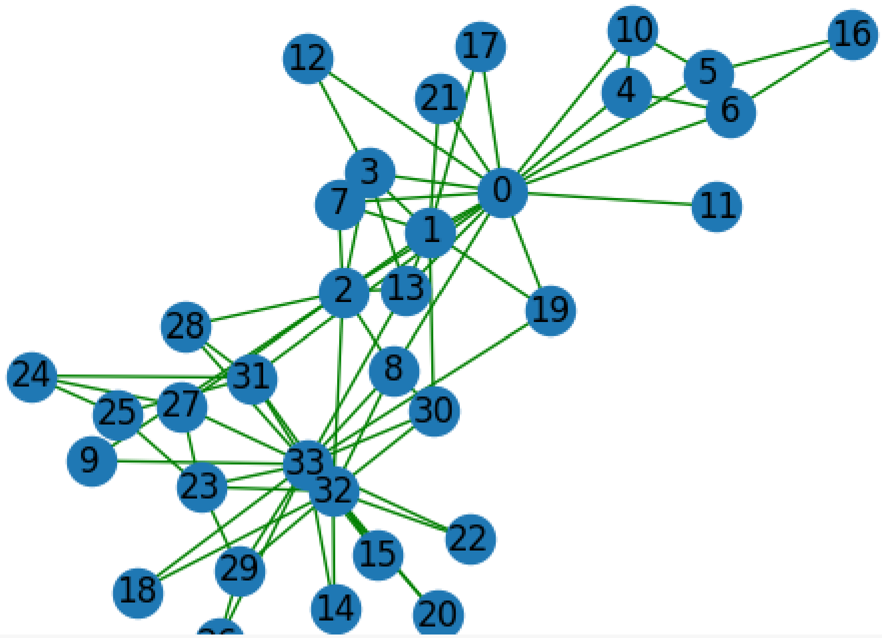
```
%matplotlib inline
```

```
# Community Algorithms. Clustering and Component methods
```

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

# https://networkx.org/documentation/stable/reference/algorithms/community.html

# Load karate graph and find communities using Girvan-Newman
# https://networkx.org/documentation/stable/reference/generated/networkx.generators.social.karate_club_graph.html
# https://networkx.org/documentation/stable/auto_examples/algorithms/plot_girvan_newman.html
# https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.centrality.girvan_n

# https://networkx.guide/algorithms/community-detection/
# Girvan-Newman alg. detects communities by progressively removing edges from the original network.

G = nx.karate_club_graph()

print("G Karate Graph")
nx.draw_networkx(
    G,
    nx.spring_layout(G),
    node_size=300,
    edge_color="g",
    with_labels=True,
)
plt.axis("off")
plt.show()

plt.show()
```

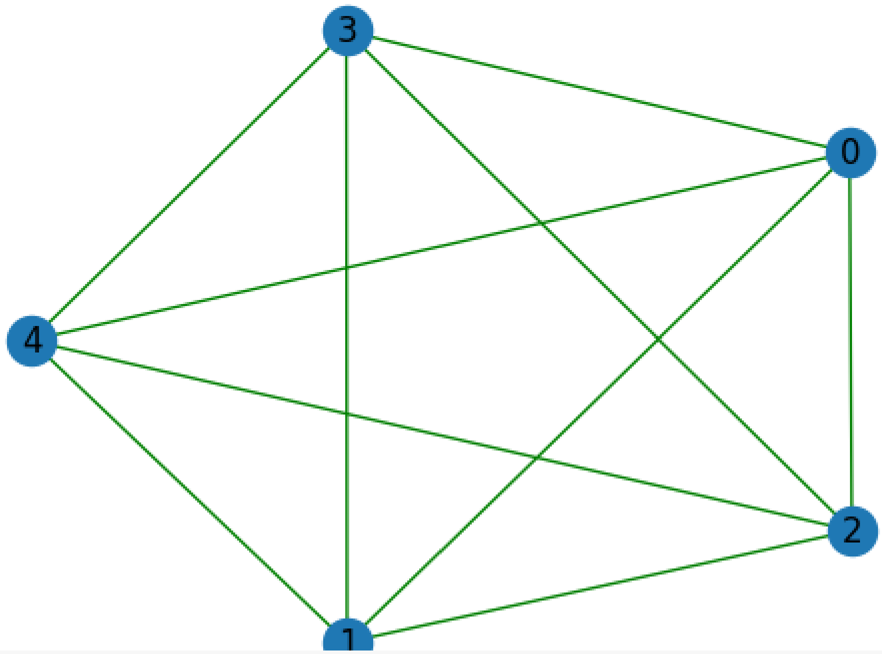
G Karate Graph




```
from networkx.generators.small import complete_graph

# Example 2
G2 = nx.complete_graph(5)

print("Graph complet dimensiune 5, G2 = nx.complete_graph(5)")
nx.draw_networkx( G2,nx.spring_layout(G2),
                 node_size=300, edge_color="g",with_labels=True,)
plt.axis("off")
plt.show()

# Community functions similar with Neo4J

# Clustering
# https://networkx.org/documentation/stable/reference/algorithms/clustering.html
# triangles(G, nodes=None) Finds the number of triangles that include a node as one vertex.

# https://networkx.org/documentation/stable/reference/generated/networkx.generators.classic.complete_graph.html
```

```
# triangles(G, nodes=None) Compute the number of triangles.

# https://networkx.org/documentation/stable/reference/algorithms/community.html
# https://networkx.guide/algorithms/community-detection/

# Example 1
# G = nx.karate_club_graph()
# communities = list(nx.community.girvan_newman(G))
print("Triangles, G graph karate ")
print(nx.triangles(G,0))
print(nx.triangles(G))
print("nodes (0,1) ",list(nx.triangles(G, (0, 1)).values()))

print(" ")
# Example 2
print("Triangles, G2 graph 5 complete")
print(" node 0 ",nx.triangles(G2, 0))
# 6
print("all nodes ",nx.triangles(G2))
# {0: 6, 1: 6, 2: 6, 3: 6, 4: 6}

print("nodes (0,1) ",list(nx.triangles(G2, (0, 1)).values()))
# [6, 6]

print(" ")
```

```
Graph complet dimensiune 5, G2 = nx.complete_graph(5)
```




```
# Clustering
# clustering(G, nodes=None, weight=None) Compute the clustering coefficient for nodes.
# https://networkx.org/documentation/stable/reference/algorithms/clustering.html
# https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.cluster.clustering.html

# Example 1
# G = nx.karate_club_graph()
# Example 2
# G2 = nx.complete_graph(5)

# Example 1
print("Clustering coefficient, G graph karate ")
print("nodes 0 ",nx.clustering(G, 0))
print("all nodes ", nx.clustering(G))

print(" ")
# Example 2
print("Clustering coefficient, G2, 5 complete ")
```

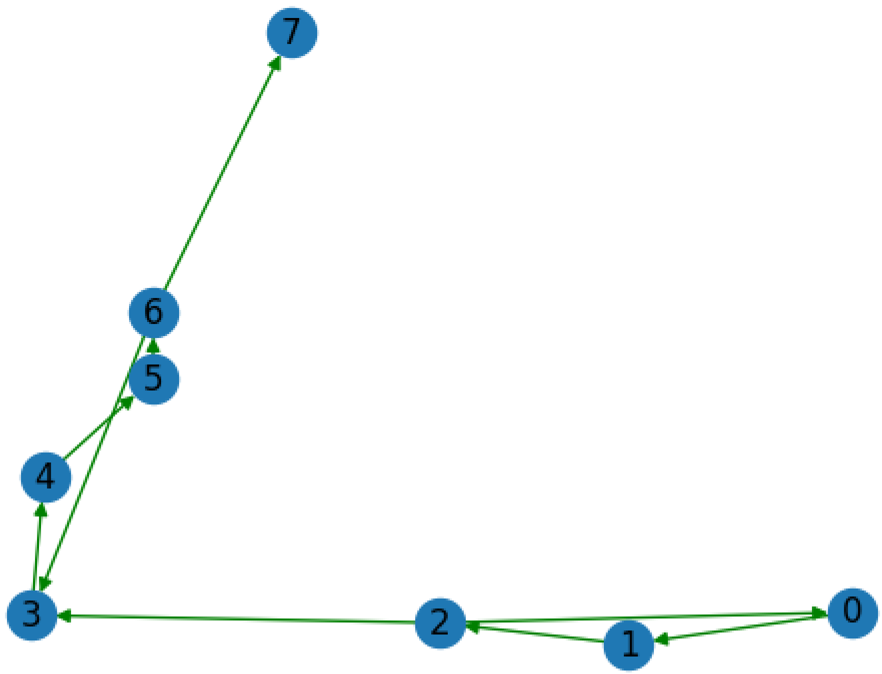
```
print("nodes 0 ",nx.clustering(G2, 0))
# 1.0
print("all nodes ", nx.clustering(G2))
# {0: 1.0, 1: 1.0, 2: 1.0, 3: 1.0, 4: 1.0}
```

```
Clustering coefficient, G graph karate
nodes 0  0.15
all nodes  {0: 0.15, 1: 0.3333333333333333, 2: 0.24444444444444444, 3: 0.6666666666666666, 4: 0.6666666666666666, 5: 0.

Clustering coefficient, G2, 5 complete
nodes 0  1.0
all nodes  {0: 1.0, 1: 1.0, 2: 1.0, 3: 1.0, 4: 1.0}
```

```
# Component
# https://networkx.org/documentation/stable/reference/algorithms/component.html
# https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.components.node_connected_com
# Example 1  G = nx.karate_club_graph()
# Example 2  G2 = nx.complete_graph(5)

# node_connected_component(G, n) (G undirected; n:node label; A node) Returns the set  of nodes in the component of graph co
# number_strongly_connected_components( G)
# Example 3
# https://networkx.org/documentation/stable/reference/classes/digraph.html
# DiGraph—Directed graphs with self loops (Grafuri orientatecu bucle "loop")
G3 = nx.DiGraph([(0, 1), (1, 2), (2, 0), (2, 3), (4,5), (3, 4), (5, 6), (6, 3), (6, 7)]
)
print("Graph DiGraph")
nx.draw_networkx( G3,nx.spring_layout(G3),
                 node_size=300, edge_color="g",with_labels=True,)
plt.axis("off")
plt.show()

#  A directed graph: grafuri orientate cu loops
print("Strongly connected components, G3 digraph  ")
print("number of strongly connected components:", nx.number_strongly_connected_components(G3))
# 3
```

```
print("Nu se aplica Exemples: G: Graph Karate si G2, 5 complet; lipsa bucle, loops: connected component")
```

```
Graph DiGraph
```




```
Strongly connected components, G3 digraph
number of strongly connected components: 3
Nu se aplica Exemples: G: Graph Karate si G2, 5 complet; lipsa bucle, loops: connected component
```

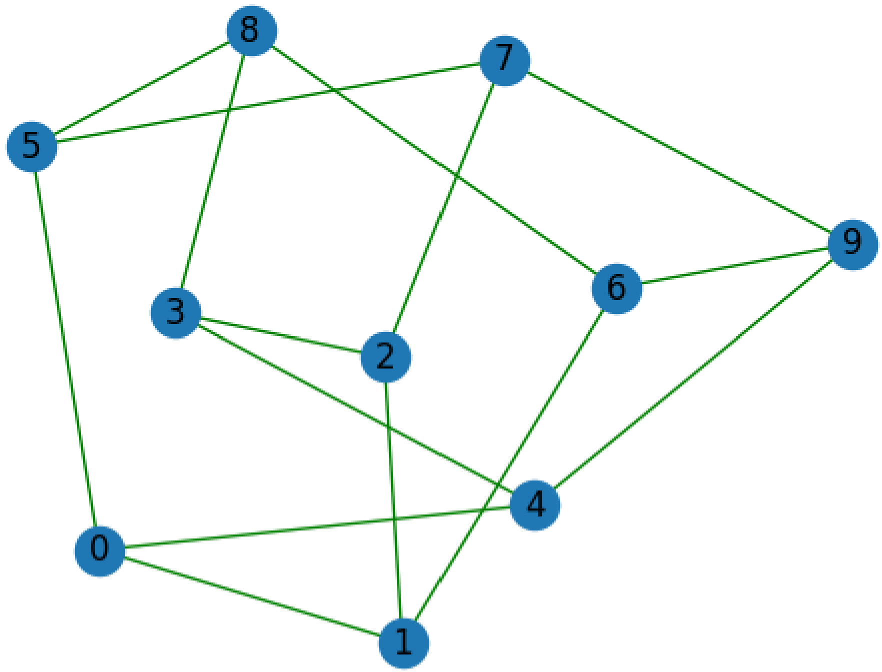
```
# Component
# https://networkx.org/documentation/stable/reference/algorithms/component.html
# https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.components.node_connected_com

# Example 4
G4 = nx.Graph([(0, 1), (1, 2), (5, 6), (3, 4)])
print("Graph G4 ([(0, 1), (1, 2), (5, 6), (3, 4)])")
```

```
nx.draw_networkx( G4,nx.spring_layout(G4), arrows=None,
                 node_size=300, edge_color="g",with_labels=True,)
plt.axis("off")
plt.show()
print("Connected components, G4 digraph [(0, 1), (1, 2), (5, 6), (3, 4)])")
print("number of connected components:",nx.number_connected_components(G4))
# 3
print("number of connected components that contains node 0: ")
nx.node_connected_component(G4, 0) # nodes of component that contains node 0
# {0, 1, 2}

print("Nu se aplica Exemples: G: Graph Karate si G2, 5 complet; lipsa bucle, loops: connected component")
```

```
Graph G4 ([(0, 1), (1, 2), (5, 6), (3, 4)])
```

```
# Community
# Community detection = compute a partitioning of communities that maximizes modularity.
# Modularity = metrica (valori [-0.5, 1])
# https://arxiv.org/pdf/1410.1237.pdf  https://www.youtube.com/watch?v=akfiGPBtCuM
# Louvain Find the best partition of a graph using the Louvain Community Detection Algorithm.
# https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.louvain.louvain_comm

# Louvain: unsupervised algorithm (not required as input:number of communities/sizes before execution)
# Louvain: 2 phases: Modularity Optimization and Community Aggregation executed until: no more changes are achived.
#
# import networkx as nx
G5 = nx.petersen_graph()
print("Petersen Graph G6=nx.petersen_graph()")
nx.draw_networkx( G5,nx.spring_layout(G5), arrows=None,
                 node_size=300, edge_color="g",with_labels=True,)
plt.axis("off")
plt.show()

print("Community Louvain Communities(G5, seed=123)")
print("Example 5 Petersen Graph:", nx.community.louvain_communities(G5, seed=123))
# [{0, 4, 5, 7, 9}, {1, 2, 3, 6, 8}]

print("Example 4: G4=nx.Graph([(0, 1), (1, 2), (5, 6), (3, 4)]) Louvain Communities(G4, seed=123)")
print(nx.community.louvain_communities(G4, seed=123))

print("Example 3: G3 = nx.DiGraph([(0, 1), (1, 2), (2, 0), (2, 3), (4,5), (3, 4), (5, 6), (6, 3), (6, 7)]) Louvain Communiti
print(nx.community.louvain_communities(G3, seed=123))

print("Example 2: G2 = nx.complete_graph(5)Community Louvain Communities(G2, seed=123)")
print(nx.community.louvain_communities(G2, seed=123))

print("Louvain does not work for graphs without loops Examples: G=nx.karate_club_graph()")
```

```
Petersen Graph G6=nx.petersen_graph()
```




```
Community Louvain Communities(G5, seed=123)
Example 5 Petersen Graph: [{0, 4, 5, 7, 9}, {1, 2, 3, 6, 8}]
Example 4: G4=nx.Graph([(0, 1), (1, 2), (5, 6), (3, 4)]) Louvain Communities(G4, seed=123)
[{0, 1, 2}, {5, 6}, {3, 4}]
Example 3: G3 = nx.DiGraph([(0, 1), (1, 2), (2, 0), (2, 3), (4,5), (3, 4), (5, 6), (6, 3), (6, 7)]) Louvain Communities
[{0, 1, 2}, {3, 4}, {5, 6, 7}]
Example 2: G2 = nx.complete_graph(5)Community Louvain Communities(G2, seed=123)
[{0, 1, 2, 3, 4}]
Louvain does not work for graphs without loops Examples: G=nx.karate_club_graph()
```

```
# Community
# Label propagation  -  Grafuri Neorientate (Not implemented for directed graphs)
# label_propagation_communities(G) (just for G undirected)
# semi-supervised ML algorithm: assigns labels to previously unlabeled data points
# Generates community sets determined by label propagation
# Memgraph implemented the asynchronous label propagation.
# https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.label_propagation.a
# asyn_lpa_communities(G, weight=None, seed=None)
```

```
# https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.label_propagation.la
# label_propagation_communities(G)
# Generates community sets determined by label propagation.
# Finds communities: using semi-synchronous label propagation method [https://arxiv.org/pdf/1103.4550.pdf]
# Combines: advantages: synchronous & asynchronous models.

# http://scikit.ml/_modules/skmultilearn/cluster/networkx.html
# https://python.hotexamples.com/examples/networkx.asyn_lpa/-/asyn_lpa_communities/python-asyn_lpa_communities-function-examp
# https://programtalk.com/python-examples/networkx.asyn_lpa.asyn_lpa_communities/

import networkx as nx
import networkx.algorithms.community as nxcom
#asyn_lpa_G = list(nxcom.asyn_lpa_communities(G,weight="weight"))

# G5 = nx.petersen_graph()
print("G5 Petersen asyn LPA LIST: Label propagation")
asyn_lpa_G5 = list(nxcom.asyn_lpa_communities(G5))
print(asyn_lpa_G5)
print("G5 Petersen asyn LPA Length LIST: Label propagation")
print(len(asyn_lpa_G5))
print("G5 Petersen asyn LPA Length of all list communities: Label propagation")
for i in asyn_lpa_G5 :
    print(len(i))

print("     ")
# G5 = nx.petersen_graph()
print("G5 Petersen LPA label_propagation_communities(G) LIST: Label propagation")
lpa_G5 = list(nxcom.label_propagation_communities(G5))
print(lpa_G5)
print("G5 Petersen LPA label_propagation_communities(G) Length LIST: Label propagation")
print(len(lpa_G5))
print("G5 Petersen LPA Length of all list communities: Label propagation")
for i in lpa_G5 :
    print(len(i))

#
print("     ")
```

```
print("Example 4: G4=nx.Graph([(0, 1), (1, 2), (5, 6), (3, 4)])Label propagation")

print("G4 asyn LPA LIST: Label propagation")
asyn_lpa_G4 = list(nxcom.asyn_lpa_communities(G4))
print(asyn_lpa_G4)
print("G4 asyn LPA Length LIST: Label propagation")
print(len(asyn_lpa_G4))
print("G4 asyn LPA Length of all list communities: Label propagation")
for i in asyn_lpa_G4 :
    print(len(i))

print("     ")
print("G4 LPA label_propagation_communities(G) LIST: Label propagation")
lpa_G4 = list(nxcom.label_propagation_communities(G4))
print(lpa_G4)
print("G6 Petersen label_propagation_communities(G) Length LIST: Label propagation")
print(len(lpa_G4))
print("G6 Petersen graph asyncr. Length of all list communities: Label propagation")
for i in lpa_G4 :
    print(len(i))

#
print("     ")

print("G3  asyn LPA LIST: Label propagation")
asyn_lpa_G3 = list(nxcom.asyn_lpa_communities(G3,weight="weight"))
print(asyn_lpa_G3)
print("G3 asyn LPA Length LIST: Label propagation")
print(len(asyn_lpa_G3))
print("G3 asyn LPA Length of all list communities: Label propagation")
for i in asyn_lpa_G3 :
    print(len(i))

print("     ")

print("label_propagation_communities(G) not implemented for oriented graphs as G3")

print("     ")
```

```
#
print("Example 2: G2 = nx.complete_graph(5) Label propagation")

print("G2  asyn LPA LIST: Label propagation")
asyn_lpa_G2 = list(nxcom.asyn_lpa_communities(G2))
print(asyn_lpa_G2)
print("G2 asyn LPA Length LIST: Label propagation")
print(len(asyn_lpa_G2))
print("G2 asyn LPA Length of all list communities: Label propagation")
for i in asyn_lpa_G2 :
    print(len(i))

print("     ")
print("G2 LPA label_propagation_communities(G) LIST: Label propagation")
lpa_G2 = list(nxcom.label_propagation_communities(G2))
print(lpa_G2)
print("G2 LPA label_propagation_communities(G) Length LIST: Label propagation")
print(len(lpa_G2))
print("G2 LPA Length of all list communities: Label propagation")
for i in lpa_G2 :
    print(len(i))

#

print("     ")

print("G Karate graph asyn LPA LIST: Label propagation")
asyn_lpa_G = list(nxcom.asyn_lpa_communities(G,weight="weight"))
print(asyn_lpa_G)
print("G asyn LPA Length LIST: Label propagation")
print(len(asyn_lpa_G))
print("G asyn LPA Length of all list communities: Label propagation")
for i in asyn_lpa_G :
    print(len(i))

print("label_propagation_communities(G) not implemented for oriented graphs as G")
```

```
G5 Petersen asyn LPA LIST: Label propagation
[{0, 1, 2, 5, 7}, {9, 3, 4}, {8, 6}]
G5 Petersen asyn LPA Length LIST: Label propagation
3
G5 Petersen asyn LPA Length of all list communities: Label propagation
5
3
2

G5 Petersen LPA label_propagation_communities(G) LIST: Label propagation
[{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}]
G5 Petersen LPA label_propagation_communities(G) Length LIST: Label propagation
1
G5 Petersen LPA Length of all list communities: Label propagation
10

Example 4: G4=nx.Graph([(0, 1), (1, 2), (5, 6), (3, 4)])Label propagation
G4 asyn LPA LIST: Label propagation
[{0, 1, 2}, {5, 6}, {3, 4}]
G4 asyn LPA Length LIST: Label propagation
3
G4 asyn LPA Length of all list communities: Label propagation
3
2
2

G4 LPA label_propagation_communities(G) LIST: Label propagation
[{0, 1, 2}, {5, 6}, {3, 4}]
G6 Petersen label_propagation_communities(G) Length LIST: Label propagation
3
G6 Petersen graph asyncr. Length of all list communities: Label propagation
3
2
2

G3  asyn LPA LIST: Label propagation
[{0, 1, 2}, {3, 4, 5, 6, 7}]
G3 asyn LPA Length LIST: Label propagation
```

```
G3 asyn LPA Length of all list communities: Label propagation
3
5

label_propagation_communities(G) not implemented for oriented graphs as G3

Example 2: G2 = nx.complete_graph(5) Label propagation
G2  asyn LPA LIST: Label propagation
[{0, 1, 2, 3, 4}]
G2 asyn LPA Length LIST: Label propagation
1
G2 asyn LPA Length of all list communities: Label propagation
5

G2 LPA label_propagation_communities(G) LIST: Label propagation
[{0, 1, 2, 3, 4}]
G2 LPA label_propagation_communities(G) Length LIST: Label propagation
```

```
# Studiu Optional
# Community Detection using Girvan-Newman
https://networkx.guide/algorithms/community-detection/girvan-newman/#:~:text=The%20Girvan%2DNewman%20algorithm%20for,between%20nodes%20passing%20through%20them.

https://networkx.org/documentation/stable/auto_examples/algorithms/plot_girvan_newman.html

https://networkx.org/documentation/stable/_downloads/c6c0d374543fe8d55d85e1e1256ab098/plot_girvan_newman.ipynb

This example shows the detection of communities in the Zachary Karate Club dataset using the Girvan-Newman method.
We plot the change in modularity as important edges are removed.
```

Studiu Optional

Community Detection using Girvan-Newman

https://networkx.guide/algorithms/community-detection/girvan-newman/#:~:text=The%20Girvan%2DNewman%20algorithm%20for,between%20nodes%20passing%20through%20them.

https://networkx.org/documentation/stable/auto_examples/algorithms/plot_girvan_newman.html

```
Graph is coloured and plotted based on community
detection when number of iterations are 1 and 4
respectively.

Pseudocode
REPEAT
    LET n BE number of edges in the graph
    FOR i=0 to n-1
        LET B[i] BE betweenness centrality of edge i
        IF B[i] > max_B THEN
            max_B = B[i]
            max_B_edge = i
        ENDIF
    ENDFOR
    remove edge i from graph
UNTIL number of edges in graph is 0
```

https://networkx.org/documentation/stable/_downloads/c6c0d374543fe8d55d85e1e1256ab098/plot_girvan_newman.ipynb

This example shows the detection of communities in the Zachary Karate Club dataset using the Girvan-Newman method. We plot the change in modularity as important edges are removed. Graph is coloured and plotted based on community detection when number of iterations are 1 and 4 respectively.

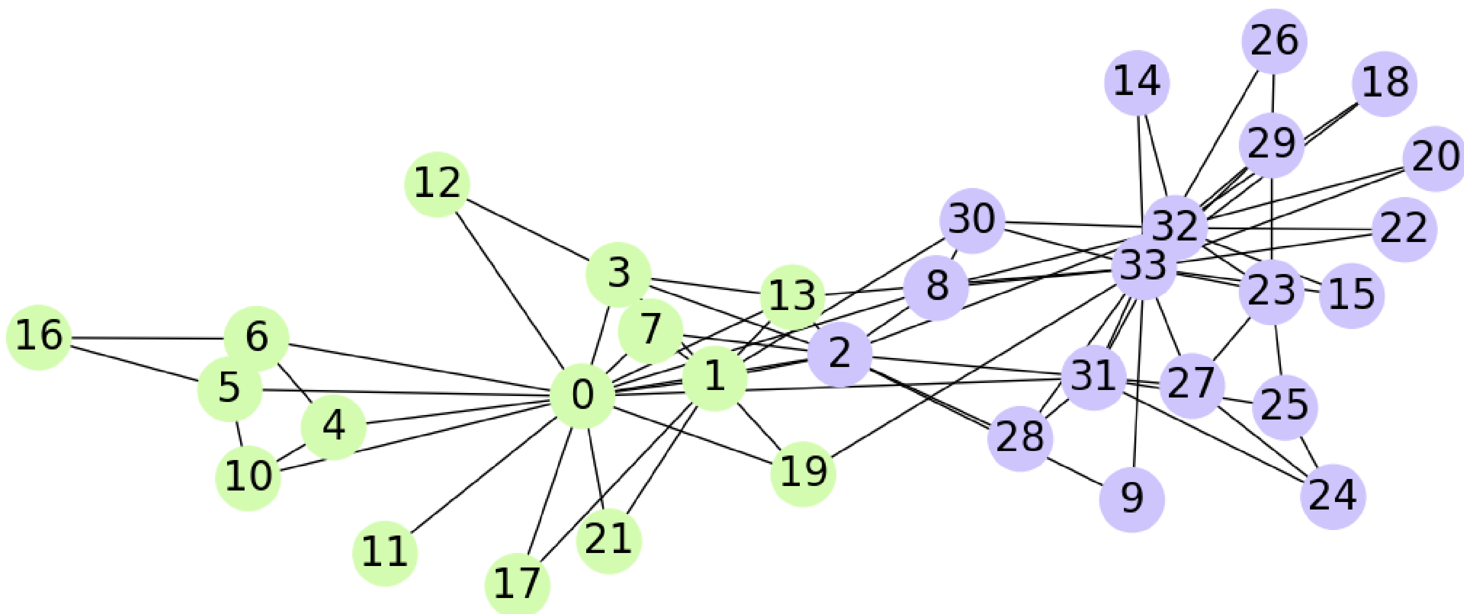
Pseudocode REPEAT LET n BE number of edges in the graph FOR i=0 to n-1 LET B[i] BE betweenness centrality of edge i IF B[i] > max_B THEN max_B = B[i] max_B_edge = i ENDIF ENDFOR remove edge i from graph UNTIL number of edges in graph is 0

```
# https://networkx.org/documentation/stable/reference/algorithms/community.html

# Load karate graph and find communities using Girvan-Newman
# https://networkx.org/documentation/stable/reference/generated/networkx.generators.social.karate_club_graph.html
# https://networkx.org/documentation/stable/auto_examples/algorithms/plot_girvan_newman.html
# https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.centrality.girvan_n

# https://networkx.guide/algorithms/community-detection/
# Girvan-Newman alg. detects communities by progressively removing edges from the original network.

# G = nx.karate_club_graph()
communities = list(nx.community.girvan_newman(G))

# Modularity -> measures the strength of division of a network into modules
modularity_df = pd.DataFrame(
    [
        [k + 1, nx.community.modularity(G, communities[k])]
        for k in range(len(communities))
    ],
    columns=["k", "modularity"],
```

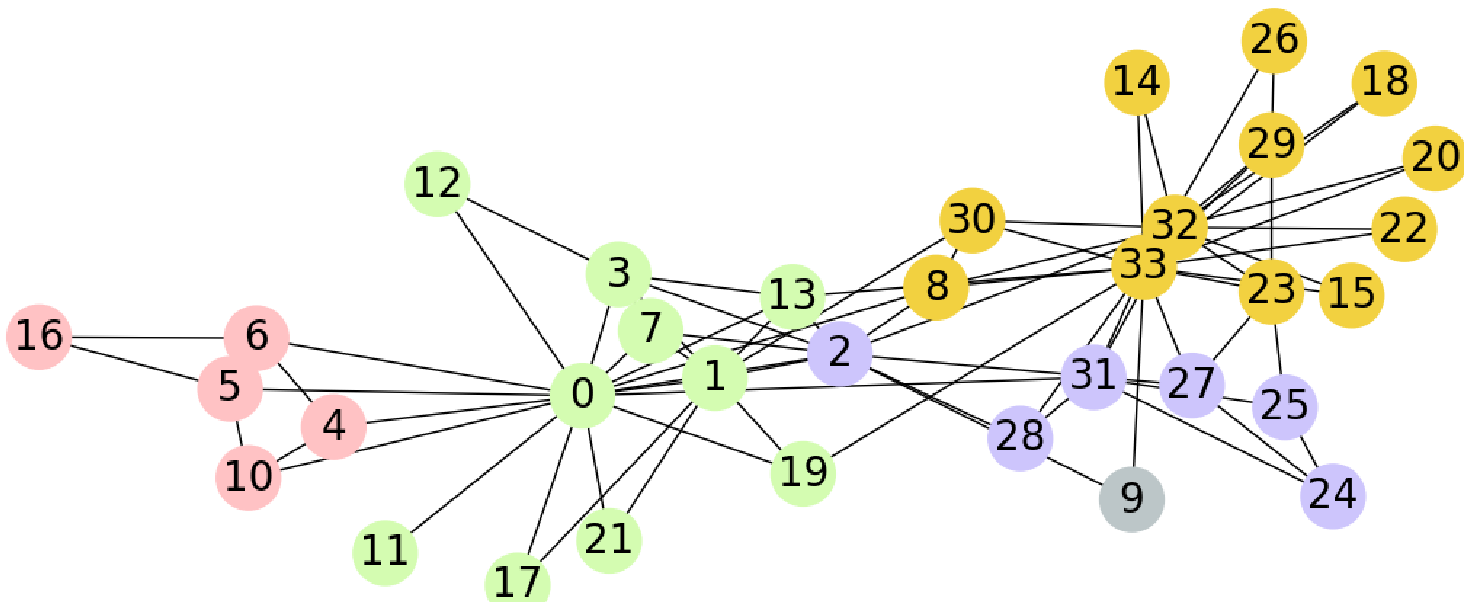
```
)


# function to create node colour list
def create_community_node_colors(graph, communities):
    number_of_colors = len(communities[0])
    colors = ["#D4FCB1", "#CDC5FC", "#FFC2C4", "#F2D140", "#BCC6C8"][:number_of_colors]
    node_colors = []
    for node in graph:
        current_community_index = 0
        for community in communities:
            if node in community:
                node_colors.append(colors[current_community_index])
                break
            current_community_index += 1
    return node_colors


# function to plot graph with node colouring based on communities
def visualize_communities(graph, communities, i):
    node_colors = create_community_node_colors(graph, communities)
    modularity = round(nx.community.modularity(graph, communities), 6)
    title = f"Community Visualization of {len(communities)} communities with modularity of {modularity}"
    pos = nx.spring_layout(graph, k=0.3, iterations=50, seed=2)
    plt.subplot(3, 1, i)
    plt.title(title)
    nx.draw(
        graph,
        pos=pos,
        node_size=1000,
        node_color=node_colors,
        with_labels=True,
        font_size=20,
        font_color="black",
    )


fig, ax = plt.subplots(3, figsize=(15, 20))
```

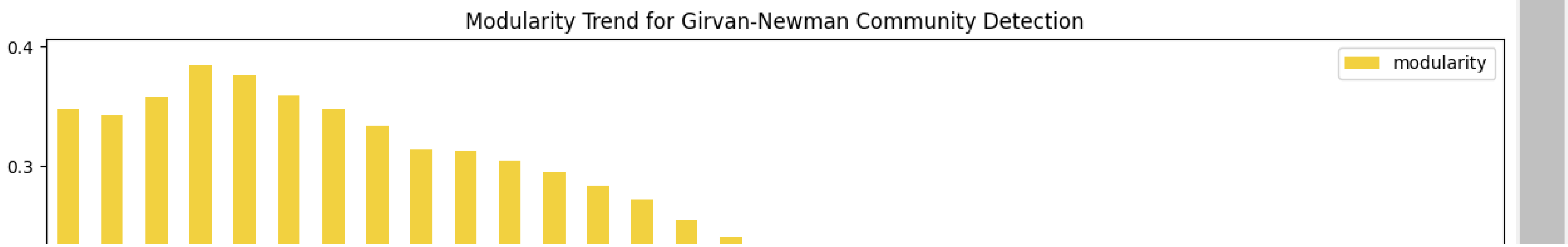
```python
# Plot graph with colouring based on communities
visualize_communities(G, communities[0], 1)
visualize_communities(G, communities[3], 2)

# Plot change in modularity as the important edges are removed
modularity_df.plot.bar(
    x="k",
    ax=ax[2],
    color="#F2D140",
    title="Modularity Trend for Girvan-Newman Community Detection",
)
plt.show()
```

Community Visualization of 2 communities with modularity of 0.34766



Community Visualization of 5 communities with modularity of 0.384972





Modularity Trend for Girvan-Newman Community Detection
0.4
0.3
modularity