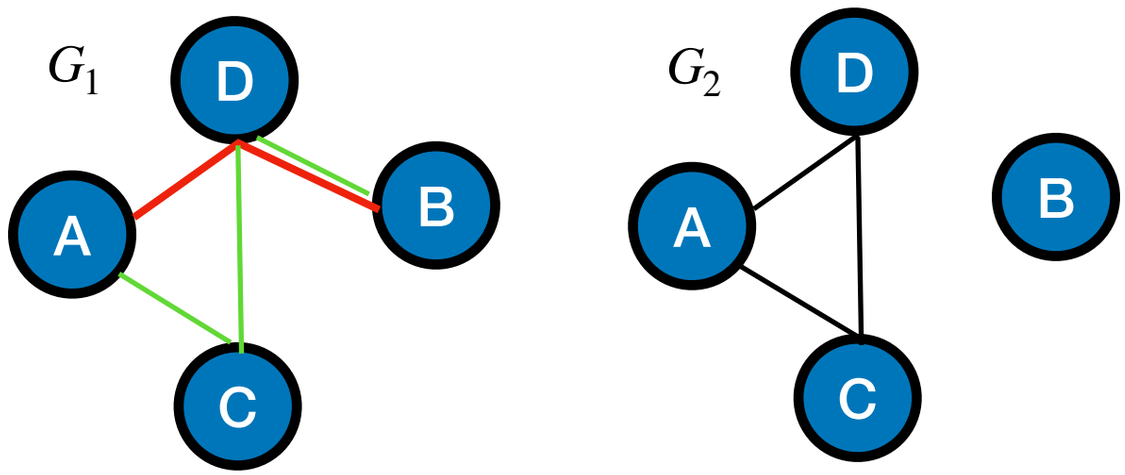
PATHS AND DISTANCE

Paths

- **Existence**: A path between nodes A and B exists if it is possible to go from node-A to node-B by traversing the links in the network



- **singleton**: a node, with no neighbors (i.e. degree=zero)

- The path is the **sequence** of links traversed. $P_1 = (e_{AD}, e_{DB})$ $P_2 = (e_{AC}, e_{CD}, e_{DB})$

  ** Typically there are multiple paths between the same two nodes.

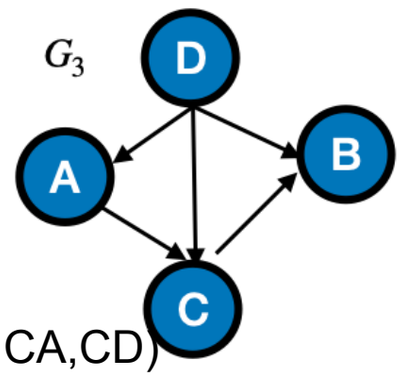
- **Path length:** The number of links in a path. $l_{AB}^1 = 2$ $l_{AB}^2 = 3$

- **Simple path:** Path that never goes through the same link more than once;

- **Distance** l_{AB} between two nodes is defined as the minimum number of links that must be traversed in a connecting path between them. Such a path is called the **shortest path**, and its length is called the **shortest-path length.**

  ** You can have multiple shortest paths between two nodes, but they all must have the same (shortest) length.

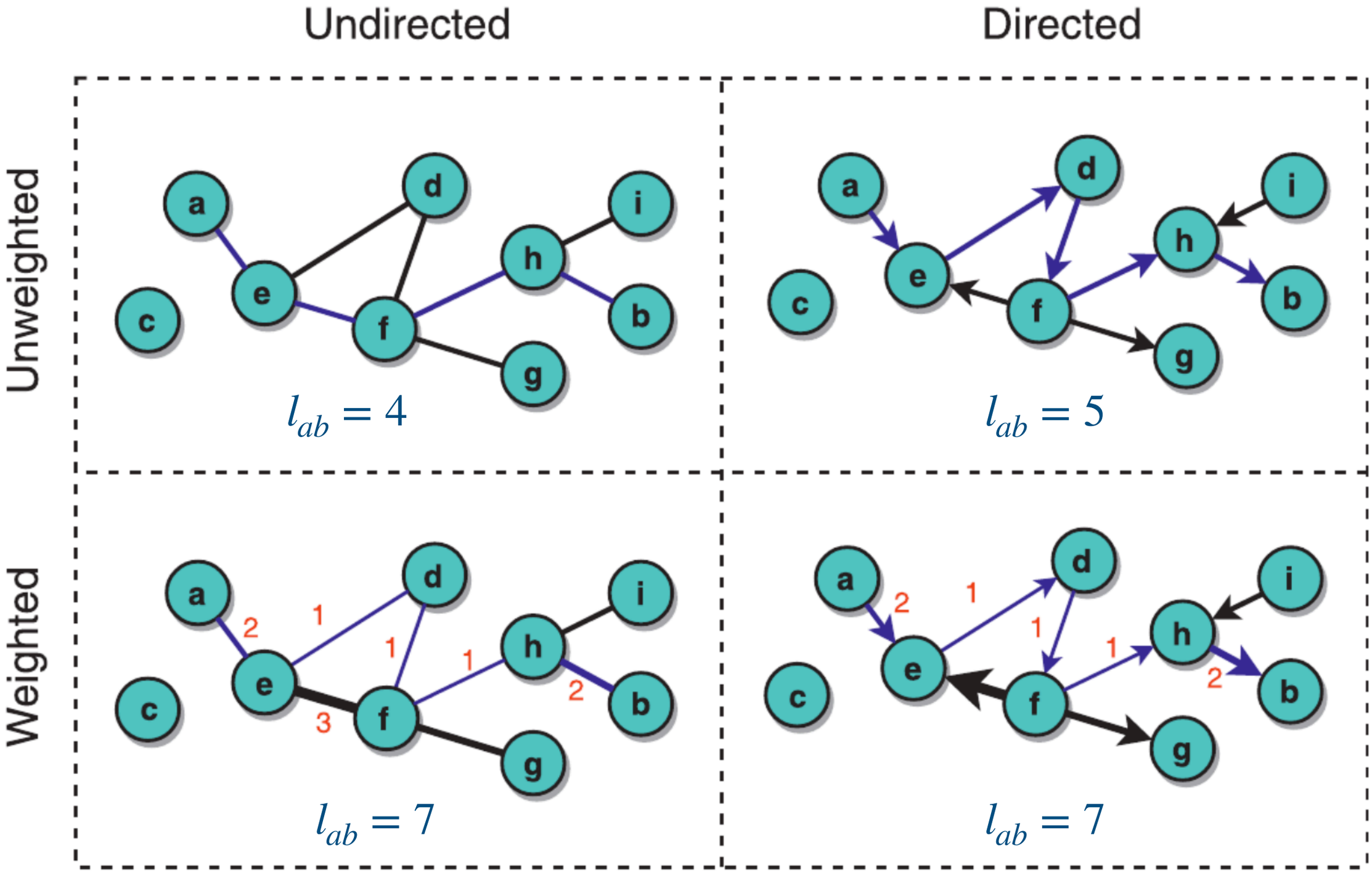
- **Directed networks**, paths must OBEY directed links
  - i.e. (paths can't go down one-way streets)



  ** Beware that in a directed network there may be no paths between some pairs of nodes. (e.g. CA,CD)

Examples

Sometimes, we can **associate a numeric distance with links** between adjacent nodes (e.g. geographic distance). In such cases we can redefine the path length as the sum of these distances
You can think of an unweighted network as a special case in which all links have distance one.

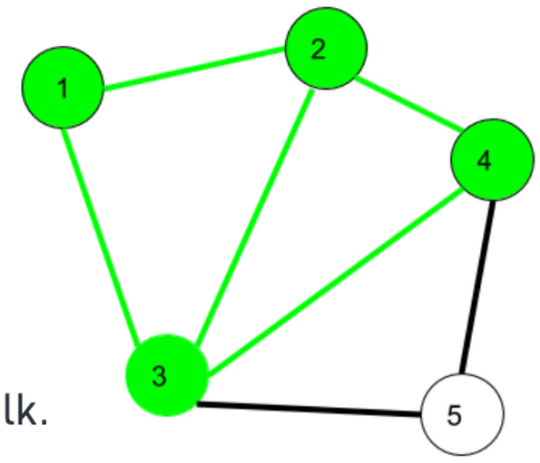


Shortest paths (blue) in undirected, directed, unweighted, and weighted networks. Link weights represent distances and are shown in red.

Graph theory terminology-1

1. Walk: A sequence of vertices and edges of a graph i.e. if we traverse a graph then we get a walk.

Note: Vertices and Edges can be repeated.

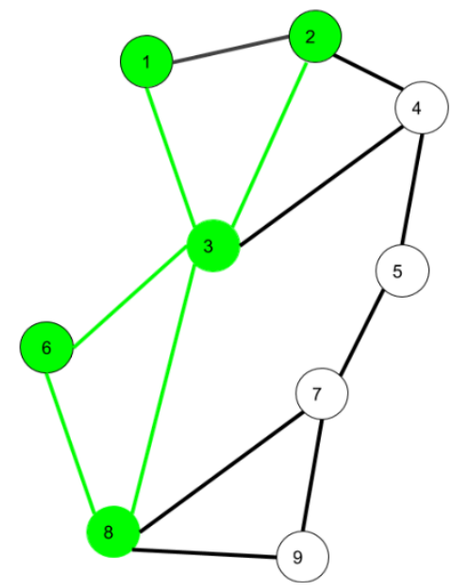


Open walk- A walk is said to be an open walk if the starting and ending vertices are different i.e. the origin vertex and terminal vertex are different. 1->2->3->4->5->3 is an open walk.

Closed walk- A walk is said to be a closed walk if the starting and ending vertices are identical i.e. if a walk starts and ends at the same vertex, then it is said to be a closed walk.

1->2->3->4->5->3->1 is a closed walk.

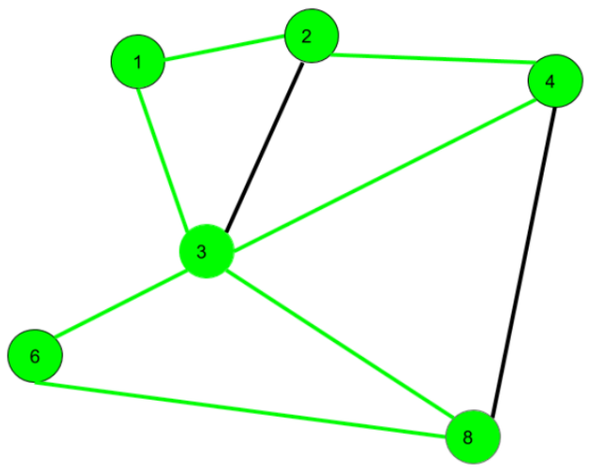
2. Trail: An open walk in which no edge is repeated (vertex can be repeated)



Here 1->3->8->6->3->2 is trail

Also 1->3->8->6->3->2->1 will be a closed trail

3. Circuit: Traversing a graph such that no edge is repeated but vertex can be repeated and it is closed also i.e. it is a closed trail.



Vertex can be repeated.

Edge can not be repeated.

```
Circuit is a closed trail.
These can have repeated vertices only.
```

1->2->4->3->6->8->3->1 is a circuit.

https://www.geeksforgeeks.org/mathematics-walks-trails-paths-cycles-and-circuits-in-graph/

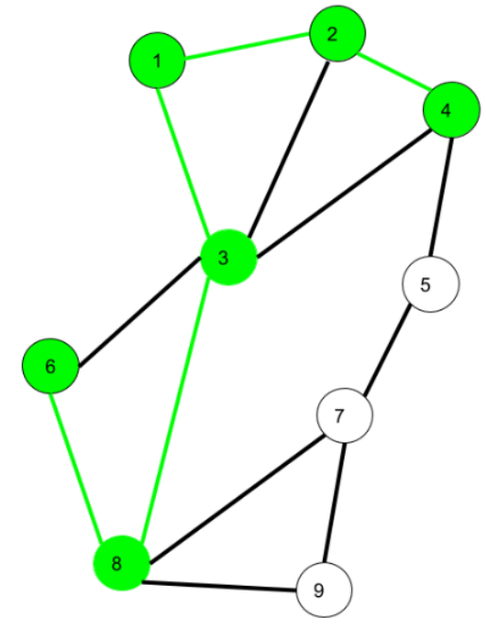
Graph theory terminology-2

4. Path –

It is a trail in which neither vertices nor edges are repeated i.e. if we traverse a graph such that we do not repeat a vertex and nor we repeat an edge. As path is also a trail, thus it is also an open walk.

Vertex not repeated

Edge not repeated

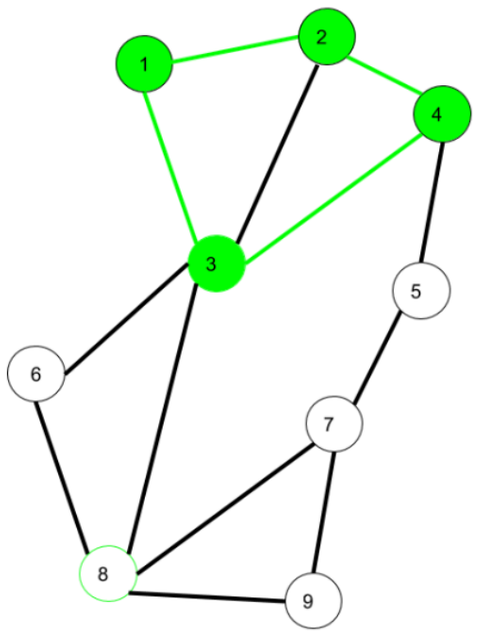


6->8->3->1->2->4 is a Path

5. Cycle – special path that can be traversed to go from one node back to itself.

Traversing a graph such that we do not repeat a vertex nor we repeat a edge but the starting and ending vertex must be same i.e. we can repeat starting and ending vertex only then we get a cycle.

Vertex not repeated



1->2->4->3->1 is a cycle

Edge not repeated

```
Cycle is a closed path.
These can not have repeat anything (neither edges nor vertices).
```

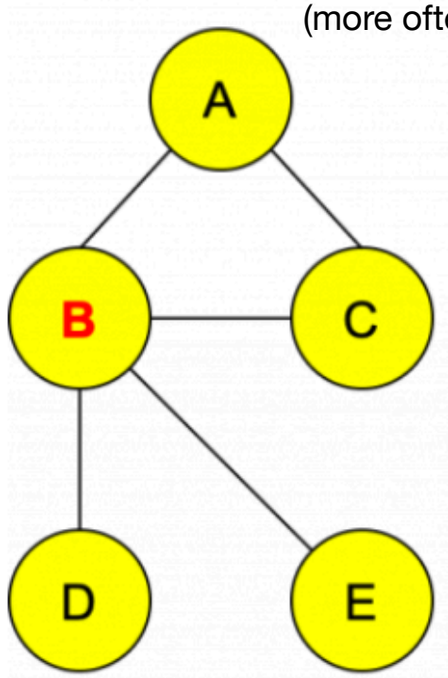
Summary:	Repeat Vertex	Repeat Edge	Closed	Open
Walk	✓	✓	✓	✓
Trail	✓	✗	✗	✓
Path	✗	✗	✗	✓
Cycle	✗	✗	✓	✗
Circuit	✓	✗	✓	✗

https://www.geeksforgeeks.org/mathematics-walks-trails-paths-cycles-and-circuits-in-graph/

Distance matrix

Matrix containing the **distances (i.e. length of shortest paths)** between all Node in the Network

Un-directed Graph:



(more often distances are represented with a dictionary (only consider paths that exist)

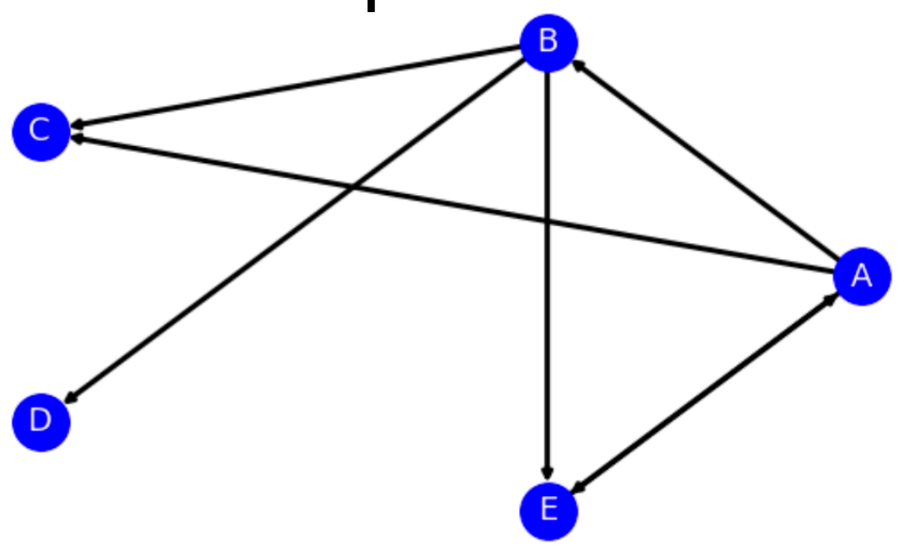
	A	B	C	D	E
A	0	1	1	2	2
B	1	0	1	1	1
C	1	1	0	2	2
D	2	1	2	0	2
E	2	1	2	2	0

(symetric)

$= l_{ij}$

zero represents the absence of a path, in this case technically $l_{ij} = \infty$

Directed Graph:




```
[[0. 1. 1. 2. 1.]
 [2. 0. 1. 1. 1.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [1. 2. 2. 3. 0.]]
```

$= l_{ij}$

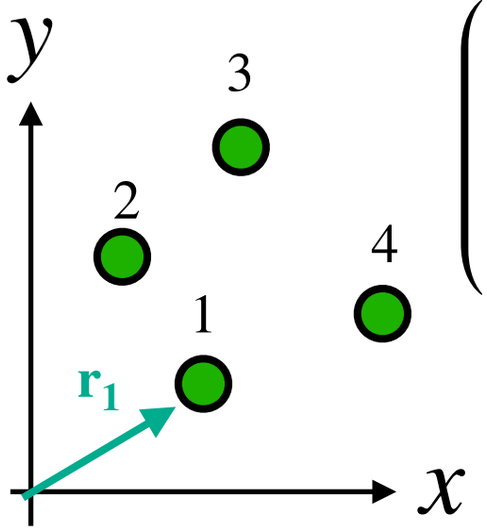
(asymetric)

Average path length (Network property)**:** The average shortest-path length across all pairs of nodes.

$$\langle l \rangle = \underset{l_{ij} \neq 0}{\text{mean}}(l_{ij})$$

```
np.mean(lij[lij!=0]    print(nx.average_shortest_path_length(G))
```

Cartesian space

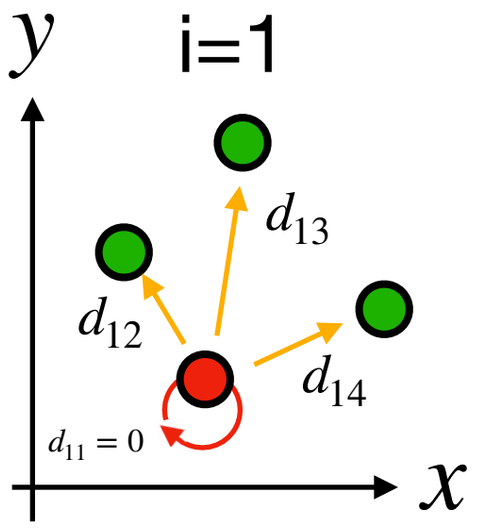


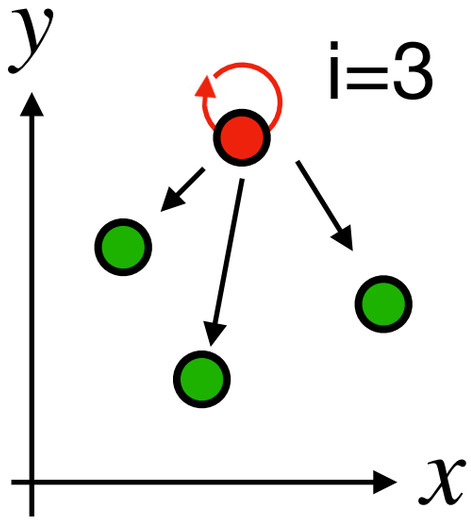
$$\begin{pmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \\ \mathbf{r}_4 \end{pmatrix} = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{pmatrix}$$

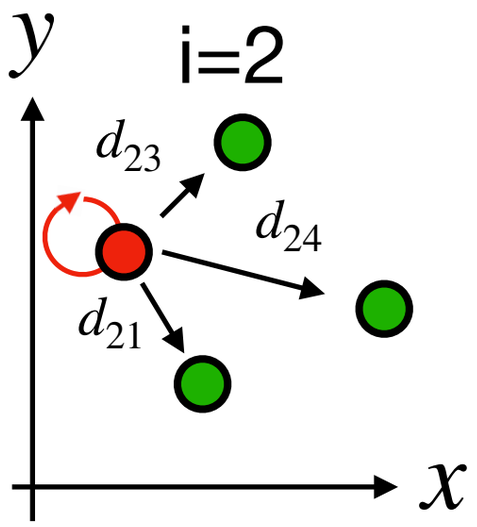
$N = 4$ **data point**

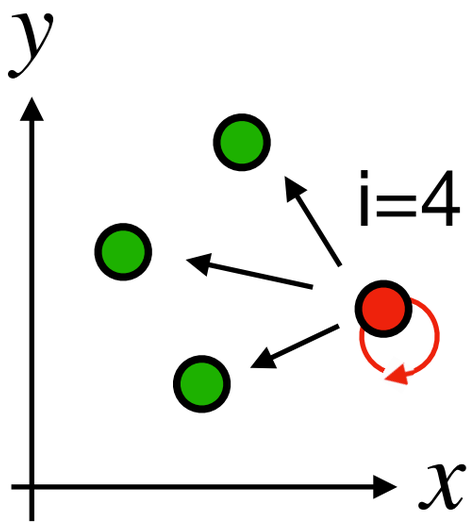
L_2 **norm: Euclidean distance**

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$









$N^2 = 16$ **pairwise distances**

$$\mathbf{d}_{ij} = \begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \end{pmatrix}$$

zero main diagonal

$\mathbf{d}_{ii} = 0$

symetric

$\mathbf{d}_{ij} = \mathbf{d}_{ji}$

$$\frac{N^2}{2} - N = 6$$ **unique pairwise distances**

COLLECTIONS OF POINTS CAN BE THOUGHT OF AS FULLY CONNECTED NETWORK

Equations

Assume that the shortest-path length is defined for each pair of nodes.

average path length of an undirected, unweighted network as

$$\langle \ell \rangle = \frac{\sum_{i,j} \ell_{ij}}{\binom{N}{2}} = \frac{2 \sum_{i,j} \ell_{ij}}{N(N-1)}, \tag{2.2}$$

ℓ_{ij} is the shortest-path length between nodes i and j, and N is the number of nodes.

Directed network: average path length

$$\langle \ell \rangle = \frac{\sum_{ij} \ell_{ij}}{N(N-1)}$$

Technically, If there are any pairs without a path, then the average path length and diameter are not defined.

We can think of such missing paths as paths with infinite distance.

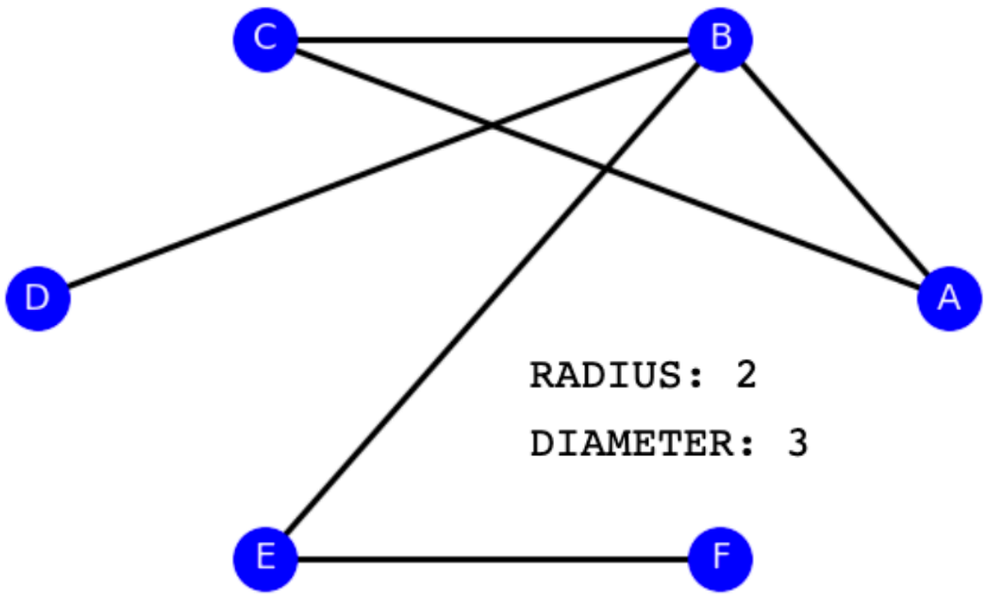
If one wishes to define the average path length in a network where some of the paths do not exist, the following formula can be used for undirected networks:

$$\langle \ell \rangle = \left(\frac{\sum_{i,j} \frac{1}{\ell_{ij}}}{\binom{N}{2}} \right)^{-1}. \tag{2.5}$$

Note that if there is no path between i and j, $\ell_{ij} = \infty$ and therefore $1/\ell_{ij} = 0$ is defined. The same trick can be used for directed networks.

Eccentricity: (Node property)

The eccentricity of a node v is the **maximum distance (maximum length of a node's shortest paths)** from v to all other nodes in G.



eccentricity vector: e_i

$$l_{ij} = \begin{matrix}[[0. & 1. & 1. & 2. & 2. & 3.] \\ [1. & 0. & 1. & 1. & 1. & 2.] \\ [1. & 1. & 0. & 2. & 2. & 3.] \\ [2. & 1. & 2. & 0. & 2. & 3.] \\ [2. & 1. & 2. & 2. & 0. & 1.] \\ [3. & 2. & 3. & 3. & 1. & 0.]]\end{matrix} \rightarrow \begin{matrix}3 \\ 2 \\ 3 \\ 3 \\ 2 \\ 3\end{matrix}$$

take maximum accross rows

```
print(nx.eccentricity(G))
```

```
{'A': 3, 'B': 2, 'C': 3, 'D': 3, 'E': 2, 'F': 3}
```

RADIUS: The radius is the minimum eccentricity. (Network property)

$$R = \min(e_i)$$ `np.min(lij.max(axis=1))`

DIAMETER: The diameter is the maximum eccentricity. (Network property)

```
print("RADIUS:",nx.radius(G))
print("DIAMETER:",nx.diameter(G))
```

$D = l_{max} = \max(l_{ij})$ same as max shortest path length

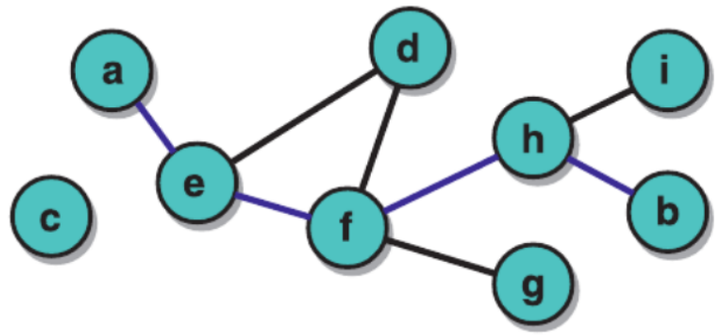
The name "diameter" is inspired by geometry, where the diameter is the longest distance between any two points on a circle.

Both average path length and diameter can be used to describe the typical distance of a network.

NetworkX:

Undirected, unweighted network

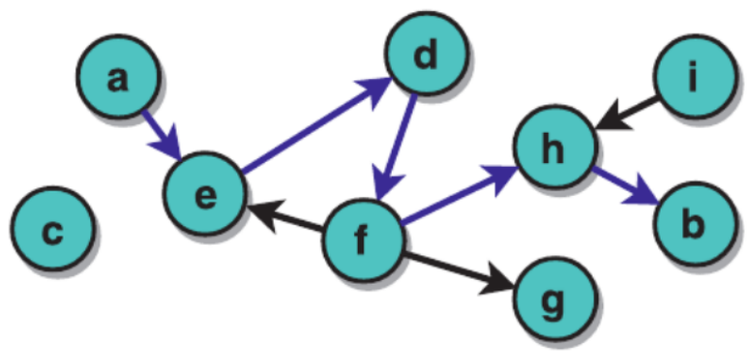
```
nx.has_path(G, 'a', 'c')              # False
nx.has_path(G, 'a', 'b')              # True
nx.shortest_path(G, 'a', 'b')         # ['a','e','f',h','b']
nx.shortest_path_length(G,'a','b') # 4
nx.shortest_path(G, 'a')              # dictionary
nx.shortest_path_length(G, 'a')       # dictionary
nx.shortest_path(G)                   # all pairs
nx.shortest_path_length(G)            # all pairs
nx.average_shortest_path_length(G) # error
G.remove_node('c')                    # make G connected
nx.average_shortest_path_length(G) # now okay
```



When only the source node is specified, we obtain a dictionary with all shortest paths, or all shortest-path lengths, from the source.

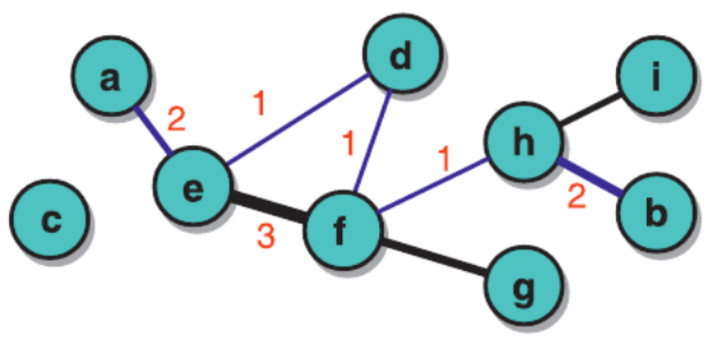
Directed, unweighted network

```
nx.has_path(D, 'b', 'a')       # False
nx.has_path(D, 'a', 'b')       # True
nx.shortest_path(D, 'a', 'b') # ['a','e','d','f',h','b']
```



Weighted networks

```
nx.shortest_path_length(W, 'a', 'b')              # 4
nx.shortest_path_length(W, 'a', 'b', 'weight') # 7
```



Finding the shortest path

Breadth-First Search

Breadth-first search takes a source node as input. To implement the algorithm, each node must have an attribute used to store its distance from the source. In addition, we must keep a queue of nodes that we call the *frontier*. A queue is a first-in-first-out data structure: nodes are extracted (dequeued) in the order in which they are inserted (queued).

Initially, the source node s is queued into the frontier. Its distance is set to $\ell(s, s) = 0$, and for all other nodes the distance is set to a common unrealistic value, say -1. The network that will eventually become the shortest-path tree is initialized with no links.

In each iteration, we visit the next node i in the frontier. The node is dequeued. Then, for each successor j of i (or each neighbor if the network is undirected), we follow three steps unless j already has its distance set:

1. Queue j into the frontier.
2. Set the distances of j from the source to $\ell_{s,j} = \ell_{s,i} + 1$.
3. Add a directed link $(i \rightarrow j)$ to the shortest-path tree.

The procedure ends when the frontier is empty. If any nodes remain with unknown distance, they are not reachable from the source; they must be in a different connected component of the network.

See section 2.5 Finding Shortest Paths for more detail