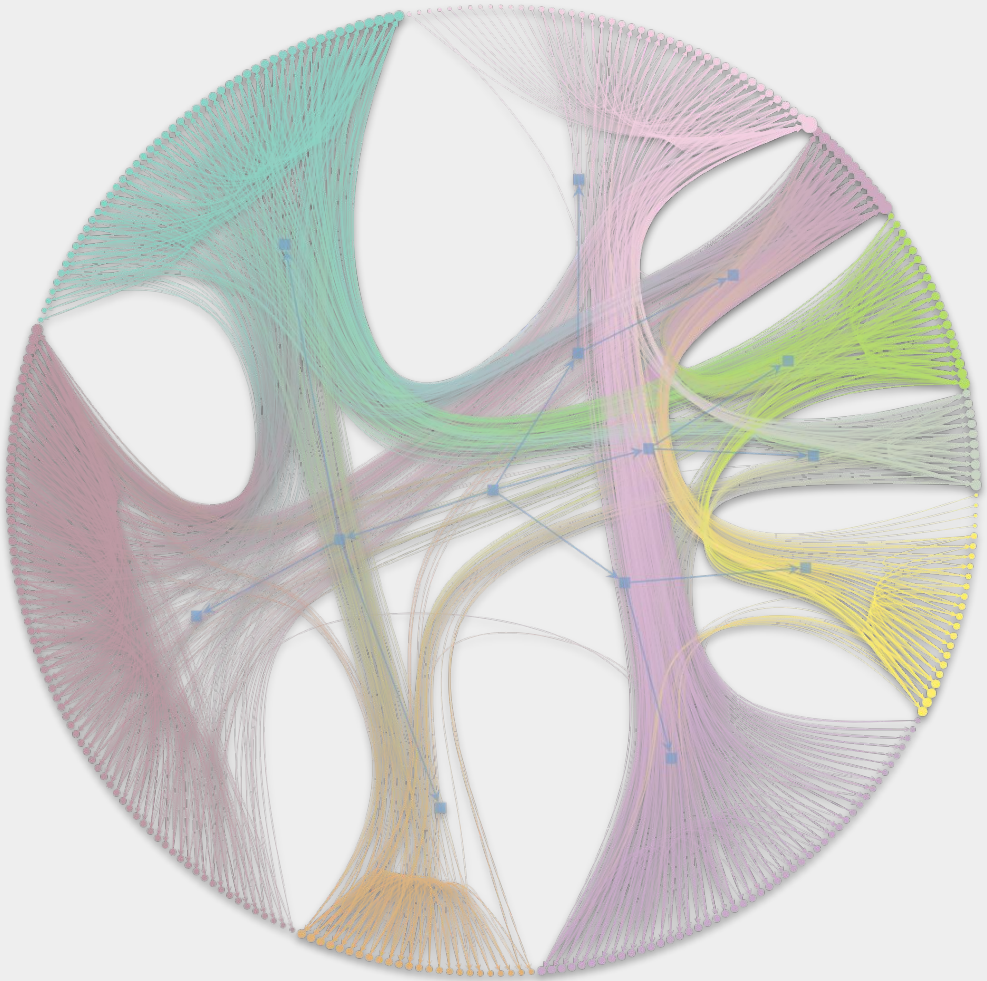
Network Elements

Part 02

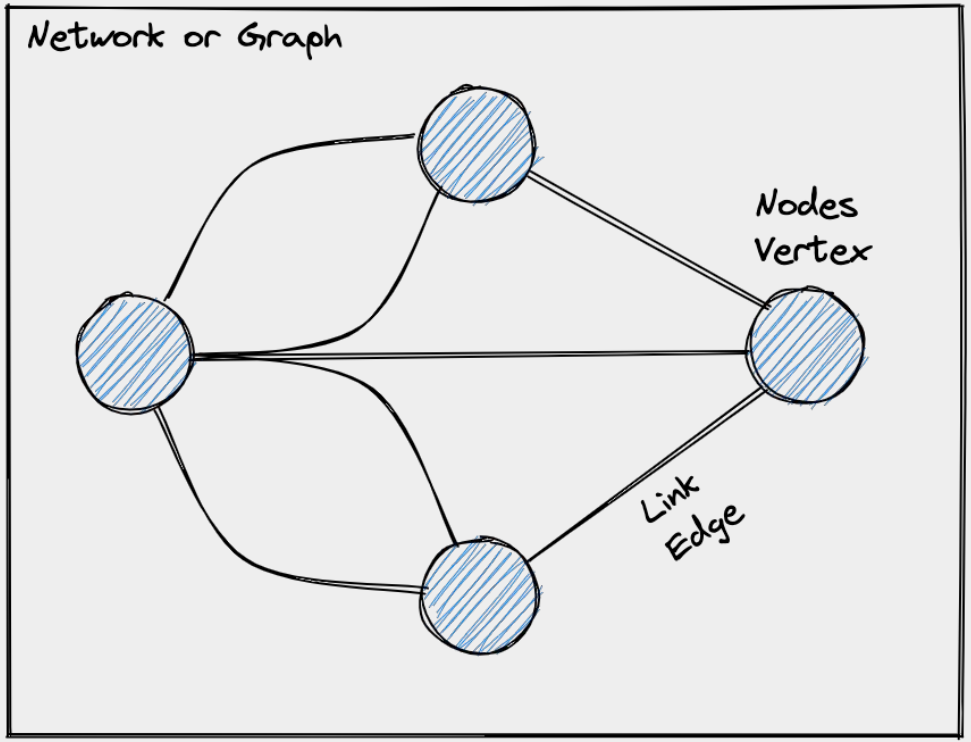
ivanovitch.silva@ufrn.br
@ivanovitchm

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

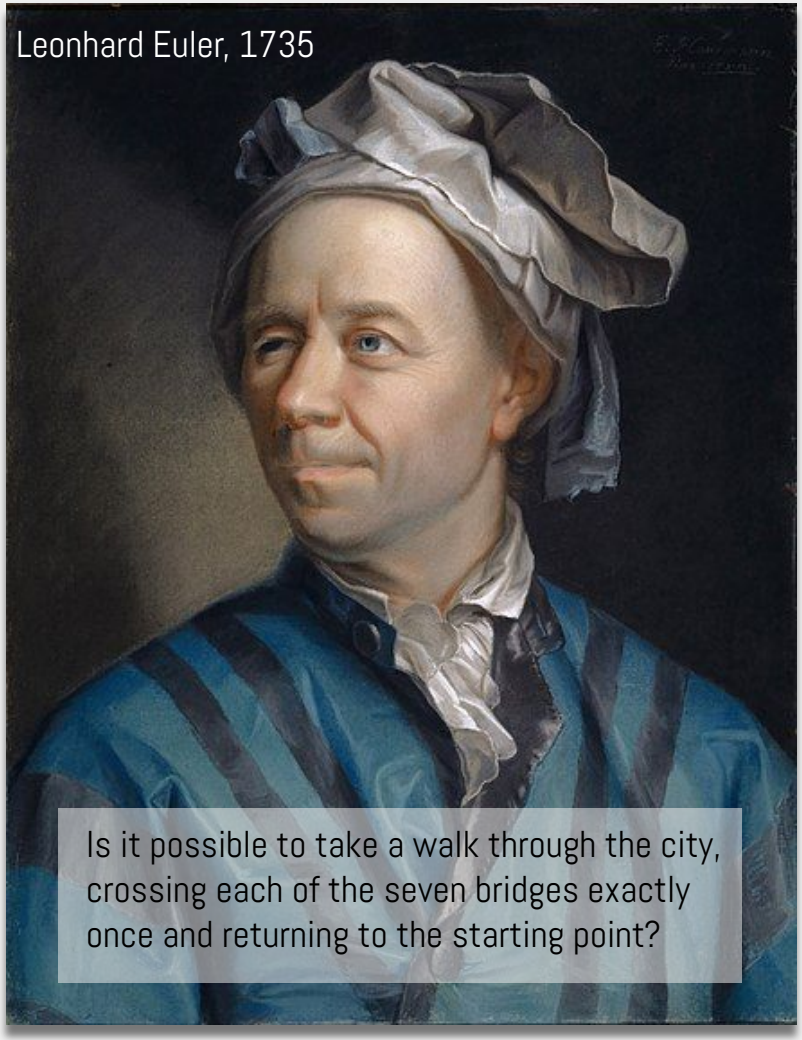
DCA



Basic Definition

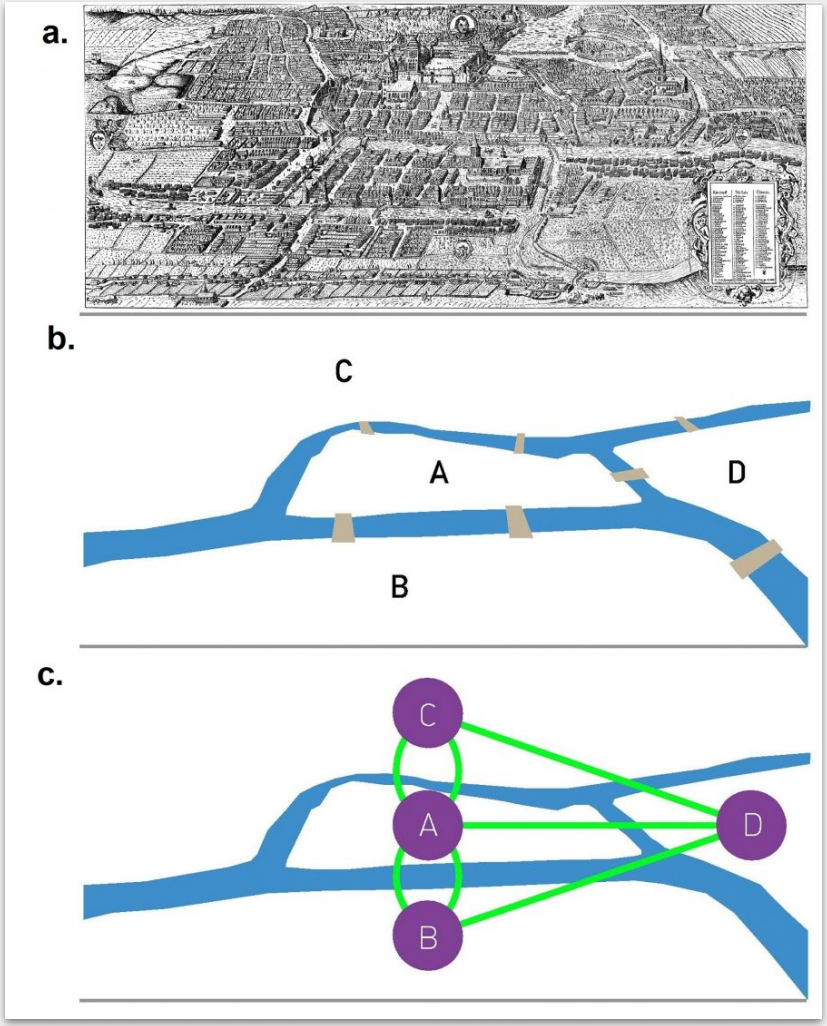


In very general terms a network, or graph, is a set of elements, which we call **nodes** , along with a set of connections between pairs of nodes, which we call **links** . The links represent the presence of a relation among the elements represented by the nodes.

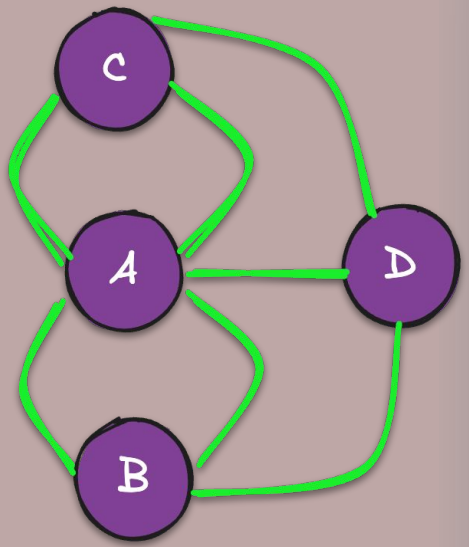


Seven Bridges of Königsberg
Can one walk across all seven bridges and never cross the same one twice?

The rigorous language for the description of networks is found in graph theory, a field of mathematics that can be traced back to the pioneering work of Leonhard Euler in the eighteenth century.



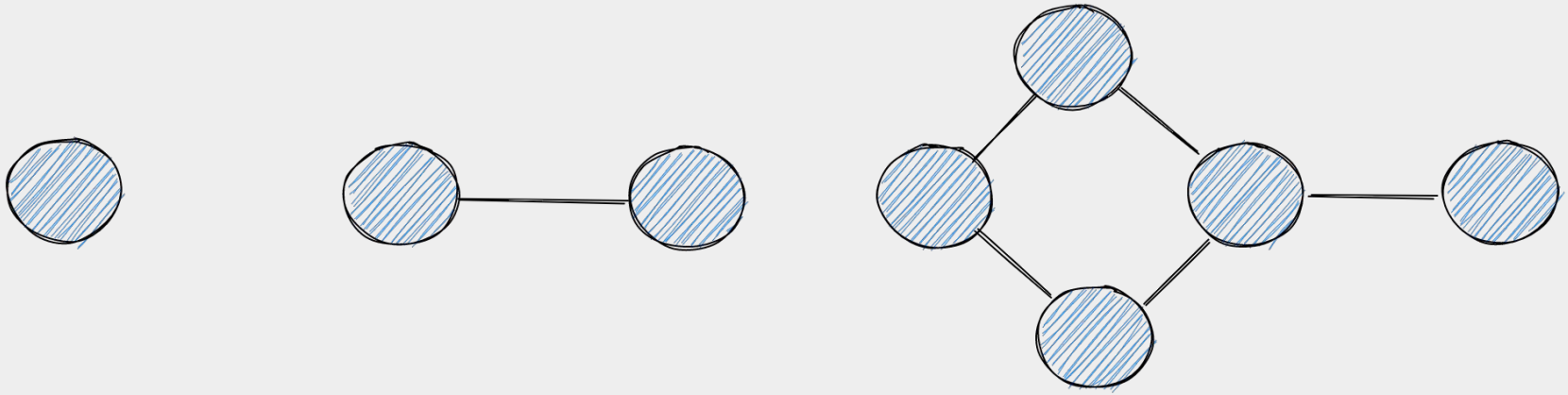
Despite many attempts, no one could find such path. The problem remained unsolved until 1735, when Leonhard Euler, offered a rigorous mathematical proof that such path does not exist.




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

# Create an undirected graph
G = nx.Graph()

# Add nodes (land masses)
G.add_nodes_from(['A', 'B', 'C', 'D'])

# Add edges (bridges between the land masses)
edges = [
    ('A', 'B'), ('A', 'B'),        # Two bridges between A and B
    ('A', 'C'), ('A', 'C'),        # Two bridges between A and C
    ('A', 'D'),                    # One bridge between A and D
    ('B', 'D'),                    # One bridge between B and D
    ('C', 'D')                     # One bridge between C and D
]

G.add_edges_from(edges)
```

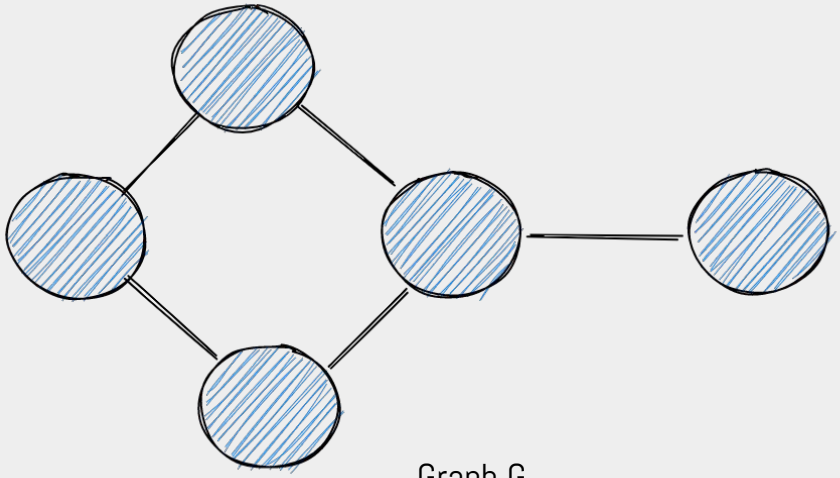
A more rigorous definition of a network

$$G = (V, E), with \ \ E \subseteq V \times V$$

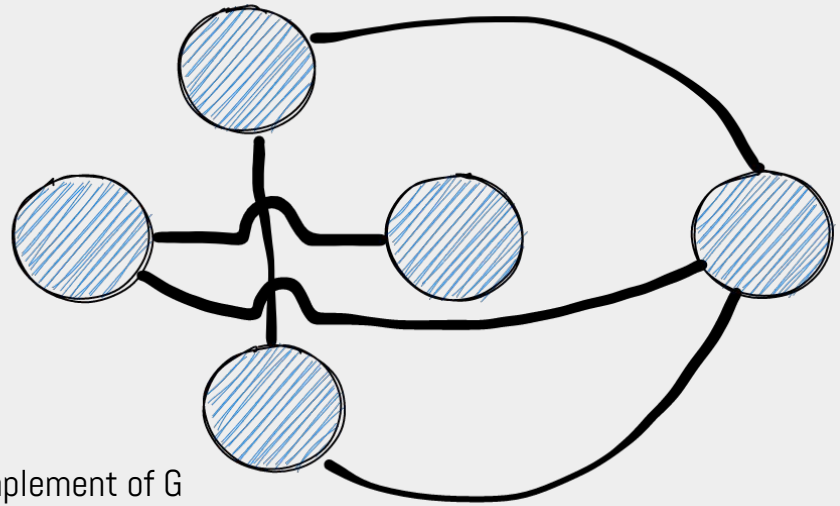


We can derive a series of special graphs

For instance, we can derive the complement of G. This is equivalent to remove all of the original edges of G, and then connect all the unconnected pairs of nodes in G.



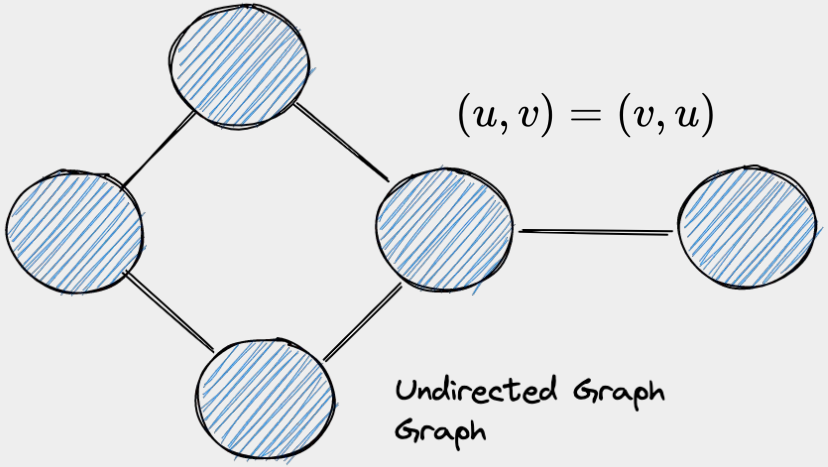
Graph G

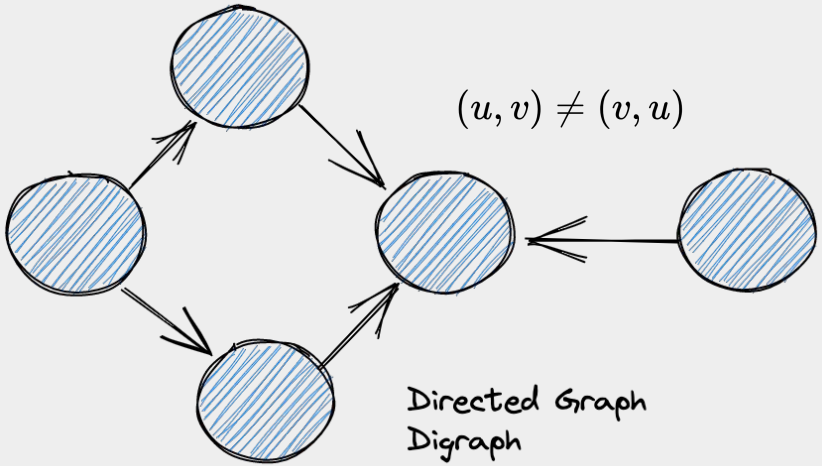


Complement of G

Sometimes you really need to complicate stuff

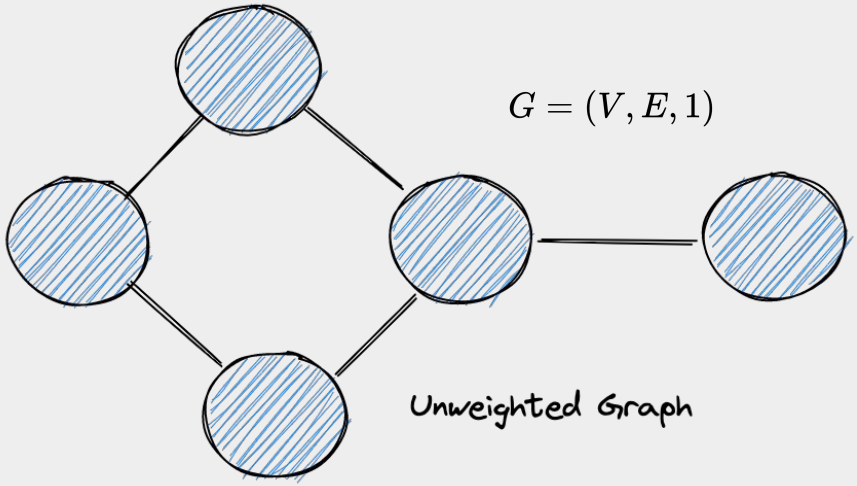
A first thing we will do is realizing that **not all relations are reciprocal** . We can introduce this asymmetry in the graph model.

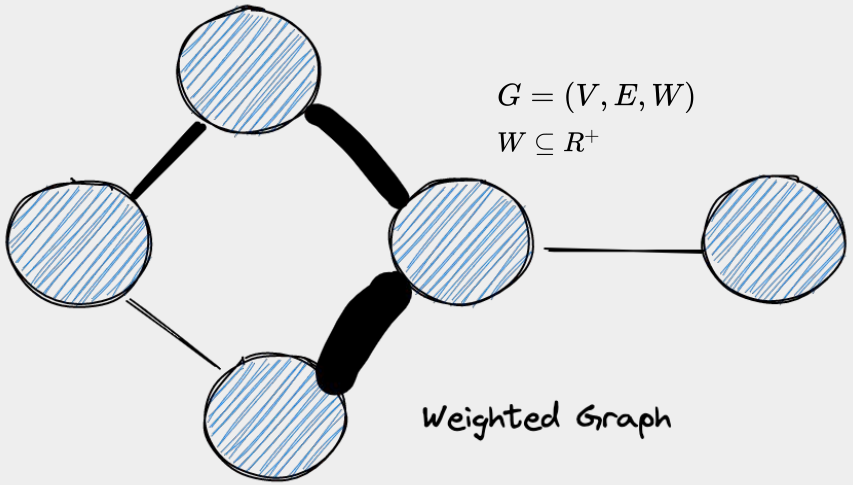


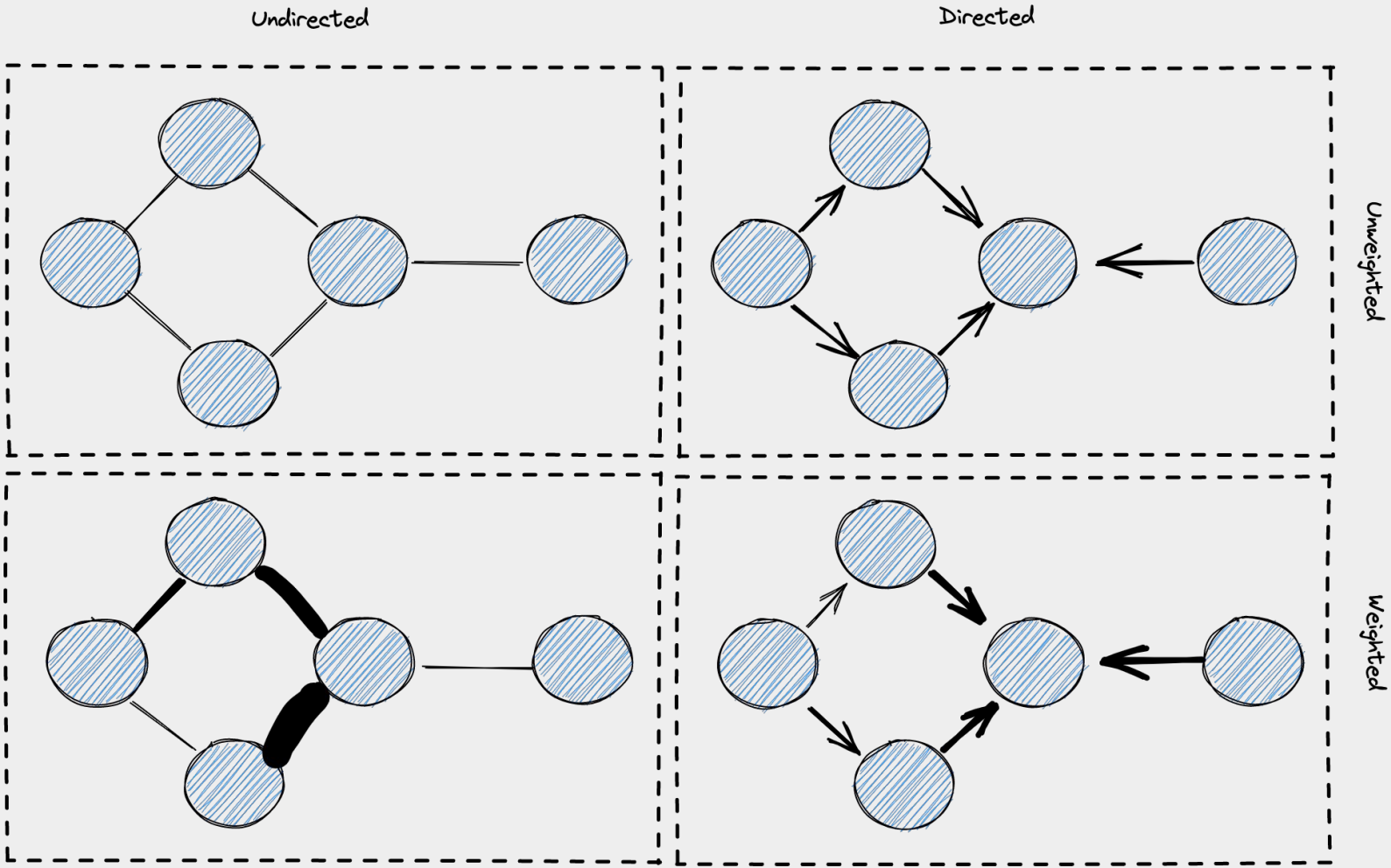


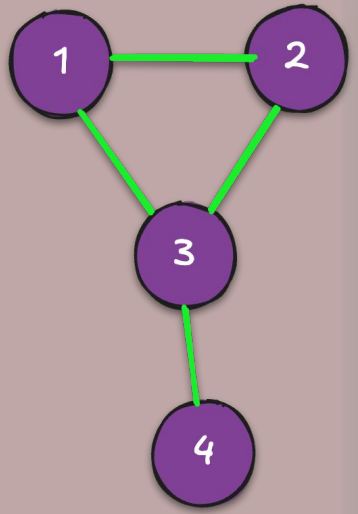
Sometimes you really need to complicate stuff

Another way to make edges more interesting is realizing that **two connections are not necessarily equally important** in the network.

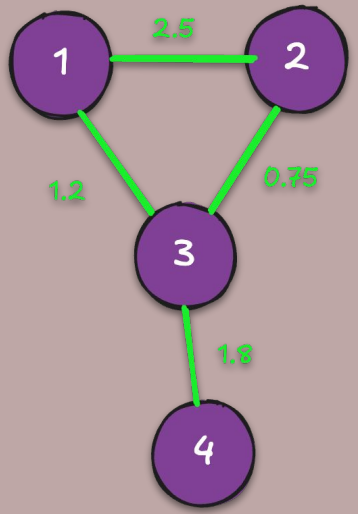





Undirected
Directed
Unweighted
Weighted

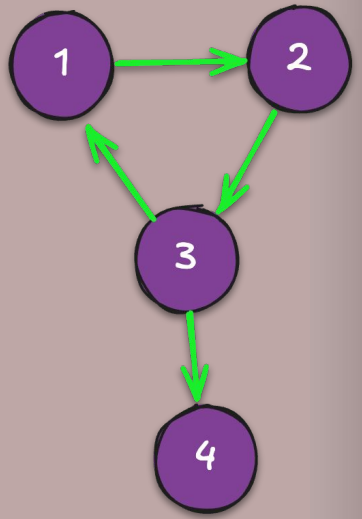



```python
import networkx as nx
import matplotlib.pyplot as plt

# Create an undirected graph
G = nx.Graph()

# Add nodes
G.add_nodes_from([1, 2, 3, 4])

# Add edges (unweighted)
edges = [(1, 2), (1, 3), (2, 3), (3, 4)]
G.add_edges_from(edges)

# Draw the graph
nx.draw(G, with_labels=True, node_color='lightblue', node_size=500)
plt.title("Undirected and Unweighted Graph")
plt.show()
```

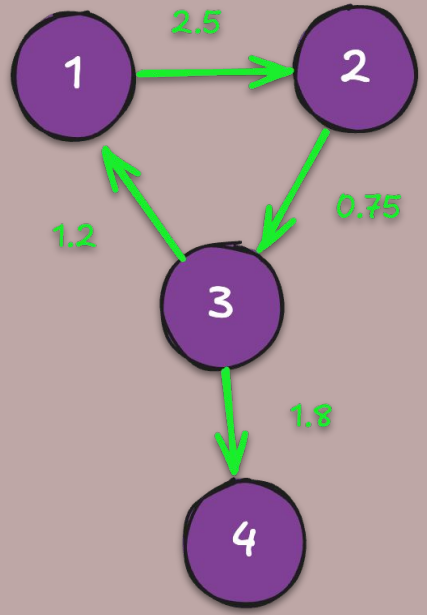



```python
import networkx as nx
import matplotlib.pyplot as plt

# Create an undirected graph
G = nx.Graph()

# Add nodes
G.add_nodes_from([1, 2, 3, 4])

# Add weighted edges
edges = [
    (1, 2, {'weight': 2.5}),
    (1, 3, {'weight': 1.2}),
    (2, 3, {'weight': 0.75}),
    (3, 4, {'weight': 1.8})
]
G.add_edges_from(edges)

# Get edge weights
edge_labels = nx.get_edge_attributes(G, 'weight')

# Draw the graph
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='lightgreen', node_size=500)
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.title("Undirected and Weighted Graph")
plt.show()
```

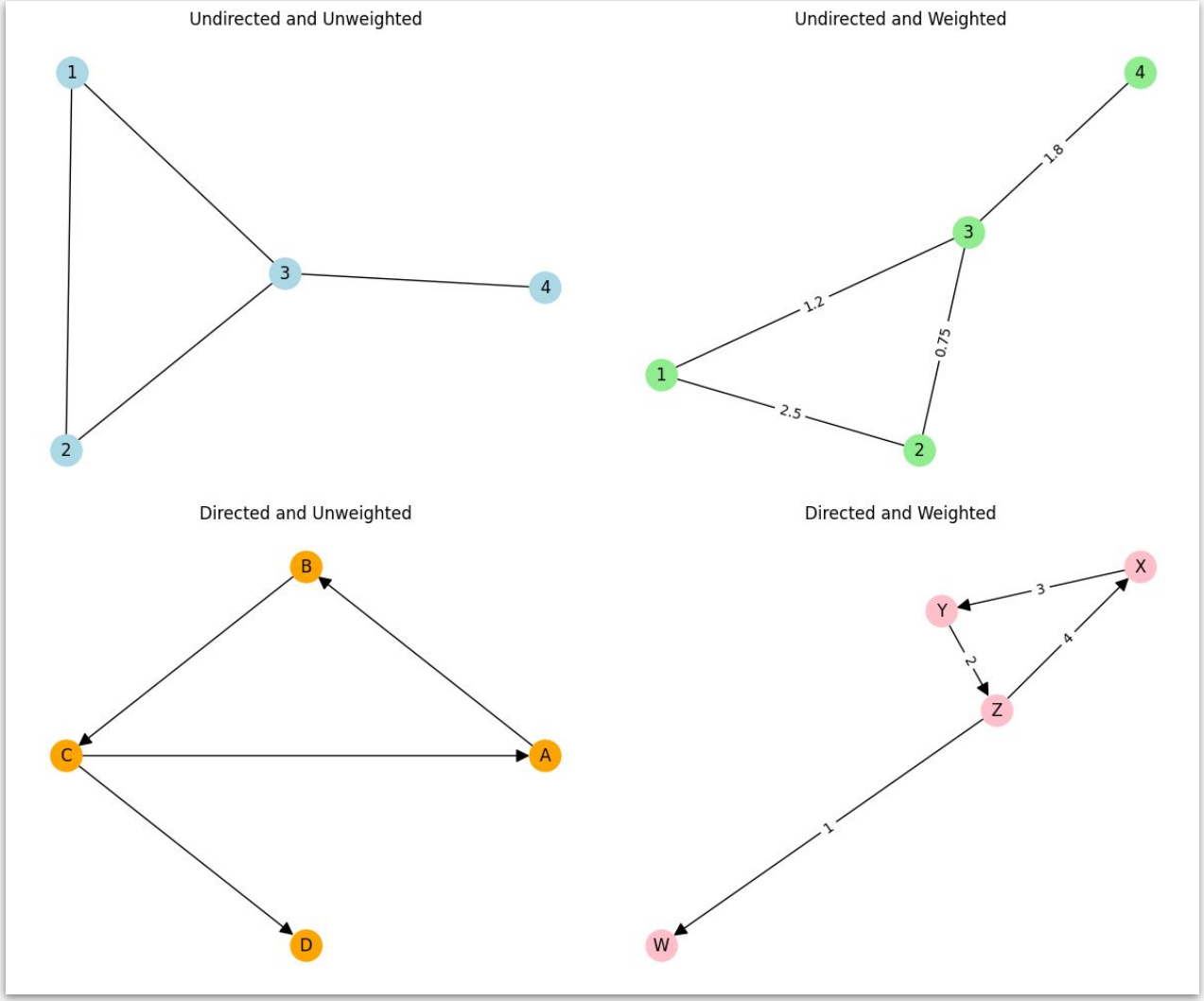
```
{(1, 2): 2.5, (1, 3): 1.2, (2, 3): 0.75, (3, 4): 1.8}
```



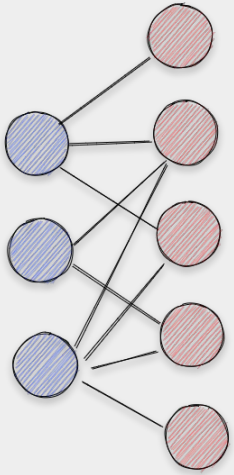

```python
import networkx as nx
import matplotlib.pyplot as plt

# Create a directed graph
G = nx.DiGraph()

# Add nodes
G.add_nodes_from(['1', '2', '3', '4'])

# Add edges (unweighted)
edges = [('1', '2'), ('2', '3'), ('3', '1'), ('3', '4')]
G.add_edges_from(edges)

# Draw the graph
pos = nx.circular_layout(G)
nx.draw(G, pos, with_labels=True, node_color='orange', node_size=500, arrowsize=20)
plt.title("Directed and Unweighted Graph")
plt.show()
```

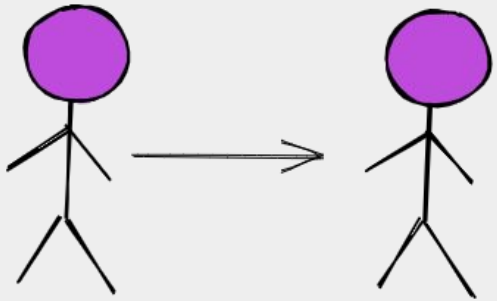



```python
import networkx as nx
import matplotlib.pyplot as plt

# Create a directed graph
G = nx.DiGraph()

# Add nodes
G.add_nodes_from([1, 2, 3, 4])

# Add weighted edges
edges = [
    (1, 2, {'weight': 2.5}),
    (3, 1, {'weight': 1.2}),
    (2, 3, {'weight': 0.75}),
    (3, 4, {'weight': 1.8})
]
G.add_edges_from(edges)

# Get edge weights
edge_labels = nx.get_edge_attributes(G, 'weight')

# Draw the graph
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='pink', node_size=500, arrowsize=20)
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.title("Directed and Weighted Graph")
plt.show()
```

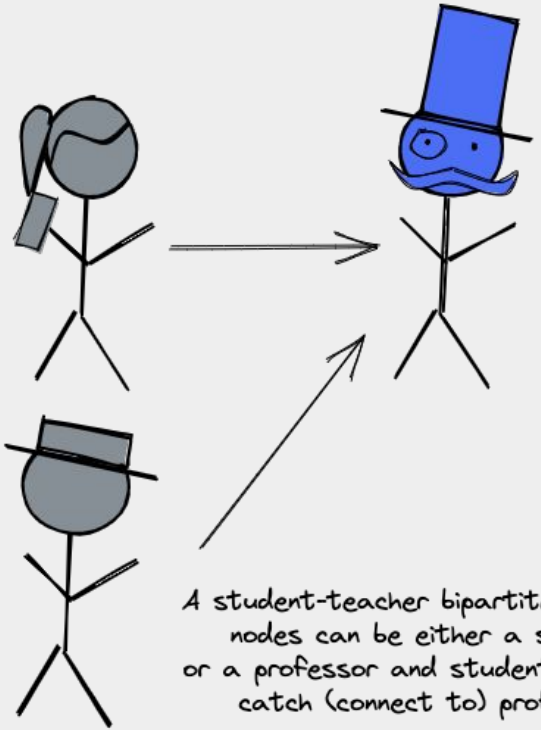

Undirected and Unweighted
1
2
3
4
Undirected and Weighted
1
2
3
4
1.2
2.5
0.75
1.8
Directed and Unweighted
A
B
C
D
Directed and Weighted
W
X
Y
Z
3
2
4
1

Extended Graphs (bipartite network)



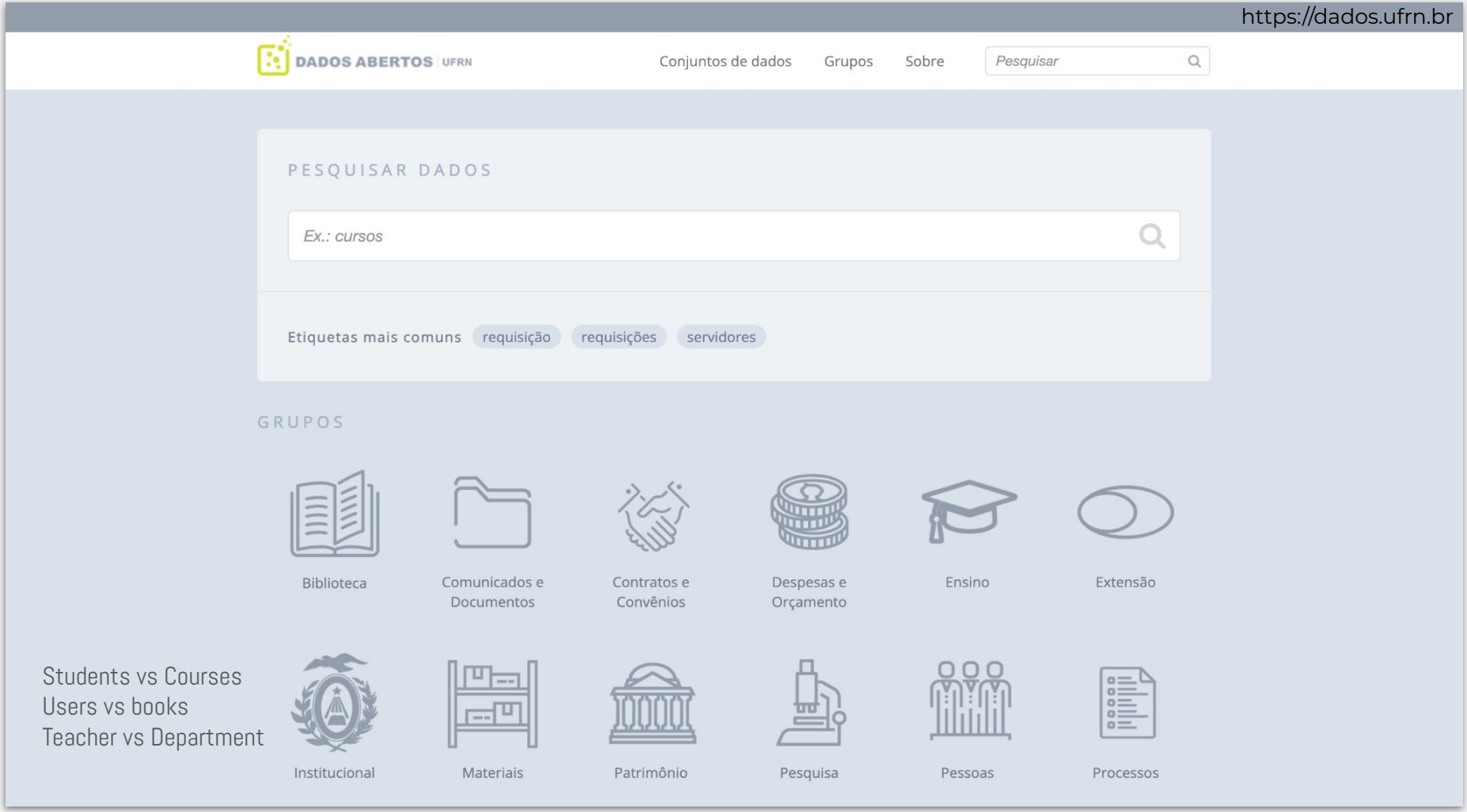


A simple graph representing
a social network with
no additional constraints.

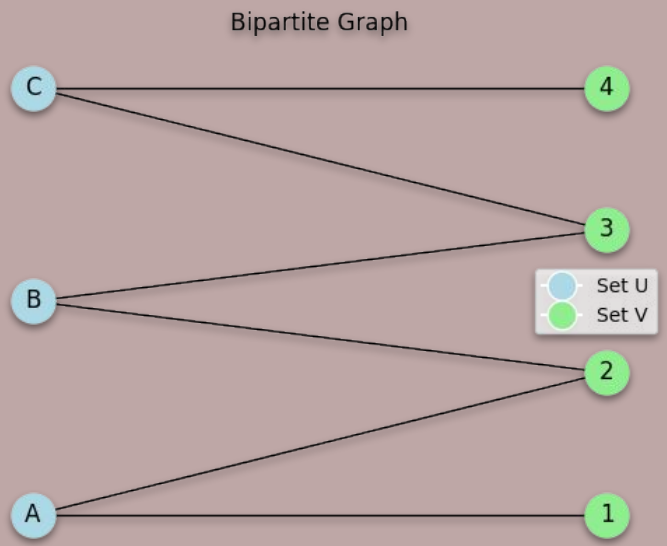


A student-teacher bipartite network:
nodes can be either a student
or a professor and students can only
catch (connect to) professor.

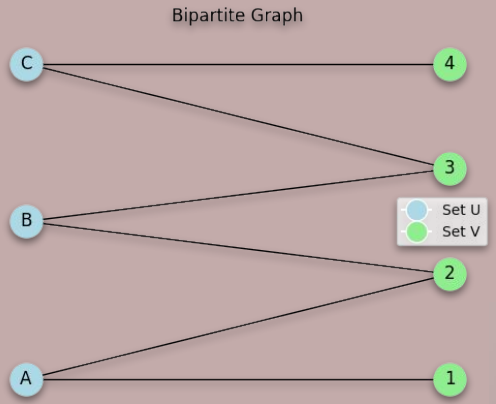
https://dados.ufrn.br

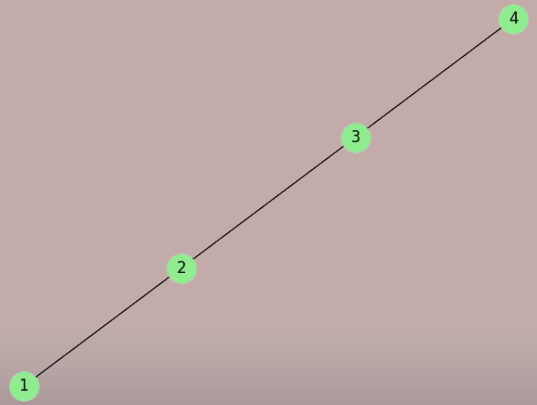


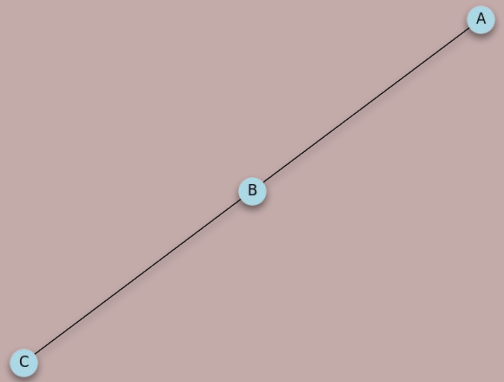
Students vs Courses
Users vs books
Teacher vs Department




```python
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D  # Import Line2D for custom legend handles

# Step 1: Create an empty graph
G = nx.Graph()

# Step 2: Define the two node sets
U = {'A', 'B', 'C'}
V = {1, 2, 3, 4}

# Step 3: Add nodes with the bipartite attribute
G.add_nodes_from(U, bipartite=0)
G.add_nodes_from(V, bipartite=1)

# Step 4: Add edges between nodes from U to V
edges = [('A', 1), ('A', 2), ('B', 2), ('B', 3), ('C', 3), ('C', 4)]
G.add_edges_from(edges)

# Step 5: Visualize the bipartite graph
pos = nx.bipartite_layout(G, U)

# Draw nodes for each set
nx.draw_networkx_nodes(G, pos, nodelist=U, node_color='lightblue', node_size=500)
nx.draw_networkx_nodes(G, pos, nodelist=V, node_color='lightgreen', node_size=500)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos)

# Create custom legend handles
legend_elements = [
    Line2D([0], [0], marker='o', color='w', label='Set U',
           markerfacecolor='lightblue', markersize=15),
    Line2D([0], [0], marker='o', color='w', label='Set V',
           markerfacecolor='lightgreen', markersize=15)
]

# Add the legend to the plot
plt.legend(handles=legend_elements)
plt.title("Bipartite Graph")
plt.axis('off')
# save the figure
plt.savefig('grafo_bipartite.png', transparent=True)
plt.show()
```





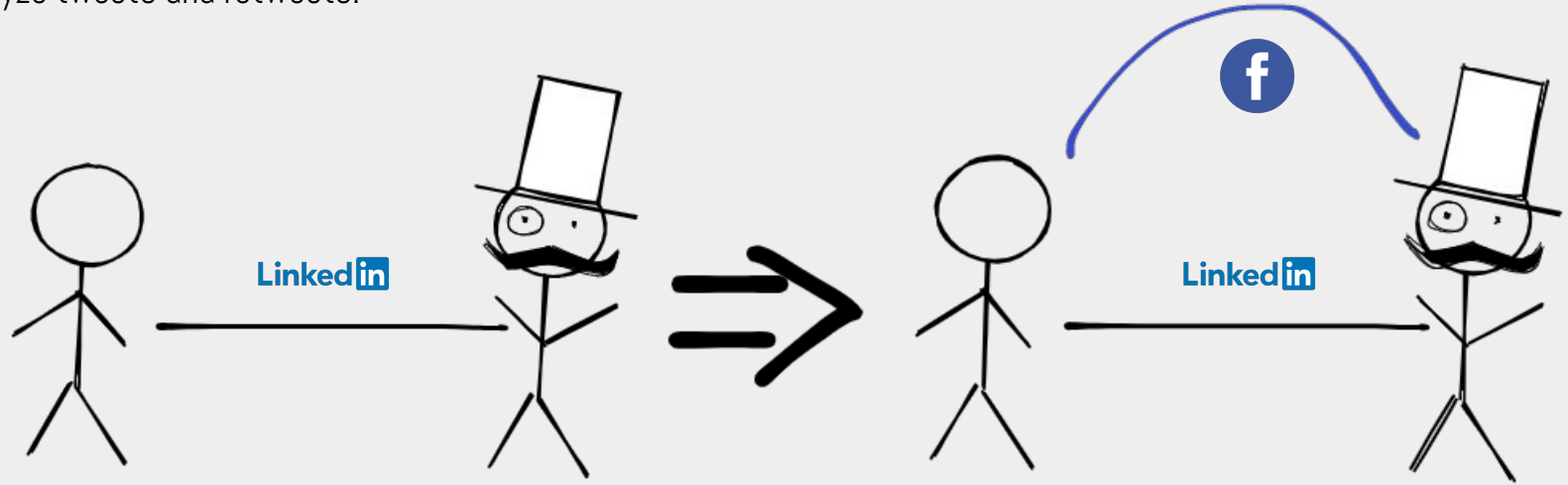



```python
from networkx.algorithms import bipartite

is_bipartite = nx.is_bipartite(G)
print(f"Is the graph bipartite? {is_bipartite}")
# Is the graph bipartite? True

# projections
projected_U = bipartite.projected_graph(G, U)
nx.draw(projected_U, with_labels=True, node_color='lightblue',node_size=500)
plt.title("Projection onto Set U")

# save the figure
plt.savefig('projected.png', transparent=True)
plt.show()
```

Extended Graphs (multilayer graph)

Traditionally, network scientists try to focus on one thing at a time. For instance, they will download a sample of the Linkedin graph. Or they will analyze tweets and retweets.

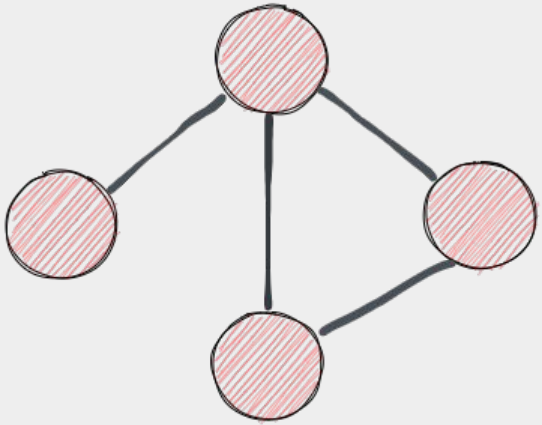
Two people might have started working in the same company and thus first connected on Linkedin, and then became friends and connected on Facebook. Such scenario could not be captured by simply looking at one of the two networks.



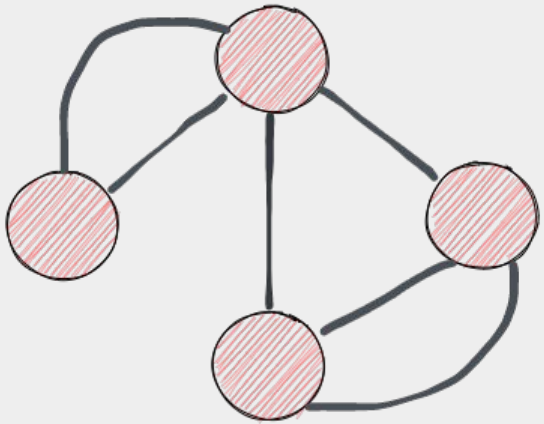
Extended Graphs (multilayer graph)

$$G = (V, E, L)$$

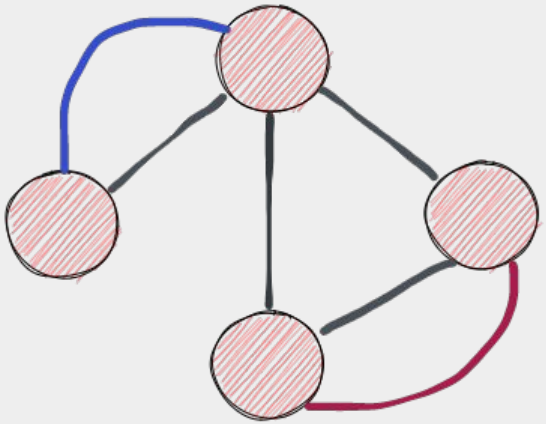
$$(u, v, l) \in E,\ with\ u, v \in V and\ l \in L$$



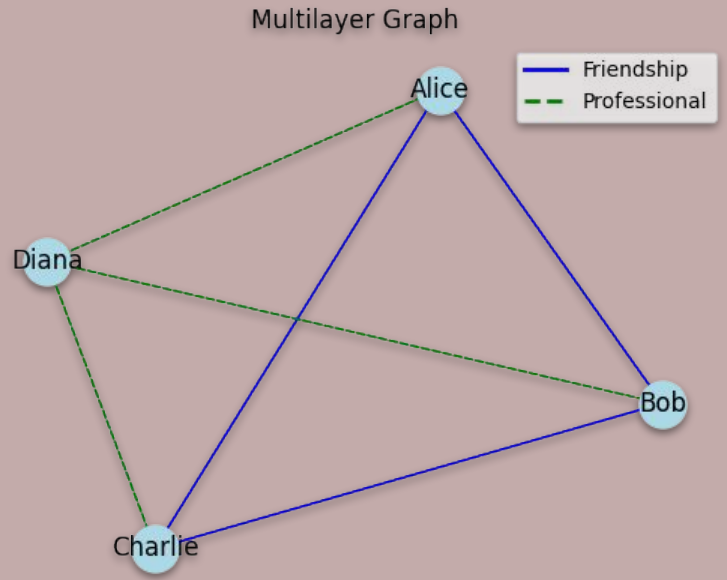
A simple graph



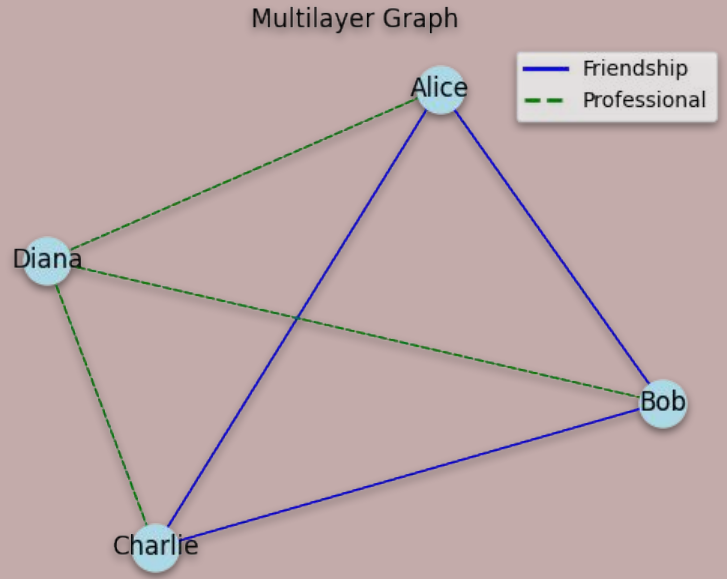
A multigraph, with multiple edges between the same node pairs



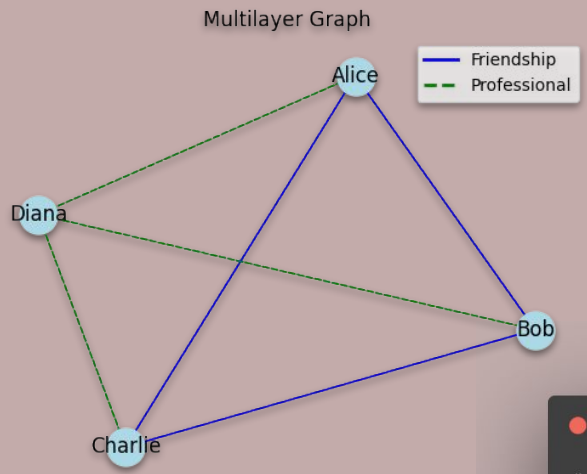
A multilayer network, where each edge has a type (represented by its color)




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

# Create a MultiGraph to handle multiple edges
G = nx.MultiGraph()

# List of nodes
nodes = ['Alice', 'Bob', 'Charlie', 'Diana']

# Add nodes to the graph
G.add_nodes_from(nodes)

# Edges in Layer 1 (Friendships)
friendships = [
    ('Alice', 'Bob'),
    ('Alice', 'Charlie'),
    ('Bob', 'Charlie')
]

# Edges in Layer 2 (Professional Relationships)
professional_relationships = [
    ('Alice', 'Diana'),
    ('Bob', 'Diana'),
    ('Charlie', 'Diana')
]
```



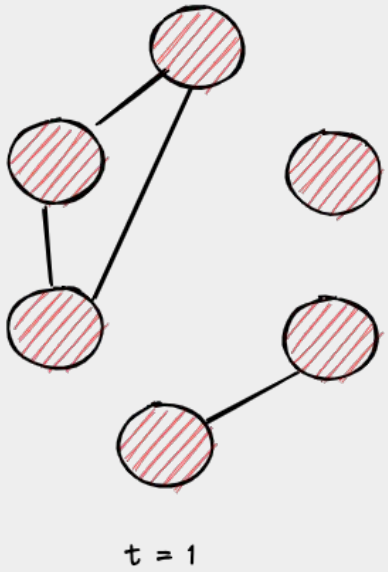

```python
# Add edges with layer attribute
G.add_edges_from(friendships, layer='friendship')
G.add_edges_from(professional_relationships, layer='professional')

# Adding edges with more attributes
G.add_edge('Alice', 'Bob', layer='friendship', weight=1)
G.add_edge('Alice', 'Charlie', layer='friendship', weight=1)
G.add_edge('Bob', 'Charlie', layer='friendship', weight=1)

G.add_edge('Alice', 'Diana', layer='professional', weight=2)
G.add_edge('Bob', 'Diana', layer='professional', weight=2)
G.add_edge('Charlie', 'Diana', layer='professional', weight=2)
```

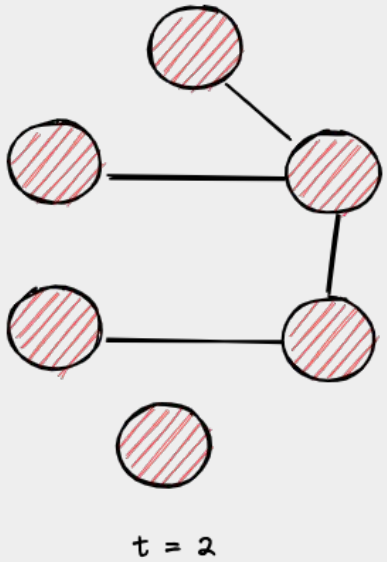


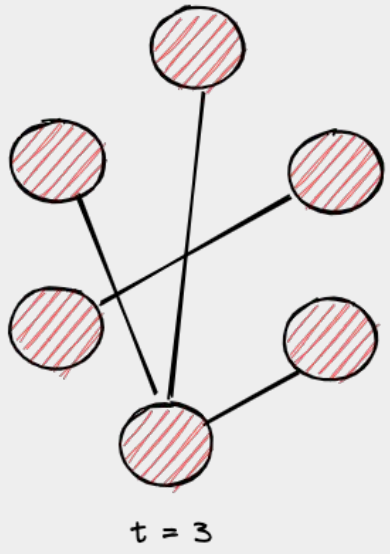

```python
# Positions for nodes
pos = nx.spring_layout(G, seed=42)  # Fixed seed for reproducibility

# Separate edges by layer
edges_friendship = [(u, v) for u, v, d in G.edges(data=True) if d['layer'] == 'friendship']
edges_professional = [(u, v) for u, v, d in G.edges(data=True) if d['layer'] == 'professional']

# Draw nodes
nx.draw_networkx_nodes(G, pos, node_size=500, node_color='lightblue')

# Draw edges for friendship layer
nx.draw_networkx_edges(G, pos, edgelist=edges_friendship, edge_color='blue', label='Friendship')

# Draw edges for professional layer
nx.draw_networkx_edges(G, pos, edgelist=edges_professional, edge_color='green', style='dashed',
label='Professional')

# Draw labels
nx.draw_networkx_labels(G, pos)
```

Extended Graphs (dynamic graph)

Imagine that your network represents a road graph. Nodes are intersections, and edges are stretches of the street connecting them. Roadworks might cut off a segment for a few days. If your network model cannot take this into account, you would end up telling drivers to use a road that is blocked, creating traffic jams and a lot of discomfort

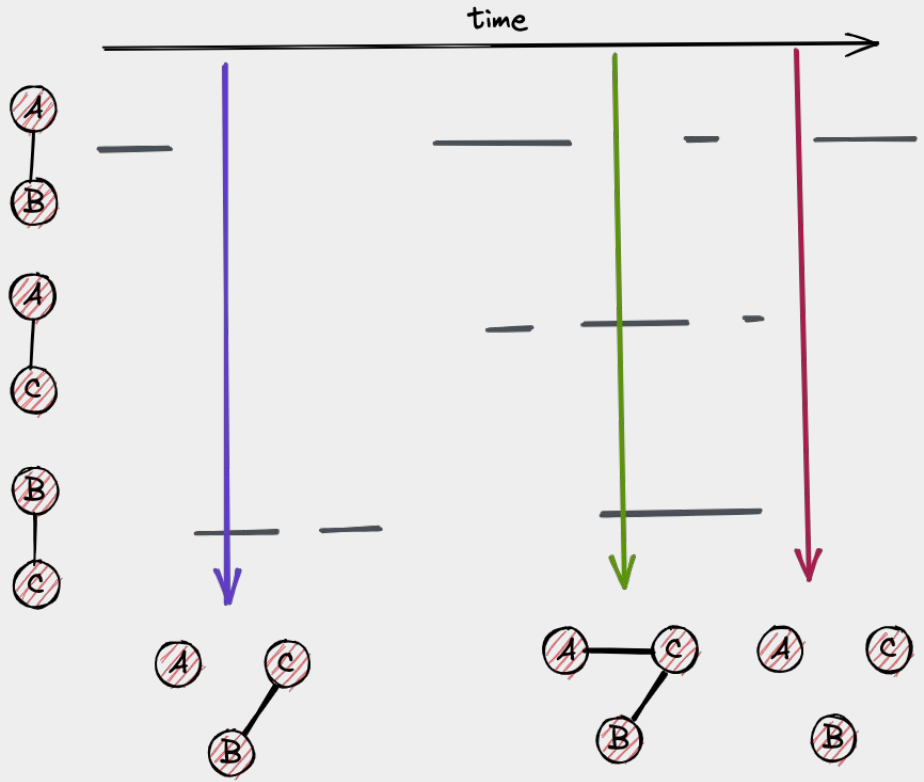




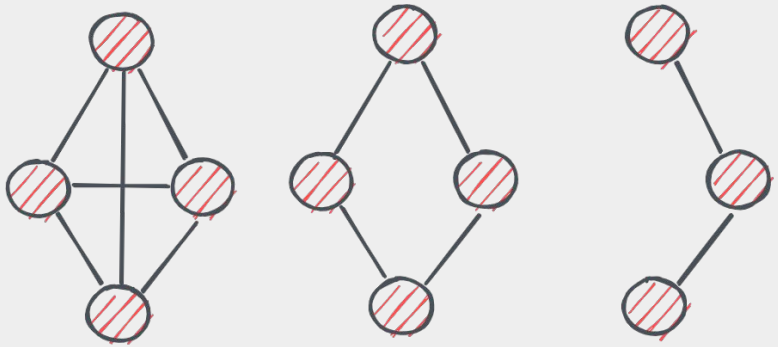


Extended Graphs (dynamic graph)

$$G = (G_1, G_2, \ldots, G_n)$$
$$G_i = (V_i, E_i)$$

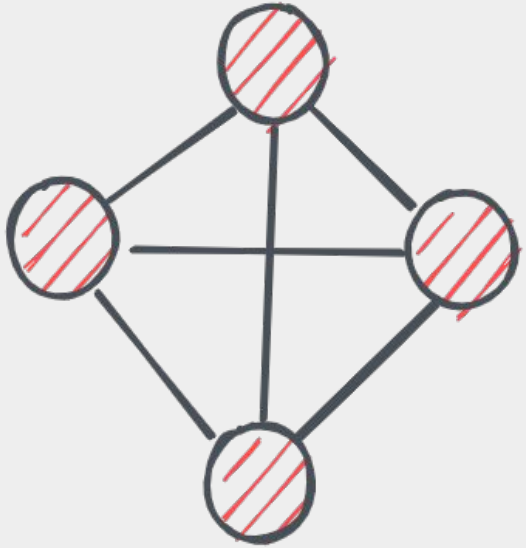


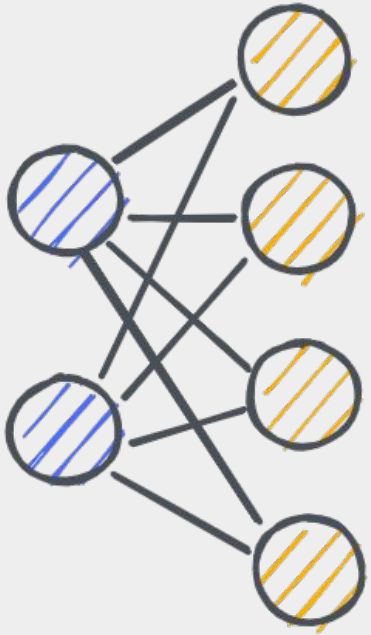
Density and Sparsity, Subnetworks



Density and Sparsity

The maximum number of links in a network is bounded by the possible number of distinct connections among the nodes of the system. The maximum number of links is therefore given by the number of pairs of nodes. A network with the maximum number of links, in which all possible pairs of nodes are connected by links, is called a **complete network.**





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

$$L_{max} = N_1 \times N_2$$

$$L_{max}^{directed} = N(N-1)$$

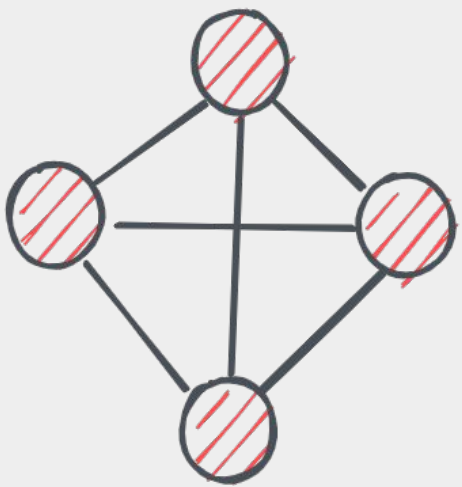
Density and Sparsity

$$d = \frac{L}{L_{max}} = \frac{2L}{N(N-1)}$$

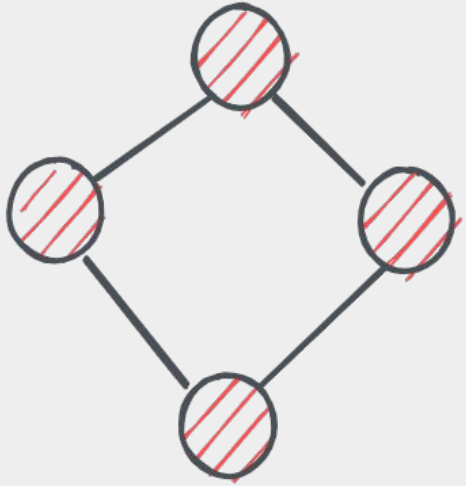
$$d = \frac{L}{L_{max}^{directed}} = \frac{L}{N(N-1)}$$

The fraction of possible links that actually exist, which is the same as the fraction of pairs of nodes that are actually connected, is called the **density of the network.**

$$d = \frac{L}{L_{max}}$$



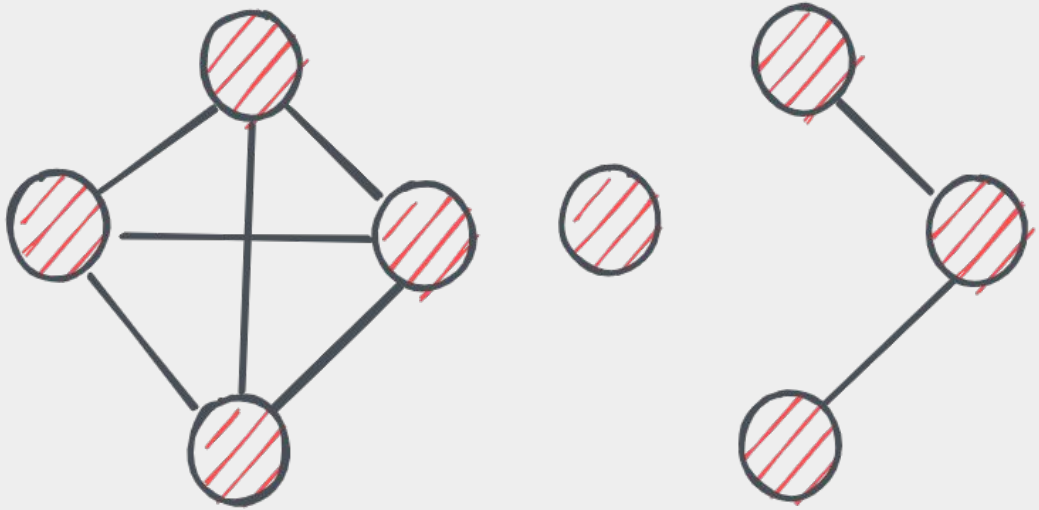
$d = 1$



$d < 1$

Density and Sparsity

We say that the **network is sparse** if the number of links grows proportionally to the number of nodes (L ~ N), or even slower. If instead the number of links grows faster, e.g. quadratically with network size (L ~ N^2), then we say that the **network is dense.**



$$L \ll L_{max} \rightarrow d \ll 1$$

- SNAP for C++
- SNAP for Python
- SNAP Datasets
- BIOSNAP Datasets
- What's new
- People
- Papers
- Projects
- Citing SNAP
- Links
- About
- Contact us

Open positions

Open research positions in **SNAP** group are available at **undergraduate, graduate** and **postdoctoral** levels.

Enron email network

Dataset information

Enron email communication network covers all the email communication within a dataset of around half million emails. This data was originally made public, and posted to the web, by the Federal Energy Regulatory Commission during its investigation. Nodes of the network are email addresses and if an address *i* sent at least one email to address *j*, the graph contains an undirected edge from *i* to *j*. Note that non-Enron email addresses act as sinks and sources in the network as we only observe their communication with the Enron email addresses.

The Enron email data was originally released by William Cohen at CMU.

Dataset statistics	
Nodes	36692
Edges	183831
Nodes in largest WCC	33696 (0.918)
Edges in largest WCC	180811 (0.984)
Nodes in largest SCC	33696 (0.918)
Edges in largest SCC	180811 (0.984)
Average clustering coefficient	0.4970
Number of triangles	727044
Fraction of closed triangles	0.03015
Diameter (longest shortest path)	11
90-percentile effective diameter	4.8

Source (citation)

- J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29--123, 2009.
- B. Klimmt, Y. Yang. Introducing the Enron corpus. CEAS conference, 2004.

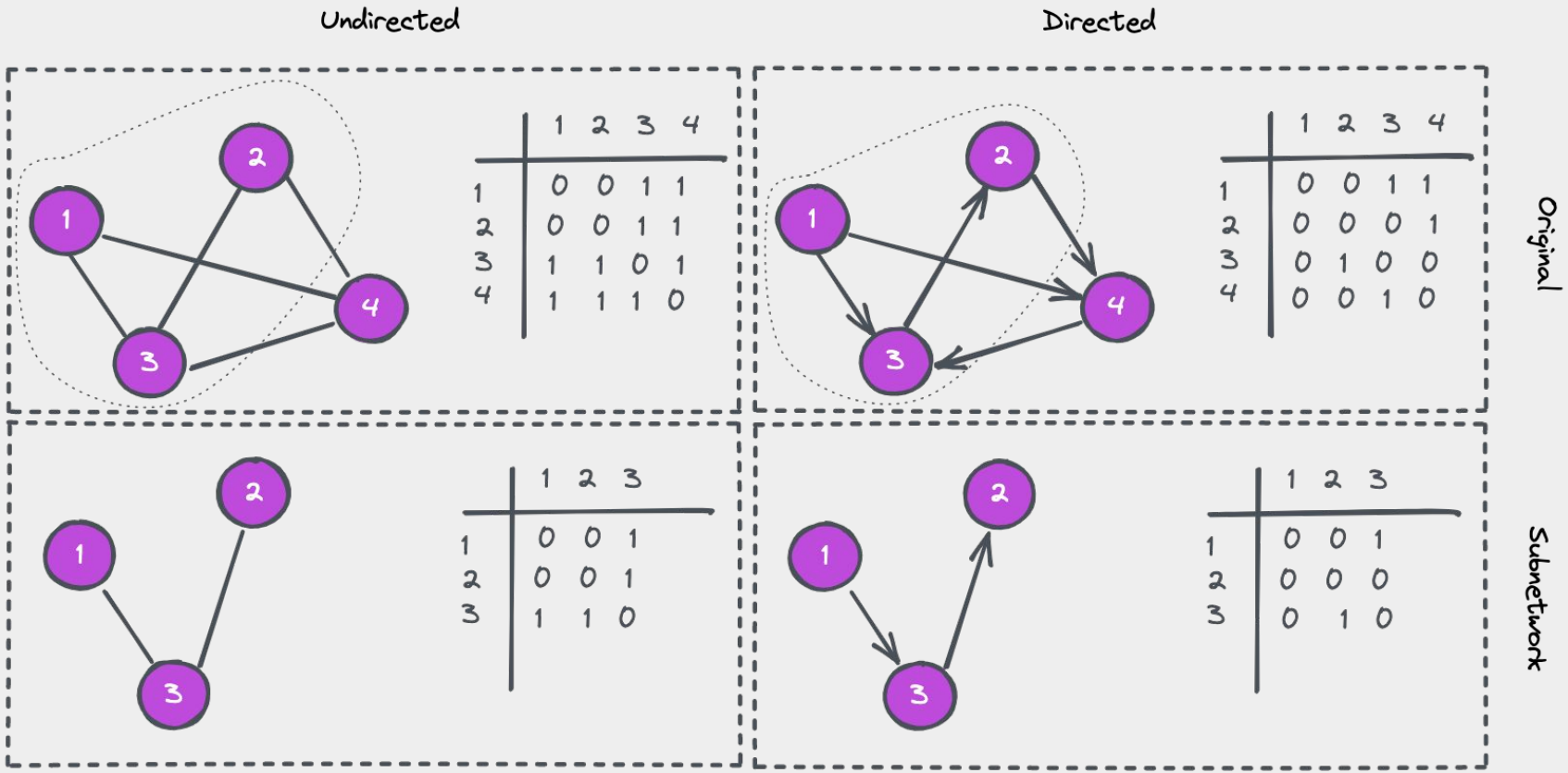
```python
import networkx as nx
import gzip
import urllib.request

# URL of the dataset (compressed file of Enron email network)
url = "https://snap.stanford.edu/data/email-Enron.txt.gz"
filename = "email-Enron.txt.gz"

# Download the file from the URL
urllib.request.urlretrieve(url, filename)

# Create an undirected graph
G = nx.Graph()

# Open and read the compressed file line by line
with gzip.open(filename, 'rt') as f:
    for line in f:
        if line.startswith("#"):
            continue  # Skip comment lines
        src, dst = map(int, line.strip().split())  # Extract source and destination node IDs
        G.add_edge(src, dst)  # Add an undirected edge between the nodes
```

```
# Directed graph (each unordered pair of nodes is saved once): Email-Enron.txt
# Enron email network (edge indicated that email was exchanged, undirected edges)
# Nodes: 36692 Edges: 367662
# FromNodeId    ToNodeId
0       1
1       0
1       2
1       3
1       4
1       5
1       6
1       7
1       8
1       9
1       10
1       11
1       12
1       13
1       14
1       15
1       16
1       17
1       18
1       19
1       20
```

```python
# Calculate number of nodes (vertices)
num_nodes = G.number_of_nodes()

# Calculate number of edges (links)
num_edges = G.number_of_edges()

# Calculate network density
density = nx.density(G)

manual_density = 2 * num_edges / (num_nodes * (num_nodes - 1))

# Print basic network statistics
print(f"Number of nodes: {num_nodes}")
print(f"Number of edges: {num_edges}")
print(f"Network density: {density:.6f}")
print(f"Network density calc: {manual_density:.6f}")

Number of nodes: 36692
Number of edges: 183831
Network density: 0.000273
Network density calc: 0.000273
```

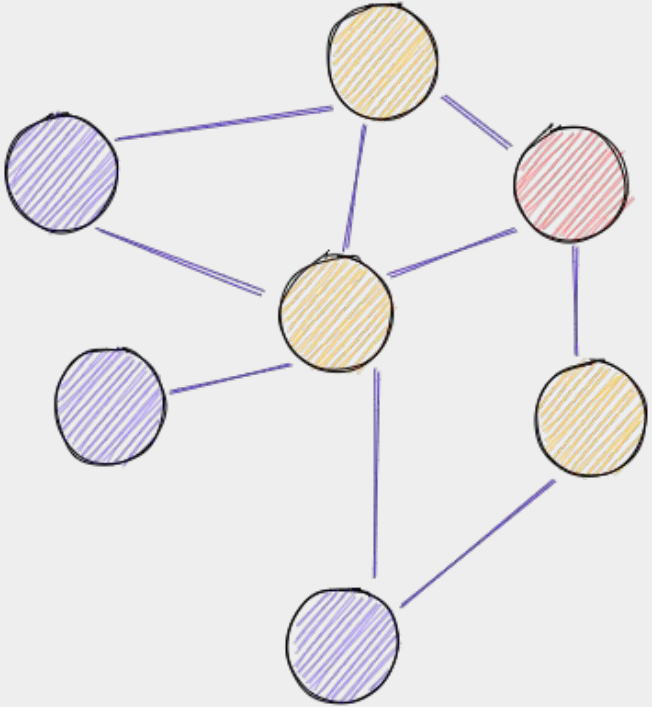
Network	Type	Nodes (N)	Links (L)	Density (d)	Average degree ($\langle k \rangle$)
Facebook Northwestern Univ.		10,567	488,337	0.009	92.4
IMDB movies and stars		563,443	921,160	0.000006	3.3
IMDB co-stars	W	252,999	1,015,187	0.00003	8.0
Twitter US politics	DW	18,470	48,365	0.0001	2.6
Enron email	DW	87,273	321,918	0.00004	3.7
Wikipedia math	D	15,220	194,103	0.0008	12.8
Internet routers		190,914	607,610	0.00003	6.4
US air transportation		546	2,781	0.02	10.2
World air transportation		3,179	18,617	0.004	11.7
Yeast protein interactions		1,870	2,277	0.001	2.4
C. elegans brain	DW	297	2,345	0.03	7.9
Everglades ecological food web	DW	69	916	0.2	13.3

Subnetwork

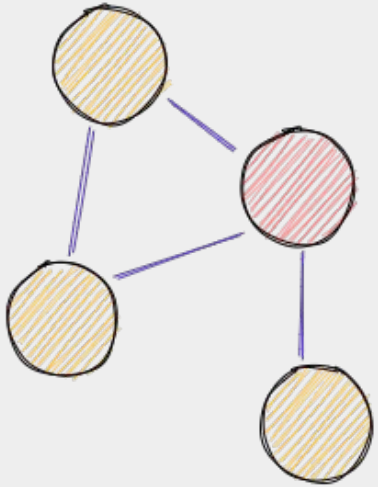

Undirected
Directed
Original
Subnetwork
1 2 3 4
1 0 0 1 1
2 0 0 1 1
3 1 1 0 1
4 1 1 1 0
1 2 3 4
1 0 0 1 1
2 0 0 0 1
3 0 1 0 0
4 0 0 1 0
1 2 3
1 0 0 1
2 0 0 1
3 1 1 0
1 2 3
1 0 0 1
2 0 0 0
3 0 1 0

Subnetwork

A special type of subnetwork is the ego network of a node, which is the subnetwork consisting of the chosen node — called the ego — and its neighbors. Ego networks are often studied in social network analysis.



Original Network



Ego Network

- SNAP for C++
- SNAP for Python
- SNAP Datasets
- BIOSNAP Datasets
- What's new
- People
- Papers
- Projects
- Citing SNAP
- Links
- About
- Contact us

Open positions

Open research positions in **SNAP** group are available at **undergraduate, graduate** and **postdoctoral** levels.

Patent citation network

Dataset information

U.S. patent dataset is maintained by the National Bureau of Economic Research. The data set spans 37 years (January 1, 1963 to December 30, 1999), and includes all the utility patents granted during that period, totaling 3,923,922 patents. The citation graph includes all citations made by patents granted between 1975 and 1999, totaling 16,522,438 citations. For the patents dataset there are 1,803,511 nodes for which we have no information about their citations (we only have the in-links).

The data was originally released by NBER.

Dataset statistics	
Nodes	3774768
Edges	16518948
Nodes in largest WCC	3764117 (0.997)
Edges in largest WCC	16511741 (1.000)
Nodes in largest SCC	1 (0.000)
Edges in largest SCC	0 (0.000)
Average clustering coefficient	0.0757
Number of triangles	7515023
Fraction of closed triangles	0.02343
Diameter (longest shortest path)	22
90-percentile effective diameter	9.4

Source (citation)

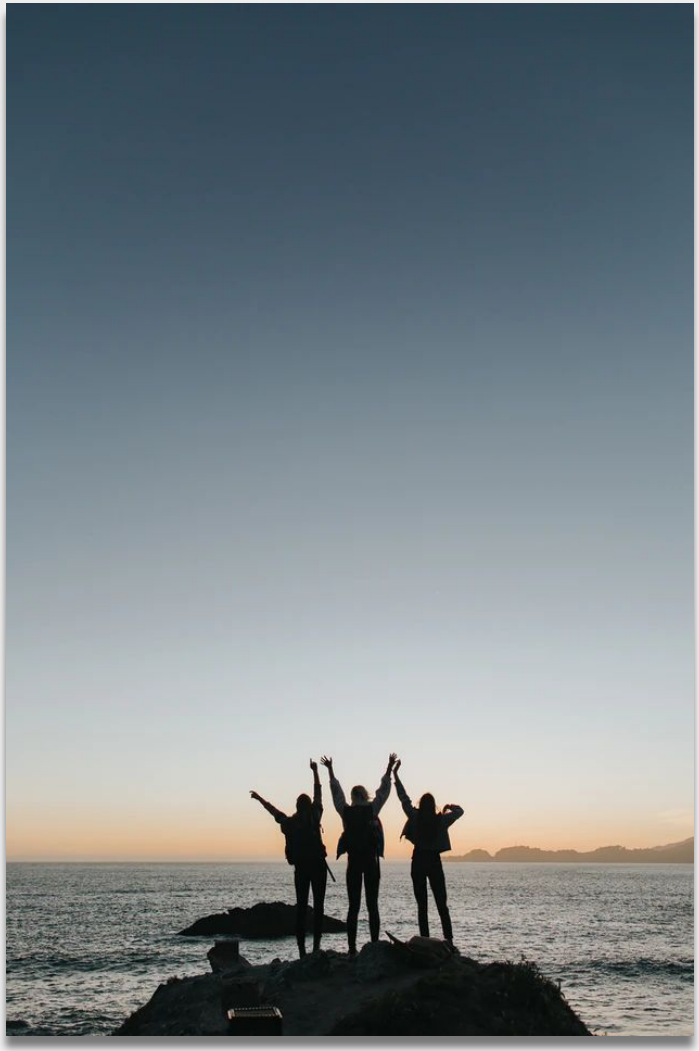
- J. Leskovec, J. Kleinberg and C. Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2005.

```python
import networkx as nx
import gzip
import urllib.request
import random

# Step 1: Download the citation network dataset
url = "https://snap.stanford.edu/data/cit-Patents.txt.gz"
filename = "cit-Patents.txt.gz"
urllib.request.urlretrieve(url, filename)
```

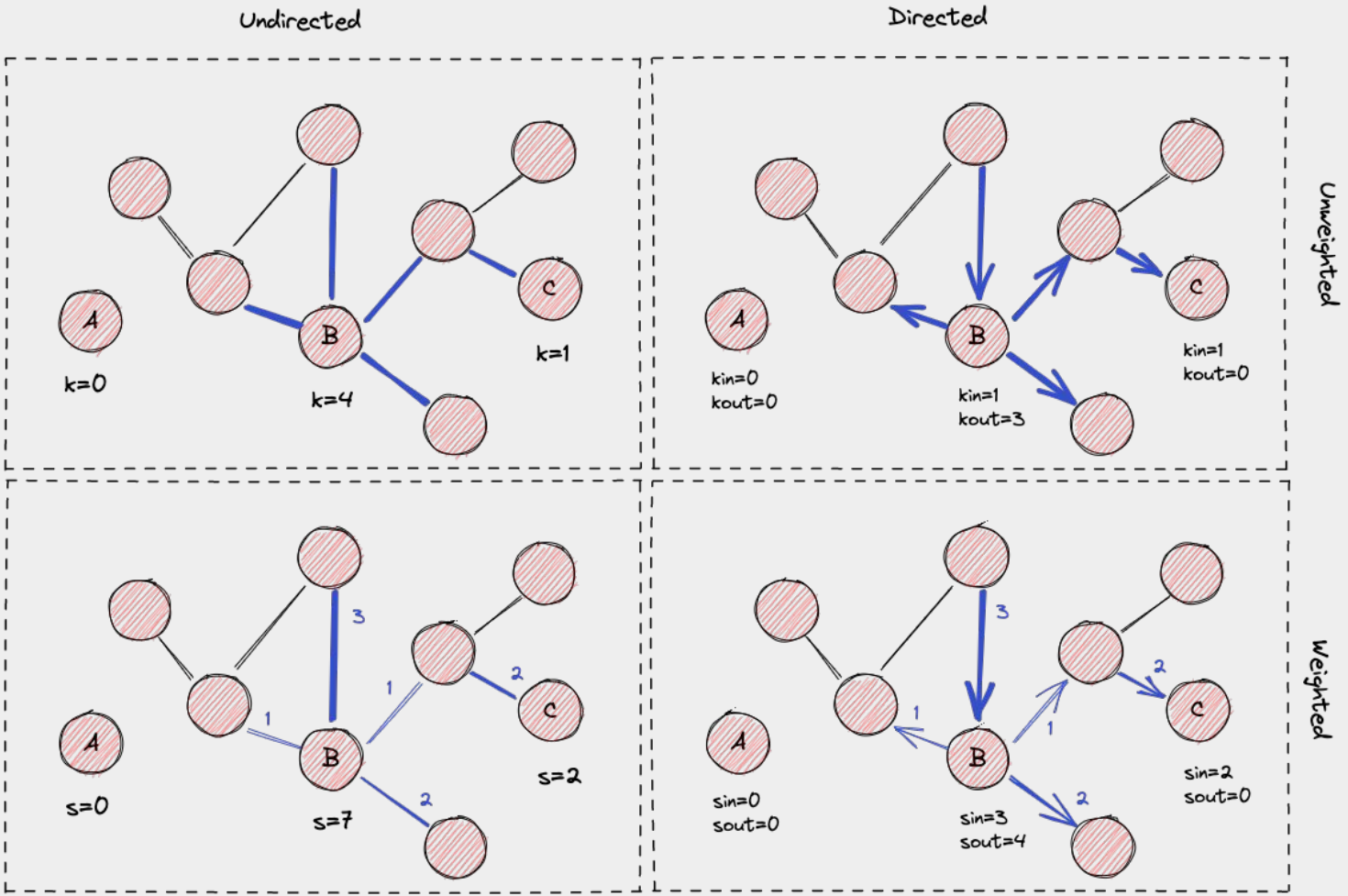
```python
# Step 2: Create a directed graph and load data
G = nx.DiGraph()
with gzip.open(filename, 'rt') as f:
    for line in f:
        if line.startswith("#"):
            continue  # Skip comment lines
        src, dst = map(int, line.strip().split())
        G.add_edge(src, dst)

print(f"Full graph: {G.number_of_nodes()} nodes,{G.number_of_edges()} edges, {nx.density(G)}")
```

```
Full graph: 3774768 nodes,16518948 edges, 1.1593163789181623e-06
```

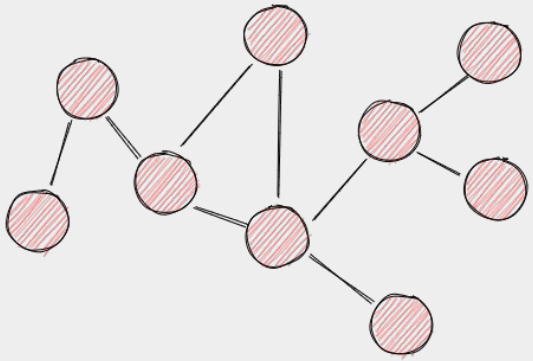
```python
# Step 3: Randomly sample 10% of the nodes
num_sample = int(0.10 * G.number_of_nodes())
sample_nodes = random.sample(list(G.nodes()), num_sample)

# Step 4: Create the subgraph induced by the sampled nodes
subG = G.subgraph(sample_nodes)

# Step 5: Compute and print basic metrics
print(f"\nSampled subgraph (10% of nodes):")
print(f"Nodes: {subG.number_of_nodes()}")
print(f"Edges: {subG.number_of_edges()}")
print(f"Density: {nx.density(subG):.6f}")

Sampled subgraph (10% of nodes):
Nodes: 377476
Edges: 164582
Density: 0.000001
```



Degree and Network Representation


Undirected
Directed
Unweighted
Weighted
A
B
C
k=0
k=4
k=1
kin=0
kout=0
kin=1
kout=3
kin=1
kout=0
s=0
s=7
s=2
sin=0
sout=0
sin=3
sout=4
sin=2
sout=0
1
2
3

The average degree of a network is defined as:



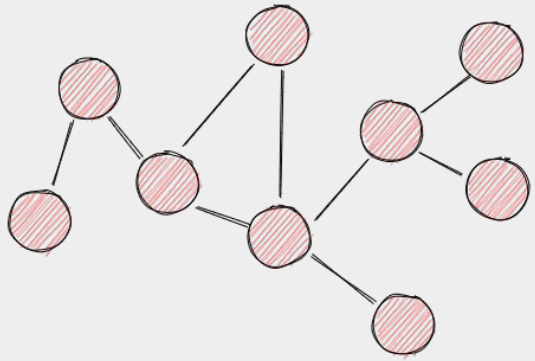
$$\langle k \rangle = \frac{\sum_i k_i}{N}$$

Since each link contributes to the degree of two nodes in an undirected network:

$$\langle k \rangle = \frac{2L}{N} = d(N-1) \qquad d = \frac{2L}{N(N-1)}$$

The average degree of a network is defined as:

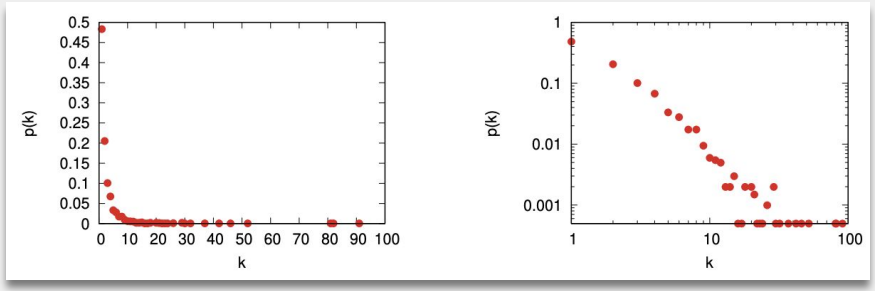
$$\langle k \rangle = \frac{2L}{N} = d(N-1)$$

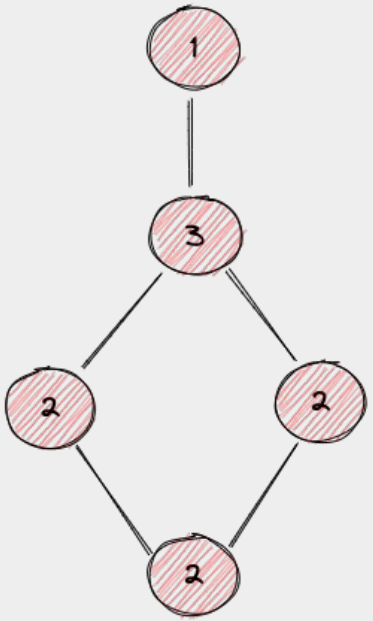


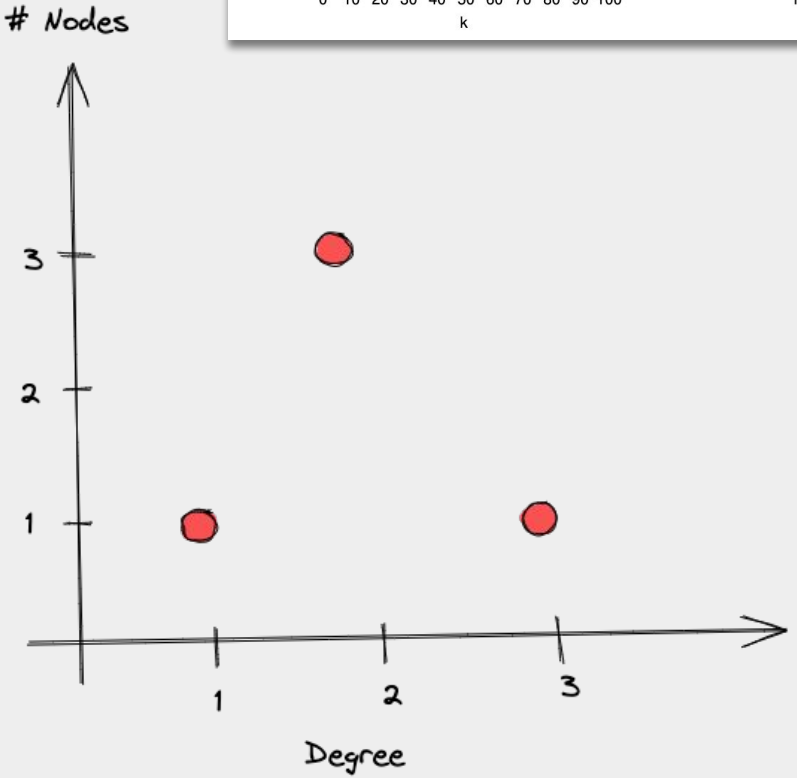
$$d = \frac{\langle k \rangle}{N-1}$$

This makes sense because the **maximum possible degree** of a node is $k_{max} = N - 1$, obtained when the node is connected to every other node. Intuitively, the **density is the ratio between the average and maximum degree** .

Degree Distribution







- SNAP for C++
- SNAP for Python
- SNAP Datasets
- BIOSNAP Datasets
- What's new
- People
- Papers
- Projects
- Citing SNAP
- Links
- About
- Contact us

Open positions

Open research positions in **SNAP** group are available at **undergraduate, graduate** and **postdoctoral** levels.

Twitch Gamers Social Network

Dataset information

A social network of Twitch users which was collected from the public API in Spring 2018. Nodes are Twitch users and edges are mutual follower relationships between them. The graph forms a single strongly connected component without missing attributes. The machine learning tasks related to the graph are count data regression and node classification. There are 6 specific tasks:

- Explicit content streamer identification.
- Broadcaster language prediction.
- User lifetime estimation.
- Churn prediction.
- Affiliate status identification.
- View count estimation.

Twitch Gamers paper: arxiv.org
Twitch Gamers project: Github

Dataset statistics	
Directed	No.
Node features	No.
Edge features	No.
Node labels	Yes.
Temporal	No.
Nodes	168,114
Edges	6,797,557
Density	0.0005
Transitvity	0.0184

Possible tasks
Node level regression
Binary node classification

```python
import networkx as nx
import urllib.request
import zipfile
import os
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from collections import Counter


# Step 1: Download and unzip the dataset
url = "https://snap.stanford.edu/data/twitch_gamers.zip"
zip_filename = "twitch_gamers.zip"
urllib.request.urlretrieve(url, zip_filename)
```

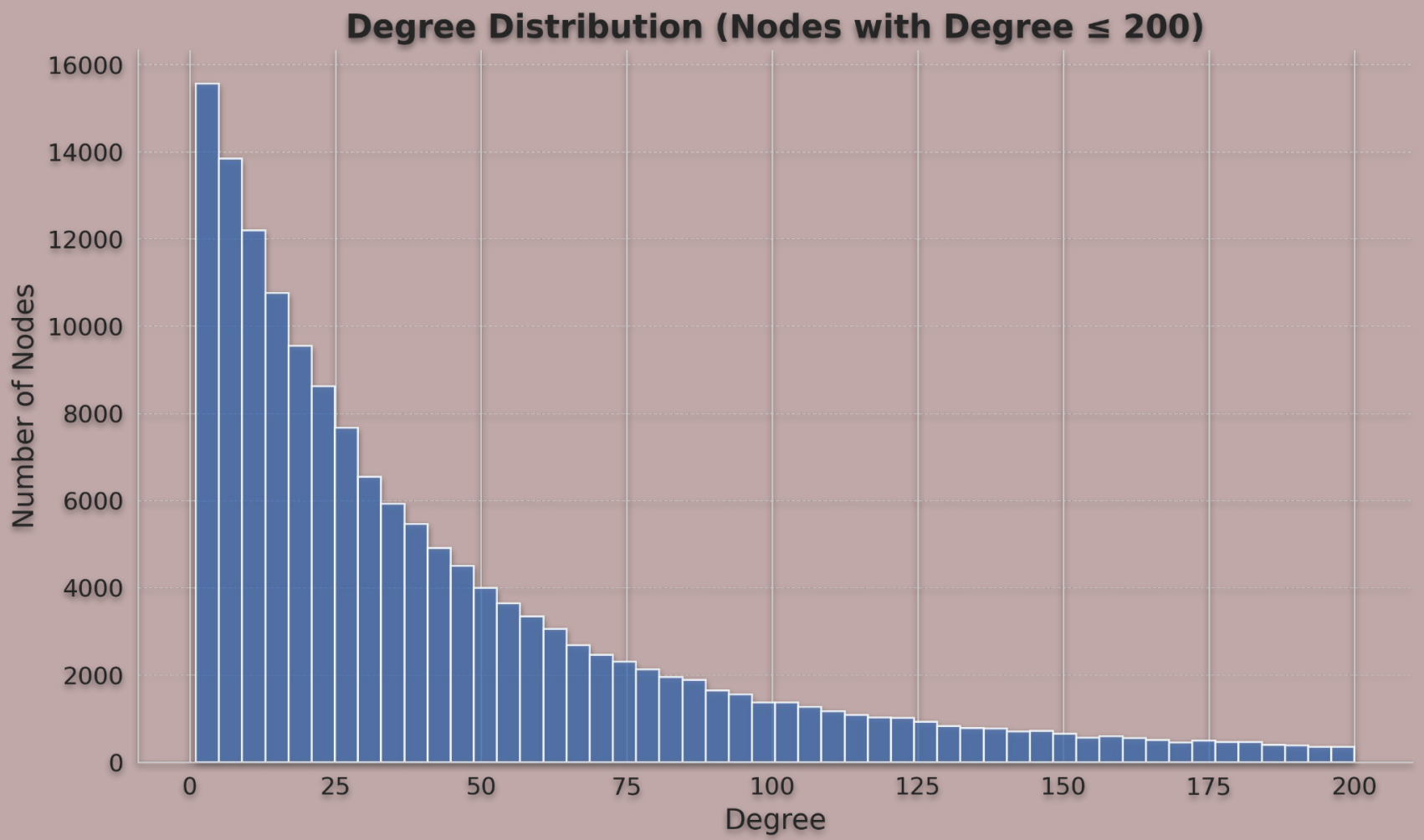
```python
# Step 2: Extract the ZIP file
with zipfile.ZipFile(zip_filename, 'r') as zip_ref:
    zip_ref.extractall("twitch_data")

# Step 3: Load the edge list from CSV using pandas
edge_file = "twitch_data/large_twitch_edges.csv"
df = pd.read_csv(edge_file)

# Step 4: Create an undirected graph from the edge list
G = nx.from_pandas_edgelist(df, source='numeric_id_1',
                            target='numeric_id_2')

# Step 5: Print basic info
print(f"Number of nodes: {G.number_of_nodes()}")
print(f"Number of edges: {G.number_of_edges()}")

Number of nodes: 168114
Number of edges: 6797557
```

```python
# Step 4: Compute degrees
degrees = [deg for node, deg in G.degree()]

# Step 5: Show degree statistics
print(f"Max degree: {max(degrees)}")
print(f"Average degree: {sum(degrees)/len(degrees):.2f}")
print(f"Average degree alternative: {nx.density(G)*(G.number_of_nodes()-1):.2f}")
```

```
Max degree: 35279
Average degree: 80.87
Average degree alternative: 80.87
```

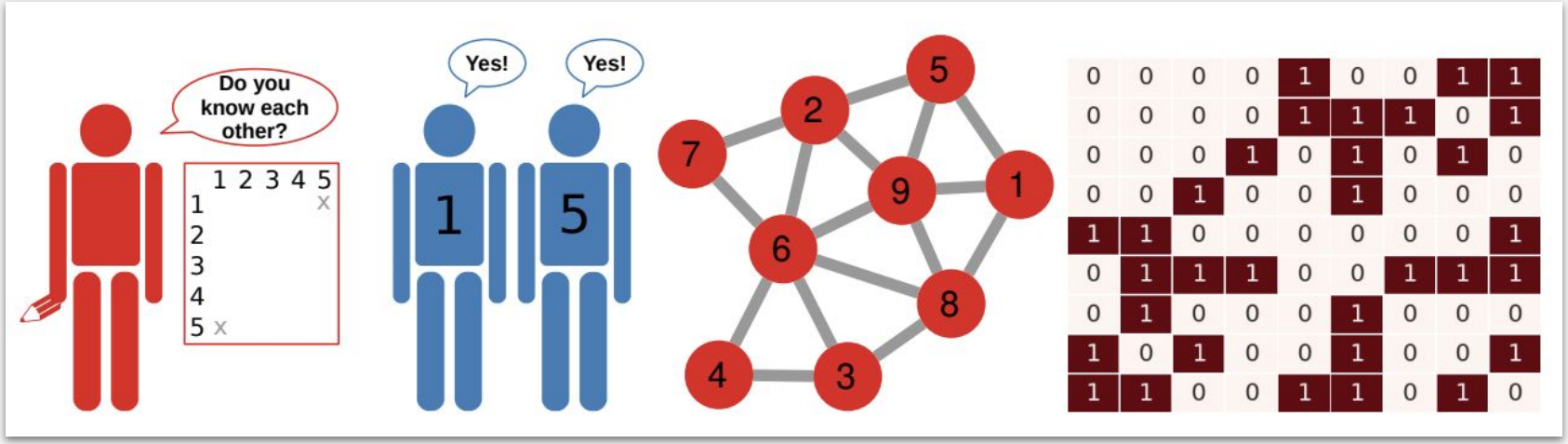
```python
# Count the frequency of each degree
degree_count = Counter(degrees)

# Sort by degree (optional, for readability)
sorted_degree_count = dict(sorted(degree_count.items()))

# Print the histogram values
print("Degree\tNumber of Nodes")
for degree, count in sorted_degree_count.items():
    print(f"{degree}\t{count}")
```

```
Degree  Number of Nodes
1   4037
2   3891
3   3826
4   3811
5   3585
6   3572
7   3419
8   3273
9   3217
10  3166
```


Degree Distribution (Nodes with Degree ≤ 200)
Number of Nodes
16000
14000
12000
10000
8000
6000
4000
2000
0
0
25
50
75
100
125
150
175
200
Degree

Network Representation

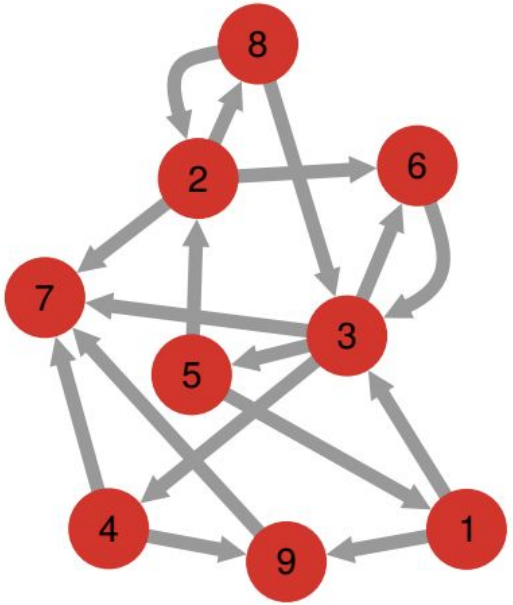
The **adjacency matrix** is the basic representation of a graph as a **matrix** . Each row/column corresponds to a node. Each cell represents an edge, set to one if the edge exists, and zero otherwise.



Network Representation

A non-symmetric adjacency matrix corresponding to a directed graph

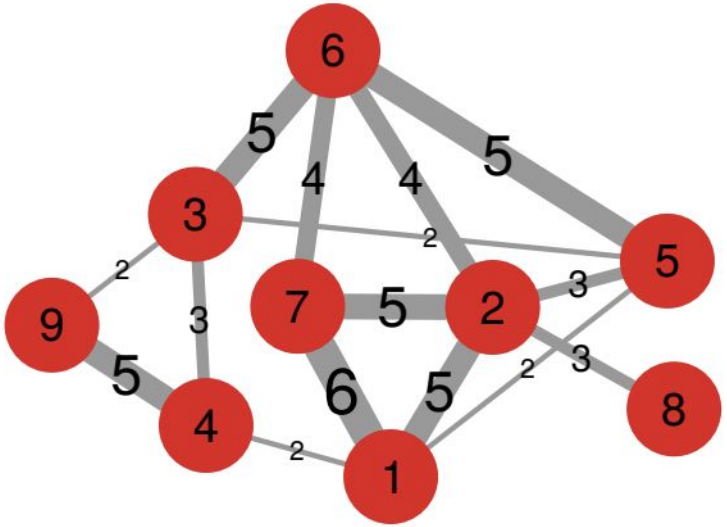
0	0	1	0	0	0	0	0	1
0	0	0	0	0	1	1	1	0
0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	1	0	1
1	1	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0



Network Representation

A non-binary adjacency matrix corresponding a weighted graph

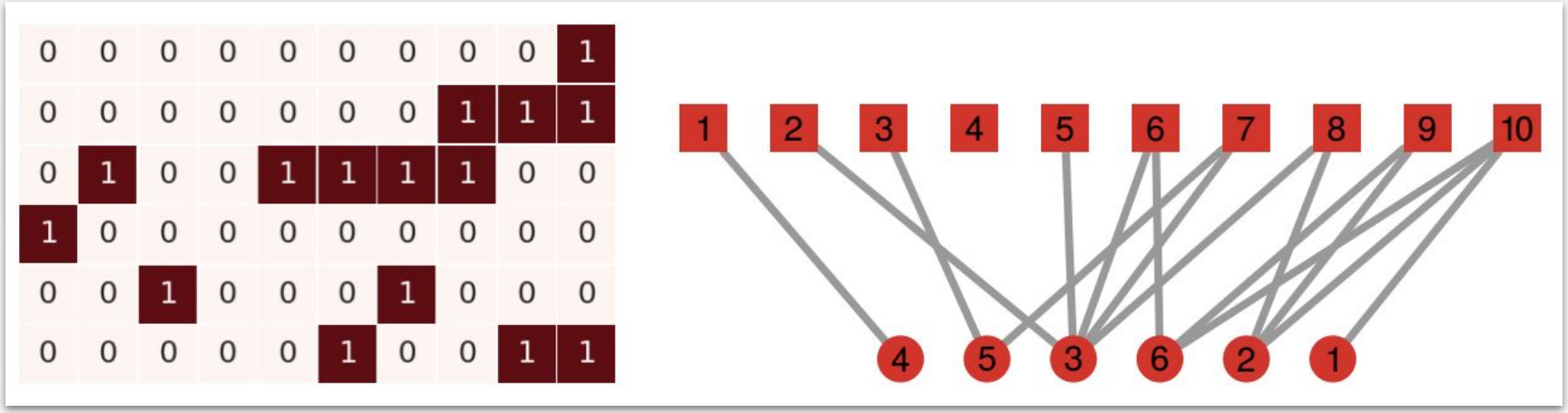
0	5	0	2	2	0	6	0	0
5	0	0	0	3	4	5	3	0
0	0	0	3	2	5	0	0	2
2	0	3	0	0	0	0	0	5
2	3	2	0	0	5	0	0	0
0	4	5	0	5	0	4	0	0
6	5	0	0	0	4	0	0	0
0	3	0	0	0	0	0	0	0
0	0	2	5	0	0	0	0	0



Network Representation

A non-square adjacency matrix corresponding a bipartite graph

$$\begin{array}{c} \\ |V_1| \\ |V_2| \end{array} \begin{array}{c} \begin{array}{cc} |V_1| & |V_2| \end{array} \\ \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \end{array}$$



Contents

MICHELE COSCIA

IT UNIVERSITY OF COPENHAGEN

THE ATLAS FOR THE ASPIRING NETWORK SCIENTIST

https://www.networkatlas.eu/