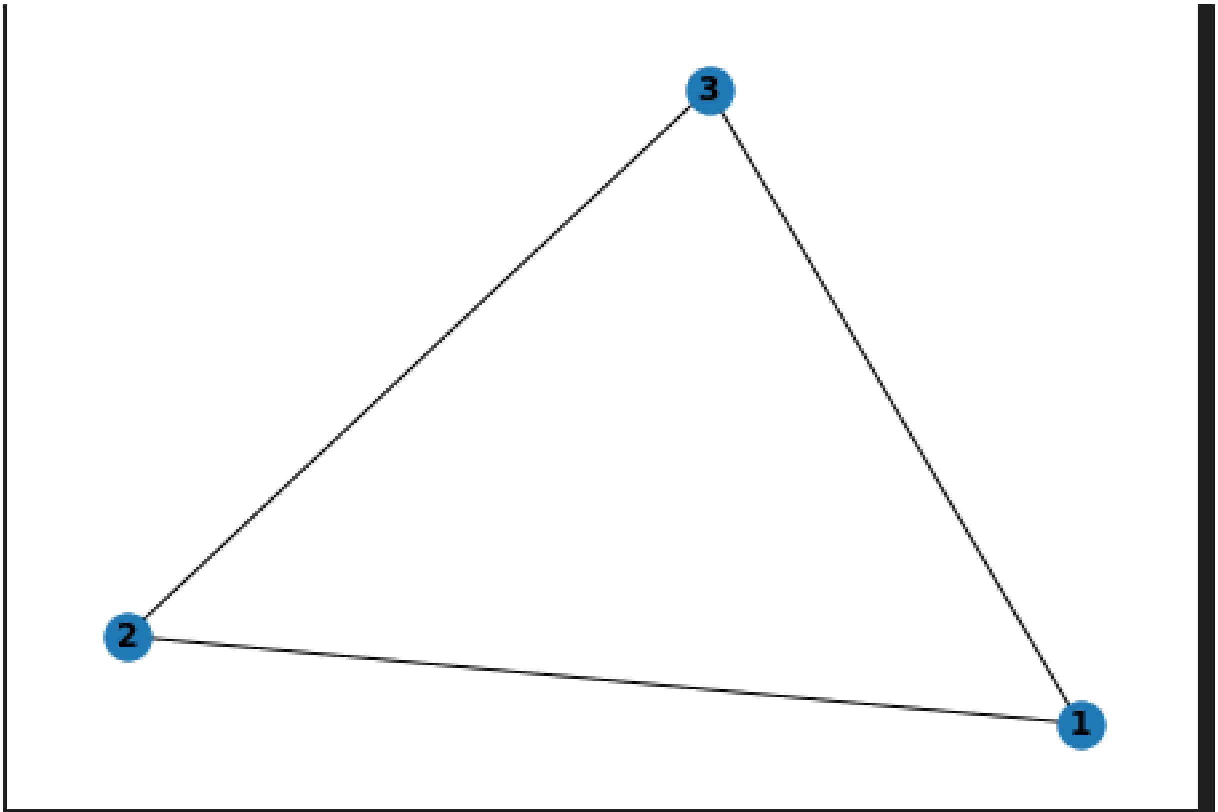
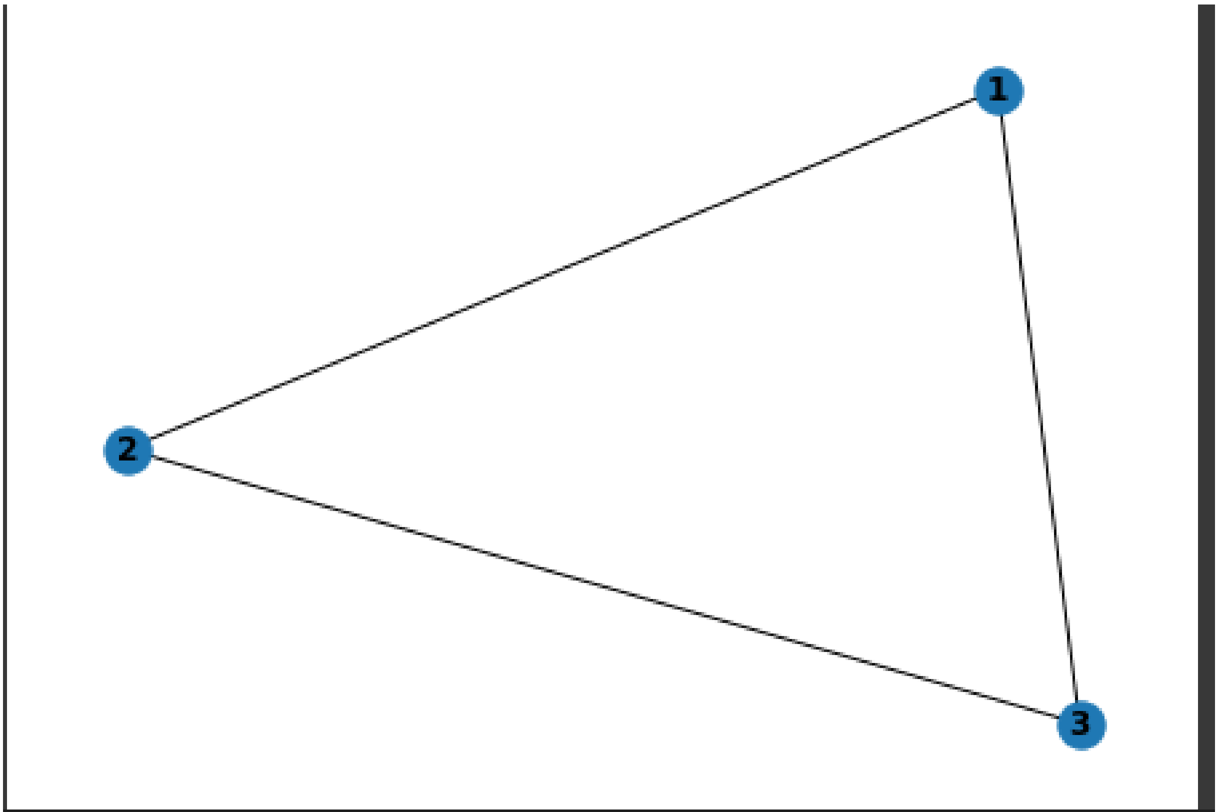
Experiment 1 : Topological searches and analyses for complex network

```
# Simple example
import networkx as nx      # nx can be seemed as an alias of networkx mo
dule
import numpy as np
G = nx.Graph()
import numpy as np
import random
import cmath
import pandas as pd
G.clear()
G.add_nodes_from([1,2,3])
G.add_edge(3,2)
G.add_edge(1,2)
G.add_edge(1,3)
nx.draw(G, with_labels=True, font_weight='bold')
```



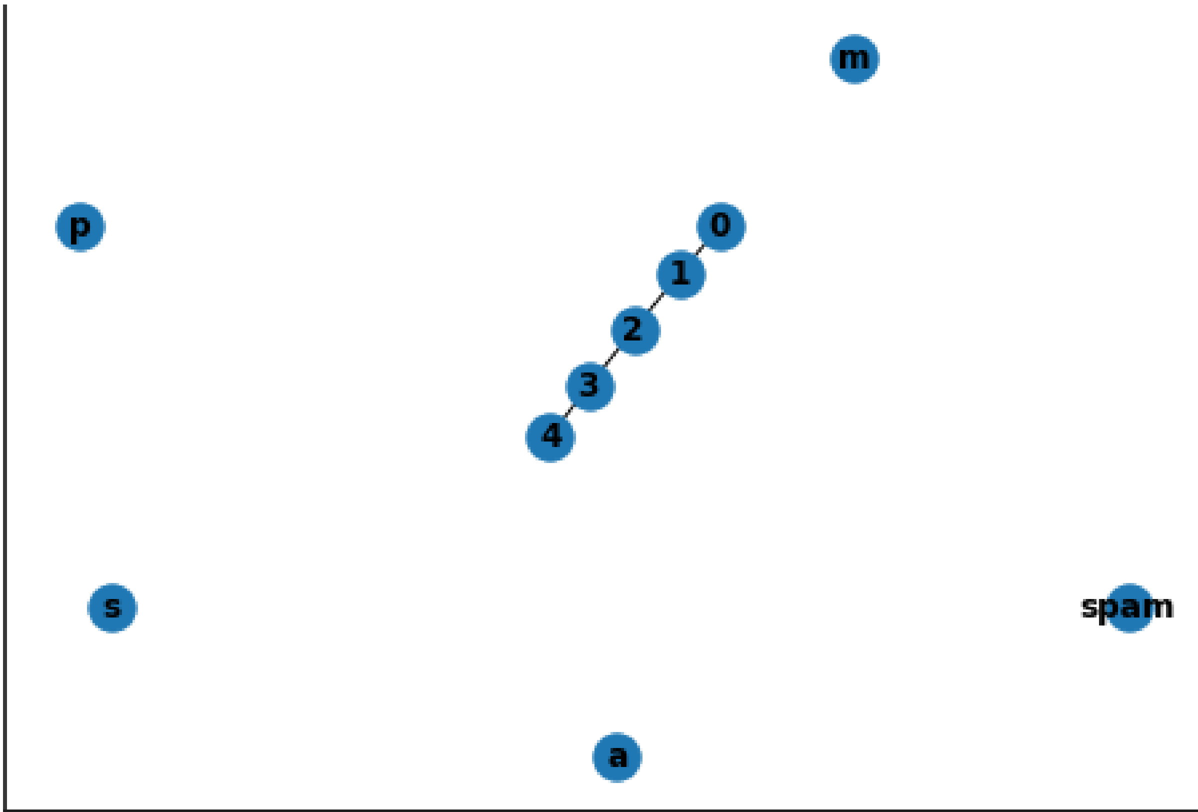
```
G.remove_edge(1,2)

print(G.edges())
G.add_edges_from([(1,2), (1,3)]) # add edges from a edge list
print(G.edges())
G.add_edges_from([(1,2)]) # adding an edge that is already present
print(G.edges()) # No difference! NetworkX quietly ignores, instead of overwriting the edge if it already exists.
nx.draw(G, with_labels=True, font_weight='bold')
```

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



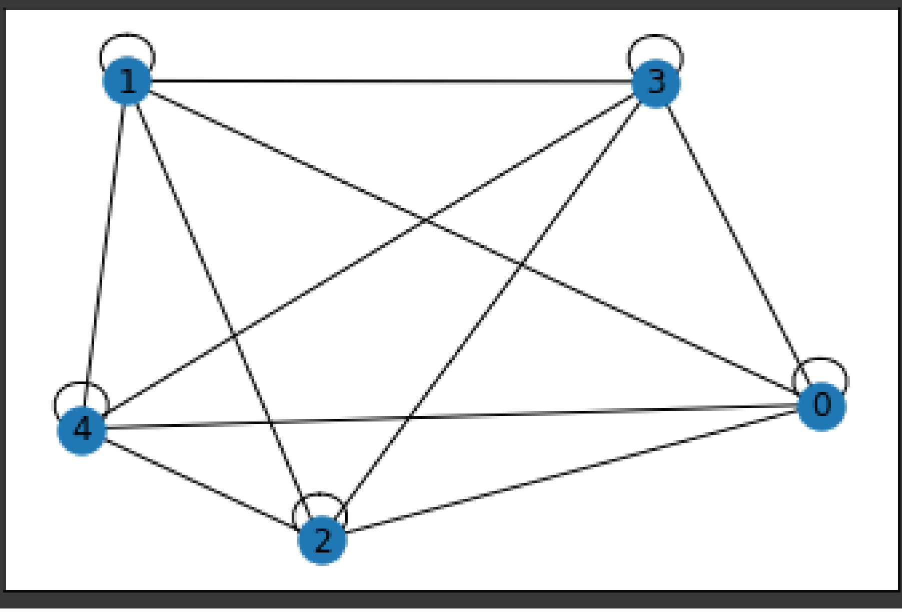

```
G=nx.path_graph(5) # 0 -> 1 -> 2 -> 3 -> 4
G.add_node("spam")  # add one node called "spam"
G.add_nodes_from("spam") # add 4 nodes: 's', 'p', 'a', 'm', since strin
g "spam" in python is actually a list ['s', 'p', 'a', 'm']
print(G.nodes())
print('number of edges in the graph:', G.number_of_edges())
print('edges in the graph:', G.edges())
print('degree counts per node:', G.degree())
nx.draw(G, with_labels=True, font_weight='bold')
```

```
[0, 1, 2, 3, 4, 'spam', 's', 'p', 'a', 'm']
number of edges in the graph: 4
edges in the graph: [(0, 1), (1, 2), (2, 3), (3, 4)]
degree counts per node: [(0, 1), (1, 2), (2, 2), (3, 2), (4, 1), ('spam', 0), ('s', 0), ('p', 0), ('a', 0), ('m', 0)]
```

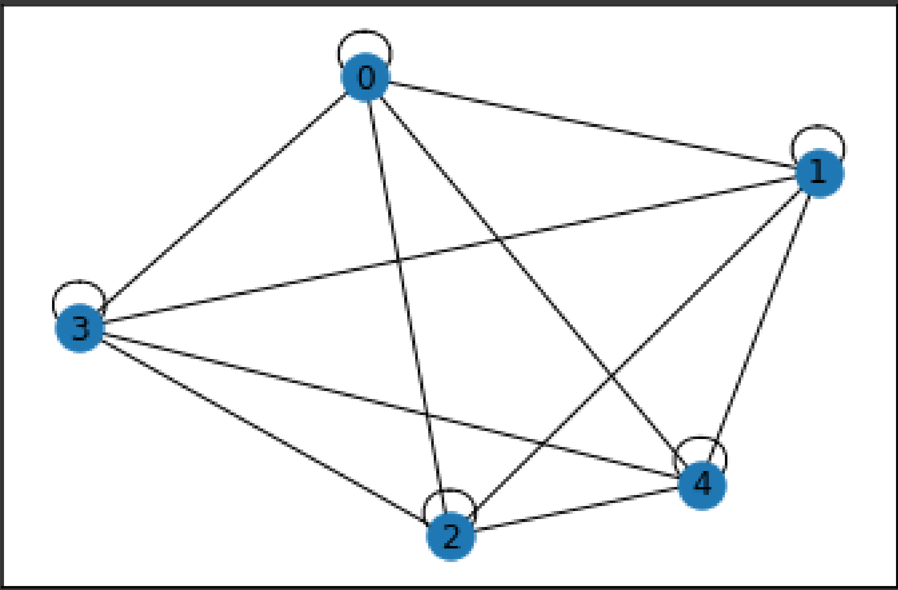



```
adj = np.array([  [1,  1,  1,1,1],
                  [1,  1,  1,1,1],
                  [1,  1,  1,1,1],
                  [1,  0,  1,1,1],
                  [1,1,1,1,1]])
G=nx.from_numpy_matrix(adj)
nx.draw_networkx(G, with_labels=True)
print(nx.to_dict_of_lists(G))
```

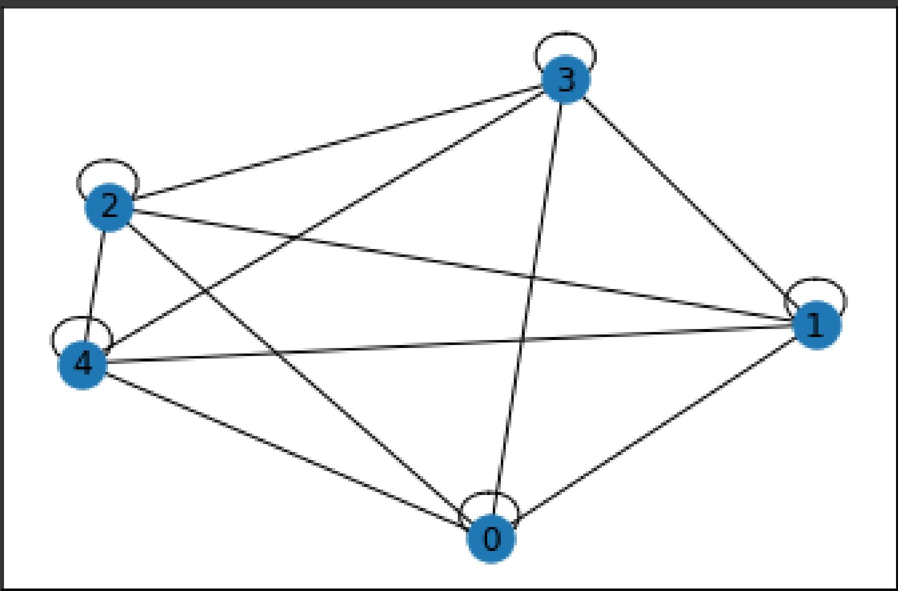
```
{0: [0, 1, 2, 3, 4], 1: [0, 1, 2, 3, 4], 2: [0, 1, 2, 3, 4], 3: [0, 1, 2,
3, 4], 4: [0, 1, 2, 3, 4]}
```



```
A=nx.to_numpy_matrix(G) # Incidence matrix A.shape
print(A)
H=nx.from_numpy_matrix(A)
print(H)
nx.draw_networkx(H, with_labels=True)
```




```
[[ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1. 10.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1. 10.  1.  1.]]
Graph with 5 nodes and 15 edges
```




```
G.add_node(11)     # add a single node
print(G.nodes())
nx.draw(G, with_labels=True, font_weight='bold')
G.add_nodes_from([12,13])     # add a list of nodes
print(G.nodes())
print('number of nodes in the graph:', G.number_of_nodes())
```

```
nx.draw(G, with_labels=True, font_weight='bold')
```

Directed Graphs, Multigraphs and Visualization in Networkx

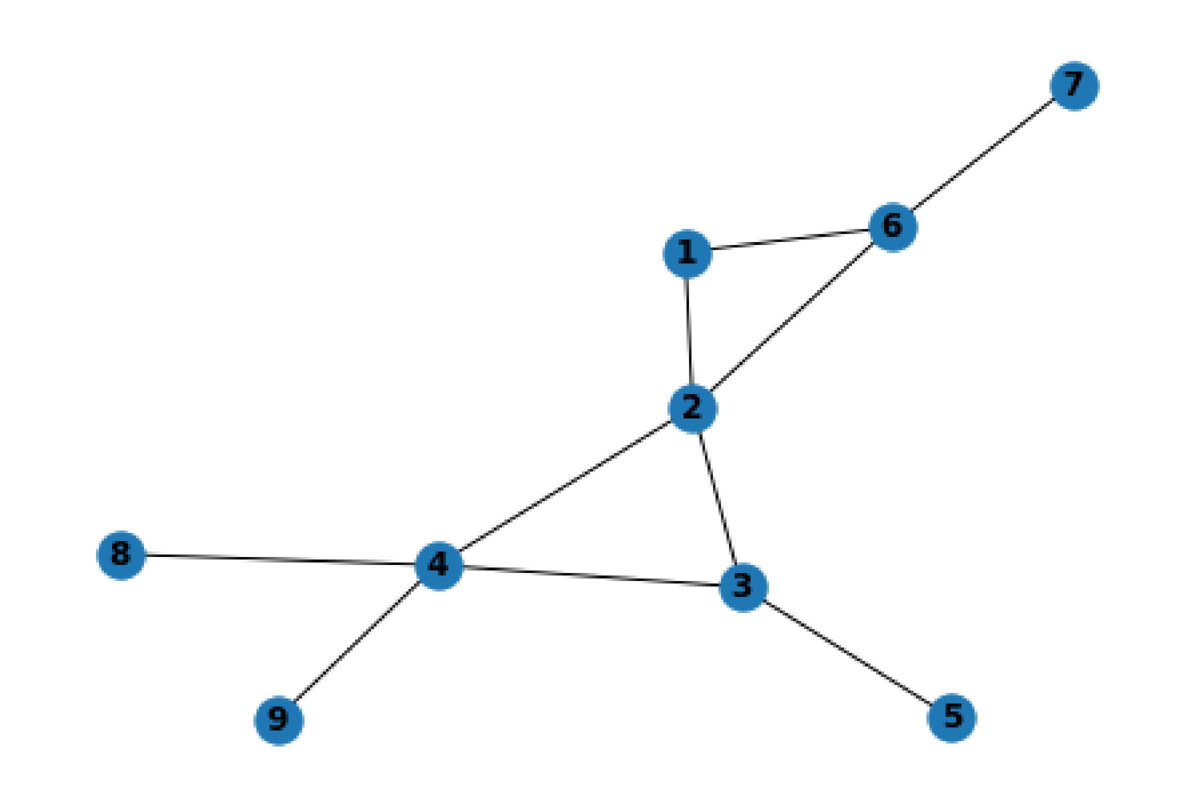
```
# Python program to create an undirected
# graph and add nodes and edges to a graph

# To import package
import networkx

# To create an empty 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(7)
G.add_node(9)

# To add an edge
# Note graph is undirected
# Hence order of nodes in edge doesn't matter
G.add_edge(1,2)
G.add_edge(3,1)
G.add_edge(2,4)
G.add_edge(4,1)
G.add_edge(9,1)
G.add_edge(1,7)
G.add_edge(2,9)

# To get all the nodes of a graph
node_list = G.nodes()
print("#1  all the nodes of a graph ", node_list)

# To get all the edges of a graph
edge_list = G.edges()
print("#2  all the edges of a graph", edge_list)

# To remove a node of a graph
G.remove_node(3)
node_list = G.nodes()
print("#3 node list" , node_list)

# To remove an edge of a graph
G.remove_edge(1,2)
```

```
edge_list = G.edges()
print("#4  edgelist" , edge_list)

# To find number of nodes
n = G.number_of_nodes()
print("#5  number of nodes ", n)

# To find number of edges
m = G.number_of_edges()
print("#6  number of edges", m)

# To find degree of a node
# d will store degree of node 2
d = G.degree(2)
print("#7   degree of node(2)", d)

# To find all the neighbor of a node
neighbor_list = G.neighbors(2)
print("#8 neighbor of a node(2)",list(neighbor_list))

#To delete all the nodes and edges
G.clear()
```

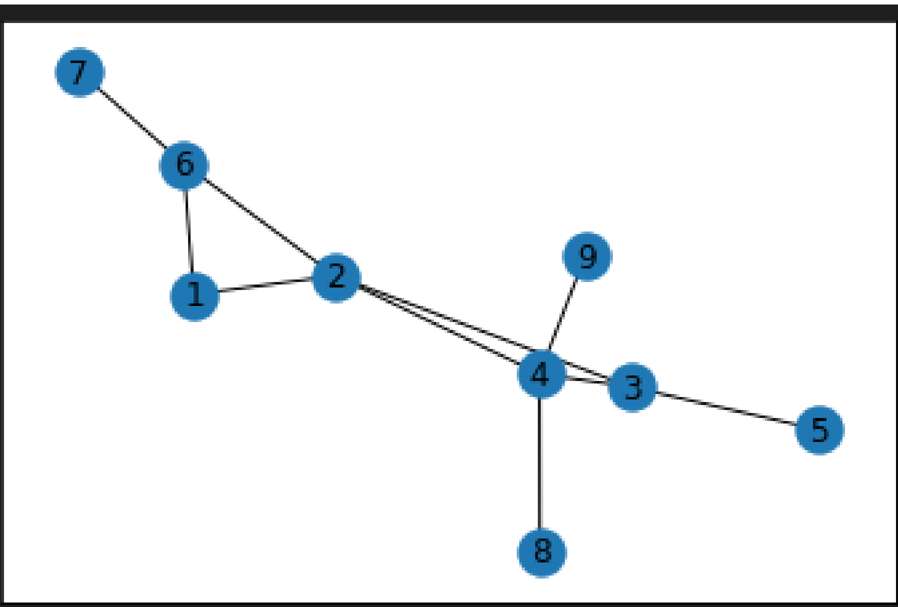
```
#1  all the nodes of a graph  [1, 2, 3, 4, 7, 9]
#2  all the edges of a graph [(1, 2), (1, 3), (1, 4), (1, 9), (1, 7), (2, 4), (2, 9)]
#3 node list [1, 2, 4, 7, 9]
#4  edgelist [(1, 4), (1, 9), (1, 7), (2, 4), (2, 9)]
#5  number of nodes  5
#6  number of edges 5
#7   degree of node(2) 2
#8 neighbor of a node(2) [4, 9]
```

```
import networkx as nx


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

G.add_edges_from(edges)
nx.draw(G, with_labels=True, font_weight='bold')

print("Total number of nodes: ", int(G.number_of_nodes()))
print("Total number of edges: ", int(G.number_of_edges()))
print("List of all nodes: ", list(G.nodes()))
print("List of all edges: ", list(G.edges(data = True)))
print("Degree for all nodes: ", dict(G.degree()))
#print("Total number of self-loops: ", list(G.number_of_selfloops()))
#print("List of all nodes with self-
loops: ",list(G.nodes_with_selfloops()))

print("List of all nodes we can go to in a single step from node 2: ",
```

```
list(G.neighbors(2)))
Total number of nodes:  9
Total number of edges:  10
List of all nodes:  [1, 2, 6, 3, 4, 5, 8, 9, 7]
List of all edges:  [(1, 2, {}), (1, 6, {}), (2, 3, {}), (2, 4, {}), (2, 6,
{}), (6, 7, {}), (3, 4, {}), (3, 5, {}), (4, 8, {}), (4, 9, {})]
Degree for all nodes:  {1: 2, 2: 4, 6: 3, 3: 3, 4: 4, 5: 1, 8: 1, 9: 1, 7:
1}
List of all nodes we can go to in a single step from node 2:  [1, 3, 4, 6]
```




```
import networkx as nx
G = nx.Graph()
edges = [(1, 2, 19), (1, 6, 15), (2, 3, 6), (2, 4, 10),
     (2, 6, 22), (3, 4, 51), (3, 5, 14), (4, 8, 20),
     (4, 9, 42), (6, 7, 30)]
G.add_weighted_edges_from(edges)
nx.draw_networkx(G, with_labels = True)
new_array = np.array(edges)
#print(new_array)
np.savetxt("edge_list.txt", new_array, delimiter =", ")
```




```
import pandas as pd


df = pd.read_csv('edge_list.txt', delim_whitespace = True,
         header = None, names =['n1', 'n2', 'weight'])

#G = nx.from_pandas_dataframe(df, 'n1', 'n2', edge_attr ='weight')

# The Graph diagram does not show the edge weights.
# However, we can get the weights by printing all the
# edges along with the weights by the command below
print(list(G.edges(data = True)))
```

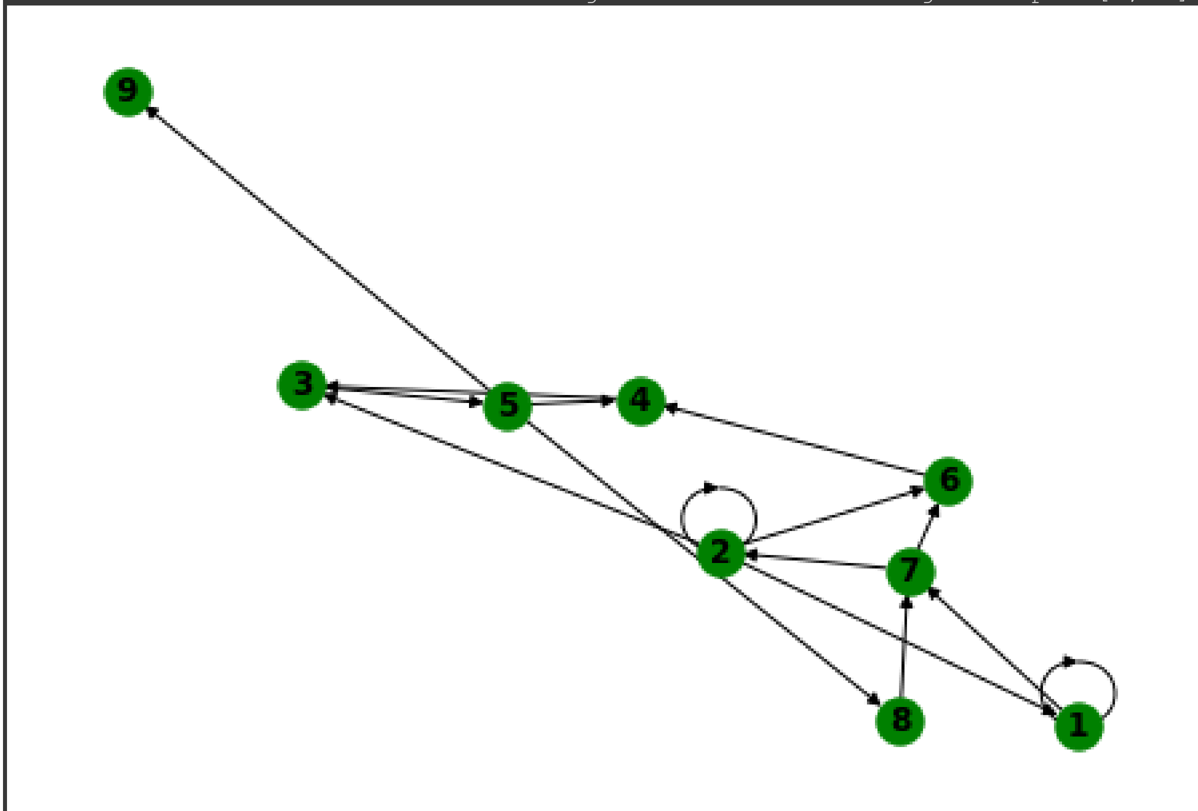
```
[(1, 1, {}), (1, 7, {}), (7, 2, {}), (7, 6, {}), (2, 1, {}), (2, 2,
{}), (2, 3, {}), (2, 6, {}), (3, 5, {}), (6, 4, {}), (5, 4, {}), (5, 8,
{}), (5, 9, {}), (4, 3, {}), (8, 7, {})]
```

Creating Directed Graph –

```
import networkx as nx
G = nx.DiGraph()
G.add_edges_from([(1, 1), (1, 7), (2, 1), (2, 2), (2, 3),
         (2, 6), (3, 5), (4, 3), (5, 4), (5, 8),
         (5, 9), (6, 4), (7, 2), (7, 6), (8, 7)])


nx.draw(G, with_labels=True, font_weight='bold',node_color ='green')
# getting different graph attributes
print("Total number of nodes: ", int(G.number_of_nodes()))
print("Total number of edges: ", int(G.number_of_edges()))
print("List of all nodes: ", list(G.nodes()))
print("List of all edges: ", list(G.edges()))
print("In-degree for all nodes: ", dict(G.in_degree()))
print("Out degree for all nodes: ", dict(G.out_degree))
```

```python
print("List of all nodes we can go to in a single step from node 2: ",
                            list(G.successors(2)))
print("List of all nodes from which we can go to node 2 in a single ste
p: ",
                              list(G.predecessors(2)))
```

```
Total number of nodes:  9
Total number of edges:  15
List of all nodes:  [1, 7, 2, 3, 6, 5, 4, 8, 9]
List of all edges:  [(1, 1), (1, 7), (7, 2), (7, 6), (2, 1), (2, 2), (2,
3), (2, 6), (3, 5), (6, 4), (5, 4), (5, 8), (5, 9), (4, 3), (8, 7)]
In-degree for all nodes:  {1: 2, 7: 2, 2: 2, 3: 2, 6: 2, 5: 1, 4: 2, 8: 1,
9: 1}
Out degree for all nodes:  {1: 2, 7: 2, 2: 4, 3: 1, 6: 1, 5: 3, 4: 1, 8: 1,
9: 0}
List of all nodes we can go to in a single step from node 2:  [1, 2, 3, 6]
List of all nodes from which we can go to node 2 in a single step:  [2, 7]
```



Experiment 2 : Social Network Analysis

In this prcatice we will use NetworkX. NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. You can see the full documentation of NetworkX HERE

```
# Import Library

import networkx as nx
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
!wget http://vlado.fmf.uni-lj.si/pub/networks/data/Ucinet/zachary.dat
```

```
--2022-03-30 07:21:54--  http://vlado.fmf.uni-lj.si/pub/networks/data/Ucinet/zachary.dat
Resolving vlado.fmf.uni-lj.si (vlado.fmf.uni-lj.si)... 193.2.67.80
Connecting to vlado.fmf.uni-lj.si (vlado.fmf.uni-lj.si)|193.2.67.80|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4849 (4.7K) [text/plain]
Saving to: 'zachary.dat'

zachary.dat         100%[===================>]   4.74K  --.-KB/s    in 0.1s

2022-03-30 07:21:54 (40.7 KB/s) - 'zachary.dat' saved [4849/4849]
```

```
from google.colab import drive

drive.mount('/content/gdrive')
df = pd.read_csv('/content/zachary.dat', sep=',')
df.head()
```

```
Mounted at /content/gdrive
```

	DL
0	N=34 NM=2
1	FORMAT = FULLMATRIX DIAGONAL PRESENT
2	LEVEL LABELS:
3	ZACHE
4	ZACHC

```
#df.tail
#df.head()
#df.describe()
#df.shape
#df.dtypes
```

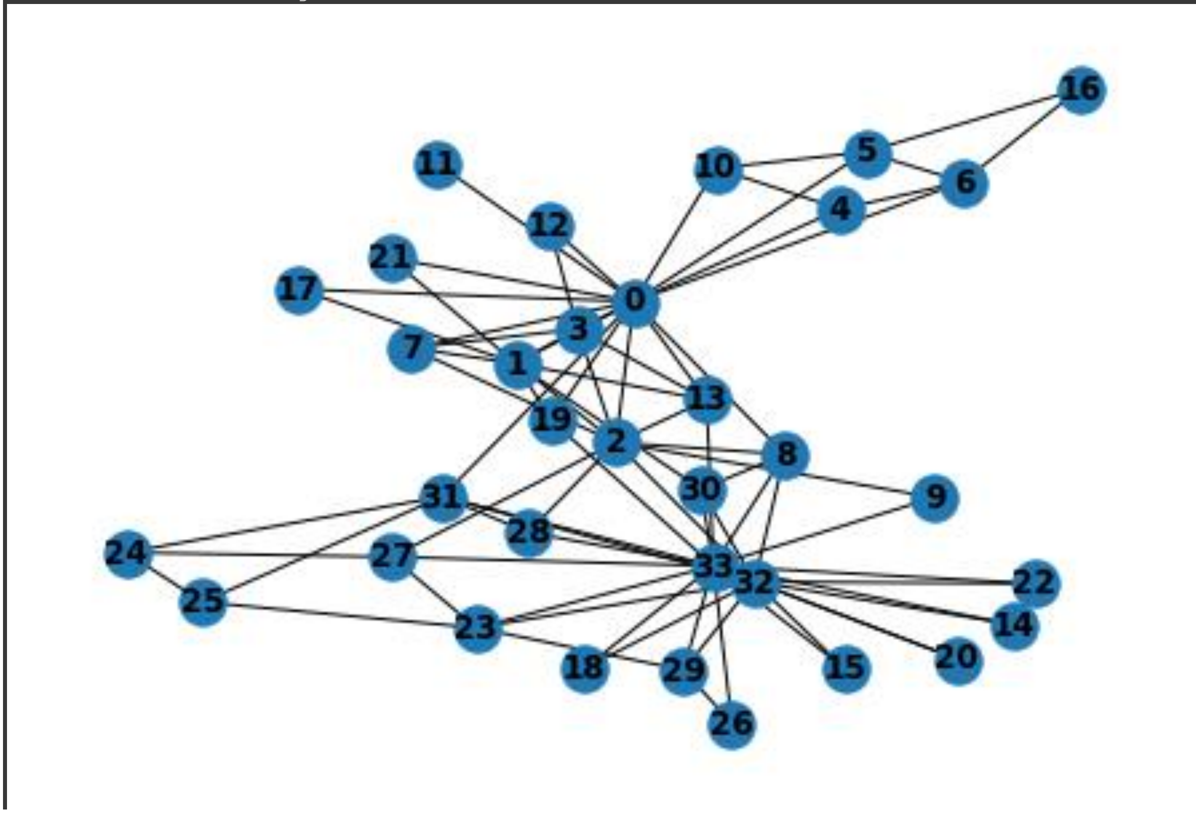
Import Dataset

will use Zachary's karate Club Dataset that avaiable in NetworkX

```
# Import Dataset
```

```
G = nx.karate_club_graph()
print('#nodes:', len(G.nodes()), 'and', '#edges:', len(G.edges()))
nx.draw(G, with_labels=True, font_weight='bold')
plt.show()
```

```
#nodes: 34 and #edges: 78
```



The data was collected from the members of a university karate club by Wayne Zachary in 1977. Each node represents a member of the club. Each edge represents a tie between two members of the club. The network is undirected and unweighted. There are 34 members of a university karate club and 78 interactions between members.

Show Nodes and Edges

```
# Show the nodes
nx.nodes(G)
#list(G.nodes)
```

```
NodeView((0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33))
```

```
# Show the edges
nx.edges(G)
#list(G.edges)
```

```
EdgeView([(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 10), (0, 11), (0, 12), (0, 13), (0, 17), (0, 19), (0, 21), (0, 31), (1, 2), (1, 3), (1, 7), (1, 13), (1, 17), (1, 19), (1, 21), (1, 30), (2, 3), (2, 7), (2, 8), (2, 9), (2, 13), (2, 27), (2, 28), (2, 32), (3, 7), (3, 12), (3, 13), (4, 6), (4, 10), (5, 6), (5, 10), (5, 16), (6, 16), (8, 30), (8, 32), (8, 33), (9, 33), (13, 33), (14, 32), (14, 33), (15, 32), (15, 33), (18, 32), (18, 33), (19, 33), (20, 32),
```

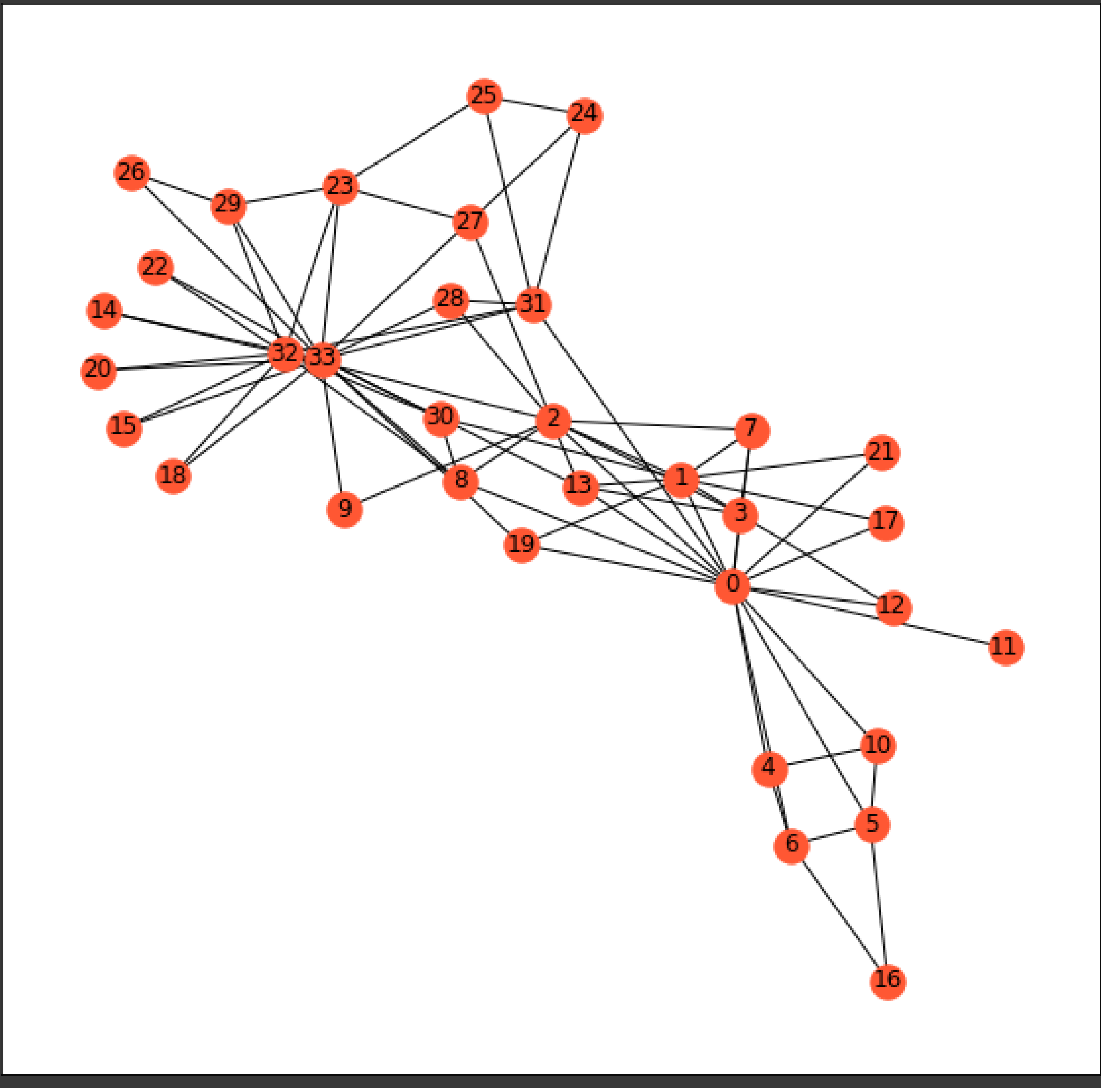
```
(20, 33), (22, 32), (22, 33), (23, 25), (23, 27), (23, 29), (23, 32),
(23, 33), (24, 25), (24, 27), (24, 31), (25, 31), (26, 29), (26, 33),
(27, 33), (28, 31), (28, 33), (29, 32), (29, 33), (30, 32), (30, 33),
(31, 32), (31, 33), (32, 33)])
```

Drawing/Visualization

Drawing/Visualization documentation: https://networkx.github.io/documentation/stable/reference/drawing.html

```
# Drawing
import matplotlib.pyplot as plt
layout = nx.fruchterman_reingold_layout(G)
plt.figure(figsize=(10,10))
plt.axis("on")
nx.draw_networkx(G, layout, with_labels=True, node_color = '#FF5733' )
list(G.adj[30])
```

```
[1, 8, 32, 33]
```

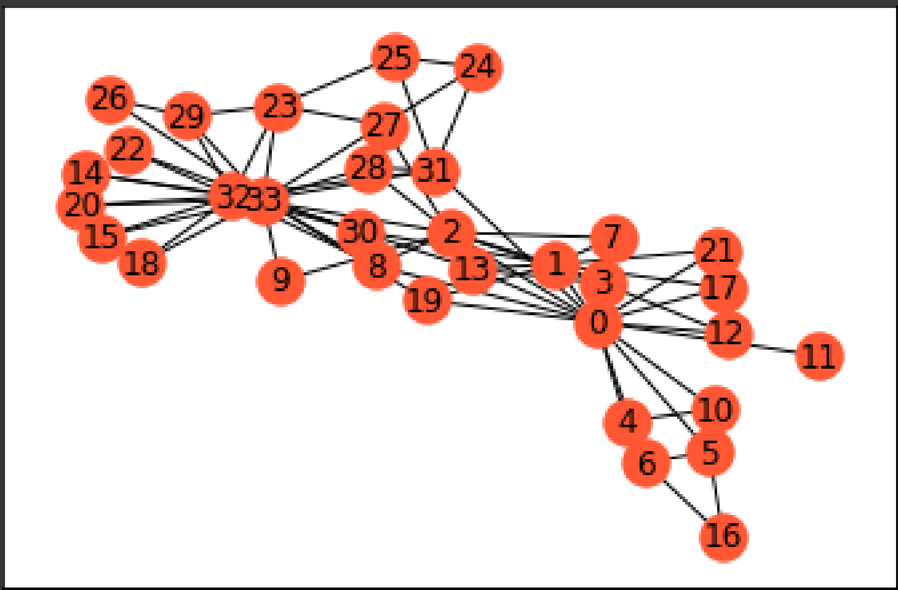


Graph Density and Matrix

Closer to 0 , the interactions between karate members are more distant. Closer to 1 , the interactions are denser.

```
# Graph Density
nx.density(G)
nx.draw_networkx(G, layout, with_labels=True, node_color = '#FF5733' )
A=nx.to_numpy_matrix(G) # Incidence matrix A.shape
print(A)
H=nx.from_numpy_matrix(A)
print(H)
```

```
[[0. 1. 1. ... 1. 0. 0.]
 [1. 0. 1. ... 0. 0. 0.]
 [1. 1. 0. ... 0. 1. 0.]
 ...
 [1. 0. 0. ... 0. 1. 1.]
 [0. 0. 1. ... 1. 0. 1.]
 [0. 0. 0. ... 1. 1. 0.]]
Graph with 34 nodes and 78 edges
```



Average Shortest Path Lenght

Average distance among karate members.

```
# Average Shortest Path Lenght
nx.average_shortest_path_length(G)
#Graph=nx.from_numpy_matrix('numpy_adj_matrix.npy')
```

Network Diameter

Maximum distance between two farthest members.

```
nx.diameter(G)
5
```

Degree

```
# Show node degree
nx.degree(G)
```

```
DegreeView({0: 16, 1: 9, 2: 10, 3: 6, 4: 3, 5: 4, 6: 4, 7: 4, 8: 5, 9: 2, 10: 3, 11: 1, 12: 2, 13: 5, 14: 2, 15: 2, 16: 2, 17: 2, 18: 2, 19: 3, 20: 2, 21: 2, 22: 2, 23: 5, 24: 3, 25: 3, 26: 2, 27: 4, 28: 3, 29: 4, 30: 4, 31: 6, 32: 12, 33: 17})
```

```
# Sorted from the highest Degree
sorted(nx.degree(G), key=lambda x: x[1], reverse=True)
```

```
[(33, 17),
 (0, 16),
 (32, 12),
 (2, 10),
 (1, 9),
 (3, 6),
 (31, 6),
 (8, 5),
 (13, 5),
 (23, 5),
 (5, 4),
 (6, 4),
 (7, 4),
 (27, 4),
 (29, 4),
 (30, 4),
 (4, 3),
 (10, 3),
 (19, 3),
 (24, 3),
 (25, 3),
 (28, 3),
 (9, 2),
 (12, 2),
 (14, 2),
 (15, 2),
 (16, 2),
 (17, 2),
 (18, 2),
 (20, 2),
 (21, 2),
 (22, 2),
 (26, 2),
 (11, 1)]
```

Betweenness Centrality

```
# Calculate betweeness centrality and sort from the highest value
sorted(nx.betweenness_centrality(G, normalized=True).items(), key=lambda x:x[1], reverse=True)[0:10]
```

```
[(0, 0.43763528138528146),
 (33, 0.30407497594997596),
 (32, 0.145247113997114),
 (2, 0.14365680615680618),
 (31, 0.13827561327561325),
```

```
 (8, 0.05592682780182781),
 (1, 0.05393668831168830 4),
 (13, 0.04586339586339586),
 (19, 0.03247504810004811),
 (5, 0.02998737373737374)]
```

Closeness Centrality

```
# Calculate closeness centrality and sort from the highest value
sorted(nx.closeness_centrality(G).items(), key=lambda x:x[1], reverse=True)[0:10]
```

```
[(0, 0.5689655172413793),
 (2, 0.559322033898305),
 (33, 0.55),
 (31, 0.5409836065573771),
 (8, 0.515625),
 (13, 0.515625),
 (32, 0.515625),
 (19, 0.5),
 (1, 0.4852941176470588),
 (3, 0.4647887323943662)]
```

Experiment 3 : Network Analysis of Line Segment Data

```
import networkx as nx
G= nx.Graph()
G.clear()
G.add_nodes_from([701,702,703,704,705,706,707,708,709,710,711,712,713,714,720,727,730,732,733,734,737,738,740,744,799])
G.add_weighted_edges_from([(701,702,960),(702,705,400),(702,713,360),(702,703,1320),(703,727,240),(703,730,600),(704,714,80),(704,720,800),(705,742,320),(705,712,240),(706,725,280),(707,724,760),(707,722,120),(708,733,320),(708,732,320),(709,731,600),(709,708,320),(710,735,200),(710,736,1280),(711,741,400),(711,740,200),(713,704,520),(714,718,520),(720,707,920),(720,706,600),(727,744,280),(730,709,200),(733,734,560),(734,737,640),(734,710,520),(737,738,400),(738,711,400),(744,728,200),(744,729,280),(775,709,0),(799,701,1850)])
nx.draw(G, with_labels=True,font_weight='bold')
```