Lab 3 Network Centrality and Visualization with NetworkX

Due: Midnight, October 2nd

In lab 3, we will introduce network centrality and visualization with NetworkX. The goal of the lab includes

1. **Load graph data**
2. **Use the visualization tool with different parameters**
3. **Calculate various network centrality and visualize the network with node size proportional to the centrality score**
4. **Find and visualize the shortest path and minimum spanning tree**

Save Your Notebook!

- Click on File (upper left corner), Select “Save” or press Ctrl+S.
- Important: You may loose your modification to a notebook if you do not Save it explicitly.
- Advice: Save often.

Submission

- Please follow the instructions and finish the exercises.
- After you finish the lab, please Click on File, Select “Download .ipynb”
- After download is complete, Click on File, Select “Print”, and and Choose ``Save as PDF"
- Submit both the Notebook file and the PDF File as your submission for Lab 3.

1. Preparation

Before we start to visualize the networks, we have to install the packages and prepare the network dataset.

1.1 Connect this Colab notebook with your Google Drive

Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

```
# The following code will mount the drive
from google.colab import drive
drive.mount('/content/gdrive')
```

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

1.1. Install Packages

The plotting package matplotlib and complex graph package NetworkX should have been installed in Colab. If not, please run the following code cell to install them.

```
!pip install matplotlib
!pip install networkx
```

```
    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
    Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packa
    Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-pack
    Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dis
    Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/
    Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/
    Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-pac
    Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dis
    Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-package
    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
    Requirement already satisfied: networkx in /usr/local/lib/python3.7/dist-package
```

1.2. Load Graph Data

Please download the file "undirected_weighted.edgelist" from Canvas. The file is the edgelist of a weighted graph. Each line is in the format of

Source_Node, Target_Node, Edge_Weight

Please **upload** the file to the folder DS420 in Google Drive.

We can load the graph directly from an edgelist with function read_edgelist. This link gives details of the function parameters and some examples: https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.readwrite.edgelist.read_edgelist.html

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


# load the network
                                                                 S420/undirected_weight
```

Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

```
# we can access the node set and edge set with the following functions
print('node list: {}'.format(undirected_G.nodes()))
print('edge list: {}'.format(undirected_G.edges(data = True)))
```

```
    node list: ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13',
    edge list: [('1', '2', {'weight': 1.0}), ('1', '3', {'weight': 2.0}), ('1', '4',
```

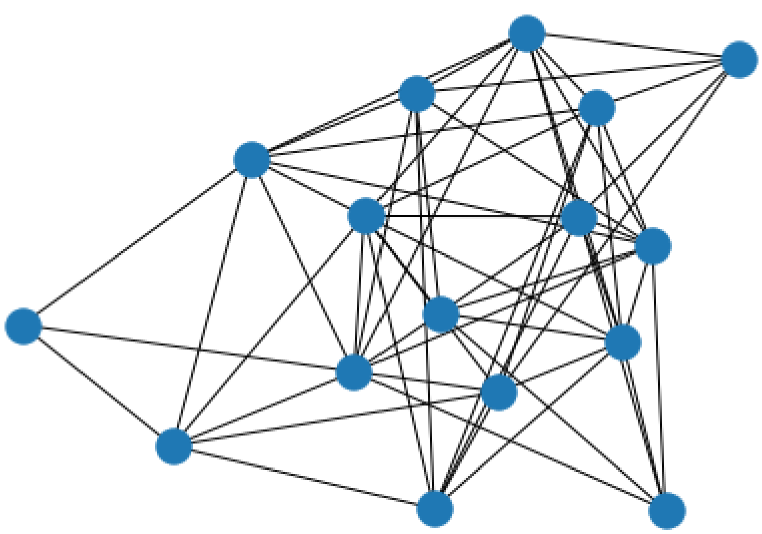
2. Visualization

In this section, we will learn how to use the NetworkX package for different graph layout and visualize the plot with matplotlib

2.1. Different Layouts

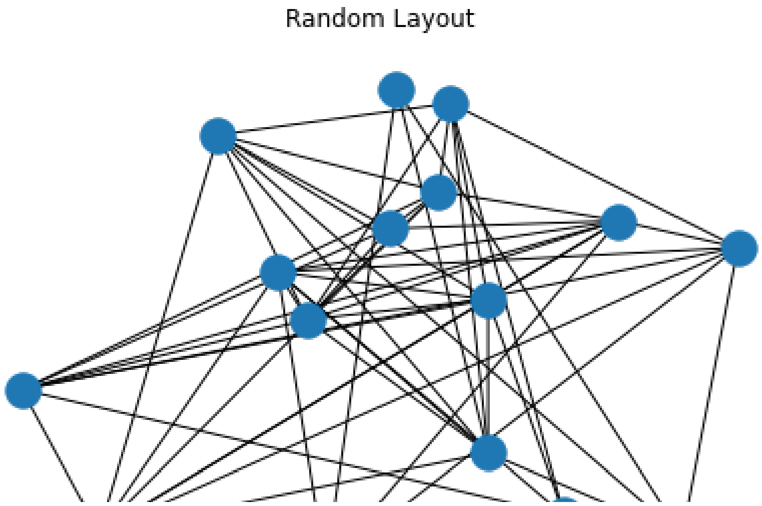
Firstly, we will try different layouts of network visualization. Currently, there are around 10 diferent layout options we have in NetworkX. The document for NetworkX layout function is available at https://networkx.github.io/documentation/stable/reference/drawing.html#module-networkx.drawing.layout. You can click the funciton name in the document, which will lead you to the detailed descrioption of the function. Let's show some of the layout

```
%matplotlib inline
nx.draw(undirected_G) # without layout, spring layout will be used, The algorithm simu
plt.title('Without Layout')
plt.show()
```

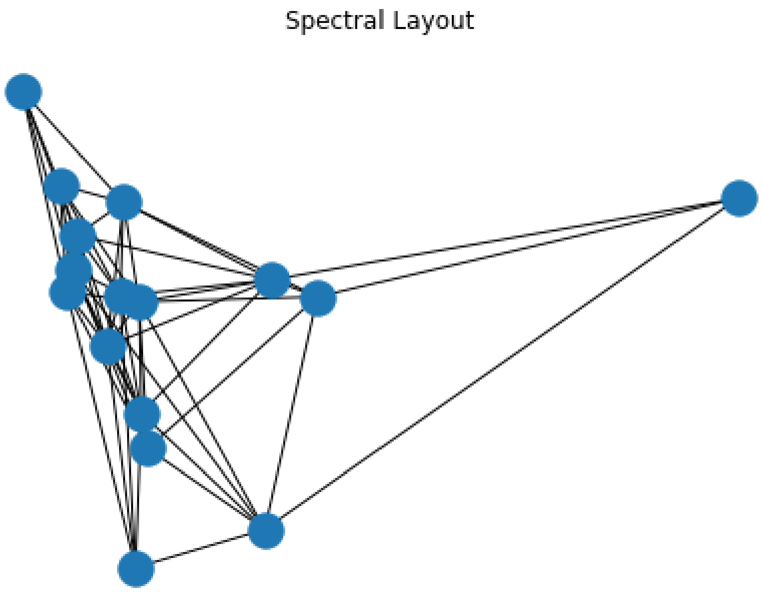


Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

```
plt.title('Random Layout')
plt.show()
```

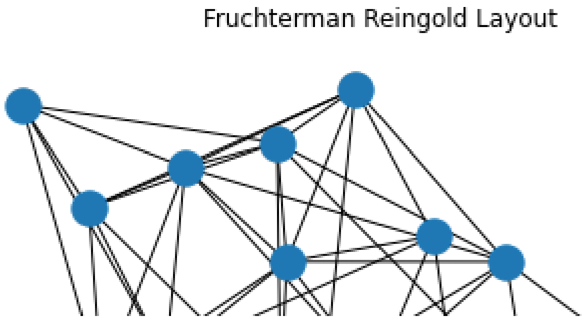



```
pos_spectral = nx.spectral_layout(undirected_G) #using spectral layout
nx.draw(undirected_G, pos_spectral)
plt.title('Spectral Layout')
plt.show()
```




```
pos_fruchterman = nx.fruchterman_reingold_layout(undirected_G) #using fruchterman rei
nx.draw(undirected_G, pos_fruchterman)
plt.title('Fruchterman Reingold Layout')
plt.show()
```

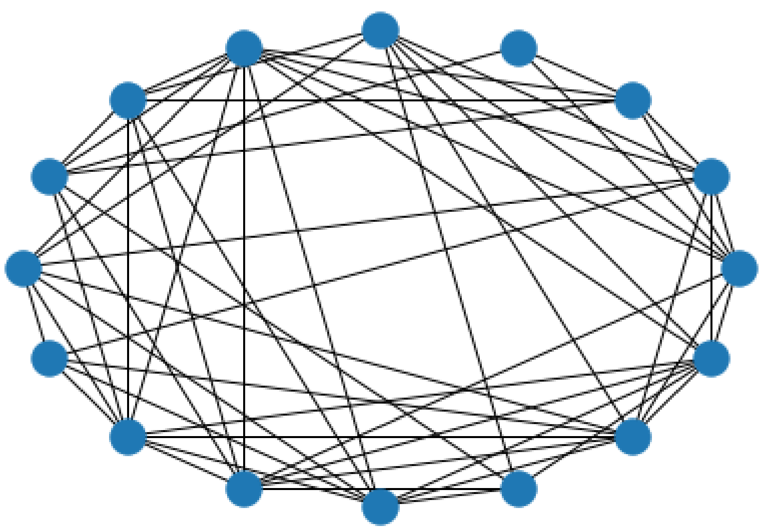
Unrecognized runtime "python36"; defaulting to "python3" Notebook settings



Exercise 1

visualize the graph with **circular layout**, you can call pos_circular=nx.circular_layout(undirected_G) to get the circular layout, then visualize the graph

```
# TODO
pos_circular= nx.circular_layout(undirected_G)
nx.draw(undirected_G, pos_circular)
```



2.2. Usage of the Visualization Tools

In the above cells, we used the default setting of network draw functions. We can chhange the

ntroduction on how to

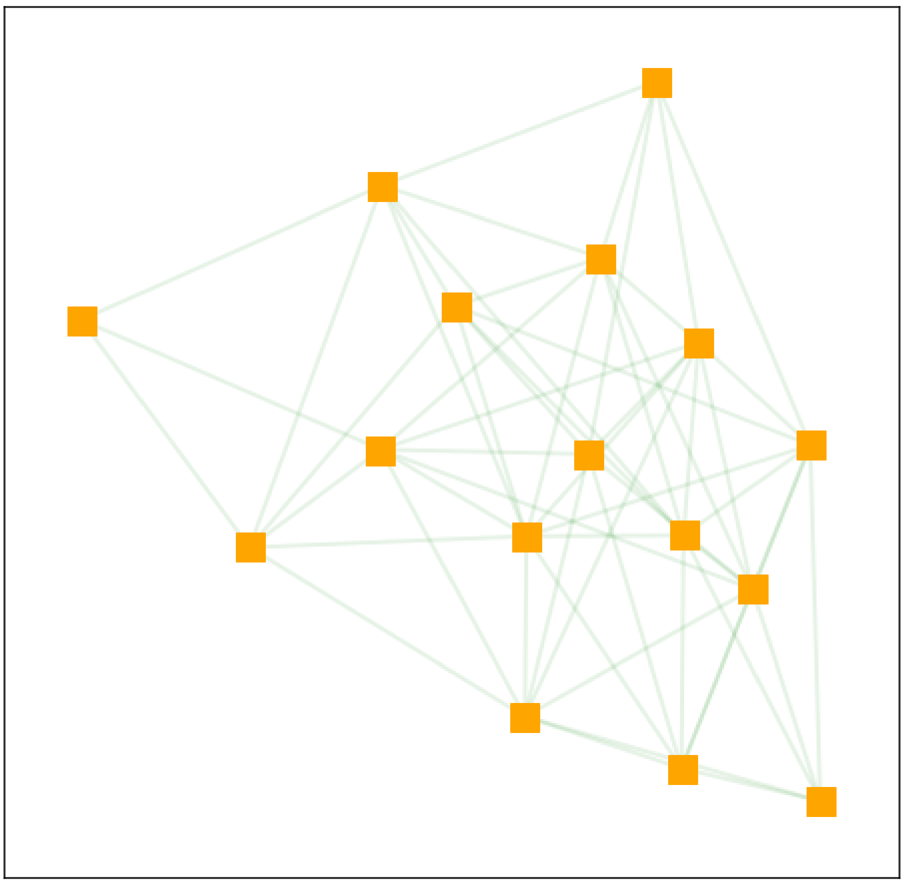
draw_networkx_nodes:
https://networkx.github.io/documentation/stable/reference/generated/networkx.drawing.nx_pylab.draw_networkx_nodes.html#networkx.drawing.nx_pylab.draw_networkx_nodes

draw_networkx_edges: https://networkx.github.io/documentation/stable/reference/generated/networkx.drawing.nx_pylab.draw_networkx_edges.html

draw_networkx_edge_labels: https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.drawing.nx_pylab.draw_networkx_edge_labels.html

```
# plot the nodes with node_size = 200, node_color as orange, node_shape as square
plt.figure(figsize=(8,8))
pos_fruchterman = nx.fruchterman_reingold_layout(undirected_G)
nx.draw_networkx_nodes(undirected_G, pos_fruchterman, node_size=200, node_color='orang
# plot the edges with width = 2 and edge_color='green'
nx.draw_networkx_edges(undirected_G, pos_fruchterman, width=2, edge_color='green', alp
```

```
<matplotlib.collections.LineCollection at 0x7fbb609f5310>
```



```
plt.figure(figsize=(8,8))
nx.draw_networkx_nodes(undirected_G, pos_fruchterman, node_size=200, node_color='orang
```

Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

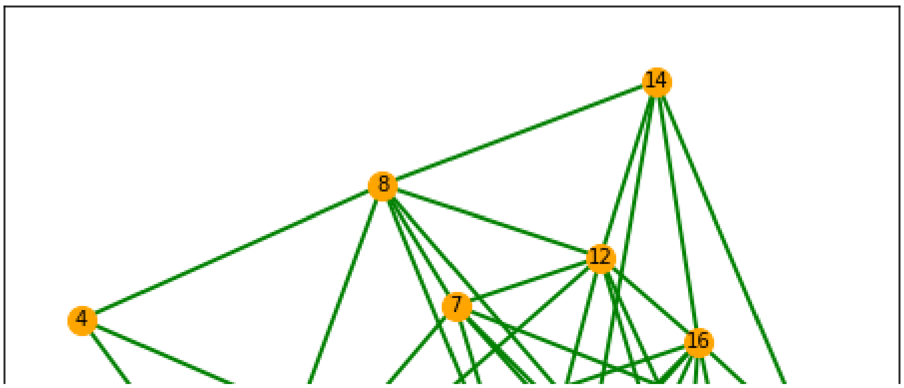
```
G.edges()]
nx.draw_networkx_edges(undirected_G, pos_fruchterman, width=weights,alpha=0.5, edge_co

# label the edge with weight
edge_labels = nx.get_edge_attributes(undirected_G,'weight')
nx.draw_networkx_edge_labels(undirected_G,pos_fruchterman,edge_labels=edge_labels, for
```

```
{('1', '2'): Text(-0.21709624586536802, -0.2228252268731071, '1.0'),
 ('1', '3'): Text(-0.5073479861398601, -0.07176377130571726, '2.0'),
 ('1', '4'): Text(-0.6844365276903346, 0.13203972813717305, '2.0'),
 ('1', '5'): Text(-0.14987704696876297, 0.011837278133387346, '2.0'),
 ('1', '6'): Text(-0.21550492712923405, -0.061692026680967564, '2.0'),
 ('1', '12'): Text(-0.13731232371903238, 0.18780822866869784, '2.0'),
 ('1', '15'): Text(0.022776977221416195, -0.10906454170894919, '1.0'),
 ('1', '16'): Text(-0.03378508054743609, 0.11198820171536995, '1.0'),
 ('2', '3'): Text(-0.35557117662455906, -0.3105058971296515, '2.0'),
 ('2', '5'): Text(0.001899762546538064, -0.22690484769054686, '2.0'),
 ('2', '6'): Text(-0.06372811761393302, -0.3004341525049018, '2.0'),
 ('2', '9'): Text(0.09993886033970342, -0.5078338106957052, '2.0'),
 ('2', '10'): Text(0.24602198499068828, -0.5373070878068842, '2.0'),
 ('2', '15'): Text(0.17455378673671723, -0.34780666753288336, '1.0'),
 ('2', '16'): Text(0.11799172896786495, -0.12675392410856423, '1.0'),
 ('3', '4'): Text(-0.8229114584495256, 0.044359057880628674, '1.0'),
 ('3', '6'): Text(-0.3539798578884251, -0.14937269693751193, '1.0'),
 ('3', '7'): Text(-0.42887742920354255, 0.05684138265709099, '1.0'),
 ('3', '8'): Text(-0.505941544669876 8, 0.1644041324621 1916, '1.0'),
 ('4', '8'): Text(-0.6830300862203513, 0.36820763190500944, '1.0'),
 ('5', '7'): Text(-0.07140649003244542, 0.14044243209619559, '1.0'),
 ('5', '9'): Text(0.16715805923630847, -0.273171305689210 8, '1.0'),
 ('5', '14'): Text(0.14071014623874434, 0.34119852465035727, '1.0'),
 ('5', '15'): Text(0.24177298563332228, -0.11314416252638895, '2.0'),
 ('5', '16'): Text(0.18521092786447, 0.10790858089793019, '2.0'),
 ('6', '7'): Text(-0.1370343701929165, 0.06691312728184068, '1.0'),
 ('6', '8'): Text(-0.2140984856592508, 0.17447587708686885, '1.0'),
 ('6', '9'): Text(0.10153017907583739, -0.3467006105035657, '2.0'),
 ('6', '11'): Text(0.10386358151545494, -0.13860117485402684, '1.0'),
 ('6', '12'): Text(0.01605580453240263, 0.11019930303690317, '1.0'),
 ('6', '13'): Text(0.23709219638975285, -0.05775460808503488, '2.0'),
 ('6', '16'): Text(0.11958304770399891, 0.034379276083575286, '2.0'),
 ('7', '8'): Text(-0.28899605697436825, 0.38068995668147176, '1.0'),
```

```
# add the node label to the figure
plt.figure(figsize=(8,8))
nx.draw_networkx_nodes(undirected_G, pos_fruchterman, node_size=200, node_color='orang
nx.draw_networkx_edges(undirected_G, pos_fruchterman, width=2, edge_color='green', alp
nx.draw_networkx_labels(undirected_G, pos_fruchterman, font_size=10)
```

Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

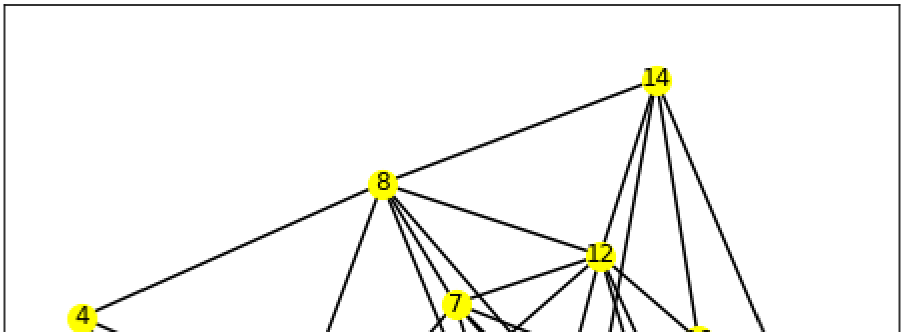
```
{'1': Text(-0.36887305538066906, 0.015916898950827107, '1'),
 '2': Text(-0.06531943635006697, -0.4615673526970413, '2'),
 '3': Text(-0.6458229168990511, -0.15944444156226162, '3'),
 '4': Text(-1.0, 0.24816255732351897, '4'),
 '5': Text(0.0691189614431431, 0.007757657315947584, '5'),
 '6': Text(-0.06213679887779906, -0.13930095231276224, '6'),
 '7': Text(-0.21193194150803396, 0.2731272068764436, '7'),
 '8': Text(-0.36606017244070255, 0.48825270648649993, '8'),
 '9': Text(0.26519715702947383, -0.5541002686943691, '9'),
 '10': Text(0.5573634063314435, -0.613046822916727, '10'),
 '11': Text(0.2698639619087089, -0.13790139739529142, '11'),
 '12': Text(0.09424840794260432, 0.3596995583865686, '12'),
 '13': Text(0.5363211916573047, 0.02379173614269248, '13'),
 '14': Text(0.2123013310343456, 0.674639391984767, '14'),
 '15': Text(0.41442700982350145, -0.23404598236872548, '15'),
 '16': Text(0.3013028942857969, 0.2080595044799128, '16')}
```




```
plt.figure(figsize=(8,8))
#change the nodes color to yellow
nx.draw_networkx_nodes(undirected_G,pos_fruchterman,node_size=200, node_color='yellow'
#increase the width of the edges
nx.draw_networkx_edges(undirected_G,pos_fruchterman,width=1.5,edge_color='black',alpha
#increase the font size
nx.draw_networkx_labels(undirected_G,pos_fruchterman,font_size=12)
#change the color of the selected nodes
nx.draw_networkx_nodes(undirected_G,pos_fruchterman,nodelist=['1','2','3'], node_colo
```

Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

```
<matplotlib.collections.PathCollection at 0x7fbb60662710>
```



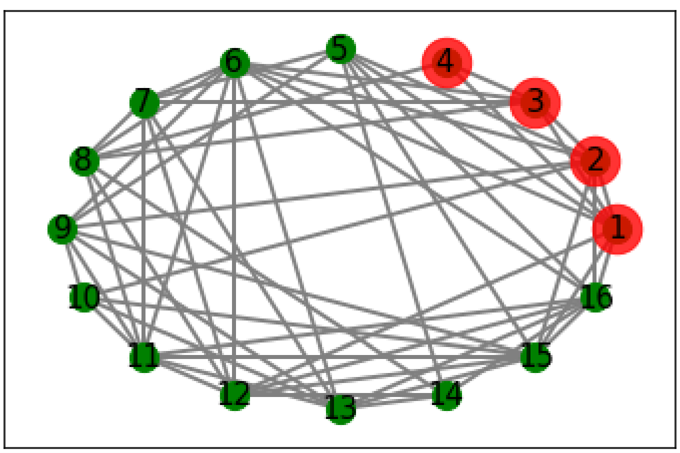
Exercise 2

Follow the above example, visualize the undirected graph with

- node_size as 200, node_color as green
- edge_color as gray, width as 2
- for node 1, 2, 3, 4, set their node color to red, nodies size to 600
- include the node label and set the font_size as 15
- use circular_layout

```
# TODO please fill in the missing part
nx.draw_networkx_nodes(undirected_G, pos_circular, node_size=200, node_color='green')
nx.draw_networkx_edges(undirected_G, pos_circular, width=2, edge_color= 'grey', alpha=
nx.draw_networkx_labels(undirected_G,pos_circular, font_size=15)
nx.draw_networkx_nodes(undirected_G, pos_circular, nodelist=['1','2','3','4'], node_co
```

```
<matplotlib.collections.PathCollection at 0x7fbb5fcad510>
```



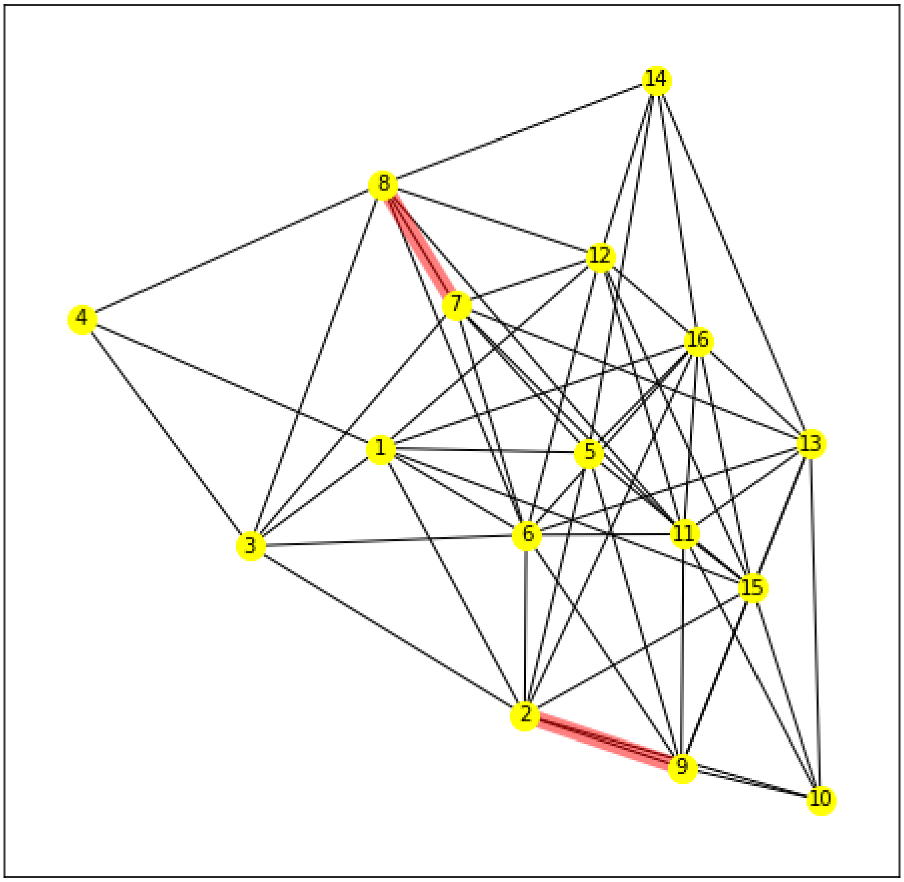
Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

...w, we change the edge ('7','8'),('2','9') with width=8,alpha=0.5,edge_color='r'

```
#change the property of the selected edges
plt.figure(figsize=(8,8))
nx.draw_networkx_nodes(undirected_G, pos_fruchterman, node_size=200, node_color='yello
```

```
nx.draw_networkx_edges(undirected_G, pos_fruchterman,width=1, edge_color='black', alph
nx.draw_networkx_labels(undirected_G, pos_fruchterman, font_size=10)
nx.draw_networkx_edges(undirected_G, pos_fruchterman,edgelist=[('7','8'),('2','9')], w
```

```
<matplotlib.collections.LineCollection at 0x7fbb5fc5f210>
```



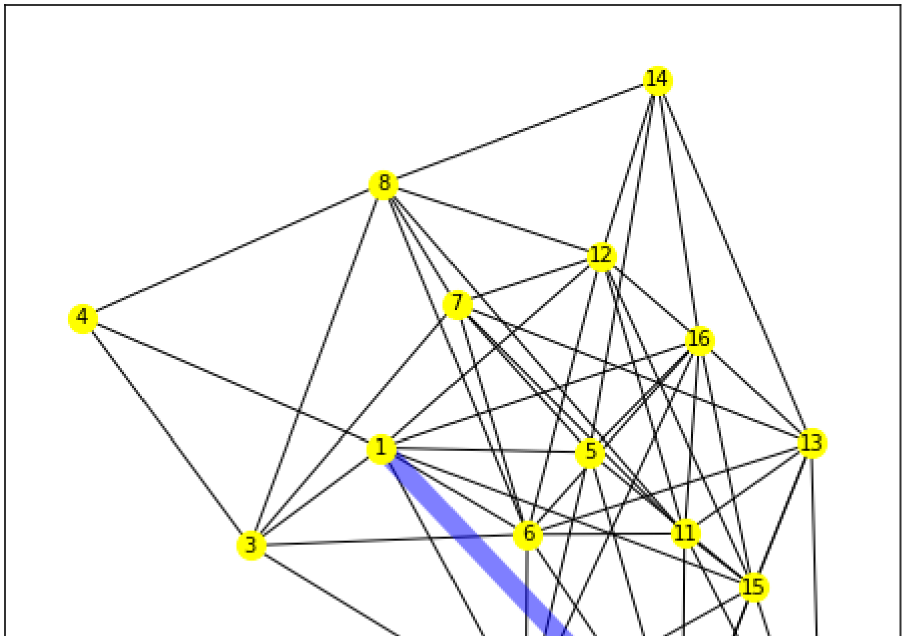
Exercise 3

Following the example in the above cell, please highlight the **edge ('1','9') with color as blue and width as 10**

```
# TODO: please finish exercise 3
plt.figure(figsize=(8,8))
nx.draw_networkx_nodes(undirected_G, pos_fruchterman, node_size=200, node_color='yello
nx.draw_networkx_edges(undirected_G, pos_fruchterman,width=1, edge_color='black', alph
nx.draw_networkx_labels(undirected_G, pos_fruchterman, font_size=10)
'1','9')], width=10,al
```

Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

<matplotlib.collections.LineCollection at 0x7fbb5fc5ab10>



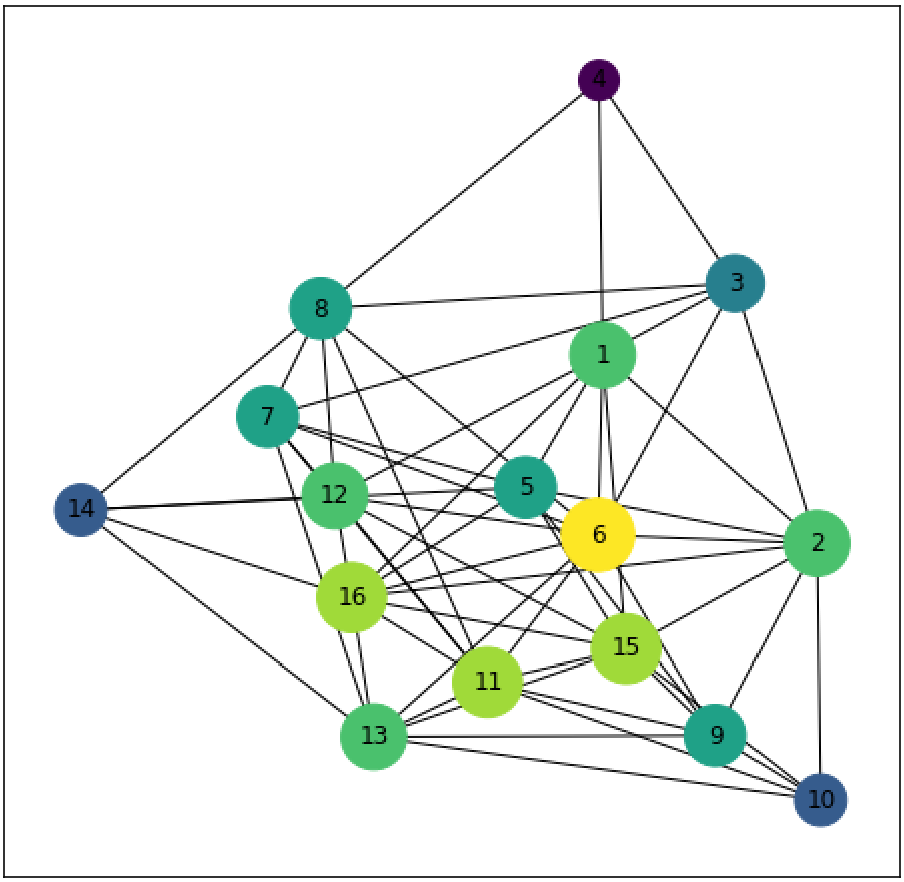
2.3. Find Inluential Nodes with Network Centrality and Visualize Them

```
plt.figure(figsize=(8,8))
pos = nx.fruchterman_reingold_layout(undirected_G)
nx.draw_networkx(undirected_G,pos,node_size=200,font_size=12)
```

Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

2.3.1. Visualize the network with degree centrality

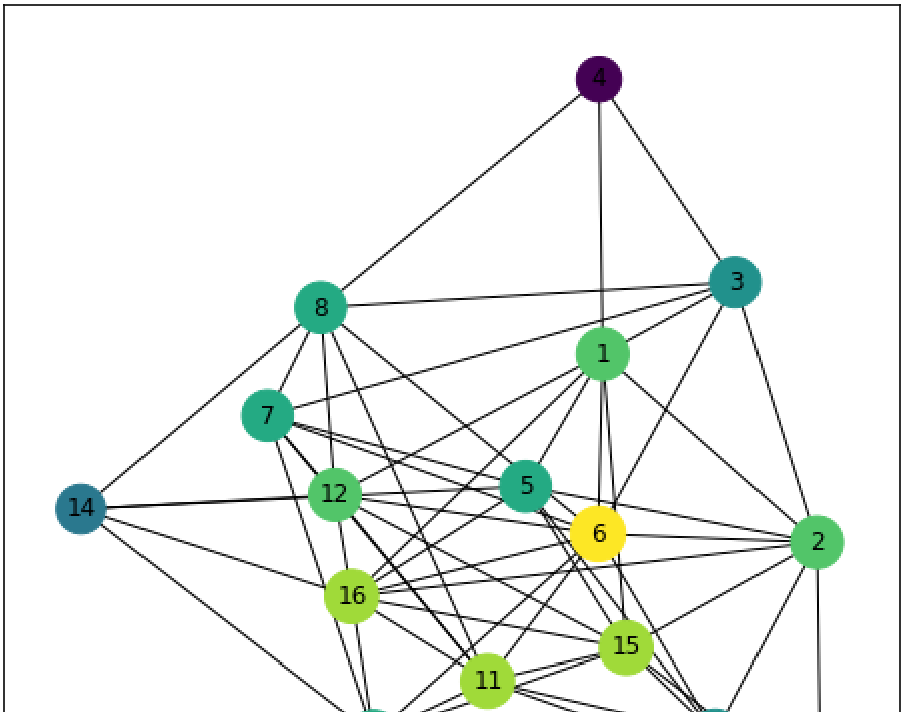
```
plt.figure(figsize=(8,8))
node_degree = nx.degree_centrality(undirected_G)  # get the degree of the nodes
nodesize=[node_degree[node]*2000 for node in undirected_G.nodes]  # multiply node degr
nx.draw_networkx(undirected_G, pos, node_size=nodesize, font_size=12, node_color=nodes
```



2.3.2. Visualize the network with closeness centrality

```
plt.figure(figsize=(8,8))
calcualte closeness ce
ed_G.nodes]
nodesize, font_size=12
```

Unrecognized runtime "python36"; defaulting to "python3" Notebook settings



2.3.3. Visualize the network with harmonic centrality

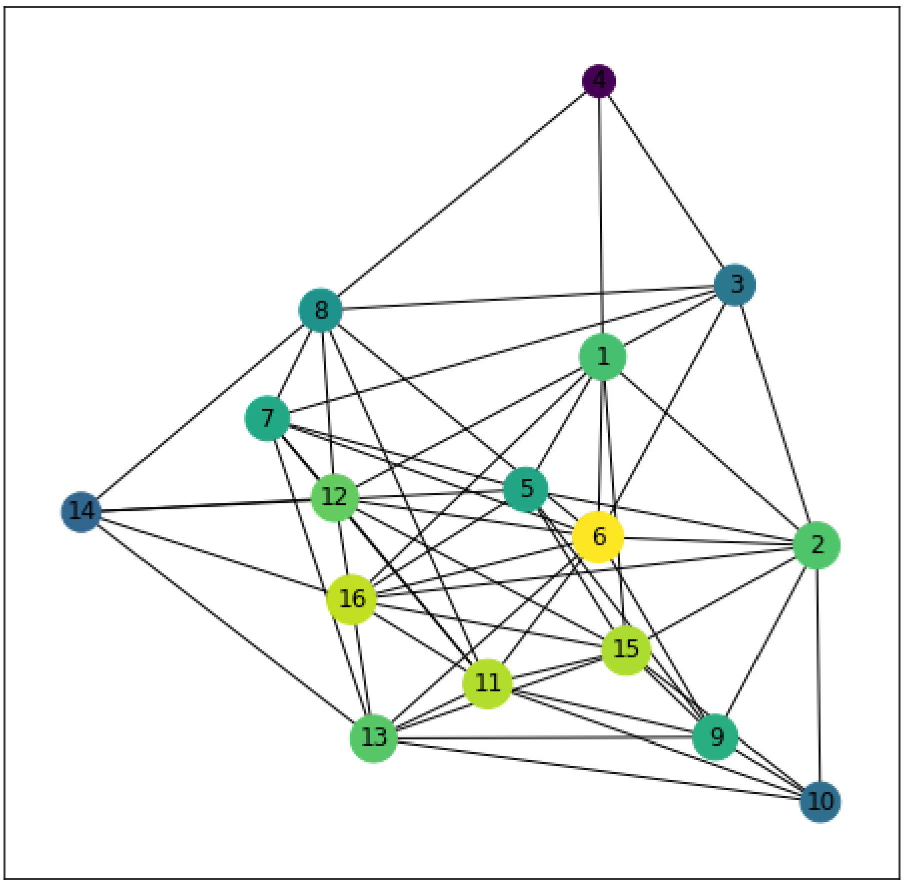
```
plt.figure(figsize=(8,8))
harmonic_centrality = nx.harmonic_centrality(undirected_G)  # calculate harmonic centı
nodesize = [harmonic_centrality[node]*60 for node in undirected_G.nodes]
nx.draw_networkx(undirected_G, pos, with_labels=True, node_size=nodesize, font_size=1:
```

Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

Exercise 4

Follow the above examples, please calcualte katz centrality and visualize the graph with node size reflecting katz centrality. You'll need to scale the katz centrality score for better visualization


```
# TODO: please finish exercise 4
plt.figure(figsize=(8,8))
katz_centrality = nx.katz_centrality(undirected_G)  # calculate katz centrality
nodesize = [katz_centrality[node]*2000 for node in undirected_G.nodes]
nx.draw_networkx(undirected_G, pos, with_labels=True, node_size=nodesize, font_size=1
```



3. Visualize the Shortest Path and Minimum Spanning Tree

Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

3.1. Shortest Path for Selected Nodes

In this example, we show how to find and visualize the shortest path for two given nodes

```
#select two nodes
source = '8'
sink = '16'
#Get the path_node_list from the dijkstra algorithm
length, path_node_list = nx.single_source_dijkstra(undirected_G, source, sink)
print('The shortest path length from node {} to node {} is {}. The path is {}'.format(
```

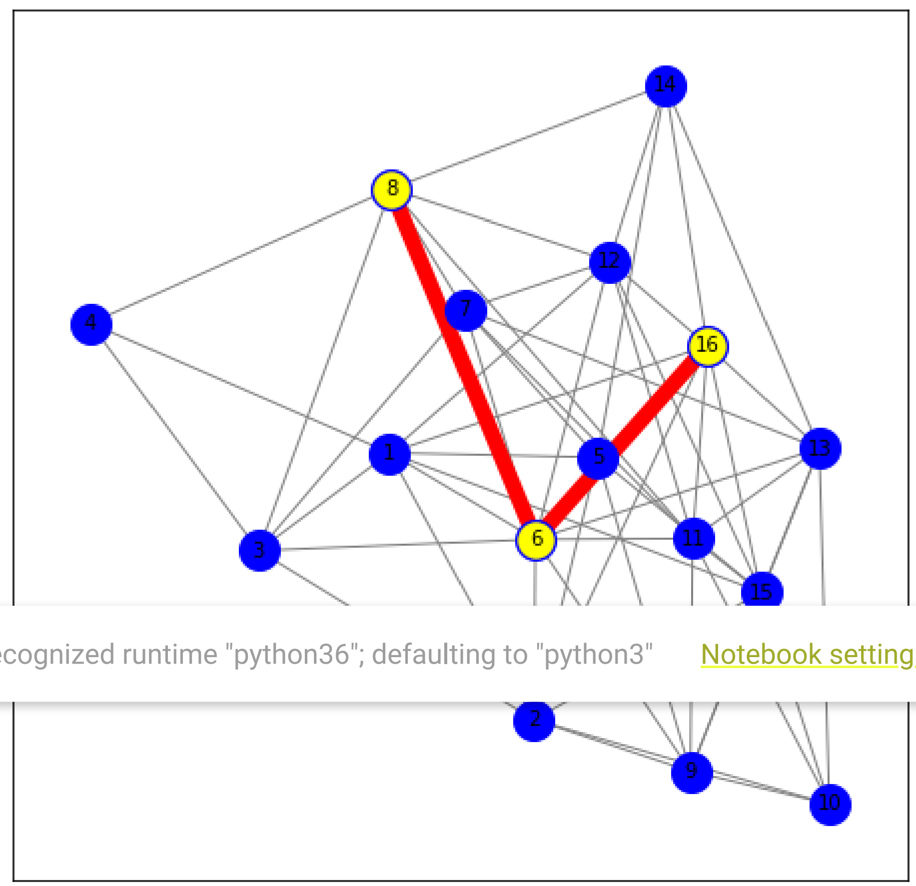
```
The shortest path length from node 8 to node 16 is 3.0. The path is ['8', '6', '
```

```
# convert the node list of the shortest path to edge list
path =[]
for i in range(len(path_node_list)-1):
    path.append((path_node_list[i], path_node_list[i+1]))
print(path)
```

```
[('8', '6'), ('6', '16')]
```

```
# now we can visulize the path with by highlighting edges and nodes in the path list
plt.figure(figsize=(8,8))
nx.draw_networkx_nodes(undirected_G, pos_fruchterman, node_size=400, node_color='blue'
nx.draw_networkx_edges(undirected_G, pos_fruchterman, width=1, edge_color='gray', alph
nx.draw_networkx_labels(undirected_G, pos_fruchterman, font_size=10)
nx.draw_networkx_edges(undirected_G, pos_fruchterman, edgelist=path, width=8, alpha=1,
nx.draw_networkx_nodes(undirected_G, pos_fruchterman, nodelist=path_node_list, node_co
```

```
<matplotlib.collections.PathCollection at 0x7fbb5f8f8410>
```



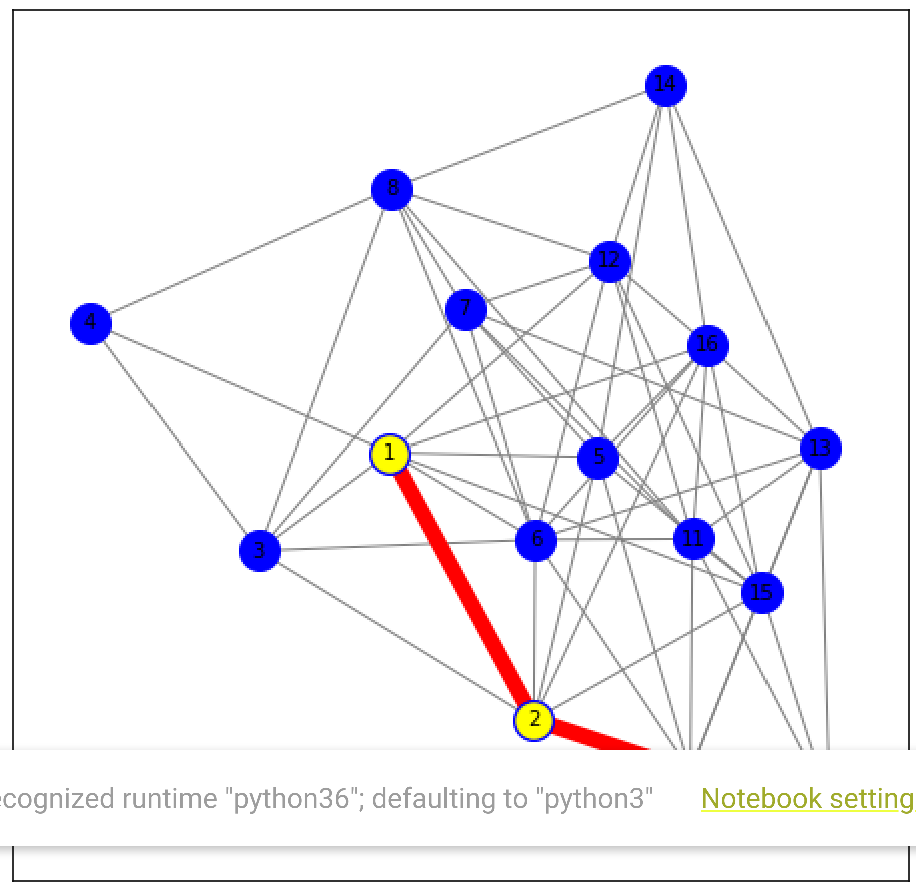
Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

Exercise 5

Follow the example in 3.1, find and visulaize the shortest path from node 1 to node 9

```
# TODO:
#select two nodes
source = '1'
sink = '9'
#Get the path_node_list from the dijkstra algorithm
length, path_node_list = nx.single_source_dijkstra(undirected_G, source, sink)
path =[]
for i in range(len(path_node_list)-1):
    path.append((path_node_list[i], path_node_list[i+1]))
plt.figure(figsize=(8,8))
nx.draw_networkx_nodes(undirected_G, pos_fruchterman, node_size=400, node_color='blue'
nx.draw_networkx_edges(undirected_G, pos_fruchterman, width=1, edge_color='gray', alpl
nx.draw_networkx_labels(undirected_G, pos_fruchterman, font_size=10)
nx.draw_networkx_edges(undirected_G, pos_fruchterman, edgelist=path, width=8, alpha=1,
nx.draw_networkx_nodes(undirected_G, pos_fruchterman, nodelist=path_node_list, node_cc
```

```
<matplotlib.collections.PathCollection at 0x7fbb5fa76050>
```



Unrecognized runtime "python36"; defaulting to "python3" Notebook settings

3.1. Minimum Spanning Tree

In this example, we show how to find and visualize the minimum spanning tree

Find and visualize the minimum spanning tree of undirected_G

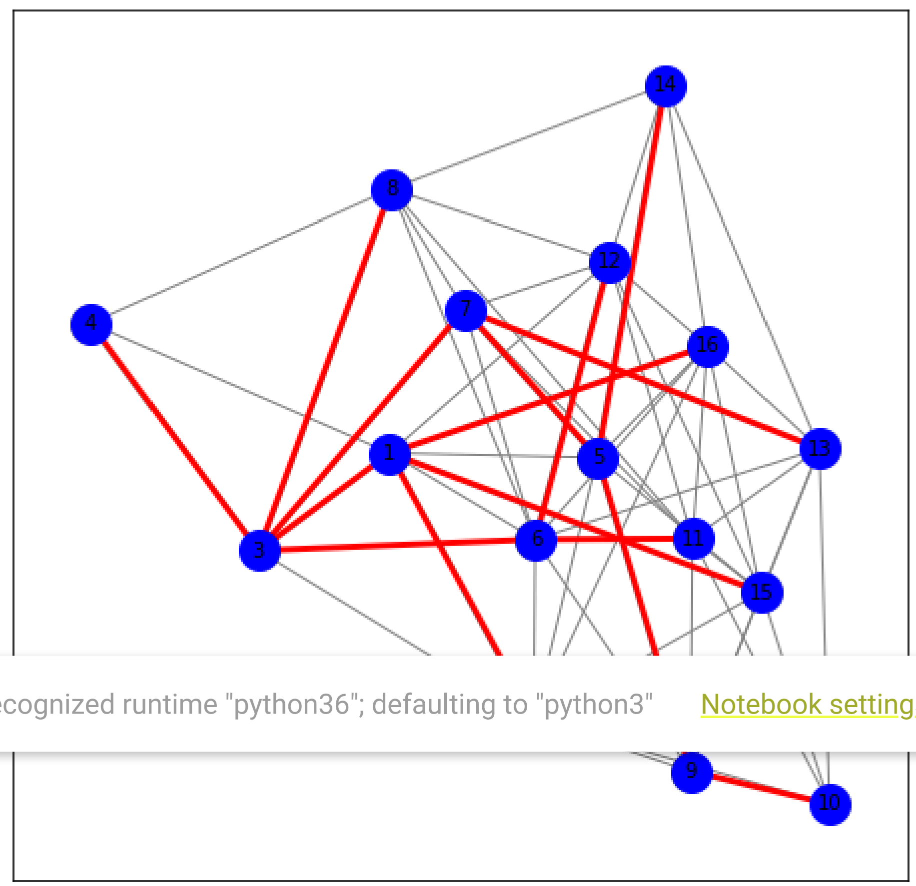
```
# get the minumum spanning tree
T=nx.minimum_spanning_tree(undirected_G)
```

```
# The edges of the trees can be retrieved as
print(T.edges)
```

```
[('1', '2'), ('1', '15'), ('1', '16'), ('1', '3'), ('3', '4'), ('3', '6'), ('3',
```

```
# visulize the MST
plt.figure(figsize=(8,8))
nx.draw_networkx_nodes(undirected_G, pos_fruchterman, node_size=400, node_color='blue'
nx.draw_networkx_edges(undirected_G, pos_fruchterman, width=1, edge_color='gray', alph
nx.draw_networkx_labels(undirected_G, pos_fruchterman, font_size=10)
nx.draw_networkx_edges(undirected_G, pos_fruchterman, edgelist=T.edges, width=3, alpha
```

```
<matplotlib.collections.LineCollection at 0x7fbb5f909950>
```



Calculate the weight of the minimum spanning tree

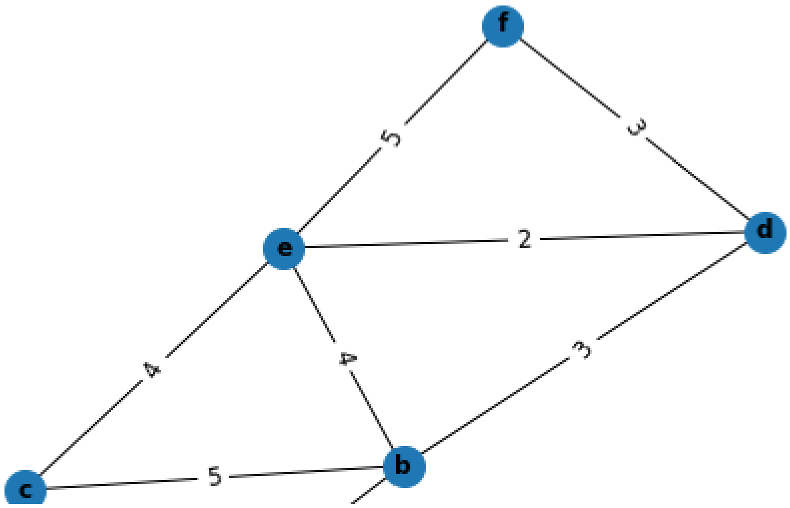
```
# Calculate the weight of the minimum spanning tree
MSTweight = 0
edgelist = list(T.edges)
for edge in edgelist:
    MSTweight += undirected_G.get_edge_data(edge[0], edge[1])['weight']
print('The weight of MST is: ' + str(MSTweight) )
```

```
The weight of MST is: 16.0
```

Exercise 6

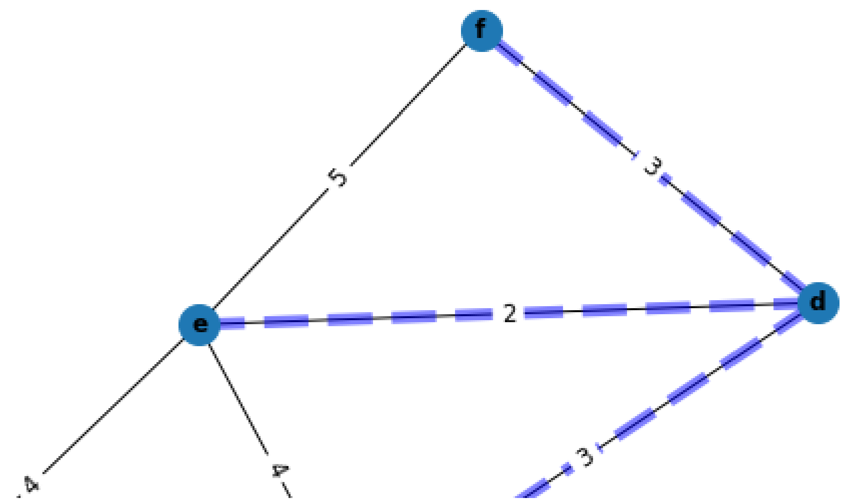
For the graph below, please find and visualize the minimum spanning tree, and report the weight of the minimum spanning tree

```
# load graph
G = nx.Graph()

G.add_edge('a', 'b', weight=2)
G.add_edge('a', 'c', weight=3)
G.add_edge('b', 'c', weight=5)
G.add_edge('b', 'd', weight=3)
G.add_edge('b', 'e', weight=4)
G.add_edge('c', 'e', weight=4)
G.add_edge('d', 'e', weight=2)
G.add_edge('d', 'f', weight=3)
G.add_edge('e', 'f', weight=5)

# visualize the graph
plt.figure(figsize=(8,8))
pos = nx.spring_layout(G)
edge_labels = nx.get_edge_attributes(G,'weight')
nx.draw_networkx_edge_labels(G,pos,edge_labels=edge_labels, font_size=12)
nx.draw(G, pos, with_labels=True, font_weight='bold', node_size=400, font_size=12)
plt.show(block=False)
```



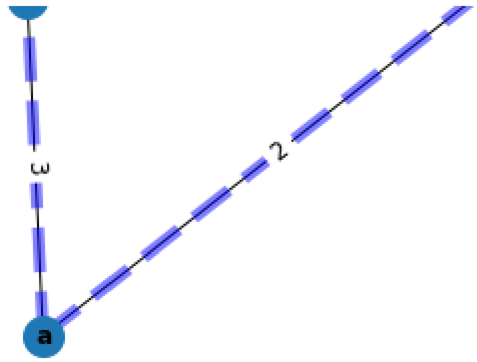

```
plt.figure(figsize=(8,8))

# find the minimum spanning tree of G
T = nx.minimum_spanning_tree(G)

nx.draw(G, pos, with_labels=True, font_weight='bold', node_size=400, font_size=12)
nx.draw_networkx_edge_labels(G,pos,edge_labels=edge_labels, font_size=12)
nx.draw_networkx_edges(G, pos=pos, edgelist=T.edges, width=6, alpha=0.5, edge_color='k
plt.show()

# please calculate the weight of the minimum spanning Tree
# TODO:
MSTweight = 0
edgelist = list(T.edges)
for edge in edgelist:
    MSTweight += G.get_edge_data(edge[0], edge[1])['weight']
print('The weight of MST is: ' + str(MSTweight) )
```

Unrecognized runtime "python36"; defaulting to "python3" Notebook settings



The weight of the minimum spanning tree is:

Answer: 13




```
The weight of MST is: 13
```