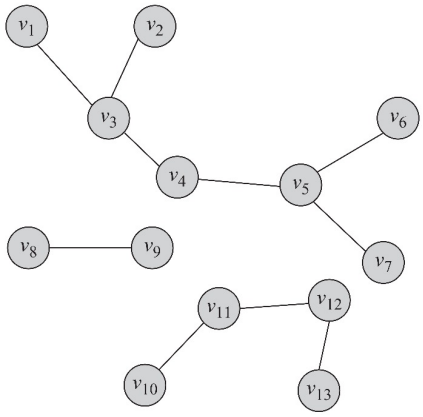
Trees and Forests

- **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**



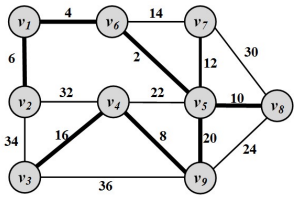
A forest containing 3 trees

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

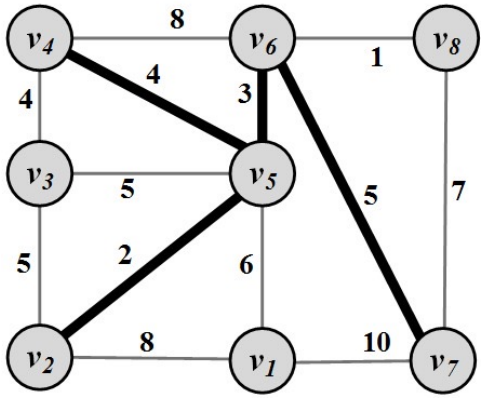
  minimum spanning tree (MST)



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

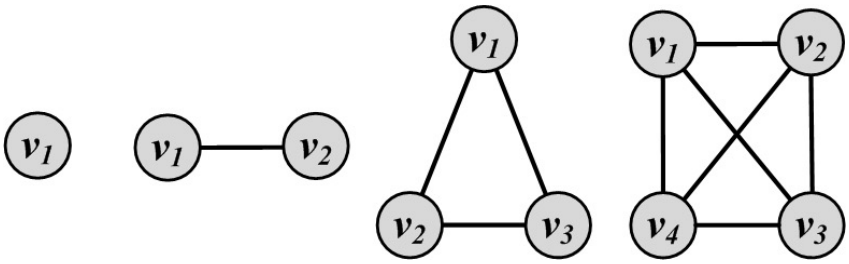
What can be the terminal set here?



Complete Graphs

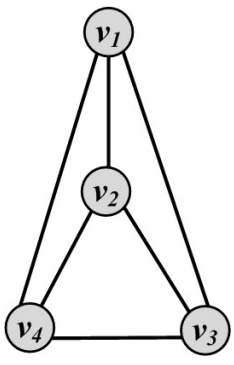
- A complete graph is a graph where for a set of nodes V, all possible edges exist in the graph
- In a complete graph, any pair of nodes are connected via an edge

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

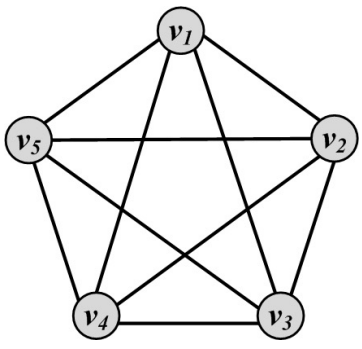


Planar Graphs

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

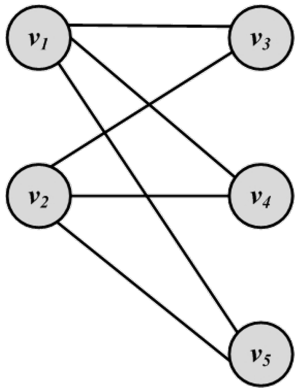


Non-planar Graph

Bipartite Graphs

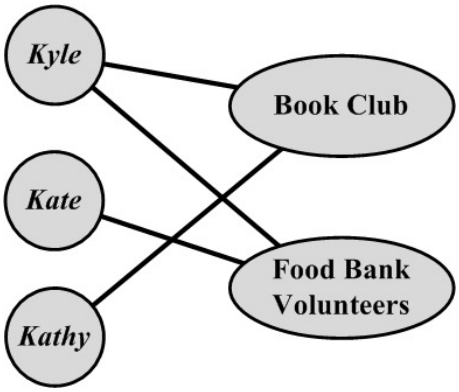
A bipartite graph $G(V, E)$ is a graph where the node set can be partitioned into two sets such that, for all edges, one end-point is in one set and the other end-point is in the other set.

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



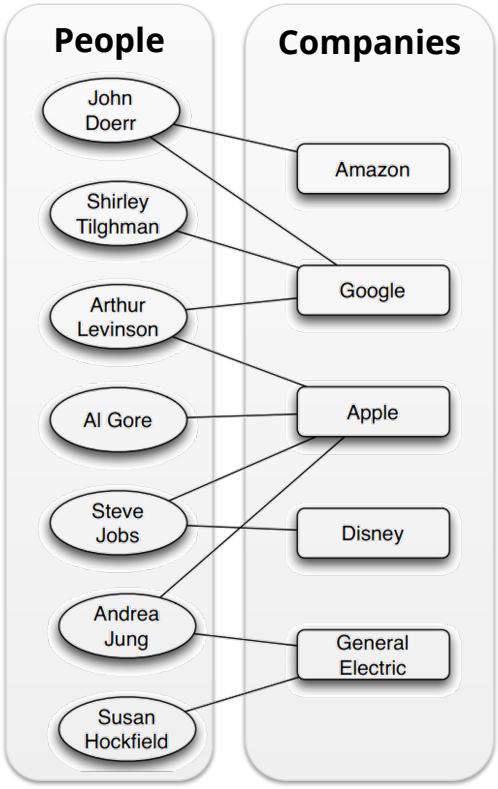
Affiliation Networks

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



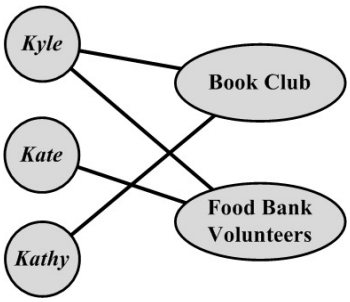
Affiliation Networks: Membership

Affiliation of people on corporate boards of directors



Bipartite Representation / one-mode Projections

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



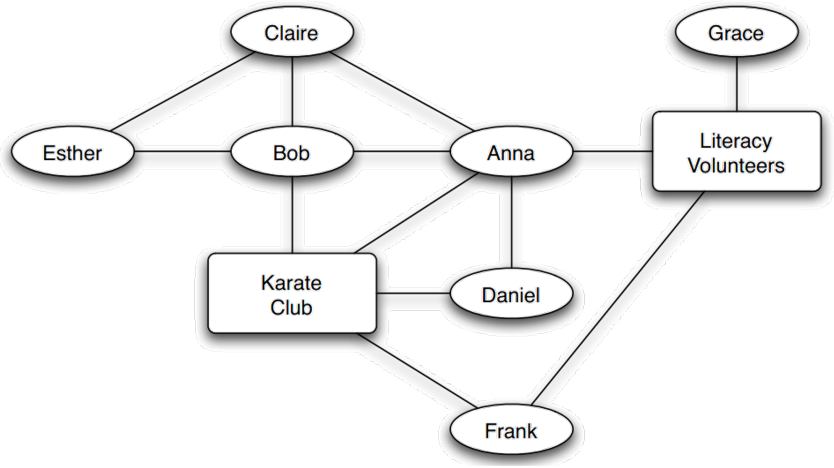
$$X = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- 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

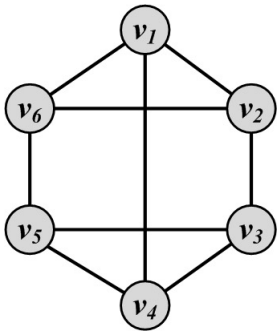
Social-Affiliation Network

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



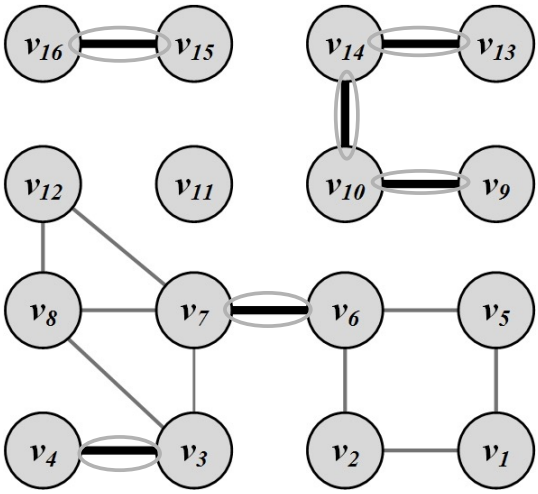
Regular graph
With $k = 3$

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

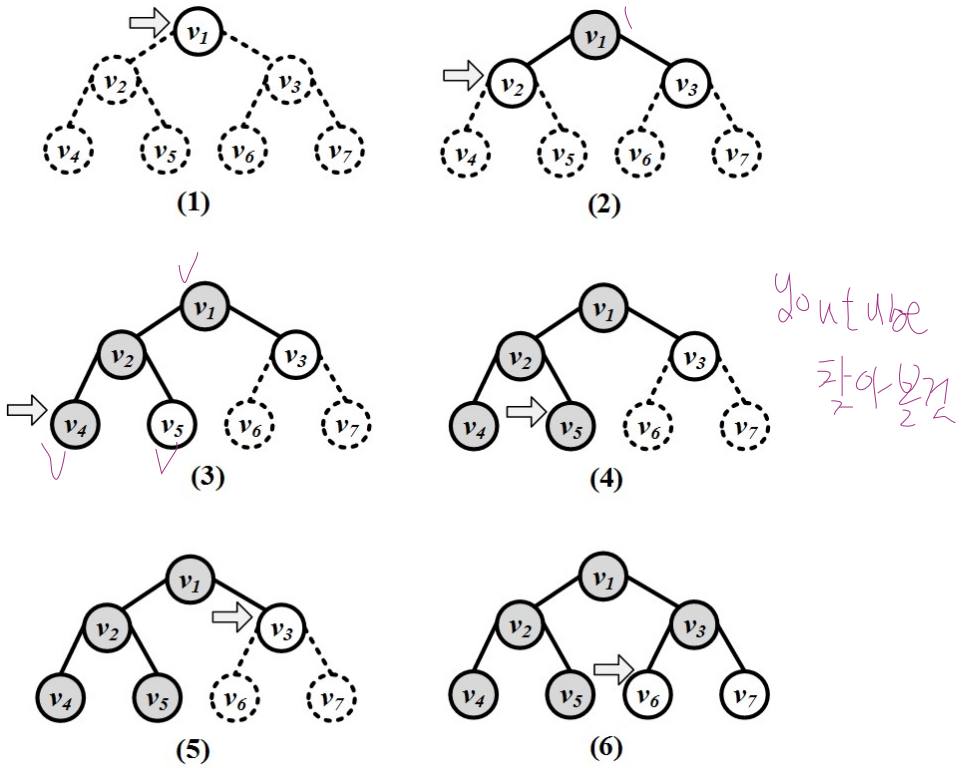
DFS Algorithm

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.
```

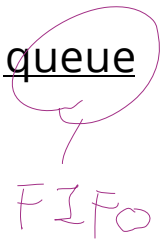

Depth-First Search (DFS): An Example



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



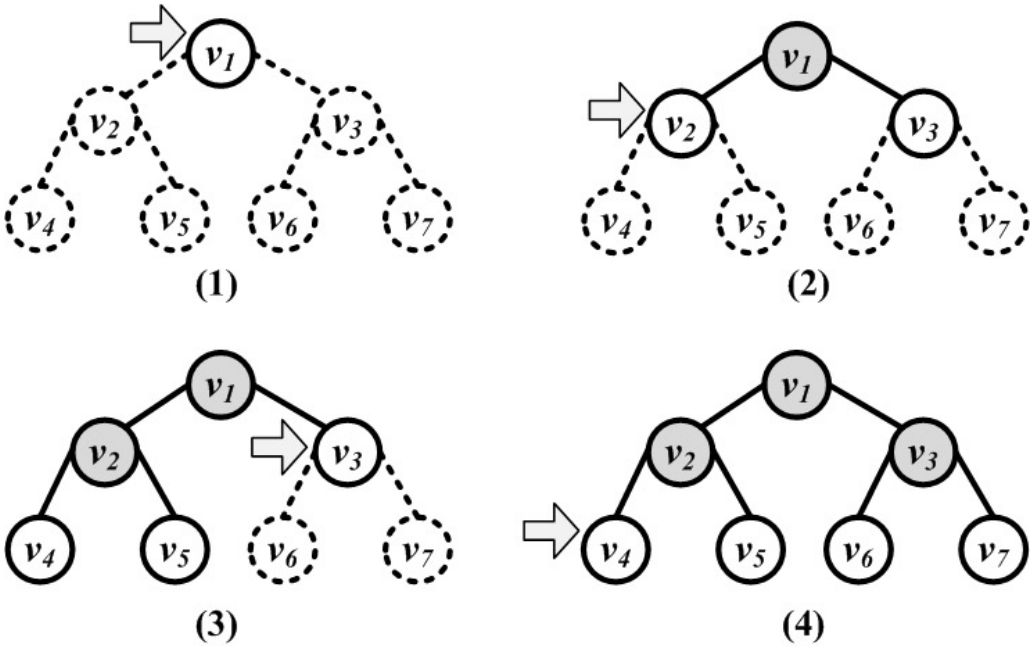
BFS Algorithm

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

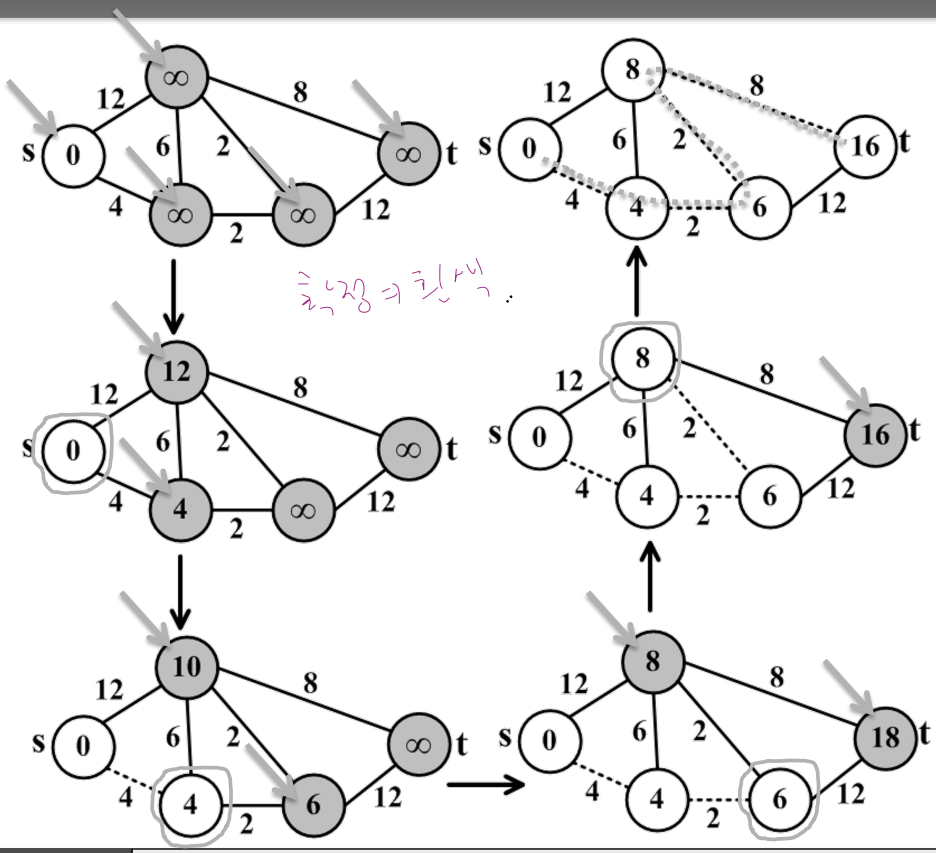
Dijkstra's Algorithm: Finding the shortest path

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
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

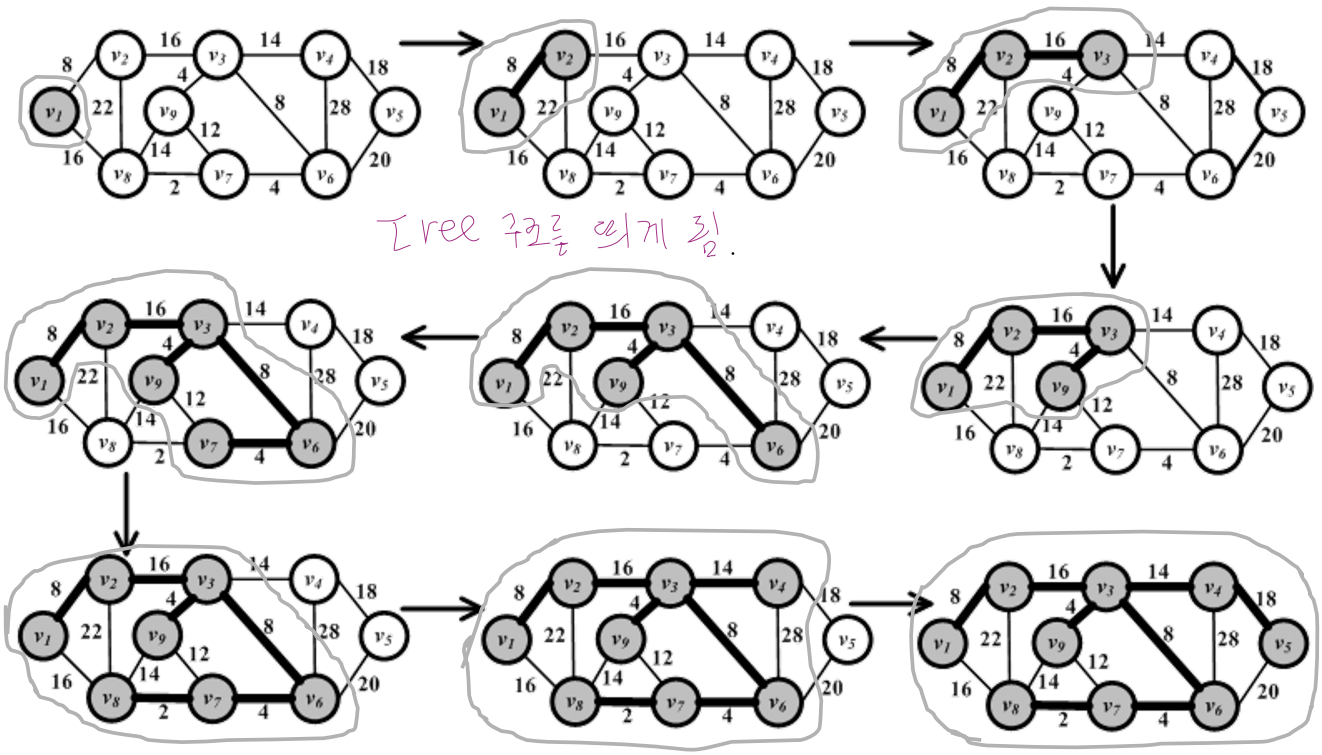
Finding Minimum Spanning Tree

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



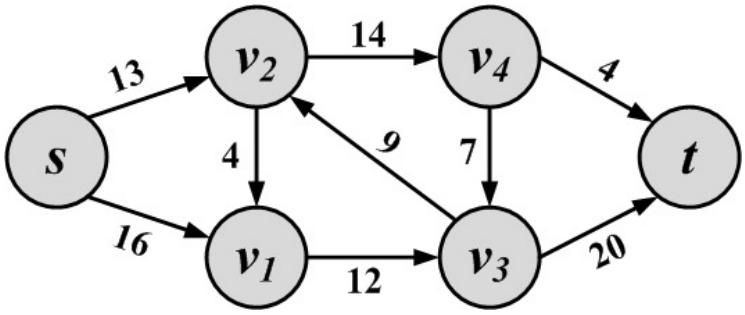
Network Flow

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

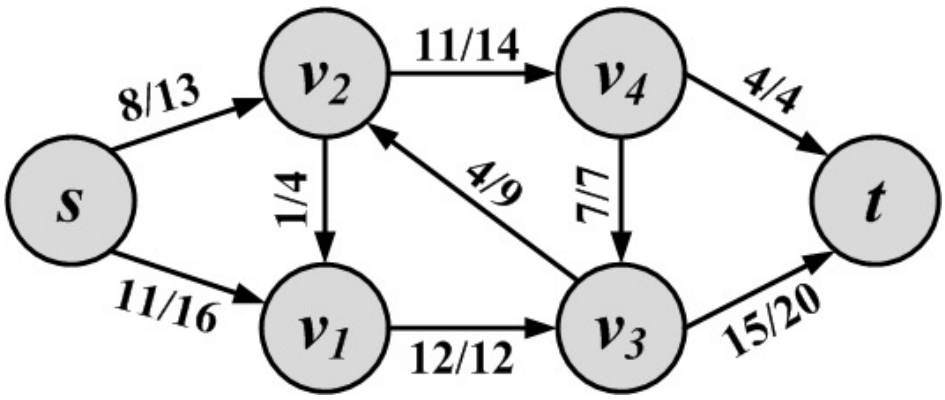
- Given edges with certain capacities, we can fill these edges with the flow up to their capacities (*capacity constraint*)
- The flow that enters any node other than source s and sink t is equal to the flow that exits it so that no flow is lost (*flow conservation constraint*)

- $\forall (u,v) \in E, f(u,v) \geq 0$ 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.

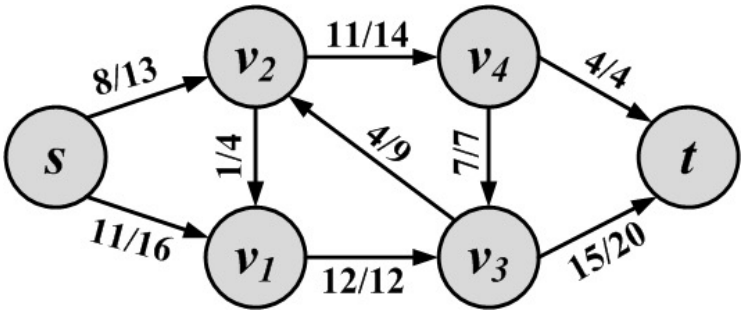


Flow Quantity

- 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

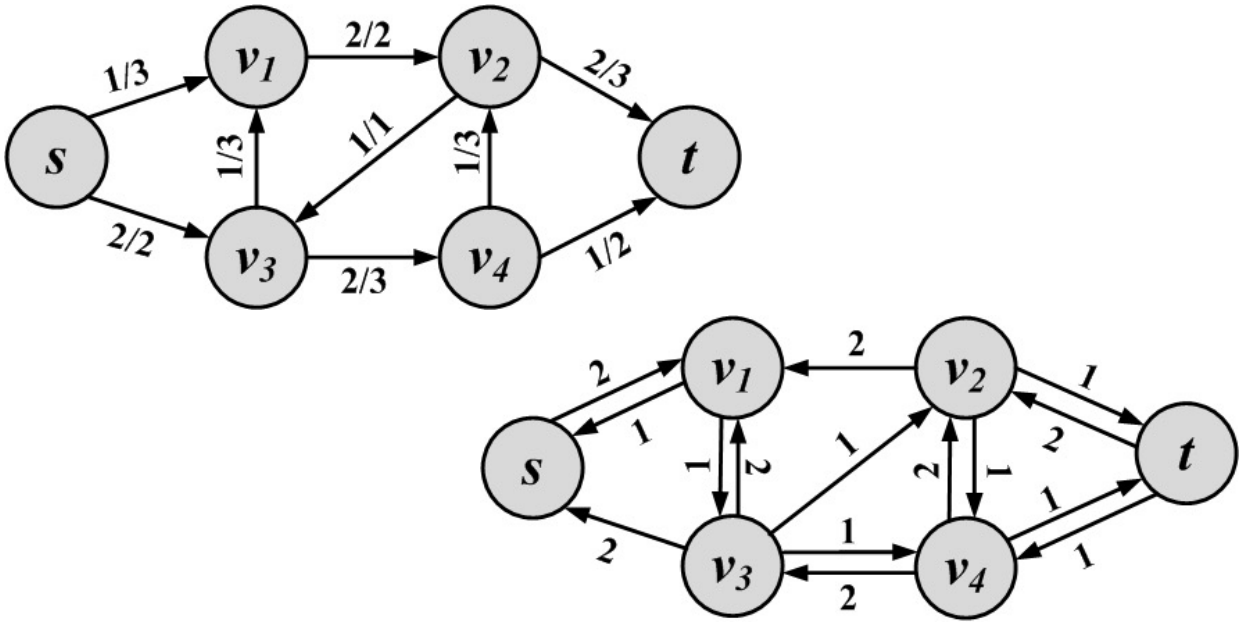
- When there is no flow going through an edge in the original network, a flow of as much as the capacity of the edge remains in the residual.

- 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.

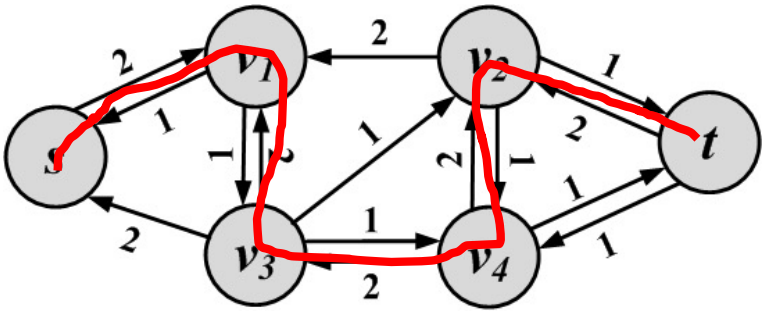
Augmentation / Augmenting Paths

- 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?

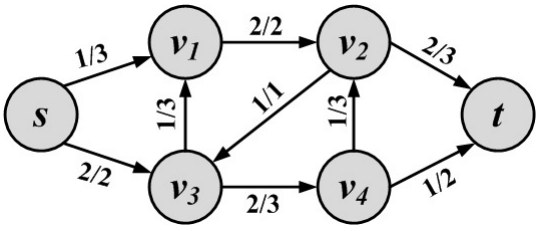
- Given flow $f\ (u, v)$ in the original graph and flow $f_R(u, v)$ and $f_R(v, u)$ in the residual graph, we can augment the flow as follows:

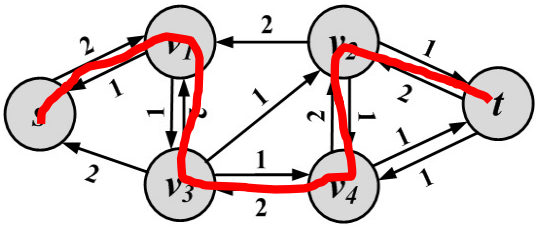
$$f_{augmented}(u, v) = f(u, v) + f_R(u, v) - f_R(v, u)$$

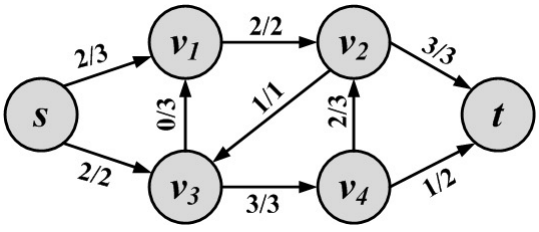


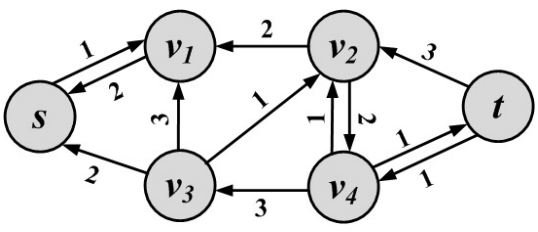
Flow Quantity: 1

Augmenting









The Ford-Fulkerson Algorithm

Algorithm 2.6 Ford-Fulkerson Algorithm

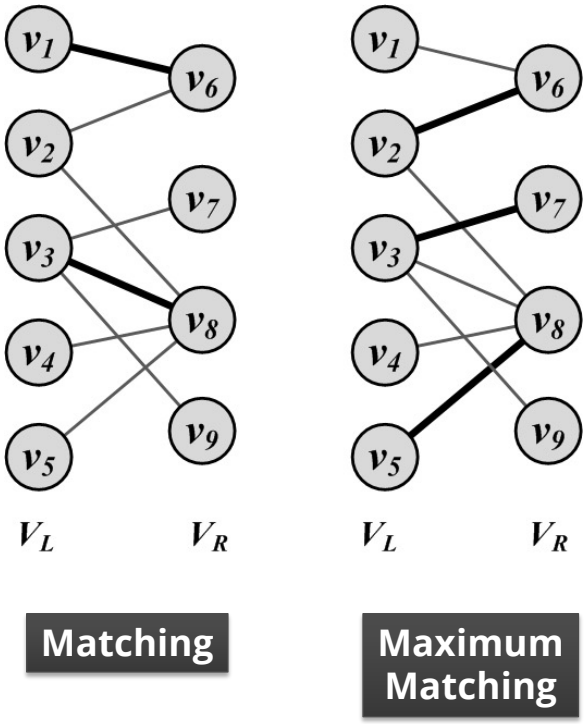
Require: Connected weighted graph $G(V, E, W)$, Source s, Sink t

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

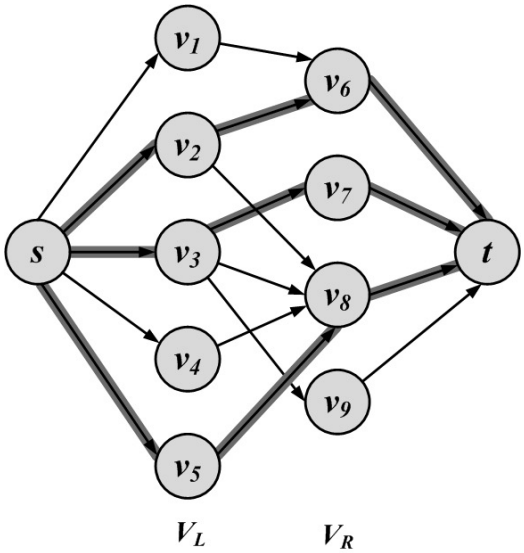


Matching **Maximum Matching**

Matching Solved with Max-Flow

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

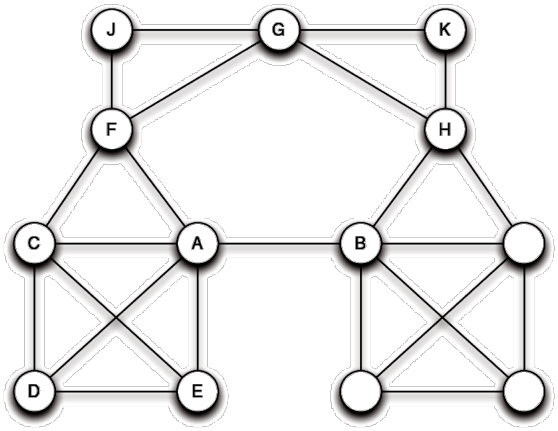
1. Set $V' = V \cup \{s\} \cup \{t\}$
2. Connect all nodes in V_L to s and all nodes in V_R to t
3. Set $c(u, v) = 1$, for all edges in E'



Bridges, Weak Ties, and Bridge Detection

Bridge and a Local Bridge

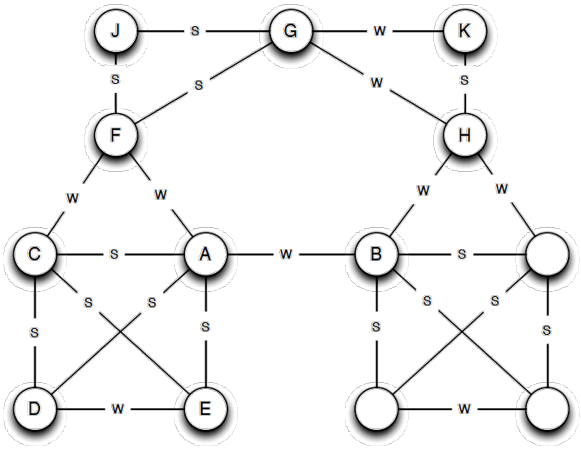
- **Bridge:** Bridges are edges whose removal will increase the number of connected components
  - Bridges are extremely rare in real-world social networks.
- **Local Bridge:** when the endpoints have no friend in common
  - the removal increases the length of shortest path to more than 2
  - **Span 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

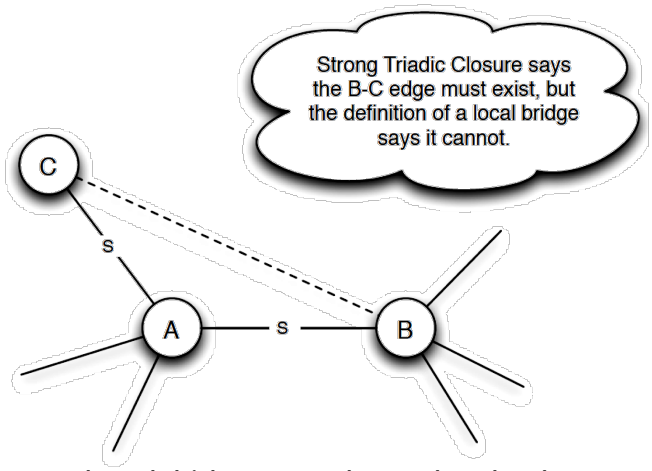


- **Strong Triadic Closure:**
  - Consider a node u that has two strong ties to nodes v and w
  - If there is no edge between v and w (weak or strong tie) then u does not exhibit a strong triadic closure

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

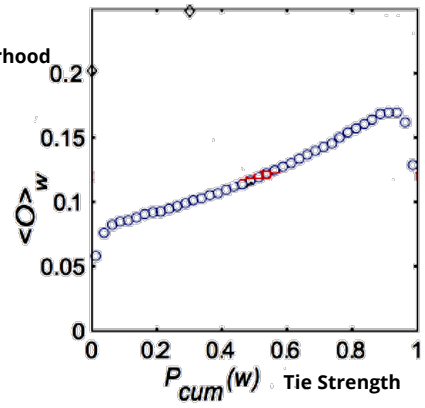
Why?



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:*



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

The numerator is called embeddedness of an edge

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
```