# Graph Algorithms

**Asst. Prof. Karl Ezra Pilario**

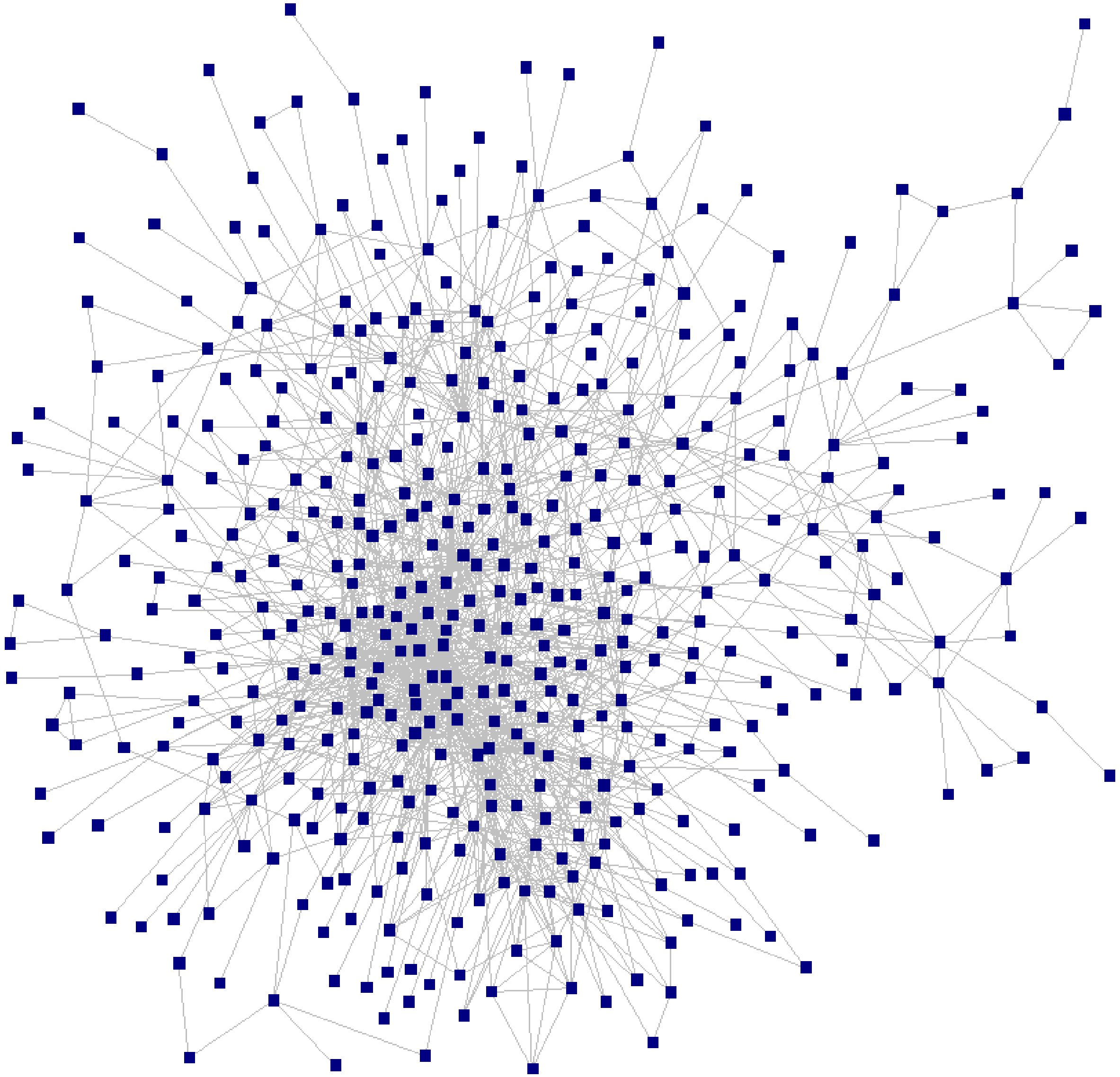
ChE 198

1s AY 2020-2021

# What is a graph?

A collection of nodes and edges.

| **Examples:** | Node | Edge |
|---|---|---|
| Social networks | Person | Friendship |
| Contagions | Person | Contact |
| Country map | City | Roads |
| Chemical plant | Equipment | Piping |
| Reaction network | Compound | Reaction |
| Power grid | Sources/loads | Power lines |



Source: http://www.orgnet.com/Erdos.html

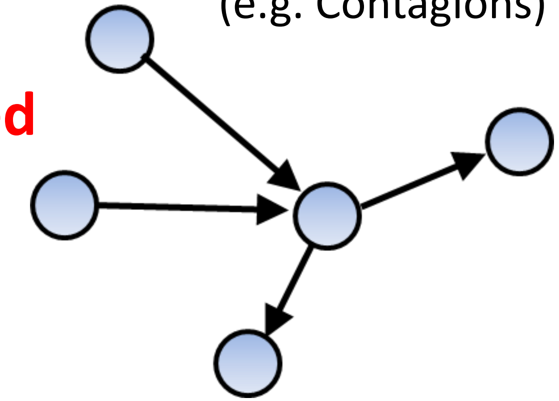
# Graph Terminologies

Different kinds of graphs:

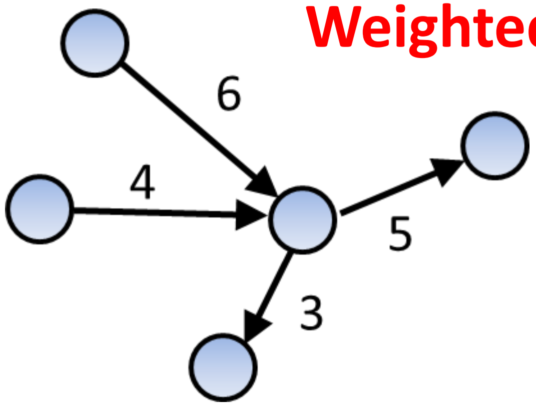
## Directed graph

(e.g. Contagions)

**Unweighted**



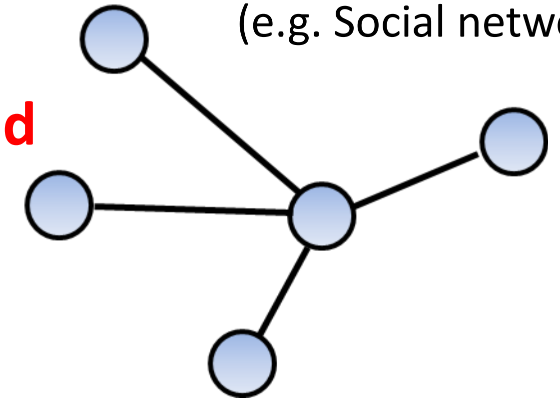
**Weighted**



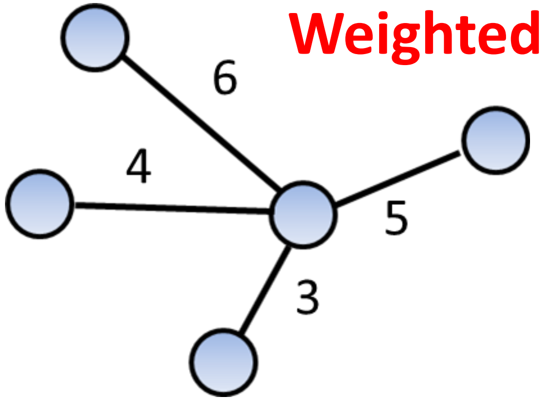

## Undirected graph

(e.g. Social networks)

**Unweighted**



**Weighted**

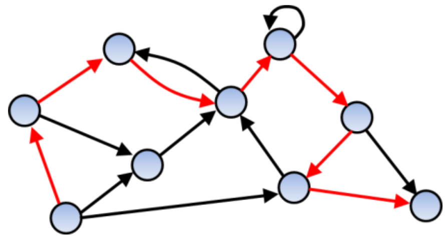


# Graph Terminologies

Other definitions:

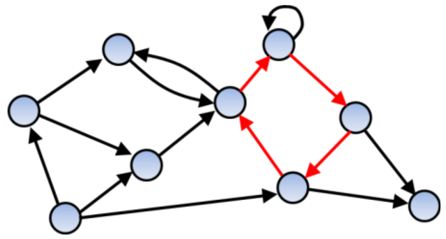
## Path

An ordered sequence of nodes connected by edges.



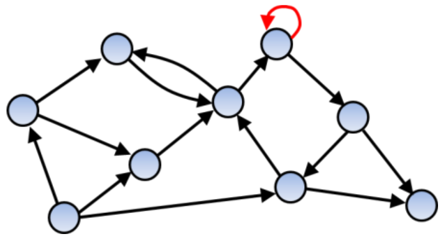
## Cycle

A path with the same start and end node.



## Self-loop

An edge connecting a node to itself.



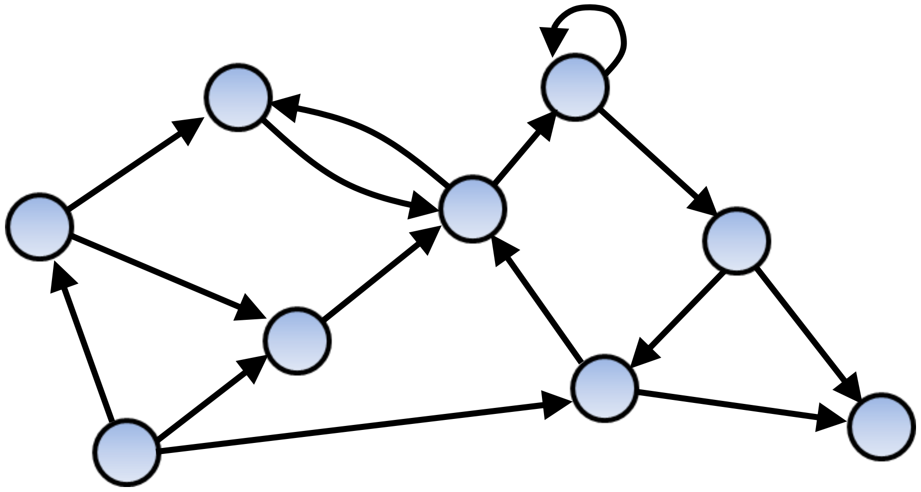
A self-loop is a *cycle of length 1*.

A path in which *all nodes are distinct*, except possibly the start and end nodes, is called a *simple path*.

A simple path with the same start and end node is a *simple cycle*.

Any graph with no self-loop is called a *simple graph*.

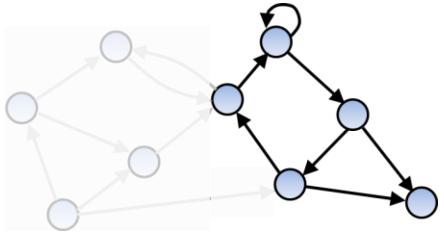
A digraph no cycles is called a *DAG (directed acyclic graph)*.



A directed graph (digraph)

## Subgraph

A graph containing only a subset of the edges and nodes of the original graph.

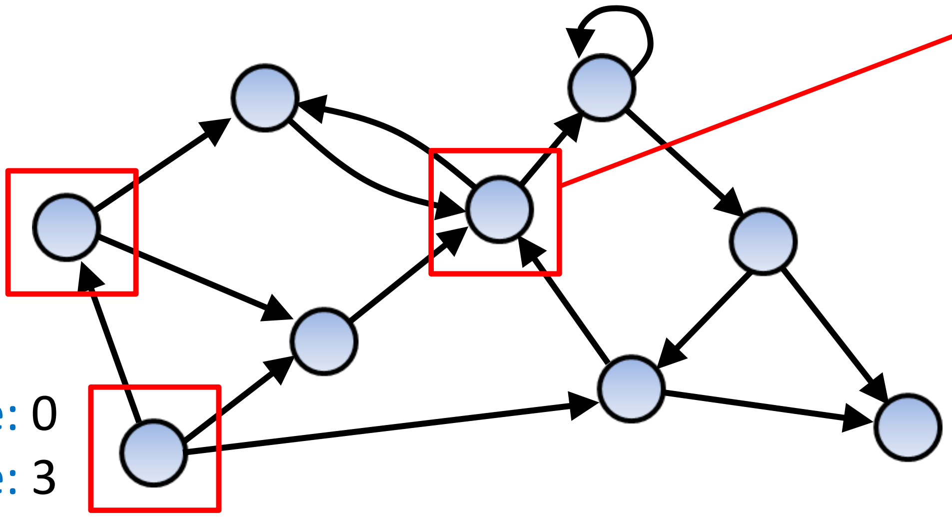


# Graph Terminologies

Other definitions:

A directed graph (digraph)

In-degree: 1
Out-degree: 2



In-degree: 0
Out-degree: 3

**In-degree: 3**
No. of edges entering it.

**Out-degree: 2**
No. of edges leaving it.

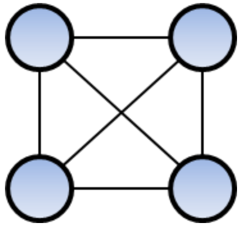
For an undirected graph, the in-degree and out-degree are the same. Hence, they are simply called *degree*.

## Complete graph

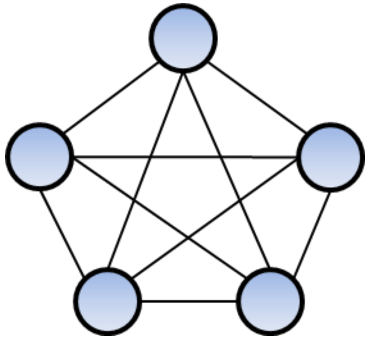
A graph where all possible edge connections between all nodes exist.

In a complete digraph, all the edges are bi-directional.

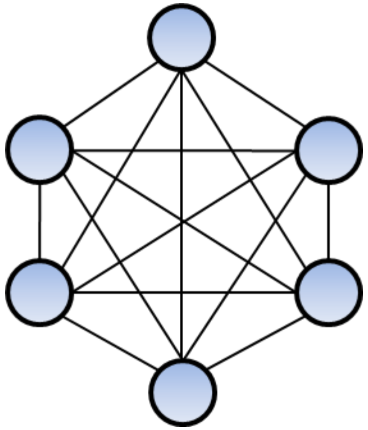
$$E = \binom{N}{2} = \frac{N(N-1)}{2}$$



4 nodes
6 edges



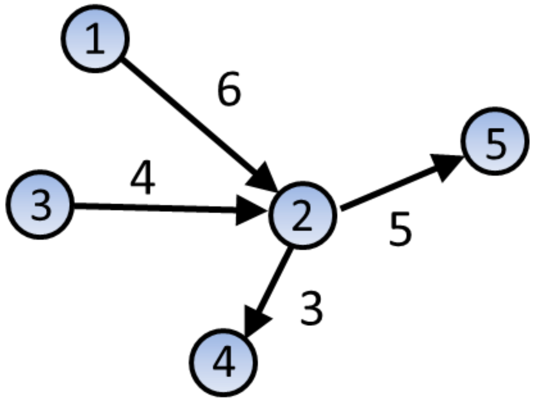
5 nodes
10 edges



6 nodes, 15 edges

# Graph Representations

Different ways to represent graphs on code:




### Source-Target-Weight Vectors

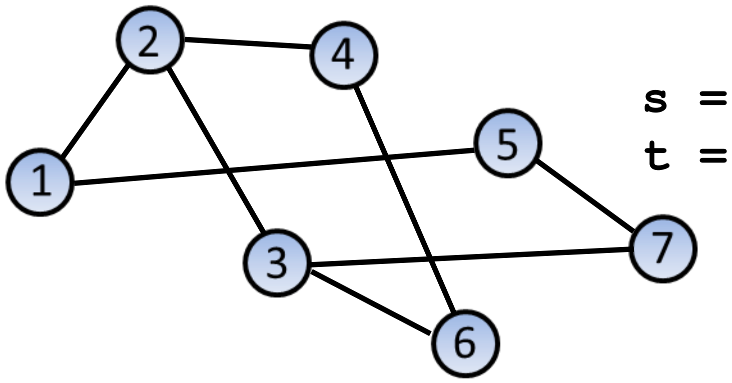
```
s = [1 3 2 2]
t = [2 2 4 5]
w = [6 4 3 5]
```

### Adjacency Matrix

$$
\begin{array}{c|ccccc}
 & N1 & N2 & N3 & N4 & N5 \\
\hline
N1 & 0 & 6 & 0 & 0 & 0 \\
N2 & 0 & 0 & 0 & 3 & 5 \\
N3 & 0 & 4 & 0 & 0 & 0 \\
N4 & 0 & 0 & 0 & 0 & 0 \\
N5 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

### Adjacency Lists

```
N1 [2]
N2 [4 5]
N3 [2]
N4 []
N5 []
```




```
s = [1 1 2 2 3 3 4 5]
t = [2 5 3 4 6 7 6 7]
```

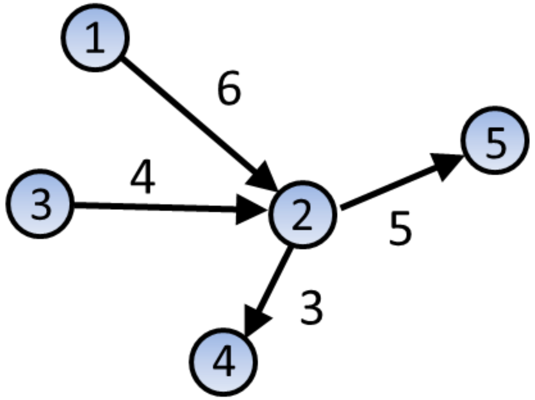
$$
\begin{array}{c|ccccccc}
 & N1 & N2 & N3 & N4 & N5 & N6 & N7 \\
\hline
N1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
N2 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
N3 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
N4 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
N5 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
N6 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
N7 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{array}
$$

```
N1 [2 5]
N2 [1 3 4]
N3 [2 7 6]
N4 [2 6]
N5 [1 7]
N6 [3 4]
N7 [3 5]
```

# Graph Representations

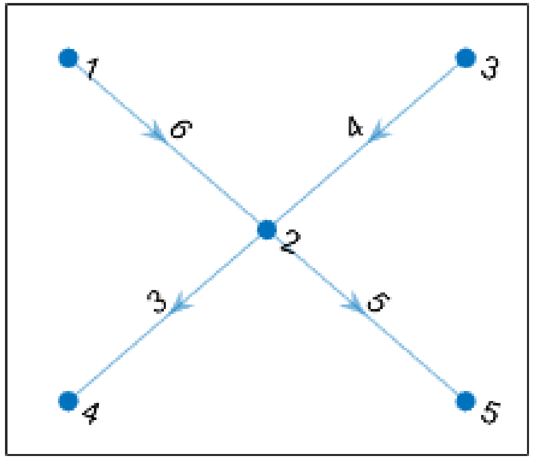
We can build graphs in MATLAB.

Enter the following codes:

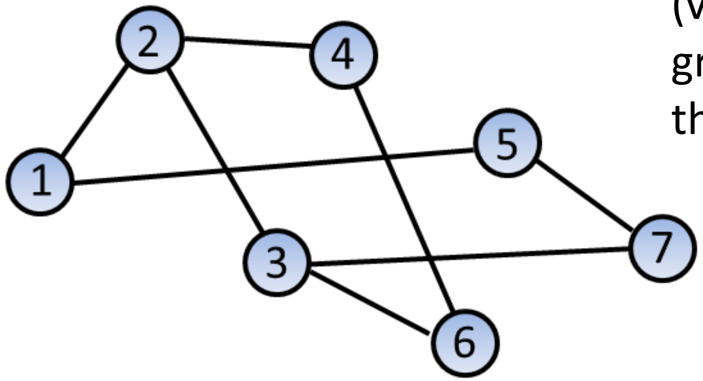



```
>> s = [1 3 2 2];
>> t = [2 2 4 5];
>> w = [6 4 3 5];
>> G = digraph(s,t,w);
>> plot(G,'EdgeLabel',G.Edges.Weight);
```

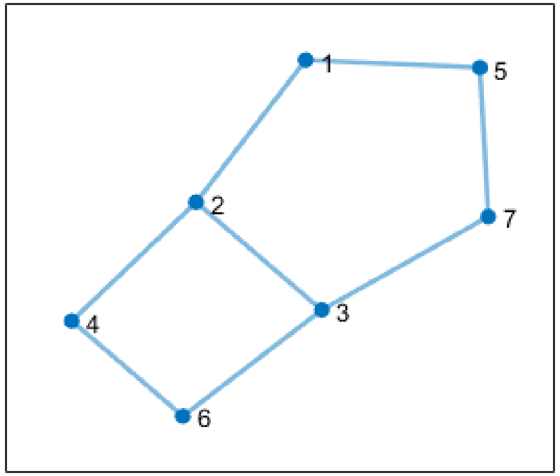
**MATLAB output:**
(verify that this graph is the same as the one on the left)



```
>> s = [1 1 2 2 3 3 4 5];
>> t = [2 5 3 4 6 7 6 7];
>> G = graph(s,t); plot(G);
```




**MATLAB output:**
(verify that this graph is the same as the one on the left)



# Graph Algorithms

I. Reachability (Transitive Closure)
II. Topological Sorting
III. Shortest Path
IV. Max Flow
V. Minimum Spanning Tree

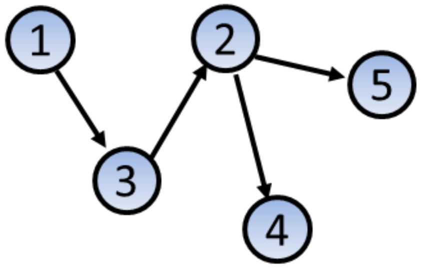
# Reachability

Reachability: Is Node ‘T’ reachable from Node ‘S’?

**Definition: Transitive closure.**

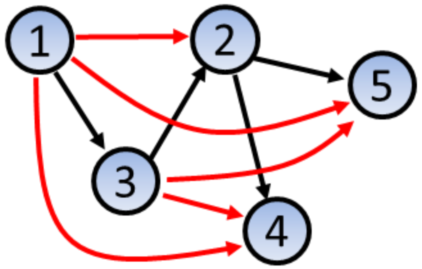
The *transitive closure* of a digraph *G* is another digraph with the same set of nodes, but where an edge from any S to T exists *if and only if* T is *reachable* from S in *G*.

Original Graph, G




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

Transitive Closure of G




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

**Related PSE problems:**

1. In a water distribution network, if a pumping station is down, which households are affected?
2. In a chemical plant, if a certain pipe is blocked, how will its effects propagate throughout the plant?
3. If a lake has been contaminated, which other sites in nature are affected?

**Solution:**

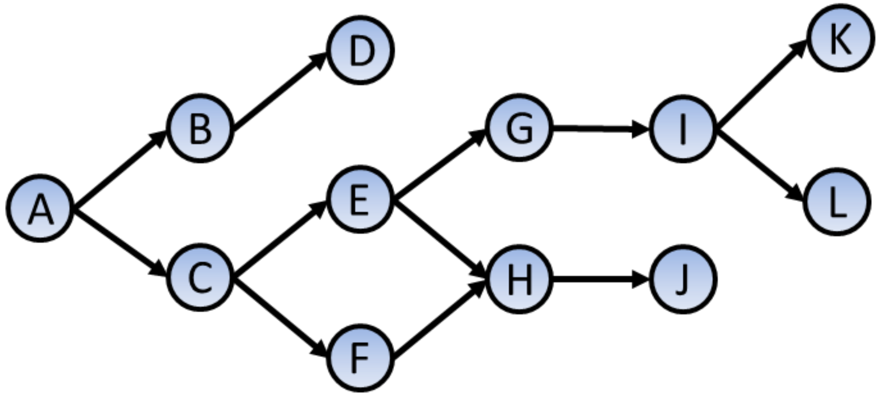
```
>> s = [1 2 2 3];
>> t = [3 4 5 2];
>> G = digraph(s,t);
>> H = transclosure(G);
>> full(adjacency(H))
```

In this new adjacency matrix, an entry of ‘**1**’ in *row S, column T* means that Node *T* is reachable from Node *S*.

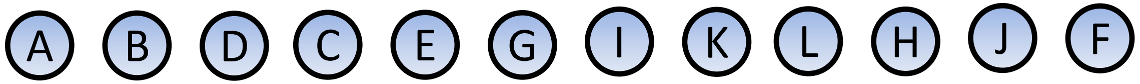
# Reachability

How can we traverse / propagate the entire graph from any starting Node *S*?

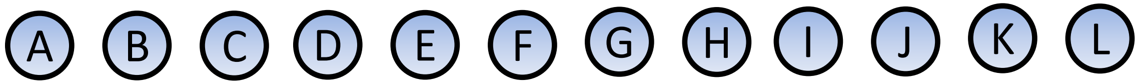
```
>> s = [1 1 2 3 3 5 5 6 7 8 9 9];
>> t = [2 3 4 5 6 7 8 8 9 10 11 12];
>> names = {'A','B','C','D','E','F',...
           'G','H','I','J','K','L'};
>> G = digraph(s,t,[],names)
```




- **Depth-first search (DFS)**
  Go deep, move out, go deep again.
  ```
  >> visit = dfsearch(G,'A')'
  ```
- **Breadth-first search (BFS)**
  View all options, go to next level, view all again.
  ```
  >> visit = bfsearch(G,'A')'
  ```




Order of nodes visited by DFS / BFS starting from Node A.

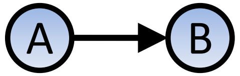


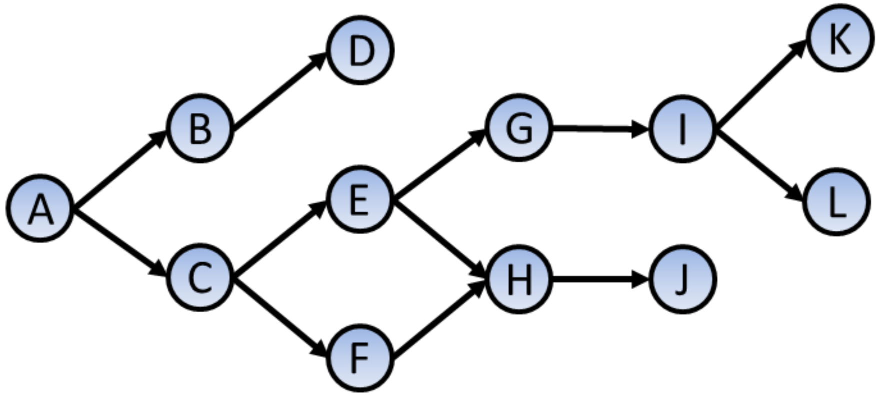
# Topological Sorting

Given a graph of dependencies, how do we sort the nodes from left to right without violating their dependencies?

**Example:**

If an edge denotes *dependency*, then A must be performed before B.




```
>> s = [1 1 2 3 3 5 5 6 7 8 9 9];
>> t = [2 3 4 5 6 7 8 8 9 10 11 12];
>> names = {'A','B','C','D','E','F',...
           'G','H','I','J','K','L'};
>> G = digraph(s,t,[],names)
```

**Related PSE problems:**

1. Given a set of tasks, where some must be performed before others, in which order should we perform them to minimize completion time?
2. In a batch process plant, how do we schedule the operation to complete all product orders in the shortest time possible?
3. [Not PSE] Given a curriculum checklist for BS ChE, find a possible ordering of subjects to be taken such that all pre-requisites are followed.

**Solution:**

```
>> N = toposort(G);
>> cell2mat(G.Nodes.Name(N,:)')

'ACFEHJGILKBD'
```

```
>> N = toposort(G,'Order','stable');
>> cell2mat(G.Nodes.Name(N,:)')

'ABCDEFGHIJKL'
```

If “stable”, smaller node labels go first.

**Note:** `Toposort` does not require you to supply a starting node, whereas `dfsearch` does.

# Graph Algorithms

I. Reachability (Transitive Closure)
II. Topological Sorting
**III. Shortest Path**
IV. Max Flow
V. Minimum Spanning Tree

# Shortest Path

Which path from Node S to T has the minimum sum of weights?

**Example:**

```
>> s = [1 1 1 2 2 2 3 3 4 5 6 7 7 8 8 9 9];
>> t = [3 5 8 4 5 9 2 9 7 3 4 1 6 4 7 6 7];
>> w = [7 20 13 12 7 6 4 8 3 5 5 6 5 7 12 10 10];
>> G = digraph(s,t,w);
>> p = plot(G,'EdgeLabel',G.Edges.Weight);
```

To find the shortest path from Node 1 to Node 6:

```
>> [path,d] = shortestpath(G,1,6);
>> highlight(p,path,'EdgeColor','g');
>> fprintf('Path length: %d\n',d);
```

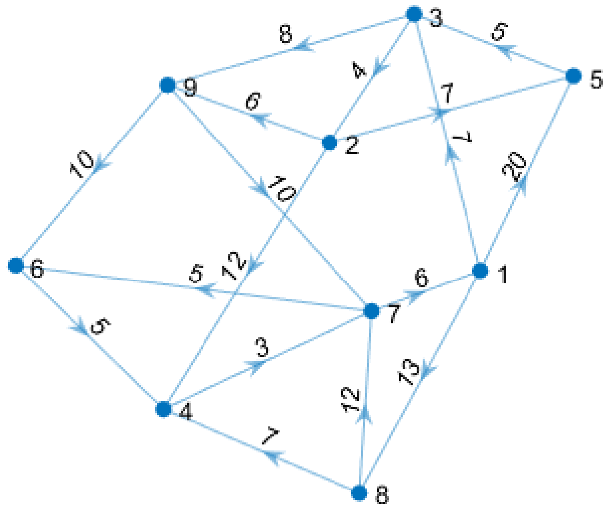
Algorithms inside MATLAB's `shortestpath`:

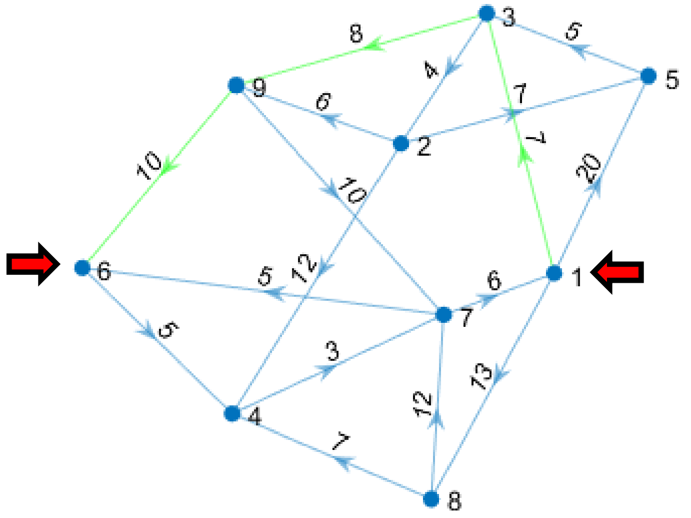
1. BFS (for unweighted)
2. Dijkstra (for digraph with all weights positive)
3. Bellman-Ford (for digraph with negative weights)

**Related PSE problems:**

1. What is the fastest route to move goods geographically from source to demand?
2. Which path of reaction steps should be taken so that the net energy needed to create a target compound is minimum?
3. Which value chain would give a maximum net profit from a given raw material?

**Answer:** `Path length: 25`



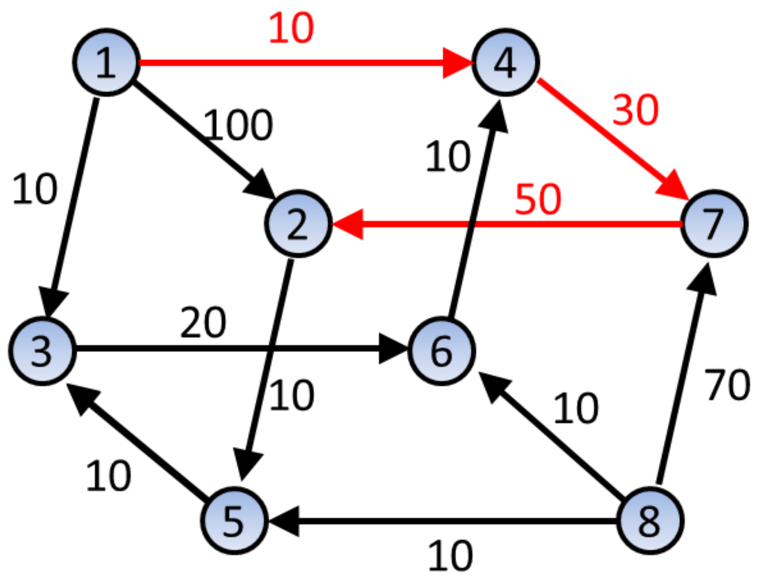



# Shortest Path

Which path from Node S to T has the minimum sum of weights?

**Example:**

```
>> s = [1 1 1 2 5 3 6 4 7 8 8 8];
>> t = [2 3 4 5 3 6 4 7 2 6 7 5];
>> w = [100 10 10 10 10 20 10 30 50 10 70 10];
>> G = digraph(s,t,w);
>> plot(G,'EdgeLabel',G.Edges.Weight)
```




To find the shortest path between **any pair of nodes**:

```
>> D = distances(G);
```

d = 8×8

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | $\boxed{90}$ | 10 | 10 | 100 | 30 | 40 | Inf |
| Inf | 0 | 20 | 50 | 10 | 40 | 80 | Inf |
| Inf | 110 | 0 | 30 | 120 | 20 | 60 | Inf |
| Inf | 80 | 100 | 0 | 90 | 120 | 30 | Inf |
| Inf | 120 | 10 | 40 | 0 | 30 | 70 | Inf |
| Inf | 90 | 110 | 10 | 100 | 0 | 40 | Inf |
| Inf | 50 | 70 | 100 | 60 | 90 | 0 | Inf |
| Inf | 100 | 20 | 20 | 10 | 10 | 50 | 0 |

D(*i*, *j*) is the length of the shortest path from node *i* to node *j*.

# Graph Algorithms

I. Reachability (Transitive Closure)
II. Topological Sorting
III. Shortest Path
**IV. Max Flow**
V. Minimum Spanning Tree

# Max Flow

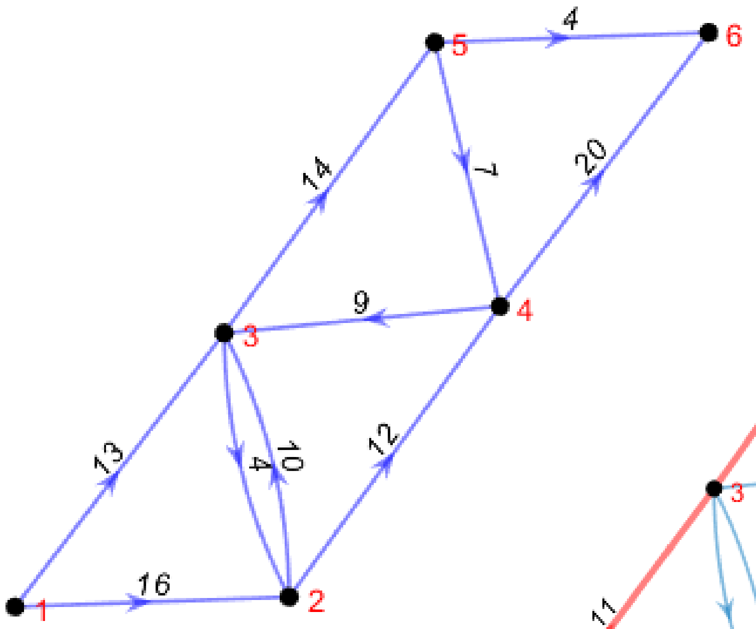
Given a *source* node S, a *sink* node T, and the flow capacity in each edge, find the maximum flow that can be pushed from S to T.

**Related PSE problems:**

1. Find the maximum production capacity of a plant, given the capacity of each pipe and equipment.
2. Find the maximum no. of goods that can flow from the supply to the demand side, given that only a limited amount of goods can flow on certain roads.

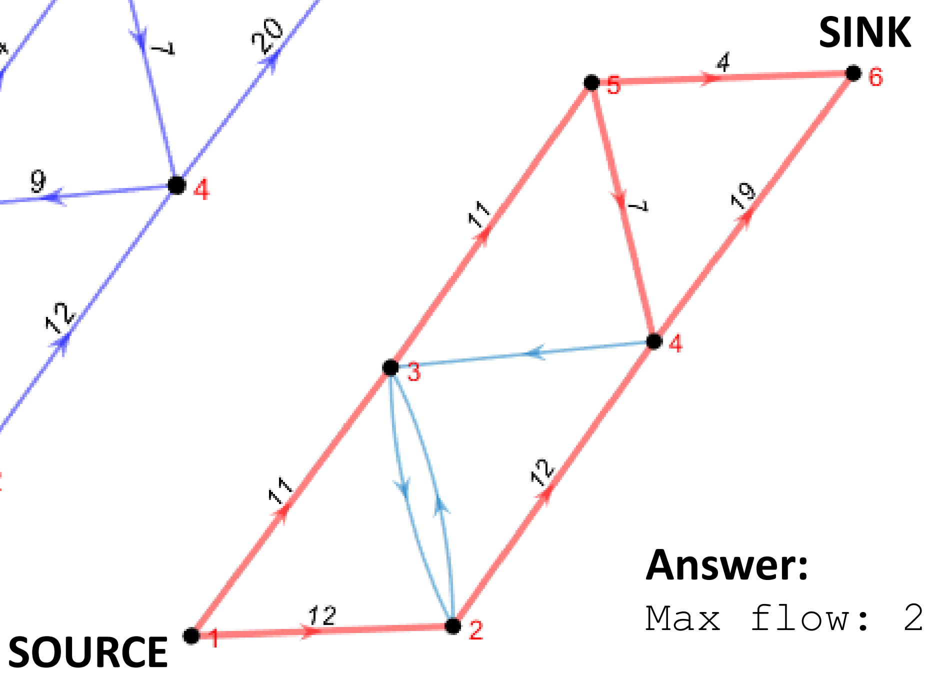
**Example:**

```
>> s = 1; t = 6;
>> cap = [ 0 16 13 0 0 0;  0 0 10 12 0 0;
           0 4 0 0 14 0;   0 0 9 0 0 20;
           0 0 0 7 0 4;    0 0 0 0 0 0];
>> G = digraph(cap);
>> H = plot(G,'EdgeLabel',G.Edges.Weight)
```




To find the maximum flow (`mf`) from S to T and the resulting flow graph (`GF`):

```
>> [mf,GF] = maxflow(G,s,t);
>> H.EdgeLabel = {};
>> highlight(H,GF,'EdgeColor','r','LineWidth',2);
>> st = GF.Edges.EndNodes;
>> labeledge(H,st(:,1),st(:,2),GF.Edges.Weight);
>> fprintf('Max flow: %d\n',mf);
```




**Answer:**
`Max flow: 23`

# Graph Algorithms

I. Reachability (Transitive Closure)
II. Topological Sorting
III. Shortest Path
IV. Max Flow
**V. Minimum Spanning Tree**

# Minimum Spanning Tree

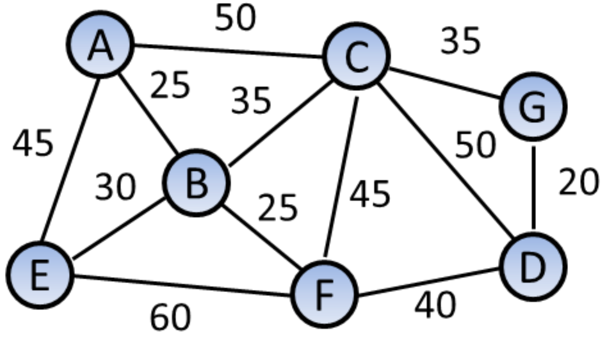
Given an undirected graph, find a subset of the edges such that the remaining graph is still connected but the total edge weight is minimum.

**Example:**

```
>> adj = [0 25 50 0 45 0 0;25 0 35 0 30 25 0;
          50 35 0 50 0 45 35; 0 0 50 0 0 40 20;
          45 30 0 0 0 60 0; 0 25 45 40 60 0 0;
          0 0 35 20 0 0 0];
>> G = digraph(adj);
>> H = plot(G,'EdgeLabel',G.Edges.Weight)
```

To find the minimum spanning tree, T:

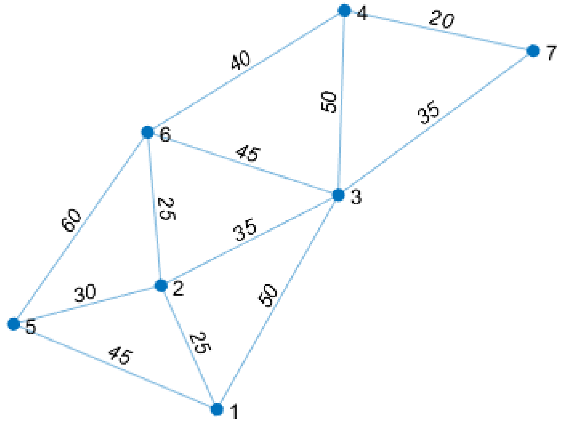
```
>> T = minspantree(G);
>> highlight(H,T);
```




## Related PSE problems:

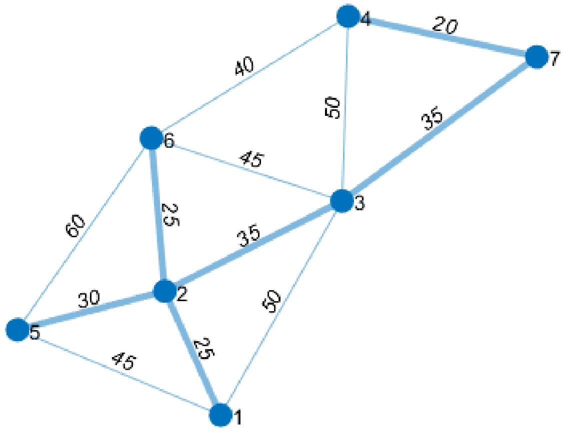
1. Given the geographical locations of power stations and loads, connect the entire power grid with the minimum total length of transmission cables.
2. Select only a few water lines to maintain during a water shortage, while ensuring that all households still have access to water.

**MATLAB output:**




**Minimum spanning tree:**

Total weight: 170



# Summary

- There are optimization problems in PSE that can be represented as graphs. Hence, graph algorithms are needed to solve them.

- You now have learned how to use new MATLAB built-in functions such as:

| | | |
|---|---|---|
| `graph` | `toposort` | `highlight` |
| `digraph` | `transclosure` | `adjacency` |
| `dfsearch` | `shortestpath` | `maxflow` |
| `bfsearch` | `distances` | `minspantree` |