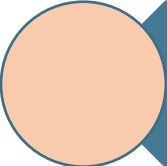
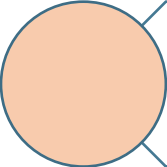
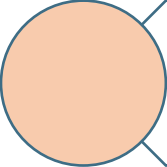
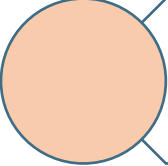


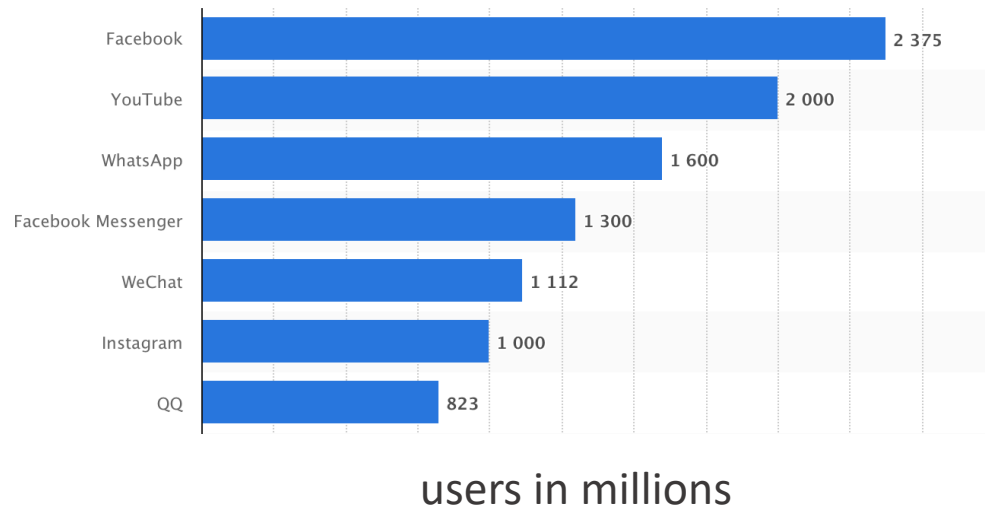
Lecture 13 – Social Network Analytics

Analytics & Applications

Agenda

-  Introduction to Social Network Analytics
-  Shortest Path Problem
-  Node-Level Centrality Metrics
-  Network Metrics

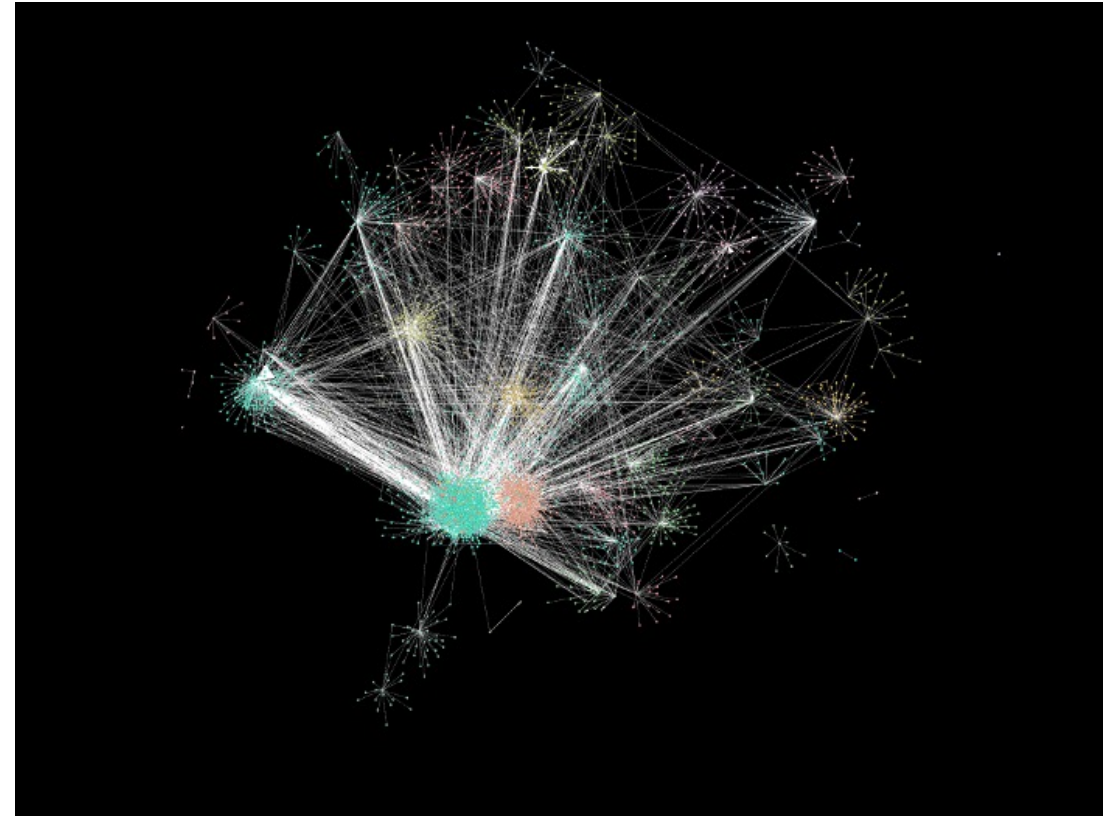
Facebook has 2.375.000.000 users by 2019



- Social Media has connected the world within 20 years
- Social Media has become omnipresent
 - Total Number of Tweets send per day: ~500 million
- These information-based companies quickly began generating a massive amount of data – especially data on **links among people**

What is Social Network Analysis?

- Social network analysis (SNA), also known as network science, is a field of data analytics that uses networks and graph theory to understand social structures. SNA techniques can also be applied to networks outside of the societal realm.

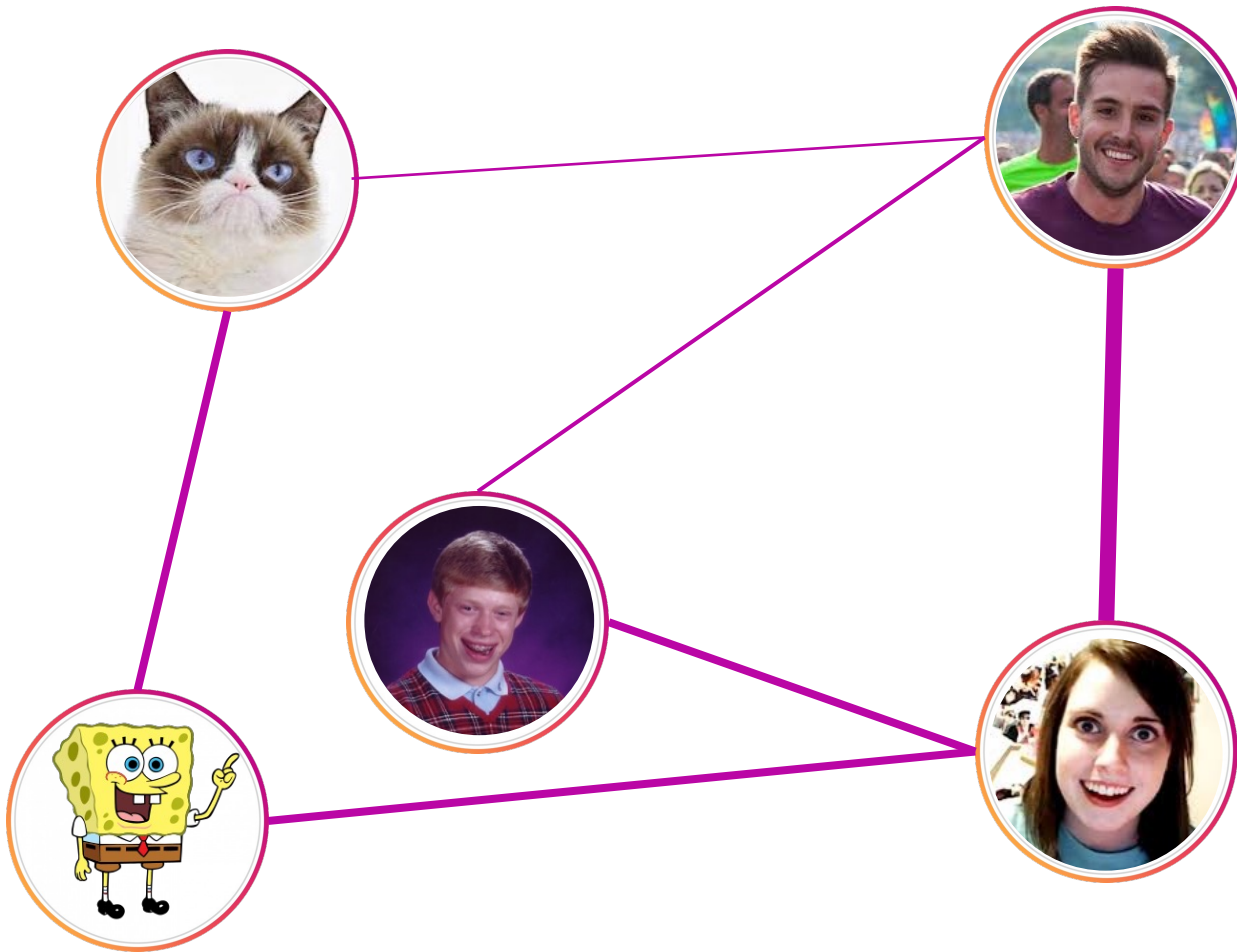


Example Social Network Graph on Community Detection of ISIS Twitter Accounts; see the article [here](#).

What can we do with social network data?

- Targeted Advertising
 - Promote a product/service to users with specific traits
- Product Recommendations
 - Promote products that friends of you liked
- Influencer Marketing
 - Compute 'central' nodes in a social network
- Link Prediction
 - Find people you might know
- Entity Resolution
 - Reconcile multiple data sources and identify underlying matching entities

How do our data look like?



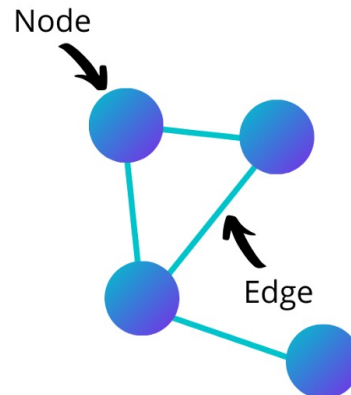
- We can represent entities as nodes and the relationship between entities as edges
- The thickness of an edge may represent the weight of an edge; the weight of an edge is usually represented as a (float) number
- A graph can be **uni-directional** (e. g. Instagram follower) or **bi-directional** (e. g. Facebook friendship)

Image Sources:

<https://imgflip.com/memegenerator>

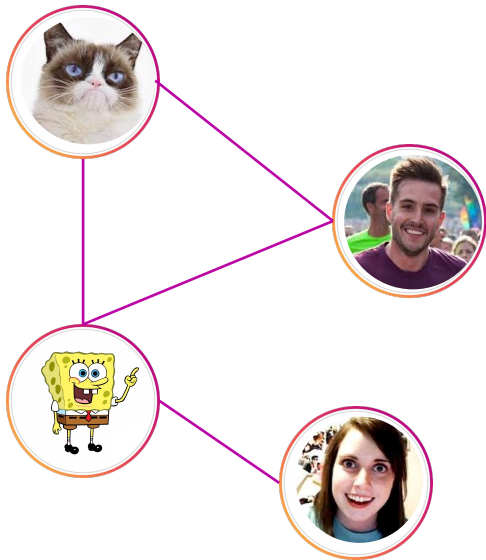
Nodes and Edges

- **Nodes** can represent a variety of **actors**. In internet networks, nodes can represent web pages. In social networks, nodes can represent people. In supply chain networks, nodes can represent organizations. In foreign relations networks, nodes can represent countries. While nodes can represent a variety of things, they are all the things, that have a relationship with another thing.
- **Edges** can represent a variety of **relationships**. In internet networks, edges can represent hyperlinks. In social networks, edges can represent connections. In supply chain networks, edges can represent the transfer of goods. In foreign relations networks, edges can represent policies. Like nodes, edges can represent a variety of things.

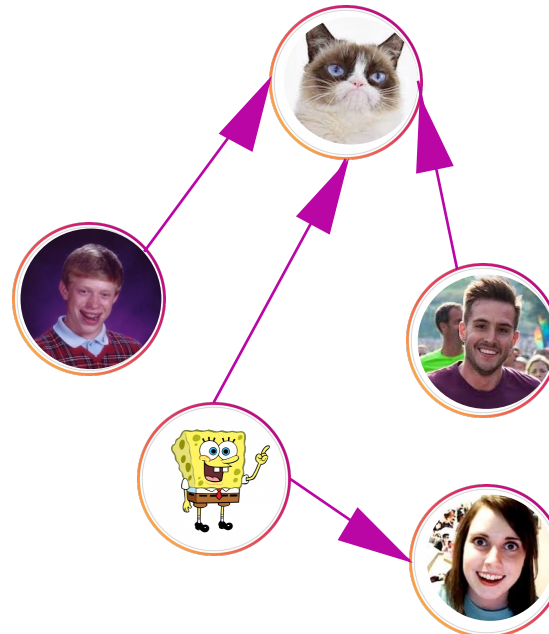


Different types of famous social (media) networks

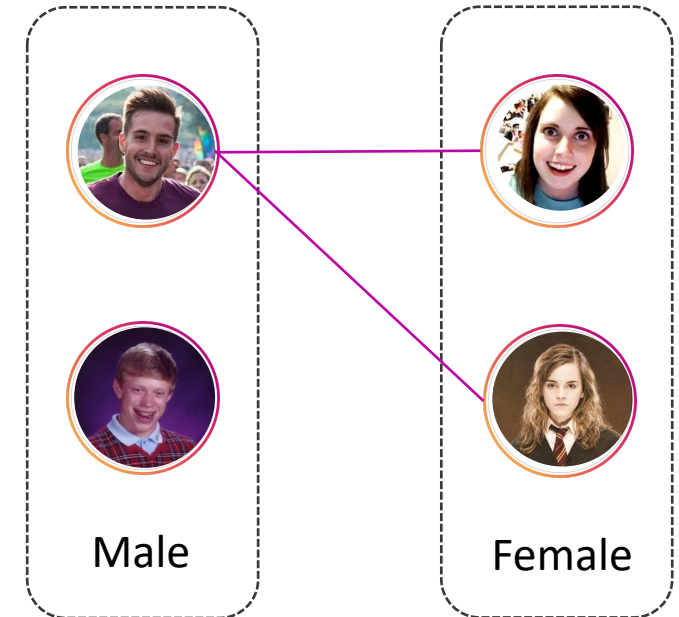
Bi-Directional



Uni-Directional



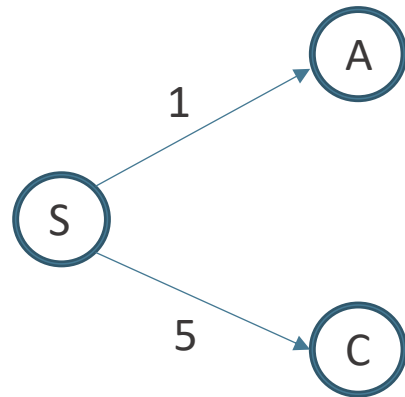
Bipartite Matching



Definition Graph

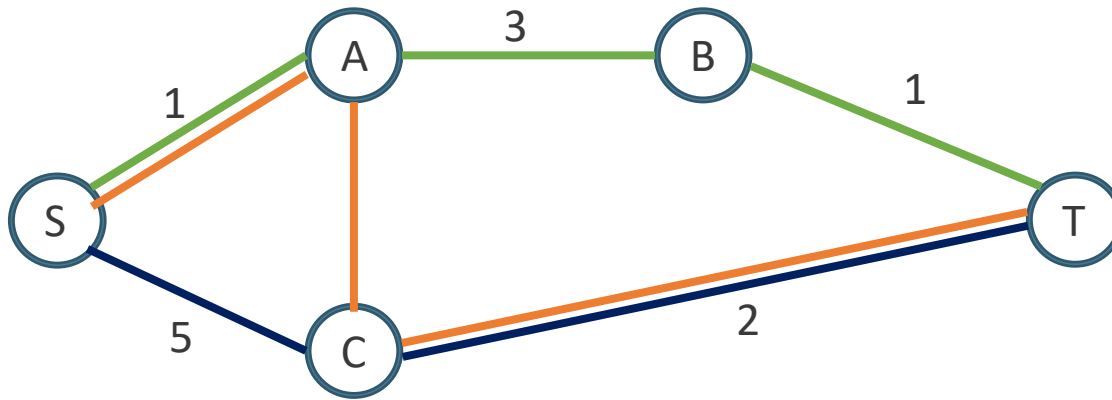
- We denote the set of all nodes/vertices with V ; the cardinality of V with $|V|$.
- An edge $e = (v, w)$ is a connection between two vertices $v, w \in V$.
- The set of edges is denoted by $E \subseteq \{(v, w) \mid (v, w) \in V \times V \text{ and } v \neq w\}$
- The ordered pair $G = (V, E)$ is called graph.
 - We call G **directed graph** if the order of the edges is relevant.
 - We call G **undirected graph** otherwise.

Edge weight represent the strength of a link between nodes



- Edge weight measures the strength between nodes in a graph
- We, therefore, define the cost function as follows
$$cost: V \times V \rightarrow \mathbb{R}.$$
- For example,
 - $cost(s, a) = 1$
 - $cost(s, c) = 5$

A path is a route of nodes needed to go from node A to node B



$$P_1 = \{ (S, A), (A, B), (B, T) \}$$

$$P_2 = \{ (S, A), (A, C), (C, T) \}$$

$$P_3 = \{ (S, C), (C, T) \}$$

- Path are important for measuring distance between nodes.

- A path P is a sequence of edges (e_1, \dots, e_k) for which following equation holds for any $i \in \{1, \dots, k-1\}$

$$e_i \cap e_{i+1} = v_{i+1}$$

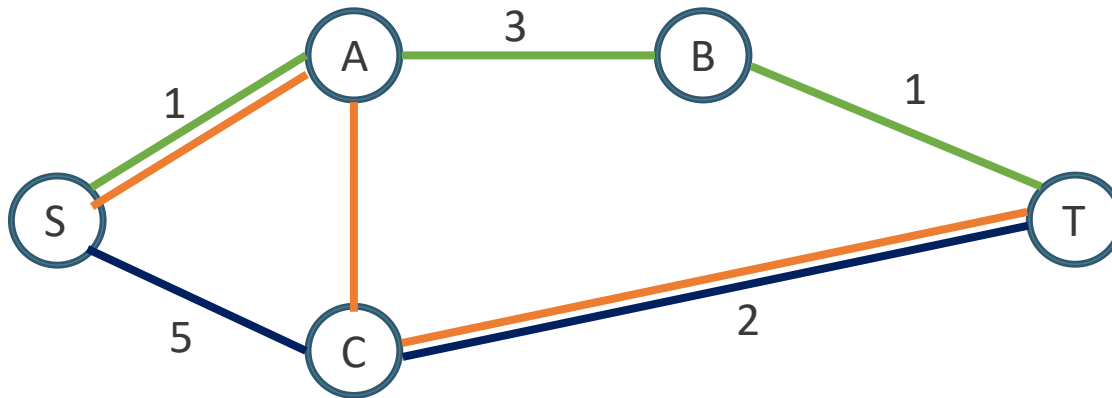
with

$$e_i = (v_i, v_{i+1})$$

$$e_{i+1} = (v_{i+1}, v_{i+2})$$

- The above equation ensures that the path is connected.

A path is a route of nodes needed to go from node A to node B



$$P_1 = \{ (S, A), (A, B), (B, T) \}$$

$$P_2 = \{ (S, A), (A, C), (C, T) \}$$

$$P_3 = \{ (S, C), (C, T) \}$$

- Path are important for measuring distance between nodes.

A path P is a sequence of edges (e_1, \dots, e_k) for which following equation holds for any $i \in \{1, \dots, k-1\}$

$$e_i \cap e_{i+1} = v_{i+1}$$

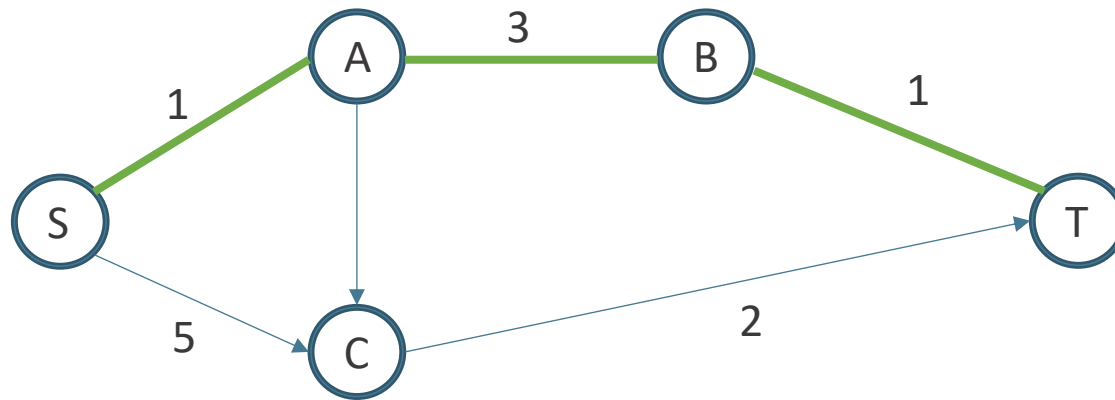
with

$$e_i = (v_i, v_{i+1})$$

$$e_{i+1} = (v_{i+1}, v_{i+2})$$

The above equation ensures that the path is connected.

Path length and path cost



■ $P_1 = \{ (S, A), (A, B), (B, T) \}$

- **Path length** is the number of edges in a path P

$$\ell(P) = \sum_{e_i \in P} 1$$

- **Path cost** calculates the total cost of taking a path P

$$\text{cost}(P) = \sum_{i=1}^k \text{cost}(e_i)$$

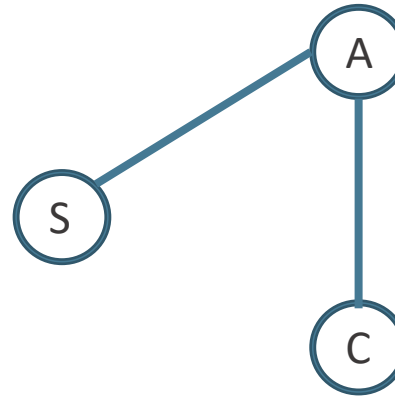
- For example,

- $\ell(P_1) = 3$

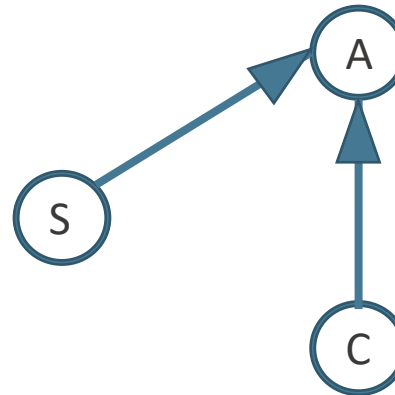
- $\text{cost}(P_1) = 1+3+1 = 5$

Connectivity indicates if each node is reachable

Connected undirected graph

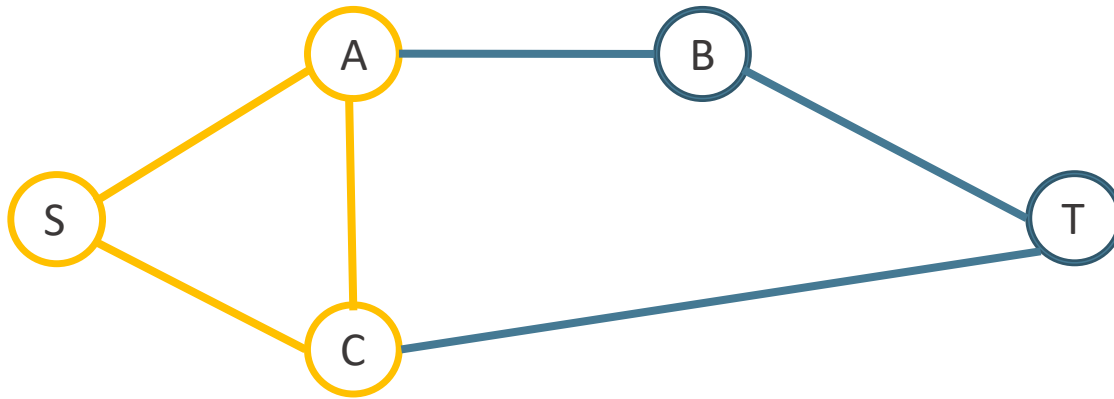


Weakly, but not strongly, connected directed graph



- An **undirected graph** $G = (V, E)$ is **connected** if there is a path p between any pair of vertices $u, v \in V$ and $|V| \geq 0$.
- A **directed graph** is **weakly connected** if replacing all of its directed edges with undirected edges leads to a connected graph.
- A **directed graph** is **strongly connected** if it contains a directed path from $u \rightarrow v$, and $v \rightarrow u$ for every pair of nodes $u, v \in V$.

Cliques



■ $C = \{S, A, C\}$

- A **clique** is a subset of nodes of an undirected graph such that every two distinct nodes are linked.
- Let be $G = (V, E)$ an **undirected graph**. A subset $C \subseteq V$ is called clique if for any pair of nodes $u, v \in C$ there exists an edge $e = (u, v)$ in E .

Agenda



Introduction to Social Network Analytics



Shortest Path Problem

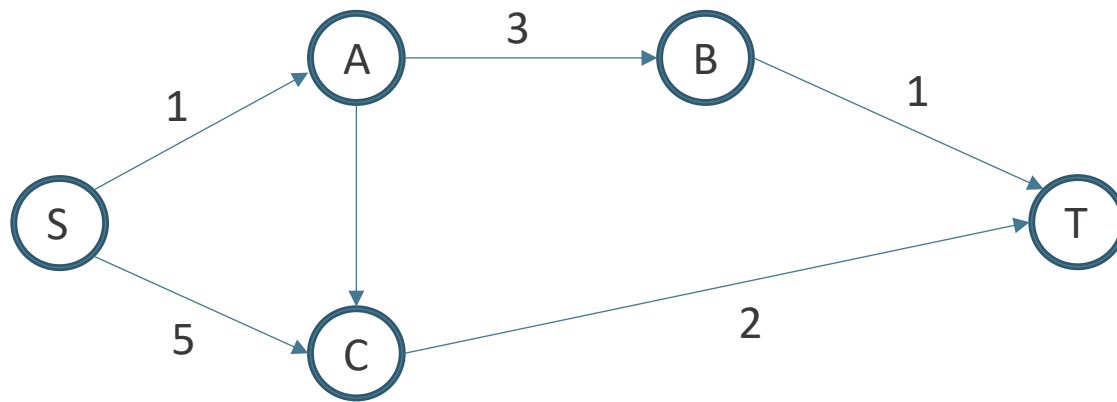


Node-Level Centrality Metrics



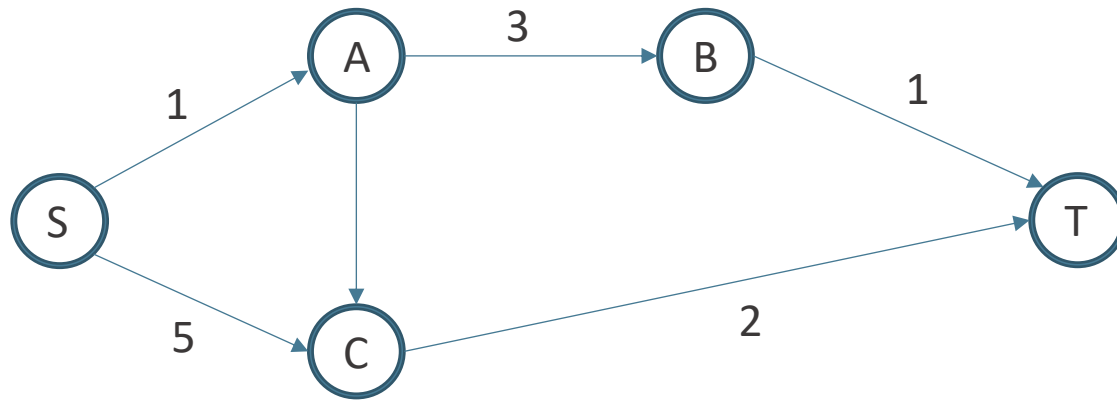
Network Metrics

How do we compute the shortest path between two vertices?



- In social network analytics we are often faced with the task of computing the shortest path between two nodes
- Given a network, how can we compute the shortest path between node s and t?
- Based on the nature of the underlying network, we can use different algorithms in order to solve the SP task.
 - (Directed) Graph with non-negative weights: **Dijkstra Algorithm**
 - Directed Graph with negative weights and without negative cycles: **Ford-Bellman Algorithm**
 - Graph without weights: **Breadth-first-search**

How do we compute the shortest path between two vertices?



- In social network analytics we are often faced with the task of computing the shortest path between two nodes
- Given a network, how can we compute the shortest path between node s and t?
- Based on the nature of the underlying network, we can use different algorithms in order to solve the SP task.
 - (Directed) Graph with non-negative weights: Dijkstra Algorithm
 - Directed Graph with negative weights and without negative cycles: Ford-Bellman Algorithm
 - Graph without weights: Breadth-first-search

Dijkstra Algorithm

- Let be $G = (V, E)$ a connected graph with nonnegative weights.
- The Dijkstra algorithm computes the cost to each node based on a given starting node
- The algorithm starts at a starting node s and progressively selects the currently most favorable paths through the next available nodes.
- If a better, in terms of **cost**, path is found for a node, the path to this node is updated.
- Having visited all nodes and no better way was found for each node, the algorithm terminates.

Dijkstra Algorithm Pseudo-Code

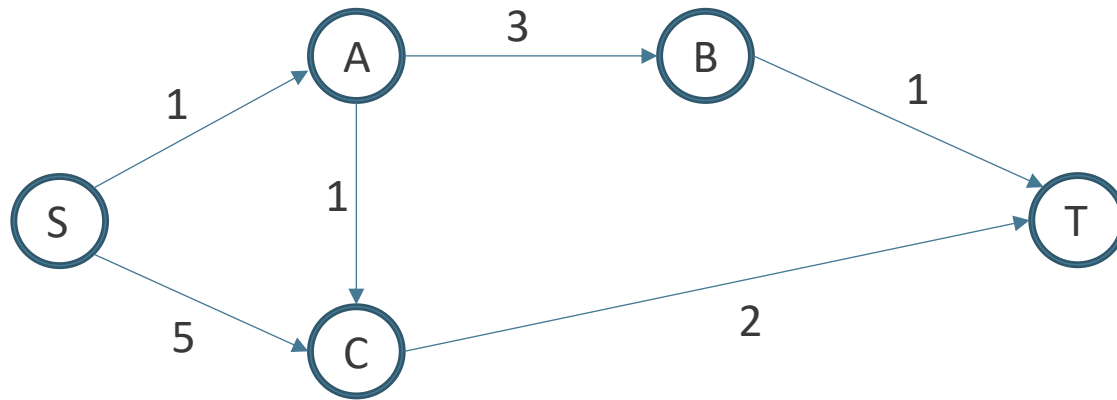
```
function dijkstra(G = (V, E), s)
  for v in V:
    if(v == s):
      v.dist = 0
    else:
      v.dist = infinity
      v.prev = null

  PQ = PriorityQueue(V)

  while(PQ not empty):
    u = PQ.removeMin()
    for all edges(u, v):
      if(v in PQ && v.dist > u.dist + cost(u, v))
        v.dist = u.dist + cost(u, v)
        v.prev = u
        PQ.replaceKey(v, v.dist)
```

1. Input parameter graph (nodes and edges) and start node s
2. Initialize all nodes by setting the path cost to infinity and the previous node to **null** (means not processed yet). However, we set the cost for the starting point to **zero**.
3. Initialize a priority queue based with
 - Keys: Nodes
 - Values: Current shortest path cost
4. Terminate algorithm if all nodes has been processed and no improvements are possible
5. Pick next node u with minimal cost from PQ and check for each outgoing edge (u, v) if we can improve the path costs. *Note that v needs to be element of PQ (otherwise this node is already processed)*
6. If an improvement is possible, update the path cost to node v and set the previous node to u

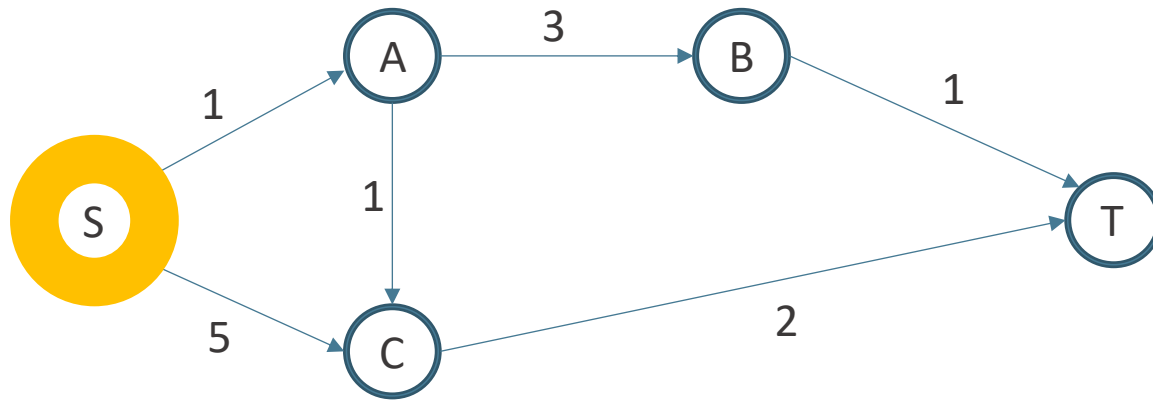
Find the shortest path from node S to T with Dijkstra algorithm



Step	S	A	C	B	T
Init	$(0, \emptyset)$	(∞, \emptyset)	(∞, \emptyset)	(∞, \emptyset)	(∞, \emptyset)

- $(0, \emptyset)$ is equivalent to A.dist = 0 and A.prev = null
- $(5, S)$ is equivalent to A.dist = 5 and A.prev = S
- and so on.

Find the shortest path from node S to T with Dijkstra algorithm

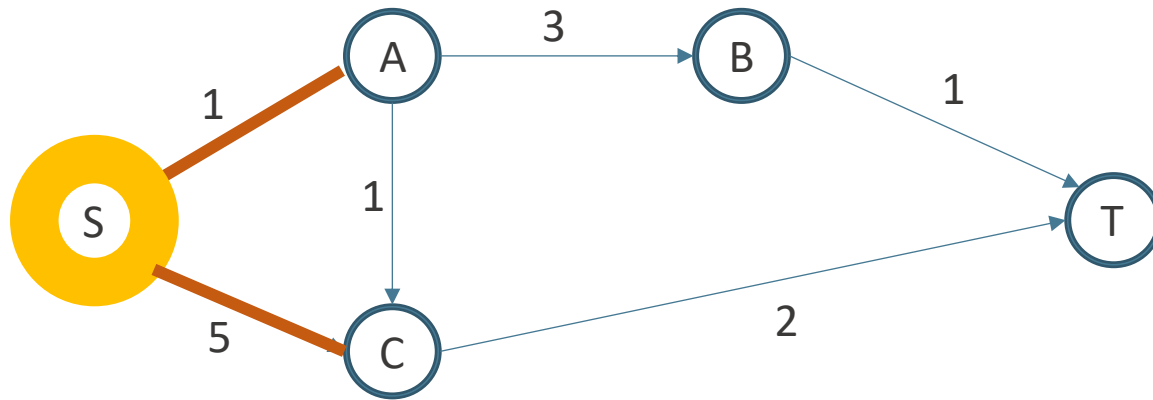


$PQ = \{ (S, 0), (A, \infty), (C, \infty), (B, \infty), (T, \infty) \}$

$PQ.removeMin()$ will set u to S

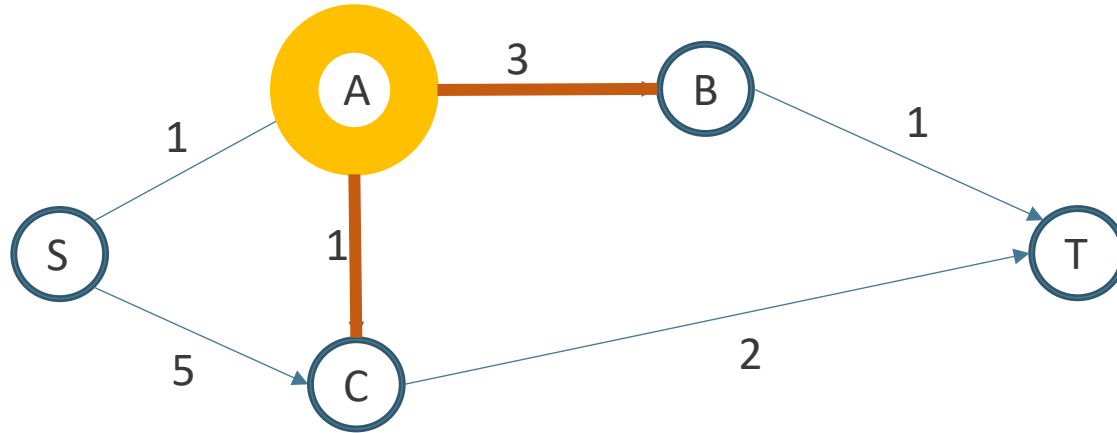
Step	S	A	C	B	T
Init	(0, \emptyset)	(∞ , \emptyset)	(∞ , \emptyset)	(∞ , \emptyset)	(∞ , \emptyset)

Find the shortest path from node S to T with Dijkstra algorithm



Step	S	A	C	B	T
Init	(0, ∅)	(∞, ∅)	(∞, ∅)	(∞, ∅)	(∞, ∅)
1	(0, ∅)	(1, S)	(5, S)	(∞, ∅)	(∞, ∅)

Find the shortest path from node S to T with Dijkstra algorithm

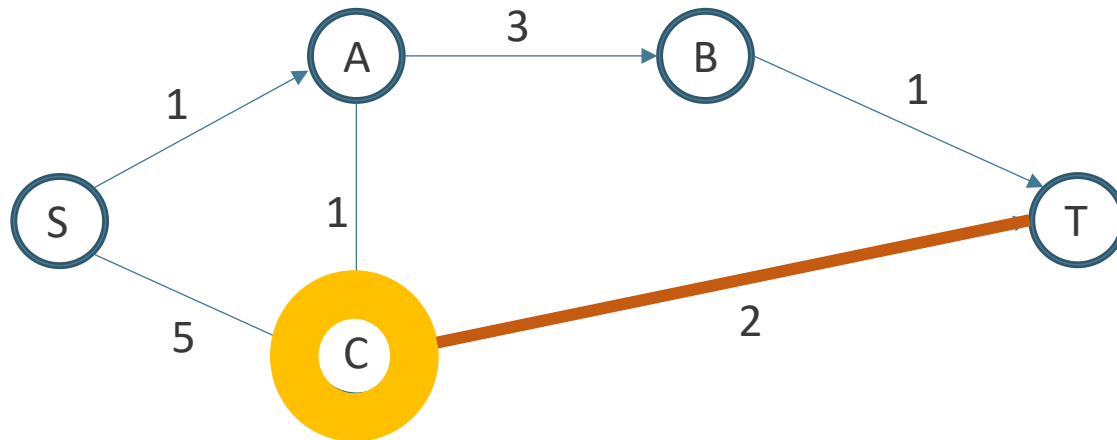


$PQ = \{ (A, 1), (C, 5), (B, \infty), (T, \infty) \}$

$\underbrace{\hspace{1.5cm}}$
PQ.removeMin() will set u to A

Step	S	A	C	B	T
Init	$(0, \emptyset)$	(∞, \emptyset)	(∞, \emptyset)	(∞, \emptyset)	(∞, \emptyset)
1	$(0, \emptyset)$	$(1, S)$	$(5, S)$	(∞, \emptyset)	(∞, \emptyset)
2	$(0, \emptyset)$	$(1, S)$	$(2, A)$	$(4, A)$	(∞, \emptyset)

Find the shortest path from node S to T with Dijkstra algorithm

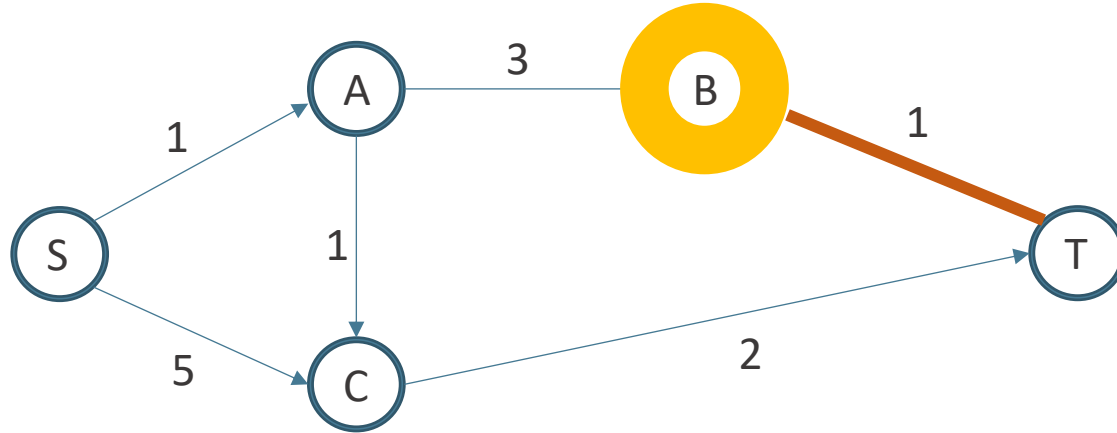


$PQ = \{ (C, 2), (B, 4), (T, \infty) \}$

$PQ.removeMin()$ will set u to C

Step	S	A	C	B	T
Init	$(0, \emptyset)$	(∞, \emptyset)	(∞, \emptyset)	(∞, \emptyset)	(∞, \emptyset)
1	$(0, \emptyset)$	$(1, S)$	$(5, S)$	(∞, \emptyset)	(∞, \emptyset)
2	$(0, \emptyset)$	$(1, S)$	$(2, A)$	$(4, A)$	(∞, \emptyset)
3	$(0, \emptyset)$	$(1, S)$	$(2, A)$	$(4, A)$	$(4, C)$

Find the shortest path from node S to T with Dijkstra algorithm



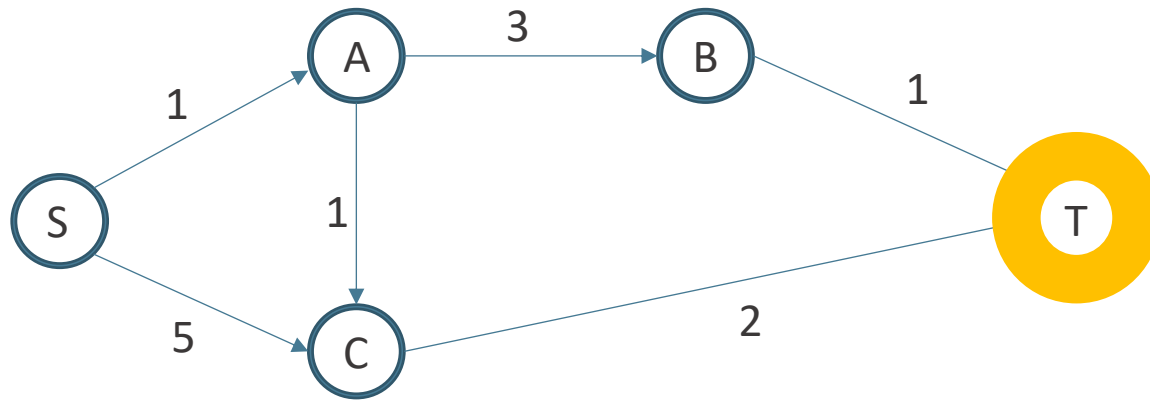
$PQ = \{ (B, 4), (T, 4) \}$

$PQ.removeMin()$ will set u to B

Step	S	A	C	B	T
Init	$(0, \emptyset)$	(∞, \emptyset)	(∞, \emptyset)	(∞, \emptyset)	(∞, \emptyset)
1	$(0, \emptyset)$	$(1, S)$	$(5, S)$	(∞, \emptyset)	(∞, \emptyset)
2	$(0, \emptyset)$	$(1, S)$	$(2, A)$	$(4, A)$	(∞, \emptyset)
3	$(0, \emptyset)$	$(1, S)$	$(2, A)$	$(4, A)$	$(4, C)$
4	$(0, \emptyset)$	$(1, S)$	$(2, A)$	$(4, A)$	$(4, C)$

No improvement!

Find the shortest path from node S to T with Dijkstra algorithm

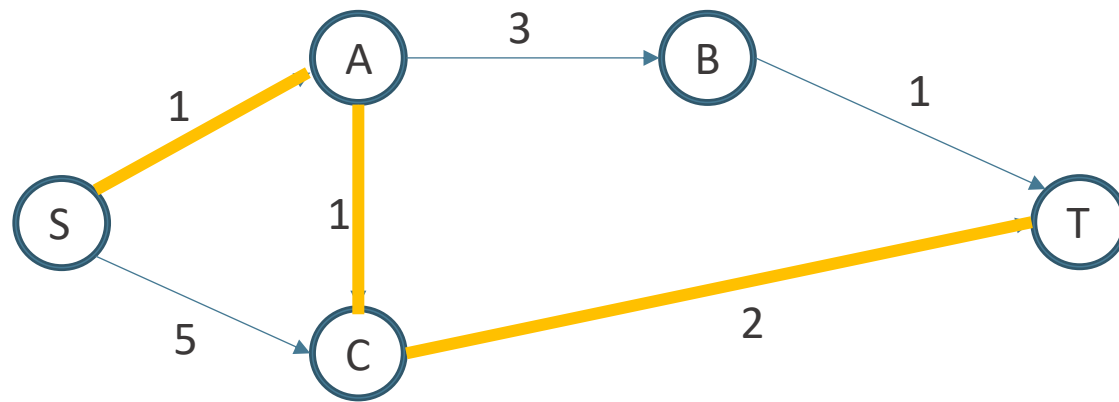


$PQ = \{ (T, 4) \}$

$PQ.removeMin()$ will set u to T

Step	S	A	C	B	T
Init	$(0, \emptyset)$	(∞, \emptyset)	(∞, \emptyset)	(∞, \emptyset)	(∞, \emptyset)
1	$(0, \emptyset)$	$(1, S)$	$(5, S)$	(∞, \emptyset)	(∞, \emptyset)
2	$(0, \emptyset)$	$(1, S)$	$(2, A)$	$(4, A)$	(∞, \emptyset)
3	$(0, \emptyset)$	$(1, S)$	$(2, A)$	$(4, A)$	$(4, C)$
4	$(0, \emptyset)$	$(1, S)$	$(2, A)$	$(4, A)$	$(4, C)$
5	$(0, \emptyset)$	$(1, S)$	$(2, A)$	$(4, A)$	$(4, C)$

Find the shortest path from node S to T with Dijkstra algorithm



PQ = \emptyset

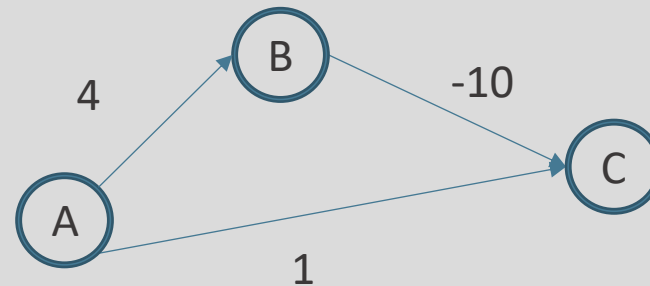
Step	S	A	C	B	T
Init	(0, \emptyset)	(∞ , \emptyset)	(∞ , \emptyset)	(∞ , \emptyset)	(∞ , \emptyset)
1	(0, \emptyset)	(1, S)	(5, S)	(∞ , \emptyset)	(∞ , \emptyset)
2	(0, \emptyset)	(1, S)	(2, A)	(4, A)	(∞ , \emptyset)
3	(0, \emptyset)	(1, S)	(2, A)	(4, A)	(4, C)
4	(0, \emptyset)	(1, S)	(2, A)	(4, A)	(4, C)
5	(0, \emptyset)	(1, S)	(2, A)	(4, A)	(4, C)

Thus, the shortest path between nodes S and T is **S -> A -> C -> T** with cost **4**.

Shortest Path Problem



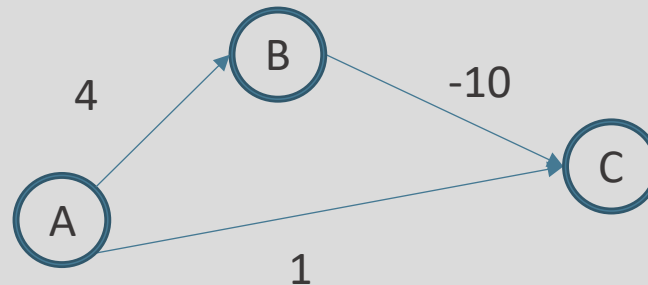
Using Dijkstra algorithm find the shortest path from A to C and its cost for the following graph.





Shortest Path Problem

Using Dijkstra algorithm find the shortest path from A to C and its cost for the following graph.



Dijkstra's algorithm starting from A discovers C and B. In the next step, it visits C and marks it as visited.

Since the node is marked visited, the algorithm assumes that the path developed to this node (A->C) is the shortest.

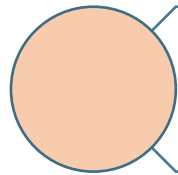
But actually, the shortest path from A to C is A->B->C (given that the negative weighted edge has some significance).



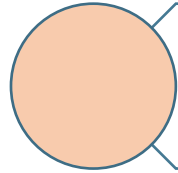
Negative edge weights causes the Dijkstra algorithm to compute a wrong shortest path.

The problem with Dijkstra's algorithm is that it is believed that all costs in the given graph are non-negative, so adding any positive number on a node that has already been visited will never change its minimality.

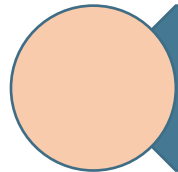
Agenda



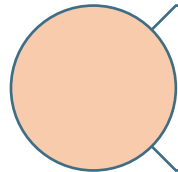
Introduction to Social Network Analytics



Shortest Path Problem

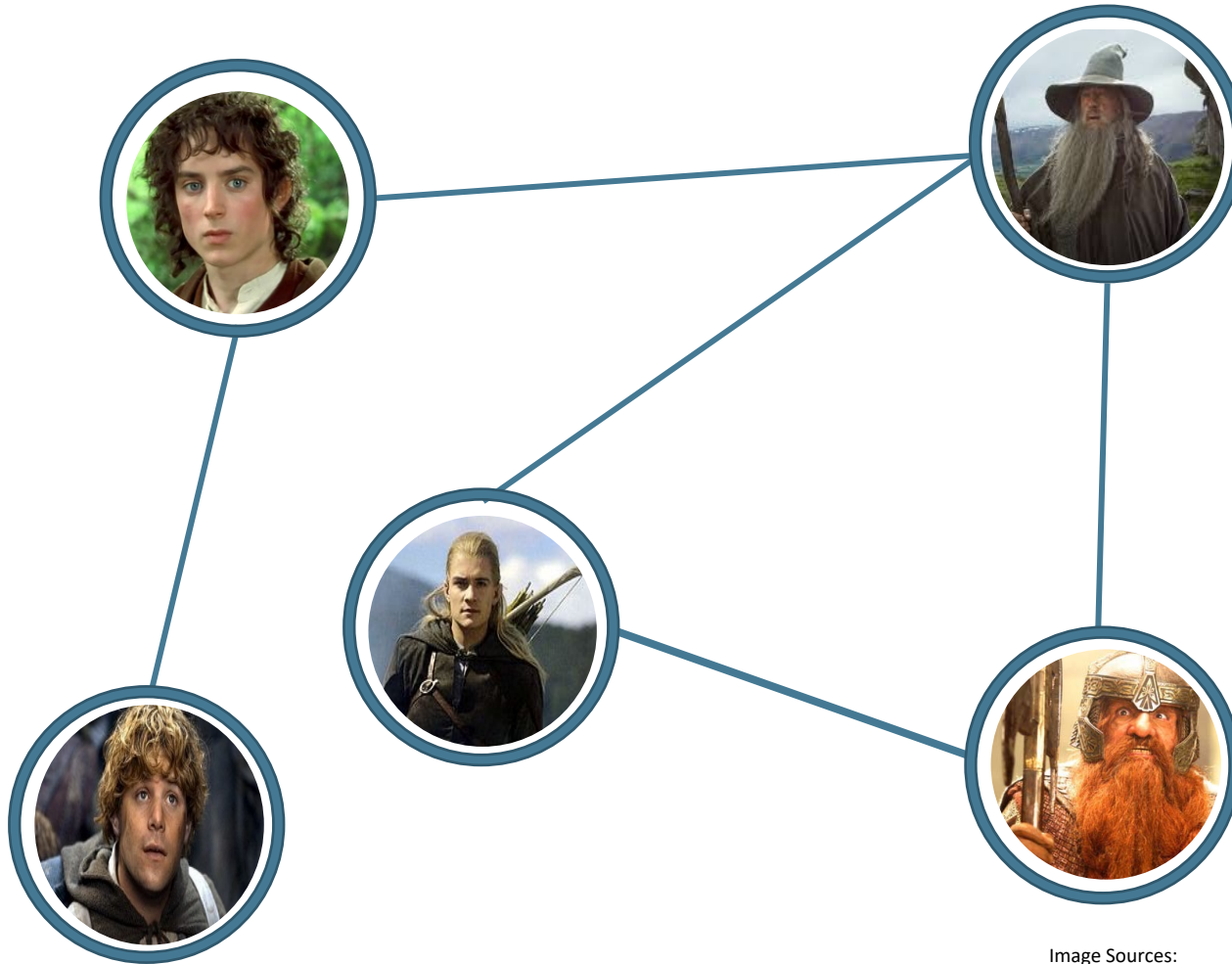


Node-Level Centrality Metrics



Network Metrics

Which nodes are important?



- The influence of a node in a (social) network is fundamental to many real-world issues.
- For example, to
 - identify terror networks
 - find Influencer on Instagram
 - compute relevant websites based on a search query (Google)
 - analyze political networks in social sciences

Image Sources:

<https://www.mdr.de/kultur/ian-mckellen-geburtstag-gandalf-100.html>

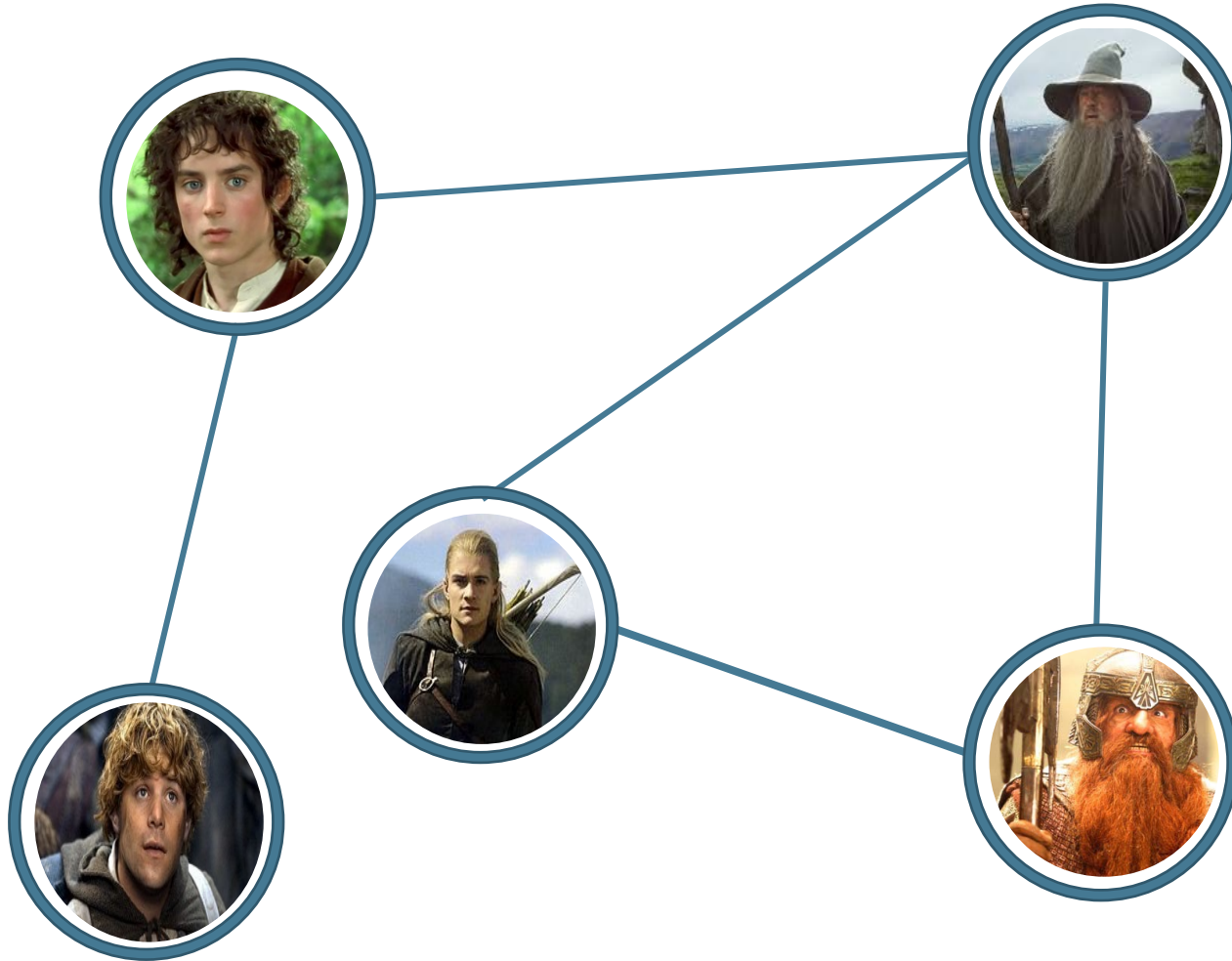
<https://www.herr-der-ringe-film.de/v3/de/filme/darsteller/gefaehrten/gimli/gimli-1.php>

<https://www.watson.ch>

https://en.wikipedia.org/wiki/Frodo_Baggins

<https://www.theonering.net/>

Obviously, Gandalf was one of the influencing characters in LOTR



- There are two common methods which we can use to compute the influence of a node to the entire network

- Degree Centrality
- Eigenvector Centrality

Adjacency matrix of the LOTR network

	Gandalf	Gimli	Legolas	Frodo	Sam
Gandalf	0	1	1	1	0
Gimli	1	0	1	0	0
Legolas	1	1	0	0	0
Frodo	1	0	0	0	1
Sam	0	0	0	1	0

- This adjacency matrix represents the example network of the LOTR characters.
 - "1" means that the nodes of the associate characters are connected
- The underlying graph is a bidirectional graph without weights.
- Note that this adjacency matrix is **symmetric** which means entry (i , j) equals entry (j , i).

Degree centrality

- Degree centrality is a measure that counts how many neighbors a node has.
- **Motivation:** A node is important if it has many neighbors, or, in the directed case, if there are many other nodes that link to it, or if it links to many other nodes.
- Let be $G = (V, E)$ a graph and $\delta: V \rightarrow N_{\geq 0}$ a function that yields the numbers of neighbors of a node. For any node $v \in V$ we can compute the degree centrality with $\delta(v)$.
- In the case of a directed graph we can compute the degree centrality based on the outgoing edges and incoming edges for a node. Thus, denote $\delta^+: V \rightarrow N_{\geq 0}$ as the degree centrality function for outgoing edges and $\delta^-: V \rightarrow N_{\geq 0}$ as the degree centrality function for incoming edges.
- In a bidirectional graph $\delta = \delta^+ = \delta^-$ holds.

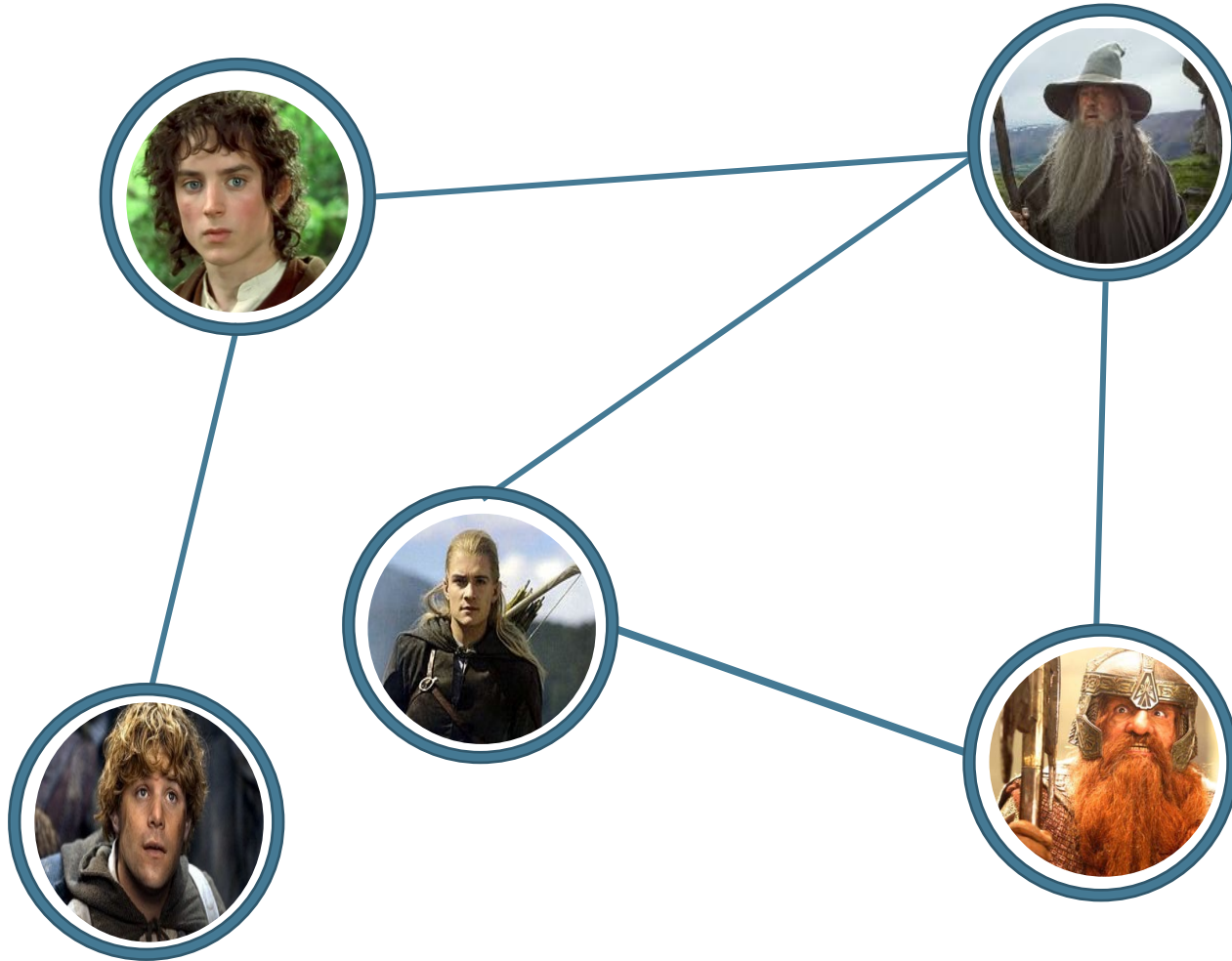
Example of degree centrality in a bidirectional graph without weights

$\delta^+ (Gandalf)$
equals the sum of
the row

	Gandalf	Gimli	Legolas	Frodo	Sam
Gandalf	0	1	1	1	0
Gimli	1	0	1	0	0
Legolas	1	1	0	0	0
Frodo	1	0	0	0	1
Sam	0	0	0	1	0

$\delta^- (Gandalf) = \delta(Gandalf)$ equals the sum of the column

Knowing important people makes me more important



- Degree centrality solely counts the number of neighbors and does not consider the fact that nodes that are linked to “important” nodes are also more important.
- **Eigenvector centrality** is a method which takes this consideration into account.

Eigenvector Centrality

- Eigenvector centrality is a measure of the influence of a node in a network.
- **Motivation:** A node is important if it is connected to many influencing nodes.
- Let be $G = (V, E)$ a graph and $A = (a_{u,v})$ the associate adjacency matrix with $a_{u,v} = 1$ if node u is linked to node v ; and 0 otherwise.
- The relative centrality score can be defined as:

$$x_u = \frac{1}{\lambda} \sum_{v \in \delta(u)} x_v = \frac{1}{\lambda} \sum_{v \in V} a_{u,v} x_v = \frac{1}{\lambda}$$

which leads to the eigenvector formulation

$$Ax = \lambda x.$$

- Solving this eigenvector formulation yields the eigenvector x whose u^{th} component gives the relative centrality score for the node u (x_u).
- Note, that we only consider the highest eigenvalue and its associated eigenvector since we require that $x_u \geq 0$.

Solving the eigenvector centrality formulation with power iteration

- There are many methods to solve the equation $Ax = \lambda x$, one of which we look at, namely **Power Iteration**.
- Let be $G = (V, E)$ a graph and $A = (a_{u,v})$ the associate adjacency matrix with $a_{u,v} = 1$ if node u is linked to node v ; and 0 otherwise.
- Also, let $b_0 \in \mathbb{R}^{|V|}$ a random vector. The power iteration algorithm is defined as

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|_2}$$

- $\|v\|_2$ denotes the euclidean norm that is $\sqrt{v \bullet v^T}$.

Computing the eigenvector centrality for the LOTR network (1/3)

Initialize

$$b_0 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 5 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Step 1

$$Ab_0 = \begin{pmatrix} 7 \\ 4 \\ 3 \\ 6 \\ 2 \end{pmatrix}$$

$$||Ab_0|| \approx 10.67$$

$$b_1 = \begin{pmatrix} 0.65561007 \\ 0.37463432 \\ 0.28097574 \\ 0.56195149 \\ 0.18731716 \end{pmatrix}$$

Computing the eigenvector centrality for the LOTR network (2/3)

Step 2

$$Ab_1 = \begin{pmatrix} 1.21756156 \\ 0.93658581 \\ 1.03024439 \\ 0.84292723 \\ 0.56195149 \end{pmatrix}$$

$$\|Ab_1\| \approx 2.10887847$$

$$b_2 = \begin{pmatrix} 0.57735027 \\ 0.44411559 \\ 0.48852715 \\ 0.39970403 \\ 0.26646936 \end{pmatrix}$$

Step 3

$$Ab_2 = \begin{pmatrix} 1.33234678 \\ 1.06587742 \\ 1.02146586 \\ 0.84381962 \\ 0.39970403 \end{pmatrix}$$

$$\|Ab_2\| \approx 2.1969137$$

$$b_3 = \begin{pmatrix} 0.60646294 \\ 0.48517036 \\ 0.46495492 \\ 0.3840932 \\ 0.18193888 \end{pmatrix}$$

Computing the eigenvector centrality for the LOTR network (3/3)

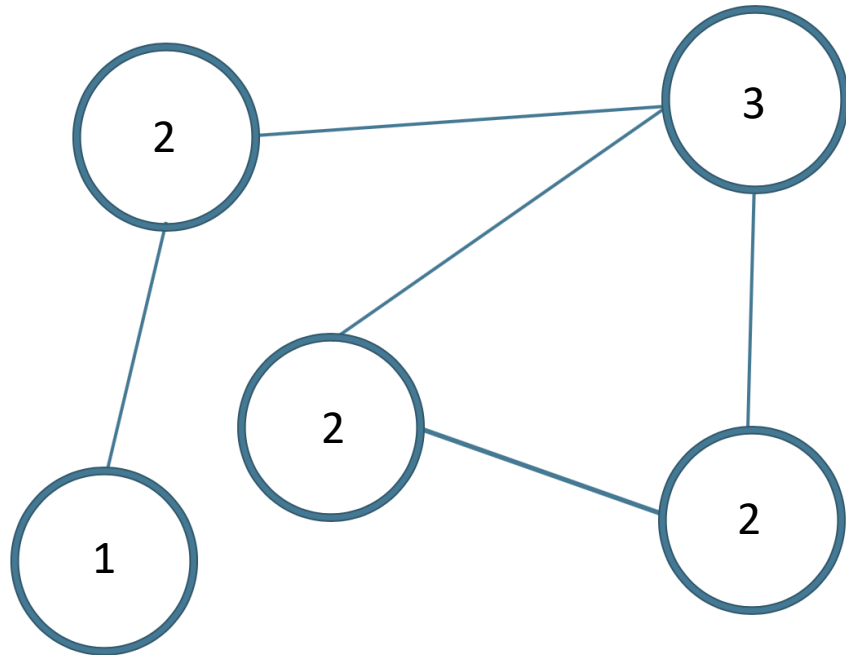
$$b_3 = \begin{pmatrix} 0.60646294 \\ 0.48517036 \\ 0.46495492 \\ 0.3840932 \\ 0.18193888 \end{pmatrix} \approx \begin{pmatrix} 0.60370353 \\ 0.49715368 \\ 0.49715368 \\ 0.34248528 \\ 0.1546684 \end{pmatrix} = v_1$$

b_3 is the eigenvector approximation after 3 steps with the power iteration method

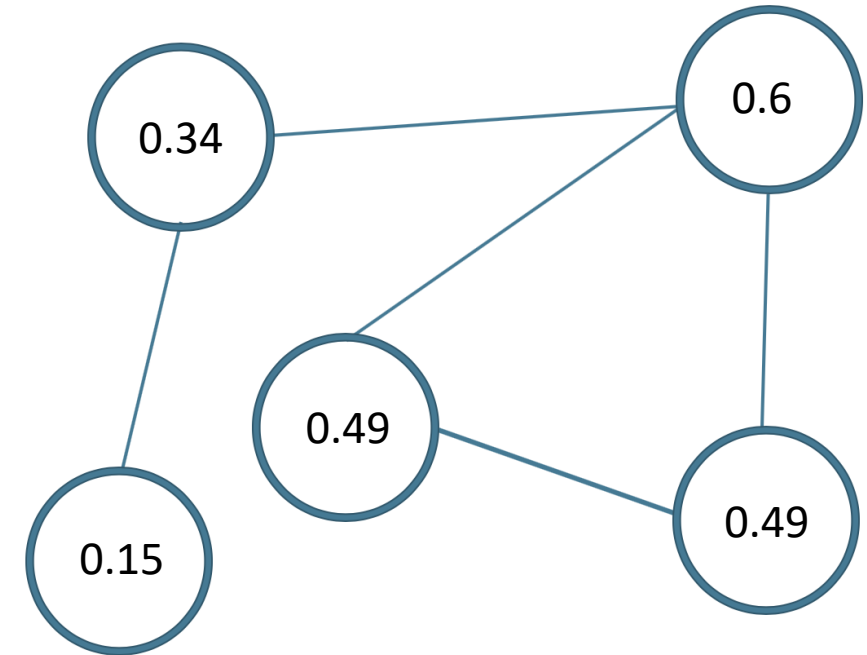
v_1 is the 'real' eigenvector of the adjacency matrix A with the highest eigenvalue $\lambda = 2.214$.

Comparing degree vs eigenvector centrality for the LOTR network

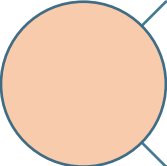
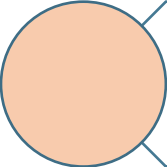
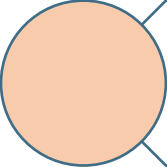
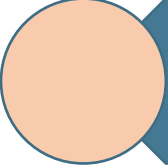
Degree Centrality



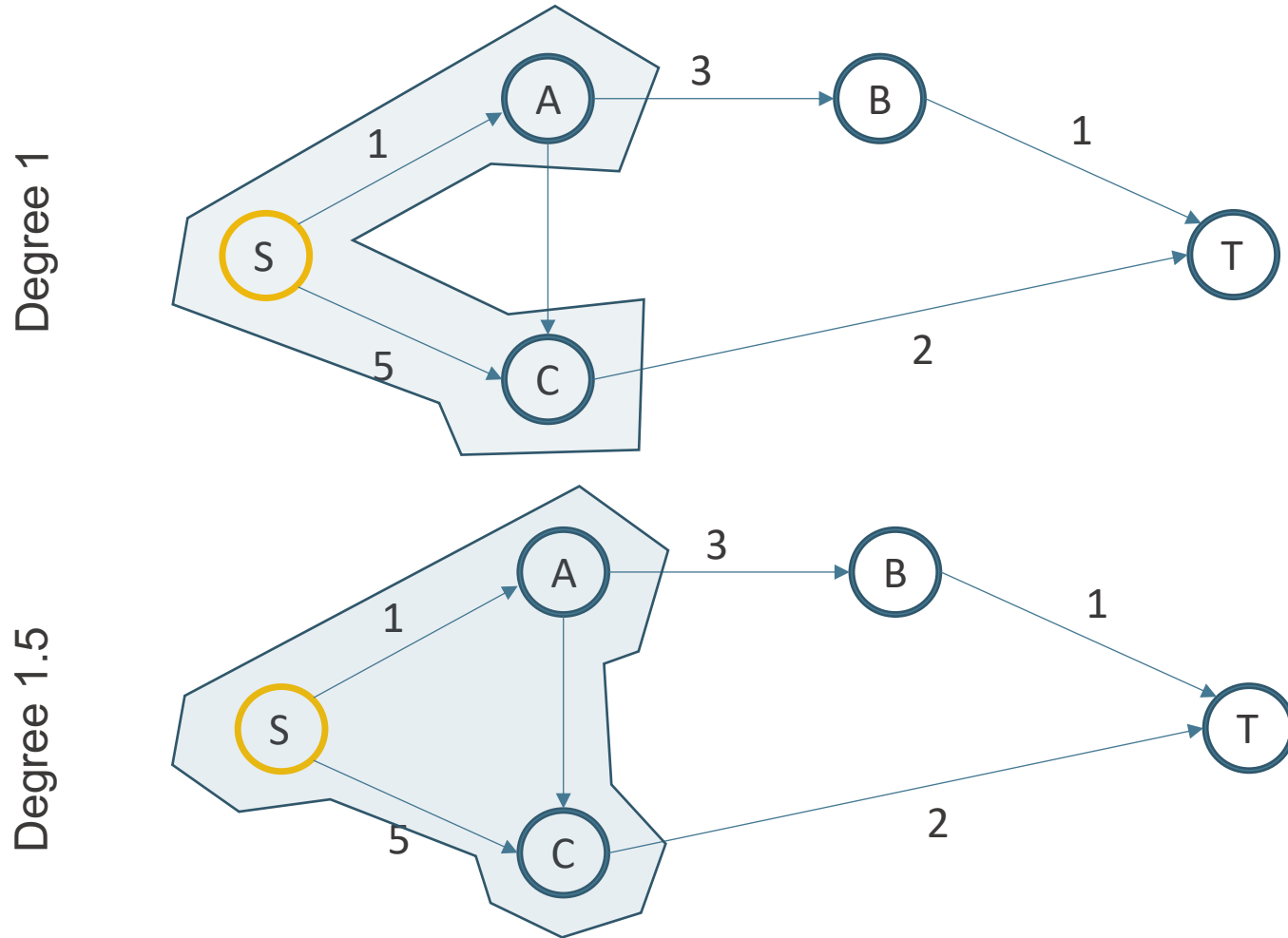
Eigenvector Centrality



Agenda

-  Introduction to Social Network Analytics
-  Shortest Path Problem
-  Node-Level Centrality Metrics
-  Network Metrics

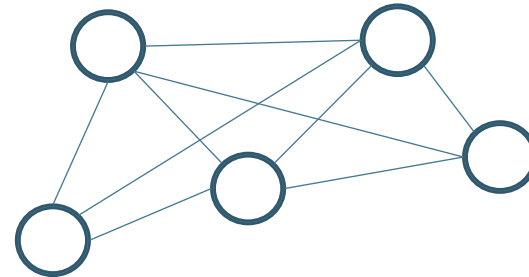
Egocentric Network



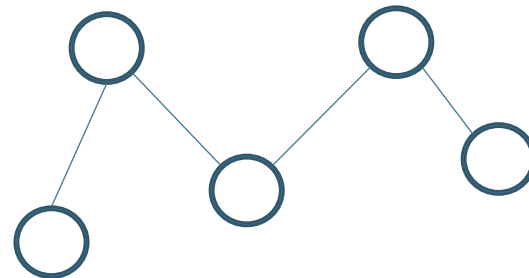
- It is often important to gain information that comes only from the analysis of **individuals** and their **connections**.
- An egocentric network is the network of connections centered around an individual node.
- A degree 1 egocentric network consists of all the edges connected to the node
- A degree 1.5 egocentric network is the network of all those nodes, plus all the edges among them.

Network Metrics

Dense Network

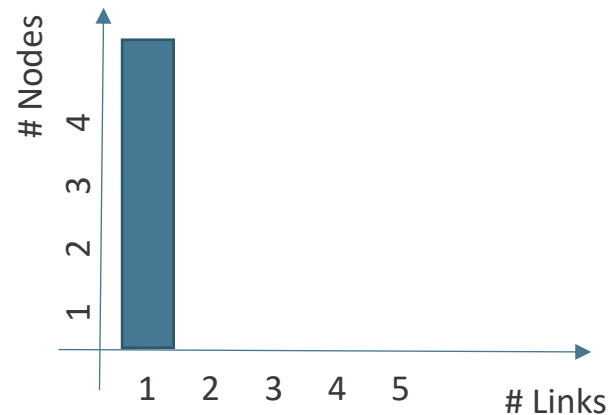
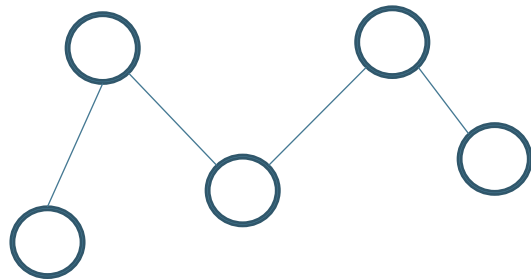
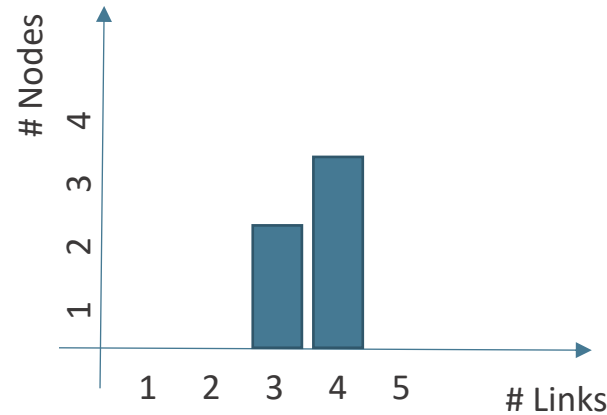
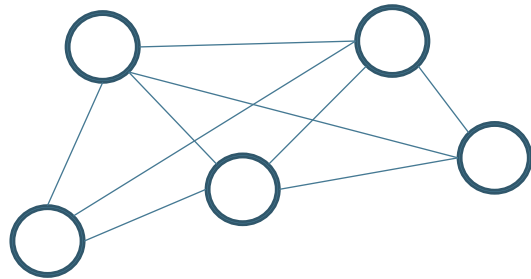


Sparse Network



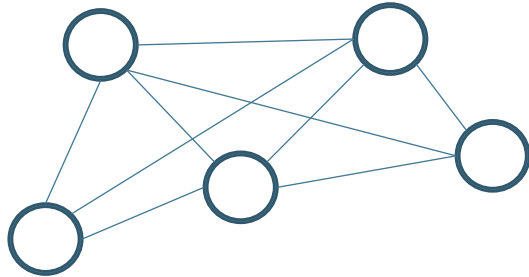
- To this point we have discussed metrics that apply to nodes and edges. However, we can also measure attributes of the network as a whole.
- **Degree distribution** describes the range of connectedness of the nodes
- **Network density** is another way to describe the connectedness of the network

Degree distribution

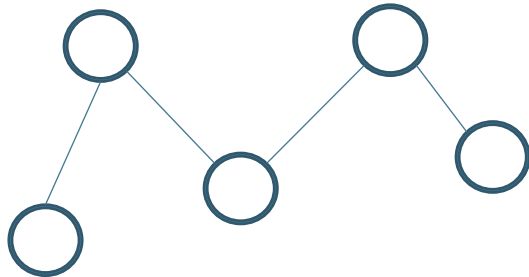


- **Degree distribution** describes the range of connectedness of the nodes
- Count how many nodes have N links, $N-1$ links, $N-2$ links and so on.
- Hence, the resulting degree distribution should
 - concentrate on a high number of links for a **dense network**,
 - concentrate on low number of links for a **sparse network**

Network density



$$\rho(G) = \frac{2 * 9}{5 * 4} = \frac{18}{20} = 0.9$$



$$\rho(G) = \frac{2 * 4}{5 * 4} = \frac{8}{20} = 0.4$$

- **Network density** focuses on the edges in the network, not on the nodes.
- The density is given by the ratio of actual number of edges to the maximum number of potential edges.
- For a **directed** graph with n nodes, there can be a maximum of $n(n-1)$ edges.
- For an **undirected** graph with n nodes, the number is $n(n-1)/2$.
- Thus, we can compute the density $\rho: G \rightarrow \mathbb{Q}$ as follows
 - Directed: $\rho(G) = \frac{|E|}{n(n-1)}$
 - Undirected: $\rho(G) = \frac{2 \cdot |E|}{n(n-1)}$

EVENT FOR MASTER STUDENTS

THE ENERGY TRANSITION NEEDS YOU!



Work with us on one of the 21st century's defining challenges: Energy!

**Whether as a PhD Student, research associate or student research assistant:
At EWI, you can make the difference**



How do we need to shape energy markets to achieve national, European, and global climate goals?



Will hydrogen become the number one energy carrier in the future?



And many more questions to be answered

Interested? Let's meet!



When? 1.2.2023, 19:00-20:00

Where? Conference room of the seminar building

Agenda?

- About us
- Career opportunities at EWI
- Current research on battery storage and the potential hydrogen economy
- "Certificate in Future Energy Business"
- Q&A

After the event



Get-together with snacks and drinks!

We kindly ask for a non-binding registration to

Melanie.Tillmann@ewi.uni-koeln.de until January 26, 2023!

CALL FOR APPLICATIONS

10-20 HOURS AT KVB

For our partner the KVB, we are searching for a motivated student that wants to apply Data Science skills in an ongoing joint project on mobility. For the project, you would need to have skills in python programming and a completed Bachelor. Additionally, sufficient german skills are required.

Role: Research Assistant

- Fulfill Software tasks of the KVB in the joint research project
- Implementation of the ideas of the research project for the KVB

The research project can be found under this link (<https://miaas.de/>). If you are interested and have questions, you can send an email to is3-teaching@wiso.uni-koeln.de

Please send any inquiries and your applications (CV summarized in one .pdf file or link to your LinkedIn Profile + overview of grades) to is3-teaching@wiso.uni-koeln.de



Contact



For general questions and enquiries on **research**, **teaching**, **job openings** and new **projects** refer to our website at www.is3.uni-koeln.de



For specific enquiries regarding this course contact us by sending an email to the **IS3 teaching** address at is3-teaching@wiso.uni-koeln.de



Follow us on **Twitter** at **@IS3_UniCologne** to stay up to date with recent publication and presentations of our group