Graph Theory & Network Analysis

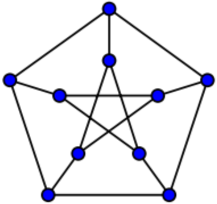
CIS 600, Spring 2018

January 30, 2018

Graphs - What is a graph?

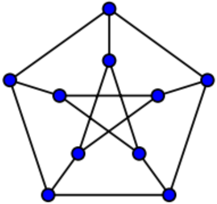
Graphs - What is a graph?

- This is not a graph!



Graphs - What is a graph?

- This is not a graph!



- It is a visual depiction of a graph.

Graphs, defined

- An undirected *graph* $G = (V, E)$ is an *ordered pair*, where V is a set of *vertices* or *nodes* and E is a set of *edges* between nodes.

Graphs, defined

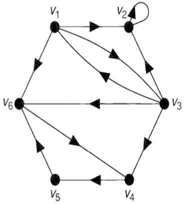
- An undirected *graph* $G = (V, E)$ is an *ordered pair*, where V is a set of *vertices* or *nodes* and E is a set of *edges* between nodes.
- The edges in E (also $E(G)$) are *defined to be* unordered pairs of vertices in V (also $V(G)$).

Graphs, defined

- An undirected *graph* $G = (V, E)$ is an *ordered pair*, where V is a set of *vertices* or *nodes* and E is a set of *edges* between nodes.
- The edges in E (also $E(G)$) are *defined to be* unordered pairs of vertices in V (also $V(G)$).
- A *directed graph* is defined similarly, except that its edges are defined to be *ordered pairs* of vertices in V.

Graphs, defined

- An undirected *graph* $G = (V, E)$ is an *ordered pair*, where V is a set of *vertices* or *nodes* and E is a set of *edges* between nodes.
- The edges in E (also $E(G)$) are *defined to be* unordered pairs of vertices in V (also $V(G)$).
- A *directed graph* is defined similarly, except that its edges are defined to be *ordered pairs* of vertices in V.
- Example:



(from Graph Theory, by Wilson)

Networks

- A network is much more complicated than a graph.

Networks

- A network is much more complicated than a graph.
- Could refer to a group of actual human beings, together with their relationships and activities.

Networks

- A network is much more complicated than a graph.
- Could refer to a group of actual human beings, together with their relationships and activities.
- Or a system of computers exchanging data over the Internet.

Networks

- A network is much more complicated than a graph.
- Could refer to a group of actual human beings, together with their relationships and activities.
- Or a system of computers exchanging data over the Internet.
- The Internet Itself!

Networks

- A network is much more complicated than a graph.
- Could refer to a group of actual human beings, together with their relationships and activities.
- Or a system of computers exchanging data over the Internet.
- The Internet Itself!
- Social networks such as Facebook and Twitter.

Networks - What is interesting?

- Identify groups or cliques within a network

Networks - What is interesting?

- Identify groups or cliques within a network
- Identify important/influential individuals in a network

Networks - What is interesting?

- Identify groups or cliques within a network
- Identify important/influential individuals in a network
- Measure the *strength* of connections in a network

Networks - What is interesting?

- Identify groups or cliques within a network
- Identify important/influential individuals in a network
- Measure the *strength* of connections in a network
- Describe global properties of a network, e.g. "Small World" phenomenon

Networks - What is interesting?

- Identify groups or cliques within a network
- Identify important/influential individuals in a network
- Measure the *strength* of connections in a network
- Describe global properties of a network, e.g. "Small World" phenomenon
- Identify anomalies or security threats in a network, e.g. bots and sock puppets

Networks - What is interesting?

- Identify groups or cliques within a network
- Identify important/influential individuals in a network
- Measure the *strength* of connections in a network
- Describe global properties of a network, e.g. "Small World" phenomenon
- Identify anomalies or security threats in a network, e.g. bots and sock puppets
- Cost & efficiency of operating a network, e.g. traffic, distribution

Networks - What is interesting?

- Identify groups or cliques within a network
- Identify important/influential individuals in a network
- Measure the *strength* of connections in a network
- Describe global properties of a network, e.g. "Small World" phenomenon
- Identify anomalies or security threats in a network, e.g. bots and sock puppets
- Cost & efficiency of operating a network, e.g. traffic, distribution
- Network analysis is a huge field in its own right.

Assignment: Graphs & Network Analysis

Assignment: read Chapter 2 of *Networks, Crowds & Markets*, and then implement some network analysis in python.
See Blackboard for details.

Graph Theory Terminology

- An *edge* $\{v, w\}$ is said to *join* nodes v, w. We can denote this edge vw.

Graph Theory Terminology

- An *edge* $\{v, w\}$ is said to *join* nodes v, w. We can denote this edge vw.
- The *union* of disjoint graphs G_1, G_2 is defined to have nodes and edges

$$V(G_1) \cup V(G_2), \text{ and } E(G_1) \cup E(G_2)$$

Graph Theory Terminology

- An *edge* $\{v, w\}$ is said to *join* nodes v, w. We can denote this edge vw.
- The *union* of disjoint graphs G_1, G_2 is defined to have nodes and edges

$$V(G_1) \cup V(G_2), \text{ and } E(G_1) \cup E(G_2)$$

- A graph is *connected* if it cannot be realized as the union of two graphs.

Graph Theory Terminology

- An *edge* $\{v, w\}$ is said to *join* nodes v, w. We can denote this edge vw.
- The *union* of disjoint graphs G_1, G_2 is defined to have nodes and edges

$$V(G_1) \cup V(G_2), \text{ and } E(G_1) \cup E(G_2)$$

- A graph is *connected* if it cannot be realized as the union of two graphs.
- The graph G_1 is a *subgraph* of the graph G_2 if

$$V(G_1) \subseteq V(G_2) \text{ and } E(G_1) \subseteq E(G_2)$$

Graph Theory Terminology

- An *edge* $\{v, w\}$ is said to *join* nodes v, w. We can denote this edge vw.
- The *union* of disjoint graphs G_1, G_2 is defined to have nodes and edges

$$V(G_1) \cup V(G_2), \text{ and } E(G_1) \cup E(G_2)$$

- A graph is *connected* if it cannot be realized as the union of two graphs.
- The graph G_1 is a *subgraph* of the graph G_2 if

$$V(G_1) \subseteq V(G_2) \text{ and } E(G_1) \subseteq E(G_2)$$

- Nodes v, w are *adjacent* in G if $vw \in E(G)$.

Graph Theory Terminology

- An *edge* $\{v, w\}$ is said to *join* nodes v, w. We can denote this edge vw.
- The *union* of disjoint graphs G_1, G_2 is defined to have nodes and edges

$$V(G_1) \cup V(G_2), \text{ and } E(G_1) \cup E(G_2)$$

- A graph is *connected* if it cannot be realized as the union of two graphs.
- The graph G_1 is a *subgraph* of the graph G_2 if

$$V(G_1) \subseteq V(G_2) \text{ and } E(G_1) \subseteq E(G_2)$$

- Nodes v, w are *adjacent* in G if $vw \in E(G)$.
- Nodes v, w are *incident* with the edge $vw \in E(G)$.

Graph Theory Terminology

- An *edge* $\{v, w\}$ is said to *join* nodes v, w. We can denote this edge vw.
- The *union* of disjoint graphs G_1, G_2 is defined to have nodes and edges

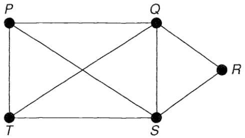
$$V(G_1) \cup V(G_2), \text{ and } E(G_1) \cup E(G_2)$$

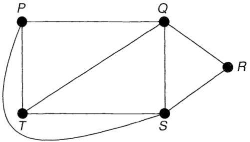
- A graph is *connected* if it cannot be realized as the union of two graphs.
- The graph G_1 is a *subgraph* of the graph G_2 if

$$V(G_1) \subseteq V(G_2) \text{ and } E(G_1) \subseteq E(G_2)$$

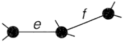
- Nodes v, w are *adjacent* in G if $vw \in E(G)$.
- Nodes v, w are *incident* with the edge $vw \in E(G)$.
- The *degree* $deg(v)$ is the number of edges incident with v in G.

Examples

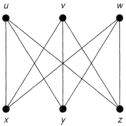


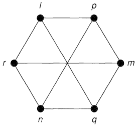


adjacent vertices

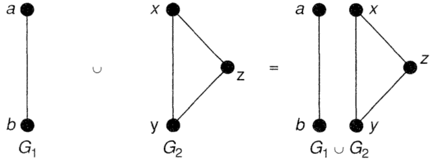


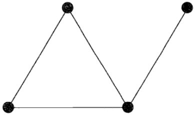
adjacent edges

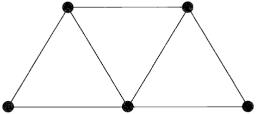




Examples







Graph Theory Terminology

- A node v is *isolated* if $deg(v) = 0$.

Graph Theory Terminology

- A node v is *isolated* if $deg(v) = 0$.
- A node v is an *end-node* of G if $deg(v) = 1$.

Graph Theory Terminology

- A node v is *isolated* if $deg(v) = 0$.
- A node v is an *end-node* of G if $deg(v) = 1$.
- The *Handshaking Lemma*:

$$\sum_{v \in V(G)} deg(v) = 2 \cdot |E(G)|$$

Graph Theory Terminology

- A node v is *isolated* if $deg(v) = 0$.
- A node v is an *end-node* of G if $deg(v) = 1$.
- The *Handshaking Lemma*:

$$\sum_{v \in V(G)} deg(v) = 2 \cdot |E(G)|$$

- Why?

Graph Theory Terminology

- The *adjacency matrix* A of a graph G is the square matrix with entries

$$A_{ij} = |\{x \in E(G) \mid \text{ x joins } v_i \text{ and } v_j\}|$$

Graph Theory Terminology

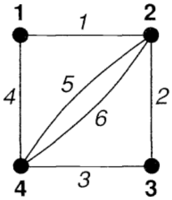
- The *adjacency matrix* A of a graph G is the square matrix with entries

$$A_{ij} = |\{x \in E(G) \mid \text{ x joins } v_i \text{ and } v_j\}|$$

- The *incidence matrix* M of a graph G is the $|V(G)| \times |E(G)|$ matrix such that

$$M_{ij} = \begin{cases} 1 & \text{if node } i \text{ is incident with edge } j \\ 0 & \text{otherwise} \end{cases}$$

Examples



$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{pmatrix} \qquad \mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Graph Theory Terminology

- A *walk* in G is a finite sequence of edges of the form $v_0 v_1, v_1 v_2, \ldots, v_m v_{m-1}$.

Graph Theory Terminology

- A *walk* in G is a finite sequence of edges of the form $v_0v_1, v_1v_2, \ldots, v_mv_{m-1}$.
- The node v_0 is called its *initial* vertex, and v_m its *final*.

Graph Theory Terminology

- A *walk* in G is a finite sequence of edges of the form $v_0v_1, v_1v_2, \ldots, v_mv_{m-1}$.
- The node v_0 is called its *initial* vertex, and v_m its *final*.
- Such a walk is called a *walk from* v_0 *to* v_m.

Graph Theory Terminology

- A *walk* in G is a finite sequence of edges of the form $v_0 v_1, v_1 v_2, \ldots, v_m v_{m-1}$.
- The node v_0 is called its *initial* vertex, and v_m its *final*.
- Such a walk is called a *walk from v_0 to v_m*.
- The number of edges in a walk is called its *length*.

Graph Theory Terminology

- A *walk* in G is a finite sequence of edges of the form $v_0v_1, v_1v_2, \ldots, v_mv_{m-1}$.
- The node v_0 is called its *initial* vertex, and v_m its *final*.
- Such a walk is called a *walk from v_0 to v_m*.
- The number of edges in a walk is called its *length*.
- A *trail* is a walk with distinct edges.

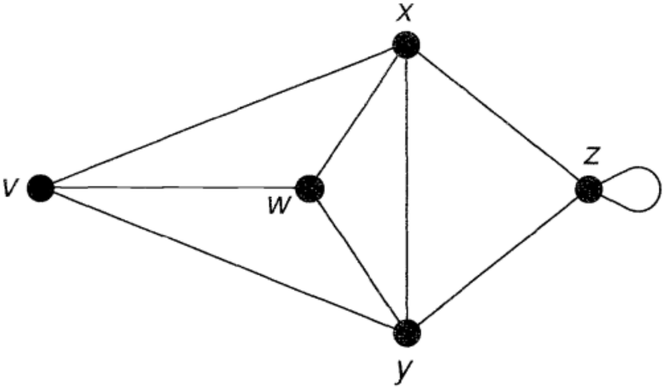
Graph Theory Terminology

- A *walk* in G is a finite sequence of edges of the form $v_0v_1, v_1v_2, \ldots, v_mv_{m-1}$.
- The node v_0 is called its *initial* vertex, and v_m its *final*.
- Such a walk is called a *walk from* v_0 *to* v_m.
- The number of edges in a walk is called its *length*.
- A *trail* is a walk with distinct edges.
- A *path* is a trail with distinct nodes, **with the possible exception of the initial and final**.

Graph Theory Terminology

- A *walk* in G is a finite sequence of edges of the form $v_0 v_1, v_1 v_2, \ldots, v_m v_{m-1}$.
- The node v_0 is called its *initial* vertex, and v_m its *final*.
- Such a walk is called a *walk from* v_0 *to* v_m.
- The number of edges in a walk is called its *length*.
- A *trail* is a walk with distinct edges.
- A *path* is a trail with distinct nodes, **with the possible exception of the initial and final**.
- A *cycle* is a path having $v_0 = v_m$.

Examples



Graph Theory Terminology

- A *disconnecting set* in G is a subset of $E(G)$ whose removal leaves G disconnected.

Graph Theory Terminology

- A *disconnecting set* in G is a subset of $E(G)$ whose removal leaves G disconnected.
- A *cutset* is a minimal disconnecting set.

Graph Theory Terminology

- A *disconnecting set* in G is a subset of $E(G)$ whose removal leaves G disconnected.
- A *cutset* is a minimal disconnecting set.
- The *edge connectivity* $\lambda(G)$ is the size of a smallest cutset.

Graph Theory Terminology

- A *disconnecting set* in G is a subset of $E(G)$ whose removal leaves G disconnected.
- A *cutset* is a minimal disconnecting set.
- The *edge connectivity* $\lambda(G)$ is the size of a smallest cutset.
- A *separating set* in a connected graph is a subset of $V(G)$ whose removal disconnects G.

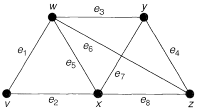
Graph Theory Terminology

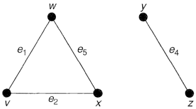
- A *disconnecting set* in G is a subset of $E(G)$ whose removal leaves G disconnected.
- A *cutset* is a minimal disconnecting set.
- The *edge connectivity* $\lambda(G)$ is the size of a smallest cutset.
- A *separating set* in a connected graph is a subset of $V(G)$ whose removal disconnects G.
- The *node connectivity* $\kappa(G)$ of a connected graph G that is not the complete graph is the size of a smallest separating set in G.

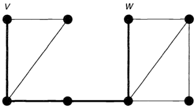
Graph Theory Terminology

- A *disconnecting set* in G is a subset of $E(G)$ whose removal leaves G disconnected.
- A *cutset* is a minimal disconnecting set.
- The *edge connectivity* $\lambda(G)$ is the size of a smallest cutset.
- A *separating set* in a connected graph is a subset of $V(G)$ whose removal disconnects G.
- The *node connectivity* $\kappa(G)$ of a connected graph G that is not the complete graph is the size of a smallest separating set in G.
- G is called *k-connected* or *k-edge connected* if $\lambda(G) \geq k$ or $\kappa(G) \geq k$, respectively.

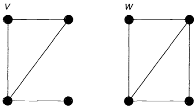
Examples







connected



disconnected

Graph Theory Terminology

- An *edge weighting* for a graph G is a function

$$\omega : E(G) \rightarrow \mathbb{R}^{+}$$

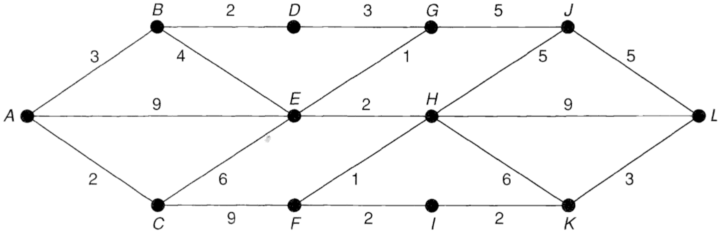
Graph Theory Terminology

- An *edge weighting* for a graph G is a function

$$\omega : E(G) \to \mathbb{R}^+$$

- The pair (G, ω) is then called a *weighted graph*.

Examples



Graph Theory Terminology

- A *forest* is a graph with no cycles.

Graph Theory Terminology

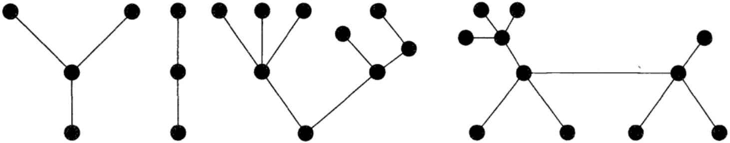
- A *forest* is a graph with no cycles.
- A *tree* is a connected forest.

Graph Theory Terminology

- A *forest* is a graph with no cycles.
- A *tree* is a connected forest.
- Many equivalent definitions, e.g.

$$|E(G)| = |V(G)| - 1$$

Examples



Graph Theory Problems

- Shortest path - Dijsktra's Algorithm

Graph Theory Problems

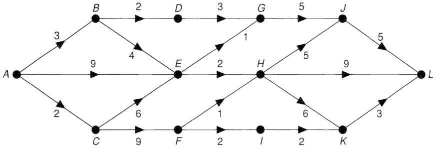
- Shortest path - Dijsktra's Algorithm
- Chinese postman - “route inspection problem”, Kwan Mei-Ko

Graph Theory Problems

- Shortest path - Dijsktra's Algorithm
- Chinese postman - "route inspection problem", Kwan Mei-Ko
- Traveling Salesman Problem - visit all nodes (optimally). NP-Hard, whereas two above are polynomial-time.

Graph Theory Problems

- Shortest path - Dijsktra's Algorithm
- Chinese postman - "route inspection problem", Kwan Mei-Ko
- Traveling Salesman Problem - visit all nodes (optimally). NP-Hard, whereas two above are polynomial-time.
- Max-Flow, Min-Cut - maximize 'flow' between nodes in a directed network.



Centrality - Degree

- The node v_i has *degree centrality* $C_D(v_i)$ defined by

$$C_D(v_i) = \sum_j A_{ij}$$

(where A is the adjacency matrix).

Centrality - Degree

- The node v_i has *degree centrality* $C_D(v_i)$ defined by

$$C_D(v_i) = \sum_j A_{ij}$$

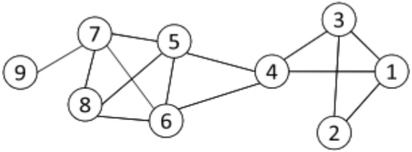
(where A is the adjacency matrix).

- The *normalized degree centrality* is defined by

$$C'_D(v_i) = \frac{1}{n-1} C_D(v_i)$$

(where n is the number of nodes).

Examples



Node	1	2	3	4	5	6	7	8	9
1	-	1	1	1	0	0	0	0	0
2	1	-	1	0	0	0	0	0	0
3	1	1	-	1	0	0	0	0	0
4	1	0	1	-	1	1	0	0	0
5	0	0	0	1	-	1	1	1	0
6	0	0	0	1	1	-	1	1	0
7	0	0	0	0	1	1	-	1	1
8	0	0	0	0	1	1	1	-	0
9	0	0	0	0	0	0	1	0	-

Node	Degree Centrality	Node	Degree Centrality
1	0.375	6	**0.5**
2	0.25	7	**0.5**
3	0.375	8	0.375
4	**0.5**	9	0.125
5	**0.5**		

Centrality - Closeness

- A nodes *closeness centrality* is a measure of how close it is, along paths, to other nodes.

Centrality - Closeness

- A nodes *closeness centrality* is a measure of how close it is, along paths, to other nodes.
- The distance $g(v_i, v_j)$ between v_i, v_j is the length of a shortest path from v_i to v_j.

Centrality - Closeness

- A nodes *closeness centrality* is a measure of how close it is, along paths, to other nodes.
- The distance $g(v_i, v_j)$ between v_i, v_j is the length of a shortest path from v_i to v_j.
- Closeness centrality $C_C(v_i)$ is defined by

$$C_C(v_i) = \frac{n-1}{\sum_{j \neq i} g(v_i, v_j)}$$

Examples

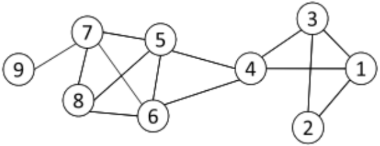


Table 2.1: Pairwise geodesic distance

Node	1	2	3	4	5	6	7	8	9
1	0	1	1	1	2	2	3	3	4
2	1	0	1	2	3	3	4	4	5
3	1	1	0	1	2	2	3	3	4
4	1	2	1	0	1	1	2	2	3
5	2	3	2	1	0	1	1	1	2
6	2	3	2	1	1	0	1	1	2
7	3	4	3	2	1	1	0	1	1
8	3	4	3	2	1	1	1	0	2
9	4	5	4	3	2	2	1	2	0

$$C_C(3) = \frac{9-1}{1+1+1+2+2+3+3+4} = 8/17 = 0.47,$$

$$C_C(4) = \frac{9-1}{1+2+1+1+1+2+2+3} = 8/13 = 0.62.$$

Node 4 is more central than node 3

Centrality - Betweennes

- For indices i, k, the value σ_{ik} is the number of shortest paths from v_i to v_k.

Centrality - Betweennes

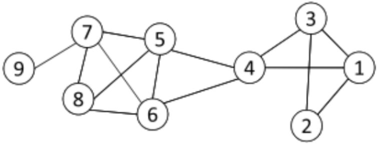
- For indices i, k, the value σ_{ik} is the number of shortest paths from v_i to v_k.
- The value $\sigma_{ik}(v_j)$ is the number of shortest paths from v_i to v_k which pass through v_j.

Centrality - Betweennes

- For indices i, k, the value σ_{ik} is the number of shortest paths from v_i to v_k.
- The value $\sigma_{ik}(v_j)$ is the number of shortest paths from v_i to v_k which pass through v_j.
- The *betweenness centrality* $C_B(v_j)$ of v_j is then

$$C_B(v_j) = \sum_{i,k \neq j, i<k} \frac{\sigma_{ik}(v_j)}{\sigma_{ik}}$$

Examples



$$C_B(4) = 15$$

Table 2.2: $\sigma_{st}(4)/\sigma_{st}$

	$s=1$	$s=2$	$s=3$
$t=5$	1/1	2/2	1/1
$t=6$	1/1	2/2	1/1
$t=7$	2/2	4/4	2/2
$t=8$	2/2	4/4	2/2
$t=9$	2/2	4/4	2/2

What's the betweenness centrality for node 5?

Centrality - Eigenvector

- The *eigenvector centrality* $C_E(v_i)$ of node v_i is a recursive concept.

- The *eigenvector centrality* $C_E(v_i)$ of node v_i is a recursive concept.
- Supposing that we have the values, they should satisfy

$$C_E(v_i) \propto \sum_j A_{ij} C_E(v_j)$$

- The *eigenvector centrality* $C_E(v_i)$ of node v_i is a recursive concept.
- Supposing that we have the values, they should satisfy

$$C_E(v_i) \propto \sum_j A_{ij} C_E(v_j)$$

- This is the eigenvalue problem

$$AC_E = \lambda C_E$$

Centrality - Eigenvector

- The *eigenvector centrality* $C_E(v_i)$ of node v_i is a recursive concept.
- Supposing that we have the values, they should satisfy

$$C_E(v_i) \propto \sum_j A_{ij} C_E(v_j)$$

- This is the eigenvalue problem

$$AC_E = \lambda C_E$$

- Can be solved by the *power method*

Centrality - Eigenvector

- The *eigenvector centrality* $C_E(v_i)$ of node v_i is a recursive concept.
- Supposing that we have the values, they should satisfy

$$C_E(v_i) \propto \sum_j A_{ij} C_E(v_j)$$

- This is the eigenvalue problem

$$AC_E = \lambda C_E$$

- Can be solved by the *power method*
- By iteration, naively, without linear algebra

Centrality - Eigenvector

- The *eigenvector centrality* $C_E(v_i)$ of node v_i is a recursive concept.
- Supposing that we have the values, they should satisfy

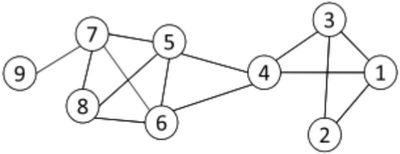
$$C_E(v_i) \propto \sum_j A_{ij} C_E(v_j)$$

- This is the eigenvalue problem

$$AC_E = \lambda C_E$$

- Can be solved by the *power method*
- By iteration, naively, without linear algebra
- Let's use `numpy`.

Examples



Node	1	2	3	4	5	6	7	8	9
1	-	1	1	1	0	0	0	0	0
2	1	-	1	0	0	0	0	0	0
3	1	1	-	1	0	0	0	0	0
4	1	0	1	-	1	1	0	0	0
5	0	0	0	1	-	1	1	1	0
6	0	0	0	1	1	-	1	1	0
7	0	0	0	0	1	1	-	1	1
8	0	0	0	0	1	1	1	-	0
9	0	0	0	0	0	0	1	0	-

Node	Eigenvector Centrality	Node	Eigenvector Centrality
1	0.195751798216	6	**0.468084576647**
2	0.111686197298	7	0.409977787365
3	0.195751798216	8	0.384018975728
4	0.37874977785	9	0.116955395176
5	**0.468084576647**		