INFO 1368
Introduction to Network Science

Filippo Menczer

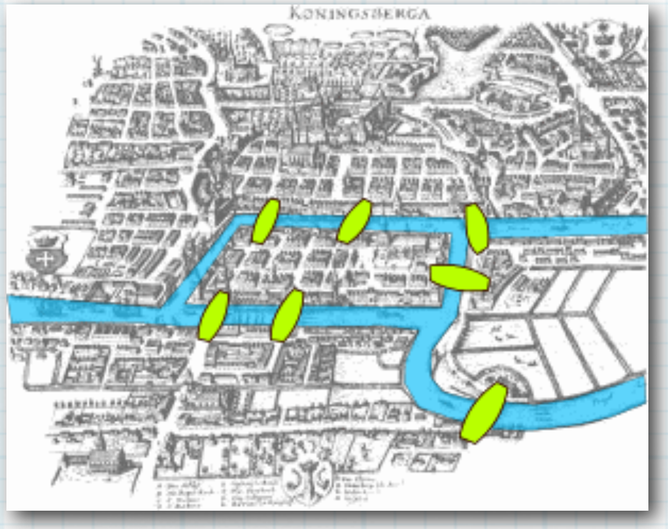
CNetS Center for Complex Networks and Systems Research

INDIANA UNIVERSITY BLOOMINGTON

SCHOOL OF Informatics and Computing

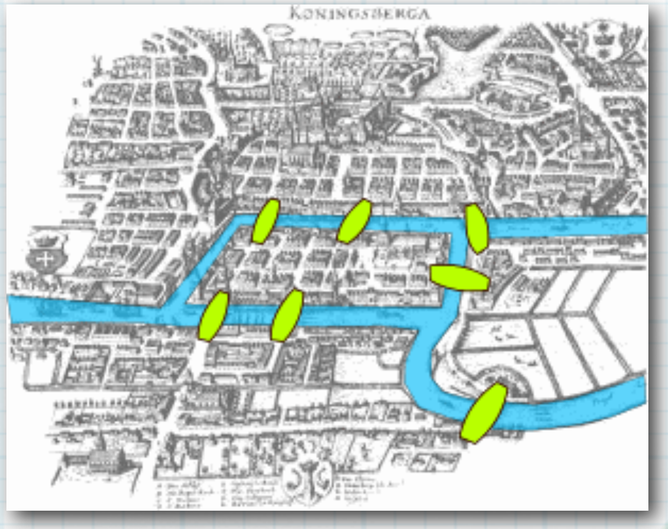
- Attendance
- Schedule update
- Review

Euler circa 1736: Koningsberg bridges

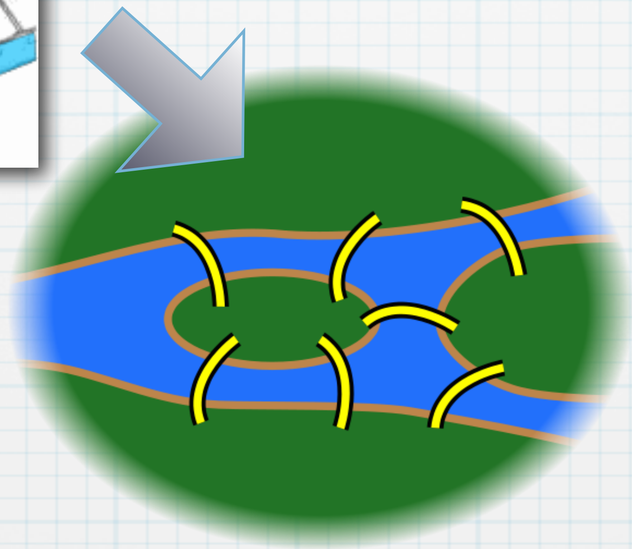


Can you cross all 7 bridges just once each?

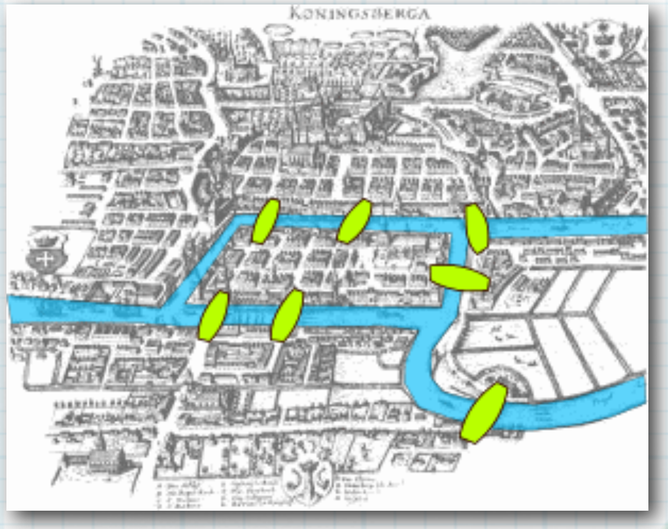
Euler circa 1736: Koningsberg bridges



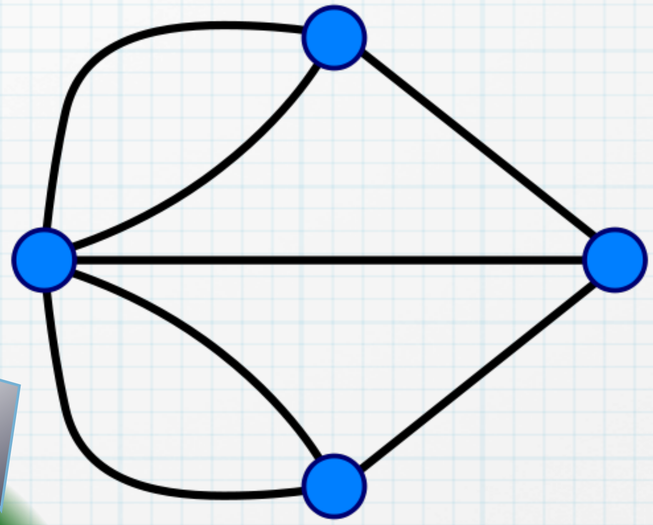
Can you cross all 7 bridges just once each?

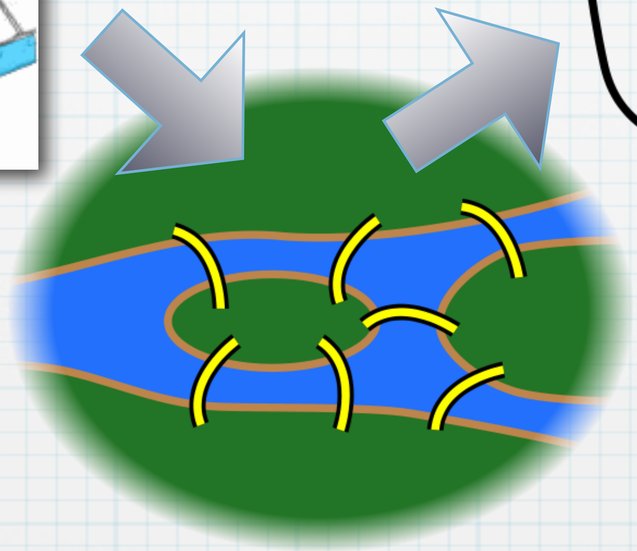


Euler circa 1736: Koningsberg bridges

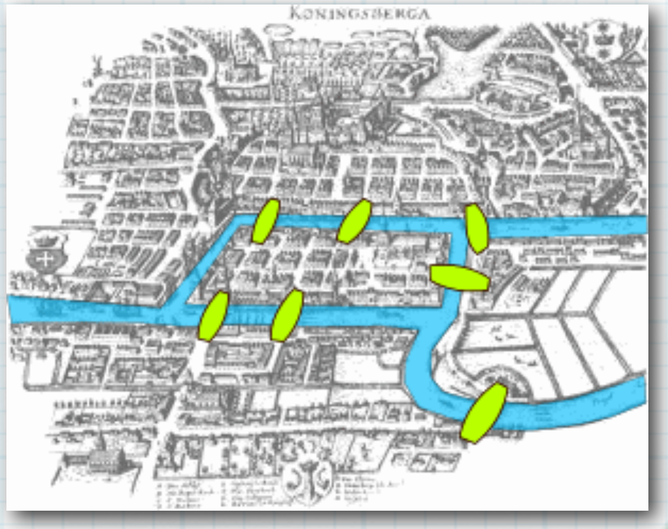


Can you cross all 7 bridges just once each?

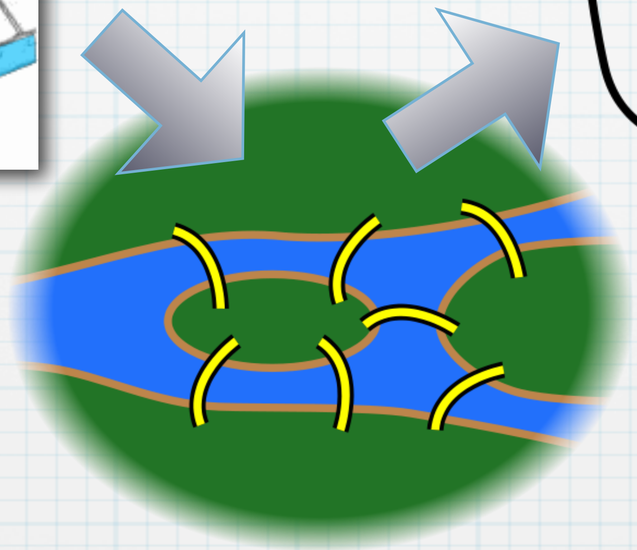


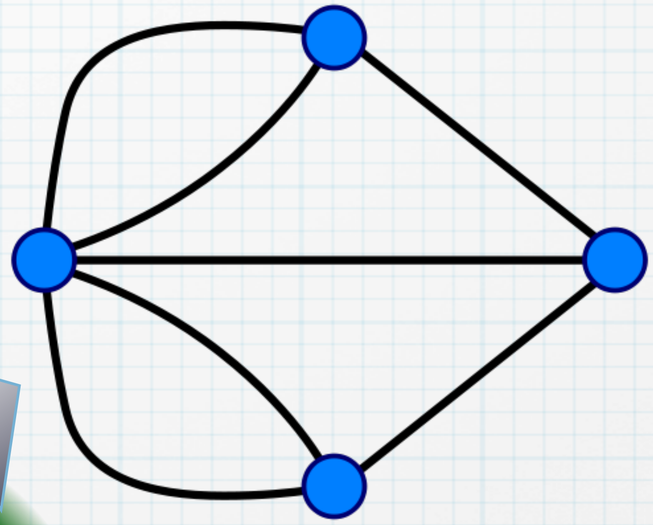


Euler circa 1736: Koningsberg bridges



Can you cross all 7 bridges just once each?





Answer: no. At most two nodes (start, end) may have odd degree

What's new?

* 1930-40's Moreno (sociogram, sociometry) – psychology and social sciences
* 1960-70's Erdos (random graph theory) – math
* 1970-80's Granovetter (weak ties) – economics and sociology
* 1980-90's Waxman (topology generator) – computer science
* 2000's Non-equilibrium (statistical mechanics) – physics

The age of networks

- Data gathering and integration
- Biology
- Internet, WWW, P2P, ad-hoc.....
- Social sciences
- Social media
- Human mobility/activity

New challenges & new understanding

- Data size shifts (10^2 -> 10^8 elements) **(complexity)**
- Different domains (biology, info-structures, infrastructures, social, scientometrics) **(universality)**
- Large scale longitudinal studies (time series) **(dynamical modeling)**

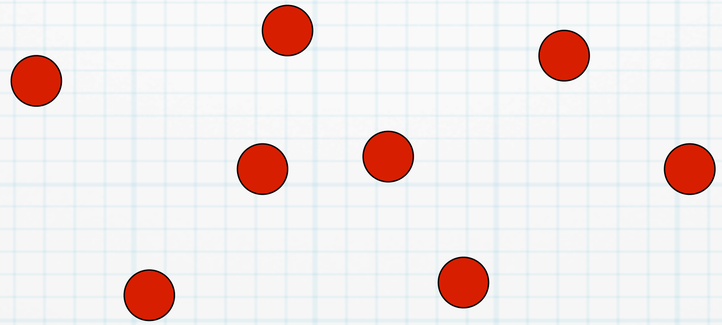
The plan

* Last week we saw a bunch of interesting networks arising in many different domains
* Now let us learn the language of networks
  * The components: nodes, links
  * Types of networks and representations
  * Features of nodes and edges

We want to be able to talk about...

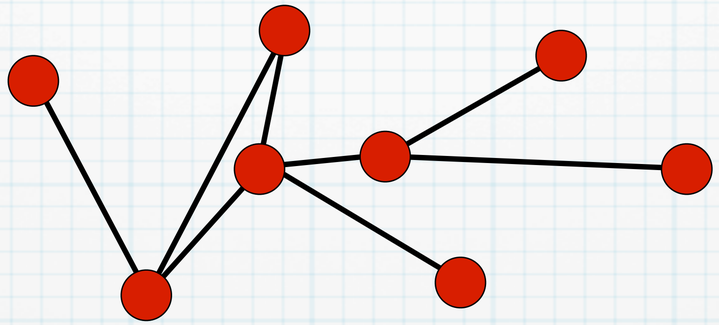
* properties to characterize structure & behavior of networks
* roles of networks in affecting processes occurring on network structures

Basic components



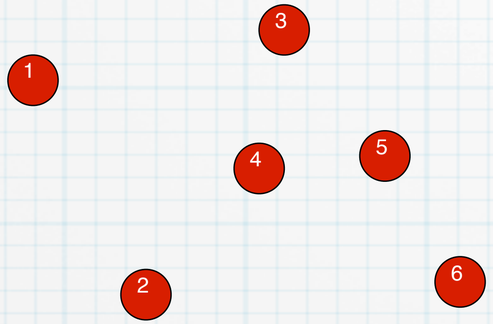
- **elements:** nodes, vertices, …

Basic components



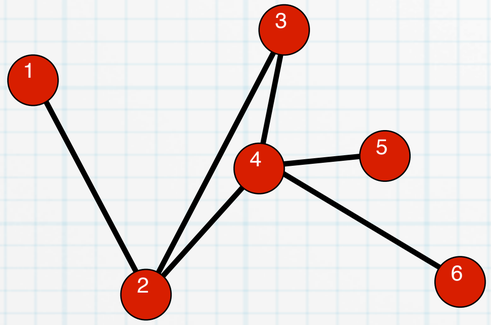
- **elements**: nodes, vertices, …
- **interactions**: links, edges, neighbors…

Python and NetworkX



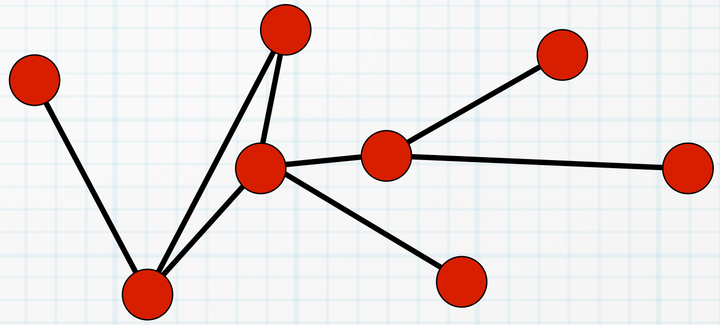

```
import networkx as nx # always!

G = nx.Graph()
G.add_node(1)
G.add_nodes_from([2,3,…])
…
G.add_edge(1,2)
G.add_edges_from([(2,3),(2,4),…])
…
G.nodes()
G.edges()
G.neighbors(4)

for n in G.nodes:
    print(n, G.neighbors(n))
for u,v in G.edges:
    print(u, v)
```

Python and NetworkX



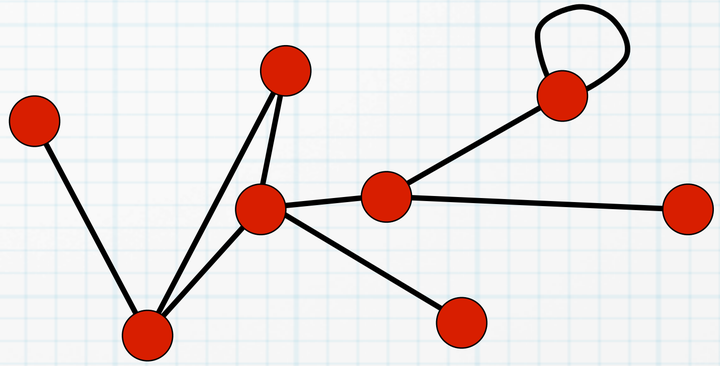

```
import networkx as nx # always!

G = nx.Graph()
G.add_node(1)
G.add_nodes_from([2,3,…])
…
G.add_edge(1,2)
G.add_edges_from([(2,3),(2,4),…])
…
G.nodes()
G.edges()
G.neighbors(4)

for n in G.nodes:
    print(n, G.neighbors(n))
for u,v in G.edges:
    print(u, v)
```

Some definitions



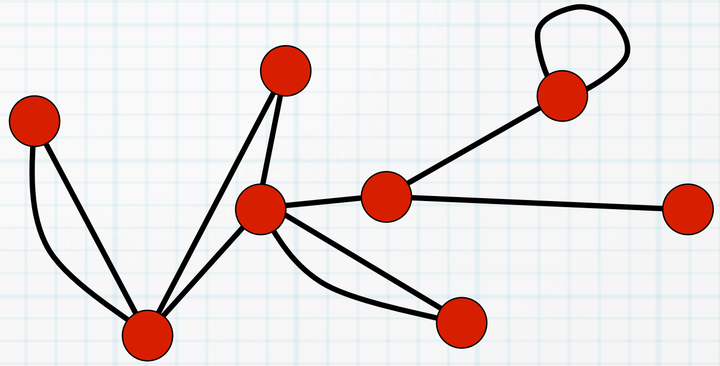
* Vertices + edges
* size: # Vertices
* ~~self loops~~
* ~~multiple edges (multigraphs)~~
* directed networks: directed edges or links

Some definitions



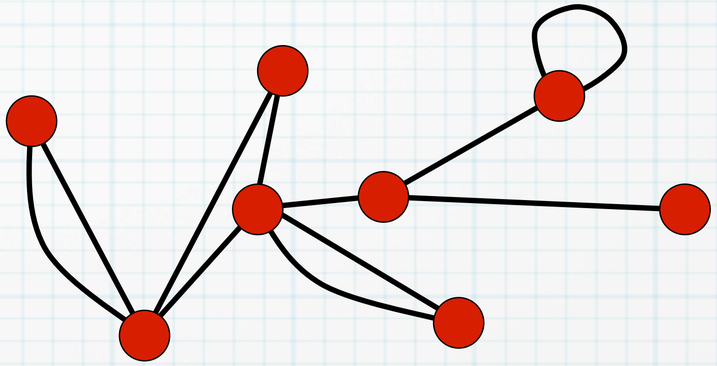
* Vertices + edges
* size: # Vertices
* ~~self loops~~
* ~~multiple edges (multigraphs)~~
* directed networks: directed edges or links

Some definitions

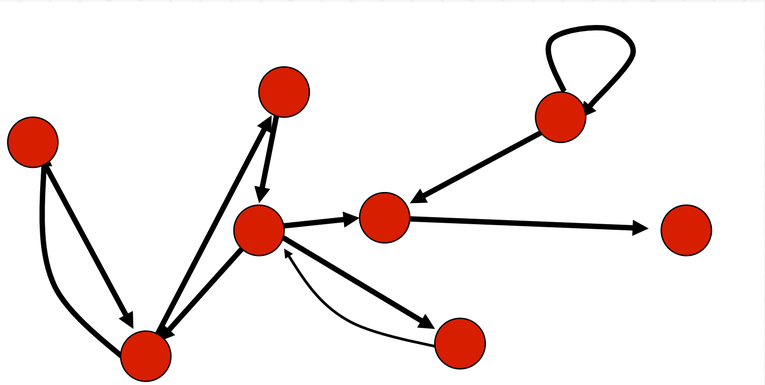


* Vertices + edges
* size: # Vertices
* ~~self loops~~
* ~~multiple edges (multigraphs)~~
* directed networks: directed edges or links

Some definitions

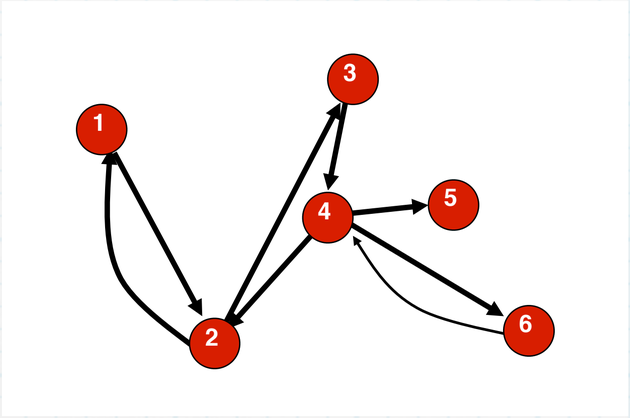


* Vertices + edges
* size: # Vertices
* ~~self loops~~
* ~~multiple edges (multigraphs)~~



* directed networks: directed edges or links

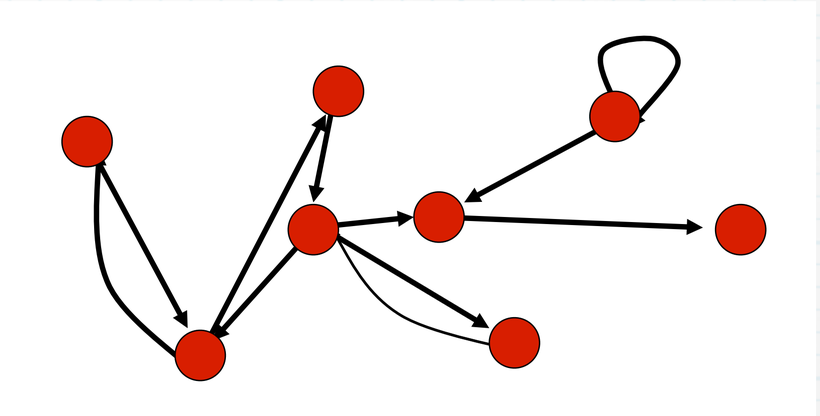
Python and NetworkX



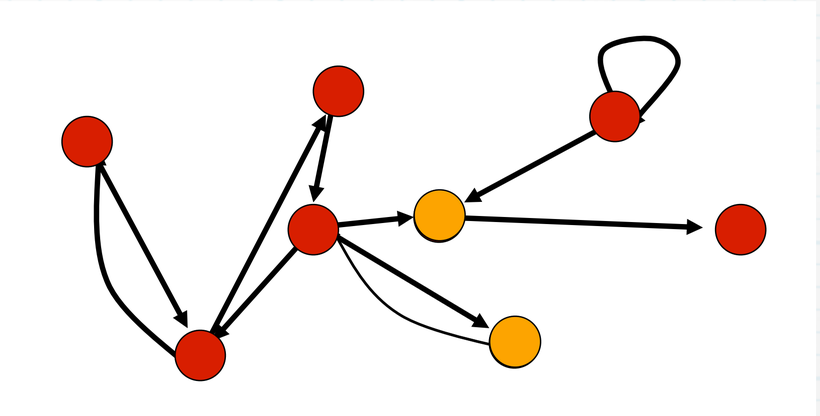

```
import networkx as nx # don't forget!

D = nx.DiGraph()
D.add_edge(1,2)
D.add_edge(2,1)
D.add_edges_from([(2,3),(3,4),…])
…
D.number_of_nodes()
D.number_of_edges()
D.edges()
D.successors(2)
D.predecessors(2)
D.neighbors(2)
```

More definitions

- different types of vertices
- different types of edges
  - weights
- bipartite network:
  - 2 types of nodes
  - links between different types



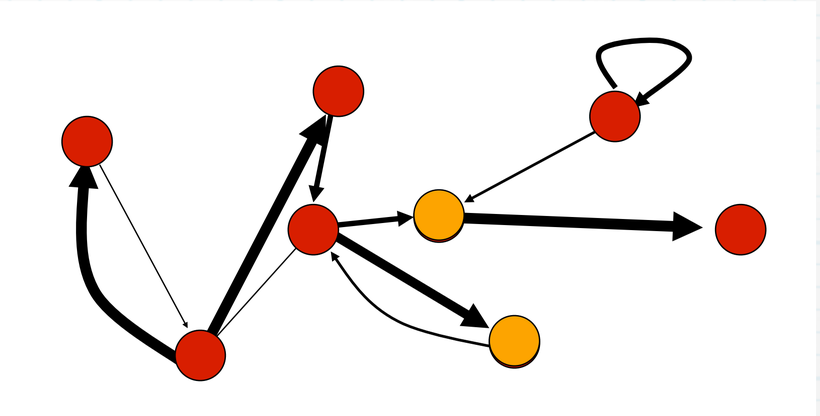
More definitions



* different types of vertices
* different types of edges
  * weights
* bipartite network:
  * 2 types of nodes
  * links between different types

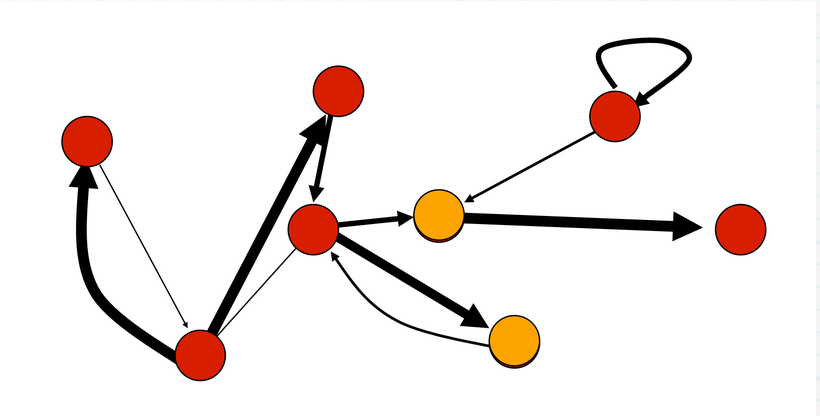
More definitions

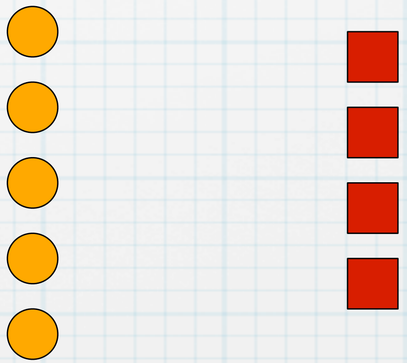
- different types of vertices
- different types of edges
  - weights
- bipartite network:
  - 2 types of nodes
  - links between different types



More definitions

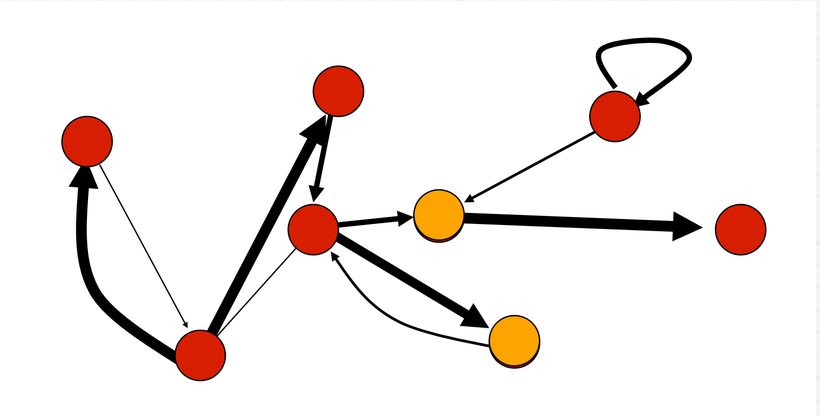
- different types of vertices
- different types of edges
  - weights
- bipartite network:
  - 2 types of nodes
  - links between different types

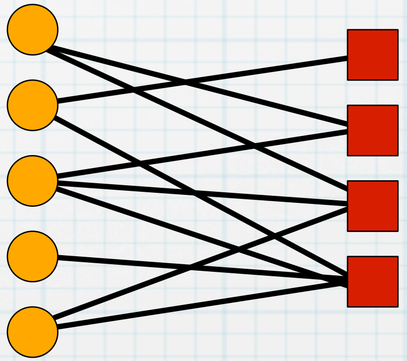




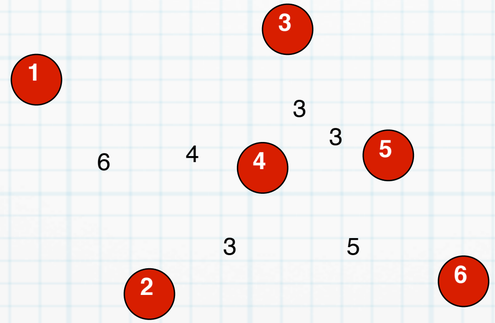
More definitions

* different types of vertices
* different types of edges
  * weights
* bipartite network:
  * 2 types of nodes
  * links between different types



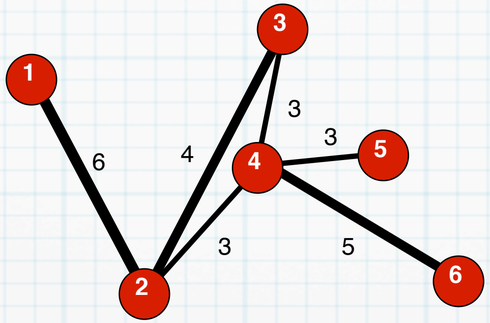


Python and NetworkX




```
W = nx.Graph()
W.add_edge(1,2,weight=6)
…
W.add_weighted_edges_from([(4,5,3),(4,6,5),…])
…
W.edges()
W.edges(data='weight')
for (u,v,d) in W.edges(data='weight'):
    if d>3:
        print('(%d, %d, %d)'%(u,v,d))
B = nx.complete_bipartite_graph(4,5)
```

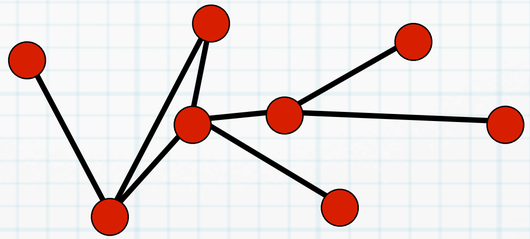
Python and NetworkX




```
W = nx.Graph()
W.add_edge(1,2,weight=6)
…
W.add_weighted_edges_from([(4,5,3),(4,6,5),…])
…
W.edges()
W.edges(data='weight')
for (u,v,d) in W.edges(data='weight'):
    if d>3:
        print('(%d, %d, %d)'%(u,v,d))
B = nx.complete_bipartite_graph(4,5)
```

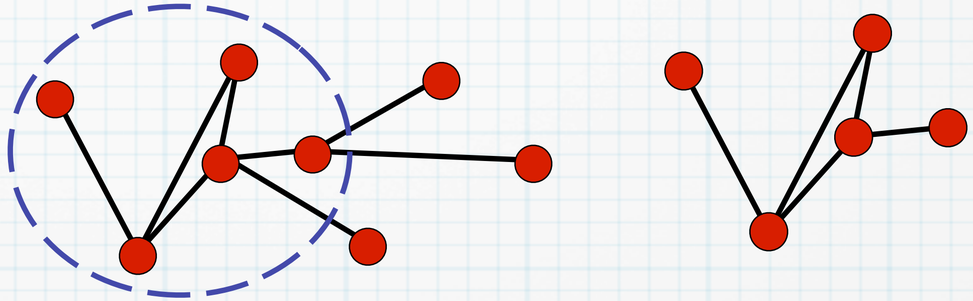
- Attendance
- Schedule update
- Review

More definitions



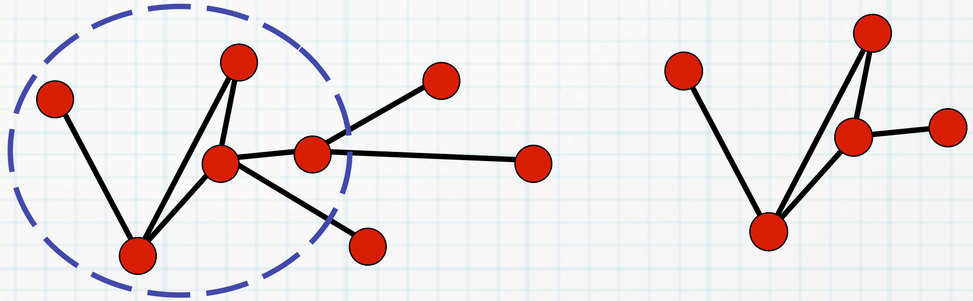
* subgraph
* complete graph (clique)
  * size N → E = N * (N-1) / 2
* tree: connected, no cycles
  * size N → E = N-1

More definitions

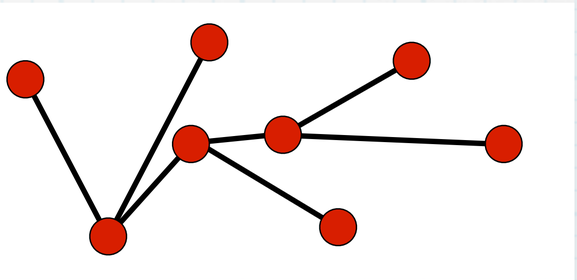


* subgraph
* complete graph (clique)
  * size N → E = N * (N-1) / 2
* tree: connected, no cycles
  * size N → E = N-1

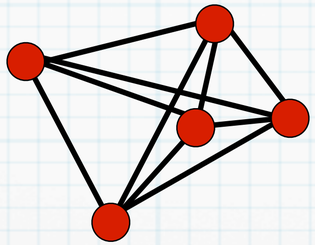
More definitions



* subgraph
* complete graph (clique)
  * size N → E = N * (N-1) / 2
* tree: connected, no cycles
  * size N → E = N-1



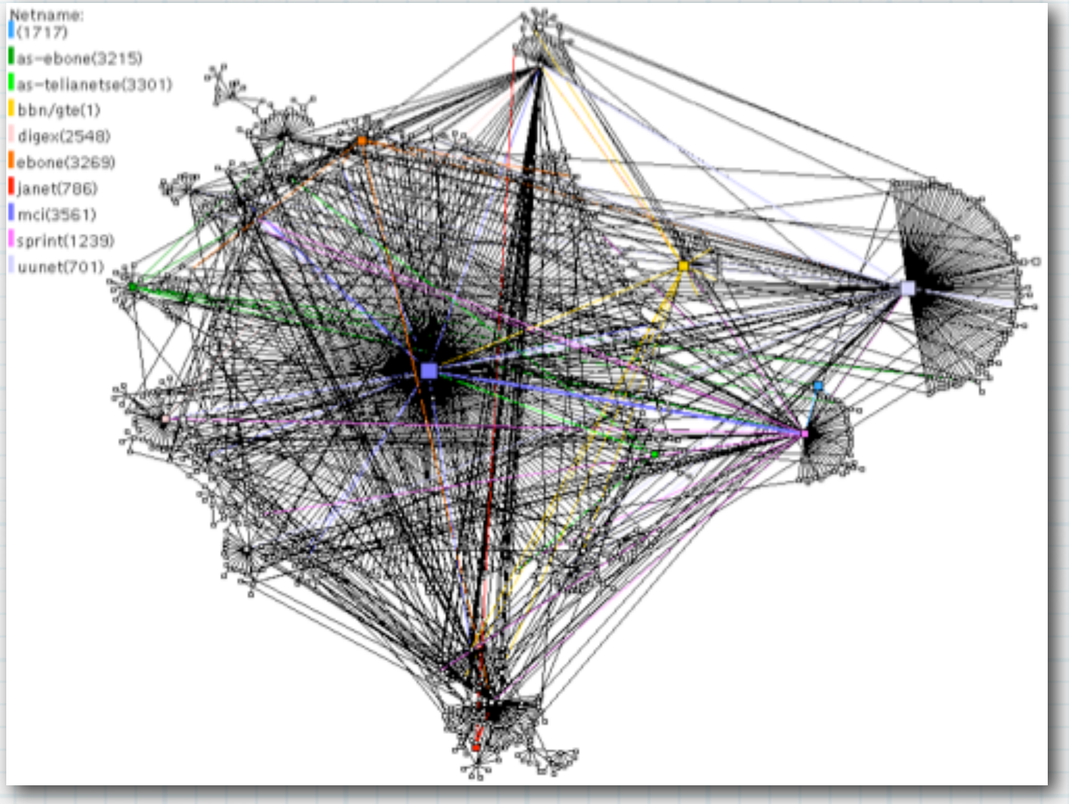
Python and NetworkX



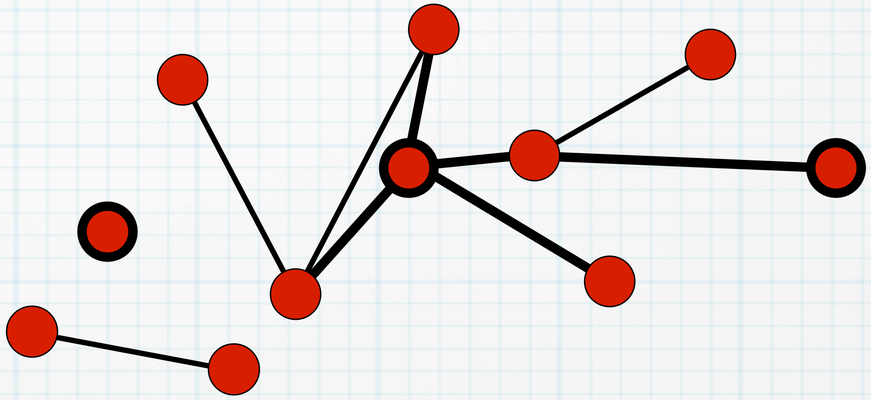
```
K5 = nx.complete_graph(5)
K5.edges()
clique = nx.subgraph(K5, (0,1,2))
nx.is_tree(K5)
c = nx.cycle_graph(4)
nx.is_tree(c)
p = nx.path_graph(5)
nx.is_tree(p)
s = nx.star_graph(6)
nx.is_tree(s)
```

Centrality

Degree as a measure of influence, importance, prestige...



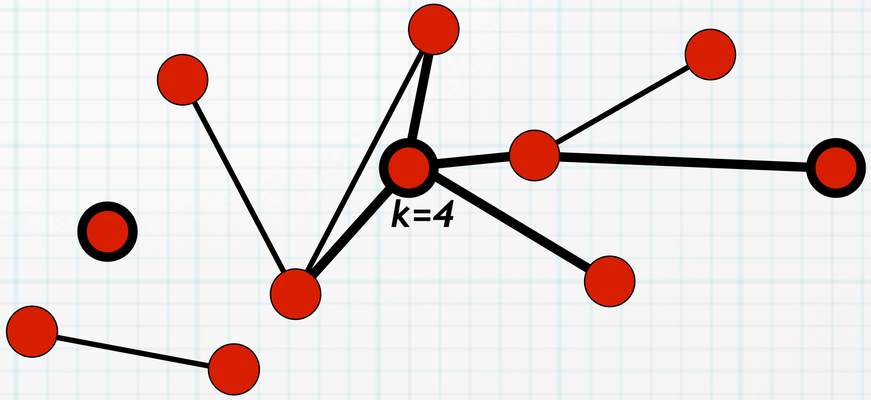
More definitions



* degree k
  * indegree k_{in}
  * outdegree k_{out}

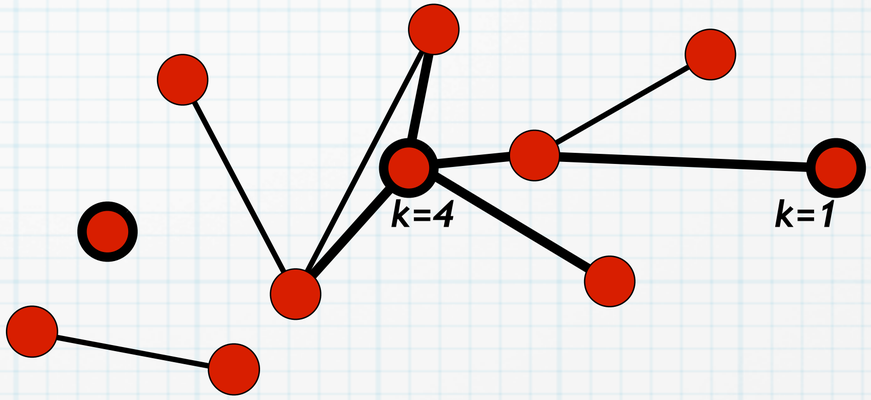
More definitions

* degree k
  * indegree k_{in}
  * outdegree k_{out}



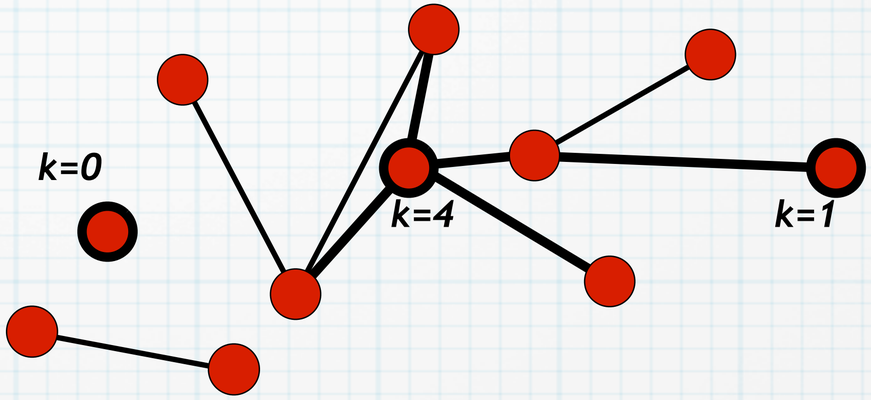
More definitions

* degree k
  * indegree k_{in}
  * outdegree k_{out}



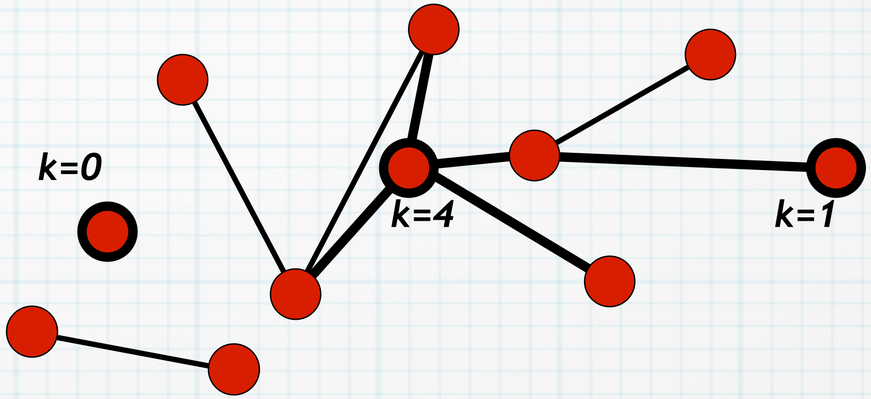
More definitions

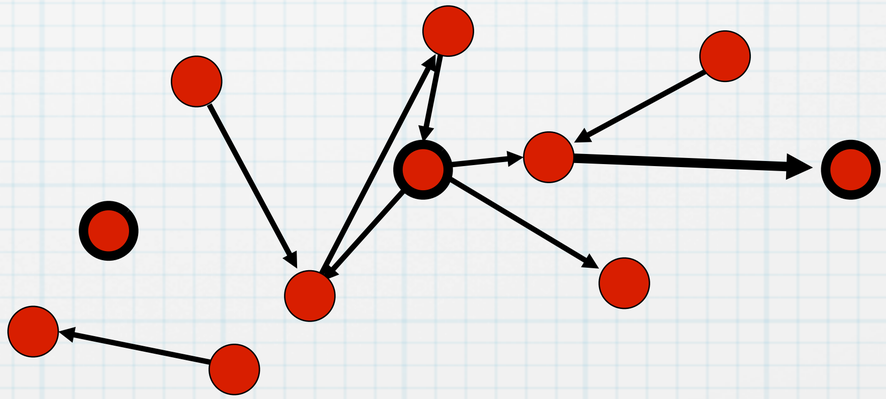
* degree k
  * indegree k_{in}
  * outdegree k_{out}



More definitions

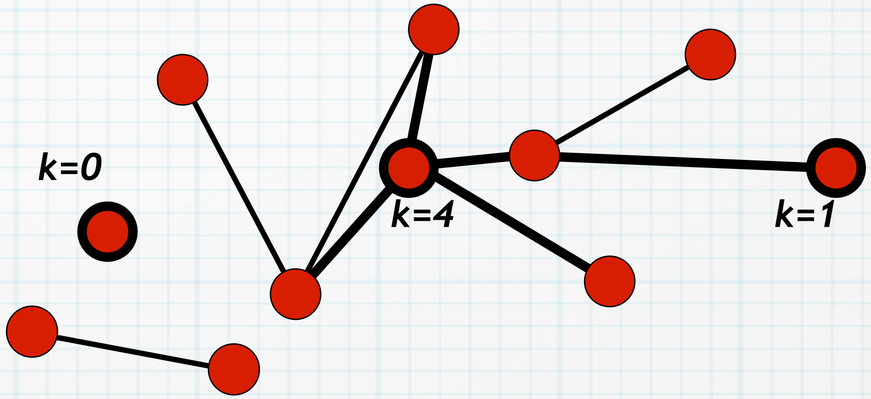
* degree k
  * indegree k_{in}
  * outdegree k_{out}

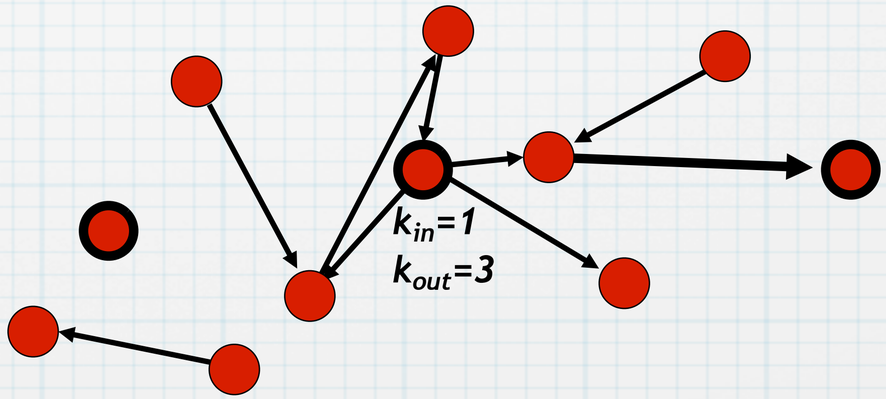




More definitions

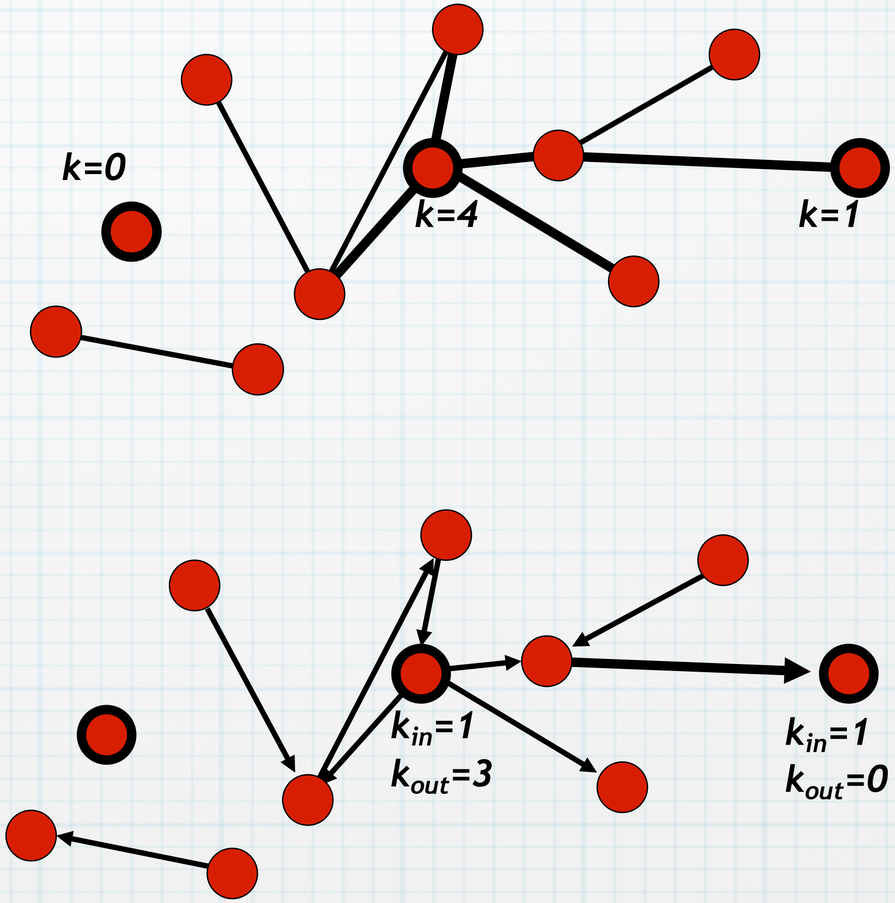
* degree k
  * indegree k_{in}
  * outdegree k_{out}





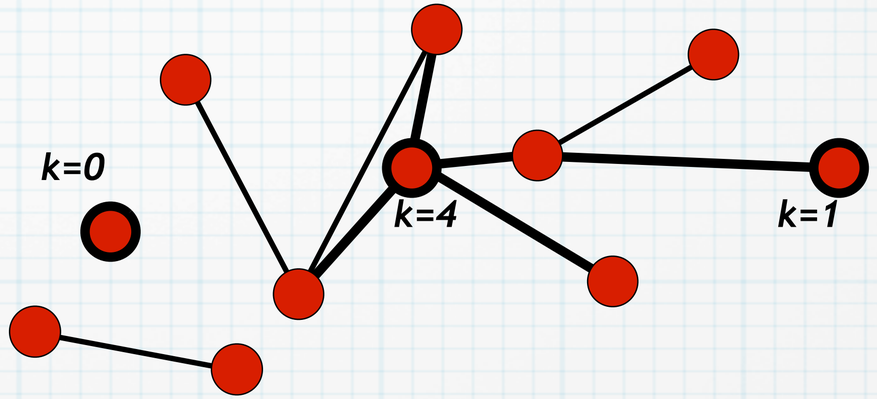
More definitions

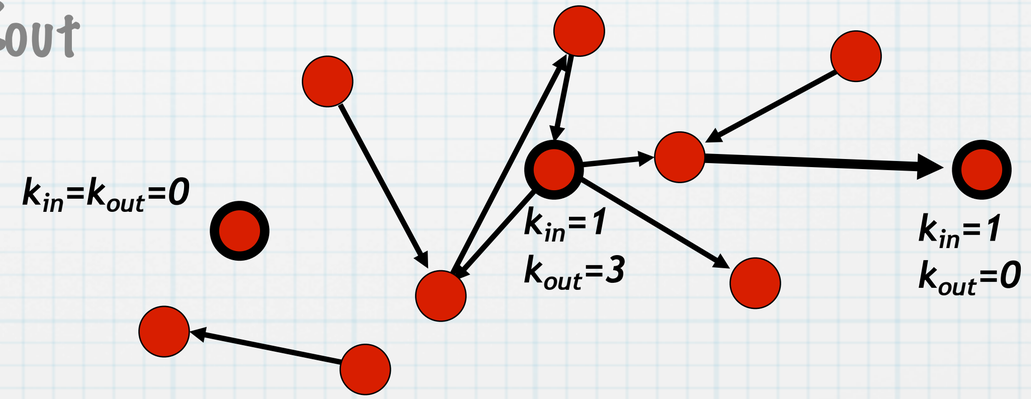
* degree k
  * indegree k_{in}
  * outdegree k_{out}



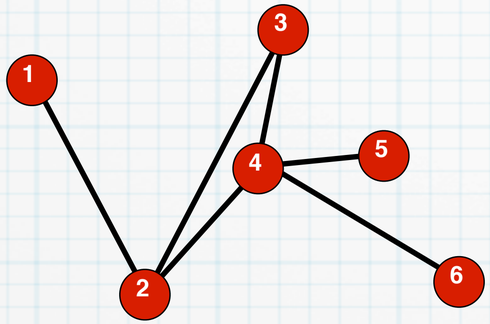
More definitions

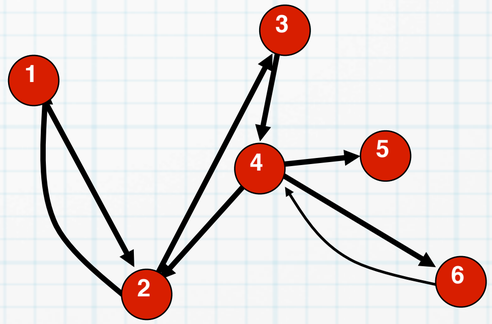
* degree k
  * indegree k_{in}
  * outdegree k_{out}





Python and NetworkX

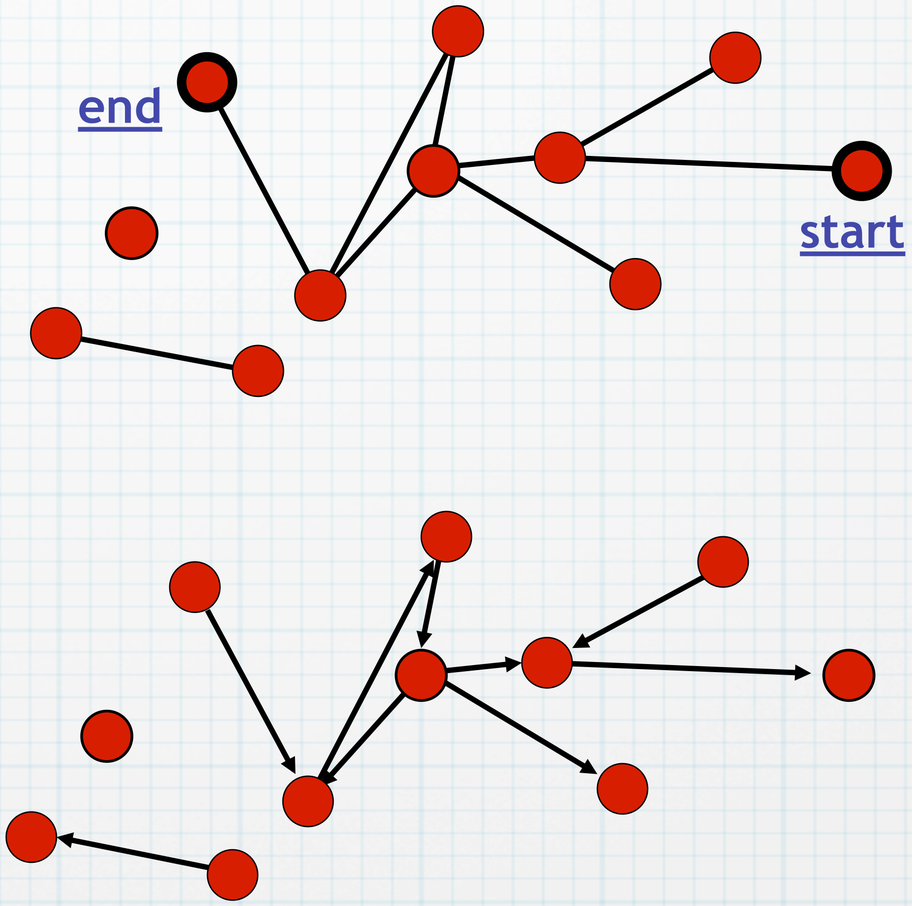





```
G.degree(2)
G.degree()
G.degree(4)

D.degree(4)
D.in_degree(4)
D.out_degree(4)
```

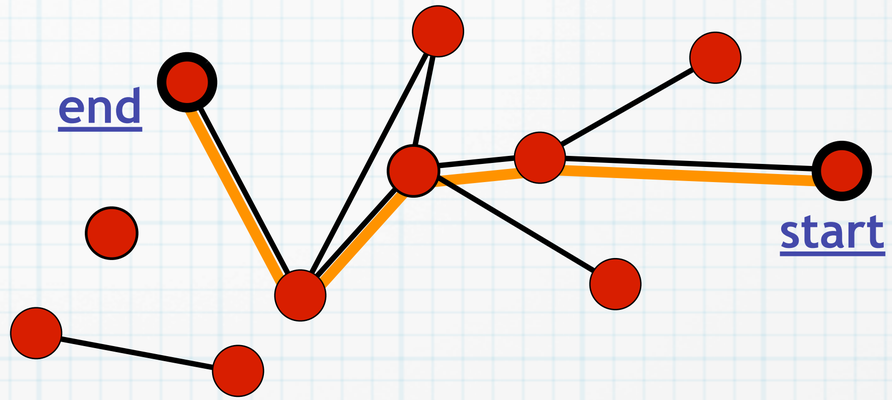
Distance

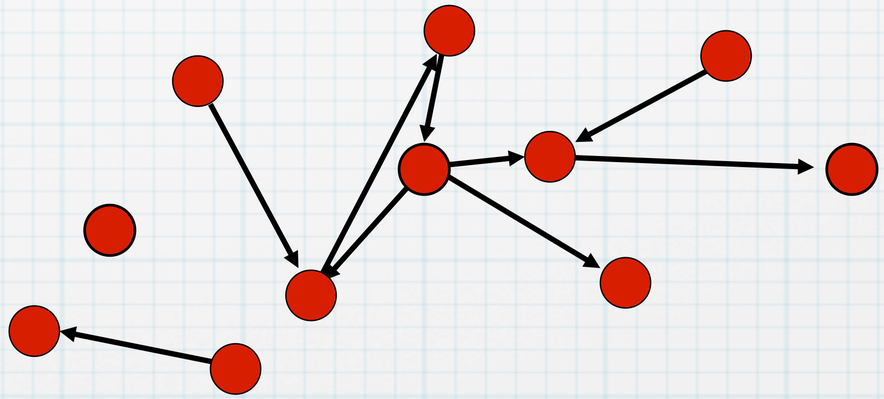
* path
* shortest path



Distance

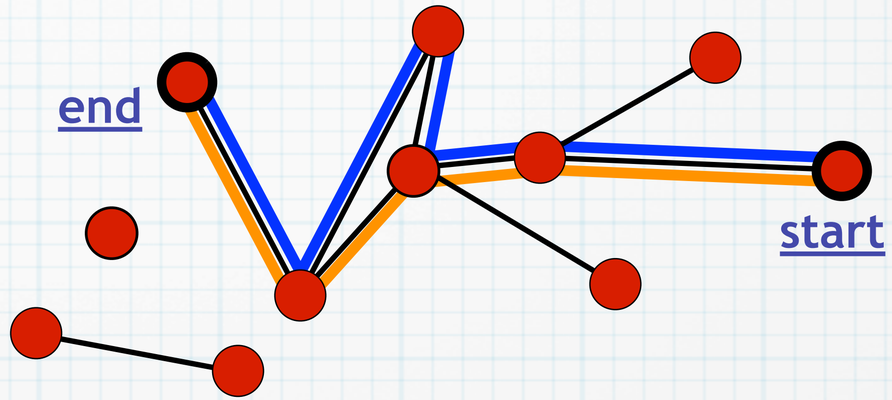
* path
* shortest path

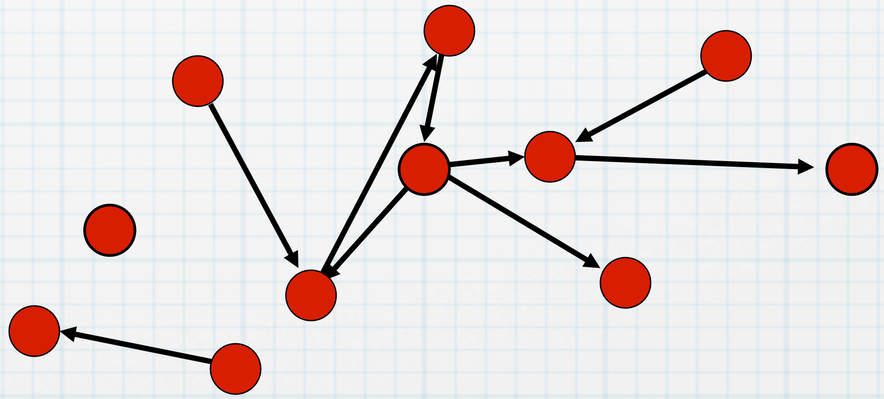




Distance

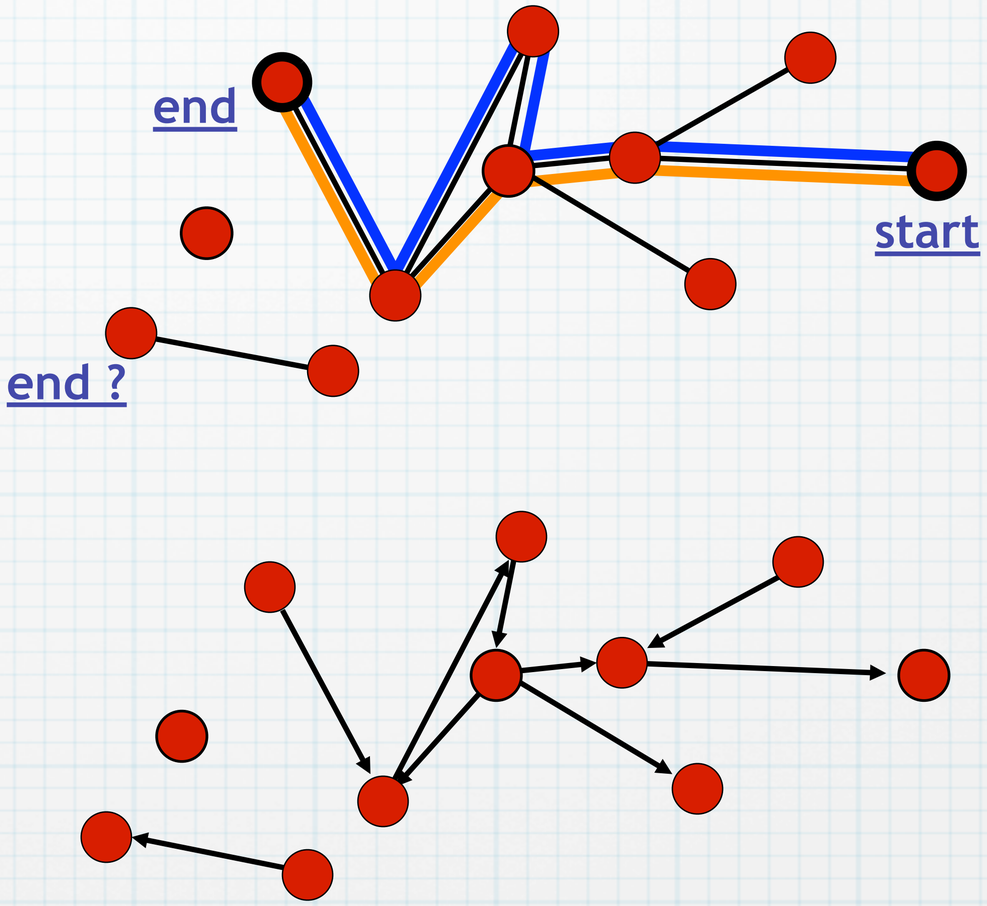
* path
* shortest path





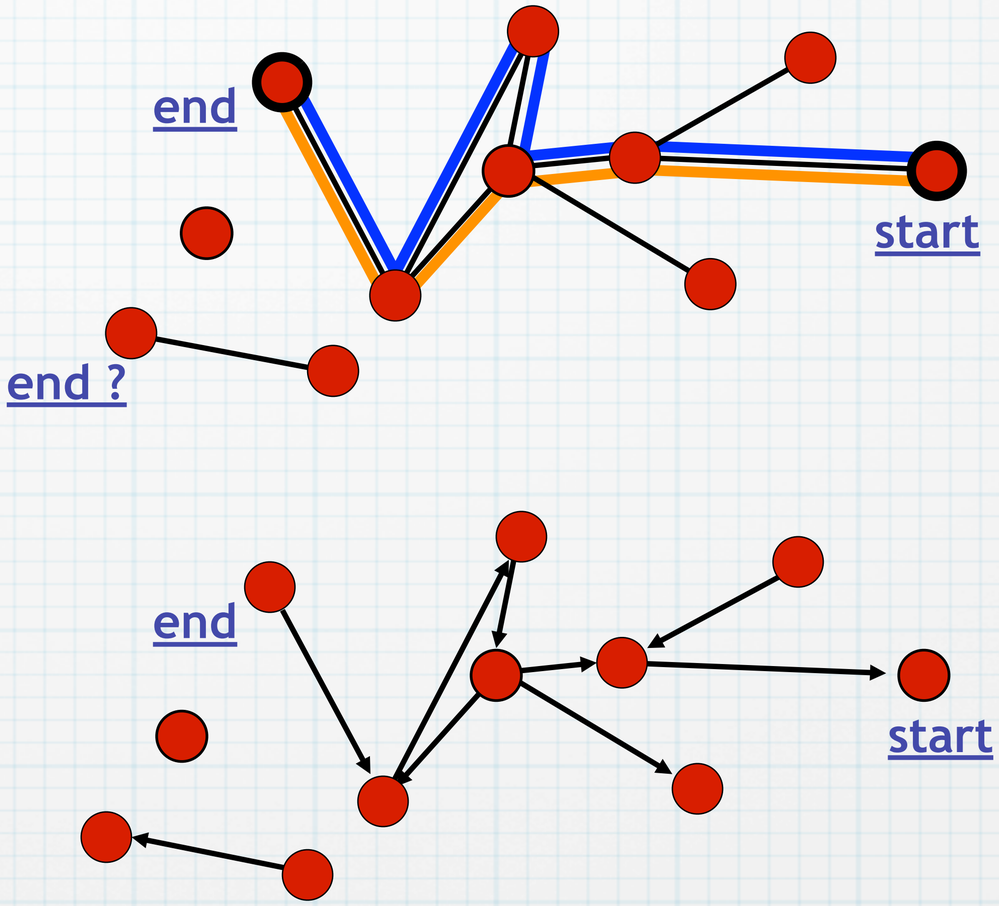
Distance

* path
* shortest path



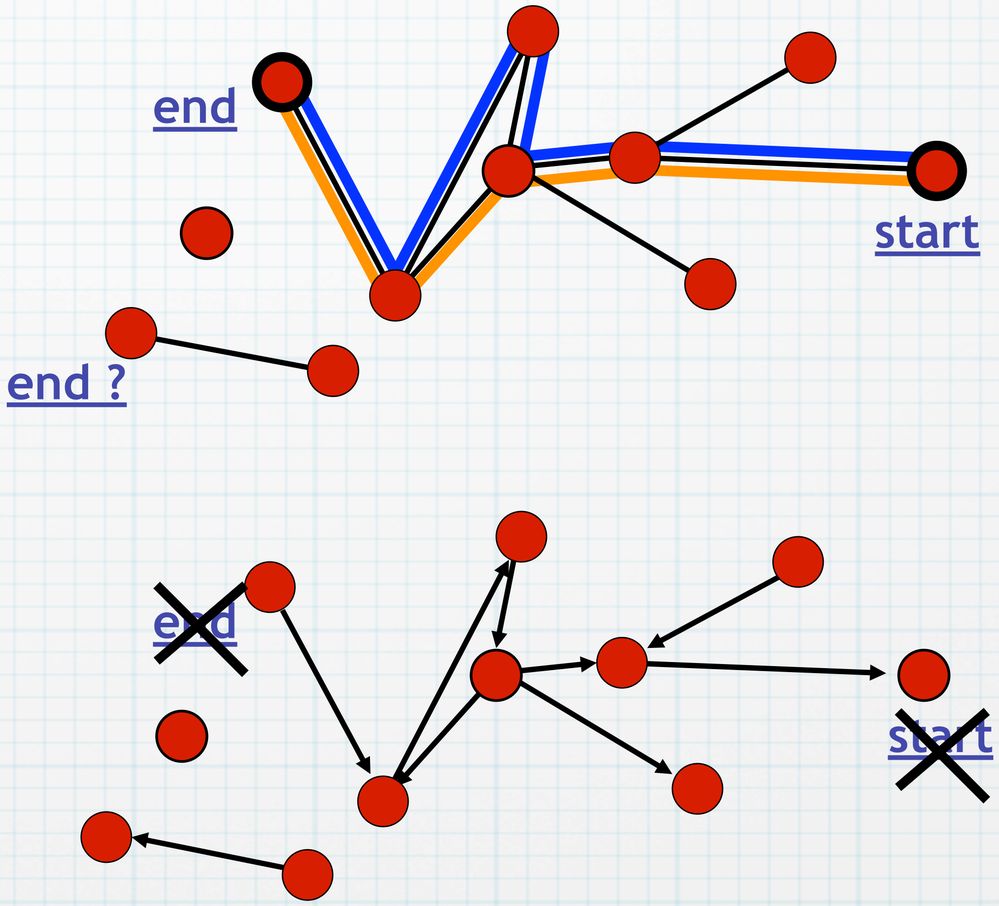
Distance

* path
* shortest path



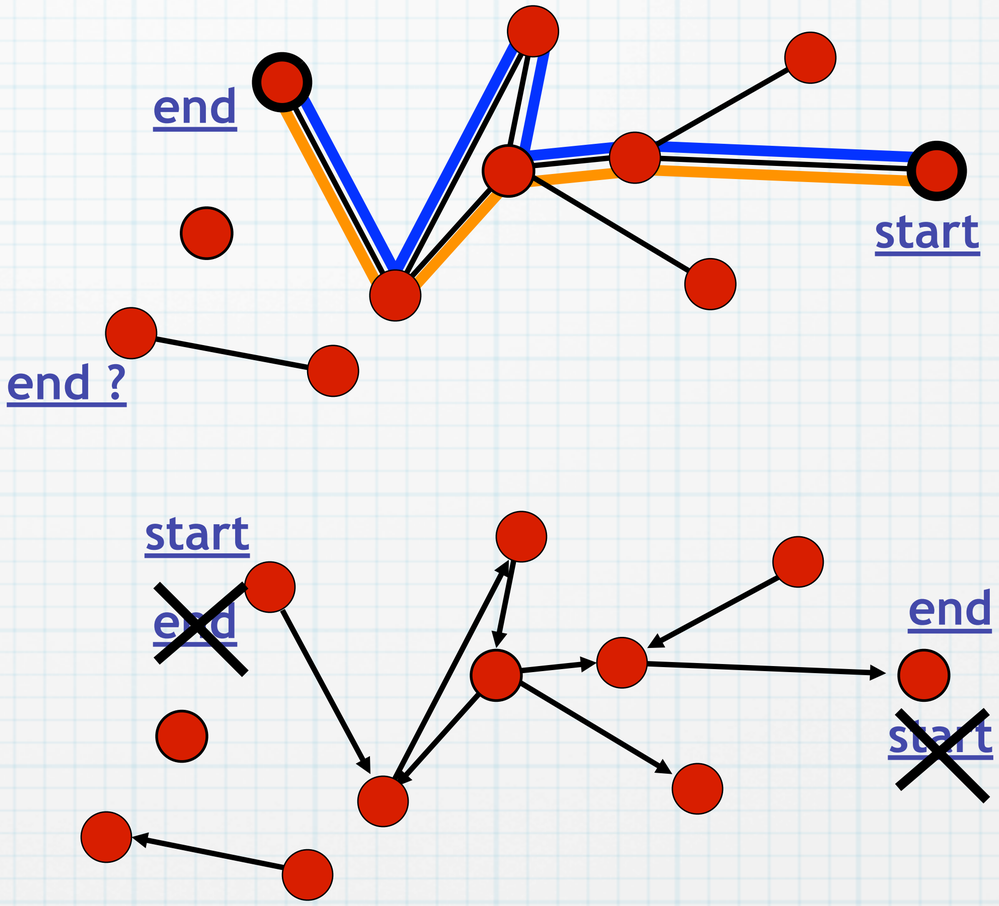
Distance

* path
* shortest path



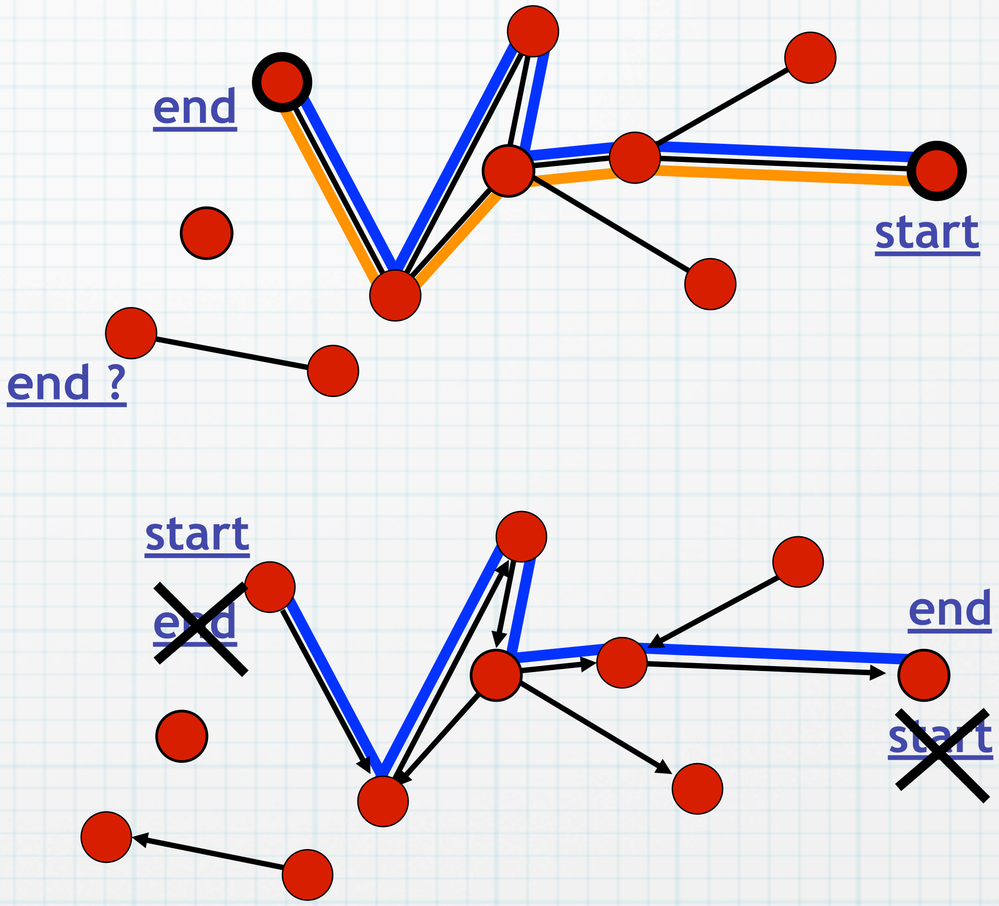
Distance

* path
* shortest path



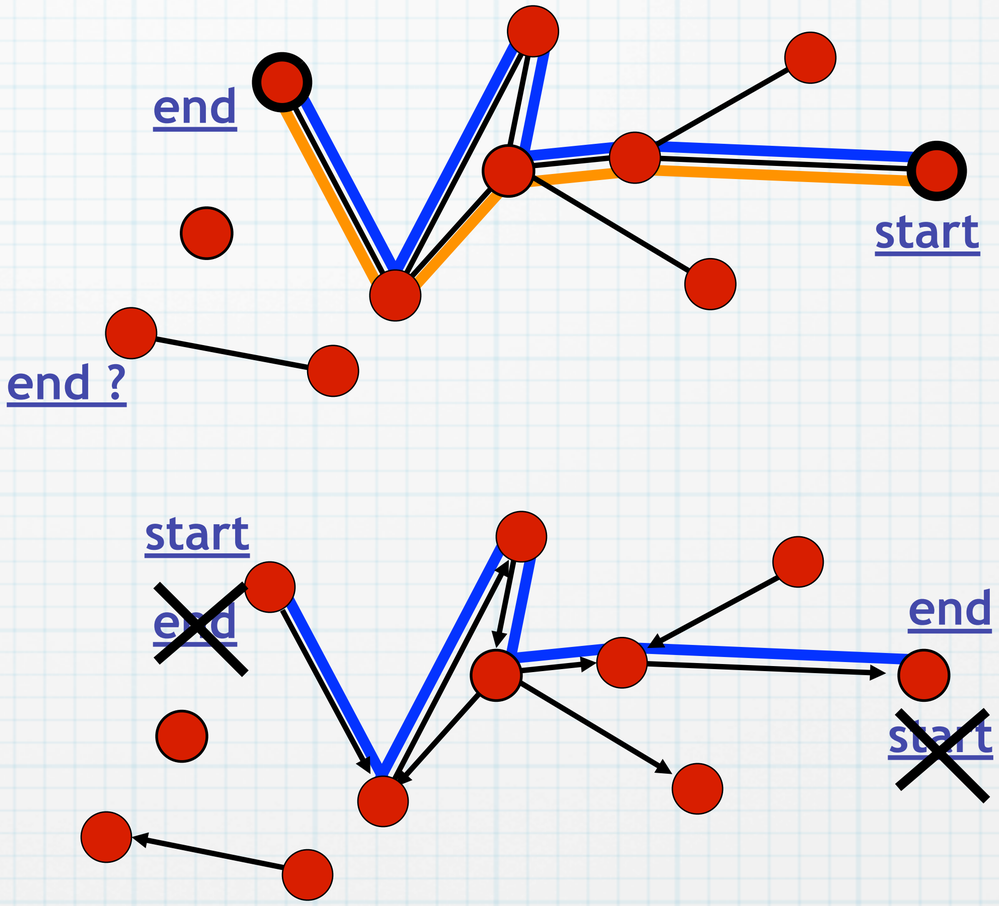
Distance

* path
* shortest path



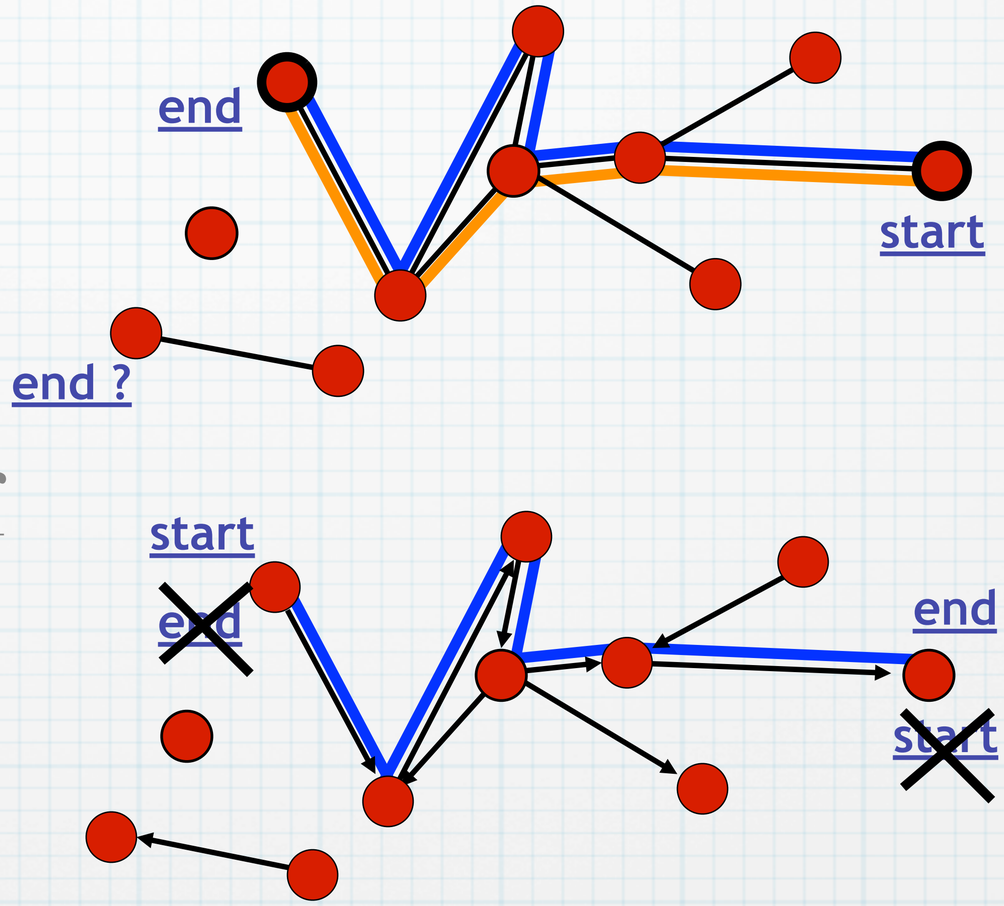
Distance

* path
* shortest path
* tree: # paths?



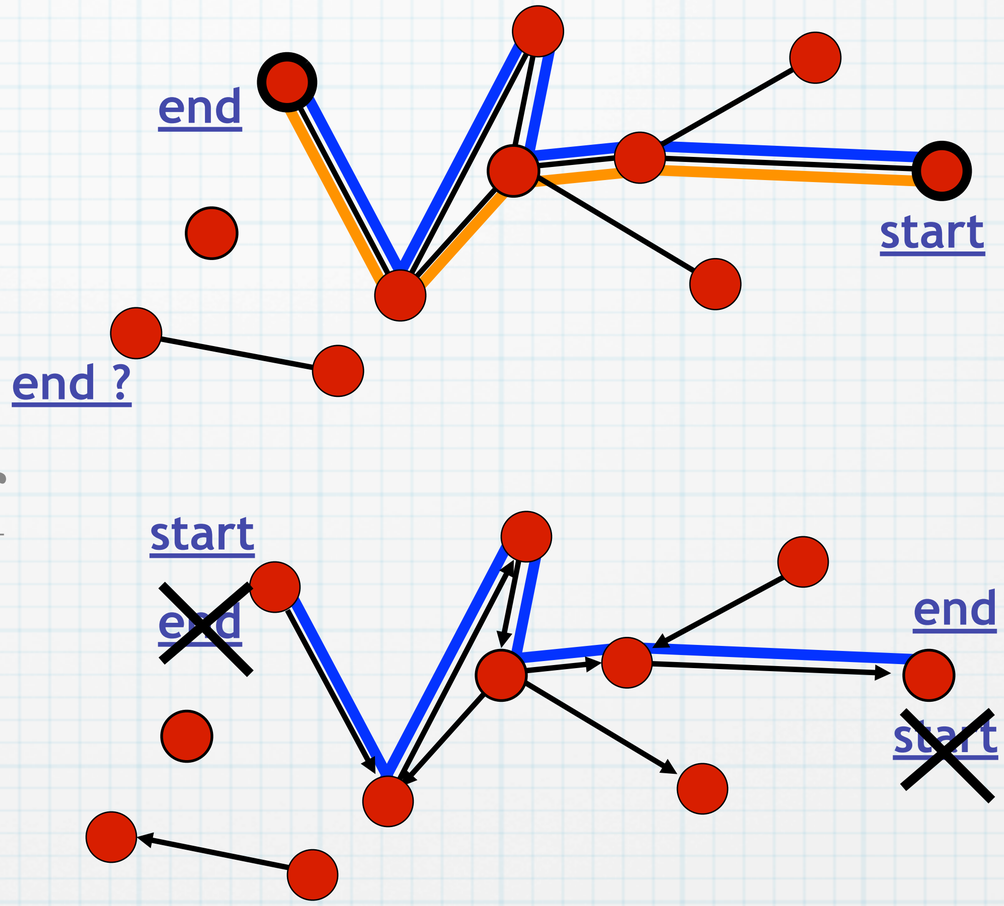
Distance

* path
* shortest path
* tree: # paths?
* Diameter: longest shortest path length

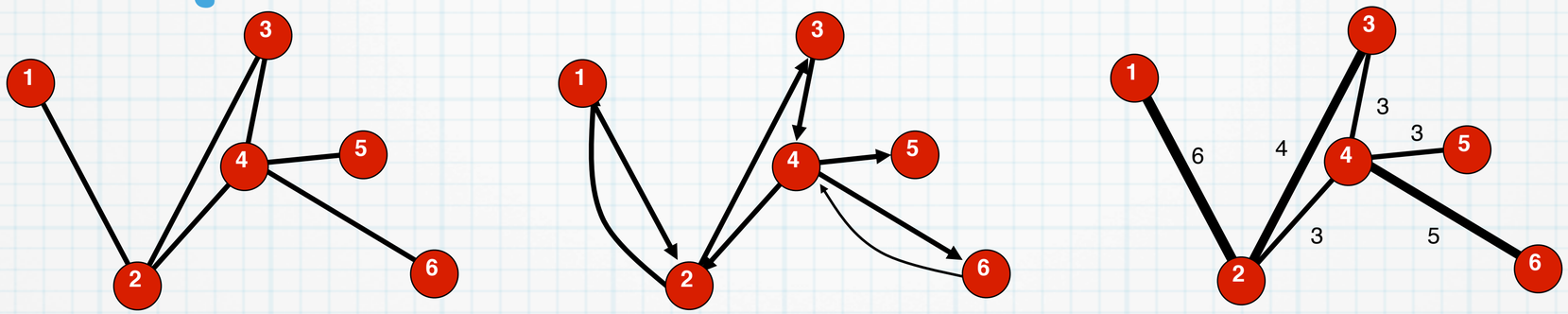


Distance

* path
* shortest path
* tree: # paths?
* Diameter: longest shortest path length
* APL: average shortest path length



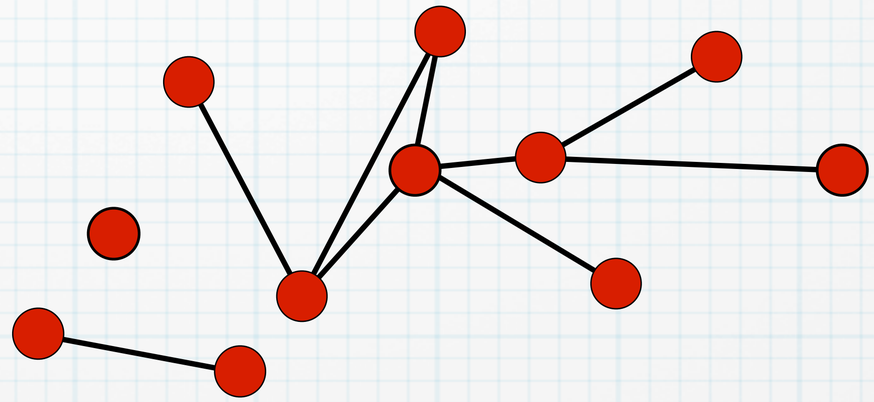
Python and NetworkX

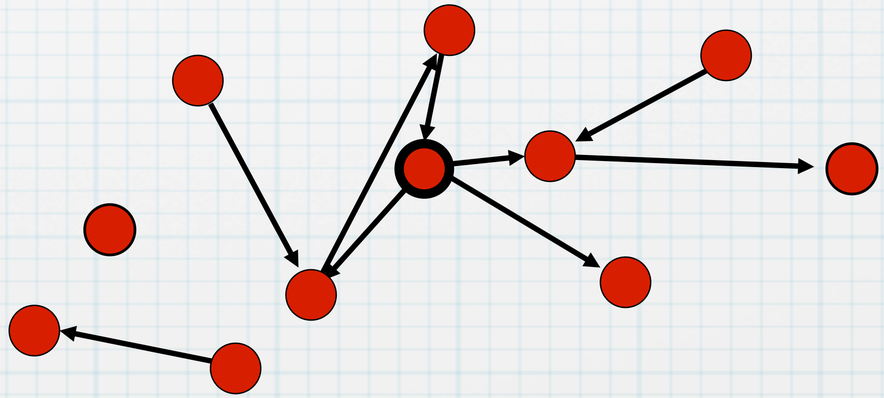



```
nx.has_path(G,1,6)
nx.shortest_path(G,1,6)
nx.shortest_path_length(G,1,6)
nx.shortest_path(G,1)
nx.shortest_path(G)
nx.average_shortest_path_length(G)

nx.shortest_path(D,1,6)
nx.has_path(D,1,5)
nx.has_path(D,5,1)

nx.shortest_path_length(W,1,6)
nx.shortest_path_length(W,1,6,'weight')
```

More definitions

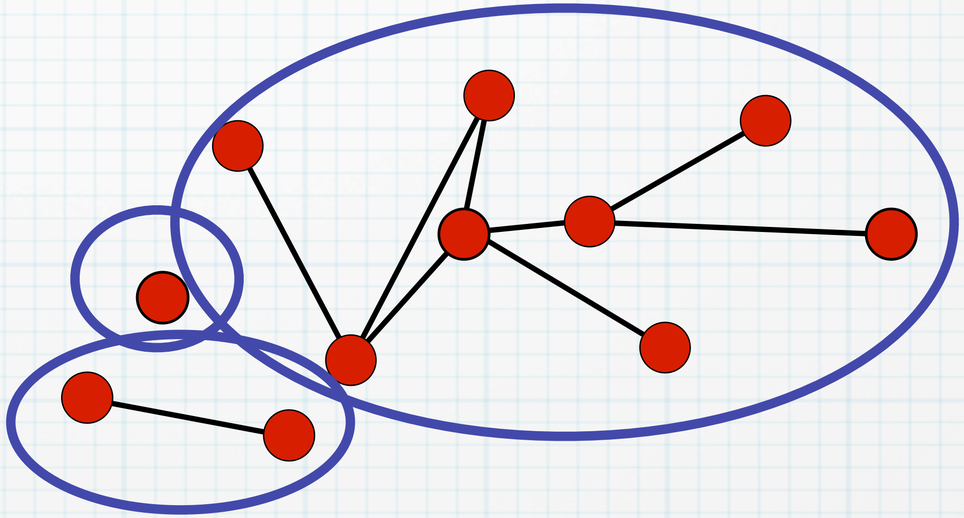
* connectedness
  * giant component
  * isolates
* components
  * in- and out-
  * weak, strong

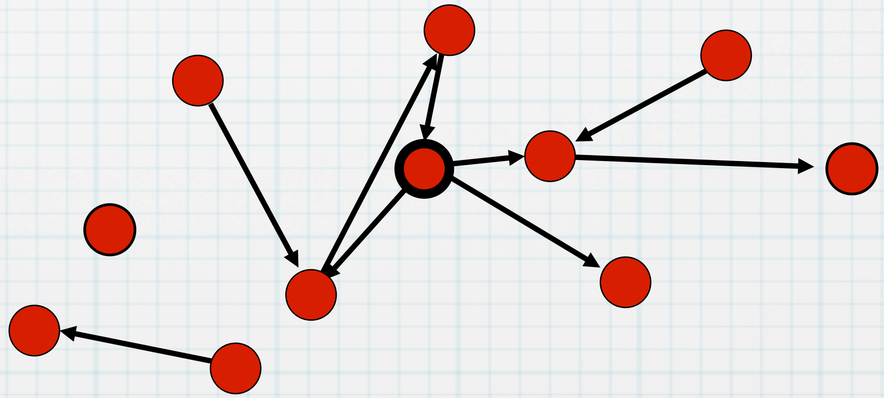




More definitions

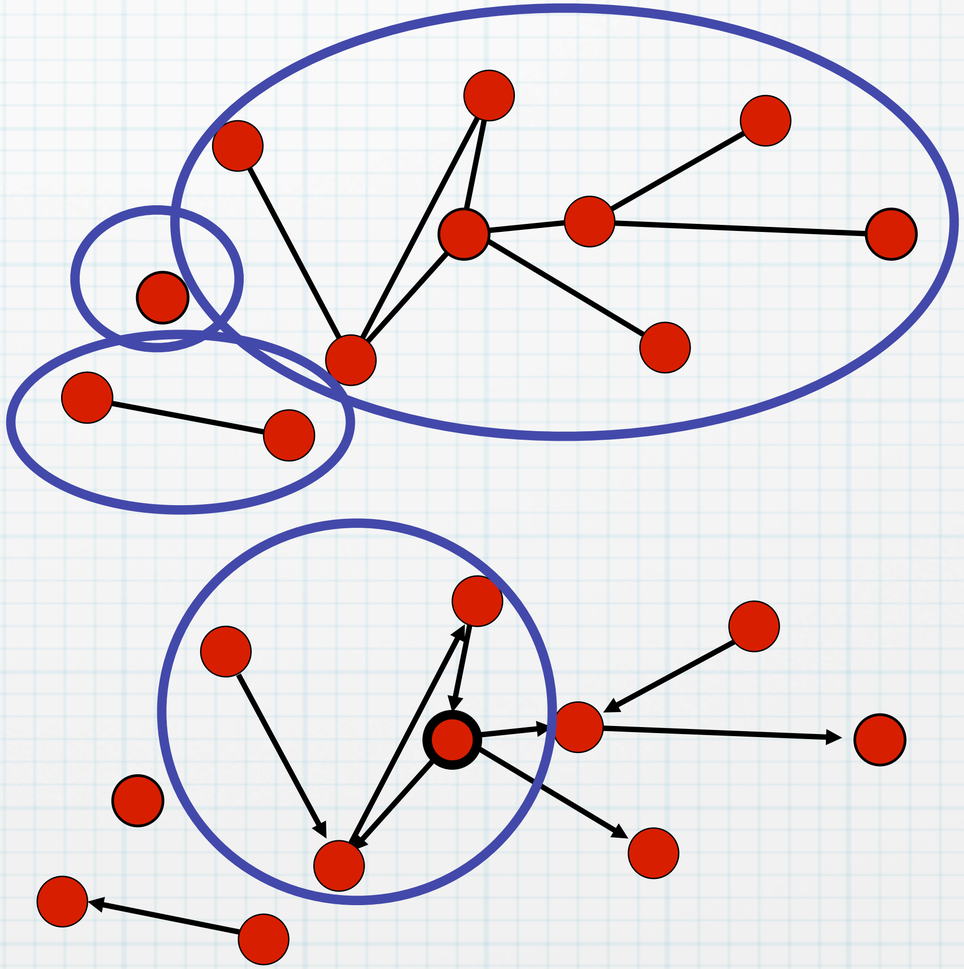
* connectedness
  * giant component
  * isolates
* components
  * in- and out-
  * weak, strong





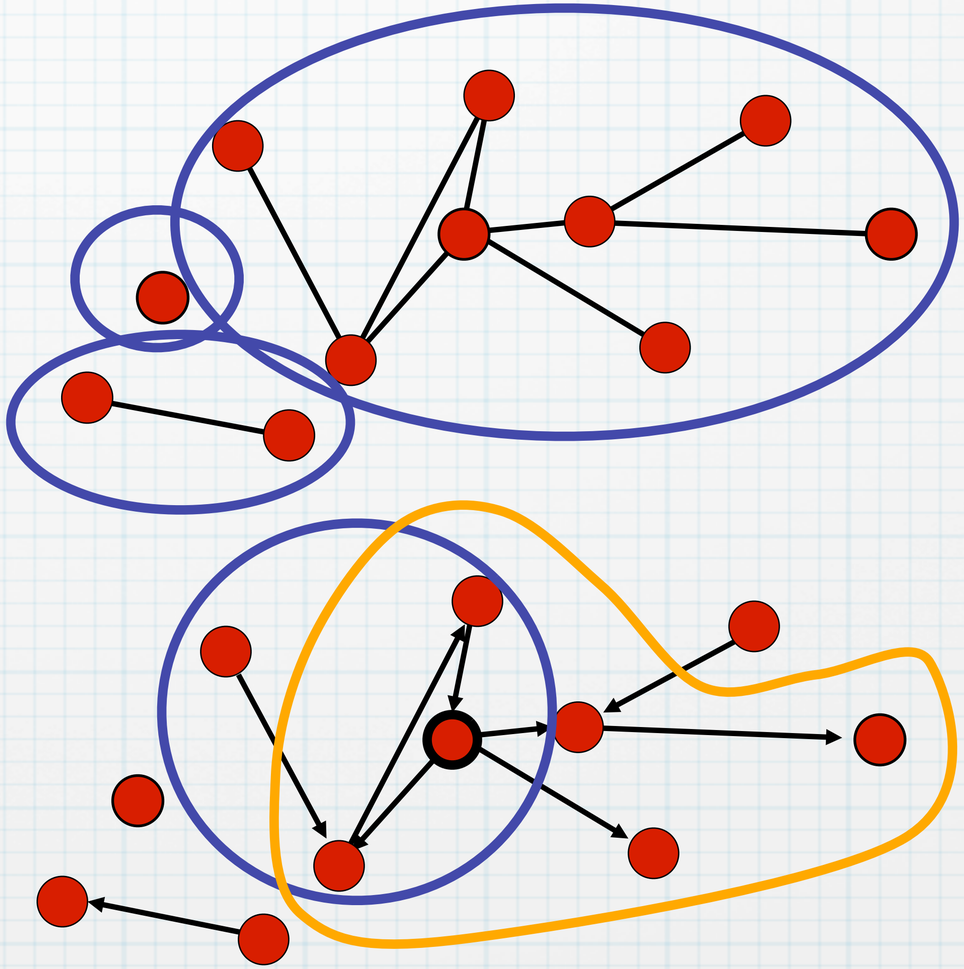
More definitions

* connectedness
  * giant component
  * isolates
* components
  * in- and out-
  * weak, strong

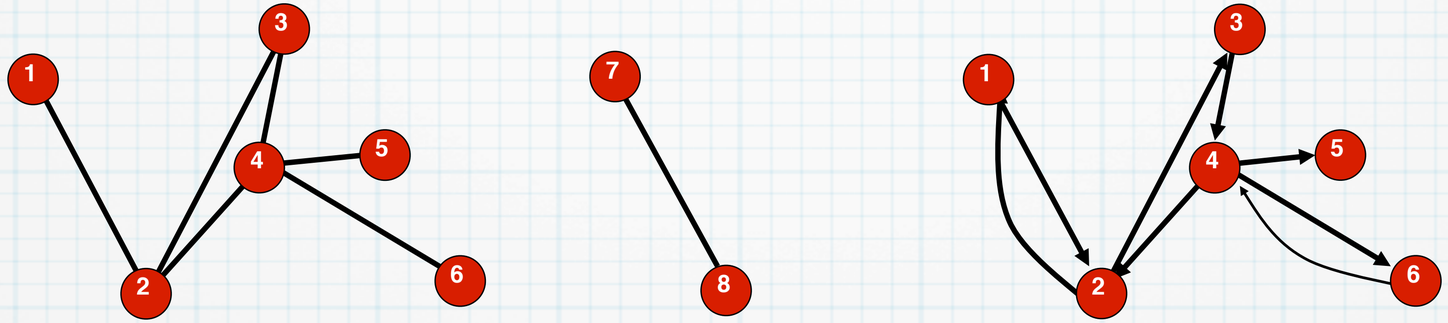


More definitions

* connectedness
  * giant component
  * isolates
* components
  * in- and out-
  * weak, strong



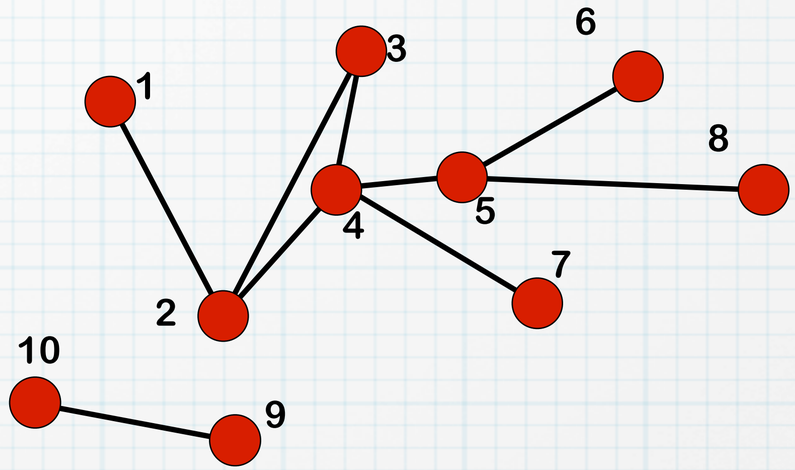
Python and NetworkX



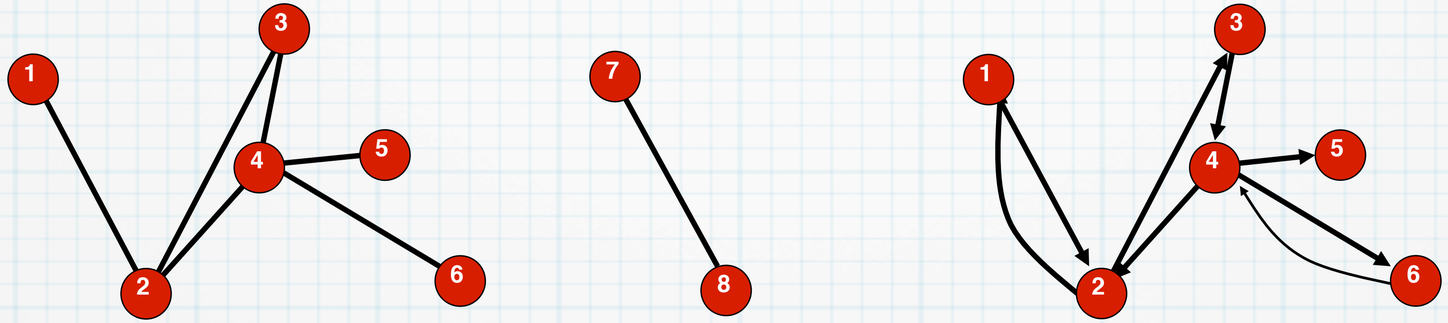

```
nx.is_connected(G)
sorted(nx.connected_components(G))
G.add_edge(7,8)
nx.is_connected(G)
sorted(nx.connected_components(G))

nx.is_weakly_connected(D)
nx.is_strongly_connected(D)
sorted(nx.strongly_connected_components(D))
```

Basic measures

- Networks:
  - size N: # vertices
  - E: # edges
  - $E_{max} = N(N-1)/2$ → complete graph
  - density: E/E_{max}
  - if density <<1 (low) → sparse graph



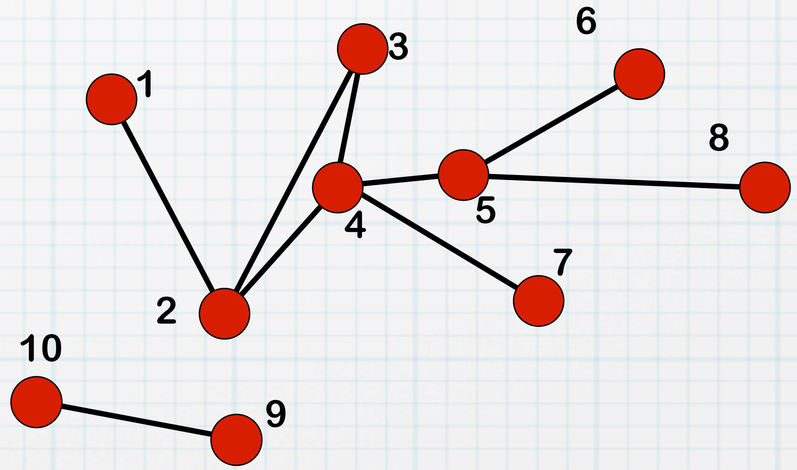
Python and NetworkX



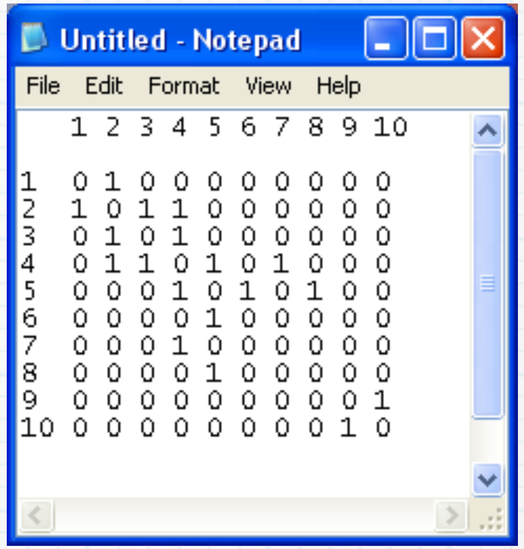

```
G.number_of_nodes()
G.number_of_edges()
nx.density(G)
nx.density(D)
```

Mathematical representation

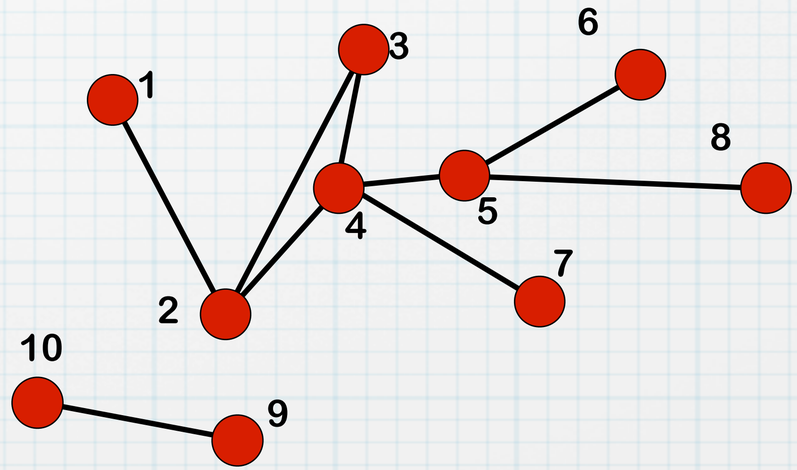
* Adjacency matrix $A=(A_{ij})$
* $A_{ij} = 1$ if i linked to j
* $A_{ij} = 0$ otherwise
  * $A_{47} = ?$ $A_{74} = ?$
  * $A_{13} = ?$
* a.k.a. sociomatrix, connectivity matrix



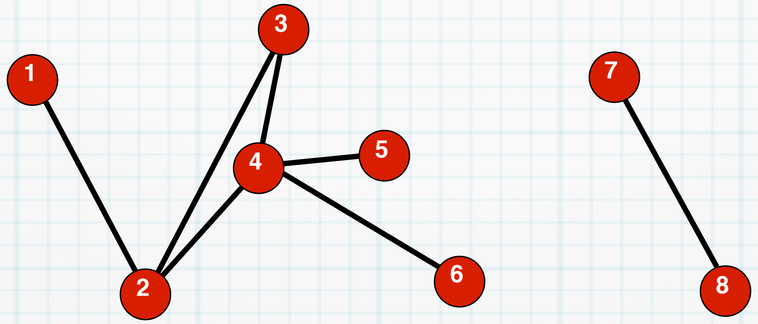
Mathematical representation



- no self-loops: $A_{ii} = 0$
- undirected network: $A_{ij} = A_{ji}$
- → symmetric matrix
- no multiple edges: $A_{ij} = 0$ or 1



Python and NetworkX




```
nx.adjacency_matrix(G)
print(nx.adjacency_matrix(G))
G.edge[3][4]
G.edge[3][4]['color']='blue'
G.edge[3][4]
G.edge[4]
```

Local measures

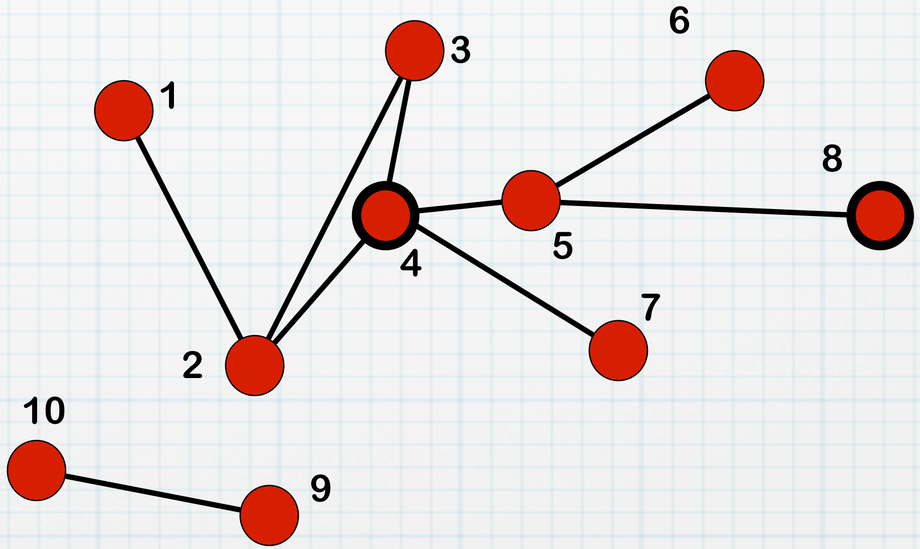
Degree: # links departing from the node

$$k_i = A_{i1} + A_{i2} + \ldots + A_{iN} = \Sigma_j A_{ij}$$

Untitled - Notepad

File Edit Format View Help

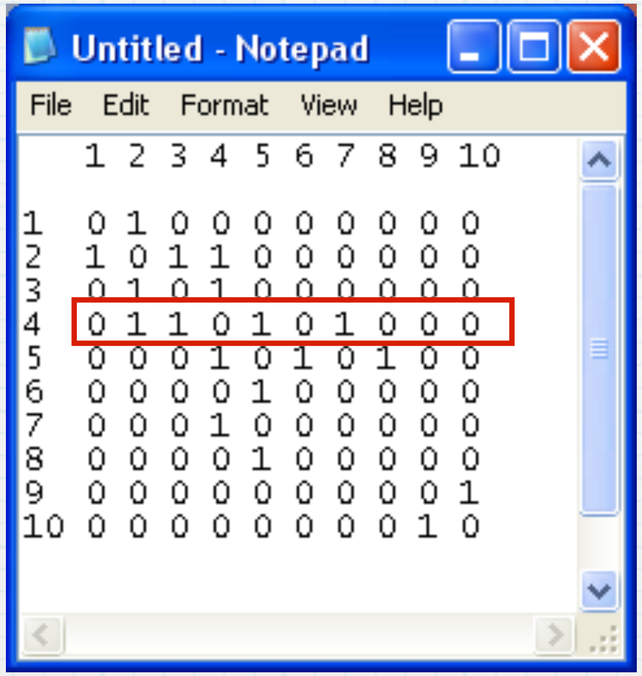
	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0	0
4	0	1	1	0	1	0	1	0	0	0
5	0	0	0	1	0	1	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	1	0

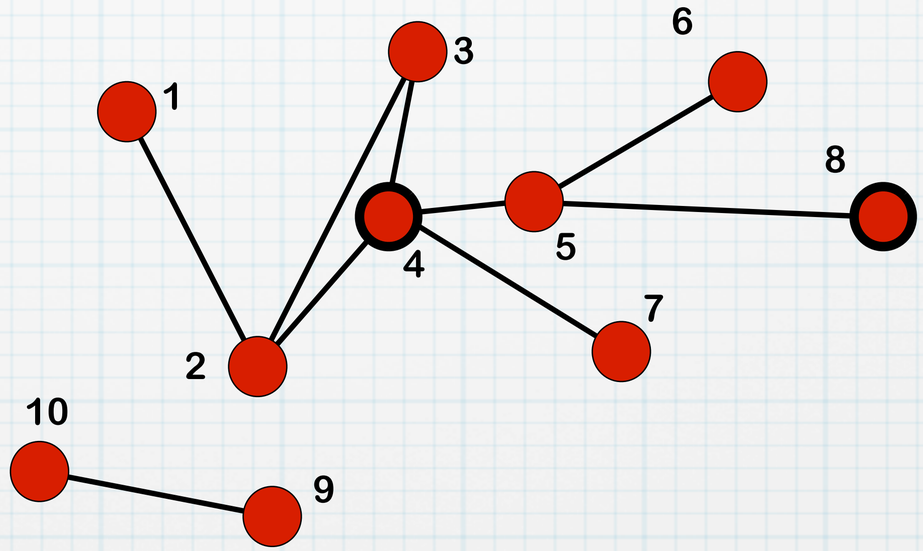


Local measures

Degree: # links departing from the node

$$k_i = A_{i1} + A_{i2} + \ldots + A_{iN} = \Sigma_j A_{ij}$$

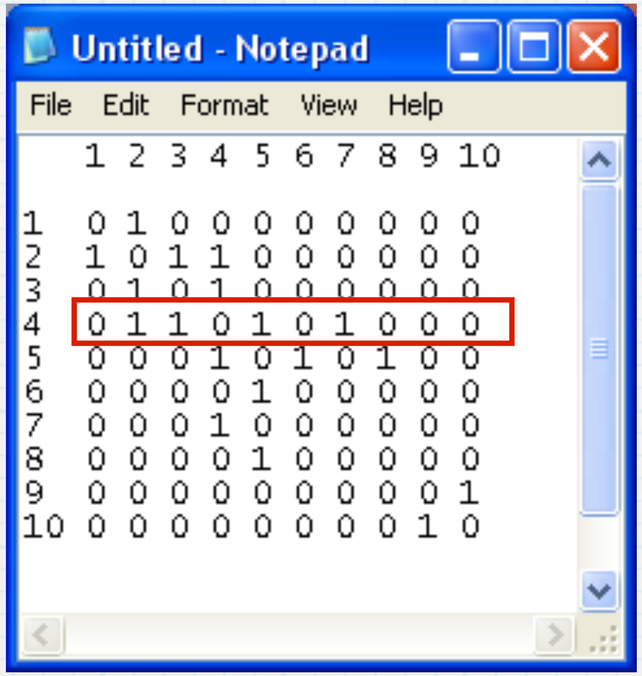


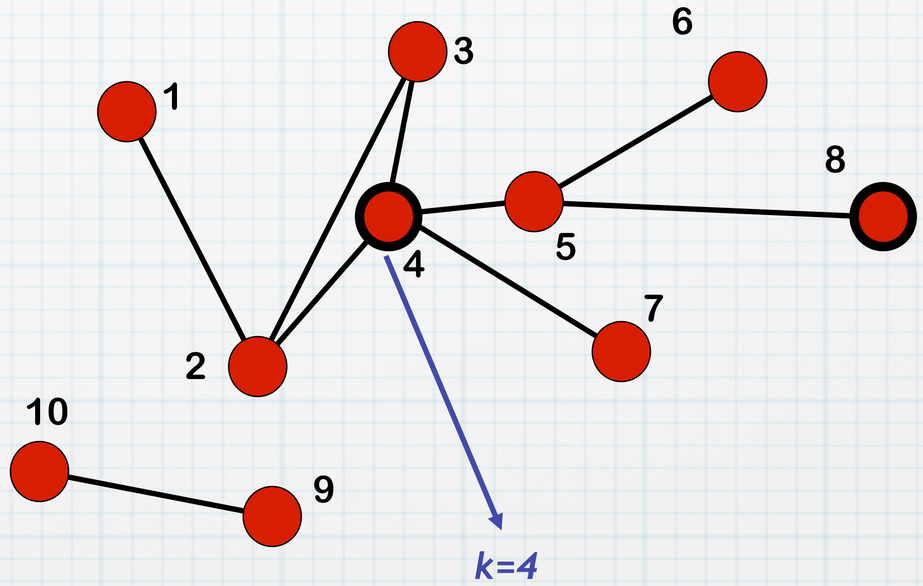


Local measures

Degree: # links departing from the node

$$k_i = A_{i1} + A_{i2} + \ldots + A_{iN} = \Sigma_j A_{ij}$$

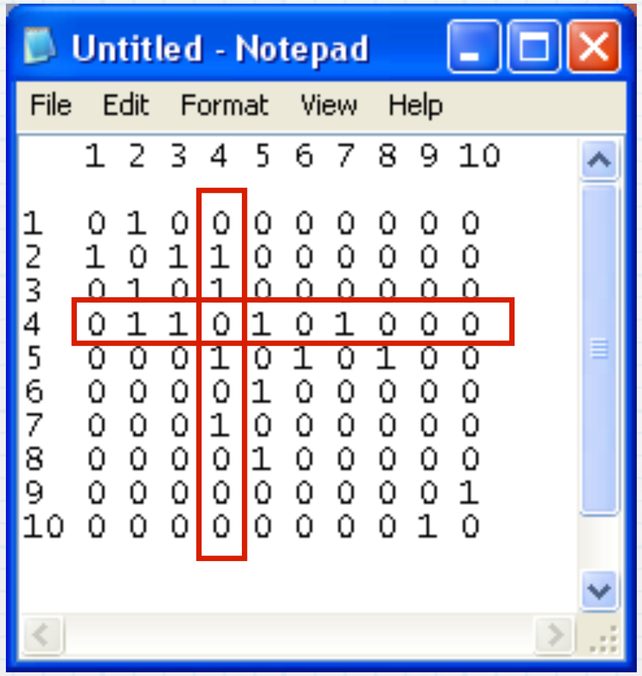


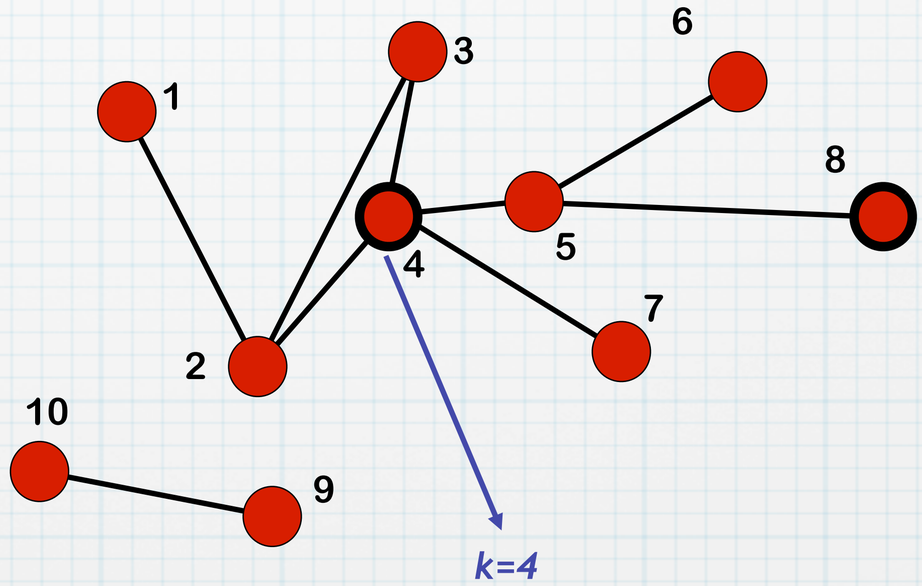


Local measures

Degree: # links departing from the node

$$k_i = A_{i1} + A_{i2} + \ldots + A_{iN} = \Sigma_j A_{ij}$$

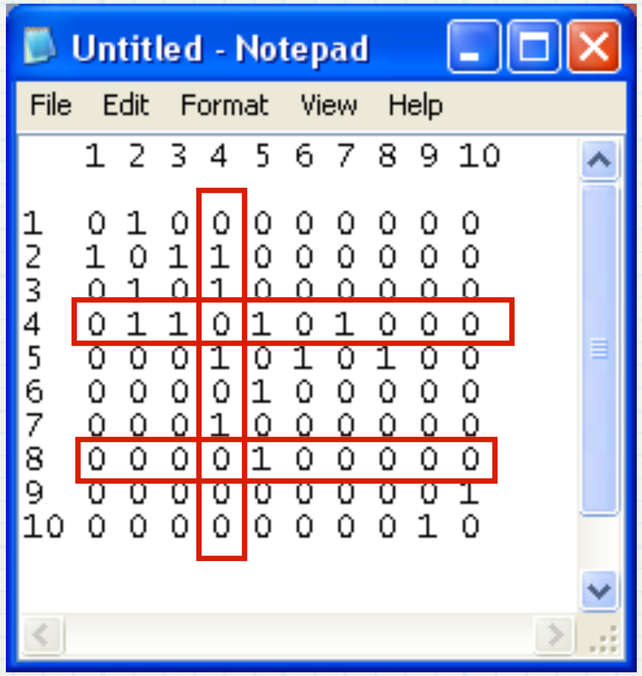


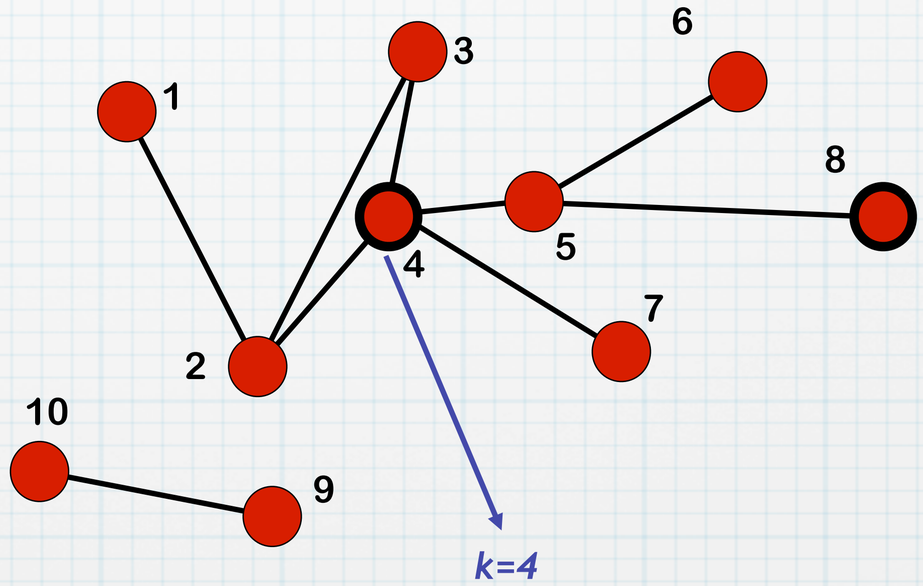


Local measures

Degree: # links departing from the node

$$k_i = A_{i1} + A_{i2} + \ldots + A_{iN} = \Sigma_j A_{ij}$$

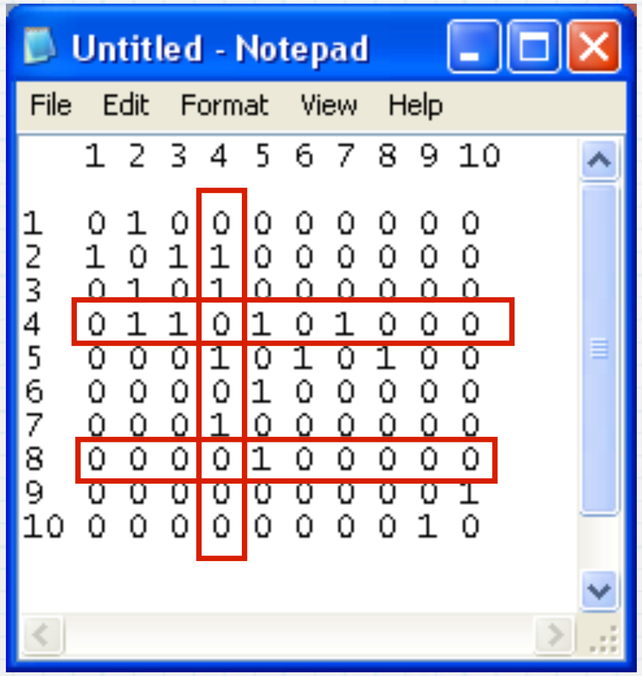


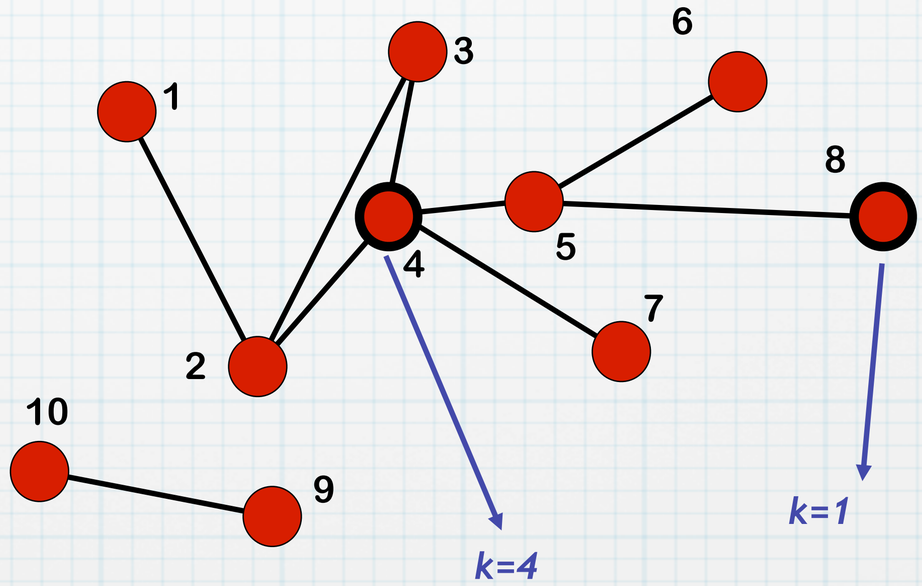


Local measures

Degree: # links departing from the node

$$k_i = A_{i1} + A_{i2} + \ldots + A_{iN} = \Sigma_j A_{ij}$$





- Attendance
- Schedule update
- Review

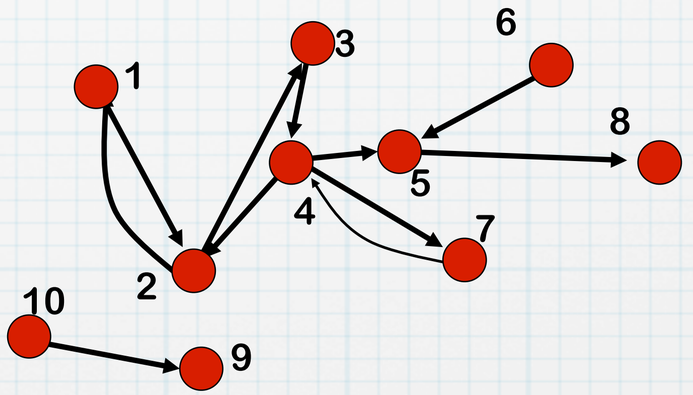
Mathematical representation

directed_matrix - No...

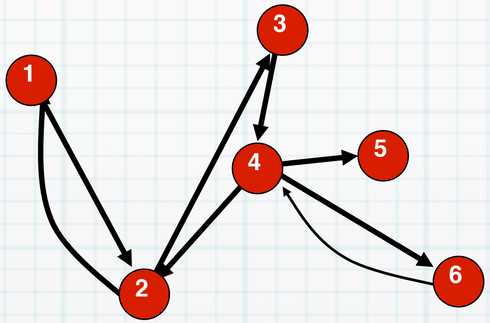
File Edit Format View Help

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	1	0	1	0	0	0
5	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	1	0	

* directed networks: not symmetric !!!
* $i \rightarrow j \qquad A_{ij} = 1$



Python and NetworkX




```
print(nx.adjacency_matrix(D))
D.edge[3][4]
D.edge[4][3]
D.edge[4]
```

More local measures

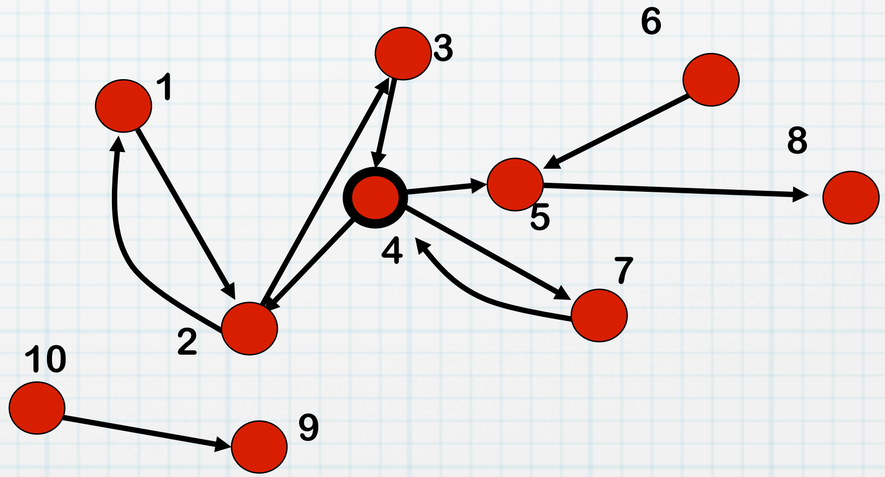
Directed networks have in-degree and out-degree:

$k_{in}(i) = \sum_j A_{ji}$ $k_{out}(i) = \sum_j A_{ij}$

directed_matrix - No...

File Edit Format View Help

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	1	0	1	0	0	0
5	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1	0



More local measures

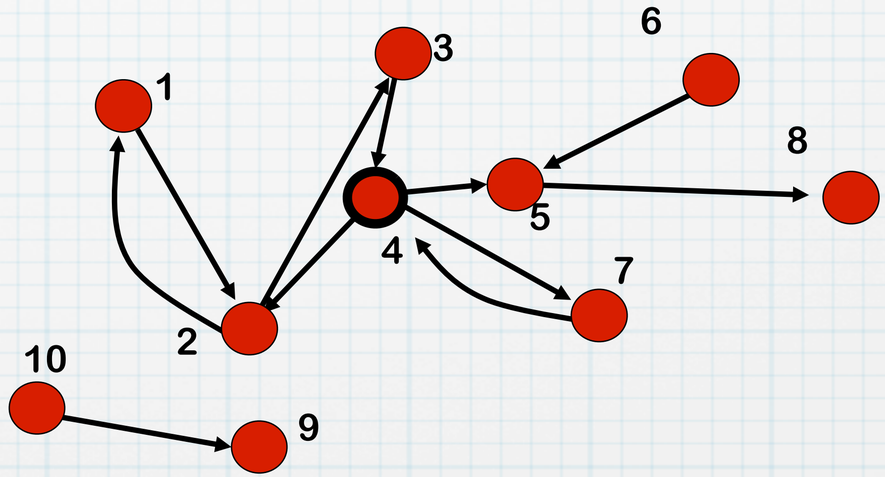
Directed networks have in-degree and out-degree:

$k_{in}(i) = \sum_j A_{ji}$ $k_{out}(i) = \sum_j A_{ij}$

directed_matrix - No...

File Edit Format View Help

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	1	0	1	0	0	0
5	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1	0



More local measures

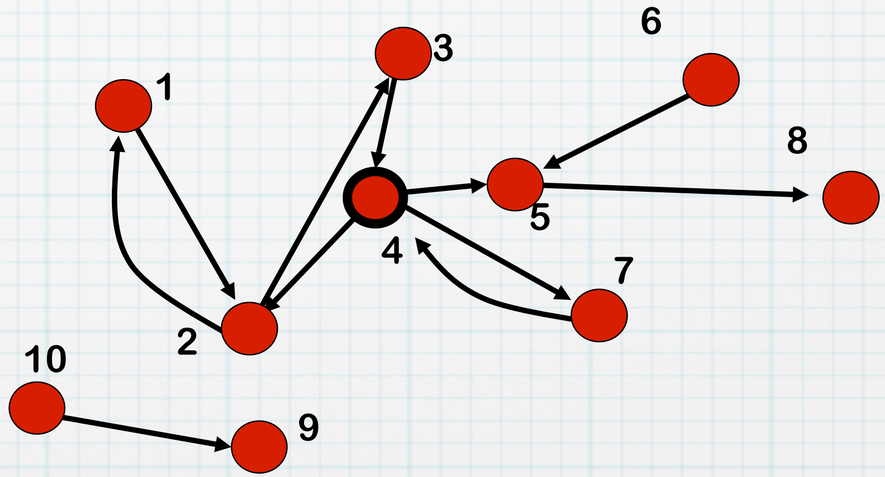
Directed networks have in-degree and out-degree:

$k_{in}(i) = \sum_j A_{ji}$ $k_{out}(i) = \sum_j A_{ij}$

directed_matrix - No...

File Edit Format View Help

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	1	0	1	0	0	0
5	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1	0



More local measures

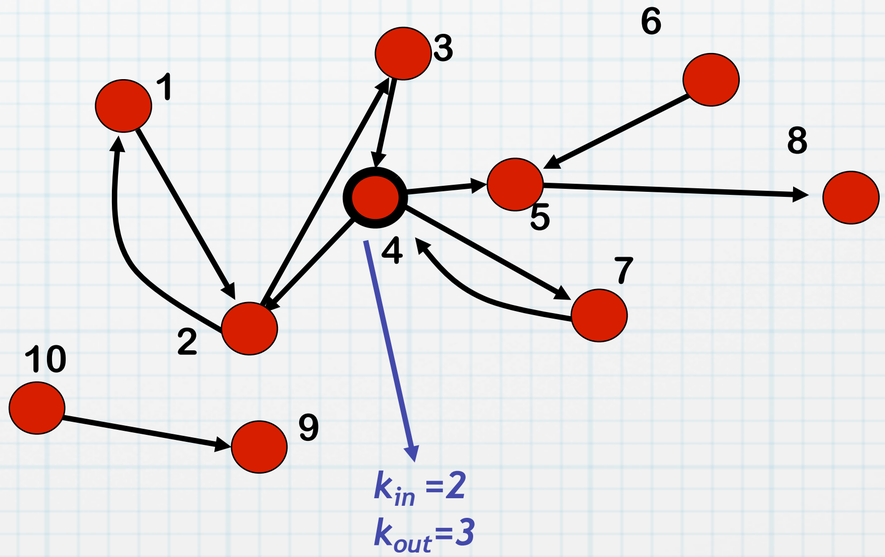
Directed networks have in-degree and out-degree:

$k_{in}(i) = \sum_j A_{ji}$ $k_{out}(i) = \sum_j A_{ij}$

directed_matrix - No...

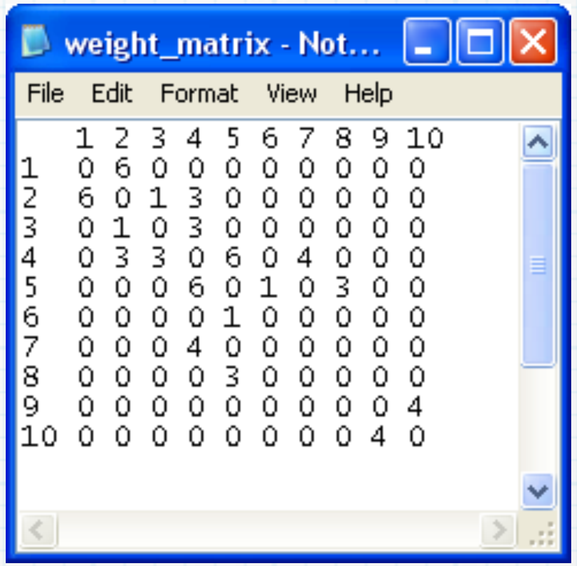
File Edit Format View Help

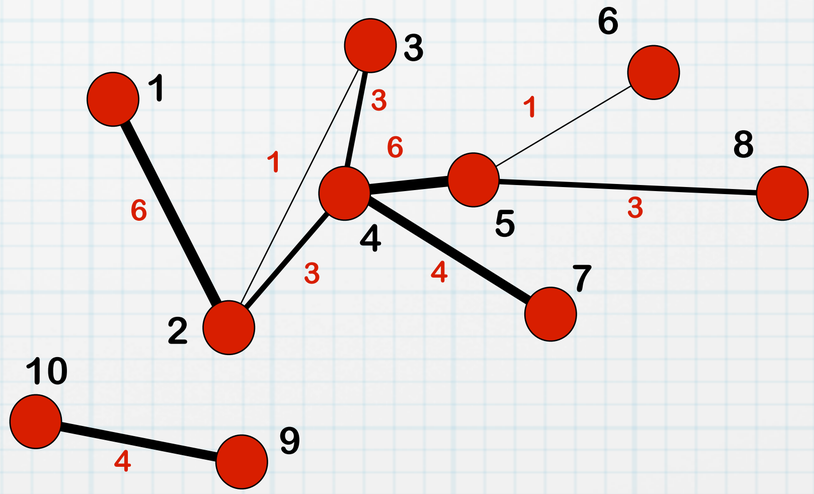
	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	1	0	1	0	0	0
5	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1	0



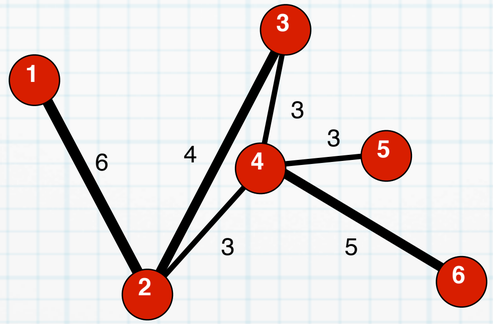
Mathematical representation

- weighted networks: weight matrix
  - undirected → symmetric
  - directed → asymmetric
- w_{ij} = weight of link i → j
- w_{ij} = 0 if no links





Python and NetworkX




```
print(nx.adjacency_matrix(W))
W.edge[2][3]
W.edge[2]
W.edge[2][3]['weight'] = 1
W.edge[2][3]
W.edge[2]
```

More local measures

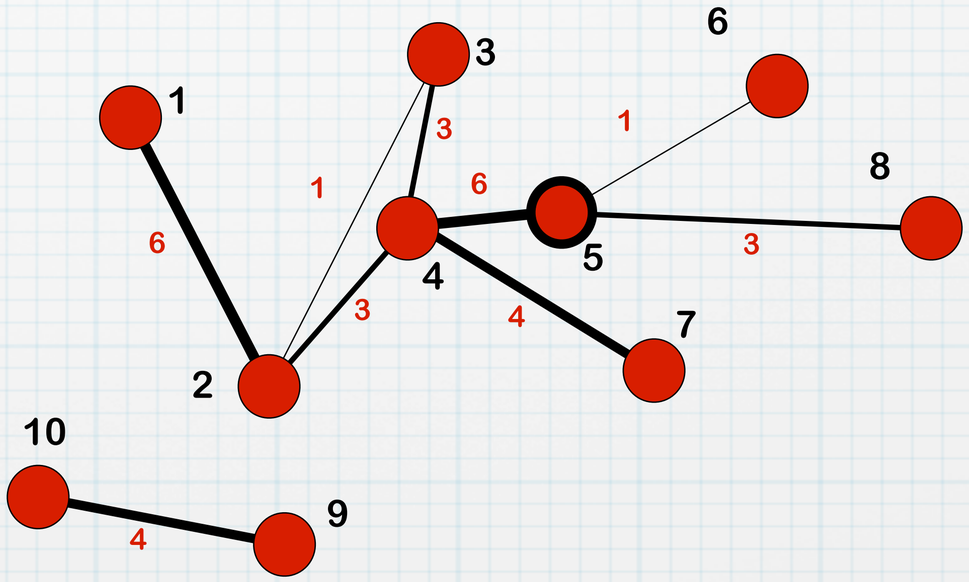
In weighted networks, degree generalizes to strength (a.k.a. weighted degree):

$$s_i = w_{i1} + w_{i2} + \ldots + w_{iN} = \sum_j w_{ij}$$

weight_matrix - Not...

File Edit Format View Help

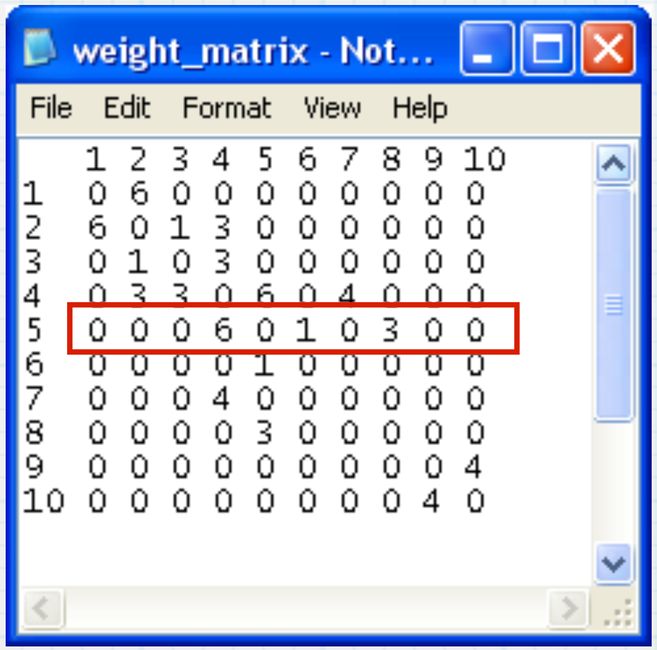
	1	2	3	4	5	6	7	8	9	10
1	0	6	0	0	0	0	0	0	0	0
2	6	0	1	3	0	0	0	0	0	0
3	0	1	0	3	0	0	0	0	0	0
4	0	3	3	0	6	0	4	0	0	0
5	0	0	0	6	0	1	0	3	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	4	0	0	0	0	0	0
8	0	0	0	0	3	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	4
10	0	0	0	0	0	0	0	0	4	0

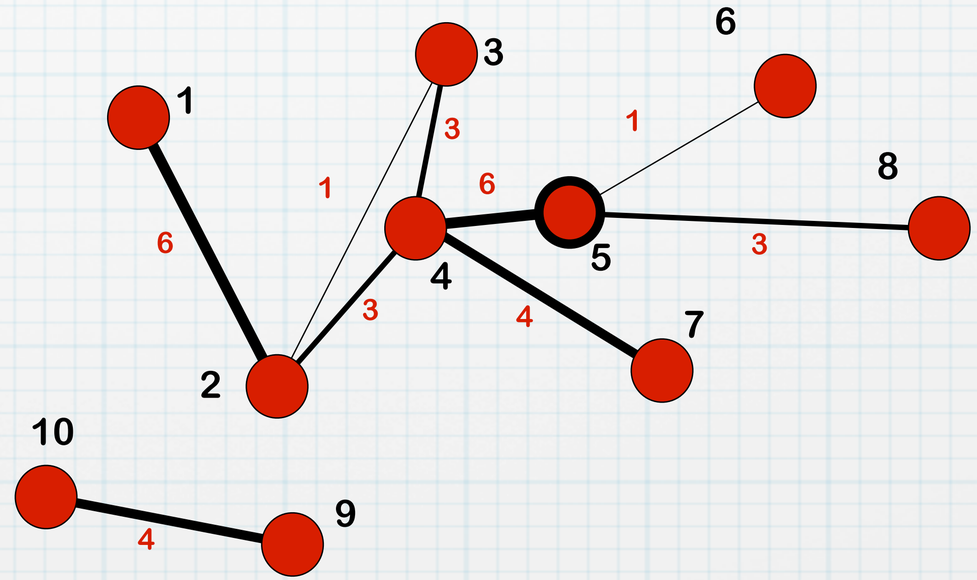


More local measures

In weighted networks, degree generalizes to strength (a.k.a. weighted degree):

$$s_i = w_{i1} + w_{i2} + \ldots + w_{iN} = \sum_j w_{ij}$$

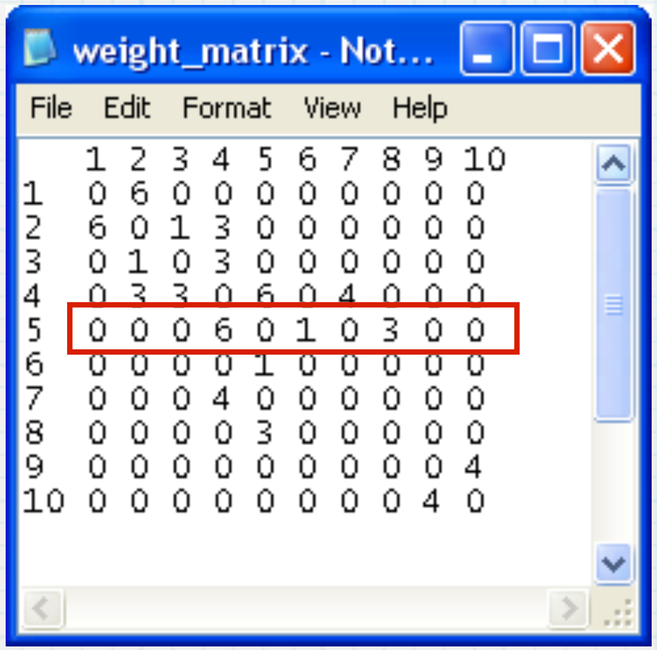


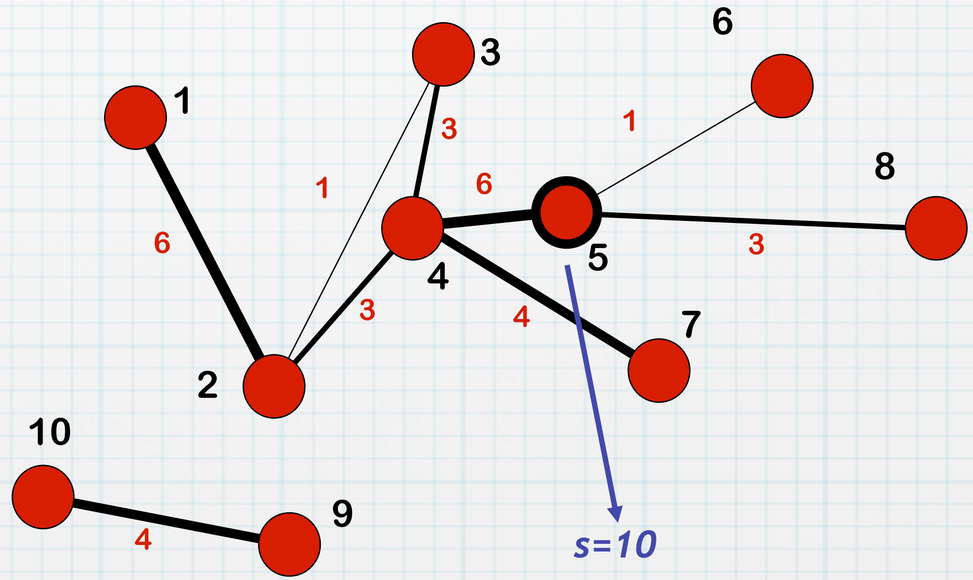


More local measures

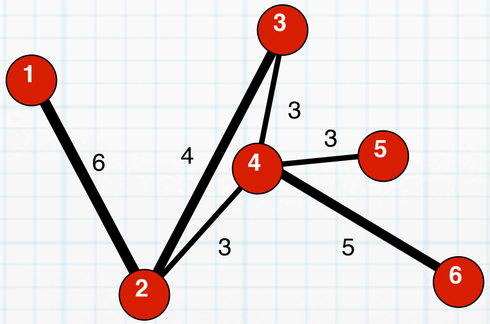
In weighted networks, degree generalizes to strength (a.k.a. weighted degree):

$$s_i = w_{i1} + w_{i2} + \ldots + w_{iN} = \sum_j w_{ij}$$



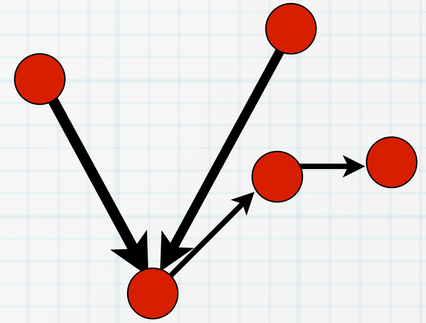


Python and NetworkX




```
W.degree(4) # degree
W.degree(4, weight='weight') # strength
```

More local measures



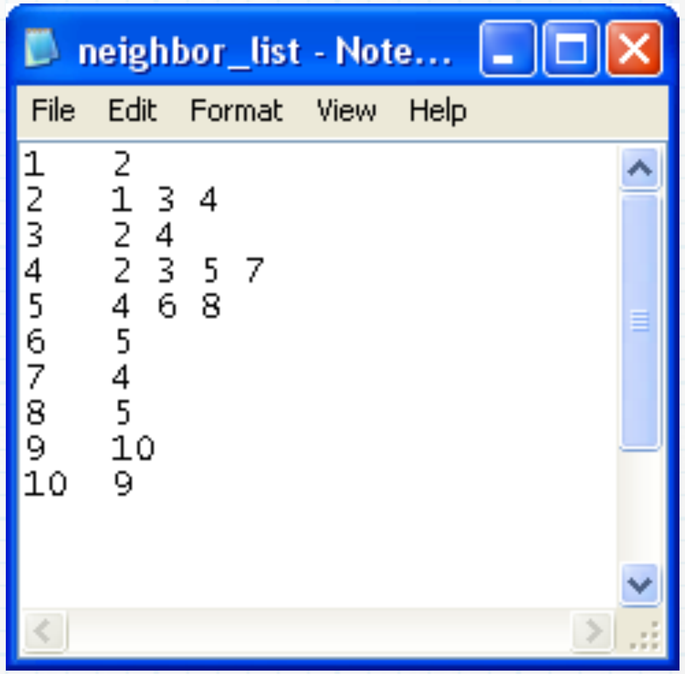
What about a directed, weighted network?

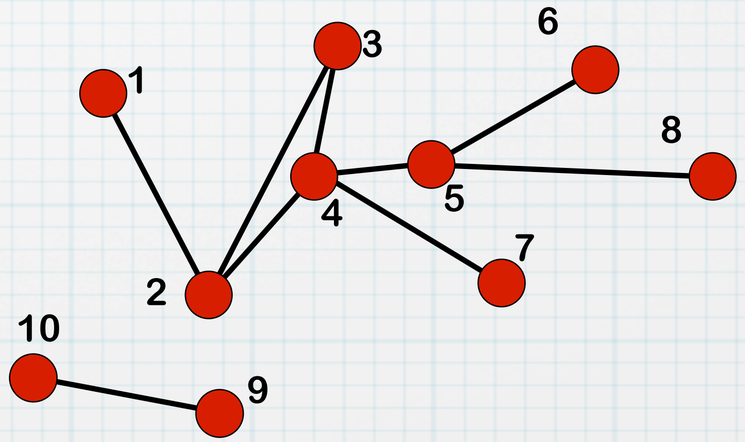
- In-strengh (weighted in-degree)
- Out-strength (weighted out-degree)

Exercise: create a small, weighted, directed network to represent "like" relationships among a group of friends. Print in- and out-strength of each node. (Hint: a DiGraph can have weight attributes, too.)

Mathematical representation

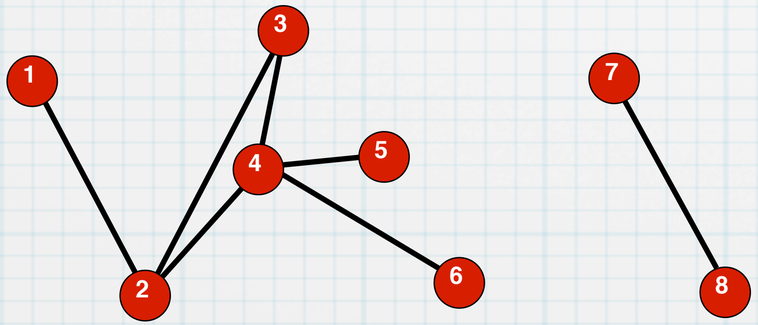
- Neighbor list
  - i $n_1(i), n_2(i), n_3(i), \ldots$
  - useful for sparse graphs
  - for every vertex: list of neighbors, i.e. k elements instead of N
- a.k.a. adjacency list





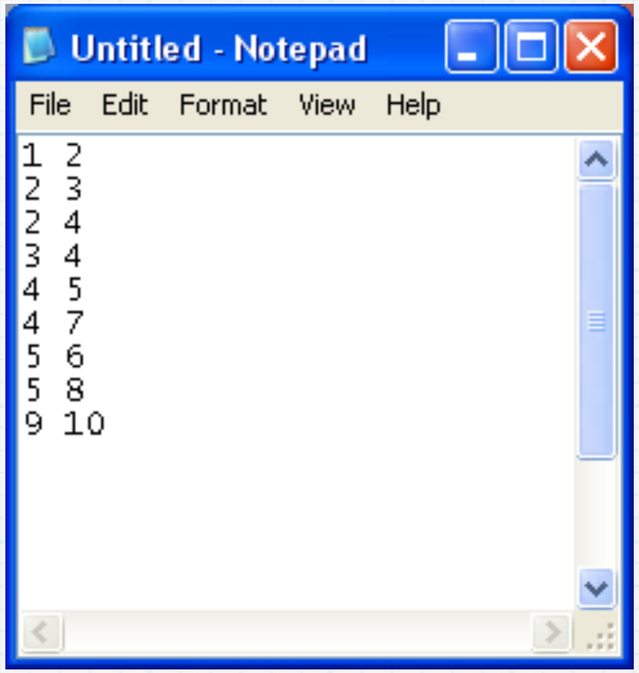
Python and NetworkX

```
G.neighbors(2)

for n,nbrs in G.adjacency():
    for nbr,eattr in nbrs.items():
        print('(%d, %d)' % (n,nbr))

nx.write_adjlist(G, "netfile.adjlist")
G2 = nx.read_adjlist("netfile.adjlist")
GM = nx.isomorphism.GraphMatcher(G,G2)
GM.is_isomorphic()
```




```
with open('netfile.adjlist') as f:
    for line in f:
        print(line)
```

Mathematical representation

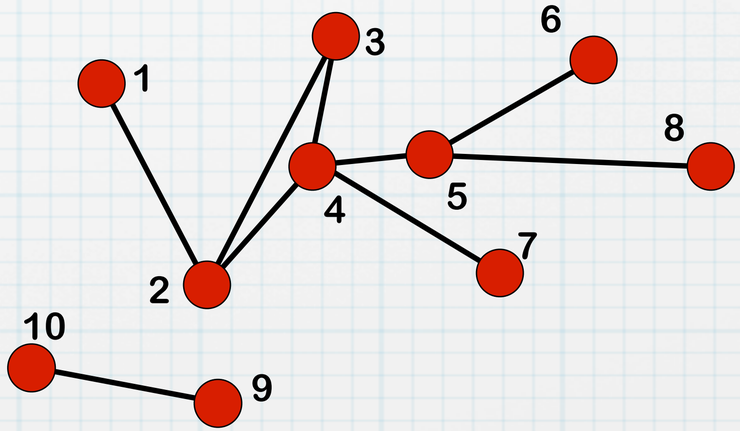


* Edge list

  i_1 j_1
  i_2 j_2
  i_3 j_3

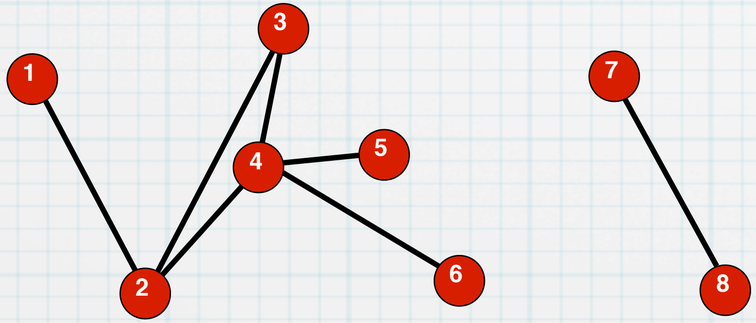
  ... ...

* list of E edges (each once)



Python and NetworkX

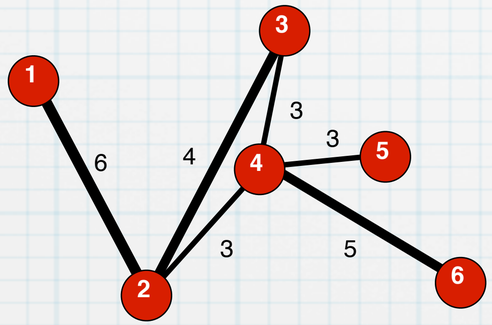
```
for i,j in G.edges:
    print('%d %d' %(i,j))

nx.write_edgelist(G, "netfile.edgelist")
G3 = nx.read_edgelist("netfile.edgelist")
GM = nx.isomorphism.GraphMatcher(G,G3)
GM.is_isomorphic()
```




```
with open('netfile.edgelist') as f:
    for line in f:
        print(line)
```

Python and NetworkX

```
nx.write_weighted_edgelist(W, "netfile.edgelist")
W2 = nx.read_weighted_edgelist("netfile.edgelist")
GM = nx.isomorphism.GraphMatcher(W,W2)
GM.is_isomorphic()
```




```
with open('netfile.edgelist') as f:
    for line in f:
        print(line)
```

Drawing
(with matplotlib)

```
In [96]: nx.draw(G)
```