Network Basics 1

Graph ML

Department of Computer Science
University of Massachusetts, Lowell

Hadi Amiri
hadi@cs.uml.edu

Lecture Topics

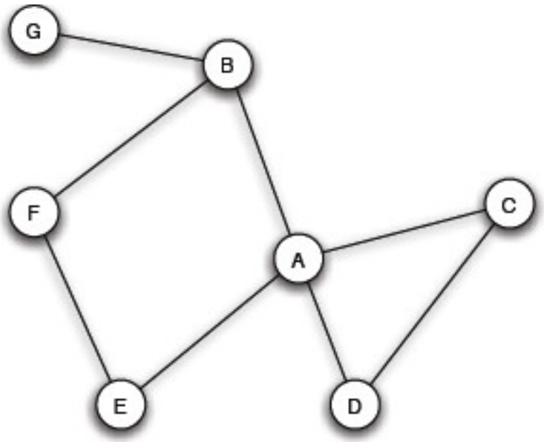
- Graph Theory
    - Node degree
    - Graph density
    - Complete Graph
    - Distance and Diameter
    - Adjacency matrix
    - Graph Connectivity
    - Reachability
    - Sub-graphs
    - Graph Types

Graph Theory

- A graph consists of
  - **N**: a set of nodes (items, entities, people, etc), and
  - **E**: a set of links or edges between nodes

- Graph is a way to **specify relationships** / links amongst a set of nodes.

- We define
  - N=|**N**| → size of **N**
  - E=|**E**| → size of **E**



Graph Theory. Cnt.

- Nodes *i* and *j* are *adjacent* or *neighbors* if:
  - There is an edge btw them!
    - $i \in \mathbf{N}$
    - $j \in \mathbf{N}$
    - $(i, j) \in \mathbf{E}$

Sample Graphs 1.

- “Lives Near” Graph

nodes

	Actor	Lives near:
n_1	Allison	Ross, Sarah
n_2	Drew	Eliot
n_3	Eliot	Drew
n_4	Keith	Ross, Sarah
n_5	Ross	Allison, Keith, Sarah
n_6	Sarah	Allison, Keith, Ross

Links or edges

$l_1 = (n_1, n_5)$

$l_2 = (n_1, n_6)$

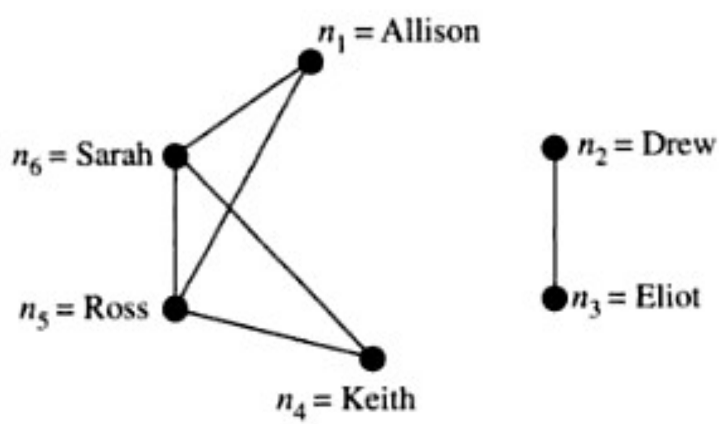
$l_3 = (n_2, n_3)$

$l_4 = (n_4, n_5)$

$l_5 = (n_4, n_6)$

$l_6 = (n_5, n_6)$

Graph



Source: Social network analysis: Methods and applications. Wasserman, Stanley. Cambridge university press, 1994.

Node Degree *d*(*i*)

- Given Node *i*, its degree *d*(*i*) is:
  - the number nodes adjacent to it.

	Actor	Lives near:	Degree
n_1	Allison	Ross, Sarah	2
n_2	Drew	Eliot	1
n_3	Eliot	Drew	1
n_4	Keith	Ross, Sarah	2
n_5	Ross	Allison, Keith, Sarah	3
n_6	Sarah	Allison, Keith, Ross	3

$l_1 = (n_1, n_5)$

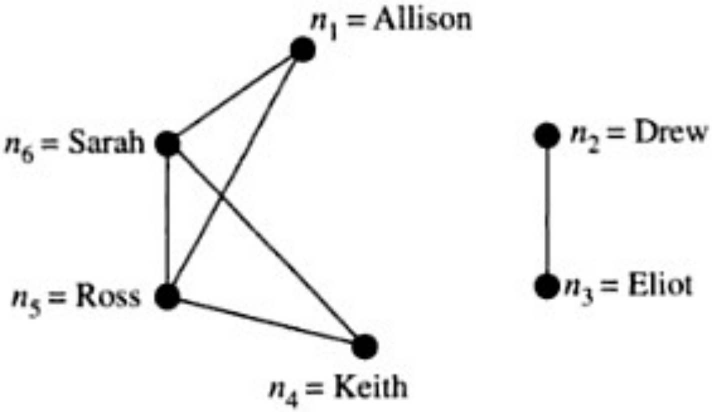
$l_2 = (n_1, n_6)$

$l_3 = (n_2, n_3)$

$l_4 = (n_4, n_5)$

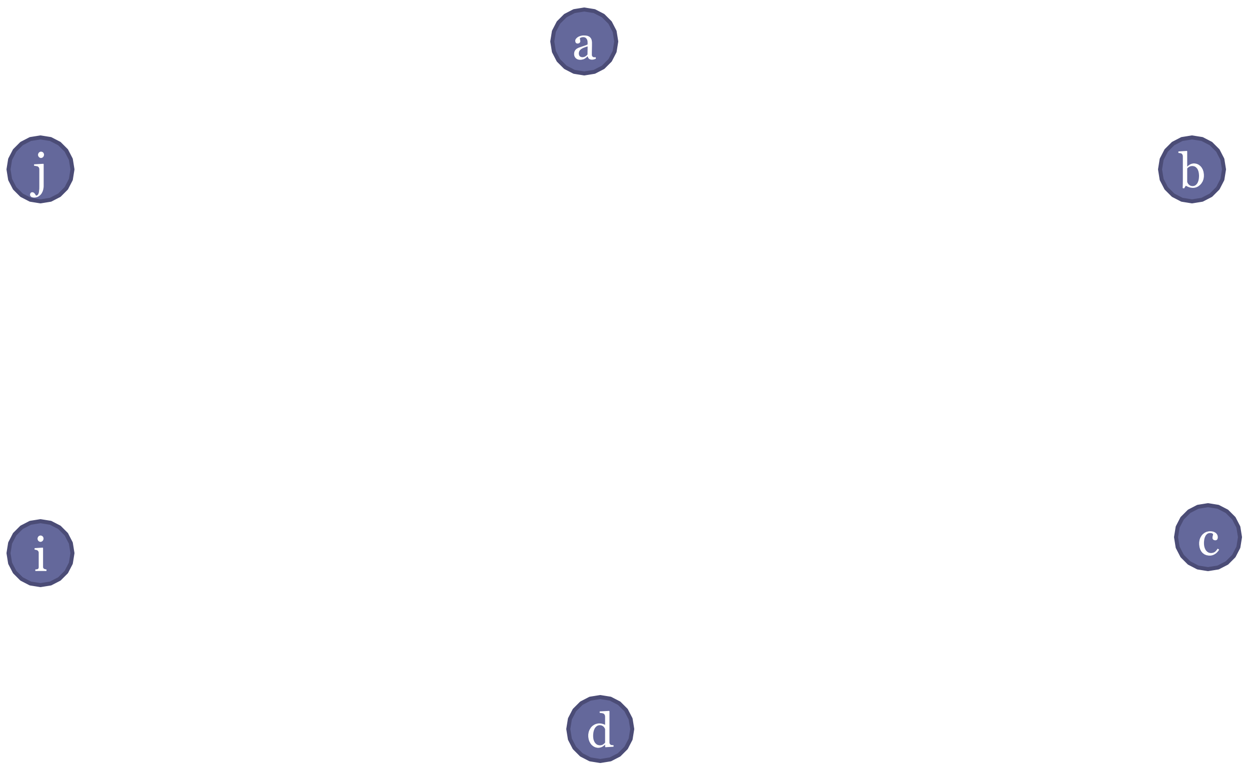
$l_5 = (n_4, n_6)$

$l_6 = (n_5, n_6)$



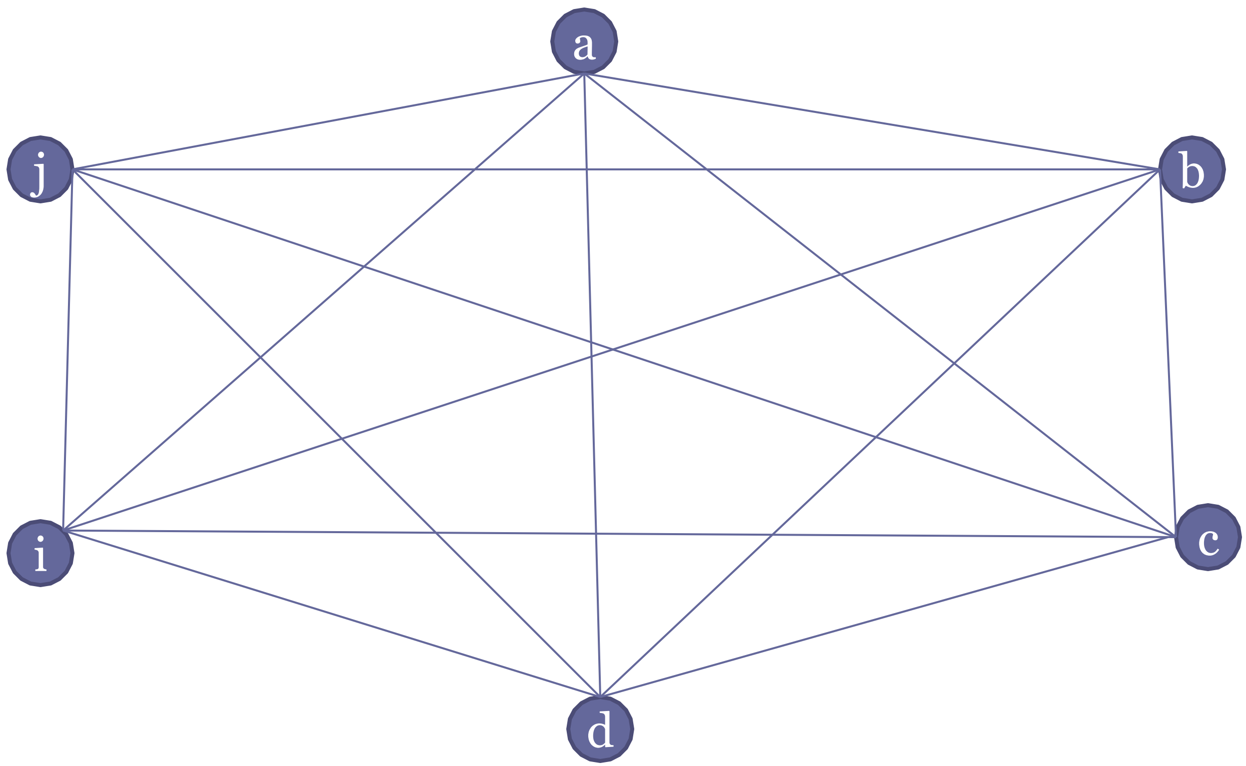
Graph Density

- How many edges are possible?



Graph Density- Cnt.

- $(N-1) + (N-2) + (N-3) + \ldots + 1 = N * (N-1) / 2$



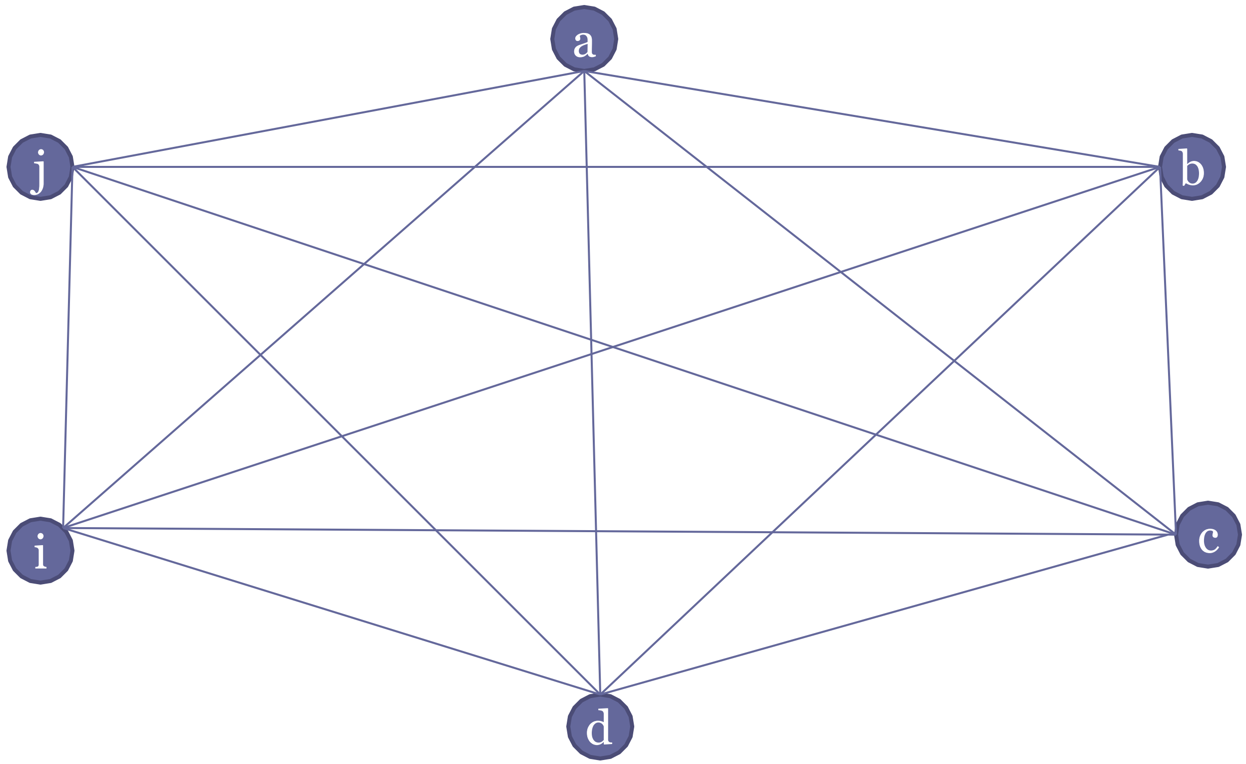
Graph Density- Cnt.

- Graph Density of a given graph G is determined by:
  - the proportion of all possible edges that are present in the graph.
  - with N nodes and E edges, graph density is:

$$\text{Density} = 2 * E / N * (N-1)$$

Complete Graph

- If all edges are present, then all nodes are adjacent (neighbors), and the graph is a *Complete Graph*.



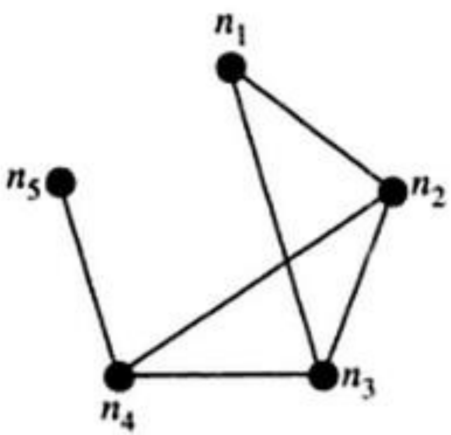
What is the density of a complete graph?

Distance and Diameter

- Distance btw node i and j: $d(i,j)$
  - length of the ***<u>shortest path</u>*** between i and j
- Diameter of a graph
  - the maximum value of $d(i,j)$ for all i and j

The path with min number of edges.

Distance and Diameter- Cnt.



distance

$d(1,2) = 1$

$d(1,3) = 1$

$d(1,4) = 2$

$d(1,5) = 3$

$d(2,3) = 1$

$d(2,4) = 1$

$d(2,5) = 2$

$d(3,4) = 1$

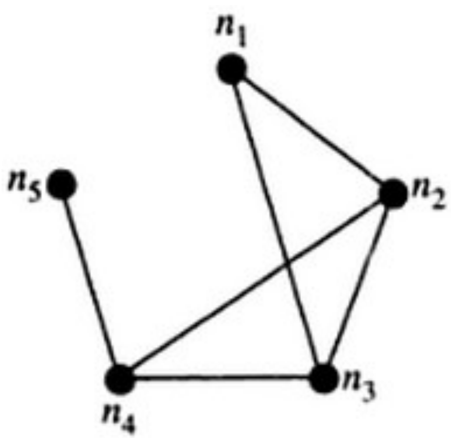
$d(3,5) = 2$

$d(4,5) = 1$

Diameter of graph $= \max d(i,j) = d(1,5) = 3$

What is the distance and diameter of a complete graph?

Adjacency Matrix



$$A = \begin{array}{c|c} & \begin{array}{ccccc} n_1 & n_2 & n_3 & n_4 & n_5 \end{array} \\ \begin{array}{c} n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

- Each row or column represents a node!

$A = A^T$

Properties of adjacency matrix → next session

Graph Connectivity

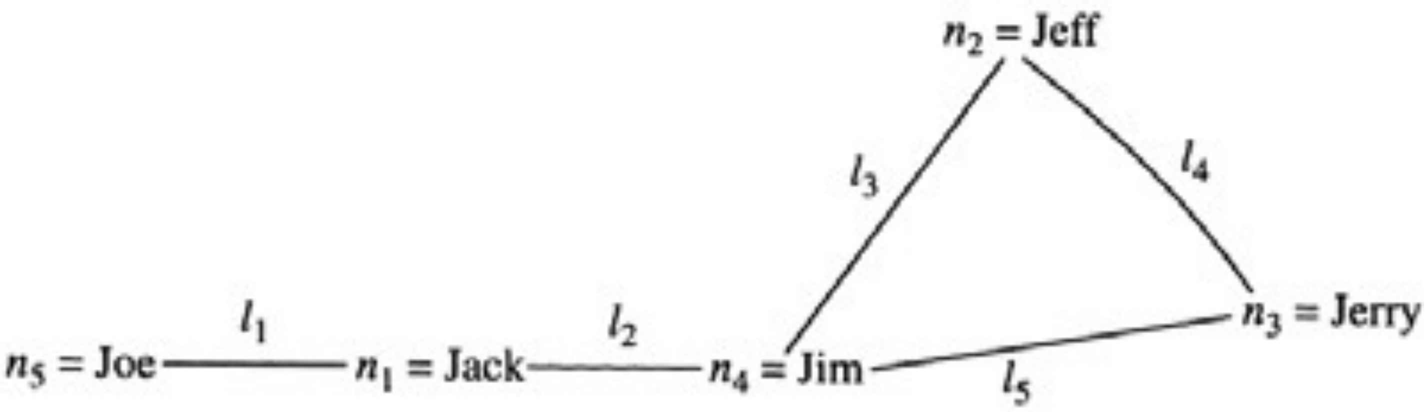
- Indirect connections between nodes:
  - Walks
  - Trails
  - Paths

Graph Connectivity- Cnt.

- Walk
  - A sequence of nodes and edges that starts and ends with nodes where each node is incident to the edges following and preceding it.
- Trail
  - A trail is a walk with distinct edges
- Path
  - A path is a walk with distinct nodes & edges.
- The length of a walk, trail, or path is the number of edges in it.

Graph Connectivity- Cnt.

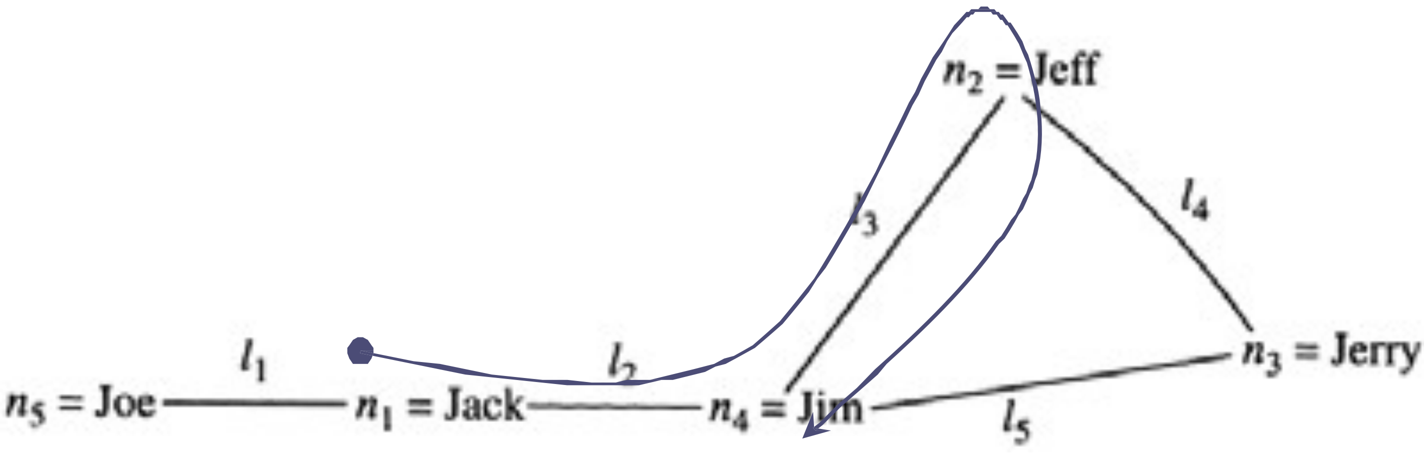
- Walk
  - A sequence of nodes and edges that starts and ends with nodes where each node is incident to the edges following and preceding it.



Source: Social network analysis: Methods and applications. Wasserman, Stanley. Cambridge university press, 1994.

Graph Connectivity- Cnt.

- Walk
  - A sequence of nodes and edges that starts and ends with nodes where each node is incident to the edges following and preceding it.

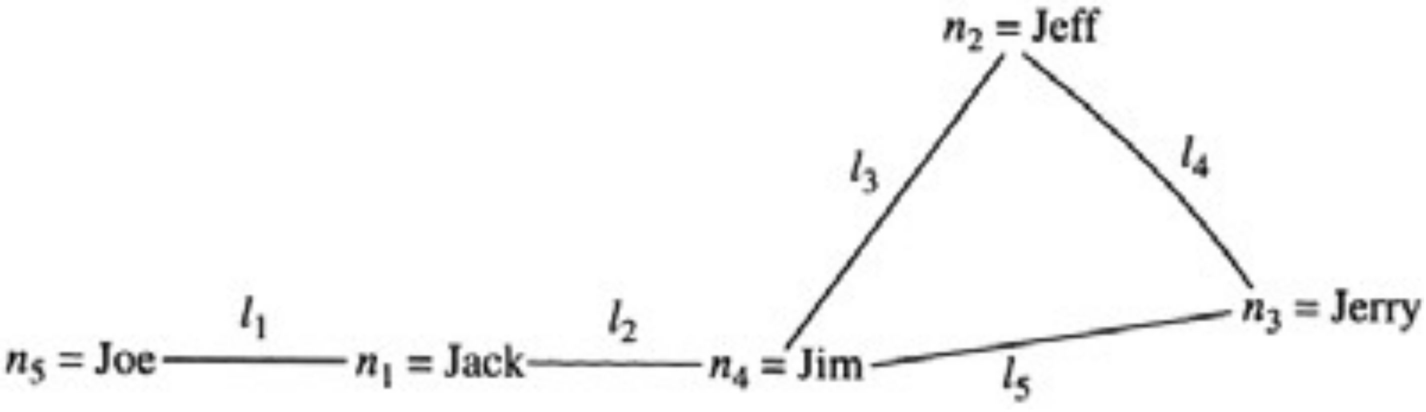


Sample Walk:

$W = n_1\, l_2\, n_4\, l_3\, n_2\, l_3\ \ n_4$

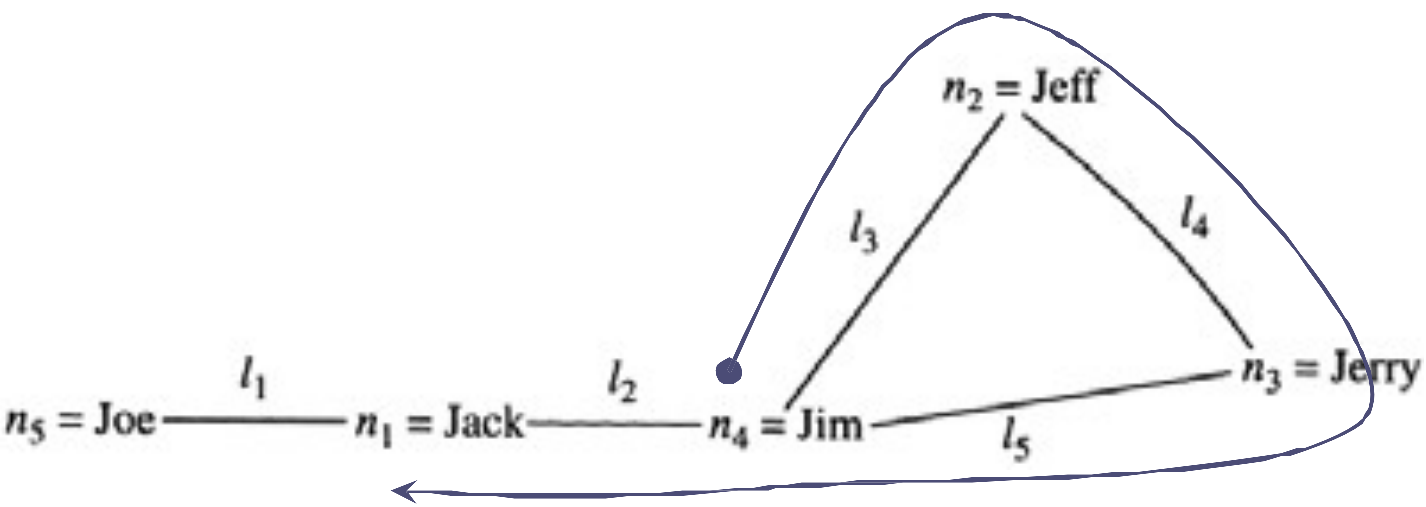
Graph Connectivity- Cnt.

- Trail
  - A trail is a walk in which all edges are distinct, although some node(s) may be included more than once.



Graph Connectivity- Cnt.

- Trail
  - A trail is a walk in which all edges are distinct, although some node(s) may be included more than once.

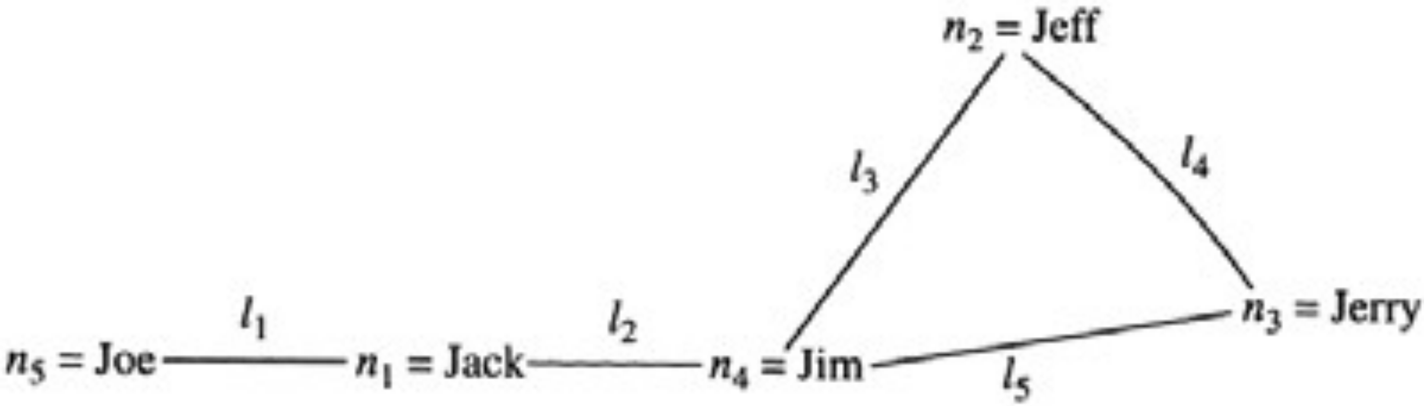


Sample Trail:

$T = n_4\, l_3\, n_2\, l_4\, n_3\, l_5\, n_4\, l_2\, n_1$

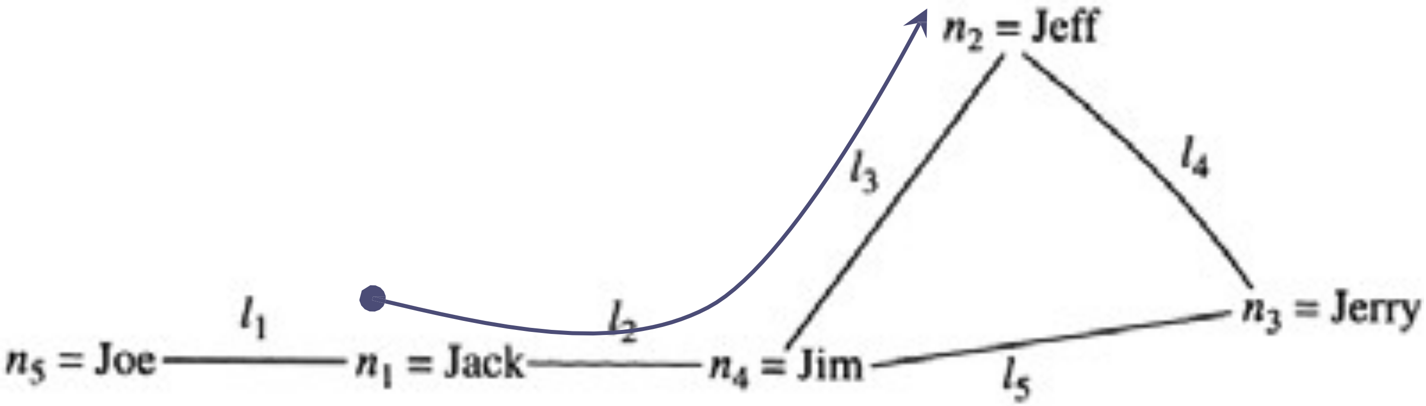
Graph Connectivity- Cnt.

- Path
  - A path is a walk in which all nodes and all edges are distinct.



Graph Connectivity- Cnt.

- Path
  - A path is a walk in which all nodes and all edges are distinct.

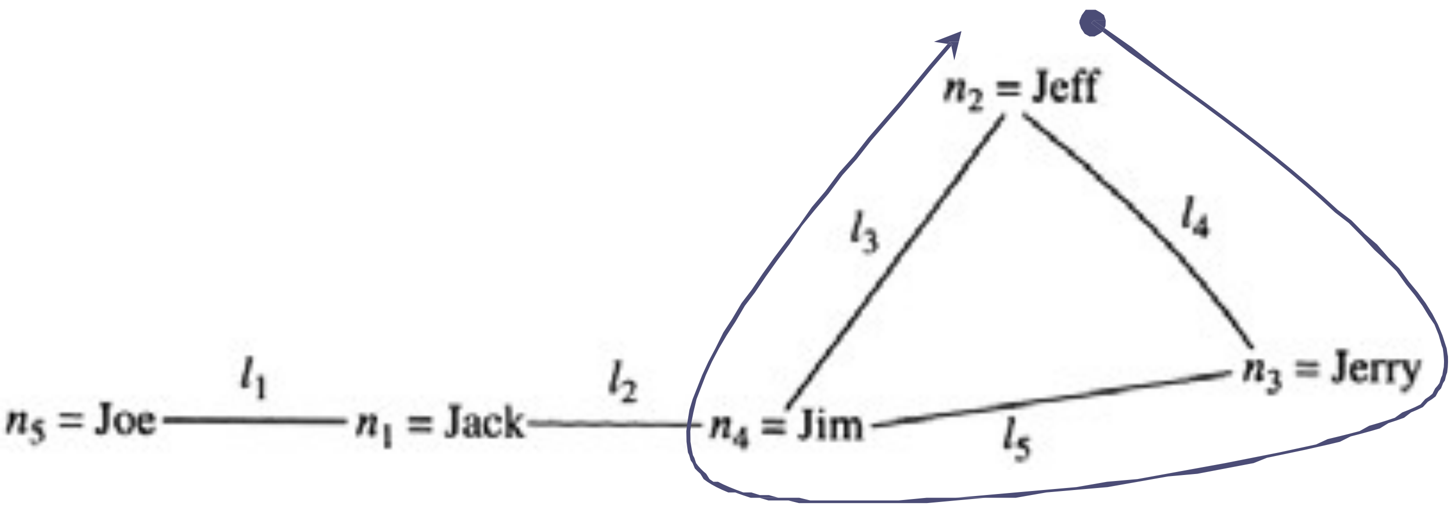


Sample Path:

$P = n_1 l_2 n_4 l_3 \ n_2$

Graph Connectivity- Cnt.

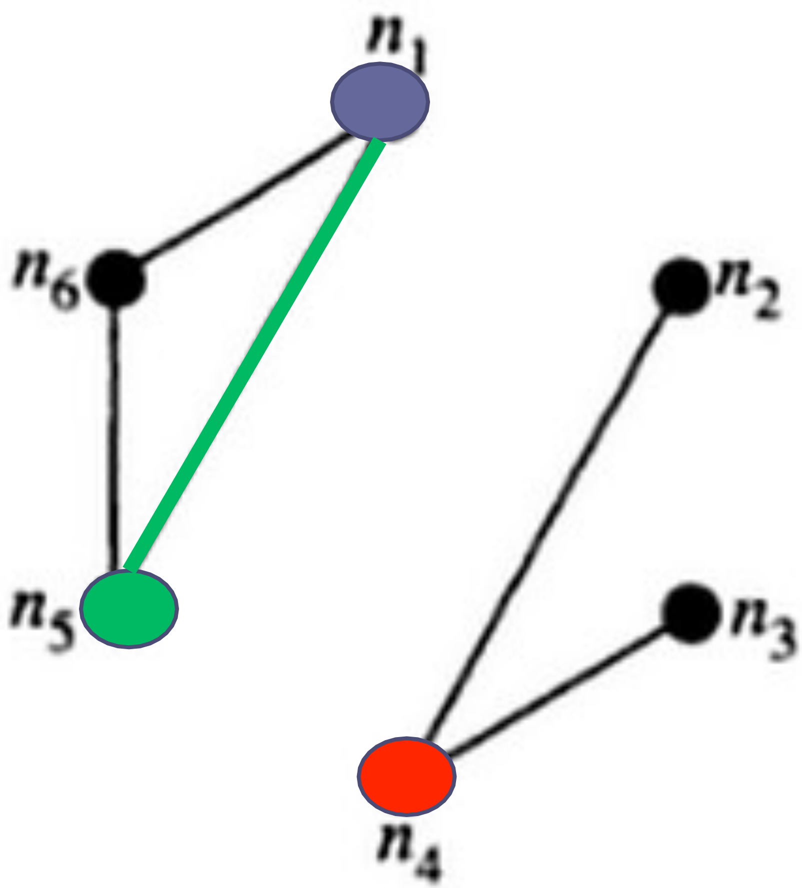
- Is this a Walk? Trail? Path?
  - We call a *closed path* is a Cycle!



$n_2\, l_4\, n_3\, l_5\, n_4\, l_3\;\; n_2$

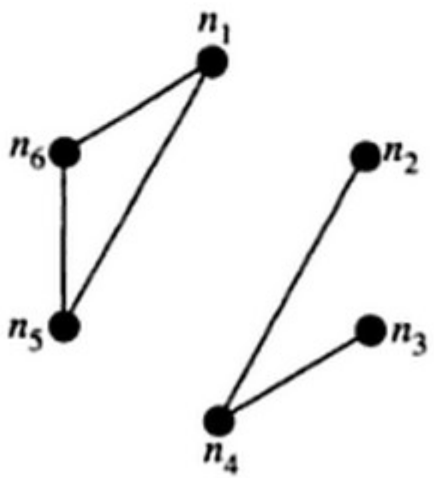
Reachability

- If there is a **path between nodes** *i* and *j*, then *i* and *j* are reachable from each other.



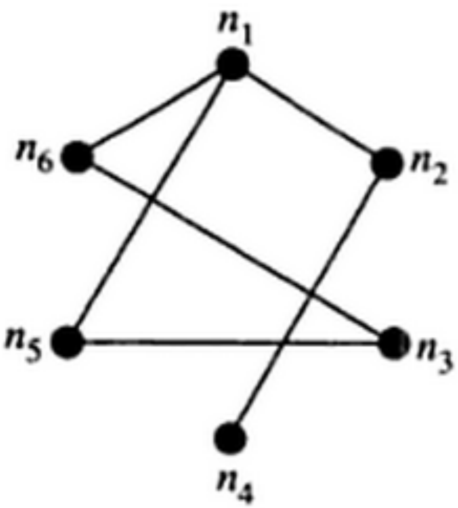
Connected Graph

- A graph is connected if ***every pair of its nodes*** are reachable from each other
  - i.e. there is a path between them.



Disconnected Graph

How can we make this graph connected?



Connected Graph

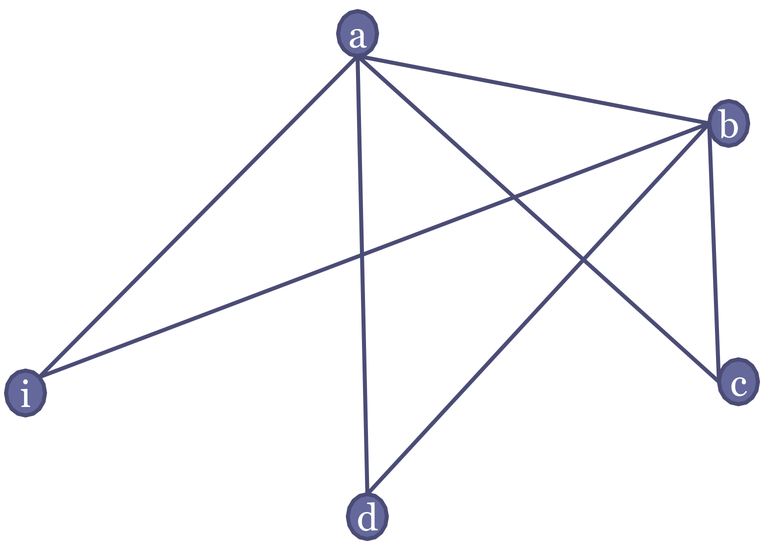
and this graph disconnected?

Sub-graphs

- Graph G_s is a sub-graph of G if its nodes and edges are a subset of G's nodes and edges respectively.

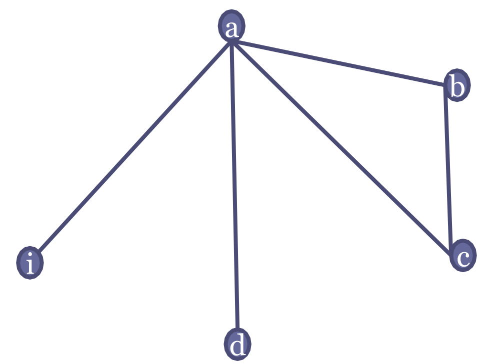
Sub-graphs- Cnt.

- Graph G_s is a sub-graph of G if its nodes and edges are a subset nodes and edges of G respectively.

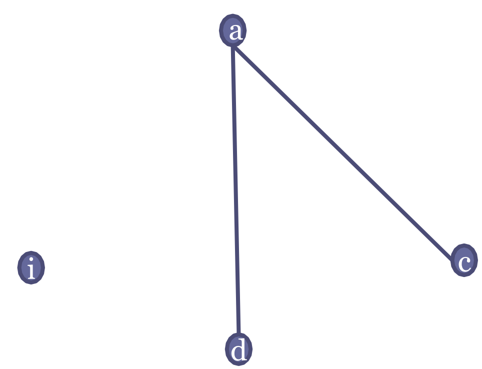


G

G_{s1}



G_{s2}

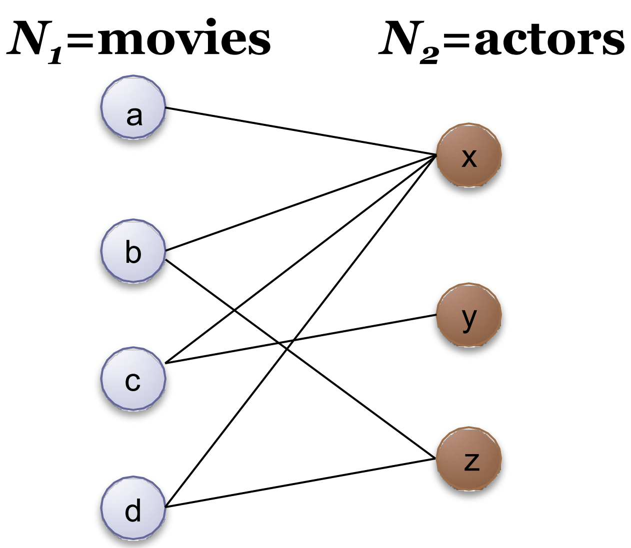


Graph Types

- Several types of graphs:
  - Bipartite graphs
  - Digraphs
  - Multigraphs
  - Hypergraphs
  - Weighted/Signed

Graph Types- Bipartite Graphs

- A bipartite graph is an undirected graph in which
  - nodes can be partitioned into two (disjoint) sets N_1 and N_2 such that:
    - $(u, v) \in E$ implies either $u \in N_1$ and $v \in N_2$ or vice versa
  - So, all edges go between the two sets N_1 and N_2 but not within N_1 or N_2.



N_1={a,b,c,d}

N_2={x,y,z}

Graph Types- Digraphs

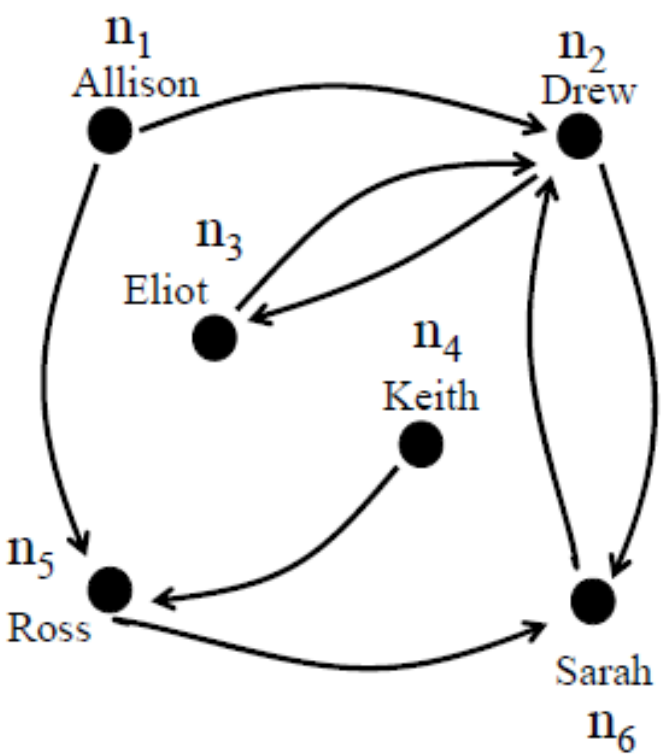
- Digraphs or Directed Graphs
  - Edges are directed
- Adjacency:
  - There is a direct edge btw nodes!
    - $i \in N$
    - $j \in N$
    - $(i, j) \in E$

Graph Types- Digraphs- Cnt.

- Node Indegree and Outdegree
  - Indegree
    - The indegree of a node, $d_I(i)$, is the number of nodes that link to *i*,
  - Outdegree
    - The outdegree of a node, $d_O(i)$, is the number of nodes that are linked by *i*,

- Indegree: number of edges terminating at *i*.
- Outdegree: number of edges originating at *i*.

Graph Types- Digraphs- Cnt.



$$d_O(n_i) = \sum_{j=1}^{n} A_{ij}$$

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \begin{array}{|c|} \hline 2 \\ \hline 2 \\ \hline 1 \\ \hline 1 \\ \hline 1 \\ \hline 1 \\ \hline \end{array}$$

$$d_I(n_j) = \sum_{i=1}^{n} A_{ij}$$

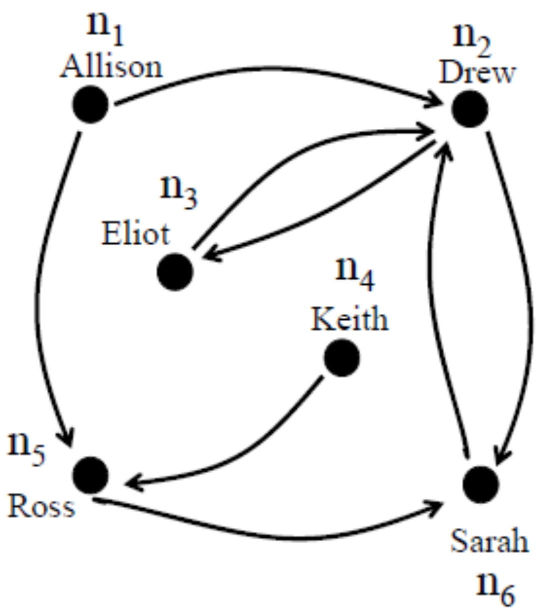
0	3	1	0	2	2

$A \mathrel{!}= A^T$

Graph Types- Digraphs- Cnt.

- Density of Digraph:
  - Number of all possible edges in Digraph?
    - N * (N-1)

$$\frac{E}{N * (N-1)}$$



Graph Types- Digraphs- Cnt.

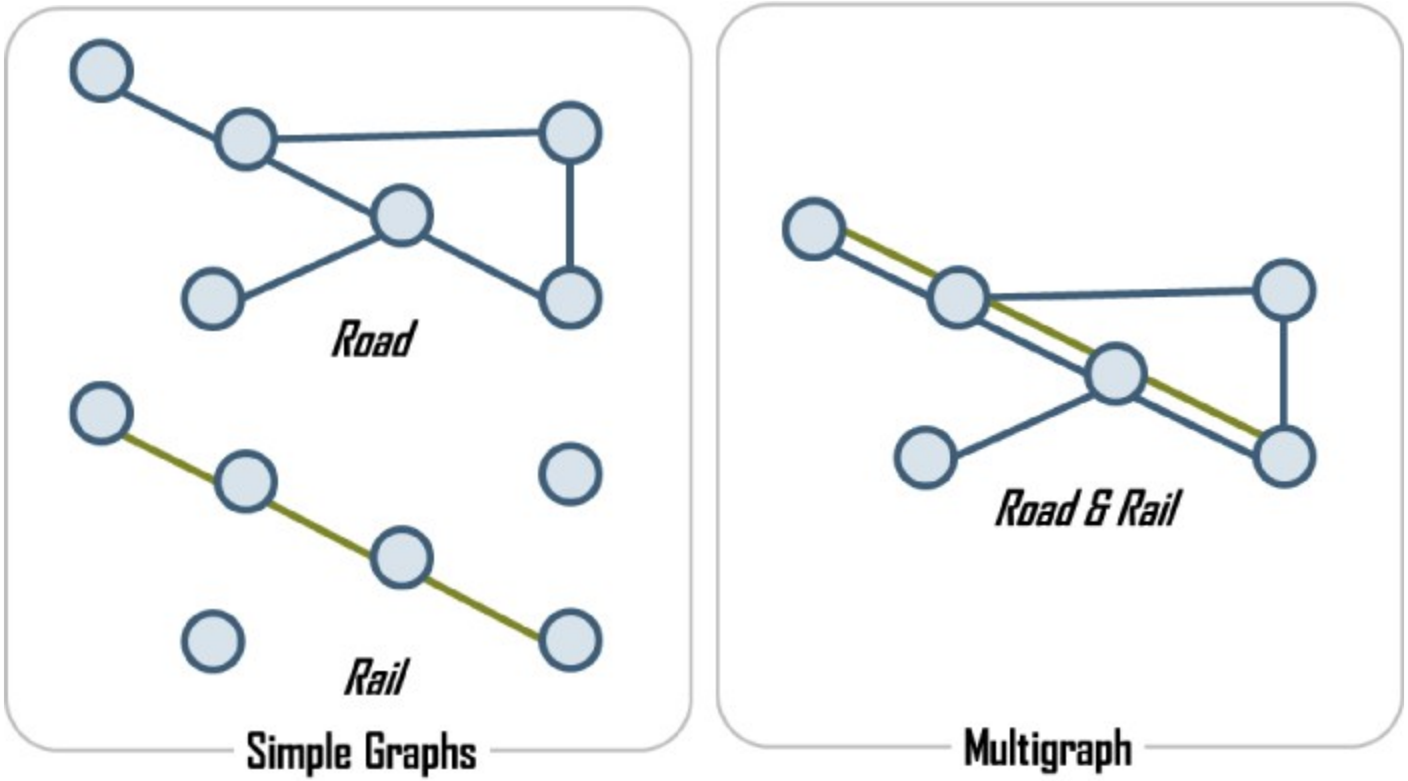
- Connectivity
  - Walks
  - Trails
  - Paths

- The same as before just links are directed!

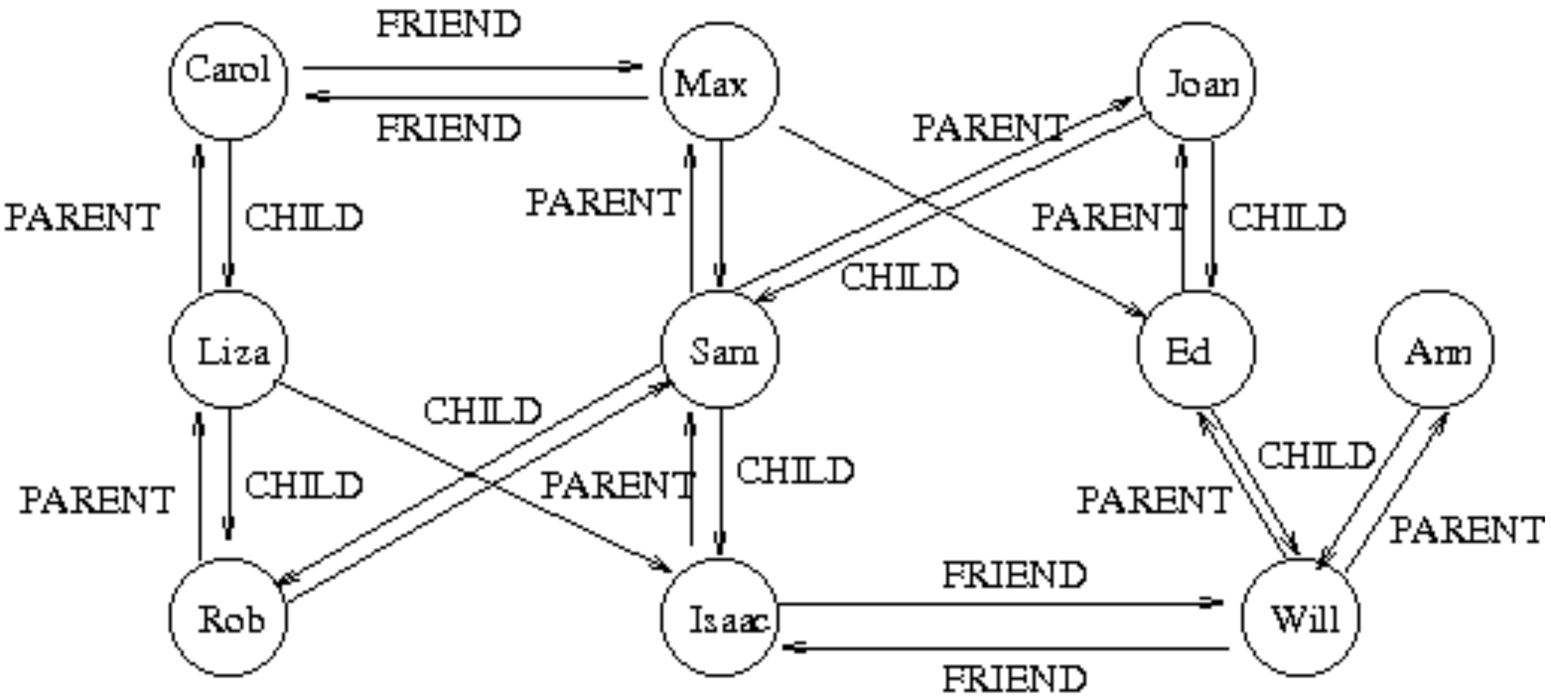
Graph Types- Multigraphs

- A Multigraph (or multivariate graph) *G* consists of:
  - a set of nodes, *and*
  - two or more sets of edges, $E^{+} = \{\boldsymbol{E_1}, \boldsymbol{E_2}, \ldots, \boldsymbol{E_r}\}$, r is the number of edge sets.

Multigraph 1.



Multigraph 2.



Graph Types- Multigraphs- Cnt.

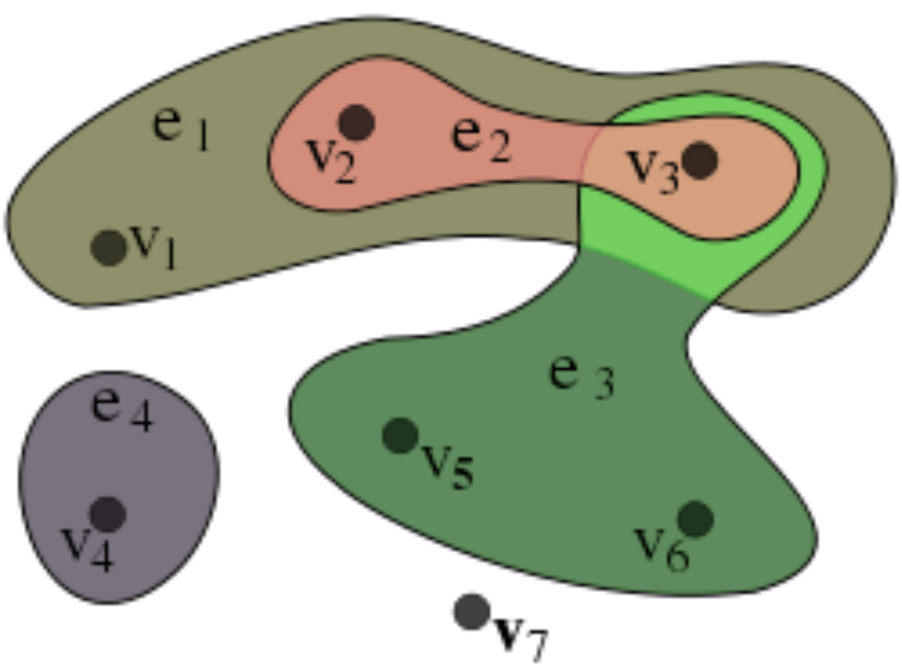
- Number of edges btw any two nodes in a multigraph?
  - $E^+ = \{\boldsymbol{E_1}, \boldsymbol{E_2}, \ldots, \boldsymbol{E_r}\}$, r is the number of sets of edges
    - Undirected multigraph
      - [0, r]
    - Directed multigraph
      - [0, 2*r]

Graph Types- Hypergraphs

- A hypergraph is a graph in which an edge can connect any number of nodes.
- In a hypergraph, $\boldsymbol{E}$ is a set of non-empty subsets of $\boldsymbol{N}$ called *hyperedges*.

Graph Types- Hypergraphs- Cnt.

- A hypergraph is a graph in which an edge can connect any number of nodes.
- In a hypergraph, ***E*** is a set of non-empty subsets of ***N*** called *hyperedges*.



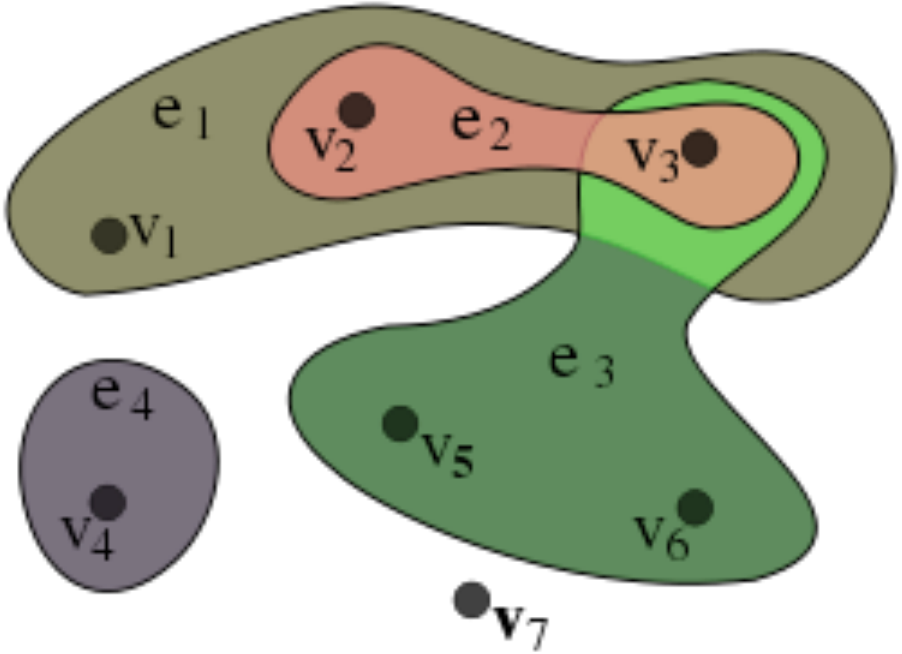
N$=\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$

E$=\{e_1, e_2, e_3, e_4\}=$

$\{\{v_1, v_2, v_3\}, \{v_2, v_3\}, \{v_3, v_5, v_6\}, \{v_4\}\}$

Graph Types- Hypergraphs- Cnt.

- Applications:
  - Recom. systems (communities as edges),
  - Image retrieval (correlations as edges),
  - Bioinformatics (interactions or semantic types as edges).



N$=\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$

E$=\{e_1, e_2, e_3, e_4\}=$

$\{\{v_1, v_2, v_3\}, \{v_2, v_3\}, \{v_3, v_5, v_6\}, \{v_4\}\}$

Weighted/Signed Graphs

- Edges may carry additional information
  - Tie strength → how good are two nodes as friends?
  - Distance → how long is the distance btw two cities?
  - Delay → how long does the transmission take btw two cities?
  - Signs → two nodes are friends or enemies?

Reading

- Ch. 22 Elementary Graph Algorithms [CLRS]

Network Basics 2

Graph ML

Department of Computer Science
University of Massachusetts, Lowell

Hadi Amiri
hadi@cs.uml.edu

Lecture Topics

- Connected Components
- Breadth-First Search
- Depth-First Search
- Shortest Path Algorithm
  - Dijkstra’s algorithm

Connected Components

- Connected component of a graph is a subset of nodes such that:
  - every node in the subset has a path to every other; and
  - the subset is not part of a bigger component.

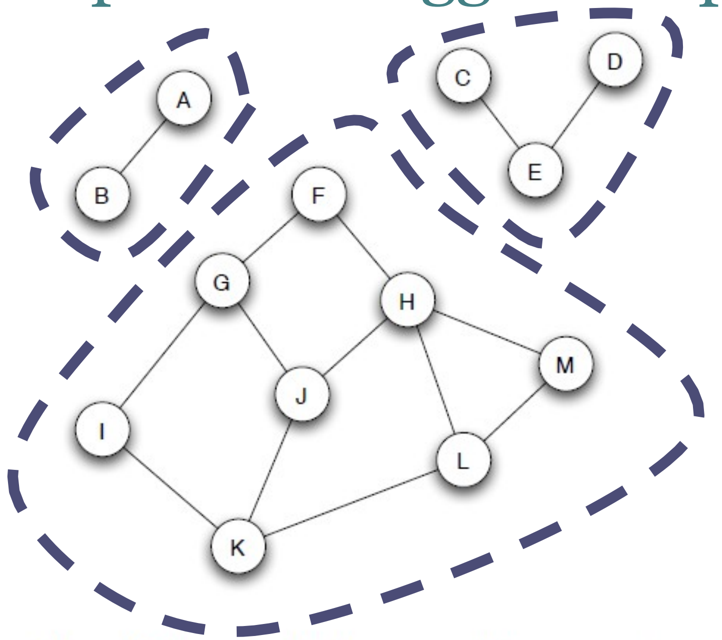


Figure 2.5: A graph with three connected components.

Connected Components

- Connected component of a graph is a subset of nodes such that:
  - every node in the subset has a path to every other; and
  - the subset is not part of a bigger component.

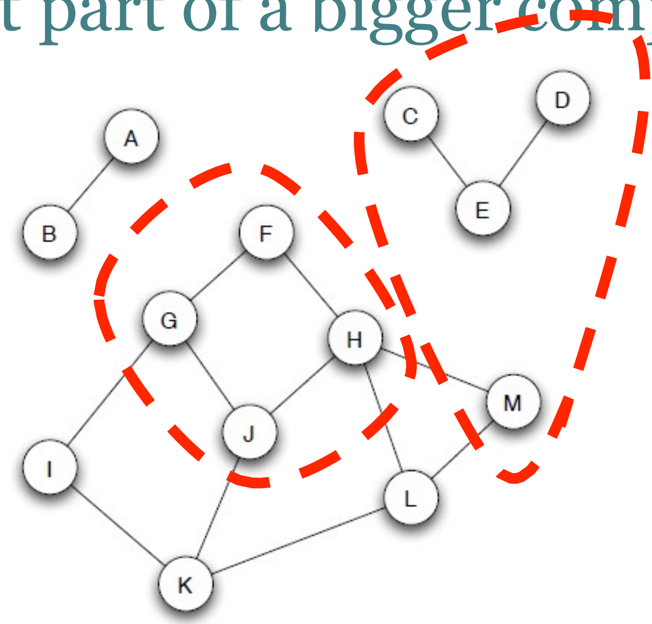
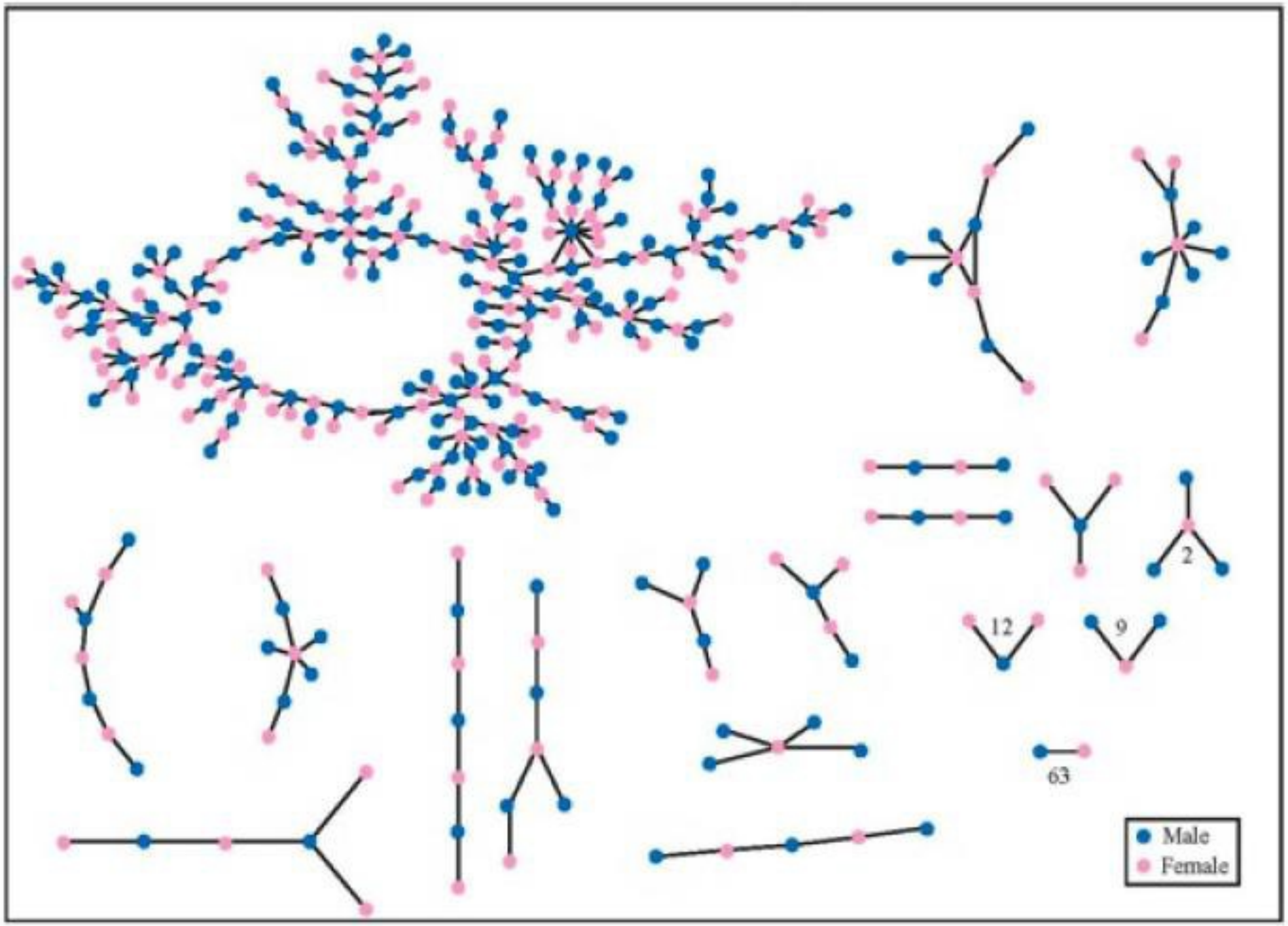


Figure 2.5: A graph with three connected components.

Connected Components- Cnt.



Connected Components- Cnt.

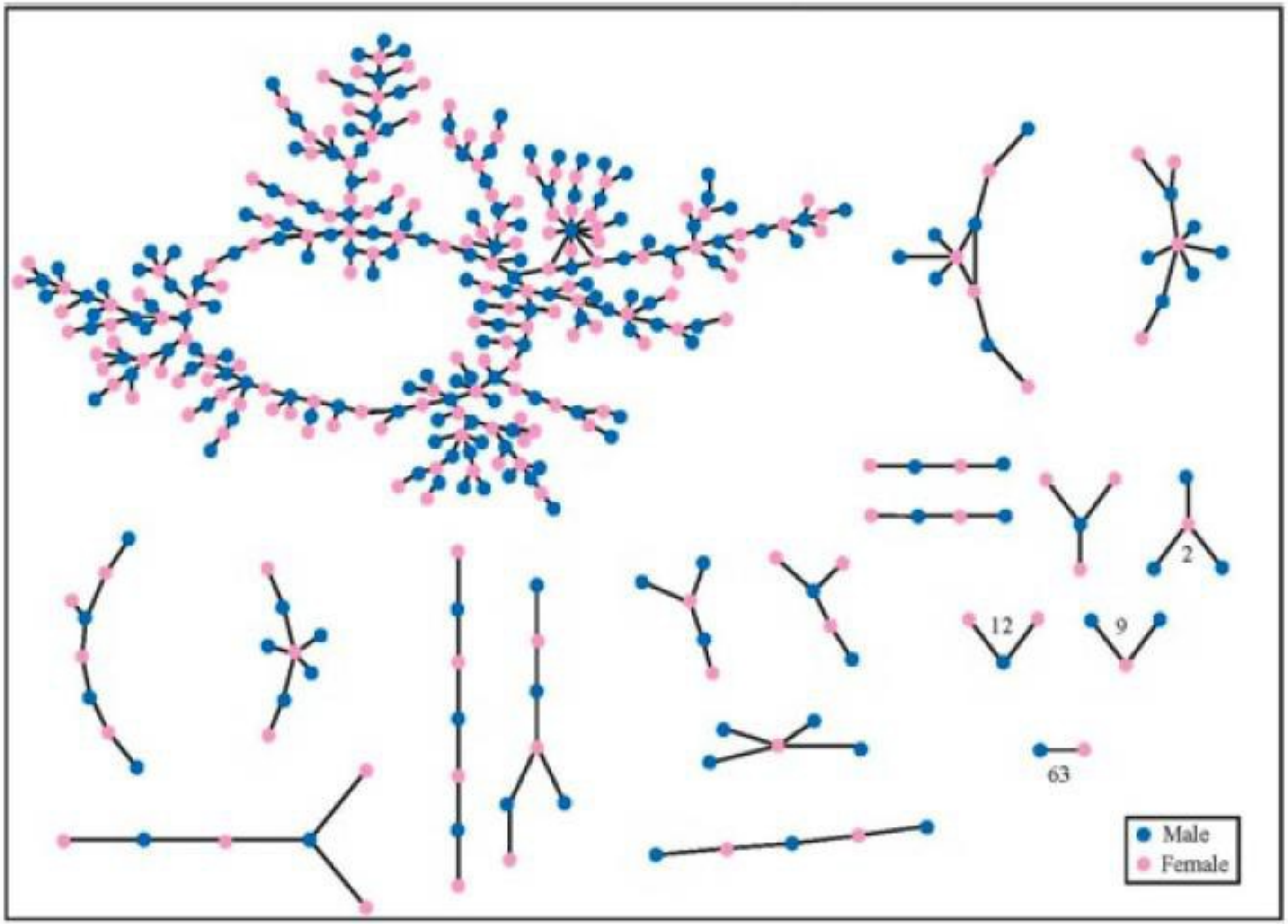


Figure 2.7: A network in which the nodes are students in a large American high school, and an edge joins two who had a romantic relationship at some point during the 18-month period in which the study was conducted [49].

Breadth & Depth-First Search

- General techniques for traversing graphs!
  - Start from a given node *s* (i.e. start node) and visit all nodes and edges in the graph.
- Compute the connected components of graph!
  - Use components to determine whether graph is connected!
    - How?
  - Use components to determine if there is a path btw node pairs!
    - How?

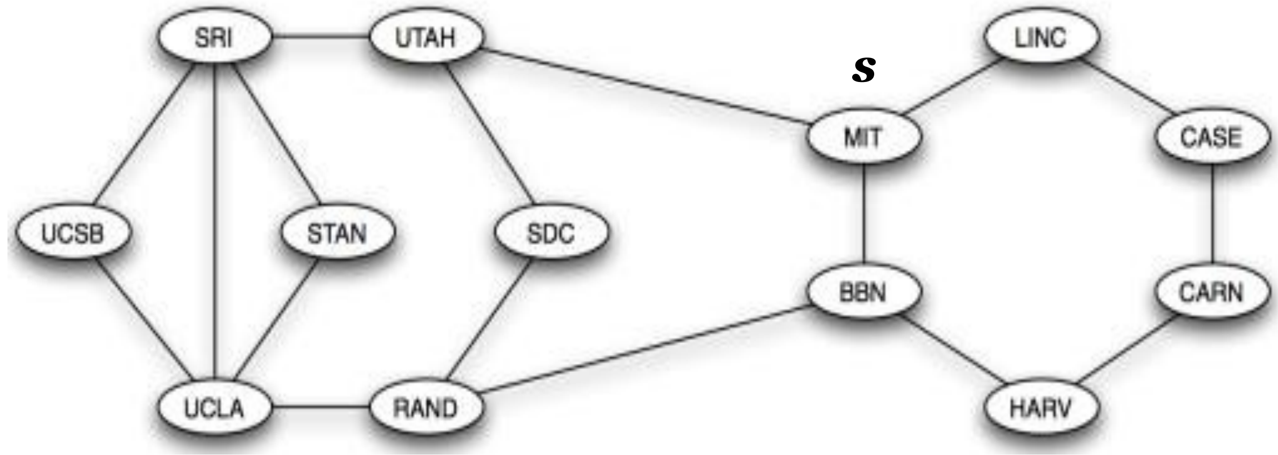
Breadth-First Search

- Start with *s*
- Visit all neighbors of *s*
  - these are called **level-1 nodes**
- Visit all neighbors of level-1 nodes
  - these are called **level-2 nodes**
- Repeat until all nodes are visited.
  - Each Node is only visited once.

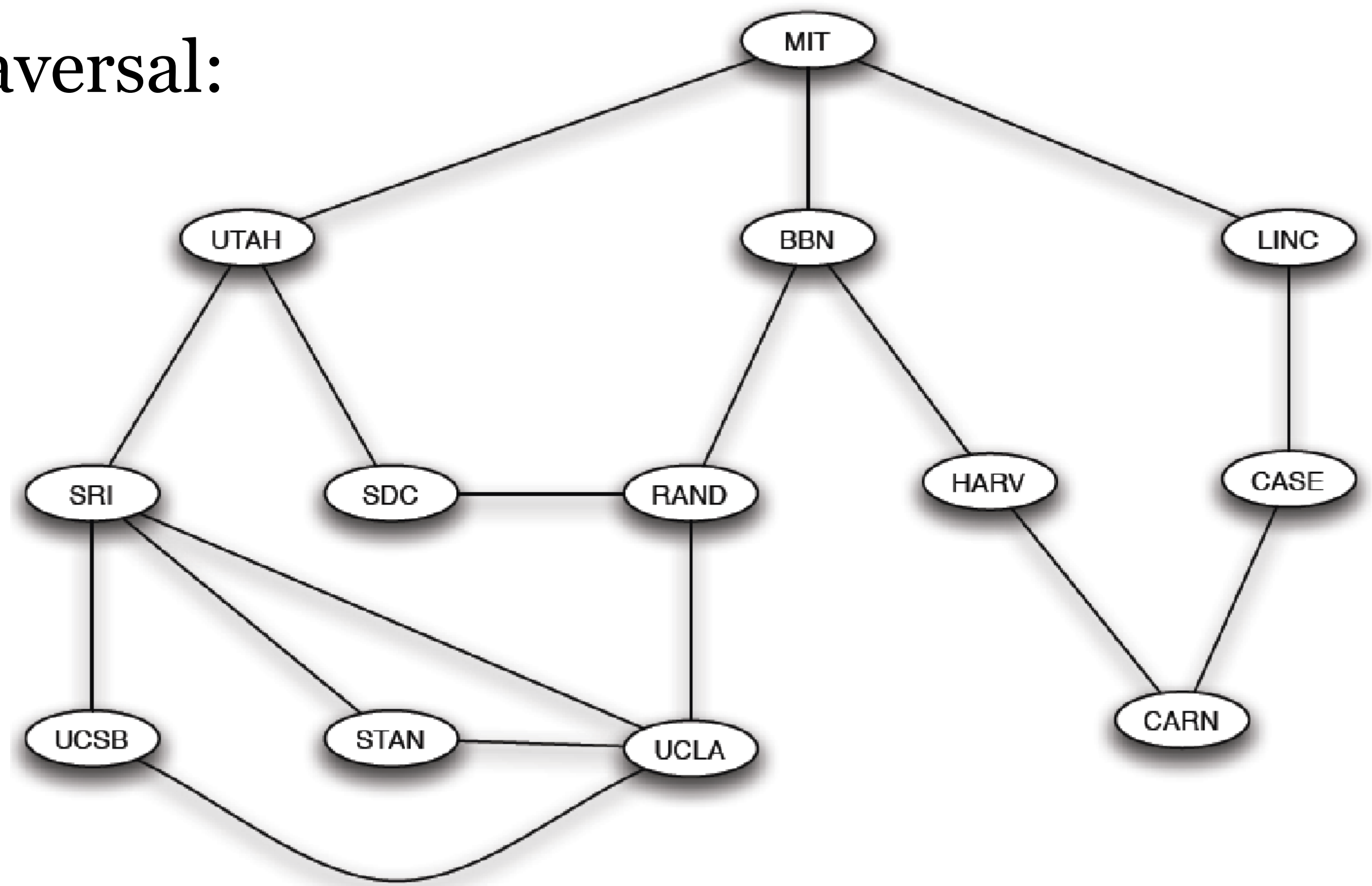
- Key Point:
  - All level-k nodes should be visited before any level-(k+1) node!

Example 1.

- Graph G:

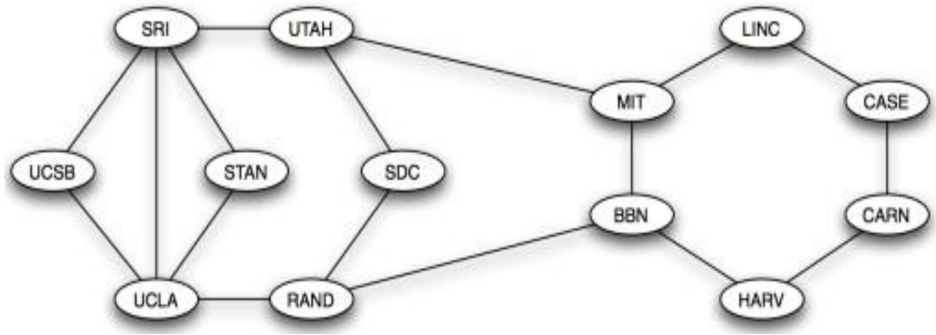


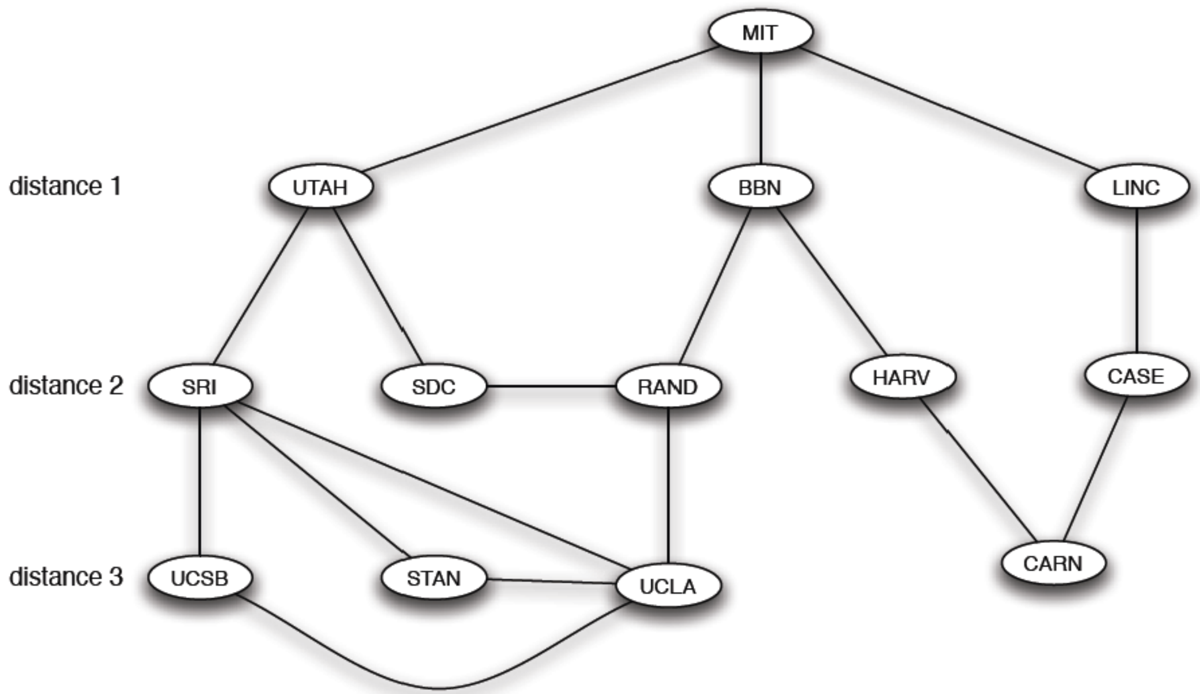
- Its BFS traversal:



Example 1. BFS -Cnt.

- BFS traversal:
  - Distance to root at level-*i*?
  - Components?
    - Connectivity?
    - Paths?





Depth-First Search

- Starts from s
- Explores as far as possible along each branch before backtracking.
    - Visit a neighbor of *s* [say v_1]
    - Visit a neighbor of v_1 [say v_2]
    - Repeat until all nodes are visited.

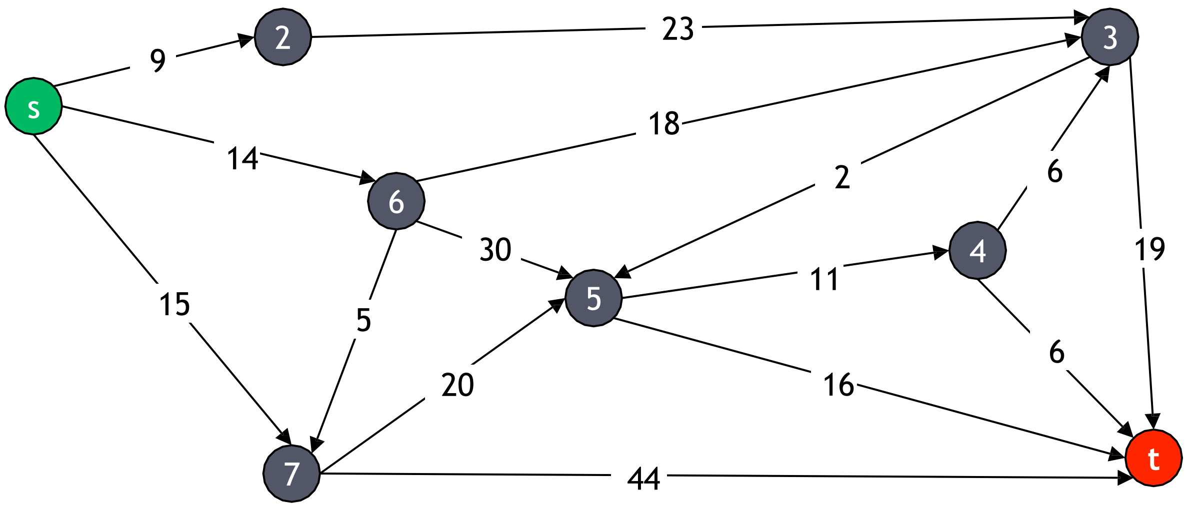
Shortest Path Algorithms

- Given a weighted directed graph and two nodes *s* and *t*, find the shortest path from *s* to *t*.
  - Cost of path = sum of edge weights in path

Shortest Path Algorithms- Cnt.

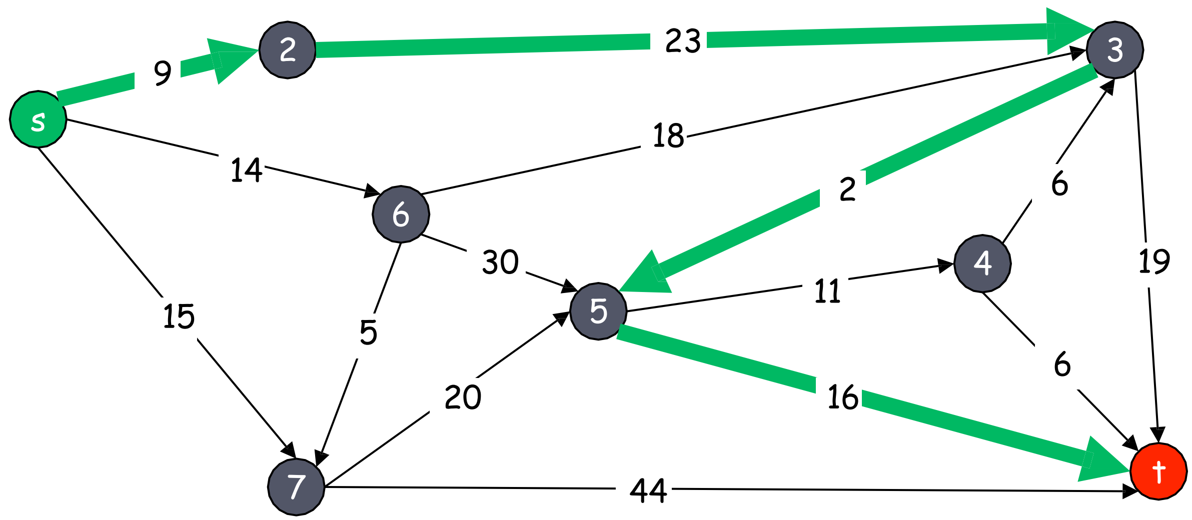
- Dijkstra’s algorithm
- The Bellman-Ford algorithm
- The Floyd-Warshall algorithm
- Johnson's algorithm
- Etc.

Shortest Path Algorithms- Cnt.



- Shortest path from *s* to *t*?

Shortest Path Algorithms- Cnt.



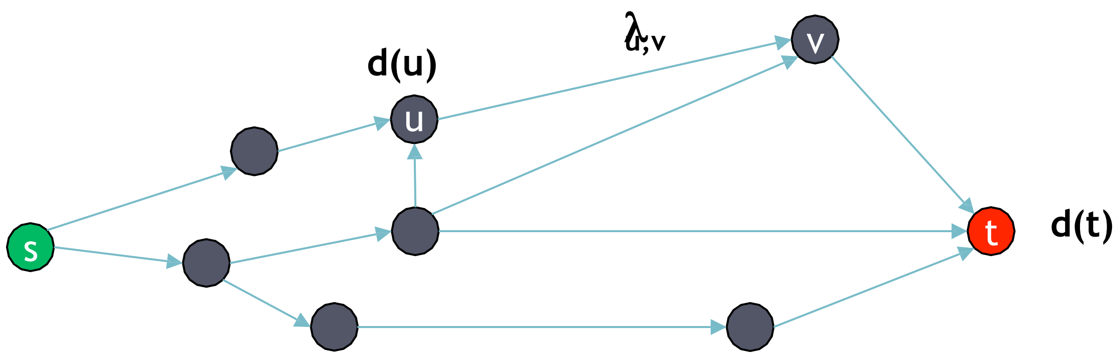
- Shortest Path= s-2-3-5-t
- Cost of path = 9 + 23 + 2 + 16 = 48.

Shortest Path Algorithms- Cnt.

- Applications
  - Small World Phenomenon
  - Internet packet routing
  - Flight reservations
  - Driving directions
  - …

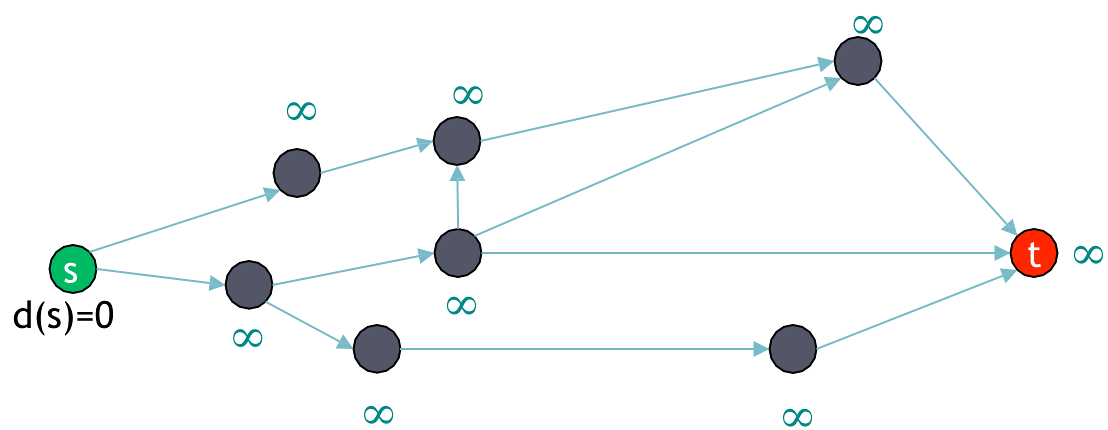
Dijkstra algorithm

- Weighted Directed graph G = (**N**, **E**),
  - s: source node
  - t: target node
  - $l_{(u,v)}$: weight of the edge btw nodes u and v
  - d(u): shortest path distance from s to u.
    - sum of edge weights in path
- We aim to compute d(t)!



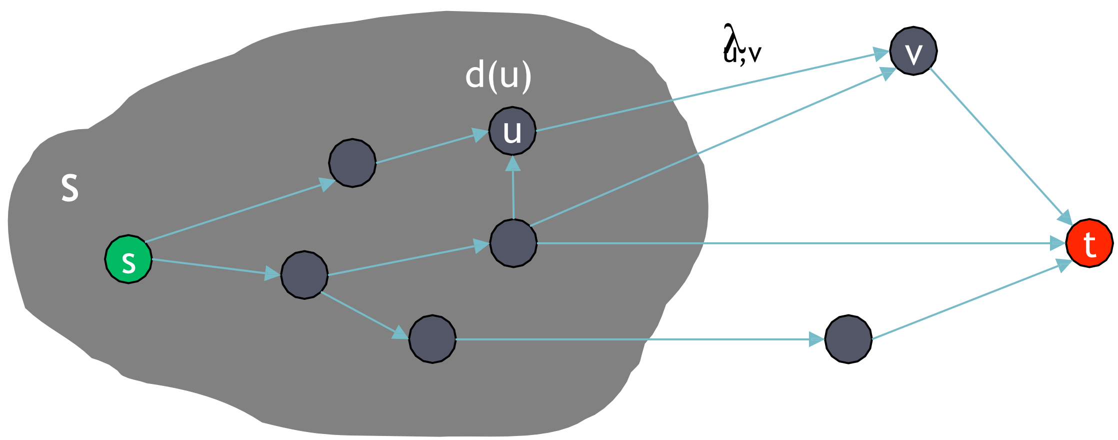
Dijkstra algorithm- Cnt.

- Initialization?
  - d(s) = 0
  - d(u)= ∞ for all other nodes



Dijkstra algorithm- Cnt.

- To find the shortest path from *s* to *t*:
  - Maintain a set of ***explored nodes*** **S** for which we have determined the shortest path distance from s to any u ϵ **S**.
  - Repeatedly expand **S**.

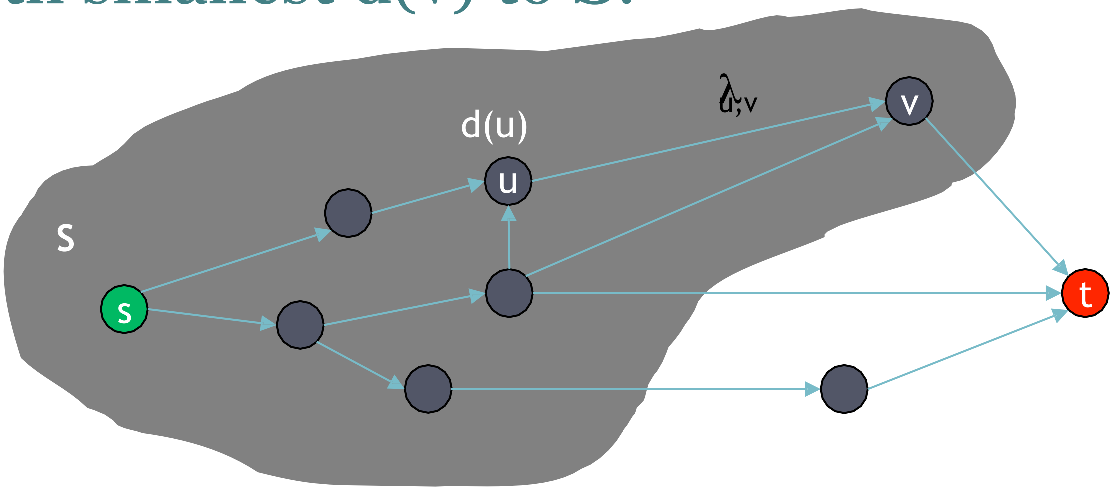


Dijkstra algorithm- Cnt.

- Repeatedly expand **S**?
  - Repeatedly update *d(.)* for the unexplored nodes:

$$\textbf{if } d(v) > d(u) + l_{(u,v)}$$
$$\textbf{then } d(v) \leftarrow \quad d(u) + l_{(u,\mathrm{v})}$$

  - add v with smallest d(v) to **S**.



Dijkstra algorithm- Cnt.

- $d(s) \leftarrow 0$
- **for** each $v \in N - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $S \leftarrow \emptyset$
- $Q \leftarrow N$ ▷ Q is a set maintaining $N - S$
- **while** $Q \neq \emptyset$
  - do $u \leftarrow$ EXTRACT-MIN(Q)
    - $S \leftarrow S \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$

Set of explored nodes

Set of unexplored nodes

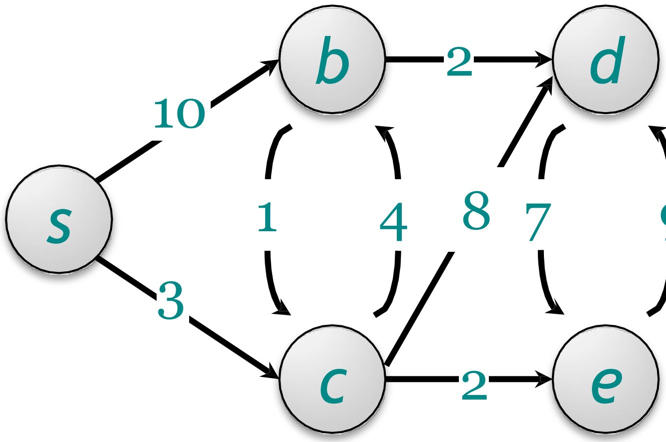
Returns node u ∈ Q that has minimum d(u)

Add it to explored nodes

Update d(.) for all neighbors of u: this is called **relaxation**!

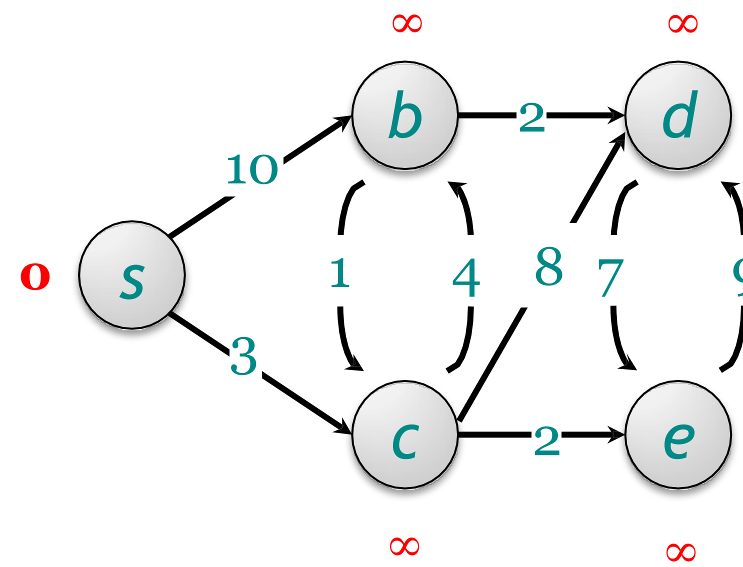
Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in \boldsymbol{N} - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $\boldsymbol{S} \leftarrow \varnothing$
- $\boldsymbol{Q} \leftarrow \boldsymbol{N}$
- **while** $\boldsymbol{Q} \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN($\boldsymbol{Q}$)
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,\,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$



Example 1.

- $d(s) \leftarrow 0$
- **for each** $v \in N - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $S \leftarrow \varnothing$
- $Q \leftarrow N$
- **while** $Q \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN(Q)
    - $S \leftarrow S \cup \{u\}$
    - **for each** $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$

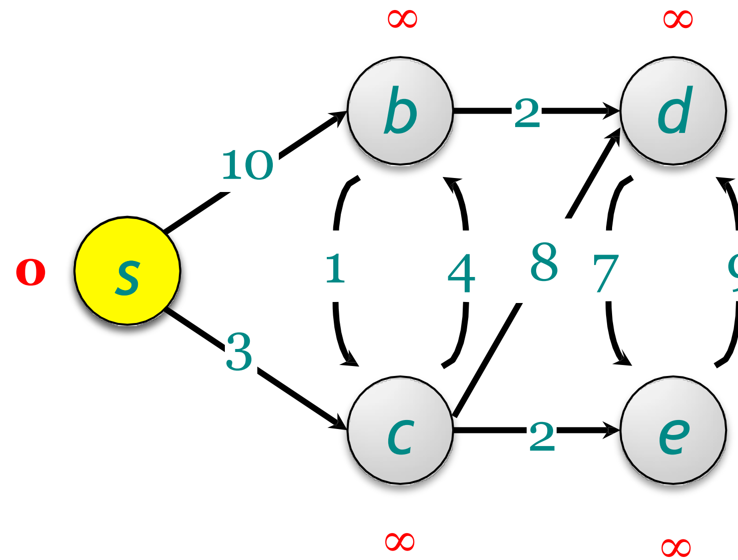


S={}

Q={s, b, c, d, e}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in \boldsymbol{N} - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $\boldsymbol{S} \leftarrow \varnothing$
- $\boldsymbol{Q} \leftarrow \boldsymbol{N}$
- **while** $\boldsymbol{Q} \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN($\boldsymbol{Q}$)
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$



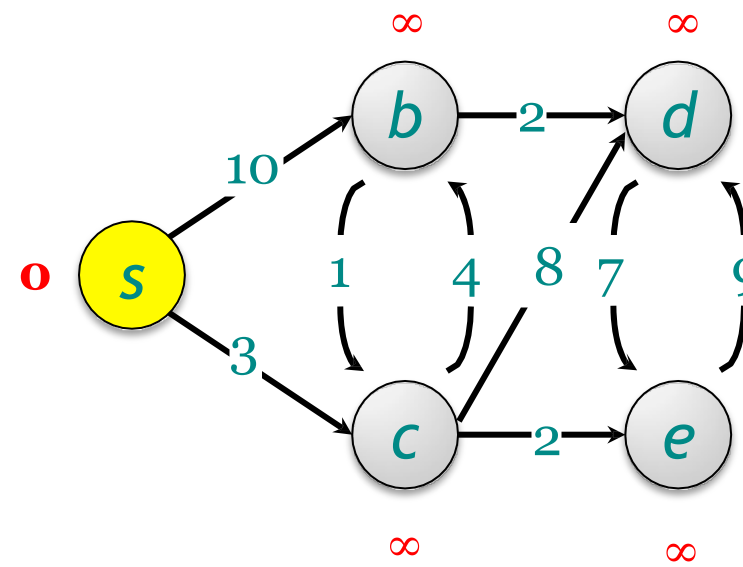
S={}

s

Q={b, c, d, e}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in N - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $S \leftarrow \varnothing$
- $Q \leftarrow N$
- **while** $Q \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN(Q)
    - $S \leftarrow S \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u, v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u, v)}$

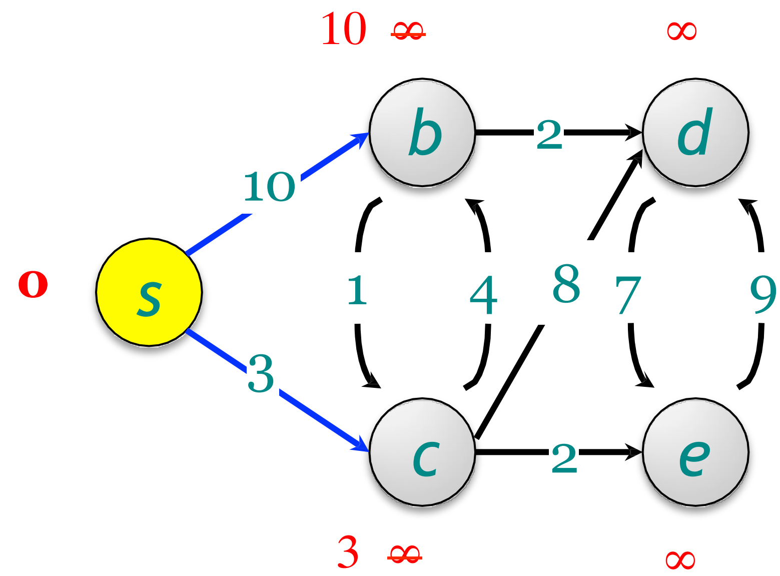


S={s}

Q={b, c, d, e}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in \boldsymbol{N} - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $\boldsymbol{S} \leftarrow \varnothing$
- $\boldsymbol{Q} \leftarrow \boldsymbol{N}$
- **while** $\boldsymbol{Q} \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN($\boldsymbol{Q}$)
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$

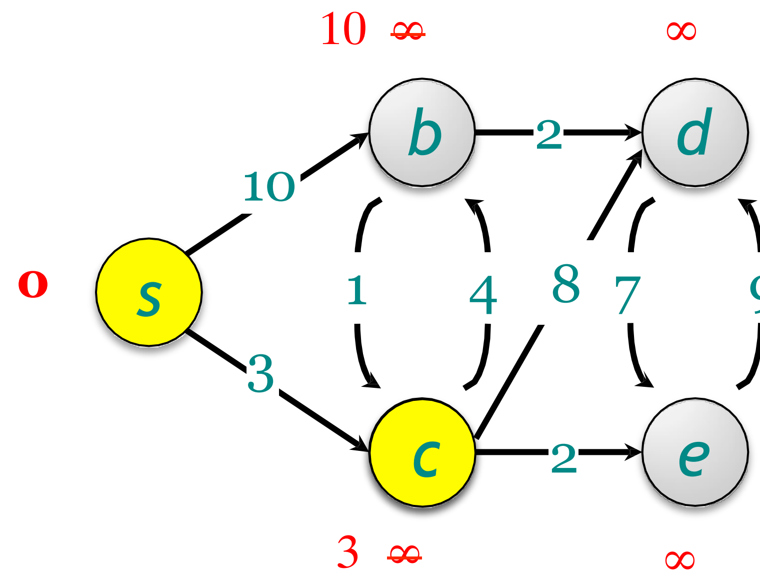


S={s}

Q={b, c, d, e}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in N - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $S \leftarrow \emptyset$
- $Q \leftarrow N$
- **while** $Q \neq \emptyset$
  - **do** $u \leftarrow$ EXTRACT-MIN(Q)
    - $S \leftarrow S \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$



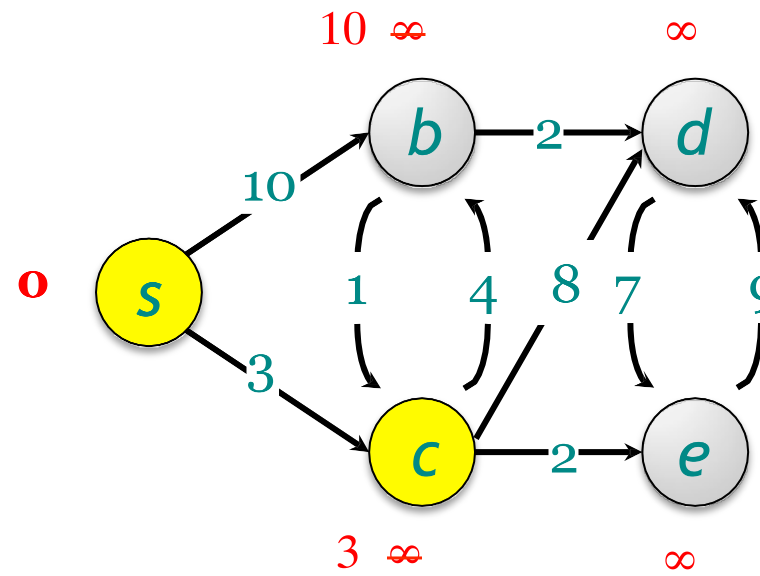
S={s}

c

Q={b, d, e}

Example 1.

- $d(s) \leftarrow 0$
- **for each** $v \in N - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $S \leftarrow \emptyset$
- $Q \leftarrow N$
- **while** $Q \neq \emptyset$
  - **do** $u \leftarrow$ EXTRACT-MIN(Q)
    - $S \leftarrow S \cup \{u\}$
    - **for each** $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$

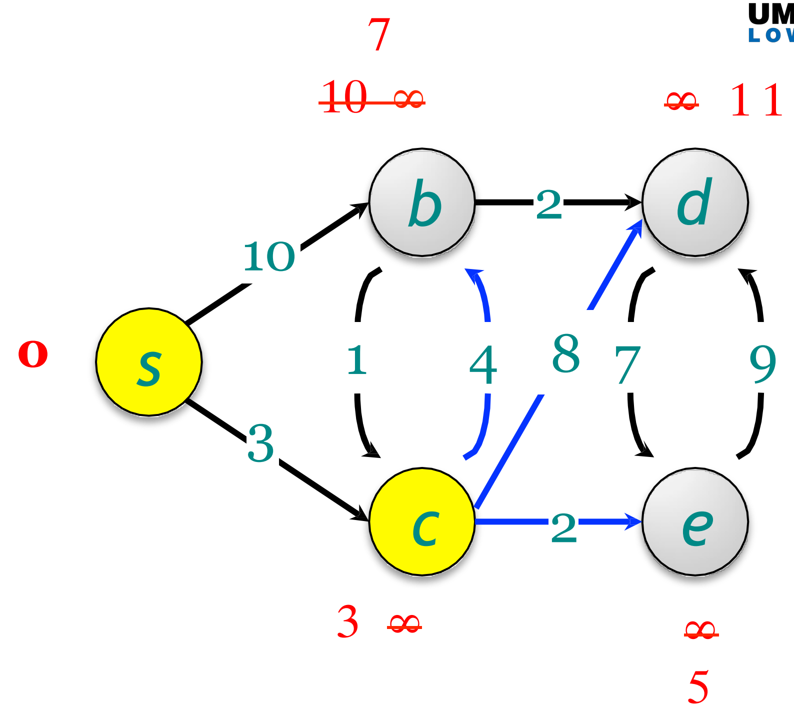


S={s, c}

Q={b, d, e}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in \boldsymbol{N} - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $\boldsymbol{S} \leftarrow \emptyset$
- $\boldsymbol{Q} \leftarrow \boldsymbol{N}$
- **while** $\boldsymbol{Q} \neq \emptyset$
  - **do** $u \leftarrow$ EXTRACT-MIN($\boldsymbol{Q}$)
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,\, v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,\, v)}$

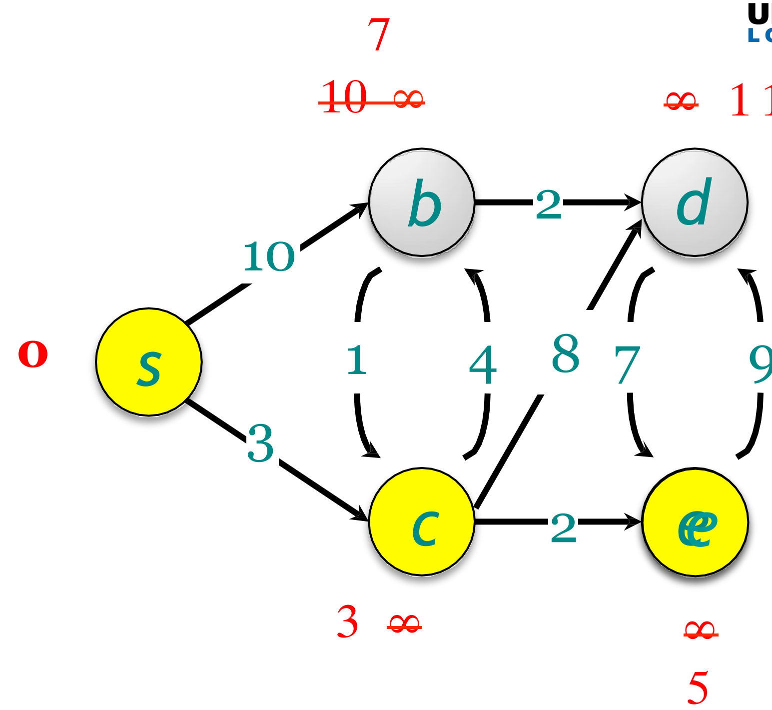


S={s, c}

Q={b, d, e}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in \boldsymbol{N} - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $\boldsymbol{S} \leftarrow \varnothing$
- $\boldsymbol{Q} \leftarrow \boldsymbol{N}$
- **while** $\boldsymbol{Q} \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN($\boldsymbol{Q}$)
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$



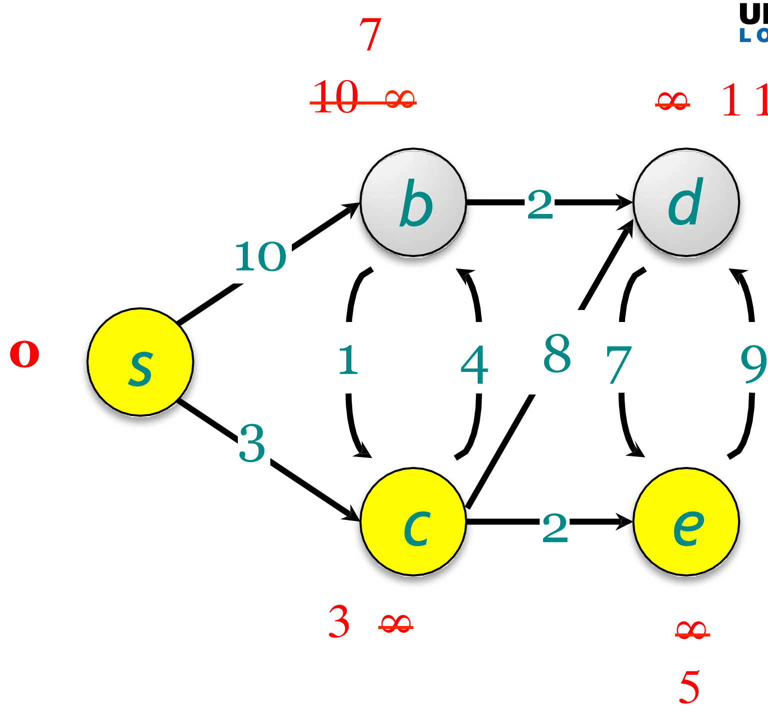
S={s, c}

e

Q={b, d}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in \boldsymbol{N} - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $\boldsymbol{S} \leftarrow \varnothing$
- $\boldsymbol{Q} \leftarrow \boldsymbol{N}$
- **while** $\boldsymbol{Q} \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN($\boldsymbol{Q}$)
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u, v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u, v)}$

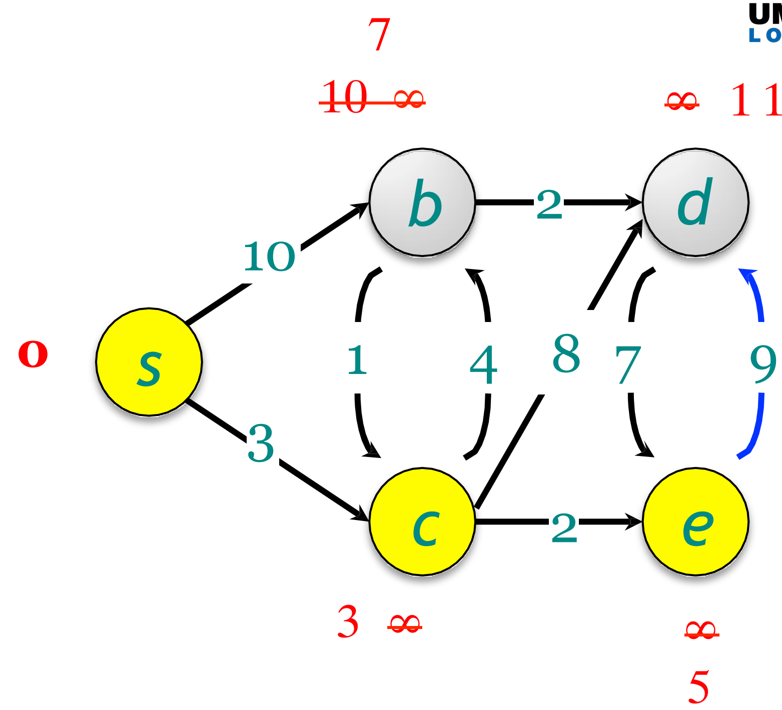


S={s, c, e}

Q={b, d}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in N - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $S \leftarrow \varnothing$
- $Q \leftarrow N$
- **while** $Q \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN(Q)
    - $S \leftarrow S \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$

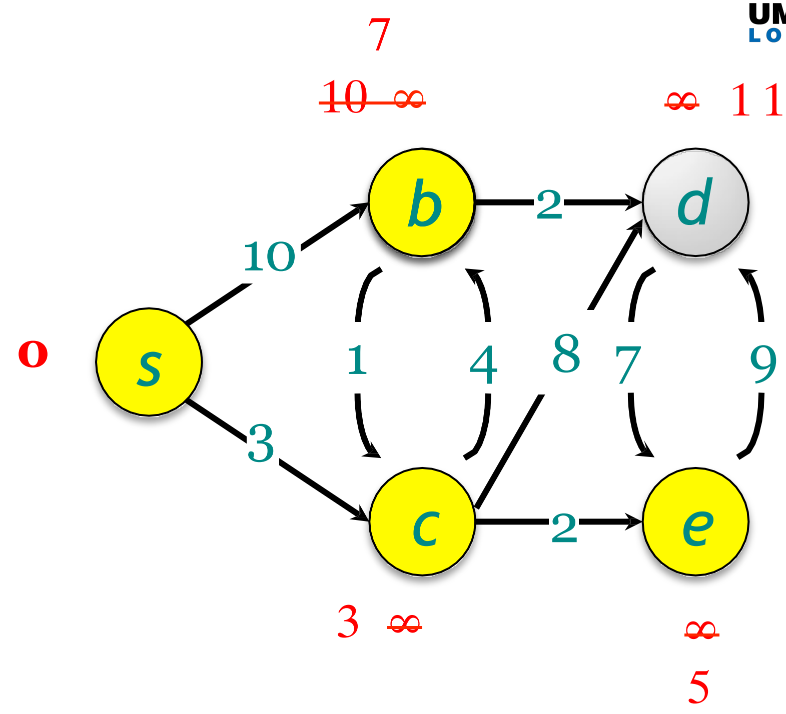


S={s, c, e}

Q={b, d}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in N - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $S \leftarrow \varnothing$
- $Q \leftarrow N$
- **while** $Q \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN(Q)
    - $S \leftarrow S \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$



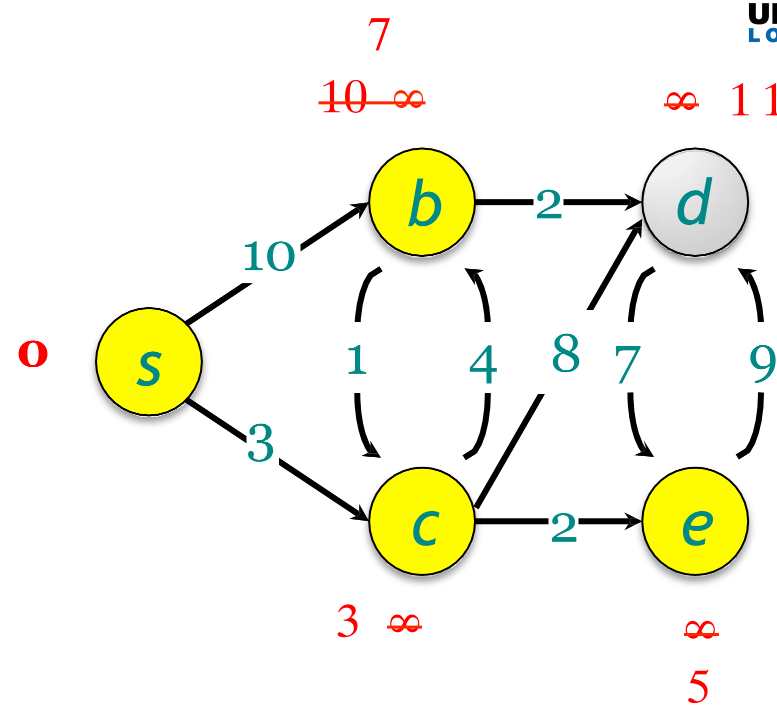
S={s, c, e}

b

Q={d}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in \boldsymbol{N} - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $\boldsymbol{S} \leftarrow \varnothing$
- $\boldsymbol{Q} \leftarrow \boldsymbol{N}$
- **while** $\boldsymbol{Q} \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN($\boldsymbol{Q}$)
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$

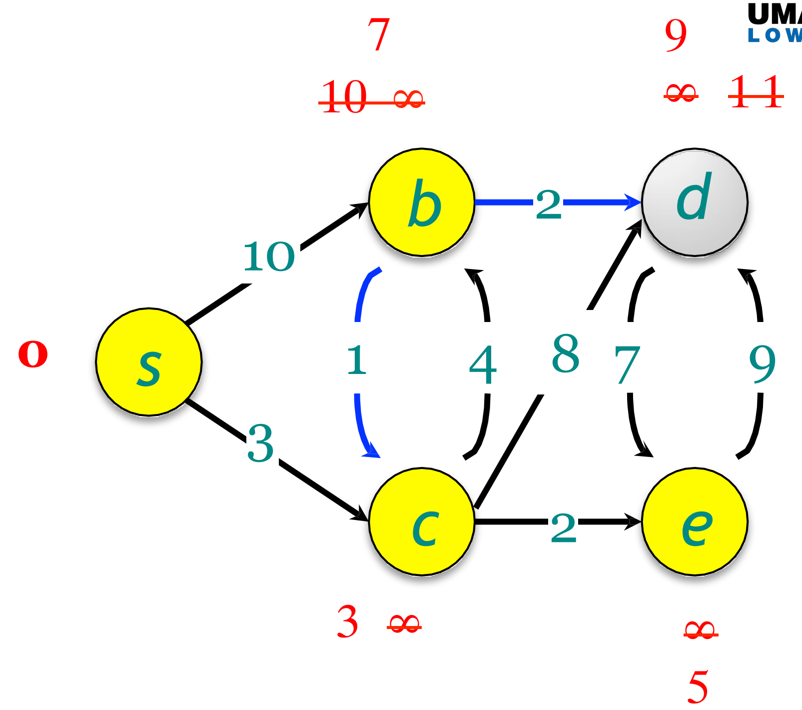


S={s, c, e, b}

Q={d}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in N - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $S \leftarrow \emptyset$
- $Q \leftarrow N$
- **while** $Q \neq \emptyset$
  - **do** $u \leftarrow$ EXTRACT-MIN(Q)
    - $S \leftarrow S \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$

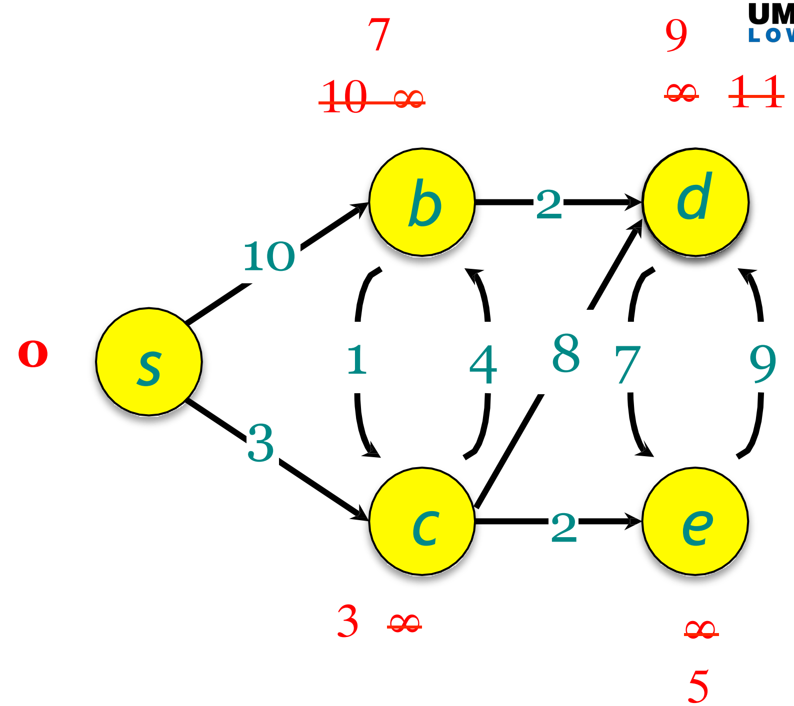


S={s, c, e, b}

Q={d}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in \boldsymbol{N} - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $\boldsymbol{S} \leftarrow \varnothing$
- $\boldsymbol{Q} \leftarrow \boldsymbol{N}$
- **while** $\boldsymbol{Q} \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN($\boldsymbol{Q}$)
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$



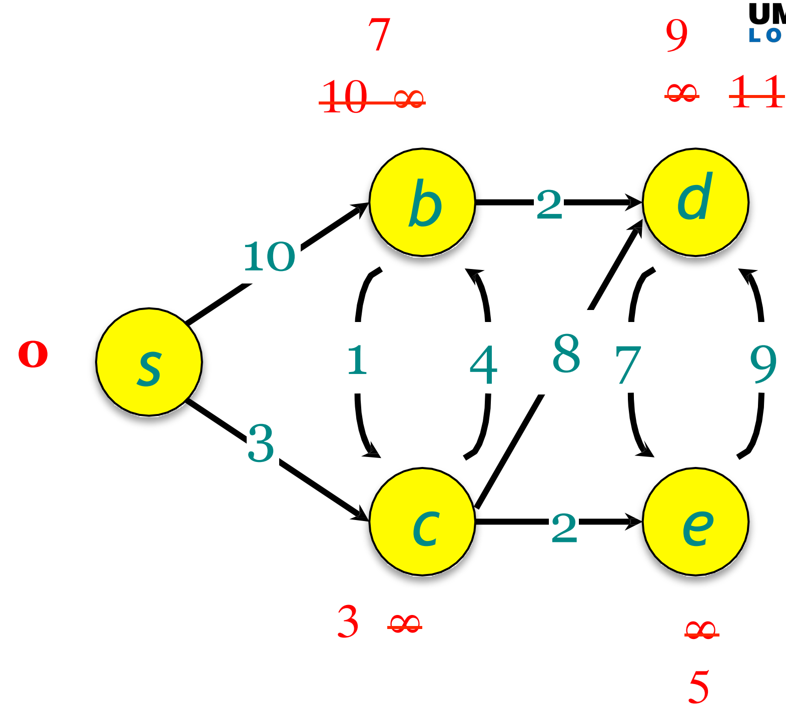
S={s, c, e, b}

d

Q={}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in \boldsymbol{N} - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $\boldsymbol{S} \leftarrow \varnothing$
- $\boldsymbol{Q} \leftarrow \boldsymbol{N}$
- **while** $\boldsymbol{Q} \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN($\boldsymbol{Q}$)
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$

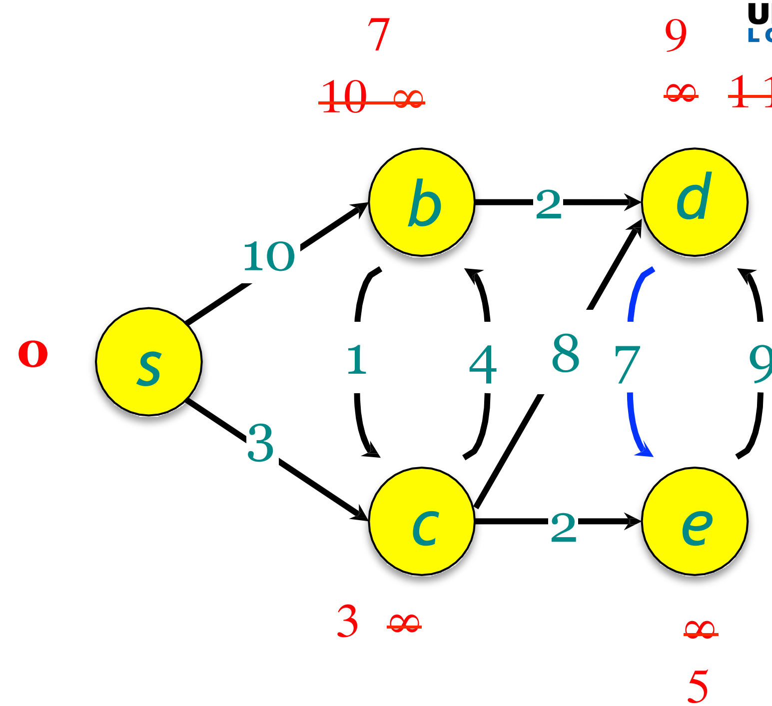


S={s, c, e, b, d}

Q={}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in \boldsymbol{N} - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $\boldsymbol{S} \leftarrow \emptyset$
- $\boldsymbol{Q} \leftarrow \boldsymbol{N}$
- **while** $\boldsymbol{Q} \neq \emptyset$
  - **do** $u \leftarrow$ EXTRACT-MIN($\boldsymbol{Q}$)
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u, v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u, v)}$

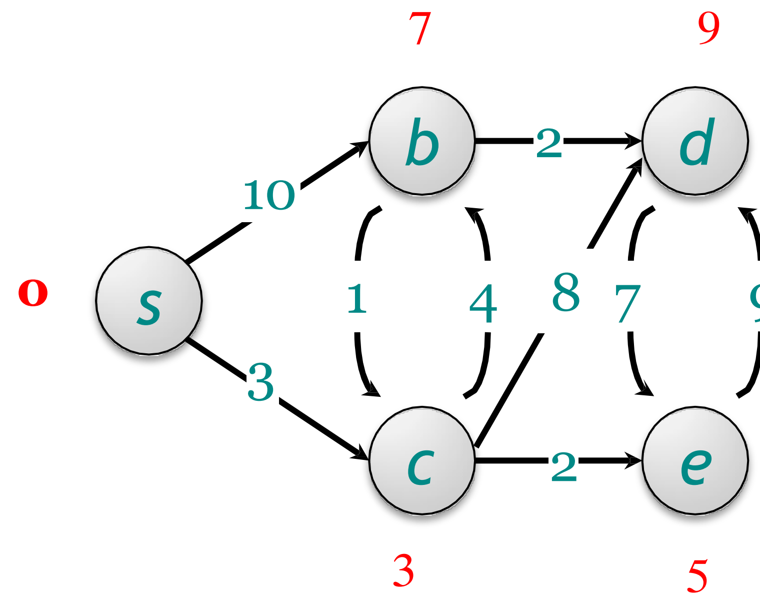


S={s, c, e, b, d}

Q={}

Example 1.

- $d(s) \leftarrow 0$
- **for** each $v \in N - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $S \leftarrow \varnothing$
- $Q \leftarrow N$
- **while** $Q \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN(Q)
    - $S \leftarrow S \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u, v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u, v)}$



S={s, c, e, b, d}

Q={}

Dijkstra's algorithm- Cnt.

- Dijkstra's algorithm computes the shortest distances btw a start node and all other nodes in the graph (not only a target node)!

- Assumptions:
  - the graph is connected, and
  - the weights are nonnegative

Dijkstra's algorithm- Analysis

- $d(s) \leftarrow 0$
- **for** each $v \in \boldsymbol{N} - \{s\}$
  - **do** $d(v) \leftarrow \infty$
- $\boldsymbol{S} \leftarrow \varnothing$
- $\boldsymbol{Q} \leftarrow \boldsymbol{N}$
- **while** $\boldsymbol{Q} \neq \varnothing$
  - **do** $u \leftarrow$ EXTRACT-MIN($\boldsymbol{Q}$)
    - $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$
    - **for** each $v \in Adj(u)$
      - **do if** $d(v) > d(u) + l_{(u,v)}$
        - **then** $d(v) \leftarrow d(u) + l_{(u,v)}$

degree (u) times

$|\boldsymbol{N}|$ times

Time = $\Theta(N \cdot T_{\text{EXTRACT-MIN}} + E \cdot T_{\text{Relaxation}})$, Handshaking Lemma!

Dijkstra's algorithm- Analysis- Cnt.

$$\text{Time} = \Theta\left(N \cdot T_{\text{EXTRACT-MIN}} + E \cdot T_{\text{Relaxation}}\right)$$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
Array	$O(N)$	$O(1)$	$O(N^2)$

Reading

- Ch.24 Single Source Shortest Paths [CLRS]

Network Basics 3

Graph ML

Department of Computer Science
University of Massachusetts, Lowell

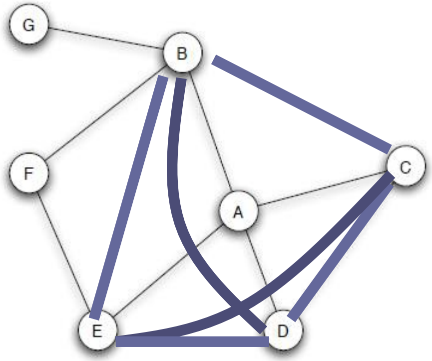
Hadi Amiri
hadi@cs.uml.edu

Lecture Topics

- Triadic closure and Bridges
- Neighborhood overlap
- The Strength of Weak Ties
- Structural Holes
- Node Centrality
- Edge Centrality
- Homophily
- Snapshot Algorithm
- Network Segregation

Triadic Closure

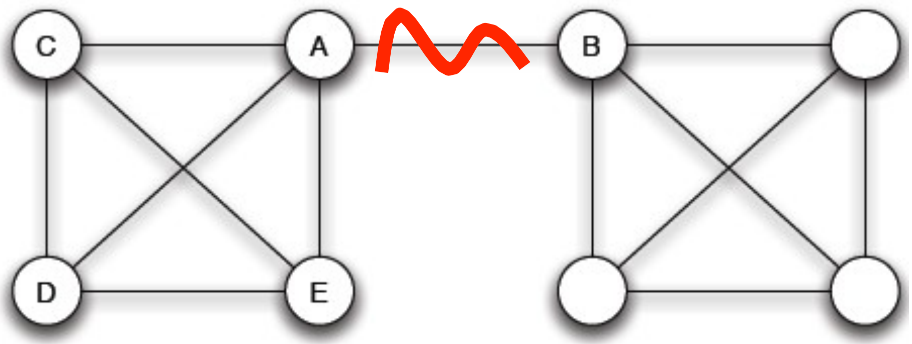
- If two **nodes** in a network have a **neighbor** in common, then there is an increased likelihood they will become **connected** themselves.
  - Reasons for Triadic Closure:
    - Opportunity, Trust, Incentives
- Clustering Coefficient
  - A measure to capture the prevalence of Triadic Closure
  - Defined for **nodes**



$$CF(A) = \frac{\textit{Number of connections btw A's friends}}{\textit{Possible Number of connections btw A's friends}} = 1/6$$

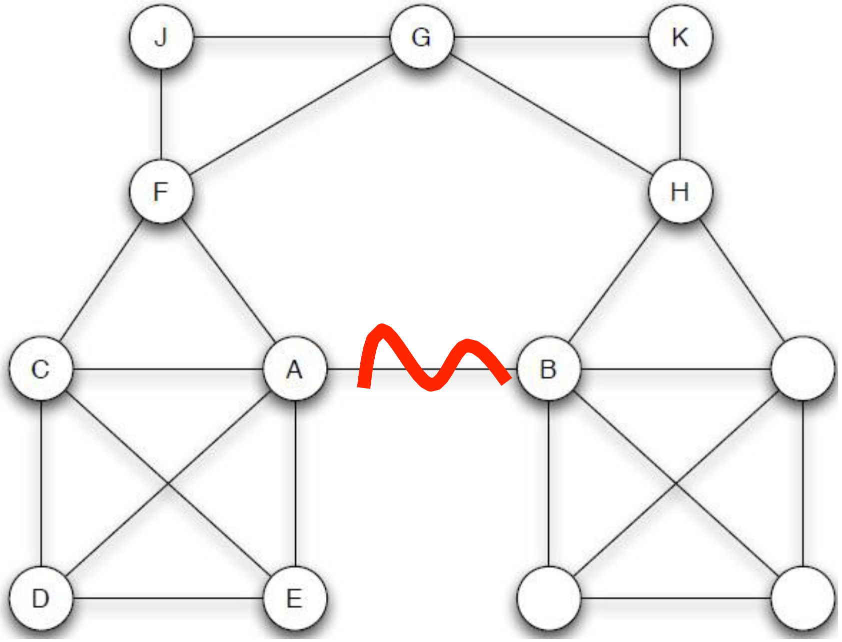
Bridge

- An edge is bridge if deleting it would put its two ends into two different connected components.
  - Bridges provide access to parts of the network that are unreachable by other means!



Local Bridge

- An edge such that its endpoints have no friends in common! → edge not in a triangle!
  - deleting a local bridge increases the distance btw its endpoints to a value strictly > **2**.



The Strength of Weak Ties

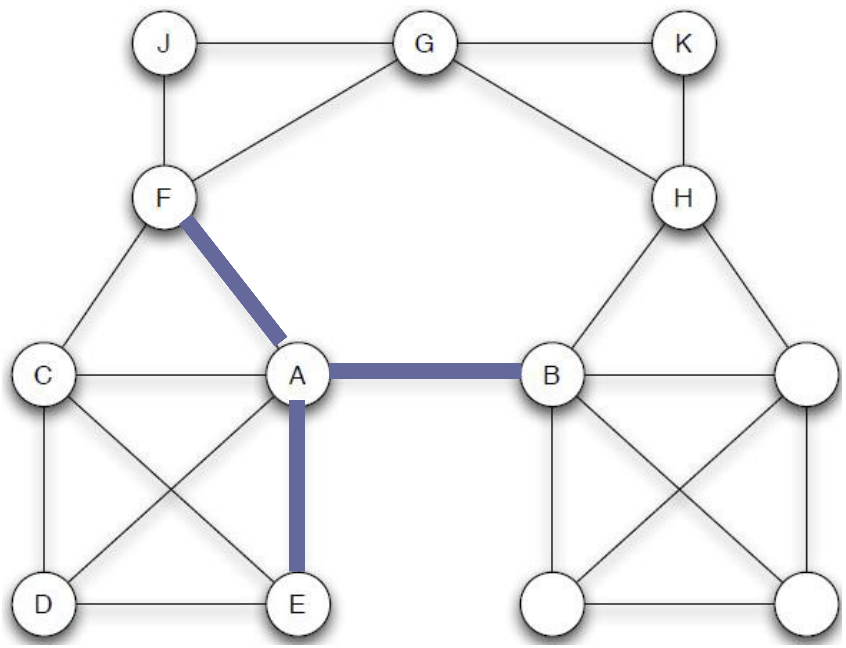
- Weak ties (acquaintances) connect us to new sources of information.
  - This dual role - as weak connections but also valuable links to hard-to-reach parts of the network - is the surprising strength of weak ties.

Neighborhood Overlap

- A measure to capture bridgeness of an edge!

$$\frac{\text{number of nodes who are neighbors of } \textit{both } A \text{ and } B}{\text{number of nodes who are neighbors of } \textit{at least one of } A \text{ or } B\text{'}}$$

Don't count A and B here!



Nodes	Neighborhood overlap
A-E	2/4
A-F	1/6
A-B	0/8 (**Overlap = 0** for local bridges)

Edges with very small neighborhood overlap can be considered as "almost" local bridges

Questions

1. Relation btw neighborhood overlap of an edge and its tie strength?

Questions

1. Relation btw neighborhood overlap of an edge and its tie strength?
   - Neighborhood overlap should grow as tie strength Grows.

Questions

2. How weak ties serve to link different communities that each contain large number of stronger ties?

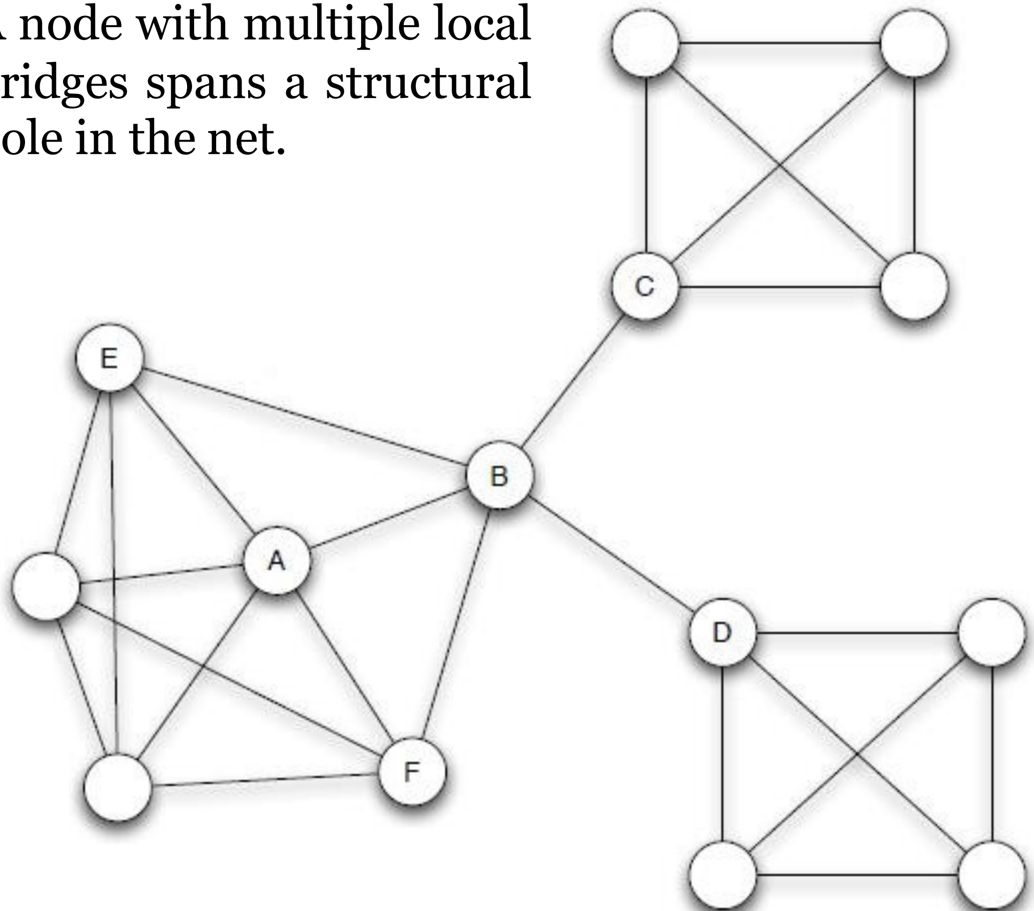
Questions

2. How weak ties serve to link different communities that each contain large number of stronger ties?
   - Delete edges from the network one at a time, start with the weakest ties first!
     - The giant component shrinks rapidly.

Structural Holes

Structural hole: the “empty space” in the net btw 2 sets of nodes that don’t interact closely!

A node with multiple local bridges spans a structural hole in the net.



B has early access to info!

B is a gatekeeper and controls the ways in which groups learn about info. She has power!

B may try to prevent triangles from forming around the local bridges she is part of!

How long these local bridges last before triadic closure produces short-cuts around them?

Node Centrality

- Degree centrality
  - A node is central if it has ties to many other nodes
- Closeness centrality
  - A node is central if it is “close” to other nodes
- Betweenness centrality
  - A node is central if other nodes have to go through it to get to each other

Edge Centrality

- Betweenness:
  - Let's assume 1 unit of "flow" will pass over all shortest path btw any pair of nodes A and B.
  - Betweenness of an edge is the total amount of flow it carries!
  - If there are *k* shortest path btw A and B, then 1/k units of flow will go along each shortest path!
- Girvan-Newman Algorithm:
  - Repeat until no edges are left:
    - Calculate betweenness of edges
    - Remove edges with highest betweenness

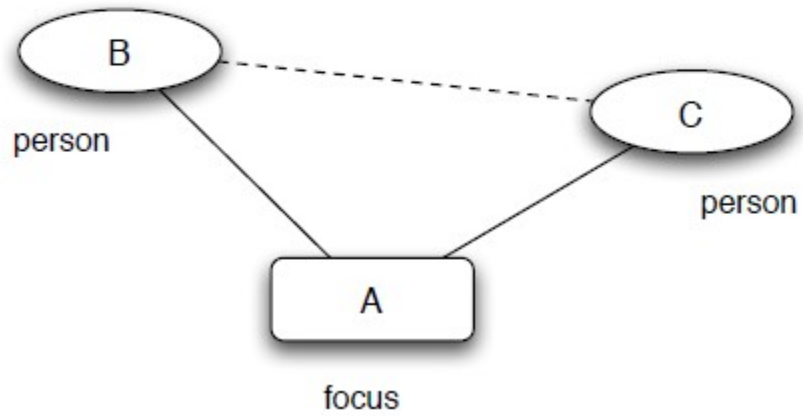
Homophily

- Links connect people with *similar* characteristics.
- Homophily has two mechanisms for link formation:
  - Selection:
    - Selecting friends with similar characteristics
      - Individual characteristics drive the formation of links
      - Immutable characteristics
  - Social Influence (socialization)
    - Modify behaviors to make them close to behaviors of friends
      - Existing links influence the individual characteristics of the nodes
      - Mutable characteristics

Homophily- Cnt.

- **Focal Closure**:
  B and C people, A focus

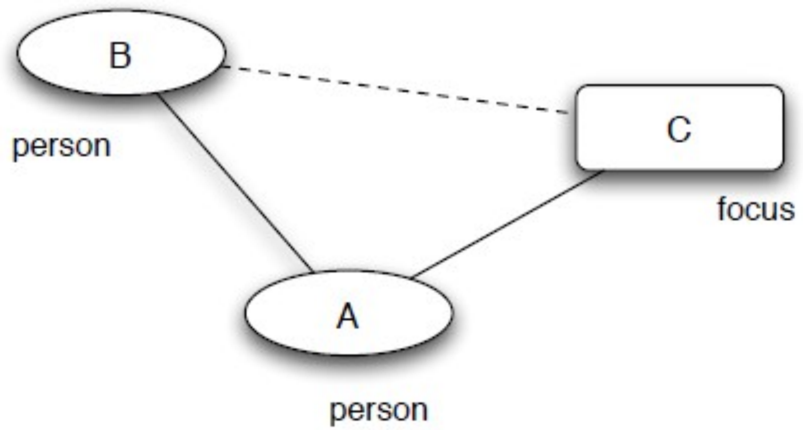
- **Selection**: B links to similar C (common focus)



(b) *Focal closure*

Homophily- Cnt.

- **Membership Closure**: A and B people, C focus
- **Social Influence**: B links to C influenced by A



Snapshot Algorithm

Tracking link formation in large scale datasets based on the above mechanisms

1) Take 2 snapshots of network at different times: **S(1)**, **S(2)**.

2) For each ***k***, find all pairs of nodes in **S(1)** that are not directly connected but have ***k*** common friends.

3) Compute ***T*(*k*)** as the fraction of these pairs connected in **S(2)**.

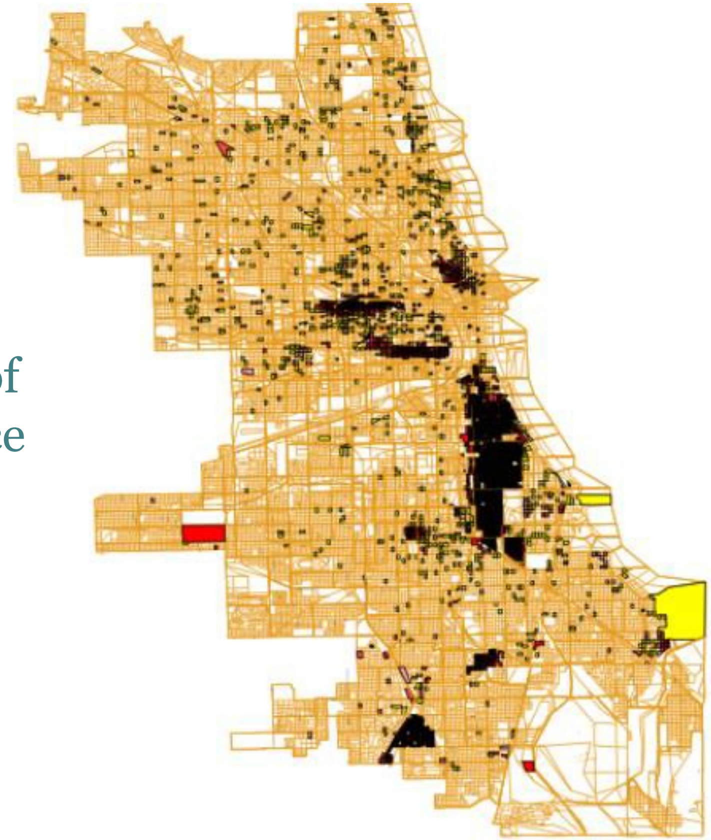
estimate for the probability that a link will form btw 2 people with *k* common friends.

4) Plot ***T*(*k*)** as a function of ***k***

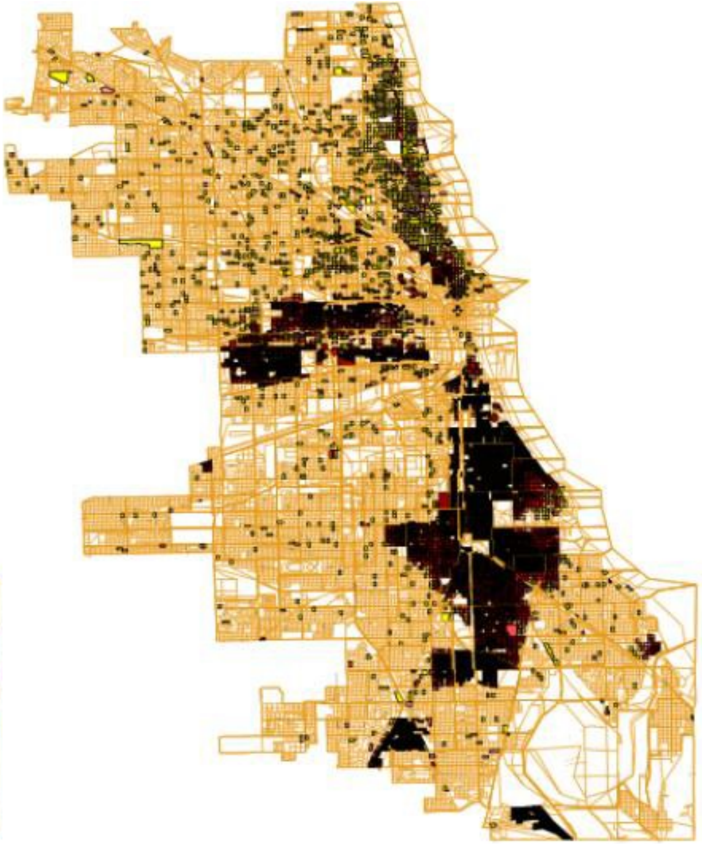
T(0) is the rate of link formation when it does not close a triangle

Spatial Model of Segregation

Schelling model
Local preferences of individuals can produce **unintended global patterns**.



(a) *Chicago, 1940*



(b) *Chicago, 1960*

Effects of homophily in the formation of ethnically and racially **homogeneous neighborhoods** in cities.

People live near others like them!!

Color the map wrt to a given race :

--Lighter: Lowest percentage of the race

--Darker: highest percentage of the race.

Questions?

(Optional) Reading

- Ch.02 Graphs [NCM]
- Ch.03 Strong and Weak Ties [NCM]