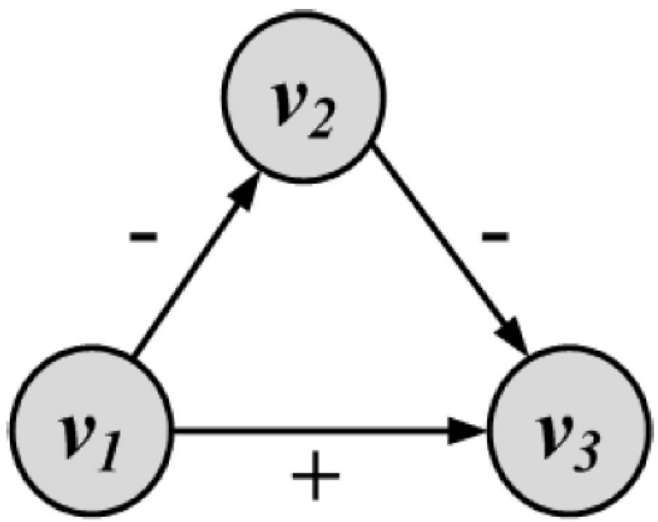
Webgraph

- A webgraph is a way of representing how internet sites are connected on the web
- In general, a webgraph 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 (i.e., links pointing to themselves)

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

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 one end-node

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

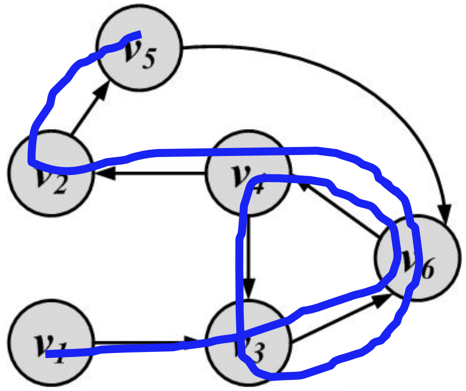
Walk, Trail, Tour, Path, and Cycle

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

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

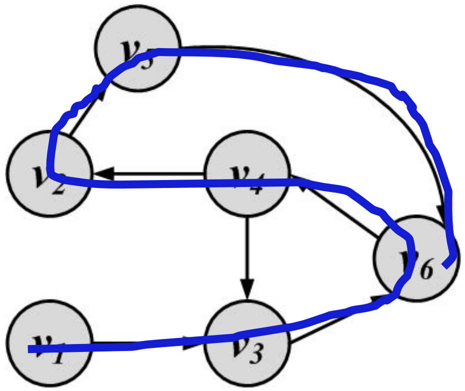
- Representing a walk:
  - A sequence of edges: $e_1, e_2, \ldots, e_n$
  - A sequence of nodes: $v_1, v_2, \ldots, v_n$
- *Length* of a walk: the number of visited edges

Length of walk=
8



Trail and Tour

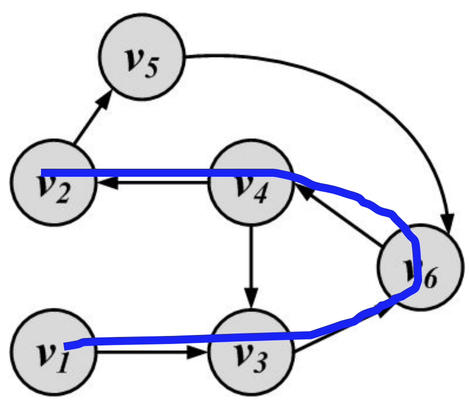
- A *trail* is a walk where **no edge is visited more than once** and all walk edges are distinct
- A closed trail (one that ends where it starts) is called a **tour** or **circuit**



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

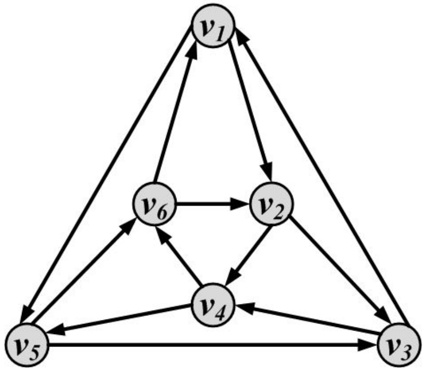
Length of path= 4



Examples

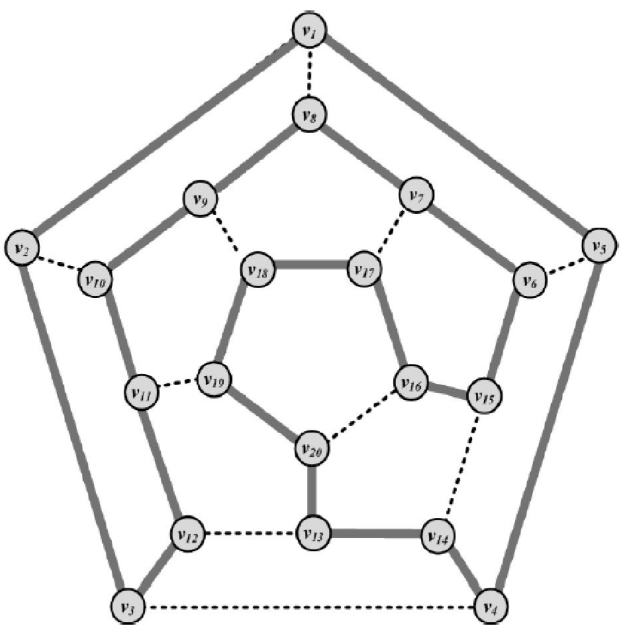
Eulerian Tour

- All edges are traversed only once
- Konigsberg bridges



Hamiltonian Cycle

- A cycle that visits all nodes



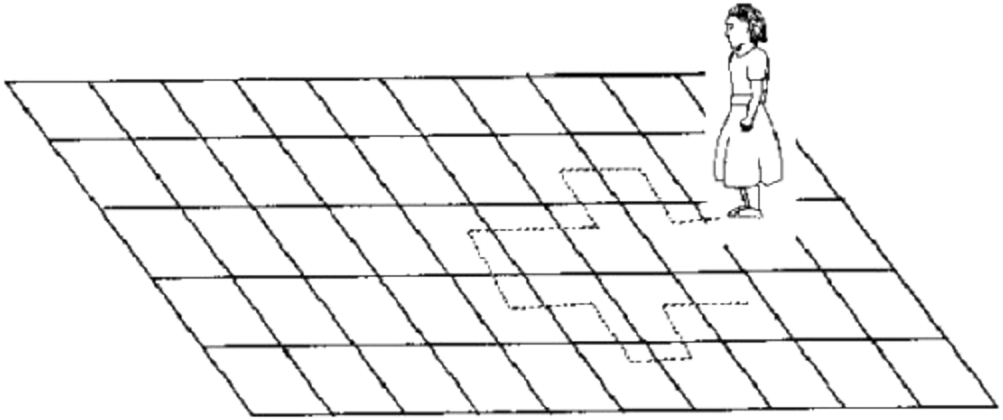
Random walk

- A walk that the next node in each step 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

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

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

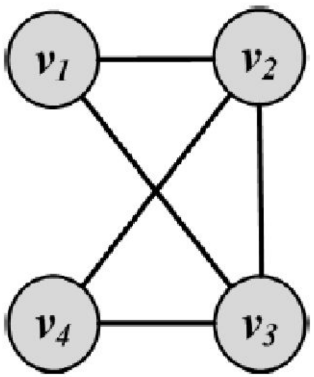


Connectivity

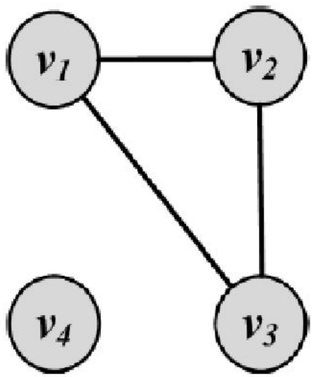
- **A node v_i is connected** to another node v_j (or reachable from v_j) if it is adjacent to it or there exists a path from v_i to v_j.

- **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 is not connected.

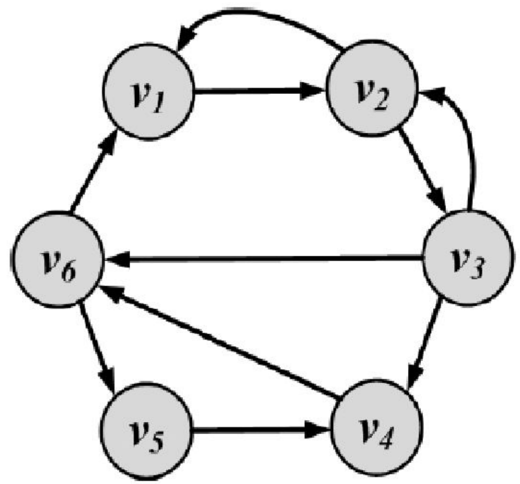
Connectivity: Example



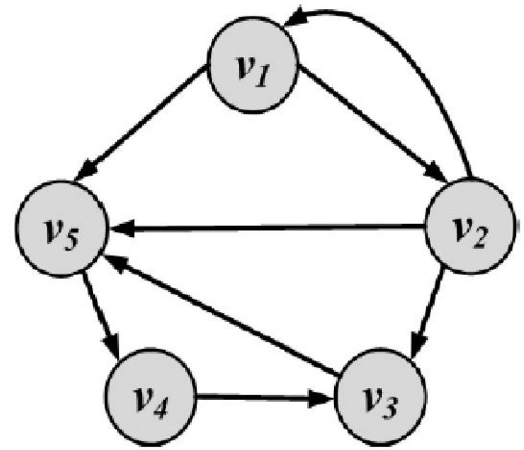
Connected



Disconnected



Strongly connected



Weakly connected

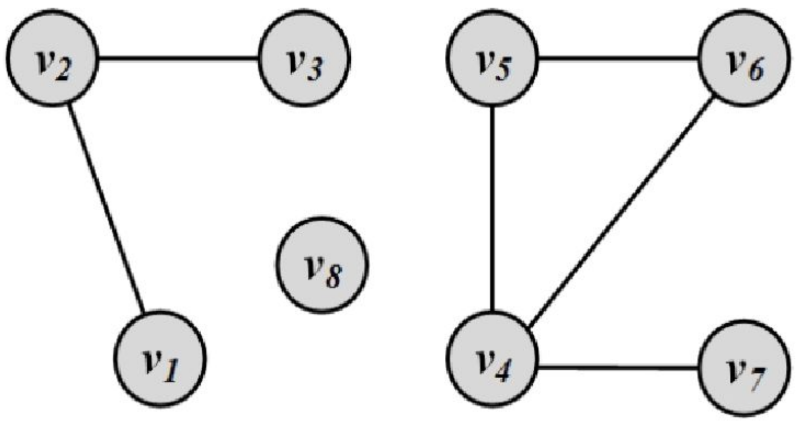
Components

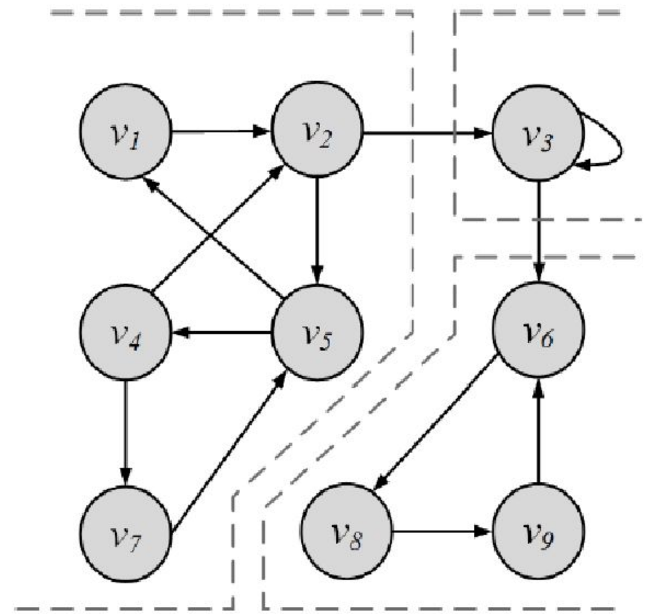
- A **component** in an undirected graph is a connected **subgraph**, i.e., there is a path between every pair of nodes inside the component

- 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 (u,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* pairs of nodes in the graph

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

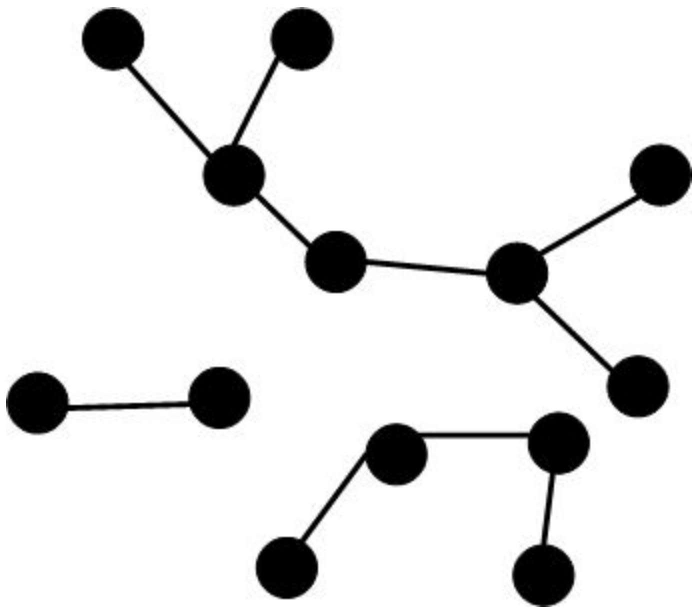
Special Graphs and Subgraphs

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**

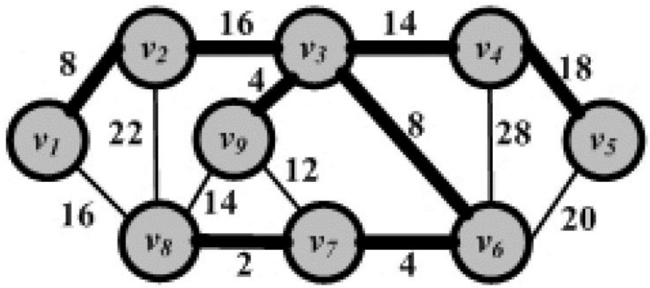


A forest containing 3 trees

Spanning Trees

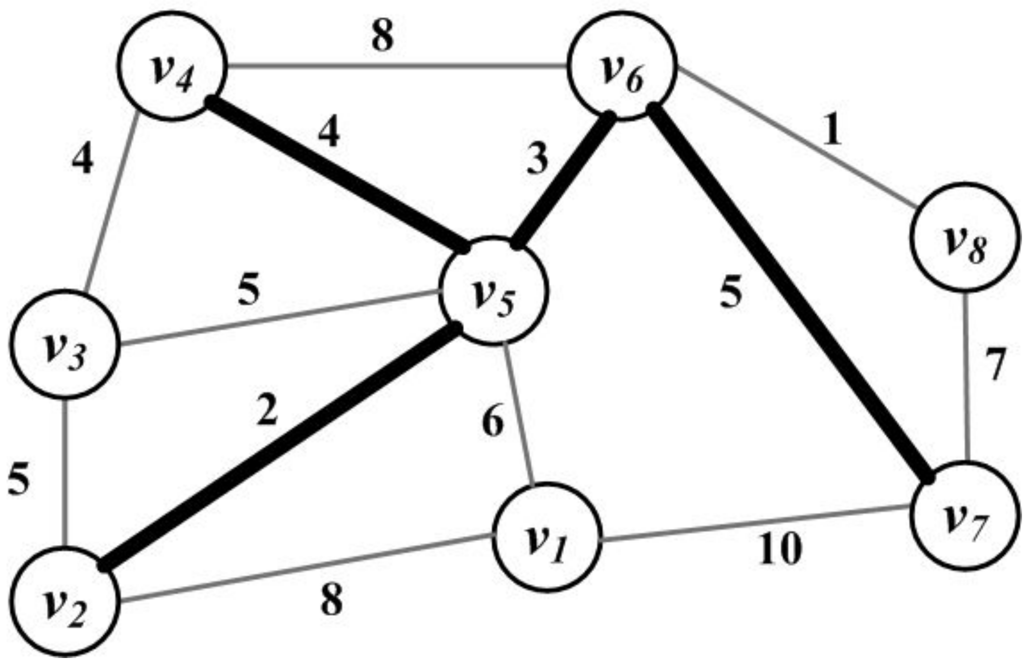
- For any connected graph, the spanning tree is a subgraph and a tree that includes ***all*** the nodes of the graph
- There may exist *multiple* spanning trees for a graph.
- For a weighted graph and one of its spanning tree, the weight of that spanning tree is the summation of the edge weights in the 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

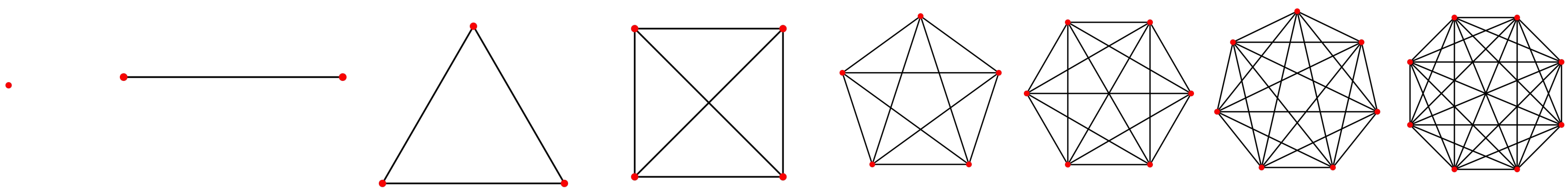
- Given a weighted graph G : (V, E, W) and a ***subset*** of nodes V' $\subseteq$ V (terminal nodes), the Steiner tree problem is to find a tree such that it spans all the V' nodes and the weight of this tree is minimized



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

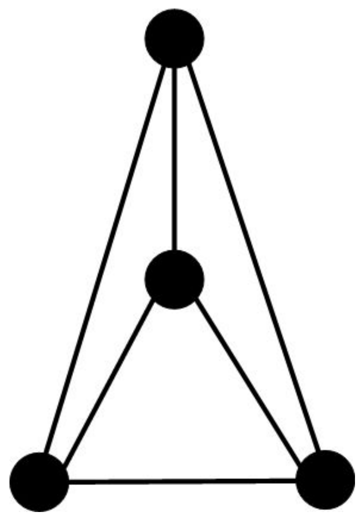
$$E = \binom{|V|}{2}$$



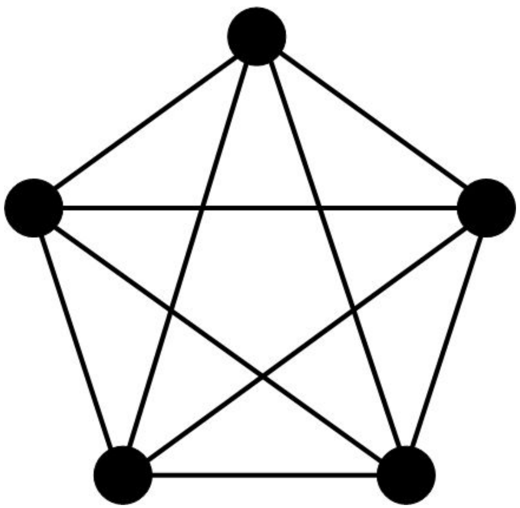
|v|=1 |v|=2 |v|=3 |v|=4 |v|=5 |v|=6 |v|=7 |v|=8

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

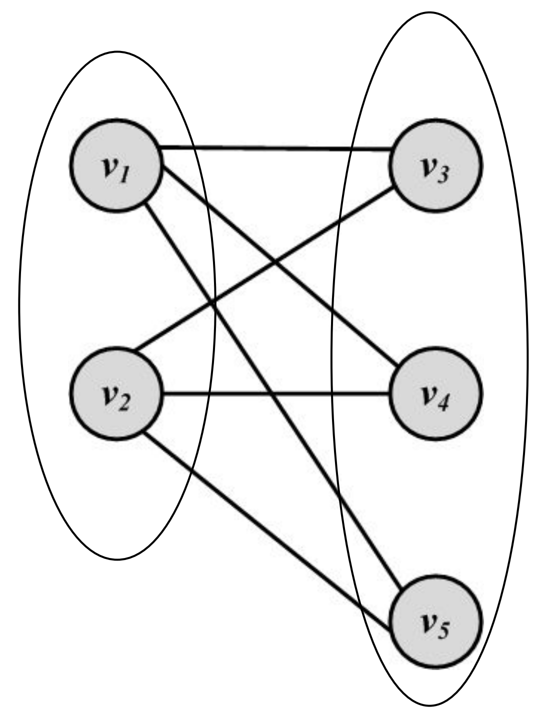


Non-planar Graph

Bipartite Graphs

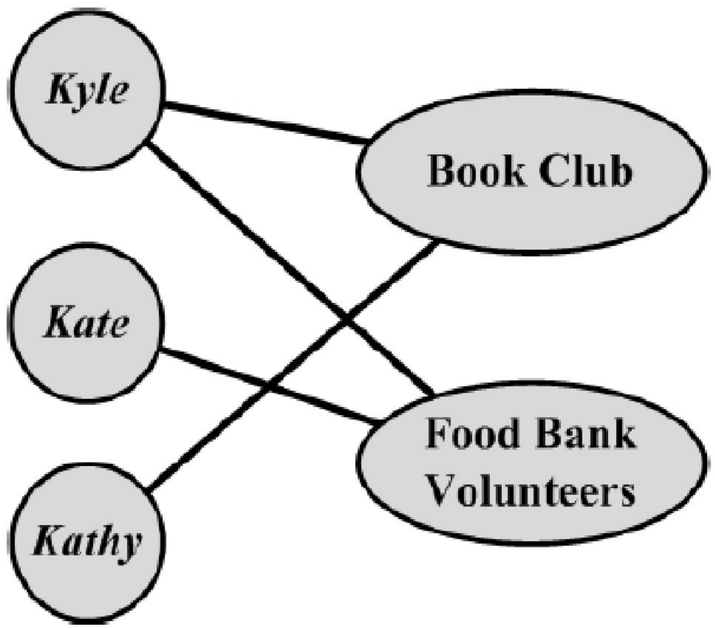
- A bipartite graph G(V; E) is a graph where the node set can be partitioned into **two sets** such that, for all edges, one end-point is in one set and the other end-point is in the other set.

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



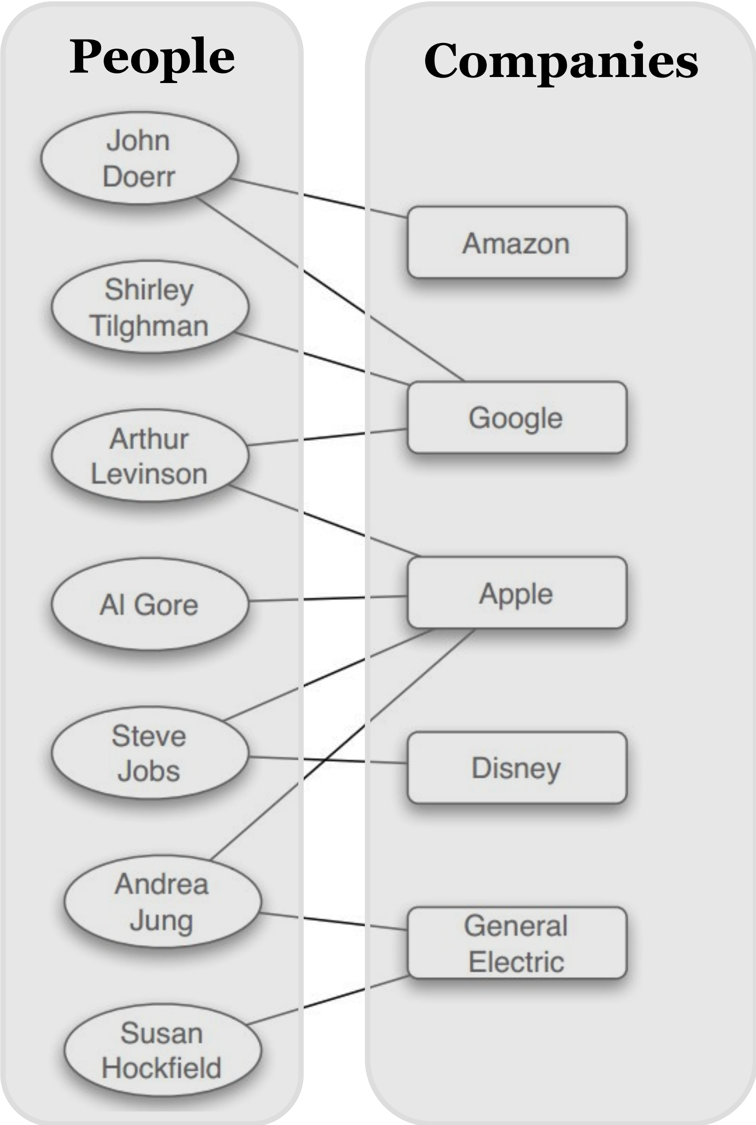
Examples: Affiliation Networks

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



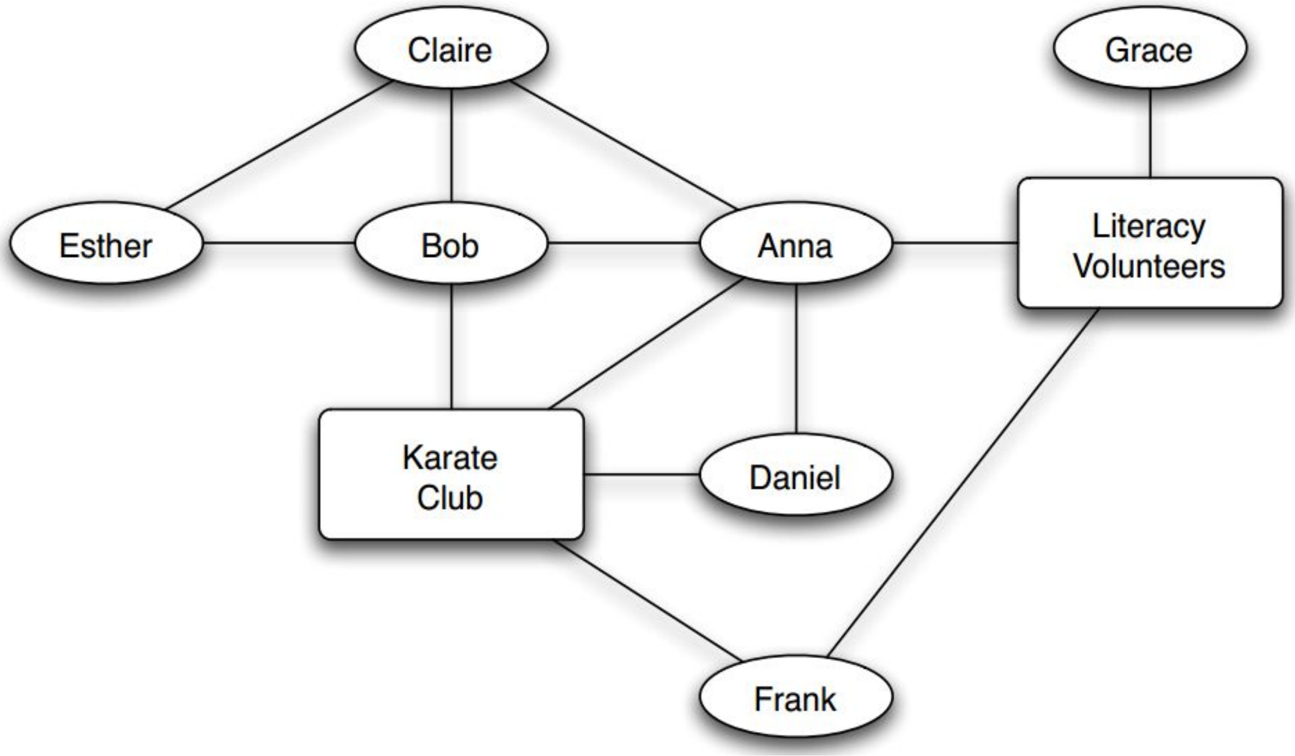
Affiliation Networks: Membership

Affiliation of people on corporate boards of directors



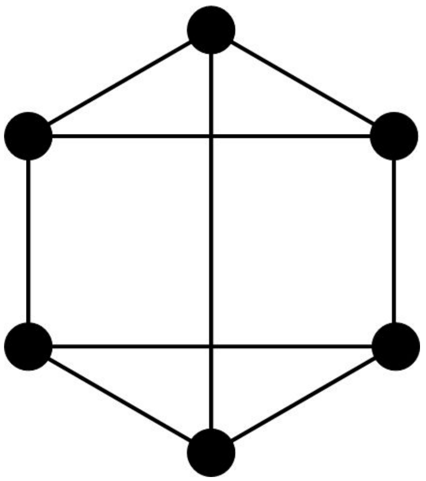
Social-Affiliation Networks

- A 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

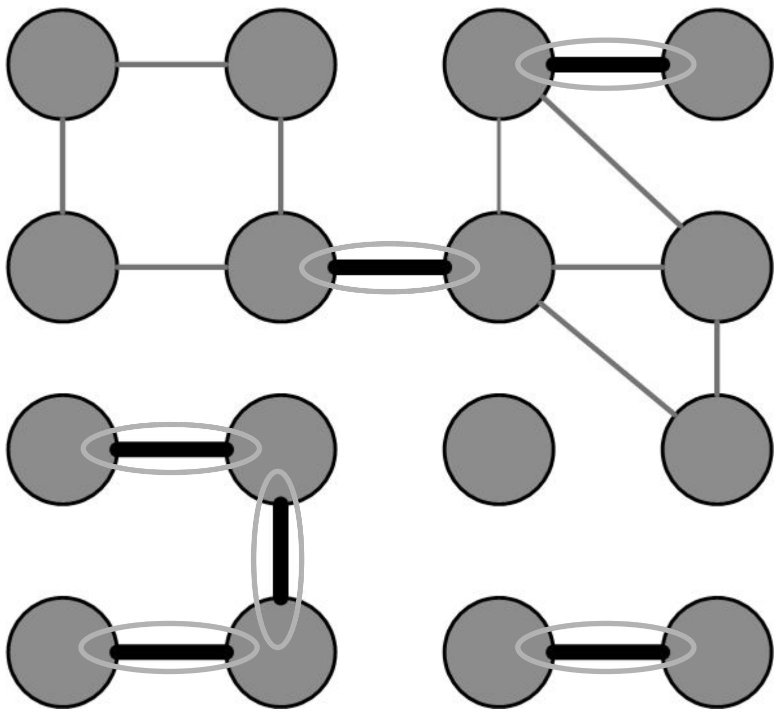


Egocentric Networks

- An **egocentric** network consists of a focal actor (**ego**), and a set of **alters** who have ties with the ego
- In an egocentric network, usually there are limitations for nodes to connect to other nodes or have relation with other nodes
  - For 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/Network Traversal Algorithms

Graph/Tree Traversal

- Consider a social media site that has many users and we are interested in surveying the site and computing the average age of its users. The traversal technique should guarantee 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 form $N(v_i)$ 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>

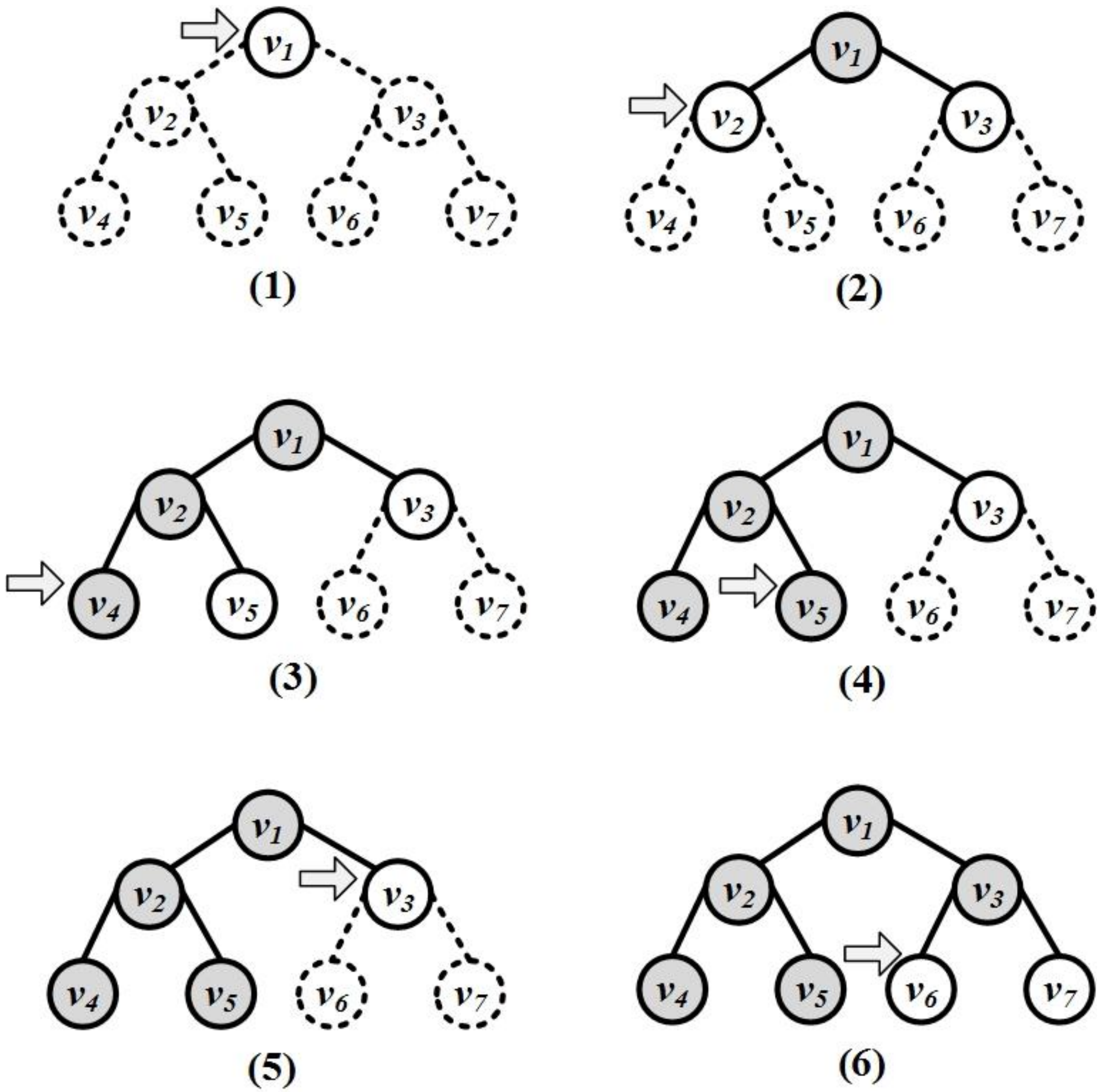
DFS Algorithm

Algorithm 2.2 Depth-First Search (DFS)

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

```
return An ordering on how nodes in G are visited
Push v into S;
visitOrder = 0;
while S not empty do
    node = pop from S;
    if node not visited then
        visitOrder = visitOrder +1;
        Mark node as visited with order visitOrder; //or print node
        Push all neighbors/children of node into S;
    end if
end while
Return all nodes with their visit order.
```

Depth-First Search (DFS): An Example



Breadth-First Search (BFS)

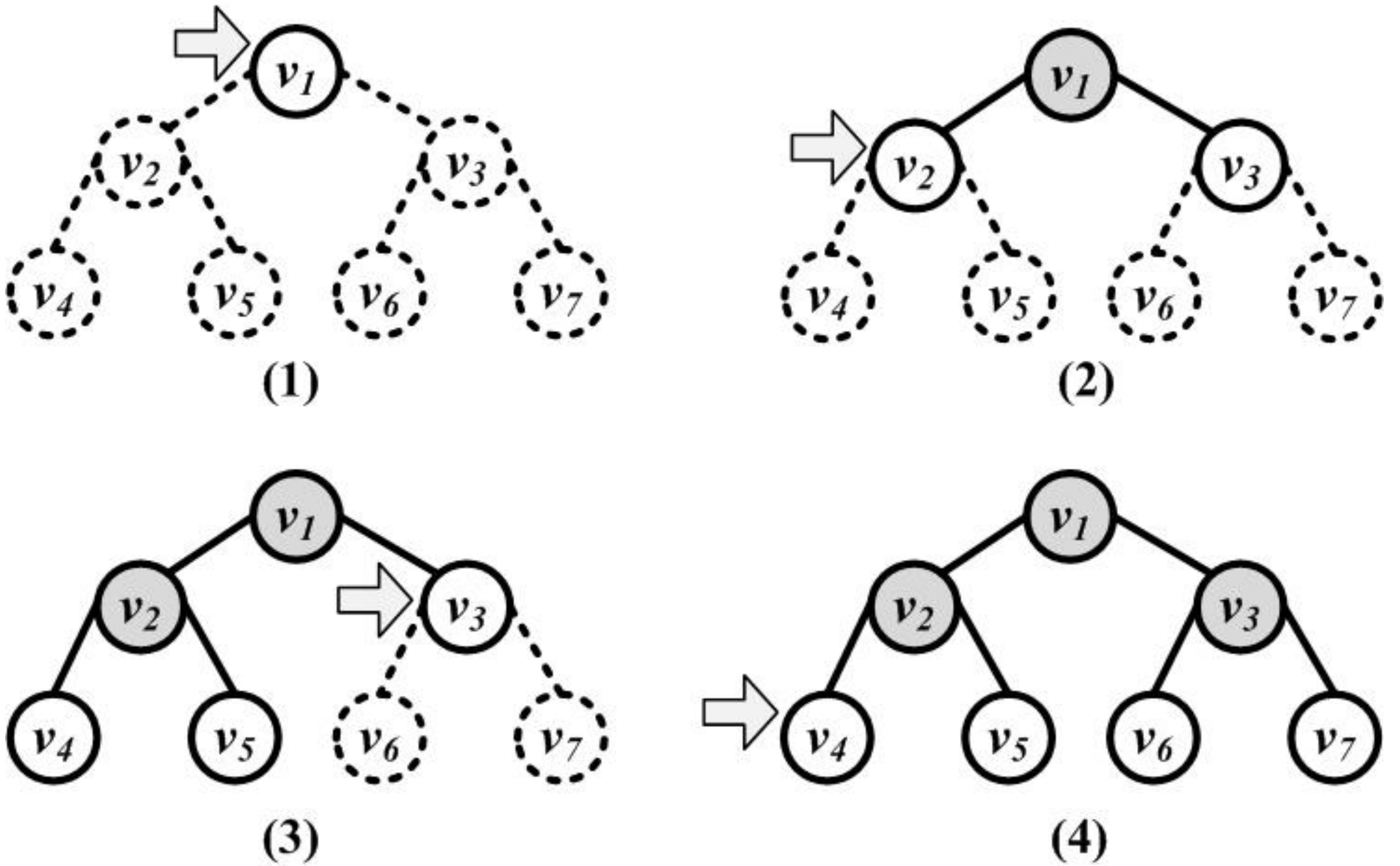
- BFS starts from a node, visits all its **immediate** neighbors first, and then moves to the second level by traversing their neighbors.
- The algorithm can be used both for trees and graphs
  - The algorithm can be implemented using a <u>queue structure</u>

BFS Algorithm

Algorithm 2.3 Breadth-First Search (BFS)

```
Require: Initial node v, graph/tree G(V, E), queue Q
return An ordering on how nodes are visited
Enqueue v into queue Q;
visitOrder = 0;
while Q not empty do
    node = dequeue from Q;
    if node not visited then
        visitOrder = visitOrder +1;
        Mark node as visited with order visitOrder; //or print node
        Enqueue all neighbors/children of node into Q;
    end if
end while
```

Breadth-First Search (BFS)



Shortest Path

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

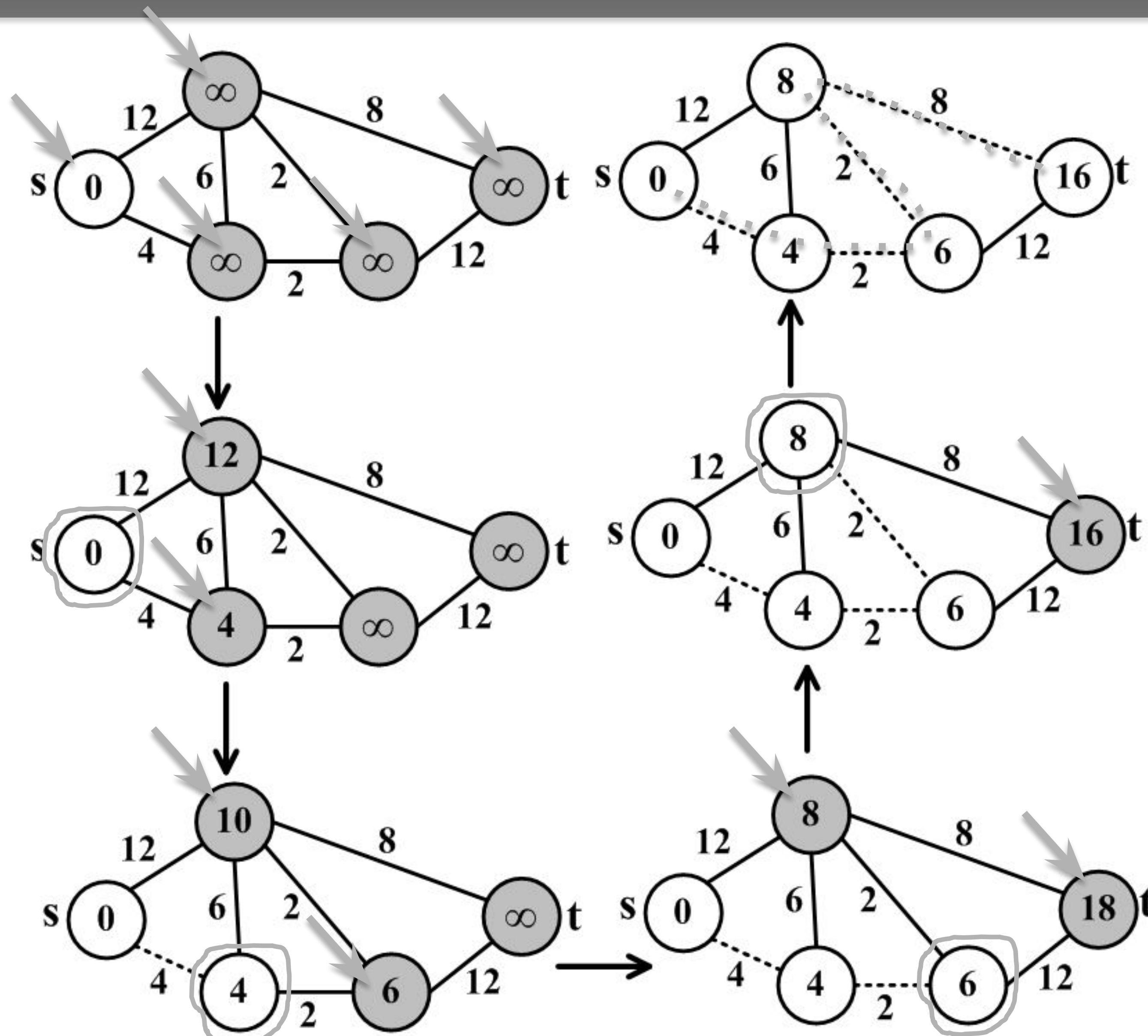
- In many scenarios, we want the shortest path between two nodes in a graph

- **Dijkstra's Algorithm**
  - It is 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 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 (current node's distance + edge weight) 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*
   - A visited node will never be checked again and its distance recorded now is final and minimal
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

Dijkstra's Algorithm Execution Example



Prim’s Algorithm: Finding Minimum Spanning Tree

- Find minimal spanning trees in a weighted graph
  - Start by selecting a random node and adding it to the spanning tree
  - Grow 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)
  - Iterate until the graph is fully spanned

Prim's Algorithm Execution Example