CPE 352 Data Science

3 – Network Data

Asst. Prof. Dr. Santitham Prom-on

Department of Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology Thonburi

Overview

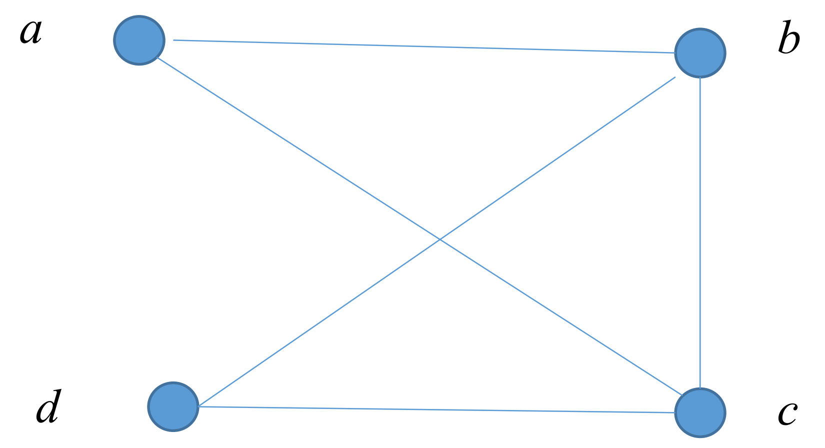
- Graph representation
- Graph analysis
- Shortest path
- Connected components

Graphs

Definition: A *graph* $G = (V, E)$ consists of a nonempty set V of *vertices* (or *nodes*) and a set E of *edges*. Each edge has either one or two vertices associated with it, called its *endpoints*. An edge is said to *connect* its endpoints.

Example:

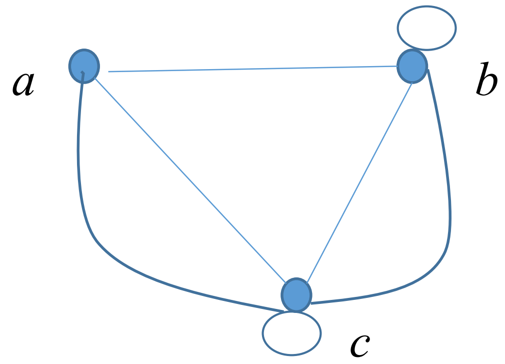
This is a graph with four vertices and five edges.



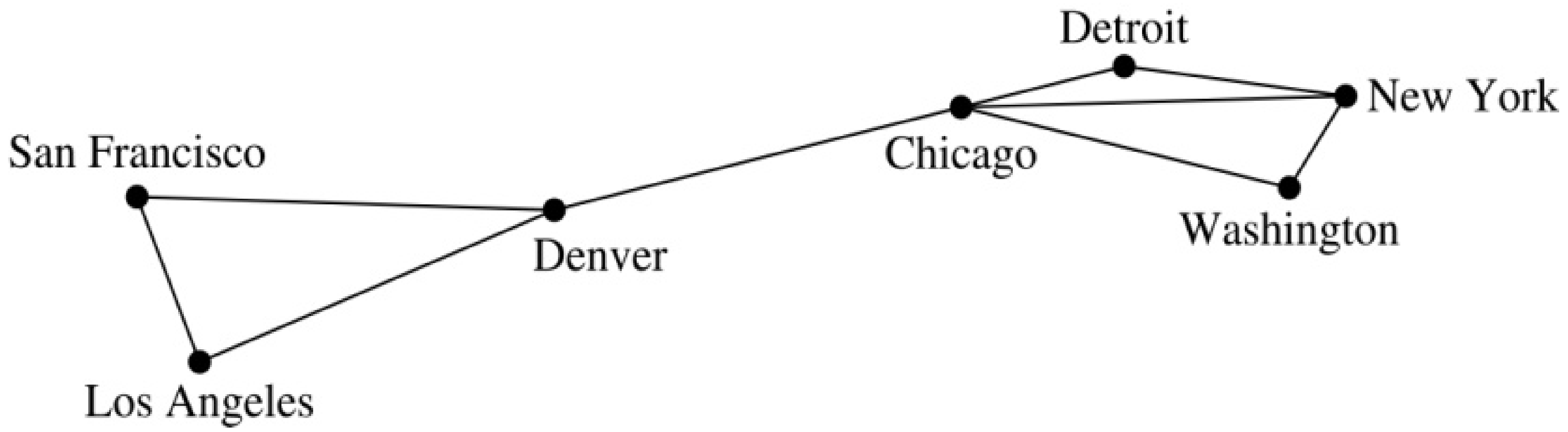
Some Terminology

- In a *simple graph* each edge connects two different vertices and no two edges connect the same pair of vertices.
- *Multigraphs* may have multiple edges connecting the same two vertices. When *m* different edges connect the vertices *u* and *v*, we say that $\{u,v\}$ is an edge of *multiplicity m*.
- An edge that connects a vertex to itself is called a *loop*.
- A *pseudograph* may include loops, as well as multiple edges connecting the same pair of vertices.

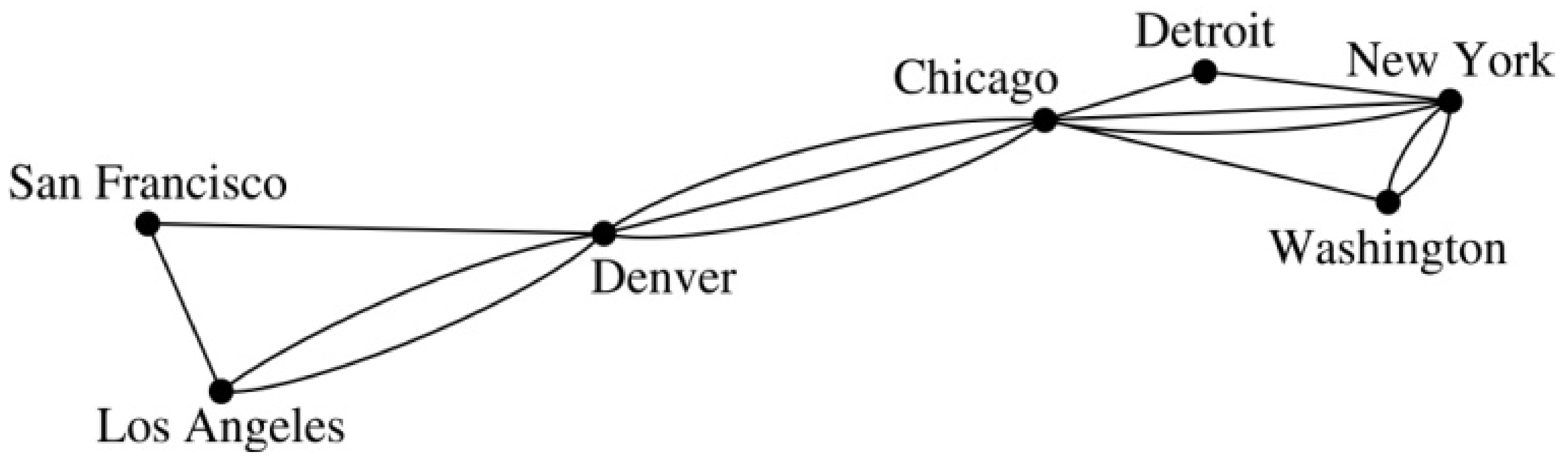
Example:
This pseudograph has both multiple edges and a loop.



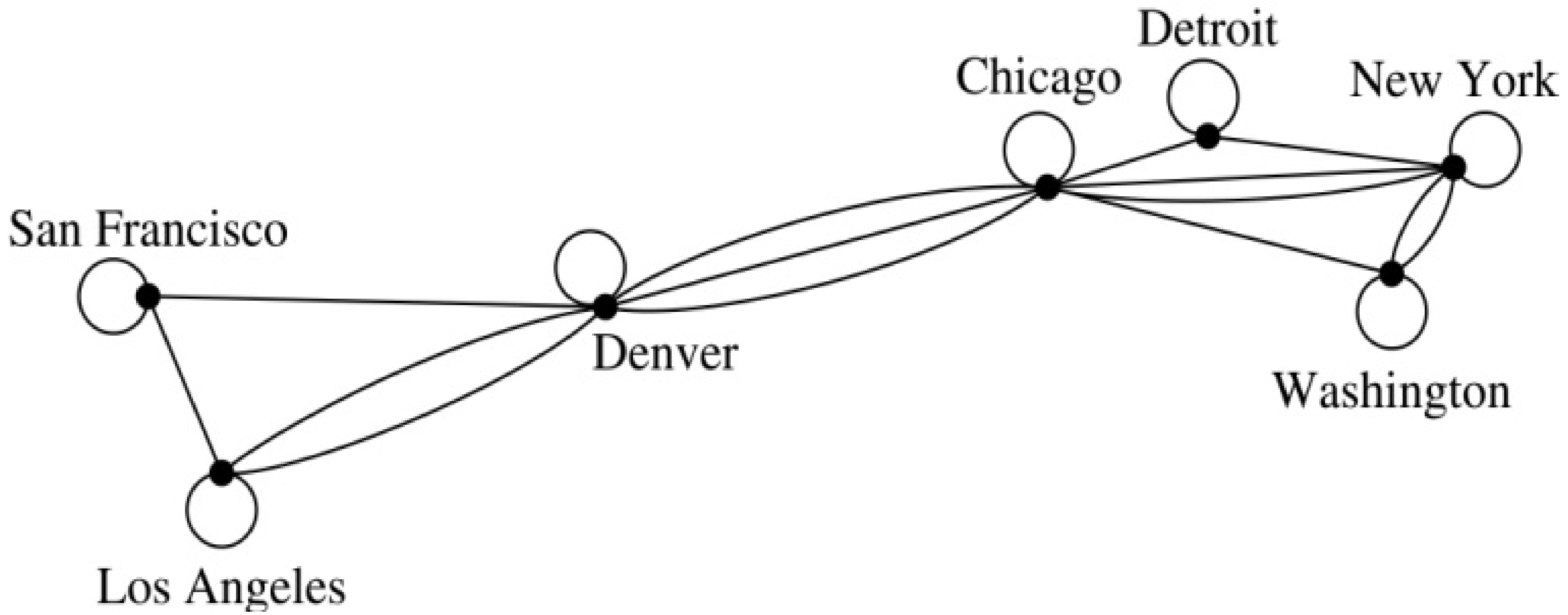
Graph Models: Computer Networks



Computer Network with Multiple Links Multigraph



Computer Network with Diagnosis Links Pseudograph



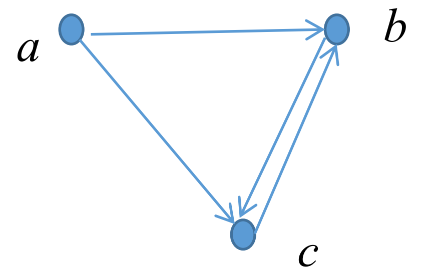
Directed Graphs

Definition: A *directed graph* (or *digraph*) $G = (V, E)$ consists of a nonempty set V of *vertices* (or *nodes*) and a set E of *directed edges* (or *arcs*). Each edge is associated with an ordered pair of vertices. The directed edge associated with the ordered pair (u,v) is said to *start at u* and *end at v*.

Some Terminology (*continued*)

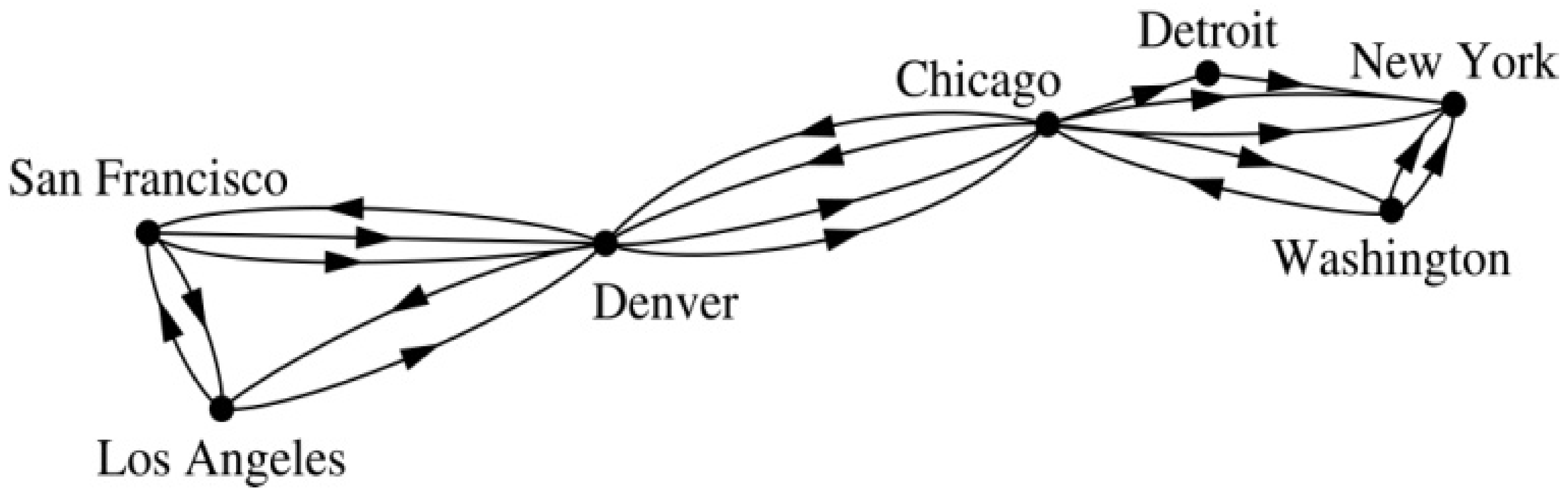
- A *simple directed graph* has no loops and no multiple edges.

  Example:
  This is a directed graph with three vertices and four edges.



- A *directed multigraph* may have multiple directed edges. When there are *m* directed edges from the vertex *u* to the vertex *v*, we say that (*u*,*v*) is an edge of *multiplicity m*.

Network with Dedicated One-way Links



Summary of Graph Terminology

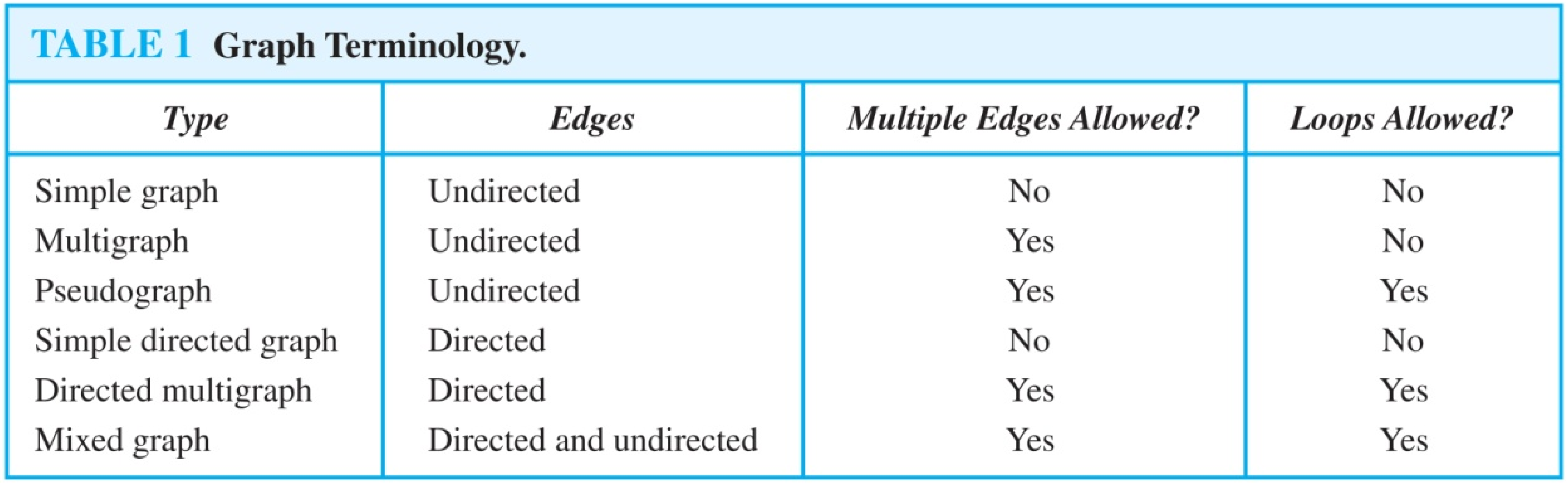


TABLE 1 **Graph Terminology.**

Type	*Edges*	*Multiple Edges Allowed?*	*Loops Allowed?*
Simple graph	Undirected	No	No
Multigraph	Undirected	Yes	No
Pseudograph	Undirected	Yes	Yes
Simple directed graph	Directed	No	No
Directed multigraph	Directed	Yes	Yes
Mixed graph	Directed and undirected	Yes	Yes

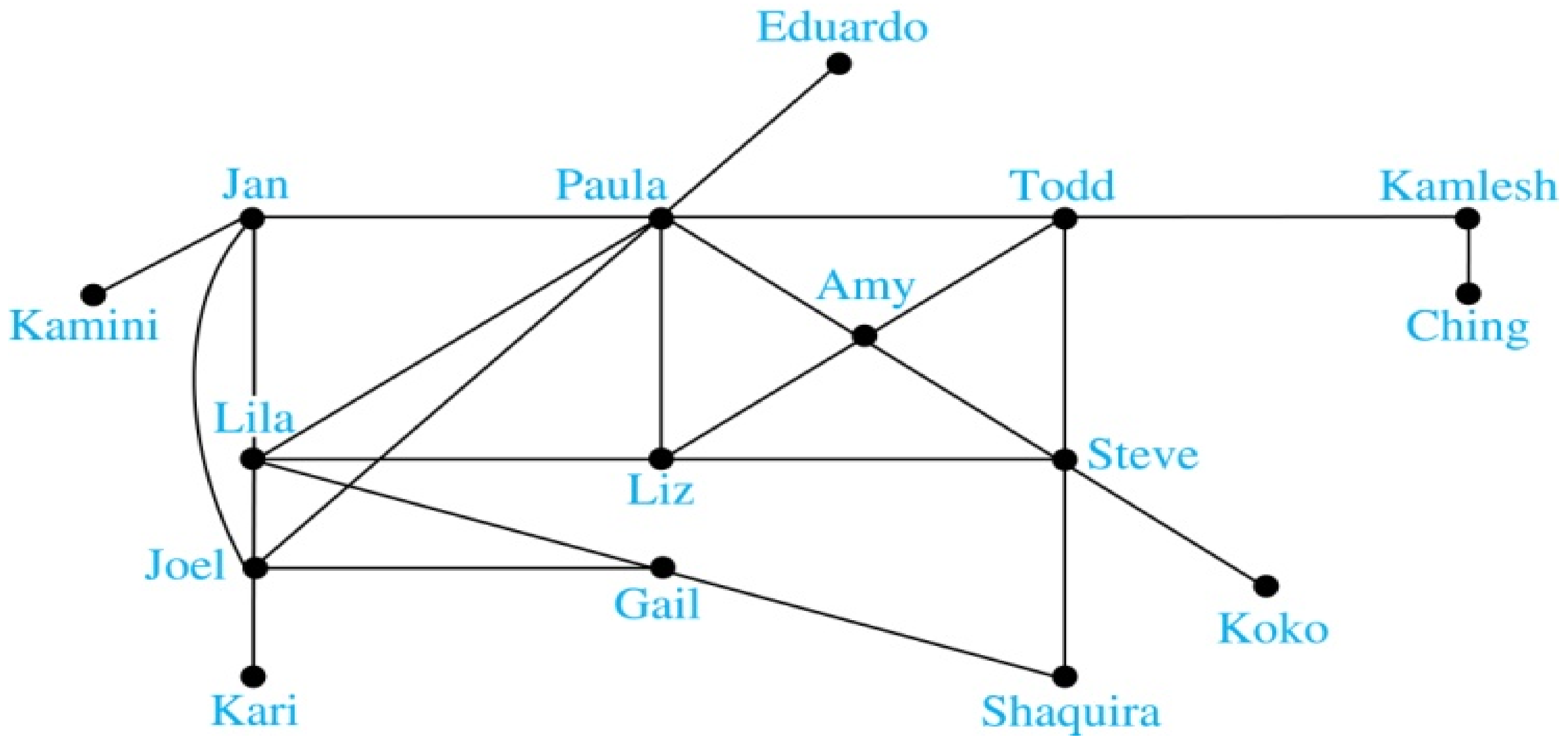
Other Applications of Graphs

- We will illustrate how graph theory can be used in models of:
  - Social networks
  - Communications networks
  - Information networks
  - Software design
  - Transportation networks
  - Biological networks

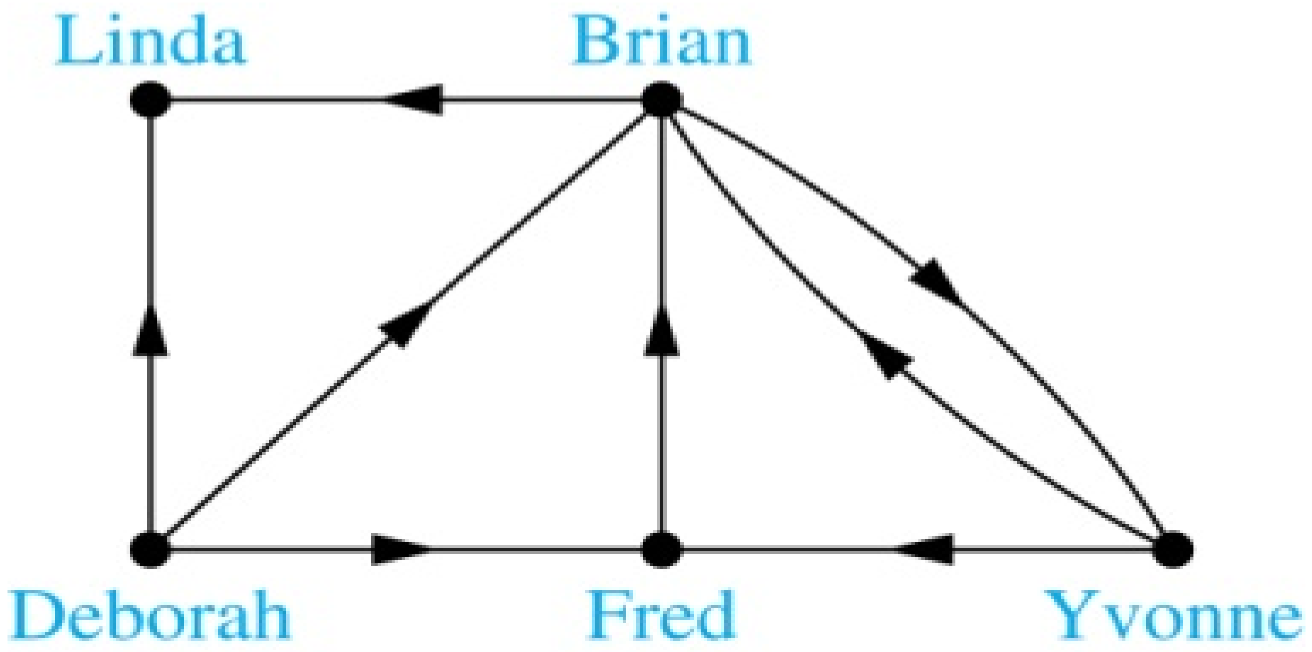
Graph Models: Social Networks

- Graphs can be used to model social structures based on different kinds of relationships between people or groups.
- In a *social network*, vertices represent individuals or organizations and edges represent relationships between them.
- Useful graph models of social networks include:
  - *friendship graphs* - undirected graphs where two people are connected if they are friends (in the real world, on Facebook, or in a particular virtual world, and so on.)
  - *collaboration graphs* - undirected graphs where two people are connected if they collaborate in a specific way
  - *influence graphs* - directed graphs where there is an edge from one person to another if the first person can influence the second person

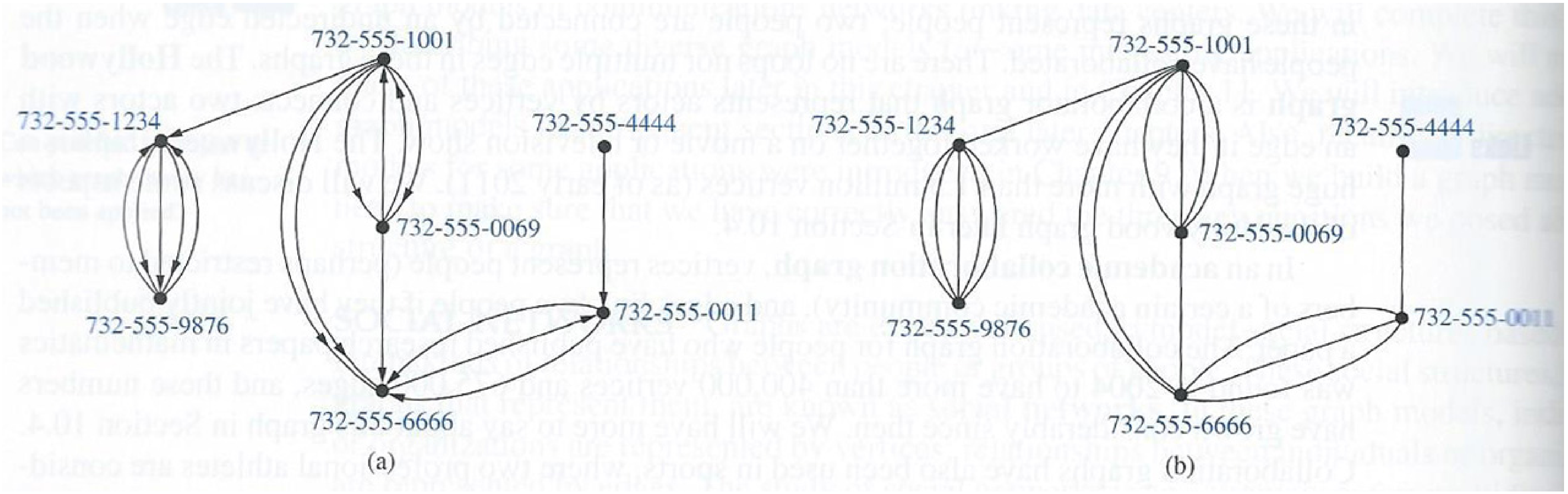
Facebook Friendship Graph



Influence Graph

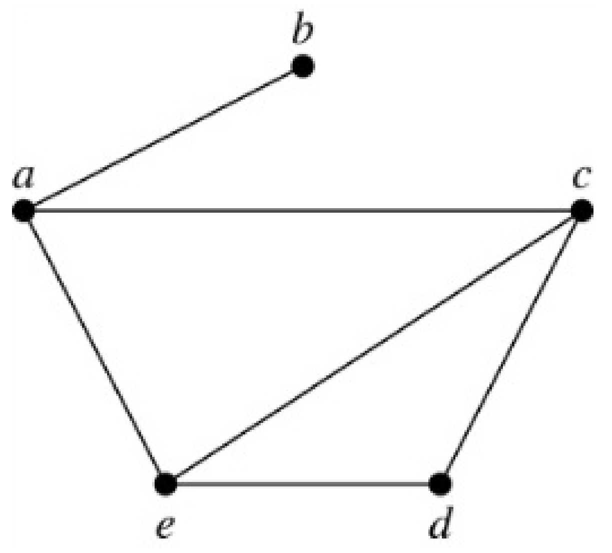


Communication Network Call Graph



Graph Representation

Representing Graphs
Adjacency Lists 1 – Undirected Graph



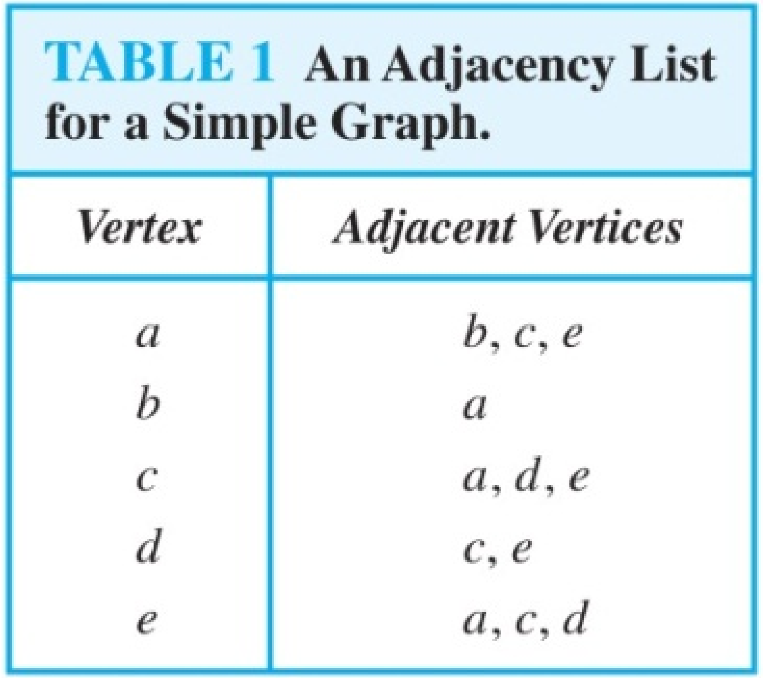
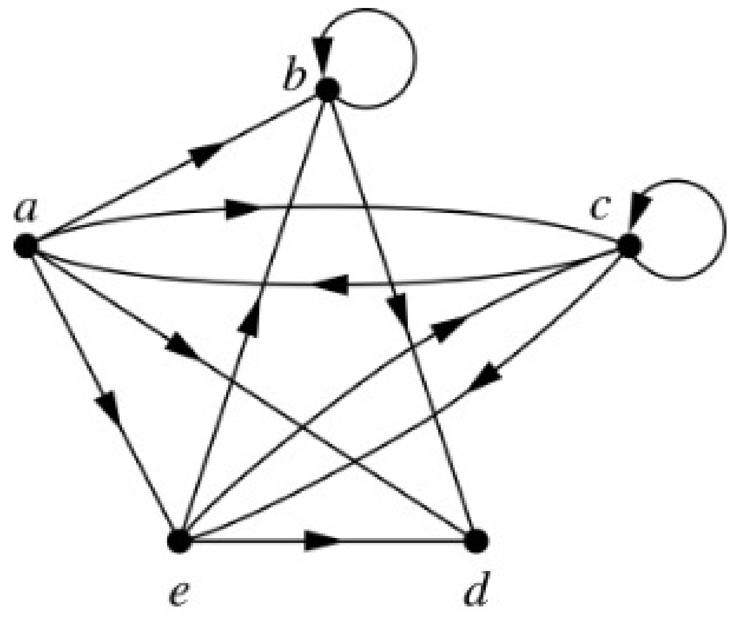


TABLE 1 An Adjacency List for a Simple Graph.

Vertex	*Adjacent Vertices*
a	*b*, *c*, *e*
b	*a*
c	*a*, *d*, *e*
d	*c*, *e*
e	*a*, *c*, *d*

Representing Graphs
Adjacency Lists 2 – Directed Graph



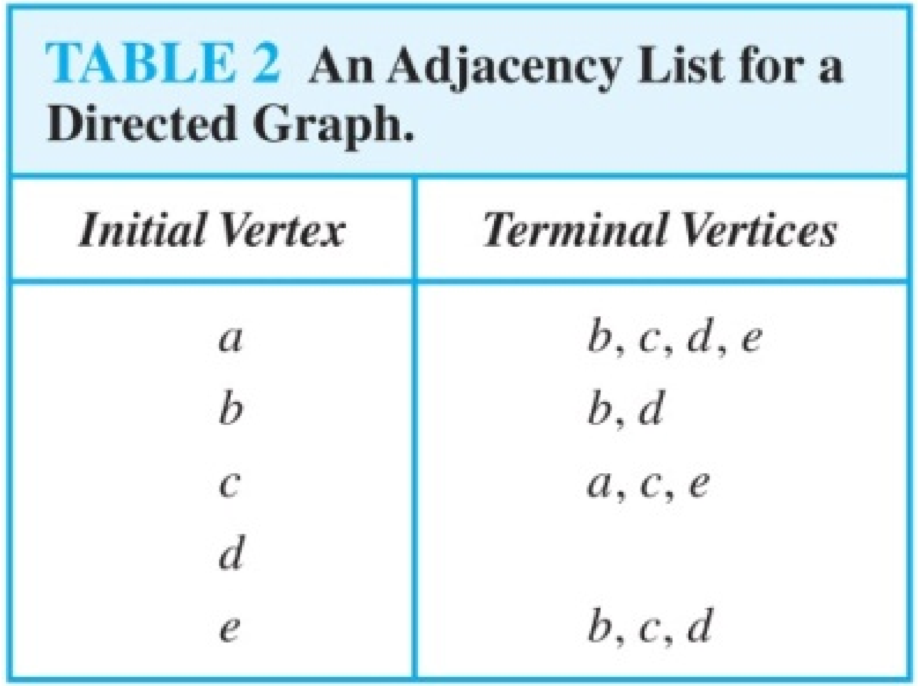


TABLE 2 **An Adjacency List for a Directed Graph.**

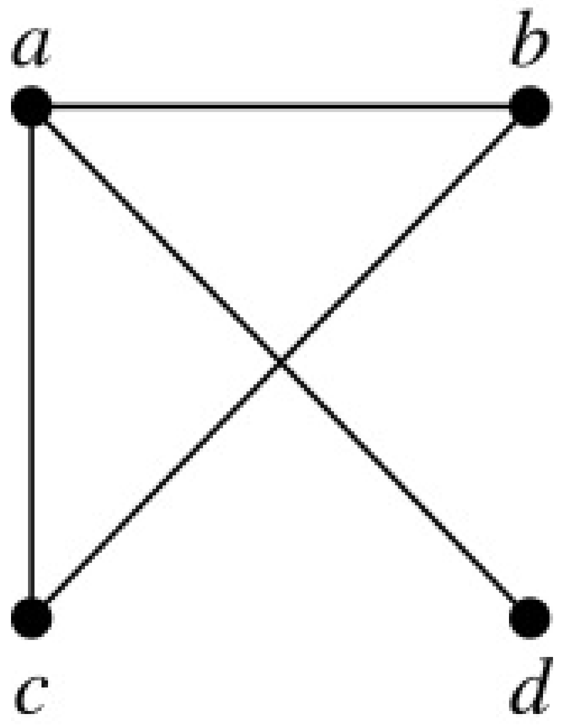
Initial Vertex	*Terminal Vertices*
a	*b*, *c*, *d*, *e*
b	*b*, *d*
c	*a*, *c*, *e*
d	
e	*b*, *c*, *d*

Representing Graphs Adjacency Matrices

Definition: Suppose that $G = (V, E)$ is a simple graph where $|V| = n$. Arbitrarily list the vertices of G as $v_1, v_2, \ldots, v_n$. The *adjacency matrix* $\mathbf{A}_G$ of G, with respect to the listing of vertices, is the $n \times n$ zero-one matrix with 1 as its (i, j)th entry when v_i and v_j are adjacent, and 0 as its (i, j)th entry when they are not adjacent.

$$\mathbf{A}_G = \left[a_{ij} \right] \text{ where } a_{ij} = \begin{cases} 1, & \text{if } \{v_i, v_j\} \text{ is an edge of G} \\ 0, & \text{otherwise} \end{cases}$$

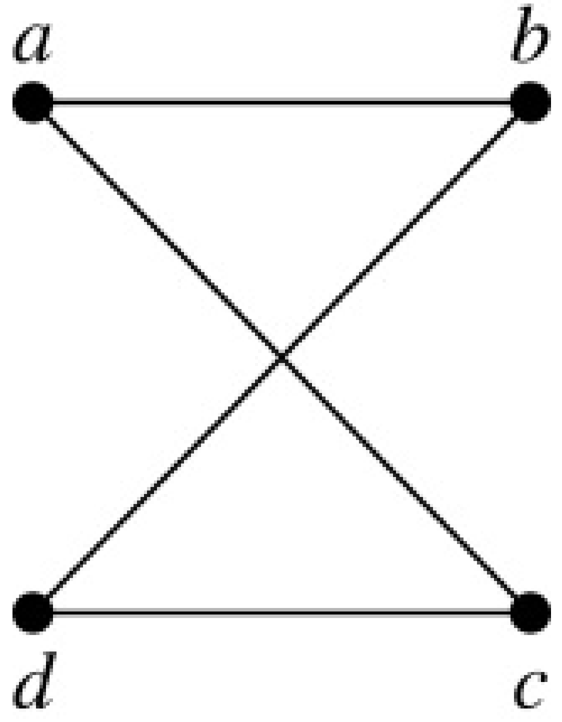
Adjacency Matrices Example 1



The ordering of vertices is a, b, c, d.

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Adjacency Matrices Example 2

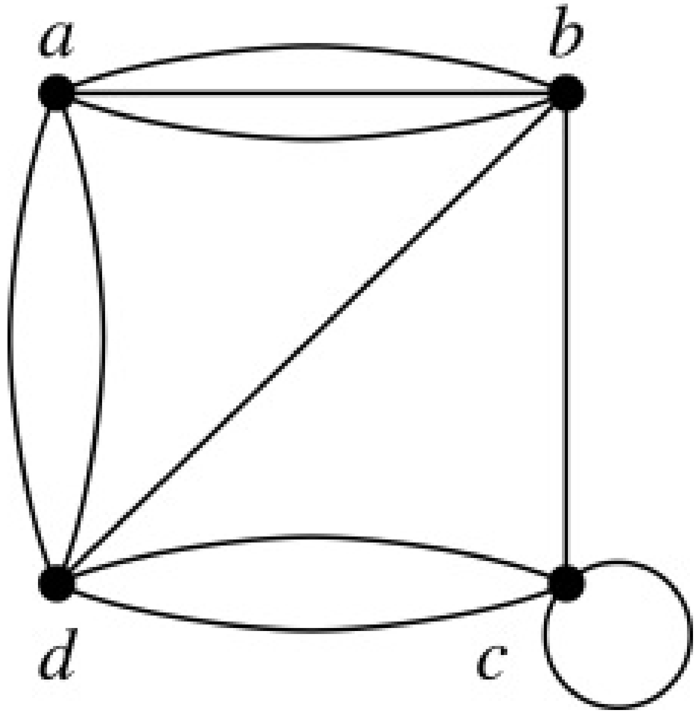


The ordering of vertices is a, b, c, d.

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Adjacency Matrices Multiple Edges/Loops

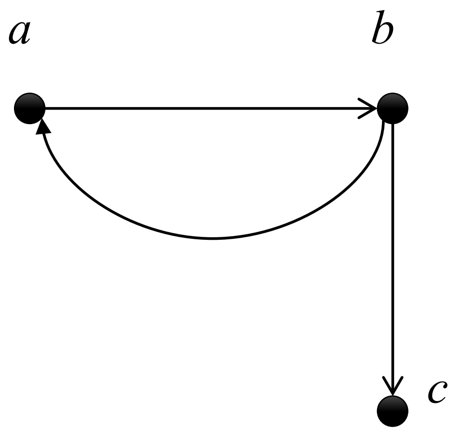
When multiple edges connect the same pair of vertices v_i and v_j, (or if multiple loops are present at the same vertex), the (i, j)th entry equals the number of edges connecting the pair of vertices.



$$\begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 2 & 0 \end{bmatrix}$$

Adjacency Matrices Directed Graph

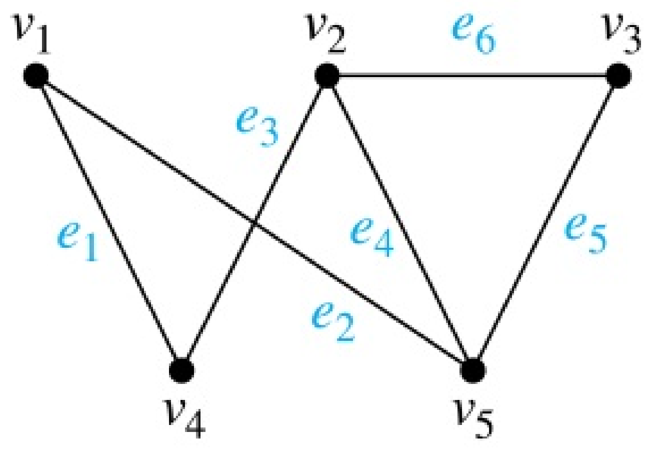
The matrix for a directed graph $G = (V, E)$ has a 1 in its (i, j)th position if there is an edge from v_i to v_j, where $v_1, v_2, \ldots v_n$ is a list of the vertices.



$$\begin{array}{cc} & a \rightarrow b \\ b \rightarrow a & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad b \rightarrow c \end{array}$$

Representing Graphs Incidence Matrices

$$\mathbf{M} = [m_{ij}], \text{ where } m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } v_i \\ 0 & \text{otherwise} \end{cases}$$



$$\begin{array}{c|cccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ \hline v_1 & 1 & 1 & 0 & 0 & 0 & 0 \\ v_2 & 0 & 0 & 1 & 1 & 0 & 1 \\ v_3 & 0 & 0 & 0 & 0 & 1 & 1 \\ v_4 & 1 & 0 & 1 & 0 & 0 & 0 \\ v_5 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

Matrix representation of graph

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
```

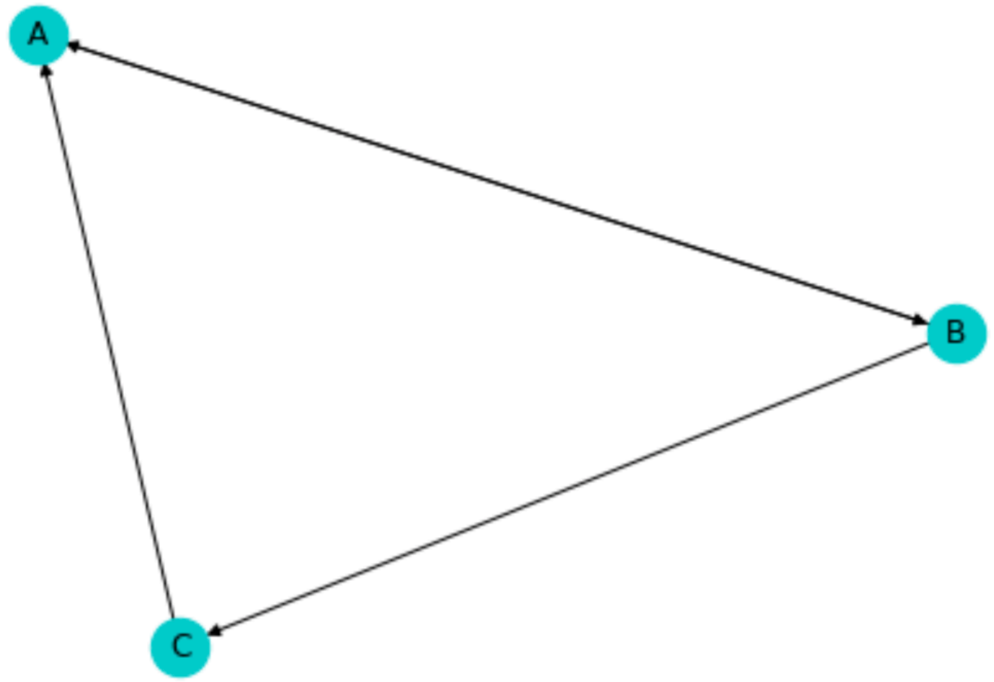
```
A = np.array([[0,1,0],
              [1,0,1],
              [1,0,0]])
print(A)
```

```
[[0 1 0]
 [1 0 1]
 [1 0 0]]
```

Create networkx graph object

```
G = nx.from_numpy_array(A, create_using=nx.MultiDiGraph)
G = nx.relabel_nodes(G, {0:'A',1:'B',2:'C'})
```

```
nx.draw(G, with_labels=True, node_size=500, node_color='#00CCCC')
```



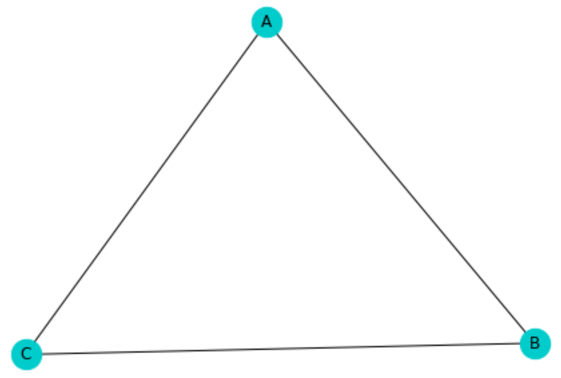
From Pandas adjacency

```
import pandas as pd
A_df = pd.DataFrame(A,
                    columns=['A','B','C'],
                    index=['A','B','C'])
A_df
```

	A	B	C
A	0	1	0
B	1	0	1
C	1	0	0

```
G_df = nx.from_pandas_adjacency(A_df)
```

```
nx.draw(G_df, with_labels=True, node_size=500, node_color='#00CCCC')
```



Traversal as a matrix operation

```
A2 = np.matmul(A,A)
print(A2)
```

```
[[1 0 1]
 [1 1 0]
 [0 1 0]]
```

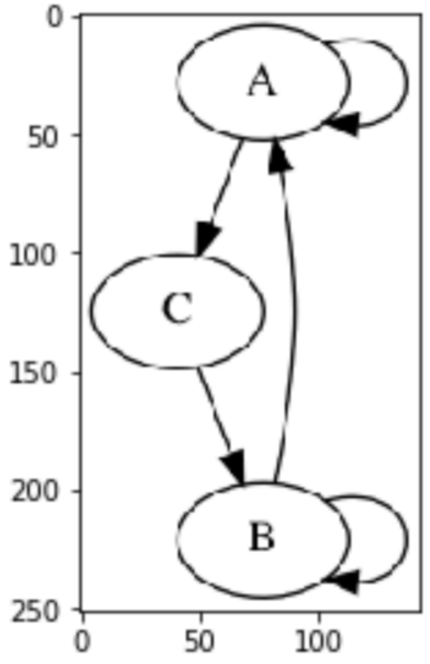
```
G2 = nx.from_numpy_array(A2, create_using=nx.MultiDiGraph)
G2 = nx.relabel_nodes(G2, {0:'A',1:'B',2:'C'})
```

```
# Need to install pydot
G2_v = nx.nx_pydot.to_pydot(G2)
```

```
G2_v.write_png('test.png')
```

```
plt.imshow(plt.imread('test.png'))
```

```
<matplotlib.image.AxesImage at 0xa1e71b9e8>
```



This gives the number of path length of 2.

(Walking 2 steps in graph)

Edge List Representation

- It is much more compact to just list the edges. We can use a two-column matrix

$$E = \begin{bmatrix} A & B \\ B & A \\ B & C \\ C & A \end{bmatrix}$$

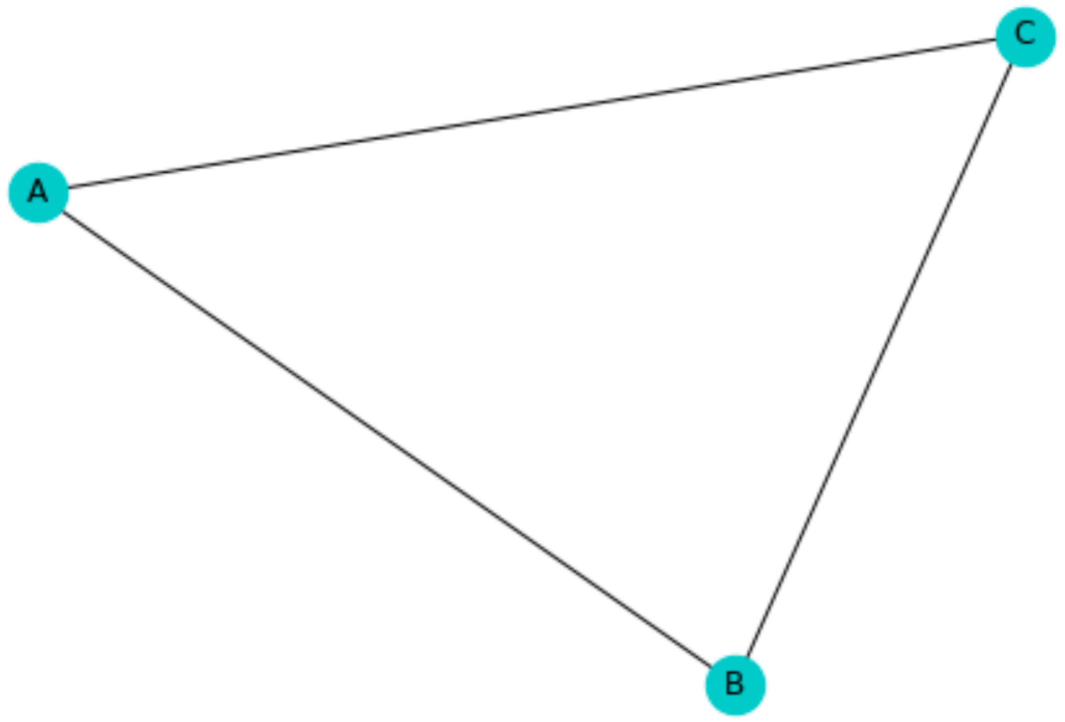
which is represented in Python Networkx as

```
el = pd.DataFrame({'source':['A','B','B','C'],
                   'target':['B','A','C','A']})
print(el)
```

```
  source target
0      A      B
1      B      A
2      B      C
3      C      A
```

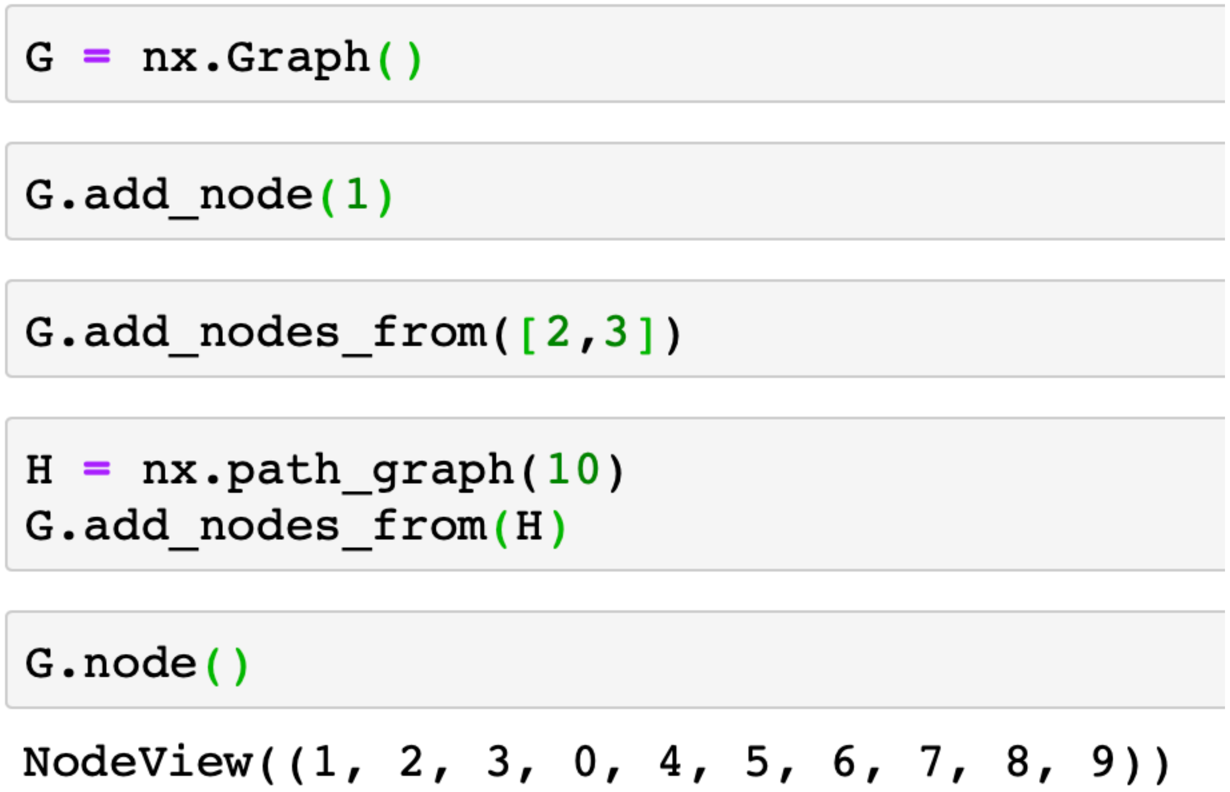
Convert edgelist to networkx object

```
G_el = nx.from_pandas_edgelist(el)
nx.draw(G_el, with_labels=True, node_size=500, node_color='#00CCCC')
```



Networkx object - Graph

- Networkx Graph object carries necessary information for graph representation.



```
G = nx.Graph()
```

```
G.add_node(1)
```

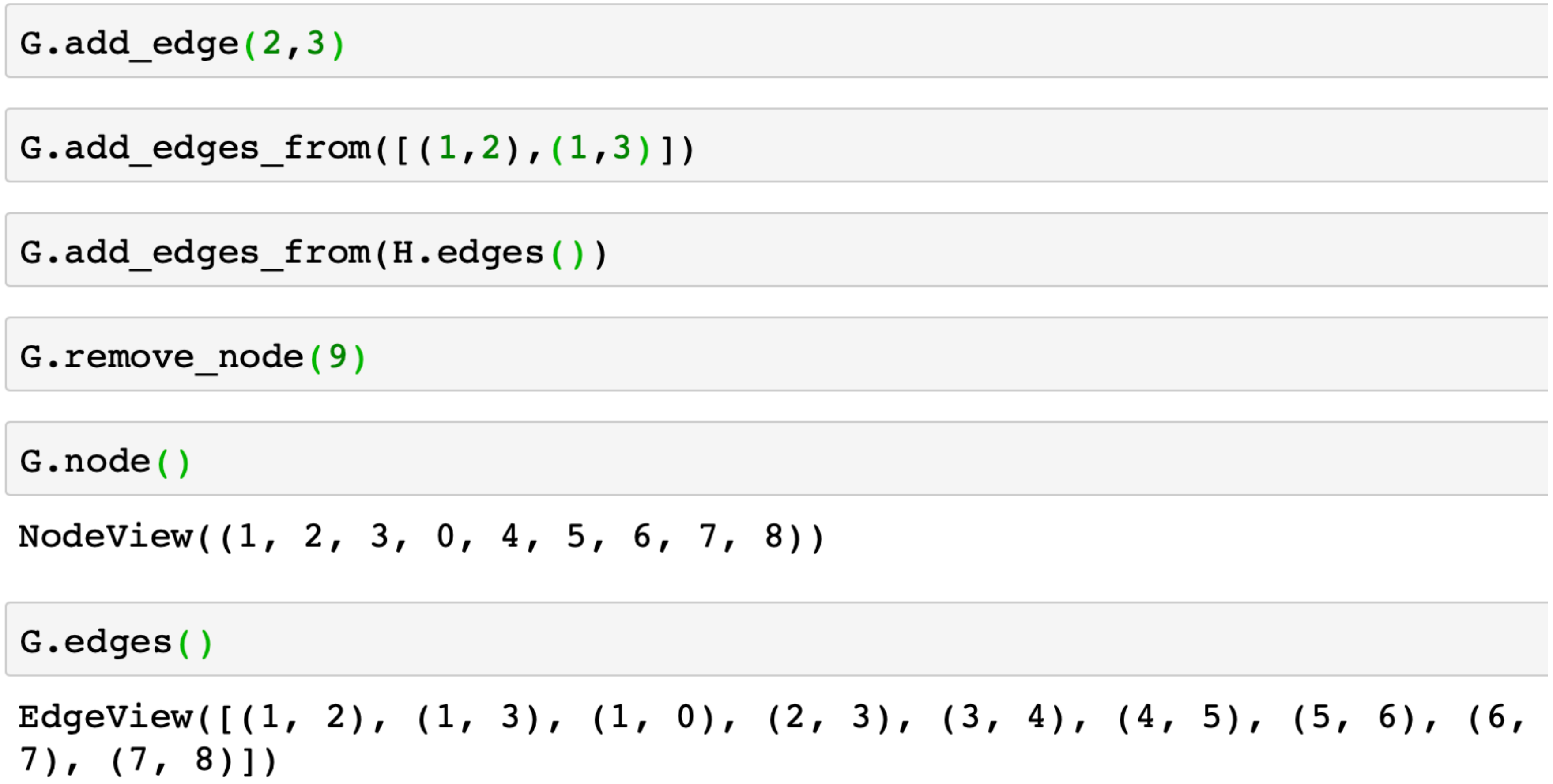
```
G.add_nodes_from([2,3])
```

```
H = nx.path_graph(10)
G.add_nodes_from(H)
```

```
G.node()
```

```
NodeView((1, 2, 3, 0, 4, 5, 6, 7, 8, 9))
```

Networkx – Add edge to graph



```
G.add_edge(2,3)
```

```
G.add_edges_from([(1,2),(1,3)])
```

```
G.add_edges_from(H.edges())
```

```
G.remove_node(9)
```

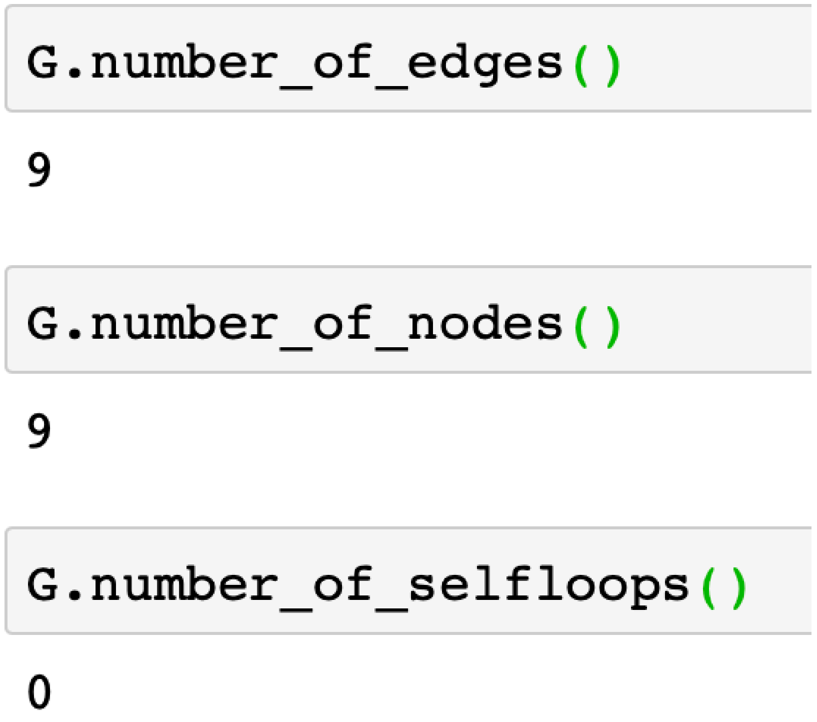
```
G.node()
```

```
NodeView((1, 2, 3, 0, 4, 5, 6, 7, 8))
```

```
G.edges()
```

```
EdgeView([(1, 2), (1, 3), (1, 0), (2, 3), (3, 4), (4, 5), (5, 6), (6,
7), (7, 8)])
```

Networkx – Check number of nodes/edges



```
G.number_of_edges()
```

```
9
```

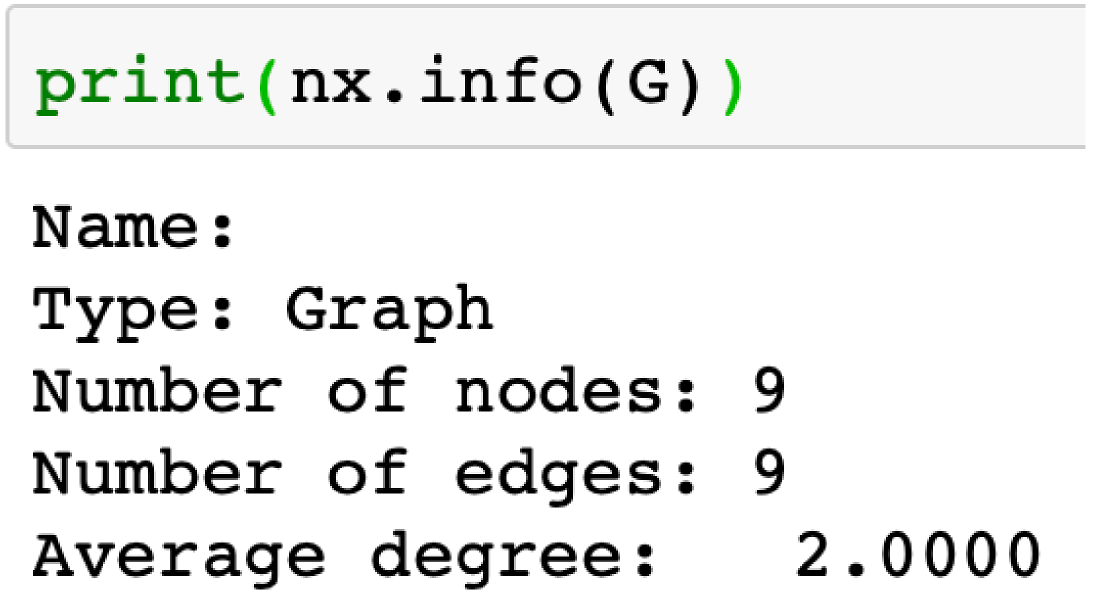
```
G.number_of_nodes()
```

```
9
```

```
G.number_of_selfloops()
```

```
0
```

Graph information



```
print(nx.info(G))
```

```
Name:
Type: Graph
Number of nodes: 9
Number of edges: 9
Average degree:   2.0000
```

Graph Terminology and Special Types of Graphs

Basic Terminology

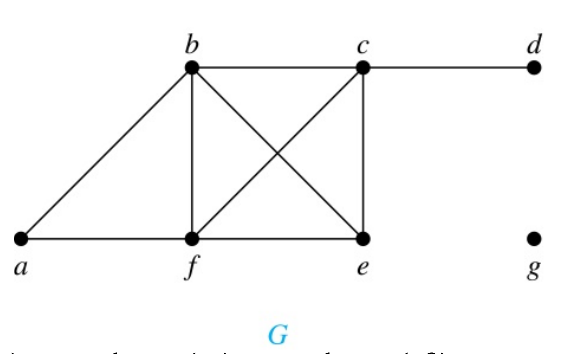
Definition 1. Two vertices u, v in an undirected graph G are called *adjacent* (or *neighbors*) in G if there is an edge e between u and v. Such an edge e is called *incident with* the vertices u and v and e is said to *connect* u and v.

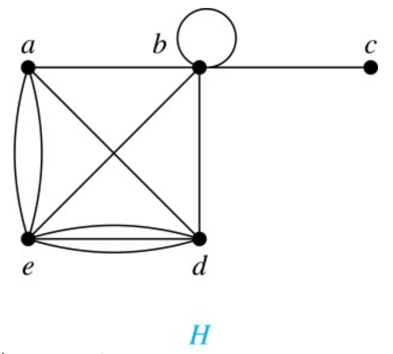
Definition 2. The set of all neighbors of a vertex v of $G = (V, E)$, denoted by $N(v)$, is called the *neighborhood* of v. If A is a subset of V, we denote by $N(A)$ the set of all vertices in G that are adjacent to at least one vertex in A. So, $N(A) = \bigcup_{v\in A} N(v)$.

Definition 3. The *degree of a vertex in a undirected graph* is the number of edges incident with it, except that a loop at a vertex contributes two to the degree of that vertex. The degree of the vertex v is denoted by $\deg(v)$.

Degrees and Neighborhoods of Vertices

Example: What are the degrees and neighborhoods of the vertices in the graphs G and H?





Solution:

G: $\deg(a) = 2$, $\deg(b) = \deg(c) = \deg(f) = 4$, $\deg(d) = 1$, $\deg(e) = 3$, $\deg(g) = 0$.

$N(a) = \{b, f\}$, $N(b) = \{a, c, e, f\}$, $N(c) = \{b, d, e, f\}$, $N(d) = \{c\}$, $N(e) = \{b, c, f\}$, $N(f) = \{a, b, c, e\}$, $N(g) = \phi$.

H: $\deg(a) = 4$, $\deg(b) = \deg(e) = 6$, $\deg(c) = 1$, $\deg(d) = 5$.

$N(a) = \{b, d, e\}$, $N(b) = \{a, b, c, d, e\}$, $N(c) = \{b\}$, $N(d) = \{a, b, e\}$, $N(e) = \{a, b, d\}$.

Directed Graphs

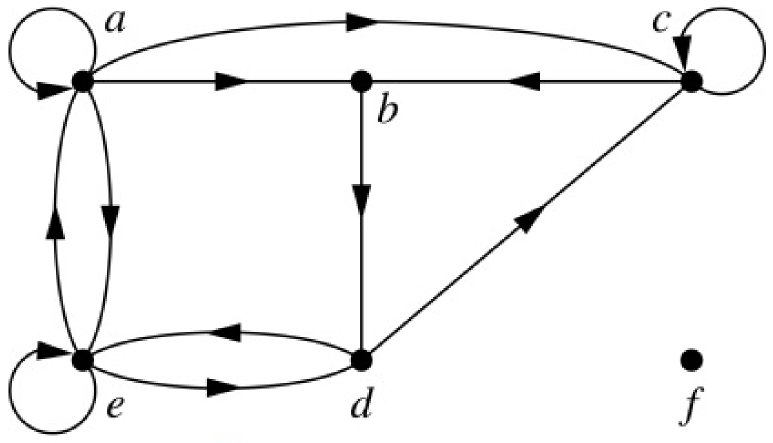
Definition: An *directed graph* $G = (V, E)$ consists of V, a nonempty set of *vertices* (or *nodes*), and E, a set of *directed edges* or *arcs*. Each edge is an ordered pair of vertices. The directed edge (u,v) is said to start at u and end at v.

Definition: Let (u,v) be an edge in G. Then u is the *initial vertex* of this edge and is *adjacent to* v and v is the *terminal* (or *end*) *vertex* of this edge and is *adjacent from* u. The initial and terminal vertices of a loop are the same.

Directed Graphs (*continued*)

Definition: The *in-degree of a vertex v*, denoted *deg*$^-$(v), is the number of edges which terminate at *v*. The *out-degree of v*, denoted *deg*$^+$(v), is the number of edges with *v* as their initial vertex. Note that a loop at a vertex contributes 1 to both the in-degree and the out-degree of the vertex.

Example: In the graph *G* we have



G

$\deg^-(a) = 2$, $\deg^-(b) = 2$, $\deg^-(c) = 3$, $\deg^-(d) = 2$, $\deg^-(e) = 3$, $\deg^-(f) = 0$.

$\deg^+(a) = 4$, $\deg^+(b) = 1$, $\deg^+(c) = 2$, $\deg^+(d) = 2$, $\deg^+(e) = 3$, $\deg^+(f) = 0$.

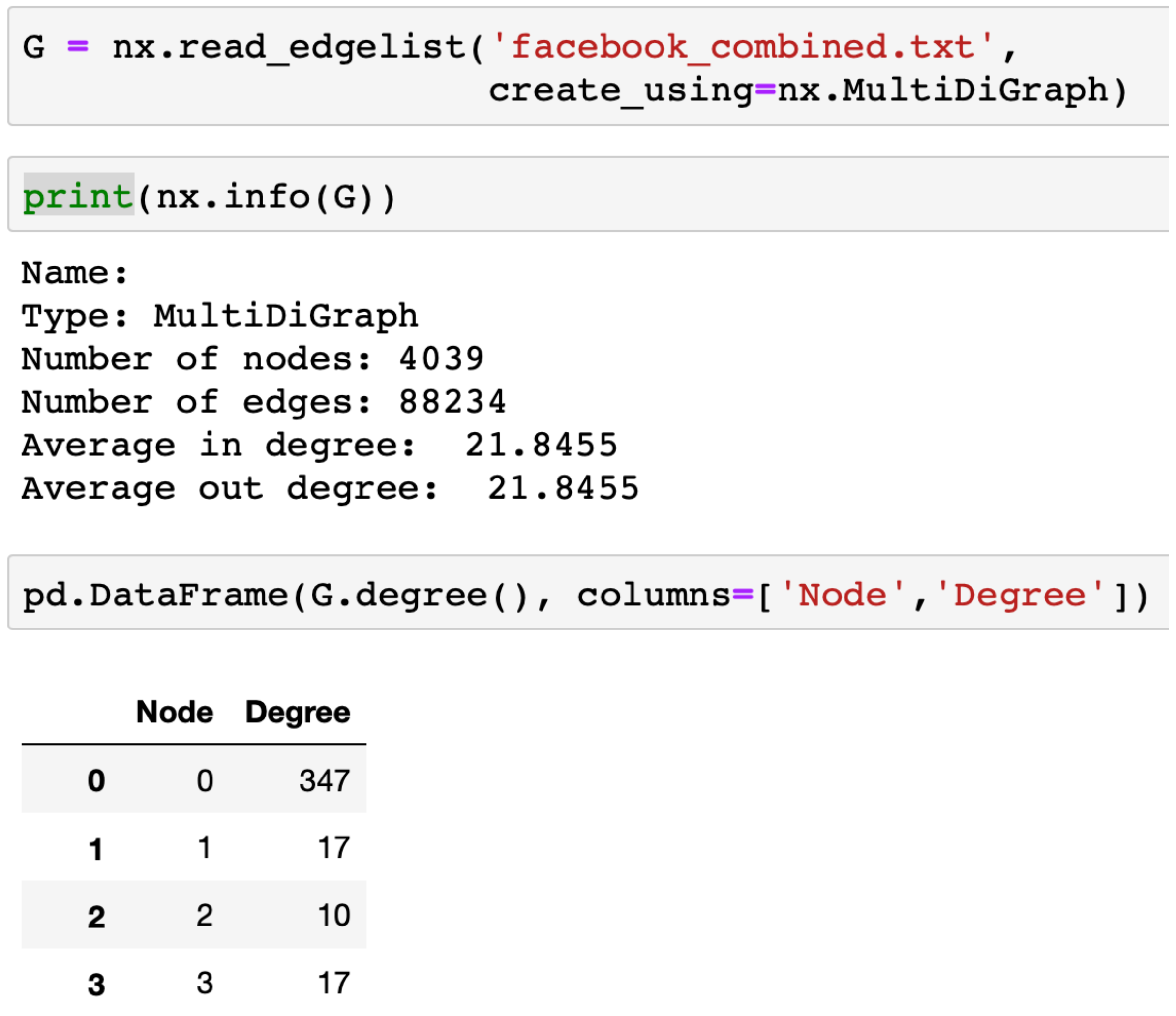
Directed Graphs (*continued*)

Theorem 3: Let $G = (V, E)$ be a graph with directed edges. Then:

$$|E| = \sum_{v \in V} deg^-(v) = \sum_{v \in V} deg^+(v).$$

Proof: The first sum counts the number of outgoing edges over all vertices and the second sum counts the number of incoming edges over all vertices. It follows that both sums equal the number of edges in the graph. ◀

Python: Degree



```python
G = nx.read_edgelist('facebook_combined.txt',
                     create_using=nx.MultiDiGraph)
```

```python
print(nx.info(G))
```

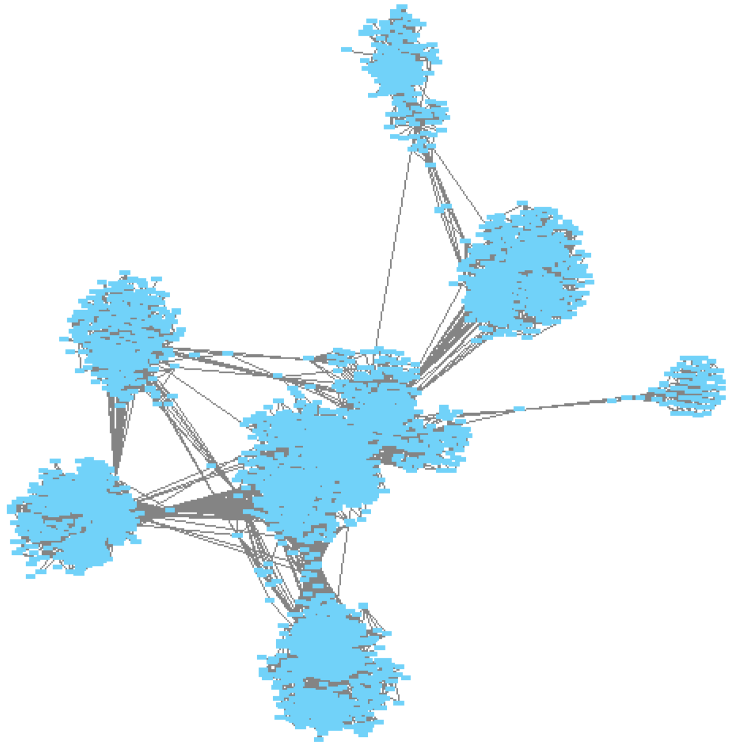
```
Name:
Type: MultiDiGraph
Number of nodes: 4039
Number of edges: 88234
Average in degree:  21.8455
Average out degree:  21.8455
```

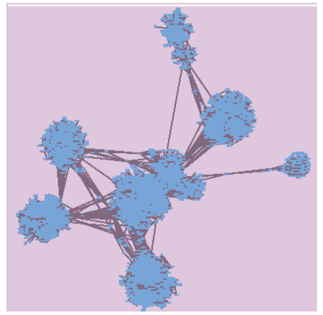
```python
pd.DataFrame(G.degree(), columns=['Node','Degree'])
```

	Node	Degree
0	0	347
1	1	17
2	2	10
3	3	17

Export to graphml and use Cytoscape

```
nx.write_graphml(G,'test.graphml')
```





In/Out Degree

```python
pd.DataFrame(G.in_degree, columns=['Node','Degree'])
```

	Node	Degree
0	0	0
1	1	1
2	2	1
3	3	1

```python
pd.DataFrame(G.out_degree, columns=['Node','Degree'])
```

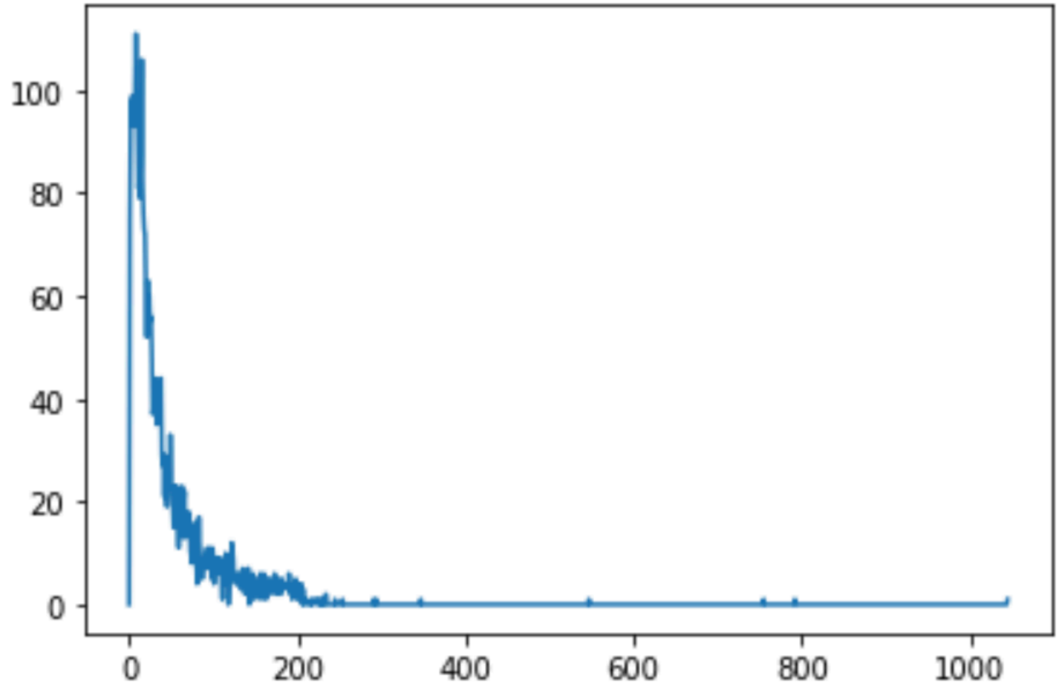
	Node	Degree
0	0	347
1	1	16
2	2	9
3	3	16

BIG DATA
EXPERIENCE

Degree histogram

```
plt.plot(nx.degree_histogram(G))
```

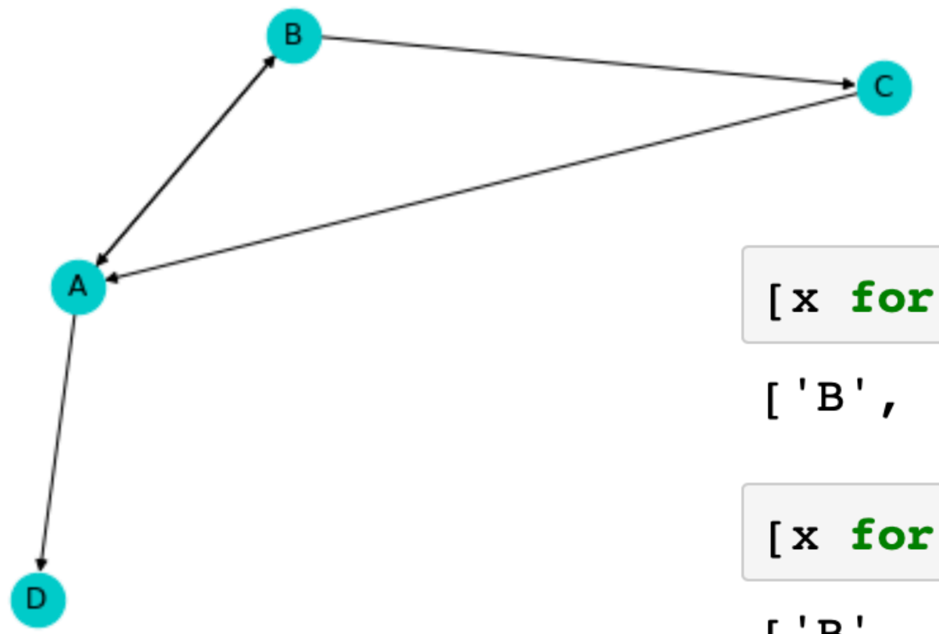
```
[<matplotlib.lines.Line2D at 0xa253bf4e0>]
```



KMUTT g·able

Python: neighbors

```
G1 = nx.from_pandas_adjacency(A_df, create_using=nx.MultiDiGraph)
G1.add_edge('A','D')
nx.draw(G1, with_labels=True, node_size=500, node_color='#00CCCC')
plt.show()
```



```
[x for x in nx.neighbors(G1,'A')]
```

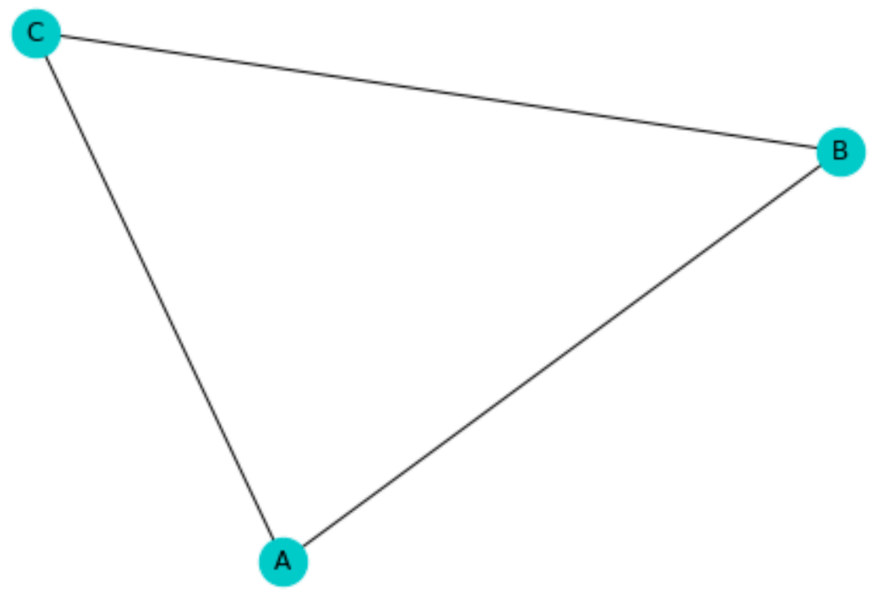
```
['B', 'D']
```

```
[x for x in nx.all_neighbors(G1,'A')]
```

```
['B', 'C', 'B', 'D']
```

Common neighbors (undirected graph)

```
nx.draw(G_df, with_labels=True, node_size=500, node_color='#00CCCC')
plt.show()
```

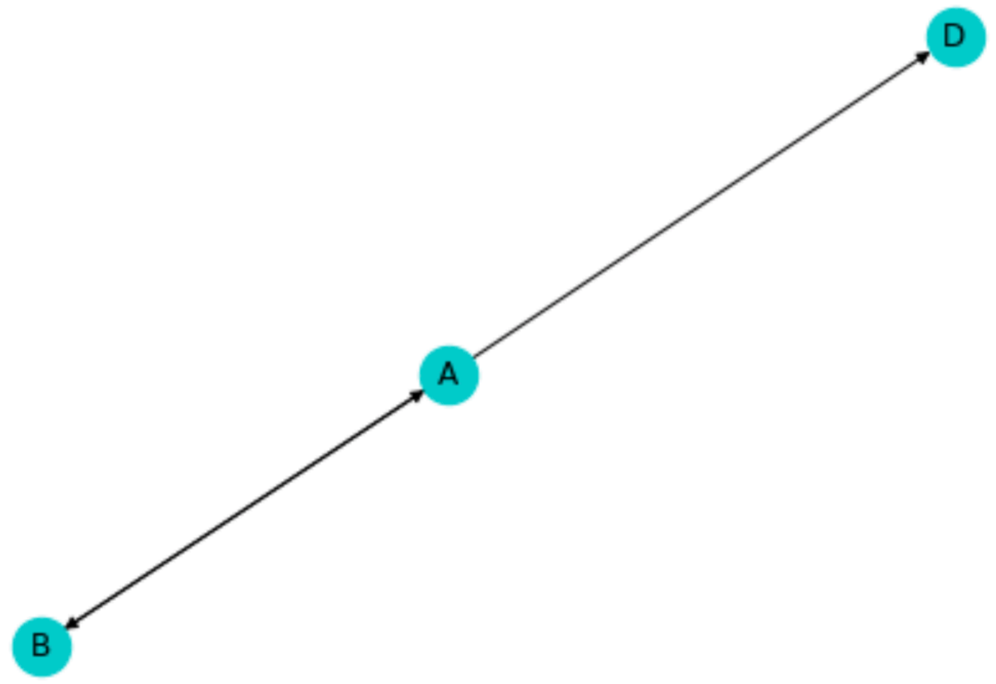


```
[x for x in nx.common_neighbors(G_df,'B','A')]
```

```
['C']
```

Subgraph

```
G1_sub = nx.subgraph(G1,['A','B','D'])
nx.draw(G1_sub, with_labels=True, node_size=500, node_color='#00CCCC')
```



Paths

- **Informal Definition:** A *path* is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph.
- As the path travels along its edges, it visits the vertices along this path, that is, the endpoints of these.

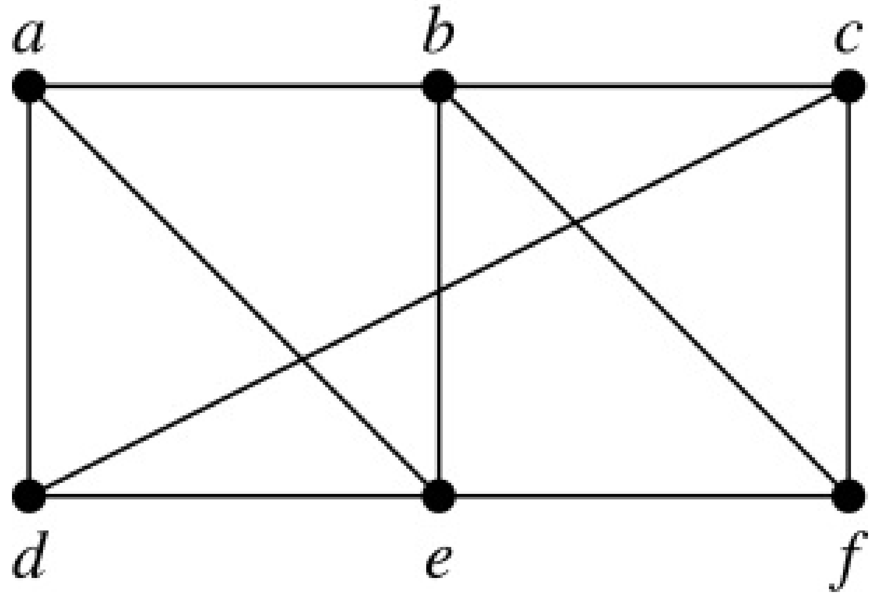
  Applications: Numerous problems can be modeled with paths formed by traveling along edges of graphs such as:
  - determining whether a message can be sent between two computers.
  - efficiently planning routes for mail delivery.
  - finding directions in Google Map
  - Etc.

Paths

Definition: Let n be a nonnegative integer and G an undirected graph. A *path* of *length n* from u to v in G is a sequence of n edges $e_1, \ldots, e_n$ of G for which there exists a sequence $x_0 = u, x_1, \ldots, x_{n-1}, x_n = v$ of vertices such that e_i has, for $i = 1, \ldots, n$, the endpoints x_{i-1} and x_i.

- When the graph is simple, we denote this path by its vertex sequence $x_0, x_1, \ldots, x_n$ (since listing the vertices uniquely determines the path).
- The path is a *circuit* if it begins and ends at the same vertex ($u = v$) and has length greater than zero.
- The path or circuit is said to *pass through* the vertices $x_1, x_2, \ldots, x_{n-1}$ and *traverse* the edges $e_1, \ldots, e_n$.
- A path or circuit is *simple* if it does not contain the same edge more than once.

Paths



Example: In the simple graph here:

a, *d*, *c*, *f*, *e* is a simple path of length 4.

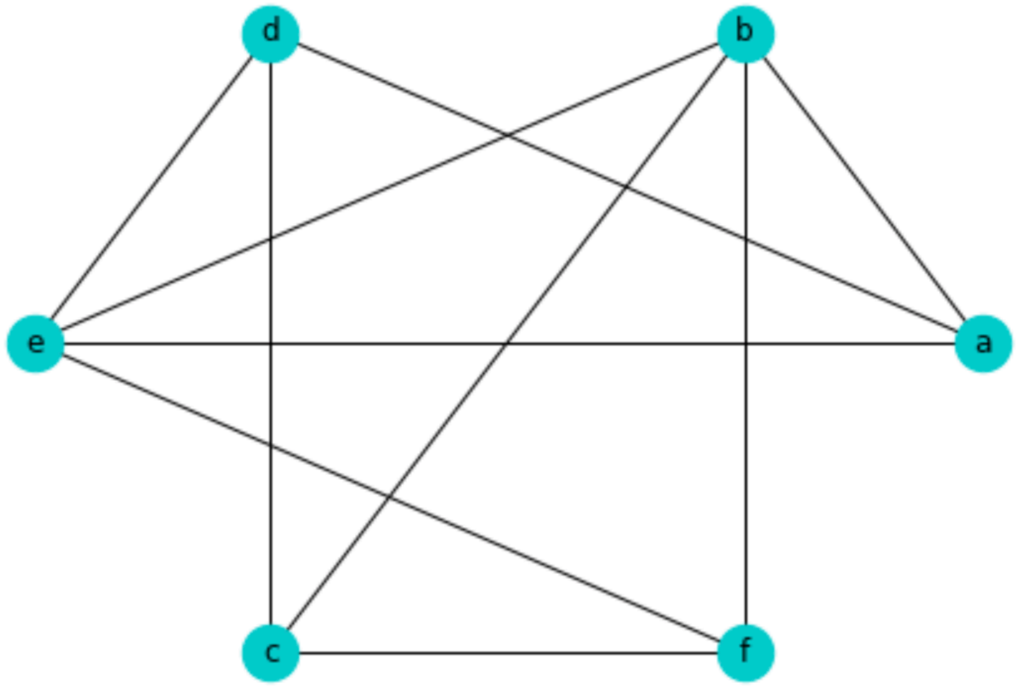
d, *e*, *c*, *a* is not a path because *e* is not connected to *c*.

b, *c*, *f*, *e*, *b* is a circuit of length 4.

a, *b*, *e*, *d*, *a*, *b* is a path of length 5, but it is not a simple path.

Simple path

```
el2 = pd.DataFrame([['a','b'],
                    ['a','d'],
                    ['a','e'],
                    ['b','c'],
                    ['b','e'],
                    ['b','f'],
                    ['c','d'],
                    ['c','f'],
                    ['d','e'],
                    ['e','f']], columns=['source','target'])
G2 = nx.from_pandas_edgelist(el2)
nx.draw_shell(G2, with_labels=True, node_size=500, node_color='#00CCCC')
```



Simple path

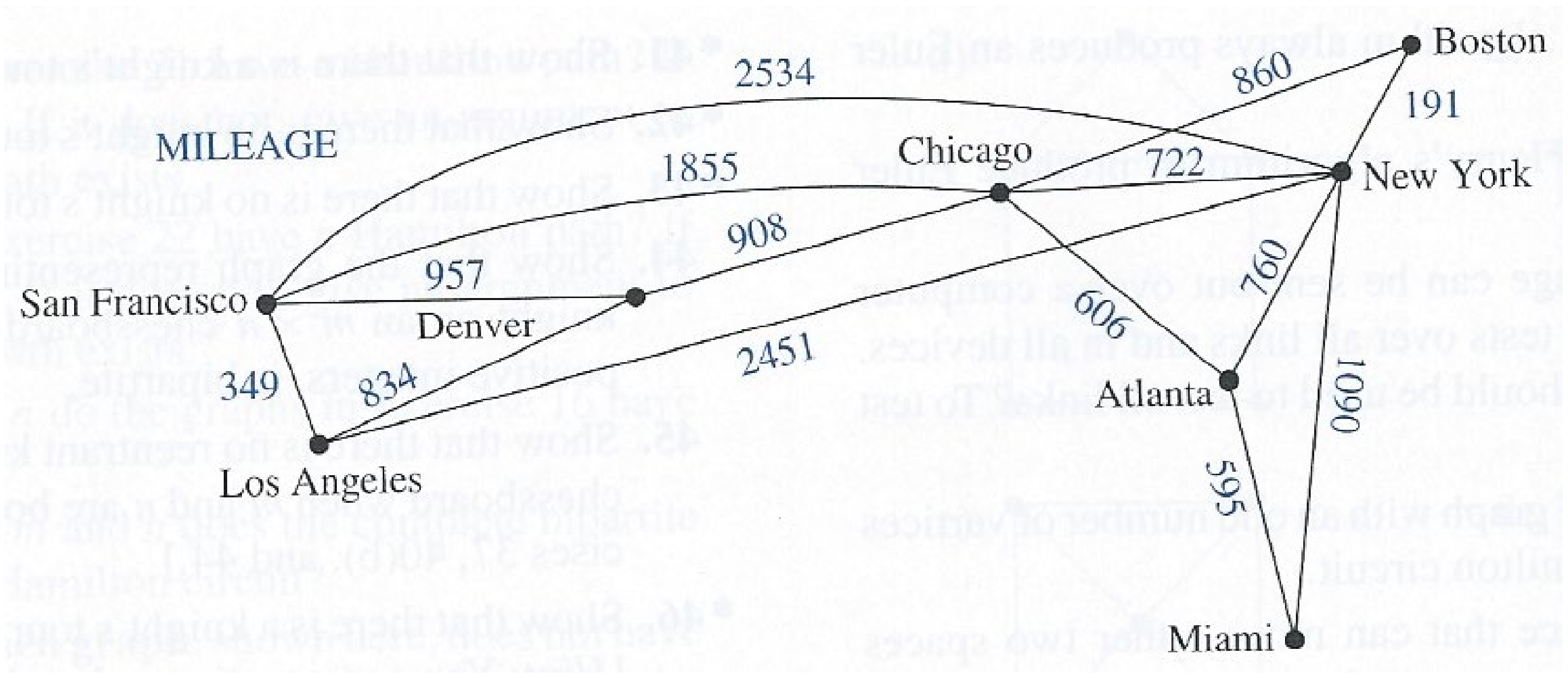
```
[x for x in nx.simple_paths.all_simple_paths(G2,'a','c')]
```

```
[['a', 'b', 'c'],
 ['a', 'b', 'e', 'd', 'c'],
 ['a', 'b', 'e', 'f', 'c'],
 ['a', 'b', 'f', 'c'],
 ['a', 'b', 'f', 'e', 'd', 'c'],
 ['a', 'd', 'c'],
 ['a', 'd', 'e', 'b', 'c'],
 ['a', 'd', 'e', 'b', 'f', 'c'],
 ['a', 'd', 'e', 'f', 'b', 'c'],
 ['a', 'd', 'e', 'f', 'c'],
 ['a', 'e', 'b', 'c'],
 ['a', 'e', 'b', 'f', 'c'],
 ['a', 'e', 'd', 'c'],
 ['a', 'e', 'f', 'b', 'c'],
 ['a', 'e', 'f', 'c']]
```

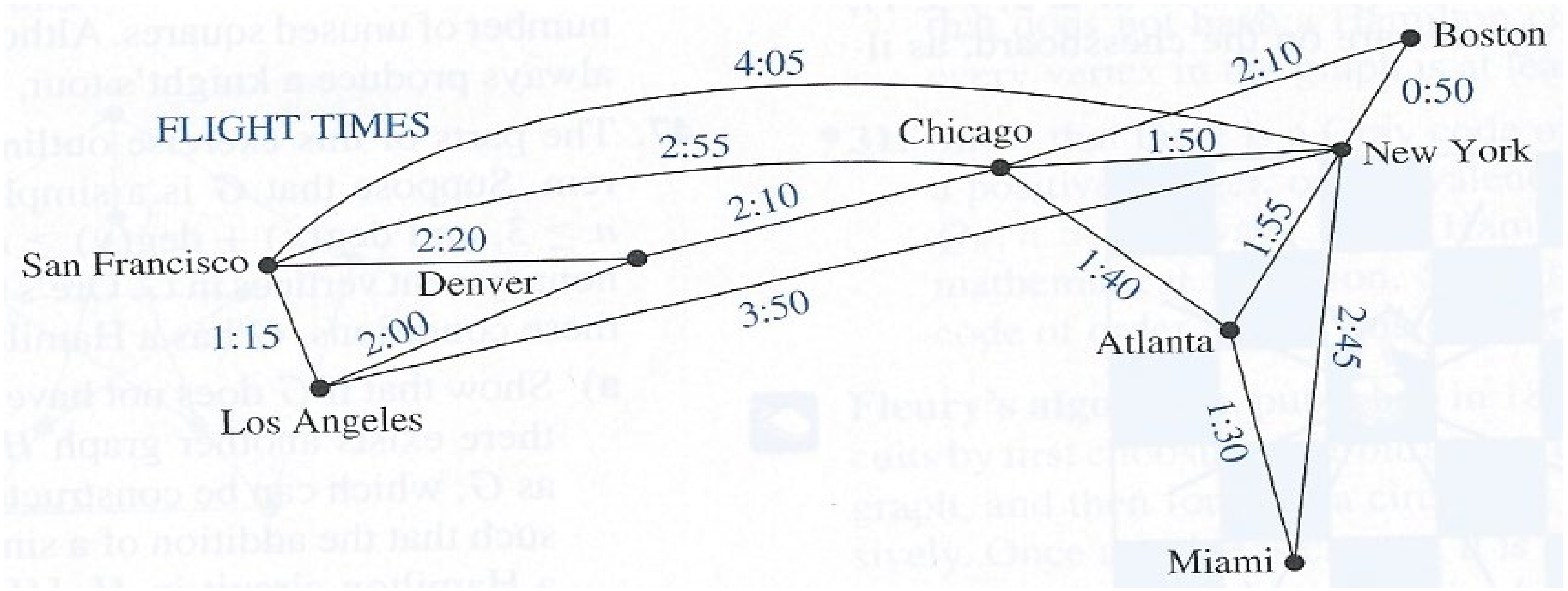
Shortest path algorithm

Weighted Graph

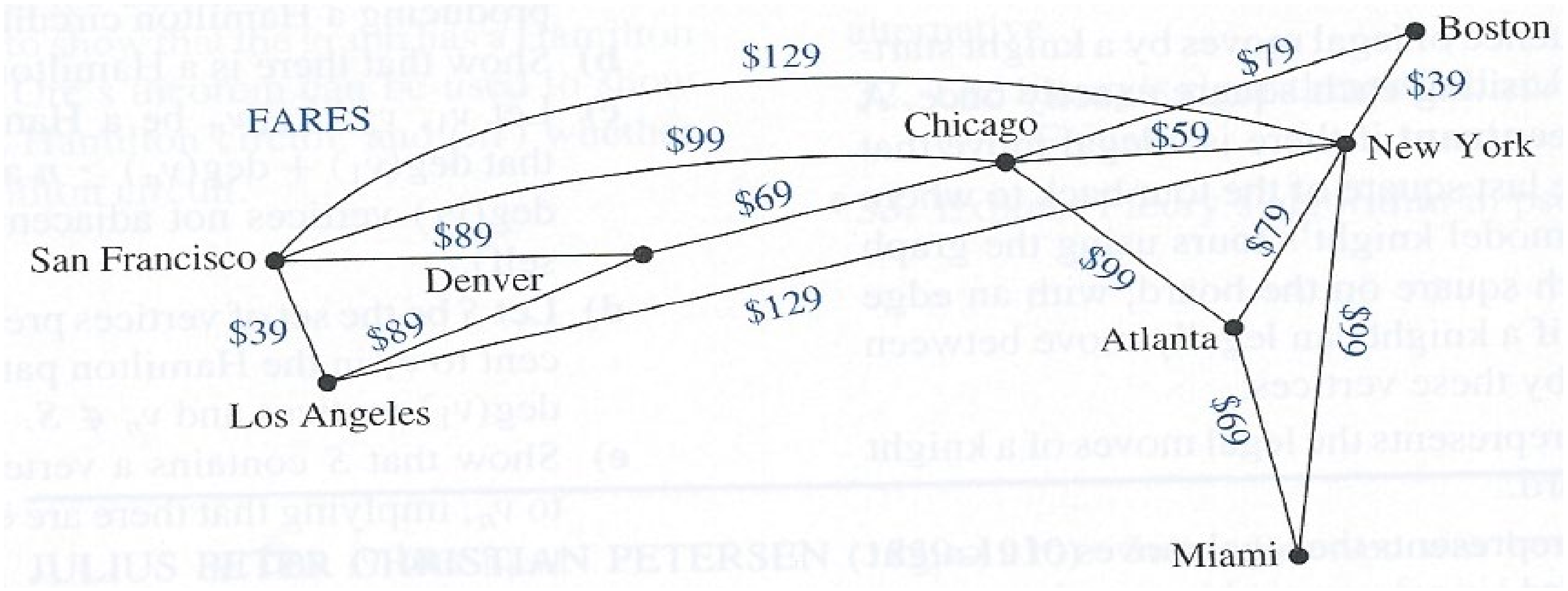
- Many problems can be modeled using graphs with weight assigned to their edges.



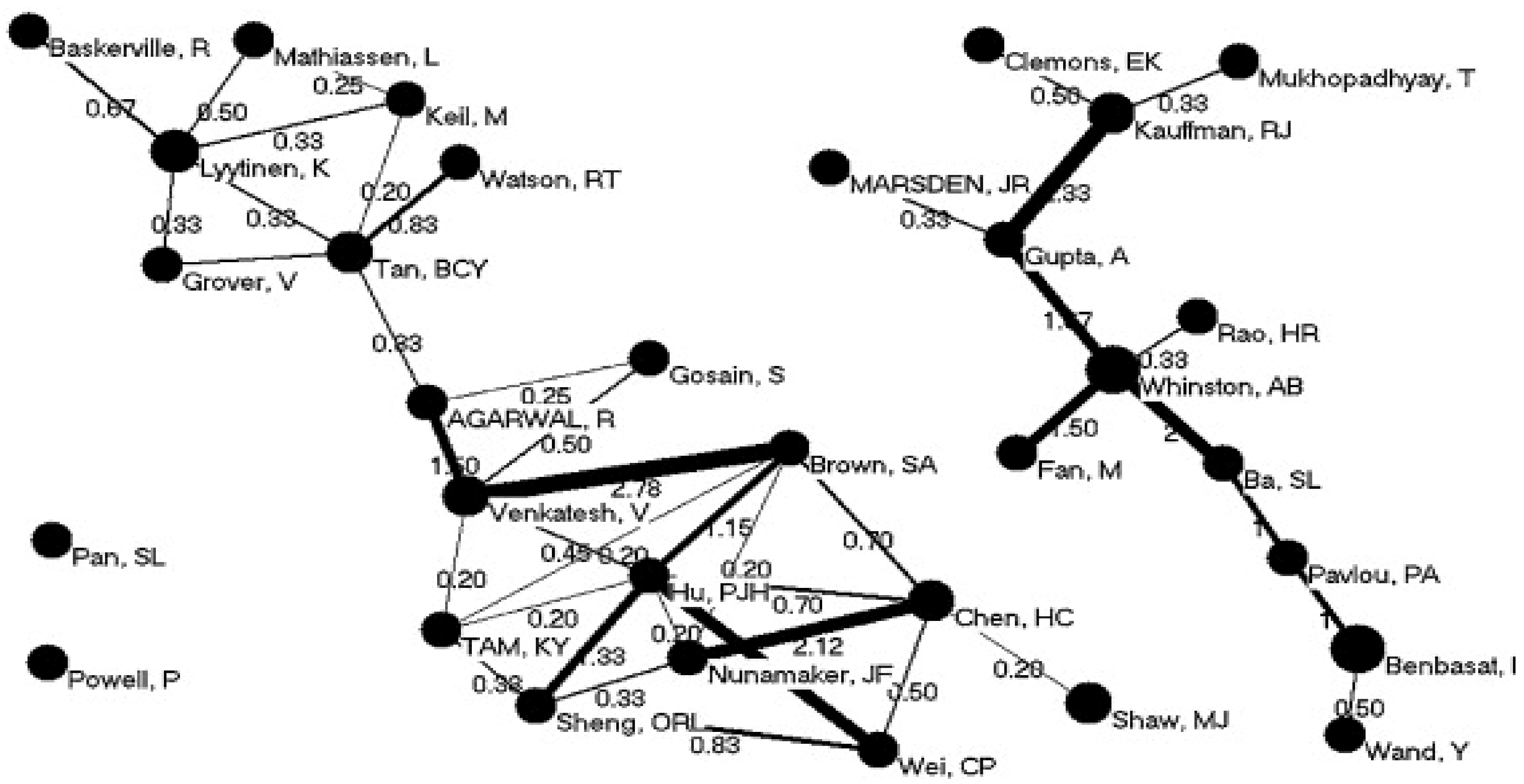
Weighted Graph



Weighted Graphs



Weighted Graph
Measuring Collaborations

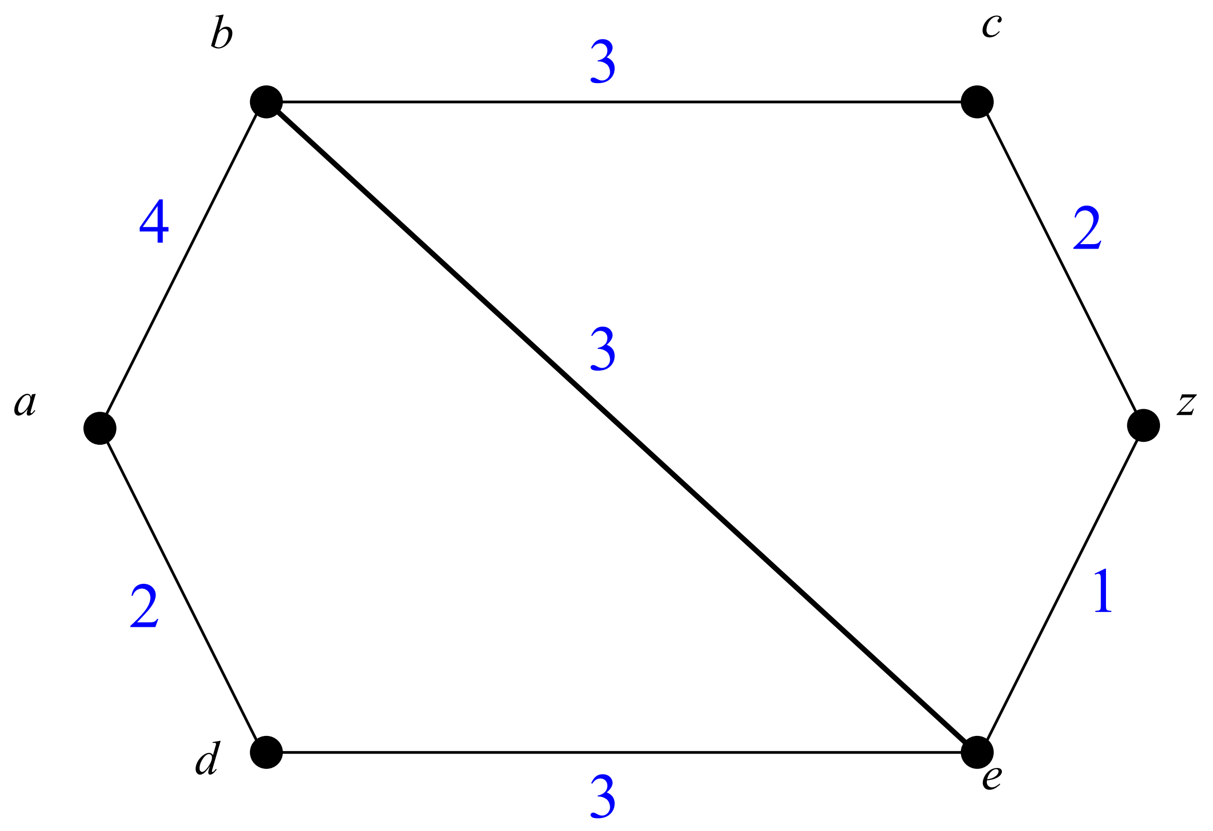


Shortest Path

- Determining a path of **least length** between two vertices in a network is one of problems that often arise frequently.
- The **length** of a path in a weighted graph is the sum of the weights of the edges in this path.
- The main question is

  "What is the shortest path between two given vertices?"
- In Google Map, this indicates the suggested route between two places
- In the airline system, this is the suggested routes that may have the least fare, changes or distances

How can you find a shortest path from *a* to *z*



Dijkstra’s algorithm finds the length of a shortest path between two vertices in a connected simple undirected weighted graph.

Dijkstra’s Algorithm

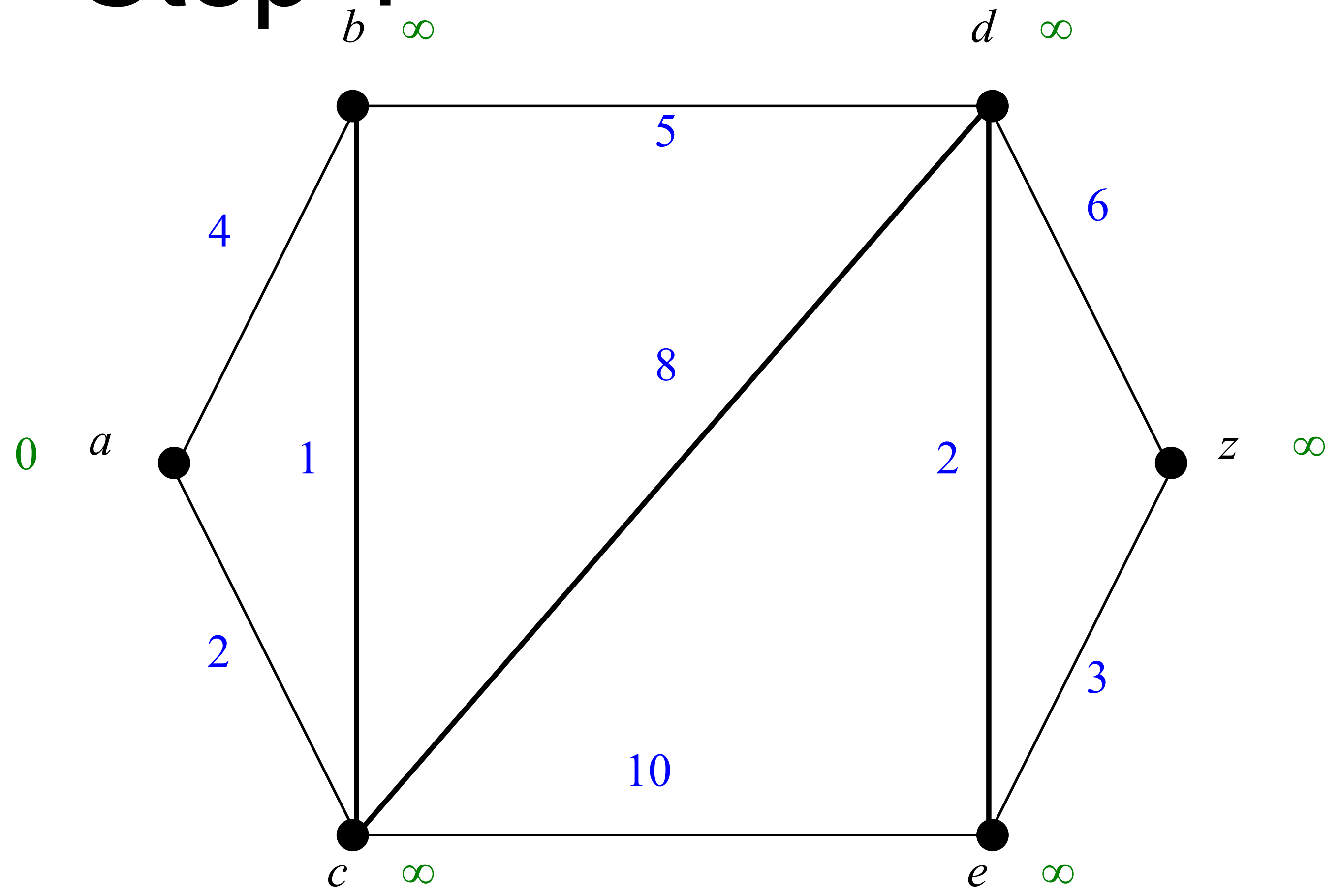
```
procedure Dijkstra(G: weighted connected simple graph, with all
weights positive) [G has vertices $a=v_0,v_1,\ldots,v_n=z$ and the lengths
$w(v_i,v_j)$ where $w(v_i,v_j) = \infty$ if $[v_i,v_j]$ is not an edge in G]
for $i := 1$ to $n$
    $L(v_i) := \infty$
$L(a) := 0$
$S := \emptyset$
while $z \notin S$
    $u :=$ a vertex not in $S$ with $L(u)$ minimal
    $S := S \cup \{u\}$
    for all vertices $v$ not in S
        if $L(u) + w(u,v) < L(v)$ then $L(v) := L(u) + w(u,v)$
return $L(z)$ {$L(z)$ = length of the shortest path}
```

the labels are now initialized so that the label of a is 0 and all other labels are ∞, and S is the empty set

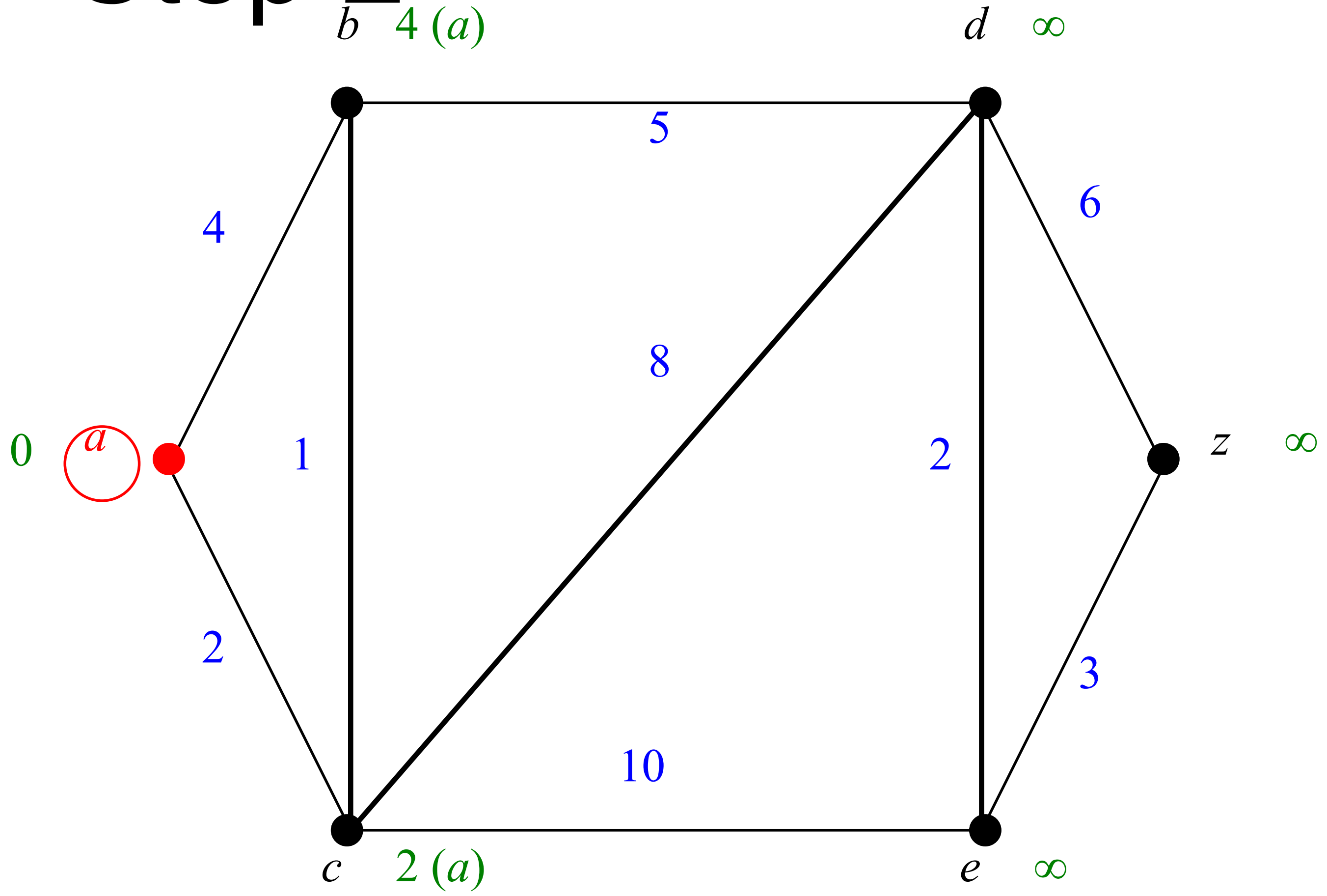
L(*v*) is the length of a shortest path from *the original vertex* to vertex *v* containing only vertices in S

This adds a vertex to S with minimal label and updates the labels of vertices not in S

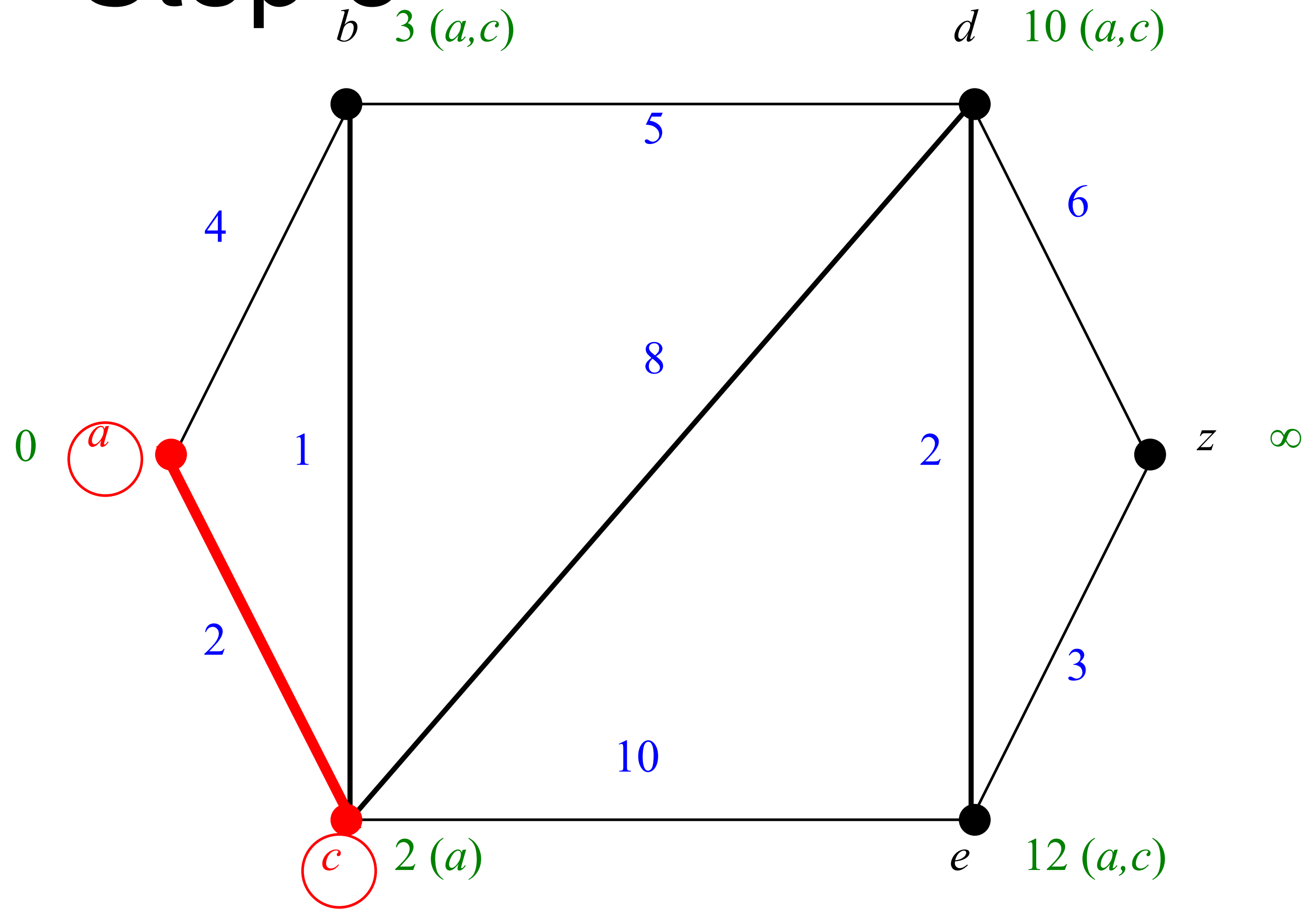
Example
Step 1



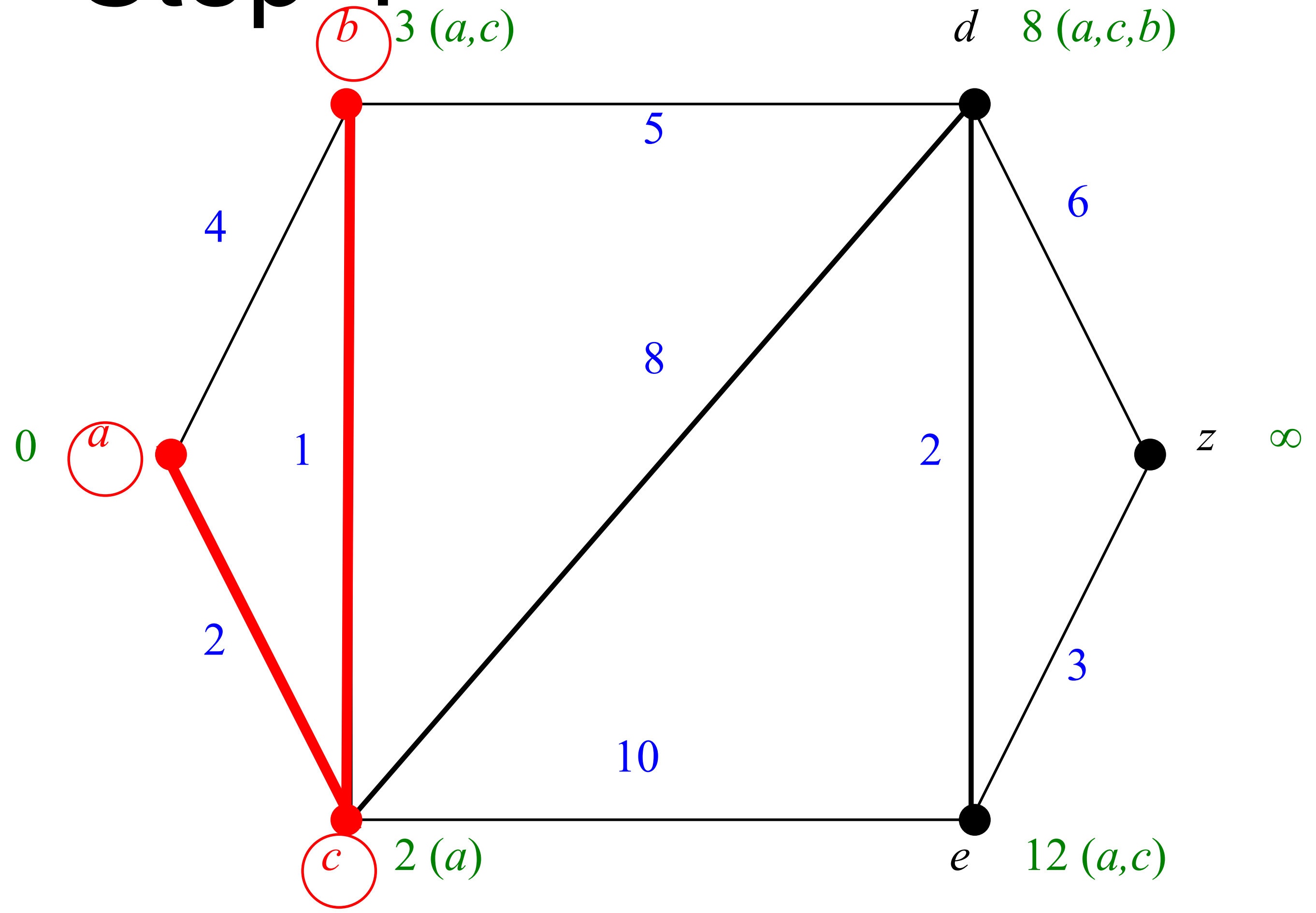
Example Step 2



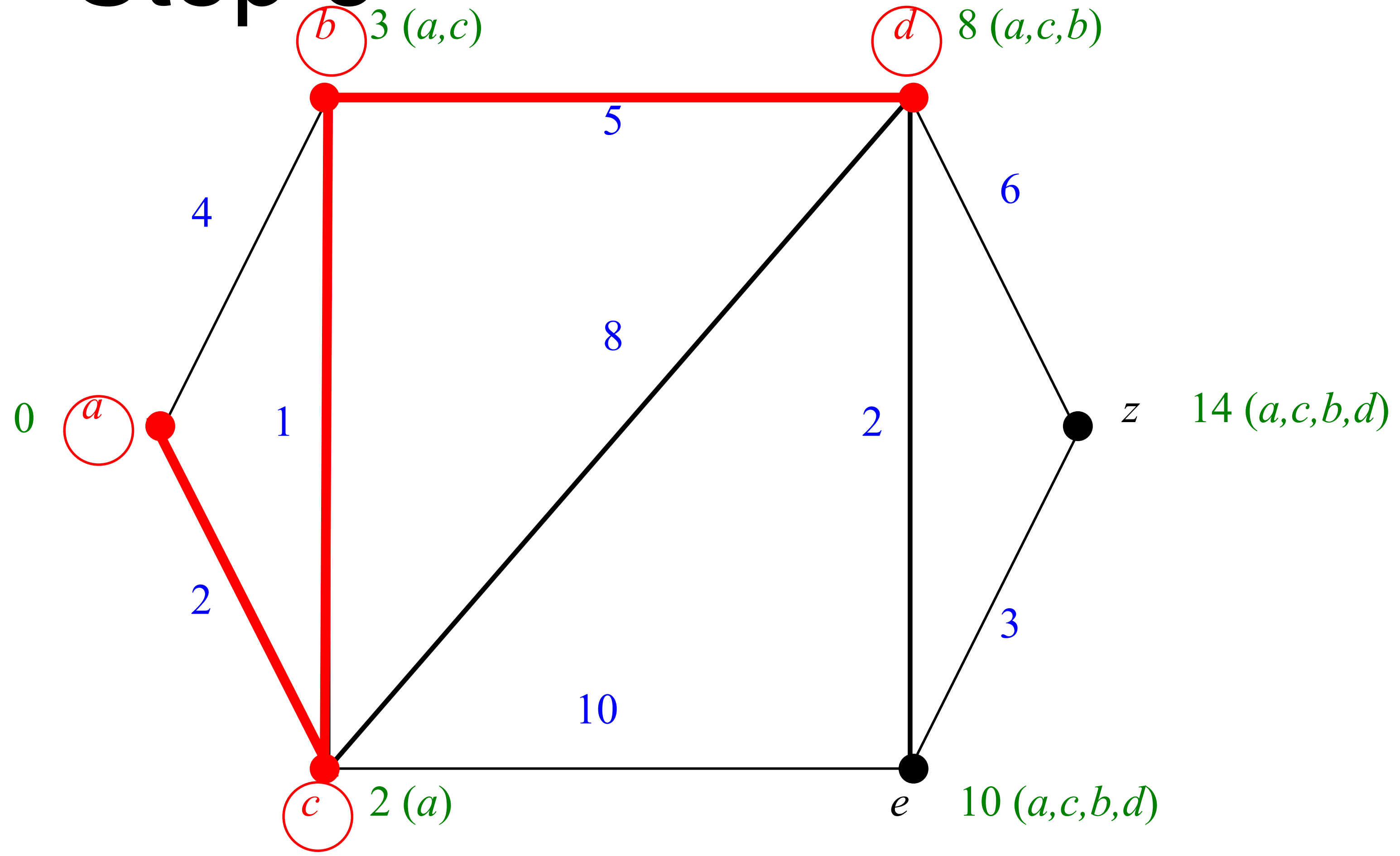
Example Step 3



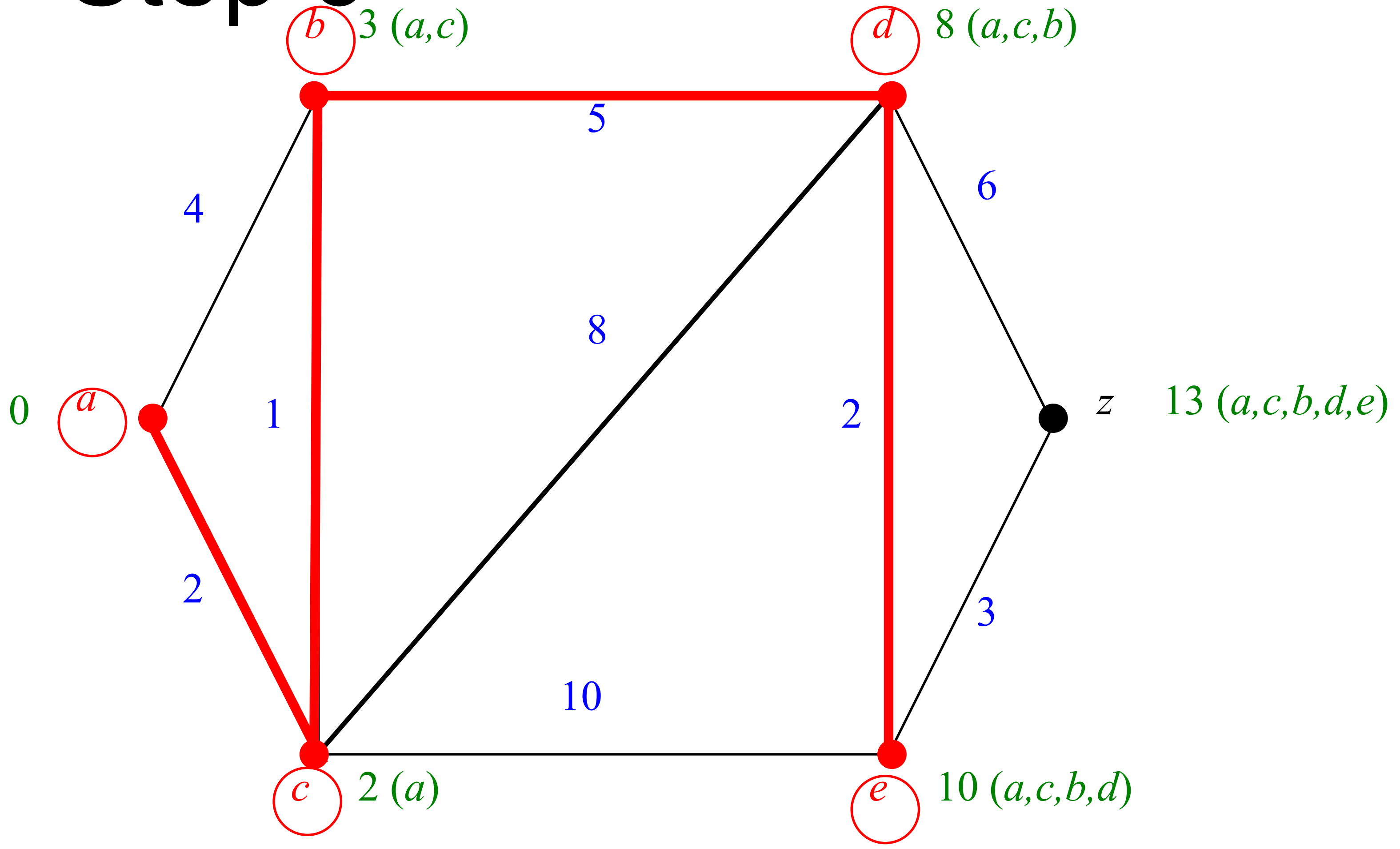
Example
Step 4



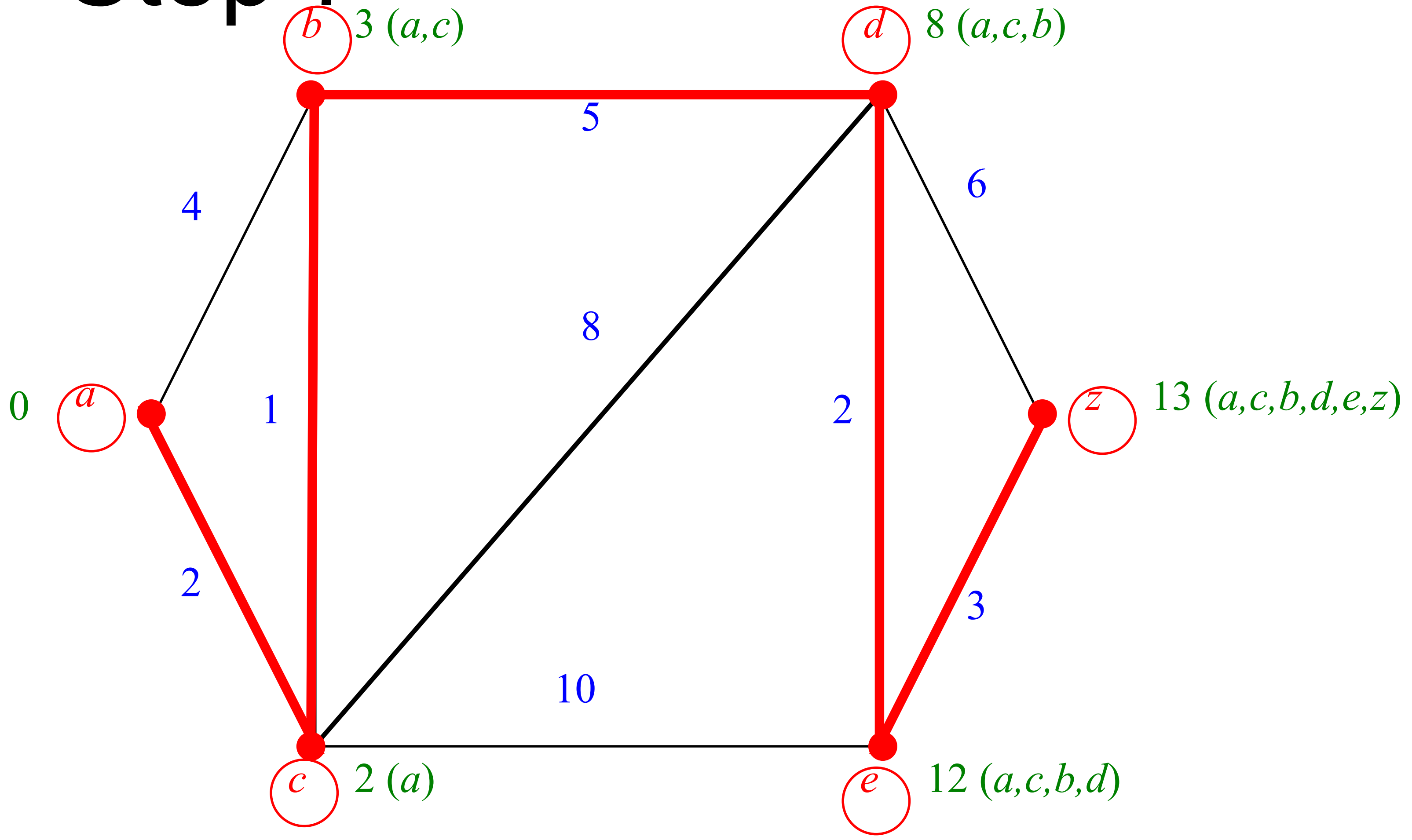
Example
Step 5



Example
Step 6

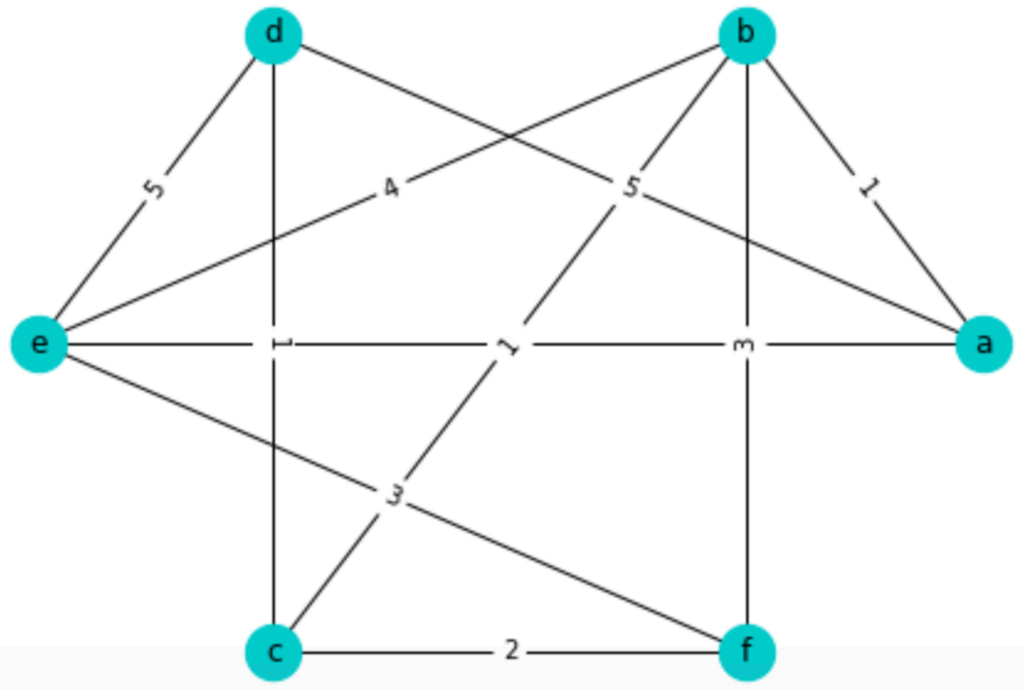


Example Step 7

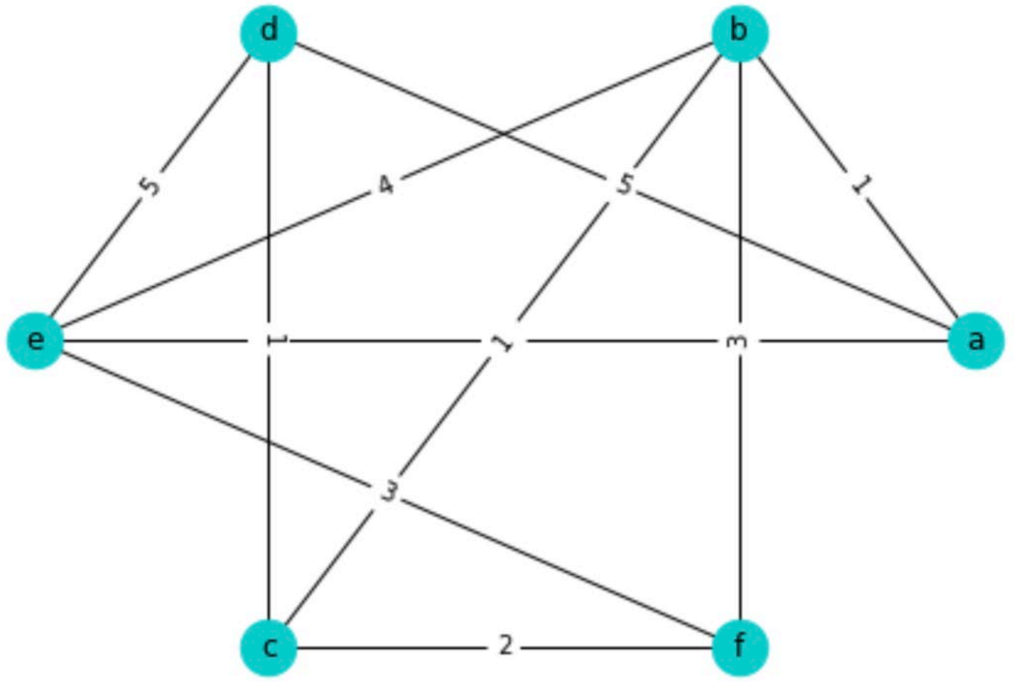


Create graph with weight

```
elw = pd.DataFrame([['a','b',1],
                    ['a','d',5],
                    ['a','e',2],
                    ['b','c',1],
                    ['b','e',4],
                    ['b','f',3],
                    ['c','d',1],
                    ['c','f',2],
                    ['d','e',5],
                    ['e','f',3]], columns=['source','target','weight'])
G2 = nx.from_pandas_edgelist(elw,edge_attr='weight')
nx.draw_shell(G2, with_labels=True, node_size=500, node_color='#00CCCC')
labels = nx.get_edge_attributes(G2,'weight')
nx.draw_networkx_edge_labels(G2,pos=nx.shell_layout(G2),edge_labels=labels)
plt.show()
```



Shortest path

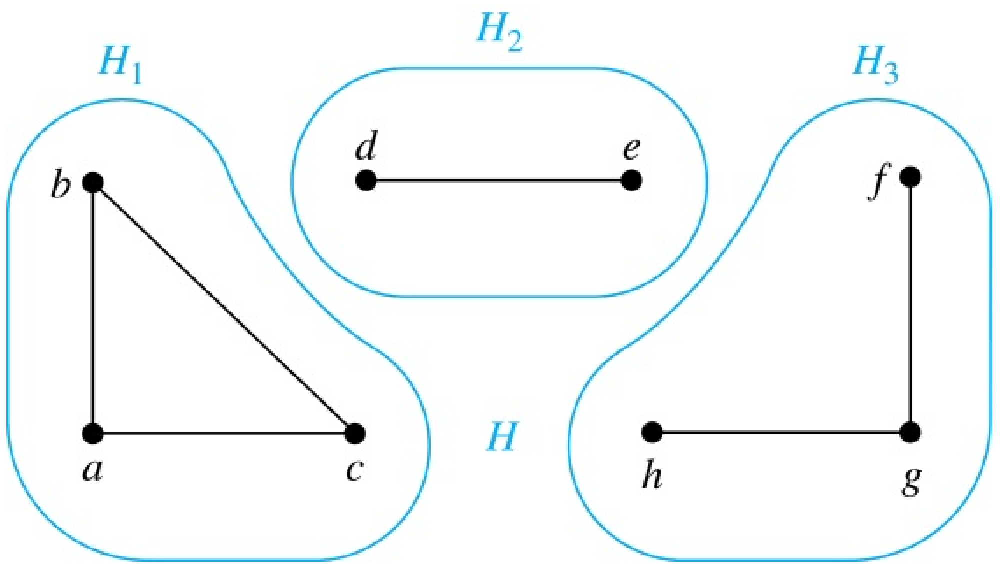


```
nx.shortest_path(G2,'a','d',weight='weight')
```

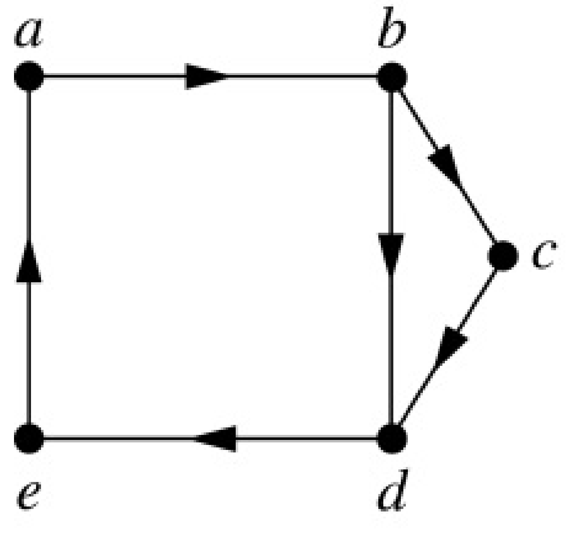
```
['a', 'b', 'c', 'd']
```

Connected Components

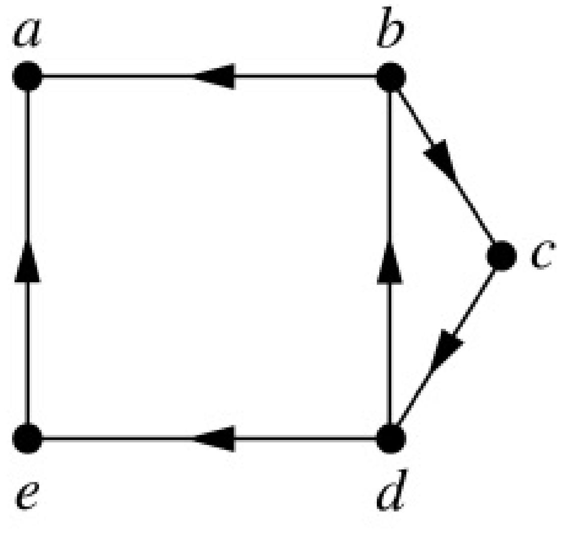
- A *connected component* of a graph G is a connected subgraph of G that is not a proper subgraph of another connected subgraph of G.
- A graph G that is not connected has two or more connected components that are disjoint and have G as their union.



Connectedness in Directed Graph



G

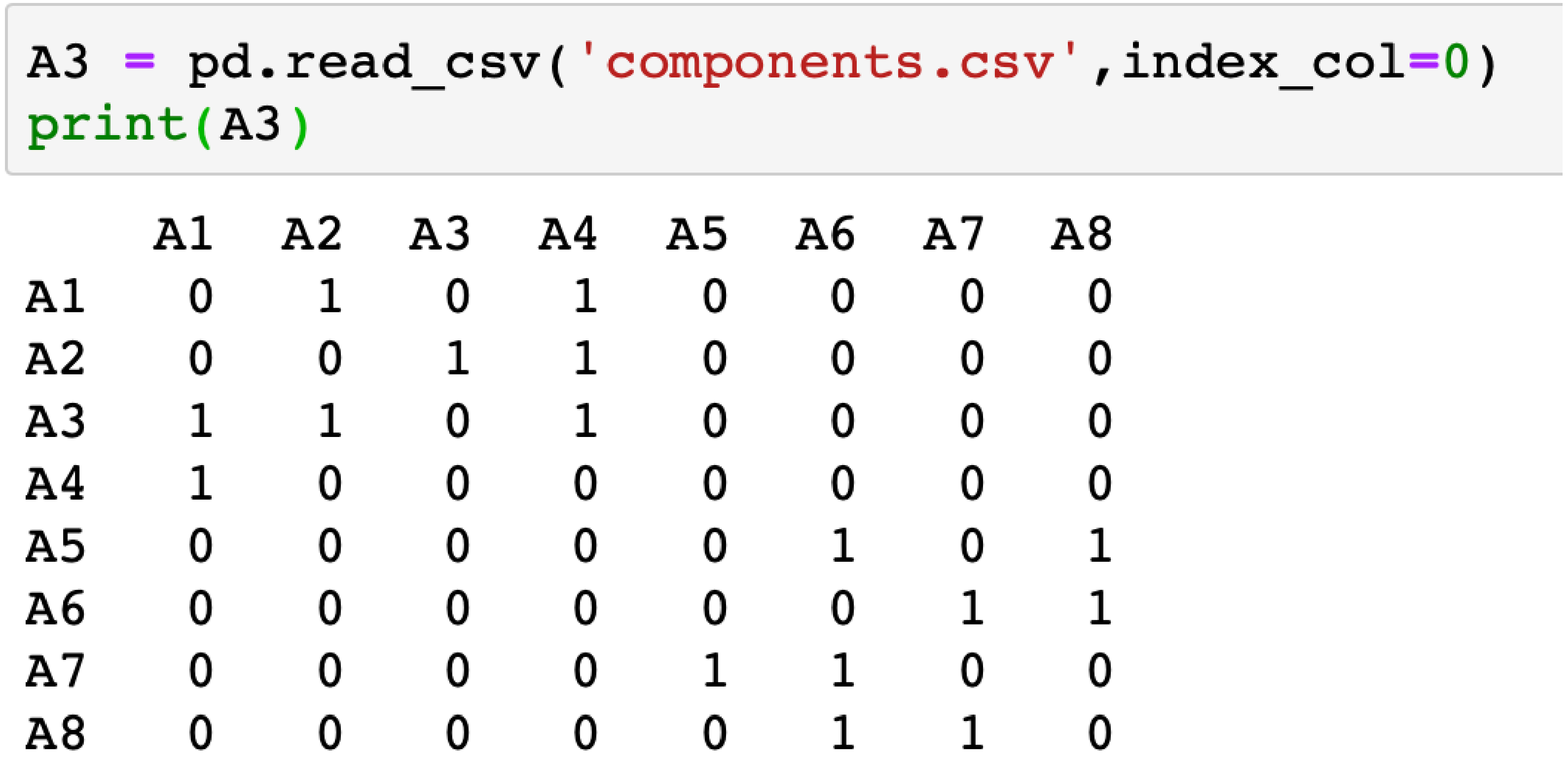


H

Strongly connected because there is a path from every pair for vertices in G

Weakly connected because there is no path from *a* to *b*

Finding connected component
Create graph

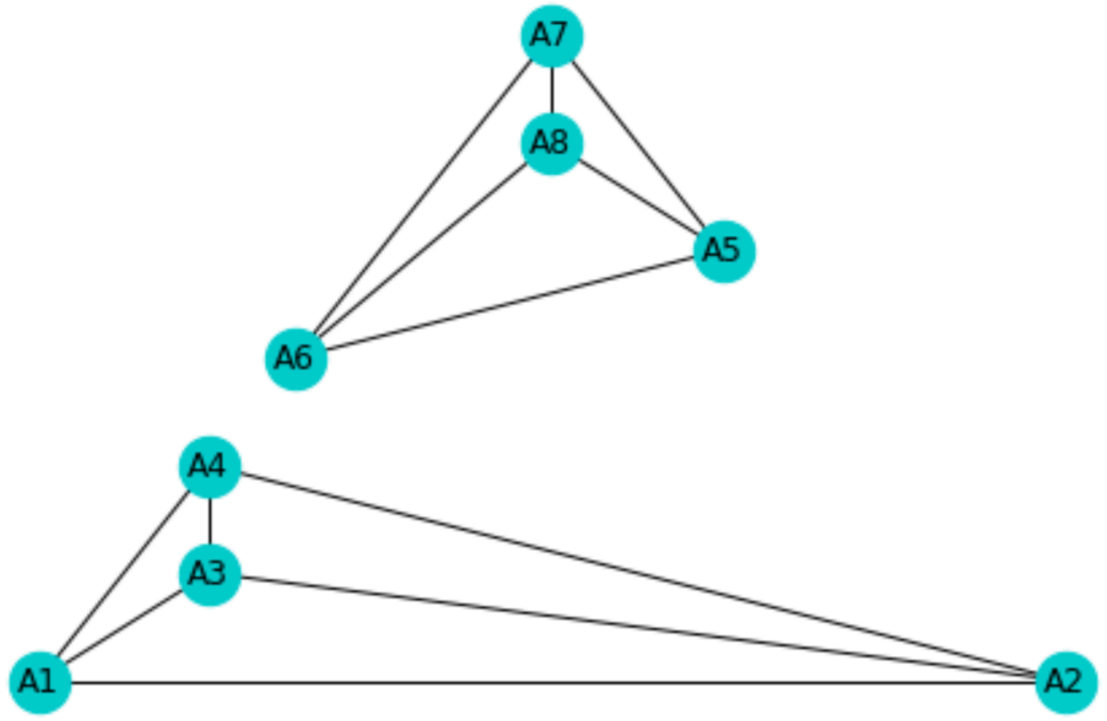


```
A3 = pd.read_csv('components.csv',index_col=0)
print(A3)
```

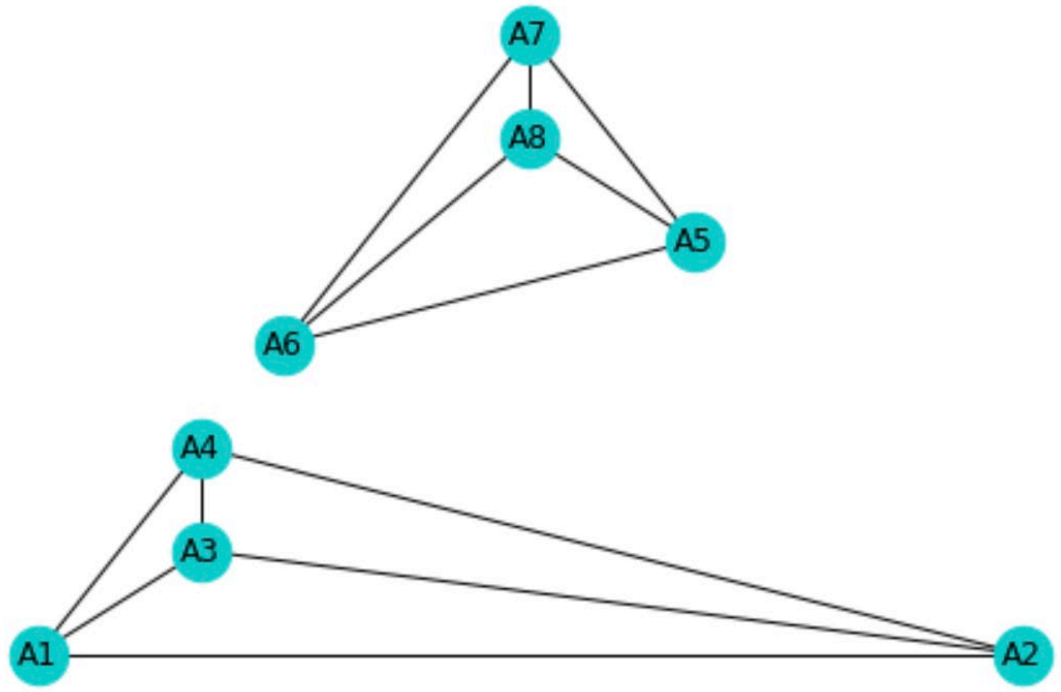
```
    A1  A2  A3  A4  A5  A6  A7  A8
A1   0   1   0   1   0   0   0   0
A2   0   0   1   1   0   0   0   0
A3   1   1   0   1   0   0   0   0
A4   1   0   0   0   0   0   0   0
A5   0   0   0   0   0   1   0   1
A6   0   0   0   0   0   0   1   1
A7   0   0   0   0   1   1   0   0
A8   0   0   0   0   0   1   1   0
```

Finding connected component
Create graph

```
G3 = nx.from_pandas_adjacency(A3)
nx.draw_planar(G3, with_labels=True, node_size=500, node_color='#00CCCC')
```



Finding connected component
Connected components



```
[x for x in nx.connected_components(G3)]
```

```
[{'A1', 'A2', 'A3', 'A4'}, {'A5', 'A6', 'A7', 'A8'}]
```

Lab

- Given the social network data, find the following answer
    - Who are the most popular?
    - Pick 2 individuals, find the shortest path between them
    - What does the shortest path mean?
    - Find how many components are there in the network

BIG DATA
EXPERIENCE

Thank you

Question?

BIG DATA
EXPERIENCE
CENTER

KMUTT g-able