HW5 - Graph Partitioning

Adeniran Adeniyi

Sunday, March 28, 2021 by 11:59pm

Q1

Draw the original Karate club graph (before the split) and color the nodes according to the factions they belong to (John A or Mr. Hi).

Answer

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 25 16:00:19 2021
@author: adeni
"""

import networkx as nx
import matplotlib.pyplot as plt
import re
import numpy as np
"""
TASK
1. Final result of the graph after breakdown
2. Iteration, color code the nodes connecting path and break.
"""

"""
https://gawron.sdsu.edu/python_for_ss/course_core/book_draft/Social_Networks/Networkx.html
def GirvanNewman():
  while (no edge left or desired number of communities unreached):
      calculate Betweeness of all edges
      remove the edge with the highest edge betweenness
      calculate the number of strongly connected component (communities)
"""
def colorCode(p,flag,listT):
    color_map = []
    lent = len(p)
    if flag == "" and len(listT) ==0:
        color_map = ['yellow'] * 34
        color_map[0] = "red"
```

```python
        color_map[33] = "green"
    elif(flag=="final" and len(listT) == 2):

        color_map = ['blue'] * 34
        for n in listT[0]:
            color_map[n] = "red"
        for t in listT[1]:
            color_map[t] ="green"
    return color_map
def find_best_edge(G0):
    """
    Networkx implementation of edge_betweenness
    returns a dictionary. Make this into a list,
    sort it and return the edge with highest betweenness.
    """
    eb = nx.edge_betweenness_centrality(G0)
    eb_li = list(eb.items())
    eb_li.sort(key=lambda x: x[1], reverse=True)
    return eb_li[0][0]

def getComponent(G):
    if len(G.nodes()) == 1:
        return [G.nodes()]
    components = (G.subgraph(c) for c in nx.connected_components(G))
    components = list(components)
    count = 0
    while len(components) == 1:
        count +=1
        G.remove_edge(*find_best_edge(G))

        components =(G.subgraph(c) for c in nx.connected_components(G))
        components = list(components)
    return components
def plot_theGraph(G, color, pathname,spaceing,edge_cl_map,weight_map):
    match = re.search(r'((Q[0-9]\/)([0-9a-zA-Z]*\.png))',pathname)
    name = match.group(3)
    plt.figure(figsize=(15,8.8))
    plt.title(name,fontsize=20)
    if spaceing == "":
        nx.draw_kamada_kawai(G,with_labels=True, node_color = color)
    else:

        pos = nx.spring_layout(G, k=0.3*1/np.sqrt(len(G.nodes())) +0.1, iterations=20)
        nx.draw(G,with_labels=True ,node_color = color,pos=pos, edge_color=edge_cl_map,width = list(weight_map))

    plt.savefig(pathname, format="PNG")
    plt.show()
    plt.close()
    return 0
def set_color_edges(G,tuplesEdgeToRemove,reset):
    """
    This builds the edge attributes color and weight

    """
    totalEdges = G.number_of_edges()
    color_edge_map = [’black’] * totalEdges
    weight_map = [1.5] * totalEdges
    if(reset =="n"):
        total = -1
        for n in G.edges:
            total += 1
            if tuplesEdgeToRemove == n:
                color_edge_map[total] = ’blue’
                weight_map[total] = 3.2
    return color_edge_map,weight_map
def girvan_newman(G):
    components = (G.subgraph(c) for c in nx.connected_components(G))
    components = list(components)
    count =0
    while len(components) == 1:
        count +=1
        path = "Q2/" +str(count) +"a.png"
        #find the best Edge and return as a list
        bestEdge = find_best_edge(G)
        edge_color_mapped,weightMap = set_color_edges(G, bestEdge,"n")

        #print(edge_color_mapped)
        plot_theGraph(G,color_map,path,"spacing",edge_color_mapped,
   weightMap)
        #Remove the best edge
        G.remove_edge(*bestEdge)
        #ReSet everything back to black and reset the weight too
        edge_color_mapped,weightMap = set_color_edges(G, bestEdge,"")
        #Build string path after edge as been removed
        path = "Q2/" + str(count) +"b.png"

        plot_theGraph(G,color_map,path,"spacing",edge_color_mapped,
   weightMap)
        #if(count > 18):
        #    break
    return 0
try:
```

```python

    karate = nx.karate_club_graph()
    t = karate
    """
    ===================================================
    Question 1a
    show group leaders in color coding
    ===================================================
    Got the data from networkx
    parsed the data to assign colorcoding for the two main leaders
    then plotted the graph
    """
    karate = nx.karate_club_graph()
    color_map = colorCode(karate, "","")
    plot_theGraph(karate,color_map, "Q1/karataHighlight.png","","","")
    """
    =======================================================================
    Question 1b , Question 2 , Question 3 and Extra Credit Q1
    show the categories based on the distribution as a color coded
    =======================================================================
    passed the retrieved data to the girvan_newman algorithm
    color coded the result of the splitted group
    then plotted the graph
    """
    #returns the broken components as two NodeView list)
    final = getComponent(karate)
    #print(final)
    #Set the colors based on the list received
    color_map = colorCode(karate,"final",final)
    plot_theGraph(karate,color_map,"Q1/finalGroup.png","","","")
    girvan_newman(t)
except Exception as e:
    print(e)
```

Listing 1: graphing.py

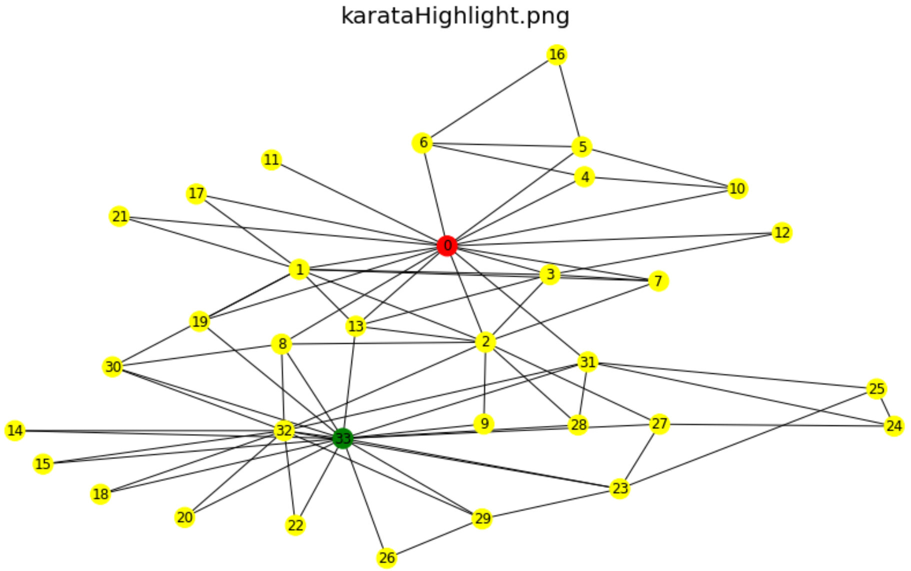

Figure 1: Dataset with two main group highlighted Red(John A), Green (My Hi), and Yellow for all others for visibility

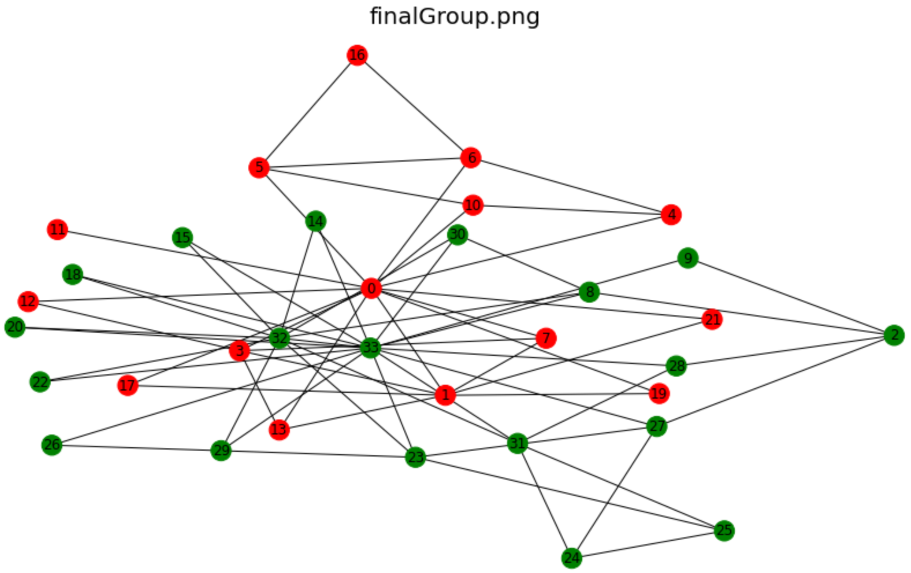

Figure 2: Appropriately distritbuted node colors, edges still as it is

Discussion

I used networkx library, where the Zachary Karate club dataset is on.

```
    karate = nx.karate_club_graph()
```

Listing 2: Getting data from networkx snapcshto of graphing.py

For this graph, I highlighed the two major parts of the Zachary karate club Main Leader, John A which represents the Red color and node number 0, while Mr Hi represent the color green and node number 33. They reset are will be given a color yellow for more visibility.
Major parts that ensured the result are reflect in the code snapshot of graphing.py:

- The driver for the question

```
    """
    ===================================================
    Question 1a
    show group leaders in color coding
    ===================================================
    Got the data from networkx
    parsed the data to assign colorcoding for the two main leaders
    then plotted the graph
    """
    karate = nx.karate_club_graph()
    color_map = colorCode(karate, "","")
    plot_theGraph(karate,color_map, "Q1/karataHighlight.png","","",
   "")
```

Listing 3: graphing.py

- colorCode function builds the list of the color for each nodes

```
def colorCode(p,flag,listT):
    color_map = []
    lent = len(p)
    if flag == "" and len(listT) ==0:
        color_map = ['yellow'] * 34
        color_map[0] = "red"
        color_map[33] = "green"
    elif(flag=="final" and len(listT) == 2):

        color_map = ['blue'] * 34
        for n in listT[0]:
            color_map[n] = "red"
        for t in listT[1]:
            color_map[t] ="green"
    return color_map
```

Listing 4: Building list for the color node in the graph (snapshot in graphing.py)

- A graphing function that handles all my graphing needs. For this part, I used the first conditional statement in line 69 -70

```
64 def plot_theGraph(G, color, pathname,spaceing,edge_cl_map, weight_map):
65     match = re.search(r'((Q[0-9]\/)([0-9a-zA-z]*\.png))',pathname)
66     name = match.group(3)
67     plt.figure(figsize=(15,8.8))
68     plt.title(name,fontsize=20)
69     if spaceing == "":
70         nx.draw_kamada_kawai(G,with_labels=True, node_color = color)
71     else:
72 
73         pos = nx.spring_layout(G, k=0.3*1/np.sqrt(len(G.nodes()))+0.1, iterations=20)
74         nx.draw(G,with_labels=True ,node_color = color,pos=pos, edge_color=edge_cl_map,width = list(weight_map))
75 
76     plt.savefig(pathname, format="PNG")
77     plt.show()
78     plt.close()
79     return 0
```

Listing 5: Making the plot for all the graphs (snapshot in graphing.py)

Q2

Run multiple iterations of the Girvan-Newman graph partioning algorithm (see Week-07 Social Networks, slides 90-99) on the Karate Club graph until the graph splits into two connected components. Keep the node colors the same as they were set in Q1. How many iterations did it take?
Your report should include images of all of the iterations. It will be easier to see the splits if you use a force-directed layout (such as Kamada-Kawai) rather than a circular layout.

Answer

```python
def girvan_newman(G):
    components = (G.subgraph(c) for c in nx.connected_components(G))
    components = list(components)
    count =0
    while len(components) == 1:
        count +=1
        path = "Q2/" +str(count) +"a.png"
        #find the best Edge and return as a list
        bestEdge = find_best_edge(G)
        edge_color_mapped,weightMap = set_color_edges(G, bestEdge,"n")

        #print(edge_color_mapped)
        plot_theGraph(G,color_map,path,"spacing",edge_color_mapped, weightMap)
        #Remove the best edge
        G.remove_edge(*bestEdge)
        #ReSet everything back to black and reset the weight too
        edge_color_mapped,weightMap = set_color_edges(G, bestEdge,"")
        #Build string path after edge as been removed
        path = "Q2/" + str(count) +"b.png"

        plot_theGraph(G,color_map,path,"spacing",edge_color_mapped, weightMap)
        #if(count > 18):
        #    break
    return 0
```

Listing 6: Girvan-newman algorithm (snapshot of graphing.py)

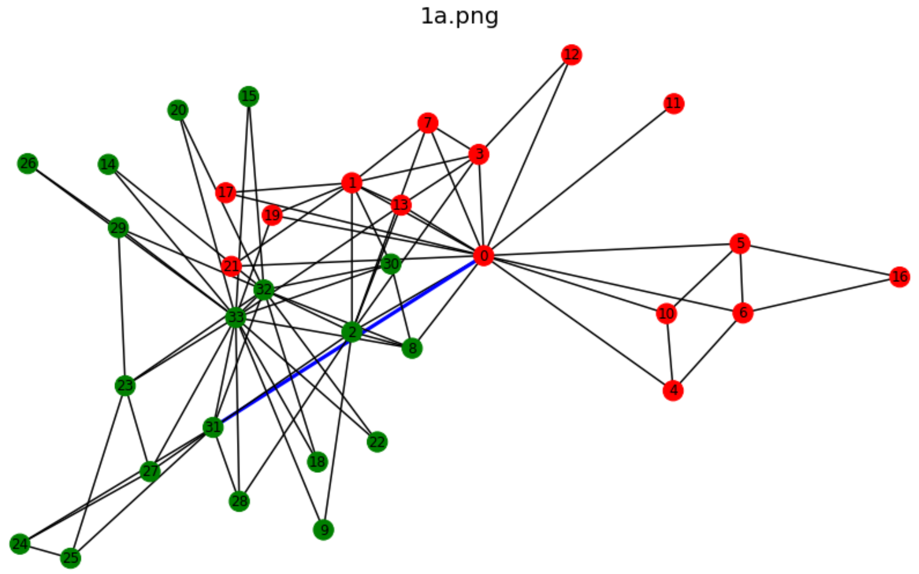

Figure 3: Iteration 1 Highlighted nodeedge to remove in blue

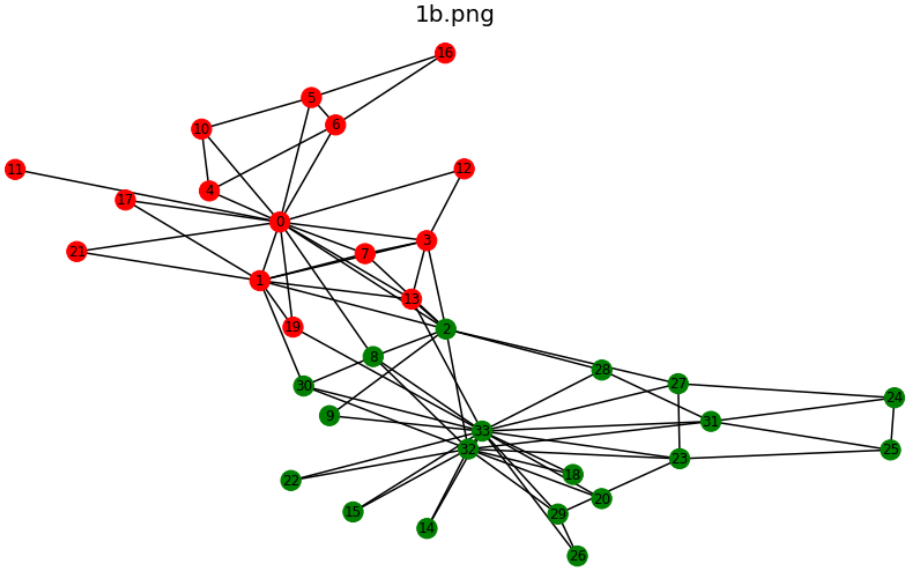

Figure 4: Iteration 1 confirm nodeedge has been removed

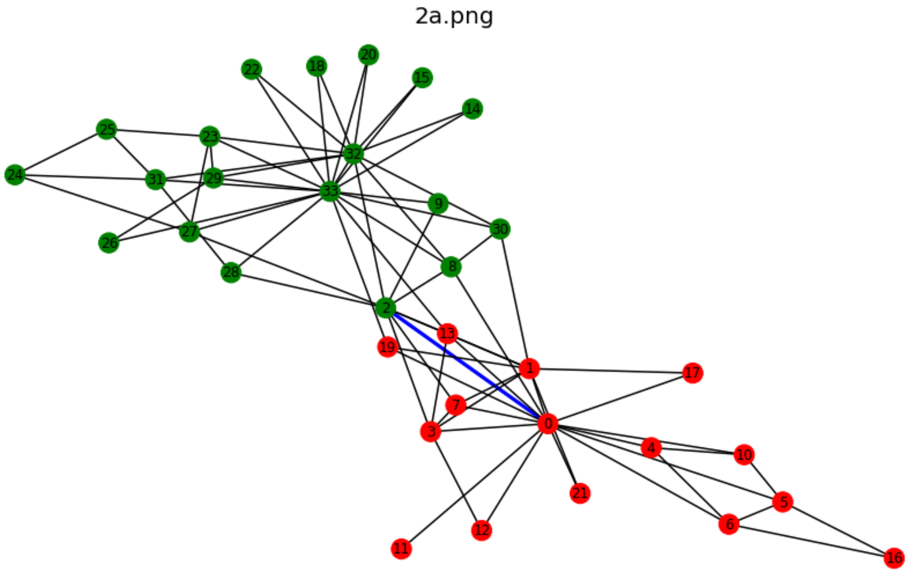

Figure 5: Iteration 2 Highlighted nodeedge to remove in blue

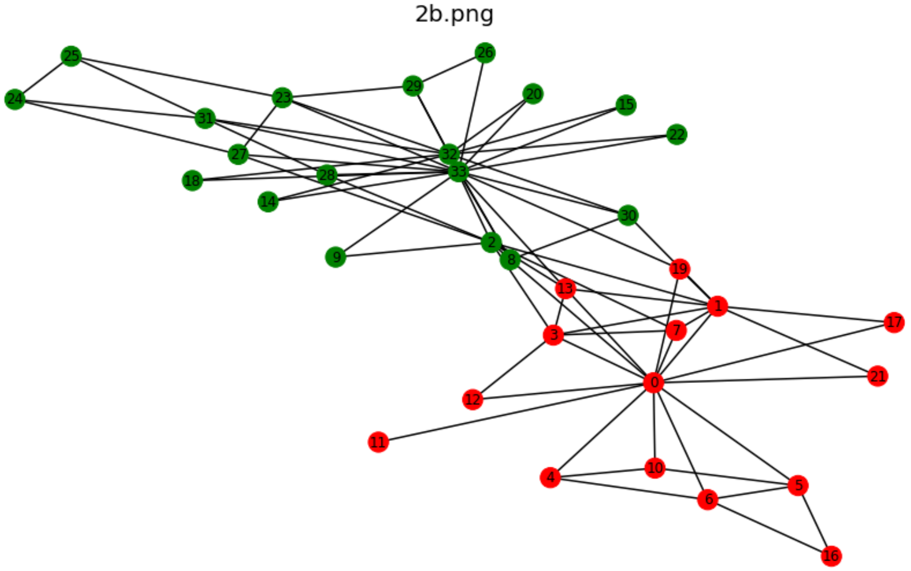

Figure 6: Iteration 2 confirm nodeedge has been removed

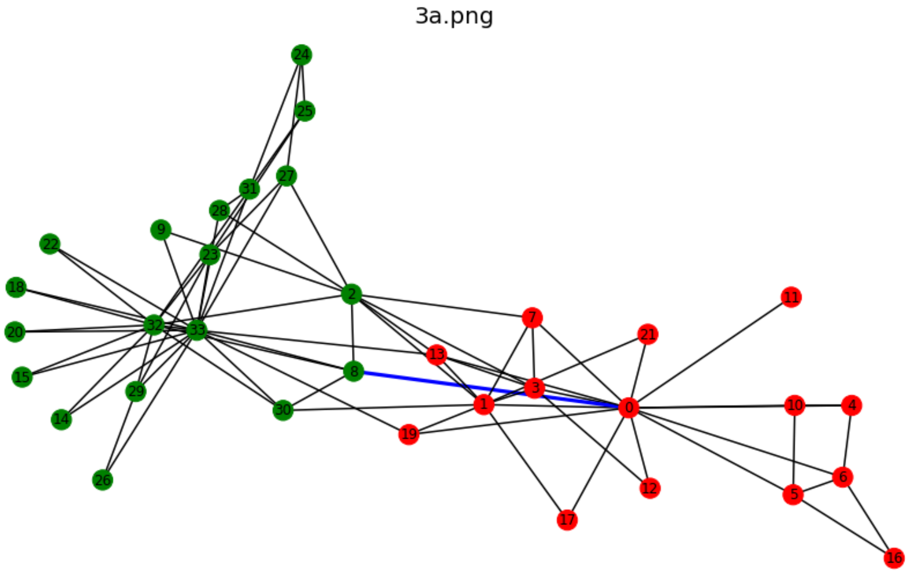

Figure 7: Iteration 3 Highlighted nodeedge to remove in blue

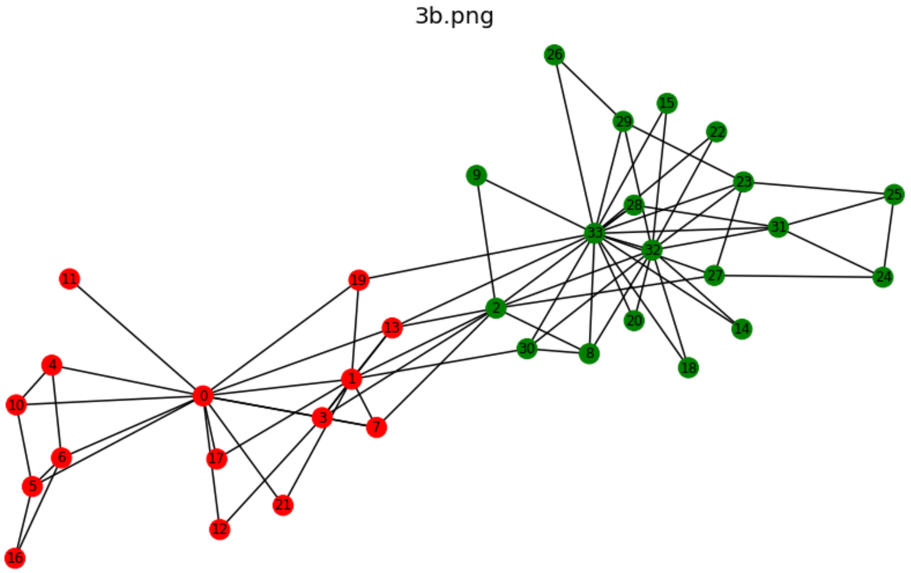

Figure 8: Iteration 3 confirm nodeedge has been removed

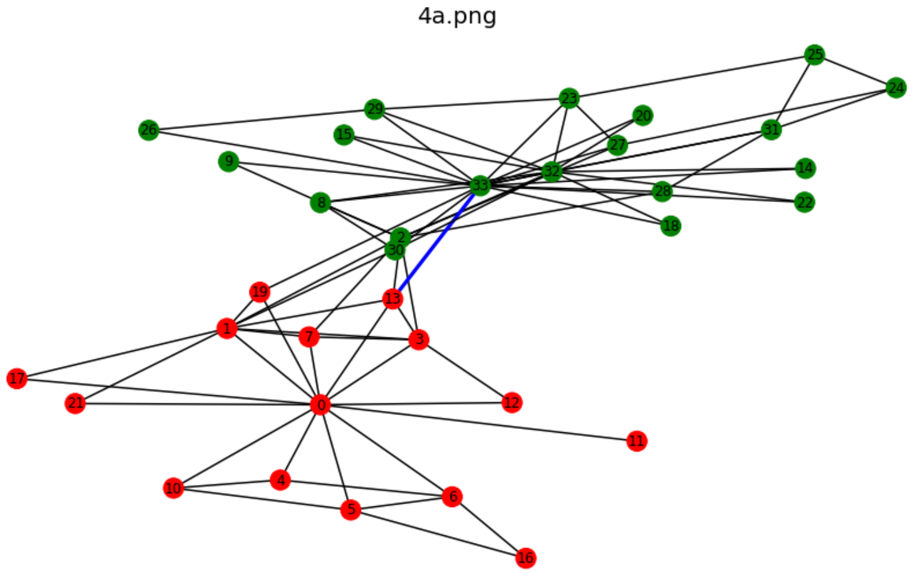

Figure 9: Iteration 4 Highlighted nodeedge to remove in blue

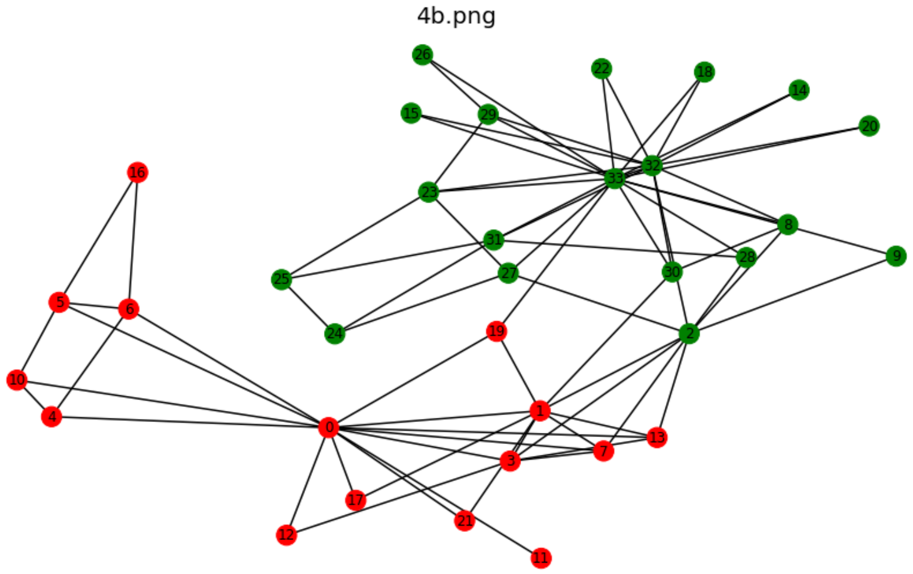

Figure 10: Iteration 4 confirm nodeedge has been removed

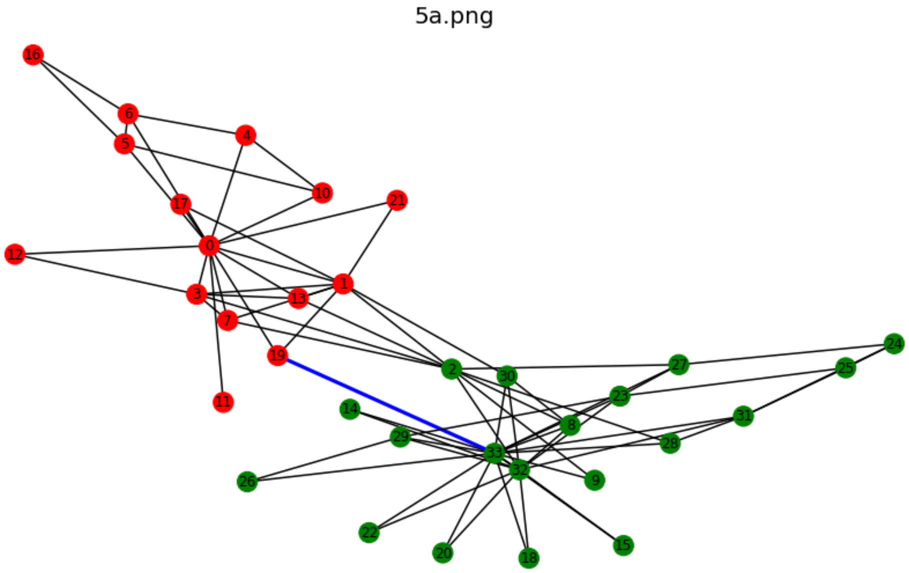

Figure 11: Iteration 5 Highlighted nodeedge to remove in blue

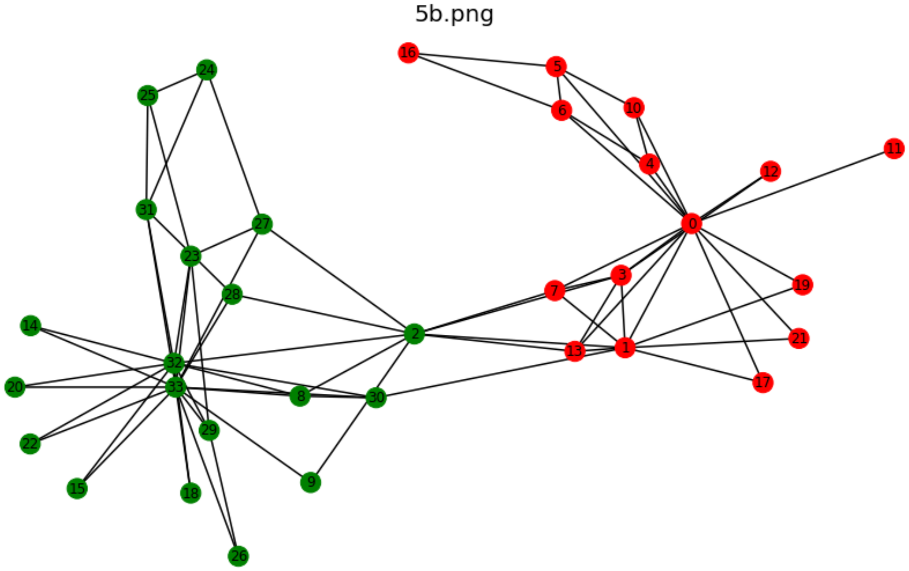

Figure 12: Iteration 5 confirm nodeedge has been removed

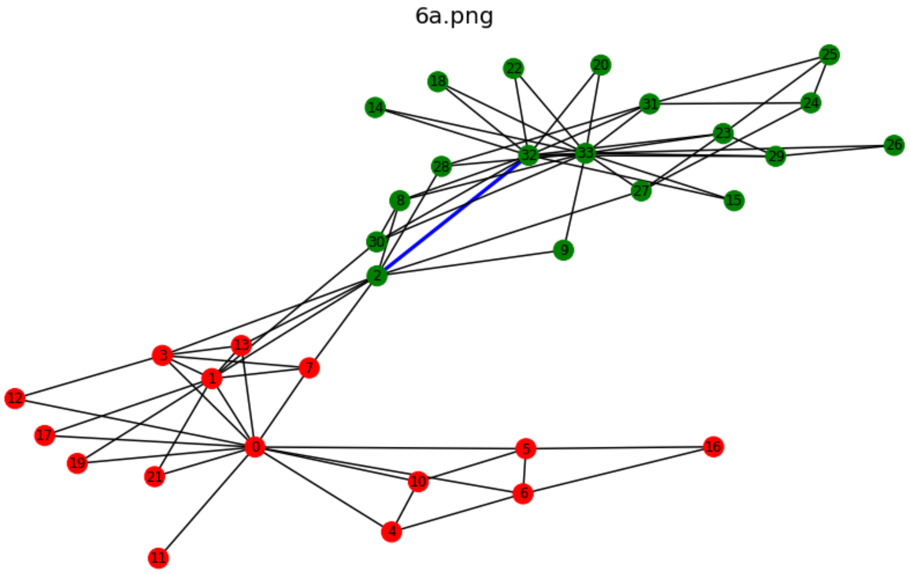

Figure 13: Iteration 6 Highlighted nodeedge to remove in blue

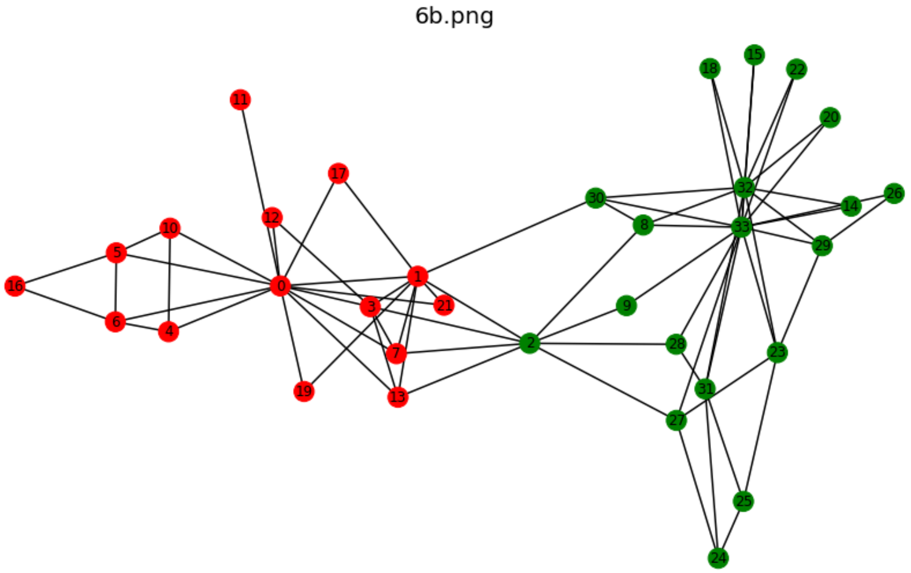

Figure 14: Iteration 6 confirm nodeedge has been removed

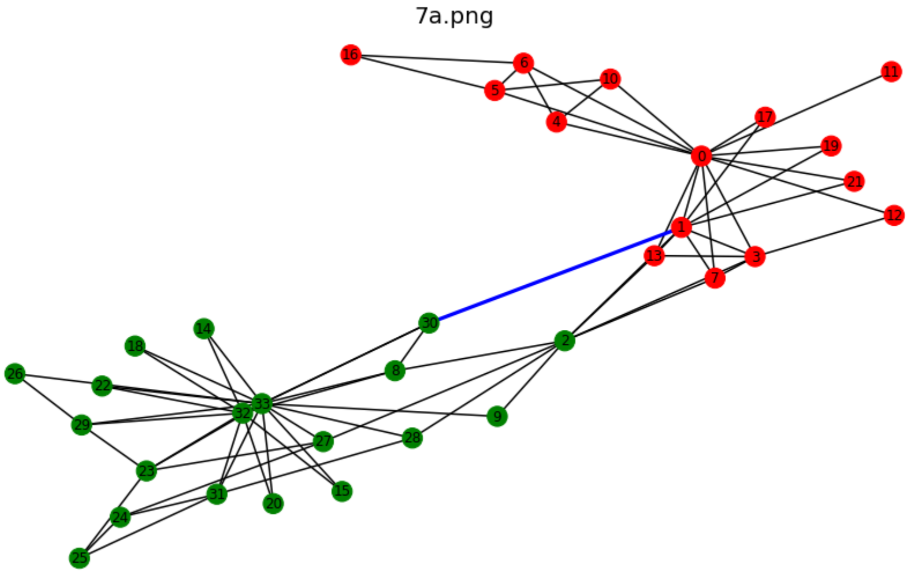

Figure 15: Iteration 7 Highlighted nodeedge to remove in blue

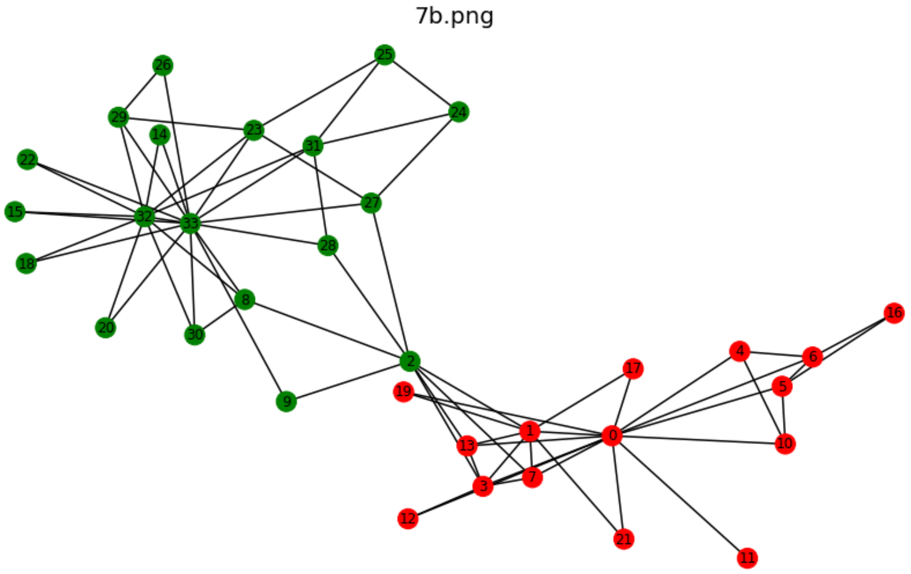

Figure 16: Iteration 7 confirm nodeedge has been removed

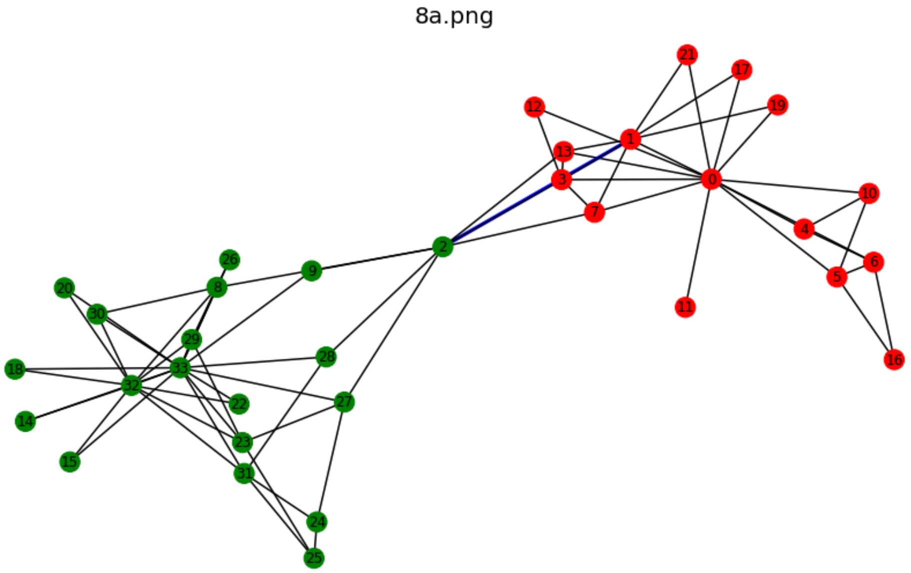

Figure 17: Iteration 8 Highlighted nodeedge to remove in blue

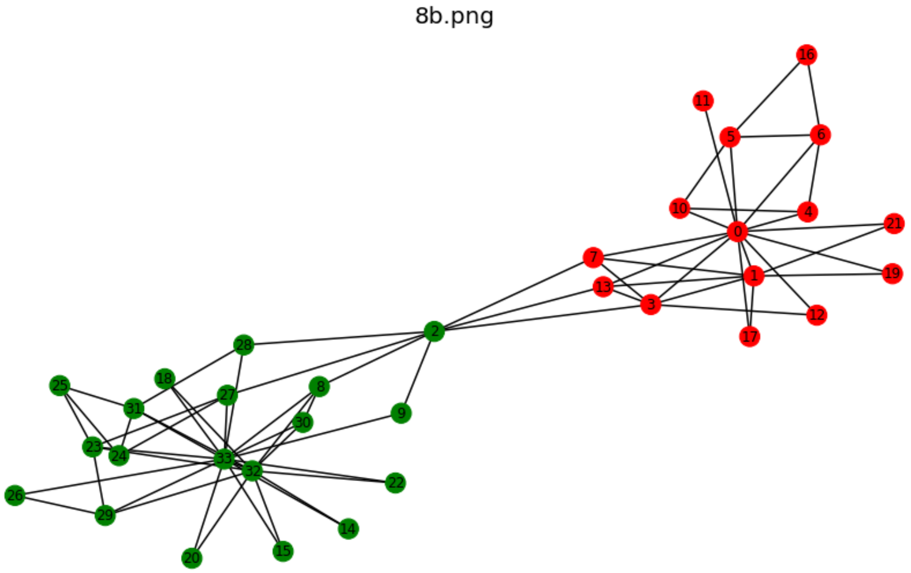

Figure 18: Iteration 8 confirm nodeedge has been removed

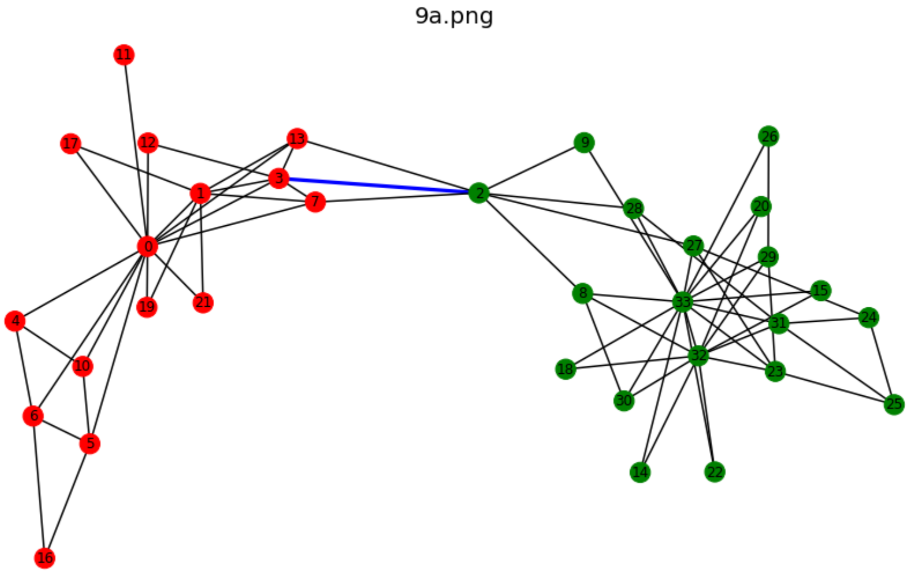

Figure 19: Iteration 9 Highlighted nodeedge to remove in blue

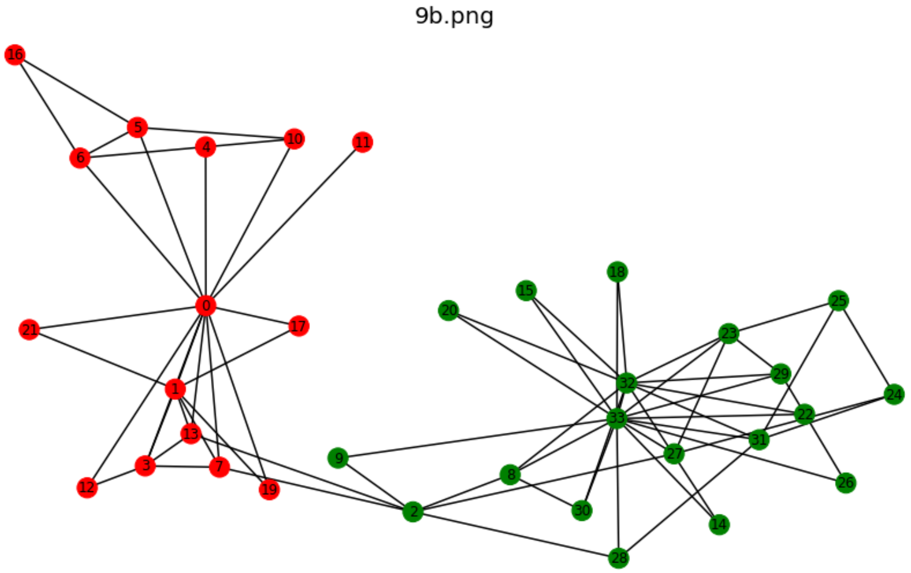

Figure 20: Iteration 9 confirm nodeedge has been removed

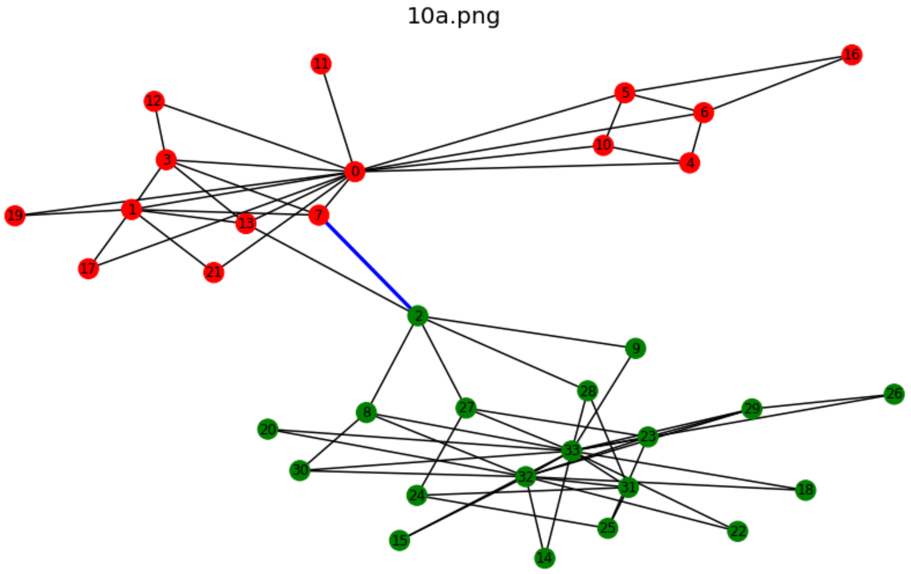

Figure 21: Iteration 10 Highlighted nodeedge to remove in blue

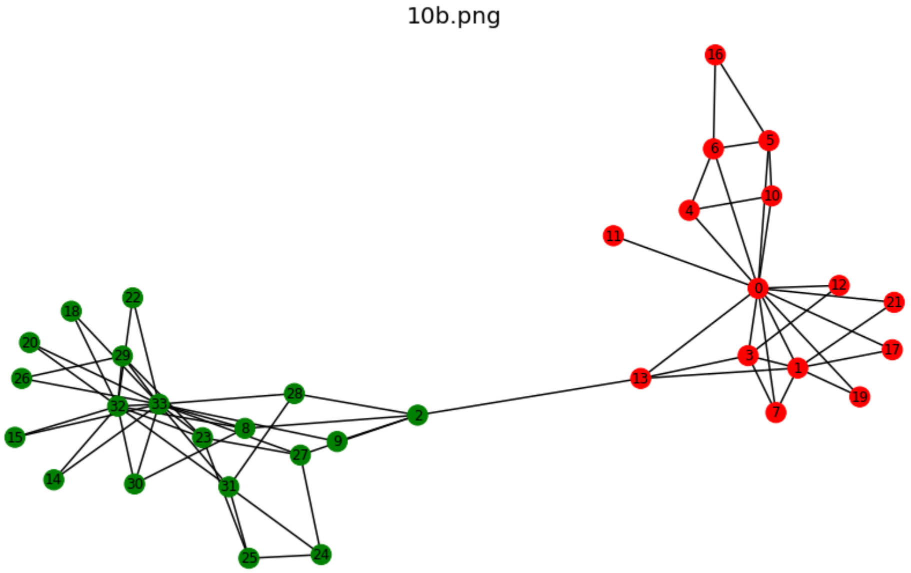

Figure 22: Iteration 10 confirm nodeedge has been removed

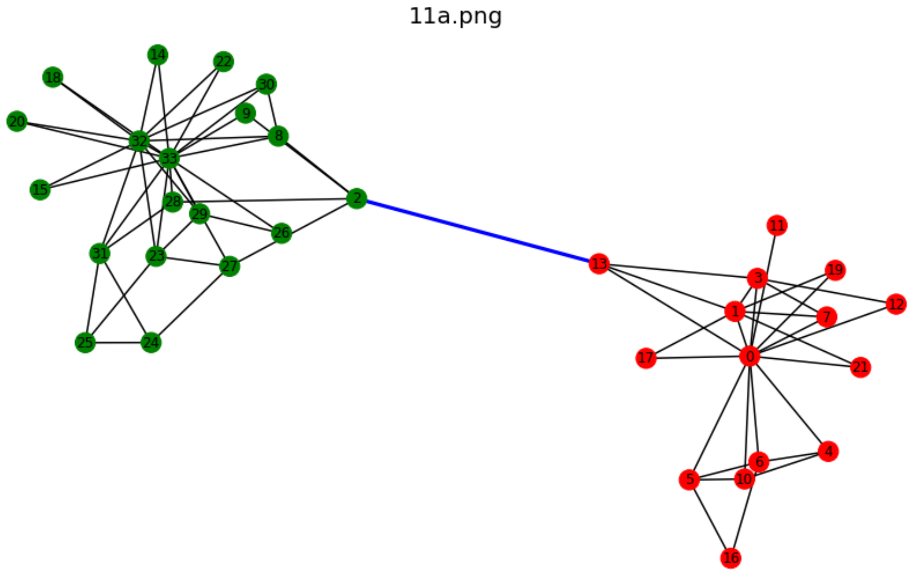

Figure 23: Iteration 11 Highlighted nodeedge to remove in blue

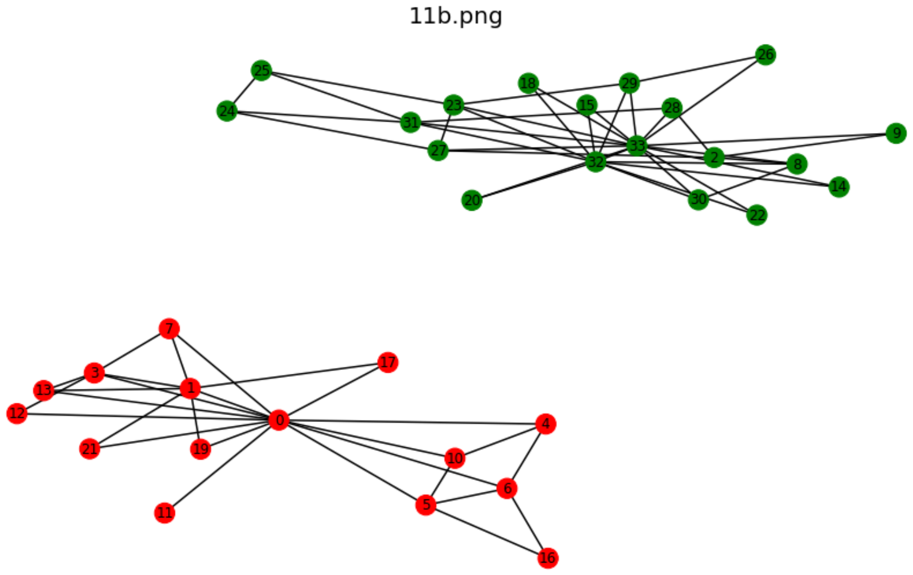

Figure 24: Iteration 11 confirm nodeedge has been removed

Discussion

It took 11 iteration to complete break the graph into two

Q3

Compare the connected components of the experimental graph (Step 2) with the connected components of the split Karate club graph (Step 1). Are they similar? Did all of the same colored nodes end up in the same group? If not, what is different?

Answer

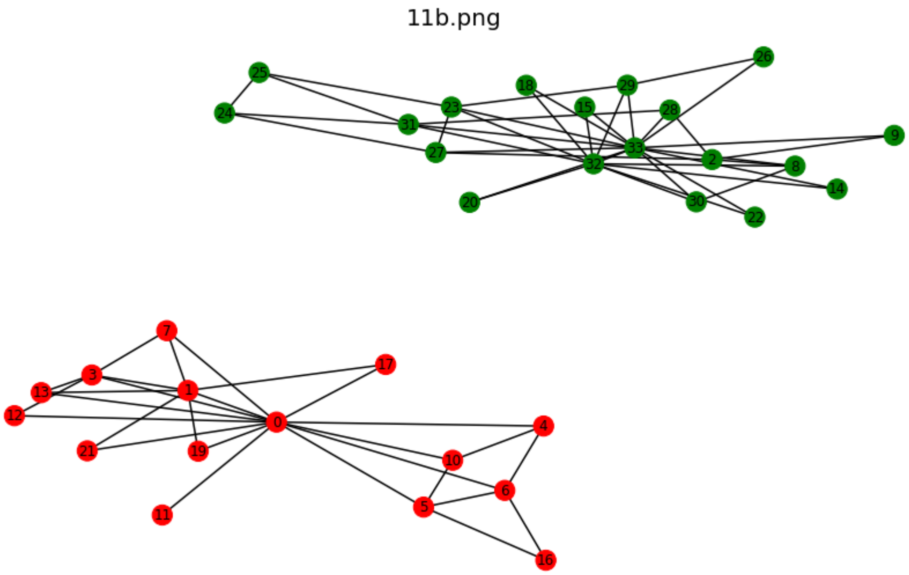

Figure 25: girvan-newman algorithm final split

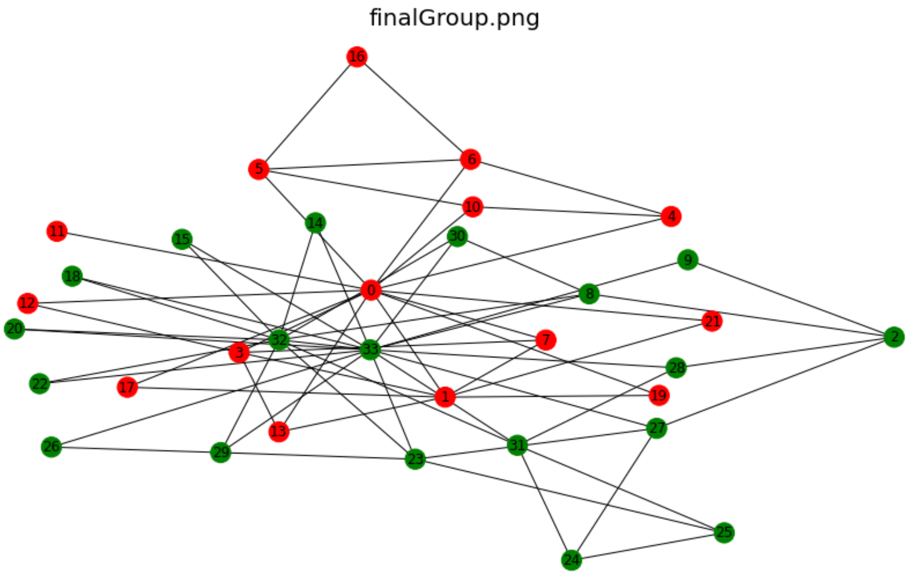

Figure 26: attached nodesedges with color splitting

Discussion

- Are they similar? Yes the look very similar to each other
- Did all of the same colored nodes end up in the same group? If not, what is different? Yes all the colored node ended up in the same group
- The diver of the function

```
    """
   =======================================================================

    Question 1b , Question 2 , Question 3 and Extra Credit Q1
    show the categories based on the distribution as a color coded
   =======================================================================

    passed the retrieved data to the girvan_newman algorithm
    color coded the result of the splitted group
    then plotted the graph
    """
    #returns the broken components as two NodeView list)
    final = getComponent(karate)
```

```
147     #print(final)
148     #Set the colors based on the list received
149     color_map = colorCode(karate,"final",final)
150     plot_theGraph(karate,color_map,"Q1/finalGroup.png","","","")
151     girvan_newman(t)
152 except Exception as e:
153     print(e)
```

Listing 7: driver for the algorithm (snapshot of graphing.py)

- getComponent function in line 146 was used to breakdown the nodes into various node colors of green and red

```
51 def getComponent(G):
52     if len(G.nodes()) == 1:
53         return [G.nodes()]
54     components = (G.subgraph(c) for c in nx.connected_components(G))
55     components = list(components)
56     count = 0
57     while len(components) == 1:
58         count +=1
59         G.remove_edge(*find_best_edge(G))
60 
61         components =(G.subgraph(c) for c in nx.connected_components(G))
62         components = list(components)
63     return components
```

Listing 8: Get the two NodeView object lists separating them into red and green categories (snapshot of graphing.py)

- Used the colorCode funciton to build the node list color based on the result of getComponent
- Launched the driver for girvan-new man algorithm on line 151

```
151     girvan_newman(t)
```

Listing 9: the driver for newman-algorithm (snapshot of graphing.py)

- In the girvan_newman function in line 96 to 119 in listing 6
- Found the maximum edge called it the bestEdge

```
40 def find_best_edge(G0):
41     """
42     Networkx implementation of edge_betweenness
43     returns a dictionary. Make this into a list,
44     sort it and return the edge with highest betweenness.
45     """
46     eb = nx.edge_betweenness_centrality(G0)
```

```python
    eb_li = list(eb.items())
    eb_li.sort(key=lambda x: x[1], reverse=True)
    return eb_li[0][0]
```

Listing 10: Best edge finder (snapshot of graphing.py)

- Built the path for before the removal of edges
- Built the edge color list and width of the edge list, to set the color coding and width variable

```python
def set_color_edges(G,tuplesEdgeToRemove,reset):
    """
    This builds the edge attributes color and weight

    """
    totalEdges = G.number_of_edges()
    color_edge_map = ['black'] * totalEdges
    weight_map = [1.5] * totalEdges
    if(reset =="n"):
        total = -1
        for n in G.edges:
            total += 1
            if tuplesEdgeToRemove == n:
                color_edge_map[total] = 'blue'
                weight_map[total] = 3.2
    return color_edge_map,weight_map
```

Listing 11: Setting and reset edges color and width values (snapshot of graphing.py)

- Made the first Plot using the plot function
- Reset the colors and width to original state
- Built the path for after the removal of edges

```python
        path = "Q2/" +str(count) +"a.png"
        #find the best Edge and return as a list
        bestEdge = find_best_edge(G)
        edge_color_mapped,weightMap = set_color_edges(G, bestEdge,"n")

        #print(edge_color_mapped)
        plot_theGraph(G,color_map,path,"spacing",edge_color_mapped,weightMap)
        #Remove the best edge
        G.remove_edge(*bestEdge)
        #ReSet everything back to black and reset the weight too
        edge_color_mapped,weightMap = set_color_edges(G, bestEdge,"")
        #Build string path after edge as been removed
        path = "Q2/" + str(count) +"b.png"

        plot_theGraph(G,color_map,path,"spacing",edge_color_mapped,
    weightMap)
```

Listing 12: logic in the algorithm (snapshot of graphing.py)

Extra Credit Q 1

We know the group split in two different groups. Suppose the disagreements in the group were more nuanced. What would the clubs look like if they split into 3, 4, and 5 groups? A single node can be considered as a ”group”.

Answer

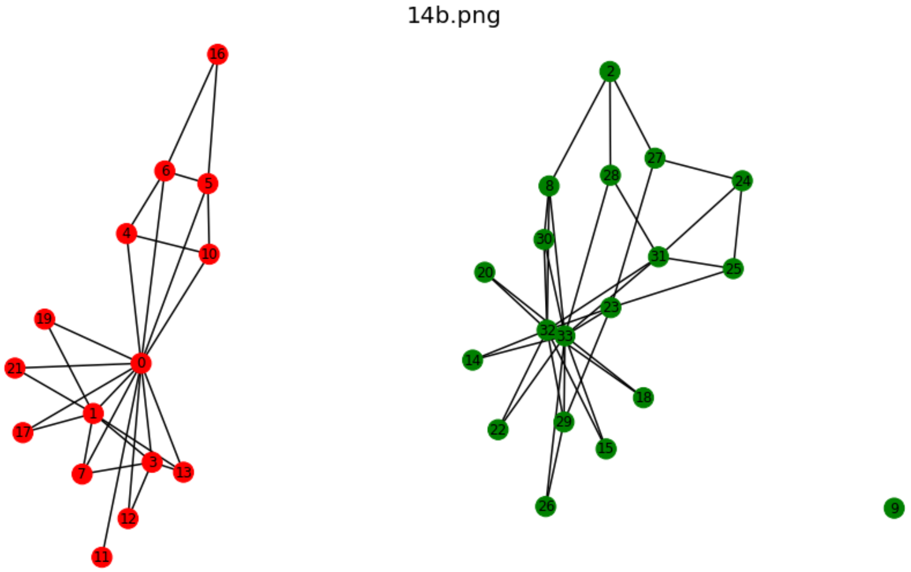

Figure 27: Group 3

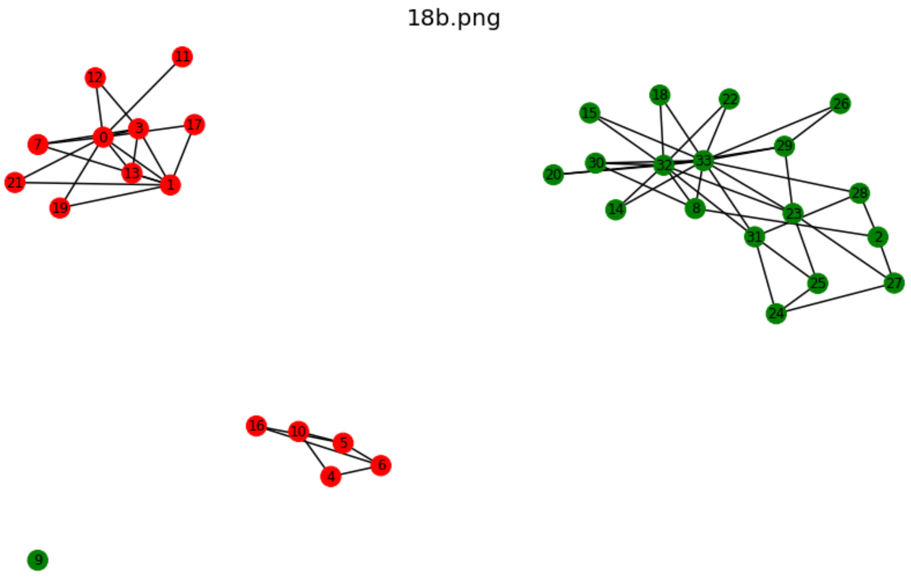

Figure 28: Group 4

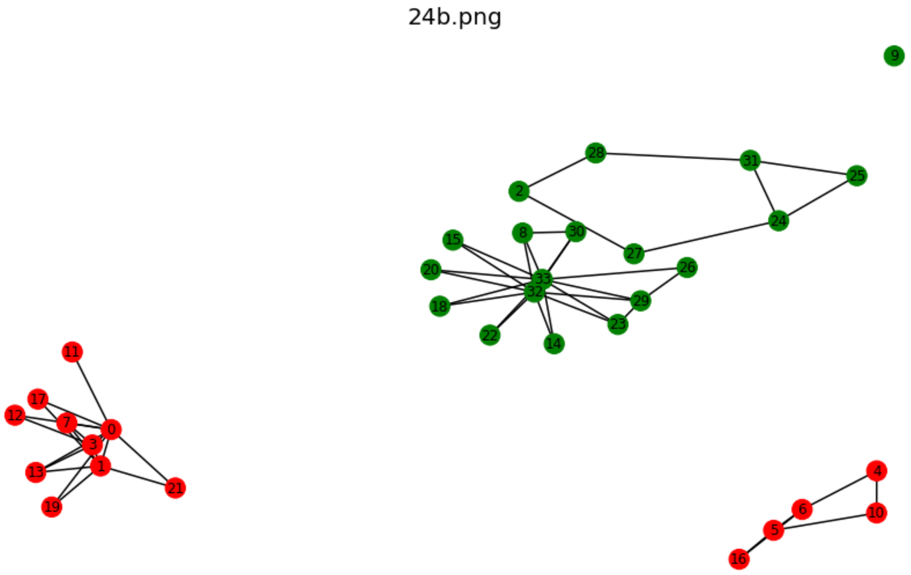

Figure 29: Group 5

Discussion

Same algorithm of previous question produced these outputs. All output will be included in github

References

- https://stackoverflow.com/questions/9012487/matplotlib-pyplot-savefig-outputs-blank-image
- https://gawron.sdsu.edu/python_for_ss/course_core/book_draft/Social_Networks/Networkx.html
- https://stackoverflow.com/questions/332289/how-do-you-change-the-size-of-figures-drawn-with-matplotlib
- https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.centrality.girvan_newman.html
- https://networkx.org/documentation/stable/tutorial.html
- https://networkx.org/documentation/stable/reference/drawing.html#module-networkx.drawing.nx_pylab