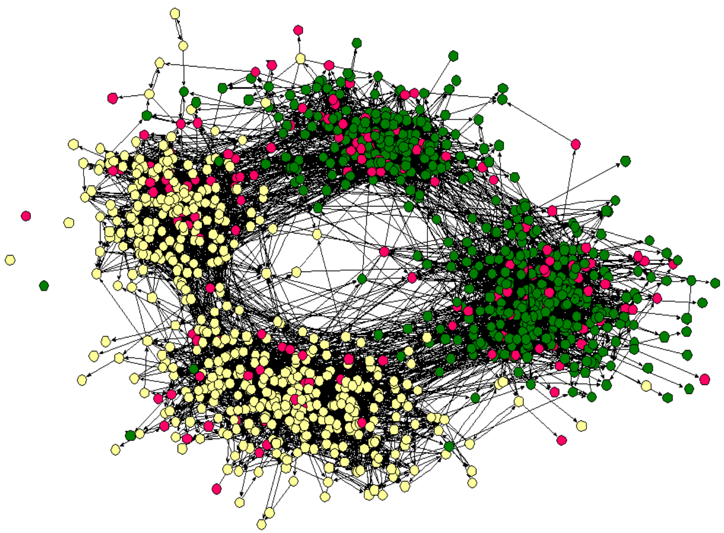
- This property does not hold in this problem



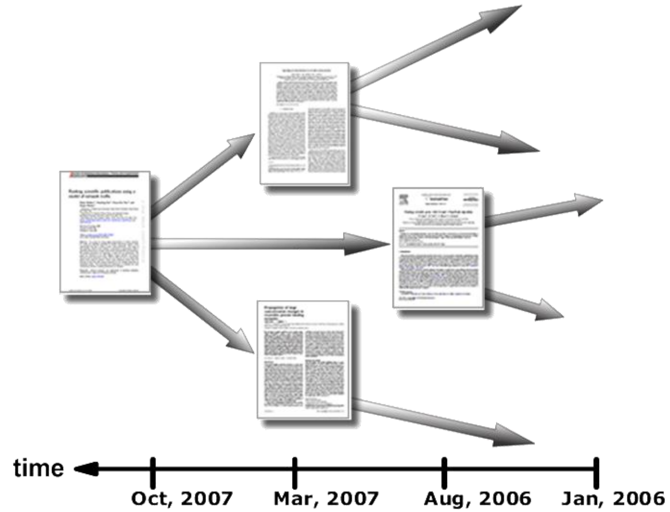
Why Graphs?

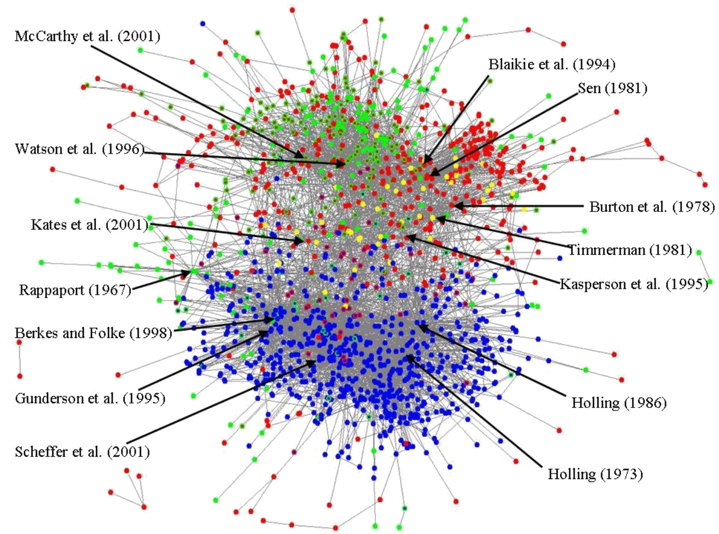
Natural representation of relational data, which are pervasive!

Friendship Networks



High school friendship





Citation Networks

Network of the US Interstate Highways

Social Networks and Social Network Analysis

- Terminology: A network is a graph, and vice versa.
- 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

Outline

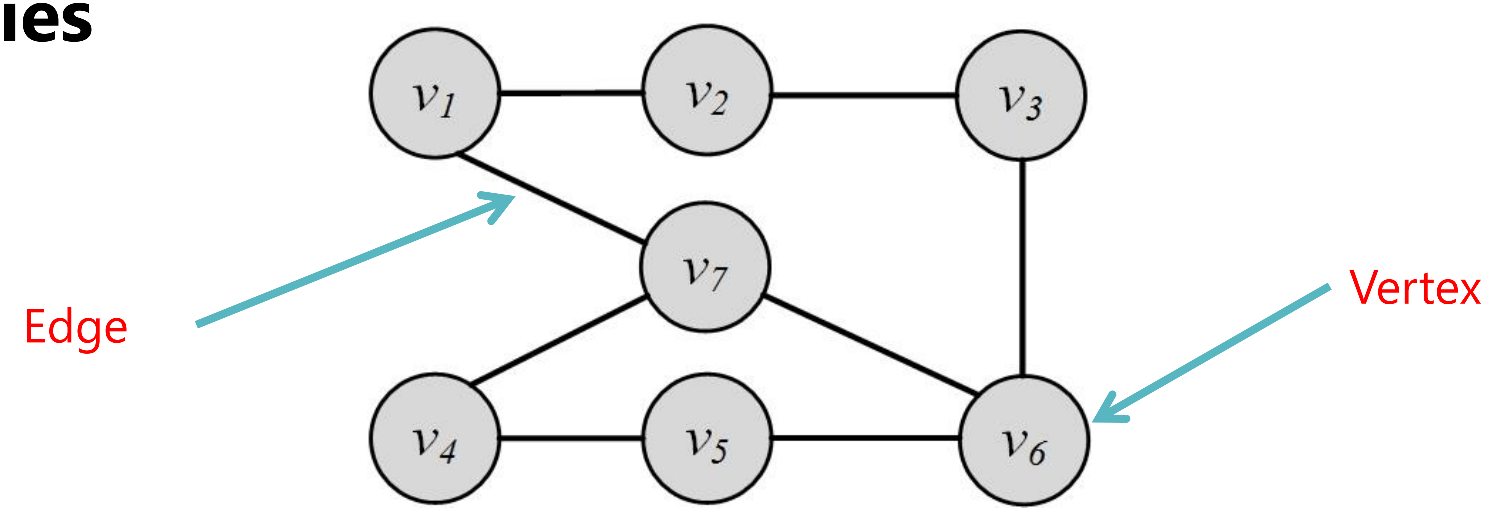
- Motivation for studying graphs and networks
- Graph basics
- Graph representation
- Types of graphs
- Connectivity
- Subgraphs
- Graph Algorithms

Graph Basics

Vertices and Edges

A network or graph, G(V,E) is **a set of vertices** V connected by a **set of edges** E connecting a pair of vertices

- **Vertices** (plural of **vertex**) also known as **nodes** or **actors**
- Connections are referred to as **edges**, **links** or **ties**



Vertices and Edges

- Let the set of vertices be denoted as

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

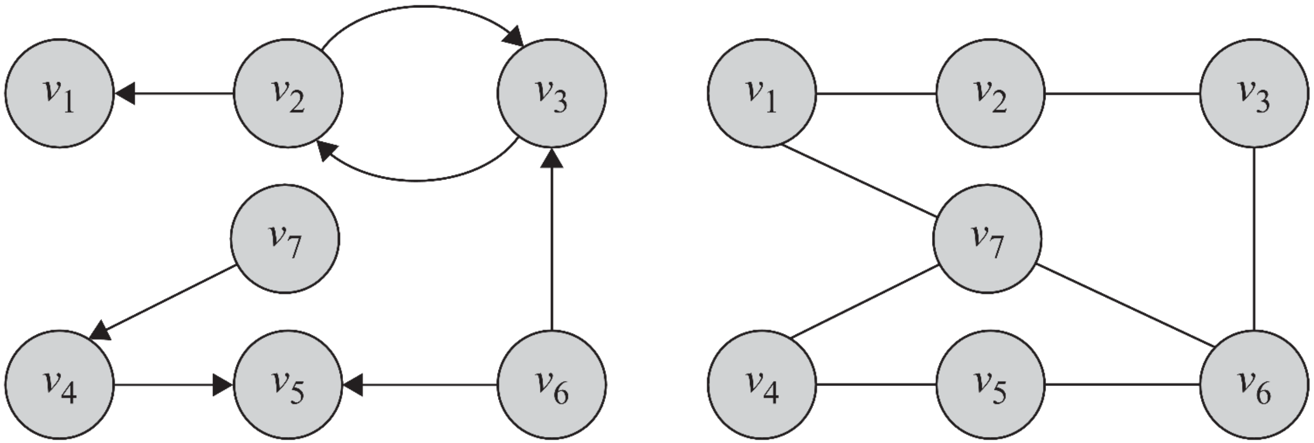
- The size of the graph is $|V| = \mathbf{n}$
- Let the set of edges be denoted as

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

- Number is edges (size of the edge-set) is $|E| = \mathbf{m}$
- Edges are represented using their end-points $e(v_2, v_1)$

Directed Edges and Directed Graphs

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



(a) Directed Graph (b) Undirected Graph

- In undirected graphs edges are symmetric

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 v_i is usually presented using notation d_i

In Directed graphs:

- In-degree, d_i^{in}, is the number of edges pointing towards a node v_i
- Out-degree, d_i^{out}, is the number of edges pointing away from a node v_i

Example of Neighbourhood

Degree Distribution

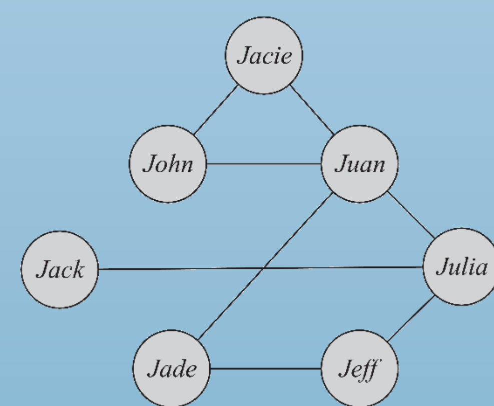
When dealing with very large graphs, how nodes' degrees are distributed is an important concept to analyze and is called ***Degree Distribution***

$\pi(d) = \{p_1, p_2, \ldots, p_n\}$ (Degree sequence)

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

n_d is the number of nodes with degree d

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



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

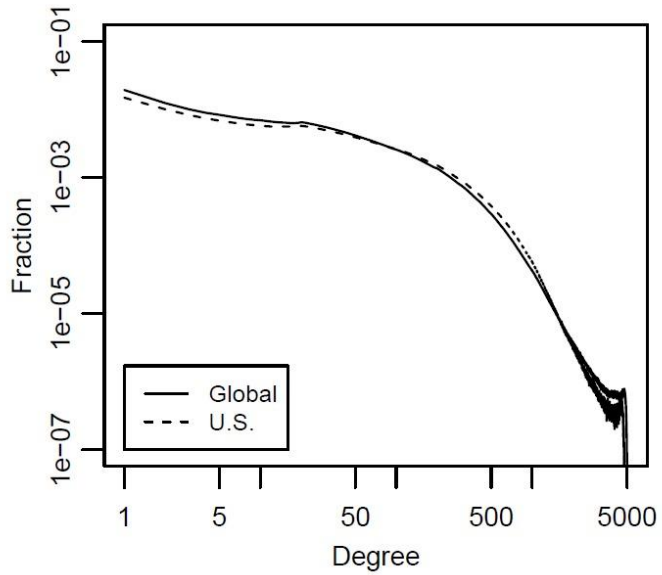
Degree Distribution Plot

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**)



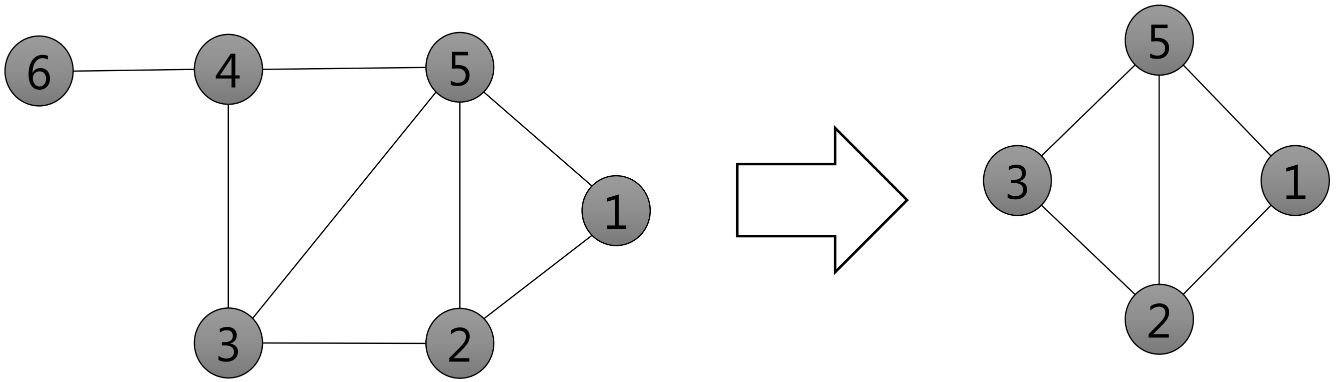
Facebook Degree Distribution

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 $G(V,E)$

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

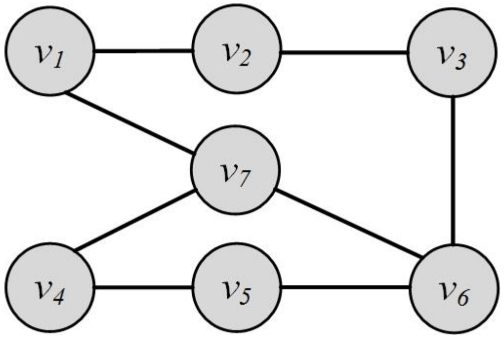


Outline

- Motivation for studying graphs and networks
- Graph basics
- Graph representation
- Types of graphs
- Connectivity
- Subgraphs
- Graph Algorithms

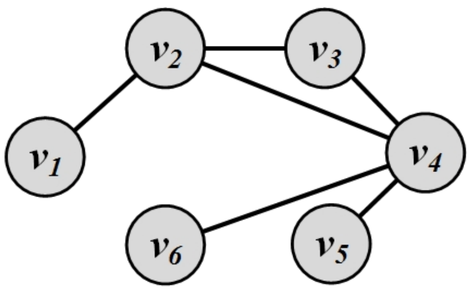
Graph Representation

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



Graph Representation

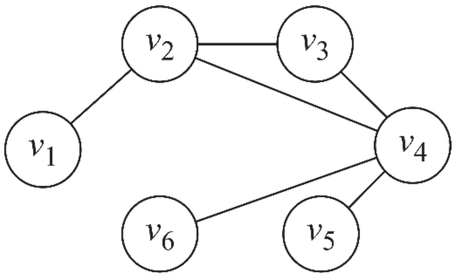
- Graph representation is straightforward and intuitive, but it cannot be effectively manipulated using mathematical and computational tools



- 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

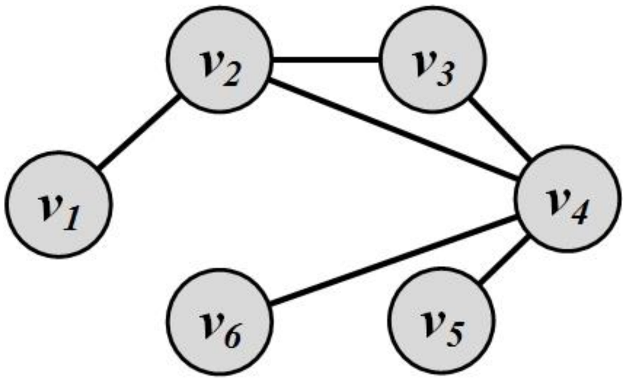
(b) Adjacency Matrix

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 or its neighbours
- 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

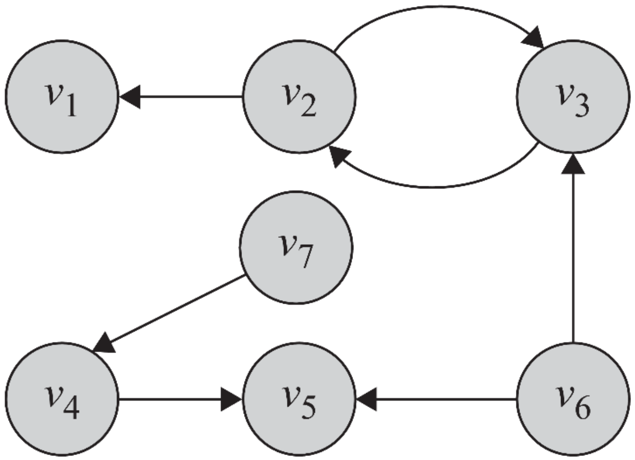
Outline

- Motivation for studying graphs and networks
- Graph basics
- Graph representation
- Types of graphs
- Connectivity
- Subgraphs
- Graph Algorithms

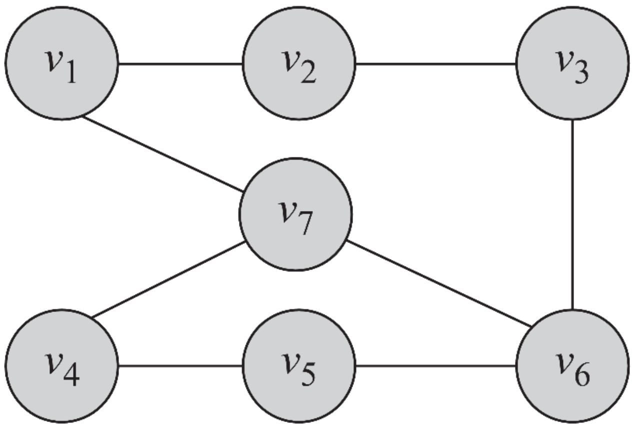
Types of Graphs

- Directed/Undirected/Mixed, Simple/Multigraph, Weighted, Signed Graph

Directed/Undirected/Mixed Graphs



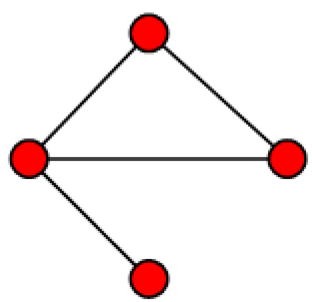
- The adjacency matrix for directed graphs is often not symmetric ($\boldsymbol{A} \neq \boldsymbol{A}^{\boldsymbol{T}}$)
  - Generally, $\boldsymbol{A}_{ij} \neq \boldsymbol{A}_{ji}$
  - We can have equality though



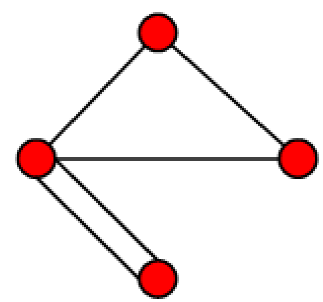
The adjacency matrix for undirected graphs is symmetric ($\boldsymbol{A} = \boldsymbol{A}^{\boldsymbol{T}}$)

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



Simple graph

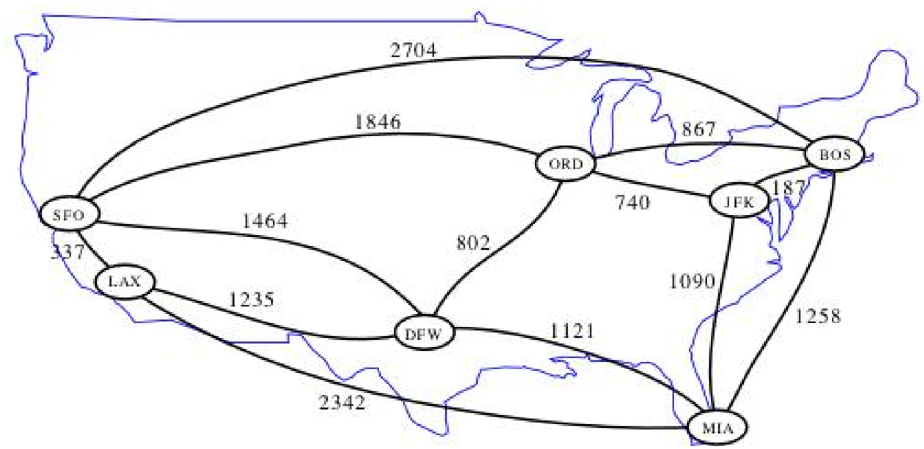


Multigraph

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

Weighted Graph

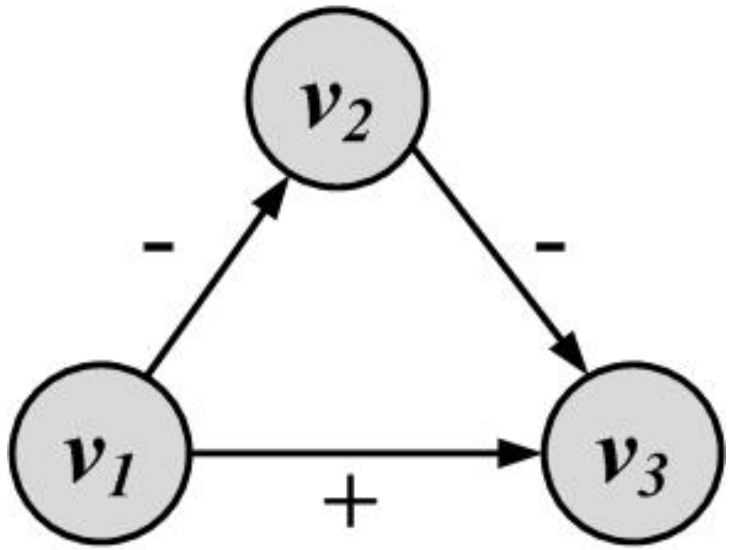
- A weighted graph $\boldsymbol{G}(\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{W})$ is one where edges are associated with weights
    - For example, a graph could represent a map where nodes are cities and edges are routes between them
        - The weight associated with each edge could represent the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



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

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

Break time

- **Trivia**: What does the phrase "Six degrees of separation" mean? Where does it come from?

Outline

- Motivation for studying graphs and networks
- Graph basics
- Graph representation
- Types of graphs
- Connectivity
- Subgraphs
- Graph Algorithms

Connectivity in Graphs

- **Adjacent nodes/Edges, Walk/Path**

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

Walk

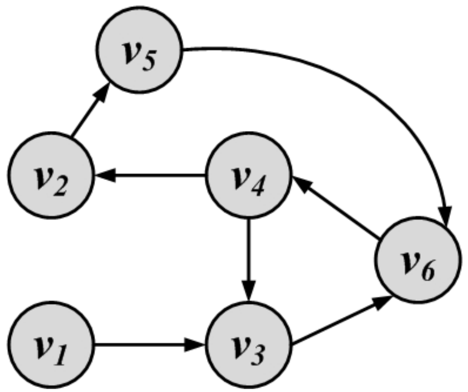
Walk: A walk is a sequence of incident edges visited one after another

- **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:

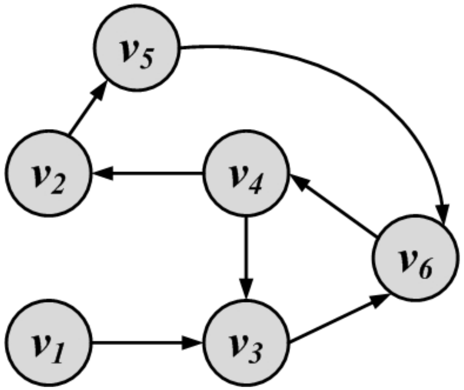
  the number of visited edges



Length of walk=

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



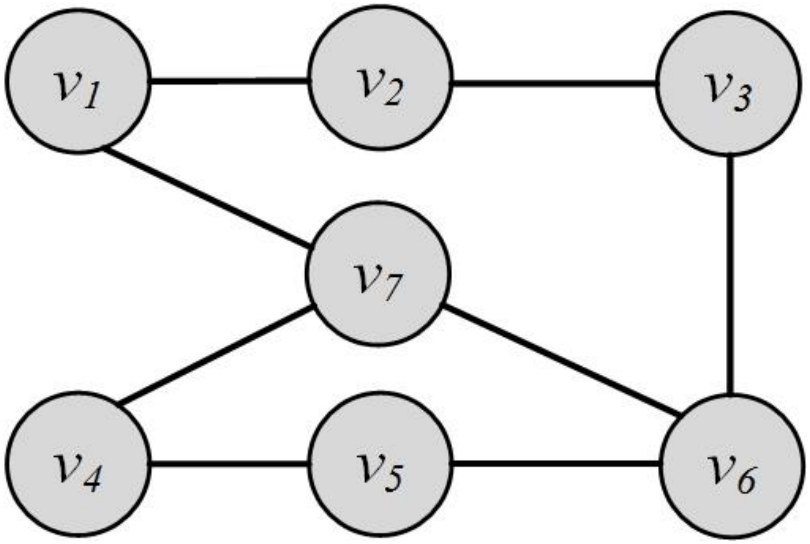
Length of path=

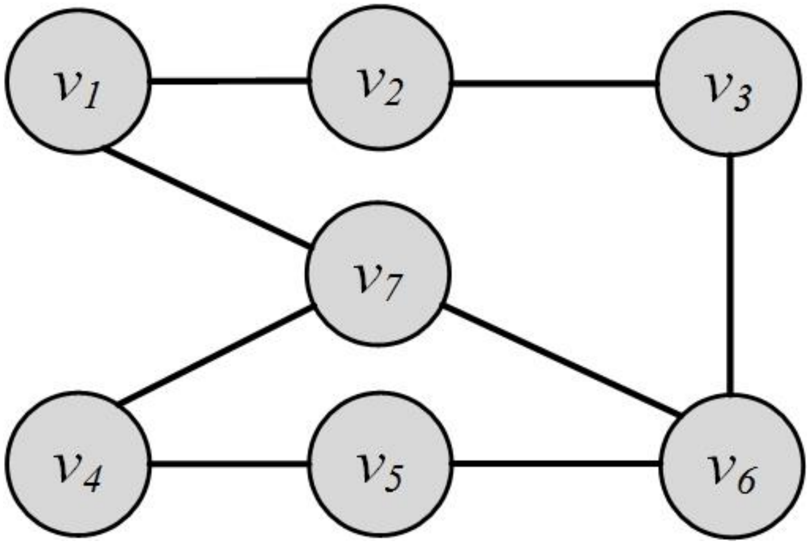
Random walk

- A walk that in each step the next node is selected randomly among the neighbors
- The random walk can also be guided:
  - 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 Examples





Connectivity

- **A node v_i is connected to node v_j** (or reachable from v_j) if 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

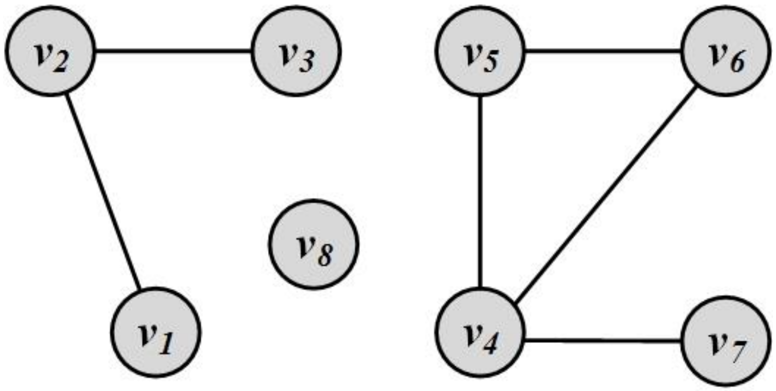
Component

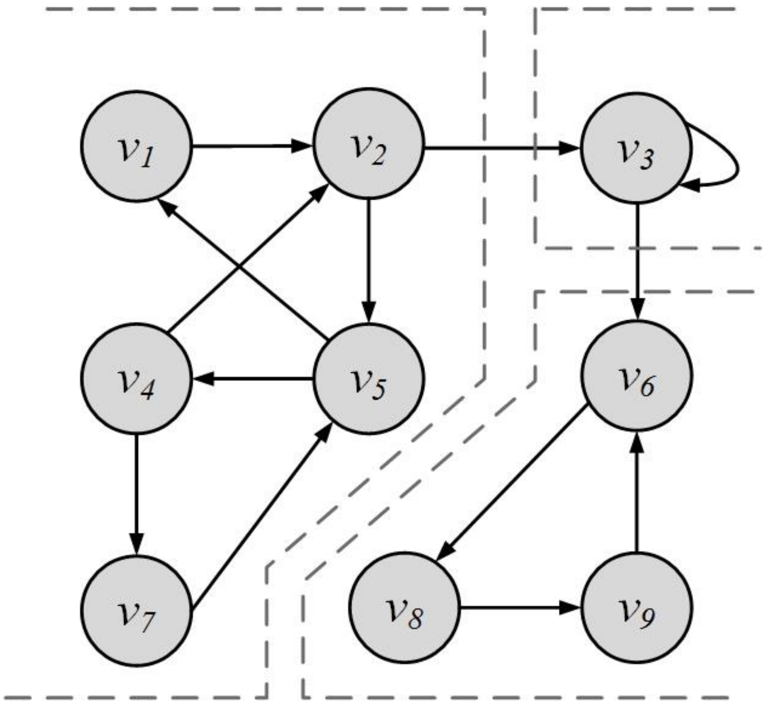
- A **component** in an undirected graph is a connected **subgraph**, i.e., there is a path between every pair of nodes inside the component

- 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 a 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

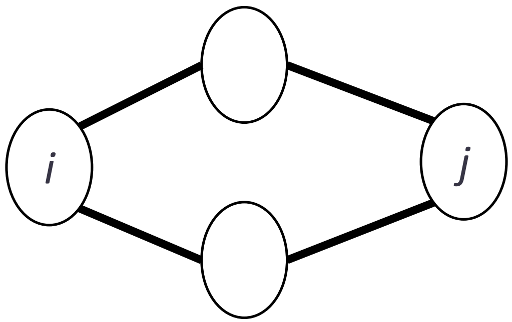
$$diameter_G = \max_{(v_i,v_j)\in V\times V} l_{i,j}$$

- Trivia: What is the diameter of the web?

Adjacency Matrix and Connectivity

- Consider the following adjacency matrix

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



- Want to compute whether two vertices are connected?
- Number of Common neighbors between node i and node j =

$$N(i) \cap N(j) = \sum_{k=1}^{n} A_{ik} A_{jk} = A_i A_j^T$$

- Equivalent to [*ij*] entry of matrix $A \times A^T = A^2$
- Common neighbors are paths of length 2, so A^2 gives number of paths of length 2 between any pairs of vertices
- Similarly, what is A^3?

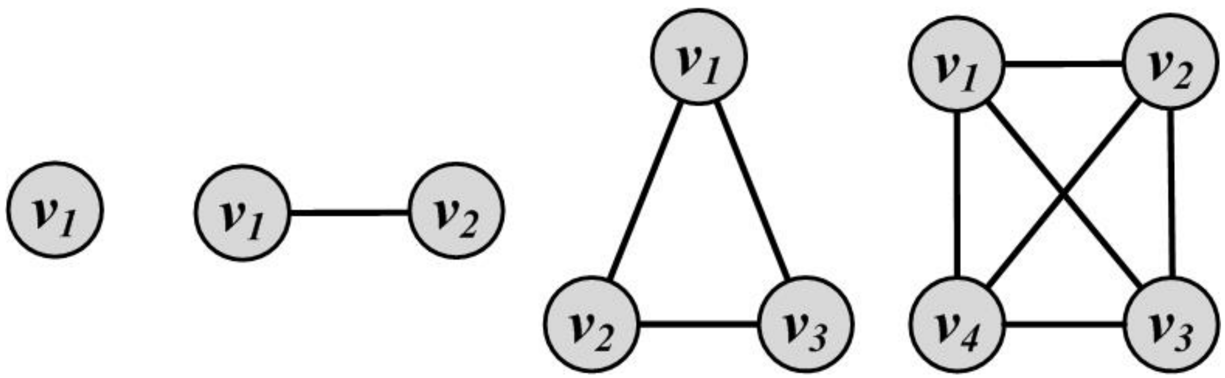
Outline

- Motivation for studying graphs and networks
- Graph basics
- Graph representation
- Types of graphs
- Connectivity
- Subgraphs
- Graph Algorithms

Special Subgraphs

Complete Graphs

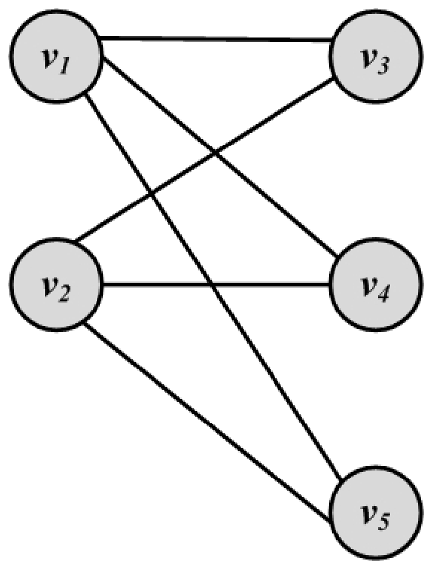
- A complete graph is a graph where for a set of vertices V, all possible edges exist in the graph
- In a complete graph, any pair of vertices are connected via an edge
  - For undirected, no-self edges graph, $|E| = \binom{|V|}{2}$

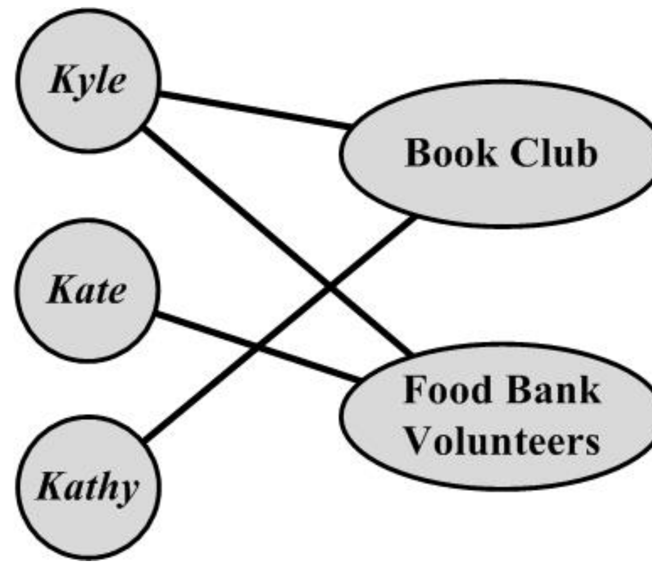


Bipartite Graphs

- 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.

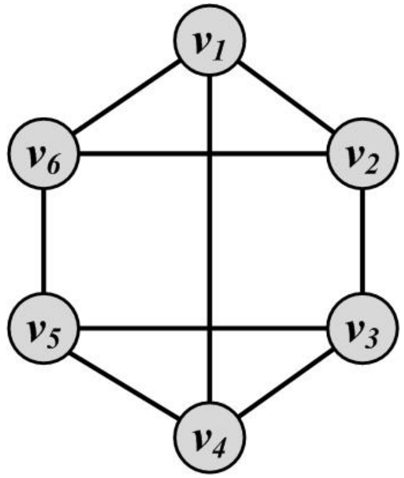
$$\begin{cases} V = V_L \cup V_R, \\ V_L \cap V_R = \emptyset, \\ E \subset V_L \times V_R. \end{cases}$$





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



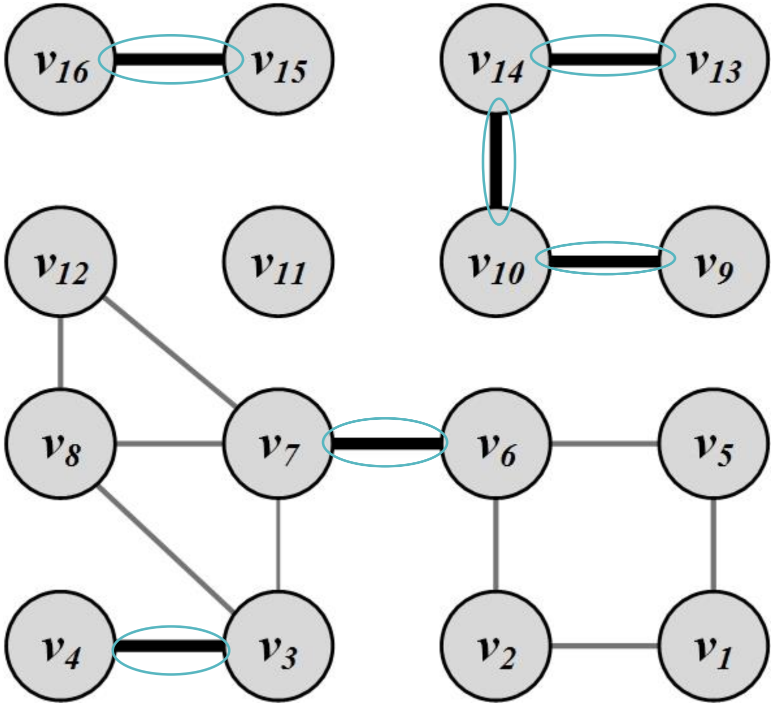
Regular graph
With $k = 3$

Egocentric Networks

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

Bridges (cut-edges)

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



Outline

- Motivation for studying graphs and networks
- Graph basics
- Graph representation
- Types of graphs
- Connectivity
- Subgraphs
- Graph Algorithms

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)

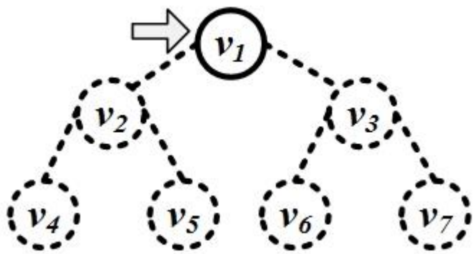
- Depth-First Search (DFS) starts from a node v_i, selects one of its neighbors v_j from $N(v_i)$ and performs Depth-First Search on v_j before visiting other neighbors in $N(v_i)$

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

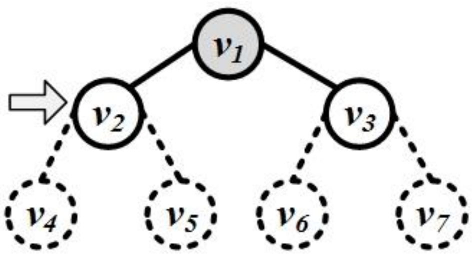
DFS Algorithm

1. Choose an arbitrary vertex and mark it visited.
2. From the current vertex, proceed to an **unvisited**, **adjacent** vertex and mark it visited.
3. Repeat 2nd step until a vertex is reached which has no adjacent, unvisited vertices (dead-end).
4. At each dead-end, **backtrack** to the last visited vertex and proceed down to the **next unvisited, adjacent** vertex.
5. The algorithm halts there are no **unvisited** vertices or we have reached our goal vertex.

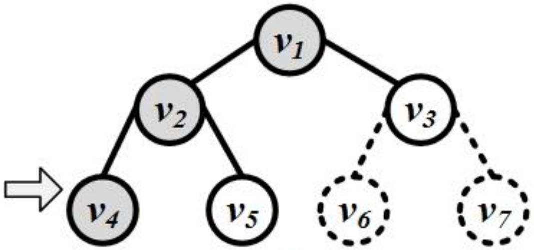
Depth-First Search (DFS): An Example



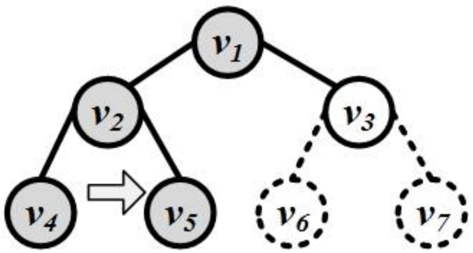
(1)



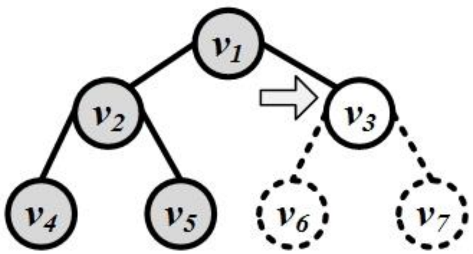
(2)



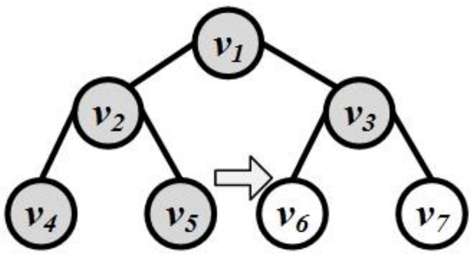
(3)



(4)



(5)



(6)

Breadth-First Search (BFS)

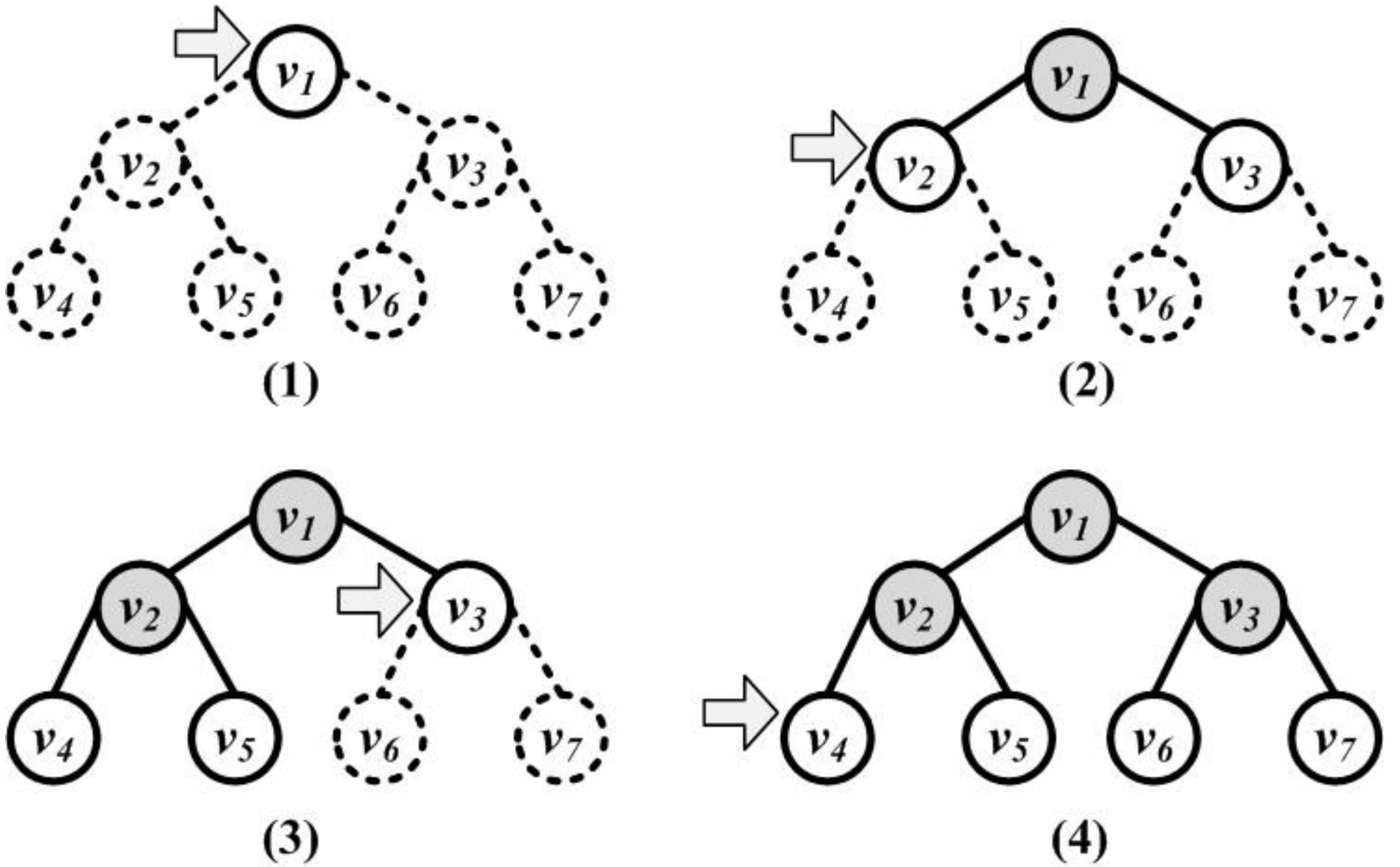
- BFS starts from a node and visits all its immediate neighbors first, and then moves to the second level by traversing their neighbors.

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

BFS Algorithm

1. Choose an arbitrary vertex v and mark it visited.
2. Visit and mark (visited) **each of the adjacent** (neighbour) vertices of v in **turn**.
3. Once **all** neighbours of v have been visited, select the first neighbour that was visited, and visit all its (unmarked) neighbours.
4. Then select the second visited neighbour of v, and visit all its unmarked neighbours.
5. The algorithm halts when we visited all vertices or reached our goal vertex.

Breadth-First Search (BFS)



Finding Shortest Paths

(not examinable)

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*

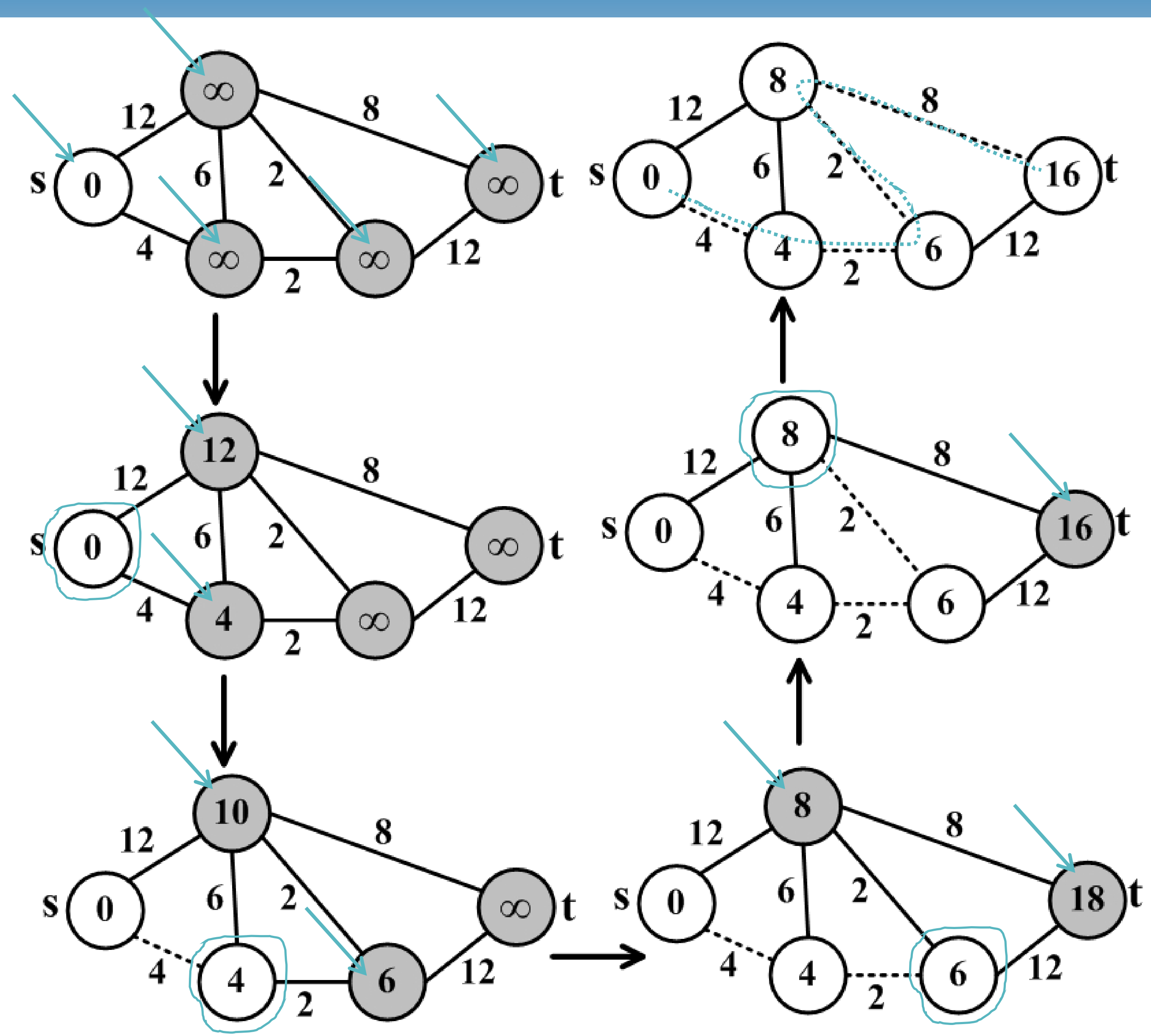
4. if the smallest tentative distance among the nodes in the *unvisited set* is infinity, then stop

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

Summary

- Introduced Graphs & Notion
  - Vertices, edges
  - Varies types of graphs
  - Walks, paths
  - Components
- Graph Algorithms
  - DFS, BFS
  - Shortest paths