Introduction to Graph Mining

Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr

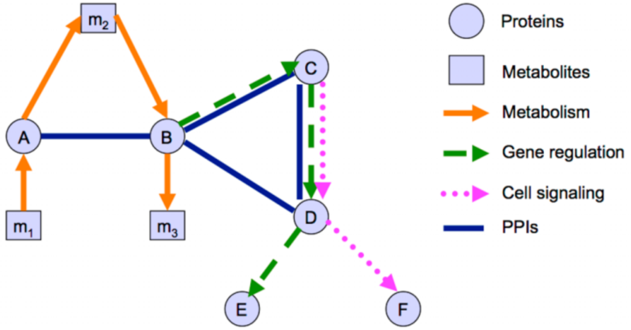
네트워크 과학 연구실 NETWORK SCIENCE LAB

가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA

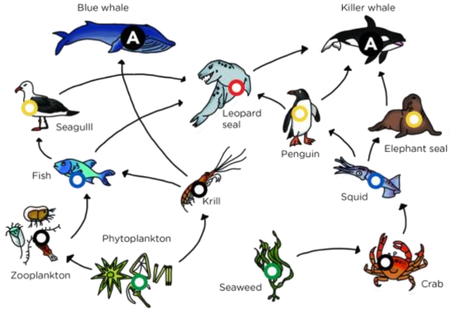
Contents

- The overview of graph mining
- Graph definition
- Terminology
- Types of graphs
- Graph applications in real life
- Sample code: Creating a simple graph using NetworkX

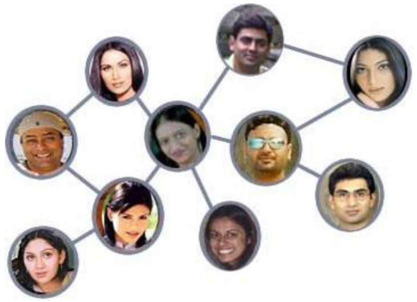
Graphs are everywhere



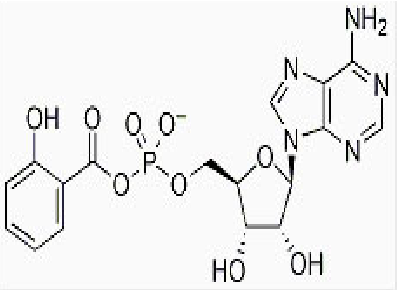
Biological network



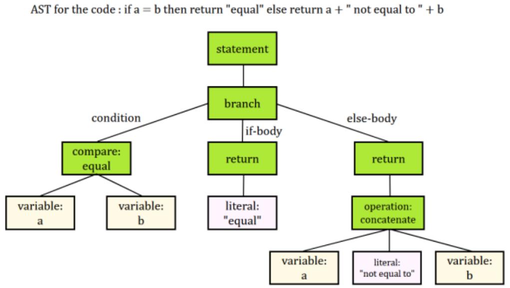
Ecological network



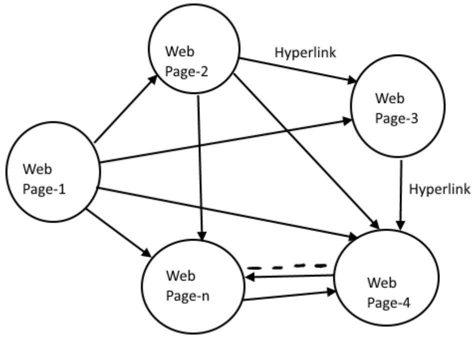
Social media



Chemical network



Program flow

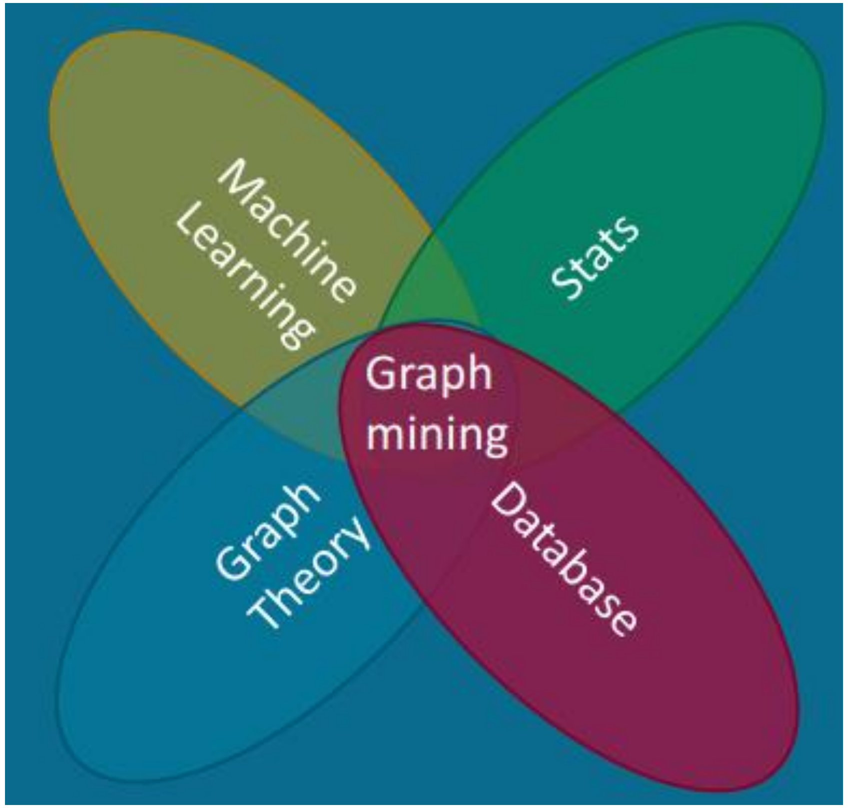


Web graph

Why Graphs? Why now?

- Describe complex data with a simple structure
  - Nature, social, concepts, roads, circuits …
- Same representation for many disciplines
  - Computer science, biology, physics, economics, ...
- Availability of (BIG) data
  - Large networks are now available and require complex algorithms.
  - Networks are evolving over time (e.g., new users/friends in Facebook).
- Usefulness:
  - They reveal user behaviours.
  - They are valuable (Facebook, Twitter,... All of them based on graphs).

- Graph mining is the process of discovering, retrieving and analyzing non trivial patterns in graph shaped data.

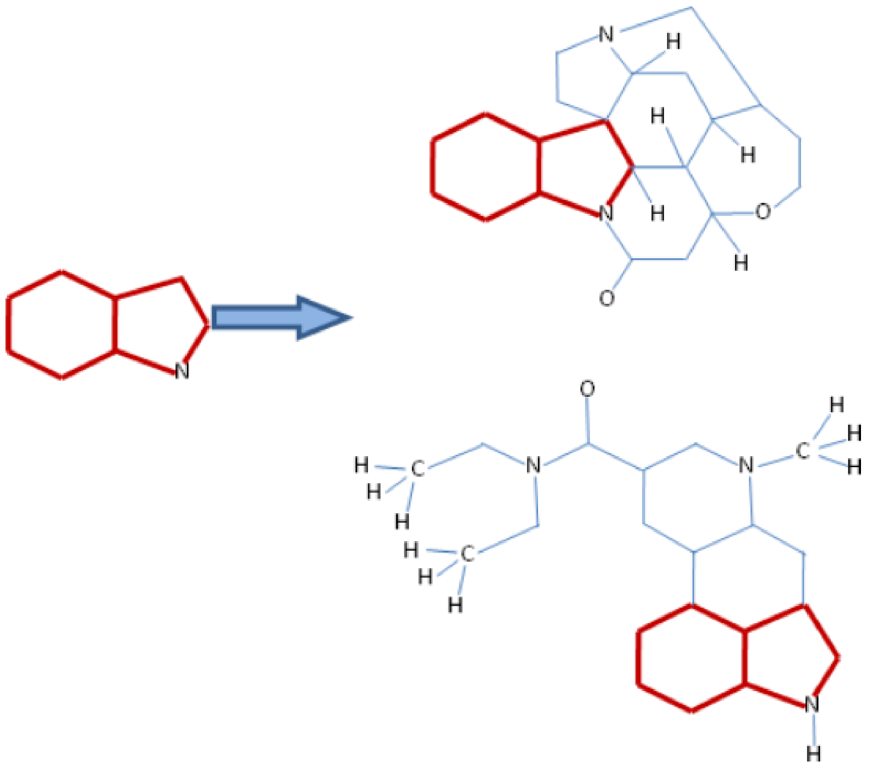


What can we do with graph mining?

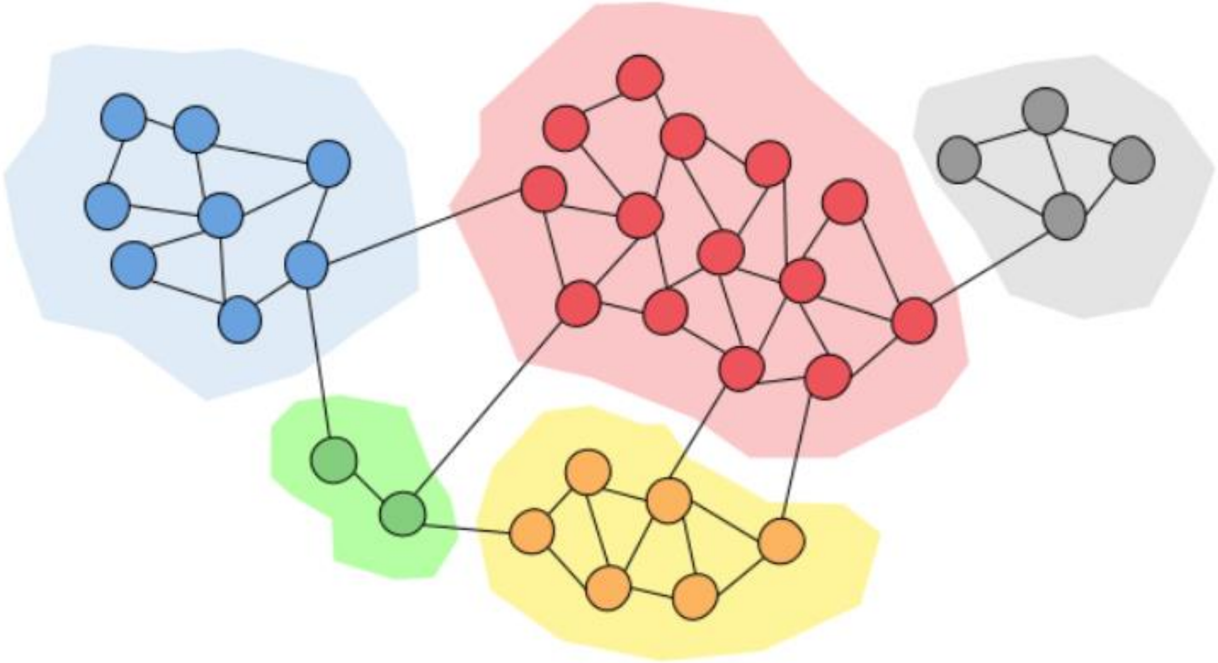
- Compressing graphs without losing information
- Finding complex structures fast
- Recognizing communities and social patterns
- Study the propagation of viruses
- Predicting if two people will become friends
- Understanding what are the important nodes
- Showing how the network will evolve
- Helping the visualization of complex structures
- Finding roles, positive and negative influence prediction

Practical applications of graph mining

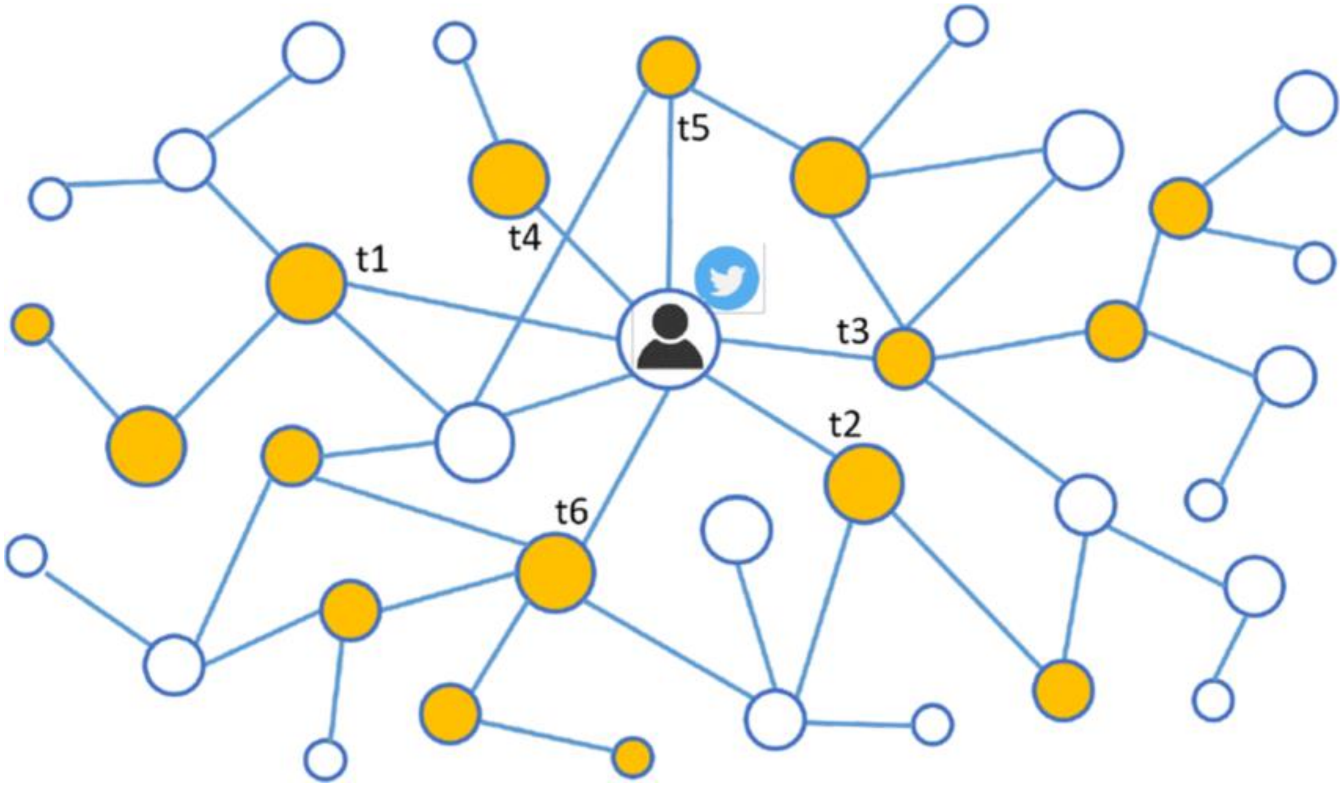
- Finding substructures



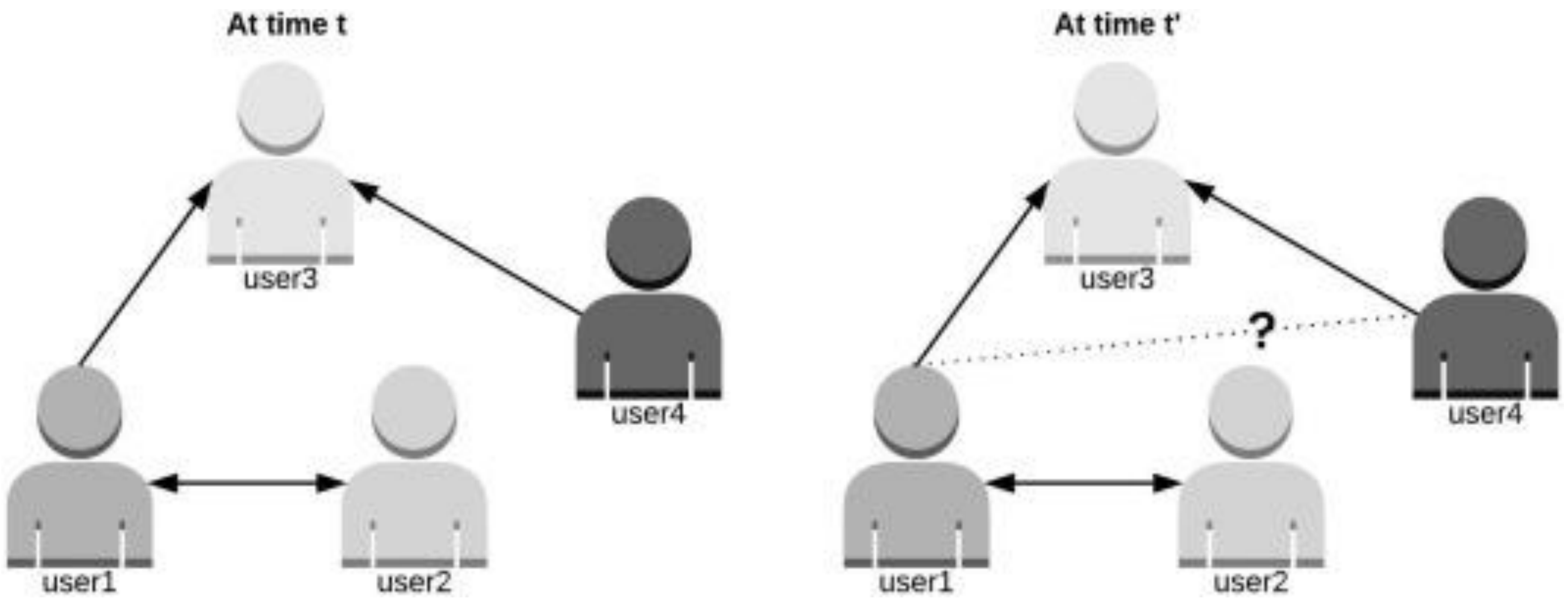
- Community detection in social networks



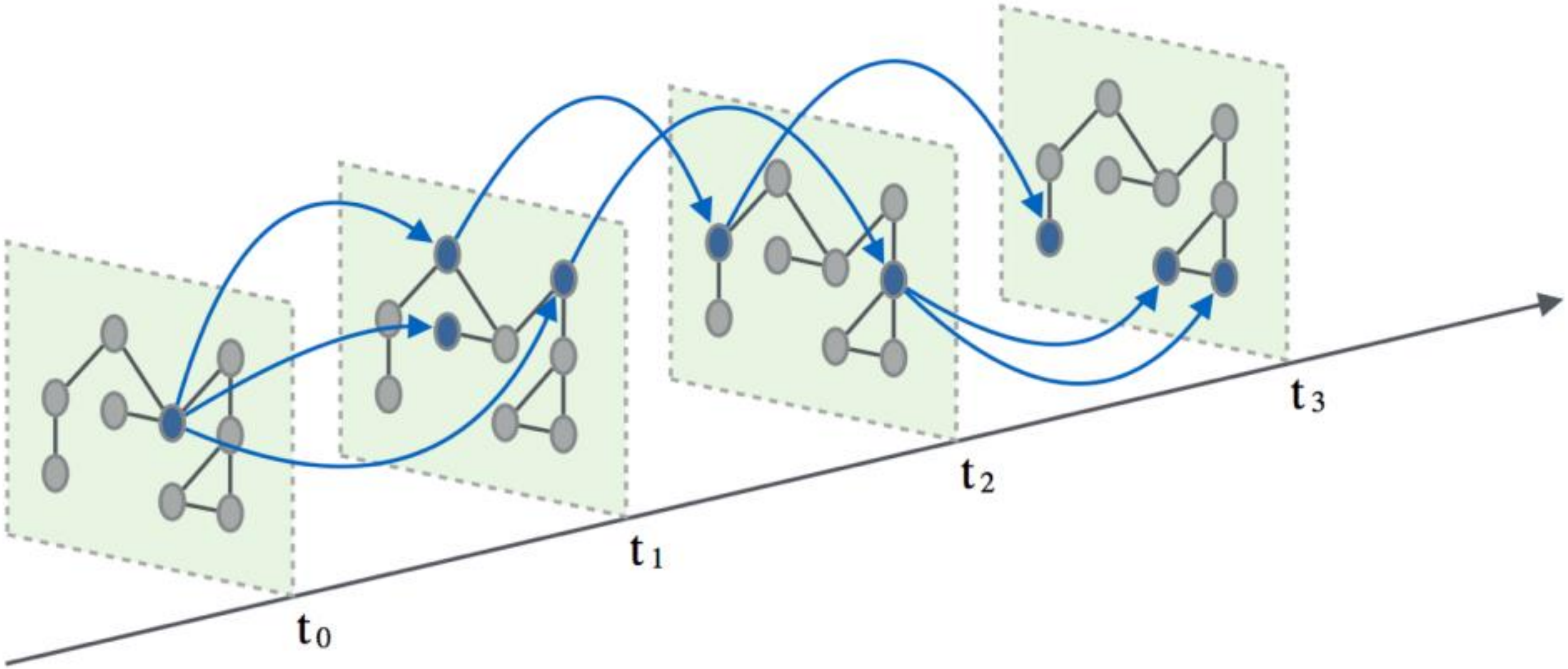
- Influence propagation



- Link prediction

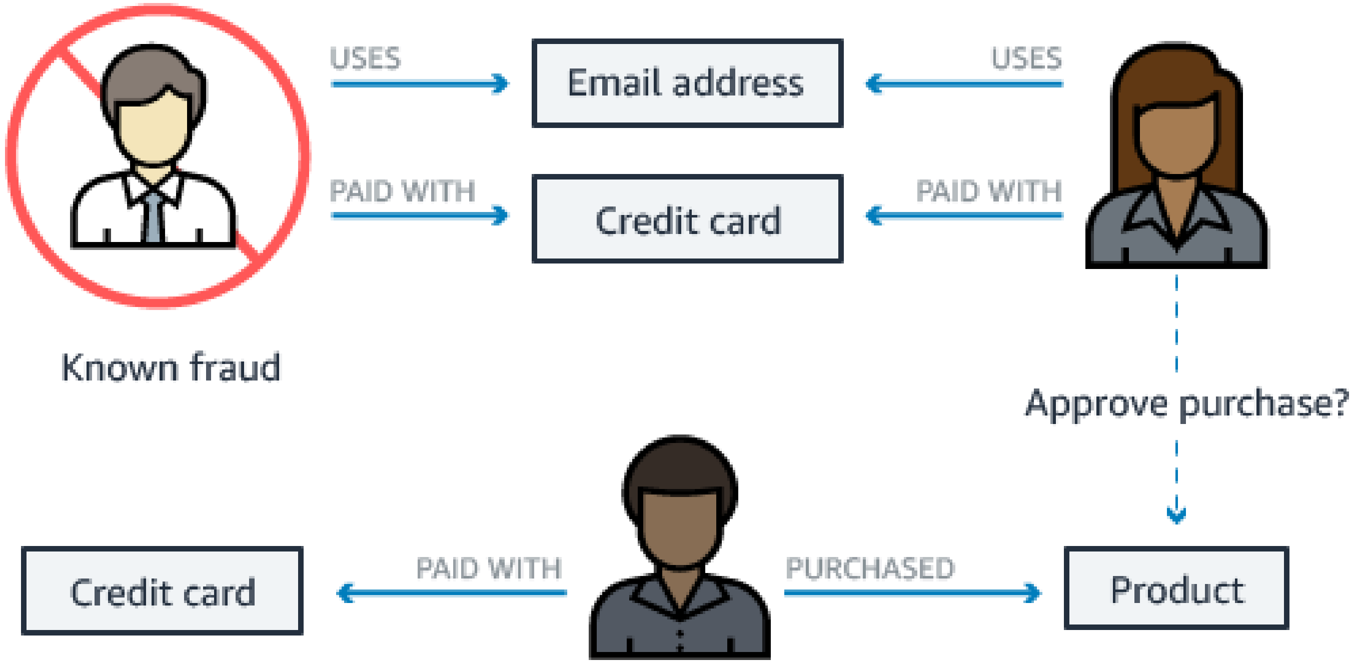


- Graph evolution



Practical applications of graph mining

- Detecting frauds

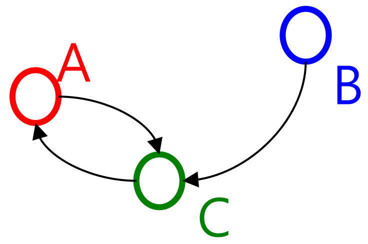


Graph definition

A graph is a pair: G = (V, E):

- A set of nodes, also known as nodes: $V = \{v_1, v_2, \ldots, v_n\}$
- A set of edges $E = \{e_1, e_2, \ldots, e_m\}$
  - Each edge e_i is a pair of nodes (v_j, v_k)
  - An edge "connects" the nodes

Graphs can be *directed* or *undirected*



V = { A, B, C }
E = {(B ,C), (A, C), (C , A)}

Several graphs in real life

For each example, what are the nodes and what are the edges?

- Web pages with links
- Facebook friends
- Road maps
- Airline routes
- Family trees

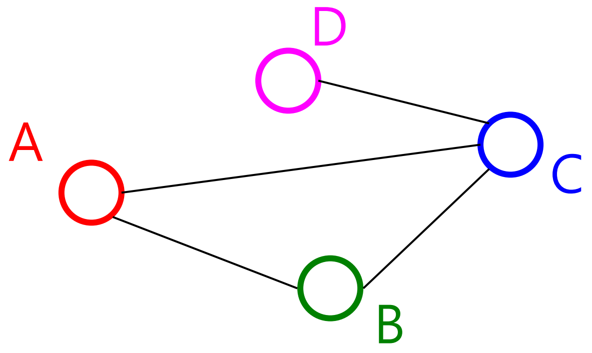
- To make formulating graphs easy and standard, we have a lot of *standard terminology* for graphs

Terminology

- Network = Graph
- Nodes = Vertices = Actors = Entities
- Links = Edges = Relations
- Clusters = Communities

Undirected Graphs

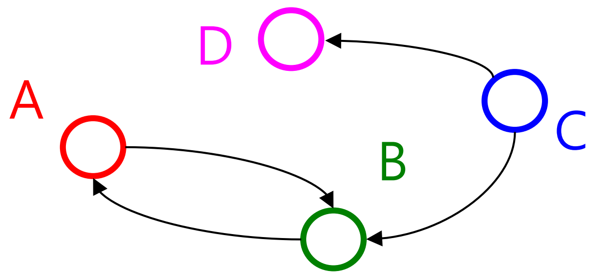
- In undirected graphs, edges have no specific direction
  - Edges are always "two-way“
  - Thus, (u, v) ∊ E implies (v, u) ∊ E.



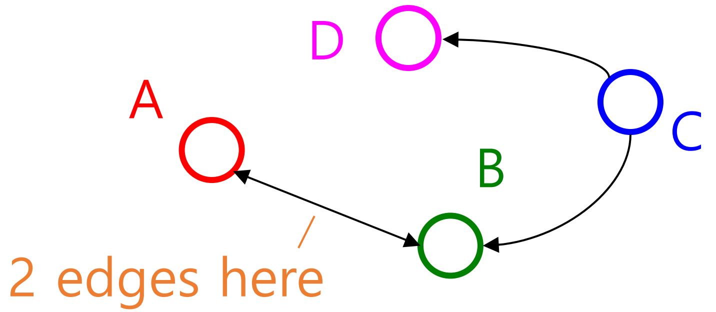
- Degree of a vertex: number of edges containing that vertex

Directed Graphs

In directed graphs (or digraphs), edges have direction



or



Thus, (u, v) ∊ E does not imply (v, u) ∊ E.

Let (u, v) $\in$ E mean u $\rightarrow$ v

- Call u the source and v the destination
- In-Degree of a vertex: number of in-bound edges (edges where the vertex is the destination)
- Out-Degree of a vertex: number of out-bound edges (edges where the vertex is the source)

Self-Edges, Connectedness

- A self-edge a.k.a. a loop edge is of the form (u, u)

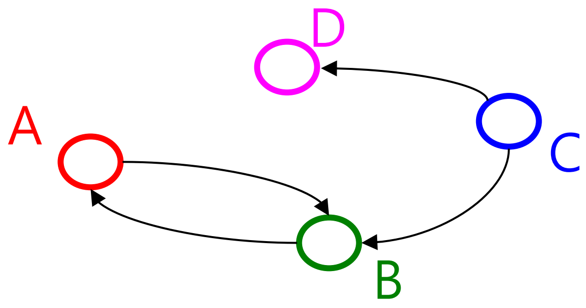
- The use/algorithm usually dictates if a graph has
  - No self edges
  - Some self edges
  - All self edges

- A node can have a(n) degree / in-degree / out-degree of zero
- A graph does not have to be connected
  - Even if every node has non-zero degree

More Notation

For a graph G = (V, E):

- |V| is the number of vertices
- |E| is the number of edges
  - Minimum?
  - Maximum for undirected?
  - Maximum for directed?



$V = \{A, B, C, D\}$

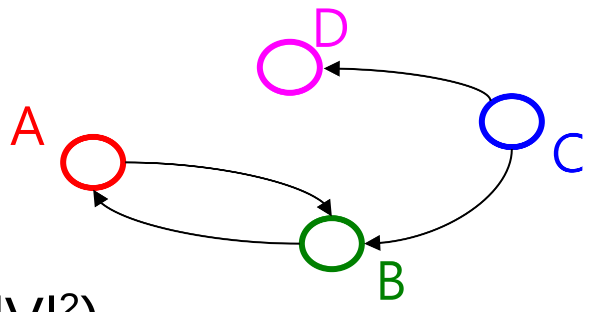
$E = \{(C, B), (A, B), (B, A), (C, D)\}$

If (u, v) ∊ E , then v is a neighbor of u (i.e., v is adjacent to u)

- Order matters for directed edges:
  u is not adjacent to v unless (v, u) ∈ E

For a graph G = (V, E):

- $|V|$ is the number of vertices
- $|E|$ is the number of edges
  - Minimum? 0
  - Maximum for undirected? $|V||V+1|/2 \in O(|V|^2)$
  - Maximum for directed? $|V|^2 \in O(|V|^2)$



If (u, v) ∈ E , then v is a neighbor of u (i.e., v is adjacent to u)

- Order matters for directed edges:
  u is not adjacent to v unless (v, u) ∈ E

The degree of nodes

- The degree of a node is the number of edges incident to that node

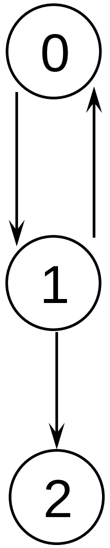
- For directed graph:
  - The in-degree of a vertex *v* is the number of edges that have *v* as the head
  - The out-degree of a vertex *v* is the number of edges that have *v* as the tail
  - If d_i is the degree of a vertex *i* in *G* with *n* vertices and *e* edges, the number of edges is:

$$e = \left(\sum_{i=0}^{n-1} d_i\right) / 2$$

Directed graph

directed graph

in-degree
out-degree



in:1, out: 1

in: 1, out: 2

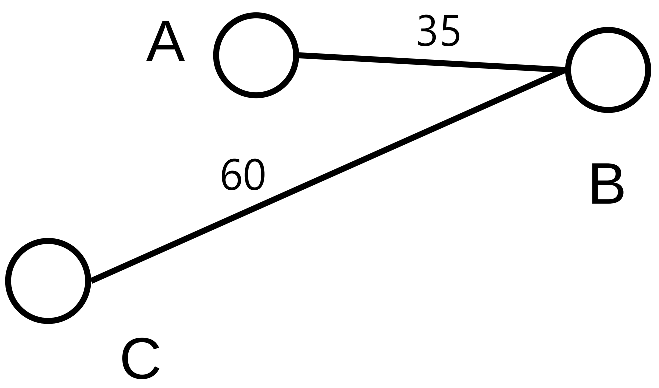
in: 1, out: 0

G3

Weighted Graphs

In a weighted graph, each edge has a weight or cost:

- Typically numeric (nits, decimals, doubles, etc.)
- Orthogonal to whether graph is directed
- Some graphs allow negative weights, many do not



Examples

What, if anything, might weights represent for each of these?

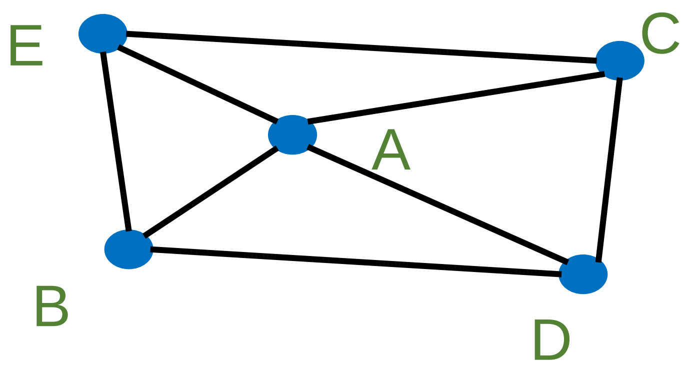
Do negative weights make sense?

- Web pages with links
- Facebook friends
- Road maps
- Airline routes
- Family trees
- Course pre-requisites

Paths and Cycles

We say "a path exists from v_0 to v_n" if there is a list of vertices $[v_0, v_1, \ldots, v_n]$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i < n$.

A cycle is a path that begins and ends at the same node ($v_0 == v_n$)



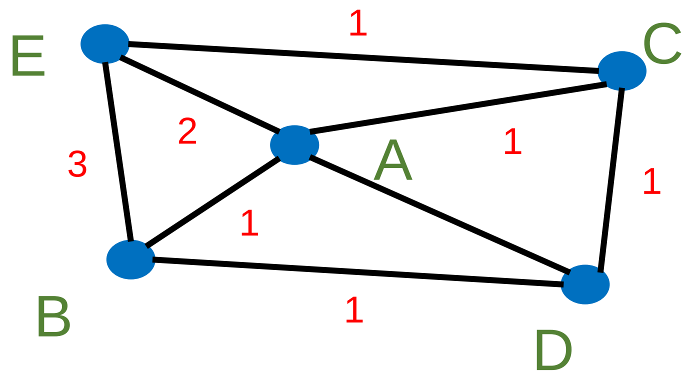
Example path (that also happens to be a cycle):

[E, B, D, C, A, E]

Path Length and Cost

- Path length: Number of edges in a path
- Path cost: Sum of the weights of each edge

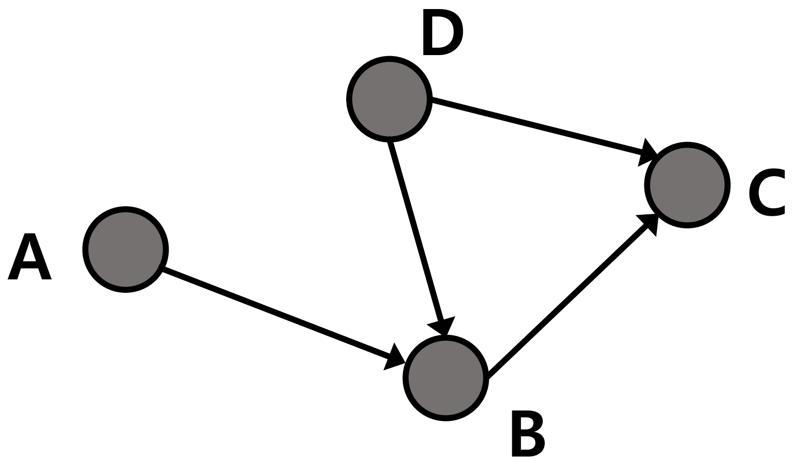
- Example:
  - Path: [E, B, D]



length(**P**) = 2
cost(**P**) = 4

Length is sometimes called "unweighted cost"

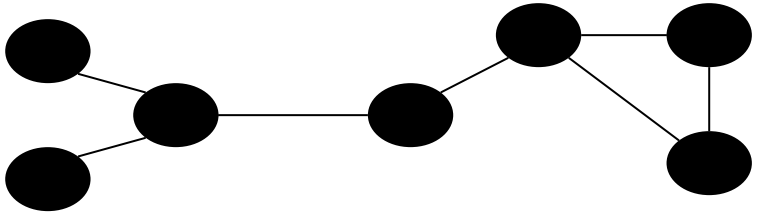
Example:



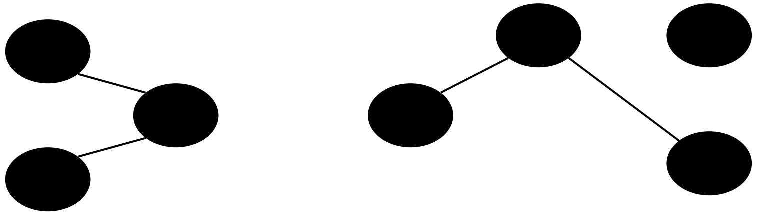
- Is there a path from A to D? No
- Does the graph contain any cycles? No

Undirected Graph Connectivity

An undirected graph is connected if for all pairs of vertices *u≠v*, there exists a *path* from *u* to *v*

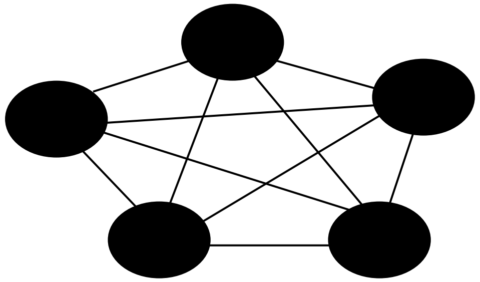


Connected graph



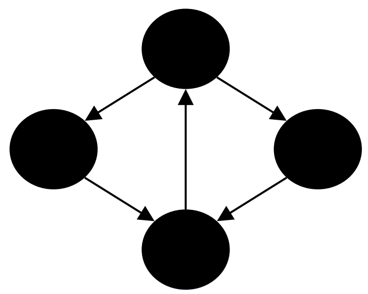
Disconnected graph

An undirected graph is complete, or fully connected, if for all pairs of vertices *u≠v* there exists an *edge* from *u* to *v*

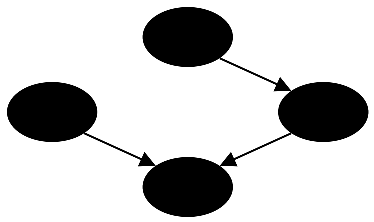


Directed Graph Connectivity

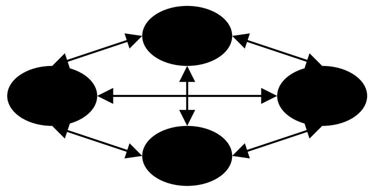
- A directed graph is strongly connected if there is a path from every vertex to every other vertex



- A directed graph is weakly connected if there is a path from every vertex to every other vertex *ignoring direction of edges*

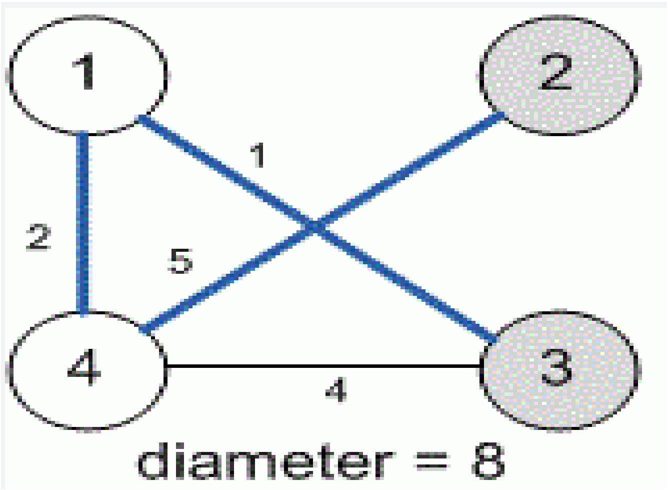


- A direct graph is complete or fully connected, if for all pairs of vertices $u \neq v$, there exists an *edge* from *u* to *v*



Diameter

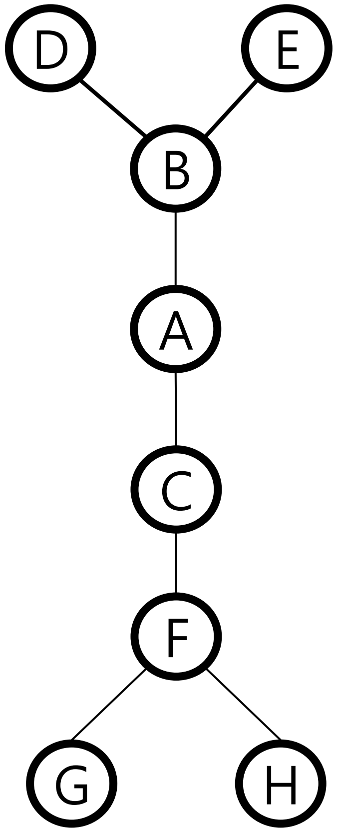
- The diameter of a graph is the largest shortest paths (maximal distance betwee n any two nodes) in the network.



https://www.eolymp.com/en/problems/1796

Trees as Graphs

- When talking about graphs, we say a tree is a graph that is:
  - Undirected
  - Acyclic
  - Connected
- All trees are graphs, but NOT all graphs are trees
- How does this relate to the trees we know ?

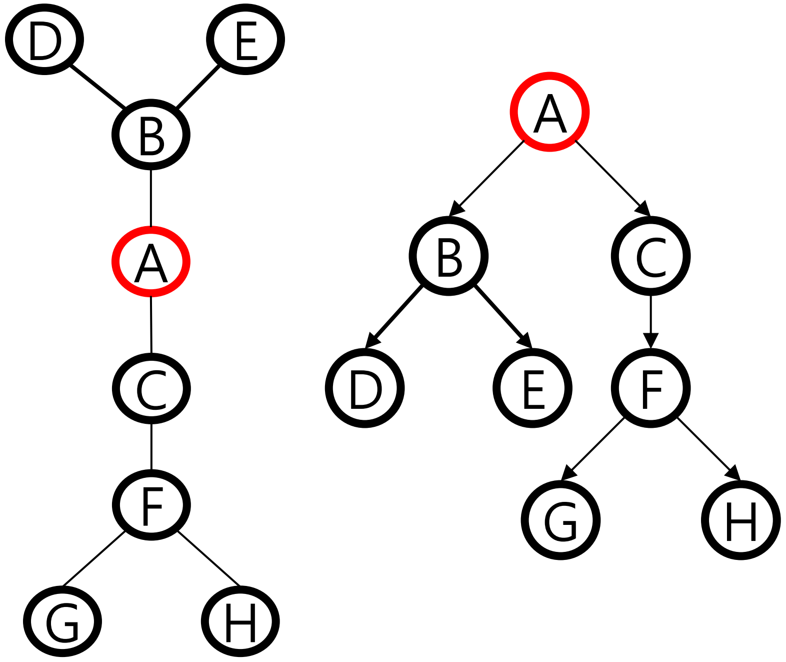


Rooted Trees

- We are more accustomed to rooted trees where:
  - We identify a unique root
  - We think of edges as directed: parent to children

Picking a root gives a unique rooted tree

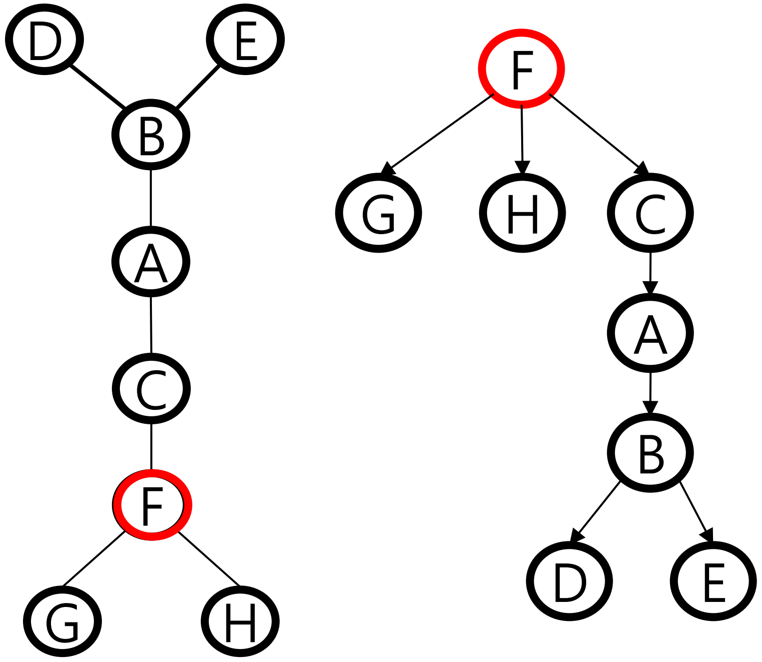
- The tree is simply drawn differently and with undirected edges



- We are more accustomed to rooted trees where:
  - We identify a unique root
  - We think of edges as directed: parent to children

Picking a root gives a unique rooted tree

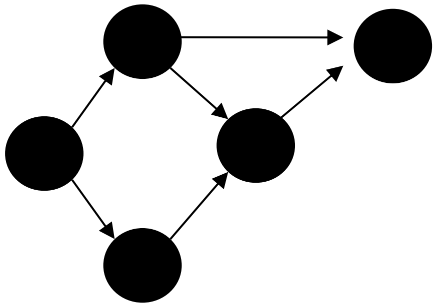
- The tree is simply drawn differently and with undirected edges

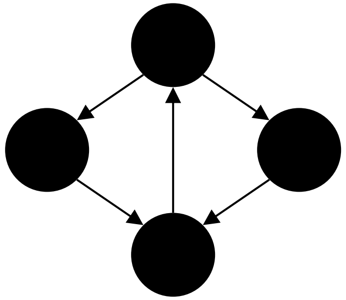


Directed Acyclic Graphs (DAGs)

A DAG is a directed graph with no directed cycles

- Every rooted directed tree is a DAG
- But not every DAG is a rooted directed tree
- Every DAG is a directed graph
- But not every directed graph is a DAG





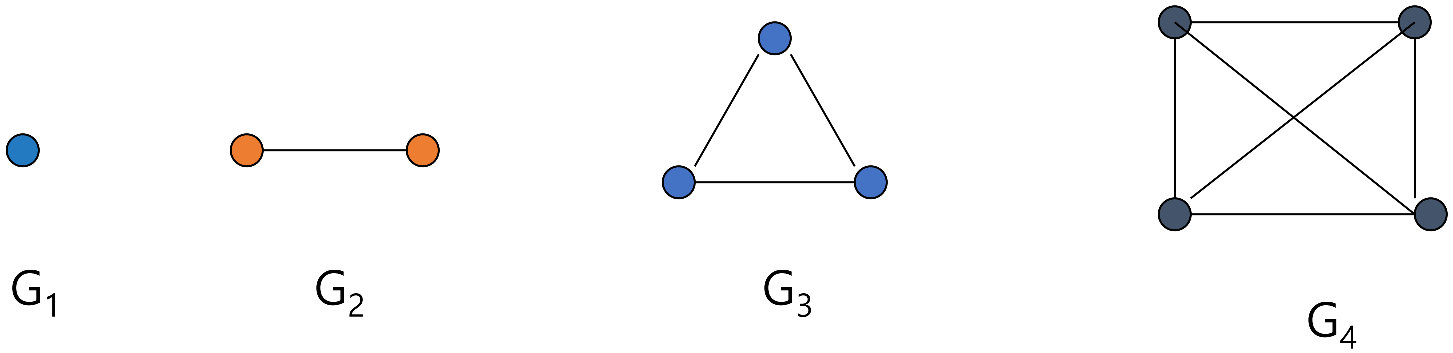
- Recall:
  In an undirected graph, $0 \le |E| < |V|^2$
- Recall:
  In a directed graph, $0 \le |E| \le |V|^2$

- So for any graph, $|E|$ is $O(|V|^2)$

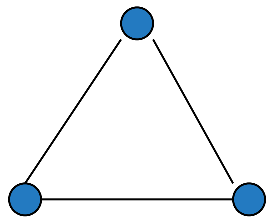
- Another fact: If an undirected graph is *connected*, then $|E| \ge |V|-1$ (pigeonhole principle)

Simple graphs – special cases

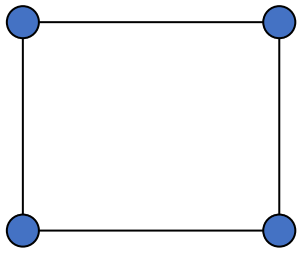
- **Complete graph:** G_n, is the simple graph that contains exactly one edge between each pair of distinct vertices.
- **Representation Example**: G_1, G_2, G_3, G_4



- **Cycle:**
  - C_n, $n \geq 3$ consists of n vertices $v_1, v_2, v_3 \dots v_n$ and edges $\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\} \dots \{v_{n-1}, v_n\}, \{v_n, v_1\}$
- Representation Example: C_3, C_4



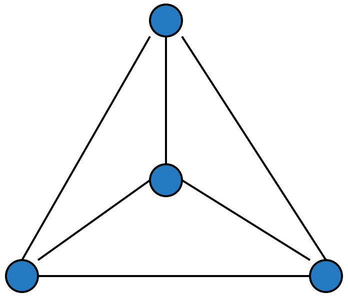
C_3



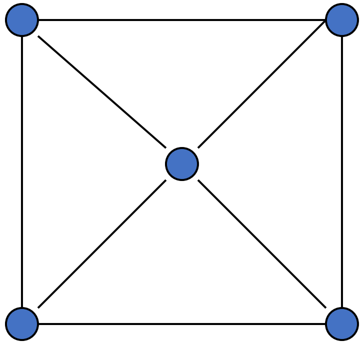
C_4

Simple graphs – special cases

- **Wheels:**
  - W_n obtained by adding additional vertex to C_n and connecting all vertices to this new vertex by new edges.
- Representation Example: W_3, W_4

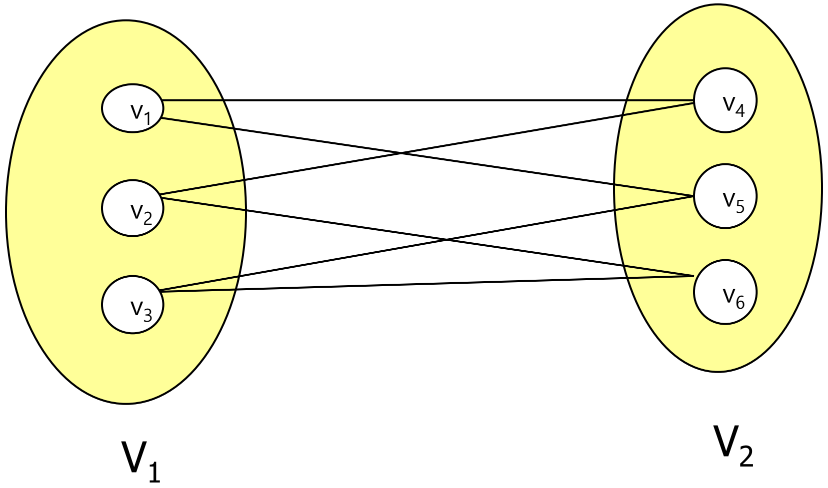


W_3

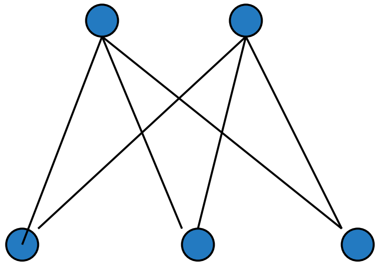


W_4

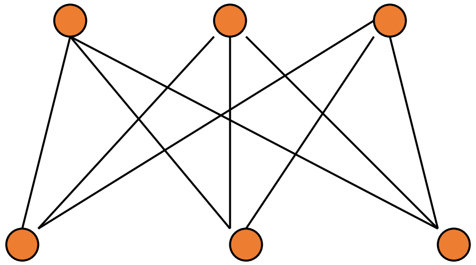
- In a simple graph G:
  - if V can be partitioned into two disjoint sets V_1 and V_2 such that every edge in the graph connects a vertex in V_1 and a vertex V_2 (so that no edge in G connects either two vertices in V_1 or two vertices in V_2)
- Application example: Representing Relations
- Representation example: $V_1 = \{v_1, v_2, v_3\}$ and $V_2 = \{v_4, v_5, v_6\}$,



- $G_{m,n}$ is the graph that has its vertex set portioned into two subsets of m and n vertices, respectively
  - There is an edge between two vertices if and only if one vertex is in the first subset and the other vertex is in the second subset.
- Representation example: $G_{2,3}$, $G_{3,3}$



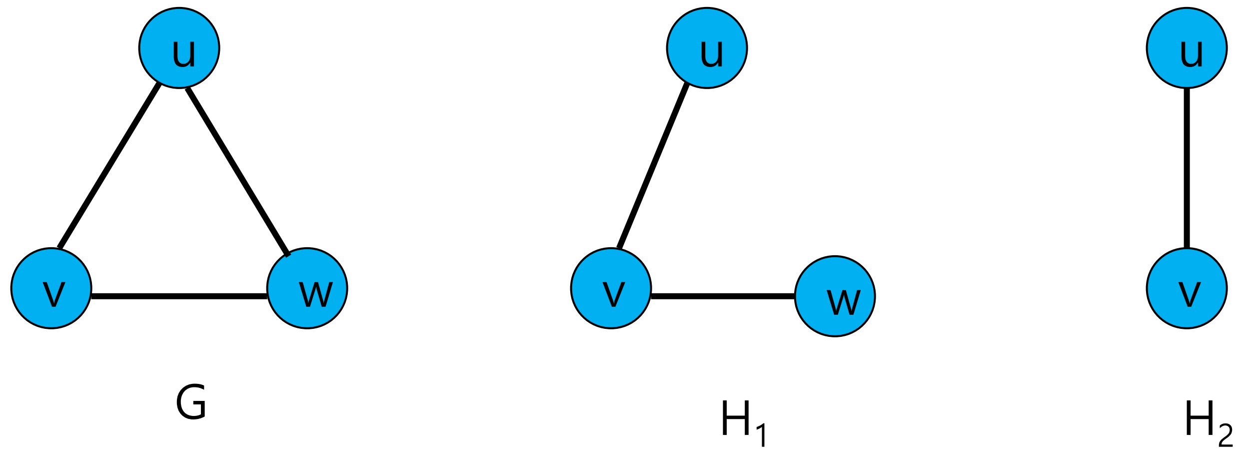
$G_{2,3}$



$G_{3,3}$

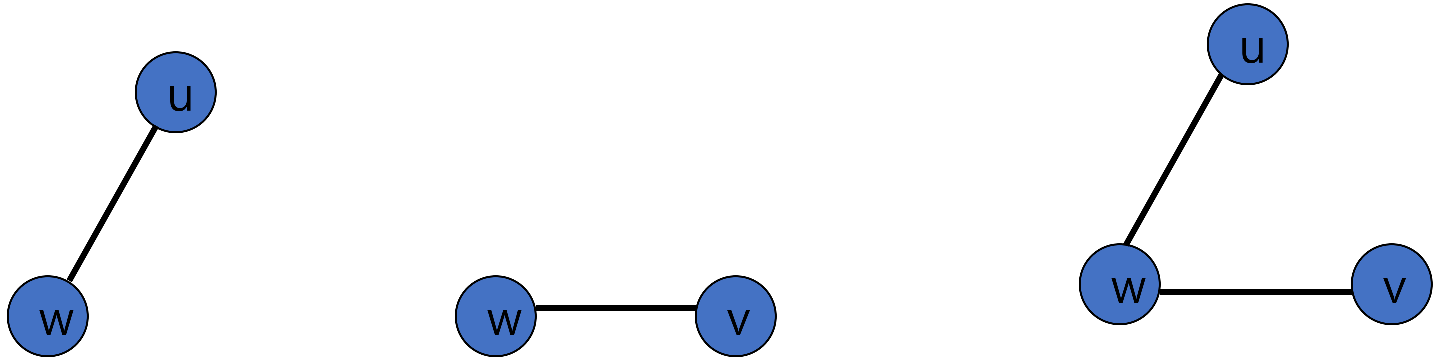
Subgraphs

- A subgraph of a graph G = (V, E) is a graph H =(V', E') where V' is a subset of V and E' is a subset of E
- Application example: solving sub-problems within a graph
- Representation example:
  - V = {u, v, w}, E = ({u, v}, {v, w}, {w, u}}, H_1 , H_2



Subgraphs

- $G = G_1 \cup G_2$ wherein $E = E_1 \cup E_2$ and $V = V_1 \cup V_2$, G, G_1 and G_2 are simple graphs of G
- Representation example:
  - $V_1 = \{u, w\}$, $E_1 = \{\{u, w\}\}$,
  - $V_2 = \{w, v\}$, $E_2 = \{\{w, v\}\}$,
  - $V = \{u, v ,w\}$, $E = \{\{\{u, w\}, \{\{w, v\}\}$

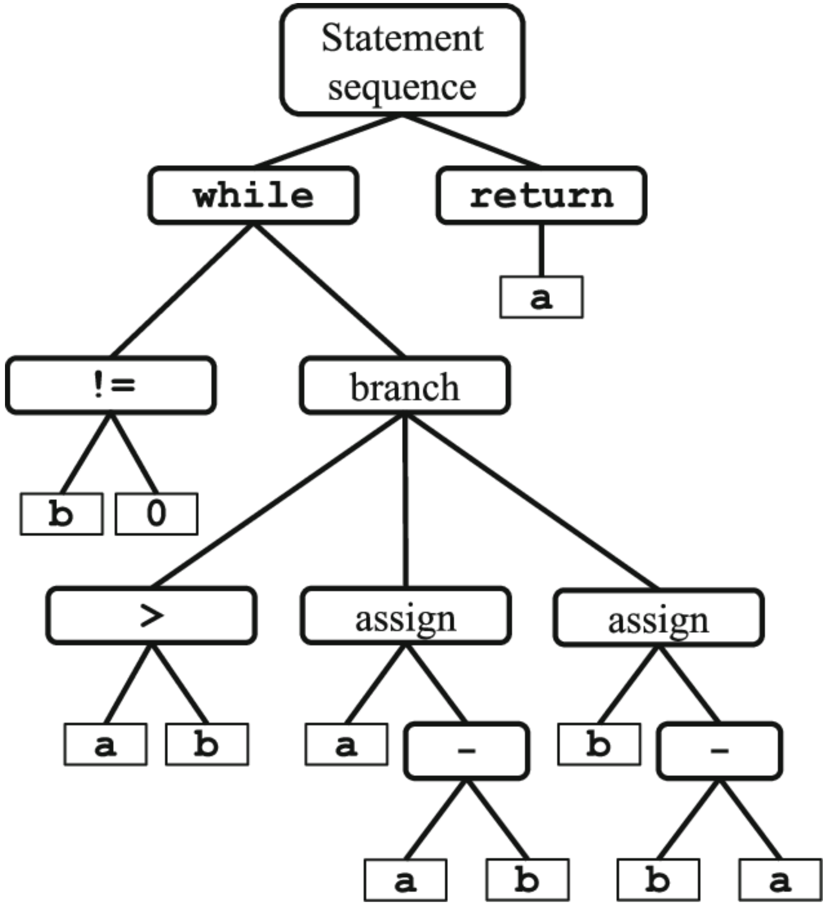


G1 G2 G

Applications of Graph in real life

- Computer software
- Semiconductor manufacturing
- Linguistics
- Physics and Chemistry
- Computer Network
- Biology
- Social Sciences

- Graphs are used to define the flow of computation program.
- Automated program repair
- Automated error location detection
- Program synthesis
  - construct a program that provably satisfies a given high-level formal specification.



Source code as abstract syntax tree

https://www.researchgate.net/publication/300802898_Rule-Based_Techniques_Using_Abstract_Syntax_Tree_for_Code_Optimization_and_Secure_Programming_in_Java

- Graph are used in designing of circuit connections.
  - semiconductor material screening
  - circuit design
  - chip design
  - semiconductor manufacturing and supply chain management

Linguistics

- In linguistics, graphs are mostly used for parsing of a language tree and grammar of a language tree.
- Semantics networks are used within lexical semantics, especially as applied to computers, modelling word meaning is easier when a given word is understood in terms of related words.

Physics and Chemistry

- In physics and chemistry, graph theory is used to study molecules.
- The 3D structure of complicated simulated atomic structures can be studied quantitatively by gathering statistics on graph-theoretic properties related to the topology of the atoms.
- Graph is also helpful in constructing the molecular structure as well as lattice of the molecule.
  - It also helps us to show the bond relation in between atoms and molecules, also help in comparing structure of one molecule to other.

- In computer network, the relationships among interconnected computers within the network, follow the principles of graph theory.
- Graph theory is also used in network security.
  - Building Intrusion detection system
  - Malware detection
- Vertex colouring algorithm may be used for assigning at most four different frequencies for any GSM (Grouped Special Mobile) mobile phone networks.

- Graph theory is also used in sociology.
  - For example, to explore rumour spreading, or to measure actors' prestige notably through the use of social network analysis software.
- Acquaintanceship and friendship graphs describe whether people know each other or not.
- In influence graphs model, certain people can influence the behavior of others.

Biology

- Nodes in biological networks represent bimolecular such as genes, proteins or metabolites, and edges connecting these nodes indicate functional, physical or chemical interactions between the corresponding bimolecular.
- Graph theory is used in transcriptional regulation networks.
- It is also used in Metabolic networks.
- In PPI (Protein - Protein interaction) networks graph theory is also useful.
- Characterizing drug - drug target relationships.

Sample code

- Creating a simple graph using NetworkX

Sample code: Setting up your environment

- I suggest you to install free Anaconda Python distribution
- Python environment: 3.8
- Run command: pip install networkx (v. 3.0)
- Run code on Jupyter notebook

Sample code: Import libraries into the project

Import libraries in the project

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

Sample code: Undirected graph

```
G = networkx.Graph()
```

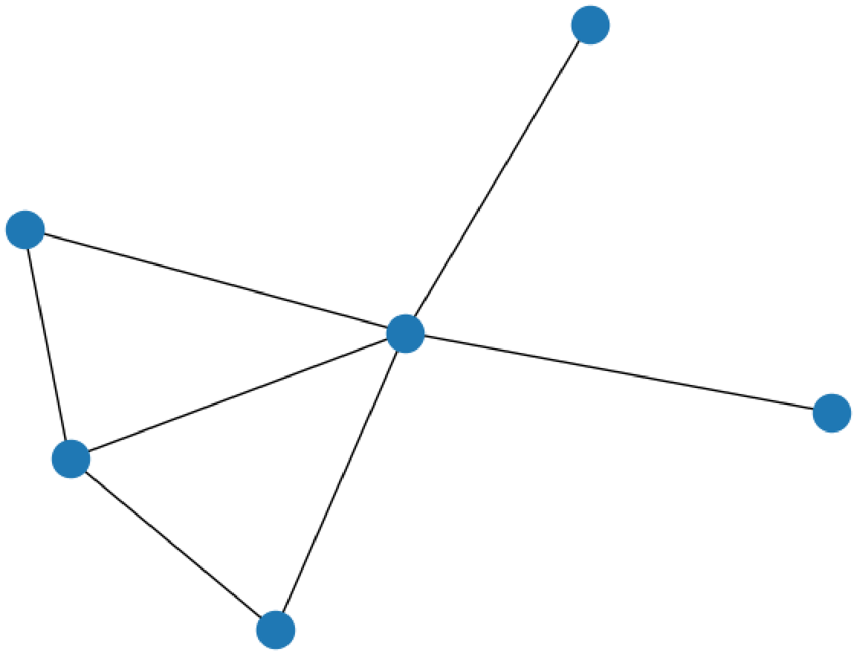
To add a node

```
G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)
G.add_node(6)
```

To add an edge

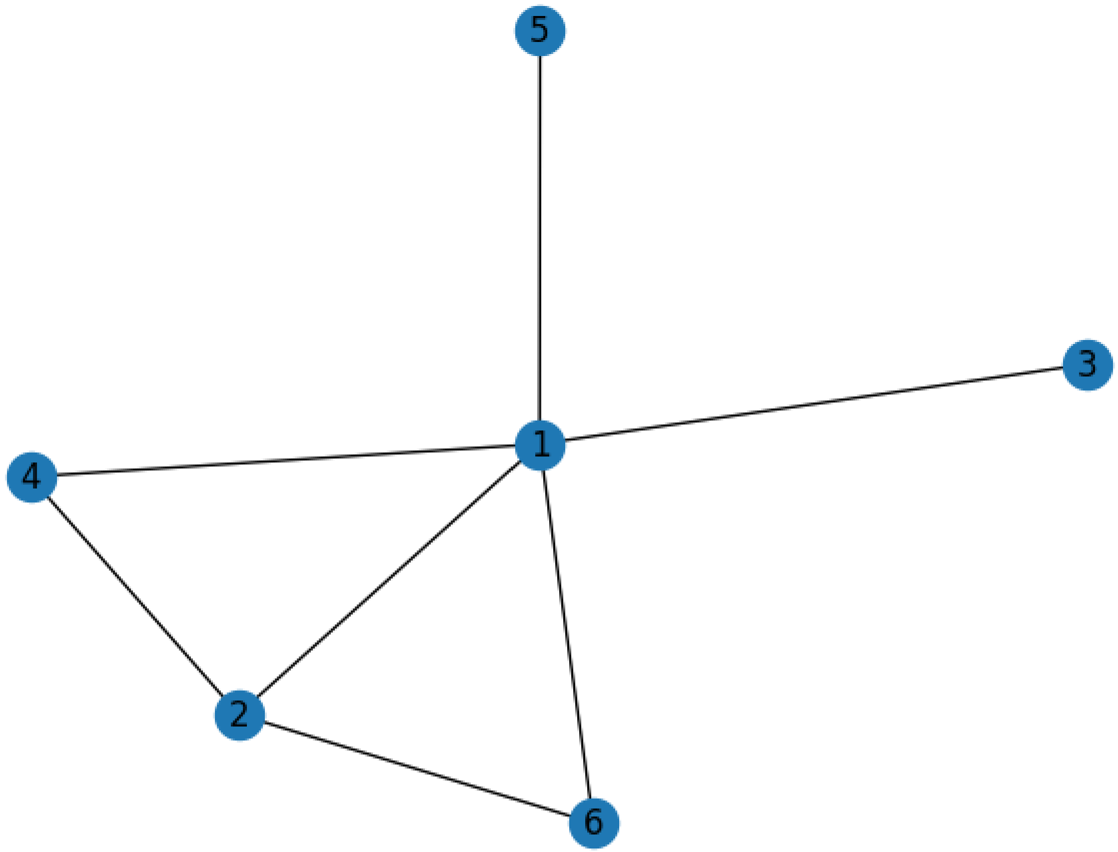
```
G.add_edge(1,2)
G.add_edge(3,1)
G.add_edge(2,4)
G.add_edge(4,1)
G.add_edge(5,1)
G.add_edge(1,6)
G.add_edge(6,2)
```

```
nx.draw(G)
```



Sample code: Undirected graph


```
nx.draw(G, with_labels = True)
```



Sample code: Undirected graph

To get all the nodes, edges of a graph

```
node_list = G.nodes()

print(node_list)

edge_list = G.edges()

print(edge_list)
```

```
[1, 2, 3, 4, 5, 6]
[(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 4), (2, 6)]
```

To remove a node, edge of a graph

```
G.remove_node(4)
node_list = G.nodes()

print(node_list)

G.remove_edge(1,2)
edge_list = G.edges()

print(edge_list)

nx.draw(G, with_labels = True)
```

```
[1, 2, 3, 5, 6]
[(1, 3), (1, 5), (1, 6), (2, 6)]
```

Sample code: Undirected graph

To find number of nodes, edges

```
n = G.number_of_nodes()
print(n)

m = G.number_of_edges()
print(m)
```

```
5
4
```

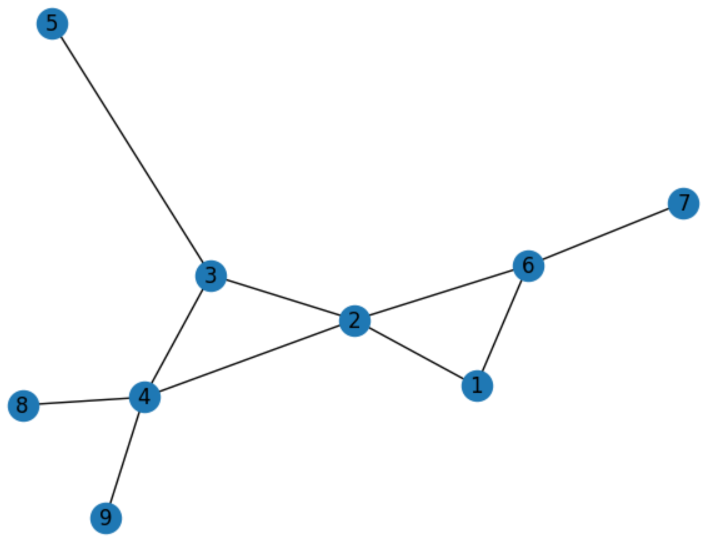
Finding the neighbours of node

```
list(G.adj[1])
```

```
[3, 5, 6]
```

Creating undirected Graph from edge list

```
import networkx as nx
G = nx.Graph()
edges = [(1, 2, 5), (1, 6, 2), (2, 3, 6), (2, 4, 3),
         (2, 6, 5), (3, 4, 6), (3, 5, 1), (4, 8, 9),
         (4, 9, 8), (6, 7, 4)]
G.add_weighted_edges_from(edges)
#nx.draw_networkx(G)
nx.draw(G, with_labels = True)
```



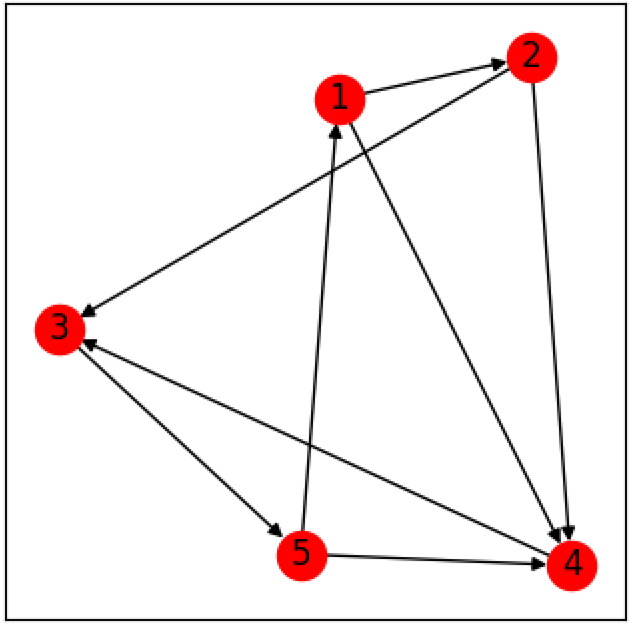
Create a directed graph

```
import networkx as nx
G = nx.DiGraph()
G.add_edges_from([(1, 2), (1, 4),
                  (2, 3), (2, 4),
                  (3, 5),
                  (4, 3),
                  (5, 4), (5, 1)])

plt.figure(figsize =(4, 4))
nx.draw_networkx(G, with_labels = True, node_color ='red')

print("# nodes: ", int(G.number_of_nodes()))
print("# edges: ", int(G.number_of_edges()))
print("List of nodes: ", list(G.nodes()))
print("List of edges: ", list(G.edges()))
print("In-degree for nodes: ", dict(G.in_degree()))
print("Out degree for nodes: ", dict(G.out_degree))
```

```
# nodes:  5
# edges:  8
List of nodes:  [1, 2, 4, 3, 5]
List of edges:  [(1, 2), (1, 4), (2, 3), (2, 4), (4, 3), (3, 5), (5, 4), (5, 1)]
In-degree for nodes:  {1: 1, 2: 1, 4: 3, 3: 2, 5: 1}
Out degree for nodes:  {1: 2, 2: 2, 4: 1, 3: 1, 5: 2}
```

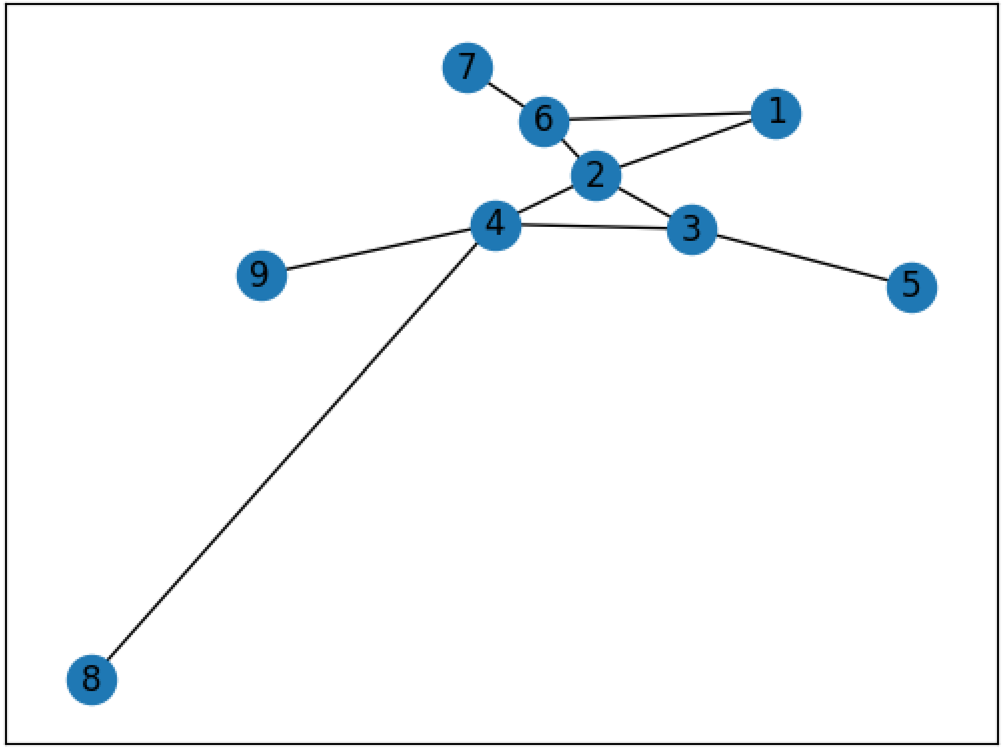


3. Weighted undirected Graph

```
G = nx.Graph()

edges = [(1, 2, 2), (1, 6, 1), (2, 3, 6), (2, 4, 7),
         (2, 6, 9), (3, 4, 3), (3, 5, 2), (4, 8, 0),
         (4, 9, 2), (6, 7, 6)]

G.add_weighted_edges_from(edges)
nx.draw_networkx(G, with_labels = True)
```



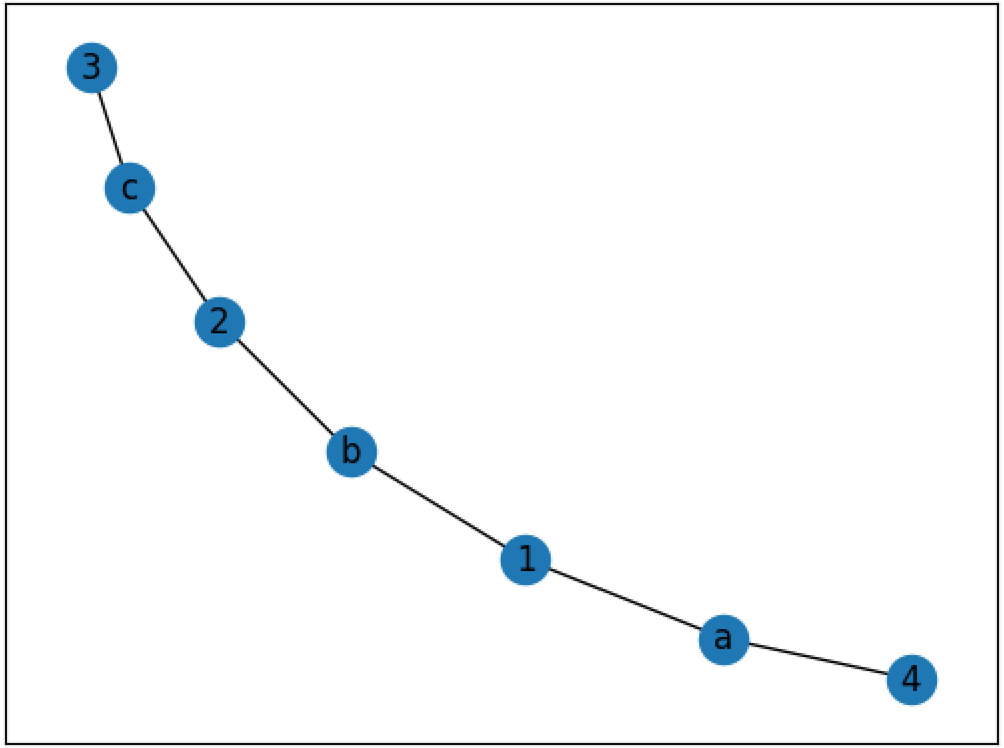
Sample code: Bipartite graph

4. Bipartite graph

```
B = nx.Graph()


B.add_nodes_from([1, 2, 3, 4], bipartite=0)
B.add_nodes_from(["a", "b", "c"], bipartite=1)

B.add_edges_from([(1, "a"), (1, "b"), (2, "b"), (2, "c"), (3, "c"), (4, "a")])
nx.draw_networkx(B, with_labels = True)
```



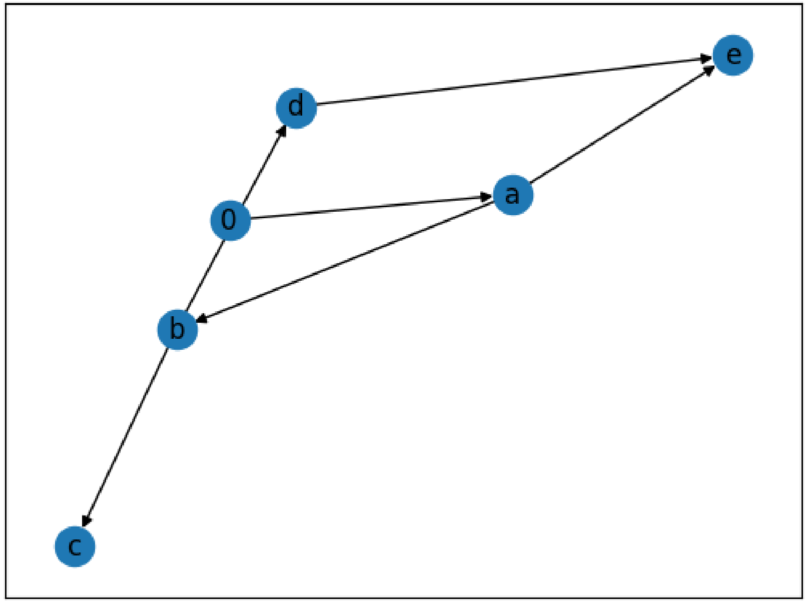
5. The diameter of the graph G

```
G = nx.Graph([(1, 2), (1, 3), (1, 4), (3, 4), (3, 5), (4, 5)])
nx.diameter(G)
```

```
3
```

6. DAGs / Directed Acyclic Graphs

```
import networkx as nx
G = nx.DiGraph()
G.add_edges_from([("0", "a"), ("a", "b"), ("a", "e"), ("b", "c"), ("b", "d"), ("d", "e")])
nx.draw_networkx(G, with_labels = True)
```




Q
A